NGUYỄN ĐỨC NGHĨA - NGUYỄN TÔ THÀNH

GIÁO TRÌNH

TOÁN RỜI RẠC

NXB ĐẠI HỌC QUỐC GIA HÀ NỘI -2009

Lời nói đầu

Toán rời rạc là một lĩnh vực của toán học nghiên cứu các đối tượng rời rạc. Chúng ta sẽ sử dụng công cụ của toán rời rạc khi phải đếm các đối tượng, khi nghiên cứu quan hệ giữa các tập rời rạc, khi phân tích các quá trình hữu hạn. Một trong những nguyên nhân chủ yếu làm nâng tầm quan trọng của toán rời rạc là việc cất giữ và xử lý thông tin trên máy tính bản chất là các quá trình rời rạc. Cuốn sách này nhằm giới thiệu các kiến thức cơ bản trong ba lĩnh vực có nhiều ứng dụng của toán rời rạc là: lý thuyết tổ hợp, lý thuyết đồ thị và hàm đại số lôgic. Nội dung cuốn sách được trình bày thành ba phần.

Phần I trình bày các vấn đề của lý thuyết tổ hợp xoay quanh 4 bài toán cơ bản: Bài toán đếm, Bài toán tồn tại, Bài toán liệt kê và Bài toán tối ưu tổ hợp. Nội dung của phần 1 không những giúp nâng cao tư duy toán, mà còn làm quen với tư duy thuật toán trong việc giải quyết các vấn đề thực tế, đồng thời cũng rèn luyện kỹ thuật lập trình giải các bài toán tổ hợp.

Phần II đề cập đến lý thuyết đồ thị - một cấu trúc rời rạc tìm được những ứng dụng rộng rãi trong nhiều lĩnh vực của khoa học kỹ thuật và đời sống. Trong phần này sau phần giới thiệu các khái niệm cơ bản, các bài toán ứng dụng quan trọng của lý thuyết đồ thị như Bài toán cây khung nhỏ nhất, Bài toán đường đi ngắn nhất, Bài toán luồng cực đại trong mạng... và những thuật toán để giải quyết chúng đã được trình bày chi tiết cùng với việc phân tích và hướng dẫn cài đặt chương trình trên máy tính.

Phần III liên quan đến lý thuyết hàm đại số lôgic là cơ sở để nắm bắt những vấn đề phức tạp của kỹ thuật máy tính. Sau phần trình bày các khái niệm cơ bản, phần này đi sâu vào vấn đề tối thiểu hoá các hàm đại số lôgic và mô tả một số thuật toán quan trọng để giải quyết vấn đề đặt ra như thuật toán Quine - McCluskey, Black - Poreski.

Các vấn đề được trình bày trong cuốn sách đều được minh hoạ trên nhiều thí dụ, các thuật toán được mô tả trên ngôn ngữ PASCAL mô phỏng thuận tiện cho việc cài đặt các chương trình thực hiện thuật toán trên máy tính, trong đó nhiều thuật toán chọn lọc đã được cài đặt trên ngôn ngữ PASCAL.

Mục lục

PHẦN I

LÝ THUYẾT TỔ HỢP

1

MỞ ĐẦU

1.1. Sơ lược về tổ hợp

Tổ hợp như là một lĩnh vực của toán học rời rạc, xuất hiện vào đầu thế kỷ 17. Trong một thời gian dài, dường như tổ hợp nằm ngoài guồng máy phát triển của toán học cũng như ứng dụng của nó. Tình thế bắt đầu đổi khác khi xuất hiện các máy tính và cùng với nó là sự phát triển của toán hữu hạn. Hiện nay lý thuyết tổ hợp được áp dụng trong nhiều lĩnh vực khác nhau: lý thuyết số, hình học hữu hạn, biểu diễn nhóm, đại số không giao hoán, quá trình ngẫu nhiên, thống kê xác suất, quy hoạch thực nghiệm, ...

1.1.1. Các bài toán tổng quát

Tổ hợp dụng chạm đến nhiều vấn đề khác nhau của toán học, do đó khó có thể định nghĩa nó một cách hình thức. Nói chung, lý thuyết tổ hợp gắn liền với việc nghiên cứu phân bố các phần tử vào các tập hợp. Thông thường, các phần tử này là hữu hạn và việc phân bố chúng phải thoả mãn những điều kiện nhất định nào đấy, tuỳ theo yêu cầu của bài toán cần nghiên cứu. Mỗi cách phân bố như thế được gọi là một *cấu hình tổ hợp*.

Trong các tài liệu về tổ hợp, thường gặp các dạng bài toán dưới đây:

a) Bài toán đếm: đây là các bài toán nhằm trả lời câu hỏi "có bao nhiêu cấu hình thoả mãn điều kiện đã nêu ?". Phương pháp đếm thường dựa vào một số nguyên lý cơ bản và một số kết quả đếm các cấu hình đơn giản. Bài toán đếm được áp dụng một

cách có hiệu quả vào những công việc mang tính chất đánh giá như tính xác suất của một sự kiện, tính độ phức tạp của một thuật toán, ...

b) Bài toán liệt kê: bài toán này quan tâm đến tất cả cấu hình có thể có được, vì thế lời giải của nó cần được biểu diễn dưới dạng thuật toán "vét cạn" tất cả các cấu hình. Lời giải trong từng trường hợp cụ thể sẽ được máy tính điện tử giải quyết theo thuật toán đã nêu. Bài toán liệt kê được làm "nền" cho nhiều bài toán khác. Hiện nay, một số bài toán đếm, tối ưu, tồn tại vẫn chưa có cách nào giải, ngoài cách giải liệt kê. Nếu trước đây, cách giải liệt kê còn mang nặng tính lý thuyết, thì bây giờ nó ngày càng khả thi nhờ sự phát triển nhanh chóng của máy tính điện tử.

c) *Bài toán tối ưu:* khác với bài bài toán liệt kê, bài toán tối ưu chỉ quan tâm đến một cấu hình "tốt nhất" theo một nghĩa nào đấy. Đây là bài toán có nhiều ứng dụng trong thực tiễn và lý thuyết tổ hợp đã đóng góp một phần đáng kể trong việc xây dựng được những thuật toán hữu hiệu.

d) Bài toán tồn tại: nếu như trong các bài toán trên, việc tồn tại các cấu hình là hiển nhiên thì trong bài toán này, vấn đề "có hay không có" cấu hình còn là điều nghi vấn. Các bài toán loại này thường bị kẹt trong tình huống nan giải: không chỉ ra được cấu hình nào nhưng cũng không khẳng định được là chúng không tồn tại. Lịch sử toán học thường để lại những bài toán khó trong lĩnh vực này và việc cố gắng giải quyết chúng đã thúc đẩy không ít sự phát triển của nhiều ngành toán học.

1.1.2. Vài nét về lịch sử

Có thể nói tư duy về tổ hợp ra đời từ rất sớm. Vào thời nhà Chu, người ta đã biết đến các hình vẽ có liên quan đến những hình vuông thần bí. Thời cổ Hy lạp, nhà triết học Kxenokrat, sống ở thế kỷ thứ 4 trước công nguyên, đã biết cách tính số các từ khác nhau, lập từ một bảng chữ cái cho trước. Nhà toán học Pitagor và các học trò của ông đã tìm ra được nhiều con số có các tính chất đặc biệt, chẳng hạn số 36 không những là tổng của 4 số chẵn và 4 số lẻ đầu tiên mà còn là tổng lập phương của 3 số tự nhiên đầu tiên. Một định lý nổi tiếng của trường phái này là định lý về độ dài các cạnh của một tam giác vuông, và từ đó họ đã tìm ra các số mà bình phương của một số này bằng tổng bình phương của hai số khác. Việc tìm ra được các số như vậy, đòi hỏi phải có một nghệ thuật tổ hợp nhất định. Tuy nhiên, có thể nói rằng, lý thuyết tổ hợp được hình thành như một ngành toán học mới, vào quãng thế kỷ 17 bằng một loạt các công trình nghiên cứu nghiêm túc của các nhà toán học xuất sắc như Pascal, Fermat, Leibnitz, Euler, ... Mặc dù vậy, trong suốt hai thế kỷ rưỡi, vai trò quan trọng trong việc nghiên cứu thế giới tự nhiên vẫn thuộc về các ngành toán học cổ điển như toán giải tích, các phép tính vi tích phân, phương trình vi phân, phương trình toán lý...

Trong thời gian hiện nay, mối tương quan giữa toán học hữu hạn và toán học cổ điển đã có nhiều thay đổi, đặc biệt từ khi máy tính điện tử ra đời và phát triển. Nhiều bài toán nổi tiếng đã được giải trên máy tính bằng những thuật toán của toán hữu hạn. Các lĩnh vực trừu tượng của toán học như đại số lôgic, ngôn ngữ hình thức, ... đã trở thành khoa học ứng dụng để xây dựng các ngôn ngữ lập trình cho máy tính. Trong thời đại phát triển của toán học hữu hạn, vai trò của lý thuyết tổ hợp cũng khác xưa. Từ lĩnh vực nghiên cứu các trò chơi tiêu khiển, hay phân tích giải mã các bức thư cổ, tổ hợp đã chuyển sang lĩnh vực toán ứng dụng với sự phát triển mạnh mẽ.

1.2. Nhắc lại lý thuyết tập hợp

1.2.1. Các khái niệm và ký hiệu

Trong giáo trình này, tập hợp được ký hiệu bằng những chữ cái lớn $A, B,..., X, Y, ...$ còn những phần tử được ký hiệu bằng các chữ cái nhỏ $a, b, ..., x, y,...$ Để chỉ x là phần tử của X, ta viết $x \in X$, trái lại ta viết $x \notin X$. Nếu mỗi phần tử của A cũng là những phần tử của B thì ta nói A là tập con của B và viết $A \subseteq B$. Nếu $A \subseteq B$ và $B \subseteq A$ thì A và B là hai tập hợp bằng nhau và viết $A = B$.

Số các phần tử của tập hợp A sẽ được ký hiệu là $N(A)$ hoặc $|A|$. Một tập gồm n phần tử được gọi là một n-tập. Các tập hợp có thể xem như là những tập con của một tập hợp vũ trụ X. Tập rỗng là tập hợp không có phần tử nào, nó được xem như tập con của mọi tập hợp.

1.2.2. Các phép toán tập hợp

Các phép toán cho trên tập hợp là:

- Phần bù của A trong X, ký hiệu $\overline{A}$, là tập các phần tử của X không thuộc vào A:
$$\overline{A} = \{ x \in X: x \notin A \}.$$
- Hợp của A và B, ký hiệu $A \cup B$, là tập các phần tử hoặc thuộc vào A hoặc thuộc vào B hoặc thuộc vào cả hai tập A và B:
$$A \cup B = \{ x: x \in A \text{ hoặc } x \in B \}.$$
- Giao của A và B, ký hiệu $A \cap B$, là tập các phần tử đồng thời thuộc vào cả hai tập A và B:
$$A \cap B = \{ x: x \in A \text{ và } x \in B \}.$$
- Hiệu của tập A và B, ký hiệu là $A \setminus B$ (hoặc $A - B$):
$$A \setminus B = \{ x: x \in A \text{ và } x \notin B \}.$$

Các tập hợp, cùng với các phép toán trên nó, lập nên một đại số, gọi là *đại số tập hợp*. Dưới đây là một vài tính chất của các phép toán tập hợp:

- kết hợp

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$(A \cap B) \cap C = A \cap (B \cap C)$$

- giao hoán

$$A \cup B = B \cup A$$
$$A \cap B = B \cap A$$

- phân bố

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

- (đối ngẫu)

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

1.2.3. Các tính chất cho trên tập hợp

Mỗi tập con của một tập hợp được tương ứng với tính chất (còn được gọi là *mệnh đề*) xác định nó trên tập hợp đã cho. Với tương ứng này, các phép toán tập hợp được chuyển sang các phép toán mệnh đề:

- phủ định A, ký hiệu $\overline{A}$ (hay $NOT\ A$) tương ứng với phần bù $\overline{A}$
- tuyển của A và B, ký hiệu $A \vee B$ (hay A ***or*** B) tương ứng với $A \cup B$
- hội của A và B, ký hiệu $A\ \&\ B$ (hay A ***and*** B) tương ứng với $A \cap B$

Các mệnh đề, cùng với các phép toán trên nó, lập nên một đại số, gọi là *đại số mệnh đề* (còn gọi là *đại số lôgic*). Như thế, đại số mệnh đề và đại số tập hợp là hai *đại số đẳng cấu* với nhau. Tuỳ tình huống, một bài toán có thể phát biểu bằng ngôn ngữ của đại số tập hợp hoặc bằng ngôn ngữ của đại số mệnh đề.

1.2.4. Tích Đêcac của các tập hợp

Ngoài các phép toán của đại số tập hợp, người ta còn định nghĩa một phép toán cho phép ghép hai tập hợp để được một tập hợp mới, đó là tích Đêcac

$$A \times B = \{(a, b) \mid a \in A, b \in B\}.$$

Tích Đêcac được mở rộng tự nhiên cho trường hợp nhiều tập hợp:

$$A_1 \times A_2 \times \ldots \times A_k = \{(a_1, a_2, \ldots, a_k) \mid a_i \in A_i,\ i = 1, 2, \ldots, k\}.$$

Người ta cũng dùng ký hiệu luỹ thừa để biểu diễn tích Đêcac của cùng một tập hợp:

$$A^k = A \times A \times \ldots \times A \qquad (k \text{ lần}).$$

Thí dụ:

R biểu diễn các điểm trên đường thẳng,
R^2 biểu diễn các điểm trên mặt phẳng,
R^3 biểu diễn các điểm trong không gian.

1.2.5. Quan hệ tương đương và phân hoạch

Trong nhiều vấn đề, người ta cần quan tâm đến một quan hệ nào đó giữa hai phần tử của tập hợp đang xét. Một quan hệ hai ngôi R trên tập hữu hạn phần tử X được định nghĩa như là tập con $R(X)$ của tích Đề các $X \times X$. Người ta quan tâm đến các tính chất sau đây của một quan hệ trên tập X:

- đối xứng (a có quan hệ với b kéo theo b có quan hệ với a),
- phản xạ (mọi phần tử có quan hệ với chính nó),
- truyền ứng (nếu a có quan hệ với b và b có quan hệ với c thì a có quan hệ với c).

Thí dụ: Xét $X = \{1, 2, 3, 4\}$. Ta xác định mối quan hệ ρ giữa các phần tử của X như sau: Giả sử $a, b \in X$, ta nói a có quan hệ (ρ) đối với b nếu a chia hết cho b. Khi đó

$$\rho(X) = \{(2,1), (3, 1), (4, 1), (4, 2)\} \subset X \times X.$$

Dễ thấy ρ có tính chất phản xạ (vì rõ ràng là a chia hết cho a), truyền ứng (vì a chia hết cho b và b chia hết cho c kéo theo a chia hết cho c), nhưng không có tính chất phản xạ (vì a chia hết cho b không nhất thiết kéo theo b chia hết cho a).

Có nhiều kiểu quan hệ, nhưng quan hệ được quan tâm nhiều nhất là *quan hệ tương đương*. Một quan hệ được gọi là tương đương nếu nó thoả mãn cả 3 tính chất: đối xứng, phản xạ và truyền ứng. Một quan hệ tương đương trên tập hợp đang xét sẽ chia tập hợp đó thành các lớp (gọi là các *lớp tương đương*) sao cho hai phần tử thuộc cùng một lớp là có quan hệ với nhau và hai phần tử khác lớp là không có quan hệ với nhau. Các lớp tương đương có tính chất phủ kín tập hợp đã cho (tức là một phần tử bất kỳ phải thuộc vào một lớp nào đó) và rời nhau (từng cặp giao với nhau bằng rỗng). Người ta gọi một họ các tập con khác rỗng của một tập hợp có tính chất vừa nêu là một *phân hoạch* của tập hợp đó. Từ đấy suy ra một quan hệ tương đương trên một tập hợp sẽ xác định một phân hoạch trên tập đó và ngược lại, một phân hoạch bất kỳ trên tập hợp đã cho sẽ tương ứng với một quan hệ tương đương trên nó.

Thí dụ: Giả sử xét tập m ($m > 1$) số nguyên dương $N_m = \{1, 2, ..., m\}$. Giả sử k là số nguyên dương, $k < m$. Ta nói hai số nguyên dương $a, b \in N_m$ là có quan hệ với nhau và ký hiệu là $a \leftrightarrow b$ nếu như a và b có cùng số dư khi chia cho k (ký hiệu là $a \equiv b \pmod{k}$). Như vậy,

$a \leftrightarrow b \Leftrightarrow a \equiv b \pmod{k}$.

Dễ dàng kiểm tra được rằng mối quan hệ "$\leftrightarrow$" vừa xác định trên tập N_m là mối quan hệ tương đương. Gọi

$$A_i = \{a \in N_m: a \equiv i \pmod{k}\}, i = 0, 1, ..., k\text{-}1.$$

Khi đó dễ dàng kiểm tra được rằng

$$A_i \cap A_j = \varnothing, i \neq j \quad \text{và} \quad N_m = \bigcup_{i=0}^{k-1} A_i .$$

Điều đó có nghĩa là các tập N_0, N_1, ..., N_{k-1} tạo thành một phân hoạch của tập N_m. Trường hợp riêng khi k=2, tập N_m được phân hoạch thành hai tập: tập các số chẵn (N_0) và tập các số lẻ (N_1).

1.3. Một số nguyên lý cơ bản

1.3.1. Nguyên lý cộng

Nếu A và B là hai tập hợp rời nhau thì

$$N(A \cup B) = N(A) + N(B).$$

Nguyên lý cộng được mở rộng cho nhiều tập con rời nhau:

Nếu $\{A_1, A_2, ..., A_k\}$ là một phân hoạch của tập hợp X thì

$$N(X) = N(A_1) + N(A_2) + ... + N(A_k).$$

Một trường hợp riêng hay dùng của nguyên lý cộng:

Nếu A là một tính chất cho trên tập X thì $N(A) = N(X) - N(\overline{A})$.

Thí dụ 1. Một đoàn vận động viên gồm 2 môn bắn súng và bơi được cử đi thi đấu ở nước ngoài. Nam có 10 người. Số vận động viên thi bắn súng (kể cả nam và nữ) là 14. Số nữ vận động viên thi bơi bằng số nam vận động viên thi bắn súng. Hỏi toàn đoàn có bao nhiêu người?

Giải: Chia đoàn thành 2 lớp: nam và nữ. Lớp nữ lại được chia 2: thi bắn súng và thi bơi. Thay số nữ thi bơi bằng số nam thi bắn súng (2 số này bằng nhau theo đầu bài), ta được số nữ bằng tổng số đấu thủ thi bắn súng. Từ đó, theo nguyên lý cộng, toàn đoàn có 10 + 14 = 24 người.

Thí dụ 2. Trong một đợt phổ biến đề tài tốt nghiệp, Ban chủ nhiệm Khoa công bố danh sách các đề tài bao gồm 80 đề tài về chủ đề "xây dựng hệ thông tin quản lý", 10 đề tài về chủ đề "thiết kế phần mềm dạy học" và 10 đề tài về chủ đề "Hệ chuyên gia". Hỏi một sinh viên có bao nhiêu khả năng lựa chọn đề tài?

Giải: Sinh viên có thể lựa chọn đề tài theo chủ đề thứ nhất bởi 80 cách, theo chủ đề thứ hai bởi 10 cách, theo chủ đề thứ ba bởi 10 cách. Vậy tất cả có 100 cách lựa chọn.

Thí dụ 3. Hỏi rằng giá trị của k sẽ là bao nhiêu sau khi đoạn chương trình PASCAL sau được thực hiện?

```
n1:=10;
n2:=20;
n3:=30;
k:=0;
for i1:= 1 to n1 do k:=k+1;
for i2:= 1 to n2 do k:=k+1;
for i3:= 1 to n3 do k:=k+1;
```

Giải: Đầu tiên giá trị của k được gán bằng 0. Có 3 vòng lặp for độc lập. Sau mỗi lần lặp của mỗi một trong 3 vòng for, giá trị của k tăng lên 1. Vòng for thứ nhất lặp 10 lần, vòng for thứ hai lặp 20 lần, vòng for thứ ba lặp 30 lần. Vậy, kết thúc 3 vòng lặp for giá trị của k sẽ là 10+20+30= 60.

1.3.2. Nguyên lý nhân

Nếu mỗi thành phần a_i của bộ có thứ tự k thành phần $(a_1, a_2, ..., a_k)$ có n_i khả năng chọn $(i = 1, 2, ..., k)$, thì số bộ sẽ được tạo ra là tích số của các khả năng này $n_1 n_2 ... n_k$.

Một hệ quả trực tiếp của nguyên lý nhân:

$$N(A_1 \times A_2 \times ... \times A_k) = N(A_1)\, N(A_2) ... N(A_k),$$

với $A_1, A_2, ..., A_k$ là những tập hợp nào đó, nói riêng:

$$N(A^k) = N(A)^k .$$

Thí dụ 1. Từ Hà nội đến Huế có 3 cách đi: máy bay, ô tô, tàu hoả. Từ Huế đến Sài gòn có 4 cách đi: máy bay, ô tô, tàu hoả, tàu thuỷ. Hỏi từ Hà nội đến Sài gòn (qua Huế) có bao nhiêu cách đi?

Giải: Mỗi cách đi từ Hà nội đến Sài gòn (qua Huế) được xem gồm 2 chặng: Hà nội - Huế và Huế - Sài gòn. Từ đó, theo nguyên lý nhân, số cách đi từ Hà nội đến Sài gòn là $3 \times 4 = 12$ cách.

Thí dụ 2. Hỏi rằng giá trị của k sẽ là bao nhiêu sau khi đoạn chương trình PASCAL sau được thực hiện?

```
n1:=10;
n2:=20;
n3:=30;
k:=0;
for i1:= 1 to n1 do
    for i2:= 1 to n2 do
        for i3:= 1 to n3 do k:=k+1;
```

Giải: Đầu tiên giá trị của k được gán bằng 0. Có 3 vòng lặp for lồng nhau. Sau mỗi lần lặp của vòng for, giá trị của k tăng lên 1. Vòng for thứ nhất lặp 10 lần, vòng for thứ hai lặp 20 lần, vòng for thứ ba lặp 30 lần. Vậy, theo nguyên lý nhân, kết thúc 3 vòng lặp for lồng nhau, giá trị của k sẽ là $10 \times 20 \times 30 = 6000$.

Thí dụ 3. Có bao nhiêu tên biến trong PASCAL độ dài 10 chỉ chứa hai chữ cái A, B, bắt đầu bởi AAA hoặc ABA?

Giải: Tập các tên biến cần đếm được phân hoạch thành hai tập: một tập gồm các biến bắt đầu bởi AAA, còn tập kia gồm các tên biến bắt đầu bởi ABA. Mỗi tên biến độ dài 8 bắt đầu bởi AAA có thể xây dựng như sau: chọn ký tự thứ 4, thứ 5, ..., thứ 10. Mỗi một trong 7 ký tự còn lại này có 2 khả năng chọn (hoặc chọn A, hoặc chọn B), nên theo nguyên lý nhân có

$$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^7 = 128$$

tên biến bắt đầu bởi AAA. Lập luận tương tự ta cũng đếm được 128 tên biến bắt đầu bởi ABA. Vì vậy, theo nguyên lý cộng, có tất cả 128 + 128 = 256 tên biến độ dài 10 chỉ chứa hai chữ A, B hoặc bắt đầu bởi AAA hoặc bắt đầu bởi ABA.

Trong việc giải các bài toán đếm cụ thể, nếu như đếm trực tiếp số cấu hình là khó, ta có thể phân hoạch tập các cấu hình cần đếm ra thành các tập con sao cho việc đếm các phần tử của các tập con này là đơn giản hơn. Khi đó sử dụng nguyên lý cộng để đếm số cấu hình đặt ra.

Nếu chúng ta cần đếm các cấu hình có thể xây dựng theo từng bước, thì khi đó có thể sử dụng nguyên lý nhân.

Nói chung, điều quan trọng khi giải một bài toán đếm là phải xác định được cần sử dụng nguyên lý nào (tổng quát hơn, là công cụ nào) để giải bài toán và điều đó đòi hỏi tư duy của người giải.

1.4. Các cấu hình tổ hợp đơn giản

Dưới đây trình bày một số cấu hình tổ hợp đơn giản, những cấu hình này thường được làm cơ sở cho phép đếm.

1.4.1. Chỉnh hợp lặp

Định nghĩa. *Một chỉnh hợp lặp chập k của n phần tử là một bộ có thứ tự gồm k thành phần lấy từ n phần tử đã cho. Các thành phần có thể được lặp lại.*

Như thế, một chỉnh hợp lặp chập k của n có thể xem như một phần tử của tích Đêcac A^k với A là tập đã cho. Theo nguyên lý nhân, số tất cả các chỉnh hợp lặp chập k của n sẽ là n^k.

Thí dụ 1. Tính số hàm từ một k-tập vào một n-tập.

Giải: Biểu diễn mỗi hàm bằng một bộ k thành phần, trong đó thành phần thứ i là ảnh của phần tử i ($1 \le i \le k$). Mỗi thành phần được lấy từ một trong n giá trị. Từ đó nhận được số cần tìm là n^k.

Thí dụ 2. Tính số dãy nhị phân độ dài n.

Giải: Mỗi dãy nhị phân độ dài n là một bộ gồm n thành phần, trong đó mỗi thành phần chỉ nhận một trong hai giá trị (1 hoặc 0). Từ đó suy ra số các dãy nhị phân độ dài n là 2^n.

Thí dụ 3. Tính số tập con của một n-tập.

Giải: Giả sử n-tập đã cho là $X = \{ x_1, x_2, ..., x_n \}$. Biểu diễn mỗi tập con A của tập đã cho X bằng một dãy nhị phân độ dài n:

$$b = (b_1, b_2, ..., b_n)$$

trong đó $b_i = 1$ nếu phần tử $x_i \in A$ và $b_i = 0$ trong trường hợp ngược lại ($i = 1, 2, ..., n$). Từ đó nhận được số tập con là 2^n.

1.4.2. Chỉnh hợp không lặp

Định nghĩa. *Một chỉnh hợp không lặp chập k của n phần tử là một bộ có thứ tự gồm k thành phần lấy từ n phần tử đã cho. Các thành phần không được lặp lại.*

Để xây dựng một chỉnh hợp không lặp, ta xây dựng dần từ thành phần đầu tiên. Thành phần này có n khả năng chọn. Mỗi thành phần tiếp theo, số khả năng chọn giảm đi 1 so với thành phần đứng trước.Từ đó, theo nguyên lý nhân, số chỉnh hợp không lặp chập k của n sẽ là $n(n\text{-}1)...(n\text{-}k\text{+}1)$. Để tồn tại cấu hình, cần phải thoả mãn $k \leq n$.

Thí dụ. Tính số đơn ánh từ một k-tập vào một n-tập.

Giải: Biểu diễn mỗi đơn ánh bằng bộ ảnh của tập nguồn như trong thí dụ 1 mục trên. Chú ý rằng các ảnh không được lặp lại, ta nhận được số cần tìm là $n(n\text{-}1)...(n\text{-}k\text{+}1)$.

1.4.3. Hoán vị

Định nghĩa. *Ta gọi một hoán vị của n phần tử là một cách xếp thứ tự các phần tử đó.*

Một hoán vị của n phần tử được xem như một trường hợp riêng của chỉnh hợp không lặp khi $k = n$. Do đó số hoán vị của n phần tử là $1.2 \ldots n = n!$

Có thể đồng nhất một hoán vị của n phần tử với một song ánh của một tập n phần tử lên chính nó. Một song ánh như vậy còn được gọi là một *phép thế*. Các phép thế có nhiều tính chất thú vị và việc nghiên cứu nó đã đóng góp một phần quan trọng trong toán học.

Thí dụ 1. 6 người đứng xếp thành một hàng ngang để chụp ảnh. Hỏi có thể bố trí bao nhiêu kiểu?

Giải: Mỗi kiểu ảnh là một hoán vị của 6 người. Từ đó nhận được số kiểu ảnh có thể bố trí là $6! = 720$.

Thí dụ 2. Cần bố trí việc thực hiện n chương trình trên một máy vi tính. Hỏi có bao nhiêu cách?

Giải: Đánh số các chương trình bởi 1, 2,..., n. Mỗi cách bố trí việc thực hiện các chương trình trên máy có thể biểu diễn bởi một hoán vị của 1, 2, ..., n. Từ đó suy ra số cách bố trí cần tìm là $n!$

1.4.4. Tổ hợp

Định nghĩa. *Một tổ hợp chập k của n phần tử là một bộ không kể thứ tự gồm k thành phần khác nhau lấy từ n phần tử đã cho. Nói cách khác, ta có thể coi một tổ hợp chập k của n phần tử là một tập con k phần tử của nó.*

Việc đếm các tổ hợp có khó khăn hơn chút ít so với các cấu hình đã trình bày, tuy nhiên cách đếm dưới đây cho biết cách vận dụng các nguyên lý cùng với các kết quả đếm đã biết trong việc đếm một cấu hình mới.

Xét tập hợp tất cả các chỉnh hợp không lặp chập k của n phần tử. Chia chúng thành những lớp sao cho hai chỉnh hợp thuộc cùng một lớp chỉ khác nhau về thứ tự. Rõ ràng các lớp này là một phân hoạch trên tập đang xét và mỗi lớp như thế là tương ứng với một tổ hợp chập k của n. Số chỉnh hợp trong mỗi lớp là bằng nhau và bằng $k!$ (số hoán vị). Số các lớp là bằng số tổ hợp chập k của n. Theo nguyên lý cộng, tích của $k!$ với số này là bằng số các chỉnh hợp không lặp chập k của n, nghĩa là bằng $n(n\text{-}1)...(n\text{-}k\text{+}1)$. Từ đó nhận được số tổ hợp chập k của n là

$$\frac{n(n-1)(n-2)...(n-k+1)}{k!} \quad hay \quad \frac{n!}{k!(n-k)!} \tag{1}$$

Số tổ hợp này gặp khá nhiều trong toán học, nó thường được ký hiệu bởi C_n^k và được gọi là *hệ số tổ hợp*.

Khi nhận xét rằng, giá trị của phép chia trong (1) là một số nguyên, ta nhận được một kết quả lý thú trong số học: *tích của k số tự nhiên liên tiếp bao giờ cũng chia hết cho k!*.

Thí dụ 1. Có n đội bóng thi đấu vòng tròn. Hỏi phải tổ chức bao nhiêu trận đấu?

Giải: Cứ 2 đội thì có một trận. Từ đó suy ra số trận đấu sẽ bằng số cách chọn 2 đội từ n đội, nghĩa là bằng

$$C_n^2 = \frac{n(n-1)}{2}.$$

Thí dụ 2. Hỏi có bao nhiêu giao điểm của các đường chéo của một đa giác lồi n $(n \geq 4)$ đỉnh nằm ở trong đa giác, nếu biết rằng không có ba đường chéo nào đồng quy tại điểm ở trong đa giác?

Giải: Cứ 4 đỉnh của đa giác thì có một giao điểm của hai đường chéo nằm trong đa giác. Từ đó suy ra số giao điểm cần đếm là

$$C_n^4 = \frac{n(n-1)(n-2)(n-3)}{24}.$$

Dưới đây là một vài tính chất của các hệ số tổ hợp:

a) Đối xứng

$$C_n^k = C_n^{n-k} \tag{2}$$

b) Điều kiện đầu

$$C_n^0 = C_n^n = 1 \tag{3}$$

c) Công thức đệ qui

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k, \quad n > k > 0 \tag{4}$$

Các công thức (2), (3) được suy trực tiếp từ định nghĩa tổ hợp, còn công thức (4) có thể được chứng minh từ (1).

Từ (3) và (4), ta có thể tính tất cả các hệ số tổ hợp chỉ bằng phép cộng. Các hệ số này được tính và viết lần lượt theo từng dòng (mỗi dòng ứng với một giá trị n=0, 1, ...), trên mỗi dòng chúng được tính và viết lần lượt theo từng cột (mỗi cột ứng với một giá trị k = 0, 1, ..., n) theo bảng tam giác dưới đây:

$$\begin{array}{ccccccccccc}
 & & & & C_0^0 & & & & & & \\
 & & & C_1^0 & & C_1^1 & & & & & \\
 & & \cdots & & \cdots & & \cdots & & & & \\
 & \cdots & & \cdots & & \cdots & & \cdots & & & \\
C_n^0 & & C_n^1 & & \cdots & & C_n^{n-1} & & C_n^n & & \\
\cdots & & \cdots & & \cdots & & \cdots & & \cdots & & \cdots
\end{array}$$

Bảng này được gọi là *tam giác Pascal.*

Dưới đây là tam giác Pascal kích thước 8:

$$\begin{array}{ccccccccccccccc}
 & & & & & & & 1 & & & & & & & \\
 & & & & & & 1 & & 1 & & & & & & \\
 & & & & & 1 & & 2 & & 1 & & & & & \\
 & & & & 1 & & 3 & & 3 & & 1 & & & & \\
 & & & 1 & & 4 & & 6 & & 4 & & 1 & & & \\
 & & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & & \\
 & 1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1 & \\
1 & & 7 & & 21 & & 35 & & 35 & & 21 & & 7 & & 1 \\
1 & & 8 & & 28 & & 56 & & 70 & & 56 & & 28 & & 8 & & 1
\end{array}$$

Các hệ số tổ hợp có liên quan chặt chẽ với việc khai triển luỹ thừa của một nhị thức. Thật vậy, trong tích

$$(x + y)^n = (x + y)(x + y)...(x + y)$$

hệ số của $x^k y^{n-k}$ sẽ là số cách chọn k nhân tử $(x + y)$ mà từ đó ta lấy ra x và đồng thời trong n-k nhân tử còn lại ta lấy ra y, nghĩa là

$$(x+y)^n = C_n^0 x^n + C_n^1 x^{n-1} y + \ldots + C_n^{n-1} x\, y^{n-1} + C_n^n y^n \qquad (5)$$
$$= \sum_{k=0}^{n} C_n^k x^{n-k} y^k .$$

Công thức (5) còn được gọi là *khai triển nhị thức Newton* và các hệ số tổ hợp còn được gọi là các *hệ số nhị thức*.

Chẳng hạn, căn cứ vào dòng cuối của tam giác Pascal kích thước 8 (đã tính ở trên), ta nhận được:

$$(x+y)^8 = x^8 + 8x^7y + 28x^6y^2 + 56x^5y^3 + 70x^4y^4 + 56x^3y^5 + 28x^2y^6 + 8xy^7 + y^8$$

Thông thường, công thức (5) được gặp dưới dạng đa thức một ẩn:

$$(x+1)^n = C_n^0 x^n + C_n^1 x^{n-1} + \ldots + C_n^{n-1} x + C_n^n \qquad (6)$$

Rất nhiều đẳng thức về hệ số tổ hợp được suy từ (6). Chẳng hạn, trong (6) chọn $x = 1$ ta được:

$$C_n^0 + C_n^1 + \ldots + C_n^{n-1} + C_n^n = 2^n \qquad (7)$$

tức là nhận được kết quả đã biết (thí dụ 3, mục 1.4.1): số các tập con của một n-tập bằng 2^n, còn nếu chọn $x = -1$ ta được:

$$C_n^0 - C_n^1 + \ldots + (-1)^n C_n^n = 0 \qquad (8)$$

tức là số các tập con chẵn (có số phần tử là số chẵn) bằng các số tập con lẻ và bằng 2^{n-1}.

Nhiều tính chất của hệ số tổ hợp có thể thu được từ (6) bằng cách lấy đạo hàm hoặc tích phân theo x hai vế của đẳng thức này một số hữu hạn lần, sau đó gán cho x những giá trị cụ thể. Chẳng hạn, công thức sau đây thu được bằng cách lấy đạo hàm hai vế theo x và sau đó trong đẳng thức thu được đặt $x = 1$:

$$n\,2^{n-1} = n\,C_n^0 + (n-1)\,C_n^1 + \ldots + C_n^{n-1} .$$

Còn công thức sau đây thu được bằng cách lấy tích phân hai vế theo x và sau đó trong đẳng thức thu được đặt $x = 1$:

$$(n+1)2^{n+1} = (n+1)C_n^0 + nC_n^1 + \ldots + C_n^n .$$

Bài tập

1. Cho biết trong các hệ thức dưới đây hệ thức nào là đúng hệ thức nào là sai

a) $A \subseteq A \cap B$

b) $C \subseteq (A \cap B) \cup C$

c) $A \cup B \subseteq A \cap B$

d) $A \cap (B \cup A) = A \cap B$

e) $(A \cup B) \backslash (A \cap B) = A \backslash B$

2. Ký hiệu Z là tập các số nguyên. Xét hai tập con của Z:

$A = \{x \in Z : x = 4p\text{-}1$ với một $p \in Z$ nào đó$\}$

$B = \{y \in Z : y = 4q\text{-}5$ với một $q \in Z$ nào đó$\}$.

Chứng minh rằng $A = B$.

3. Xét hai tập

$A_1 = \{n \in Z : n < 0\}$ và $A_2 = \{n \in Z : n > 0\}$.

Hỏi A_1, A_2 có tạo thành phân hoạch của Z hay không? Nếu đúng hãy giải thích câu trả lời, nếu sai hãy đưa ra phân hoạch đúng của Z.

4. Cho $A = \{0, 1, 2, 3, 4\}$ và xác định quan hệ R trên A bởi:

$R = \{ (0, 0), (2, 1), (0, 3), (1, 1), (3, 0), (1, 4), (4, 1), (2, 2), (2, 4), (3, 3),$
$(4, 4), (1, 2), (4, 2) \}$.

Chỉ ra rằng quan hệ R là quan hệ tương đương hay không? Nếu câu trả lời là khẳng định hãy đưa ra phân hoạch của A thành các lớp tương đương theo quan hệ R đã cho.

5. Xét các tập với các phần tử là các số nguyên:

$A_0 = \{ ..., -10, -5, 0, 5, 10, 15, 20, 25, ...\}$;
$A_1 = \{ ..., -9, -4, 1, 6, 11, 16, 21, 26, ...\}$;
$A_2 = \{ ..., -8, -3, 2, 7, 12, 17, 22, 27, ...\}$;
$A_3 = \{ ..., -7, -2, 3, 8, 13, 18, 23, 28, ...\}$;
$A_4 = \{ ..., -6, -1, 4, 9, 14, 19, 24, 29, ...\}$.

a) Chỉ ra rằng các tập A_0, A_1, A_2, A_3 và A_4 tạo thành phân hoạch của tập số nguyên Z.

b) Chỉ ra quan hệ s tương ứng với phân hoạch này.

2

BÀI TOÁN ĐẾM

2.1. Giới thiệu bài toán

Một trong những vấn đề đầu tiên của việc nghiên cứu tổ hợp là đếm xem có bao nhiêu cấu hình tổ hợp có thể được tạo ra với những quy tắc đã nêu? Những bài toán như vậy được gọi là *bài toán đếm tổ hợp*. Thông thường, lời giải của bài toán đếm phụ thuộc vào một số giá trị tham số ban đầu và người ta cố gắng biểu diễn sự phụ thuộc này bằng những công thức toán học. Nói chung, để đếm các cấu hình đã cho, người ta tìm cách đưa về các cấu hình quen thuộc bằng cách thiết lập một tương quan 1-1 giữa chúng. Nhiều khi một bài toán đếm được phân thành những bài toán đếm nhỏ hơn bằng cách chia việc đếm thành từng lớp để áp dụng nguyên lý cộng hoặc phân tích cấu hình cần đếm như là việc ghép một số cấu hình khác để áp dụng nguyên lý nhân. Dưới đây là một số thí dụ đơn giản nhằm minh họa một số kỹ thuật đếm.

Thí dụ 1. Có bao nhiêu cách xếp 5 người đứng thành một hàng ngang sao cho A không đứng cạnh B ?

Giải: Để đếm số cách xếp này, ta đếm phần còn lại: số cách xếp mà A đứng cạnh B. Xem A và B như một chỗ, ta có 4! = 24 cách xếp. Số này cần được nhân 2 vì A có thể

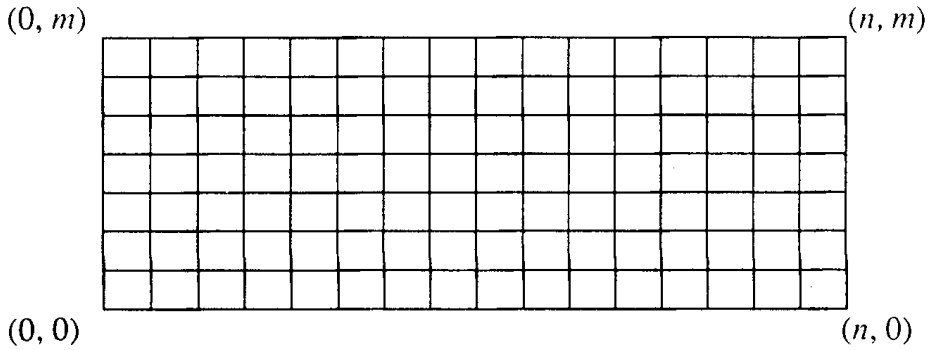
đứng bên trái cũng như bên phải B. Như vậy có tất cả 48 cách xếp A đứng cạnh B. Toàn bộ có 5! = 120 cách xếp. Từ đó nhận được số cách xếp mà A không đứng cạnh B là 120 - 48 = 72 cách.

Thí dụ 2. Một đợt phát hành xổ số với các số vé gồm 2 phần: phần đầu gồm 2 chữ cái lấy từ A đến Z (26 phần tử) và phần sau gồm 4 chữ số lấy từ 0 đến 9 (10 phần tử). Hỏi xác suất để trúng giải độc đắc là bao nhiêu ?

Giải: Trước hết ta đếm số vé được phát hành. Mỗi vé gồm 2 phần: phần chữ và phần số. Phần chữ có 26^2 khả năng, phần số có 10^4 khả năng. Theo nguyên lý nhân, số vé được phát hành là $26^2 \times 10^4 = 6\,760\,000$. Từ đó nhận được xác suất để trúng giải độc đắc là

$$1 / 6\,760\,000 \approx 1{,}48 \times 10^{-7}$$

Thí dụ 3. Cho một lưới gồm các ô vuông. Các nút được đánh số từ 0 đến n theo chiều từ trái sang phải và từ 0 đến m theo chiều từ dưới lên trên (xem hình vẽ). Hỏi có bao nhiêu đường đi khác nhau từ nút (0, 0) đến nút (n, m) nếu chỉ cho phép đi trên cạnh các ô vuông theo chiều sang phải hoặc lên trên ?



Giải: Một đường đi như thế được xem gồm $n+m$ đoạn (mỗi đoạn là một cạnh ô vuông). Tại mỗi đoạn chỉ được chọn một trong 2 giá trị: đi lên (mà ta mã là 1) hay sang phải (mà ta mã là 0). Số đoạn đi lên đúng bằng m và số đoạn sang phải đúng bằng n. Bài toán dẫn về việc tìm xem có bao nhiêu dãy nhị phân độ dài $n + m$ trong đó có đúng m thành phần bằng 1. Đây cũng chính là số tập con m phần tử của một tập $n + m$ phần tử, vì thế số đường đi cần đếm bằng C_{n+m}^{m}.

Thí dụ 4. Thuật toán "nổi bọt" dùng để xếp tăng dần dãy a_i (i = 1,2,..., n) được mô tả bằng đoạn chương trình PASCAL dưới đây:

```
For i := 2 to n do
    For j := n downto i do
```

If a[j-1] > a[j] then Swap(a[j-1], a[j]);

Hãy đếm xem phải làm bao nhiêu phép so sánh ?

Giải: Ta chia số phép so sánh thành các lớp theo vòng lặp i (i đi từ 2 đến n). Với mỗi i xác định, phải thực hiện n-i+1 phép so sánh. Từ đó nhận được, theo nguyên lý cộng, số các phép so sánh là:

$$(n-1)+(n-2)+\ldots+1 = \frac{n(n-1)}{2}$$

Có thể lý luận gọn hơn: thuật toán "nổi bọt" viết trong đoạn chương trình đã cho phải so sánh tất cả các cặp phần tử khác nhau. Từ đó nhận được số phép so sánh là

$$C_n^2 = \frac{n(n-1)}{2}$$

Một đặc tính của các bài toán đếm tổ hợp là số cấu hình tăng rất nhanh khi số giá trị tham gia vào việc tạo nên cấu hình đó tăng. Điều này thường dẫn đến các con số khổng lồ mặc dù các con số tham gia ban đầu không lớn. Hiện tượng này thường được gọi là sự bùng nổ tổ hợp và chính nó là nguyên nhân làm cho các thuật toán dựa vào việc duyệt toàn bộ trở nên không khả thi. Thí dụ dưới đây cho thấy rằng, dù qui cách tạo cấu hình có vẻ rất hạn chế nhưng số cấu hình được tạo, hoá ra lại rất lớn.

Thí dụ 5. Ngôn ngữ PASCAL chuẩn qui định đặt tên biến không quá 8 ký tự. Các ký tự trong tên biến chỉ được phép là các chữ cái (từ A đến Z) hoặc các chữ số (từ 0 đến 9) và phải bắt đầu bằng chữ cái. Hỏi có thể định nghĩa bao nhiêu biến khác nhau ?

Giải: Ta phân các biến thành các lớp: 1-ký tự, 2-ký tự, ... Số các biến thuộc lớp k-ký tự, theo nguyên lý nhân, bằng $26 \times 36^{k-1}$ (k = 1, 2, ..., 8). Từ đó, theo nguyên lý cộng, ta nhận được số các biến khác nhau là:

$26.(1 + 36 + 36^2 + \ldots + 36^7) = 2\,095\,681\,645\,538.$

2.2. Nguyên lý bù trừ

Một số bài toán đếm phức tạp hơn, được dựa vào nguyên lý tổng quát của nguyên lý cộng. Nếu không có giả thiết gì về sự rời nhau giữa 2 tập A và B thì

$$N(A \cup B) = N(A) + N(B) - N(A \cap B). \qquad (1)$$

Công thức (1) được mở rộng cho trường hợp nhiều tập như sau.

Định lý. *Giả sử $A_1, A_2, \ldots, A_m$ là các tập hữu hạn. Khi đó*

$$N(A_1 \cup A_2 \cup \ldots A_m) = N_1 - N_2 + \ldots + (-1)^{m-1} N_m \ , \qquad (2)$$

trong đó N_k là tổng phần tử của tất cả các giao của k tập lấy từ m tập đã cho (nói riêng $N_1 = N(A_1) + ... + N(A_m)$, $N_m = N(A_1 \cap A_2 \cap ... \cap A_m)$).

Chứng minh. Chú ý rằng, số các giao của k tập lấy từ m tập bằng C_m^k , $k = 1, 2, ..., m$. Để chứng minh công thức (1), ta sẽ tính xem mỗi phần tử của tập $A_1 \cup A_2 \cup ... \cup A_m$ được đếm bao nhiêu lần trong vế phải của nó. Xét một phần tử tuỳ ý $a \in A_1 \cup A_2 \cup ... \cup A_m$. Giả sử a là phần tử của k tập trong số m tập đã cho. Khi đó a được đếm ở vế phải của công thức (1)

$$C_k^1 - C_k^2 + C_k^3 - ... + (-1)^{k-1} C_k^k$$

lần. Do

$$C_k^1 - C_k^2 + C_k^3 - ... + (-1)^{k-1} C_k^k$$

$$= 1 - [\, 1 - C_k^1 + C_k^2 - C_k^3 + ... + (-1)^k C_k^k \,] = 1 - (1 - 1)^k = 1,$$

suy ra mỗi phần tử $a \in A_1 \cup A_2 \cup ... \cup A_m$ được tính đúng 1 lần ở vế phải của công thức (1), điều đó đã chứng minh tính đúng đắn của công thức (1).

Bây giờ ta đồng nhất tập A_k với tính chất A_k cho trên một tập X nào đó và đếm xem có bao nhiêu phần tử của X không thoả mãn bất cứ một tính chất A_k nào cả.

Gọi $\overline{N}$ là số cần đếm, N là số phần tử của X, ta có:

$$\overline{N} = N - N(A_1 \cup A_2 \cup ... A_m) = N - N_1 + N_2 - ... + (-1)^m N_m \qquad (3)$$

trong đó N_k là tổng các phần tử của X thoả mãn k tính chất lấy từ m tính chất đã cho.

Công thức (3) được gọi là *nguyên lý bù trừ*. Nó cho phép tính $\overline{N}$ qua các N_k trong trường hợp các số này dễ tính toán hơn.

Ta sẽ xét một số thí dụ minh hoạ cho việc sử dụng nguyên lý bù trừ để giải các bài toán đếm.

Thí dụ 1. Hỏi trong tập $X = \{1, 2, ..., 10000\}$ có bao nhiêu số không chia hết cho bất cứ số nào trong các số 3, 4, 7?

Giải. Gọi

$$A_i = \{\, x \in X : x \text{ chia hết cho } i \,\}, \; i = 3, 4, 7.$$

Khi đó $A_3 \cup A_4 \cup A_7$ là tập các số trong X chia hết cho ít nhất một trong 3 số 3, 4, 7, suy ra theo công thức (3), số lượng các số cần đếm sẽ là

$$N(X) - N(A_3 \cup A_4 \cup A_7) = N_1 - N_2 + N_3.$$

Ta có

$$N_1 = N(A_3) + N(A_4) + N(A_7)$$
$$= [10000/3] + [10000/4] + [10000/7]$$
$$= 3333 + 2500 + 1428 = 7261,$$

$N_2 = N(A_3 \cap A_4) + N(A_3 \cap A_7) + N(A_4 \cap A_7)$
$= [10000/(3\times4)] + [10000/(3\times7)] + [10000/(4\times7)]$
$= 833 + 476 + 357 = 1666,$
$N_3 = N(A_3 \cap A_4 \cap A_7) = [10000/(3\times4\times7)] = 119,$

ở đây ký hiệu $[r]$ để chỉ số nguyên lớn nhất không vượt quá r.

Từ đó số lượng các số cần đếm là 10000 - 7261 + 1666 - 119 = 4286.

Thí dụ 2. Có bao nhiêu xâu nhị phân độ dài 10 hoặc là bắt đầu bởi 00 hoặc là kết thúc bởi 11?

Giải. Dễ thấy là số xâu nhị phân độ dài 10 bắt đầu bởi 00 là $2^8 = 256$ và số xâu nhị phân độ dài 10 kết thúc bởi 11 là $2^8 = 256$. Ngoài ra, số xâu nhị phân độ dài 10 bắt đầu bởi 00 và kết thúc bởi 11 là $2^6 = 64$. Theo công thức (1) suy ra số xâu nhị phân hoặc bắt đầu bởi 00 hoặc kết thúc bởi 11 là

$$256 + 256 - 64 = 448.$$

Kết thúc mục này, ta xét bài toán cổ điển dưới đây trong lý thuyết xác suất:

Bài toán bỏ thư. Có n lá thư và n phong bì ghi sẵn địa chỉ. Bỏ ngẫu nhiên các lá thư vào các phong bì. Hỏi xác suất để xảy ra không một lá thư nào bỏ đúng địa chỉ là bao nhiêu?

Giải: Có tất cả $n!$ cách bỏ thư. Vấn đề còn lại là đếm số cách bỏ thư sao cho không có lá thư nào đúng địa chỉ. Gọi X là tập hợp tất cả các cách bỏ thư và A_k là tính chất lá thư thứ k bỏ đúng địa chỉ. Khi đó theo công thức (3) ta có:

$$\overline{N} = N - N_1 + N_2 - \ldots + (-1)^n N_n$$

trong đó $\overline{N}$ là số cần tìm, $N = n!$, còn N_k là số tất cả các cách bỏ thư sao cho có k lá thư đúng địa chỉ. Nhận xét rằng, N_k là tổng theo mọi cách lấy k lá thư từ n lá, với mỗi cách lấy k lá thư, có $(n\text{-}k)!$ cách bỏ để k lá này đúng địa chỉ, ta nhận được:

$$N_k = C_n^k (n-k)! = \frac{n!}{k!}$$

và

$$\overline{N} = n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \ldots + \frac{(-1)^n}{n!}\right)$$

Từ đó xác suất cần tìm là

$$1 - \frac{1}{1!} + \frac{1}{2!} - \ldots + \frac{(-1)^n}{n!}$$

Một điều lý thú là xác suất này dần đến e^{-1} (nghĩa là còn lớn hơn 1/3) khi n khá lớn. Số $\overline{N}$ trong bài toán trên được gọi là *số mất thứ tự* và được ký hiệu là D_n.. Dưới đây là một vài giá trị của D_n, cho ta thấy D_n tăng nhanh thế nào so với n:

n	2	3	4	5	6	7	8	9	10	11
D_n	1	2	9	44	265	1854	14833	133496	1334961	4890741

2.3. Quy về các bài toán đơn giản

Một trong những phương pháp đếm là quy bài toán đang xét về những bài toán đơn giản hơn. Điều này không phải lúc nào cũng dễ vì nó thường đòi hỏi một sự phân tích sâu sắc cấu hình cần đếm. Thí dụ dưới đây trình bày một bài toán nổi tiếng của Lucas (1891), qua đó rút ra được nhiều điều bổ ích trong nghệ thuật đếm.

Bài toán xếp khách của Lucas. Có một bàn tròn, xung quanh có $2n$ ghế. Cần sắp chỗ cho n cặp vợ chồng sao cho các ông ngồi xen kẽ các bà và không có cặp vợ chồng nào ngồi cạnh nhau. Hỏi có tất cả bao nhiêu cách xếp ?

Giải: Gọi số phải tìm là M_n. Xếp cho các bà trước (cứ một ghế xếp thì một ghế để trống dành cho các ông). Số cách xếp cho các bà là $2n!$. Gọi số cách xếp các ông ứng với một cách xếp các bà là U_n, ta được số cách xếp là

$$M_n = 2n! \times U_n.$$

Vấn đề còn lại là đếm số U_n.

Đánh số các bà (đã xếp) từ 1 đến n, đánh số các ông tương ứng với các bà (ông i là chồng bà i), sau đó đánh số các ghế trống theo nguyên tắc: ghế số i nằm giữa bà i và bà $i+1$ (để tiện trình bày, các phép cộng chỉ số trong phần này đều được hiểu là thực hiện vòng tròn, nghĩa là $n+1 = 1$). Mỗi cách xếp các ông được biểu diễn bằng một phép thế φ trên tập $\{1, 2, ..., n\}$ với quy ước $\varphi(i) = j$ có nghĩa là ghế i được xếp cho ông j. Theo đầu bài, φ phải thoả mãn

$$\varphi(i) \neq i \text{ và } \varphi(i) \neq i+1. \qquad (*)$$

Như vậy U_n là số tất cả các phép thế φ thoả mãn điều kiện (*). Trong toán học, U_n được gọi là *số phân bố*.

Xét tập hợp tất cả các phép thế φ của $\{1, 2, ..., n\}$. Trên tập này, gọi P_i là tính chất $\varphi(i) = i$ và Q_i là tính chất $\varphi(i) = i+1$. Đặt $P_{n+i} = Q_i$ và ta được, theo nguyên lý bù trừ (tương ứng với $2n$ tính chất P_i):

$$U_n = \overline{N} = n! - N_1 + N_2 - ...$$

trong đó N_k là tổng số tất cả các phép thế thoả mãn k tính chất, lấy từ $2n$ tính chất đang xét.

Chú ý rằng, không thể xảy ra đồng thời thoả mãn P_i và Q_i hoặc đồng thời thoả mãn P_{i+1} và Q_i, do đó trong các cách lấy ra k tính chất từ $2n$ tính chất đang xét, cần thêm vào điều kiện: các P_i và Q_i hoặc P_{i+1} và Q_i không được đồng thời có mặt. Gọi số các cách này là $g(2n, k)$ (nói riêng $g(2n, k) = 0$ khi $k > n$). Với mỗi cách lấy ra k tính chất như vậy ($k \le n$), ta có $(n-k)!$ phép thế thoả mãn chúng. Từ đó nhận được $N_k = g(2n, k).(n-k)!$ và

$$U_n = n! - g(2n, 1).(n-1)! + g(2n, 2).(n-2)! - ... + (-1)^n g(2n, n) .$$

Bây giờ còn phải tính các hệ số $g(2n, k)$, $k = 1, 2, ..., n$.

Xếp $2n$ tính chất đang xét trên một vòng tròn theo thứ tự $P_1, Q_1, P_2, Q_2, ..., P_n, Q_n$, ta thấy rằng $g(2n, k)$ chính là số cách lấy ra k phần tử trong $2n$ phần tử xếp thành vòng tròn sao cho không có 2 phần tử nào kề nhau cùng được lấy ra.

Để tính $g(2n, k)$ ta giải 2 bài toán con sau đây:

Bài toán 1. Có bao nhiêu cách lấy ra k phần tử trong n phần tử xếp trên đường thẳng sao cho không có 2 phần tử kề nhau cùng được lấy ra ?

Giải: Khi lấy ra k phần tử, ta còn $n-k$ phần tử. Giữa $n-k$ phần tử này có $n-k+1$ khoảng trống (kể cả 2 đầu). Mỗi cách lấy ra k khoảng từ các khoảng này, sẽ tương ứng với một cách chọn k phần tử thoả mãn yêu cầu đã nêu. Vậy số cách cần tìm là C_{n-k+1}^k.

Bài toán 2. Giống như bài toán 1, nhưng với n phần tử xếp trên vòng tròn.

Giải: Cố định phần tử a trong n phần tử. Chia các cách lấy thành 2 lớp:

1. Các cách mà a được chọn, khi đó 2 phần tử kề a sẽ không được chọn và ta phải lấy $k-1$ phần tử từ $n-3$ phần tử còn lại. Các phần tử này được xem như trên đường thẳng. Theo bài toán 1, số cách thuộc lớp này là C_{n-k-1}^{k-1}.

2. Các cách mà a không được chọn, khi đó bỏ a đi, ta đưa về bài toán lấy k phần tử từ $n-1$ phần tử xếp trên đường thẳng. Theo bài toán 1, số cách thuộc lớp này là C_{n-k}^k .

Vậy, theo nguyên lý cộng, số cách cần tìm là

$$C_{n-k-1}^{k-1} + C_{n-k}^k = \frac{n}{n-k} C_{n-k}^k \quad .$$

Từ kết quả của bài toán 2, ta nhận được

$$g(2n, k) = \frac{2n}{2n-k} C_{2n-k}^k$$

và số phân bố U_n được tính bằng

$$U_n = n! - \frac{2n}{2n-1} C_{2n-1}^1 (n-1)! + \frac{2n}{2n-2} C_{2n-2}^2 (n-2)! - ... + (-1)^n \frac{2n}{n} C_n^n$$

(Touchard J. 1934 France).

Dưới đây là một vài giá trị của U_n, một lần nữa minh hoạ hiện tượng bùng nổ tổ hợp:

n	2	3	4	5	6	7	8	9	10
U_n	0	1	2	13	80	579	4738	43387	439792

2.4. Công thức truy hồi

Thông thường, người ta quan tâm đến những bài toán đếm, trong đó kết quả đếm phụ thuộc vào một tham số tại đầu vào (mà ta ký hiệu là n), thí dụ các số D_n, U_n, ... Việc biểu diễn kết quả này, như một hàm của n, bằng một số hữu hạn các phép toán, không phải là đơn giản. Người ta nhận thấy rằng, trong nhiều trường hợp, việc tìm kiếm một công thức trực tiếp giữa kết quả đếm và giá trị n là rất khó khăn (hoặc không thể được), trong khi đó, công thức liên hệ giữa kết quả đếm ứng với n và các kết quả đếm ứng với các giá trị n bé hơn lại đơn giản và dễ tìm. Nhờ công thức này và một vài giá trị ban đầu, ta có thể tính mọi giá trị còn lại khác. Công thức đó gọi là công thức truy hồi hay công thức đệ quy. Do tính kế thừa, công thức truy hồi rất thích hợp với việc lập trình trên máy tính. Nó cho phép giảm đáng kể độ phức tạp cũng như gia tăng độ ổn định của quá trình tính toán.

2.4.1. Các thí dụ minh họa

Trước tiên chúng ta xét một số thí dụ minh hoạ việc xây dựng công thức truy hồi để giải các bài toán đếm.

Thí dụ 1. Tính số mất thứ tự D_n.

Giải: Đánh số thư và phong bì từ 1 đến n (thư i gửi đúng địa chỉ nếu bỏ vào phong bì i). Một cách bỏ thư được đồng nhất với hoán vị $(a_1, ..., a_n)$ của $\{1, 2, ..., n\}$. Một mất thứ tự được định nghĩa là một hoán vị $(a_1, ..., a_n)$ sao cho $a_i \neq i$ với mọi i. Thành phần a_1 có thể nhận n-1 giá trị ngoài 1. Với mỗi giá trị k ($k \neq 1$) của a_1, xét 2 trường hợp:

1. $a_k = 1$, khi đó các thành phần còn lại được xác định như một mất thứ tự của n-2 phần tử, tức là số các mất thứ tự thuộc loại này bằng D_{n-2}.

2. $a_k \neq 1$, khi đó các thành phần từ 2 đến n được xác định như một mất thứ tự của n-1 phần tử (xem giá trị 1 như là giá trị k), tức là số mất thứ tự thuộc loại này bằng D_{n-1}.

Từ đó nhận được công thức

$$D_n = (n - 1)(D_{n-2} + D_{n-1}), \quad n \geq 3.$$

Các giá trị ban đầu dễ dàng được tìm trực tiếp: $D_1 = 0, D_2 = 1$.
Mọi giá trị còn lại được tìm đơn giản nhờ luật kế thừa:

$D_3 = (3 - 1)(0 + 1) = 2$
$D_4 = (4 - 1)(1 + 2) = 9$
$D_5 = (5 - 1)(2 + 9) = 44$
$D_6 = (6 - 1)(9 + 44) = 265$
$D_7 = (7-1)(265+44) = 1854$
$D_8 = (8-1)(1854+265) = 14833$
.....

Để công thức truy hồi đúng cả đối với $n = 2$, ta xem như $D_0 = 1$.

Có thể nhận được công thức trực tiếp qua công thức truy hồi. Thật vậy, từ

$$D_n = (n - 1)(D_{n-2}+D_{n-1})$$

suy ra

$$D_n - nD_{n-1} = -(D_{n-1} - (n -1)D_{n-2}).$$

Đặt $V_n = D_n - n D_{n-1}$, ta có

$$D_n - nD_{n-1} = V_n = -V_{n-1} = \ldots = (-1)^{n-1} V_1 = (-1)^n$$

hay

$$\frac{D_n}{n!} - \frac{D_{n-1}}{(n-1)!} = \frac{(-1)^n}{n!}$$

Cộng các hệ thức trên, với $n = 1, 2, \ldots$, ta được

$$\frac{D_n}{n!} = 1 - \frac{1}{1!} + \frac{1}{2!} - \ldots + \frac{(-1)^n}{n!}$$

và nhận lại công thức đã biết: $D_n = n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \ldots + \frac{(-1)^n}{n!}\right)$.

Thí dụ 2. Trên mặt phẳng, kẻ n đường thẳng sao cho không có 2 đường nào song song và 3 đường nào đồng quy. Hỏi mặt phẳng được chia thành mấy phần ?

Giải: Gọi số phần mặt phẳng được chia bởi n đường thẳng là S_n. Giả sử đã kẻ n-1 đường thẳng. Bây giờ kẻ thêm đường thẳng thứ n thì số phần được thêm sẽ bằng số giao điểm được thêm cộng với 1. Số giao điểm được thêm là số giao điểm mà đường thẳng vừa kẻ cắt n-1 đường thẳng cũ, nghĩa là bằng n-1. Từ đó nhận được công thức truy hồi

$$S_n = S_{n-1} + n, \quad n \geq 1,$$

với giá trị ban đầu $S_0 = 1$. Từ công thức này, dễ dàng tính mọi giá trị của S_n, $n = 1, 2, \ldots$

$S_1 = 1 + 1 = 2$
$S_2 = 2 + 2 = 4$

$$S_3 = 4 + 3 = 7$$
$$S_4 = 7 + 4 = 11$$
$$S_5 = 11 + 5 = 16$$
$$S_6 = 16 + 6 = 22$$
$$S_7 = 22 + 7 = 29$$
.....

Để tìm công thức trực tiếp, ta cộng các hệ thức $S_k = S_{k-1} + k$ với $k = 1, 2, ..., n$. Sau khi khử các số hạng giống nhau ở hai vế, ta nhận được:

$$S_n = S_0 + 1 + 2 + ... + n = 1 + \frac{n(n+1)}{2} = \frac{n^2 + n + 2}{2}$$

Công thức truy hồi được mở rộng một cách tự nhiên cho trường hợp có nhiều tham số. Khi đó cần một họ các giá trị ban đầu (mà chúng thường được gọi là các giá trị biên). Việc tiếp cận các hệ số tổ hợp như trình bày dưới đây là một thí dụ tìm một công thức như vậy.

Thí dụ 3. Tính hệ số tổ hợp C_n^k .

Giải: Chọn phần tử cố định a trong n phần tử đang xét. Chia số cách chọn tập con k phần tử của tập này thành 2 lớp: chứa a và không chứa a. Nếu a được chọn thì ta phải bổ xung k-1 phần tử từ n-1 phần tử còn lại, từ đó lớp chứa a gồm C_{n-1}^{k-1} cách. Nếu a không được chọn thì ta phải chọn k phần tử từ n-1 phần tử còn lại, từ đó lớp không chứa a gồm C_{n-1}^k cách. Theo nguyên lý cộng, ta được công thức truy hồi:

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$

với các giá trị biên được suy trực tiếp từ định nghĩa:

$$C_n^0 = C_n^n = 1.$$

Rõ ràng việc lập trình theo công thức truy hồi (xem tam giác Pascal) là hiệu quả hơn nhiều so với việc lập trình theo công thức trực tiếp.

Phương pháp tìm công thức trực tiếp từ công thức truy hồi trình bày trong các thí dụ 1-3 được gọi là *phương pháp khử*. Không phải lúc nào cũng dễ dàng khử được công thức truy hồi để đưa được về công thức trực tiếp. Tuy nhiên, trong một số trường hợp đặc biệt ta có thể đưa ra phương pháp tổng quát để giải công thức truy hồi (tức là tìm công thức trực tiếp cho số hạng tổng quát của dãy số thoả mãn công thức đã cho).

Thí dụ 4. (Bài toán tháp Hà nội). Trò chơi tháp Hà nội được trình bày như sau: "*Có 3 cọc a, b, c. Trên cọc a có một chồng gồm n cái đĩa đường kính giảm dần từ dưới lên trên. Cần phải chuyển chồng đĩa từ cọc a sang cọc c tuân thủ qui tắc: mỗi lần chỉ*

chuyển 1 đĩa và chỉ được xếp đĩa có đường kính nhỏ hơn lên trên đĩa có đường kính lớn hơn. Trong quá trình chuyển được phép dùng cọc b làm cọc trung gian". Bài toán đặt ra là: Tìm số lần di chuyển đĩa ít nhất cần thực hiện để thực hiện xong nhiệm vụ đặt ra trong trò chơi tháp Hà nội.

Giải: Gọi h_n là số lần di chuyển đĩa ít nhất cần thực hiện để giải xong bài toán tháp Hà nội. Ta xây dựng công thức đệ qui để tính h_n. Rõ ràng:

$$h_1 = 1.$$

Giả sử $n \geq 2$. Việc di chuyển đĩa gồm các bước:

(i) Chuyển n-1 đĩa từ cọc a đến cọc b sử dụng cọc c làm trung gian. Bước này được thực hiện nhờ giả thiết quy nạp.

(ii) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a đến cọc c.

(iii) Chuyển n-1 đĩa từ cọc b đến cọc c (sử dụng cọc a làm trung gian). Bước này được thực hiện nhờ giả thiết quy nạp.

Bước (i) và (ii) đòi hỏi giải bài toán tháp Hà nội với n-1 đĩa, vì vậy số lần di chuyển đĩa ít nhất cần thực hiện trong hai bước này là $2h_{n-1}$. Do đó ta có công thức đệ qui sau:

$$h_n = h_{n-1} + 1, n \geq 2.$$

Sử dụng công thức đệ qui và điều kiện đầu vừa tìm được đối với h_n ta có thể dễ dàng chứng minh bằng qui nạp là

$$h_n = 2^n - 1, n \geq 1.$$

2.4.2. Giải công thức truy hồi tuyến tính thuần nhất hệ số hằng

Xét công thức truy hồi tuyến tính thuần nhất hệ số hằng bậc k:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + ... + c_k a_{n-k}, \quad (1)$$

trong đó $c_1, c_2, ..., c_k$ là các hằng số, $c_k \neq 0$.

Ta cần tìm công thức trực tiếp cho số hạng a_n của dãy số $\{a_n\}$ thoả mãn công thức (1) (dãy số như vậy sẽ gọi là nghiệm của công thức truy hồi đã cho). Rõ ràng, dãy số $\{a_n\}$ thoả mãn công thức (1) sẽ được xác định duy nhất, nếu như nó phải thoả mãn k điều kiện đầu sau:

$$a_0 = C_0, a_1 = C_1, ..., a_{k-1} = C_{k-1}, \quad (2)$$

trong đó $C_0, C_1, ..., C_{k-1}$ là các hằng số. Ta tìm nghiệm dưới dạng

$$a_n = r^n, \quad (3)$$

trong đó r là hằng số. Dãy số $\{a_n\}$ xác định theo công thức (3) sẽ thoả mãn (1) nếu r là nghiệm của phương trình:

$$r^n = c_1 r^{n-1} + c_2 r^{n-2} + \ldots + c_k r^{n-k},$$

hay là

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \ldots - c_k = 0. \tag{4}$$

Phương trình (4) được gọi là phương trình đặc trưng của công thức (1), và nghiệm của nó được gọi là nghiệm đặc trưng. Chúng ta sẽ xét cách sử dụng nghiệm đặc trưng để xây dựng công thức nghiệm dưới dạng hiện. Trước hết chúng ta xét trường hợp riêng, khi $k = 2$. Sau đó các kết quả sẽ được phát biểu cho trường hợp tổng quát.

Định lý 1. *Cho c_1, c_2 là các hằng số thực. Giả sử phương trình $r^2 - c_1 r - c_2 = 0$ có hai nghiệm phân biệt r_1 và r_2. Khi đó dãy số $\{a_n\}$ là nghiệm của công thức truy hồi*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2}$$

khi và chỉ khi

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n \tag{5}$$

$n = 0, 1, \ldots$, trong đó α_1, α_2 là các hằng số.

Chứng minh. Trước hết ta chứng minh rằng nếu r_1 và r_2 là hai nghiệm phân biệt của phương trình đặc trưng, và α_1, α_2 là các hằng số, thì dãy số $\{a_n\}$ xác định bởi công thức (5) là nghiệm của công thức truy hồi đã cho. Thực vậy, do r_1 và r_2 là nghiệm đặc trưng nên

$$r_1^2 = c_1 r_1 + c_2 \ , \quad r_2^2 = c_1 r_2 + c_2$$

từ đó suy ra

$$\begin{aligned} c_1 a_{n-1} + c_2 a_{n-2} &= c_1(\alpha_1 r_1^{n-1} + \alpha_2 r_2^{n-1}) + c_2(\alpha_1 r_1^{n-2} + \alpha_2 r_2^{n-2}) \\ &= \alpha_1 r_1^{n-2}(c_1 r_1 + c_2) + \alpha_2 r_2^{n-2}(c_1 r_2 + c_2) \\ &= \alpha_1 r_1^{n-2} r_1^2 + \alpha_2 r_2^{n-2} r_2^2 \\ &= \alpha_1 r_1^n + \alpha_2 r_2^n \\ &= a_n . \end{aligned}$$

Bây giờ ta sẽ chỉ ra rằng nghiệm $\{a_n\}$ của hệ thức $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ luôn có dạng (5) với α_1, α_2 nào đó. Thực vậy, giả sử $\{a_n\}$ là nghiệm của hệ thức đã cho với điều kiện đầu

$$a_0 = C_0 \, , \, a_1 = C_1 \ , \tag{6}$$

ta chỉ ra rằng có thể tìm được các số α_1, α_2 để cho (5) là nghiệm của hệ thức với điều kiện đầu này. Ta có

$$\begin{aligned} a_0 &= C_0 = \alpha_1 + \alpha_2 \, , \\ a_1 &= C_1 = \alpha_1 r_1 + \alpha_2 r_2 . \end{aligned}$$

Giải hệ phương trình tuyến tính phụ thuộc hai ẩn α_1, α_2 thu được, do $r_1 \neq r_2$, ta tìm được nghiệm duy nhất

$$\alpha_1 = (C_1 - C_0 r_2)/(r_1 - r_2), \qquad (7)$$

$$\alpha_2 = (C_0 r_1 - C_1)/(r_1 - r_2).$$

Với những giá trị của α_1 , α_2 vừa tìm được, dãy $\{a_n\}$ xác định theo (5) là nghiệm của hệ thức đã cho với điều kiện đầu (6). Do hệ thức đã cho cùng với điều kiện đầu (6) xác định duy nhất một dãy số, nên nghiệm của hệ thức được cho bởi công thức (5). Định lý được chứng minh.

Thí dụ 4. Dãy Fibonaci trong toán học được định nghĩa bằng hệ thức truy hồi:

$$F_n = F_{n-1} + F_{n-2}, n \geq 2,$$

$$F_0 = F_1 = 1.$$

Tìm công thức hiện cho F_n.

Giải: Giải phương trình đặc trưng:

$$r^2 - r - 1 = 0,$$

ta thu được hai nghiệm

$$r_1 = \frac{1+\sqrt{5}}{2}, \qquad r_2 = \frac{1-\sqrt{5}}{2}$$

và công thức hiện có dạng:

$$F_n = \alpha_1.(r_1)^n + \alpha_2.(r_2)^n$$

trong đó α_1, α_2 là các hằng số cần xác định từ các giá trị ban đầu F_0, F_1. Từ công thức (7), ta có:

$$\alpha_1 = \frac{1}{\sqrt{5}}\frac{1+\sqrt{5}}{2}, \alpha_2 = -\frac{1}{\sqrt{5}}\frac{1-\sqrt{5}}{2}$$

và nhận được

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right).$$

Một điều lý thú là công thức này phải dùng các phép toán vô tỉ để biểu diễn một giá trị nguyên.

Định lý 1 không áp dụng được khi phương trình đặc trưng có nghiệm kép. Trong trường hợp đó ta cần sử dụng kết quả của định lý sau.

Định lý 2. *Cho c_1, c_2 là các hằng số thực, $c_2 \neq 0$. Giả sử phương trình $r^2 - c_1 r - c_2 = 0$ có nghiệm kép r_0. Khi đó dãy số $\{a_n\}$ là nghiệm của công thức đệ qui*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2}$$

khi và chỉ khi

$$a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$$

$n = 0, 1, ...$, trong đó α_1 , α_2 là các hằng số.

Chứng minh. Hoàn toàn tương tự như chứng minh định lý 1.

Thí dụ 5. Tìm nghiệm cho công thức truy hồi

$$a_n = 6 a_{n-1} - 9 a_{n-2}$$

với điều kiện đầu $a_0 = 1$ và $a_1 = 6$.

Giải: Phương trình đặc trưng $r^2 - 6 r + 9 = 0$ có nghiệm kép $r = 3$. Do đó nghiệm của hệ thức có dạng:

$$a_n = \alpha_1 3^n + \alpha_2 n 3^n.$$

Để tìm α_1, α_2 , sử dụng điều kiện đầu ta có

$$a_0 = 1 = \alpha_1 ,$$
$$a_1 = 6 = \alpha_1 . 3 + \alpha_2 . 3.$$

Giải hệ này ta tìm được $\alpha_1 = 1$ *và* $\alpha_2 = 1$. Từ đó nghiệm của hệ thức đã cho là:

$$a_n = 3^n + n 3^n .$$

Định lý 3 sau đây là sự tổng quát hoá kết quả của định lý 1 cho trường hợp hệ thức đệ qui tuyến tính thuần nhất hệ số hằng bậc $k > 2$.

Định lý 3. *Cho $c_1, c_2, ..., c_n$ là các số thực. Giả sử phương trình đặc trưng*

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \ldots - c_k = 0$$

có k nghiệm phân biệt $r_1, r_2, ..., r_k$. Khi đó dãy số $\{a_n\}$ là nghiệm của hệ thức

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + ... + c_k a_{n-k},$$

khi và chỉ khi

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n + \ldots + \alpha_k r_k^n$$

với $n = 0, 1, 2,...$, trong đó $\alpha_1, \alpha_2, ..., \alpha_k$ là các hằng số.

Chứng minh: Tương tự như chứng minh định lý 1.

Thí dụ 6. Tìm nghiệm của hệ thức

$$a_n = 6\,a_{n-1} - 11\,a_{n-2} + 6\,a_{n-3}$$

với điều kiện đầu

$$a_0 = 2,\ a_1 = 5,\ a_2 = 15.$$

Giải: Phương trình đặc trưng

$$r^3 - 6r^2 + 11r - 6 = 0$$

có 3 nghiệm $r_1 = 1,\ r_2 = 2, r_3 = 3$. Vì vậy, nghiệm có dạng

$$a_n = \alpha_1 1^n + \alpha_2 2^n + \alpha_3 3^n.$$

Sử dụng các điều kiện đầu ta có hệ phương trình sau đây để xác định các hằng số α_1, α_2, α_3:

$$a_0 = 2 = \alpha_1 + \alpha_2 + \alpha_3$$
$$a_1 = 5 = \alpha_1 + \alpha_2.2 + \alpha_3.3$$
$$a_2 = 15 = \alpha_1 + \alpha_2.4 + \alpha_3.9.$$

Giải hệ phương trình trên ta thu được $\alpha_1 = 1,\ \alpha_2 = -1$ *và* $\alpha_3 = 2$. Vậy nghiệm của hệ thức đã cho là

$$a_n = 1 - 2^n + 2.3^n.$$

Chú ý: Việc tìm nghiệm của hệ thức (1) trong trường hợp tổng quát dẫn về việc giải một phương trình bậc k:

$$r^k - c_1 r^{k-1} - \ldots - c_k = 0$$

mà việc biểu diễn nghiệm của phương trình này qua một số hữu hạn các phép toán, như đã biết, không phải lúc nào cũng làm được với $k \geq 5$ (định lý Abel).

2.5. Phương pháp hàm sinh

2.5.1. Hàm sinh và bài toán đếm

Giả sử $\{h_n \mid n = 0, 1, 2, \ldots.\}$ là một dãy số. Ta viết dãy này như là dãy vô hạn phần tử, tuy nhiên ta coi rằng nó bao gồm cả trường hợp dãy hữu hạn. Nếu $h_0, h_1, \ldots, h_m$ là dãy hữu hạn, thì ta sẽ biến nó thành dãy vô hạn bằng cách đặt $h_i = 0,\ i > m$.

Định nghĩa. *Hàm sinh $g(x)$ của dãy số $\{h_n \mid n = 0, 1, 2, \ldots.\}$ là chuỗi vô hạn*

$$g(x) = h_0 + h_1 x + h_2 x^2 + ... = \sum_{i=0}^{\infty} h_i x^i .$$

Như vậy hàm $g(x)$ sinh ra dãy số đã cho như là dãy các hệ số của nó. Nếu dãy là hữu hạn thì sẽ tìm được m sao cho $h_i = 0$, $i > m$. Trong trường hợp này $g(x)$ là một đa thức bậc m.

Thí dụ 1. Một trong những nguồn gốc dẫn đến định nghĩa hàm sinh chính là định lý về khai triển nhị thức: Hàm

$$g(x) = (1 + x)^m$$

sinh ra dãy các hệ số tổ hợp

$$\{h_k = C(m, k), k=0, 1,..., m\}$$

bởi vì

$$(1+x)^m = \sum_{k=0}^{m} C(m,k)x^k .$$

Thí dụ 2. Hàm

$$g(x) = 1/(1-x)$$

sinh ra dãy

1, 1, 1, ...

Dễ dàng chứng minh điều đó bằng cách thực hiện phép chia:

$$1/(1- x) = 1 + x + x^2 + ...$$

Thí dụ 3. Với $k > 0$, hàm

$$g(x) = 1/(1-x)^k$$

sinh ra dãy

$$\{C(n+k-1, n): n = 0, 1, 2, ...\}.$$

Như vậy hệ số thứ n sẽ là số khả năng chọn n vật từ k loại đồ vật. Thực vậy, ta có

$$1/(1-x)^k = [\ 1/(1-x)]^k = (1 + x + x^2 + ...)^k.$$

Nếu ta khai triển biểu thức này bằng cách thực hiện nhân phá ngoặc, thì số lần xuất hiện số hạng x^n sẽ bằng số nghiệm nguyên không âm của phương trình

$$t_1 + t_2 + ... + t_k = n,$$

mà ta có thể dễ dàng tính được là $C(n+k-1, n)$.

Ví dụ này có thể gợi ý cho ta cách giải nhiều bài toán đếm. Chẳng hạn xét hàm sinh

$$g(x) = (1 + x + x^2 + x^3)(1 + x + x^2)(1 + x + x^2 + x^3 + x^4).$$

Giả sử x^a, x^b, x^c tương ứng là các số hạng lấy từ các thừa số thứ nhất, hai, ba của vế phải, điều đó có nghĩa là $0 \le a \le 3$, $0 \le b \le 2$, $0 \le c \le 4$. Khi khai triển vế phải các thừa số này sẽ cho ta số hạng x^n, với $n = a + b + c$. Như vậy hệ số của x^n trong $g(x)$ sẽ là số nghiệm nguyên không âm của phương trình

$$n = a + b + c$$

thoả mãn

$$0 \le a \le 3, 0 \le b \le 2, 0 \le c \le 4.$$

Suy ra hệ số này cũng cho ta số cách chọn n bông hoa từ 3 bông cúc, 2 bông layơn và 4 bông hồng.

Tất nhiên việc sử dụng hàm sinh để giải bài toán đếm sẽ đòi hỏi nhiều tính toán khi thực hiện phép nhân các đa thức, và không thích hợp cho việc tính tay. Tuy nhiên, việc đó lại có thể thực hiện nhanh chóng trên máy tính, và vì thế hàm sinh sẽ là một công cụ hữu hiệu để giải nhiều bài toán đếm trên máy tính. Hơn nữa hàm sinh sẽ còn là công cụ hữu ích để nghiên cứu các bài toán đếm một cách trừu tượng.

Ta dẫn ra một số khai triển đại số rất hay sử dụng trong việc sử dụng hàm sinh:

$$x^k/(1-x) = x^k(1 + x + x^2 + \ldots) = x^k + x^{k+1} + x^{k+2} + \ldots$$

$$(1-x^{k+1})/(1-x) = 1 + x + x^2 + \ldots + x^k.$$

$$1/(1-x^2) = 1 + x^2 + x^4 + x^6 + \ldots$$

$$x/(1-x^2) = x(1 + x^2 + x^4 + x^6 + \ldots) = x + x^3 + x^5 + x^7 + \ldots$$

Thí dụ 4. Có bao nhiêu cách chọn ra n quả từ 4 loại quả: táo, chuối, cam và đào (mỗi loại đều có số lượng ít ra là n) mà trong đó có một số chẵn quả táo, số lẻ quả chuối, không quá 4 quả cam và ít ra 2 quả đào?

Giải. Hàm sinh để giải bài toán này là

$$(1 + x^2 + x^4 + x^6 + \ldots)(x^3 + x^5 + x^7 + \ldots)(1 + x + x^2 + x^3 + x^4)(x^2 + x^3 + x^4 + \ldots)$$

Trong công thức trên có 4 thừa số để đếm số quả táo (các số mũ chẵn), chuối (số mũ lẻ), cam (chỉ có đến số mũ 4) và đào (số mũ bắt đầu từ 2). Hàm sinh sẽ là

$$g(x) = [1/(1-x^2)]\ [x/(1-x^2)]\ [(1-x^5)/(1-x)]\ [x^2/(1-x)]$$

$= [x^3(1-x^5)]/[(1-x^2)^2(1-x)^2]$.

Câu trả lời là: Số cách cần đếm là hệ số thứ n trong khai triển $g(x)$ dưới dạng chuỗi luỹ thừa. Tuy là chúng ta không có câu trả lời bằng số, nhưng hàm xây dựng được chứa dữ liệu để có thể lập trình trên máy tính đưa ra bảng đáp số cho các giá trị của n mà ta mong muốn.

Trong nhiều trường hợp, việc khai triển hàm sinh dưới dạng chuỗi luỹ thừa có thể thực hiện được bằng tay, chúng ta có thể thu được công thức đếm dưới dạng hiện.

Thí dụ 5. Tìm hàm sinh cho số nghiệm nguyên dương lẻ của phương trình

$$t_1 + t_2 + \ldots + t_k = n.$$

Giải. Hàm sinh cần tìm là

$$g(x) = (x + x^3 + x^5 + x^7 + \ldots)^k$$
$$= [x\,(1 + x^2 + x^4 + x^6 + \ldots)]^k$$
$$= [x\,(1-x^2)]^k = x^k\,(1-x^2)^k.$$

Thí dụ 6. Tìm hàm sinh cho h_n là số cách chọn ra n quả từ 4 loại quả: táo, chuối, cam và đào (mỗi loại đều có số lượng ít ra là n) mà trong đó có một số chẵn quả táo, số lượng chuối chia hết cho 5, không quá 4 quả cam và không quá 1 quả đào?

Giải. Hàm sinh có dạng

$$g(x) = (1 + x^2 + x^4 + x^6 + \ldots)\,(1 + x^5 + x^{10} + x^{15} + \ldots)\,(1 + x + x^2 + x^3 + x^4)\,(1 + x)$$
$$= [1/(1-x^2)]\;[1/(1-x^5)]\;[(1-x^5)/(1-x)]\;(1+x)$$
$$= [1/((1-x)(1+x)]\;[1/(1-x)]\;(1+x)$$
$$= 1/(1-x)^2$$

Từ đó ta có thể tìm công thức hiện cho lời giải, bởi vì

$$1/(1-x)^2 = \sum_{n=0}^{\infty} C^n_{n+2-1}\,x^n = \sum_{n=0}^{\infty} C^n_{n+1}\,x^n = \sum_{n=0}^{\infty}(n+1)\,x^n\,.$$

Vậy $h_n = n + 1$.

Thí dụ 7. Tìm hàm sinh cho số cách đổi n (nghìn đồng) sử dụng các loại giấy bạc mệnh giá 1 nghìn đồng, 5 nghìn đồng, 10 nghìn đồng, 50 nghìn đồng (giả thiết là ta có một số lượng không hạn chế mỗi loại giấy bạc).

Giải. Số lượng tiền cần đổi ra loại giấy bạc 5 nghìn đồng phải chia hết cho 5, số lượng tiền cần đổi ra loại giấy bạc 10 nghìn đồng phải chia hết cho 10, số lượng tiền cần đổi ra loại giấy bạc 50 nghìn đồng phải chia hết cho 50. Vì vậy

$$g(x) = (1 + x + x^2 + ...)\,(1 + x^5 + x^{10} + ...)\,(1 + x^{10} + x^{20} + ...)\,(1 + x^{50} + x^{100} + ...)$$
$$= [1/(1-x)]\ [1/(1-x^5)]\ [1/(1-x^{10})]\ [1/(1-x^{50})].$$

2.5.2. Hàm sinh và công thức đệ qui

Mục này sẽ trình bày phương pháp hàm sinh để tìm công thức dưới dạng hiện cho số hạng tổng quát của dãy số xác định bởi công thức đệ qui. Nội dung của phương pháp có thể trình bày như sau. Giả sử ta có $\{h_n\}$ là dãy số được xác định theo công thức đệ qui. Xây dựng hàm sinh $g(x)$ của dãy số này theo công thức

$$g(x) = h_0 + h_1 x + h_2 x^2 + ... = \sum_{i=0}^{\infty} h_i x^i .$$

Sử dụng các tính chất của dãy số (suy từ công thức đệ qui xác định nó) ta có thể tìm được công thức giải tích cho hàm sinh $g(x)$. Từ công thức tìm được ta sẽ khai triển hàm $g(x)$ dưới dạng chuỗi luỹ thừa, và từ đó tìm được công thức cho h_n.

Trước hết ta đưa ra một số phép toán đối với hàm sinh. Giả sử

$$f(x) = \sum_{i=0}^{\infty} a_i x^i \ , \ g(x) = \sum_{i=0}^{\infty} b_i x^i$$

là hai hàm sinh còn α là số thực, khi đó

$$f(x) + g(x) = \sum_{i=0}^{\infty} (a_i + b_i) x^i ,$$

$$\alpha f(x) = \sum_{i=0}^{\infty} \alpha\, a_i x^i ,$$

Tích Côsi của hai hàm sinh $g(x)$ và $f(x)$:

$$f(x)g(x) = \sum_{i=0}^{\infty} c\, x^i ,$$

trong đó

$$c_k = a_0 b_k + a_1 b_{k-1} + ... + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i} x^i .$$

Từ giải tích ta biết rằng nếu chuỗi $g(x) = \sum_{i=0}^{\infty} h_i x^i$ hội tụ ở lân cận điểm 0 thì tổng của nó $g(x)$ luôn là hàm giải tích trong lân cận này và

$$h_k = g^{(k)}(0)/k! \ , \ k = 0, 1, ...$$

Khi đó chuỗi $\sum_{i=0}^{\infty} h_i x^i$ chính là khai triển Macloren của hàm $g(x)$. Như vậy có một tương ứng 1-1 giữa một hàm giải tích và một chuỗi hội tụ trong lân cận 0.

Trong việc áp dụng hàm sinh ta thường sử dụng công thức sau:

$$1/(1 - rx)^n = \sum_{k=0}^{\infty} C_{n+k-1}^{k} \ r^k x^k \ ,$$

mà trường hợp riêng của nó là

$$1/(1 - rx) = 1 + rx + r^2 x^2 + r^3 x^3 + ...$$

Thí dụ 1. Giải công thức đệ qui

$$h_n = 4 h_{n-2},$$

$$h_0 = 0, h_1 = 1.$$

Giải. Gọi $g(x) = h_0 + h_1 x + h_2 x^2 + ...$ là hàm sinh của dãy số cần tìm. Ta có

$$g(x) = h_0 + h_1 x + h_2 x^2 + ... + h_n x^n + ...$$

$$-4 x^2 g(x) = -4h_0 x^2 - 4h_1 x^3 - ... - 4h_{n-2} x^n - ...$$

Do $h_n = 4 h_{n-2}$ và $h_0 = 0, h_1 = 1$, nên cộng hai đẳng thức trên ta thu được

$$g(x) - 4 x^2 g(x) = h_0 + h_1 x = x,$$

hay

$$g(x) = x/(1 - 4x^2) = 1/(1 - 2x)(1 + 2x).$$

Sử dụng phép tách phân thức, ta viết $g(x)$ dưới dạng

$$g(x) = a/(1 - 2x) + b/(1 + 2x),$$

trong đó a và b là các hằng số cần xác định. Dễ dàng tính được $a = 1/4$ và $b = -1/4$. Từ đó ta có

$$g(x) = \frac{1}{4}[1/(1 - 2x) - 1/(1 + 2x)]$$

$$= \frac{1}{4}\sum_{k=0}^{\infty}(2^k - (-2)^k)x^k \ .$$

Vì vậy

$$h_k = \frac{1}{4}[2^k - (-2)^k],\ k = 0, 1, \ldots.$$

Thí dụ 2. Dãy số Fibonaci (Leonardo di Pisa hay Fibonaci (quãng 1170 - 1226) là nhà toán học Ý). Dãy số Fibonaci là dãy số được xác định bởi công thức đệ qui

$$f_n = f_{n-1} + f_{n-2},\ n \geq 2,$$

$$f_0 = 1, f_1 = 1.$$

Ta sẽ tìm công thức cho số hạng tổng quát của dãy số nhờ phương pháp hàm sinh.
Xét hàm sinh

$$F(x) = \sum_{n=0}^{\infty} f_n x^n \ .$$

Ta có

$$F(x) = \sum_{n=0}^{\infty} f_n x^n = f_0 + f_1 x + \sum_{n=2}^{\infty} f_n x^n$$

$$= f_0 + f_1 x + \sum_{n=2}^{\infty}(f_{n-1} + f_{n-2})x^n$$

$$= f_0 + f_1 x + \sum_{n=1}^{\infty} f_n x^{n+1} + \sum_{n=0}^{\infty} f_n x^{n+2}$$

$$= f_0 + f_1 x + x(F(x) - 1) + x^2 F(x).$$

Vậy

$$F(x) = f_0 + f_1 x + x(F(x) - 1) + x^2 F(x),$$

suy ra

$$F(x) = \frac{1}{1 - x - x^2}.$$

Ta có $(1 - x - x^2) = (1 - \alpha x)(1 + \beta x)$, với $\alpha = \frac{1-\sqrt{5}}{2}$, $\beta = \frac{1+\sqrt{5}}{2}$.

Viết lại $F(x)$ dưới dạng

$$F(x) = \frac{A}{1 - \alpha x} + \frac{B}{1 - \beta x},$$

ta tìm được $A = \alpha/(\alpha - \beta)$, $B = -\beta/(\alpha - \beta)$. Do đó

$$F(x) = \frac{1}{\alpha - \beta}\left[\frac{\alpha}{1-\alpha x} - \frac{\beta}{1-\beta x}\right] = \sum_{k=0}^{\infty} \frac{\alpha^{k+1} - \beta^{k+1}}{\alpha - \beta} x^k .$$

Từ đó

$$f_k = \frac{\alpha^{k+1} - \beta^{k+1}}{\alpha - \beta} = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{k+1}\right].$$

Thí dụ 3. Số Catalan (Charles Catalan (1814-1894) là nhà toán học Bỉ)
Số Catalan là một con số quan trọng trong tổ hợp, là lời giải của nhiều bài toán đếm tổ hợp quan trọng. Ta dẫn ra ở đây một trong những bài toán như vậy. Xét việc tính tích của các ma trận:

$$A = A_0 A_1 \ldots A_n.$$

Do tích ma trận có tính chất kết hợp nên có nhiều cách để thực hiện việc tính tích trên. Ví dụ khi $n = 3$, ta có thể thực hiện việc tính tích

$$A = A_0 A_1 A_2 A_3$$

theo 5 cách sau

$$A = A_0(A_1(A_2A_3)) = A_0((A_1A_2)A_3)$$

$$= (A_0A_1)(A_2A_3) = (A_0(A_1A_2))A_3 = ((A_0A_1)A_2)A_3.$$

Gọi c_n là số cách thực hiện việc tính A. Dễ thấy

$$c_0 = 1,\ c_1 = 1,$$

Nếu ta đặt dấu ngoặc phân tách đầu tiên vào sau thừa số A_k:

$$A = (A_0A_1 \ldots A_k)(A_{k+1}A_{k+2} \ldots A_n),$$

thì do có c_k cách thực hiện việc tính $A_0A_1 \ldots A_k$ và c_{n-k-1} cách thực hiện việc tính $A_{k+1}A_{k+2} \ldots A_n$, suy ra có $c_k\, c_{n-k-1}$ cách tính A trong trường hợp này. Do dấu ngoặc phân tách đầu tiên có thể đặt sau A_i, $i = 0, 1, \ldots, n-1$, nên ta thu được

$$c_n = \sum_{k=0}^{n-1} c_k c_{n-k-1} .$$

Xét hàm sinh của dãy số $\{c_n\}$: $\mathcal{C}(x) = \sum_{i=0}^{\infty} c_i x^i$. Ta có

$$\mathbf{C}^2(x) = \sum_{m=0}^{\infty} c_m x^m \sum_{n=0}^{\infty} c_n x^n = \sum_{r=0}^{\infty} \left(\sum_{m=0}^{r} c_m c_{r-m} \right) x^r = \sum_{r=0}^{\infty} c_{r+1} x^r$$

Vì thế

$$\mathbf{C}(x) = x\,\mathbf{C}^2(x) + 1.$$

Giải phương trình này theo $\mathbf{C}(x)$ ta thu được

$$\mathbf{C}(x) = \frac{1 \pm \sqrt{1-4x}}{x}.$$

Ta phân tích $f(x) = \sqrt{1-4x}$ thành chuỗi dựa vào công thức Taylor

$$f(x) = f(0) + \sum_{k=1}^{\infty} \frac{f^{(k)}(0)}{k!} x^k .$$

Ta có

$$\frac{d^k}{dx^k}(1-4x)^{\frac{1}{2}} = \frac{1}{2}\left(\frac{1}{2}-1\right)\ldots\left(\frac{1}{2}-k+1\right)(1-4x)^{\frac{1}{2}-k}(-4)^k$$

$$= -2^k.1.3\ldots(2k-3)(1-4x)^{\frac{1}{2}-k} = -2(k-1)!C_{2k-2}^{k-1}(1-4x)^{\frac{1}{2}-k}$$

vì thế

$$f(x) = 1 - 2\sum_{k=1}^{\infty} \frac{1}{k} C_{2k-2}^{k-1} x^k .$$

Thay vào công thức tính $\mathbf{C}(x)$, trong đó rõ ràng cần chọn nghiệm $\frac{1-\sqrt{1-4x}}{x}$ để phù hợp với điều kiện $c_n \geq 0$, ta thu được

$$\mathbf{C}(x) = \sum_{k=1}^{\infty} \frac{1}{k} C_{2k-2}^{k-1} x^k = \sum_{k=0}^{\infty} \frac{1}{k+1} C_{2k}^{k} x^k .$$

Từ đó ta tìm được

$$c_k = \frac{1}{k+1} C_{2k}^{k}$$

Để ý đến công thức Stirling tính gần đúng $n!$

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

ta có thể tính gần đúng c_k theo công thức:

$$c_k \approx \frac{4^k}{k^2}.$$

2.6. Liệt kê

Việc tìm một công thức cho kết quả đếm, ngay cả trong trường hợp công thức truy hồi, không phải là dễ thực hiện. Cho đến nay, còn rất nhiều bài toán đếm chưa có lời giải dưới dạng một công thức. Đối với những bài toán như vậy, người ta chỉ còn cách chỉ ra một phương pháp liệt kê, theo đó có thể đi qua tất cả các cấu hình cần đếm. Rõ ràng bản thân phương pháp liệt kê không chỉ ra được một kết quả cụ thể nào, nhưng qua nó, người ta có thể lập trình cho máy tính điện tử để nhờ máy tính "đếm" hộ.

Để thí dụ, ta xét một cấu hình tổ hợp nổi tiếng do hàng loạt những công trình xung quanh nó và cho đến nay còn nhiều vấn đề cần giải quyết, đó là các hình chữ nhật la tinh.

Giả sử S là một n-tập. Không mất tính tổng quát, ta có thể giả thiết S là tập $\{1, 2, ..., n\}$. *Một hình chữ nhật la tinh* trên S là một bảng p dòng, q cột, sao cho mỗi dòng của nó là một chỉnh hợp không lặp chập q của S và mỗi cột của nó là một chỉnh hợp không lặp chập p của S.

Theo định nghĩa, ta có $p \le n$, $q \le n$. Đặc biệt, trong trường hợp $q = n$, mỗi dòng của hình chữ nhật la tinh là một hoán vị của S, sao cho không có một cột nào được chứa phần tử lặp lại. Hình chữ nhật la tinh dạng này được gọi là *chuẩn* nếu dòng đầu của nó là hoán vị 1, 2, ..., n.

Thí dụ

1	2	3	4	5	6	7
2	3	4	5	6	7	1
3	4	5	6	7	1	2

là một hình chữ nhật la tinh chuẩn trên tập $S = \{1, 2, 3, 4, 5, 6, 7\}$.

Gọi $L(p, n)$ là số hình chữ nhật la tinh $p \times n$, còn $K(p, n)$ là số hình chữ nhật la tinh chuẩn $p \times n$. Ta có

$$L(p, n) = n!\, K(p, n)$$

Dễ dàng thấy rằng, số mất thứ tự D_n là số các hình chữ nhật la tinh chuẩn $2 \times n$, còn số phân bố U_n là số các hình chữ nhật la tinh chuẩn $3 \times n$, với 2 dòng đầu là

1	2	...	n-1	n
2	3	...	n	1

Riordan J. (1946) đã chứng minh công thức

$$K(3, n) = \sum_{k=0}^{m} C_n^k . D_{n-k} . D_k . U_{n-2k}$$

trong đó $m = [n/2]$, $U_0 = 1$.

Bài toán đếm số hình chữ nhật la tinh với số dòng nhiều hơn cho đến nay chưa được giải quyết. Người ta mới chỉ đưa ra được một vài dạng tiệm cận của $L(p, n)$ (Erdos P. (1946), Yamamoto K. (1951)).

Nếu $p = q = n$, thì hình chữ nhật la tinh được gọi là hình vuông la tinh. Một hình vuông la tinh cấp n được gọi là chuẩn nếu có dòng đầu và cột đầu là hoán vị 1 2 ... n. Thí dụ một hình vuông la tinh chuẩn cấp 7

1	2	3	4	5	6	7
2	3	4	5	6	7	1
3	4	5	6	7	1	2
4	5	6	7	1	2	3
5	6	7	1	2	3	4
6	7	1	2	3	4	5
7	1	2	3	4	5	6

Gọi l_n là số hình vuông la tinh như thế, ta có

$$L(n, n) = n! (n-1)! l_n .$$

Việc tìm một công thức cho l_n đến nay còn để ngỏ. Những công thức tính $K(p, n)$ cho thấy rằng điều này không phải dễ. Tuy nhiên ta có thể lập một chương trình cho máy tính, liệt kê tất cả các hình vuông la tinh chuẩn cấp n. Dưới đây là một vài giá trị tính được:

n	1	2	3	4	5	6	7
l_n	1	1	1	4	56	9408	16942080

Bài tập

Nguyên lý cộng và Nguyên lý nhân

1. Cho 5 ký tự A, B, C, D, E.

(a) Có bao nhiêu xâu ký tự độ dài 4 có thể lập được từ các ký tư đã cho, nếu không cho phép lặp lại ký tự?

(b) Có bao nhiêu xâu ký tự trong (a) bắt đầu từ B?

(c) Có bao nhiêu xâu ký tự trong (a) không bắt đầu từ B?

2. Cho X là tập n phần tử. Có bao nhiêu bộ có thứ tự (A, B) thoả mãn $A \subseteq B \subseteq X$?

3. Đoàn chủ tịch của một cuộc họp gồm 6 người A, B, C, D, E, F cần bầu ra Ban lãnh đạo gồm 1 chủ tịch, 1 phó chủ tịch và 1 thư ký.

(a) Hỏi có bao nhiêu cách khác nhau?

(b) Có bao nhiêu cách mà trong đó một trong hai người A, B là chủ tịch?

(c) Có bao nhiêu cách mà trong đó E là thành viên của Ban lãnh đạo?

(d) Có bao nhiêu cách mà trong đó D và F là thành viên của Ban lãnh đạo?

4. Có bao nhiêu xâu nhị phân độ dài 10 bắt đầu bởi hoặc là 101 hoặc là 111?

5. Có 10 cuốn sách khác nhau, trong đó có 5 cuốn sách thuộc lĩnh vực Tin học, 3 cuốn sách thuộc lĩnh vực toán học và 2 cuốn sách về lĩnh vực nghệ thuật. Hỏi có bao nhiêu cách chọn ra 2 cuốn sách có nội dung thuộc các lĩnh vực khác nhau từ 10 cuốn sách nói trên?

6. Có 10 cuốn sách khác nhau, trong đó có 5 cuốn sách thuộc lĩnh vực Tin học, 3 cuốn sách thuộc lĩnh vực toán học và 2 cuốn sách về lĩnh vực nghệ thuật.

(a) Hỏi có bao nhiêu cách xếp 10 cuốn sách này lên 1 giá sách?

(b) Hỏi có bao nhiêu cách xếp 10 cuốn sách này lên 1 giá sách sao cho tất cả các cuốn sách Tin học được xếp ở phía trái giá sách còn hai cuốn sách về nghệ thuật được xếp bên phải?

(c) Hỏi có bao nhiêu cách xếp 10 cuốn sách này lên 1 giá sách sao cho tất cả các cuốn sách thuộc cùng lĩnh vực được xếp cạnh nhau?

(d) Hỏi có bao nhiêu cách xếp 10 cuốn sách này lên 1 giá sách sao cho hai cuốn sách nghệ thuật không được xếp cạnh nhau?

7. Có bao nhiêu số có bốn chữ số có thể tạo thành từ các chữ số 0, 1, 2, 3 , 4, 5 thoả mãn

(a) không có chữ số nào được lặp lại,

(b) các chữ số được lặp lại,

(c) các số chẵn trong (b).

8. Trên cạnh bên của một tam giác ta lấy n điểm, trên cạnh bên thứ hai lấy m điểm. Mỗi một trong hai đỉnh của cạnh đáy được nối với các điểm được chọn trên cạnh bên đối diện bởi các đường thẳng. Hỏi

(a) Có bao nhiêu giao điểm của các đường thẳng nằm trong đa giác?

(b) Các đường thẳng chia tam giác ra làm bao nhiêu phần?

9. Một cán bộ tin học do đãng trí nên đã quên mật khẩu của phần mềm máy tính của mình. May mắn là anh ta còn nhớ mật khẩu có dạng NNN-XX, trong đó NNN là các chữ số, còn XX là các chữ cái lấy trong bảng chữ cái có 26 chữ. Hỏi trong trường hợp xấu nhất cần phải thử bao nhiêu mật khẩu để có thể tìm lại mật khẩu đã đặt?

10. Hỏi có bao nhiêu bộ có thứ tự gồm 3 tập X_1, X_2, X_3 thoả mãn

$$X_1 \cup X_2 \cup X_3 = \{ 1, 2, 3, 4, 5, 6, 7, 8 \} \text{ và } X_1 \cap X_2 \cap X_3 = \varnothing.$$

Ví dụ: Hai bộ

$$X_1 = \{1, 2, 3\},\ X_2 = \{1, 4, 8\},\ X_3 = \{2, 5, 6, 7\}$$

và

$$X_1 = \{1, 4, 8\},\ X_2 = \{1, 2, 3\},\ X_3 = \{2, 5, 6, 7\}$$

được coi là khác nhau.

Chỉnh hợp, Hoán vị, Tổ hợp

11. Có bao nhiêu hoán vị của các chữ cái trong xâu ABCDEF mà trong đó có chứa xâu con DEF?

12. Có bao nhiêu hoán vị của các chữ cái trong xâu ABCDEF mà trong đó có chứa ba chữ cái D, E, F đứng cạnh nhau?

13. Có bao nhiêu cách xếp 6 người vào ngồi quanh cái bàn tròn (hai cách xếp không coi là khác nhau nếu chúng có thể thu được từ nhau bởi phép quay bàn tròn)?

14. Có bao nhiêu cách xếp 7 học sinh nam và 5 học sinh nữ ra thành một hàng ngang sao cho không có hai nữ sinh nào đứng cạnh nhau?

15. Có bao nhiêu xâu nhị phân độ dài 32 mà trong đó có đúng 6 số 1?

16. Có bao nhiêu xâu ký tự có thể tạo được từ các chữ cái

MISSISSIPPI

17. Có 8 cuốn sách khác nhau. Hỏi có bao nhiêu cách phân các cuốn sách này cho 3 học sinh: Mơ, Mai, Mận sao cho Mơ nhận được 4 cuốn còn Mai và Mận mỗi người nhận hai cuốn?

18. Giả sử X là tập t phần tử. Ta gọi *tổ hợp lặp* chập k từ t phần tử của X là một bộ không có thứ tự gồm k thành phần lấy từ các phần tử của X.
Ví dụ: $X=\{a, b, c\}$, các tổ hợp lặp chập 2 từ các phần tử của X là

$$(a\ a), (a\ b), (a\ c), (b\ b), (b\ c), (c\ c).$$

Chứng minh rằng số tổ hợp lặp chập k từ t là:

$$C(k+t-1, t-1) = C(k+t-1, k).$$

19. Có 3 rổ đựng các quả cầu xanh, đỏ, tím. Mỗi giỏ chỉ chứa các quả cầu cùng mầu và mỗi giỏ chứa ít ra là 8 quả cầu.

(a) Có bao nhiêu cách chọn ra 8 quả cầu?

(b) Có bao nhiêu cách chọn ra 8 quả cầu mà trong đó có ít nhất một quả cầu đỏ, một quả cầu xanh, 1 quả cầu tím?

20. Xét phương trình:

$$x_1 + x_2 + x_3 + x_4 = 29.$$

(a) Hỏi phương trình đã cho có bao nhiêu nghiệm nguyên dương?

(b) Hỏi phương trình đã cho có bao nhiêu nghiệm nguyên không âm?

Nguyên lý bù trừ

1. Hỏi trong đoạn từ 1 đến 1000 có bao nhiêu số hoặc là số lẻ hoặc là số chính phương

2. Có bao nhiêu xâu nhị phân độ dài 8 không chứa 6 số 0 liền nhau?

3. Có bao nhiêu số có 10 chữ số với các chữ số chỉ là 1, 2, 3 mà trong đó mỗi chữ số 1, 2, 3 có mặt ít nhất 1 lần?

4. Có bao nhiêu xâu nhị phân độ dài 10 hoặc là bắt đầu bởi 3 số 1, hoặc là kết thúc bởi 4 số 0?

5. Có bao nhiêu số nguyên dương nhỏ hơn 10000 chia hết cho 7 nhưng không chia hết cho 5 và 2?

6. Có bao nhiêu hoán vị của các số tự nhiên 1,2,...,10 mà trong đó 3 số 1, 2, 3 không đứng cạnh nhau theo thứ tự tăng dần?

7. Hỏi phương trình

$$x_1 + x_2 + x_3 + x_4 = 29$$

có bao nhiêu nghiệm nguyên không âm thoả mãn

$$x_1 \le 3,\ x_2 \le 12,\ x_3 \le 5, x_4 \le 10\ .$$

8. Một lớp gồm 50 học sinh làm bài kiểm tra gồm 3 câu hỏi. Biết rằng mỗi học sinh làm được ít nhất 1 câu và số học sinh làm được câu 1 là 40, câu 2 là 35, câu 3 là 30. Chứng minh rằng số học sinh làm được cả 3 câu không vượt quá 27.

Hệ thức truy hồi

1. Giải các hệ thức truy hồi sau

(a) $a_n = 2\,a_{n-1}$, $n \ge 1$,
$a_0 = 3$.

(b) $a_n = 5\,a_{n-1} - 6\,a_{n-2}$, $n \ge 2$,
$a_0 = 1$, $a_1 = 0$.

(c) $a_n = 4\,a_{n-1} - 4\,a_{n-2}$, $n \ge 2$,
$a_0 = 6$, $a_1 = 8$.

(d) $a_n = 4\,a_{n-2}$, $n \ge 2$,
$a_0 = 0$, $a_1 = 4$.

(e) $a_n = a_{n-2}/4$, $n \ge 2$,
$a_0 = 1$, $a_1 = 0$.

2. Lập công thức truy hồi cho S_n là số cách chia một hình chữ nhật kích thước $2 \times n$ ra thành các hình chữ nhật con có cạnh song song với cạnh của hình chữ nhật đã cho và với kích thước là 1×2, 2×1, 2×2. Giải hệ thức thu được.

3. Lập công thức truy hồi để đếm F_n là số xâu nhị phân độ dài n không chứa ba số 0 liên tiếp. Từ đó tính F_{10}.

4. Lập công thức truy hồi để đếm Q_n là số chỉnh hợp lặp chập n từ ba chữ số 0, 1, 2 không chứa hoặc là hai số 0 liên tiếp hoặc là hai số 1 liên tiếp. Từ đó tính Q_6. Giải hệ thức thu được.

5. Xét ma trận vuông

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

a) Chứng minh rằng

$$A^n = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix},$$

trong đó F_n là số hạng thứ n của dãy số Fibonaci.

b) Tính $\det(A^n)$. Từ đó suy ra công thức: $F_{n-1}F_n - (F_n)^2 = (-1)^n$.

Hàm sinh

1. Viết công thức dưới dạng giải tích cho hàm sinh của các dãy số sau

a) $a_n = 3^n$, $n = 0, 1, 2, ...$

b) $\{a_0, a_1, a_2, a_3, ...\} = \{0, 1, 0, 1, ...\}$

2. Tìm công thức cho số hạng tổng quát a_n của dãy số $\{a_n\}$ có hàm sinh là

a) $G(x) = 1/(1 - 2x)$;

b) $G(x) = 1/(1 - x)^2$;

c) $G(x) = 1/(1 + x - 2x^2)$.

3. Sử dụng hàm sinh để tìm công thức dưới dạng hiện cho dãy số cho bởi công thức đệ qui sau đây:

a) $a_{n+1} = a_n + 2$, $a_0 = -3$;

b) $2a_{n+1} = a_n + a_{n-1}$, $a_0 = 0$, $a_1 = 1$;

c) $a_{n+2} = 3a_{n+1} - 2a_n + 2(3)^n$, $a_0 = 1$; $a_1 = 2$.

3

BÀI TOÁN TỒN TẠI

3.1. Giới thiệu bài toán

Trong mục trước, ta đã tập trung chú ý vào việc đếm số các cấu hình tổ hợp (số phần tử của các đối tượng tổ hợp) thoả mãn những tính chất nào đó, chẳng hạn đếm số tổ hợp, chỉnh hợp, hoán vị, ... Trong những bài toán đó sự tồn tại của các cấu hình là hiển nhiên và công việc chính là đếm số phần tử thoả mãn tính chất đặt ra. Tuy nhiên, trong rất nhiều bài toán tổ hợp, việc chỉ ra sự tồn tại của một cấu hình thoả mãn các tính chất cho trước là hết sức khó khăn. Chẳng hạn, khi một kỳ thủ cần phải tính toán các nước đi của mình để giải đáp xem liệu có khả năng thắng hay không, hoặc là một người giải mật mã cần tìm kiếm chìa khoá giải cho một bức mật mã mà anh ta không biết liệu đây có đúng là bức điện thật được mã hoá của đối phương hay không, hay chỉ là bức mật mã giả của đối phương tung ra nhằm đảm bảo an toàn cho bức điện thật ... Như vậy, trong tổ hợp xuất hiện một vấn đề thứ hai rất quan trọng là: xét sự tồn tại của các cấu hình tổ hợp với các tính chất cho trước. Các bài toán thuộc dạng này được gọi là các *bài toán tồn tại tổ hợp*.

Một bài toán tồn tại tổ hợp xem như giải xong nếu hoặc chỉ ra một cách xây dựng cấu hình, hoặc chứng minh rằng chúng không có. Tuy nhiên cả hai khả năng đều không phải dễ. Để thấy rõ sự phức tạp của vấn đề, dưới đây ta sẽ xét một số bài toán tồn tại tổ hợp cổ điển nổi tiếng.

3.1.1. Bài toán về 36 sĩ quan

Bài toán này được Euler đề nghị, nội dung của nó như sau: có một lần người ta triệu tập từ 6 trung đoàn mỗi trung đoàn 6 sĩ quan thuộc 6 cấp bậc khác nhau: thiếu úy, trung uý, thượng uý, đại uý, thiếu tá, trung tá về tham gia duyệt binh ở sư đoàn bộ. Hỏi rằng có thể xếp 36 sĩ quan này thành một đội ngũ hình vuông sao cho trong mỗi một hàng ngang cũng như mỗi một hàng dọc đều có đại diện của cả 6 trung đoàn và của cả 6 cấp bậc.

Để đơn giản ta sẽ dùng các chữ cái in hoa A, B, C, D, E, F để chỉ các phiên hiệu trung đoàn còn các chữ cái thường a, b, c, d, e, f để chỉ các cấp bậc. Bài toán này có thể tổng quát hoá nếu thay con số 6 bởi n. Trong trường hợp $n = 4$, một lời giải của bài toán 16 sỹ quan là

Ab Dd Ba Cc
Bc Ca Ad Db
Cd Bb Dc Aa
Da Ac Cb Bd

Một lời giải trong trường hợp $n = 5$ là

Aa Bb Cc Dd Ee
Cd De Ea Ab Bc
Eb Ac Bd Ce Da
Be Ca Db Ec Ad
Dc Ed Ae Ba Cb

Do lời giải của bài toán có thể biểu diễn bởi 2 hình vuông với các chữ cái la tinh hoa và thường chồng cạnh nhau nên bài toán tổng quát đặt ra còn được biết dưới tên gọi bài toán về *hình vuông la tinh trực giao*. Trong hai thí dụ trên ta có hình vuông la tinh trực giao cấp 4 và 5.

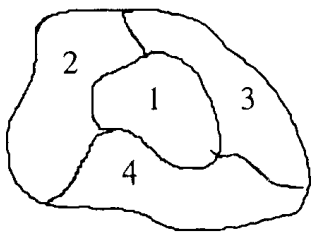
Euler đã mất rất nhiều công sức để tìm lời giải cho bài toán 36 sĩ quan thế nhưng ông đã không thành công. Vì vậy ông đã đề ra giả thuyết là cách xếp như vậy không tồn tại. Giả thuyết này được nhà toán học Pháp Tarri chứng minh năm 1901 bằng cách duyệt tất cả mọi khả năng xếp. Euler căn cứ vào sự không tồn tại lời giải khi n=2 và n=6 còn đề ra một giả thuyết tổng quát hơn là: không tồn tại hình vuông la tinh trực giao cấp $n = 4k + 2$. Giả thuyết này đã tồn tại suốt hai thế kỷ, mãi đến năm 1960 ba nhà toán học Mỹ là Boce, Parker, Srikanda mới chỉ ra được một lời giải với $n = 10$ và

sau đó chỉ ra phương pháp xây dựng hình vuông la tinh trực giao cho mọi $n = 4k + 2$, với $k > 1$.

Tưởng chừng bài toán đặt ra chỉ có ý nghĩa thuần tuý của một bài toán đố hóc búa thử trí tuệ con người. Thế nhưng gần đây người ta đã phát hiện những ứng dụng quan trọng của vấn đề trên vào quy hoạch thực nghiệm, sắp xếp các lịch thi đấu trong các giải cờ quốc tế, hình học xạ ảnh, ...

3.1.2. Bài toán 4 mầu

Có những bài toán mà nội dung của nó có thể giải thích cho bất kỳ ai, tuy nhiên lời giải của nó thì ai cũng có thể thử tìm, nhưng mà khó có thể tìm được. Ngoài định lý Fermat thì bài toán 4 màu là một bài toán như vậy. Bài toán có thể phát biểu trực quan như sau: chứng minh rằng mọi bản đồ trên mặt phẳng đều có thể tô bằng 4 màu sao cho không có hai nước láng giềng nào lại bị tô bởi cùng một màu. Chú ý rằng, ta xem như mỗi nước là một vùng liên thông và hai nước được gọi là láng giềng nếu chúng có chung biên giới là một đường liên tục.



Hình 1. Bản đồ không tô được bởi ít hơn 4 mầu

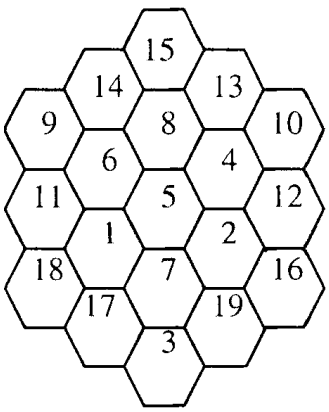
Con số 4 không phải là ngẫu nhiên. Người ta đã chứng minh được rằng mọi bản đồ đều được tô với số mầu lớn hơn 4, còn với số mầu ít hơn 4 thì không tô được, chẳng hạn bản đồ gồm 4 nước trên hình 1 không thể tô được với số mầu ít hơn 4.

Bài toán này xuất hiện vào khoảng những năm 1850-1852 từ một nhà buôn người Anh là Gazri khi tô bản đồ hành chính nước Anh đã cố gắng chứng minh rằng nó có thể tô bằng 4 màu. Sau đó, năm 1852 ông ta đã viết thư cho De Morgan để thông báo về giả thuyết này. Năm 1878, Keli trong một bài báo đăng ở Tuyển tập các công trình của Hội toán học Anh có hỏi rằng bài toán này đã được giải quyết hay chưa? Từ đó bài toán trở thành nổi tiếng, và trong suốt hơn một thế kỷ qua đã có rất nhiều người làm toán, nghiệp dư cũng như chuyên nghiệp, đã cố gắng chứng minh giả thuyết này. Tuy nhiên

mãi đến năm 1976 hai nhà toán học Mỹ là K.Appel và W.Haken mới chứng minh được giả thuyết này bằng máy tính điện tử. Tất nhiên một chứng minh với sự giúp đỡ của máy tính điện tử không thực sự thoả mãn được nhu cầu của công chúng muốn kiểm tra tính đúng đắn của cách chứng minh. Vì vậy, chính hai tác giả trên vào cuối những năm 1990 đã cho công bố một cuốn sách trình bày về phương pháp chứng minh của mình (cuốn sách dày trên 800 trang). Cũng vào những năm cuối của thế kỷ 20, một nhóm các nhà toán học Mỹ đã đưa ra một chứng minh *có thể kiểm tra bằng tay*! Rất tiếc là chứng minh này cũng không phải là đơn giản. Cho đến nay các nhà toán học vẫn đang nỗ lực nghiên cứu để tìm ra một cách chứng minh dễ hiểu như bản thân nội dung của bài toán.

3.1.3. Hình lục giác thần bí

Năm 1910 Clifford Adams đề ra bài toán hình lục giác thần bí sau: trên 19 ô lục giác (xem hình vẽ ở dưới) hãy điền vào các số từ 1 đến 19 sao cho tổng theo 6 hướng của lục giác là bằng nhau (và đều bằng 38).



Hình 2. Hình lục giác thần bí

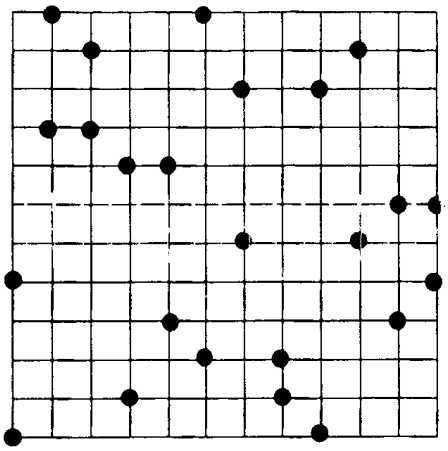
Sau 47 năm trời kiên nhẫn cuối cùng ông ta đã tìm được lời giải. Sau đó vì sơ ý đánh mất bản thảo ông ta đã tốn thêm 5 năm để khôi phục lại. Năm 1962 Adams đã công bố lời giải đó (xem hình 2).

Thật không thể ngờ là đó là lời giải duy nhất (nếu không tính đến các lời giải sai khác nhau bởi phép biến hình đơn giản).

3.1.4. Bài toán chọn $2n$ điểm trên lưới $n \times n$ điểm

Cho một lưới ô vuông gồm $n \times n$ điểm. Hỏi có thể chọn trong số chúng $2n$ điểm, sao cho không có 3 điểm được chọn nào là thẳng hàng hay không? Hiện nay người ta mới biết được rất ít về lời giải của bài toán trong những tình huống không tầm thường. Câu hỏi

về sự tồn tại của lời giải của bài toán với những giá trị lớn của n vẫn còn để ngỏ. Hình 3 cho một lời giải với $n = 12$:



Hình 3. Một lời giải bài toán của bài toán với $n = 12$

3.2. Phương pháp phản chứng

Một trong những cách giải bài toán tồn tại là dùng lập luận phản chứng: giả thiết điều định chứng minh là sai, từ đó dẫn đến mâu thuẫn.

Thí dụ 1. Cho 7 đoạn thẳng có độ dài lớn hơn 10 và nhỏ hơn 100. Chứng minh rằng luôn tìm được 3 đoạn để có thể ghép thành một tam giác.

Giải: Chú ý rằng, cần và đủ để 3 đoạn có thể ghép thành một tam giác là tổng độ dài của 2 đoạn nhỏ phải lớn hơn độ dài của đoạn lớn, ta sắp các đoạn đã cho theo thứ tự tăng dần của độ dài $a_1, a_2, ..., a_7$ và chứng minh rằng trong dãy đã xếp luôn tìm được 3 đoạn liên tiếp sao cho tổng của 2 đoạn đầu lớn hơn đoạn cuối. Giả thiết điều này không xảy ra, nghĩa là đồng thời xảy ra các bất đẳng thức:

$$a_1 + a_2 \leq a_3,$$
$$a_2 + a_3 \leq a_4,$$
$$a_3 + a_4 \leq a_5,$$
$$a_4 + a_5 \leq a_6,$$
$$a_5 + a_6 \leq a_7.$$

Từ giả thiết a_1, a_2 có giá trị lớn hơn 10, ta nhận được $a_3 > 20$. Từ $a_2 > 10$ và $a_3 > 20$, ta nhận được $a_4 > 30$, ..., cứ như vậy ta nhận được $a_5 > 50$, $a_6 > 80$ và $a_7 > 130$. Bất đẳng thức cuối cùng mâu thuẫn với giả thiết các độ dài nhỏ hơn 100 và điều đó chứng minh kết luận của bài toán.

Thí dụ 2. Các đỉnh của một thập giác đều được đánh số bởi các số nguyên 0, 1, ... , 9 một cách tuỳ ý. Chứng minh rằng luôn tìm được ba đỉnh liên tiếp có tổng các số là lớn hơn 13.

Giải: Gọi $x_1, x_2, \ldots, x_{10}$ là các số gán cho các đỉnh của 1, 2, . . ., 10 của thập giác. Giả sử ngược lại là không tìm được ba đỉnh nào thoả mãn khẳng định của thí dụ. Khi đó ta có

$$k_1 = x_1 + x_2 + x_3 \leq 13,$$
$$k_2 = x_2 + x_3 + x_4 \leq 13,$$
$$\ldots\ldots$$
$$k_9 = x_9 + x_{10} + x_1 \leq 13,$$
$$k_{10} = x_{10} + x_1 + x_2 \leq 13,$$

Từ đó suy ra

$$k_1 + k_2 + \ldots + k_{10} \leq 130.$$

Mặt khác do

$$\begin{aligned} k_1 + k_2 + \ldots + k_{10} &= 3(x_1 + x_2 + \ldots + x_{10}) \\ &= 3\,(0 + 1 + 2 + \ldots + 9) \\ &= 135, \end{aligned}$$

suy ra

$$135 = k_1 + k_2 + \ldots + k_{10} \leq 130.$$

Mâu thuẫn thu được đã chứng tỏ khẳng định trong ví dụ là đúng.

Thí dụ 3. Chứng minh rằng không thể nối 31 máy vi tính thành một mạng sao cho mỗi máy được nối với đúng 5 máy khác.

Giải: Giả sử ngược lại là tìm được cách nối 31 máy sao cho mỗi máy được nối với đúng 5 máy khác. Khi đó số lượng kênh nối là 5×31/2 = 75,5 ?! Mâu thuẫn thu được đã chứng minh khẳng định trong thí dụ là đúng.

3.3. Nguyên lý Dirichlet

Trong rất nhiều bài toán tổ hợp, để chứng minh sự tồn tại của một cấu hình với những tính chất cho trước, người ta sử dụng nguyên lý đơn giản sau, gọi là nguyên lý Dirichlet:

Nguyên lý Dirichlet. *Nếu đem xếp nhều hơn n đối tượng vào n cái hộp, thì luôn tìm được một cái hộp chứa không ít hơn 2 đối tượng.*

Chứng minh. Việc chứng minh nguyên lý trên chỉ là một lập luận phản chứng đơn giản. Giả sử ngược lại là không tìm được cái hộp nào chứa không ít hơn 2 đối tượng. Điều đó có nghĩa là mỗi cái hộp chứa không quá một đối tượng. Từ đó suy ra tổng số đối tượng xếp trong n cái hộp là không vượt quá n, trái với giả thiết là có nhiều hơn n đối tượng được xếp trong chúng.

Lập luận hoàn toàn tương tự, có thể chứng minh Nguyên lý Dirichlet tổng quát sau.

Nguyên lý Dirichlet tổng quát. *Nếu đem xếp n đối tượng vào k cái hộp, thì luôn tìm được một cái hộp chứa không ít hơn n/k đối tượng.*

Nguyên lý trên được nhà toán học nổi tiếng người Đức là Dirichlet đề xuất từ thế kỷ 19 và ông đã áp dụng nó để giải nhiều bài toán tồn tại tổ hợp. Các thí dụ dưới đây cho ta thấy nguyên lý được sử dụng như thế nào.

Thí dụ 1. Trong số 367 người bao giờ cũng tìm được hai người có ngày sinh nhật giống nhau bởi vì chỉ có tất cả 366 ngày sinh nhật khác nhau.

Thí dụ 2. Trong kỳ thi học sinh giỏi điểm bài thi được đánh giá bởi một số nguyên trong khoảng từ 0 đến 100. Hỏi rằng ít nhất phải có bao nhiêu học sinh dự thi để cho chắc chắn tìm được hai học sinh có kết quả thi như nhau?

Giải. Theo nguyên lý Dirichlet, số học sinh cần tìm là 102, vì ta có 101 kết quả điểm thi khác nhau.

Thí dụ 3. Trong số những người có mặt trên trái đất luôn tìm được hai người có hàm răng giống nhau, bởi vì chỉ có tất cả

$$2^{32} = 4\,294\,967\,296$$

hàm răng khác nhau mà số người trên hành tinh chúng ta hiện nay đã vượt quá 5 tỷ.

Thí dụ 4. Trong 100 người có ít nhất 9 người sinh cùng một tháng.

Giải: Xếp những người cùng sinh một tháng vào một nhóm. Có 12 tháng tất cả. Vậy theo nguyên lý Dirichlet, tồn tại ít nhất một nhóm có không ít hơn 100/12 = 8,3... nghĩa là 9 người.

Thí dụ 5. Có năm loại học bổng khác nhau. Hỏi rằng phải có ít nhất bao nhiêu sinh viên để chắc chắn rằng có ít ra là sáu người cùng nhận học bổng như nhau?

Giải: Số sinh viên ít nhất cần có để đảm bảo chắc chắn có 6 sinh viên cùng nhận học bổng như nhau là số nguyên nhỏ nhất n sao cho $n/5 > 5$. Số nguyên nhỏ nhất đó là $n = 5\times5+1 = 26$. Vậy 26 là số lượng sinh viên nhỏ nhất đảm bảo chắc chắn là có sáu sinh viên cùng hưởng một loại học bổng.

Thí dụ 6. Biển số xe máy phân khối lớn gồm 7 ký tự:

NN - NNN - XX,

trong đó hai ký tự đầu là mã số địa danh, ba ký tự tiếp theo là số hiệu xe, mỗi ký tự là một số từ 0 đến 9, hai ký tự cuối là mã đăng ký gồm hai chữ cái lấy trong bảng chữ cái la tinh gồm 26 chữ cái. Hỏi rằng, để có 2 triệu biển số xe máy khác nhau thì cần phải có ít nhất bao nhiêu mã địa danh khác nhau?

Giải: Với mỗi một mã địa danh ta có $10^3.26^2 = 676.10^3$ biển số xe máy khác nhau. Vì vậy để có 2 triệu biển số xe máy khác nhau, cần có ít nhất

$$2.10^6/(676.10^3),$$

nghĩa là 3 mã địa danh khác nhau.

Trong nhiều ứng dụng thú vị của nguyên lý Dirichlet, khái niệm đối tượng và cái hộp cần phải được lựa chọn một cách khôn khéo hơn. Tiếp theo, ta sẽ dẫn ra một vài thí dụ như vậy.

Thí dụ 7. Trong một phòng họp bao giờ cũng tìm được hai người có số người quen trong số những người dự họp là bằng nhau.

Giải: Gọi số người dự họp là n, khi đó số người quen của một người nào đó trong phòng họp chỉ có thể nhận các giá trị từ 0 đến n-1. Rõ ràng trong phòng không thể đồng thời có người có số người quen là 0 (tức là không quen ai cả) và có người có số người quen là n-1 (tức là quen tất cả). Vì vậy, theo số lượng người quen ta chỉ có thể phân n người ra thành n-1 nhóm. Theo nguyên lý Dirichlet suy ra có ít nhất một nhóm phải có không ít hơn hai người, tức là luôn tìm được ít ra là hai người có số người quen là bằng nhau.

Bài toán này có thể phát biểu dưới dạng ngôn ngữ hình học như sau: trên mặt phẳng cho n điểm, giữa chúng có một số điểm được nối với nhau bởi các đoạn thẳng. Khi đó bao giờ cũng tìm được hai điểm có cùng một số cạnh nối phát ra từ chúng.

Thí dụ 8. Trong một tháng gồm 30 ngày một đội bóng chuyền thi đấu mỗi ngày ít nhất một trận, nhưng không chơi quá 45 trận. Hãy chứng minh rằng phải tìm được một giai đoạn gồm một số ngày liên tục nào đó trong tháng sao cho trong giai đoạn đó đội chơi đúng 14 trận.

Giải: Giả sử a_j là tổng số trận thi đấu cho đến hết ngày thứ j của đội. Khi đó

$$a_1, a_2, ..., a_{30}$$

là dãy tăng các số nguyên dương và đồng thời $1 \le a_j \le 45$. Suy ra dãy

$$a_1+14, a_2+14, ..., a_{30}+14$$

cũng là dãy tăng các số nguyên dương và $15 \le a_j+14 \le 59$.

Tất cả có 60 số nguyên dương

$$a_1, a_2, ..., a_{30}, a_1+14, a_2+14, ..., a_{30}+14,$$

trong đó tất cả đều nhỏ hơn hoặc bằng 59. Vì vậy theo nguyên lý Dirichlet, hai trong số các số nguyên này phải là bằng nhau. Vì các số $a_1, ..., a_{30}$ là đôi một khác nhau và các số $a_1+14, ..., a_{30}+14$ cũng là đôi một khác nhau, nên suy ra phải tìm được chỉ số i và j sao cho $a_i = a_j+14$. Điều đó có nghĩa là có đúng 14 trận đấu trong giai đoạn từ ngày $j+1$ đến ngày i.

Thí dụ 9. Chứng minh rằng, trong số $n+1$ số nguyên dương, mỗi số không lớn hơn $2n$, bao giờ cũng tìm được hai số sao cho số này chia hết cho số kia.

Giải: Gọi các số đã cho là

$$a_1, a_2, \ldots, a_{n+1}.$$

Viết mỗi một số a_j trong $n+1$ số trên dưới dạng:

$$a_j = 2^{k_j} q_j, \quad j = 1, 2, ..., n+1$$

trong đó k_j là nguyên không âm, q_j là số lẻ. Các số $q_1, q_2, ..., q_{n+1}$ là các số nguyên lẻ mỗi số không lớn hơn $2n$. Do trong đoạn từ 1 đến $2n$ chỉ có n số lẻ, nên từ nguyên lý Dirichlet suy ra là hai trong số các số $q_1, q_2, ..., q_{n+1}$ là bằng nhau, tức là tìm được hai chỉ số i và j sao cho $q_i = q_j = q$. Khi đó $a_i = 2^{k_i} q$, $a_j = 2^{k_j} q$. Suy ra nếu $k_i < k_j$ thì a_j chia hết cho a_i, còn nếu $k_i \ge k_j$ thì a_i chia hết cho a_j.

Thí dụ 10. Trong mặt phẳng cho 6 điểm được nối với nhau từng đôi một bởi các cung màu xanh hoặc màu đỏ. Chứng minh rằng luôn tìm được 3 điểm sao cho các cung nối chúng có cùng một màu (ta sẽ nói là chúng tạo thành tam giác xanh hoặc đỏ).

Giải: Chọn điểm P nào đó. Từ nó có 5 cung nối với 5 điểm còn lại. Theo nguyên lý Dirichlet, có 3 trong số 5 cung đó phải có cùng một màu, chẳng hạn là màu xanh. Giả sử đó là các cung PA, PB, PC. Nếu như một trong số 3 cung AB, AC, BC có màu xanh thì nó cùng với hai trong số ba cung PA, PB, PC tạo thành một tam giác xanh. Nếu ngược lại thì tam giác ABC là một tam giác đỏ.

Thí dụ 11. Trên mặt phẳng cho 9 điểm được nối với nhau đôi một bởi các đoạn nối có mầu xanh hoặc đỏ sao cho trong số 3 điểm bất kỳ bao giờ cũng tìm được hai điểm được nối với nhau bởi đoạn nối màu đỏ. Chứng minh rằng trong số các điểm đã cho luôn tìm được 4 điểm mà các đoạn thẳng nối chúng đều có màu đỏ.

Giải: Gọi 9 điểm đã cho là A, B, C, D, E, F, G, H, I. Xét 2 trường hợp:

a) Tìm được một điểm là đầu mút của ít nhất 4 đoạn nối màu xanh chẳng hạn điểm đó là A và các đoạn màu xanh đó là AB, AC, AD, AE. Theo giả thiết, trong số các đoạn nối bất kỳ 3 điểm nào cũng có ít nhất một đoạn màu đỏ, suy ra các đoạn BC, BE, BD, CD, CE, ED là màu đỏ. Vậy B, C, D, E là bốn điểm cần tìm.

b) Mỗi điểm đều là đầu mút của nhiều nhất là 3 đoạn nối màu xanh. Trong trường hợp này, không thể tất cả 9 điểm đều là đầu mút của đúng 3 đoạn nối màu xanh (chứng minh tương tự như trong thí dụ 3, mục 3.2), từ đó suy ra phải tìm được điểm (I chẳng hạn) là đầu mút của nhiều nhất là 2 đoạn nối màu xanh. Khi đó I là đầu mút của ít nhất 6 đoạn màu đỏ, chẳng hạn IA, IB, IC, ID, IE, IF. Theo kết quả của thí dụ 10, trong số 6 điểm A, B, C, D, E, F phải có ít nhất 3 điểm, chẳng hạn A, B, C, sao cho các đoạn nối chúng có cùng màu, và từ giả thiết suy ra màu đó phải là màu đỏ. Vậy I, A, B, C là bốn điểm cần tìm.

3.4. Hệ đại diện phân biệt

Trong nhiều tình huống, sự tồn tại của cấu hình tổ hợp phụ thuộc vào một số điều kiện ràng buộc các tham số ban đầu. Một trong những hướng giải quyết là người ta cố gắng phát hiện ra các điều kiện đó. Bài toán *hệ đại diện phân biệt* trình bày dưới đây là một minh hoạ cho hướng tìm kiếm này.

Giả sử $S_1, S_2, ..., S_m$ là một họ các tập con của một tập hợp S (các S_i không nhất thiết khác nhau). Ta gọi một bộ có thứ tự $a_1, a_2, ..., a_m$ là một *hệ đại diện phân biệt* của họ này nếu $a_i \in S_i$ và $a_i \neq a_j$ $(i \neq j)$. Hệ đại diện phân biệt được viết tắt là TRAN (transversal) và thành phần a_i của hệ được gọi là *đại diện* của tập con S_i $(i = 1, ..., m)$.

Thí dụ. $S = \{1, 2, 3, 4, 5\}$, $S_1 = \{2, 5\}$, $S_2 = \{2, 5\}$, $S_3 = \{1, 2, 3, 4\}$, $S_4 = \{1, 2, 5\}$ có TRAN là (2, 5, 3, 1). Một TRAN khác của họ này là (5, 2, 4, 1).

Không phải lúc nào cũng tìm được TRAN. Một điều dễ nhận thấy là nếu họ đang xét có TRAN, thì mọi hợp của k tập bất kỳ trong họ phải có ít nhất k phần tử (vì luôn tìm được k đại diện khác nhau của k tập đó). Nói khác đi, nếu tìm được k tập nào đó của họ, mà hợp của chúng có ít hơn k phần tử, thì chắc chắn họ đang xét sẽ không có

TRAN. Chẳng hạn trong thí dụ trên, nếu thay tập S_1 của họ đang xét bởi tập {2, 5}, thì họ này sẽ không tồn tại TRAN, vì $S_1 \cup S_2 \cup S_4 = \{2, 5\}$ có ít hơn 3 phần tử.

P. Hall đã chứng minh được điều kiện cần vừa nêu, cũng đồng thời là đủ cho sự tồn tại TRAN, qua định lý đánh giá cận dưới của số TRAN dưới đây:

Định lý Hall. *Giả sử các tập con $S_1, S_2, ..., S_m$ thoả mãn điều kiện:*

$$N(S_i \cup S_{i_2} \cup ... \cup S_{i_k}) \geq k \qquad (1)$$

với mọi $1 \leq k \leq m$, $1 \leq i_1 < i_2 < ... < i_k \leq m$ và mỗi tập con này chứa ít nhất t phần tử. Khi đó:

. nếu $t \leq m$ thì họ đang xét có ít nhất $t!$ TRAN

. nếu $t > m$ thì họ đang xét có ít nhất $t!/(t-m)!$ TRAN.

Điều kiện (1) được gọi là *điều kiện Hall* và ta gọi một họ con của họ $S_1, S_2, ..., S_m$ là *tới hạn* nếu đối với nó bất đẳng thức (1) trở thành đẳng thức.

Chứng minh. Quy nạp theo m. Với $m = 1$, ta có $t = t!/(t-1)!$ TRAN, định lý đúng. Giả sử định lý đúng cho mọi họ tập con của S có ít hơn m tập, ta cần chứng minh định lý đúng cho họ tập con gồm m tập. Chia làm 2 trường hợp:

. Không có họ con tới hạn. Chọn a_1 là một phần tử của S_1. Loại nó ra khỏi $S_2, S_3, ..., S_m$ (nếu có mặt) và gọi họ nhận được là $S_2', S_3', ..., S_m'$. Dễ dàng thử lại họ này thoả mãn điều kiện Hall và mỗi tập thuộc họ có ít nhất $t-1$ phần tử. Theo giả thiết quy nạp họ này có ít nhất $(t-1)!$ TRAN khi $t-1 \leq m-1$ hay $t \leq m$ và có ít nhất $(t-1)!/(t-m)!$ khi $t-1 > m-1$ hay $t > m$. Mặt khác, mỗi TRAN của $S_2', S_3', ..., S_m'$ cùng với a_1, xác định một TRAN của $S_1, S_2, ..., S_m$ (a_1 đại diện cho S_1). Điều này đúng cho mỗi cách chọn a_1 trong số ít nhất t cách chọn nó từ S_1. Từ đó nhận được đánh giá cần chứng minh.

. Có một họ con tới hạn. Không mất tính tổng quát, có thể giả thiết họ đó là $S_1, S_2, ..., S_k$ ($k < m$). Từ sự tồn tại của họ con tới hạn suy ra $t \leq k$, vì vậy theo giả thiết quy nạp, họ $S_1, S_2, ..., S_k$ có ít nhất $t!$ TRAN. Gọi $T' = (a_1, a_2, ..., a_t)$ là một TRAN như thế. Bỏ các phần tử của T', nếu có mặt, ra khỏi các tập $S_{k+1}, ..., S_m$ và gọi các tập thu được là $S'_{k+1}, ..., S'_m$. Khi đó họ $S'_{k+1}, ..., S'_m$ sẽ thoả mãn điều kiện Hall. Thật vậy, nếu có một họ con gồm k' tập của họ đang xét, mà hợp của chúng ít hơn k' phần tử, thì họ con gồm $k+k'$ tập của họ $S_1, S_2, ..., S_m$, nhận được bằng cách ghép họ con này với họ $S_1, S_2, ..., S_k$ sẽ có hợp ít hơn $k+k'$ phần tử và điều này là mâu thuẫn với giả thiết của định lý. Như vậy họ $S'_{k+1}, ..., S'_m$ có ít nhất một TRAN. Lấy ít nhất $t!$ TRAN của họ $S_1, S_2, ..., S_k$ ghép với TRAN này, ta được ít nhất $t!$ TRAN của họ $S_1, S_2, ..., S_m$. Định lý được chứng minh.

Việc xét sự tồn tại cũng như xây dựng TRAN có nhiều ứng dụng trong thực tế. Dưới đây là một số bài toán mà việc giải quyết nó được đưa về việc xây dựng TRAN.

Bài toán người thi hành. Có m người thi hành và n công việc. Giả sử với mỗi người thứ i, ta biết được tập S_i là tập hợp các công việc mà người đó có thể làm. Hỏi có thể phân công mỗi người làm một việc không?

Lời giải của bài toán được dẫn về việc xét sự tồn tại TRAN của họ $\{S_i\}$ và việc xây dựng một TRAN chính là xây dựng một sự phân công như thế.

Bài toán chuyển mạch. Xét một hệ thống chuyển mạch đơn giản gồm 2 nhóm các cực: đầu vào và đầu ra. Tại đầu vào sẽ xuất hiện đòi hỏi về nối mạch. Đòi hỏi này có thể được thoả mãn bằng cách nối nó với một đầu ra nào đó. Tập hợp các đầu vào có đòi hỏi nối mạch được gọi là danh mục đòi hỏi. Đầu vào nối với đầu ra qua một mạch nối và mạch nối này cần phải không được bận nghĩa là nó chưa phục vụ cho đầu vào nào. Các mạch nối như vậy gọi là danh mục tự do. Không giảm tổng quát, ta có thể coi rằng danh mục đòi hỏi gồm m đầu vào đầu tiên và S_i là tập các chỉ số các mạch nối tự do mà theo đó, đòi hỏi từ đầu vào i có thể được chuyển tới một đầu ra. Nếu họ $S_1, S_2, ..., S_m$ có TRAN thì dòng vào gồm m đòi hỏi có thể được phục vụ bởi thiết bị chỉnh mạch. Trong trường hợp không tồn tại TRAN thì cần phải điều chỉnh lại các mạch nối để chọn cách nối khác.

Bài toán đám cưới vùng quê. Tại một làng quê nọ có m chàng trai đến tuổi lấy vợ. Với mỗi chàng trai i, ta biết tập S_i các cô gái mà chàng ta thích. Hỏi rằng có thể ghép mỗi cô cho mỗi chàng mà chàng nào cũng vừa ý hay không? Rõ ràng bài toán được dẫn về việc xét sự tồn tại TRAN của họ $\{S_i\}$. Trong trường hợp tồn tại, mỗi TRAN sẽ tương ứng với một cách ghép mong muốn.

Trong những trường hợp sự tồn tại của TRAN là hiển nhiên thì người ta quan tâm đến việc đếm hoặc liệt kê chúng. Dưới đây là hai thí dụ đếm TRAN mà kết quả đếm được dẫn về các cấu hình đã biết.

Thí dụ 1. Xét tập $\{1, 2, ..., n\}$. Đếm số TRAN của họ tập con

$$S_i = S\text{-}\{i\},\ 1 \le i \le n.$$

Giải: Mỗi TRAN là một hoán vị $(a_1, a_2, ..., a_n)$ của $\{1, 2, ..., n\}$ sao cho $a_i \neq i$ với mọi i, do vậy có thể đồng nhất mỗi TRAN với một mất thứ tự trên tập đang xét. Từ đó nhận được số TRAN cần đếm là D_n (số mất thứ tự - xem chương 2).

Thí dụ 2. Đếm số TRAN của họ tập con của tập $\{1, 2, ..., n\}$: $S_1 = \{1, 2\}$, $S_2 = \{1, 2, 3\}$, $S_3 = \{2, 3, 4\}$, ..., $S_{n-1} = \{n\text{-}2, n\text{-}1, n\}$, $S_n = \{n\text{-}1, n\}$.

Giải: Để bài toán xác định cả với $n = 1$, ta xem trong trường hợp này $S_1 = \{1\}$. Gọi F_n là số TRAN cần đếm (ứng với $n > 1$). Chia các TRAN này thành 2 loại:

- 1 là đại diện cho S_1. Khi đó các thành phần còn lại sẽ là một hệ đại diện của họ $\{2, 3\}, \{2, 3, 4\}, ..., \{n-1, n\}$. Do vậy loại này có F_{n-1} TRAN.
- 2 là đại diện cho S_1. Khi đó bắt buộc 1 phải là đại diện cho S_2 và các thành phần còn lại sẽ là một hệ đại diện của họ $\{3, 4\}, \{3, 4, 5\}, ..., \{n-1, n\}$.Vậy loại này có F_{n-2} TRAN.

Từ đó nhận được hệ thức $F_n = F_{n-1} + F_{n-2}$. Các giá trị $F_1 = 1$, $F_2 = 2$ được tính trực tiếp. Đây cũng là hệ thức truy hồi xác định các số Fibonaci (xem chương 2).

3.5. Định lý Ramsey

3.5.1. Bài toán mở đầu

Trong mục 3.3 ta đã xét bài toán: Trong mặt phẳng cho 6 điểm được nối với nhau từng đôi một bởi các đoạn nối màu xanh hoặc màu đỏ. Chứng minh rằng luôn tìm được 3 điểm sao cho các đoạn nối chúng có cùng một màu (ta sẽ nói là chúng tạo thành tam giác xanh hoặc đỏ).

Lập luận để giải bài toán được dựa trên nguyên lý Dirichlet.

Một cách phát biểu khác của kết quả vừa chứng minh là: Trong số 6 người tại một bàn tiệc luôn tìm được hoặc ba người đôi một quen nhau hoặc ba người đôi một không quen nhau.

Trong mục này chúng ta sẽ còn khảo sát các vấn đề: Hỏi ít nhất phải có bao nhiêu người để chắc chắn tìm được hoặc 4 người đôi một quen nhau hoặc 4 người đôi một không quen nhau? Hỏi ít nhất phải có bao nhiêu người để chắc chắn tìm được hoặc 5 người đôi một quen nhau hoặc 5 người đôi một không quen nhau?

Con số nhỏ nhất vừa nói đến trong các câu hỏi vừa nêu được gọi là các số Ramsey, mang tên nhà toán học người Anh đã chứng minh được định lý nổi tiếng trong lý thuyết tập hợp là sự tổng quát hoá nguyên lý Dirichlet.

3.5.2. Các số Ramsey

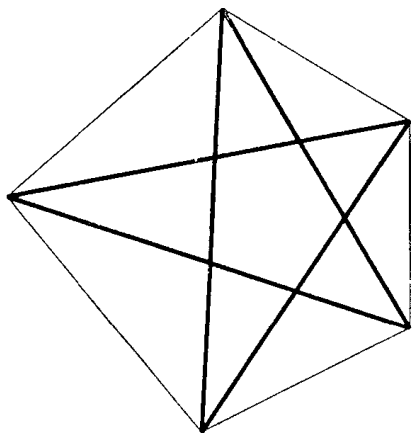
Để có thể phát biểu những kết quả tổng quát hơn chúng ta cần đến một số khái niệm.

Định nghĩa 1. *Gọi K_n là bộ gồm hai tập V, E, trong đó V là tập gồm n điểm còn E là tập các đoạn nối giữa tất cả các cặp điểm trong E. Ta sẽ ký hiệu $K_n = (V, E)$. Ta sẽ gọi các phần tử của V là các đỉnh, và V là tập đỉnh của K_n. Mỗi đoạn nối hai đỉnh $u, v \in V$ sẽ được gọi là một cạnh của K_n và ký hiệu là (u, v), và tập E được gọi là tập cạnh của K_n.*

Ta có thể phát biểu lại kết quả nêu trong mục trước như sau: Giả sử mỗi cạnh của K_6 được tô bởi một trong hai màu xanh hoặc đỏ. Khi đó K_6 luôn chứa hoặc K_3 với tất cả các cạnh được tô màu xanh (gọi tắt là K_3 xanh) hoặc K_3 với tất cả các cạnh được tô màu đỏ (gọi tắt ià K_3 đỏ). Chúng ta sẽ nói rằng số 6 có tính chất (3,3)-Ramsey. Tổng quát hơn ta có định nghĩa sau.

Định nghĩa 2. *Giả sử i và j là hai số nguyên sao cho $i \geq 2, j \geq 2$. Số nguyên dương m có tính chất (i,j)-Ramsey nếu K_m với mỗi cạnh được tô bởi một trong hai màu xanh, đỏ luôn chứa hoặc là K_i đỏ hoặc là K_j xanh.*

Từ kết quả ở mục trước ta thấy 6 có tính chất (3,3)-Ramsey. Nhưng liệu 6 có phải là số nhỏ nhất có tính chất này hay không? Giả sử các cạnh của K_5 được tô màu như chỉ ra trong hình vẽ dưới đây (đỏ - đậm, xanh - nhạt).



Rõ ràng không thể tìm được K_3 đỏ (đậm) cũng như không thể tìm được K_3 xanh (nhạt). Như vậy số 5 không có tính chất (3,3)-Ramsey. Dễ dàng thấy rằng mọi số nguyên dương nhỏ hơn 5 cũng không có tính chất (3,3)-Ramsey. Vậy 6 là số nhỏ nhất có tính chất này.

Định nghĩa 3. *Số Ramsey $R(i,j)$ là số nguyên dương nhỏ nhất có tính chất (i,j)-Ramsey.*

Chẳng hạn, từ kết quả vừa trình bày ở trên, ta có $R(3,3) = 6$, vì 6 có tính chất (3,3)-Ramsey, và những số nguyên dương nhỏ hơn nó không có tính chất này.

Thí dụ 1. Tìm $R(2,7)$ - số nguyên dương nhỏ nhất có tính chất (2,7)-Ramsey.

Giải: Trước hết ta tìm số nguyên dương n sao cho với mọi cách tô các cạnh của K_n bởi hai màu xanh, đỏ luôn tìm được hoặc K_2 đỏ hoặc K_7 xanh. $R(2,7)$ là số nhỏ nhất có tính chất này. Xét một cách tô màu (tuỳ ý) các cạnh của K_7. Rõ ràng hoặc là tìm được ít nhất

một cạnh của K_7 được tô màu đỏ, hoặc là tất cả các cạnh của nó đều được tô bởi màu xanh. Nếu có cạnh tô màu đỏ thì rõ ràng ta có K_2 đỏ. Còn nếu tất cả các cạnh đều tô bởi màu xanh thì ta có K_7 xanh. Vậy số 7 có tính chất (2,7)-Ramsey, và vì thế $R(2,7) \le 7$.

Nhưng $R(2,7)$ không thể nhỏ hơn 7, bởi vì nếu tô tất cả các cạnh của K_6 bởi màu xanh ta sẽ không tìm được K_2 đỏ và cũng không tìm được K_7 xanh.

Vậy $R(2,7) = 7$.

Sử dụng lập luận trong ví dụ vừa trình bày ta có thể chỉ ra rằng:

$R(2,k) = k$, với mọi $k \ge 2$.

Các tính chất cơ bản sau đây của số Ramsey $R(i,j)$ có thể chứng minh bằng các lập luận tương tự như trong các ví dụ đã trình bày:

1. $R(i,j) = R(j,i)$;
2. Nếu m có tính chất (i,j)-Ramsey, thì mọi số $n > m$ cũng có tính chất này;
3. Nếu m không có tính chất (i,j)-Ramsey, thì mọi số $n < m$ cũng không có tính chất này;
4. Nếu $i_1 \ge i_2$ thì $R(i_1,j) \ge R(i_2,j)$.

Để ý rằng khi tìm số $R(i,j)$ ta cần xét sự tồn tại của K_i đỏ hoặc K_j xanh; nghĩa là màu đỏ liên quan đến biến i, còn màu xanh liên quan đến biến j. Tương tự như vậy, khi tìm $R(j,i)$, màu đỏ liên quan đến j còn màu xanh liên quan đến i. Do $R(i,j) = R(j,i)$ nên ta chỉ cần quan tâm đến hoặc là R_i có các cạnh được tô bởi cùng một màu (gọi tắt là R_i cùng màu), hoặc R_j cùng màu.

Việc xác định số Ramsey $R(i,j)$ đòi hỏi chúng ta phải tìm số nguyên dương nhỏ nhất có tính chất (i,j)-Ramsey. Một câu hỏi đặt ra là: Liệu số này có tồn tại với mọi $i \ge 2, j \ge 2$ hay không? Bổ đề và định lý dưới đây sẽ trả lời câu hỏi đặt ra.

Bổ đề 1. *Nếu $i \ge 3$ và $j \ge 3$ thì*

$$R(i,j) \le R(i,j-1) + R(i-1,j). \qquad (1)$$

Chứng minh. Giả sử $m = R(i,j-1) + R(i-1,j)$. Ta sẽ chứng minh rằng m có tính chất (i,j)-Ramsey. Giả sử các cạnh của K_m được tô bởi hai màu xanh, đỏ, và v là một đỉnh của K_m. Ta phân tập đỉnh V của K_m ra làm hai tập:

A - tập tất cả các đỉnh nối với v bởi cạnh đỏ;

B - tập tất cả các đỉnh nối với v bởi cạnh xanh.

Do

$$|A| + |B| = |A \cup B| = m - 1 = R(i,j-1) + R(i-1,j) - 1$$

nên hoặc là $|A| \geq R(i-1,j)$ hoặc là $|B| \geq R(i,j-1)$. Thực vậy, nếu trái lại ta có $|A| < R(i-1,j)$ và $|B| < R(i,j-1)$, từ đó suy ra điều phi lý sau

$$m - 1 = |A \cup B| < R(i,j-1) + R(i-1,j) - 1 = m - 1.$$

Xét trường hợp $|A| \geq R(i-1,j)$. Gọi $K_{|A|}$ là bộ gồm tập đỉnh A và tập cạnh là các cạnh nối các đỉnh trong A của K_m. Ta sẽ chỉ ra rằng $K_{|A|}$ hoặc chứa K_i đỏ hoặc chứa K_j xanh. Do $|A| \geq R(i-1,j)$, nên $K_{|A|}$ hoặc chứa K_{i-1} đỏ hoặc chứa K_j xanh. Nếu $K_{|A|}$ chứa R_{i-1} đỏ thì bổ sung vào nó đỉnh v và các cạnh nối v với các đỉnh trong A ta thu được R_i đỏ. Vậy $K_{|A|}$ và do đó K_m luôn chứa hoặc K_i đỏ hoặc K_j xanh.

Trường hợp $|B| \geq R(i,j-1)$ được xét tương tự.

Như vậy m có tính chất (i,j)-Ramsey, từ đó suy ra bất đẳng thức (1) được chứng minh.

Từ kết quả của bổ đề sử dụng phép qui nạp toán học ta có thể chứng minh kết quả sau đây:

Định lý 1 (Định lý Ramsey). *Nếu $i \geq 2, j \geq 2$ là các số nguyên dương thì luôn tìm được số nguyên dương với tính chất (i,j)-Ramsey (từ đó suy ra số $R(i,j)$ là tồn tại).*

Chứng minh. Giả sử $P(n)$ là mệnh đề sau:

$P(n)$: Nếu $i + j = n$ thì luôn tìm được số nguyên với tính chất (i,j)-Ramsey.

Khi $n = 4$, ta có $i = j = 2$, từ kết quả của ví dụ 1 suy ra $P(4)$ là đúng. Giả sử $P(n)$ đúng, ta phải chứng minh $P(n+1)$ cũng đúng. Giả sử $i + j = n+1$. Suy ra $i + (j-1) = n$ và $(i-1) + j = n$. Theo giả thiết qui nạp, luôn tìm được số nguyên có tính chất $(i, j-1)$-Ramsey và số nguyên với tính chất $(i-1, j)$-Ramsey. Từ đó suy ra các số $R(i,j-1)$ và $R(i-1,j)$ là tồn tại. Từ đó và từ bất đẳng thức (1) suy ra số $R(i,j)$ cũng tồn tại. Vậy $P(n+1)$ là đúng.

Theo nguyên lý qui nạp $P(n)$ đúng với mọi $i \geq 2, j \geq 2$. Từ đó suy ra $R(i,j)$ luôn tồn tại với mọi $i \geq 2, j \geq 2$.

Chúng ta đã có các kết quả sau:

$$R(2,k) = R(k,2) = k;$$
$$R(3,3) = 6.$$

Khi $i \geq 2, j \geq 2$, việc tìm các số $R(i,j)$ càng khó khi i, j càng lớn. Hiện nay mới chỉ biết rất ít các số Ramsey. Chúng ta sẽ tính thêm một vài số Ramsey.

Thí dụ 2. Tìm $R(3,4)$.

Giải: Từ bổ đề 1 ta có

$$R(3,4) \le R(3,3) + R(2,4) = 6 + 4 = 10.$$

Để xác định xem có phải $R(3,4) < 10$, ta phải xét tất cả các cách tô màu cạnh của K_9. Nếu K_9 không chứa K_3 đỏ cũng như không chứa K_4 xanh với một cách tô màu cạnh của nó nào đó, thì $R(3,4) = 10$. Còn nếu như ứng với mọi cách tô màu, K_9 có hoặc K_3 đỏ hoặc K_4 xanh thì ta lại phải xét vấn đề đó cho K_8. Việc xét tất cả các cách tô màu cạnh của K_9 không phải là dễ dàng, do K_9 có 36 cạnh nên có tất cả $2^{36} > 60\times10^9$ cách tô màu khác nhau.

Rất may là ta không cần khảo sát các cách tô màu K_9, vì có thể sử dụng kết quả sau đây:

Nếu $i \ge 3, j \ge 3$ và nếu $R(i,j-1)$ và $R(i-1,j)$ là các số chẵn thì

$$R(i,j) \le R(i,j-1) + R(i-1,j) - 1 \qquad (2)$$

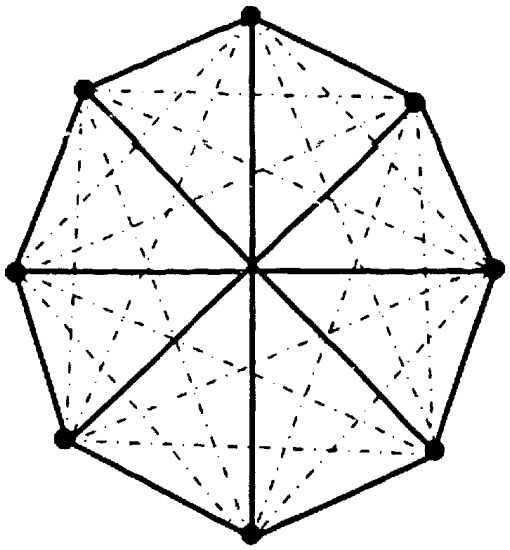
Chứng minh bất đẳng thức (2) dựa vào chứng minh của Bổ đề 1.

Sử dụng (2), khi $i = 3, j = 4$, ta thu được

$$R(3,4) \le R(3,3) + R(2,4) - 1 = 6 + 4 - 1 = 9,$$

và như vậy $R(3,4) \le 9$.

Để chỉ ra 8 không có tính chất (3,4)-Ramsey, xét cách tô màu K_8 mô tả trong hình vẽ 2, trong đó các cạnh liền nét được tô màu đỏ, còn các cạnh đứt nét được tô màu xanh.



Hình 2. Tô màu K_8

Rõ ràng không thể tìm được K_3 đỏ cũng như K_4 xanh. Vậy $R(3,4) = 9$.

Ví dụ 3. Tìm $R(3,5)$.

Giải: Từ (1) ta có

$$R(3,5) \le R(3,4) + R(2,5) = 9 + 5 = 14.$$

Ta sẽ chứng minh $R(3,5) = 14$, bằng cách chỉ ra rằng số 13 không có tính chất (3,5)-Ramsey. Vẽ K_{13}, các đỉnh của nó đánh số từ 1 đến 13. Tô màu đỏ cho cạnh (i,j) nếu $|i - j| = 1, 5, 8$ hay 12. Tô màu xanh cho các cạnh còn lại. Có thể kiểm tra được rằng với cách tô màu đó K_{13} không chứa K_3 đỏ cũng không chứa K_5 xanh.

Bảng dưới đây cho ta những giá trị đã biết của $R(i,j)$

$i \setminus j$	2	3	4	5	6
2	2				
3	3	6			
4	4	9	18		
5	5	14	25 - 27	43 – 52	
6	6	18	34 - 43	57 – 94	102 - 169
7	7	23	≥ 49	≥ 76	
8	8	28	≥ 53	≥ 94	
9	9	36	≥ 69		

Các số $R(3,3)$, $R(4,3)$, $R(5,3)$ và $R(4,4)$ được tìm thấy từ năm 1955 bởi A.M. Gleason và R.E Gleenwood, R(6,3) - J.G. Kalbfleisch năm 1966, $R(7,3)$ - J.E. Graver, J. Yackel năm 1968, $R(8,3)$ - B. McKay và Z.Ke Min mới gần đây, $R(9,3)$ - C.M. Grinstead và S.M. Robets năm 1982.

3.5.3. Tổng quát hoá

Các số Ramsey giới thiệu trong mục trước chỉ là một trong họ các số Ramsey. Trong mục này chúng ta sẽ xét một họ các số Ramsey tổng quát hơn.

Chẳng hạn, nếu ta tô màu các cạnh của K_n bởi ba màu xanh, đỏ, tím, thì số n ít nhất phải là bao nhiêu để chắc chắn tìm được hoặc K_3 đỏ, hoặc K_3 xanh hoặc K_3 tím? Số n nhỏ nhất có tính chất như vậy ta sẽ ký hiệu là $R(3,3,3;2)$. Con số 2 được viết như là một thành phần của $R(3,3,3; 2)$ bởi vì các cạnh (đối tượng được tô màu) được xác định bởi 2 đỉnh. Con số 2 này cũng có thể thay bởi một số nguyên dương bất kỳ. Ba số 3 cũng có thể thay bởi các số nguyên dương tuỳ ý để có thể thu được một họ mới các số Ramsey. Ví dụ: $R(5,4,7; 2)$ là số nguyên dương nhỏ nhất n sao cho với mọi cách tô màu các cạnh của K_n bởi 3 màu xanh, đỏ, tím K_n luôn chứa hoặc K_5 đỏ hoặc K_4 xanh hoặc K_7 tím.

Định nghĩa 4. *Giả sử $i_1, i_2, ..., i_n$ là các số nguyên dương, trong đó $i_j \geq 2$, với mọi j. Số nguyên dương m được nói là có tính chất $(i_1, i_2, ..., i_n; 2)$-Ramsey nếu với mọi cách tô màu các cạnh của K_m bởi n màu 1, 2, ..., n luôn tìm được trong nó K_{i_j} màu i_j với ít nhất một j nào đó.*

Số nguyên dương nhỏ nhất với tính chất $(i_1, i_2, .., i_n; 2)$-Ramsey được gọi là số Ramsey $R(i_1, i_2, ..., i_n; 2)$.

Chú ý rằng, nếu $n = 2$, số Ramsey $R(i_1, i_2 ; 2)$ chính là các số Ramsey $R(i_1, i_2)$ trong mục trước.

Chúng ta lại có thể đặt ra câu hỏi là: Tồn tại hay chăng các số Ramsey vừa được định nghĩa? Câu trả lời là khẳng định. Để chứng minh sự tồn tại của các số Ramsey có thể sử dụng phương pháp chứng minh bằng quy nạp.

Chúng ta biết rất ít về các số Ramsey $R(i_1, i_2, ..., i_n; 2)$ khi $n \geq 3$. Tuy nhiên, nếu $i_j = 2$ với mọi j, thì có thể chứng minh được rằng: $R(2,...,2; 2) = 2$. Khi mỗi $i_j \geq 3$, cho đến thời điểm hiện tại mới biết duy nhất một giá trị $R(3,3,3; 2) = 17$.

Thí dụ 4. (R.E. Greenwood, A.M. Gleason, 1955) Chứng minh rằng R(3,3,3; 2) = 17.

Giải. Xét một cách tô màu nào đó các cạnh của K_{17} bởi ba màu xanh, đỏ, tím. Chọn v là một đỉnh nào đó của K_{17}. Trong số 16 đoạn nối v với các đỉnh còn lại, theo nguyên lý Dirichlet luôn tìm được 6 đoạn có cùng một màu. Không giảm tổng quát có thể coi các cạnh nối v với các đỉnh 1, 2, 3, 4, 5, 6 có cùng màu đỏ. Nếu có một trong số các cạnh nối giữa các đỉnh 1,2, ..., 6 có màu đỏ thì cạnh này cùng với đỉnh v lập thành K_3 đỏ. Nếu trái lại, 6 đỉnh 1, 2, ..., 6 chỉ nối với nhau bởi các cạnh có hai màu xanh hoặc tím, thì theo ví dụ mở đầu trong số chúng luôn tìm được hoặc K_3 xanh hoặc K_3 tím. Vậy, K_{17} luôn chứa K_3 đỏ, xanh hoặc tím. Do đó, $R(3,3,3; 2) \leq 17$.

Để chỉ ra $R(3,3,3; 2) > 16$ cần xây dựng một cách tô màu K_{16} sao cho không tìm được K_3 cùng màu (xem trong cuốn sách của C. Berge).

Có một cách khác để tiếp tục phát triển các số Ramsey và tiếp tục tổng quát hoá các ý tưởng mà ta vừa xét. Xét bài toán mở đầu, trong đó ta xét sự tồn tại K_3 đỏ hoặc xanh trong K_6. Bắt đầu với K_6 với tập đỉnh V. Ta xét tất cả các tập con 2 phần tử của V (tức là các cạnh của K_6) và chia các tập con này ra làm hai họ C_1 và C_2. Số 6 có tính chất (3,3)-Ramsey nếu và chỉ nếu:

i) Tìm được tập con 3 phần tử của V sao cho mọi tập con 2 phần tử của nó đều thuộc vào C_1 hoặc

ii) Tìm được tập con 3 phần tử của V sao cho mọi tập con 2 phần tử của nó đều thuộc vào C_2.

Nếu coi C_1 là tập các cạnh được tô màu đỏ và C_2 là tập các cạnh được tô màu xanh, thì rõ ràng ta có tam giác đỏ khi và chỉ khi điều kiện i) được thực hiện và có tam giác xanh khi và chỉ khi điều kiện ii) được thực hiện. Tuy nhiên trong cách mô tả này chúng ta không cần dùng đến khái niệm cạnh. Các tính chất được phát biểu trong ngôn ngữ tập hợp và các tính chất của một họ các tập con của nó. Cách mô tả này cho phép xét việc phân các tập con có kích thước r tuỳ ý (không phải chỉ có 2) ra thành một số các họ con (không nhất thiết chỉ phân làm hai họ C_1, C_2 như trong ví dụ vừa nêu). Ta đi đến định nghĩa tổng quát sau của số Ramsey

Định nghĩa 5. *Giả sử $i_1, i_2, ..., i_n, r$ là các số nguyên dương, trong đó $n \geq 2$, và $i_j \geq r$ với mọi j. Số nguyên dương m được nói là có tính chất $(i_1, ..., i_n ; r)$-Ramsey nếu như mệnh đề sau đây là đúng:*

> *Nếu S là tập m phần tử và các tập con r phần tử của S được phân chia vào n họ $C_1, C_2, ..., C_n$, thì với một j nào đó tìm được tập con của S có lực lượng i_j sao cho mỗi tập con r phần tử của nó đều thuộc vào C_j.*

Số nguyên dương nhỏ nhất có tính chất $(i_1, ..., i_n ; r)$-Ramsey được gọi là số Ramsey $R(i_1, ..., i_n ; r)$.

Định lý 3 (Ramsey, 1930). *Nếu $i_1, ..., i_n, r$ là các số nguyên dương, trong đó $n \geq 2$ và $i_j \geq r$ với mỗi j, thì số Ramsey $R(i_1, ..., i_n ; r)$ luôn tồn tại.*

Khi r = 1, số $R(i_1, ..., i_n ; 1)$ có thể xác định khá dễ dàng, bởi vì chúng ta chỉ phải xét các tập con 1 phần tử của S. Định lý dưới đây cho công thức tính số này.

Định lý 4. $R(i_1, ..., i_n ; 1) = i_1 + ... + i_n - (n - 1)$.

Chứng minh. Đặt

$$m = i_1 + ... + i_n - (n-1).$$

Trước hết ta chỉ ra rằng m có tính chất $(i_1, ..., i_n; r)$-Ramsey. Lấy S là tập m phần tử và chia các tập con 1 phần tử của nó ra làm n lớp $C_1, ..., C_n$. Trước hết nhận thấy rằng phải tìm được chỉ số $j0$ sao cho $|C_{j0}| \geq i_{j0}$ (Nếu trái lại, $|C_j| < i_j$ với mọi j, thì $|C_j| \leq i_j - 1$. Suy ra $m = |C_1| + ... + |C_n| \leq (i_1-1) + ... + (i_n-1) = i_1 + ... + i_n - n = m - 1$?!). Nếu ta lấy i_{j0} phần tử của C_{j0} thì ta có tập con của S với lực lượng i_{j0} sao cho mọi tập con 1 phần tử của nó đều thuộc C_{j0}. Điều đó chứng tỏ rằng $R(i_1, ..., i_n ; 1) \leq i_1 + ... + i_n - (n-1)$.

Bây giờ ta sẽ chỉ ra rằng $m-1 = i_1 + ... + i_n - n$ không có tính chất $(i_1, ..., i_n ; r)$-Ramsey. Lấy tập S gồm $i_1 + ... + i_n - n$ phần tử. Phân các tập con 1 phần tử của nó vào n

lớp C_1, ..., C_n sao cho $|C_j| = i_j - 1$. Rõ ràng với cách phân chia này không thể tìm được tập con của S có lực lượng i_j sao cho mọi tập con 1 phần tử của nó thuộc cùng một lớp C_j.

Khi $i_1 = ... = i_n = 2$, ta có

$$R(2,...,2; 1) = n + 1.$$

Sự kiện này cho thấy định lý Ramsey là sự tổng quát hoá của nguyên lý Dirichlet. Bởi vì trong ngôn ngữ của số Ramsey, $R(2,...,2; 1) = n + 1$ có nghĩa là $n + 1$ là số nguyên dương nhỏ nhất có tính chất sau: Nếu tập S có lực lượng $n + 1$ và nếu phân các phần tử của nó vào n tập C_1, ..., C_n thì phải tìm được tập con 2 phần tử của S sao cho các phần tử của nó thuộc vào cùng một tập C_j nào đó. Nói cách khác luôn tìm được chỉ số j sao cho tập C_j có ít ra là 2 phần tử. Đó chính là nội dung của nguyên lý Dirichlet.

Bài tập

1. Trên mặt phẳng cho $n \geq 6$ điểm, khoảng cách giữa các cặp điểm là khác nhau từng đôi. Mỗi điểm được nối với điểm gần nó nhất. Chứng minh rằng mỗi điểm được nối với không quá 5 điểm.

2. Một trung tâm máy tính có 151 máy vi tính. Các máy của trung tâm được đặt tên bởi một số nguyên dương trong khoảng từ 1 đến 300 sao cho không có hai máy nào được đặt tên trùng nhau. Chứng minh rằng luôn tìm được 2 máy có tên là các số nguyên liên tiếp.

3. Các học sinh của một lớp học gồm 45 nam và 35 nữ được xếp ra thành một hàng ngang. Chứng minh rằng, trong hàng đó luôn tìm được hai học sinh nam mà ở giữa họ có 8 người đứng xen vào.

4. 12 cầu thủ bóng rổ đeo áo với số từ 1 đến 12 đứng tập trung thành một vòng tròn giữa sân. Chứng minh rằng luôn tìm được 3 người liên tiếp có tổng các số trên áo là lớn hơn hoặc bằng 20

5. Chứng minh rằng trong số 10 người bất kỳ bao giờ cũng tìm được hoặc là hai người có tổng số tuổi là chia hết cho 16, hoặc là hai người mà hiệu số tuổi của họ là chia hết cho 16.

6. Cần có ít nhất bao nhiêu bộ có thứ tự gồm 2 số nguyên (a, b) sao cho chắc chắn tìm được trong số đó hai bộ (c, d) và (e, f) sao cho $c - e$ và $d - f$ là các số có chữ số tận cùng bằng 0?

7. 17 nhà bác học đôi một viết thư trao đổi với nhau về 3 chủ đề, mỗi cặp chỉ trao đổi với nhau về 1 chủ đề. Chứng minh rằng luôn tìm được 3 nhà bác học đôi một viết thư trao đổi với nhau về cùng một chủ đề.

8. Trong không gian cho 9 điểm có toạ độ nguyên. Chứng minh rằng trong số 9 điểm đã cho luôn tìm được hai điểm sao cho đoạn thẳng nối chúng đi qua điểm có toạ độ nguyên.

9. Chứng minh rằng trong số 10 người bất kỳ luôn tìm được hoặc là 4 người đôi một quen nhau và 3 người đôi một không quen nhau hoặc là 4 người đôi một không quen nhau và 4 người đôi một quen nhau.

4

BÀI TOÁN LIỆT KÊ

4.1. Giới thiệu bài toán

Nếu như trong bài toán đếm (xem chương 2), ta chỉ đòi hỏi đếm số cấu hình tổ hợp là bao nhiêu thì trong nhiều tình huống, ta còn phải cần chỉ rõ những cấu hình tổ hợp đó là những cấu hình nào. Bài toán đưa ra danh sách tất cả cấu hình tổ hợp có thể có, được gọi là *bài toán liệt kê tổ hợp*. Vì thế, khác với bài toán đếm tìm kiếm một công thức cho lời giải, bài toán liệt kê lại cần *xác định một thuật toán* để theo đó có thể lần lượt xây dựng được tất cả các cấu hình đang quan tâm. Rõ ràng có nhiều cách liệt kê, tuy nhiên chúng phải đảm bảo 2 nguyên tắc:

- không được lặp lại một cấu hình,
- không được bỏ sót một cấu hình.

Có thể nói rằng phương pháp liệt kê là cách cuối cùng để có thể giải được một số bài toán tổ hợp hiện nay. Khó khăn chính của phương pháp này là sự "bùng nổ tổ hợp". Để xây dựng chừng 1 tỷ cấu hình (con số này không phải là lớn đối với các bài toán tổ hợp - xem lại các số mất thứ tự D_n, số phân bố U_n, số hình vuông la tinh l_n, ...) và giả thiết rằng mỗi thao tác xây dựng mất khoảng 1 giây, ta phải bỏ ra quãng 31 năm mới giải xong. Tuy nhiên với sự phát triển của máy tính điện tử, bằng phương pháp liệt kê, nhiều bài toán tổ hợp đã tìm thấy lời giải. Mặt khác, chính sự nỗ lực tìm kiếm những giải pháp hữu hiệu cho những bài toán khó thuộc lĩnh vực này, đã thúc đẩy mạnh mẽ sự phát triển của nhiều ngành toán học. Trong chương này, sau phần giới thiệu khái niệm

thuật toán, chúng ta sẽ trình bày hai phương pháp liệt kê thường sử dụng nhất, đó là Thuật toán sinh và đặc biệt là thuật toán quay lui, một thuật toán có tính phổ dụng cao.

4.2. Thuật toán và độ phức tạp tính toán

Như đã giới thiệu ở trên, ta hiểu việc giải bài toán liệt kê là xây dựng một thuật toán để theo đó có thể lần lượt xây dựng được tất cả các cấu hình cần quan tâm. Vậy cần hiểu thuật toán là gì? Mục này dành để giới thiệu khái niệm thuật toán và một số vấn đề liên quan cần thiết cho việc trình bày các chương sau.

4.2.1. Khái niệm thuật toán

Thuật toán đã được biết đến từ rất lâu. Bản thân thuật ngữ Thuật toán (Algorithm) là viết tắt tên của nhà toán học thế kỷ thứ IX: Abu Ja'fa Mohammed ibn Musa al-Khowarizmi. Đầu tiên, thuật toán được hiểu như là các qui tắc thực hiện các phép tính số học với các con số được viết trong hệ cơ đếm thập phân. Cùng với sự phát triển của máy tính, khái niệm thuật toán càng được hiểu theo nghĩa rộng hơn và chính xác hơn. Khái niệm thuật toán được định nghĩa một cách hình thức chính xác thông qua máy Turing. Tuy nhiên, trong giáo trình này chúng ta chưa cần đến định nghĩa chính xác này, mà có thể hiểu thuật toán một cách trực quan hơn qua định nghĩa sau.

Định nghĩa. *Ta hiểu thuật toán giải bài toán đặt ra là một thủ tục xác định bao gồm một dãy hữu hạn các bước cần thực hiện để thu được lời giải của bài toán.*

Thuật toán có các đặc trưng sau đây:

- *Đầu vào (Input):* Thuật toán nhận dữ liệu vào từ một tập nào đó.
- *Đầu ra (Output):* Với mỗi tập các dữ liệu đầu vào, thuật toán đưa ra các dữ liệu tương ứng với lời giải của bài toán.
- *Chính xác (Precision):* Các bước của thuật toán được mô tả chính xác.
- *Hữu hạn (Finiteness):* Thuật toán cần phải đưa được đầu sau một số hữu hạn (có thể rất lớn) bước với mọi đầu vào.
- *Đơn trị (Uniqueness):* Các kết quả trung gian của từng bước thực hiện thuật toán được xác định một cách đơn trị và chỉ phụ thuộc vào đầu vào và các kết quả của các bước trước.
- *Tổng quát (Generality):* Thuật toán có thể áp dụng để giải mọi bài toán có dạng đã cho.

Thí dụ 1. Cho 3 số nguyên a, b, c. Mô tả thuật toán tìm số lớn nhất trong ba số đã cho.

Giải. Tuy rằng bài toán đặt ra là rất đơn giản nhưng mục đích của chúng ta là dùng nó để giải thích khái niệm thuật toán. Thuật toán gồm các bước sau:

Bước 1. Đặt $x := a$;
Bước 2. Nếu $b > x$ thì đặt $x:=b$;
Bước 3. Nếu $c > x$, thì đặt $x:=c$.

Tư tưởng của thuật toán là duyệt lần lượt giá trị của từng số và giữ lại giá trị lớn nhất vào biến x. Kết thúc thuật toán x cho số nguyên lớn nhất trong 3 số đã cho.

Ký hiệu $y := z$ trong mô tả thuật toán ở trên có nghĩa là thay thế giá trị đang có của y bởi giá trị của z. Khi phép toán $y := z$ được thực hiện xong, giá trị của z không bị thay đổi. Ta gọi := là *toán tử gán*.

Bây giờ ta sẽ theo dõi quá trình thực hiện thuật toán với những giá trị cụ thể của a, b, c. Trước hết giả sử

$$a = 1, \quad b = 5, \quad c = 3.$$

Tại bước 1, ta đặt giá trị của x là a (1). Tại bước 2, do $b > x$ (5 > 1), nên x được đặt bằng b (5). Tại bước 3, do điều kiện $c > x$ (3 > 5) không được thực hiện, nên ta không phải làm đông tác gán. Kết thúc thuật toán, x có giá trị 5 là giá trị lớn nhất trong 3 số a, b, c.

Giả sử

$$a = 7, \quad b = 2, \quad c = 31.$$

Tại bước 1, ta đặt giá trị của x là a (7). Tại bước 2, do điều kiện $b > x$ (2 > 7) không thoả mãn, nên ta không làm gì cả.. Tại bước 3, do $c > x$ (31 > 7), nên ta gán x bằng 31. Kết thúc thuật toán, x có giá trị 31 là giá trị lớn nhất trong 3 số a, b, c.

Bây giờ ta sẽ thấy rằng thuật toán vừa mô tả có các tính chất nêu ở trên.

Thuật toán nhận *đầu vào* là ba số a, b, c và đưa kết quả ở *đầu ra* là x.

Các bước của thuật toán được mô tả *chính xác* đến mức ta có thể viết chương trình theo thuật toán trên ngôn ngữ lập trình và thực hiện trên máy tính.

Nếu đầu vào là đã xác định, kết quả tại mỗi bước của thuật toán được xác định duy nhất. Chẳng hạn, với đầu vào

$$a = 7, \quad b = 2, \quad c = 31,$$

tại bước 3 của thuật toán, x luôn được đặt bằng 31, không phụ thuộc vào việc thuật toán được thực hiện bằng tay hay bởi máy tính.

Thuật toán kết thúc sau ba bước và đưa ra lời giải của bài toán, vì vậy, thuật toán là hữu hạn.

Thuật toán trình bày trong ví dụ luôn đưa ra giá trị của số lớn nhất trong ba số *bất kỳ*, như vậy, thuật toán có tính tổng quát.

4.2.2. Mô tả thuật toán bằng ngôn ngữ phỏng PASCAL

Cách mô tả thuật toán bằng lời như trình bày trong thí dụ ở mục trên không thật thuận tiện cho việc cài đặt thuật toán trên những ngôn ngữ lập trình cụ thể, chẳng hạn PASCAL, C, BASIC,... Tất nhiên có thể mô tả thuật toán sử dụng một ngôn ngữ lập trình nào đó. Khi đó ta chỉ có thể sử dụng những cấu trúc lệnh của ngôn ngữ đã chọn. Điều đó có thể làm cho việc mô tả thuật toán trở nên phức tạp đồng thời cũng rất khó hiểu. Vì thế, để mô tả thuật toán ta sử dụng ngôn ngữ phỏng PASCAL, trong đó cho phép vừa mô tả thuật toán bằng ngôn ngữ đời thường vừa sử dụng những cấu trúc lệnh tương tự như của ngôn ngữ lập trình PASCAL (tất nhiên, các qui tắc cú pháp của ngôn ngữ lập trình PASCAL không nhất thiết phải tuân thủ). Dưới đây ta liệt kê một số câu lệnh chính được sử dụng để mô tả thuật toán.

Câu lệnh procedure(function). Mô tả thuật toán trong ngôn ngữ phỏng PASCAL được bắt đầu từ câu lệnh **procedure (function),** trong đó ta đặt tên cho thuật toán và mô tả danh sách biến của thuật toán. Chẳng hạn, câu lệnh

function *maximum(a,b,c);*

cho biết tên thuật toán được mô tả là *maximum*, có 3 biến *a, b, c;*

procedure *Hoanvi(n);*

cho biết tên thuật toán được mô tả là *Hoanvi* với biến là *n.*

Các bước của thuật toán được mô tả trong *thân thủ tục (hàm)* được bắt đầu bởi **begin** và kết thúc bởi **end.**

Thí dụ:

```
function maximum(a,b,c);
begin
    (Thân hàm).
end;

procedure Hoanvi(n);
begin
    (Thân thủ tục).
end;
```

Chú ý: Khi mô tả thuật toán bắt đầu bằng **function,** khi kết thúc làm việc, thuật toán sẽ đưa ra giá trị được ghi nhận trong tên hàm. Vì vậy trong thân hàm phải có mặt câu lệnh gán giá trị cho hàm. Trong ví dụ trên *maximum* sẽ ghi nhận giá trị lớn nhất trong ba biến *a, b, c.*

Câu lệnh gán. Câu lệnh gán được sử dụng để gán giá trị cho các biến. Vế trái của câu lệnh là tên của biến, còn vế phải là biểu thức của các hằng, biến đã gán giá trị hoặc các hàm đã được định nghĩa. Ký hiệu := được sử dụng để biểu diễn phép gán.

Thí dụ:

- *variable := expression;*
- *max := a;*
- *x* := số lớn nhất trong các số $a_1, a_2, ..., a_n$;

Khối câu lệnh. Các câu lệnh có thể nhóm lại thành một khối. Để mô tả khối lệnh ta sử dụng *begin* và *end.*

Thí dụ:

```
begin
    Câu lệnh 1;
    Câu lệnh 2;
    ......
    Câu lệnh n;
end;
```

Các câu lệnh trong khối được thực hiện tuần tự. Dưới đây, thuật ngữ *câu lệnh* được dùng để chỉ chung một câu lệnh cũng như một khối câu lệnh.

Chú giải. Để diễn giải thêm nội dung của các đoạn lệnh có thể sử dụng các câu chú giải. Các dòng chú giải được đặt trong dấu hai dấu { } hoặc (* *).

Thí dụ:

```
(*  x là phần tử lớn nhất trong danh sách L *)
{  d là số phần tử của danh sách L }
```

Câu lệnh điều kiện. Câu lệnh đơn giản là

if *điều kiện* **then** *câu lệnh*;

Khi thực hiện câu lệnh, *điều kiện* sẽ được kiểm tra, nếu nó được thoả mãn thì *câu lệnh* sẽ được thực hiện.

Nhiều khi ta cần phải thực hiện một thao tác nào đó khi điều kiện được thực hiện, còn nếu ngược lại ta lại phải thực hiện một thao tác khác. Khi đó, có thể sử dụng câu lệnh phức tạp hơn sau đây

if *điều kiện* **then** *câu lệnh 1*
else *câu lệnh 2*;

Các câu lệnh lặp. Ta sẽ sử dụng các câu lệnh sau đây

for *biến := giá_trị_đầu* **to** *giá_ trị_ cuối* **do** *Câu lệnh*;

Tại đầu vòng lặp, *biến* sẽ được gán cho *giá_ trị_ đầu*, nếu giá trị đầu nhỏ hơn hơn hoặc bằng giá trị cuối và *câu lệnh* sẽ được thực hiện với giá trị này của *biến*. Tiếp đến, giá trị của *biến* sẽ tăng lên 1 và *câu lệnh* sẽ được thực hiện với giá trị mới của *biến*. Quá trình sẽ được tiếp tục cho đến khi *biến* bằng *giá_ trị_ cuối*. Sau khi thực hiện xong *câu lệnh* với giá trị của *biến* bằng *giá_ trị_ cuối*, sẽ chuyển sang thực hiện câu lệnh tiếp theo. Nếu *giá_ trị_ đầu* lớn hơn *giá_ trị_ cuối* thì không có câu lệnh nào được thực hiện.

Câu lệnh lặp thứ hai được sử dụng là câu lệnh "while" sau đây

while *điều kiện* **do** *câu lệnh*;

Khi câu lệnh này được sử dụng, điều kiện sẽ được kiểm tra, nếu nó là đúng thì *câu lệnh* được thực hiện, điều đó có thể dẫn tới sự thay đổi giá trị của các biến trong *điều kiện*. Nếu điều kiện vẫn là đúng sau khi thực hiện *câu lệnh* thì *câu lệnh* lại được thực hiện. Điều đó sẽ tiếp diễn cho đến khi *điều kiện* là sai.

Câu lệnh lặp thứ ba được sử dụng là câu lệnh "repeat" sau đây

repeat *câu lệnh* **until** *điều kiện*;

Khi câu lệnh này được sử dụng, *câu lệnh* được thực hiện, điều đó có thể dẫn tới sự thay đổi giá trị của các biến trong *điều kiện*. Nếu điều kiện vẫn là đúng, thì *câu lệnh* lại được thực hiện. Điều đó sẽ tiếp diễn cho đến khi *điều kiện* là đúng.

Thí dụ 2. Thuật toán tìm số lớn nhất trong 3 số trong thí dụ 1 có thể mô tả như sau

```
function maximum(a,b,c);
begin
    x:=a;
    if  b > x  then x:=b;    (* Nếu b lớn hơn x thì gán lại x *)
    if  c > x  then  x:=c;  (* Nếu c lớn hơn x thì gán lại x *)
    maximum:= x;
end;
```

Thí dụ 3. Thuật toán tìm số lớn nhất trong dãy hữu hạn số.

Đầu vào: Dãy gồm n số $a_1, a_2, \ldots, a_n$.

Đầu ra: *large* - số lớn nhất trong dãy đã cho.

```
procedure Find_Large(a, n, large);
begin
    large:=a1;
    for i:= 2 to n do
        (* Nếu ai lớn hơn large thì gán lại large *)
        if ai > large then large:=ai;
end;
```

Trong việc giải các bài toán phức tạp, ta thường phải phân rã nó ra thành các bài toán con. Ta sẽ xây dựng các thủ tục để giải các bài toán con, sau đó các thủ tục này sẽ được tập hợp để giải bài toán đặt ra. Thí dụ dưới đây minh hoạ cho tư tưởng này.

Thí dụ 4. Tìm số nguyên tố lớn hơn số nguyên dương n.

Giải. Trước hết ta xây dựng thủ tục kiểm tra xem một số nguyên dương m có phải là số nguyên tố hay không (Thủ tục *Nguyên_tố*). Sử dụng thủ tục này ta xây dựng thuật toán giải bài toán đặt ra. Do ước số nguyên tố của số nguyên dương m bao giờ cũng không vượt quá $\sqrt{m}$, nên m sẽ là số nguyên tố nếu như nó không có ước số nào trong các số nguyên dương từ 2 đến $[\sqrt{m}]$.

- *Thuật toán kiểm tra một số nguyên dương có phải là nguyên tố hay không.*

  Đầu vào: Số nguyên dương m.

  Đầu ra: **true** nếu m là số nguyên tố, **false** nếu ngược lại.

```
function Nguyen_to(m);
begin
    i:=2;
    while (i <= √m ) and (m mod i = 0) do i:=i+1;
    Nguyen_to := i > √m ;
end;
```

- *Thuật toán tìm số nguyên tố lớn hơn số nguyên dương n.*

  Thuật toán này sử dụng thuật toán Nguyen_to ở trên như thủ tục con.

  Đầu vào: Số nguyên dương n.

  Đầu ra : m - số nguyên tố lớn hơn n.

```
procedure Lagre_Prime(n);
begin
```

```
    m:=n+1;
    while not Nguyen_to(m) do m:=m+1;
end;
```

Do tập các số nguyên tố là vô hạn, nên thuật toán Lagre_Prime là hữu hạn.

4.2.3. Độ phức tạp của thuật toán.

Một chương trình máy tính, mặc dù được cài đặt theo một thuật toán đúng, có thể không cho kết quả mong muốn đối với một bộ dữ liệu nào đó vì hoặc là nó đòi hỏi quá nhiều thời gian, hoặc là không có đủ bộ nhớ để lưu giữ dữ liệu và các biến của chương trình. Vì vậy, để có thể đánh giá khả năng ứng dụng của chương trình ta cần phải phân tích hiệu quả của thuật toán. *Phân tích thuật toán* là quá trình tìm ra những đánh giá về *thời gian tính* cũng như *dung lượng bộ nhớ* cần thiết để thực hiện thuật toán. *Độ phức tạp tính toán* của một thuật toán là lượng thời gian và bộ nhớ cần thiết để thực hiện thuật toán. Trong mục này ta quan tâm đến việc đánh giá thời gian cần thiết để thực hiện thuật toán (ta sẽ gọi là *thời gian tính* của thuật toán).

Rõ ràng, thời gian tính của một thuật toán là hàm của dữ liệu đầu vào. Thông thường khó có thể xây dựng công thức dưới dạng hiện cho hàm này, vì thế ta đặt vấn đề đơn giản hơn. Thay vì làm việc với dữ liệu đầu vào, ta sẽ làm việc với một đặc trưng quan trọng của dữ liệu đầu vào, đó là *kích thước* của nó. Chúng ta sẽ quan tâm đến

- Thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n. Thời gian như vậy sẽ được gọi là *thời gian tính tốt nhất* của thuật toán với đầu vào kích thước n.
- Thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n. Thời gian như vậy sẽ được gọi là *thời gian tính tồi nhất* của thuật toán với đầu vào kích thước n.
- Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước n. Thời gian như vậy sẽ được gọi là *thời gian tính trung bình* của thuật toán.

Để tính toán thời gian tính của thuật toán ta sẽ đếm *số câu lệnh* mà nó phải thực hiện, hoặc trong một số trường hợp có thể đếm cụ thể số phép tính số học, so sánh, gán,.... mà thuật toán đòi hỏi thực hiện. Rõ ràng từ thông số này ta có thể tính được thời gian thực sự mà thuật toán đòi hỏi nếu như nó được cài đặt trên một ngôn ngữ lập trình và chạy trên một máy tính cụ thể. Mặt khác thông số này không phụ thuộc vào người lập trình và ngôn ngữ lập trình được chọn để cài đặt thuật toán cũng như máy tính mà trên đó nó được thực hiện. Vì thế nó là tiêu chuẩn khách quan để đánh giá hiệu quả của thuật toán.

Thí dụ 5. Xét thuật toán tìm số lớn nhất trong dãy hữu hạn số ở thí dụ 3. Trong thuật toán này, số lượng phần tử của dãy số n là đại lượng hợp lý nhất có thể dùng để đánh giá kích thước đầu vào. Rõ ràng vòng lặp trong thuật toán luôn thực hiện đúng n-1 lần nên tất cả thời gian tính tốt nhất, tồi nhất cũng như trung bình của thuật toán đều bằng n-1.

Thông thường, trong các ứng dụng thực tế thời gian chính xác mà thuật toán đòi hỏi để thực hiện nó được quan tâm ít hơn so với việc xác định tốc độ tăng của thông số này khi tăng kích thước của đầu vào. Thí dụ, giả sử thời gian tính tồi nhất của một thuật toán là

$$t(n) = 60\ n^2 + 9n + 9$$

với đầu vào kích thước n. Khi n lớn số hạng $60n^2$ xấp xỉ bằng $t(n)$ (xem bảng 1). Trong trường hợp này $t(n)$ có tốc độ tăng giống như $60n^2$.

n	$t(n) = 60n^2 + 9n + 9$	$60n^2$
10	9099	6000
100	600909	600000
1000	60009009	60000000
10000	6000090009	6000000000

Nếu như $t(n)$ được tính bằng giây, thì

$$T(n) = n^2 + 0.15\ n + 0.15$$

sẽ cho ta thời gian tính đo bằng phút của thuật toán. Rõ ràng sự thay đổi này không ảnh hưởng đến tốc độ tăng của thời gian tính khi kích thước đầu vào n tăng, mà chỉ thay đổi đơn vị tính thời gian. Như vậy khi mô tả tốc độ tăng của thời gian tính của thuật toán khi kích thước đầu vào tăng, không những chúng ta chỉ cần quan tâm đến số hạng trội ($60\ n^2$), mà có thể bỏ qua các hằng số. Với giả thiết như vậy, thời gian tính $t(n)$ tăng giống như n^2 khi n tăng. Ta sẽ nói $t(n)$ có bậc là n^2 và viết

$$t(n) = \Theta(n^2).$$

Tư tưởng cơ bản ở đây là thay thế biểu thức $t(n) = 60n^2 + 9n + 9$ bởi biểu đơn giản hơn n^2 có cùng tốc độ tăng với $t(n)$. Ta đi đến định nghĩa sau

Định nghĩa. *Giả sử f và g là các hàm đối số nguyên dương. Ta viết*

$$f(n) = O(g(n))$$

và nói $f(n)$ có bậc không quá $g(n)$ nếu tồn tại hằng số dương C_1 và số nguyên dương N_1 sao cho

$$|f(n)| \leq C_1 |g(n)|$$

với mọi $n \geq N_1$.

Ta viết

$$f(n) = \Omega(g(n))$$

và nói $f(n)$ có bậc ít nhất là $g(n)$ nếu tồn tại hằng số dương C_2 và số nguyên dương N_2 sao cho

$$|f(n)| \leq C_2 |g(n)|$$

với mọi $n \geq N_2$.

Ta viết

$$f(n) = \Theta(g(n))$$

và nói $f(n)$ có bậc là $g(n)$ nếu $f(n) = O(g(n))$ và $f(n) = \Omega(g(n))$.

Định nghĩa trên có thể phát biểu bằng lời như sau: $f(n) = O(g(n))$, nếu ngoại trừ hằng số và một số hữu hạn ngoại lệ f bị chặn dưới bởi g. $f(n) = \Omega(g(n))$, nếu ngoại trừ hằng số và một số hữu hạn ngoại lệ f bị chặn trên bởi g. $f(n) = \Theta(g(n))$, nếu ngoại trừ hằng số và một số hữu hạn ngoại lệ f bị chặn dưới và chặn trên bởi g.

Các biểu thức $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$ và $f(n) = \Theta(g(n))$ thường được gọi là ký hiệu ô lớn, ômêga, têta đối với hàm f.

Thí dụ 5. Do

$$60n^2 + 9n + 9 \leq 60\, n^2 + 9n^2 + n^2 = 70\, n^2 \text{ với mọi } n \geq 1,$$

chọn $C_1 = 70$ trong định nghĩa trên ta có

$$60n^2 + 9n + 9 = O(n^2).$$

Do

$$60n^2 + 9n + 9 \geq 60\, n^2 \text{ với mọi } n \geq 1,$$

chọn $C_2 = 70$ trong định nghĩa trên ta có

$$60n^2 + 9n + 9 = \Omega(n^2).$$

Do $60n^2 + 9n + 9 = O(n^2)$ và $60n^2 + 9n + 9 = \Omega(n^2)$ nên

$$60n^2 + 9n + 9 = \Theta(n^2).$$

Phương pháp chứng minh trong thí dụ 5 có thể sử dụng để chỉ ra rằng mọi đa thức bậc k với hệ số dương:

$$P_k(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$$

có bậc là $\Theta(n^k)$: $P_k(n) = \Theta(n^k)$.

Thí dụ 6. Giả sử k là số nguyên dương. Ta có

$$1^k + 2^k + \ldots + n^k \leq n^k + n^k + \ldots + n^k = n.\, n^k = n^{k+1}$$

khi $n \geq 1$; suy ra

$$1^k + 2^k + \ldots + n^k = O(n^{k+1}).$$

Mặt khác, vì

$$1^k + 2^k + \ldots + n^k \geq [n/2]^k + \ldots + (n-1)^k + n^k$$

$$\geq [n/2]^k + \ldots + [n/2]^k$$
$$\geq (n/2).(n/2)^k = n^{k+1}/2^{k+1}$$

nên

$$1^k + 2^k + \ldots + n^k = \Omega(n^{k+1}),$$

do đó

$$1^k + 2^k + \ldots + n^k = \Theta(n^{k+1}).$$

Thí dụ 7. Ta sẽ chứng minh

$$\lg n! = \Theta(n \lg n).$$

Ta có

$$\lg n! = \lg n + \lg (n-1) + \ldots + \lg 2 + \lg 1.$$

Do hàm lg là hàm tăng, nên

$$\lg n + \lg (n-1) + \ldots + \lg 2 + \lg 1 \leq \lg n + \lg n + \ldots + \lg n = n \lg n.$$

Vì vậy

$$\lg n! = O(n \lg n).$$

Bây giờ, do

$$\lg n + \lg (n-1) + \ldots + \lg 2 + \lg 1 \geq \lg n + \lg (n-1) + \ldots + \lg [n/2]$$
$$\geq \lg [n/2] + \lg [n/2] + \ldots + \lg [n/2]$$
$$\geq (n/2) \lg(n/2) \geq (n \lg n)/4,$$

suy ra

$$\lg n! = \Omega(n \lg n).$$

Vậy

$$\lg n! = \Theta(n \lg n).$$

Bây giờ ta có thể định nghĩa bậc của thời gian tính tốt nhất, tồi nhất, và trung bình của thuật toán như sau

Định nghĩa. *Nếu thuật toán đòi hỏi thời gian tính tốt nhất là $t(n)$ với độ dài đầu vào n và*

$$t(n) = O(g(n))$$

thời gian tính tốt nhất của thuật toán có bậc không quá $g(n)$ hay thời gian tính tốt nhất của thuật toán là $O(g(n))$.

Nếu thuật toán đòi hỏi thời gian tính tồi nhất là $t(n)$ với độ dài đầu vào n và

$$t(n) = O(g(n))$$

thời gian tính tồi nhất của thuật toán có bậc không quá $g(n)$ hay thời gian tính tồi nhất của thuật toán là $O(g(n))$.

Nếu thuật toán đòi hỏi thời gian tính trung bình là $t(n)$ với độ dài đầu vào n và

$$t(n) = O(g(n))$$

thời gian tính trung bình của thuật toán có bậc không quá g(n) hay thời gian tính trung bình của thuật toán là O(g(n)).

Nếu thay O bởi Ω và "*không quá*" bởi "*ít nhất*" trong định nghĩa trên ta thu được định nghĩa bậc ít nhất của thời gian tính tốt nhất, tồi nhất, trung bình của thuật toán. Nếu thời gian tính tốt nhất của thuật toán vừa là $O(g(n))$ vừa là $\Omega(g(n))$, ta sẽ nói thời gian tính tốt nhất của thuật toán là $\Theta(g(n))$. Tương tự như vậy, ta cũng định nghĩa thời gian tính tồi nhất và trung bình của thuật toán là $\Theta(g(n))$.

Thí dụ 8. Đánh giá số lần thực hiện câu lệnh $x:=x+1$ trong đoạn chương trình sau như là hàm của đầu vào n:

```
for i:=1 to n do
    for j:=1 to i do
        x:=x+1;
```

Đầu tiên i được đặt bằng 1, và j sẽ chạy từ 1 đến 1, lệnh $x:=x+1$ được thực hiện 1 lần, tiếp đến i được đặt bằng 2, và j sẽ chạy từ 1 đến 2, lệnh $x:=x+1$ được thực hiện 2 lần,... Do đó tổng số lần thực hiện câu lệnh $x:=x+1$ là

$$1 + 2 + \ldots + n = \Theta(n^2).$$

Thí dụ 9. Đánh giá $t(n)$ - số lần thực hiện câu lệnh $x:=x+1$ với độ dài đầu vào là n trong đoạn chương trình sau:

```
j:=n;
while j >= 1 do
begin
    for i:=1 to j do x := x+1;
    j:=j div 2;
end;
```

Lần thực hiện đầu tiên trong vòng lặp while câu lệnh $x:=x+1$ được thực hiện n lần. Vì vậy, $t(n) = \Omega(n)$. Sau lần thực hiện đầu tiên giá trị của j được thay bởi $[j/2]$, vì thế $j \leq n/2$. Nếu $j \geq 1$, thì câu lệnh $x := x+1$ được thực hiện không quá $n/2$ lần trong lần lặp thứ hai,... Nếu ký hiệu k là số lần thực hiện các lệnh trong thân của câu lệnh while, thì số lần thực hiện câu lệnh $x := x+1$ là không quá

$$n + n/2 + n/4 + \ldots + n/2^{k-1} = n(1-1/2^k)/(1-1/2).$$

Bây giờ, do

$$t(n) \le n(1 - 1/2^k) / (1 - 1/2) = 2n(1 - 1/2^k) \le 2n,$$

nên $t(n) = O(n)$.

Cuối cùng, do ở trên ta đã có $t(n) = \Omega(n)$, nên $t(n) = \Theta(n)$.

Thí dụ 10. Để giải bài toán: Tìm trong dãy số

$$s_1, s_2 \ldots, s_n$$

một phần tử có giá trị bằng *key* cho trước, có thể áp dụng thuật toán sau

Đầu vào: n và dãy số $s_1, s_2, \ldots, s_n$.
Đầu ra: Vị trí phần tử có giá trị *key* hoặc là n+1 nếu không tìm thấy.

```
function Linear_Search(s,n,key);
begin
    i:=0;
    repeat
        i:=i+1;
    until (i>n) or (key = s[i]);
    Linear_Search := i;
end;
```

Cần đánh giá thời gian tính tốt nhất, tồi nhất, trung bình của thuật toán với độ dài đầu vào là n. Rõ ràng thời gian tính của thuật toán có thể đánh giá bởi số lần thực hiện câu lệnh $i:=i+1$ trong vòng lặp repeat.

Nếu $s_1 = key$ thì câu lệnh $i:=i+1$ trong thân vòng lặp repeat thực hiện 1 lần. Do đó thời gian tính tốt nhất của thuật toán là $\Theta(1)$.

Nếu *key* không có mặt trong dãy đã cho, thì câu lệnh $i:= i+1$ thực hiện n lần. Vì thế thời gian tính tồi nhất của thuật toán là $\Theta(n)$.

Cuối cùng, ta tính thời gian tính trung bình của thuật toán. Nếu *key* tìm thấy ở vị trí thứ i của dãy ($key = s_i$) thì câu lệnh $i := i+1$ phải thực hiện i lần ($i = 1, 2, ..., n$), còn nếu *key* không có mặt trong dãy đã cho thì câu lệnh $i := i+1$ phải thực hiện n lần. Từ đó suy ra số lần trung bình phải thực hiện câu lệnh $i := i+1$ là

$$[(1 + 2 + \ldots + n) + n] /(n+1).$$

Ta có

$$[(1 + 2 + \ldots + n) + n] /(n+1) \le (n^2 + n) /(n+1) = n.$$

Vậy thời gian tính trung bình của thuật toán là $O(n)$.
Mặt khác, do

$$[(1 + 2 + \ldots + n) + n] /(n+1) \geq (n^2/4 + n) /(n+1)$$
$$\geq (n^2/4 + n/4) /(n+1) = n/4,$$

nên thời gian tính trung bình của thuật toán là $\Omega(n)$. Vì vậy, thời gian tính trung bình của thuật toán là $\Theta(n)$.

Đối với thuật toán trong thí dụ này, cả thời gian tính tồi nhất lẫn thời gian tính trung bình đều bằng $\Theta(n)$.

Thí dụ 11. Phân tích thuật toán Euclide tìm ước chung lớn nhất của hai số nguyên dương a, b trình bày dưới đây

Thuật toán Euclide.
Đầu vào: a và b là hai số nguyên dương.
Đầu ra: Ước chung lớn nhất của a và b.

```
function Gcd(a,b);
begin
    (* Đổi chỗ nếu cần để đạt được a > b *)
    if a < b then <Đổi chỗ a và b>;
    while b ≠ 0 do
    begin
        Lấy a chia b thu được a = bq + r , 0 ≤ r < b ;
        a:=b;
        b:=r;
    end;
    Gcd:= a;
end;
```

Để đánh giá độ phức tạp của thuật toán, ta sẽ đếm số phép chia phải thực hiện theo thuật toán.

Trước hết ta chứng minh bằng qui nạp mệnh đề sau.

Mệnh đề 1. *Giả sử cặp số a, b, $a > b$, đòi hỏi $n \geq 1$ phép chia trong thuật toán Euclide. Khi đó $a \geq f_{n+1}$, $b \geq f_n$, trong đó $\{f_n\}$ là dãy số Fibonaci (xác định bởi công thức truy hồi $f_n = f_{n-1} + f_{n-2}$, $n \geq 2$, $f_0 = f_1 = 1$).*

Chứng minh. Dễ dàng kiểm tra mệnh đề đúng với n=1. Giả sử mệnh đề đúng với $n \geq 1$. Ta chứng minh khẳng định của mệnh đề đúng với n+1. Giả sử cặp a, b, $a > b$ đòi hỏi n+1 phép chia. Sau khi thực hiện phép chia a cho b:

$$a = b\,q + r, \qquad 0 \leq r < b,$$

thuật toán tiếp tục làm việc với cặp số b và r, $b > r$. Cặp số này đòi hỏi n phép chia. Do đó theo giả thiết qui nạp

$$b \geq f_{n+1}, \qquad r \geq f_n.$$

Từ đó suy ra

$$a = b\,q + r \geq b + r \geq f_{n+1} + f_n = f_{n+2},$$

tức là ta có

$$a \geq f_{n+2}, \qquad b \geq f_{n+1},$$

và theo qui nạp mệnh đề được chứng minh.

Sử dụng mệnh đề vừa chứng minh ta có thể đánh giá được số phép chia nhiều nhất cần phải thực hiện trong thuật toán Euclide.

Mệnh đề 2. *Giả sử các số nguyên ở đầu vào của thuật toán Euclide là không vượt quá m, $m \geq 8$. Khi đó thuật toán đòi hỏi không quá $\log_{3/2}(2m/3)$ phép chia.*

Chứng minh. Thực vậy giả sử n là số phép chia lớn nhất cần thực hiện theo thuật toán với đầu vào thoả mãn điều kiện đặt ra trong mệnh đề. Giả sử cặp số a, b, $m \geq a > b$ đòi hỏi n phép chia. Do $m \geq 8$, nên, bằng cách thử trực tiếp các khả năng, có thể kiểm tra được là $n \geq 4$. Theo khẳng định của mệnh đề 1, ta có $a \geq f_{n+1}$. Suy ra $f_{n+1} \leq m$. Do $n + 1 \geq 5$, nên có thể chứng minh được là

$$(3/2)^{n+1} < f_{n+1}.$$

Vì thế suy ra

$$(3/2)^{n+1} < m,$$

hay là

$$n + 1 < \log_{3/2} m.$$

Suy ra

$$n < \log_{3/2} m - 1 = \log_{3/2} m - \log_{3/2} 3/2 = \log_{3/2}(2m/3).$$

Mệnh đề được chứng minh.

Nếu như các thí dụ 8-10 có thể tạo ra cảm giác là đánh giá độ phức tạp của một thuật toán là một vấn đề giản đơn, thì thí dụ 11 đã cho thấy vấn đề không hẳn là như vậy. Nói chung, phân tích độ phức tạp tính toán cuả một thuật toán là một vấn đề phức tạp, nó đòi hỏi tư duy sáng tạo và có phương pháp và cách tiếp cận riêng. Mặt khác trong rất nhiều trường hợp, ta không thể thu được những đánh giá đẹp đẽ như trong các thí dụ vừa trình bày ở trên.

Dạng đánh giá	*Tên gọi*
$\Theta(1)$	Hằng số
$\Theta(\lg \lg n)$	Log log
$\Theta(\lg n)$	Logarithm
$\Theta(n)$	Tuyến tính
$\Theta(n \lg n)$	n log n
$\Theta(n^2)$	Bậc hai
$\Theta(n^3)$	Bậc ba
$\Theta(n^m)$	Đa thức
$\Theta(m^n)$, $m \geq 2$	Hàm mũ
$\Theta(n!)$	Giai thừa

Bảng 1. Các đánh giá thông dụng.

Các hằng số bị bỏ qua trong cách đánh giá vừa đưa vào ở trên có thể có ý nghĩa quan trọng trong ứng dụng thuật toán. Chẳng hạn, giả sử với mọi dữ liệu đầu vào độ dài n, thuật toán A đòi hỏi thời gian tính là $C_1 n$ còn thuật toán B đòi hỏi $C_2 n^2$. Khi đó với một số kích thước đầu vào nhất định thuật toán B có thể vẫn là tốt hơn. Thí dụ, nếu $C_1 = 300$, $C_2 = 5$, thì với $n = 5$, thuật toán A đòi hỏi thời gian 1500 trong khi thuật toán B đòi hỏi thời gian 125. Tất nhiên, khi n đủ lớn, thuật toán A là luôn nhanh hơn thuật toán B. Một số đánh giá tốc độ tang đặc biệt được đặt tên riêng như chỉ ra trong bảng 1.

Trong bảng 1, các đánh giá được sắp xếp theo thứ tự tăng dần của tốc độ tăng (ngoại trừ trường hợp $\Theta(n^m)$). Bảng 2 dưới đây cho thấy thời gian tính tăng như thế nào với các đánh giá số bước lặp khác nhau (giả thiết: mỗi bước lặp đòi hỏi 1 micro giây):

	Thời gian	tính nếu	$n =$	
Đánh giá	6	12	50	100
1	10^{-6} sec	10^{-6} sec	10^{-6} sec	10^{-6} sec
lg lg n	10^{-6} sec	2×10^{-6} sec	2×10^{-6} sec	3×10^{-6} sec
lg n	3×10^{-6} sec	4×10^{-6} sec	6×10^{-6} sec	7×10^{-6} sec
n	6×10^{-6} sec	10^{-5} sec	5×10^{-5} sec	10^{-4} sec
n lg n	2×10^{-5} sec	4×10^{-5} sec	3×10^{-4} sec	7×10^{-4} sec
n^2	4×10^{-5} sec	10^{-4} sec	3×10^{-3} sec	0.01 sec
n^3	2×10^{-4} sec	2×10^{-3} sec	0.13 sec	1 sec
2^n	6×10^{-5} sec	4×10^{-3} sec	36 năm	4×10^{6} năm

Bảng 2. Thời gian tính với các đánh giá khác nhau

Nếu như đối với một bài toán ta có thể xây dựng thuật toán với thời gian tính tồi nhất là đa thức, thì bài toán đặt ra gọi là *được giải tốt*. Rất tiếc là cho đến thời điểm hiện tại, chỉ có một số không nhiều lắm bài toán là được giải tốt.

Nếu bài toán không có thuật toán với thời gian tồi nhất đa thức để giải thì nó được gọi là *khó giải* (intractable). Nếu như có thuật toán giải bài toán như vậy, thì chắc chắn nó sẽ đòi hỏi rất nhiều thời gian.

Có một số bài toán lại khó đến mức là ta không thể xây dựng thuật toán để giải nó. Những bài toán như vậy được gọi là *không giải được* (unsolvable problem). Một trong những bài toán như vậy là Bài toán về tính dừng: Cho một chương trình và tập đầu vào, hỏi rằng chương trình có dừng hay không?

Việc nghiên cứu lý thuyết về độ phức tạp tính toán của các bài toán và các thuật toán là một trong những vấn đề trung tâm của khoa học tính toán, mà ở giáo trình này ta không có điều kiện để đi sâu hơn nữa.

4.3. Phương pháp sinh

Phương pháp sinh có thể áp dụng để giải bài toán liệt kê tổ hợp đặt ra nếu như hai điều kiện sau được thực hiện:

1) Có thể xác định được một thứ tự trên tập các cấu hình tổ hợp cần liệt kê. Từ đó có thể xác định được cấu hình đầu tiên và cấu hình cuối cùng trong thứ tự đã xác định.

2) Xây dựng được thuật toán từ cấu hình chưa phải là cuối cùng đang có, đưa ra cấu hình kế tiếp nó.

Ta sẽ gọi thuật toán nói trong điều kiện 2) là thuật toán sinh kế tiếp. Rõ ràng là thứ tự trong điều kiện 1) cần được lựa chọn sao cho có thể xây dựng được thuật toán sinh kế tiếp. Giả thiết rằng, hai điều kiện nêu trên đã được thực hiện, khi đó thuật toán sinh để giải bài toán liệt kê đặt ra được mô tả như sau:

```
procedure Generate;
Begin
    <Xây dựng cấu hình ban đầu>;
    Stop:=false;
    while not stop do
```

```
    begin
        <Đưa ra cấu hình đang có>;
        Sinh_kế_tiếp;
    end;
End.
```

trong đó *Sinh_kế_tiếp* là thủ tục sinh cấu hình kế tiếp theo thuật toán sinh kế tiếp đã xây dựng. Nếu cấu hình đang có đã là cuối cùng, thủ tục này cần gán cho biến *Stop* giá trị *true*, ngược lại thủ tục này sẽ xây dựng cấu hình kế tiếp của cấu hình đang có trong thứ tự đã xác định.

Dưới đây ta xét một số thí dụ minh hoạ cho việc áp dụng thuật toán sinh.

Liệt kê tất cả các dãy nhị phân độ dài n.

Viết dãy nhị phân dưới dạng $b_1\, b_2 \ldots b_n$, trong đó $b_i \in \{0, 1\}$. Xem mỗi dãy nhị phân $b = b_1\, b_2 \ldots b_n$ là biểu diễn nhị phân của một số nguyên $p(b)$. Khi đó thứ tự hiển nhiên nhất có thể xác định trên tập các dãy nhị phân là *thứ tự tự nhiên* (còn gọi là *thứ tự từ điển*) được xác định như sau.Ta nói dãy nhị phân $b = b_1\, b_2 \ldots b_n$ *đi trước* dãy nhị phân $b' = b'_1\, b'_2 \ldots b'_n$ trong thứ tự tự nhiên và ký hiệu là $b \prec b'$ nếu $p(b) < p(b')$.

Thí dụ: Khi n=3, các dãy nhị phân độ dài 3 được liệt kê theo thứ tự tự nhiên như sau

b	$p(b)$
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

Như thế dãy đầu tiên sẽ là 0 0 ... 0, còn dãy cuối cùng là 1 1 ... 1. Giả sử $b_1\, b_2 \ldots b_n$ là dãy đang có. Nhận xét rằng, nếu dãy này gồm toàn chữ số 1 thì quá trình liệt kê kết thúc, trái lại, dãy kế tiếp sẽ nhận được bằng cách cộng thêm 1 (theo modun 2, có nhớ) vào dãy hiện tại. Từ đó ta nhận được qui tắc sinh dãy kế tiếp như sau:

- Tìm i đầu tiên (theo thứ tự $i = n, n\text{-}1, \ldots, 1$) thoả mãn $b_i = 0$.
- Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là dãy cần tìm.

Thí dụ. Xét dãy nhị phân độ dài 10: b = 1101011111. Ta có i = 5. Do đó, đặt b_5 = 1, và b_i = 0, i = 6, 7, 8, 9, 10, ta thu được xâu nhị phân kế tiếp là 1101100000.

Thuật toán sinh kế tiếp được mô tả trong thủ tục sau

```
procedure Next_Bit_String;
(*  Sinh xâu nhị phân kế tiếp theo thứ tự từ điển từ
    xâu đang có  b1 b2 ... bn ≠ 1 1 ... 1            *)
begin
    i:=n;
    while bi = 1 do
    begin
        bi = 0;
        i:=i-1;
    end;
    bi := 1;
end;
```

Dưới đây là chương trình PASCAL thực hiện việc liệt kê các dãy nhị phân độ dài n bằng phương pháp sinh. Trong chương trình có thêm một biến *count* dùng để đếm số lượng dãy nhị phân được sinh bởi chương trình.

```
{Chương trình liệt kê dãy nhị phân}
var
    n, i: integer;
    b: array[1..20] of 0..1;
    count: word;
    stop: boolean;

    procedure Init;
    var i: integer;
    begin
        write('Cho biết độ dài dãy nhị phân: '); readln(n);
        for i := 1 to n do b[i] := 0;
        stop := false;
        count := 0;
    end;

    procedure Next_Bit_String;
```

```
var i: integer;
begin
    {sinh dãy nhị phân kế tiếp}
    i := n;
    while (i >= 1) and (b[i] = 1) do
    begin
        b[i] := 0;
        i := i - 1;
    end;
    if  i <1 then stop := true
    else b[i] := 1;
end;

Begin    {main program}
    Init;
    while not stop do
    begin
        {đưa ra dãy nhị phân hiện tại}
        count := count+1;
         write(count:5, '. ');
        for i := 1 to n  do write(b[i]:2); writeln;
        Next_Bit_String
    end;
    write('Gõ Enter để kết thúc... '); readln;
End.
```

Liệt kê các tập con m phần tử của tập n phần tử.

Bài toán đặt ra là: Cho $X = \{1, 2, \ldots, n\}$. Hãy liệt kê các tập con m phần tử của X. Mỗi tập con m phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm m thành phần

$$a = (a_1, a_2, \ldots, a_m)$$

thoả mãn

$$1 \le a_1 < a_2 < \ldots < a_m \le n.$$

Trên tập các tập con m phần tử của X có thể xác định nhiều thứ tự khác nhau. Thứ tự đơn giản nhất là *thứ tự từ điển* được định nghĩa như sau: Ta nói tập con $a = (a_1, a_2, \ldots, a_m)$ đi trước tập con $a' = (a'_1, a'_2, \ldots, a'_m)$ trong thứ tự từ điển và ký hiệu là $a \prec a'$, nếu tìm được chỉ số k ($1 \le k \le m$) sao cho

$$a_1 = a'_1, a_2 = a'_2, \ldots, a_{k-1} = a'_{k-1}, a_k < a'_k.$$

Thí dụ: $X = \{1, 2, 3, 4, 5\}$, $m = 3$. Các tập con 3 phần tử của X được liệt kê theo thứ tự từ điển như sau

1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5

Như vậy, tập con đầu tiên trong thứ tự từ điển là

$(1, 2, \ldots, m)$

và tập con cuối cùng là

$(n\text{-}m+1, n\text{-}m+2, \ldots, n)$.

Giả sử $a=(a_1, a_2, \ldots, a_m)$ là tập con đang có chưa phải cuối cùng, khi đó có thể chứng minh được rằng, tập con kế tiếp trong thứ tự từ điển có thể xây dựng bằng cách thực hiện các quy tắc biến đổi sau đối với tập đang có:

- Tìm từ bên phải dãy $a_1, a_2, \ldots, a_m$ phần tử $a_i \neq n\text{-}m+i$,
- Thay a_i bởi $a_i + 1$;
- Thay a_j bởi $a_i + j - i$, với $j = i+1, i+2, \ldots, m$.

Thí dụ: $n = 6$, $m = 4$. Giả sử đang có tập con (1, 2, 5, 6), cần xây dựng tập con kế tiếp nó trong thứ tự từ điển. Ta có $i=2$, thay $a_2 = 3$, và $a_3 = 4$, $a_4 = 5$, ta được tập con kế tiếp (1, 3, 4, 5).

Từ đó, thuật toán sinh kế tiếp có thể mô tả trong thủ tục PASCAL mô phỏng sau đây:

```
procedure Next_Combination;
(*  Sinh m-tập  con kế tiếp theo thứ tự từ điển
    của tập con  (a1, a2,..., am)  ≠ {n-m+1,...,n } *)
begin
    i:=m;
```

```
    while a_i = n-m+i do i:=i-1;
    a_i:= a_i + 1;
    for j:=i+1 to m do a_j := a_i + j - i;
end;
```

Liệt kê các hoán vị của tập n phần tử.

Bài toán đặt ra là: Cho $X = \{1, 2, \ldots, n\}$. Hãy liệt kê các hoán vị từ n phần tử của X. Mỗi hoán vị từ n phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm n thành phần $a = (a_1, a_2, \ldots, a_n)$ thoả mãn

$$a_i \in X,\ i = 1, 2, \ldots, n,\ a_p \neq a_q,\ p \neq q.$$

Trên tập các hoán vị từ n phần tử của X có thể xác định nhiều thứ tự khác nhau. Thứ tự đơn giản nhất là *thứ tự từ điển* được định nghĩa như sau: Ta nói hoán vị $a = (a_1, a_2, \ldots, a_n)$ đi trước hoán vị $a' = (a'_1, a'_2, \ldots, a'_n)$ trong thứ tự từ điển và ký hiệu là $a \prec a'$, nếu tìm được chỉ số k $(1 \leq k \leq n)$ sao cho

$$a_1 = a'_1, a_2 = a'_2, \ldots, a_{k-1} = a'_{k-1},\ a_k < a'_k.$$

Thí dụ: $X = \{1, 2, 3\}$. Các hoán vị từ 3 phần tử của X được liệt kê theo thứ tự từ điển như sau

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Như vậy, hoán vị đầu tiên trong thứ tự từ điển là

$(1, 2, \ldots, n)$

và hoán vị cuối cùng là

$(n, n-1, \ldots, 1)$.

Giả sử $a = (a_1, a_2, \ldots, a_n)$ là hoán vị đang có chưa phải cuối cùng, khi đó có thể chứng minh được rằng, hoán vị kế tiếp trong thứ tự từ điển có thể xây dựng bằng cách thực hiện các quy tắc biến đổi sau đối với hoán vị đang có:

- Tìm từ phải qua trái hoán vị đang có chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$ (nói cách khác: j là chỉ số lớn nhất thoả mãn $a_j < a_{j+1}$);
- Tìm a_k là số nhỏ nhất còn lớn hơn a_j trong các số ở bên phải a_j;
- Đổi chỗ a_j với a_k;
- Lật ngược đoạn từ a_{j+1} đến a_n.

Thí dụ: Giả sử đang có hoán vị (3, 6, 2, 5, 4. 1), cần xây dựng tập con kế tiếp nó trong thứ tự từ điển. Ta có chỉ số $j = 3$ ($a_3 = 2 < a_4 = 5$). Số nhỏ nhất còn lớn hơn a_3 trong các số bên phải của a_3 là $a_5 = 4$. Đổi chỗ a_3 với a_5 ta thu được (3, 6, 4, 5, 2, 1), và cuối cùng, lật ngược thứ tự đoạn $a_4\, a_5\, a_6$ ta thu được hoán vị kế tiếp (3, 6, 4, 1, 2, 5).

Từ đó, thuật toán sinh kế tiếp có thể mô tả trong thủ tục PASCAL mô phỏng sau đây:

```
procedure Next_Permutation;
(*   Sinh hoán vị kế tiếp theo thứ tự từ điển
     của hoán vị (a1, a2, ..., an) ≠(n, n-1, ..., 1)   *)
begin
     (* Tìm j là chỉ số lớn nhất thoả aj < aj+1 *)
     j:=n-1;
     while aj > aj+1 do j:=j-1;
     (* Tìm ak là số nhỏ nhất còn lớn hơn aj ở bên phải aj *)
     k:=n;
     while aj > ak do k:=k-1;
     Swap(aj, ak); (* đổi chỗ aj với ak *)
     (* Lật ngược đoạn từ aj+1 đến an *)
     r:=n;
     s:=j+1;
     while r>s do
     begin
          Swap(ar, as); (* đổi chỗ ar với as *)
          r:=r-1;
          s:= s+1;
     end;
end;
```

Nói chung, phương pháp sinh không có tính phổ dụng: không phải cấu hình kế tiếp nào cũng được sinh một cách đơn giản từ cấu hình hiện tại, mặt khác cấu hình ban đầu không phải dễ tìm vì ngay cả sự tồn tại một cấu hình nhiều khi vẫn còn là nghi vấn. Vì vậy, thông thường thuật toán sinh chỉ có thể xây dựng được đối với những bài toán liệt kê tổ hợp đơn giản. Để giải những bài toán liệt kê phức tạp, người ta thường dùng thuật toán được trình bày trong mục tiếp theo, có tính phổ dụng cao hơn. Đó là thuật toán quay lui.

4.4. Thuật toán quay lui

Nội dung chính của thuật toán này là việc xây dựng dần các thành phần của cấu hình bằng cách thử tất cả các khả năng. Giả thiết cấu hình cần tìm được mô tả bằng một bộ gồm n thành phần $x_1, x_2, ..., x_n$. Giả sử đã xác định được i-1 thành phần $x_1, x_2, ..., x_{i-1}$ (mà ta sẽ gọi là *lời giải bộ phận cấp* i-1), bây giờ ta xác định thành phần x_i bằng cách duyệt tất cả các khả năng có thể đề cử cho nó (đánh số các khả năng từ 1 đến n_i). Với mỗi khả năng j, kiểm tra xem j có chấp nhận được không. Xảy ra 2 trường hợp:

. nếu chấp nhận j thì xác định x_i theo j, sau đó nếu $i = n$ thì ta được một cấu hình, còn trái lại ta tiến hành việc xác định x_{i+1}.

. nếu thử tất cả các khả năng mà không có khả năng nào được chấp nhận thì quay lại bước trước để xác định lại x_{i-1}.

Điểm quan trọng của thuật toán là phải ghi nhớ tại mỗi bước đã đi qua, những khả năng nào đã thử để tránh trùng lặp. Rõ ràng những thông tin này cần được lưu trữ theo cơ cấu ngăn xếp (*stack - vào sau ra trước*). Vì thế thuật toán này rất phù hợp với việc lập trình trên một ngôn ngữ cho phép gọi đệ qui. Bước xác định x_i có thể diễn tả qua thủ tục được tổ chức đệ qui dưới đây:

```
procedure Try(i: integer);
var j: integer;
begin
    for j := 1 to n_i do
    if <chấp nhận j> then
    begin
        <xác định x_i theo j>
        if i = n then <ghi nhận một cấu hình>
        else Try(i+1);
    end;
end;
```

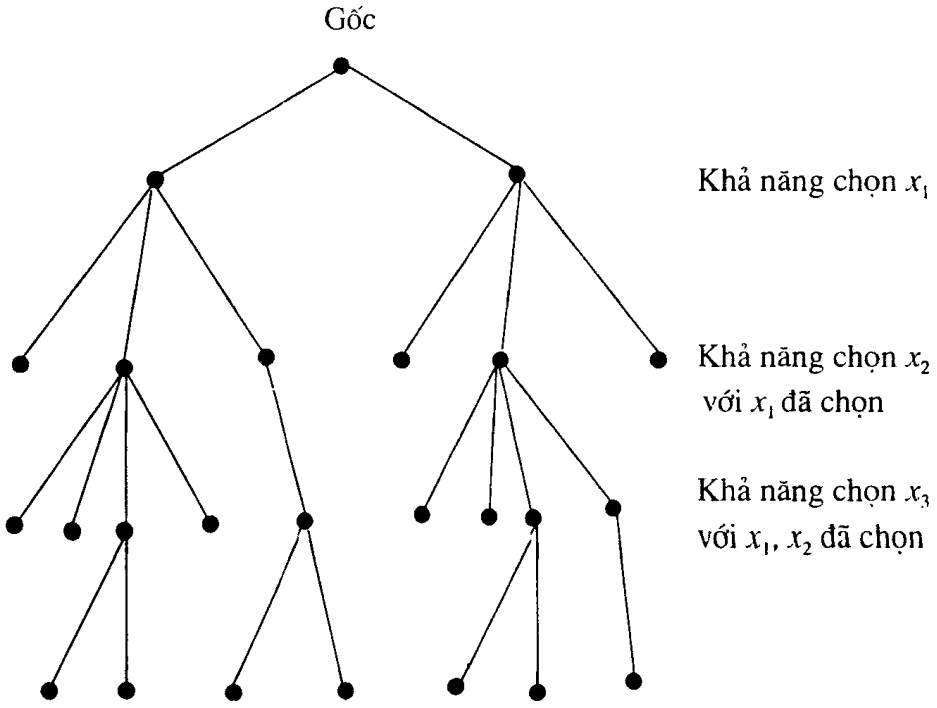
Phần quan trọng nhất trong thủ tục trên là việc đưa ra được một danh sách các khả năng đề cử và việc xác định giá trị của biểu thức logic <*chấp nhận j*>. Thông thường giá trị này, ngoài việc phụ thuộc j, còn phụ thuộc vào việc đã chọn các khả năng tại i-1 bước trước. Trong những trường hợp như vậy, cần ghi nhớ *trạng thái mới* của quá trình tìm kiếm sau khi <*xác định x_i theo j*> và trả lại *trạng thái cũ* sau lời gọi *Try(i+1)*. Các trạng thái này được ghi nhận nhờ một số biến tổng thể (global), gọi là các *biến trạng thái*.

Sau khi xây dựng thủ tục đệ qui *Try*, đoạn chương trình chính giải bài toán liệt kê có dạng:

```
Begin
    Init; Try(1);
End.
```

trong đó *Init* là thủ tục khởi tạo các giá trị ban đầu (nhập các giá trị tham số của bài toán, khởi gán các biến trạng thái, biến đếm, ...).

Quá trình tìm kiếm lời giải theo thuật toán quay lui có thể mô tả bởi Cây tìm kiếm lời giải sau đây



Hình 1. Cây liệt kê lời giải theo thuật toán quay lui

Dưới đây là 3 thí dụ liệt kê 3 cấu hình cơ bản: dãy nhị phân, hoán vị và tổ hợp, bằng thuật toán quay lui.

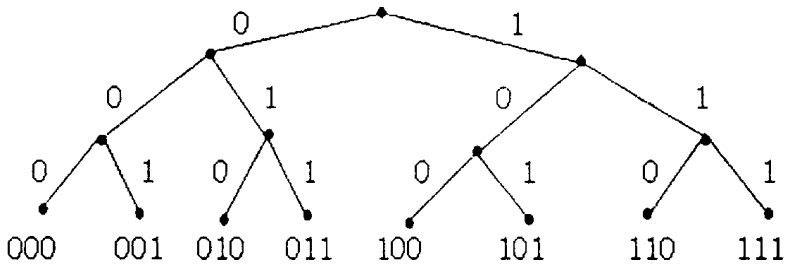
Thí dụ 1. Liệt kê các dãy nhị phân độ dài n.

Giải: Biểu diễn dãy nhị phân dưới dạng $b_1\ b_2\ ...\ b_n$, trong đó $b_i \in \{0,1\}$ Thủ tục đệ qui *Try(i)* xác định b_i, trong đó các giá trị đề cử là 0 và 1. Các giá trị này mặc nhiên được chấp nhận mà không phải thoả mãn điều kiện gì (vì thế bài toán không cần biến trạng

thái). Thủ tục *Init* nhập giá trị *n* và khởi gán biến đếm *count*. Thủ tục *Result* đưa ra dãy tìm được.

```
{Chương trình liệt kê dãy nhị phân}
var n: integer;
    b: array[1..20] of 0..1;
    count: word;
    procedure Init;
    begin
        write('n = '); readln(n);
        count := 0;
    end;

    procedure Result;
    var i: integer;
    begin
        count := count+1;
        write(count:5, '. ');
        for i := 1 to n do write(b[i]:2);
        writeln;
    end;
    procedure Try(i: integer);
    var j: integer;
    begin
        for j := 0 to 1 do
        begin
            b[i] := j;
            if i = n then Result
           else Try(i+1);
        end;
    end;
Begin   {main program}
    Init; Try(1);
    write('Gõ Enter để kết thúc... '); readln;
End.
```

Cây liệt kê lời giải theo thuật toán quay lui để liệt kê các dãy nhị phân độ dài 3 được mô tả trong hình 2.



Hình 2. Cây liệt kê dãy nhị phân độ dài 3

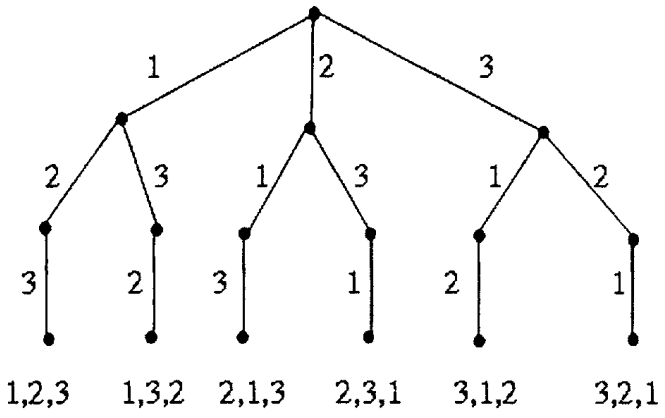
Thí dụ 2. Liệt kê các hoán vị của $\{1, 2, ..., n\}$.

Giải: Biểu diễn hoán vị dưới dạng $p_1 p_2 ... p_n$, trong đó p_i nhận giá trị từ 1 đến n và $p_i \neq p_j$ với $i \neq j$. Các giá trị từ 1 đến n được lần lượt đề cử cho p_i, trong đó giá trị j được chấp nhận nếu nó chưa được dùng. Vì thế cần phải ghi nhớ đối với mỗi giá trị j xem nó đã được dùng hay chưa. Điều này được thực hiện nhờ một dãy biến lôgic b_j, trong đó b_j bằng *true* nếu j chưa được dùng. Các biến này cần phải được khởi gán giá trị *true* trong thủ tục *Init*. Sau khi gán j cho p_i cần ghi nhận *false* cho b_j và phải gán lại *true* khi thực hiện xong *Result* hay *Try(i+1)*. Các phần còn lại giống như bài toán trước.

```
{Chương trình liệt kê các hoán vị}
var
    n: integer;
    p: array[1..20] of integer;
    b: array[1..20] of boolean;
    count: word;

    procedure Init;
    var i: integer;
    begin
        write('n = '); readln(n);
        for i := 1 to n do b[i] := true;
        count := 0;
    end;

    procedure Result;
    var i: integer;
    begin
        count := count+1;
        write(count:5, '. ');
```

```
        for i := 1 to n do write(p[i]:3);
        writeln;
    end;

    procedure Try(i: integer);
    var j: integer;
    begin
        for j := 1 to n do
        if b[j] then     {chấp nhận j}
        begin
            p[i] := j;
            b[j] := false; {ghi nhận trạng thái mới}
            if i = n then Result else Try(i+1);
            b[j] := true;  {trả lại trạng thái cũ}
        end;
    end;

Begin   {main program}
    Init; Try(1);
    write('Gõ Enter để kết thúc... '); readln;
End.
```

Cây liệt kê các hoán vị của 1, 2, 3 được cho trong hình 3. Các hoán vị được liệt kê theo thứ tự từ điển tăng dần.



Hình 3. Cây liệt kê các hoán vị của 1, 2, 3.

Thí dụ 3. Liệt kê các tổ hợp chập m của $\{1, 2, ..., n\}$.

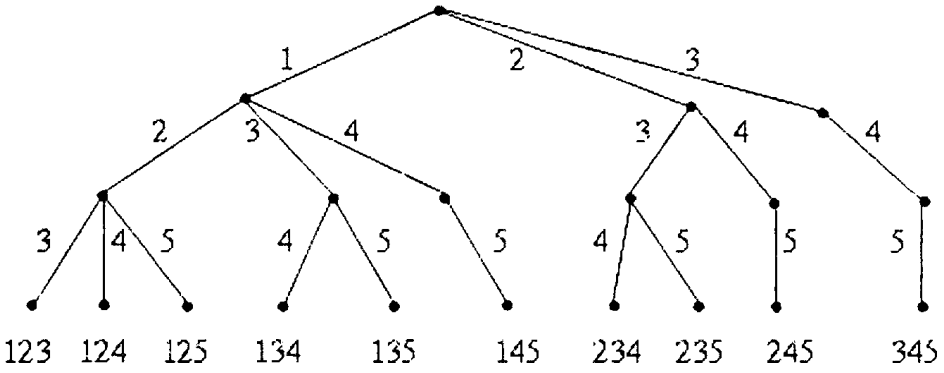
Giải: Biểu diễn tổ hợp dưới dạng $c_1\ c_2\ ...\ c_m$, trong đó

$$1 \le c_1 < c_2 < ... < c_m \le n$$

Từ đó suy ra các giá trị đề cử cho c_i là từ $c_{i-1}+1$ đến $n-m+i$. Để điều này đúng cho cả trường hợp $i = 1$, cần thêm vào c_0 với $c_0 = 0$. Các giá trị đề cử này mặc nhiên được chấp nhận. Các thủ tục *Init*, *Result* được xây dựng giống như bài toán trước.

```
{Chương trình liệt kê các tổ hợp}
var
    n, m: integer;
    c: array[0..20] of integer;
    count: word;
    procedure Init;
    begin
        write('n, m = '); readln(n, m);
        c[0] := 0; count := 0;
    end;

    procedure Result;
    var i: integer;
    begin
        count := count+1; write(count:5, '. ');
        for i := 1 to m do write(c[i]:3); writeln;
    end;

    procedure Try(i: integer);
    var j: integer;
    begin
        for j := c[i-1]+1 to n-m+i do
        begin
            c[i] := j;
            if i = m then Result else Try(i+1);
        end;
    end;

Begin   {main program}
    Init;
    Try(1);
    write('Gõ Enter để kết thúc... '); readln;
End.
```

Cây liệt kê các tổ hợp chập 3 từ {1, 2, 3, 4, 5} được cho trong hình 4. Dễ thấy các tổ hợp được liệt kê theo thứ tự từ điển tăng dần.



Hình 4. Cây liệt kê tổ hợp chập 3 từ {1, 2, 3, 4, 5}

Trong nhiều bài toán, việc xác định điều kiện <*chấp nhận j*> không phải là đơn giản, nó đòi hỏi phải có một trình độ tổ chức nhất định. Rõ ràng độ phức tạp của bài toán phụ thuộc rất nhiều vào độ phức tạp của điều kiện này. Nói chung, người ta cố gắng thu hẹp diện đề cử các khả năng j và tinh giản tối đa việc tính điều kiện chấp nhận j. Đây cũng là cách chủ yếu để nâng cao tính khả thi của bài toán liệt kê.

Dưới đây trình bày bài toán Xếp Hậu, là một bài toán kinh điển để minh hoạ thuật toán quay lui.

Thí dụ 4. *Bài toán Xếp Hậu.* Liệt kê tất cả các cách xếp n quân Hậu trên bàn cờ $n \times n$ sao cho chúng không ăn được lẫn nhau.

Giải: Đánh số cột và dòng của bàn cờ từ 1 đến n. Mỗi dòng được xếp đúng một quân Hậu. Vấn đề còn lại là xem mỗi quân Hậu được xếp vào cột nào. Từ đó dẫn đến việc biểu diễn một cách xếp bằng bộ n thành phần $x_1\ x_2\ ...\ x_n$, trong đó $x_i = j$ nghĩa là quân hậu dòng i được xếp vào cột j. Các giá trị đề cử cho x_i là từ 1 đến n. Giá trị j là được chấp nhận nếu ô (i, j) chưa bị các quân Hậu trước chiếu đến. Để kiểm soát được điều này, ta cần phải ghi nhận trạng thái của bàn cờ trước cũng như sau khi xếp được một quân Hậu. Để ý rằng, theo luật cờ, quân Hậu ăn ngang, dọc và hai đường chéo.

Việc kiểm soát theo chiều ngang là không cần thiết vì mỗi dòng được xếp đúng một quân Hậu. Việc kiểm soát theo chiều dọc được ghi nhận nhờ dãy biến lôgic a_j với qui ước a_j bằng *true* nếu cột j còn trống. Đối với 2 đường chéo ta nhận xét rằng một đường có phương trình $i+j = const$, còn đường kia $i-j = const$ $(2 \le i+j \le 2n,\ 1-n \le i-j \le n-1)$, từ đó đường chéo thứ nhất được ghi nhận nhờ dãy biến lôgic b_j $(2 \le j \le 2n)$ và đường chéo thứ hai, nhờ dãy biến lôgic c_j $(1-n \le j \le n-1)$ với qui ước các đường này còn trống nếu biến tương ứng có giá trị *true*. Các biến trạng thái a_j, b_j, c_j cần được khởi gán giá trị *true* trong thủ tục *Init*. Như vậy giá trị j được chấp nhận khi và chỉ khi cả 3 biến

a_j, b_{i+j}, c_{i-j} cùng có giá trị *true*. Các biến này cần gán lại *false* khi xếp xong quân Hậu thứ i và trả lại *true* sau khi gọi *Result* hay $Try(i+1)$. Các phần khác được giải quyết như các thí dụ trước.

```
{Chương trình giải bài toán Xếp Hậu}
var
    n: integer;
    x: array[1..20] of integer;
    a: array[1..20] of boolean;
    b: array[2..40] of boolean;
    c: array[-19..19] of boolean;
    count: word;

    procedure Init;
    var i: integer;
    begin
        write('n = '); readln(n);
        count := 0;
        for i := 1 to n do a[i] := true;
        for i := 2 to 2*n do b[i] := true;
        for i := 1-n to n-1 do c[i] := true;
    end;

    procedure Result;
    var i: integer;
    begin
        count := count+1;
        write(count:5, '. ');
        for i := 1 to n do write(x[i]:3);
        writeln;
    end;

    procedure Try(i: integer);
    var j: integer;
    begin
        for j := 1 to n do
        if a[j] and b[i+j] and c[i-j] then
        begin {chấp nhận j}
            x[i] := j;
```

```
            {ghi nhận trạng thái mới}
            a[j] := false; b[i+j] := false; c[i-j] := false;
            if i = n then Result else Try(i+1);
            {trả lại trạng thái cũ}
            a[j] := true; b[i+j] := true; c[i-j] := true;
        end;
    end;

Begin   {main program}
    Init;
    Try(1);
    write('Gõ Enter để kết thúc... '); readln;
End.
```

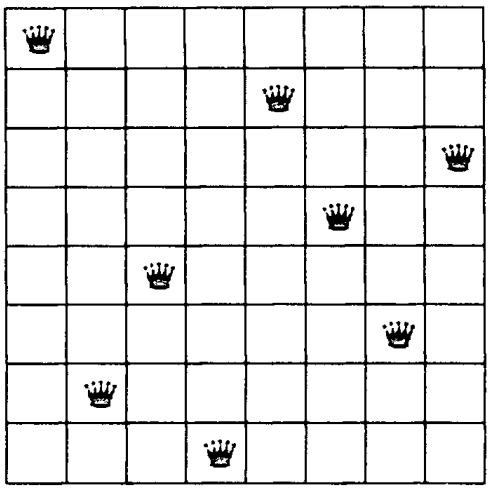
Dưới đây là số cách xếp Hậu ứng với một số giá trị n:

n	4	7	8	9	10	11	12	13	14
H_n	2	40	92	352	724	2680	14200	73712	365596

Nghiệm đầu tiên mà chương trình tìm được ứng với $n = 8$ là

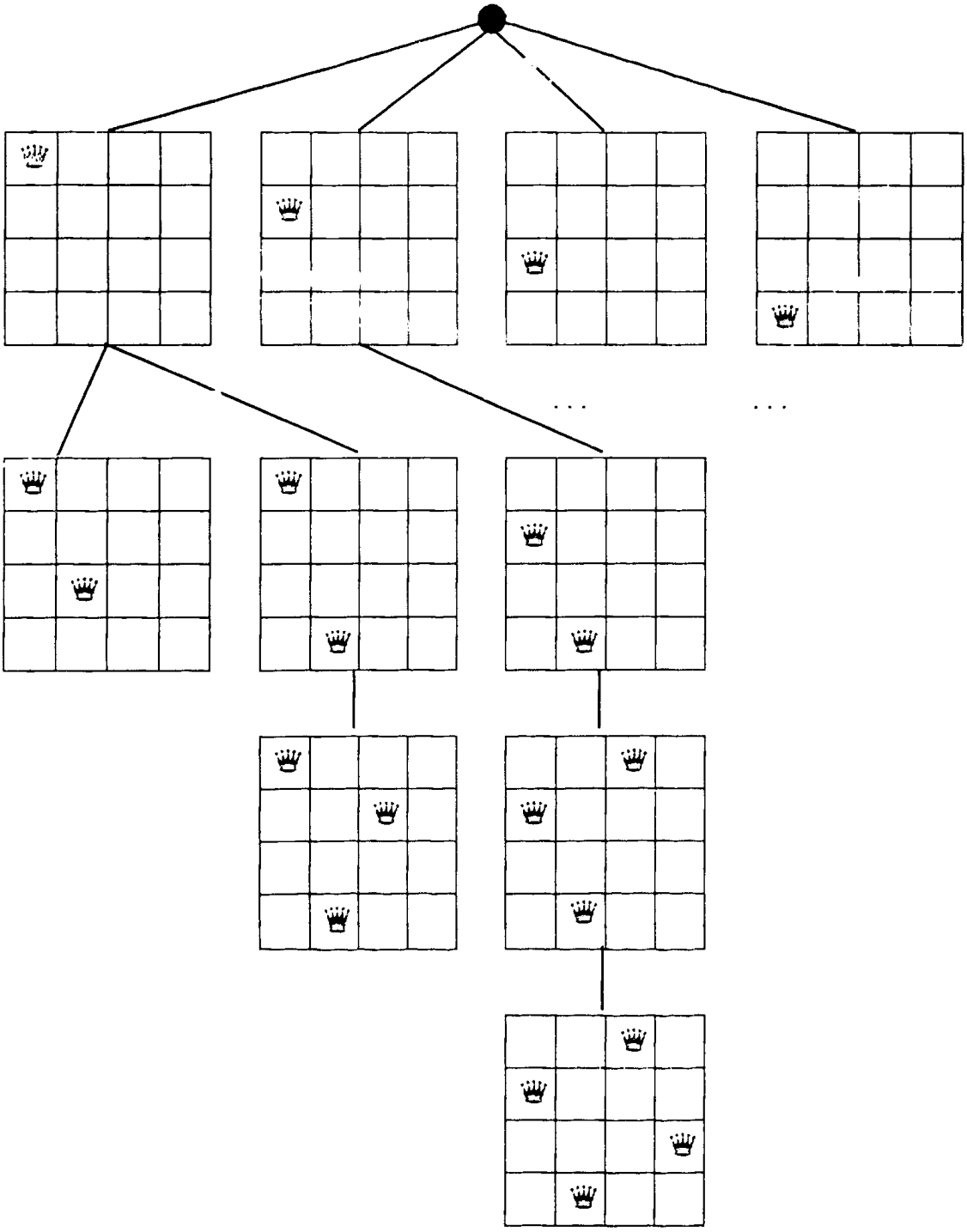
$$x = (1, 5, 8, 6, 3, 7, 2, 4)$$

nó tương ứng với cách xếp Hậu cho bởi hình 6.



Hình 6. Một lời giải của bài toán xếp hậu khi $n = 8$

Cây liệt kê lời giải theo thuật toán quay lui để liệt kê các cách xếp hậu với n=4 được mô tả trong hình 5 (hai nhánh còn lại là đối xứng với hai nhánh đầu).



Cách xếp (2, 4, 1, 3)

Hình 5. Cây liệt kê các cách xếp hậu trên bàn cờ 4×4

Thí dụ 5. Liệt kê tất cả các cách xếp $k \le n$ quân Hậu trên bàn cờ $n \times n$ sao cho chúng không ăn được lẫn nhau đồng thời chúng khống chế tất cả các ô của bàn cờ.

Giải. Ta có thể giải bài toán đặt ra bằng cách liệt kê tất cả các cách chọn ra k dòng để xếp các con hậu, ứng với mỗi cách chọn đó áp dụng phương pháp ở thí dụ 4 để liệt kê các cách xếp hậu vào k dòng đã chọn sao cho chúng không ăn được lẫn nhau đồng thời khống chế tất cả các ô của bàn cờ. Chương trình giải bài toán này ta không dẫn ra ở đây. Điều đáng lưu ý là khác với bài toán trong thí dụ 4, bài toán ở đây trong rất nhiều trường hợp là không có lời giải. Trong bài toán này, người ta còn quan tâm đến số k_{min} nhỏ nhất thoả mãn điều kiện đặt ra. Bảng dưới đây cho thấy một số giá trị đã biết của k_{min}.

n	1	2	3	4	5	6	7	8	9	10	11	12	13
k_{min}	1	1	1	3	3	4	4	5	5	5	5	7	7

n	14	**15**	**16**	17	18	19	20	21	22	23	24	25
k_{min}	8	**9**	**9**	9	≤10	≤11	≤11	11	≤13	≤13	≤13	13

Các ô có dấu bất đẳng thức là còn chưa xác định được chính xác giá trị của k_{min}.

Với n=15, để liệt kê tất cả các lời giải của bài toán khi k =9 và chứng minh bài toán không có lời giải khi k=8 chương trình viết trên ngôn ngữ C của các tác giả Gibbons và Webb đã phải sử dụng thời gian tương ứng là 58.4 và 23.5 giờ trên máy DEC ALPHA 3000/600. Để chỉ ra với n=16, khi k=8 bài toán không có lời giải, chương trình nói trên đã chạy mất 113.7 giờ máy.

Hy vọng rằng thí dụ cuối cùng này giúp chúng ta, một mặt, hình dung được sự phức tạp của các bài toán liệt kê tổ hợp, mặt khác, thấy rằng liệt kê với sự trợ giúp của máy tính là biện pháp để giải nhiều vấn đề tồn tại trong tổ hợp.

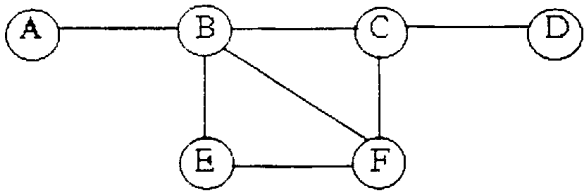
Bài tập

Trong các bài 1-5 hãy vẽ cây liệt kê lời giải theo thuật toán quay lui để giải các bài toán liệt kê tổ hợp đặt ra.

1. Giả sử A, B, . . ., F trên hình 1 là các hòn đảo và các đoạn nối là các cây cầu nối chúng. Một người du lịch khởi hành từ A đi từ hòn đảo này sang hòn đảo khác. Người du lịch sẽ dừng lại để ăn trưa nếu như tiếp tục đi sẽ phải đi qua cái cầu nào đó hai lần.

(a) Liệt kê các cách mà người du lịch có thể đi cho đến khi dừng lại ăn trưa.

(b) Cho biết ở những điểm nào người du lịch có thể dừng lại ăn trưa?



Hình 1.

2. Hai đội bóng chuyền A, B thi đấu trong một giải vô địch quốc gia. Đội thắng trong trận đấu sẽ là đội giành được ba hiệp thắng trước. Hãy liệt kê tất cả các khả năng có thể của trận đấu giữa hai đội.

3. Liệt kê tất cả các cách mất thứ tự của 4 số tự nhiên 1, 2, 3, 4.

4. Liệt kê các xâu nhị phân độ dài 5 không chứa hai số 0 liên tiếp.

5. Liệt kê các chỉnh hợp lặp chập 4 từ 3 chữ A, B, C không chứa hai chữ A hoặc hai chữ B.

Lập trình trên PASCAL giải các bài toán liệt kê sau bằng thuật toán quay lui

6. Liệt kê tất cả các phần tử của

$$D = \{ x = (x_1, x_2, ..., x_n): \quad \sum_{j=1}^{n} a_j x_j = b, \quad x_j \in \{0, 1\}, j = 1,2,...,n \}$$

trong đó $a_1, a_2, \ldots, a_n, b$ là các số nguyên dương.

7. Liệt kê tất cả các phần tử của

$$D = \{ x = (x_1, x_2, ..., x_n): \quad \sum_{j=1}^{n} a_j x_j = b, \quad x_j \in Z_+, j = 1,2,...,n \}$$

trong đó $a_1, a_2, \ldots, a_n, b$ là các số nguyên dương, Z_+ là tập các số nguyên không âm.

8. Cho tập $X = \{1, 2, ..., n\}$. Hãy liệt kê tất cả các phân hoạch tập X ra thành k tập con $X_1, X_2, \ldots, X_k$ ($k \leq n$).

9. Cho số nguyên dương N. Hãy liệt kê tất cả các cách biểu diễn N dưới dạng tổng của một số số nguyên dương.

10. Hình vuông thần bí (ma phương) bậc n là ma trận vuông cấp n với các phần tử là các số tự nhiên từ 1 đến n^2 thoả mãn tính chất: "*Tổng các phần tử trên mỗi dòng, mỗi cột và mỗi một trong hai đường chéo đều có cùng một giá trị*".
Thí dụ: Dưới đây là một ma phương bậc 3

8	1	6
3	5	7
4	9	2

Đối với ma phương này, ta có tổng các phần tử trên các dòng là:

$$8 + 1 + 6 = 3 + 5 + 7 = 4 + 9 + 2 = 15;$$

tổng các phần tử trên các cột là:

$$8 + 3 + 4 = 1 + 5 + 9 = 6 + 7 + 2 = 15;$$

tổng các phần tử trên hai đường chéo là:

$$8 + 5 + 2 = 6 + 5 + 4 = 15.$$

Hãy viết chương trình liệt kê tất cả các ma phương bậc $n = 3, 4$ không sai khác nhau bởi các phép biến hình đơn giản (quay, đối xứng).
(Chú ý: Bài toán đếm số lượng ma phương bậc n mới chỉ giải được với $n = 3, 4, 5$.)

11. Tam giác thần bí. Cho một lưới ô vuông gồm $n \times n$ ô và số nguyên dương k. Tìm cách điền các số tự nhiên từ 1 đến $3n-3$ vào các ô ở cột đầu tiên, dòng cuối cùng và đường chéo chính sao cho tổng các số điền trong cột đầu tiên, dòng cuối cùng và đường chéo chính của lưới đều bằng k.
Ví dụ: Với $n = 5$, $k = 35$ ta có cách điền sau:

11				
10	2			
9		3		
1			7	
4	5	6	8	12

Hãy phát triển thuật toán dựa trên thuật toán quay lui để chỉ ra với giá trị của n và k cho trước bài toán đặt ra có lời giải hay không? Nếu câu trả lời là khẳng định chỉ cần đưa ra một lời giải.

12. Bài toán về 8 quân cờ: Cho 8 quân cờ: Vua (V), Hậu (H), 2 con Ngựa (N), 2 con Tượng (T) khác màu, 2 con Xe (X). Tìm cách xếp chúng lên bàn cờ quốc tế 8×8 ô sao cho chúng khống chế được nhiều ô của bàn cờ nhất. Lưu ý rằng: Vị trí quân cờ đứng không coi là bị nó khống chế, Các vị trí nằm trong khả năng khống chế của quân cờ nhưng bị một quân nào đó chắn đường di chuyển đến đó cũng coi là chưa bị quân cờ đó khống chế.

Thí dụ: Cách xếp các quân cờ trong hình vẽ dưới đây khống chế được 49 ô:

8				T				
7				N				
6				N				
5					X			
4						T		
3				H			X	
2				V				
1								
	a	b	c	d	e	f	g	h

(Các ô bôi đen là chưa bị khống chế bởi các quân cờ xếp trên bàn cờ).

Hãy sử dụng thuật toán quay lui giải bài toán đặt ra.

13. Điền số. Viết chương trình giải bài toán sau đây:

Cho bảng vuông gồm $N \times N$ ô vuông. Các dòng của bảng được đánh số từ 1 đến N, từ trên xuống dưới. Các cột của bảng được đánh số từ 1 đến N từ trái sang phải. Bên cạnh mỗi cột và mỗi dòng của bảng đều có ghi một số nguyên dương. Gọi X_i là số viết bên cạnh cột i, còn Y_j là số viết bên cạnh dòng j.

Yêu cầu: Cần điền các số nguyên dương vào một số ô của bảng sao cho:

- Các số được điền phải khác nhau từng đôi;
- Mỗi dòng và mỗi cột cần điền đúng hai số;
- Tích của hai số điền trên cột i phải bằng X_i;
- Tích của hai số điền trên dòng j phải bằng Y_j;

Hình dưới đây cho thí dụ về lời giải của một bài toán dạng vừa nêu.

6	3				18
9		1			9
		12		11	132
			4	8	32
	2		5		10
54	6	12	20	88	

Dữ liệu: Vào từ file văn bản NUMBOARD.INP:

- Dòng đầu tiên ghi số nguyên N ($2 \le N \le 10$);
- Dòng thứ hai ghi N số $X_1, ..., X_N$ ($1 \le X_i \le 1000, i = 1, ..., N$);
- Dòng thứ ba ghi N số $Y_1, ..., Y_N$ ($1 \le Y_j \le 1000, j = 1, ..., N$).

Kết quả: Ghi ra file văn bản NUMBOARD.OUT: Dòng thứ 1 ghi N số trên dòng thứ 1 của bảng; Dòng thứ hai ghi N số trên dòng thứ hai của bảng,..., Dòng thứ N ghi N số trên dòng thứ N của bảng. Qui ước: Ghi số 0 ở vị trí các ô không được điền số.

Ví dụ: Nội dung của các file dữ liệu và kết quả tương ứng có thể như sau

NUMBOARD.INP	NUMBOARD.OUT
2 2 12 3 8	1 3 2 4
3 5 8 18 2 30 12	1 2 0 5 0 6 0 4 3
5 54 6 12 20 88 18 9 132 32 10	6 3 0 0 0 9 0 1 0 0 0 0 12 0 11 0 0 0 4 8 0 2 0 5 0

5

BÀI TOÁN TỐI ƯU

5.1. Phát biểu bài toán

Trong rất nhiều vấn đề ứng dụng thực tế của tổ hợp các cấu hình tổ hợp còn được gán cho một giá trị bằng số đánh giá giá trị sử dụng của cấu hình đối với mục đích sử dụng cụ thể nào đó. Khi đó xuất hiện bài toán: Hãy lựa chọn trong số các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất. Các bài toán như vậy chúng ta sẽ gọi là bài toán tối ưu tổ hợp. Dưới dạng tổng quát bài toán tối ưu tổ hợp có thể phát biểu như sau:

Tìm cực tiểu (hay cực đại) của phiếm hàm

$$f(x) \to \min (\max),$$

với điều kiện

$$x \in D,$$

trong đó D là tập hữu hạn phần tử.

Hàm $f(x)$ được gọi là hàm mục tiêu của bài toán, mỗi phần tử $x \in D$ được gọi là một phương án còn tập D gọi là tập các phương án của bài toán.

Thông thường tập D được mô tả như là tập các cấu hình tổ hợp thoả mãn một số tính chất cho trước nào đó.

Phương án $x^* \in D$ đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là phương án tối ưu, khi đó giá trị $f^* = f(x^*)$ được gọi là giá trị tối ưu của bài toán.

Dưới đây chúng ta sẽ giới thiệu một số mô hình tối ưu hoá tổ hợp truyền thống. Các mô hình này, một mặt, là những mô hình có rất nhiều ứng dụng thực tế, mặt khác, chúng giữ vai trò quan trọng trong việc nghiên cứu và phát triển lý thuyết tối ưu hoá tổ hợp.

Bài toán người du lịch.

Một người du lịch muốn đi tham quan n thành phố $T_1, T_2, ..., T_n$. Xuất phát từ một thành phố nào đó người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đúng một lần, rồi quay trở lại thành phố xuất phát. Biết c_{ij} là chi phí đi từ thành phố T_i đến thành phố T_j ($i, j = 1, 2,..., n$), hãy tìm hành trình (một cách đi thoả mãn điều kiện đặt ra) với tổng chi phí là nhỏ nhất.

Rõ ràng ta có thể thiết lập tương ứng 1-1 giữa hành trình

$$T_{\pi(1)} \to T_{\pi(2)} \to ... \to T_{\pi(n)} \to T_{\pi(1)}$$

với một hoán vị $\pi = (\pi(1), \pi(2),..., \pi(n))$ của n số tự nhiên 1, 2,..., n. Đặt

$$f(\pi) = c_{\pi(1),\pi(2)} + ... + c_{\pi(n-1),\pi(n)} + c_{\pi(n),\pi(1)},$$

và ký hiệu Π là tập tất cả các hoán vị $\pi = (\pi(1), \pi(2),..., \pi(n))$ của n số tự nhiên 1, 2,..., n. Khi đó bài toán người du lịch có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

$$\min \{ f(\pi) : \pi \in \Pi \}.$$

Có thể thấy rằng tổng số hành trình của người du lịch là $n!$, trong đó chỉ có $(n-1)!$ hành trình thực sự khác nhau (bởi vì có thể xuất phát từ một thành phố bất kỳ, nên có thể cố định một thành phố nào đó là thành phố xuất phát).

Bài toán cái túi.

Một nhà thám hiểm cần đem theo một cái túi có trọng lượng không quá b. Có n đồ vật có thể đem theo. Đồ vật thứ j có trọng lượng là a_j và giá trị sử dụng là c_j ($j = 1, 2,..., n$). Hỏi rằng nhà thám hiểm cần đem theo các đồ vật nào để cho tổng giá trị sử dụng của các đồ vật đem theo là lớn nhất?

Một phương án đem đồ của nhà thám hiểm có thể biểu diễn bởi vectơ nhị phân độ dài n: $x = (x_1, x_2,..., x_n)$, trong đó $x_j = 1$ có nghĩa là đồ vật thứ j được đem theo và $x_j = 0$ có nghĩa trái lại. Với phương án x, giá trị đồ vật đem theo là

$$f(x) = \sum_{j=1}^{n} c_j x_j ,$$

tổng trọng lượng đồ vật đem theo là

$$g(x) = \sum_{j=1}^{n} a_j x_j$$

và bài toán cái túi có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

Trong số các vectơ nhị phân độ dài n thoả mãn điều kiện $g(x) \le b$, hãy tìm vectơ x^* cho giá trị nhỏ nhất của hàm mục tiêu $f(x)$:

$$\min \{ f(x): g(x) \le b \}.$$

Bài toán cho thuê máy.

Một ông chủ có một cái máy để cho thuê. Đầu tháng ông ta nhận được yêu cầu thuê máy của m khách hàng. Mỗi khách hàng i sẽ cho biết tập N_i các ngày trong tháng cần sử dụng máy ($i = 1, 2,..., m$). Ông chủ chỉ có quyền hoặc là từ chối yêu cầu của khách hàng i, hoặc là nếu nhận thì phải bố trí máy phục vụ khách hàng i đúng những ngày mà khách hàng này yêu cầu. Hỏi rằng ông chủ phải tiếp nhận các yêu cầu của khách như thế nào để cho tổng số ngày sử dụng máy là lớn nhất.

Ký hiệu $I = \{1, 2,..., m\}$ là tập chỉ số khách hàng, S là tập hợp các tập con của I. Khi đó tập hợp tất cả các phương án cho thuê máy là

$$D = \{ J \subset S : N_k \cap N_p = \varnothing, \ \forall k \ne p, \ k, p \in J \}.$$

và với mỗi phương án $J \in D$

$$f(J) = \sum_{j \in J} | N_j |$$

sẽ là tổng số ngày sử dụng máy theo phương án đó. Bài toán đặt ra có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

$$\max \{ f(J) : J \in D \}.$$

Bài toán đóng thùng.

Có n đồ vật với trọng lượng là $w_1, w_2, ..., w_n$. Cần tìm cách xếp các đồ vật này vào các cái thùng có cùng dung lượng là b sao cho số thùng cần sử dụng là nhỏ nhất có thể được.

Ta có thể giả thiết là

$$w_i \le b, i = 1, 2,.., n.$$

Do đó số thùng cần sử dụng để chứa tất cả các đồ vật là không quá n. Vấn đề là cần số thùng ít nhất. Ta sẽ mở sẵn n cái thùng. Bài toán đặt ra là hãy xác định xem mỗi một trong số n đồ vật cần được xếp vào cái thùng nào trong số n cái thùng đã mở để cho số thùng chứa đồ là ít nhất. Phân tích vừa nêu cho thấy bài toán đóng thùng là bài toán tối ưu tổ hợp.

Bài toán phân công.

Có n công việc và n thợ. Biết c_{ij} là chi phí cần trả để thợ i hoàn thành công việc j (i, j = 1, 2,..., n). Cần phải thuê thợ sao cho các công việc đều hoàn thành và mỗi thợ chỉ thực hiện một công việc, mỗi công việc chỉ do một thợ thực hiện. Hãy tìm cách thuê sao cho tổng chi phí thuê thợ là nhỏ nhất.

Rõ ràng mỗi một phương án bố trí thợ thực hiện các công việc

Công việc	Thợ thực hiện
1	$\pi(1)$
2	$\pi(2)$
...	...
n	$\pi(n)$

tương ứng với một hoán vị $\pi = (\pi(1), \pi(2),..., \pi(n))$ của n số tự nhiên 1, 2,..., n. Chi phí theo phương án bố trí trên là

$$f(\pi) = c_{\pi(1),1} + c_{\pi(2),2} + ... + c_{\pi(n),n}.$$

Bài toán đặt ra, được dẫn về bài toán tối ưu tổ hợp:

$$\min \{ f(\pi) : \pi \in \Pi \}.$$

Bài toán lập lịch.

Mỗi một chi tiết trong số n chi tiết $D_1, D_2,..., D_n$ cần phải được lần lượt gia công trên m máy $M_1, M_2,..., M_m$. Thời gian gia công chi tiết D_i trên máy M_j là t_{ij}. Hãy tìm lịch (trình tự gia công) các chi tiết trên các máy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất có thể được.

Ta sẽ xét bài toán trên với thêm giả thiết là các chi tiết phải được gia công một cách liên tục, nghĩa là quá trình gia công của mỗi một chi tiết cần phải được tiến hành một cách liên tục hết máy này sang máy khác không cho phép có khoảng thời gian dừng khi chuyển từ máy này sang máy khác. Tình huống như vậy rất hay gặp trong các ngành sản xuất công nghiệp. Chẳng hạn, trong công nghiệp luyện thép, vật liệu cần phải được gia công một cách liên tục vì việc gián đoạn sẽ dẫn đến sự giảm nhiệt độ của vật liệu và

điều đó cản trở việc gia công tiếp theo. Tình huống tương tự cũng có thể xảy ra trong một số ngành của công nghiệp hoá chất.

Rõ ràng mỗi một lịch gia công các chi tiết trên các máy trong tình huống như vậy sẽ tương ứng với một hoán vị $\pi = (\pi(1), \pi(2),..., \pi(n))$ của n số tự nhiên 1, 2,..., n. Thời gian hoàn thành theo lịch trên được tính bởi hàm số

$$f(\pi) = \sum_{j=1}^{n-1} c_{\pi(j)\,\pi(j+1)} + \sum_{k=1}^{m} t_{k,\pi(n)}$$

trong đó $c_{ij} = S_j - S_i$, S_j - thời điểm bắt đầu thực hiện việc gia công chi tiết j ($i, j = 1, 2,..., n$). Ý nghĩa của hệ số c_{ij} có thể giải thích như sau: nó là tổng thời gian gián đoạn (được tính từ khi bắt đầu gia công chi tiết i) gây ra bởi chi tiết j khi nó được gia công sau chi tiết i trong lịch gia công. Vì vậy, c_{ij} có thể tính theo công thức:

$$c_{ij} = \max_{1 \le k \le m} \left[\sum_{l=1}^{k} t_{lj} - \sum_{l=1}^{k-1} t_{li} \right], \quad i, j = 1,2,...,n.$$

Vì vậy, bài toán đặt ra dẫn về bài toán tối ưu tổ hợp sau

$$\min \{ f(\pi): \pi \in \Pi \}.$$

Trong thực tế lịch gia công thường còn phải thoả mãn thêm nhiều điều kiện khác.Vì những ứng dụng quan trọng của vấn đề này mà trong tối ưu hoá tổ hợp đã hình thành một lĩnh vực lý thuyết riêng về các bài toán lập lịch gọi là lý thuyết lập lịch (hay quy hoạch lịch).

5.2. Các thuật toán duyệt

5.2.1. Duyệt toàn bộ

Một trong những phương pháp hiển nhiên nhất để giải bài toán tối ưu tổ hợp đặt ra là: Trên cơ sở các thuật toán liệt kê tổ hợp ta tiến hành duyệt từng phương án của bài toán, đối với mỗi phương án ta đều tính giá trị hàm mục tiêu tại nó, sau đó so sánh giá trị hàm mục tiêu tại tất cả các phương án được liệt kê để tìm ra phương án tối ưu. Phương pháp xây dựng theo nguyên tắc như vậy có tên gọi là phương pháp duyệt toàn bộ. Duyệt toàn bộ là khó có thể thực hiện được ngay cả trên những máy tính điện tử hiện đại nhất. Ví dụ để liệt kê hết

$$15! = 1\,307\,674\,368\,000$$

hoán vị trên máy tính điện tử với tốc độ tính toán 1 tỷ phép tính một giây, nếu để liệt kê một hoán vị cần phải làm 100 phép tính, thì ta cần một khoảng thời gian là 130767 giây

> 36 tiếng đồng hồ! Vì vậy cần phải có những biện pháp nhằm hạn chế việc tìm kiếm thì mới có hy vọng giải được các bài toán tối ưu tổ hợp thực tế. Tất nhiên để có thể đề ra những biện pháp như vậy cần phải nghiên cứu kỹ tính chất của bài toán tối ưu tổ hợp cụ thể. Nhờ những nghiên cứu như vậy, trong một số trường hợp cụ thể ta có thể xây dựng những thuật toán hiệu quả để giải bài toán đặt ra. Tuy nhiên phải nhấn mạnh rằng trong nhiều trường hợp (ví dụ trong các bài toán người du lịch, bài toán cái túi, bài toán cho thuê máy nêu ở trên) chúng ta chưa thể xây dựng được phương pháp hữu hiệu nào khác ngoài phương pháp duyệt toàn bộ. Khi đó, một vấn đề đặt ra là trong quá trình liệt kê lời giải ta cần tận dụng các thông tin đã tìm được để loại bỏ những phương án chắc chắn không phải là tối ưu. Trong mục tiếp theo chúng ta sẽ xét một sơ đồ tìm kiếm như vậy để giải các bài toán tối ưu tổ hợp mà trong tài liệu tham khảo được biết đến với tên gọi: thuật toán nhánh cận.

5.2.1. Thuật toán nhánh cận.

Ta sẽ mô tả tư tưởng của thuật toán trên mô hình bài toán tối ưu tổ hợp tổng quát sau

$$\min \{ f(x) : x \in D \},$$

trong đó D là tập hữu hạn phần tử.

Giả thiết rằng tập D được mô tả như sau

$$D = \{x = (x_1, x_2, ..., x_n) \in A_1 \times A_2 \times ... \times A_n: x \text{ thoả mãn tính chất } P\},$$

với $A_1, A_2, ..., A_n$ là các tập hữu hạn, còn P là tính chất cho trên tích Đêcac $A_1 \times A_2 \times ... \times A_n$.

Nhận thấy rằng, các bài toán vừa trình bày ở mục trước đều có thể mô tả dưới dạng bài toán trên.

Với giả thiết về tập D nêu trên, chúng ta có thể sử dụng thuật toán quay lui để liệt kê các phương án của bài toán. Trong quá trình liệt kê theo thuật toán quay lui, ta sẽ xây dựng dần các thành phần của phương án. Một bộ gồm k thành phần $(a_1, a_2, ..., a_k)$ xuất hiện trong quá trình thực hiện thuật toán sẽ gọi là phương án bộ phận cấp k.

Thuật toán nhánh cận có thể áp dụng để giải bài toán đặt ra nếu như có thể tìm được một hàm g xác định trên tập tất cả các phương án bộ phận của bài toán thoả mãn bất đẳng thức sau:

$$g(a_1, a_2, \ldots, a_k) \le \min \{f(x): x \in D, x_i = a_i, i = 1, 2,..., k\} \qquad (*)$$

với mọi lời giải bộ phận $(a_1, a_2, ..., a_k)$, và với mọi $k = 1, 2, ...$

Bất đẳng thức (*) có nghĩa là giá trị của hàm g tại phương án bộ phận $(a_1, a_2, ..., a_k)$ là không vượt quá giá trị nhỏ nhất của hàm mục tiêu của bài toán trên tập con các phương án

$$D(a_1, a_2, ..., a_k) = \{ x \in D : x_i = a_i , i = 1, 2,..., k \},$$

hay nói một cách khác, $g(a_1, a_2, \ldots, a_k)$ là cận dưới của giá trị hàm mục tiêu trên tập $D(a_1, a_2, ..., a_k)$. Vì lẽ đó, hàm g được gọi là hàm cận dưới, và giá trị $g(a_1, a_2, \ldots, a_k)$ được gọi là cận dưới của tập $D(a_1, a_2, ..., a_k)$. Do có thể đồng nhất tập $D(a_1, a_2, ..., a_k)$ với phương án bộ phận $(a_1, a_2, ..., a_k)$. nên ta cũng gọi giá trị $g(a_1, a_2, \ldots, a_k)$ là cận dưới của phương án bộ phận $(a_1, a_2, ..., a_k)$.

Giả sử đã có hàm g. Ta xét cách sử dụng hàm này để giảm bớt khối lượng duyệt trong quá trình duyệt tất cả các phương án theo thuật toán quay lui. Trong quá trình liệt kê các phương án có thể đã thu được một số phương án của bài toán. Gọi $\bar{x}$ là phương án với giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã tìm được, ký hiệu $\bar{f} = f(\bar{x})$. Ta sẽ gọi $\bar{x}$ là phương án tốt nhất hiện có, còn $\bar{f}$ là kỷ lục. Giả sử đã có $\bar{f}$, khi đó nếu

$$g(a_1, a_2, ..., a_k) > \bar{f},$$

thì từ bất đẳng thức (*) suy ra

$$\bar{f} < g(a_1, a_2, \ldots, a_k) \le \min \{f(x): x \in D, x_i = a_i , i = 1, 2,..., k\},$$

vì thế tập con các phương án của bài toán $D(a_1, a_2, ..., a_k)$ chắc chắn không chứa phương án tối ưu. Trong trường hợp này ta không cần tiếp tục phát triển phương án bộ phận $(a_1, a_2, \ldots, a_k)$, nói cách khác là ta có thể loại bỏ các phương án trong tập $D(a_1, a_2, \ldots, a_k)$ khỏi quá trình tìm kiếm.

Thuật toán quay lui liệt kê phương án cần sửa đổi lại như sau

```
procedure Try(k);
(*  Phát triển phương án bộ phận (a1, a2, ..., ak-1)
    theo thuật toán quay lui có kiểm tra cận dưới
    trước khi tiếp tục phát triển phương án *)
begin
    for ak ∈ Ak do
        if <chấp nhận ak> then
        begin
            xk := ak;
            if k = n then < Cập nhật kỷ lục>
            else
            if g(a1, a2, ..., ak) ≤ f̄ then Try(k+1);
        end;
end;
```

Khi đó, thuật toán nhánh cận được thực hiện nhờ thủ tục sau

```
procedure Nhanh_can;
begin
    f̄ := +∞;
    (* Nếu biết một phương án x̄ nào đó thì có thể đặt f̄ = f(x̄) *)
    Try(1);
    if f̄ < +∞ then < f̄ là giá trị tối ưu, x̄ là phương án tối ưu >
    else < bài toán không có phương án >;
end;
```

Chú ý rằng nếu trong thủ tục Try ta thay câu lệnh

```
    if k = n then <Cập nhật kỷ lục>
    else
    if g(a1, a2, ..., ak) ≤ f̄ then Try(k+1)
```

bởi

```
    if k = n then <Cập nhật kỷ lục>
    else Try(k+1)
```

thì thủ tục Try sẽ liệt kê toàn bộ các phương án của bài toán, và ta thu được thuật toán duyệt toàn bộ.

Việc xây dựng hàm g phụ thuộc vào từng bài toán tối ưu tổ hợp cụ thể. Thông thường ta cố gắng xây dựng nó sao cho:

- Việc tính giá trị của g phải đơn giản hơn việc giải bài toán tối ưu tổ hợp ở vế phải của (*).
- Giá trị của $g(a_1, a_2, \ldots, a_k)$ phải sát với giá trị của vế phải của (*).

Rất tiếc là hai yêu cầu này trong thực tế thường đối lập nhau.

Dưới đây chúng ta sẽ xét một số thí dụ minh hoạ cho thuật toán vừa trình bày.

a) ***Bài toán cái túi.***

Chúng ta sẽ xét một dạng bài toán cái túi nữa rất hay được sử dụng trong các ứng dụng thực tế (về nguyên tắc, mô hình ở đây và mô hình đã trình bày trong mục 5.1 là có thể qui dẫn về nhau). Thay vì có n đồ vật như mô hình trong mục 5.1, ở đây ta giả thiết rằng có n loại đồ vật và số lượng đồ vật mỗi loại là không hạn chế. Khi đó ta có mô hình bài toán cái túi biến nguyên sau đây: Có n loại đồ vật, đồ vật thứ j có trọng lượng a_j và giá trị sử dụng là c_j ($j = 1, 2, \ldots, n$). Cần chất các đồ vật này vào một cái túi có trọng lượng là b sao cho tổng giá trị sử dụng của các đồ vật chất trong túi là lớn nhất.

Mô hình toán học của bài toán có dạng sau: Tìm

$$f^* = \max \{ f(x) = \sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \le b, x_j \in Z_+, j = 1,2,\ldots,n \}, \qquad (1)$$

trong đó Z_+ là tập các số nguyên không âm.

Ký hiệu D là tập các phương án của bài toán (1):

$$D = \{ x=(x_1, x_2, \ldots, x_n): \sum_{j=1}^{n} a_j x_j \le b, x_j \in Z_+, j=1,2,\ldots,n \}.$$

Không giảm tổng quát ta giả thiết rằng các đồ vật được đánh số sao cho bất đẳng thức sau được thoả mãn

$$c_1/a_1 \ge c_2/a_2 \ge \ldots \ge c_n/a_n. \qquad (2)$$

Để xây dựng hàm tính cận dưới, cùng với bài toán cái túi (1) ta xét bài toán cái túi biến liên tục sau: Tìm

$$g^* = \max \{ f(x) = \sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \le b, x_j \ge 0, j = 1,2,\ldots,n \}, \qquad (3)$$

Mệnh đề. *Phương án tối ưu của bài toán (3) là vectơ* $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ *với các thành phần được xác định bởi công thức:*

$$\bar{x}_1 = b/a_1, \quad \bar{x}_2 = \bar{x}_3 = \ldots = \bar{x}_n = 0.$$

và giá trị tối ưu là $g^* = c_1 b_1 / a_1$.

Chứng minh. Thực vậy, xét $x = (x_1, x_2, \ldots, x_n)$ là một phương án tuỳ ý của bài toán (3). Khi đó từ bất đẳng thức (3) và do $x_j \ge 0$, ta suy ra

$$c_j x_j \le (c_1/a_1) a_j x_j, \; j = 1, 2, \ldots, n$$

suy ra

$$\sum_{j=1}^{n} c_j x_j \le \sum_{j=1}^{n} (c_1/a_1) a_j x_j = (c_1/a_1) \sum_{j=1}^{n} a_j x_j \le (c_1/a_1) b = g^*.$$

Mệnh đề được chứng minh.

Bây giờ, giả sử ta có phương án bộ phận cấp k: $(u_1, u_2, \ldots, u_k)$. Khi đó giá trị sử dụng của các đồ vật đang có trong túi là

$$\sigma_k = c_1 u_1 + c_2 u_2 + \ldots + c_k u_k$$

và trọng lượng còn lại của cái túi là

$$b_k = b - a_1 u_1 + a_2 u_2 + \ldots + a_k u_k.$$

Ta có

$$\max\{ f(x): x \in D, x_j = u_j, j = 1, 2, ..., k \}$$

$$= \max \{\sigma_k + \sum_{j=k+1}^{n} c_j x_j : \sum_{j=k+1}^{n} a_j x_j \le b_k, x_j \in Z_+, j = k+1, k+2, ..., n\}$$

$$\le \sigma_k + \max \{ \sum_{j=k+1}^{n} c_j x_j : \sum_{j=k+1}^{n} a_j x_j \le b_k, x_j \ge 0, j = k+1, k+2, ..., n\}$$

(theo Mệnh đề: giá trị số hạng thứ hai là $c_{k+1} b_k / a_{k+1}$)

$$= \sigma_k + c_{k+1} b_k / a_{k+1}.$$

Vậy ta có thể tính cận trên cho phương án bộ phận $(u_1, u_2, ..., u_k)$ bởi công thức

$$g(u_1, u_2, ..., u_k) = \sigma_k + c_{k+1} b_k / a_{k+1}.$$

Chú ý: Khi tiếp tục xây dựng thành phần thứ $k+1$ của lời giải, các giá trị đề cử cho x_{k+1} sẽ là $0, 1, ..., [b_k / a_{k+1}]$. Do có kết quả của mệnh đề, khi chọn giá trị cho x_{k+1} ta sẽ duyệt các giá trị đề cử theo thứ tự giảm dần.

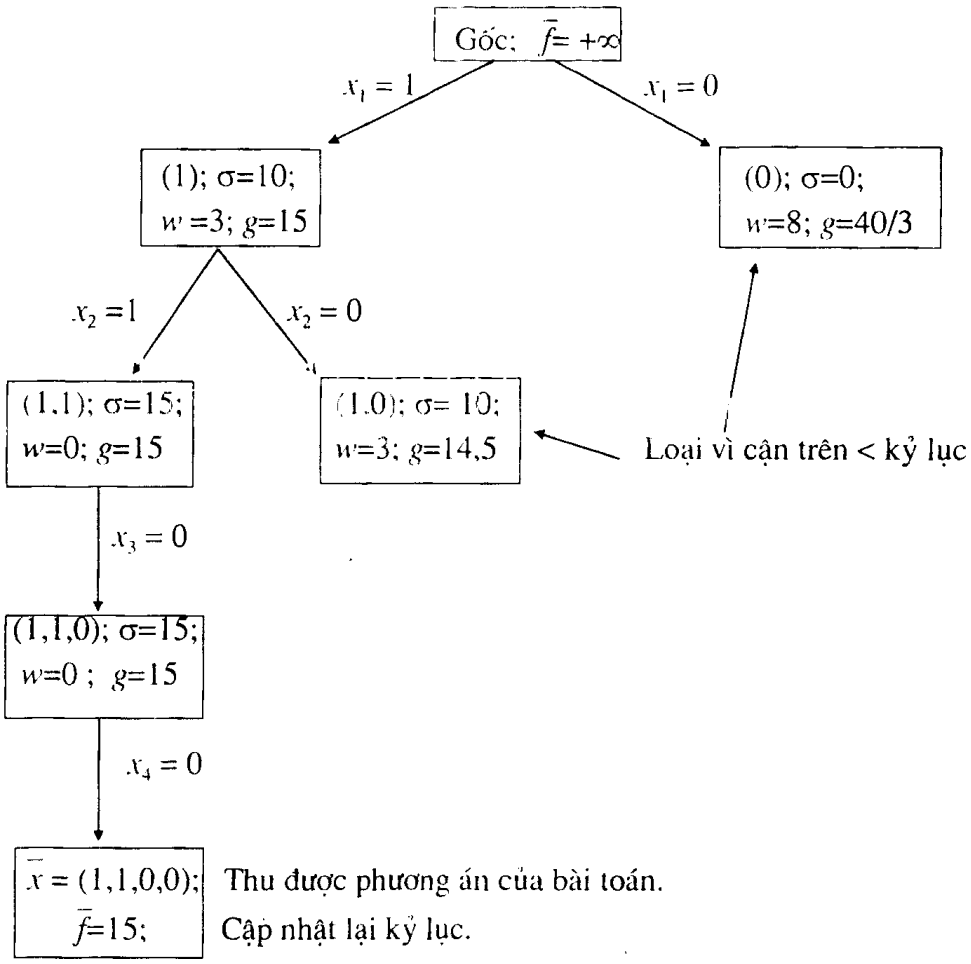
Thí dụ. Giải bài toán cái túi sau theo thuật toán nhánh cận vừa trình bày

$$\begin{aligned} f(x) = \; & 10 x_1 + 5 x_2 + 3 x_3 + 6 x_4 \to \max, \\ & 5 x_1 + 3 x_2 + 2 x_3 + 4 x_4 \le 8, \\ & x_j \in Z_+, j = 1, 2, 3, 4. \end{aligned}$$

Giải.

Quá trình giải bài toán được mô tả trong cây tìm kiếm trong hình 1. Thông tin về một phương án bộ phận trên cây được ghi trong các ô trên hình vẽ tương ứng theo thứ tự sau: đầu tiên là các thành phần của phương án, tiếp đến, σ là giá trị của các đồ vật đang chất trong túi, w - trọng lượng còn lại của túi và g - cận trên.

Kết thúc thuật toán, ta thu được phương án tối ưu là $x^* = (1, 1, 0, 0)$, và giá trị tối ưu là $f^* = 15$.



Hình 1. Giải bài toán cái túi bằng thuật toán nhánh cận

Dưới đây là chương trình trên PASCAL giải bài toán cái túi theo thuật toán nhánh cận vừa trình bày.

```
uses crt;
const inf = 25000;
type   arrn  = array[1..50] of integer;
       arrn1 = array[1..50] of real;
var
   c, a          : arrn1;
   x, xopt, ind  : arrn;
   n             : integer;
   w, fopt, cost, weight: real;
procedure Init;
var i,j,tg:integer;
   t:real;
```

```
begin
  (* Sap xep cac do vat theo thu tu khong tang cua c[i]/a[i] *)
  fopt:=0;  weight:=0;
  for i:=1 to n do ind[i]:=i;
  for i:=1 to n-1 do
  begin
    for j:=i+1 to n do
      if (c[i]/a[i]< c[j]/a[j]) then
      begin
        t:=c[i]; c[i]:=c[j]; c[j]:=t;
        t:=a[i]; a[i]:=a[j]; a[j]:=t;
        tg:=ind[i]; ind[i]:=ind[j]; ind[j]:=tg;
      end;
  end;
end;
procedure Readfile;
var j   : integer;
   f   : text;  name : string;
begin
  write('Cho ten file du lieu: ');readln(name);
  assign(f,name);reset(f); read(f,n,w);
  for j:=1 to n do  read(f,a[j]);
  for j:=1 to n do  read(f,c[j]);
  close(f);
  writeln('Trong luong cai tui: ',w:1:0);
  writeln('VEC TO GIA TRI DO VAT');
  for j:=1 to n do write(c[j]:4:0);  writeln;
  writeln('VEC TO TRONG LUONG DO VAT');
  for j:=1 to n do write(a[j]:4:0);  writeln;
end;

procedure Ghinhan_ky_luc;
var i: integer;
begin
  if cost > fopt then
  begin
    xopt:=x; fopt:=cost;
  end;
end;
```

```
procedure Branch_and_Bound(i:integer);
var j,t:integer;
begin
    t:=trunc((w-weight)/a[i]);
  for j:=t downto 0 do
  begin
    x[i]:=j;
    weight:=weight+a[i]*x[i];
    cost:=cost+c[i]*x[i];
    if  i=n then Ghinhan_ky_luc
    else
      if cost + c[i+1]*(w-weight)/a[i+1] > fopt
      then Branch_and_Bound(i+1);
    weight:=weight-a[i]*x[i];
    cost:=cost-c[i]*x[i];
  end;
end;

procedure Inkq;
var j,i:integer;
begin
  writeln('********** KET QUA TINH TOAN **********');
  writeln('Tong gia tri cua cac do vat: ',fopt:1:0);
  writeln('Phuong an dem do: ');
  for j:=1 to n do writeln('So luong do vat ', ind[j],': ',xopt[j]:4);
  writeln('*****************************************');
  write('Go Enter de tiep tuc...');readln;
end;

BEGIN
  clrscr;
  Readfile;
  Init;
  Branch_and_Bound(1);
  Inkq; readln;
END.
```

b) Bài toán người du lịch.

Mô hình của bài toán đã trình bày trong mục trước. Cố định thành phố xuất phát là T_1, bài toán người du lịch dẫn về bài toán:

Tìm cực tiểu của hàm

$$f(x_2, x_3, \ldots, x_n) = c[1,x_2] + c[x_2, x_3] + \ldots + c[x_{n-1}, x_n] + c[x_n, 1] \to \min,$$

với điều kiện

$(x_2, x_3, \ldots, x_n)$ là hoán vị của các số 2, 3, ..., n.

Ký hiệu

$$c_{min} = \min \{ c[i, j] , i, j = 1, 2, \ldots, n, i \neq j \}$$

là chi phí đi lại nhỏ nhất giữa các thành phố.

Để có được thuật toán nhánh cận giải bài toán người du lịch, trước hết cần đưa ra cách đánh giá cận dưới cho phương án bộ phận. Giả sử ta đang có phương án bộ phận $(u_1, u_2, \ldots, u_k)$. Phương án này tương ứng với hành trình bộ phận qua k thành phố:

$$T_1 \to T(u_2) \to \ldots \to T(u_{k-1}) \to T(u_k).$$

Vì vậy, chi phí phải trả theo hành trình bộ phận này sẽ là

$$\sigma = c[1,u_2] + c[u_2, u_3] + \ldots + c[u_{k-1}, u_k].$$

Để phát triển hành trình bộ phận này thành hành trình đầy đủ, ta còn phải đi qua n-k thành phố còn lại rồi quay trở về thành phố T_1, tức là còn phải đi qua n-k+1 đoạn đường nữa. Do chi phí phải trả cho việc đi qua mỗi một trong số n-k+1 đoạn đường còn lại đều không ít hơn c_{min}, nên cận dưới cho phương án bộ phận $(u_1, u_2, \ldots, u_k)$ có thể tính theo công thức

$$g(u_1, u_2, \ldots, u_k) = \sigma + (n\text{-}k\text{+}1)\, c_{min} .$$

Sử dụng cách tính cận dưới vừa nêu ta có thể áp dụng thuật toán nhánh cận để giải bài toán người du lịch.

Thí dụ. Giải bài toán người du lịch với ma trận chi phí sau

$$C = \begin{vmatrix} 0 & 3 & 14 & 18 & 15 \\ 3 & 0 & 4 & 22 & 20 \\ 17 & 9 & 0 & 16 & 4 \\ 6 & 2 & 7 & 0 & 12 \\ 9 & 15 & 11 & 5 & 0 \end{vmatrix}$$

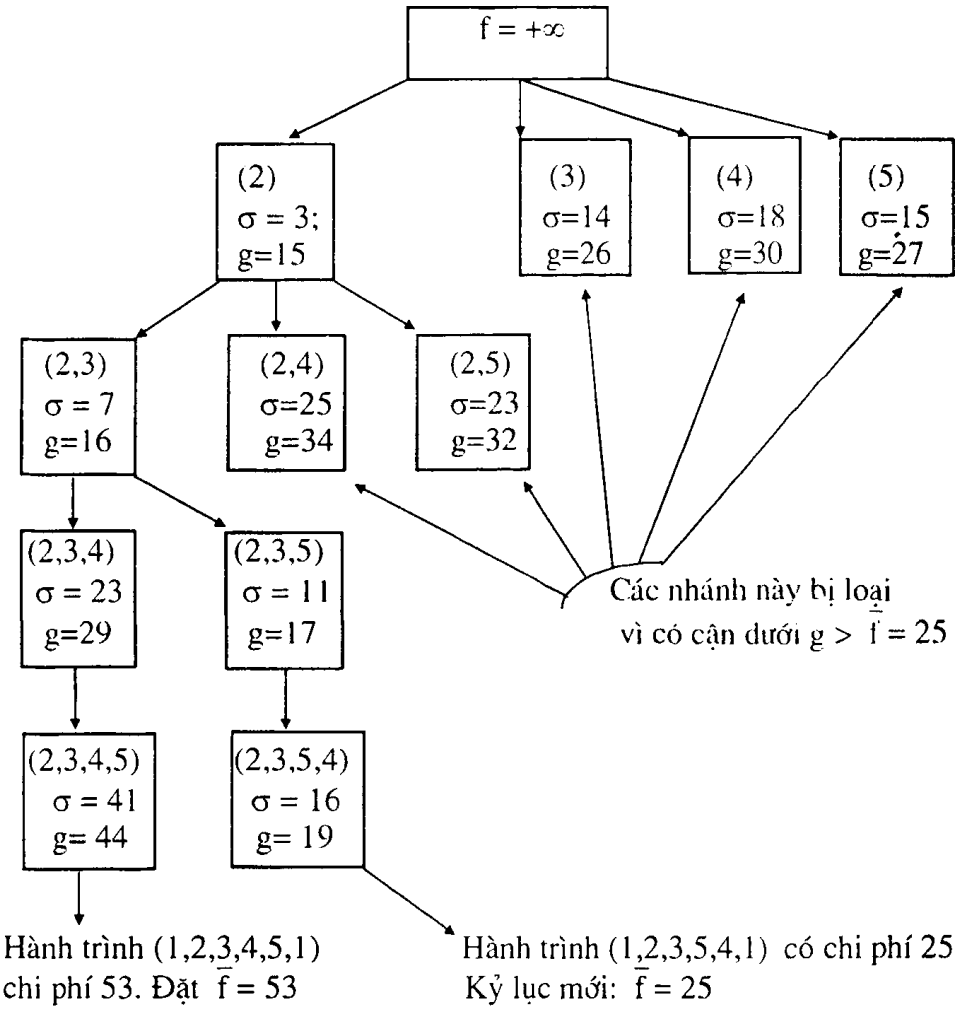
Ta có $c_{min} = 3$. Quá trình thực hiện thuật toán được mô tả bởi cây tìm kiếm lời giải cho trong hình 2.

Thông tin về một phương án bộ phận trên cây được ghi trong các ô trên hình vẽ tương ứng theo thứ tự sau: đầu tiên là các thành phần của phương án, tiếp đến, σ là chi phí theo hành trình bộ phận và g - cận dưới.

Kết thúc thuật toán, ta thu được phương án tối ưu (1, 2, 3, 5, 4, 1) tương ứng với hành trình

$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_5 \rightarrow T_4 \rightarrow T_1$$

và chi nhỏ nhất là 25.



Hình 2. Giải bài toán người du lịch

Chương trình trên PASCAL thực hiện thuật toán có thể viết như sau:

```
uses crt;
var
   c                : array[1..20,1..20] of integer;
   a, xopt          : array[1..20] of integer;
   chuaxet          : array[1..20] of boolean;
   n,fopt, cmin, Can : integer;

procedure Readfile;
var  f:text;
    name:string;
    i,j :integer;
begin
   write('Cho ten file du lieu: ');
   readln(name);
   assign(f,name);reset(f);
   read(f,n);
   for i:=1 to n do
     for j:=1 to n do  read(f,c[i,j]);
   close(f);
   cmin:=maxint;
   for i:=1 to n do
     for j:=1 to n do
        if (i<>j) and (cmin>c[i,j]) then cmin:=c[i,j];
end;

procedure Ghinhan;
var sum:integer;
begin
   sum:= Can+c[a[n],a[1]];
   if sum < fopt then
   begin
     xopt:=a;
     fopt:=sum;
   end;
end;

procedure Try(i:integer);
var j:integer;
begin
```

```
  (* Co dinh thanh pho xuat phat la Thanh pho 1 *)
  (* Duyet (n-1)! hanh trinh theo nhanh can    *)
  for j:=2 to n do
    if chuaxet[j] then
    begin
      a[i]:=j;
      chuaxet[j]:=false;
      can:=can+c[a[i-1],a[i]];
      if i=n then Ghinhan
      else
        if Can+(n-i+1)*cmin <fopt then Try(i+1);
      can:=can-c[a[i-1],a[i]];
      chuaxet[j]:=true;
    end;
end;

procedure Inkq;
var i,j:integer;
begin
  writeln('MA TRAN CHI PHI');
  for i:=1 to n do
  begin
    for j:=1 to n do  write(c[i,j]:4);
    writeln;
  end;
  writeln('Hanh trinh toi uu co chi phi : 'fopt);
  for i:=1 to n do write(xopt[i],' --> ');
  writeln(xopt[1]);
  write('Go Enter de tiep tuc...');readln;
end;

procedure Init;
var i,j : integer;
begin
  cmin:=maxint;
  for i:=1 to n do
  begin
    chuaxet[i]:=true;
    for j:=1 to n do
```

```
        if (i<>j) and (cmin>c[i,j]) then cmin:=c[i,j];
    end;
    fopt:=maxint;
    Can:=0;
    a[1]:=1;
end;

BEGIN
    Readfile; Init;
    Try(2);
    Inkq;
END.
```

Hiệu quả của thuật toán nhánh cận phụ thuộc rất nhiều vào việc xây dựng hàm tính cận dưới. Việc xây dựng hàm tính cận dưới lại phụ thuộc vào cách xây dựng thủ tục duyệt các phương án của bài toán (được gọi là thủ tục phân nhánh). Trên đây chúng ta trình bày cách xây dựng cận dưới khá đơn giản cho hai bài toán nổi tiếng của tối ưu tổ hợp. Các chương trình được cài đặt theo các thuật toán đó, tuy rằng làm việc tốt hơn nhiều so với duyệt toàn bộ, nhưng cũng chỉ có thể áp dụng để giải các bài toán với kích thước nhỏ. Muốn giải được các bài toán đặt ra với kích thước lớn hơn cần có cách đánh giá cận tốt hơn. Một trong những phương pháp xây dựng trên tư tưởng của thuật toán nhánh cận cho phép giải bài toán người du lịch với kích thước lớn hơn sẽ được trình bày trong mục tiếp theo.

5.3. Thuật toán nhánh cận giải bài toán người du lịch

Thuật toán nhánh cận là một trong những phương pháp giải chủ yếu của tối ưu tổ hợp. Như trong mục trước đã thấy, tư tưởng cơ bản của nó là trong quá trình tìm kiếm lời giải ta sẽ phân hoạch tập các phương án của bài toán ra thành hai hay nhiều tập con được biểu diễn như là các nút của cây tìm kiếm và cố gắng bằng phép đánh giá cận cho các nút, tìm cách loại bỏ những nhánh của cây tìm kiếm (những tập con các phương án của bài toán) mà ta biết chắc chắn là không chứa phương án tối ưu. Mặc dù trong tình huống tồi nhất thuật toán sẽ trở thành duyệt toàn bộ, nhưng trong nhiều trường hợp cụ thể, kỹ thuật đó cho phép rút ngắn được một cách đáng kể quá trình tìm kiếm. Mục này sẽ trình bày một cách thể hiện khác những tư tưởng của thuật toán nhánh cận vào việc xây dựng thuật toán giải bài toán người du lịch.

Xét bài toán người du lịch phát biểu trong mục trước. Gọi

$$C = \{ c_{ij} : i, j = 1, 2, ..., n \}$$

là ma trận chi phí. Mỗi hành trình của người du lịch

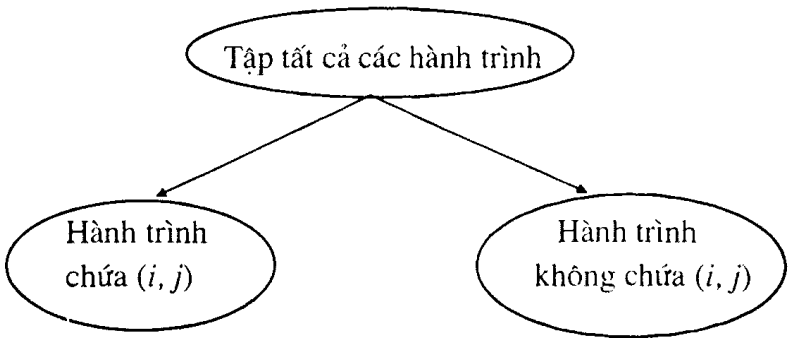
$$T_{\pi(1)} \to T_{\pi(2)} \to ... \to T_{\pi(n)} \to T_{\pi(1)},$$

có thể viết lại dưới dạng

$$(\pi(1), \pi(2)), (\pi(2), \pi(3)), ..., (\pi(n-1), \pi(n)), (\pi(n), \pi(1)),$$

trong đó mỗi thành phần $(\pi(j-1), \pi(j))$ sẽ được gọi là một cạnh của hành trình.

Trong bài toán người du lịch khi tiến hành tìm kiếm lời giải chúng ta sẽ phân tập các hành trình ra thành hai tập con: một tập gồm những hành trình chứa một cạnh (i, j) nào đó còn tập kia gồm những hành trình không chứa cạnh này. Ta gọi việc làm đó là *phân nhánh* và mỗi tập con nói trên sẽ được gọi là một nhánh hay một nút của cây tìm kiếm. Việc phân nhánh được minh hoạ bởi cây tìm kiếm:



Việc phân nhánh sẽ được thực hiện dựa trên một quy tắc oristic nào đó cho phép ta rút ngắn quá trình tìm kiếm phương án tối ưu. Sau khi phân nhánh ta sẽ tính cận dưới của giá trị hàm mục tiêu trên mỗi một trong hai tập con nói trên. Việc tìm kiếm sẽ được tiếp tục trên tập con có giá trị cận dưới nhỏ hơn. Thủ tục này sẽ được tiếp tục cho đến khi thu được một hành trình đầy đủ, tức là một phương án của bài toán người du lịch. Khi đó ta chỉ cần xét những tập con các phương án nào có cận dưới nhỏ hơn giá trị hàm mục tiêu tại phương án đã tìm được. Quá trình phân nhánh và tính cận trên tập các phương án của bài toán thông thường cho phép rút ngắn một cách đáng kể quá trình tìm kiếm do ta loại được khá nhiều tập con chắc chắn không chứa phương án tối ưu.

Một kỹ thuật cơ bản nữa của thuật toán là tính cận dưới sẽ được xây dựng dựa trên thủ tục rút gọn mà chúng ta sẽ trình bày dưới đây. Sau đó những bước chính của thuật toán nhánh cận sẽ được mô tả thông qua một ví dụ số và cuối cùng ta sẽ trình bày sơ đồ nguyên tắc của thuật toán.

a) Thủ tục rút gọn.

Rõ ràng tổng chi phí của một hành trình của người du lịch sẽ chứa đúng một phần tử của mỗi dòng và đúng một phần tử của mỗi cột trong ma trận chi phí C. Do đó, nếu ta trừ bớt mỗi phần tử của một dòng (hay cột) của ma trận C đi cùng một số α thì độ dài của tất cả các hành trình sẽ cùng giảm đi α, vì thế hành trình tối ưu cũng sẽ không thay đổi. Vì vậy nếu ta tiến hành trừ bớt các phần tử của mỗi dòng và mỗi cột đi một hằng số sao cho thu được ma trận gồm các phần tử không âm mà trong mỗi dòng và mỗi cột của nó đều có ít nhất một số 0 thì tổng các hằng số trừ đó sẽ cho ta cận dưới của mọi hành trình. Thủ tục trừ bớt này sẽ được gọi là thủ tục rút gọn, các hằng số trừ ở mỗi dòng (cột) sẽ được gọi là hằng số rút gọn theo dòng (cột), còn ma trận thu được sẽ gọi là ma trận rút gọn. Hàm số sau đây cho phép rút gọn một ma trận A kích thước là $k \times k$ đồng thời tính tổng các hằng số rút gọn (để tiện trình bày, các tham số có mặt trong các hàm và các thủ tục PASCAL dưới đây được giả thiết khai báo sao cho phù hợp):

```
function Reduce(A, k): real;
    (***   Thủ tục rút gọn ma trận   ***)
begin
  sum := 0;
  for i := 1 to k do  (* k - kích thước của A  *)
  begin
    r[i]:= <phần tử nhỏ nhất trong dòng i>;
    if r[i] > 0 then
    begin
      <Bớt mỗi phần tử của dòng i đi r[i]>;
      sum := sum + r[i];
    end;
  end;
  for j := 1 to k do
  begin
    s[j] := <phần tử nhỏ nhất trong cột j>;
    if s[j] > 0 then
    begin
      <Bớt mỗi phần tử của cột j đi s[j]>;
      sum := sum + s[j];
    end;
  end;
  Reduce := sum;
end;
```

Thí dụ. Ta có ma trận chi phí của bài toán người du lịch với $n = 6$ thành phố sau

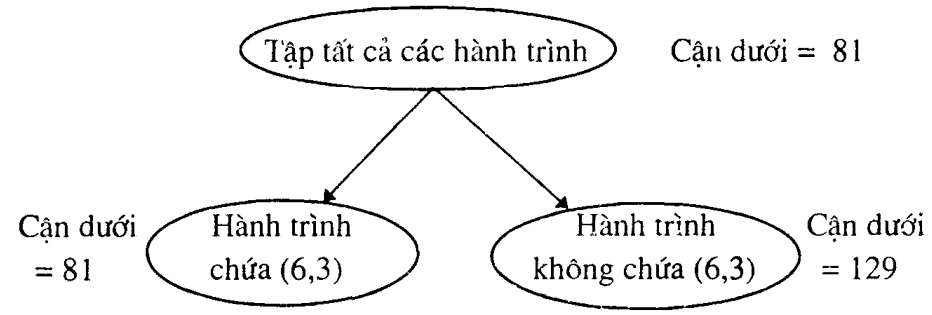
	1	2	3	4	5	6	r[i]
1	∞	3	93	13	33	9	3
2	4	∞	77	42	21	16	4
3	45	17	∞	36	16	28	16
4	39	90	80	∞	56	7	7
5	28	46	88	33	∞	25	25
6	3	88	18	46	92	∞	3
s[j]	0	0	15	8	0	0	

Đầu tiên trừ bớt mỗi phần tử của các dòng 1, 2, 3, 4, 5, 6 cho các hằng số rút gọn tương ứng là 3, 4, 16, 7, 25, 3, sau đó trong ma trận thu được, trừ bớt các phần tử của các cột 3 và 4 cho các hằng số rút gọn tương ứng là 15 và 8, ta thu được ma trận rút gọn sau

	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	0	35	89	∞

Tổng các hằng số rút gọn là 81, vì vậy cận dưới cho tất cả các hành trình là 81 (nghĩa là không thể tìm được hành trình có tổng chi phí nhỏ hơn 81).

Bây giờ, ta xét cách phân tập các phương án ra thành hai tập. Giả sử là ta chọn cạnh (6, 3) để phân nhánh. Khi đó tập các hành trình sẽ được phân thành hai tập con, một tập là các hành trình chứa cạnh (6, 3), còn tập kia là các hành trình không chứa cạnh (6, 3). Vì biết cạnh (6, 3) là không được tham gia vào hành trình, nên ta có thể cấm việc đi theo cạnh này bằng cách đặt $c_{63} = \infty$. Ma trận thu được sẽ có thể rút gọn bằng cách bớt mỗi phần tử của cột 3 đi 48 và không bớt gì các phần tử của dòng 6. Như vậy ta thu được cận dưới cho các hành trình không chứa cạnh (6, 3) là 81 + 48 = 129. Còn đối với tập các hành trình chứa cạnh (6, 3) ta phải loại dòng 6 và cột 3 khỏi ma trận tương ứng với nó, bởi vì đã đi theo cạnh (6, 3) thì không thể đi từ 6 sang bất cứ nơi nào khác và cũng không được phép đi từ bất cứ đâu vào 3. Kết quả ta thu được ma trận với bậc giảm đi 1. Ngoài ra, do đã đi theo cạnh (6, 3) nên không được phép đi từ 3 đến 6 nữa, vì vậy ta cần cấm đi theo cạnh (3, 6) bằng cách đặt $c_{3,6} = \infty$. Cây tìm kiếm cho đến bước này, có dạng cho trong hình 1 sau đây:



	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	0
5	3	21	0	∞	0

	1	2	3	4	5	6
1	∞	0	27	2	30	6
2	0	∞	10	30	17	12
3	29	1	∞	12	0	12
4	32	83	10	∞	49	0
5	3	21	0	0	∞	0
6	0	85	∞	35	89	∞

Hình 1.

Cạnh (6, 3) được chọn để phân nhánh vì phân nhánh theo nó ta thu được cận dưới của nhánh bên phải là lớn nhất so với việc phân nhánh theo các cạnh khác. Quy tắc này sẽ được sử dụng để phân nhánh ở mỗi đỉnh của cây tìm kiếm. Trong quá trình tìm kiếm chúng ta luôn đi theo nhánh bên trái trước. Nhánh bên trái sẽ có ma trận rút gọn với bậc giảm đi một. Trong ma trận của nhánh bên phải ta thay một số bởi ∞, và có thể rút gọn thêm được ma trận này khi tính lại các hằng số rút gọn theo dòng và cột tương ứng với cạnh phân nhánh, nhưng kích thước của ma trận vẫn giữ nguyên.

Do cạnh cần chọn để phân nhánh phải là cạnh làm tăng cận dưới của nhánh bên phải lên nhiều nhất, nên để tìm nó ta sẽ chọn số không nào trong ma trận mà khi thay nó bởi ∞ sẽ cho ta tổng hằng số rút gọn theo dòng và cột chứa nó là lớn nhất. Ta có thủ tục sau đây để chọn cạnh phân nhánh (r,c):

b) Thủ tục chọn cạnh phân nhánh (r,c)

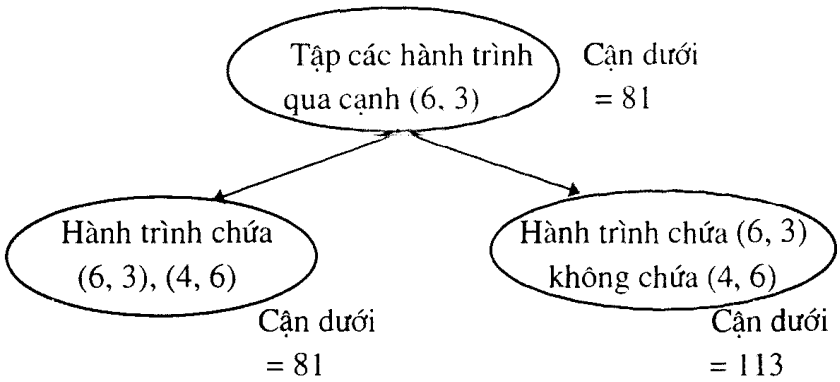
Đầu vào: Ma trận rút gọn A kích thước *k x k.*

Đầu ra : Cạnh phân nhánh (*r, c*) và tổng hằng số rút gọn theo dòng *r* cột *c* là beta.

```
procedure BestEdge(A,k,r,c,beta);
(* Thủ tục phân nhánh *)
begin
  beta := -∞;
  for i := 1 to k do
```

```
  for j := 1 to k do
    if a[i,j] = 0 then
    begin
      minr := <phần tử nhỏ nhất trên dòng i khác với a[i,j]>;
      minc := <phần tử nhỏ nhất trên cột j khác với a[i,j]>;
      total := minr + minc;
      if total > beta then
      begin
        beta := total;
        r := i; (* chỉ số dòng của cạnh tốt nhất *)
        c := j; (* chỉ số cột của cạnh tốt nhất *)
      end;
    end;
end;
```

Trong ma trận rút gọn 5 x 5 của nhánh bên trái ở hình 1, số không ở vị trí (4, 6) sẽ cho tổng hằng số rút gọn là 32 (theo dòng 4 là 32, cột 6 là 0). Đây là giá trị lớn nhất đối với các số không của ma trận này. Vì vậy, việc phân nhánh tiếp theo sẽ dựa vào cạnh (4, 6). Khi đó cận dưới của nhánh bên phải tương ứng với tập các hành trình đi qua cạnh (6,3) nhưng không đi qua (4,6) sẽ là 81 + 32 = 113. Còn nhánh bên trái sẽ tương ứng với ma trận 4 x 4, vì rằng ta phải loại bỏ dòng 4 và cột 6. Tình huống phân nhánh này được mô tả trong hình 2.



	1	2	4	5
1	∞	0	2	30
2	0	∞	30	17
4	29	1	∞	0
5	3	21	0	∞

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	0	51	∞	17	∞
5	3	21	0	∞	0

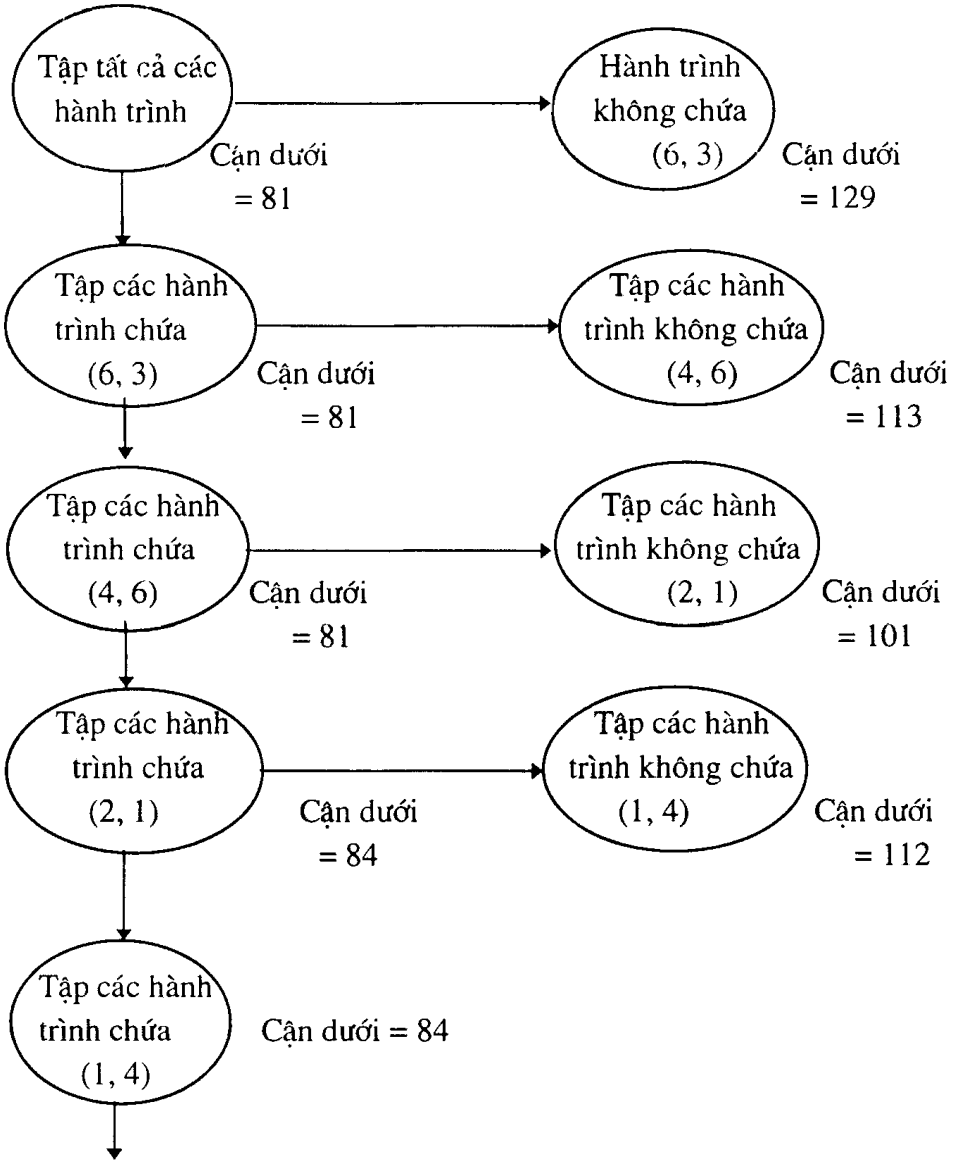
Hình 2.

Nhận thấy rằng vì cạnh (4, 6) và (6, 3) đã nằm trong hành trình nên cạnh (3, 4) không thể được đi qua nữa (nếu không ta sẽ có một hành trình con từ những thành phố này). Để ngăn ngừa việc tạo thành hành trình con ta sẽ gán cho phần tử ở vị trí (3, 4) giá trị ∞.

Ngăn cấm tạo thành hành trình con.

Tổng quát hơn, khi phân nhánh dựa vào cạnh (i_u, j_v) ta phải thêm cạnh này vào danh sách các cạnh của nút trái. Nếu i_u là đỉnh cuối của một đường đi $(i_1, i_2, \ldots, i_u)$ và j_v là đỉnh đầu của đường đi $(j_v, j_2, \ldots, j_k)$ thì để ngăn ngừa khả năng tạo thành hành trình con ta phải cấm cạnh (j_k, i_1).

Để tìm i_1 ta có thể đi ngược từ i_u, còn để tìm j_k ta có thể đi xuôi từ j_v theo danh sách các cạnh đã được kết nạp vào hành trình.



Hành trình (1, 4, 6, 3, 5, 2, 1). Độ dài hành trình: 104.

Hình 3.

Tiếp tục, ta lại phân nhánh từ đỉnh bên trái bằng cách sử dụng cạnh (2, 1), vì số không ở vị trí này có tổng các hằng số rút gọn là 17 + 3 = 20 (theo dòng 2 là 17, theo cột 1 là 3). Sau khi phân nhánh theo cạnh (2, 1) ma trận của nhánh trái có kích thước là 3 x 3. Vì đã đi qua (2, 1) nên ta cấm cạnh (1, 2) bằng cách đặt $c_{12} = \infty$, ta thu được ma trận sau

	2	4	5
1	∞	2	30
3	1	∞	0
5	21	0	∞

Ma trận này có thể rút gọn được bằng cách bớt 1 từ cột 2 và bớt đi 2 ở dòng 1. Điều đó dẫn đến ma trận

	2	4	5
1	∞	0	28
3	0	∞	0
5	20	0	∞

Ta có cận dưới của nhánh tương ứng là 81 + 1 + 2 = 84. Cây tìm kiếm cho đến bước này được thể hiện trong hình 3.

Chú ý rằng sau khi đã chấp nhận n-2 cạnh vào hành trình thì ma trận còn lại sẽ có kích thước là 2 x 2. Hai cạnh còn lại của hành trình sẽ không phải chọn lựa nữa, mà được kết nạp ngay vào chu trình. Trong ví dụ của chúng ta ở đây, sau khi đã có các cạnh (6, 3), (4, 6), (2, 1) và (1, 4) ma trận của nhánh trái có dạng

	2	5
3	∞	0
5	0	∞

vì vậy ta kết nạp nốt hai cạnh (3, 5) và (5, 2) vào và thu được hành trình 1, 4, 6, 3, 5, 2, 1 với chi phí là 104.

Chú ý. Trong quá trình tìm kiếm mỗi một nút của cây tìm kiếm sẽ tương ứng với một ma trận chi phí A. ở bước đầu tiên ma trận chi phí tương ứng với gốc chính là ma trận C. Khi chuyển động từ gốc theo nhánh bên trái xuống phía dưới kích thước của các ma trận chi phí A sẽ giảm dần. Cuối cùng khi ma trận A có kích thước 2x2 thì ta chấm dứt việc phân nhánh và kết nạp 2 cạnh còn lại để thu được hành trihnh của người du lịch. Dễ thấy rằng ma trận rút gọn cuối cùng này chỉ có thể có một trong hai dạng sau:

$$\begin{array}{c|cc|} & w & x \\ u & 0 & \infty \\ v & \infty & 0 \end{array} \quad , \quad \begin{array}{c|cc|} & w & x \\ u & \infty & 0 \\ v & 0 & \infty \end{array}$$

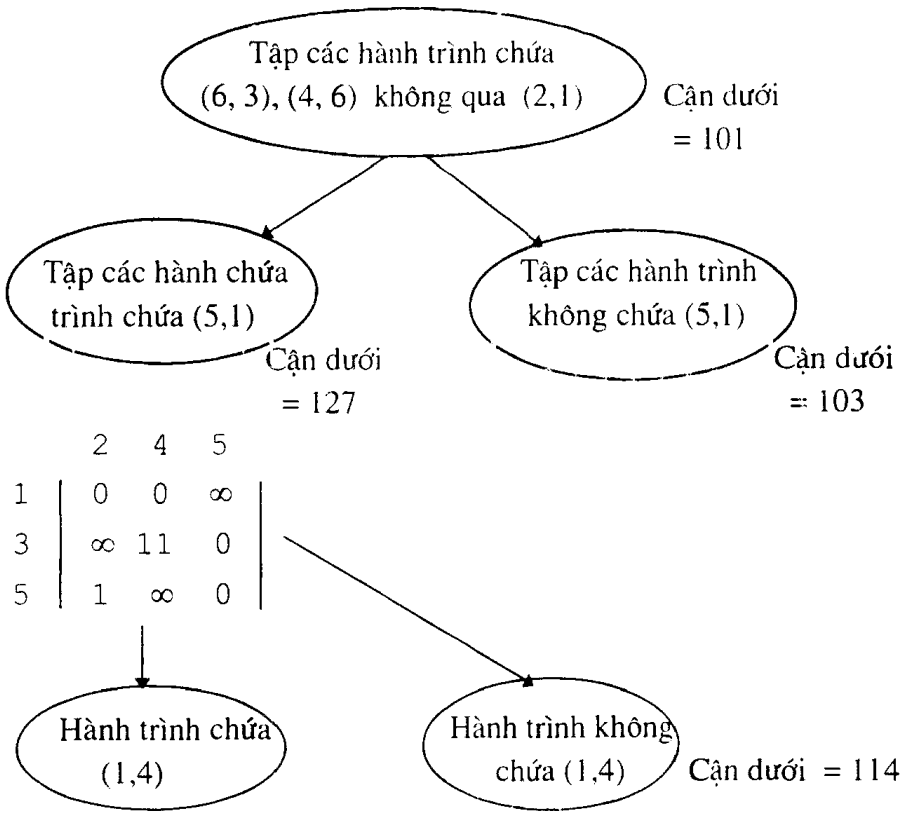
trong đó u, v, w, x có thể là 4 đỉnh khác nhau hoặc chỉ có 3 đỉnh khác nhau. Trong mọi trường hợp để xác định xem hai cạnh nào cần phải kết nạp nốt ta chỉ cần xét một phần tử của ma trận A:

```
if A[1,1}] = ∞ then
        Kết nạp cạnh (u,x) và (v,w)
else
        Kết nạp cạnh (u,w) và (v,x);
```

Bây giờ tất cả các nút của cây có cận dưới lớn hơn 104 có thể loại bỏ vì chúng không chứa hành trình rẻ hơn. Trên hình 3 ta thấy chỉ còn nút có cận dưới 101 là cần phải xét tiếp. Nút này tương ứng với tập những hành trình chứa các cạnh (6, 3), (4, 6) và không chứa cạnh (2, 1). Ma trận chi phí tương ứng với đỉnh này có dạng

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	26	1	∞	0
5	0	21	0	∞

Việc phân nhánh sẽ dựa vào cạnh (5, 1) với tổng hằng số rút gọn là 26. Việc rẽ nhánh tiếp theo từ nút này được cho trong hình 4.



Hành trình 1, 4, 6, 3, 2, 5, 1
Độ dài : 104

Hình 4.

Như vậy chúng ta thu được hai hành trình tối ưu với chi phí là 104. Thí dụ trên cho thấy rằng bài toán người du lịch có thể có nhiều phương án tối ưu. Trong thí dụ này hành trình đầu tiên tìm được đã là tối ưu, tất nhiên điều đó không thể trông đợi trong trường hợp tổng quát. Đối với thí dụ trên ta chỉ phải xét 13 nút, trong khi tổng số hành trình của người du lịch là 120.

c) Thuật toán nhánh cận giải bài toán người du lịch.

Bây giờ ta có thể mô tả các bước chính của thuật toán nhánh cận giải bài toán người du lịch trong thủ tục đệ quy TSP sau đây. Thủ tục TSP xét hành trình bộ phận với Edges cạnh đã được chọn và tiến hành tìm kiếm tiếp theo. Ta sử dụng các biến:

Edges - số cạnh trong hành trình bộ phận;
A - ma trận chi phí tương ứng với kích thước (n-edges) x (n-edges);
cost - chi phí của hành trình bộ phận;
MinCost - chi phí của hành trình tốt nhất đã tìm được.

Trong thủ tục sử dụng hàm **Reduce**(A, k) và thủ tục **BestEdge**(A, k, r, c, beta) đã mô tả ở trên.

```
procedure TSP(edges, cost, A);
begin
   cost:= cost + Reduce(A, n-edges);
   if cost < MinCost then
        if edges = n-2 then
        begin
            <Bổ sung nốt hai cạnh còn lại>;
            MinCost := Cost;
            <Ghi nhận hành trình tốt nhất>;
        end
        else
        begin
            BestEdge(A,n-edges,r,c,beta);
            LowerBound:= cost + beta;
            <Ngăn cấm tạo thành hành trình con>;
            NewA := <A loại bỏ dòng r cột c>
            TSP(edges+1,cost,NewA);       (* đi theo nhánh trái *)
            <Khôi phục A bằng cách bổ sung lại dòng r cột k>;
            if LowerBound < MinCost then
            begin                          (* đi theo nhánh phải *)
                A[r,c]:= ∞;
                TSP(edges, cost, A);
                A[r,c]:=0;
            end;
        end;
        <Khôi phục ma trận A> ;  (*  thêm lại các hằng số rút gọn
                                    vào các dòng và cột tương ứng *)
end; (* of TSP *)
```

5.4 Bài toán lập lịch gia công trên hai máy. Thuật toán Jonhson

Trong mục này ta sẽ thấy việc nghiên cứu kỹ các tính chất của bài toán tối ưu tổ hợp dẫn đến việc xây dựng được thuật toán rất hiệu quả để giải nó.

Xét bài toán lập lịch gia công trên 2 máy, là trường hợp riêng của bài toán lập lịch nêu trong mục 5.1: Mỗi một chi tiết trong số n chi tiết D_1 D_2,... D_n cần phải được lần lượt gia công trên 2 máy A, B. Thời gian gia công chi tiết D_i trên máy A là a_i, trên máy B là b_i (i = 1, 2,..., n). Hãy tìm lịch (trình tự gia công) các chi tiết trên hai máy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất có thể được.

Giả thiết rằng, trình tự gia công các chi tiết trên hai máy là như nhau. Khi đó mỗi lịch gia công sẽ tương ứng với một hoán vị

$$\pi = (\pi(1), \pi(2),..., \pi(n))$$

của n số tự nhiên 1, 2,..., n.

Ký hiệu s_{jX}, t_{jX} là thời điểm bắt đầu và kết thúc việc gia công chi tiết j trên máy X, j = 1, 2,..., n; X = A, B. Giả sử π là một lịch gia công. Theo điều kiện của bài toán, máy A có thể bắt đầu thực hiện công việc π(1) vào thời điểm $s_{\pi(1)} = 0$ và công việc $\pi(k)$ sau khi thực hiện xong công việc $\pi(k\text{-}1)$, tức là

$$s_{\pi(k)A} \geq t_{\pi(k-1)A}, \quad k = 2, 3,..., n. \tag{1}$$

Máy B có thể bắt đầu thực hiện công việc $\pi(1)$ ngay sau khi máy A kết thúc việc gia công nó tức là vào thời điểm

$$s_{\pi(1)B} \geq t_{\pi(1)A}. \tag{2}$$

Máy B có thể bắt đầu việc gia công chi tiết $\pi(k)$ (k = 2, 3,..., n) sau khi công việc này được thực hiện xong trên máy A và đồng thời nó phải hoàn thành việc gia công chi tiết $\pi(k\text{-}1)$, tức là:

$$s_{\pi(k)B} \geq \max (t_{\pi(k)A}, t_{\pi(k-1)B}), \quad k = 2, 3,..., n. \tag{3}$$

Thời gian để hoàn thành việc gia công tất cả các chi tiết trên hai máy là $T(\pi) = t_{\pi(n)B}$.

Rõ ràng, với π cố định, $T(\pi)$ đạt giá trị nhỏ nhất khi tất cả các dấu bất đẳng thức ở (1), (2), (3) được thay bởi dấu đẳng thức, tức là

$$\begin{aligned} &s_{\pi(1)A} = 0, \\ &s_{\pi(k)A} = t_{\pi(k-1)A}, \; k = 2, 3,..., n, \\ &s_{\pi(1)B} = t_{\pi(1)A}, \\ &s_{\pi(k)B} = \max (t_{\pi(k)A}, t_{\pi(k-1)B}), \quad k = 2, 3,..., n. \end{aligned} \tag{4}$$

nghĩa là các máy sẽ thực hiện ngay các công việc một khi điều kiện cho phép.

Thí dụ 1. Xét bài toán khi $n = 5$. Thời gian gia công các chi tiết trên các máy được cho trong bảng sau:

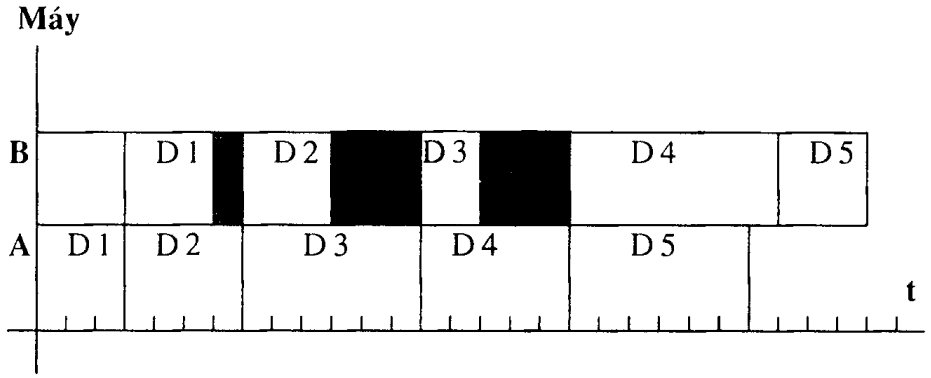
Chi tiết / Máy	D_1	D_2	D_3	D_4	D_5
A	3	4	6	5	6
B	3	3	2	7	3

Giả sử thực hiện việc gia công các chi tiết theo lịch $\pi = (1, 2, 3, 4, 5)$. Khi đó, theo các công thức (4) ta tính được

$s_{1A} = 0$; $t_{1A} = 3$;
$s_{2A} = 3$; $t_{2A} = 7$;
$s_{3A} = 7$; $t_{3A} = 13$;
$s_{4A} = 13$; $t_{4A} = 18$;
$s_{5A} = 18$; $t_{5A} = 24$;

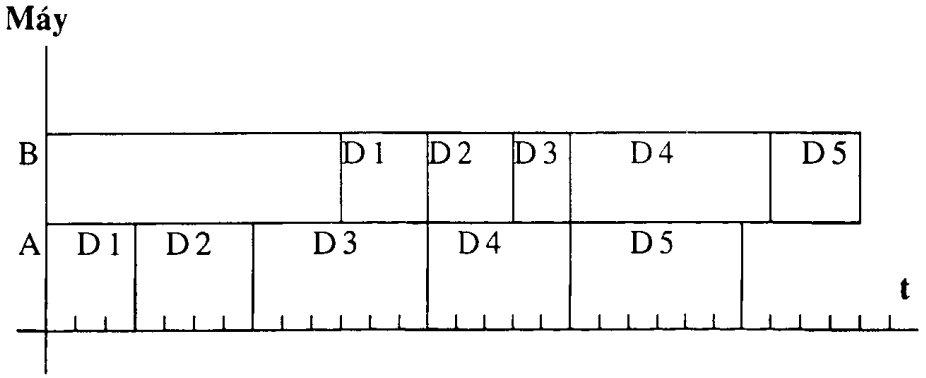
$s_{1B} = 3$; $t_{1B} = 6$;
$s_{2B} = 7$; $t_{2B} = 10$;
$s_{3B} = 13$; $t_{3B} = 15$;
$s_{4B} = 18$; $t_{4B} = 25$;
$s_{5B} = 25$; $t_{5B} = 28$.

Để biểu diễn lịch gia công người ta thường sử dụng sơ đồ Gantt, trong đó các máy được biểu thị theo trục tung, còn trục hoành để biểu diễn thời gian. Sơ đồ Gantt, theo lịch gia công thu được trong thí dụ đã cho, có dạng trong hình 1:



Hình 1.

Thời gian hoàn thành việc gia công tất cả các chi tiết theo lịch thu được là $T(\pi) = t_{5B} = 28$. Từ hình 5 nhận thấy rằng trong cách bố trí máy B thực hiện việc gia công các chi tiết theo lịch gia công π có nhiều khoảng thời gian máy chết (trên hình vẽ đánh dấu bằng cách tô mầu sẫm). Rõ ràng ta luôn có thể bố trí lại việc gia công của máy B sao cho không có các khoảng thời gian chết này bằng cách dồn chúng vào đoạn đầu để sau đó máy B hoạt động liên tục và việc này không làm tăng thời gian hoàn thành việc gia công (giá trị $t_{\pi(n)B}$). Chẳng hạn, để thoát khỏi các khoảng thời gian chết của máy B trong thí dụ, ta có thể bắt đầu gia công trên máy B vào thời điểm $d_B = 10$ (tức là bằng tổng các khoảng thời gian chết trong hình 5, cộng với $t_{\pi(1)A}$). Sơ đồ Gantt của cách bố trí này cho trong hình 2:



Hình 2.

Vì vậy luôn có thể giả thiết rằng, hai máy sẽ thực hiện việc gia công một cách liên tục. Máy A bắt đầu thực hiện việc gia công vào thời điểm $d_A = 0$. Gọi d_B là thời điểm máy B bắt đầu thực hiện việc gia công các chi tiết. Rõ ràng ta có

$$T(\pi) = d_B(\pi) + \sum_{j=1}^{n} b_j,$$

trong đó số hạng thứ hai là không phụ thuộc vào lịch gia công π. Ta cần tìm công thức tính $d_B(\pi)$. Dễ thấy là $d_B(\pi)$ là bằng tổng của $t_{\pi(1)A}$ và các khoảng thời gian chết của máy B khi ta bố trí máy B thực hiện việc gia công các chi tiết theo công thức (4). Vì thế, ta có công thức sau đây, để tính $d_B(\pi)$:

$$d_B(\pi) = \max_{1 \le u \le n} \Delta_u(\pi)$$

trong đó

$$\Delta_1(\pi) = a_{\pi(1)},$$

$$\Delta_u(\pi) = \sum_{j=1}^{u} a_{\pi(j)} - \sum_{j=1}^{u-1} b_{\pi(j)}, \qquad u = 2, ..., n.$$

Trong thí dụ 1, ta có

$$\Delta_1(\pi) = 3;$$
$$\Delta_2(\pi) = 3 + 4 - 3 = 4;$$
$$\Delta_3(\pi) = (3 + 4 + 6) - (3 + 3) = 7;$$
$$\Delta_4(\pi) = (3 + 4 + 6 + 5) - (3 + 3 + 2) = 10;$$
$$\Delta_5(\pi) = (3 + 4 + 6 + 5 + 6) - (3 + 3 + 2 + 7) = 9;$$

Vậy $d_B(\pi) = 10$.

Như vậy bài toán đặt ra dẫn về bài toán tối ưu tổ hợp sau

$$\min \{ d_B(\pi) : \pi \in P \}.$$

trong đó P là tập các hoán vị của 1, 2, ..., n.

Bổ đề 1. *Giả sử* $\pi = (\pi(1),..., \pi(k\text{-}1), \pi(k), \pi(k\text{+}1),..., \pi(n))$ *là một lịch gia công còn* π' *là lịch gia công thu được từ* π *bằng cách hoán vị hai phần tử* $\pi(k)$ *và* $\pi(k\text{+}1)$:

$$\pi' = (\pi(1),..., \pi(k\text{-}1), \pi(k\text{+}1), \pi(k),..., \pi(n)).$$

Khi đó nếu

$$\min (a_{\pi(k)}, b_{\pi(k+1)}) \le \min (b_{\pi(k)}, a_{\pi(k+1)}) \qquad (5)$$

thì

$$d_B(\pi) \le d_B(\pi'). \qquad (6)$$

Chứng minh. Do π, π' chỉ khác nhau ở vị trí thứ k và k+1 nên ta có

$$\Delta_u(\pi) = \Delta_u(\pi') \quad \text{với} \quad u = 1,..., k\text{-}1, k\text{+}2,..., n.$$

Từ đó để chứng minh (6) ta chỉ cần chứng tỏ

$$\max (\Delta_k(\pi), \Delta_{k+1}(\pi)) \le \max (\Delta_k(\pi'), \Delta_{k+1}(\pi')) \qquad (7)$$

Thật vậy, bất đẳng thức (7) tương đương với

$$\max (\Delta_k(\pi) - \delta, \Delta_{k+1}(\pi) - \delta) \le \max (\Delta_k(\pi') - \delta, \Delta_{k+1}(\pi') - \delta) \qquad (8)$$

với mọi giá trị δ. Chọn

$$\delta = \sum_{j=1}^{k+1} a_{\pi(j)} - \sum_{j=1}^{k-1} b_{\pi(j)}$$

và để ý rằng

$$\Delta_k(\pi) = \sum_{j=1}^{k} a_{\pi(j)} - \sum_{j=1}^{k-1} b_{\pi(j)},$$

ta nhận được (8) dưới dạng

$$\max(-a_{\pi(k+1)}, -b_{\pi(k)}) \le \max(-a_{\pi(k)}, -b_{\pi(k+1)})$$
$$\Leftrightarrow -\min(a_{\pi(k+1)}, b_{\pi(k)}) \le -\min(a_{\pi(k)}, b_{\pi(k+1)})$$
$$\Leftrightarrow \min(a_{\pi(k)}, b_{\pi(k+1)}) \le \min(b_{\pi(k)}, a_{\pi(j+1)})$$

nghĩa là (7) tương đương với (5). Bổ đề được chứng minh.

Bổ đề 2. *Nếu i, j, k là ba chỉ số thoả mãn*

$$\min(a_i, b_j) \le \min(a_j, b_i) \quad (9)$$
$$\min(a_j, b_k) \le \min(a_k, b_j) \quad (10)$$

trong đó có ít nhất một trong hai bất đẳng thức trên là bất đẳng thức chặt, thì

$$\min(a_i, b_k) \le \min(a_k, b_i) \quad (11)$$

Chứng minh. Giả sử trong (9) ta có $a_i \le b_j$ và $a_j \le b_i$, còn trong (10) ta có $a_j \le b_k$ và $a_k \le b_j$. Khi đó từ (9) suy ra $a_i \le a_j$, còn từ (10) suy ra $a_j \le a_k$. Tức là ta có $a_i \le b_i$, $a_i \le b_k$ và $a_i \le a_k$, từ đó suy ra có (11).

Chứng minh tương tự cho 15 trường hợp còn lại, ta thu được bổ đề.

Định lý Johnson (1954). *$T(\pi)$ đạt giá trị nhỏ nhất khi lịch gia công $\pi = (\pi(1), \pi(2),..., \pi(n))$ thoả mãn*

$$\min(a_{\pi(k)}, b_{\pi(k+1)}) \le \min(b_{\pi(k)}, a_{\pi(k+1)}) \quad (12)$$

với mọi $k = 1, 2,..., n-1$.

Chứng minh. Thực vậy giả sử

$$\pi' = (\pi'(1), \pi'(2),..., \pi'(n))$$

là lịch gia công tối ưu. Nếu π' không thoả mãn (12), thì theo bổ đề 1, khi thay đổi vị trí của hai phần tử liền nhau tương ứng trong nó, ta thu được lịch gia công mới π với $d_B(\pi)$ không lớn hơn $d_B(\pi')$. Quá trình này được lặp lại đối với π cho đến khi thu được lịch thoả mãn (12). Bổ đề 2 đảm bảo được rằng việc lặp như thế là kết thúc. Định lý được chứng minh.

Định lý vừa chứng minh chỉ cho chúng ta cơ sở xây dựng thuật toán giải bài toán đặt ra. Giả sử

$$x = \min_{1 \le i \le n} (a_i, b_i).$$

Xét hai trường hợp:

1) Nếu $x = a_k$ với một k nào đó thì ta có $\min(a_k, b_j) \le \min(b_k, a_j)$ với mọi $j \ne k$. Vì thế chi tiết D_k phải được gia công đầu tiên trong lịch tối ưu.

2) Nếu $x = b_p$ với một p nào đó thì ta có $\min(a_p, b_j) \geq \min(b_p, a_j)$ với mọi $j \neq p$. Vì thế chi tiết D_p phải được gia công cuối cùng trong lịch tối ưu.

Từ đó nhận được thuật toán sau đây

Thuật toán JOHNSON

1. Chia các chi tiết thành 2 nhóm: nhóm N_1 gồm các chi tiết D_i thoả mãn $a_i < b_i$, tức là $\min(a_i, b_i)$ đạt được tại a_i, và nhóm N_2 gồm các chi tiết D_i thoả mãn $a_i > b_i$, tức là $\min(a_i, b_i)$ đạt được tại b_i. Các chi tiết D_i thoả mãn $a_i = b_i$ xếp vào nhóm nào cũng được.
2. Sắp xếp các chi tiết trong N_1 theo chiều tăng của các a_i và sắp xếp các chi tiết trong N_2 theo chiều giảm của các b_i.
3. Nối N_2 vào đuôi N_1. Dãy thu được (đọc từ trái sang phải) sẽ là lịch gia công tối ưu.

Quay trở lại giải bài toán trong thí dụ1 theo thuật toán Johnson: Các kết quả được tính trong từng bước như sau:

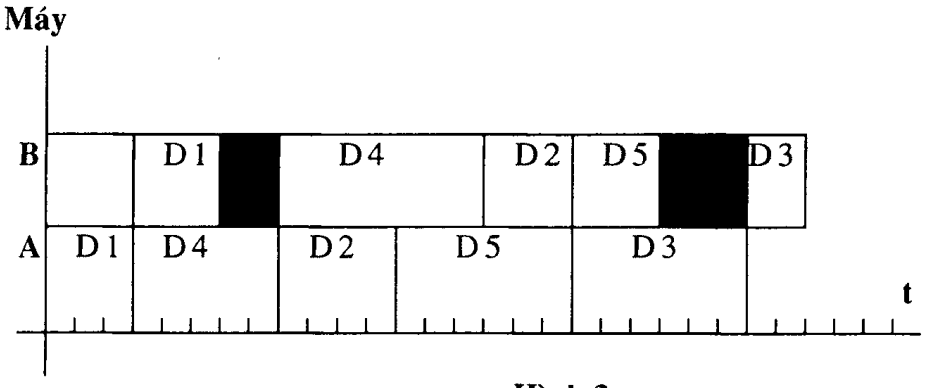
1. Chia nhóm: $N_1 = \{D_1, D_4\}$; $N_2 = \{D_2, D_3, D_5\}$

2. Sắp xếp N_1 theo chiều tăng của các a_i và sắp xếp N_2 theo chiều giảm của các b_i: $N_1 = (D_1, D_4)$; $N_2 = (D_2, D_5, D_3)$

3. Nối N_2 vào đuôi của N_1, ta được lịch gia công tối ưu:

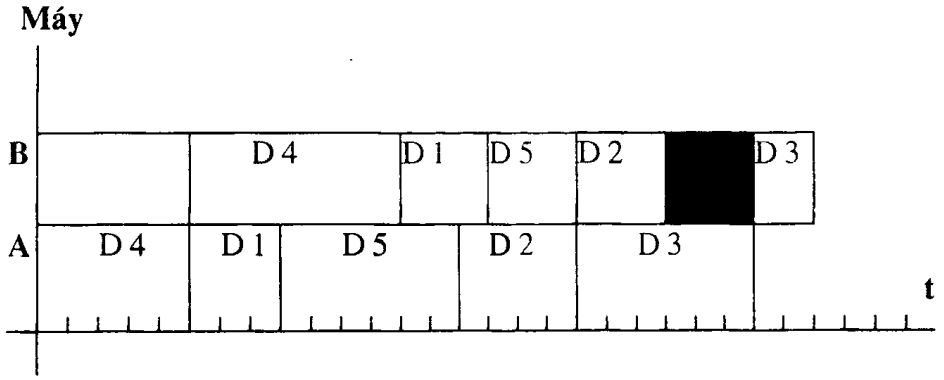
$$\pi = (D_1, D_4, D_2, D_5, D_3).$$

Sơ đồ Gantt của lịch này được cho bởi hình 3 với thời gian hoàn thành việc gia công là $T(\pi) = 26$:



Hình 3.

Rõ ràng có nhiều lịch tối ưu, chúng có thể khác nhau về thời điểm bắt đầu của máy B nhưng đều chung nhau một thời điểm kết thúc. Chẳng hạn một lịch tối ưu khác của thí dụ trên là $\pi' = (D_4, D_1, D_5, D_2, D_3)$ với sơ đồ Gantt cho bởi hình 4:



Hình 4.

Chú ý.

1) Có thể chứng minh được rằng việc tìm lịch gia công dưới dạng mỗi máy một trình tự gia công riêng không dẫn tới việc hoàn thành gia công các chi tiết sớm hơn. Vì vậy, thuật toán Johnson vẫn cho kết quả đúng của bài toán mà không cần có giả thiết rằng trình tự gia công trên hai máy phải như nhau.

2) Kỹ thuật chứng minh định lý Johnson dựa trên bổ đề 1 là một kỹ thuật cơ bản trong lý thuyết lập lịch, nó được biết dưới tên gọi: *Thủ thuật hoán vị*.

3) Không thể thu được định lý tương tự như định lý Johnson cho trường hợp bài toán 3 máy hoặc nhiều hơn. Trong trường hợp tổng quát, hiện nay chưa có phương pháp hữu hiệu nào để giải chúng ngoài việc sử dụng phương pháp nhánh cận.

Một số trường hợp riêng có thể dẫn về bài toán 2 máy.

Xét bài toán gia công n chi tiết trên 3 máy theo thứ tự A, B, C với bảng thời gian a_i, b_i, c_i, $i = 1, 2,..., n$, thoả mãn:

$$\max_i b_i \le \min_i a_i \quad \text{hoặc} \quad \max_i b_i \le \min_i c_i$$

tức là thời gian gia công của máy B khá nhỏ so với A hoặc C.

Khi đó, lịch gia công tối ưu trên 3 máy sẽ trùng với lịch gia công tối ưu trên 2 máy: máy thứ nhất với thời gian $a_i + b_i$ và máy thứ hai với thời gian $b_i + c_i$ (để ý rằng chỉ có lịch tối ưu của chúng là trùng nhau còn thời gian gia công của chúng là khác nhau).

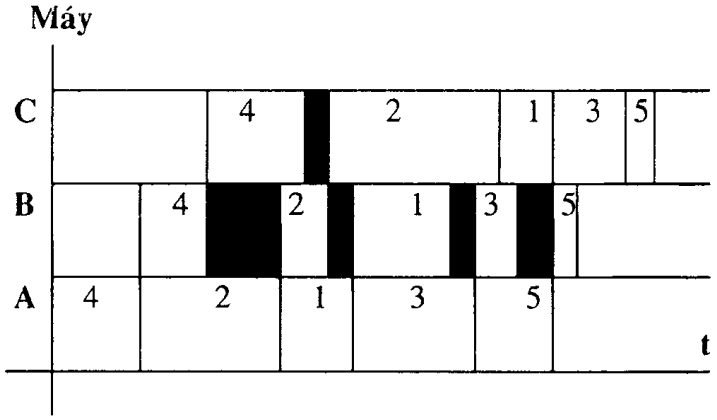
Thí dụ 2. Thời gian gia công 5 chi tiết trên các máy A, B, C cho bởi bảng sau:

Chi tiết / Máy	D_1	D_2	D_3	D_4	D_5
A	7	11	8	7	6
B	6	5	3	5	3
C	4	12	7	8	3

Thử lại $\max b_i = 6 \leq \min a_i = 6$, do đó bài toán được dẫn về việc tìm lịch gia công tối ưu trên 2 máy A', B':

Chi tiết / Máy	D_1	D_2	D_3	D_4	D_5
A'	13	16	11	12	9
B'	10	17	10	13	6

Lịch gia công tối ưu tìm được là $\pi = (D_4, D_2, D_1, D_3, D_5)$ với thời gian hoàn thành $T(\pi) = 49$ và sơ đồ Gantt cho bởi hình 5:



Hình 5.

Bài tập

1. Áp dụng thuật toán nhánh cận giải bài toán người du lịch với ma trận chi phí sau

a)

	A	B	C	D	E
A	0	8	5	22	11
B	4	0	9	17	27
C	15	7	0	12	35
D	5	27	17	0	29
E	23	21	19	7	0

b)

	A	B	C	D	E
A	0	5	37	21	29
B	42	0	31	7	33
C	31	27	0	31	8
D	49	33	14	0	39
E	6	41	32	38	0

c)

	A	B	C	D	E
A	0	8	5	22	11
B	4	0	9	17	27
C	15	7	0	12	35
D	5	27	17	0	29
E	23	21	19	7	0

Thành phố xuất phát là A. Quá trình giải theo thuật toán trình bày trên cây lời giải.

2. Giải các bài toán cái túi sau đây bằng thuật toán nhánh cận

a)

$$17x_1 + 8x_2 + 6x_3 + 3x_4 \to \max$$
$$7x_1 + 6x_2 + 4x_3 + 2x_4 \le 19$$
$$x_j \ge 0, \text{ nguyên}, j = 1, 2, 3, 4.$$

b)

$$16x_1 + 9x_2 + 7x_3 + 5x_4 \to \max$$
$$6x_1 + 5x_2 + 3x_3 + 2x_4 \le 17$$
$$x_j \ge 0, \text{ nguyên}, j = 1, 2, 3, 4.$$

c)

$$16x_1 + 8x_2 + 6x_3 + x_4 \to \max$$

$$7x_1 + 6x_2 + 4x_3 + x_4 \le 26$$
$$x_j \ge 0, \text{ nguyên}, j = 1, 2, 3, 4.$$

Quá trình thực hiện thuật toán mô tả trên cây tìm kiếm lời giải.

3. Mô tả thuật toán quay lui để giải bài toán sau:

Cho n số nguyên dương $a_1, a_2, ..., a_n$. Tìm các số $s_i \in \{-1, 1\}$, $i = 1, 2, ..., n$ sao cho

$$\left| \sum_{i=1}^{n} a_i s_i \right|$$

là nhỏ nhất.

4. Bất đẳng thức sau đây có tên là bất đẳng thức hoán vị được sử dụng trong việc phát triển thuật toán giải nhiều bài toán lập lịch: Cho hai dãy số thực $a_1, a_2, ..., a_n$ và $b_1, b_2, ..., b_n$, trong đó $b_1 \ge b_2 \ge ... \ge b_n$. Giả sử $(\sigma(1), \sigma(2), ..., \sigma(n))$ và $(\gamma(1), \gamma(2), ..., \gamma(n))$ là các hoán vị của các số 1, 2, ..., n thoả mãn

$$a_{\nu(1)} \ge a_{\nu(2)} \ge ... \ge a_{\nu(n)}, \quad a_{\sigma(1)} \le a_{\sigma(2)} \le ... \le a_{\sigma(n)}.$$

Khi đó bất đẳng thức

$$\sum_{i=1}^{n} a_{\sigma(i)} b_i \le \sum_{i=1}^{n} a_{\pi(i)} b_i \le \sum_{i=1}^{n} a_{\nu(i)} b_i,$$

được thực hiện với mọi hoán vị $(\pi(1), \pi(2), ..., \pi(n))$ của các số 1, 2, ..., n.

Như là ví dụ ứng dụng, xét bài toán lập lịch sau đây: Có n khách hàng đến thuê thực hiện chương trình trên máy tính song song tại trung tâm máy tính hiệu năng cao. Biết rằng thời gian cần thiết để chạy xong chương trình của khách hàng i là t_i, $i = 1, 2, ..., n$. Giả thiết là thời gian để máy chuyển từ việc thực hiện chương trình này sang thực hiện chương trình khác là bằng 0, hãy tìm trình tự thực hiện các chương trình sao cho tổng thời gian chờ đợi của tất cả n khách hàng là nhỏ nhất.

Sử dụng bất đẳng thức hoán vị để chỉ ra rằng trình tự cần tìm là trình tự không giảm của thời gian thực hiện các chương trình.

PHẦN II

LÝ THUYẾT ĐỒ THỊ

1

CÁC KHÁI NIỆM CƠ BẢN CỦA LÝ THUYẾT ĐỒ THỊ

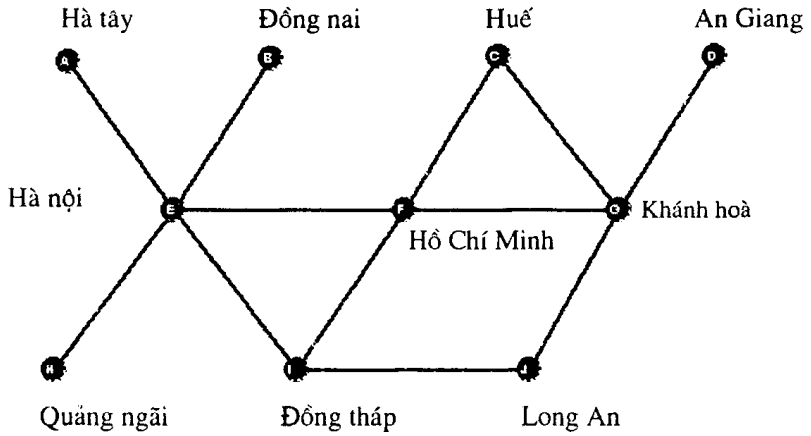
Lý thuyết đồ thị là một lĩnh vực nghiên cứu đã có từ lâu và có nhiều ứng dụng hiện đại. Những tư tưởng cơ bản của lý thuyết đồ thị được đề xuất vào những năm đầu của thế kỷ 18 bởi nhà toán học lỗi lạc người Thuỵ sỹ Leonhard Euler. Chính ông là người đã sử dụng đồ thị để giải bài toán nổi tiếng về các cái cầu ở thành phố Konigsberg.

Đồ thị được sử dụng để giải các bài toán trong nhiều lĩnh vực khác nhau. Chẳng hạn, đồ thị có thể sử dụng để xác định các mạch vòng trong vấn đề giải tích mạch điện. Chúng ta có thể phân biệt các hợp chất hoá học hữu cơ khác nhau với cùng công thức phân tử nhưng khác nhau về cấu trúc phân tử nhờ đồ thị. Chúng ta có thể xác định xem hai máy tính trong mạng có thể trao đổi thông tin được với nhau hay không nhờ mô hình đồ thị của mạng máy tính. Đồ thị có trọng số trên các cạnh có thể sử dụng để giải các bài toán như: Tìm đường đi ngắn nhất giữa hai thành phố trong một mạng giao thông. Chúng ta cũng còn sử dụng đồ thị để giải các bài toán về lập lịch, thời khoá biểu, và phân bố tần số cho các trạm phát thanh và truyền hình...

1.1. Định nghĩa đồ thị

Đồ thị là một cấu trúc rời rạc bao gồm các đỉnh và các cạnh nối các đỉnh này. Chúng ta phân biệt các loại đồ thị khác nhau bởi ***kiểu*** và ***số lượng*** cạnh nối hai đỉnh nào đó của đồ thị. Để có thể hình dung được tại sao lại cần đến các loại đồ thị khác nhau, chúng ta sẽ nêu ví dụ sử dụng chúng để mô tả một mạng máy tính. Giả sử ta có một mạng gồm

các máy tính và các kênh điện thoại (gọi tắt là kênh thoại) nối các máy tính này. Chúng ta có thể biểu diễn các vị trí đặt máy tính bởi các điểm và các kênh thoại nối chúng bởi các đoạn nối, xem hình 1.

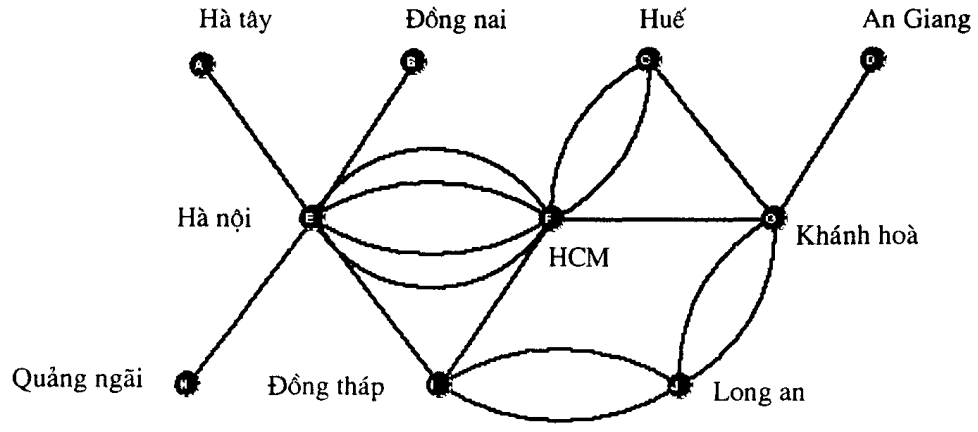


Hình 1. Sơ đồ mạng máy tính

Nhận thấy rằng trong mạng ở hình 1, giữa hai máy bất kỳ chỉ có nhiều nhất là một kênh thoại nối chúng, kênh thoại này cho phép liên lạc cả hai chiều và không có máy tính nào lại được nối với chính nó. Sơ đồ mạng máy tính cho trong hình 1 được gọi là *đơn đồ thị vô hướng*. Ta đi đến định nghĩa sau

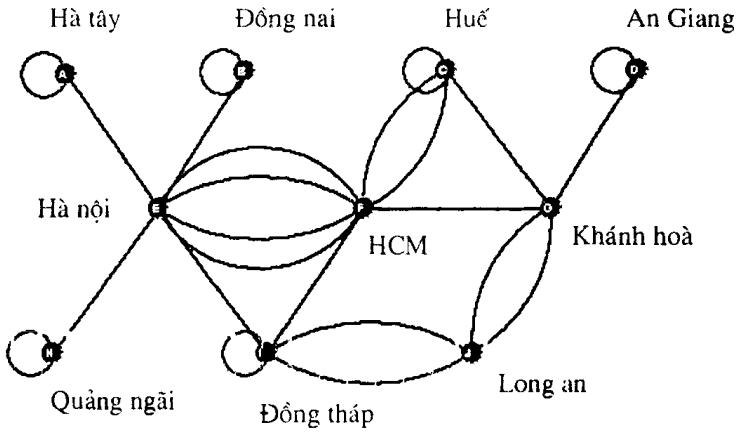
Định nghĩa 1. *Đơn đồ thị vô hướng $G = (V,E)$ bao gồm V là tập các đỉnh, và E là tập các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh.*

Trong trường hợp giữa hai máy tính nào đó thường xuyên phải truyền tải nhiều thông tin người ta phải nối hai máy này bởi nhiều kênh thoại. Mạng với đa kênh thoại giữa các máy được cho trong hình 2.



Hình 2. Sơ đồ mạng máy tính với đa kênh thoại

Định nghĩa 2. *Đa đồ thị vô hướng $G = (V,E)$ bao gồm V là tập các đỉnh, và E là họ các cặp không có thứ tự gồm hai phần tử khác nhau của V gọi là các cạnh. Hai cạnh e_1 và e_2 được gọi là cạnh lặp nếu chúng cùng tương ứng với một cặp đỉnh.*



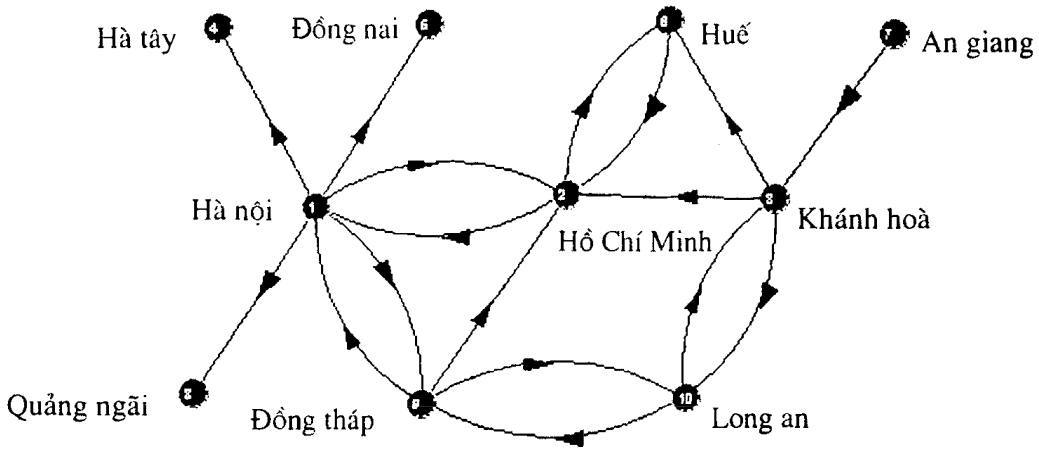
Hình 3. Sơ đồ mạng máy tính với kênh thông báo

Rõ ràng mỗi đơn đồ thị đều là đa đồ thị, nhưng không phải đa đồ thị nào cũng là đơn đồ thị, vì trong đa đồ thị có thể có hai (hoặc nhiều hơn) cạnh nối một cặp đỉnh nào đó.

Trong mạng máy tính có thể có những kênh thoại nối một máy nào đó với chính nó (chẳng hạn với mục đích thông báo). Mạng như vậy được cho trong hình 3. Khi đó đa đồ thị không thể mô tả được mạng như vậy, bởi vì có những *khuyên* (cạnh nối một đỉnh với chính nó). Trong trường hợp này chúng ta cần sử dụng đến khái niệm *giả đồ thị vô hướng*, được định nghĩa như sau

Định nghĩa 3. *Giả đồ thị vô hướng $G = (V,E)$ bao gồm V là tập các đỉnh, và E là họ các cặp không có thứ tự gồm hai phần tử (không nhất thiết phải khác nhau) của V gọi là các cạnh. Cạnh e được gọi là khuyên nếu nó có dạng $e=(u,u)$.*

Các kênh thoại trong mạng máy tính có thể chỉ cho phép truyền tin theo một chiều. Chẳng hạn, trong hình 4 máy chủ ở Hà nội chỉ có thể nhận tin từ các máy ở địa phương, có một số máy chỉ có thể gửi tin đi, còn các kênh thoại cho phép truyền tin theo cả hai chiều được thay thế bởi hai cạnh có hướng ngược chiều nhau.



Hình 4. Mạng máy với các kênh thoại một chiều

Ta đi đến định nghĩa sau.

Định nghĩa 4. Đơn đồ thị có hướng $G = (V, E)$ bao gồm V là tập các đỉnh, và E là tập các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung.

Nếu trong mạng có thể có đa kênh thoại một chiều, ta sẽ phải sử dụng đến khái niệm *đa đồ thị có hướng:*

Định nghĩa 5. Đa đồ thị có hướng $G = (V, E)$ bao gồm V là tập các đỉnh, và E là họ các cặp có thứ tự gồm hai phần tử khác nhau của V gọi là các cung. Hai cung e_1, e_2 tương ứng với cùng một cặp đỉnh được gọi là cung lặp.

Trong các phần tiếp theo chủ yếu chúng ta sẽ làm việc với đơn đồ thị vô hướng và đơn đồ thị có hướng. Vì vậy, để cho ngắn gọn, ta sẽ bỏ qua tính từ **đơn** khi nhắc đến chúng.

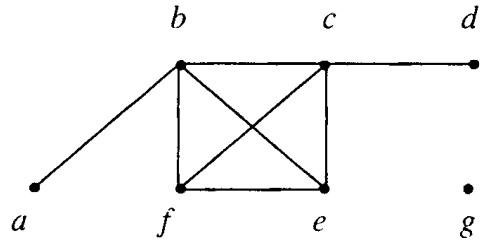
1.2. Các thuật ngữ cơ bản

Trong mục này chúng ta sẽ trình bày một số thuật ngữ cơ bản của lý thuyết đồ thị. Trước tiên, ta xét các thuật ngữ mô tả các đỉnh và cạnh của đồ thị vô hướng.

Định nghĩa 1. *Hai đỉnh u và v của đồ thị vô hướng G được gọi là kề nhau nếu (u,v) là cạnh của đồ thị G. Nếu e=(u,v) là cạnh của đồ thị thì ta nói cạnh này là liên thuộc với hai đỉnh u và v, hoặc cũng nói là cạnh e là nối đỉnh u và đỉnh v, đồng thời các đỉnh u và v sẽ được gọi là các đỉnh đầu của cạnh (u,v).*

Để có thể biết có bao nhiêu cạnh liên thuộc với một đỉnh, ta đưa vào định nghĩa sau

Định nghĩa 2. *Ta gọi bậc của đỉnh v trong đồ thị vô hướng là số cạnh liên thuộc với nó và sẽ ký hiệu là deg(v).*



Hình 1. Đồ thị vô hướng G

Thí dụ 1. Xét đồ thị cho trong hình 1, ta có

$deg(a) = 1$, $deg(b) = 4$, $deg(c) = 4$, $deg(f) = 3$,

$deg(d) = 1$, $deg(e) = 3$, $deg(g) = 0$.

Đỉnh bậc 0 gọi là *đỉnh cô lập*. Đỉnh bậc 1 được gọi là *đỉnh treo*. Trong ví dụ trên đỉnh g là đỉnh cô lập, a và d là các đỉnh treo. Bậc của đỉnh có tính chất sau:

Định lý 1. *Giả sử $G = (V, E)$ là đồ thị vô hướng với m cạnh. Khi đó*

$$2m = \sum_{v \in V} \deg(v)$$

Chứng minh. Rõ ràng mỗi cạnh $e=(u,v)$ được tính một lần trong $deg(u)$ và một lần trong $deg(v)$. Từ đó suy ra tổng tất cả các bậc của các đỉnh bằng hai lần số cạnh.

Thí dụ 2. Đồ thị với n đỉnh và mỗi đỉnh có bậc là 6 có bao nhiêu cạnh?

Giải: Theo định lý 1, ta có $2m = 6n$. Từ đó suy ra số cạnh của đồ thị là $3n$.

Hệ quả. *Trong đồ thị vô hướng, số đỉnh bậc lẻ (nghĩa là có bậc là số lẻ) là một số chẵn.*

Chứng minh. Thực vậy, gọi O và U tương ứng là tập đỉnh bậc lẻ và tập đỉnh bậc chẵn của đồ thị. Ta có

$$2m = \sum_{v \in V} \deg(v) = \sum_{v \in O} \deg(v) + \sum_{v \in U} \deg(v)$$

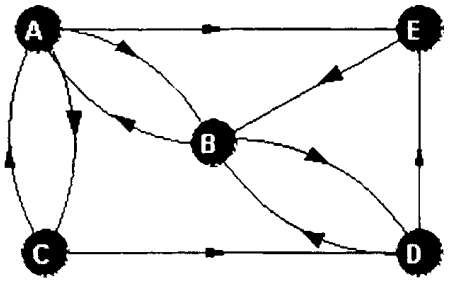
Do $deg(v)$ là chẵn với v là đỉnh trong U nên tổng thứ hai trong vế phải ở trên là số chẵn. Từ đó suy ra tổng thứ nhất (chính là tổng bậc của các đỉnh bậc lẻ) cũng phải là số chẵn, do tất cả các số hạng của nó là số lẻ, nên tổng này phải gồm một số chẵn các số hạng. Vì vậy, số đỉnh bậc lẻ phải là số chẵn.

Ta xét các thuật ngữ tương tự cho đồ thị có hướng.

Định nghĩa 3. *Nếu $e=(u,v)$ là cung của đồ thị có hướng G thì ta nói hai đỉnh u và v là kề nhau, và nói cung (u,v) nối đỉnh u với đỉnh v hoặc cũng nói cung này là đi ra khỏi đỉnh u và đi vào đỉnh v. Đỉnh u (v) sẽ được gọi là đỉnh đầu (cuối) của cung (u,v).*

Tương tự như khái niệm bậc, đối với đồ thị có hướng ta có khái niệm bán bậc ra (vào) của một đỉnh.

Định nghĩa 4. *Ta gọi bán bậc ra (bán bậc vào) của của đỉnh v trong đồ thị có hướng là số cung của đồ thị đi ra khỏi nó (đi vào nó) và ký hiệu là $deg^+(v)$ ($deg^-(v)$)*



Hình 2. Đồ thị có hướng G

Thí dụ 3. Xét đồ thị cho trong hình 2. Ta có

$deg^-(A) = 2, deg^-(B) = 3, deg^-(C) = 1, deg^-(D) = 2, deg^-(E) = 2.$

$deg^+(A) = 3, deg^+(B) = 2, deg^+(C) = 2, deg^+(D) = 2, deg^+(E) = 1.$

Do mỗi cung (u,v) sẽ được tính một lần trong bán bậc vào của đỉnh v và một lần trong bán bậc ra của đỉnh u nên ta có:

Định lý 2. *Giả sử $G=(V,E)$ là đồ thị có hướng. Khi đó*

$$\sum_{v \in V} \deg^+(v) = \sum_{v \in V} \deg^-(v) = |E|$$

Rất nhiều tính chất của đồ thị có hướng không phụ thuộc vào hướng trên các cung của nó. Vì vậy, trong nhiều trường hợp sẽ thuận tiện hơn nếu ta bỏ qua hướng trên các cung của đồ thị. Đồ thị vô hướng thu được bằng cách bỏ qua hướng trên các cung được gọi là *đồ thị vô hướng tương ứng* với đồ thị có hướng đã cho.

1.3. Đường đi, Chu trình. Đồ thị liên thông

Định nghĩa 1. ***Đường đi độ dài n*** *từ đỉnh u đến đỉnh v, trong đó n là số nguyên dương, trên đồ thị vô hướng $G=(V,E)$ là dãy*

$$x_0, x_1, \ldots, x_{n-1}, x_n$$

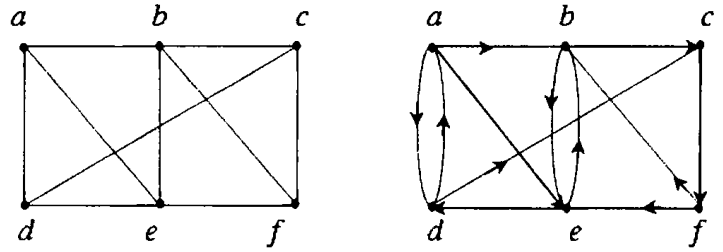
trong đó $u = x_0$, $v = x_n$, $(x_i, x_{i+1}) \in E$, $i = 0, 1, 2, \ldots, n-1$.

Đường đi nói trên còn có thể biểu diễn dưới dạng dãy các cạnh:

$$(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n).$$

Đỉnh u gọi là đỉnh đầu, còn đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là $u = v$) được gọi là ***chu trình****. Đường đi hay chu trình được gọi là* ***đơn*** *nếu như không có cạnh nào bị lặp lại.*

Thí dụ 1. Xét đồ thị vô hướng cho trong hình 1:



Hình 1. Đường đi trên đồ thị

Ta có: *a, d, c, f, e* là đường đi đơn độ dài 4. Còn *d, e, c, a* không là dường đi, do (*e, c*) không phải là cạnh của đồ thị. Dãy *b, c, f, e, b* là chu trình độ dài 4. Đường đi *a, b, e, d, a, b* có độ dài là 5 không phải là đường đi đơn, do cạnh (*a, b*) có mặt trong nó hai lần.

Khái niệm đường đi và chu trình trên đồ thị có hướng được định nghĩa hoàn toàn tương tự như trường hợp đồ thị vô hướng, chỉ khác là ta có chú ý đến hướng trên các cung.

Định nghĩa 2. ***Đường đi độ dài n*** *từ đỉnh u đến đỉnh v, trong đó n là số nguyên dương, trên đồ thị có hướng G=(V,A) là dãy*

$$x_0, x_1, \ldots, x_{n-1}, x_n$$

trong đó $u = x_0$, $v = x_n$, $(x_i, x_{i+1}) \in A$, $i = 0, 1, 2, \ldots, n-1$.

Đường đi nói trên còn có thể biểu diễn dưới dạng dãy các cung:

$$(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n).$$

Đỉnh u gọi là đỉnh đầu, còn đỉnh v gọi là đỉnh cuối của đường đi. Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là u = v) được gọi là ***chu trình****. Đường đi hay chu trình được gọi là* ***đơn*** *nếu như không có cung nào bị lặp lại.*

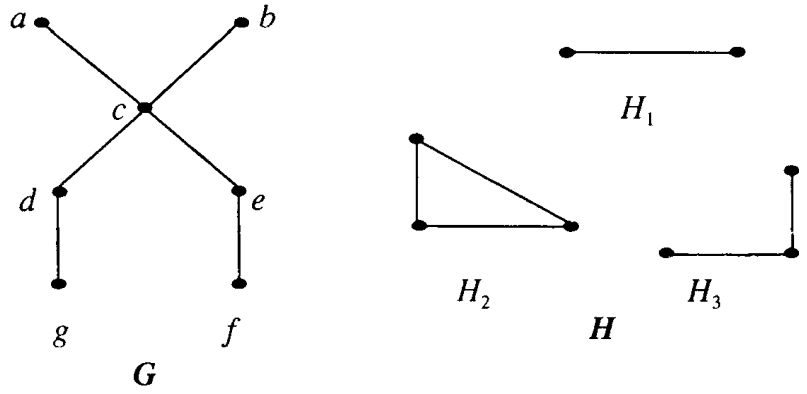
Thí dụ 2. Trên đồ thị có hướng cho trong hình 1: *a, d, c, f, e* là đường đi đơn độ dài 4. Còn *d, e, c, a* không là đường đi, do *(e,c)* không phải là cung của đồ thị. Dãy *b, c, f, e, b* là chu trình độ dài 4. Đường đi *a, b, e, d, a, b* có độ dài là 5 không phải là đường đi đơn, do cung *(a,b)* có mặt trong nó hai lần.

Xét một mạng máy tính. Một câu hỏi đặt ra là hai máy tính bất kỳ trong mạng này có thể trao đổi thông tin được với nhau hoặc là trực tiếp qua kênh nối chúng hoặc thông qua một hoặc vài máy tính trung gian trong mạng? Nếu sử dụng đồ thị để biểu diễn mạng máy tính này (trong đó các đỉnh của đồ thị tương ứng với các máy tính, còn các cạnh tương ứng với các kênh nối) câu hỏi đó được phát biểu trong ngôn ngữ đồ thị như sau: Tồn tại hay chăng đường đi giữa mọi cặp đỉnh của đồ thị?

Định nghĩa 3. *Đồ thị vô hướng G= (V,E) được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.*

Như vậy hai máy tính bất kỳ trong mạng có thể trao đổi thông tin được với nhau khi và chỉ khi đồ thị tương ứng với mạng này là đồ thị liên thông.

Thí dụ 3. Trong hình 2: Đồ thị *G* là liên thông, còn đồ thị *H* là không liên thông.



Hình 2. Đồ thị liên thông G và đồ thị H gồm 3 thành phần liên thông H_1, H_2, H_3.

Định nghĩa 4. *Ta gọi đồ thị con của đồ thị $G = (V,E)$ là đồ thị $H = (W, F)$, trong đó $W \subseteq V$ và $F \subseteq E$.*

Trong trường hợp đồ thị là không liên thông, nó sẽ rã ra thành một số đồ thị con liên thông đôi một không có đỉnh chung. Những đồ thị con liên thông như vậy ta sẽ gọi là các ***thành phần liên thông*** của đồ thị.

Thí dụ 4. Đồ thị H trong hình 2 gồm 3 thành phần liên thông H_1, H_2, H_3.

Trong mạng máy tính có thể có những máy (những kênh nối) mà sự hỏng hóc của nó sẽ ảnh hưởng đến việc trao đổi thông tin trong mạng. Các khái niệm tương ứng với tình huống này được đưa ra trong định nghĩa sau.

Định nghĩa 5. *Đỉnh v được gọi là **đỉnh rẽ nhánh** nếu việc loại bỏ v cùng với các cạnh liên thuộc với nó khỏi đồ thị làm tăng số thành phần liên thông của đồ thị. Cạnh e được gọi là **cầu** nếu việc loại bỏ nó khỏi đồ thị làm tăng số thành phần liên thông của đồ thị.*

Thí dụ 5. Trong đồ thị G ở hình 2, đỉnh d và e là đỉnh rẽ nhánh, còn các cạnh (d,g) và (e,f) là cầu.

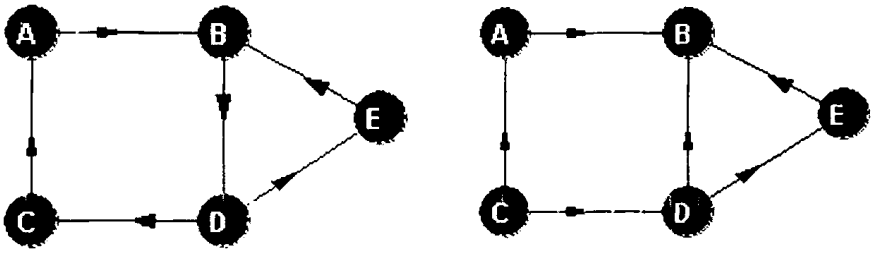
Đối với đồ thị có hướng có hai khái niệm liên thông phụ thuộc vào việc ta có xét đến hướng trên các cung hay không.

Định nghĩa 6. *Đồ thị có hướng $G= (V,A)$ được gọi là liên thông mạnh nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.*

Định nghĩa 7. *Đồ thị có hướng $G= (V,A)$ được gọi là liên thông yếu nếu đồ thị vô hướng tương ứng với nó là đồ thị vô hướng liên thông.*

Rõ ràng nếu đồ thị là liên thông mạnh thì nó cũng là liên thông yếu, nhưng điều ngược lại là không luôn đúng, như chỉ ra trong thí dụ dưới đây.

Thí dụ 6. Trong hình 3 đồ thị G là liên thông mạnh, còn H là liên thông yếu nhưng không là liên thông mạnh.



G H

Hình 3. Đồ thị liên thông mạnh G và đồ thị liên thông yếu H

Một câu hỏi đặt ra là khi nào có thể định hướng các cạnh của một đồ thị vô hướng liên thông để có thể thu được đồ thị có hướng liên thông mạnh? Ta sẽ gọi đồ thị như vậy là đồ thị định hướng được. Định lý dưới đây cho ta tiêu chuẩn nhận biết một đồ thị có là định hướng được hay không.

Định lý 1. *Đồ thị vô hướng liên thông là định hướng được khi và chỉ khi mỗi cạnh của nó nằm trên ít nhất một chu trình.*

Chứng minh. ***Điều kiện cần***. Giả sử (u,v) là một cạnh của đồ thị. Từ sự tồn tại đường đi có hướng từ u đến v và ngược lại suy ra (u,v) phải nằm trên ít nhất một chu trình.

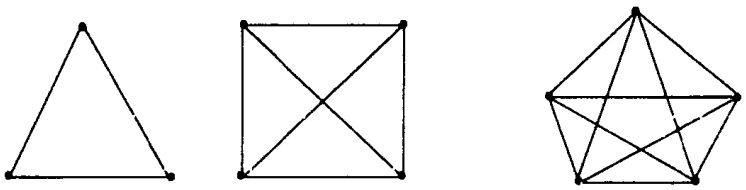
Điều kiện đủ. Thủ tục sau đây cho phép định hướng các cạnh của đồ thị để thu được đồ thị có hướng liên thông mạnh. Giả sử C là một chu trình nào đó trong đồ thị. Định hướng các cạnh trên chu trình này theo một hướng đi vòng theo nó. Nếu tất cả các cạnh của đồ thị là đã được định hướng thì kết thúc thủ tục. Ngược lại, chọn e là một cạnh chưa định hướng có chung đỉnh với ít nhất một trong số các cạnh đã định hướng. Theo giả thiết tìm được chu trình C' chứa cạnh e. Định hướng các cạnh chưa được định hướng của C' theo một hướng dọc theo chu trình này (không định hướng lại các cạnh đã có hướng). Thủ tục trên sẽ được lặp lại cho đến khi tất cả các cạnh của đồ thị được định hướng. Khi đó ta thu được đồ thị có hướng liên thông mạnh.

1.4. Một số dạng đồ thị đặc biệt

Trong mục này ta xét một số dạng đơn đồ thị vô hướng đặc biệt xuất hiện trong nhiều vấn đề ứng dụng thực tế.

Đồ thị đầy đủ. Đồ thị đầy đủ n đỉnh, ký hiệu bởi K_n, là đơn đồ thị vô hướng mà giữa hai đỉnh bất kỳ của nó luôn có cạnh nối.

Các đồ thị K_3, K_4, K_5 cho trong hình 1 dưới đây.



K_3 K_4 K_5

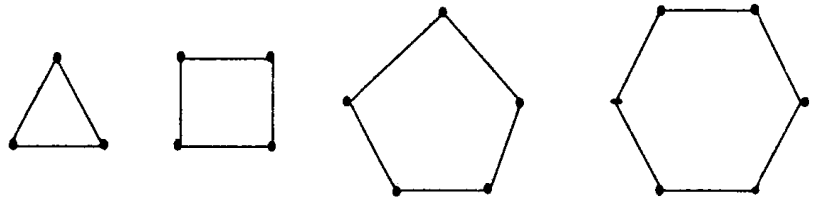
Hình 1. Đồ thị đầy đủ

Đồ thị đầy đủ K_n có tất cả $n(n\text{-}1)/2$ cạnh, nó là đơn đồ thị có nhiều cạnh nhất.

Đồ thị vòng. Đồ thị vòng C_n, $n \geq 3$, gồm n đỉnh $v_1, v_2, \ldots, v_n$ và các cạnh

$$(v_1, v_2),\ (v_2, v_3),\ \ldots,\ (v_{n-1}, v_n),\ (v_n, v_1).$$

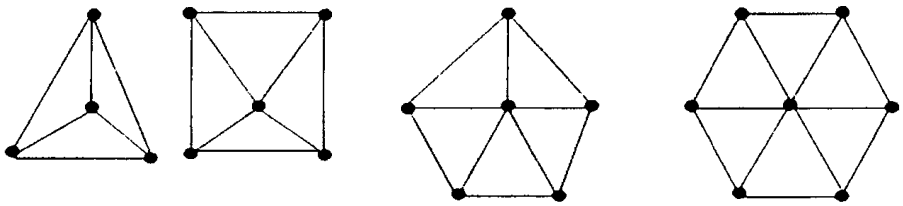
Đồ thị vòng C_3, C_4, C_5, C_6 cho trong hình 2.



C_3 C_4 C_5 C_6

Hình 2. Đồ thị vòng C_3, C_4, C_5, C_6

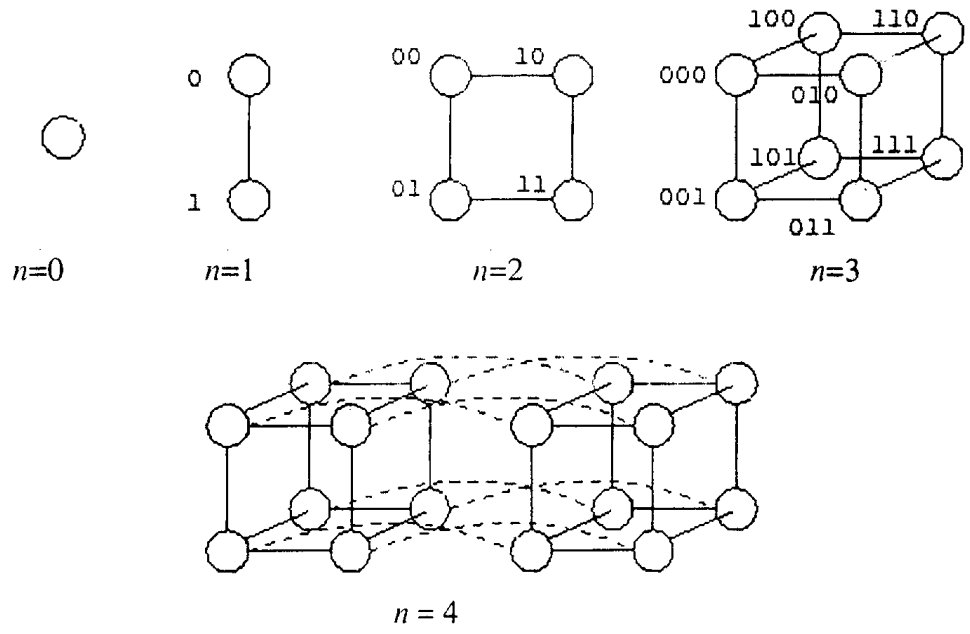
Đồ thị bánh xe. Đồ thị W_n thu được từ C_n bằng cách bổ sung vào một đỉnh mới nối với tất cả các đỉnh của C_n (xem hình 3).



W_3 W_4 W_5 W_6

Hình 3. Đồ thị bánh xe W_3, W_4, W_5, W_6

Đồ thị lập phương. Đồ thị lập phương n đỉnh Q_n là đồ thị với các đỉnh biểu diễn 2^n xâu nhị phân độ dài n. Hai đỉnh của nó là kề nhau nếu như hai xâu nhị phân tương ứng chỉ khác nhau 1 bit. Hình 4 cho thấy Q_n, với $n = 0,1, 2, 3, 4$.



Hình 4. Đồ thị lập phương Q_n

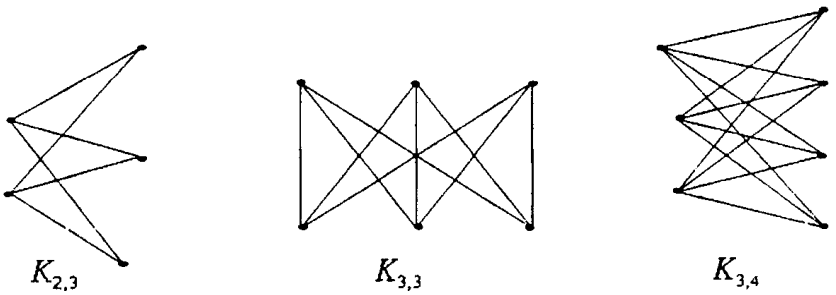
Đồ thị hai phía. Đơn đồ thị $G = (V,E)$ được gọi là *hai phía* nếu như tập đỉnh V của nó có thể phân hoạch thành hai tập X và Y sao cho mỗi cạnh của đồ thị chỉ nối một đỉnh nào đó trong X với một đỉnh nào đó trong Y. Khi đó ta sẽ sử dụng ký hiệu $G=(X \cup Y, E)$ để chỉ đồ thị hai phía với tập đỉnh $X \cup Y$.

Định lý sau đây cho phép nhận biết một đơn đồ thị có phải là hai phía hay không.

Định lý 1. *Đơn đồ thị là đồ thị hai phía khi và chỉ khi nó không chứa chu trình độ dài lẻ.*

Để kiểm tra xem một đồ thị liên thông có phải là hai phía hay không có thể áp dụng thủ tục sau. Chọn v là một đỉnh bất kỳ của đồ thị. Đặt $X = \{v\}$, còn Y là tập các đỉnh kề của v. Khi đó các đỉnh kề của các đỉnh trong Y phải thuộc vào X. Ký hiệu tập các đỉnh như vậy là T. Vì thế nếu phát hiện $T \cap Y \neq \varnothing$ thì đồ thị không phải là hai phía, kết thúc. Ngược lại, đặt $X = X \cup T$. Tiếp tục xét như vậy đối với T' là tập các đỉnh kề của T,...

Đồ thị hai phía $G = (X \cup Y, E)$ với $|X| = m, |Y| = n$ được gọi là ***đồ thị hai phía đầy đủ*** và ký hiệu là $K_{m,n}$ nếu mỗi đỉnh trong tập X được nối với mỗi đỉnh trong Y. Các đồ thị $K_{2,3}$, $K_{3,3}$, $K_{3,4}$ được cho trong hình 5.



Hình 5. Đồ thị hai phía

Đồ thị phẳng. Đồ thị được gọi là *đồ thị phẳng* nếu ta có thể vẽ nó trên mặt phẳng sao cho các cạnh của nó không cắt nhau ngoài ở đỉnh. Cách vẽ như vậy sẽ được gọi là *biểu diễn phẳng* của đồ thị.

Thí dụ đồ thị K_4 là phẳng, vì có thể vẽ nó trên mặt phẳng sao cho các cạnh của nó không cắt nhau ngoài ở đỉnh (xem hình 6).

Hình 6. Đồ thị K_4 là đồ thị phẳng

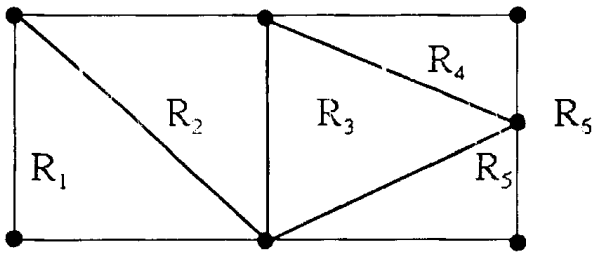
Một điểm đáng lưu ý là nếu đồ thị là phẳng thì luôn có thể vẽ nó trên mặt phẳng với các cạnh nối là các đoạn thẳng không cắt nhau ngoài ở đỉnh (ví dụ xem cách vẽ K_4 trong hình 3).

Để nhận biết xem một đồ thị có phải là đồ thị phẳng có thể sử dụng định lý Kuratovski, mà để phát biểu nó ta cần một số khái niệm sau: Ta gọi một ***phép chia cạnh*** (u,v) của đồ thị là việc loại bỏ cạnh này khỏi đồ thị và thêm vào đồ thị một đỉnh mới w cùng với hai cạnh (u,w), (w,u). Hai đồ thị $G = (V, E)$ và $H = (W, F)$ được gọi là ***đồng cấu*** nếu chúng có thể thu được từ cùng một đồ thị nào đó nhờ các phép chia cạnh.

Định lý 2 (Kuratovski). *Đồ thị là phẳng khi và chỉ khi nó không chứa đồ thị con đồng cấu với $K_{3,3}$ hoặc K_5.*

Trong trường hợp riêng, đồ thị $K_{3,3}$ và K_5 không phải là đồ thị phẳng. Bài toán về tính phẳng của đồ thị $K_{3,3}$ là bài toán đố nổi tiếng về *ba căn hộ và ba hệ thống cung cấp năng lượng* cho chúng: Cần xây dựng hệ thống đường cung cấp điện, hơi đốt và nước cho ba căn hộ, nối mỗi một trong ba nguồn cung cấp năng lượng với mỗi một căn hộ nói trên sao cho chúng không cắt nhau.

Đồ thị phẳng còn tìm được những ứng dụng quan trọng trong công nghệ chế tạo mạch in. Biểu diễn phẳng của đồ thị sẽ chia mặt phẳng ra thành các miền, trong đó có thể có cả miền không bị chặn. Thí dụ, biểu diễn phẳng của đồ thị cho trong hình 7 chia mặt phẳng ra thành 6 miền $R_1, R_2, \ldots, R_6$.



Hình 7. Các miền tương ứng với biểu diễn phẳng của đồ thị

Euler đã chứng minh được rằng các cách biểu diễn phẳng khác nhau của một đồ thị đều chia mặt phẳng ra thành cùng một số miền. Để chứng minh điều đó, Euler đã tìm được mối liên hệ giữa số miền, số đỉnh của đồ thị và số cạnh của đồ thị phẳng sau đây.

Định lý 3 (Công thức Euler). Giả sử G là đồ thị phẳng liên thông với n đỉnh, m cạnh. Gọi r là số miền của mặt phẳng bị chia bởi biểu diễn phẳng của G. Khi đó

$$r = m - n + 2.$$

Có thể chứng minh định lý bằng qui nạp. Xét thí dụ minh hoạ cho áp dụng công thức Euler.

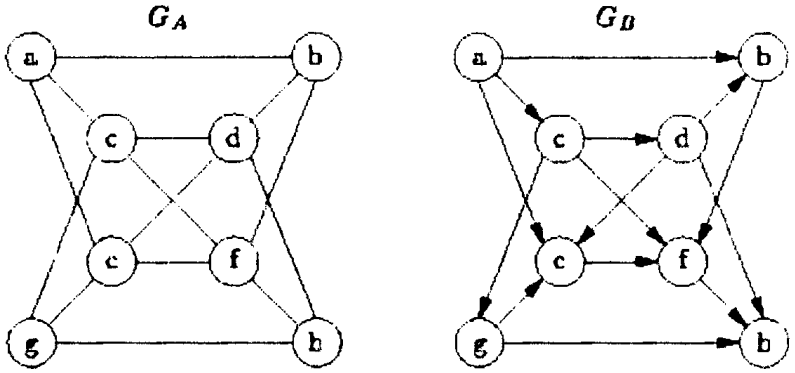
Thí dụ. Cho G là đồ thị phẳng liên thông với 20 đỉnh, mỗi đỉnh đều có bậc là 3. Hỏi mặt phẳng bị chia làm bao nhiêu phần bởi biểu diễn phẳng của đồ thị G?

Giải. Do mỗi đỉnh của đồ thị đều có bậc là 3, nên tổng bậc của các đỉnh là $3\times 20 = 60$. Từ đó suy ra số cạnh của đồ thị $m = 60/2 = 30$. Vì vậy, theo công thức Euler, số miền cần tìm là

$$r = 30 - 20 + 2 = 12.$$

Bài tập

1. Xác định bậc của các đỉnh trong đồ thị G_A. Xác định bán bậc ra và bán bậc vào của các đỉnh của đồ thị G_B.



2. Vẽ đồ thị vô hướng $G = (V,E)$ cho bởi:

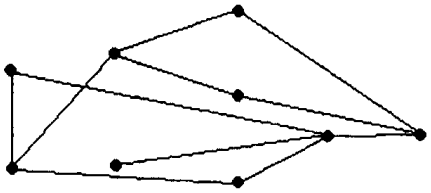
$$V = \{A, B, C, D, E, F\}$$

và

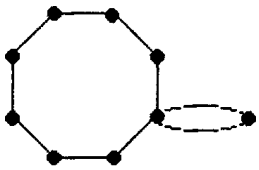
$$E = \{(E,G), (B,F), (D,C), (D,F), (F,B), (C,F), (A,F), (E,D)\}$$

3. Với mỗi đồ thị trong các đồ thị sau đây hãy cho biết nó có là đồ thị hai phía hay không? Nếu câu trả lời là khẳng định, hãy chỉ rõ cách phân hoạch tập đỉnh thành hai tập đỉnh sao cho cạnh nối chỉ có giữa hai đỉnh thuộc hai tập khác nhau.

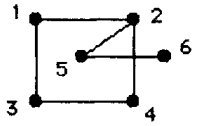
a.



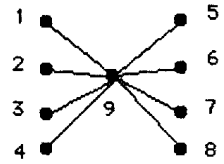
b.



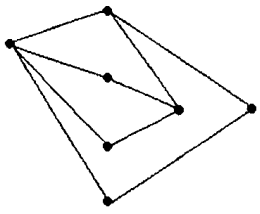
d.



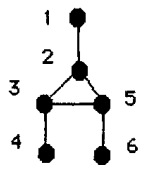
e.



f.



g.



3. Cho đơn đồ thị vô hướng liên thông $G = (V, E)$ với n đỉnh.

a) Chứng minh rằng luôn tồn tại đường đi đơn nối hai đỉnh u, v bất kỳ của đồ thị.

(Gợi ý: Đường đi cần tìm là đường đi ngắn nhất theo số cạnh)

b) Chứng minh rằng luôn tồn tại đường đi qua không quá n đỉnh nối hai đỉnh u, v bất kỳ của đồ thị.

(Gợi ý: Đường đi cần tìm là đường đi ngắn nhất theo số cạnh).

4. Cho G là đơn đồ thị vô hướng với n đỉnh, m cạnh, k thành phần liên thông. Chứng minh rằng:

$$n-k \leq (n-k)(n-k+1)/2.$$

Từ đó suy ra đồ thị n đỉnh với số cạnh lớn hơn $(n-1)(n-2)/2$ là liên thông.

(Gợi ý: Chứng minh bằng qui nạp theo số cạnh của đồ thị).

5. Chứng minh rằng trong đơn đồ thị với $n > 1$ đỉnh luôn tìm được hai đỉnh không là đỉnh rẽ nhánh.

6. Chứng minh rằng đỉnh u trong đơn đồ thị liên thông G là đỉnh rẽ nhánh khi và chỉ khi tìm được hai đỉnh v và w $(v,w \neq u)$ sao cho mọi đường đi nối v và w đều đi qua đỉnh u.

7. Chứng minh rằng cạnh e trong đơn đồ thị G là cầu khi và chỉ khi nó không thuộc bất cứ chu trình nào trong G.

8. Cho G là đồ thị hai phía với n đỉnh và m cạnh. Chứng minh rằng $m \leq n^2/4$.

9. Cho G là đồ thị hai phía với n đỉnh và m cạnh. Gọi K và k là bậc lớn nhất và nhỏ nhất của các đỉnh của G. Chứng minh rằng

$$m \leq 2m/n \leq M.$$

10. Đồ thị vô hướng được gọi là **chính qui bậc k** nếu tất cả các đỉnh của nó đều có bậc là k. Với giá trị nào của n đồ thị sau là chính qui?

a) K_n

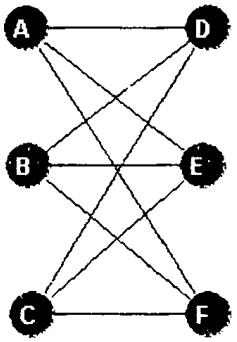
b) C_n

c) W_n

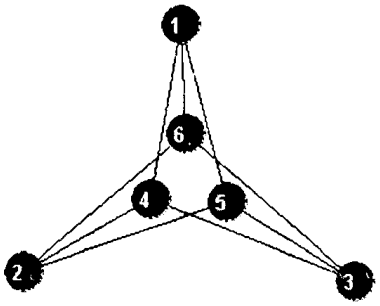
d) Q_n

11. Ta gọi đồ thị G' là **đồ thị bù** của đơn đồ thị G nếu các đỉnh của nó là đỉnh của đồ thị G và hai đỉnh của G' là kề nhau khi và chỉ khi chúng là không kề nhau trên G. Hãy vẽ các đồ thị bù của K_n, $K_{m,n}$, C_n, Q_n.

12. Hai đơn đồ thị vô hướng $G_1 = (V_1, E_1)$ và $G_2 = (V_2, E_2)$ được gọi là ***đẳng cấu*** nếu tồn tại một song ánh f: $V_1 \to V_2$ sao cho $(u,v) \in E_1$ khi và chỉ khi $(f(u), f(v)) \in E_2$. Thí dụ, hai đồ thị G_1 và G_2 cho trong hình dưới đây là đẳng cấu



G_1

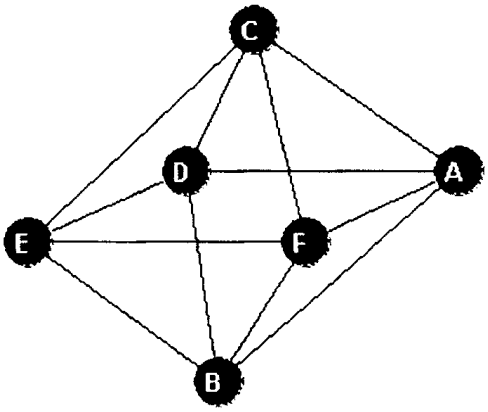


G_2

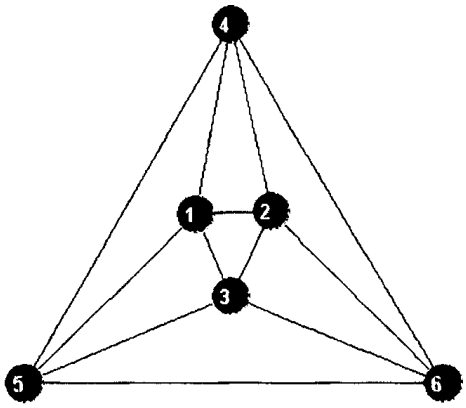
Song ánh f được xác định như sau: f(A)=1, f(B)=2, f(C)=3; f(D)=4, f(E)=5, f(F)=6.

Hỏi hai đồ thị sau đây có đẳng cấu hay không?

a)

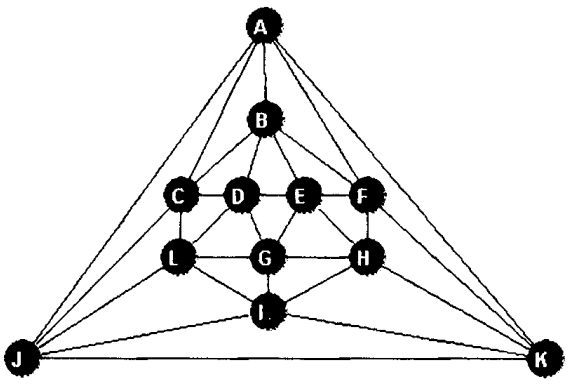


G_1

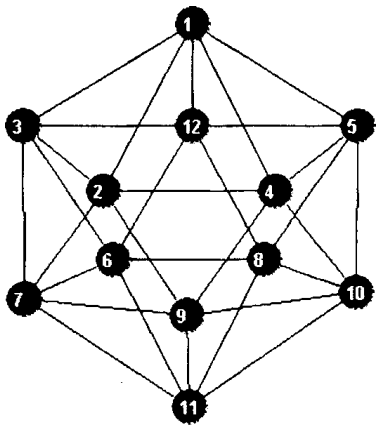


G_2

b)



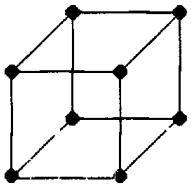
H_1



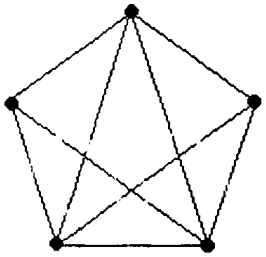
H_2

13. Với mỗi đồ thị trong các đồ thị sau đây hãy cho biết nó có là đồ thị phẳng hay không. Nếu câu trả lời là khẳng định hãy trình bày cách vẽ đồ thị sao cho các cạnh không cắt nhau ngoài ở đỉnh:

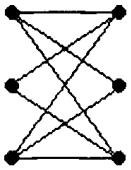
a)



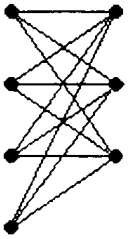
b)



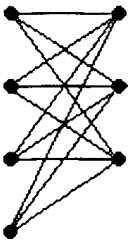
c)



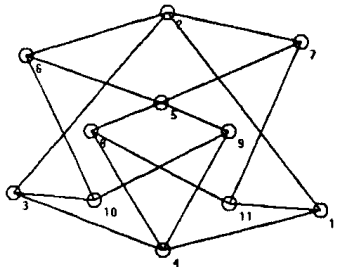
d)



e)



f)



14. Hỏi rằng đồ thị Q_3 có phải là phẳng không? Trong trường hợp câu trả lời là khẳng định hãy vẽ nó trên mặt phẳng sao cho không có cạnh nào cắt nhau.

15. Cho G là đơn đồ thị phẳng liên thông với 20 đỉnh và mỗi đỉnh của nó đều có bậc là 3. Hỏi rằng khi vẽ G trên mặt phẳng thì mặt phẳng bị chia làm bao nhiêu phần ?

16. Bài toán tô màu đồ thị: Cho đơn đồ thị vô hướng $G = (V,E)$. Hãy tìm cách gán cho mỗi đỉnh của đồ thị một màu sao cho hai đỉnh kề nhau không bị tô bởi cùng một màu. Một phép gán màu cho các đỉnh như vậy được gọi là một phép tô màu đồ thị. Bài toán tô màu đòi hỏi tìm phép tô màu với số màu phải sử dụng là ít nhất.

Ta gọi sắc số của đồ thi G, ký hiệu là $\chi(G)$, là số màu ít nhất cần dùng để tô màu đồ thị. Dưới đây là một số kết quả liên quan đến tô màu đồ thị

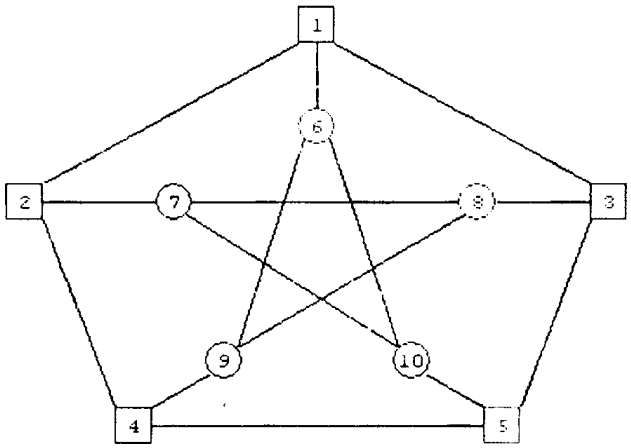
a) Định lý 4 màu: "Mọi đồ thị phẳng đều có thể tô bởi 4 mầu".

b) Đơn đồ thị G là hai phía khi và chỉ khi $\chi(G) = 2$.

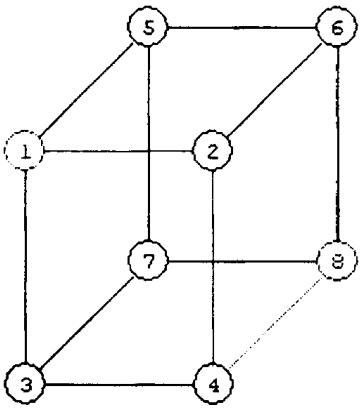
Hãy chứng minh mệnh đề b).

17. Hãy tính sắc số của các đồ thị cho trong các hình vẽ sau

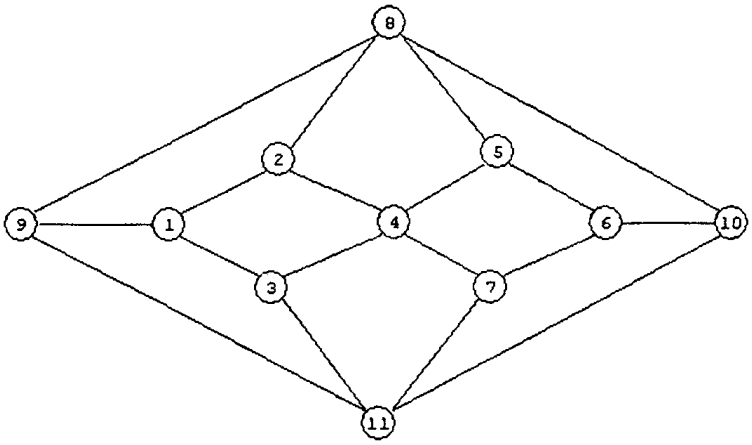
a) Đồ thị Petersen



b) Đồ thị lập phương



c) Đồ thị Herschel



2

BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH

Để lưu trữ đồ thị và thực hiện các thuật toán khác nhau với đồ thị trên máy tính cần phải tìm những cấu trúc dữ liệu thích hợp để mô tả đồ thị. Việc chọn cấu trúc dữ liệu nào để biểu diễn đồ thị có tác động rất lớn đến hiệu quả của thuật toán. Vì vậy, việc chọn lựa cấu trúc dữ liệu để biểu diễn đồ thị phụ thuộc vào từng tình huống cụ thể (bài toán và thuật toán cụ thể). Trong mục này chúng ta sẽ xét một số phương pháp cơ bản được sử dụng để biểu diễn đồ thị trên máy tính, đồng thời cũng phân tích một cách ngắn gọn những ưu điểm cũng như những nhược điểm của chúng.

2.1. Ma trận kề. Ma trận trọng số

Xét đơn đồ thị vô hướng $G = (V, E)$, với tập đỉnh $V = \{1, 2,..., n\}$, tập cạnh $E=\{e_1, e_2,..., e_m\}$. Ta gọi ma trận kề của đồ thị G là (0,1)-ma trận

$$A = \{a_{ij} : i, j = 1, 2,..., n\}$$

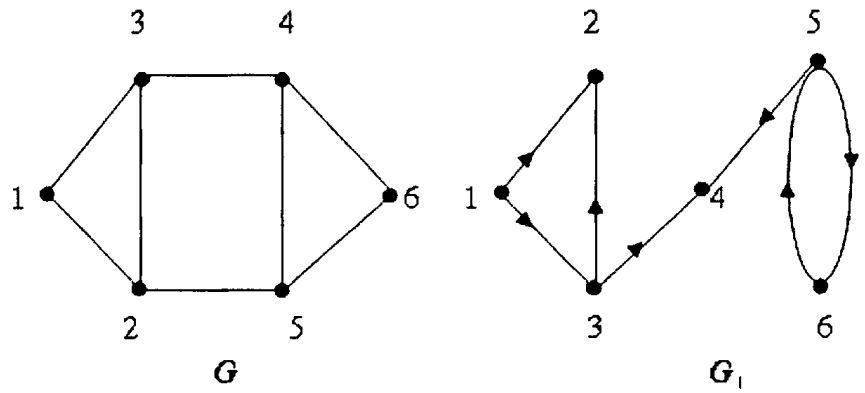
với các phần tử được xác định theo quy tắc sau đây:

$$a_{ij} = 0, \text{ nếu } (i,j) \notin E \quad \text{và} \quad a_{ij} = 1, \text{ nếu } (i,j) \in E,$$

$i, j = 1,2,..., n.$

Thí dụ 1. Ma trận kề của đồ thị vô hướng G cho trong hình 1 là

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



Hình 1. Đồ thị vô hướng G và Đồ thị có hướng G_1

Các tính chất của ma trận kề:

1) Rõ ràng ma trận kề của đồ thị vô hướng là ma trận đối xứng, tức là

$$a[i, j] = a[j, i], \quad i, j = 1, 2,..., n.$$

Ngược lại, mỗi (0,1)-ma trận đối xứng cấp n sẽ tương ứng, chính xác đến cách đánh số đỉnh (còn nói là: chính xác đến đẳng cấu) , với một đơn đồ thị vô hướng n đỉnh.

2) Tổng các phần tử trên dòng i (cột j) của ma trận kề chính bằng bậc của đỉnh i (đỉnh j).

3) Nếu ký hiệu

$$a_{ij}^p \ , \ i, j = 1, \ 2,..., \ n$$

là các phần tử của ma trận tích

$$A^p = \underbrace{A.A\ldots A}_{p}$$

Khi đó

$$a_{ij}^p\ ,\ i, j = 1,\ 2,\ldots,\ n$$

cho ta số đường đi khác nhau từ đỉnh i đến đỉnh j qua p-1 đỉnh trung gian.

Ma trận kề của đồ thị có hướng được định nghĩa một cách hoàn toàn tương tự.

Thí dụ 2. Đồ thị có hướng G_1 cho trong hình 1 có ma trận kề là ma trận sau

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Lưu ý rằng ma trận kề của đồ thị có hướng không phải là ma trận đối xứng.

Chú ý: Trên đây chúng ta chỉ xét đơn đồ thị. Ma trận kề của đa đồ thị có thể xây dựng hoàn toàn tương tự, chỉ khác là thay vì ghi 1 vào vị trí $a[i,j]$ nếu (i, j) là cạnh của đồ thị, chúng ta sẽ ghi k là số cạnh nối hai đỉnh i và j.

Trong rất nhiều vấn đề ứng dụng của lý thuyết đồ thị, mỗi cạnh $e = (u, v)$ của đồ thị được gán với một con số $c(e)$ (còn viết là $c(u,v)$) gọi là trọng số của cạnh e. Đồ thị trong trường hợp như vậy được gọi là đồ thị có trọng số. Trong trường hợp đồ thị có trọng số, thay vì ma trận kề, để biểu diễn đồ thị ta sử dụng ma trận trọng số

$$C = c[i, j],\ i, j = 1, 2,\ldots, n,$$

với

$$c[i, j] = c(i, j), \qquad \text{nếu } (i, j) \in E$$

và

$$c[i, j] = \theta, \qquad \text{nếu } (i, j) \notin E,$$

trong đó số θ , tuỳ từng trường hợp cụ thể, có thể được đặt bằng một trong các giá trị sau: 0, $+\infty$, $-\infty$.

Ưu điểm lớn nhất của phương pháp biểu diễn đồ thị bằng ma trận kề (hoặc ma trận trọng số) là để trả lời câu hỏi: Hai đỉnh u, v có kề nhau trên đồ thị hay không, chúng ta chỉ phải thực hiện một phép so sánh. Nhược điểm lớn nhất của phương pháp này là:

không phụ thuộc vào số cạnh của đồ thị, ta luôn phải sử dụng n^2 đơn vị bộ nhớ để lưu trữ ma trận kề của nó.

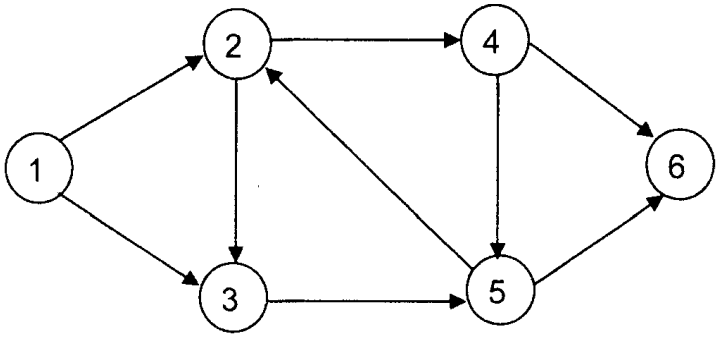
2.2. Ma trận liên thuộc đỉnh-cạnh

Xét $G = (V, E)$, ($V = \{1, 2, ..., n\}$, $E = \{e_1, e_2, ..., e_m\}$), là đơn đồ thị có hướng. Xây dựng ma trận $A = (a_{ij}: i = 1, 2, ..., n; j = 1, 2, ..., m)$, trong đó

$$a_{ij} = \begin{cases} 1, & \text{nếu đỉnh } i \text{ là đỉnh đầu của cung } e_j \\ -1, & \text{nếu đỉnh } i \text{ là đỉnh cuối của cung } e_j \\ 0, & \text{nếu đỉnh } i \text{ không là đầu mút của cung } e_j \end{cases}$$

Ma trận A xây dựng theo qui tắc vừa nêu được gọi là ma trận liên thuộc đỉnh-cạnh.

Ví dụ. Xét đồ thị cho trên hình 3



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccccc} (1,2) & (1,3) & (2,3) & (2,4) & (3,5) & (4,5) & (4,6) & (5,2) & (5,6) \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \end{bmatrix} \end{array}$$

Hình 2. Đồ thị có hướng và ma trận liên thuộc đỉnh cạnh

Ma trận liên thuộc đỉnh-cạnh là một trong những cách biểu diễn rất hay được sử dụng trong các bài toán liên quan đến đồ thị có hướng mà trong đó phải xử lý các cung của đồ thị.

2.3. Danh sách cạnh (cung)

Trong trường hợp đồ thị thưa (đồ thị có số cạnh m thoả mãn bất đẳng thức: $m < 6n$) người ta thường dùng cách biểu diễn đồ thị dưới dạng *danh sách cạnh*.

Trong cách biểu diễn đồ thị bởi danh sách cạnh (cung) chúng ta sẽ lưu trữ danh sách tất cả các cạnh (cung) của đồ thị vô hướng (có hướng). Mỗi cạnh (cung) $e = (x, y)$ của đồ thị sẽ tương ứng với hai biến Dau[e], Cuoi[e]. Như vậy, để lưu trữ đồ thị ta cần sử dụng $2m$ đơn vị bộ nhớ. Nhược điểm của cách biểu diễn này là để xác định những đỉnh nào của đồ thị là kề với một đỉnh cho trước chúng ta phải làm cỡ m phép so sánh (khi duyệt qua danh sách tất cả các cạnh của đồ thị).

Chú ý: Trong trường hợp đồ thị có trọng số ta cần thêm m đơn vị bộ nhớ để lưu trữ trọng số của các cạnh.

Thí dụ 3. Danh sách cạnh (cung) của đồ thị G (G_1) cho trong hình 1 là:

Dau	Cuoi
1	2
1	3
1	5
2	3
2	5
3	4
4	5
4	6
5	6

Danh sách cạnh của G

Dau	Cuoi
1	2
1	3
3	2
3	4
5	4
5	6
6	5

Danh sách cung của G_1

2.4. Danh sách kề

Trong rất nhiều vấn đề ứng dụng của lý thuyết đồ thị, cách biểu diễn đồ thị dưới dạng *danh sách kề* là cách biểu diễn thích hợp nhất được sử dụng.

Trong cách biểu diễn này, với mỗi đỉnh v của đồ thị chúng ta lưu trữ danh sách các đỉnh kề với nó, mà ta sẽ ký hiệu là $Ke(v)$, tức là

$$Ke(v) = \{ u \in V: (v,u) \in E \}.$$

Khi đó vòng lặp thực hiện với mỗi một phần tử trong danh sách này theo thứ tự các phần tử được xắp xếp trong nó sẽ được viết như sau:

for $u \in Ke(v)$ **do...**

Chẳng hạn, trên PASCAL có thể mô tả danh sách này như sau (Gọi là cấu trúc **Forward Star**):

```
Const
    m = 1000; { m - số cạnh}
    n =  100;  { n -số đỉnh}
var
    Ke : array[1..m] of  integer;
    Tro: array[1..n+1] of  integer;
```

trong đó Tro[i] ghi nhận vị trí bắt đầu của danh sách kề của đỉnh i, i=1,2,...,n, Tro[n+1] = 2m+1.

Khi đó dòng lệnh qui ước

```
for u ∈ Ke(v) do
begin
    .......
end;
```

có thể thay thế bởi cấu trúc lệnh cụ thể trên PASCAL sau

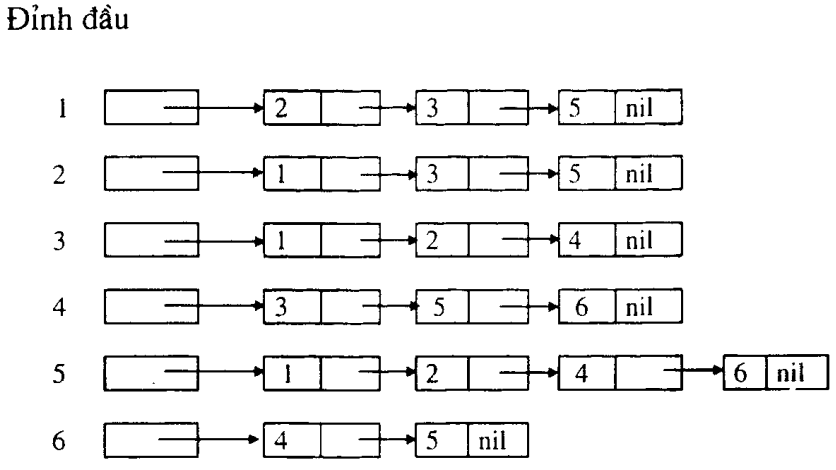
```
for i:= Tro[v] to Tro[v+1]-1 do
begin
   u:=Ke[i];
   . . . . . .
end;
```

Trong rất nhiều thuật toán làm việc với đồ thị chúng ta thường xuyên phải thực hiện các thao tác: Thêm hoặc bớt một số cạnh. Trong trường hợp này cấu trúc dữ liệu dùng ở trên là không thuận tiện. Khi đó nên chuyển sang sử dụng danh sách kề liên kết (***Linked Adjancency List***) như mô tả trong chương trình nhập danh sách kề của đồ thị từ bàn phím và đưa danh sách đó ra màn hình sau đây:

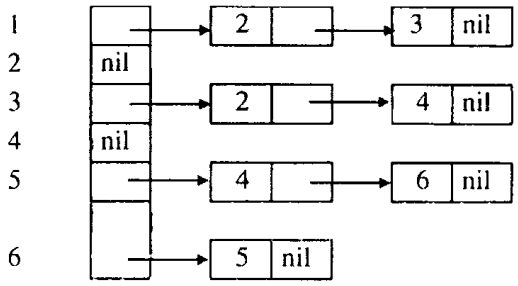
```
program AdjList;
const
     maxV = 100;
type
    link = ^node;
    node = record
              v    : integer;
              next : link
           end;
```

```
var
    j, x, y, m, n, u, v : integer;
    t                   : link;
    Ke                  : array[1..maxV] of link;

BEGIN
    write('Cho số cạnh và đỉnh của đồ thị: '); readln( m,n);
    (*  Khởi tạo  *)
    for j:=1 to n do Ke[j]:=nil;
    for j:=1 to m do
    begin
        write('Cho đỉnh đầu và cuối của cạnh ',j, ': ');
        readln(x,y);
       new(t); t^.v:=x; t^.next:=Ke[y]; Ke[y]:=t;
       new(t); t^.v:=y; t^.next:=Ke[x]; Ke[x]:=t;
    end;
    writeln('Danh sách kề của các đỉnh của đồ thị: ');
    for j:=1 to m do
    begin
        writeln('Danh sách các đỉnh kề của đỉnh ', j, ': ');
        t:=Ke[j];
        while t^.next < > nil do
        begin
            write(t^.v:4);
            t:=t^.next;
        end;
    end;
    readln;
END.
```

Thí dụ 4. Danh sách kề của các đồ thị trong hình 1 được mô tả trong hình sau:



Đỉnh đầu



Hình 2. Danh sách kề của đồ thị vô hướng G
và có hướng G_1 cho trong hình 1.

Để ý rằng trong cách biểu diễn này chúng ta cần phải sử dụng cỡ $m+n$ đơn vị bộ nhớ.

Trong các thuật toán mô tả ở các phần tiếp theo hai cấu trúc danh sách kề và ma trận trọng số được sử dụng thường xuyên.

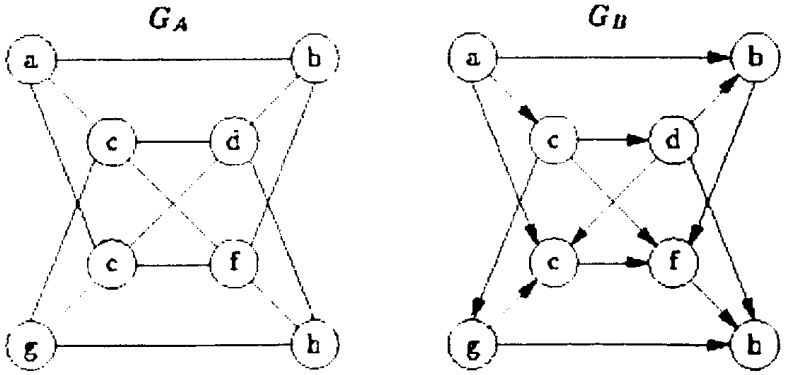
Bài tập

1. Lập trình nhập đồ thị với các cấu trúc dữ liệu đã mô tả.

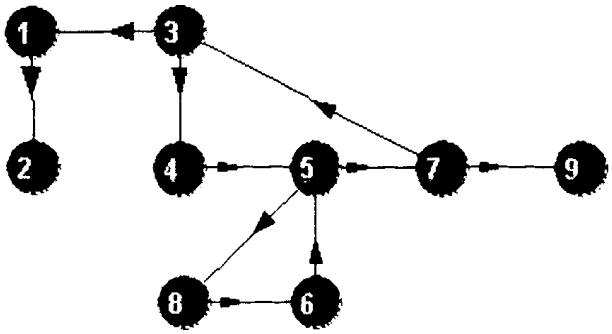
2. Lập trình cho phép chuyển đổi từ cấu trúc dữ liệu biểu diễn đồ thị dưới dạng ma trận kề sang danh sách kề và ngược lại.

3. Hãy xây dựng ma trận kề, danh sách kề của các đồ thị cho trong các hình vẽ sau đây

a)

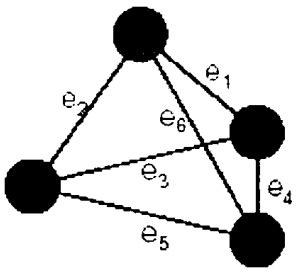


b)

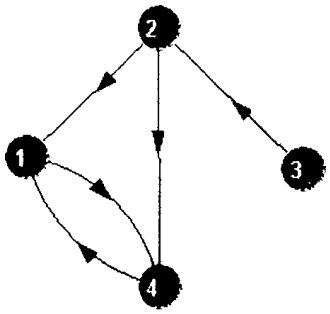


4. Hãy xây dựng ma trận liên thuộc đỉnh cạnh của các đồ thị cho trong các hình vẽ sau

a)

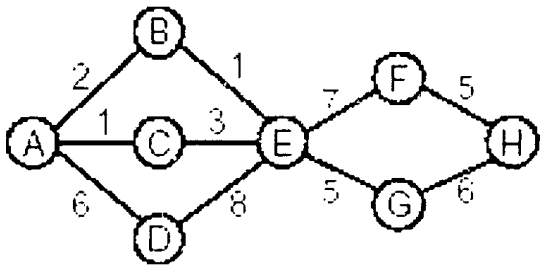


b)

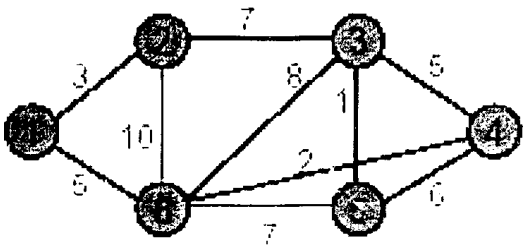


5. Hãy xây dựng ma trận trọng số của các đồ thị cho trong các hình vẽ sau

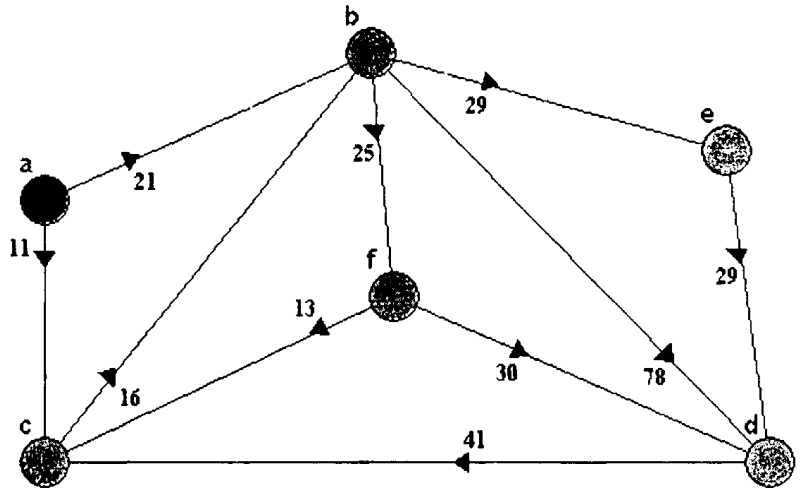
a)



b)



c)



6. Ma trận liên thuộc đỉnh cạnh có một tính chất rất đặc biệt hay được sử dụng trong việc phát triển thuật toán tối ưu trên đồ thị. Để phát biểu tính chất này ta cần khái niệm sau.

Định nghĩa. Ma trận A được gọi là ma trận hoàn toàn đơn mô đun, nếu mọi định thức con khác không của nó đều có trị tuyệt đối là bằng 1.

Chứng minh rằng ma trận liên thuộc đỉnh cạnh của đơn đồ thị có hướng là hoàn toàn đơn mô đun.

3

CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ VÀ ỨNG DỤNG

Rất nhiều thuật toán trên đồ thị được xây dựng dựa trên cơ sở duyệt tất cả các đỉnh của đồ thị sao cho mỗi đỉnh của nó được viếng thăm đúng một lần. Vì vậy, việc xây dựng những thuật toán cho phép duyệt một cách hệ thống tất cả các đỉnh của đồ thị là một vấn đề quan trọng thu hút sự quan tâm nghiên cứu của nhiều tác giả. Những thuật toán như vậy chúng ta sẽ gọi là thuật toán tìm kiếm trên đồ thị. Các thuật toán này giữ một vai trò quan trọng trong việc thiết kế các thuật toán trên đồ thị. Trong mục này chúng ta sẽ giới thiệu hai thuật toán tìm kiếm cơ bản trên đồ thị: ***Thuật toán tìm kiếm theo chiều sâu*** (Depth First Search) và ***Thuật toán tìm kiếm theo chiều rộng*** (Breadth First Search) và ứng dụng của chúng vào việc giải một số bài toán trên đồ thị.

Trong mục này chúng ta sẽ xét đồ thị vô hướng $G = (V, E)$, với n đỉnh và m cạnh. Chúng ta sẽ quan tâm đến việc đánh giá hiệu quả của các thuật toán trên đồ thị, mà một trong những đặc trưng quan trọng nhất là *độ phức tạp tính toán*, tức là *số phép toán mà thuật toán cần phải thực hiện trong tình huống xấu nhất* được biểu diễn như là *hàm của kích thước đầu vào* của bài toán. Trong các thuật toán trên đồ thị, đầu vào là đồ thị $G = (V, E)$, vì vậy, kích thước của bài toán là số đỉnh n và số cạnh m của đồ thị. Khi đó độ phức tạp tính toán của thuật toán sẽ được biểu diễn như là hàm của hai biến số $f(n,m)$ là số phép toán nhiều nhất cần phải thực hiện theo thuật toán đối với mọi đồ thị với n đỉnh

và m cạnh. Khi so sánh tốc độ tăng của hai hàm nhận giá trị không âm $f(n)$ và $g(n)$ chúng ta sẽ sử dụng ký hiệu sau:

$f(n) = O(g(n))$

$\Leftrightarrow$ tìm được các hằng số $C, N > 0$ sao cho

$f(n) \leq C\, g(n)$ với mọi $n \geq N$.

Tương tự như vậy nếu $f(n_1, n_2,..., n_k)$, $g(n_1, n_2,...,n_k)$ là các hàm nhiều biến, ta viết

$f(n_1, n_2,...,n_k) = O(g(n_1, n_2,...,n_k))$

$\Leftrightarrow$ tìm được các hằng số $C, N > 0$ sao cho

$f(n_1, n_2,...,n_k) \leq C\, g(n_1, n_2,...,n_k)$ với mọi $n_1, n_2,...,n_k \geq N$.

Nếu độ phức tạp tính toán của thuật toán là $O(g(n))$ thì ta sẽ còn nói là nó đòi hỏi thời gian tính cỡ $O(g(n))$.

3.1. Tìm kiếm theo chiều sâu trên đồ thị

Ý tưởng chính của thuật toán có thể trình bày như sau. Ta sẽ bắt đầu tìm kiếm từ một đỉnh v_0 nào đó của đồ thị. Sau đó chọn u là một đỉnh tuỳ ý kề với v_0 và lặp lại quá trình đối với u. ở bước tổng quát, giả sử ta đang xét đỉnh v. Nếu như trong số các đỉnh kề với v tìm được đỉnh w là chưa được xét thì ta sẽ xét đỉnh này (nó sẽ trở thành đã xét) và bắt đầu từ nó ta sẽ tiếp tục quá trình tìm kiếm. Còn nếu như không còn đỉnh nào kề với v là chưa xét thì ta sẽ nói rằng đỉnh này là đã duyệt xong và quay trở lại tiếp tục tìm kiếm từ đỉnh mà trước đó ta đến được đỉnh v (nếu $v = v_0$, thì kết thúc tìm kiếm). Có thể nói nôm na là tìm kiếm theo chiều sâu bắt đầu từ đỉnh v được thực hiện trên cơ sở tìm kiếm theo chiều sâu từ tất cả các đỉnh chưa xét kề với v. Quá trình này có thể mô tả bởi thủ tục đệ qui sau đây.

```
procedure DFS(v);
(*  Tìm kiếm theo chiều sâu bắt đầu từ đỉnh v;
    Các biến Chuaxet, Ke là biến toàn cục  *)
begin
     Thăm_đỉnh(v);
     Chuaxet[v]:=false;
     for u ∈ Ke(v) do
              if  Chuaxet[u] then DFS(u);
end;  (* đỉnh v là đã duyệt xong *)
```

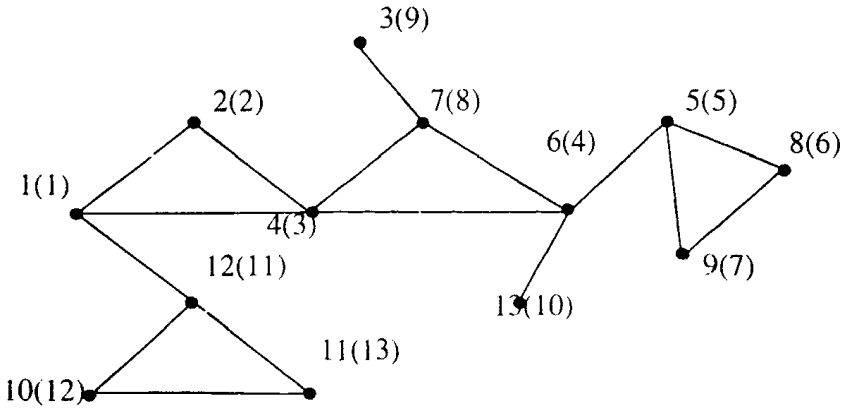
Khi đó, Tìm kiếm theo chiều sâu trên đồ thị được thực hiện nhờ thuật toán sau:

```
BEGIN
   (*  Initialization *)
    for v ∈ V do Chuaxet[v]:=true;
    for v ∈ V do
          if  Chuaxet[v] then DFS(v);
END.
```

Rõ ràng lệnh gọi DFS(v) sẽ cho phép đến thăm tất cả các đỉnh thuộc cùng thành phần liên thông với đỉnh v, bởi vì sau khi thăm đỉnh là lệnh gọi đến thủ tục DFS đối với tất cả các đỉnh kề với nó. Mặt khác, do mỗi khi thăm đỉnh v xong, biến Chuaxet[v] được đặt lại giá trị *false* nên mỗi đỉnh sẽ được thăm đúng một lần. Thuật toán lần lượt sẽ tiến hành tìm kiếm từ các đỉnh chưa được thăm, vì vậy, nó sẽ xét qua tất cả các đỉnh của đồ thị (không nhất thiết phải là liên thông).

Để đánh giá độ phức tạp tính toán của thủ tục, trước hết nhận thấy rằng số phép toán cần thực hiện trong hai chu trình của thuật toán (hai vòng for ở chương trình chính) là cỡ n. Thủ tục DFS phải thực hiện không quá n lần. Tổng số phép toán cần phải thực hiện trong các thủ tục này là $O(n+m)$, do trong các thủ tục này ta phải xét qua tất cả các cạnh và các đỉnh của đồ thị. Vậy độ phức tạp tính toán của thuật toán là $O(n+m)$.

Thí dụ 1. Xét đồ thị cho trong hình 1. Các đỉnh của nó được đánh số lại theo thứ tự chúng được thăm theo thủ tục tìm kiếm theo chiều sâu mô tả ở trên. Giả thiết rằng các đỉnh trong danh sách kề của đỉnh v (Kc(v)) được sắp xếp theo thứ tự tăng dần của chỉ số.



Hình 1. Chỉ số mới (trong ngoặc) của các đỉnh được đánh lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu

Thuật toán tìm kiếm theo chiều sâu trên đồ thị vô hướng trình bày ở trên dễ dàng có thể mô tả lại cho đồ thị có hướng. Trong trường hợp đồ thị có hướng, thủ tục DFS(v) sẽ cho phép thăm tất cả các đỉnh u nào mà từ v có đường đi đến u. Độ phức tạp tính toán của thuật toán là $O(n+m)$.

3.2. Tìm kiếm theo chiều rộng trên đồ thị

Để ý rằng trong thuật toán tìm kiếm theo chiều sâu đỉnh được thăm càng muộn sẽ càng sớm trở thành đã duyệt xong. Điều đó là hệ quả tất yếu của việc các đỉnh được thăm sẽ

được kết nạp vào trong ngăn xếp (STACK). Tìm kiếm theo chiều rộng trên đồ thị, nếu nói một cách ngắn gọn, được xây dựng dựa trên cơ sở thay thế ngăn xếp (STACK) bởi hàng đợi (QUEUE). Với sự cải biên như vậy, đỉnh được thăm càng sớm sẽ càng sớm trở thành đã duyệt xong (tức là càng sớm dời khỏi hàng đợi). Một đỉnh sẽ trở thành đã duyệt xong ngay sau khi ta xét xong tất cả các đỉnh kề (chưa được thăm) với nó. Thủ tục có thể mô tả như sau:

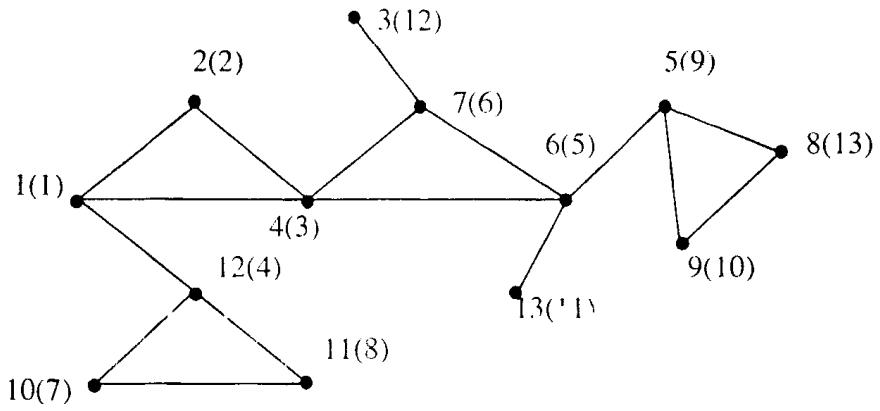
```
procedure BFS(v);
(*  Tìm kiếm theo chiều rộng bắt đầu từ đỉnh v;
    Các biến Chuaxet, Ke là biến toàn cục  *)
begin
     QUEUE:=∅;
     QUEUE ⇐ v; (* Kết nạp v vào QUEUE *)
     Chuaxet[v]:=false;
     while QUEUE ≠ ∅ do
    begin
        p ⇐ QUEUE; (* Lấy p từ QUEUE *)
      Thăm_đỉnh(p);
      for u ∈ Ke(p) do
        if  Chuaxet[u] then
        begin
           QUEUE ⇐ u; Chuaxet[u]:=false;
        end;
    end;
end;
```

Khi đó, Tìm kiếm theo chiều rộng trên đồ thị được thực hiện nhờ thuật toán sau:

```
BEGIN
    (*  Initialization *)
    for v ∈ V do Chuaxet[v]:=true;
    for v ∈ V do
          if  Chuaxet[v] then BFS(v);
END.
```

Lập luận tương tự như trong thủ tục tìm kiếm theo chiều sâu, có thể chỉ ra được rằng lệnh gọi BFS(v) sẽ cho phép đến thăm tất cả các đỉnh thuộc cùng thành phần liên thông với đỉnh v, và mỗi đỉnh của đồ thị sẽ được thăm đúng một lần. Độ phức tạp tính toán của thuật toán là $O(n+m)$.

Thí dụ 2. Xét đồ thị trong hình 2. Thứ tự thăm đỉnh của đồ thị này theo thuật toán tìm kiếm theo chiều rộng được ghi trong ngoặc.



Hình 2. Chỉ số mới (trong ngoặc) của các đỉnh được đánh lại theo thứ tự chúng được thăm trong thuật toán tìm kiếm theo chiều sâu

3.3. Tìm đường đi và kiểm tra tính liên thông

Trong mục này ta xét ứng dụng các thuật toán tìm kiếm mô tả trong các mục trước vào việc giải hai bài toán cơ bản trên đồ thị: bài toán tìm đường đi và bài toán về xác định các thành phần liên thông của đồ thị.

a) Bài toán tìm đường đi giữa hai đỉnh: *Giả sử s và t là hai đỉnh nào đó của đồ thị. Hãy tìm đường đi từ s đến t.*

Như trên đã phân tích, thủ tục DFS(s) (BFS(s)) sẽ cho phép thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với s. Vì vậy, sau khi thực hiện xong thủ tục, nếu Chuaxet[t]=true, thì điều đó có nghĩa là không có đường đi từ s đến t, còn nếu Chua xet[t]=false thì t thuộc cùng thành phần liên thông với s, hay nói một cách khác: tồn tại đường đi từ s đến t. Trong trường hợp tồn tại đường đi, để ghi nhận đường đi, ta dùng thêm biến Truoc[v] để ghi nhận đỉnh đi trước đỉnh v trong đường đi tìm kiếm từ s đến v. Khi đó, đối với thủ tục DFS(v) cần sửa sửa đổi câu lệnh **if** trong nó như sau:

```
if Chuaxet[u] then
begin
   Truoc[u]:=v;
   DFS(u);
end;
```

Còn đối với thủ tục BFS(v) cần sửa sửa đổi câu lệnh **if** trong nó như sau:

```
if Chuaxet[u] then
begin
   QUEUE ⇐ u; Chuaxet[u]:=false;
   Truoc[u]:=p;
end;
```

Đường đi cần tìm sẽ được khôi phục theo quy tắc sau:

$$t \leftarrow p1:=Truoc[t] \leftarrow p2:=Truoc[p1] \leftarrow \ldots \leftarrow s .$$

Chú ý: Đường đi tìm được theo thuật toán tìm kiếm theo chiều rộng là ***đường đi ngắn nhất*** (theo số cạnh) từ đỉnh *s* đến đỉnh *t*. Điều này suy trực tiếp từ thứ tự thăm đỉnh theo thuật toán tìm kiếm thec chiều rộng.

b) Tìm các thành phần liên thông của đồ thị: *Hãy cho biết đồ thị gồm bao nhiêu thành phần liên thông và từng thành phần liên thông của nó là gồm những đỉnh nào.*

Do thủ tục DFS(s) (BFS(s)) cho phép thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với *s*, nên số thành phần liên thông của đồ thị chính bằng số lần gọi đến thủ tục này. Vấn đề còn lại là cách ghi nhận các đỉnh trong từng thành phần liên thông. Ta dùng thêm biến *Index[v]* để ghi nhận chỉ số của thành phần liên thông chứa đỉnh *v*, và dùng thêm biến *Inconnect* để đếm số thành phần liên thông (biến này cần được khởi tạo giá trị là 0). Thủ tục Thăm_đỉnh(v) trong các thủ tục DFS(v) và BFS(v) có nhiệm vụ gán: *Index[v]:= Inconnect*, còn câu lệnh **if** trong các chương trình chính gọi đến các thủ tục này cần được sửa lại như sau

```
Inconnect:=0;
if  Chuaxet[v]  then
begin
   Inconnect:=Inconnect+1;
    DFS(v);  (* BFS(v) *)
end;
```

Kết thúc vòng lặp thứ hai trong chương trình chính, Inconect cho số thành phần liên thông của đồ thị, còn biến mảng Index[v], v $\in V$ cho phép liệt kê các đỉnh thuộc cùng một thành phần liên thông.

Chương trình trên PASCAL giải hai bài toán nói trên có thể viết như sau.

```
{ CHUONG TRINH TIM DUONG DI VA KIEM TRA TINH LIEN THONG
      THEO CAC THUAT TOAN TIM KIEM TREN DO THI
      ========================================          }
uses crt;
var
  a                          : array[1..20,1..20] of byte;
  QUEUE, Chuaxet, Truoc  : array[1..20] of byte;
  i,j,n,Solt,k,s,t     : integer;
  stop                 : boolean;
  ch                   : char;
```

```
procedure Nhapsolieu;
begin
  write('Cho so dinh cua do thi:');readln(n);
  writeln('Nhap so lieu ma tran ke:');
  for i:=1 to n do
  begin
    for j:= i+1 to n do
    begin
      write(' a[',i,',',j,']='); read(a[i,j]); a[j,i]:=a[i,j];
    end;
     a[i,i]:=0; writeln;
  end;
end;

procedure ReadFile;
var   f:text;  fn:string;
begin
  write('Cho ten file du lieu:');readln(fn);
  assign(f,fn);reset(f); readln(f,n);
  writeln('Nhap so lieu ma tran ke:');
  for i:=1 to n do
    for j:=1 to n do read(f,a[i,j]);
  close(f);
end;
procedure Insolieu;
begin
  writeln('Ma tran ke:');
  for i:=1 to n do
  begin
     for j:=1 to n do write(a[i,j]:3); writeln;
  end;
end;

procedure Ketqualienthong;
begin
   Insolieu;
   if  Solt=1 then writeln('Do thi la lien thong')
   else
   begin
      writeln('So thanh phan lien thong cua do thi la: ',Solt);
```

```
      for i:=1 to Solt do
      begin
        writeln('Thanh phan lien thong thu ',i, ' gom cac dinh:');
        for j:=1 to n do if Chuaxet[j]=i then write(j:3); writeln;
      end;
   end;
   write('Go Enter de tiep tuc...'#7);readln;
end;

procedure BFS(i:integer);
(* Tìm kiếm theo chiều rộng bắt đầu từ đỉnh i *)
var u, dauQ, cuoiQ: integer;
begin
  dauQ:=1; cuoiQ:=1;
  QUEUE[cuoiQ]:=i; Chuaxet[i]:=Solt;
  while dauQ <= cuoiQ do
  begin
    u:=QUEUE[dauQ]; dauQ:=dauQ+1;
    for j:=1 to n do
     if (a[u,j]=1)and(Chuaxet[j]=0) then
     begin
       cuoiQ:=cuoiQ+1; QUEUE[cuoiQ]:=j;
       Chuaxet[j]:=Solt;  Truoc[j]:=u;
     end;
  end;
end; { of procedure BFS }

procedure DFS(v:integer);
(* Tìm kiếm theo chiều sâu bắt đầu từ đỉnh v *)
var u:integer;
begin
   Chuaxet[v]:=Solt;
   for u:=1 to n do
    if (a[v,u]=1)and(Chuaxet[u]=0) then
    begin
      Truoc[u]:=v; DFS(u);
    end;
end;

procedure Lienthong;
begin
  { Khoi tao so lieu }
```

```
  for j:=1 to n do Chuaxet[j]:=0; Solt:=0;
  for i:=1 to n do
    if Chuaxet[i]=0 then
    begin
      Solt:=Solt+1;
      { BFS(i);} DFS(i);
    end;
  Ketqualienthong;
end;

procedure Ketquaduongdi;
begin
  if Truoc[t]=0 then writeln('Khong co duong di tu ',s,' den ',t)
  else
  begin
    writeln('Duong di tu ',s,' den ',t,' la:');
    j:=t;
    write(t,' <== ');
    while Truoc[j]<>s do
    begin
      write(Truoc[j],' <== ');
      j:=Truoc[j];
    end;
    writeln(s);
  end;
  write('Go Enter de tiep tuc...'#7);readln;
end;

procedure Duongdi;
begin
  Insolieu;
  write(' Tim duong di tu dinh: '); readln(s);
  write('          den dinh: '); readln(t);
  for j:=1 to n do   { Khoi tao so lieu }
  begin
    Truoc[j]:=0; Chuaxet[j]:=0;
  end;
  Solt:=1; BFS(s); { DFS(s); }
  Ketquaduongdi;
end;
```

```
procedure Menu;
begin
  clrscr;
  writeln('         TIM DUONG DI VA KIEM TRA TINH LIEN THONG');
  writeln('  CUA DO THI THEO THUAT TOAN TIM KIEM TREN DO THI');
  writeln('     ========================================');
  writeln('     1. Nhap so lieu tu ban phim.');
  writeln('     2. Nhap so lieu tu file.');
  writeln('     3. Kiem tra tinh lien thong.');
  writeln('     4. Tim duong di giua hai dinh.');
  writeln('     5. Thoat.');
  writeln('-------------------------------------------------------');
  write  ('    Hay go phim so de chon chuc nang...'#7);
  ch:=readkey;writeln(ch);
end;

{ Main program }
BEGIN
   repeat
     Menu;
     case ch of
       '1' : Nhapsolieu;
       '2' : ReadFile;
       '3' : Lienthong;
       '4' : Duongdi;
     end;
   until (ch='5') or (upcase(ch)='Q');
END.
```

Bài tập

1. Giả sử đồ thị $G = (V,E)$ được cho bởi danh sách kề. Hãy viết thủ tục loại bỏ cạnh (u,v), thêm cạnh (x,y) vào đồ thị.

2. Thủ tục sau đây cho phép duyệt qua tất cả các đỉnh của đồ thị được cho bởi danh sách kề:

```
procedure Find(x);
begin
      Thamdinh(x); Chuatham[x]:= Đã thăm;
      while (Trong V còn đỉnh đã thăm) do
       begin
            v:= đỉnh đã thăm nào đó;
            if (Tìm được đỉnh chưa thăm u ∈ Ke(v) ) then
            begin
                 u:= đỉnh chưa thăm đầu tiên trong Ke(v);
                 Thamdinh(u); Chuatham[u]:= Đã thăm;
            end
            else  Chuatham[v] := Đã duyệt xong;
       end;
end;

BEGIN
      for  v ∈ V do  Chuatham[v]:=Chưa thăm;
      for  v ∈ V do
            if  (Chuatham[v]= Chưa thăm)  then  Find(v);
END.
```

Hãy chỉ ra rằng cả hai thủ tục tìm kiếm theo chiều sâu và tìm kiếm theo chiều rộng đều có thể xem như trường hợp riêng của thủ tục này. Viết chương trình trên PASCAL thực hiện thuật toán mô tả ở trên.

3. Áp dụng thủ tục tìm kiếm theo chiều sâu tìm tất cả các cầu trên đồ thị vô hướng. (Cầu là cạnh mà việc loại bỏ nó làm tăng số thành phần liên thông của đồ thị).

4. Áp dụng thủ tục tìm kiếm theo chiều sâu kiểm tra xem đồ thị có hướng $G = (V,A)$ có chứa chu trình hay không?

5. Cho một bảng ô vuông gồm $m \times n$ ô, ô nằm trên dòng i cột j gọi là ô (i,j), $i = 1, 2,..., m$, $j = 1, 2,..., n$. Trong mỗi ô (i, j) của nó ta viết một số $a[i,j] \in \{0,1\}$. Từ một ô nào đó ta chỉ có thể di chuyển sang ô chứa số 1 có chung cạnh với nó . Giả sử cho ô (p, q) là ô

xuất phát, hãy viết chương trình tìm xem có cách di chuyển từ ô này ra một ô ở mép của bảng hay không? (Ô (u, v) gọi là ô ở mép bảng nếu hoặc là $u = 1$, hoặc là $u = m$, hoặc là $v = 1$, hoặc là $v = n$).

6. Cho một bảng ô vuông gồm $m \times n$ ô, ô nằm trên dòng i cột j gọi là ô (i,j), $i = 1, 2,..., m$, $j = 1, 2,..., n$. Trong mỗi ô (i, j) của nó ta viết một số a[i,j] $\in \{0,1\}$. Hãy viết chương trình đếm số *miền con toàn 0* của bảng. Ví dụ số *miền con toàn 0* của bảng kích thước 5x5 được chỉ ra trong hình sau đây

1	0	1	0	0
1	1	1	1	0
0	0	0	1	0
1	0	1	1	0
1	0	1	1	0

7. Có N người khách (N < 100) mang số hiệu từ 1 đến N. được mời đến dự tiệc. Giữa họ có một số người quen biết nhau. Dữ liệu về mối quan hệ quen biết này được cho trong một file văn bản có tên là KHACH.DAT có cấu trúc như sau: Dòng đầu tiên của nó chứa số lượng khách mời N. Mỗi dòng thứ i trong số N dòng tiếp theo chứa số hiệu của các người quen của khách i, các số hiệu được ghi cách nhau bởi ít nhất một dấu cách.

Người ta muốn xếp các khách này vào các phòng tiệc, sao cho hai khách ở trong cùng một phòng hoặc là quen biết nhau hoặc là có thể làm quen nhau thông qua những người quen biết trung gian của họ.

Viết chương trình nhập dữ liệu vào từ file, sau đó tìm cách phân khách vào các phòng tiệc sao cho số phòng phải sử dụng là ít nhất. Kết quả phân công đưa ra một file văn bản, dòng đầu tiên chứa số phòng tiệc cần sử dụng K, mỗi dòng thứ i trong số K dòng tiếp theo ghi số hiệu của khách xếp vào phòng tiệc i.

8. Bản đồ giao thông được cho bởi n nút giao thông đánh số là 1, 2,..., n và hệ thống gồm m đoạn đường $E_1, E_2,..., E_m$ mỗi đoạn nối 2 trong số n nút giao thông nói trên. Bản đồ giao thông được gọi là liên thông đơn nếu giữa hai nút giao thông bất kỳ chỉ có đúng một đường đi nối chúng. Dữ liệu cho trong file văn bản có tên GT.DAT, dòng đầu tiên chứa các số n, m, hai số ghi cách nhau bởi dấu cách. Mỗi dòng i trong số m dòng tiếp theo chứa số hiệu hai nút giao thông được nối bởi đoạn đường E_i.

Viết chương trình nhập dữ liệu vào từ file sau đó thông báo xem bản đồ có phải là liên thông đơn hay không. Trong trường hợp bản đồ là không liên thông đơn hãy tìm cách xây dựng bổ sung và loại bỏ một số đoạn đường để cho bản đồ là liên thông đơn và sao cho số đoạn đường phải xây dựng bổ sung là ít nhất. Các kết quả đưa ra màn hình.

4

ĐỒ THỊ EULER VÀ ĐỒ THỊ HAMILTON

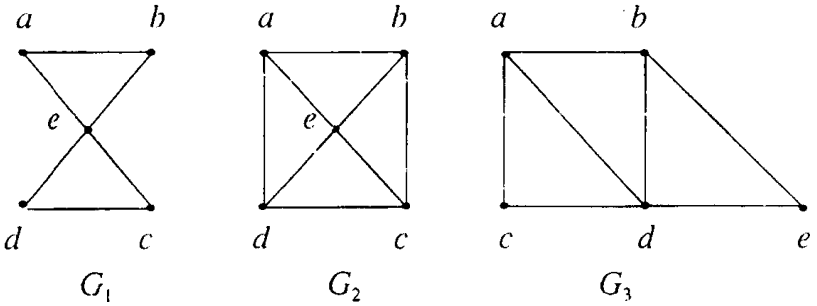
Trong chương này chúng ta sẽ nghiên cứu hai dạng đồ thị đặc biệt là đồ thị Euler và đồ thị Hamilton. Dưới đây, nếu không có giải thích bổ sung, thuật ngữ đồ thị được dùng để chỉ chung đa đồ thị vô hướng và có hướng, và thuật ngữ cạnh sẽ dùng để chỉ chung cạnh của đồ thị vô hướng cũng như cung của đồ thị có hướng.

4.1. Đồ thị Euler

Định nghĩa 1. *Chu trình đơn trong G đi qua mỗi cạnh của nó một lần được gọi là chu trình Euler. Đường đi đơn trong G đi qua mỗi cạnh của nó một lần được gọi là đường đi Euler. Đồ thị được gọi là đồ thị Euler nếu nó có chu trình Euler, và gọi là đồ thị nửa Euler nếu nó có đường đi Euler.*

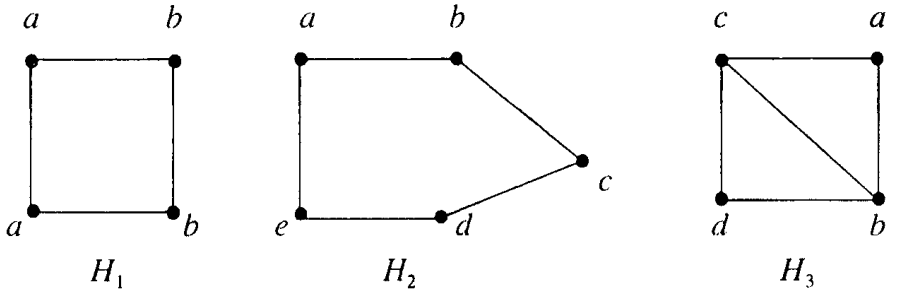
Rõ ràng mọi đồ thị Euler luôn là nửa Euler, nhưng điều ngược lại không luôn đúng.

Thí dụ 1. Đồ thị G_1 trong hình 1 là đồ thị Euler vì nó có chu trình Euler a, e, c, d, e, b, a. Đồ thị G_3 không có chu trình Euler nhưng nó có đường đi Euler a, c, d, e, b, d, a, b, vì thế G_3 là đồ thị nửa Euler. Đồ thị G_2 không có chu trình cũng như đường đi Euler.



Hình 1. Đồ thị G_1, G_2, G_3

Thí dụ 2. Đồ thị H_2 trong hình 2 là đồ thị Euler vì nó có chu trình Euler a, b, c, d, e, a. Đồ thị H_3 không có chu trình Euler nhưng nó có đường đi Euler c, a, b, c, d, b vì thế H_3 là đồ thị nửa Euler. Đồ thị H_1 không có chu trình cũng như đường đi Euler.



Hình 2. Đồ thị H_1, H_2, H_3

Điều kiện cần và đủ để một đồ thị là một đồ thị Euler được Euler tìm ra vào năm 1736 khi ông giải quyết bài toán hóc búa nổi tiếng thời đó về bảy cái cầu ở thành phố Konigsberg, và đây là định lý đầu tiên của lý thuyết đồ thị.

Định lý 1 (Euler). *Đồ thị vô hướng liên thông G là đồ thị Euler khi và chỉ khi mọi đỉnh của G đều có bậc chẵn.*

Để chứng minh định lý trước hết ta chứng minh bổ đề:

Bổ đề. *Nếu bậc của mỗi đỉnh của đồ thị G không nhỏ hơn 2 thì G chứa chu trình.*

Chứng minh. Nếu G có cạnh lặp thì khẳng định của bổ đề là hiển nhiên. Vì vậy giả sử G là đơn đồ thị. Gọi v là một đỉnh nào đó của G. Ta sẽ xây dựng theo quy nạp đường đi

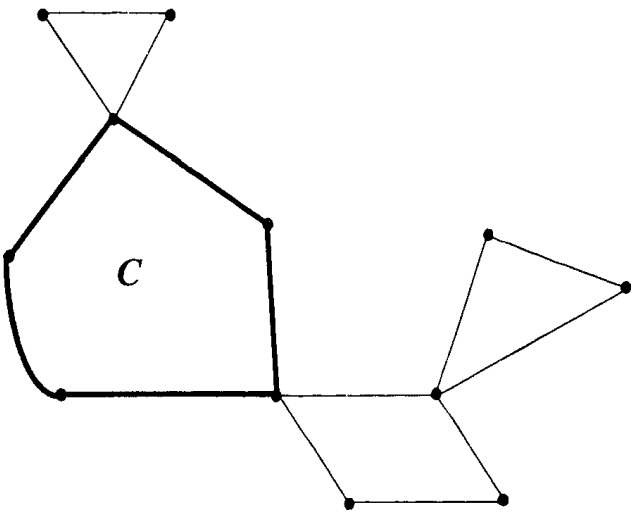
$$v \to v_1 \to v_2 \to \dots$$

trong đó v_1 là đỉnh kề với v, còn với $i \geq 1$ chọn v_{i+1} là kề với v_i và $v_{i+1} \neq v_{i-1}$ (có thể chọn v_{i+1} như vậy là vì $deg(v_i) \geq 2$). Do tập đỉnh của G là hữu hạn, nên sau một số hữu hạn bước ta phải quay lại một đỉnh đã xuất hiện trước đó. Gọi đỉnh đầu tiên như thế là v_k. Khi đó, đoạn của đường đi xây dựng nằm giữa hai đỉnh v_k là 1 chu trình cần tìm.

Chứng minh định lý.

Cần. Giả sử G là đồ thị Euler tức là tồn tại chu trình Euler P trong G. Khi đó cứ mỗi lần chu trình P đi qua 1 đỉnh nào đó của G thì bậc của đỉnh đó tăng lên 2. Mặt khác mỗi cạnh của đồ thị xuất hiện trong P đúng 1 lần, suy ra mỗi đỉnh của đồ thị đều có bậc chẵn.

Đủ. Quy nạp theo số cạnh của G. Do G liên thông và $deg(v)$ là số chẵn nên bậc của mỗi đỉnh của nó không nhỏ hơn 2. Từ đó theo bổ đề G phải chứa chu trình C. Nếu C đi qua tất cả các cạnh của G thì nó chính là chu trình Euler. Giả sử C không đi qua tất cả các cạnh của G. Khi đó loại bỏ khỏi G tất cả các cạnh thuộc C ta thu được 1 đồ thị mới H (không nhất thiết là liên thông). Số cạnh trong H nhỏ hơn trong G và rõ ràng mỗi đỉnh của H vẫn có bậc là chẵn. Theo giả thiết quy nạp trong mỗi thành phần liên thông của H đều tìm được chu trình Euler. Do G là liên thông nên mỗi thành phần trong H có ít nhất 1 đỉnh chung với chu trình C. Vì vậy, ta có thể xây dựng chu trình Euler trong G như sau: Bắt đầu từ một đỉnh nào đó của chu trình C, đi theo các cạnh của chu trình C chừng nào chưa gặp phải đỉnh không cô lập của H. Nếu gặp phải đỉnh như vậy thì ta đi theo chu trình Euler của thành phần liên thông của H chứa đỉnh đó. Sau đó lại tiếp tục đi theo cạnh của C cho đến khi gặp phải đỉnh không cô lập của H thì lại theo chu trình Euler của thành phần liên thông tương ứng trong H v.v... (xem hình 3). Quá trình sẽ kết thúc khi ta trở về đỉnh xuất phát, tức là thu được chu trình đi qua mỗi cạnh của đồ thị đúng một lần.



Hình 3. Minh hoạ cho chứng minh định lý 1

Hệ quả 2. *Đồ thị vô hướng liên thông G là nửa Euler khi và chỉ khi nó có không quá 2 đỉnh bậc lẻ.*

Chứng minh. Thực vậy nếu G có không quá 2 đỉnh bậc lẻ thì số đỉnh bậc lẻ của nó chỉ có thể là 0 hoặc 2. Nếu G không có đỉnh bậc lẻ thì theo định lý 1 nó là đồ thị Euler. Giả sử G có hai đỉnh bậc lẻ là u và v. Gọi H là đồ thị thu được từ G bằng cách thêm vào G một đỉnh mới w và hai cạnh (w,u) và (w,v). Khi đó tất cả các đỉnh của H đều có bậc là chẵn, vì thế theo định lý 1 nó có chu trình Euler C. Xoá bỏ khỏi chu trình này đỉnh w và hai cạnh kề nó ta thu được đường đi Euler trong đồ thị G.

Giả sử G là đồ thị Euler, từ chứng minh định lý ta có thủ tục sau để tìm chu trình Euler trong G.

```
procedure Euler_Cycle;
begin
    STACK := ∅ ; CE := ∅ ;
    Chọn u là một đỉnh nào đó của đồ thị;
    STACK ⇐ u;
    while STACK ≠ ∅ do
     begin
          x:= top(STACK); (*  x là phần tử ở đầu STACK *)
          if  Ke(x) ≠ ∅  then
          begin
              y:= đỉnh đầu tiên trong danh sách Ke(x);
              STACK  ⇐ y;
              (* Loại bỏ cạnh (x,y) khỏi đồ thị  *)
               Ke(x) := Ke(x) \ {y};  Ke(y) := Ke(y) \ {x};
           end  else
           begin  x  ⇐ STACK;  CE ⇐ x;  end;
    end;
end;
```

Giả sử G là đồ thị Euler, thuật toán đơn giản sau đây cho phép xác định chu trình Euler khi làm bằng tay.

Thuật toán Flor

Xuất phát từ 1 đỉnh u nào đó của G ta đi theo các cạnh của nó 1 cách tuỳ ý chỉ cần tuân thủ 2 quy tắc sau:

i) Xoá bỏ cạnh đã đi qua và đồng thời xoá cả những đỉnh cô lập tạo thành.

ii) ở mỗi bước ta chỉ đi qua cầu khi không còn cách lựa chọn nào khác.

Chứng minh tính đúng đắn của thuật toán. Trước tiên ta chỉ ra rằng thủ tục trên có thể thực hiện được ở mỗi bước. Giả sử ta đi đến một đỉnh v nào đó, khi đó nếu $v \neq u$ thì đồ thị con còn lại H là liên thông và chứa đúng hai đỉnh bậc lẻ là v và u. Theo hệ quả trong H có đường đi Euler P từ v đến u. Do việc xoá bỏ cạnh đầu tiên của đường đi P không làm mất tính liên thông của H, từ đó suy ra thủ tục có thể thực hiện ở mỗi bước. Nếu $v = u$ thì lập luận ở trên sẽ vẫn đúng chừng nào vẫn còn cạnh kề với u.

Như vậy chỉ còn phải chỉ ra là thủ tục trên dẫn đến đường đi Euler. Thực vậy trong G không thể còn cạnh chưa đi qua khi mà ta sử dụng cạnh cuối cùng kề với u (trong trường hợp ngược lại, việc loại bỏ 1 cạnh nào đó kề với 1 trong số những cạnh còn lại chưa đi qua sẽ dẫn đến một đồ thị không liên thông, và điều đó là mâu thuẫn với giả thiết ii)).

Chứng minh tương tự như trong định lý 1 ta thu được kết quả sau đây cho đồ thị có hướng.

Định lý 2. *Đồ thị có hướng liên thông mạnh là đồ thị Euler khi và chỉ khi*

$$\deg^+(v) = \deg^-(v), \quad \forall v \in V.$$

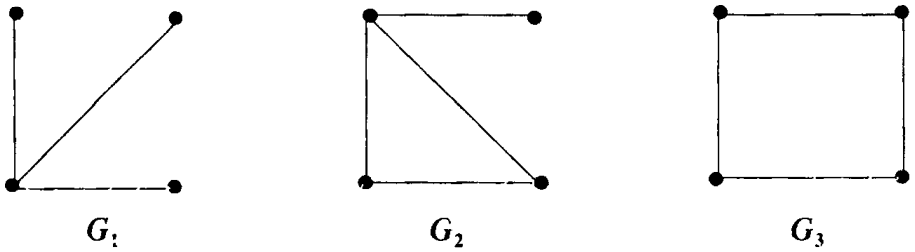
4.2. Đồ thị Hamilton

Trong mục này chúng ta xét bài toán tương tự như trong mục trước chỉ khác là ta quan tâm đến đường đi qua tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần. Sự thay đổi tưởng chừng như là không đáng kể này trên thực tế đã dẫn đến sự phức tạp hoá vấn đề cần giải quyết.

Định nghĩa 2. *Đường đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần được gọi là đường đi Hamilton. Chu trình bắt đầu từ một đỉnh v nào đó qua tất cả các đỉnh còn lại mỗi đỉnh đúng một lần rồi quay trở về v được gọi là chu trình Hamilton. Đồ thị G được gọi là đồ thị Hamilton nếu nó chứa chu trình Hamilton, và gọi là nửa Hamilton nếu nó chứa đường đi Hamilton.*

Rõ ràng đồ thị Hamilton là nửa Hamilton, nhưng điều ngược lại không luôn đúng.

Thí dụ 3. Trong hình 3: G_3 là Hamilton, G_2 là nửa Hamilton còn G_1 không là nửa Hamilton.



Hình 3. Đồ thị Hamilton G_3, nửa Hamilton G_2, và G_1

Cho đến nay việc tìm một tiêu chuẩn nhận biết đồ thị Hamilton vẫn còn là mở, mặc dù đây là một vấn đề trung tâm của lý thuyết đồ thị. Hơn thế nữa, cho đến hiện nay cũng chưa có thuật toán hiệu quả để kiểm tra một đồ thị có là Hamilton hay không. Các kết quả thu được phần lớn là các điều kiện đủ để một đồ thị là đồ thị Hamilton. Phần lớn chúng đều có dạng *"nếu G có số cạnh đủ lớn thì G là Hamilton"*. Một kết quả như vậy được phát biểu trong định lý sau đây:

Định lý 3 (Dirak 1952). *Đơn đồ thị vô hướng G với n >2 đỉnh, mỗi đỉnh có bậc không nhỏ hơn n/2 là đồ thị Hamilton.*

Chứng minh. Thêm vào đồ thị G k đỉnh mới và nối chúng với tất cả các đỉnh của G. Giả sử k là số nhỏ nhất các đỉnh cần thêm vào để cho đồ thị thu được G' là đồ thị Hamilton. Ta sẽ chỉ ra rằng $k = 0$. Thực vậy, giả sử ngược lại là $k > 0$. Ký hiệu

$$v,\ p,\ w, \ldots, v$$

là chu trình Hamilton trong G', trong đó v, w là đỉnh của G còn p là một trong số các đỉnh mới. Khi đó w không kề với v vì nếu ngược lại, ta không cần sử dụng p và điều đó là mâu thuẫn với giả thiết k nhỏ nhất. Hơn thế nữa đỉnh (w'chẳng hạn) kề với w không thể đi liền sau đỉnh v' (kề với v) vì rằng khi đó có thể thay

$$v \to p \to w \to \ldots \to v' \to w' \to \ldots \to v$$

bởi

$$v \to v' \to \ldots \to w \to w' \ldots \to v$$

bằng cách đảo ngược đoạn của chu trình nằm giữa w và v'. Từ đó suy ra là số đỉnh của đồ thị G' không kề với w là không nhỏ hơn số đỉnh kề với nó (tức là ít nhất cũng là bằng $n/2 + k$), đồng thời số đỉnh của G' kề với w cũng ít ra là phải bằng $n/2 + k$. Do không có đỉnh nào của G' vừa không kề, lại vừa kề với w, cho nên tổng số đỉnh của đồ thị G' (G' có $n + k$ đỉnh) không ít hơn $n + 2k$. Mâu thuẫn thu được đã chứng minh định lý.

Định lý sau là tổng quát hoá của định lý Dirak cho đồ thị có hướng:

Định lý 4. *Giả sử G là đồ thị có hướng liên thông mạnh với n đỉnh. Nếu*

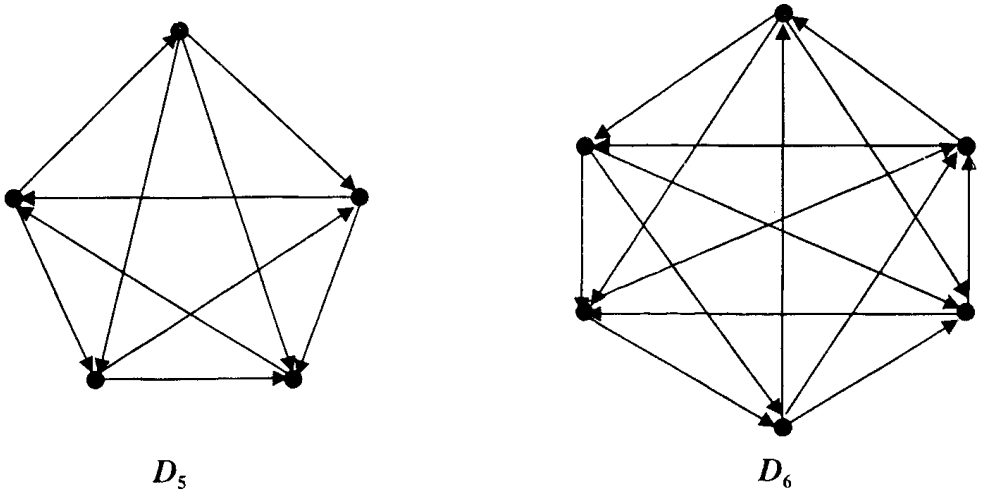
$$\deg^+(v) \geq n/2, \quad \deg^-(v) \geq n/2, \quad \forall v$$

thì G là Hamilton.

Có một số dạng đồ thị mà ta có thể biết khi nào nó là đồ thị Hamilton. Một ví dụ như vậy là đồ thị đấu loại. *Đồ thị đấu loại* là đồ thị có hướng mà trong đó 2 đỉnh bất kỳ của nó được nối với nhau bởi đúng một cung. Tên gọi *đấu loại* xuất hiện vì đồ thị như vậy có thể dùng để biểu diễn kết quả thi đấu bóng chuyền, bóng bàn hay bất cứ một trò chơi nào mà không cho phép hoà. Ta có định lý sau.

Định lý 5. *i) Mọi đồ thị đấu loại là nửa Hamilton;*
ii) Mọi đồ thị đấu loại liên thông mạnh là Hamilton.

Thí dụ 4. Đồ thị đấu loại D_5, D_6 được cho trong hình 4.



D_5 D_6

Hình 4. Đồ thị đấu loại D_5, đấu loại liên thông mạnh D_6

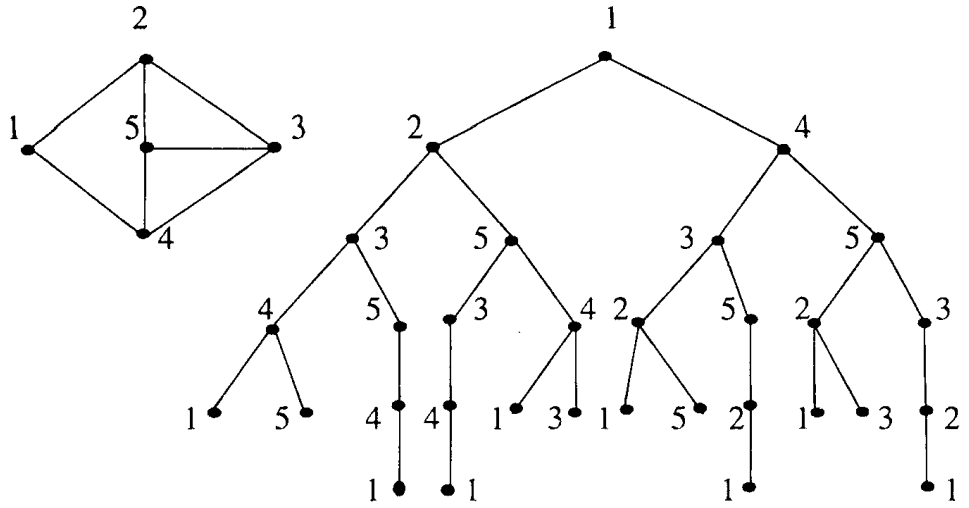
Thuật toán liệt kê tất cả các chu trình Hamilton của đồ thị.

Thuật toán sau đây được xây dựng dựa trên cơ sở thuật toán quay lui cho phép liệt kê tất cả các chu trình Hamilton của đồ thị.

```
procedure Hamilton(k);

(*  Liệt kê các chu trình Hamilton thu được bằng việc
    phát triển dãy đỉnh (X[1], . . ., X[k-1])
    của đồ thị  G=(V, E)  cho bởi danh sách kề:  Ke(v), v ∈ V
*)
```

```
begin
    for  y ∈ Ke(X[k-1]) do
        if  ( k = n+1 )  and ( y = v0 ) then  Ghinhan(X[1],...,X[n],v0)
        else
        if  Chuaxet[y]  then
        begin
            X[k] := y;
            Chuaxet[y] := false;
            Hamilton(k+1);
            Chuaxet[y] := true;
        end;
end;

(*   Main Program  *)
BEGIN
    for  v ∈ V  do  Chuaxet[v] := true;
    X[1] := v0 ;  (*  v0  là một đỉnh nào đó của đồ thị  *)
    Chuaxet[v0] := false ;
    Hamilton(2);
END.
```

Ví dụ 5. Hình 5 dưới đây mô tả cây tìm kiếm theo thuật toán vừa mô tả.



Hình 5. Đồ thị và cây liệt kê chu trình Hamilton của nó theo thuật toán quay lui

Trong trường hợp đồ thị có không quá nhiều cạnh thuật toán trên có thể sử dụng để kiểm tra xem đồ thị có phải là Hamilton hay không.

Bài tập

1. Chứng minh rằng đa đồ thị có hướng có đường đi Euler khi và chỉ khi nó là liên thông yếu và bán bậc vào và bán bậc ra của tất cả các đỉnh là bằng nhau, ngoại trừ hai đỉnh đặc biệt u, v ta có

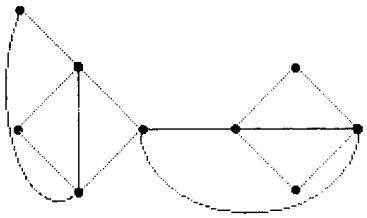
$$\deg^+(u) = \deg^-(u)+1,\ \deg^-(v) = \deg^+(v)+1$$

2. a) Với những giá trị nào của m và n đồ thị hai phía đầy đủ $K_{m,n}$ là đồ thị Euler, đồ thị nửa Euler?

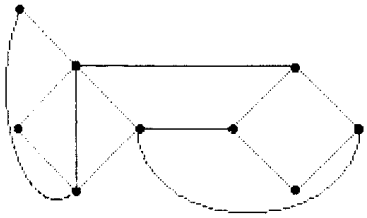
b) Chứng minh rằng đồ thị Q_n luôn có chu trình Hamilton.

3. Với mỗi đồ thị sau sử dụng thuật toán Fleury để đưa ra chu trình Euler của đồ thị hoặc chỉ ra rằng đồ thị không phải là đồ thị Euler:

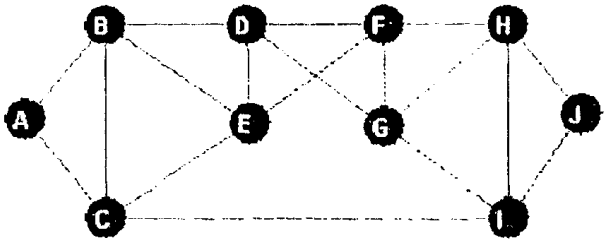
a.



b.



c)



4. Viết chương trình kiểm tra xem một đồ thị có là đồ thị Euler, đồ thị nửa Euler, và trong trường hợp câu trả lời là khẳng định, hãy tìm chu trình Euler hoặc đường đi Euler trong đồ thị theo thuật toán mô tả trong 4.1.

5. Viết chương trình kiểm tra xem một đồ thị có phải là đồ thị Hamilton, đồ thị nửa Hamilton hay không? (Sử dụng thuật toán mô tả trong 4.2)

6. Trong giải vô địch bóng chuyền có n đội đánh số từ 1 đến n thi đấu vòng tròn một lượt. Kết quả thi đấu được cho bởi một bảng số(a[i,j], i, j = 1, 2, ..., n), trong đó a[i, j]=1 nếu đội i thắng trong trận gặp đội j và a[i,j]=0, trong trường hợp đội i thua trong trận gặp đội j. Dữ liệu vào được cho trong một file văn bản có tên là KQTD.INP, dòng đầu tiên chứa số n, các dòng tiếp theo chứa các số a[i,j], i, j =1, 2,..., n, các số cách nhau

bởi dấu cách hoặc dấu xuống dòng. Kết thúc giải, ban tổ chức muốn mời các đội trưởng của đội bóng ra xếp thành một hàng ngang để chụp ảnh. Hãy tìm cách xếp các đội trưởng thành một hàng ngang sao ngoại trừ hai người đứng ở hai mép của hàng mỗi người trong hàng đều đứng cạnh một đội trưởng của đội thắng, một đội trưởng của đội thua đội mình ở trận đấu của giải. Kết quả đưa ra màn hình: thứ tự đứng các đội trưởng của các đội trong hàng ngang tìm được.

7. Tìm cách viết 9 số 1, 9 số 2, 9 số 3 thành dãy

$$a_1, a_2, \ldots, a_{27},$$

sao cho 27 số có 3 chữ số

$$\overline{a_1a_2a_3}, \overline{a_2a_3a_4}, \ldots, \overline{a_{25}a_{26}a_{27}}, \overline{a_{26}a_{27}a_1}, \overline{a_{27}a_1a_2}$$

là đôi một khác nhau.

8. Cho n xâu ký tự S_1, S_2,..., S_n, mỗi xâu có độ dài không quá 80 ký tự. Các xâu này được ghi trong một file văn bản có tên là XAU.INP, mỗi dòng của nó chứa một xâu ký tự nói trên. Giả sử P và Q là hai xâu nào đó trong các xâu đã cho. Ta nói P có thể nối được với Q nếu ký tự đầu tiên của P là trùng với ký tự cuối cùng của Q. Lập trình kiểm tra xem có thể nối n xâu đã cho thành một xâu theo quy tắc nêu trên hay không.

9. Một mạng máy tính gồm n máy tính đánh số từ 1 đến n, trong đó có một máy gọi là máy chủ được đánh số là 1. Hai máy trong mạng có thể được nối với nhau bởi nhiều hơn một kênh truyền tin. Tất cả có m kênh truyền tin giữa các máy được đánh số từ 1, 2,..., m. Dữ liệu nối mạng được cho trong một file văn bản có tên là NET.INP, mỗi dòng của nó chứa thông tin về một kênh truyền tin trong mạng gồm hai số d_i, c_i là chỉ số của hai máy được nối bởi kênh truyên tin thứ i trong mạng (i = 1, 2,..., m). Để kiểm tra các kênh truyền tin, người ta gửi một thông điệp từ máy chủ trong mạng, thông điệp này cần phải lần lượt thông qua các máy trong mạng truyền đi qua tất cả các kênh truyền tin trong mạng mỗi kênh đúng một lần rồi lại quay trở về máy chủ. Hãy lập trình nhập dữ liệu vào từ file, sau đó cho biết có cách truyền tin như vừa mô tả ở trên hay không. Trong trường hợp câu trả lời là khẳng định hãy đưa ra màn hình trình tự các kênh trong mạng mà thông điệp cần lần lượt được truyền qua.

10. Có 17 người bạn gặp nhau ở một lớp bồi dưỡng nâng cao trình độ. Trong suốt thời gian của đợt học tập họ cùng nhau ăn tối ở một nhà hàng có nhiều món ăn hợp khẩu vị với họ. Biết rằng trong tất cả các bữa ăn tối đó, họ đều ngồi quanh một cái bàn tròn và mỗi cặp chỉ ngồi cạnh nhau đúng một lần. Hỏi rằng đợt học tập đó kéo dài nhiều nhất bao nhiêu ngày? Hãy lập trình đưa ra cách xếp chỗ ngồi của họ trong những ngày đó.

5

CÂY VÀ CÂY KHUNG CỦA ĐỒ THỊ

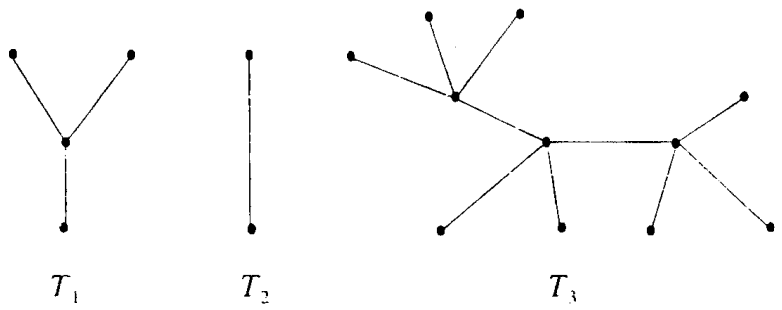
Đồ thị vô hướng liên thông không có chu trình được gọi là cây. Khái niệm cây lần đầu tiên được Cayley đưa ra vào năm 1857, khi ông sử dụng chúng để đếm một số dạng cấu trúc phân tử của các hợp chất hoá học trong hoá học hữu cơ. Cây còn được sử dụng rộng rãi trong rất nhiều lĩnh vực khác nhau, đặc biệt trong tin học, cây được sử dụng để xây dựng các thuật toán tổ chức các thư mục, các thuật toán cất giữ, truyền dữ liệu và tìm kiếm ...

5.1. Cây và các tính chất cơ bản của cây

Định nghĩa 1. *Ta gọi cây là đồ thị vô hướng liên thông không có chu trình. Đồ thị không có chu trình được gọi là rừng.*

Như vậy, rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Thí dụ 1. Trong hình 1 là một rừng gồm 3 cây T_1, T_2, T_3.



Hình 1. Rừng gồm ba cây T_1, T_2, T_3

Có thể nói cây là đồ thị vô hướng đơn giản nhất. Định lý sau đây cho ta một số tính chất của cây.

Định lý 1. *Giả sử $T=(V,E)$ là đồ thị vô hướng n đỉnh. Khi đó các mệnh đề sau đây là tương đương:*

(1) T là cây;

(2) T không chứa chu trình và có n-1 cạnh;

(3) T liên thông và có n-1 cạnh;

(4) T liên thông và mỗi cạnh của nó đều là cầu;

(5) Hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn;

(6) T không chứa chu trình nhưng hễ cứ thêm vào nó một cạnh ta thu được đúng một chu trình.

Chứng minh. Ta sẽ chứng minh định lý theo sơ đồ sau:

$$(1) \Rightarrow (2) \Rightarrow (3) \Rightarrow (4) \Rightarrow (5) \Rightarrow (6) \Rightarrow (1).$$

(1) $\Rightarrow$ (2) Theo định nghĩa T không chứa chu trình. Ta sẽ chứng minh bằng quy nạp theo số đỉnh n khẳng định: Số cạnh của cây với n đỉnh là n-1. Rõ ràng khẳng định đúng với $n = 1$. Giả sử $n > 1$. Trước hết nhận thấy rằng trong mọi cây T có n đỉnh đều tìm được ít nhất một đỉnh là đỉnh treo (tức là đỉnh có bậc là 1). Thực vậy, gọi $v_1, v_2, \ldots, v_k$ là đường đi dài nhất (theo số cạnh) trong T. Khi đó rõ ràng v_1 và v_k là các đỉnh treo, vì từ v_1 (v_k) không có cạnh nối tới bất cứ đỉnh nào trong số các đỉnh $v_2, v_3, \ldots, v_k$ (do đồ thị không chứa chu trình), cũng như tới bất cứ đỉnh nào khác của đồ thị (do đường đi đang xét là dài nhất). Loại bỏ v_1 và cạnh (v_1, v_2) khỏi T ta thu được cây T_1 với n-1 đỉnh, mà theo giả thiết qui nạp có n-2 cạnh. Vậy cây T có n-2+1 = n-1 cạnh.

(2) ⇒ (3) Ta chứng minh bằng phản chứng. Giả sử T không liên thông. Khi đó T phân rã thành $k \geq 2$ thành phần liên thông $T_1, T_2,...,T_k$. Do T không chứa chu trình nên mỗi T_i ($i = 1, 2,..., k$) cũng không chứa chu trình, vì thế mỗi T_i là cây. Do đó nếu gọi $n(T_i)$ và $e(T_i)$ theo thứ tự là số đỉnh và cạnh của T_i, ta có

$$e(T_i) = n(T_i) - 1, \quad i = 1, 2, \ldots, k,$$

suy ra

$$\begin{aligned} n - 1 = e(T) &= e(T_1) + \ldots + e(T_k) \\ &= n(T_1) + \ldots + n(T_k) - k \\ &= n(T) - k < n - 1 \text{ ?!} \end{aligned}$$

Mâu thuẫn thu được chứng tỏ T là liên thông.

(3) ⇒ (4) Việc loại bỏ một cạnh bất kỳ khỏi T dẫn đến đồ thị với n đỉnh và n-2 cạnh rõ ràng là đồ thị không liên thông. Vậy mọi cạnh trong T đều lă cầu.

(4) ⇒ (5) Do T là liên thông nên hai đỉnh bất kỳ của nó được nối với nhau bởi một đường đi đơn. Nếu có cặp đỉnh nào của T có hai đường đi đơn khác nhau nối chúng, thì từ đó suy ra đồ thị chứa chu trình, và vì thế các cạnh trên chu trình này không phải là cầu ?!

(5) ⇒ (6) T không chứa chu trình, bởi vì nếu có chu trình thì hoá ra tìm được cặp đỉnh của T được nối với nhau bởi hai đường đi đơn. Bây giờ, nếu thêm vào T một cạnh e nối hai đỉnh u và v nào đó của T. Khi đó cạnh này cùng với đường đi đơn nối u với v sẽ tạo thành chu trình trong T. Chu trình thu được này là duy nhất, vì nếu thu được nhiều hơn một chu trình thì suy ra trong T trước đó phải có sẵn chu trình.

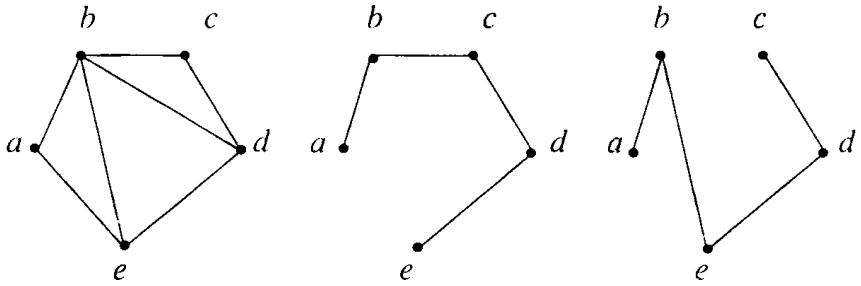
(6) ⇒ (1) Giả sử T không liên thông. Khi đó nó gồm ít ra là 2 thành phần liên thông. Vì vậy, nếu thêm vào T một cạnh nối hai đỉnh thuộc hai thành phần liên thông khăc nhau ta không thu được thêm một chu trình nào cả. Điều đó mâu thuẫn với giả thiết (6).

Định lý được chứng minh.

5.2. Cây khung của đồ thị

Định nghĩa 2. *Giả sử $G=(V,E)$ là đồ thị vô hướng liên thông. Cây $T=(V,F)$ với $F \subset E$ được gọi là* ***cây khung*** *của đồ thị G.*

Thí dụ 2. Đồ thị G và cây khung của nó được cho trong hình 2



Hình 2. Đồ thị và các cây khung của nó

Định lý sau đây cho biết số lượng cây khung của đồ thị đầy đủ K_n:

Định lý 2 (Cayley). *Số cây khung của đồ thị* K_n *là* n^{n-2}.

Định lý 2 cho thấy số lượng cây khung của một đồ thị là một số rất lớn. Bây giờ ta xét áp dụng của thuật toán tìm kiếm theo chiều sâu và theo chiều rộng trên đồ thị để xây dựng cây khung của đồ thị vô hướng liên thông. Trong cả hai trường hợp mỗi khi ta đến được đỉnh mới u (tức Chuaxet[u] = true) từ đỉnh v thì cạnh (v, u) sẽ được kết nạp vào cây khung. Hai thuật toán tương ứng được trình bày trong hai thủ tục sau đây.

```
procedure STREE_DFS(v);
(* Tìm kiếm theo chiều sâu áp dụng vào tìm tập cạnh của cây khung T
   của đồ thị vô hướng liên thông G cho bởi danh sách kề.
   Các biến Chuaxet, Ke, T là toàn cục                              *)
begin
      Chuaxet[v] := false ;
      for  u ∈ Ke(v)  do
           if  Chuaxet[u]  then
           begin
                T:= T ∪ (v,u);
                STREE_DFS(u);
           end;
end;
(*     Main Program    *)
BEGIN
      (* Initialiation *)
      for  u ∈ V  do    Chuaxet[u]:=true;
      T := ∅;              (*  T là tập cạnh của cây khung  *)
      STREE_DFS(root); (*  root   là đỉnh nào đó của đồ thị   *)
END.
```

```
procedure STRE_BFS(r);
(* Tìm kiếm theo chiều rộng áp dụng tìm tập cạnh của cây khung T
   của đồ thị vô hướng liên thông G cho bởi danh sách Ke      *)
begin
      QUEUE :=∅ ;
      QUEUE ⇐ r ;
      Chuaxet[r]:=false;
      while QUEUE ≠ ∅ do
      begin
           v ⇐ QUEUE;
           for u ∈ Ke(v) do
               if  Chuaxet[u] then
               begin
                    QUEUE ⇐ u; Chuaxet[u]:=false;
                    T:= T ∪ (v,u);
               end;
      end;
end;

(* Main Program  *)
BEGIN
      for  u ∈ V  do   Chuaxet[u]:=true;
      T := ∅;        (* T là tập cạnh của cây khung  *)
       STREE_BFS (root); (* root là một đỉnh tuỳ ý của đồ thị  *)
END.
```

Chú ý:

1. Lập luận tương tự như trong phần trước có thể chỉ ra được rằng các thuật toán mô tả ở trên có độ phức tạp tính toán $O(n+m)$.
2. Cây khung tìm được theo thủ tục STREE_BFS(r) là cây đường đi ngắn nhất từ gốc r đến tất cả các đỉnh còn lại của đồ thị.

5.3. Xây dựng tập các chu trình cơ bản của đồ thị

Bài toán xây dựng cây khung của đồ thị liên quan chặt chẽ với một bài toán ứng dụng khác của lý thuyết đồ thị: Bài toán xây dựng tập các chu trình cơ bản của đồ thị mà ta sẽ xét trong mục này.

Giả sử $G = (V, E)$ là đơn đồ thị vô hướng liên thông, $H=(V,T)$ là cây khung của nó. Các cạnh của đồ thị thuộc cây khung ta sẽ gọi là các cạnh trong, còn các cạnh còn lại sẽ gọi là cạnh ngoài.

Định nghĩa 3. *Nếu thêm một cạnh ngoài $e \in E \setminus T$ vào cây khung H chúng ta sẽ thu được đúng một chu trình trong H, ký hiệu chu trình này là C_e. Tập các chu trình*

$$\Omega = \{ C_e : e \in E \setminus T \}$$

*được gọi là **tập các chu trình cơ bản** của đồ thị G.*

Giả sử A và B là hai tập hợp, ta đưa vào phép toán sau

$$A \oplus B = (A \cup B) \setminus (A \cap B).$$

Tập $A \oplus B$ được gọi là ***hiệu đối xứng*** của hai tập hợp A và B.

Tên gọi chu trình cơ bản gắn liền với sự kiện là mỗi chu trình của đồ thị đều có thể thu được từ các chu trình cơ bản như chỉ ra trong định lý sau đây:

Định lý 3. *Giả sử $G=(V,E)$ là đồ thị vô hướng liên thông, $H=(V,T)$ là cây khung của nó. Khi đó mọi chu trình của đồ thị G đều có thể biểu diễn như là hiệu đối xứng của một số các chu trình cơ bản.*

Việc tìm tập các chu trình cơ bản giữ một vai trò quan trọng trong vấn đề giải tích mạng điện. Cụ thể hơn, theo mỗi chu trình cơ bản của đồ thị tương ứng với mạng điện cần phân tích ta sẽ thiết lập được một phương trình tuyến tính theo định luật Kirchoff: *Tổng hiệu điện thế dọc theo một mạch vòng là bằng không.* Hệ thống phương trình tuyến tính thu được cho phép tính toán hiệu điện thế trên mọi đoạn đường dây của lưới điện.

Ta sẽ mô tả thuật toán xây dựng tập các chu trình cơ bản dựa trên thủ tục tìm kiếm theo chiều sâu trên đồ thị. Thuật toán có cấu trúc tương tự như thuật toán xây dựng cây khung theo thủ tục tìm kiếm theo chiều sâu mô tả trong mục trước.

Thuật toán xây dựng tập các chu trình cơ bản.

Giả thiết rằng đồ thị $G = (V, E)$ được mô tả bằng danh sách kề $Ke(v)$, $v \in V$.

```
procedure Cycle(v);
(* Tìm tập các chu trình cơ bản của thành phần liên thông chứa đỉnh v;
   Các biến d, num, STACK, Index là toàn cục    *)
begin
    d:=d+1;
    STACK[d] := v;
    num := num+1;
```

```
    Index[v] := num;
    for  u ∈ Ke(v)  do
        if  Index[u]=0 then Cycle(u)
        else
        if  (u ≠ STACK[d-1]) and (Index[v] > Index[u]) then
            < Ghi nhận chu trình với các đỉnh:
            STACK[d], STACK[d-1], ... , STACK[c], với STACK[c]=u >;
        d := d-1;
end;

(*  Main Program  *)
BEGIN
     for  v ∈ V  do  Index[v] := 0;
      num := 0;  d := 0; STACK[0] := 0;
      for  v ∈ V  do
          if  Index[v] = 0  then  Cycle(v);
END.
```

Chú ý: Độ phức tạp tính toán của thuật toán vừa mô tả là $O(|E||V|)$.

5.4. Bài toán cây khung nhỏ nhất

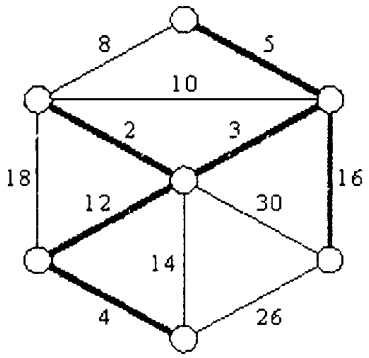
Bài toán cây khung nhỏ nhất của đồ thị là một trong số những bài toán tối ưu trên đồ thị tìm được ứng dụng trong nhiều lĩnh vực khác nhau của đời sống. Trong mục này chúng ta sẽ trình bày những thuật toán cơ bản để giải bài toán này. Trước hết chúng ta phát biểu nội dung của bài toán.

Cho $G = (V, E)$ là đồ thị vô hướng liên thông với tập đỉnh $V = \{1, 2,..., n\}$ và tập cạnh E gồm m cạnh. Mỗi cạnh e của đồ thị G được gán với một số thực $c(e)$, gọi là độ dài của nó. Giả sử $H = (V,T)$ là cây khung của đồ thị G. Ta gọi độ dài $c(H)$ của cây khung H là tổng độ dài của các cạnh của nó:

$$c(H) = \sum_{e \in T} c(e).$$

Bài toán đặt ra là trong số tất cả các cây khung của đồ thị G hãy tìm cây khung với độ dài nhỏ nhất. Cây khung như vậy được gọi là cây khung nhỏ nhất của đồ thị và bài toán đặt ra được gọi là bài toán cây khung nhỏ nhất.

Thí dụ 3. Hình 3 cho một thí dụ về đồ thị có trọng số trên cạnh và cây khung nhỏ nhất của đồ thị được chỉ ra bởi các cạnh tô đậm.



Hình 3. Cây khung nhỏ nhất của đồ thị

Để minh hoạ cho những ứng dụng của bài toán cây khung nhỏ nhất, dưới đây, ta phát biểu hai mô hình thực tế tiêu biểu của nó.

Bài toán xây dựng hệ thống đường sắt. Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất cứ một thành phố nào đến bất kỳ một trong số các thành phố còn lại. Mặt khác trên quan điểm kinh tế đòi hỏi là chi phí về xây dựng hệ thống đường phải là nhỏ nhất. Rõ ràng là đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng với phương án xây dựng tối ưu phải là cây. Vì vậy, bài toán đặt ra dẫn về bài toán tìm cây khung nhỏ nhất trên đồ thị đầy đủ n đỉnh, mỗi đỉnh tương ứng với một thành phố, với độ dài trên các cạnh chính là chi phí xây dựng đường ray nối hai thành phố tương ứng (chú ý là trong bài toán này ta giả thiết là không được xây dựng tuyến đường sắt có các nhà ga phân tuyến nằm ngoài các thành phố).

Bài toán nối mạng máy tính. Cần nối mạng một hệ thống gồm n máy vi tính đánh số từ 1 đến n. Biết chi phí nối máy i với máy j là $c[i, j]$, $i, j = 1, 2,..., n$ (thông thường chi phí này phụ thuộc vào độ dài cáp nối cần sử dụng). Hãy tìm cách nối mạng sao cho tổng chi phí nối mạng là nhỏ nhất.

Để giải bài toán cây khung nhỏ nhất, tất nhiên có thể liệt kê tất cả các cây khung của đồ thị và chọn trong số chúng cây khung nhỏ nhất. Phương pháp như vậy, trong trường hợp đồ thị đầy đủ, sẽ đòi hỏi thời gian cỡ n^{n-2}, và rõ ràng không thể thực hiện được ngay cả với những đồ thị với số đỉnh cỡ hàng chục. Rất may là đối với bài toán cây khung nhỏ nhất chúng ta đã có những thuật toán rất hiệu quả để giải chúng. Chúng ta sẽ xét hai trong số những thuật toán như vậy: Thuật toán Kruskal và Thuật toán Prim.

5.4.1. Thuật toán Kruskal

Thuật toán sẽ xây dựng tập cạnh T của cây khung nhỏ nhất $H = (V,T)$ theo từng bước. Trước hết xắp xếp các cạnh của đồ thị G theo thứ tự không giảm của độ dài. Bắt đầu từ tập $T = \varnothing$, ở mỗi bước ta sẽ lần lượt duyệt trong danh sách cạnh đã xắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bổ sung nó vào tập T không tạo thành chu trình trong tập này. Thuật toán sẽ kết thúc khi ta thu được tập T gồm $n-1$ cạnh. Cụ thể, thuật toán có thể mô tả như sau:

```
procedure  Kruskal;
begin
      T := ∅;
       while  | T | < (n-1) and ( E ≠ ∅ ) do
       begin
          Chọn e là cạnh có độ dài nhỏ nhất trong E;
          E := E \ {e};
          if  ( T ∪ {e} không chứa chu trình )  then  T := T ∪ {e};
       end;
       if  ( | T | < n-1 )  then   Đồ thị không liên thông;
end;
```

Thí dụ 4. Tìm cây khung nhỏ nhất của đồ thị cho trong hình 4.

Bước khởi tạo. Đặt $T := \varnothing$. Sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài ta có dãy:

(3,5), (4,6), (4,5), (5,6), (3,4), (1,3), (2,3), (2,4), (1,2)

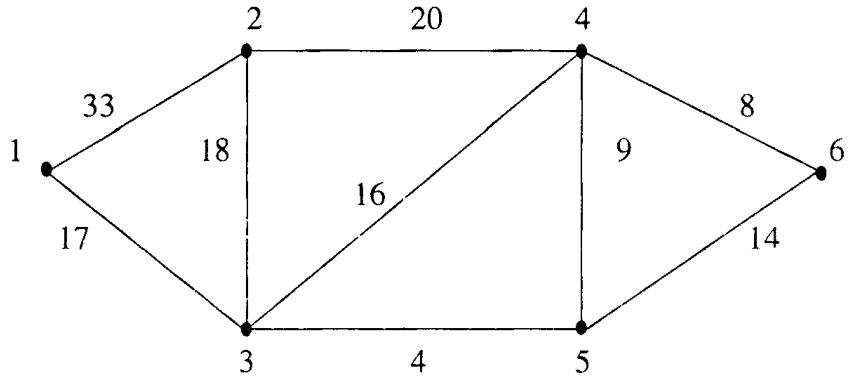
dãy độ dài tương ứng của chúng

4, 8, 9, 14, 16, 17, 18, 20, 23.

ở ba lần lặp đầu tiên ta lần lượt bổ sung vào tập T các cạnh (3,5), (4,6), (4,5). Rõ ràng nếu thêm cạnh (5,6) vào T thì nó sẽ tạo thành với 2 cạnh (4,5), (4,6) đã có trong T chu trình. Tình huống tương tự cũng xảy ra đối với cạnh (3,4) là cạnh tiếp theo trong dãy. Tiếp theo ta bổ sung cạnh (1,3), (2,3) vào T và thu được tập T gồm 5 cạnh:

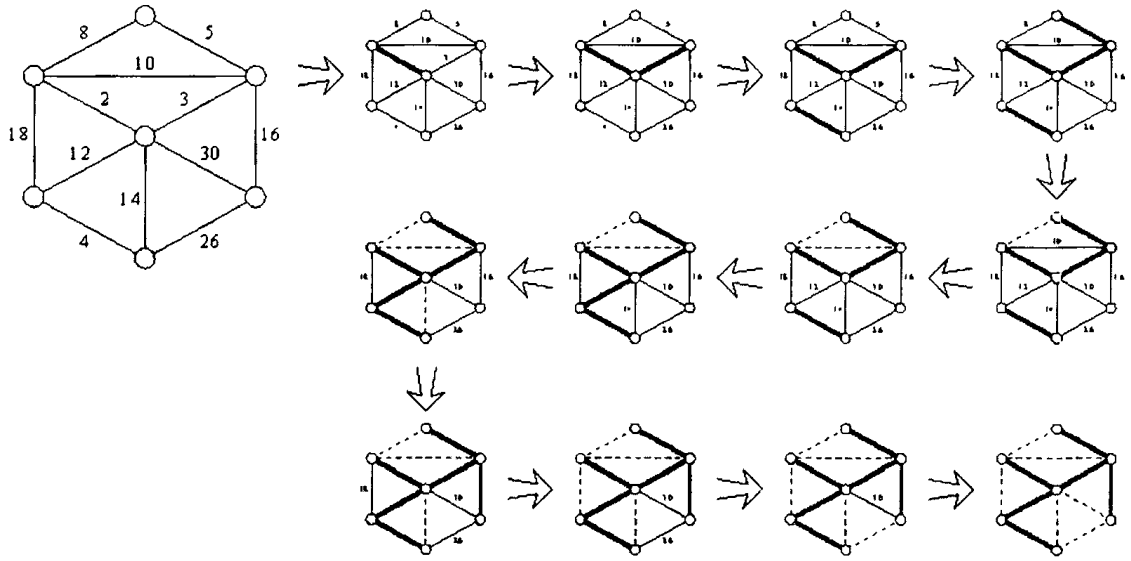
$$T = \{(3,5), (4,6), (4,5), (1,3), (2,3)\}$$

chính là tập cạnh của cây khung nhỏ nhất cần tìm.



Hình 4. Đồ thị có trọng số

Thí dụ 5. Hình 5 minh hoạ quá trình tìm cây khung nhỏ nhất của đồ thị cho trong hình 3: Các cạnh đậm là các cạnh được chọn vào cây khung, các cạnh đứt nét là các cạnh được bỏ qua trong quá trình duyệt qua các cạnh.



Hình 5. Tìm cây khung nhỏ nhất theo thuật toán Kruscal

Chứng minh tính đúng đắn của thuật toán.

Rõ ràng đồ thị thu được theo thuật toán có n-1 cạnh và không có chu trình, vì vậy theo định lý 1 nó là cây khung của đồ thị G. Như vậy, chỉ còn phải chỉ ra rằng T có độ dài nhỏ nhất. Giả sử tồn tại cây S của đồ thị G mà $c(S) < c(T)$. Ký hiệu e_k là cạnh đầu tiên trong dãy các cạnh của T xây dựng theo thuật toán vừa mô tả không thuộc S. Khi đó đồ thị con của G sinh bởi cây S được bổ sung cạnh e_k sẽ chứa 1 chu trình duy nhất C đi qua e_k. Do chu trình C phải chứa cạnh e thuộc S nhưng không thuộc T nên đồ thị con thu

được từ S bằng cách thay cạnh e của nó bởi e_k (ký hiệu đồ thị này là S') sẽ là cây khung. Theo cách xây dựng $c(e_k) \le c(e)$ do đó $c(S') \le c(S)$, đồng thời số cạnh chung của S' và T đã tăng thêm một so với số cạnh chung của S và T. Lặp lại quá trình trên từng bước một ta có thể biến đổi S thành T và trong mỗi bước tổng độ dài không tăng, tức là $c(T) \le c(S)$. Mâu thuẫn thu được chứng tỏ T là cây khung nhỏ nhất.

Về việc lập trình thực hiện thuật toán.

Khối lượng tính toán nhiều nhất của thuật toán chính là ở bước sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài để lựa chọn cạnh bổ sung. Đối với đồ thị có m cạnh cần phải thực hiện cỡ $m \log m$ phép toán để sắp xếp các cạnh của đồ thị thành dãy không giảm theo độ dài. Tuy nhiên, để xây dựng cây khung nhỏ nhất với n-1 cạnh, nói chung, ta không cần phải sắp thứ tự toàn bộ các cạnh mà chỉ cần xét phần trên của dãy đó chứa $r < m$ cạnh. Để làm việc đó ta có thể sử dụng các thủ tục sắp xếp dạng Vun đống (Heap Sort). Trong thủ tục này, để tạo đống đầu tiên ta mất cỡ O(m) phép toán, mỗi phần tử lớn tiếp theo trong đống có thể tìm sau thời gian O(log m). Vì vậy, với cải tiến này thuật toán sẽ mất thời gian cỡ O(m+p log m) cho việc sắp xếp các cạnh. Trong thực tế tính toán số p nhỏ hơn rất nhiều so với m.

Vấn đề thứ hai trong việc thể hiện thuật toán Kruskal là việc lựa chọn cạnh để bổ sung đòi hỏi phải có một thủ tục hiệu quả kiểm tra tập cạnh $T \cup \{e\}$ có chứa chu trình hay không. Để ý rằng, các cạnh trong T ở các bước lặp trung gian sẽ tạo thành một rừng. Cạnh e cần khảo sát sẽ tạo thành chu trình với các cạnh trong T khi và chỉ khi cả hai đỉnh đầu của nó thuộc vào cùng một cây con của rừng nói trên. Do đó, nếu cạnh e không tạo thành chu trình với các cạnh trong T, thì nó phải nối hai cây khác nhau trong T. Vì thế, để kiểm tra xem có thể bổ sung cạnh e vào T ta chỉ cần kiểm tra xem nó có nối hai cây khác nhau trong T hay không. Một trong các phương pháp hiệu quả để thực hiện việc kiểm tra này là ta sẽ phân hoạch tập các đỉnh của đồ thị ra thành các tập con không giao nhau, mỗi tập xác định bởi một cây con trong T (được hình thành ở các bước do việc bổ sung cạnh vào T). Chẳng hạn, đối với đồ thị trong ví dụ 3, đầu tiên ta có sáu tập con 1 phần tử:{1}, {2}, {3}, {4}, {5}, {6}. Sau khi bổ sung cạnh (3, 5), ta có các tập con {1}, {2}, {3, 5}, {4}, {6}. Tiếp theo, sau khi cạnh (4,6) được chọn, ta có các tập con {1}, {2}, {3, 5}, {4, 6}. ở bước thứ 3, ta chọn cạnh (4,5), khi đó hai tập con được nối lại và danh sách các tập con là {1}, {2}, {3, 4, 5, 6}. Cạnh có độ dài tiếp theo là (4, 6), do hai đầu của nó thuộc vào cùng một tập con {3, 4, 5, 6}, nên nó sẽ tạo thành chu trình trong tập này. Vì vậy cạnh này không được chọn. Và thuật toán sẽ tiếp tục chọn cạnh tiếp theo để khảo sát ...

Như vậy, để giải quyết vấn đề thứ hai này ta phải xây dựng hai thủ tục: Kiểm tra xem hai đầu u, v của cạnh e=(u, v) có thuộc vào hai tập con khác nhau hay không, và trong trường hợp câu trả lời là khẳng định, nối hai tập con tương ứng thành một tập.

Chú ý rằng mỗi tập con trong phân hoạch có thể lưu trữ như là một cây có gốc, và khi đó mỗi gốc sẽ được sử dụng làm nhãn nhận biết tập con tương ứng.

Chương trình trên Pascal thực hiện thuật toán Kruskal với những nhận xét vừa nêu có thể viết như sau:

```
(*   Tìm cây khung nhỏ nhất theo thuật toán Kruskal
                  của đồ thị cho bởi  danh sách cạnh          *)
uses crt;
type
   arrn = array[1..50] of integer;
   arrm = array[1..5000] of integer;
var
    n, m, MinL        : integer;
    Dau, Cuoi, W      : arrm;
    DauT, CuoiT, Father: arrn;
    Connect           : boolean;

procedure Nhapdl;
var     i : integer;
   fname: string;
   f      : text;
begin
     write('Cho Tên file dữ liệu: '); readln(fname);
     assign(f, fname);
     reset(f);
     readln(f,n,m);
     for i := 1 to m do readln( f, Dau[i], Cuoi[i], W[i] );
     close(f);
end;

procedure  Indulieu;
var  i: integer;
begin
      writeln('Số đỉnh: ',n,'.  Số cạnh: ',m);
      writeln(' Đỉnh đầu  Đỉnh cuối   Độ dài');
      for  i:=1 to m do
         writeln(Dau[i]:4, Cuoi[i]:10, W[i]:12);
end;
```

```
procedure  Heap(First, Last:integer);
var   j, k, t1, t2, t3: integer;
begin
    j := first;
    while (j <= trunc(last/2) ) do
    begin
        if (2*j < last) and (W[2*j+1] < W[2*j]) then
           k := 2*j+1
        else
           k := 2*j;
        if W[k] < W[j] then
        begin
           t1 := Dau[j];   t2 := Cuoi[j];   t3 := W[j];
           Dau[j] := Dau[k];  Cuoi[j] := Cuoi[k];  W[j] := W[k];
           Dau[k] := t1;  Cuoi[k] := t2;  W[k] := t3;
           j := k;
        end
        else j := Last;
    end;
end;

function   Find(i: integer) : integer;
var    Tro : integer;
begin
     Tro := i;
     while   Father[Tro] > 0 do
                 Tro:= Father[Tro];
     Find:=Tro;
end;

procedure  Union(i,j : integer);
var   x: integer;
begin
     x := Father[i] + Father[j];
     if Father[i]> Father[j] then
     begin
        Father[i] := j;
        Father[j] := x;
     end
     else
```

```
    begin
        Father[j] := i;
        Father[i] := x;
    end;
end;

procedure  Kruskal;
var i, Last, u, v, r1, r2, Ncanh, Ndinh : integer;
begin
    (*  Khởi tạo mảng Father đánh dấu cây con
        và khởi tạo Heap   *)
    for  i:=1  to  n  do Father[i]:=-1;
    for  i:= trunc(m/2)  downto  1  do  Heap(i,m);
    last:=m;  Ncanh := 0; Ndinh := 0;
    MinL := 0;
    Connect := true;
    while (Ndinh < n-1) and (Ncanh < m) do
    begin
        Ncanh := Ncanh+1;
        u := Dau[1]; v := Cuoi[1];
        (*  Kiểm tra u và v có thuộc cùng một cây con?  *)
        r1 := Find(u);
        r2 :=  Find(v);
        if   r1 <> r2 then
        begin
         (*   Kết nạp cạnh (u,v) vào cây khung   *)
            Ndinh:=Ndinh+1;  Union(r1,r2);
            DauT[Ndinh]:=u;
            CuoiT[Ndinh]:=v;
            MinL:=MinL+W[1];;
        end;
       (*   Tổ chức lại Heap   *)
        Dau[1] := Dau[Last];
        Cuoi[1] := Cuoi[Last];
        W[1] := W[Last];
        Last := Last-1;
        Heap(1,Last);
    end;
    if   Ndinh < > n-1  then  Connect := false;
end;
```

```
procedure Inketqua;
var    i: integer;
begin
     writeln('*************************************');
     writeln('***              Kết quả tính toán              ****');
     writeln('*************************************');
     writeln('Độ dài của cây khung nhỏ nhất: ',MinL);
     writeln('Các cạnh của cây khung nhỏ nhất:');
     for   i := 1   to  n-1  do
        writeln('(',DauT[i]:2,', ',CuoiT[i]:2,') ');
     writeln('*************************************');
end;

BEGIN
    clrscr;
    Nhapdl;
    Indulieu;
    Kruskal;
    if Connect then Inketqua
    else
       writeln('Đồ thị không liên thông');
    readln;
END.
```

File dữ liệu của bài toán trong thí dụ 4 có dạng sau:

```
6   9
3   5   4
4   6   8
4   5   9
5   6  14
3   4   16
1   3   17
2   3   18
2  4    20
1  2    23
```

Kết quả tìm được: Độ dài của cây khung nhỏ nhất và tập cạnh của cây khung nhỏ nhất được đưa ra màn hình

5.4.2. Thuật toán Prim

Thuật toán Kruskal làm việc kém hiệu quả đối với những đồ thị **dày** (đồ thị với số cạnh $m \approx n(n-1)/2$). Trong trường hợp đó thuật toán Prim tỏ ra hiệu quả hơn. Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất. Trong phương pháp này, bắt đầu từ một đỉnh tuỳ ý của đồ thị s, đầu tiên ta nối s với đỉnh lân cận gần nó nhất, chẳng hạn là đỉnh y. Nghĩa là trong số các cạnh kề của đỉnh s, cạnh (s, y) có độ dài nhỏ nhất. Tiếp theo, trong số các cạnh kề với hai đỉnh s hoặc y ta tìm cạnh có độ dài nhỏ nhất, cạnh này dẫn đến đỉnh thứ ba z, và ta thu được cây bộ phận gồm ba đỉnh và hai cạnh. Quá trình này sẽ được tiếp tục cho đến khi ta thu được cây gồm n đỉnh và n-1 cạnh sẽ chính là cây khung nhỏ nhất cần tìm.

Giả sử đồ thị cho bởi ma trận trọng số $C = \{c[i,j], i, j = 1, 2,..., n\}$. Trong quá trình thực hiện thuật toán, ở mỗi bước để có thể nhanh chóng chọn đỉnh và cạnh cần bổ sung vào cây khung, các đỉnh của đồ thị sẽ được gán cho các nhãn. Nhãn của một đỉnh v sẽ gồm hai phần và có dạng [$d[v]$, $near[v]$], trong đó $d[v]$ dùng để ghi nhận độ dài của cạnh có độ dài nhỏ nhất trong số các cạnh nối đỉnh v với các đỉnh của cây khung đang xây dựng (ta sẽ gọi là khoảng cách từ đỉnh v đến tập đỉnh của cây khung), nói một cách chính xác

$$d[v] := \min\{ c[v, w] : w \in V_H \} \ (= c[v, z]),$$

còn $near[v]$ ghi nhận đỉnh của cây khung gần v nhất ($near[v] := z$).

Thuật toán Prim được mô tả đầy đủ trong thủ tục sau:

```
procedure Prim;
begin
    (*     Bước khởi tạo      *)
    Chọn  s  là một đỉnh nào đó của đồ thị ;
    VH := { s }; T := ∅ ;
    d[s] := 0; near[s] := s.
    for  v ∈ V \ VH  do
    begin
        d[v] := c[s,v];
        near[v] := s;
    end;
    (*     Bước lặp      *)
    Stop := false;
    while  not Stop  do
    begin
        Tìm  u ∈ V \ VH  thoả mãn:  d[u] = min { d[v] : u ∈ V \ VH };
```

```
        VH := VH ∪ { u };   T := T ∪ { ( u, near[u] ) } ;
        if   | VH | = n  then
        begin
             H = ( VH , T ) là cây khung nhỏ nhất của đồ thị ;
             Stop := true;
        end
        else
        for  v∈ V \ VH  do
            if  d[v] > c[u,v] then
            begin
                d[v] := c[u,v] ;
                near[v] := u;
            end;
    end;
end;
```

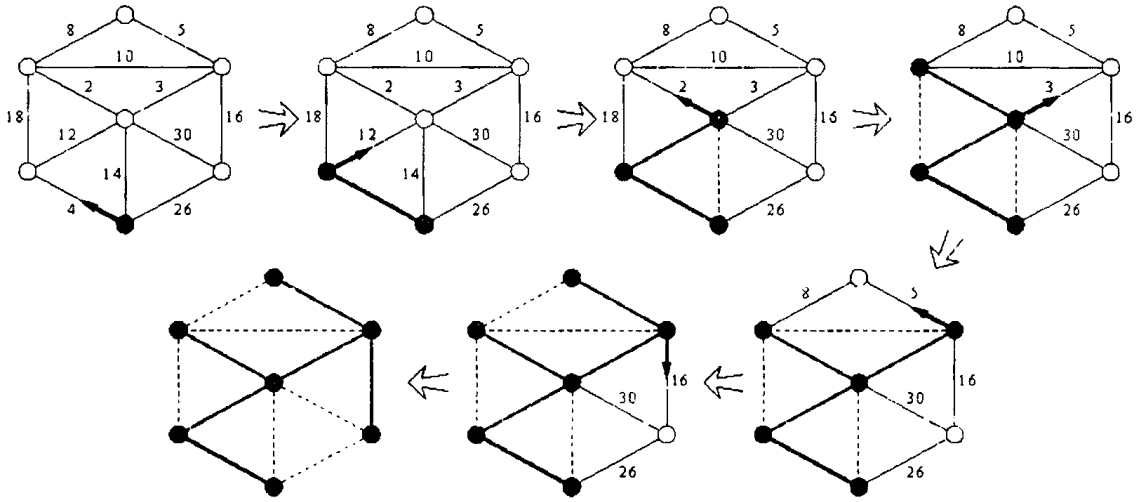
Thí dụ 6. Tìm cây khung nhỏ nhất cho đồ thị xét trong thí dụ 4 theo thuật toán Prim. Ma trận trọng số của đồ thị có dạng

$$C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 33 & 17 & \infty & \infty & \infty \\ 33 & 0 & 18 & 20 & \infty & \infty \\ 17 & 18 & 0 & 16 & 4 & \infty \\ \infty & 20 & 16 & 0 & 9 & 8 \\ \infty & \infty & 4 & 9 & 0 & 14 \\ \infty & \infty & \infty & 8 & 14 & 0 \end{bmatrix} \end{array}$$

Bảng dưới đây ghi nhãn của các đỉnh trong các bước lặp của thuật toán, đỉnh đánh dấu * là đỉnh được chọn để bổ sung vào cây khung (khi đó nhãn của nó không còn bị biến đổi trong các bước lặp tiếp theo, vì vậy ta đánh dấu - để ghi nhận điều đó) :

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6	V_H	T
Khởi tạo	[0, 1]	[33, 1]	[17, 1]*	[∞, 1]	[∞, 1]	[∞, 1]	1	∅
1	-	[18, 3]	-	[16, 3]	[4, 3]*	[∞, 1]	1, 3	(3,1)
2	-	[18, 3]	-	[9,5]*	-	[14, 5]	1, 3, 5	(3,1), (5,3)
3	-	[18, 3]	-	-	-	[8, 4]*	1, 3, 5, 4	(3,1), (5,3), (4,5)
4	-	[18, 3]*	-	-	-	-	1, 3, 5, 4, 6	(3,1), (5,3), (4,5), (6,4)
5	-	-	-	-	-	-	1, 3, 5, 4, 6, 2	(3,1), (5,3), (4,5), (6,4), (2,3)

Thí dụ 6. Hình 6 minh hoạ quá trình tìm cây khung nhỏ nhất của đồ thị cho trong hình 3: Các cạnh đậm là các cạnh được chọn vào cây khung, Mũi tên chỉ hướng chọn đỉnh tiếp theo, Cạnh đứt nét không cần phải xét trong các bước tiếp theo.



Hình 6. Tìm cây khung theo thuật toán Prim

5.4.3. Một số bài toán dẫn về bài toán cây khung nhỏ nhất

Trong mục này ta nêu hai bài toán có thể dẫn về bài toán cây khung nhỏ nhất, và vì thế có thể giải được nhờ các thuật toán mô tả trong mục trước

a) Bài toán cây khung lớn nhất. Dễ dàng nhận thấy rằng trong các thuật toán trên ta không cần sử dụng đến đòi hỏi về dấu của độ dài. Vì vậy có thể áp dụng chúng đối với đồ thị có các cạnh với độ dài có dấu tuỳ ý. Vì vậy, giả sử ta phải tìm cây lớn nhất (tức là có độ dài $c(H)$ lớn nhất) thì chỉ cần đổi dấu tất cả các độ đo và áp dụng một trong hai thuật toán vừa mô tả.

b) Bài toán tìm mạng điện với độ tin cậy lớn nhất. Cho lưới điện có n nút. Đường dây nối nút i với nút j có độ tin cậy $1 > p[i, j] > 0$, $i, j = 1, 2,..., n$. Gọi $G = (V, E)$ là đồ thị tương ứng với lưới điện này. Hãy tìm cây khung H của đồ thị G với độ tin cậy

$$\prod_{e \in H} p(e)$$

lớn nhất.

Bài toán này dẫn về bài toán tìm cây khung với tổng độ dài nhỏ nhất trên đồ thị G với độ dài của mỗi cạnh $e \in E$ là $-\log p(e)$. Thực vậy, giả sử H là cây khung nhỏ nhất trên đồ thị với độ dài $-\log p(e)$, ta có

$$-\prod_{e\in H} \log p(e) \quad \leq \quad -\prod_{e\in H'} \log p(e)$$

với mọi cây khung H' của đồ thị G. Từ đó suy ra

$$\sum_{e\in H} \log p(e) \quad \geq \quad \sum_{e\in H'} \log p(e)$$

do đó

$$\log\prod_{e\in H} p(e) \quad \geq \quad \log\prod_{e\in H'} p(e)$$

hay là

$$\prod_{e\in H} p(e) \quad \geq \quad \prod_{e\in H'} p(e)$$

với mọi cây khung H'. Vậy H là cây khung có độ tin cậy lớn nhất.

Bài tập

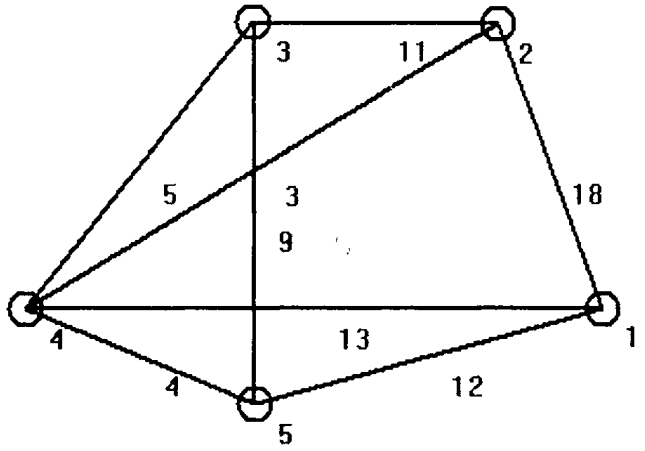
1. Hãy viết các chương trình trên Pascal thực hiện các thuật toán:

a) Xây dựng cây khung của đồ thị;

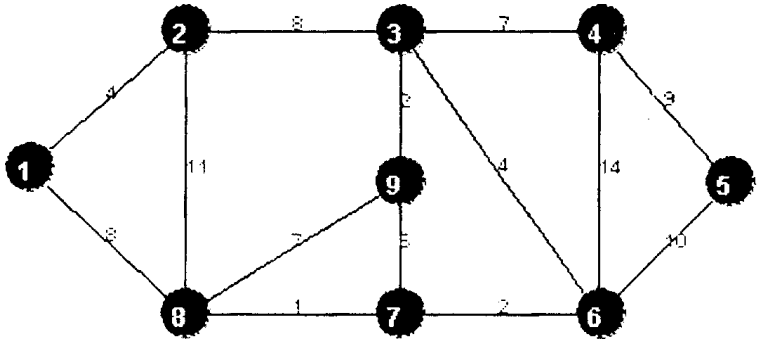
b) Xây dựng tập các chu trình cơ bản của đồ thị;

c) Xây dựng cây khung nhỏ nhất theo thuật toán Prim.

2. Áp dụng thuật toán Kruscal tìm cây khung ngắn nhất cho đồ thị sau đây

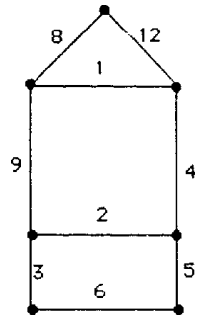


3. Áp dụng thuật toán Prim tìm cây khung ngắn nhất cho đồ thị sau đây, lấy đỉnh xuất phát là đỉnh 1

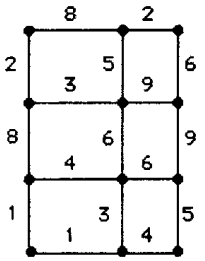


4. Tìm cây khung ngắn nhất cho đồ thị có trọng số cho trong hình vẽ dưới đây theo thuật toán Kruscal và thuật toán Prim

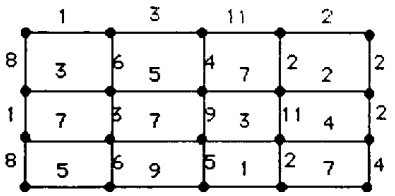
a)



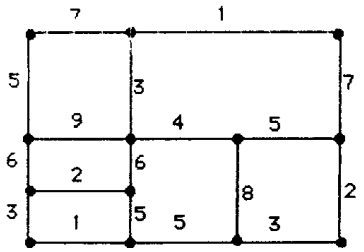
b)



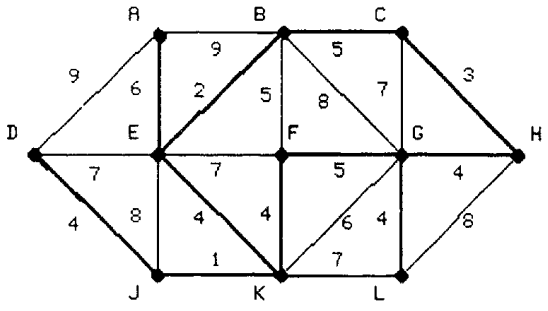
c)



d)



e)



5. Xét đồ thị gồm 6 đỉnh A, B, C, D, E, F cho bởi ma trận trọng số

	A	B	C	D	E	F
A	0	33	17	85	85	85
B	33	0	18	20	85	85
C	17	18	0	16	4	85
D	85	20	16	0	9	8
E	85	85	4	9	0	14
F	85	85	85	8	14	0

Tìm cây khung ngắn nhất theo thuật toán Prim. Yêu cầu viết rõ kết quả trung gian trong từng bước. Kết quả cuối cùng: Tập cạnh của cây khung nhỏ nhất và độ dài của nó.

6. Một công ty lập một dự án mắc điện thoại cho n (n<100) nhân viên của mình bằng một lưới các đoạn nối từ máy của một người đến máy của một số người khác (không phải hai người nào cũng được nối với nhau). Dự án phải đảm bảo yêu cầu sau (gọi là yêu cầu về ***tính thông suốt*** của mạng): ***Trên lưới điện thoại đó mỗi nhân viên của Công ty đều có thể nhắn tin cho bất cứ một nhân viên khác hoặc trực tiếp hoặc***

thông qua một số nhân viên trung gian. Dữ liệu về dự án được ghi trong file văn bản có tên NETW.INP có cấu trúc như sau:

- Dòng đầu tiên chứa số n;
- Dòng tiếp theo chứa số đường nối;
- Trong các dòng tiếp theo chứa các đường nối của dự án: mỗi dòng gồm 3 số nguyên theo thứ tự là chỉ số của hai máy được nối và chi phí nối hai máy này.

Hãy lập trình nhập dữ liệu vào tù file, sau đó:

a) Kiểm tra xem dự án có đáp ứng yêu cầu về tính thông suốt hay không?

b) Trong trường hợp dự án đáp ứng yêu cầu thông suốt, hãy tìm cách loại bỏ một số đường nối sao cho mạng vẫn là thông suốt đồng thời giảm đến mức tối thiểu chi phí nối mạng.

7. Người ta định tổ chức một mạng máy tính gồm $m \times n$ máy được bố trí theo một mạng ô vuông gồm $m \times n$ nút. Nút của lưới ở hàng i cột j gọi là nút (i,j), $i=1,2,...,m$; $j=1,2,...,n$. Mỗi máy sẽ được đặt ở một nút và các máy được nối với nhau theo cạnh của các ô vuông (gọi là các đường nối của mạng). Trong khi mạng còn đang trong quá trình xây dựng (mới chỉ có một số đường nối nào đó được xây dựng xong), vì phải đưa mạng vào phục vụ kịp thời kỳ thi Olimpic Tin học Quốc tế, người ta muốn kiểm tra xem tất cả các máy đã trao đổi thông tin được với nhau hay chưa (hai máy bất kỳ trong mạng gọi là trao đổi thông tin được với nhau nếu như chúng có thể trao đổi thông tin được với nhau hoặc là trực tiếp, hoặc thông qua các máy trung gian). Trong trường hợp câu trả lời là phủ định, cần xác định số đường nối ít nhất cần phải gấp rút xây dựng trước sao cho tất cả các máy đều trao đổi thông tin được với nhau, nhằm phục vụ kịp thời nhiệm vụ đột xuất nói trên. Dữ liệu về các đường nối đã xây dựng xong được cho trong một file văn bản có tên COMNET.INP có cấu trúc như sau

- Dòng đầu tiên chứa hai số m, n;
- Trong các dòng tiếp theo, mỗi dòng chứa thông tin về một đường nối đã xây dựng xong trong thời điểm hiện tại gồm 4 số nguyên dương xác định toạ độ của hai máy được nối với nhau bởi đường nối này.

Hãy lập trình nhập dữ liệu vào từ file, sau đó thực hiện các yêu cầu nói trên.

6

BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT

Trong các ứng dụng thực tế, Bài toán tìm đường đi ngắn nhất giữa hai đỉnh của một đồ thị liên thông có một ý nghĩa to lớn. Có thể dẫn về bài toán như vậy nhiều bài toán thực tế quan trọng. Ví dụ, Bài toán chọn một hành trình tiết kiệm nhất (theo tiêu chuẩn khoảng cách hoặc thời gian hoặc chi phí) trên một mạng giao thông đường bộ, đường thuỷ hoặc đường không; Bài toán chọn một phương pháp tiết kiệm nhất để đưa một hệ động lực lực từ trạng thái xuất phát đến một trạng thái đích, Bài toán lập lịch thi công các công đoạn trong một công trình thi công lớn, Bài toán lựa chọn đường truyền tin với chi phí nhỏ nhất trong mạng thông tin, v..v... Hiện nay có rất nhiều phương pháp để giải các bài toán như vậy. Thế nhưng, thông thường, các thuật toán được xây dựng dựa trên cơ sở lý thuyết đồ thị tỏ ra là các thuật toán có hiệu quả cao nhất. Trong chương này chúng ta sẽ xét một số thuật toán như vậy.

6.1. Các khái niệm mở đầu

Trong chương này chúng ta chỉ xét đồ thị có hướng $G = (V, E)$, $|V|=n$, $|E|=m$ với các cung được gán trọng số, nghĩa là, mỗi cung $(u, v) \in E$ của nó được đặt tương ứng với một số thực $a(u,v)$ gọi là trọng số của nó. Chúng ta sẽ đặt $a(u, v) = \infty$, nếu $(u, v) \notin E$. Nếu dãy

$$v_0, v_1, \ldots, v_p$$

là một đường đi trên G, thì độ dài của nó được định nghĩa là tổng sau

$$\sum_{i=1}^{p} a(v_{i-1}, v_i).$$

tức là, độ dài của đường đi chính là tổng các trọng số trên các cung của nó. (Chú ý rằng nếu chúng ta gán trọng số cho tất cả các cung đều bằng 1, thì ta thu được định nghĩa độ dài của đường đi như là số cung của đường đi giống như trong các chương trước đã xét).

Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể phát biểu như sau: Tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát $s \in V$ đến đỉnh cuối (đích) $t \in V$. Đường đi như vậy ta sẽ gọi là *đường đi ngắn nhất từ s đến t* còn độ dài của nó ta sẽ ký hiệu là $d(s,t)$ và còn gọi là *khoảng cách* từ s đến t (khoảng cách định nghĩa như vậy có thể là số âm). Nếu như không tồn tại đường đi từ s đến t thì ta sẽ đặt $d(s,t) = \infty$. Rõ ràng, nếu như mỗi chu trình trong đồ thị đều có độ dài dương, thì trong đường đi ngắn nhất không có đỉnh nào bị lặp lại (đường đi không có đỉnh lặp lại sẽ được gọi là *đường đi cơ bản*). Mặt khác, nếu trong đồ thị có chu trình với độ dài âm (chu trình như vậy, để ngắn gọn, ta sẽ gọi là *chu trình âm*) thì khoảng cách giữa một số cặp đỉnh nào đó của đồ thị có thể là không xác định, bởi vì, bằng cách đi vòng theo chu trình này một số đủ lớn lần, ta có thể chỉ ra đường đi giữa các đỉnh này có độ dài nhỏ hơn bất cứ số thực cho trước nào. Trong những trường hợp như vậy, có thể đặt vấn đề tìm đường đi cơ bản ngắn nhất, tuy nhiên bài toán đặt ra sẽ trở nên phức tạp hơn rất nhiều, bởi vì nó chứa bài toán xét sự tồn tại đường đi Hamilton trong đồ thị như là một trường hợp riêng. Một tính chất của đường đi ngắn nhất nữa có thể phát biểu một cách không hình thức như sau: Mọi đoạn đường con của đường đi ngắn nhất cũng là đường đi ngắn nhất. Tính chất tuy rất hiển nhiên này nhưng lại hàm chứa một nội dung rất sâu sắc, và người ta thường gọi nó là nguyên lý tối ưu. Việc chứng minh tính đúng đắn của hầu hết các thuật toán tìm đường đi ngắn nhất đều được xây dựng dựa vào nguyên lý này.

Trước hết cần chú ý rằng nếu biết khoảng cách từ s đến t, thì đường đi ngắn nhất từ s đến t, trong trường hợp trọng số không âm, có thể tìm được một cách dễ dàng. Để tìm

đường đi, chỉ cần để ý là đối với cặp đỉnh $s, t \in V$ tuỳ ý ($s \neq t$) luôn tìm được đỉnh v sao cho

$$d(s,t) = d(s,v) + a(v,t).$$

Thực vậy, đỉnh v như vậy chính là đỉnh đi trước đỉnh t trong đường đi ngắn nhất từ s đến t. Tiếp theo ta lại có thể tìm được đỉnh u sao cho $d(s,v) = d(s,u) + a(u,v)$, ... Từ giả thiết về tính không âm của các trọng số dễ dàng suy ra rằng dãy $t, v, u, ...$ không chứa đỉnh lặp lại và kết thúc ở đỉnh s. Rõ ràng dãy thu được xác định (nếu lật ngược thứ tự các đỉnh trong nó) đường đi ngắn nhất từ s đến t. Từ đó ta có thuật toán sau đây để tìm đường đi ngắn nhất từ s đến t khi biết độ dài của nó.

```
procedure Find_Path;
(*
Đầu vào:
    d[v]    - khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại v ∈ V;
    t       - đỉnh đích;
    a[u,v] , u, v ∈ V - ma trận trọng số trên các cung.
Đầu ra:
    Mảng STACK chứa dãy đỉnh xác định đường đi ngắn nhất từ s đến t
*)
begin
   STACK := ∅ ;  STACK ⇐ t ;   v := t ;
   while  v ≠ s do
   begin
       u:= đỉnh thoả mãn  d[v] = d[u] + a[u,v] ;
       STACK ⇐ u ;
       v := u ;
   end;
end;
```

Chú ý rằng độ phức tạp tính toán cuả thuật toán là $O(n^2)$, do để tìm đỉnh u ta phải xét qua tất cả các đỉnh của đồ thị. Tất nhiên, ta cũng có thể sử dụng kỹ thuật ghi nhận đường đi đã trình bày trong chương 3: dùng biến mảng Truoc[v], $v \in V$, để ghi nhớ đỉnh đi trước v trong đường đi tìm kiếm.

Cũng cần lưu ý thêm là trong trường hợp trọng số trên các cạnh là không âm, bài toán tìm đường đi ngắn nhất trên đồ thị vô hướng có thể dẫn về bài toán trên đồ thị có hướng, bằng cách thay mỗi cạnh của nó bởi hai cung có hướng ngược chiều nhau với cùng trọng số là trọng số của cạnh tương ứng. Tuy nhiên, trong trường hợp có trọng số âm, việc thay như vậy có thể dẫn đến chu trình âm.

6.2. Đường đi ngắn nhất xuất phát từ một đỉnh

Phần lớn các thuật toán tìm khoảng cách giữa hai đỉnh s và t được xây dựng nhờ kỹ thuật tính toán mà ta có thể mô tả đại thể như sau: từ ma trận trọng số $a[u,v]$, $u, v \in V$, ta tính cận trên $d[v]$ của khoảng cách từ s đến tất cả các đỉnh $v \in V$. Mỗi khi phát hiện

$$d[u] + a[u,v] < d[v] \qquad (1)$$

cận trên $d[v]$ sẽ được là tốt lên: $d[v] := d[u] + a[u,v]$.

Quá trình đó sẽ kết thúc khi nào chúng ta không làm tốt thêm được bất cứ cận trên nào. Khi đó, rõ ràng giá trị của mỗi $d[v]$ sẽ cho ta khoảng cách từ đỉnh s đến đỉnh v. Khi thể hiện kỹ thuật tính toán này trên máy tính, cận trên $d[v]$ sẽ được gọi là nhãn của đỉnh v, còn việc tính lại các cận trên này sẽ gọi là phép gán nhãn cho đồ thị và toàn bộ thủ tục thường gọi là thủ tục gán nhãn. Nhận thấy rằng để tính khoảng cách từ s đến t, ở đây, ta phải tính khoảng cách từ s đến tất cả các đỉnh còn lại của đồ thị. Hiện nay vẫn chưa biết thuật toán nào cho phép tìm đường đi ngắn nhất giữa hai đỉnh làm việc thực sự hiệu quả hơn những thuật toán tìm đường đi ngắn nhất từ một đỉnh đến tất cả các đỉnh còn lại.

Sơ đồ tính toán mà ta vừa mô tả còn chưa là xác định, bởi vì còn phải chỉ ra thứ tự chọn các đỉnh u và v để kiểm tra điều kiện (1). Thứ tự chọn này có ảnh hưởng rất lớn đến hiệu quả của thuật toán.

Bây giờ ta sẽ mô tả thuật toán Ford - Bellman tìm đường đi ngắn nhất từ đỉnh s đến tất cả các đỉnh còn lại của đồ thị. Thuật toán làm việc trong trường hợp trọng số của các cung là tuỳ ý, nhưng giả thiết rằng trong đồ thị không có chu trình âm.

```
procedure Ford_Bellman;
(*
Đầu vào: Đồ thị có hướng G=(V,E) với n đỉnh,
         s ∈ V là đỉnh xuất phát,
         a[u,v], u,v ∈ V, ma trận trọng số;
Đầu ra:  Khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại d[v], v ∈ V.
         Truoc[v], v ∈ V, ghi nhận đỉnh đi trước v trong đường đi ngắn
         nhất từ s đến v
Giả thiết: Đồ thị không có chu trình âm.
*)

begin
```

```
(*  Khởi tạo  *)
for  v ∈ V  do
begin
    d[v] := a[s,v] ;
    Truoc[v]:=s;
end;
d[s]:=0;
for k := 1  to  n-2  do
    for  v ∈ V \ {s}  do
        for  u ∈ V do
            if  d[v] > d[u] + a[u,v]  then
            begin
                d[v] := d[u] + a[u,v] ;
                Truoc[v] := u ;
            end;
end;
```

Tính đúng đắn của thuật toán có thể chứng minh trên cơ sở nguyên lý tối ưu của quy hoạch động. Rõ ràng là độ phức tạp tính toán của thuật toán là $O(n^3)$.

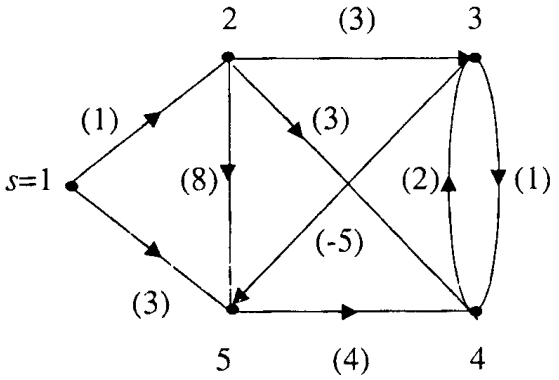
Lưu ý rằng chúng ta có thể chấm dứt vòng lặp theo k khi phát hiện trong quá trình thực hiện hai vòng lặp trong không có biến $d[v]$ nào bị đổi giá trị. Việc này có thể xảy ra đối với $k < n$-2, và điều đó làm tăng hiệu quả của thuật toán trong việc giải các bài toán thực tế. Tuy nhiên, cải tiến đó không thực sự cải thiện được đánh giá độ phức tạp của bản thân thuật toán.

Đối với đồ thị thưa tốt hơn là sử dụng danh sách kề Ke(v), $v \in V$, để biểu diễn đồ thị, khi đó vòng lặp theo u cần viết lại dưới dạng

```
for  u ∈ Ke(v)  do
    if  d[v] > d[u] + a[u,v]  then
    begin
        d[v] := d[u] + a[u,v] ;
        Truoc[v] := u ;
    end;
```

Trong trường hợp này ta thu được thuật toán với độ phức tạp $O(n.m)$.

Thí dụ 1. Xét đồ thị cho trong hình 1. Các kết quả tính toán theo thuật toán được mô tả trong bảng dưới đây



$$A = \begin{bmatrix} \infty & 1 & \infty & \infty & 3 \\ \infty & \infty & 3 & 3 & 8 \\ \infty & \infty & \infty & 1 & -5 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 4 & \infty \end{bmatrix}$$

Hình 1. Minh hoạ cho thuật toán Ford - Bellman

k	d[1], Truoc[1]	d[2], Truoc[2]	d[3], Truoc[3]	d[4], Truoc[4]	d[5], Truoc[5]
	0, 1	1, 1	∞, 1	∞, 1	3, 1
1	0, 1	1, 1	4, 2	4, 2	-1, 3
2	0, 1	1, 1	4, 2	3, 5	-1, 3
3	0, 1	1, 1	4, 2	3, 5	-1, 3

Bảng kết quả tính toán theo thuật toán Ford - Bellman

Trong các mục tiếp theo chúng ta sẽ xét một số trường hợp riêng của bài toán tìm đường đi ngắn nhất mà để giải chúng có thể xây dựng những thuật toán hiệu quả hơn thuật toán Ford - Bellman. Đó là khi trọng số của tất cả các cung là các số không âm hoặc là khi đồ thị không có chu trình.

6.3. Trường hợp ma trận trọng số không âm. Thuật toán Dijkstra

Trong trường hợp trọng số trên các cung là không âm thuật toán do Dijkstra đề nghị để giải bài toán tìm đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị làm việc hữu hiệu hơn rất nhiều so với thuật toán trình bày trong mục trước. Thuật toán được xây dựng dựa trên cơ sở gán cho các đỉnh các nhãn tạm thời. Nhãn của mỗi đỉnh cho biết cận trên của độ dài đường đi ngắn nhất từ s đến nó. Các nhãn này sẽ được biến đổi theo một thủ tục lặp, mà ở mỗi một bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh nào đó trở thành cố định thì nó sẽ cho ta không phải là cận trên mà là độ dài của đường đi ngắn nhất từ đỉnh s đến nó. Thuật toán được mô tả cụ thể như sau.

```
procedure Dijkstra;
(* Đầu vào: Đồ thị có hướng G=(V,E) với n đỉnh,
            s ∈ V là đỉnh xuất phát, a[u,v], u,v ∈ V, ma trận trọng số;
   Giả thiết: a[u,v] ≥ 0, u, v ∈ V.
   Đầu ra: Khoảng cách từ đỉnh s đến tất cả các đỉnh còn lại d[v], v∈ V.
   Truoc[v], v∈ V, ghi nhận đỉnh đi trước v trong đường đi ngắn nhất từ s đến v.
*)
begin
   (* Khởi tạo *)
   for v ∈ V do
   begin
      d[v] := a[s,v] ;
      Truoc[v]:=s;
   end;
   d[s] := 0; T := V \ {s}; (* T là tập các đỉnh có nhãn tạm thời *)
   (* Bước lặp *)
   while T ≠ ∅ do
   begin
      Tìm đỉnh u ∈ T thoả mãn d[u] = min{ d[z] : z ∈ T};
      T := T\{u}; (* Cố định nhãn của đỉnh u *)
      for v ∈ T do (* Gán nhãn lại cho các đỉnh trong T *)
         if d[v] > d[u] + a[u,v] then
         begin
            d[v] := d[u] + a[u,v] ;
            Truoc[v] := u ;
         end;
   end;
end;
```

Định lý 1. *Thuật toán Dijkstra tìm được đường đi ngắn nhất trên đồ thị sau thời gian cỡ* $O(n^2)$.

Chứng minh. Trước hết ta chứng minh là thuật toán tìm được đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị. Giả sử rằng ở một bước lặp nào đó các nhãn cố định cho ta độ dài các đường đi ngắn nhất từ s đến các đỉnh có nhãn cố định, ta sẽ chứng minh rằng ở lần lặp tiếp theo nếu đỉnh u^* thu được nhãn cố định thì $d[u^*]$ chính là độ dài đường đi ngắn nhất từ s đến u^*.

Ký hiệu S_1 là tập các đỉnh có nhãn cố định còn S_2 là tập các đỉnh có nhãn tạm thời ở bước lặp đang xét. Kết thúc mỗi bước lặp nhãn tạm thời $d[v]$ cho ta độ dài của đường đi ngắn nhất từ s đến v chỉ qua những đỉnh nằm hoàn toàn trong tập S_1. Giả sử rằng đường đi ngắn nhất từ s đến u^* không nằm trọn trong tập S_1, tức là nó đi qua ít nhất một đỉnh của tập S_2. Gọi $z \in S_2$ là đỉnh đầu tiên như vậy trên đường đi này. Do trọng số trên các cung là không âm, nên đoạn đường từ z đến u^* có độ dài $L > 0$ và

$$d[z] < d[u^*] - L < d[u^*].$$

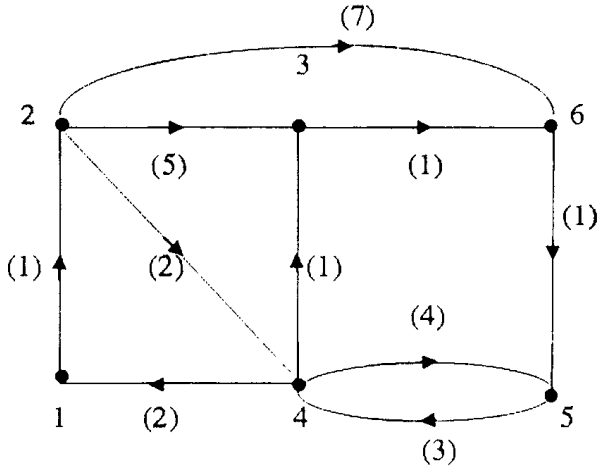
Bất đẳng thức này là mâu thuẫn với cách xác định đỉnh u^* là đỉnh có nhãn tạm thời nhỏ nhất. Vậy đường đi ngắn nhất từ s đến u^* phải nằm trọn trong S_1, và vì thế $d[u^*]$ là độ dài của nó. Do ở lần lặp đầu tiên $S_1 = \{s\}$ và sau mỗi lần lặp ta chỉ thêm vào S_1 một đỉnh u^* nên giả thiết là $d[v]$ cho độ dài đường đi ngắn nhất từ s đến v với mọi $v \in S_1$ là đúng với bước lặp đầu tiên. Theo qui nạp suy ra thuật toán cho ta đường đi ngắn nhất từ s đến mọi đỉnh của đồ thị.

Bây giờ sẽ đánh giá số phép toán cần thực hiện theo thuật toán. Ở mỗi bước lặp để tìm ra đỉnh u cần phải thực hiện $O(n)$ phép toán, và để gán nhãn lại cũng cần phải thực hiện một số lượng phép toán cũng là $O(n)$. Thuật toán phải thực hiện n-1 bước lặp, vậy thời gian tính toán của thuật toán là cỡ $O(n^2)$.

Định lý được chứng minh.

Khi đã tìm được độ dài của đường đi ngắn nhất $d[v]$ thì đường đi này có thể tìm dựa vào nhãn $Truoc[v]$, $v \in V$, theo qui tắc giống như chúng ta đã xét trong chương 3.

Thí dụ 2. Tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị ở hình 2.



Hình 2. Minh hoạ thuật toán Dijkstra

Kết quả tính toán theo thuật toán được trình bày trong bảng dưới đây. Qui ước viết hai thành phần của nhãn theo thứ tự: $d[v]$, $Truoc[v]$. Đỉnh được đánh dấu '*' là đỉnh

được chọn để cố định nhãn ở bước lặp đang xét, nhãn của nó không biến đổi ở các bước tiếp theo, vì thế ta đánh dấu -.

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6
Khởi tạo	0, 1	1, 1 *	∞, 1	∞, 1	∞, 1	∞, 1
1	-	-	6, 2	3, 2 *	∞, 1	8, 2
2	-	-	4, 4 *	-	7, 4	8, 2
3	-	-	-	-	7, 4	5, 3 *
4	-	-	-	-	6, 6 *	-
5						

Bảng kết quả tính toán theo thuật toán Dijkstra

Chú ý:

1) Nếu chỉ cần tìm đường đi ngắn nhất từ s đến một đỉnh t nào đó thì có thể kết thúc thuật toán khi đỉnh t trở thành có nhãn cố định.

2) Tương tự như trong mục 6.2, dễ dàng mô tả lại thuật toán cho trường hợp đồ thị cho bởi danh sách kề. Để có thể giảm bớt khối lượng tính toán trong việc xác định đỉnh u ở mỗi bước lặp, có thể sử dụng thuật toán Heapsort (tương tự như trong chương 5 khi thể hiện thuật toán Kruskal). Khi đó có thể thu được thuật toán với độ phức tạp tính toán là $O(m \log n)$.

6.4. Đường đi trong đồ thị không có chu trình

Bây giờ ta xét trường hợp riêng thứ hai của bài toán đường đi ngắn nhất, mà để giải nó có thể xây dựng thuật toán với độ phức tạp tính toán $O(n^2)$, đó là khi đồ thị không có chu trình (còn trọng số trên các cung có thể là các số thực tuỳ ý). Trước hết ta chứng minh định lý sau.

Định lý 2. *Giả sử G là đồ thị không có chu trình. Khi đó các đỉnh của nó có thể đánh số sao cho mỗi cung của đồ thị chỉ hướng từ đỉnh có chỉ số nhỏ hơn đến đỉnh có chỉ số lớn hơn, nghĩa là mỗi cung của nó có thể biểu diễn dưới dạng* $(v[i], v[j])$, *trong đó* $i < j$.

Để chứng minh định lý ta mô tả thuật toán sau đây, cho phép tìm ra cách đánh số thoả mãn điều kiện định lý.

procedure Numbering;

(* **Đầu vào:** *Đồ thị có hướng* $G=(V,E)$ *với n đỉnh không chứa chu trình được cho bởi danh sách kề* $Ke(v)$, $v \in V$.

Đầu ra: *Với mỗi đỉnh $v \in V$ chỉ số $NR[v]$ thoả mãn:*
*Với mọi cung (u,v) của đồ thị ta đều có $NR[u] < NR[v]$. *)*

```
begin
    for  v ∈ V do  Vao[v] := 0;
    (* Tính Vao[v] = deg-(v) *)
    for  u ∈ V do
       for  v ∈ Ke(u) do  Vao[v] := Vao[v] + 1 ;
     QUEUE := ∅ ;
     for  v ∈ V do
          if  Vao[v] = 0 then QUEUE ⇐ v ;
     num := 0;
     while QUEUE ≠ ∅ do
     begin
          u ⇐ QUEUE ;
          num := num + 1 ;  NR[u] := num ;
          for  v ∈ Ke(u) do
          begin
              Vao[v] := Vao[v] - 1 ;
              if  Vao[v] = 0 then QUEUE ⇐ v ;
          end;
     end;
end;
```

Thuật toán được xây dựng dựa trên ý tưởng rất đơn giản sau: Rõ ràng trong đồ thị không có chu trình bao giờ cũng tìm được đỉnh có bán bậc vào bằng 0 (không có cung đi vào). Thực vậy, bắt đầu từ đỉnh v_1 nếu có cung đi vào nó từ v_2 thì ta lại chuyển sang xét đỉnh v_2. Nếu có cung từ v_3 đi vào v_2, thì ta lại chuyển sang xét v_3, ... Do đồ thị là không có chu trình nên sau một số hữu hạn lần chuyển như vậy ta phải đi đến đỉnh không có cung đi vào. Thoạt tiên, tìm các đỉnh như vậy của đồ thị. Rõ ràng ta có thể đánh số chúng theo một thứ tự tuỳ ý bắt đầu từ 1. Tiếp theo, loại bỏ khỏi đồ thị những đỉnh đã được đánh số cùng các cung đi ra khỏi chúng, ta thu được đồ thị mới cũng không có chu trình, và thủ tục được lặp lại với đồ thị mới này. Quá trình đó sẽ được tiếp tục cho đến khi tất cả các đỉnh của đồ thị được đánh số.

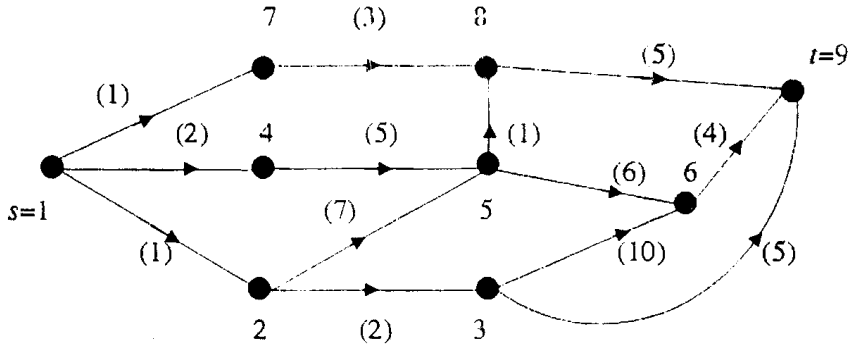
Chú ý:

1) Rõ ràng trong bước khởi tạo ta phải duyệt qua tất cả các cung của đồ thị khi tính bán bậc vào của các đỉnh, vì vậy ở đó ta tốn cỡ $O(m)$ phép toán, trong đó m là số cung của đồ thị. Tiếp theo, mỗi lần đánh số một đỉnh, để thực hiện việc loại bỏ đỉnh đã đánh số cùng với các cung đi ra khỏi nó, chúng ta lại duyệt qua tất cả các cung này. Suy ra để

đánh số tất cả các đỉnh của đồ thị chúng ta sẽ phải duyệt qua tất cả các cung của đồ thị một lần nữa. Vậy độ phức tạp của thuật toán là O(m).

2) Thuật toán có thể áp dụng để kiểm tra xem đồ thị có chứa chu trình hay không? Thực vậy, nếu kết thúc thuật toán vẫn còn có đỉnh chưa được đánh số ($num < n$) thì điều đó có nghĩa là đồ thị chứa chu trình.

Thí dụ 3. Đồ thị trong hình 3 có các đỉnh được đánh số thoả mãn điều kiện nêu trong định lý.



Hình 3. Đồ thị không có chu trình

Do có thuật toán đánh số trên, nên khi xét đồ thị không có chu trình ta có thể giả thiết là các đỉnh của nó được đánh số sao cho mỗi cung chỉ đi từ đỉnh có chỉ số nhỏ đến đỉnh có chỉ số lớn hơn. Thuật toán tìm đường đi ngắn nhất trên đồ thị không có chu trình được mô tả trong sơ đồ sau đây.

procedure Critical_Path;

(* ***Tìm đường đi ngắn nhất từ đỉnh nguồn đến tất cả các đỉnh còn lại trênđồ thị không có chu trình***

Đầu vào: *Đồ thị $G = (V, E)$, trong đó $V = \{ v[1], v[2], \ldots, v[n] \}$.*
Đối với mỗi cung $(v[i], v[j]) \in E$, ta có $i < j$.
Đồ thị được cho bởi danh sách kề $Ke(v)$, $v \in V$.

Đầu ra : *Khoảng cách từ $v[1]$ đến tất cả các đỉnh còn lại được ghi trong mảng $d[v[i]]$, $i = 2, 3, \ldots, n$* *)

```
begin
    d[v[1]] := 0;
    for  j:= 2  to  n do
         d[v[j]] := a[v[1], v[j] ] ;
    for  j:= 2  to  n  do
         for  v ∈ Ke[v[j]]  do
                d[v] :=  min ( d[v],  d[v[j]] + a[v[j], v] ) ;
end;
```

Độ phức tạp tính toán của thuật toán là O(m), do mỗi cung của đồ thị phải xét qua đúng một lần.

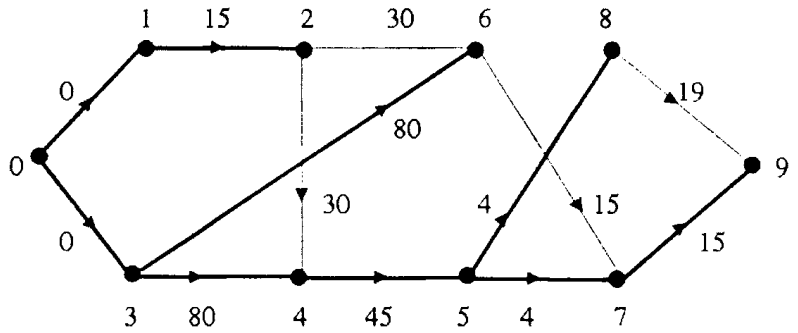
Các thuật toán mô tả ở trên thường được ứng dụng vào việc xây dựng những phương pháp giải bài toán điều khiển việc thực hiện những dự án lớn, gọi tắt là PERT (Project Evaluation and Review Technique) hay CDM (Critical path Method). Một thí dụ đơn giản cho ứng dụng này được mô tả trong thí dụ dưới đây.

Thí dụ 4. Việc thi công một công trình lớn được chia ra làm n công đoạn, đánh số từ 1 đến n. Có một số công đoạn mà việc thực hiện nó chỉ được tiến hành sau khi một số công đoạn nào đó đã hoàn thành. Đối với mỗi công đoạn i biết $t[i]$ là thời gian cần thiết để hoàn thành nó (i = 1, 2,..., n). Các dữ liệu với n = 8 được cho trong bảng sau đây

Công đoạn	t[i]	Các công đoạn phải được hoàn thành trước nó
1	15	không có
2	30	1
3	80	không có
4	45	2 , 3
5	124	4
6	15	2 , 3
7	15	5 , 6
8	19	5

Giả sử thời điểm bắt đầu tiến hành thi công công trình là 0. Hãy tìm tiến độ thi công công trình (chỉ rõ mỗi công đoạn phải được bắt đầu thực hiện vào thời điểm nào) để cho công trình được hoàn thành xong trong thời điểm sớm nhất có thể được.

Ta có thể xây dựng đồ thị có hướng n đỉnh biểu diễn hạn chế về trình tự thực hiệc các công việc như sau: Mỗi đỉnh của đồ thị tương ứng với một công việc, nếu công việc i phải được thực hiện trước công đoạn j thì trên đồ thị có cung (i,j), trọng số trên cung này được gán bằng t[i], xem hình 4 dưới đây.



Hình 4. Đồ thị minh hoạ PERT

Thêm vào đồ thị 2 đỉnh 0 và $n+1$ tương ứng với hai sự kiện đặc biệt: đỉnh số 0 tương ứng với công đoạn *Lễ khởi công*, nó phải được thực hiện trước tất cả các công đoạn khác, và đỉnh $n+1$ tương ứng với công đoạn *Cắt băng khánh thành công trình*, nó phải thực hiện sau tất cả các công đoạn, với $t[0] = t[n+1] = 0$ (trên thực tế chỉ cần nối đỉnh 0 với tất cả các đỉnh có bán bậc vào bằng 0 và nối tất cả các đỉnh có bán bậc ra bằng 0 với đỉnh $n+1$). Gọi đồ thị thu được là G. Rõ ràng bài toán đặt ra dẫn về bài toán tìm đường đi dài nhất từ đỉnh 0 đến tất cả các đỉnh còn lại trên đồ thị G. Do đồ thị G rõ ràng là không chứa chu trình, nên để giải bài toán đặt ra có thể áp dụng các thuật toán mô tả ở trên, chỉ cần đổi dấu tất cả các trọng số trên các cung thành dấu ngược lại, hoặc đơn giản hơn chỉ cần đổi toán tử ***min*** trong thuật toán Critical_Path thành toán tử ***max***. Kết thúc thuật toán, chúng ta thu được $d[v]$ là độ dài đường đi dài nhất từ đỉnh 0 đến đỉnh v. Khi đó $d[v]$ cho ta thời điểm sớm nhất có thể bắt đầu thực hiện công đoạn v, nói riêng $d[n+1]$ là thời điểm sớm nhất có thể cắt băng khánh thành, tức là thời điểm sớm nhất có thể hoàn thành toàn bộ công trình.

Cây đường đi dài nhất của bài toán trong thí dụ 4 tìm được theo thuật toán được chỉ ra trong hình 4.

6.5. Đường đi ngắn nhất giữa tất cả các cặp đỉnh

Rõ ràng ta có thể giải bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị bằng cách sử dụng n lần thuật toán mô tả ở mục trước, trong đó ta sẽ chọn s lần lượt là các đỉnh của đồ thị. Rõ ràng, khi đó ta thu được thuật toán với độ phức tạp là $O(n^4)$ (nếu sử dụng thuật toán Ford - Bellman) hoặc $O(n^3)$ đối với trường hợp trọng số không âm hoặc đồ thị không có chu trình. Trong trường hợp tổng quát, sử dụng thuật toán Ford - Bellman n lần không phải là cách làm tốt nhất. ở đây ta sẽ mô tả một thuật toán giải bài toán trên với độ phức tạp tính toán $O(n^3)$: Thuật toán Floyd. Thuật toán được mô tả trong thủ tục dưới đây.

procedure Floyd;

(* ***Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh***

- **Đầu vào:** *Đồ thị cho bởi ma trận trọng số* $a[i,j]$, $i, j = 1,2,...,n$.
- **Đầu ra :**

Ma trận đường đi ngắn nhất giữa các cặp đỉnh

$d[i,j]$, $i, j = 1,2,...,n$,

trong đó $d[i,j]$ *cho độ dài đường đi ngắn nhất từ* i *đến* j.

Ma trận ghi nhận đường đi

$p[i,j]$, $i, j = 1,2,...,n$,

trong đó $p[i,j]$ *ghi nhận đỉnh đi trước đỉnh* j *trong đường đi ngắn nhất từ* i *đến* j.

*)

```
begin
      (*  Khởi tạo  *)
      for i := 1  to  n  do
          for j := 1  to  n  do
          begin
              d[i,j] := a[i,j];
              p[i,j] := i;
          end;
      (*  Bước lặp  *)
      for  k:= 1  to  n  do
         for i := 1  to  n  do
             for j := 1  to  n  do
                 if  d[i,j] > d[i,k] + d[k,j]  then
                 begin
                     d[i,j] := d[i,k] + d[k,j] ;
                     p[i,j] := p[k,j];
                 end;
end;
```

Rõ ràng độ phức tạp tính toán của thuật toán là $O(n^3)$.

Kết thúc chương này chúng ta trình bày một cách thể hiện thuật toán Dijkstra trên ngôn ngữ PASCAL:

```
(* Chương trình tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t theo thuật toán Dijkstra *)
uses crt;
const
  max = 50;
var
   n, s, t : integer;
   chon : char;
   Truoc : array [1..max] of  byte;
   d      : array [1..max] of  integer;
   a      : array [1..max,1..max] of  integer;
   final  : array [1..max] of  boolean;

procedure NhapSoLieu;
var
   f      : text;
   fname : string;
```

```
  i,j     : integer;
begin
  write ('Vao ten file du lieu can doc: ');
  readln(fname);
  assign(f,fname);
  reset(f);
  readln(f,n);
  for i:= 1 to n do
    for j:= 1 to n do read(f,a[i,j]);
  close(f);
end;

procedure InSoLieu;
var
  i, j : integer;
begin
  writeln(' So dinh cua do thi : ',n);
  writeln(' Ma tran khoang cach:');
  for i := 1 to n do
  begin
    for j:= 1 to n do write(a[i,j]:3,' ');
    writeln;
  end;
end;

procedure InKetQua;
var
  i, j : integer;
begin
  writeln (' DUONG DI NGAN NHAT TU ',s,' DEN ',t);
  write(t,' <= ');
  i:=Truoc[t];
  while i<>s do
  begin
    write(i,' <= ');
    i:=Truoc[i];
  end;
  writeln(s);
  writeln('Do dai cua duong di la: ',d[t]);
end;
```

```
procedure Dijkstra;
var  v, u, minp : integer;
begin
   write('Tim duong di tu s = ');
   readln(s);
   write('          den t = ');
   readln(t);
   for v:=1 to n do
   begin   (* Khởi tạo nhãn *)
       d[v]:=a[s,v];
       Truoc[v]:=s;
       final[v]:=false;
   end;
   Truoc[s]:=0;
   d[s]:=0;
   final[s]:=true;
   while not final[t] do   (*****  Bước lặp  *****)
   begin
      {  Tìm u là đỉnh có nhãn tạm thời nhỏ nhất  }
      minp:= maxint;
      for v:=1 to n do
         if (not final[v]) and (minp > d[v]) then
         begin
            u:= v;
            minp:= d[v];
         end;
      final[u]:= true;
      if  not final[t] then
          for v:=1 to n do
              if  ( not final[v] ) and ( d[u] + a[u,v] < d[v] ) then
                 begin
                    d[v]:= d[u] + a[u,v];
                    Truoc[v]:= u;
                 end;
              end;
   end;
end;
```

```
procedure Menu;
begin
  clrscr;
  writeln('  CHUONG TRINH TIM  DUONG DI NGAN NHAT TU s DEN t  ');
  writeln('          THEO THUAT TOAN DIJKSTRA');
  writeln('        ******************************');
  writeln('          1. Nhap so lieu tu file.');
  writeln('          2. Giai bai toan.');
  writeln('          3. Ket thuc.');
  writeln('-------------------------------------------------------');
  write('       Hay chon chuc nang: '#7);
end;

BEGIN  (*  Chương trình chính  *)
  repeat
     Menu;  chon:=readkey;  writeln(chon);
     case chon of
       '1': NhapSoLieu;
       '2': begin
              InSoLieu;  Dijkstra;  InKetQua;
            end;
       '3': Exit;
     end;
  until false;
END.
```

Bài tập

1. Lập trình thực hiện các thuật toán mô tả trong chương 6:

a) Thuật toán Ford - Bellman;

b) Thuật toán Dijkstra;

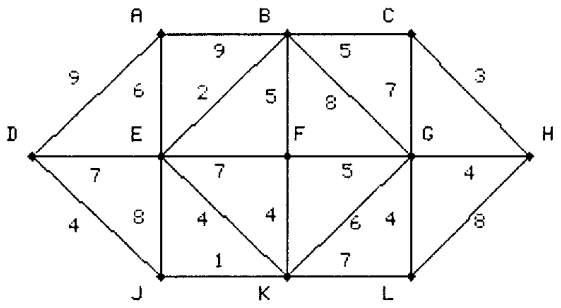
c) Thuật toán Floyd;

d) Thuật toán tìm đường đi dài nhất trên đồ thị không có chu trình (PERT).

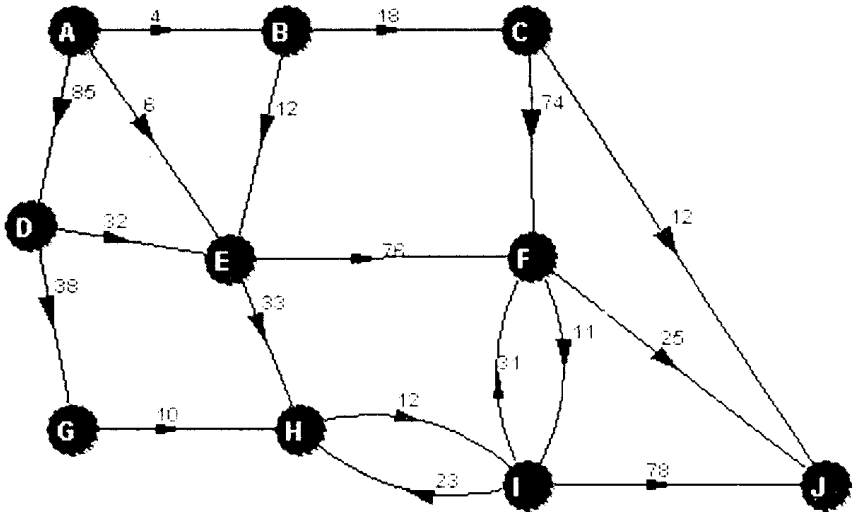
2. Trong trường hợp đồ thị thưa, các thuật toán Ford - Bellman, Dijkstra, Floyd sẽ làm việc hiệu quả hơn nếu ta sử dụng danh sách kề để biểu diễn đồ thị. Hãy mô tả lại các thuật toán trên cho trường hợp này, và đánh giá độ phức tạp tính toán của chúng.

3. Tìm đường đi ngắn nhất theo thuật toán Dijkstra từ đỉnh A đến tất cả các đỉnh còn lại trên đồ thị cho trong các hình vẽ dưới đây:

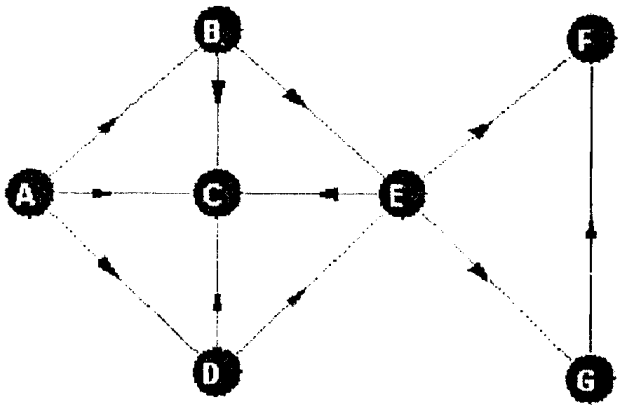
a)



b)



4. Áp dụng thuật toán đánh số đỉnh để kiểm tra đồ thị sau đây có chứa chu trình hay không



5. Xét đồ thị gồm 7 đỉnh A, B, C, D, E, F, G cho bởi ma trận trọng số

	A	B	C	D	E	F	G
A	0	11	65	17	65	65	65
B	65	0	12	65	65	10	16
C	65	65	0	13	14	65	19
D	65	65	65	0	65	65	18
E	65	65	65	65	0	65	15
F	65	13	18	65	65	0	10
G	65	65	65	65	65	65	0

Tìm đường đi ngắn nhất từ đỉnh A đến đỉnh G theo thuật toán Dijkstra. Yêu cầu viết rõ kết quả trung gian trong từng bước. Kết quả cuối cùng: Đường đi ngắn nhất từ A đến G và độ dài của nó.

6. Cho một bảng gồm $m \times n$ ô. Nếu một ô nằm ở hàng i cột j của bảng thì ta sẽ nói ô này có toạ độ (i,j) (i=1,2,...,m; j=1,2,...,n). Trong mỗi ô viết một số nguyên (gọi là giá trị của ô). Từ một ô bất kỳ của bảng chỉ có thể di chuyển đến những ô ở bên phải có chung cạng hoặc đỉnh với nó. Dữ liệu về bảng nói trên được cho trong một file văn bản có tên DATA3.INP, dòng đầu tiên của nó chứa hai số m, n được ghi cách nhau bởi dấu cách. Mỗi dòng thứ i trong số m dòng tiếp theo chứa n số ghi trong n ô của dòng thứ i của bảng, các số được ghi cách nhau bởi dấu cách (i=1,2,...,m). Hãy lập trình nhập dữ liệu vào từ file, sau đó tìm dãy các di chuyển từ mép trái của bảng sang mép phải của bảng sao cho tổng số các giá trị của các ô đi qua là nhỏ nhất. Yêu cầu kết quả đưa ra dưới dạng dãy toạ độ của các ô cần di chuyển qua bắt đầu từ một ô xuất phát ở mép trái và kết thúc ở một ô nào đó ở mép phải, và tổng số các giá trị của các ô này.

7. Cho dãy n số nguyên $x[1], x[2],..., x[n]$, trong đó $1<n<1000$ và $1<x[i]<200$, $i = 1, 2,...,n$. Dãy số đã cho được ghi trong một file văn bản có tên là DATA4.INP, các số được ghi cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng. Nhập dữ liệu vào từ file, sau đó tìm cách xoá bỏ ít nhất một số các phần tử của dãy đã cho sao cho các phần tử còn lại (giữ nguyên thứ tự của chúng trong dãy ban đầu) tạo thành một dãy con không giảm. Đưa ra màn hình tổng số các phần tử cần xoá và chỉ số của chúng trong dãy đã cho.

8. Dãy số nguyên a[1], a[2], ... , a[m] được gọi là dãy con của dãy con của dãy số nguyên b[1], b[2],..., b[n] (m ≤ n) nếu tìm được dãy các chỉ số j[1], j[2],...,j[m] sao cho

$$1 \le j[1] < j[2] < ... < j[m] \le n$$

và

$$a[i] = b[j[i]], i=1,2,....m. \quad (1)$$

Cho hai dãy số nguyên

$$c[1], c[2], ... , c[p], \quad (2)$$

$$d[1], d[2], ... , d[q]. \quad (3)$$

Dữ liệu về hai dãy số được ghi trong file văn bản có tên là DATA5.INP, dòng đầu tiên chứa hai số p, q được ghi cách nhau bởi dấu cách. Trong cách dòng tiếp theo ghi lần lượt các phần tử của dãy (2) rồi tiếp đến các phần tử của dãy (3), các số ghi cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng. Hãy lập trình nhập dữ liệu vào từ file, sau đó tìm dãy con chung có số phần tử là lớn nhất của hai dãy đã cho. Kết quả đưa ra file có tên KQUA5.OUT: dòng đầu tiên chứa số phần của dãy con chung tìm được, trong các dòng tiếp theo lần lượt ghi các phần tử của hai dãy chỉ số tương ứng chỉ rõ vị trí của dãy con này trong hai dãy ban đầu (các số ghi cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng.

9. Cho bàn cờ quốc tế gồm $n.n$ ô ($n < 50$). Trong mỗi ô (i,j) của nó ta điền một số nguyên dương $a[i,j] <100$, $i, j = 1,2,...,n$. Từ một ô bất kỳ (i,j) của bảng ta có thể di chuyển sang ô có cùng mầu với nó nếu như $a[i,j]$ là số chẵn, và sang ô khác màu với nó nếu $a[i,j]$ là số lẻ. Dữ liệu được cho trong một file văn bản có tên là DATA6.INP, dòng đầu tiên chứa số n. Trong các dòng tiếp theo chứa các phần tử $a[1,1]$, $a[1,2]$, ..., $a[1,n]$, ..., $a[n,1]$, $a[n,2]$, ..., $a[n,n]$, các số được lần lượt ghi theo thứ tự vừa nêu và cách nhau bởi ít nhất một dấu cách hoặc dấu xuống dòng.

Yêu cầu: Tìm một cách di chuyển một ô nào đó ở cột thứ nhất sang một ô nào đó ở cột thứ n của bàn cờ sao cho tổng các số ghi trong các ô di chuyển qua là nhỏ nhất. Đưa ra màn hình kết quả dưới dạng ba số nguyên x, y, L, trong đó x là toạ độ dòng của ô xuất phát, y - toạ độ dòng của ô kết thúc, L - tổng các số ghi trong các ô cần di chuyển qua.

7

BÀI TOÁN LUỒNG CỰC ĐẠI TRONG MẠNG

Bài toán luồng cực đại trong mạng là một trong số những bài toán tối ưu trên đồ thị tìm được những ứng dụng rộng rãi trong thực tế cũng như những ứng dụng thú vị trong lý thuyết tổ hợp. Bài toán được đề xuất vào đầu những năm 1950, và gắn liền với tên tuổi của hai nhà toán học Mỹ là Ford và Fulkerson. Trong chương này chúng ta sẽ trình bày thuật toán của Ford và Fulkerson để giải bài toán đặt ra và nêu một số ứng dụng của bài toán.

7.1. Mạng. Luồng trong mạng. Bài toán luồng cực đại

Định nghĩa 1. *Ta gọi mạng là đồ thị có hướng $G = (V, E)$, trong đó có duy nhất một đỉnh s không có cung đi vào gọi là điểm phát, duy nhất một đỉnh t không có cung đi ra gọi là điểm thu và mỗi cung $e = (v,w) \in E$ được gán với một số không âm $c(e) = c(v,w)$ gọi là khả năng thông qua của cung e.*

Để thuận tiện cho việc trình bày ta sẽ qui ước rằng nếu không có cung (v,w) thì khả năng thông qua $c(v,w)$ được gán bằng 0.

Định nghĩa 2. *Giả sử cho mạng $G=(V, E)$. Ta gọi luồng f trong mạng $G = (V, E)$ là ánh xạ $f: E \to R_+$ gán cho mỗi cung $e = (v,w) \in E$ một số thực không âm $f(e) = f(v,w)$, gọi là luồng trên cung e, thoả mãn các điều kiện sau:*

1) *Luồng trên mỗi cung $e \in E$ không vượt quá khả năng thông qua của nó:*

$$0 \le f(e) \le c(e),$$

2) *Điều kiện cân bằng luồng trên mỗi đỉnh của mạng: Tổng luồng trên các cung đi vào đỉnh v bằng tổng luồng trên các cung đi ra khỏi đỉnh v, nếu $v \ne s, t$:*

$$Div_f(v) = \sum_{w \in \Gamma^-(v)} f(w,v) - \sum_{w \in \Gamma^+(v)} f(v,w).$$

trong đó $\Gamma^-(v)$ - tập các đỉnh của mạng mà từ đó có cung đến v, $\Gamma^+(v)$ - tập các đỉnh của mạng mà từ v có cung đến nó:

$$\Gamma^-(v) = \{w \in V: (w, v) \in E\}, \quad \Gamma^+(v) = \{w \in V: (v, w) \in E\}.$$

3) *Ta gọi giá trị của luồng f là số*

$$val(f) = \sum_{w \in \Gamma^+(s)} f(s,w) = \sum_{w \in \Gamma^-(t)} f(w,t).$$

Chú ý là đẳng thức thứ hai trong định nghĩa giá trị luồng là hệ quả của điều kiện cân bằng luồng.

Bài toán luồng cực đại trong mạng: Cho mạng $G= (V, E)$. Hãy tìm luồng f^* trong mạng với giá trị luồng $val(f^*)$ là lớn nhất. Luồng như vậy ta sẽ gọi là luồng cực đại trong mạng.

Bài toán như vậy có thể xuất hiện trong rất nhiều ứng dụng thực tế. Chẳng hạn khi cần xác định cường độ lớn nhất của dòng vận tải giữa 2 nút của một bản đồ giao thông. Trong ví dụ này lời giải của bài toán luồng cực đại sẽ chỉ cho ta các đoạn đường đông xe nhất và chúng tạo thành "chỗ hẹp" tương ứng với dòng giao thông xét theo 2 nút được chọn. Một ví dụ khác là nếu xét đồ thị tương ứng với một hệ thống đường ống dẫn dầu. Trong đó các ống tương ứng với các cung, điểm phát có thể coi là tàu chở dầu, điểm thu là bể chứa, còn những điểm nối giữa các ống là các nút của đồ thị. Khả năng thông qua của các cung tương ứng với tiết diện các ống. Cần phải tìm luồng dầu lớn nhất có thể bơm từ tàu chở dầu vào bể chứa.

7.2. Lát cắt. Đường tăng luồng. Định lý Ford - Fulkerson

Định nghĩa 3. *Ta gọi lát cắt (X, X^*) là một cách phân hoạch tập đỉnh V của mạng ra thành hai tập X và $X^* = V \setminus X$, trong đó $s \in X$ và $t \in X^*$. Khả năng thông qua của lát cắt (X, X^*) là số*

$$c(X, X^*) = \sum_{\substack{v \in X \\ w \in X^*}} c(v, w).$$

Lát cắt với khả năng thông qua nhỏ nhất được gọi là lát ***cắt hẹp nhất.***

Bổ đề 1. *Giá trị của mọi luồng f trong mạng luôn nhỏ hơn hoặc bằng khả năng thông qua của lát cắt (X, X^*) bất kỳ trong nó: $val(f) \le c(X, X^*)$.*

Chứng minh. Cộng các điều kiện cân bằng luồng $Div_f(v) = 0$ với mọi $v \in X$. Khi đó ta có

$$\sum_{v \in X} \Big(\sum_{w \in \Gamma^-(v)} f(w, v) - \sum_{w \in \Gamma^+(v)} f(v, w) \Big) = -val(f)$$

tổng này sẽ gồm các số hạng dạng $f(u,v)$ với dấu cộng hoặc dấu trừ mà trong đó có ít nhất một trong hai đỉnh u, v phải thuộc tập X. Nếu cả hai đỉnh u, v đều trong tập X, thì $f(u,v)$ xuất hiện với dấu cộng trong $Div_f(v)$ và với dấu trừ trong $Div_f(u)$, vì thế, chúng triệt tiêu lẫn nhau. Do đó, sau khi giản ước các số hạng như vậy ở vế trái, ta thu được

$$-\sum_{\substack{v \in X \\ w \in X^*}} f(v, w) + \sum_{\substack{v \in X^* \\ w \in X}} f(v, w) = -val(f),$$

hay là

$$val(f) = \sum_{\substack{v \in X \\ w \in X^*}} f(v, w) - \sum_{\substack{v \in X^* \\ w \in X}} f(v, w).$$

Mặt khác, từ điều kiện 1 rõ ràng là

$$\sum_{\substack{v \in X \\ w \in X^*}} f(v, w) \le \sum_{\substack{v \in X \\ w \in X^*}} c(v, w),$$

còn

$$-\sum_{\substack{v \in X^* \\ w \in X}} f(v, w) \le 0.$$

suy ra $val(f) \le c(X, X^*)$. Bổ đề được chứng minh.

Từ bổ đề 1 suy ra

Hệ quả 1. *Giá trị luồng cực đại trong mạng không vượt quá khả năng thông qua của lát cắt hẹp nhất trong mạng.*

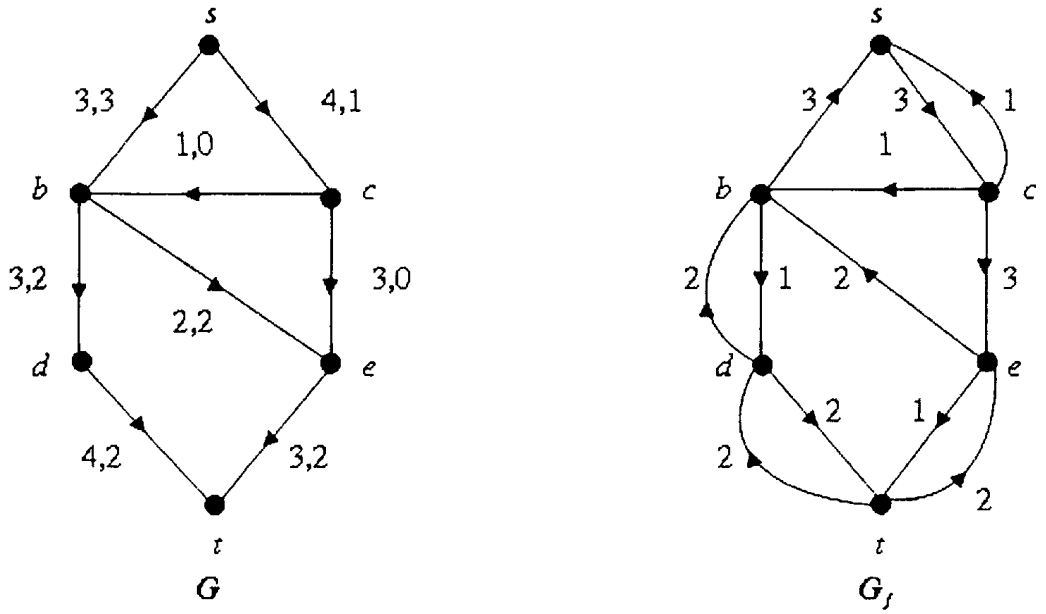
Ford và Fulkerson đã chứng minh rằng giá trị luồng cực đại trong mạng đúng bằng khả năng thông qua của lát cắt hẹp nhất. Để có thể phát biểu và chứng minh kết quả này chúng ta sẽ cần thêm một số khái niệm.

Giả sử f là một luồng trong mạng $G = (V, E)$. Từ mạng $G = (V, E)$ ta xây dựng đồ thị có trọng số trên cung $G_f = (V, E_f)$, với tập cung E_f và trọng số trên các cung được xác định theo quy tắc sau:

1) Nếu $e = (v,w) \in E$ với $f(v,w) = 0$, thì $(v,w) \in E_f$ với trọng số $c(v,w)$;
2) Nếu $e = (v,w) \in E$ với $f(v,w) = c(v,w)$, thì $(w,v) \in E_f$ với trọng số $f(v,w)$;
3) Nếu $e = (v,w) \in E$ với $0 < f(v,w) < c(v,w)$, thì $(v,w) \in E_f$ với trọng số $c(v,w) - f(v,w)$ và $(w,v) \in E_f$ với trọng số $f(v,w)$.

Các cung của G_f đồng thời cũng là cung của G được gọi là ***cung thuận***, các cung còn lại được gọi là ***cung nghịch.*** Đồ thị G_f được gọi là ***đồ thị tăng luồng***.

Thí dụ: Các số viết cạnh các cung của G ở hình 1 theo thứ tự là khả năng thông qua và luồng trên cung.



Hình 1. Mạng G và luồng f. Đồ thị có trọng số G_f tương ứng.

Giả sử $P = (s = v_0, v_1, v_2, \ldots, v_k = t)$ là một đường đi từ s đến t trên đồ thị tăng luồng G_f. Gọi δ là giá trị nhỏ nhất của các trọng số của các cung trên đường đi P. Xây dựng luồng f' trên mạng G theo quy tắc sau:

$$f'(u,v) = \begin{cases} f(u,v) + \delta, & \text{nếu } (u,v) \in P \text{ là cung thuận,} \\ f(u,v) - \delta, & \text{nếu } (u,v) \in P \text{ là cung nghịch,} \\ f(u,v), & \text{nếu } (u,v) \notin P. \end{cases}$$

Dễ dàng kiểm tra được rằng f' được xây dựng như trên là luồng trong mạng và $val(f') = val(f) + \delta$. Ta sẽ gọi thủ tục biến đổi luồng vừa nêu là ***tăng luồng dọc theo đường P.***

Định nghĩa 4. *Ta gọi đường tăng luồng f là mọi đường đi từ s đến t trên đồ thị tăng luồng $G(f)$.*

Định lý dưới đây cho mối liên hệ giữa luồng cực đại, đường tăng luồng và lát cắt.

Định lý 1. *Các mệnh đề dưới đây là tương đương:*

(i) f là luồng cực đại trong mạng;

(ii) Không tìm được đường tăng luồng f;

(iii) $val(f) = c(X,X^*)$ với một lát cắt (X, X^*) nào đó.

Chứng minh.

$(i) \Rightarrow (ii)$. Giả sử ngược lại, tìm được đường tăng luồng P. Khi đó ta có thể tăng giá trị luồng bằng cách tăng luồng dọc theo đường P. Điều đó mâu thuẫn với tính cực đại của luồng f.

$(ii) \Rightarrow (iii)$. Giả sử không tìm được đường tăng luồng. Ký hiệu X là tập tất cả các đỉnh có thể đến được từ đỉnh s trong đồ thị G_f, và đặt $X^* = V \setminus X$. Khi đó (X, X^*) là lát cắt, và $f(v,w) = 0$ với mọi $v \in X^*$, $w \in X$ nên

$$val(f) = \sum_{\substack{v \in X \\ w \in X^*}} f(v,w) - \sum_{\substack{v \in X^* \\ w \in X}} f(v,w) = \sum_{\substack{v \in X \\ w \in X^*}} f(v,w)$$

Với $v \in X$, $w \in X^*$, do $(v, w) \notin G_f$, nên $f(v, w) = c(v, w)$. Vậy

$$val(f) = \sum_{\substack{v \in X \\ w \in X^*}} f(v,w) = \sum_{\substack{v \in X \\ w \in X^*}} c(v,w) = c(X, X^*).$$

$(iii) \Rightarrow (i)$. Theo Bổ đề 1, $val(f) \le c(X, X^*)$ với mọi luồng f và với mọi lát cắt (X, X^*). Vì vậy, từ đẳng thức $val(f) = c(X, X^*)$ suy ra luồng f là luồng cực đại trong mạng.

Định lý được chứng minh.

7.3. Thuật toán tìm luồng cực đại trong mạng

Định lý 1 là cơ sở để xây dựng thuật toán lặp sau đây để tìm luồng cực đại trong mạng: Bắt đầu từ luồng với luồng trên tất cả các cung bằng 0 (ta sẽ gọi luồng như vậy là luồng không), và lặp lại bước lặp sau đây cho đến khi thu được luồng mà đối với nó không còn đường tăng:

Bước lặp tăng luồng (Ford - Fulkerson): Tìm đường tăng P đối với luồng hiện có. Tăng luồng dọc theo đường P.

Khi đã có luồng cực đại, lát cắt hẹp nhất có thể tìm theo thủ tục mô tả trong chứng minh định lý 1. Sơ đồ của thuật toán Ford - Fulkerson có thể mô tả trong thủ tục sau đây:

```
procedure Max_Flow;
(* Thuật toán Ford - Fulkerson *)
begin
    (* Khởi tạo: Bắt đầu từ luồng với giá trị 0 *)
    for u ∈ V do
        for v ∈ V do f(u,v):=0;
    Stop:=false;
    while not Stop do
        if <Tìm được đường tăng luồng P> then <Tăng luồng dọc theo P>
        else Stop:=true;
end;
```

Để tìm đường tăng luồng trong G_f có thể sử dụng thuật toán tìm kiếm theo chiều rộng (hay tìm kiếm theo chiều sâu) bắt đầu từ đỉnh s, trong đó không cần xây dựng tường minh đồ thị G_f. Ford - Fulkerson đề nghị thuật toán gán nhãn chi tiết sau đây để giải bài toán luồng cực đại trong mạng. Thuật toán bắt đầu từ luồng chấp nhận được nào đó trong mạng (có thể bắt đầu từ luồng không), sau đó ta sẽ tăng luồng bằng cách tìm các đường tăng luồng. Để tìm đường tăng luồng ta sẽ áp dụng phương pháp gán nhãn cho các đỉnh. Mỗi đỉnh trong quá trình thực hiện thuật toán sẽ ở một trong 3 trạng thái: chưa có nhãn, có nhãn chưa xét, có nhãn đã xét. Nhãn của một đỉnh v gồm 2 phần và có một trong 2 dạng sau: $[+p(v),\varepsilon(v)]$ hoặc $[-p(v),\varepsilon(v)]$. Phần thứ nhất $+p(v)$ (hoặc $-p(v)$) chỉ ra là cần tăng (hoặc giảm) luồng theo cung $(p(v), v)$ (cung $(v, p(v))$ còn phần thứ hai $\varepsilon(v)$ chỉ ra lượng lớn nhất có thể tăng hoặc giảm luồng theo cung này. Đầu tiên chỉ có đỉnh s được khởi tạo nhãn và nhãn của nó là chưa xét, còn tất cả các đỉnh còn lại đều chưa có nhãn. Từ s ta gán nhãn cho tất cả các đỉnh kề với nó và nhãn của đỉnh s sẽ trở thành đã xét. Tiếp theo, từ mỗi đỉnh v có nhãn chưa xét ta lại gán nhãn cho tất cả các

đỉnh chưa có nhãn kề với nó và nhãn của đỉnh v trở thành đã xét. Quá trình sẽ được lặp lại cho đến khi hoặc là đỉnh t trở thành có nhãn hoặc là nhãn của tất cả các đỉnh có nhãn đều là đã xét nhưng đỉnh t vẫn không có nhãn. Trong trường hợp thứ nhất ta tìm được đường tăng luồng, còn trong trường hợp thứ hai đối với luồng đang xét không tồn tại đường tăng luồng (tức là luồng đã là cực đại). Mỗi khi tìm được đường tăng luồng, ta lại tăng luồng theo đường tìm được, sau đó xoá tất cả các nhãn và đối với luồng mới thu được lại sử dụng phép gán nhãn các đỉnh để tìm đường tăng luồng. Thuật toán sẽ kết thúc khi nào đối với luồng đang có trong mạng không tìm được đường tăng luồng.

Hai thủ tục Tìm đường tăng luồng và Tăng luồng có thể mô tả như sau.

```
procedure Find_Path;
(* Thủ tục gán nhãn tìm đường tăng luồng
    p[v], ε[v] là nhãn của đỉnh v;
    V_T - danh sách các đỉnh có nhãn nhưng chưa xét;
    c[u,v] - khả năng thông qua của cung (u, v), u, v ∈ V;
    f[u,v] - luồng trên cung (u, v), (u, v ∈ V)    *)
begin
    p[s] := s;
    ε[s]:= +∞;
    V_T = V\{s};
    PathFound:=true;
    while V_T ≠ ∅ do
    begin
        u ⇐ V_T ; (* Lấy u từ V_T *)
        for v ∈ V\V_T do
        begin
            if (c[u, v] > 0) and (f[u, v] < c[u,v]) then
            begin
                p[v]:= u;
                ε[v]:= min { ε[u], c[u,v] - f[u,v];
                V_T = V_T ∪ {v}; (* Nạp v vào danh sách đỉnh có nhãn *)
                if v = t then exit;
            end;
            if (c[v,u] > 0) and (f[v, u] > 0) then
            begin
                p[v]:= -u;
                ε[v]:= min { ε[u], f[v,u]};
                V_T = V_T ∪ {v}; (* Nạp v vào danh sách đỉnh có nhãn *)
```

```
                if v = t then exit;
            end;
    end;
    PathFound:=false;
end;
```

procedure Inc_Flow;

```
(* Tăng luồng theo đường tăng *)
begin
    v := p[t]; u:=t; tang:= ε[t];
    while u ≠ s do
    begin
        if v > 0 then f[v, u] := f[v, u] + tang
        esle
        begin
            v:= -v;
            f[u,v]:= f[u,v] - tang;
        end;
        u:=v; v:=p[u];
    end;
end;
```

Thuật toán Ford - Fulkerson được thực hiện nhờ thủ tục:

procedure Max_Flow;

```
(* Thuật toán Ford - Fulkerson *)
begin
    (* Khởi tạo: Bắt đầu từ luồng với giá trị 0 *)
    for u ∈ V do
        for v ∈ V do f[u,v]:=0;
    Stop:=false;
    while not Stop do
    begin
        Find_Path;
        if PathFound then Inc_Flow
        else Stop:=true;
    end;
    < Luồng cực đại trong mạng là f[u,v], u, v ∈V >
    < Lát cắt hẹp nhất là (V_T, V\V_T) >
end;
```

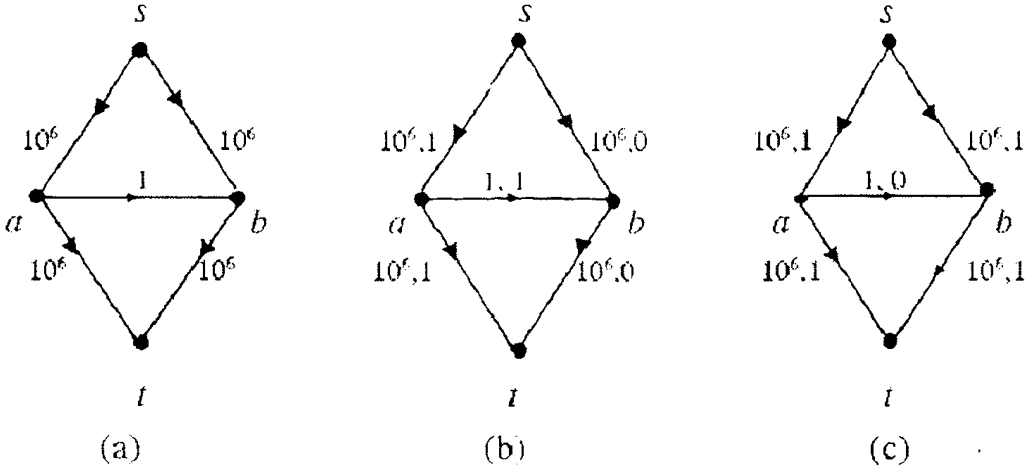
Giả sử là khả năng thông qua của tất cả các cung của đồ thị là các số nguyên. Khi đó sau mỗi lần tăng luồng, giá trị luồng sẽ tăng lên ít nhất là 1. Từ đó suy ra thuật toán Ford - Fulkerson sẽ dừng sau không quá $val(f^*)$ lần tăng luồng và cho ta luồng cực đại trong mạng. Đồng thời, rõ ràng $f^*(u,v)$ sẽ là số nguyên đối với mỗi cung $(u, v) \in E$. Từ đó ta có các kết quả sau:

Định lý 2 ***(Định lý về luồng cực đại trong mạng và lát cắt hẹp nhất).*** *Luồng cực đại trong mạng bằng khả năng thông qua của lát cắt hẹp nhất.*

Định lý 3. (Định lý về tính nguyên). *Nếu tất cả các khả năng thông qua là các số nguyên thì luôn tìm được luồng cực đại với luồng trên các cung là các số nguyên.*

Hai định lý vừa nêu được sử dụng trong việc ứng dụng luồng cực đại vào việc giải quyết nhiều bài toán trong tổ hợp.

Tuy nhiên, nếu các khả năng thông qua là các số rất lớn thì giá trị của luồng cực đại cũng có thể là rất lớn và khi đó thuật toán mô tả ở trên sẽ đòi hỏi thực hiện rất nhiều bước tăng luồng. Thí dụ trong hình 2 sẽ minh hoạ cho điều này. Hình 2(a) mô tả mạng cần xét với các khả năng thông qua trên các cung. Hình 2(b) mô tả luồng trên các cung (số thứ hai bên cạnh cung) sau khi thực hiện tăng luồng dọc theo đương tăng luồng (s, a, b, t). Hình 2(c) mô tả luồng trên các cung sau khi thực hiện tăng luồng dọc theo đường tăng luồng (s, b, a, t). Rõ ràng, sau 2.10^6 lần tăng luồng theo đường (s, a, b, t) và (s, b, a, t) một cách luân phiên ta thu được luồng cực đại.



Hình 2. Ví dụ tồi tệ đối với thuật toán Ford - Fulkerson.

Hơn thế nữa, nếu các khả năng thông qua là các số vô tỷ, người ta còn xây dựng được ví dụ để cho thuật toán không dừng, và tệ hơn là nếu dãy các giá trị luồng xây dựng theo thuật toán hội tụ thì nó còn không hội tụ đến giá trị luồng cực đại. Như vậy,

muốn thuật toán làm việc hiệu quả, việc lựa chọn đường tăng luồng cần được tiến hành hết sức cẩn thận.

Edmonds và Karp chỉ ra rằng nếu đường tăng luồng được chọn là đường ngắn nhất từ s đến t trên đồ thị G_f. Điều đó có thể thực hiện, nếu trong thủ tục tìm đường tăng Find_Path mô tả ở trên, danh sách V_f được tổ chức dưới dạng QUEUE (nghĩa là ta thực hiện tìm đường tăng bởi thủ tục tìm kiếm theo chiều rộng) thì thuật toán sẽ kết thúc sau không quá $nm/2$ lần sử dụng đường tăng luồng. Nếu để ý rằng, tìm kiếm theo chiều rộng trên đồ thị đòi hỏi thời gian $O(n+m)$, thì thuật toán thu được sẽ có độ phức tạp tính toán là $O(nm^2)$.

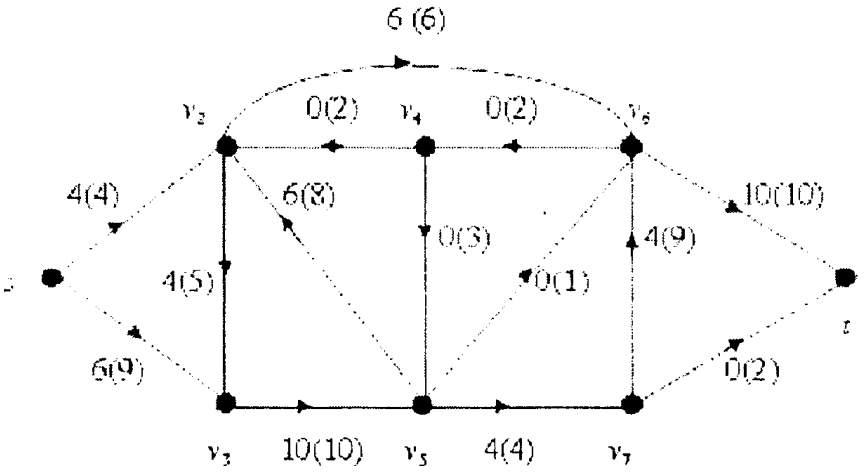
Nhờ cách tổ chức tìm đường tăng khéo léo hơn, người ta đã xây dựng được thuật toán với độ phức tạp tính toán tốt hơn như: $O(n^2m)$ (Dinic, 1970), $O(n^3)$ (Karzanov, 1974), $O(n^2m^{1/2})$ (Cherkasky, 1977), $O(nm \log n)$ (Sleator -Tarjan, 1980).

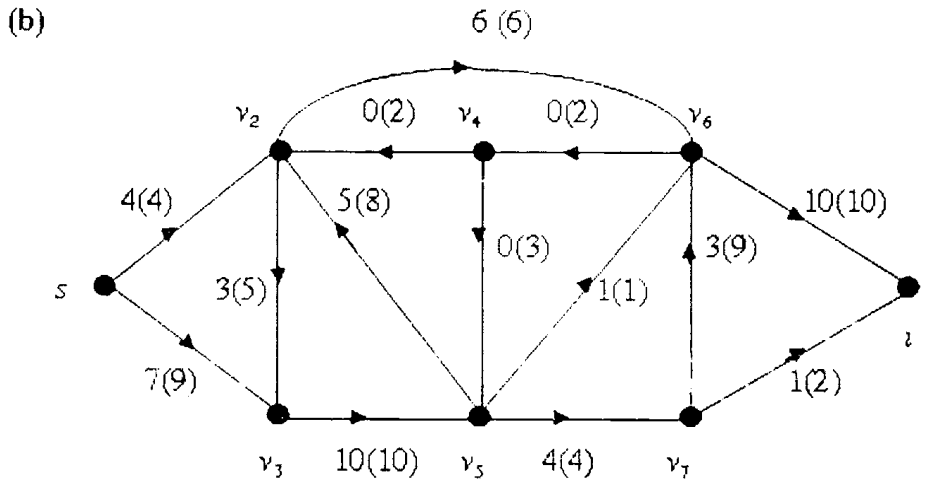
Ta kết thúc mục này bởi ví dụ minh hoạ cho thuật toán Ford - Fulkerson sau đây. Hình 3 (a) cho mạng G cùng với khả năng thông qua của các cung và luồng giá trị 10 trong nó. Hai số viết bên cạnh mỗi cung là khả năng thông qua của cung (số trong ngoặc) và luồng trên cung. Đường tăng luồng có dạng (s, v_3, v_2, v_5, v_6, v_7, t). Ta tính được $\varepsilon(t) = 1$, giá trị luồng tăng từ 10 lên 11. Hình 3 (b) cho luồng thu được sau khi tăng luồng theo đường tăng tìm được.

Luồng trong hình 3 (b) đã là cực đại. Lát cắt hẹp nhất

$$X = \{s, v_2, v_3, v_5\}, \bar{X} = \{v_4, v_6, v_7, t\}.$$

Giá trị luồng cực đại là 11.





Hình 3. Tăng luồng dọc theo đường tăng

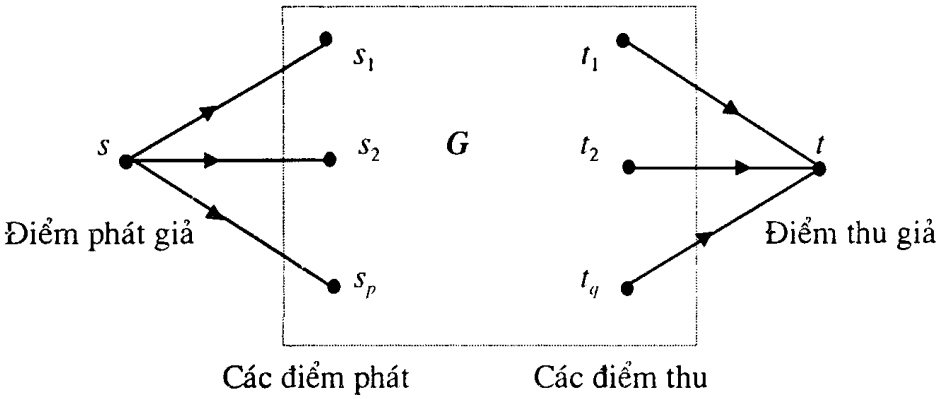
7.4. Một số bài toán luồng tổng quát

Trong phần này ta nêu ra một số dạng bài toán về luồng tổng quát mà việc giải chúng có thể dẫn về bài toán luồng cực đại trình bày ở trên.

a) Mạng với nhiều điểm phát và điểm thu.

Xét mạng G với p điểm phát $s_1, s_2, ..., s_p$ và q điểm thu $t_1, t_2, ..., t_q$. Giả sử rằng luồng có thể đi từ một điểm phát bất kỳ đến tất cả các điểm thu. Bài toán tìm luồng cực đại từ các điểm phát đến các điểm thu có thể đưa về bài toán với một điểm phát và một điểm thu bằng cách đưa vào một điểm phát giả s và một điểm thu giả t và các cạnh nối s với tất cả các điểm phát và các cạnh nối các điểm thu với t.

Hình 4 minh hoạ cho cách đưa mạng với nhiều điểm phát và nhiều điểm thu về mạng chỉ có một điểm phát và một điểm thu. Khả năng thông qua của cung nối s với điểm phát s_k sẽ bằng $+\infty$ nếu không có hạn chế về lượng phát của điểm phát s_k, và nếu lượng phát của s_k bị hạn chế bởi b_k thì cung (s, s_k) có khả năng thông qua là b_k. Cũng như vậy, đối với các cung nối t_k với điểm thu t, giá trị khả năng thông qua của (t_k, t) sẽ là giới hạn hoặc không giới hạn tuỳ theo lượng thu của điểm thu này có bị giới hạn hay không.



Hình 4. Mạng với nhiều điểm phát và thu

Trường hợp một số điểm thu chỉ nhận "hàng" từ một số điểm phát ta có bài toán nhiều luồng là một bài toán phức tạp hơn rất nhiều so với bài toán luồng cực đại giữa điểm phát s và điểm thu t.

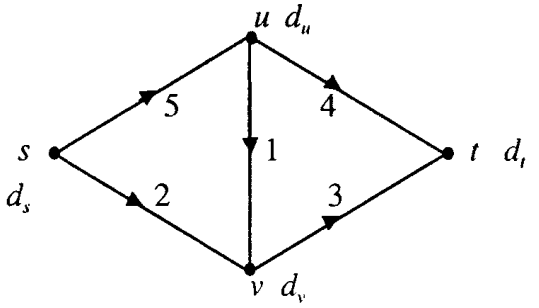
b) Bài toán với khả năng thông qua của các cung và các đỉnh.

Giả sử trong đồ thị G, ngoài khả năng thông qua của các cung $c(u, v)$, ở mỗi đỉnh $v \in V$ còn có khả năng thông qua của đỉnh là $d(v)$, và đòi hỏi tổng luồng đi vào đỉnh v không được vượt quá $d(v)$, tức là

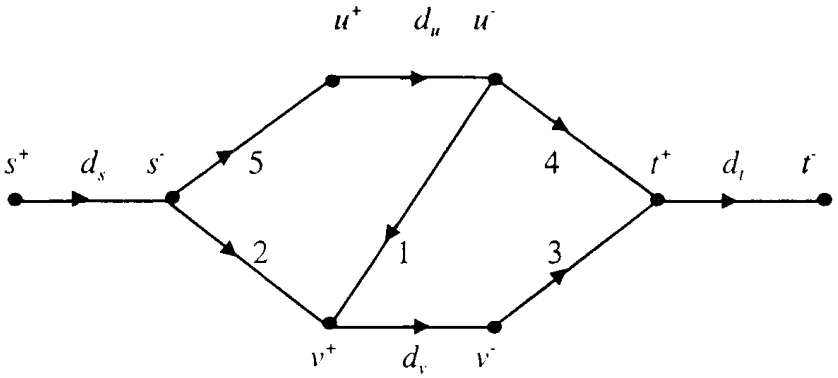
$$\sum_{w \in V} f(w,v) \leq d(v).$$

Cần phải tìm luồng cực đại giữa s và t trong mạng như vậy.

Xây dựng một mạng G' sao cho: mỗi đỉnh v của G tương ứng với 2 đỉnh v^+, v^- trong G', mỗi cung (u, v) trong G ứng với cung (u^-, v^+) trong G', mỗi cung (v,w) trong G ứng với cung (v^-, w^+) trong G'. Ngoài ra, mỗi cung (v^+, v^-) trong G' có khả năng thông qua là $d(v)$, tức là bằng khả năng thông qua của đỉnh v trong G.



Hình 5a. Mạng G với khả năng thông qua ở cung và đỉnh.



Hình 5.b. Mạng G' tương ứng chỉ có khả năng thông qua trên các cung.

Hình 5. Mạng với khả năng thông qua trên các đỉnh và các cung.

Do luồng đi vào đỉnh v^+ phải đi qua cung (v^+, v^-) với khả năng thông qua $d(v)$, nên luồng cực đại trong G' sẽ bằng luồng cực đại trong G với khả năng thông qua của các cung và các đỉnh.

c) Mạng trong đó khả năng thông qua của mỗi cung bị chặn hai phía.

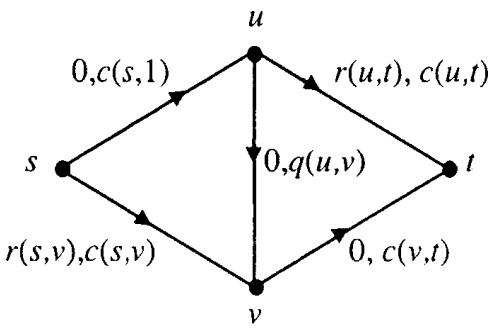
Xét mạng G mà trong đó mỗi cung (u, v) có khả năng thông qua (cận trên của luồng trên cung) $c(u, v)$ và cận dưới của luồng là $d(u, v)$. Bài toán đặt ra là liệu có tồn tại luồng tương thích từ s đến t, tức là luồng $\{f(u, v): u, v \in V\}$ thoả mãn thêm ràng buộc

$$d(u, v) \leq f(u, v) \leq c(u, v), \quad \forall (u, v) \in E,$$

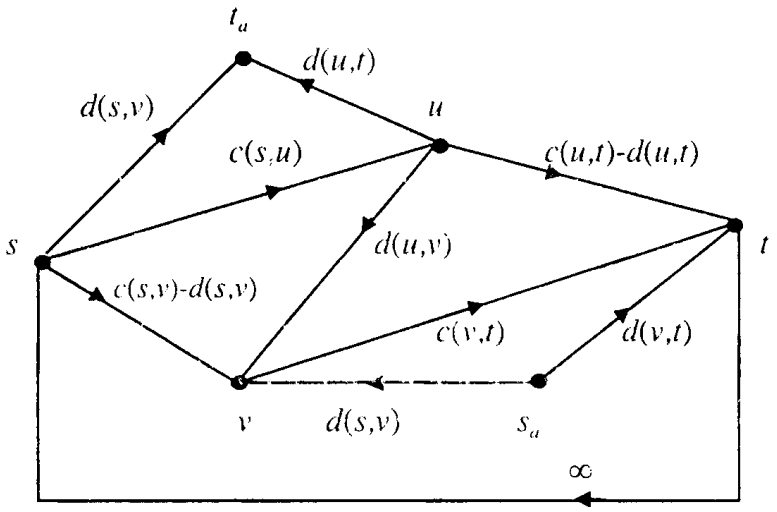
hay không?

Đưa vào mạng G đỉnh phát giả s_a và đỉnh thu giả t_a và xây dựng mạng G_a theo qui tắc: Mỗi cung (u, v) mà $d(u, v) \neq 0$ sẽ tương ứng với 2 cung (s_a, v) và (u, t_a) với khả năng thông qua là $d(u, v)$. Giảm $c(u, v)$ đi $d(u, v)$ tức là thay khả năng thông qua của cung (u, v) bởi $c(u, v) - d(u, v)$ còn cận dưới cuả nó đặt bằng 0. Ngoài ra thêm vào cung (t, s) với $c(t, s) = \infty$.

Hình 6a cho ví dụ mạng G với khả năng thông qua của các cung bị chặn cả hai phía. Đồ thị G_a tương ứng được cho trong Hình 6(b).



Hình 6a



Hình 6b

Hình 6. Mạng với khả năng thông qua bị chặn hai phía.

Ký hiệu

$$d^* = \sum_{d(u,v)\neq 0} d(u,v).$$

Ta có kết quả sau đây.

Định lý 4. *1) Nếu luồng lớn nhất trong mạng G_a từ s_a đến t_a bằng d^* thì tồn tại luồng tương thích trong G.*

2) Nếu luồng lớn nhất trong mạng G_a từ s_a đến t_a là khác d^* thì không tồn tại luồng tương thích trong G.

7.5. Một số ứng dụng trong tổ hợp

Bài toán luồng cực đại có rất nhiều ứng dụng trong việc giải nhiều bài toán tổ hợp. Khó khăn chính ở đây là phải xây dựng mạng tương ứng sao cho việc tìm luồng cực đại trong nó sẽ tương đương với việc giải bài toán tổ hợp đặt ra. Mục này sẽ giới thiệu một số bài toán như vậy.

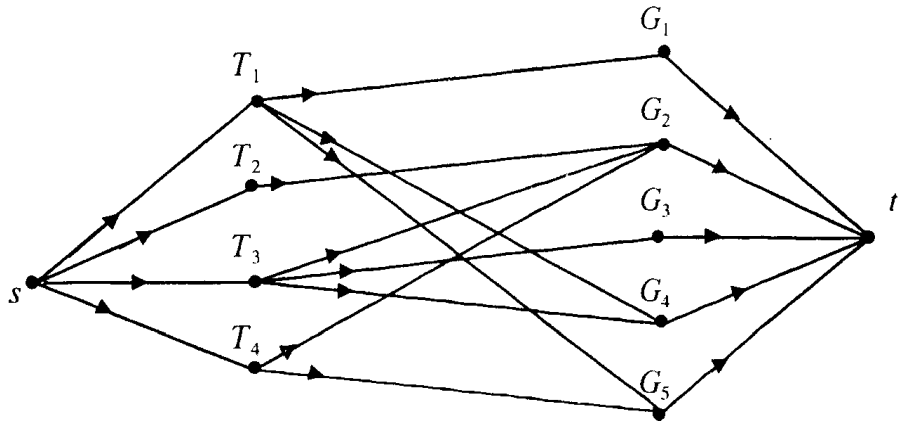
a) Bài toán đám cưới vùng quê. Có *m* chàng trai ở một làng quê nọ. Đối với mỗi chàng trai ta biết các cô gái mà anh ta vừa ý. Hỏi khi nào thì có thể tổ chức các đám cưới trong đó chàng trai nào cũng sánh duyên với cô gái mà mình vừa ý.

Ta có thể xây dựng đồ thị với các đỉnh biểu thị các chàng trai và các cô gái, còn các cung biểu thị sự vừa ý của các chàng trai đối với các cô gái. Khi đó ta thu được một đồ thị hai phía.

Thí dụ. Có 4 chàng trai $\{T_1, T_2, T_3, T_4\}$ và 5 cô gái $\{G_1, G_2, G_3, G_4, G_5\}$. Sự vừa ý cho trong bảng sau

Chàng trai	Các cô gái mà chàng trai ưng ý
T_1	G_1, G_4, G_5
T_2	G_2
T_3	G_2, G_3, G_4
T_4	G_2, G_4

Đồ thị tương ứng được cho trong hình 7.



Hình 7. Mạng tương ứng với Bài toán đám cưới vùng quê

Đưa vào điểm phát s và điểm thu t. Nối s với tất cả các đỉnh biểu thị các chàng trai, và nối t với tất cả các đỉnh biểu thị các cô gái. Tất cả các cung của đồ thị đều có khả năng thông qua bằng 1. Bắt đầu từ luồng 0, ta tìm luồng cực đại trong mạng xây dựng được theo thuật toán Ford - Fulkerson. Từ định lý về tính nguyên, luồng trên các cung là các số 0 hoặc 1. Rõ ràng là nếu luồng cực đại trong đồ thị có giá trị $V_{max} = m$, thì bài toán có lời giải, và các cung với luồng bằng 1 sẽ chỉ ra cách tổ chức đám cưới thoả mãn điều kiện đặt ra. Ngược lại, nếu bài toán có lời giải thì $V_{max} = m$. Bài toán về các đám cưới vùng quê là một trường hợp riêng của bài toán về cặp ghép trên đồ thị hai phía mà để giải nó có thể xây dựng thuật toán hiệu quả hơn.

b) Bài toán về hệ thống đại diện chung.

Cho tập m phần tử $X = \{z_1, z_2, ..., z_m\}$. Giả sử $<A_1, A_2, ..., A_n>$ và $<B_1, B_2, ..., B_n>$ là hai dãy các tập con của tập X. Dãy gồm n phần tử khác nhau của X: $<a_1, a_2, ..., a_n>$ được gọi là hệ

thống các đại diện chung của hai dãy đã cho nếu như tìm được một hoán vị σ của tập{1, 2,..., n} sao cho <a_1, a_2,..., a_n> là hệ thống các đại diện phân biệt của hai dãy <A_1, A_2,..., A_n> và <$B_{\sigma(1)}$, $B_{\sigma(2)}$,..., $B_{\sigma(n)}$>, tức là điều kiện sau được thoả mãn: $a_i \in A_i \cap B_{\sigma(i)}$, $i = 1, 2,..., n$. Xây dựng mạng $G = (V, E)$ với tập đỉnh

$$V = \{s, t\} \cup \{x_1, x_2,..., x_n\} \cup \{u_1, u_2,..., u_m\}$$
$$\cup \{v_1, v_2,..., v_m\} \cup \{y_1, y_2,..., y_n\},$$

trong đó đỉnh x_i tương ứng với tập A_i, đỉnh y_i tương ứng với tập B_i, các phần tử u_j, v_j tương ứng với phần tử z_j. Tập các cung của mạng G được xác định như sau

$$E = \{ (s,x_i): 1 \le i \le n\} \cup \{(x_i,u_j): \text{với } z_j \in A_i, 1 \le i \le n, 1 \le j \le m\} \cup$$
$$\cup \{(u_j, v_j): 1 \le j \le m\} \cup$$
$$\cup \{(v_j, y_i): \text{với } z_j \in B_i, 1 \le i \le n, 1 \le j \le m\} \cup \{(y_i, t): 1 \le i \le n\}.$$

Khả năng thông qua của tất cả các cung được đặt bằng 1. Dễ dàng thấy rằng *hệ thống đại diện chung của hai dãy <A_1, A_2,..., A_n> và <B_1, B_2,..., B_n> tồn tại khi và chỉ khi trong mạng $G = (V, E)$ tìm được luồng với giá trị n.* Để xét sự tồn tại của luồng như vậy có thể sử dụng thuật toán tìm luồng cực đại từ s đến t trong mạng $G = (V, E)$.

c) Về một bài toán tối ưu rời rạc.

Trong mục này ta sẽ trình bày thuật toán được xây dựng dựa trên thuật toán tìm luồng cực đại để giải một bài toán tối ưu rời rạc là mô hình toán học cho một số bài toán tối ưu tổ hợp.

Xét bài toán tối ưu rời rạc:

$$f(x_1, x_2,..., x_n) = \max_{1 \le i \le m} \sum_{j=1}^{n} x_{ij} \to \min \qquad (1)$$

với điều kiện

$$\sum_{j=1}^{n} a_{ij} x_{ij} = p_i, \quad i = 1,2,...,m, \qquad (2)$$

$$x_{ij} = 0 \text{ hoặc } 1, \ j = 1, 2,..., n \qquad (3)$$

trong đó $a_{ij} \in \{0,1\}$, $i = 1, 2, ..., m$; $j = 1, 2, ..., n$, p_i -nguyên dương, $i = 1, 2, ..., m$.

Bài toán (1)-(3) là mô hình toán học cho nhiều bài toán tối ưu tổ hợp thực tế. Dưới đây ta dẫn ra một vài ví dụ điển hình.

Bài toán phân nhóm sinh hoạt. Có m sinh viên và n nhóm sinh hoạt chuyên đề. Với mỗi sinh viên i, biết

$a_{ij} = 1$, nếu sinh viên i có nguyện vọng tham gia vào nhóm j,
$a_{ij} = 0$, nếu ngược lại,

và p_i là số lượng nhóm chuyên đề mà sinh viên i phải tham dự, $i = 1, 2, ..., m$; $j = 1, 2, ..., n$.

Trong số các cách phân các sinh viên vào các nhóm chuyên đề mà họ có nguyện vọng tham gia và đảm bảo mỗi sinh viên i phải tham gia đúng p_i nhóm, hãy tìm cách phân phối với số người trong nhóm có nhiều sinh viên tham gia nhất là nhỏ nhất có thể được.

Đưa vào biến số

$x_{ij} = 1$, nếu sinh viên i tham gia vào nhóm j,

$x_{ij} = 0$, nếu ngược lại,

$i = 1, 2, ..., m, j = 1, 2, ..., n$, khi đó dễ thấy mô hình toán học cho bài toán đặt ra chính là bài toán (1) - (3).

Bài toán lập lịch cho hội nghị. Một hội nghị có m tiểu ban, mỗi tiểu ban cần sinh hoạt trong một ngày tại phòng họp phù hợp với nó. Có n phòng họp dành cho việc sinh hoạt của các tiểu ban. Biết

$a_{ij} = 1$, nếu phòng họp i là thích hợp với tiểu ban j,

$a_{ij} = 0$, nếu ngược lại,

$i = 1, 2, ..., m; j = 1, 2, ..., n$. Hãy bố trí các phòng họp cho các tiểu ban sao cho hội nghị kết thúc sau ít ngày làm việc nhất.

Đưa vào biến số

$x_{ij} = 1$, nếu bố trí tiểu ban i làm việc ở phòng j,

$x_{ij} = 0$, nếu ngược lại,

$i = 1, 2, ..., m, j = 1, 2, ..., n$, khi đó dễ thấy mô hình toán học cho bài toán đặt ra chính là bài toán (1) - (3), trong đó $p_i = 1$, $i = 1, 2, ..., m$.

Bổ đề 2. *Bài toán (1)-(3) có phương án tối ưu khi và chỉ khi*

$$\sum_{j=1}^{n} a_{ij} \geq p_i, \quad i = 1,2,...,m, \tag{4}$$

Chứng minh. Điều kiện cần của bổ đề là hiển nhiên vì từ sự tồn tại phương án của bài toán suy ra các bất đẳng thức trong (4) được thực hiện ít nhất dưới dạng dấu đẳng thức. Để chứng minh điều kiện đủ, chỉ cần chỉ ra rằng nếu điều kiện (4) được thực hiện thì bài toán luôn có phương án. Thực vậy, giả sử điều kiện (4) được thực hiện. Khi đó nếu ký hiệu

$$I_{+i} = \{ 1 \leq j \leq n : a_{ij} = 1 \},$$

thì $|I_{+i}| \geq p_i$, $i = 1, 2, ..., m$. Do đó nếu gọi

$I_i \subset I_{+i}$, $|I_i| = p_i$, $i = 1, 2, ..., m$,

thì $X^* = (x^*_{ij})_{m\times n}$ với các thành phần được xác định theo công thức

$$x^*_{ij} = 1,\ j \in I_i,\ x^*_{ij} = 0,\ j \notin I_i,\ i = 1, 2, ..., m, \qquad (5)$$

là phương án của bài toán (1) - (3). Bổ đề được chứng minh.

Do (4) là điều kiện cần để bài toán (1)-(3) có phương án, nên trong trong phần tiếp theo ta sẽ luôn giả thiết rằng điều kiện này được thực hiện.

Bây giờ ta sẽ chỉ ra rằng việc giải bài toán (1)-(3) có thể dẫn về việc giải một số hữu hạn bài toán luồng cực đại trong mạng. Trước hết, với mỗi số nguyên dương k, xây dựng mạng $G(k) = (V,E)$ với tập đỉnh

$$V = \{s\} \cup \{u_i : i = 1, 2, ..., m\} \cup \{w_j : j = 1, 2,..., n\} \cup \{t\},$$

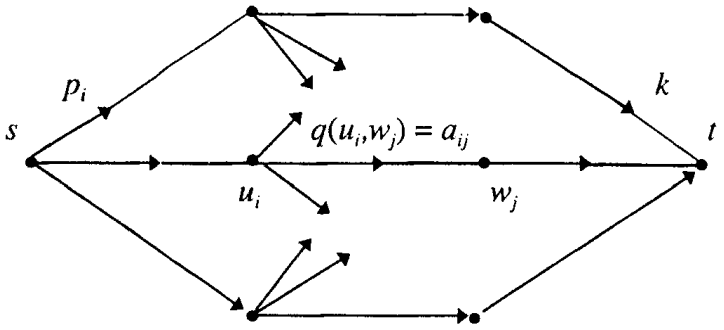
trong đó s là điểm phát, t là điểm thu, và tập cung

$$E = \{(s,u_i) : i=1, 2,..., m\} \cup \{(u_i,w_j) : i=1, 2,..., m; j = 1, 2,..., n\}$$
$$\cup \{(w_j,t) : j =1, 2,..., n\}.$$

Mỗi cung $e \in E$ được gán với khả năng thông qua $q(e)$ theo qui tắc sau:

$$q(s,u_i) = p_i,\ i= 1, 2,..., m,$$
$$q(u_i,w_j) = a_{ij},\ i = 1, 2,..., m; j = 1, 2,..., n;$$
$$q(w_j, t) = k, j = 1, 2,..., n.$$

Hình 8 chỉ ra cách xây dựng mạng $G(k)$.



Hình 8. Mạng $G(k)$.

Ký hiệu: $\sigma = \sum_{i=1}^{m} p_i$.

Bổ đề sau đây cho thấy mối liên hệ giữa luồng cực đại trong mạng $G(k)$ và phương án của bài toán (1)-(3).

Bổ đề 3. *Giả sử đối với số nguyên dương k nào đó, luồng cực đại nguyên ξ^* trong mạng G(k) có giá trị là σ. Khi đó $X^* = (x^*_{ij})_{m\times n}$ với các thành phần được xác định theo công thức*

$$x^*_{ij} = \xi^*(u_i,w_j),\ i = 1,2,...,m;\ j=1,2,...,n,$$

là phương án của bài toán (1)-(3).

Chứng minh. Thực vậy, do luồng cực đại trong mạng có giá trị là σ và là luồng nguyên nên

$$\xi^*(s,u_i) = p_i,\ i = 1, 2,..., m,$$

$$\xi^*(u_i,w_j) \in \{0, 1\},\ i = 1, 2,..., m;\ j = 1, 2,..., n,$$

từ đó suy ra

$$\sum_{j=1}^{n} x^*_{ij} = \sum_{j=1}^{n} \xi^*(u_i,v_j) = \xi^*(s,u_i) = p_i,\ i = 1,2,...,m.$$

Vậy X^* là phương án của bài toán (1)-(3). Bổ đề được chứng minh.

Bổ đề 4. Giả sử $X^* = (x^*_{ij})$ *là phương án tối ưu và k^* là giá trị tối ưu của bài toán* (1)-(3). *Khi đó luồng cực đại trong mạng $G(k^*)$ có giá trị là σ.*

Chứng minh. Do giá trị của luồng cực đại trong mạng $G(k^*)$ không vượt quá σ, nên để chứng minh bổ đề ta chỉ cần chỉ ra luồng với giá trị σ trong mạng $G(k^*)$. Xây dựng luồng ξ^* theo công thức sau:

$$\xi^*(s,u_i) = p_i,\ \ \xi^*(u_i,w_j) = x^*_{ij},$$

$$\xi^*(w_j,t) = \sum_{i=1}^{m} x^*_{ij},\ i = 1,2,...,m,\ j = 1,2,...,n.$$

Dễ dàng kiểm tra được rằng ξ^* là luồng trong mạng $G(m)$ có giá trị σ. Bổ đề được chứng minh.

Bổ đề 5. *Nếu k = m thì luồng cực đại trong mạng G(m) có giá trị là σ.*

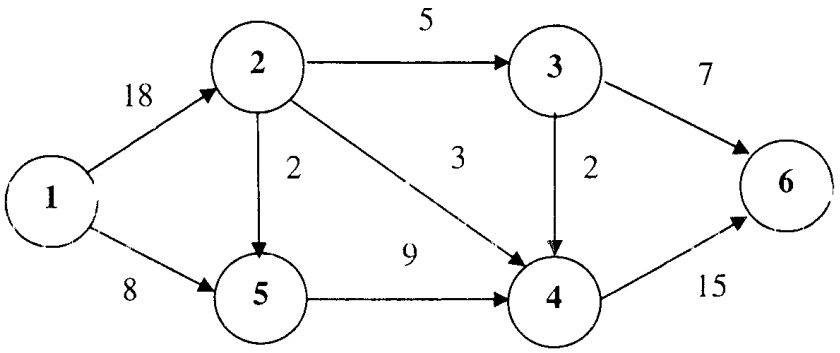
Chứng minh. Lập luận tương tự như trong Bổ đề 4, ta chỉ cần chỉ ra luồng với giá trị σ trong mạng $G(m)$. Thực vậy, giả sử $X^* = (x^*_{ij})_{m\times n}$ là phương án của bài toán (1)-(3) xây dựng theo công thức (5). Xây dựng luồng ξ^* theo công thức giống như trong chứng minh bổ đề 4, ta có luồng với giá trị σ. Bổ đề được chứng minh.

Từ bổ đề 3 và 4 suy ra việc giải bài toán (1) - (3) dẫn về việc tìm giá trị k^* nguyên dương nhỏ nhất sao cho luồng cực đại trong mạng $G(k^*)$ có giá trị σ. Bổ đề 5 cho thấy giá trị $k^* \in [1, m]$. Vì vậy để giải bài toán (1) - (3) ta có thể áp dụng phương pháp tìm kiếm nhị phân trên đoạn $[1, m]$ để tìm giá trị k^*, trong đó ở mỗi bước cần giải một bài toán luồng cực đại. Để giải bài toán tìm luồng cực đại trong mạng có thể sử dụng các thuật toán đa thức như đã nói ở trên. Từ đó suy ra kết quả sau

Định lý 5. *Bài toán* (1)-(3) *giải được nhờ thuật toán đa thức với thời gian tính là* $\log_2 m \times O_{NF}$, *trong đó* O_{NF} *là thời gian tính của thuật toán giải bài toán tìm luồng cực đại trong mạng* $G(k)$.

Bài tập

1. Xét mạng cho trong hình vẽ sau, các số bên cạnh các cung là khả năng thông qua của nó



Thực hiện thuật toán Ford-Fulkerson tìm luồng cực đại trên mạng đã cho bắt đầu từ luồng 0. Trình bày các kết quả tính toán trong mỗi bước lặp bao gồm:

- Đồ thị tăng luồng,
- Đường tăng luồng tìm được theo tìm kiếm theo chiều rộng và khả năng thông qua của nó (Giả thiết khi duyệt các đỉnh kề của một đỉnh ta duyệt theo thứ tự tăng dần của chỉ số),
- Mạng cùng luồng thu được sau khi tăng luồng.

Kết quả cuối cùng: Cần đưa ra giá trị của luồng cực đại và lát cắt hẹp nhất

2. Cho $G = (V, E)$ đồ thị có hướng trong đó không có cung (s, t). Chứng minh rằng số đường đi cơ bản nối hai đỉnh s và t là bằng số ít nhất các đỉnh của đồ thị cần loại bỏ để trong đồ thị không còn đường đi nối s với t.

3. Xây dựng thuật toán tìm tập E_1 tất cả các cung của đồ thị mà việc tăng khả năng thông qua của bất kỳ cung nào trong E đều dẫn đến tăng giá trị của luồng cực đại trong mạng.

4. Cho hai dãy số nguyên dương $\{p_i, i=1, 2,..., m\}$ và $\{q_j, j=1, 2,..., n\}$. Hãy xây dựng ma trận $A = \{a_{ij} : i=1, 2,..., m; j=1, 2,..., n\}$ với các phần tử $a_{ij} \in \{0, 1\}$ có tổng các phần tử trên dòng i là p_i, tổng các phần tử trên cột j là q_j.

5. Có m chàng trai, n cô gái và k bà mối. Mỗi bà mối p ($p = 1, 2,..., k$) có một danh sách L_p một số chàng trai và cô gái trong số các chàng trai và cô gái nói trên là khách hàng của bà ta. Bà mối p có thể se duyên cho bất cứ cặp trai-gái nào là khách hàng của bà ta, nhưng không đủ sức tổ chức quá d_p đám cưới. Hãy xây dựng thuật toán căn cứ vào danh sách L_p, b_p, $p = 1, 2,..., k$, đưa ra cách tổ chức nhiều nhất các đám cưới giữa m chàng trai và n cô gái với sự giúp đỡ của các bà mối.

6. Bài toán phủ đỉnh nhỏ nhất: Cho đồ thị vô hướng $G = (V, E)$ với trọng số trên các đỉnh $c(v)$, $v \in V$. Một tập con các đỉnh của đồ thị S được gọi là phủ đỉnh nếu như mỗi cạnh của đồ thị có ít nhất một đầu mút trong S. Trọng lượng của phủ đỉnh S là tổng các trọng số của các đỉnh trong nó. Bài toán đặt ra là tìm phủ đỉnh có trọng lượng nhỏ nhất (mà ta sẽ gọi là phủ đỉnh nhỏ nhất). Hãy chỉ ra cách qui dẫn bài toán tìm phủ đỉnh nhỏ nhất trên đồ thị hai phía về bài toán luồng cực đại.

7. Cho đơn đồ thị vô hướng $G = (V, E)$. Giả sử s, t là hai đỉnh của đồ thị. Sử dụng lý thuyết luồng cực đại trong mạng chứng minh khẳng định sau: "*Số lượng lớn nhất các đường đi đôi một không có cạnh chung nối hai đỉnh s và t là bằng số cạnh ít nhất cần loại bỏ khỏi đồ thị G để không còn đường đi nối s và t*".

8. Bài toán ghép cặp trên đồ thị hai phía: Cho đồ thị hai phía $G = (X \cup Y, E)$. Một tập con các cạnh của đồ thị được gọi là một cặp ghép nếu như hai cạnh bất kỳ trong nó không có đỉnh chung. Bài toán đặt ra là tìm cặp ghép có lực lượng lớn nhất của G. Hãy chỉ ra cách qui dẫn bài toán đặt ra về bài toán luồng cực đại trong mạng.

9. Xây dựng hình chữ nhật latinh. Hãy chỉ ra cách xây dựng hình chữ nhật la tinh cấp $m \times n$ ($m \leq n$) nhờ sử dụng thuật toán tìm cặp ghép lớn nhất trên đồ thị hai phía.

PHẦN III

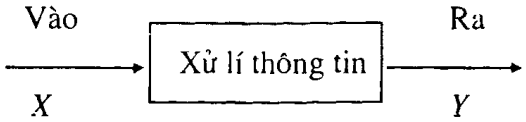
HÀM ĐẠI SỐ LÔGIC

1

MỞ ĐẦU

1.1. Mô hình xử lí thông tin và hàm đại số lôgic

Trong nhiều vấn đề, ta thường gặp một mô hình xử lý thông tin dạng đơn giản, gồm một đầu vào và một đầu ra:



Đầu vào và đầu ra lấy giá trị từ những tập hữu hạn X và Y. Ta xem mô hình hoạt động trên một trục thời gian được phân chia thành những thời điểm rời rạc. Tại thời điểm t, tín hiệu vào là $x_t \in X$ và tín hiệu ra là $y_t \in Y$. Giả thiết rằng mô hình hoạt động không nhớ, tức là tín hiệu ra tại thời điểm t chỉ phụ thuộc vào tín hiệu vào tại thời điểm đó. Như vậy, có thể mô tả sự hoạt động của mô hình bằng một hàm $f: X \to Y$, trong đó $y = f(x)$ là tín hiệu ra tương ứng với tín hiệu vào x.

Trong kỹ thuật điều khiển tự động, các tín hiệu thường được truyền đi và chế biến dưới dạng chỉ có 2 giá trị, mà ta sẽ ký hiệu là 0 và 1. Vì vậy có thể xem X như tập hợp E^n và Y như tập hợp E^m, trong đó $E = \{0, 1\}$ (với n và m được chọn thích hợp). Mỗi

phần tử $x \in X$ có dạng $x = (x_1, x_2, ..., x_n)$ và mỗi phần tử $y \in Y$ có dạng $y = (y_1, y_2, ..., y_m)$ trong đó $x_i, y_i \in E$.

Một hàm $f: X \to Y$ được phân tích thành một hệ m hàm

$$y_1 = f_1(x_1, x_2, ..., x_n)$$
$$y_2 = f_2(x_1, x_2, ..., x_n)$$
$$\cdots\cdots\cdots\cdots$$
$$y_m = f_m(x_1, x_2, ..., x_n)$$

Mỗi hàm $y_i = f_i(x_1, x_2, ..., x_n)$ có các đối số nhận giá trị 0, 1 và hàm cũng nhận các giá trị 0,1. Một hàm như thế được gọi là một hàm đại số lôgic.

Định nghĩa. *Một hàm đại số lôgic của n đối số là một ánh xạ* $E^n \to E$.

Một hàm đại số lôgic thường được xác định bằng bảng giá trị của nó, nghĩa là trong đó liệt kê tất cả các ảnh của các bộ đối số có thể có. Thí dụ bảng dưới đây xác định một hàm đại số lôgic 3 đối số:

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Hình 1

Vì $|E^n| = 2^n$ và $|E| = 2$ nên số các hàm đại số lôgic của n đối số là 2^{2^n}, chẳng hạn với $n = 3$, con số này là $2^8 = 256$.

Một hàm đại số lôgic $y = f(x_1, ..., x_n)$ được gọi là phụ thuộc thực sự vào đối số x_i nếu có 2 bộ giá trị của đối số chỉ khác nhau một thành phần thứ i mà giá trị của hàm tại 2 bộ này là khác nhau.

Không phải mọi hàm đại số lôgic của n đối số đều phụ thuộc thực sự vào cả n đối số của nó. Chẳng hạn các hàm hằng $y = 0$ hay $y = 1$ không phụ thuộc thực sự vào đối số nào cả. Gọi A_n là số tất cả các hàm đại số lôgic phụ thuộc thực sự vào n đối số. Khi đó dễ dàng tính được A_n theo hệ thức truy hồi dưới đây:

$A_0 = 2$,

$A_n = 2^{2^n} - C_n^{n-1} A_{n-1} - \ldots - C_n^1 A_1 - A_0$.

Thông thường, việc mã hoá nhị phân tập X bị dư thừa, nghĩa là có một số phần tử của X tại đó hàm f không xác định (chẳng hạn nếu X có 6 phần tử thì khi mã hoá nó bằng tập E^3, ta thừa 2 phần tử). Một hàm $f: M \to E$, với M là một tập con thực sự của E^n được gọi là một hàm đại số lôgic không đầy đủ và tập con M được gọi là miền xác định của nó. Rõ ràng tại những nơi mà f không xác định, ta có thể cho nó một giá trị (0 hoặc 1) tuỳ ý để được một hàm đại số lôgic đầy đủ. Nói cách khác, với mọi hàm đại số lôgic không đầy đủ f có miền xác định M, luôn xây dựng được một hàm đại số lôgic đầy đủ F để $F(x_1,\ldots,x_n) = f(x_1,\ldots,x_n)$ với mọi $(x_1,\ldots,x_n) \in M$. Hàm F như vậy được gọi là hàm phủ của hàm f. Dễ thấy rằng, số lượng các hàm phủ f bằng 2^k, trong đó $k = 2^n - |M|$.

1.2. Các hàm đại số lôgic sơ cấp

Dưới đây ta sẽ xét cụ thể các hàm đại số lôgic phụ thuộc thực sự vào n đối số với $n \le 2$. Các hàm này được gọi là các hàm sơ cấp.

a) $n = 0$: có 2 hàm là hàm hằng 0 và hàm hằng 1.

b) $n = 1$: có 2 hàm cho bởi bảng

x	f_1	f_2
0	0	1
1	1	0

Hàm $f_2(x)$ được gọi là phủ định của x và được ký hiệu là $\neg x$ hay $\bar{x}$

c) $n = 2$: có 10 hàm cho bởi bảng

x	y	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
0	0	0	0	0	1	1	1	1	1	0	0
0	1	0	1	1	1	0	0	1	0	1	0
1	0	0	1	1	0	0	0	1	1	0	1
1	1	1	1	0	1	1	0	0	1	0	0

trong đó có một số hàm thông dụng như sau:

- hàm $g_1(x, y)$ được gọi là hội (hay tích) của x và y, ký hiệu là $x \,\&\, y$ hay $x.y$, xy
- hàm $g_2(x, y)$ được gọi là tuyển của x và y, ký hiệu $x \vee y$
- hàm $g_3(x, y)$ được gọi là tổng (theo modun 2) của x và y, ký hiệu là $x + y$
- hàm $g_4(x, y)$ được gọi là hàm kéo theo, ký hiệu $x \to y$

- hàm $g_5(x, y)$ được gọi là hàm tương đương, ký hiệu $x \Leftrightarrow y$
- hàm $g_6(x, y)$ được gọi là hàmVebb, ký hiệu x o y
- hàm $g_7(x, y)$ được gọi là hàm Sheffer, ký hiệu $x \mid y$

Quan trọng nhất là các hàm tuyển, hội và phủ định. Chúng tương ứng với các phép toán "hoặc", "và", "không" trong đại số mệnh đề, trong đó giá trị 1 ứng với "đúng" và giá trị 0 ứng với "sai". Dễ thấy rằng các tính chất dưới đây của các hàm tuyển, hội phủ định được suy từ các tính chất quen thuộc của đại số mệnh đề (hoặc cũng có thể suy từ bảng giá trị):

- *Kết hợp*

$$x \vee (y \vee z) = (x \vee y) \vee z = x \vee y \vee z,$$
$$x \,\&\, (y \,\&\, z) = (x \,\&\, y) \,\&\, z = x \,\&\, y \,\&\, z.$$

- *Giao hoán*

$$x \vee y = y \vee x,$$
$$x \,\&\, y = y \,\&\, x.$$

- *Phân bố*

$$x \vee (y \,\&\, z) = (x \vee y) \,\&\, (x \vee z),$$
$$x \,\&\, (y \vee z) = (x \,\&\, y) \vee (x \,\&\, z).$$

- *Đối ngẫu*

$$\overline{x \vee y} = \bar{x} \,\&\, \bar{y},$$

$$\overline{x \,\&\, y} = \bar{x} \vee \bar{y}.$$

Trong kỹ thuật, các hàm tuyển, hội, phủ định được thực hiện nhờ những linh kiện đơn giản. Mục trình bày dưới đây khẳng định rằng, mọi hàm đại số lôgic đều có thể được xây dựng từ những linh kiện đơn giản này.

1.3. Biểu diễn các hàm đại số lôgic qua hệ tuyển, hội và phủ định

Để tiện trình bày, ta đưa vào quy ước sau đây: giả sử x là một biến và $\sigma \in \{0, 1\}$. Khi đó

$$x^{\sigma} = \begin{cases} x, & \text{nếu } \sigma = 1, \\ \bar{x}, & \text{nếu } \sigma = 0. \end{cases}$$

Từ định nghĩa trên dễ thấy

$$x^{\sigma} = 1 \quad \Leftrightarrow \quad x = \sigma.$$

Giả sử $\{f_i : i \in I\}$ là một họ các hàm đại số lôgic. Ta sẽ dùng ký hiệu

$$\bigvee_{i \in I} f_i \qquad \text{và} \qquad \&_{i \in I} f_i$$

để chỉ tuyển và hội của tất cả các hàm này, nói riêng nếu $I = \{1, 2, ..., m\}$, thì ta sẽ dùng các ký hiệu

$$\bigvee_{i=1}^{m} f_i \qquad \text{và} \qquad \&_{i=1}^{m} f_i$$

để thay thế. Ta cũng quy ước rằng nếu I là tập rỗng thì

$$\bigvee_{i \in I} f_i = 0 \qquad \text{và} \qquad \&_{i \in I} f_i = 1.$$

Với mỗi hàm $f(x_1, ..., x_n)$, ký hiệu T_f là tập

$$T_f = \{(x_1, ..., x_n) \in E^n : f(x_1, ..., x_n) = 1\}$$

và gọi nó là tập đặc trưng của f. Dễ dàng chứng minh tương ứng giữa hàm đại số lôgic với tập đặc trưng của nó là tương ứng 1-1 và có các tính chất:

$$T_{\bar{f}} = \bar{T}_f \ , \ T_{f \vee g} = T_f \cup T_g \ , \ T_{f \& g} = T_f \cap T_g.$$

Điều này cho phép chuyển các chứng minh trên đại số lôgic sang các chứng minh tương ứng trên đại số tập hợp.

Định lý. *Mọi hàm đại số lôgic $f(x_1, ..., x_n)$ đều có thể được biểu diễn dưới dạng:*

$$f(x_1, ..., x_n) = \bigvee_{(\sigma_1, ..., \sigma_i) \in E^i} x_1^{\sigma_1} x_2^{\sigma_2} ... x_i^{\sigma_i} \, f(\sigma_1, \sigma_2, ..., \sigma_i, x_{i+1}, ..., x_n)$$

trong đó i là số tự nhiên bất kỳ, $1 \le i \le n$.

Chứng minh. Giả sử $(x_1, ..., x_n) \in T_f$, khi đó số hạng ứng với bộ giá trị $\sigma_1 = x_1, ..., \sigma_i = x_i$ trong tuyển vế phải

$$x_1^{\sigma_1} \cdot \cdot \cdot x_i^{\sigma_i} \, f(\sigma_1, ..., \sigma_i, x_{i+1}, ..., x_n)$$

sẽ bằng 1, điều này kéo theo toàn bộ vế phải sẽ bằng 1. Ngược lại, nếu vế phải bằng 1, thì phải xảy ra bằng 1 tại một số hạng nào đó, chẳng hạn tại số hạng ứng với bộ giá trị $(\sigma_1, ..., \sigma_i)$. Khi đó $x_1 = \sigma_1, ..., x_i = \sigma_i$ và $(x_1, ..., x_n) \in T_f$. Định lý được chứng minh.

Cho $i = 1$ trong định lý và nhận xét rằng vai trò của các biến x_i là như nhau, ta được:

Hệ quả 1. *Hàm đại số lôgic $f(x_1,...,x_n)$ có thể được khai triển theo một đối số x_i:*

$$f(x_1,...,x_n) = \bar{x}_i f(x_1,..., x_{i-1}, 0, x_{i+1},..., x_n) \vee x_i f(x_1,..., x_{i-1}, 1, x_{i+1},..., x_n)$$

Cho $i = n$ trong định lý và bỏ đi các nhân tử bằng 1 trong một tích, ta được:

Hệ quả 2. *Hàm đại số lôgic $f(x_1,...,x_n)$ có thể được khai triển dưới dạng:*

$$f(x_1,...,x_n) = \bigvee_{(\sigma_1,...,\sigma_n) \in T_f} x_1^{\sigma_1} \ldots x_n^{\sigma_n}$$

Công thức khai triển này còn được gọi là dạng tuyển chuẩn tắc hoàn toàn của f và mỗi số hạng của nó được gọi là một cấu tạo đơn vị của f.

Thí dụ. Dạng tuyển chuẩn tắc hoàn toàn của hàm f cho bởi bảng ở hình 1 là

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 x_3.$$

Từ hệ quả 2, ta nhận được khẳng định sau đây: *Mọi hàm đại số lôgic đều có thể xây dựng từ các biến nhờ các hàm tuyển, hội và phủ định.*

Bằng luật đối ngẫu, ta có thể chứng minh một kết quả tương tự bằng cách thay phép tuyển bằng phép hội và ngược lại, từ đó dẫn đến việc biểu diễn f qua một hội các tuyển. Biểu diễn này được gọi là dạng hội chuẩn tắc hoàn toàn của f. Chẳng hạn, trong thí dụ vừa nêu, dạng hội chuẩn tắc hoàn toàn của f là

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \,\&\, (x_1 \vee x_2 \vee \bar{x}_3) \,\&\, (\bar{x}_1 \vee \bar{x}_2 \vee x_3).$$

Ngoài hệ tuyển, hội và phủ định, tồn tại nhiều hệ khác cũng có tính chất mọi hàm đại số lôgic đều được biểu diễn qua các thành viên của hệ. Một hệ hàm như vậy được gọi là một hệ đầy đủ. Chẳng hạn có thể chứng minh các hệ

$$\{0, 1, x + y, x \,\&\, y\}, \{x, x \vee y\}, \{x, x \,\&\, y\}, \{x \mid y\}, \{x \circ y\}$$

đều là những hệ hàm đầy đủ.

Việc nghiên cứu tính đầy đủ của một hệ hàm có một ý nghĩa thực tiễn quan trọng, nó trả lời câu hỏi có thể xây dựng mọi hàm lôgic từ một số hàm đơn giản chọn trước hay không? Về vấn đề này, Post đã thực hiện một cách hệ thống từ năm 1921, bạn đọc có thể tìm hiểu một cách chi tiết qua cuốn sách tiếng Nga "Hàm đại số logic và các lớp Post" xuất bản năm 1966 của các tác giả S.B.Iablonski, G.P. Gavrilov, V.B.Kudriavcev.

1.4. Biểu diễn tối thiểu của hàm đại số lôgic

Biểu diễn một hàm đại số lôgic f qua một hệ hàm đầy đủ H là không duy nhất. Thí dụ hàm Sheffer, khi biểu diễn qua hệ tuyển, hội và phủ định, có thể có các cách

$$x|y = \bar{x}\,\bar{y} \vee \bar{x}\,y \vee x\,\bar{y} = \bar{x} \vee \bar{y}.$$

Mỗi một biểu diễn f tương ứng với một cách "ghép" các thành viên của H (mà ta gọi là các yếu tố cơ bản) để thu được f. Hiển nhiên, một vấn đề có ý nghĩa thực tế quan trọng là, cần tìm một biểu diễn sao cho việc ghép như thế là tốn ít yếu tố cơ bản nhất. Theo một nghĩa nào đó, điều này dẫn về việc tìm một công thức trên hệ H biểu diễn hàm f với số ký hiệu các yếu tố này là ít nhất. Một công thức như vậy, được gọi là một biểu diễn tối thiểu của hàm f trong hệ H.

Về nguyên tắc, số công thức biểu diễn f là hữu hạn, nên bằng cách duyệt tất cả các khả năng, ta luôn tìm được biểu diễn tối thiểu của f. Tuy nhiên, số khả năng này là rất lớn và việc duyệt nó đòi hỏi một khối lượng tính toán khổng lồ, do đó trên thực tế khó mà thực hiện được dù rằng ngay cả với những siêu máy tính.

Việc xây dựng những thuật toán hữu hiệu tìm biểu diễn tối thiểu của các hàm đại số lôgic, vì thế càng trở nên cấp bách nhưng đồng thời nó cũng là bài toán rất khó. Cho đến nay, bài toán này vẫn chưa được giải quyết thoả đáng ngay cả trong một số trường hợp đơn giản và còn đang được tiếp tục nghiên cứu.

Một hệ đầy đủ được nghiên cứu nhiều nhất là hệ tuyển, hội và phủ định. Bài toán tìm biểu diễn tối thiểu của các hàm đại số lôgic trong hệ này đã được nghiên cứu nhiều trong vài chục năm gần đây. Tuy nhiên, các kết quả đạt được thường mới chỉ đề cập đến một dạng biểu diễn riêng biệt trong hệ, đó là dạng tuyển chuẩn tắc của các hàm đại số lôgic mà ta sẽ xét kỹ trong chương sau.

Bài tập

1. Xét các hàm đại số lôgic 3 đối số $f(x, y, z)$. Cho tập đặc trưng T_f. Xác định dạng tuyển chuẩn tắc hoàn toàn của f:

a) $T_f = \{(0, 1, 1), (1, 0, 0), (1, 1, 0)\}$

b) $T_f = \{(0, 0, 1), (0, 1, 1), (1. 0, 0), (1, 0, 1), (1, 1, 1)\}$

c) $T_f = \{(0, 0, 0), (1, 1, 1), (0, 1, 1), (1, 0, 0)\}$

2. Có bao nhiêu hàm đại số lôgic 3 đối số mà tập đặc trưng T_f của nó có 5 phần tử? Khái quát hoá: có bao nhiêu hàm đại số lôgic n đối số mà tập đặc trưng T_f của nó có k ($k \le 2^n$) phần tử?

3. Tìm dạng tuyển chuẩn tắc hoàn toàn của các hàm dưới đây:

a) $f(x, y) = x + y$ (hàm cộng modun 2)

b) $f(x, y) = x \circ y$ (hàm Vebb)

c) $f(x, y) = x \mid y$ (hàm Sheffer)

d) $f(x, y, z) = (x \vee y) \,\&\, \bar{z}$

e) $f(x, y, z) = x \vee y \vee z$

f) $f(x, y, z) = (x \,\&\, y) \vee (x \,\&\, z) \vee (y \,\&\, z)$

g) $f(x, y, z) = (x \mid y) \circ z$

4. Hàm đại số lôgic $f(x, y, z)$ nhận giá trị 0 tại (0, 0, 0). Nếu thay đổi giá trị bất cứ đối số nào của nó thì giá trị của hàm cũng thay đổi (so với giá trị trước đấy). Tìm dạng tuyển chuẩn tắc hoàn toàn của f.

5. Xét xem các hàm f, g dưới đây khác nhau hay bằng nhau:

a) $f(x, y, z) = (x \,\&\, y) \vee \bar{z}$

$g(x, y, z) = (\bar{x} \vee y \vee \bar{z}) \,\&\, (x \vee \bar{y} \vee \bar{z}) \,\&\, (x \vee y \vee \bar{z})$

b) $f(x, y, z) = (x \mid y) \circ z$

$g(x, y, z) = x \,\&\, (y + z)$

6. Chứng minh các hệ hàm sau đây là hệ đầy đủ:

$\{0, 1, x + y, x \,\&\, y\}$, $\{\bar{x}, x \vee y\}$, $\{\bar{x}, x \,\&\, y\}$, $\{x \mid y\}$, $\{x \circ y\}$

7. Biểu diễn hàm $f(x, y) = x \vee \bar{y}$ qua các hệ hàm:

a) $\{\bar{x}, x \,\&\, y\}$

b) $\{x \mid y\}$

c) $\{x \circ y\}$

2

DẠNG TUYỂN CHUẨN TẮC CỦA HÀM ĐẠI SỐ LÔGIC

2.1. Các khái niệm cơ bản

Giả sử cho n biến $x_1, x_2, ..., x_n$. Một biểu thức dạng

$$x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} ... x_{i_j}^{\sigma_j}$$

trong đó $\sigma_1,..., \sigma_j \in \{0, 1\}$, $1 \le i_1, ..., i_j \le n$ và $i_t \ne i_s$ nếu $t \ne s$ được gọi là một hội sơ cấp của n biến $x_1, ..., x_n$.

Số các biến xuất hiện trong một hội sơ cấp được gọi là hạng của hội sơ cấp đó. Chẳng hạn, hội sơ cấp $x_1 \bar{x}_3 x_4$ có hạng 3, hội sơ cấp $\bar{x}_2 x_3 x_5 \bar{x}_7$ có hạng 4.

Giả sử $f(x_1,..., x_n)$ là một hàm đại số lôgic. Một công thức biểu diễn f dưới dạng tuyển của một số hội sơ cấp khác nhau của các biến $x_1, ..., x_n$ được gọi là một dạng tuyển chuẩn tắc của hàm đó.

Thí dụ 1.

$\bar{x}y \vee x\bar{y}$ là một dạng tuyển chuẩn tắc của hàm $x + y$.

$\bar{x} \vee \bar{y}$ và $\bar{x}\bar{y} \vee \bar{x}y \vee x\bar{y}$ là các dạng tuyển chuẩn tắc của hàm Sheffer $x \mid y$.

Dễ thấy rằng, dạng tuyển chuẩn tắc hoàn toàn của f là dạng chuẩn tắc duy nhất của f mà trong đó mọi hội sơ cấp đều có hạng n. Mỗi hội sơ cấp (hạng n) trong dạng tuyển chuẩn tắc hoàn toàn của f được gọi là một cấu tạo đơn vị của f.

Ta gọi độ phức tạp của một dạng chuẩn tắc là số các ký hiệu biến xuất hiện trong dạng chuẩn tắc đó.

Thí dụ 2.

$\bar{x}\bar{y} \vee \bar{x}y \vee x\bar{y}$ có độ phức tạp là 6,

$\bar{x} \vee \bar{y}$ có độ phức tạp là 2.

Dạng tuyển chuẩn tắc của f có độ phức tạp bé nhất được gọi là dạng tuyển chuẩn tắc tối thiểu của f. Dạng tuyển chuẩn tắc tối thiểu, theo một nghĩa nào đó, tương ứng với một sơ đồ tối thiểu (thuộc một loại nhất định) thực hiện f. Những phần tiếp theo của chương này, giới thiệu một số phương pháp tìm dạng tuyển chuẩn tắc tối thiểu của một hàm đại số lôgic.

Ta đưa thêm một khái niệm quan trọng sau đây.

Giả sử $f(x_1,..., x_n)$ là một hàm đại số lôgic. Một hàm $g(x_1,..., x_n)$ được gọi là một nguyên nhân (implicant) của f nếu T_g là tập con của T_f, nói cách khác, nếu $(g \to f) = 1$ (vì thế có tên gọi là nguyên nhân).

Dễ dàng thấy rằng mỗi hội sơ cấp trong một dạng tuyển chuẩn tắc của f là một nguyên nhân của f.

Xét một hội sơ cấp A. Việc xoá bớt một biến trong nó làm tập đặc trưng T_A của nó "nở" ra. Điều đó khiến cho, nếu trước đó A là một nguyên nhân của f, thì sau khi xoá một biến, A có thể không phải là nguyên nhân của f nữa. Một hội sơ cấp A được gọi là một nguyên nhân nguyên tố của f, nếu A là một nguyên nhân của f, sao cho không thể xoá đi bất cứ biến nào trong nó (cùng với dấu phủ định nếu có) để A vẫn còn là nguyên nhân của f.

Thí dụ 3.

$\bar{x}y$ và $x\bar{y}$ là các nguyên nhân nguyên tố của hàm $x + y$.

$\bar{x}$ và $\bar{y}$ là các nguyên nhân nguyên tố của hàm $x \mid y$ nhưng các hội $\bar{x}\bar{y}$, $\bar{x}y$, $x\bar{y}$ không phải là các nguyên nhân nguyên tố của nó.

2.2. Dạng tuyển chuẩn tắc thu gọn

Định lý 1. *Tuyển của một số bất kỳ các nguyên nhân của hàm f cũng là một nguyên nhân của hàm đó.*

Chứng minh. Tập đặc trưng của tuyển này là hợp của các tập đặc trưng của các nguyên nhân đang xét, vì thế nó cũng là tập con của tập đặc trưng của hàm *f*.

Định lý được chứng minh.

Giả sử S là một hệ các nguyên nhân của *f*. Ta nói rằng hệ S là đầy đủ nếu với mọi giá trị $\sigma \in T_f$ luôn tìm được một nguyên nhân g thuộc hệ để $\sigma \in T_g$.

Định lý 2. *Giả sử S là một hệ đầy đủ các nguyên nhân của hàm f. Khi đó tuyển của tất cả các nguyên nhân trong S sẽ trùng với f (ta cũng nói tuyển này thực hiện f).*

Chứng minh. Từ định nghĩa của hệ đầy đủ, ta nhận được

$$\bigcup_{g \in S} T_g = T_f$$

nhưng vế trái của đẳng thức trên chính là tập đặc trưng của tuyển đang xét. Từ đó nhận được định lý.

Định lý 3. *Tuyển của tất cả các nguyên nhân nguyên tố của hàm f là thực hiện f. Nói khác đi, hệ các nguyên nhân nguyên tố của f là một hệ đầy đủ.*

Chứng minh. Để chứng minh định lý, ta sẽ chứng minh rằng

$$\bigcup_{g \in S} T_g = T_f$$

trong đó S là hệ các nguyên nhân nguyên tố của *f*.

Vì S gồm các nguyên nhân của *f* nên vế trái của đẳng thức trên là tập con của vế phải. Ta chỉ cần chứng tỏ điều ngược lại. Thật vậy, gọi S' là hệ các cấu tạo đơn vị của *f*, ta có (dạng tuyển chuẩn tắc hoàn toàn của *f*):

$$\bigcup_{g' \in S'} T_{g'} = T_f$$

Xét $g' \in S'$, nếu g' không phải là nguyên nhân nguyên tố của *f*, thì bằng cách xoá bớt một số biến (cùng với dấu phủ định nếu có) trong g', ta thu được một nguyên nhân nguyên tố của *f*. Gọi nguyên nhân này là g ($g \in S$). Rõ ràng $T_{g'}$ là tập con của T_g. Từ đó nhận được

$$\bigcup_{g' \in S'} T_{g'} = T_f \subset \bigcup_{g \in S} T_g$$

$$g' \in S' \qquad\qquad g \in S$$

hay

$$T_f \subset \bigcup_{g \in S} T_g$$

và định lý được chứng minh.

Tuyển của tất cả các nguyên nhân nguyên tố của f được gọi là dạng tuyển chuẩn tắc thu gọn của f. Vì tập hợp các nguyên nhân nguyên tố của f là hoàn toàn xác định, nên dạng tuyển chuẩn tắc thu gọn của nó là duy nhất.

Chú ý rằng, dạng tuyển chuẩn tắc thu gọn của một hàm đại số lôgic, nói chung còn khác xa với dạng tuyển chuẩn tắc tối thiểu của hàm đó. Chẳng hạn, hàm $(x + y) \vee z\,(x \vee y)$ có dạng tuyển chuẩn tắc thu gọn là

$$\bar{x}y \vee x\bar{y} \vee xz \vee yz,$$

nhưng một dạng chuẩn tắc tối thiểu của nó là

$$\bar{x}y \vee x\bar{y} \vee xz.$$

Mặc dù dạng tuyển chuẩn tắc thu gọn chưa phải là tối thiểu, nhưng việc tìm nó, như trong những phương pháp trình bày dưới đây, là một bước trung gian quan trọng trong quá trình tìm dạng tuyển chuẩn tắc tối thiểu.

2.3. Dạng tuyển chuẩn tắc nghẽn và dạng tuyển chuẩn tắc tối thiểu

Một hệ nguyên nhân nguyên tố của hàm f được gọi là một hệ nghẽn, nếu nó đầy đủ và không một hệ con thực sự nào của nó là đầy đủ. Nếu S là một hệ nghẽn của f thì tuyển của các thành viên trong S sẽ thực hiện f. Tuyển này được gọi là dạng tuyển chuẩn tắc nghẽn của f. Nói khác đi dạng tuyển chuẩn tắc nghẽn của f là một dạng tuyển chuẩn tắc gồm các nguyên nhân nguyên tố của f, thực hiện f mà không thể bỏ bớt đi một số hạng nào để vẫn thực hiện được f.

Chú ý rằng, dạng tuyển chuẩn tắc nghẽn của một hàm đại số lôgic là không duy nhất, chẳng hạn hàm

$$f(x, y, z) = \bar{x}\bar{y} \vee x\bar{y} \vee xz \vee yz$$

có hai dạng tuyển chuẩn tắc nghẽn là

$$\bar{x}y \vee x\bar{y} \vee xz \quad \text{và} \quad \bar{x}y \vee x\bar{y} \vee yz.$$

Định lý 4. *Mọi dạng tuyển chuẩn tắc tối thiểu của một hàm đại số lôgic f đều là một dạng tuyển chuẩn tắc nghẽn của hàm đó.*

Chứng minh. Để chứng minh định lý, ta chỉ cần chứng tỏ rằng nếu p là một hội sơ cấp của một dạng tuyển chuẩn tắc tối thiểu của f, thì p phải là một nguyên nhân nguyên tố của f.

Giả sử p không phải là nguyên nhân nguyên tố của f. Khi đó, bằng cách bỏ đi một số biến (cùng với dấu phủ định nếu có), ta luôn tìm được một phần con thực sư q của p, để q là một nguyên nhân nguyên tố của f. Gọi $p \vee P$ là dạng tuyển chuẩn tắc tối thiểu của f (P là một tuyển của một số hội sơ cấp nào đó). Thay p bởi q trong tuyển này, ta nhận được $q \vee P$ cũng là một dạng tuyển chuẩn tắc của f. Điều này mâu thuẫn với tính tối thiểu của $p \vee P$, và định lý được chứng minh.

Chú ý rằng, điều ngược lại của định lý 4 là không đúng: có những dạng tuyển chuẩn tắc nghẽn mà không phải là dạng tuyển chuẩn tắc tối thiểu, thí dụ hàm

$$f(x, y, z) = (x + y) \vee (x + z)$$

có một dạng chuẩn tắc nghẽn là

$$x\,\bar{z} \vee x\,\bar{y} \vee \bar{x}y \vee \bar{x}z,$$

nhưng đó không phải là tối thiểu, một dạng như thế của f là

$$x\,\bar{z} \vee \bar{x}y \vee \bar{y}z.$$

Định lý 4 cho thấy rằng, để tìm dạng tuyển chuẩn tắc tối thiểu, ta chỉ cần xét các dạng tuyển chuẩn tắc nghẽn. Mà để tìm các dạng tuyển chuẩn tắc nghẽn thì cần biết dạng tuyển chuẩn tắc thu gọn. Đó cũng là cơ sở cho cách tìm dạng tuyển chuẩn tắc tối thiểu trình bày trong chương sau.

Bài tập

1. Cho hàm đại số lôgic $f(x, y, z)$ xác định bởi bảng:

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

a) Liệt kê tất cả các nguyên nhân của f, trong đó mỗi nguyên nhân được viết dưới dạng tuyển của một số hội sơ cấp. Số lượng của chúng là bao nhiêu (kể cả tuyển rỗng và tuyển chuẩn tắc hoàn toàn)?

b) Xác định các nguyên nhân nguyên tố trong các nguyên nhân đã tìm.

2. Cho hàm đại số lôgic n đối số. Biết dạng tuyển chuẩn tắc hoàn toàn của nó gồm k hội sơ cấp. Đếm tất cả các nguyên nhân của hàm đã cho.

3

THUẬT TOÁN TÌM DẠNG TUYỂN CHUẨN TẮC TỐI THIỂU

3.1. Chú ý mở đầu

Từ những cơ sở lý thuyết đã trình bày trong chương 2, ta có thể chia quá trình tìm dạng tuyển chuẩn tắc tối thiểu của một hàm đại số lôgic $f(x_1, ..., x_n)$ thành 2 giai đoạn:

a) Xuất phát từ dạng tuyển chuẩn tắc hoàn toàn của f, tìm dạng tuyển chuẩn tắc thu gọn của f.

b) Xuất phát từ dạng tuyển chuẩn tắc thu gọn của f, tìm các dạng tuyển chuẩn tắc nghẽn của f và lựa chọn từ các dạng này để được các dạng tuyển chuẩn tắc tối thiểu của f.

Vì dạng tuyển chuẩn tắc thu gọn của f là duy nhất, nên giai đoạn a) hoàn toàn xác định. Trái lại, giai đoạn b) cần phải xây dựng các dạng nghẽn từ dạng rút gọn, sau đó lựa chọn trong chúng để được dạng tối thiểu. Số các dạng nghẽn của một hàm đại số

lôgic có thể rất lớn, điều này làm cho việc lựa chọn trở nên khó khăn, nhiều khi không khác mấy so với việc "lựa chọn toàn bộ". Iablonski đã xây dựng được những hàm đại số lôgic $f(x_1,..., x_n)$ mà số các nguyên nhân nguyên tố của nó lớn gấp $2^{n/2}$ lần số các hội sơ cấp trong dạng tuyển chuẩn tắc hoàn toàn của f. Glagolev chứng minh rằng, đối với hầu hết các hàm đại số lôgic f của n đối số, số các dạng tuyển chuẩn tắc nghẽn của f là vượt quá $2^{2^{n(1-e)}}$, trong đó $e \to 0$ khi $n \to \infty$. Juravliev đưa ra một thí dụ đơn giản sau đây: khi $n \geq 3$, hàm

$$f(x_1,..., x_n) = [(x_1 + x_2) \vee (x_2 + x_3)] (x_1 + ... + x_n)$$

có $5^{2^{n-4}}$ dạng tuyển chuẩn tắc nghẽn.

Như vậy, về nguyên tắc, việc tìm dạng tuyển chuẩn tắc tối thiểu từ dạng tuyển chuẩn tắc thu gọn đòi hỏi phải tiến hành những sự lựa chọn phức tạp. Tuy nhiên, các phương pháp được trình bày dưới đây, mặc dù nói chung, chúng không tránh được những khó khăn và phức tạp nói trên, trong nhiều trường hợp cụ thể, chúng cũng tỏ ra có hiệu quả.

Dưới đây là một số phép toán mà ta sẽ dùng nhiều lần trong quá trình biến đổi

(i) phép nuốt sơ cấp

$$A B \vee A = A$$

(ii) phép dán

$$A x \vee A \bar{x} = A$$

(iii) phép dán không đầy đủ

$$A x \vee A \bar{x} = A \vee A x \vee A \bar{x}$$

(iv) phép dán mở rộng

$$A C \vee B \bar{C} = A C \vee B \bar{C} \vee A B$$

Trong các phép toán đó A, B và C là các biểu thức bất kỳ, x là biến.

3.2. Tìm dạng tuyển chuẩn tắc thu gọn

Mỗi hội sơ cấp hạng k

$$x_{i_1}^{\sigma_1} x_{i_2}^{\sigma_2} ... x_{i_j}^{\sigma_j}$$

được gọi là một nguyên nhân hạng k của hàm f, nếu nó là một nguyên nhân của f. Một cấu tạo đơn vị của f là một nguyên nhân hạng n của nó.

Định lý 1.

a) Một nguyên nhân hạng k của hàm f ($k < n$) là kết quả của phép dán hai nguyên nhân hạng k+1 của hàm f đó.

b) Một nguyên nhân hạng k của hàm f ($k \leq n$) là nguyên nhân nguyên tố của f nếu không thể dán được với bất kỳ một nguyên nhân hạng k nào của f.

Chứng minh.

a) Giả sử A là một nguyên nhân hạng k ($k < n$) của f. Tìm được biến x không có mặt trong A. Khi đó các hội $A\,x$ và $A\,\bar{x}$ sẽ là các nguyên nhân hạng $k+1$ của f, và A nhận được từ phép dán hai nguyên nhân này.

b) Giả sử A là một nguyên nhân hạng k ($k \leq n$) của f. Nếu A không phải là nguyên nhân nguyên tố thì luôn tìm được một biến x trong A để sau khi xoá nó (cùng với dấu phủ định nếu có) khỏi A, ta được phần con B là một nguyên nhân của f. Có thể giả thiết $A = B\,x$. Khi đó $A' = B\,\bar{x}$ cũng là một nguyên nhân hạng k của f và A có thể dán với A'.

Từ định lý 1, ta có thuật toán Quine sau đây để tìm dạng tuyển chuẩn tắc thu gọn của hàm đại số lôgic $f(x_1,..., x_n)$:

Bước 1. Tìm dạng tuyển chuẩn tắc hoàn toàn của f, ký hiệu f_0.

Bước 2. Từ f_i xây dựng f_{i+1} bằng cách trong f, thực hiện tất cả các phép dán không đầy đủ đối với các hội sơ cấp hạng n-i, sau đó xoá bỏ tất cả các hội sơ cấp hạng n-i có thể được bằng phép nuốt sơ cấp.

Bước 3. Lặp lại bước 2 cho đến khi thu được $f_{k+1} = f_k$. Khi đó f_k sẽ là dạng tuyển chuẩn tắc thu gọn của f.

Từ cách xây dựng f_i, có thể chứng minh quy nạp điều khẳng định: trongmỗi f_i, có mặt tất cả các nguyên nhân nguyên tố hạng không nhỏ hơn n-k+1 và tất cả các nguyên nhân hạng n-k của f, và chỉ có chúng.

Nếu $f_{k+1} = f_k$ thì theo phần b) của định lý 1, mọi nguyên nhân hạng n-k của f trong f_k đều là nguyên tố và như vậy f_k chứa tất cả các nguyên nhân nguyên tố của f, do đó nó là dạng tuyển chuẩn tắc thu gọn của f.

Thí dụ 1. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = x\,y\,z \vee \bar{x}\,y\,z \vee x\,\bar{y}\,z \vee x\,y\,\bar{z} \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Ta có $f_0 = f$. Dùng các phép dán không đầy đủ đối với các hội sơ cấp hạng 3, ta được

$$y\,z \vee x\,z \vee x\,y \vee x\,y\,z \vee \bar{x}\,y\,z \vee x\,\bar{y}\,z \vee x\,y\,\bar{z} \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Sau đó dùng các phép nuốt sơ cấp ta được

$$f_1 = yz \vee xz \vee xy \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Đến đây các hội sơ cấp hạng 2 không dán được với nhau, tức là $f_2 = f_1$ và f_1 là dạng tuyển chuẩn tắc thu gọn của f.

Thuật toán Quine không chỉ rõ việc tìm cho hết các phép dán có thể có. Vì thế McCluskey đề nghị bổ sung thuật toán Quine một thủ tục hình thức như trình bày dưới đây.

Giả sử f là hàm của n biến $x_1,..., x_n$. Mỗi hội sơ cấp của n biến đó được biểu diễn bằng một dãy n ký hiệu trong bảng {0, 1, -} theo quy ước: ký tự thứ i là 1 hay 0 nếu x_i có mặt trong hội là bình thường hay với dấu phủ định, còn nếu x_i không có mặt thì ký tự này là -. Chẳng hạn hội sơ cấp của 5 biến $x_1,..., x_5$: $x_1\,\bar{x}_3\,x_5$ được biểu diễn bởi 1-0-1. Hai hội sơ cấp được gọi là lân cận nhau, nếu các biểu diễn nói trên của chúng chỉ khác nhau ở một vị trí. Rõ ràng các hội sơ cấp chỉ có thể dán được với nhau nếu chúng là lân cận nhau.

Thuật toán Quine - McCluskey.

Thuật toán Quine, theo thủ tục McCluskey, được tiến hành như sau: Lập một bảng gồm nhiều cột để ghi kết quả các phép dán. Sau đó lần lượt thực hiện các bước:

Bước 1. Viết vào cột thứ nhất, các biểu diễn của các nguyên nhân hạng n (tức là các cấu tạo đơn vị) của hàm f. Các biểu diễn được chia thành từng nhóm, các biểu diễn trong mỗi nhóm có số các ký hiệu 1 bằng nhau và các nhóm xếp theo thứ tự số các ký hiệu 1 tăng dần (các nhóm được đánh số từ 1).

Bước 2. Lần lượt thực hiện tất cả các phép dán các biểu diễn trong nhóm i với các biểu diễn trong nhóm $i+1$ (i = 1, 2, ...). Biểu diễn nào tham gia ít nhất một phép dán sẽ được ghi nhận một dấu * bên cạnh. Kết quả dán được ghi vào cột tiếp theo.

Bước 3. Lặp lại bước 2 cho cột kế tiếp cho đến khi không thu thêm được cột nào mới (tức là tại cột hiện hành, không thực hiện được một phép dán nào cả). Khi đó, tất cả các biểu diễn không có dấu * sẽ cho ta tất cả các nguyên nhân nguyên tố của f.

Thí dụ 2. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = \bar{x}\,\bar{y}\,\bar{z}\,u \vee \bar{x}\,y\,\bar{z}\,u \vee \bar{x}\,\bar{y}\,z\,u \vee x\,\bar{y}\,\bar{z}\,u$$
$$\vee\, x\,\bar{y}\,z\,u \vee \bar{x}\,y\,z\,u \vee x\,y\,z\,u.$$

Thủ tục McCluskey được tiến hành bởi bảng sau đây

0001 *	0-01 *	0--1
0101 *	00-1 *	-0-1
0011 *	-001 *	--11
1001 *	-011 *	
1011 *	10-1 *	
0111 *	01-1 *	
1111 *	0-11 *	
	1-11 *	
	-111 *	

Dạng tuyển chuẩn tắc thu gọn của hàm f là

$$f = \bar{x}\,u \vee \bar{y}\,u \vee z\,u.$$

Thí dụ 3. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = \bar{x}_1\,\bar{x}_2 x_3\,\bar{x}_4 \vee \bar{x}_1\,\bar{x}_2 x_3 x_4 \vee x_1\,\bar{x}_2 x_3 x_4 \vee x_1 x_2\,\bar{x}_3\,\bar{x}_4$$
$$\vee\, x_1 x_2\,\bar{x}_3 x_4 \vee x_1 x_2 x_3\,\bar{x}_4 \vee x_1 x_2 x_3 x_4$$

Thủ tục McCluskey được tiến hành bởi bảng sau đây.

0010 *	001-	11--
0011 *	-011	
1100 *	110- *	
1011 *	11-0 *	
1101 *	1-11	
1110 *	11-1 *	
1111 *	111- *	

Dạng tuyển chuẩn tắc thu gọn của hàm f là

$$f = x_1 x_2 \vee \bar{x}_1\,\bar{x}_2 x_3 \vee \bar{x}_2 x_3 x_4 \vee x_1 x_2 x_3\,.$$

Thuật toán Quine cùng với sự cải tiến của McCluskey gọi chung là thuật toán Quine -McCluskey. Thuật toán này cho phép tìm dạng tuyển chuẩn tắc thu gọn từ dạng tuyển chuẩn tắc hoàn toàn.

Dưới đây trình bày thêm phương pháp Blake - Poreski, cho phép tìm dạng tuyển chuẩn tắc thu gọn từ một dạng tuyển chuẩn tắc tuỳ ý. Cơ sở của phương pháp này là định lý sau đây:

Định lý 2. *Nếu trong một dạng tuyển chuẩn tắc tuỳ ý của hàm đại số lôgic $f(x_1, ..., x_n)$ ta liên tiếp thực hiện tất cả các phép dán mở rộng có thể có được rồi sau đó thực hiện tất cả các phép nuốt sơ cấp, thì sẽ thu được dạng tuyển chuẩn tắc thu gọn của hàm f.*

Có thể chứng minh định lý này bằng quy nạp theo số các đối số của hàm f. Tuy nhiên chúng tôi không trình bày chi tiết ở đây.

Thí dụ 4. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = xy\bar{z} \vee zx \vee \bar{x}y.$$

Thực hiện liên tiếp các phép dán mở rộng và các phép nuốt sơ cấp ta được

$$\begin{aligned} f &= xy\bar{z} \vee zx \vee \bar{x}y \\ &= xy\bar{z} \vee zx \vee \bar{x}y \vee xy \vee y\bar{z} \vee yz \\ &= xz \vee \bar{x}y \vee xy \vee y\bar{z} \vee yz \vee y \\ &= xz \vee y. \end{aligned}$$

Vậy $xz \vee y$ là dạng tuyển chuẩn tắc thu gọn của hàm f.

Thí dụ 5. Tìm dạng tuyển chuẩn tắc thu gọn của hàm

$$f = (x + y) \vee (y + z).$$

Ta có

$$\begin{aligned} f &= \bar{x}y \vee x\bar{y} \vee \bar{y}z \vee y\bar{z} \\ &= \bar{x}y \vee x\bar{y} \vee \bar{y}z \vee y\bar{z} \vee \bar{x}z \vee x\bar{z} \end{aligned}$$

Đó là dạng tuyển chuẩn tắc thu gọn của hàm f. Thí dụ 5 chứng tỏ rằng, dạng tuyển chuẩn tắc thu gọn của một hàm f có thể "dài" hơn dạng nguyên thuỷ của nó.

3.3. Tìm dạng tuyển chuẩn tắc tối thiểu

Sau khi tìm được dạng tuyển chuẩn tắc thu gọn của f, nghĩa là tìm được tất cả các nguyên nhân nguyên tố của nó, ta tiếp tục phương pháp Quine tìm dạng tuyển chuẩn tắc tối thiểu của f như sau: Lập một bảng chữ nhật, mỗi cột ứng với một cấu tạo đơn vị của f và mỗi đòng ứng với một nguyên nhân nguyên tố của f. Tại ô (i, j), ta đánh dấu + nếu nguyên nhân nguyên tố ở dòng i là một phần con của cấu tạo đơn vị ở cột j. Ta cũng nói rằng khi đó, nguyên nhân nguyên tố i là phủ cấu tạo đơn vị j. Một hệ S các nguyên nhân nguyên tố của f được gọi là phủ hàm f, nếu mọi cấu tạo đơn vị của f đều được phủ ít

nhất bởi một thành viên thuộc hệ. Dễ thấy rằng, nếu hệ S là phủ hàm f thì nó là đầy đủ, nghĩa là tuyển của các thành viên trong S là thực hiện f.

Một nguyên nhân nguyên tố được gọi là cốt yếu, nếu thiếu nó thì một hệ các nguyên nhân nguyên tố không thể phủ hàm f. Các nguyên nhân nguyên tố cốt yếu được tìm như sau: tại những cột chỉ có duy nhất một dấu +, xem dấu + đó thuộc dòng nào thì dòng đó ứng với một nguyên nhân nguyên tố cốt yếu.

Việc lựa chọn các nguyên nhân nguyên tố trên bảng đã đánh dấu, để được một dạng tuyển chuẩn tắc tối thiểu, có thể tiến hành theo các bước sau đây:

Bước 1. Phát hiện tất cả các nguyên nhân nguyên tố cốt yếu.

Bước 2. Xoá tất cả các cột được phủ bởi các nguyên nhân nguyên tố cốt yếu, tức là tất cả các cột có ít nhất một dấu + tại những dòng ứng với các nguyên nhân nguyên tố cốt yếu.

Bước 3. Trong bảng còn lại, xoá nốt những dòng không còn dấu + và sau đó nếu có hai cột giống nhau thì xoá bớt một cột.

Bước 4. Sau các bước trên, tìm một hệ S các nguyên nhân nguyên tố với số biến ít nhất phủ tất cả các cột còn lại.

Tuyển của các nguyên nhân nguyên tố cốt yếu và các nguyên nhân trong hệ S sẽ là dạng tuyển chuẩn tắc tối thiểu của hàm f.

Các bước 1, 2, 3 có tác dụng rút gọn bảng trước khi lựa chọn. Độ phức tạp chủ yếu nằm ở bước 4. Tình huống tốt nhất là mọi nguyên nhân nguyên tố đều là cốt yếu. Trường hợp này không phải lựa chọn gì và hàm f có duy nhất một dạng tuyển chuẩn tắc tối thiểu cũng chính là dạng tuyển chuẩn tắc thu gọn. Tình huống xấu nhất là không có nguyên nhân nguyên tố nào là cốt yếu. Trường hợp này ta phải lựa chọn toàn bộ bảng. Các thí dụ sau đây minh hoạ các tình huống có thể xảy ra.

Thí dụ 1. Tìm dạng tuyển chuẩn tắc tối thiểu của hàm f cho trong thí dụ 1, mục 2. Bảng sau khi đã đánh dấu + có dạng

	xyz	$\bar{x}yz$	$x\bar{y}z$	$xy\bar{z}$	$\bar{x}\bar{y}\bar{z}$
xy	+			+	
xz	+		+		
yz	+	+			
$\bar{x}\bar{y}\bar{z}$					+

Trong trường hợp này mọi nguyên nhân nguyên tố đều là cốt yếu. Hàm f có một dạng tuyển chuẩn tắc tối thiểu, đồng thời cũng là dạng tuyển chuẩn tắc thu gọn:

$$f = xy \vee xz \vee yz \vee \bar{x}\,\bar{y}\,\bar{z}.$$

Thí dụ 2. Tìm dạng tuyển chuẩn tắc tối thiểu của hàm f cho trong thí dụ 3, mục 2. Sau khi đánh dấu + , bảng có dạng

	$\bar{x}_1\ \bar{x}_2$ $x_3\ \bar{x}_4$	$\bar{x}_1\ \bar{x}_2$ $x_3\ x_4$	$x_1\ \bar{x}_2$ $x_3\ x_4$	$x_1 x_2$ $\bar{x}_3\ \bar{x}_4$	$x_1\ x_2$ $\bar{x}_3\ x_4$	$x_1\ x_2$ $x_3\ \bar{x}_4$	$x_1\ x_2$ $x_3\ x_4$
$x_1\ x_2$				+	+	+	+
$\bar{x}_1\ \bar{x}_2 x_3$	+	+					
$\bar{x}_2\ x_3\ x_4$		+	+				
$x_1\ x_3\ x_4$			+				+

Có hai nguyên nhân nguyên tố cốt yếu nằm ở dòng 1 và 2. Sau khi rút gọn, bảng còn hai dòng 3, 4 và một cột 3. Việc chọn S khá đơn giản: có thể chọn một trong hai nguyên nhân nguyên tố còn lại. Vì vậy ta được hai dạng tuyển chuẩn tắc tối thiểu là

$$f = x_1 x_2 \vee \bar{x}_1\ \bar{x}_2 x_3 \vee \bar{x}_2 x_3 x_4,$$
$$f = x_1 x_2 \vee \bar{x}_1\ \bar{x}_2 x_3 \vee x_2 x_3 x_4.$$

Thí dụ 3. Tiếp tục thí dụ 5, mục 2. Dạng tuyển chuẩn tắc hoàn toàn của hàm f là

$$f = \bar{x}\,\bar{y}z \vee \bar{x}y\,\bar{z} \vee \bar{x}yz \vee x\,\bar{y}\,\bar{z} \vee x\,\bar{y}z \vee xy\,\bar{z}.$$

Ta lập bảng

	$\bar{x}\,\bar{y}z$	$\bar{x}y\,\bar{z}$	$\bar{x}yz$	$x\,\bar{y}\,\bar{z}$	$x\,\bar{y}z$	$xy\,\bar{z}$
$\bar{x}y$		+	+			
$x\,\bar{y}$				+	+	
$\bar{y}z$	+				+	
$y\,\bar{z}$		+				+
$\bar{x}z$	+		+			
$x\,\bar{z}$				+		+

Không có nguyên nhân nào là nguyên nhân nguyên tố cốt yếu. Trường hợp này phải lựa chọn toàn bộ. Có hai hệ phủ hàm f với số biến ít nhất, chúng tương ứng với hai dạng tuyển chuẩn tắc tối thiểu:

$$f = \bar{x}y \vee \bar{y}z \vee x\bar{z},$$

$$f = x\bar{y} \vee y\bar{z} \vee \bar{x}z.$$

Hệ $\{ \bar{x}y, x\bar{y}, \bar{y}z, y\bar{z} \}$ phủ f không thừa, nó cho ta một dạng tuyển chuẩn tắc nghẽn

$$f = \bar{x}y \vee x\bar{y} \vee \bar{y}z \vee y\bar{z}$$

nhưng không phải là tối thiểu.

3.4. Sơ đồ tối thiểu

Dạng tuyển chuẩn tắc của một hàm đại số lôgic là một loại biểu diễn đơn giản, nó tương ứng với một loại sơ đồ nào đó thực hiện hàm đã cho. Trong ứng dụng kỹ thuật, vấn đề là với một hàm đại số lôgic cho trước, làm thế nào để xây dựng một sơ đồ thực hiện được hàm đó, với ít yếu tố cơ bản nhất. Sơ đồ như vậy được gọi là sơ đồ tối thiểu của hàm đang xét. Nói chung, sơ đồ ứng với dạng tuyển chuẩn tắc tối thiểu mà ta tìm trong mục trước, chưa phải đã là sơ đồ tối thiểu.

Thí dụ. Sơ đồ thực hiện hàm f theo dạng tuyển chuẩn tắc tối thiểu

$$f = xy \vee xz \vee yz \vee \bar{x}\bar{y}\bar{z}$$

cần dùng 8 yếu tố & (hội) và $\vee$ (tuyển), trong khi đó sơ đồ theo biểu diễn

$$f = x(y \vee z) \vee yz \vee \bar{x}\bar{y}\bar{z}$$

tốn tất cả là 7 yếu tố & và $\vee$.

Vấn đề tìm sơ đồ tối thiểu (không nhất thiết là dạng tuyển chuẩn tắc) của một hàm đại số lôgic (trong hệ $\vee$, &, -) là một vấn đề phức tạp. Hiện nay chưa có một thuật toán

hữu hiệu nào để giải quyết vấn đề này. Thông thường, người ta tìm dạng tuyển chuẩn tắc tối thiểu, rồi sau đó dùng các luật phân bố để rút bớt số các chữ biến. Làm như vậy, tuy chưa hẳn đạt được một biểu diễn tối thiểu, nhưng nói chung, cũng thu được một biểu diễn cho phép xây dựng một sơ đồ tương đối tiết kiệm.

Rõ ràng là vấn đề còn trở nên phức tạp hơn nhiều, nếu xét việc biểu diễn trên một hệ đầy đủ bất kỳ. Ngoài ra, trên thực tế, ta thường gặp việc xây dựng một sơ đồ thực hiện đồng thời một hệ hàm đại số lôgic. Có thể xây dựng riêng rẽ sơ đồ tối thiểu cho từng hàm, rồi ghép chúng lại, với chú ý là có thể lợi dụng những phần chung trong chúng. Tuy nhiên, cần xét vấn đề này một cách căn bản và có hệ thống hơn.

Một vấn đề nữa là, trên thực tế, ta lại hay gặp những hàm đại số lôgic xác định không đầy đủ (do việc mã hoá nhị phân các tín hiệu rời rạc). Việc chọn những giá trị cho hàm tại những nơi không xác định, càng làm tăng thêm các khả năng được xét.

Tóm lại, đối với việc tổng hợp các sơ đồ thực hiện các hàm đại số lôgic còn rất nhiều bài toán khó khăn và lý thú. Giáo trình này chỉ đề cập đến một vài khía cạnh cơ bản nhất. Bạn đọc nào quan tâm, cần tham khảo thêm các tài liệu khác.

Bài tập

1. Tìm tất cả các nguyên nhân nguyên tố của các hàm dưới đây theo thuật toán *Quine - McCluskey*:

a) $f(x, y, z) = x y \bar{z} \vee x \bar{y} \bar{z} \vee \bar{x} y z \vee \bar{x} \bar{y} \bar{z}$

b) $f(x, y, z) = x \bar{y} z \vee x \bar{y} \bar{z} \vee \bar{x} y z \vee \bar{x} \bar{y} z \vee \bar{x} \bar{y} \bar{z}$

c) $f(x, y, z) = x y z \vee x \bar{y} z \vee \bar{x} y z \vee \bar{x} \bar{y} z \vee \bar{x} \bar{y} \bar{z}$

d) $f(x, y, z) = (x \mid y) \mid z$

e) $f(x, y, z) = x \mid (y + z)$

f) $f(x, y, z) = x \circ (y \mid z)$

g) $f(x, y, z) = (x \circ y) \mid z$

h) $f(x, y, z) = (x + y) \circ z$

2. Tìm dạng tuyển chuẩn tắc thu gọn của các hàm cho trong bài **1**.

3. Viết chương trình tìm dạng tuyển chuẩn tắc thu gọn của một hàm đại số lôgic n đối số dựa trên dạng tuyển chuẩn tắc hoàn toàn của nó.

Dữ liệu vào là file văn bản có dạng:

- dòng đầu ghi giá trị n (là số đối số của hàm)
- dòng sau ghi giá trị k (là số hội sơ cấp trong dạng tuyển chuẩn tắc hoàn toàn của hàm)
- k dòng tiếp, mỗi dòng ghi một hội sơ cấp tương ứng dưới dạng một dãy n ký tự viết liền nhau, trong đó ký tự thứ i mô tả trạng thái của đối thứ i theo quy ước: 0 nếu đối có dấu phủ định, 1 nếu đối không có dấu phủ định.

Kết quả ghi ra file văn bản có dạng (nếu có nhiều dạng tối thiểu thì chỉ ghi một):

- dòng đầu ghi giá trị m (là số hội sơ cấp trong dạng tuyển chuẩn tắc tối thiểu của hàm)
- m dòng sau, mỗi dòng ghi một hội sơ cấp tương ứng dưới dạng một dãy n ký tự viết liền nhau, trong đó ký tự thứ i mô tả trạng thái của đối thứ i theo quy ước: - nếu đối không có mặt, 0 nếu đối có dấu phủ định, 1 nếu đối không có dấu phủ định.

Thí dụ: File dữ liệu vào cho hàm (thí dụ 3, mục 3)

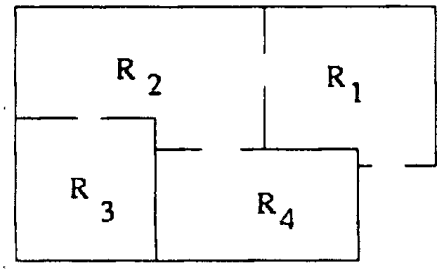
$$f(x, y, z) = \bar{x} \bar{y} z \vee \bar{x} y \bar{z} \vee \bar{x} y z \vee x \bar{y} \bar{z} \vee x \bar{y} z \vee x y \bar{z}.$$

và file kết quả tương ứng với dạng tuyển chuẩn tắc tối thiểu $\bar{x} y \vee \bar{y} z \vee x \bar{z}$ có nội dung:

File dữ liệu vào	File kết quả
3 6 001 010 011 100 101 110	3 01- -01 1-0

4. *Hệ thống cảnh báo tự động.*

Người ta cần thiết kế một mạch logic điều khiển hệ thống cảnh báo tự động trong một ngôi nhà. Trong hệ thống cảnh báo này người ta sử dụng hai dạng bộ cảm biến (sensor): tích cực và thụ động. Bộ cảm biến tích cực sẽ phát tín hiệu 1 khi mọi việc đều bình thường và tín hiệu 0 nếu phát hiện có sự cố (cửa ra vào hoặc của sổ bị mở). Bộ cảm biến thụ động sẽ phát tín hiệu 0 nếu mọi việc bình thường và tín hiệu 1 nếu phát hiện sự cố (làm chuyển động hoặc va đập vào nó). Ngôi nhà có 4 phòng R_1, R_2, R_3, R_4. Trong mỗi phòng R_i có một bộ cảm biến tích cực phát tín hiệu x_i và một bộ cảm biến thụ động phát tín hiệu y_i. Sơ đồ của ngôi nhà được mô tả trong hình 1 (các đoạn đứt là các cửa thông phòng hoặc ra vào).



Hình 1

Mạch logic điều khiển phải phát ra tín hiệu 0 khi không có sự cố nào xảy ra và phát tín hiệu 1 để kích còi báo động. Ta cần kích còi báo động nếu xảy ra ít nhất một trong các tình huống sau đây:

- Trong phòng R_1 có ít nhất một trong hai bộ cảm biến x_1 hoặc y_1 phát hiện sự cố;
- Trong bất cứ phòng R_i, $i > 1$, cả hai bộ cảm biến x_i và y_i đều phát hiện sự cố;
- Trong hai phòng có cửa thông nhau có hai bộ cảm biến phát hiện sự cố.

a) Hãy xây dựng biểu thức cho hàm biến bun $f(x_1,x_2,x_3,x_4,y_1,y_2,y_3,y_4)$ nhận giá trị 1 khi và chỉ khi cần kích còi báo động.

b) Tìm dạng tuyển chuẩn tắc hoàn toàn của f;

c) Áp dụng thuật toán Quine-McCluskey tìm dạng tuyển chuẩn tắc thu gọn của f.

Tài liệu tham khảo

1. Hall M. *Combinatorial Theory*. Blaisdell Publishing Company, London, 1967.
2. Kaufman A. *Introduction à la combinatorique en vue des applications*. Dunod, Paris, 1968.
3. Aho A.W., Hopcroft J.E., Ultman J.D. *The design and analysis of computer algorithms*. Addison - Wesley Publishing Co., Inc., 1974.
4. Reingold E.M., Nievergelt J., Deo N. *Combinatorial Algorithms. Theory and Practice*. Prentice-Hall Inc. Englewood Cliff, New Jersey, 1977.
5. Papadmitriou C.H., Steiglitz K. *Combinatorial Optimization*. Prentice Hall Inc., N. J., 1982.
6. Rubnikov K.A. *Introduction in combinatorial analysis*. Nauka, Moscow, 1972. (in Russ.)
7. Sachkov V.N. *Combinatorial Methods of Discrete Mathematics*. Nauka, Moscow, 1977. (in Russ.)
8. Conway R.W., Maxwell W.L., Miller L.W. *Theory of Scheduling*. Addison - Wesley Reading, Mass., 1967.
9. Tanaev V.C., Skurba V.V. *Introduction to Scheduling Theory*. Nauka, Moscow, 1975. (in Russ.)
10. Kovaliev M.M. *Discrete Optimization*. Izd.BGU, Minsk, 1977 (in Russ.)
11. Berg C. *Theorie des Graphes et ses Applications*. Dunod, Paris, 1958.
12. Ore O. *Theory of Graphs*. American Mathematical Society, 1962.
13. Ford L.R., Fulkerson D.R. *Flows in Networks*. Princeton Univ. Press, Princeton, N.J., 1962.
14. Busacker R.G., Saaty T.L. *Finite Graphs and Networks. An Introduction with Application*. McGraw Hill, N.Y., 1965.
15. Harary F. *Graph Theory*. Addison Wesley Publishing Academic Press, N.Y., 1973.

16. Christofides N. Graph Theory. *An algorithmic approach.* Academic Press, N.Y., 1975.

17. Iablonski S.V. *Introduction to Discrete Mathematics.* Nauka, Moscow, 1979. (in Russ.).

18. Liu C.L. *Elements of Discrete Mathematics.* McGraw - Hill Book Company, 1985.

19. Rosen K.H. *Discrete Mathematics and its Applications.* McGraw - Hill Book Company, 1991.

20. Johnsonbaugh R. *Discrete Mathematics.* Prentice Hall Inc., N. J., 1997.

21. Hoàng Tuy. *Đồ thị hữu hạn và các ứng dụng trong vận trù học*, NXB Khoa học, Hà nội, 1964.

22. Phan đình Diệu. *Lý thuyết ôtômat hữu hạn và thuật toán.* NXB ĐHTHCN, Hà nội, 1977.

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI
16 Hàng Chuối - Hai Bà Trưng - Hà Nội
Điện thoại: Biên tập -Chế bản: (04) 39714896;
Hành chính: (04) 39714899; Tổng Biên tập: (04) 39714897;
Fax: (04) 39714899

Chịu trách nhiệm xuất bản:

Giám đốc: PGS. TS. PHÙNG QUỐC BẢO

Tổng biên tập: TS. PHẠM THỊ TRÂM

Biên tập: PHẠM PHÚ TRIÊM

Biên tập tái bản: ĐỖ HỮU PHÚ

HỒ ĐỒNG

Trình bày bìa: VĂN SÁNG

Đối tác liên kết xuất bản:

HỒ ĐỒNG