

LỜI GIỚI THIỆU

Toán rời rạc là một lĩnh vực nghiên cứu và xử lý các đối tượng rời rạc dùng để đếm các đối tượng, và nghiên cứu mối quan hệ giữa các tập rời rạc. Một trong những yếu tố làm Toán rời rạc trở nên quan trọng là việc lưu trữ, xử lý thông tin trong các hệ thống máy tính về bản chất là rời rạc. Chính vì lý do đó, Toán học rời rạc là một môn học bắt buộc mang tính chất kinh điển của các ngành Công nghệ thông tin và Điện tử Viễn thông. Tài liệu hướng dẫn môn học Toán học rời rạc được xây dựng cho hệ đào tạo từ xa Học viện công nghệ Bưu chính Viễn thông được xây dựng dựa trên cơ sở kinh nghiệm giảng dạy môn học và kế thừa từ giáo trình “Toán học rời rạc ứng dụng trong tin học” của Kenneth Rossen. Tài liệu được trình bày thành hai phần:

Phần I trình bày những kiến thức cơ bản về lý thuyết tổ hợp thông qua việc giải quyết bốn bài toán cơ bản đó là: Bài toán đếm, Bài toán tồn tại, Bài toán liệt kê và Bài toán tối ưu. Phần II trình bày những kiến thức cơ bản về Lý thuyết đồ thị: khái niệm, định nghĩa, các thuật toán trên đồ thị, đồ thị Euler, đồ thị Hamilton. Một số bài toán có ứng dụng thực tiễn quan trọng khác của lý thuyết đồ thị cũng được chú trọng giải quyết đó là Bài toán tô màu đồ thị, Bài toán tìm đường đi ngắn nhất và Bài toán luồng cực đại trong mạng.

Trong mỗi phần của tài liệu, chúng tôi cố gắng trình bày ngắn gọn trực tiếp vào bản chất của vấn đề, đồng thời cài đặt hầu hết các thuật toán bằng ngôn ngữ lập trình C nhằm đạt được hai mục tiêu chính cho người học: Nâng cao tư duy toán học trong phân tích, thiết kế thuật toán và rèn luyện kỹ năng lập trình với những thuật toán phức tạp. Mặc dù đã rất cẩn trọng trong quá trình biên soạn, tuy nhiên tài liệu không tránh khỏi những thiếu sót và hạn chế. Chúng tôi rất mong được sự góp ý quý báu của tất cả độc giả và các bạn đồng nghiệp. Mọi góp ý xin gửi về: Khoa Công nghệ Thông tin – Học viện Công nghệ Bưu chính Viễn thông.

Hà nội, tháng 05 năm 2006

cuu duong than cong . com

PHẦN I. LÝ THUYẾT TỔ HỢP

CHƯƠNG 1- NHỮNG KIẾN THỨC BẢN

Nội dung chính của chương này đề cập đến những kiến thức cơ bản về logic mệnh đề và lý thuyết tập hợp. Bao gồm:

- ✓ Giới thiệu tổng quan về lý thuyết tổ hợp
- ✓ Những kiến thức cơ bản về logic.
- ✓ Những kiến thức cơ bản về lý thuyết tập hợp.
- ✓ Một số ứng dụng của logic và lý thuyết tập hợp trong tin học.

Bạn đọc có thể tìm thấy những kiến thức sâu hơn và chi tiết hơn trong các tài liệu [1] và [2] của tài liệu tham khảo.

1.1- Giới thiệu chung

Tổ hợp là một lĩnh vực quan trọng của toán học rời rạc đề cập tới nhiều vấn đề khác nhau của toán học. Lý thuyết Tổ hợp nghiên cứu việc phân bố các phần tử vào các tập hợp. Thông thường các phần tử của tập hợp là hữu hạn và việc phân bố chúng phải thoả mãn những điều kiện nhất định nào đó tùy theo yêu cầu của bài toán nghiên cứu. Mỗi cách phân bố được coi là một **“cấu hình của tổ hợp”**. Nguyên lý chúng để giải quyết bài toán tổ hợp được dựa trên những nguyên lý cơ sở đó là nguyên lý cộng, nguyên lý nhân và một số nguyên lý khác, nhưng một đặc thù không thể tách rời của toán học tổ hợp đó là việc chứng minh và kiểm chứng các phương pháp giải quyết bài toán không thể tách rời máy tính.

Những dạng bài toán quan trọng mà lý thuyết tổ hợp đề cập đó là bài toán đếm, bài toán liệt kê, bài toán tồn tại và bài toán tối ưu.

Bài toán đếm: đây là dạng bài toán nhằm trả lời câu hỏi “có bao nhiêu cấu hình thoả mãn điều kiện đã nêu?”. Bài toán đếm được áp dụng có hiệu quả vào những công việc mang tính chất đánh giá như xác suất của một sự kiện, độ phức tạp thuật toán.

Bài toán liệt kê: bài toán liệt kê quan tâm đến tất cả các cấu hình có thể có được, vì vậy lời giải của nó được biểu diễn dưới dạng thuật toán **“vét cạn”** tất cả các cấu hình. Bài toán liệt kê thường được làm nền cho nhiều bài toán khác. Hiện nay, một số bài toán tồn tại, bài toán tối ưu, bài toán đếm vẫn chưa có cách nào giải quyết ngoài phương pháp liệt kê. Phương pháp liệt kê càng trở nên quan trọng hơn khi nó được hỗ trợ bởi các hệ thống máy tính.

Bài toán tối ưu: khác với bài toán liệt kê, bài toán tối ưu chỉ quan tâm tới cấu hình **“tốt nhất”** theo một nghĩa nào đó. Đây là một bài toán có nhiều ứng dụng thực tiễn và lý thuyết tổ hợp đã đóng góp một phần đáng kể trong việc xây dựng các thuật toán để đưa ra được những mô hình tối ưu.

Bài toán tồn tại: nếu như bài toán đếm thực hiện đếm bao nhiêu cấu hình có thể có, bài toán liệt kê: liệt kê tất cả các cấu hình có thể có, bài toán tối ưu chỉ ra một cấu hình tốt nhất thì bài toán tồn tại giải quyết những vấn đề còn nghi vấn nghĩa là ngay kể cả vấn đề có hay không một cấu hình cũng chưa biết. Những bài toán này thường là những bài toán khó, việc sử dụng máy tính để chứng tỏ bài toán đó tồn tại hay không tồn tại ít nhất (hoặc không) một cấu hình càng trở nên hết sức quan trọng.

1.2. Những kiến thức cơ bản về Logic

Các qui tắc cơ bản của Logic cho ta ý nghĩa chính xác của các mệnh đề. Những qui tắc này được sử dụng giữa các lập luận toán học đúng và không đúng. Vì mục tiêu cơ bản của giáo trình này là trang

bị cho sinh viên hiểu và xây dựng được những phương pháp lập luận toán học đúng đắn, nên chúng ta sẽ bắt đầu nghiên cứu toán học rời rạc bằng những kiến thức cơ bản của môn logic học.

Hiểu được phương pháp lập luận toán học có ý nghĩa hết sức quan trọng trong tin học. Những qui tắc của logic chính là công cụ cơ sở để chúng ta có thể xây dựng nên các ngôn ngữ lập trình, các mạng máy tính, kiểm chứng tính đúng đắn của chương trình và nhiều ứng dụng quan trọng khác.

1.2.1- Định nghĩa & phép toán

Đối tượng nghiên cứu của logic học là những mệnh đề. Một mệnh đề được hiểu là một câu khẳng định hoặc đúng hoặc sai chứ không thể vừa đúng vừa sai.

Ví dụ: Những câu khẳng định sau đây là một mệnh đề:

☐ “Hà nội là thủ đô của Việt nam.”

☐ $1 + 1 = 2$

☐ $2 + 2 = 3$

Các mệnh đề “Hà nội là thủ đô của việt nam”, “ $1 + 1 = 2$ ” là những mệnh đề đúng, mệnh đề “ $2 + 2 = 3$ ” là sai. Nhưng những câu trong ví dụ sau sẽ không phải là một mệnh đề vì nó những câu đó không cho ta khẳng định đúng cũng chẳng cho ta khẳng định sai.

☐ “Bây giờ là mấy giờ?”

☐ “Hãy suy nghĩ điều này cho kỹ lưỡng”

☐ $x + 1 = 2$

☐ $x + y = z$

Ta ký hiệu những chữ cái A, B, C, D, p, q, r, s, \dots là những mệnh đề. Giá trị của một mệnh đề đúng được ký hiệu là T, giá trị mệnh đề sai được ký hiệu là F. Tập giá trị $\{T, F\}$ còn được gọi là giá trị chân lý của một mệnh đề.

Định nghĩa 1. Mệnh đề p tuyển với mệnh đề q (ký hiệu $p \vee q$) là một mệnh đề mà nó chỉ nhận giá trị T khi và chỉ khi ít nhất một trong hai mệnh đề p, q nhận giá trị T. Mệnh đề $p \vee q$ nhận giá trị F khi và chỉ khi cả p, q đều nhận giá trị F.

Định nghĩa 2. Mệnh đề p hội mệnh đề q (ký hiệu $p \wedge q$) là một mệnh đề mà nó chỉ nhận giá trị T khi và chỉ khi p, q nhận giá trị T. Mệnh đề $p \wedge q$ nhận giá trị F khi và chỉ khi hoặc p, q , hoặc cả hai nhận giá trị F.

Định nghĩa 3. Phủ định mệnh đề p (ký hiệu $\neg p$) là một mệnh đề nhận giá trị F khi và chỉ khi mệnh đề p nhận giá trị T, nhận giá trị F khi và chỉ khi p nhận giá trị T.

Định nghĩa 4. Mệnh đề tuyển loại của p và q , được ký hiệu là $p \oplus q$, là một mệnh đề chỉ đúng khi một trong p hoặc q là đúng và sai trong các trường hợp khác còn lại.

Định nghĩa 5. Mệnh đề p suy ra mệnh đề q (ký hiệu $p \rightarrow q$) nhận giá T khi và chỉ khi p nhận giá trị F hoặc p và q cùng nhận giá trị T. Mệnh đề $p \rightarrow q$ nhận giá trị F khi và chỉ khi p nhận giá trị T và q nhận giá trị F.

Định nghĩa 6. Hai mệnh đề p, q được gọi là kéo theo nhau (ký hiệu : $p \Leftrightarrow q$) có giá trị đúng khi p và q có cùng giá trị chân lý và sai trong các trường hợp khác còn lại.

Các phép toán : $\vee, \wedge, \neg, \oplus, \rightarrow, \Leftrightarrow$ có thể được định nghĩa thông qua bảng giá trị chân lý sau:

Bảng 1.1: Bảng giá trị chân lý của các phép toán $\vee, \wedge, \neg, \oplus, \rightarrow, \Leftrightarrow$

p	q	$p \vee q$	$p \wedge q$	$\neg p$	$p \oplus q$	$p \rightarrow q$	$p \Leftrightarrow q$
T	T	T	T	F	F	T	T
T	F	T	F	F	T	F	F
F	T	T	F	T	T	T	F
F	F	F	F	T	F	T	T

1.2.2- Sự tương đương giữa các mệnh đề

Một vấn đề hết sức quan trọng trong lập luận toán học là việc thay thế này bằng một mệnh đề khác có cùng giá trị chân lý. Hai mệnh đề có cùng một giá trị chân lý chúng ta có thể hiểu theo cách thông thường là chúng tương đương nhau về ngữ nghĩa. Do vậy, ta sẽ tiếp cận và phân loại các mệnh đề phức hợp thông qua các giá trị chân lý của chúng.

Định nghĩa 1. Một mệnh đề phức hợp mà luôn luôn đúng với bất kể các giá trị chân lý của các mệnh đề thành phần của nó được gọi là hằng đúng (tautology). Một mệnh đề luôn luôn sai với mọi giá trị chân lý của các mệnh đề thành phần của nó được gọi là mâu thuẫn.

Ví dụ: mệnh đề phức hợp $p \vee \neg q$ là hằng đúng, $p \wedge \neg q$ là mâu thuẫn vì giá trị chân lý của các mệnh đề trên luôn luôn đúng, hoặc luôn luôn sai như được chỉ ra trong bảng 1.2.

Bảng 1.2. Ví dụ về mệnh đề hằng đúng & mệnh đề mâu thuẫn

p	$\neg p$	$p \vee \neg q$	$p \wedge \neg q$
T	F	T	F
F	T	T	F

Định nghĩa 2. Hai mệnh đề p, q được gọi là tương đương logic với nhau (ký hiệu : $p \equiv q$) khi và chỉ khi các cột cho giá trị chân lý của chúng giống nhau. Hay mệnh đề $p \rightarrow q$ là hằng đúng.

Ví dụ: hai mệnh đề $\neg(p \vee q)$ và $\neg p \wedge \neg q$ là tương đương logic vì các cột giá trị chân lý của chúng được thể hiện qua bảng sau:

Bảng 1.3. Bảng giá trị chân lý đối với $\neg(p \vee q)$ và $\neg p \wedge \neg q$

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

Dùng bảng giá trị chân lý để chứng minh tính tương đương logic giữa hai mệnh đề phức hợp cho ta một phương pháp trực quan dễ hiểu. Tuy nhiên, với những mệnh đề logic phức hợp có k mệnh đề thì cần tới 2^k giá trị chân lý để biểu diễn bảng giá trị chân lý. Trong nhiều trường hợp chúng ta có thể

chứng minh tính tương logic bằng việc thay thế một mệnh đề phức hợp bằng những tương đương logic có trước.

Bằng phương pháp bảng chân lý, dễ dàng chứng minh được sự tương đương của các công thức dưới đây:

$$p \rightarrow q \equiv \neg p \vee q$$

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$

$$\neg(\neg p) \equiv p$$

Bảng 1.4. Bảng các tương đương logic	
TƯƠNG ĐƯƠNG	TÊN GỌI
$p \wedge T \equiv p$ $p \wedge F \equiv p$	Luật đồng nhất
$p \vee T \equiv T$ $p \wedge F \equiv F$	Luật nuốt
$p \vee p \equiv p$ $p \wedge p \equiv p$	Luật lũy đẳng
$\neg(\neg p) \equiv p$	Luật phủ định kép
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Luật giao hoán
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Luật kết hợp
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Luật phân phối
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	Luật De Morgan

Ví dụ: Chứng minh rằng $\neg(p \wedge (\neg q \wedge q))$ là tương đương logic với $\neg p \wedge \neg q$.

Chứng minh:

$$\neg(p \wedge (\neg q \wedge q)) \equiv \neg p \wedge \neg(\neg p \wedge q)$$

$$\equiv \neg p \wedge [\neg(\neg p) \vee \neg q]$$

$$\equiv \neg p \wedge [p \vee \neg q]$$

$$\equiv (\neg p \wedge p) \vee (\neg p \wedge \neg q)$$

$$\equiv F \vee (\neg p \wedge \neg q)$$

$$\equiv \neg p \wedge \neg q$$

theo luật De Morgan thứ 2

theo luật De Morgan thứ 2

theo luật phủ định kép

theo luật phân phối

vì $\neg p \wedge p \equiv F$

Mệnh đề được chứng minh.

1.2.3. Dạng chuẩn tắc

Các công thức (mệnh đề) tương đương được xem như các biểu diễn khác nhau của cùng một mệnh đề. Để dễ dàng viết các chương trình máy tính thao tác trên các công thức, chúng ta cần chuẩn hóa các công thức, đưa chúng về dạng biểu diễn chuẩn được gọi là **dạng chuẩn hội**. Một công thức được gọi là ở dạng chuẩn hội nếu nó là hội của các mệnh đề tuyến.

Phương pháp để biến đổi một công thức bất kỳ về dạng chuẩn hội bằng cách áp dụng các thủ tục sau:

- ✓ Bỏ các phép kéo theo (\rightarrow) bằng cách thay $(p \rightarrow q)$ bởi $(\neg p \vee q)$.
- ✓ Chuyển các phép phủ định (\neg) vào sát các ký hiệu mệnh đề bằng cách áp dụng luật De Morgan và thay $\neg(\neg p)$ bởi p .
- ✓ Áp dụng luật phân phối thay các công thức có dạng $(p \vee (q \wedge r))$ bởi $(p \vee q) \wedge (p \vee r)$.

Ví dụ. Ta chuẩn hóa công thức $(p \rightarrow q) \vee \neg(r \vee \neg s)$:

$$\begin{aligned}(p \rightarrow q) \vee \neg(r \vee \neg s) &\equiv (\neg p \vee q) \vee (\neg r \wedge s) \\ &\equiv ((\neg p \vee q) \vee \neg r) \wedge ((\neg p \vee q) \vee s) \\ &\equiv (\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee s)\end{aligned}$$

Như vậy công thức $(p \rightarrow q) \vee \neg(r \vee \neg s)$ được đưa về dạng chuẩn hội $(\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee s)$

1.3- Vị từ và lượng từ

Trong toán học hay trong các chương trình máy tính chúng ta rất hay gặp những khẳng định chưa phải là một mệnh đề. Những khẳng định đó đều có liên quan đến các biến. Chẳng hạn khẳng định:

$P(x) = "x > 3"$ không phải là một mệnh đề nhưng tại những giá trị cụ thể của $x = x_0$ nào đó thì $P(x_0)$ lại là một mệnh đề. Hoặc trong những đoạn chương trình gặp câu lệnh:

if ($x > 3$) then $x := x + 1$;

thì chương trình sẽ đặt giá trị cụ thể của biến x vào $P(x)$, nếu mệnh đề $P(x)$ cho giá trị đúng x sẽ được tăng lên 1 bởi câu lệnh $x := x + 1$, $P(x)$ có giá trị sai giá trị của x được giữ nguyên sau khi thực hiện câu lệnh if.

Chúng ta có thể phân tích mỗi khẳng định thành hai phần chủ ngữ và vị ngữ (hay vị từ), trong câu “ x lớn hơn 3” ta có thể coi x là chủ ngữ, “lớn hơn 3” là vị ngữ, hàm $P(x)$ được gọi là hàm mệnh đề. Một hàm mệnh đề có thể có một hoặc nhiều biến, giá trị chân lý của hàm mệnh đề tại những giá trị cụ thể của biến được xác định như những mệnh đề thông thường.

Ví dụ: Cho $Q(x, y, z)$ là hàm mệnh đề xác định câu $x^2 = y^2 + z^2$ hãy xác định giá trị chân lý của các mệnh đề $Q(3, 2, 1)$, $Q(5, 4, 3)$.

Giải:

Đặt giá trị cụ thể của x, y, z vào $Q(x, y, z)$ ta có :

$Q(3, 2, 1)$ là mệnh đề “ $3^2 = 2^2 + 1^2$ ” là sai do đó $Q(3, 2, 1)$ là mệnh đề sai. Trong đó, $Q(5, 4, 3)$ là mệnh đề “ $5^2 = 4^2 + 3^2$ ” đúng, do đó $Q(5, 4, 3)$ là mệnh đề đúng.

Tổng quát, giả sử M là một tập hợp các phần tử nào đó. M thường được gọi là trường hay miền xác định của các phần tử thuộc M . Khi đó, biểu thức $P(x)$ gọi là vị từ xác định trên trường M nếu khi thay x bởi một phần tử bất kỳ của trường M thì $P(x)$ sẽ trở thành một mệnh đề trên trường M .

Khi tất cả các biến của hàm mệnh đề đều được gán những giá trị cụ thể, thì mệnh đề tạo ra sẽ xác định giá trị chân lý. Tuy nhiên, có một phương pháp quan trọng khác để biến một hàm mệnh đề thành một mệnh đề mà không cần phải kiểm chứng mọi giá trị chân lý của hàm mệnh đề tương ứng với các giá trị của biến thuộc trường đang xét. Phương pháp đó gọi là sự lượng hoá hay lượng từ. Chúng ta xét hai lượng từ quan trọng là lượng từ với mọi (ký hiệu : \forall), lượng từ tồn tại (ký hiệu : \exists).

Định nghĩa 1. Lượng từ với mọi của $P(x)$ ký hiệu là $\forall x P(x)$ là một mệnh đề “ $P(x)$ đúng với mọi phần tử x thuộc trường đang xét”.

Ví dụ : Cho hàm mệnh đề $P(x) = X^2 + X + 41$ là nguyên tố. Xác định giá trị chân lý của mệnh đề $\forall P(x)$ với x thuộc không gian bao gồm các số tự nhiên $[0..39]$.

Giải: vì $P(x)$ đúng với mọi giá trị của $x \in [0..39] \Rightarrow \forall P(x)$ là đúng.

Ví dụ : Cho $P(x)$ là hàm mệnh đề “ $x + 1 > x$ ”. Xác định giá trị chân lý của mệnh đề $\forall x P(x)$, trong không gian các số thực.

Giải : vì $P(x)$ đúng với mọi số thực x nên $\forall x P(x)$ là đúng.

Định nghĩa 2. Lượng từ tồn tại của hàm mệnh đề $P(x)$ (được ký hiệu là: $\exists x P(x)$) là một mệnh đề “ Tồn tại một phần tử x trong không gian sao cho $P(x)$ là đúng “.

Ví dụ: Cho $P(x)$ là hàm mệnh đề “ $x > 3$ ”. Hãy tìm giá trị chân lý của mệnh đề $\exists x P(x)$ trong không gian các số thực.

Giải: vì $P(4)$ là “ $4 > 3$ ” đúng nên $\exists x P(x)$ là đúng.

Ví dụ: Cho $Q(x)$ là “ $x + 1 > x$ ”. Hãy tìm giá trị chân lý của mệnh đề $\exists x Q(x)$ trong không gian các số thực.

Giải: vì $Q(x)$ sai với mọi $x \in \mathbb{R}$ nên mệnh đề $\exists x Q(x)$ là sai.

Bảng 1.5: Giá trị chân lý của lượng từ \forall, \exists		
$\forall x P(x)$	$P(x)$ đúng với mọi x	Có một giá trị của x để $P(x)$ sai
$\exists x P(x)$	Có một giá trị của x để $P(x)$ đúng	$P(x)$ sai với mọi x

Dịch những câu thông thường thành biểu thức logic: Dịch một câu được phát biểu bằng ngôn ngữ tự nhiên (câu hỏi thông thường) thành một biểu thức logic có vai trò hết sức quan trọng trong xây dựng các ngôn ngữ lập trình, chương trình dịch và xử lý ngôn ngữ tự nhiên. Quá trình dịch một câu từ ngôn ngữ tự nhiên thành một biểu thức sẽ làm mất đi tính tự nhiên của ngôn ngữ vì đa số các ngôn ngữ đều không rõ ràng, nhưng một biểu thức logic lại rất rõ ràng chặt chẽ từ cú pháp thể hiện đến ngữ nghĩa của câu. Điều này dẫn đến phải có một tập hợp các giả thiết hợp lý dựa trên một hàm xác định ngữ nghĩa của câu đó. Một khi câu đã được chuyển dịch thành biểu thức logic, chúng ta có thể xác định được giá trị chân lý của biểu thức logic, thao tác trên biểu thức logic, biến đổi tương đương trên biểu thức logic.

Chúng ta sẽ minh hoạ việc dịch một câu thông thường thành biểu thức logic thông qua những sau.

Ví dụ dịch câu “Bạn không được lái xe máy nếu bạn cao dưới 1.5 mét trừ phi bạn trên 18 tuổi” thành biểu thức logic.

Giải:

Ta gọi p là câu : Bạn được lái xe máy.

q là câu : Bạn cao dưới 1.5m.

r là câu : Bạn trên 18 tuổi.

Khi đó: Câu hỏi trên được dịch là: $(q \wedge \neg r) \rightarrow \neg p$

Ví dụ: Dịch câu “ Tất cả các sinh viên học tin học đều học môn toán học rời rạc”

Giải: Gọi $P(x)$ là câu “x cần học môn toán học rời rạc” và x được xác định trong không gian của các sinh viên học tin học. Khi đó chúng ta có thể phát biểu: $\forall x P(x)$

Ví dụ: Dịch câu “Có một sinh viên ở lớp này ít nhất đã ở tất cả các phòng của ít nhất một nhà trọ ký túc xá”.

Giải : Gọi tập sinh viên trong lớp là không gian xác định sinh viên x, tập các nhà trọ ký túc xá là không gian xác định căn nhà y, tập các phòng là không gian xác định phòng z. Ta gọi $P(z,y)$ là “ z thuộc y”, $Q(x,z)$ là “ x đã ở z”. Khi đó ta có thể phát biểu :

$$\exists x \exists y \forall z (P(z,y) \rightarrow Q(x,z));$$

1.4. Một số ứng dụng trên máy tính

Các phép toán bit: Các hệ thống máy tính thường dùng các bit (binary digit) để biểu diễn thông tin. Một bit có hai giá trị chân lý hoặc 0 hoặc 1. Vì giá trị chân lý của một biểu thức logic cũng có hai giá trị hoặc đúng (T) hoặc sai (F). Nếu ta coi giá trị đúng có giá trị 1 và giá trị sai là 0 thì các phép toán với các bit trong máy tính được tương ứng với các liên từ logic.

Một chuỗi bit (hoặc chuỗi nhị phân) là dãy không hoặc nhiều bit. Chiều dài của chuỗi là số các bit trong chuỗi đó.

Ví dụ:

Chuỗi nhị 101010011 có độ dài là 9.

Một số nguyên được biểu diễn như một chuỗi nhị phân có độ dài 16 bit.

Các phép toán với bit được xây dựng trên các chuỗi bit có cùng độ dài, bao gồm : AND bit (phép và cấp bit), OR (phép hoặc cấp bit), XOR (phép tuyển loại trừ cấp bit). Ví dụ: cho hai chuỗi bit 01101 10110 và 11000 11101 hãy tìm chuỗi AND bit, OR bit, XOR bit.

Phép AND

01101 10110

11000 11101

01000 10100

Phép OR

01101 10110

11000 11101

11101 11111

Phép XOR

01101 10110

11000 11101

10101 01011

Thuật toán các phép tính số nguyên: Các thuật toán thực hiện các phép tính với các số nguyên khi dùng khai triển nhị phân là hết sức quan trọng trong bộ xử lý số học của máy tính. Như chúng ta đã biết, thực chất các số nguyên được biểu diễn trong máy tính là các chuỗi bit nhị phân, do vậy chúng ta có thể sử dụng biểu diễn nhị phân của các số để thực hiện các phép tính.

Giả sử khai triển nhị phân của các số nguyên a và b tương ứng là:

$a = (a_{n-1}a_{n-2} \dots a_1a_0)_2$, $b = (b_{n-1}b_{n-2} \dots b_1b_0)_2$. Khai triển của a và b có đúng n bit (chấp nhận những bit 0 ở đầu để làm đặc n bit).

Xét bài toán cộng hai số nguyên viết ở dạng nhị phân. Thủ tục thực hiện việc cộng cũng giống như làm trên giấy thông thường. Phương pháp này tiến hành bằng cách cộng các bit nhị phân tương ứng có nhớ để tính tổng hai số nguyên. Sau đây là mô tả chi tiết cho quá trình cộng hai chuỗi bit nhị phân.

Để cộng a với b , trước hết ta cộng hai bit phải nhất, nghĩa là:

$a_0 + b_0 = c_0 * 2 + s_0$; trong đó s_0 là bit phải nhất của số nguyên tổng $a + b$, c_0 là số cần để nhớ nó có thể bằng 0 hoặc 1. Sau đó ta cộng hai bit tiếp theo và số nhớ:

$a_1 + b_1 + c_0 = c_1 * 2 + s_1$; s_1 là bit tiếp theo của số $a + b$, c_1 là số nhớ. Tiếp tục quá trình này bằng cách cộng các bit tương ứng trong khai triển nhị phân và số nhớ, ở giai đoạn cuối cùng: $a_{n-1} + b_{n-1} + c_{n-2} = c_{n-1} * 2 + s_{n-1}$. Bit cuối cùng của tổng là c_{n-1} . Khi đó khai triển nhị phân của tổng $a + b$ là $(s_n a_{n-1} \dots s_1 s_0)_2$.

Ví dụ: cộng $a = (1110)_2$, $b = (1011)_2$

Giải:

Trước hết lấy:

$$a_0 + b_0 = 0 + 1 = 0 * 2 + 1 \Rightarrow c_0 = 0, s_0 = 1$$

Tiếp tục:

$$a_1 + b_1 + c_0 = 1 + 1 + 0 = 1 * 2 + 0 \Rightarrow c_1 = 1, s_1 = 0$$

$$a_2 + b_2 + c_1 = 1 + 0 + 1 = 1 * 2 + 0 \Rightarrow c_2 = 1, s_2 = 0$$

$$a_3 + b_3 + c_2 = 1 + 1 + 1 = 1 * 2 + 1 \Rightarrow c_3 = 1, s_3 = 1$$

Cuối cùng:

$$s_4 = c_3 = 1 \Rightarrow a + b = (11001)_2$$

Thuật toán cộng:

void Cong(a, b: positive integer)

{

*/** $a = (a_{n-1}a_{n-2} \dots a_1a_0)_2$, $b = (b_{n-1}b_{n-2} \dots b_1b_0)_2$ **/*

$c = 0$;

for ($j = 0$; $j \leq n-1$; $j++$) {

$d = [(a_j + b_j + c) / 2]$;

$s_j = a_j + b_j + c - 2d$;

```

        c = d;
    }
    sn = c;
    khai triển nhị phân của tổng là (snan-1 . . . s1s0)2;
}

```

Thuật toán nhân: Để nhân hai số nguyên n bit a, b ta bắt đầu từ việc phân tích:

$$a = (a_{n-1}a_{n-2} \dots a_1a_0), b = (b_{n-1}b_{n-2} \dots b_1b_0)$$

$$\Rightarrow ab = a \sum_{j=0}^{n-1} b_j 2^j = \sum_{j=0}^{n-1} a(b_j 2^j)$$

Ta có thể tính a.b từ phương trình trên. Trước hết, ta nhận thấy $ab_j = a$ nếu $b_j=1$, $ab_j=0$ nếu $b_j=0$. Mỗi lần tính ta nhân với 2^j hay dịch chuyển sang trái j bit 0 bằng cách thêm j bit 0 vào bên trái kết quả nhận được. Cuối cùng, cộng n số nguyên $ab_j 2^j$ ($j=0..n-1$) ta nhận được a.b. Ví dụ sau đây sẽ minh họa cho thuật toán nhân:

Ví dụ: Tìm tích của $a = (110)_2$, $b = (101)_2$

Giải: Ta nhận thấy

$$ab_0 2^0 = (110)_2 * 1 * 2^0 = (110)_2$$

$$ab_1 2^1 = (110)_2 * 0 * 2^1 = (0000)_2$$

$$ab_2 2^2 = (110)_2 * 1 * 2^2 = (11000)_2$$

Sử dụng thuật toán tính tổng hai số nguyên a, b có biểu diễn n bit ta nhận được (ta có thể thêm số 0 vào đầu mỗi toán hạng):

$$(0 \ 110)_2 + (0000)_2 = (0110)_2 ;$$

$$(0 \ 0110)_2 + (11000)_2 = (11110)_2 = ab.$$

Thuật toán nhân hai số nguyên n bit có thể được mô phỏng như sau:

void Nhan(a, b : Positive integer){

/ khai triển nhị phân tương ứng của a = (a_{n-1}a_{n-2} . . . a₁a₀),*

*b = (b_{n-1}b_{n-2} . . . b₁b₀) */*

for (j=0; j≤n-1; j++) {

if ((b_j=1)

*c_j = a * 2^j; /* a được dịch trái j bit 0 */*

else c_j = 0;

}

*/*c₀, c₁.., c_{n-1} là những tích riêng của ab_j 2^j(j=0..n-1) */*

p=0;

for (j=0 ; j≤ n-1; j++)

p= p + c_j;

/ p là giá trị của tích ab */*

}

1.5. Những kiến thức cơ bản về lý thuyết tập hợp

1.5.1- Khái niệm & định nghĩa

Các tập hợp dùng để nhóm các đối tượng lại với nhau. Thông thường, các đối tượng trong tập hợp có các tính chất tương tự nhau. Ví dụ, tất cả sinh viên mới nhập trường tạo nên một tập hợp, tất cả sinh viên thuộc khoa Công nghệ thông tin là một tập hợp, các số tự nhiên, các số thực . . . cũng tạo nên các tập hợp. Chú ý rằng, thuật ngữ đối tượng được dùng ở đây không chỉ rõ cụ thể một đối tượng nào, sự mô tả một tập hợp nào đó hoàn toàn mang tính trực giác về các đối tượng.

Định nghĩa 1. Tập các đối tượng trong một tập hợp được gọi là các phần tử của tập hợp. Các tập hợp thường được ký hiệu bởi những chữ cái in hoa đậm như A, B, X, Y . . ., các phần tử thuộc tập hợp hay được ký hiệu bởi các chữ cái in thường như a, b, c, u, v . . . Để chỉ a là phần tử của tập hợp A ta viết $a \in A$, trái lại nếu a không thuộc A ta viết $a \notin A$.

Tập hợp không chứa bất kỳ một phần tử nào được gọi là tập rỗng (kí hiệu là ϕ hoặc $\{ \}$)

Tập hợp A được gọi là bằng tập hợp B khi và chỉ khi chúng có cùng chung các phần tử và được kí hiệu là $A=B$. Ví dụ tập $A=\{ 1, 3, 5 \}$ sẽ bằng tập $B = \{ 3, 5, 1 \}$.

Định nghĩa 2. Tập A được gọi là một tập con của tập hợp B và ký hiệu là $A \subseteq B$ khi và chỉ khi mỗi phần tử của A là một phần tử của B. Hay $A \subseteq B$ khi và chỉ khi lượng từ

$$\forall x (x \in A \rightarrow x \in B) \text{ cho ta giá trị đúng.}$$

Từ định nghĩa trên chúng ta rút ra một số hệ quả sau:

- ☐ Tập rỗng ϕ là tập con của mọi tập hợp.
- ☐ Mọi tập hợp là tập con của chính nó.
- ☐ Nếu $A \subseteq B$ và $B \subseteq A$ thì $A=B$ hay mệnh đề :

$$x (x \in A \rightarrow x \in B) \vee \forall x (x \in B \rightarrow x \in A) \text{ cho ta giá trị đúng.}$$

- ☐ Nếu $A \subseteq B$ và $A \neq B$ thì ta nói A là tập con thực sự của B và ký hiệu là $A \subset B$.

Định nghĩa 3. Cho S là một tập hợp. Nếu S có chính xác n phần tử phân biệt trong S, với n là số nguyên không âm thì ta nói S là một tập hữu hạn và n được gọi là bản số của S. Bản số của S được ký hiệu là $|S|$.

Định nghĩa 4. Cho tập hợp S. Tập lũy thừa của S ký hiệu là $P(S)$ là tập tất cả các tập con của S.

$$\text{Ví dụ } S = \{ 0, 1, 2 \} \Rightarrow P(S) = \{ \phi, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\} \}.$$

Định nghĩa 5. Dãy sắp thứ tự (a_1, a_2, \dots, a_n) là một tập hợp sắp thứ tự có a_1 là phần tử thứ nhất, a_2 là phần tử thứ 2, ..., a_n là phần tử thứ n.

Chúng ta nói hai dãy sắp thứ tự là bằng nhau khi và chỉ khi các phần tử tương ứng của chúng là bằng nhau. Nói cách khác (a_1, a_2, \dots, a_n) bằng (b_1, b_2, \dots, b_n) khi và chỉ khi $a_i = b_i$ với mọi $i = 1, 2, \dots, n$.

Định nghĩa 6. Cho A và B là hai tập hợp. Tích đề các của A và B được ký hiệu là $A \times B$, là tập hợp của tất cả các cặp (a,b) với $a \in A, b \in B$. Hay có thể biểu diễn bằng biểu thức:

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

Định nghĩa 7. Tích đề các của các tập A_1, A_2, \dots, A_n được ký hiệu là $A_1 \times A_2 \times \dots \times A_n$ là tập hợp của dãy sắp thứ tự (a_1, a_2, \dots, a_n) trong đó $a_i \in A_i$ với $i = 1, 2, \dots, n$. Nói cách khác:

$$A_1 \times A_2 \times \dots \times A_n = \{ (a_1, a_2, \dots, a_n) \mid a_i \in A_i \text{ với } i = 1, 2, \dots, n \}$$

1.5.2. Các phép toán trên tập hợp

Các tập hợp có thể được tổ hợp với nhau theo nhiều cách khác nhau thông qua các phép toán trên tập hợp. Các phép toán trên tập hợp bao gồm: Phép hợp (Union), phép giao (Intersection), phép trừ (Minus).

Định nghĩa 1. Cho A và B là hai tập hợp. Hợp của A và B được ký hiệu là $A \cup B$, là tập chứa tất cả các phần tử hoặc thuộc tập hợp A hoặc thuộc tập hợp B. Nói cách khác:

$$A \cup B = \{ x \mid x \in A \vee x \in B \}$$

Định nghĩa 2. Cho A và B là hai tập hợp. Giao của A và B được ký hiệu là $A \cap B$, là tập chứa tất cả các phần tử thuộc A và thuộc B. Nói cách khác:

$$A \cap B = \{ x \mid x \in A \wedge x \in B \}$$

Định nghĩa 3. Hai tập hợp A và B được gọi là rời nhau nếu giao của chúng là tập rỗng ($A \cap B = \emptyset$).

Định nghĩa 4. Cho A và B là hai tập hợp. Hiệu của A và B là tập hợp được ký hiệu là $A - B$, có các phần tử thuộc tập hợp A nhưng không thuộc tập hợp B. Hiệu của A và B còn được gọi là phần bù của B đối với A. Nói cách khác:

$$A - B = \{ x \mid x \in A \wedge x \notin B \}$$

Định nghĩa 5. Cho tập hợp A. Ta gọi \overline{A} là phần bù của A là một tập hợp bao gồm những phần tử không thuộc A. Hay :

$$\overline{A} = \{ x \mid x \notin A \}$$

Định nghĩa 6. Cho các tập hợp A_1, A_2, \dots, A_n . Hợp của các tập hợp là tập hợp chứa tất cả các phần tử thuộc ít nhất một trong số các tập hợp A_i ($i=1, 2, \dots, n$). Ký hiệu:

$$\bigcup_{i=1}^n A_i = A_1 \cup A_2 \cup \dots \cup A_n$$

Định nghĩa 7: Cho các tập hợp A_1, A_2, \dots, A_n . Giao của các tập hợp là tập hợp chứa các phần tử thuộc tất cả n tập hợp A_i ($i=1, 2, \dots, n$).

$$\bigcap_{i=1}^n A_i = A_1 \cap A_2 \cap \dots \cap A_n$$

1.5.3. Các hằng đẳng thức trên tập hợp

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

- ☐ Phủ định của A, ký hiệu \overline{A} (hay NOT A) tương ứng với phần bù \overline{A}
- ☐ Tuyến 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 $A \wedge B$ (hay A and B) tương ứng với $A \cap B$

Các mệnh đề cùng với các phép toán trên nó lập thành một đại số mệnh đề (hay đại số logic). Như thế, đại số tập hợp và đại số logic là hai đại số đẳng cấu với nhau (những mệnh đề phát biểu trên đại số logic tương đương với mệnh đề phát biểu trên đại số tập hợp). Với những trường hợp cụ thể, tùy theo tình huống, một bài toán có thể được phát biểu bằng ngôn ngữ của đại số logic hay ngôn ngữ của đại số tập hợp. Bảng 1.5 thể hiện một số hằng đẳng thức của đại số tập hợp.

Ta gọi U là tập hợp vũ trụ hay tập hợp của tất cả các tập hợp.

Bảng 1.5: Một số hằng đẳng thức trên tập hợp	
HÀNG ĐẲNG THỨC	TÊN GỌI
$A \cup \phi = A$ $A \cap U = A$ (U là tập vũ trụ)	Luật đồng nhất
$A \cup U = U$ $A \cap \phi = A$	Luật nuốt
$A \cap A = A$ $A \cup A = A$	Luật lũy đẳng
$\overline{\overline{A}} = A$	Luật bù
$A \cap B = B \cap A$ $A \cup B = B \cup A$	Luật giao hoán
$A \cup (B \cap C) = (A \cup B) \cap C$ $A \cap (B \cup C) = (A \cap B) \cup C$	Luật kết hợp
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Luật phân phối
$\overline{A \cup B} = \overline{A} \cap \overline{B}$ $\overline{A \cap B} = \overline{A} \cup \overline{B}$	Luật De Morgan

1.6- Biểu diễn tập hợp trên máy tính

Có nhiều cách khác nhau để biểu diễn tập hợp trên máy tính, phương pháp phổ biến là lưu trữ các phần tử của tập hợp không sắp thứ tự. Với việc lưu trữ bằng phương pháp này, ngoài những lãng phí bộ nhớ không cần thiết, thì quá trình tính hợp, giao, hiệu các tập hợp gặp nhiều khó khăn và mất nhiều thời gian vì mỗi phép tính đòi hỏi nhiều thao tác tìm kiếm trên các phần tử. Một phương pháp lưu trữ các phần tử bằng cách biểu diễn có thứ tự của các phần tử của một tập vũ trụ tỏ ra hiệu quả hơn rất nhiều trong quá trình tính toán.

Giả sử tập vũ trụ U là hữu hạn gồm n phần tử (hữu hạn được hiểu theo nghĩa các phần tử của U lưu trữ được trong bộ nhớ máy tính). Giả sử ta muốn biểu diễn tập hợp $A \subseteq U$. Trước hết ta chọn một thứ tự tùy ý nào đó đối với các phần tử của tập vũ trụ U, giả sử ta được bộ có thứ tự a_1, a_2, \dots, a_n .

Sau đó xây dựng một chuỗi nhị phân có độ dài n , sao cho nếu bit thứ i có giá trị 1 thì phần tử $a_i \in A$, nếu $a_i = 0$ thì $a_i \notin A$ ($i=1,2,\dots,n$). Ví dụ sau sẽ minh họa kỹ thuật biểu diễn tập hợp bằng chuỗi nhị phân.

Ví dụ: Giả sử $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Hãy biểu diễn tập hợp $A \subseteq U$ là

- 1- Tập các số nguyên lẻ $A \subseteq U$.
- 2- Tập các số nguyên chẵn $B \subseteq U$.
- 3- Tập các số nguyên nhỏ hơn 5 $C \subseteq U$.
- 4- Tìm $A \cup B$
- 5- Tìm $A \cap C \dots$

Giải : Trước hết ta coi thứ tự các phần tử được sắp xếp theo thứ tự tăng dần tức $a_i = i$ ($i=1,2,\dots,10$). Khi đó :

1- Chuỗi bit biểu diễn các số lẻ trong U ($\{1, 3, 5, 7, 9\}$) là chuỗi có độ dài $n = 10$ trong đó các bit ở vị trí thứ 1, 3, 5, 7, 9 có giá trị là 1, các bit còn lại có giá trị là 0. Từ đó ta có chuỗi bit biểu diễn tập hợp A là: 1 0 1 0 1 0 1 0 1 0.

2- Chuỗi bit biểu diễn các số chẵn trong U ($\{2, 4, 6, 8, 10\}$) là chuỗi có độ dài $n = 10$ trong đó các bit ở vị trí thứ 2, 4, 6, 8, 10 có giá trị là 1, các bit còn lại có giá trị là 0. Từ đó ta có chuỗi bit biểu diễn tập hợp B là: 0 1 0 1 0 1 0 1 0 1.

3- Chuỗi bit biểu diễn các số nhỏ hơn 5 trong U ($\{1, 2, 3, 4\}$) là chuỗi có độ dài $n = 10$ trong đó các bit ở vị trí thứ 1, 2, 3, 4 có giá trị là 1, các bit còn lại có giá trị là 0. Từ đó ta có chuỗi bit biểu diễn tập hợp C là: 1 1 1 0 0 0 0 0 0 0.

4- Chuỗi bit biểu diễn tập hợp $A \cup B$ là: $(1 0 1 0 1 0 1 0 1 0 \vee 0 1 0 1 0 1 0 1 0 1)$ là chuỗi 1 1 1 1 1 1 1 1. Như vậy, $A \cup B = U$.

5- Tương tự như vậy với $A \cap C \Leftrightarrow (1 0 1 0 1 0 1 0 1 0 \wedge 1 1 1 0 0 0 0 0 0 0)$ là chuỗi 1 0 1 0 0 0 0 0 0 0. Như vậy $A \cap C = \{1, 3\}$

1.7. Những nội dung cần ghi nhớ

Cần hiểu và nắm vững được những nội dung sau:

- ✓ Các phép toán hội, tuyển, tuyển loại, suy ra, kéo theo của logic mệnh đề.
- ✓ Các phương pháp chứng minh định lý dùng bảng chân lý và các tương đương logic.
- ✓ Phương pháp biểu diễn các câu hỏi thông thường bằng logic vị từ.
- ✓ Định nghĩa và các phép toán trên tập hợp.
- ✓ Phương pháp biểu diễn tập hợp trên máy tính

BÀI TẬP CHƯƠNG 1

1. Lập bảng giá trị chân lý cho các mệnh đề phức hợp sau:

- | | |
|--|--|
| a) $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$ | b) $(p \rightarrow q) \rightarrow (q \rightarrow p)$ |
| c) $(p \leftrightarrow q) \vee (p \oplus \neg q)$ | d) $(p \oplus q) \rightarrow (p \oplus \neg q)$ |
| e) $(p \leftrightarrow q) \vee (p \oplus \neg q)$ | f) $(\neg p \leftrightarrow \neg q) \leftrightarrow (p \leftrightarrow q)$ |
| g) $(p \vee q) \wedge \neg r$ | h) $(p \wedge q) \vee \neg r$ |
| i) $(p \leftrightarrow q) \vee (\neg q \leftrightarrow r)$ | j) $(\neg p \leftrightarrow \neg q) \leftrightarrow (q \leftrightarrow r)$ |

2- Dùng bảng chân lý chứng minh:

a) Luật giao hoán

$$p \vee q \Leftrightarrow q \vee p$$

$$p \wedge q \Leftrightarrow q \wedge p$$

b) Luật kết hợp

$$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$$

$$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$$

c) Luật phân phối

$$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$$

3- Chứng minh các công thức sau đây là đồng nhất đúng bằng cách lập bảng giá trị chân lý:

- a) $(X \rightarrow (Y \rightarrow Z)) \rightarrow ((X \rightarrow Y) \rightarrow (X \rightarrow Z));$
b) $(X \rightarrow Y) \rightarrow ((X \rightarrow Z) \rightarrow (X \rightarrow (Y \wedge Z)));$
c) $(X \rightarrow Z) \rightarrow ((Y \rightarrow Z) \rightarrow ((X \vee Y) \rightarrow Z)).$

4. Chứng minh các công thức sau đây là tương đương logic

$$a) X \vee (Y_1 \wedge Y_2 \wedge \dots \wedge Y_n) \Leftrightarrow (X \vee Y_1) \wedge (X \vee Y_2) \wedge \dots \wedge (X \vee Y_n)$$

$$b) X \wedge (Y_1 \vee Y_2 \vee \dots \vee Y_n) \Leftrightarrow (X \wedge Y_1) \vee (X \wedge Y_2) \vee \dots \vee (X \wedge Y_n)$$

$$c) \overline{(X_1 \vee X_2 \vee \dots \vee X_n)} \Leftrightarrow \overline{X_1} \wedge \overline{X_2} \wedge \dots \wedge \overline{X_n}$$

$$d) \overline{X_1 \wedge X_2 \wedge \dots \wedge X_n} \Leftrightarrow \overline{X_1} \vee \overline{X_2} \vee \dots \vee \overline{X_n}$$

5. Cho A, B, C là các tập hợp. Chứng minh rằng:
- a) $\overline{A \cap B \cap C} = \bar{A} \cup \bar{B} \cup \bar{C}$
 - b) $(A \cap B \cap C) \subseteq (A \cap B)$
 - c) $(A - B) - C \subseteq (A - C)$
 - d) $(A - C) \cap (C - B) = \Phi$
 - e) $(B - A) \cup (C - A) = (B \cup C) - A$
 - f) $A - B = A \cap \bar{B}$
 - g) $(A \cap B) \cup (A \cap \bar{B}) = A$

cuu duong than cong . com

cuu duong than cong . com

CHƯƠNG 2. BÀI TOÁN ĐẾM & BÀI TOÁN TỒN TẠI

Đếm các đối tượng có những tính chất nào đó là một bài toán quan trọng của lý thuyết tổ hợp. Giải quyết tốt bài toán đếm giúp ta giải nhiều bài toán khác nhau trong đánh giá độ phức tạp tính toán của các thuật toán và tìm xác suất rời rạc các biến cố. Phương pháp chung để giải bài toán đếm được dựa trên các nguyên lý đếm cơ bản (nguyên lý cộng, nguyên lý nhân). Một số bài toán đếm phức tạp hơn được giải bằng cách quy về các bài toán con để sử dụng được các nguyên lý đếm cơ bản hoặc tìm ra hệ thức truy hồi tổng quát.

Nội dung chính được đề cập trong chương này bao gồm:

- ✓ Các nguyên lý đếm cơ bản
- ✓ Nguyên lý bù trừ
- ✓ Hoán vị và tổ hợp
- ✓ Hệ thức truy hồi
- ✓ Quy về các bài toán con
- ✓ Giới thiệu bài toán tồn tại

Bạn đọc có thể tìm hiểu nhiều kỹ thuật đếm cao cấp hơn trong tài liệu [1], [2] trong phần tham khảo của tài liệu này.

2.1- Những nguyên lý đếm cơ bản

2.1.1- Nguyên lý cộng

Giả sử có hai công việc. Việc thứ nhất có thể tiến hành bằng n_1 cách, việc thứ hai có thể tiến hành bằng n_2 cách và nếu hai việc này không thể tiến hành đồng thời. Khi đó sẽ có $n_1 + n_2$ cách để giải quyết một trong hai việc trên.

Chúng ta có thể mở rộng quy tắc cộng cho trường hợp nhiều hơn hai công việc. Giả sử các việc T_1, T_2, \dots, T_m có thể làm tương ứng bằng n_1, n_2, \dots, n_m cách và giả sử không có hai việc T_i, T_j nào làm việc đồng thời ($i, j = 1, 2, \dots, m; i \neq j$). Khi đó, có $n_1 + n_2 + \dots + n_m$ cách thực hiện một trong các công việc T_1, T_2, \dots, T_m .

Quy tắc cộng được phát biểu dưới dạng của ngôn ngữ tập hợp như sau:

- ☐ Nếu A và B là hai tập rời nhau ($A \cap B = \emptyset$) thì: $N(A \cup B) = N(A) + N(B)$.
- ☐ Nếu A_1, A_2, \dots, A_n là những tập hợp rời nhau thì:

$$N(A_1 \cup A_2 \cup \dots \cup A_n) = N(A_1) + N(A_2) + \dots + N(A_n).$$

Ví dụ 1. Giả sử cần chọn hoặc một cán bộ hoặc một sinh viên tham gia một hội đồng của một trường đại học. Hỏi có bao nhiêu cách chọn vị đại biểu này nếu như có 37 cán bộ và 63 sinh viên.

Giải: gọi việc thứ nhất là chọn một cán bộ từ tập cán bộ ta có 37 cách. Gọi việc thứ hai là chọn một sinh viên từ tập sinh viên ta có 63 cách. Vì tập cán bộ và tập sinh viên là rời nhau, theo nguyên lý cộng ta có tổng số cách chọn vị đại biểu này là $37 + 63 = 100$ cách chọn.

Ví dụ 2. một đoàn vận động viên gồm môn bắn súng và bơi được cử đi thi đấu ở nước ngoài. Số vận động viên nam là 10 người. Số vận động viên thi bắn súng kể cả nam và nữ là 14 người. Số nữ vận động viên thi bơi bằng số vận động viên nam thi bắn súng. Hỏi đoàn có bao nhiêu người.

Giải: chia đoàn thành hai tập, tập các vận động viên nam và tập các vận động viên nữ. Ta nhận thấy tập nữ lại được chia thành hai: thi bắn súng và thi bơi. Thay số nữ thi bơi bằng số nam thi bắn súng, ta được số nữ bằng tổng số vận động viên thi bắn súng. Từ đó theo nguyên lý cộng toàn đoàn có $14 + 10 = 24$ người.

Ví dụ 3. giá trị của biến k sẽ bằng bao nhiêu sau khi thực hiện đoạn chương trình sau :

```

k := 0
for i1:= 1 to n1
    k:=k+1
for i2:= 1 to n2
    k:=k+1
.....
.....

.....
for im:= 1 to nm
    k:=k+1

```

Giải: coi mỗi vòng for là một công việc, do đó ta có m công việc T_1, T_2, \dots, T_m . Trong đó T_i thực hiện bởi n_i cách ($i=1, 2, \dots, m$). Vì các vòng for không lồng nhau hay các công việc không thực hiện đồng thời nên theo nguyên lý cộng tổng tất cả các cách để hoàn thành T_1, T_2, \dots, T_m là $k = n_1 + n_2 + \dots + n_m$.

2.1.2- Nguyên lý nhân

Giả sử một nhiệm vụ nào đó được tách ra hai công việc. Việc thứ nhất được thực hiện bằng n_1 cách, việc thứ hai được thực hiện bằng n_2 cách sau khi việc thứ nhất đã được làm, khi đó sẽ có $n_1.n_2$ cách thực hiện nhiệm vụ này.

Nguyên lý nhân có thể được phát biểu tổng quát bằng ngôn ngữ tập hợp như sau:

Nếu A_1, A_2, \dots, A_m là những tập hợp hữu hạn, khi đó số phần tử của tích đề các các tập này bằng tích số các phần tử của mỗi tập thành phần. Hay đẳng thức:

$$N(A_1 \times A_2 \times \dots \times A_m) = N(A_1) N(A_2) \dots N(A_m).$$

$$\text{Nếu } A_1 = A_2 = \dots = A_m \text{ thì } N(A^k) = N(A)^k$$

Ví dụ 1. giá trị của k sẽ bằng bao nhiêu sau khi ta thực hiện đoạn chương trình sau:

```

k:=0
for i1 = 1 to n1
    for i2 = 1 to n2
        .....
        .....
    for in =1 to nm

```

$$k:=k+1$$

Giải : Giá trị khởi tạo $k=0$. Mỗi vòng lặp lồng nhau đi qua giá trị của k được tăng lên 1 đơn vị. Gọi T_i là việc thi hành vòng lặp thứ i . Khi đó, số lần vòng lặp là số cách thực hiện công việc. Số cách thực hiện công việc T_j là n_j ($j=1,2, \dots, n$). Theo qui tắc nhân ta vòng lặp kép được duyệt qua $n_1 + n_2 + \dots + n_m$ lần và chính là giá trị của k .

Ví dụ 2. Người ta có thể ghi nhãn cho những chiếc ghế của một giảng đường bằng một chữ cái và sau đó là một số nguyên nhỏ hơn 100. Bằng cách như vậy hỏi có nhiều nhất bao nhiêu chiếc ghế có thể ghi nhãn khác nhau.

Giải: có nhiều nhất là $26 \times 100 = 2600$ ghế được ghi nhãn. Vì kí tự gán nhãn đầu tiên là một chữ cái vậy có 26 cách chọn các chữ cái khác nhau để ghi kí tự đầu tiên, tiếp theo sau là một số nguyên dương nhỏ hơn 100 do vậy có 100 cách chọn các số nguyên để gán tiếp sau của một nhãn. Theo qui tắc nhân ta nhận được $26 \times 100 = 2600$ nhãn khác nhau.

Ví dụ 3. Có bao nhiêu xâu nhị phân có độ dài 7.

Giải: một xâu nhị phân có độ dài 7 gồm 7 bit, mỗi bit có hai cách chọn (hoặc giá trị 0 hoặc giá trị 1), theo qui tắc nhân ta có $2.2.2.2.2.2.2 = 2^7 = 128$ xâu bit nhị phân độ dài 7.

Ví dụ 4. Có bao nhiêu hàm đơn ánh xác định từ một tập A có m phần tử nhận giá trị trên tập B có n phần tử.

Giải : Trước tiên ta nhận thấy, nếu $m > n$ thì tồn tại ít nhất hai phần tử khác nhau của A cùng nhận một giá trị trên B , như vậy với $m > n$ thì số các hàm đơn ánh từ $A \rightarrow B$ là 0. Nếu $m \leq n$, khi đó phần tử đầu tiên của A có n cách chọn, phần tử thứ hai có $n-1$ cách chọn, \dots , phần tử thứ k có $n-k+1$ cách chọn. Theo qui tắc nhân ta có $n(n-1)(n-2) \dots (n-m+1)$ hàm đơn ánh từ tập A sang tập B .

Ví dụ 5. Dạng của số điện thoại ở Bắc Mỹ được qui định như sau: số điện thoại gồm 10 chữ số được tách ra thành một nhóm mã vùng gồm 3 chữ số, nhóm mã chỉ nhánh gồm 3 chữ số và nhóm mã máy gồm 4 chữ số. Vì những nguyên nhân kỹ thuật nên có một số hạn chế đối với một số con số. Ta giả sử, X biểu thị một số có thể nhận các giá trị từ 0..9, N là số có thể nhận các chữ số từ 2..9, Y là các số có thể nhận các chữ số 0 hoặc 1.

Hỏi theo hai dự án đánh số NYX NNX XXXX và NXX NXX XXXX có bao nhiêu số điện thoại được đánh số khác nhau ở Bắc Mỹ.

Giải: đánh số theo dự án NYX NNX XXXX được nhiều nhất là :

$$8 \times 2 \times 10 \times 8 \times 8 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 = 2 \times 8^3 \times 10^6 = 1\,024 \cdot 10^6$$

đánh số theo dự án NXX NXX XXXX được nhiều nhất là :

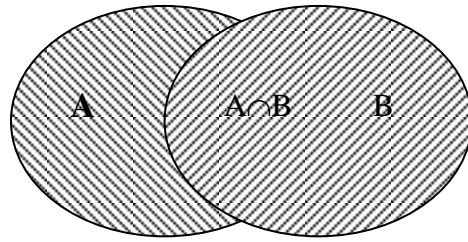
$$8 \times 10 \times 10 \times 8 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 = 8^2 \times 10^8 = 64 \cdot 10^8$$

Ví dụ 6. Dùng qui tắc nhân hãy chỉ ra rằng số tập con của một tập S hữu hạn là $2^{N(S)}$.

Giải: ta liệt kê các phần tử của tập S là $s_1, s_2, \dots, s_{N(S)}$. Xây dựng một xâu bit nhị phân dài $N(S)$ bit, trong đó nếu bit thứ i có giá trị 0 thì phần tử $s_i \notin S$, nếu bit thứ i có giá trị 1 thì phần tử $s_i \in S$ ($i=1, 2, \dots, N(S)$). Như vậy, theo nguyên lý nhân, số tập con của tập hợp S chính là số xâu bit nhị phân có độ dài $N(S)$. Theo ví dụ 3, chúng ta có $2^{N(S)}$ xâu bit nhị phân độ dài $N(S)$.

2.2- Nguyên lý bù trừ

Trong một số bài toán đếm phức tạp hơn. Nếu không có giả thiết gì về sự rời nhau giữa hai tập A và B thì $N(A \cup B) = N(A) + N(B) - N(A \cap B)$.



Ví dụ 1. lớp toán học rời rạc có 25 sinh viên giỏi tin học, 13 sinh viên giỏi toán và 8 sinh viên giỏi cả toán và tin học. Hỏi lớp có bao nhiêu sinh viên nếu mỗi sinh viên hoặc giỏi toán hoặc học giỏi tin học hoặc giỏi cả hai môn?

Giải: Gọi A tập là tập các sinh viên giỏi Tin học, B là tập các sinh viên giỏi toán. Khi đó $A \cap B$ là tập sinh viên giỏi cả toán học và tin học. Vì mỗi sinh viên trong lớp hoặc giỏi toán, hoặc giỏi tin học hoặc giỏi cả hai nên ta có tổng số sinh viên trong lớp là $N(A \cup B)$. Do vậy ta có:

$$N(A \cup B) = N(A) + N(B) - N(A \cap B) = 25 + 13 - 8 = 30.$$

Ví dụ 2. Có bao nhiêu số nguyên không lớn hơn 1000 chia hết cho 7 hoặc 11.

Giải : Gọi A là tập các số nguyên không lớn hơn 1000 chia hết cho 7, B là tập các số nguyên không lớn hơn 1000 chia hết cho 11. Khi đó tập số nguyên không lớn hơn 1000 hoặc chia hết cho 7 hoặc chia hết cho 11 là $N(A \cup B)$. Theo công thức 1 ta có:

$$\begin{aligned} N(A \cup B) &= N(A) + N(B) - N(A \cap B) = \lfloor 1000/7 \rfloor + \lfloor 1000/11 \rfloor - \lfloor 1000/7 \cdot 11 \rfloor \\ &= 142 + 90 - 12 = 220. \end{aligned}$$

Trước khi đưa ra công thức tổng quát cho n tập hợp hữu hạn. Chúng ta đưa ra công thức tính số phần tử của hợp 3 tập A, B, C.

Ta nhận thấy $N(A) + N(B) + N(C)$ đếm một lần những phần tử chỉ thuộc một trong ba tập hợp. Như vậy, số phần tử của $A \cap B$, $A \cap C$, $B \cap C$ được đếm hai lần và bằng $N(A \cap B)$, $N(A \cap C)$, $N(B \cap C)$, được đếm ba lần là những phần tử thuộc $A \cap B \cap C$. Như vậy, biểu thức:

$N(A \cup B \cup C) - N(A \cap B) - N(A \cap C) - N(B \cap C)$ chỉ đếm các phần tử chỉ thuộc một trong ba tập hợp và loại bỏ đi những phần tử được đếm hai lần. Như vậy, số phần tử được đếm ba lần chưa được đếm, nên ta phải cộng thêm với giao của cả ba tập hợp. Từ đó ta có công thức đối với 3 tập không rời nhau:

$$N(A \cup B \cup C) = N(A) + N(B) + N(C) - N(A \cap B) - N(A \cap C) - N(B \cap C) + N(A \cap B \cap C)$$

Định lý. Nguyên lý bù trừ. Giả sử A_1, A_2, \dots, A_m là những tập hữu hạn. Khi đó

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

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

$$N(A_1 \cup A_2 \cup \dots \cup A_n) = \sum_{1 \leq i \leq n} N(A_i) - \sum_{1 \leq i, j < n} N(A_i \cap A_j) + \sum_{1 \leq i < j < k \leq n} N(A_i \cap A_j \cap A_k) - \dots + (-1)^{n+1} N(A_1 \cap A_2 \cap \dots \cap A_n)$$

Định lý được chứng minh bằng cách chỉ ra mỗi phần tử của hợp n tập hợp được đếm đúng một lần. Bạn đọc có thể tham khảo cách chứng minh trong tài liệu [1].

Ví dụ 3. Tìm công thức tính số phần tử của 4 tập hợp.

Giải : Từ nguyên lý bù trừ ta có

$$N(A_1 \cup A_2 \cup A_3 \cup A_4) = N(A_1) + N(A_2) + N(A_3) + N(A_4) - N(A_1 \cap A_2) - N(A_1 \cap A_3) - N(A_1 \cap A_4) - N(A_2 \cap A_3) - N(A_2 \cap A_4) - N(A_3 \cap A_4) + N(A_1 \cap A_2 \cap A_3) + N(A_1 \cap A_2 \cap A_4) + N(A_1 \cap A_3 \cap A_4) + N(A_2 \cap A_3 \cap A_4) - N(A_1 \cap A_2 \cap A_3 \cap A_4).$$

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

Giải: Gọi A là tập các số nhỏ hơn 10000 chia hết cho 3, B là tập các số nhỏ hơn 10000 chia hết cho 4, C là tập các số nhỏ hơn 10000 chia hết cho 7. Theo nguyên lý bù trừ ta có:

$$N(A \cup B \cup C) = N(A) + N(B) + N(C) - N(A \cap B) - N(A \cap C) - N(B \cap C) + N(A \cap B \cap C)$$

trong đó :

$$\begin{aligned} N(A) + N(B) + N(C) &= [10\,000/3] + [10\,000/4] + [10\,000/7] \\ &= 3333 + 2500 + 1428 = 7261 \end{aligned}$$

$$N(A \cap B) = N(A) + N(B) - N(A \cup B) = 3333 + 2500 - [10000/3 \times 4] = 833$$

$$N(A \cap C) = N(A) + N(C) - N(A \cup C) = 3333 + 1428 - [10000/3 \times 7] = 476$$

$$N(B \cap C) = N(B) + N(C) - N(B \cup C) = 2500 + 1428 - [10000/4 \times 7] = 357$$

$$N(A \cap B) + N(A \cap C) + N(B \cap C) = 833 + 476 + 357 = 1666$$

$$N(A \cap B \cap C) = [10000/3 \times 4 \times 7] = 119.$$

=> Số các số nhỏ hơn 10000 cần đếm là :

$$1000 - N(A \cup B \cup C) = 7261 - 1666 + 119 = 4286.$$

Ví dụ 5. Có bao nhiêu xâu nhị phân độ dài 10 bắt đầu bởi 00 hoặc kết thúc bởi 11.

Giải : Gọi A là số xâu nhị phân độ dài 10 bắt đầu bởi 00, B là số xâu nhị phân độ dài 10 kết thúc bởi 11. Dễ dàng nhận thấy, $N(A) = N(B) = 2^8 = 64$. Theo nguyên lý bù trừ ta có:

$$\begin{aligned} N(A \cup B) &= N(A) + N(B) - N(A \cap B) \\ &= 256 + 256 - 64 = 448. \end{aligned}$$

Ví dụ 6. 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 đặt ra là đếm số cách bỏ thư sao cho không 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 k bỏ đúng địa chỉ. Khi đó theo nguyên lý bù trừ ta có:

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

Trong đó \overline{N} là số cần tìm, $N = n!$, 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 mọi cách lấy k lá thư từ n lá, với mỗi cách lấy k lá thư, có $(n-k)!$ cách bỏ để k lá thư này đúng địa chỉ, từ đó ta nhận được.

$$N_k = C(n, k)(n-k)! = \frac{n!}{k!} \text{ và } \overline{N} = n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \dots + \frac{(-1)^n}{n!} \right)$$

Từ đó ta có xác suất cần tìm là:

$$1 - \frac{1}{1!} + \frac{1}{2!} - \dots + \frac{(-1)^n}{n!} = e^{-1}$$

Số được tính như trên được gọi là số mắt thứ tự và được ký hiệu là D_n . Dưới đây là một vài giá trị của D_n , sự tăng nhanh của D_n một lần nữa cho ta thấy rõ sự bùng nổ tổ hợp.

N	2	3	4	5	6	7	8	9	10	11
D_n	1	2	9	44	265	1845	14833	133496	1334961	4890741

2.3. Đếm các hoán vị và tổ hợp

2.3.1- Chính hợp lặp

Định nghĩa 1. Một chính hợp lặp chập k của n phần tử là bộ có thứ tự gồm k thành phần lấy từ n phần tử của tập đã cho.

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

Ví dụ 1. Tính số hàm từ tập có k phần tử vào tập có n phần tử.

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

Ví dụ 2. Từ bảng chữ cái tiếng Anh có thể tạo ra được bao nhiêu xâu có độ dài n .

Giải : Bảng chữ cái tiếng Anh gồm 26 kí tự ['A'..'Z'], số các xâu có độ dài n được chọn từ 26 chữ cái chính là chính hợp lặp n của 26 phần tử và bằng 26^n .

Ví dụ 3. Tính xác suất lấy ra liên tiếp được 3 quả bóng đỏ ra khỏi bình kín chứa 5 quả đỏ, 7 quả xanh nếu sau mỗi lần lấy một quả bóng ra lại bỏ nó trở lại bình.

Giải: Số kết cục có lợi để ta lấy ra liên tiếp 3 quả bóng đỏ là 5^3 vì có 5 quả đỏ ta phải lấy 3 quả (chú ý vì có hoàn lại). Toàn bộ kết cục có thể để lấy ra ba quả bóng bất kỳ trong 12 quả bóng là 12^3 . Như vậy, xác suất để có thể lấy ra 3 quả bóng đỏ liên tiếp là $5^3/12^3$.

2.3.2- Chính hợp không lặp

Định nghĩa 2. Chính hợp không lặp chập k của n phần tử là bộ có thứ tự gồm k thành phần lấy ra từ n phần tử đã cho. Các phần tử không được lặp lại.

Để xây dựng một chính hợp không lặp, ta xây dựng từ thành phần đầu tiên. Thành phần này có n khả năng chọn. Mỗi thành phần tiếp theo những khả năng chọn giảm đi 1 (vì không được lấy lặp lại). Tới thành phần thứ k có $n-k+1$ khả năng chọn. Theo nguyên lý nhân ta có số chính hợp lặp k của tập hợp n phần tử ký hiệu là $P(n, k)$ được tính theo công thức:

$$P(n, k) = n(n-1) \dots (n-k+1) = \frac{n!}{(n-k)!}$$

Ví dụ 1. Tìm số hàm đơn ánh có thể xây dựng được từ tập k phần tử sang tập n phần tử.

Giải: Số hàm đơn ánh từ tập k phần tử sang tập n phần tử chính là $P(n, k)$.

Ví dụ 2. Giả sử có tám vận động viên chạy thi. Người về nhất sẽ được nhận huân chương vàng, người về nhì nhận huân chương bạc, người về ba nhận huy chương đồng. Hỏi có bao nhiêu cách trao huy chương nếu tất cả các kết cục đều có thể xảy ra.

Giải: Số cách trao huy chương chính là số chỉnh hợp chập 3 của tập hợp 8 phần tử. Vì thế có $P(8,3) = 8.7.6 = 336$ cách trao huy chương.

Ví dụ 3. Có bao nhiêu cách chọn 4 cầu thủ khác nhau trong đội bóng gồm 10 cầu thủ để tham gia các trận đấu đơn.

Giải : Có $P(10,4) = 10.9.8.7 = 5040$ cách chọn.

2.3.3- Hoán vị

Định nghĩa 3. Ta gọi các hoán vị của n phần tử là một cách xếp có thứ tự các phần tử đó. Số các hoán vị của tập n phần tử có thể coi là trường hợp riêng của chỉnh hợp không lặp với $k = n$.

Ta cũng có thể đồng nhất một hoán vị với một song ánh từ tập n phần tử lên chính nó. Như vậy, số hoán vị của tập gồm n phần tử là $P(n, n) = n!$.

Ví dụ 1. Có 6 người xếp thành hàng để chụp ảnh. Hỏi có thể bố trí chụp được bao nhiêu kiểu khác nhau.

Giải: Mỗi kiểu ảnh là một hoán vị của 6 người. Do đó có $6! = 720$ kiểu ảnh khác nhau có thể chụp.

Ví dụ 2. Cần bố trí thực hiện n chương trình trên một máy tính. Hỏi có bao nhiêu cách bố trí khác nhau.

Giải: Số chương trình được đánh số từ 1, 2, ..., n . Như vậy, số chương trình cần thực hiện trên một máy tính là số hoán vị của 1, 2, ..., n .

Ví dụ 3. Một thương nhân đi bán hàng tại tám thành phố. Chị ta có thể bắt đầu hành trình của mình tại một thành phố nào đó nhưng phải qua 7 thành phố kia theo bất kỳ thứ tự nào mà chị ta muốn. Hỏi có bao nhiêu lộ trình khác nhau mà chị ta có thể đi.

Giải: Vì thành phố xuất phát đã được xác định. Do vậy thương nhân có thể chọn tùy ý 7 thành phố còn lại để hành trình. Như vậy, tất cả số hành trình của thương nhân có thể đi qua là $7! = 5040$ cách.

2.3.4- Tổ hợp

Định nghĩa 4. 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 lấy trong n phần tử. Số tổ hợp chập k của n phần tử kí hiệu là $C(n,k)$.

Ta có thể tính được trực tiếp số các tổ hợp chập k của tập n phần tử thông qua chỉnh hợp không lặp của k phần tử.

Xét tập hợp tất cả các chỉnh hợp không lặp chập k của n phần tử. Sắp xếp chúng thành những lớp sao cho hai chỉnh hợp thuộc cùng một lớp chỉ khác nhau về thứ tự. Rõ ràng mỗi lớp như vậy là một tổ hợp chập k của n phần tử ($P(n,k)$). Số chỉnh hợp trong mỗi lớp đều bằng nhau và bằng $k!$ (số hoán vị k phần tử: $P(k,k)$). Số các lớp bằng số tổ hợp chập k của n ($P(n,k)$). Từ đó ta có:

$$P(n,k) = C(n,k).P(k,k) \Rightarrow C(n,k) = \frac{P(n,k)}{k!} = \frac{n!}{k!(n-k)!} \quad (1)$$

Ví dụ 1. Cho $S = \{ a, b, c, d \}$ tìm $C(4,2)$.

Giải. Rõ ràng $C(4,2) = 6$ tương ứng với 6 tập con $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, $\{b,c\}$, $\{b, d\}$ $\{c,d\}$.

Ví dụ 2. Có n đội bóng thi đấu vòng tròn. Hỏi phải tổ chức bao nhiêu trận đấu.

Giải: Cứ hai đội bóng thì có một trận. Từ đó suy ra số trận đấu sẽ bằng số cách chọn 2 trong n đội, nghĩa là bằng $C(n, 2) = n! / 2!(n-2)! = n(n-1)/2$ trận đấu.

Ví dụ 3. Chứng minh

$$a) C(n, k) = C(n, n-k) \quad (2)$$

$$b) C(n, 0) = C(n, n) = 1 \quad (3)$$

$$c) C(n, k) = C(n-1, k-1) + C(n-1, k) \quad (4)$$

Giải a. $C(n, n-k) = n! / (n-k)! (n-(n-k))! = n! / k! (n-k)! = C(n, k)$.

$$\text{Hoặc } C(n, k) = n! / k! (n-k)! = n! / (n-k)! (n-(n-k))! = C(n, n-k);$$

b) Chú ý $0! = 1 \Rightarrow$ b hiển nhiên đúng

$$c) C(n, k) = C(n-1, k-1) + C(n-1, k)$$

$$\begin{aligned} C(n-1, k-1) + C(n-1, k) &= \frac{(n-1)!}{(k-1)!(n-1-k+1)!} + \frac{(n-1)!}{k!(n-k-1)!} \\ &= \frac{(n-1)!}{(k-1)!(n-k-1)!} \left(\frac{1}{n-k} + \frac{1}{k} \right) = \frac{(n-1)!n}{(k-1)!k(n-k-1)!(n-k)} \\ &= \frac{n!}{k!(n-k)!} = C(n, k) \end{aligned}$$

Từ những tính chất trên, 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 dòng, trên mỗi dòng ta tính và thực hiện theo cột. Bảng có dạng tam giác chính là tam giác Pascal.

Các hệ số tổ hợp có liên quan chặt chẽ tới việc khai triển lũy thừa của một nhị thức. Thực vậy, trong tích

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

$$(x+y)^n = C(n, 0)x^n + C(n, 1)x^{n-1}y + \dots + C(n, n-1)xy^{n-1} + C(n, n)y^n = \sum_{k=0}^n C(n, k)x^{n-k}y^k \quad (5)$$

Công thức (5) còn được gọi là khai triển nhị thức Newton, các hệ số tổ hợp còn được gọi là hệ số nhị thức. Chẳng hạn lũy thừa bậc 8 của nhị thức $(x+y)^8$ được khai triển như sau:

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

Trong trường hợp $y=1$, tức khai triển $(x+1)^n$ ta có:

$$(x+1)^n = C(n, 0)x^n + C(n, 1)x^{n-1} + \dots + C(n, n-1)x + C(n, n)$$

Hoặc đẳng thức sau sẽ được rút ra từ khai triển nhị thức Newton:

$$2^n = (1+1)^n = C(n, 0) + C(n, 1) + \dots + C(n, n-1) + C(n, n)$$

Có thể nói rất nhiều đẳng thức về hệ số tổ hợp sẽ được suy ra. Như tính các tập lẻ, đạo hàm . . .

2.4- Hệ thức truy hồi

2.4.1- Định nghĩa và ví dụ

Thông thường người ta thường quan tâm tới những bài toán đếm trong đó kết quả đếm phụ thuộc vào một tham số đầu vào (mà ta ký hiệu là n), chẳng hạn như các số mất thứ tự D_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. Trong nhiều trường hợp, việc tìm ra một công thức trực tiếp giữa kết quả đếm và n là hết sức khó khăn và nhiều khi không giải quyết được, trong khi đó công thức liên hệ giữa kết quả đếm ứng với giá trị n với các kết quả bé hơn n lại đơn giản và dễ tìm. Thông qua 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. Đặc biệt, công thức truy hồi rất thích hợp với lập trình trên máy tính. Nó cũng cho phép giảm đáng kể độ phức tạp cũng như gia tăng độ ổn định của quá trình tính toán.

Định nghĩa 1. Hệ thức truy hồi đối với dãy số $\{a_n\}$ là công thức biểu diễn a_n qua một hay nhiều số hạng đi trước của dãy, cụ thể là a_1, a_2, \dots, a_{n-1} với mọi $n \geq 0$ nguyên dương. Dãy số được gọi là lời giải hay nghiệm của hệ thức truy hồi nếu các số hạng của nó thỏa mãn hệ thức truy hồi.

Ví dụ 1. Lãi kép. Giả sử một người gửi 10000 đô la vào tài khoản của mình tại một ngân hàng với lãi suất kép 11% mỗi năm. Hỏi sau 30 năm anh ta có bao nhiêu tiền trong tài khoản của mình?

Giải: Gọi P_n là tổng số tiền có trong tài khoản sau n năm. Vì số tiền có trong tài khoản sau n năm bằng số tiền có được trong $n-1$ năm cộng với lãi xuất năm thứ n . Nên dãy $\{P_n\}$ thỏa mãn hệ thức truy hồi :

$$P_n = P_{n-1} + 0.11P_{n-1} = 1.11P_{n-1}$$

Chúng ta có thể dùng phương pháp lặp để tìm công thức trên cho P_n . Dễ nhận thấy rằng:

$$P_0 = 10000$$

$$P_1 = 1.11P_0$$

$$P_2 = 1.11P_1 = (1.11)^2P_0$$

.....

$$P_n = 1.11P_{n-1} = (1.11)^{n-1}P_0$$

Ta có thể chứng minh tính đúng đắn của công thức truy hồi bằng qui nạp.

Thay $P_0 = 10000$, và $n = 30$ ta được:

$$P_{30} = (1.11)^{30}10000 = 228922,97 \$$$

Ví dụ 2. Họ nhà thỏ và số Fibonacci. Một cặp thỏ sinh đôi (một con đực và một con cái) được thả lên một hòn đảo. Giả sử rằng cặp thỏ sẽ chưa sinh sản được trước khi đầy hai tháng tuổi. Từ khi chúng đầy hai tháng tuổi, mỗi tháng chúng sinh thêm được một cặp thỏ. Tìm công thức truy hồi tính số cặp thỏ trên đảo sau n tháng với giả sử các cặp thỏ là trường thọ.

Số tháng	Số cặp sinh sản	Số cặp thỏ con	Tổng số cặp thỏ
1	0	1	1
2	0	1	1

3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
.....

Giải: Giả sử f_n là số cặp thỏ sau n tháng. Ta sẽ chỉ ra rằng f_1, f_2, \dots, f_n ($n=1, 2, \dots, n$) là các số của dãy fibonacci.

Cuối tháng thứ nhất số cặp thỏ trên đảo là $f_1 = 1$. Vì tháng thứ hai cặp thỏ vẫn chưa đến tuổi sinh sản được nên trong tháng thứ hai $f_2 = 1$. Vì mỗi cặp thỏ chỉ được sinh sản sau ít nhất hai tháng tuổi, nên ta tìm số cặp thỏ sau tháng thứ n bằng cách cộng số cặp thỏ sau tháng $n-2$ và tháng $n-1$ hay $f_n = f_{n-1} + f_{n-2}$. Do vậy, dãy $\{f_n\}$ thỏa mãn hệ thức truy hồi

$$f_n = f_{n-1} + f_{n-2} \text{ với } n \geq 3 \text{ và } f_1 = 1, f_2 = 1.$$

Ví dụ 3: Tính số mất thứ tự D_n .

Giải: Đánh số thư và phong bì thư 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_2, \dots, a_n) của $\{1, 2, \dots, n\}$. Một mất thứ tự được định nghĩa là là một hoán vị (a_1, a_2, \dots, a_n) sao cho $a_i \neq i$ với mọi i . Thành phần a_1 có thể chấp nhận mọi giá trị ngoài 1. Với mỗi giá trị k ($k \neq 1$) của a_1 , xét hai 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 mất thứ tự 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 còn lại, tức là số mất thứ tự loại này thuộc loại D_{n-1} .

Từ đó ta nhận được công thức

$$D_n = (n-1)(D_{n-1} + D_{n-2}), n \geq 3 \text{ với } D_1 = 0, D_2 = 1.$$

Mọi giá trị còn lại được tính đơn giản nhờ luật kế thừa.

$$D_3 = (3-1)(0+1) = 2$$

$$D_4 = (4-1)(1+2) = 9$$

$$D_5 = (5-1)(9+2) = 44$$

$$D_6 = (6-1)(9+44) = 265$$

$$D_7 = (7-1)(44+265) = 1854$$

$$D_8 = (8-1)(265+1854) = 14833$$

.....

Để công thức đúng với $n = 2$, ta coi $D_0 = 1$

Có thể nhận được số mất thứ tự thông qua công thức truy hồi trên vì:

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

Đặt $V_n = D_n - nD_{n-1}$ ta có:

$D_n - nD_{n-1} = V_n = -V_{n-1} = \dots = (-1)^{n-1}V_1 = (-1)^n$. Hay ta có thể viết:

$\frac{D_n}{n!} - \frac{D_{n-1}}{(n-1)!} = \frac{(-1)^n}{n!}$. Cộng các hệ thức trên với $n = 1, 2, \dots, n$ ta được:

$\frac{D_n}{n!} = 1 - \frac{1}{1!} + \frac{1}{2!} - \dots + \frac{(-1)^n}{n!}$. Từ đó thu lại được công thức cũ:

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

Ví 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 này thành hai lớp (lớp chứa a và lớp không chứa a). Nếu a được chọn thì ta cần 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 ra trực tiếp:

$$C(n, 0) = C(n, n) = 1.$$

Phương pháp này được gọi là phương pháp khử. Không phải lúc nào cũng dễ dàng khử được công thức truy hồi 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.

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

Định nghĩa 1. Một hệ thức truy hồi tuyến tính thuần nhất bậc k với hệ số hằng số là hệ thức truy hồi có dạng:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k} \quad (1), \text{ trong đó } c_1, c_2, \dots, c_k \text{ là các số thực và } c_k \neq 0$$

Ta cần tìm công thức trực tiếp cho số hạng a_n của dãy số $\{a_n\}$ thỏa mãn công thức (1). Theo nguyên lý thứ hai của qui nạp toán học thì dãy số thỏa mãn định nghĩa trên được xác định duy nhất nếu như nó thỏa mãn k điều kiện đầu:

$$a_0 = C_0, a_1 = C_1, \dots, a_{k-1} = C_{k-1}, \text{ trong đó } C_1, C_2, \dots, C_{k-1} \text{ là các hằng số.}$$

Ví dụ 1. Hệ thức truy hồi $P_n = (1.11)P_{n-1}$ là hệ thức truy hồi tuyến tính thuần nhất bậc 1. Hệ thức truy hồi $f_n = f_{n-1} + f_{n-2}$ là hệ thức truy hồi tuyến tính thuần nhất bậc 2. Hệ thức truy hồi $a_n = a_{n-5}$ là hệ thức truy hồi tuyến tính thuần nhất bậc 5. Hệ thức truy hồi $B_n = nB_{n-1}$ không phải là hệ thức truy hồi tuyến tính thuần nhất vì nó không có hệ số hằng số.

Phương pháp cơ bản để giải hệ thức truy hồi tuyến tính thuần nhất là tìm nghiệm dưới dạng $a_n = r^n$, trong đó r là hằng số. Cũng cần chú ý rằng $a_n = r^n$ là nghiệm của hệ thức truy hồi $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k}$ nếu và chỉ nếu

$$a_n = c_1 r_{n-1} + c_2 r_{n-2} + \dots + c_k r_{n-k}.$$

Chia cả hai vế cho r^{n-k} ta nhận được

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \dots - c_{k-1} r - c_k = 0 \quad (2)$$

Vậy dãy $\{a_n\}$ với $a_n=r^n$ là nghiệm nếu và chỉ nếu r là nghiệm của (2). Phương trình 2 còn được gọi là phương trình đặc trưng của hệ thức truy hồi, nghiệm của nó là nghiệm đặc trưng của hệ thức truy hồi. Nghiệm của phương trình đặc trưng dùng để biểu diễn công thức tất cả các nghiệm của hệ thức truy hồi.

Chúng ta sẽ trình bày các kết quả với hệ thức truy hồi tuyến tính thuần nhất bậc hai. Sau đó ta sẽ nêu ra những kết quả tương tự cho trường hợp tổng quát khi bậc lớn hơn hai.

Định lý 1. Cho c_1, c_2 là các hằng số thực. Giả sử $r^2 - c_1r + c_2=0$ có hai nghiệm phân biệt r_1, r_2 . Khi đó dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi $a_n = c_1a_{n-1} + c_2a_{n-2}$ khi và chỉ khi $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ với $n = 1, 2, \dots$, α_1, α_2 là các hằng số.

Chứng minh: Để chứng minh định lý này ta cần thực hiện hai việc. Đầu tiên ta cần chỉ ra rằng nếu r_1, r_2 là hai nghiệm của phương trình đặc trưng và α_1, α_2 là hai hằng số thì dãy $\{a_n\}$ với $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ là nghiệm của hệ thức truy hồi. Ngược lại, cần phải chứng minh rằng nếu $\{a_n\}$ là nghiệm thì $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ với α_1, α_2 là các hằng số nào đó.

(\Rightarrow): Giả sử r_1 và r_2 là hai nghiệm phân biệt của $r^2 - c_1r + c_2=0$, khi đó $r_1^2 = c_1r_1 + c_2$; $r_2^2 = c_1r_2 + c_2$ đồng thời ta thực hiện dãy các phép biến đổi sau:

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

Điều này chứng tỏ dãy $\{a_n\}$ với $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ là nghiệm của hệ thức truy hồi đã cho.

(\Leftarrow): Để chứng minh ngược lại, ta giả sử dãy $\{a_n\}$ là một nghiệm bất kỳ của hệ thức truy hồi. Ta chọn α_1, α_2 sao cho dãy $\{a_n\}$ với $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ thỏa mãn các điều kiện đầu $a_0 = C_0, a_1 = C_1$. Thực vậy,

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

Từ phương trình đầu ta có $\alpha_2 = C_0 - \alpha_1$ thế vào phương trình thứ hai ta có:

$$C_1 = \alpha_1 r_1 + (C_0 - \alpha_1) r_2 = \alpha_1 (r_1 - r_2) + C_0 - r_2; \text{ Từ đây suy ra:}$$

$$\alpha_1 = \frac{(C_1 - C_0 r_2)}{r_1 - r_2}; \alpha_2 = C_0 - \alpha_1 = C_0 - \frac{(C_1 - C_0 r_2)}{r_1 - r_2} = \frac{(C_0 r_1 - C_1)}{r_1 - r_2}.$$

Như vậy, khi chọn những giá trị trên cho α_1, α_2 dãy $\{a_n\}$ với $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ thỏa mãn các điều kiện đầu. Vì hệ thức truy hồi và các điều kiện đầu được xác định duy nhất nên $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$. Định lý được chứng minh.

Ví dụ 1. Tìm nghiệm của hệ thức truy hồi $a_n = a_{n-1} + 2a_{n-2}$ với $a_0 = 2, a_1 = 7$.

Giải: Phương trình đặc trưng của hệ thức truy hồi có dạng $r^2 - r - 2 = 0$. Nghiệm của nó là $r=2$ và $r = -1$. Theo định lý 1, dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi nếu và chỉ nếu :

$a_n = \alpha_1 2^n + \alpha_2 (-1)^n$ với α_1, α_2 là các hằng số nào đó. Từ các điều kiện đầu suy ra:

$$a_0 = 2 = \alpha_1 + \alpha_2$$

$$a_1 = 7 = \alpha_1 2 + \alpha_2 (-1)$$

Giải ra ta được $\alpha_1 = 3, \alpha_2 = -1$. Vậy nghiệm của biểu thức truy hồi với điều kiện đầu là dãy $\{a_n\}$ với $a_n = 3 \cdot 2^n - (-1)^n$.

Ví dụ 2. Tìm công thức hiển của các số fibonacci.

Giải: Các số fibonacci thỏa mãn hệ thức $f_n = f_{n-1} + f_{n-2}$ và các điều kiện đầu $f_0 = 0, f_1 = 1$. Các nghiệm của phương trình đặc trưng là:

$$r_1 = \left(\frac{1 + \sqrt{5}}{2} \right); \quad r_2 = \left(\frac{1 - \sqrt{5}}{2} \right) \text{ theo định lý 1 ta suy ra số fibonacci được cho bởi công thức}$$

sau:

$$f_n = \alpha_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + \alpha_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n \text{ với } \alpha_1, \alpha_2 \text{ là hai hằng số. Các điều kiện đầu } f_0 = 0, f_1 = 1$$

được dùng để xác định các hằng số α_1, α_2 .

$$f_0 = \alpha_1 + \alpha_2 = 0$$

$$f_1 = \alpha_1 \left(\frac{1 + \sqrt{5}}{2} \right) + \alpha_2 \left(\frac{1 - \sqrt{5}}{2} \right) = 1$$

Từ hai phương trình này ta suy ra $\alpha_1 = \frac{1}{\sqrt{5}}; \alpha_2 = -\frac{1}{\sqrt{5}}$ do đó các số fibonacci được cho bằng công thức dạng hiển như sau:

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

Định lý 1 không dùng được trong trường hợp nghiệm của phương trình đặc trưng là nghiệm bội. Khi phương trình đặc trưng có nghiệm bội ta sử dụng định lý sau.

Định lý 2. Cho c_1, c_2 là các hằng số thực, $c_2 \neq 0$. Giả sử $r^2 - c_1 r - c_2 = 0$ chỉ có một nghiệm r_0 . Dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ khi và chỉ khi $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$ với $n = 1, 2, \dots$ trong đó α_1, α_2 là những hằng số.

Chứng minh tương tự như định lý 1.

Ví dụ 3. Tìm nghiệm của công thức truy hồi $a_n = 6a_{n-1} - 9a_{n-2}$ với các điều kiện đầu $a_0 = 1, a_1 = 6$.

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

$$a_n = \alpha_1 3^n + \alpha_2 n 3^n \text{ với } \alpha_1, \alpha_2 \text{ là các hằng số nào đó. Từ các điều kiện đầu ta suy ra:}$$

$$a_0 = 1 = \alpha_1$$

$a_1 = 6 = \alpha_1 3 + \alpha_2 3 \Rightarrow \alpha_1 = 1, \alpha_2 = 1$ vậy nghiệm của hệ thức truy hồi và các điều kiện đầu đã cho là:

$$a_n = 3^n + n3^n$$

Bây giờ ta phát biểu kết quả tổng quát về nghiệm các hệ thức truy hồi tuyến tính thuần nhất với các hệ số hằng số.

Định lý 3. Cho c_1, c_2, \dots, c_k là các số thực. Giả sử phương trình đặc trưng

$r^k - c_1 r^{k-1} - \dots - c_k = 0$ có k nghiệm phân biệt r_1, r_2, \dots, r_k . Khi đó dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi

$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k}$ khi và chỉ khi $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n + \dots + \alpha_k r_k^n$ với $n=0, 1, 2, \dots$, trong đó $\alpha_1, \alpha_2, \dots, \alpha_k$ là các hằng số.

Ví dụ 4. Tìm nghiệm của hệ thức truy hồi $a_n = 6a_{n-1} - 11a_{n-2} + 6a_{n-3}$ với điều kiện đầu $a_0=2, a_1=5, a_2=15$.

Giải: Đa thức đặc trưng của hệ thức truy hồi là :

$r^3 - 6r^2 + 11r - 6$ có các nghiệm là $r_1=1, r_2=2, r_3=3$. Do vậy nghiệm của hệ thức truy hồi có dạng: $a_n = \alpha_1 1^n + \alpha_2 2^n + \alpha_3 3^n$.

Để tìm các hằng số $\alpha_1, \alpha_2, \alpha_3$ ta dựa vào những điều kiện ban đầu:

$$a_0 = 2 = \alpha_1 + \alpha_2 + \alpha_3$$

$$a_1 = 5 = \alpha_1 + \alpha_2 2 + \alpha_3 3$$

$$a_2 = 15 = \alpha_1 + \alpha_2 4 + \alpha_3 9$$

Giải: hệ phương trình này ta nhận được $\alpha_1 = 1, \alpha_2 = -1, \alpha_3 = 2$. Vì vậy nghiệm duy nhất của hệ thức truy hồi này và các điều kiện đầu đã cho là dãy $\{a_n\}$ với:

$$a_n = 1 - 2^n + 2 \cdot 3^n$$

2.5- Quy về các bài toán đơn giản

Một trong những phương pháp giải quyết bài toán đếm phức tạp là quy bài toán đang xét về những bài toán nhỏ hơn. Sự phân chia này được thực hiện một cách liên tiếp cho tới khi nhận được lời giải của bài toán nhỏ một cách dễ dàng. Tuy nhiên điều này không phải lúc nào cũng thực hiện được vì nó đòi hỏi một sự phân tích sâu sắc cấu hình cần đếm.

Giả sử rằng có một thuật toán phân chia bài toán cỡ n thành a bài toán nhỏ, trong đó mỗi bài toán nhỏ có cỡ n/b (để đơn giản ta giả sử n chia hết cho b); trong thực tế các bài toán nhỏ thường có cỡ là số nguyên gần nhất với n/b . Giả sử tổng các phép toán thêm vào khi thực hiện phân chia bài toán cỡ n thành các bài toán cỡ nhỏ hơn là $g(n)$. Khi đó nếu $f(n)$ là số các phép toán cần thiết để giải bài toán đã cho thì f thỏa mãn hệ thức truy hồi sau:

$$f(n) = af\left(\frac{n}{b}\right) + g(n); \text{ hệ thức này có tên là hệ thức chia để trị.}$$

Ví dụ 1. Xét thuật toán nhân hai số nguyên kích cỡ $2n$ bit. Kỹ thuật này gọi là thuật toán nhân nhanh có dùng kỹ thuật chia để trị.

Giải: Giả sử a và b là các số nguyên có biểu diễn nhị phân là $2n$ bit (có thể thêm các bit 0 vào đầu để chúng có thể dài bằng nhau).

$$a = (a_{2n-1}a_{2n-2} \cdots a_1a_0)_2 \text{ và } b = (b_{2n-1}b_{2n-2} \cdots b_1b_0)_2$$

$$\text{Giả sử } a = 2^n A_1 + A_0, b = 2^n B_1 + B_0$$

trong đó

$$A_1 = (a_{2n-1}a_{2n-2} \cdots a_{n+1}a_n)_2; A_0 = (a_{n-1}a_{n-2} \cdots a_1a_0)_2$$

$$B_1 = (b_{2n-1}b_{2n-2} \cdots b_{n+1}b_n)_2; bA_0 = (a_{n-1}a_{n-2} \cdots a_1a_0)_2$$

Thuật toán nhân nhanh được dựa trên đẳng thức:

$$ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0$$

Điều này chỉ ra rằng phép nhân hai số nguyên $2n$ bit có thể thực hiện bằng cách dùng 3 phép nhân các số nguyên n bit và các phép cộng, trừ dịch chuyển. Như vậy, nếu $f(n)$ là tổng các phép toán nhị phân cần thiết để nhân hai số n bit thì

$$f(2n) = 3f(n) + Cn$$

Ba phép nhân các số nhị phân n bit cần $3f(n)$ phép toán nhị phân. Mỗi một phép toán cộng, trừ, dịch chuyển dùng một hằng số nhân với n lần chính là Cn .

Ví dụ 2. 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 sen kẽ các bà và không có hai 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 xếp một ghế thì một ghế để trống dành cho các ông), số cách xếp cho các bà là $2n!$ cách. Gọi số cách xếp cho các ông ứng với một cách xếp các bà là U_n ta được số cách xếp là

$$M_n = 2n! \times U_n. \text{ Vấn đề còn lại là tính số } U_n.$$

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 $i+1$ (các phép cộng được hiểu lấy modul n nghĩa là $n+1=1$). Mỗi cách xếp các ông được biểu diễn bằng một phép thế φ trên tập $\{1, 2, \dots, n\}$ với qui ước $\varphi(i) = j$ có nghĩa là ghế i được xếp cho ông j . Theo giả thiết φ phải thoả mãn:

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

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

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

$U_n = \bar{N} = n! - N_1 + N_2 + \dots$ trong đó N_k là tổng số tất cả các phép thế thoả mãn k tính chất lấy từ $2n$ tính chất đang xét. Cần chú ý rằng, không thể xảy ra đồng thời thoả mãn P_i và Q_i . Do đó trong các phép lấy ra k tính chất từ $2n$ tính chất đang xét cần thêm vào điều kiện: 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 ($k \leq n$) ta có $(n-k)!$ phép thế thoả mãn chúng. Từ đó ta nhận được $N_k = g(2n, k) (n-k)!$ và

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

Bây giờ chúng ta phải tính các hệ số $g(2n,k)$, $k = 1, 2, \dots, n$.

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

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

Giải: Khi lấy k phần tử, ta còn $n-k$ phần tử. Giữa $n-k$ phần tử còn lại có $n-k+1$ khoảng trống (kể cả hai 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 chọn cần tìm là $C(n-k+1, k)$.

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

Giải: Cố định phần tử a được chọn 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 phải lấy $k-1$ phần tử từ $n-3$ phần tử còn lại. Các phần tử này xem như kết quả của bài toán 1. Theo bài toán 1, số cách thuộc lớp kiểu này là $C(n-k-1, k-1)$.
- 2- Các cách mà a không được chọn, khi đó bỏ a đi và bài toán trở về bài toán 1 chọn k phần tử từ $n-1$ phần tử xếp trên đường thẳng. Theo bài toán 1 số cách xếp kiểu này là $C(n-k, k)$.

Vậy theo nguyên lý cộng số cách cần tìm là :

$$C(n-k-1, k-1) + C(n-k, k) = \frac{n}{n-k} C(n-k, k)$$

Từ kết quả của hai bài toán trên ta nhận được:

$$g(2n, k) = \frac{2n}{2n-k} C(2n-k, k) \text{ và số phân bố } U_n \text{ được tính bằng}$$

$$U_n = n! - \frac{2n}{2n-1} C(2n-1, 1)(n-1)! + \frac{2n}{2n-2} C(2n-2, 2)(n-2)! - \dots + (-1)^n \frac{2n}{n} C(2n, n)$$

Dưới đây là một số giá trị của U_n , một lần nữa chúng ta lại được quan sát hiện tượng bùng nổ tổ hợp.

n	2	3	4	5	6	7	8	9	10
U_n	0	1	2	13	80	579	4783	43387	439792

2.6- Phương pháp liệt kê

Việc tìm một công thức cho kết quả đếm ngay cả trong trường hợp công thức truy hồi không phải dễ dàng và lúc nào cũng thực hiện được. Cho đến nay còn 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 được 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 đó người ta có thể lập trình cho máy tính điện tử đếm hộ.

Để minh họa cho phương pháp liệt kê, ta xét một cấu hình tổ hợp nổi tiếng đó là các hình chữ nhật la tinh.

Giả sử S là tập gồm n phần tử. Không mất tính tổng quát ta giả sử $S = \{1, 2, \dots, n\}$. Một hình chữ nhật la tinh trên S là một bảng gồm 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 $p \leq n, q \leq n$. 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 cột nào chứa hai phần tử lặp lại. Hình chữ nhật la tinh dạng này được gọi là chuẩn nếu dòng đầu của nó là hoán vị $1, 2, \dots, n$.

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 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 nhận thấy rằng, số mất D_n là số hình la tinh chuẩn $2 \times n$, số phân bố U_n là số hình chữ nhật la tinh chuẩn $3 \times n$ với hai dòng đầu là:

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

Riordan J(1946) đã chứng minh công thức:

$$K(3, n) = \sum_{k=0}^m C(n, k) D_{n-k} D_k U_{n-2k} \text{ trong đó } m = \lfloor n/2 \rfloor, U_0 = 1.$$

Bài toán đếm với số dòng nhiều hơn đến nay vẫn chưa được giải quyết. Người ta mới chỉ đưa ra được một vài dạng tiệm cận của $L(p, n)$.

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 $1, 2, \dots, n$. 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 I_n là số các hình vuông như thế ta có $L(n, n) = n!(n-1)!I_n$

Việc tìm một công thức cho I_n đến nay vẫn bỏ ngỏ. Tuy nhiên ta có thể nhờ máy tính liệt kê tất cả các hình vuông 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
-------	---	---	---	---	----	------	----------

2.7. Bài toán tồn tại

Chúng ta đã giải quyết bài toán đếm số các cấu hình tổ hợp thoả mãn một tính chất nào đó, chẳng hạn như đếm số tổ hợp, số chỉnh hợp, hoặc số hoán vị. Trong những bài toán đó sự tồn tại của các cấu hình là hiển nhiên và công việc chính là chúng ta cần đếm số các cấu hình tổ hợp thoả mãn tính chất đặt ra. Tuy nhiên, trong nhiều bài toán tổ hợp, việc chỉ ra sự tồn tại của một cấu hình thoả mãn các tính chất cho trước đã là một việc làm hết sức khó khăn. Dạng bài toán như vậy được gọi là bài toán tồn tại.

2.7.1. Giới thiệu bài toán

Một bài toán tồn tại tổ hợp được xem như giải xong nếu hoặc chỉ ra một cách xây dựng cấu hình, hoặc chứng minh rằng chúng không tồn tại. Mọi khả năng đều không dễ dàng. Dưới đây là một số bài toán tồn tại tổ hợp nổi tiếng.

Bài toán 1. Bài toán về 36 sĩ quan

Bài toán này được Euler đề nghị với nội dung 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 úy, thượng úy, đại úy, 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 hàng ngang cũng như mỗi hàng dọc đều có đại diện của cả sáu trung đoàn và của 6 cấp bậc.

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

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

Một lời giải với $n=5$ là

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

Do lời giải bài toán có thể biểu diễn bởi hai hình vuông với các chữ cái la tinh hoa và la tinh thường nên bài toán tổng quát đặt ra còn được biết với tên gọi “*hình vuông la tinh trực giao*”. Trong hai ví dụ trên ta có hình vuông la tinh trực giao cấp 4 và 5.

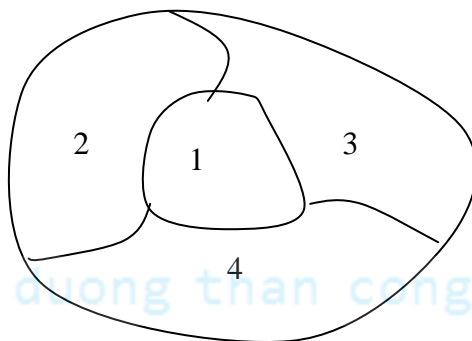
Euler đã mất rất nhiều công sức để tìm ra lời giải cho bài toán 36 sĩ quan thế nhưng ông đã không thành công. Vì vậy, ông giả thuyết là cách sắp xếp như vậy không tồn tại. Giả thuyết này đã được nhà toán học pháp Tarri chứng minh năm 1901 bằng cách duyệt tất cả mọi khả năng xếp. Euler căn cứ vào sự không tồn tại lời giải khi $n=2$ và $n=6$ còn đề ra giả thuyết tổng quát hơn là không tồn tại

hình vuông trục giao cấp $4n + 2$. Giả thuyết này đã tồn tại hai thế kỷ, mãi đến năm 1960 ba nhà toán học Mỹ là Bore, Parker, Srikanda mới chỉ ra được một lời giải với $n = 10$ và sau đó chỉ ra phương pháp xây dựng hình vuông trục giao cho mọi $n = 4k + 2$ với $k > 1$.

Tưởng chừng bài toán chỉ mang ý nghĩa thử thách trí tuệ con người thuần túy như một bài toán đố. Nhưng gần đây, người ta phát hiện những ứng dụng quan trọng của vấn đề trên vào qui hoạch, thực nghiệm và hình học xạ ảnh.

Bài toán 2. Bài toán 4 màu

Có nhiều bài toán mà nội dung của nó có thể giải thích được với bất kỳ ai, lời giải của nó ai cũng cố gắng thử tìm nhưng khó có thể tìm được. Ngoài định lý Fermat thì bài toán bốn màu cũng là một bài toán như vậy. Bài toán có thể được phát biểu như sau: Chứng minh rằng mọi bản đồ đều có thể tô bằng 4 màu sao cho không có hai nước láng giềng nào lại bị tô bởi cùng một màu. Trong đó, mỗi nước trên bản đồ được coi là một vùng liên thông, hai nước được gọi là láng giềng nếu chúng có chung đường biên giới là một đường liên tục.



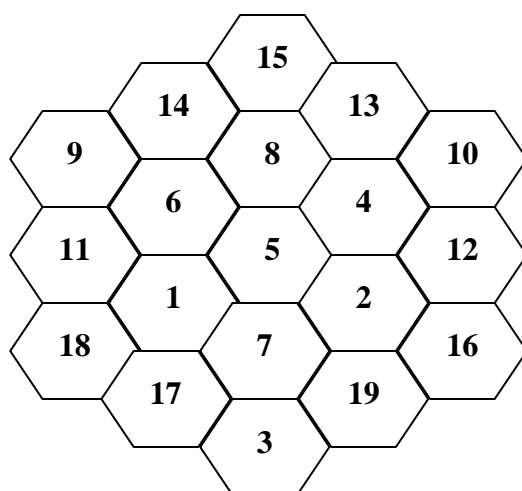
Hình 2.2. Bản đồ tô bởi ít nhất bốn màu

Con số bốn màu không phải là ngẫu nhiên. Người ta đã chứng minh được rằng mọi bản đồ đều được tô bởi số màu lớn hơn 4, còn với số màu ít hơn 4 thì không thể tô được, chẳng hạn bản đồ gồm 4 nước như trên hình 5.1 không thể tô được với số màu ít hơn 4.

Bài toán này xuất hiện vào những năm 1850 từ một lái buôn người Anh là Gazri khi tô bản đồ hành chính nước Anh đã cố gắng chứng minh rằng nó có thể tô bằng bốn màu. Sau đó, năm 1852, ông đã viết thư cho De Morgan đề thông báo về giả thuyết này. Năm 1878, keli trong một bài báo đăng ở tuyển tập các công trình nghiên cứu của Hội toán học Anh có hỏi rằng bài toán này đã được giải quyết hay chưa? Từ đó bài toán trở nên nổi tiếng, trong suốt hơn một thế kỷ qua, nhiều nhà toán học đã cố gắng chứng minh giả thuyết này. Tuy vậy, mãi tới năm 1976 hai nhà toán học Mỹ là K. Appel và W. Haken mới chứng minh được nó nhờ máy tính điện tử.

Bài toán 3. Hình lục giác thần bí

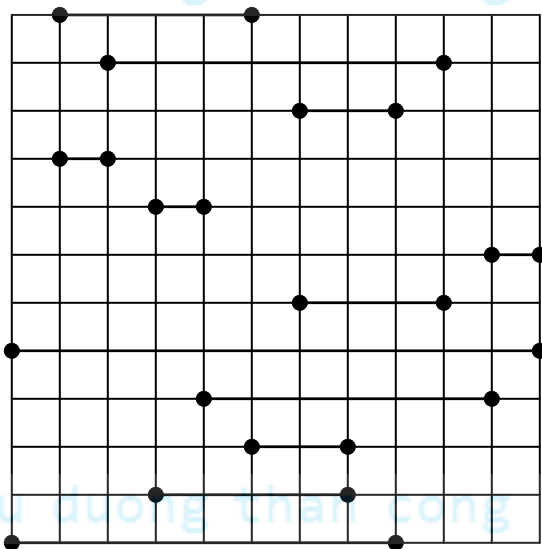
Năm 1890 Clifford Adams đề ra bài toán hình lục giác thần bí sau: trên 19 ô lục giác (như hình 2.3) hãy điền các số từ 1 đến 19 sao cho tổng theo 6 hướng của lục giác là bằng nhau (và đều bằng 38). Sau 47 năm trời kiên nhẫn cuối cùng Adams cũng đã tìm được lời giải. Sau đó vì sơ ý đánh mất bản thảo ông đã tốn thêm 5 năm để khôi phục lại. Năm 1962 Adams đã công bố lời giải đó. Nhưng thật không thể ngờ được đó là lời giải duy nhất.



Hình 2.3. Hình lục giác thần bí

Bài toán 3. Bài toán chọn $2n$ điểm trên lưới $n \times n$ điểm

Cho một lưới gồm $n \times n$ điểm. Hỏi có thể chọn trong số chúng $2n$ điểm sao cho không có ba điểm nào được chọn là thẳng hàng? Hiện nay người ta mới biết được lời giải của bài toán này khi $n \leq 15$. Hình 3.3 cho một lời giải với $n = 12$.



Hình 2.4. Một lời giải với $n = 12$.

2.7.2. Phương pháp phản chứng

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

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

Giải: Điều kiện cần và đủ để 3 đoạn là cạnh của một tam giác là tổng của hai cạnh phải lớn hơn một cạnh. Ta sắp các đoạn thẳng theo thứ tự tăng dần của độ dài a_1, a_2, \dots, a_7 và chứng minh rằng dãy đã xếp luôn tìm được 3 đoạn mà tổng của hai đoạn đầu lớn hơn đoạn cuối. Để chứng minh, ta giả sử không tìm được ba đoạn nào mà tổng của hai đoạn nhỏ hơn một đoạn, nghĩa là các bất đẳng thức sau đồng thời xảy ra:

$$a_1 + a_2 \leq a_3 \Rightarrow a_3 \geq 20 \text{ (vì } a_1, a_2 \geq 10 \text{)}$$

$$a_2 + a_3 \leq a_4 \Rightarrow a_4 \geq 30 \text{ (vì } a_2 \geq 10, a_3 \geq 20 \text{)}$$

$$a_3 + a_4 \leq a_5 \Rightarrow a_5 \geq 50 \text{ (vì } a_3 \geq 20, a_4 \geq 30 \text{)}$$

$$a_4 + a_5 \leq a_6 \Rightarrow a_6 \geq 80 \text{ (vì } a_4 \geq 30, a_5 \geq 50 \text{)}$$

$$a_5 + a_6 \leq a_7 \Rightarrow a_7 \geq 130 \text{ (vì } a_5 \geq 50, a_6 \geq 80 \text{)}$$

\Rightarrow Mâu thuẫn (bài toán được giải quyết).

Ví dụ 2. Các đỉnh của một thập giác đều được đánh số bởi các số nguyên $0, 1, \dots, 9$ một cách tùy 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, \dots, x_{10} là các số gán cho các đỉnh của thập giác đều. Giả sử ngược lại ta không tìm được 3 đỉnh liên tiếp nào thỏa mãn khẳng định trên. Khi đó ta có

$$k_1 = x_1 + x_2 + x_3 \leq 13$$

$$k_2 = x_2 + x_3 + x_4 \leq 13$$

$$k_3 = x_3 + x_4 + x_5 \leq 13$$

$$k_4 = x_4 + x_5 + x_6 \leq 13$$

$$k_5 = x_5 + x_6 + x_7 \leq 13$$

$$k_6 = x_6 + x_7 + x_8 \leq 13$$

$$k_7 = x_7 + x_8 + x_9 \leq 13$$

$$k_8 = x_8 + x_9 + x_{10} \leq 13$$

$$k_9 = x_9 + x_{10} + x_1 \leq 13$$

$$k_{10} = x_{10} + x_1 + x_2 \leq 13$$

$$\Rightarrow 130 \geq k_1 + k_2 + \dots + k_{10} = 3(x_1 + x_2 + \dots + x_{10})$$

$$= 3(0 + 1 + 2 + \dots + 9)$$

$$= 135 \Rightarrow \text{Mâu thuẫn vì một số bằng 135 không thể hơn 130. Khẳng}$$

định chứng minh.

2.7.3 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 nhiều hơn n đối tượng vào n hộp thì luôn tìm được một cái hộp chứa không ít hơn 2 đối tượng.

Chứng minh. Việc chứng minh nguyên lý trên chỉ cần sử dụng một lập luận phản chứng đơn giản. Giả sử không tìm được một hộp nào chứa không ít hơn hai đối tượng. Điều đó nghĩa là mỗi hộp

không chứa quá một đối tượng. Từ đó suy ra tổng các đối tượng không vượt quá n trái với giả thiết bài toán là có nhiều hơn n đối tượng được xếp vào chúng.

Ví dụ 1. Trong bất kỳ một nhóm có 367 người thế nào cũng có ít nhất hai người có cùng ngày sinh.

Giải: Vì một năm có nhiều nhất 366 ngày. Như vậy, theo nguyên lý Dirichlet thì có ít nhất một ngày có hai người cùng một ngày sinh.

Ví dụ 2. Trong bất kỳ 27 từ tiếng Anh nào cũng đều có ít nhất hai từ cùng bắt đầu bằng một chữ cái.

Giải: Vì bảng chữ cái tiếng Anh chỉ có 26 chữ cái. Nên theo nguyên lý Dirichlet tồn tại ít nhất 2 từ sẽ bắt đầu bởi cùng một chữ cái.

Ví dụ 3. Bài thi các môn học cho sinh viên được chấm theo thang điểm 100. Hỏi lớp phải có ít nhất bao nhiêu sinh viên để có ít nhất hai sinh viên được nhận cùng một điểm.

Giải: Cần có ít nhất 102 sinh viên vì thang điểm tính từ 0 . . 100 gồm 101 số. Do vậy, theo nguyên lý Dirichlet muốn có 2 sinh viên nhận cùng một điểm thì lớp phải có ít nhất là $101 + 1 = 102$ sinh viên.

Nguyên lý Dirichlet tổng quát. Nếu đem xếp n đối tượng vào k hộp thì luôn tìm được một hộp chứa ít nhất $\lceil n/k \rceil$ đối tượng.

Nguyên lý trên được nhà toán học người Đức Dirichlet đề xuất từ thế kỷ 19 và ông đã áp dụng để giải nhiều bài toán tổ hợp.

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

Giải: Một năm có 12 tháng. Xếp tất cả những người sinh nhật vào cùng một nhóm. Theo nguyên lý Dirichlet ta có ít nhất $\lceil 100/12 \rceil = 9$ người cùng sinh nhật một tháng.

Ví dụ 5. Có năm loại học bổng khác nhau để phát cho sinh viên. Hỏi phải có ít nhất bao nhiêu sinh viên để chắc chắn có 5 người được nhận học bổng như nhau.

Giải. Số sinh viên ít nhất để có 5 sinh viên cùng được nhận một loại học bổng là số n thỏa mãn $\lceil n/5 \rceil > 5$. Số nguyên bé nhất thỏa mãn điều kiện trên là $n = 25 + 1 = 26$. Như vậy phải có ít nhất 26 sinh viên để có ít nhất 5 sinh viên cùng được nhận một loại học bổng.

Ví dụ 6. Trong một tháng có 30 ngày một đội bóng chày chơi ít nhất mỗi ngày một trận, nhưng cả tháng chơi không quá 45 trận. Hãy chỉ ra 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 số trận thi đấu cho tới ngày thứ j của đội. Khi đó

$$a_1, a_2, \dots, a_{30}$$

là dãy tăng của các số nguyên dương và $1 \leq a_j \leq 45$. Suy ra dãy

$$a_1 + 14, a_2 + 14, \dots, a_{30} + 14 \text{ cũng là dãy tăng các số nguyên dương và}$$

$$15 \leq a_j \leq 59$$

Như vậy, dãy 60 số nguyên dương

$$a_1, a_2, \dots, a_{30}, a_1 + 14, a_2 + 14, \dots, a_{30} + 14 \text{ trong đó tất cả các số đều nhỏ hơn hoặc bằng 59.}$$

Theo nguyên lý Dirichlet thì phải tồn tại ít nhất hai số trong số hai số nguyên này bằng nhau. Vì các số a_1, a_2, \dots, a_{30} là đôi một khác nhau và $a_1 + 14, a_2 + 14, \dots, a_{30} + 14$ cũng đôi một khác nhau. Nên ta suy ra phải tồn tại 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 thứ i .

2.8. Những nội dung cần ghi nhớ

Bạn đọc cần ghi nhớ một số kiến thức quan trọng sau:

- ✓ Những nguyên lý đếm cơ bản: nguyên lý cộng, nguyên lý nhân & nguyên lý bù trừ.
- ✓ Sử dụng những nguyên lý cơ bản tron đếm các hoán vị, tổ hợp.
- ✓ Hiểu phương pháp cách giải quyết bài toán đếm bằng hệ thức truy hồi.
- ✓ Nắm vững cách thức qui một bài toán đếm về những bài toán con.
- ✓ Cách giải phổ biến cho bài toán tồn tại là sử dụng phương pháp phản chứng hoặc sử dụng nguyên lý Dirichlet.

cuu duong than cong . com

cuu duong than cong . com

BÀI TẬP CHƯƠNG 2

1. Xâu thuận nghịch độc là một xâu khi viết theo thứ tự ngược lại cũng bằng chính nó. Hãy đếm số xâu nhị phân có độ dài n là thuận nghịch độc.
2. Cô dâu và chú rể mời bốn bạn đứng thành một hàng để chụp ảnh. Hỏi có bao nhiêu cách xếp hàng nếu:
 - a) Cô dâu đứng cạnh chú rể
 - b) Cô dâu không đứng cạnh chú rể
 - c) Cô dâu đứng ở phía bên phải chú rể
3. Có bao nhiêu xâu nhị phân độ dài 10 có năm số 0 liên nhau hoặc năm số 1 liên nhau.
4. Có bao nhiêu xâu nhị phân độ dài bằng 8 có 3 số 0 liên nhau hoặc 4 số 1 liên nhau.
5. Mỗi sinh viên lớp toán học rời rạc hoặc giỏi toán hoặc giỏi tin học hoặc giỏi cả hai môn này. Trong lớp có bao nhiêu sinh viên nếu 38 người giỏi tin (kể cả người giỏi cả hai môn), 23 người giỏi toán (kể cả người giỏi cả hai môn), và 7 người giỏi cả hai môn.
6. Chứng tỏ rằng, trong $n+1$ số nguyên dương không vượt quá $2n$ tồn tại ít nhất một số chia hết cho một số khác.
7. Chứng minh rằng, trong dãy gồm $n^2 + 1$ số thực phân biệt đều có một dãy con dài $n+1$ hoặc thực sự tăng, hoặc thực sự giảm.
8. Giả sử trong một nhóm 6 người mỗi cặp hai hoặc là bạn, hoặc là thù. Chứng tỏ rằng trong nhóm có ba người là bạn của nhau hoặc là kẻ thù của nhau.
9. Hãy chỉ ra rằng, trong 102 người có chiều cao khác nhau đứng thành một hàng có thể tìm được 11 người có chiều cao tăng dần hoặc giảm dần mà không cần thay đổi thứ tự của họ trong hàng.
10. Một đô vật tay tham gia thi đấu giành chức vô địch trong 75 giờ. Mỗi giờ anh ta thi đấu ít nhất một trận, nhưng toàn bộ anh ta không thi đấu quá 125 trận. Chứng tỏ rằng, có những giờ liên tiếp anh ta thi đấu 24 trận.
11. Một nhân viên bắt đầu làm việc tại công ty từ năm 1987 với mức lương khởi điểm là 50000 đô la. Hàng năm anh ta được nhận thêm 1000 đô la và 5% lương của năm trước.
 - a) Hãy thiết lập hệ thức truy hồi tính lương của nhân viên đó n năm sau năm 1987.
 - b) Lương vào năm 1995 của anh ta là bao nhiêu?
 - c) Hãy tìm công thức tường minh tính lương của nhân viên này n năm sau năm 1987.
12. Tìm hệ thức truy hồi cho số hoán vị của tập n phần tử. Dùng hệ thức truy hồi đó tính hoán vị của tập n phần tử.
13. Một máy bán tem tự động chỉ nhận các đồng xu một đô la và các loại tờ tiền 1 đô la và 5 đô la.
 - a) Hãy tìm hệ thức truy hồi tính số cách đặt n đô la vào trong máy bán hàng, trong đó thứ tự các đồng xu, các tờ tiền là quan trọng.
 - b) Tìm các điều kiện đầu.
 - c) Có bao nhiêu cách đặt 10 đô la vào máy để mua một bộ tem.

14. Giải các hệ thức truy hồi với các điều đầu sau:

- a) $a_n = a_{n-1} + 6a_{n-2}$ với $n \geq 2$, $a_0 = 3$, $a_1 = 6$.
b) $a_n = 7a_{n-1} - 6a_{n-2}$ với $n \geq 2$, $a_0 = 2$, $a_1 = 1$.
c) $a_n = 6a_{n-1} - 8a_{n-2}$ với $n \geq 2$, $a_0 = 4$, $a_1 = 10$.
d) $a_n = 2a_{n-1} - a_{n-2}$ với $n \geq 2$, $a_0 = 4$, $a_1 = 1$.
e) $a_n = a_{n-2}$ với $n \geq 2$, $a_0 = 5$, $a_1 = -1$.
f) $a_n = -6a_{n-1} - 9a_{n-2}$ với $n \geq 2$, $a_0 = 3$, $a_1 = -3$.
g) $a_{n+2} = -4a_{n+1} + 5a_n$ với $n \geq 0$, $a_0 = 2$, $a_1 = 8$.

15. Tìm các nghiệm đặc trưng của hệ thức truy hồi tuyến tính thuần nhất

- a) $a_n = 2a_{n-1} - 2a_{n-2}$
b) Tìm nghiệm thỏa mãn hệ thức truy hồi trên và các điều kiện đầu $a_0 = 1$, $a_1 = 2$.

16. a) Tìm nghiệm đặc trưng của hệ thức truy hồi tuyến tính thuần nhất $a_n = an - 4$

- b) Tìm nghiệm thỏa mãn hệ thức truy hồi trên và các điều kiện đầu $a_0 = 1$, $a_1 = 0$, $a_2 = -1$, $a_3 = 1$.

17. Một báo cáo về thị trường máy tính cá nhân cho biết có 65000 người sẽ mua modem cho máy tính của họ trong năm tới, 1 250 000 người sẽ mua ít nhất một sản phẩm phần mềm. Nếu báo cáo này nói rằng 1.450.000 người sẽ mua hoặc là modem hoặc là ít nhất một sản phẩm phần mềm thì sẽ có bao nhiêu người sẽ mua cả modem và mua ít nhất một sản phẩm phần mềm.

cuu duong than cong . com

CHƯƠNG 3. BÀI TOÁN LIỆT KÊ & BÀI TOÁN TỒN TẠI

Đối với một bài toán, khi chưa tìm được giải thuật tốt để giải thì liệt kê là biện pháp cuối cùng để thực hiện với sự hỗ trợ của máy tính. Có thể nói, liệt kê là phương pháp phổ dụng nhất để giải quyết một bài toán trên máy tính. Trái lại, bài toán tồn tại chỉ cần chỉ ra được bài toán có nghiệm hay không có nghiệm và thường là những bài toán khó. Nhiều bài toán tồn tại đã được phát biểu trong nhiều thập kỉ nhưng vẫn chưa được giải quyết. Giải quyết được chúng sẽ thúc đẩy sự phát triển của nhiều ngành toán học. Nội dung chính của chương này tập chung giải quyết những vấn đề cơ bản sau:

- ✓ Giới thiệu bài toán liệt kê và bài toán tồn tại.
- ✓ Giải quyết bài toán liệt kê bằng phương pháp sinh.
- ✓ Giải quyết bài toán liệt kê bằng phương pháp quay lui dựa trên giải thuật đệ qui.
- ✓ Phương pháp phản chứng giải quyết bài toán tồn tại.
- ✓ Nguyên lý Dirichlet giải quyết bài toán tồn tại.

Bạn đọc có thể tìm thấy cách giải nhiều bài toán liệt kê và bài toán tồn tại hay trong các tài liệu [1] và [2] trong tài liệu tham khảo.

3.1- Giới thiệu bài toán

Bài toán đưa ra danh sách tất cả các cấu hình tổ hợp có thể có được gọi là bài toán liệt kê tổ hợp. Khác với bài toán đếm là tìm kiếm một công thức cho lời giải, bài toán liệt kê lại cần xác định một thuật toán để theo đó có thể xây dựng được lần lượt tất cả các cấu hình cần quan tâm. Một thuật toán liệt kê phải đảm bảo hai nguyên tắc:

- ☐ Không được lặp lại một cấu hình
- ☐ Không được bỏ sót một cấu hình

Ví dụ 1. Cho tập hợp các số a_1, a_2, \dots, a_n và số M . Hãy tìm tất cả các tập con k phần tử của dãy số $\{a_n\}$ sao cho tổng số các phần tử trong tập con đó đúng bằng M .

Giải: Như chúng ta đã biết, số các tập con k phần tử của tập gồm n phần tử là $C(n, k)$. Như vậy chúng ta cần phải duyệt trong số $C(n, k)$ tập k phần tử để lấy ra những tập có tổng các phần tử đúng bằng M . Vì không thể xác định được có bao nhiêu tập k phần tử từ tập n phần tử có tổng các phần tử đúng bằng M nên chúng ta chỉ còn cách liệt kê các cấu hình thỏa mãn điều kiện đã cho.

Ví dụ 2. Một thương nhân đi bán hàng tại tám thành phố. Chị ta có thể bắt đầu hành trình của mình tại một thành phố nào đó nhưng phải qua 7 thành phố kia theo bất kỳ thứ tự nào mà chị ta muốn. Hãy chỉ ra lộ trình ngắn nhất mà chị ta có thể đi.

Giải: Vì thành phố xuất phát đã được xác định. Do vậy thương nhân có thể chọn tùy ý 7 thành phố còn lại để hành trình. Như vậy, tất cả số hành trình của thương nhân có thể đi qua là $7! = 5040$ cách. Tuy nhiên trong 5040 cách chúng ta phải duyệt toàn bộ để chỉ ra một hành trình là ngắn nhất.

Có thể nói phương pháp liệt kê là biện pháp cuối cùng nhưng cũng là biện pháp phổ dụng nhất để giải quyết các bài toán tổ hợp. 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 như số mất thứ tự D_n , số phân bố U_n , số hình vuông la tinh ln), ta giả sử cần 1 giây để liệt kê một cấu hình thì chúng ta cũng cần 31 năm mới giải quyết xong. Tuy nhiên với sự phát triển nhanh chóng của máy tính, bằng phương pháp liệt kê, nhiều bài toán khó của lý thuyết tổ hợp đã được giải quyết và góp phần thúc đẩy sự phát triển của nhiều ngành toán học.

3.2. Đệ qui

3.2.1. Định nghĩa bằng đệ qui

Trong thực tế, chúng ta gặp rất nhiều đối tượng mà khó có thể định nghĩa nó một cách tường minh, nhưng lại dễ dàng định nghĩa đối tượng qua chính nó. Kỹ thuật định nghĩa đối tượng qua chính nó được gọi là kỹ thuật đệ qui (recursion). Đệ qui được sử dụng rộng rãi trong khoa học máy tính và lý thuyết tính toán. Các giải thuật đệ qui đều được xây dựng thông qua hai bước: bước phân tích và bước thay thế ngược lại.

Ví dụ 1. Để tính tổng $S(n) = 1 + 2 + \dots + n$, chúng ta có thể thực hiện thông qua hai bước như sau:

Bước phân tích:

- Để tính toán được $S(n)$ trước tiên ta phải tính toán trước $S(n-1)$ sau đó tính $S(n) = S(n-1) + n$.
- Để tính toán được $S(n-1)$, ta phải tính toán trước $S(n-2)$ sau đó tính $S(n-1) = S(n-2) + n-1$.
-
- Để tính toán được $S(2)$, ta phải tính toán trước $S(1)$ sau đó tính $S(2) = S(1) + 2$.
- Và cuối cùng $S(1)$ chúng ta có ngay kết quả là 1.

Bước thay thế ngược lại:

Xuất phát từ $S(1)$ thay thế ngược lại chúng ta xác định $S(n)$:

- $S(1) = 1$
- $S(2) = S(1) + 2$
- $S(3) = S(2) + 3$
-
- $S(n) = S(n-1) + n$

Ví dụ 2. Định nghĩa hàm bằng đệ qui

Hàm $f(n) = n!$

Dễ thấy $f(0) = 1$.

Vì $(n+1)! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n(n+1) = n! \cdot (n+1)$, nên ta có:

$f(n+1) = (n+1) \cdot f(n)$ với mọi n nguyên dương.

Ví dụ 3. Tập hợp định nghĩa bằng đệ qui

Định nghĩa đệ qui tập các chuỗi: Giả sử Σ^* là tập các chuỗi trên bộ chữ cái Σ . Khi đó Σ^* được định nghĩa bằng đệ qui như sau:

- $\lambda \in \Sigma^*$, trong đó λ là chuỗi rỗng
- $wx \in \Sigma^*$ nếu $w \in \Sigma^*$ và $x \in \Sigma^*$

3.2.2. Giải thuật đệ qui

Một thuật toán được gọi là đệ qui nếu nó giải bài toán bằng cách rút gọn bài toán ban đầu thành bài toán tương tự như vậy sau một số hữu hạn lần thực hiện. Trong mỗi lần thực hiện, dữ liệu đầu vào tiệm cận tới tập dữ liệu dừng.

Ví dụ : để giải quyết bài toán tìm ước số chung lớn nhất của hai số nguyên dương a và b với $b > a$, ta có thể rút gọn về bài toán tìm ước số chung lớn nhất của $(b \bmod a)$ và a vì $USCLN(b \bmod a, a) = USCLN(a, b)$. Dãy các rút gọn liên tiếp có thể đạt được cho tới khi đạt điều kiện dừng $USCLN(0, a) = USCLN(a, b) = a$. Dưới đây là ví dụ về một số thuật toán đệ qui thông dụng.

Thuật toán 1: Tính a^n bằng giải thuật đệ qui, với mọi số thực a và số tự nhiên n .

```
double power( float a, int n ){
    if ( n == 0)
        return(1);
    return(a * power(a, n-1));
}
```

Thuật toán 2: Thuật toán đệ qui tính ước số chung lớn nhất của hai số nguyên dương a và b .

```
int USCLN( int a, int b){
    if (a == 0)
        return(b);
    return(USCLN( b % a, a));
}
```

Thuật toán 3: Thuật toán đệ qui tính $n!$

```
long factorial( int n){
    if (n == 1)
        return(1);
    return(n * factorial(n-1));
}
```

Thuật toán 4: Thuật toán đệ qui tính số fibonacci thứ n

```
int fibonacci( int n) {
    if (n == 0) return(0);
    else if (n == 1) return(1);
    return(fibonacci(n-1) + fibonacci(n-2));
}
```

3.3- Phương pháp sinh

Phương pháp sinh có thể áp dụng để giải các bài toán liệt kê tổ hợp đặt ra nếu như hai điều kiện sau được thực hiện:

- (i) 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 tổ hợp đầu tiên và cuối cùng trong thứ tự đã được xác định.
- (ii) Xây dựng được thuật toán từ cấu hình chưa phải là cuối cùng đang có để đưa ra cấu hình kế tiếp sau nó.

Ta gọi thuật toán trong điều kiện (ii) là thuật toán sinh kế tiếp. Rõ ràng thuật toán này chỉ thực hiện được khi có một cấu hình được xác định theo điều kiện (i). Giả sử một bài toán đều thỏa mãn các điều kiện trên, khi đó phương pháp sinh kế tiếp có thể được mô tả bằng thủ tục như sau:

```
void Generate(void){  
    <Xây dựng cấu hình ban đầu>;  
    stop = false  
    while (not stop) {  
        <Đưa ra cấu hình đang có>;  
        Sinh_Kế_Tiếp;  
    }  
}
```

Trong đó Sinh_Kế_Tiếp là thủ tục sinh cấu hình kế tiếp từ cấu hình ban đầu. Nếu cấu hình là cấu hình cuối cùng, thủ tục này cần gán giá trị True cho stop, 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 là một số ví dụ điển hình mô tả thuật toán sinh kế tiếp.

Ví dụ 1. Liệt kê tất cả các dãy nhị phân độ dài n .

Giải: Viết dãy nhị phân dưới dạng $b_1b_2..b_n$, trong đó $b_i \in \{0, 1\}$. Xem mỗi dãy nhị phân $b = b_1b_2..b_n$ 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 từ điển được xác định như sau:

Ta nói dãy nhị phân $b = b_1b_2..b_n$ đi trước dãy nhị phân $b' = b'_1b'_2..b'_n$ theo thứ tự từ điển và kí hiệu $b < b'$ nếu $p(b) < p(b')$.

Ví dụ với $n=4$, các xâu nhị phân độ dài 4 được liệt kê theo thứ tự từ điển là:

b	p(b)	b	p(b)
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12

0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

Như vậy, dãy đầu tiên là 0000 dãy cuối cùng là 1111. Nhận xét rằng, nếu xâu nhị phân chứa toàn bit 1 thì quá trình liệt kê kết thúc, trái lại dãy kế tiếp sẽ nhận được bằng cách cộng thêm 1 (theo modul 2 có nhớ) vào dãy hiện tại. Từ đó ta nhận được qui tắc sinh kế tiếp như sau:

- Tìm i đầu tiên từ phải sang trái ($i=n, n-1, \dots, 1$) thỏa mãn $b_i=0$.
- Gán lại $b_i=1$ và $b_j=0$ với tất cả $j>i$. Dãy thu được là dãy cần tìm.

Ví dụ ta có xâu nhị phân độ dài 10: 1100111011. Ta có $i=8$, ta đặt $b_8=1$, $b_9, b_{10}=0$ ta được xâu nhị phân kế tiếp: 1100111100.

Thuật toán sinh kế tiếp được mô tả trong thủ tục sau:

```
void Next_Bit_String( int *B, int n ){
    i = n;
    while (bi == 1 ) {
        bi = 0
        i = i-1;
    }
    bi = 1;
}
```

Dưới đây là chương trình liệt kê các xâu nhị phân có độ dài n .

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#include <conio.h>
#define MAX 100
#define TRUE 1
#define FALSE 0
int Stop, count;
void Init(int *B, int n){
    int i;
    for(i=1; i<=n ;i++)
        B[i]=0;
    count =0;
}
```

```

void Result(int *B, int n){
    int i;count++;
    printf("\n Xau nhi phan thu %d:",count);
    for(i=1; i<=n;i++)
        printf("%3d", B[i]);
}

void Next_Bits_String(int *B, int n){
    int i = n;
    while(i>0 && B[i]){
        B[i]=0; i--;
    }
    if(i==0 )
        Stop=TRUE;
    else
        B[i]=1;
}

void Generate(int *B, int n){
    int i;
    Stop = FALSE;
    while (!Stop) {
        Result(B,n);
        Next_Bits_String(B,n);
    }
}

void main(void){
    int i, *B, n;clrscr();
    printf("\n Nhap n=");scanf("%d",&n);
    B =(int *) malloc(n*sizeof(int));
    Init(B,n);Generate(B,n);free(B);getch();
}

```

Ví dụ 2. Liệt kê tập con m phần tử của tập n phần tử. Cho $X = \{ 1, 2, \dots, n \}$. Hãy liệt kê tất cả các tập con k phần tử của X ($k \leq n$).

Giải: Mỗi tập con của tập hợp X có thể biểu diễn bằng bộ có thứ tự gồm k thành phần $a = (a_1 a_2 \dots a_k)$ thoả mãn $1 \leq a_1 \leq a_2 \leq \dots \leq a_k \leq n$.

Trên tập các tập con k phần tử của X có thể xác định nhiều thứ tự khác nhau. Thứ tự dễ nhìn thấy nhất là thứ tự từ điển được định nghĩa như sau:

Ta nói tập con $a = a_1 a_2 \dots a_k$ đi trước tập con $a' = a'_1 a'_2 \dots a'_k$ trong thứ tự từ điển và ký hiệu là $a < a'$, nếu tìm được chỉ số j ($1 \leq j \leq k$) sao cho

$$a_1 = a'_1, a_2 = a'_2, \dots, a_{j-1} = a'_{j-1}, a_j < a'_j.$$

Chẳng hạn $X = \{1, 2, 3, 4, 5\}$, $k = 3$. 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 $(1, 2, \dots, k)$ và tập con cuối cùng là $(n-k+1, n-k+2, \dots, n)$. Giả sử $a = (a_1, a_2, \dots, a_k)$ là tập con hiện tại và chưa phải là cuối cùng, khi đó có thể chứng minh được rằng tập con kế tiếp trong thứ tự từ điển có thể được xây dựng bằng cách thực hiện các qui tắc biến đổi sau đối với tập con đang có.

- ☐ Tìm từ bên phải dãy a_1, a_2, \dots, a_k phần tử $a_i \neq n - k + i$
- ☐ Thay a_i bởi $a_i + 1$,
- ☐ Thay a_j bởi $a_i + j - i$, với $j := i+1, i+2, \dots, k$

Chẳng hạn với $n = 6, k = 4$. Giả sử ta đang có tập con $(1, 2, 5, 6)$, cần xây dựng tập con kế tiếp nó trong thứ tự từ điển. Duyệt từ bên phải ta nhận được $i = 2$, thay a_2 bởi $a_2 + 1 = 2 + 1 = 3$. Duyệt j từ $i + 1 = 3$ cho đến k , ta thay thế $a_3 = a_2 + 3 - 2 = 3 + 3 - 2 = 4$, $a_4 = a_2 + 4 - 2 = 3 + 4 - 2 = 5$ ta nhận được tập con kế tiếp là $(1, 3, 4, 5)$.

Với qui tắc sinh như trên, chúng ta có thể mô tả bằng thuật toán sau:

Thuật toán liệt kê tập con kế tiếp m phần tử của tập n phần tử:

```
void Next_Combination( int *A, int m){
    i = m;
    while ( a_i == m-n+i)
        i = i -1;
    a_i = a_i + 1;
    for ( j = i+1; j <= m; j++)
```

$$a_j = a_i + j - i;$$

}

Văn bản chương trình liệt kê tập các tập con m phần tử của tập n phần tử được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#define TRUE 1
#define FALSE 0
#define MAX 100
int n, k, count, C[MAX], Stop;
void Init(void){
    int i;
    printf("\n Nhap n="); scanf("%d", &n);
    printf("\n Nhap k="); scanf("%d", &k);
    for(i=1; i<=k; i++)
        C[i]=i;
}
void Result(void){
    int i; count++;
    printf("\n Tap con thu %d:", count);
    for(i=1; i<=k; i++)
        printf("%3d", C[i]);
}
void Next_Combination(void){
    int i,j;
    i = k;
    while(i>0 && C[i]==n-k+i)
        i--;
    if(i>0) {
        C[i]= C[i]+1;
        for(j=i+1; j<=k; j++)
            C[j]=C[i]+j-i;
    }
    else Stop = TRUE;
}
```

```

void Combination(void){
    Stop=FALSE;
    while (!Stop){
        Result();
        Next_Combination();
    }
}

void main(void){
    clrscr(); Init();Combination();getch();
}

```

Ví dụ 3. Liệt kê các hoán vị của tập n phần tử. Cho $X = \{ 1, 2, \dots, n \}$. Hãy liệt kê các hoán vị từ n phần tử của X .

Giải : Mỗi hoán vị từ n phần tử của X có thể biểu diễn bởi bộ có thứ tự n thành phần

$a = (a_1, a_2, \dots, a_n)$ thoả mãn $a_i \in X, i = 1, 2, \dots, n, a_p \neq a_q, p \neq q$.

Trên tập các hoán vị từ n phần tử của X có thể xác định nhiều thứ tự khác nhau. Tuy nhiên, thứ tự dễ thấy nhất là thứ tự từ điển được định nghĩa như sau:

Ta nói hoán vị $a = a_1 a_2 \dots a_n$ đi trước hoán vị $a' = a'_1 a'_2 \dots a'_n$ trong thứ tự từ điển và ký hiệu là $a < a'$, nếu tìm được chỉ số k ($1 \leq k \leq n$) sao cho

$a_1 = a'_1, a_2 = a'_2, \dots, a_{k-1} = a'_{k-1}, a_k < a'_k$.

Chẳng hạn $X = \{ 1, 2, 3, 4 \}$. Các hoán vị các phần tử của X được liệt kê theo thứ tự từ điển như sau:

1	2	3	4	3	1	2	4
1	2	4	3	3	1	4	2
1	3	2	4	3	2	1	4
1	3	4	2	3	2	4	1
1	4	2	3	3	4	1	2
1	4	3	2	3	4	2	1
2	1	3	4	4	1	2	3
2	1	4	3	4	1	3	2
2	3	1	4	4	2	1	3
2	3	4	1	4	2	3	1
2	4	1	3	4	3	1	2
2	4	3	1	4	3	2	1

Như vậy, hoán vị đầu tiên trong thứ tự từ điển là $(1, 2, \dots, n)$ và hoán vị cuối cùng là $(n, n-1, \dots, 1)$. Giả sử $a = a_1 a_2 \dots a_n$ là một hoán vị chưa phải là cuối cùng. Khi đó ta có thể chứng minh được rằng,

hoán vị kế tiếp trong thứ tự từ điển có thể xây dựng bằng cách thực hiện các qui tắc biến đổi sau đối với hoán vị hiện tại:

- ☐ Tìm từ phải qua trái hoán vị có chỉ số j đầu tiên thỏa mãn $a_j < a_{j+1}$ (hay j là chỉ số lớn nhất để $a_j < a_{j+1}$);
- ☐ 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 .

Chẳng hạn ta đang có hoán vị (3, 6, 2, 5, 4, 1), cần xây dựng hoán vị kế tiếp theo thứ tự từ điển. Ta duyệt từ $j = n-1$ sang bên trái để tìm j đầu tiên thỏa mãn $a_j < a_{j+1}$ ta nhận được $j=3$ ($a_3=2 < a_4=5$). Số nhỏ nhất còn lớn hơn a_3 trong các số bên phải a_3 là a_5 ($a_5=4$). Đổi chỗ a_3 cho a_5 ta thu được (3, 6, 4, 5, 2, 1), lật ngược đoạn từ a_4 đến a_6 ta nhận được (3, 6, 4, 1, 2, 5).

Từ đó thuật toán sinh kế tiếp có thể được mô tả bằng thủ tục sau:

Thuật toán sinh hoán vị kế tiếp:

```
void Next_Permutation( int *A, int n){
    int j, k, r, s, temp;
    j = n;
    while (a_j > a_{j+1} )
        j = j - 1;
    k = n;
    while (a_j > a_k )
        k = k - 1;
    temp = a_j; a_j = a_k; a_k = temp;
    r = j + 1; s = n;
    while ( r < s ) {
        temp = a_r; a_r = a_s; a_s = temp;
        r = r + 1; s = s - 1;
    }
}
```

Văn bản chương trình liệt kê các hoán vị của tập hợp gồm n phần tử như sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAX 20
#define TRUE 1
#define FALSE 0
```

```

int P[MAX], n, count, Stop;
void Init(void){
    int i;count =0;
    printf("\n Nhap n=");scanf("%d", &n);
    for(i=1; i<=n; i++)
        P[i]=i;
}
void Result(void){
    int i;count++;
    printf("\n Hoan vi %d:",count);
    for(i=1; i<=n;i++)
        printf("%3d",P[i]);
}
void Next_Permutation(void){
    int j, k, r, s, temp;
    j = n-1;
    while(j>0 && P[j]>P[j+1])
        j--;
    if(j==0)
        Stop=TRUE;
    else {
        k=n;
        while(P[j]>P[k]) k--;
        temp = P[j]; P[j]=P[k]; P[k]=temp;
        r=j+1; s=n;
        while(r<s){
            temp=P[r];P[r]=P[s]; P[s]=temp;
            r++; s--;
        }
    }
}
void Permutation(void){
    Stop = FALSE;
    while (!Stop){

```

```

        Result();
        Next_Permutation();
    }
}

void main(void){
    Init();clrscr(); Permutation();getch();
}

```

Ví dụ 4. Bài toán: Cho n là số nguyên dương. Một cách phân chia số n là biểu diễn n thành tổng các số tự nhiên không lớn hơn n . Chẳng hạn $8 = 2 + 3 + 2$.

Giải. Hai cách chia được gọi là đồng nhất nếu chúng có cùng các số hạng và chỉ khác nhau về thứ tự sắp xếp. Bài toán được đặt ra là, cho số tự nhiên n , hãy duyệt mọi cách phân chia số n .

Chọn cách phân chia số $n = b_1 + b_2 + \dots + b_k$ với $b_1 \geq b_2 \geq \dots \geq b_k$, và duyệt theo trình tự từ điển ngược. Chẳng hạn với $n = 7$, chúng ta có thứ tự từ điển ngược của các cách phân chia như sau:

```

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

```

Như vậy, cách chia đầu tiên chính là n . Cách chia cuối cùng là dãy n số 1. Bây giờ chúng ta chỉ cần xây dựng thuật toán sinh kế tiếp cho mỗi cách phân chia chưa phải là cuối cùng.

Thuật toán sinh cách phân chia kế tiếp:

```

void Next_Division(void){
    int i, j, R, S, D;
    i = k;
    while(i>0 && C[i]==1)

```



```

        i--;
    if(i>0){
        C[i] = C[i]-1;
        D = k - i + 1;
        R = D / C[i];
        S = D % C[i];
        k = i;
        if(R>0){
            for(j=i+1; j<=i+R; j++)
                C[j] = C[i];
            k = k+R;
        }
        if(S>0){
            k=k+1; C[k] = S;
        }
    }
    else Stop=TRUE;
}

```

Văn bản chương trình được thể hiện như sau:

```

#include    <stdio.h>
#include    <conio.h>
#include    <stdlib.h>
#define MAX 100
#define TRUE 1
#define FALSE 0
int n, C[MAX], k, count, Stop;
void Init(void){
    printf("\n Nhập n="); scanf("%d", &n);
    k=1;count=0; C[k]=n;
}
void Result(void){
    int i; count++;
    printf("\n Cach chia %d:", count);
}

```

```

        for(i=1; i<=k; i++)
            printf("%3d", C[i]);
    }
    void Next_Division(void){
        int i, j, R, S, D;
        i = k;
        while(i>0 && C[i]==1)
            i--;
        if(i>0){
            C[i] = C[i]-1;
            D = k - i + 1;
            R = D / C[i];
            S = D % C[i];
            k = i;
            if(R>0){
                for(j=i+1; j<=i+R; j++)
                    C[j] = C[i];
                k = k+R;
            }
            if(S>0){
                k=k+1; C[k] = S;
            }
        }
        else Stop=TRUE;
    }
    void Division(void){
        Stop = FALSE;
        while (!Stop){
            Result();
            Next_Division();
        }
    }
    void main(void){
        clrscr(); Init(); Division(); getch();
    }

```

}

3.4- Thuật toán quay lui (Back track)

Phương pháp sinh kế tiếp có thể giải quyết được các bài toán liệt kê khi ta nhận biết được cấu hình đầu tiên & cấu hình cuối cùng của bài toán. Tuy nhiên, không phải cấu hình sinh kế tiếp nào cũng được sinh một cách đơn giản từ cấu hình hiện tại, ngay kể cả việc phát hiện cấu hình ban đầu cũng không phải dễ tìm vì nhiều khi chúng ta phải chứng minh sự tồn tại của cấu hình. Do vậy, thuật toán sinh kế tiếp chỉ giải quyết được những bài toán liệt kê đơn giản. Để giải quyết những bài toán tổ hợp phức tạp, người ta thường dùng thuật toán quay lui (Back Track) sẽ được trình bày dưới đây.

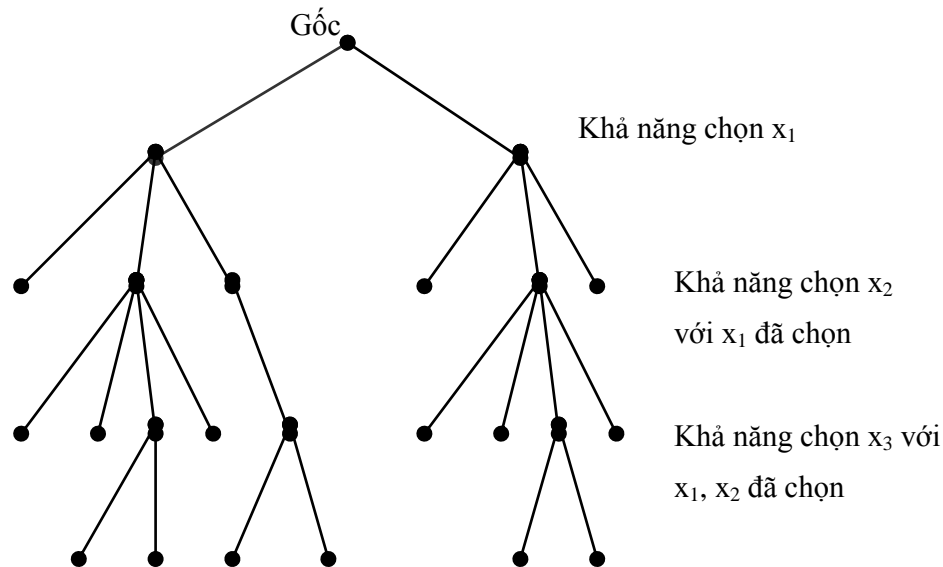
Nội dung chính của thuật toán này là xây dựng dần các thành phần của cấu hình bằng cách thử tất cả các khả năng. Giả sử cần phải tìm một cấu hình của bài toán $x = (x_1, x_2, \dots, x_n)$ mà $i-1$ thành phần x_1, x_2, \dots, x_{i-1} đã được xác định, bây giờ ta xác định thành phần thứ i của cấu hình bằng cách duyệt tất cả các khả năng có thể có và đánh số các khả năng từ $1 \dots n_i$. Với mỗi khả năng j , kiểm tra xem j có chấp nhận được hay không. Khi đó có thể xảy ra hai trường hợp:

- ☐ Nếu chấp nhận j thì xác định x_i theo j , nếu $i=n$ thì ta được một cấu hình cần tìm, ngược lại xác định tiếp thành phần x_{i+1} .
- ☐ Nếu thử tất cả các khả năng mà không có khả năng nào được chấp nhận thì quay lại bước trước đó để xác định lại x_{i-1} .

Điểm quan trọng nhất của thuật toán là phải ghi nhớ lại mỗi bước đã đi qua, những khả năng nào đã được thử để tránh sự trùng lặp. Để nhớ lại những bước duyệt trước đó, chương trình cần phải được tổ chức theo cơ chế ngăn xếp (Last in first out). Vì vậy, thuật toán quay lui rất phù hợp với những phép gọi đệ quy. Thuật toán quay lui xác định thành phần thứ i có thể được mô tả bằng thủ tục Try(i) như sau:

```
void Try( int i ) {  
    int j;  
    for ( j = 1; j < ni; j ++ ) {  
        if ( <Chấp nhận j > ) {  
            <Xác định xi theo j>  
            if ( i == n )  
                <Ghi nhận cấu hình>;  
            else Try(i+1);  
        }  
    }  
}
```

Có thể mô tả quá trình tìm kiếm lời giải theo thuật toán quay lui bằng cây tìm kiếm lời giải sau:

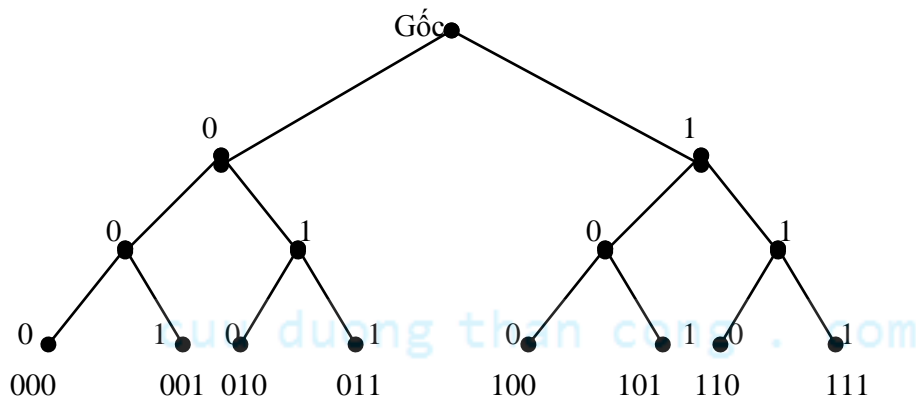


Hình 3.1. Cây liệt kê lời giải theo thuật toán quay lui.

Dưới đây là một số ví dụ điển hình sử dụng thuật toán quay lui.

Ví dụ 1. Liệt kê các xâu nhị phân độ dài n.

Biểu diễn các xâu nhị phân dưới dạng b_1, b_2, \dots, b_n , trong đó $b_i \in \{0, 1\}$. Thủ tục đệ quy Try(i) xác định b_i với các giá trị đề cử cho b_i là 0 và 1. Các giá trị này mặc nhiên được chấp nhận mà không cần phải thỏa mãn điều kiện gì (do đó bài toán không cần đến biến trạng thái). Thủ tục Init khởi tạo giá trị n và biến đếm count. Thủ tục kết quả in ra dãy nhị phân tìm được. Chẳng hạn với $n=3$, cây tìm kiếm lời giải được thể hiện như hình 3.2.



Hình 3.2. Cây tìm kiếm lời giải liệt kê dãy nhị phân độ dài 3

Văn bản chương trình liệt kê các xâu nhị phân có độ dài n sử dụng thuật toán quay lui được thực hiện như sau:

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
```

```

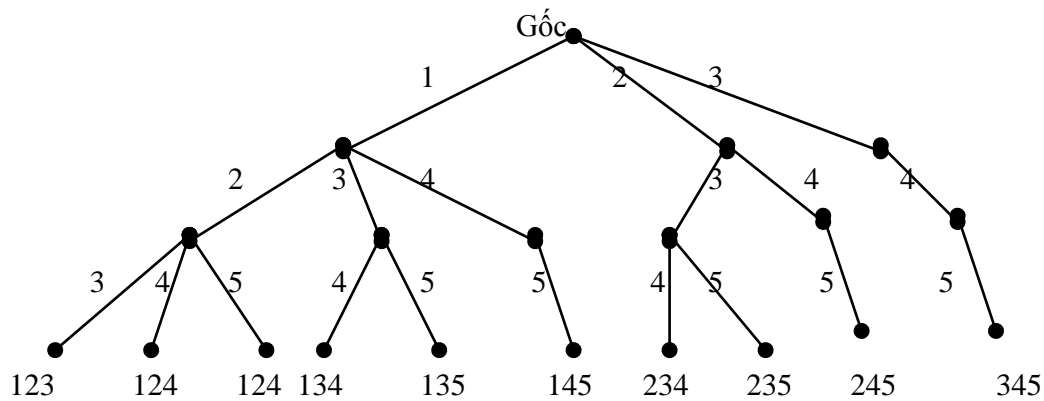
#include <stdlib.h>
void Result(int *B, int n){
    int i;
    printf("\n ");
    for(i=1;i<=n;i++){
        printf("%3d",B[i]);
    }
}
void Init(int *B, int n){
    int i;
    for(i=1;i<=n;i++){
        B[i]=0;
    }
}
void Try(int i, int *B, int n){
    int j;
    for(j=0; j<=1;j++){
        B[i]=j;
        if(i==n) {
            Result(B,n);
        }
        else Try(i+1, B, n);
    }
}
}
void main(void){
    int *B,n;clrscr();
    printf("\n Nhập n=");scanf("%d",&n);
    B=(int *) malloc(n*sizeof(int));
    Init(B,n); Try(1,B,n);free(B);
    getch();
}

```

Ví dụ 2. Liệt kê các tập con k phần tử của tập n phần tử

Giải. Biểu diễn tập con k phần tử dưới dạng c_1, c_2, \dots, c_k , trong đó $1 < c_1 < c_2 < \dots < c_k \leq n$. Từ đó suy ra các giá trị đề cử cho c_i là từ $c_{i-1} + 1$ cho đến $n - k + i$. Cần thêm vào $c_0 = 0$. Các giá trị đề cử này mặc nhiên được chấp nhận mà không cần phải thêm điều kiện gì. Các thủ tục Init, Result được xây dựng như những ví dụ trên.

Cây tìm kiếm lời giải bài toán liệt kê tập con k phần tử của tập n phần tử với $n=5$, $k=3$ được thể hiện như trong hình 3.3.



Hình 3.3. Cây liệt kê tổ hợp chập 3 từ $\{1, 2, 3, 4, 5\}$

Chương trình liệt kê các tập con k phần tử trong tập n phần tử được thể hiện như sau:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int B[MAX], n, k, count=0;
void Init(void){
    printf("\n Nhap n="); scanf("%d", &n);
    printf("\n Nhap k="); scanf("%d", &k);
    B[0]=0;
}
void Result(void){
    int i; count++;
    printf("\n Tap thu %d:", count);
    for(i=1; i<=k; i++){
        printf("%3d", B[i]);
    }
    getch();
}
void Try(int i){
    int j;
    for(j=B[i-1]+1; j<=(n-k+i); j++){
```

```

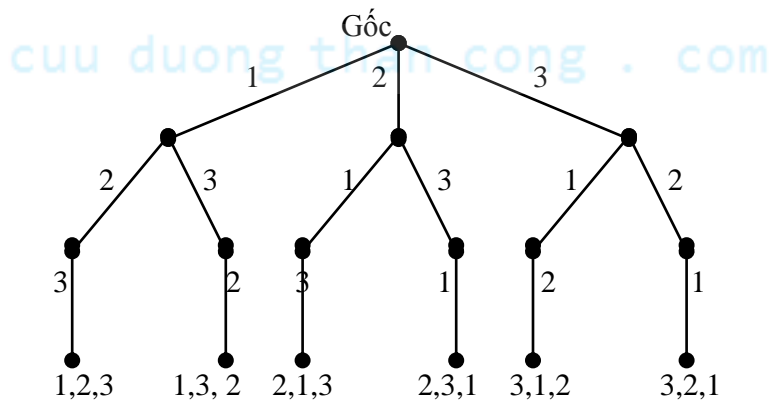
        B[i]=j;
        if(i==k) Result();
        else Try(i+1);
    }
}

void main(void){
    clrscr();Init();Try(1);
}

```

Ví dụ 3. Liệt kê các hoán vị của tập n phần tử.

Giải. Biểu diễn hoán vị dưới dạng p_1, p_2, \dots, 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 lần lượt được đề cử cho p_i , trong đó giá trị j được chấp nhận nếu nó chưa được dùng. Vì vậy, cần phải ghi nhớ với mỗi giá trị j xem nó đã được dùng hay chưa. Điều này được thực hiện nhờ một dãy các biến logic b_j , trong đó $b_j = \text{true}$ nếu j chưa được dùng. Các biến này phải được khởi đầu giá trị true trong thủ tục Init. Sau khi gán j cho p_i , cần ghi nhận false cho b_j và phải gán true khi thực hiện xong Result hay Try($i+1$). Các thủ tục còn lại giống như ví dụ 1, 2. Hình 3.4 mô tả cây tìm kiếm lời giải bài toán liệt kê hoán vị của 1, 2, ..., n với $n = 3$.



Hình 3.4. Cây tìm kiếm lời giải bài toán liệt kê hoán vị của $\{1, 2, 3\}$

Sau đây là chương trình giải quyết bài toán liệt kê các hoán vị của 1, 2, ..., n .

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

#define MAX 100
#define TRUE 1
#define FALSE 0

int P[MAX], B[MAX], n, count=0;

void Init(void){

```

```

    int i;
    printf("\n Nhap n="); scanf("%d", &n);
    for(i=1; i<=n; i++)
        B[i]=TRUE;

}

void Result(void){
    int i; count++;
    printf("\n Hoan vi thu %d:",count);
    for (i=1; i<=n; i++)
        printf("%3d",P[i]);
    getch();
}

void Try(int i){
    int j;
    for(j=1; j<=n;j++){
        if(B[j]) {
            P[i]=j;
            B[j]=FALSE;
            if(i==n) Result();
            else Try(i+1);
            B[j]=TRUE;
        }
    }
}

void main(void){
    Init(); Try(1);
}

```

Ví 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 nhau.

Giải. Bàn cờ có n hàng được đánh số từ 0 đến $n-1$, n cột được đánh số từ 0 đến $n-1$; Bàn cờ có $n^2 - 1$ đường chéo xuôi được đánh số từ 0 đến $2*n - 2$, $2 * n - 1$ đường chéo ngược được đánh số từ $2*n - 2$. Ví dụ: với bàn cờ 8×8 , chúng ta có 8 hàng được đánh số từ 0 đến 7, 8 cột được đánh số từ 0 đến 7, 15 đường chéo xuôi, 15 đường chéo ngược được đánh số từ 0 . . 15.

Vì trên mỗi hàng chỉ xếp được đúng một quân hậu, nên chúng ta chỉ cần quan tâm đến quân hậu được xếp ở cột nào. Từ đó dẫn đến việc xác định bộ n thành phần x_1, x_2, \dots, x_n , trong đó $x_i = j$ được hiểu là quân hậu tại dòng i xếp vào cột thứ j . Giá trị của i được nhận từ 0 đến $n-1$; giá trị của j cũng được nhận từ 0 đến $n-1$, nhưng thoả mãn điều kiện ô (i,j) chưa bị quân hậu khác chiếu đến theo cột, đường chéo xuôi, đường chéo ngược.

Việc kiểm soát theo hàng ngang là không cần thiết vì trên mỗi hàng chỉ xếp đúng một quân hậu. Việc kiểm soát theo cột được ghi nhận nhờ dãy biến logic a_j với qui ước $a_j=1$ nếu cột j còn trống, $a_j=0$ nếu cột j không còn trống. Để ghi nhận đường chéo xuôi và đường chéo ngược có chiếu tới ô (i,j) hay không, ta sử dụng phương trình $i + j = \text{const}$ và $i - j = \text{const}$, đường chéo thứ nhất được ghi nhận bởi dãy biến b_j , đường chéo thứ 2 được ghi nhận bởi dãy biến c_j với qui ước nếu đường chéo nào còn trống thì giá trị tương ứng của nó là 1 ngược lại là 0. Như vậy, cột j được chấp nhận khi cả 3 biến a_j, b_{i+j}, c_{i-j} đều có giá trị 1. Các biến này phải được khởi đầu giá trị 1 trước đó, gán lại giá trị 0 khi xếp xong quân hậu thứ i và trả lại giá trị 1 khi đưa ra kết quả.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#define N      8
#define D      (2*N-1)
#define SG      (N-1)
#define TRUE  1
#define FALSE 0
void hoanghau(int);
void inloigiai(int    loigiai[]);FILE *fp;
int    A[N], B[D], C[D], loigiai[N];
int soloigiai =0;
void hoanghau(int i){
    int j;
    for (j=0; j<N;j++){
        if (A[j] && B[i-j+SG] && C[i+j] ) {
            loigiai[i]=j;
            A[j]=FALSE;
            B[i-j+SG]=FALSE;
            C[i+j]=FALSE;
            if (i==N-1){
                soloigiai++;
                inloigiai(loigiai);
            }
        }
    }
}
```

```

        delay(500);
    }
    else
        hoanghai(i+1);
    A[j]=TRUE;
    B[i-j+SG]=TRUE;
    C[i+j]=TRUE;
}
}
}

void inloigiai(int *loigiai){
    int j;
    printf("\n Lờ i giải %3d:",soloigiai);
    fprintf(fp,"\n Lờ i giải %3d:",soloigiai);
    for (j=0;j<N;j++){
        printf("%3d",loigiai[j]);
        fprintf(fp,"%3d",loigiai[j]);
    }
}

void main(void){
    int i;clrscr();fp=fopen("loigiai.txt","w");
    for (i=0;i<N;i++)
        A[i]=TRUE;
    for(i=0;i<D; i++){
        B[i]=TRUE;
        C[i]=TRUE;
    }
    hoanghai(0);fclose(fp);
}

```

Dưới đây là số cách xếp hậu ứng với 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 đương với cách xếp trên hình 5.

3.5. Những nội dung cần ghi nhớ

- ✓ Thế nào là bài toán liệt kê, bài toán tồn tại?
- ✓ Những điều kiện bắt buộc của một thuật toán liệt kê.
- ✓ Hiểu và nắm vững lớp các bài toán có thể giải được bằng phương pháp sinh.
- ✓ Hiểu và nắm vững những yếu tố cần thiết để thực hiện giải thuật quay lui.
- ✓ Phương pháp chứng minh phản chứng giải quyết bài toán tồn tại
- ✓ Sử dụng nguyên lý Dirichlet giải quyết bài toán tồn tại.

cuu duong than cong . com

cuu duong than cong . com

BÀI TẬP

Bài 1. Liệt kê tất cả các xâu nhị phân độ dài 5 không chứa hai số 0 liên tiếp.

Bài 2. Liệt kê tất cả các phần tử của tập

$$D = \{x = (x_1, x_2, \dots, x_n) : \sum_{j=1}^n a_j x_j = b, \quad x_j \in \{0,1\}, j = 1, 2, \dots, n\}$$

Trong đó a_1, a_2, \dots, a_n, b là các số nguyên dương.

Bài 3. Liệt kê tất cả các phần tử của tập

$$D = \{x = (x_1, x_2, \dots, x_n) : \sum_{j=1}^n a_j x_j = b, \quad x_j \in \mathbb{Z}_+, j = 1, 2, \dots, n\}$$

Trong đó a_1, a_2, \dots, a_n, b là các số nguyên dương.

Bài 4. 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 thỏa mãn các tính chất: Tổng các phần tử trên mỗi dòng, mỗi cột và mỗi một trong hai đường chéo có cùng một giá trị. Hãy liệt kê tất cả các ma phương bậc 3, 4 không sai khác nhau bởi các phép biến hình đơn giản (quay, đối xứng).

Ví dụ dưới đây là một ma phương bậc 3 thỏa mãn tính chất tổng hàng, cột, đường chéo đều là 15.

$$\begin{vmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{vmatrix}$$

Bài 5. 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 $n=5, k=35$ ta có cách điền sau:

11				
10	3			
9		2		
1			7	
4	5	6	8	12

Phát triển thuật toán dựa trên thuật toán quay lui để chỉ ra với giá trị của n, k cho trước bài toán có lời giải hay không. Nếu có câu trả lời chỉ cần đưa ra một lời giải.

Bài 6. Tìm tập con dài nhất có thứ tự tăng dần, giảm dần. Cho dãy số a_1, a_2, \dots, a_n . Hãy tìm dãy con dài nhất được sắp xếp theo thứ tự tăng hoặc giảm dần. Dữ liệu vào cho bởi file tapcon.in, dòng đầu tiên ghi lại số tự nhiên n ($n \leq 100$), dòng kế tiếp ghi lại n số, mỗi số được phân biệt với nhau bởi một hoặc vài ký tự rỗng. Kết quả ghi lại trong file tapcon.out. Ví dụ sau sẽ minh họa cho file tapcon.in và tapcon.out.

tapcon.in	tapcon.out
5	5
7 1 3 8 9 6 12	1 3 8 9 12

Bài 7. Duyệt các tập con thỏa mãn điều kiện. Cho dãy số a_1, a_2, \dots, a_n và số M . Hãy tìm tất cả các dãy con trong dãy số a_1, a_2, \dots, a_n sao cho tổng các phần tử trong dãy con đúng bằng M . Dữ liệu vào cho bởi file tapcon.in, dòng đầu tiên ghi lại hai số tự nhiên N và số M ($N \leq 100$), dòng kế tiếp ghi lại N số mỗi số được phân biệt với nhau bởi một và dấu trống. Kết quả ghi lại trong file tapcon.out. Ví dụ sau sẽ minh họa cho file tapcon.in và tapcon.out

tapcon.in	
7 50	
5 10 15 20 25 30 35	
tapcon.out	
20 30	
15 35	
10 15 25	
5 20 25	
5 15 30	
5 10 35	
5 10 15 20	

Bài 8. Cho lưới hình chữ nhật gồm $(n \times m)$ hình vuông đơn vị. Hãy liệt kê tất cả các đường đi từ điểm có tọa độ $(0, 0)$ đến điểm có tọa độ (n, m) . Biết rằng, điểm $(0, 0)$ được coi là đỉnh dưới của hình vuông dưới nhất góc bên trái, mỗi bước đi chỉ được phép thực hiện hoặc lên trên hoặc xuống dưới theo cạnh của hình vuông đơn vị. Dữ liệu vào cho bởi file bai14.inp, kết quả ghi lại trong file bai14.out. Ví dụ sau sẽ minh họa cho file bai14.in và bai14.out.

bai14.in	
2 2	
bai14.out	
0 0 1 1	
0 1 0 1	
0 1 1 0	
1 0 0 1	
1 0 1 0	
1 1 0 0	

Bài 9. Tìm bộ giá trị rời rạc để hàm mục tiêu $\sin(x_1 + x_2 + \dots + x_k)$ đạt giá trị lớn nhất. Dữ liệu vào cho bởi file bai4.inp, kết quả ghi lại trong file bai4.out.

Bài 10. Duyệt mọi phép toán trong tính toán giá trị biểu thức. Viết chương trình nhập từ bàn phím hai số nguyên M, N . Hãy tìm cách thay các dấu ? trong biểu thức sau bởi các phép toán $+, -, *, \%, /$ (chia nguyên) sao cho giá trị của biểu thức nhận được bằng đúng N :
 $((((M?M)?M)?M)?M)?M$

Nếu không được hãy đưa ra thông báo là không thể được.

CHƯƠNG 4. BÀI TOÁN TỐI ƯU

Nội dung chính của chương này là giới thiệu các phương pháp giải quyết bài toán tối ưu đồng thời giải quyết một số bài toán có vai trò quan trọng của lý thuyết tổ hợp. Những nội dung được đề cập bao gồm:

- ✓ Giới thiệu bài toán và phát biểu bài toán tối ưu cho các mô hình thực tế.
- ✓ Phân tích phương pháp liệt kê giải quyết bài toán tối ưu.
- ✓ Phương pháp nhánh cận giải quyết bài toán tối ưu.
- ✓ Phương pháp rút gọn giải quyết bài toán tối ưu.

Bạn đọc có thể tìm thấy phương pháp giải chi tiết cho nhiều bài toán tối ưu quan trọng trong các tài liệu [1], [2].

4.1- Giới thiệu bài toán

Trong nhiều bài toán thực tế, các cấu hình tổ hợp còn được gán một giá trị bằng số đánh giá giá trị sử dụng của cấu hình đối với một mục đích sử dụng cụ thể nào đó. Khi đó xuất hiện bài toán: Hãy lựa chọn trong số tất cả các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất. Các bài toán như vậy được gọi là bài toán tối ưu tổ hợp. Chúng ta có thể phát biểu bài toán tối ưu tổ hợp dưới dạng tổng quát như sau:

Tìm cực tiểu (hay cực đại) của phiếm hàm $f(x) = \min(\max)$ với điều kiện $x \in D$, trong đó D là tập hữu hạn các 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 thỏa mãn một số tính chất nào đó 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 bài toán tối ưu tổ hợp kinh điển. Các bài toán này là những mô hình có nhiều ứng dụng thực tế và 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 thăm quan n thành phố T_1, T_2, \dots, 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 đi qua 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 ($i, j = 1, 2, \dots, n$), hãy tìm hành trình với tổng chi phí là nhỏ nhất (một hành trình là một cách đi thỏa mãn điều kiện).

Rõ ràng, ta có thể thiết lập được một tương ứng 1-1 giữa hành trình $T_{\pi(1)} \rightarrow T_{\pi(2)} \rightarrow \dots \rightarrow T_{\pi(n)} \rightarrow T_{\pi(1)}$ với một hoán vị $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ của n số tự nhiên $1, 2, \dots, n$. Đặt

$$f(\pi) = C_{\pi(1), \pi(2)} + C_{\pi(2), \pi(3)} + \dots + C_{\pi(n-1), \pi(n)} + C_{\pi(n), \pi(1)},$$

kí hiệu Π là tập tất cả các hoán vị $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ của n số tự nhiên $1, 2, \dots, n$. 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 làm điểm 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 a_j và giá trị sử dụng c_j ($j = 1, 2, \dots, n$). Hỏi nhà thám hiểm cần đem theo những đồ vật nào để cho tổng giá trị sử dụng là lớn nhất?

Một phương án của nhà thám hiểm có thể biểu diễn như một vector nhị phân độ dài n : $x = (x_1, x_2, \dots, x_n)$, trong đó $x_i = 1$ có nghĩa là đồ vật thứ i được đem theo, $x_i = 0$ có nghĩa trái lại. Với phương án đem theo x , giá trị sử dụng các đồ vật đem theo là

$$f(x) = \sum_{i=1}^n c_i x_i, \text{ tổng trọng lượng đồ vật đem theo là } g(x) = \sum_{i=1}^n a_i x_i, \text{ như vậy bài toán}$$

cái túi được phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

Trong số các vector nhị phân độ dài n thoả mãn điều kiện $g(x) \leq b$, hãy tìm vector x^* để hàm mục tiêu $f(x)$ đạt giá trị nhỏ nhất. Nói cách khác:

$$\min \{ f(x) : g(x) \leq b \}$$

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 ($i = 1, 2, \dots, m$). Ông chủ chỉ có quyền hoặc từ chối yêu cầu của khách hàng i , hoặc 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 thế nào để cho tổng số ngày sử dụng máy là lớn nhất.

Ký hiệu, $I = \{ 1, 2, \dots, m \}$ 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

$$D = \{ J \subset S : N_k \cap N_p = \emptyset, \forall k \neq p \in J \}. \text{ Với mỗi phương án } J \in D \text{ } f(J) = \sum_{j \in J} |N_j| \text{ sẽ là tổng số}$$

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 phân công: Có n công việc và n thợ. Biết c_{ij} là chi phí cần trả để thợ i hoàn thành công việc thứ j ($i, j = 1, 2, \dots, n$). 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 n nhân công sao cho tổng chi phí thuê thợ là nhỏ nhất.

Rõ ràng, mỗi phương án bố trí thợ thực hiện các công việc tương ứng với một hoán vị $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ của n số tự nhiên $\{ 1, 2, \dots, n \}$. Chi phí theo phương án trên là $f(\pi) = C_{\pi(1),1} + C_{\pi(2),2} + \dots + C_{\pi(n),n}$.

Công việc	Thợ thực hiện
1	$\pi(1)$
2	$\pi(2)$
...	...
n	$\pi(n)$

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, \dots, D_n cần phải lần lượt được gia công trên m máy M_1, M_2, \dots, 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. Biết rằng, các chi tiết được gia công một cách liên tục, nghĩa là quá trình gia công của mỗi một chi tiết 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.

Rõ ràng, mỗi một lịch gia công các chi tiết trên các máy sẽ tương ứng với một hoán vị $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ của n số tự nhiên $1, 2, \dots, n$. Thời gian hoàn thành theo các lịch trên được xác định bởi hàm số

$$f(\pi) = \sum_{j=1}^{n-1} C_{\pi(j), \pi(j+1)} + \sum_{k=1}^m t_{k, \pi(n)}, \text{ trong đó } c_{ij} = S_j - S_i, S_j \text{ là thời điểm bắt đầu thực hiện}$$

việc gia công chi tiết j ($i, j = 1, 2, \dots, n$). Ý nghĩa của hệ số c_{ij} có thể được 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:

$$c_{ij} = \max_{1 \leq k \leq m} \left[\sum_{l=1}^k t_{lj} - \sum_{l=1}^{k-1} t_{li} \right], i, j = 1, 2, \dots, n. \text{ Vì vậy bài toán đặt ra dẫn về bài toán tối ưu tổ}$$

hợp sau:

$$\min \{ f(\pi) : \pi \in \Pi \}.$$

Trong thực tế, lịch gia công còn phải thỏa mãn thêm nhiều điều kiện khác nữa. Vì những ứng dụng quan trọng của những bài toán loại 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 qui hoạch lịch.

4.2- 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 cho phương án đó, sau đó so sánh giá trị của hàm mục tiêu tại tất cả các phương án đã được liệt kê để tìm ra phương án tối ưu. Phương pháp xây dựng theo nguyên tắc như vậy được gọi là phương pháp duyệt toàn bộ. Hạn chế của phương pháp duyệt toàn bộ là sự bùng nổ của các cấu hình tổ hợp. Chẳng hạn để duyệt được $15! = 1\,307\,674\,368\,000$ cấu hình, trên máy có tốc độ 1 tỷ phép tính giây, nếu mỗi hoán vị cần liệt kê mất khoảng 100 phép tính, thì ta cần khoảng thời gian là 130767 giây (lớn hơn 36 tiếng đồng hồ). Vì vậy, cần phải có biện pháp hạn chế việc kiểm tra hoặc tìm kiếm trên các cấu hình tổ hợp thì mới có hy vọng giải được các bài toán tối ưu tổ hợp thực tế. Tất nhiên, để đưa ra được một thuật toán cần phải nghiên cứu kỹ tính chất của mỗi bài toán tổ hợp cụ thể. Chính nhờ những nghiên cứu đó, trong một số trường hợp cụ thể ta có thể xây dựng được thuật toán hiệu quả để giải quyết bài toán đặt ra. Nhưng chúng ta cũng cần phải chú ý rằng, trong nhiều trường hợp (bài toán người du lịch, bài toán cái túi, bài toán cho thuê máy) chúng ta vẫn chưa tìm ra được một phương pháp hữu hiệu nào ngoài phương pháp duyệt toàn bộ đã được đề cập ở trên.

Để hạn chế việc duyệt, trong quá trình liệt kê cần tận dụng triệt để những thông tin đã tìm để loại bỏ những phương án chắc chắn không phải là tối ưu. Dưới đây là một bài toán tối ưu tổ hợp rất thường gặp trong kỹ thuật.

Ví dụ. Duyệt mọi bộ giá trị trong tập các giá trị rời rạc.

Bài toán. Tìm

$\max \{f(x_1, x_2, \dots, x_n) : x_i \in D_i; i = 1, 2, \dots, n\}$ hoặc

$\min \{f(x_1, x_2, \dots, x_n) : x_i \in D_i; i = 1, 2, \dots, n\}.$

Trong đó, D_i là một tập hữu hạn các giá trị rời rạc thỏa mãn một điều kiện ràng buộc nào đó.

Giải. Giả sử số các phần tử của tập giá trị rời rạc D_i là r_i ($i=1, 2, \dots, n$). Gọi $R = r_1 + r_2 + \dots + r_n$ là số các phần tử thuộc tất cả các tập D_i ($i=1, 2, \dots, n$). Khi đó, ta có tất cả $C(R, n)$ bộ có thứ tự các giá trị gồm n phần tử trong R phần tử, đây chính là số các phương án ta cần duyệt. Trong số $C(R, n)$ các bộ n phần tử, ta cần lọc ra các bộ thỏa mãn điều kiện $x_i \in D_i$ ($i=1, 2, \dots, n$) để tính giá trị của hàm mục tiêu $f(x_1, x_2, \dots, x_n)$. Như vậy, bài toán được đưa về bài toán duyệt các bộ gồm n phần tử (x_1, x_2, \dots, x_n) từ tập hợp gồm $R = r_1 + r_2 + \dots + r_n$ phần tử thỏa mãn điều kiện $x_i \in D_i$.

Ví dụ: với tập $D_1 = (1, 2, 3)$,

$D_2 = (3, 4)$,

$D_3 = (5, 6, 7)$.

Khi đó chúng ta cần duyệt bộ các giá trị rời rạc sau:

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

Với cách phân tích như trên, ta có thể sử dụng thuật toán quay lui để duyệt kết hợp với việc kiểm tra thành phần $x_i \in D_i$. Dưới đây là toàn văn chương trình duyệt các bộ giá trị trong tập các giá trị rời rạc.

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <conio.h>
#define MAX 2000000
#define TRUE 1
#define FALSE 0
int n, k, H[100]; float *B; int *C, count = 0, m;
FILE *fp;
void Init(void){
    int i, j; float x; C[0] = 0; H[0] = 0;
```

```

fp=fopen("roirac.in","r");
fscanf(fp,"%d",&n);
printf("\n So tap con roi rac n=%d",n);
for(i=1; i<=n; i++){
    fscanf(fp,"%d",&H[i]);
    printf("\n Hang %d co so phan tu la %d",i, H[i]);
}
H[0]=0;
for (i=1; i<=n; i++){
    printf("\n");
    for(j=1; j<=H[i]; j++){
        fscanf(fp,"%f",&x);
        B[++k]=x;
    }
}
printf("\n B=");
for(i=1; i<=k; i++){
    printf("%8.2f", B[i]);
}
fclose(fp);
}

int In_Set(int i){
    int canduoi=0, cantren=0,j;
    for(j=1; j<=i; j++)
        cantren = cantren + H[j];
    canduoi=cantren-H[j-1];
    if (C[i]> canduoi && C[i]<=cantren)
        return(TRUE);
    return(FALSE);
}

void Result(void){
    int i;
    count++; printf("\n Tap con thu count=%d:",count);
    for(i=1; i<=n ; i++){

```

```

        printf("%8.2f", B[C[i]]);
    }
}

void Try(int i){
    int j;
    for(j = C[i-1]+1; j<=(k-n+i); j++){
        C[i]=j;
        if(In_Set(i)){
            if (i==n ) Result();
            else Try(i+1);
        }
    }
}

void main(void){
    clrscr();
    B = (float *) malloc(MAX *sizeof(float));
    C = (int *) malloc(MAX *sizeof(int));
    Init();Try(1);free(B); free(C);getch();
}

```

4.3. Thuật toán nhánh cận

Giả sử chúng ta cần giải quyết bài toán tối ưu tổ hợp với mô hình tổng quát như sau:

$\min \{f(x) : x \in D\}$. Trong đó D là tập hữu hạn phần tử. Ta giả thiết D được mô tả như sau:

$D = \{x = (x_1, x_2, \dots, x_n) \in A_1 \times A_2 \times \dots \times A_n; x \text{ thỏa mãn tính chất } P\}$, với $A_1 \times A_2 \times \dots \times A_n$ là các tập hữu hạn, P là tính chất cho trên tích đề xác $A_1 \times A_2 \times \dots \times A_n$.

Như vậy, các bài toán chúng ta vừa trình bày ở trên đều có thể được mô tả dưới dạng trên.

Với giả thiết về tập D như 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. Ta gọi, một bộ phận gồm k thành phần (a_1, a_2, \dots, a_k) xuất hiện trong quá trình thực hiện thuật toán sẽ được gọi là phương án bộ phận cấp k .

Thuật toán nhánh cận có thể được áp dụng 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 thỏa mãn bất đẳng thức sau:

$$g(a_1, a_2, \dots, a_k) \leq \min \{f(x) : x \in D, x_i = a_i, i = 1, 2, \dots, k\} \quad (*)$$

với mọi lời giải bộ phận (a_1, a_2, \dots, a_k) , và với mọi $k = 1, 2, \dots$

Bất đẳng thức (*) có nghĩa là giá trị của hàm tại phương án bộ phận (a_1, a_2, \dots, a_k) không vượt quá giá trị nhỏ nhất của hàm mục tiêu bài toán trên tập con các phương án.

$$D(a_1, a_2, \dots, a_k) \{ x \in D: x_i = a_i, i = 1, 2, \dots, k \},$$

nói cách khác, $g(a_1, a_2, \dots, a_k)$ là cận dưới của tập $D(a_1, a_2, \dots, a_k)$. Do có thể đồng nhất tập $D(a_1, a_2, \dots, a_k)$ với phương án bộ phận (a_1, a_2, \dots, a_k) , nên ta cũng gọi giá trị $g(a_1, a_2, \dots, a_k)$ là cận dưới của phương án bộ phận (a_1, a_2, \dots, a_k) .

Giả sử ta đã có được hàm g . Ta xét cách sử dụng hàm này để hạn chế 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 giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã duyệt, ký hiệu $\bar{f} = f(\bar{x})$. Ta gọi \bar{x} là phương án tốt nhất hiện có, còn \bar{f} là kỷ lục. Giả sử ta có được \bar{f} , khi đó nếu

$g(a_1, a_2, \dots, a_k) > \bar{f}$ thì từ bất đẳng thức (*) ta suy ra

$\bar{f} < g(a_1, a_2, \dots, a_k) \leq \min \{ f(x): x \in D, x_i = a_i, i = 1, 2, \dots, k \}$, vì thế tập con các phương án của bài toán $D(a_1, a_2, \dots, a_k)$ chắc chắn không chứa phương án tối ưu. Trong trường hợp này ta không cần phải phát triển phương án bộ phận (a_1, a_2, \dots, a_k) , nói cách khác là ta có thể loại bỏ các phương án trong tập $D(a_1, a_2, \dots, a_k)$ khỏi quá trình tìm kiếm.

Thuật toán quay lui liệt kê các phương án cần sửa đổi lại như sau:

```
void Try(int k){
    /*Phát triển phương án bộ phận  $(a_1, a_2, \dots, a_{k-1}$ 
    theo thuật toán quay lui có kiểm tra cận dưới
    Trước khi tiếp tục phát triển phương án*/
    for (  $a_k \in A_k$  ) {
        if ( chấp nhận  $a_k$  ){
             $x_k = a_k$ ;
            if (  $k == n$  )
                < cập nhật kỷ lục >;
            else if (  $g(a_1, a_2, \dots, a_k) \leq \bar{f}$  )
                Try (  $k+1$  );
        }
    }
}
```

Khi đó, thuật toán nhánh cận được thực hiện nhờ thủ tục sau:

```
void Nhanh_Can(void) {
     $\bar{f} = +\infty$ ;
```

```

/* Nếu biết một phương án  $\bar{x}$  nào đó thì có thể đặt  $\bar{f} = f(\bar{x})$ . */
Try(1);
if (  $\bar{f} \leq +\infty$  )
    <  $\bar{f}$  là giá trị tối ưu ,  $\bar{x}$  là phương án tối ưu >;
else
    < bài toán không có phương án >;
}

```

Chú ý rằng nếu trong thủ tục Try ta thay thế câu lệnh

```

if (  $k == n$  )
    < cập nhật kỷ lục >;
else if (  $g(a_1, a_2, \dots, a_k) \leq \bar{f}$  )
    Try( $k+1$ );

```

bởi

```

if (  $k == n$  )
    < 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 lại thu được thuật toán duyệt toàn bộ. Việc xây dựng hàm g phụ thuộc vào từng bài toán tối ưu tổ hợp cụ thể. Nhưng chúng ta cố gắng xây dựng sao cho đạt được những điều kiện dưới đây:

- ☐ Việc tính giá trị của g phải đơn giản hơn việc giải bài toán tổ hợp trong vế phải của (*).
- ☐ Giá trị của $g(a_1, a_2, \dots, a_k)$ phải sát với giá trị vế phải của (*).

Rất tiếc, hai yêu cầu này trong thực tế thường đối lập nhau.

Ví dụ 1. Bài toán cái túi. Chúng ta sẽ xét bài toán cái túi tổng quát hơn mô hình đã được trình bày trong mục 4.1. Thay vì có n đồ vật, 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 c_j ($j = 1, 2, \dots, n$). 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 đựng trong túi là lớn nhất.

Mô hình toán học của bài toán có dạng sau tìm

$$f^* = \max \left\{ f(x) = \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x_j \in Z_+, j = 1, 2, \dots, n \right\}, (1).$$

Trong đó Z^+ là tập các số nguyên không âm.

Ký hiệu D là tập các phương án của bài toán (1):

$$D = \left\{ x = (x_1, x_2, \dots, x_n) : \sum_{j=1}^n a_j x_j \leq b, x_j \in Z_+, j = 1, 2, \dots, n \right\}.$$

Không giảm tính tổng quát ta giả thiết rằng, các đồ vật được đánh số sao cho bất đẳng thức sau được thoả mãn

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \dots \geq \frac{c_n}{a_n} \quad (2)$$

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

$$g^* = \max \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x_j \geq 0, j = 1, 2, \dots, n \right\}. \quad (3)$$

Mệnh đề. Phương án tối ưu của bài toán (3) là vector $\bar{x} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ với các thành phần được xác định bởi công thức:

$$\bar{x}_1 = \frac{b}{a_1}, \bar{x}_2 = \bar{x}_3 = \dots = \bar{x}_n = 0 \text{ và giá trị tối ưu là } g^* = \frac{c_1 b}{a_1}.$$

Chứng minh. Thực vậy, xét $x = (x_1, x_2, \dots, x_n)$ là một phương án tùy ý của bài toán (3). Khi đó từ bất đẳng thức (3) và do $x_j \geq 0$, ta suy ra

$$c_j x_j \geq (c_1 / a_1) a_j x_j, j = 1, 2, \dots, n.$$

suy ra:

$$\sum_{j=1}^n c_j x_j \leq \sum_{j=1}^n \left(\frac{c_1}{a_1} \right) a_j x_j = \left(\frac{c_1}{a_1} \right) \sum_{j=1}^n a_j x_j \leq \frac{c_1}{a_1} b = g^*. \text{ Mệnh đề được chứng minh.}$$

Bây giờ ta giả sử có phương án bộ phận cấp k : (u_1, u_2, \dots, u_k) . Khi đó giá trị sử dụng của các đồ vật đang có trong túi là

$$\partial_k = c_1 u_1 + c_2 u_2 + \dots + c_k u_k, \text{ và trọng lượng còn lại của túi là}$$

$$b_k = b - c_1 u_1 + c_2 u_2 + \dots + c_k u_k,$$

ta có

$$\begin{aligned} & \max \{ f(x) : x \in D, x_j = u_j, j = 1, 2, \dots, n \} \\ &= \max \left\{ \partial_k + \sum_{j=k+1}^n c_j x_j : \sum_{j=k+1}^n a_j x_j \leq b_k, x_j \in Z_+, j = k+1, k+2, \dots, n \right\} \\ &\leq \partial_k + \max \left\{ \sum_{j=k+1}^n c_j x_j : \sum_{j=k+1}^n a_j x_j \leq b_k, x_j \geq 0, j = k+1, k+2, \dots, n \right\} \\ &= \partial_k + \frac{c_{k+1} b_k}{a_{k+1}} \end{aligned}$$

$$(\text{Theo mệnh đề giá trị số hạng thứ hai là } \frac{c_{k+1} b_k}{a_{k+1}})$$

Vậy ta có thể tính cận trên cho phương án bộ phận (u_1, u_2, \dots, u_k) theo công thức

$$g(u_1, u_2, \dots, u_k) = \partial_k + \frac{c_{k+1}b_k}{a_{k+1}} \dots$$

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 $0, 1, \dots, \lfloor b_k/a_{k+1} \rfloor$. 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.

Ví dụ. Giải bài toán cái túi sau theo thuật toán nhánh cận trình bày trên.

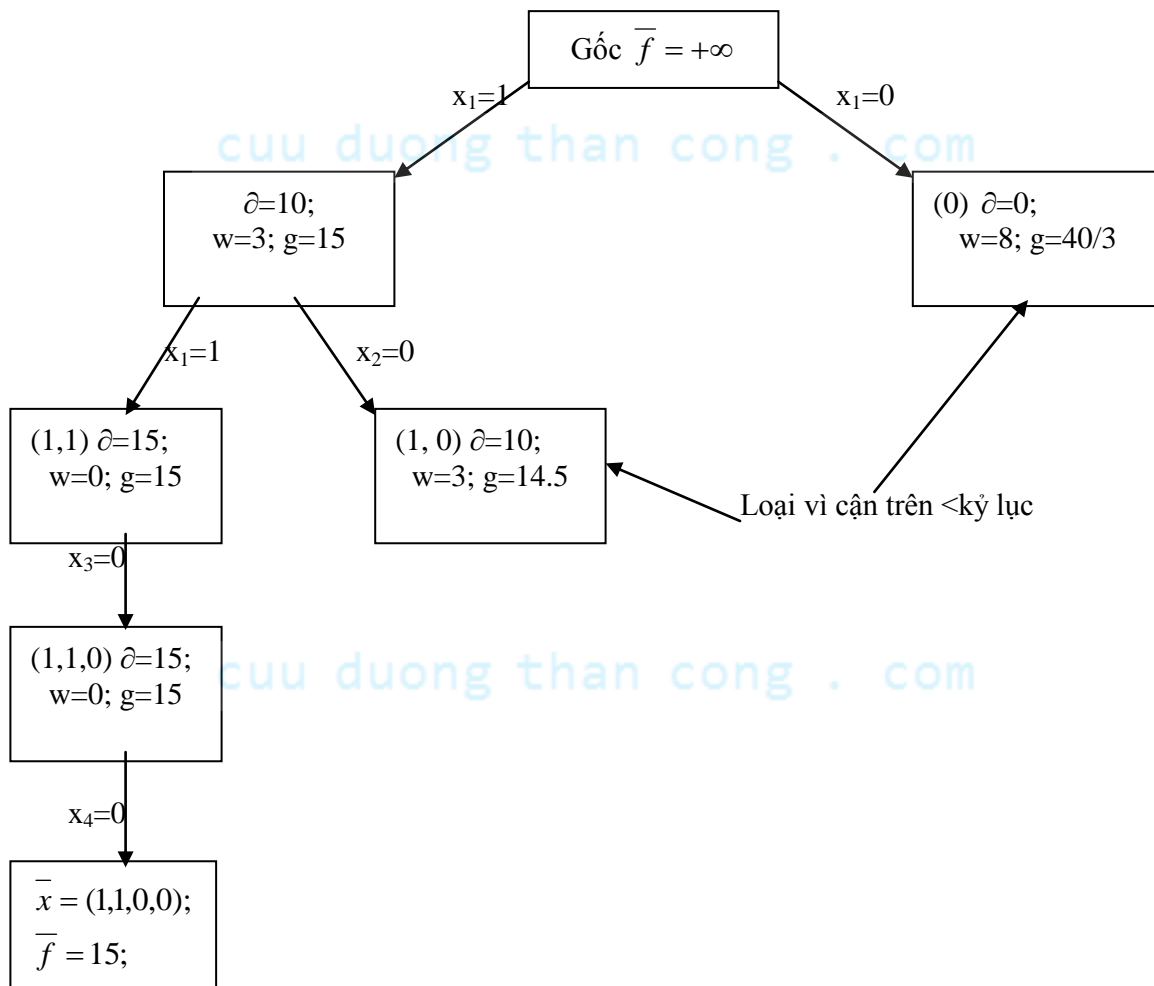
$$f(x) = 10x_1 + 5x_2 + 3x_3 + 6x_4 \rightarrow \max$$

$$5x_1 + 3x_2 + 2x_3 + 4x_4 \leq 8$$

$$x_j \in Z_+, j = 1, 2, 3, 4.$$

Giải. Quá trình giải bài toán được mô tả trong cây tìm kiếm trong hình 4.1. 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 ∂ là giá trị của các đồ vật chất trong túi, w là trọng lượng còn lại của túi và g là cận trên.

Kết thúc thuật toán, ta thu được phương án tối ưu là $x^* = (1, 1, 0, 1)$, giá trị tối ưu $f^* = 15$.



Hình 4.1. Giải bài toán cái túi theo thuật toán nhánh cận.

Chương trình giải bài toán cái túi theo thuật toán nhánh cận được thể hiện như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define TRUE 1
#define FALSE 0
#define MAX 100
int x[MAX], xopt[MAX];
float fopt, cost, weight;
void Init(float *C, float *A, int *n, float *w){
    int i; FILE *fp;
    fopt=0; weight=0;
    fp=fopen("caitui.in", "r");
    if(fp==NULL){
        printf("\n Không có file input");
        delay(2000); return;
    }
    fscanf(fp, "%d %f", n, w);
    for(i=1; i<=*n; i++) xopt[i]=0;
    printf("\n Số lượng đồ vật %d:", *n);
    printf("\n Giới hạn túi %8.2f:", *w);
    printf("\n Vecto giá trị:");
    for(i=1; i<=*n; i++) {
        fscanf(fp, "%f", &C[i]);
        printf("%8.2f", C[i]);
    }
    printf("\n Vector trong lượng:");
    for(i=1; i<=*n; i++){
        fscanf(fp, "%f", &A[i]);
        printf("%8.2f", A[i]);
    }
    fclose(fp);
}
```



```

}
void swap(int n){
    int i;
    for(i=1; i<=n; i++)
        xopt[i]=x[i];
}
void Update_Kyluc(int n){
    if(cost>fopt){
        swap(n);
        fopt=cost;
    }
}
void Try(float *A, float *C, int n, float w, int i){
    int j, t=(w-weight)/A[i];
    for(j=t; j>=0; j--){
        x[i]=j;
        cost = cost + C[i]*x[i];
        weight = weight + x[i]*A[i];
        if(i==n) Update_Kyluc(n);
        else if(cost + C[i+1]*(w-weight)/A[i+1]> fopt){
            Try(A, C, n, w, i+1);
        }
        weight = weight-A[i]*x[i];
        cost = cost-C[i]*x[i];
    }
}
void Result(int n){
    int i;
    printf("\n Gia tri do vat %8.2f:", fopt);
    printf("\n Phuong an toi uu:");
    for(i=1; i<=n; i++)
        printf("%3d", xopt[i]);
}

```

```

void main(void){

    int    n;

    float  A[MAX], C[MAX], w;

    clrscr();Init(C, A, &n, &w);

    Try(C, A, n, w,1);Result(n);

    getch();

}

```

Ví dụ 2. Bài toán Người du lịch. Một người du lịch muốn đi thăm quan n thành phố T_1, T_2, \dots, 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 đi qua 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 ($i = 1, 2, \dots, n$), hãy tìm hành trình với tổng chi phí là nhỏ nhất (một hành trình là một cách đi thỏa mãn điều kiện).

Giải. Cố định thành phố xuất phát là T_1 . Bài toán Người du lịch được đưa về bài toán: Tìm cực tiểu của phiếm hàm:

$$f(x_1, x_2, \dots, x_n) = c[1, x_2] + c[x_2, x_3] + \dots + c[x_{n-1}, x_n] + c[x_n, x_1] \rightarrow \min$$

với điều kiện

$$c_{\min} = \min\{c[i, j], i, j = 1, 2, \dots, n; i \neq j\} \text{ là chi phí đi lại giữa các thành phố.}$$

Giả sử ta đang có phương án bộ phận (u_1, u_2, \dots, u_k) . Phương án tương ứng với hành trình bộ phận qua k thành phố:

$$T_1 \rightarrow T(u_2) \rightarrow \dots \rightarrow T(u_{k-1}) \rightarrow T(u_k)$$

Vì vậy, chi phí phải trả theo hành trình bộ phận này sẽ là tổng các chi phí theo từng node của hành trình bộ phận.

$$\partial = c[1, u_2] + c[u_2, u_3] + \dots + c[u_{k-1}, u_k] .$$

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 trong $n-k+1$ đoạn đường còn lại đều không nhiều hơn c_{\min} , nên cận dưới cho phương án bộ phận (u_1, u_2, \dots, u_k) có thể được tính theo công thức

$$g(u_1, u_2, \dots, u_k) = \partial + (n - k + 1) c_{\min}.$$

Chẳng hạn ta giải bài toán người du lịch với ma trận chi phí như sau

$$C = \begin{vmatrix} 0 & 3 & 14 & 18 & 15 \\ 3 & 0 & 4 & 22 & 20 \\ 17 & 9 & 0 & 16 & 4 \\ 6 & 2 & 7 & 0 & 12 \\ 9 & 15 & 11 & 5 & 0 \end{vmatrix}$$

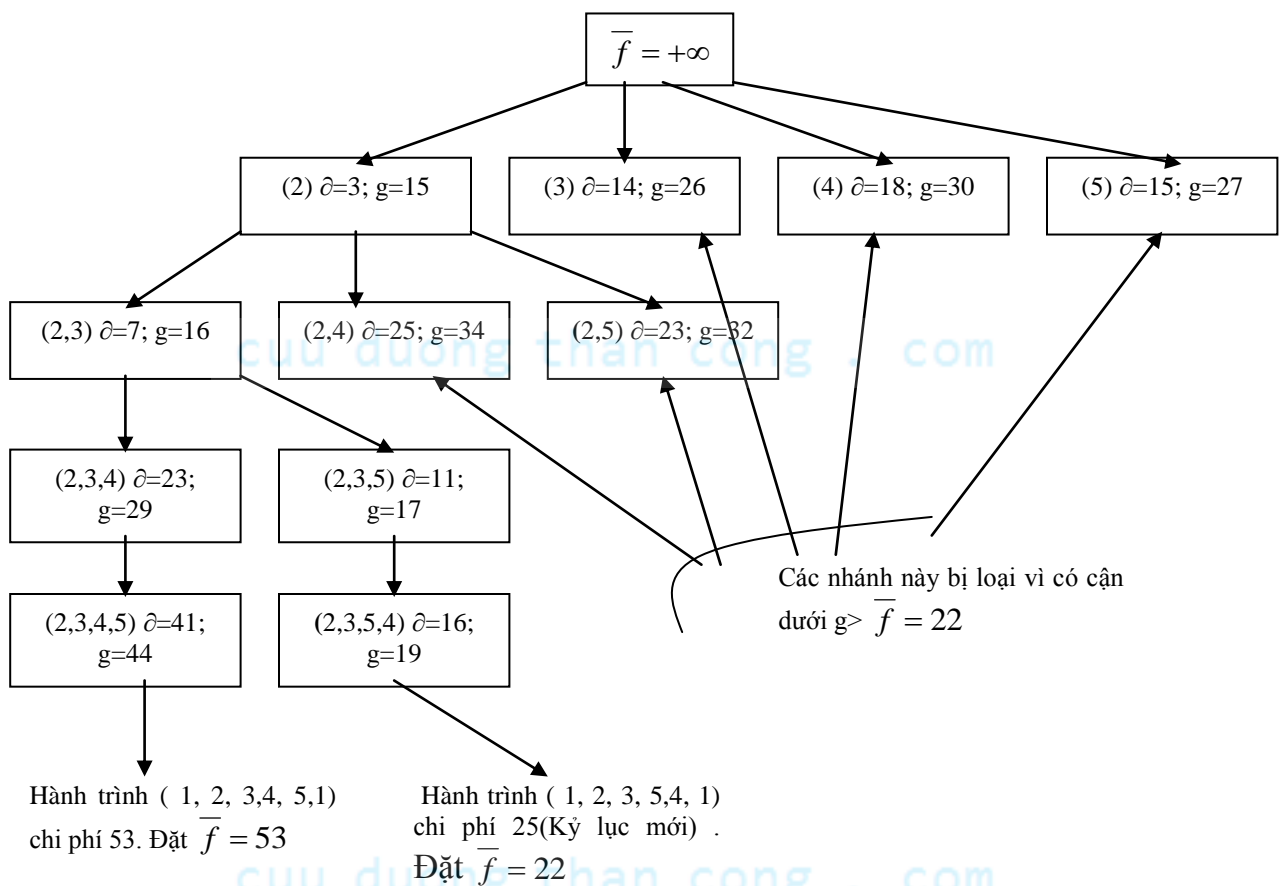
Ta có $c_{\min} = 2$. Quá trình thực hiện thuật toán được mô tả bởi cây tìm kiếm lời giải được thể hiện trong hình 4.2.

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 ∂ là chi phí theo hành trình bộ phận
- ☐ g là cận dưới

Kết thúc thuật toán, ta thu được phương án tối ưu (1, 2, 3, 5, 4, 1) tương ứng với phương án tối ưu với hành trình

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_5 \rightarrow T_4 \rightarrow T_1$ và chi phí nhỏ nhất là 22



Hình 4.2. Cây tìm kiếm lời giải bài toán người du lịch.

Chương trình giải bài toán theo thuật toán nhánh cận được thể hiện như sau:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <io.h>
```

```

#define MAX 20
int n, P[MAX], B[MAX], C[20][20], count=0;
int A[MAX], XOPT[MAX];
int can, cmin, fopt;
void Read_Data(void){
    int i, j; FILE *fp;
    fp = fopen("dulich.in", "r");
    fscanf(fp, "%d", &n);
    printf("\n So thanh pho: %d", n);
    printf("\n Ma tran chi phi:");
    for (i=1; i<=n; i++){
        printf("\n");
        for(j=1; j<=n; j++){
            fscanf(fp, "%d", &C[i][j]);
            printf("%5d", C[i][j]);
        }
    }
}
int Min_Matrix(void){
    int min=1000, i, j;
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++){
            if (i!=j && min>C[i][j])
                min=C[i][j];
        }
    }
    return(min);
}
void Init(void){
    int i;
    cmin=Min_Matrix();
    fopt=32000; can=0; A[1]=1;
    for (i=1; i<=n; i++)
        B[i]=1;
}

```

```

}
void Result(void){
    int i;
    printf("\n Hanh trinh toi uu %d:", fopt);
    printf("\n Hanh trinh:");
    for(i=1; i<=n; i++)
        printf("%3d->", XOPT[i]);
    printf("%d",1);
}
void Swap(void){
    int i;
    for(i=1; i<=n; i++)
        XOPT[i]=A[i];
}
void Update_Kyluc(void){
    int sum;
    sum=can+C[A[n]][A[1]];
    if(sum<fopt) {
        Swap();
        fopt=sum;
    }
}
void Try(int i){
    int j;
    for(j=2; j<=n; j++){
        if(B[j]){
            A[i]=j; B[j]=0;
            can=can+C[A[i-1]][A[i]];
            if (i==n) Update_Kyluc();
            else if( can + (n-i+1)*cmin< fopt){
                count++;
                Try(i+1);
            }
            B[j]=1; can=can-C[A[i-1]][A[i]];
        }
    }
}

```

```

    }
}
}
void main(void){
    clrscr();Read_Data();Init();
    Try(2);Result();
    getch();

}

```

4.4- Kỹ thuật rút gọn giải quyết bài toán người du lịch

Thuật toán nhánh cận là phương pháp chủ yếu để giải các bài toán tối ưu tổ hợp. Tư tưởng cơ bản của thuật toán là trong quá trình tìm kiếm lời giải, ta sẽ phân hoạch tập các phương án của bài toán thành hai hay nhiều tập con biểu diễn như một node của cây tìm kiếm và cố gắng bằng phép đánh giá cận các node, tìm cách loại bỏ những nhánh cây (những tập con các phương án của bài toán) mà ta biết chắc chắn không phương án tối ưu. Mặc dù trong trường hợp tồi nhất thuật toán sẽ trở thành duyệt toàn bộ, nhưng trong những trường hợp cụ thể nó có thể rút ngắn đáng kể thời gian tìm kiếm. Mục này sẽ thể hiện khác những tư tưởng của thuật toán nhánh cận vào việc giải quyết bài toán người du lịch.

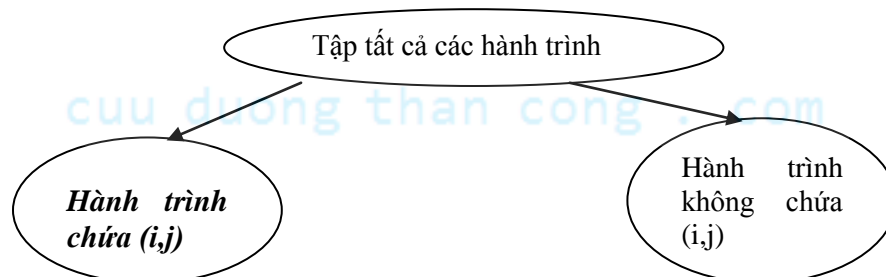
Xét bài toán người du lịch như đã được phát biểu. Gọi $C = \{c_{ij}: i, j = 1, 2, \dots, n\}$ là ma trận chi phí. Mỗi hành trình của người du lịch

$T_{\pi(1)} \rightarrow T_{\pi(2)} \rightarrow \dots \rightarrow T_{\pi(n)} \rightarrow T_{\pi(1)}$ có thể viết lại dưới dạng

$(\pi(1), \pi(2), \pi(2), \pi(3), \dots, \pi(n-1), \pi(n), \pi(n), \pi(1))$, trong đó mỗi thành phần

$\pi(j-1), \pi(j)$ sẽ được gọi là một cạnh của hành trình.

Khi tiến hành tìm kiếm lời giải bài toán người du lịch chúng ta phân tập các hành trình thành 2 tập con: Tập những hành trình chứa một cặp cạnh (i, j) nào đó còn tập kia gồm những hành trình không chứa cạnh này. Ta gọi việc làm đó là sự phân nhánh, mỗi tập con như vậy được gọi là một nhánh hay một node của cây tìm kiếm. Quá trình phân nhánh được minh họa bởi cây tìm kiếm như trong hình 4.3.



(Hình 4.3)

Việc phân nhánh sẽ được thực hiện dựa trên một qui tắc heuristic nào đó cho phép ta rút ngắn quá trình tìm kiếm phương án tối ưu. Sau khi phân nhánh và tính cận dưới giá trị hàm mục tiêu trên mỗi tập con. Việc tìm kiếm sẽ tiếp tục trên tập con có giá trị cận dưới nhỏ hơn. Thủ tục này được tiếp tục

cho đến khi ta nhận được một hành trình đầy đủ tức là một phương án của bài toán. Khi đó ta chỉ cần xét những tập con các phương án nào có cận dưới nhỏ hơn giá trị của 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 rất nhiều tập con chắc chắn không chứa phương án tối ưu. Sau đây, là một kỹ thuật nữa của thuật toán.

4.4.1. 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 cộng hay 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 đều giảm đi α vì thế hành trình tối ưu cũng sẽ không bị thay đổi. Vì vậy, nếu ta tiến hành bớt đi các phần tử của mỗi dòng và mỗi cột đi một hằng số sao cho ta thu được một ma trận gồm các phần tử không âm mà trên mỗi dòng, mỗi cột đều có ít nhất một số 0, thì tổng các số trừ đó cho ta cận dưới của mọi hành trình. Thủ tục bớt này 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), ma trận thu được được gọi là ma trận rút gọn. Thủ tục sau cho phép rút gọn ma trận một ma trận A kích thước $k \times k$ đồng thời tính tổng các hằng số rút gọn.

```
float Reduce( float A[][max], int k) {
    sum = 0;
    for (i = 1; i ≤ k; i++) {
        r[i] = <phần tử nhỏ nhất của dòng i>;
        if (r[i] > 0) {
            <Bớt mỗi phần tử của dòng i đi r[i]>;
            sum = sum + r[i];
        }
    }
    for (j = 1; j ≤ k; j++) {
        s[j] := <Phần tử nhỏ nhất của cột j>;
        if (s[j] > 0)
            sum = sum + S[j];
    }
    return(sum);
}
```

Ví dụ. Giả sử ta có ma trận chi phí 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

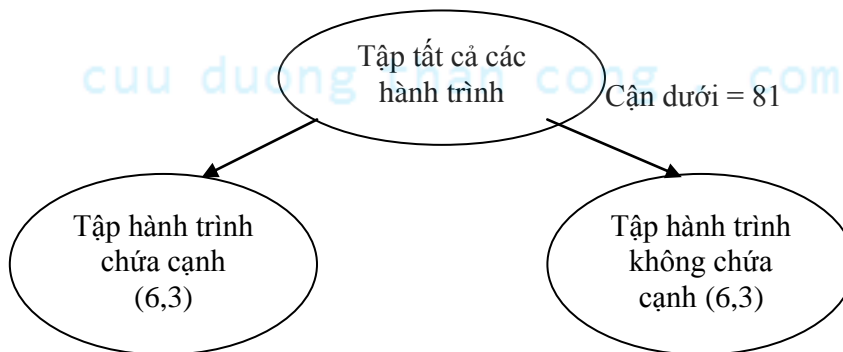
4	39	90	80	∞	56	7	7
5	28	46	88	33	∞	25	25
6	3	88	18	46	92	∞	3
	0	0	15	8	0	0	

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 ta tìm được phần tử nhỏ khác 0 của cột 3 và 4 tương ứng là (15, 8). Thực hiện rút gọn theo cột ta nhận được ma trận 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 (không thể có hành trình có chi phí nhỏ hơn 81).

Bây giờ ta xét cách phân tập các phương án ra thành hai tập. Giả sử ta chọn cạnh (6, 3) để phân nhánh. Khi đó tập các hành trình 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) không tham gia vào hành trình nên ta cấm hành trình đi qua cạnh này bằng cách đặt $C[6, 3] = \infty$. Ma trận thu được sẽ có thể rút gọn bằng cách bớt đi mỗi phần tử của cột 3 đi 48 (hàng 6 giữ nguyên). Như vậy ta thu được cận dưới của hành trình không chứa cạnh (6,3) là $81 + 48 = 129$. Còn đối với tập chứa cạnh (6, 3) ta phải loại dòng 6, 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 kỳ nơi nào khác và cũng không được phép đi bất cứ đâu từ 3. Kết quả nhận được là 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 cần cấm đi theo cạnh (3, 6) bằng cách đặt $C(3, 6) = \infty$. Cây tìm kiếm lúc này có dạng như trong hình 4.4.



(Hình 4.4)

Cận dưới =81						Cận dưới = 129						
	1	2	4	5	6		1	2	3	4	5	6
1	∞	0	2	30	6	1	∞	0	27	2	30	6
2	0	∞	30	17	12	2	0	∞	10	30	17	12
3	29	1	12	0	∞	3	29	1	∞	12	0	12
4	32	83	∞	49	0	4	32	83	10	∞	49	0
5	3	21	0	∞	0	5	3	21	0	0	∞	0
						6	0	85	∞	35	89	∞

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 áp dụng ở 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 1. 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 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. Thủ tục đó có thể được mô tả như sau để chọn cạnh phân nhánh (r, c).

4.4.2. Thủ tục chọn cạnh phân nhánh (r,c)

void BestEdge(A, k, r, c, beta)

Đầu vào: Ma trận rút gọn A kích thước $k \times k$

Kết quả ra: Cạnh phân nhánh (r,c) và tổng hằng số rút gọn theo dòng r cột c là beta.

{

beta = $-\infty$;

for (i = 1; i ≤ k; i++) {

for (j = 1; j ≤ k; j++) {

if (A[i,j] == 0) {

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) {

beta = total;

r = i; / Chỉ số dòng tốt nhất*/*

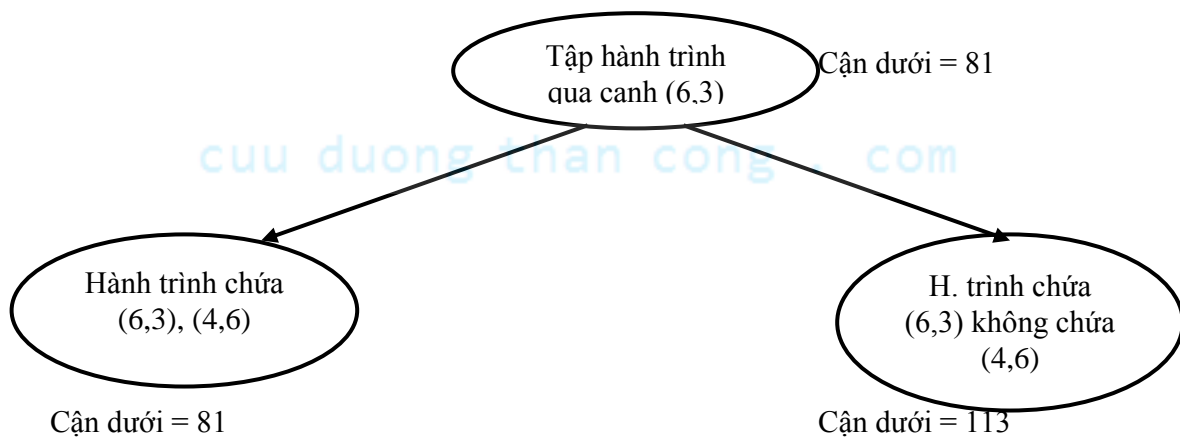
```

        c = j; /* Chỉ số cột tốt nhất*/
    }
}
}
}
}

```

Trong ma trận rút gọn 5×5 của nhánh bên trái hình 5.4, số không ở vị trí (4, 6) sẽ cho tổng hàng số rút gọn là 32 (theo dòng 4 là 32, cột 6 là 0). Đây là hệ số rút gọn có giá trị lớn nhất đối với các số không của ma trận này. Việc phân nhánh tiếp tục sẽ dựa vào cạnh (4, 6). Khi đó cận dưới của nhánh bên phải tương ứng với tập hành trình đi qua cạnh (6,3) nhưng không đi qua cạnh (4, 6) sẽ là $81 + 32 = 113$. Còn nhánh bên trái sẽ tương ứng với ma trận 4×4 (vì ta phải loại bỏ dòng 4 và cột 6). Tình huống phân nhánh này được mô tả trong hình 4.5.

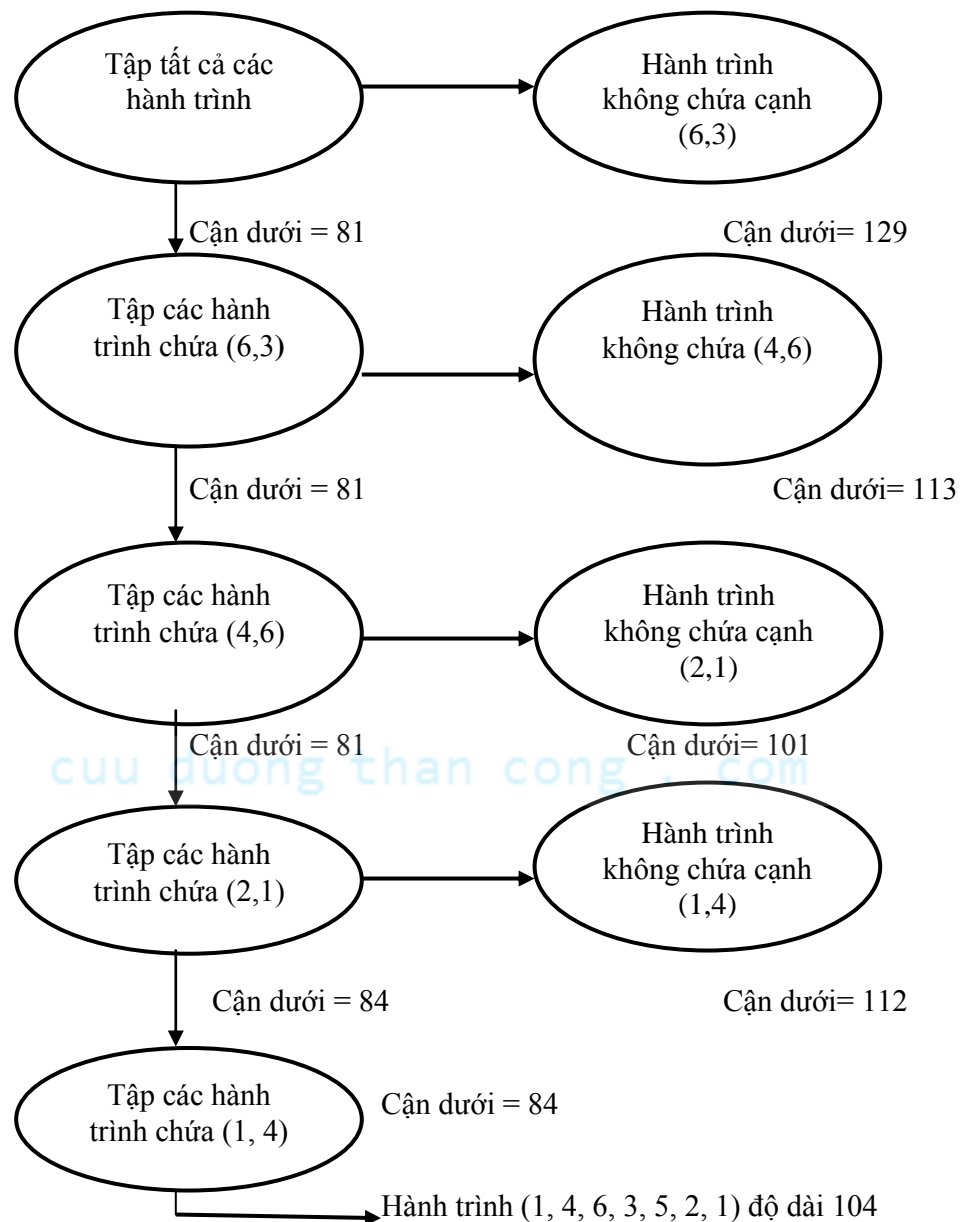
Nhận thấy rằng vì cạnh (4, 6) và (6, 3) đã nằm trong hành trình nên cạnh (3, 4) không thể đi qua được nữa (nếu đi qua ta sẽ có một hành trình con từ những thành phố này). Để ngăn cấm việc tạo thành các hành trình con ta sẽ gán cho phần tử ở vị trí (3, 4) giá trị ∞ .



(Hình 4.5)

Ngăn cấm tạo thành hành trình con:

Tổng quát hơn, khi phân nhánh dựa vào cạnh (i_u, i_v) ta phải thêm cạnh này vào danh sách các cạnh của node bên trái nhất. Nếu i_u là đỉnh cuối của một đường đi (i_1, i_2, \dots, i_u) và j_v là đỉnh đầu của đường đi (j_1, j_2, \dots, j_k) thì để ngăn ngừa khả năng tạo thành hành trình con ta phải ngăn ngừa khả năng tạo thành hành trình con ta phải cấm cạnh (j_k, i_1) . Để tìm i_1 ta đi ngược từ i_u , để tìm j_k ta đi xuôi từ j_1 theo danh sách các cạnh đã được kết nạp vào hành trình.



Hình 4.6 mô tả quá trình tìm kiếm giải pháp tối ưu

Tiếp tục phân nhánh từ đỉnh bên trái bằng cách sử dụng cạnh (2,1) vì số không ở vị trí này có hằng số rút gọn lớn nhất 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 bên trái có kích thước là 3×3 . Vì đã đi qua (2, 1) nên ta cấm cạnh (2, 1) bằng cách đặt $C[1, 2] = \infty$, 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 tại cột 1 và bớt 2 đi ở dòng 1 để nhận được ma trận cấp 3:

	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 5.5.

Chú ý rằng, sau khi đã chấp nhận $n-2$ cạnh vào hành trình thì ma trận còn lại sẽ có kích thước là 2×2 . Hai cạnh còn lại của hành trình sẽ không phải chọn lựa nữa mà được kết nạp ngay vào chu trình (vì nó chỉ còn sự lựa chọn duy nhất). Trong ví dụ trên sau khi đã có các cạnh (6, 3), (4, 6), (2, 1), (1, 4) ma trận của nhánh bên trái nhất có dạng:

	2	5
3	∞	0
5	0	∞

Vì vậy ta kết nạp nốt cạnh (3, 5), (5, 2) vào chu trình và thu được hành trình:

1, 4, 6, 3, 5, 2, 1 với chi phí là 104.

Trong quá trình tìm kiếm, mỗi node 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 xuống 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 2×2 thì ta chấm dứt việc phân nhánh và kết nạp hai cạnh còn lại để thu được hành trình của người du lịch. Dễ dàng nhận thấy ma trận cuối cùng rút gọn chỉ có thể ở một trong hai dạng sau:

	w	x
u	∞	0
v	0	∞

	w	x
u	0	∞
v	∞	0

Trong đó u, v, x, y có thể là 4 đỉnh khác nhau hoặc 3 đỉnh khác nhau. Để xác định xem hai cạnh nào cần được nạp vào hành trình ta chỉ cần xét một phần tử của ma trận A:

if $A[1, 1] = \infty$ *then*
 <Kết nạp cạnh (u, x), (v, w)>
else

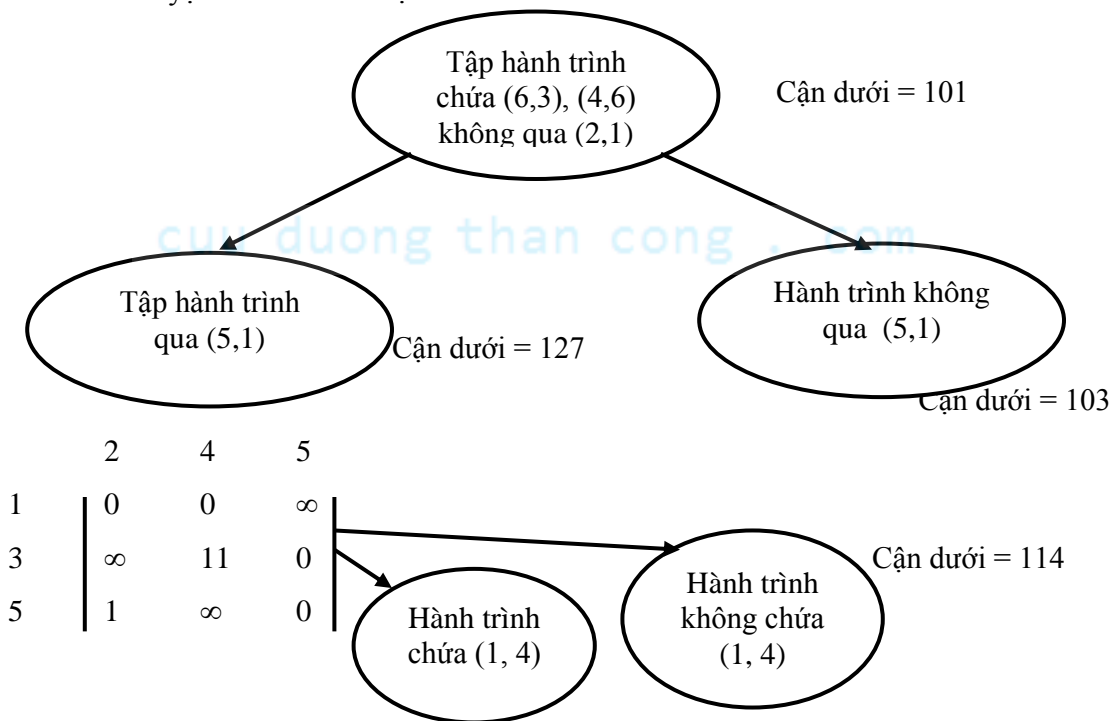
< Kết nạp cạnh $(u, w), (v, x) >;$

Bây giờ tất cả các node có cận dưới lớn hơn 104 có thể bị loại bỏ vì chúng không chứa hành trình rẻ hơn 104. Trên hình 5.4 chúng ta thấy chỉ có node có cận dưới là $101 < 104$ là cần phải xét tiếp. Node này 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 số rút gọn là 26. Quá trình rẽ nhánh tiếp theo được chỉ ra như trong hình 5.6.

Hình 5.6. Duyệt hành trình có cận dưới là 101.



Hành trình 1, 4, 6, 3, 2, 5, 1 ; Độ dài 104.

Như vậy chúng ta thu được hai hành trình tối ưu với chi phí là 104. Ví dụ trên cho thấy bài toán người du lịch có thể có nhiều phương án tối ưu. Trong ví dụ này hành trình đầu tiên nhận được đã là tối ưu, tuy nhiên điều này không thể mong đợi đối với những trường hợp tổng quát. Trong ví dụ trên chúng ta chỉ cần xét tới 13 node, trong khi tổng số hành trình của người du lịch là 120.

4.4.3. Thuật toán nhánh cận giải bài toán người du lịch

Các bước chính của thuật toán nhánh cận giải bài toán người du lịch được thể hiện trong thủ tục TSP. Thủ tục TSP xét hành trình bộ phận với Edges là cạnh đã được chọn và tiến hành tìm kiếm tiếp theo. Các biến được sử dụng trong thủ tục này là:

Edges - Số cạnh trong hành trình bộ phận;

A - Ma trận chi phí tương ứng với kích thước (n-edges, n-edges)

cost - Chi phí của hành trình bộ phận.

Mincost- Chi phí của hành trình tốt nhất đã tìm được.

Hàm Reduce(A, k), BestEdge(A, k, r, c, beta) đã được xây dựng ở trên.

void TSP(Edges, cost, A) {

cost=cost + Reduce(A, n-Edges);

if (cost <MinCost){

if (edges == n-2){

<bổ xung nốt hai cạnh còn lại>;

MinCost :=Cost;

}

else {

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 xung dòng r cột c>;

if (LowerBound < MinCost){

/ đi theo nhánh phải*/*

A[r, c] = ∞;

TSP (edges, cost, A);

A[r,c] :=0;

}

}

< Khôi phục ma trận A>;/ thêm lại các hằng số rút gọn vào*

các dòng và cột tương ứng/*

}

/ end of TSP*/;*

4.5. Những điểm cần ghi nhớ

Bạn đọc cần ghi nhớ một số nội dung quan trọng dưới đây:

- ✓ Thế nào là một bài toán tối ưu? Ý nghĩa của bài toán tối ưu trong các mô hình thực tế.
- ✓ Phân tích ưu điểm, nhược điểm của phương pháp liệt kê.
- ✓ Hiểu phương pháp nhánh cận, phương pháp xây dựng cận và những vấn đề liên quan
- ✓ Hiểu phương pháp rút gọn ma trận trong giải quyết bài toán người du lịch.

BÀI TẬP CHƯƠNG 4

Bài 1. Giải các bài toán cái túi sau:

$$\text{a) } \begin{cases} 5x_1 + x_2 + 9x_3 + 3x_4 \rightarrow \max, \\ 4x_1 + 2x_2 + 7x_3 + 3x_4 \leq 10, \\ x_j \in \{0,1\}, j = 1,2,3,4. \end{cases}$$

$$\text{b) } \begin{cases} 7x_1 + 3x_2 + 2x_3 + x_4 \rightarrow \max, \\ 5x_1 + 3x_2 + 6x_3 + 4x_4 \leq 12, \\ x_j \in \{0,1\}, j = 1,2,3,4 \end{cases}$$

Bài 2. Giải bài toán cái túi sau:

$$\begin{cases} 30x_1 + 19x_2 + 13x_3 + 38x_4 + 20x_5 + 6x_6 + 8x_7 + 19x_8 + 10x_9 + 11x_{10} \rightarrow \max, \\ 15x_1 + 12x_2 + 9x_3 + 27x_4 + 15x_5 + 5x_6 + 8x_7 + 20x_8 + 12x_9 + 15x_{10} \leq 62 \\ x_j \in \{0,1\}, j = 1,2,\dots,10. \end{cases}$$

Bài 3. Giải bài toán người du lịch với ma trận chi phí như sau:

∞	31	15	23	10	17
16	∞	24	07	12	12
34	03	∞	25	54	25
15	20	33	∞	50	40
16	10	32	03	∞	23
18	20	13	28	21	∞

Bài 4. Giải bài toán người du lịch với ma trận chi phí như sau:

∞	03	93	13	33	09
04	∞	77	42	21	16
45	17	∞	36	16	28
39	90	80	∞	56	07
28	46	88	33	∞	25
03	88	18	46	92	∞

cuu duong than cong . com

cuu duong than cong . com