HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

TOÁN RỜI RẠC 1

NGUYỄN DUY PHƯƠNG

Hà Nội 2016

CHƯƠNG 3. BÀI TOÁN LIỆT KÊ

Nội dung bài toán đếm là đếm xem có bao nhiêu cấu hình tổ hợp thỏa mãn một số tính chất nào đó. Bài toán liệt không chỉ đếm được các cấu hình tổ hợp thỏa mãn các tính chất đặt ra mà còn xem xét từng cấu hình tổ hợp đó là gì. Đối với mỗi bài toán, khi chưa tìm được thuật giải thì liệt kê được xem là biện pháp cuối cùng để thực hiện với sự hỗ trợ của máy tính. Có thể nói, liệt kê được xem là phương pháp giải vạn năng một bài toán bằng máy tính. Nội dung chính của chương này tập chung giải quyết những vấn đề cơ bản sau:

- ✓ Giới thiệu bài toán liệt kê.
- ✓ Thuật toán và độ phức tạp tính toán.
- ✓ Giải quyết bài toán liệt kê bằng phương pháp sinh.
- ✓ Giải quyết bài toán liệt kê bằng phương pháp quay lui.

Bạn đọc có thể tìm thấy cách giải nhiều bài toán liệt kê 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 bất kỳ một cấu hình nào.*
- *Không được bỏ sót bất kỳ một cấu hình nào.*

Ví dụ 1. Cho tập hợp các số $a_1, a_2, .., a_n$ 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.

Lời 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 thoả 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.

Lời giải: Vì thành phố xuất phát đã được xác định. Do vậy thương nhân có thể chọn tuỳ 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. Thuật toán và độ phức tạp tính toán

3.2.1. Ví dụ và Định nghĩa

Định nghĩa. Dãy hữ hạn các thao tác sơ cấp $F=F_1F_2..F_n(Input)\rightarrow Output$ được gọi là một thuật toán trên tập thông tin vào Input để có được kết qua ra Output. Dãy các thao tác sơ cấp ở đây được hiểu là các phép toán số học, các phép toán logic, các phép toán so sánh.

Một thuật toán cần thỏa mãn các tính chất dưới đây:

- **Tính đơn định**. Ở mỗi bước của thuật toán, các thao tác sơ cấp phải hết sức rõ ràng, không gây nên sự lộn xộn, nhập nhằng, đa nghĩa. Thực hiện đúng các bước của thuật toán trên tập dữ liệu vào, chỉ cho duy nhất một kết quả ra.
- **Tính dừng**. Thuật toán không được rơi vào quá trình vô hạn. Phải dừng lại và cho kết quả sau một số hữu hạn các bước.
- **Tính đúng**. Sau khi thực hiện tất cả các bước của thuật toán theo đúng qui trình đã định, ta phải nhận được kết quả mong muốn với mọi bộ dữ liệu đầu vào. Kết quả đó được kiểm chứng bằng yêu cầu của bài toán.
- **Tính phổ dụng**. Thuật toán phải dễ sửa đổi để thích ứng được với bất kỳ bài toán nào trong lớp các bài toán cùng loại và có thể làm việc trên nhiều loại dữ liệu khác nhau.
- **Tính khả thi**. Thuật toán phải dễ hiểu, dễ cài đặt, thực hiện được trên máy tính với thời gian cho phép.

3.2.2. Phương pháp biểu diễn thuật toán:

Thông thường, để biểu diễn một thuật toán ta có thể sử dụng các phương pháp sau:

- **Biểu diễn bằng ngôn ngữ tự nhiên**. Ngôn ngữ tự nhiên là phương tiện giao tiếp giữa con người với con người. Ta có thể sử dụng chính ngôn ngữ này vào việc biểu diễn thuật toán.

- **Ngôn ngữ hình thức**. Ngôn ngữ hình thức là phương tiện giao tiếp trung gian giữa con người và hệ thống máy tính. Ví dụ ngôn ngữ sơ đồ khối, ngôn ngữ tựa tự nhiên, ngôn ngữ đặc tả. Đặc điểm chung của các loại ngôn ngữ này là việc sử dụng nó rất gần với ngôn ngữ tự nhiên và ngôn ngữ máy tính.
- **Ngôn ngữ máy tính**. Là phương tiện giao tiếp giữa máy tính và máy tính. Trong trường hợp này ta có thể sử dụng bất kỳ nôn ngữ lập trình nào để mô tả thuật toán.

Ghi chú. Trong các phương pháp biểu diễn thuật toán, phương pháp biểu diễn bằng ngôn ngữ hình thức được sử dụng rộng dãi vì nó gần với ngôn ngữ tự nhiên và không phụ thuộc vào ngôn ngữ máy tính.

Ví dụ 1. Biểu diễn thuật toán tìm USCLN (a, b) bằng ngôn ngữ tự nhiên.

Đầu vào (Input). Hai số tự nhiên a, b.

Đầu ra (Output). Số nguyên u lớn nhất để a và b đều chia hết cho u.

Thuật toán (Euclide Algorithm):

Bước 1. Đưa vào hai số tự nhiên a và b.

Bước 2. Nếu b≠ 0 thì chuyển đến bước 3, nếu b=0 thì thực hiện bước 4.

Bước 3. Đặt r = a mod b; a = b; b = r ; Sau đó quay trở lại bước 2.

Bước 4 (Output). Kết luận u=a là số nguyên cần tìm.

Ví dụ 2. Biểu diễn Biểu diễn thuật toán tìm USCLN (a, b)bằng ngôn ngữ hình thức.

Thuật toán Euclide:

Đầu vào (*Input*): a∈N, a∈N.

Đầu ra (*Output*): s = max { u ∈N : a mod u =0 and b mod u =0}.

Format : s = Euclide (a, b).

Actions :

```
while (b ≠ 0 )  do
        r = a mod b; a = b; b = r;
endwhile;
return(a);
```

Endactions.

Ví dụ 3. Biểu diễn thuật toán tìm USCLN (a, b) bằng ngôn ngữ máy tính (C++).

```
Int     USCLN( int  a, int  b) {
        while ( b != 0 ) {
                r = a % b; a = b; b = r;
```

```
        }
        return(a);
}
```

3.2.3. Độ phức tạp tính toán

Một bài toán có thể thực hiện bằng nhiều thuật toán khác nhau. Chọn giải thuật nhanh nhất giải bài toán là một nhu cầu của thực tế. Vì vậy ta cần phải có sự ước lượng cụ thể để minh chứng bằng toán học mức độ nhanh chậm của mỗi giải thuật.

Khái niệm độ phức tạp thuật toán:

Thời gian thực hiện một giải thuật bằng chương trình máy tính phụ thuộc vào các yếu tố:

- Kích thước dữ liệu vào: Dữ liệu càng lớn thì thời gian xử lý càng chậm.
- Phần cứng máy tính: máy có tốc độ cao thực hiện nhanh hơn trên máy có tốc độ thấp. Tuy vậy, yếu tố này không ảnh hưởng đến quá trình xác định thời gian thực hiện của thuật toán nếu xem xét thời gian thực hiện thuật toán như một hàm của độ dài dữ liệu T(n).

Tổng quát, *cho hai hàm f(x), g(x) xác định trên tập các số nguyên dương hoặc tập các số thực vào tập các số thực. Hàm f(x) được gọi là O(g(x)) nếu tồn tại một hằng số C>0 và* n_0 *sao cho:*

$$|f(x)| \le C.|g(x)| \text{ với mọi } x \ge n_0.$$

Điều này có nghĩa với các giá trị $x \ge n_0$ hàm f(x) bị chặn trên bởi hằng số C nhân với g(x). Nếu f(x) là thời gian thực hiện của một thuật toán thì ta nói giải thuật đó có cấp g(x) hay độ phức tạp thuật toán là O(g(x)).

Ghi chú. Các hằng số C, n0 thỏa mãn điều kiện trên là không duy nhất. Nếu có đồng thời f(x) là O(g(x)) và h(x) thỏa mãn g(x) < h(x) với x>n0 thì ta cũng có f(x) là O(h(n)).

Ví dụ 1. Cho $f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$. Trong đó, a_i là các số thực (i =0,1, 2, ..,n). Khi đó $f(x) = O(x^n)$.

Chứng minh. Thực vậy, với mọi x>1:

$$
\begin{aligned}
|f(x)| &= \left|a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0\right| \\
&\cdots \le |a_n| x^n + |a_{n-1}| x^{n-1} + ... + |a_1| x + |a_0| \\
&\cdots \le |a_n| x^n + |a_{n-1}| x^n + ... + |a_1| x^n + |a_0| x^n \\
&\cdots \le x^n \left(|a_n| + |a_{n-1}| + ... + |a_1| + |a_0|\right) \\
&\cdots \le C.x^n = O(x^n).
\end{aligned}
$$

$$
C = \left(|a_n| + |a_{n-1}| + ... + |a_1| + |a_0|\right)
$$

Ví dụ 2. Tìm độ phức tạp thuật toán sắp xếp kiểu Bubble-Sort?

```
Void Bubble-Sort ( int  A[],  int n ) {
        for ( i=1; i<n; i++) {
                for ( j = i+1; j<=n; j++){
                        if (A[i] > A[j]) {
                                t = A[i]; A[i] = A[j]; A[j] = t;
                        }
                }
        }
}
```

Lời giải. Sử dụng trực tiếp nguyên lý cộng ta có:

- Với i =1 ta cần sử dụng n-1 phép so sánh A[i] với A[j];
- Với i = 2 ta cần sử dụng n-1 phép so sánh A[i] với A[j];
- . . . . .
- Với i = n-1 ta cần sử dụng 1 phép so sánh A[i] với A[j];

Vì vậy tổng số các phép toán cần thực hiện là:

$$S = (n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2 \leq n2 = O(n2).$$

Ghi chú. Độ phức tạp thuật toán cũng là số lần thực hiện phép toán tích cực. Phép toán tích cực là phép toán thực hiện nhiều nhất đối với thuật toán.

Một số tính chất của độ phức tạp thuật toán:

- Với P(n) là một đa thức bậc k thì $O(P(n)) = O(n^k)$. Vì thế ta nói, một thuật toán có độ phức tạp cấp đa thức là O(nk).
- Với a, b là hai cơ số tùy ý và f(n) là một hàm xác định dương thì $\log_a f(n)=\log_a b.\log_b(f(n))$. Vì vậy độ phức tạp thuật toán cấp logarit được ký hiệu là O(log(f(n)) mà không cần quan tâm đến cơ số.
- Nếu độ phức tạp thuật toán là hằng số, nghĩa là thời gian tính toán không phụ thuộc vào độ dài dữ liệu được ký hiệu là O(1).
- Một giải thuật có cấp 2^n, n!, n^n được gọi là giải thuật hàm mũ. Những giải thuật này thường có tốc đọ rất chậm.
- Độ phức tạp tính toán của một đoạn chương trình P chính bằng số lần thực hiện một phép toán tích cực. Trong đó, phép toán tích cực trong một đoạn chương trình là phép toán mà số lần thực hiện nó không ít hơn các phép toán khác.

Các dạng hàm đánh giá độ phức tạp thuật toán:

Dạng đánh giá	Tên gọi
O(1)	Hằng số
O(lg lg n)	Log log
O(lg n)	Logarithm
O(n)	Tuyến tính
O(n2)	Bậc hai
O(n3)	Bậc 3
O(nm)	Đa thức
O(mn)	Hàm mũ
O(n!)	Giai thừa

3.2.4. Qui tắc xác định độ phức tạp thuật toán

Qui tắc tổng: Nếu $f_1(x)$ có độ phức tạp là $O(g_1(x))$ và $f_2(x)$ có độ phức tạp là $O(g_2(x))$ thì độ phức tạp của $(f_1(x) + f_2(x)$ là $O(Max(g_1(x), g_2(x))$.

Chứng minh.

- Vì $f_1(x)$ có độ phức tạp là $O(g_1(x)$ nên tồn tại hằng số C_1 và k_1 sao cho $|f_1(x)|\leq|g_1(x)|$ với mọi $x \leq k1$;
- Vì $f_2(x)$ có độ phức tạp là $O(g_2(x))$ nên tồn tại hằng số C_2 và k_2 sao cho $|f2(x)|\leq|g2(x)|$ với mọi $x \leq k2$;
- Ta lại có :

$$|f_1(x)+ f_2(x)| \leq |f_1(x)| + |f_2(x)|$$

$$\leq C_1|g_1(x)| + C_2|g_2(x)|$$

$$\leq C|g(x)| \text{ với mọi } x > k;$$

Trong đó, $C = C_1 + C_2$; $g(x) = \max(g_1(x), g_2(x))$; $k = \max (k_1, k_2)$.

Tổng quát. Nếu độ phức tạp của $f_1(x)$, $f_2(x)$,.., fm(x) lần lượt là $O(g_1(x))$, $O(g_2(x))$,.., $O(g_n(x))$ thì độ phức tạp của $f_1(x) + f_2(x) + ..+f_m(x)$ là $O(\max(g_1(x), g_2(x),..,g_m(x))$.

Qui tắc nhân: Nếu f(x) có độ phức tạp là O(g(x) thì độ phức tạp của $f^n(x)$ là O($g^n(x)$. Trong đó:

$f^n(x) = f(x).f(x)….f(x)$. //n lần f(x).

$g^n(x) = g(x).g(x)…g(x)$.//n lần g(x)

Nói cách khác, đoạn chương trình P có thời gian thực hiện T(n)= O(f(n)). Khi đó, nếu thực hiện k(n) lần đoạn chương trình P với k(n) là O(g(n)) thì độ phức tạp tính toán là O(f(n). g(n)).

Chứng minh. Thật vậy theo giả thiết f(x) là O(g(x)) nên tồn tại hằng số C và k sao cho với mọi x>k thì |f(x)|≤ C.|g(x). Ta có:

$$\begin{aligned} \left|f^n(x)\right| &= \left|f^1(x).f^2(x)...f^n(x)\right| \\ &\le \left|C.g^1(x).C.g^2(x)...C.g^n(x)\right| \\ &\le C^n\left|g^n(x)\right| = O\left(g^n(x)\right) \end{aligned}$$

3.2.5. Độ phức tạp của các cấu trúc lệnh

Để đánh giá độ phức tạp của một thuật toán đã được mã hóa thành chương trình máy tính ta thực hiện theo một số qui tắc sau.

Độ phức tạp hằng số O(1): đoạn chương trình không chứa vòng lặp hoặc lời gọi đệ qui có tham biến là một hằng số.

Ví dụ 1.7. Đoạn chương trình dưới đây có độ phức tạp hằng số.

```
for (i=1; i<=c; i++) {
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

Độ phức tạp O(n): Độ phức tạp của hàm hoặc đoạn code là O(n) nếu biến trong vòng lặp tăng hoặc giảm bởi mộ hằng số c.

Ví dụ 1.8. Đoạn code dưới đây có độ phức tạp hằng số.

```
for (i=1; i<=n; i = i + c ) {
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
for (i=n; i>0; i = i - c ){
        <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

Độ phức tạp đa thức O(n^c): Độ phức tạp của *c* vòng lặp lồng nhau, mỗi vòng lặp đều có độ phức tạp O(n) là **O(n^c).**

Ví dụ 1.9. Đoạn code dưới đây có độ phức tạp O(n^2).

```
for (i=1; i<=n; i = i + c ) {
```

```
        for (j=1; j<=n; j = j + c ){
            <Tập các chỉ thị có độ phức tạp O(1)>;
        }
    }
    for (i = n; i >0 ; i = i - c ) {
        for (j = i- 1; j>1; j = j -c ){
            <Tập các chỉ thị có độ phức tạp O(1)>;
        }
    }
```

Độ phức tạp logarit O(Log(n)): Độ phức tạp của vòng lặp là log(n) nếu biểu thức khởi đầu lại của vòng lặp được chia hoặc nhân với một hằng số c.

Ví dụ 1.10. Đoạn code dưới đây có độ phức tạp Log(n).

```
for (i=1; i <=n; i = i *c ){
    <Tập các chỉ thị có độ phức tạp O(1)>;
}
for (j=n; j >0 ; j = j / c ){
    <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

Độ phức tạp hằng số O(Log (Log(n))): nếu biểu thức khởi đầu lại của vòng lặp được nhân hoặc chia cho một hàm mũ.

Ví dụ 1.11. Đoạn code dưới đây có độ phức tạp Log Log(n).

```
for (i=1; j<=n; j*= Pow(i, c) ){
    <Tập các chỉ thị có độ phức tạp O(1)>;
}
for (j=n; j>=0; j = j- Function(j)  ){  //Function(j) =sqrt(j) hoặc lớn hơn 2.
    <Tập các chỉ thị có độ phức tạp O(1)>;
}
```

Độ phức tạp của chương trình: độ phức tạp của một chương trình bằng số lần thực hiện một chỉ thị tích cực trong chương trình đó. Trong đó, một chỉ thị được gọi là tích cực trong chương trình nếu chỉ thị đó phụ thuộc vào độ dài dữ liệu và thực hiện không ít hơn bất kỳ một chỉ thị nào khác trong chương trình.

Ví dụ 1.12. Tìm độ phức tạp thuật toán sắp xếp kiểu Bubble-Sort?

```
Void Bubble-Sort ( int  A[],  int n ) {
    for ( i=1; i<n; i++) {
        for ( j = i+1; j<=n; j++){
            if (A[i] > A[j]) {//đây chính là chỉ thị tích cực
                t = A[i]; A[i] = A[j]; A[j] = t;
```

```
                    }
                }
            }
        }
```

Lời giải. Sử dụng trực tiếp nguyên lý cộng ta có:

- Với i =1 ta cần sử dụng n-1 phép so sánh A[i] với A[j];
- Với i = 2 ta cần sử dụng n-1 phép so sánh A[i] với A[j];
- . . . . .
- Với i = n-1 ta cần sử dụng 1 phép so sánh A[i] với A[j];

Vì vậy tổng số các phép toán cần thực hiện là:

$S = (n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2 \leq n^2 = O(n^2)$.

Ghi chú. Độ phức tạp thuật toán cũng là số lần thực hiện phép toán tích cực. Phép toán tích cực là phép toán thực hiện nhiều nhất đối với thuật toán.

3.3. Phương pháp sinh

Mô hình thuật toán sinh được dùng để giải lớp các bài toán liệt kê, bài toán đếm, bài toán tối ưu, bài toán tồn tại thỏa mãn hai điều kiện:

- **Điều kiện 1:** *Có thể xác định được một thứ tự trên tập các cấu hình cần liệt kê của bài toán. Biết cấu hình đầu tiên, biết cấu hình cuối cùng.*
- **Điều kiện 2:** *Từ một cấu hình chưa phải cuối cùng, ta xây dựng được thuật toán sinh ra cấu hình đứng ngay sau nó.*

Mô hình thuật toán sinh được biểu diễn thành hai bước: bước khởi tạo và bước lặp. Tại bước khởi tạo, cấu hình đầu tiên của bài toán sẽ được thiết lập. Điều này bao giờ cũng thực hiện được theo giả thiết của bài toán. Tại bước lặp, quá trình lặp được thực hiện khi gặp phải cấu hình cuối cùng. Điều kiện lặp của bài toán bao giờ cũng tồn tại theo giả thiết của bài toán. Hai chỉ thị cần thực hiện trong thân vòng lặp là đưa ra cấu hình hiện tại và sinh ra cấu hình kế tiếp. Mô hình sinh kế tiếp được thực hiện tùy thuộc vào mỗi bài toán cụ thể. Tổng quát, mô hình thuật toán sinh được thể hiện như dưới đây.

```
Thuật toán Generation;
begin
    Bước1 (Khởi tạo):
        <Thiết lập cấu hình đầu tiên>;
    Bước 2 (Bước lặp):
        while (<Lặp khi cấu hình chưa phải cuối cùng>) do
            <Đưa ra cấu hình hiện tại>;
            <Sinh ra cấu hình kế tiếp>;
    endwhile;
```

End.

Ví dụ 1. Vector $X = (x_1, x_2, .., x_n)$, trong đó $x_i = 0, 1$ được gọi là một xâu nhị phân có độ dài *n*. Hãy liệt kê các xâu nhị phân có độ dài *n*. Ví dụ với n=4, ta sẽ liệt kê được 24 xâu nhị phân độ dài 4 như trong Bảng 2.1.

Bảng 2.1. Các xâu nhị phân độ dài 4

STT	$X=(x_1, x_2, x_3, x_4)$	STT	$X=(x_1, x_2, x_3, x_4)$
0	**0 0 0 0**	**8**	**1 0 0 0**
1	**0 0 0 1**	**9**	**1 0 0 1**
2	**0 0 1 0**	**10**	**1 0 1 0**
3	**0 0 1 1**	**11**	**1 0 1 1**
4	**0 1 0 0**	**12**	**1 1 0 0**
5	**0 1 0 1**	**13**	**1 1 0 1**
6	**0 1 1 0**	**14**	**1 1 1 0**
7	**0 1 1 1**	**15**	**1 1 1 1**

Lời giải:

Điều kiện 1: Gọi thứ tự của xâu nhị phân $X=(x_1, x_2,.., x_n)$ là $f(X)$. Trong đó, $f(X)= k$ là số chuyển đổi xâu nhị *X* thành số ở hệ cơ số 10. Ví dụ, xâu $X = (1, 0, 1, 1)$ được chuyển thành số hệ cơ số 10 là 11 thì ta nói xâu *X* có thứ tự 11. Với cách quan niệm này, xâu đứng sau xâu có thứ tự 11 là 12 chính là xâu đứng ngay sau xâu $X = (1, 0, 1, 1)$. Xâu đầu tiên có thứ tự là *0* ứng với xâu có *n* số *0*. Xâu cuối cùng có thứ tự là 2^n-1 ứng với xâu có *n* số *1*. Như vậy, điều kiện 1 của thuật toán sinh đã được thỏa mãn.

Điều kiện 2: Về nguyên tắc ta có thể lấy $k = f(X)$ là thứ tự của một xâu bất kỳ theo nguyên tắc ở trên, sau đó lấy thứ tự của xâu kế tiếp là $(k + 1)$ và chuyển đổi $(k+1)$ thành số ở hệ cơ số 10 ta sẽ được xâu nhị phân tiếp theo. Xâu cuối cùng sẽ là xâu có n số 1 ứng với thứ tự $k = 2^n-1$. Với cách làm này, ta có thể coi mỗi xâu nhị phân là một số, mỗi thành phần của xâu là một bít và chỉ cần cài đặt thuật toán chuyển đổi cơ số ở hệ 10 thành số ở hệ nhị phân. Ta có thể xây dựng thuật toán tổng quát hơn bằng cách xem mỗi xâu

nhị phân là một mảng các phần tử có giá trị 0 hoặc 1. Sau đó, duyệt từ vị trí bên phải nhất của xâu nếu gặp số 1 ta chuyển thành 0 và gặp số 0 đầu tiên ta chuyển thành 1. Ví dụ với xâu X = (0, 1, 1, 1) được chuyển thành xâu X= (1, 0, 0, 0), xâu X = (1,0,0,0) được chuyển thành xâu X =(1, 0, 0, 1). Lời giải và thuật toán sinh xâu nhị phân kế tiếp được thể hiện trong chương trình dưới đây. Trong đó, thuật toán sinh xâu nhị phân kế tiếp từ một xâu nhị phân bất kỳ là hàm *Next_Bits_String*().

```
#include <iostream>
#include <iomanip>
#define MAX 100
using namespace std;
int X[MAX], n, dem = 0; //sử dụng các biến toàn cục X[], n, OK, dem
bool OK =true;
void Init(void){ //khởi tạo xâu nhị phân đầu tiên
        cout<<"Nhập n="; cin>>n;
        for(int i = 1; i<=n; i++) //thiết lập xâu với n số 0
                X[i]=0;
}
void Result(void){ //đưa ra xâu nhị phân hiện tại
        cout<<"\n Xâu thứ "<<++dem<<":";
        for(int i=1; i<=n; i++)
           cout<<X[i]<<setw(3);
}
void Next_Bits_String(void){ //thuật toán sinh xâu nhị phân kế tiếp
        int i=n;
        while(i>0 && X[i]){ //duyệt từ vị trí bên phải nhất
                X[i]=0; //nếu gặp X[i] = 1 ta chuyển thành 0
                i--; //lùi lại vị trí sau
        }
        if (i>0)  X[i]=1; //gặp X[i] =0 đầu tiên ta chuyển thành 1
        else OK = false; //kết thúc khi gặp xâu có n số 1
}
int main(void){ //đây là thuật toán sinh
        Init(); //thiết lập cấu hình đầu tiên
        while(OK){//lặp khi chưa phải cấu hình cuối cùng
                Result(); //đưa ra cấu hình hiện tại
                Next_Bits_String(); //sinh ra cấu hình kế tiếp
        }
```

}

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

Lời giải: Mỗi tập con của tập hợp X có thể biểu diễn bằng bộ có thứ tự gồm k thành phần $a = (a_1 a_2 .. a_k)$ thoả mãn $1 \le a_1 \le a_2 \le . . \le a_k \le n$. 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 $a = a_1 a_2 . . . a_k$ đi trước tập con $a' = a_1' a_2' . . . a_k'$ trong thứ tự từ điển và ký hiệu là a<a', nếu tìm được chỉ số j ($1 \le j \le k$) sao cho

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

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, . ., k) và tập con cuối cùng là (n-k+1, n-k+2, . ., n). Giả sử $a = (a_1, a_2, . ., a_k)$ 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 $a_1, a_2, . ., a_k$ phần tử $a_i \ne n - k + i$
- Thay a_i bởi $a_i + 1$,
- Thay a_j bởi $a_i + j - i$, với j:= i+1, i + 2, . . ., 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 $a_3 = a_2 + 3 - 2 = 3 + 3 - 2 = 4$, $a_4 = a2 + 4 - 2 = 3 + 4 - 2 = 5$ 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(void){

```
i = k; //Xuất phát từ vị trí thứ k
while ( i> 0 && a_i == n-k+i) //Xác định i để a_i ≠ n-k+i
        i = i -1;
if (i>0) { //Nếu chưa phải là tổ hợp cuối cùng thì i>0
        a_i = a_i + 1;
        for ( j = i+1; j <=k; j++)
                a_j = a_i + j - i;
}
else OK =False; ////Nếu là tổ hợp cuối cùng thì i=0
```

Dưới đây là chương trình liệt kê tổ hợp chập k của 1, 2, .., n.

Chương trình cài đặt thuật toán sinh tập con k phần tử được thể hiện như dưới đây. Trong đó, thuật toán sinh tổ hợp kế tiếp có tên là Next_Combination().

```
#include <iostream>
#include <iomanip>
#define MAX 100
int X[MAX], n, k, dem=0;
bool OK = true;
using namespace std;
void Init(void){ //thiết lập tập con đầu tiên
        cout<<"\n Nhập n, k:"; cin>>n>>k;
        for(int i=1; i<=k; i++)  //tập con đầu tiên là 1, 2, .., k
                X[i] = i;
}



void Result(void){ //đưa ra tập con hiện tại
        cout<<"\n Kết quả "<<++dem<<":";
                for(int i=1; i<=k; i++) //đưa ra X[] =( x1, x2, .., xk)
                cout<<X[i]<<setw(3);
}
void Next_Combination(void){ //sinh tập con k phần tử từ tập con bất kỳ
        int i = k; //duyệt từ vị trí bên phải nhất của tập con
        while(i>0 && X[i]== n-k+i) //tìm i sao cho x_i ≠n-k+i
                i--;
```

```
        if (i>0){//nếu chưa phải là tập con cuối cùng
                X[i]= X[i]+1; //thay đổi giá trị tại vị  trí i: x_i = x_i +1;
                for(int j=i+1; j<=k; j++) //các vị trí j từ i+1,.., k
                        X[j] = X[i] + j - i; // được thay đổi là x_j = x_i +j - i;
        }
        else  //nếu là tập con cuối cùng
                OK = false; //ta kết thúc duyệt
}
int main(void){
        Init(); //khởi tạo cấu hình đầu tiên
        while(OK){ //lặp trong khi cấu hình chưa phải cuối cùng
                Result(); //đưa ra cấu hình hiện tại
                Next_Combination(); //sinh ra cấu hình kế tiếp
        }
}
```

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

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

$a = (a_1, a_2, .., a_n)$ thoả mãn $a_i \in X$, i = 1, 2, .., 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 $a = a_1a_2 \ldots a_n$ đi trước hoán vị $a' = a_1'a_2' \ldots a_n'$ trong thứ tự từ điển và ký hiệu là $a<a'$, nếu tìm được chỉ số k ($1 \le k \le n$) sao cho

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

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

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

Như vậy, hoán vị đầu tiên trong thứ tự từ điển là (1, 2, …, n) và hoán vị cuối cùng là (n, n-1, . . ., 1). Giả sử $a = a_1a_2 \ldots a_n$ 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 thoả 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 thoả 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( void  ){
    j = n-1; //Duyệt từ vị trí j=n-1
    while (j> 0 && a_j > a_j+1 ) //Tìm vị trí j để a_j > a_j+1
        j = j -1;
    if (j>0) { // Nếu j>0 thì hoán vị chưa phải cuối cùng
        k = n; //Xuất phát từ vị trí k=n
        while (a_j > a_k ) // Tìm k để  a_j < a_k
            k= k - 1;
        temp =a_j;  a_j = a_k; a_k = temp;//Đổi chỗ a_j cho a_k
        r = j + 1; s = n;
        while ( r < s) {//Lật ngược lại đoạn từ j+1 đến n
            temp = a_r; a_r = a_s; a_s = temp;
            r = r +1;       s = s - 1;
        }
    }
    else OK = False;//Nếu là hoán vị cuối cùng thì i=0
}
```

Chương trình liệt kê hoán vị được thể hiện như sau:

```
#include <iostream>
```

```
#include <iomanip>
#define MAX 100
int X[MAX], n,  dem=0;
bool OK = true;
using namespace std;
void Init(void){ //thiết lập hoán vị đầu tiên
        cout<<"\n Nhap n:"; cin>>n;
        for(int i=1; i<=n; i++) //thiết lập X[] = (1, 2, ..,n)
                X[i] = i;
}
void Result(void){ //đưa ra hoán vị hiện tại
        cout<<"\n Kết quả "<<++dem<<":";
        for(int i=1; i<=n; i++)
           cout<<X[i]<<setw(3);
}
void Next_Permutation(void){ //sinh ra hoán vị kế tiếp
        int j = n-1; //xuất phát từ vị trí j = n-1
        while(j>0 && X[j]>X[j+1]) //tìm chỉ số j sao cho X[j] < X[j+1]
                j--;
        if ( j > 0){ // nếu chưa phải hoán vị cuối cùng
                int k = n; //xuất phát từ vị trí k = n
                while(X[j]>X[k]) //tìm chỉ số k sao cho X[j] < X[k]
                        k--;
                int t = X[j]; X[j] = X[k]; X[k]=t; //đổi chỗ X[j] cho X[k]
                int r = j+1, s = n;
                while (r<=s){ //lật ngược lại đoạn từ j+1,..,n
                        t=X[r]; X[r]=X[s]; X[s]=t;
                        r++; s--;
                }
        }
        else  //nếu là cấu hình cuối cùng
                OK = false; //ta kết thúc duyệt
}
int main(void){ //đây là thuật toán sinh
        Init(); //thiết lập cấu hình đầu tiên
        while(OK){ //lặp trong khi cấu hình chưa phải cuối cùng
                Result(); //đưa ra cấu hình hiện tại
```

```
        Next_Permutation(); //sinh ra cấu hình kế tiếp
    }
    }
```

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.

Lời 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. Chọn cách phân chia số $n = b_1 + b_2 + \ldots + b_k$ với $b_1 > b_2 > \ldots > b_k$, và duyệt theo trình tự từ điển ngược. Chẳng hạn với n = 5, chúng ta có thứ tự từ điển ngược của các cách phân chia như sau:

5				
4	1			
3	2			
3	1	1		
2	2	1		
2	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; //Xuất phát từ cuối cách chia trước đó
    while(i>0 && C[i]==1) //Tìm i sao cho C[i]≠1
        i--;
    if(i>0){ // Nếu chưa phải là cách chia cuối cùng thì i>0
        C[i] = C[i]-1; //Giảm C[i] đi một đơn vị.
        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;
}
```

Chương trình liệt kê các cách chia số n thành tổng các số nhỏ hơn:

```
#include <iostream.h>
#include <stdlib.h>
#define  MAX   100
#define  TRUE  1
#define  FALSE 0
int n, k, X[MAX], dem =0, OK =TRUE;
void Init(void ){
      cout<<"\n Nhap n=";cin>>n; k = 1; X[k] = n;
}
void Result(void) {
      cout<<"\n Cach chia "<<++dem<<":";
      for (int i=1; i<=k; i++)
     cout<<X[i]<<"   ";
}
void Next_Division(void ){
      int i = k, j, R, S,D;
      while (i > 0 && X[i]==1 ) i--;
      if (i>0 ) {
            X[i] = X[i] - 1;
            D = k - i + 1;
            R = D / X[i];
            S = D % X[i];
            k= i;
            if (R>0) {
                  for ( j = i +1; j<=i + R; j++) X[j] = X[i];
                  k = k + R;
            }
            if (S>0 ){ k = k +1; X[k] = S; }
      }
      else OK =0;
}
int main() {
      Init();
      while (OK ) {
            Result();
            Next_Division();
```

```
        }
        system("PAUSE");
        return 0;
}
```

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 $x = (x_1, x_2, . ., x_n)$ mà i-1 thành phần $x_1, x_2, . ., x_{i-1}$ 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 . .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}.

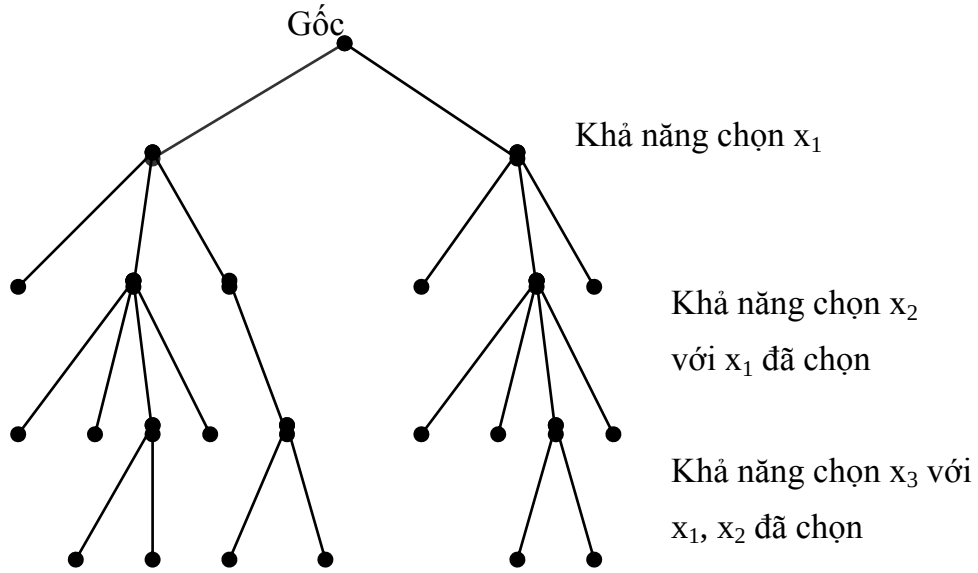
Điểm quan trọng nhất của thuật toán là phải ghi nhớ lại mỗi bước đã đi qua, những khả năng nào đã được thử để tránh sự trùng lặp. Để nhớ lại những bước duyệt trước đó, chương trình cần phải được tổ chức theo cơ chế ngăn xếp (Last in first out). Vì vậy, thuật toán quay lui rất phù hợp với những phép gọi đệ qui. 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 )  {
    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:

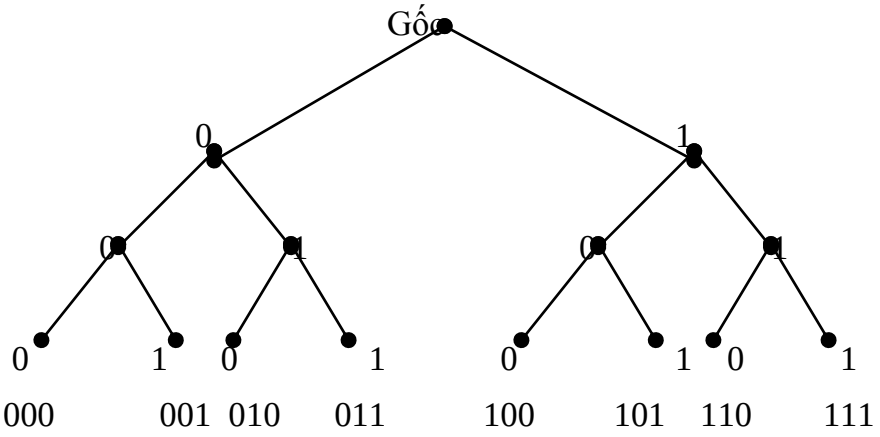


Hình 3.1. Cây liệt kê lời giải theo thuật toán quay lui.

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 $b_1, b_2,..., b_n$, trong đó $bi \in \{0, 1\}$. Thủ tục đệ qui 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 thoả 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.



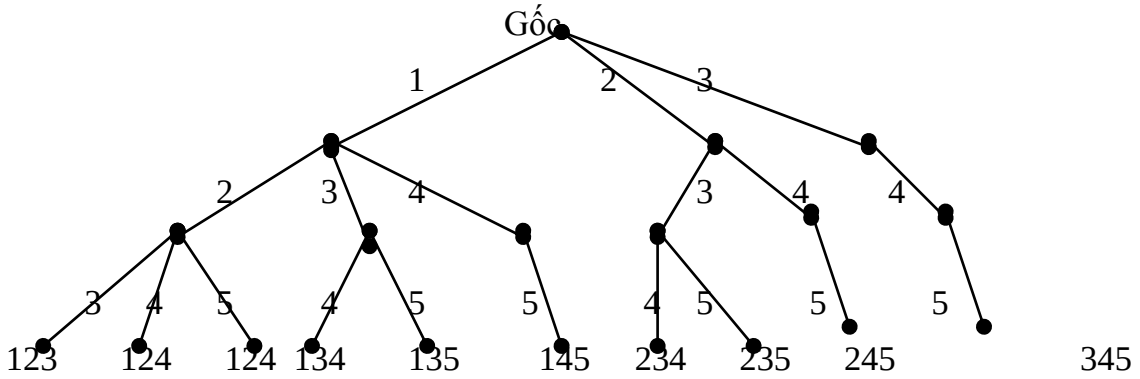
Hình 3.2. Cây tìm kiếm lời giải liệt kê dãy nhị phân độ dài 3

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 <iostream >
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n, X[MAX], dem=0;
void Init (void ){      cout<<"\n Nhap n=";cin>>n;}
void Result(void){
        cout<<"\n Ket qua buoc "<<++dem<<":";
        for (int i=1; i<=n; i++)
                cout<<X[i]<<"   ";
}
void Try (int i) {
        for (int j=0; j<=1; j++){
                X[i] = j;
                if (i==n) Result();
                else Try(i+1);
        }
}
int main(){
        Init(); //Nhap n = 4
        Try(1);system("PAUSE");return 0;
}
```

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 $c_1, c_2, . ., c_k$, trong đó $1< c_1<c_2 . .\leq n$. 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.



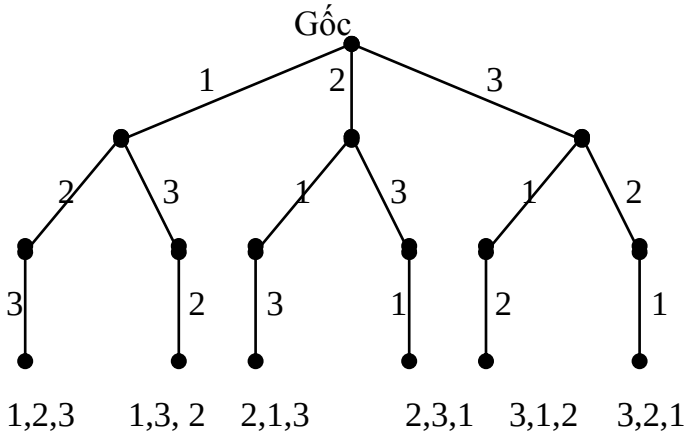
Hình 3.3. Cây liệt kê tổ hợp chập 3 từ {1, 2, 3, 4, 5 }

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

```
#include <iostream.h>
#include <stdlib.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n,k,X[MAX],dem=0;
void Init (void ){
        cout<<"\n Nhap n=";cin>>n;
        cout<<"\n Nhap k=";cin>>k;
        X[0] = 0;
}
void Result(void){
        cout<<"\n Ket qua buoc "<<++dem<<":";
        for (int i=1; i<=k; i++)
                cout<<X[i]<<"   ";
}
void Try(int i ) {
        for (int j = X[i-1]+1; j<= n-k+i; j++){
                X[i] = j;
                if (i==k) Result();
                else Try(i+1);
        }
}
int main(){
    Init(); //Nhap n = 5, k = 3
    Try(1);
    system("PAUSE");
    return 0;
}
```

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

Lời giải. Biểu diễn hoán vị dưới dạng $p_1, p_2, .., p_n$, trong đó p_i nhận giá trị từ 1 đến n và $p_i \neq p_j$ với $i \neq j$. Các giá trị từ 1 đến n 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 = 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 <iostream.h>
#include <stdlib.h>
#define  MAX  100
#define  TRUE  1
#define  FALSE 0
int n,X[MAX],chuaxet[MAX],dem=0;
void Init (void ){
        cout<<"\n Nhap n=";cin>>n;
        for (int i=1; i<=n; i++)   chuaxet[i] = TRUE;
}
void Result(void){
        cout<<"\n Ket qua buoc "<<++dem<<":";
        for (int i=1; i<=n; i++)   cout<<X[i]<<"   ";
}
void Try(int i){
      for (int j =1; j<=n; j++){
            if (chuaxet[j]) {
                  X[i] = j; chuaxet[j] = FALSE;
                  if (i == n) Result();
                  else Try(i+1);
                  chuaxet[j] = TRUE;
            }
      }
}
int main(){   Init(); //Nhap n = 4
      Try (1);  system("PAUSE");  return 0;
}
```

Ví dụ 4. Bài toán Xếp Hậu. Liệt kê tất cả các cách xếp n quân hậu trên bàn cờ n x n sao cho chúng không ăn được nhau.

Lời giải. Bàn cờ có n hàng được đánh số từ 1 đến n, n cột được đánh số từ 1 đến n; Bàn cờ có n*2 -1 đường chéo xuôi được đánh số từ 1 đến 2*n -1, 2 *n -1 đường chéo ngược được đánh số từ 1đến 2*n -1. Ví dụ: với bàn cờ 8 x 8, chúng ta có 8 hàng được đánh số từ 1 đến 8, 8 cột được đánh số từ 1 đến 8, 15 đường chéo xuôi, 15 đường chéo ngược được đánh số từ 1 . .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 $x_1, x_2, . ., x_n$, 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 1 đến n; giá trị của j cũng được nhận từ 1 đến n, 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 $chuaxet_j$ với qui ước $chuaxet_j$=True nếu cột j còn trống, cột $chuaxet_j$= False 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 = const và i - j = const, đường chéo thứ nhất được ghi nhận bởi dãy biến $XUOI_j$, đường chéo thứ 2 được ghi nhận bởi dãy biến $NGUOC_j$ với qui ước nếu đường chéo nào còn trống thì giá trị tương ứng của nó là True ngược lại là False. Như vậy, cột j được chấp nhận khi cả 3 biến $chuaxet_j$, $XUOI_{i+j}$, $NGUOC_{i+j}$ đều có giá trị 1. Các biến này phải được khởi đầu giá trị True trước đó, gán lại giá trị False khi xếp xong quân hậu thứ i và trả lại giá trị 1 khi đưa ra kết quả.

Dưới đây là chương trình bằng thuật toán quay lui.

```
#include <iostream.h>
#include <stdlib.h>
#define MAX 100
#define TRUE 1
#define FALSE 0
int X[MAX], XUOI[MAX], NGUOC[MAX], chuaxet[MAX];
int n, dem =0;
void Init (void ) {
        cout<<"\n Nhap n ="; cin>>n;
        for (int i=1; i<=n; i++) chuaxet[i]=TRUE;
        for (int i=1; i<=(2*n-1); i++) {
                XUOI[i] = TRUE; NGUOC[i]=TRUE;
        }
}
void Result(void ) {
        cout<<"\n Phuon an "<<++dem<<":";
        for (int i=1; i<=n; i++)        cout<<X[i]<<"  ";
}
```

```
void Try(int i){
        for (int j =1; j<=n; j++){
                if (chuaxet[j] && XUOI[i-j+n] && NGUOC[i+j-1]){
                        X[i] = j;chuaxet[j]=FALSE;
                        XUOI[i-j+n]=FALSE;NGUOC[i+j-1]=FALSE;
                        if (i==n ) Result();
                        else Try(i+1);
                        chuaxet[j]=TRUE;XUOI[i-j+n]=TRUE;
                       NGUOC[i+j-1]=TRUE;
                }
        }
 }
int main(){
        Init(); Try(1);
        system("PAUSE");
        return 0;
}
```

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?
- ✓ 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.

BÀI TẬP CHƯƠNG 3

1. Liệt kê tất cả các xâu nhị phân độ dài 5 khôn chứa hai số 0 liên tiếp.
2. Liệt kê tất cả các phần tử của tập

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

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

3. **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.

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

4. Tam giác thần bí. Cho một lưới ô vuông gồm n x 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.

5. **Tìm tập con dài nhất có thứ tự tăng dần, giảm dần.** Cho dãy số $a_1, a_2, \ldots, a_n$. 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≤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
```
5
7  1  3  8  9  6  12
```
tapcon.out
```
5
1  3  8  9  12
```

6. **Duyệt các tập con thoả mãn điều kiện**. Cho dãy số $a_1, a_2, \ldots, a_n$ và số M. Hãy tìm tất cả các dãy con dãy con trong dãy số $a_1, a_2, \ldots, a_n$ 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≤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
```

7. **Cho lưới hình c**hữ nhật gồm (n×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
```

8. Tìm bộ giá trị rời rạc để hàm mục tiêu sin(x1+x2 + . . .+ xk) đạt giá trị lớn nhất. Dữ liệu vào cho bởi file bai4.inp, kết quả ghi lại trong file bai4.out.
9. 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)?M

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

CHƯƠNG 4. BÀI TOÁN TỐI ƯU

Bài toán đếm thực hiện đếm các cấu hình tổ hợp thỏa mãn một số tính chất nào đó. Bài toán liệt kê xem xét từng cấu hình tổ hợp thỏa mãn các tính chất đặt ra. Bài toán tối ưu chỉ quan tâm đến nghiệm tốt nhất (xấu nhất) theo một nghĩa nào đó đặt ra của bài toán. 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.
- ✓ Phương pháp liệt kê giải quyết bài toán tối ưu.
- ✓ Phương pháp nhánh cận giải quyết bài toán tối ưu.
- ✓ Phương pháp qui hoạch động 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 náo 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.*

- Tập D gọi là tập các phương án của bài toán.
- Mỗi phần tử $x \in D$ được gọi là một phương án.
- Hàm $f(x)$ được gọi là hàm mục tiêu của bài toán.
- Phương án $x^* \in D$ đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là phương án tối ưu.
- Giá trị $f^* = f(x^*)$ được gọi là giá trị tối ưu của bài toán.

Dưới đây là 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 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 , . . , n). Hỏi nhà thám hiểm cần đem theo những đồ vật nào để cho tổng giá trị sử dụng là lớn nhất ?

Gọi $A = (a_1, a_2,..,a_n)$, $C = (c_1, c_2,..,c_n)$ tương ứng với vector trọng lượng và giá trị sử dụng các đồ vật. Khi đó ta có thể xác định được tập các phương án và hàm mục tiêu của bài toán như sau:

Tập các phương án của bài toán: Một phương án của nhà thám hiểm có thể biểu diễn như một vector nhị phân độ dài *n:* $X = (x_1,x_2, \ . \ ., x_n)$, trong đó $x_i = 1$ có nghĩa là đồ vật thứ *i* được đem theo, $x_i = 0$ có nghĩa đồ vật thứ i không được đem theo. Tập các xâu nhị phân $X =(x_1,..,x_n)$ còn phải thỏa mãn điều kiện tổng trọng lượng không vượt quá b. Nói cách khác, tập phương án *D* của bài toán được xác định như công thức dưới đây.

$$D = \left\{ X = (x_1, x_2,..., x_n) : \sum_{i=1}^{n} a_i x_i \le b \right\}$$

Hàm mục tiêu của bài toán: Với mỗi phương án $X \in D$, giá trị sử dụng các đồ vật đem theo là $f(X) = \sum_{i=1}^{n} c_i x_i$, tổng trọng lượng đồ vật đem theo là $g(X) = \sum_{i=1}^{n} a_i x_i$. Như vậy bài toán cái túi được phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

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

$$\text{tìm } \min\{ f(X) : X \in D \text{ và } g(x) \le b\}$$

Bài toán Người du lịch. Một người du lịch muốn đi thăm quan n thành phố $T_1, T_2, ..., T_n$. Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố 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 $(i,j = 1, 2, \ . \ ., n)$, hãy tìm hành trình với tổng chi phí là nhỏ nhất cho người du lịch (một hành trình là một cách đi thoả mãn điều kiện).

Không hạn chế tính tổng quát của bài toán ta giả sử người du lịch luôn xuất phát tại một thành phố cố định (giả sử là thành phố số 1). Khi đó, tập phương án và hàm mục tiêu của bài toán được xác định như sau.

Tập phương án của bài toán: Rõ ràng, mỗi hành trình của người du lịch có dạng $X = (x_1, x_2, .., x_n, x_1)$. $X = (x_1, x_2, .., x_n)$ là hoán vị của 1, 2,..,n và $x_1 = 1$ ở vị trí đầu tiên và cuối cùng *vì ta xem người du lịch xuất phát từ thành phố số 1* và sau đó người du lịch phải trở lại điểm xuất phát. Như vậy, ta chỉ có (n-1)! hành trình thực sự. Vì vậy, tập phương án của bài toán là

$$D = \{X = (x_1, x_2,..., x_n) : x_1 = 1; x_i \ne x_j : \forall i \ne j; i, j = 2,3,.., n\}.$$

Hàm mục tiêu của bài toán: Chi phí của mỗi hành trình ứng với mỗi $X \in D$ là tổng chi phí đi từ thành phố thứ *i* đến thành phố thứ *j*. Do vậy hàm mục tiêu của bài toán là:

$$f(X) = \sum_{i=1}^{n-1} C[X[i]][X[i+1]] \;\; C[X[n]][X[1]]$$
$$= C[X[1]][X[2]] + C[X[2]][X[3]] + ... + C[X[n-1]][X[n]] + C[X[n]][X[1]]$$

Như vậy, bài toán người du lịch có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

Tìm *min { f(X) : X* $\in$*D }*

Bài toán cho thuê máy. Một ông chủ có một cái máy để cho thuê. Đầu tháng ông ta nhận được yêu cầu thuê máy của *m* khách hàng. Mỗi khách hàng *i* sẽ cho biết tập N_i các ngày trong tháng cần sử dụng máy *(i = 1, 2, . ., m)*. Ông chủ chỉ có quyền hoặc từ chối yêu cầu của khách hàng *i*, hoặc nếu nhận thì phải bố trí mãy 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.

Tập phương án của bài toán. Ký hiệu, *I = { 1, 2, . ., m }* là tập chỉ số khách hàng, *S* là tập hợp các tập con của *I*. Khi đó, tập hợp tất cả các phương án cho thuê máy là

$$D = \{J \subset S : N_k \cap N_p = \phi, \forall k \neq p \in J \;\}.$$

Hàm mục tiêu của bài toán. Với mỗi phương án J $\in$ D, mỗi phương án cho thuê máy là tổng số ngày sử dụng máy theo phương án đó. Do vậy, hàm mục tiêu của bài toán là $f(j) = \sum_{j \in J} |N_j|$. Khi đó, bài toán có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

$$\max\{ f(j) : j \in D\}.$$

Bài toán phân công công việc. 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, . . ., 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.

Tập phương án của bài toán. Rõ ràng, mỗi phương án bố trí thợ thực hiện các công việc tương ứng với một hoán vị π *= (π(1), π(2) , . . ., π(n))*. Do vậy, tập các phương án của bài toán là tập các hoán vị của 1, 2,..,n.

Hàm mục tiêu của bài toán. Ứng với mỗi phương án, chi phí theo phương án là

$$f(\pi) = C_{\pi(1),1} + C_{\pi(2),2} + \cdots + C_{\pi(n),n}.$$

Công việc	Thợ thực hiện
1	π(1)
2	π(2)
. . .	. . .

N	π(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 $D_1, D_2, \ldots, D_n$ cần phải lần lượt được gia công trên m máy $M_1, M_2, \ldots, M_m$. Thời gian gia công chi tiết D_i trên mãy M_j là t_{ij}. Hãy tìm lịch (trình tự gia công) các chi tiết trên các mãy 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.

Tập phương án của bài toán. Rõ ràng, mỗi một lịch gia công các chi tiết trên các máy sẽ tương ứng với một hoán vị $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ của n số tự nhiên *1, 2,.., n*.

Hàm mục tiêu của bài toán. Thời gian hoàn thành theo các lịch trên được xác định bởi hàm số

$f(\pi) = \sum_{j=1}^{n-1} C_{\pi(j),\pi(j+1)} + \sum_{k=1}^{m} t_{k,\pi(n)}$, trong đó $c_{ij} = S_j - S_i$, S_j là thời điểm bắt đầu thực hiện việc gia công chi tiết *j (i, j = 1, 2, . . ., n)*. Ý nghĩa của hệ số c_{ij} có thể 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:

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

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

Trong thực tế, lịch gia công còn phải thoả 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. Phương pháp duyệt toàn bộ

Một trong những phương pháp hiển nhiên có thể giải bài toán tối ưu tổ hợp là duyệt tất cả các phương án của bài toán. Ứng với với mỗi phương án, ta đều tính được giá trị của 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 trên được gọi là phương pháp duyệt toàn bộ. Thuật toán tổng quát thực hiện theo phương pháp duyệt toàn bộ được thể hiện trong Hình 4.1.

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. Tuy vậy, cũng cần phải chú ý rằng nhiều bài toán tối ưu hiện nay vẫn chưa tìm ra được một phương pháp hữu hiệu nào ngoài phương pháp duyệt toàn bộ đã được đề cập ở trên.

Thuật toán duyệt toàn bộ:

Bước 1 (*Khởi tạo*)**:**

XOPT=∅; //*Khởi tạo phương án tối ưu ban đầu*

FOPT= - ∞ (+ ∞);//*Khởi tạo giá trị tối ưu ban đầu*

Bước 2(*Lặp*):

```
for each X∈D do { //lấy mỗi phần tử trên tập phương án
    S = f(X);// tính giá trị hàm mục tiêu cho phương án X
    if ( FOPT<S ) { //Cập nhật phương án tối ưu
        FOPT = S; //Giá trị tối ưu mới được xác lập
        XOPT = X;// Phương án tối ưu mới
    }
}
```

Bước 3 (*Trả lại kết quả*):

Return(XOPT, FOPT);

Hình 4.1. Thuật toán duyệt toàn bộ**.**

Ghi chú. Tại Bước 1 của thuật toán, ta khởi tạo giá trị của hàm mục tiêu FOPT = - ∞ với bài toán tìm **max**, khởi tạo giá trị của hàm mục tiêu FOPT = + ∞ với bài toán tìm **min**.

Ví dụ 1. *Giải bài toán cái túi dưới đây bằng phương pháp duyệt toàn bộ***.**

$$\begin{cases} F(X) = 4x_1 + 6x_2 + 3x_3 + 5x_4 + 2x_5 \to \max, \\ 9x_1 + 8x_2 + 5x_3 + 3x_4 + 2x_5 \le 21, \\ x_j \in \{0,1\},\ j = 1,2,3,4,5. \end{cases}$$

Lời giải. Gọi $A = (a_1, a_2,..,a_n)$, $C = (c_1, c_2,..,c_n)$ tương ứng với vector trọng lượng và giá trị sử dụng các đồ vật. Như đã phân tính trong Mục 4.1, ta có:

Tập các phương án của bài toán: $D = \left\{ X = (x_1, x_2,..., x_n) : \sum_{i=1}^{n} a_i x_i \le b \right\}$.

Hàm mục tiêu của bài toán: $f(X) = \sum_{i=1}^{n} c_i x_i$.

Nhiệm vụ của chúng ta là tìm phương án tối ưu XOPT và giá trị tối ưu FOPT. Kết quả thực theo thuật toán duyệt toàn bộ được thể hiện theo Bảng 4.1 dưới đây.

$X=(x_1,x_2,x_3,x_4,x_5)$	$X \in D = \{x_1, x_2, x_3, x_4, x_5 : \sum_{i=1}^{5} a_i x_i \le 21$	$F(X)= \sum_{i=1}^{5} c_i x_i = ?$
0, 0, 0, 0, 0	0≤21: X∈D	0
0, 0, 0, 0, 1	2≤21: X∈D	2
0, 0, 0, 1, 0	3≤21: X∈D	5
0, 0, 0, 1, 1	5≤21: X∈D	7
0, 0, 1, 0, 0	5≤21: X∈D	3
0, 0, 1, 0, 1	7≤21: X∈D	5
0, 0, 1, 1, 0	8≤21: X∈D	8
0, 0, 1, 1, 1	10≤21: X∈D	10
0, 1, 0, 0, 0	8≤21: X∈D	6
0, 1, 0, 0, 1	10≤21: X∈D	8
0, 1, 0, 1, 0	11≤21: X∈D	11
0, 1, 0, 1, 1	13≤21: X∈D	13
0, 1, 1, 0, 0	13≤21: X∈D	9
0, 1, 1, 0, 1	15≤21: X∈D	11
0, 1, 1, 1, 0	16≤21: X∈D	14
0, 1, 1, 1, 1	***18≤21: X∈D***	***16***
1, 0, 0, 0, 0	9≤21: X∈D	4
1, 0, 0, 0, 1	11≤21: X∈D	6
1, 0, 0, 1, 0	12≤21: X∈D	9
1, 0, 0, 1, 1	14≤21: X∈D	9
1, 0, 1, 0, 0	14≤21: X∈D	10
1, 0, 1, 0, 1	16≤21: X∈D	9
1, 0, 1, 1, 0	*16≤21: X∈D*	*12*
1, 0, 1, 1, 1	19>21: X∈D	14
1, 1, 0, 0, 0	17≤21: X∈D	10
1, 1, 0, 0, 1	19≤21: X∈D	12
1, 1, 0, 1, 0	20≤21: X∈D	15
1, 1, 0, 1, 1	22>21: X∉D	∞
1, 1, 1, 0, 0	22>21: X∉D	∞
1, 1, 1, 0, 1	24>21: X∉D	∞
1, 1, 1, 1, 0	25>21: X∉D	∞

1, 1, 1, 1, 1	27>21: X$\notin$D	∞
XOPT=(0,1,1,1,1); FOPT = 16		

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:

Tìm $\min\{f(x): x \in D\}$.

Trong đó D là tập hữu hạn phần tử. Ta giả thiết D được mô tả như sau:

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

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 $(a_1, a_2, \ldots, a_k)$ 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 thoả mãn bất đẳng thức sau:

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

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

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

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

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

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

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

$\bar{f} < g(a_1, a_2, \ldots, a_k) \le \min\{ f(x): x \in D, x_i = a_i, i=1, 2, \ldots, k \}$, vì thế tập con các phương án của bài toán $D(a_1, a_2, \ldots, a_k)$ chắc chắn không chứa phương án tối ưu. Trong

trường hợp này ta không cần phải phát triển phương án bộ phận $(a_1, a_2, \ldots, a_k)$, nói cách khác là ta có thể loại bỏ các phương án trong tập $D(a_1, a_2, \ldots, a_n)$ khỏi quá trình tìm kiếm.

Thuật toán quay lui liệt kê các phương án cần sửa đổi lại như Hình 4.2:

```
Procedure Try( k ){
/*Phát triển phương án bộ phận (a1, a2, . . ., ak-1 theo thuật toán quay
lui có kiểm tra cận dưới Trước khi tiếp tục phát triển phương án*/
    for ( ak ∈ Ak ) {
        if ( chấp nhận ak ){
            xk = ak;
            if (k == n)
                < cập nhật kỷ lục>;
            else if (g(a1, a2, . . ., ak) ≤ f̄ )
                Try (k+1);
        }
    }
}
```

Hình 4.2. Thuật toán quay lui dựa vào hàm đánh giá cận.

Khi đó, thuật toán nhánh cận được thực hiện như Hình 4.3.

```
Procedure Nhanh_Can( ) {
    f̄ = +∞; /* Nếu biết một phương án x̄ nào đó thì có thể đặt f̄ = f(x̄). */
        Try(1); //Thực hiện thuật toán quay lui trong Hình 4.2
        if ( f̄ ≤ +∞ )
            < f̄ là giá trị tối ưu , x̄ là phương án tối ưu >;
        else
            < bài toán không có phương án>;
}
```

Hình 4.3. Thuật toán nhánh cận.

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, . ., a_k) ≤ $\overline{f}$) 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. 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, . ., 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, . ., 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

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

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

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

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

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

$$\frac{c_1}{a_1} \ge \frac{c_2}{a_2} \ge \cdots \ge \frac{c_n}{a_n} \qquad (2)$$

Để xây dựng hàm tính cận dưới, cùng với bài toán cái túi *(1)* ta xét bài toán cái túi biến liên tục sau: Tìm

$$g^* = \max\left\{\sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \le b, x_j \ge 0,\ j = 1,2,\cdots,n\right\}. \qquad (3)$$

Mệnh đề. Phương án tối ưu của bài toán (3) là vector $\bar{x} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})$ với các thành phần được xác định bởi công thức:

$\overline{x_1} = \dfrac{b}{a_1}$, $\overline{x_2} = \overline{x_3} = \cdots = \overline{x_n} = 0$ và giá trị tối ưu là $g^* = \dfrac{c_1 b_1}{a_1}$.

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

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

suy ra:

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

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

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

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

ta có

$$\max\left\{f(x) : x \in D, x_j = u_j,\ j = 1,2,\cdots n\right\}$$

$$= \max\left\{\partial_k + \sum_{j=k+1}^{n} c_j x_j : \sum_{j=k+1}^{n} a_j x_j \le b_k, x_j \in Z_+,\ j = k+1, k+2, \cdots, n\right\}$$

$$\le \partial_k + \max\left\{\sum_{j=k+1}^{n} c_j x_j : \sum_{j=k+1}^{n} a_j x_j \le b_k, x_j \ge 0,\ j = k+1, k+2, \cdots, n\right\}$$

$$= \partial_k + \frac{c_{k+1} b_k}{a_{k+1}}$$

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

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

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

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 $0, 1, \ldots, [b_k / a_{k+1}]$. Do có kết quả của mệnh đề, khi chọn giá trị cho x_{k+1} ta sẽ duyệt

các giá trị đề cử theo thứ tự giảm dần. Thuật toán nhánh cận giải bài toán cái túi được mô tả như Hình 4.4 dưới đây.

Thuật toán Brach_And_Bound (i) {

t = (b - b_i)/A[i]; //Khởi tạo số lượng đồ vật thứ i

for j = t; j ≥0; j--){

x[i] = j; //Lựa chọn x[i] là j;

$b_i = b_i + a_i x_i$; *// Trọng lượng túi cho bài toán bộ phận thứ i.*

$\sigma_i = \sigma_i + c_i x_i$;*// Giá trị sử dụng cho bài toán bộ phận thứ i*

If (k==n) <Cập nhật kỷ lục>;

*else if (δ_k + (c_{k+1}*b_k)/a_{k+1} >FOPT) //Nhánh cận được triển khai tiếp theo*

Branch_And_Bound(k+1);

$b_i = b_i - a_i x_i$;

$\sigma_i = \sigma_i - c_i x_i$;

}

}

Hình 4.4. Thuật toán nhánh cận giải bài toán cái túi

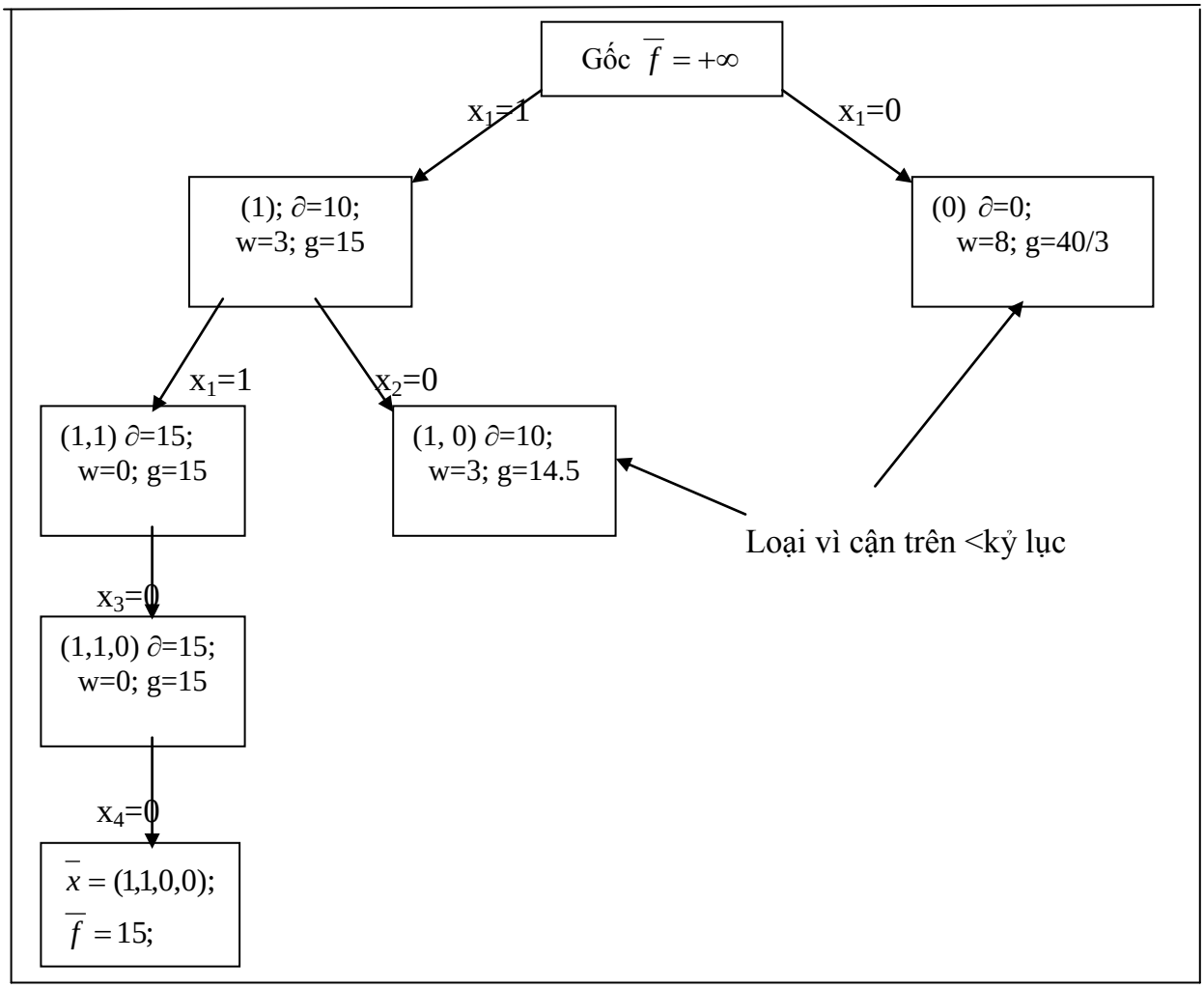
Ví dụ. Giải bài toán cái túi sau theo thuật toán nhánh cận trình bày ở trên.

$$f(x) = 10x_1 + 5x_2 + 3x_3 + 6x_4 \to \max$$
$$5x_1 + 3x_2 + 2x_3 + 4x_4 \le 8$$
$$x_j \in Z_+,\ j = 1,2,3,4.$$

Lời giải. Quá trình giải bài toán được mô tả trong cây tìm kiếm trong Hình 4.4. 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 bộ phận thứ i : $X = (x_1, x_2, .., x_i)$;
- Tiếp đến ∂ là giá trị của các đồ vật theo phương án bộ phận thứ i;
- Kế tiếp là *w* tương ứng với trọng lượng còn lại của túi;
- Cuối cùng là cận trên g.

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



Hình 4.4. Lời 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 <iostream.h>
#include <stdio.h>
#include <conio.h>
#define MAX 100
int A[MAX], C[MAX], F[MAX][MAX];
int XOPT[MAX], X[MAX];
int n, b, ind;
float W, FOPT=-32000, cost, weight=0;

FILE *fp;
```

```
void Init(void){
     fp = fopen("caitui1.in","r");
     fscanf(fp,"%d%d", &n, &b);
     cout<<"\n So luong do vat:"<<n;
     cout<<"\n Trong luong tui:"<<b;
     for( int i=1; i<=n; i++)
          fscanf(fp,"%d%d", &A[i],&C[i]);
     cout<<"\n Vector trong luong:";
     for( i=1; i<=n; i++)
          cout<<A[i]<<"   ";
     cout<<"\n Vector gia tri su dung:";
     for( i=1; i<=n; i++)
          cout<<C[i]<<"   ";
     fclose(fp);
}
void Update_Kyluc(void){
     if (cost>FOPT) { FOPT =cost;
          for(int i=1; i<=n; i++)
               XOPT[i] = X[i];
     }
}
void Result(void) {
     cout<<"\n Ket qua toi uu:"<<FOPT;
     cout<<"\n Phuong an toi uu:";
     for(int i=1; i<=n; i++)
          cout<<XOPT[i]<<"  ";
}
void Branch_And_Bound( int i) {
     int j, t =(b-weight)/A[i];
     for(j=t; j>=0; j--){   X[i] = j;
          weight = weight+A[i]*X[i];
          cost = cost + C[i]*X[i];
          if (i==n) Update_Kyluc();
          else if ( cost+C[i+1]*(b-weight)/A[i+1]>FOPT)
               Branch_And_Bound(i+1);
          weight = weight-A[i]*X[i];
          cost = cost - C[i]*X[i];
     }
}
void main(void) {
     Init();
     Branch_And_Bound(1);
     Result();
}
```

Ví dụ 2. Bài toán Người du lịch. Một người du lịch muốn đi thăm quan n thành phố T_1, T_2, ..., T_n. Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố 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, . ., n)*, hãy tìm hành trình với tổng chi phí là nhỏ nhất (một hành trình là một cách đi thoả 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:

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

với điều kiện

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

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

$$T_1 \to T(u_2) \to \cdots \to T(u_{k-1}) \to T(u_k)$$

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.

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

Để phát triển hành trình bộ phận này thành hành trình đầy đủ, ta còn phải đi qua *n-k* thành phố còn lại rồi quay trở về thành phố T_1, tức là còn phải đi qua *n-k+1* đoạn đường nữa. Do chi phí phải trả cho việc đi qua mỗi 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, . . ., u_k) có thể được tính theo công thức

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

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.

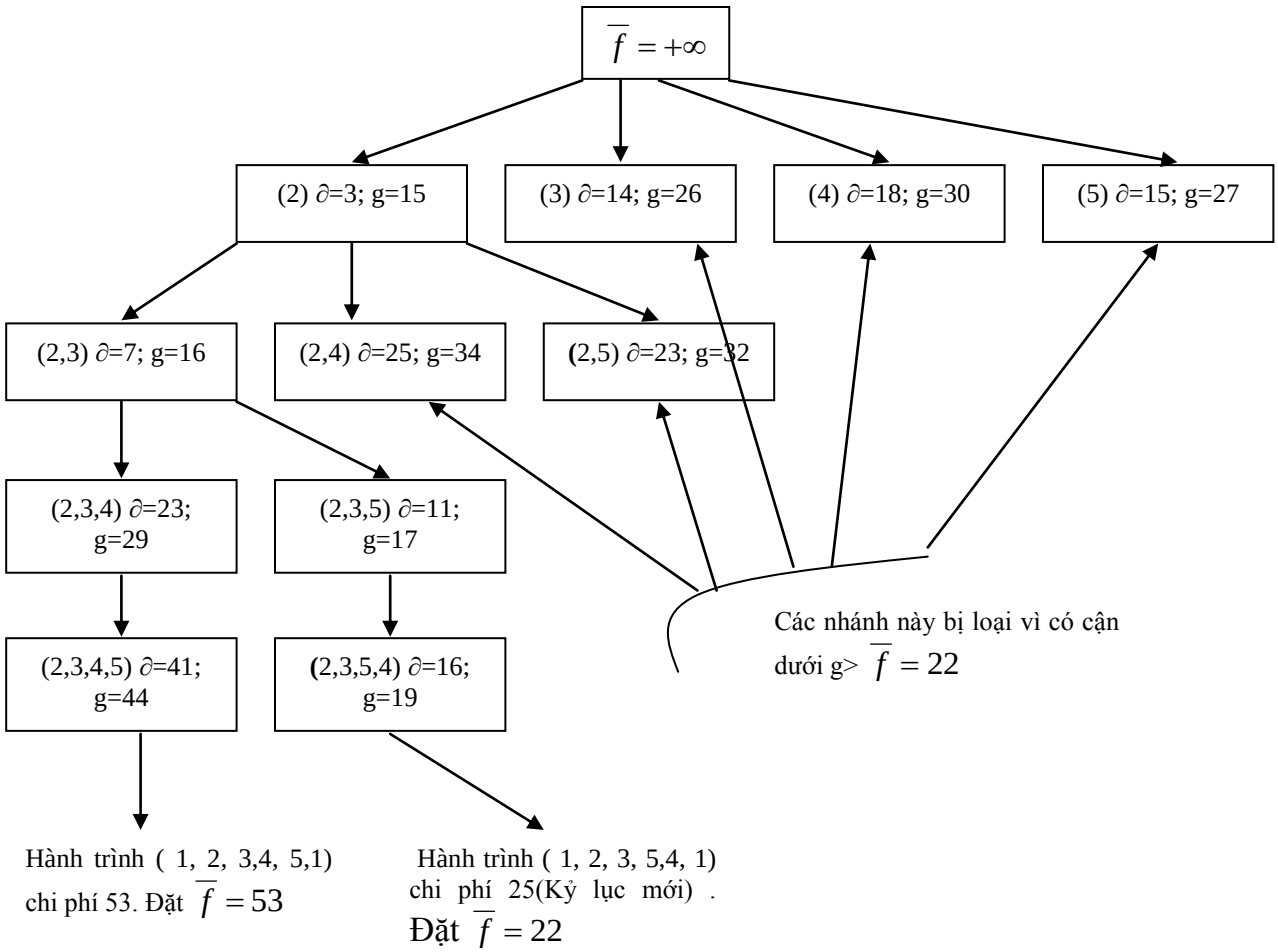
Thông tin về một phương án bộ phận trên cây được ghi trong các ô trên hình vẽ tương ứng theo thứ tự sau:

- 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

$T_1 \to T_2 \to T_3 \to T_5 \to T_4 \to T_1$ và chi phí nhỏ nhất là 22 .



Hình 4.5. 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.

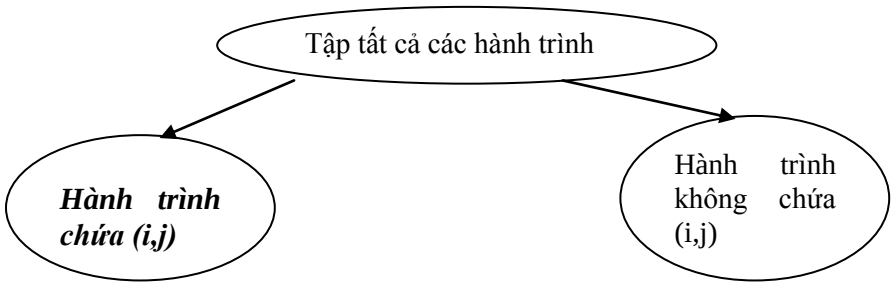
Xét bài toán người du lịch như đã được phát biểu. Gọi $C = \{ c_{ij}: i, j = 1, 2, \ldots n\}$ là ma trận chi phí. Mỗi hành trình của người du lịch

$T_{\pi(1)} \rightarrow T_{\pi(2)} \rightarrow \ldots \rightarrow T_{\pi(n)} \rightarrow T_{\pi(1)}$ có thể viết lại dưới dạng

$(\pi(1), \pi(2), \pi(2), \pi(3), \ldots, \pi(n-1), \pi(n), \pi(n), \pi(1)$, 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 hoạ bởi cây tìm kiếm như trong Hình 4.6.



(Hình 4.6)

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 cuả 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:

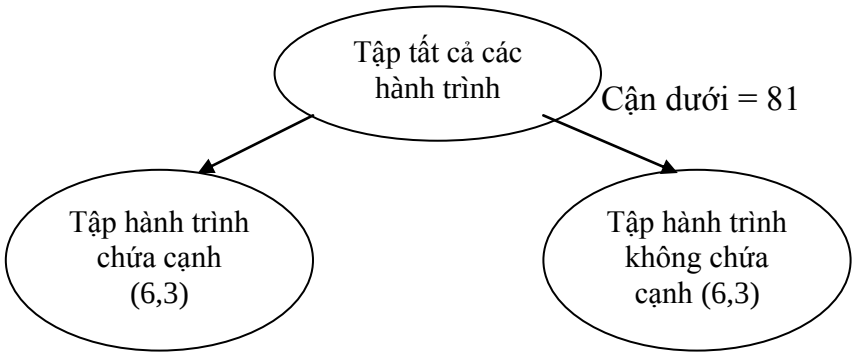
	1	2	3	4	5	6	| r[i]
1	∞	3	93	13	33	9	3
2	4	∞	77	42	21	16	4
3	45	17	∞	36	16	28	16
4	39	90	80	∞	56	7	7
5	28	46	88	33	∞	25	25
6	3	88	18	46	92	∞	3
	0	0	15	8	0	0	

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] = ∞. 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 sang bất cứ 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) = ∞. Cây tìm kiếm lúc này có dạng như trong hình 4.7.



(Hình 4.7)

Cận dưới =81 Cận dưới = 129

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	0
5	3	21	0	∞	0

	1	2	3	4	5	6
1	∞	0	27	2	30	6
2	0	∞	10	30	17	12
3	29	1	∞	12	0	12
4	32	83	10	∞	49	0
5	3	21	0	0	∞	0
6	0	85	∞	35	89	∞

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. Qui 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).

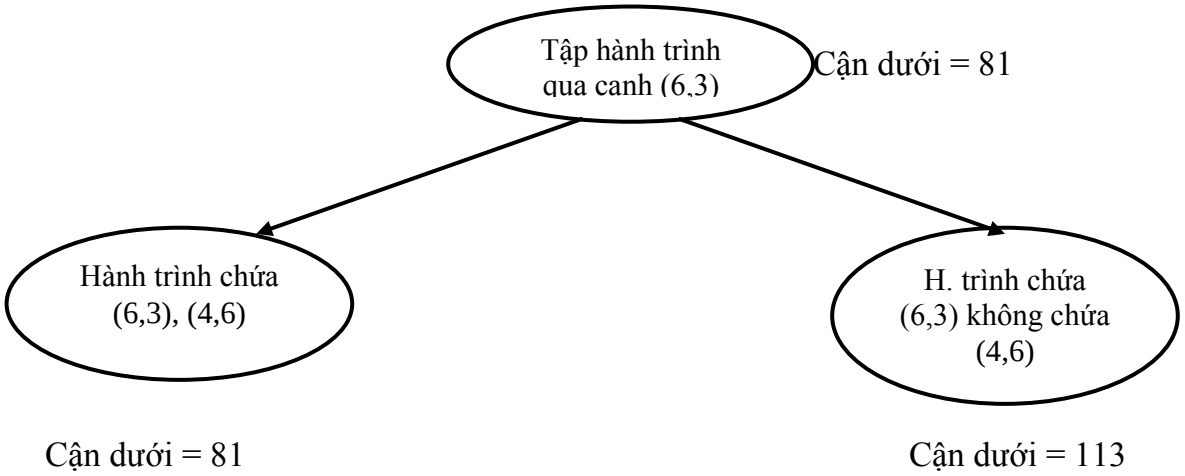
4.4.2.Thủ tục chọn cạnh phân nhánh (r,c)

```
void BestEdge(A, k, r, c, beta)
    Đầu vào: Ma trận rút gọn A kích thước k × k
    Kết quả ra: Cạnh phân nhánh (r,c) và tổng hằng số rút gọn theo dòng r cột
c là beta.
{
    beta = -∞;
    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 hằng 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 4.7.

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 ∞.

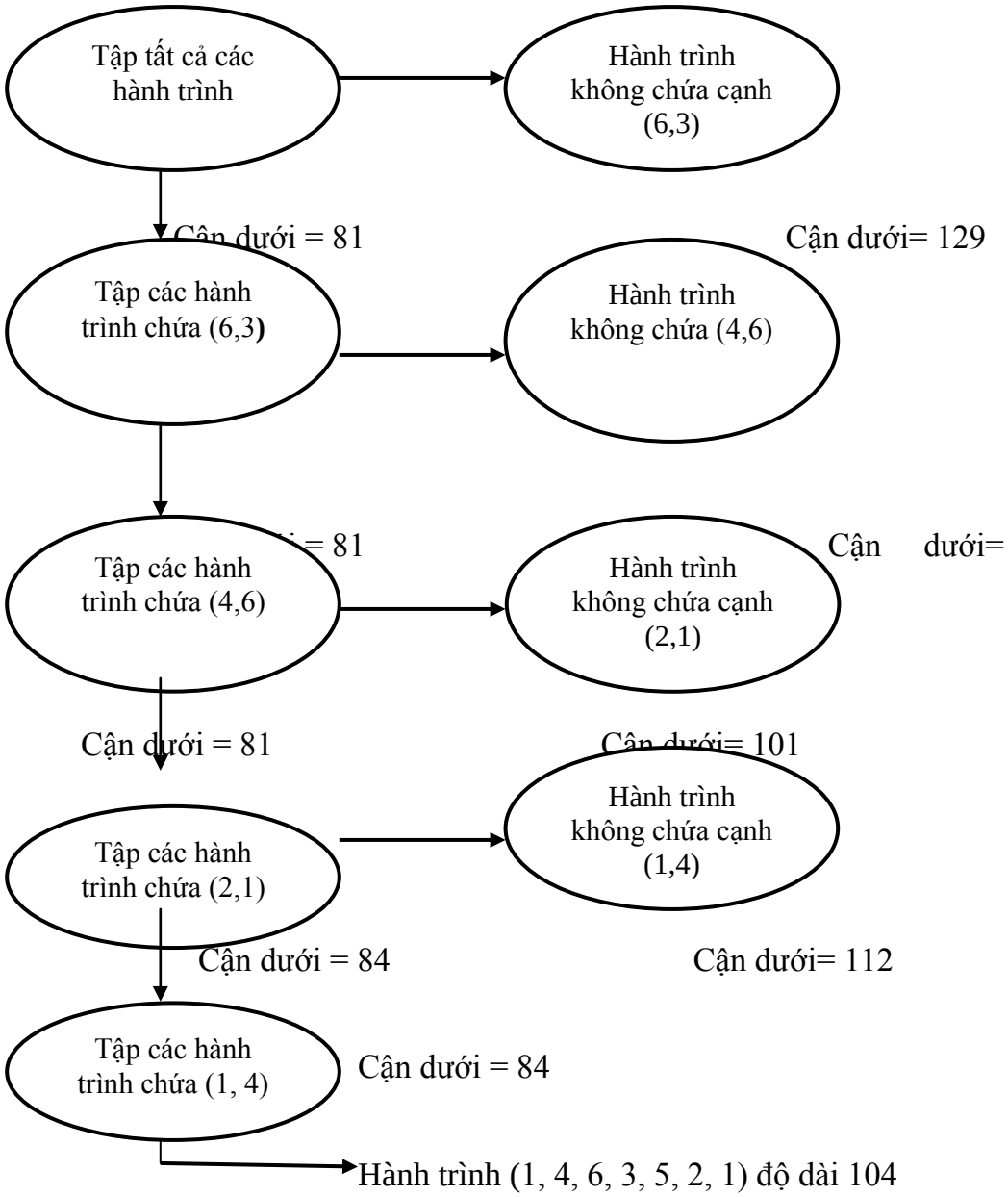


(Hình 4.8)

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 iu là đỉnh cuối của một đường đi (i_1, i_2, . ., i_u) và j_v là đỉnh đầu của đường đi (j_1, j_2, . ., j_k) thì để ngăn ngừa khả năng tạo thành hành trình con ta phải ngăn ngừa khả năng tạo thành hành hành trình con ta phải cấm cạnh (j_k, i_1). Để

tìm i_1 ta đi ngược từ iu, để tìm j_k ta đi xuôi từ j_1 theo danh sách các cạnh đã được kết nạp vào hành trình.



Hình 4.9 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] = ∞, 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 phĩa 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] = ∞ then

<Kết nạp cạnh (u, x), (v, w)>

else

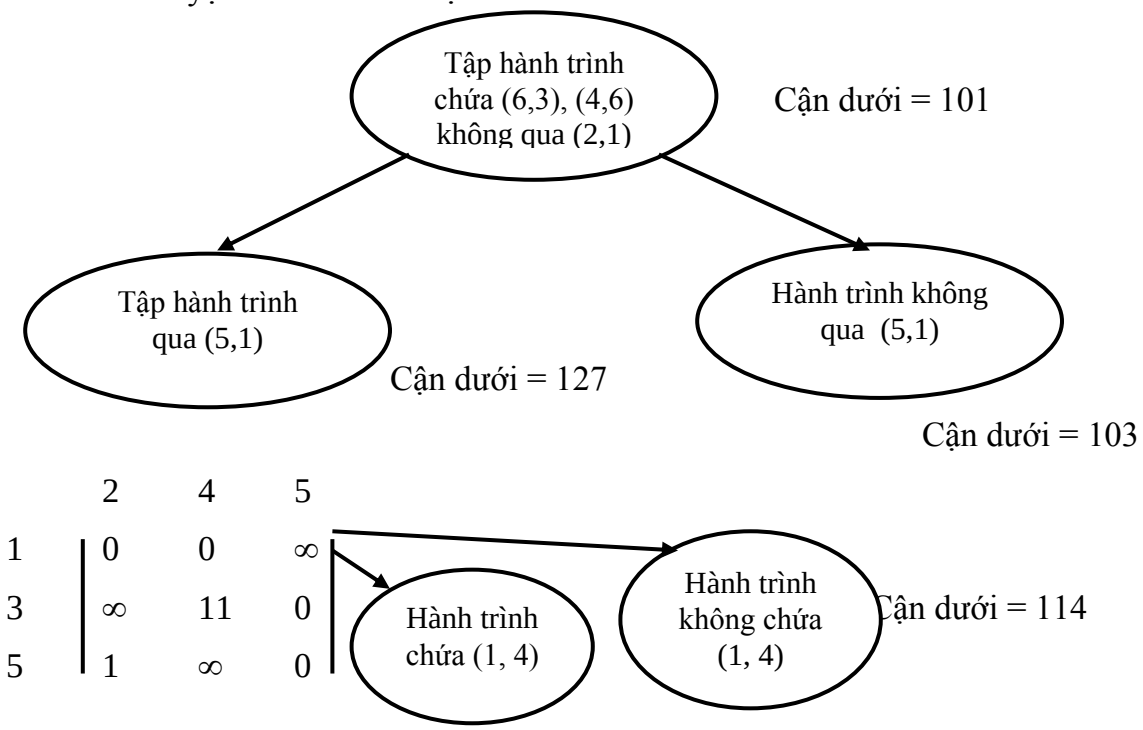
< Kết nạp cạnh (u, w), (v, x) >;

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.



	2	4	5
1	0	0	∞
3	∞	11	0
5	1	∞	0

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), BestEgde(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-eges, 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:

a) $\begin{cases} 5x_1 + x_2 + 9x_3 + 3x_4 \to \max, \\ 4x_1 + 2x_2 + 7x_3 + 3x_4 \le 10, \\ x_j \in \{0,1\}, j = 1,2,3,4. \end{cases}$

b) $\begin{cases} 7x_1 + 3x_2 + 2x_3 + x_4 \to \max, \\ 5x_1 + 3x_2 + 6x_3 + 4x_4 \le 12, \\ x_j \in \{0,1\}, j = 1,2,3,4 \end{cases}$

Bài 2. Giải bài toán cái túi sau:

$$\begin{cases} 30x_1 + 19x_2 + 13x_3 + 38x_4 + 20x_5 + 6x_6 + 8x_7 + 19x_8 + 10x_9 + 11x_{10} \to \max, \\ 15x_1 + 12x_2 + 9x_3 + 27x_4 + 15x_5 + 5x_6 + 8x_7 + 20x_8 + 12x_9 + 15x_{10} \le 62 \\ x_j \in \{0,1\},\ j = 1,2\cdots,10. \end{cases}$$

Bài 3. Giải bài toán người du lịch với ma trận chi phí như sau:

$$\begin{vmatrix} \infty & 31 & 15 & 23 & 10 & 17 \\ 16 & \infty & 24 & 07 & 12 & 12 \\ 34 & 03 & \infty & 25 & 54 & 25 \\ 15 & 20 & 33 & \infty & 50 & 40 \\ 16 & 10 & 32 & 03 & \infty & 23 \\ 18 & 20 & 13 & 28 & 21 & \infty \end{vmatrix}$$

Bài 4. Giải bài toán người du lịch với ma trận chi phí như sau:

$$\begin{vmatrix} \infty & 03 & 93 & 13 & 33 & 09 \\ 04 & \infty & 77 & 42 & 21 & 16 \\ 45 & 17 & \infty & 36 & 16 & 28 \\ 39 & 90 & 80 & \infty & 56 & 07 \\ 28 & 46 & 88 & 33 & \infty & 25 \\ 03 & 88 & 18 & 46 & 92 & \infty \end{vmatrix}$$

CHƯƠNG 5. 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.

5.1. 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 uý, trung uý, thượng uý, đại uý, thiếu tá, trung tá về tham gia duyệt binh ở sư đoàn bộ. Hỏi rằng, có thể xếp 36 sĩ quan này thành một đội ngũ hình vuông sao cho trong mỗi 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	Ed	Ab	Bc
Eb	Ac	Bd	Ce	Da
Be	Ca	Db	Ec	Ad
Dc	Ed	Ae	Ba	Cb

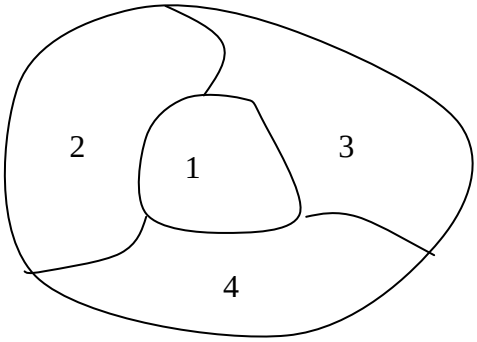
Do lời giải bài toán có thể biểu diễn bởi hai hình vuông với các chữ cái la tinh hoa và la tinh thường nên bài toán tổng quát đặt ra còn được biết với tên gọi *"hình vuông la tinh trực giao"*. Trong hai ví dụ trên ta có hình vuông la tinh trực giao cấp 4 và 5.

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 trực 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 trực giao cho mọi n = 4k + 2 với k > 1.

Tưởng chừng bài toán chỉ mang ý nghĩa thử thách trí tuệ con người thuần tuý 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 5.1. Bản đồ tô bởi ít nhất bốn màu

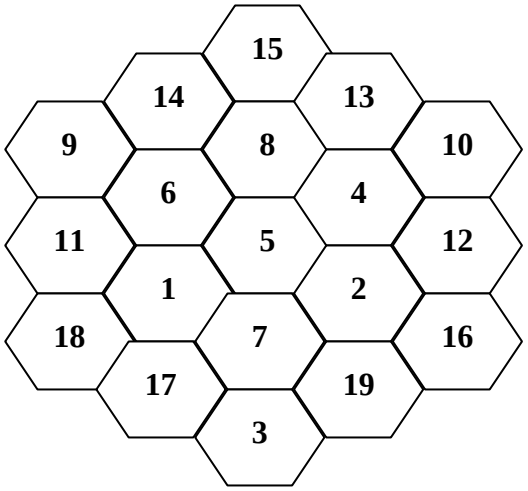
Con số bốn màu không phải là ngẫu nhiên. Người ta đã chứng minh được rằng mọi bản đồ đều được tô bởi số màu lớn hơn 4, còn với số màu ít hơn 4 thì không thể tô được, chẳng hạn bản đồ gồm 4 nước như trên Hình 5.1 không thể tô được với số màu ít hơn 4.

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 để 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 xuốt 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:

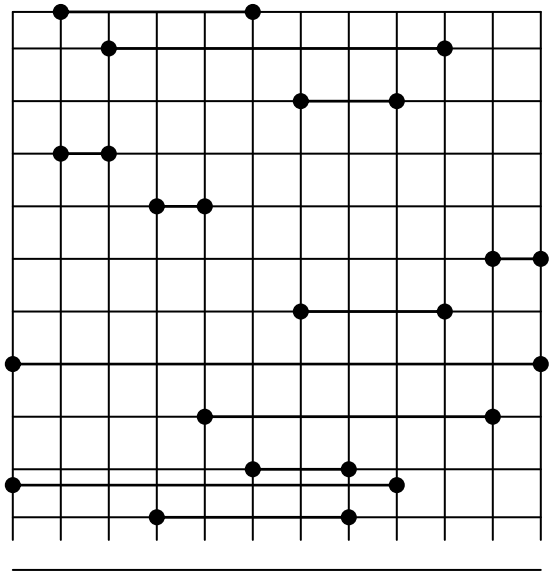
Trên 19 ô lục giác (như Hình 5.2) hãy điền các số từ 1 đến 19 sao cho tổng theo 6 hướng của lục giác là bằng nhau (và đều bằng 38). Sau 47 năm trời kiên nhẫn cuối cùng Adams cũng đã tìm được lời giải. Sau đó vì sơ ý đánh mất bản thảo ông đã tốn thêm 5 năm để khôi phục lại. Năm 1962 Adams đã công bố lời giải đó. Nhưng thật không thể ngờ được đó là lời giải duy nhất.



Hình 5.2. Hình lục giác thần bí

Bài toán 3. Bài toán chọn 2n điểm trên lưới n × n điểm

Cho một lưới gồm n × 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 ≤ 15. Hình 3.3 cho một lời giải với n = 12.



Hình 2.4. Một lời giải với n = 12.

5.2. 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 $a_1, a_2, \ldots, a_7$ 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:

$a_1 + a_2 \le a_3 \quad \Rightarrow a_3 \ge 20$ (vì $a_1, a_2 \ge 10$)

$a_2 + a_3 \le a_4 \quad \Rightarrow a_4 \ge 30$ (vì $a_2 \ge 10, a_3 \ge 20$)

$a_3 + a_4 \le a_5 \quad \Rightarrow a_5 \ge 50$ (vì $a_3 \ge 20, a_4 \ge 30$)

$a_4 + a_5 \le a_6 \quad \Rightarrow a_6 \ge 80$ (vì $a_4 \ge 30, a_5 \ge 50$)

$a_5 + a_6 \le a_7 \quad \Rightarrow a_7 \ge 130$ (vì $a_5 \ge 50, a_6 \ge 80$)

$\Rightarrow$ 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, . . , 9 một cách tuỳ ý. Chứng minh rằng luôn tìm được ba đỉnh liên tiếp có tổng các số là lớn hơn 13.

Giải : Gọi $x_1, x_2, \ldots, x_{10}$ là các số gán cho các đỉnh của thập giác đều. Giả sử ngược lại ta không tìm được 3 đỉnh liên tiếp nào thoả mãn khẳng định trên. Khi đó ta có

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

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

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

$k_4 = x4 + x5 + x6 \le 13$

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

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

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

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

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

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

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

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

$= 135 \Rightarrow$ Mâu thuẫn vì một số bằng 135 không thể hơn 130. Khẳng định chứng minh.

5.3 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 Diriclet 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 thoả mãn $\lceil n/5 \rceil > 5$. Số nguyên bé nhất thoả 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

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

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

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

$15 \le a_j \le 59$

Như vậy, dãy 60 số nguyên dương

$a_1, a_2, \ldots, a_{30}, a_1 + 14, a_2 + 14, \ldots, a_{30} + 14$ 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 $a_1, a_2, \ldots, a_{30}$ là đôi một khác nhau và $a_1 + 14, a_2 + 14, \ldots, a_{30} + 14$ 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.

5.4. 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.

BÀI TẬP

1 . Dùng bảng chân lý để chứng minh luật giao hoán:
   a) $p \vee q \Leftrightarrow q \vee p$
   b) $p \wedge q \Leftrightarrow q \wedge p$
2 . Dùng bảng chân lý để chứng minh luật kết hợp
   a) $(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$
   b) $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$
3 . Dùng bảng chân lý để chứng minh luật phân phối
   a) $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$
   b) $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$
4 . Dùng bảng chân lý để chứng minh luật De Morgan
   a) $\overline{(p \wedge q)} \Leftrightarrow \overline{p} \vee \overline{q}$
   b) $\overline{(p \vee q)} \Leftrightarrow \overline{p} \wedge \overline{q}$
5. Dùng bảng chân lý để chứng minh các mệnh đề kéo theo dưới đây là hằng đúng.
   a) $(p \wedge q) \rightarrow p$
   b) $p \rightarrow (p \vee q)$
   c) $\overline{p} \rightarrow (p \rightarrow q)$
6. Dùng bảng chân lý để chứng minh các mệnh đề kéo theo dưới đây là hằng đúng.
   a) $(p \wedge q) \rightarrow (p \rightarrow q)$
   b) $\overline{(p \rightarrow q)} \rightarrow p$
   c) $\overline{(p \rightarrow q)} \rightarrow \overline{q}$
7 . Dùng bảng chân lý để chứng minh các mệnh đề kéo theo dưới đây là hằng đúng.
   a) $\left[\overline{p} \wedge (p \vee q)\right] \rightarrow q$
   b) $[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$
   c) $[p \wedge (p \rightarrow q)] \rightarrow q$
   d) $[(p \vee q) \wedge (p \rightarrow r) \wedge (q \rightarrow r)] \rightarrow r$
8. Chứng minh các cặp mệnh đề dưới đây là tương đương.

a) $(p \leftrightarrow q) \Leftrightarrow (p \wedge q) \vee (\overline{p} \wedge \overline{q})$

b) $(p \rightarrow q) \Leftrightarrow \overline{q} \rightarrow \overline{p}$

c) $\overline{(p \oplus q)} \Leftrightarrow (p \leftrightarrow q)$

d) $\overline{(p \leftrightarrow q)} \Leftrightarrow (\overline{p} \leftrightarrow q)$

9. Không dùng bảng chân lý chứng minh các mệnh đề kéo theo dưới đây là hằng đúng.

a) $(p \wedge q) \rightarrow p$

b) $p \rightarrow (p \vee q)$

c) $\overline{p} \rightarrow (p \rightarrow q)$

d) $(p \wedge q) \rightarrow (p \rightarrow q)$

e) $\overline{(p \rightarrow q)} \rightarrow p$

f) $\overline{(p \rightarrow q)} \rightarrow \overline{q}$

10. Không dùng bảng chân lý chứng minh các mệnh đề kéo theo dưới đây là hằng đúng.

a) $[\overline{p} \wedge (p \vee q)] \rightarrow q$

b) $[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$

c) $[p \wedge (p \rightarrow q)] \rightarrow q$

d) $[(p \vee q) \wedge (p \rightarrow r) \wedge (q \rightarrow r)] \rightarrow r$

11. Không dùng bảng chân lý, chứng minh các cặp mệnh đề dưới đây là tương đương.

a) $(p \leftrightarrow q) \Leftrightarrow (p \wedge q) \vee (\overline{p} \wedge \overline{q})$

b) $(p \rightarrow q) \Leftrightarrow \overline{q} \rightarrow \overline{p}$

c) $\overline{(p \oplus q)} \Leftrightarrow (p \leftrightarrow q)$

d) $\overline{(p \leftrightarrow q)} \Leftrightarrow (\overline{p} \leftrightarrow q)$

12. Cho A, B, C là các tập hợp. Chứng minh rằng:

a) $(B - A) \cup (C - A) = (B \cup C) - A$

b) $A - B = A \cap \overline{B}$

c) $(A \cap B) \cup (A \cap \overline{B}) = A$

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

e) $(A - B) - C = (A - B) - (B - C)$

12.

a) Trình bày thuật toán sinh hoán vị kế tiếp của 1, 2, .., n ?

b) Cho tập A = { 1, 2, 3, 4, 5, 6, 7, 8, 9}. Sử dụng phương pháp sinh hoán vị theo thứ tự từ điển, tìm 4 hoán vị liền kề tiếp theo của hoán vị 568397421 ?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kê tất cả các hoán vị của 1, 2, .., n ?

13.

a) Trình bày thuật toán sinh hoán vị kế tiếp của 1, 2, .., n ?

b) Cho tập A = { 1, 2, 3, 4, 5, 6, 7, 8, 9}. Sử dụng phương pháp sinh hoán vị theo thứ tự từ điển, tìm 4 hoán vị liền kề tiếp theo của hoán vị 458796321 ?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kêt tất cả các hoán vị của 1, 2, .., n ?

14.

a) Trình bày thuật toán quay lui liệt kê các hoán vị của 1, 2, .., n ?

b) Kiểm nghiệm thuật toán với n=3 ?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kêt tất cả các hoán vị của 1, 2, .., n ?

15 .

a) Trình bày thuật toán sinh tổ hợp chập k của 1, 2,..,n ?

b) Cho tập A = { 1, 2, 3, 4, 5, 6, 7, 8, 9}. Sử dụng phương pháp sinh tổ hợp chập k của một tập hợp theo thứ tự từ điển, hãy tạo 4 tổ hợp chập 4 liền kề tiếp theo của tổ hợp 2,6,8,9?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kêt tất cả các tổ hợp chập k của 1, 2, .., n ?

16.

a) Trình bày thuật toán quay lui liệt kê các tổ hợp chập k của 1, 2,..,n ?

b) Kiểm nghiệm thuật toán với n=5, k =3 ?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kêt tất cả các tổ hợp chập k của 1, 2, .., n ?

17.

a) Trình bày thuật toán sinh xâu nhị phân có độ dài n?

b) Cho xâu nhị phân X = { 1, 0, 1, 1, 1, 1, 1, 1, 1}. Sử dụng phương pháp sinh xâu nhị phân theo thứ tự từ điển, tìm 4 xâu nhị phân liền kề tiếp theo của X?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kêt tất cả các xauu nhị phân có độ dài n?

18.

a) Trình bày thuật toán sinh xâu nhị phân có độ dài n?

b) Cho xâu nhị phân X = { 1, 0, 1, 1, 0, 0, 1, 1, 1}. Sử dụng phương pháp sinh xâu nhị phân theo thứ tự từ điển, tìm 4 xâu nhị phân liền kề tiếp theo của X?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kêt tất cả các xauu nhị phân có độ dài n?

19.

a) Trình bày thuật toán quay lui liệt kê các xâu nhị phân có độ dài n?

b) Kiểm nghiệm thuật toán với n=3 ?

c) Áp dụng thuật toán tại Mục a, viết chương trình liệt kêt tất cả các xâu nhị phân có độ dài n ?

20. Có bao nhiêu biển số xe bắt đầu bằng 2 hoặc 3 chữ cái in hoa và kết thúc là 3 hoặc 4 chữ số, biết rằng có 26 chữ cái trong bảng chữ cái tiếng anh? (VD : RS 0912 là 1 biển số).

21. Có bao nhiêu biển số xe bắt đầu bằng 3 hoặc 4 chữ cái in hoa và kết thúc là 2 hoặc 3 chữ số, biết rằng có 26 chữ cái trong bảng chữ cái tiếng anh? (VD : ABZ 09 là 1 biển số).

22. Có bao nhiêu số nguyên trong khoảng từ 1000 đến 5000 chia hết cho 6 hoặc 9 ?

23. Có bao nhiêu số nguyên trong khoảng từ 5000 đến 9999 chia hết cho 8 hoặc 12 ?

24. Giả sử tất cả các số điện thoại trên thế giới đều theo quy tắc, bắt đầu bằng mã quốc gia dài từ 1 đến 3 chữ số, tức là có dạng X, XX hoặc XXX ; tiếp theo là 10 chữ số dạng NXX-NXX-XXXX trong đó N có thể nhận giá trị từ 1 đến 6, X biểu thị một chữ số từ 0 đến 9. Theo cách đánh số này, sẽ có tối đa bao nhiêu số điện thoại có thể dùng ?

25. Giả sử tất cả các số điện thoại trên thế giới đều theo quy tắc, bắt đầu bằng mã quốc gia dài từ 1 đến 3 chữ số, tức là có dạng X, XX hoặc XXX ; tiếp theo là 10 chữ số dạng NNX-NXX-XXXX trong đó N có thể nhận giá trị từ 5 đến 9, X biểu thị một chữ số từ 0 đến 9. Theo cách đánh số này, sẽ có tối đa bao nhiêu số điện thoại có thể dùng ?

26. Lớp học có 55 bạn nam và 35 bạn nữ. Hãy cho biết có bao nhiêu cách chọn đội văn nghệ của lớp sao cho số bạn nam bằng số bạn nữ, biết rằng đội văn nghệ cần ít nhất 6 thành viên và nhiều nhất 10 thành viên.

27. Lớp học có 60 bạn nam và 42 bạn nữ. Hãy cho biết có bao nhiêu cách chọn đội văn nghệ của lớp sao cho số bạn nam bằng số bạn nữ, biết rằng đội văn nghệ cần ít nhất 4 thành viên và nhiều nhất 8 thành viên.

28. Lớp học có 50 bạn nam và 20 bạn nữ. Hãy cho biết có bao nhiêu cách chọn đội văn nghệ của lớp sao cho số bạn nam đúng bằng 2 lần số bạn nữ, biết rằng đội văn nghệ cần ít nhất 6 thành viên và nhiều nhất 12 thành viên.

29. : Lớp học có 60 bạn nam và 25 bạn nữ. Hãy cho biết có bao nhiêu cách chọn đội văn nghệ của lớp sao cho số bạn nam đúng bằng 2 lần số bạn nữ, biết rằng đội văn nghệ cần ít nhất 3 thành viên và nhiều nhất 9 thành viên.

30. Trong kỳ thi tuyển sinh đại học khối A, các thí sinh thi trắc nghiệm môn Lý và Hóa, mỗi môn thi có 50 câu hỏi. Mỗi câu hỏi có đúng 4 phương án trả lời và chỉ được lựa chọn tối đa 1 phương án. Mỗi câu trả lời đúng được 0.2 điểm, câu trả lời sai hoặc không trả lời thì không được điểm.

a) Hãy cho biết có bao nhiêu cách điền phiếu trắc nghiệm môn Lý.
b) Cần có ít nhất bao nhiêu thí sinh tham gia để có ít nhất 10 sinh viên có tổng điểm Lý và Hóa bằng nhau. Biết rằng điểm thi không được làm tròn.

31. Trong kỳ thi tuyển sinh đại học khối A, các thí sinh thi trắc nghiệm môn Lý và Hóa, mỗi môn thi có 40 câu hỏi. Mỗi câu hỏi có đúng 5 phương án trả lời và chỉ được lựa chọn tối đa 1 phương án. Mỗi câu trả lời đúng được 0.25 điểm, câu trả lời sai hoặc không trả lời thì không được điểm.

a) Hãy cho biết có bao nhiêu cách điền phiếu trắc nghiệm môn Hóa.
b) Cần có ít nhất bao nhiêu thí sinh tham gia để có ít nhất 10 sinh viên có tổng điểm Lý và Hóa bằng nhau, biết rằng điểm thi không được làm tròn.

32. Một bài thi trắc nghiệm có 30 câu hỏi, mỗi câu hỏi có 5 phương án trả lời và chỉ có 1 phương án đúng. Mỗi câu trả lời đúng được 3 điểm, trả lời sai bị trừ 1 điểm, nếu không trả lời thì câu đó nhận 0 điểm. Biết rằng tổng điểm thấp nhất là 0. Hãy cho biết:

a) Có bao nhiêu cách điền phiếu trắc nghiệm (mỗi câu chỉ được chọn tối đa 1 phương án).
b) Cần bao nhiêu sinh viên tham gia thi để đảm bảo có ít nhất 2 sinh viên có cùng kết quả thi.

33. Một bài thi trắc nghiệm có 35 câu hỏi, mỗi câu hỏi có 4 phương án trả lời và chỉ có 1 phương án đúng. Mỗi câu trả lời đúng được 3 điểm, trả lời sai bị trừ 1 điểm, nếu không trả lời thì câu đó nhận 0 điểm. Biết rằng tổng điểm thấp nhất là 0. Hãy cho biết:

a) Có bao nhiêu cách điền phiếu trắc nghiệm (mỗi câu chỉ được chọn tối đa 1 phương án).
b) Cần bao nhiêu sinh viên tham gia thi để đảm bảo có ít nhất 2 sinh viên có cùng kết quả thi.

34. Phương trình $x_1 + x_2 + x_3 = 13$ có bao nhiêu nghiệm nguyên không âm thỏa mãn

a) $x_1 \geq 1, x_2 \geq 3, x_3 \geq 0$

b) $x_1 \geq 0, x_2 \geq 3, x_3 \leq 5$

35. Phương trình $x_1 + x_2 + x_3 = 15$ có bao nhiêu nghiệm nguyên không âm thỏa mãn

a) $x_1 \geq 2, x_2 \geq 0, x_3 \geq 4$

b) $x_1 \geq 1, x_2 \geq 0, x_3 \leq 7$

36. Phương trình $x_1 + x_2 + x_3 = 14$ có bao nhiêu nghiệm nguyên không âm thỏa mãn

a) $x_1 \geq 0, x_2 \geq 3, x_3 \geq 1$

b) $x_1 \geq 0, x_2 \leq 6, x_3 \geq 3,$

37. Phương trình $x_1 + x_2 + x_3 = 16$ có bao nhiêu nghiệm nguyên không âm thỏa mãn

a) $x_1 \geq 2, x_2 \geq 0, x_3 \geq 2$

b) $x_1 \leq 6, x_2 \geq 3, x_3 \geq 0$

38.

a) Giải hệ thức truy hồi sau

$a_0 = 2, a_1 = 6, a_n = 3a_{n-1} - 2a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n chứa 3 số 0 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 7.

39.

a) Giải hệ thức truy hồi sau

$a_0 = 4, a_1 = 8, a_n = a_{n-1} + 2a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n chứa 3 số 1 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 6.

40.

a) Giải hệ thức truy hồi sau

$a_0 = 1, a_1 = 5, a_n = -a_{n-1} + 6a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, bắt đầu bằng số 1 và có chứa 2 số 1 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 7.

41.

a) Giải hệ thức truy hồi sau

$a_0 = 6, a_1 = 7, a_n = a_{n-1} + 6a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, kết thúc bằng số 1 và có chứa 2 số 1 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 6.

42.

a) Giải hệ thức truy hồi sau

$a_0 = 5, a_1 = 4, a_n = a_{n-1} + 2a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, bắt đầu bằng số 0 và có chứa 2 số 1 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 7.

43.

a) Giải hệ thức truy hồi sau

$a_0 = 8, a_1 = 3, a_n = -a_{n-1} + 2a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, kết thúc bằng số 0 và có chứa 2 số 1 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 6.

44.

a) Giải hệ thức truy hồi sau

$a_0 = 5, a_1 = 2, a_n = -3a_{n-1} + 4a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, bắt đầu bằng số 1 và có chứa 2 số 0 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 7.

45.

a) Giải hệ thức truy hồi sau

$a_0 = 6, a_1 = 9, a_n = 3a_{n-1} + 4a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, kết thúc bằng số 1 và có chứa 2 số 0 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 6.

46.

a) Giải hệ thức truy hồi sau

$a_0 = 6, a_1 = 9, a_n = 7a_{n-1} - 12a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, bắt đầu bằng số 0 và có chứa 2 số 0 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 7.

47.

a) Giải hệ thức truy hồi sau

$a_0 = 8, a_1 = 7, a_n = -a_{n-1} + 12a_{n-2}$ với $n \geq 2$

b) Tìm hệ thức truy hồi để tính số các xâu nhị phân độ dài n, kết thúc bằng số 0 và có chứa 2 số 0 liên tiếp.

c) Tính số xâu nhị phân thỏa mãn điều kiện ở câu b với n = 6.

48. Hãy tìm nghiệm của công thức truy hồi với điều kiện đầu dưới đây:

a) $a_n = 3a_{n-1}$ với $a_0 = 2$.

b) $a_n = -4a_{n-1} - 4a_{n-2}$ với $n \geq 2$ và $a_0 = 0$ và $a_1 = 1$.

c) $a_n = 14a_{n-1} - 49a_{n-2}$ với $n \geq 2$ và $a_0 = 3$ và $a_1 = 35$.

49. Hãy tìm nghiệm của công thức truy hồi với điều kiện đầu dưới đây:

a) $a_n = a_{n-1} + 2$ với $a_0 = 3$.

b) $a_n = -4a_{n-1} - 4a_{n-2}$ với $n \geq 2$ và $a_0 = 0$ và $a_1 = 1$.

c) $an = 13a_{n-1} - 22a_{n-2}$ với $n \geq 2$ và $a_0 = 3$ và $a_1 = 15$.

50. Hãy tìm nghiệm của công thức truy hồi với điều kiện đầu dưới đây:

a) $a_n = a_{n-1} + 2n + 3$ với $a_0 = 4$.

b) $a_n = -6a_{n-1} - 9a_{n-2}$ với $n \geq 2$ và $a_0 = 3$ và $a_1 = -3$.

c) $a_n = 2a_{n-1} + 5a_{n-2} - 6a_{n-3}$ với $n \geq 3$ và $a_0 = 7$ và $a_1 = -4$, $a_2 = 8$.

51. Hãy tìm nghiệm của công thức truy hồi với điều kiện đầu dưới đây:

a) $a_n = a_{n-1} + 2^n$ với $a_0 = 1$.

b) $a_n = 14a_{n-1} - 49a_{n-2}$ với $n \geq 2$ và $a_0 = 3$ và $a_1 = 35$.

c) $a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3}$ với $n \geq 3$ và $a_0 = 3$ và $a_1 = 6$, $a_2 = 0$.

52. Hãy tìm nghiệm của công thức truy hồi với điều kiện đầu dưới đây:

a) $a_n = a_{n-1} + 2^n$ với $a_0 = 1$.

b) $a_n = -13a_{n-1} - 22a_{n-2}$ với $n \geq 2$ và $a_0 = 3$ và $a_1 = 15$.

c) $a_n = 2a_{n-1} + 5a_{n-2} - 6a_{n-3}$ với $n \geq 3$ và $a_0 = 7$ và $a_1 = -4$, $a_2 = 8$.

53. Hãy tìm nghiệm của công thức truy hồi với điều kiện đầu dưới đây:

a) $a_n = -4a_{n-1} - 4a_{n-2}$ với $n \geq 2$ và $a_0 = 0$ và $a_1 = 1$.

b) $a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3}$ với $n \geq 3$ và $a_0 = 3$ và $a_1 = 6$, $a_2 = 0$.

c) $a_n = 7a_{n-2} + 6a_{n-3}$ với $n \geq 3$ và $a_0 = 9$ và $a_1 = 10$, $a_2 = 32$.

54. Phương trình

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 24$$

có bao nhiêu nghiệm nguyên không âm sao cho

a) $x_i \geq 2$ *với* $i = 1, 2, 3, 4, 5, 6$*?*

b) $1 \leq x_1 \leq 5$ *và* $x_3 \geq 8$*?*

c) $1 \leq x_1 \leq 5$ *và* $3 \leq x_2 \leq 7$*?*

d) $1 \leq x_1 \leq 5$ *và* $3 \leq x_2 \leq 7$ *và* $x_3 \geq 8$*?*

55. Hãy tìm tất cả các số tự nhiên có 7 chữ số thỏa mãn:

a) Số có 7 chữ số tạo thành một số thuận nghịch;

b) Số có 7 chữ số tạo thành một số thuận nghịch và có tất cả các chữ số đều khác 0;

c) Số có 7 chữ số có tổng các chữ số là 18;

56. Hãy tìm tất cả các số tự nhiên có 9 chữ số thỏa mãn:

a) Số có 9 chữ số tạo thành một số thuận nghịch;

b) Số có 9 chữ số tạo thành một số thuận nghịch và có tất cả các chữ số đều khác 0;

c) Số có 7 chữ số có tổng các chữ số là 19;

57. Hãy tìm tất cả các số tự nhiên có 10 chữ số thỏa mãn:

a) Số có 10 chữ số tạo thành một số thuận nghịch;

b) Số có 10 chữ số tạo thành một số thuận nghịch và có tất cả các chữ số đều khác 0;

c) Số có 10 chữ số có tổng các chữ số là 18.

58.

a) Tìm hệ thức truy hồi và cho điều kiện đầu để tính số các xâu nhị phân độ dài n và không có k số 1 liên tiếp?

b) Tìm hệ thức truy hồi và cho điều kiện đầu để tính số các xâu nhị phân độ dài n có ít nhất một dãy k số *1* liên tiếp?

59.

a) Tìm hệ thức truy hồi và cho điều kiện đầu để tính số các xâu nhị phân độ dài n và không có k số 0 liên tiếp?

b) Tìm hệ thức truy hồi và cho điều kiện đầu để tính số các xâu nhị phân độ dài n có ít nhất một dãy k số 0 liên tiếp?

60.

a) Một hệ thống máy tính coi một xâu các chữ số hệ thập phân là một từ mã hợp lệ nếu nó chứa một số chẵn chữ số 1. Ví dụ 1231407869 là hợp lệ, 120987045608 là không hợp lệ. Giả sử a_n là số các từ mã độ dài n. Hãy tìm hệ thức truy hồi và điều kiện đầu cho a_n?

b) Giải hệ thức truy hồi $a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3}$ với $n \geq 3$ và $a_0 = 3$ và $a_1 = 6$, $a_2 = 0$.

61. Phương trình $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 25$ có bao nhiêu nghiệm nguyên không âm thỏa mãn

a) $x_1 \geq 1$, $x_2 \geq 2$, $x_3 \geq 3$, $x_4 \geq 4$, $x_5 \geq 5$, $x_6 \geq 6$?

b) $2 \leq x_1 \leq 7$, $4 \leq x_2 \leq 8$; $x_3 \geq 5$?

62.

a) Một hệ thống máy tính coi một xâu các chữ số hệ thập phân là một từ mã hợp lệ nếu nó chứa một số lẻ chữ số 0. Ví dụ 1231407869 là hợp lệ, 12098704568 là không hợp lệ. Giả sử a_n là số các từ mã độ dài n. Hãy tìm hệ thức truy hồi và điều kiện đầu cho a_n?

b) Giải hệ thức truy hồi $a_n = 7a_{n-2} + 6a_{n-3}$ với $n \geq 3$ và $a_0 = 9$ và $a_1 = 10$, $a_2 = 32$.

63. Phương trình $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 28$ có bao nhiêu nghiệm nguyên không âm thỏa mãn

c) $x_1 \geq 1$, $x_2 \geq 2$, $x_3 \geq 3$, $x_4 \geq 4$, $x_5 \geq 5$, $x_6 \geq 6$?

b) $1 \leq x_1 \leq 6$, $4 \leq x_2 \leq 9$; $x_3 \geq 4$?

64.

a) Trình bày thuật toán nhánh cận giải bài toán cái túi?

b) Áp dụng thuật toán nhánh cận giải bài toán cái túi dưới đây, chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

$$\begin{cases} 5x_1 + x_2 + 9x_3 + 3x_4 \to \max, \\ 4x_1 + 2x_2 + 7x_3 + 3x_4 \leq 10, \\ x_j \in \{0,1\}, j = 1,2,3,4. \end{cases}$$

65.

a) Trình bày thuật toán nhánh cận giải bài toán cái túi?

b) Áp dụng thuật toán nhánh cận giải bài toán cái túi dưới đây, chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

$$\begin{cases} 7x_1 + 3x_2 + 2x_3 + x_4 \to \max, \\ 5x_1 + 3x_2 + 6x_3 + 4x_4 \le 12, \\ x_j \in \{0,1\},\ j = 1,2,3,4 \end{cases}$$

66.

a) Trình bày thuật toán nhánh cận giải bài toán cái túi?

b) Áp dụng thuật toán nhánh cận giải bài toán cái túi dưới đây, chỉ rõ kết quả theo mỗi bước thực hiện của thuật toán?

$$\begin{cases} 30x_1 + 19x_2 + 13x_3 + 38x_4 + 20x_5 + 6x_6 + 8x_7 + 19x_8 + 10x_9 + 11x_{10} \to \max, \\ 15x_1 + 12x_2 + 9x_3 + 27x_4 + 15x_5 + 5x_6 + 8x_7 + 20x_8 + 12x_9 + 15x_{10} \le 62 \\ x_j \in \{0,1\},\ j = 1,2\cdots,10. \end{cases}$$

67. Giải bài toán người du lịch với ma trận chi phí như sau:

$$\begin{vmatrix} \infty & 31 & 15 & 23 & 10 & 17 \\ 16 & \infty & 24 & 07 & 12 & 12 \\ 34 & 03 & \infty & 25 & 54 & 25 \\ 15 & 20 & 33 & \infty & 50 & 40 \\ 16 & 10 & 32 & 03 & \infty & 23 \\ 18 & 20 & 13 & 28 & 21 & \infty \end{vmatrix}$$

68. Giải bài toán người du lịch với ma trận chi phí như sau:

$$\begin{vmatrix} \infty & 03 & 93 & 13 & 33 & 09 \\ 04 & \infty & 77 & 42 & 21 & 16 \\ 45 & 17 & \infty & 36 & 16 & 28 \\ 39 & 90 & 80 & \infty & 56 & 07 \\ 28 & 46 & 88 & 33 & \infty & 25 \\ 03 & 88 & 18 & 46 & 92 & \infty \end{vmatrix}$$