# CHƯƠNG 3

# DUYỆT VÀ ĐỆ QUI

## 3.1- Định nghĩa bằng đệ qui

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

**Ví dụ 3.1.** Để tính tổng S(n) = 1 + 2 + . . .+ n, chúng ta có thể thực hiện thông qua hai bước như sau:

**Bước phân tích:**

Để tính toán được S(n) trước tiên ta phải tính toán trước S(n-1) sau đó tính S(n) = S(n-1) +n.

Để tính toán được S(n-1), ta phải tính toán trước S(n-2) sau đó tính S(n-1) = S(n-2) + n-1.

. . . . . .

Để tính toán được S(2), ta phải tính toán trước S(1) sau đó tính S(2) = S(1) + 2.

Và cuối cùng S(1) chúng ta có ngay kết quả là 1

**Bước thay thế ngược lại:**

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

S(1) = 1

S(2) = S(1) + 2

S(3) = S(2) + 3

. . . . . .

S(n) = S(n - 1) + n

**Ví dụ 3.2. Định nghĩa hàm bằng đệ qui**

**Hàm f(n) = n!**

Dễ thấy f(0) = 1.

Vì (n+1) ! = 1 . 2.3 . . . n(n+1) = n! (n+1), nên ta có:

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

Hàm $f(n) = a^n$

Vì $a^0 = 1$; $f(n+1) = a^{n+1} = a.a^n = a.$ f(n) nên f (n) = a. f(n) với mọi số thực a và số tự nhiên n.

**Ví dụ 3.3. Tập hợp định nghĩa bằng đệ qui**

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

*, trong đó là xâu rỗng

wx * nếu w * và x *

**Ví dụ 3.4. Cấu trúc tự trỏ được định nghĩa bằng đệ qui**

```
struct node {
        int infor;
        struct node *left;
        struct node *right;
};
```

**3.2- Giải thuật đệ qui**

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

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

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

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

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

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

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

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

**Thuật toán 4: Thuật toán đệ qui tìm kiếm nhị phân:**

```
int binary_search( float *A, int x, int i, int j){
        int  mid = ( i + j)/2;
        if (  x > A[mid] && i<mid)
                return(binary_search(A, x, mid +1,j);
        else if (x<A[mid] && j > mid)
                return(binary_search(A, x, i,mid-1);
        else if (x == A[mid])
                return(mid);
        return(-1);
}
```

**Thuật toán 5: Thuật toán đệ qui tính số fibonacci thứ n**

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

### 3.3- Thuật toán sinh kế tiếp

Phương pháp sinh kế tiếp dùng để giải quyết bài toán liệt kê của lý thuyết tổ hợp. Thuật toán sinh kế tiếp chỉ được thực hiện trên lớp các bài toán thỏa mãn hai điều kiện sau:

Có thể xác định được một thứ tự trên tập các cấu hình tổ hợp cần liệt kê, từ đó xác định được cấu hình đầu tiên và cấu hình cuối cùng.

Từ một cấu hình bất kỳ chưa phải là cuối cùng, đều có thể xây dựng được một thuật toán để suy ra cấu hình kế tiếp.

Tổng quát, thuật toán sinh kế tiếp có thể được mô tả bằng thủ tục genarate, trong đó Sinh_Kế_Tiếp là thủ tục sinh cấu hình kế tiếp theo thuật toán sinh đã được xây dựng. Nếu cấu hình hiện tại là cấu hình cuối cùng thì thủ tục Sinh_Kế_Tiếp sẽ gán cho stop giá trị true, ngược lại cấu hình kế tiếp sẽ được sinh ra.

```
Procedure    generate;
    begin
        <Xây dựng cấu hình ban đầu>
        stop :=false;
        while not stop do
        begin
            <Đưa ra cấu hình đang có >;
            Sinh_Kế_Tiếp;
        end;
    end;
```

Sau đây chúng ta xét một số ví dụ minh họa cho thuật toán sinh.

#### 3.3.1- Bài toán liệt kê các tập con của tập n phần tử

Một tập hợp hữu hạn gồm n phần tử đều có thể biểu diễn tương đương với tập các số tự nhiên 1, 2, . . . n. Bài toán được đặt ra là: Cho một tập hợp gồm n phần tử X = { $X_1$, $X_2$, . ., $X_n$ }, hãy liệt kê tất cả các tập con của tập hợp X.

Để liệt kê được tất cả các tập con của X, ta làm tương ứng mỗi tập Y  X với một xâu nhị phân có độ dài n là B = { $B_1$, $B_2$, . . , $B_n$ } sao cho $B_i$ = 0 nếu $X_i$  Y và $B_i$ = 1 nếu $X_i$  Y; như vậy, phép liệt kê tất cả các tập con của một tập hợp n phần tử tương đương với phép liệt kê tất cả các xâu nhị phân có độ dài n. Số các xâu nhị phân có độ dài n là $2^n$. Bây giờ ta đi xác định thứ tự các xâu nhị phân và phương pháp sinh kế tiếp.

Nếu xem các xâu nhị phân b = { $b_1$, $b_2$, . . , $b_n$ } như là biểu diễn của một số nguyên dương p(b). Khi đó thứ tự hiển nhiên nhất là thứ tự tự nhiên được xác định như sau:

Ta nói xâu nhị phân b = { $b_1$, $b_2$, . . , $b_n$ } có thứ tự trước xâu nhị phân b' = { $b'_1$, $b'_2$, . . , $b'_n$ } và kí hiệu là b<b' nếu p(b) < p(b'). Ví dụ với n= 4: chúng ta có $2^4 = 16$ xâu nhị phân (tương ứng với 16 tập con của tập gồm n phần tử) được liệt kê theo thứ tự từ điển như sau:

| b | p(b) |
|---|---|
| 0 0 0 0 | 0 |
| 0 0 0 1 | 1 |
| 0 0 1 0 | 2 |
| 0 0 1 1 | 3 |
| 0 1 0 0 | 4 |
| 0 1 0 1 | 5 |
| 0 1 1 0 | 6 |
| 0 1 1 1 | 7 |
| 1 0 0 0 | 8 |
| 1 0 0 1 | 9 |
| 1 0 1 0 | 10 |
| 1 0 1 1 | 11 |
| 1 1 0 0 | 12 |
| 1 1 0 1 | 13 |
| 1 1 1 0 | 14 |
| 1 1 1 1 | 15 |

Từ đây ta xác định được xâu nhị phân đầu tiên là 00. .00 và xâu nhị phân cuối cùng là 11..11. Quá trình liệt kê dừng khi ta được xâu nhị phân 1111. Xâu nhị phân kế tiếp là biểu diễn nhị phân của giá trị xâu nhị phân trước đó cộng thêm 1 đơn vị. Từ đó ta nhận được qui tắc sinh kế tiếp như sau:

Tìm chỉ số i đầu tiên theo thứ tự i = n, n-1, . ., 1 sao cho $b_i = 0$.

Gán lại $b_i = 1$ và $b_j = 0$ với tất cả j>i. Dãy nhị phân thu được là dãy cần tìm

**Thuật toán sinh xâu nhị phân kế tiếp**

```
void Next_Bit_String( int *B, int n ){
```

```
        i = n;
        while (b_i ==1 ) {
                b_i = 0
                i = i-1;
        }
        b_i = 1;
}
```

Sau đây là văn bản chương trình liệt kê các xâu nhị phân có độ dài n:

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#include <conio.h>
#define MAX 100
#define TRUE        1
#define FALSE       0
int       Stop, count;
void Init(int *B, int n){
        int i;
        for(i=1; i<=n ;i++)
                B[i]=0;
        count =0;
}
void Result(int *B, int n){
        int i;count++;
        printf("\n Xau nhi phan thu %d:",count);
        for(i=1; i<=n;i++)
                        printf("%3d", B[i]);

}
void Next_Bits_String(int *B, int n){
```

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

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

### 3.3.2- Bài toán liệt kê tập con m phần tử của tập n phần tử

**Bài toán:** Cho tập hợp X = { 1, 2, . ., n }. Hãy liệt kê tất cả các tập con m < n phần tử của X.

Mỗi tập con m phần tử của X có thể biểu diễn như một bộ có thứ tự

a = ($a_1$, $a_2$, .., $a_m$) thoả mãn  1 <= $a_1$< $a_2$ < ..<$a_m$<=n

Trên các tập con m phần tử của X, ta định nghĩa thứ tự của các tập con như sau:

Ta nói tập con $a = (a_1, a_2, .., a_m)$ có thứ tự trước tập $a' = (a'_1, a'_2, .., a'_m)$ theo thứ tự từ điển và kí hiệu $a < a'$ nếu tìm được chỉ số k sao cho:

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

Ví dụ X = { 1, 2, 3, 4, 5 }, n = 5, m = 3. Các tập con 3 phần tử của X được liệt kê theo thứ tự từ điển như sau:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 1 | 2 | 4 |
| 1 | 2 | 5 |
| 1 | 3 | 4 |
| 1 | 3 | 5 |
| 1 | 4 | 5 |
| 2 | 3 | 4 |
| 2 | 3 | 5 |
| 2 | 4 | 3 |
| 3 | 4 | 5 |

Như vậy, tập con đầu tiên là 1 , 2, . ., m. Tập con cuối cùng là n-m+1, n-m+2, . ., n. Giả sử ta có tập con chưa phải là cuối cùng (nhỏ hơn so với tập con n-m+1, n-m+2, . ., n), khi đó tập con kế tiếp của a được sinh bởi các qui tắc biến đổi sau:

Tìm từ bên phải dãy $a_1, a_2, .., a_m$ phần tử $a_i$ n-m+i,

Thay thế $a_i = a_i + 1$;

Thay $a_j$ bởi $a_i + j - i$, với j = i +1, i + 2, . ., m.

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

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

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

$a_j = a_i + j - i;$

}

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

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

```
            for(j=i+1; j<=k; j++)
                C[j]=C[i]+j-i;
        }
        else Stop = TRUE;
}
void Combination(void){
        Stop=FALSE;
        while (!Stop){
            Result();
            Next_Combination();
        }
}
void main(void){
        clrscr(); Init();Combination();getch();
}
```

### 3.3.3- Bài toán liệt kê các hoán vị của tập n phần tử

Bài toán: Cho tập hợp X = { 1, 2, . ., n }. Hãy liệt kê tất cả các hoán vị của X.

Mỗi hoán vị n phần tử của tập hợp X có thể biểu diễn bởi bộ có thứ tự gồm n thành phần a = ($a_1$, $a_2$, .., $a_n$) thoả mãn:

$a_i$ X; i = 1, 2, . ., n; $a_p$ $a_q$ nếu p q.

Trên tập có thứ tự các hoán vị n phần tử của X, ta định nghĩa thứ tự của các hoán vị đó như sau:

Hoán vị a = ($a_1$, $a_2$, .., $a_n$) được gọi là có thứ tự trước hoán vị a' = ($a'_1$, $a'_2$, .., $a'_n$) và kí hiệu a <a' nếu tìm được chỉ số k sao cho:

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

Ví dụ X= { 1, 2, 3, 4} khi đó các hoán vị của 4 phần tử được sắp xếp theo thứ tự từ điển như sau:

1 2 3 4

| | | | |
|---|---|---|---|
| 1 | 2 | 4 | 3 |
| 1 | 3 | 2 | 4 |
| 1 | 3 | 4 | 2 |
| 1 | 4 | 2 | 3 |
| 1 | 4 | 3 | 2 |
| 2 | 1 | 3 | 4 |
| 2 | 1 | 4 | 3 |
| 2 | 3 | 1 | 4 |
| 2 | 3 | 4 | 1 |
| 2 | 4 | 1 | 3 |
| 2 | 4 | 3 | 1 |
| 3 | 1 | 2 | 4 |
| 3 | 1 | 4 | 2 |
| 3 | 2 | 1 | 4 |
| 3 | 2 | 4 | 1 |
| 3 | 4 | 1 | 2 |
| 3 | 4 | 2 | 1 |
| 4 | 1 | 2 | 3 |
| 4 | 1 | 3 | 2 |
| 4 | 2 | 1 | 3 |
| 4 | 2 | 3 | 1 |
| 4 | 3 | 1 | 2 |
| 4 | 3 | 2 | 1 |

Như vậy, hoán vị đầu tiên là 1, 2, . .,n, hoán vị cuối cùng là n, n-1,. .1. Giả sử hoán vị a = ($a_1$, $a_2$, .., $a_n$) chưa phải là hoán vị cuối cùng, khi đó hoán vị kế tiếp của a được sinh bởi qui tắc sau:

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 $a_j$ cho $a_k$;

Lật ngược lại đoạn từ $a_{j+1}, . ., a_n$.

**Thuật toán sinh hoán vị kế tiếp:**

```
void Next_Permutation( int *A, int n){
        int j, k, r, s, temp;
        j = n;
        while (aj > aj +1 )
                j = j -1;
        k = n;
        while (aj > ak )
                k= k - 1;
        temp =aj;  aj = ak; ak = temp;
        r = j + 1; s = n;
        while ( r < s) {
                temp = ar; ar = as; as = temp;
                r = r +1;       s = s - 1;
        }
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAX         20
#define TRUE        1
#define         FALSE       0
int P[MAX], n, count, Stop;
void Init(void){
        int i;count =0;
        printf("\n Nhap n=");scanf("%d", &n);
        for(i=1; i<=n; i++)
```

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

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

### 3.3.4. Bài toán chia số tự nhiên n thành tổng các số nhỏ hơn

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.

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

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

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

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

**Thuật toán sinh cách phân chia kế tiếp:**

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

Văn bản chương trình được thể hiện như sau:

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

```
#include	<stdlib.h>
#define MAX 100
#define TRUE 1
#define FALSE 0
int n, C[MAX], k, count, Stop;
void Init(void){
	printf("\n Nhap n="); scanf("%d", &n);
	k=1;count=0; C[k]=n;
}
void Result(void){
	int i; count++;
	printf("\n Cach chia %d:", count);
	for(i=1; i<=k; i++)
		printf("%3d", C[i]);
}
void Next_Division(void){
	int i, j, R, S, D;
	i = k;
	while(i>0 && C[i]==1)
		i--;
	if(i>0){
		C[i] = C[i]-1;
		D = k - i +1;
		R = D / C[i];
		S = D % C[i];
		k = i;
		if(R>0){
			for(j=i+1; j<=i+R; j++)
				C[j] = C[i];
			k = k+R;
```

```
                }
                if(S>0){
                        k=k+1; C[k] = S;
                }
        }
        else Stop=TRUE;
}
void Division(void){
        Stop = FALSE;
        while (!Stop){
                Result();
                Next_Division();
        }
}
void main(void){
        clrscr(); Init(); Division(); getch();
}
```

### 3.4- Thuật toán quay lui (Back track)

Phương pháp sinh kế tiếp có thể giải quyết được các bài toán liệt kê khi ta nhận biết được cấu hình đầu tiên 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:

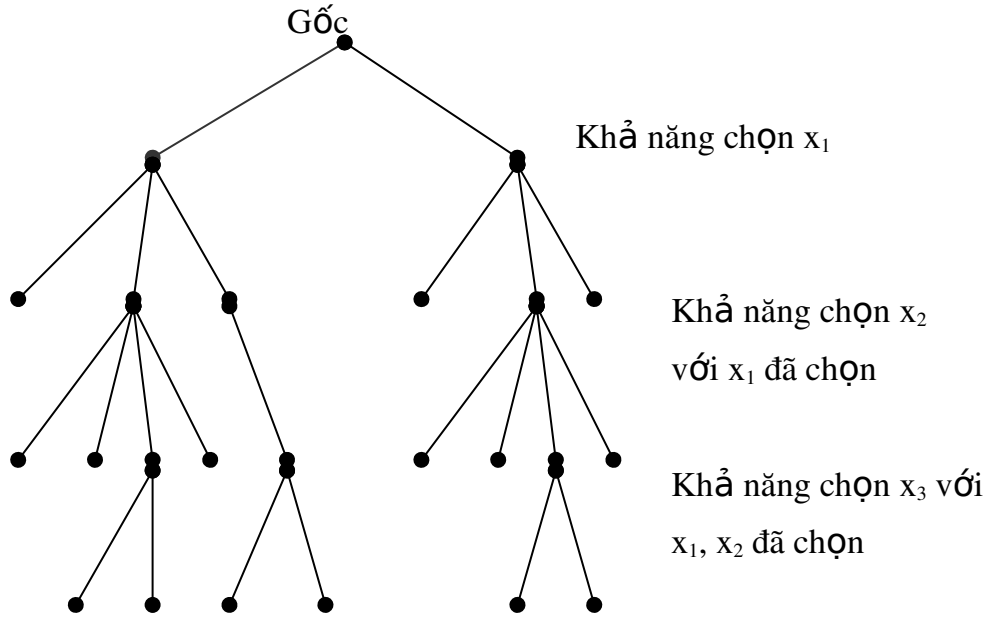
Nếu chấp nhận j thì xác định $x_i$ theo j, nếu i=n thì ta được một cấu hình cần tìm, ngược lại xác định tiếp thành phần $x_{i+1}$.

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

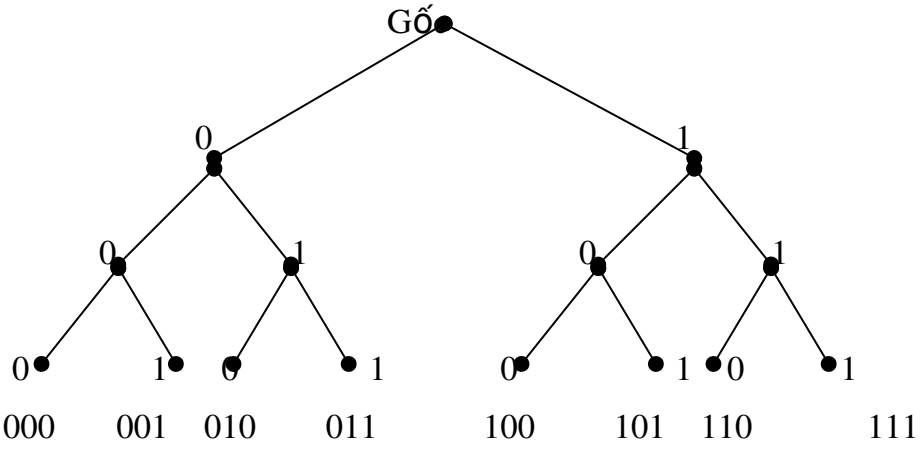
Có thể mô tả quá trình tìm kiếm lời giải theo thuật toán quay lui bằng cây tìm kiếm lời giải sau:



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

### 3.4.1. Thuật toán quay lui liệt kê các xâu nhị phân độ dài n

Biểu diễn các xâu nhị phân dưới dạng $b_1, b_2, \ldots, b_n$, trong đó bi {0, 1 }. Thủ tục đệ qui Try(i) xác định bi với các giá trị đề cử cho bi 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:

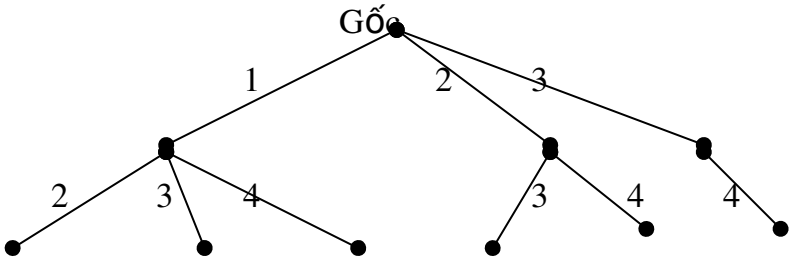
```
#include <stdio.h>
#include <alloc.h>
#include  <conio.h>
#include <stdlib.h>
void Result(int *B, int n){
        int i;
        printf("\n ");
        for(i=1;i<=n;i++)
                printf("%3d",B[i]);
}
void Init(int *B, int n){
        int i;
```

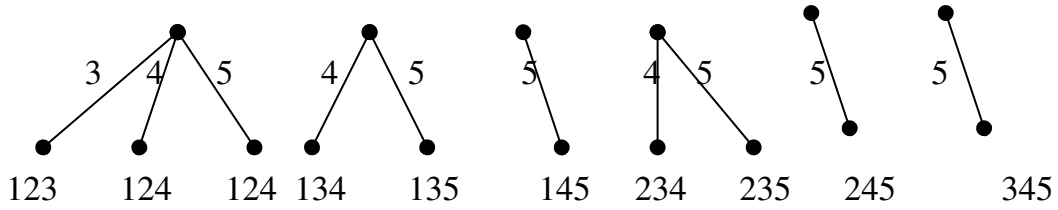
```
        for(i=1;i<=n;i++)
                B[i]=0;
}
void Try(int i, int *B, int n){
        int j;
        for(j=0; j<=1;j++){
                B[i]=j;
                if(i==n) {
                        Result(B,n);
                }
                else Try(i+1, B, n);
        }
}
void main(void){
        int *B,n;clrscr();
        printf("\n Nhap n=");scanf("%d",&n);
        B=(int *) malloc(n*sizeof(int));
        Init(B,n); Try(1,B,n);free(B);
        getch();
}
```

### 3.4.2. Thuật toán quay lui liệt kê các tập con m phần tử của tập n phần tử

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

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






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

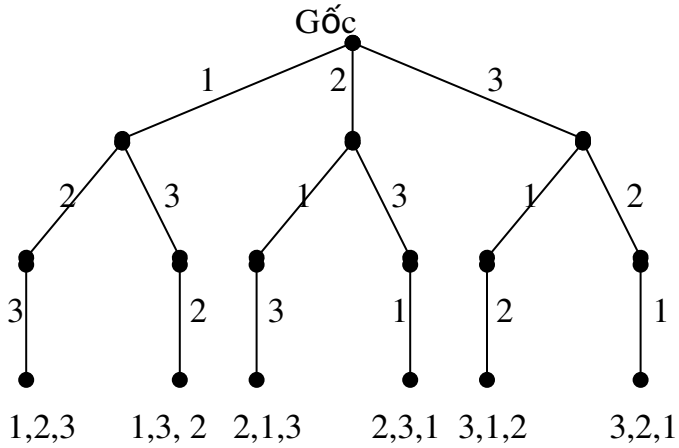
Chương trình liệt kê các tập con k phần tử trong tập n phần tử được thể hiện như sau:

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

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

### 3.4.3- Thuật toán quay lui liệt kê các hoán vị của tập n phần tử

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    <stdio.h>
#include    <conio.h>
#include    <stdlib.h>
#define MAX 100
#define TRUE 1
#define FALSE 0
```

```
int P[MAX],B[MAX], n, count=0;
void Init(void){
	int i;
	printf("\n Nhap n="); scanf("%d", &n);
	for(i=1; i<=n; i++)
		B[i]=TRUE;

}
void Result(void){
	int i; count++;
	printf("\n Hoan vi thu %d:",count);
	for (i=1; i<=n; i++)
		printf("%3d",P[i]);
	getch();
}
void Try(int i){
	int j;
	for(j=1; j<=n;j++){
		if(B[j]) {
			P[i]=j;
			B[j]=FALSE;
			if(i==n) Result();
			else Try(i+1);
			B[j]=TRUE;
		}
	}
}
void main(void){
	Init(); Try(1);
}
```

### 3.4.4- Bài toán Xếp Hậu

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

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

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

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

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#define N     8
#define       D       (2*N-1)
#define       SG      (N-1)
#define       TRUE 1
#define       FALSE        0
void hoanghau(int);
```

```
void inloigiai(int      loigiai[]);FILE *fp;
int      A[N], B[D], C[D], loigiai[N];
int soloigiai =0;
void hoanghau(int i){
        int j;
        for (j=0; j<N;j++){
                if (A[j] && B[i-j+SG] && C[i+j] ) {
                        loigiai[i]=j;
                        A[j]=FALSE;
                        B[i-j+SG]=FALSE;
                        C[i+j]=FALSE;
                        if (i==N-1){
                                soloigiai++;
                                inloigiai(loigiai);
                                delay(500);
                        }
                        else
                                hoanghau(i+1);
                        A[j]=TRUE;
                        B[i-j+SG]=TRUE;
                        C[i+j]=TRUE;
                }
        }
}
void inloigiai(int *loigiai){
        int j;
                printf("\n Lời giải %3d:",soloigiai);
                fprintf(fp,"\n Lời giải %3d:",soloigiai);
                for (j=0;j<N;j++){
                        printf("%3d",loigiai[j]);
```

```
            fprintf(fp,"%3d",loigiai[j]);
        }
}
void main(void){
    int i;clrscr();fp=fopen("loigiai.txt","w");
    for (i=0;i<N;i++)
        A[i]=TRUE;
    for(i=0;i<D; i++){
        B[i]=TRUE;
        C[i]=TRUE;
    }
    hoanghau(0);fclose(fp);
}
```

### 3.5.- Bài toán tối ưu

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

Tìm cực tiểu (hay cực đại) của phiếm hàm $f(x) = min(max)$ với điều kiện $x \in D$, trong đó D là tập hữu hạn các phần tử.

Hàm $f(x)$ được gọi là hàm mục tiêu của bài toán, mỗi phần tử $x \in D$ được gọi là một phương án, tập $D$ được gọi là tập các phương án của bài toán. Thông thường tập $D$ được mô tả như là tập các cấu hình tổ hợp thỏa mãn một số tính chất cho trước nào đó.

Phương án $x^* \in D$ đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là phương án tối ưu, khi đó giá trị $f^* = f(x^*)$ được gọi là giá trị tối ưu của bài toán. Dưới đây chúng ta sẽ giới thiệu một số mô hình tối ưu hóa tổ hợp truyền thống. Các mô hình này là những mô hình có nhiều ứng dụng thực tế, đồng thời chúng giữ vai trò quan trọng trong việc nghiên cứu và phát triển lý thuyết tối ưu hoá tổ hợp.

**Bài toán Người du lịch:** Một người du lịch muốn đi thăm quan n thành phố $T_1, T_2, \ldots, T_n$. 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).

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

$$f(\ ) \quad C_{(1),\ (2)} \quad C_{(2),\ (3)} \quad ... \quad C_{(n\ 1),\ (n)} \quad C_{(n),\ (1)},$$

kí hiệu là tập tất cả các hoán vị $=(\ (1),\ (2) , . . .,\ (n))$ của n số tự nhiên 1, 2, . ., n. Khi đó bài toán người du lịch có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

$$\min \{ f(\ ) : \quad \}$$

Có thể thấy rằng, tổng số hành trình của người du lịch là n!, trong đó chỉ có (n-1)! hành trình thực sự khác nhau (bởi vì có thể xuất phát từ một thành phố bất kỳ nên có thể cố định một thành phố nào đó làm điểm xuất phát).

**Bài toán cái túi:** Một nhà thám hiểm cần đem theo một cái túi có trọng lượng không quá *b*. Có n đồ vật có thể đem theo. Đồ vật thứ j có trọng lượng $a_j$ và giá trị sử dụng $c_j$ (j =1 , 2 , . . , n). Hỏi nhà thám hiểm cần đem theo những đồ vật nào để cho tổng giá trị sử dụng là lớn nhất ?

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

$f(x) \quad {}^{n}_{i\ 1} c_i x_i$, tổng trọng lượng đồ vật đem theo là $g(x) \quad {}^{n}_{i\ 1} a_i x_i$, 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) b, hãy tìm vector x* cho giá trị lớn nhất của hàm mục tiêu f(x). Nói cách khác:

Max { f(x) : g(x) b }

**Bài toán cho thuê máy:** Một ông chủ có một cái máy để cho thuê. Đầu tháng ông ta nhận được yêu cầu thuê máy của m khách hàng. Mỗi khách hàng i sẽ cho biết tập $N_i$ các ngày trong tháng cần sử dụng máy ( i = 1, 2, . ., m). Ông chủ chỉ có quyền hoặc từ chối yêu cầu của khách hàng i, hoặc nếu nhận thì phải bố trí máy phục vụ khách hàng i đúng những ngày mà khách hàng này yêu cầu. Hỏi rằng ông chủ phải tiếp nhận các yêu cầu của khách thế nào để cho tổng số ngày sử dụng máy là lớn nhất.

Ký hiệu, I = { 1, 2, . ., 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

$D \quad J \quad S : N_k \quad N_p \quad , \quad k \quad p \quad J$ . Với mỗi phương án J D $f(j)$ $_{j\ J} | N_j |$ sẽ là tổng số ngày sử dụng máy theo phương án đó. Bài toán đặt ra có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

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

**Bài toán phân công:** Có n công việc và n thợ. Biết $c_{ij}$ là chi phí cần trả để thợ i hoàn thành công việc thứ j (i, j = 1, 2, . . ., n ). Cần phải thuê thợ sao cho các công việc đều hoàn thành và mỗi thợ chỉ thực hiện một công việc, mỗi công việc chỉ do một thợ thực hiện. Hãy tìm cách thuê n nhân công sao cho tổng chi phí thuê thợ là nhỏ nhất.

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

| Công việc | Thợ thực hiện |
|---|---|
| 1 | (1) |
| 2 | (2) |
| . . . | . . . |
| n | (n) |

Bài toán đặt ra được dẫn về bài toán tối ưu tổ hợp: min $f( ) :$ }.

**Bài toán lập lịch:** Mỗi một chi tiết trong số n chi tiết $D_1, D_2, . ., D_n$ cần phải lần lượt được gia công trên m máy $M_1, M_2, . ., M_m$. 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 các máy khác nhau.

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

$f( ) \quad \sum_{j\ 1}^{n\ 1} C_{(j),\ (j\ 1)} \quad \sum_{k\ 1}^{m} t_{k,\ (n)}$ , 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:

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

min { f( ) : }.

Trong thực tế, lịch gia công còn phải thỏa mãn thêm nhiều điều kiện khác nữa. Vì những ứng dụng quan trọng của những bài toán loại này mà trong tối ưu hoá tổ hợp đã hình thành một lĩnh vực lý thuyết riêng về các bài toán lập lịch , đó là lý thuyết lập lịch hay qui hoạch lịch.

### 3.6.Thuật toán duyệt

Một trong những phương pháp hiển nhiên nhất để giải bài toán tối ưu tổ hợp đặt ra là duyệt toàn bộ. Trên cơ sở các thuật toán lệt kê tổ hợp, ta tiến hành duyệt từng phương án của bài toán. Đối với mỗi phương án, ta đều tính giá trị hàm mục tiêu tại nó, sau đó so sánh giá trị của hàm mục tiêu tại tất cả các phương án đã được liệt kê để tìm ra phương án tối ưu. Phương pháp xây dựng theo nguyên tắc như vậy được gọi là phương pháp duyệt toàn bộ. Hạn chế của phương pháp duyệt toàn bộ là sự bùng nổ của các cấu hình tổ hợp. Chẳng hạn, để duyệt được 15! = 1 307 674 368 000 cấu hình, trên máy có tốc độ 1 tỷ phép tính giây, nếu mỗi hoán vị cần liệt kê mất khoảng 100 phép tính, 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 ta có thể xây dựng được thuật toán hiệu quả để giải quyết bài toán đặt ra. Nhưng cũng cần phải chú ý rằng, trong nhiều trường hợp ( bài toán người du lịch, bài toán cái túi, bài toán cho thuê máy) chúng ta vẫn chưa tìm ra được một phương pháp hữu hiệu nào ngoài phương pháp duyệt toàn bộ đã được đề cập ở trên. Ví dụ sau đây là một điển hình cho phương pháp duyệt toàn bộ.

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

Trong thực tế rất thường hay gặp bài toán tối ưu tổ hợp được cho dưới dạng sau:

Tìm

$\max f(x_1, x_2, ..., x_n) : x_i \quad D_i; i \quad 1,2,...,n$

hoặc $\min f(x_1, x_2, ..., x_n) : x_i \quad D_i; i \quad 1,2,...,n$ .

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

Giả sử số các phần tử của tập giá trị rời rạc $D_i$ là $r_i$ ( i=1, 2, . . ., n). Gọi $R = r_1 + r_2 + . . . + r_n$ là số các phần tử thuộc tất cả các tập Di (i=1, 2, . . ., n). Khi đó, ta có tất

cả C(R, n) bộ có thứ tự các giá trị gồm n phần tử trong R phần tử. Trong số C(R,n) các bộ n phần tử, ta cần lọc ra các bộ thoả mãn điều kiện $x_i \in D_i$ (i=1, 2, . . ., n) để tính giá trị của hàm mục tiêu $f(x_1, x_2, . . ., x_n)$. Như vậy, bài toán được đưa về bài toán duyệt các bộ gồm n phần tử $(x_1, x_2, . . ., x_n)$ từ tập hợp gồm $R = r_1 + r_2 + . . + r_n$ phần tử thoả mãn điều kiện $x_i \in D_i$.

Ví dụ: với tập D1 = (1, 2, 3),

D2 = (3, 4),

D3 = (5, 6, 7).

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

| | | | | | |
|---|---|---|---|---|---|
| 1 | 3 | 5 | 2 | 4 | 5 |
| 1 | 3 | 6 | 2 | 4 | 6 |
| 1 | 3 | 7 | 2 | 4 | 7 |
| 1 | 4 | 5 | 3 | 3 | 5 |
| 1 | 4 | 6 | 3 | 3 | 6 |
| 1 | 4 | 7 | 3 | 3 | 7 |
| 2 | 3 | 5 | 3 | 4 | 5 |
| 2 | 3 | 6 | 3 | 4 | 6 |
| 2 | 3 | 7 | 3 | 4 | 7 |

Văn bản chương trình được thể hiện như sau:

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

```
        fscanf(fp,"%d",&n);
        printf("\n So tap con roi rac n=%d",n);
        for(i=1; i<=n; i++){
                fscanf(fp,"%d",&H[i]);
                printf("\n Hang %d co so phan tu la %d",i, H[i]);
        }
        H[0]=0;
        for (i=1; i<=n; i++){
                printf("\n");
                for(j=1; j<=H[i]; j++){
                        fscanf(fp,"%f",&x);
                        B[++k]=x;
                }
        }
        printf("\n B=");
        for(i=1; i<=k; i++){
                printf("%8.2f", B[i]);
        }
        fclose(fp);
}
int In_Set(int i){
        int canduoi=0, cantren=0,j;
        for(j=1; j<=i; j++)
                cantren = cantren + H[j];
        canduoi=cantren-H[j-1];
        if (C[i]> canduoi && C[i]<=cantren)
                return(TRUE);
        return(FALSE);
}
void Result(void){
```

```
        int i;
        count++; printf("\n Tap con thu count=%d:",count);
        for(i=1; i<=n ; i++){
                printf("%8.2f", B[C[i]]);
        }
}
void Try(int i){
        int j;
        for(j = C[i-1]+1; j<=(k-n+i); j++){
                C[i]=j;
                if(In_Set(i)){
                        if (i==n )  Result();
                        else Try(i+1);
                }
        }
}
void main(void){
        clrscr();
        B = (float *) malloc(MAX *sizeof(float));
        C = (int *) malloc(MAX *sizeof(int));
        Init();Try(1);free(B); free(C);getch();
}
```

### 3.7- Thuật toán nhánh cận

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

$$\min f(x) : x \quad D$$

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

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

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. Một bộ phận gồm k thành phần ($a_1$, $a_2$, . . ., $a_k$) 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:

$$g(a_1, a_2, .., a_k) \quad \min f(x) : x \quad D, x_i \quad a_i, i \quad 1,2,...,k \qquad (*)$$

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

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

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

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

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

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

```
Procedure Try(k);
(*Phát triển phương án bộ phận (a1, a2, . . ., ak-1
theo thuật toán quay lui có kiểm tra cận dưới
trước khi tiếp tục phát triển phương án*)
begin
    for ak    Ak do
    begin
        if < chấp nhận ak > then
        begin
            xk := ak;
            if k = n then < cập nhật kỷ lục>
            else if g(a1, a2, . . ., ak)     f  then Try (k+1);
        end;
    end;
end;
```

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

```
Procedure Nhanh_Can;
Begin
    f = +   ;
(* Nếu biết một phương án x nào đó thì có thể đặt f    f(x)   *)
    Try(1);
    if  f     +    then  < f là giá trị tối ưu , x là phương án tối ưu >
    else < bài toán không có phương án>;
end;
```

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

```
    if k = n then < cập nhật kỷ lục >
    else if g(a1, a2, . ., ak)     f  then Try(k+1);
```

bởi

```
    if k = n then < cập nhật kỷ lục >
```

else Try(k+1);

thì thủ tục Try sẽ liệt kê toàn bộ các phương án của bài toán, và ta lại thu được thuật toán duyệt toàn bộ. Việc xây dựng hàm g phụ thuộc vào từng bài toán tối ưu tổ hợp cụ thể. Nhưng chúng ta cố gắng xây dựng sao cho việc tính giá trị của g phải đơn giản và giá trị của $g(a_1, a_2, . ., a_k)$ phải sát với giá trị của hàm mục tiêu.

### 3.7.1. Thuật toán nhánh cận giải bài toán cái túi

Chúng ta sẽ xét bài toán cái túi tổng quát hơn mô hình đã được trình bày trong mục trên. 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:

$$f^* \quad \max \; f(x) \quad \sum_{j\ 1}^{n} c_j x_j : \sum_{j\ 1}^{n} a_j x_j \quad b, x_j \quad Z \ , j \quad 1,2,...,n \ , (1)$$

.

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

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

$$D \quad x \quad (x_1, x_2, ..., x_n : \sum_{j\ 1}^{n} a_j x_j \quad b, x_j \quad Z \ , j \quad 1,2,...,n \ .$$

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

$$\frac{c_1}{a_1} \quad \frac{c_2}{a_2} \quad ... \quad \frac{c_n}{a_n} \qquad (2)$$

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

$$g^* \quad \max \sum_{j\ 1}^{n} c_j x_j : \sum_{j\ 1}^{n} a_j x_j \quad b, x_j \quad 0, j \quad 1,2,...,n \ . \qquad (3)$$

Người ta đã chứng minh được rằng phương án tối ưu của bài toán (3) là vector

với các thành phần được xác định bởi công thức:

$$\overline{x_1} \quad \frac{b}{a_1}, \overline{x_2} \quad \overline{x_3} \quad ... \quad \overline{x_n} \quad 0$$

và giá trị tối ưu là

$$g^* \quad \frac{c_1 b_1}{a_1}.$$

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

$$_k \quad c_1u_1 \quad c_2u_2 \quad ... \quad c_ku_k,$$

và trọng lượng còn lại của túi là

$$b_k \quad b \quad c_1u_1 \quad c_2u_2 \quad ... \quad c_ku_k,$$

ta có

$$\max \ f(x) : x \quad D, x_j \quad u_j, j \quad 1,2,...,n$$

$$\max \quad _k \quad \sum_{j\ k\ 1}^{n} c_j x_j : \sum_{j\ k\ 1}^{n} a_j x_j \quad b_k, x_j \quad Z \ , j \quad k \quad 1, k \quad 2,...,n$$

$$_k \quad \max \sum_{j\ k\ 1}^{n} c_j x_j : \sum_{j\ k\ 1}^{n} a_j x_j \quad b_k, x_j \quad 0, j \quad k \quad 1, k \quad 2,...,n$$

$$_k \quad \frac{c_{k\ 1} b_k}{a_{k\ 1}}$$

(Như vậy giá trị số hạng thứ hai là $c_{k\ 1} b_k / a_{k\ 1}$)

Từ đó ta có thể tính cận trên cho phương án bộ phận ($u_1$, $u_2$, . . ., $u_k$) theo công thức

$$g(u_1, u_2, ..., u_k) \quad _k \quad \frac{c_{k\ 1} b_k}{a_{k\ 1}}$$

Chú ý: Khi tiếp tục xây dựng thành phần thứ k+1 của lời giải, các giá trị đề cử cho $x_{k+1}$ sẽ là 0, 1, . . ., $[b_k/a_{k+1}]$. Do có kết quả của mệnh đề, khi chọn giá trị cho $x_{k+1}$ ta sẽ duyệt các giá trị đề cử theo thứ tự giảm dần. Chẳng hạn, giải bài toán cái túi sau theo thuật toán nhánh cận trình bày

$$f(x) \quad 10x_1 \quad 5x_2 \quad 3x_3 \quad 6x_4 \quad \max$$
$$5x_1 \quad 3x_2 \quad 2x_3 \quad 4x_4 \quad 8$$
$$x_j \quad Z \ , j \quad 1,2,3,4.$$

Quá trình giải bài toán được mô tả trong cây tìm kiếm trong hình 3.5. Thông tin về một phương án bộ phận trên cây được ghi trong các ô trên hình vẽ tương ứng theo thứ tự sau:

đầu tiên là các thành phần của phương án

tiếp đến là giá trị của các đồ vật chất trong túi

u là trọng lượng còn lại của túi

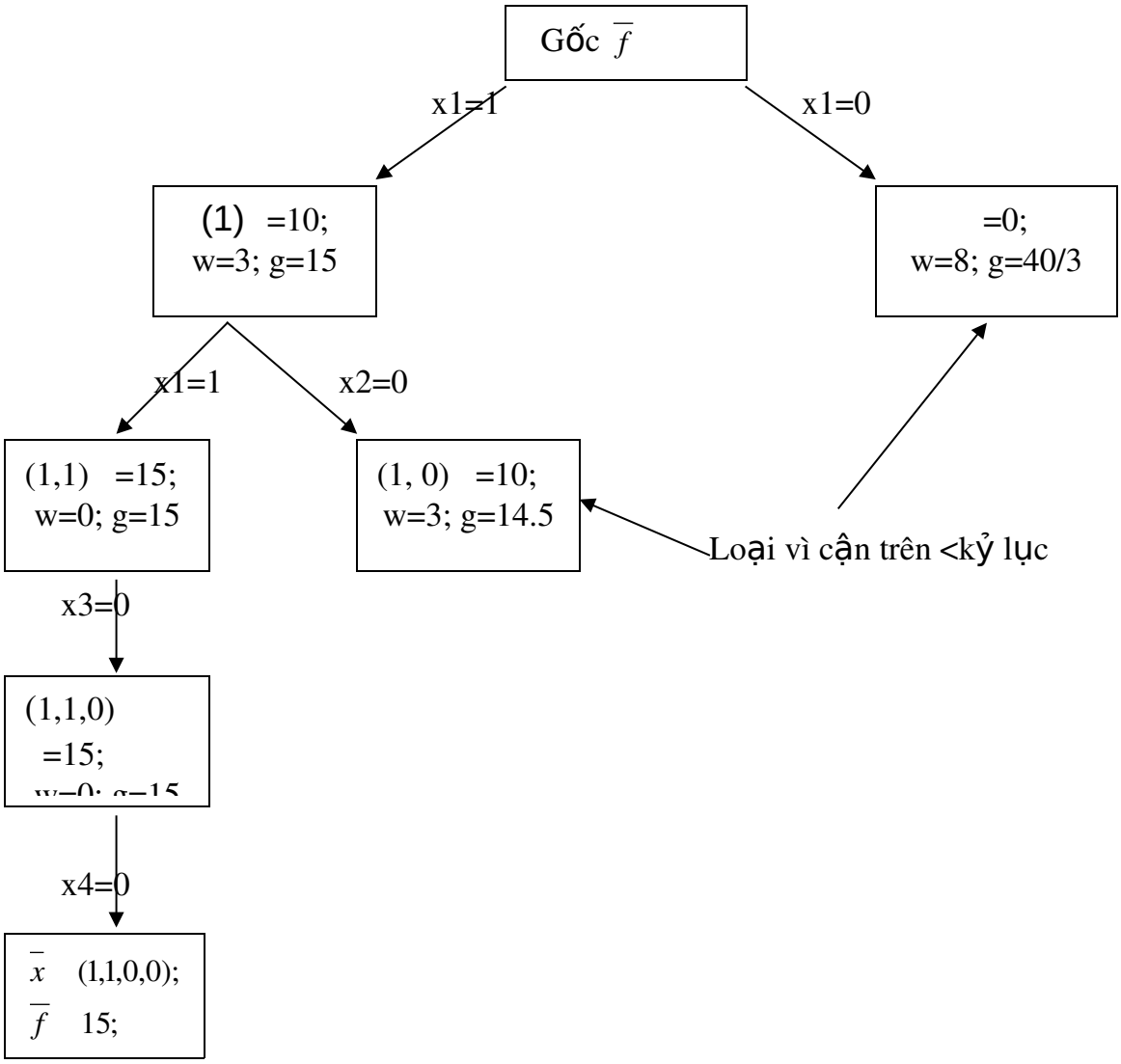
g là cận trên.

Kết thúc thuật toán, ta thu được phương án tối ưu là

x* =(1, 1, 0, 1),

giá trị tối ưu f*= 15.

*Hình 3.5. Giải bài toán cái túi theo thuật toán nhánh cận.*




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

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#define TRUE        1
#define FALSE       0
```

```
#define MAX          100
int      x[MAX], xopt[MAX];
float    fopt, cost, weight;
void Init(float *C, float *A, int *n, float *w){
        int i;FILE *fp;
        fopt=0; weight=0;
        fp=fopen("caitui.in","r");
        if(fp==NULL){
                printf("\n Khong co file input");
                delay(2000); return;
        }
        fscanf(fp,"%d %f", n,w);
        for(i=1; i<=*n;i++) xopt[i]=0;
        printf("\n So luong do vat %d:", *n);
        printf("\n Gioi han tui %8.2f:", *w);
        printf("\n Vecto gia tri:");
        for(i=1; i<=*n; i++) {
                fscanf(fp,"%f", &C[i]);
                printf("%8.2f", C[i]);
        }
        printf("\n Vector trong luong:");
        for(i=1; i<=*n; i++){
                fscanf(fp,"%f", &A[i]);
                printf("%8.2f", A[i]);
        }
        fclose(fp);
}
void swap(int n){
        int i;
        for(i=1; i<=n; i++)
```

```
                xopt[i]=x[i];
}
void Update_Kyluc(int n){
        if(cost>fopt){
                swap(n);
                fopt=cost;
        }
}
void Try(float *A, float *C, int n, float w, int i){
        int j, t=(w-weight)/A[i];
        for(j=t; j>=0;j--){
                x[i]=j;
                cost = cost + C[i]*x[i];
                weight = weight + x[i]*A[i];
                if(i==n) Update_Kyluc(n);
                else if(cost + C[i+1]*(w-weight)/A[i+1]> fopt){
                        Try(A, C, n, w, i+1);
                }
                weight = weight-A[i]*x[i];
                cost = cost-C[i]*x[i];

        }
}
void Result(int n){
        int i;
        printf("\n Gia tri do vat %8.2f:", fopt);
        printf("\n Phuong an toi uu:");
        for(i=1; i<=n; i++)
                printf("%3d", xopt[i]);
}
```

```
void main(void){

        int     n;
        float   A[MAX], C[MAX], w;
        clrscr();Init(C, A, &n, &w);
        Try(C, A, n, w,1);Result(n);
        getch();
}
```

### 3.7.2. Thuật toán nhánh cận giải bài toán người du lịch

**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 (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:

$$f(x_1, x_2, ..., x_n) \quad c[1, x_2] \quad c[x_2, x_3] \quad ... \quad c[x_{n\ 1}, x_n] \quad c[x_n, x_1] \quad \min$$

với điều kiện

$c_{\min}$ $\min\ c[i, j], i, j$ $1,2,...,n; i$ $j$ 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:

$$T_1 \quad T(u_2) \quad ... \quad T(u_{k\ 1}) \quad T(u_k)$$

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.

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

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

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

Chẳng hạn, 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 3.6.

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

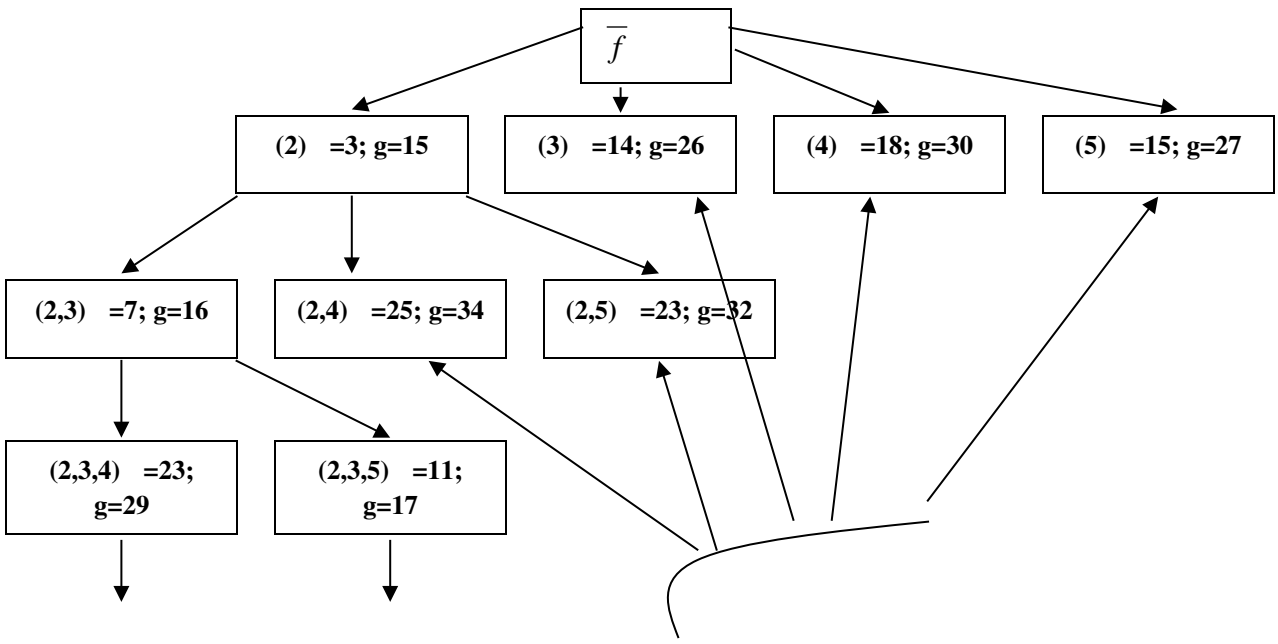
g là cận dưới

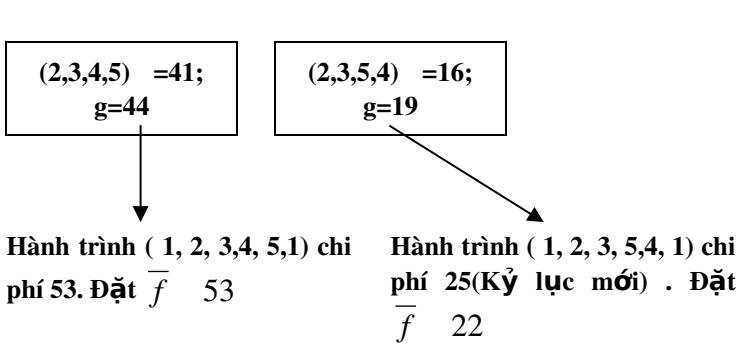
Kết thúc thuật toán, ta thu được phương án tối ưu ( 1, 2, 3, 5, 4, 1) tương ứng với phương án tối ưu với hành trình

$T_1 \quad T_2 \quad T_3 \quad T_5 \quad T_4 \quad T_1$

và chi phí nhỏ nhất là 22

***Hình 3.6. 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();

}
```

## BÀI TẬP CHƯƠNG 3

**3.1. Duyệt mọi tập con của tập hợp 1, 2, . . ., n.** Dữ liệu vào cho bởi file tapcon.in, kết quả ghi lại trong file bai11.out. Ví dụ sau sẽ minh họa cho file tapcon.in và tapcon.out.

| tapcon.in | tapcon.out |
|---|---|
| 3 | 1 |
| | 2 |
| | 2 1 |
| | 3 |
| | 3 1 |
| | 3 2 |
| | 3 2 1 |

**3.2. 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 | tapcon.out |
|---|---|
| 5 | 5 |
| 7 1 3 8 9 6 12 | 1 3 8 9 12 |

**3.3. 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

3.4. Cho lưới hình chữ 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

**3.5. Duyệt mọi tập con k phần tử từ tập gồm n phần tử.** Dữ liệu vào cho bởi file tapcon.in, kết quả ghi lại trong file tapcon.out. Ví dụ sau sẽ minh họa cho tapcon.in và tapcon.out.

tapcon.in

5 3

tapcon.out

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

3.6. Duyệt các tập con k phần tử thỏa mãn điều kiện. Cho dãy số $a_1, a_2, . . ., a_n$ và số M. Hãy tìm tất cả các dãy con dãy con k phần tử trong dãy số $a_1, a_2, . . ., a_n$ 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 số tự nhiên n , k và số M, hai số được viết cách nhau bởi một vài ký tự trống, dòng kế tiếp ghi lại n số mỗi số được viết cách nhau bởi một hoặc vài ký tự 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 3 50

5 10 15 20 25 30 35

tapcon.out

5 10 35

5 15 35

5 20 25

10 15 25

3.7. Duyệt mọi hoán vị của từ COMPUTER. Dữ liệu vào cho bởi file hoanvi.in, kết quả ghi lại trong file hoanvi.out.

3.8. Duyệt mọi ma trận các hoán vị. Cho hình vuông gồm n n (n 5, n lẻ) hình vuông đơn vị. Hãy điền các số từ 1, 2, . . ., n vào các hình vuông đơn vị sao cho những điều kiện sau được thoả mãn:

a) Đọc theo hàng ta nhận được n hoán vị khác nhau của 1, 2, . . ., n;

b) Đọc theo cột ta nhận được n hoán vị khác nhau của 1, 2, . . ., n;

c) Đọc theo hai đường chéo ta nhận được 2 hoán vị khác nhau của 1, 2, . . ., n;

Hãy tìm ít nhất 1 (hoặc tất cả) các hình vuông thoả mãn 3 điều kiện trên. Dữ liệu vào cho bởi file hoanvi.in, kết quả ghi lại trong file hoanvi.out. Ví dụ sau sẽ minh họa cho file input & output của bài toán.

hoanvi.in

```
5
```

hoanvi.out

```
5  3  4  1  2
1  2  5  3  4
3  4  1  2  5
2  5  3  4  1
4  1  2  5  3
```

3.9. Duyệt mọi cách chia số tự nhiên n thành tổng các số nguyên nhỏ hơn. Dữ liệu vào cho bởi file chiaso.in, kết quả ghi lại trong file chiaso.out. Ví dụ sau sẽ minh họa cho file input & output của bài toán.

chiaso.in

```
4
```

chiaso.out

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

**3.10. Duyệt mọi bộ giá trị trong tập các giá trị rời rạc.** Cho k tập hợp các số thực $A_1, A_2, . . ., A_k$(k 10) có số các phần tử tương ứng là $N_1, N_2, . . ., N_k$ ( các tập có thể có những phần tử giống nhau). Hãy duyệt tất cả các bộ k phần tử $a=(a_1, a_2, . . ., a_k)$ sao cho $a_i$ $A_i$(i=1, 2, . . ., k). Dữ liệu vào cho bởi file chiaso.in, dòng đầu tiên ghi lại k+1 số tự nhiên, mỗi số được phân biệt với nhau bởi một vài dấu trống là giá trị của n, $N_1$,

$N_2$, . . ., $N_k$; k dòng kế tiếp ghi lại các phần tử của tập hợp $A_1$, $A_2$, . . ., $A_k$. Kết quả ghi lại trong file chiaso.out, mỗi phần tử được phân biệt với nhau bởi một vài dấu trống. Ví dụ sau sẽ minh họa cho file input & output của bài toán.

Chiaso.inp

| | | | |
|---|---|---|---|
| 3 | 3 | 2 | 2 |
| 1 | 2 | 3 | |
| 4 | 5 | | |
| 6 | 7 | | |

chiaso.out

| | | |
|---|---|---|
| 1 | 4 | 6 |
| 1 | 4 | 7 |
| 1 | 5 | 6 |
| 1 | 5 | 7 |
| 2 | 4 | 6 |
| 2 | 4 | 7 |
| 2 | 5 | 6 |
| 2 | 5 | 7 |
| 3 | 4 | 6 |
| 3 | 4 | 7 |
| 3 | 5 | 6 |
| 3 | 5 | 7 |

3.11. Tìm bộ giá trị rời rạc trong bài 21 để hàm mục tiêu $\sin(x_1+x_2 + . . .+ x_k)$ đạt giá trị lớn nhất. Dữ liệu vào cho bởi file bai22.inp, kết quả ghi lại trong file bai22.out.

3.12. 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.

3.13. Bài toán cái túi với số lượng đồ vật không hạn chế. Một nhà thám hiểm đem theo một cái túi có trọng lượng không quá b. Có n đồ vật cần đem theo, đồ vật thứ i có trọng lượng tương ứng là một số $a_i$ và giá trị sử dụng $c_i$ (1 i n). Hãy tìm cách bỏ các đồ vật vào túi sao cho tổng giá trị sử dụng các đồ vật là lớn nhất. Biết rằng số lượng các đồ vật là không hạn chế. Dữ liệu vào cho bởi file caitui.in, dòng đầu tiên ghi lại số tự nhiên n và số thực b hai số được viết cách nhau bởi một dấu trống, hai dòng kế tiếp ghi n số trên mỗi dòng, tương ứng với vector giá trị sử dụng $c_i$ và vector trọng lượng $a_i$. Kết quả ghi lại trong file caitui.out trên 3 dòng, dòng đầu ghi lại giá trị sử dụng tối ưu, dòng kế tiếp ghi lại loại đồ vật cần đem theo, dòng cuối cùng ghi lại số lượng của mỗi loại đồ vật. Ví dụ sau sẽ minh họa cho file input & output của bài toán.

caitui.in

```
4   8
    10   5   3   6
    5    3   2   4
```

caitui.out

```
    15
    1    1   0   0
    1    1   0   0
```

3.14. Bài toán cái túi với số lượng đồ vật hạn chế. Một nhà thám hiểm đem theo một cái túi có trọng lượng không quá b. Có n đồ vật cần đem theo, đồ vật thứ i có trọng lượng tương ứng là một số $a_i$ và giá trị sử dụng $c_i$ (1 i n). Hãy tìm cách bỏ các đồ vật vào túi sao cho tổng giá trị sử dụng các đồ vật là lớn nhất. Biết rằng số lượng mỗi đồ vật là 1. Dữ liệu vào cho bởi file caitui.in, dòng đầu tiên ghi lại số tự nhiên n và số thực b hai số được viết cách nhau bởi một dấu trống, hai dòng kế tiếp ghi n số trên mỗi dòng, tương ứng với vector giá trị sử dụng $c_i$ và vector trọng lượng $a_i$. Kết quả ghi lại trong file caitui.out trên 2 dòng, dòng đầu ghi lại giá trị sử dụng tối ưu, dòng kế tiếp ghi lại loại đồ vật cần đem theo. Ví dụ sau sẽ minh hoạ cho file input & output của bài toán.

caitui.in

```
    4   8
    8   5   3   1
    4   3   2   1
```

caitui.out

```
    14
```

1 1 0 1

3.15. Bài toán người du lịch. Một người du lịch muốn đi tham quan tại n thành phố khác nhau. Xuất phát tại một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại mỗi thành phố đúng một lần rồi quay trở lại thành phố ban đầu. Biết Cij là chi phí đi lại từ thành phố thứ i đến thành phố thứ j. Hãy tìm hành trình có chi phí thấp nhất cho người du lịch. Dữ liệu vào cho bởi file dulich.in, dòng đầu tiên ghi lại số tự nhiên n, n dòng kế tiếp ghi lại ma trận chi phí Cij. Kết quả ghi lại trong file dulich.out, dòng đầu tiên ghi lại chi phí tối ưu, dòng kế tiếp ghi lại hành trình tối ưu. Ví dụ sau sẽ minh họa cho file input & output của bài toán.

dulich.in

5

00 48 43 54 31

20 00 30 63 22

29 64 00 04 17

06 19 02 00 08

01 28 07 18 00

dulich.out

81

1 5 3 4 2 1