Phần thứ nhất

LÝ THUYẾT TỔ HỢP

Combinatorial Theory

GV: Nguyễn Huy Đức

Bộ môn: Tin học và Kỹ thuật Máy tính – ĐH Thủy lợi

 0903 402 655

 ducnghuy@gmail.com

Nội dung

Chương 5

Bài toán tối ưu tổ hợp

NỘI DUNG

- 5.1 Giới thiệu bài toán
- 5.2 Phương pháp duyệt toàn bộ
- 5.3 Thuật toán nhánh cận

5.1 Giới thiệu bài toán

- Trong rất nhiều vấn đề ứng dụng thực tế của tổ hợp, các cấu hình tổ hợp được gán cho một giá trị bằng số đánh giá giá trị sử dụng của cấu hình đối với mục đích sử dụng cụ thể nào đó.
- Khi đó xuất hiện bài toán: *Hãy lựa chọn trong số các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất*. Các bài toán như vậy chúng ta sẽ gọi là bài toán tối ưu tổ hợp.

5.1 Giới thiệu bài toán

- Dưới dạng tổng quát, bài toán tối ưu tổ hợp có thể phát biểu như sau:

Tìm cực tiểu (hay cực đại) của hàm mục tiêu f(x)

$f(x) \rightarrow min\ (max)$,

với điều kiện $x \in D$.

Trong đó, D là tập hữu hạn các phương án của bài toán, được mô tả như là tập các cấu hình tổ hợp thoả mãn một số tính chất cho trước.

Phương án $x^* \in D$ để hàm mục tiêu $f(x)$ đạt giá trị nhỏ nhất (hoặc lớn nhất) gọi là phương án tối ưu, khi đó giá trị $f^* = f(x^*)$ gọi là giá trị tối ưu của bài toán.

Một số bài toán tối ưu tổ hợp

- **Bài toán người du lịch**
  (Traveling Salesman Problem – TSP)
- **Bài toán cái túi**
  (Knapsack Problem)
- **Bài toán đóng thùng**
  (Bin Packing)

Bài toán người du lịch
(Traveling Salesman Problem – TSP)

- Một người du lịch muốn đi tham quan n thành phố $T_1, T_2, ..., T_n$.
- *Hành trình là cách đi xuất phát từ một thành phố nào đó, đi qua tất cả các thành phố còn lại, mỗi thành phố đúng một lần, rồi quay trở lại thành phố xuất phát.*
- Biết c_{ij} là chi phí đi từ thành phố T_i đến thành phố T_j ($i, j = 1, 2,..., n$),
- **Tìm hành trình với tổng chi phí là nhỏ nhất.**

Bài toán người du lịch
(Traveling Salesman Problem – TSP)

- Ta có mỗi một hành trình

$$T_{\pi(1)} \to T_{\pi(2)} \to \ldots \to T_{\pi(n)} \to T_{\pi(1)}$$

tương ứng 1-1 với một hoán vị của n số tự nhiên {1, 2,..., n} là $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$.

- Đặt $f(\pi) = c_{\pi(1),\pi(2)} + \ldots + c_{\pi(n-1),\pi(n)} + c_{\pi(n),\pi(1)}$.

- Ký hiệu:

Π - tập tất cả các hoán vị của n số tự nhiên {1, 2,..., n}.

Bài toán người du lịch
(Traveling Salesman Problem – TSP)

- Khi đó bài toán người du lịch có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

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

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

Bài toán cái túi
(Knapsack Problem)

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

Bài toán cái túi
(Knapsack Problem)

- Một phương án đem đồ của nhà thám hiểm có thể biểu diễn bởi bộ n thành phần:

$$x = (x_1, x_2, \ldots, x_n)$$

trong đó: $x_j = 1$ nếu đồ vật thứ j được đem theo,

$x_j = 0$ nếu ngược lại.

- Với phương án $x = (x_1, x_2, \ldots, x_n)$

- Tổng giá trị đồ vật đem theo là: $f(x) = \sum_{j=1}^{n} c_j x_j$,

- Tổng trọng lượng đồ vật đem theo là: $g(x) = \sum_{j=1}^{n} a_j x_j$

Bài toán cái túi
(Knapsack Problem)

- Bài toán cái túi có thể phát biểu dưới dạng bài toán tối ưu tổ hợp sau:

Trong số các bộ n thành phần giá trị nhị phân thoả mãn điều kiện $g(x) \le b$, hãy tìm bộ x^* cho giá trị lớn nhất của hàm mục tiêu $f(x)$, tức là

$$Tìm\ max\ \{\ f(x): x \in B^n, g(x) \le b\ \}.$$

Bài toán đóng thùng
(Bin Packing)

- Có n đồ vật với trọng lượng là w_1, w_2, ..., w_n. Cần tìm cách xếp các đồ vật này vào các cái thùng có cùng dung lượng là b sao cho số thùng cần sử dụng là nhỏ nhất có thể được.
- Ta có thể giả thiết là: $w_i \le b$, $i = 1, 2,.., n$.

Do đó số thùng cần sử dụng để chứa tất cả các đồ vật là không quá n thùng.

- Bài toán đặt ra: xác định xem mỗi một trong số n đồ vật cần được xếp vào cái thùng nào trong số n cái thùng đã mở để cho số thùng chứa đồ là ít nhất.

Bài toán đóng thùng
(Bin Packing)

- Đưa vào biến bool: $x_{ij} = 1$, nếu đồ vật i được xếp vào thùng j, bằng 0 nếu ngược lại.
- Khi đó bài toán đóng thùng có thể phát biểu dưới dạng:

$$\sum_{j=1}^{n} sign(\sum_{i=1}^{n} x_{ij}) \to \min,$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} w_i x_{ij} \le b, \quad j = 1, 2, ..., n;$$

$$x_{ij} \in \{0,1\}, \ i, j = 1, 2, ..., n.$$

Bài toán đóng thùng
(Bin Packing)

- Ví dụ:

- Có n=6 đồ vật,

- Dãy trọng lượng: 5, 1, 6, 2, 7, 3

- Dung lượng thùng b=8.

Một phương án xếp

	Thùng 1	Thùng 2	Thùng 3	Thùng 4	Thùng 5	Thùng 6
Đồ vật 1	1	0	0	0	0	0
Đồ vật 2	0	1	0	0	0	0
Đồ vật 3	0	0	0	1	0	0
Đồ vật 4	0	1	0	0	0	0
Đồ vật 5	0	0	1	0	0	0
Đồ vật 6	0	1	0	0	0	0

Số thùng cần sử dụng của phương án này:

$$\sum_{j=1}^{6} sign(\sum_{i=1}^{6} x[i,j])$$

$$= sign(1) + sign(3) + sign(1) + sign(1) + sign(0) + sign(0)$$

$$= 1+1+1+1+0+0 = 4$$

5.2 Phương pháp duyệt toàn bộ

- Một trong những phương pháp hiển nhiên để giải bài toán tối ưu tổ hợp là:
  - ✓ *Dùng một thuật toán liệt kê tổ hợp (ví dụ: thuật toán quay lui), để duyệt từng phương án của bài toán*;
  - ✓ *Với mỗi phương án, đều tính giá trị hàm mục tiêu của nó;*
  - ✓ *So sánh giá trị hàm mục tiêu tại tất cả các phương án được liệt kê để tìm ra phương án tối ưu.*
- Phương pháp xây dựng theo nguyên tắc như vậy gọi là phương pháp duyệt toàn bộ (còn gọi là *PP vét cạn*).

Ví dụ: Bài toán người du lịch

- Một người du lịch muốn đi tham quan n thành phố T_1, T_2, ..., T_n.
- *Hành trình là cách đi xuất phát từ một thành phố nào đó, đi qua tất cả các thành phố còn lại, mỗi thành phố đúng một lần, rồi quay trở lại thành phố xuất phát.*
- Biết c_{ij} là chi phí đi từ thành phố T_i đến thành phố T_j ($i, j = 1, 2,..., n$),
- **Tìm hành trình với tổng chi phí là nhỏ nhất.**

Ví dụ: Bài toán người du lịch

- Để thuận tiện trong lập trình ta đặt tên các thành phố là 1,2,...,n, thành phố xuất phát là 1.
- Dữ liệu vào từ file TSP.INP:
  - ✓ Dòng đầu ghi số n (1<=n<=20).
  - ✓ Các dòng sau ghi ma trận chi phí, gồm n dòng mỗi dòng n số: c[i,j] là chi phí đi từ thành phố i đến thành phố j, nếu không có đường đi từ i đến j thì a[i,j] = 0.

Ví dụ: Bài toán người du lịch

- Ví dụ file dữ liệu TSP.INP như sau:

```
5
0 20 35 42 25
20 0 34 30 37
35 34 0 12 19
42 30 12 0 28
25 37 19 28 0
```

Ví dụ: Bài toán người du lịch

- Phương án (cấu hình) của bài toán là bộ n thành phần:

  (x[1], x[2], …, x[n])

- Thành phố xuất phát là 1 nên gán cho: x[1] = 1,
- (x[2], …, x[n]) là hoán vị của (n-1) phần tử {2,…, n},
- x[i] = j nghĩa là bước thứ i người du lịch thăm thành phố j (với i = 1, 2,…, n), ở đó x[i] ≠ x[j] với i ≠ j;
- Chặng đường cuối, quay về đỉnh xuất phát 1 không cần biểu diễn tường minh.

Chương trình trên C++

```
#include <bits/stdc++.h>
using namespace std;
const char ginp[]="TSP.IN1";
const int maxN=100;
int n;
int c[maxN+1][maxN+1];
int x[maxN],kq[maxN];
bool cx[maxN+1];
long Min,cp;
void khoitao()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    freopen(ginp, "r", stdin);
```

```
cin>>n; //Doc du lieu vao
 for (int i=1; i<=n; i++)
     for (int j=1; j<=n; j++) cin>>c[i][j];}
//In du lieu
cout<<"So thanh pho n = "<<n<<endl;
cout<<"Ma tran chi phi: "<<endl;
for (int i=1; i<=n; i++){
   for (int j=1; j<=n; j++)
              cout<<c[i][j]<<' ';
    cout<<endl; }
cout<<endl;

for (int i=1; i<=n; i++) cx[i]=1;
x[1]=1; cx[1]=0; Min=1000000000;
cp=0;
}
```

Chương trình trên C++

```
void capnhat()
{  if (cp + c[x[n]][1] < Min)
   {Min= cp + c[x[n]][1];
    for (int i=1; i<=n; i++) kq[i]=x[i]; }
}
void Try(int i)
{ for (int j=2; j<=n; j++)
    if (cx[j]) {
       x[i]=j;
       cp=cp+c[x[i-1]][j];
       cx[j]=0;
       if (i==n) capnhat(); else Try(i+1);
       cx[j]=1;
       cp=cp-c[x[i-1]][j]; }
}
```

```
void inkq()
{
 cout<<"Hanh trinh co chi phi nho
     nhat:"<<Min<<endl;
 cout<<"Thu tu tham cac thanh pho:"<<endl;
 for (int i=1; i<=n; i++)
          cout<<kq[i]<<" --> ";
  cout<<1<<endl;
}
int main()
{
   khoitao();
   Try(2);
   inkq();
}
```

Nhận xét

- Duyệt toàn bộ là khó có thể thực hiện được ngay cả trên những máy tính điện tử hiện đại. Phương pháp này chỉ thực hiện được khi kích thước dữ liệu nhỏ.
- Cần phải có những biện pháp nhằm hạn chế việc tìm kiếm thì mới có hy vọng giải được các bài toán tối ưu tổ hợp thực tế. Để có thể đề ra những biện pháp như vậy cần phải nghiên cứu kỹ tính chất của bài toán cụ thể.
- Nhờ nghiên cứu kỹ bài toán, trong một số trường hợp cụ thể, ta có thể xây dựng những thuật toán hiệu quả để giải bài toán đặt ra.

Nhận xét

- Tuy nhiên phải nhấn mạnh rằng, trong nhiều trường hợp (ví dụ trong các bài toán người du lịch, bài toán cái túi, bài toán đóng thùng) chúng ta chưa thể xây dựng được phương pháp hữu hiệu nào khác ngoài phương pháp duyệt toàn bộ.
- **Vấn đề đặt ra:** trong quá trình liệt kê lời giải ta cần tận dụng các thông tin đã có để loại bỏ những phương án chắc chắn không phải là tối ưu.
- Trong mục tiếp theo chúng ta sẽ xét một sơ đồ tìm kiếm như vậy để giải các bài toán tối ưu tổ hợp được biết đến với tên gọi: **thuật toán nhánh cận.**

5.3 Thuật toán nhánh cận
(Branch and Bound Algorithm)

Thuật toán bao gồm hai thủ tục:

- **Phân nhánh** *(Branching Procedure):*

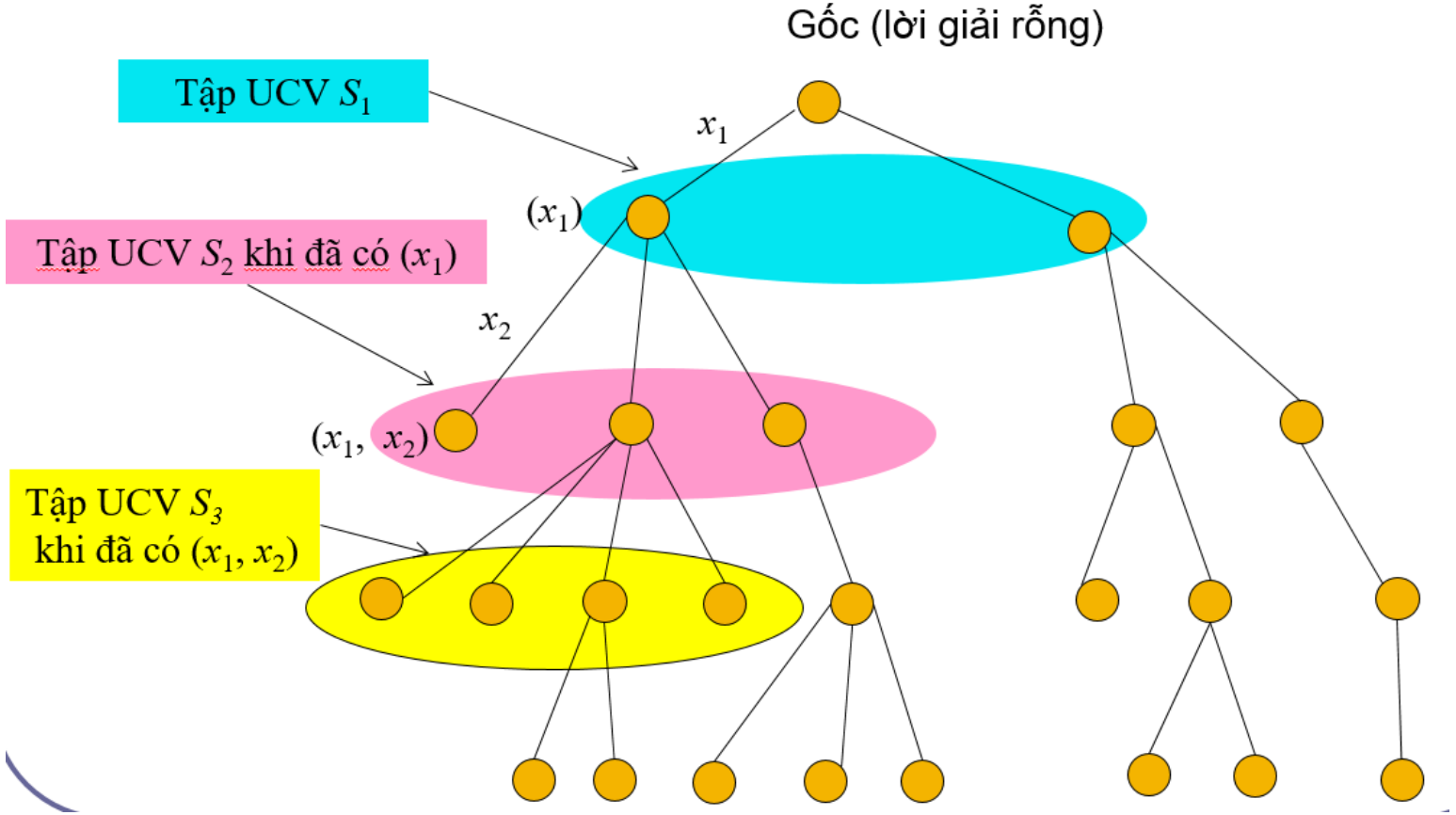
  Tổ chức duyệt nhờ thuật toán quay lui

- **Tính cận** *(Bounding Procedure)*

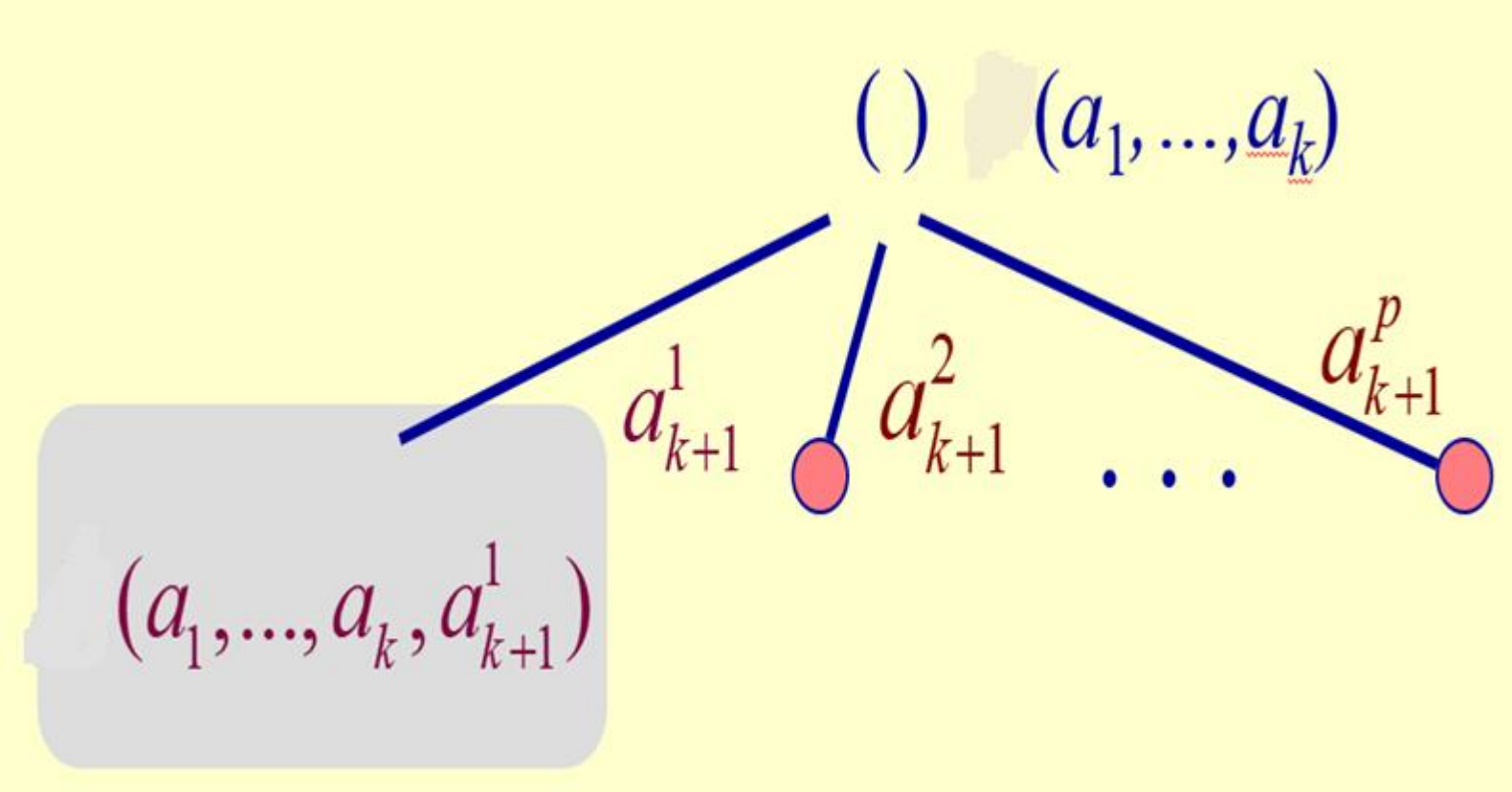
  - ✓ *Với mỗi phương án bộ phận $(a_1,a_2,\ldots,a_k)$ xác định cận của phương án, ký hiệu $g(a_1,a_2,\ldots,a_k)$. Tính cận trên/dưới là tùy bài toán cụ thể.*
  - ✓ *Dùng giá trị cận $g(a_1,a_2,\ldots,a_k)$ này để cắt bỏ những nhánh không có khả năng tìm được phương án tốt hơn.*

Phân nhánh – dựa trên thuật toán quay lui

Cây liệt kê của TT quay lui, các nhánh được phát triển từ gốc.



Phân nhánh ở bước (k+1)



Tính cận và cắt nhánh

Xét bài toán Min *(hàm mục tiêu đạt giá trị nhỏ nhất):*

- ✓ **Tính cận dưới** của phương án bộ phận $(a_1, a_2, ..., a_k)$, nhận được: $\boldsymbol{g(a_1, a_2, ..., a_k)}$.
- ✓ **Cắt nhánh:** Ký hiệu $\bar{x}$ là phương án tốt nhất hiện có và $\bar{f} = f(\bar{x})$ là giá trị hàm mục tiêu nhỏ nhất hiện có.

Nếu $\boldsymbol{g(a_1, a_2, ..., a_k) > \bar{f}}$, *(tức cận dưới của phương án bộ phận* $(a_1, a_2, ..., a_k)$ *lớn hơn gía trị hàm mục tiêu của phương án tối ưu hiện có)*

thì phương án bộ phận này phát triển tiếp cũng không nhận được phương án nào tốt hơn phương án tối ưu đang có, do vậy cắt bỏ nhánh này.

Tính cận và cắt nhánh

Tức là:

Thuật toán chỉ phát triển tiếp phương án bộ phận $(a_1, a_2, ..., a_k)$ khi:

$$g(a_1, a_2, ..., a_k) <= \bar{f}$$

Thuật toán nhánh cận

```
procedure  Branch(k);
(*   Phát triển phương án bộ phận (x1, x2, ..., xk-1)  *)
begin
    for  ak∈Ak  do
        if   ak∈Sk then
            begin
                xk :=  ak;
                if   (k == n)   then  < Cập nhật kỷ lục>
                else
                    if   g(x1,..., xk) ≤  f¯   then  Branch(k+1);
            end;
end;
```

Thuật toán nhánh cận

```
procedure  BranchAndBound;
begin
   f̄ := +∞;
    ( hoặc nếu biết p/án  x̄  nào đó thì đặt  f̄ = f( x̄ ) )
   Branch(1);
   if   f̄ <  +∞  then
            <  f̄  là giá trị tối ưu,  x̄  là p/án tối ưu >
   else  <  bài toán không có nghiệm>;
end;
```

Chú ý

❖ Nếu trong thủ tục Branch, thay câu lệnh

if (k == n) **then** < Cập nhật kỷ lục>

else

if $g(a_1,..., a_k) \leq \bar{f}$ **then** Branch(k+1);

bằng câu lệnh:

if (k == n) **then** < Cập nhật kỷ lục>

else Branch(k+1)

thì ta thu được thuật toán duyệt toàn bộ.

Chú ý

- Việc xây dựng hàm tính cận $g(a_1,..., a_k)$ phụ thuộc vào từng bài toán tối ưu tổ hợp cụ thể. Thông thường ta cố gắng xây dựng nó sao cho:
  - ✓ *Việc tính giá trị của g đơn giản.*
  - ✓ *Giá trị của cận $g(a_1,..., a_k)$ phải sát với giá trị max/min của hàm mục tiêu của phương án tối ưu.*
- Rất tiếc là hai yêu cầu này trong thực tế thường đối lập nhau.

5.4 Ví dụ áp dụng - Bài toán người du lịch

- Tìm C_{min} là giá trị nhỏ nhất trong ma trận chi phí C.
- Ở bước i: ta đã xây dựng phương án bộ phận $(x_1,x_2\ldots,x_i)$
  - ✓ Từ bước i+1 đến bước n còn n – i bước đi, sau đó thêm bước quay lại thành phố xuất phát, do vậy còn (n – i +1) chặng đường nữa.
  - ✓ Chi phí của mỗi chặng đường đó là một phần tử của ma trận C, do đó lớn hơn hoặc bằng C_{min}, vậy tổng chi phí của các bước này sẽ lớn hơn hoặc bằng $(n – i +1)*C_{min}$.
  - ✓ Cận dưới của tổng chi phí các chặng đường còn lại là

$$(n – i +1)*C_{min}$$

  - ✓ Do đó, chỉ phát triển tiếp phương án bộ phận khi:

$$\text{Chi phí đi qua i thành phố} + (n\text{-}i+1) * C_{min} < \overline{f}$$

Chương trình trên C++

```
#include <bits/stdc++.h>
using namespace std;
const char ginp[]="TSP.IN1";
const char gout[]="TSP.OUT";
const int maxN=100;
int n;
int c[maxN+1][maxN+1];
int x[maxN],kq[maxN];
bool cx[maxN+1];
long Min,cp, Cmin;
void khoitao()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    freopen(ginp, "r", stdin);
```

```
 //Doc du lieu vao
cin>>n;
for (int i=1; i<=n; i++)
     for (int j=1; j<=n; j++) cin>>c[i][j];
 //In du lieu
cout<<"So thanh pho n = "<<n<<endl;
cout<<"Ma tran chi phi: "<<endl;
for (int i=1; i<=n; i++) {
for (int j=1; j<=n; j++) out<<c[i][j]<<‘ ‘;
cout<<endl; }
cout<<endl;
for (int i=1; i<=n; i++) cx[i]=1;
x[1]=1; cx[1]=0;
Min=1000000000; cp=0;
```

Chương trình trên C++

```
//Tim PT nho nhat cua ma tran chi phi
Cmin=1000000000;
for (int i=1; i<=n; i++)
  for (int j=1; j<=n; j++)
    if (c[i][j] != 0 && c[i][j] < Cmin)
     Cmin=c[i][j];
}
void capnhat()
{
    if (cp + c[x[n]][1] < Min)
    {
        Min= cp + c[x[n]][1];
        for (int i=1; i<=n; i++) kq[i]=x[i];
    }
}
```

```
void Try(int i)
{
 for (int j=2; j<=n; j++)
  if (cx[j])
    {  x[i]=j;
       cp=cp+c[x[i-1]][j];
       cx[j]=0;
       if (i==n) capnhat();
       else
            if (cp + (n-i+1) * Cmin < Min)
                      Try(i+1);
        cx[j]=1;
       cp=cp-c[x[i-1]][j];
    }
}
```

Chương trình trên C++

```
void inkq()
{  cout<<"Hanh trinh voi chi phi nho nhat:"<<endl;
   for (int i=1; i<=n; i++) cout<<kq[i]<<" --> ";
   cout<<1<<endl;
   cout<<"Chi phi: "<<Min<<endl;
}
int main()
{
   khoitao();
   Try(2);
   inkq();
}
```

- Chạy thử nghiệm chương trình trên các bộ dữ liệu.
- So sánh thời gian thực hiện TT nhánh cận với TT duyệt toàn bộ.