Toán rời rạc

Phần thứ nhất

LÝ THUYẾT TỔ HỢP

Combinatorial Theory

Nội dung phần 1: Lý thuyết tổ hợp

Chương 1. Bài toán đếm

Chương 2. Bài toán tồn tại

Chương 3. Bài toán liệt kê tổ hợp

Chương 4. Bài toán tối ưu tổ hợp

Chương 4. BÀI TOÁN TỐI ƯU TỔ HỢP

1. **Giới thiệu bài toán**
2. Duyệt toàn bộ
3. Thuật toán nhánh cận

1. Giới thiệu bài toán

1.1. Bài toán tổng quát

1.2. Bài toán người du lịch

1.3. Bài toán cái túi

1.1. Bài toán tổng quát

- 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.

Phát biểu bài toán tối ưu tổ hợp

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

Tìm cực tiểu (hay cực đại) của hàm

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

với điều kiện

$$x \in D,$$

trong đó D là tập hữu hạn phần tử.

Các thuật ngữ:

- $f(x)$ - hàm mục tiêu của bài toán,
- $x \in D$ - phương án
- D - tập các phương án của bài toán.
- Thông thường tập D được mô tả như là tập các cấu hình tổ hợp thoả mãn một số tính chất cho trước nào đó.
- Phương án $x^* \in D$ đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là ***phương án tối ưu***, khi đó giá trị $f^* = f(x^*)$ được gọi là ***giá trị tối ưu*** của bài toán.

1. Giới thiệu bài toán

1.1. Bài toán tổng quát

1.2. Bài toán người du lịch

1.3. Bài toán cái túi

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

Ta có tương ứng 1-1 giữa một hành trình

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

với một hoán vị $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ của n số tự nhiên $1, 2, \ldots, n$.

Đặt chi phí hành trình

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

Ký hiÖu:

Π - tËp tÊt c¶ c¸c ho¸n vÞ cña n sè tù nhiªn $1, 2, \ldots, n$.

Bài toán người du lịch

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

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

- Có thể thấy rằng tổng số hành trình của người du lịch là $n!$, trong đó chỉ có $(n\text{-}1)!$ 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).

1. Giới thiệu bài toán

1.1. Bài toán tổng quát

1.2. Bài toán người du lịch

1.3. Bài toán cái túi

1.3. 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 w_j và
  - giá trị sử dụng là v_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?

Phát biểu bài toán

- Một phương án đem đồ của nhà thám hiểm có thể biểu diễn bởi vectơ nhị phân độ dài n: $x = (x_1, x_2,..., x_n)$, trong đó $x_j = 1$ nếu đồ vật thứ j được đem theo và $x_j = 0$ nếu trái lại.
- Với phương án x, giá trị đồ vật đem theo là

$$f(x) = \sum_{j=1}^{n} v_j x_j,$$

tổng trọng lượng đồ vật đem theo là

$$g(x) = \sum_{j=1}^{n} w_j x_j$$

1.3. Bài toán cái túi

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

Trong số các vectơ nhị phân độ dài n thỏa mãn điều kiện $g(x) \le b$, hãy tìm vectơ x^* cho giá trị lớn nhất của hàm mục tiêu $f(x)$:

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

Chương 4. BÀI TOÁN TỐI ƯU TỔ HỢP

1. Giới thiệu bài toán
2. **Duyệt toàn bộ**
3. Thuật toán nhánh cận

Mô tả phương pháp

- Một trong những phương pháp hiển nhiên nhất để giải bài toán tối ưu tổ hợp đặt ra là: Trên cơ sở các thuật toán liệt kê tổ hợp ta tiến hành duyệt từng phương án của bài toán, đối với mỗi phương án ta đều tính giá trị hàm mục tiêu tại nó, sau đó so sánh giá trị hàm mục tiêu tại tất cả các phương án được liệt kê để tìm ra phương án tối ưu.
- Phương pháp xây dựng theo nguyên tắc như vậy có tên gọi là phương pháp duyệt toàn bộ.

Ví dụ: Giải bài toán cái túi

- Xét bài toán cái túi biến 0-1 :

$$\max\{ f(x) = \sum_{j=1}^{n} v_j x_j : x \in D\},$$

trong đó $D = \{x = (x_1, x_2, ..., x_n) \in B^n : \sum_{j=1}^{n} w_j x_j \le b\}$

- v_j , w_j, b là các số nguyên dương, j=1,2,…, n.
- CÇn cã thuËt to¸n liÖt kª c¸c phÇn tö cña D

Thuật toán quay lui
liệt kê các phương án chất đồ

- **Xây dựng S_k:**
  - S_1={ 0, t_1 }, với t_1=1 nếu $b \geq w_1$; t_1 = 0, nếu trái lại
  - Giả sử đã có phương án (x_1, …, x_{k-1}). Khi đó:
    - Dung lượng còn lại là:

    $b_{k-1} = b - w_1x_1 - \ldots - w_{k-1}x_{k-1}$

    - Giá trị của các đồ vật chất vào túi là

    $$f_{k-1} = v_1x_1 + \ldots + v_{k-1}x_{k-1}$$

    Do đó: S_k = {0, t_k}, với t_k=1 nếu $b_{k-1} \geq w_k$; t_k = 0, nếu trái lại
- **Mô tả S_k?**

for (y = 0; y++;y<= t_k)

```
void Try(int k)
{
  <Xây dựng Sk là tập chứa các ứng cử viên cho vị trí
    thứ k của lời giải>;
  for y ∈ Sk  //Với mỗi UCV y từ Sk
  {
    ak = y;
    if (k == n) then<Ghi nhận lời giải (a1, a2, ..., ak) >;
    else Try(k+1);
    Trả các biến về trạng thái cũ;
  }
}
```

Chương trình trên C++

```
int x[20], xopt[20], v[20], w[20];
int  n,b, bk, fk, fopt;

void Nhapdl ( )
{
     <Nhập vào n, w, v, b>;
}
void Inkq ( )
{
    <Phương án tối ưu: xopt;
     Giá trị tối ưu: fopt>;
}
```

```
int main( )
{
    Nhapdl( );
    bk=b;
    fk= 0;
    fopt= 0;
    Try(1);
    InKq( );
}
```

```
void  Try(int k)
{
    int j, t;
   if (bk >= w[k])  t=1  else t=0;
   for (j= t; j>=0; j--)
   {
      x[k] = j;
       bk= bk-w[k]*x[k];
       fk= fk + v[k]*x[k];
      if  (k == n)
       {
           if (fk>fopt) {
                  xopt=x; fopt=fk;
           }
        }
       else Try(k+1);
        bk= bk+w[k]*x[k];
        fk= fk - v[k]*x[k];
   }
}
```

Bình luận

- Duyệt toàn bộ là khó có thể thực hiện được ngay cả trên những máy tính điện tử hiện đại nhất. Ví dụ để liệt kê hết

  15! = 1 307 674 368 000

  hoán vị trên máy tính điện tử với tốc độ tính toán 1 tỷ phép tính một giây, nếu để liệt kê một hoán vị cần phải làm 100 phép tính, thì ta cần một khoảng thời gian là 130767 giây > 36 tiếng đồng hồ!

  20! ===> 7645 năm

- Vì vậy cần phải có những biện pháp nhằm hạn chế việc tìm kiếm thì mới có hy vọng giải được các bài toán tối ưu tổ hợp thực tế. Tất nhiên để có thể đề ra những biện pháp như vậy cần phải nghiên cứu kỹ tính chất của bài toán tối ưu tổ hợp cụ thể.
- Nhờ những nghiên cứu như vậy, trong một số trường hợp cụ thể ta có thể xây dựng những thuật toán hiệu quả để giải bài toán đặt ra.

Bình luận

- 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.
- Khi đó, một vấn đề đặt ra là trong quá trình liệt kê lời giải ta cần tận dụng các thông tin đã tìm được để loại bỏ những phương án chắc chắn không phải là tối ưu.
- Trong mục tiếp theo chúng ta sẽ xét một sơ đồ tìm kiếm như vậy để giải các bài toán tối ưu tổ hợp mà trong tài liệu tham khảo được biết đến với tên gọi: thuật toán nhánh cận.

Chương 4. BÀI TOÁN TỐI ƯU TỔ HỢP

1. Giới thiệu bài toán
2. Duyệt toàn bộ
3. **Thuật toán nhánh cận**

3. Thuật toán nhánh cận

3.1. Sơ đồ chung

3.2. Ví dụ bài toán người du lịch

3.1. Sơ đồ chung

- Thuật toán bao gồm hai thủ tục:
  - Phân nhánh (Branching Procedure)
  - Tính cận (Bounding Procedure)
- **Phân nhánh:** *Quá trình phân hoạch tập các phương án ra thành các tập con với kích thước càng ngày càng nhỏ cho đến khi thu được phân hoạch tập các phương án ra thành các tập con một phần tử*
- **Tính cận:** *Cần đưa ra cách tính cận cho giá trị hàm mục tiêu của bài toán trên mỗi tập con A trong phân hoạch của tập các phương án.*

3.1. Sơ đồ chung

- Ta sẽ mô tả tư tưởng của thuật toán trên mô hình bài toán tối ưu tổ hợp tổng quát sau

$$\min \{ f(x) : x \in D \},$$

trong đó D là tập hữu hạn phần tử.

- Giả thiết rằng tập D được mô tả như sau

$$D = \{x = (x_1, x_2, ..., x_n) \in A_1 \times A_2 \times ... \times A_n:$$
$$x \text{ thoả mãn tính chất } P\},$$

với $A_1, A_2, ..., A_n$ là các tập hữu hạn, còn P là tính chất trên tích Đề các $A_1 \times A_2 \times ... \times A_n$.

Nhận xét

- Yêu cầu về mô tả của tập D là để có thể sử dụng thuật toán quay lui để liệt kê các phương án của bài toán.
- Bài toán

$$\max \{f(x): x \in D\}$$

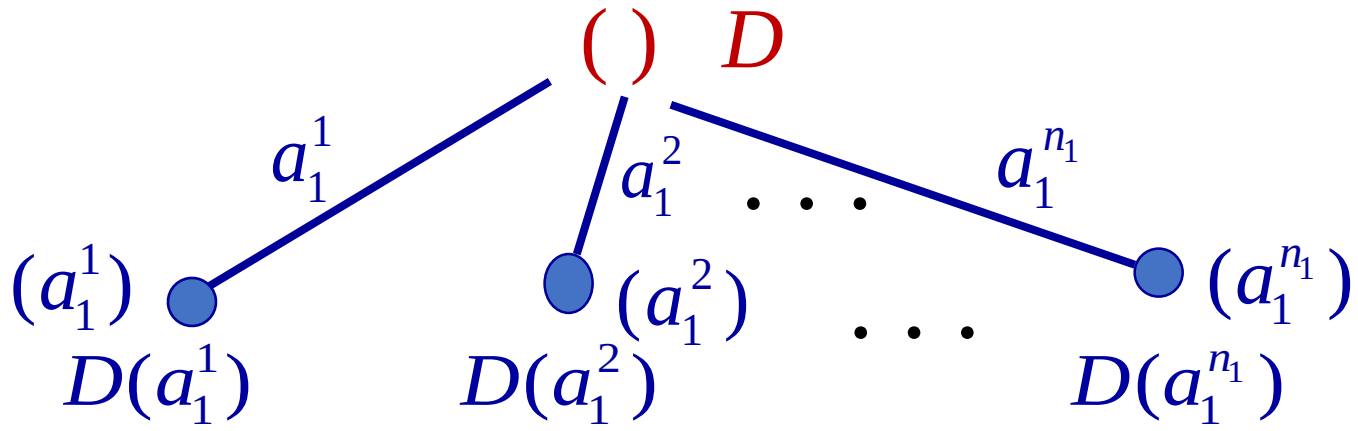
là tương đương với bài toán

$$\min \{g(x): x \in D\}, \text{ trong đó } g(x) = -f(x)$$

Do đó ta có thể hạn chế ở việc xét bài toán min.

Phân nhánh

Quá trình phân nhánh được thực hiện nhờ thuật toán quay lui:



trong ®ã $D(a_1^i) = \{x \in D : x_1 = a_1^i\}, i = 1, 2, ..., n_1$

lµ tËp c¸c ph-¬ng ¸n cã thÓ ph¸t triÓn tõ pabp (a_1^i)

Ta có phân hoạch:

$$D = D(a_1^1) \cup D(a_1^2) \cup ... \cup D(a_1^{n_1})$$

Phân nhánh

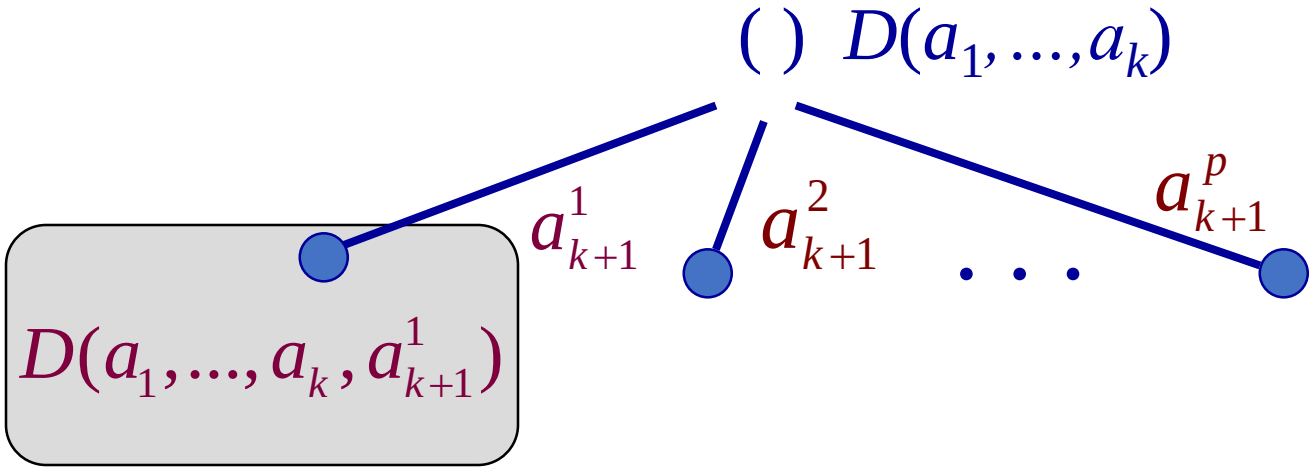
- Như vậy ta có thể đặt tương ứng mỗi phương án bộ phận (a_1, a_2, ..., a_k) với một tập con các phương án của bài toán:

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

- Ở bước tổng quát của thuật toán quay lui ta sẽ làm việc với phương án bộ phận (a_1, a_2, ..., a_k) và xét các cách tiếp tục phát triển phương án này.
- Điều đó tương đương với việc phân hoạch tập D ra thành các tập con nhỏ hơn.

Phân nhánh

- Quá trình phân nhánh có thể diễn tả như sau:



- Ta cã ph©n ho¹ch:

$$D(a_1,\ldots,a_k)=\bigcup_{i=1}^{p} D(a_1,\ldots,a_k,a_{k+1}^i)$$

Tính cận

- Cần có hàm g xác định trên tập tất cả các phương án bộ phận của bài toán thoả mãn bất đẳng thức sau:

Giá trị hàm mục tiêu của mọi lời giải có k thành phần đầu tiên là $(a_1, a_2,\ldots,a_k)$

$$g(a_1,..., a_k) \le \min\{f(x): x \in D(a_1, ..., a_k)\} \quad (*)$$

với mỗi phương án bộ phận cấp k $(a_1, a_2, ..., a_k)$, và với mọi $k = 1, 2, ...$

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

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

hay nói một cách khác, $g(a_1, a_2, \ldots, a_k)$ là **cận dưới** của giá trị hàm mục tiêu trên tập $D(a_1, a_2, ..., a_k)$.

Cắt nhánh nhờ sử dụng cận dưới

- Giả sử đã có hàm g. Ta xét cách sử dụng hàm này để giảm bớt khối lượng duyệt trong quá trình duyệt tất cả các phương án theo thuật toán quay lui.
- Trong quá trình liệt kê các phương án có thể đã thu được một số phương án của bài toán. Gọi x^* là phương án với giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã tìm được, ký hiệu $f^* = f(x^*)$
- Ta sẽ gọi
  - x^* là phương án tốt nhất hiện có (phương án kỷ lục),
  - f^* là giá trị tốt nhất hiện có (giá trị kỷ lục).

Cắt nhánh nhờ sử dụng cận dưới

Giả sử đã có f^* (*giá trị tốt nhất hiện có*), khi đó nếu

$$g(a_1, a_2, ..., a_k) > f^*$$

thì từ bất đẳng thức (*) suy ra

$$f^* < g(a_1,..., a_k) \leq \min\{f(x): x \in D(a_1,...,a_k)\},$$

Vì thế tập $D(a_1,..., a_k)$ chắc chắn không chứa phương án tối ưu và có thể loại bỏ khỏi quá trình duyệt.

Thuật toán nhánh cận

```
void Try(int k)
{
    //Xây dựng x_k từ phương án bộ phận (x_1, x_2, ..., x_{k-1})
   for  a_k∈A_k
      if   (a_k∈S_k )
      {
           x_k =  a_k;
           if   (k == n) then < Cập nhật kỷ lục>;
           else if   (g(x_1,..., x_k)  ≤ f*) Try(k+1);
       }
}

void  BranchAndBound ( )
{
    f* = +∞;
   // Nếu biết p/án x* nào đó thì đặt f* = f(x*)
   Try(1);
   if  (f*  <  +∞)
           <f* là giá trị tối ưu, x* là p/án tối ưu >
   else  <  bài toán không có phương án >;
}
```

Chú ý rằng nếu thủ tục **Try** ta thay câu lệnh

```
if   (k == n) then < Cập nhật kỷ lục>;
else if   (g(x_1,..., x_k)  ≤ f*) Try(k+1);
```

bởi

```
if   (k == n) then < Cập nhật kỷ lục>;
else Try(k+1);
```

thì ta thu được thuật toán **duyệt toàn bộ.**

Chú ý:

$$g(a_1,..., a_k) \le \min\{f(x): x \in D(a_1, ..., a_k)\} \quad (*)$$

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

3. Thuật toán nhánh cận

3.1. Sơ đồ chung

3.2. Ví dụ Bài toán người du lịch

Sir William Rowan Hamilton
1805 - 1865

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

- Một người du lịch muốn đi tham quan n thành phố 1,2,..,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 i đến thành phố j (i, j = 1, 2,..., n),
- Tìm hành trình với tổng chi phí là nhỏ nhất.

Bài toán người du lịch

Cố định thành phố xuất phát là 1, bài toán người du lịch dẫn về bài toán:

- Tìm cực tiểu của hàm

  $f(1,x_2,..., x_n) = c[1,x_2]+c[x_2,x_3]+...+c[x_{n-1},x_n] + c[x_n,1]$

  với điều kiện

  $(1, x_2, x_3, ..., x_n)$ là hoán vị của các số 1, 2, ..., n.

Cài đặt

```
void  Try(int k)
{
    for (int v = 2; v<=n;v++) {
       if (visited[v] == FALSE) {
              x_k = v; visited[v] = TRUE;
              f = f + c(x_{k-1},x_k);
              if (k == n) //Cập nhật kỉ lục:
              {  if (f + c(x_n,x_1) < f*) f* = f + c(x_n,x_1); }
              else Try(k+1);
               f = f - c(x_{k-1},x_k);
              visited[v] = FALSE;
        }//end if
    }//end for
}
```

```
void Try(int k)
{
  <Xây dựng S_k là tập chứa các ứng cử viên cho vị trí
    thứ k của lời giải>;
  for y ∈ S_k  //Với mỗi UCV y từ S_k
  {
     a_k = y;
     if (k == n) then<Ghi nhận lời giải (a_1, a_2, ..., a_k) >;
     else Try(k+1);
     Trả các biến về trạng thái cũ;
  }
}
```

Hàm cận dưới

- Ký hiệu

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

  là chi phí đi lại nhỏ nhất giữa các thành phố.

- Cần đánh giá cận dưới cho phương án bộ phận $(1, u_2, ..., u_k)$ tương ứng với hành trình bộ phận đã đi qua k thành phố:

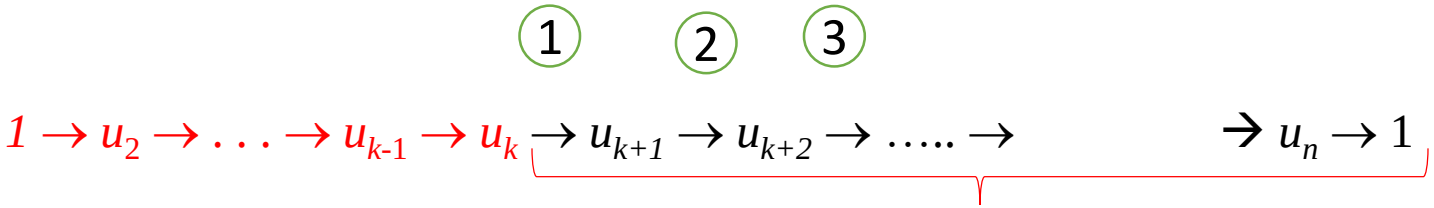
  $$1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_{k-1} \rightarrow u_k.$$

Hàm cận dưới

- Chi phí phải trả theo hành trình bộ phận này là

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

- Để phát triển thành hành trình đầy đủ:



ta còn phải đi qua n-k+1 đoạn đường nữa, mỗi đoạn có chi phí không ít hơn c_{min}, nên cận dưới cho phương án bộ phận $(1, u_2, ..., u_k)$ có thể tính theo công thức:

$$g(1, u_2, ..., u_k) = \sigma + (n\text{-}k+1)\, c_{min}\,.$$

Cài đặt

```
void  BranchAndBound()
{
    f* = +∞; cmin = min{c_ij : 1 ≤i,j ≤n}
    for (v = 1; v<=n;v++) visited[v] = FALSE;
    f=0; x_1 = 1; visited[x_1] = TRUE;
    Try(2);
    return f*;
}
```

```
void  BranchAndBound ( )
{
    f* = +∞;
    // Nếu biết p/án x* nào đó thì đặt f* = f(x*)
    Try(1);
    if  (f*  <  +∞)
            <f* là giá trị tối ưu, x* là p/án tối ưu >
   else  <  bài toán không có phương án >;
}
```

Cài đặt

```
void  Try(int k)
{
    for (int v = 1; v<=n;v++) {
       if (visited[v] == FALSE) {
              x_k = v; visited[v] = TRUE;
              f = f + c(x_{k-1},x_k);
              if (k == n) //Cập nhật kỉ lục:
              {  if (f + c(x_n,x_1) < f*) f* = f + c(x_n,x_1); }
              else {
                     g = f + (n-k + 1)*cmin; //tính cận
                     if (g < f* ) Try(k + 1);
                   }
               f = f - c(x_{k-1},x_k);
               visited[v] = FALSE;
        }//end if
    }//end for
}
```

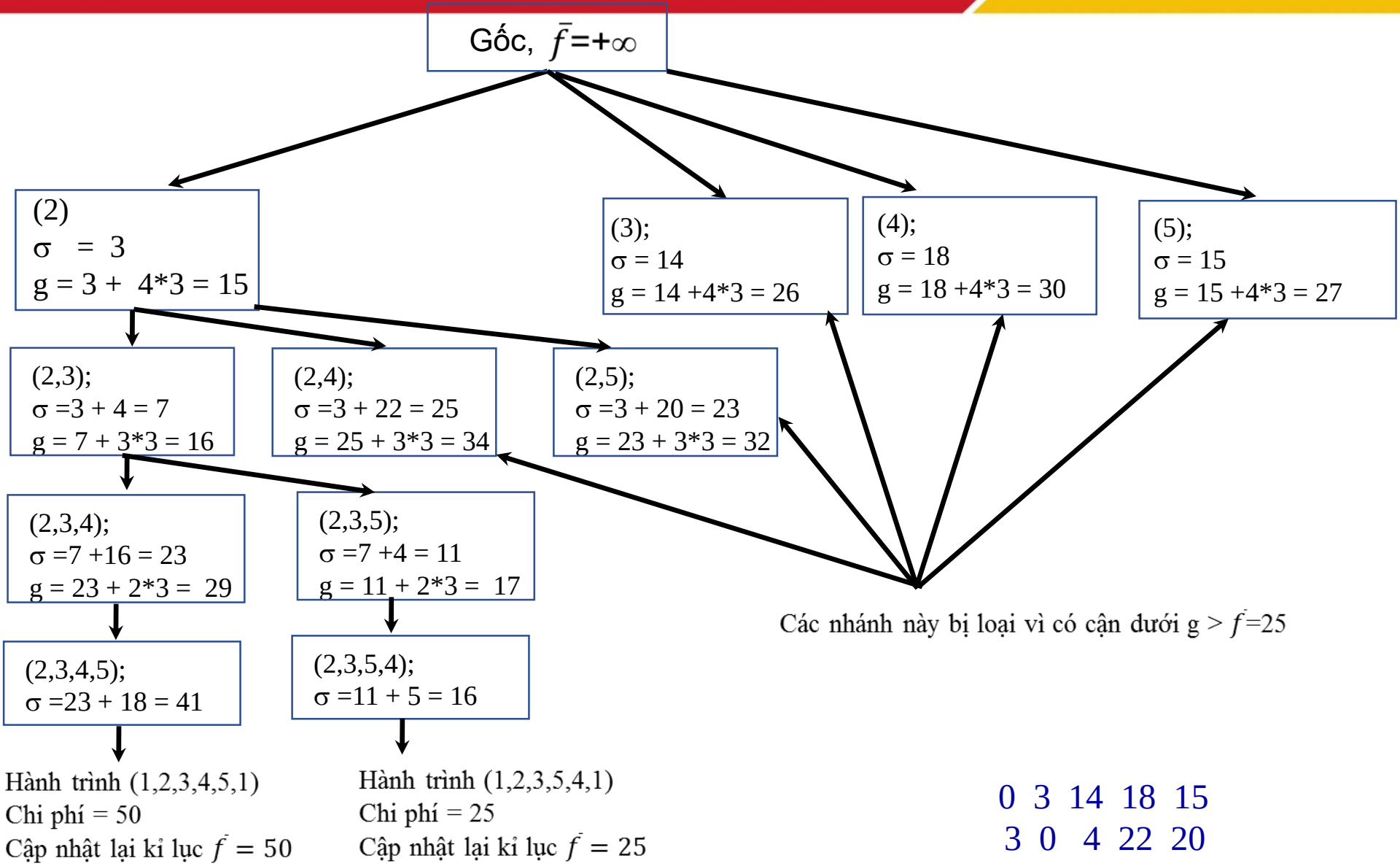
```
void Try(int k)
{
    //Xây dựng x_k từ phương án bộ phận (x_1, x_2, ..., x_{k-1})
    for  a_k∈A_k
       if   (a_k∈S_k )
       {
          x_k =  a_k;
          if   (k == n) then < Cập nhật kỷ lục>;
          else if   (g(x_1,..., x_k)  ≤ f*) Try(k+1);
       }
}
```

Ví dụ

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

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

- Ta có c_{min} = 3. Quá trình thực hiện thuật toán được mô tả bởi cây tìm kiếm lời giải.
- Thông tin được ghi trong các ô trên hình vẽ theo thứ tự sau:
  - các thành phần của phương án bộ phận,
  - σ là chi phí theo hành trình bộ phận,
  - g – cận dưới của phương án bộ phận.



Hành trình (1,2,3,4,5,1)
Chi phí = 50
Cập nhật lại kỉ lục $\bar{f}$ = 50

Hành trình (1,2,3,5,4,1)
Chi phí = 25
Cập nhật lại kỉ lục $\bar{f}$ = 25

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

Kết quả

Kết thúc thuật toán, ta thu được:

- phương án tối ưu (1, 2, 3, 5, 4, 1) tương ứng với hành trình

$$1 \to 2 \to 3 \to 5 \to 4 \to 1,$$

- chi phí nhỏ nhất là 25.