Đệ quy, Quay lui, Nhánh cận

Khoa Công Nghệ Thông Tin,
Trường Đại Học Thủy Lợi.

Ngày 23 tháng 11 năm 2017

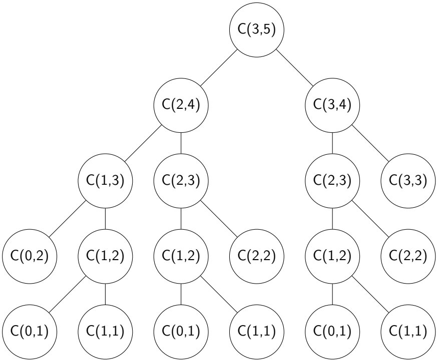
Hàm đệ quy

- Một hàm mà trong nội dung hàm có lời gọi đến chính nó
- Các trường hợp cơ bản của hàm đệ quy: kết quả được tính toán một cách tầm thường

```
int fact(int n){
  if(n <= 1) return 1;
  return n*fact(n-1);
}
int C(int k, int n){
  if(k == n || k == 0) return 1;
  return C(k-1,n-1) + C(k,n-1);
}
```

Đệ quy và Bộ nhớ

- Các hàm với cùng tham số có thể được gọi nhiều lần.
- Một hàm, với một tập cho trước các tham số được kích hoạt lần đầu, được thực thi, và kết quả được lưu trữ trong bộ nhớ.
- Sau đó, nếu hàm đó với cùng tập tham số đã được kích hoạt trước đó, nó sẽ không được thực thi. Thay vào đó, kết quả của hàm này, đã sẵn có trong bộ nhớ, sẽ được trả lại trực tiếp.



Đệ quy và Bộ nhớ

```
public class Ckn {
  private int[][] M;
  public int C(int k, int n){
    if(k == 0 || k == n) M[k][n] = 1;
    else if(M[k][n] < 0){
      M[k][n] = C(k-1,n-1) + C(k,n-1);
    }
    return M[k][n];
  }
  public void test(){
    M = new int[100][100];
    for(int i = 0; i < 100; i++)
      for(int j = 0; j < 100; j++)
        M[i][j] = -1;

    System.out.println(C(15,30));
  }
}
```

Giới thiệu

- Liệt kê tất cả cấu hình thỏa mãn những rằng buộc cho trước
  - hoán vị
  - tập con của một tập cho trước
  - etc.
- $A_1, \ldots, A_n$ là những tập con xác định và $X = \{(a_1, \ldots, a_n) \mid a_i \in A_i, \forall 1 \le i \le n\}$
- $\mathcal{P}$ là một thuộc tính trên X
- Sinh ra tất cả cấu hình $(a_1, \ldots, a_n)$ có $\mathcal{P}$

Giới thiệu

- Trong nhiều trường hợp, liệt kê là cách cuối cùng để giải quyết một vài vấn đề tổ hợp
- Hai phương thức chung:
  - Phương thức sinh (không xem xét trong bài học)
  - Thuật toán quay lui

Thuật toán quay lui

Xây dựng những thành phần của cấu hình từng bước một

- Khởi tạo: cấu hình được xây dựng là *null*
- Bước 1:
  - Tính toán (dựa trên $\mathcal{P}$) một tập S_1 các khả năng cho vị trí đầu tiên của cấu hình,
  - Chọn một phần tử của tập S_1 và đặt nó vào vị trí đầu tiên.

Thuật toán quay lui

SAMSUNG

Tại bước k: giả sử chúng ta có một phần cấu hình $a_1, \ldots, a_{k-1}$

- Tính toán (dựa trên $\mathcal{P}$) một tập S_k các khả năng cho vị trí thứ k của cấu hình,
  - Nếu $S_k \neq \emptyset$, chọn một phần tử của tập S_k và đặt nó vào vị trí thứ k và nhận được $(a_1, \ldots, a_{k-1}, a_k)$
    - Nếu $k = n$, xử lý cấu hình hoàn thiện $(a_1, \ldots, a_n)$
    - Ngược lại, xây dựng phần tử thứ $k + 1$ cho cấu hình hiện tại theo cùng sơ đồ
  - Nếu $S_k = \emptyset$, quay lui để thử với phần tử khác a'_{k-1} cho vị trí $k - 1$
    - Nếu a'_{k-1} tồn tại, đặt nó vào vị trí thứ $k - 1$
    - Ngược lại, quay lui để thử với phần tử khác cho vị trí thứ $k - 2$, ...

Thuật toán quay lui

Algorithm 1: TRY(k)

Construct a candidate set S_k;
foreach $y \in S_k$ **do**
    $a_k \leftarrow y$;
    if $(a_1, \ldots, a_k)$ *is a complete configuration* **then**
        ProcessConfiguration$(a_1, \ldots, a_k)$;
    else
        TRY$(k+1)$;

Algorithm 2: Main()

TRY(1);

Thuật toán quay lui - chuỗi nhị phân

- Một cấu hình được biểu diễn bởi $b_1, b_2, \ldots, b_n$
- Các khả năng cho b_i là $\{0, 1\}$

Thuật toán quay lui - chuỗi nhị phân

```java
public class ListingBinary {
  private int[] a;
  private int n;
  private void TRY(int i){
    for(int v = 0; v <= 1; v++){
      a[i] = v;
      if(i == n-1){
        for(int j = 0; j < n; j++) System.out.print(a[j]);
        System.out.println();
      }else{
        TRY(i+1);
      }
    }
  }
  public void list(int n){
    this.n = n;
    a = new int[n];
    TRY(0);
  }
  public static void main(String[] args) {
    ListingBinary LB = new ListingBinary();
    LB.list(4);
  }

}
```

Thuật toán quay lui - Tổ hợp

- Một cấu hình được biểu diễn bởi $(c_1, c_2, \ldots, c_k)$
  - Đặt $c_0 = 1$
  - Các khả năng cho c_i dựa trên nhận biết về các giá trị ở trước $\langle c_1, c_2, \ldots, c_{i-1} \rangle$: $c_{i-1} + 1 \leq c_i \leq n - k + i, \forall i = 1, 2, \ldots, k$

Thuật toán quay lui - Tổ hợp

Algorithm 3: TRY(i)

foreach $v = c_{i-1} + 1, \ldots, n - k + i$ **do**
- $c_i \leftarrow v$;
- **if** $i == k$ **then**
  - printConfiguration();
- **else**
  - TRY($i + 1$);

Algorithm 4: MainCombinationGeneration(n, k)

$c_0 \leftarrow 0$;
TRY(1);

Thuật toán quay lui - Tổ hợp

```java
public class ListingCombination {
  private int[] a;
  private int k;
  private int n;
  private void TRY(int i){
    for(int v = a[i-1]+1; v <= n-k+i; v++){
      a[i] = v;
      if(i == k){
        for(int j = 1; j <= k; j++) System.out.print(a[j] + " ");
        System.out.println();
      }else
        TRY(i+1);
    }
  }
  public void list(int k, int n){
    this.k = k; this.n = n;
    a = new int[k+1];
    a[0] = 0;
    TRY(1);
  }
  public static void main(String[] args) {
    ListingCombination LC = new ListingCombination();
    LC.list(3, 5);
  }

}
```

Thuật toán quay lui - Hoán vị

- Một cấu hình: $p_1, p_2, \ldots, p_k$
- Các khả năng cho p_i dựa trên nhận biết về các giá trị ở trước $\langle p_1, p_2, \ldots, p_{i-1} \rangle$: $\{1, 2, \ldots, n\} \setminus \{p_1, p_2, \ldots, p_{i-1}\}$
- Sử dụng một mảng boolean để đánh dấu những giá trị được sử dụng $b_1, b_2, \ldots, b_n$
  - $b_v = 1$, nếu giá trị v đã được sử dụng (xuất hiện trong $p_1, p_2, \ldots, p_{i-1}$)
  - $b_v = 0$, nếu ngược lại

Thuật toán quay lui - Hoán vị

Algorithm 5: TRY(i)

foreach $v = 1, \ldots, n$ **do**
  if $visited[v] = FALSE$ **then**
    $p_i \leftarrow v$;
    $visited[v] \leftarrow TRUE$;
    if $i == n$ **then**
      printConfiguration();
    else
      TRY($i + 1$);
    $visited[v] \leftarrow FALSE$;

Algorithm 6: MainPermutationGeneration(n, k)

foreach $v = 1, \ldots, n$ **do**
  $visited[v] \leftarrow FALSE$;
TRY(1);

Thuật toán quay lui - Hoán vị

```
public class ListingPermutation {
  private int[] a;
  private boolean[] visited;
  private int n;
  private void print(){
    for(int i = 1; i <= n; i++)  System.out.print(a[i] + "  ");
    System.out.println();
  }
  private void TRY(int i){
    for(int v = 1; v <= n; v++){
      if(!visited[v]){
        a[i] = v;
        visited[v] = true;
        if(i == n)
          print();
        else
          TRY(i+1);
        visited[v] = false;
      }
    }
  }
  [...]
}
```

Thuật toán quay lui - Hoán vị

```
public class ListingPermutation {
  [...]
  public void list(int n){
    this.n = n;
    a = new int[n+1];
    visited = new boolean[n+1];
    for(int v = 1; v <= n; v++)
    visited[v] = false;
    count  = 0;
    TRY(1);
  }
  public static void main(String[] args) {
    ListingPermutation LP = new ListingPermutation();
    LP.list(4);;
  }

}
```

Thuật toán quay lui - Phương trình số nguyên tuyến tính

Giải quyết các phương trình tuyến tính trong một tập các số nguyên dương

$$x_1 + x_2 + \cdots + x_n = M$$

với $(a_i)_{1 \leq i \leq n}$ và M là các số nguyên dương

- Giải pháp từng phần $(x_1, x_2, \ldots, x_{k-1})$
- $m = \sum_{i=1}^{k-1} x_i$
- $A = n - k$
- $\overline{M} = M - m - A$
- Các khả năng của x_k là $\{v \in \mathbb{Z} \mid 1 \leq v \leq \overline{M}\}$

Algorithm 7: TRY(i)

if $i = n$ **then**
  $\overline{M} \leftarrow M - f$;
  $\underline{M} \leftarrow M - f$;
else
  $\overline{M} \leftarrow M - f - (n - i)$;
  $\underline{M} \leftarrow 1$;

foreach $v = \underline{M}, \ldots, \overline{M}$ **do**
  $x_i \leftarrow v$;
  $f \leftarrow f + v$;
  if $i == n$ **then**
    printConfiguration();
  else
    TRY($i + 1$);
  $f \leftarrow f - v$;

Algorithm 8: MainLinearEquation(n, M)

$f \leftarrow 0$;
TRY(1);

Thuật toán quay lui - Bài toán n-quân-hậu

- Bài toán: Đặt n quân hậu trên một bàn cờ sao cho 2 quân hậu bất kỳ không thể tấn công nhau
- Mô hình giải pháp: $(x_1, x_2, \ldots, x_n)$ với x_i biểu diễn giá trị hàng mà quân hậu ở cột i đang đứng
- Các rằng buộc:
  - $x_i \neq x_j, \forall 1 \leq i < j \leq n$
  - $|x_i - x_j| \neq |i - j|, \forall 1 \leq i < j \leq n$

Algorithm 9: Candidate(v, i)

```
foreach j = 1, ..., i − 1 do
    if x_j = v ∨ |x_j − v| = |j − i| then
        return FALSE;
    return TRUE;
```

Algorithm 10: TRY(i)

```
foreach v = 1, ..., n do
    if Candidate(v, i) then
        x_i ← v;
        if i == n then
            Solution();
        else
            TRY(i + 1);
```

Algorithm 11: MainQueen(n)

```
TRY(1);
```

Thuật toán quay lui - Bài toán n-quân-hậu - tinh chỉnh

- Sử dụng mảng để đánh dấu những ô bị cấm
  - $r[1..n]$: $r[i] = \mathit{false}$ nếu những ô trên cột i bị cấm
  - $d_1[1-n..n-1]$: $d_1[q] = \mathit{false}$ nếu những ô (r, c) sao cho $c - r = q$ bị cấm
    - ★ trong Java, chỉ số của các phần tử trong mảng không thể là số nguyên âm (chỉ số là 0, 1, ...). Do đó có một sự dịch chuyển: $d_1[q+n-1]$ thay cho $d_1[q]$
  - $d_2[2..2n-2]$: $d_2[q] = \mathit{false}$ nếu những ô (r, c) sao cho $r + c = q$ bị cấm

Thuật toán quay lui - Bài toán n-quân-hậu - tinh chỉnh

SAMSUNG

Algorithm 12: TRY(i)

foreach $v = 1, \ldots, n$ **do**
- **if** $r[v] \wedge d_1[i - v] \wedge d_2[i + v]$ **then**
  - $x_i \leftarrow v$;
  - $r[v] \leftarrow$ FALSE;
  - $d_1[i - v] \leftarrow$ FALSE;
  - $d_2[i + v] \leftarrow$ FALSE;
  - **if** $i = n$ **then**
    - Solution();
  - **else**
    - TRY($i + 1$);
  - $r[v] \leftarrow$ TRUE;
  - $d_1[i - v] \leftarrow$ TRUE;
  - $d_2[i + v] \leftarrow$ TRUE;

Thuật toán quay lui - Bài toán n-quân-hậu - tinh chỉnh

Algorithm 13: MainQueenRefine(n)

foreach $v = 1, \ldots, n$ **do**
  $r[v] \leftarrow$ TRUE;

foreach $v = 1 - n, \ldots, n - 1$ **do**
  $d_1[v] \leftarrow$ TRUE;

foreach $v = 2, \ldots, 2n$ **do**
  $d_2[v] \leftarrow$ TRUE;

TRY(1);

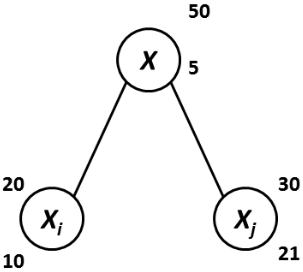
Bài toán tối ưu tổ hợp

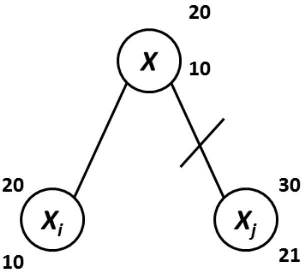
- $z = \min \{f(x) : x \in X\}$
- Ứng dụng
  - Định tuyến xe
  - Lập lịch
  - Lập kế hoạch (Timetabling)
  - Bài toán đóng gói (Bin Packing)
  - Cấp phát tài nguyên
  - ...

Sơ đồ chung của Nhánh cận

- Nhánh cận chia bài toán đang xét thành các bài toán nhỏ hơn cho đến khi chúng được giải quyết một cách dễ dàng (Phân nhánh)
  - X được chia ra thành các tập con $X_1 \ldots, X_k (k \geq 2)$ sao cho $\bigcup_{i=1,\ldots,k} X_i = X$
  - Ứng dụng đệ quy của việc chia nhỏ định nghĩa một cấu trúc cây: *cây tìm kiếm* (mỗi nút là một cập con của X)
- Thông thường, kích thước của cây tìm kiếm là rất lớn
- Cắt nhánh (Bounding)
  - Với mỗi tập $X_i (\forall i = 1, \ldots, k)$
    - ★ $z^i = \min \{f(x) : x \in X_i\}$
    - ★ tính toán $\underline{z}^i$ và $\overline{z}^i$ tương ứng là giới hạn dưới và giới hạn trên của z^i: $\underline{z}^i \leq z^i \leq \overline{z}^i$
  - Nếu tồn tại $i \neq j$ sao cho $\overline{z}^i \leq \underline{z}^j$, thì tập X_j có thể được xóa khỏi không gian tìm kiếm do $z^j \geq z^i$ (không cần thiết phải duyệt X_j)
  - Giả sử rằng z^* là giải pháp tốt nhất tìm được vào thời điểm hiện tại. Nếu $\underline{z}^i \geq z^*$, thì X_i có thể được xóa đi (không cần thiết phải duyệt X_i do $z^* \leq \underline{z}^i \leq z^i$)

Sơ đồ chung của Nhánh cận - Ví dụ





Sơ đồ chung của Thuật toán Nhánh cận (bài toán cực tiểu)

Algorithm 14: TRY(k)

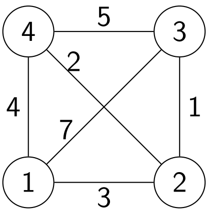
Xây dựng một tập ứng viên S_k;
foreach $y \in S_k$ **do**
  $a_k \leftarrow y$;
  if $(a_1, \ldots, a_k)$ *là một cấu hình hoàn thiện* **then**
    if $f(a_1, \ldots, a_k) < z^*$ **then**
      $z^* \leftarrow f(a_1, \ldots, a_k)$;
  else
    if $\underline{z}(a_1, \ldots, a_k) < z^*$ **then**
      TRY$(k+1)$;

Algorithm 15: Main()

$z^* \leftarrow +\infty$;
TRY(1);

Bài toán Người bán hàng

- Cho một danh sách n thành phố và khoảng cách giữa từng cặp thành phố
- Tìm một lộ trình ngắn nhất để thăm mỗi thành phố chính xác một lần và trở về thành phố ban đầu
- $x = (x_1, \ldots, x_n)$, lộ trình là $x_1 \to x_2 \to \cdots \to x_n \to x_1$
- $f(x) = c(x_1, x_2) + c(x_2, x_3) + \cdots + c(x_n, x_1)$



$$\Rightarrow c = \begin{pmatrix} 0 & 3 & 7 & 4 \\ 3 & 0 & 1 & 2 \\ 7 & 1 & 0 & 5 \\ 4 & 2 & 5 & 0 \end{pmatrix}$$

Bài toán Người bán hàng - Nhánh cận cơ bản

- Bài toán con
  - Tương ứng với một tiền tố của giải pháp: $x_1, x_2, ..., x_k$
  - Giới hạn dưới (cận dưới):
    $\underline{z}(x_1, ..., x_k) = c(x_1, x_2) + ... + c(x_{k-1}, x_k) + (n - k + 1) * cmin$ với $cmin$ là phần tử nhỏ nhất trong ma trận khoảng cách (không tính các phần tử thuộc đường chéo)
  - Thủ tục đệ quy **extend**($\langle x_1, ..., x_{k-1} \rangle$) sẽ mở rộng lời giải bộ phận

Bài toán Người bán hàng - Nhánh cận cơ bản

Algorithm 16: TRY(k)

Input: k: chỉ số của thành phố thứ k^{th} được thăm
$n, c, x, f^*, f, visited$ là các biến toàn cục
Output: Mở rộng lời giải cục bộ hiện tại $x_1, \ldots, x_{k-1}$ bằng cách gán một giá trị cho x_k

```
foreach v = 1, ..., n do
    if visited[v] = FALSE then
        x_k ← v;
        visited[v] ← TRUE;
        f ← f + c(x_{k-1}, x_k);
        if k = n then
            if f + c(x_n, x_1) < f* then
                f* ← f + c(x_n, x_1);
        else
            z ← f + (n − k + 1) * cmin;
            if z < f* then
                TRY(k + 1);
        f ← f − c(x_{k-1}, x_k);
        visited[v] ← FALSE;
```

Bài toán Người bán hàng - Nhánh cận cơ bản

Algorithm 17: MainSimpleBBTSP(n, c)

Input: n, c: số lượng thành phố n và ma trận khoảng cách c
$x, f^*, f, visited$ được khởi tạo như các biến toàn cục
Output: Chiều dài của hành trình ngắn nhất
foreach $v = 1, \ldots, n$ **do**
  $visited[v] \leftarrow FALSE$;

$f^* \leftarrow \infty$;
$f \leftarrow 0$;
$x_1 \leftarrow 1$;
$visited[x_1] \leftarrow TRUE$;
TRY(2);
return f^*;

Bài toán Người bán hàng - Nhánh cận cơ bản

```
public class TSP {
  private int[][] c;
  private int cmin;
  private int n;
  private int[] x;
  private int f;
  private int[] x_best;
  private int f_best;
  private boolean[] visited;
  [...]
}
```

Bài toán Người bán hàng - Nhánh cận cơ bản

```
public class TSP {
  public void readData(String fn){
    try{
      Scanner in = new Scanner(new File(fn));
      n = in.nextInt();
      c = new int[n][n];
      cmin = 100000000;
      for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++){
          c[i][j] = in.nextInt();
          if(i != j && cmin > c[i][j]) cmin = c[i][j];
        }

    }catch(Exception ex){
      ex.printStackTrace();
    }
  }

}
```

Bài toán Người bán hàng - Nhánh cận cơ bản

```java
public class TSP {

  private void updateBest(){
    if(f + c[x[n-1]][x[0]] < f_best){
      f_best = f + c[x[n-1]][x[0]];
      System.arraycopy(x,0,x_best,0,x.length);
      System.out.print("Update Best, new  best: ");
      printBest();
    }
  }
}
```

Bài toán Người bán hàng - Nhánh cận cơ bản

```java
public class TSP {
  private void extend(int i){
    int v;
    for(v = 0; v < n; v++){
      if(!visited[v]){
        x[i] = v;
        visited[v] = true;
        f += c[x[i-1]][x[i]];
        if(i == n-1){
          updateBest();
        }else{
          int g = f += cmin*(n-i);
          if(g < f_best)
            extend(i+1);
        }
        f -= c[x[i-1]][x[i]];
        visited[v] = false;
      }
    }
  }
}
```

Bài toán Người bán hàng - Nhánh cận cơ bản

```
public class TSP {
  public void print(){
    int ff = f + c[x[n-1]][x[0]];
    for(int i = 0; i < n; i++)
      System.out.print(x[i] + " ");
    System.out.println(", f = " + ff);
  }
  public void printBest(){
    for(int i = 0; i < n; i++)
      System.out.print(x_best[i] + " ");
    System.out.println(", f_best = " + f_best);
  }
}
```

Bài toán Người bán hàng - Nhánh cận cơ bản

```
public class TSP {
  public void solve(){

    visited = new boolean[n];
    x = new int[n];
    x_best = new int[n];
    for(int v= 0; v < n; v++)
    visited[v] = false;
    f = 0;
    f_best = 100000000;
    x[0] = 0;
    visited[x[0]] = true;
    extend(1);
  }
  public static void main(String[] args) {
    TSP tsp = new TSP();
    tsp.readData("data\\week4\\TSP\\tsp-15.txt");
    tsp.solve();
  }

}
```

Bài toán Người bán hàng - Nhánh cận lần hai

- Giới hạn dưới
  - Một hành trình (Tour) được gắn với một tập S của n ô trong ma trận khoảng cách trong đó mỗi hàng, cột của ma trận khoảng cách chứa chính xác một phần tử của S
  - Do đó hành trình tối ưu không thay đổi nếu chúng ta loại trừ mỗi ô của một dòng (hoặc hàng) với cùng một giá trị
  - Thuật toán **reduce** sẽ tính giới hạn dưới của hành trình tối ưu

Algorithm 18: reduce(C)

```
1..k là kích thước của ma trận chi phí C;
S ← 0;
foreach i ∈ 1..k do
    minRow ← giá trị nhỏ nhất trên hàng i của C;
    if minRow > 0 then
        foreach j ∈ 1..k do
            C[i][j] = C[i][j] − minRow;
        S ← S + minRow;
foreach j ∈ 1..k do
    minCol ← giá trị nhỏ nhất trên cột j của C;
    if minCol > 0 then
        foreach i ∈ 1..k do
            C[i][j] = C[i][j] − minCol;
        S ← S + minCol;
return S;
```

Bài toán Người bán hàng - Rẽ nhánh

- Chọn một cạnh (u, v) để rẽ nhánh (được tính bởi hàm **bestEdge**, *tìm cạnh tốt nhất*, dưới đây)
  - Với những hành trình chứa (u, v)
    - ★ Loại bỏ hàng u và cột v
    - ★ Đặt $C[v][u] = \infty$
    - ★ Nếu u là nột nút kết thúc của một đường đi $\langle x_1, x_2, ..., u \rangle$ và v là nút bắt đầu của một đường đi $\langle v, y_1, ..., y_k \rangle$, thì đặt $C[y_k][x_1] = \infty$ để tránh tạo hành trình con
  - Với những hành trình không chứa (u, v)
    - ★ Đặt $C[u][v] = \infty$

Bài toán Người bán hàng - Rẽ nhánh

- Khi ma trận giảm có kích thước 2×2

$$\begin{array}{c|cc} & w & x \\ \hline u & 0 & \infty \\ v & \infty & 0 \end{array} \qquad\qquad \begin{array}{c|cc} & w & x \\ \hline u & \infty & 0 \\ v & 0 & \infty \end{array}$$

a. chấp nhận (u, w) và (v, x) b. chấp nhận (u, x) và (v, w)

Algorithm 19: bestEdge(C)

```
1..k là kích thước của ma trận chi phí C;
best ← −∞;
foreach i ∈ 1..k do
    foreach j ∈ 1..k do
        if C[i][j] = 0 then
            minRow ← phần tử nhỏ nhất trên hàng i mà khác C[i][j];
            minCol ← phân tử nhỏ nhất trên cột j mà khác C[i][j];
            total ← minRow + minCol;
            if total > best then
                best ← total;
                selRow ← i;
                selCol ← j;
return (selRow, selCol)
```

Bài toán Người bán hàng

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

a. Ma trận khoảng cách ban đầu

	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	0	35	89	∞

Lower bound = 81

b. Ma trận giảm

Bài toán Người bán hàng

Tập các hành trình được chia ra 2 trường hợp:

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

Tours contain (6,3), lower bound = 81

	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	∞	35	89	∞

Tours do not contain (6,3), lower bound = 129

Bài toán Người bán hàng

Tập các hành trình chứa (6,3) được chia ra 2 trường hợp:

	1	2	4	5
1	∞	0	2	30
2	0	∞	30	17
3	29	1	∞	0
5	3	21	0	∞

Tours contain (6,3), (4,6), lower bound = 81

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

Tours contain (6,3), not (4,6), lower bound = 113

Bài toán Người bán hàng

Tập các hành trình chứa (6,3), (4,6) được chia ra 2 trường hợp:

	2	4	5
1	∞	2	28
3	0	∞	0
5	20	0	∞

Tours contain (6,3), (4,6), (2,1), lower bound = 84

	1	2	4	5
1	∞	0	2	30
2	∞	∞	30	17
3	29	1	∞	0
5	3	21	0	∞

Tours contain (6,3), (4,6), not (2,1), lower bound = 101

Bài toán Người bán hàng

Tập các hành trình chứa (6,3), (4,6), (2,1) được chia ra 2 trường hợp:

	2	5
3	∞	0
5	0	∞

Tours contain (6,3), (4,6), (2,1), (1,4), lower bound = 84

Add arcs (3,5) and (5,2), we obtain a solution cost = 104

	2	4	5
1	∞	∞	0
3	0	∞	0
5	20	0	∞

Tours contain (6,3), (4,6), (2,1), not (1,4), lower bound = 112

Bài toán Người bán hàng

Tập các hành trình chứa (6,3), (4,6) nhưng không chứa (2,1) được chia ra 2 trường hợp:

	2	4	5
1	0	0	∞
2	∞	11	0
3	1	∞	0

Tours contain (6,3), (4,6), not (2,1), (5,1), lower bound = 103

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	0	1	∞	0
5	∞	21	0	∞

Tours contain (6,3), (4,6), not (2,1), not (5,1), lower bound = 127

Bài toán Người bán hàng

Tập các hành trình chứa (6,3), (4,6), (5,1) nhưng không chứa (2,1) được chia ra 2 trường hợp:

	2	5
2	∞	0
3	1	∞

Tours contain (6,3), (4,6), not (2,1), (5,1), (1,4), lower bound = 103

	2	4	5
1	0	∞	∞
2	∞	0	0
3	1	∞	0

Tours contain (6,3), (4,6), not (2,1), (5,1), not (1,4), lower bound = 114

- Cuối cùng, hành trình tốt nhất có khoảng cách là 104

Nhánh cận: MaxClique

- Miêu tả
  - Đầu vào: một đồ thị vô hướng $G = (V, E)$,
  - Đồ thị con: Đặt $G(S)$ là đồ thị (S, E_S) trong đó $E_S = \{(u, v) \mid u, v \in S \wedge (u, v) \in E\}$. $G(S)$ được gọi là đồ thị con với tập S $(\forall S \subseteq V)$
  - Đầu ra: đồ thị con hoàn chỉnh cực đại (hay còn gọi *clique*) của G
- Nhánh cận
  - Giải pháp bộ phận Q: tập các nút, hai nút của Q đều kề nhau
  - Các nút tiềm năng $Cand$ cho sự mở rộng: mỗi nút của $Cand$ liền kề với mọi nút của Q
  - Giới hạn trên (cận trên)
    - ★ Δ là số lượng màu được sử dụng để tô màu các nút của $Cand$ sao cho hai nút liền kề $u, v \in Cand$ phải được tô màu khác nhau.
    - ★ Kích thước của mỗi đồ thị con hoàn chỉnh của $G(Cand)$ nhỏ hơn hoặc bằng Δ
    - ★ $|Q| + \Delta$ là giới hạn trên của kích thước của clique được mở rộng từ Q
    - ★ Nếu $|Q| + \Delta \leq |Qmax|$, thì không mở rộng Q

Nhánh cận: MaxClique

Algorithm 20: MaxClique($G = (V, E)$)

Input: Đồ thị $G = (V, E)$
Output: Đồ thị con hoàn thiện tối đa của G
$Qmax \leftarrow \{\}$;
$Q \leftarrow \{\}$;
$Cand \leftarrow$ danh sách các nút của V;
$\Delta \leftarrow$ Sort($Cand$);
Expand($Cand$);
return $Qmax$;

Nhánh cận: MaxClique

Algorithm 21: Expand($Cand$)

Input: Danh sách được sắp xếp của các ứng viên $Cand$, $G = (V, E)$ và $Q, Qmax$ là các biến toàn cục

Output: Mở rộng lời giải cục bộ Q

foreach $i = 0, \ldots, lenght(Cand) - 1$ **do**

  $u \leftarrow Cand[i]$;

  $Q \leftarrow Q \cup \{u\}$;

  if $|Q| > |Qmax|$ **then**

    $Qmax \leftarrow Q$;

  $Cand' \leftarrow \{v \in Cand \mid v \neq u \wedge (u, v) \in E\}$;

  $\Delta \leftarrow$ Sort($Cand'$);

  if $|Q| + \Delta > |Qmax|$ **then**

    Expand($Cand'$);

  $Q \leftarrow Q \setminus \{u\}$;

Algorithm 22: Sort($Cand$)

Input: Sắp xếp danh sách các ứng viên $Cand$
Output: Cập nhật $Cand$ và trả về số lớp

```
maxNo ← 0;
C_1 ← {};
foreach u ∈ Cand do
    k ← 1;
    while ∃v ∈ C_k | (u, v) ∈ E do
        k ← k + 1;
    if k > maxNo then
        maxNo ← k;
        C_k ← {};
    C_k ← C_k ∪ {u}
L ← [];
foreach k = 1, ..., maxNo do
    foreach v ∈ C_k do
        L ← L :: v;
foreach i = 0, ..., length(L) − 1 do
    Cand[i] ← L[length(L) − i − 1];
return maxNo;
```