Nội dung

1. Phương pháp chia để trị
2. Phương pháp quay lui
3. Phương pháp tham lam
4. Phương pháp quy hoạch động

PHƯƠNG PHÁP CHIA ĐỂ TRỊ
(Divide and Conquer)

- **Ý tưởng**
  - Phân rã bài toán kích thước n thành các bài toán con nhỏ hơn mà việc tìm lời giải của chúng là cùng một cách.
  - Lời giải của bài toán đã cho được xây dựng từ lời giải của các bài toán con này

❖ **Mô hình:**

Nếu gọi D&C(ℜ) - Với ℜ là miền dữ liệu - là hàm thể hiện cách giải bài toán theo phương pháp chia để trị thì ta có thể viết:

```
void D&C(ℜ)
{
        If (ℜ đủ nhỏ)
                giải bài toán;
        Else
        {
                Chia ℜ thành ℜ1, …, ℜm;
                for (i = 1; i <=m; i++)
                        D&C(ℜi);
                Tổng hợp kết quả;
        }
}
```

Bài toán MinMax

❖ **Phát biểu bài toán:**

Tìm giá trị Min, Max trong đoạn a[l..r] của mảng a[1..n].

❖ **Ý tưởng**

Tại mỗi bước, chia đôi đoạn cần tìm rồi tìm Min, Max của từng đoạn, sau đó tổng hợp lại kết quả.

Nếu đoạn chia chỉ có 1 phần tử thì Min = Max và bằng phần tử đó.

❖ **Minh họa:**

i	1	2	3	4	5	6	7	8
a[i]	10	1	5	0	9	3	15	19

Tìm giá trị Min, Max trong đoạn a[2..7] của mảng a[1..8].

Ký hiệu:

MinMax(a,l,r,Min,Max): cho Min và Max trong đoạn a[l..r].

Khi đó:

MinMax(a,2,7,Min,Max) Cho Min = 0 và Max = 15 trong đoạn a[2..7]

❖ **Minh họa:**

MinMax(a,2,7,Min,Max)

Ban đầu

i	2	3	4	5	6	7
a[i]	1	5	0	9	3	15

MinMax(a,2,4,Min1,Max1)

i	2	3	4
a[i]	1	5	0

MinMax(a,5,7,Min2,Max2)

i	5	6	7
a[i]	9	3	15

MinMax(a,2,3,Min3,Max3)

i	2	3
a[i]	1	5

MinMax(a,4,4,Min4,Max4)

i	4
a[i]	0

Min4=0
Max4=0

MinMax(a,5,6,Min5,Max5)

i	5	6
a[i]	9	3

MinMax(a,7,7,Min6,Max6)

i	7
a[i]	15

Min6=15
Max6=15

MinMax(a,2,2,Min7,Max7)

i	2
a[i]	1

Min7=1
Max7=1

MinMax(a,3,3,Min8,Max8)

i	3
a[i]	5

Min8=5
Max8=5

MinMax(a,5,5,Min9,Max9)

i	5
a[i]	9

Min9=9
Max9=9

MinMax(a,6,6,Min10,Max10)

i	6
a[i]	3

Min10=3
Max10=3

Min3=1
Max3=5 → Min1=0
Max1=5 → **Min=0**
Max=15 ← Min2=3
Max2=15 ← Min5=3
Max5=9

❖ **Thuật toán:**

Input: a[l..r], (l ≤ r)
Output: Min = Min (a[l],..,a[r]),
Max = Max (a[l],..,a[r]).

```
MinMax(a,l, r, Min, Max) ≡
    if (l == r)
    {
        Min = a[l];
        Max = a[l];
    }
    Else
    {
        MinMax(a,l, (l+r) / 2, Min1, Max1);
        MinMax(a,(l+r) /2 + 1, r , Min2, Max2);
        If (Min1 < Min2)
            Min = Min1;
        Else
            Min = Min2;
        If (Max1 > Max2)
            Max = Max1
        Else
            Max = Max2;
    }
```

❖ **Độ phức tạp thuật toán:**

Gọi T(n) là số phép toán so sánh cần thực hiện. Khi đó ta có:

$$T(n) = \begin{cases} T(n/2) + T(n/2) + 2 \ ; \ n > 2 \\ 1 \ ; \ n = 2 \\ 0 \ ; \ n = 1 \end{cases}$$

Với $n = 2^k$, thì:

$$T(n) = 2 + 2T(n/2) = 2 + 2^2 + 2^2 T(n/2^2)$$
$$= \cdots = 2^{k-1} T(2) + \sum_{i=1}^{k-1} 2^i$$
$$= \sum_{i=1}^{k} 2^i - 2^{k-1} = 2^{k+1} - 2^{k-1} - 2 = \frac{3n}{2} - 2$$

Vậy $T(n) \in O(n)$.

```
void MinMax(int a[ ], int l, int r, int &Min, int &Max )
{
   int Min1,Min2,Max1,Max2;
  if (l == r )
  {
        Min = a[l];
        Max= a[l];
  }
```

```
  else
  {
        MinMax(a,l,(l+r)/2 , Min1, Max1);
        MinMax(a,(l+r) /2 + 1,r, Min2, Max2);
        if (Min1 < Min2)
              Min = Min1;
        else
              Min = Min2;
        if (Max1 > Max2)
              Max = Max1;
        else
              Max = Max2;
   }
}
```

Phương pháp tham lam (The greedy method)

- **Ý tưởng**
  - Phương pháp tham lam là kỹ thuật thiết kế thường được dùng để giải các bài toán tối ưu. Phương pháp được tiến hành trong nhiều bước. Tại mỗi bước, theo một chọn lựa nào đó (xác định bằng một hàm chọn), sẽ tìm một lời giải tối ưu cho bài toán nhỏ tương ứng. Lời giải của bài toán được bổ sung dần từng bước từ lời giải của các bài toán con.
  - Lời giải được xây dựng như thế có chắc là lời giải tối ưu của bài toán?
  - Các lời giải tìm được bằng phương pháp tham lam thường chỉ là chấp nhận được theo điều kiện nào đó, chưa chắc là tối ưu.

❖ **Mô hình:**

- Cho trước một tập A gồm n đối tượng, ta cần phải chọn một tập con S của A. Với một tập con S được chọn ra thỏa mãn các yêu cầu của bài toán, ta gọi là một nghiệm chấp nhận được. Một hàm mục tiêu gắn mỗi nghiệm chấp nhận được với một giá trị. Nghiệm tối ưu là nghiệm chấp nhận được mà tại đó hàm mục tiêu đạt giá trị nhỏ nhất (lớn nhất).

- Đặc trưng tham lam của phương pháp thể hiện bởi: trong mỗi bước việc xử lí sẽ tuân theo một sự chọn lựa trước, không kể đến tình trạng không tốt có thể xảy ra khi thực hiện lựa chọn lúc đầu.

Chọn S từ tập A.

Tính chất tham lam của thuật toán định hướng bởi hàm Chọn.

Các bước thực hiện:

- Khởi động S = ∅;
- Trong khi A ≠ ∅:
  - Chọn phần tử tốt nhất của A gán vào x: x = Chọn (A);
  - Cập nhật các đối tượng đã chọn: A = A-{x};
  - Nếu S∪ {x} thỏa mãn yêu cầu bài toán thì

    Cập nhật lời giải: S = S∪ {x};

❖ ***Thủ tục tham lam:***

- Input A[1..n]
- Output S // lời giải;

```
greedy (A,n) ≡
S = ∅;
while (A ≠ ∅)
{
        x= Chọn (A);
        A = A-{x}
        if (S∪ {x} chấp nhận được)
                S = S∪ {x};
}
return S;
```

Bài toán người du lịch

❖ **Phát biểu bài toán**

- Một người du lịch muốn tham quan n thành phố T_1,..., 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 1 lần rồi quay trở lại thành phố xuất phát.
- Gọi C_{ij} là chi phí đi từ thành phố T_i đến T_j. Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

Ý tưởng:

- Đây là bài toán tìm chu trình có trọng số nhỏ nhất trong một đơn đồ thị có hướng có trọng số.
- Thuật toán tham lam cho bài toán là chọn thành phố có chi phí nhỏ nhất tính từ thành phố hiện thời đến các thành phố chưa qua.

Thuật toán

Input C= (Cij), u

output TOUR //Hành trình tối ưu,

COST;//Chi phí tương ứng

Mô tả:

TOUR:= 0; COST:= 0; v:= u; // Khởi tạo

$\forall$k:= 1 $\rightarrow$ n://Thăm tất cả các thành phố

// Chọn tp kế)

- Chọn <v, w> là đoạn nối 2 thành phố có chi phí nhỏ nhất tính từ TP v đến các thành phố chưa qua.
- TOUR:= TOUR + <v, w>; //Cập nhật lời giải
- COST:= COST + Cvw ; //Cập nhật chi phí

// Chuyến đi hoàn thành

TOUR:= TOUR + <v, u>;

COST:= COST + Cvu ;

1. TOUR = 0; COST = 0; u = v = 1;
   ⇒ w = 2;
2. TOUR = <1,2>; COST = 1; v = 2;

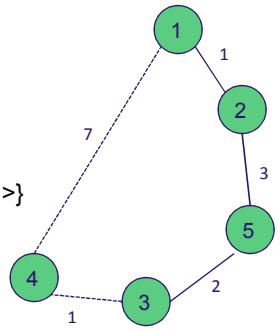
   ⇒ w = 5;
3. TOUR = {<1,2>, <2,5>}
   COST = 4; v = 5;

   ⇒ w = 3;
4. TOUR = {<1,2>, <2,5>,<5,3>}
   COST = 6; v = 3;

   ⇒ w = 4;
5. TOUR = {<1,2>, <2,5>,<5,3>,<3,4>}
   COST = 7; v = 4;

TOUR = {<1,2>, <2,5>,<5,3>,<3,4>,<4,1>}
COST = 14

$$C = \begin{bmatrix} 0 & 1 & 2 & 7 & 5 \\ 1 & 0 & 4 & 4 & 3 \\ 2 & 4 & 0 & 1 & 2 \\ 7 & 4 & 1 & 0 & 3 \\ 5 & 3 & 2 & 3 & 0 \end{bmatrix}$$



❖ **Độ phức tạp của thuật toán**

Thao tác chọn đỉnh thích hợp trong n đỉnh được tổ chức bằng một vòng lặp để duyệt. Nên chi phí cho thuật toán xác định bởi 2 vòng lặp lồng nhau, nên $T(n) \in O(n^2)$.

Cài đặt

```
int GTS (mat a, int n, int TOUR[max], int Ddau)
{
    int    v,          //Dinh dang xet
           k,          //Duyet qua n dinh de chon
           w;          //Dinh duoc chon trong moi buoc
    int    mini;       //Chon min cac canh(cung) trong moi buoc
    int COST;          //Trong so nho nhat cua chu trinh
    int daxet[max];            //Danh dau cac dinh da duoc su dung
    for(k = 1; k <= n; k++)
           daxet[k] = 0;       //Chua dinh nao duoc xet
    COST = 0;      //Luc dau, gia tri COST == 0
    int i;  // Bien dem, dem tim du n dinh thi dung
    v = Ddau;      //Chon dinh xuat phat la 1
    i = 1;
    TOUR[i] = v; //Dua v vao chu trinh
    daxet[v] = 1;  //Dinh v da duoc xet
```

```
    while(i < n)
    {
           mini = VC;
           for (k = 1; k <= n; k++)
                  if(!daxet[k])
                         if(mini > a[v][k])
                         {
                                mini = a[v][k];
                                w = k;
                         }
           v = w;
           i++;
           TOUR[i] = v;
           daxet[v] = 1;
           COST += mini;
    }
    COST += a[v][Ddau];
    return COST;
}
```

Bài toán hoán đổi 2 phần trong 1 dãy

❖ **Phát biểu bài toán**

a[1..n] là một mảng gồm n phần tử. Ta hoán chuyển m phần tử đầu tiên của mảng với phần còn lại của mảng (n-m phân tử) mà không dùng một mảng phụ.

Chẳng hạn, với n = 8, a[1..8] = (1, 2, 3, 4, 5, 6, 7, 8)
Nếu m = 3, thì kết quả là: (4, 5, 6, 7, 8, 1, 2, 3)
Nếu m = 5, thì kết quả là: (6, 7, 8, 1, 2, 3, 4, 5)
Nếu m = 4, thì kết quả là: (5, 6, 7, 8, 1, 2, 3, 4)

❖ **Ý tưởng**

- **Nếu m = n – m: Hoán đổi các phần tử của 2 nửa mảng có độ dài bằng nhau.**
- **Nếu m ≠ n – m:**
  - Nếu m < n – m: hoán đổi m phần tử đầu với m phân tử cuối của phần còn lại. Sau đó trong mảng a[1..n-m] ta chỉ cần hoán đổi m phần tử đầu với phần còn lại.
  - Nếu m > n – m: hoán đổi n-m phần tử đầu tiên với n-m phần tử của phần sau. Sau đó trong mảng a[n-m+1.. n] ta hoán đổi n-m phần tử cuối mảng với các phần tử của phần đầu.

Như vậy, bằng cách áp dụng phương pháp chia để trị, ta chia bài toán thành 2 bài toán con:

- Bài toán thứ nhất là hoán đổi hai mảng con có độ dài bằng nhau, cụ thể là hoán đổi nửa số phần tử đầu và cuối của mảng cho nhau bằng cách đổi chỗ từng cặp phần tử tương ứng.
- Bài toán thứ hai cùng dạng với bài toán đã cho nhưng kích thước nhỏ hơn, nên có thể gọi thuật toán đệ qui để giải và quá trình gọi đệ qui sẽ dừng khi đạt tới sự hoán đổi 2 phần có độ dài bằng nhau.

Vậy mấu chốt của thuật toán là thực hiện thao tác hoán đổi 2 nhóm phần tử, lặp lại thao tác này trong khi số lượng phần tử của 2 nhóm còn khác nhau. Nên ta sẽ thay thuật toán đệ qui bằng thuật toán lặp.

❖ **Thuật toán:**

Input: a[1..n], m (m ≤n)

Output: a[1..n] với tính chất m phần tử đầu mảng a (mảng nhập) nằm cuối mảng kết quả, n-m phần tử cuối mảng nhập nằm ở đầu mảng kết quả.

❖ Mô tả:

```
Transpose(a,n,m) ≡
        i = m; j = n-m; m = m+1;
        while (i ≠ j)
                if (i > j)
                {
                        Exchange(a,m-i,m,j);
                        i = i – j;
                }
                else
                {
                        j = j – i;
                        Exchange(a,m-i,m+j,i);
                }
        Exchange(a,m-i,m,i);
```

❖ **Thuật toán exchange**

input a,

i,j, //vị trí

m; // Số phần tử cần hoán đổi

output a

Mô tả:

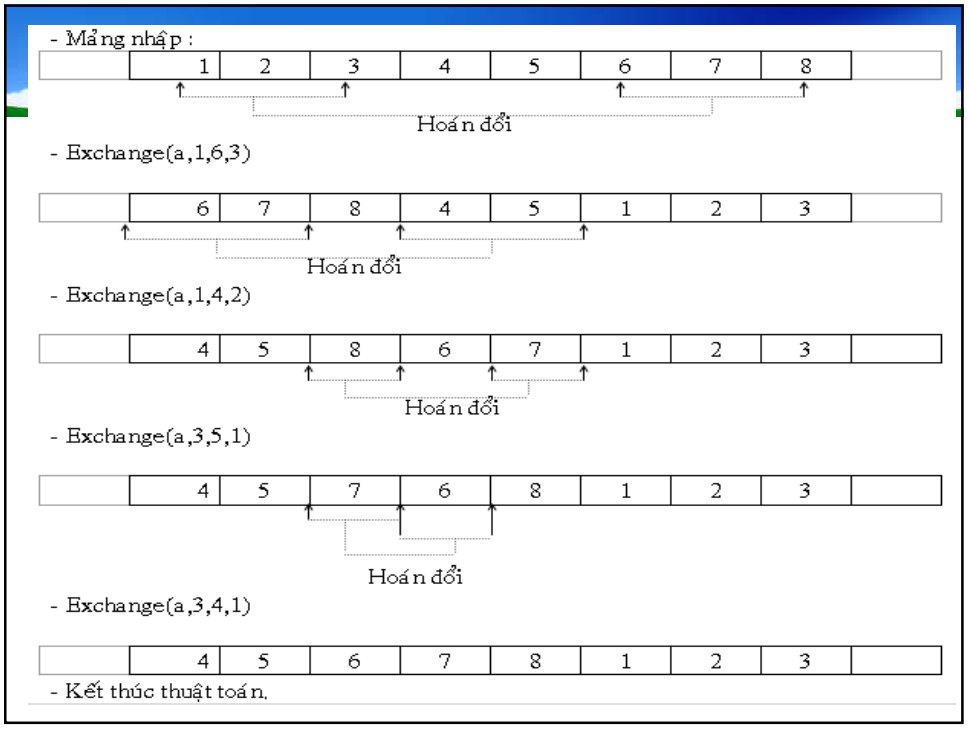
Exchange(a,i,j,m) ≡

Cho p = 0 → m-1

Đổichỗ (a[i+p], a[j+p]);

Minh họa:

n = 8, a[8] = (1, 2, 3, 4, 5, 6, 7, 8) , m = 3.



❖ **Độ phức tạp thuật toán:**

Kí hiệu: T(i, j) là số phần tử cần đổi chỗ để hoán đổi dãy i phần tử và dãy j phần tử, ta có công thức truy hồi sau:

$$T(i,j) = \begin{cases} i; \text{ neáu } i = j \\ j + T(i-j, j); \text{ neáu } i > j \\ i + T(i, j-i); \text{ neáu } i < j \end{cases}$$

T(i,j) = i + j – UCLN(i,j)

UCLN(i,j) là ước chung lớn nhất của i, j.

❖ **Cài đặt:**

```
void Transpose(day a,int n,int m)
{
    int i, j;
    i = m;
    j = n-m;
    m = m+1;
    while (i != j )
        if(i > j)
        {
                Exchange(a,m-i,m,j);
                i = i - j;
        }
        else
        {
                j = j - i;
                Exchange(a,m-i,m+j,i);
        }
    Exchange(a,m-i,m,i);
}
//************************
void Exchange(day a, int i, int j, int m)
{
    for (int k = 0; k <= m-1; k++)
        doicho(a[i+k], a[j+k]);
}
```

Phương pháp quay lui (Back Tracking)

❖ **Ý tưởng**

Đặc trưng: Các bước hướng tới lời giải cuối cùng của bài toán hoàn toàn được làm thử.

Tại mỗi bước:

Nếu có một lựa chọn được chấp nhận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo.

Nếu không có sự lựa chọn nào cả thì làm lại bước trước, xoá bỏ sự ghi nhận và quay về tiến trình thử các lựa chọn còn lại.

- Hành động này được gọi là quay lui, thuật toán thể hiện phương pháp này gọi là thuật toán quay lui.
- Điểm quan trọng là phải ghi nhớ lại tại mỗi bước để tránh trùng lặp khi quay lui.

Các thông tin này lưu trữ vào một ngăn xếp, nên thuật toán thể hiện cách tổ chức đệ quy.

❖ **Mô hình:**

Lời giải của bài toán: $x = (x_1,.., x_n)$, $\forall i$, x_i phải thỏa mãn các điều kiện nào đó.

Cần xác định các thành phần lời giải x_i.

Tại mỗi bước i:

Đã xây dựng xong các thành phần $x_1,.., x_{i-1}$

Xây dựng thành phần x_i bằng cách lần lượt thử tất cả các khả năng mà x_i có thể chọn:

- Nếu có một khả năng j nào đó phù hợp cho x_i thì xác định x_i theo j.
- Thường có thêm thao tác ghi nhận trạng thái mới của bài toán để hỗ trợ cho bước quay lui.
- Nếu i = n thì ta có được một lời giải, ngược lại thì tiến hành bước i+1 để xác định x_{i+1}.
- Ngược lại, ta lùi về bước trước (bước i-1) để xác định lại thành phần x_{i-1}.

▪ Giả định tại mỗi bước, số lượng các khả năng chọn lựa cho x_i là như nhau.

❖ ***Mô hình phương pháp quay lui: Với n là số bước cần phải thực hiện, k là số khả năng mà x_i có thể chọn lựa.***

```
Try(i) ≡
        for (j = 1 → k)
                If (xi chấp nhận được khả năng j)
                {
                        Xác định xi theo khả năng j;
                        Ghi nhận trạng thái mới;
                        if(i < n)
                                Try(i+1);
                        else
                                Ghi nhận nghiệm;
                        Trả lại trạng thái cũ cho bài toán;
                }
```

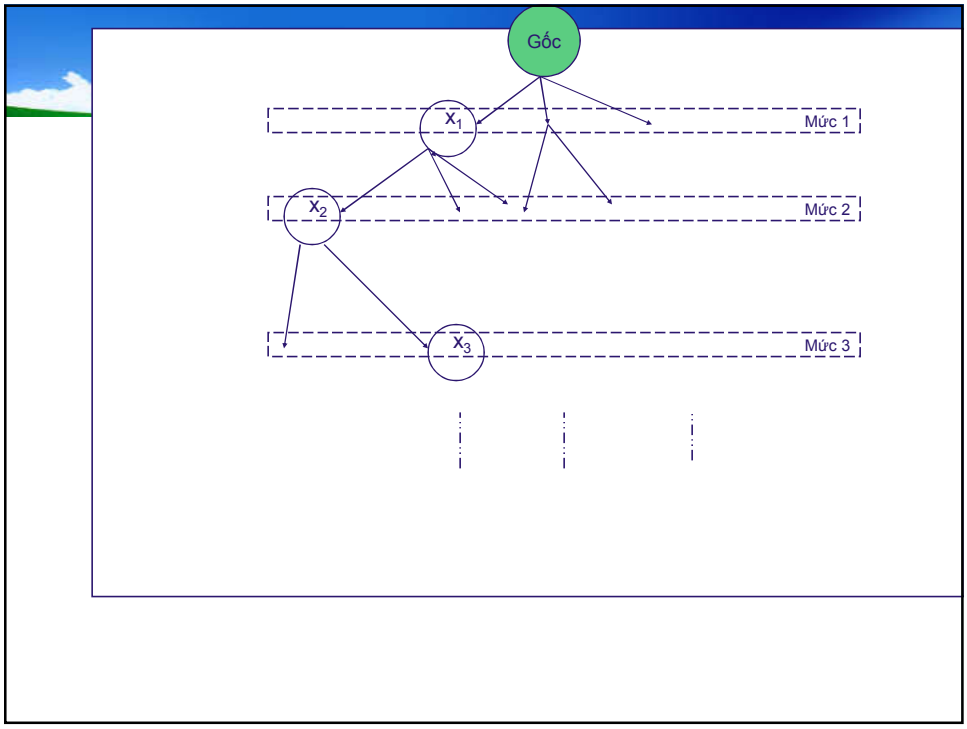
❖ ***Ghi chú:***

Tìm nghiệm bằng phương pháp quay lui có thể chuyển về tìm kiếm trên cây không gian các trạng thái, với cây được xây dựng từng mức như sau:

- Các nút con của gốc (thuộc mức 1) là các khả năng có thể chọn cho x_1.
- Giả sử x_{i-1} là một nút ở mức thứ i-1, khi đó các nút con của x_{i-1} là các khả năng mà x_i có thể chọn.

Như vậy, mỗi nút x_i của cây biểu diễn một lời giải bộ phận, đó là các nút nằm trên đường đi từ gốc đến nút đó.

Ta có thể nói việc tìm kiếm nghiệm bằng phương pháp quay lui chính là tìm kiếm theo chiều sâu trên cây không gian các trạng thái.



Bài toán Ngựa đi tuần

❖ **Phát biểu bài toán**

Cho bàn cờ có n x n ô. Một con ngựa đi theo luật cờ vua, đầu tiên được đặt ở ô có tọa độ x_0 , y_0.

Vấn đề: chỉ ra các hành trình (nếu có) của ngựa (đó là đường đi của ngựa qua tất cả các ô của bàn cờ, mỗi ô đi qua đúng một lần).

❖ **Thiết kế thuật toán**

Mấu chốt vấn đề là xét xem có thể thực hiện một nước đi kế nữa hay không. Sơ đồ đầu tiên có thể phát thảo như sau:

```
Try(i) ≡
        for (j = 1 → k)
                If (x_i chấp nhận được khả năng j)
                {
                        Xác định x_i theo khả năng j;
                        Ghi nhận trạng thái mới;
                        if(i < n^2 )
                                Try(i+1);
                        else
                                Ghi nhận nghiệm;
                        Trả lại trạng thái cũ cho bài toán;
                }
```

❖ **Các vấn đề cần giải quyết:**

1. Biểu diễn bàn cờ.
2. Các khả năng chọn lựa cho x_i?
3. Cách thức xác định x_i theo j.
4. Cách thức ghi nhận trạng thái mới, trả về trạng thái cũ.
5. Ghi nhận nghiệm.

. . .

Biểu diễn bàn cờ (bằng 1 ma trận vuông cấp n): int h[n][n];

Ta qui ước như sau

h[x][y] = 0 ≡ Ô <x,y> ngựa chưa đi qua;

h[x][y] = i ≡ Ô <x,y> ngựa đã đi qua ở bước thứ i ($1 \le i \le n^2$).

- Các khả năng chọn lựa cho x_i? Đó chính là các nước đi của ngựa mà x_i có thể chấp nhận được.
- Với cặp tọa độ bắt đầu <x,y> như trong hình vẽ, có tất cả 8 ô <u,v> mà con ngựa có thể đi đến. Giả sử chúng được đánh số từ 0 đến 7 như hình sau:

Tọa độ (a,b)			1	2	3	4	5			Tọa độ (a,b)
(-2,-1)				4		3		1		(-2,1)
(-1,-2)			5				2	2		(-1,2)
	hàng	x →						3		
(1,-2)			6				1	4		(1,2)
(2,-1)				7		0		5		(2,1)
				cột	↑ y					

(8 bước đi có thể có của con ngựa)

Dùng 2 mảng a và b lưu trữ các sai biệt về tọa độ cũ và mới:

u = x +a[k]; v = y + b[k];

Điều kiện "chấp nhận được" có thể được biểu diễn như kết hợp của các điều kiện:

- Ô mới phải thuộc bàn cờ ($1 \le u \le n$ và $1 \le v \le n$)
- và chưa đi qua ô đó, nghĩa là h[u,v] = 0;

Ghi nhận nước đi hợp lệ ở bước i, ta gán h[u][v] = i;

Hủy một nước đi thì ta gán h[u][v] = 0.

Ma trận h ghi nhận kết quả nghiệm:

- Nếu có <x,y> sao cho h<x,y> = 0: đó không là lời giải của bài toán
- Ngược lại h<x,y> chứa đường đi của ngựa.

Mô tả thuật toán:

Input n, //Kích thước bàn cờ
x, y;//Toạ độ xuất phát ở bước i
Output h;

```
Try(i, x, y) ≡
    for(k = 0; k <= 7; k++)
    {
        u = x + a[k];
        v = y + b[k];
        if (1 <= u ,v <= n &&h[u][v] == 0)
        {
            h[u][v] = i;
            if (i < n*n)
                Try(i+1,u,v);
            else
                xuat_h(); // In ma traän h
            h[u][v] = 0;
        }
    }
```

Thủ tục này xuất tất cả các lời giải, nếu có.

Thủ tục đệ qui được khởi động bằng một lệnh gọi các tọa độ đầu x_0, y_0 là tham số. Ô xuất phát có trị 1, còn các ô khác được đánh dấu còn trống.

```
H[x0 ][y0 ] = 1;
Try(2,x0 , y0 );
```

Các mảng a và b có thể khởi đầu như sau:

```
int a[8]= {2,1,-1,-2,-2,-1,1,2};
int b[8]= {1,2,2,1,-1,-2,-2,-1};
```

PHƯƠNG PHÁP QUY HOẠCH ĐỘNG (Dynamic Programming)

Phương pháp tổng quát

- Phương pháp quy hoạch động kết hợp chặt chẽ với cách "Chia để trị": Khi không biết cần phải giải bài toán lớn, ta giải tất cả các bài toán con và lưu trữ những lời giải này (để khỏi phải tính toán lại) nhằm sử dụng lại chúng để giải bài toán lớn hơn.
- Phương pháp này tổ chức tìm kiếm lời giải theo kiểu từ dưới lên (bottom up). Xuất phát từ các bài toán con nhỏ và đơn giản nhất, tổ hợp các lời giải của chúng để có lời giải của bài toán con lớn hơn... và cứ như thế để tìm lời giải của bài toán ban đầu.
- Khi sử dụng phương pháp quy hoạch động để giải quyết vấn đề, ta có thể gặp 2 khó khăn sau:

  1.Số lượng lời giải của các bài toán con có thể rất lớn không chấp nhận được.

  2.Không phải lúc nào sự kết hợp lời giải của các bài toán con cũng cho ra lời giải của bài toán lớn hơn.

- Để giải quyết những trường hợp như vậy, phương pháp quy hoạch động dựa vào một nguyên lý, gọi là nguyên lý tối ưu (The principle of optimality) của Bellman:

  " Nếu lời giải của bài toán là tối ưu thì
  lời giải của các bài toán con cũng tối ưu".

- Trong thuật toán quy hoạch động thường dùng các thao tác:
  - Xây dựng một hàm quy hoạch động (hoặc phương trình quy hoạch động).
  - Lập bảng lưu lại các giá trị của hàm.
  - Truy xuất lời giải tối ưu của bài toán từ bảng lưu.
  - ...
- Trong phần này ta giới thiệu một số bài toán có thể dùng quy hoạch động giải quyết một cách hiệu quả.

  Những vấn đề này đều liên quan đến bài toán tìm phương án tối ưu để thực hiện một công việc nào đó, và chúng có chung một tính chất là đáp án tốt nhất cho một bài toán con vẫn được duy trì khi bài toán đó trở thành một phần trong một bài toán lớn.

THUẬT TOÁN FLOYD

TÌM ĐƯỜNG ĐI NGẮN NHẤT GIỮA CÁC CẶP ĐỈNH

Bài toán:

Cho G = (V,E) là một đơn đồ thị có hướng có trọng số. V = {1,..,n} là tập các đỉnh. E là tập các cung. Tìm đường đi ngắn nhất giữa các cặp đỉnh của đồ thị.

- ***Ý tưởng:***

Thuật toán Floyd được thiết kế theo phương pháp quy hoạch động. Nguyên lý tối ưu được vận dụng cho bài toán này là:

"Nếu k là đỉnh nằm trên đường đi ngắn nhất từ i đến j thì đoạn đường từ i đến k và từ k đến j cũng phải ngắn nhất".

- ***Thiết kế:***

Đồ thị được biểu diễn bởi ma trận kề các trọng số của cung:

Ta ký hiệu : $a = (a_{ij})_{nxn}$

$$\forall i, j \in \{1,\ldots,n\} : a_{ij} = \begin{cases} Troïng\ soá\ (i, j); (i, j) \in E \\ 0; i = j \\ \infty; (i, j) \notin E \end{cases}$$

- **Ma trận trọng số đường đi ngắn nhất giữa các cặp đỉnh: d = (d_{ij})**

  d_{ij}: Trọng số của đường đi ngắn nhất từ i đến j.

- **Ma trận xác định các đỉnh trung gian của đường đi ngắn nhất từ i đến j: p = (p_{ij})**

  p_{ij}: đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian p_{ij} hay không?

  - p_{ij} = 0; đường đi ngắn nhất từ i đến j không có đi qua đỉnh trung gian pij.
  - $p_{ij} \neq 0$; đường đi ngắn nhất từ i đến j đi qua đỉnh trung gian pij.

- **Ở bước k:**
  - Ký hiệu ma trận d là d^k cho biết chiều dài nhỏ nhất của đường đi từ i đến j.
  - Ký hiệu ma trận p là p^k cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian thuộc tập đỉnh {1,..,k}.

Input a
Output d,p;

❖ **Mô tả:**

Bước 0:

- Khởi động d: d = a ; (= d^0)
- Khởi động p: p_{ij} = 0;

Bước 1:

- Kiểm tra mỗi cặp đỉnh i, j: Có/không một đường đi từ i đến j đi qua đỉnh trung gian 1, mà có trọng số nhỏ hơn bước 0? Trọng số của đường đi đó là:

  $d^1_{ij} = \text{Min}\{ d^0_{ij} , d^0_{i1} + d^0_{1j} \}$
- Nếu $d^1_{ij} = d^0_{i1} + d^0_{1j}$ thì p^1_{ij} = 1, tức là đường đi tương ứng đi qua đỉnh 1.

Bước 2:

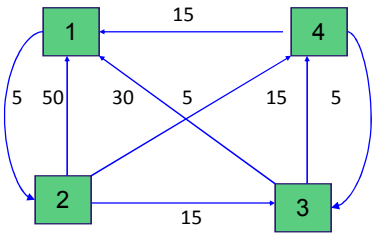
- Kiểm tra mỗi cặp đỉnh i, j: Có/không một đường đi từ i đến j đi qua đỉnh trung gian 2, mà có trọng số nhỏ hơn bước 1? Trọng số của đường đi đó là:

  $d^2_{ij} = \text{Min}\{ d^1_{ij} , d^1_{i2} + d^1_{2j}\}$
- Nếu $d^2_{ij} = d^1_{i2} + d^1_{2j}$ thì p^2_{ij} = 2: tức là đường đi tương ứng đi qua đỉnh 2.
- ...

Cứ tiếp tục như vậy, thuật toán kết thúc sau bước n, ma trận d xác định trọng số đường đi ngắn nhất giữa 2 đỉnh bất kỳ i, j. Ma trận p cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian p_{ij}.

❖ **Minh hoạ:**

Tìm đường đi ngắn nhất giữa các cặp đỉnh của đồ thị:



$$\begin{bmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{bmatrix}$$

❖ **Hoạt động của thuật toán như sau :**

b^1		1	2	3	4
d^1	1	0	5	∞	∞
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^1	1	0	0	0	0
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

$d^1_{ij} = \text{Min}\{ d^0_{ij} , d^0_{i1} + d^0_{1j} \}$

$$\begin{bmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{bmatrix}$$

Nếu $d^1_{ij} = d^0_{i1} + d^0_{1j}$ thì $p^1_{ij} = 1$, tức là đường đi tương ứng đi qua đỉnh 1.

b^2		1	2	3	4
d^2	1	0	5	20	10
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^2	1	0	0	2	2
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

$d^2_{ij} = \text{Min}\{ d^1_{ij} , d^1_{i2} + d^1_{2j}\}$

Nếu $d^2_{ij} = d^1_{i2} + d^1_{2j}$ thì $p^2_{ij} = 2$: tức là đường đi tương ứng đi qua đỉnh 2.

5	20	10
0	15	5
35	0	15
20	5	0

b^3		1	2	3	4
d^3	1	0	5	20	10
	2	45	0	15	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^3	1	0	0	2	2
	2	3	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b^4		1	2	3	4
d^4 = d	1	0	5	15	10
	2	20	0	10	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^4 = p	1	0	0	4	2
	2	4	0	4	0
	3	0	1	0	0
	4	0	1	0	0

- **Căn cứ vào ma trận d, ta chỉ ra khoảng cách đường đi ngắn nhất từ i đến j, và dựa vào p có thể xác định các đỉnh nằm trên đường đi ngắn nhất này.**

❖ **Chẳng hạn, với i = 1, j = 3.**

- Theo d, d_{13} = 15. Nên đường đi ngắn nhất từ 1 đến 3 có khoảng cách là 15.
- Theo p, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3 đi qua đỉnh trung gian p_{13} = 4, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 4 đi qua đỉnh trung gian p_{14} = 2, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 2 không đi qua đỉnh trung gian nào (p_{12} = 0).
- Vậy đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3 là:

$$1 \rightarrow 2 \rightarrow 4 \rightarrow 3.$$

❖ ***Cài đặt:***

```
void floyd()
{
    int i, j, k;
    // Khoi dong ma tran d va p
    for (i = 1; i<= n; i++)
            for (j = 1; j<= n; j++)
            {
                    d[i][j] = a[i][j];
                    p[i][j] = 0;
            }
    for (k = 1; k <= n; k++) // Tính ma trận d và p ở bước lặp k
      for (i = 1; i <= n; i++)
            if (d[i][k] > 0 && d[i][k] < vc )
                for (j = 1; j<= n; j++)
                        if (d[k][j] > 0 && d[k][j] < vc )
                           if (d[i][k] + d[k][j] < d[i][j] )
                                {
                                        d[i][j] = d[i][k] + d[k][j];
                                        p[i][j] = k;
                                }
}
```

Hàm xuất đường đi ngắn nhất từ x đến y cài đặt như sau:

```
void xuatdd(int x, int y)
{
    int r;
    if (p[x][y] == 0)
    {
        cout<<y<<" -> ";
        return;
    }
    else
    {
        r = p[x][y];
        xuatdd(x,r);
        xuatdd(r,y);
    }
}
```

Nhân tổ hợp các ma trận

- **Bài toán:**

Xét tích các ma trận: $A = A_1 \times ... \times A_n$, với giả thiết phép nhân có nghĩa.

Do tính kết hợp của phép nhân ma trận, các ma trận A_i có thể nhóm lại theo nhiều cách khác nhau, mà ma trận kết quả A không đổi. Tuy nhiên có sự khác biệt về chi phí khi thay đổi các tổ hợp các ma trận A_i. Ta lưu ý rằng tích 2 ma trận cấp (m×n) và (n×p) sẽ có chi phí là m×n×p.

Vấn đề là tìm trình tự thực hiện các ma trận sao cho có chi phí ít nhất.

Cho các ma trận, với các kích thước tương ứng:

A	×	B	×	C	×	D
30×1		1×40		40×10		10×25

Nhân các ma trận trên với các thứ tự sau:

Thöù töï	Chi phí
((AB)C)D	30×1×40 + 30×40×10 + 30×10×25 = 20700
(A(B(CD))	40×10×25 + 1×40×25 + 30×1×25 = 11750
(AB)(CD)	30×1×40 + 40×10×25 + 30×40×25 = 41200
A((BC)D)	1×40×10 + 1×10×25 + 30×1×25 = 1400

- **Ý tưởng:**
  - Ta giải bài toán bằng cách tiếp cận từ dưới lên. Ta sẽ tính toán và lưu trữ lời giải tối ưu cho từng phần nhỏ để tránh tính toán lại cho bài toán lớn hơn.
  - Trước hết là cho các bộ 2 ma trận, các bộ 3 ma trận...
    - Chẳng hạn, để tính A×B×C ta có thể tính theo các cách: (A×B)×C hoặc A×(B×C).
    - Nên chi phí để tính A×B×C là chi phí tính được từ 2 phần:
      - Phần một là chi phí kq1×C, với kq1 = A×B (chi phí này đã tính và được lưu trữ)
      - Phần hai là chi phí A × kq2, với kq2 = B×C (chi phí này đã được lưu trữ)
      - So sánh 2 phần trên và lưu trữ chi phí nhỏ hơn...

        ...

- **Thiết kế:**
  - Mấu chốt là tính chi phí nhân bộ các ma trận:

    $A_i \times ... \times A_j$, với $1 \le i < j \le n$,

    trong đó các bộ nhỏ hơn đã được tính và lưu trữ kết quả.
  - Với một cách tổ hợp các ma trận:

    $A_i \times ... \times A_j = (A_i \times ... \times A_k) \times (A_{k+1} \times ... \times A_j)$

    Chi phí để nhân các ma trận $A_i,...,A_j$ sẽ bằng tổng:
    - Chi phí để nhân $A_i \times ... \times A_k$ (= kq1),
    - Chi phí để nhân $A_{k+1} \times ... \times A_j$ (= kq2),
    - Chi phí kq1×kq2.
- **Gọi M_{ij} là chi phí nhỏ nhất để nhân bộ các ma trận $A_i \times ... \times A_j$, $1 \le i < j \le n$, khi đó:**
  - M_{ik} là chi phí nhỏ nhất để nhân bộ các ma trận $A_i \times ... \times A_k$
  - $M_{k+1,j}$ là chi phí nhỏ nhất để nhân bộ các ma trận $A_{k+1} \times ... \times A_j$
- **Vì ma trận kq1 cỡ $d_{i-1} \times d_k$ và kq2 có cỡ $d_k \times d_j$, nên chi phí để nhân kq1×kq2 là $d_{i-1} d_k d_j$. Vậy ta có:**

$$\begin{cases} M_{ij} = \underset{i \le k \le j-1}{Min} \{M_{ik} + M_{k+1,j} + d_{i-1} d_k d_j\}; 1 \le i < j \le n \\ M_{ii} = 0 \end{cases}$$

- Ta có thể xem M là ma trận tam giác trên: $(M_{ij})1 \le i < j \le n$. Ta cần tính và làm đầy các phần tử của ma trận này cho đến khi xác định được M_{1n}. Đường chéo chính bằng 0 tất cả.

- Tính M_{ij} , ta cần biết M_{ik} , $M_{k+1,j}$. Ta tính bảng dọc theo các đường chéo bắt đầu từ đường chéo kế trên đường chéo chính và thay đổi về hướng góc phải trên.

- Ta muốn biết thứ tự tốt nhất để nhân dãy các ma trận (theo nghĩa chi phí nhỏ nhất). Mỗi lần ta xác định được tổ hợp tốt nhất , ta dùng biến k để lưu trữ thứ tự này.

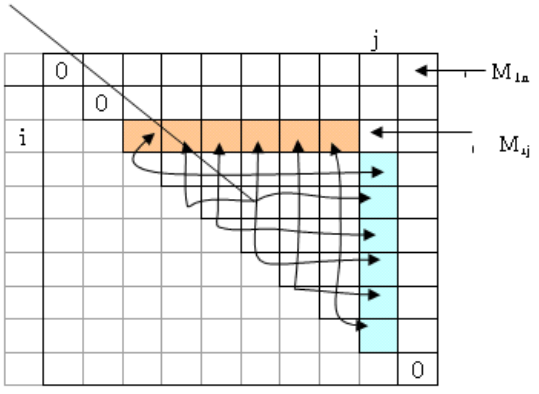
  Đó là Oij = k, với $M_{ik} + M_{k+1,j} + d_{i-1} d_k d_j$ đạt min.

- Các M_{ij} ta lưu trữ trong mảng 2 chiều M.

Các giá trị k ta lưu trữ trong mảng 2 chiều O.

Kích thước của các ma trận ta lưu trữ trong mảng 1 chiều d: A_i là ma trận có d_{i-1} hàng , d_i cột.

Bảng tính M_{ij}



Kết quả thuật toán, với:

$d_0 = 30$; $d_1 = 1$; $d_2 = 40$; $d_3 = 10$; $d_4 = 25$:

$$\begin{cases} M_{ij} = \underset{i \le k \le j-1}{Min}\{M_{ik} + M_{k+1,j} + d_{i-1}d_k d_j\}, 1 \le i < j \le n \\ M_{ii} = 0 \end{cases}$$

$$M = \begin{bmatrix} 0 & 1200 & 700 & 1400 \\ - & 0 & 400 & 650 \\ - & - & 0 & 10000 \\ - & - & - & 0 \end{bmatrix} \qquad O = \begin{bmatrix} - & 1 & 1 & 1 \\ - & - & 2 & 3 \\ - & - & - & 3 \\ - & - & - & - \end{bmatrix}$$

$$\begin{cases} M_{12} = \underset{1 \le k \le 2-1}{Min}\{M_{1k} + M_{k+1,2} + d_0 d_k d_2\} \\ = M_{11} + M_{22} + d_0 d_1 d_2 = 0 + 0 + 30*1*40 = 1200 \\ O_{12} = k = 1 \end{cases}$$

$$\begin{cases} M_{13} = \underset{1 \le k \le 3-1}{Min}\{M_{1k} + M_{k+1,3} + d_0 d_k d_3\} \\ = Min\{M_{11} + M_{23} + d_0 d_1 d_3, M_{12} + M_{33} + d_0 d_2 d_3\} \\ = Min\{0 + 400 + 300, 1200 + 0 + 12000\} \\ = Min\{700, 13200\} = 700 \\ O_{13} = k = 1 \end{cases}$$

$$\begin{cases} M_{23} = \underset{2 \le k \le 3-1}{Min}\{M_{2k} + M_{k+1,3} + d_1 d_k d_3\} \\ = M_{22} + M_{33} + d_1 d_2 d_3 = 0 + 0 + 1*40*10 = 400 \\ O_{23} = k = 2 \end{cases}$$

$$\begin{cases} M_{24} = \underset{2 \le k \le 4-1}{Min}\{M_{2k} + M_{k+1,4} + d_1 d_k d_4\} \\ = Min\{M_{22} + M_{34} + d_1 d_2 d_4, M_{23} + M_{44} + d_1 d_3 d_4\} \\ = Min\{0 + 10000 + 1000, 400 + 0 + 250\} = 650 \\ O_{24} = k = 3 \end{cases}$$

$$\begin{cases} M_{34} = \underset{3 \le k \le 4-1}{Min}\{M_{3k} + M_{k+1,4} + d_2 d_k d_4\} \\ = M_{33} + M_{44} + d_2 d_3 d_4 = 0 + 0 + 40*10*25 = 10000 \\ O_{34} = k = 3 \end{cases}$$

- **Thuật toán có thể viết như sau:**

Input d = (d_0,d_1,...,d_n) ;

Output M = (M_{ij}) ,
O = (O_{ij});

- **Mô tả:**

MO(d,n,O,M) ≡

int i, j, k, diag;

*** for (i = 1; i <= n; i++)**
M[i][i] = 0;

*** for (diag = 1; diag <= n-1; diag++)**
for (i = 1; i <= n - diag; i++)
+ j = i + diag;

+ $M_{ij} = \underset{i \le k \le j-1}{Min}\{M_{ik} + M_{k+1,j} + d_{i-1}d_k d_j\};$

+ O[i][j] = k; // vôùi Mij ñaït min

*** return m[1][n];**

$$\begin{cases} M_{ij} = \underset{i \le k \le j-1}{Min}\{M_{ik} + M_{k+1,j} + d_{i-1}d_k d_j\}, 1 \le i < j \le n \\ M_{ii} = 0 \end{cases}$$

Kết quả thuật toán, với:

$d_0 = 30$; $d_1 = 1$; $d_2 = 40$; $d_3 = 10$; $d_4 = 25$:

$$M = \begin{bmatrix} 0 & 1200 & 700 & 1400 \\ - & 0 & 400 & 650 \\ - & - & 0 & 10000 \\ - & - & - & 0 \end{bmatrix} \qquad O = \begin{bmatrix} - & 1 & 1 & 1 \\ - & - & 2 & 3 \\ - & - & - & 3 \\ - & - & - & - \end{bmatrix}$$

$$\begin{cases} M_{14} = \underset{1 \le k \le 4-1}{Min}\{M_{1k} + M_{k+1,4} + d_0 d_k d_4\} \\ = Min\{M_{11} + M_{24} + d_0 d_1 d_4, M_{12} + M_{34} + d_0 d_2 d_4, M_{13} + M_{44} + d_0 d_3 d_4\} \\ = Min\{0 + 650 + 750, 1200 + 10000 + 30000, 700 + 0 + 7500\} \\ = Min\{1400, 41200, 8200\} = 1400 \quad (= M_{11} + M_{24} + d_0 d_1 d_4) \\ O_{14} = k = 1 \end{cases}$$

- **Độ phức tạp của thuật toán:**

  $T(n) \in O(n^3)$