Một số vấn đề cơ sở của Tin học

Cấu trúc dữ liệu và thuật giải

Giáo viên: Tạ Thúc Nhu

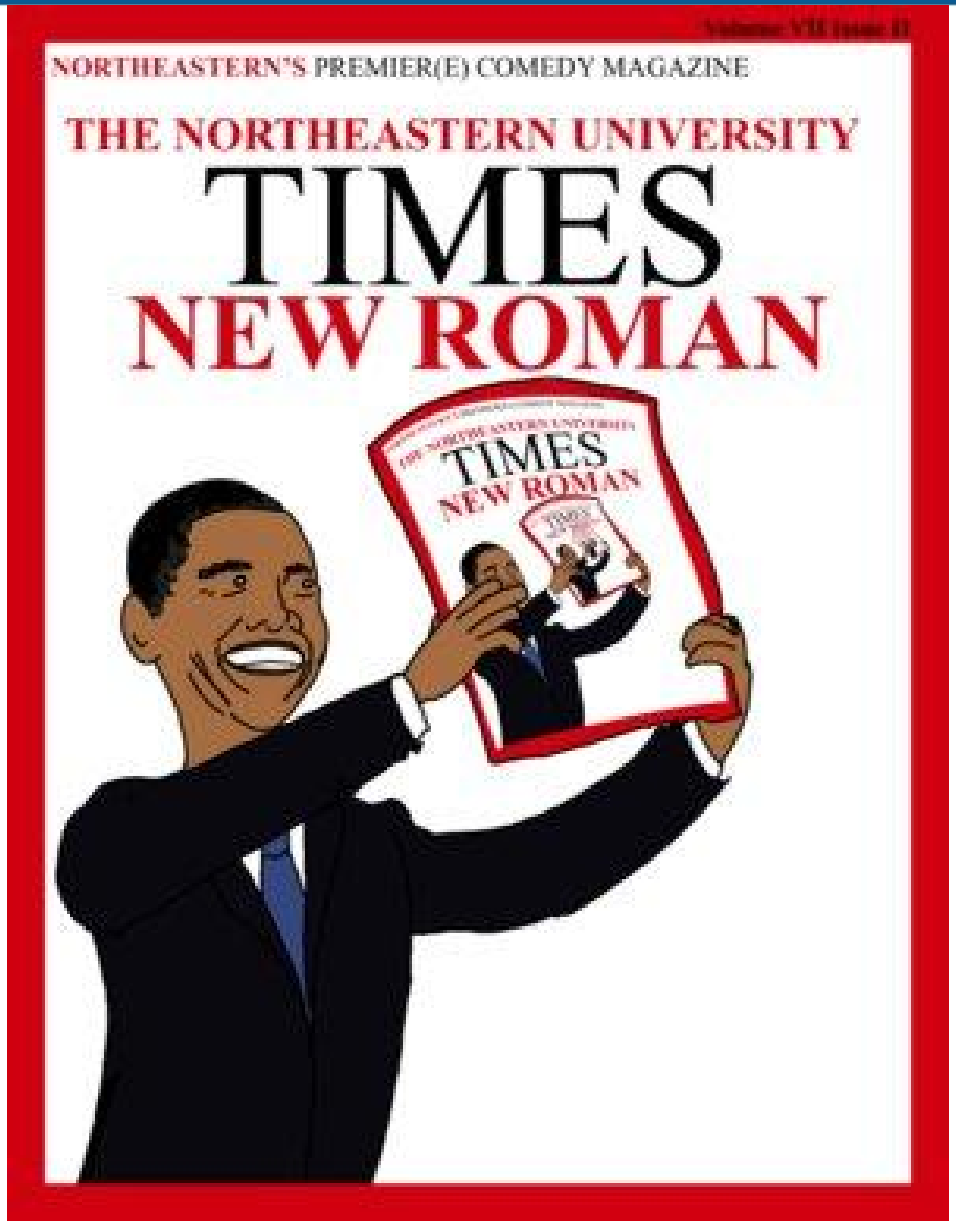
Khoa CNTT trường ĐH Lạc Hồng

ĐỆ QUI

RECURVE

Khái niệm Đệ Qui

Một khái niệm X được định nghĩa theo kiểu đệ qui nếu trong định nghĩa của X có sử dụng trực tiếp hoặc gián tiếp lại chính khái niệm X.



Ví dụ

1. **Định nghiã giai thừa của một số tự nhiên N, ký hiệu N!**
   a) N! = 1 nếu N = 0
   b) N! = N*(N-1)! nếu N > 0
2. **Định nghiã UCLN của 2 số M và N, ký hiệu: UCLN(M, N)**
   a) UCLN(M, N) = M nếu N = 0
   b) UCLN(M, N) = UCLN(N, phần dư của M/N) nếu N<>0
3. **Dãy số Fibonacci được định nghĩa như sau:**
   a) F(0) = F(1)= 1
   b) F(n) =F(n - 2) + F(n - 1) với n>=2.

Một định nghiã đệ qui phải có 2 thành phần:

- **Thành phần dừng:** Không chứa khái niệm đang định nghiã

  Ví dụ: N! = 1 nếu N = 0

- **Thành phần đệ qui:** có chứa khái niệm đang định nghiã

Định nghiã giai thừa của một số tự nhiên N, ký hiệu N!

1. **N!** = 1 nếu N = 0
2. **N!** = N * **(N - 1)!** nếu N > 0

Chương trình đệ qui

- **Một chương trình là đệ qui nếu trong chương trình có lời gọi thực hiện lại chính nó.**

Ví dụ: Định nghĩa hàm tính N! theo đệ qui.

```
int Fac(int N)
{
    if (N == 0)  return 1;
    return N * Fac(N - 1);
}
```

Ví dụ: Tính UCLN(M, N) theo thuật toán Euclide

a) UCLN(M, N) = M nếu N = 0

b) UCLN(M, N) = UCLN(N, M mod N) nếu N>0

```
int UCLN(int M, int N)
{
    if (N == 0) return M;
  return UCLN(N, M % N);
}
```

Một số dạng đệ qui thường gặp

1- Đệ qui tuyến tính.

Hàm F

{

Nếu (**thỏa điều kiện dừng)** thì <**thực hiện S>**

Ngược lại **<gọi đệ qui hàm F>**

}

Với S là thao tác không đệ qui .

2- Đệ qui nhị phân

Hàm F

{

Nếu (**thỏa điều kiện dừng)** thì <**thực hiện S>**

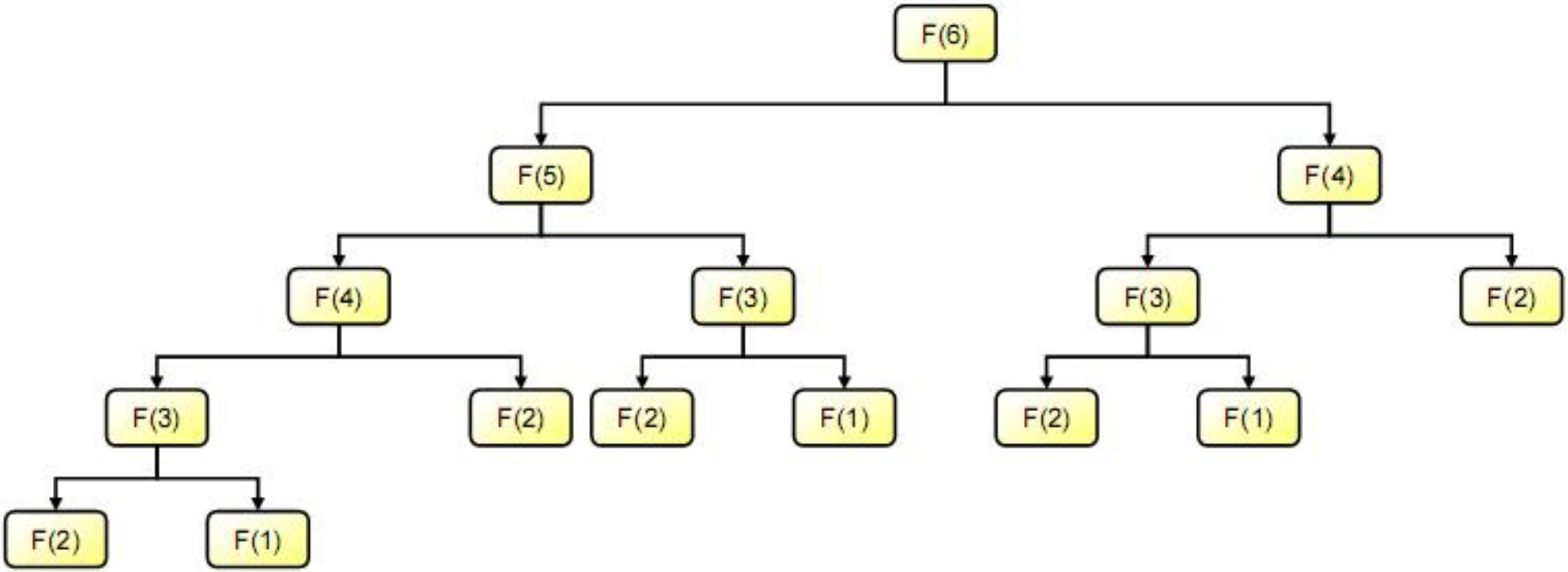
Ngược lại **{ gọi hàm F ; gọi hàm F }**

}

- Với S là thao tác không đệ qui

Ví dụ: Hàm tìm giá trị phần tử F(n) của dãy Fibonacci

```
int F(int n)
{
   if ( n < 2 ) return 1 ;
   return F(n -1) + F(n -2) ;
}
```


F(6)
F(5)
F(4)
F(4)
F(3)
F(3)
F(2)
F(3)
F(2)
F(2)
F(1)
F(2)
F(1)
F(2)
F(1)

3- Đệ qui phi tuyến:

Lời gọi đệ qui được thực hiện bên trong vòng lặp

Hàm F()

```
{
    for (giá tri đầu tới giá trị cuối)
    {
        Nếu (thỏa điều kiện dừng) thì <thực hiện S>
        Ngược lại < gọi đệ qui F >
    }
}
```

THUẬT GIẢI QUAY LUI

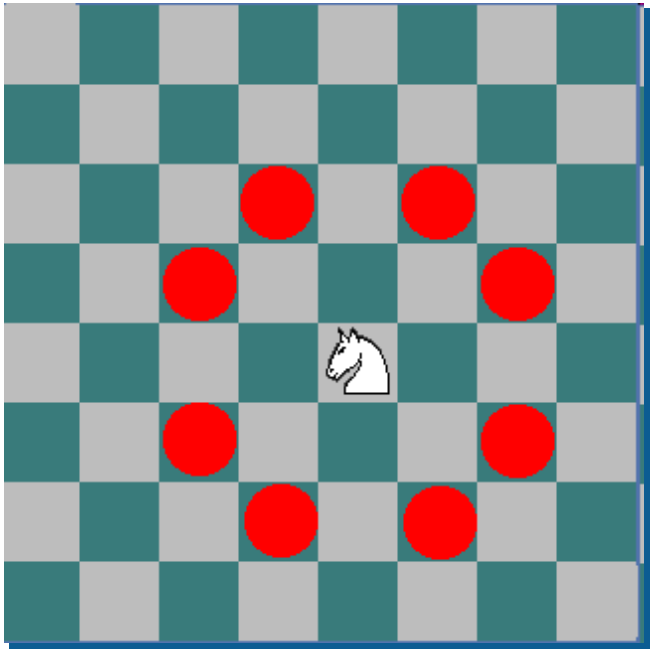
BACK TRACKING

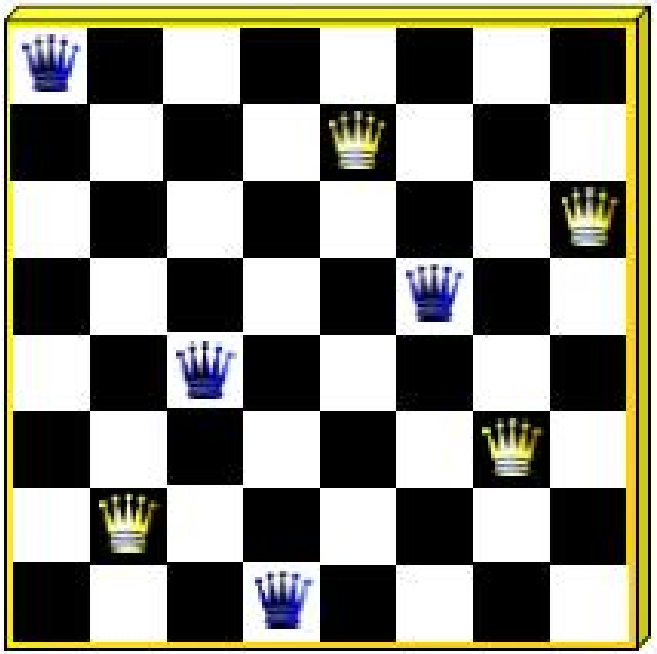
Công dụng:

- Giải bài toán liệt kê tất cả các lời giải thỏa yêu cầu bài toán.

Ví dụ:

1. Liệt kê các dãy nhị phân có độ dài N
2. Liệt kê các tập con k phần tử của tập S = {1, 2,.., n}
3. Liệt kê các chỉnh hợp không lặp chập k của tập S = {1, 2,.., n}



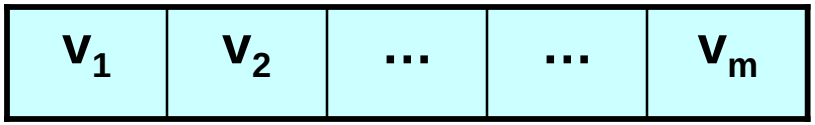


Tổng quan thuật giải Quay lui (BackTracking)

- Cấu trúc lời giải là một tập hợp có N phần tử cùng kiểu
- Việc xác định giá trị từng phần tử dựa trên tập giá trị đề cử.
- Để tìm một lời giải khác, ta quay lui chọn giá trị khác cho mỗi phần tử.

Cấu hình một lời giải

Tập giá trị đề cử



Thuật giải 1: *Xác định giá trị phần tử i của lời giải*


```
void Try( int i )
{
    for ( mọi v thuộc tập giá trị đề cử cho Xi )
    {
        Xi = v ;
        if ( Xi là phần tử cuối cùng)
            < Thông báo lời giải tìm được>;
        else
            Try( i+1); //Gọi đệ qui để xác định phần tử X[ i+1 ]
    }
}
```

Bài toán: Liệt kê các dãy nhị phân có độ dài n

Phân tích:

- **Cấu trúc lời giải là dãy có n phần tử: X[1] .. X[n]**
- **Mỗi phần tử X[i] nhận giá trị trong tập đề cử {0, 1}**

Cấu trúc lời giải

1	0	0	1	…	1

Tập giá trị đề cử

0	1

Xác định giá trị phần tử X[i] của dãy nhị phân

```
void XacDinhPhanTu(int i)
{
   for (int v = 0; v<= 1; v++)
   {
         X[ i ] = v;
         if ( i == n  )
               <Xuất dãy nhị phân tìm được>;
         else
               XacDinhPhanTu ( i + 1 );
   }
}
```

Bài toán: Liệt kê các tập con có K phần tử của tập S = {1, 2, .., n} (K <= n)

- **Cấu trúc lời giải là dãy có K phần tử: X[1] .. X[K]**
- **X[i] nhận giá trị trong tập S = { 1, 2, .., n }**
- **Yêu cầu lời giải:**
  - Giá trị các phần tử trong một lời giải phải phân biệt.

    X[i] <> X[j] với i <> j
  - Không xét thứ tự phần tử trong tập hợp

    {1, 2, 3} = {2, 1, 3}

Cách giải:

Đưa ra điều kiện cho mỗi tập con là :

$$1 <= X[1] < X[2] < .. < X[i] < .. < X[K-1] < X[K] <= N$$

- **Nhận xét:**

  X[K] <= N

  X[K-1] <= X[K] - 1 <= N – 1

  X[K-2] <= X[K] - 2 <= N – 2

  …

  X[i] = X[K-(K-i)] <= N - (K - i)

- **Giới hạn giá trị đề cử cho thành phần X[i] trong khoảng từ : X[i-1]+1 đến (N – K + i)**
- **Để điều này cũng đúng cho cả trường hợp i = 1, ta thêm vào X[0] = 0.**

Xác định giá trị phần tử Xi của một tập con

```
void XacDinhPhanTu(int i)
{
    for (int v = X[i-1]+1; v <= N-K+i; v++)
    {
        X[i] = v;
        if ( i == K  )
            <Thông báo tập con tìm được>;
        else
            XacDinhPhanTu( i + 1 );
     }
}
```

Thuật giải 2: Xác định giá trị cho phần tử i với giá trị đề cử có điều kiện

```
void Try( int i )
{
    for ( mọi v thuộc tập giá trị đề cử cho Xi)
        if ( Chấp nhận v )
        {    Xi = v;
             If ( Xi là phần tử cuối cùng )
                        < Thông báo lời giải tìm được>;
             else
             {          Ghi nhận giá trị v đã được chọn;
                        Try( i + 1); //Gọi đệ qui để xác định phần tử Xi+1
                        Bỏ ghi nhận giá trị v đã chọn; (nếu cần)
             }
        }
}
```

Bài toán: Liệt kê các chỉnh hợp không lặp chập k của tập S={1, 2,.., n}

- Cấu trúc lời giải là một dãy k phần tử: X[1], .., X[k]
- X[i] nhận giá trị trong tập S = { 1, 2, .., n }
- **Yêu cầu lời giải:**
  - Giá trị các phần tử trong một lời giải phải phân biệt.

    X[i] <> X[j] với i <> j
  - Có tính đến thứ tự phần tử trong chỉnh hợp:

    Chỉnh hợp {1, 2, 3} <> chỉnh hợp {2, 1, 3}
- Hướng giải quyết chung: tổ chức các biến trạng thái phục vụ cho việc kiểm tra giá trị đề cử:

  Dùng mảng F[1..n] ghi nhớ tình trạng đã chọn của từng giá trị trong tập S={1, 2, .., n}, với qui ước:
  - F[v] = 0 nếu v chưa chọn
  - F[v] = 1 nếu v đã chọn

Thuật giải xác định phần tử Xi của một chỉnh hợp không lặp chập k của tập S={1, 2,.., n}

```
void Try(int i)
{
    for (int v = 1; v<= n; v++)
        if (F[v] == 0)
        {       X[i] = v;
                if ( i == k  ) <Thông báo lời giải tìm được>;
                else
                {       F[v] = 1;
                        Try( i + 1 );
                        F[v] = 0;
                }
        }
}
```

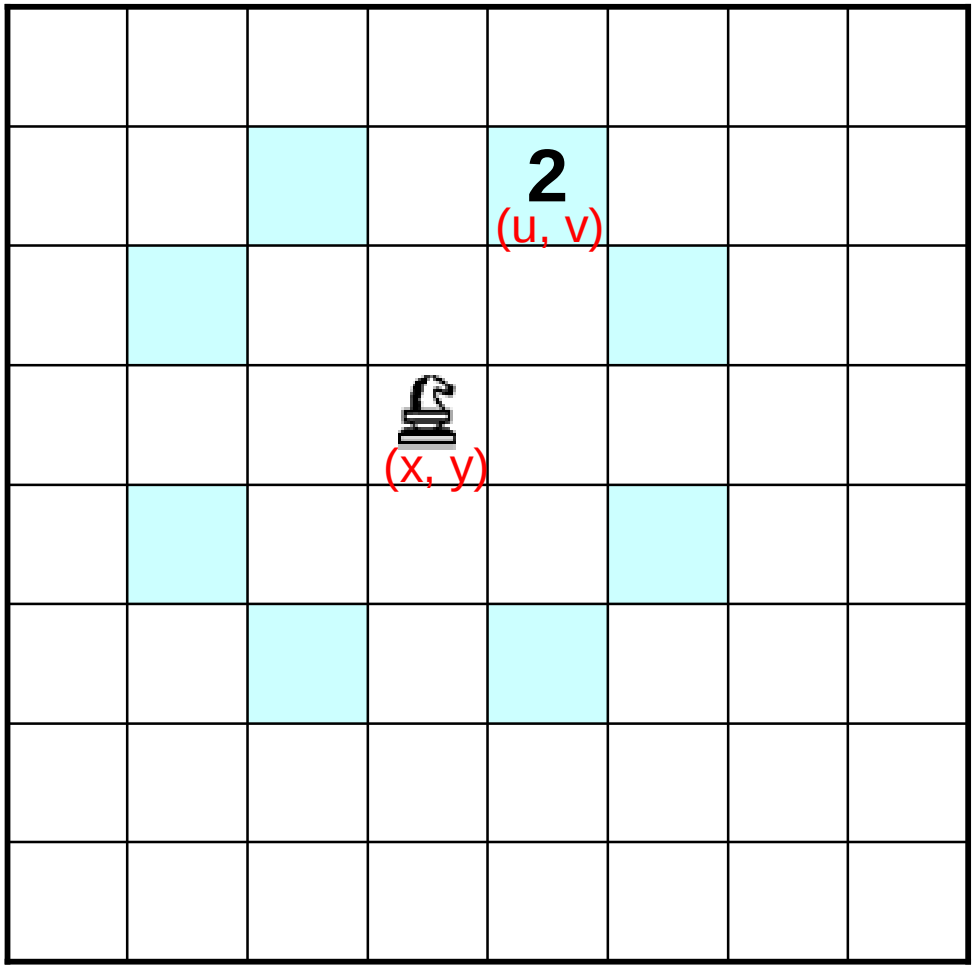
Bài toán Mã đi tuần: chỉ ra hành trình của quân Mã xuất phát từ một ô trên bàn cờ đi qua tất cả các ô còn lại của bàn cờ, mỗi ô đúng 1 lần.

Phân tích:

- Cấu trúc lời giải là BC[1..n][1..n] chứa số thứ tự hành trình của quân Mã.
- Tập giá trị đề cử chứa các giá trị dùng tính tọa độ các ô kế tiếp

  dx[1..8] = {-2,-1, 1, 2, 2, 1, -1, -2}

  dy[1..8] = { 1, 2, 2, 1, -1, -2, -2, -1}
- Điều kiện chọn giá trị đề cử cho bước đi thứ i phải đến một Ô :
  - Thuộc bàn cờ
  - Và chưa đi qua: BC[u, v] = 00



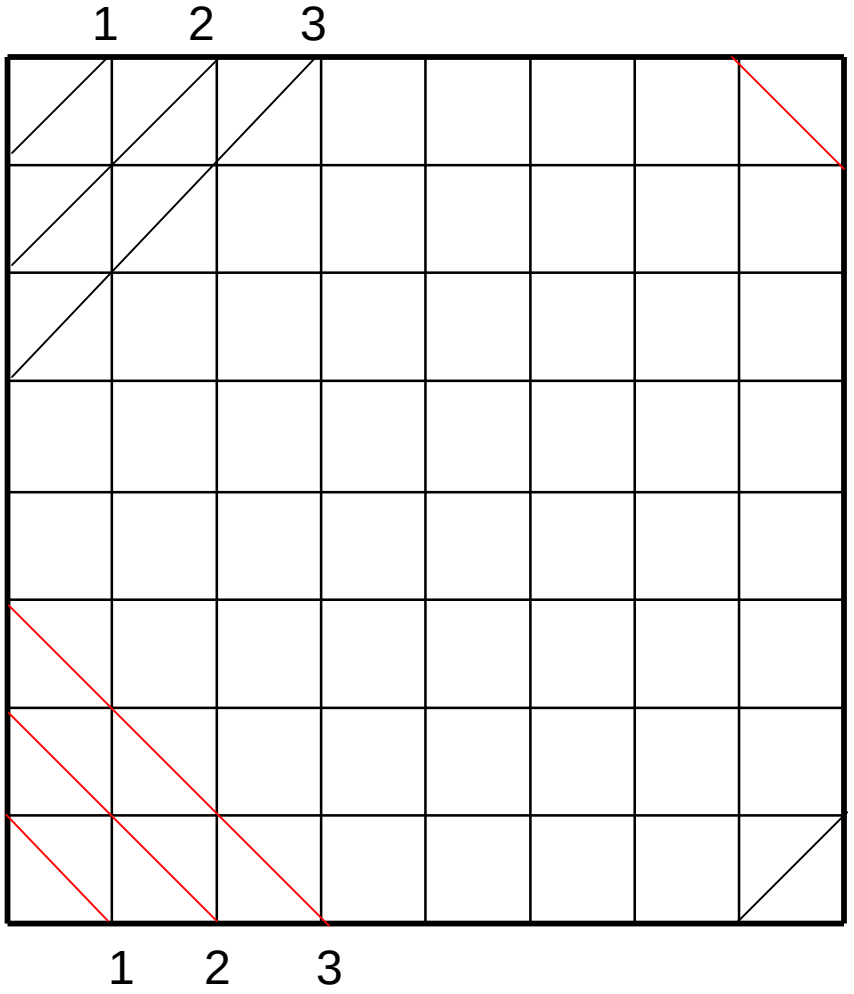
Thuật giải xác định bước đi thứ i của quân Mã

```
void Try(int i)
{   int  j,u,v;
    for ( j = 1; j <=  8 ; j++)
    {
        u =x + dx[ j ] ;    v = y + dy[ j ];
        if (u >= 1 && u <= n && v >= 1 && v<=n && BC[u,v] = 0)
        {
            BC[u, v] = i;
            if (i == n*n) <Thông báo lời giải tìm được>;
            else
            {       x = u;  y = v;
                    Try(i+1);
                    x = u - dx[ j ];    y = v - dy[ j ];
                    BC[u, v] = 0;
            }
        }
    }
}
```

Bài toán: Đặt 8 Quân Hậu trên bàn cờ quốc tế 8x8 sao cho các quân Hậu không ăn nhau

Phân tích:

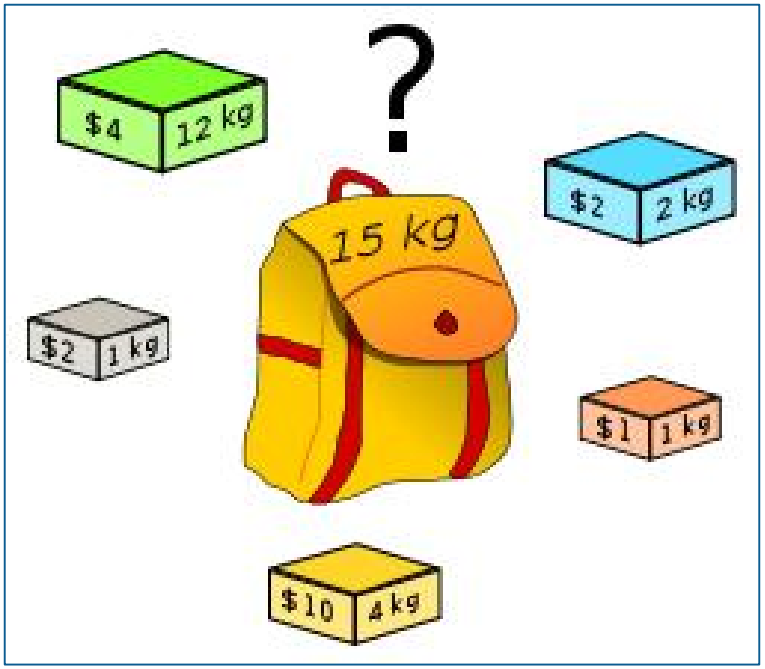
- Cấu trúc lời giải là mảng Dong[1..8] lưu chỉ số của cột có chứa quân hậu
- Tập giá trị đề cử {1..8 } là chỉ số của cột sẽ đặt quân hậu.
- Tổ chức các biến trạng thái để ghi nhận đường chéo nào đã có quân hậu:
  - Đường chéo ngược: **CN[1..15]**
    Ô(d, c) thuộc đường CN[c + d - 1]
  - Đường chéo xuôi: **CX[1..15]**
    Ô(d, c) thuộc đường CX[8 + c – d]
- Tổ chức biến trạng thái để ghi nhận cột đã có quân hậu: **Cot[1..8]**

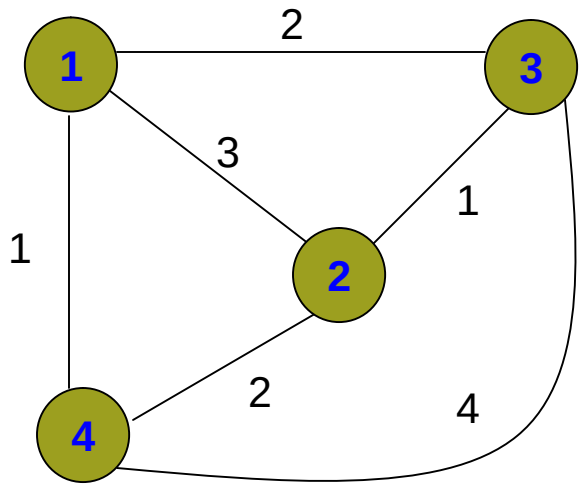


Thuật giải xác định cột đặt quân hậu trên dòng *d*

```
void DienDong(int d)
{     for (int c = 1; c <= 8; c++)
          If (Cot[c] = 0 && CN[c + d -1]=0 && CX[8+c-d] = 0)
          {          Dong[d] = c ;
                     if (d==8) <Thông báo lời giải tìm được>;
                     else
                     {          Cot[ c ] = CN[c + d -1] = CX[8+c-d] = 1;
                                DienDong(d+1);
                                Cot[ c ] = CN[c + d -1] = CX[8+c-d] = 0;
                     }
          }
}
```

Vận dụng thuật toán quay lui để giải các bài toán cực trị



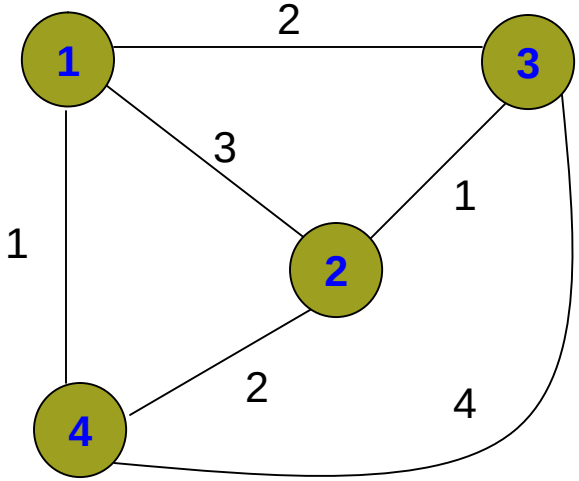


Bài toán người du lịch

Có n thành phố (được đánh số từ 1 đến n), chi phí đi từ thành phố i đến thành phố v là C[i, v].

Một người đi du lịch xuất phát từ một thành phố muốn đi thăm các thành phố khác, mỗi thành phố đúng một lần rồi quay về nơi xuất phát.

Hãy tìm 1 hành trình cho người du lịch để tổng chi phí theo hành trình này là ít nhất.

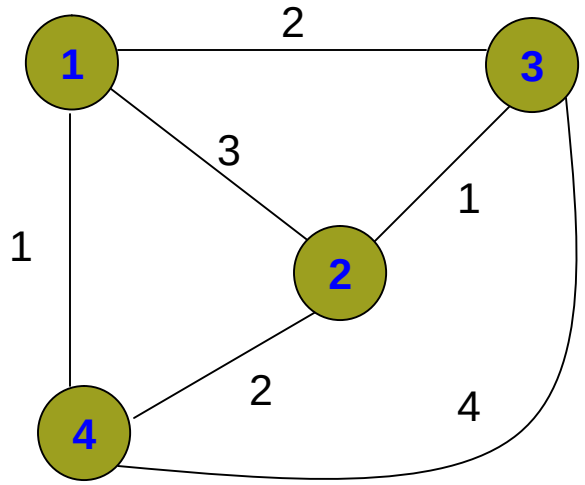


0	3	2	1
3	0	1	2
2	1	0	4
1	2	4	0

Ma trận chi phí C

Phân tích:

- **Mỗi hành trình là một chu trình đi qua các thành phố {1, 2, .., n}.**
- Cấu trúc lời giải là dãy X[1..n] trong đó:
  - X[1] = 1 được xem là thành phố xuất phát.
  - X[2..n] chứa hoán vị của dãy {2, 3, .. ,n}
- Tổng chi phí của một hành trình bằng: $S = \left(\sum_{i=1}^{n-1} C[X_i, X_{i+1}] \right) + C[X_n, X_1]$



0	3	2	1
3	0	1	2
2	1	0	4
1	2	4	0

Ma trận chi phí C

Thuật giải xác định phần tử X_i

```
void Try(int i)
{   for (int v = 2; v <= n; v++)
        if (F[ v ] == 0)
        {           X[ i ] = v;
                    if ( i == n  ) <Cập nhật hành trình tối ưu>;
                    else
                    {       F[ v ] = 1;
                            Try( i + 1 );
                            F[ v ] = 0;
                    }
         }
}
```

Cập nhật hành trình tối ưu:

Tổ chức ghi nhận hành trình tốt nhất:

- Best[1..n] : chứa hành trình có chi phí thấp nhất.
- Smin là tổng chi phí thấp nhất của hành trình Best.

Khởi tạo ban đầu cho Smin = Cmax * (n + 1),

trong đó Cmax là chi phí lớn nhất trong các chi phí C[i, j] đã cho

Thuật toán:

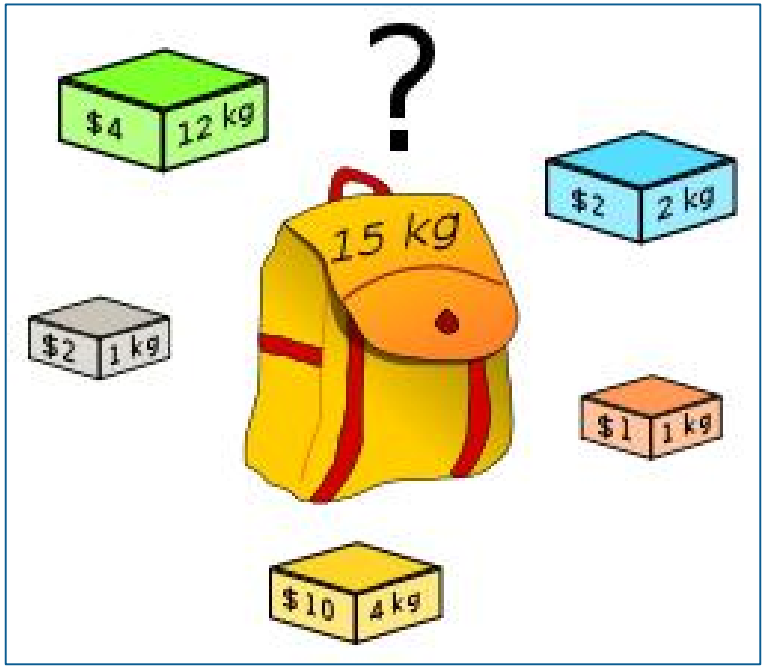
Tính $S = \left(\sum_{i=1}^{n-1} C[X_i, X_{i+1}] \right) + C[X_n, X_1]$

```
if ( S < Smin )
{
        Smin = S;
        Ghi nhận X là hành trình tốt nhất: Best = X;
}
```

Bài toán Knapsack

Có N đồ vật, đồ vật thứ i có trọng lượng là A_i và có giá trị là C_i (i=1,..,n).

Xác định các đồ vật cần bỏ vào ba lô sao cho tổng trọng lượng không quá M nhưng có tổng giá trị là lớn nhất.



Phân tích:

- Vật được chọn có cờ hiệu là 1, vật không được chọn có cờ hiệu là 0.
- Đưa về bài toán liệt kê dãy số nhị phân
- Cấu trúc lời giải là dãy: X[1..n]
  - X[i] = 1 : nếu vật i được chọn.
  - X[i] = 0 : nếu vật i không được chọn.

Xác định giá trị phần tử X[i] của dãy nhị phân

```
void Try(int i)
{
   for (int v =0; v<= 1; v++)
   {
      X[ i ] = v;
      if ( i == n )
         <Cập nhật vật chọn tối ưu>;
      else
         Try( i + 1 );
   }
}
```

Cập nhật vật chọn tối ưu:

Tổ chức ghi nhận danh sách vật chọn tốt nhất:

- Best[1..n] : chứa dãy nhị phân ghi nhận cách chọn tối ưu.
- Smax : tổng giá trị của các vật trong cách chọn tối ưu.
  Khởi tạo ban đầu cho Smax = 0
- S : tổng giá trị của các vật trong cách chọn hiện hành

$$S = \left(\sum_{i=1}^{n-1} X[i] * C[i] \right)$$

- W : Tổng trọng lượng của các vật trong cách chọn hiện hành

$$W = \left(\sum_{i=1}^{n-1} X[i] * A[i] \right)$$

Cập nhật vật chọn tối ưu

Thuật toán: Cập nhật vật chọn tối ưu

{

Tính $S = \left(\sum_{i=1}^{n-1} X[i] * C[i] \right)$

và

$W = \left(\sum_{i=1}^{n-1} X[i] * A[i] \right)$

if (W <= M && S > Smax)

{

Smax = S;

Ghi nhận X là lựa chọn tốt nhất: Best = X;

}

}

KỸ THUẬT NHÁNH CẬN

Công dụng

- **Giảm thời gian thực hiện bài toán tìm lời giải tốt nhất trong các lời giải được liệt kê bằng thuật toán quay lui.**
- **Ý tưởng:**
  - Thêm vào thuật toán quay lui khả năng đánh giá lời giải tối ưu ở từng bước xác định phần tử X_i.
  - Nếu tại bước thứ i đánh giá được lời giải sẽ không tối ưu thì quay lui ngay không cần phải tìm tiếp các phần tử khác.

Mô hình đánh giá nhánh cận trong thuật toán quay lui:

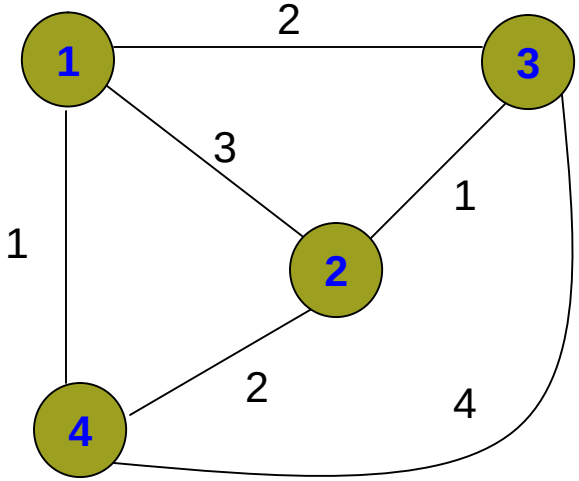
```
void Try( int i )
{
    for ( mọi v thuộc tập khả năng đề cử cho Xi)
        if ( Chấp nhận v )
        {
            Chọn v cho Xi;
            If (Xi là phần tử cuối cùng)
                <Cập nhật lời giải tối ưu>;
            else
            {
                Ghi nhận v đã chọn cho Xi ;
                if (còn hy vọng tìm ra lời giải tối ưu) Try( i+1);
                Bỏ ghi nhận v đã chọn cho Xi;
            }
        }
}
```

Bài toán người du lịch

Có n thành phố (được đánh số từ 1 đến n), chi phí đi từ thành phố i đến thành phố v là C[i, v].

Một người đi du lịch xuất phát từ một thành phố muốn đi thăm các thành phố khác, mỗi thành phố đúng một lần rồi quay về nơi xuất phát.

Hãy tìm 1 hành trình cho người du lịch để tổng chi phí theo hành trình này là ít nhất.



0	3	2	1
3	0	1	2
2	1	0	4
1	2	4	0

Ma trận chi phí C

Phân tích:

- Cấu trúc lời giải là dãy: X[1..n]
  - X[1] = 1 được xem là thành phố xuất phát.
  - X[2..n] là một hoán vị của các thành phố 2, .. ,n.
- Tổ chức ghi nhận lời giải tối ưu:
  - Best[1..n] : chứa hành trình có chi phí thấp nhất.
  - Smin là tổng chi phí thấp nhất của hành trình Best tìm được.

    Khởi tạo ban đầu cho Smin = Cmax * n

    trong đó Cmax là chi phí lớn nhất trong các chi phí C[i, v] đã cho

Thuật giải xác định phần tử X_i

```
void Try(int i)
{   for (int v = 2; v <= n; v++)
        if (F[ v ] == 0)
        {   X[ i ] = v;
            if ( i == n  ) <Cập nhật hành trình tối ưu>;
            else
            {       F[ v ] = 1;
                     if (còn hy vọng tìm ra lời giải tối ưu) Try( i + 1 );
                    F[ v ] = 0;
            }
        }
}
```

Tổ chức đánh giá lời giải tại bước thứ i:

- Thêm mảng T[1..n]: T[i] chứa tổng chi phí từ X[1] đến X[i]
  - T[i] = T[i -1] + C[X[i -1], X[i]];
  - Khởi tạo: T[1] = 0
- Đánh giá lời giải sau khi xác định X[i]:
  - Gọi Cmin là chi phí thấp nhất trong các chi phí C[i, v].
  - Nếu đi tiếp (n – i) thành phố nữa thì chi phí tối thiểu phải là: T[i] + (n - i)*Cmin.
  - Nếu T[i] + (n - i)*Cmin < Smin thì có hy vọng tìm được lời giải tối ưu, ngược lại thì chắc chắn không tìm được lời giải tối ưu.

Thuật giải xác định phần tử X_i

```
void Try(int i)
{   for (int v = 2; v <= n; v++)
        if (F[ v ] == 0)
        {    X[ i ] = v;
             T[ i ] = T[i -1] + C[X[i -1], v];
             if ( i == n  ) <Cập nhật hành trình tối ưu>;
             else
             {      F[ v ] = 1;
                    if (T[ i ] + (n-i)*Cmin < Smin) Try( i + 1 );
                    F[ v ] = 0;
             }
        }
}
```

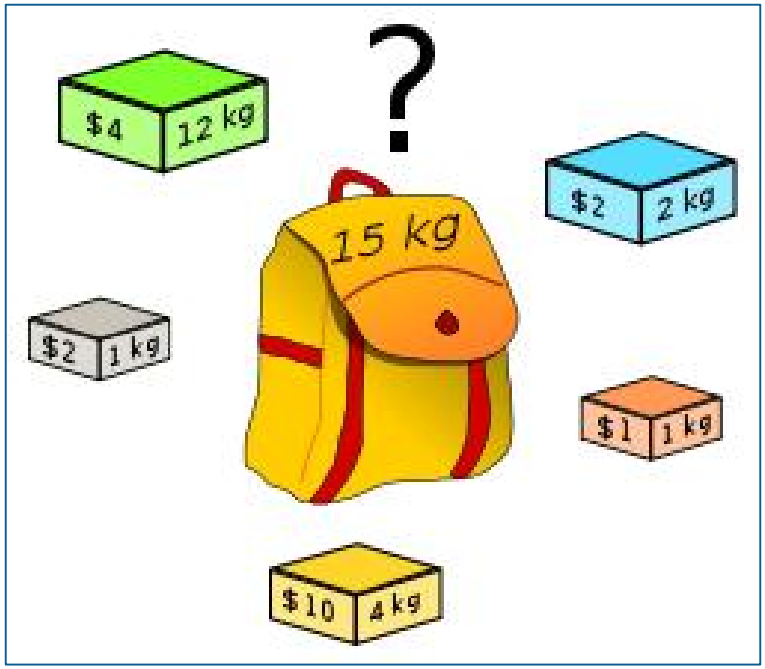
Cập nhật hành trình tối ưu:

Thuật toán:

```
{
    S = T[n] + C[X[n] , 1];
    if (S < Smin )
    {
        Smin = S;
        Ghi nhận X là hành trình tốt nhất: Best = X;
    }
}
```

Bài toán Knapsack

- Có N đồ vật, đồ vật thứ i có trọng lượng là Ai và có giá trị là Ci (i=1,..,n). Xác định các đồ vật cần bỏ vào ba lô sao cho tổng trọng lượng không quá M nhưng có tổng giá trị là lớn nhất.



Phân tích:

- Cấu trúc lời giải là dãy: X[1..n]
  - X[i] = 1 : nếu vật i được chọn.
  - X[i] = 0 : nếu vật i không được chọn.
- Tổ chức ghi nhận lời giải tối ưu:
  - Best[1..n] : chứa dãy nhị phân cho kết quả tối ưu.
  - Smax là tổng giá trị lớn nhất của các vật đã chọn. Khởi tạo ban đầu cho Smax = 0

Xác định giá trị phần tử X[i] của dãy nhị phân

```
void Try(int i)
{
   for (int v =0; v<= 1; v++)
   {
      X[ i ] = v;
      if ( i == n ) <Cập nhật vật chọn tối ưu>;
      else
         if (còn hy vọng tìm ra lời giải tối ưu) Try( i + 1 );
   }
}
```

Tổ chức đánh giá lời giải tại bước thứ i:

- Mảng T[0..n]: T[i] chứa tổng giá trị vật đã chọn trong miền [1..i]
  - T[i] = T[i-1] + X[i]*C[i]
- Mảng K[0..n]: K[i] chứa tổng khối lượng các vật đã chọn trong miền [1..i]
  - K[i] = K[i-1] + X[i]*A[i]
- Đánh giá lời giải sau khi xác định X[i]:
  - Gọi Cmax là giá trị lớn nhất trong các giá trị C[i].
  - Nếu chọn hết (n – i) vật còn lại thì tổng giá trị tối đa của lời giải là: T[i] + (n-i)*Cmax
  - Nếu K[i] <= M và T[i] + (n-i)*Cmax > Smax: thì có hy vọng tìm được lời giải tối ưu.

Xác định giá trị phần tử X[i] của dãy nhị phân

```
void Try(int i)
{
    for (int v =0; v<= 1; v++)
    {
        X[ i ] = v;
        T[i] = T[i-1] + X[i]*C[i]
        K[i] = K[i-1] + X[i]*A[i]
        if ( i == n ) <Cập nhật vật chọn tối ưu>;
        else
            if (K[i] <= M && T[i] + (n-i)*Cmax > Smax) Try( i + 1 );
    }
}
```

Cập nhật vật chọn tối ưu:

Thuật toán:

```
{
    if ( K[n] <= M && T[n] > Smax )
    {
        Smax = T[n];
        Ghi nhận X là lựa chọn tốt nhất: Best = X;
    }
}
```

Một số bài toán

1. Có N gói kẹo, gói thứ i có Ai cục kẹo. Xây dựng thuật toán chia N gói kẹo thành hai phần sao cho độ chênh lệch số kẹo giữa hai phần là ít nhất. Yêu cầu, không được thay đổi số kẹo trong mỗi gói ; in độ chênh lệch nhỏ nhất giữa hai phần có thể được và in danh sách gói kẹo của từng nhóm.
2. Cho 1 mảng gồm n các số nguyên a[1], a[2],.., a[n] và một số nguyên S. Hãy tìm tất cả các dãy con : 1 <= x1 < x2 < .. < xk <= n sao cho: a[x1] + a[x2] + ..+ a[xk] = S
3. Một dây chuyền sản xuất có N (N<=100) vị trí. Có N công nhân, cho biết năng suất của công nhân thứ i mà làm ở vị trí thứ j là Cij (Cij : Integer). Hãy sắp xếp N công nhân vào N vị trí sao cho đạt năng suất cao nhất.
4. Tính số cách và in tất cả các cách phân tích số tự nhiên N >1 thành tổng các số tự nhiên nhỏ hơn nó (mọi phân tích chỉ kể đúng một lần: 4+3+1 và 1+4+3 chỉ là một)
5. Hãy tìm tập hợp các dấu '+', '-' và không dấu giữa dãy số 123456789 sao cho được một biểu thức có giá trị bằng = N cho trước. Ví dụ: N = 280 ta có các tổ hợp sau: 1+2+345-67+8-9; 1+234-5+67-8-9; 123-4+5+67+89

6. Cho một dãy N số nguyên. Hãy loại bỏ khỏi dãy một số phần tử để được một dãy con, có ít nhất 2 phần tử, không giảm và dài nhất. In ra dãy con đó.

   Ví dụ: N = 10: 2 6 -7 5 8 1 -3 5 15 9

   Kết quả tìm được dãy con không giảm dài nhất có 4 phần tử:

   -7 -3 5 9

7. Một người cha mang theo số tiền là M vào một cửa hàng để mua K món quà để tặng cho các con. Trong cửa hàng có N mặt hàng, mặt hàng thứ i có giá tiền là Ai. Người cha cần chọn K (K < N) mặt hàng khác nhau để làm quà sao cho tổng số tiền của K mặt hàng này là lớn nhất nhưng không lớn hơn số tiền mang theo.

8. Mỗi hột xí ngầu có 6 mặt, mỗi mặt chứa từ 1 đến 6 dấu chấm. Liệt kê các kết quả phân biệt có thể có khi đổ cùng lúc 3 hột xí ngầu, không kể thứ tự xuất hiện trên các hột xí ngầu, ví dụ {1, 2, 3} và {2, 3, 1} là như nhau.

9. Trên bàn cờ ô vuông 4x4 xếp 8 quân cờ gồm 4 quân màu đen và 4 quân màu trắng sao cho trên mỗi hàng và mỗi cột có đúng một quân màu đen và 1 quân màu trắng. Thể hiện trên màn hình các cách sắp xếp này

10. Một cơ sở sản xuất cần phân công M nhân viên tham gia thực hiện N hợp đồng sản xuất sản phẩm (M >= N). Mỗi nhân viên chỉ tham gia thực hiện một hợp đồng. Người ta dự tính rằng, nếu phân công i nhân viên tham gia thực hiện hợp đồng j thì có thời gian hoàn thành hợp đồng là T[i,j]. Hãy tìm phương án phân công mỗi hợp đồng bao nhiêu nhân viên sao cho tổng số thời gian hoàn thành N hợp đồng là ít nhất.

11. Cho ma trận vuông cấp 8 chứa các số nguyên. Tìm giá trị lớn nhất của tổng 8 số hạng trên ma trận số trên sao cho 2 số hạng bất kỳ trong 8 số hạng trên không nằm trên cùng một hàng, không cùng nằm trên một cột và không cùng nằm trên đường chéo .

12. Cho một bảng A có M hàng, N cột ($3 \le M, N \le 50$), Mỗi phần tử của bảng là một số nguyên nhận giá trị từ 0 đến 99. Cho một số K ($2 \le K \le Min(M, N)$). Tìm K phần tử trong bảng A để tổng của K phần tử này là lớn nhất, với điều kiện là trên mỗi hàng chọn nhiều nhất một phần tử, mỗi cột chọn nhiều nhất một phần tử.