

# THIẾT KẾ GIẢI THUẬT

Chuyên đề này trình bày các chiến lược thiết kế thuật giải như: Quay lui (Backtracking), Nhánh và cận (Branch and Bound), Tham ăn (Greedy Method), Chia để trị (Divide and Conquer) và Quy hoạch động (Dynamic Programming). Đây là các chiến lược tổng quát, nhưng mỗi phương pháp chỉ áp dụng được cho một số lớp bài toán nhất định, chứ không tồn tại một phương pháp vạn năng để thiết kế thuật toán giải quyết mọi bài toán. Các phương pháp thiết kế thuật toán trên chỉ là chiến lược, có tính định hướng tìm thuật toán. Việc áp dụng chiến lược để tìm ra thuật toán cho một bài toán cụ thể còn đòi hỏi nhiều sáng tạo. Trong chuyên đề này, ngoài phần trình bày về các phương pháp, chuyên đề còn có những ví dụ cụ thể, cùng với thuật giải và cài đặt, để có cái nhìn chi tiết từ việc thiết kế giải thuật đến xây dựng chương trình.

## 1. Quay lui (Backtracking)

Quay lui, vét cạn, thử sai, duyệt ... là một số tên gọi tuy không đồng nghĩa nhưng cùng chỉ một phương pháp trong tin học: *tìm nghiệm của một bài toán bằng cách xem xét tất cả các phương án có thể*. Đối với con người phương pháp này thường là không khả thi vì số phương án cần kiểm tra lớn. Tuy nhiên đối với máy tính, nhờ tốc độ xử lí nhanh, máy tính có thể giải rất nhiều bài toán bằng phương pháp quay, lui vét cạn.

Ưu điểm của phương pháp quay lui, vét cạn là *luôn đảm bảo tìm ra nghiệm đúng, chính xác*. Tuy nhiên, hạn chế của phương pháp này là *thời gian thực thi lâu, độ phức tạp lớn*. Do đó vét cạn thường chỉ phù hợp với các bài toán có kích thước nhỏ.

### 1.1. Phương pháp

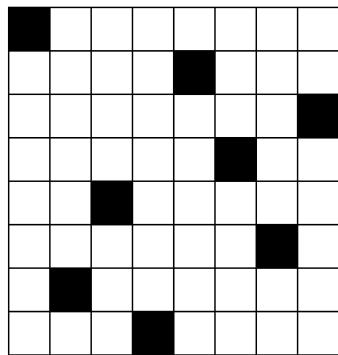
Trong nhiều bài toán, việc tìm nghiệm có thể quy về việc tìm vector hữu hạn  $(x_1, x_2, \dots, x_n, \dots)$ , độ dài vector có thể xác định trước hoặc không. Vector này cần phải thoả mãn một số điều kiện tùy thuộc vào yêu cầu của bài toán. Các thành phần  $x_i$  được chọn ra từ tập hữu hạn  $A_i$ .

Tuỳ từng trường hợp mà bài toán có thể yêu cầu: tìm một nghiệm, tìm tất cả nghiệm hoặc đếm số nghiệm.

**Ví dụ:** Bài toán 8 quân hậu.

Cần đặt 8 quân hậu vào bàn cờ vua  $8 \times 8$ , sao cho chúng không tấn công nhau, tức là không có hai quân hậu nào cùng hàng, cùng cột hoặc cùng đường chéo.

Ví dụ: hình bên là một cách đặt hậu thỏa mãn yêu cầu bài toán, các ô được tô màu là vị trí đặt hậu.



Do các quân hậu phải nằm trên các hàng khác nhau, ta đánh số các quân hậu từ 1 đến 8, quân hậu  $i$  là quân hậu nằm trên hàng thứ  $i$  ( $i=1,2,\dots,8$ ). Gọi  $x_i$  là cột mà quân hậu  $i$  đứng. Như vậy nghiệm của bài toán là vector  $(x_1, x_2, \dots, x_8)$ , trong đó  $1 \leq x_i \leq 8$ , tức là  $x_i$  được chọn từ tập  $A_i = \{1, 2, \dots, 8\}$ . Vector  $(x_1, x_2, \dots, x_8)$  là nghiệm nếu  $x_i \neq x_j$  và hai ô  $(i, x_i), (j, x_j)$  không nằm trên cùng một đường chéo. Ví dụ:  $(1,5,8,6,3,7,2,4)$  là một nghiệm.

Tư tưởng của phương pháp quay lui vét cạn như sau: Ta xây dựng vector nghiệm dần từng bước, bắt đầu từ vector không ( ). Thành phần đầu tiên  $x_1$  được chọn ra từ tập  $S_1 = A_1$ . Giả sử đã chọn được các thành phần  $x_1, x_2, \dots, x_{i-1}$  thì từ các điều kiện của bài toán ta xác định được tập  $S_i$  (các ứng cử viên có thể chọn làm thành phần  $x_i$ ,  $S_i$  là tập con của  $A_i$ ). Chọn một phần tử  $x_i$  từ  $S_i$  ta mở rộng nghiệm được  $x_1, x_2, \dots, x_i$ . Lặp lại quá trình trên để tiếp tục mở rộng nghiệm. Nếu không thể chọn được thành phần  $x_{i+1}$  ( $S_{i+1}$  rỗng) thì ta quay lại chọn một phần tử khác của  $S_i$  cho  $x_i$ . Nếu không còn một phần tử nào khác của  $S_i$  ta quay lại chọn một phần tử khác của  $S_{i-1}$  làm  $x_{i-1}$  và cứ thế tiếp tục. Trong quá trình mở rộng nghiệm, ta phải kiểm tra nghiệm đang xây dựng đã là nghiệm của bài toán chưa. Nếu chỉ cần tìm một nghiệm thì khi gặp nghiệm ta dừng lại. Còn nếu cần tìm tất cả các nghiệm thì quá trình chỉ dừng lại khi tất cả các khả năng lựa chọn của các thành phần của vector nghiệm đã bị vét cạn.

Lược đồ tổng quát của thuật toán quay lui vét cạn có thể biểu diễn bởi thủ tục *Backtrack* sau:

```
procedure Backtrack;
begin
  S1:=A1;
  k:=1;
  while k>0 do begin
```

```

while  $S_k \neq \emptyset$  do begin
    <chọn  $x_k \in S_i$ >;
     $S_k := S_k - \{x_k\}$ ;
    if ( $x_1, x_2, \dots, x_k$ ) là nghiệm then <Đưa ra nghiệm>;
         $k := k + 1$ ;
        <Xác định  $S_k$ >;
    end;
     $k := k - 1$ ; // quay lui
end;
end;

```

Trên thực tế, thuật toán quay lui vét cạn thường được dùng bằng mô hình đệ quy như sau:

```

procedure Backtrack(i); // xây dựng thành phần thứ i
begin
<Xác định  $S_i$ >;
for  $x_i \in S_i$  do begin
    <ghi nhận thành phần thứ i>;
    if (tìm thấy nghiệm) then <Đưa ra nghiệm>
    else Backtrack(i+1);
    <loại thành phần i>;
end;
end;

```

Khi áp dụng lược đồ tổng quát của thuật toán quay lui cho các bài toán cụ thể, có ba vấn đề quan trọng cần làm:

- Tìm cách biểu diễn nghiệm của bài toán dưới dạng một dãy các đối tượng được chọn dần từng bước ( $x_1, x_2, \dots, x_i, \dots$ ).
- Xác định tập  $S_i$  các ứng cử viên được chọn làm thành phần thứ i của nghiệm. Chọn cách thích hợp để biểu diễn  $S_i$ .
- Tìm các điều kiện để một vector đã chọn là nghiệm của bài toán.

## 1.2. Một số ví dụ áp dụng

### 1.2.1. Tổ hợp

Một tổ hợp chập k của n là một tập con k phần tử của tập n phần tử.

Chẳng hạn tập {1,2,3,4} có các tổ hợp chập 2 là:

$$\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}$$

Vì trong tập hợp các phần tử không phân biệt thứ tự nên tập  $\{1,2\}$  cũng là tập  $\{2,1\}$ , do đó, ta coi chúng chỉ là một tổ hợp.

Bài toán: Hãy xác định tất cả các tổ hợp chập  $k$  của tập  $n$  phần tử. Để đơn giản ta chỉ xét bài toán tìm các tổ hợp của tập các số nguyên từ 1 đến  $n$ . Đối với một tập hữu hạn bất kì, bằng cách đánh số thứ tự của các phần tử, ta cũng đưa được về bài toán đối với tập các số nguyên từ 1 đến  $n$ .

Nghiệm của bài toán tìm các tổ hợp chập  $k$  của  $n$  phần tử phải thoả mãn các điều kiện sau:

- Là một vector  $X = (x_1, x_2, \dots, x_k)$
- $x_i$  lấy giá trị trong tập  $\{1, 2, \dots, n\}$
- Ràng buộc:  $x_i < x_{i+1}$  với mọi giá trị  $i$  từ 1 đến  $k - 1$  (vì tập hợp không phân biệt thứ tự phần tử nên ta sắp xếp các phần tử theo thứ tự tăng dần).

Ta có:  $1 \leq x_1 < x_2 < \dots < x_k \leq n$ , do đó tập  $S_i$  (tập các ứng cử viên được chọn làm thành phần thứ  $i$ ) là từ  $x_{i-1} + 1$  đến  $(n - k + i)$ . Để điều này đúng cho cả trường hợp  $i = 1$ , ta thêm vào  $x_0 = 0$ .

Sau đây là chương trình hoàn chỉnh, chương trình sử dụng mô hình đệ quy để sinh tất cả các tổ hợp chập  $k$  của  $n$ .

```
program ToHop;
const MAX      = 20;
type  vector   = array[0..MAX] of longint;
var   x         : vector;
      n, k       : longint;
procedure GhiNghiem(x:vector);
var i : longint;
begin
  for i:=1 to k do write(x[i], ' ');
  writeln;
end;
procedure ToHop(i:longint);
var j:longint;
begin
  for j := x[i-1]+1 to n-k+i do begin
    x[i] := j;
    if i=k then GhiNghiem(x)
    else ToHop(i+1);
  end;
end;
```

```

    end;
end;
BEGIN
    write('Nhập n, k:'); readln(n, k);
    x[0]:=0;
    ToHop(1);
END.

```

Ví dụ về Input / Output của chương trình:

n = 4, k=2	1. 1 2 2. 1 3 3. 1 4 4. 2 3 5. 2 4 6. 3 4
------------	--

Theo công thức, số lượng tổ hợp chập k=2 của n=4 là:

$$C_n^k = \frac{n(n-1) \dots (n-k+1)}{k!} = \frac{n!}{k!(n-k)!} = C_4^2 = 6$$

### 1.2.2. Chính hợp lặp

Chính hợp lặp chập k của n là một dãy k thành phần, mỗi thành phần là một phần tử của tập n phần tử, có xét đến thứ tự và không yêu cầu các thành phần khác nhau.

Một ví dụ dễ thấy nhất của chính hợp lặp là các **dãy nhị phân**. Một dãy nhị phân độ dài m là một chính hợp lặp chập m của tập 2 phần tử {0,1}. Các dãy nhị phân độ dài 3:

000, 001, 010, 011, 100, 101, 110, 111.

Vì có xét thứ tự nên dãy 101 và dãy 011 là 2 dãy khác nhau.

Như vậy, bài toán xác định tất cả các chính hợp lặp chập k của tập n phần tử yêu cầu tìm các nghiệm như sau:

- Là một vector  $X = (x_1, x_2, \dots, x_k)$
- $x_i$  lấy giá trị trong tập  $\{1, 2, \dots, n\}$
- Không có ràng buộc nào giữa các thành phần.

Chú ý là cũng như bài toán tìm tổ hợp, ta chỉ xét đối với tập n số nguyên từ 1 đến n. Nếu phải tìm chỉnh hợp không phải là tập các số nguyên từ 1 đến n thì ta có thể đánh số các phần tử của tập đó để đưa về tập các số nguyên từ 1 đến n.

{sử dụng một mảng  $x[1..n]$  để biểu diễn chỉnh hợp lặp}.

Thủ tục để quy sau sinh tất cả chỉnh hợp lặp chập k của n}

```
procedure ChinhHopLap(i:longint);
var j:longint;
begin
    for j := 1 to n do begin
        x[i] := j;
        if i=k then GhiNghiem(x)
        else ChinhHopLap(i+1);
    end;
end;
```

Ví dụ về Input/Output của chương trình:

n = 2, k=3	<table style="border-collapse: collapse; margin: auto;"> <tr><td>1.</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>2.</td><td>1</td><td>1</td><td>2</td></tr> <tr><td>3.</td><td>1</td><td>2</td><td>1</td></tr> <tr><td>4.</td><td>1</td><td>2</td><td>2</td></tr> <tr><td>5.</td><td>2</td><td>1</td><td>1</td></tr> <tr><td>6.</td><td>2</td><td>1</td><td>2</td></tr> <tr><td>7.</td><td>2</td><td>2</td><td>1</td></tr> <tr><td>8.</td><td>2</td><td>2</td><td>2</td></tr> </table>	1.	1	1	1	2.	1	1	2	3.	1	2	1	4.	1	2	2	5.	2	1	1	6.	2	1	2	7.	2	2	1	8.	2	2	2
1.	1	1	1																														
2.	1	1	2																														
3.	1	2	1																														
4.	1	2	2																														
5.	2	1	1																														
6.	2	1	2																														
7.	2	2	1																														
8.	2	2	2																														

Theo công thức, số lượng chỉnh hợp lặp chập k=3 của n=2 là:

$$\bar{A}_n^k = n^k = 2^3 = 8$$

### 1.2.3. Chỉnh hợp không lặp

Khác với chỉnh hợp lặp là các thành phần được phép lặp lại (tức là có thể giống nhau), chỉnh hợp không lặp chập k của tập n ( $k \leq n$ ) phần tử cũng là một dãy k thành phần lấy từ tập n phần tử có xét thứ tự nhưng các thành phần không được phép giống nhau.

Ví dụ: Có n người, một cách chọn ra k người để xếp thành một hàng là một chỉnh hợp không lặp chập k của n.

Một trường hợp đặc biệt của chỉnh hợp không lặp là **hoán vị**. Hoán vị của một tập n phần tử là một chỉnh hợp không lặp chập n của n. Nói một cách trực quan thì

hoán vị của tập n phần tử là phép thay đổi vị trí của các phần tử (do đó mới gọi là hoán vị).

Nghiệm của bài toán tìm các chỉnh hợp không lặp chập k của tập n số nguyên từ 1 đến n là các vector  $X$  thoả mãn các điều kiện:

- $X$  có k thành phần:  $X = (x_1, x_2, \dots, x_k)$
- $x_i$  lấy giá trị trong tập  $\{1, 2, \dots, n\}$
- Ràng buộc: các giá trị  $x_i$  đôi một khác nhau, tức là  $x_i \neq x_j$  với mọi  $i \neq j$ .

Sau đây là chương trình hoàn chỉnh, chương trình sử dụng mô hình đệ quy để sinh tất cả các chỉnh hợp không lặp chập k của n phần tử.

```
program ChinhHopKhongLap;
const MAX = 20;
type vector = array[0..MAX] of longint;
var x :vector;
     d :array[1..MAX] of longint; { mảng d để kiểm
soát ràng buộc các giá trị x_i đôi một khác nhau, x_i ≠ x_j với mọi i ≠ j}
     n, k :longint;
procedure GhiNghiem(x:vector);
var i :longint;
begin
    for i:=1 to k do write(x[i], ' ');
    writeln;
end;
procedure ChinhHopKhongLap(i:longint);
var j:longint;
begin
    for j := 1 to n do
        if d[j]=0 then begin
            x[i] := j;
            d[j] := 1;
            if i=k then GhiNghiem(x)
            else ChinhHopKhongLap(i+1);
            d[j] := 0;
        end;
    end;
end;
BEGIN
    write('Nhập n, k(k<=n):'); readln(n,k);
```

```

fillchar(d, sizeof(d), 0);
ChinhHopKhongLap(1);
END.

```

Ví dụ về Input / Output của chương trình:

$n = 3, k=3$	<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr><td>1.</td><td>1</td><td>2</td><td>3</td></tr> <tr><td>2.</td><td>1</td><td>3</td><td>2</td></tr> <tr><td>3.</td><td>2</td><td>1</td><td>3</td></tr> <tr><td>4.</td><td>2</td><td>3</td><td>1</td></tr> <tr><td>5.</td><td>3</td><td>1</td><td>2</td></tr> <tr><td>6.</td><td>3</td><td>2</td><td>1</td></tr> </table>	1.	1	2	3	2.	1	3	2	3.	2	1	3	4.	2	3	1	5.	3	1	2	6.	3	2	1
1.	1	2	3																						
2.	1	3	2																						
3.	2	1	3																						
4.	2	3	1																						
5.	3	1	2																						
6.	3	2	1																						

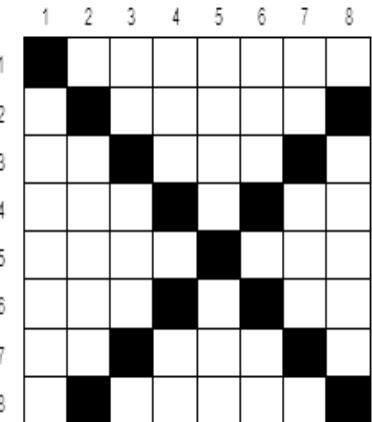
Theo công thức, số lượng chỉnh hợp không lặp chập  $k=3$  của  $n=3$  là:

$$A_n^k = n(n-1)\dots(n-k+1) = \frac{n!}{(n-k)!} = 6$$

#### 1.2.4. Bài toán xếp 8 quân hậu

Trong bài toán 8 quân hậu, nghiệm của bài toán có thể biểu diễn dưới dạng vector  $(x_1, x_2, \dots, x_8)$  thoả mãn:

- 1)  $x_i$  là tọa độ cột của quân hậu đang đứng ở dòng thứ  $i$ ,  $x_i \in \{1, 2, \dots, 8\}$ .
- 2) Các quân hậu không đứng cùng cột tức là  $x_i \neq x_j$  với  $i \neq j$ .
- 3) Có thể dễ dàng nhận ra rằng hai ô  $(x_1, y_1)$  và  $(x_2, y_2)$  nằm trên cùng đường chéo chính (trên xuống dưới) nếu:  $x_1 - y_1 = x_2 - y_2$ , hai ô  $(x_1, y_1)$  và  $(x_2, y_2)$  nằm trên cùng đường chéo phụ (từ dưới lên trên) nếu:  $x_1 + y_1 = x_2 + y_2$ , nên điều kiện để hai quân hậu xếp ở hai ô  $(i, x_i), (j, x_j)$  không nằm trên cùng một đường chéo là:  $\begin{cases} (i - x_i) \neq (j - x_j) \\ (i + x_i) \neq (j + x_j) \end{cases}$



Do đó, khi đã chọn được  $(x_1, x_2, \dots, x_{k-1})$  thì  $x_k$  được chọn phải thoả mãn các điều kiện:

$$\begin{cases} x_k \neq x_i \\ k - x_k \neq i - x_i & \text{với mọi } 1 \leq i < k \\ k + x_k \neq i + x_i \end{cases}$$

Sau đây là chương trình đầy đủ, để liệt kê tất cả các cách xếp 8 quân hậu lên bàn cờ vua 8×8.

```
program XepHau;
type vector = array[1..8] of longint;
var x :vector;
procedure GhiNghiem(x:vector);
var i :longint;
begin
  for i:=1 to 8 do write(x[i], ' ');
  writeln;
end;
procedure XepHau(k:longint);
var Sk :array[1..8] of longint;
  xk, i, nSk :longint;
  ok :boolean;
begin
  {Xác định tập  $S_k$  là tập các ứng cử viên có thể chọn làm thành phần  $x_k$ }
  nSk:=0; {lực lượng của tập  $S_k$ }
  for xk:=1 to 8 do {thứ lần lượt từng giá trị 1, 2, ..., 8}
    begin
      ok:=true;
      {kiểm tra giá trị có thể chọn làm ứng cử viên cho  $x_k$  được hay không}
      for i:=1 to k-1 do
        if not ((xk<>x[i]) and (k-xk<>i-x[i]) and (k+xk<>i+x[i])) then
          begin
            ok:=false;
            break;
          end;
      if ok then begin {có thể chọn làm ứng cử viên cho  $x_k$ , kết nạp vào tập  $S_k$ }
        inc(nSk);
        Sk[nSk]:=xk;
      end;
    end;
  {chọn giá trị  $x_k$  từ tập  $S_k$ }
  for i:=1 to nSk do begin
    x[k]:=Sk[i];
  end;
end;
```

```

        if k=8 then GhiNghiem(x)
        else XepHau(k+1);
        x[k]:=0;
    end;
end;
BEGIN
    XepHau(1);
END.

```

Việc xác định tập  $S_k$  có thể thực hiện đơn giản và hiệu quả hơn bằng cách sử dụng các mảng đánh dấu. Cụ thể, khi ta đặt hậu i ở ô  $(i, x[i])$ , ta sẽ đánh dấu cột  $x[i]$  (dùng một mảng đánh dấu như ở bài toán *chỉnh hợp không lặp*), đánh dấu đường chéo chính ( $i-x[i]$ ) và đánh dấu đường chéo phụ ( $i+x[i]$ ).

```

const n = 8;
type vector = array[1..n] of longint;
var cot :array[1..n] of longint;
    cheoChinh :array[1-n..n-1] of longint;
    cheoPhu :array[1+1..n+n] of longint;
    x :vector;
procedure GhiNghiem(x:vector);
var i :longint;
begin
    for i:=1 to n do write(x[i], ' ');
    writeln;
end;
procedure xepHau(k:longint);
var i :longint;
begin
    for i:=1 to n do
        if (cot[i]=0) and (cheoChinh[k-i]=0) and
(cheoPhu[k+i]=0) then
            begin
                x[k]:=i;
                cot[i]:=1;
                cheoChinh[k-i]:=1;
                cheoPhu[k+i]:=1;
                if k=n then GhiNghiem(x)
                else xepHau(k+1);
                cot[i]:=0;
            end;
end;

```

```

        cheoChinh[k-i]:=0;
        CheoPhu[k+i]:=0;
    end;
end;
BEGIN
    fillchar(cot,sizeof(cot),0);
    fillchar(cheoChinh,sizeof(cheoChinh),0);
    fillchar(cheoPhu,sizeof(cheoPhu),0);
    xepHau(1);
END.

```

Bài toán xếp hậu có tất cả 92 nghiệm, mươi nghiệm đầu tiên mà chương trình tìm được là:

1.	1	5	8	6	3	7	2	4
2.	1	6	8	3	7	4	2	5
3.	1	7	4	6	8	2	5	3
4.	1	7	5	8	2	4	6	3
5.	2	4	6	8	3	1	7	5
6.	2	5	7	1	3	8	6	4
7.	2	5	7	4	1	8	6	3
8.	2	6	1	7	4	8	3	5
9.	2	6	8	3	1	4	7	5
10.	2	7	3	6	8	5	1	4

### 1.2.5. Bài toán máy rút tiền tự động ATM

Một máy ATM hiện có  $n$  ( $n \leq 20$ ) tờ tiền có giá  $t_1, t_2, \dots, t_n$ . Hãy đưa ra một cách trả với số tiền đúng bằng  $S$ .

Dữ liệu vào từ file “ATM.INP” có dạng:

- Dòng đầu là 2 số  $n$  và  $S$
- Dòng thứ 2 gồm  $n$  số  $t_1, t_2, \dots, t_n$

Kết quả ra file “ATM.OUT” có dạng: Nếu có thể trả đúng  $S$  thì đưa ra cách trả, nếu không ghi -1.

ATM.INP	ATM.OUT
10 390 200 10 20 20 50 50 50 50 100 100	20 20 50 50 50 100 100

Nghiệm của bài toán là một dãy nhị phân độ dài  $n$ , trong đó thành phần thứ  $i$  bằng 1 nếu tờ tiền thứ  $i$  được sử dụng để trả, bằng 0 trong trường hợp ngược lại.

$X = (x_1, x_2, \dots, x_n)$  là nghiệm nếu:  $x_1 \times t_1 + x_2 \times t_2 + \dots + x_n \times t_n = S$

Trong chương trình dưới đây có sử dụng một biến ok để kiểm soát việc tìm nghiệm. Ban đầu chưa có nghiệm, do đó khởi trị ok=False. Khi tìm được nghiệm, ok sẽ được nhận giá trị bằng TRUE. Nếu ok=TRUE (đã tìm thấy nghiệm) ta sẽ không cần tìm kiếm nữa.

```
const      MAX          =20;
           fi           ='ATM.INP';
           fo           ='ATM.OUT';
type       vector        =array[1..MAX]of longint;
var        t             :array[1..MAX]of longint;
           x, xs         :vector;
           n, s, sum     :longint;
           ok            :boolean;
procedure  input;
var        f             :text;
           i             :longint;
begin
  assign(f,fi); reset(f);
  readln(f,n, s);
  for i:=1 to n do read(f,t[i]);
  close(f);
end;
procedure check(x:vector);
var i      :longint;
    f      :text;
begin
  if sum = s then begin
    xs:=x;
    ok:=true;
  end;
end;
procedure printResult;
var i      :longint;
    f      :text;
begin
  assign(f,fo); rewrite(f);
  if ok then begin
    for i:=1 to n do
```

```

        if xs[i]=1 then write(f,t[i], ' ');
    end
    else write(f, '-1');
    close(f);
end;
procedure backTrack(i:longint);
var j :longint;
begin
    for j:=0 to 1 do begin
        x[i]:=j;
        sum:=sum + x[i]*t[i];
        if (i=n) then check(x)
        else if sum<=s then backTrack(i+1);
        if ok then exit; {nếu đã tìm được nghiệm thì không duyệt nữa}
        sum:=sum - x[i]*t[i];
    end;
end;
BEGIN
    input;
    ok:=false;
    sum:=0;
    backTrack(1);
    PrintResult;
END.

```

## 2. Nhánh và cận

### 2.1. Phương pháp

Trong thực tế, có nhiều bài toán yêu cầu tìm ra một phương án thoả mãn một số điều kiện nào đó, và phương án đó là tốt nhất theo một tiêu chí cụ thể. Các bài toán như vậy được gọi là bài toán tối ưu. Có nhiều bài toán tối ưu không có thuật toán nào thực sự hữu hiệu để giải quyết, mà cho đến nay vẫn phải dựa trên mô hình xem xét toàn bộ các phương án, rồi đánh giá để chọn ra phương án tốt nhất.

Phương pháp nhánh và cận là một dạng cải tiến của phương pháp quay lui, được áp dụng để tìm nghiệm của bài toán tối ưu.

Giả sử nghiệm của bài toán có thể biểu diễn dưới dạng một vector  $(x_1, x_2, \dots, x_n)$ , mỗi thành phần  $x_i$  ( $i = 1, 2, \dots, n$ ) được chọn ra từ tập  $S_i$ . Mỗi nghiệm của bài toán

$X = (x_1, x_2, \dots, x_n)$ , được xác định “độ tốt” bằng một hàm  $f(X)$  và mục tiêu cần tìm nghiệm có giá trị  $f(X)$  đạt giá trị nhỏ nhất (hoặc đạt giá trị lớn nhất).

Tư tưởng của phương pháp nhánh và cận như sau: Giả sử, đã xây dựng được k thành phần  $(x_1, x_2, \dots, x_k)$  của nghiệm và khi mở rộng nghiệm  $(x_1, x_2, \dots, x_{k+1})$ , nếu biết rằng tất cả các nghiệm mở rộng của nó  $(x_1, x_2, \dots, x_{k+1}, \dots)$  đều không tốt bằng nghiệm tốt nhất đã biết ở thời điểm đó, thì ta không cần mở rộng từ  $(x_1, x_2, \dots, x_k)$  nữa. Như vậy, với phương pháp nhánh và cận, ta không phải duyệt toàn bộ các phương án để tìm ra nghiệm tốt nhất mà bằng cách đánh giá các nghiệm mở rộng, ta có thể cắt bỏ đi những phương án (nhánh) không cần thiết, do đó việc tìm nghiệm tối ưu sẽ nhanh hơn. Cái khó nhất trong việc áp dụng phương pháp nhánh và cận là đánh giá được các nghiệm mở rộng, nếu đánh giá được tốt sẽ giúp bỏ qua được nhiều phương án không cần thiết, khi đó thuật toán nhánh cận sẽ chạy nhanh hơn nhiều so với thuật toán vét cạn.

Thuật toán nhánh cận có thể mô tả bằng mô hình đệ quy sau:

```
procedure BranchBound(i); // xây dựng thành phần thứ i
begin
    <Đánh giá các nghiệm mở rộng>;
    if (các nghiệm mở rộng đều không tốt hơn
        BestSolution) then exit;
    <Xác định  $S_i$ >;
    for  $x_i \in S_i$  do begin
        <ghi nhận thành phần thứ i>;
        if (tìm thấy nghiệm) then <Cập nhật BestSolution>
        else BranchBound(i+1);
        <loại thành phần i>;
    end;
end;
```

Trong thủ tục trên, **BestSolution** là nghiệm tốt nhất đã biết ở thời điểm đó. Thủ tục <cập nhật **BestSolution**> sẽ xác định “độ tốt” của nghiệm mới tìm thấy, nếu nghiệm mới tìm thấy tốt hơn **BestSolution** thì **BestSolution** sẽ được cập nhật lại là nghiệm mới tìm được.

## 2.2. Giải bài toán người du lịch bằng phương pháp nhánh cận.

**Bài toán.** Cho  $n$  thành phố đánh số từ 1 đến  $n$  và các tuyến đường giao thông hai chiều giữa chúng, mạng lưới giao thông này được cho bởi mảng  $C[1..n, 1..n]$ , ở đây  $C_{ij} = C_{ji}$  là chi phí đi đoạn đường trực tiếp từ thành phố  $i$  đến thành phố  $j$ .

Một người du lịch xuất phát từ thành phố 1, muốn đi thăm tất cả các thành phố còn lại mỗi thành phố đúng 1 lần và cuối cùng quay lại thành phố 1. Hãy chỉ ra cho người đó hành trình với chi phí ít nhất. Bài toán được gọi là bài toán người du lịch hay bài toán người chào hàng (Travelling Salesman Problem - TSP)

Dữ liệu vào file “TSP.INP” có dạng:

- Dòng đầu chứa số  $n$  ( $1 < n \leq 20$ ), là số thành phố.
- $n$  dòng tiếp theo, mỗi dòng  $n$  số mô tả mảng  $C$

Kết quả ra file “TSP.OUT” có dạng:

- Dòng đầu là chi phí ít nhất
- Dòng thứ hai mô tả hành trình

**Ví dụ 1:**

TSP.INP	TSP.OUT	Hình minh họa
4 0 20 35 42 20 0 34 30 35 34 0 12 42 30 12 0	97 1->2->4->3->1	

**Ví dụ 2:**

TSP.INP	TSP.OUT	Hình minh họa
4 0 20 35 10 20 0 90 50 35 90 0 12 10 50 12 0	117 1->2->4->3->1	

## Giải

- 1) Hành trình cần tìm có dạng ( $x_1 = 1, x_2, \dots, x_n, x_{n+1} = 1$ ), ở đây giữa  $x_i$  và  $x_{i+1}$ : hai thành phố liên tiếp trong hành trình phải có đường đi trực tiếp; trừ thành phố 1, không thành phố nào được lặp lại hai lần, có nghĩa là dãy ( $x_1, x_2, \dots, x_n$ ) lập thành một hoán vị của  $(1, 2, \dots, n)$ .
- 2) Duyệt quay lui:  $x_2$  có thể chọn một trong các thành phố mà  $x_1$  có đường đi trực tiếp tới, với mỗi cách thử chọn  $x_2$  như vậy thì  $x_3$  có thể chọn một trong các thành phố mà  $x_2$  có đường đi tới (ngoài  $x_1$ ). Tổng quát:  $x_i$  có thể chọn 1 trong các thành phố chưa đi qua mà từ  $x_{i-1}$  có đường đi trực tiếp tới. ( $2 \leq i \leq n$ ).
- 3) Nhánh cận: Khởi tạo cấu hình BestSolution có chi phí =  $+\infty$ . Với mỗi bước thử chọn  $x_i$  xem chi phí đường đi cho tới lúc đó có nhỏ hơn chi phí của cấu hình BestSolution không? nếu không nhỏ hơn thì thử giá trị khác ngay bởi có đi tiếp cũng chỉ tốn thêm. Khi thử được một giá trị  $x_n$  ta kiểm tra xem  $x_n$  có đường đi trực tiếp về 1 không? Nếu có đánh giá chi phí đi từ thành phố 1 đến thành phố  $x_n$  cộng với chi phí từ  $x_n$  đi trực tiếp về 1, nếu nhỏ hơn chi phí của đường đi BestSolution thì cập nhật lại BestSolution bằng cách đi mới.

```
program TSP;
const      MAX          =20;
           oo           =1000000;
           fi           ='TSP.INP';
           fo           ='TSP.OUT';
var        c             :array[1..MAX,1..MAX]of longint;
           x,bestSolution :array[1..MAX]of longint;
           d             :array[1..MAX]of longint;
           n             :longint;
           sum,best     :longint;
procedure input;
var f       :text;
    i,j,k   :longint;
begin
    assign(f,fi); reset(f);
    read(f,n);
    for i:=1 to n do
        for j:=1 to n do read(f,C[i,j]);
    close(f);
```

```

end;

procedure update;
begin
  if sum+C[x[n],x[1]]<best then begin
    best:=sum+C[x[n],x[1]];
    bestSolution:=x;
  end;
end;

procedure branchBound(i:longint);
var j :longint;
begin
  if sum>=best then exit;
  for j:=1 to n do
    if d[j]=0 then begin
      x[i]:=j;
      d[j]:=1;
      sum:=sum + C[x[i-1],j];
      if i=n then update
        else branchBound(i+1);
      sum:=sum - C[x[i-1],j];
      d[j]:=0;
    end;
end;

procedure init;
begin
  fillchar(d,sizeof(d),0);
  d[1]:=1;
  x[1]:=1;
  best:=oo;
end;

procedure output;
var f :text;
  i :longint;
begin
  assign(f,fo); rewrite(f);
  writeln(f,best);
  for i:=1 to n do write(f,bestSolution[i],'->');
  write(f,bestSolution[1]);
  close(f);

```

```

end;
BEGIN
    input;
    init;
    branchBound(2);
    output;
END.

```

Chương trình trên là một giải pháp nhánh cận rất thô sơ giải bài toán TSP, có thể có nhiều cách đánh giá nhánh cận chặt hơn nữa làm tăng hiệu quả của chương trình.

## 2.3. Bài toán máy rút tiền tự động ATM

### Bài toán

Một máy ATM hiện có  $n$  ( $n \leq 20$ ) tờ tiền có giá  $t_1, t_2, \dots, t_n$ . Hãy tìm cách trả ít tờ nhất với số tiền đúng bằng  $S$ .

*Dữ liệu vào từ file “ATM.INP” có dạng:*

- Dòng đầu là 2 số  $n$  và  $S$
- Dòng thứ 2 gồm  $n$  số  $t_1, t_2, \dots, t_n$

*Kết quả ra file “ATM.OUT” có dạng:* Nếu có thể trả tiền đúng bằng  $S$  thì đưa ra số tờ ít nhất cần trả và đưa ra cách trả, nếu không ghi -1.

ATM.INP	ATM.OUT
10 390	5
200 10 20 20 50 50 50 50 100 100	20 20 50 100 200

### Giải

Như ta đã biết, nghiệm của bài toán là một dãy nhị phân độ dài  $n$ , giả sử đã xây dựng được  $k$  thành phần  $(x_1, x_2, \dots, x_k)$ , đã trả được  $sum$  và sử dụng  $c$  tờ. Để đánh giá được các nghiệm mở rộng của  $(x_1, x_2, \dots, x_k)$ , ta nhận thấy:

- Còn phải trả  $S - sum$
- Gọi  $tmax[k]$  là giá cao nhất trong các tờ tiền còn lại ( $tmax[k] = MAX\{t_{k+1}, \dots, t_n\}$ ) thì ít nhất cần sử dụng thêm  $\frac{S-sum}{tmax[k]}$  tờ nữa.

Do đó, nếu  $c + \frac{S-sum}{tmax[k]}$  mà lớn hơn hoặc bằng số tờ của cách trả tốt nhất hiện có thì không cần mở rộng các nghiệm của  $(x_1, x_2, \dots, x_k)$  nữa.

```

const      MAX      =20;
          fi       ='ATM.INP';
          fo       ='ATM.OUT';
type       vector   =array[1..MAX]of longint;
var        t,tmax  :array[1..MAX]of longint;
          x,xbest :vector;
          c,cbest :longint;
          n,s,sum :longint;
procedure  input;
var        f      :text;
          i      :longint;
begin
  assign(f,fi); reset(f);
  readln(f,n, s);
  for i:=1 to n do read(f,t[i]);
  close(f);
end;
procedure init;
var i :longint;
begin
  tmax[n]:=t[n];
  for i:=n-1 downto 1 do begin
    tmax[i]:=tmax[i+1];
    if tmax[i]<t[i] then tmax[i]:=t[i];
  end;
  sum:=0;
  c:=0;
  cbest:=n+1;
end;
procedure update;
var i   :longint;
    f   :text;
begin
  if (sum = s) and (c<cbest) then begin
    xbest:=x;
    cbest:=c;
  end;
end;
procedure printResult;

```

```

var i    :longint;
      f    :text;
begin
  assign(f,fo); rewrite(f);
  if cbest<n+1 then begin
    writeln(f,cbest);
    for i:=1 to n do
      if xbest[i]=1 then write(f,t[i],' ');
  end
  else write(f,'-1');
  close(f);
end;
procedure branchBound(i:longint);
var j :longint;
begin
  if c + (s-sum)/tmax[i] >= cbest then exit;
  for j:=0 to 1 do begin
    x[i]:=j;
    sum:=sum + x[i]*t[i];
    c:=c + j;
    if (i=n) then update
    else if sum<=s then branchBound(i+1);
    sum:=sum - x[i]*t[i];
    c:=c - j;
  end;
end;
BEGIN
  input;
  init;
  branchBound(1);
  PrintResult;
END.

```

### 3. Tham ăn (Greedy Method)

Phương pháp nhánh cận là cải tiến phương pháp quy lui, đã đánh giá được các nghiệm mở rộng để loại bỏ đi những phương án không cần thiết, giúp cho việc tìm nghiệm tối ưu nhanh hơn. Tuy nhiên, không phải lúc nào chúng ta cũng có thể đánh giá được nghiệm mở rộng, hoặc nếu có đánh giá được thì số phương án cần

xét vẫn rất lớn, không thể đáp ứng được trong thời gian cho phép. Khi đó, người ta chấp nhận tìm những nghiệm gần đúng so với nghiệm tối ưu. Phương pháp tham ăn được sử dụng trong các trường hợp như vậy. Ưu điểm nổi bật của phương pháp tham ăn là độ phức tạp nhỏ, thường nhanh chóng tìm được lời giải.

### 3.1. Phương pháp

Giả sử nghiệm của bài toán có thể biểu diễn dưới dạng một vector  $(x_1, x_2, \dots, x_n)$ , mỗi thành phần  $x_i$  ( $i = 1, 2, \dots, n$ ) được chọn ra từ tập  $S_i$ . Mỗi nghiệm của bài toán  $X = (x_1, x_2, \dots, x_n)$ , được xác định “**độ tốt**” bằng một hàm  $f(X)$  và mục tiêu cần tìm nghiệm có giá trị  $f(X)$  càng lớn càng tốt (hoặc càng nhỏ càng tốt).

Tư tưởng của phương pháp tham ăn như sau: Ta xây dựng vector nghiệm X dần từng bước, bắt đầu từ vector không ( ). Giả sử đã xây dựng được (k-1) thành phần  $(x_1, x_2, \dots, x_{k-1})$  của nghiệm và khi mở rộng nghiệm ta sẽ chọn  $x_k$  “**tốt nhất**” trong các ứng cử viên trong tập  $S_k$  để được  $(x_1, x_2, \dots, x_k)$ . Việc lựa chọn như thế được thực hiện bởi một hàm chọn. Cứ tiếp tục xây dựng, cho đến khi xây dựng xong hết thành phần của nghiệm.

Lược đồ tổng quát của phương pháp tham ăn.

```
procedure Greedy;
begin
  X:=∅;
  i:=0;
  while (chưa xây dựng xong hết thành phần của nghiệm) do
    begin
      i:=i+1;
      <Xác định  $S_i$ >;
       $x \leftarrow \text{select}(S_i)$ ; // chọn ứng cử viên tốt nhất trong tập  $S_i$ 
    end;
end;
```

Trong lược đồ tổng quát trên, Select là hàm chọn, để chọn ra từ tập các ứng cử viên  $S_i$  một ứng cử viên được xem là tốt nhất, nhiều hứa hẹn nhất.

Cần nhấn mạnh rằng, thuật toán tham ăn trong một số bài toán, nếu xây dựng được hàm thích hợp có thể cho nghiệm tối ưu. Trong nhiều bài toán, thuật toán tham ăn chỉ tìm được nghiệm gần đúng với nghiệm tối ưu.

### 3.2. Bài toán người du lịch

(Bài toán ở mục 2.2)

Có nhiều thuật toán tham ăn cho bài này, một thuật toán với ý tưởng đơn giản như sau: Xuất phát từ thành phố 1, tại mỗi bước ta sẽ chọn thành phố tiếp theo là thành phố chưa đến thăm mà chi phí từ thành phố hiện tại đến thành phố đó là nhỏ nhất, cụ thể:

- + Hành trình cần tìm có dạng ( $x_1 = 1, x_2, \dots, x_n, x_{n+1} = 1$ ), trong đó dãy ( $x_1, x_2, \dots, x_n$ ) lập thành một hoán vị của ( $1, 2, \dots, n$ ).
- + Ta xây dựng nghiệm từng bước, bắt đầu từ  $x_1=1$ , chọn  $x_2$  là thành phố gần  $x_1$  nhất, sau đó chọn  $x_3$  là thành phố gần  $x_2$  nhất ( $x_3$  khác  $x_1$ )... Tổng quát: chọn  $x_i$  là thành phố chưa đi qua mà gần  $x_{i-1}$  nhất. ( $2 \leq i \leq n$ ).

```

program TSP;
const      MAX          =100;
           oo           =1000000;
           fi           ='TSP.INP';
           fo           ='TSP.OUT';
var        C            :array[1..MAX,1..MAX] of
longint;
           x            :array[1..MAX] of longint;
           d            :array[1..MAX] of longint;
           n            :longint;
           sum          :longint;
procedure input;
var f       :text;
    i,j,k :longint;
begin
    assign(f,fi); reset(f);
    read(f,n);
    for i:=1 to n do
        for j:=1 to n do read(f,C[i,j]);
    close(f);
end;
procedure output;
var f     :text;
    i     :longint;
begin
    assign(f,fo); rewrite(f);
    writeln(f,sum);
    for i:=1 to n do write(f,x[i],'->');
    write(f,x[1]);

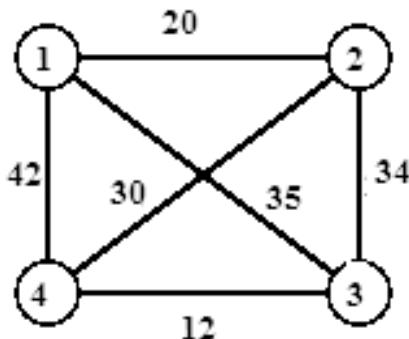
```

```

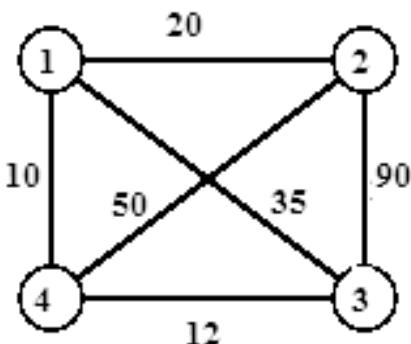
        close(f);
end;
procedure Greedy;
var i,j,xi :longint;
    best    :longint;
begin
    x[1]:=1;
    d[1]:=1;
    i:=1;
    while i<n do begin
        inc(i);
        //chọn 1 trong các viên tốt nhất
        best:=oo;
        for j:=1 to n do
            if (d[j]=0) and (c[x[i-1],j]<best) then begin
                best:=c[x[i-1],j];
                xi:=j;
            end;
        x[i]:=xi; //ghi nhận thành phần nghiệm thứ i
        d[xi]:=1;
        sum:=sum+c[x[i-1],x[i]];
    end;
    sum:=sum+c[x[n],x[1]];
end;
BEGIN
    input;
    Greedy;
    output;
END.

```

**Ví dụ 1.** Xuất phát từ thành phố 1, ta xây dựng được hành trình  $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$  với chi phí 97, đây là phương án tối ưu.



**Ví dụ 2.** Xuất phát từ thành phố 1, ta xây dựng được hành trình  $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$  với chi phí 132, nhưng kết quả tối ưu là 117.



### 3.3. Bài toán máy rút tiền tự động ATM

(bài toán ở mục 2.3)

Thuật toán với ý tưởng tham ăn đơn giản, hàm chọn như sau: Tại mỗi bước ta sẽ chọn tờ tiền lớn nhất còn lại không vượt quá lượng tiền còn phải trả, cụ thể:

- Sắp xếp các tờ tiền giảm dần theo giá trị.
- Lần lượt xét các tờ tiền từ giá trị lớn đến giá trị nhỏ, nếu vẫn còn chưa lấy đủ  $S$  và tờ tiền đang xét có giá trị nhỏ hơn hoặc bằng  $S$  thì lấy luôn tờ tiền đó.

```

const      MAX      =100;
          fi       ='ATM.INP';
          fo       ='ATM.OUT';
type       vector   =array[1..MAX]of longint;
var        t        :array[1..MAX]of longint;
          x        :vector;
          c        :longint;
          n,s     :longint;
procedure input;
var        f        :text;
          i        :longint;
begin
  assign(f,fi); reset(f);
  readln(f,n, s);
  for i:=1 to n do read(f,t[i]);
  close(f);
end;
procedure greedy;
var        i,j     :longint;

```

```

    tmp    :longint;
begin
    fillchar(x,sizeof(x),0);
    {sắp xếp các tờ theo giá trị giảm dần}
    for i:=1 to n-1 do
        for j:=i+1 to n do
            if t[i]<t[j] then begin
                tmp:=t[i];
                t[i]:=t[j];
                t[j]:=tmp;
            end;
    c:=0;
    for i:=1 to n do
        if s>=t[i] then
            begin
                inc(c); {số lượng tờ lấy}
                x[i]:=1; {tờ i được lấy}
                s:=s-t[i];
            end;
    end;
procedure printResult;
var i    :longint;
    f    :text;
begin
    assign(f,fo); rewrite(f);
    if s=0 then begin
        writeln(f,c);
        for i:=1 to n do
            if x[i]=1 then write(f,t[i],' ');
    end
    else write(f,'-1'); {nếu không lấy được đủ S, S>0}
    close(f);
end;
BEGIN
    input;
    greedy;
    PrintResult;
END.

```

## Các bộ test thử nghiệm

test	Dữ liệu vào	Kết quả tìm được
1	10 390 200 10 20 20 50 50 50 50 100 100	5 200 100 50 20 20
2	11 100 50 20 20 20 20 20 2 2 2 2 2	8 50 20 20 2 2 2 2 2
3	6 100 50 20 20 20 20 20	-1

Với bộ test (1), thuật toán tham ăn cũng cho được nghiệm tối ưu. Tuy nhiên, với bộ test (2), thuật toán tham ăn không cho nghiệm tối ưu và với bộ test (3), thuật toán tham ăn không tìm nghiệm mặc dù có nghiệm.

### 3.4. Bài toán lập lịch giảm thiểu trễ hạn

#### Bài toán:

Có  $n$  công việc đánh số từ 1 đến  $n$  và có một máy để thực hiện, biết:

- $p_i$  là thời gian cần thiết để hoàn thành công việc  $i$ .
- $d_i$  là thời hạn hoàn thành công việc  $i$ .

Máy bắt đầu hoạt động từ thời điểm 0. Mỗi công việc cần được thực hiện liên tục từ lúc bắt đầu cho tới khi kết thúc, không được phép ngắt quãng. Giả sử  $c_i$  là thời điểm hoàn thành công việc  $i$ . Khi đó, nếu  $c_i > d_i$  ta nói công việc  $i$  bị hoàn thành trễ hạn, còn nếu  $c_i \leq d_i$  thì ta nói công việc  $i$  được hoàn thành đúng hạn.

Yêu cầu: Tìm trình tự thực hiện các công việc sao cho số công việc hoàn thành trễ hạn là ít nhất (hay số công việc hoàn thành đúng hạn là nhiều nhất).

Dữ liệu vào trong file “JS.INP” có dạng:

- Dòng đầu là số  $n$  ( $n \leq 100$ ) là số công việc
- Dòng thứ hai gồm  $n$  số là thời gian thực hiện các công việc
- Dòng thứ ba gồm  $n$  số là thời hạn hoàn thành các công việc

Kết quả file “JS.OUT” có dạng: gồm một dòng là trình tự thực hiện các công việc.

Ví dụ: giả sử có 5 công việc với thời gian thực hiện và thời gian hoàn thành như sau:

$i$	1	2	3	4	5
$p_i$	6	3	5	7	2

$d_i$	8	4	15	20	3
-------	---	---	----	----	---

Nếu thực hiện theo thứ tự 1, 2, 3, 4, 5 thì sẽ có 3 công việc bị trễ hạn là công việc 2, 4 và 5. Còn nếu thực hiện theo thứ tự 5, 1, 3, 4, 2 thì chỉ có 1 công việc bị trễ hạn là công việc 2, đây là thứ tự thực hiện mà số công việc bị trễ hạn ít nhất (nghiệm tối ưu).

### Giải

Ta có hai nhận xét sau:

+ Nếu thứ tự thực hiện các công việc mà có công việc bị trễ hạn được xếp trước một công việc đúng hạn thì ta sẽ nhận được trình tự tốt hơn bằng cách chuyển công việc trễ hạn xuống cuối cùng (vì đâu nào công việc này cũng bị trễ hạn).

Ví dụ: thứ tự 1, 2, 3, 4, 5 có công việc 2 bị trễ hạn xếp trước công việc 3 đúng hạn, ta chuyển công việc 2 xuống cuối cùng để nhận được thứ tự: 1, 3, 4, 5, 2, thứ tự này chỉ có 2 công việc bị quá hạn là công việc 5 và 2.

Như vậy, ta chỉ quan tâm đến việc xếp lịch cho các công việc hoàn thành đúng hạn, còn các công việc bị trễ hạn có thể thực hiện theo trình tự bất kì.

+ Giả sử  $Js$  là tập gồm  $k$  công việc (mà cả  $k$  công việc này đều có thể thực hiện đúng hạn) và  $\sigma = (i_1, i_2, \dots, i_k)$  là một hoán vị của các công việc trong  $Js$  sao cho  $d_{i_1} \leq d_{i_2} \leq \dots \leq d_{i_k}$  thì thứ tự  $\sigma$  là thứ tự để hoàn thành đúng hạn được cả  $k$  công việc.

Ví dụ:  $Js$  gồm 4 công việc 1, 3, 4, 5 (4 công việc này đều có thể thực hiện đúng hạn), ta có thứ tự thực hiện  $\sigma = (5, 1, 3, 4)$  vì  $d_5 = 3 \leq d_1 = 8 \leq d_3 = 15 \leq d_4 = 20$  để cả 4 công việc đều thực hiện đúng hạn.

Sử dụng chiến lược tham ăn, ta xây dựng tập công việc  $Js$  theo từng bước, ban đầu  $Js = \emptyset$ . Hàm chọn được xây dựng như sau: tại mỗi bước ta sẽ chọn công việc  $Job_i$  mà có thời gian thực hiện nhỏ nhất trong số các công việc còn lại cho vào tập  $Js$ . Nếu sau khi kết nạp  $Job_i$ , các công việc trong tập  $Js$  đều có thể thực hiện đúng hạn thì cố định việc kết nạp  $Job_i$  vào tập  $Js$ , nếu không thì không kết nạp  $Job_i$ . Để đơn giản, ta giả sử rằng các công việc được đánh số theo thứ tự thời gian thực hiện tăng dần  $p_1 \leq p_2 \leq \dots \leq p_n$ . Ta có lược đồ thuật toán tham ăn như sau:

```

procedure JobScheduling;
begin
  Js := ∅;
  for i:=1 to n do
    
```

```

    if các công việc trong tập ( $Js \cup \{Job_i\}$ ) hoàn thành đúng
    hạn then
 $Js := Js \cup \{i\};$ 
    for i:=1 to n do
        if  $Job_i \notin Js$  then  $Js \cup \{Job_i\});$ 
end;

```

Sau đây là chương trình hoàn chỉnh:

```

const MAX = 100;
        fi = 'js.inp';
        fo = 'js.out';
type TJob = record
            p, d :longint;
            name :longint;
        end;
        TArrJobs = array[1..MAX] of TJob;
var jobs, Js :TArrJobs;
    d :array[1..MAX] of longint;
    n, m :longint;

procedure input;
var f :text;
    i :longint;
begin
    assign(f,fi);
    reset(f);
    readln(f,n);
    for i:=1 to n do read(f,jobs[i].p);
    for i:=1 to n do read(f,jobs[i].d);
    close(f);
    for i:=1 to n do jobs[i].name:=i;
end;
procedure swap(var j1,j2:TJob);
var tmp :TJob;
begin
    tmp:=j1;
    j1:=j2;
    j2:=tmp;
end;
function check(var Js:TArrJobs; nJob:longint):boolean;

```

```

var i,j :longint;
t :longint;
begin
  for i:=1 to nJob-1 do
    for j:=i+1 to nJob do
      if Js[i].d>Js[j].d then swap(Js[i],Js[j]);
t:=0;
for i:=1 to nJob do begin
  if t+Js[i].p>Js[i].d then exit(false);
  t:=t+Js[i].p;
end;
exit(true);
end;

procedure Greedy;
var i,j :longint;
  Js2 :TArrJobs;
begin
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if jobs[i].p > jobs[j].p then
        swap(jobs[i],jobs[j]);
  fillchar(d,sizeof(d),0);
m:=0;
for i:=1 to n do begin
  Js2:=Js;
  Js2[m+1]:=jobs[i];
  if check(Js2,m+1) then begin
    m:=m+1;
    Js:=Js2;
    d[i]:=1;
  end;
end;
//writeln(m);
for i:=1 to n do
  if d[i]=0 then begin
    m:=m+1;
    Js[m]:=jobs[i];
  end;
end;

```

```

procedure printResult;
var f :text;
    i :longint;
begin
    assign(f,fo); rewrite(f);
    for i:=1 to n do write(f,Js[i].name, ' ');
    close(f);
end;
BEGIN
    input;
    Greedy;
    printResult;
END.

```

Chú ý: Thuật toán tham ăn trình bày trên luôn cho phương án tối ưu.

## 4. Chia để trị (Divide and Conquer)

### 4.1. Phương pháp

Tư tưởng của chiến lược chia để trị như sau: Người ta phân bài toán cần giải thành các bài toán con. Các bài toán con lại được tiếp tục phân thành các bài toán con nhỏ hơn, cứ thế tiếp tục cho tới khi ta nhận được các bài toán con hoặc đã có thuật giải hoặc là có thể dễ dàng đưa ra thuật giải. Sau đó ta tìm cách kết hợp các nghiệm của các bài toán con để nhận được nghiệm của bài toán con lớn hơn, để cuối cùng nhận được nghiệm của bài toán cần giải. Thông thường các bài toán con nhận được trong quá trình phân chia là cùng dạng với bài toán ban đầu, chỉ có cỡ của chúng là nhỏ hơn.

Thuật toán chia để trị có thể biểu diễn bằng mô hình đệ quy như sau:

```

procedure DivideConquer (A, x) ; // tìm nghiệm x của bài toán A
begin
if (A đủ nhỏ) then Solve(A)
else begin
    Phân A thành các bài toán con A1, A2, ..., Am;
    for i:=1 to m do DivideConquer (Ai, xi);
    Kết hợp các nghiệm xi (i=1,2,...,m) của các bài toán
    con Ai để nhận được nghiệm của bài toán A;
end;
end;

```

Trong thủ tục trên, Solve(A) là thuật giải bài toán A trong trường hợp A có cỡ đủ nhỏ.

Trong thuật toán tìm kiếm nhị phân và thuật toán sắp xếp nhanh-QuickSort (ở chuyên đề sắp xếp) là hai thuật toán được thiết kế dựa trên chiến lược chia để trị. Sau đây, chúng ta sẽ tìm hiểu một số ví dụ minh họa cho phương pháp chia để trị.

## 4.2. Bài toán tính $a^n$

**Bài toán:** Cho số  $a$  và số nguyên dương  $n$ , tính  $a^n$

*Cách 1:* Sử dụng thuật toán lặp, mất  $n$  phép nhân để tính  $a^n$

```
procedure power(a,n:longint; var p:longint);  
{giá trị  $a^n$  sẽ được lưu vào biến p}  
var i : longint;  
begin  
    p:=1;  
    for i:=1 to n do p:=p*a;  
end;  
var a, n, p : longint;  
BEGIN  
    write('Nhập a, n:'); readln(a, n);  
    power(a,n,p);  
    write(p);  
END.
```

*Cách 2:* Áp dụng kĩ thuật chia để trị, ta tính  $a^n$  dựa vào  $a^k$  (trong đó  $k = n \text{ div } 2$ ) như sau:

- nếu  $n$  chẵn:  $a^n = a^k \times a^k$
- nếu  $n$  lẻ:  $a^n = a^k \times a^k \times a$

Để tính  $a^k$  ta lại dựa vào  $a^k \text{ div } 2$ , quá trình chia nhỏ cho đến khi nhận được bài toán tính  $a^1$  thì dừng.

Ví dụ: tính  $9^{13}$

- bài toán được tính dựa trên bài toán con  $9^6$ , ta có  $9^{13} = 9^6 \times 9^7$
- bài toán  $9^6$  được tính dựa trên bài toán con  $9^3$ , ta có  $9^6 = 9^3 \times 9^3$
- bài toán  $9^3$  được tính dựa trên bài toán con  $9^1$ , ta có  $9^{13} = 9^1 \times 9^1 \times 9^1$

Thủ tục để quy power(a, n, p) sau thể hiện ý tưởng trên.

```
procedure power(a,n:longint; var p:longint);  
var tmp : longint;
```

```

begin
if (n=1) then p:=a
else begin
    power(a,n div 2,tmp);
    if (n mod 2=1) then p:=tmp*tmp*a
    else p:=tmp*tmp;
end;
end;

```

hoặc viết dưới dạng hàm như sau:

```

function power(a,n:longint):longint;
var tmp : longint;
begin
    if (n=1) then exit(a)
else begin
    tmp:=power(a,n div 2);
    if (n mod 2=1) then exit(tmp*tmp*a)
    else exit(tmp*tmp);
end;
end;

```

Để đánh giá thời gian thực hiện thuật toán, ta tính số phép nhân phải sử dụng, gọi  $T(n)$  là số phép nhân thực hiện, ta có:

$$\begin{cases} T(1) = 0 \\ T(n) \leq T\left(\frac{n}{2}\right) + 1 + 1 \text{ nếu } n > 1 \end{cases}$$

$$T(n) \leq T\left(\frac{n}{2}\right) + 2 \leq T\left(\frac{n}{2^2}\right) + 2 + 2 \leq \dots \leq 2\log n$$

Như vậy, thuật toán chia để trị mất không quá  $2\log n$  phép nhân, nhỏ hơn rất nhiều so với  $n$  phép nhân.

### 4.3. Bài toán *Diff*

**Bài toán:** Cho mảng số nguyên  $A[1..n]$ , cần tìm  $\text{Diff}(A[1..n]) = A[j] - A[i]$  đạt giá trị lớn nhất mà  $1 \leq i \leq j \leq n$ .

Ví dụ: mảng gồm 6 số 4, 2, 5, 8, 1, 7 thì độ lệch cần tìm là: 6

*Cách 1:* Thử tất cả các cặp chỉ số  $(i, j)$ , độ phức tạp  $O(N^2)$

```

procedure find(var maxDiff:longint);
var i,j :longint;

```

```

begin
    maxDiff:=0;
    for i:=1 to n do
        for j:=i to n do
            if a[j]-a[i]>maxDiff then maxDiff:=a[j]-a[i];
end;

```

Cách 2: Áp dụng kĩ thuật chia để trị, ta chia mảng  $A[1..n]$  thành hai mảng con  $A[1..k]$  và  $A[(k + 1)..n]$  trong đó  $k = n \text{ div } 2$ , ta có:

$$Diff(A[1..n]) = \begin{cases} Diff(A[1..k]) \\ Diff(A[(k + 1)..n]) \\ MAX(A[(k + 1)..n]) - MIN(A[1..k]) \end{cases}$$

Nếu tìm được độ lệch ( $Diff$ ), giá trị lớn nhất ( $MAX$ ) và giá trị nhỏ nhất ( $MIN$ ) của hai mảng con  $A[1..k]$  và  $A[(k + 1)..n]$ , ta sẽ dễ dàng xác định được giá trị  $Diff(A[1..n])$ . Để tìm độ lệch, giá trị lớn nhất và giá trị nhỏ nhất của hai mảng con  $A[1..k]$  và  $A[(k + 1)..n]$ , ta lại tiếp tục chia đôi chúng. Quá trình phân nhỏ bài toán dừng lại khi ta nhận được bài toán mảng con chỉ có 1 phần tử. Từ phương pháp đã trình bày ở trên, ta xây dựng thủ tục đệ quy

```
find2(l, r, maxDiff, maxValue, minValue)
```

tìm giá trị độ lệch, giá trị lớn nhất, giá trị nhỏ nhất trên mảng  $A[l..r]$  với  $1 \leq l \leq r \leq n$ .

```

const      MAXN      =100000;
           fi         ='';
           fo         ='';
var       a          :array[1..MAXN] of longint;
           n          :longint;
           maxdiff   :longint;
           tmp1, tmp2 :longint;
procedure
           find2(l,r:longint;var
maxDiff,maxValue,minValue :longint);
var       mid         :longint;
           maxD1, maxV1, minV1 :longint;
           maxD2, maxV2, minV2 :longint;
begin
    if l=r then begin
        maxDiff:=0;
        maxValue:=a[r];

```

```

    minValue:=a[r];
end
else begin
    mid:=(l+r) div 2;
    find2(l, mid, maxD1, maxV1,minV1);
    find2(mid+1, r, maxD2, maxV2, minV2);
    maxDiff:=maxV2 - minV1;
    if maxDiff < maxD1 then maxDiff := maxD1;
    if maxDiff < maxD2 then maxDiff := maxD2;
    if maxV1 > maxV2 then
        maxValue:=maxV1 else maxValue:=maxV2;
    if minV1 < minV2 then
        minValue:=minV1 else minValue:=minV2;
    end;
end;
procedure input;
var f    :text;
    i    :longint;
begin
    assign(f,fi); reset(f);
    readln(f,n);
    for i:=1 to n do read(f,a[i]);
    close(f);
end;
BEGIN
    input;
    find2(1,n,maxDiff,tmp1,tmp2);
    writeln(maxDiff);
END.

```

Gọi  $T(n)$  là số phép toán cần thực hiện trên mảng  $n$  phần tử  $A[1..n]$ , ta có:

$$T(n) = \begin{cases} 0 & \text{nếu } n = 1 \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \alpha & \text{nếu } n > 1 \end{cases}$$

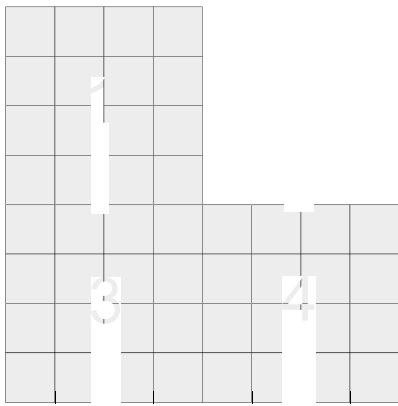
Giả sử,  $n = 2^k$ , bằng phương pháp thê ta có:

$$\begin{aligned} T(n) &= T(2^k) = 2T(2^{k-1}) + \alpha = 2(2T(2^{k-2}) + \alpha) + \alpha \\ &= 2^2T(2^{k-2}) + 2\alpha + \alpha = \dots = 2^3T(2^{k-3}) + 2^2 + 2 + 1 \\ &= 2^kT(1) + 2^{k-1}\alpha + \dots + 2\alpha + \alpha = (2^k - 1)\alpha = (n - 1)\alpha \end{aligned}$$

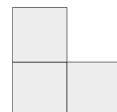
Độ phức tạp thuật toán là:  $O(N)$

#### 4.4. Lát nền

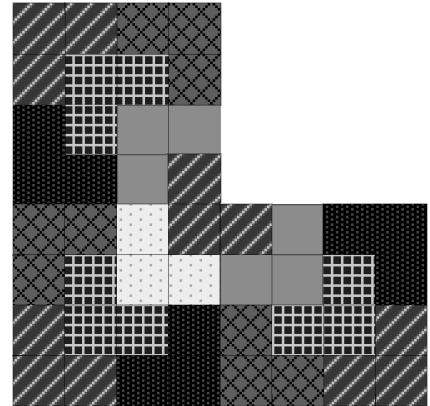
Hãy lát nền nhà hình vuông cạnh  $n = 2^k$  ( $2 \leq k \leq 10$ ) bị khuyết một phần tư tại góc trên phải (khuyết phần 2) bằng những viên gạch hình thóp tạo bởi 3 ô vuông đơn vị.



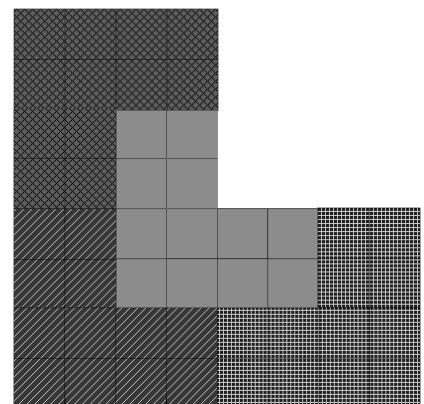
Nền nhà ( $k = 3$ )



Gạch hình thóp



Một cách lát nền



**Giải.** Ta chia nền nhà thành 4 phần (hình bên phải), mỗi phần có hình dạng giống với hình ban đầu nhưng có cạnh giảm đi một nửa. Như vậy, nếu có thể lát nền với kích thước  $2^k$  thì ta hoàn toàn có thể lát nền với kích thước  $2^{k+1}$ .

Thủ tục để quy cover ( $x, y, s, t$ ) dưới đây sẽ lát nền có kích thước  $s$ , bị khuyết phần  $t$  ( $t = 1,2,3,4$ ) có tọa độ trái trên là  $(x, y)$ .

```

const MAXSIZE = 1 shl 10;
var a :array[1..MAXSIZE,1..MAXSIZE]of longint;
    count,k :longint;
procedure cover(x,y,s,t:longint);
begin
  if s = 2 then begin
    inc(count);
    if t<>1 then a[x,y]:=count;
    if t<>2 then a[x,y+1]:=count;
    if t<>3 then a[x+1,y]:=count;
    if t<>4 then a[x+1,y+1]:=count;
    exit;
  end;
end;
  
```

```

end;

if t=1 then begin
  cover(x,y+s div 2,s div 2,3);
  cover(x+s div 2,y,s div 2,2);
  cover(x+s div 2,y+s div 2,s div 2,1);
  cover(x+s div 4,y+s div 4,s div 2,1);
end;
if t=2 then begin
  cover(x,y,s div 2,4);
  cover(x+s div 2,y,s div 2,2);
  cover(x+s div 2,y+s div 2,s div 2,1);
  cover(x+s div 4,y+s div 4,s div 2,2);
end;
if t=3 then begin
  cover(x,y,s div 2,4);
  cover(x,y+s div 2,s div 2,3);
  cover(x+s div 2,y+s div 2,s div 2,1);
  cover(x+s div 4,y+s div 4,s div 2,3);
end;
if t=4 then begin
  cover(x,y,s div 2,4);
  cover(x+s div 2,y,s div 2,2);
  cover(x,y+s div 2,s div 2,3);
  cover(x+s div 4,y+s div 4,s div 2,4);
end;
end;

procedure output;
var i,j :longint;
    f    :text;
begin
  assign(f,'cover.out'); rewrite(f);
  for i:=1 to 1 shl k do
    begin
      for j:=1 to 1 shl k do write(f,a[i,j],' ');
      writeln(f);
    end;

```

```

    close(f);
end;
BEGIN
    write('k='); readln(k);
    cover(1,1,1 shl k,2);
    output;
END.

```

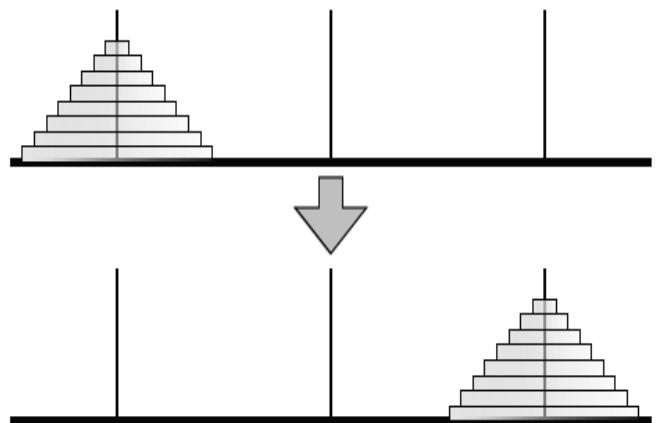
Ví dụ về Input / Output của chương trình:

$k = 3$	1 1 3 3 0 0 0 0 1 4 4 3 0 0 0 0 2 4 13 13 0 0 0 0 2 2 13 16 0 0 0 0 5 5 14 16 16 15 9 9 5 8 14 14 15 15 12 9 6 8 8 7 10 12 12 11 6 6 7 7 10 10 11 11
---------	---

## 4.5. Tháp Hà Nội

Cho 3 cái cọc và  $n$  đĩa có kích thước khác nhau. Ban đầu cả  $n$  đĩa đều ở cọc 1 và được xếp theo thứ tự đĩa to ở dưới, đĩa nhỏ ở trên. Hãy di chuyển cả  $n$  đĩa từ cọc 1 sang cọc 3 theo quy tắc sau:

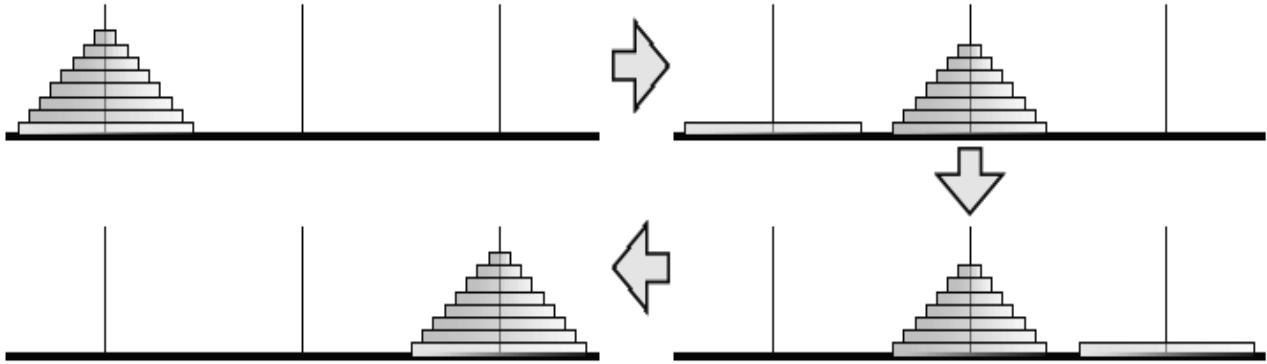
- Một lần chỉ được chuyển một đĩa
- Trong quá trình chuyển đĩa, có thể sử dụng cọc 2 làm cọc trung gian và một đĩa chỉ được đặt lên một đĩa lớn hơn.



### Giải

Để chuyển  $n$  đĩa từ cọc 1 sang cọc 3 ta sẽ thực hiện như sau:

- chuyển  $(n - 1)$  đĩa từ cọc 1 sang cọc 2, sử dụng cọc 3 làm cọc trung gian
- chuyển 1 đĩa từ cọc 1 sang cọc 3
- chuyển  $(n - 1)$  đĩa từ cọc 2 sang cọc 3, sử dụng cọc 1 làm cọc trung gian



```

procedure move(n : longint; src, dst, tmp: longint);
begin
  if n = 1 then writeln('move ', src, ' ', dst)
  else begin
    move(n-1, src, tmp, dst);
    move(1, src, dst, tmp);
    move(n-1, tmp, dst, src);
  end;
end;

```

#### 4.6. Bài toán sắp xếp mảng bằng thuật toán trộn (Merge Sort)

Bài toán: Cho mảng số nguyên  $A[1..n]$ , cần sắp xếp các phần tử của mảng theo thứ tự tăng dần.

Giải: Ta chia mảng  $A[1..n]$  thành hai mảng con  $A[1..k]$  và  $A[(k+1)..n]$  trong đó  $k = n \text{ div } 2$ . Giả sử, hai mảng con  $A[1..k]$  và  $A[(k+1)..n]$  đã được sắp xếp tăng dần, ta sẽ trộn hai mảng con để được mảng  $A[1..n]$  cũng sắp xếp tăng dần. Để sắp xếp hai mảng con  $A[1..k]$  và  $A[(k+1)..n]$  ta lại tiếp tục chia đôi chúng. Thủ tục đệ quy MergeSort( $i, j$ ) sắp xếp tăng dần mảng con  $A[i..j]$  với  $1 \leq i \leq j \leq n$ . Để sắp xếp cả mảng  $A[1..n]$ , ta chỉ cần gọi thủ tục này với  $i = 1, j = n$ .

```

procedure MergeSort(i, j: longint);
var k : longint;
begin
  if (i < j) then begin
    k := (i+j) div 2;
    MergeSort(i, k);
    MergeSort(k+1, j);
    Merge(i, k, j);
    {thủ tục Merge(i, k, j) trộn hai mảng con A[i..k], A[(k+1)..j] đã được
    sắp xếp thành mảng A[i..j] cũng được sắp xếp}
  end;
end;

```

Việc trộn hai mảng con đã được sắp xếp  $A[i..k]$  và  $A[(k+1)..j]$  thành mảng  $A[i..j]$  cũng được sắp xếp có thể thực hiện trong thời gian  $O(j-i+1)$ , là bài tập 3.8 ở chuyên đề Sắp xếp. *Thuật toán MergeSort có độ phức tạp là  $O(n \log n)$ .*

## 5. Quy hoạch động (Dynamic programming)

### 5.1. Phương pháp

Trong chiến lược chia để trị, người ta phân bài toán cần giải thành các bài toán con. Các bài toán con lại được tiếp tục phân thành các bài toán con nhỏ hơn, cứ thế tiếp tục cho tới khi ta nhận được các bài toán con có thể giải được dễ dàng. Tuy nhiên, trong quá trình phân chia như vậy, có thể ta sẽ gặp rất nhiều lần cùng một bài toán con. Tư tưởng cơ bản của phương pháp quy hoạch động là sử dụng một bảng để lưu giữ lời giải của các bài toán con đã được giải. Khi giải một bài toán con cần đến nghiệm của bài toán con cỡ nhỏ hơn, ta chỉ cần lấy lời giải ở trong bảng mà không cần phải giải lại. Chính vì thế mà các thuật toán được thiết kế bằng quy hoạch động sẽ rất hiệu quả.

Để giải quyết một bài toán bằng phương pháp quy hoạch động, chúng ta cần tiến hành những công việc sau:

- Tìm nghiệm của các bài toán con nhỏ nhất.
- Tìm ra công thức (hoặc quy tắc) xây dựng nghiệm của bài toán con thông qua nghiệm của các bài toán con cỡ nhỏ hơn.
- Tạo ra một bảng lưu giữ các nghiệm của các bài toán con. Sau đó tính nghiệm của các bài toán con theo công thức đã tìm ra và lưu vào bảng.
- Từ các bài toán con đã giải để tìm nghiệm của bài toán.

Sau đây, chúng ta sẽ tìm hiểu một số ví dụ minh họa cho phương pháp quy hoạch động.

### 5.2. Số Fibonacci

Số Fibonacci được xác định bởi công thức:

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \text{ với } n \geq 2 \end{cases}$$

Hãy xác định số Fibonacci thứ  $n$ .

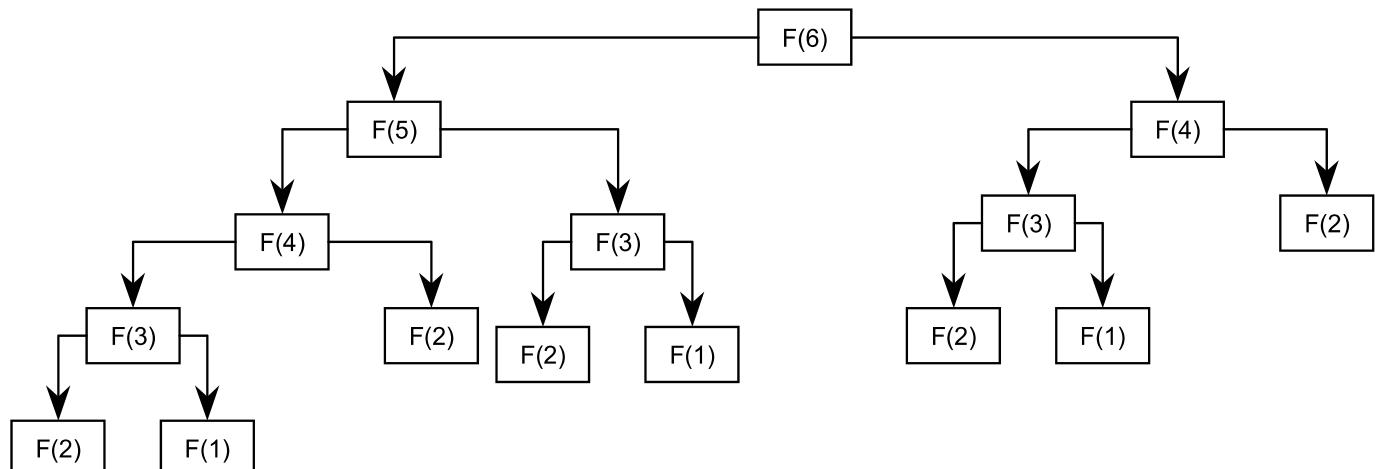
Cách 1: Áp dụng phương pháp chia để trị, ta tính  $F_n$  dựa vào  $F_{n-1}$  và  $F_{n-2}$ .

```

function F(n: longint): int64;
begin
  if n <=1 then F := n
  else F := F(i - 1) + F(i - 2);
end;
BEGIN
  readln(n);
  Writeln(F(n));
END.

```

Hàm đệ quy  $F(n)$  để tính số Fibonacci thứ  $n$ . Ví dụ  $n = 6$ , chương trình chính gọi  $F(6)$ , nó sẽ gọi tiếp  $F(5)$  và  $F(4)$  để tính ... Quá trình tính toán có thể vẽ như cây dưới đây. Ta nhận thấy để tính  $F(6)$  nó phải tính 1 lần  $F(5)$ , hai lần  $F(4)$ , ba lần  $F(3)$ , năm lần  $F(2)$ , ba lần  $F(1)$ .



*Cách 2: Phương pháp quy hoạch động.*

Ta sử dụng mảng  $S[0..MaxN]$ ,  $S[i]$  để lưu lại lời giải cho bài toán tính số Fibonacci thứ  $i$ .

```

const MaxN = 50;
var S : array[0..MaxN] of int64;
n, k : longint;

function F(n:longint):int64;
begin
  if S[n]==-1 then
  begin
    {bài toán chưa được giải thì sẽ tiến hành giải}
    if n<=1 then S[n]:=n
    else S[n]:=F(n-1) + F(n-2);
  end;
  {nếu bài toán đã được giải thì không cần giải nữa mà lấy luôn kết quả}

```

```

F:=S[n];
end;
BEGIN
  readln(n);
  for k:=0 to MaxN do S[k]:=-1;
  writeln(F(n));
END.

```

Ta nhận thấy, mỗi bài toán con chỉ được giải đúng một lần. Hãy cài đặt cả 2 chương trình trên và thử chạy với  $n = 40$  để thấy được sự khác biệt!

Ta cũng có thể cài đặt phương pháp quy hoạch động cho bài toán như sau:

```

const      maxN      =50;
var        S          : array[0..maxN] of Int64;
           i          : longint;
BEGIN
  readln(n);
  S[0] := 1; S[1] := 1;
  for i := 2 to n do
    S[i] := S[i - 1] + S[i - 2];
  Writeln(S[n]);
END.

```

Trước hết nó tính sẵn  $S[0]$  và  $S[1]$ , từ đó tính tiếp  $S[2]$ , lại tính tiếp được  $S[3]$ ,  $S[4], \dots, S[n]$ . Đảm bảo rằng mỗi giá trị Fibonacci chỉ phải tính 1 lần.

### 5.3. Dãy con đơn điệu tăng dài nhất

Cho dãy số nguyên  $A = a_1, a_2, \dots, a_n$ . ( $n \leq 1000, -10000 \leq a_i \leq 10000$ ). Một dãy con của  $A$  là một cách chọn ra trong  $A$  một số phần tử giữ nguyên thứ tự. Như vậy  $A$  có  $2^n$  dãy con.

Yêu cầu: Tìm dãy con đơn điệu tăng của  $A$  có độ dài lớn nhất.

Ví dụ:  $A = (1, 2, 3, 4, 9, 10, 5, 6, 7, 8)$ . Dãy con đơn điệu tăng dài nhất là:  $(1, 2, 3, 4, 5, 6, 7, 8)$ .

#### *Giải*

Bổ sung vào  $A$  hai phần tử:  $a_0 = -\infty$  và  $a_{n+1} = +\infty$ . Khi đó dãy con đơn điệu tăng dài nhất chắc chắn sẽ bắt đầu từ  $a_0$  và kết thúc ở  $a_{n+1}$ .

Với  $\forall i: 0 \leq i \leq n + 1$ . Ta sẽ tính  $L[i] = \text{độ dài dãy con đơn điệu tăng dài nhất bắt đầu tại } a_i$ .

## 1. Bài toán nhỏ nhất

$L[n + 1] = \text{Độ dài} dãy con đơn điệu tăng dài nhất bắt đầu tại } a_{n+1} = +\infty.$  Dãy con này chỉ gồm mỗi một phần tử  $(+\infty)$  nên  $L[n + 1] = 1.$

## 2. Công thức

Giả sử với  $i$  từ  $n$  đến  $0$ , ta cần tính  $L[i]$ : độ dài dãy con tăng dài nhất bắt đầu tại  $a_i$ .  $L[i]$  được tính trong điều kiện  $L[i + 1], L[i + 2], \dots, L[n + 1]$  đã biết:

Dãy con đơn điệu tăng dài nhất bắt đầu từ  $a_i$  sẽ được thành lập bằng cách lấy  $a_i$  ghép vào đầu một trong số những dãy con đơn điệu tăng dài nhất bắt đầu tại vị trí  $a_j$  đứng sau  $a_i$ .

Ta sẽ chọn dãy nào để ghép  $a_i$  vào đầu? Tất nhiên là chỉ được ghép  $a_i$  vào đầu những dãy con bắt đầu tại  $a_j$  nào đó lớn hơn  $a_i$  (để đảm bảo tính tăng) và dĩ nhiên ta sẽ chọn dãy dài nhất để ghép  $a_i$  vào đầu (để đảm bảo tính dài nhất). Vậy  $L[i]$  được tính như sau:

Xét tất cả các chỉ số  $j$  trong khoảng từ  $i + 1$  đến  $n + 1$  mà  $a_j > a_i$ , chọn ra chỉ số  $j_{\max}$  có  $L[j_{\max}]$  lớn nhất. Đặt  $L[i] := L[j_{\max}] + 1.$

## 3. Truy vết

Tại bước xây dựng dãy  $L$ , mỗi khi tính  $L[i] := L[j_{\max}] + 1$ , ta đặt  $T[i] = j_{\max}$ .

Để lưu lại rằng: Dãy con dài nhất bắt đầu tại  $a_i$  sẽ có phần tử thứ hai kế tiếp là  $a_{j_{\max}}$ . Sau khi tính xong hay dãy  $L$  và  $T$ , ta bắt đầu từ  $0$ .  $T[0]$  là phần tử đầu tiên được chọn,

$T[T[0]]$  là phần tử thứ hai được chọn,

$T[T[T[0]]]$  là phần tử thứ ba được chọn ...

Quá trình truy vết có thể diễn tả như sau:

```
i := T[0];
while i <> n + 1 do
    {Chừng nào chưa duyệt đến số  $a_{n+1}=+\infty$  ở cuối}
    begin
        <Thông báo chọn  $a_i$ >
        i := T[i];
    end;
```

Ví dụ: với  $A = (5, 2, 3, 4, 9, 10, 5, 6, 7, 8).$

Hai dãy Length và Trace sau khi tính sẽ là:

$i$	0	1	2	3	4	5	6	7	8	9	10	11
$a_i$	$-\infty$	5	2	3	4	9	10	5	6	7	8	$+\infty$
Length[i]	9	5	8	7	6	3	2	5	4	3	2	1
Trace[i]	2	8	3	4	7	6	11	8	9	10	11	

Truy vết → → → → → → → → → → →

```

Const      max = 1000;
var
    a, L, T: array[0..max + 1] of longint;
    n: longint;
procedure Enter;           {Nhập dữ liệu}
var
    i: longint;
begin
    Write('n = '); Readln(n);
    for i := 1 to n do
        begin
            Write('a[ ', i, ' ] = '); Readln(a[i]);
        end;
    end;
procedure Optimize;     {Quy hoạch động}
var
    i, j, jmax: longint;
begin
    a[0] := -32768; a[n + 1] := 32767;   {Thêm hai phần tử cạnh hai
đầu dãy a}
    L[n + 1] := 1;           {Điền cơ sở quy hoạch động vào bảng phương án}
    for i := n downto 0 do
        begin
{Chọn trong các chỉ số j đứng sau i thoả mãn aj > ai ra chỉ số jmax có L[jmax] lớn nhất}
            jmax := n + 1;
            for j := i + 1 to n + 1 do
                if (a[j] > a[i]) and (L[j] > L[jmax]) then jmax
                := j;
            L[i] := L[jmax] + 1; {Lưu độ dài dãy con tăng dài nhất bắt đầu tại ai}
            T[i] := jmax;         {Lưu vết: phần tử đứng liền sau a_i trong dãy con tăng
dài nhất đó là a_{jmax}}
        end;
end;

```

```

Writeln('Length of result : ', L[0] - 2); {Chiều dài dãy con
tăng dài nhất}
i := T[0]; {Bắt đầu truy vết tìm nghiệm}
while i <> n + 1 do
begin
  Writeln('a[', i, '] = ', a[i]);
  i := T[i];
end;
end;
begin
  Enter;
  Optimize;
end.

```

## 5.4. Dãy con chung dài nhất

Cho hai số nguyên dương  $M, N$  ( $0 < M, N \leq 100$ ) và hai dãy số nguyên:  $A_1, A_2, \dots, A_M$  và  $B_1, B_2, \dots, B_N$ . Tìm một dãy dài nhất  $C$  là dãy con chung dài nhất của hai dãy  $A$  và  $B$ , nhận được từ  $A$  bằng cách xoá đi một số số hạng và cũng nhận được từ  $B$  bằng cách xoá đi một số số hạng.

*Dữ liệu vào trong file LCS.INP có dạng:*

- + Dòng thứ nhất chứa  $M$  số  $A_1, A_2, \dots, A_M$
- + Dòng thứ hai chứa  $N$  số  $B_1, B_2, \dots, B_N$ .

*Dữ liệu ra trong file LCS.OUT có dạng:*

- + Dòng thứ nhất ghi số  $k$  là số số hạng của dãy  $C$ .
- + Dòng thứ hai chứa  $k$  số là các số hạng của dãy  $C$ .

*Giải*

Cần xây dựng mảng  $L[0..M, 0..N]$  với ý nghĩa:  $L[i, j]$  là độ dài của dãy chung dài nhất của hai dãy  $A[0..i]$  và  $B[0..j]$ .

Đương nhiên nếu một dãy là rỗng (số phần tử là 0) thì dãy con chung cũng là rỗng vì vậy  $L[0, j] = 0 \forall j, j = 1..N$ ,  $L[i, 0] = 0 \forall i, i = 1..M$ . Với  $M \geq i > 0$  và  $N \geq j > 0$  thì  $L[i, j]$  được tính theo công thức truy hồi sau:

$$L[i, j] = \text{Max} \{L[i, j-1], L[i-1, j], L[i-1, j-1] + x\}$$

(với  $x = 0$  nếu  $A[i] \neq B[j]$ ,  $x=1$  nếu  $A[i]=B[j]$ )

```

const fi      = 'LCS.INP';
        fo      = 'LCS.OUT';
        MaxMN = 100;
var   f       : text;
      a,b    : array[0..MaxMN] of longint;
      l      : array[0..MaxMN,0..MaxMN] of longint;
      m,n    : longint;
      p      : array[0..MaxMN] of longint;
      count : longint;
procedure Enter;
var f : text;
begin
  m := 0;
  n := 0;
  assign(f,fi);
  reset(f);
  while not eoln(f) do
  begin
    inc(m);
    read(f,a[m]);
  end;
  writeln(f);
  while not eoln(f) do
  begin
    inc(n);
    read(f,b[n]);
  end;
  close(f);
end;
function max(x,y : longint) : longint;
begin
  if x>y then max := y
  else max := y;
end;
procedure Optimize;
var i,j : longint;
begin
  for i:=1 to m do l[i,0] := 0;
  for j:=1 to n do l[0,j] := 0;
  for i:=1 to m do
    for j:=1 to n do
      begin

```

```

        if a[i]=b[j] then l[i,j] := l[i-1,j-1] + 1
        else l[i,j] := max(l[i,j-1],l[i-1,j]);
    end;
end;
procedure Trace;
var      f      : text;
          i,j : longint;
begin
    assign(f,fo);
    rewrite(f);
    writeln(f,l[m,n]);
    i := m;
    j := n;
    fillchar(p,sizeof(p),0);
    count:= 0;
    while (i>0) and (j>0) do
begin
    if a[i]=b[j] then
    begin
        inc(count);
        p[count] := a[i];
        dec(i);
        dec(j);
    end
    else if l[i,j]=l[i,j-1] then dec(j)
    else dec(i);
end;
    for i:=count downto 1 do write(f,p[i],' ');
    close(f);
end;
BEGIN
    Enter;
    Optimize;
    Trace;
END.

```

## 5.5. Bài toán cái túi

Trong siêu thị có  $n$  gói hàng ( $n \leq 100$ ), gói hàng thứ  $i$  có trọng lượng là  $W_i \leq 100$  và trị giá  $V_i \leq 100$ . Một tên trộm đột nhập vào siêu thị, sức của tên trộm không thể mang được trọng lượng vượt quá  $M$  ( $M \leq 100$ ). Hỏi tên trộm sẽ lấy đi những gói hàng nào để được tổng giá trị lớn nhất.

## **Giải**

Nếu gọi  $B[i, j]$  là giá trị lớn nhất có thể có bằng cách chọn trong các gói  $\{1, 2, \dots, i\}$  với giới hạn trọng lượng  $j$ . Thì giá trị lớn nhất khi được chọn trong số  $n$  gói với giới hạn trọng lượng  $M$  chính là  $B[n, M]$ .

### **1. Công thức tính $B[i, j]$ .**

Với giới hạn trọng lượng  $j$ , việc chọn tối ưu trong số các gói  $\{1, 2, \dots, i - 1, i\}$  để có giá trị lớn nhất sẽ có hai khả năng:

- Nếu không chọn gói thứ  $i$  thì  $B[i, j]$  là giá trị lớn nhất có thể bằng cách chọn trong số các gói  $\{1, 2, \dots, i - 1\}$  với giới hạn trọng lượng là  $j$ . Tức là  $B[i, j] = B[i - 1, j]$
- Nếu có chọn gói thứ  $i$  (tất nhiên chỉ xét tới trường hợp này khi mà  $W_i \leq j$ ) thì  $B[i, j]$  bằng giá trị gói thứ  $i$  là  $V_i$  cộng với giá trị lớn nhất có thể có được bằng cách chọn trong số các gói  $\{1, 2, \dots, i - 1\}$  với giới hạn trọng lượng  $j - W_i$ . Tức là về mặt giá trị thu được:  $B[i, j] = V_i + B[i - 1, j - W_i]$

Vì theo cách xây dựng  $B[i, j]$  là giá trị lớn nhất có thể nên nó sẽ là max trong hai giá trị thu được ở trên.

### **2. Cơ sở quy hoạch động:**

Để thấy  $B[0, j] =$  giá trị lớn nhất có thể bằng cách chọn trong số 0 gói = 0.

### **3. Tính bảng phương án:**

Bảng phương án  $B$  gồm  $n + 1$  dòng,  $M + 1$  cột, trước tiên được điền cơ sở quy hoạch động: Dòng 0 gồm toàn số 0. Sử dụng công thức truy hồi, dùng dòng 0 tính dòng 1, dùng dòng 1 tính dòng 2, v.v... đến khi tính hết dòng  $n$ .

	0	1	...	M
0	0	0	0	0
1				
2				
...	...			
n				



#### 4. Truy vết:

Tính xong bảng phương án thì ta quan tâm đến  $b[n, M]$  đó chính là giá trị lớn nhất thu được khi chọn trong cả n gói với giới hạn trọng lượng M. Nếu  $b[n, M] = b[n - 1, M]$  thì tức là không chọn gói thứ n, ta truy tiếp  $b[n - 1, M]$ . Còn nếu  $b[n, M] \neq b[n - 1, M]$  thì ta thông báo rằng phép chọn tối ưu có chọn gói thứ n và truy tiếp  $b[n - 1, M - W_n]$ . Cứ tiếp tục cho tới khi truy lên tới hàng 0 của bảng phương án.

```
const
    max          = 100;
var
    W, V        : array[1..max] of longint;
    B           : array[0..max, 0..max] of longint;
    n, M         : longint;
procedure Enter;
var
    i: longint;
begin
    Write('n = '); Readln(n);
    for i := 1 to n do
        begin
            Writeln('Pack ', i);
            Write('  + Weight : '); Readln(W[i]);
            Write('  + Value   : '); Readln(V[i]);
        end;
    Write('M = '); Readln(M);
end;
procedure Optimize;
var
    i, j: longint;
begin
    FillChar(B[0], SizeOf(B[0]), 0);
    for i := 1 to n do
        for j := 0 to M do
            begin
                B[i, j] := B[i - 1, j];
                if (j >= W[i]) and (B[i, j] < B[i-1, j-W[i]] + V[i])
                then
                    B[i, j] := B[i - 1, j - W[i]] + V[i];
            end;

```

```

end;
procedure Trace;
begin
  Writeln('Max Value : ', B[n, M]);
  Writeln('Selected Packs: ');
  while n <> 0 do
    begin
      if B[n, M] <> B[n - 1, M] then
        begin
          Writeln('Pack ', n, ' W = ', W[n], ' Value = ', V[n]);
          M := M - W[n];
        end;
      Dec(n);
    end;
  end;
BEGIN
  Enter;
  Optimize;
  Trace;
END.

```

## Bài tập

- 4.1.** Cho danh sách tên của  $n$  ( $n \leq 10$ ) học sinh (các tên đôi một khác nhau) và một số nguyên dương  $k$  ( $k \leq n$ ). Hãy liệt kê tất cả các cách chọn  $k$  học sinh trong  $n$  học sinh.

Ví dụ:

Dữ liệu vào	Kết quả ra
$n = 4, k = 2$ , danh sách tên học sinh như sau: An Bin Hong Minh	Có 6 cách chọn 2 học sinh trong 4 học sinh: 1. An Bin 2. An Hong 3. An Minh 4. Bin Hong 5. Bin Minh 6. Hong Minh

- 4.2. Một dãy nhị phân độ dài  $n$  ( $n \leq 10$ ) là một dãy  $x = x_1x_2 \dots x_n$  trong đó  $x_i \in \{0,1\}$ ,  $i = 1,2,\dots,n$ . Hãy liệt kê tất cả các dãy nhị phân độ dài  $n$

Dữ liệu vào	Kết quả ra
$n = 3$	Có 8 dãy nhị phân độ dài 3 1. 000 2. 001 3. 010 4. 011 5. 100 6. 101 7. 110 8. 111

- 4.3. Cho xâu S (độ dài không vượt quá 10) chỉ gồm các kí tự 'A' đến 'Z' (các kí tự trong xâu S đôi một khác nhau). Hãy liệt kê tất cả các hoán vị khác nhau của xâu S.

Dữ liệu vào	Kết quả ra
$S = 'XYZ'$	Có 6 hoán vị khác nhau của 'XYZ' 1. XYZ 2. XZY 3. YXZ 4. YZX 5. ZXY 6. ZYX

- 4.4. Cho số nguyên dương  $n$  ( $n \leq 20$ ), hãy liệt kê tất cả các xâu độ dài  $n$  chỉ gồm 2 kí tự 'A' hoặc 'B' mà không có 2 kí tự 'B' nào đứng cạnh nhau.

Dữ liệu vào	Kết quả ra
$n = 4$	Có 8 xâu độ dài 4 1. AAAA 2. AAAB 3. AABA 4. ABAA 5. ABAB

	6. BAAA 7. BAAB 8. BABA
--	-------------------------------

- 4.5. Cho dãy số  $A$  gồm  $N(N \leq 10)$  số nguyên  $a_1, a_2, \dots, a_N$  và một số nguyên dương  $K$  ( $1 < K < N$ ). Hãy đưa ra một cách chia dãy số thành  $K$  nhóm mà các nhóm có tổng bằng nhau.

Dữ liệu vào	Kết quả ra
$N=5, S=3$	nhóm 1: 4, 6
Dãy số $a$ :	nhóm 2: 1, 9
1, 4, 6, 9, 10	nhóm 3: 10

- 4.6. Một xâu  $X = x_1x_2..x_M$  được gọi là xâu con của xâu  $Y = y_1y_2..y_N$  nếu ta có thể nhận được xâu  $X$  từ xâu  $Y$  bằng cách xoá đi một số kí tự, tức là tồn tại một dãy các chỉ số:

$$1 \leq i_1 < i_2 < \dots < i_M \leq N \text{ để } x_1 = y_{i_1}, x_2 = y_{i_2}, \dots, x_M = y_{i_M}$$

Ví dụ:  $X='adz'$  là xâu con của xâu  $Y='baczdtz'$ ;  $i_1 = 2 < i_2 = 5 < i_3 = 7$ .

Nhập vào một xâu  $S$  (độ dài không quá 15, chỉ gồm các kí tự 'a' đến 'z'), hãy liệt kê tất cả các xâu con khác nhau của xâu  $S$ .

Dữ liệu vào	Kết quả ra
$S='aba'$	Có 6 xâu con khác nhau của 'aba' 1. a 2. b 3. aa 4. ab 5. ba 6. aba

- 4.7. Cho số nguyên dương  $n$  ( $n \leq 10$ ), liệt kê tất cả các cách khác nhau đặt  $n$  dấu ngoặc mở và  $n$  dấu ngoặc đóng đúng đắn?

Dữ liệu vào	Kết quả ra
$n = 3$	Có 5 cách $(( ))$ , $(( )( ))$ , $(( ))()$ , $( )( ( ))$ , $( ) ( ) ( )$

- 4.8.** Cho  $n$  ( $n \leq 10$ ) số nguyên dương  $a_1, a_2, \dots, a_n$  ( $a_i \leq 10^9$ ). Tìm số nguyên dương  $m$  nhỏ nhất sao cho  $m$  không phân tích được dưới dạng tổng của một số các số (mỗi số sử dụng không quá một lần) thuộc  $n$  số trên.

Dữ liệu vào	Kết quả ra
$n=4$ Dãy số $a$ : $1, 2, 3, 6$	13

- 4.9.** Cho xâu S (độ dài không vượt quá 10) chỉ gồm các kí tự 'A' đến 'Z' (các kí tự trong xâu S không nhất thiết phải khác nhau). Hãy liệt kê tất cả các hoán vị khác nhau của xâu S.

Dữ liệu vào	Kết quả ra
$S = 'ABA'$	Có 3 hoán vị khác nhau của 'ABA' 1. AAB 2. ABA 3. BAA

#### 4.10. Bài toán mā đi tuần

Cho bàn cờ  $n \times n$  ô, tìm cách di chuyển một quân mā (mā di chuyển theo luật cờ vua) trên bàn cờ xuất phát từ ô  $(1,1)$  đi qua tất cả các ô, mỗi ô qua đúng một lần.

Ví dụ: N=5

1	24	13	18	7
14	19	8	23	12
9	2	25	6	17
20	15	4	11	22
3	10	21	16	5

- 4.11.** Số siêu nguyên tố là số nguyên tố mà khi bỏ một số tuỳ ý các chữ số bên phải của nó thì phần còn lại vẫn tạo thành một số nguyên tố.

Ví dụ: 2333 là một số siêu nguyên tố có 4 chữ số vì 233, 23, 2 cũng là các số nguyên tố.

Cho số nguyên dương  $N$  ( $0 < N < 10$ ), đưa ra các số siêu nguyên tố có  $N$  chữ số cùng số lượng của chúng.

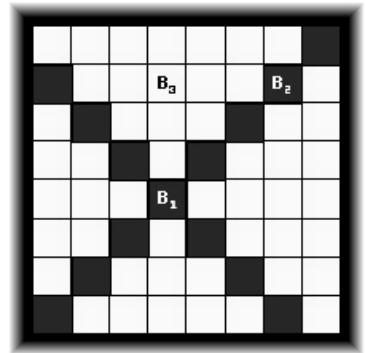
Ví dụ: Với  $N=4$

Có 16 số: 2333 2339 2393 2399 2939 3119 3137 3733 3739 3793 3797 5939 7193 7331 7333 7393

- 4.12.** Cho một xâu  $S$  (chỉ gồm các kí tự '0' đến '9', độ dài nhỏ hơn 10) và số nguyên  $M$ , hãy đưa ra một cách chèn vào  $S$  các dấu '+' hoặc '-' để thu được số  $M$  cho trước (nếu có thể).

Ví dụ:  $M = 8$ ,  $S='123456789'$  một cách chèn: '-1+2-3+4+5-6+7';

- 4.13.** Trong cờ vua quân tượng chỉ có thể di chuyển theo đường chéo và hai quân tượng có thể chiếu nhau nếu chúng nằm trên đường di chuyển của nhau. Trong hình bên, hình vuông tô đậm thể hiện các vị trí mà quân tượng  $B_1$  có thể đi tới được, quân tượng  $B_1$  và  $B_2$  chiếu nhau, quân  $B_1$  và  $B_3$  không chiếu nhau. Cho kích thước  $N$  của bàn cờ và  $K$  quân tượng, hỏi có bao nhiêu cách đặt các quân tượng vào bàn cờ mà các quân tượng không chiếu nhau.



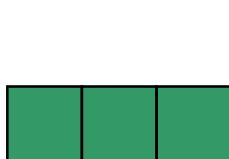
Dữ liệu vào trong file: "bishops.inp" có dạng:

- Dòng đầu là số  $t$  là số test ( $t \leq 10$ )
- $t$  dòng sau mỗi dòng chứa 2 số nguyên dương  $N, K$  ( $2 \leq N \leq 10, 0 < K \leq N^2$ )

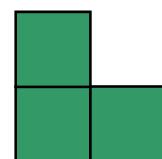
Kết quả ra file: "bishops.out" gồm  $t$  dòng, mỗi chứa một số duy nhất là số cách đặt các quân tượng vào bàn cờ tương ứng với dữ liệu vào.

- 4.14.**  $N$ -mino là hình thu được từ  $N$  hình vuông  $1 \times 1$  ghép lại (cạnh kề cạnh). Hai  $n$ -mino được gọi là đồng nhất nếu chúng có thể đặt chồng khít lên nhau. Cho số nguyên dương  $N$  ( $1 < N < 8$ ), tính và vẽ ra tất cả các  $N$ -mino trên màn hình.

Ví dụ: Với  $N=3$  chỉ có hai loại  $N$ -mino sau đây:



3-mino thẳng



3-mino hình thợ thợ

- 4.15.** Trong mục 2.2, lời giải bài toán TSP là một giải pháp nhánh cận rất thô sơ. Hãy thử chạy chương trình với trường hợp như sau: số thành phố  $n = 20$ ,

khoảng cách giữa các thành phố bằng 1 (nghĩa là  $C[i, j] = 1$  với  $i \neq j$ ). Hãy rút ra nhận xét và có thể đánh giá nhánh cận chặt hơn nữa làm tăng hiệu quả của chương trình.

- 4.16.** Cho bàn cờ quốc tế  $8 \times 8$  ô, mỗi ô ghi một số nguyên dương không vượt quá 32000.

Yêu cầu: Xếp 8 quân hậu lên bàn cờ sao cho không quân nào không chế được quân nào và tổng các số ghi trên các ô mà quân hậu đứng là lớn nhất.

Dữ liệu vào: gồm 8 dòng, mỗi dòng ghi 8 số nguyên dương, giữa các số cách nhau một dấu cách.

Kết quả ra: một số duy nhất là đáp số của bài toán.

Dữ liệu vào	Kết quả ra
1 2 4 9 3 2 1 4	66
6 9 5 4 2 3 1 4	
3 6 2 3 4 1 8 3	
2 3 7 3 2 1 4 2	
1 2 3 2 3 9 2 1	
2 1 3 4 2 4 2 8	
2 1 3 2 8 4 2 1	
8 2 3 4 2 3 1 2	

- 4.17.** Một chiếc ba lô có thể chứa được một khối lượng  $w$ . Có  $n$  ( $n \leq 20$ ) đồ vật được đánh số  $1, 2, \dots, n$ . Đồ vật  $i$  có khối lượng  $a_i$  và có giá trị  $c_i$ . Cần chọn các đồ vật cho vào ba lô để tổng giá trị các đồ vật là lớn nhất.

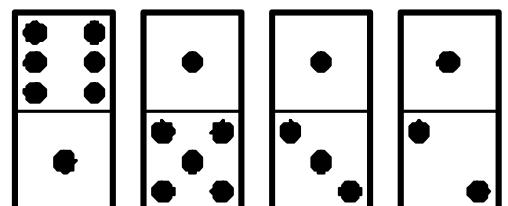
#### 4.18. *Dominoes*

Có  $N$  quân Domino xếp thành một hàng như hình vẽ

Mỗi quân Domino được chia làm hai phần, phần trên và phần dưới. Trên mặt mỗi phần có từ 1 đến 6 dấu chấm.

Ta nhận thấy rằng:

Tổng số dấu chấm ở phần trên của  $N$  quân Domino bằng:  $6+1+1+1=9$ , tổng số dấu chấm ở phần dưới của  $N$  quân Domino bằng  $1+5+3+2=11$ , độ chênh lệch giữa tổng trên và tổng dưới bằng  $|9-11|=2$



Với mỗi quân, bạn có thể quay  $180^\circ$  để phần trên trở thành phần dưới, phần dưới trở thành phần trên, và khi đó độ chênh lệch có thể được thay đổi. Ví dụ như ta quay quân Domino cuối cùng của hình trên thì độ chênh lệch bằng 0

Bài toán đặt ra là: Cần quay ít nhất bao nhiêu quân Domino nhất để độ chênh lệch giữa phần trên và phần dưới là nhỏ nhất.

*Dữ liệu vào trong file: “DOMINO.INP” có dạng:*

- Dòng đầu là số nguyên dương N ( $1 \leq N \leq 20$ )
- N dòng sau, mỗi dòng hai số  $a_i, b_i$  là số dấu chấm ở phần trên, số dấu chấm ở phần dưới của quân Domino thứ i ( $1 \leq a_i, b_i \leq 6$ )

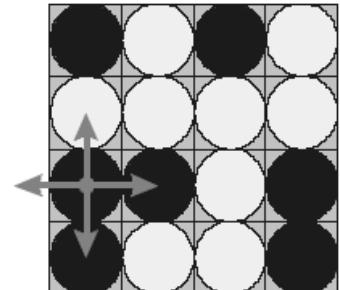
*Kết quả ra file: “DOMINO.OUT” có dạng:* Gồm 1 dòng duy nhất chứa 2 số nguyên cách nhau một dấu cách là độ chênh lệch nhỏ nhất và số quân Domino cần quay ít nhất để được độ chênh lệch đó.

**4.19.** Cho một lưới  $M \times N$  ( $M, N \leq 10$ ) ô, mỗi ô đặt một bóng đèn bật hoặc tắt. Trên mỗi dòng và mỗi cột có một công tắc. Nếu tác động vào công tắc dòng  $i$  ( $i=1..M$ ) hoặc công tắc cột  $j$  ( $j=1..N$ ) thì tất cả các bóng đèn trên dòng  $i$  hoặc cột  $j$  sẽ thay đổi trạng thái. Hãy tìm cách tác động vào các công tắc để được nhiều đèn sáng nhất.

**4.20.** Có 16 đồng xu xếp thành bảng  $4 \times 4$ , mỗi đồng xu có thể úp hoặc ngửa như hình vẽ sau:

Màu đen thể hiện đồng xu úp, màu trắng thể hiện đồng xu ngửa.

Tại mỗi bước ta có phép biến đổi sau: Chọn một đồng xu và thay đổi trạng thái của đồng xu đó và tắt cả các đồng xu nằm ở các ô chung cạnh (úp thành ngửa, ngửa thành úp). Cho trước một trạng thái các đồng xu, hãy lập trình tìm số phép biến đổi ít nhất để đưa về trạng thái tất cả các đồng xu hoặc đều úp hoặc đều ngửa.



*Dữ liệu vào trong file “COIN.INP” có dạng:* Gồm 4 dòng, mỗi dòng 4 ký tự 'w' - mô tả trạng thái ngửa hoặc 'b' - mô tả trạng thái úp.

*Kết quả ra file “COIN.OUT” có dạng:* Nếu có thể biến đổi được ghi số phép biến đổi ít nhất nếu không ghi “**Impossible**”

COIN.INP	COIN.OUT	COIN.INP	COIN.OUT
bwbw wwww	<b>Impossible</b>	bwwb bbwb	4

bbwb		bwwb	
bwwb		bwww	

4.21. Có N file chương trình với dung lượng  $S_1, S_2, \dots, S_n$  và loại đĩa CD có dung lượng D. Hỏi cần ít nhất bao nhiêu đĩa CD để có thể copy đủ tất cả các file chương trình (một file chương trình chỉ nằm trong một đĩa CD).

a) Giải bài toán bằng phương pháp nhánh cành với  $N \leq 10$ .

b) Giải bài toán bằng một thuật toán tham ăn với  $N \leq 100$ .

Dữ liệu vào	Kết quả ra
$N=5, D=700$ $320, 100, 300, 560, 50$	Cần ít nhất 2 đĩa CD Đĩa 1: 320, 300, 50 Đĩa 2: 100, 560

4.22. Chương trình giải bài toán lập lịch giảm thiểu trễ hạn (ở mục 3.4) có độ phức tạp  $O(N^3)$ , hãy cải tiến hàm check để nhận được chương trình với độ phức tạp  $O(N^2)$ .

4.23. Cho một xâu S (độ dài không quá 200) chỉ gồm 3 loại kí tự 'A', 'B', 'C'. Ta có phép đổi chỗ hai kí tự bất kì trong xâu, hãy tìm cách biến đổi ít bước nhất để được xâu theo thứ tự tăng dần.

Dữ liệu vào	Kết quả ra
$S = 'CBABA'$	Cần ít nhất 2 phép biến đổi <b>C</b> BABA <b>A</b> $\rightarrow$ <b>A</b> B <b>A</b> BC $\rightarrow$ AABBC

4.24. Cho  $N$  ( $N \leq 1000$ ) đoạn số nguyên  $[a_i, b_i]$ , hãy chọn một tập gồm ít số nhất mà mỗi đoạn số nguyên trên đều có ít nhất 2 số thuộc tập.

$$(|a_i|, |b_i| \leq 10^9)$$

Ví dụ: có 5 đoạn  $[0,10], [2,3], [4,7], [3,5], [5,8]$ , ta chọn tập gồm 4 số

$$\{2, 3, 5, 7\}$$

4.25. Cho phân số  $M/N$  ( $0 < M < N$ ,  $M, N$  nguyên). Hãy phân tích phân số này thành tổng các phân số có tử số bằng 1, càng ít số hạng càng tốt.

*Dữ liệu vào từ file "PS.IN" chứa 2 số M, N*

*Kết quả ra file "PS.OUT"*

- Dòng đầu là số lượng số tách

- Các dòng sau mỗi dòng chứa mẫu số của các số hạng

Dữ liệu vào	Kết quả ra
5 6	2
	2 3

4.26. Cho một số tự nhiên N. Hãy tìm cách phân tích số N thành các số nguyên dương  $p_1, p_2, \dots, p_k$  (với  $k > 1$ ) sao cho:

- $p_1, p_2, \dots, p_k$  đôi một khác nhau
- $p_1 + p_2 + \dots + p_k = N$
- $S = p_1 * p_2 * \dots * p_k$  đạt giá trị lớn nhất

Dữ liệu vào trong file: “PT.INP” có dạng: Gồm nhiều test, mỗi dòng là một test chứa một số N ( $5 \leq N \leq 1000$ )

Kết quả ra file: “PT.OUT” có dạng: Gồm nhiều dòng, mỗi dòng là tích lớn nhất đạt được (số S) cho test đó

Dữ liệu vào	Kết quả ra
5	6
7	12

4.27. Cho hai phép toán \*2 (nhân với 2) và /3 (chia nguyên cho 3). Cho trước số 1, bằng cách sử dụng hai phép toán trên ta xây dựng được biểu thức có giá trị bằng N.

Ví dụ N=6 thì  $1 * 2 * 2 * 2 * 2 * 2 / 3 / 3 * 2 = 6$  (thực hiện từ trái qua phải)

Dữ liệu vào từ file “BT.INP” chứa số N (N có không quá 100 chữ số)

Ghi kết quả ra file “BT.OUT” biểu thức ngắn nhất có thể

4.28. Cho số nguyên dương N ( $N \leq 10^{100}$ ), hãy tách N thành tổng ít các số Fibonacci nhất.

Ví dụ:  $N = 16 = 1 + 5 + 13$

4.29. Cần phải tổ chức việc thực hiện N chương trình đánh số từ 1 đến N trên một máy tính. Mỗi chương trình thứ i đòi hỏi thời gian tính là 1 giờ, và nếu nó được hoàn thành trước thời điểm  $d[i]$  (giả sử thời điểm bắt đầu thực hiện các chương trình là 0) thì người chủ máy tính sẽ được trả tiền công là  $w[i]$  ( $i = 1, 2, \dots, N$ ). Việc thực hiện mỗi chương trình phải được tiến hành liên tục từ lúc bắt đầu cho đến khi kết thúc không cho phép ngắt quãng, đồng thời tại mỗi thời điểm máy chỉ có thể thực hiện một chương trình).

Hãy tìm trình tự thực hiện các chương trình sao cho tổng tiền công nhận được là lớn nhất.

Dữ liệu vào được cho trong JOB.INP:

- Dòng đầu tiên chứa số N ( $N \leq 5000$ ),
- Dòng thứ i trong N dòng tiếp theo chứa 2 số d[i], w[i] được ghi cách nhau bởi dấu cách.

Kết quả đưa ra file JOB.OUT:

- Dòng đầu tiên chứa tổng tiền công nhận được theo trình tự tìm được.
- Dòng tiếp theo ghi trình tự thực hiện các chương trình.

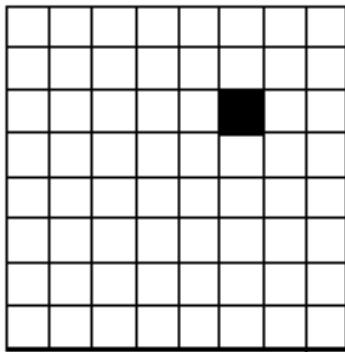
**4.30.** Tìm K chữ số cuối cùng của  $M^N$  ( $0 < K \leq 9$ ,  $0 \leq M, N \leq 10^9$ )

Ví dụ: K=2, M=2, N=10, ta có  $2^{10} = 1024$ , như vậy 2 chữ số cuối cùng của  $2^{10}$  là 24

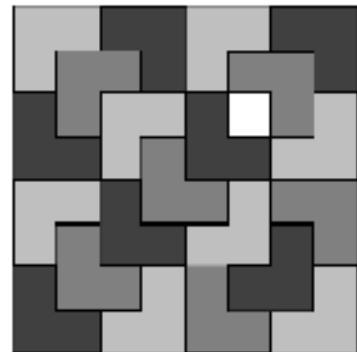
**4.31.** Viết hàm kiểm tra tính nguyên tố của số  $N$  ( $N \leq 10^9$ ) theo Fermat.

**4.32. Lát gạch**

Cho một nền nhà hình vuông có kích thước  $2^k$  bị khuyết một ô, hãy tìm cách lát nền nhà bằng loại gạch hình thớ (tạo bởi 3 hình vuông đơn vị).



nền nhà (ô màu đen là ô khuyết)



một cách lát nền

**4.33.** Cho dãy  $a_1, a_2, \dots, a_n$ , các số đôi một khác nhau và số nguyên dương  $k$  ( $1 \leq k \leq n$ ). Hãy đưa ra giá trị nhỏ thứ  $k$  trong dãy.

Ví dụ: dãy gồm 5 phần tử: 5, 7, 1, 3, 4 và  $k = 3$  thì giá trị nhỏ thứ  $k$  là 4.

**4.34. Dãy con lòi**

Dãy số nguyên  $A_1, A_2, \dots, A_N$  được gọi là lòi, nếu nó giảm dần từ  $A_1$  đến một  $A_i$  nào đó, rồi tăng dần tới  $A_N$ .

Ví dụ dãy lòi: 10 5 4 2 -1 4 6 8 12

**Yêu cầu:** Cho một dãy số nguyên, bằng cách xóa bỏ một số phần tử của dãy và giữ nguyên trình tự các phần tử còn lại, ta nhận được dãy con lòi dài nhất.

Dữ liệu vào trong file: DS.INP

- Đầu là N ( $N \leq 10000$ )
- Các dòng sau là N số nguyên của dãy số (các số kiểu longint)

Kết quả ra file: DS.OUT

- Ghi số phần tử của dãy con tìm được
- Các dòng tiếp theo ghi các số thuộc dãy con

#### 4.35. *Palindrome*

Một xâu được gọi là xâu đối xứng nếu đọc từ trái qua phải cũng giống như đọc từ phải qua trái. Ví dụ xâu “madam” là một xâu đối xứng. Bài toán đặt ra là cho một xâu S gồm các kí tự thuộc tập  $['a'..'z']$ , hãy tìm cách chèn vào xâu S ít nhất các kí tự để xâu S thành xâu đối xứng.

Ví dụ: xâu “adbhbca” ta sẽ chèn thêm 2 kí tự (c và d) để được xâu đối xứng “adcbhbcda

Dữ liệu vào trong file PALIN.INP có dạng: Gồm một dòng chứa xâu S. (độ dài mỗi xâu không vượt quá 200)

Kết quả ghi ra file PALIN.OUT có dạng: Gồm một dòng là một xâu đối xứng sau khi đã chèn thêm ít kí tự nhất vào xâu S.

Palin.inp	Palin.out
acbcd	<u>a</u> <u>d</u> <u>c</u> <u>b</u> <u>c</u> <u>d</u> <u>a</u>

#### 4.36. *Stones*

Có N đống sỏi xếp thành một hàng, đống thứ i có  $A_i$  viên sỏi. Ta có thể ghép hai đống sỏi kề nhau thành một đống và mất một chi phí bằng tổng hai đống sỏi đó.

**Yêu cầu:** Hãy tìm cách ghép N đống sỏi này thành một đống với chi phí là nhỏ nhất.

Ví dụ: Có 5 đống sỏi

$$\begin{array}{ccccccc} 4 & & \underline{\underline{1}} & \underline{\underline{2}} & 7 & & 5 \\ & \underline{\underline{4}} & & \underline{\underline{3}} & & 7 & 5 \\ & & 7 & & \underline{\underline{7}} & \underline{\underline{5}} & \\ & & 7 & & & 12 & \\ & & \hline & & & & 19 \end{array}$$

$$\text{Phạt} = 3 + 7 + 12 + 19 = 41$$

Dữ liệu vào trong file “STONES.INP” có dạng:

- Dòng đầu là số N ( $N < 101$ ) là số đồng sỏi
  - Dòng thứ 2 gồm N số nguyên là số sỏi của N đồng sỏi. ( $0 < A_i < 1001$ )
- Kết quả ra file “STONES.OUT” có dạng: gồm một số là chi phí nhỏ nhất để ghép N đồng thành một đồng.

STONES.INP	STONES.OUT
5	41
4 1 2 7 5	

#### 4.37. Cắt hình 1

Có một hình chữ nhật  $M \times N$  ô, mỗi lần ta được phép cắt một hình chữ nhật thành hai hình chữ nhật con theo chiều ngang hoặc chiều dọc và lại tiếp tục cắt các hình chữ nhật con cho đến khi được hình vuông thì dừng.

Hỏi có thể cắt hình chữ nhật  $M \times N$  thành ít nhất bao nhiêu hình vuông.

Dữ liệu vào trong file HCN.INP:

Gồm 1 số dòng, mỗi dòng là 1 test là một cặp số M, N ( $1 \leq M, N \leq 100$ )

Kết quả ra file HCN.INP:

Gồm 1 số dòng là kết quả tương ứng với dữ liệu vào

#### 4.38. Cắt hình 2

Cho một bảng số A gồm M dòng, N cột, các giá trị của bảng A chỉ là 0 hoặc 1. Ta muốn cắt bảng A thành các hình chữ nhật con sao cho các hình chữ nhật con có giá trị toàn bằng 1 hay toàn bằng 0. Một lần cắt là một nhát cắt thẳng theo dòng hoặc theo cột của một hình chữ nhật thành hai hình chữ nhật riêng biệt. Cứ tiếp tục cắt cho đến khi hình chữ nhật toàn bằng 1 hay toàn bằng 0. Hãy tìm cách cắt để được ít hình chữ nhật nhất mà các hình chữ nhật con có giá trị toàn bằng 1 hay toàn bằng 0.

Ví dụ: Bảng số  $5 \times 5$  sau được chia thành 8 hình chữ nhật con.

0	1	0	0	1
0	1	0	0	1
1	1	0	0	1
1	1	1	0	0
0	0	1	0	0

0	1	0	0	1
0	1	0	0	1
1	1	0	0	1
1	1	1	0	0
0	0	1	0	0

Dữ liệu vào trong file HCN2.INP

- Dòng đầu là 2 số nguyên dương M, N ( $M, N \leq 30$ )
  - M dòng tiếp theo, mỗi dòng N số chỉ gồm 0 hoặc 1 thể hiện bảng số A
- Kết quả ra file HCN2*
- Gồm 1 dòng duy nhất chứa một số duy nhất là số hình chữ nhật ít nhất

#### 4.39. TKSEQ

Cho dãy số A gồm N số nguyên và số nguyên K. Tìm dãy chỉ số  $1 \leq i_1 < i_2 < \dots < i_{3K} \leq N$  sao cho:

$$S = (a_{i_1} - a_{i_2} + a_{i_3}) + (a_{i_4} - a_{i_5} + a_{i_6}) + \dots + (a_{i_{3K-2}} - a_{i_{3K-1}} + a_{i_{3K}})$$

đạt giá trị lớn nhất.

*Dữ liệu vào trong file “TKSEQ.INP” có dạng:*

- Dòng đầu là gồm 2 số nguyên N, K ( $0 < 3K \leq N \leq 500$ )
- Dòng 2 gồm N số nguyên  $a_1, a_2, \dots, a_N$  ( $|a_i| < 10^9$ )

*Kết quả ra file “TKSEQ.OUT” có dạng:* gồm một số duy nhất S lớn nhất tìm được

TKSEQ.INP	TKSEQ.OUT
5 1	4
1 2 3 4 5	

#### 4.40. Least-Squares Segmentation

Ta định nghĩa trọng số của đoạn số từ số ở vị trí thứ i đến vị trí thứ j của dãy số nguyên A[1], A[2], ..., A[N] là:

$$\sum_{k=i}^j (A[k] - mean)^2 \text{ trong đó } mean = (\sum_{k=i}^j A[k]) / (j - i + 1)$$

**Yêu cầu:** Cho dãy số nguyên A gồm N số A[1], A[2], ..., A[N] và số nguyên dương G ( $1 < G^2 < N$ ). Hãy chia dãy A thành đúng G đoạn để tổng trọng số là nhỏ nhất.

*Dữ liệu vào trong file văn bản “LSS.INP” có dạng:*

- Dòng đầu gồm hai số N và G ( $1 < G^2 < N < 1001$ )
- N dòng tiếp theo, mỗi dòng một số nguyên mô tả dãy số A ( $0 < A[i] < 10^6$ )

*Kết quả ra file văn bản “LSS.OUT” có dạng:* gồm một dòng chứa một số thực duy nhất là đáp án của bài toán. (đưa ra theo quy cách :0:2)

LSS.INP	LSS.OUT
5 2	0.50
3	

3	
3	
4	
5	

#### 4.41. Phân trang (Đề thi chọn đội tuyển quốc gia 1999)

Văn bản là một dãy gồm N từ đánh số từ 1 đến N. Từ thứ i có độ dài là  $w_i$  ( $i=1, 2, \dots, N$ ). Phân trang là một cách xếp lần lượt các từ của văn bản vào dãy các dòng, mỗi dòng có độ dài L, sao cho tổng độ dài của các từ trên cùng một dòng không vượt quá L. Ta gọi hệ số phạt của mỗi dòng trong cách phân trang là hiệu số (L-S), trong đó S là tổng độ dài của các từ xếp trên dòng đó. Hệ số phạt của cách phân trang là giá trị lớn nhất trong số các hệ số phạt của các dòng.

**Yêu cầu:** Tìm cách phân trang với hệ số phạt nhỏ nhất.

Dữ liệu vào từ tệp văn bản PTRANG.INP

- Dòng 1 chứa 2 số nguyên dương N, L ( $N \leq 4000, L \leq 70$ )
- Dòng thứ i trong số N dòng tiếp theo chứa số nguyên dương  $w_i$  ( $w_i \leq L$ ),  $i=1, 2, \dots, N$

Kết quả ghi ra file văn bản PTRANG.OUT

- Dòng 1 ghi 2 số P, Q theo thứ tự là hệ số phạt và số dòng theo cách phân trang tìm được
- Dòng thứ i trong số Q dòng tiếp theo ghi chỉ số của các từ trong dòng thứ i của cách phân trang.

#### 4.42. Chọn số

Cho mảng A có kích thước NxN gồm các số nguyên không âm. Hãy chọn ra K số sao cho mỗi dòng có nhiều nhất 1 số được chọn, mỗi cột có nhiều nhất 1 số được chọn để tổng K số là lớn nhất.

Dữ liệu vào từ tệp văn bản SELECT.INP

- Dòng thứ nhất gồm 2 số N và K ( $K \leq N \leq 15$ )
- N dòng sau, mỗi dòng N số nguyên không âm  $A_{ij} < 10000$

Kết quả ghi ra file văn bản SELECT.OUT

Tổng lớn nhất chọn được và số cách chọn (cách nhau đúng một dấu cách)

Select.inp	Select.out
3 2	6 3
1 2 3	
2 3 1	
3 1 2	

#### 4.43. *Puzzle of numbers*

Khi một số phần chữ số trong đẳng thức đúng của tổng hai số nguyên bị mất (được thay bởi các dấu sao “\*”). Có một câu đố là: Hãy thay các dấu sao bởi các chữ số để cho đẳng thức vẫn đúng.

Ví dụ bắt đầu từ đẳng thức sau:

9334

789

-----

10123      (9334+789=10123)

Các ví dụ các chữ số bị mất được thay bằng các dấu sao như sau:

*3*4	hay	****
78*		***
10123		*****

Nhiệm vụ của bạn là viết chương trình thay các dấu sao thành các chữ số để được một đẳng thức đúng. Nếu có nhiều lời giải thì đưa ra một trong số đó. Nếu không có thì đưa ra thông báo: “**No Solution**”.

*Chú ý các chữ số ở đầu mỗi số phải khác 0.*

*Dữ liệu vào trong file “REBUSS.INP”:* gồm 3 dòng, mỗi dòng là một xâu kí tự gồm các chữ số hoặc kí tự “\*”. Độ dài mỗi xâu không quá 50 kí tự. Dòng 1, dòng 2 thể hiện là hai số được cộng, dòng 3 thể hiện là tổng hai số.

*Kết quả ra file “REBUSS.OUT”:* Nếu có lời giải thì file kết quả gồm 3 dòng tương ứng với file dữ liệu vào, nếu không thì thông báo “**No Solution**”

REBUSS.INP	REBUSS.OUT
*3*4	9334
78*	789
10123	10123

#### 4.44. Xếp lịch giảng

Một giáo viên cần giảng  $n$  ván đề được đánh số từ 1 đến  $n$  ( $n \leq 10000$ ). Mỗi một ván đề  $i$  cần có thời gian là  $t_i$  ( $i = 1..n$ ). Để giảng  $n$  ván đề đó thì giáo viên có các buổi đã được phân có độ dài là  $L$  ( $L \leq 500$ ).

- Một ván đề thì phải giải quyết trong một buổi .
- Ván đề  $i$  phải được giảng trước ván  $i + 1$  với mọi  $i = 1..(n - 1)$ .

Học sinh có thể ra về sớm nếu như buổi giảng đã kết thúc, tuy nhiên nếu thời gian ra về đó quá sớm so với buổi giảng thì thật là phí. Chính vì thế người ta đánh giá buổi lên lớp bằng giá trị  $DI$  như sau :

$$DI = \begin{cases} 0 & \text{nu } t = 0 \\ -C & \text{nu } 1 \leq t \leq 10 \\ (t - 10)^2 & \text{nu } t > 10 \end{cases}$$

Trong đó  $t$  là thời gian thừa của buổi lên lớp đó,  $C$  là một hằng số .

**Yêu cầu:** Hãy xếp lịch dạy sao cho tổng số các buổi là cần ít nhất có thể được. Trong các lịch dạy ít nhất đó, hãy tìm lịch dạy sao cho tổng số  $DI$  là nhỏ nhất có thể được.

*Dữ liệu vào từ file SCHEDULING.INP*

- Dòng đầu là số  $n$  (số ván đề cần giảng) .
- Dòng tiếp theo là  $L$  và  $C$
- Dòng cuối cùng là  $N$  số thể hiện cho  $t_1, t_2, \dots, t_n$  .

*Kết quả ra file SCHEDULING.OUT*

- Dòng đầu tiên là số buổi .
- Dòng tiếp theo là tổng  $DI$  nhỏ nhất đạt được .

SCHEDULING.INP	SCHEDULING.OUT
10	6
120 10	2700
80 80 10 50 30 20 40 30 120 100	

#### 4.45. Khu vườn (IOI 2008)

Ramsesses II thắng trận trở về. Để ghi nhận chiến tích của mình ông quyết định xây một khu vườn tráng lệ. Khu vườn phải có một hàng cây chạy dài từ cung điện của ông tại Luxor tới thánh đường Karnak. Hàng cây này chỉ chứa hai loại cây là sen và cói giấy, bởi vì chúng tương ứng là biểu tượng của miền Thượng Ai Cập và Hạ Ai Cập.

Vườn phải có đúng  $N$  cây. Ngoài ra, phải có sự cân bằng: ở mọi đoạn cây liên tiếp của vườn, số lượng sen và số lượng cói giấy phải không lệch nhau quá 2.

Vườn cây được biểu diễn dưới dạng xâu các kí tự 'L' (lotus – sen) và 'P' (papyrus – cói giấy). Ví dụ, với  $N = 5$  có tất cả 14 vườn đảm bảo cân bằng. Theo thứ tự từ điển, các vườn đó là: LLPLP, LLPPL, LPLLP, LPLPL, LPLPP, LPPLL, PPPLP, PLLPL, PLPLL, PLPLP, PLPPL, PPLLP và PPLPL.

Các vườn cân bằng với độ dài xác định cho trước được sắp xếp theo thứ tự từ điển và được đánh số từ 1 trở đi. Ví dụ, với  $N=5$ , vườn số 12 sẽ là vườn PLPPL.

### NHIỆM VỤ

Cho số cây  $N$  và xâu biểu diễn một vườn cân bằng, hãy lập trình tính số thứ tự của vườn này theo modun  $M$ , trong đó  $M$  là số nguyên cho trước.

Lưu ý rằng giá trị của  $M$  không đóng vai trò quan trọng trong việc giải bài toán, nó chỉ làm cho việc tính toán trở nên đơn giản.

### HẠN CHẾ

$$1 \leq N \leq 1\,000\,000$$

$$7 \leq M \leq 10\,000\,000$$

### CHẤM ĐIỂM

Có 40 điểm dành cho các dữ liệu vào với  $N$  không vượt quá 40.

### INPUT

Chương trình của bạn phải đọc từ file “GARDEN.INP” các dữ liệu sau:

- Dòng 1 chứa số nguyên  $N$ , số cây trong vườn,
- Dòng 2 chứa số nguyên  $M$ ,
- Dòng 3 chứa xâu gồm  $N$  kí tự 'L' (sen) hoặc 'P' (cói giấy) biểu diễn vườn cân bằng.

### OUTPUT

Chương trình của bạn phải ghi ra file “GARDEN.OUT” một dòng chứa một số nguyên trong phạm vi từ 0 đến  $M-1$ , là số thứ tự theo modun  $M$  của vườn được mô tả trong đầu vào.

Input ví dụ 1	Output ví dụ 1	Giải thích

5 7 PLPPL	5	Số thứ tự của PLPPL là 12. Như vậy output là 12 theo môđun 7, tức là 5.
-----------------	---	---

Input ví dụ 2	Output ví dụ 2
12	39
10000	
LPLLPLPPLPLL	

#### 4.46. Số rõ ràng

Bờm mới tìm được một tài liệu định nghĩa số rõ ràng như sau: Với số nguyên dương  $n$ , ta tạo số mới bằng cách lấy tổng bình phương các chữ số của nó, với số mới này ta lại lặp lại công việc trên. Nếu trong quá trình đó, ta nhận được số mới là 1, thì số  $n$  ban đầu được gọi là số rõ ràng. Ví dụ, với  $n = 19$ , ta có:

$$19 \rightarrow 82 (= 1^2 + 9^2) \rightarrow 68 \rightarrow 100 \rightarrow 1$$

Như vậy, 19 là số rõ ràng.

Không phải mọi số đều rõ ràng. Ví dụ, với  $n = 12$ , ta có:

$$12 \rightarrow 5 \rightarrow 25 \rightarrow 29 \rightarrow 85 \rightarrow 89 \rightarrow 145 \rightarrow 42 \rightarrow 20 \rightarrow 4 \rightarrow 16 \rightarrow 37 \rightarrow 58 \rightarrow 89 \rightarrow 145$$

Bờm rất thích thú với định nghĩa số rõ ràng này và thách đố phú ông: Cho một số nguyên dương  $n$ , tìm số  $S(n)$  là số rõ ràng liền sau số  $n$ , tức là  $S(n)$  là số rõ ràng nhỏ nhất lớn hơn  $n$ . Tuy nhiên, câu hỏi đó quá dễ với phú ông và phú ông đã đố lại Bờm: *Cho hai số nguyên dương  $n$  và  $k$  ( $1 \leq n, k \leq 10^{15}$ ), hãy tìm số  $S^k(n) = S(S(\dots S(n)))$  là số rõ ràng liền sau thứ  $k$  của  $n$ .*

Bạn hãy giúp Bờm giải câu đố này nhé!

Dữ liệu vào từ file văn bản *CLEAR.INP* có dạng:

- Dòng đầu là số  $t$  ( $0 < t \leq 20$ )
- $t$  dòng sau, mỗi dòng chứa 2 số nguyên  $n$  và  $k$ .

Kết quả ra file văn bản *CLEAR.OUT* gồm  $t$  dòng, mỗi dòng là kết quả tương ứng với dữ liệu vào.

CLEAR.INP	CLEAR.OUT
2	19
18 1	1000000000
1 145674807	

#### 4.47. Hải nấm

Bé Bông đi hái nấm trong N khu rừng đánh số từ 1 đến N, nhưng chỉ có M khu rừng có nấm. Việc di chuyển từ khu rừng thứ i sang khu rừng thứ j tốn  $t_{ij}$  đơn vị thời gian. Đến khu rừng i có nấm, cô bé có thể dừng lại để hái nấm. Nếu tổng số đơn vị thời gian cô bé dừng lại ở khu rừng thứ i là  $d_i$  ( $d_i > 0$ ), thì cô bé hái được:  $\left\lceil \frac{S_i}{2} \right\rceil + \left\lceil \frac{S_i}{4} \right\rceil + \dots + \left\lceil \frac{S_i}{2^{d_i}} \right\rceil$  cây nấm tại khu rừng đó (trong đó  $S_i$  là số lượng nấm có tại khu rừng i,  $[x]$  là phần nguyên của x). Giả thiết rằng ban đầu cô bé ở khu rừng thứ nhất và đi hái nấm trong thời gian không quá P đơn vị.

*Yêu cầu:* Hãy tính số lượng cây nấm nhiều nhất mà cô bé có thể hái được.

*Dữ liệu vào file văn bản MUSHROOM.INP:*

- Dòng đầu tiên chứa ba số nguyên dương M ( $M \leq 10$ ), N ( $0 < M \leq N \leq 100$ ) và P ( $P \leq 10000$ );
- M dòng tiếp theo, mỗi dòng chứa 2 số nguyên dương r và  $S_r$  nghĩa là khu rừng r có  $S_r$  nấm ( $S_r \leq 10^9$ );
- Dòng thứ i trong N dòng cuối cùng chứa N số nguyên dương  $t_{ij}$  ( $t_{ij} \leq 10000$ ), ( $i, j = 1, \dots, N$ ).

*Kết quả ghi ra file văn bản MUSHROOM.OUT:* số lượng cây nấm nhiều nhất bé Bông có thể hái được.

MUSHROOM.INP	MUSHROOM.OUT
2 2 2	3
1 5	
2 10	
0 3	
3 0	