

NGUYỄN XUÂN HUY

**SÁNG TẠO  
TRONG THUẬT TOÁN  
VÀ  
LẬP TRÌNH**

**với ngôn ngữ C++**

Tuyển các bài toán Tin nâng cao  
cho học sinh và sinh viên giỏi  
Tái bản và cập nhật

## M U C L U C

<b>CHƯƠNG 1 GIẢI MỘT BÀI TOÁN TIN</b>	<b>6</b>
<b>Bài 1.1. Số thân thiện</b>	<b>6</b>
<b>Bài 1.2. Số cấp cộng</b>	<b>8</b>
<b>Bài 1.3. Số cấp nhân</b>	<b>10</b>
<b>Bài 1.4. Mảng ngẫu nhiên</b>	<b>11</b>
<b>Bài 1.5. Chia mảng tỉ lệ 1:1</b>	<b>13</b>
<b>Bài 1.6. Chia mảng tỉ lệ 1:k</b>	<b>18</b>
<b>CHƯƠNG 2 SINH DỮ LIỆU VÀO VÀ RA</b>	<b>23</b>
<b>Bài 2.1. Sinh ngẫu nhiên theo khoảng</b>	<b>23</b>
<b>Bài 2.2. Sinh ngẫu nhiên tăng</b>	<b>24</b>
<b>Bài 2.3. Sinh hoán vị ngẫu nhiên</b>	<b>25</b>
<b>Bài 2.4. Sinh ngẫu nhiên đều</b>	<b>26</b>
<b>Bài 2.5. Sinh ngẫu nhiên tỉ lệ</b>	<b>28</b>
<b>Bài 2.6. Sinh ngẫu nhiên tệp tăng</b>	<b>31</b>
<b>Bài 2.7. Sinh ngẫu nhiên tệp cấp số cộng</b>	<b>32</b>
<b>Bài 2.8. Sinh ngẫu nhiên mảng đối xứng</b>	<b>34</b>
<b>Bài 2.9. Số độ cao h</b>	<b>36</b>
<b>Bài 2.10. Tệp các hoán vị</b>	<b>40</b>
<b>Bài 2.11. Đọc dữ liệu từ tệp vào mảng biết hai kích thước</b>	<b>42</b>
<b>Bài 2.12. Đọc dữ liệu từ tệp vào mảng [][ ] biết một kích thước</b>	<b>44</b>
<b>Bài 2.13. Đọc dữ liệu từ tệp vào mảng đối xứng</b>	<b>47</b>
<b>Bài 2.14. Đếm tàu</b>	<b>49</b>

Sáng tạo trong Thuật toán và Lập trình Tập I	3
<b>2.15 Sắp mảng</b>	<b>51</b>
<b>Bài 2.16. Sắp đoạn</b>	<b>60</b>
<b>CHƯƠNG 3 BÀN PHÍM VÀ MÀN HÌNH</b>	<b>67</b>
<b>Bài 3.1. Bảng mã ASCII</b>	<b>67</b>
<b>Bài 3.2. Bộ Tú lơ khớ</b>	<b>70</b>
<b>Bài 3.3. Trò chơi 15</b>	<b>73</b>
<b>Bài 3.4. Bảng nhảy</b>	<b>80</b>
<b>CHƯƠNG 4 TỔ CHỨC DỮ LIỆU</b>	<b>83</b>
<b>Bài 4.1. Cùm</b>	<b>83</b>
<b>Bài 4.2. Bài gộp</b>	<b>85</b>
<b>Bài 4.3. Chuỗi hạt</b>	<b>91</b>
<b>Bài 4.4. Sắp mảng rồi ghi tệp</b>	<b>97</b>
<b>Bài 4.5. abc - sắp theo chỉ dẫn</b>	<b>103</b>
<b>Bài 4.6. Xâu mẫu</b>	<b>107</b>
<b>CHƯƠNG 5 PHƯƠNG PHÁP THAM LAM</b>	<b>113</b>
<b>Bài 5.1. Bể nhạc</b>	<b>113</b>
<b>Bài 5.2. Xếp việc</b>	<b>115</b>
<b>Bài 5.3. Xếp ba lô</b>	<b>120</b>
<b>Bài 5.4. Cây khung ngắn nhất</b>	<b>123</b>
<b>Bài 5.5. Trộn hai tệp</b>	<b>129</b>
<b>Bài 5.6. Trộn nhiều tệp</b>	<b>132</b>
<b>Bài 5.7 Heap</b>	<b>136</b>

Sáng tạo trong Thuật toán và Lập trình Tập I	4
Bài 5.8 Thuật toán Huffman	140
<b>CHƯƠNG 6 PHƯƠNG PHÁP QUAY LUI</b>	<b>145</b>
Bài 6.1. Các quân Hậu	146
Bài 6.2. Từ chuẩn	168
Bài 6.3. Tìm đường trong mê cung.	173
<b>CHƯƠNG 7 QUY HOẠCH ĐỘNG</b>	<b>179</b>
Bài 7.1. Chia thưởng	179
Bài 7. 2. Palindrome	185
Bài 7.3. Cắm hoa	189
Bài 7.4. Tìm các đường ngắn nhất	197
<b>CHƯƠNG 8 SUY NGẪM</b>	<b>212</b>
Bài 8.1. Lát nền	212
Bài 8.2. Chữ số cuối khác 0	217
Bài 8.3. Hình chữ nhật tối đại trong ma trận 0/1.	221
Bài 8.4. Ma phương	228
Bài 8.5. Tháp Hà Nội cổ	245
Bài 8.6. Tháp Hà Nội xuôi	248
Bài 8.7. Tháp Hà Nội ngược	252
Bài 8.8. Tháp Hà Nội thẳng	255
Bài 8.9. Tháp Hà Nội sắc màu (Hà Nội Cầu vồng)	258



# CHƯƠNG 1

## GIẢI MỘT BÀI TOÁN TIN

### Bài 1.1. Số thân thiện

*Tìm tất cả các số tự nhiên hai chữ số mà khi đảo trật tự của hai chữ số đó sẽ thu được một số nguyên tố cùng nhau với số đã cho.*

#### Hiểu đầu bài

Ta kí hiệu  $(a, b)$  là ước chung lớn nhất (*ucln*) của hai số tự nhiên  $a$  và  $b$ . Hai số tự nhiên  $a$  và  $b$  được gọi là *nguyên tố cùng nhau* khi và chỉ khi  $a$  và  $b$  chỉ đồng thời chia hết cho 1, tức là  $(a, b) = 1$ . Khi đó, chẳng hạn:

✎  $(23, 32) = 1$ , vậy 23 là một số cần tìm. Theo tính chất *đối xứng*, ta có ngay 32 cũng là một số cần tìm.

✎  $(12, 21) = 3$ , vậy 12 và đồng thời 21 không phải là những số cần tìm.

#### Đặc tả

Gọi hai chữ số của số tự nhiên cần tìm  $x$  là  $a$  và  $b$ , ta có:

- $x = 10a + b$
- $a, b = 0..9$  ( $a$  và  $b$  biến thiên trong khoảng từ 0 đến 9).
- $a > 0$  vì  $x$  là số có hai chữ số.
- $(10a+b, 10b+a) = 1$ .

Ta kí hiệu  $x'$  là số đối xứng của số  $x$  theo nghĩa của đầu bài, khi đó ta có đặc tả như sau:

$x = 10..99$  ( $x$  biến thiên từ 10 đến 99)  
 $x = 10a+b \rightarrow x' = 10b+a$   
 $(x, x') = 1$ .

Nếu  $x = ab$  thì  $x' = ba$ . Ta có thể tính giá trị của  $x'$  theo công thức:

$$x' = (\text{chữ số hàng đơn vị của } x) * 10 + (\text{chữ số hàng chục của } x).$$

Kí hiệu  $\text{Đơn}(x)$  là toán tử lấy chữ số hàng đơn vị của số tự nhiên  $x$  và kí hiệu  $\text{Chục}(x)$  là toán tử lấy chữ số hàng chục của  $x$ , ta có:

$$x' = \text{Đơn}(x) * 10 + \text{Chục}(x).$$

Tổng hợp lại ta có đặc tả:

*Số cần tìm  $x$  phải thoả các tính chất sau:*

$x = 10..99$  ( $x$  nằm trong khoảng từ 10 đến 99).  
 $x' = \text{Đơn}(x) * 10 + \text{Chục}(x)$ .  
 $(x, x') = 1$  (ước chung lớn nhất của  $x$  và  $x'$  bằng 1).  
 $\text{Đơn}(x) = (x \bmod 10)$ : số dư của phép chia nguyên  $x$  cho 10, ví dụ:  
 $\text{Đơn}(19) = 19 \bmod 10 = 9$ .  
 $\text{Chục}(x) = (x \div 10)$ : thương nguyên của phép chia  $x$  cho 10, ví dụ:  
 $\text{Chục}(19) = 19 \div 10 = 1$ .  
 $\text{Lấy}(x)$ : hiển thị hoặc nạp giá trị  $x$  vào mảng  $s$

Đặc tả trên được thể hiện qua ngôn ngữ phỏng trình như sau:

```
for x = 10 to 99 do
  if Ucln(x, Đơn(x)*10+Chục(x))=1 then Lấy(x);
```

trong đó,  $ucln(a, b)$  là hàm cho ước chung lớn nhất của hai số tự nhiên  $a$  và  $b$ ;  $Lấy(x)$  là toán tử hiển thị  $x$  lên màn hình hoặc ghi  $x$  vào một mảng nào đó với mục đích sử dụng lại, nếu cần.

Ta làm mịn đặc tả:

$ucln(a, b)$ : Theo thuật toán Euclid là chia liên tiếp, thay số thứ nhất bằng dư của nó khi chia cho số thứ hai rồi hoán vị hai số.

```
// C++
// Uoc chung lon nhat cua hai so
// a va b. Thuat toan Euclid
int Ucln(int a, int b) {
  int r;
  while (b > 0) {
    r = a % b; // số dư r
    a = b; b = r; // chuyển vị
  }
  return a;
}
```

## Chương trình DevC

```
// So than thien
// cac so tu nhien 2 chu so xy:
// (xy, yx) = 1
#include <iostream>

using namespace std;

// Uoc chung lon nhat cua hai so
// a va b. Thuat toan Euclid
int Ucln(int a, int b) {
  int r;
  while (b > 0) {
    r = a % b; a = b; b = r;
  }
  return a;
}

int Lat(int x) {
  if (x < 10) return x;
  int y = 0;
  while(x != 0) {
    y = y * 10 + (x % 10);
    x /= 10;
  }
}
```

```

    }
    return y;
}

void BaiGiai() {
    int n = 0;
    for (int x = 10; x < 100; ++x) {
        if (Ucln(x, Lat(x)) == 1) {
            cout << " " << x;
            ++n;
            if (n % 10 == 0) {
                cout << endl;
            }
        }
    }
    cout << "\n Tong cong " << n << " so.";
}

main() {
    BaiGiai();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

10 13 14 16 17 19 23 25 29 31
32 34 35 37 38 41 43 47 49 52
53 56 58 59 61 65 67 71 73 74
76 79 83 85 89 91 92 94 95 97
98
Tong cong 41 so.
T h e   E n d
-----
Process exited after 0.07465 seconds with return value 0
Press any key to continue . . .

```

## Bài 1.2. Số cấp cộng

*Tìm các số tự nhiên lẻ có ba chữ số. Ba chữ số này, theo trật tự từ trái qua phải tạo thành một cấp số cộng.*

### Đặc tả

1.  $x$  là số tự nhiên có ba chữ số:  $x = 100*a + 10*b + c$ .
2.  $x$  là số lẻ nên chữ số hàng đơn vị  $c$  phải là số lẻ:  $c = 1, 3, 5, 7, 9$ .
3. Chữ số hàng trăm của  $x$  phải khác 0:  $a = 1..9$ .
4. Nếu dãy  $a, b, c$  lập thành một cấp số cộng thì số đứng giữa  $b$  là trung bình cộng của hai số đầu và cuối:  $b = (a + c)/2$  hay  $2b = a + c$ .  
 Từ (4) ta suy ra  $(a + c)$  là số chẵn. Do  $c$  lẻ,  $(a + c)$  chẵn nên  $a$  lẻ.  
 Nếu biết  $a$  và  $c$  ta tính được  $x = 100a + 10(a + c) / 2 + c$



$$= 100a + 5(a + c) + c = 105a + 6c.$$

Vì chỉ có 5 chữ số lẻ là 1, 3, 5, 7 và 9 nên tổ hợp của  $a$  và  $c$  sẽ cho ta 25 số.

### Thuật toán

[105\*a + 6\*c]  $a, c \in \{1, 3, 5, 7, 9\}$

Mô tả chi tiết

for each  $a = 1, 3, 5, 7, 9$  do

    for each  $c = 1, 3, 5, 7, 9$  do

        Hiển thị (105\*a + 6\*c)

    end for

end for

### Chương trình DevC

```
// Cac so tu nhien le 3 chu so abc
// tao thanh cap cong a, b, c

#include <iostream>

using namespace std;

int chuSoLe[] = {1,3,5,7,9};

void BaiGiai() {
    int a, x;
    int n = 0;
    for(int i = 0; i < 5; ++i) {
        a = 105*chuSoLe[i];
        for(int j = 0; j < 5; ++j) {
            x = a + 6*chuSoLe[j];
            cout << " " << x;
            ++n;
            if (n % 10 == 0) {
                cout << endl;
            }
        }
    }
    cout << "\n Tong cong " << n << " so.";
}

main() {
    BaiGiai();
    cout << "\n T h e   E n d";
    return 0;
}
```

### Output

```

111 123 135 147 159 321 333 345 357 369
531 543 555 567 579 741 753 765 777 789
951 963 975 987 999
Tong cong 25 so.
T h e   E n d

```

```

-----
Process exited after 0.06888 seconds with return value 0
Press any key to continue . . .

```

### Phương pháp sinh

*Thay vì duyệt tìm các đối tượng  
hãy sinh ra chúng.*

## Bài 1.3. Số cấp nhân

*Tìm các số tự nhiên có ba chữ số. Ba chữ số này, theo trật tự từ trái qua phải tạo thành một cấp số nhân với công bội là một số tự nhiên khác 0.*

### Đặc tả

Chú ý rằng ta chỉ xét các cấp số trên dãy số tự nhiên với công bội  $d$  là một số nguyên dương. Gọi  $x$  là số cần tìm, ta có:

1.  $x$  là số có ba chữ số:  $x = 100*a + 10*b + c$ .
2.  $a = 1..9$ ;  $b = a*d$ ;  $0 < c = a*d*d \leq 9$ .

Hệ thức 2 cho phép ta tính giới hạn trên của  $d$ :

$$ad^2 \leq 9$$

$$d \leq \sqrt{9/a}$$

Ta cho  $a$  biến thiên trong khoảng 1..9 rồi cho công bội  $d$  biến thiên trong khoảng từ 1 đến  $3/\sqrt{a}$ . Với mỗi cặp số  $a$  và  $d$  ta tính

$$x = 100*a + 10*a*d + a*d*d = a*(100 + 10*d + d^2)$$

Tuy nhiên, ta có thể nhẩm tính trước cận trên của  $d$  thì sẽ đỡ phải gọi các hàm làm tròn *trunc* và tính căn *sqrt* là những hàm thao tác trên số thực do đó sẽ tốn thời gian.

$a$	1	2	3	4	5	6	7	8	9
Cận trên $d$	3	2	1	1	1	1	1	1	1

Bảng trên cho ta biết,

- ✎ với  $a = 1$  thì  $d$  biến thiên trong khoảng 1..3
- ✎ với  $a = 2$  thì  $d$  biến thiên trong khoảng 2..2
- ✎ với  $a = 3..9$  thì  $d$  biến thiên trong khoảng 1..2
- ✎ Nhắc lại rằng ký pháp  $a..b$  cho ta dãy số nguyên từ  $a$  đến  $b$ .

## Chương trình C++

```
// Cac so tu nhien 3 chu so xyz
// tao thanh cap nhan x, y, z

#include <iostream>

using namespace std;

int d[] = {0,3,2,1,1,1,1,1,1,1};

void BaiGiai() {
    int x, n = 0;
    for(int a = 1; a <= 9; ++a) {
        for(int j = 1; j <= d[a]; ++j) {
            x = a*(100 + 10*j + j*j);
            cout << " " << x;
            ++n;
            if (n % 10 == 0) {
                cout << endl;
            }
        }
    }
    cout << "\n Tong cong " << n << " so.";
}

main() {
    BaiGiai();
    cout << "\n T h e   E n d";
    return 0;
}
```


111 124 139 222 248 333 444 555 666 777  
888 999  
Tong cong 12 so.  
T h e E n d

## Bài 1.4. Mảng ngẫu nhiên


*Sinh ngẫu nhiên  $n$  số nguyên không âm cho mảng nguyên  $a$ .*

### Đặc tả

Để sinh số ngẫu nhiên trong khoảng  $a..b$ , trong C++ ta thực hiện như sau:

 Đặt hai thư viện

```
#include <time.h>
#include <windows.h>
```

 Khởi tạo bộ sinh số ngẫu nhiên

```
srand(time(NULL));
```

Để khởi tạo srand cần một giá trị ban đầu. Ta có thể chọn giá trị này theo đồng hồ máy tính là `time(NULL)`. Lệnh này cho ta tổng thời gian tính từ thời điểm gốc NULL.

✧ Mỗi lần sinh một số ngẫu nhiên nguyên đủ lớn:

```
rand()
```

✧ Hoặc sinh một số ngẫu nhiên nguyên trong khoảng 0..n

```
rand() % n
```

✧ Hoặc sinh một số ngẫu nhiên nguyên trong khoảng a..b

```
x = a + rand() % (b-a+1);
```

Hàm `rand()` sẽ sinh một số ngẫu nhiên đủ lớn. Do đó `rand() % n` sẽ chuyển số đó thành giá trị trong khoảng 0..(n-1). Như vậy,

```
x = a + rand() % (b-a+1) = a + 0..(b-a) = a..(a+b-a) = a..b
```

Trong Python ta thực hiện như sau:

✧ Đặt thư viện

```
import random
```

✧ Gọi

```
random.randint(0,n)
```

để sinh một số ngẫu nhiên nguyên trong khoảng 0..n

✧ hoặc gọi

```
a + random.randint(0,b-a))
```

để sinh một số ngẫu nhiên nguyên trong khoảng a..b

## Chương trình C++

```
// Sinh n so nguyen ngau nhien cho mang a
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

// Hien thi mang nguyen a[0:n] kem chu thich
// moi dong 10 so
void Print(int a[], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
        if ((i+1) % 10 == 0) {
            cout << endl;
        }
    }
}

void BaiGiai(int n) {
```

```

    srand(time(NULL));
    for (int i = 0; i < n; ++i) {
        a[i] = rand() % 1000;
    }
    Print(a, n, "\n a: ");
}

main() {
    BaiGiai(100);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

a:  644 688 485 904 193 59 847 343 745 466
48 962 560 543 246 237 421 696 352 55
146 654 783 668 429 514 866 698 701 473
678 216 294 218 733 412 752 541 551 668
868 471 495 919 778 643 776 806 431 275
767 53 402 759 466 814 361 740 385 303
152 881 332 498 666 39 278 991 682 391
552 509 380 968 337 470 960 761 53 124
519 172 411 654 305 891 107 896 670 751
367 858 675 7 578 95 368 446 119 600

T h e   E n d

```

## Bài 1.5. Chia mảng tỉ lệ 1:1

*Tìm cách chia dãy số nguyên không âm  $a_0, a_1, \dots, a_{n-1}$ ,  $n > 1$  cho trước thành hai đoạn có tổng các phần tử trong mỗi đoạn bằng nhau.*

### Đặc tả

Kí hiệu  $\text{sum}(a[d..c])$  là tổng các phần tử liên tiếp nhau từ  $a[d]$  đến  $a[c]$  của dãy  $a$ :

$$\text{sum}(a[d..c]) = a[d] + a[d+1] + \dots + a[c].$$

Gọi  $t$  là tổng các phần tử của mảng:  $t = \text{sum}(a[0..n-1])$ .

Muốn chia  $a$  thành hai đoạn  $a[0..i]$  và  $a[i+1..n-1]$  có tổng bằng nhau ta phải có:

1.  $t$  là số chẵn ( $t$  chia hết cho 2). Đặt  $t2 = t \div 2$ , và
2. tồn tại một chỉ số  $i$  trong khoảng  $0..n-1$  thỏa điều kiện  $\text{sum}(a[0..i]) = t2$ .

Ta gọi  $i$  là điểm chia. Nếu không tìm được điểm chia ta đặt  $i = -1$ .

### Thuật toán

Hàm Chia cho giá trị  $i$  nếu

$$\text{sum}(a[0..i]) = \text{sum}(a[i+1..n-1]) = t2$$

Trong trường hợp vô nghiệm, ta đặt Chia = -1.

Ta gọi  $i$  là điểm chia và dùng biến  $tr$  (tổng riêng) để tích lũy tổng các phần tử của tiền tố,  $tr = \text{sum}(a[0..i])$ . Khi  $tr = t2$  bài toán có nghiệm  $i$ . Ngược lại, khi  $tr > t2$  bài toán vô nghiệm.

---

```

Thuật toán Chia
Input: a[0..n-1]
Output: i,  $0 \leq i < n$ 
         $\text{sum}(a[0..i]) = \text{sum}(a[i+1..n-1])$ 
        or -1
Begin
    t = sum(a[0..n-1])
    t2 = t div 2
    // t lẻ: vô nghiệm
    if  $2*t2 \neq t$  then return -1 endif
    tr = 0
    for i = 0 to n-1 do
        tr = tr + a[i]
        if tr > t2 return -1 endif
        if tr = t2 return i endif
    endfor
    return -1
End Chia

```

---

Để có dữ liệu test ta cần sinh ngẫu nhiên  $n$  giá trị nguyên không âm cho mảng  $a$ . Ta cũng phân biệt hai trường hợp là *sinh dữ liệu có nghiệm* và *sinh dữ liệu vô nghiệm*.

Để sinh dữ liệu có nghiệm ta chọn ngẫu nhiên một điểm chia  $d$  rồi sinh ngẫu nhiên các giá trị cho tiền tố  $a[0..d-1]$  đồng thời tính tổng  $t2 = \text{sum}(a[0..d-1])$ . Sau đó trong phần hậu tố  $a[d..n-1]$  ta sinh ngẫu nhiên các giá trị sao cho  $\text{sum}(a[d..n-1]) = t2$ .

---

```

Thuật toán Gen
Input: n, d
Output:  $\text{sum}(a[0..d-1]) = \text{sum}(a[d..n-1])$ 
         $0 \leq a[i] \leq m-1$ ,  $a[i]$  là số ngẫu nhiên
Begin
    //sinh ngẫu nhiên a[0..d-1] trong khoảng 0..m-1
    //và tính tổng sum = sum(a[0..d-1])
    sum = 0
    for i = 0 to d-1 do
        a[i] = rand() % m;
        sum += a[i];
    endfor
    // phần hậu tố
    for i = d to n-2 do
        a[i] = random % sum
        sum -= a[i]
    endfor
    a[n-1] = sum // phần tử cuối cùng
End Gen

```

---

Để sinh dữ liệu vô nghiệm trước hết ta sinh dữ liệu có nghiệm rồi chọn ngẫu nhiên một phần tử của a để tăng thêm phần tử đó một đơn vị.

Chương trình sẽ hoạt động theo vòng lặp vô hạn với sơ đồ sau:

---

Thuật toán BaiGiai

---

Input: a[0:n-1]

---

Output:

---

Begin

```
while(true) do // lặp vô hạn
    d = n/3 + rand() % (n/2)
    Gen(n, d)
    if quyết định vô nghiệm then
        chọn ngẫu nhiên phần tử i
        tăng a[i] thêm 1
    endif
    j = Chia(n);
    Thông báo điểm chia j
    Go();
endwhile
End BaiGiai
```

---

Hàm Go hỏi ý kiến người giao tiếp: nếu người giao tiếp muốn dừng chương trình sẽ bấm phím [.] (dot key).

```
// C++
void Go() {
    cout << "\n Băm phím [.] de stop: ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}
```

Dòng lệnh

```
fflush(stdin);
```

có nhiệm vụ lau sạch biến đệm (buffer) stdin để nhận một ký tự mới từ bàn phím.

## Chương trình C++

```
// Chia mang ty le 1:1
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

void Go() {
    cout << "\n Băm phím [.] de stop: ";
```

```

    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Hien thi mang nguyen a[0:n] voi chu thich msg
void Print(int a[], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
    }
}

// Hien thi mang nguyen a[0:d|d:n] voi chu thich msg
// va diem chia d
void Print(int a[], int n, int d, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < d; ++i) {
        cout << " " << a[i];
    }
    cout << " |";
    for (int i = d; i < n; ++i) {
        cout << " " << a[i];
    }
}

// sinh mot so ngau nhien trong khoang a..b
int RandInt(int a, int b) {
    return a + rand() % (b-a+1);
}

// sinh ngau nhien n so nguyen khong am
// diem chia d, khoang ngau nhien m
void Gen(int n, int d) {
    int sum = 0;
    // sinh ngau nhien a[0:d]
    for (int i = 0; i < d; ++i) {
        a[i] = RandInt(1,10);
        sum += a[i];
    }
    // phan con lai a[d:n]
    --n;
    for (int i = d; i < n; ++i) {
        a[i] = RandInt(0, sum-1);
        sum -= a[i];
    }
    a[n-1] = sum;
}

int Chia(int n) {
    int t = 0, t2, tr;
    Print(a, n, "\n Input: ");
    for (int i = 0; i < n; ++i) {
        t += a[i];
    }
}

```



```

    }
    // t = sum(a)
    t2 = t / 2;
    if (2*t2 != t) return -1;
    tr = 0; // tong rieng cua 1 doan
    for (int i = 0; i < n; ++i) {
        tr += a[i];
        if (tr > t2) return -1;
        if (tr == t2) {
            cout << "\n Sum = " << tr;
            Print(a, n, i+1, "\n Result: ");
            return i;
        }
    }
    return -1;
}

void BaiGiai(int n) {
    int j;
    int d;
    srand(time(NULL));
    while(true) {
        // chon ngau nhien diem chia
        d = RandInt(2, n-1);
        Gen(n, d);
        if (RandInt(0,10) == 0) { // 1/10 test la vo nghiem
            ++a[RandInt(0,n-1)];
        }
        j = Chia(n);
        cout << "\n Diem chia: " << j;
        Go();
    }
}

main() {
    int n = 12;
    BaiGiai(n);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Input:  9 4 3 1 4 8 4 9 0 4 8 0
Diem chia: -1
Bam phim [.] de stop:

Input:  0 7 0 6 0 11 0 0 1 0 1 0
Result: 0 7 0 6 | 0 11 0 0 1 0 1 0
Sum = 13
Diem chia: 3
Bam phim [.] de stop:

```

```

Input:  8 4 2 1 6 12 7 0 1 0 1 0
Result: 8 4 2 1 6 | 12 7 0 1 0 1 0
Sum = 21
Diem chia: 4
Bam phim [.] de stop:

Input:  2 2 4 3 3 1 4 1 1 0 1 0
Result: 2 2 4 3 | 3 1 4 1 1 0 1 0
Sum = 11
Diem chia: 3
Bam phim [.] de stop:

```

## Bài 1.6. Chia mảng tỉ lệ 1:k

*Tìm cách chia dãy số nguyên không âm  $a_0, a_1, \dots, a_{n-1}$ ,  $n > 1$  cho trước thành hai đoạn có tổng các phần tử trong một đoạn gấp  $k$  lần tổng các phần tử trong đoạn kia,  $k$  nguyên dương.*

### Đặc tả

Gọi  $t$  là tổng các phần tử của  $a$ ,  $t = \text{sum}(a[0..n-1])$ . Muốn chia  $a$  thành hai đoạn  $a[0..i-1]$  và  $a[i..n-1]$  có tổng gấp nhau  $k$  lần ta phải có:

1.  $t$  chia hết cho  $(k+1)$ . Đặt  $t1 = t \text{ div } (k+1)$  và  $tk = t - t1$ .
2. ( $\exists i, 1 \leq i < n$ ):  $\text{sum}(a[0..i-1]) = t1$  hoặc  $\text{sum}(a[0..i-1]) = tk$ .

Đề ý rằng nếu  $k = 1$  thì  $t1 = tk$ ; nếu  $k > 1$  thì  $t1 < tk$ , do đó bài này là trường hợp riêng của bài trước khi  $k = 1$ .

Trong chương trình dưới đây, hàm Chia( $k$ ) cho giá trị  $i$  nếu mảng  $a$  chia được thành hai đoạn  $a[0..i-1]$  và  $a[i..n-1]$  có tổng gấp  $k$  lần nhau. Trong trường hợp vô nghiệm Chia = -1. Ta gọi  $i$  là điểm chia và dùng biến  $tr$  (tổng riêng) để tích lũy tổng các phần tử của đoạn đang xét  $a[0..i]$ . Khi  $tr = t1$  hoặc  $tr = tk$  thì bài toán có nghiệm  $i$ , ngược lại, thì ta kết luận là bài toán vô nghiệm.

Để sinh dữ liệu kiểm thử, ta có thể dùng thuật toán Gen của bài trước sinh hai đoạn có tổng bằng nhau sau đó tăng các phần tử của một trong hai đoạn lên  $k$  lần.

---

#### Thuật toán Genk

---

Input:  $n, d$

Output:  $\text{sum}(a[0:d]) = k * \text{sum}(a[d:n])$

hoặc

$k * \text{sum}(a[0:d]) = \text{sum}(a[d:n])$

$0 \leq a[i] \leq m-1$ ,  $a[i]$  là số ngẫu nhiên

---

Begin

Gen( $n, d$ )

if random % 2 = 0 then

// tăng tiền tố  $k$  lần

---

---

```

        a[0:d] = k * a[0:d]
    else // tăng hậu tố k lần
        a[d:n] = k * a[d:n]
    End k

```

---

### Chú ý

Bạn cần phân biệt hai kí hiệu:

```

a[d:c] = (a[d], a[d+1], ..., a[c-1])
a[d..c] = (a[d], a[d+1], ..., a[c])

```

### Chương trình C++

```

// Chia mạng ty le 1:k
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

void Go() {
    cout << "\n Bam phim [.] de stop: ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Hien thi mang nguyen a[0:n] voi chu thich msg
void Print(int a[], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
    }
}

// Hien thi mang nguyen a[0:n] voi chu thich msg
void Print(int a[], int n, int d, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < d; ++i) {
        cout << " " << a[i];
    }
    cout << " |";
    for (int i = d; i < n; ++i) {
        cout << " " << a[i];
    }
}

// sinh mot so ngau nhien trong khoang a..b

```

```

int RandInt(int a, int b) {
    return a + rand() % (b-a+1);
}

// sinh ngẫu nhiên n số nguyên không âm
// điểm chia d, khoảng ngẫu nhiên m
void Gen(int n, int d) {
    int sum = 0;
    int m = 10;
    // sinh ngẫu nhiên a[0:d]
    for (int i = 0; i < d; ++i) {
        a[i] = RandInt(0, m-1);
        sum += a[i];
    }
    // phân con lại
    --n;
    for (int i = d; i < n; ++i) {
        a[i] = RandInt(0, sum);
        sum -= a[i];
    }
    a[n-1] = sum;
}

void Genk(int n, int d, int k) {
    Gen(n, d);
    int ii, jj;
    // chọn nửa trái hay phải
    if (RandInt(0,1) == 0) {
        ii = 0; jj = d;
    }
    else {
        ii = d; jj = n;
    }
    for (int i = ii; i < jj; ++i) {
        a[i] *= k;
    }
}

int Chia(int n, int k) {
    int t = 0, t1, tk, tr;
    Print(a, n, "\n Input: ");
    cout << "\n k = " << k;
    for (int i = 0; i < n; ++i) {
        t += a[i];
    }
    t1 = t/(k+1);
    if ((k+1)*t1 != t) return -1;
    tk = t1*k;
    tr = 0;
    for (int i = 0; i < n; ++i) {
        tr += a[i];
        if (tr > t1 && tr > tk) return -1;
        if (tr == t1 || tr == tk) {

```

```

        cout << "\n T1 = " << tr << " tk = " << t-tr;
        Print(a, n, i+1, "\n Result: ");
        return i;
    }
}
return -1;
}

void BaiGiai(int n) {
    int j;
    int d, k;
    srand(time(NULL));
    // chon diem chia d
    while(true) {
        d = RandInt(2, n/2);
        k = RandInt(1,6);
        Genk(n, d, k);
        if (RandInt(0,5) == 0) {
            ++a[RandInt(0, n-1)];
        }
        j = Chia(n, k);
        cout << "\n Diem chia: " << j;
        Go();
    }
}

main() {
    int n = 12;
    BaiGiai(n);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Input:  0 24 3 21 24 24 21 19 6 0 8 0
k = 3
Diem chia: -1
Bam phim [.] de stop:

Input:  7 2 9 5 4 10 28 2 0 0 2 0
k = 2
T1 = 23 tk = 46
Result:  7 2 9 5 | 4 10 28 2 0 0 2 0
Diem chia: 3
Bam phim [.] de stop:

Input:  9 6 5 1 2 0 12 0 1 1 6 0
k = 1
Diem chia: -1
Bam phim [.] de stop:

```

Input: 7 2 0 3 9 0 12 6 0 1 1 0

k = 1

Diem chia: -1

Bam phim [.] de stop:

Input: 0 18 42 48 30 0 18 42 6 28 1 0

k = 6

Diem chia: -1

Bam phim [.] de stop:

Input: 7 8 4 8 0 3 17 4 2 0 1 0

k = 1

T1 = 27 tk = 27

Result: 7 8 4 8 | 0 3 17 4 2 0 1 0

Diem chia: 3

Bam phim [.] de stop:

Input: 16 4 24 20 12 12 0 3 2 0 2 0

k = 4

T1 = 76 tk = 19

Result: 16 4 24 20 12 | 12 0 3 2 0 2 0

Diem chia: 4

Bam phim [.] de stop:

Input: 9 9 6 3 1 6 5 2 7 12 9 0

k = 1

Diem chia: -1

Bam phim [.] de stop:

Input: 6 3 1 8 5 4 3 9 21 11 6 0

k = 1

Diem chia: -1

Bam phim [.] de stop:

Input: 18 18 24 6 18 17 2 4 4 0 1 0

k = 3

T1 = 84 tk = 28

Result: 18 18 24 6 18 | 17 2 4 4 0 1 0

Diem chia: 4

Bam phim [.] de stop:

## CHƯƠNG 2

### SINH DỮ LIỆU VÀO VÀ RA

Hầu hết các bài toán tin đều đòi hỏi *dữ liệu vào và ra*. Người ta thường dùng ba phương thức sinh và nạp dữ liệu sau đây:

1. *Nạp dữ liệu trực tiếp từ bàn phím*. Phương thức này được dùng khi dữ liệu không nhiều.

2. *Sinh dữ liệu nhờ hàm random* (xem chương 1). Phương thức này nhanh chóng và tiện lợi, nếu khéo tổ chức có thể sinh ngẫu nhiên được các dữ liệu đáp ứng được một số điều kiện định trước.

3. *Đọc dữ liệu từ một tệp, thường là tệp văn bản*. Phương thức này khá tiện lợi khi phải chuẩn bị trước những tệp dữ liệu phức tạp.

Kết quả thực hiện chương trình cũng thường được thông báo trực tiếp trên màn hình hoặc ghi vào một tệp văn bản.

#### Bài 2.1. Sinh ngẫu nhiên theo khoảng

*Sinh ngẫu nhiên cho mảng nguyên  $a$   $n$  phần tử trong khoảng  $-M..M$ ;  $M > 0$ .*

##### Đặc tả

Hàm `rand()` trong C++ sinh một số ngẫu nhiên nguyên không âm lớn. Do đó lời gọi `rand() % M` sẽ cho ta một số nguyên trong khoảng  $0..M-1$ .

Trong Python, hàm `randint(a, b)` sẽ cho ta một số ngẫu nhiên trong khoảng  $a..b$  (kể cả hai giá trị đầu  $a$  và cuối  $b$ ).

Tóm lại, ta có

```
C++: x = -M + rand() % (2M + 1);
```

```
Python: random.randint(-M, M)
```

#### Chương trình C++

```
// Sinh ngẫu nhiên n phần tử -M:M+1
// cho mảng nguyên a
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

// Hiện thị mảng nguyên a[0:n] với chuỗi thích msg
void Print(int a[], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
```

```

        cout << " " << a[i];
    }
}

void Gen(int n, int M) {
    srand(time(NULL));
    int M2 = M + M + 1;
    for (int i = 0; i < n; ++i) {
        a[i] = -M + rand() % M2;
    }
}

main() {
    int n = 12;
    Gen(n, 10);
    Print(a, n, "\n Result: ");
    cout << "\n T h e    E n d";
    return 0;
}

```

## Output

```

Result:  -5 1 -4 -6 3 -1 -5 9 11 10 -10 6
T h e    E n d

```

## Bài 2.2. Sinh ngẫu nhiên tăng

*Sinh ngẫu nhiên  $n$  phần tử được sắp không giảm cho mảng nguyên  $a$ .*

### Thuật toán

1. Sinh ngẫu nhiên phần tử đầu tiên:  $a[0]$
2. Từ phần tử thứ hai trở đi, trị được sinh bằng trị của phần tử sát trước nó cộng thêm một đại lượng ngẫu nhiên.

### Chương trình C++

```

// Sinh ngau nhien n phan tu sap khong giam
// ghi mang nguyen a
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

// Hien thi mang nguyen a[0:n] voi chu thich msg
void Print(int a[], int n, const char * msg = "") {
    cout << msg;
}

```



```

        for (int i = 0; i < n; ++i) {
            cout << " " << a[i];
        }
    }

    void Gen(int n, int d) {
        srand(time(NULL));
        a[0] = rand() % d;
        for (int i = 1; i < n; ++i) {
            a[i] = a[i-1] + rand() % d;
        }
    }

    main() {
        int n = 12;
        Gen(n,5);
        Print(a, n, "\n Result: ");
        cout << "\n T h e    E n d";
        return 0;
    }

```

## Output

```

Result:  2 6 10 14 18 19 19 19 19 20 24 25
T h e    E n d

```

## Bài 2.3. Sinh hoán vị ngẫu nhiên

*Sinh ngẫu nhiên cho mảng nguyên a một hoán vị của 1..n.*

### Đặc tả

Trước hết sinh một hoán vị đơn vị 1, 2..n cho mảng. Sau đó đảo ngẫu nhiên các phần tử của mảng.

### Chương trình C++

```

// Sinh ngau nhien hoan vi 1...n
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

// Hien thi mang nguyen a[0:n] voi chu thich msg
void Print(int a[], int n, const char * msg = "") {
    cout << msg;

```

```

    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
    }
}

// doi cho a[i] va a[j]
void Swap(int i, int j) {
    int x = a[i];
    a[i] = a[j]; a[j] = x;
}

void Gen(int n) {
    srand(time(NULL));
    for (int i = 0; i < n; ++i) {
        a[i] = i+1;
    }
    for (int i = 0; i < n; ++i) {
        Swap(i, rand() % n);
    }
}

main() {
    int n = 12;
    Gen(n);
    Print(a, n, "\n Result: ");
    cout << "\n T h e    E n d";
    return 0;
}

```

## Output

```

Result:  5 12 7 9 4 11 8 6 2 1 3 10
T h e    E n d

```

## Bài 2.4. Sinh ngẫu nhiên đều

*Sinh ngẫu nhiên  $n$  phần tử cho mảng nguyên  $a$  thoả điều kiện:  $n$  phần tử tạo thành  $k$  đoạn liên tiếp có tổng các phần tử trong mỗi đoạn bằng nhau và bằng giá trị  $t$  cho trước.*

### Thuật toán

1. Chọn số lượng các phần tử trong mỗi đoạn là  $\text{random}(n \text{ div } k) + 1$ , khi đó số lượng các phần tử được phát sinh ngẫu nhiên sẽ không vượt quá

$$k * (n \text{ div } k) \leq n$$

Sau đó ta sẽ chỉnh sao cho số lượng các phần tử đúng bằng  $n$ .

2. Giả sử  $a[d..c]$  là đoạn thứ  $j$  cần được sinh ngẫu nhiên sao cho

$$a[d] + a[d + 1] + \dots + a[c] = t$$

Ta sinh đoạn này như sau:

2.1. Gọi  $tr$  là giá trị còn lại của tổng. Lúc đầu ta có  $tr = t$ .

2.2. Gán trị ngẫu nhiên  $0..tr-1$  cho các phần tử đầu đoạn  $a$ , trừ lại phần tử cuối  $a[c]$ , đồng thời chỉnh giá trị còn lại của  $tr$ :

```
tr = t
for each i = d:c do
    a[i] = random(tr)
    tr = tr - a[i]
end for
```

2.4. Đặt giá trị còn lại của tổng riêng vào phần tử cuối đoạn:

```
a[c] = tr
```

## Chương trình C++

```
// Sinh ngau nhien deu
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

// Hien thi mang nguyen a[d..c] voi chu thich msg
void Print(int a[], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << a[i];
    }
}

// sinh ngau nhien doan d..c tong t
void GenDoan(int d, int c, int t) {
    for (int i = d; i < c; ++i) {
        a[i] = rand() % t;
        t -= a[i];
    }
    a[c] = t;
}

void Gen(int n, int k, int t) {
    srand(time(NULL));
    int m = n / k; // k doan, moi doan m phan tu
    int r = n - k*m; // phan du
    cout << n << " phan tu, " << k << " doan " << ", tong = " << t;
    cout << "\n Moi doan dau co " << m << " pt. Doan cuoi co "
        << (m+r) << " pt.";
    int d = 0, c = m-1;
    for (int i = 1; i < k; ++i) {
        GenDoan(d, c, t);
        Print(a, d, c, "\n");
        d = c + 1; c = d + m - 1;
    }
}
```

```

    }
    // doan cuoi cung k co them r phan tu
    c = n-1;
    GenDoan(d, c, t);
    Print(a, d, c, "\n");
}

main() {
    int n = 31, k = 4, t = 20;
    Gen(n, k, t);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

31 phan tu, 4 doan , tong = 20
Moi doan co 7 pt. Doan cuoi cung co 10 pt.
13 2 1 3 0 0 1
6 4 4 4 1 0 1
8 8 2 0 1 0 1
6 4 1 0 2 3 0 1 0 3
T h e   E n d



```

## Bài 2.5. Sinh ngẫu nhiên tỉ lệ

*Sinh ngẫu nhiên cho mảng nguyên  $a$  có  $n$  phần tử tạo thành hai đoạn liên tiếp có tổng các phần tử trong một đoạn gấp  $k$  lần tổng các phần tử của đoạn kia.*

### Thuật toán

Xem bài 1.6

-  Chọn ngẫu nhiên điểm chia  $d$ , sinh ngẫu nhiên hai đoạn  $a[0:d]$  và  $a[d:n]$  có tổng bằng nhau:  $\text{Gen}(n,d)$ .
-  Chọn ngẫu nhiên nửa trái hoặc phải để tăng các phần tử của nửa đó lên  $k$  lần.

### Chương trình C++

```

// Sinh ngau nhien ty le 1:k
#include <iostream>
#include <time.h>
#include <windows.h>

using namespace std;

const int MN = 1000000;
int a[MN];

void Go() {

```

```

    cout << "\n Bam phim [.] de stop: ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Hien thi mang nguyen a[0:n] voi chu thich msg
void Print(int a[], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
    }
}

// Hien thi mang nguyen a[0:d|d:n] voi chu thich msg
// diem chia d
void Print(int a[], int n, int d, const char * msg = "") {
    cout << msg;
    int t = 0;
    cout << "\n Doan thu nhât: ";
    for (int i = 0; i < d; ++i) {
        cout << " " << a[i];
        t += a[i];
    }
    cout << "\n Tong = " << t;
    t = 0;
    cout << "\n Doan thu hai: ";
    for (int i = d; i < n; ++i) {
        cout << " " << a[i];
        t += a[i];
    }
    cout << "\n Tong = " << t;
}

// sinh ngau nhien n so nguyen khong am
// diem chia d, khoang ngau nhien m
// sum(a[0:d]) = sum(a[d:n])
void Gen(int n, int d) {
    int sum = 0;
    int m = 10;
    // sinh ngau nhien a[0:d]
    for (int i = 0; i < d; ++i) {
        a[i] = rand() % m;
        sum += a[i];
    }
    // phan con lai
    --n;
    m = sum;
    for (int i = d; i < n-1; ++i) {
        a[i] = rand() % m;
        m -= a[i];
    }
    a[n-1] = m;
}

```

```

void Genk(int n, int d, int k) {
    cout << "\n Ti le k = " << k;
    Gen(n, d); // sum(a[0:d]) = sum(a[d:n])
    int ii, jj;
    if (rand() % 2 == 0) {
        ii = 0; jj = d; // lay doan dau
    }
    else {
        ii = d; jj = n; // lay doan sau
    }
    // k*sum(a[ii:jj])
    for (int i = ii; i < jj; ++i) {
        a[i] *= k;
    }
}

void BaiGiai(int n) {
    int d, k;
    srand(time(NULL));
    while(true) {
        // chon diem chia
        d = n/4 + rand() % (n/2) + 1;
        // chon ti le
        k = rand() % 6 + 1;
        Genk(n, d, k);
        Print(a, n, d, "\n Ket qua Gen ");
        Go();
    }
}

main() {
    int n = 12;
    BaiGiai(n);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Ti le k = 6
Ket qua Gen
Doan thu nhat:  4 4 8 2 7
Tong = 25
Doan thu hai:  36 12 0 30 48 24 0
Tong = 150
Bam phim [.] de stop:

Ti le k = 2
Ket qua Gen
Doan thu nhat:  18 12 2 12 0 2 14 4 0
Tong = 64

```

```
Doan thu hai: 18 14 0
Tong = 32
Bam phim [.] de stop:
```

## Bài 2.6. Sinh ngẫu nhiên tệp tăng

*Sinh ngẫu nhiên n số tự nhiên sắp tăng và ghi vào một tệp văn bản có tên cho trước.*

### Thuật toán

Xem bài 2.2

### Chương trình C++

```
// Sinh ngau nhien n phan tu sap khong giam de ghi file
#include <iostream>
#include <fstream>
#include <time.h>
#include <windows.h>

using namespace std;

const char * fn = "NUM.DAT";

void Gen(int n, const char * fn) {
    srand(time(NULL));
    ofstream f(fn);
    int d = 5;
    int x = rand() % d;
    f << x << endl;
    for (int i = 1; i < n; ++i) {
        x += rand() % d;
        f << x << endl;
    }
    f.close();
}

main() {
    Gen(20,fn);
    cout << "\n T h e    E n d";
    return 0;
}
```

### Chú thích

Dòng lệnh

```
ofstream f(fn);
```

mở file tên fn để ghi

Dòng lệnh

```
f << x << endl;
```

Ghi giá trị x vào file f rồi xuống dòng mới.

Dòng lệnh

```
f.close();
```

đóng file f sau khi hoàn tất ghi.

## Bài 2.7. Sinh ngẫu nhiên tệp cấp số cộng

*Sinh ngẫu nhiên một cấp số cộng gồm n số hạng và ghi vào một tệp văn bản có tên cho trước.*

### Thuật toán

Cấp số cộng có dạng

```
a, a+d, a+2d, ..., a +kd, ...  
d là công sai
```

Cấp số cộng có thể tăng chặt (công sai  $d > 0$ ) hoặc giảm chặt (công sai  $d < 0$ ) hoặc không đổi (công sai  $d = 0$ ).

- ✎ Sinh ngẫu nhiên số hạng thứ nhất x và công sai d, ghi x vào file.
- ✎ Ghi file các phần tử  $x+di$ ,  $i = 2..n$ .
- ✎ Đóng file.

Độ phức tạp: n.

### Chương trình C++

```
// Sinh ngẫu nhiên cấp cộng n phần tử để ghi file  
#include <iostream>  
#include <fstream>  
#include <time.h>  
#include <windows.h>  
  
using namespace std;  
  
const char * fn = "CAPCONG.INP";  
  
void Gen(int n, const char * fn) {
```



```
    srand(time(NULL));
    ofstream f(fn);
    int d = rand() % 5 + 1;
    int x = rand() % 10 + 1;
    f << x << endl;
    for (int i = 1; i < n; ++i) {
        x += d;
        f << x << endl;
    }
    f.close();
}

int Read(const char * fn) {
    ifstream f(fn);
    int x, m = 0;
    while (true) {
        if (f.eof()) break;
        f >> x;
        if (f.eof()) break;
        ++m;
        cout << m << ": " << x << endl;
    }
    f.close();
    return m;
}

main() {
    int n = 20;
    Gen(n,fn);
    int m = Read(fn);
    cout << "\n Tong cong: " << m << " phan tu.";
    cout << "\n T h e    E n d";
    return 0;
}
```

## Output

```
1: 1
2: 5
3: 9
4: 13
5: 17
6: 21
7: 25
8: 29
9: 33
10: 37
11: 41
12: 45
13: 49
14: 53
```

```

15: 57
16: 61
17: 65
18: 69
19: 73
20: 77

Tong cong: 20 phan tu.
T h e   E n d

```

### Chú thích

Khi ghi file ta đã ghi dấu xuống dòng cho mỗi số  $x$ , do đó khi đọc lại file ta phải kiểm tra kết thúc file hai lần, trước và sau khi đọc mỗi số  $x$ :

```

if (f.eof()) break;
f >> x;
if (f.eof()) break;

```

## Bài 2.8. Sinh ngẫu nhiên mảng đối xứng

*Sinh ngẫu nhiên các giá trị để ghi vào một mảng hai chiều  $a[0:n][0:n]$  sao cho các phần tử đối xứng nhau qua đường chéo chính*

### Thuật toán

1. Sinh ngẫu nhiên các phần tử *trên đường chéo chính*  
 $a[i, i] = \text{random}, i = 1:n.$
2. Sinh ngẫu nhiên các phần tử nằm *dưới đường chéo chính*  
 rồi lấy đối xứng  
 $a[i, j] = a[j][i] = \text{random}, i = 0:n, j = 0:i$   
 Độ phức tạp:  $n^2$ .

$a$	$x'$	$y'$	$v'$	$m'$
$x$	$b$	$z'$	$u'$	$n'$
$y$	$z$	$c$	$t'$	$p'$
$v$	$u$	$t$	$d$	$q'$
$m$	$n$	$p$	$q$	$e$

### Chương trình C++

```

// Sinh ngau nhien mang a[][] doi xung
#include <iostream>
#include <fstream>

```

```

#include <time.h>
#include <windows.h>

using namespace std;
const int MN = 500;
int a[MN][MN];

// print a line b[0:n]
void Print(int b[], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << b[i];
    }
}

// print an array a[][MN]
void Print(int a[][MN], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        Print(a[i], n, "\n");
    }
}

void Gen(int n, int d) {
    srand(time(NULL));
    for (int i = 0; i < n; ++i) {
        a[i][i] = rand() % d;
        for (int j = 0; j < i; ++j) {
            a[i][j] = a[j][i] = rand() % d;
        }
    }
}

main() {
    int n = 5, d = 30;
    Gen(n, d);
    Print(a, n, "\n Result:");
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Result:
13 0 22 7 4
0 2 1 2 29
22 1 15 20 26
7 2 20 5 8
4 29 26 8 9
T h e   E n d

```

## Bài 2.9. Số độ cao h

*Độ cao của một số tự nhiên là tổng các chữ số của số đó. Bạn cần tìm toàn bộ các số tự nhiên có tối đa ba chữ số và có độ cao h cho trước. Ghi kết quả vào một tệp văn bản có tên cho trước.*

### Thuật toán

Một tiếp cận tự nhiên là duyệt các số dưới 1000 để kiểm tra tổng các chữ số của số  $x$  bằng  $h$  thì ghi kết quả. Gọi  $H(x)$  là hàm cho ra độ cao của số  $x$ , ta có:

```
for x in 100..999 do
  if H(x) == h then
    Ghi(x)
  end if
end for
```

Theo tiếp cận thứ hai ta sinh ra vừa đủ các số đáp ứng điều kiện của đề bài như sau:

Bài toán này có cách phát biểu khác và tổng quát như sau: *có  $n$  cốc nước dung tích 9 thìa mỗi cốc. Cho một bình đựng  $h$  thìa nước. Hãy xác định mọi phương án chia nước vào các cốc.*

Ta xét lời giải với  $n = 3$ . Ta có  $h_{min} = 0+0+0 = 0$ ,  $h_{max} = 9+9+9 = 27$ .

1. Các số cần tìm  $y$  có dạng  $y = abc$ ,  $a + b + c = h$  và  $a$  biến thiên từ  $mina$  đến  $maxa$ , trong đó  $mina$  là lượng nước ít nhất trong cốc đầu tiên  $a$ ,  $maxa$  là lượng nước lớn nhất trong cốc  $a$ . Nếu đổ đầy hai cốc  $b$  và  $c$ , mỗi cốc 9 thìa nước thì lượng nước còn lại sẽ là tối thiểu cho cốc  $a$ . Ngược lại, nếu tổng cộng chỉ có  $h < 9$  thìa nước thì lượng nước tối đa trong cốc  $a$  phải là  $h$ . Ta có

```
if h ≤ 18 then mina = 0 else mina = h-18;
if h ≥ 9 then maxa = 9 else maxa = h;
```

2. Với mỗi  $a = mina..maxa$  ta tính

2.1.  $bc = h - a$  (biến  $bc$  chứa tổng các chữ số  $b$  và  $c$ ).

2.2. Giải bài toán nhỏ với  $n = 2$ .

```
if bc ≤ 9 then minb = 0 else minb = bc-9;
```

```
if bc ≥ 9 then maxb = 9 else maxb = bc;
```

2.3. Với mỗi  $b = minb..maxb$  ta tính

```
y = 100*a + 10*b + (bc - b).
```

Ghi số này vào file.

---

Thuật toán SoDoCao

---

Input:  $h \in 0..27$

Output: Các số  $x$  dạng  $abc$ ,  $a+b+c = h$

---

Begin

```
mina = (h <= 18) ? 0 : h-18
```

```
maxa = (h >= 9) ? 9 : h
```

```
for a = mina..maxa do
```

---

---

```

        bc = h-a
        minb = (bc <= 9) ? 0: bc-9
        maxb = (bc >= 9) ? 9 : bc
        for b = minb..maxb do
            x = 100*a + 10*b + (bc-b)
            add x to file
        endfor b
    endfor a
End SoDoCao

```

---

## Chương trình C++

```

// So do cao h
#include <iostream>
#include <fstream>

using namespace std;

// Do cao cua x
int H(int x) {
    int d = 0;
    while(x != 0) {
        d += x % 10;
        x /= 10;
    }
    return d;
}

// Tiep can 1: thuat giai tu nhien
int Find(int h) {
    int c = 0; // dem ket qua
    for(int x = 1; x < 1000; ++x) {
        if (H(x) == h) {
            cout << "\n " << x;
            ++c;
        }
    }
    cout << "\n Tong cong " << c << " so.";
    return c;
}

// Tiep can 2: sinh vua du
void BaiGiai(int h, const char * fn) {
    ofstream f(fn);
    int bc, minb, maxb;

```

```

int x, d = 0;
int mina = (h <= 18) ? 0 : h-18;
int maxa = (h >= 9) ? 9 : h;
for (int a = mina; a <= maxa; ++a) {
    bc = h-a;
    minb = (bc <= 9) ? 0 : bc-9;
    maxb = (bc >= 9) ? 9 : bc;
    for (int b = minb; b <= maxb; ++b) {
        d++;
        x = 100*a + 10*b + (bc-b);
        f << x << endl;
    }
}
f.close();
// cout << "\n Tong cong " << d << " so.";
}

main() {
    int h = 10;
    Find(h);
    BaiGiai(h, "HNUM.DAT");
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

19
28
37
46
55
64
73
82
91
109
118
127
136
145
154
163
172
181
190
208
217
226
235
244
253

```

```
262
271
280
307
316
325
334
343
352
361
370
406
415
424
433
442
451
460
505
514
523
532
541
550
604
613
622
631
640
703
712
721
730
802
811
820
901
910
Tong cong 63 so
T h e   E n d
```

## Bình luận

1. Có thể giải bài toán trên bằng phương pháp vét cạn dùng ba vòng for như sau:

```
for a = 1..9 do
  for b = 0..9 do
    for c = 0..9 do
      if a+b+c = h then
        write(f,a,b,c,#32);
    . . .
```

2. Phương pháp vét cạn đòi hỏi  $10 \times 10 \times 10 = 1000$  lần duyệt trong khi với  $h = 10$  chỉ có 63 số thoả mãn điều kiện của đầu bài. Dùng phương pháp sinh ta nhận được đúng 63 số cần tìm, không phải duyệt thừa số nào.

## Bài 2.10. Tập các hoán vị

Với mỗi số  $n$  cho trước trong khoảng  $1..9$ , ghi vào một tệp văn bản có tên cho trước toàn bộ các hoán vị của  $1..n$ . Hoán vị được sắp xếp tăng theo thứ tự từ điển, thí dụ  $78921345 < 78921354$ .

### Thuật toán

1. Khởi tạo và ghi hoán vị *nhỏ nhất* là hoán vị đơn vị  $h[1..n] = 1..n$   
 2. Giả sử ta đã ghi được hoán vị  $h$  vào tệp. Hoán vị tiếp theo sẽ được tạo từ  $h$  thông qua hàm `Next` như sau:

2.1 **Tìm điểm gãy:** Tìm ngược từ  $h[n-1]$  trở về trước đến vị trí  $i$  đầu tiên thoả điều kiện  $h[i] < h[i+1]$ .

- Nếu không tìm được  $i$  tức là  $s$  là hoán vị lớn nhất,  $h[1..n] = [9, 8, 7, 6, 5, 4, 3, 2, 1]$ . Đặt trị `false` cho hàm `Next` và dừng thuật toán. `Next = false` có nghĩa là không tồn tại hoán vị sát sau hoán vị  $h$  hay  $h$  là hoán vị lớn nhất.

- Nếu tìm được: thực hiện bước 2.2.

2.2 **Tìm điểm vượt:** Tìm ngược từ  $h[n-1]$  trở về trước đến vị trí  $j$  đầu tiên thoả điều kiện  $h[j] > h[i]$ .

2.3. **Đổi chỗ**  $h[i]$  với  $h[j]$ .

2.4. **Lật:** Đảo lại trật tự của dãy  $h[i+1..n-1]$  ta sẽ thu được hoán vị sát sau hoán vị  $h$ .

3. Đặt trị `true` cho hàm `Next`. `Next = true` có nghĩa là tìm được hoán vị sát sau hoán vị  $h$ .

### Ví dụ

Với  $n = 8$ , giả sử ta đã ghi được hoán vị  $h = 342865971$ , khi đó hoán vị sát sau  $h$  sẽ được xây dựng như sau:

chỉ số	0	1	2	3	4	5	6	7	8
$h$	3	4	2	8	6	5	9	7	1
điểm gãy						i			
điểm vượt								j	
đổi chỗ $h[i] \leftrightarrow h[j]$	3	4	2	8	6	7	9	5	1
lật đoạn $h[i+1..n-1]$	3	4	2	8	6	7	<u>1</u>	<u>5</u>	<u>9</u>

Ta thu được hoán vị sát sau của hoán vị  $h = [3, 4, 2, 8, 6, 5, 9, 7, 1]$  là `Next(h) = [3, 4, 2, 8, 6, 7, 1, 5, 9]`.



## Chương trình C++

```
// Cac hoan vi
#include <iostream>
#include <fstream>

using namespace std;

const char * filename = "HOANVI.DAT";
string h;

void Swap(int i, int j) {
    int t = h[i];
    h[i] = h[j];
    h[j] = t;
}

bool Next(int n) {
    int i, j;
    for (i = n-2; i >= 0; --i) {
        if (h[i] < h[i+1])
            break;
    }
    if (i < 0) // h la hoan vi max
        return false;
    for (j = n-1; j > i; --j) {
        if (h[j] > h[i])
            break;
    }
    Swap(i,j);
    ++i; j = n-1;
    while (i < j) {
        Swap(i++,j--);
    }
    return true;
}

void BaiGiai(int n) {
    ofstream f(filename);
    // Khoi tao hoan vi don vi 1..n
    h = "";
    for (int c = 1; c <= n; ++c)
        h += '0' + c;
    int d = 0;
    do {
        f << h << endl; // ghi dong h vao file f
        ++d;
    } while (Next(n));
    f.close();
    cout << "\n Da ghi " << d << " hoan vi.";
}

main() {
```

```
BaiGiai(9);
cout << "\n T h e   E n d";
return 0;
}
```

## Output

```
123456789
123456798
123456879
123456897
. . .
123458697
123458769
123458796
123458967
. . .
Đã ghi 362880 hoán vị.
```

## Chú thích

Có cả thấy  $n!$  hoán vị.

## Bài 2.11. Đọc dữ liệu từ tệp vào mảng hai kích thước

*Đọc dữ liệu kiểu nguyên từ một tệp văn bản vào một mảng hai chiều.*

*Tệp có cấu trúc như sau:*

- Hai số đầu tiên  $n, m$  là kích thước của mảng gồm  $n$  dòng và  $m$  cột.
- Tiếp đến là các dữ liệu ghi liên tiếp nhau theo từng dòng của mảng.
- Các số cách nhau ít nhất một dấu cách.

Ví dụ

DATA.INP			cột	cột	cột	
			0	1	2	
2	3	$n = 2$ dòng, $m = 3$ cột	dòng 0	-1	4	5
-1	4 5		dòng 1	3	7	1
3	7 1					

*Mảng a*

## Thuật toán

- ✈ Mở file f tên "DATA.INP"
- ✈ Đọc hai giá trị  $n$  (số dòng) và  $m$  (số cột)
- ✈ Đọc  $n$  dòng
- ✈ Mỗi dòng đọc  $m$  giá trị

Trong C++, bạn có thể sử dụng kiểu dữ liệu vector thay cho mảng vì tính cơ động cao. Vài hàm đầu tiên dùng cho vector là nh

sau:

```
a.clear(); // xóa nội dung của vector a
a.push_back(x); // nạp trị x vào cuối vector a
a[i]; // truy cập phần tử a[i] của vector a
a.empty(); // true nếu vector a rỗng.
a.size(); // số phần tử trong vector a
```

## Chương trình C++

```
// Doc du lieu tu file vao mang a[][]
#include <iostream>
#include <fstream>
#include <bits/stdc++.h>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

const int MN = 100; // kích thước tối đa của mảng a
const char * fn = "Data.inp"; // file name
VVI a;
int n, m; // n dòng, m cột

// print a vector<int> b
void Print(VI b, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < b.size(); ++i) {
        cout << " " << b[i];
    }
}

// print a vector<VI> a
void Print(VVI a, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < a.size(); ++i) {
        Print(a[i], "\n");
    }
}

void ReadFile() {
    ifstream f("DATA.INP");
    f >> n >> m;
    a.clear();
    VI b;
    int x;
    for (int i = 0; i < n; ++i) {
        b.clear();
        for (int j = 0; j < m; ++j) {
            f >> x;
        }
    }
}
```

```

        b.push_back(x);
    }
    a.push_back(b);
}
f.close();
}

void BaiGiai() {
    ReadFile();
    Print(a, "\n a: ");
}

main() {
    BaiGiai();
    cout << "\n T h e   E n d";
    return 0;
}

main() {
    BaiGiai(fn);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

a:
-1 4 5
3 7 1
T h e   E n d

```

## Bài 2.12. Đọc dữ liệu từ tệp vào mảng `[][]` biết một kích thước

Đọc dữ liệu kiểu nguyên từ một tệp văn bản vào một mảng hai chiều  $a[0:n][0:m]$  cho biết một kích thước  $m$  (số cột).

Tệp có cấu trúc như sau:

- Số đầu tiên ghi số lượng cột  $m$  của mảng tức là số phần tử trên một dòng.
- Tiếp đến là các dữ liệu ghi liên tiếp nhau theo từng dòng của mảng.
- Các số cách nhau ít nhất một dấu cách.

## Ví dụ

DATA.INP

	cột	cột	cột			
	0	1	2			
3	$m = 3$ cột	dòng 0	<table border="1"> <tr> <td>-1</td> <td>4</td> <td>5</td> </tr> </table>	-1	4	5
-1	4	5				

-1 4 5

dòng 1

3	7	1
---	---	---

3 7 1

Mảng a

## Thuật toán

- 🐦 Mở file f tên "DATA.INP"
- 🐦 Đọc giá trị m (số cột)
- 🐦  $n = 0$
- 🐦 Mỗi lần đọc xong một dòng gồm m giá trị ta tăng biến đếm dòng ( $n$ ) thêm 1. Khi gặp dòng trống ta coi là hết dữ liệu số.

Hàm GetInt(string s, int &k) đọc và xuất một số nguyên từ vị trí k của string s. Sau khi đọc xong một số, biến k sẽ trở đến vị trí sát sau số đã đọc.

```
// doc mot so nguyen tu vi tri k cua string s
int GetInt(string s, int &k) {
    const char BL = 32; // dấu cách
    int v = 0;
    int dau = 1;
    for (; k < s.length(); ++k) {
        if (s[k] != BL) break; // bỏ qua các dấu cách
    }
    if (s[k] == '-' || s[k] == '+') { // Gặp dấu + hoặc -
        if (s[k] == '-') dau = -1;
        ++k;
    }
    for (; k < s.length(); ++k) {
        if (s[k] >= '0' && s[k] <= '9') {
            // đọc và gom từng chữ số
            v = v*10 + (s[k] - '0');
        }
        else break;
    }
    return dau*v;
}
```

## Ví dụ

k	0	1	2	3	4	5	6	7	8
s		-	1	2		3	5	9	

```
k = 0;
m = GetInt(s,k); // m = -12; k = 4
m = GetInt(s,k); // m = 359; k = 8
```

## Chương trình C++

```
// Doc du lieu tu file vao mang a[][]
#include <iostream>
#include <fstream>
```

```

#include <bits/stdc++.h>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

const int MN = 100; // kích thước tối đa của mảng a
const char * fn = "Data.inp"; // file name
VVI a;
int n, m; // n dòng, m cột

// print a vector<int> b
void Print(VI b, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < b.size(); ++i) {
        cout << " " << b[i];
    }
}

// print a vector<VI> a
void Print(VVI a, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < a.size(); ++i) {
        Print(a[i], "\n");
    }
}

// đọc một số nguyên từ vị trí k của string s
int GetInt(string s, int &k) {
    const char BL = 32;
    int v = 0;
    int dau = 1;
    for (; k < s.length(); ++k) {
        if (s[k] != BL) break;
    }
    if (s[k] == '-' || s[k] == '+') {
        if (s[k] == '-') dau = -1;
        ++k;
    }
    for (; k < s.length(); ++k) {
        if (s[k] >= '0' && s[k] <= '9') {
            v = v*10 + (s[k] - '0');
        }
        else break;
    }
    return dau*v;
}

void ReadFile(const char * fn) {
    ifstream f("DATA.INP");
    int k = 0;
    a.clear();

```

```

string s;
getline(f,s);
m = GetInt(s,k);
cout << "\n so cot = " << m;
VI b;
while (true) {
    getline(f,s);
    if (s == "") break;
    k = 0;
    b.clear();
    for (int j = 0; j < m; ++j) {
        b.push_back(GetInt(s,k));
    }
    a.push_back(b);
}
f.close();
}

void BaiGiai() {
    ReadFile(fn);
    Print(a, "\n a: ");
}

main() {
    BaiGiai();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

so cot = 3
a:
-1 4 5
3 7 1
T h e   E n d

```

## Bài 2.13. Đọc dữ liệu từ tệp vào mảng đối xứng

*Đọc dữ liệu kiểu nguyên từ một tệp văn bản có tên fn vào một mảng hai chiều đối xứng.*

Tệp có cấu trúc như sau:

- Số đầu tiên ghi số lượng cột (và đồng thời là số lượng dòng) của mảng.
- Tiếp đến là các dữ liệu ghi liên tiếp nhau theo nửa tam giác trên tính từ đường chéo chính, kể cả các số liệu trên đường chéo chính.
- Các số cùng dòng cách nhau ít nhất một dấu cách.

## Ví dụ

DATA.INP

a

a





```

        f >> a[i][i];
        for (int j = i + 1; j < n; ++j) {
            f >> a[i][j];
            a[j][i] = a[i][j];
        }
    }
    f.close();
}

void BaiGiai(const char * fn) {
    ReadFile(fn);
    Print(a, n, "\n a: ");
}

main() {
    BaiGiai(fn);
    cout << "\n T h e    E n d";
    return 0;
}

```

## Output

```

n = 3
a:
1 2 3
2 4 6
3 6 8
T h e    E n d

```

## Bài 2.14. Đếm tàu

Một tệp văn bản có tên *fn* ghi sơ đồ một vùng biển hình chữ nhật chiều ngang dài tối đa 250 ký tự, chiều dọc (số dòng) không hạn chế. Trên biển có các con tàu hình chữ nhật chứa các ký tự 1, vùng nước được biểu thị qua các ký tự 0. Biết rằng các con tàu không dính nhau và các dòng thể hiện vùng biển đều dài bằng nhau. Hãy đếm số lượng tàu. Kết quả hiển thị trên màn hình.

### Ví dụ

SHIPS.INP

111100111

000000111

110000000

110011001

5 tàu

## Thuật toán

Vì các tàu không dính nhau nên ta phân biệt các tàu qua mũi tàu, tức là góc A - góc Tây-Bắc hay là góc trên-trái của tàu. Ta có,

$$\text{số lượng tàu} = \text{số lượng mũi tàu}$$

Mũi tàu là điểm nhận giá trị 1 và nếu bước một bước sang trái hoặc lên trên sẽ lên bờ hoặc rơi xuống biển.

A	00	<u>11111</u>	B
	00	<u>11111</u>	
D	0000000		C

Tàu ABCD

Trên các hình, mũi tàu nhận giá trị 1 và được gạch dưới.

Sau khi mở tệp ta đọc và xử lý từng dòng văn bản  $y$  và so sánh nó với dòng  $x$  đã xử lý trước đó. Nếu  $y$  là dòng đầu tiên, tức là dòng nằm sát bờ Bắc, ta khởi trị cho  $x$  với  $n$  ký tự 0 tức là ta loại trừ trường hợp bước lên bờ Bắc. Khi xử lý  $y$ , ta chú ý tách riêng trường hợp tàu nằm sát bờ Tây, tức là xét riêng  $y[0]$ . Sau mỗi lần xử lý dòng  $y$  ta copy dòng  $y$  sang  $x$  và luôn giữ cho  $x$  có chiều dài tối đa  $m$  ký tự như yêu cầu của đầu bài.

## Chương trình C++

```
// Dem tau
#include <iostream>
#include <fstream>

using namespace std;

const char BOONG = '1'; // diem tren boong tau
const char NUOC = '0';

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

void DemTau(const char * fn) {
    int sotau = 0;
    string x, y;
    ifstream f(fn);
    // Khoi tri x toan 0
    x = "";
    x.append(251, '0');
```

```
// x: dòng trên, y: dòng dưới
while (true) {
    getline(f,y);
    if (y == "") break;
    if (y[0] == BOONG && x[0] == NUOC) ++sotau;
    for (int i = 1; i < y.length(); ++i) {
        if (y[i] == BOONG) {
            if (y[i-1] == NUOC && x[i] == NUOC) ++sotau;
        }
    }
    x = y;
}
f.close();
cout << "\n So tau = " << sotau;
}

main() {
    DemTau("SHIPS.INP");
    cout << "\n T h e   E n d";
    return 0;
}
```

## Output

```
So tau = 5
T h e   E n d
```

## 2.15 Sắp mảng

### Sort trong C++

#### Sort theo tiêu chí định sẵn

Đôi khi ta gọi kiểu sort này là sắp xếp tự nhiên.

Trong C++ để *sắp tăng* một đoạn trong mảng một chiều a từ phần tử a[i] đến phần tử a[j], bạn chỉ cần gọi

```
sort(a+i, a+j+1);
```

Muốn sắp tăng toàn bộ n phần tử của mảng a, bạn gọi

```
sort(a, a+n);
```

### Chương trình minh họa C++

```
// ArraySort.CPP
#include <iostream>
```

```
#include <bits/stdc++.h>
using namespace std;

const int n = 10;
int a[] = {2, 6, 0, 8, 1, 7, 4, 9, 3, 5};
int b[n];

// Hien thi mang 1D a[d..c]
void Print(int a[], int n, int c, int d, char * msg = "") {
    cout << msg;
    for (int i = 0; i < c; ++i) {
        cout << " " << a[i];
    }
    cout << " |";
    for (int i = c; i <= d; ++i) {
        cout << " " << a[i];
    }
    cout << " |";
    for (int i = d+1; i < n; ++i) {
        cout << " " << a[i];
    }
}

int main() {
    memcpy(b,a, sizeof(a));
    Print(a, n, 0, n-1, "\n Given a: ");
    sort(a, a+n);
    Print(a, n, 0, n-1, "\n Sorted a: ");
    Print(b, n, 0, n-1, "\n Given b: ");
    sort(b+2, b+7);
    Print(b, n, 2, 7, "\n Sorted b[2:7]: ");
    cout << "\n T h e      E n d .\n";
    return 0;
}
```

## Kết quả

```
Given a: | 2 6 0 8 1 7 4 9 3 5 |
Sorted a: | 0 1 2 3 4 5 6 7 8 9 |
Given b: | 2 6 0 8 1 7 4 9 3 5 |
Sorted b[2:7]: 2 6 | 0 1 4 7 8 9 | 3 5
T h e      E n d .
```

Bạn muốn sắp giảm mảng số thực  $a[i:j]$ , bạn viết

```
sort(a+i, a + j + 1, greater<float>());
```

Như vậy, hàm `greater<float>()` sẽ so sánh hai đối tượng kiểu `float` trong `a[]` theo chiều giảm.

### Chương trình minh họa C++

```
// ArraySort.CPP
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

const int n = 10;
float a[] = {2.14,6.8,0,8.3,1.7,7.6,4.13,9.1,3.2,5.1};
float b[n];

// Hien thi mang 1D a[d..c]
void Print(float a[], int n, int c, int d, char * msg = "") {
    cout << msg;
    for (int i = 0; i < c; ++i) {
        cout << " " << a[i];
    }
    cout << " |";
    for (int i = c; i <= d; ++i) {
        cout << " " << a[i];
    }
    cout << " |";
    for (int i = d+1; i < n; ++i) {
        cout << " " << a[i];
    }
}

int main() {
    memcpy(b, a, sizeof(a));
    Print(a, n, 0, n-1, "\n Given a:");
    sort(a, a+n, greater<float>());
    Print(a, n, 0, n-1, "\n dec sorted a:");
    Print(b, n, 0, n-1, "\n Given b:");
    sort(b+3, b+7, greater<float>());
    Print(b, n, 3, 7, "\n dec Sorted b[3:7]:");
    cout << "\n T h e      E n d .\n";
    return 0;
}
```

### Output

```
Given a: | 2.14 6.8 0 8.3 1.7 7.6 4.13 9.1 3.2 5.1 |
dec sorted a: | 9.1 8.3 7.6 6.8 5.1 4.13 3.2 2.14 1.7 0 |
Given b: | 2.14 6.8 0 8.3 1.7 7.6 4.13 9.1 3.2 5.1 |
dec Sorted b[3:7]: 2.14 6.8 0 | 8.3 7.6 4.13 1.7 9.1 | 3.2 5.1
T h e      E n d .
```

## Sort theo tiêu chí tự đặt

Đôi khi bạn muốn thay đổi tiêu chí so sánh các phần tử của mảng, ví dụ bạn muốn sắp tăng mảng nguyên không âm  $a$  theo độ cao  $H$  của mỗi số.

Độ cao  $H$  của một số nguyên không âm  $x$  là tổng các chữ số của số  $x$ .

### Ví dụ

$H(123) = 1 + 2 + 3 = 6$ ,  $H(10) = 1 + 0 = 1$ ,  $H(0) = 0$ .

Bạn muốn cài đặt hàm  $H$  theo tùy biến, cụ thể là  $H(x)$  sẽ cho ra tổng các chữ số của số nguyên  $x$  theo hệ đếm 10 (hệ thập phân), còn  $H(x, 2)$  sẽ cho ra tổng các chữ số của số nguyên  $x$  theo hệ đếm 2. Nói cách khác,  $H(x, 2)$  cho biết trong dạng nhị phân biểu diễn số nguyên  $x$  có bao nhiêu bit 1.

### Ví dụ

$H(19) = 1 + 9 = 10$ ,  $H(19, 2) = 3$ , vì  $19_2 = 10011$ .

Việc cài đặt hàm  $H(x)$  khá dễ, bạn chỉ việc lấy tổng các chữ số của  $x$  là  $x \bmod \text{base}$  tính theo hệ đếm nào thì ta chỉ việc chia theo hệ đếm đó. Hàm  $x \div \text{base}$  bỏ đi chữ số đơn vị của  $x$ .

### Hàm $H(x)$

```
int H(int x, base = 10) {
    int digit_sum = 0;
    while (x != 0) {
        digit_sum += x % base;
        x /= base;
    }
    return digit_sum;
}
```

Tiếp đến bạn phải tự cài đặt phương thức so sánh hai phần tử của mảng  $a$ . Bạn quy định rằng

$x$  đứng trước  $y$  khi và chỉ khi  $H(x) < H(y)$ :

```
bool Less(int x, int y) {
    return H(x) < H(y);
}
```

Đến đây bạn gọi hàm `sort`:

```
sort(a, a+n, Less);
```

Nếu muốn sắp giảm bạn gọi

```
sort(a, a+n, Great);
```

với hàm Great do bạn tự viết như sau:

```
bool Great(int x, int y) {
    return H(x) > H(y);
}
```

## Chương trình minh họa C++

```
// ArraySort.CPP
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

const int N = 9;
const string DIGIT = "01234567891abcdef";
int a[] = {20, 16, 10, 111, 73, 46, 19, 33, 15};
int b[N];

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Do cao theo base 10 | 2
int H(int x, int base = 10) {
    int digit_sum = 0;
    while (x != 0) {
        digit_sum += x % base;
        x /= base;
    }
    return digit_sum;
}

// so x theo he dem base
string Str(int x, int base = 10) {
    if (x == 0) return "0";
    string s = "";
    while (x != 0) {
        s = DIGIT[x % base] + s;
        x /= base;
    }
    return s;
}

// Hien thi mang 1D a[d..c]
void Print(int a[], int n, int c, int d, char * msg = "", int base = 10) {
    cout << msg;
    for (int i = 0; i < c; ++i) {
        cout << " " << Str(a[i], base);
    }
}
```

```

    }
    cout << " |";
    for (int i = c; i <= d; ++i) {
        cout << " " << Str(a[i], base);
    }
    cout << " |";
    for (int i = d+1; i < n; ++i) {
        cout << " " << Str(a[i], base);
    }
}

// x < y theo base 10
bool Less(int x, int y) {
    return H(x) < H(y);
}

// x > y theo base 10
bool Great(int x, int y) {
    return H(x) > H(y);
}

// x < 2 theo base 2
bool Less2(int x, int y) {
    return H(x, 2) < H(y, 2);
}

// x > y theo base 2
bool Great2(int x, int y) {
    return H(x, 2) > H(y, 2);
}

int main() {
    memcpy(b, a, sizeof(a)); // b = a
    // b = a
    Print(a, N, 0, N-1, "\nGiven a in base 10:");
    sort(a, a+N, Less);
    Print(a, N, 0, N-1, "\ninc sorted a by H in base 10:");
    sort(a, a+N, Great);
    Print(a, N, 0, N-1, "\ndec sorted a by H in base 10:");
    Print(a, N, 0, N-1, "\nNow a in base 2: ", 2);
    sort(a, a+N, Less2);
    Print(a, N, 0, N-1, "\ninc sorted a by H in base 2:", 2);
    sort(a, a+N, Great2);
    Print(a, N, 0, N-1, "\ndec sorted a by H in base 2:", 2);
    int i1 = 2, i2 = 7;
    Print(b, N, 0, N-1, "\nGiven b in base 10:");
    sort(b+i1, b+i2+1, Less);
    Print(b, N, i1, i2, "\n inc sorted b[2:7] by H in base 10:");
    sort(b+i1, b+i2+1, Great);
    Print(b, N, i1, i2, "\ndec sorted b[2:7] by H in base 10:");
    Print(b, N, 0, N-1, "\nNow b in base 2: ", 2);
    sort(b+i1, b+i2+1, Less2);
    Print(b, N, i1, i2, "\ninc sorted b[2:7] by H in base 2:", 2);
}

```



```
sort(b+i1, b+i2+1, Great2);
Print(b, N, i1, i2, "\ndec sorted b[2:7] by H in base 2:", 2);
cout << "\n T h e      E n d .\n";
return 0;
}
```

Output

```
Given a in base 10: | 20 16 10 111 73 46 19 33 15 |
inc sorted a by H in base 10: | 10 20 111 33 15 16 73 46 19 |
dec sorted a by H in base 10: | 73 46 19 16 33 15 111 20 10 |
Now a in base 2: | 1001001 101110 10011 10000 100001 1111 1101111
10100 1010 |
inc sorted a by H in base 2: | 10000 100001 10100 1010 1001001 10011
101110 1111 1101111 |
dec sorted a by H in base 2: | 1101111 101110 1111 1001001 10011
100001 10100 1010 10000 |
Given b in base 10: | 20 16 10 111 73 46 19 33 15 |
inc sorted b[2:7] by H in base 10: 20 16 | 10 111 33 73 46 19 | 15
dec sorted b[2:7] by H in base 10: 20 16 | 73 46 19 33 111 10 | 15
Now b in base 2: | 10100 10000 1001001 101110 10011 100001 1101111
1010 1111 |
inc sorted b[2:7] by H in base 2: 10100 10000 | 100001 1010 1001001
10011 101110 1101111 | 1111
dec sorted b[2:7] by H in base 2: 10100 10000 | 1101111 101110
1001001 10011 100001 1010 | 1111
T h e      E n d .
```

Sort theo chỉ dẫn

Trong quá trình sort, các đối tượng trong mảng thường phải đổi chỗ với nhau cho nên chi phí cho các phép toán copy thường làm tăng thời gian sắp xếp, đặc biệt là với những đối tượng có kích thước lớn như string hoặc bản ghi. Sort theo chỉ dẫn cho phép chúng ta chỉ ra *một trật tự*, ví dụ, tăng dần, các đối tượng của mảng nhưng vẫn *phân giữ nguyên vị trí ban đầu* của chúng.

Ví dụ

Ta có mảng name chứa danh sách các bạn trẻ. Ta cần duyệt danh sách này theo trật tự từ điển như dòng 4 trong bảng dưới đây:

init name	HOA	KHANH	AN	TUAN	BINH	VINH	CHINH	VAN	DUNG	MAI
Init id	0	1	2	3	4	5	6	7	8	9
sorted id	2	4	6	8	0	1	9	3	7	5
Print name by id	AN	BINH	CHINH	DUNG	HOA	KHANH	MAI	TUAN	VAN	VINH

Để thực hiện điều này ta cần một mảng phụ tạm gọi là id để quản lý số thứ tự của mảng a.

Quy trình sort sẽ được thực hiện qua các bước sau:

Bước 1. Khởi trị cho mảng id:

```
id[i] = i, 0 ≤ i < n
```

trong đó n là số phần tử của mảng a.

Dòng 2 của bảng cho ta kết quả của bước khởi trị id.

Bước 2. Cài đặt hàm Less(i,j) chỉ rõ phương thức so sánh hai đối tượng name[i] và name[j], cụ thể là cho biết name[i] < name[j] ?

```
bool Less(int i, int j) {
    return name[i] < name[j];
}
```

Bước 3. Gọi hàm sort mảng id chứ không phải mảng name:

```
sort(id, id+n, Less);
```

với ý nghĩa sắp xếp lại mảng phụ id từ vị trí đầu đến cuối theo phương thức so sánh Less.

Như vậy chỉ có *các phần tử trong id được sắp xếp lại*, còn bản thân các string trong mảng name vẫn được giữ nguyên.

Dòng 3 của bảng cho ta kết quả của bước 3

```
id = [2, 4, 6, 8, 0, 1, 9, 3, 7, 5]
```

với ý nghĩa:

- id[0] = 2 cho biết phần tử name[2] (AN) sẽ đứng ở vị trí đầu tiên,
- id[1] = 4 cho biết phần tử name[4] (BINH) sẽ đứng ở vị trí thứ hai...

Ngoài ưu thế tiết kiệm thời gian, sort theo chỉ dẫn còn được ứng dụng trong các bài toán cần tham chiếu đến vị trí ban đầu của dữ liệu.

## Chương trình minh hoạ C++

```
// ArraySort.CPP
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

string name[] = {"HOA", "KHANH", "AN", "TUAN", "BINH",
                "VINH", "CHINH", "VAN", "DUNG", "MAI"};

int n = 10;
int *id;

void Print(string a[], int n, char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
    }
}

void Print(int x[], int n, char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << x[i];
    }
}

void PrintById(string *a, int *id, int n, char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << a[id[i]];
    }
}

bool Less(int i, int j) {
    return name[i] < name[j];
}

int main() {
    Print(name, n, "\n Given a: ");
    id = new int[n];
    for (int i = 0; i < n; ++i) {
        id[i] = i;
    }
    Print(id, n, "\n Init id: ");
    sort(id, id+n, Less);
    Print(id, n, "\n Now id: ");
    PrintById(name, id, n, "\n Now Names: ");
    cout << "\n T h e      E n d .\n";
    return 0;
}
```

```
}

```

## Output

```
Given a:  HOA KHANH AN TUAN BINH VINH CHINH VAN DUNG MAI
Init id:  0 1 2 3 4 5 6 7 8 9
Now id:   2 4 6 8 0 1 9 3 7 5
Now Names: AN BINH CHINH DUNG HOA KHANH MAI TUAN VAN VINH
T h e      E n d .
```

## Tổng kết các hàm Sort trong C++

**sort 1D array a[i:j]**

<code>sort(a+i, a+j+1)</code>	<i>sắp tăng tự nhiên</i>
<code>sort(a+i, a+j+1, Less)</code>	<i>sắp tăng theo tiêu chí tự đặt</i>
<code>sort(a+i, a+j+1, greater&lt;type&gt;())</code>	<i>sắp giảm tự nhiên</i>
<code>sort(a+i, a+j+1, Great)</code>	<i>sắp giảm theo tiêu chí tự đặt</i>
<code>type: int, float, string,..., Less, Great: tự đặt</code>	

**ID sort 1D array a[i:j]**

`id[i] = i,  $0 \leq i < n$`

`sort(id+i, id+j, Less)` *sắp tăng*

`sort(id+i, id+j, Great)` *sắp giảm*

`Less, Great tự đặt`

## Lật mảng a[i:j] trong C++

```
void Rev(int a[], int i, int j) {
    int t;
    while (i < j) {
        t = a[i]; a[i] = a[j]; a[j] = t;
        ++i; --j;
    }
}
```

## Bài 2.16. Sắp đoạn

Trong một tệp văn bản chứa tối đa 200 đoạn trên trục số. Mỗi đoạn có thể là một trong các dạng sau đây:

$[d, c]$ : đoạn đóng là dãy số nguyên từ  $d..c$ , ví dụ,  $[3, 7] = 3, 4, 5, 6, 7$ .

$[d, c)$ : đoạn mở đầu phải là dãy số nguyên từ  $d..c-1$ , ví dụ,  $[3, 7) = 3, 4, 5, 6$ .

$(d, c]$  đoạn mở đầu trái là dãy số nguyên từ  $d+1..c$ , ví dụ,  $(3, 7] = 4, 5, 6, 7$ .

$(d, c)$  đoạn mở là dãy số nguyên từ  $d+1..c-1$ , ví dụ,  $(3, 7) = 4, 5, 6$ .

$d$  và  $c$  có thể là các biểu thức dạng  $x + y$ ,  $x - y$  hoặc  $x * y$ ,  $x$  và  $y$  là các số nguyên dương. Ta luôn có  $d \leq c$ . Chiều dài của đoạn là chiều dài của dãy số dạng khai triển của đoạn đó. Ví dụ,  $[3, 7]$  có chiều dài 5,  $[3, 7)$  có chiều dài 4,  $(3, 7]$  có chiều dài 4,  $(3, 7)$  có chiều dài 3.

Hãy sắp xếp các đoạn tăng theo chiều dài và ghi chúng vào một tệp văn bản theo đúng dạng thức đọc được của mỗi đoạn. Có thể thêm, bớt một số dấu cách trong và ngoài các đoạn. Trên mỗi dòng của tệp input luôn luôn chứa trọn một số đoạn được ghi cách nhau qua dấu cách. Trong tệp output, mỗi đoạn được ghi trên một dòng.

DOAN.INP	DOAN.OUT
[2+1, 7) (4, 4*3)	(5, 6]
(5, 6]	[2+1, 7)
	(4, 4*3)

## Thuật toán

Ta mô tả cấu trúc của mỗi đoạn như sau:

$\langle \text{dau} \rangle \langle x1 \rangle \langle t1 \rangle \langle y1 \rangle \langle , \rangle \langle x2 \rangle \langle t2 \rangle \langle y2 \rangle \langle \text{cuoi} \rangle$

trong đó:

- ♦  $\text{dau}$  là một trong hai dấu mở ngoặc: ( hoặc [.
- ♦  $x1, y1, x2$  và  $y2$  là các số tự nhiên xuất hiện trong thành phần của đoạn.
- ♦  $t1$  và  $t2$  là dấu các phép toán (+, -, \*), nếu có trong thành phần của đoạn.
- ♦  $\text{cuoi}$  là một trong hai dấu đóng ngoặc: ) hoặc ].

Trong mô tả trên, chúng ta sử dụng ký pháp  $[_]$  để chỉ ra thành phần  $_$  có thể bỏ qua.

Nếu thành phần thứ  $i$  ( $i = 1..2$ ) của đoạn không có dấu phép toán là  $t1$  hoặc  $t2$ , thì cũng không có toán hạng thứ hai là  $y1$  hoặc  $y2$ .

## Ví dụ

Đoạn	dau	x1	t1	y1	,	x2	t2	y2	cuoi
[2+10, 7*6)	[	2	+	10	,	7	*	6	)
[2+10, 7)	[	2	+	10	,	7	BL	0	)
(2, 7+5]	(	2	BL	0	,	7	+	5	]

Ngoài ra ta thêm một thành phần len để xác định chiều dài của đoạn. len của mỗi đoạn được tính theo công thức sau:

```
len = (x2 t2 y2) - (x1 t1 y1) + 1
if dau == '(' then len -= 1
if cuoi == ')' then len -= 1
```

trong đó  $(x1 \ t1 \ y1)$  và  $(x2 \ t2 \ y2)$  là kết quả tính toán của hai biểu thức tương ứng,  $\text{dau}$  và  $\text{cuoi}$  là các dấu mở và đóng ngoặc tương ứng.

## Thuật toán

1. Đọc toàn bộ dữ liệu vào string s

2. Bỏ các dấu cách trong s
3. Cắt s thành n đoạn ghi vào mảng Doan
4. Xử lý từng đoạn để tính Len
5. Sắp tăng n đoạn theo Len
6. Ghi file

Lệnh

```
getline(f,line,END); // doc toan bo file
```

đọc toàn bộ dữ liệu từ file f vào biến string line.

Thủ tục

```
GetDoan(int &i)
```

tách một đoạn từ string s đưa vào string w. Biến chỉ số i sẽ trở đến vị trí sát sau cuối mỗi đoạn.

Mỗi đoạn sẽ được xử lý theo hai cụm là

<x1>[<t1><y1>] và  
<x2>[<t2><y2>]

Kết quả xử lý mỗi cụm là giá trị của biểu thức

x1 = <x1><t1><y1> và  
x2 = <x1><t1><y1>

Chương trình vận dụng kỹ thuật xử lý dãy kí tự s với mục đích sau: Giả sử ta cần xuất phát từ vị trí i trong string s để đọc một số nguyên trong s. Ta viết thủ tục

```
int GetInt(int &i)
```

nhận trị vào là sting s và chỉ số i, trong đó ta khai báo s là biến tổng thể, i là biến con trỏ, nghĩa là i vừa là biến vào vừa là biến ra, ta gọi i là *biến địa chỉ* hay là *biến vào/ra*.

s	0	1	2	3	4	5	6	7	8	9		s = "(25+7,8-2]"
s	(	2	5	+	7	,	8	-	2	]		i = 0
												x = GetNum(i)
												// x = 25
												// i = 3

Trước khi gọi hàm ta có

```
s = "(25+7,8-2]"
i = 0
```

Sau khi gọi hàm x = GetInt(i) ta thu được

```
x = 25
i = 3
```

Sở dĩ ta muốn biến i ghi nhận lại vị trí mới trong s là để có thể tiếp tục xử lý s từ vị trí mới này.

Trong C++, thủ tục GetNum(int &i) khi đó sẽ hoạt động theo sơ đồ sau:

```
// doc mot so nguyen tu w[i]
int GetNum(int &i) {
    int v = 0;
    while(IsDigit(w[i])) {
        v = v * 10 + (w[i] - '0');
        ++i;
    }
    // cout << "\n Num = " << v;
    return v;
}
```

Kỹ thuật xử lý string theo vị trí biến thiên cũng được dùng trong việc cắt string s chứa input thành các đoạn theo kí tự đầu đoạn '(' hoặc '[' và cuối đoạn ')' hoặc ']'.

## Chương trình C++

```
// Sap doan
#include <iostream>
#include <fstream>
#include <bits/stdc++.h>

using namespace std;

const int MN = 200;

typedef struct Doan {
    string Data;
    int Len;
} Doan;

Doan d[MN];

int n; // so doan
char BL = 32;
char END = '\0';
string s, w;

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

// ki tu c la chu so 0..9
inline bool IsDigit(char c) {
    return (c >= '0' && c <= '9');
}

inline bool IsOperator(char c) {
    return (c == '+' || c == '-' || c == '*');
}
```

```

// doc mot so nguyen tu w[i]
int GetNum(int &i) {
    int v = 0;
    while(IsDigit(w[i])) {
        v = v * 10 + (w[i] - '0');
        ++i;
    }
    // cout << "\n Num = " << v;
    return v;
}

bool Less(Doan x, Doan y) {
    return x.Len < y.Len;
}

// Mot cum trong doan: <x>[<t><y>]
int Cum(int &i) {
    int x, y;
    char t;
    x = GetNum(i);
    t = w[i];
    if (IsOperator(t)) {
        ++i; y = GetNum(i);
        //cout << "\n y = " << y;
        switch(t) {
            case '+': x += y; break;
            case '-': x -= y; break;
            case '*': x *= y; break;
        } // switch
    }
    return x; // gia tri cua bieu thuc x t y
}

// Thong tin cua doan n
// <dau><x1>[<t1><y1>]<,><x2>[<t2><y2>]<cuoi>
int DoanMoi() {
    int x1, x2;
    int m = w.length();
    char dau = w[0], cuoi = w[m-1];
    d[n].Data = w;
    int i = 1;
    x1 = Cum(i); // cum thu nhat
    //cout << " x1 = " << x1;
    if (w[i] == ',') ++i;
    x2 = Cum(i); // cum thu hai
    //cout << " x2 = " << x2;
    int len = x2-x1 + 1;
    if (dau == '(') --len;
    if (cuoi == ')') --len;
    d[n].Len = len;
    return len;
}

```



```

// tach 1 doan vao w
void GetDoan(int &i) {
    w = "";
    for (; i < s.length(); ++i) {
        if (s[i] == '(' || s[i] == '[') {
            w = s[i];
            for (++i; i < s.length(); ++i) {
                w += s[i];
                if (s[i] == ')' || s[i] == ']')
                    return;
            } // for
        } // if
    } // for
}

// Cat s thanh n doan
// ghi tung doan vao w
void CatDoan() {
    int i = 0;
    n = 0;
    while(true) {
        GetDoan(i);
        if (w == "") break;
        cout << "\n Doan w = " << w;
        // xu li doan w
        DoanMoi();
        ++n;
    }
}

void SapDoan() {
    ifstream f("DOAN.INP");
    string line = "";
    getline(f, line, END); // doc toan bo file
    f.close();
    // bo cac dau cach va dau ket dong
    s = "";
    for (int i = 0; i < line.length(); ++i) {
        if (line[i] != BL && line[i] != '\n')
            s += line[i];
    }
    CatDoan(); // Cat s thanh tung doan
    cout << "\n so doan = " << n;
    sort(d, d+n, Less);
    ofstream g("DOAN.OUT");
    for (int i = 0; i < n; ++i) {
        g << d[i].Data << endl;
    }
    g.close();
}

main() {

```

```
SapDoan();  
cout << "\n T h e    E n d";  
return 0;  
}
```

## Output

```
DOAN.OUT  
(5, 6]  
[2 + 1, 7)  
(4, 4*3)
```

## CHƯƠNG 3

### BÀN PHÍM VÀ MÀN HÌNH

---

#### Bài 3.1. Bảng mã ASCII

*Sinh tệp có tên ASCII.DAT chứa mã ASCII để tiện dùng.*

##### Chú ý

ASCII (đọc là a-ski) là bộ mã *chuẩn* dùng trong trao đổi thông tin của Mĩ và đầu tiên được cài đặt trong các máy tính sử dụng hệ điều hành MS-DOS. Trong bảng mã này, mỗi ký tự có một mã số riêng biệt chiếm 1 byte. Trong C++ ta viết 65 là để biểu thị mã số 65, viết char(65) là để biểu thị ký tự có mã số 65, tức là chữ 'A'. Các ký tự mang mã số từ 0 đến 31 là các ký tự điều khiển, thí dụ, ký tự thứ 13 điều khiển con trỏ văn bản xuống dòng mới, ký tự thứ 10 điều khiển con trỏ văn bản về đầu dòng. Trong các ngôn ngữ lập trình ta thường dùng các kí hiệu điều khiển sau đây:

```
\n : xuống đầu dòng kế tiếp
\t : cách một đoạn trắng thường là 7 dấu cách
\r : về đầu dòng
\a : phát tiếng kêu bip
\\ : In ra dấu \
\" : In ra dấu "
\' : In ra dấu '
%%: In ra dấu %
\b: lùi một vị trí
```

Chương trình dưới đây ghi vào tệp văn bản có tên ASCII.DAT các kí tự từ 32 (dấu cách) đến 127. Tất cả có 256 ký tự chia làm hai phần. 128 ký tự đầu tiên mã số từ 0 đến 127 là *các ký tự cơ sở*, 128 ký tự còn lại, mã số từ 128 đến 255 là *các ký tự mở rộng*.

Sau khi thực hiện chương trình, bạn có thể mở tệp ASCII.DAT để xem từng ký tự và mã của chúng. Lưu ý rằng có ký tự hiển thị được và có ký tự không hiển thị được trên màn hình, chẳng hạn như các ký tự điều khiển.

Sau khi xuất hiện màn hình đồ họa và đặc biệt là sau khi có chuẩn UNICODE việc điều khiển màn hình đòi hỏi thiết lập một loạt tham số như kích thước cửa sổ màu chữ, màu nền, cỡ chữ, khả năng co giãn, dịch chuyển vị trí cửa sổ, vv thường gây rối cho những bạn mới học lập trình do đó chương này chỉ giới hạn ở một vài kiến thức cơ sở. Khi bạn đã thành thạo việc thiết lập các mẫu kiến trúc việc tạo các giao diện sẽ trở nên dễ dàng.

#### Chương trình C++

```
// ASCII
#include <iostream>
#include <fstream>
```

```
using namespace std;

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

void ASCII() {
    ofstream f("ASCII.DAT");
    for (int i = 32; i < 127; ++i) {
        f << (char)i << ": " << i << endl;
    }
    f.close();
}

main() {
    ASCII();
    cout << "\n T h e   E n d";
    return 0;
}
```

### File ASCII.DAT

```
: 32
!: 33
": 34
#: 35
$: 36
%: 37
&: 38
': 39
(: 40
): 41
*: 42
+: 43
,: 44
-: 45
.: 46
/: 47
0: 48
1: 49
2: 50
3: 51
4: 52
5: 53
6: 54
7: 55
8: 56
9: 57
:: 58
```

;: 59  
<: 60  
=: 61  
>: 62  
?: 63  
@: 64  
A: 65  
B: 66  
C: 67  
D: 68  
E: 69  
F: 70  
G: 71  
H: 72  
I: 73  
J: 74  
K: 75  
L: 76  
M: 77  
N: 78  
O: 79  
P: 80  
Q: 81  
R: 82  
S: 83  
T: 84  
U: 85  
V: 86  
W: 87  
X: 88  
Y: 89  
Z: 90  
[: 91  
\\: 92  
]: 93  
^: 94  
\_: 95  
` : 96  
a: 97  
b: 98  
c: 99  
d: 100  
e: 101  
f: 102  
g: 103  
h: 104  
i: 105  
j: 106  
k: 107  
l: 108  
m: 109  
n: 110  
o: 111

```

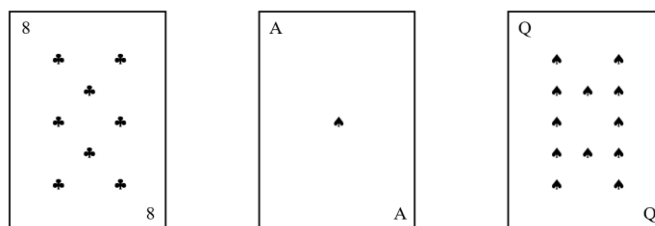
p: 112
q: 113
r: 114
s: 115
t: 116
u: 117
v: 118
w: 119
x: 120
y: 121
z: 122
{: 123
|: 124
}: 125
~: 126

```

### Bài 3.2. Bộ Tú lơ khơ

Lập chương trình hiển thị trên màn hình các quân bài Tú lơ khơ gồm Rô, Cơ, Pích, Nhép theo quy định quân A mang mã số 1 và có 1 hình đơn vị, các quân mã số  $i$  từ 2 đến 10 có  $i$  hình đơn vị, các quân J, Q và K lần lượt có 11, 12 và 13 hình đơn vị tương ứng. Hình đơn vị gồm bốn loại ký tự có mã ASCII tương ứng như sau:

♦ (Rô) : 4, ♥ (Cơ) : 3, ♠ (Pích): 6, ♣ (Nhép): 5.

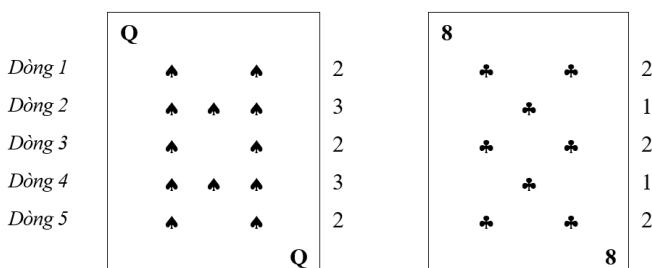


Ba quân bài Tú lơ khơ

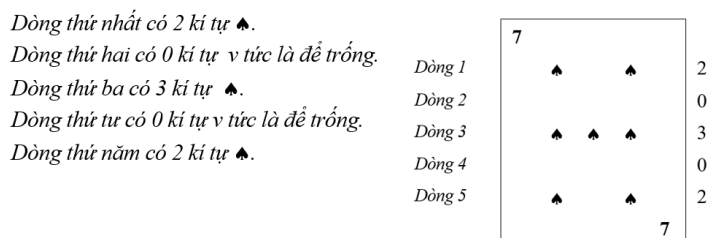
### Thuật toán

Trước hết ta cần thống nhất một số quy định sau:

- ♦ Mỗi quân bài có hai thuộc tính là loại (Rô, Cơ, Pích hoặc Nhép) và nhân (mã số). Nhân của quân A là 1, J là 11, Q là 12 và K là 13. Các quân còn lại mang nhân từ 2 đến 10 ứng với số ghi trên quân bài đó.
- ♦ Trên nền các quân bài J, Q và K không vẽ hình người mà vẽ số lượng hình đơn vị (Rô, Cơ, Pích hoặc Nhép) tương ứng với nhân của quân đó.



Để bố trí số lượng hình đơn vị trên mỗi quân bài cho cân đối ta cần 5 dòng. Thủ tục `QuanBai(int nhan, char v)` vẽ 5 dòng chứa hình đơn vị loại `v` theo dấu hiệu ghi trong xâu mẫu `s`. Ví dụ, lời gọi với xâu mẫu `s = '20302'` sẽ vẽ 5 dòng thể hiện cho quân mang mã số 7 thuộc loại `v = Pích` như sau:



Vì trong xâu mẫu `s` tổng cộng có  $2 + 3 + 2 = 7$  kí tự ♠ nên quân bài mang mã số 7.

Các mẫu dòng và nhãn được tính toán trước và khởi trị như sau:

```
string MauDong[] = {"",
    "00100", "01010", "10101", "20002", "20102",
    "20202", "20302", "21212", "30303", "22222",
    "22322", "23232", "23332"};

string Nhan[] = {"",
    "A", "2", "3", "4", "5",
    "6", "7", "8", "9", "10",
    "J", "Q", "K"};
```

## Chương trình C++

```
// TuLoKho
#include <iostream>
#include <fstream>

using namespace std;
const int MN = 14;

const char RO = 4; // ?
const char CO = 3; // ?
const char PIC = 6; // ?
```

```

const char NHEP = 5; // ?
const char BL = 32;

string MauDong[] = {"",
    "00100", "01010", "10101", "20002", "20102",
    "20202", "20302", "21212", "30303", "22222",
    "22322", "23232", "23332"};

string Nhan[] = {"",
    "A", "2", "3", "4", "5",
    "6", "7", "8", "9", "10",
    "J", "Q", "K"};

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

// m dau cach
void Cach(int m) {
    for (int i = 1; i <= m; ++i)
        cout << BL;
}

void QuanBai(int n, char loai) {
    string s = MauDong[n];
    cout << "-----" << endl;
    cout << Nhan[n] << endl;
    for (int i = 0; i < s.length(); ++i) {
        switch (s[i]) {
            case '1': Cach(3); cout << loai; Cach(2); break;
            case '2': Cach(1); cout << loai; Cach(3);
                       cout << loai; break;
            case '3': cout << BL << loai << BL << loai << BL << loai;
                       break;
        } // switch
        cout << endl;
    } // for
    Cach(7); cout << Nhan[n] << endl;
    cout << "-----" << endl;
}

void TuLoKho() {
    QuanBai(1,R0); Go();
    QuanBai(2,C0); Go();
    QuanBai(10,PIC); Go();
    QuanBai(12,NHEP); Go();
}

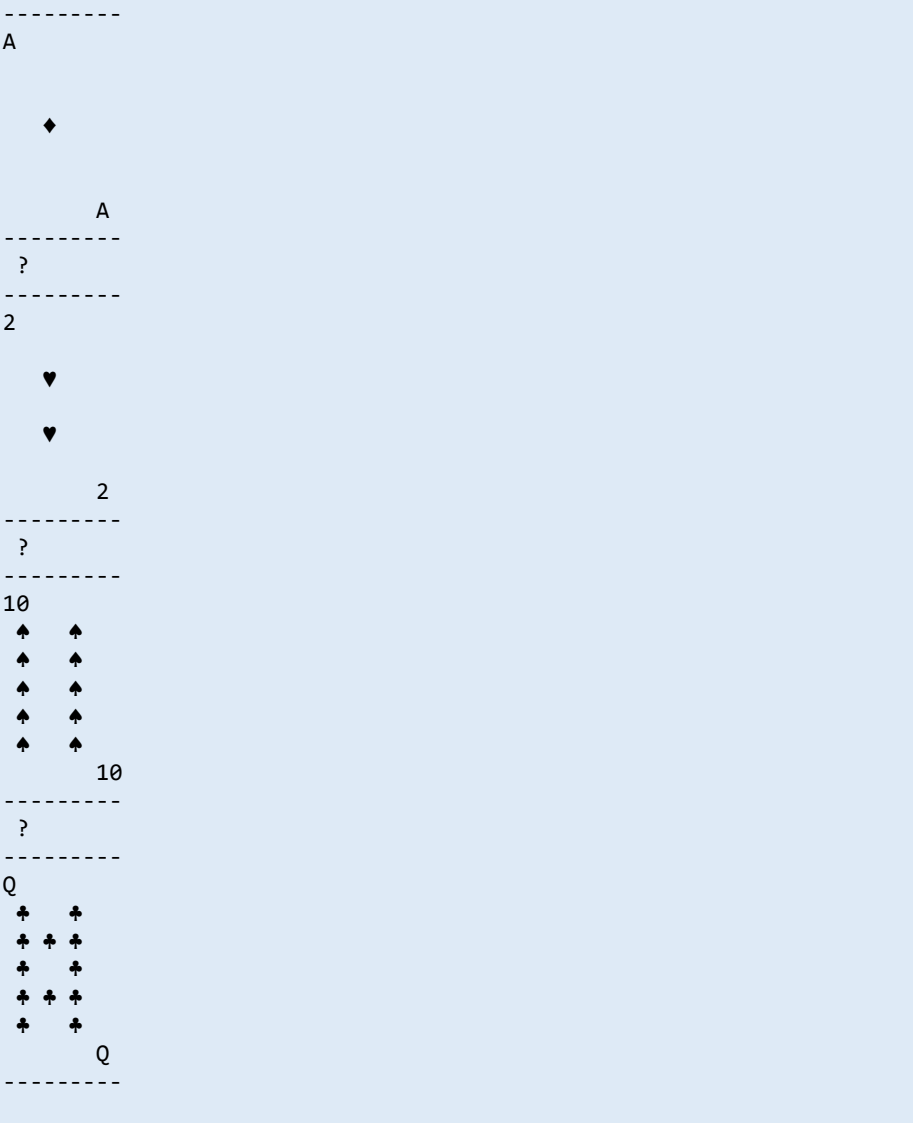
main() {
    TuLoKho();
}

```



```
    cout << "\n T h e   E n d";
    return 0;
}
```

Output



Bài 3.3. Trò chơi 15

Có 15 quân cờ được đánh mã số từ 1 đến 15 được đặt trong một bàn cờ hình vuông  $4 \times 4$  ô theo hình trạng ban đầu như trong hình. Mỗi bước đi, ta được phép di chuyển một quân nằm cạnh ô trống vào trong ô trống.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Trò chơi 15

Viết chương trình thực hiện hai chức năng sau đây:

a) Đảo ngẫu nhiên các quân cờ để chuyển từ hình trạng ban đầu về một hình trạng  $H$  nào đó.

b) Nhận phím điều khiển của người chơi rồi di chuyển quân cờ theo phím đó. Khi nào người chơi đạt được hình trạng ban đầu thì kết thúc một ván.

Trò chơi này có tên là *Trò chơi 15*, từng nổi tiếng ở thế kỉ XIX như trò chơi Rubic ở thời đại chúng ta vậy.

## Thuật toán

Trò chơi này khá dễ lập trình.

Thủ tục Init khởi trị cấu hình gốc có dạng như hình trong đề bài.

```
void Init() {
    int k = 0;
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j)
            a[i][j] = b[i][j] = ++k;
    }
    ei = ej = 3;
    a[ei][ej] = b[ei][ej] = EMPTY;
}
```

Trong đó

$MN = 4$ ,  $EMPTY = 0$

$ei$  và  $ej$  là tọa độ xuất phát của ô trống.

$a[][]$  là mảng  $4 \times 4$  dùng cho người chơi, ta tạm gọi là bảng số.

$b[][]$  là bản sao của bảng số  $a$  theo cấu hình ban đầu.

Thủ tục Dao( $m$ ) sẽ chuyển vị ngẫu nhiên  $m$  lần các số trong bảng số  $a$  với ô trống bên cạnh. Giá trị ngầm định là  $m = 100$ .

Ta quy ước sử dụng các phím sau đây:

l (đẩy lên): đẩy quân dưới ô trống vào ô trống

x (đẩy xuống): đẩy quân trên ô trống vào ô trống

t (đẩy trái): đẩy quân bên phải ô trống vào ô trống

p (đẩy phải): đẩy quân bên trái ô trống vào ô trống

Theo nguyên tắc xuôi ngược ta đẩy số vào ô trống cũng có nghĩa là đẩy ô trống vào vị trí của số theo chiều ngược lại. Bạn nên lưu ý đến nguyên tắc này khi triển khai chương trình. Ta cũng muốn ngầm cài đặt thêm một số chức năng tiện ích cho trò chơi, ví dụ:

Nếu bạn bấm phím [.] tức là bạn muốn dừng cuộc chơi.

Nếu bạn bấm phím [:] ngay sau bước đảo ngẫu nhiên, tức là bạn muốn xem đáp án. Làm theo đáp án này bạn sẽ được chuyển các số về hình trạng ban đầu.

Bạn có thể mở rộng thêm một vài tiện ích khác như demo tự động hoặc giải trình các bước của người chơi.

Trong quá trình đảo ngẫu nhiên bạn cần lưu lại các bước đảo theo nguyên tắc xuôi ngược và ghi vào biến string `dapAn` để hiển thị cho người chơi khi họ có yêu cầu. Khi dịch chuyển số, tức là dịch chuyển ô trống theo chiều ngược lại bạn cần kiểm tra vị trí dòng `ei` và cột `ej` của ô trống khi ô trống nằm trên biên.

## Chương trình C++

```
// Puzzle 15
#include <iostream>
#include <windows.h>
#include <ctime>

using namespace std;

const int MN = 4; // kích thước bảng số 4 X 4
int a[MN][MN]; // bảng số a
int b[MN][MN]; // lưu ban gốc của a
const char BL = 32; // dấu cách
const int EMPTY = 0; // ô trống
const int soLanDao = 3; // số lần đảo
// Day o so Len, Xuong, Phai, Trai vào ô trống
// Day o trống Xuong, Len, Trai, Phai vào ô số
const string SS = "LXTP"; // eLn Xuong Phai Trai
int ei, ej; // vị trí ô trống trong dòng ei, cột ej
string dao; // chưa day giá trị LXPT khi đảo ngẫu nhiên
string dapAn; // Dung cho đáp án hoặc demo

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// tri của c là LXPT ?
bool Accept(char c) {
    for (int i = 0; i < SS.length(); ++i)
        if (SS[i] == c) return true;
    return false;
}

Print() {
```

```

for (int i = 0; i < MN; ++i) {
    cout << endl;
    for (int j = 0; j < MN; ++j) {
        cout << "[";
        if (a[i][j] < 10) cout << BL;
        if (a[i][j] == 0) cout << BL << " ";
        else cout << a[i][j] << " ";
    }
}

bool Verify() {
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j) {
            if (a[i][j] != b[i][j]) return false;
        }
    }
    return true;
}

/*-----
Khoi tri cau hinh a, b: 1..15
[ 1][ 2][ 3][ 4]
[ 5][ 6][ 7][ 8]
[ 9][10][11][12]
[13][14][15][ 0]
b la ban so de kiem tra ket qua
Verify: a = b ?
-----*/

void Init() {
    int k = 0;
    cout << "\n Xin moi...";
    for (int i = 0; i < MN; ++i) {
        for (int j = 0; j < MN; ++j)
            a[i][j] = b[i][j] = ++k;
    }
    ei = ej = 3;
    a[ei][ej] = b[ei][ej] = EMPTY;
}

void Swap(int i, int j) {
    int ii = ei + i;
    int jj = ej + j;
    int x = a[ei][ej];
    a[ei][ej] = a[ii][jj];
    a[ii][jj] = x;
    ei = ii; ej = jj;
}

/*-----
Chuyen o [so] hay chuyen [] theo chieu nguoc lai
Toa do o [] la (ei, ej)
Xuong: [so] Xuong | [] Len: Swap(-1, 0): +++-ei

```

```

    Len: [so] Len | [] Xuong: Swap(1, 0): ++ei
    Phai: [so] Phai | [] Trai: Swap(0, -1): --ej
    Trai: [so] Trai | [] Phai: Swap(0, 1): ++-j
    Sys = true: ghi nhan lai dap an
    -----*/
void Move(char c, bool sys = true) {
    switch(c) {
        case 'X':
            if (ei > 0) {
                Swap(-1, 0);
                if (sys) dapAn = 'L' + dapAn;
            }
            break;
        case 'L':
            if (ei < 3) {
                Swap(1, 0);
                if (sys) dapAn = 'X' + dapAn;
            }
            break;
        case 'P':
            if (ej > 0) {
                Swap(0, -1);
                if (sys) dapAn = 'T' + dapAn;
            }
            break;
        case 'T':
            if (ej < 3) {
                Swap(0, 1);
                if (sys) dapAn = 'P' + dapAn;
            }
            break;
    } // switch
}

void SysDao() {
    dapAn = "";
    for (int i = 0; i < dao.length(); ++i) {
        Move(dao[i]);
    }
}

void Dao(int m = soLanDao) {
    dao = "";
    for (int v = 0; v < m; ++v)
        dao += SS[rand() % MN];
    SysDao();
    if (Verify()) {
        dao = "XP";
        SysDao();
    }
}

void Demo() {

```

```

Init();
Dao(7);
Print();
cout << "\n Demo: " << dapAn << "...";
for (int i = 0; i < dapAn.length(); ++i) {
    cout << "\n " << dapAn[i];
    Move(dapAn[i], false);
    Print();
    Go();
}
}

// Nhan 1 ki tu tu ban phim
char GetKey() {
    char ch;
    while (true) {
        cout << endl << SS << " ? " ;
        fflush(stdin);
        if ((ch = toupper(cin.get())) == '.')
            exit(0);
        if (ch == ':') { // xem dap an
            cout << "\n " << dapAn;
            return ch;
        }
        if (ch == 'D') {
            Demo();
            return ch;
        }
        if (Accept(ch))      return ch;
    }
}

void Puzzle15() {
    Init();
    int m = 3;
    Dao(m);
    Print();
    char ch;
    while(true) {
        ch = GetKey();
        Move(ch, false);
        Print();
        if (Verify()) {
            cout << "\n Chuc mung thanh cong!";
            m += soLanDao;
            Init();
            Dao(m);
            Print();
        }
    }
}

main() {

```

```

    srand(time(0));
    Puzzle15();
    cout << "\n T h e    E n d";
    return 0;
}

```

## Vài nước đi

```

[ 1][ 2][ ][ 4]
[11][ 7][ 5][12]
[13][ 8][ 3][ 9]
[10][14][ 6][15]
LXTP ? 1

```

```

[ 1][ 2][ 5][ 4]
[11][ 7][ ][12]
[13][ 8][ 3][ 9]
[10][14][ 6][15]
LXTP ? 1

```

```

[ 1][ 2][ 5][ 4]
[11][ 7][ 3][12]
[13][ 8][ ][ 9]
[10][14][ 6][15]
LXTP ? t

```

```

[ 1][ 2][ 5][ 4]
[11][ 7][ 3][12]
[13][ 8][ 9][ ]
[10][14][ 6][15]
LXTP ? x

```

```

[ 1][ 2][ 5][ 4]
[11][ 7][ 3][ ]
[13][ 8][ 9][12]
[10][14][ 6][15]
LXTP ? x

```

```

[ 1][ 2][ 5][ ]
[11][ 7][ 3][ 4]
[13][ 8][ 9][12]
[10][14][ 6][15]
LXTP ? p

```

```

[ 1][ 2][ ][ 5]
[11][ 7][ 3][ 4]
[13][ 8][ 9][12]
[10][14][ 6][15]
LXTP ? 1

```

```

[ 1][ 2][ 3][ 5]

```

```
[11][ 7][ ][ 4]
[13][ 8][ 9][12]
[10][14][ 6][15]
LXTP ?
```

### Bài 3.4. Bảng nhảy

Bảng nhảy bước  $b$ , bậc  $k$  là một tấm bảng có đặc tính kì lạ sau đây: nếu bạn viết lần lượt lên bảng  $n$  số nguyên thì sau khi viết số thứ  $i$ , số thứ  $(i - b)$  đã viết trước đó sẽ được tăng thêm  $k$  đơn vị mà ta gọi là nhảy số.

Với mỗi cặp số nguyên dương  $b$  và  $k$  cho trước hãy lập trình để biến màn hình máy tính của bạn thành một bảng nhảy sau đó thử viết lên tấm bảng đó để nhận được dãy  $N$  số tự nhiên đầu tiên  $1\ 2\ \dots\ N$  với mỗi  $N$  cho trước.

Ví dụ, để thu được dãy số  $1\ 2\ \dots\ 10$  trên bảng nhảy bước  $b = 3$  bậc  $k = 6$  bạn cần viết dãy số sau:

-5 -4 -3 -2 -1 0 1 2 9 10

Tổng quát, bạn thử viết lên tấm bảng để nhận được dãy số tự nhiên liên tiếp  $a..b$  (tính cả hai đầu  $a$  và  $b$ ).

Ví dụ, nếu bạn viết trên bảng nhảy bước  $b = 3$  bậc  $k = 6$  dãy số sau:

4 5 6 7 14 15

thì trên bảng sẽ hiển thị 6 số tự nhiên liên tiếp  $10..15$ :

10 11 12 13 14 15

#### Gợi ý

Muốn hiện dãy số  $d..c$  trên bảng nhảy bậc  $k$  bước  $b$  ta cần viết số  $x$  tại lần viết thứ  $i$  thì viết số  $y = x - k$

Hiện thị dãy số d..c trên bảng nhảy bậc k, bước b							
d-k	(d-k)+1	...	(d-k)+(c-d)-b + 1	c-(b-1)	...	c-1	c
(c-d) + 2 – b số đầu tăng dần				b-1 số cuối giảm dần			
Tổng cộng (c-d+1) số							

### Chương trình C++

```
// Bang Nhay
#include <iostream>
#include <bits/stdc++.h>

using namespace std;

// Hien thi vector a
void Print(vector<int> a, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < a.size(); ++i)
```



```

        cout << " " << a[i];
    }

    void BangNhay(int k, int b) {
        cout << "\n Bang nhay bac " << k << ", buoc " << b;
        int num;
        vector<int> a;
        a.clear();
        int i = 0;
        int it = i-b; // it la chi so truoc i
        string s;
        while (true) {
            cout << "\n Viet mot so. Bam [.] de ket thuc: ";
            fflush(stdin);
            s.clear();
            cin >> s;
            if (s[0] == '.') break;
            // s.c_str chuyen doi string s sang dang char * cua C
            // atoi(c) chuyen doi char * C thanh so
            a.push_back(atoi(s.c_str()));

            ++i; ++it;
            if (it >= 0) a[it] += k; // nhay so
            Print(a, "\n Bang nhay: ");
        }
    }

    main() {
        BangNhay(6, 3);
        // -5 -4 -3 -2 -1 0 1 2 9 10
        // -> 1 2 3 4 5 6 7 8 9 10
        // 4 5 6 7 14 15
        // -> 10 11 12 13 14 15
        cout << "\n T h e   E n d";
        return 0;
    }

```

## Màn hình

```

Bang nhay bac 6, buoc 3
Viet mot so. Bam [.] de ket thuc: 4

Bang nhay:  4
Viet mot so. Bam [.] de ket thuc: 5

Bang nhay:  4 5
Viet mot so. Bam [.] de ket thuc: 6

Bang nhay:  10 5 6
Viet mot so. Bam [.] de ket thuc: 7

Bang nhay:  10 11 6 7

```

Viet mot so. Bam [.] de ket thuc: 14

Bang nhay: 10 11 12 7 14

Viet mot so. Bam [.] de ket thuc: 15

Bang nhay: 10 11 12 13 14 15

Viet mot so. Bam [.] de ket thuc: .

T h e E n d

CHƯƠNG 4

TỔ CHỨC DỮ LIỆU

Bài 4.1. Cụm

Một *cụm* trong một biểu thức toán học là đoạn nằm giữa hai dấu đóng và mở ngoặc đơn (). Với mỗi biểu thức cho trước hãy tách các cụm của biểu thức đó.

Dữ liệu vào: Tập văn bản CUM.INP chứa một dòng kiểu xâu ký tự (string) là biểu thức cần xử lí.

Dữ liệu ra: Tập văn bản CUM.OUT dòng đầu tiên ghi d là số lượng cụm. Tiếp đến là d dòng, mỗi dòng ghi một cụm được tách từ biểu thức.

Thí dụ:

CUM.INP	CUM.OUT
$x*(a+1)*((b-2)/(c+3))$	4 (a+1) (b-2) (c+3) ((b-2)/(c+3))

Thuật toán

Giả sử string s chứa biểu thức cần xử lí. Ta duyệt lần lượt từ đầu đến cuối s, với mỗi ký tự s[i] ta xét hai trường hợp:

- Trường hợp thứ nhất: s[i] là dấu mở ngoặc '(': ta ghi nhận i là vị trí xuất hiện đầu cụm vào một ngăn xếp (stack) st:  
 $st[++p] = i;$   
trong đó p là con trỏ ngăn xếp. p luôn luôn trỏ đến ngọn, tức là phần tử cuối cùng của ngăn xếp. Thủ tục này gọi là nạp trị i vào ngăn xếp. Bạn có thể cài đặt stack st qua một mảng int với số phần tử tối đa bằng chiều dài của string s.
- Trường hợp thứ hai: s[i] là dấu đóng ngoặc ')': ta lấy phần tử ngọn ra khỏi ngăn xếp kết hợp với vị trí i để ghi nhận các vị trí đầu và cuối cụm trong s.

Hàm này gọi là lấy phần tử ra khỏi ngăn xếp. Khi lấy xong một phần tử ra khỏi ngăn xếp ta giảm con trỏ ngăn xếp 1 đơn vị.

```
j = st[p--];
```

Ta dùng biến SoCum để đếm và ghi nhận các cụm xuất hiện trong quá trình duyệt biểu thức. Hai biến đầu và cuối ghi nhận vị trí xuất hiện của ký tự đầu cụm và ký tự cuối cụm.

Hàm

```
s.substr(d, m)
```

xuất ra một string gồm m kí tự tính từ s[d].

## Chương trình C++

```
// Cum
#include <iostream>
#include <fstream>

using namespace std;

const char MO_NGOAC = '(';
const char DONG_NGOAC = ')';

void Cum() {
    ifstream f("CUM.INP");
    ofstream g("CUM.OUT");
    string s = "";
    getline(f,s,'\0'); // doc toan bo noi dung input file
    f.close();
    cout << s;
    int n = s.length();
    int st[n]; // stack
    int p = -1; // pointer for stack
    int dau, cuoi;
    int socum = 0;
    // so dau ( chinh la so cum
    for (int i = 0; i < n; ++i) {
        if (s[i] == MO_NGOAC) ++socum;
    }
    g << socum << endl;
    for (int i = 0; i < n; ++i) {
        if (s[i] == MO_NGOAC) {
            st[++p] = i;
            continue;
        }
        if (s[i] == DONG_NGOAC) {
            dau = st[p--]; // vi tri (
            cuoi = i; // vi tri )
            cout << "\n " << dau << " .. " << cuoi << ": "
                << s.substr(dau,cuoi-dau+1);
            g << s.substr(dau,cuoi-dau+1) << endl;
            continue;
        }
    }
```

```

    }
    g.close();
}

main() {
    Cum();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

x*( a + 1)*((b - 2) / (c + 3))

2 .. 9: ( a + 1)
12 .. 18: (b - 2)
22 .. 28: (c + 3)
11 .. 29: ((b - 2) / (c + 3))
T h e   E n d

```

## Bài 4.2. Bài gộp

Bộ bài bao gồm  $n$  quân, được gán mã số từ 1 đến  $n$ . Lúc đầu bộ bài được chia cho  $n$  người, mỗi người nhận 1 quân. Mỗi lượt chơi, trọng tài chọn ngẫu nhiên hai số  $x$  và  $y$  trong khoảng  $1..n$ . Nếu có hai người khác nhau, một người có trong tay quân bài  $x$  và người kia có quân bài  $y$  thì một trong hai người đó phải trao toàn bộ số bài của mình cho người kia theo nguyên tắc sau: mỗi người trong số hai người đó trình ra một quân bài tùy chọn của mình, Ai có quân bài mang mã số nhỏ hơn sẽ được nhận bài của người kia. Trò chơi kết thúc khi có một người cầm trong tay cả bộ bài. Biết số quân bài  $n$  và các quân bài trọng tài chọn ngẫu nhiên sau  $m$  lượt chơi, hãy cho biết số lượng người còn có bài trên tay.

Dữ liệu vào: Tập văn bản **BAIGOP.INP**.

- Dòng đầu tiên: hai số  $n$  và  $m$ , trong đó  $n$  là số lượng quân bài trong bộ bài,  $m$  là số lần trọng tài chọn ngẫu nhiên hai số  $x$  và  $y$ . Các quân bài được gán mã số từ 1 đến  $n$ . Mã số này được ghi trên quân bài.
- Tiếp đến là  $m$  dòng, mỗi dòng ghi hai số tự nhiên  $x$  và  $y$  do trọng tài cung cấp. Các số trên cùng một dòng cách nhau qua dấu cách.

Dữ liệu ra: Hiển thị trên màn hình số lượng người còn có bài trên tay.

Giới hạn  $n = 1000$ .

Thí dụ:

BAIGOP.INP	Kết quả hiển thị trên màn hình
10 6 2 5 3 3 4 7 1 5 2 8	5

Ý nghĩa: bộ bài có 10 quân mã số lần lượt 1, 2, ..., 10 và có 10 người chơi. Sáu lần trọng tài chọn ngẫu nhiên các cặp số  $(x, y)$  là (2, 5), (3, 3), (4, 7), (1, 5), (2, 8) và (9, 3). Cuối ván chơi còn lại 5 người có bài trên tay: {1, 2, 5, 8}, {3, 9}, {4, 7}, {6}, {10}.

9 3	
-----	--

## Thuật toán

Đây là bài toán có nhiều ứng dụng hữu hiệu nên bạn đọc cần tìm hiểu kĩ và cố gắng cài đặt cho nhuần nhuyễn. Như sau này sẽ thấy, nhiều thuật toán xử lí đồ thị như tìm cây khung, xác định thành phần liên thông, xác định chu trình... sẽ phải vận dụng cách tổ chức dữ liệu tương tự như thuật toán sẽ trình bày dưới đây.

Bài này đòi hỏi tổ chức các tập quân bài sao cho thực hiện nhanh nhất các thao tác sau đây:

$\text{Find}(x)$ : cho biết tên của tập chứa phần tử  $x$ .

$\text{Union}(x, y)$ : hợp tập chứa  $x$  với tập chứa  $y$ .

Mỗi tập là nhóm các quân bài có trong tay một người chơi. Như vậy mỗi tập là một tập con của bộ bài  $\{1, 2, \dots, n\}$ . Ta gọi bộ bài là *tập chủ* hay *tập nền*. Do tính chất của trò chơi, ta có hai nhận xét quan trọng sau đây:

1. Hợp của tất cả các tập con (mỗi tập con này có trên tay một người chơi) đúng bằng tập chủ.
2. Hai tập con khác nhau không giao nhau: tại mỗi thời điểm của cuộc chơi, mỗi quân bài nằm trong tay đúng một người.

Họ các tập con thỏa hai tính chất nói trên được gọi là một *phân hoạch* của tập chủ.

Các thao tác nói trên phục vụ trực tiếp cho việc tổ chức trò chơi theo sơ đồ sau:

```

Khởi trị;
for i:= 1 to n do
begin
  Trọng tài chọn hai số x và y: 1..n:
  Hợp tập chứa x với tập chứa y: Union(x,y);
end;
Thông báo kết quả

```

Để thực hiện thủ tục  $\text{Union}(x, y)$  trước hết ta cần biết quân bài  $x$  và quân bài  $y$  đang ở trong tay ai? Sau đó ta cần biết người giữ quân bài  $x$  (hoặc  $y$ ) có quân bài nhỏ nhất là gì? Quân bài nhỏ nhất được xác định trong toàn bộ các quân bài mà người đó có trong tay. Đây chính là điểm dễ nhầm lẫn.

## Ví dụ

Người chơi A đang giữ trong tay các quân bài 3, 4 và 7,

$A = \{\underline{3}, 4, 7\}$

Người chơi B đang giữ các quân bài 2, 5, 9 và 10,

$B = \{\underline{2}, 5, 9, 10\}$

Các số gạch chân là số hiệu của quân bài nhỏ nhất trong tay mỗi người.

Trọng tài chọn ngẫu nhiên hai số:

$x = 9$  và  $y = 7$

thì A (đang giữ quân  $y = 7$ ) và B (đang giữ quân  $x = 9$ ) sẽ phải đấu với nhau. Vì trong tay A có quân nhỏ nhất là  $\underline{3}$  và trong tay B có quân nhỏ nhất là  $\underline{2}$  nên A sẽ phải nộp bài cho B và ra khỏi cuộc chơi. Ta có,

$B = \{2, 3, 4, 5, 7, 9, 10\}$

Ta kết hợp việc xác định quân bài  $x$  trong tay ai và người đó có quân bài nhỏ nhất là bao nhiêu làm một đề xây dựng hàm  $\text{Find}(x)$ . Cụ thể là hàm  $\text{Find}(x)$  sẽ cho ta *quân bài nhỏ nhất có trong tay người giữ quân bài  $x$* . Trong thí dụ trên ta có:

$$\text{Find}(x) = \text{Find}(9) = 2 \text{ và } \text{Find}(y) = \text{Find}(7) = 3$$

Lưu ý rằng hàm  $\text{Find}(x)$  không chỉ rõ ai là người đang giữ quân bài  $x$  mà cho biết quân bài có số hiệu *nhỏ nhất* có trong tay người đang giữ quân bài  $x$ , nghĩa là  $\text{Find}(9) = 2$  chứ không phải  $\text{Find}(9) = B$ . Để giải quyết sự khác biệt này ta hãy chọn phần tử có số hiệu nhỏ nhất trong tập các quân bài có trong tay một người làm *phần tử đại diện của tập đó*. Ta cũng đồng nhất phần tử đại diện với *mã số của người giữ tập quân bài*. Theo quy định này thì biểu thức  $\text{Find}(9) = 2$  có thể được hiểu theo một trong hai nghĩa tương đương như sau:

- ♦ Người số 2 đang giữ quân bài 9.
- ♦ Tập số 2 chứa phần tử 9.

Tổ chức hàm  $\text{Find}$  như trên có lợi là sau khi gọi  $i = \text{Find}(x)$  và  $j = \text{Find}(y)$  ta xác định ngay được ai phải nộp bài cho ai. Nếu  $i < j$  thì  $j$  phải nộp bài cho  $i$ , ngược lại, nếu  $i > j$  thì  $i$  phải nộp bài cho  $j$ . Trường hợp  $i = j$  cho biết hai quân bài  $x$  và  $y$  đang có trong tay một người, ta không phải làm gì.

Tóm lại ta đặt ra các nguyên tắc sau:

- a) *Lấy phần tử nhỏ nhất trong mỗi tập làm tên riêng đại diện cho tập đó.*
- b) *Phần tử có giá trị nhỏ quản lí các phần tử có giá trị lớn hơn nó theo phương thức: mỗi phần tử trong một tập đều trở trực tiếp đến một phần tử nhỏ hơn nó và có trong tập đó. Phần tử nhỏ nhất trong tập trở tới chính nó.*

Trong ví dụ trên ta có

$$A = \{ \underline{3}, 4, 7 \}, B = \{ \underline{2}, 5, 9, 10 \}, x = 9 \text{ và } y = 7$$

Như vậy, tập  $A$  có phần tử đại diện là 3 và tập  $B$  có phần tử đại diện là 2.

Dữ liệu của tập  $A$  khi đó sẽ được tổ chức như sau:

$$A = \{ 3 \rightarrow 3, 4 \rightarrow 3, 7 \rightarrow 3 \}$$

Như vậy 3 là phần tử đại diện của tập này, do đó ta không cần dùng biến  $A$  để biểu thị nó nữa mà có thể viết:

$$\{ 3 \rightarrow 3, 4 \rightarrow 3, 7 \rightarrow 3 \} \text{ hoặc gọn hơn } \{ 3, 4, 7 \} \rightarrow 3.$$

Tương tự, dữ liệu của tập  $B$  sẽ có dạng:

$$\{ 2 \rightarrow 2, 5 \rightarrow 2, 9 \rightarrow 2, 10 \rightarrow 2 \} \text{ hoặc gọn hơn } \{ 2, 5, 9, 10 \} \rightarrow 2.$$

Khi đó  $\text{Find}(9) = \underline{2}$  và  $\text{Find}(7) = \underline{3}$ , và do đó, tập  $\underline{3}$  phải được gộp vào tập  $\underline{2}$ . Phép Union(9, 7) sẽ tạo ra tập sau đây:

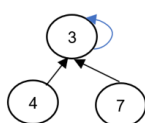
$$\{ 3 \rightarrow 2, 4 \rightarrow 3, 7 \rightarrow 3, 2 \rightarrow 2, 5 \rightarrow 2, 9 \rightarrow 2, 10 \rightarrow 2 \},$$

tức là ta thực hiện đúng một thao tác sửa  $3 \rightarrow 3$  thành  $3 \rightarrow \underline{2}$ : để hợp hai tập ta chỉ việc đối sánh hai phần tử đại diện  $i$  và  $j$  của chúng:

- ♦ Nếu  $i > j$  thì cho tập  $i$  trở đến tập  $j$ .
- ♦ Nếu  $j > i$  thì cho tập  $j$  trở đến  $i$ .
- ♦ Nếu  $i = j$  thì không làm gì.

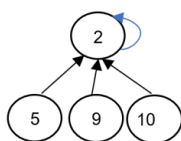
Kĩ thuật nói trên được gọi là *hợp các tập con rời nhau*.

Ta dùng một mảng nguyên  $d$  thể hiện tất cả các tập. Khi đó hai tập  $A$  và  $B$  nói trên được thể hiện trong  $d$  như sau:



$d[3] = 3; d[4] = 3; d[7] = 3;$

tập A: phần tử đại diện là 3,  
các phần tử 3, 4 và 7 đều trỏ đến 3



$d[2] = 2; d[5] = 2;$

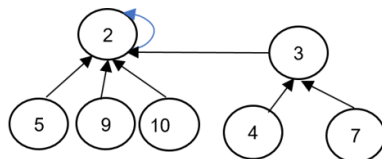
$d[9] = 2; d[10] = 2;$

Tập B: phần tử đại diện là 2,  
các phần tử 2, 5, 9 và 10 đều trỏ đến 2.

Mảng d có dạng:

chỉ số	1	2	3	4	5	6	7	8	9	10
d		2	3	3	2		3		2	2
tập		B	A	A	B		A		B	B

Sau khi hợp nhất A với B ta thu được:



Union(7,9)

Find(7) = 3; Find(9) = 2;

$d[3] = 2$

$d[3] = 2;$  (hỗ sửa duy nhất)

$d[4] = 3; d[7] = 3; d[2] = 2; d[5] = 2; d[9] = 2; d[10] = 2;$

chỉ số	1	2	3	4	5	6	7	8	9	10
d		2	2	3	2		3		2	2
tập		B	A	A	B		A		B	B

Tại bước cuối cùng, nếu muốn, bạn có thể sửa lại các giá trị gián tiếp của mảng trỏ d thành trực tiếp. Ví dụ, bạn có thể thay cặp trỏ gián tiếp (còn gọi là *trở bắc cầu*)  $d[4] = 3$  và  $d[3] = 2$  thành  $d[4] = 2$ .

chỉ số	1	2	3	4	5	6	7	8	9	10
d		2	2	2	2		2		2	2
tập		B	A	A	B		A		B	B

Tóm lại, để quản lý hợp các tập rời nhau ta cần các thủ tục sau:

Khởi trị:  $d[i] = i, i = 1..n$

```
// Tìm tập chứa x
int Find(int x) {
```



```

    while (d[x] != x)
        x = d[x];
    return x;
}

// Hợp tập chứa x với tập chứa y
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x;
    else d[x] = y;
    return 1;
}

```

Hàm Union cho ra giá trị 1 nếu tập chứa x và tập chứa y thực sự được hợp nhất, ngược lại, khi x và y thuộc cùng một tập, thì hàm cho ra giá trị 0.

Nếu  $d[i] = i$  thì  $i$  là đại diện của một tập. Ta lại biết  $i$  là phần tử nhỏ nhất trong tập, do đó ta chỉ cần duyệt các phần tử  $j > i$  và thỏa mãn điều kiện  $\text{Find}(j) = i$  là biết được  $j$  chính là phần tử thuộc tập  $i$ .

Hàm Union cần tối đa hai lần duyệt  $n$  phần tử. Sau  $k$  lần trọng tài xướng hai quân bài  $x$  và  $y$  thì độ phức tạp của thuật toán sẽ là  $O(kn)$ .

## Chương trình C++

```

// Bai gop
#include <bits/stdc++.h>

using namespace std;

const int MN = 1001; // kích thước tối đa

int d[MN]; // mảng tro

void Go() {
    cout << " ? " ;
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

// Hiện thị mảng a kèm chú thích msg
Print(int a[], int n, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
    }
}

// Tìm tập chứa x
int Find(int x) {

```

```

    while (d[x] != x)
        x = d[x];
    return x;
}

// Hop tap chua x voi tap chua y
int Union(int x, int y) {
    x = Find(x); y = Find(y);
    if (x == y) return 0;
    if (x < y) d[y] = x;
    else d[x] = y;
    return 1;
}

void BaiGop() {
    ifstream f("BAIGOP.INP");
    int n, m;
    int x, y;
    f >> n >> m;
    cout << " So nguoi = " << n << " so luot = " << m;
    // Khoi tri d
    for (int i = 1; i <= n; ++i)
        d[i] = i;
    int k = n; // luc dau co k tap roi nhau
    for (int i = 1; i <= m; ++i) {
        f >> x >> y;
        cout << "\n x = " << x << " y = " << y;
        k -= Union(x,y);
    }
    f.close();
    cout << "\n So nguoi con bai: " << k;
    for (int i = 1; i <= n; ++i) {
        if (d[i] == i) {
            cout << "\n Nguoi con bai " << i;
            for (int j = i+1; j <= n; ++j) {
                if (Find(j) == i) {
                    d[j] = i;
                    cout << " " << j;
                }
            }
        }
    }
}

main() {
    BaiGop();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

So nguoi = 10   so luot = 6
x = 2   y = 5
x = 3   y = 3
x = 4   y = 7
x = 1   y = 5
x = 2   y = 8
x = 9   y = 3
So nguoi con bai: 5
Nguoi con bai 1 2 5 8
Nguoi con bai 3 9
Nguoi con bai 4 7
Nguoi con bai 6
Nguoi con bai 10
T h e   E n d

```

### Bài 4.3. Chuỗi hạt

Trong một tệp văn bản tên CHUOI.DAT biểu diễn một chuỗi hạt, mỗi hạt có thể nhận một trong số các màu mã số từ 1 đến 30.

Lập trình thực hiện các việc sau:

- Đọc chuỗi hạt từ tệp vào mảng nguyên dương  $a$ .
- Hiển thị số màu có trong chuỗi.
- Tìm một điểm để cắt chuỗi rồi căng thẳng ra sao cho tổng số các hạt cùng màu ở hai đầu là lớn nhất.

Chuỗi được thể hiện trong tệp dưới dạng hình thoi, dòng đầu tiên và dòng cuối cùng mỗi dòng có một hạt.

Mỗi dòng còn lại có hai hạt (xem hình).

Các hạt của chuỗi được đánh số từ 0 bắt đầu từ hạt trên cùng theo chiều kim đồng hồ.

CHUOI.DAT	
	4
4	7
1	4
5	8
5	8
5	8
	8

Chuỗi hạt

Trong thí dụ này, các thông báo trên màn hình sẽ là:

Số màu trong chuỗi: 5

Cắt giữa hạt 6 và hạt 7, tổng số lớn nhất là 7.

chỉ số	0	1	2	3	4	5	6	7	8	9	10	11
màu	4	7	4	8	8	8	8	5	5	5	1	4

điểm cắt

### Thuật toán

Khung chương trình được phác thảo như sau:

Đọc dữ liệu;

Tính và thông báo số màu  
Tìm điểm cắt

Ta đọc dữ liệu vào hai mảng  $a$  và  $b$ , sau đó nối ngược mảng  $b$  vào cuối mảng  $a$ . Lưu ý rằng chỉ số của chương trình luôn tính từ 0.

chỉ số	0	1	2	3	4	5	6	0	1	2	3	4
màu	4	7	4	8	8	8	8	4	1	5	5	5
	a						b					

Sau khi nối ngược  $b$  vào  $a$  ta thu được:

chỉ số của a	0	1	2	3	4	5	6	7	8	9	10	11
a	4	7	4	8	8	8	8	5	5	5	1	4

```
ReadInput:
open file f "CHUOI.INP"
a.clear(); b.clear(); // 2 vector int a, b
int x, y
read(f, x) // hat dau tien
ins(a, x) // nạp x vào a
while true do
    y = -1;
    read(f, x, y) // đọc 2 hat x và y
    if y < 0:
        ins(a, x) // nạp x vào dãy a
        break while
    end if
    ins(b, x) // nạp x vào dãy b
    ins(a, y) // nạp y vào dãy a
end while
close(f);
// nối ngược b vào a
a = a + reverse(b)
end ReadInput
```

Thủ tục đọc dữ liệu cho ta chuỗi  $n$  hạt  $a$  với độ phức tạp  $O(n)$ .

Để đếm số màu ta dùng một mảng bool  $b$  đánh dấu các màu đã xuất hiện:  $b[i] = \text{true}$  cho biết màu  $i$  đã xuất hiện. Tổng các giá trị true (1) sẽ cho ta số màu trong chuỗi. Thủ tục này có độ phức tạp  $O(n)$ .

```
// Đếm số màu
Somau:
set b[30] all 0 // đánh dấu màu
for i = 0..n-1 do
    b[a[i]] = 1
end for
return sum(b)
end Somau
```

Để tìm điểm cắt với tổng chiều dài hai đầu lớn nhất ta thực hiện như sau. Trước hết ta định nghĩa điểm đổi màu trên chuỗi hạt là hạt (chỉ số) mà màu của nó khác với màu của hạt đứng sát nó (sát phải hay sát trái, tùy theo chiều duyệt xuôi từ trái qua phải hay duyệt ngược từ phải qua trái). Ta cũng định nghĩa một đoạn trong chuỗi hạt là một dãy liên tiếp các hạt cùng màu với chiều dài tối đa. Mỗi đoạn đều có điểm đầu và điểm cuối. Vì điểm cuối của mỗi đoạn chỉ lệch 1 đơn vị so với điểm đầu của đoạn tiếp theo, cho nên với mỗi đoạn ta chỉ cần quản lý một trong hai điểm: điểm đầu hoặc điểm cuối của đoạn đó. Ta chọn điểm đầu. Kỹ thuật này được gọi là *quản lý theo đoạn hoặc theo làn*.

chỉ số của a	0	1	2	3	4	5	6	7	8	9	10	11
a	4	7	4	8	8	8	8	5	5	5	1	4
điểm đầu đoạn d	0	1	2	3				7			10	11
số hiệu đoạn	0	1	2	3	4					5		6

$m = 7$  đoạn và các điểm đầu mỗi đoạn

Thủ tục tách đoạn duyệt chuỗi a một lần nên có độ phức tạp  $O(n)$ .

Ta gọi điểm cắt tạo ra tổng số hạt ở hai đầu lớn nhất là *điểm cắt tối ưu*. Dễ thấy điểm cắt tối ưu là một trong các điểm đầu đoạn. Như vậy ta duyệt lần lượt m điểm đầu đoạn. Với mỗi điểm đầu đoạn i ta tính tổng số hạt của hai đoạn sát trước đoạn i là i-1 và i-2. Nếu tổng này đạt trị max thì i-1 chính là điểm cắt tối ưu. Tổng này chính là:

$$t = d[i] - d[i-2]$$

Với đoạn đầu tiên, đoạn 0 và đoạn cuối cùng, đoạn m-1 có thể là hai đoạn cùng màu nên ta phải xử lý riêng như sau:

Nếu hạt cuối a[n-1] cùng màu với hạt đầu a[0] thì ta sửa lại đoạn cuối để thêm vào tổng số hạt cùng màu ở phần đầu chuỗi. Như vậy, ta sẽ thêm một đoạn nữa dùng làm lính canh cho tiện tính toán. Đoạn thêm vào sẽ có điểm đầu là n hoặc n + số hạt đầu chuỗi có cùng màu ở hạt cuối chuỗi. Thủ tục tách đoạn sẽ như sau:

```
// Tach a thanh cac doan cung mau
// 4 | 7 | 4 | 8 8 8 8 | 5 5 5 | 1 | 4
TachDoan:
d.clear(); // vector d ghi nhận đầu mỗi đoạn
m = a[n-1]; // m là màu của hạt cuối cùng
// xác định đoạn đầu tiên
for i = 0..n-1 do
    if (a[i] != m)
        ins(d,i) // đoạn số 0 đầu tiên
        break;
    end if
end for
for i = i+1..n-1 do
    if (a[i] != a[i-1])
        ins(d,i) // thêm đoạn mới
    end if
end for
// them doan cuoi lam linh canh
```

```
m = (a[0] == a[n-1]) ? n + d[0] : n;
ins(d, m)
```

Đến đây thì việc xác định điểm cắt tối ưu trở thành dễ dàng.

Trước hết ta duyệt các điểm đầu các đoạn  $i = 3..m-1$  từ  $d[3]$  đến  $d[m-1]$  để xác định điểm cắt tối ưu trong số các điểm  $d[i-1]..d[m-2]$ :

```
for i = 2..m-1 do
    if maxlen < d[i]-d[i-2]
        maxlen = d[i]-d[i-2]
        diemCat = d[i-1];
    end if
end for
```

Cuối cùng, ta xét riêng điểm cắt  $d[0]$ :

```
t = (d[1] - d[0]) + (d[m-1] - d[m-2]);
if (maxlen < t)
    maxlen = t;
    diemCat = d[0];
end if
```

## Chương trình C++

```
// Chuoi hat
#include <bits/stdc++.h>

using namespace std;

typedef vector<int> VI;

VI a, b, d;

int n; // so hat
int mau; // so mau
int m; // so doan
int maxlen;
int diemCat;

void Go() {
    cout << " ? " ;
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

Print(VI v, const char * msg) {
    cout << msg;
    for (int i = 0; i < v.size(); ++i) {
        cout << " " << v[i];
    }
}
```

```

Print(VI v, int diemcat, const char * msg) {
    cout << msg;
    for (int i = 0; i < diemcat; ++i)
        cout << " " << v[i];
    cout << " | ";
    for (int i = diemcat; i < v.size(); ++i)
        cout << " " << v[i];
}

// Dem so mau O(n)
int SoMau() {
    bool mau[30]; // danh dau cac mau
    memset(mau, false, sizeof(mau));
    for (int i = 0; i < n; ++i) {
        mau[a[i]] = true;
    }
    int c = 0;
    for (int i = 0; i < n; ++i) {
        c += mau[i];
    }
    return c;
}

// Tach a thanh cac doan cung mau
// 4 | 7 | 4 | 8 8 8 8 | 5 5 5 | 1 | 4
void TachDoan() {
    d.clear(); // ghi nhan dau moi doan
    m = a[n-1]; // mau cua hat cuoi cung
    // xac dinh doan dau tien
    int i;
    for (i = 0; i < n; ++i) {
        if (a[i] != m) {
            d.push_back(i);
            break;
        }
    }

    for (++i; i < n; ++i) {
        if (a[i] != a[i-1]) {
            // tao them doan moi
            d.push_back(i); // diem dau cua doan tiep theo
        }
    }
    // them doan cuoi
    m = (a[0] == a[n-1]) ? n + d[0] : n;
    d.push_back(m);
    m = d.size();
    cout << "\n Tong cong " << m << " doan";
    Print(d, "\n Diem dau cua cac doan: ");
}

// Xac dinh diem cat toi uu

```

```

void DiemCat() {
    maxlen = 0;
    diemCat = 0;
    int t; // tong cac hat
    for (int i = 2; i < m; ++i) {
        t = d[i]-d[i-2];
        if (maxlen < t) {
            maxlen = t;
            diemCat = d[i-1];
        }
    }
    // xet them diem cat d[0]
    t = (d[1] - d[0]) + (d[m-1] - d[m-2]);
    if (maxlen < t) {
        maxlen = t;
        diemCat = d[0];
    }
    int x, y = diemCat;
    x = (y == 0) ? n-1 : y-1;
    cout << "\n Cat giua hat " << x << " va hat " << y
        << " maxlen = " << maxlen;
    Print(a, diemCat, "\n a: ");
}

void ReadInput() {
    ifstream f("CHUOI.INP");
    a.clear(); b.clear();
    int x, y;
    f >> x; // hat dau tien
    a.push_back(x);
    while (true) {
        y = -1;
        f >> x >> y; // 2 hat x va y
        if (y < 0) {
            a.push_back(x);
            break;
        }
        b.push_back(x);
        a.push_back(y);
    }
    f.close();
    // noi nguoc b vao a
    for (int i = b.size()-1; i >= 0; --i)
        a.push_back(b[i]);
    n = a.size();
}

void ChuoiHat() {
    ReadInput();
    Print(a, "\n a: ");
    mau = SoMau();
    cout << "\n So mau: " << mau;
    if (mau == 1) {

```



```

        cout << "\n Chuoi hat dong mau. Chon diem cat tuy y: "
            << " maxlen = " << n;
        return;
    }
    TachDoan();
    DiemCat();
}

main() {
    ChuoiHat();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

a:  4 7 4 8 8 8 8 5 5 5 1 4
So mau: 5
Tong cong 7 doan
Diem dau cua cac doan:  1 2 3 7 10 11 13
Cat giua hat 6 va hat 7 maxlen = 7
a:  4 7 4 8 8 8 8 |  5 5 5 1 4
T h e   E n d

```

## Bài 4.4. Sắp mảng rồi ghi tệp

*Sinh ngẫu nhiên  $n$  phần tử cho mảng nguyên  $a$ . Sắp  $a$  theo trật tự tăng dần rồi ghi vào một tệp văn bản có tên tùy đặt.*

### Gợi ý

Bạn đọc tham khảo các bài trong Chương 2 về các thuật toán sinh ngẫu nhiên và sắp mảng theo các tiêu chí. Chương trình trong mục này giới thiệu hai phương án. Phương án thứ nhất vận dụng các thuật toán sort có sẵn. Phương án thứ hai giới thiệu ba thủ tục sắp mảng phổ biến là MinSort với độ phức tạp  $O(n^2)$ , BubbleSort với độ phức tạp  $O(n^2)$  và QuickSort với độ phức tạp  $O(n \log(n))$ .

### Sort theo thuật toán có sẵn

Muốn sắp một vector `int a` trong C++, bạn gọi thủ tục

```

sort(a.begin(), a.end()); // sắp tăng
sort(a.begin(), a.end(), greater<int>()); // sắp giảm

```

### Chương trình C++

```

/*****
Sort theo thuật toán có sẵn:
1. Sinh ngẫu nhiên  $n$  số nguyên cho vector  $a$ 
2. Sắp tăng  $a$ 

```

```

3. Ghi file SORT.DAT
*****/

#include <bits/stdc++.h>

using namespace std;

typedef vector<int> VI;

VI a;

// sinh ngẫu nhiên n số
// trong khoảng -50:50
// Sort, ghi mảng a
void Gen(int n) {
    srand(time(0));
    int M = 50;
    int M2 = 2 * M + 1;
    a.clear();
    for (int i = 0; i < n; ++i)
        a.push_back((rand() % M2) - M);
    sort(a.begin(), a.end());
    // Ghi file mỗi dòng 1 số
    ofstream f("SORT.DAT");
    for (int i = 0; i < n; ++i) {
        f << a[i] << endl;
    }
    f.close();
}

main() {
    Gen(30);
    cout << "\n T h e   E n d";
    return 0;
}

```

### Nội dung file SORT.DAT

```

-50
-48
-46
-45
-42
-41
-37
-36
-29
-25
-23
-23
-20
-3
0
1

```

```

6
14
15
15
16
16
28
31
32
36
39
46
46
48

```

## Chương trình Python

```

"""
*****
Sort theo thuật toán cơ bản:
1. Sinh ngẫu nhiên n số nguyên cho mảng a
2. Sắp xếp mảng a
3. Ghi file SORT.DAT
*****
"""

a = []
n = 0

from random import randint

# sinh ngẫu nhiên n số
# trong khoảng -50..50
# ghi mảng a
def Gen(n):
    global a
    a = []
    for i in range(n):
        a.append(randint(-50,50))
    a.sort()
    with open('SORT.DAT','w',encoding = 'utf-8') as f:
        for i in range(n):
            f.write(str(a[i])+'\n')
    return n

n = Gen(30)

```

## Sort theo thuật toán tự làm

Theo phương pháp MinSort với mỗi phần tử  $i: 0..n-1$  ta tìm phần tử nhỏ nhất  $a[j]$  trong hậu tố  $a[i..n-1]$  sau đó ta đổi chỗ phần tử này với phần tử  $a[i]$ . Như vậy mảng được chia thành hai đoạn: đoạn trái là tiền tố,  $a[1..i]$  được sắp tăng, còn đoạn phải  $a[i+1..n-1]$  chưa xử lí. Mỗi bước ta thu hẹp đoạn phải cho đến khi còn một phần tử là xong.

```
// sắp tăng đoạn a[d..c] O(n^2)
MinSort(int a[d..c]):
  for i = d..c do
    a[j] = min(a[i..c])
    Swap([i], a[j]);
  end for
end MinSort
```

Để hiểu được sắp nổi bọt BubbleSort, bạn mừng tượng là ta nhúng mảng a vào bể nước theo chiều dọc từ trên xuống, sau đó kiểm tra các cặp phần tử cạnh nhau tính từ dưới lên. Nếu phần tử dưới nhẹ hơn phần tử đứng trên thì ta cho phần tử dưới nổi lên một mức thông qua thao tác đổi chỗ hai phần tử đó.

```
BubbleSort(int a[d..c]): // O(n^2)
  for i = d..c do
    for j = c:i do
      if (a[j] < a[j-1])
        Swap(a[j], a[j-1])
      end if
    end for j
  end for i
end BubbleSort
```

QuickSort được triển khai theo phương pháp chia để trị. Pha đầu tiên của QuickSort là san các phần tử của mảng a thành ba khúc: khúc đầu gồm những phần tử nhỏ hơn hoặc bằng mẫu m, khúc cuối gồm những phần tử lớn hơn hoặc bằng mẫu m, các phần tử còn lại nằm ở giữa. Pha tiếp theo sẽ lặp lại việc chia ba các khúc đầu và khúc cuối, nếu mỗi khúc có trên một phần tử.

```
// O(nlog(n))
QuickSort(int a[d..c]):
  m = a[(d+c)/2] // mẫu
  i = d; j = c
  while (i <= j) do
    while(a[i] < m) do ++i end while
    while(a[j] > m) do --j end while
    if (i <= j)
      Swap(a[i], a[j]);
      ++i; --j;
    end if
  end while
  if (d < j) QuickSort(a[d..j]) end if
  if (i < c) QuickSort(a[i..c]) end if
```

```
end QuickSort
```

Các chương trình dưới đây gọi thêm các thủ tục sort có sẵn của hệ thống để bạn đọc tự so sánh với các thủ tục tự làm.

## Chương trình C++

```

/*****
Sort
1. Sinh ngẫu nhiên n số nguyên cho mảng a
2. Sắp tang a
3. Ghi file SORT.DAT
*****/
#include <bits/stdc++.h>

using namespace std;

const int MN = 4000;
int a[MN];
int n;

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Hien thi a[d..c]
void Print(int x[], int d, int c, const char *msg = "") {
    cout << msg;
    for(int i = d; i <= c; ++i)
        cout << " " << x[i];
}

// hoan vi a[i] va a[j]
void Swap(int a[], int i, int j) {
    int x = a[i];
    a[i] = a[j]; a[j] = x;
}

// sap tang doan a[d..c]
// O(n^2)
void MinSort(int a[], int d, int c) {
    for (int i = d; i <= c; ++i) {
        int j = i;
        for (int k = i+1; k <= c; ++k) {
            if (a[k] < a[j]) j = k;
        } // for k
        // a[j] = min(a[i..c])
        Swap(a, i, j);
    } // for i
}

```

```

// Sáp nổi bọt:  $O(n^2)$ 
void BubbleSort(int a[], int d, int c) {
    for (int i = d; i <= c; ++i) {
        for (int j = c; j > i; --j) {
            if (a[j] < a[j-1])
                Swap(a, j, j-1);
        }
    }
}

//  $O(n\log(n))$ 
void QuickSort(int a[], int d, int c) {
    int i = d, j = c, m = a[(d+c)/2];
    while (i <= j) {
        while(a[i] < m) ++i;
        while(a[j] > m) --j;
        if (i <= j) {
            Swap(a, i++, j--);
        }
    }
    if (d < j) QuickSort(a, d, j);
    if (i < c) QuickSort(a, i, c);
}

// sinh ngẫu nhiên n số v
// trong khoảng -50..50
// ghi mảng a
int Gen(int a[], int n) {
    srand(time(0));
    int M = 50;
    int M2 = 2*M + 1;
    for (int i = 0; i < n; ++i) {
        a[i] = (rand() % M2) - M;
    }
}

void GhiFile() {
    ofstream f("SORT.DAT");
    for (int i = 0; i < n; ++i) {
        f << a[i] << endl;
    }
    f.close();
}

void Run() {
    n = Gen(a, 30);
    Print(a, 0, n-1, "\n Init a:");
    // sort(a, a + n); // Hàm có sẵn
    // MinSort(a, 0, n-1); // tu làm
    // BubbleSort(a, 0, n-1); // tu làm
    QuickSort(a, 0, n-1); // tu làm
    Print(a, 0, n-1, "\n Sorted a:");
}

```

```

    GhiFile();
}

main() {
    Run();
    cout << "\n T h e    E n d";
    return 0;
}

```

## Bài 4.5. abc - sắp theo chỉ dẫn

Cho xâu  $S$  gồm  $N$  ký tự tạo từ các chữ cái 'a'..'z'. Ta gọi  $S$  là xâu mẫu. Từ xâu mẫu  $S$  này người ta tạo ra  $N$  xâu thứ cấp bằng cách dịch xâu  $S$  qua trái  $i$  vị trí theo dạng vòng tròn, tức là  $i$  ký tự đầu xâu lần lượt được chuyển về cuối xâu,  $i = 0, 1, \dots, N - 1$ . Như vậy xâu thứ cấp với  $i = 0$  sẽ trùng với xâu mẫu  $S$ . Giả sử ta đã sắp tăng  $N$  xâu thu được theo trật tự từ điển. Hãy tìm xâu thứ  $k$  trong dãy.

Dữ liệu vào: tệp văn bản `abc.inp` có cấu trúc như sau:

- Dòng thứ nhất chứa hai số tự nhiên  $N$  và  $k$  cách nhau qua dấu cách,  $6 \leq N \leq 500$ ,  $1 \leq k \leq N$ .  $N$  cho biết chiều dài xâu  $S$ ,  $k$  cho biết vị trí của xâu thứ cấp trong dãy được sắp tăng theo thứ tự từ điển.
- Dòng thứ hai: xâu mẫu  $S$ .

Dữ liệu ra: tệp văn bản `abc.out` gồm một dòng chứa xâu thứ  $k$  trong dãy được sắp.

### Ví dụ

<code>abc.inp</code>	<code>abc.out</code>
6 3 dabdec	cdabde

### Thuật toán

Ta lưu ý đến chỉ số của các đối tượng. Xâu văn bản  $s$  có chỉ số từ 0 đến  $N-1$ , trong đó  $N$  là chiều dài của  $s$ , trong khi các xâu mẫu được sắp tăng và được gán chỉ số từ 1 đến  $N$ .

Ta gọi xâu  $s$  ban đầu là xâu mẫu, các xâu được sinh ra bởi phép quay là xâu thứ cấp. Để ý rằng các xâu mẫu cũng là một xâu thứ cấp ứng với phép quay 0 vị trí. Ta có thể nhận được xâu thứ cấp thứ  $i$  bằng cách duyệt xâu mẫu theo vòng tròn kể từ vị trí thứ  $i$  về cuối, đến vị trí thứ  $n$ . Sau đó duyệt tiếp tục từ vị trí thứ 1 đến vị trí thứ  $i - 1$ . Ta kí hiệu xâu thứ cấp thứ  $i$  là  $[i]$ . Thí dụ, nếu xâu mẫu  $s = \text{'dabdec'}$  thì xâu thứ cấp thứ 2 sẽ là  $[2] = \text{'abdec d'}$ . Để tìm xâu thứ  $k$  trong dãy được sắp, trước hết ta cần sắp tăng các xâu đó theo trật tự từ điển sau đó lấy xâu thứ  $k$  trong dãy được sắp làm kết quả.

		Xâu thứ cấp	Sắp tăng	Sắp tăng theo chỉ dẫn id
①	[1] = S	dabdec	abdec d	[2]
②	[2]	abdec d	bdec d a	[3]
③	[3]	bdec d a	<b>cdabde</b>	[6]
④	[4]	dec d ab	dabdec	[1]
⑤	[5]	ec d ab d	dec d ab	[4]
⑥	[6]	cdabde	ec d ab d	[5]

Xâu mẫu  $s = \text{"dabdec"}$

Sắp chỉ dẫn các xâu thứ cấp


Để sắp tăng được các xâu này mà không phải sinh ra các xâu mới ta dùng một mảng phụ  $id$  gọi là mảng chỉ dẫn. Trước khi sắp ta gán  $id[i] := i$ . Sau khi sắp thì  $id[i]$  sẽ cho biết tại vị trí thứ  $i$  trong dãy được sắp là xâu thứ cấp nào. Trong thí dụ trên,  $id[3] = 6$  là xâu thứ cấp [6]. Giá trị 3 cho biết cần tìm xâu thứ  $k = 3$  trong dãy sắp tăng các xâu thứ cấp. Giá trị 6 cho biết cần tìm là xâu thứ 6 trong dãy các xâu thứ cấp được sinh ra lúc đầu, tức là lúc chưa sắp.


Tóm lại, bài toán quy về tìm xâu thứ  $k$  trong dãy sắp tăng các xâu quay nhận được từ xâu mẫu. Thuật toán *QuickSort* sắp nhanh các xâu thứ cấp do *Hoare* đề xuất có độ phức tạp  $N \log_2 N$ . Thuật toán này đòi hỏi cài đặt hàm  $Less(i, j)$  so sánh hai xâu thứ cấp  $[i]$  và  $[j]$  theo thứ tự từ điển và cho giá trị true nếu xâu thứ cấp  $[i]$  nhỏ hơn xâu thứ cấp  $[j]$ , ngược lại, hàm cho ra giá trị false. Để so sánh hai xâu theo trật tự từ điển ta lần lượt duyệt từng cặp ký tự của mỗi xâu. Nếu hai xâu giống nhau tại mọi vị trí thì ta gán true cho hàm  $Less$ . Ngược lại, nếu tìm được vị trí khác nhau đầu tiên thì ta so sánh hai ký tự  $s[i]$  và  $s[j]$  tại vị trí đó và gán cho hàm  $Less$  giá trị true nếu  $s[i] < s[j]$ , ngược lại, tức là khi  $s[i] \geq s[j]$  thì gán giá trị false cho hàm  $Less$ .

Ta chỉ cần lưu ý là việc duyệt xâu phải thực hiện trên vòng tròn theo chiều quay của kim đồng hồ.

```
// [i] < [j] ?
Less(i, j):
  for k = 0:n do
    if (s[i] ≠ s[j]) return (s[i] < s[j]) end if
    ++i; if (i == n) i = 0 end if
    ++j; if (j == n) j = 0 end if
  end for
  return false
end Less
```

## Phương án 1

 Sắp tăng xâu mẫu  $s$  theo chỉ dẫn  $id$

 Tạo xâu thứ cấp thứ  $k-1$  trong dãy được sắp  $id$

Để xuất xâu thứ cấp thứ  $k$  ta đơn giản duyệt  $s$  theo vòng tròn hai pha: pha thứ nhất duyệt từ  $k$  đến  $n-1$ ; pha thứ hai duyệt từ  $0$  đến  $k-1$ .



```
// Xuất string vòng [k]
string GetStr(k):
    return s[k:n] + s[0:k]
end GetStr
```

$s[i:j]$  cho ra string con từ string  $s[i]$  đến  $s[j-1]$ .

```
PhuongAn1:
    sort(id, id+n, Less);
    GetStr(id[k-1]);
end PhuongAn1
```

## Phương án 2

Hoare cũng cung cấp thêm thuật toán tìm phần tử thứ  $k$  trong dãy được sắp với độ phức tạp  $O(N)$ . Ta vận dụng thuật toán này cho bài toán *abc*. Bản chất thuật toán này là như sau. Ta cũng sắp tăng các xâu thứ cấp theo thuật toán QuickSort nhưng không sắp hết mà chỉ quan tâm đến đoạn dữ liệu trong mảng có chứa phần tử thứ  $k$ . Lưu ý rằng sau một lần chia dữ liệu của đoạn  $id[d..c]$  ta thu được ba đoạn: đoạn  $id[d..j]$ ,  $id[j+1..i-1]$  và  $id[i..c]$ , trong đó đoạn giữa là  $id[j+1..i-1]$  đã được sắp. Nếu  $k$  rơi vào đoạn thứ nhất là  $id[d..j]$  hoặc đoạn thứ ba là  $id[i..c]$  thì ta chỉ cần sắp tiếp đoạn đó. Hàm Find( $k$ ) cho ra vị trí gốc của xâu thứ cấp sẽ đứng thứ  $k$  trong dãy đã sắp. Trong thí dụ trên Find(3)=6.

```
// Tìm phần tử thu k
Find(k):
    d = 0, c = n-1;
    while (d <= c) do
        i = d; j = c;
        m = id[(i+j) / 2]; // phần tử giữa
        while (i <= j) do
            while (Less(id[i], m)) ++i end while
            while (Less(m, id[j])) --j end while
            if (i <= j)
                Swap(s[id[i]], s[id[j]])
                ++i; --j;
            end if
        end while
        // d---j....k?...m....i---c
        if (j < k) d = i end if
        if (k < i) c = j end if
    end while
    return id[k];
end Find
```

```
PhuongAn2:
    GetStr(Find(k-1));
end PhuongAn2
```

## Chương trình C++

```
// abc
#include <bits/stdc++.h>

using namespace std;
string s;
int n, k;
int *id;

// Hien thi string vong s(i)
Print(int v) {
    cout << endl;
    for (int i = v; i < n; ++i)
        cout << s[i];
    for (int i = 0; i < v; ++i)
        cout << s[i];
}

// Xuat string vong s(i)
string GetStr(int v) {
    string w = "";
    for (int i = v; i < n; ++i)
        w += s[i];
    for (int i = 0; i < v; ++i)
        w += s[i];
    return w;
}

// s(i) < s(j)
bool Less(int i, int j) {
    for (int k = 0; k < n; ++k) {
        if (s[i] != s[j]) return (s[i] < s[j]);
        ++i; if (i == n) i = 0;
        ++j; if (j == n) j = 0;
    }
    return false;
}

int Swap(int a[], int i, int j) {
    int x = a[i];
    a[i] = a[j]; a[j] = x;
}

// Timphan tu thu k
int Find(int k) {
    int d = 0, c = n-1;
    int i, j, m;
    while (d <= c) {
        i = d; j = c;
        m = id[(i+j) / 2]; //phan tu giua
        while (i <= j) {
            while (Less(id[i], m)) ++i;
```

```

        while (Less(m, id[j])) --j;
        if (i <= j)
            Swap(id, i++, j--);
    }
    // d---j....k?...m....i---c
    if (j < k) d = i;
    if (k < i) c = j;
}
return id[k];
}

// sort
void PhuongAn1() {
    sort(id, id+n, Less);
    ofstream g("ABC.OUT");
    g << GetStr(id[k-1]);
    g.close();
}

// Tim phan tu thu k
void PhuongAn2() {
    ofstream g("ABC.OUT");
    g << GetStr(Find(k-1));
    g.close();
}

void Run() {
    ifstream f("ABC.INP");
    char c;
    f >> n >> k;
    getline(f, s, '\n'); // xuong dong moi
    getline(f, s, '\n');
    f.close();
    cout << "\n n = " << n << " k = " << k;
    cout << "\n s = |" << s << "|";
    id = new int[n];
    for (int i = 0; i < n; ++i)
        id[i] = i;
    PhuongAn1();
    //PhuongAn2();
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Bài 4.6. Xâu mẫu

Một tệp văn bản  $f$  có tên  $STRINGS.INP$  chứa các xâu ký tự, mỗi dòng ghi một xâu có chiều dài tối đa 250 ký tự. Xâu đầu tiên được gọi là xâu mẫu  $s$ . Lập trình:

Đọc xâu mẫu  $s$  từ tệp  $f$ , ghi vào tệp văn bản  $g$  có tên  $STRINGS.OUT$ . Sau đó đọc từng xâu  $x$  còn lại của  $f$ , với mỗi xâu  $x$  cần ghi vào  $g$  các thông tin sau:

- nội dung xâu  $x$ ;
  - hai số  $v$  và  $d$  cách nhau qua dấu cách, trong đó  $v$  là vị trí xuất hiện và  $d$  là chiều dài lớn nhất của khúc đầu của  $x$  trong xâu mẫu  $s$ . Nếu vô nghiệm thì ghi -1.
- Các chỉ số được tính từ 0.

### Ví dụ

STRINGS.INP	STRINGS.OUT
cabxabc dab	cabxabc dab
abcd	abcd
ycd	4 4
cdaeh	ycd
	-1
	cdaeh
	6 3

### Thuật toán

Với mỗi xâu ký tự  $w$  ta kí hiệu  $w[i..j]$ ,  $i \leq j$ , và gọi là đoạn, là xâu gồm dãy ký tự liên tiếp từ  $w[i]$  đến  $w[j]$  trong xâu  $w$ . Thí dụ, nếu  $w = \text{'cabxabc dab'}$  thì  $w[4..7] = \text{'abcd'}$ . Gọi  $s$  là xâu mẫu,  $w$  là xâu cần khảo sát. Nhiệm vụ của ta là tìm vị trí  $v$  và chiều dài lớn nhất  $d$  để  $w[0..d-1] = s[v..(v+d-1)]$ . Ta vận dụng kĩ thuật tổ chức hậu tố như sau.

Hậu tố của một xâu là đoạn cuối của xâu đó. Như vậy một xâu có chiều dài  $n$  sẽ có  $n$  hậu tố. Ta kí hiệu  $s(i)$  là hậu tố của  $s$  tính từ vị trí  $i$  về cuối. Ví dụ, với xâu mẫu  $s[0..9] = \text{'cabxabc dab'}$  ta có 10 hậu tố sau đây:

$s(0) = s[0..9] = \text{'cabxabc dab'}$   
 $s(1) = s[1..9] = \text{'abxabc dab'}$   
 $s(2) = s[2..9] = \text{'bxabc dab'}$   
 $s(3) = s[3..9] = \text{'xabc dab'}$   
 $s(4) = s[4..9] = \text{'abc dab'}$   
 $s(5) = s[5..9] = \text{'bc dab'}$   
 $s(6) = s[6..9] = \text{'c dab'}$   
 $s(7) = s[7..9] = \text{'d ab'}$   
 $s(8) = s[8..9] = \text{'a b'}$   
 $s(9) = s[9..9] = \text{'b'}$

Như vậy, hậu tố sau sẽ nhận được từ hậu tố sát trước nó bằng cách bỏ đi ký tự đầu tiên.

Trước hết ta sắp tăng các hậu tố của xâu mẫu  $s$  theo trật tự từ điển. Sử dụng một mảng chỉ dẫn  $id$ , trong đó  $id[i]$  trỏ đến vị trí đầu tiên của hậu tố trong xâu mẫu. Cụ thể là,

nếu  $id[i] = k$  thì hậu tố tương ứng sẽ là  $s[k:n]$ . Sau khi sắp tăng các hậu tố của xâu mẫu  $s[0..9] = 'cabxabc dab'$  ta thu được:

xâu mẫu: cabxabc dab			
i	id[i]	Hậu tố	
0	8	$S[8..9]$	ab
1	4	$S[4..9]$	abc dab
2	1	$S[1..9]$	abxabc dab
3	9	$S[9..9]$	b
4	5	$S[5..9]$	bcdab
5	2	$S[2..9]$	bxabc dab
6	0	$S[0..9]$	cabxabc dab
7	6	$S[6..9]$	cdab
8	7	$S[7..9]$	dab
9	3	$S[3..9]$	xabc dab

Sắp tăng theo chỉ dẫn các hậu tố của xâu  
 $s[0..9] = 'cabxabc dab'$

Các hậu tố đều có chiều dài khác nhau do đó hàm  $Less(i,j)$  cho biết  $s(i) < s(j)$  sẽ thao tác như sau:

Việc còn lại là so sánh xâu  $w$  với các hậu tố  $s(i)$  để tìm khúc đầu chung dài nhất giữa chúng. Ví dụ, với  $w[0..3] = 'abcd'$  thì khúc đầu chung dài nhất tìm được với hậu tố  $s[4..9]$  do  $id[2]$  trở tới. Vị trí  $v$  tìm được sẽ là 4 và chiều dài lớn nhất  $d$  sẽ là 4.

Để tìm khúc đầu chung dài nhất giữa xâu  $w$  và hậu tố  $s(i)$  ta duyệt lần lượt từng cặp kí tự của  $w$  và hậu tố  $s(i)$  cho đến khi gặp hai kí tự khác nhau hoặc hết một trong hai xâu.

Với mỗi xâu  $w$ , trước hết ta gọi hàm Search tìm kiếm nhị phân kí tự đầu tiên  $w[0]$  trong dãy sắp tăng các hậu tố  $s(i)$ . Nếu  $w[0]$  không xuất hiện thì ta ghi kết quả -1. Ngược lại, nếu  $w[0]$  xuất hiện tại hậu tố thứ  $v$  thì ta duyệt lần lượt các hậu tố từ hậu tố  $v$  trở đi để xác định chiều dài lớn nhất của tiền tố  $w$  trong các hậu tố  $s$ .

## Chương trình C++

```
// Xâu mẫu
#include <bits/stdc++.h>

using namespace std;

string s, w;
```

```

int *id;
int n; // len(s)

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

void Print(int x[], int d, int c, const char * msg="") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << x[i];
    }
}

// Hien thi hau to s(i)
void Print(int i, const char * msg="") {
    cout << msg;
    for (int j = i; j < n; ++j) {
        cout << s[j];
    }
}

// So sanh hai hau to s(i) va s(j)
// s(i) < s(j) ?
bool Less(int i, int j) {
    if (i == j) return false; // hai xau la mot
    // xet doan ngan
    for (int v = max(i,j); v < n; ++v) {
        if (s[i] != s[j]) return (s[i] < s[j]);
        ++i; ++j;
    }
    return i > j;
}

// vi tri xuat hien cua char ch
// dau day cua day sap tang cac hau to cua s
int Search(char ch) {
    int d = 0, c = n-1, m;
    while(d < c) {
        m = (d + c) / 2;
        if (s[id[m]] < ch) d = m+1;
        else c = m;
    } // d == c
    return (s[id[d]] == ch) ? d : -1;
}

// chieu dai max cua w voi hau to s(i)
int MaxPrefix(int i) {
    if (w[0] != s[i]) return 0;
    // w[0] == s[i]

```

```

    int k = 1;
    int lenw = w.length();
    for (++i; i < n; ++i) {
        if (s[i] != w[k]) break;
        // s[i] == w[k]
        ++k;
        if (k == lenw) break;
    }
    return k;
}

// chiều dài max của w trong day
// các hậu tố kế từ id[v]..id[n-1]
vector<int> MaxLen(int v) {
    int maxlen = 0;
    int len;
    int p = 0;
    for (int i = v; i < n; ++i) {
        // cout << "\n so " << w << " và "; Print(id[i]);
        len = MaxPrefix(id[i]);
        if (len == 0) break;
        if (len > maxlen) {
            p = id[i];
            maxlen = len;
        }
    }
    // cout << ": " << maxlen;
    vector<int> plen;
    plen.push_back(p);
    plen.push_back(maxlen);
    return plen;
    // plen[0] = vị trí xuất hiện, plen[1] = chiều dài max
}

void XauMau() {
    ifstream f("STRINGS.INP");
    ofstream g("STRINGS.OUT");
    getline(f,s); // đọc xâu mau
    n = s.length();
    cout << "\n Xâu mau: " << s << " len = " << n;
    g << s << endl; // ghi xâu mau
    id = new int[n]; // cấp phát mảng id
    // khởi tạo index id
    for (int i = 0; i < n; ++i) {
        id[i] = i;
    }
    sort(id,id+n,Less); // sort theo chỉ số
    cout << "\n Day hau to sap tang: ";
    for (int i = 0; i < n; ++i) {
        cout << "\n " << id[i] << ". "; Print(id[i]);
    }
    cout << "\n -----";
    // Đọc các xâu w

```

```

cout << "\n Xet cac doan:\n";
int v, maxlen, len;
while (true) {
    getline(f,w); // doc xau w
    if (w == "") break;
    cout << "\n  . " << w;
    g << w << endl;    // ghi xau w
    v = Search(w[0]);
    if (v == -1) { // ko tim thay w[0]
        g << v << endl; // ghi ket qua -1
        continue;
    }
    cout << " Search " << w[0] << " at id[" << v << "] = " <<
id[v];
    vector<int> plen = MaxLen(v);
    // cout << "\n " << plen[0] << " " << plen[1];
    g << plen[0] << " " << plen[1] << endl;
}
f.close(); g.close();
}

main() {
    XauMau();
    cout << "\n T h e   E n d";
    return 0;
}

```



## CHƯƠNG 5

### PHƯƠNG PHÁP THAM LAM

Phương pháp tham lam gợi ý chúng ta tìm một trật tự hợp lý để duyệt dữ liệu nhằm đạt được mục tiêu một cách chắc chắn và nhanh chóng. Thông thường, dữ liệu được duyệt theo một trong hai trật tự là tăng hoặc giảm dần theo một chỉ tiêu nào đó. Một số bài toán đòi hỏi những dạng thức cải biên của hai dạng nói trên.

#### Bài 5.1. Băng nhạc

*Người ta cần ghi  $N$  bài hát, được mã số từ 1 đến  $N$ , vào một băng nhạc có thời lượng tính theo phút đủ chứa toàn bộ các bài đã cho. Với mỗi bài hát ta biết thời lượng phát của bài đó. Băng sẽ được lắp vào một máy phát nhạc đặt trong một siêu thị. Khách hàng muốn nghe bài hát nào chỉ việc nhấn phím ứng với bài đó. Để tìm và phát bài thứ  $i$  trên băng, máy xuất phát từ đầu cuộn băng, quay băng để bỏ qua  $i - 1$  bài ghi trước bài đó. Thời gian quay băng bỏ qua mỗi bài và thời gian phát bài đó được tính là như nhau. Tính trung bình, các bài hát trong một ngày được khách hàng lựa chọn với số lần (tần suất) như nhau. Hãy tìm cách ghi các bài trên băng sao cho tổng thời gian quay băng trong mỗi ngày là ít nhất.*

Dữ liệu vào được ghi trong tệp văn bản tên BANGNHAC.INP.

- Dòng đầu tiên là số tự nhiên  $N$  cho biết số lượng bài hát.
- Tiếp đến là  $N$  số nguyên dương thể hiện dung lượng tính theo phút của mỗi bài. Mỗi đơn vị dữ liệu cách nhau qua dấu cách.

#### Ví dụ

Có $N = 3$ bài hát:	BANGNHAC.INP	BANGNHAC.OUT
- Bài 1 phát trong thời gian 7 phút.	3	2 2
- Bài 2 phát trong thời gian 2 phút.	7 2 3	3 5
- Bài 3 phát trong thời gian 3 phút.		1 12
		19

Dữ liệu ra được ghi trong tệp văn bản BANGNHAC.OUT theo dạng thức sau:

- $N$  dòng đầu tiên thể hiện trật tự ghi bài hát trên băng: mỗi dòng gồm hai số nguyên dương  $j$  và  $d$  cách nhau bởi dấu cách, trong đó  $j$  là mã số của bài hát cần ghi,  $d$  là thời gian tìm và phát bài đó theo trật tự ghi này.
- Dòng thứ  $n + 1$  ghi tổng số thời gian quay băng nếu mỗi bài hát được phát một lần trong ngày.

Với ví dụ trên, kết quả thu được sẽ như sau:

- Cần ghi lần lượt trên băng các bài theo trật tự : bài 2, bài 3, bài 1;
- Để tìm và phát bài 2 cần 2 phút;
- Để tìm và phát bài 3 cần 5 phút;
- Để tìm và phát bài 1 cần 12 phút;
- Tổng thời gian để tìm và phát mỗi bài một lần là: 19 phút.

## Thuật toán

Giả sử ta có ba bài hát với số phút lần lượt như sau:

Mã số bài hát	①	②	③
Thời gian quay băng	7	2	3

Ta xét vài tình huống ghi băng để rút ra kết luận cần thiết.

Trật tự ghi trên băng	Thời gian tìm và phát
$(x, y, z)$	$t(x)+t(y)+t(z); t(i)$
①②③	$(7)+(7+2)+(7+2+3) = 28'$
①③②	$(7)+(7+3)+(7+3+2) = 29'$
②①③	$(2)+(2+7)+(2+7+3) = 23'$
②③①	$(2)+(2+3)+(2+3+7) = 19' \text{ (phương án tối ưu)}$
③①②	$(3)+(3+7)+(3+7+2) = 25'$
③②①	$(3)+(3+2)+(3+2+7) = 20'$

Vậy phương án tối ưu sẽ là ②③①: ghi bài 2 rồi đến bài 3, cuối cùng ghi bài 1. Tổng thời gian theo phương án này là 19 phút.

Để có phương án tối ưu ta chỉ cần ghi băng theo trật tự *tăng dần* của thời lượng. Bài toán được cho với giả thiết băng đủ lớn để ghi được toàn bộ các bài. Dễ dàng sửa chương trình để vận dụng cho trường hợp dung lượng băng hạn chế trong M phút. Chương trình sắp xếp dữ liệu theo chỉ dẫn.

## Chương trình C++

```
// Bang nhac
#include <bits/stdc++.h>

using namespace std;

int n;
int *a; // thoi luong moi bai
int *id;
int *t; // thoi luong tim va phat moi bai

// Hien thi mang a[d..c]
void Print(int a[], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i)
        cout << " " << a[i];
}

// so sanh id[i] va id[j]
bool Less(int i, int j) {
    return a[i] < a[j];
}

void ReadInput() {
```

```

ifstream f("BANGNHAC.INP");
f >> n;
cout << " So bai hat " << n << endl;
a = new int[n];
id = new int[n];
t = new int[n];
for (int i = 0; i < n; ++i) {
    f >> a[i];
    id[i] = i;
}
f.close();
Print(a, 0, n-1, "\n a:");
}

void WriteResult() {
    ofstream g("BANGNHAC.OUT");
    int sumt = 0;
    for (int i = 0; i < n; ++i) {
        g << id[i]+1 << " " << t[i] << endl;
        sumt += t[i];
    }
    g << sumt << endl;
    g.close();
}

void BangNhac() {
    ReadInput();
    sort(id, id+n, Less); // ghi bang
    // tinh thoi luong cho moi bai
    t[0] = a[id[0]]; // bai dau tien
    for (int i = 1; i < n; ++i) {
        t[i] = t[i-1] + a[id[i]];
    }
    WriteResult();
}

main() {
    BangNhac();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Bài 5.2. Xếp việc

*Có  $N$  công việc cần thực hiện trên một máy tính, mỗi việc đòi hỏi đúng 1 giờ máy. Với mỗi việc ta biết thời hạn phải nộp kết quả thực hiện sau khi hoàn thành việc đó và tiền thưởng thu được nếu nộp kết quả trước hoặc đúng thời điểm quy định. Chỉ có*

*một máy tính trong tay, hãy lập lịch thực hiện đủ  $N$  công việc trên máy tính sao cho tổng số tiền thưởng thu được là lớn nhất và thời gian hoạt động của máy là nhỏ nhất. Giả thiết rằng máy được khởi động vào đầu ca, thời điểm  $t = 0$  và chỉ tắt máy sau khi đã hoàn thành đủ  $N$  công việc.*

Dữ liệu vào: tệp văn bản `viiec.inp`:

- Dòng đầu tiên là số  $N$ .
- $N$  dòng tiếp theo: mỗi việc được mô tả bằng hai số tự nhiên, số thứ nhất là thời hạn giao nộp, số thứ hai là tiền thưởng. Các số cách nhau bởi dấu cách.

### Ví dụ

```
viiec.inp
4
1 15
3 10
5 100
1 27
```

**Ý nghĩa:** Cho biết có 4 việc với các thông tin sau:

- Việc thứ nhất phải nộp không muộn hơn thời điểm 1 (giờ) với tiền thưởng 15 V (đơn vị tiền tệ);
- Việc thứ hai phải nộp không muộn hơn thời điểm 3 (giờ) với tiền thưởng 10 V;
- Việc thứ ba phải nộp không muộn hơn thời điểm 5 (giờ) với tiền thưởng 100 V;
- Việc thứ tư phải nộp không muộn hơn thời điểm 1 (giờ) với tiền thưởng 27 V.

Dữ liệu ra: tệp văn bản `viiec.out`:

- $N$  dòng đầu tiên, dòng thứ  $t$  ghi một số tự nhiên  $i$  cho biết việc thứ  $i$  được làm trong giờ  $t$ .
- Dòng cuối cùng ghi tổng số tiền thu được.

Với ví dụ trên, tệp `viiec.out` sẽ như sau:

```
viiec.out
4
2
3
1
137
```

**Ý nghĩa:**

- Giờ thứ 1 thực hiện việc 4 và nộp đúng hạn nên được thưởng 27 V;
- Giờ thứ 2 thực hiện việc 2 và nộp trước hạn nên được thưởng 10 V;
- Giờ thứ 3 thực hiện việc 3 và nộp trước hạn nên được thưởng 100 V;
- Giờ thứ 4 thực hiện việc 1;
- Tổng tiền thưởng thu được do đã hoàn thành đúng hạn ba việc 4, 2 và 3 là  $27 + 10 + 100 = 137$  (V).

### Thuật toán

Ta ưu tiên cho những việc có tiền thưởng cao, do đó ta sắp các việc giảm dần theo tiền thưởng. Với mỗi việc  $v$  ta đã biết thời hạn giao nộp việc đó là  $i = t[v]$ . Ta xét trục thời gian  $h$ . Nếu giờ  $i$  trên trục đó đã bận do việc khác thì ta tìm từ thời điểm  $i$  trở về trước một thời điểm  $j$  có thể thực hiện được việc  $v$  đó. Nếu tìm được một thời điểm  $j$  như vậy, ta đánh dấu bằng mã số của việc đó trên trục thời gian  $h$ ,  $h[j] = v$ . Sau khi đã duyệt xong các việc, có thể trên trục thời gian còn những thời điểm rỗi, ta dồn các việc đã xếp về

phía trước nhằm thu được một lịch làm việc trù mật, tức là không có giờ trống. Cuối cùng ta xếp tiếp những việc trước đó đã xét nhưng không xếp được. Đây là những việc phải làm nhưng không thể nộp đúng hạn nên sẽ không có tiền thưởng. Với ví dụ đã cho,  $N = 4$ , thời hạn giao nộp  $t = (1, 3, 5, 1)$  và tiền thưởng  $a = (15, 10, 100, 27)$  ta tính toán như sau:

- Khởi trị: trục thời gian với 5 thời điểm ứng với  $T_{\max} = 5$  là thời điểm muộn nhất phải nộp kết quả,  $T_{\max} = \max \{ \text{thời hạn giao nộp} \}$

$i$	1	2	3	4	5
$h$	-1	-1	-1	-1	-1
thưởng					

*Khởi trị*

- Chọn việc 3 có tiền thưởng lớn nhất là 100. Xếp việc 3 với thời hạn  $t[3] = 5$  vào  $h$ :  $h[5] = 3$ .

$i$	0	1	2	3	4	5
$h$	-1	-1	-1	-1	-1	3
thưởng						100

*Xếp việc 3*

- Chọn tiếp việc 4 có tiền thưởng 27. Xếp việc 4 với thời hạn  $t[4] = 1$  vào  $h$ :  $h[1] = 4$ .

$i$	1	2	3	4	5
$h$	4	-1	-1	-1	3
thưởng	27				100

*Xếp việc 4*

- Chọn tiếp việc 1 có tiền thưởng 15. Xếp việc 1 với thời hạn  $t[1] = 1$  vào  $h$ : Không xếp được vì từ thời điểm 1 trở về trước trục thời gian  $h$  đã kín.

$i$	1	2	3	4	5
$h$	4	-1	-1	-1	3
thưởng	27				100

*Xét việc 1 ?*

- Chọn nốt việc 2 có tiền thưởng 10. Xếp việc 2 với thời hạn  $t[2] = 3$  vào  $h$ :  $h[3] = 2$ .

$i$	1	2	3	4	5
$h$	4	-1	2	-1	3
thưởng	27		10		100

*Xếp việc 2*

- Đồn việc trên trục thời gian  $h$

$i$	1	2	3	4	5
$h$	4	2	3	-1	-1

*Đồn việc*

- Xếp nốt việc phải làm mà không có thưởng, ta thu được  $h = (4, 2, 3, 1)$ .

$i$	1	2	3	4	5
$h$	4	2	3	1	-1

*Xếp nốt các việc còn lại*

- Ca làm việc kéo dài đúng  $N = 4$  giờ.
- Tổng tiền thưởng: 137

Vì các chỉ số mảng được tính từ 0 nên lúc đầu ta mã số các việc từ 0, khi ghi kết quả ta sẽ dịch thêm 1.

## Chương trình C++

```
// Xep viec
#include <bits/stdc++.h>

using namespace std;

int *a; // tien thuong
int *h; // truc thoi gian h[i] = viec k
int *t; // thoi han nop
int tmax;
int *id; // index
int n; // so viec
int thuong;
vector<int> viecChuaXep;

void Print(int x[], int d, int c, const char * msg="") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << x[i];
    }
}

void IdPrint(int x[], int d, int c, const char * msg="") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << x[id[i]];
    }
}

bool Greater(int i, int j) {
    return a[i] > a[j];
}

void ReadInput() {
    ifstream f("VIEC.INP");
    f >> n;
    cout << "\n " << n << " viec";
    int n1 = n;
    a = new int[n];
    id = new int[n];
```

```

    t = new int[n];
    tmax = 0;
    for (int i = 0; i < n; ++i) {
        f >> t[i] >> a[i]; // thoi han, tien thuong
        id[i] = i; // chi so
        if (tmax < t[i]) tmax = t[i];
    }
    f.close();
}

void WriteResult() {
    // Ghi file h = (4,2,3,1) 137
    ofstream g("VIEC.OUT");
    for (int i = 0; i < n; ++i) {
        g << h[i]+1 << endl;
    }
    g << thuong;
    g.close();
}

void XepViec() {
    ReadInput();
    h = new int[tmax+1];
    // Khoi tri truc thoi gian h
    for (int i = 0; i <= tmax; ++i) h[i] = -1;
    Print(a, 0, n-1, "\n Thuong a: ");
    Print(t, 0, n-1, "\n Han nop t: ");
    cout << "\n tmax = " << tmax;
    sort(id,id+n,Greater); // sap giam theo tien thuong
    cout << "\n Sau khi sap giam theo tien thuong";
    IdPrint(a, 0, n-1, "\n Thuong a: ");
    IdPrint(t, 0, n-1, "\n Han nop t: ");
    // Xep viec
    thuong = 0;
    int i, j, v;
    viecChuaXep.clear();
    cout << "\n CHU Y: cac chi so bi dich -1";
    for (i = 0; i < n; ++i) {
        v = id[i]; // viec v
        cout << "\n Xet viec " << v << ". Han nop " << t[v];
        for (j = t[v]; j > 0; --j) {
            if (h[j] < 0) { // gio trong
                h[j] = v;
                thuong += a[v];
                cout << "\n      * Xep duoc viec " << v
                    << " Tong thuong = " << thuong;
                break; // for j
            }
        } // for j
        if (j < 1) { // Ko xep duoc viec v
            viecChuaXep.push_back(v); // tam lu lai
        }
    } // for i
}

```

```

// Don viec
Print(h, 0, tmax, "\n Viec da xep h: ");
// h = (-1,3,-1,1,-1,2).
// -> h = (3,1,2,-1,-1)
int m = 0; // so luong viec sau khi don
for (int i = 1; i <= tmax; ++i) {
    if (h[i] != -1) {
        h[m++] = h[i];
        h[i] = -1;
    }
}
Print(h, 0, tmax, "\n Sau khi don viec h: ");
// Xep not cac viec con lai
for (int i = 0; i < viecChuaXep.size(); ++i) {
    h[m++] = viecChuaXep[i];
}
cout << "\n Tong cong " << m << " viec.";
Print(h, 0, tmax, "\n Tong ket h: ");
// -> h = (3,1,2,0,-1)
WriteResult();
}

main() {
    XepViec();
    cout << "\n T h e   E n d";
    return 0;
}

```

### Bài 5.3. Xếp ba lô

Có  $N$  vật (mặt hàng), với mỗi vật ta biết trọng lượng và giá trị của nó. Hãy xác định trọng lượng cần lấy ở một số vật để xếp vào một ba lô có sức chứa tối đa là  $M$  kg sao cho giá trị chứa trong ba lô là lớn nhất. Giả thiết là có thể lấy một số kg ở mỗi vật.

Dữ liệu vào: Tập văn bản `balo.inp`:

- Dòng đầu tiên: hai giá trị nguyên dương  $N$  và  $M$ .
- $N$  dòng tiếp theo, mỗi dòng chứa hai giá trị nguyên dương  $d$  và  $v$  cho mỗi vật, trong đó  $d$  là trọng lượng,  $v$  là giá trị tính theo một đơn vị trọng lượng của vật đó (đơn giá). Các số cách nhau qua dấu cách.

#### BALO.INP

$N = 5$  vật, sức chứa tối đa của ba lô là  
 $M = 30$  (kg).  
 5 30  
 8 5  
 5 4  
 4 2  
 3 8  
 16 6  
 Trọng lượng: kg  
 Đơn giá  $V$  / kg ( $V$  là đơn vị tiền)

#### BALO.OUT

8  
 3  
 0  
 3  
 16  
 172



*Dữ liệu ra:* Tập văn bản tên BALO.OUT :

- $N$  dòng, dòng thứ  $i$  cho biết trọng lượng cần lấy ở vật thứ  $i$ .
- Dòng cuối cùng ghi tổng giá trị thu được.

## Thuật toán

Có nhiều bài toán thuộc họ xếp ba lô, thuật toán cho bài này thuộc lớp tham lam.

Dễ thấy tiêu chuẩn chọn là giá đơn vị cao. Ta duyệt các vật theo giá đơn vị từ cao trở xuống. Vật được chọn sẽ được lấy tối đa. Như vậy, nếu tổng trọng lượng các vật bằng hoặc lớn hơn sức mang của ba lô thì bao giờ ba lô cũng được xếp đủ. Vật cuối trong balo có thể chỉ được lấy một phần.

## Chương trình C++

```
// Balo

#include <bits/stdc++.h>

using namespace std;

int *d; // trong luong
int *v; // don gia
int *id; // Sort theo index
int n; // so vat
int m; //suc chua cua balo
int *balo;
int t; // Tong gia tri

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

void Print(int x[], int d, int c, const char * msg="") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << x[i];
    }
}

// Hien thi theo chi dan Id
void IdPrint(int x[], int d, int c, const char * msg="") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << x[id[i]];
    }
}

// So sanh hai don gia
```

```
bool Greater(int i, int j) {
    return v[i] > v[j];
}

void ReadInput(){
    ifstream f("BALO.INP");
    f >> n >> m;
    cout << "\n " << n << " vat."
         << " Suc chua cua balo: " << m;
    d = new int[n]; // trong luong cac vat
    v = new int[n]; // don gia
    id = new int[n];
    balo = new int[n];
    for (int i = 0; i < n; ++i) {
        f >> d[i] >> v[i];
        id[i] = i;
        balo[i] = 0;
    }
    Print(d, 0, n-1, "\n Trong luong d:");
    Print(v, 0, n-1, "\n Don gia v:");
    Print(id, 0, n-1, "\n id:");
    f.close();
}

void WriteResult() {
    // Ghi file
    ofstream g("BALO.OUT");
    for (int i = 0; i < n; ++i) {
        g << balo[i] << endl;
    }
    g << t;
    g.close();
}

void Balo() {
    ReadInput();
    sort(id,id+n,Greater); // sap giam theo don gia
    IdPrint(d, 0, n-1, "\n Sorted d:");
    IdPrint(v, 0, n-1, "\n Sorted v:");
    // Xep
    t = 0; // tong gia tri
    for (int i = 0; i < n; ++i) {
        int j = id[i]; // chon vat j
        balo[j] = (d[j] <= m) ? d[j] : m;
        m -= balo[j];
        t += balo[j] * v[j];
        if (m == 0) break;
    }
    Print(balo, 0, n-1, "\n Balo: ");
    cout << "\n Tong gia tri: " << t;
    WriteResult();
}
```

```
main() {
    Balo();
    cout << "\n T h e   E n d";
    return 0;
}
```

## Output

```
5 vat. Suc chua cua balo: 30
Trong luong d: 8 5 4 3 16
Don gia v: 5 4 2 8 6
id: 0 1 2 3 4
Sorted d: 3 16 8 5 4
Sorted v: 8 6 5 4 2
Balo: 8 3 0 3 16
Tong gia tri: 172
T h e   E n d
```

## Bài 5.4. Cây khung ngắn nhất

Cho một đồ thị liên thông  $G$  vô hướng bao gồm  $n$  đỉnh, mã số từ 1 đến  $n$ , và  $m$  cạnh nối hai đỉnh với nhau. Mỗi cạnh có chiều dài cho trước. Tính liên thông của đồ thị cho biết với hai đỉnh tùy ý, ta luôn tìm được các cạnh nối nhau để đi từ đỉnh này đến đỉnh kia. Hãy chỉ ra một phần  $P$  của đồ thị thỏa các tính chất sau:

- (i)  $P$  chứa tất cả các đỉnh của  $G$ ;
- (ii)  $P$  chứa một số ít nhất các cạnh của  $G$ ;
- (iii)  $P$  là đồ thị liên thông;
- (iv) Tổng chiều dài các cạnh của  $P$  là ngắn nhất.

Đồ thị  $P$  thỏa ba tính chất (i), (ii) và (iii) được gọi là *cây khung* của đồ thị  $G$ . Nếu  $P$  thỏa thêm tính chất (iv) thì  $P$  được gọi là *cây khung ngắn nhất* của  $G$ . Một số tài liệu dùng thuật ngữ *cây bao trùm* thay cho *cây khung* và *cây bao trùm ngắn nhất* thay cho *cây khung cực tiểu*.

Bài toán trên có nhiều ứng dụng thực tiễn. Một trong số ứng dụng đó được mô tả thông qua ví dụ sau:

Có  $n$  máy tính được nối với nhau thành mạng bằng cáp quang là một loại dây truyền tin đắt tiền. Trong mạng này, hai máy tính bất kì đều có thể liên lạc được với nhau trực tiếp hoặc thông qua một vài máy trung gian. Ta gọi tính chất này là *tính liên thông* của mạng máy tính. Hãy bỏ bớt một số dây nối để  $n$  máy tính trên vẫn liên thông được với nhau. Mạng tối thiểu thu được chính là một cây khung ngắn nhất của mạng ban đầu.

Dữ liệu vào: tệp văn bản tên DOTHI.INP.

- Dòng đầu tiên ghi hai số tự nhiên  $n$  và  $m$  cách nhau qua dấu cách, biểu thị số đỉnh ( $n$ ) và số cạnh ( $m$ ) của đồ thị.
- Mỗi dòng thứ  $i = 1, 2, \dots, m$  trong số  $m$  dòng tiếp theo ghi ba giá trị  $x$   $y$  và  $d$  cách nhau qua dấu cách với ý nghĩa cạnh ( $x, y$ ) của đồ thị có chiều dài  $d$ .

Dữ liệu ra: tệp văn bản tên DOTHI.OUT bao gồm:

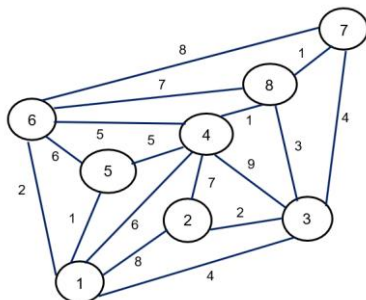
- Danh sách các cạnh được chọn.
- Dòng cuối cùng ghi tổng chiều dài tìm được.

### Ví dụ

DOTHI.INP

```
8 17
1 2 8
1 3 4
1 4 6
1 5 1
1 6 2
2 3 2
2 4 7
3 4 9
3 7 4
3 8 3
4 5 5
4 6 5
4 8 1
5 6 6
6 7 8
6 8 7
7 8 1
```

**Ý nghĩa:** Đồ thị có 8 đỉnh và 17 cạnh. Cạnh (1, 2) dài 8, cạnh (1, 3) dài 4, cạnh (1, 4) dài 6, ..., cạnh (7, 8) dài 1 đơn vị.



DOTHI.OUT

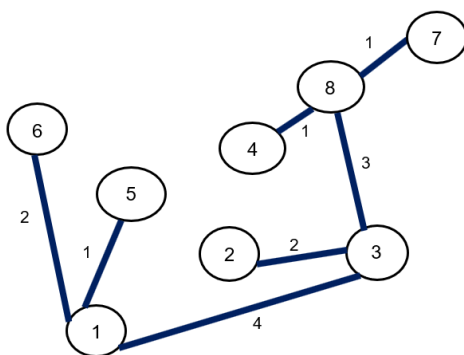
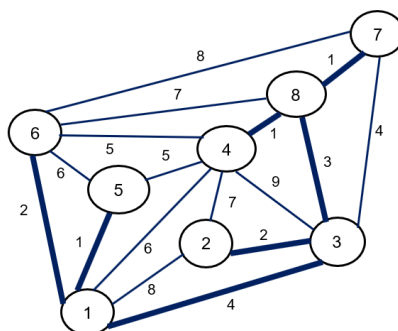
```
7 - 8
4 - 8
1 - 5
1 - 6
2 - 3
3 - 8
1 - 3
14
```

**Ý nghĩa:** Cây khung ngắn nhất của đồ thị đã cho gồm 8 đỉnh và 7 cạnh là (chiều dài mỗi cạnh được ghi sau dấu hai chấm):  
 cạnh 7 – 8: 1  
 cạnh 4 – 8: 1  
 cạnh 1 – 5: 1  
 cạnh 1 – 6: 2  
 cạnh 2 – 3: 2  
 cạnh 3 – 8: 3  
 cạnh 1 – 3: 4  
 Tổng chiều dài 7 cạnh đã chọn là: 14.

### Thuật toán

Ta dùng thuật giải Kruskal với quy trình như sau. Duyệt các cạnh từ chiều dài nhỏ đến lớn. Cạnh được chọn sẽ là cạnh không tạo thành chu trình khi ghép nó vào đồ thị kết quả.

1 5 1 lấy  
 4 8 1 lấy  
 7 8 1 lấy  
 1 6 2 lấy  
 2 3 2 lấy  
 3 8 3 lấy  
 1 3 4 lấy  
 3 7 4  
 4 5 5  
 4 6 5  
 1 4 6  
 5 6 6  
 2 4 7  
 6 8 7  
 1 2 8  
 6 7 8  
 3 4 9



*Cây khung cực tiểu*

Lưu ý rằng đồ thị kết quả thu được ở các bước trung gian có thể không liên thông mà bao gồm nhiều mảnh liên thông (cây con). Loại đồ thị này được gọi là rừng. Kết quả cuối cùng sẽ là cây vì nó liên thông và được tạo thành từ  $n - 1$  cạnh. Ta vận dụng tổ chức find-union cho các tập đỉnh rời nhau để quản lí các tập đỉnh được chọn nhằm phát hiện chu trình. Sau khi sắp các cạnh tăng dần theo chiều dài ta duyệt từng cạnh  $(x, y)$  và thực hiện hàm  $\text{Union}(x, y)$ . Cạnh  $(x, y)$  khi được ghép vào đồ thị trung gian sẽ tạo thành chu trình khi và chỉ khi các đỉnh  $x$  và  $y$  cùng nằm trong một cây của đồ thị (rừng) trung gian đó. Như vậy mỗi cây con của đồ thị trung gian được quản lí như một tập con của tập các đỉnh  $1..n$  của đồ thị ban đầu. Tập con này có phần tử đại diện chính là gốc của cây tương ứng. Phần tử này được chọn theo mã số nhỏ nhất. Các đỉnh còn lại của cây con đều trở về gốc đó.

Để thấy cây khung luôn luôn có  $n$  đỉnh và  $n - 1$  cạnh.

Khi lập trình ta lưu ý mã số của các đỉnh trong đề bài là 1..n, trong khi mã số của các đối tượng trong chương trình là 0..(n-1).

## Tổ chức dữ liệu

Mỗi cạnh cyar đồ thị được biểu diễn qua một bản ghi

```
typedef struct Canh {
    int X;
    int Y;
    int Len;
    int Khung;
} Canh;
```

trong đó X và Y là hai đỉnh thuộc cạnh đó, Len là chiều dài cạnh, Khung = 1 cho biết cạnh sẽ được chọn trong cây khung. Vì đồ thị là vô hướng nên ta quy ước  $X < Y$ .

## Chương trình C++

```
// Cay khung min
#include <bits/stdc++.h>

using namespace std;

typedef struct Canh {
    int X;
    int Y;
    int Len;
    int Khung;
} Canh;
Canh *canh;

int *id; // Sort tang Canh theo index
int *d; // mang tro tap dung cho union
int n; // so dinh
int m; // so canh

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

void Print(int x[], int d, int c, const char * msg="") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << x[i];
    }
}

// Hien thi canh i
// tang ma so canh len 1
void Print(int i) {
    cout << "\n " << canh[i].X+1 << " - " << canh[i].Y+1
```

```

        << " : " << canh[i].Len
        << " (" << canh[i].Khung << ")";
    }

    // Hien thi canh
    void Print(int d, int c, const char * msg="") {
        cout << msg;
        for (int i = d; i <= c; ++i) {
            Print(i);
        }
    }

    // Hien thi canh theo chi dan Id
    void IdPrint(int d, int c, const char * msg="") {
        cout << msg;
        for (int j = d; j <= c; ++j) {
            Print(id[j]);
        }
    }

    // So sanh hai hau to s(i) va s(j)
    // s(i) < s(j) ?
    bool Less(int i, int j) {
        return canh[i].Len < canh[j].Len;
    }

    // Kruskal
    // Tim dai dien cua tap chua dinh x
    int Find(int x) {
        while (d[x] != x) x = d[x];
        return x;
    }

    // Hop tap chua x voi tap chua y
    int Union(int x, int y) {
        x = Find(x); y = Find(y);
        if (x == y) return 0;
        if (x < y) d[y] = x;
        else d[y] = y;
        return 1;
    }

    void CayKhungMin() {
        ifstream f("DOTHI.INP");
        f >> n >> m;
        cout << "\n " << n << " dinh " << m << " canh";
        canh = new Canh[m];
        d = new int [n]; // cap phat d
        id = new int[n]; // cap phat id
        for (int i = 0; i < m; ++i) { // doc cac canh
            f >> canh[i].X >> canh[i].Y >> canh[i].Len;
            --canh[i].X; --canh[i].Y; // dat lai so hieu canh
            canh[i].Khung = 0;
        }
    }

```

```

        if (canh[i].X > canh[i].Y) {
            int z = canh[i].X;
            canh[i].X = canh[i].Y;
            canh[i].Y = z;
        }
        id[i] = i;
    }
    f.close();
    for(int i = 0; i < n; ++i) d[i] = i;
    Print(d, 0, n-1, "\n d: ");
    Print(id, 0, m-1, "\n id: ");
    sort(id, id+m, Less);
    IdPrint(0, m-1, "\n Sort Canh: ");
    int c = 0; // so canh trong cay khung
    for (int j = 0; j < m; ++j) {
        int i = id[j];
        canh[i].Khung = Union(canh[i].X, canh[i].Y);
        if (canh[i].Khung == 1) {
            ++c;
            if(c == n - 1) break;
        }
    }
    // Ket qua
    cout << "\n Ket qua: ";
    ofstream g("DOTHI.OUT");
    int minlen = 0;
    for (int j = 0; j < m; ++j) {
        int i = id[j];
        if (canh[i].Khung == 1) {
            g << canh[i].X+1 << " " << canh[i].Y+1 << endl;
            Print(i);
            minlen += canh[i].Len;
        }
    }
    g << minlen;
    g.close();
    cout << "\n Tong len cua cay khung min = " << minlen;
}

main() {
    CayKhungMin();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

8 dinh 17 canh
d:  0 1 2 3 4 5 6 7
id:  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Sort Canh:
7 - 8 : 1 (0)

```



```

4 - 8 : 1 (0)
1 - 5 : 1 (0)
1 - 6 : 2 (0)
2 - 3 : 2 (0)
3 - 8 : 3 (0)
3 - 7 : 4 (0)
1 - 3 : 4 (0)
4 - 5 : 5 (0)
4 - 6 : 5 (0)
1 - 4 : 6 (0)
5 - 6 : 6 (0)
2 - 4 : 7 (0)
6 - 8 : 7 (0)
1 - 2 : 8 (0)
6 - 7 : 8 (0)
3 - 4 : 9 (0)
Ket qua:
7 - 8 : 1 (1)
4 - 8 : 1 (1)
1 - 5 : 1 (1)
1 - 6 : 2 (1)
2 - 3 : 2 (1)
3 - 8 : 3 (1)
1 - 3 : 4 (1)
Tong len cua cay khung min = 14
T h e   E n d

```

## Bài 5.5. Trộn hai tệp

Cho hai tệp văn bản *data1.inp* và *data2.inp* chứa các số nguyên được sắp tăng. Viết chương trình trộn hai dãy dữ liệu trong hai tệp này thành một dãy dữ liệu sắp tăng duy nhất và ghi trong tệp văn bản *data.out*.

Dòng đầu tiên là số lượng các số trong mỗi tệp. Tiếp đến là giá trị nguyên của các phần tử trong tệp, mỗi số được ghi trên một dòng.

- Với dữ liệu đã cho trong tệp thứ nhất là 5 số, tệp thứ hai là 6 số thì tệp kết quả sẽ chứa 11 số.
- Số lượng các số trong mỗi tệp tối đa là 50 nghìn.
- Các số có giá trị kiểu nguyên, được tách nhau bởi dấu cách và có thể nằm trên nhiều dòng.
- Khi trộn hai tệp nói trên ta phải thực hiện tối thiểu 22 lần đọc-ghi bao gồm 11 lần đọc và 11 lần ghi.

### Ví dụ

<u>data1.inp</u>	<u>data2.inp</u>	<u>data.out</u>
5	6	11
2	3	2
3	3	3
5	4	3
5	7	3
10	12	4
	20	5
		5
		7
		10
		12
		20

## Thuật toán

Ta dùng phương pháp cân. Gọi hai tệp chứa dữ liệu cần trộn là  $f$  và  $g$ , tệp chứa kết quả trộn là  $h$ . Hãy tưởng tượng, ta dùng tay trái lấy lần lượt, mỗi lần một phần tử của tệp  $f$  (ghi vào biến  $t$ ) và dùng tay phải lấy lần lượt mỗi lần một phần tử của tệp  $g$  (ghi vào biến  $p$ ). So sánh vật nặng trên hai tay  $t$  và  $p$ . Tay nào cầm phần tử nhẹ hơn thì đặt phần tử đó vào tệp kết quả  $h$  và do tay đó rỗng, nên lấy tiếp phần tử từ tệp tương ứng. Quá trình này kết thúc khi nào một trong hai tệp  $f$  hoặc  $g$  được duyệt xong. Cuối cùng ta chuyển nốt các phần tử còn lại của tệp chưa duyệt hết (tệp  $f$  hoặc  $g$ ) vào tệp kết quả  $h$ .

## Chương trình C++

```
// Merge File
#include <iostream>
#include <windows.h>
#include <bits/stdc++.h>

using namespace std;

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

void Run(){
    ifstream f("DATA1.INP");
    ifstream g("DATA2.INP");
    ofstream h("DATA.OUT");
    int n; // so phan tu trong file f
    int m; // so phan tu trong file g
    int x, y; // x thuoc f, y thuoc g
    f >> n; g >> m;
    cout << "\n size of f: " << n;
    cout << "\n size of g: " << m;
```

```

    h << (n+m) << endl;
    cout << "\n size of h: " << (n+m);
    int i = 0; // id of f
    int j = 0; // id of g
    // moi file co it nhat 1 phan tu
    f >> x; g >> y;
    while(i < n && j < m) {
        if (x <= y) {
            h << x << endl;
            f >> x;
            ++i;
        }
        else {
            h << y << endl;
            g >> y;
            ++j;
        }
    }
    while(i < n) {
        h << x << endl;
        f >> x;
        ++i;
    }
    while(j < m) {
        h << y << endl;
        g >> y;
        ++j;
    }
    f.close(); g.close(); h.close();
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

### File DATA.OUT

```

11
2
3
3
3
4
5
5
7
10
12
20

```

## Chương trình Python

### Output

```
n = 5    m = 6
Tong so lan doc-ghi: 11
T h e    E n d .
```

*Chú thích* Dĩ nhiên, bạn có thể đọc dữ liệu từ hai file vào hai mảng tương ứng rồi trộn hai mảng này. Tuy nhiên chúng ta muốn minh họa lời giải với hạn chế là mỗi lần chỉ được phép đọc một đơn vị dữ liệu từ mỗi file.

### Bài 5.6. Trộn nhiều tệp

Cho  $n$  tệp văn bản mã số từ 1 đến  $n$ . Tệp thứ  $i$  chứa  $d_i$  phần tử được sắp tăng. Hãy lập một lịch chỉ ra trình tự trộn mỗi lần hai tệp để cuối cùng thu được một tệp sắp tăng duy nhất với tổng số lần ghi dữ liệu vào tệp là nhỏ nhất. Biết rằng thủ tục trộn hai tệp chỉ có thể đọc tuần tự hoặc ghi tuần tự mỗi lần một phần tử.

Dữ liệu vào: Tệp văn bản MF.INP.

- Dòng đầu tiên là số lượng  $n$  các tệp chứa dữ liệu sắp tăng.
- Tiếp đến là  $n$  số tự nhiên  $d_i$ ,  $i = 1..n$  cho biết số phần tử trong tệp thứ  $i$ . Mỗi số ghi trên một dòng.

Dữ liệu ra: Tệp văn bản MF.OUT.

- Dòng đầu tiên:  $m$  là số lần thực hiện trộn hai tệp.
- Tiếp đến là  $m$  dòng, mỗi dòng chứa ba số tự nhiên  $i$ ,  $j$  và  $k$  cho biết cần lấy tệp  $i$  trộn với tệp  $j$  và ghi kết quả vào tệp  $k$ . Các số trên cùng một dòng cách nhau qua dấu cách.

Tệp chứa kết quả trung gian phải có mã số khác với mã số của các tệp tạo lập trước đó.

### Ví dụ

<u>MF.INP</u>	<u>MF.OUT</u>	
5	4	<b>Ý nghĩa:</b> Cho 5 tệp sắp tăng với số phần tử lần lượt là 10,
10	5 3 6	5, 4, 4, 3. Cần thực hiện 4 lần trộn, mỗi lần 2 tệp.
5	4 2 7	Lần thứ nhất: trộn tệp 5 với tệp 3 ghi vào tệp 6.
4	6 7 8	Lần thứ hai: trộn tệp 4 với tệp 2 ghi vào tệp 7.
4	1 8 9	Lần thứ ba: trộn tệp 6 với tệp 7 ghi vào tệp 8.
3	58	Lần thứ tư: trộn tệp 1 với tệp 8 ghi vào tệp 9.
		Tổng số lần ghi là 58.

### Thuật toán

Trước hết đề ý rằng nếu trộn tệp sắp tăng  $f$  gồm  $n$  phần tử với tệp sắp tăng  $g$  gồm  $m$  phần tử để thu được tệp sắp tăng  $h$  thì đối với các phần tử trong hai tệp nguồn ta chỉ cần

thực hiện thao tác đọc, còn thao tác ghi chỉ thực hiện đối với tệp đích  $h$ . Kí hiệu  $|f|$  là số phần tử trong tệp  $f$ , ta có:

$$|f| = n, |g| = m$$

Do tổng số các phần tử của hai tệp là  $m + n$  nên số phần tử trong tệp đích  $h$  sẽ là

$$|h| = n + m = |f| + |g|$$

và do đó số lần ghi (tối thiểu) các phần tử vào tệp  $h$  sẽ là  $n + m$ . Như vậy, khi trộn hai tệp thì số lần đọc và số lần ghi là bằng nhau. Chính vì vậy nên đề bài chỉ đòi hỏi tính số lần ghi là đủ.

Ta có nhận xét sau: Muốn xây dựng một quy trình trộn mỗi lần hai tệp cho nhiều tệp ban đầu với yêu cầu tổng số thao tác ghi tệp là tối thiểu thì ta phải tạo ra các tệp trung gian càng ít phần tử càng tốt.

Ta dùng kí hiệu  $f \oplus g \rightarrow h$  với ý nghĩa là trộn hai tệp nguồn  $f$  và  $g$  để thu được tệp  $h$ . Ta có

$$\text{Nếu } f \oplus g \rightarrow h \text{ thì } |h| = |f| + |g|$$

Để ý rằng trộn tệp  $f$  với tệp  $g$  hay trộn tệp  $g$  với tệp  $f$  thì số thao tác ghi tệp như nhau và cùng bằng  $|f| + |g|$ . Giả sử ta có ba tệp với số phần tử tương ứng là

$$s[1..3] = (5, 1, 2).$$

Giả sử ta thực hiện quy trình ①  $\oplus$  ②  $\oplus$  ③ như sau:

Bước 1: Trộn tệp ① với tệp ② ghi tạm vào tệp ④. Số thao tác ghi sẽ là  $(5 + 1) = 6$  và tệp ④ có 6 phần tử.

Bước 2: Trộn tệp ④ với tệp ③ ghi vào tệp ⑤. Số thao tác ghi sẽ là  $6 + 2 = 8$  và tệp ⑤ có 8 phần tử.

Kết quả thu được tệp ⑤. Tổng số thao tác ghi trong cả hai bước trên sẽ là:

$$6 + 8 = 14.$$

Tổng quát, với ba tệp  $a$ ,  $b$  và  $c$  được trộn theo quy trình:

$$(a \oplus b) \oplus c$$

ta dễ dàng tính được tổng số thao tác ghi tệp cho quy trình trên là

$$(|a| + |b|) + (|a| + |b|) + c = 2(|a| + |b|) + c.$$

Bảng dưới đây tính toán cho ba phương án để phát hiện ra phương án tối ưu.

Phương án	Quy trình thực hiện	Tổng số thao tác ghi tệp
1	(① $\oplus$ ②) $\oplus$ ③	$2(5 + 1) + 2 = 2 \times 6 + 2 = 14$
2	(① $\oplus$ ③) $\oplus$ ②	$2(5 + 2) + 1 = 2 \times 7 + 1 = 15$
3	(② $\oplus$ ③) $\oplus$ ①	$2(1 + 2) + 5 = 2 \times 3 + 5 = 11$

(phương án tối ưu)

Khảo sát các quy trình trộn ba tệp

$$s[1..3] = (5, 1, 2)$$

Thuật toán tham lam do Huffman đề xuất khi đó sẽ như sau:

---



---

Thuật toán Huffman

---



---

---

Xuất phát từ danh sách các tệp cần xử lí  
 Lặp (đến khi chỉ còn một tệp duy nhất)

- Lấy hai tệp  $u$  và  $v$  có số phần tử nhỏ nhất.
- Trộn  $u \oplus v \rightarrow h$ . Ta có  $|h| = |u| + |v|$ .
- Loại bỏ  $u$  và  $v$  khỏi danh sách các tệp cần xử lí.
- Kết nạp  $h$  vào danh sách các tệp cần xử lí

xong lặp

---

Với  $n$  tệp ban đầu, dễ thấy rằng mỗi lần lặp ta loại bỏ được hai tệp ( $u$  và  $v$  có số phần tử min) và thêm một tệp ( $h$ ) tức là mỗi lần lặp ta loại bỏ được một tệp, do đó số lần lặp sẽ là  $n - 1$ .

Thuật toán trên mang tên nhà toán học Mĩ Huffman là người đầu tiên đề xuất.

Ta minh hoạ thuật toán trên với dữ liệu vào như sau:

$$s[1..5] = (10, 5, 4, 4, 3).$$

Ý nghĩa: Trộn 5 tệp sắp tăng với số phần tử lần lượt là 10, 5, 4, 4 và 3 để thu được một tệp sắp tăng duy nhất.

Lần lặp	Danh sách các tệp cần xử lí	Hai tệp có số phần tử min	Trộn	Số thao tác ghi tệp
1	(①:10, ②:5, ③:4, ④:4, ⑤:3)	⑤:3, ③:4	⑤ $\oplus$ ③ $\rightarrow$ ⑥	7
2	(①:10, ②:5, ④:4, ⑥:7)	②:5, ④:4	② $\oplus$ ④ $\rightarrow$ ⑦	9
3	(①:10, ⑥:7, ⑦:9)	⑥:7, ⑦:9	⑥ $\oplus$ ⑦ $\rightarrow$ ⑧	16
4	(①:10, ⑧:16)	①:10, ⑧:16	① $\oplus$ ⑧ $\rightarrow$ ⑨	26
Kết quả	(⑨:26)		Tổng số lần ghi	58

Minh hoạ thuật toán Huffman với dữ liệu vào  
 (①:10, ②:5, ③:4, ④:4, ⑤:3)

Vì  $n = 5$  nên số lần lặp sẽ là  $n - 1 = 4$ . Sau 4 lần lặp ta thu được tệp mã số 9 với 26 phần tử. Để tính tổng số thao tác ghi ta chỉ cần lấy tổng số phần tử của các tệp tham gia trong mỗi lần trộn hai tệp. Tổng đó là:

$$tt = (3 + 4) + (5 + 4) + (7 + 9) + (10 + 16) = 7 + 9 + 16 + 26 = 58.$$

## Chương trình C++

```
// Merge Files
#include <bits/stdc++.h>

using namespace std;

const int MAXVAL = INT_MAX;

int *d; // f[i] = số phần tử trong file i
int n; // số lượng file sắp tăng
int m;

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}
```

```

}

void Print(int x[], int d, int c, const char * msg="") {
    cout << msg;
    for (int i = d; i <= c; ++i) {
        cout << " " << x[i];
    }
}

// Tim min trong d
int Min() {
    // Print(d, 0, m-1, "\n d:"); Go();
    int imin;
    int minval = MAXVAL;
    for (int i = 0; i < m; ++i) {
        if (d[i] < minval) {
            minval = d[i];
            imin = i;
        }
    }
    return imin;
}

void HuffMan() {
    int ii, jj; // luc dau co m files
    int x, y;
    int t = 0; // tong so lan doc ghi
    m = n;
    ofstream g("MF.OUT");
    g << (n-1) << endl;
    for (int i = 1; i < n; ++i) { // n-1 lan lap
        ii = Min();
        x = d[ii];
        // cout << "\n Min1 = " << ii;
        d[ii] = MAXVAL; // danh dau file ii
        jj = Min();
        y = d[jj];
        // cout << "\n Min2 = " << jj;
        d[jj] = MAXVAL; // danh dau file jj
        d[m++] = x + y; // file moi: ket qua trung gian
        t += (x + y); // so lan ghi
        // cout << "\n x = " << x << " y = " << y;
        cout << "\n Merge files " << ii+1 << " (+) " << jj+1
            << " -> " << m;
        g << ii+1 << " " << jj+1 << " " << m << endl;
    } // for
    cout << "\n Tong so lan doc-ghi " << t;
    g << t << endl;
    g.close();
}

void ReadInput() {
    ifstream f("MF.INP");

```

```

f >> n;
cout << "\n " << n << " files";
d = new int[n+n-1];
for (int i = 0; i < n; ++i) {
    f >> d[i];
}
f.close();
Print(d, 0, n-1, "\n d: ");
}

void Run() {
    ReadInput();
    HuffMan();
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

5 files
d:  10 5 4 4 3
Merge files 5 (+) 3 -> 6
Merge files 4 (+) 2 -> 7
Merge files 6 (+) 7 -> 8
Merge files 1 (+) 8 -> 9
Tong so lan doc-ghi 58
T h e   E n d

```

## Bài 5.7 Heap

Hàm Min được gọi  $n-1$  lần, mỗi lần duyệt  $m$  phần tử nên có độ phức tạp  $O(n^2)$ . Ta có thể giảm độ phức tạp xuống đến  $O(n \log(n))$  thông qua tổ chức dữ liệu heap như sau.

Heap là tổ chức dữ liệu dạng *cây nhị phân cân bằng* luôn cho ta phần tử min hoặc max của dãy các phần tử biến thiên sau mỗi lần cập nhật. Có hai loại min heap và max heap chỉ khác nhau ở giá trị ra là trị số min hoặc max. Ta sẽ vận dụng min heap.

Min heap là cây nhị phân chứa các giá trị  $h[0], h[1], \dots, h[n-1]$  trong đó phần tử cha không nhỏ hơn hai con. Các đặc điểm đồng thời là những ưu thế quan trọng nhất của heap là:

- ✧ Heap được tổ chức tại chỗ, nghĩa là ngoài mảng  $h$  ra, không cần dùng mảng phụ
- ✧ Sau mỗi lần cập nhật với độ phức tạp  $O(\log(n))$ , ta luôn luôn nhận được phần tử  $h[0]$  chứa giá trị min của  $h$ .

Min heap  $h$  được tổ chức cho một dãy số  $a[0:n-1]$  như sau.

- ✧ Khởi trị  $h$  là heap rỗng dưới dạng `vector<int>`
- ✧ Mỗi lần ta nạp một phần tử  $a[i]$  vào cuối heap  $h$ . Phần tử mới này có thể vi phạm tính chất của min heap  $h$ , tức là có thể gây nên hiện tượng node cha không nhỏ hơn hai node con, do đó ta phải sửa lại một cách liên hoàn dãy các phần tử nằm từ cuối



heap,  $h[n-1] = a[i]$  lên đến  $h[0]$ . Thủ tục chỉnh lại heap từ dưới lên trên được gọi là Up sẽ hoạt động như sau:




---



---

**Thuật toán Up**

---

Input: vị trí  $c$   
Output: heap  $h$  sau khi được cập nhật

---

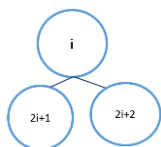
Begin  
 $vc = h[c]$  // Xác định trị  $vc$  của node con  $c$   
Xác định node cha trực tiếp  $p$  của  $c$   
while  $h[p] > vc$  do  
 $h[c] = h[p]$  // day trị của cha xuống  
 $p = c$   
Xác định node cha trực tiếp  $p$  của  $c$   
end while  
 $h[c] = vc$   
End Up

---



---

Các node trong heap được mã số từ 0, do đó các node cha và con được xác định theo quy luật sau:



Node: cha  $i$   
Node con trái:  $2i + 1$   
Node con phải:  $2i + 2$   
-----  
Node con:  $c$   
Node cha:  $(c+1) \div 2 - 1$

## Ví dụ 1

Cha	0	1	2	3	4	5	6	7	8
Con trái	1	3	5	7	*	*	*	*	*
Con phải	2	4	6	8	*	*	*	*	*

Nếu  $n = 9$ ,  $h$  có số hiệu 0..8  
thì các node cha  $> 3$  không có con

## Ví dụ 2

Con	0	1	2	3	4	5	6	7	8
Cha	*	0	0	1	1	2	2	3	3

Mỗi lần lấy một phần tử tại  $h[0]$  ra khỏi heap  $h$  ta nhận được một giá trị min trong số các phần tử còn trong  $h$ . Sau khi lấy  $h[0]$  khỏi  $h$ , ta đưa phần tử cuối  $h$  lên vị trí  $h[0]$  và cập nhật lại  $h$  bằng thủ tục Down như sau:

---

**Thuật toán Down**

---

Input: vị trí p (node cha)

Output: heap h sau khi được cập nhật

---

Begin

vp = h[p] // Xác định trị vp của node cha p

Xác định con c nhận giá trị min

của hai con trái và phải (nếu có) của p

while vp &gt; h[c] do

h[p] = h[c];

p = c;

Xác định con c nhận giá trị min

của hai con trái và phải (nếu có) của

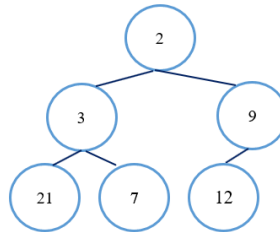
end while

h[p] = vp

End Down

---

Các chương trình dưới đây tạo heap h từ dãy 6 số 7, 3, 9, 21, 2, 12 sau đó lấy ra các phần tử min từ h[0] và hiển thị: 2 3 7 9 12 21.

**Chương trình C++**

```

/*****
Min Heap
*****/
#include <iostream>
#include <fstream>
#include <windows.h>
#include <bits/stdc++.h>

using namespace std;

typedef vector<int> VI;
VI h; // heap

void Go() {
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

void Print(VI x, const char * msg = "") {

```

```

    cout << msg;
    for (int i = 0; i < x.size(); ++i)
        cout << " " << x[i];
}

// Tu node h[p] dich xuong
// cap nhat lai heap h
int Down(int p) {
    int n = h.size();
    int vp = h[p]; // Gia tri hien hanh cua node cha
    int c; // c: node con, p: node cha
    while(true) {
        c = p + p + 1; // con trai
        if (c >= n) break; // Het node con
        if (c + 1 < n) { // neu co con phai
            // c la node nho nhat tong 2 con
            if (h[c+1] < h[c]) ++c;
        }
        // so sanh cha con
        if (vp <= h[c]) break;
        // cha > con
        h[p] = h[c]; // cha <= con
        p = c; // chuyen cha xuong
    }
    h[p] = vp;
    return 1;
}

// Chinh lai h tu h[v] ve truoc
int Up(int c) { // con
    int p;
    int vc = h[c]; // gia tri tai node con
    while (true) {
        if (c == 0) break;
        p = (c+1) / 2 - 1; // cha
        if (h[p] <= vc) break; // cha <= con: OK
        // h[p] > vc: cha > con
        h[c] = h[p]; // chuyen con len vi tri cha
        c = p;
    }
    h[c] = vc;
    return 1;
}

// Them tri v vao cuoi heap h
void Ins(int v) {
    h.push_back(v);
    Up(h.size()-1);
}

// Lay phan tu min tai heap h[0]
int Take() {
    if (h.size() == 0) return -1;

```

```

    int x = h[0];
    h[0] = h[h.size()-1];
    h.pop_back();
    // chỉnh lại heap sau khi mất đầu
    if (!h.empty()) Down(0);
    return x;
}

void Run() {
    h.clear();
    int a[] = {7, 3, 9, 21, 2, 12};
    int n = 10;
    cout << "\n a: ";
    for (int i = 0; i < n; ++i) {
        cout << " " << a[i];
        Ins(a[i]);
    }
    Print(h, "\n Min heap h: ");
    cout << "\n Take min: ";
    while(!h.empty()) {
        cout << " " << Take();
    }
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

a:  7 3 9 21 2 12
Min heap h:  2 3 9 21 7 12
Take min:  2 3 7 9 12 21
T h e   E n d

```

## Bài 5.8 Thuật toán Huffman

Đến đây ta có thể triển khai hàm Huffman thông qua các thủ tục tìm các file có ít phần tử nhất trong số các file chưa xử lý. Lưu ý rằng bạn cần thay đổi chút ít nội dung trong heap  $h$ , cụ thể là thay vì ghi giá trị của các phần tử trong dãy dữ liệu vào  $h$ , ta chỉ ghi các chỉ số của các file.

Thuật toán Huffman (dùng heap)
Tạo heap từ danh sách các tệp cần xử lý
Lặp $n-1$ lần
<ul style="list-style-type: none"> <li>• <math>u = \text{Take}()</math> // min heap</li> <li>• <math>v = \text{Take}()</math> // min heap</li> <li>• Trộn <math>u \oplus v \rightarrow h</math>. Ta có <math> h  =  u  +  v </math>.</li> <li>• <math>\text{Ins}()</math> // nạp <math>h</math> vào heap</li> </ul>
xong lặp

## Chương trình C++

```
// Merge Files (dung min heap)
#include <bits/stdc++.h>

using namespace std;

typedef vector<int> VI;
VI h; // heap chua cac id cua d
VI d;
int n; // so luong file sap tang
int m;

// Tu node h[p] dich xuong
// cap nhat lai heap h
int Down(int p) {
    int n = h.size();
    int hp = h[p]; // Gia tri hien hanh cua node cha
    int vp = d[hp];
    int c; // c: node con, p: node cha
    while(true) {
        c = p + p + 1; // con trai
        if (c >= n) break; // Het node con
        if (c + 1 < n) { // neu co con phai
            // c la node nho nhat tong 2 con
            if (d[h[c+1]] < d[h[c]]) ++c;
        }
        // so sanh cha con
        if (vp <= d[h[c]]) break;
        // cha > con
        h[p] = h[c]; // cha <= con
        p = c; // chuyen cha xuong
    }
    h[p] = hp;
    return 1;
}

// Chinh lai h tu h[c] ve truoc
int Up(int c) { // con
    int p;
    int hc = h[c]; // so hieu tai node con
    int vc = d[hc]; // gia tri tai node con
    while (true) {
        if (c == 0) break;
        p = (c+1) / 2 - 1; // cha
        if (d[h[p]] <= vc) break; // cha <= con: OK
        // h[p] > vc: cha > con
        h[c] = h[p]; // chuyen con len vi tri cha
        c = p;
    }
}
```

```

    h[c] = hc;
    return 1;
}

// Them tri d[i] vao cuoi heap h
void Ins(int i) {
    h.push_back(i);
    Up(h.size()-1);
}

// Lay phan tu min tai heap h[0]
int Take() {
    if (h.size() == 0) return -1;
    int x = h[0];
    h[0] = h[h.size()-1];
    h.pop_back();
    // chinh lai heap sau khi mat dau
    if (!h.empty()) Down(0);
    return x;
}

void Print(VI x, const char * msg = "") {
    cout << msg;
    for (int i = 0; i < x.size(); ++i)
        cout << " " << x[i];
}

void HuffMan() {
    int ii, jj; // luc dau co m files
    int t = 0; // tong so lan doc ghi
    m = n;
    ofstream g("MF.OUT");
    g << (n-1) << endl;
    for (int i = 1; i < n; ++i) { // n-1 lan lap
        ii = Take(); // get min
        // cout << "\n Min1 = " << ii;
        jj = Take(); // get min
        // cout << "\n Min2 = " << jj;
        d.push_back(d[ii] + d[jj]); // node moi d[ii] (+) d[jj]
        t += d[m];
        Ins(m); // nap m vao heap
        ++m;
        // cout << "\n x = " << x << " y = " << y;
        cout << "\n Merge files " << ii+1 << " (+) " << jj+1
            << " -> " << m;
        g << ii+1 << " " << jj+1 << " " << m << endl;
    } // for
    cout << "\n Tong so lan doc-ghi " << t;
    g << t << endl;
    g.close();
}

// Tao min heap tu d[i]

```

```

void MinHeap() {
    h.clear();
    for (int i = 0; i < n; ++i) {
        Ins(i); // nạp d[i] vào min heap h
    }
}

void ReadInput() {
    ifstream f("MF.INP");
    f >> n;
    cout << "\n " << n << " files";
    d.clear();
    int x;
    for (int i = 0; i < n; ++i) {
        f >> x;
        d.push_back(x);
    }
    f.close();
}

void Run() {
    ReadInput();
    Print(d, "\n d: ");
    MinHeap();
    Print(h, "\n h: ");
    HuffMan();
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

5 files
d:  10 5 4 4 3
h:  4 2 1 0 3
Merge files 5 (+) 4 -> 6
Merge files 3 (+) 2 -> 7
Merge files 6 (+) 7 -> 8
Merge files 1 (+) 8 -> 9
Tong so lan doc-ghi 58
T h e   E n d

```

## Chú ý

Trật tự xử lý theo min heap có thể khác với thuật toán dùng mảng, nhưng kết quả tối ưu không thay đổi.

### **Chú ý**

Thuật ngữ *tham lam* không có nghĩa là lấy nhiều nhất mà chỉ là xác định một chiến lược xử lý dữ liệu hiệu quả nhất.



## CHƯƠNG 6

### PHƯƠNG PHÁP QUAY LUI

---

Giả sử ta phải tìm trong một tập dữ liệu  $D$  cho trước một dãy dữ liệu:

$$v = (v[1], v[2], \dots, v[n])$$

thoả mãn đồng thời hai tính chất  $P$  và  $Q$ . Trước hết ta chọn một trong hai tính chất đã cho để làm nền, giả sử ta chọn tính chất  $P$ .

Sau đó ta thực hiện các bước sau đây:

Bước 1. (Khởi trị) Xuất phát từ một dãy ban đầu  $v = (v[1], \dots, v[i])$  nào đó của các phần tử trong  $D$  sao cho  $v$  thoả  $P$ .

Bước 2. Nếu  $v$  thoả  $Q$  ta dừng thuật toán và thông báo kết quả là dãy  $v$ , ngược lại ta thực hiện Bước 3.

Bước 3. Tìm tiếp một phần tử  $v[i+1]$  để bổ sung cho  $v$  sao cho

$$v = (v[1], \dots, v[i], v[i+1]) \text{ thoả } P.$$

Có thể xảy ra các trường hợp sau đây:

3.1. Tìm được phần tử  $v[i+1]$ : quay lại bước 2.

3.2. Không tìm được  $v[i+1]$  như vậy, tức là với mọi  $v[i+1]$  có thể lấy trong  $D$ , dãy  $v = (v[1], \dots, v[i], v[i+1])$  không thoả  $P$ . Điều này có nghĩa là đi theo đường

$$v = (v[1], \dots, v[i])$$

sẽ không dẫn tới kết quả. Ta phải đổi hướng tại một vị trí nào đó. Để thoát khỏi ngõ cụt này, ta tìm cách thay  $v[i]$  bằng một giá trị khác trong  $D$ . Nói cách khác, ta loại  $v[i]$  khỏi dãy  $v$ , giảm  $i$  đi một đơn vị rồi quay lại Bước 3.

Cách làm như trên được gọi là *quay lui*: lùi lại một bước.

Dĩ nhiên ta phải đánh dấu  $v[i]$  là phần tử đã loại tại vị trí  $i$  để sau đó không lấy lại phần tử đó.

Khi nào thì có thể trả lời là không tồn tại dãy  $v$  thoả đồng thời hai tính chất  $P$  và  $Q$ ? Nói cách khác, khi nào thì ta có thể trả lời là bài toán vô nghiệm?

Dễ thấy, bài toán vô nghiệm khi ta đã duyệt hết mọi khả năng. Ta nói là đã vét cạn mọi khả năng. Chú ý rằng có thể đến một lúc nào đó ta phải lùi liên tiếp nhiều lần. Từ đó suy ra rằng, thông thường bài toán vô nghiệm khi ta không còn có thể lùi được nữa. Có nhiều sơ đồ giải các bài toán quay lui, dưới đây là một sơ đồ khá đơn giản, không đệ quy.

---

*Sơ đồ 1: Giải bài toán quay lui*

*(tìm 1 nghiệm)*

---

```

Khởi trị dãy thoả P
while true
  if (v thoả Q)
    Ghi nhận nghiệm
    exit
  end if
  if (hết cách đi)
    Thông báo vô nghiệm
    exit
  end if
  if (Tìm được 1 nước đi) Tiến

```

---

---

```

    else Lùi
  end if
end while

```

---

Thông thường ta khởi trị cho  $v$  là dãy rỗng (không chứa phần tử nào) hoặc dãy có một phần tử. Ta chỉ yêu cầu dãy  $v$  được khởi trị sao cho  $v$  thoả  $P$ . Lưu ý là cả dãy  $v$  thoả  $P$  chứ không phải từng phần tử trong  $v$  thoả  $P$ .

Có bài toán yêu cầu tìm toàn bộ (mọi nghiệm) các dãy  $v$  thoả đồng thời hai tính chất  $P$  và  $Q$ . Nếu biết cách tìm một nghiệm ta dễ dàng suy ra cách tìm mọi nghiệm như sau: mỗi khi tìm được một nghiệm, ta thông báo nghiệm đó trên màn hình hoặc ghi vào một tệp rồi thực hiện thao tác Lùi, tức là giả vờ như không công nhận nghiệm đó, do đó phải loại  $v[i]$  cuối cùng trong dãy  $v$  để tiếp tục tìm hướng khác. Phương pháp này có tên là phương pháp *giả sai*. Sơ đồ trên sẽ được sửa một chút như sau để tìm mọi nghiệm.

---

*Sơ đồ 1: Giải bài toán quay lui  
(tìm mọi nghiệm theo phương pháp giả sai)*

---

```

Khởi trị dãy thoả P
while true
  if (v thoả Q)
    Ghi nhận nghiệm
    Lùi (giả sai)
  end if
  if (hết cách đi)
    Thông báo số nghiệm
    exit
  end if
  if (Tìm được 1 nước đi) Tiến
    else Lùi
  end if
end while

```

---

## Bài 6.1. Các quân Hậu

*Quân Hậu trên bàn cờ Vua có thể diệt quân đối phương theo hàng, theo cột chứa nó hoặc theo đường chéo của hình vuông nhận nó làm đỉnh.*

- Tìm một cách đặt  $N$  quân Hậu trên bàn cờ Vua kích thước  $N \times N$  ô sao cho không quân nào ăn được quân nào.*
- Tìm mọi cách đặt  $N$  quân Hậu theo điều kiện trên.*

*Hiển thị kết quả trên màn hình.*

### Ví dụ



1 2 3 4 5 6 7 8  $v = [1, 5, 8, 6, 3, 7, 2, 4]$

Một phương án đặt Hậu.

Hậu 1 đặt tại dòng 1

Hậu 2 đặt tại dòng 5

Hậu 3 đặt tại dòng 8

Hậu 4 đặt tại dòng 6

Hậu 5 đặt tại dòng 3

Hậu 6 đặt tại dòng 7

Hậu 7 đặt tại dòng 2

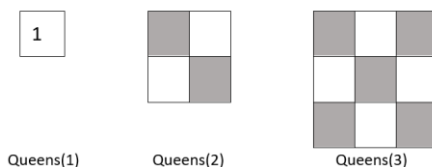
Hậu 8 đặt tại dòng 4

## Queens Version 1. Tìm một nghiệm

Ta biết quân Hậu trên bàn cờ Vua (cờ Quốc tế) kiểm soát hàng dọc, hàng ngang và các đường chéo cùng màu với ô Hậu đứng. Ta giải bài toán tổng quát với  $n$  quân Hậu trên bàn cờ  $n \times n$ . Dễ hiểu là chỉ có thể đặt tối đa  $n$  quân hậu, vì mỗi quân Hậu sẽ kiểm duyệt một hàng dọc. Ta mã hóa mỗi quân Hậu theo hàng dọc  $1, 2, \dots, n$ .

Với  $n = 1$  ta có nghiệm tầm thường: đặt Hậu tại ô duy nhất trên bàn cờ.

Với  $n = 2$  và  $n = 3$  bài toán vô nghiệm.

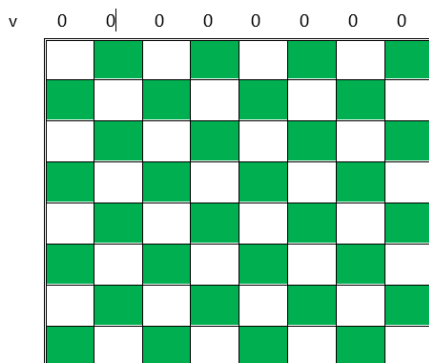


## Tổ chức dữ liệu

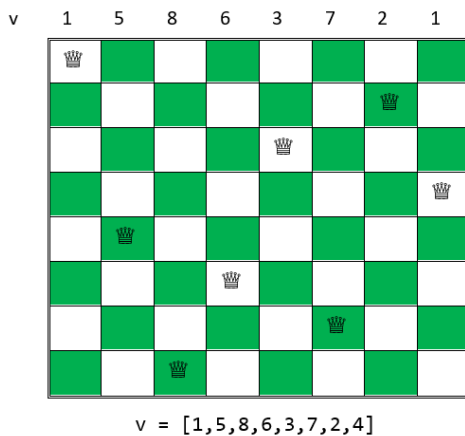
Ta sẽ dùng một mảng  $v$  để ghi nhận vị trí của mỗi Hậu trên bàn cờ. Khi khởi trị với tám Hậu ta đặt các Hậu tại trước vạch xuất phát ngoài bàn cờ, tức là:

$v = [0, 0, 0, 0, 0, 0, 0, 0]$

trong đó ta không sử dụng phần tử  $v[0]$ .



Sau khi tìm được nghiệm đầu tiên ta có



$v = [0, 1, 5, 8, 6, 3, 7, 2, 4]$

với ý nghĩa:

- Đặt Hậu 1 tại dòng 1
- Đặt Hậu 2 tại dòng 5
- Đặt Hậu 3 tại dòng 8
- Đặt Hậu 4 tại dòng 6
- Đặt Hậu 5 tại dòng 3
- Đặt Hậu 6 tại dòng 7
- Đặt Hậu 7 tại dòng 2
- Đặt Hậu 8 tại dòng 4

## Thuật toán

Ta dùng phương pháp quay lui (Back Tracking) để giải bài toán này.

Ý tưởng của phương pháp quay lui là như sau:

- 🦋 Mỗi bước đi ta chọn một khả năng dẫn đến đích.
- 🦋 Nếu có một khả năng như vậy ta tiến thêm một bước theo khả năng đó.
- 🦋 Nếu không, ta sẽ lùi lại một bước, trở về cấu hình của bước trước đó.
- 🦋 Thuật toán sẽ kết thúc khi gặp một trong hai tình huống sau đây:
  - ♦ Đến đích: hiển thị nghiệm
  - ♦ Vô nghiệm: khi đã lùi về giới hạn

---

```

Thuật toán Queens
Đặt n quân Hậu trên bàn cờ n×n
Khởi trị: đặt n Hậu tại vạch 0
Cầm Hậu k = 1
while True do
  if k = 0
    Thông báo vô nghiệm
    return
  end if
end while

```

---

---

```

    if k > n
        Thông báo nghiệm v
        return
    end if
    if Tìm được một nước đi
        Tiến 1 bước: k = k+1
    else
        Lùi 1 bước: k = k-1
    end if
end while

```

---

Sơ đồ trên có ưu điểm là đơn giản và không đệ quy.

Khi khởi trị ta xếp các Hậu ở trước vạch xuất phát ngoài bàn cờ

Ta lần lượt di chuyển từng quân Hậu vào bàn cờ. Gọi k là quân Hậu ta cần di chuyển từ vị trí hiện đứng  $v[k]$  đến vị trí mới i để đặt Hậu k.

```

Khởi trị:
v[i] = 0, i = 1..n
k = 1 # cầm Hậu 1
while True:
    if k < 1:
        Thông báo vô nghiệm
        return
    if k > n:
        print(v)
        return
    if Tìm được một nước đi: # (1)
        Tiến 1 bước
    else:
        Lùi 1 bước

```

*Sơ đồ quay lui Queens(n)*

Theo sơ đồ, ta phải xác định hai điều kiện quan trọng cho tính dừng của thuật toán.

✂ Dừng khi không thành công

✂ Dừng khi đã gặp một nghiệm.

Dễ thấy, khi ta đã đặt xong Hậu thứ n, tức là ta đã xếp xong đủ n Hậu, thì theo thói quen ta sẽ chuyển qua Hậu  $n+1$ . Vậy điều kiện dừng sau khi đã gặp một nghiệm sẽ là  $k > n$ .

Tương tự, sau nhiều lần quay lui ta đưa các Hậu ra khỏi bàn cờ về vạch xuất phát lúc đầu, theo thói quen, ta sẽ cầm Hậu trước đó, tức là Hậu  $k = 0$ . Vậy điều kiện dừng khi vô nghiệm sẽ là  $k < 1$ .

Ta viết hàm `Find(k,n)` xác định điều kiện Tìm được một nước đi cho Hậu  $k$  trên bàn cờ  $n \times n$ .

✂ Dịch dẫn Hậu  $k$  từ dòng đang đứng  $v[k]$  xuống đến dòng cuối cùng  $n$ .

- Nếu tìm được một dòng  $i$  tại đó Hậu  $k$  không bị các Hậu  $1, 2, \dots, k-1$  đặt trước chiếu thì  $i$  là dòng tìm được.
- Ngược lại, với mọi  $i = v[k] + 1 \dots n$  ta không tìm được dòng đặt Hậu  $k$  thì hàm cho ra giá trị  $0$ .

```
int Find(k, n):
    for i = v[k] + 1 .. n:
        if GoodPlace(k, i):
            return i
    return 0
```

Hàm `GoodPlace(k,i)` cho giá trị `True` nếu có thể đặt Hậu  $k$  tại dòng  $i$ , ngược lại, hàm cho ra giá trị `False`.

Ta biết, Hai hậu  $k$  và  $j$  chiếu nhau khi và chỉ khi hai Hậu đứng trên cùng dòng:  $v[k] = v[j]$  hoặc tạo thành hai đỉnh đối diện của một hình vuông cạnh  $k-j$ :  $k-j = \text{abs}(v[j] - i)$ . Để ý rằng do mỗi Hậu chiếm giữ một cột và Hậu  $j$  được đặt trước Hậu  $k$  nên  $j < k$ .

```
bool GoodPlace(k, i):
    for j = 1:k:
        if v[j] = i or k - j = abs(v[j] - i):
            return false
    return true
```

## Chương trình C++

```
// Queens, Ver. 1: Tìm một nghiệm
#include <bits/stdc++.h>
using namespace std;
```

```

const int MN = 20;
int v[MN]; // Hau k dat tai dong v[k], 1 <= k <= n

using namespace std;

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

// Hien thi nghiem: cac dong dat Hau
Print(int n){
    for (int i = 1; i <= n; ++i)
        cout << " " << v[i];
}

// Hau k duoc dat tai dong i ?
bool GoodPlace(int k, int i) {
    // xet cac Hau da dat truoc Hau k
    for (int j = 1; j < k; ++j) {
        if (v[j] == i || k - j == abs(v[j] - i))
            // Hau j chieu Hau k
            return false;
    }
    return true;
}

// Tim dong dat Hau k
// tren ban co nxn
// day Hau k xuong cac dong duoi
// kiem tra co the dat Hau k tai dong do?
int Find(int k, int n) {
    for (int i = v[k] + 1; i <= n; ++i)
        if (GoodPlace(k, i)) // Hau k duoc dat tai dong i ?
            return i;
    return 0;
}

void Queens(int n) {
    cout << " Queens of " << n << ": ";
    if (n >= MN) {
        cout << " Large chess board size: " << n;
        return;
    }
    if (n < 1) {
        cout << " No solution.";
        return;
    }
    if (n == 1) {
        cout << 1;
        return;
    }
}

```

```

    }
    // n > 1: Dat n Hau ngoai ban co
    for (int i = 1; i <= n; ++i)
        v[i] = 0;
    int k = 1; // cam Hau k
    while (true) {
        if (k < 1) {
            cout << " No solution.";
            return;
        }
        if (k > n) { // nghiem
            Print(n);
            return;
        }
        v[k] = Find(k, n); // Tim dong dat Hau k
        if (v[k] > 0) // neu tim duoc
            k += 1; // Tien: xet Hau ke tiep: k + 1
        // neu ko: Lui: k - 1
        else k -= 1;
    }
}

void Run() {
    for (int n = -1; n < MN; ++n) {
        Queens(n); Go();
    }
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Queens of -1: No solution. ?
Queens of 0: No solution. ?
Queens of 1: 1 ?
Queens of 2: No solution. ?
Queens of 3: No solution. ?
Queens of 4: 2 4 1 3 ?
Queens of 5: 1 3 5 2 4 ?
Queens of 6: 2 4 6 1 3 5 ?
Queens of 7: 1 3 5 7 2 4 6 ?
Queens of 8: 1 5 8 6 3 7 2 4 ?

```

## Queens Version 2.

Ta cải tiến hàm Queens(n) với các điều chỉnh sau đây:



✂ Bằng những quan sát đơn giản, ta thấy với kích thước bàn cờ  $n$  quá lớn thì thuật toán quay lui sẽ không khả dụng. Ta chọn giới hạn trên cho  $n$  là 20, tức là ta chỉ xét số Hậu tối đa là 19.

- Nếu  $n < 1$ : ta bỏ qua, xem như vô nghiệm;
- $n = 1$  có một nghiệm duy nhất [1];
- $n = 2$  và  $n = 3$ : vô nghiệm;
- $n = 4$  có nghiệm [2, 4, 1, 3];
- Từ  $n = 5$  trở đi ta sẽ xuất phát từ nghiệm  $n = 4$  và bắt đầu duyệt quân Hậu  $k = 5$  trở đi.

✂ Để tăng tốc độ cho hàm Find ta bổ sung thêm ba mảng:

- Mảng  $row$  đánh dấu những dòng đã đặt Hậu:  $row[i] = 0$  cho biết trên dòng  $i$  đã có Hậu,  $row[i] = 1$  cho biết dòng này còn trống.
- Mảng  $x$  đánh dấu đường chéo phải gồm các ô cùng màu với ô Hậu  $k$  đang đứng:

$x[n+i-k] = 0$  cho biết đường chéo phải  $n+i-k$  bị Hậu  $k$  trên dòng  $i$  kiểm soát.

$x[n+i-k] = 1$  cho biết đường chéo này chưa bị Hậu nào kiểm soát.

- Mảng  $y$  đánh dấu đường chéo trái gồm các ô cùng màu với ô Hậu  $k$  đang đứng:

$y[i+k] = 0$  cho biết đường chéo trái  $i+k$  bị Hậu  $k$  trên dòng  $i$  kiểm soát.

$y[i+k] = 1$  cho biết trên đường chéo này còn trống.

	1	2	3	4	5	6	7	8
1			x				y	
2				x		y		
3	r	r	r	r	*	r	r	r
4				y		x		
5			y				x	
6		y						x
7	y							
8								

Hậu  $k = 5$  tại dòng  $i = 3$   
trên bàn cờ  $n = 8$  sẽ kiểm soát:  
Dòng 3:  $row[3] = 0$ ,  
Chéo phải  $x[8-3+5] = x[10] = 0$   
Chéo trái  $y[3+5] = y[8] = 0$

Activ

Tóm lại, nếu Hậu  $k$  đứng tại dòng  $i$  thì Hậu sẽ kiểm soát  
dòng  $i$ :  $row[i] = 0$   
chéo phải  $n+i-k$ :  $x[n+i-k] = 0$

chéo trái  $i+k$ :  $y[i+k] = 0$

### Ví dụ

Với  $n = 8$  ta cần ba mảng định vị các dòng (row), đường chéo phải (x) và đường chéo trái (y) cho các Hậu từ 1..8 như sau:

#### Chéo trái

Chéo trái y, $n = 8$			dòng			$i + k$		
$i$	1	2	3	4	5	6	7	8
Hậu 1	2	3	4	5	6	7	8	9
Hậu 2	3	4	5	6	7	8	9	10
Hậu 3	4	5	6	7	8	9	10	11
Hậu 4	5	6	7	8	9	10	11	12
Hậu 5	6	7	8	9	10	11	12	13
Hậu 6	7	8	9	10	11	12	13	14
Hậu 7	8	9	10	11	12	13	14	15
Hậu 8	9	10	11	12	13	14	15	16

#### Chéo phải

Chéo phải x, $n = 8$			dòng			$n+i-k$		
$i$	1	2	3	4	5	6	7	8
$n+i$	9	10	11	12	13	14	15	16
Hậu 1	8	9	10	11	12	13	14	15
Hậu 2	7	8	9	10	11	12	13	14
Hậu 3	6	7	8	9	10	11	12	13
Hậu 4	5	6	7	8	9	10	11	12
Hậu 5	4	5	6	7	8	9	10	11
Hậu 6	3	4	5	6	7	8	9	10
Hậu 7	2	3	4	5	6	7	8	9
Hậu 8	1	2	3	4	5	6	7	8

Khi nhắc Hậu  $k$  khỏi dòng  $i$  ta cần giải phóng các mảng định vị:

$$x[n+i-k] = y[i+k] = \text{row}[i] = 1$$

Giá trị 1 cho biết các hướng này hiện đang trống.

Khi đặt Hậu  $k$  vào dòng  $j$  ta cần đánh dấu các mảng định vị:

```
x[n+j-k] = y[j+k] = row[j] = 0
```

Giá trị 0 cho biết các hướng này hiện đang bị chiếm.

Nhờ ba mảng này, việc tìm một vị trí (dòng) còn trống để đặt Hậu k trở nên đơn giản:

```
def Find(k, n):
    for i in range(v[k] + 1, n + 1):
        if x[n + i - k] and y[i + k] and row[i]:
            return i
    return 0
```

trong đó giá trị k cho biết Hậu k hiện di chuyển trên cột k, v[k] là dòng Hậu k đang đứng, n là kích thước của bàn cờ.

Với n = 4 ta hiển thị ngay kết quả là nghiệm có sẵn: v = [2, 4, 1, 3].

Nếu n > 4 ta xuất phát từ nghiệm v(4) = [2, 4, 1, 4] và bắt đầu thuật toán quay lui với k = 5 trở đi.

Trước hết ta cần định vị nghiệm v(4).

```
v = v + [0] * (n - 3) # reset v = [0,2,4,1,3,0,...]
row = [1] * (n + 1) # reset row
x = [1] * (2 * n + 1) # reset right diagonal x all 1
y = [1] * (2 * n + 1) # reset left diagonal all 1
for k in range(1, 5): # set v(4)
    j = v[k]
    x[n + j - k] = y[j + k] = row[j] = 0
```

Ta viết thêm hàm Verify(n) kiểm tra các nghiệm xem việc đặt Hậu có đúng luật không. Hàm này có độ phức tạp  $O(n^2)$  với kích thước bàn cờ n khoảng vài chục thì không đáng kể.

```
def Verify(n):
    print('Verify:', v[1:(n + 1)])
    for k in range(1, n + 1):
        for j in range(1, k):
            if v[j] == v[k]: # in the same row
                print('ERROR at', j, k)
                return
            if k - j == abs(v[j] - v[k]): # in the same diagonal
                print('ERROR at', j, k)
                return
    print('CORRECT.')
```

Ta đặt thêm biến step để đếm xem thuật toán quay lui cần bao nhiêu bước lặp.

Ví dụ, với 19 Hậu ta cần đến 5068 bước tiến và lùi.

## Chương trình C++

```
// Queens, Ver. 2: Tìm một nghiệm
// Hậu k trên dòng i sẽ kiểm soát
// row[i]
// các chéo phải x[n+i-k]
// các chéo trái y[i+k]
// Đặt Hậu k tại dòng i: r[i] = x[n+i-k] = y[i+k] = false;
// Nhắc Hậu k khỏi dòng i: r[i] = x[n+i-k] = y[i+k] = true;

#include <bits/stdc++.h>

using namespace std;
const int MN = 20;
const int MN2 = 2*MN;
int v[MN]; // Hậu k đặt tại dòng v[k], 1 ≤ k ≤ n
bool r[MN];
bool x[MN2];
bool y[MN2];

using namespace std;

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

// Hiện thị nghiệm: các dòng đặt Hậu
void Print(int n){
    for (int i = 1; i ≤ n; ++i)
        cout << " " << v[i];
}

void Print(bool b[], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i ≤ c; ++i)
        cout << " " << b[i];
}

// Tìm dòng đặt Hậu k
// trên bàn cờ nxn
// đây Hậu k xuống các dòng dưới
// kiểm tra có thể đặt Hậu k tại dòng đó?
int Find(int k, int n) {
    for (int i = v[k] + 1; i ≤ n; ++i)
        if (r[i] && x[n+i-k] && y[i+k])
            return i;
    return 0;
}

void Verify(int n) {
```

```

        cout << "\n Verify: ";
        for (int k = 1; k <= n; ++k) {
            // xet cac Hau j dat truc Hau k
            for (int j = 1; j < k; ++j) {
                if (v[j] == v[k]) {
                    cout << " ERROR at " << j << " " << k;
                    return;
                }
                if (k - j == abs(v[j] - v[k])) {
                    cout << " ERROR at " << j << " " << k;
                    return;
                }
            }
        }
        cout << " CORRECT.";
    }

    void NhacHau(int k, int n) {
        if (v[k] > 0)
            r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = true;
    }

    void DatHau(int k, int n) {
        r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = false;
    }

    void Queens(int n) {
        cout << " Queens of " << n << ": ";
        if (n == 1) {
            cout << 1;
            return;
        }
        if (n >= MN) {
            cout << " Large chess board size: " << n;
            return;
        }
        if (n < 4) { // n = 2, 3: vo nghiem
            cout << " No solution.";
            return;
        }
        for (int k = 1; k <= n; ++k) {
            v[k] = 0;
            r[k] = true;
            for (int i = 1; i <= n; ++i) {
                x[n+i-k] = y[i+k] = true;
            }
        }
        // n >= 4: xuat phat tu 4 Hau: 2 4 1 3
        v[1] = 2; v[2] = 4; v[3] = 1; v[4] = 3;

        for (int k = 1; k <= 4; ++k) {
            DatHau(k, n);
        }
    }

```

```

int k = 5; // cam Hau k hien dung tai dong 0
int step = 0;
while (true) {
    ++step;
    // Cam Hau k
    if (k < 1) {
        cout << " No solution.";
        return;
    }
    if (k > n) { // nghiem
        Print(n);
        Verify(n);
        cout << "\n Step = " << step;
        return;
    }
    // nhac Hau k
    NhacHau(k, n);
    v[k] = Find(k, n); // Tim dong dat Hau k
    if (v[k] > 0) { // neu tim duoc
        // Chap nhan Hau k tai dong v[k]
        DatHau(k, n);
        k += 1; // xet Hau ke tiep: k + 1
    }
    // neu ko: xet Hau truoc: k - 1
    else k -= 1;
}
}

void Run() {
    for (int n = -1; n < MN; ++n) {
        Queens(n); Go();
    }
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Queens of -1: No solution. ?
Queens of 0: No solution. ?
Queens of 1: 1 ?
Queens of 2: No solution. ?
Queens of 3: No solution. ?
Queens of 4: 2 4 1 3
Verify: CORRECT.
Step = 1 ?
Queens of 5: 2 4 1 3 5
Verify: CORRECT.

```

```

Step = 2 ?
Queens of 6:  2 4 6 1 3 5
Verify:  CORRECT.
Step = 9 ?
Queens of 7:  2 4 1 7 5 3 6
Verify:  CORRECT.
Step = 8 ?
Queens of 8:  2 4 6 8 3 1 7 5
Verify:  CORRECT.
Step = 45 ?
Queens of 9:  2 4 1 7 9 6 3 5 8
Verify:  CORRECT.
Step = 32 ?
Queens of 10: 2 4 6 8 10 1 3 5 7 9
Verify:  CORRECT.
Step = 217 ?
Queens of 11: 2 4 1 7 10 3 11 8 5 9 6
Verify:  CORRECT.
Step = 122 ?
Queens of 12: 2 4 1 8 11 9 12 3 5 7 10 6
Verify:  CORRECT.
Step = 561 ?
Queens of 13: 2 4 1 7 9 12 6 13 3 5 11 8 10
Verify:  CORRECT.
Step = 660 ?
Queens of 14: 2 4 1 8 13 9 12 14 5 10 6 11 3 7
Verify:  CORRECT.
Step = 6433 ?
Queens of 15: 2 4 1 7 5 11 14 12 15 3 8 6 9 13 10
Verify:  CORRECT.
Step = 6232 ?
Queens of 16: 2 4 1 7 9 14 12 15 6 16 11 5 3 13 10 8
Verify:  CORRECT.
Step = 33871 ?
Queens of 17: 2 4 1 3 8 11 13 16 5 17 15 6 10 14 7 9 12
Verify:  CORRECT.
Step = 10920 ?
Queens of 18: 2 4 1 3 8 12 14 16 18 6 15 17 10 5 7 9 11 13
Verify:  CORRECT.
Step = 55133 ?
Queens of 19: 2 4 1 3 5 9 13 15 17 19 7 16 18 11 6 8 10 12 14
Verify:  CORRECT.
Step = 5068 ?

T h e   E n d

```

### Queens Version 3. Lưu nghiệm

Mấy năm gần đây, trong các đề thi quốc tế thường đề xuất các test theo quy trình test sau dựa vào các test trước. Đây là ý tưởng thường được vận dụng trong các bộ tìm kiếm trên mạng. Giả sử bạn đã tìm được các bài Toán Tin trong các kỳ thi Châu Á, nay bạn muốn tham khảo các bài Toán Tin Quốc tế, thì một bộ tìm kiếm thông minh sẽ không

phải duyệt tìm trên Internet từ đầu mà chỉ tìm kiếm tiếp những trang ngoài Châu Á để bổ sung kết quả cho bạn.

Trở lại với bài toán Queens. Giả sử bạn gọi liên tiếp hai hàm Queens(10), sau đó là Queens(12).

Nếu tại bước trước bạn đã lưu lại nghiệm  $v(10)$  của Queens(10) thì khi cần tính Queens(12) bạn chỉ cần phát triển tiếp Queens(10) để duyệt với Hậu  $k = 11$ , rồi  $k = 12$ . Trong nhiều trường hợp việc lưu lại các nghiệm có thể giảm đáng kể thời gian thực hiện.

Version 3 sẽ tập trung giải quyết vấn đề này.

Chiến lược đại thể sẽ như sau:

Sau mỗi test Queens(m) ta sẽ lưu lại nghiệm  $v(m)$ , trong đó  $v(m)$  là ký hiệu nghiệm của Queens(m).

Khi gặp test mới, Queens(n), nếu  $v(n)$  đã có sẵn trong kho thì ta chỉ việc hiển thị  $v(n)$ . Ngược lại, ta tìm trong kho nghiệm một nghiệm  $v(m)$  gần nhất với  $n$ ,  $m \leq n$  rồi phát triển  $v(m)$  để thu được  $v(n)$ , sau đó sẽ nạp  $v(n)$  vào kho.

### Cấu trúc dữ liệu

Ta biết, mỗi nghiệm Queens(n) là một vector (mảng)  $v[1..n]$ , ví dụ, Queens(4) cho ta nghiệm  $v = [2, 4, 1, 3]$ . Ta tổ chức kho lưu nghiệm là một từ điển vv với hai trường:

- ☞ Trường khóa  $key = n$  là số Hậu, tức là kích thước bàn cờ.
- ☞ Trường value là vector nghiệm  $v$ .

Ví dụ

`vv[4] = [0, 2, 4, 1, 3] # n = 4 có nghiệm [2,4,1,3]`  
cho biết Queens(4) có nghiệm là

`vv[4][1:5] = [2, 4, 1, 3] # n = 4 có nghiệm [2,4,1,3]`  
còn

`vv[3] = [0, -1] # vô nghiệm`  
cho biết Queens(3) vô nghiệm.

Như vậy,  $vv[][]$  là một mảng danh sách chứa danh sách biến thiên về kích thước, cụ thể là mỗi  $vv[n]$  là một mảng một chiều gồm  $n$  phần tử, do đó  $vv[i]$  và  $vv[j]$ ,  $i \neq j$  sẽ có kích thước khác nhau.

Dấu hiệu để biết một Queens(n) khi đó sẽ dựa vào phần tử thứ hai của  $vv[n]$ , cụ thể là:

`Queens(n) có nghiệm khi và chỉ khi vv[n][1] > 0`  
Nếu Queens(n) có nghiệm thì nghiệm đó sẽ là  $vv[n][1:n+1]$ .

Ta cũng nạp trước vào kho vv một số nghiệm ban đầu như sau:

`vv[0] = [0, -1] # vô nghiệm khi n = 0`



```

vv[1] = [0, 1]    # n = 1 có nghiệm [1]
vv[2] = [0, -1]   # n = 2: vô nghiệm
vv[3] = [0, -1]   # n = 3: vô nghiệm
vv[4] = [0, 2, 4, 1, 3] # n = 4 có nghiệm [2,4,1,3]

```

Chương trình sẽ được test tự động theo hai pha với các giá trị  $n$  từ 0 đến  $MN-1$   
 $= 25-1 = 24$ .

Pha 1 sẽ test tự động Queens( $n$ ) với  $5 \leq n \leq 10$  với mục đích nạp vào kho một số nghiệm nhỏ.

Pha 2 sẽ test tự động Queens( $n$ ) với  $0 \leq n \leq 24$  để khảo sát hoạt động của chức năng quản lý kho.

## Chương trình C++

```

// Queens, Ver. 3: Lưu nghiệm
// Đặt Hậu k tại dòng i: r[i] = x[n+i-k] = y[i+k] = false;
// Nhắc Hậu k khỏi dòng i: r[i] = x[n+i-k] = y[i+k] = true;

#include <bits/stdc++.h>

using namespace std;

const int MN = 40; // Tối đa n = 39 Hậu
const int MN2 = 2*MN;

int vv[MN][MN]; // kho nghiệm
int v[MN]; // Hậu k đặt tại dòng v[k], 1 <= k <= n
bool r[MN]; // kiểm soát dòng
bool x[MN2]; // kiểm soát chéo phải
bool y[MN2]; // kiểm soát chéo trái

using namespace std;

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

// Hiện thị nghiệm: các dòng đặt Hậu
void Print(int n, const char * msg = "") {
    cout << msg;
    for (int i = 1; i <= n; ++i)
        cout << " " << v[i];
}

```

```

void Print(bool b[], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i)
        cout << " " << b[i];
}

// Tim dong dat Hau k
// tren ban co nxn
// day Hau k xuong cac dong duoi
// kiem tra co the dat Hau k tai dong do?
int Find(int k, int n) {
    for (int i = v[k] + 1; i <= n; ++i)
        if (r[i] && x[n+i-k] && y[i+k])
            return i;
    return 0;
}

void Verify(int n) {
    cout << "\n Verify: ";
    for (int k = 1; k <= n; ++k) {
        // xet cac Hau j dat truc Hau k
        for (int j = 1; j < k; ++j) {
            if (v[j] == v[k]) {
                cout << " ERROR at " << j << " " << k;
                return;
            }
            if (k - j == abs(v[j] - v[k])) {
                cout << " ERROR at " << j << " " << k;
                return;
            }
        }
    }
    cout << " CORRECT.";
}

void LuuNghiem(int n) {
    vv[n][0] = 1;
    for (int i = 1; i <= n; ++i)
        vv[n][i] = v[i];
}

void Queens(int n) {
    cout << " Queens of " << n << ": ";
    if (n >= MN) {
        cout << " Large chess board size: " << n;
        return;
    }
    if (n < 1) {
        cout << " No solution.";
        return;
    }
    if (n == 1) {
        cout << 1;
    }
}

```

```

        return;
    }
    if (n < 4) { // n = 2, 3: vo nghiem
        cout << " No solution.";
        return;
    }
    // lau cac bien kiem soat huong
    for (int k = 1; k <= n; ++k) {
        v[k] = 0;
        r[k] = true;
        for (int i = 1; i <= n; ++i) {
            x[n+i-k] = y[i+k] = true;
        }
    }
    int d;
    // Search(n);
    for (d = n; d > 3; --d)
        if (vv[d][0] == 1) break;
    // d la nghiem gan nhat
    for (int k = 1; k <= n; ++k) {
        v[k] = vv[d][k]; // nghiem d
        // Dat cac Hau
        r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = false;
    }
    cout << "\n  Start from solution " << d;
    Print(d, ": ");
    int k = d+1; // cam Hau k hien dung tai dong 0
    int step = 0;
    while (true) {
        ++step;
        // Cam Hau k
        if (k < 1) {
            cout << "  No solution.";
            return;
        }
        if (k > n) { // nghiem
            Print(n, "\n  Nghiem: ");
            Verify(n);
            cout << "\n  Step = " << step;
            // Luu mghiem
            if (vv[n][0] == 0) LuuNghiem(n);
            return;
        }
        // nhac Hau k
        if (v[k] > 0)
            r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = true;
        v[k] = Find(k, n); // Tim dong dat Hau k
        if (v[k] > 0) { // neu tim duoc
            // Chap nha Hau k tai dong v[k]
            r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = false;
            k += 1; // xet Hau ke tiep: k + 1
        }
        // neu ko: xet Hau truoc: k - 1
    }

```

```

        else k -= 1;
    }
}

// Khoi tri kho luu nghiem
void Init() {
    memset(vv, 0, sizeof(vv));
    vv[1][0] = 1; // n = 1 co nghiem [1]
    vv[1][1] = 1;
    vv[4][0] = 1; // n = 4 co nghiem [2,4,1,3]
    vv[4][1] = 2; vv[4][2] = 4;
    vv[4][3] = 1; vv[4][4] = 3;
}

void Run() {
    // lan luot goi theo n sau day
    int a[] = {3, 5, 8, 5, 10, 13, 17, 8, 20, 12, 20, 25};
    int m = sizeof(a) / sizeof(int);
    Init();
    for (int i = 0; i < m; ++i) {
        Queens(a[i]); Go();
    }
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Queens of 3: No solution. ?
Queens of 5:
    Start from solution 4:  2 4 1 3
    Nghiem:  2 4 1 3 5
    Verify:  CORRECT.
    Step = 2 ?
Queens of 8:
    Start from solution 5:  2 4 1 3 5
    Nghiem:  3 5 2 8 1 7 4 6
    Verify:  CORRECT.
    Step = 336 ?
Queens of 5:
    Start from solution 5:  2 4 1 3 5
    Nghiem:  2 4 1 3 5
    Verify:  CORRECT.
    Step = 1 ?
Queens of 10:
    Start from solution 8:  3 5 2 8 1 7 4 6
    Nghiem:  3 5 8 2 9 7 1 4 6 10
    Verify:  CORRECT.
    Step = 113 ?

```

```

Queens of 13:
  Start from solution 10:  3 5 8 2 9 7 1 4 6 10
  Nghiem:  3 5 8 2 9 12 1 7 4 6 11 13 10
  Verify:  CORRECT.
  Step = 40 ?
Queens of 17:
  Start from solution 13:  3 5 8 2 9 12 1 7 4 6 11 13 10
  Nghiem:  3 5 8 2 9 12 14 16 1 6 4 7 10 13 15 17 11
  Verify:  CORRECT.
  Step = 757 ?
Queens of 8:
  Start from solution 8:  3 5 2 8 1 7 4 6
  Nghiem:  3 5 2 8 1 7 4 6
  Verify:  CORRECT.
  Step = 1 ?
Queens of 20:
  Start from solution 17:  3 5 8 2 9 12 14 16 1 6 4 7 10 13 15 17
11
  Nghiem:  3 5 8 2 9 12 14 16 18 20 4 7 13 11 1 17 6 10 15 19
  Verify:  CORRECT.
  Step = 1484 ?
Queens of 12:
  Start from solution 10:  3 5 8 2 9 7 1 4 6 10
  Nghiem:  3 5 8 2 9 12 6 1 10 7 11 4
  Verify:  CORRECT.
  Step = 47 ?
Queens of 20:
  Start from solution 20:  3 5 8 2 9 12 14 16 18 20 4 7 13 11 1 17
6 10 15 19
  Nghiem:  3 5 8 2 9 12 14 16 18 20 4 7 13 11 1 17 6 10 15 19
  Verify:  CORRECT.
  Step = 1 ?
Queens of 25:
  Start from solution 20:  3 5 8 2 9 12 14 16 18 20 4 7 13 11 1 17
6 10 15 19
  Nghiem:  3 5 8 2 9 12 14 16 18 21 1 22 25 6 4 7 13 11 20 17 15 10
23 19 24
  Verify:  CORRECT.
  Step = 39206 ?
T h e   E n d

```

#### Queens Version 4. Tìm mọi nghiệm

Thay đổi ít nhất nhưng hưởng lợi nhiều nhất

Như ta biết có thể có đến 92 cách xếp Hậu cho bài Queens(8).

Ta chỉ cần sửa chút ít sơ đồ tìm một nghiệm để thu được mọi nghiệm. Ý tưởng là như sau:

Phương pháp giả sai

Sau khi tìm được một nghiệm và hiển thị nghiệm đó, nghĩa là sau khi đã đặt

xong Hậu thứ  $n$ , ta coi như không xếp được Hậu  $n$  này, do đó ta phải quay lui về Hậu  $n-1$  để tìm tiếp nghiệm khác.

```

Khởi trị
while True:
    if ở dưới vạch xuất phát:
        Thông báo vô nghiệm
        return
    if Đến đích:
        Thông báo nghiệm
        Lui
    if Tìm được một nước đi:
        Tiến 1 bước
    else:
        Lui 1 bước

```

*Sơ đồ thuật toán quay lui tìm mọi nghiệm*

Phương án đơn giản dưới đây nhằm minh họa phương pháp giả sai.

Hàm Queens( $n$ ,  $sn = 1$ ) có hai tham biến:

- ✧  $n$  là kích thước bàn cờ.
- ✧  $sn$  là tham biến tùy chọn ấn định số nghiệm cần tìm:  $sn = 1$  cho biết chỉ cần tìm một nghiệm; ngược lại, khi  $sn > 1$ , thì chương trình sẽ tìm tất cả các nghiệm.
- ✧ Trong hàm còn một biến  $c$  đếm số nghiệm.

## Chương trình C++

```

// Queens, Ver. 4: Tim moi nghiem
// Hau k tren dong i se kiem soat
// row[i]
// cac cheo phai x[n+i-k]
// cac cheo trai y[i+k]
// Dat Hau k tai dong i: r[i] = x[n+i-k] = y[i+k] = false;
// Nhac Hau k khoi dong i: r[i] = x[n+i-k] = y[i+k] = true;

#include <iostream>
#include <windows.h>
#include <bits/stdc++.h>

using namespace std;
const int MN = 20;
const int MN2 = 2*MN;
int v[MN]; // Hau k dat tai dong v[k], 1 <= k <= n
bool r[MN];
bool x[MN2];
bool y[MN2];

using namespace std;

```

```

void Go() {
    cout << " ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

// Hien thi nghiêm: cac dong dat Hau
void Print(int sn, int n){
    cout << "\n Nghiêm " << sn << ": ";
    for (int i = 1; i <= n; ++i)
        cout << " " << v[i];
}

// Tim dong dat Hau k
// tren ban co nxn
// day Hau k xuong cac dong duoi
// kiem tra co the dat Hau k tai dong do?
int Find(int k, int n) {
    for (int i = v[k] + 1; i <= n; ++i)
        if (r[i] && x[n+i-k] && y[i+k])
            return i;
    return 0;
}

void Queens(int n, int songhiem = 1) {
    cout << "\n Queens of " << n << ": ";
    int c = 0; // dem so nghiêm
    if (songhiem == 1)
        cout << " Tim 1 nghiêm.";
    else
        cout << " Tim moi nghiêm.";
    if (n == 1) {
        v[1] = 1;
        c = 1;
        Print(c, n);
        return;
    }
    if (n >= MN) {
        cout << " Large chess board size: " << n;
        return;
    }
    if (n < 4) { // n = 2, 3: vo nghiêm
        cout << " No solution.";
        return;
    }
    for (int k = 1; k <= n; ++k) {
        v[k] = 0;
        r[k] = true;
        for (int i = 1; i <= n; ++i) {
            x[n+i-k] = y[i+k] = true;
        }
    }
}

```

```

int k = 1; // cam Hau k hien dung tai dong 0
while (true) {
    // Cam Hau k
    if (k < 1) {
        if (c == 0)
            cout << " \n No solution.";
        else
            cout << " \n Total " << c << " solution(s.)";
            return;
    }
    if (k > n) { // nghiem
        ++c;
        Print(c, n);
        if (songhiem == 1) return;
        k = n;
    }
    // nhac Hau k
    if (v[k] > 0)
        r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = true;
    v[k] = Find(k, n); // Tim dong dat Hau k
    if (v[k] > 0) { // neu tim duoc
        // Chap nhan Hau k tai dong v[k]
        r[v[k]] = x[n+v[k]-k] = y[v[k]+k] = false;
        ++k; // xet Hau ke tiep: k + 1
    }
    // neu ko: xet Hau truoc: k - 1
    else --k;
}
}

void Run() {
    Queens(8); // Tim 1 nghiem
    Queens(8, 2); // Tim moi nghiem
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Bài 6.2. Từ chuẩn

Một từ loại  $M$  là một dãy các chữ số, mỗi chữ số nằm trong khoảng từ 1 đến  $M$ . Số lượng các chữ số có mặt trong một từ được gọi là chiều dài của từ đó. Từ loại  $M$  được gọi là từ chuẩn nếu nó không chứa hai đoạn liền nhau mà giống nhau. Với giá trị  $N$  cho trước, hiển thị trên màn hình một từ chuẩn loại 3 có chiều dài  $N$ .

### Ví dụ

1213123 là từ chuẩn loại 3, chiều dài 7.



1213213 không phải là từ chuẩn vì nó chứa liên tiếp hai từ con giống nhau là 213.

Tương tự, 12332 không phải là từ chuẩn vì chứa liên tiếp hai từ con giống nhau là 3.

## Thuật toán

Ta dùng một string  $v$  để lưu từ cần tìm. Tại mỗi bước  $i$  ta xác định giá trị  $v[i]$  trong khoảng  $1..m$  sao cho  $v[0..i-1]$  là từ chuẩn.

Điều kiện P:  $v[0..i-1]$  là từ chuẩn;

Điều kiện Q: Dừng thuật toán theo một trong hai tình huống sau đây:

- ♦ nếu  $i \geq n$  thì bài toán có nghiệm  $v[0..i-1]$ .
- ♦ nếu  $i < 0$  thì bài toán vô nghiệm.

Để kiểm tra tính chuẩn của từ  $v[1..i]$ , ta lưu ý rằng từ  $v[1..i-1]$  đã chuẩn (tính chất P), do đó chỉ cần khảo sát các cặp từ có chứa  $v[i]$ , cụ thể là khảo sát các cặp từ có chiều dài  $k$  đứng cuối từ  $v$ ,  $k = 1..i/2$ .

---

```

Thuật toán Từ Chuẩn
v[0] = "10...0" // xuất phát từ string 1
                // n-1 số 0
k = 1 // xét kí tự v[k]
while true do
  if k < 1
    Thông báo: vô nghiệm
    return
  end if
  if k > n
    Thông báo nghiệm
    return
  end if
  if Tìm được một c ∈ 1..m: v+c là từ chuẩn
    k = k + 1 // tiến
  else
    k = k - 1 // Lùi
  end if
end while

```

---

## Ví dụ

```

m = 1..3
n = 10
Xuất phát: v = 1000000000 // chuẩn

Find c: v = |1c|00000000 là từ chuẩn?
c = 2: v = |12|00000000. Tìm được: Tiến

Find c: |12c|00000000 là từ chuẩn?
c = 1: |121|00000000. Tìm được: Tiến

Find c: |121c|00000000 là từ chuẩn?
c = 3: |1213|00000000. Tìm được: Tiến

```

```

Find c: |1213c|00000 là từ chuẩn?
c = 1: |12131|00000. Tìm được: Tiến

Find c: |12131c|0000 là từ chuẩn?
c = 2: |121312|0000. Tìm được: Tiến

Find c: |121312c|000 là từ chuẩn?
c = 1: |1213121|000. Tìm được: Tiến

Find c: |1213121c|00 là từ chuẩn?
c = ?: |1213121?|00. Không tìm được: Lùi

Find c>1: |121312c|00 là từ chuẩn?
c = 3: |1213123|000. Tìm được: Tiến.

Find c: |1213123c|00 là từ chuẩn?
c = 1: |12131231|00. Tìm được: Tiến.

Find c: |12131231c|0 là từ chuẩn?
c = 3: |121312313|0. Tìm được: Tiến.

Find c: |121312313c| là từ chuẩn?
c = 2: |1213123132|. Tìm được: len = 10. Stop

```

## Chương trình C++

```

// Tu chuan
#include<bits/stdc++.h>
using namespace std;

int n;
char * v;

using namespace std;

void Go() {
    cout << "\n Muon thoat, bam [.] ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

void Print(const char * msg = "") {
    cout << msg;
    for (int i = 0; i < n; ++i)
        cout << (int)v[i];
}

// v[i..k] = doan sat truoc ?
bool Eq(int i, int k) {
    int j = i-1;

```

```

        for (; k >= i; --k) {
            if (v[k] != v[j]) return false;
            --j; // v[k] == v[ii]
        }
        return true;
    }

    // 0-----k là tu chuan
    bool Chuan(int k) {
        if (k == 0) return true;
        // k >= 1: v[k-1] ? v[k]
        if (v[k-1] == v[k]) return false;
        int k2 = k/2;
        for (int i = k-1; i > k2; --i) {
            if (Eq(i,k)) return false;
        }
        return true;
    }

    // tìm giá trị v[k]
    char Find(int k) {
        for (++v[k]; v[k] <= 3; ++v[k]) {
            if (Chuan(k)) return v[k];
        }
        v[k] = 0;
        return v[k];
    }

    // Tìm 1 nghiệm
    void TuChuan(int inpn) {
        n = inpn;
        int lap = 0; // đếm số bước lặp
        cout << "\n Tu chuan len = " << n;
        v = new char[n+1];
        memset(v, 0, (n+1)*sizeof(char));
        v[0] = 1;
        Print("\n Init: ");
        int k = 1;
        while (true) {
            ++lap;
            if (k < 1) { // vô nghiệm
                cout << "\n Vô nghiệm.";
                cout << "\n số bước lặp: " << lap;
                return;
            }
            if (k >= n) {
                Print("\n Nghiệm: ");
                cout << "\n số bước lặp: " << lap;
                return;
            }
            v[k] = Find(k);
            if (v[k] > 0) v[++k] = 0; // tiếp
            else --k; // lùi
        }
    }

```

```

    } // while
} // Tu chuan

void Run() {
    for (int n = 1; n <= 1000000; ++n) {
        TuChuan(n);    Go();
    }
}

main() {
    Run();
    cout << "\n T h e    E n d";
    return 0;
}

```

## Output

```

Tu chuan len = 1
Init: 1
Nghiem: 1
so buoc lap: 1
Muon thoat, bam [.] ?
Tu chuan len = 2
Init: 10
Nghiem: 12
so buoc lap: 2
Muon thoat, bam [.] ?
Tu chuan len = 3
Init: 100
Nghiem: 121
so buoc lap: 3
Muon thoat, bam [.] ?
Tu chuan len = 4
Init: 1000
Nghiem: 1213
so buoc lap: 4
Muon thoat, bam [.] ?
Tu chuan len = 5
Init: 10000
Nghiem: 12131
so buoc lap: 5
Muon thoat, bam [.] ?
Tu chuan len = 6
Init: 100000
Nghiem: 121312
so buoc lap: 6
Muon thoat, bam [.] ?
Tu chuan len = 7
Init: 1000000
Nghiem: 1213121
so buoc lap: 7
Muon thoat, bam [.] ?

```

```

Tu chuan len = 8
Init: 10000000
Nghiem: 12131231
so buoc lap: 10
Muon thoat, bam [.] ?
Tu chuan len = 9
Init: 100000000
Nghiem: 121312313
so buoc lap: 11
Muon thoat, bam [.] ?
Tu chuan len = 10
Init: 1000000000
Nghiem: 1213123132
so buoc lap: 12
Muon thoat, bam [.] ? .
T h e   E n d

```

### Bài 6.3. Tìm đường trong mê cung.

Mê cung là một đồ thị vô hướng bao gồm  $N$  đỉnh, được mã số từ 1 đến  $N$ , với các cạnh, mỗi cạnh nối hai đỉnh với nhau. Giữa hai đỉnh có không quá một cạnh. Cho hai đỉnh  $S$  và  $T$  trong một mê cung. Hãy tìm một đường đi bao gồm các cạnh nối nhau liên tiếp bắt đầu từ đỉnh  $S$ , kết thúc tại đỉnh  $T$  sao cho không qua đỉnh nào quá một lần.

Dữ liệu vào: Tập văn bản tên MECUNG.INP với cấu trúc như sau:

- Dòng đầu tiên chứa ba số tự nhiên  $N$ ,  $S$  và  $T$  ghi cách nhau bởi dấu cách, trong đó  $N$  là số lượng đỉnh của mê cung,  $S$  là đỉnh xuất phát,  $T$  là đỉnh kết thúc.
- Dòng thứ  $i$ , trong số  $N-1$  dòng tiếp theo  $i = 1..(N-1)$ , mỗi dòng ghi  $N-i$  số 1 hoặc 0 cách nhau: số thứ  $j$  trên dòng  $i$  là 1 cho biết có cạnh nối đỉnh  $i$  với đỉnh  $j$ , là 0: không có cạnh nối đỉnh  $i$  với đỉnh  $j$ .

Giới hạn của  $N$ : 200.

#### Ví dụ

MECUNG.INP	
9 6 7	
1 0 1 1 1 0 0 0	
1 1 0 0 0 0 0	
0 0 0 1 0 0	
0 1 1 0 0	
0 0 0 0	
0 0 0	
0 0	
1	

cho  
Mê  
mã

= 6

Dòng 1: 1 0 1 1 1 0 0 0 - đỉnh 1 được nối với các đỉnh 2, 4, 5, và 6. Không có cạnh nối đỉnh 1 với các đỉnh 3, 7, 8 và 9.

biết

cung gồm 9 đỉnh số 1..9, cần tìm đường đi từ đỉnh  $S$  đến đỉnh  $T = 7$ .

...

Dòng 8: 1 - đỉnh 8 có nối với đỉnh 9.

Vì đồ thị là vô hướng nên cạnh nối đỉnh  $x$  với đỉnh  $y$  cũng chính là cạnh nối đỉnh  $y$  với đỉnh  $x$ .

Thông tin về đỉnh  $N$  không cần thông báo, vì với mỗi đỉnh  $i$  ta chỉ liệt kê các đỉnh  $j > i$  tạo thành cạnh  $(i, j)$ .

**Kết quả ra ghi trong tệp văn bản MECUNG.OUT :**

- Ghi lần lượt các đỉnh có trên đường đi từ dòng S đến dòng T. Nếu vô nghiệm: ghi số 0.

Với thí dụ đã cho, kết quả có thể là:

MECUNG.OUT	Từ đỉnh 6 có thể đến được đỉnh 7, qua 5 đỉnh theo đường bốn khúc:
6 1 2 3 7	$6 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 7$ .

Với mê cung đã cho, nếu yêu cầu tìm đường đi từ đỉnh 6 đến đỉnh 9, tức là với dữ liệu vào như trên thì sẽ nhận được kết quả 0 với ý nghĩa là không có đường đi từ đỉnh 6 đến đỉnh 9, do mê cung đã cho không liên thông, đỉnh 6 và đỉnh 9 nằm trong hai vùng liên thông khác nhau.

## Thuật toán

Xuất phát từ đỉnh  $v[1] = s$ , mỗi bước lặp  $i$  ta thực hiện các kiểm tra sau. Gọi  $k$  là số đỉnh đã đi qua và được tích lũy trong mảng giải trình đường đi  $v$ , cụ thể là xuất phát từ đỉnh  $v[1] = s$ , sau một số lần duyệt ta quyết định chọn đường đi qua các đỉnh  $v[1], v[2], v[3], \dots, v[k]$ . Có thể gặp các tình huống sau:

a) (Đến đích?) nếu  $v[k] = t$  tức là đã đến được đỉnh  $t$ : thông báo kết quả, dừng thuật toán, ngược lại thực hiện  $b$ .

b) (Thất bại?)  $k = 0$ : nếu đã quay trở lại vị trí xuất phát  $v[i] = s$  mà từ đó không còn đường đi nào khác thì phải lùi một bước nữa, do đó  $k = 0$ . Trường hợp này chứng tỏ bài toán vô nghiệm, tức là, do đồ thị không liên thông nên không có đường đi từ đỉnh  $s$  đến đỉnh  $t$ . Ta thông báo vô nghiệm và dừng thuật toán.

c) (Đi tiếp?) nếu từ đỉnh  $v[k]$  tìm được một cạnh chưa đi qua và dẫn đến một đỉnh  $i$  nào đó thì tiến theo đường đó, nếu không: thực hiện bước  $d$ .

d) (Lùi một bước) Bỏ đỉnh  $v[k]$ , lùi lại đỉnh  $v[k-1]$ .

Thuật toán trên có tên là *sơ chỉ Arian* được phỏng theo một truyền thuyết cổ Hy Lạp sau đây. Anh hùng Te-dây phải tìm diệt con quái vật nhân ngư (đầu người, mình trâu) Minotav ẩn náu trong một phòng của mê cung có nhiều ngõ ngách rắc rối đã từng làm lạc bước nhiều dũng sĩ và những người này đều trở thành nạn nhân của Minotav. Người yêu của chàng Te-dây là công chúa của xứ Mino đã đưa cho chàng một cuộn chỉ và dặn chàng như sau: *Chàng hãy buộc một đầu chỉ vào cửa mê cung (phòng xuất phát s), sau đó, tại mỗi phòng trong mê cung, chàng hãy tìm xem có Minotav ẩn trong đó không. Nếu có, chàng hãy chiến đấu dũng cảm để hạ thủ nó rồi cuộn chỉ quay ra cửa hang, nơi em trông ngóng chàng. Nếu chưa thấy Minotav tại phòng đó, chàng hãy kiểm tra xem chỉ có bị rối hay không. Cuộn chỉ bắt đầu rối khi nào từ phòng chàng đứng có hai sơ chỉ đi ra hai cửa*

khác nhau. Nếu chỉ rối như vậy, chàng hãy cuộn chỉ để lùi lại một phòng và nhớ đánh dấu đường đã đi để khỏi lạc bước vào đó lần thứ hai.

Nếu không gặp chỉ rối thì chàng hãy yên tâm dò tìm một cửa chưa đi để qua phòng khác. Đi đến đâu chàng nhớ nhà chỉ theo đến đó. Nếu không có cửa để đi tiếp hoặc từ phòng chàng đang đứng, mọi cửa ra đều đã được chàng đi qua rồi, thì chàng hãy cuộn chỉ để lùi lại một phòng rồi tiếp tục tìm cửa khác.

Với đỉnh xuất phát  $s = 6$  và đỉnh kết thúc  $t = 7$  quy trình tìm đường được giải trình với các bước nhà chỉ như sau:

```
v: 6
v: 6 -> 1 nhà chỉ
v: 6 -> 1 -> 2 nhà chỉ
v: 6 -> 1 -> 2 -> 3 nhà chỉ
v: 6 -> 1 -> 2 -> 3 -> 7 nhà chỉ
Nghiem: 6 -> 1 -> 2 -> 3 -> 7
```

Với đỉnh xuất phát  $s = 6$  và đỉnh kết thúc  $t = 9$  quy trình tìm đường được giải trình với các bước nhà chỉ và cuộn chỉ (quay lui) như sau:

```
v: 6
v: 6 -> 1 nhà chỉ
v: 6 -> 1 -> 2 nhà chỉ
v: 6 -> 1 -> 2 -> 3 nhà chỉ
v: 6 -> 1 -> 2 -> 3 -> 7 nhà chỉ
v: 6 -> 1 -> 2 -> 3 -> 7 -> 4 nhà chỉ
v: 6 -> 1 -> 2 -> 3 -> 7 <- cuộn chỉ
v: 6 -> 1 -> 2 -> 3 <- cuộn chỉ
v: 6 -> 1 -> 2 <- cuộn chỉ
v: 6 -> 1 <- cuộn chỉ
v: 6 -> 1 -> 5 nhà chỉ
v: 6 -> 1 <- cuộn chỉ
v: 6 Hết cách
Vo nghiem.
```

## Chương trình C++

```
// Me cung
#include <bits/stdc++.h>

using namespace std;

const int MN = 200;
const int MN1 = MN+1;
char a[MN1][MN1]; // ma tran ke
int n; // so dinh
int s, t; // s: dinh xuat phat, t: dinh ket thuc
int v[MN1]; // duong di
bitset<MN1> b; // danh dau dinh da qua

using namespace std;
```

```

void Go() {
    cout << "\n Muon thoat, bam [.] ? ";
    fflush(stdin);
    if (cin.get() == '.')
        exit(0);
}

void Print(char x[], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i)
        cout << " " << (int)x[i];
}

void Print(char x[][MN1], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i)
        Print(x[i], d, c, "\n ");
}

void Show(int k, const char * msg = "") {
    cout << msg;
    for (int i = 1; i <= k; ++i) {
        cout << " " << v[i];
    }
}

void ReadInput() {
    ifstream f("MECUNG.INP");
    f >> n >> s >> t;
    cout << "\n So dinh " << n;
    cout << "\n Dinh xuat phat: " << s;
    cout << "\n Dinh ket thuc: " << t;
    memset(a, 0, sizeof(a));
    int c;
    for (int i = 1; i < n; ++i) {
        for (int j = i + 1; j <= n; ++j) {
            f >> c;
            a[j][i] = a[i][j] = c;
        }
    }
    f.close();
    Print(a, 1, n, "\n Ma tran ke a:\n");
}

// Tim duong di tu u den v
int Find(int u) {
    for (int v = 1; v <= n; ++v) {
        if (a[u][v] && b[v]) {
            return v;
        }
    }
    return 0;
}

```



```

}

WriteResult(int k) {
    ofstream g("MECUNG.OUT");
    for(int i = 1; i <= k; ++i)
        g << " " << v[i];
    g.close();
}

// Tim 1 nghiem
void MeCung() {
    ReadInput();
    b.set();
    int k = 1; // buoc di
    v[k] = s;
    b[s] = 0; // da tham
    int d; // Dinh tu v[k] di den
    while (true) { // back tracking
        Show(k, "\n v: ");
        if (v[k] == t) { // den dich
            Show(k, "\n Nghiem: ");
            WriteResult(k);
            return;
        }
        if (k < 1) {
            cout << "\n Vo nghiem.";
            return;
        }
        d = Find(v[k]); // v[k] -> d
        if (d > 0) { // tien
            v[++k] = d;
            b[d] = 0; // danh dau phong da den
        }
        else { // v[k] = 0 Het duong
            --k;
        }
    } // while
}

main() {
    MeCung();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

So dinh 9
Dinh xuất phát: 6
Dinh kết thúc: 7
Ma tran ke a:

```

```
0 1 0 1 1 1 0 0 0
1 0 1 1 0 0 0 0 0
0 1 0 0 0 0 1 0 0
1 1 0 0 0 1 1 0 0
1 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 0
0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1 0
v: 6
v: 6 -> 1
v: 6 -> 1 -> 2
v: 6 -> 1 -> 2 -> 3
v: 6 -> 1 -> 2 -> 3 -> 7
Nghiem: 6 -> 1 -> 2 -> 3 -> 7
T h e   E n d
```

### Bình luận

Có nhiều thuật toán xác định đường đi trong đồ thị. Thuật toán quay lui vận dụng cho bài này có thể không hiệu quả. Trong chương tiếp theo về quy hoạch động, chúng ta sẽ tìm hiểu thêm thuật toán tìm đường tối ưu trong đồ thị.

## CHƯƠNG 7

### QUY HOẠCH ĐỘNG

---

Các bài toán *quy hoạch động* chiếm một vị trí khá quan trọng trong tổ chức hoạt động và sản xuất. Chính vì lẽ đó mà trong các kì thi học sinh giỏi quốc gia và quốc tế chúng ta thường gặp loại toán này.

Thông thường những bạn nào dùng phương pháp quay lui, vét cạn cho các bài toán quy hoạch động thì chỉ có thể vét được các tập dữ liệu nhỏ, kích thước chừng vài chục byte. Nếu tìm được đúng hệ thức thể hiện bản chất quy hoạch động của bài toán và khéo tổ chức dữ liệu thì ta có thể xử lí được những tập dữ liệu khá lớn.

Có thể tóm lược nguyên lí quy hoạch động do Bellman phát biểu như sau:

#### Quy hoạch động

*Quy hoạch động là lớp các bài toán mà quyết định ở bước thứ  $i$  phụ thuộc vào quyết định ở các bước đã xử lí trước hoặc sau bước đó.*

Để giải các bài toán quy hoạch động, ta có thể theo sơ đồ sau đây:

#### Sơ đồ giải bài toán quy hoạch động

1. *Lập hệ thức*: Lập hệ thức biểu diễn tương quan quyết định của bước đang xử lí với các bước đã xử lí trước hoặc sau bước đó. Khi đã có hệ thức tương quan chúng ta đã có thể xây dựng ngay thuật giải, tuy nhiên các hệ thức này thường là các biểu thức *đệ quy*, do đó dễ gây ra hiện tượng tràn miền nhớ và tốn thời gian khi ta tổ chức chương trình trực tiếp bằng đệ quy.
2. *Tổ chức dữ liệu và chương trình*: Tổ chức dữ liệu tính toán dần theo từng bước. Nên tìm cách *khử đệ quy*. Trong các bài toán quy hoạch động thuộc chương trình phổ thông thường đòi hỏi một vài mảng một hoặc hai chiều.
3. *Làm tốt*: Làm tốt thuật toán bằng cách thu gọn hệ thức quy hoạch động và giảm kích thước miền nhớ.

### Bài 7.1. Chia thưởng

*Cần chia hết  $m$  phần thưởng cho  $n$  học sinh sắp theo thứ tự từ giỏi trở xuống sao cho mỗi bạn không nhận ít phần thưởng hơn bạn xếp sau mình.*

$$1 \leq m, n \leq 70.$$

Hãy tính số cách chia.

Thí dụ, với số phần thưởng  $m = 7$ , và số học sinh  $n = 4$  sẽ có 11 cách chia 7 phần thưởng cho 4 học sinh theo yêu cầu của đầu bài. Đó là:

Phương án	①	②	③	④
1	7	0	0	0
2	6	1	0	0
3	5	2	0	0
4	5	1	1	0
5	4	3	0	0
6	4	2	1	0
7	3	3	1	0
8	3	2	2	0
9	4	1	1	1
10	3	2	1	1
11	2	2	2	1

## Thuật toán

### Lập hệ thức quy hoạch động

Gọi  $\text{Chia}(pt, hs)$  là số cách chia  $pt$  phần thưởng cho  $hs$  học sinh, ta thấy:

- Nếu không có học sinh nào ( $hs = 0$ ) thì không có cách chia nào ( $\text{Chia} = 0$ ).
- Nếu không có phần thưởng nào ( $pt = 0$ ) thì chỉ có một cách chia ( $\text{Chia}(0, hs) = 1$ , mỗi học sinh nhận 0 phần thưởng).
- Nếu số phần thưởng ít hơn số học sinh ( $pt < hs$ ) thì trong mọi phương án chia, từ học sinh thứ  $pt + 1$  trở đi sẽ không được nhận phần thưởng nào:

$$\text{Chia}(pt, hs) = \text{Chia}(pt, pt) \text{ nếu } pt < hs.$$

- Ta xét tất cả các phương án chia trong trường hợp  $pt \geq hs$ . Ta tách các phương án chia thành *hai nhóm không giao nhau* dựa trên số phần thưởng mà học sinh đứng cuối bảng thành tích, học sinh thứ  $hs$ , được nhận:
  - ♦ Nhóm thứ nhất gồm các phương án trong đó học sinh cuối bảng không được nhận thưởng, tức là  $pt$  phần thưởng chỉ chia cho  $hs - 1$  học sinh và do đó, số cách chia, tức là số phần tử của nhóm này sẽ là:  $\text{Chia}(pt, hs - 1)$ .
  - ♦ Nhóm thứ hai gồm các phương án còn lại, trong đó học sinh cuối bảng cũng được nhận thưởng. Khi đó, do học sinh đứng cuối bảng thành tích được nhận thưởng thì mọi học sinh khác cũng sẽ có thưởng. Do ai cũng được thưởng nên ta bớt của mỗi người một phần thưởng (để họ lĩnh sau), số phần thưởng còn lại ( $pt - hs$ ) sẽ được chia cho  $hs$  học sinh. Số cách chia khi đó sẽ là  $\text{Chia}(pt - hs, hs)$ .

Tổng số cách chia cho trường hợp  $pt \geq hs$  sẽ là tổng số phần tử của hai nhóm, ta có:

$$\text{Chia}(pt, hs) = \text{Chia}(pt, hs - 1) + \text{Chia}(pt - hs, hs)$$

Tổng hợp lại ta có:

<i>Điều kiện</i>	$\text{Chia}(pt, hs)$
$hs = 0$ :	0
$pt = 0 \text{ and } hs \neq 0$ :	1
$pt < hs$ :	$\text{Chia}(pt, pt)$
$pt \geq hs$ :	$\text{Chia}(pt, hs - 1) + \text{Chia}(pt - hs, hs)$

*Các tính chất của hàm  $\text{Chia}(pt, hs)$*

### Phương án đệ quy

Ta có phương án đầu tiên của giải thuật Chia như sau:

### Chương trình C++

```
// Chia thuong. Phuong an 1: De quy
#include <bits/stdc++.h>

using namespace std;

// so cach chia het ptphan thuong cho hs hos sinh
// xep tu gioi tro xuong.
// Hs dung tren nhan so phan thuong khong it hon hs dung duoi
int Chia(int pt, int hs) {
    if (hs == 0) return 0; // ko co hs
    if (pt == 0) return 1; // co hs, ko co phan thuong
    return (pt < hs) ? Chia(pt,pt) // co hs, co phan thuong < hs
        : Chia(pt, hs-1) + Chia(pt-hs,hs);
}

main() {
    cout << Chia(7,4); // 11
    cout << "\n T h e   E n d";
    return 0;
}
```

Phương án này chạy chậm vì phát sinh ra quá nhiều lần gọi hàm trùng lặp. Bảng dưới đây liệt kê số lần gọi hàm Chia khi giải bài toán chia thường với bảy phần thưởng và 4 học sinh. Ví dụ, hàm Chia(1,1) sẽ được gọi 9 lần,... Tổng số lần gọi hàm Chia là 79. 79 lần gọi hàm để sinh ra kết quả 11 là quá tốn kém.

		hs				
		0	1	2	3	4
pt	0	0	9	1	1	0
	1	9	9	2	1	0
	2	6	6	1	0	0
	3	5	5	2	1	1
	4	3	3	1	1	0
	5	2	2	1	0	0
	6	1	1	0	0	0
	7	1	1	1	1	1
Số lần gọi hàm Chia cục bộ						
khi tính hàm Chia(7,4)						

Bạn có thể kiểm tra sự tốn kém này bằng cách đặt thêm một biến đếm tổng thể d để đếm số lần gọi hàm Chia như sau:

### Chương trình C++

```
// Chia thương. Phương án 1B: Đệ quy
#include <bits/stdc++.h>

using namespace std;

int d; // đếm số lần gọi hàm Chia

// số cách chia hết pt phần thương cho hs học sinh
// xếp từ giỏi trở xuống.
// Hs đứng trên nhận số phần thương không ít hơn hs đứng dưới
int Chia(int pt, int hs) {
    ++d;
    if (hs == 0) return 0; // không có hs
    if (pt == 0) return 1; // có hs, không có phần thương
    return (pt < hs) ? Chia(pt, pt) // có hs, có phần thương < hs
        : Chia(pt, hs-1) + Chia(pt-hs, hs);
}

main() {
    d = 0;
    cout << Chia(7, 4);
    cout << "\n Số lần gọi hàm Chia: " << d; // 79
    cout << "\n T h e   E n d";
    return 0;
}
```

### Cải tiến lần 1

Phương án 1 khá dễ triển khai nhưng chương trình sẽ chạy rất lâu, bạn hãy thử gọi Chia(66,32) để trải nghiệm được điều trên. Diễn tả đệ quy thường trong sáng, nhân bản, nhưng khi thực hiện sẽ sinh ra hiện tượng gọi lặp lại những hàm đệ quy. Cải tiến đầu tiên là *tránh những lần gọi lặp* như vậy. Muốn thế chúng ta tính sẵn các giá trị của hàm theo các trị của đầu vào khác nhau và điền vào một mảng hai chiều cc theo công thức:

$$cc[pt][hs] = \text{Chia}(pt, hs)$$

Theo các tính chất của hàm hai ngôi Chia, ta có

$$\begin{aligned} cc[pt][hs] &= \text{Chia}(pt, hs) = \\ &= 0, \text{ nếu } hs = 0 \text{ and } pt > 0 \\ &= 1, \text{ nếu } pt = 0 \text{ and } hs > 0 \\ &= cc[pt][pt], \text{ nếu } 0 < pt < hs \\ &= cc[pt][hs-1] + cc[pt-hs][hs], \text{ nếu } pt \geq hs > 0 \end{aligned}$$

Từ đó ta suy ra quy trình điền trị vào bảng cc như sau:

Dòng 0,  $cc[0][*] = (1, 1, \dots, 1)$ : Tại dòng 0 ta điền toàn 1 vì không có phần thưởng thì có 1 phương án chia là mỗi hs nhận 0 phần thưởng.

Cột 0,  $cc[*][0] = (1, 0, \dots, 0)$  Tại cột 0 ta điền toàn 0, vì không có hs mà có pt thì không thể chia hết pt được. Riêng  $cc[0][0] = 1$  theo quy ước.

Cột 1,  $cc[*][1] = (1, 1, \dots, 1)$  Tại cột 1 ta điền toàn 1, vì chỉ có một hs nên có bao nhiêu phần thưởng đều dành cho em đó. Như vậy là chỉ có 1 cách chia.

	0	1	...	j	...	hs
0	1	1				1
1	0	1				1
...				*		
i			*	?		
...						
pt	0					

Các phần tử  $cc[i][j]$  còn lại sẽ được điền theo cột, từ cột  $j = 2$  đến  $j = hs$ .

```
for j = 2..hs do
  for i = 1..j-1 do
    cc[i][j] = cc[i][j-1] // vì i < j
  end for i
  for i = j..pt do
    cc[i][j] = cc[i][j-1] + cc[i-j][j] // vì i ≥ j
  end for i
Kết quả có trong ô cc[pt][hs]
```

## Chương trình C++

```
// Chia thuong.
// Phuong an 2: Dung mang 2 chieu
#include <bits/stdc++.h>

using namespace std;

const int MN = 71;
int cc[MN][MN]; // cc[pt][hs] = Chia(pt,hs)

// so cach chia het m phan thuong cho n hos sinh
// xep tu gioi tro xuong.
// Hs dung tren nhan so phan thuong khong it hon hs dung duoi
int Chia(int pt, int hs) {
  if (hs == 0)
    return (pt == 0) ? 1 : 0; // ko co hs
  if (pt == 0) return 1; // ko co pt
  if (pt < hs) hs = pt;
  // cot 1 toan 1
  for(int i = 0; i <= pt; ++i)
    cc[i][1] = 1;
  // dien theo cot j = 2..hs
  for(int j = 2; j <= hs; ++j) {
    for(int i = 0; i < j; ++i)
      // bao luu cac gia tri dau 0..j-1
      cc[i][j] = cc[i][j-1];
    for(int i = j; i <= pt; ++i)
      // tinh cac gia tri tu j..pt
      cc[i][j] = cc[i][j-1] + cc[i-j][j];
  }
  return cc[pt][hs];
}
```

```

}

main() {
    cout << "\n " << Chia(7, 4); // 11
    cout << "\n " << Chia(70, 14); // 1614987
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

11
1614987
T h e   E n d

```

### *Cải tiến lần 2*

Dùng mảng hai chiều chúng ta chỉ có thể tính toán được với dữ liệu nhỏ. Bước cải tiến sau đây khá quan trọng: chúng ta dùng mảng một chiều. Quan sát kỹ quy trình gán trị cho mảng hai chiều theo từng cột chúng ta dễ phát hiện ra rằng cột thứ  $j$  có thể được tính toán từ cột thứ  $j - 1$ . Hơn nữa, khi tính toán tại bước  $j$  ta có, nếu  $i < j$  thì

$c[i]$  tại bước  $j = cc[i][j] = cc[i][j-1] =$  giá trị  $c[i]$  tại bước  $j-1$  nghĩa là giá trị  $c[i]$  tại bước  $j$  được bảo lưu khi  $i < j$ .

Ta có phương án ba, dùng một mảng một chiều  $c$  như sau:

## Chương trình C++

```

// Chia thương.
// Phương án 3: Dùng mảng 1 chiều
#include <bits/stdc++.h>

using namespace std;

const int MN = 71;
int c[MN];

// so cach chia het m phan thương cho n hos sinh
// xep tu gioi tro xuong.
// Hs dung tren nhan so phan thương không ít hơn hs dung duoi
int Chia(int pt, int hs) {
    if (hs == 0)
        return (pt == 0) ? 1 : 0; // ko co hs
    if (pt == 0) return 1; // ko co pt
    if (pt < hs) hs = pt;
    // cot 1 toan 1
    for(int i = 0; i <= pt; ++i)
        c[i] = 1;
    // dien theo cot j = 2..hs
    for(int j = 2; j <= hs; ++j) {
        // bao luu cac gia tri dau 0:j-1

```



```

        for(int i = j; i <= pt; ++i)
            // tính các giá trị tu hs..m
            c[i] += c[i-j];
    }
    return c[pt];
}

main() {
    cout << "\n " << Chia(7, 4); // 11
    cout << "\n " << Chia(70, 14); // 1614987
    cout << "\n T h e   E n d";
    return 0;
}

```

## Bài 7. 2. Palindrome

*Olympic Quốc tế, 2000, Bắc Kinh.*

Dãy ký tự  $s$  được gọi là đối xứng (palindrome) nếu các phần tử cách đều đầu và cuối giống nhau. Cho dãy  $s$  tạo bởi  $n$  ký tự gồm các chữ cái hoa và thường phân biệt và các chữ số. Hãy cho biết cần xóa đi từ  $s$  ít nhất là bao nhiêu ký tự để thu được một dãy đối xứng. Giả thiết rằng sau khi xóa bớt một số ký tự từ  $s$  thì các ký tự còn lại sẽ tự động xích lại sát nhau.

Dữ liệu vào ghi trong tệp văn bản PALIN.INP với cấu trúc như sau:

Dòng đầu tiên là giá trị  $n$ ,  $1 \leq n \leq 1000$ .

Dòng thứ hai là  $n$  ký tự của dãy viết liền nhau.

Dữ liệu ra ghi trong tệp văn bản PALIN.OUT: số lượng ký tự cần xóa.

PALIN.INP	PALIN.OUT
9 bae <b>a</b> dbadb	4

Thí dụ, với dãy  $s$  gồm 9 ký tự,  $s = \text{'bae**a**dbadb'}$  thì cần xóa ít nhất 4 ký tự, chẳng hạn, các ký tự thứ 4, 6, 7 và 8 sẽ thu được dãy đối xứng chiều dài 5 là baeab:

bae**a**db**a**db → baeab

Dĩ nhiên là có nhiều cách xóa. Tuy nhiên đáp số là số ít nhất các ký tự cần loại bỏ khỏi  $s$  thì là duy nhất và bằng 4.

### Thuật toán

Bài toán này đã được nhiều bạn đọc công bố lời giải với một mảng hai chiều kích thước  $n^2$  hoặc vài ba mảng một chiều kích thước  $n$ , trong đó  $n$  là chiều dài của dữ liệu vào.

Với một nhận xét nhỏ ta có thể phát hiện ra rằng chỉ cần dùng một mảng một chiều kích thước  $n$  và một vài biến đơn là đủ.

**Lập hệ thức quy hoạch động**

Gọi  $p(i, j)$  là chiều dài của dãy con dài nhất thu được khi giải bài toán với dữ liệu vào là đoạn  $s[i..j]$ . Khi đó  $p(0, n-1)$  là chiều dài của dãy con đối xứng dài nhất trong dãy  $n$  ký tự  $s[0..n-1]$  và do đó số ký tự cần loại bỏ khỏi dãy  $s[0..n-1]$  sẽ là

$$n - p(0, n-1)$$

Đó chính là đáp số của bài toán.

Ví dụ, với dãy  $s = \text{baeadbadb}$ ,  $n = 9$  thì  $p(0, 8) = 5$  và số ký tự cần xóa sẽ là  $n - p(0, 8) = 9 - 5 = 4$ .

Ta liệt kê một số tính chất quan trọng của hàm hai biến  $p(i, j)$ . Ta có:

- Nếu  $i > j$ , tức là chỉ số đầu trái lớn hơn chỉ số đầu phải, ta quy ước  $p(i, j) = 0$ .
- Nếu  $i = j$  thì  $p(i, i) = 1$  vì dãy khảo sát chỉ chứa đúng 1 ký tự nên nó là đối xứng.
- Nếu  $i < j$  và  $s[i] = s[j]$  thì  $p(i, j) = p(i+1, j-1) + 2$ . Vì hai ký tự đầu và cuối dãy  $s[i..j]$  giống nhau nên chỉ cần xác định chiều dài của dãy con đối xứng dài nhất trong đoạn giữa là  $s[i+1..j-1]$  rồi cộng thêm 2 đơn vị ứng với hai ký tự đầu và cuối dãy là được.
- Nếu  $i < j$  và  $s[i] \neq s[j]$ , tức là hai ký tự đầu và cuối của dãy con  $s[i..j]$  là khác nhau thì ta khảo sát hai dãy con là  $s[i..j-1]$  và  $s[i+1..j]$  để lấy chiều dài của dãy con đối xứng dài nhất trong hai dãy này làm kết quả:

$$p(i, j) = \max(p(i, j-1), p(i+1, j))$$

Vấn đề đặt ra là cần tính  $p(0, n-1)$ . Mà muốn tính được  $p(0, n-1)$  ta phải tính được các  $p(i, j)$  với mọi  $i, j = 0..n-1$ .

#### **Phương án đệ quy**

### **Chương trình C++**

```
// Palindrome. Phuongan 1. De quy
#include <bits/stdc++.h>

using namespace std;

string s;
int n;

// so ki tu nhieu nhat con lai
int P(int i, int j) {
    if (i > j) return 0;
    if (i == j) return 1;
    return (s[i] == s[j]) ? P(i+1, j-1) + 2
        : max(P(i, j-1), P(i+1, j));
}

// so ki tu int nhat can xoa
int Pal(string inps) {
    s = inps;
    n = s.length();
    return n - P(0, n-1);
}
```

```

main() {
    cout << Pal("abcvcbababcvcb") << endl; // xoa 0
    cout << Pal("ab1cvcbaa2bcvc3ba") << endl; // xoa 3
    cout << Pal("ab1cvcba2abcv3ba") << endl; // xoa 2
    cout << "\n T h e   E n d";
    return 0;
}

```

### Cải tiến 1: dùng mảng 2 chiều

		b	a	e	a	d	b	a	d	b
		0	1	2	3	4	5	6	7	8
b	0	1	1	1	3	3	5	5	5	5
a	1	0	1	1	3	3	3	3	3	3
e	2	0	0	1	1	1	1	3	3	3
a	3	0	0	0	1	1	1	3	3	3
d	4	0	0	0	0	1	1	1	3	3
b	5	0	0	0	0	0	1	1	1	3
a	6	0	0	0	0	0	0	1	1	1
d	7	0	0	0	0	0	0	0	1	1
b	8	0	0	0	0	0	0	0	0	1

Gia trị của hàm  $p(i,j)$  đối với dãy baeadbab  
 $i,j=0..8$

Gọi đệ quy sẽ phát sinh các lời gọi hàm trùng lặp như đã phân tích trong bài toán 7.1. Ta khắc phục điều này bằng cách sử dụng một mảng hai chiều  $v$  để tính trước các giá trị của hàm  $p(i, j)$ , mỗi giá trị được tính tối đa một lần. Nếu dùng một mảng hai chiều, thí dụ mảng  $v[0..n-1][0..n-1]$  thì giá trị của  $v[i][j]$  sẽ được tính bằng  $p(i,j)$ :

$$v[i][j] = p(i,j)$$

Ta có, theo hệ thức tính  $p(i,j)$ ,

$v[i][j] = 0$ , nếu  $i > j$ : nửa tam giác dưới đường chéo chính toàn 0

$v[i][i] = 1$ , nếu  $i = j$ : đường chéo chính toàn 1

$v[i][j] = v[i+1][j-1] + 2$ , nếu  $s[i] = s[j]$

$v[i][j] = \max(v[i][j-1], v[i+1][j]) - 2$ , nếu  $s[i] \neq s[j]$

1						
0	1			*	?	
0	0	1		+	*	
0	0	0	1			
0	0	0	0	1		
0	0	0	0	0	1	

? ô  $v[i][j]$  cần tính  
 + ô  $v[i+1][j-1]$   
 \* ô  $v[i][j-1]$   
 \* ô  $v[i+1][j]$

Điền trị cho ma trận  $v$

0	0	0	0	0	0	1
---	---	---	---	---	---	---

Như vậy,  $v[i][j]$  sẽ được điền khi ta biết trị của ba ô bao quanh nó là  
 $+ v[i+1][j-1]$   
 $* v[i][j-1]$ , và  
 $* v[i+1][j]$

Từ đây ta suy ra quy trình điền mảng  $v$  như sau:

Điền 1 tại đường chéo chính,

Điền 0 vào tam giác dưới đường chéo chính,

Điền các dòng  $i$  từ dưới lên trên:  $i = n-2..0$ .

Trên mỗi dòng, ta điền trị từ trái qua phải:  $j = i+1..n-1$ .

## Chương trình C++

```
// Palindrome. Phương án 2: Dùng mảng 2 chiều

#include <bits/stdc++.h>

using namespace std;

// số kí tự int nhất cần xóa
int Pal(string s) {
    int n = s.length();
    int v[n][n];
    memset(v, 0, sizeof(v));
    for(int i = 0; i < n; ++i)
        v[i][i] = 1;
    for(int i = n-2; i >= 0; --i) {
        for(int j = i+1; j < n; ++j) {
            v[i][j] = (s[i]==s[j]) ? v[i+1][j-1] + 2
                                   : max(v[i][j-1], v[i+1][j]);
        }
    }
    return n - v[0][n-1];
}

main() {
    cout << Pal("abcvcbababcvcb") << endl; // xóa 0
    cout << Pal("ab1cvcbab2bcvc3ba") << endl; // xóa 3
    cout << Pal("ab1cvcbab2abcvcb3ba") << endl; // xóa 2
    cout << "\n T h e   E n d";
    return 0;
}
```

### Cải tiến 2: dùng hai mảng 1 chiều

Ta sẽ không theo đuổi phương án dùng mảng hai chiều mà hãy căn cứ vào quy luật điền mảng hai chiều để vận dụng cho hai mảng một chiều là  $a$  và  $b$ , trong đó  $a$  là mảng dưới ứng với dòng  $v[i+1][*]$ ,  $b$  là mảng trên ứng với dòng  $v[i][*]$ ,  $i = n-2..0$ . Cụ thể là

```

v[i][j] = b[j] // dòng i
v[i][j-1] = b[j-1] // dòng i
v[i+1][j]) = b[j] // dòng i+1
v[i+1][j-1] = b[j-1] // dòng i+1

```

Sau mỗi dòng ta hoán vị a và b như hai con trỏ mảng.

## Chương trình C++

```

// Palindrome. Phương án 3: Dùng 2 mảng 1 chiều

#include <bits/stdc++.h>

using namespace std;

// số kí tự int nhất cần xóa
int Pal(string s) {
    int n = s.length();
    int *a = new int[n]; // dòng dưới
    int *b = new int[n]; // dòng trên
    int *c;
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    a[n-1] = 1;
    for(int i = n-2; i >= 0; --i) {
        // điền b qua a
        b[i] = 1;
        for(int j = i+1; j < n; ++j) {
            b[j] = (s[i]==s[j]) ? a[j-1] + 2
                               : max(b[j-1], a[j]);
        }
        c = a; a = b; b = c;
    }
    return n - a[n-1];
}

main() {
    cout << Pal("abcvcbabaabcvcb") << endl; // xóa 0
    cout << Pal("ab1cvcbaa2bcvc3ba") << endl; // xóa 3
    cout << Pal("ab1cvcb2abcv3ba") << endl; // xóa 2
    cout << "\n T h e   E n d";
    return 0;
}

```

## Bài 7.3. Cắm hoa

*Olympic Quốc tế năm 1999.*

*Cần cắm hết k bó hoa khác nhau vào n lọ xếp thẳng hàng sao cho bó hoa có số hiệu nhỏ được đặt trước bó hoa có số hiệu lớn. Với mỗi bó hoa i ta biết giá trị thẩm mỹ khi cắm bó hoa đó vào lọ j là  $v[i][j]$ .*

**Yêu cầu:** Xác định một phương án cắm hoa sao cho tổng giá trị thẩm mỹ là lớn nhất.

Dữ liệu vào ghi trong tệp văn bản HOA.INP:

- Dòng đầu tiên là hai số  $k$  và  $n$ .
- Từ dòng thứ hai trở đi là các giá trị  $v[i][j]$  trong khoảng  $0..10$ , với  $i = 1..k$  và  $j = 1..n$ ;  $1 \leq k \leq n \leq 100$ .

Dữ liệu ra ghi trong tệp văn bản HOA.OUT: dòng đầu tiên là tổng giá trị thẩm mỹ của phương án cắm hoa tối ưu. Từ dòng thứ hai là dãy  $k$  số hiệu lọ được chọn cho mỗi bó hoa.

Các số liệu vào và ra đều là số tự nhiên và được ghi cách nhau bởi dấu cách trên mỗi dòng.

## Ví dụ

HOA.INP

```
4 6
1 1 6 4 3 10
9 1 4 7 2 7
7 2 6 10 2 3
6 10 7 1 3 9
```

HOA.OUT

```
24
1 3 4 6
```

Kết quả cho biết tổng giá trị thẩm mỹ sẽ đạt là 24 (điểm) nếu cắm hoa như sau:

- Bó hoa 1 cắm vào lọ 1;
- Bó hoa 2 cắm vào lọ 3;
- Bó hoa 3 cắm vào lọ 4;
- Bó hoa 4 cắm vào lọ 6.

Dữ liệu vào trong file HOA.INP cho biết:

Cần cắm hết 4 bó hoa vào 4 trong 6 lọ.

Nếu cắm bó hoa 1

vào lọ 1 sẽ đạt độ thẩm mỹ 1,  
vào lọ 2 sẽ đạt độ thẩm mỹ 1,  
vào lọ 3 sẽ đạt độ thẩm mỹ 6,  
vào lọ 4 sẽ đạt độ thẩm mỹ 4,  
vào lọ 5 sẽ đạt độ thẩm mỹ 5,  
vào lọ 6 sẽ đạt độ thẩm mỹ 10,

Nếu cắm bó hoa 2

vào lọ 1 sẽ đạt độ thẩm mỹ 9,  
vào lọ 2 sẽ đạt độ thẩm mỹ 1,

...

...

## Thuật toán

Trước hết ta đọc dữ liệu từ tệp HOA.INP vào các biến  $k$ ,  $n$  và  $v[][]$ .

### Lập hệ thức quy hoạch động

Gọi  $T(i, j)$  là tổng giá trị thẩm mỹ khi giải bài toán với  $i$  bó hoa mã số  $1..i$  và  $j$  lọ mã số  $1..j$ , tức là độ thẩm mỹ thu được khi cắm hết  $i$  bó hoa đầu tiên vào  $j$  trong số  $j$  lọ đầu tiên, ta thấy:

- ✎ Nếu số bó hoa nhiều hơn số lọ,  $i > j$  thì không có cách cắm nào vì đầu bài yêu cầu phải cắm hết các bó hoa, mỗi bó vào đúng 1 lọ.

$$T(i, j) = 0, \text{ nếu } i > j$$

- ✎ Nếu số bó hoa bằng số lọ ( $i = j$ ) thì chỉ có một cách cắm là bó nào vào lọ đó.

✎ Ta xét trường hợp số bó hoa ít hơn hẳn số lọ ( $i < j$ ). Có hai tình huống: lọ cuối cùng, tức lọ thứ  $j$  được chọn cho phương án tối ưu và lọ thứ  $j$  không được chọn.

- ♦ Nếu lọ cuối cùng, lọ thứ  $j$  được chọn để cắm bó hoa (cuối cùng)  $i$  thì  $i-1$  bó hoa đầu tiên sẽ được phân phối vào  $j-1$  lọ đầu tiên. Tổng giá trị thẩm mỹ khi đó sẽ là  $T(i-1, j-1) + v[i][j]$ .
- ♦ Nếu lọ thứ  $j$  không được chọn cho phương án tối ưu thì  $i$  bó hoa phải được cắm vào  $j-1$  lọ đầu tiên và do đó tổng giá trị thẩm mỹ sẽ là  $T(i, j-1)$ .

Tổng hợp lại ta có giá trị tối ưu khi cắm  $i$  bó hoa vào  $j$  lọ là:

$$T(i, j) = \max \{T(i-1, j-1) + v[i][j], T(i, j-1)\}$$

Phương án dưới đây chỉ tìm giá trị thẩm mỹ tối ưu chứ chưa xác định được lọ cắm mỗi bó hoa.

## Chương trình C++

```

/*****
    Cam Hoa. Phuong an 1: De quy
    *****/
#include <bits/stdc++.h>

using namespace std;

const int MN = 101;
int k; // so bo hoa
int n; // so lo hoa
int v[MN][MN];

void ReadInput() {
    ifstream f("HOA.INP");
    f >> k >> n;
    cout << "\n " << k << " bo hoa, " << n << " lo hoa";
    cout << "\n Do tham my:";
    for (int hoa = 1; hoa <= k; ++hoa) {
        cout << endl;
        for (int lo = 1; lo <= n; ++lo) {
            f >> v[hoa][lo];
            cout << " " << v[hoa][lo];
        }
    }
    f.close();
}

/*-----
    neu so bo hoa i = so lo
    thi hoa nao cam vao lo do
    -----*/
int SumDiagonal(int sh) { // sh: so bo hoa
    int t = 0;
    for (int i = 1; i <= sh; ++i)
        t += v[i][i];
}

```

```

    return t;
}

// Recursive
int T(int i, int j) { // cam i bo hoa vao j lo
    if (i == 0 || j == 0 || i > j)
        return 0;
    if (i == j) return SumDiagonal(i);
    // i < j
    return max(T(i-1,j-1)+v[i][j],T(i,j-1));
}

void CamHoa() {
    ReadInput();
    cout << "\n Ket qua: " << T(k, n);
}

main() {
    CamHoa();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

4 bo hoa, 6 lo hoa
Do tham my:
1 1 6 4 3 10
9 1 4 7 2 7
7 2 6 10 2 3
6 10 7 1 3 9
Ket qua: 24
T h e   E n d

```

## Cải tiến1. Dùng mảng 2 chiều

Để tránh đệ quy, ta dùng một mảng hai chiều tt để lưu các giá trị của hàm  $T(i,j)$ :

$tt[i][j] = T(i,j)$  // Cam hết i bó hoa vào j lọ

Nếu số bó hoa  $i = 0$  ta có dòng 0 chứa toàn 0,

Nếu số lọ hoa  $j = 0$  ta có cột 0 chứa toàn 0,

Nếu số bó hoa = số lọ hoa,  $i = j$ , ta gán trị trên đường chéo chính

$tt[j][j] = \text{sum} \{v[j][j], j = 1..k \text{ (số bó hoa)}\}$

Nếu số bó hoa > số lọ hoa,  $i > j$ , ta gán trị cho tam giác dưới toàn 0,

Ta xét trường hợp số bó hoa < số lọ. Ta có

$tt[i][j] = T(i,j) = \max\{tt[i-1][j-1]+v[i][j], tt[i][j-1]\}$

Tóm lại, muốn tính  $tt[i][j]$  cho trường hợp  $i < j$  ta phải biết trị của các ô  $tt[i-1][j-1]$  và  $tt[i][j-1]$ . Như vậy, các phần tử phía trên đường chéo chính sẽ được điền trị theo cột, từ phải qua trái. Trên mỗi cột ta điền các ô từ dưới lên trên.



		j-1	j	j
tt				
i-1		*		
i		*	?	

## Cải tiến 2. Dùng mảng 1 chiều

Sau khi biết trật tự điền trị cho mảng hai chiều ta suy ra ngay là có thể chỉ dùng một mảng một chiều  $t$  và dịch chuyển dần theo từng cột. Ngoài ra, ta còn cần đặt trong mỗi ô của bảng một mảng dữ liệu gồm  $n$  phần tử để đánh dấu lọ hoa nào được chọn cho mỗi tình huống. Gọi mảng dữ liệu đó là  $b$ , ta dễ thấy là nên điền bảng lần lượt theo từng cột, tại mỗi cột ta điền bảng từ dưới lên theo hai pha như sau:

Pha 1. Cập nhật mảng  $t$  và  $b$  với số lọ hoa  $j = 1..k$  ( $k$  là số bó hoa)

Tại pha này với mỗi số lọ hoa  $j$  ta cập nhật  $t$  và  $b$  theo hai bước:

Bước 1. tính trị  $t[j]$  ứng với hàm  $T(j,j)$  cắm bó hoa nào vào lọ ấy.

Bước 2. tính các trị  $t[i]$  và  $b[i]$  ứng với hàm

$$T(i,j) = \max\{T(i-1,j-1)+v[i][j], T(i,j-1)\}, i = j-1 \leftarrow 1$$

Pha 2. Cập nhật mảng  $t$  và  $b$  với số lọ hoa  $j = k+1..n$  ( $n$  là số lọ hoa)

Tại pha này ta chỉ cần thực hiện bước 2:

Bước 2. tính các trị  $t[i]$  và  $b[i]$  ứng với hàm

$$T(i,j) = \max\{T(i-1,j-1)+v[i][j], T(i,j-1)\}, i = k \leftarrow 1$$

- Nếu  $t[i-1] + v[i][j] > t[i]$  thì ta phải thực hiện hai thao tác:

o Đặt lại trị  $t[i] = t[i-1] + v[i][j]$

o Ghi nhận việc chọn lọ hoa  $j$  trong phương án mới, cụ thể lấy phương án cắm hoa  $(i-1, j-1)$  rồi bổ sung thêm thông tin chọn lọ hoa  $j$  như sau: đặt  $b[i] = b[i-1]$  và đánh dấu phần tử  $j$  trong mảng  $b[i][j]$ :  $b[i][j] = 1$ . Chương trình C++

## Chương trình C++

```
// Cam hoa. Phuong an 2. Dung mang 1 chieu
#include <bits/stdc++.h>
using namespace std;

const int MN = 101;
typedef bitset<MN> BS; // danh dau lo
int v[MN][MN];
int k; // so bo hoa
int n; // so lo hoa
int t[MN]; // tong Do tham my
BS b[MN]; // b[i][j] = 1 cho biet :hoa i cam lo h[j]

void ReadInput() {
    ifstream f("HOA.INP");
    f >> k >> n;
    cout << "\n " << k << " bo hoa, " << n << " lo hoa";
```

```

    cout << "\n Do tham my:";
    for (int i = 1; i <= k; ++i) {
        cout << endl;
        for (int j = 1; j <= n; ++j) {
            f >> v[i][j];
            cout << " " << v[i][j];
        }
        f.close();
    }

void WriteResult() {
    ofstream g("HOA.OUT");
    g << t[k] << endl;
    int bh = 0;
    for (int j = 1; j <= n; ++j)
        if (b[k][j]) {
            ++bh;
            cout << "\n Cam bo hoa " << bh << " vao lo " << j;
            g << j << " ";
        }
    g.close();
}

// cam het k bo hoa vao k trong so n lo
void CamHoa() {
    ReadInput();
    if (k == 0 || n == 0 || k > n) {
        // cout << "\n Ket qua: 0";
        return;
    }
    // k <= n
    // dien cot t toan 9
    memset(t, 0, sizeof(t));
    // khoi tri mang bit b toan 0
    for(int i = 0; i < MN; ++i)
        b[i].reset();
    int tt;
    // t[i] tai buoc j la tri toi uu
    // khi giai bai cam het i bo hoa vao j lo
    // duyet tu so lo j = 2..n
    // k <= n
    // Pha 1. so lo = 1..k
    for(int j = 1; j <= k; ++j) { // j lo
        // xet truong hop j hoa, j lo:
        // hoa nao lo ay
        t[j] = t[j-1] + v[j][j];

        b[j][j] = 1; // hoa j cam lo j
        // duyet cot voi so bo hoa i tu duoi len
        // so hoa i < so lo j
        for(int i = j-1; i > 0; --i) { // so bo hoa i
            tt = t[i-1] + v[i][j]; // hoa i cam lo j

```

```

        if (tt > t[i]) {
            t[i] = tt;
            b[i] = b[i-1];
            b[i][j] = 1;
        }
    }
}
// Pha 2. So lo = k+1..n
for(int j = k+1; j <= n; ++j) { // j lo
    // duyet cot voi so bo hoa i tu tren xuong
    // so hoa i < so lo j
    for(int i = k; i > 0; --i) { // so bo hoa i
        tt = t[i-1] + v[i][j];
        if (tt > t[i]) {
            t[i] = tt;
            b[i] = b[i-1];
            b[i][j] = 1;
        }
    }
}
cout << "\n Ket qua: " << t[k] << endl;
WriteResult();
}

main() {
    CamHoa();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

4 bo hoa, 6 lo hoa
Do tham my:
1 1 6 4 3 10
9 1 4 7 2 7
7 2 6 10 2 3
6 10 7 1 3 9
Ket qua: 24

Cam bo hoa 1 vao lo 1
Cam bo hoa 2 vao lo 3
Cam bo hoa 3 vao lo 4
Cam bo hoa 4 vao lo 6
T h e   E n d

```

Các bài toán sau đây là một cách phát biểu khác của bài toán cắm hoa:

### **Bố trí phòng học**

*Câu lạc bộ - Học sinh giỏi Tin học, Hà Nội, năm 2000*

*Cần bố trí k nhóm học sinh vào k trong số n phòng học chuyên để sao cho nhóm có số hiệu nhỏ được xếp vào phòng có số hiệu nhỏ hơn phòng chứa nhóm có số hiệu*

lớn. Với mỗi phòng có nhận học sinh, các ghế thừa phải được chuyển ra hết, nếu thiếu ghế thì phải lấy từ kho vào cho đủ mỗi học sinh một ghế. Biết số học sinh trong mỗi nhóm và số ghế trong mỗi phòng. Hãy chọn phương án bố trí sao cho tổng số lần chuyển ghế ra và chuyển ghế vào là ít nhất.

### Bóng

Tám được Bụt trao cho  $k$  chú bóng. Nhiệm vụ của Tám là phải thả  $k$  bóng này vào  $n$  giếng để chăm nuôi. Các giếng được xếp thẳng hàng theo thứ tự  $1..n$ . Bóng có số hiệu nhỏ phải được thả vào giếng số hiệu nhỏ. Nếu nuôi bóng  $b$  trong giếng  $g$  thì sau này bóng sẽ sinh ra số quà tặng là  $v(b, g)$ .

Bạn hãy giúp Tám thả cá vào các giếng để sau này nhận được số quà nhiều nhất.



Dữ liệu vào ghi trong tệp văn bản BONG.INP:

Dòng đầu tiên là hai trị  $k$  và  $n$ .

Từ dòng thứ hai trở đi là các giá trị  $v(b, g)$  trong khoảng  $0..10$ , với  $b = 1..k$

và  $g = 1..n$ ;  $1 \leq b \leq g \leq 100$ .

Kết quả hiển thị trên màn hình một phương án thả bóng và tổng giá trị quà của phương án tối ưu.

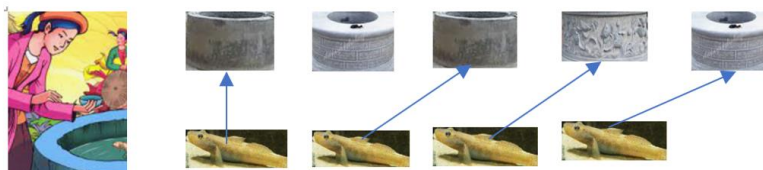
### Ví dụ

BONG.INP

4	5			
<u>1</u>	1	6	4	10
9	1	<u>4</u>	7	7
7	2	6	<u>10</u>	3
6	10	7	1	<u>9</u>

Kết quả cho biết tổng giá trị quà sẽ đạt là 24 (điểm)  
nếu thả cá như sau:

- ☞ thả bóng 1 vào giếng số 1;
- ☞ thả bóng 2 vào giếng số 3;
- ☞ thả bóng 3 vào giếng số 4;
- ☞ thả bóng 4 vào giếng số 5.



## Bài 7.4. Tìm các đường ngắn nhất

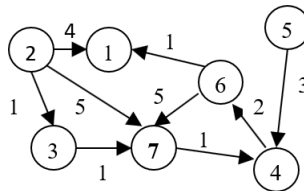
Cho một đồ thị có hướng gồm  $n$  đỉnh mã số từ  $1..n$  với các cung  $(u, v)$  có hướng đi từ đỉnh  $u$  đến đỉnh  $v$  và có chiều dài thể hiện đường đi nối từ đỉnh  $u$  đến đỉnh  $v$ . Viết chương trình tìm mọi đường đi ngắn nhất từ một đỉnh  $s$  cho trước tới các đỉnh còn lại của đồ thị.

Dữ liệu vào được ghi trong một tệp văn bản tên DIJ.INP có cấu trúc như sau:

- Dòng đầu ghi hai số tự nhiên  $n$  và  $s$  cách nhau bởi dấu cách, trong đó  $n$  là số lượng đỉnh của đồ thị,  $s$  là số hiệu của đỉnh xuất phát  $2 \leq n \leq 100, 1 \leq s \leq n$ .
- Từ dòng thứ hai ghi lần lượt độ dài đường đi từ đỉnh  $i$  đến các đỉnh  $1, 2, \dots, n$ ;  $i = 1..n$ . Giá trị 0 cho biết không có cung nối hai đỉnh tương ứng. Với mọi đỉnh  $i = 1..n$ , cung  $(i, i)$  được hiểu là không tồn tại và đường đi từ đỉnh  $i$  tới chính nó với chiều dài là 0. Các số cùng dòng cách nhau qua dấu cách. Dạng dữ liệu cho như vậy được gọi là ma trận kề của đồ thị.

Thí dụ sau đây cho biết đồ thị có bảy đỉnh, cần tìm các đường đi ngắn nhất từ đỉnh  $s = 2$  tới các đỉnh còn lại của đồ thị. Cung  $(2, 1)$  có chiều dài 4,...

DIJ.INP													
7	2												
0	0	0	0	0	0	0	0						
4	0	1	0	0	0	5							
0	0	0	0	0	0	1							
0	0	0	0	0	0	2	0						
0	0	0	3	0	0	0							
1	0	0	0	0	0	5							
0	0	0	1	0	0	0							



Dữ liệu ra được ghi trong tệp văn bản DIJ.OUT gồm  $n$  dòng. Thông tin về mỗi đường đi ngắn nhất từ đỉnh  $s$  đến các đỉnh còn lại được ghi trên 1 dòng. Số đầu tiên của dòng là chiều dài đường đi. Nếu không tồn tại đường đi thì ghi giá trị 0. Tiếp đến, trong trường hợp có đường đi từ đỉnh  $s$  đến đỉnh  $i$  thì ghi dãy đỉnh xuất hiện lần lượt trên đường đi, đỉnh đầu tiên là  $s$ , đỉnh cuối cùng là  $i$ . Đường đi từ đỉnh  $i$  tới chính đỉnh đó được coi là không tồn tại,  $i = 1..n$ . Thí dụ trên cho ta kết quả

DIJ.OUT													
4	2	1											
0													
1	2	3											
3	2	3	7	4									
0													
5	2	3	7	4	6								
2	2	3	7										

- Đường ngắn nhất từ đỉnh 2 đến đỉnh 1 có chiều dài 4:  $2 \rightarrow 1$ .
- Đường ngắn nhất từ đỉnh 2 đến đỉnh 2: không có (thực ra, theo lẽ thường là có đường chiều dài 0).
- Đường ngắn nhất từ đỉnh 2 đến đỉnh 3 có chiều dài 1:  $2 \rightarrow 3$ .
- Đường ngắn nhất từ đỉnh 2 đến đỉnh 4 có chiều dài 3:  $2 \rightarrow 3 \rightarrow 7 \rightarrow 4$ .
- Đường ngắn nhất từ đỉnh 2 đến đỉnh 5: không có.
- Đường ngắn nhất từ đỉnh 2 đến đỉnh 6 có chiều dài 5:  $2 \rightarrow 3 \rightarrow 7 \rightarrow 4 \rightarrow 6$ .
- Đường ngắn nhất từ đỉnh 2 đến đỉnh 7 có chiều dài 2:  $2 \rightarrow 3 \rightarrow 7$ .

## Thuật toán Dijkstra

Thuật giải quy hoạch động được trình bày dưới đây mang tên Dijkstra, một nhà tin học lỗi lạc người Hà Lan. Bản chất của thuật toán là *sửa đỉnh*, chính xác là *sửa trọng số* của mỗi đỉnh.

Theo sơ đồ giải các bài toán quy hoạch động trước hết ta xây dựng hệ thức cho bài toán.

### Hệ thức

Gọi  $p(i)$  là độ dài đường ngắn nhất từ đỉnh  $s$  đến đỉnh  $i$ ,  $1 \leq i \leq n$ . Ta thấy, hàm  $p(i)$  phải thoả các tính chất sau:

a)  $p(s) = 0$ : đường ngắn nhất từ đỉnh xuất phát  $s$  đến chính đỉnh đó có chiều dài 0.

b) Với  $i \neq s$ , muốn đến được đỉnh  $i$  ta phải đến được một trong các đỉnh sát trước đỉnh  $i$ . Nếu  $j$  là một đỉnh sát trước đỉnh  $i$ , theo điều kiện của đầu bài ta phải có

$$c[j, i] > 0$$

trong đó  $c[j, i]$  chính là chiều dài cung  $(j \rightarrow i)$ .

Trong số các đỉnh  $j$  sát trước đỉnh  $i$  ta cần chọn đỉnh nào?

Kí hiệu  $\text{path}(x, y)$  là đường đi ngắn nhất qua các đỉnh, xuất phát từ đỉnh  $x$  và kết thúc tại đỉnh  $y \neq x$ . Khi đó đường từ  $s$  đến  $i$  sẽ được chia làm hai đoạn: đường từ  $s$  đến đỉnh  $j$  là đỉnh sát trước đỉnh  $i$  và cung  $(j \rightarrow i)$ :

$$\text{path}(s, i) = \text{path}(s, j) + \text{path}(j, i)$$

trong đó  $\text{path}(j, i)$  chỉ gồm một cung  $j \rightarrow i$ .

Do  $p(i)$  và  $p(j)$  phải là ngắn nhất, tức là phải đạt các trị min, ta suy ra điều kiện để chọn đỉnh  $j$  sát trước đỉnh  $i$  là tổng chiều dài đường từ  $s$  đến  $j$  và chiều dài cung  $(j \rightarrow i)$  là ngắn nhất. Ta thu được hệ thức sau:

$$p(i) = \min \{p(j) + c[j][i] \mid c[j][i] > 0, j = 1..n\}$$

Để ý rằng điều kiện  $c[j, i] > 0$  cho biết  $j$  là đỉnh sát trước đỉnh  $i$ .

Điều tài tình là Dijkstra đã cung cấp thuật toán tính đồng thời mọi đường đi ngắn nhất từ đỉnh  $s$  đến các đỉnh còn lại của đồ thị. Thuật toán đó như sau.

Thuật toán thực hiện  $n-1$  lần lặp, mỗi lần lặp ta chọn và xử lí 1 đỉnh của đồ thị. Tại lần lặp thứ  $k$ , ta khảo sát phần của đồ thị gồm  $k$  đỉnh với các cung liên quan đến  $k$  đỉnh được chọn trong phần đồ thị đó. Ta gọi phần này là đồ thị con thu được tại bước xử lí thứ  $k$  của đồ thị ban đầu và kí hiệu là  $G(k)$ . Với đồ thị này ta hoàn tất bài giải tìm mọi đường đi ngắn nhất từ đỉnh xuất phát  $s$  đến mọi đỉnh còn lại của  $G(k)$ . Chiều dài thu được tại bước trung gian này, ta gán cho mỗi đỉnh  $i$  như một trọng số  $p[i]$ . Ngoài ra, để chuẩn bị cho bước tiếp theo ta đánh giá lại trọng số cho mọi đỉnh kế sau của các đỉnh trong  $G(k)$ .

Khởi trị: Gán trọng số  $p[i] = \infty$  cho mọi đỉnh, trừ đỉnh xuất phát  $s$ , gán trị  $p[s] = 0$ .



Edsger Wybe Dijkstra  
1930-2002

Ý nghĩa của thao tác này là khi mới đứng ở đỉnh xuất phát  $s$  của đồ thị con  $G(0)$ , ta coi như chưa thăm mảnh nào của đồ thị nên ta chưa có thông tin về đường đi từ  $s$  đến các đỉnh còn lại của đồ thị ban đầu. Nói cách khác ta coi như chưa có đường đi từ  $s$  đến các đỉnh khác  $s$  và do đó, độ dài đường đi từ  $s$  đến các đỉnh đó là  $\infty$ .

Giá trị  $\infty$  được chọn trong chương trình là:

**MAXLEN = tổng độ dài các cung trong đồ thị + 10**

Tại bước lặp thứ  $k$  ta thực hiện các thao tác sau:

- Trong số các đỉnh chưa xử lí, tìm đỉnh  $i$  có trọng số min.
- Với mỗi đỉnh  $j$  chưa xử lí và kề sau với đỉnh  $i$ , ta chỉnh lại trọng số  $p[j]$  của đỉnh đó theo tiêu chuẩn sau:

Nếu  $p[i] + c[i, j] < p[j]$  thì gán cho  $p[j]$  giá trị mới:

**$p[j] = p[i] + c[i, j]$**

Ý nghĩa của thao tác này là: nếu độ dài đường đi  $\text{path}(s, j)$  trong đồ thị con  $G(k-1)$  không qua đỉnh  $i$  mà lớn hơn độ dài đường đi mới  $\text{path}(s, j)$  có qua đỉnh  $i$  thì cập nhật lại theo đường mới đó.

- Sau khi cập nhật ta cần lưu lại vết cập nhật đó bằng lệnh gán  $\text{dinhTruoc}[i] = j$  với ý nghĩa là, đường ngắn nhất từ đỉnh  $s$  tới đỉnh  $j$  cần đi qua đỉnh  $i$ .
- Đánh dấu đỉnh  $i$  là đã xử lí.

Như vậy, tại mỗi bước lặp ta chỉ xử lí đúng một đỉnh  $i$  có trọng số min và đánh dấu duy nhất đỉnh đó.

Thuật toán chứa hai vòng for lồng nhau do đó có độ phức tạp là  $n^2$ .

Sau khi hoàn thành thuật toán Dijkstra ta cần gọi thủ tục Ket (kết) để ghi lại kết quả theo yêu cầu của đầu bài như sau.

Với mỗi đỉnh  $i = 1..n$  ta cần ghi vào tệp output chiều dài đường đi từ  $s$  đến  $i$  bao gồm giá trị  $p[i]$  và các đỉnh nằm trên đường đó.

Chú ý rằng nếu  $p[i]$  nhận giá trị khởi đầu tức là MAXLEN thì tức là không có đường đi từ  $s$  đến  $i$ .

Về ý nghĩa, mảng  $\text{dinhTruoc}$  chứa các con trỏ ngược từ mỗi đỉnh  $i$  đến đỉnh sát trước đỉnh  $i$  trên đường đi ngắn nhất, do đó ta phải lần ngược bằng thủ tục đệ quy  $\text{path}(i)$  để ghi vào tệp  $g$  vết của đường đi theo trật tự từ  $s$  đến  $i$ .

---

#### Thuật toán Dijkstra

---

Input: đồ thị  $G$ ; đỉnh xuất phát  $s$

Output: mọi đường ngắn nhất từ  $s$   
đến các đỉnh của  $G$

---

Begin

Gán các đỉnh của  $G$  trọng số  $p[i] = \infty$

$p[s] = 0$  // đỉnh xuất phát  $s$  có trọng số 0

lặp  $n-1$  lần với các đỉnh chưa thăm

    Tìm đỉnh  $i$  có trọng số min

    Đánh dấu đỉnh  $i$  : đã thăm

    Xét các đỉnh  $j$  kề với đỉnh  $i$

        if  $p[i] + c[i][j] < p[j]$  then

$p[j] = p[i] + c[i][j]$  // cập nhật  $j$

        end if

    kết thúc lặp

---

---

Giải trình kết quả  
End Dijkstra

---

Ta minh hoạ tiến trình hoạt động của thuật toán Dijkstra qua ví dụ đã cho.

Sau khi đọc dữ liệu từ tệp  $f = \text{DIJ.INP}$  ta có  $n = 7, s = 2$ . Đồ thị có 7 đỉnh, đỉnh xuất phát là 2. Ma trận kề  $c$  thu được như sau:

$c$	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	4	0	1	0	0	0	5
3	0	0	0	0	0	0	1
4	0	0	0	0	0	2	0
5	0	0	0	3	0	0	0
6	1	0	0	0	0	0	5
7	0	0	0	1	0	0	0

Khởi trị

Đỉnh	$p$	dinhTruoc
1	$\infty$	0
2	0	0
3	$\infty$	0
4	$\infty$	0
5	$\infty$	0
6	$\infty$	0
7	$\infty$	0

Ta sẽ thực hiện  $n-1 = 6$  bước lặp.

**Bước lặp  $k = 1$**

$i = \min = 2$  với  $p[2] = 0$ .

Các đỉnh chưa xử lí và kề với đỉnh 2 sẽ được sửa trọng số là 1, 3, 7 (có dấu ✓).

Vì  $p[2] + c[2][1] = 0 + 4 = 4 < p[1] = \infty$  nên  $p[1]$  được sửa thành 4 và  $dinhTruoc[1]$  được sửa thành 2.

Vì  $p[2] + c[2][3] = 0 + 1 = 1 < p[3] = \infty$  nên  $p[3]$  được sửa thành 1 và  $dinhTruoc[3]$  được sửa thành 2.

Vì  $p[2] + c[2][7] = 0 + 4 = 4 < p[7] = \infty$  nên  $p[7]$  được sửa thành 5 và  $dinhTruoc[7]$  được sửa thành 2.

Đỉnh		$p$	dinhTruoc	
1	✓	$\infty/4$	2	(2) → (1): 4
2	*	0	0	(2) → (3): 1
3	✓	$\infty/1$	2	(2) → (7): 5
4		$\infty$	0	
5		$\infty$	0	
6		$\infty$	0	
7	✓	$\infty/5$	2	

✓ đỉnh cần sửa trọng số  
\* đỉnh đã thăm

**Bước lặp  $k = 2$**



$i = \min = \underline{3}$  với  $p[3] = 1$ .

Đỉnh chưa xử lí và kề với đỉnh 3 sẽ được sửa trọng số là đỉnh 7.

Vì  $p[3] + c[3][7] = 1 + 1 = 2 < p[7] = 5$  nên  $p[7]$  được sửa thành 2 và  $dinhTruoc[7]$  được sửa thành 3.

Đỉnh		p	dinhTruoc	
1		$\infty/4$	2	(3) $\rightarrow$ (7): 1
2	*	0	0	
<u>3</u>	*	$\infty/1$	2	
4		$\infty$	0	
5		$\infty$	0	
6		$\infty$	0	
7	✓	$\infty/5/2$	2/3	

✓ đỉnh cần sửa trọng số  
\* đỉnh đã thăm

### Bước lặp $k = 3$

$i = \min = \underline{7}$  với  $p[7] = 2$

Đỉnh chưa xử lí và kề với đỉnh 7 sẽ được sửa trọng số là đỉnh 4.

Vì  $p[7] + c[7][4] = 2 + 1 = 3 < p[4] = \infty$  nên  $p[4]$  được sửa thành 3 và  $dinhTruoc[4]$  được sửa thành 7.

Đỉnh		p	dinhTruoc	
1		$\infty/4$	2	(7) $\rightarrow$ (4): 1
2	*	0	0	
3	*	$\infty/1$	2	
4	✓	$\infty/3$	7	
5		$\infty$	0	
6		$\infty$	0	
<u>7</u>	*	$\infty/5/2$	2/3	

✓ đỉnh cần sửa trọng số  
\* đỉnh đã thăm

### Bước lặp $k = 4$

$i = \min = \underline{4}$  với  $p[4] = 3$ .

Đỉnh chưa xử lí và kề với đỉnh 4 sẽ được sửa trọng số là đỉnh 6.

Vì  $p[4] + c[4][6] = 3 + 2 = 5 < p[6] = \infty$  nên  $p[6]$  được sửa thành 5 và  $dinhTruoc[6]$  được sửa thành 4.

Đỉnh		p	dinhTruoc	
1		$\infty/4$	2	(4) $\rightarrow$ (6): 2
2	*	0	0	
3	*	$\infty/1$	2	
<u>4</u>	*	$\infty/3$	7	
5		$\infty$	0	
6	✓	$\infty/5$	4	
7	*	$\infty/5/2$	2/3	

✓ đỉnh cần sửa trọng số  
\* đỉnh đã thăm

**Bước lặp  $k = 5$** 

$i = \min = \underline{1}$  với  $p[1] = 4$ .

Không có đỉnh chưa xử lí nào kề với đỉnh 1.

Đỉnh		p	dinhTruoc	
<u>1</u>	*	$\infty/4$	2	(1) $\rightarrow$ ?
2	*	0	0	
3	*	$\infty/1$	2	
4	*	$\infty/3$	7	
5		$\infty$	0	
6	*	$\infty/5$	4	
7	*	$\infty/5/2$	2/3	

✓ đỉnh cần sửa trọng số  
\* đỉnh đã thăm

**Bước lặp  $k = 6$** 

$i = \min = \underline{6}$  với  $p[6] = 5$ .

Không có đỉnh chưa xử lí nào kề với đỉnh 6. Chú ý rằng đỉnh 1 và đỉnh 7 kề với đỉnh 6 nhưng hai đỉnh này đã xử lí rồi.

Đỉnh		p	dinhTruoc	
1	*	$\infty/4$	2	(6) $\rightarrow$ (1)* đã thăm (6) $\rightarrow$ (5)* đã thăm
2	*	0	0	
3	*	$\infty/1$	2	
4	*	$\infty/3$	7	
5		$\infty$	0	
<u>6</u>	*	$\infty/5$	4	
7	*	$\infty/5/2$	2/3	

✓ đỉnh cần sửa trọng số  
\* đỉnh đã thăm

Thuật toán dừng sau 6 bước lặp.

Lưu ý rằng đỉnh xuất phát cho bài toán này là  $s = 2$ . Ta minh hoạ giải trình kết quả cho ba ví dụ sau.

Đỉnh	p	dinhTruoc	Ba ví dụ
1	4	2	<i>Giải trình đường ngắn nhất 2 <math>\rightarrow</math> 4?</i>
2	0	0	<i>Giải trình đường ngắn nhất 2 <math>\rightarrow</math> 5?</i>
3	1	2	<i>Giải trình đường ngắn nhất 2 <math>\rightarrow</math> 2?</i>
4	3	7	
5	$\infty$	0	
6	5	4	
7	2	3	

**Đường đi ngắn nhất từ đỉnh  $s = 2$  đến đỉnh  $t = 4$  ?**

Vì  $p[4] = 3$  nên độ dài đường đi là 3.

Để giải trình vết của đường đi từ 2 đến 4 ta dựa vào mảng  $dinhTruoc[1..7]$  như sau:

Vì  $\text{dinhTruoc}[4] = 7$ , tức là trước khi đến đỉnh 4 phải qua đỉnh 7 nên ta có

$$7 \rightarrow 4$$

Vì  $\text{dinhTruoc}[7] = 3$ , tức là trước khi đến đỉnh 7 phải qua đỉnh 3 nên ta có

$$3 \rightarrow 7 \rightarrow 4$$

Vì  $\text{dinhTruoc}[3] = 2$ , tức là trước khi đến đỉnh 3 phải qua đỉnh 2 nên ta có

$$2 \rightarrow 3 \rightarrow 7 \rightarrow 4$$

Kết quả này được ghi ở dòng thứ tư của tệp `DIJ.OUT` như sau:

$$3 \ 2 \ 3 \ 7 \ 4$$

trong đó số đầu tiên 3 cho biết chiều dài đường đi, dãy số còn lại giải trình vết của đường đi từ đỉnh 2 đến đỉnh 4.

**Đường đi ngắn nhất từ đỉnh  $s = 2$  đến đỉnh  $t = 5$  ?**

Vì  $p[5] = \infty$  ứng với giá trị dương vô cùng  $\infty$  khởi trị lúc đầu nên không có đường đi từ đỉnh 2 đến đỉnh 5.

Ta ghi kết quả 0 tại dòng 5 của tệp `DIJ.OUT`.

**Đường đi ngắn nhất từ đỉnh  $s = 2$  đến đỉnh  $t = 2$  ?**

Vì  $s = t$  nên ta coi như không có đường đi từ đỉnh 2 đến đỉnh 2.

Ta ghi kết quả 0 tại dòng 5 của tệp `DIJ.OUT`.

## Chương trình C++

```
// Dijkstra
#include <bits/stdc++.h>

using namespace std;

const int MN = 101;

int p[MN]; // trong so
int n; // so luong dinh
int s; // dinh xuat phat
int c[MN][MN]; // ma tran ke
int maxlen; // tong do dai duong
int dinhTruoc[MN];
bitset<MN> used; // dinh da tham

ofstream g("DIJ.OUT");

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Hien thi mang x[d..c]
void Print(int x[], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i)
        cout << " " << x[i];
```

```

}

void Init() {
    // Gán trong số các đỉnh
    for(int i = 1; i <= n; ++i)
        p[i] = maxlen;
    p[s] = 0; // Đỉnh xuất phát s có trọng số 0
    used.reset(); // all used = 0: các đỉnh đều chưa tham
}

// Tìm đỉnh có trọng số min
// trong số các đỉnh chưa tham
int Min() {
    int maxd = maxlen + 1;
    int imin;
    for(int i = 1; i <= n; ++i) {
        if (!used[i]) { // đỉnh i chưa tham
            if (p[i] < maxd) {
                imin = i;
                maxd = p[i];
            }
        }
    }
    return imin;
}

void ReadInput() {
    ifstream f("DIJ.INP");
    f >> n >> s;
    maxlen = 10;
    cout << "\n " << n << " đỉnh, " << s << ": đỉnh xuất phát.";
    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= n; ++j) {
            f >> c[i][j];
            maxlen += c[i][j];
        }
    }
    f.close();
    cout << "\n ma tran ke:";
    for(int i = 1; i <= n; ++i) {
        cout << endl;
        for(int j = 1; j <= n; ++j) {
            cout << " " << c[i][j];
        }
    }
    cout << "\n maxlen = " << maxlen;
    memset(dinhTruoc, 0, sizeof(dinhTruoc));
}

void Dijkstra() {
    int i;
    for(int k = 1; k < n; ++k) {
        i = Min(); // tìm đỉnh i có trọng số p[i] -> min
    }
}

```

```

        used[i] = 1; // đã tham dinh i
        // cout << "\n Dinh trong so min " << i;
        // sua lai trong so cac dinh chua tham
        for(int j = 1; j <= n; ++j) {
            if (!used[j]) { // dinh j chua tham
                if (c[i][j] > 0) { // có đường i->j
                    if (p[i] + c[i][j] < p[j]) {
                        // sua trong so dinh j
                        p[j] = p[i] + c[i][j];
                        dinhTruoc[j] = i;
                    }
                }
            }
        }
    } // for j
} // for k
}

void Path(int i) {
    if (i == 0) return;
    Path(dinhTruoc[i]);
    g << i << " ";
}

void Run() {
    ReadInput();
    Init();
    Dijkstra();
    // Giai trình ket qua
    for(int i = 1; i <= n; ++i) {
        if (p[i] == maxlen) p[i] = 0;
        g << p[i] << " ";
        if (p[i] > 0) Path(i);
        g << endl;
    }
    g.close();
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

### Cải tiến: Dùng Min Heap

Ta có thể dùng min heap để giảm độ phức tạp của thuật toán Dijkstra từ  $O(n^2)$  xuống đến  $O(n \log(n))$  như sau.

Trước hết ta xây dựng min heap h từ các trọng số đỉnh.

Tại bước khởi trị ta đặt trọng số maxlen, tức là giá trị vô cùng ( $\infty$ ) cho các đỉnh, riêng với đỉnh xuất phát ta đặt trọng số 0.

```

h.clear();
p[s] = 0; Ins(s); // dinh xuất phát s có trọng số 0
for i = 1..n do
    if i != s
        p[i] = maxlen;
        Ins(i); // nạp heap
    end if
end for

```

Điểm thứ vị là h chứa các đỉnh chưa xử lý. Tại mỗi bước lặp ta lấy phần tử i từ heap h có trọng số p[i] min rồi xét các đỉnh j trong h thỏa điều kiện:

```

j kề với i: c[i][j] > 0 và
p[i] + c[i][j] < p[j]

```

để cập nhật lại trọng số của đỉnh j.

---



---

#### Thuật toán Dijkstra (dùng min heap h)

---

Input: đồ thị G; đỉnh xuất phát s

Output: mọi đường ngắn nhất từ s  
đến các đỉnh của G

---

Begin

  Tạo heap h

    từ các đỉnh i, với trọng số maxlen, i=1..n, i ≠ s  
    và đỉnh i với trọng số 0.

  lặp n lần

    i = Take() // Lấy phần tử i min từ h: p[i] → min

    // h tự động cập nhật

    // Xét các đỉnh j trong h kề với đỉnh i

    for v = 0..h.size() do

      j = h[v] // đỉnh j

      if c[i][j] > 0 // i → j

        and p[i] + c[i][j] < p[j]

        // cập nhật trọng số đỉnh j

        p[j] = p[i] + c[i][j]

        Up(v) // Cập nhật h

        dinhTruoc[j] = i

      end if

    end for

  kết thúc lặp

  Giải trình kết quả

End Dijkstra

---



---

## Chương trình C++

```

/*****
Dijkstra (using Min Heap)
*****/

```

```

#include <bits/stdc++.h>

using namespace std;

typedef vector<int> VI;

VI h; // min heap

const int MN = 101; // so dinh toi da
int n; // so dinh
int s; // dinh xuất phát
int c[MN][MN]; // ma tran ke
int maxlen; // gia tri vo cung
int dinhTruoc[MN];
int p[MN]; // trong so cac dinh da xu li

ofstream g("DIJ.OUT"); // file ket qua

void Go() {
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Show Heap
void Show(VI h, char * msg = "") {
    cout << msg;
    for(int i = 0; i < h.size(); ++i)
        cout << "\n " << h[i] << " -> " << p[h[i]];
}

// Tu node cha di xuong
int Down(int cha) {
    int n = h.size();
    int idcha = h[cha]; // so hieu cua node cha
    int pcha = p[idcha]; // trong so node cha
    int con; // c: node con, p: node cha
    while(true) {
        con = cha + cha + 1; // con trai
        if (con >= n) break; // khong co node con
        if (con + 1 < n) { // neu co con phai
            // c la node nho nhât trong 2 con
            // con phai = con + 1
            if (p[h[con+1]] < p[h[con]]) ++con;
        }
        // so sanh cha con
        if (pcha <= p[h[con]]) break;
        // cha > con
        h[cha] = h[con]; // cha <= con
        cha = con; // chuyen con len / cha xuong
    }
    h[cha] = idcha;
    return 1;
}

```

```

// Chinh lai h tu h[con] ve truoc
int Up(int con) { // con
    int cha;
    int idcon = h[con]; // so hieu tai node con
    int pcon = p[idcon]; // gia tri tai node con
    while (true) {
        if (con == 0) break; // khong co node tren
        cha = (con+1) / 2 - 1; // cha
        if (p[h[cha]] <= pcon) break; // heap chuan: cha <= con: OK
        // p[h[cha]] > pcon: cha > con
        h[con] = h[cha]; // chuyen con len vi tri cha
        con = cha;
    }
    h[con] = idcon;
    return 1;
}

// Them tri node i vao cuoi heap h
void Ins(int i) {
    h.push_back(i);
    Up(h.size()-1);
}

// Lay phan tu min tai heap h[0]
int Take() {
    if (h.size() == 0) return -1; // heap rong
    int id = h[0]; // p[id] -> min
    h[0] = h[h.size()-1]; // dua cuoi len dau
    h.pop_back(); // bo phan tu cuoi
    // chinh lai heap sau khi mat dau
    if (!h.empty()) Down(0);
    return id;
}

void ReadInput() {
    ifstream f("DIJ.INP");
    f >> n >> s; // n: so dinh, s: dinh xuat phat
    maxlen = 10; // gia tri vo cung
    cout << "\n " << n << " dinh, " << s << ": dinh xuat phat.";
    cout << "\n ma tran ke:";
    for(int i = 1; i <= n; ++i) { // ma tran ke
        cout << endl;
        for(int j = 1; j <= n; ++j) {
            f >> c[i][j];
            cout << " " << c[i][j];
            maxlen += c[i][j];
        }
    }
    f.close();
    cout << "\n maxlen = " << maxlen;
    memset(dinhTruoc, 0, sizeof(dinhTruoc));
}

```



```

void Init() {
    // Tao min heap
    // Gan trong so cac dinh
    h.clear();
    p[s] = 0; Ins(s); // dinh xuất phát s có trọng số 0
    for(int i = 1; i <= n; ++i) {
        if (i != s) { // các đỉnh != s: p = maxlen
            p[i] = maxlen;
            Ins(i); // nạp heap
        }
    }
}

void Dijkstra() {
    int i, j;
    for(int k = 1; k <= n; ++k) { // lặp n lần
        i = Take(); // tìm đỉnh i có trọng số min
        cout << "\n Đỉnh " << i << " có trọng số Min = " << p[i];
        // Tìm các đỉnh j kề với i trong h: i -> j
        for(int v = 0; v < h.size(); ++v) {
            j = h[v];
            if (c[i][j] > 0) { // i->j
                if (p[i] + c[i][j] < p[j]) {
                    p[j] = p[i] + c[i][j];
                    dinhTruoc[j] = i;
                    cout << "\n " << i << " -> " << j;
                    Up(v);
                }
            }
        }
    }
}

void Path(int i) {
    if (i == 0) return;
    Path(dinhTruoc[i]);
    g << i << " ";
}

void WriteResult() {
    // Giai trình kết quả
    for(int i = 1; i <= n; ++i) {
        if (p[i] == maxlen) p[i] = 0;
        g << p[i] << " ";
        if (p[i] > 0) Path(i);
        g << endl;
    }
    g.close();
}

void Run() {
    ReadInput();
}

```

```

    Init();
    Show(h, "\n Min heap h: ");
    Dijkstra();
    WriteResult();
}

main() {
    Run();
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

7 dinh, 2: dinh xuất phát.
ma tran ke:
0 0 0 0 0 0 0
4 0 1 0 0 0 5
0 0 0 0 0 0 1
0 0 0 0 0 2 0
0 0 0 3 0 0 0
1 0 0 0 0 0 5
0 0 0 1 0 0 0
maxlen = 33
Min heap h:
2 -> 0
1 -> 33
3 -> 33
4 -> 33
5 -> 33
6 -> 33
7 -> 33
Dinh 2 co trong so Min = 0
2 -> 7
2 -> 1
2 -> 3
Dinh 3 co trong so Min = 1
3 -> 7
Dinh 7 co trong so Min = 2
7 -> 4
Dinh 4 co trong so Min = 3
4 -> 6
Dinh 1 co trong so Min = 4
Dinh 6 co trong so Min = 5
Dinh 5 co trong so Min = 33
T h e   E n d

```

## Các dạng khác của bài toán Dijkstra

Lưu ý rằng ma trận kề có thể chứa các giá trị thực, tuy nhiên cần giả thiết rằng mọi giá trị trong ma trận kề phải là các số không âm. Với các số âm bài toán sẽ phức tạp hơn.

Dưới đây liệt kê hai dạng khá thông dụng P1, và P2 của bài toán tìm đường đi trong đồ thị.

P1. Nếu đồ thị đã cho là vô hướng ta giải như trên, chỉ lưu ý đến tính đối xứng khi đọc dữ liệu vào ma trận kề  $a$ .

P2. Nếu đề bài chỉ yêu cầu tìm một đường đi từ đỉnh  $s$  đến đỉnh  $t$  ta thực hiện các bước sau:

1. Đọc dữ liệu.
2. Gọi thuật toán Dijkstra.
3. Ghi kết quả  $p[t]$  và giải trình một đường theo thuật toán  $\text{path}(t)$ .

## CHƯƠNG 8 SUY NGẪM

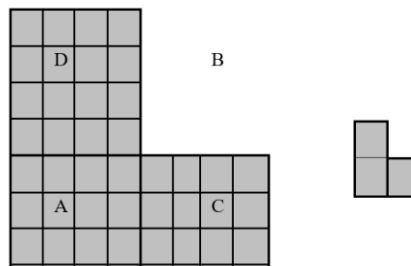
Chương này giới thiệu một số bài toán thuộc các lớp thuật giải khác nhau để bạn đọc tự luyện tập.

Thông thường, nếu chỉ biết một phương pháp giải mà gặp bài toán "trúng tủ", nghĩa là bài toán vận dụng chính phương pháp đã biết thì ta gần như không phải suy nghĩ gì. Tuy nhiên, khi đã có trong tay một số phương pháp thì việc chọn thuật giải cho mỗi bài toán cụ thể sẽ không dễ dàng chút nào.

Luyện tập và suy ngẫm để tìm kiếm đường đi trong các tình huống như trên sẽ cung cấp cho chúng ta nhiều kinh nghiệm quý.

### Bài 8.1. Lát nền

Người ta cần lát kín một nền nhà hình vuông cạnh dài  $n = 2^k$ , ( $k$  là một số tự nhiên trong khoảng 1..6) khuyết một phần tư tại góc trên phải bằng những viên gạch màu hình thước thợ (chữ L) tạo bởi 3 ô vuông đơn vị như trong hình 2b. Hai viên gạch kề cạnh nhau dù chỉ 1 đơn vị dài, phải có màu khác nhau. Hãy cho biết một cách lát với số màu ít nhất.



Hình 2

Dữ liệu vào: số tự nhiên  $n$ .

Dữ liệu ra: tệp văn bản NEN.OUT. Dòng đầu tiên là hai số tự nhiên  $m$  biểu thị số viên gạch và  $c$  là số màu cần dùng cho việc lát nền. Tiếp đến là một phương án lát nền tìm được, trong đó mỗi viên gạch lát được tạo bởi ba chữ số giống nhau thể hiện màu của viên gạch đó. Các số trên mỗi dòng cách nhau qua dấu cách.

Ví dụ, với  $n = 8$  ta có một cách lát nền như sau:

INPUT	NEN.OUT	
8	16 3	
	3 3 1 1	3 3 1 1
	3 2 2 1	3 2 2 1
	1 2 3 3	1 2 3 3
	1 1 3 2	1 1 3 2
	3 3 1 2 2 3 1 1	3 3 1 2 2 3 1 1
	3 2 1 1 3 3 2 1	3 2 1 1 3 3 2 1
	1 2 2 3 1 2 2 3	1 2 2 3 1 2 2 3
	1 1 3 3 1 1 3 3	1 1 3 3 1 1 3 3

3	3	1	1				
3	2	2	1				
1	2	3	3				
1	1	3	2				
3	3	1	2	2	3	1	1
3	2	1	1	3	3	2	1
1	2	2	3	1	2	2	3
1	1	3	3	1	1	3	3

## Thuật toán

Lời giải sau đây của bạn Lê Hoàng Hải, lớp 10 chuyên Tin, Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội (năm 2002).

Để tính số viên gạch ta lấy ba phần tư diện tích hình vuông phủ nền nhà chia cho 3 là diện tích 1 viên gạch thước thợ:

$$\text{sogach} = ((3*n*n) \text{ div } 4) \text{ div } 3;$$

Về số màu, với  $n = 2$  thì chỉ cần 1 viên gạch màu 1. Với mọi  $n = 2^k > 2$  ta sẽ trình bày một thuật toán cần tối đa ba màu.

Đầu tiên ta gọi thủ tục `Init` để khởi trị với hình vuông cạnh  $v = 2$  nằm ở góc dưới trái của nền nhà được biểu diễn dưới dạng một mảng hai chiều `a`: ba ô trong hình vuông  $2 \times 2$  sẽ được điền giá trị 1, ô nằm ở góc trên phải được điền giá trị 2 (phần tô đậm, hình 6).



3	3	1	1				
3	2	2	1				
1	2	3	3				
1	1	3	2				
3	3	1	2	2	3	1	1
3	2	1	1	3	3	2	1
1	2	2	3	1	2	2	3
1	1	3	3	1	1	3	3

Hình 6

Gọi hình được khởi trị là A. Mỗi bước tiếp theo ta thực hiện *ba thao tác biến hình* sau đây:

- Tịnh tiến A đi lên theo đường chéo để thu được hình B (xem thủ tục `DichCheo`).
- Lật A sang phải (tức là lấy đối xứng gương qua trục tung) để thu được hình C (xem thủ tục `LatPhai`).
- Lật A lên trên (tức là lấy đối xứng gương qua trục hoành) để thu được hình D (xem thủ tục `LatLen`).

Chú ý rằng khi lật ta cần thực hiện thêm phép hoán đổi trị 1 và 3 cho nhau.

- Mỗi lần lặp như vậy ta sẽ thu được hình vuông có cạnh tăng gấp đôi hình trước. Khi  $v = n$  ta gọi thủ tục Fini để ghi tệp kết quả.

```

Init // khởi trị: lát hình 2x2
for v = 2..n do
    LatPhai(v);
    LatLen(v);
    if (2*v < n) DichCheo(v) end if;
    v = v * 2
end for
Ghi kết quả;

```

Điểm  $i$  trên trục số đối xứng với điểm  $j$  qua tâm  $t$  có hệ thức

$$\frac{i + j}{2} = t$$

0	1	<b>2</b>	3	<b>4</b>	5	<b>6</b>	7	8	9
		$i$		$t$		$j$			

Biết  $i$  và tâm đối xứng  $t$  ta dễ dàng tính được giá trị của  $j$  theo công thức

$$j = 2t - i$$

Công thức này sẽ được sử dụng để tính vị trí của các chỉ số của  $a$  khi lật phải (đối xứng qua trục dọc td) hoặc lật lên (đối xứng qua trục ngang tn).

```

/*-----
Lat hình vuông cạnh v, a[n-v+1..n,1..v]
o goc duoi trai sang phai, doi mau gach
de nhan duoc manh C.
-----*/
LatPhai(v):
    td = 2*v-1; // trục dọc
    for i = n-v..n-1 do
        for j = 0..v-1 do
            a[i][td-j] = 4-a[i][j]
        end for
    end for

```

```

/*-----
Lat hình vuông cạnh v, a[n-v+1..n,1..v]
o goc duoi len tren, doi mau gach

```

```

    de nhan duoc manh D.
-----*/
LatLen(v):
    tn = 2*(n - v) - 1; // truc ngang
    for i = n-v..n-1 do
        for j = 0..v-1 do
            a[tn-i][j] = 4-a[i][j]
        end for
    end for

```

```

/*-----
Dich hình vuông A cạnh v, a[n-v+1..n,1..v]
o góc dưới trái đi lên theo đường chéo
chính của nên nhà để nhận được manh B.
-----*/
DichCheo(v):
    ii = n-v-v;
    for i = n-v..n-1 do
        jj = v
        for j = 0..v-1 do
            a[ii][jj] = a[i][j]
            ++jj;
        end for
        ++ii;
    end for

```

Ta có thể kết hợp hai thủ tục LatPhai và LatLen vào một thủ tục gọi là Expand như sau:

```

Expand(v):
    td = 2*v-1; // truc doc
    tn = 2*(n - v) - 1;
    for i = n-v..n-1 do
        for j = 0..v-1 do
            a[i][td-j] = a[tn-i][j] = 4-a[i][j];
        end for
    end for

```

## Chương trình C++

```

/*****
LATNEN - lat nen nha
hình vuông khuyết góc bằng
các viên gạch màu hình L
*****/

#include <bits/stdc++.h>

```

```

using namespace std;

const int MN = 65; // kích thước tối đa của nền nhà
const char BL = 32; // dấu cách
int n; // chiều dài cạnh nền nhà
char a[MN][MN]; // a[i] - màu viên gạch lát
int n2; // n2 = n/2

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// Hiển thị mảng x[d..c]
void Print(int x[], int d, int c, const char * msg = "") {
    cout << msg;
    for (int i = d; i <= c; ++i)
        cout << " " << x[i];
}

void Init() {
    n2 = n / 2;
    // Đặt tam 4 ô vuông (2 X 2) tại góc dưới trái
    a[n-2][0] = 1; a[n-2][1] = 2;
    a[n-1][0] = a[n-1][1] = 1;
}

void Ket() {
    ofstream g("NEN.OUT");
    g << n2*n2 << " " << ((n==2) ? 1 : 3) << endl;
    // ghi góc trên trái D
    for(int i = 0; i < n2; ++i) {
        for(int j = 0; j < n2; ++j) {
            g << (int)a[i][j] << BL;
        }
        g << endl;
    }
    // ghi phần dưới
    for(int i = n2; i < n; ++i) {
        for(int j = 0; j < n; ++j) {
            g << (int)a[i][j] << BL;
        }
        g << endl;
    }
    g.close();
}

/*-----
Dịch hình vuông A cạnh v, a[n-v+1..n,1..v]
ở góc dưới trái đi lên theo đường chéo

```



```

    chinh cua nen nha de nhan duoc manh B.
    -----*/
void DichCheo(int v) {
    int ii = n-v-v, jj;
    for(int i = n-v; i < n; ++i, ++ii)
        for(int j = 0, jj = v; j < v; ++j, ++jj)
            a[ii][jj] = a[i][j];
}

// LatPhai & LatLen
void Expand(int v) {
    int td = 2*v-1; // truc doc
    int tn = 2*(n - v) - 1;
    for(int i = n-v; i < n; ++i) {
        for(int j = 0; j < v; ++j) {
            a[i][td-j] = a[tn-i][j] = 4-a[i][j];
        }
    }
}

void LatNen(int inp) {
    n = inp;
    Init(); // Khoi tri voi n = 2
    for(int v = 2; v < n; v *= 2) {
        Expand(v);
        if (v < n2) DichCheo(v);
    }
    Ket();
}

main() {
    LatNen(8);
    cout << "\n T h e   E n d";
    return 0;
}

```

### Bình luận

Thuật giải sử dụng hai phép biến hình cơ bản trong chương trình phổ thông là phép dời hình (tịnh tiến) và đối xứng qua trục. Việc hoán đổi trị 1 và 3 cho nhau là một ý tưởng thông minh. Mỗi ô trong bảng được điền đúng một lần, do đó độ phức tạp tính toán của thuật toán là  $n^2$ , trong khi các bài giải khác đều phải sử dụng các phép dò tìm để xác định màu tô và gọi đệ quy nên thường tốn kém về miền nhớ và thời gian hơn nhiều lần.

### Bài 8.2. Chữ số cuối khác 0

*Đề thi Tin học Quốc gia Ireland, 1994.*

*Tìm chữ số cuối cùng khác 0 của  $n!$  với  $n$  trong khoảng 1..100.*

### Ví dụ

- $n = 7$ ,  $n! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 = 5040$ . Kết quả = 4.
- $n = 15$ ,  $n! = 15! = 1307674368000$ . Kết quả = 8.

### Thuật toán

Thuật giải của bạn Việt Hưng (Hà Tây, 2002) cho phép mở rộng giới hạn của  $n$  đến hàng chục vạn và nếu bạn muốn, có thể tiếp tục mở rộng đến hàng triệu.

Ý tưởng chính của Việt Hưng nằm ở công thức:  $2 \times 5 = 10$ . Thật vậy, ta biết:

$$n! = 1.2.3...n$$

Các chữ số cuối cùng bằng 0 của  $n!$  được sinh ra khi và chỉ khi trong khai triển ra tích các thừa số nguyên tố có chứa các cặp thừa số 2 và 5. Vậy thì trước hết ta đếm số lượng các thừa số 2, kí hiệu là  $d2$  và số lượng các thừa số 5, kí hiệu là  $d5$ .

Ví dụ, với  $n = 15$  ta có dạng khai triển ra thừa số nguyên tố của  $n$  giai thừa như sau:

$$\begin{aligned} n! &= 1.2.3.4.5.6.7.8.9.10.11.12.13.14.15 = \\ &= 1.2.3.(2.2).5.(2.3).7.(2.2.2).9.(2.5).11.(2.2.3).13.(2.7).(3.5) = \\ &= 2^{11} \cdot 5^3 \cdot 3^6 \cdot 7^2 \cdot 11 \cdot 13 = 10^3 \cdot 2^8 \cdot 3^6 \cdot 7^2 \cdot 11 \cdot 13 \end{aligned}$$

Do đó  $d2 = 11$  và  $d5 = 3$ . Vậy ta có ba cặp  $2 \cdot 5 = 10$  và số mũ dôi ra của thừa số 2 so với thừa số 5 sẽ là  $d2 - d5 = 11 - 3 = 8$ . Khi đó, kết quả sẽ là:

chữ số cuối cùng khác 0 của  $15!$  = chữ số cuối cùng của  $k \cdot 2^{d2-d5}$ .

Trong đó  $k$  là tích của các thừa số còn lại.

$$k = 2^8 \cdot 3^6 \cdot 7^2 \cdot 11 \cdot 13$$

Dễ thấy với mọi  $n$ , ta luôn có  $d2 \geq d5$  vì cứ hai số liên tiếp thì có một số chẵn (chia hết cho 2), còn năm số liên tiếp mới có một số chia hết cho 5.

Việc còn lại là lấy tích  $k$  của các số còn lại. Vì tích này không bao giờ tận cùng bằng 0 cho nên ta chỉ cần giữ lại một chữ số cuối cùng bằng cách lấy mod 10.

$$\begin{aligned} k &= (2^8 \cdot 3^6 \cdot 7^2 \cdot 11 \cdot 13) \bmod 10 = \\ &= ((2^8 \bmod 10) \cdot (3^6 \cdot 7^2 \cdot 11 \cdot 13) \bmod 10) \bmod 10 \end{aligned}$$

Đặt

$$A = 2^8 \bmod 10$$

$$B = (3^6 \cdot 7^2 \cdot 11 \cdot 13) \bmod 10$$

Chữ số cuối cùng khác 0 của  $n!$  khi đó sẽ là

$$(A \cdot B) \bmod 10$$

Nếu  $p$  là một số nguyên tố và  $p^k$  là nhân tử trong dạng phân tích  $n!$  ra thừa số nguyên tố, thì  $k$  được tính bằng tổng của các thương nguyên trong phép chia liên tiếp số  $n$  cho  $p$ . Ta ký hiệu  $\text{Deg}(n, p) = k$  và gọi  $k$  là bậc của thừa số nguyên tố  $p$  trong  $n!$ .

Ví dụ, với  $n = 15$ ,  $p = 2$  và kí hiệu  $a:b$  là phép chia nguyên  $a$  cho  $b$ , ta tính được

$$\begin{array}{rcl} 15 & : & 2 = 7 \\ 7 & : & 2 = 3 \\ 3 & : & 2 = 1 \end{array}$$

$$\text{Deg}(15, 2) = \overset{11}{11}$$

Do đó  $k = 7+3+1 = 11$ .

Tương tự, ta tính được  $\text{Deg}(15, 5) = 3$  như sau

$$\begin{array}{r} 15 : 5 = 3 \\ 3 \\ \text{Deg}(15, 5) = 3 \end{array}$$

Như vậy  $15! = 2^{11} \cdot 5^3 \cdot C$ .

Chứng minh điều này khá dễ, bạn chỉ cần viết dãy  $n! = 1 \cdot 2 \cdot \dots \cdot n$  thành các dòng, mỗi dòng  $p$  thừa số

$$\begin{array}{ccccccc} 1 & 2 & & \dots & p & & \\ p+1 & p+2 & & \dots & 2p & & \\ \dots & & & \dots & & & \\ \dots & & & \dots & kp & & \\ \dots & & & n & & & \end{array}$$

Vậy là trong  $n!$  chứa  $k = n:p$  thừa số  $p$ .

$$n! = C \cdot (p \cdot 2p \cdot \dots \cdot kp) = C \cdot p^k \cdot (1 \cdot 2 \cdot \dots \cdot k) = C \cdot p^k \cdot k!$$

Tương tự, trong  $k!$  sẽ chứa  $k_1 = k:p$  thừa số  $p \dots$

Tổng cộng, trong tích  $n!$  sẽ chứa  $\text{Deg}(n, p)$  thừa số  $p$ .

```
// số lần xuất hiện của p trong n!
Deg(n, p):
  c = 0;
  while(n >= p) do
    n = n / p;
    c += n;
  end while
  return c
```

Nhờ hàm Deg ta dễ dàng tính được

```
d2 = Deg(15, 2) = 11
d5 = Deg(15, 5) = 3
m = d2 - d5 = 8
```

**Tính  $A = 2^m \bmod 10$**

Để tính chữ số tận cùng của  $2^m$  với  $m = d2 - d5 > 0$  ta để ý đến tính tuần hoàn của nó, cụ thể là ta chỉ cần tính chữ số tận cùng của  $2^{(m \bmod 4)}$  với các trường hợp:

$$m \bmod 4 = 0, 1, 2 \text{ và } 3.$$

Với  $m > 0$  ta có dãy tuần hoàn sau:

Chữ số tận cùng (khác 0) của  $2^m$ ,  $m > 0$  là 6, 2, 4, 8.

```
// Chữ số tận cùng (khác 0) của 2^m là 2^m mod 10
d[] = {6, 2, 4, 8}
return d[m % 4]
```

Theo ví dụ trên ta có

$\text{Deg}(15, 2) = 11$

$\text{Deg}(15, 5) = 3$

$m = \text{Deg}(15, 2) - \text{Deg}(15, 5) = 11 - 3 = 8$

Vậy Chữ số tận cùng của  $2^m$  là  $A = d[8 \% 4] = d[0] = 6$

### **Tính $(AB) \bmod 10$**

Để tính chữ số tận cùng của  $(AB) \bmod 10$  ta lần lượt duyệt các số  $v$  từ 2 đến  $n$  bỏ đi các thừa số 2 và 5 rồi lấy tích theo mod 10

```
// AB mod 10
m = A
for i = 3..n do
    v = i
    while(v % 2 == 0) v /= 2 end while // bỏ qua các thừa số 2
    while(v % 5 == 0) v /= 5 end while // và thừa số 5
    m = (m*v) % 10
end for
return m
```

Các chương trình sau đây chứa thủ tục find tìm chữ số cuối cùng khác 0 của  $n!$ .

## **Chương trình C++**

```
/*-----
Chu số khác không cuối cùng của n!
-----*/
#include <bits/stdc++.h>

using namespace std;

// Chu số tận cùng của  $2^m$ 
int d[] = {6, 2, 4, 8};

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// số lần xuất hiện của p trong n!
int Deg(int n, int p) {
    int c = 0;
    while(n >= p) {
        n = n / p;
        c += n;
    }
    return c;
}

/*-----
Chu số cuối khác 0 của n!
-----*/
```

```

-----*/
int Find(int n) {
    if (n == 0 | n == 1) return 1; // 0! = 1! = 1
    int m = d[(Deg(n,2) - Deg(n, 5)) % 4];
    int v;
    for (int i = 3; i <= n; ++i) {
        v = i;
        while(v % 2 == 0) v /= 2; // bỏ qua các thừa số 2
        while(v % 5 == 0) v /= 5; // và thừa số 5

        m = (m*v) % 10;
    }
    return m;
}

main() {
    cout << Find(15); // 8
    cout << "\n T h e   E n d ";
    return 0;
}

```

Một dạng khác của ứng dụng hàm Deg là bài toán sau đây:

### Số chữ số 0 đứng cuối của $n!$

Cho biết  $n!$  tận cùng với bao nhiêu chữ số 0. Ví dụ,  $15!$  tận cùng với 3 chữ số 0, vì

$15! = 1307674368000$ .

Đáp án: Deg( $n,5$ ). Bạn cần giải thích vì sao?

### Bài 8.3. Hình chữ nhật tối đại trong ma trận 0/1.

Cho một ma trận biểu diễn bằng một mảng hai chiều kích thước  $n \times m$  ô và chỉ chứa các ký tự 0 và 1. Tìm hình chữ nhật chứa toàn ký tự 1 và có diện tích lớn nhất (gọi là hình chữ nhật tối đại).

Dữ liệu vào: Tập văn bản CNMAX.INP:

- Dòng đầu tiên: số tự nhiên  $m$ ,  

$$3 \leq m \leq 70$$
- Tiếp đến là các dòng dài bằng nhau thể hiện một xâu gồm  $m$  ký tự là các chữ cái 0/1 viết liền nhau.
- Số dòng của tập input có thể lên tới 60 nghìn.

Dữ liệu ra: tập văn bản CNMAX.OUT :

- Dòng đầu tiên: Diện tích của hình chữ nhật tối đại ABCD chứa toàn ký tự 1 tìm được.
- Dòng thứ hai: Tọa độ dòng và cột của đỉnh A.
- Dòng thứ ba: Tọa độ dòng và cột của đỉnh C.

	0	1	2	3	4	5	6	7	8
0	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0
2	0	0	1	1	1	1	1	0	0
3	0	0	1	1	1	1	1	0	0
4	0	0	0	0	0	0	0	0	0

**Hình 4**

Ví dụ, với ma trận  $5 \times 9$  chứa dữ liệu như hình 4 thì hình chữ nhật tối đại, tạm gọi là ABCD, có diện tích là 15 ô và có tọa độ điểm A là (dòng 1, cột 2) và điểm C là (dòng 3, cột 6).

CNMAX. INP	CNMAX. OUT
9	15
100000000	1 2
111111110	3 6
001111100	
001111100	
000000000	

## Thuật toán

Chúng ta sẽ xây dựng một thuật toán giải bài toán tổng quát hơn như sau:

### Sân bay vũ trụ

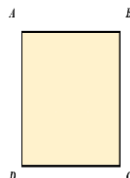
*Người ta cần xác định một vùng hình chữ nhật ABCD có diện tích lớn nhất và bằng phẳng trên hành tinh Vega để xây dựng sân bay vũ trụ. Bạn được cung cấp một mảnh bản đồ hành tinh Vega, nơi cần xác định vị trí xây dựng một sân bay. Mảnh bản đồ có dạng hình chữ nhật gồm nhiều dòng điểm có tọa độ tính từ 0, mỗi dòng có đúng m điểm mã số từ 0 đến m-1. Mỗi điểm được tô một màu thể hiện độ cao của điểm đó. Yêu cầu: xác định hình chữ nhật ABCD chứa nhiều điểm đồng màu nhất.*

Dữ liệu vào: Tập văn bản **VEGA. INP**.

- Dòng đầu tiên: số tự nhiên M,  $3 \leq M \leq 70$ .
- Tiếp đến là các dòng dài bằng nhau thể hiện một xâu gồm M ký tự là các chữ cái a..z viết liền nhau. Mỗi ký tự biểu diễn cho một màu thể hiện độ cao của điểm tương ứng trên mảnh bản đồ hành tinh Vega. Hai ký tự khác nhau thể hiện hai độ cao khác nhau. Hai điểm cùng độ cao được biểu diễn với cùng một ký tự.
- Số dòng của tệp input có thể lên tới 60 nghìn.

### Ví dụ

VEGA.INP	VEGA.OUT
20	80
bccdddeabcvvvvvvvb	1 5
bbbbccccccccbbbbb	8 14
vvvvcccccccccccb	
vvccccccccccbbbbb	
ppppccccccccabbbb	
ppppcccccccczzzzz	
sscccccccccczzzzz	
sssscccccccccczzz	
hhhhhcccccccczzzzz	
uuuuuuuczzzzzzzzzz	



Dữ liệu ra: tệp văn bản VEGA.OUT :

- Dòng đầu tiên: Diện tích của hình chữ nhật tối đại ABCD tìm được.
- Dòng thứ hai: Tọa độ dòng và cột của đỉnh A (ô Tây-Bắc).
- Dòng thứ ba: Tọa độ dòng và cột của đỉnh C (ô Đông-Nam).

Tính tổng quát của bài toán này thể hiện ở điểm sau:

- Tệp không chỉ chứa các ký tự 0/1.

## Thuật toán

Do không thể đọc toàn bộ dữ liệu từ tệp VEGA.INP vào một mảng để xử lý nên chúng ta sẽ đọc mỗi lần một dòng vào biến kiểu xâu ký tự (string)  $y$ . Vì hình chữ nhật ABCD cần tìm chứa cùng một loại ký tự cho nên các dòng của hình sẽ liên thông nhau. Để phát hiện tính liên thông chúng ta cần dùng thêm một biến string  $x$  để lưu dòng đã đọc và xử lý ở bước trước. Tóm lại là ta cần xử lý đồng thời hai dòng:  $x$  là dòng trên và  $y$  là dòng đang xét.

Nếu xét các cột trong hình chữ nhật tối đại cần tìm ta thấy chúng phải chứa cùng một loại ký tự. Ta dùng một mảng  $h$  với ý nghĩa phần tử  $h[i]$  của mảng cho biết tính từ vị trí thứ  $i$  của dòng  $y$  trở lên có bao nhiêu ký tự giống nhau (và giống với ký tự  $y[i]$ ). Ta gọi  $h[i]$  là độ cao của cột  $i$  và mảng  $h$  khi đó sẽ được gọi là độ cao của dòng đang xét.

Ví dụ, mảng tích lũy độ cao của dòng thứ 4 trong ví dụ đã cho sẽ là:

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29
4	y	p	p	p	p	p	c	c	c	c	c	c	c	c	c	c	a	b	b	b	b
	h	1	1	1	1	1	4	4	4	4	4	4	5	4	4	4	1	2	2	4	5

Mảng  $h[]$  lúc đầu được khởi trị toàn 0.

```
for i = 0..m-1 do
  h[i] = 0
end for
```

Sau mỗi lần đọc dòng  $y$  ta chỉnh lại độ cao  $h$  theo tiêu chuẩn sau.

Tại điểm  $i$  đang xét trên dòng  $y$ , nếu  $y[i] = x[i]$  độ cao  $h[i]$  được tăng thêm 1. Ngược lại, nếu  $y[i] \neq x[i]$  ta đặt lại độ cao  $h[i] = 1$ .

```
// Cap nhat do cao
for i = 0..m-1 do
    if y[i] = x[i] then ++h[i] else h[i] = 1 end if
end for
```

Giả sử ta đang xét ký tự thứ  $i = 7$  trên dòng thứ 4 như đã nói ở phần trên. Ta có  $h[7] = h[6] = h[5] = 4$ ;  $h[8] = h[9] = h[10] = 4$ ;  $h[11] = 5$ ;  $h[12] = h[13] = h[14] = 4$ ;  $h[15] = 1, \dots$  Vậy thì, khi ta đi từ  $i$  về hai phía, trái và phải, nếu gặp các ký tự giống ký tự  $y[i]$  còn độ cao thì không nhỏ hơn  $h[i]$  ta sẽ thu được hình chữ nhật lớn nhất chứa ký tự  $y[i]$ .

Với dòng thứ 4 là  $y$  đang xét, ta có:

dòng		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29
3	x	v	v	c	c	c	c	c	c	c	c	c	c	c	c	c	b	b	b	b	b
4	y	p	p	p	p	p	c	c	c	c	c	c	c	c	c	c	a	b	b	b	b
	h	1	1	1	1	1	4	4	4	4	4	4	5	4	4	4	1	2	2	4	5

trong đó  $x$  chứa dữ liệu của dòng 3,  $y$  chứa dữ liệu của dòng 4 trong tệp VEGA.INP.

Từ điểm  $i = 7$  dịch chuyển về bên trái ta thu được điểm  $t = 5$ ; dịch chuyển về bên phải ta thu được điểm  $p = 14$ . Điều kiện để dịch chuyển là:

- ♦  $(y[t] = y[i])$  and  $(h[t] \geq h[i])$  nếu qua trái và
- ♦  $(y[p] = y[i])$  and  $(h[p] \geq h[i])$  nếu qua phải.

Hai thao tác trên được đặt trong hàm tính diện tích của hình chữ nhật ABCD lớn nhất chứa điểm  $y[i]$ . Hàm cho ra ba giá trị:

- ♦  $t$ : điểm trái nhất hay là toạ độ cột của đỉnh D.
- ♦  $p$ : điểm phải nhất hay là toạ độ cột của đỉnh C.
- ♦ Diện tích của hình  $= h[i] * (p - t + 1)$ .

## Chương trình C++

```
/******
   VEGA - Diện tích hình chu nhật
   tôi dài
   *****/
#include <bits/stdc++.h>

using namespace std;

int m; // chiều rộng
const int MN = 71;
string x, y;
int h[MN]; // chiều rộng
int d; // dem dòng
// các thông số max
int dong, trai, phai; // dòng: -----trai----phai-----
```



```

int dtmax;

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

void Print(int x[], int d, int c, const char * msg = "") {
    cout << msg;
    for(int i = d; i <= c; ++i)
        cout << " " << x[i];
}

// ghi file
void Ket() {
    ofstream g("VEGA.OUT");
    g << dtmax << endl;
    if (dtmax == 0) {
        g.close();
        return;
    }
    int dai = phai-tra+1; // chieu dai
    int rong = dtmax / dai; // chieu rong
    // Toa do (dong,cot) của đỉnh a
    int ad = dong - rong + 1;
    int ac = trai;
    // Toa do (dong,cot) của đỉnh c
    int cd = dong;
    int cc = phai;
    g << ad << " " << ac << endl;
    g << cd << " " << cc << endl;
    g.close();
}

// Tính diện tích tại dòng d
// Tìm hai điểm trái nhất t và phải nhất p
int DienTich(int &t, int &p) {
    cout << "\n Tính DT tại dòng " << d << ": " << y;
    // Cập nhật độ cao
    for(int i = 0; i < m; ++i)
        // cập nhật cao độ
        h[i] = (y[i] == x[i]) ? h[i] + 1 : 1;
    Print(h, 0, m-1, "\n h:");
    int s, smax = 0;
    int tt, pp;
    for(int i = 0; i < m; ++i) {
        // Qua trái
        for(tt = i-1; tt >= 0; --tt) {
            if (y[tt] != y[i] || h[tt] < h[i]) {
                ++tt;
                break;
            }
        }
    }
}

```

```

    }
    if (tt < 0) tt = 0;
    //cout << "\n i = " << i << ":" << tt;
    // Qua phải
    for(pp = i+1; pp < m; ++pp) {
        if (y[pp] != y[i] || h[pp] < h[i]) {
            --pp;
            break;
        }
    }
    if (pp >= m) pp = m-1;
    //cout << " " << pp;
    s = h[i]*(pp-tt+1);
    //cout << "\n dt tại " << i << " = " << dt;
    //cout << " tt = " << tt << "   pp = " << pp;
    // cập nhật diện tích max
    if (s > smax) {
        smax = s;
        t = tt; p = pp;
    }
}
cout << "\n Diện tích tại dòng " << d << " = " << smax << endl;
return smax;
}

void Vega() {
    ifstream f("VEGA.INP");
    f >> m; // chiều rộng
    getline(f, x); // xuống dòng mới
    cout << "\n Chiều rộng = " << m << endl;
    x = ""; // Khởi trị x gồm m dấu cách
    dtmax = 0;
    for(int i = 0; i < m; ++i) x += ' ';
    d = -1; // dòng
    int dt, t, p;
    memset(h, 0, sizeof(h)); // cao độ
    while (getline(f, y)) {
        //cout << "\n x: " << x;
        //cout << "\n y: " << y;
        if (y == "") break;
        ++d; // dòng d = y
        dt = DienTich(t, p); // trái t, phải p
        if (dt > dtmax) {
            dtmax = dt;
            dong = d;
            trai = t; phai = p;
        }
    }
    x = y;
}
Ket();
f.close();
}

```

```
main() {
    Vega();
    cout << "\n T h e    E n d";
    return 0;
}
```

### Một vài chú giải

1. Biến đếm dòng  $d$  cho biết ta đang xử lí dòng nào của file.
2. Dòng  $x$  lúc đầu được khởi trị toàn dấu cách là ký tự không có trong văn bản thể hiện tám bản đồ, chẳng hạn dấu cách.
3. Mỗi khi xử lí xong dòng  $y$  ta cần sao chép giá trị của  $y$  cho  $x$  để lưu giữ cho bước tiếp theo. Khi đó  $x$  sẽ trở thành dòng trước.
4. Mỗi khi tìm được một hình chữ nhật, ta so sánh diện tích của hình với diện tích lớn nhất hiện có  $dtmax$  để luôn luôn đảm bảo rằng  $dtmax$  chính là diện tích lớn nhất trong vùng đã khảo sát.
5. Biết chỉ số dòng  $d$  là nơi xuất hiện diện tích  $max$  và hai điểm trái  $t$  và phải  $p$ , ta dễ dàng tính được tọa độ của các đỉnh  $A$  và  $C$  của hình chữ nhật có diện tích  $dmax$

```
chiều dài  $w = p - t + 1$ 
chiều rộng  $r = dtmax / \text{chiều dài}$ 
 $A \text{ dòng} = d - r + 1$ 
 $A \text{ cột} = t$ 
 $C \text{ dòng} = d$ 
 $C \text{ cột} = p$ 
```

Thủ tục Vega được thiết kế theo sơ đồ sau:

```
Vega():
    Mở tệp dữ liệu VEGA.INP
    Đọc giá trị chiều dài mỗi dòng vào biến m
    Khởi trị:
        Biến đếm dòng  $d = 0$ 
        Dòng trước  $x$  toàn dấu cách;
        Mảng chiều cao  $h$  toàn 0;
         $dtmax = 0$  // diện tích max
    Lặp đến khi hết tệp VEGA.INP:
        Đọc dòng  $y$ ;
         $d = d + 1$ 
        Cập nhật độ cao  $h$ ;
        Xử lí mỗi ký tự  $y[i]$  của dòng  $y$ ,  $i = 0..m-1$ :
            Duyệt từ  $i$  qua trái và qua phải để tính diện tích
            của hình chữ nhật lớn nhất chứa phần tử  $y[i]$ ;
            Ghi nhận giá trị ra là  $dt$  và hai chỉ số đầu trái  $t$ 
            và đầu phải  $p$  trên cạnh đáy của hình chữ nhật.
        Nếu  $dt > dtmax$ : chỉnh lại các giá trị
             $dtmax$ ;
            dòng xuất hiện  $dtmax$ 
            chỉ số trái và phải tại dòng xuất hiện  $dtmax$ 
        Sao chép dòng  $y$  sang  $x$ :  $x = y$ 
```

Đóng tệp CNMAX.INP.  
Thông báo kết quả.

### Độ phức tạp tính toán

Giả sử tệp VEGA.INP chứa  $n$  dòng, mỗi dòng chứa  $m$  ký tự. Khi xử lý một dòng, tại mỗi ký tự thứ  $i$  trên dòng đó ta dịch chuyển qua trái và qua phải, tức là ta phải thực hiện tối đa  $m$  phép duyệt. Vậy, với mỗi dòng gồm  $m$  ký tự ta phải thực hiện  $m^2$  phép duyệt. Tổng cộng, với  $n$  dòng ta thực hiện tối đa  $t = n.m^2$  phép duyệt.

### Ứng dụng

Bài toán tìm hình chữ nhật tối đại thường dùng trong lĩnh vực đồ hoạ và xử lý ảnh. Dưới đây liệt kê vài ứng dụng điển hình.

1. Trong khi vẽ bản đồ ta thường phải tìm một hình chữ nhật tối đại trong một vùng, chẳng hạn lãnh thổ của một quốc gia để có thể viết các ký tự vào đó như tên quốc gia, tên châu lục.
2. Trong hình học phân hình (fractal) ta thường phải tìm một số hình vuông hoặc chữ nhật tối đại thoả mãn một số tiêu chuẩn cho trước để làm mẫu.

Trong bài này, để trình bày vấn đề được đơn giản chúng ta đã thay mỗi điểm bằng một ký tự.

### Bài tập làm thêm

**Bài T1.** Với mỗi ký tự  $c$  cho trước hãy tìm trong tệp VEGA.INP một hình chữ nhật tối đại chứa toàn ký tự  $c$ .

**Bài T2.** Với cặp giá trị  $(d, c)$  cho trước hãy tìm hình chữ nhật tối đại chứa cùng một loại ký tự và đồng thời chứa điểm nằm trên dòng  $d$ , cột  $c$  của tệp VEGA.INP.

### Bài 8.4. Ma phương

*Ma phương là những bảng số hình vuông trong đó mỗi dòng, mỗi cột và mỗi đường chéo đều cùng thoả một số tính chất nào đó. Các nhà thiên văn cổ Trung Hoa cho rằng mỗi tinh tú trên bầu trời đều ứng với một ma phương. Nhiều nhà toán học cổ Ai Cập, Ấn Độ và sau này các nhà toán học phương Tây đã phát hiện những tính chất khá lí thú của các loại ma phương. Trong bài này ta giới hạn khái niệm ma phương theo nghĩa sau.*

Ma phương bậc  $n$  là một bảng số hình vuông, gồm  $n \times n$  ô chứa các số từ 1 đến  $n^2$  sao cho tổng các số trên mỗi dòng, trên mỗi cột và trên mỗi đường chéo đều bằng nhau. Tổng này được gọi là *đặc số* của ma phương.

8	1	6
---	---	---

16	2	3	13
----	---	---	----

3	5	7
4	9	2

(a)

5	11	10	8
9	7	6	12
4	14	15	1

(b)

(a) – ma phương bậc 3, đặc số  $S_3 = 15$

(b) – ma phương bậc 4, đặc số  $S_4 = 34$

Yêu cầu: Với mỗi trị  $n$  xây dựng ma phương bậc  $n$ .

### Thuật toán

Ta dễ dàng tính được đặc số của ma phương bậc  $n$  qua nhận xét: tổng các số trong một cột/dòng chính là tổng của toàn bộ các số nằm trong bảng chia cho số cột/dòng:

$$S_n = (1 + 2 + \dots + n^2)/n = n(n^2 + 1)/2.$$

Theo các ví dụ trên ta có:

Đặc số của ma phương bậc 3:  $S_3 = 3(9 + 1)/2 = 15$ .

Đặc số của ma phương bậc 4:  $S_4 = 4(16 + 1)/2 = 34$ .

Như vậy, trong ma phương bậc 3, tổng của các số nằm trên cùng hàng, cùng cột hoặc cùng đường chéo đều là 15. Trong ma phương bậc 4, tổng này là 34.

Tính chất trên không thay đổi nếu ta điền lần lượt các số hạng của một cấp số cộng vào ma phương.

Ngoài bậc  $n = 2$ , với mọi số tự nhiên  $n \geq 1$  đều tồn tại ma phương với nhiều cách bố trí khác nhau. Chúng ta sẽ tìm hiểu hai thuật toán để cài đặt.

Với mỗi  $n$  cho trước ta xét tính chẵn lẻ của nó. Nếu  $n$  lẻ ta gọi thủ tục MPL (ma phương bậc lẻ), ngược lại ta gọi thủ tục MPC (ma phương bậc chẵn).

```
MP(n):
  if n = 2 then return end if // vô nghiệm
  if Odd(n) then MPL(n) else MPC(n) end if
end MP
```

Hàm Odd( $n$ ) kiểm tra tính lẻ/chẵn của một số tự nhiên  $n$ .

**MPL ( $n$ ) : Ma phương bậc lẻ**

Ta dùng một mảng hai chiều  $a$  để chứa ma phương cần xây dựng theo các bước sau đây. Để cho tiện ta đặt tên cho ma phương cần xây dựng là hình vuông ABCD.

Bước 1. Khởi trị: Điền các số 0 vào bảng  $a$ .

Bước 2. Xác định ô xuất phát: Đó là ô giữa của dòng đầu tiên, dòng 0:

$$a[0][n/2]$$

Bước 3. Điền số: Với mỗi  $k = 2..n^2$  ta thực hiện các thao tác sau:

3.1. Xác định vị trí  $i, j$  mới để điền số  $k$

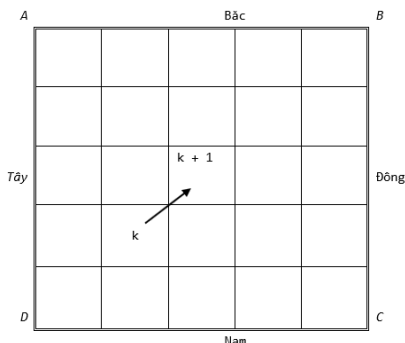
3.2. Điền trị:  $a[i][j] = k$

Vị trí  $i, j$  mới được xác định theo nguyên tắc Đông-Bắc với ý nghĩa là sau khi đã điền giá trị  $k$  vào ô  $(i, j)$  thì giá trị  $k + 1$  sẽ được viết vào ô nằm ở vị trí Đông-Bắc so với ô chứa  $k$ , tức là ô  $(i+1, j+1)$ . Như vậy,

nếu  $a[i][j] = k$

thì vị trí ô chứa  $k + 1$  sẽ là  $i = i+1, j = j+1$ .

Có thể sẽ xảy ra các tình huống sau đây:



Điền ô theo hướng Đông – Bắc  
cho ma phương bậc lẻ.

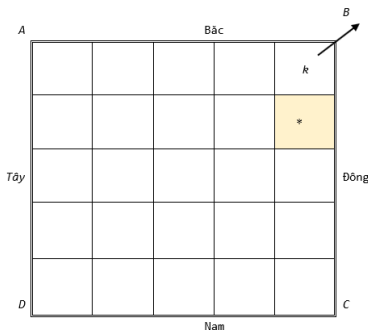
### Tình huống Góc B

Góc B:  $i < 0$  và  $j = n$ : vị trí mới rơi vào góc ngoài ma trận: ô B

Số  $k$  vừa viết nằm tại ô  $(0, n-1)$  do đó số  $k+1$  sẽ rơi vào ô  $(-1, n)$ . Ta gọi tình huống này là tình huống *Góc B* và xử lý như sau: viết số  $k + 1$  vào ô sát dưới ô chứa  $k$ , tức là ta chỉnh lại  $i, j$  như sau:

Đền dưới:  $i = i+2; j = j-1$ ;

Ta gọi phương thức xử lý này là *đền dưới*: số  $k+1$  sẽ được viết vào dưới ô vừa viết số  $k$ .

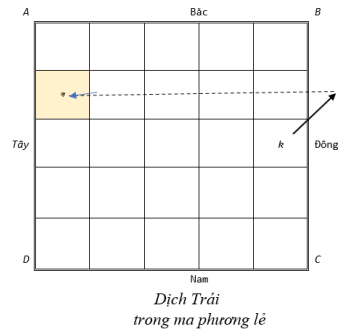


Đền dưới  
trong ma phương lẻ

### Tình huống Lề Phải

Lề Phải:  $j = n$

Số  $k$  đã viết nằm sát lề phải (cột  $n-1$ ), do đó số  $k+1$  sẽ bị rơi ra ngoài lề phải ma trận. Ta gọi tình huống này là tình huống *Lề Phải* và xử lý theo phương thức *dịch trái*: đặt lại  $j = 0$  như sau:



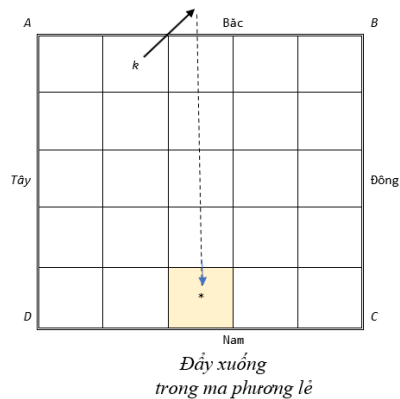
Dịch trái:  $j = 0$

Tình huống Lề Trên

Lề Trên:  $i < 0$

Số  $k$  đã viết nằm sát lề trên (dòng 0), do đó số  $k+1$  sẽ bị rơi ra ngoài lề trên, tức là rơi vào dòng  $-1$  của ma trận. Ta gọi tình huống này là tình huống *Lề Trên* và xử lý theo phương thức *Đẩy Xuống*: đặt lại  $i = n-1$  như sau:

Đẩy xuống:  $i = n-1$

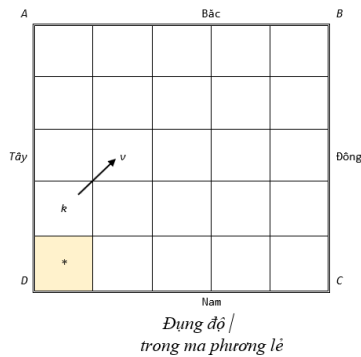


Tình huống đụng độ

Cuối cùng có thể ô mới  $(i, j)$  nơi bạn định viết số  $k+1$  đã chứa sẵn số đã viết trước đó. Ta gọi tình huống này là *đụng độ* và xử lý theo phương thức *đền dưới* giống như tình huống Góc B.

Đụng độ:  $a[i][j] > 0$

Đền dưới:  $i = i+2; j = j - 1;$



Trong các chương trình dưới đây chúng ta cài đặt thêm các thủ tục sau:

- ✧ Print: hiển thị ma phuong dưới dạng căn chỉnh các số chiều dài khác nhau cho dễ đọc
- ✧ Test: kiểm tra tổng các số trên dòng, cột, đường chéo bằng đặc số của ma phuong ?

## Chương trình C++

```

/*****
Ma phuong Le
*****/
#include <bits/stdc++.h>

using namespace std;

const int MN = 100;
const string DIGIT = "0123456789";
int a[MN][MN];
int n, nn, maxlen;
const char BL = 32;

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// So chu so cua x
int Len(int x) {
    if (x < 10) return 1;
    int c = 0;
    while(x != 0) {
        ++c;
        x /= 10;
    }
    return c;
}

```



```

}

string Str(int x) { // x > 0
    int d = maxlen - Len(x);
    string s = "";
    while (x != 0) {
        s = DIGIT[x % 10] + s;
        x /= 10;
    }
    for(int i = 0; i <= d; ++i)
        s = BL+s;
    return s;
}

// Hien thi ma phuong
void Print(const char * msg = "") {
    cout << msg;
    for(int i = 0; i < n; ++i) {
        cout << endl;
        for(int j = 0; j < n; ++j) {
            cout << Str(a[i][j]);
        }
    }
}

void Test() {
    int s = (1+nn)*n/2; // Dac so
    int c1 = 0, c2 = 0; // tong tren hai duong cheo
    int d, c; // tong tren dong, cot
    int i, j;
    for(i = 0; i < n; ++i) {
        c1 += a[i][i]; c2 += a[i][n-1-i];
        d = c = 0;
        for(j = 0; j < n; ++j) {
            d += a[i][j]; c += a[j][i];
        }
        if (d != s) {
            cout << "\n LOI dong " << i << "!";
            return;
        }
        if (c != s) {
            cout << "\n LOI cot " << j << "!";
            return;
        }
    }
    if (c1 != s) {
        cout << "\n LOI cheo 1!";
        return;
    }
    if (c2 != s) {
        cout << "\n LOI cheo 2!";
        return;
    }
}

```

```

    cout << "\n CHUAN.";
}

void MPL() {
    cout << "\n Ma phuong bac le: " << n;
    memset(a, 0, sizeof(a));
    int i = 0, j = n / 2;
    a[i][j] = 1;
    for(int k = 2; k <= nn; ++k) {
        // tim vi tri (i,j) dat k
        --i; ++j; // Huong Dong Bac
        if(i < 0 && j == n) { // den duoi
            i += 2;
            --j;
        }
        else if (i < 0) i = n-1; // dich xuong
        else if (j == n) j = 0; // dich trai
        if (a[i][j] > 0) { // den duoi
            i += 2;
            --j;
        }
        a[i][j] = k;
    }
    Print();
    Test();
}

void MPC() {
    cout << "\n Ma phuong bac chan: " << n;
}

void MP(int inp) {
    cout << "\n Bac: " << inp;
    if (inp < 1 || inp >= MN) {
        cout << ". Can chon bac trong khoang 1.." << MN-1;
        return;
    }
    if (inp == 2) {
        cout << "\n Vo nghiem";
        return;
    }
    n = inp;
    nn = n*n;
    maxlen = Len(nn);
    if (n % 2 == 1) MPL();
    else MPC();
}

main() {
    for (int n = -1; n < 15; n += 2) {
        MP(n);
        Go();
    }
}

```

```

    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Bac: -1. Can chon bac trong khoang 1..99
?
Bac: 1
Ma phuong bac le: 1
1
CHUAN.
?
Bac: 3
Ma phuong bac le: 3
8 1 6
3 5 7
4 9 2
CHUAN.
?
Bac: 5
Ma phuong bac le: 5
17 24 1 8 15
23 5 7 14 16
4 6 13 20 22
10 12 19 21 3
11 18 25 2 9
CHUAN.
?
Bac: 7
Ma phuong bac le: 7
30 39 48 1 10 19 28
38 47 7 9 18 27 29
46 6 8 17 26 35 37
5 14 16 25 34 36 45
13 15 24 33 42 44 4
21 23 32 41 43 3 12
22 31 40 49 2 11 20
CHUAN.
?
Bac: 9
Ma phuong bac le: 9
47 58 69 80 1 12 23 34 45
57 68 79 9 11 22 33 44 46
67 78 8 10 21 32 43 54 56
77 7 18 20 31 42 53 55 66
6 17 19 30 41 52 63 65 76
16 27 29 40 51 62 64 75 5
26 28 39 50 61 72 74 4 15
36 38 49 60 71 73 3 14 25
37 48 59 70 81 2 13 24 35

```

CHUAN.

?

Bac: 11

Ma phuong bac le: 11

68	81	94	107	120	1	14	27	40	53	66
80	93	106	119	11	13	26	39	52	65	67
92	105	118	10	12	25	38	51	64	77	79
104	117	9	22	24	37	50	63	76	78	91
116	8	21	23	36	49	62	75	88	90	103
7	20	33	35	48	61	74	87	89	102	115
19	32	34	47	60	73	86	99	101	114	6
31	44	46	59	72	85	98	100	113	5	18
43	45	58	71	84	97	110	112	4	17	30
55	57	70	83	96	109	111	3	16	29	42
56	69	82	95	108	121	2	15	28	41	54

CHUAN.

?

Bac: 13

Ma phuong bac le: 13

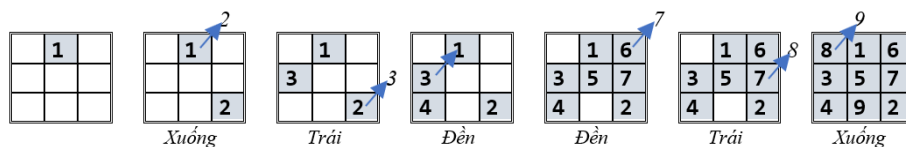
93	108	123	138	153	168	1	16	31	46	61	76	91
107	122	137	152	167	13	15	30	45	60	75	90	92
121	136	151	166	12	14	29	44	59	74	89	104	106
135	150	165	11	26	28	43	58	73	88	103	105	120
149	164	10	25	27	42	57	72	87	102	117	119	134
163	9	24	39	41	56	71	86	101	116	118	133	148
8	23	38	40	55	70	85	100	115	130	132	147	162
22	37	52	54	69	84	99	114	129	131	146	161	7
36	51	53	68	83	98	113	128	143	145	160	6	21
50	65	67	82	97	112	127	142	144	159	5	20	35
64	66	81	96	111	126	141	156	158	4	19	34	49
78	80	95	110	125	140	155	157	3	18	33	48	63
79	94	109	124	139	154	169	2	17	32	47	62	77

CHUAN.

?

T h e E n d

Ta minh họa thủ tục điền trị cho ma phuong le bậc 3.



Các từ viết tắt trong hình:

*Xuống*: Đẩy xuống; *Trái*: Dịch trái; *Đền*: Đền dưới**MPC (n) : Ma phuong bậc chẵn**

Bước 1. Khởi trị: Điền các số từ 1 đến  $n^2$  vào bảng theo trật tự từ trên xuống dưới, từ trái qua phải. Ví dụ, với  $n = 4$ , bảng a được khởi trị như sau:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Khởi trị cho ma phương bậc 4

Bước 2. Tạo chuỗi mẫu: Ta tạo chuỗi mẫu  $s$  phục vụ cho việc đổi chỗ các số trong  $a$ . Chuỗi mẫu  $s$  có chiều dài  $k = n \text{ div } 2$  và bao gồm các ký tự 'T', 'D', 'N' và 'B' với ý nghĩa sau:

- 'T' - thực hiện phép đối xứng tâm: Đổi chỗ các cặp phần tử:  
 $a[i][j] \leftrightarrow a[n-i-i][n-j-1]$   
 $a[n-i-1][j] \leftrightarrow a[i][n-j-1]$
- 'D' - thực hiện phép đối xứng theo trục dọc: Đổi chỗ cặp phần tử:  
 $a[i][j] \leftrightarrow a[i][n-j-1]$
- 'N' - thực hiện phép đối xứng theo trục ngang: Đổi chỗ cặp phần tử:  
 $a[i][j] \leftrightarrow a[n-i-1][j]$
- 'B' – bỏ qua, không làm gì.

Chuỗi mẫu  $s$  được tạo như sau:

- o Khởi trị chuỗi rỗng  $s = ""$
- o Tính  $k = n \text{ div } 2$
- o Nạp  $(k \text{ div } 2)$  ký tự 'T' cho  $s$ .

```
for i = 1..(k div 2) do
    s = s + 'T'
end for
```

- o Nếu  $k$  lẻ nạp thêm hai ký tự 'DN' cho  $s$ :

```
if Odd(k) then s = s + "DN" end if
```

- o Điền thêm các ký tự 'B' cho đủ  $k$  ký tự trong  $s$ .

```
for i = length(s)+1..k do
    s = s + 'B'
end for
```

Các thí dụ tạo chuỗi mẫu  $s$ :

- Với  $n = 4$  ta có  $k = n \text{ div } 2 = 2$  (chẵn),  $k \text{ div } 2 = 1$ , do đó  $s = "TB"$
- Với  $n = 6$  ta có  $k = n \text{ div } 2 = 3$  (lẻ),  $k \text{ div } 2 = 1$ , do đó  $s = "TDN"$
- Với  $n = 18$  ta có  $k = n \text{ div } 2 = 9$  (lẻ),  $k \text{ div } 2 = 4$ , do đó  $s = "TTTTDNBBBB"$

Bước 3. Thực hiện

MPC(n):

```
Khởi trị
Tạo chuỗi mẫu s
// Lặp n div 2 lần
for i = 0..(n div 2 - 1) do
    // cập nhật dòng i a[i] theo chuỗi mẫu s
    for j = 0..(n div 2 - 1) do
        case s[j] = 'T': Đối xứng tâm a[i][j]
```

```

        case s[j] = 'D': Đối xứng dọc a[i][j]
        case s[j] = 'N': Đối xứng ngang a[i][j]
        quay xâu mẫu s
    end for j
end for i
end MPC

```

Xâu mẫu s được quay phải 1 vị trí: ký tự cuối được chuyển về đầu:

```

Rotate(s):
    k = len(s)
    s = s[k-1] + s[0..k-2]
end Rotate

```

Ví dụ

```

Rotate("TDN") = "NTD"
Rotate("NTD") = "DNT"

```

Dưới đây là minh họa ma phương bậc chẵn  $n = 6$

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Khởi trị  
s = "TDN"

36	5	33	4	2	31
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
6	32	3	34	35	1

Cập nhật dòng 0  
s = "TDN"  
Đối xứng Tâm: 1 ↔ 36  
6 ↔ 31  
Đối xứng Dọc: 2 ↔ 5  
Đối xứng Ngang: 3 ↔ 33  
Quay xâu mẫu: s = "NTD"

36	5	33	4	2	31
25	29	10	9	26	12
13	14	15	16	17	18
19	20	21	22	23	24
7	11	27	28	8	30
6	32	3	34	35	1

Cập nhật dòng 1  
s = "NTD"  
Đối xứng Ngang: 7 ↔ 25  
Đối xứng Tâm: 8 ↔ 29  
11 ↔ 26  
Đối xứng Dọc: 9 ↔ 10  
Quay xâu mẫu: s = "DNT"

36	5	33	4	2	31
25	29	10	9	26	12
18	20	22	21	17	13
19	14	16	15	23	24
7	11	27	28	8	30
6	32	3	34	35	1

Cập nhật dòng 3  
 $s = \text{"DNT"}$   
 Đối xứng Dọc:  $18 \leftrightarrow 13$   
 Đối xứng Ngang:  $14 \leftrightarrow 20$   
 Đối xứng Tâm:  $15 \leftrightarrow 22$   
 $12 \leftrightarrow 25$   
 Quay xâu mẫu:  $s = \text{"TDN"}$

## Chương trình C++

```

/*****
                        Ma phuong
*****/
#include <bits/stdc++.h>

using namespace std;

const int MN = 100;
const string DIGIT = "0123456789";
int a[MN][MN];
int n, nn, maxlen;
const char BL = 32;

void Go() {
    cout << "\n ? ";
    fflush(stdin);
    if (cin.get() == '.') exit(0);
}

// So chu so cua x
int Len(int x) {
    if (x < 10) return 1;
    int c = 0;
    while(x != 0) {
        ++c;
        x /= 10;
    }
    return c;
}

string Str(int x) { // x > 0
    int d = maxlen - Len(x);
    string s = "";
    while (x != 0) {
        s = DIGIT[x % 10] + s;
        x /= 10;
    }
}

```

```

    }
    for(int i = 0; i <= d; ++i)
        s = BL+s;
    return s;
}

// Hien thi ma phuong
void Print(const char * msg = "") {
    cout << msg;
    for(int i = 0; i < n; ++i) {
        cout << endl;
        for(int j = 0; j < n; ++j) {
            cout << Str(a[i][j]);
        }
    }
}

void Test() {
    int s = (1+nn)*n/2; // Dac so
    int c1 = 0, c2 = 0; // tong tren hai duong cheo
    int d, c; // tong tren dong, cot
    int i, j;
    for(i = 0; i < n; ++i) {
        c1 += a[i][i]; c2 += a[i][n-1-i];
        d = c = 0;
        for(j = 0; j < n; ++j) {
            d += a[i][j]; c += a[j][i];
        }
        if (d != s) {
            cout << "\n LOI dong " << i << "!";
            return;
        }
        if (c != s) {
            cout << "\n LOI cot " << j << "!";
            return;
        }
    }
    if (c1 != s) {
        cout << "\n LOI cheo 1!";
        return;
    }
    if (c2 != s) {
        cout << "\n LOI cheo 2!";
        return;
    }
    cout << "\n CHUAN.";
}

void MPL() {
    cout << "\n Ma phuong bac le: " << n;
    memset(a, 0, sizeof(a));
    int i = 0, j = n / 2;
    a[i][j] = 1;

```



```

for(int k = 2; k <= nn; ++k) {
    // tìm vị trí (i,j) đặt k
    --i; ++j; // Hướng Đông Bắc
    if(i < 0 && j == n) { // đến dưới
        i += 2;
        --j;
    }
    else if (i < 0) i = n-1; // dịch xuống
    else if (j == n) j = 0; // dịch trái
    if (a[i][j] > 0) { // đến dưới
        i += 2;
        --j;
    }
    a[i][j] = k;
}
Print();
Test();
}

// số tại ô (i,j)
// ma phương chẵn
int Val(int i, int j) {
    return i*n + j + 1;
}

// Đối xứng tam
void Tam(int i, int j) {
    a[i][j] = Val(n-i-1, n-j-1);
    a[n-i-1][n-j-1] = Val(i, j);
    a[i][n-j-1] = Val(n-i-1, j);
    a[n-i-1][j] = Val(i, n-j-1);
}

// Đối xứng qua trục dọc
void Doc(int i, int j) {
    a[i][j] = Val(i, n-j-1);
    a[i][n-j-1] = Val(i, j);
}

// Đối xứng qua trục ngang
void Ngang(int i, int j) {
    a[i][j] = Val(n-i-1, j);
    a[n-i-1][j] = Val(i, j);
}

void MPC() {
    cout << "\n Ma phuong bac chan: " << n;
    // Khởi trị
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            a[i][j] = Val(i, j);
    // Print();
    // Tạo xau màu

```

```

    int k = n/2; // xau mau gom k ki tu TDNB
    int k2 = k / 2; // K2 ki tu T
    string s = "";
    for(int i= 0; i < k2; ++i)
        s += "T";
    if (k % 2 == 1) s += "DN"; // k le, them DN
    // Dien B cho du k ki tu
    for(int i = s.length(); i < k; ++i)
        s += "B";
    // cout << "\n xau mau: " << s;
    // sua k dong theo xau mau
    for(int i = 0; i < k; ++i) {
        for(int j = 0; j < k; ++j) {
            switch(s[j]) {
                case 'T': Tam(i,j); break;
                case 'D': Doc(i,j); break;
                case 'N': Ngang(i,j); break;
            } // switch
        }
        // s = Rotate(s);
        s = s[k-1] + s.substr(0,k-1);
        // cout << "\n Rotate: " << s;
    }
    Print();
    Test();
}

void MP(int inp) {
    cout << "\n Bac: " << inp;
    if (inp < 1 || inp >= MN) {
        cout << ". Can chon bac trong khoang 1.." << MN-1;
        return;
    }
    if (inp == 2) {
        cout << "\n Vo nghiem";
        return;
    }
    n = inp;
    nn = n*n;
    maxlen = Len(nn);
    if (n % 2 == 1) MPL();
    else MPC();
}

main() {
    for (int n = -1; n <= 12; ++n) {
        MP(n);
        Go();
    }
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```
Bac: -1. Can chon bac trong khoang 1..99
?
Bac: 0. Can chon bac trong khoang 1..99
?
Bac: 1
Ma phuong bac le: 1
1
CHUAN.
?
Bac: 2
Vo nghiem
?
Bac: 3
Ma phuong bac le: 3
8 1 6
3 5 7
4 9 2
CHUAN.
?
Bac: 4
Ma phuong bac chan: 4
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
CHUAN.
?
Bac: 5
Ma phuong bac le: 5
17 24 1 8 15
23 5 7 14 16
4 6 13 20 22
10 12 19 21 3
11 18 25 2 9
CHUAN.
?
Bac: 6
Ma phuong bac chan: 6
36 5 33 4 2 31
25 29 10 9 26 12
18 20 22 21 17 13
19 14 16 15 23 24
7 11 27 28 8 30
6 32 3 34 35 1
CHUAN.
?
Bac: 7
Ma phuong bac le: 7
30 39 48 1 10 19 28
38 47 7 9 18 27 29
```

```

46 6 8 17 26 35 37
5 14 16 25 34 36 45
13 15 24 33 42 44 4
21 23 32 41 43 3 12
22 31 40 49 2 11 20

```

CHUAN.

?

Bac: 8

Ma phuong bac chan: 8

```

64 63 3 4 5 6 58 57
9 55 54 12 13 51 50 16
17 18 46 45 44 43 23 24
40 26 27 37 36 30 31 33
32 34 35 29 28 38 39 25
41 42 22 21 20 19 47 48
49 15 14 52 53 11 10 56
8 7 59 60 61 62 2 1

```

CHUAN.

?

Bac: 9

Ma phuong bac le: 9

```

47 58 69 80 1 12 23 34 45
57 68 79 9 11 22 33 44 46
67 78 8 10 21 32 43 54 56
77 7 18 20 31 42 53 55 66
6 17 19 30 41 52 63 65 76
16 27 29 40 51 62 64 75 5
26 28 39 50 61 72 74 4 15
36 38 49 60 71 73 3 14 25
37 48 59 70 81 2 13 24 35

```

CHUAN.

?

Bac: 10

Ma phuong bac chan: 10

```

100 99 8 94 5 6 7 3 92 91
11 89 88 17 85 16 14 83 82 20
71 22 78 77 26 25 74 73 29 30
40 62 33 67 66 65 64 38 39 31
60 49 53 44 56 55 47 48 42 51
50 52 43 54 46 45 57 58 59 41
61 32 63 37 36 35 34 68 69 70
21 72 28 27 75 76 24 23 79 80
81 19 18 84 15 86 87 13 12 90
10 9 93 4 95 96 97 98 2 1

```

CHUAN.

?

Bac: 11

Ma phuong bac le: 11

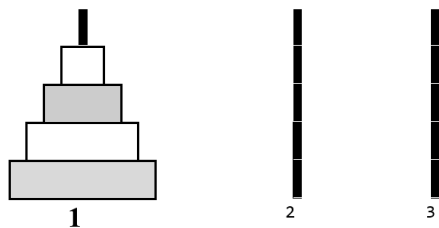
```

68 81 94 107 120 1 14 27 40 53 66
80 93 106 119 11 13 26 39 52 65 67
92 105 118 10 12 25 38 51 64 77 79
104 117 9 22 24 37 50 63 76 78 91
116 8 21 23 36 49 62 75 88 90 103

```

```
7 20 33 35 48 61 74 87 89 102 115
19 32 34 47 60 73 86 99 101 114 6
31 44 46 59 72 85 98 100 113 5 18
43 45 58 71 84 97 110 112 4 17 30
55 57 70 83 96 109 111 3 16 29 42
56 69 82 95 108 121 2 15 28 41 54
CHUAN.
?
Bac: 12
Ma phuong bac chan: 12
144 143 142 4 5 6 7 8 9 135 134 133
13 131 130 129 17 18 19 20 124 123 122 24
25 26 118 117 116 30 31 113 112 111 35 36
37 38 39 105 104 103 102 101 100 46 47 48
96 50 51 52 92 91 90 89 57 58 59 85
84 83 63 64 65 79 78 68 69 70 74 73
72 71 75 76 77 67 66 80 81 82 62 61
60 86 87 88 56 55 54 53 93 94 95 49
97 98 99 45 44 43 42 41 40 106 107 108
109 110 34 33 32 114 115 29 28 27 119 120
121 23 22 21 125 126 127 128 16 15 14 132
12 11 10 136 137 138 139 140 141 3 2 1
CHUAN.
?
T h e E n d
```

Bài 8.5. Tháp Hà Nội cổ



Hình 8.1. Bài toán tháp Hà Nội

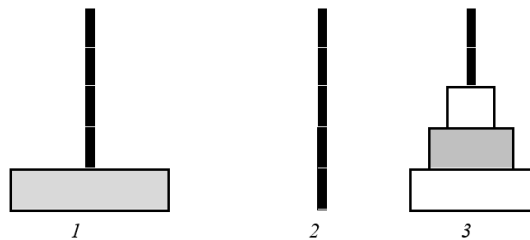
Có ba cọc cắm tại ba vị trí là 1, 2 và 3 như hình 8.1. Trên một cọc, gọi là cọc a có một chồng gồm n đĩa bằng gỗ hình tròn to nhỏ khác nhau được xuyên lỗ ở giữa tựa như những đồng xu và đặt chồng lên nhau để tạo ra một toà tháp. Người chơi phải chuyển được toà tháp sang cọc  $b \neq a$  theo các quy tắc sau:

- (1) Người chơi được sử dụng cọc còn lại để đặt tạm các tầng tháp.
  - (2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang một trong hai cọc còn lại.
  - (3) Không bao giờ được đặt tầng tháp lớn lên trên tầng tháp nhỏ.
- Hãy tìm cách giải bài toán trên với số lần chuyển ít nhất.

## Thuật toán

Chắc chắn là bạn đã biết cách giải bài toán nổi tiếng trên. Tuy nhiên để có thể giải dễ dàng các biến thể của bài toán tháp Hà Nội chúng ta thử tìm hiểu một cách lập luận sau.

Giả sử ta quan sát một người chơi giỏi, tức là người chuyển được  $n$  tầng tháp từ cọc 1 sang cọc 2 với số lần chuyển tối thiểu. Ta dùng một chiếc máy ảnh chụp từng kết quả trung gian sau mỗi bước chuyển của người chơi này. Tổng số bức ảnh, trừ tám ảnh ban đầu, chính là số bước chuyển các tầng. Trong số các bức ảnh chắc chắn phải có một bức như hình 8.2.



Hình 8.2. Một ảnh phải có

Tại sao vậy? Tại vì chừng nào chưa dỡ được  $n - 1$  tầng tháp ở phía trên của vị trí 1 để chuyển tạm sang vị trí 3 thì anh ta không thể chuyển được tầng tháp cuối, tức là tầng lớn nhất sang vị trí 2.

Gọi  $H_n(n, a, b)$  là thủ tục chuyển  $n$  tầng tháp từ vị trí  $a$  sang vị trí  $b \neq a$ , ta thấy:

- Nếu  $n = 0$ : không phải làm gì;
- Nếu  $n > 0$  ta phải thực hiện ba bước sau:

✎ Thoạt tiên chuyển  $n - 1$  tầng tháp từ vị trí  $a$  sang vị trí  $c = 6 - a - b$ :

$$H_n(n-1, a, 6-a-b)$$

✎ Sau đó chuyển tầng lớn nhất từ vị trí  $a$  sang vị trí  $b$ :

$$a \rightarrow b$$

✎ Cuối cùng chuyển lại  $n - 1$  tầng tháp từ  $c$  sang  $b$ :

$$H_n(n-1, 6-a-b, b)$$

Đề ý rằng, do ta mã hoá các cọc là 1, 2 và 3 cho nên biết hai trong ba vị trí đó, là  $x$ ,  $y$  chẳng hạn, ta dễ dàng tính được vị trí còn lại  $z$  theo hệ thức

$$z = 6 - x - y$$

Thủ tục chuyển tháp  $n$  tầng từ cọc  $a$  sang cọc  $b$  khi đó sẽ như sau:

```

HN(n, a, b):
  if (n = 0) return end if
  c = 6-a-b;
  HN(n-1,a,c);
  a → b
  HN(n-1,c, b)
end HN

```

## Chương trình C++

```

/*****
Thap Ha Noi Co
*****/
#include <bits/stdc++.h>

using namespace std;
int step;

void Move(int a, int b) {
  ++step;
  cout << "\n " << step << ". " << a << " -> " << b;
}

void HN(int n, int a, int b) {
  if (n == 0) return;
  int c = 6-a-b;
  HN(n-1,a,c);
  Move(a,b);
  HN(n-1,c, b);
}

void HaNoi(int n, int a, int b) {
  cout << "\n Thap Ha Noi " << n << " tang: "
    << "[" << a << "]" -> "[" << b << "];"
  step = 0;
  HN(n, a, b);
}

main() {
  int n = 3;
  int a = 1;
  int b = 2;
  HaNoi(n, a, b);
  cout << "\n T h e   E n d";
  return 0;
}

```

## Output

```
Thap Ha Noi 3 tang: [1] -> [2]
```

```

1. 1 -> 2
2. 1 -> 3
3. 2 -> 3
4. 1 -> 2
5. 3 -> 1
6. 3 -> 2
7. 1 -> 2
T h e   E n d

```

Chương trình sử dụng biến đếm *step* nhằm đếm số bước chuyển.

## Bài 8.6. Tháp Hà Nội xuôi

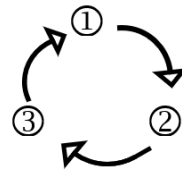
Nội dung giống như bài toán tháp Hà Nội cổ chỉ sửa lại quy tắc (2) như sau:

(2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang cọc sát nó theo chiều kim đồng hồ.

Điều kiện này quy định ba phép chuyển 1 tầng tháp giữa các cọc như sau:

① → ②, hoặc ② → ③, hoặc ③ → ①.

Hãy tìm cách giải bài toán với số lần chuyển ít nhất.



### Thuật toán

Suy nghĩ một chút bạn sẽ thấy cái lợi của nguyên tắc "*Bức ảnh buộc phải có*". Đặt tên các tầng tháp theo thứ tự từ nhỏ đến lớn là  $1..n$ . Ta mô tả mỗi tấm ảnh như một bộ ba  $(a:[A], b:[B], c:[C])$  trong đó A, B và C là các tầng tháp đặt tại mỗi vị trí tương ứng. Gọi  $a$  là vị trí xuất phát,  $b$  là vị trí đích, bài toán trên có thể được phát biểu như sau:

Gia thiết:  $(a:[1..n], b:[], c:[])$

...

Kết luận:  $(a:[], b:[1..n], c:[])$

Với ý nghĩa là cho biết bức ảnh ban đầu và cuối cùng. Hãy liệt kê ít nhất các bức ảnh cần có ở giữa ba dấu chấm (...) sao cho bộ ảnh giải thích được quá trình chuyển tháp theo các điều kiện cho trước.

Mỗi bức ảnh được gọi là một hình trạng. Ngoài hai hình trạng đầu tiên và cuối cùng, một trong những hình trạng buộc phải có sẽ là  $(a:[n], b:[], c:[1..n-1])$ . Tiếp đó là hình trạng  $(a:[], b:[n], c:[1..n-1])$  thu được sau lệnh chuyển  $a \rightarrow b$

Gọi  $H_{nx}(n, a, b)$  là thủ tục giải bài toán tháp Hà Nội xuôi, chuyển tháp  $n$  tầng từ vị trí  $a$  sang vị trí  $b$ . Ta dễ dàng tính được vị trí dự phòng  $c = 6 - a - b$ . xét hai trường hợp.

a) Trường hợp vị trí  $b$  đứng sát vị trí  $a$  theo chiều kim đồng hồ:

$(a) \rightarrow (b) \rightarrow (c)$ ,  $(c) \rightarrow (a) \rightarrow (b)$ , hoặc  $(b) \rightarrow (c) \rightarrow (a)$

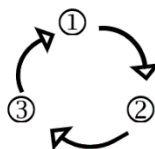
Theo trường hợp này ta có các cặp  $(a) \rightarrow (b)$  sau đây:

$(1) \rightarrow (2): a = (1), b = (2)$

$(2) \rightarrow (3): a = (2), b = (3)$



$(3) \rightarrow (1): a = (3), b = (1)$   
 $b = (a \bmod 3) + 1$



Đặc tả cho điều kiện của trường hợp này là  $b = (a \bmod 3) + 1$

Với ý nghĩa là, nếu biết mã số vị trí a thì vị trí b sát sau a theo chiều kim đồng hồ sẽ được tính theo công thức trên.

Nếu vị trí các cọc được bố trí như sau

①

②

③

thì giữa a và b có ba tình huống, cụ thể là

*Tình huống 1:*  $(1) \rightarrow (2): a = (1), b = (2)$

*Tình huống 2:*  $(2) \rightarrow (3): a = (2), b = (3)$

*Tình huống 3:*  $(3) \rightarrow (1): a = (3), b = (1)$

Dựa vào các hình trạng buộc phải có ta có thể mô tả việc chuyển tháp trong trường hợp này giống như trong bài toán tháp Hà Nội cổ, cụ thể là:

Nếu b kề a, tức là  $(a) \rightarrow (b)$  thì ta có thể chuyển thẳng tầng dưới cùng từ a qua b. Do đó ta phải

- ♦ Dỡ n-1 tầng tháp từ a qua c
- ♦ Chuyển thẳng tầng dưới cùng từ a qua b.
- ♦ Chuyển ngược lại n-1 tầng tháp từ c qua a

```
HNX(n, a, b): // b kề a: a → b
c = 6 - a - b
```

```
HNX(n-1, a, c);
a → b
HNX(n-1, c, b);
```

Còn lại là trường hợp b không kề với a theo chiều kim đồng hồ. Khi đó c sẽ kề với a và b kề với c, do đó ta phải thực hiện các bước sau:

- ♦ Dỡ n-1 tầng tháp từ a qua b
- ♦ Chuyển tầng dưới cùng từ a qua c.
- ♦ Chuyển ngược lại n-1 tầng tháp từ b qua a
- ♦ Chuyển tầng hiện có từ c qua b.
- ♦ Chuyển lại n-1 tầng tháp từ a qua b

```
HNX(n-1, a, b); // b không kề a: a → c
HNX(n-1, a, b);
a → c
HNX(n-1, b, a);
c → b
HNX(n-1, a, b);
```

Tổng hợp lại, ta thu được thuật toán Hà Nội xuôi như sau

```
HNX(n, a, b):
if n = 0 return
```

```

c = 6-a-b
if b = (a % 3) + 1 then
    // a → b
    HNX(n-1, a, c);
    a → b
    HNX(n-1, c, b);
end if
else // a → c → b
    HNX(n-1, a, b)
    a → c
    HNX(n-1, b, a);
    c → b
    HNX(n-1, a, b);
end else
end HNX

```

## Chương trình C++

```

/*****
    Thap Ha Noi Xuoi
*****/

#include <bits/stdc++.h>

using namespace std;
int step;

void Move(int a, int b) {
    ++step;
    cout << "\n " << step << ". " << a << " -> " << b;
}

void HNX(int n, int a, int b) {
    if(n == 0) return;
    int c = 6-a-b;
    if (b == (a % 3) + 1) { // a b c
        HNX(n-1, a, c);
        Move(a, b);
        HNX(n-1, c, b);
    }
    else { // a c b
        HNX(n-1, a, b);
        Move(a, c);
        HNX(n-1, b, a);
        Move(c, b);
        HNX(n-1, a, b);
    }
}

void HaNoiXuoi(int n, int a, int b) {

```

```

    cout << "\n Ha Noi Xuoi " << n << " tang: "
        << "[" << a << "]" -> "[" << b << "];
    step = 0;
    HNX(n, a, b);
    cout << "\n Total " << step << " step(s.)";
}

main() {
    int n = 3;
    int a = 1;
    int b = 2;
    HaNoiXuoi(n, a, b);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Ha Noi Xuoi 3 tang: [1] -> [2]
1. 1 -> 2
2. 2 -> 3
3. 1 -> 2
4. 3 -> 1
5. 2 -> 3
6. 1 -> 2
7. 2 -> 3
8. 1 -> 2
9. 3 -> 1
10. 1 -> 2
11. 3 -> 1
12. 2 -> 3
13. 1 -> 2
14. 3 -> 1
15. 1 -> 2
Total 15 step(s.)
T h e   E n d

```

Bạn thử so sánh HaNoiXuoi(3,1,2) và so sánh với kết quả sau đây?

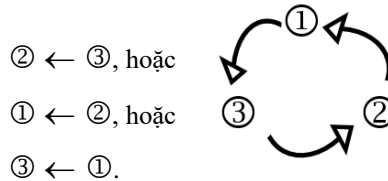
1. 1 -> 2	9. 3 -> 1
2. 2 -> 3	10. 1 -> 2
3. 1 -> 2	11. 3 -> 1
4. 3 -> 1	12. 2 -> 3
5. 2 -> 3	13. 1 -> 2
6. 1 -> 2	14. 3 -> 1
7. 2 -> 3	15. 1 -> 2
8. 1 -> 2	Total: 15 step(s)

## Bài 8.7. Tháp Hà Nội ngược

Nội dung giống như bài toán tháp Hà Nội cổ chỉ sửa lại quy tắc (2) như sau:

(2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang cọc sát nó về hướng ngược chiều kim đồng hồ.

Điều kiện này quy định ba phép chuyển 1 tầng tháp như sau:



Hãy tìm cách giải bài toán với số lần chuyển ít nhất.

### Thuật toán

Bài này tương tự như bài Hà Nội xuôi. Ta chỉ cần xác định điều kiện kê giữa hai cọc tháp và lưu ý đến chiều chuyển các tầng tháp ngược chiều quay của kim đồng hồ.

Giả sử ta cần giải bài HNN( $n, a, b$ ) chuyển  $n > 0$  tầng tháp từ cọc  $a$  sang cọc  $b$  ngược chiều quay của kim đồng hồ. Trước cũng chia ra hai trường hợp là  $b$  kê  $a$  và không kê  $a$ . Điều kiện  $b$  kê  $a$  khi đó sẽ ngược với bài Hà Nội xuôi, cụ thể là

Điều kiện  $b$  kê  $a$  trong bài toán *chuyển ngược* sẽ là điều kiện  $a$  kê  $b$  trong bài toán *chuyển xuôi*, do đó

$b$  kê  $a$  khi và chỉ khi  $a = b \bmod 3 + 1$

```

HNN( $n, a, b$ ):
  if  $n = 0$  then return end if
   $c = 6 - a - b$ ;
  if  $a = (b \% 3) + 1$  then //  $b \leftarrow a$ 
    HNN( $n-1, a, c$ )
     $b \leftarrow a$ 
    HNN( $n-1, c, b$ );
  end if
  else //  $b \leftarrow c \leftarrow a$ 
    HNN( $n-1, a, b$ );
     $c \leftarrow a$ 
    HNN( $n-1, b, a$ )
     $b \leftarrow c$ 
    HNN( $n-1, a, b$ );
  end else
end HNN
  
```

## Chương trình C++

```

/*****
                Thap Ha Noi Nguoc
*****/
#include<bits/stdc++.h>

using namespace std;
int step;

void Move(int a, int b) {
    ++step;
    cout << "\n " << step << ". " << a << " -> " << b;
}

void HNN(int n, int a, int b) {
    if(n == 0) return;
    int c = 6-a-b;
    if (a == (b % 3) + 1) { // b a c
        HNN(n-1, a, c);
        Move(a, b);
        HNN(n-1, c, b);
    }
    else { // a c b
        HNN(n-1, a, b);
        Move(a, c);
        HNN(n-1, b, a);
        Move(c, b);
        HNN(n-1, a, b);
    }
}

void HaNoiNguoc(int n, int a, int b) {
    cout << "\n Ha Noi Nguoc " << n << " tang: "
        << "[" << a << "]" -> "[" << b << "]";
    step = 0;
    HNN(n, a, b);
    cout << "\n Total " << step << " step(s.)";
}

main() {
    int n = 3;
    int a = 1;
    int b = 2;
    HaNoiNguoc(n, a, b);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

Ha Noi Nguoc 3 tang: [1] -> [2]

```

1. 1 -> 3
2. 3 -> 2
3. 1 -> 3
4. 2 -> 1
5. 3 -> 2
6. 1 -> 3
7. 3 -> 2
8. 1 -> 3
9. 2 -> 1
10. 1 -> 3
11. 2 -> 1
12. 3 -> 2
13. 2 -> 1
14. 3 -> 2
15. 1 -> 3
16. 3 -> 2
17. 1 -> 3
18. 2 -> 1
19. 3 -> 2
20. 1 -> 3
21. 3 -> 2
Total 21 step(s.)
T h e   E n d

```

## Kết quả

1. 1 -> 3	12. 3 -> 2
2. 3 -> 2	13. 2 -> 1
3. 1 -> 3	14. 3 -> 2
4. 2 -> 1	15. 1 -> 3
5. 3 -> 2	16. 3 -> 2
6. 1 -> 3	17. 1 -> 3
7. 3 -> 2	18. 2 -> 1
8. 1 -> 3	19. 3 -> 2
9. 2 -> 1	20. 1 -> 3
10. 1 -> 3	21. 3 -> 2
11. 2 -> 1	Total: 21 step(s)

## Nhận xét

Mới xem ta có cảm tưởng rằng lời gọi  $Hnn(3,1,2)$  và  $Hnx(3,1,2)$  để chuyển tháp 3 tầng từ cọc 1 sang cọc 2 phải cho cùng một số bước chuyển các tầng là 15. Tuy nhiên, lời gọi  $Hnn(3,1,2)$  cho ta 21 bước chuyển các tầng, trong khi lời gọi  $Hnx(3,1,2)$  chỉ cần 15 bước chuyển các tầng.

Suy nghĩ một chút bạn sẽ giải thích được nghịch lý này.

Hãy thử gọi Hà Nội ngược để chuyển tháp 3 tầng từ cọc 3 sang cọc 2:

$Hnn(3,3,2)$

Ta sẽ thấy chỉ cần 15 bước!!!

Lại gọi Hà Nội xuôi để chuyển tháp 3 tầng từ cọc 1 sang cọc 3:

$$H_{nx}(3, 1, 3)$$

Ta lại thấy 21 bước.

Như vậy,  $H_{nx}$  và  $H_{nn}$  là *đối xứng lặc*. Nếu hai cọc, nguồn và đích kề nhau thì số lần chuyển tháp 3 tầng sẽ là 15. Ngược lại, khi hai cọc đó không kề nhau thì số lần chuyển tháp 3 tầng sẽ là 21. Hai cọc 1 và 2 là kề nhau đối với tháp Hà Nội xuôi, nhưng không kề nhau đối với tháp Hà Nội ngược. Tương tự, hai cọc 3 và 2 là kề nhau đối với tháp Hà Nội ngược, nhưng không kề nhau đối với tháp Hà Nội xuôi.

Ta nhận xét rằng: nếu lấy hai số  $a, b$  khác nhau bất kì trong ba số 1, 2 và 3 thì giữa  $a$  và  $b$  chỉ xảy ra một trong hai trường hợp loại trừ nhau sau đây:

$$b = (a \bmod 3) + 1, \text{ hoặc } a = (b \bmod 3) + 1$$

Do đó, quan hệ kề nhau trong hai bài toán Tháp Hà Nội xuôi và ngược là phủ định đối với nhau.

	H <sub>NX</sub>	H <sub>NN</sub>
$b = (a \bmod 3) + 1$	$(a) \rightarrow (b)$	$(a) ! \rightarrow (b)$
$a = (b \bmod 3) + 1$	$(b) \leftarrow ! (a)$	$(b) \leftarrow (a)$

*Quan hệ kề nhau trong hai bài toán  
tháp Hà Nội xuôi và ngược*

## Bài 8.8. Tháp Hà Nội thẳng

Nội dung giống như bài toán tháp Hà Nội cổ chỉ sửa lại quy tắc (2) như sau:

(2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang cọc kề nó, không được vòng từ 3 sang 1 hay 1 sang 3.

Điều kiện này quy định bốn phép chuyển 1 tầng tháp như sau:

$$\textcircled{1} \rightarrow \textcircled{2}, \textcircled{2} \rightarrow \textcircled{1}, \textcircled{2} \rightarrow \textcircled{3}, \textcircled{3} \rightarrow \textcircled{2}$$

hoặc, theo cách biểu diễn khác:

$$\textcircled{1} \leftrightarrow \textcircled{2} \leftrightarrow \textcircled{3}$$

tức là chỉ được chuyển qua lại giữa hai cọc kề nhau. Giả thiết là các cọc được sắp thành hàng như sau:

$$\textcircled{1} \longleftrightarrow \textcircled{2} \longleftrightarrow \textcircled{3}$$

Hãy tìm cách giải bài toán với số lần chuyển ít nhất.

## Thuật toán

Giống như đã phân tích trong các bài toán Hà Nội trước, ta có:

Hình trạng xuất phát:  $(a:[1..n], b:[ ], c:[ ])$

...

Hình trạng kết thúc:  $(a:[ ], b:[1..n], c:[ ])$

Hình trạng buộc phải có:  $(a:[n], b:[ ], c:[1..n-1])$

Ta phân biệt hai trường hợp:

- ✎ Hai cọc  $a$  và  $b$  đứng kề nhau trên đường thẳng.
- ✎ Hai cọc  $a$  và  $b$  không kề nhau mà cách nhau qua  $c$ .

Trường hợp thứ nhất Nếu vị trí các cọc được bố trí như sau

①      ②      ③

thì giữa  $a$  và  $b$  có bốn tình huống, cụ thể là:

Tình huống	①	②	③
1	a	b	
2	b	a	
3		a	b
4		b	a

*Tháp Hà Nội thẳng*  
Đặc tả  $a$  và  $b$  kề nhau  $\text{abs}(a-b) = 1$

Dựa vào tình huống  $b$  kề  $a$  ta có ngay thủ tục xử lý HNT như sau:

```
HNT(n, a, b):
  if n = 0 then return end if
  c = 6-a-b;
  if abs(a-b) = 1 then // a→b, b→a
    HNT(n-1, a, c)
    a → b
    HNT(n-1, c, b)
  end if
  else // a→c→b
    HNT(n-1, a, b)
    a→c
    HNT(n-1, b, a);
    c→b
    HNT(n-1, a, b);
  end else
end HNT
```

## Chương trình C++



```

/*****
Thap Ha Noi Thang
*****/
#include<iostream>
#include <bits/stdc++.h>

using namespace std;
int step;

void Move(int a, int b) {
    ++step;
    cout << "\n " << step << ". " << a << " -> " << b;
}

void HNT(int n, int a, int b) {
    if(n == 0) return;
    int c = 6-a-b;
    if (abs(a-b) == 1) { // b a c
        HNT(n-1, a, c);
        Move(a, b);
        HNT(n-1, c, b);
    }
    else { // a c b
        HNT(n-1, a, b);
        Move(a, c);
        HNT(n-1, b, a);
        Move(c, b);
        HNT(n-1, a, b);
    }
}

void HaNoiThang(int n, int a, int b) {
    cout << "\n Ha Noi Thang " << n << " tang: "
        << "[" << a << "]" -> "[" << b << "];"
    step = 0;
    HNT(n, a, b);
    cout << "\n Total " << step << " step(s.)";
}

main() {
    int n = 3;
    int a = 1;
    int b = 2;
    HaNoiThang(n, a, b);
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Ha Noi Thang 3 tang: [1] -> [2]
1. 1 -> 2

```

```

2. 2 -> 3
3. 1 -> 2
4. 3 -> 2
5. 2 -> 1
6. 2 -> 3
7. 1 -> 2
8. 2 -> 3
9. 1 -> 2
10. 3 -> 2
11. 2 -> 1
12. 3 -> 2
13. 1 -> 2
Total 13 step(s.)
T h e   E n d

```

### Bài 8.9. Tháp Hà Nội sắc màu (Hà Nội Cầu vồng)

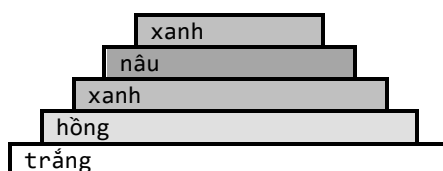
Người ta sơn mỗi tầng tháp một màu và quy định luật chuyển cho mỗi loại tầng theo màu như mô tả trong bảng sau.

Kí hiệu	Màu	Ý nghĩa chuyển tầng	Quy tắc
x	xanh	xuôi chiều kim đồng hồ	① → ② → ③ → ①
n	nâu	ngược chiều kim đồng hồ	① ← ② ← ③ ← ①
t	trắng	thăng	① ↔ ② ↔ ③
h	hồng	tự do (Hà Nội cổ)	① ↔ ② ↔ ③ ↔ ①

Ngoài ra, dĩ nhiên vẫn phải theo quy định là tầng to không được đặt lên trên tầng nhỏ.

Hãy tìm cách giải bài toán với số lần chuyển ít nhất.

Ví dụ, với các tầng tháp được tô màu từ trên (tầng nhỏ nhất) xuống dưới (tầng lớn nhất) là:



Hà Nội sắc màu 5 tầng xnxht

và cần chuyển tháp từ cọc 1 sang cọc 2 thì phải thực hiện tối thiểu 31 lần chuyển các tầng như sau:

```

1. 1 -> 2
2. 1 -> 3
3. 2 -> 3
4. 1 -> 2
5. 3 -> 1
6. 3 -> 2
7. 1 -> 2
17. 3 -> 1
18. 3 -> 2
19. 1 -> 2
20. 3 -> 1
21. 2 -> 3
22. 2 -> 1
23. 3 -> 1

```

8. 1 -> 3	24. 3 -> 2
9. 2 -> 3	25. 1 -> 2
10. 2 -> 1	26. 1 -> 3
11. 3 -> 1	27. 2 -> 3
12. 2 -> 3	28. 1 -> 2
13. 1 -> 2	29. 3 -> 1
14. 1 -> 3	30. 3 -> 2
15. 2 -> 3	31. 1 -> 2
16. 1 -> 2	Total: 31 step(s)

## Thuật toán

Điều lí thú là thủ tục Hà Nội sắc màu là khá tổng quát vì ta có thể dùng nó để gọi cho các bài toán về tháp Hà Nội đã xét. Bảng dưới đây cho biết cách sử dụng thủ tục Hà Nội sắc màu HNM cho các trường hợp riêng.

<i>Muốn gọi</i>	<i>Thì gọi</i>	<i>Chú thích</i>
HN(n, a, b)	HNM("h..h", a, b)	s chứa n ký tự 'h'
HNX(n, a, b)	HNM("x..x", a, b)	s chứa n ký tự 'x'
HNN(n, a, b)	HNM("n..n", a, b)	s chứa n ký tự 'n'
HNT(n, a, b)	HNM("t..t", a, b)	s chứa n ký tự 't'

Ta quy ước dữ liệu ban đầu được mô tả trong biến tổng thể string s. Trong ví dụ trên, s sẽ được gán trị là

```
s = "xnxht";
```

Khi đó có thể khai báo thủ tục tháp Hà Nội sắc màu như sau:

```
HaNoiSacMau(inp, a, b)
```

trong đó string inp là các màu của mỗi tầng tháp, a và b là hai cọc khác nhau cho trước và nhận các giá trị 1, 2 hoặc 3.

Hàm trên sẽ gọi hàm

```
HNM(n, a, b)
```

trong đó n là số tầng tháp. biến inp sẽ được truyền cho biến tổng thể s và n chính là chiều dài của string s. HNM sẽ xử lí tháp n tầng, tính từ trên xuống, tức là từ tầng nhỏ nhất có mã số 1 đến tầng đáy, tầng lớn nhất mang mã số n như sau:

Tầng (i)	Màu (Thap[i])
1	'x'
2	'n'
3	'x'
4	'h'
5	't'

*Gọi thủ tục cho bài  
Hà Nội Sắc màu  
runHnm('xnxht')*

Cách mã số này ngược với quy tắc gọi các tầng trong đời thường. Người ta thường mã số các tầng của toà nhà từ dưới lên là 1, 2,... Dĩ nhiên chúng ta sẽ thêm một phần tử 0 vào bên trái string s để làm phần tử đệm, vì các chỉ số của string được tính từ 0.

Sau đó chúng ta dựa vào quy tắc xác định quan hệ kề giữa hai cọc a và b theo màu của mỗi tầng tháp để quyết định các bước chuyển tầng tháp.

```
Ke(n, a, b):
    case s[n] = 'x': return  b = (a % 3)+1 // chuyển xuôi
    case s[n] = 'n': return  a = (b % 3)+1 // chuyển ngược
    case s[n] = 't': return  abs(a-b) = 1 // chuyển thẳng
    else return true; // chuyển tự do
end Ke
```

Các chương trình dưới đây sẽ lần lượt gọi các hàm:

```
HaNoiSacMau("xnxht", a, b);
HaNoiSacMau("hhh", a, b); // HaNoi
HaNoiSacMau("xxx", a, b); // HaNoiXuoi
HaNoiSacMau("nnn", a, b); // HaNoiNguoc
HaNoiSacMau("ttt", a, b); // HaNoiThang
```

với a = 1, b = 2 để đối sánh.

## Chương trình C++

```
/******
   Thap Ha Noi Sac Mau
   *****/
#include <bits/stdc++.h>

using namespace std;
int step;
string s;

void Move(int a, int b) {
    ++step;
    cout << "\n " << step << ". " << a << " -> " << b;
}

bool Ke(int n, int a, int b) {
    switch(s[n]) {
        case 'x': return  b == (a % 3)+1;
        case 'n': return  a == (b % 3)+1;
        case 't': return  abs(a-b) == 1;
        default: return true;
    } // switch
}

void HNSM(int n, int a, int b) {
    if(n == 0) return;
```

```

int c = 6-a-b;
if (Ke(n, a, b)) { // b a c
    HNSM(n-1, a, c);
    Move(a, b);
    HNSM(n-1, c, b);
}
else { // a c b
    HNSM(n-1, a, b);
    Move(a, c);
    HNSM(n-1, b, a);
    Move(c, b);
    HNSM(n-1, a, b);
}
}

void HaNoiSacMau(string inp, int a, int b) {
    cout << "\n Ha Noi Sac Mau " << inp << ": "
        << "[" << a << "]" -> "[" << b << "]" ;
    step = 0;
    s = "0" + inp;
    HNSM(s.length()-1, a, b);
    cout << "\n Total " << step << " step(s.)";
}

main() {
    int n = 3;
    int a = 1;
    int b = 2;
    HaNoiSacMau("xnxht", a, b); Go();
    HaNoiSacMau("hhh", a, b); Go(); // HaNoi
    HaNoiSacMau("xxx", a, b); Go(); // HaNoiXuai
    HaNoiSacMau("nnn", a, b); Go(); // HaNoiNguoc
    HaNoiSacMau("ttt", a, b); // HaNoiThang
    cout << "\n T h e   E n d";
    return 0;
}

```

## Output

```

Ha Noi Sac Mau xnxht: [1] -> [2]
1. 1 -> 2
2. 1 -> 3
3. 2 -> 3
4. 1 -> 2
5. 3 -> 1
6. 3 -> 2
7. 1 -> 2
8. 1 -> 3
9. 2 -> 3
10. 2 -> 1
11. 3 -> 1
12. 2 -> 3

```

```
13. 1 -> 2
14. 1 -> 3
15. 2 -> 3
16. 1 -> 2
17. 3 -> 1
18. 3 -> 2
19. 1 -> 2
20. 3 -> 1
21. 2 -> 3
22. 2 -> 1
23. 3 -> 1
24. 3 -> 2
25. 1 -> 2
26. 1 -> 3
27. 2 -> 3
28. 1 -> 2
29. 3 -> 1
30. 3 -> 2
31. 1 -> 2
Total 31 step(s.)
?
```

Ha Noi Sac Mau hhh: [1] -> [2]

```
1. 1 -> 2
2. 1 -> 3
3. 2 -> 3
4. 1 -> 2
5. 3 -> 1
6. 3 -> 2
7. 1 -> 2
Total 7 step(s.)
?
```

Ha Noi Sac Mau xxx: [1] -> [2]

```
1. 1 -> 2
2. 2 -> 3
3. 1 -> 2
4. 3 -> 1
5. 2 -> 3
6. 1 -> 2
7. 2 -> 3
8. 1 -> 2
9. 3 -> 1
10. 1 -> 2
11. 3 -> 1
12. 2 -> 3
13. 1 -> 2
14. 3 -> 1
15. 1 -> 2
Total 15 step(s.)
?
```

Ha Noi Sac Mau nnn: [1] -> [2]

```
1. 1 -> 3
2. 3 -> 2
```

```

3. 1 -> 3
4. 2 -> 1
5. 3 -> 2
6. 1 -> 3
7. 3 -> 2
8. 1 -> 3
9. 2 -> 1
10. 1 -> 3
11. 2 -> 1
12. 3 -> 2
13. 2 -> 1
14. 3 -> 2
15. 1 -> 3
16. 3 -> 2
17. 1 -> 3
18. 2 -> 1
19. 3 -> 2
20. 1 -> 3
21. 3 -> 2
Total 21 step(s.)
?
Ha Noi Sac Mau ttt: [1] -> [2]
1. 1 -> 2
2. 2 -> 3
3. 1 -> 2
4. 3 -> 2
5. 2 -> 1
6. 2 -> 3
7. 1 -> 2
8. 2 -> 3
9. 1 -> 2
10. 3 -> 2
11. 2 -> 1
12. 3 -> 2
13. 1 -> 2
Total 13 step(s.)
T h e   E n d

```

## Đọc thêm

Một số bài toán về tháp Hà Nội đã được đưa vào các kì thi Olympic Tin học tại một vài quốc gia. Chúng ta thử tìm hiểu cội nguồn của các bài toán thuộc loại này.

Năm 1883, trên một tờ báo ở Paris có đăng bài mô tả một trò chơi toán học của giáo sư Claus với tên là Tháp Hà Nội. Nội dung trò chơi được mọi người say mê làm thử chính là bài toán Tháp Hà Nội cổ.

Thời đó ở thủ đô Paris dân chúng đổ xô nhau mua đồ chơi này và suốt ngày ngồi chuyên tháp. Trong lịch sử về các trò chơi thông minh đã từng có những cơn sốt như vậy. Tính trung bình mỗi thế kỉ có một vài cơn sốt trò chơi. Thế kỉ thứ XX có cơn sốt Rubic, thế kỉ XIX là các trò chơi 15 và tháp Hà Nội. Bài toán này nổi tiếng đến mức trở

thành kinh điển trong các giáo trình về thuật giải đệ quy và được trình bày trong các thông báo chính thức của các phiên bản chuẩn của các ngữ trình như ALGOL-60, ALGOL-68, Pascal, Delphi, C, C++, Ada, Java, Python... khi muốn nhấn mạnh về khả năng đệ quy của các ngôn ngữ đó.

Theo nhà nghiên cứu Henri De Parville công bố vào năm 1884 thì tác giả của trò chơi

tháp Hà Nội có tên thật là nhà toán học Édouard Lucas, người có nhiều đóng góp trong lĩnh vực số luận. Mỗi khi viết về đề tài giải trí thì ông đổi tên là Claus. Bạn có để ý rằng Claus là một hoán vị các chữ cái của từ Lucas.



Édouard Lucas  
1842-1891

De Parville còn kể rằng bài toán tháp Hà Nội bắt nguồn từ một tích truyền kì ở Ấn Độ. Một nhóm cao tăng Ấn Độ giáo được giao trong trách chuyển dần 64 đĩa vàng giữa ba cọc kim cương theo các điều kiện đã nói ở bài toán Tháp Hà Nội cổ. Khi nào hoàn tất công việc, tức là khi chuyển xong toà tháp vàng 64 tầng từ vị trí ban đầu sang vị trí kết thúc thì cũng là thời điểm tận thế. Sự việc này có xảy ra hay không ta sẽ xét ở bài tập dưới mục này.

Lời giải được công bố đầu tiên cho bài toán tháp Hà Nội là của Allardice và Frase, năm 1884.

Năm 1994 David G. Poole ở Đại học Trent, Canada đã viết một bài khảo cứu cho tờ Mathematics Magazine số tháng 12 nhan đề "*Về các tháp và các tam giác của giáo sư Claus*" cùng với một phụ đề "*Pascal biết Hà Nội*". Poole đã liệt kê 65 công trình khảo cứu bài toán tháp Hà Nội đăng trên các tạp chí toán-tin trong khoảng mười năm. Tác giả cũng chỉ ra sự liên quan giữa các công thức tính số lần chuyển các tầng tháp và một phương pháp quen biết dùng để tính các hệ số của dạng khai triển nhị thức Newton  $(a + b)^n$ . Phương pháp này được gọi là Tam giác Pascal, mang tên nhà toán học kiêm vật lí học Pháp Blaise Pascal (1623-1662), người đã chế tạo chiếc máy tính quay tay đầu tiên trên thế giới.

Một số nhà nghiên cứu trong và ngoài nước có bàn luận về địa danh Hà Nội. Theo người viết, vấn đề này vẫn còn ngò. Hầu hết các bài viết xoay quanh đề tài chuyển tháp nói trên đều dùng thuật ngữ bài toán tháp Hà Nội. Khi giới thiệu về bài toán Hà Nội nhiều tháp Dudeney đặt tên là bài toán đồ của Reve (The Reve's Puzzle). Tuy nhiên, nhiều nhà nghiên cứu cho rằng tốt hơn cả là nên đặt tên và phân loại theo tên nguyên thủy của bài toán, nghĩa là Tháp Hà Nội.

Ngoài các dạng Tháp Hà Nội đã liệt kê ở phần trên một số tác giả còn đề xuất những dạng khá kì lạ, chẳng hạn như bài toán sau đây.

## Hà Nội nhiều tháp

Trong trò chơi này người ta làm thêm những cọc, chẳng hạn thay vì ba ta dùng bốn cọc và cũng có thể bố trí tháp tại nhiều cọc. Ý kiến này do H.E. Dudeney, một tác giả hàng đầu về toán học giải trí người Anh đưa ra vào năm 1908. Đã có nhiều bài đăng lời giải cho bài toán này, có những bài mới xuất hiện gần đây vào những năm 1988 và 1989. Dù vậy chưa ai chứng minh được rõ ràng số lần chuyển của bài giải là tối thiểu như đã làm với các dạng tháp Hà Nội khác.



## Bài tập

Bạn hãy thử lập công thức tính số lần chuyển các tầng tối thiểu cho các bài toán sau:

Tháp Hà Nội,  
Tháp Hà Nội Xuôi,  
Tháp Hà Nội Ngược và  
Tháp Hà Nội Thăng.

## Lời cảm ơn

Các tư liệu trên và một số tư liệu khác trong bài được trích dẫn từ các bài viết của giáo sư viện sĩ Nguyễn Xuân Vinh, Khoa Kỹ thuật không gian, Đại học Michigan, cộng tác viên NASA, Hoa Kỳ. Tác giả xin chân thành cảm ơn giáo sư đã cho phép trích dẫn và chỉ giáo về các phương pháp truyền thụ tri thức khoa học cho giới trẻ.

NXH

Sửa ngày 20/01/2022