MỤC LỤC

CHUYÊN ĐỀ
SẮP XẾP – TÌM KIẾM

1. Sắp xếp

Sắp xếp đóng vai trò rất quan trọng trong cuộc sống nói chung và trong tin học nói riêng, thử hình dung xem, một cuốn từ điển nếu các từ không được sắp xếp theo thứ tự, sẽ khó khăn như thế nào trong việc tra cứu các từ.

Theo D.Knuth thì 40% thời gian tính toán của máy tính là dành cho việc sắp xếp. Không phải ngẫu nhiên thuật toán sắp xếp nhanh (Quick Sort) được bình chọn là một trong 10 thuật toán tiêu biểu của thế kỉ 20.

1.1. Khái niệm

Sắp xếp là quá trình bố trí lại vị trí các đối tượng của một danh sách theo một trật tự nhất định.

Do đặc điểm dữ liệu (kiểu số hay phi số, kích thước bé hay lớn, lưu trữ ở bộ nhớ trong hay bộ nhớ ngoài, truy cập tuần tự hay ngẫu nhiên...) mà người ta có các thuật toán sắp xếp khác nhau. Trong chuyên đề này, chúng ta chỉ quan tâm đến các thuật toán sắp xếp trong trường hợp dữ liệu được lưu trữ ở bộ nhớ trong (nghĩa là toàn bộ dữ liệu cần sắp xếp phải được đưa vào bộ nhớ chính của máy tính).

1.2. Phát biểu bài toán

Giả sử các đối tượng cần sắp xếp được biểu diễn bởi bản ghi gồm một số trường. Một trong các trường đó được gọi là khoá sắp xếp. Kiểu của khoá là kiểu có thứ tự (chẳng hạn, kiểu số nguyên, kiểu ký tự, kiểu số thực,…)

```
Const
     Nmax = 1000;
Type
     Obj = record
          Key: KeyType;
          [các trường khác];
     End;
     TArray = Array[1..Nmax] of Obj;
Var
     A:TArray;
     n:Longint;
```

Bài toán sắp xếp được phát biểu như sau: Cho mảng *A* các đối tượng, cần sắp xếp lại các thành phần (phần tử) của mảng *A* để nhận được mảng *A* mới với các thành phần có các giá trị khoá tăng dần: $A[1].key \le A[2].key \le ... \le A[n].key$

1.3. Sắp xếp chọn (Selection Sort)

Ý tưởng thuật toán: Ý tưởng cơ bản của thuật toán là chọn phần tử có khóa nhỏ nhất trong dãy đổi chỗ cho phần tử đầu tiên của dãy rồi loại bỏ phần tử đầu tiên ra khỏi dãy (thực chất là nó vẫn nằm trong dãy nhưng nó không còn được xét đến). Tiếp tục thực hiện lặp lại thao tác trên đối với dãy còn lại cho đến khi dãy chỉ còn một phần tử.

Cụ thể:

- Lượt 1: Tìm phần tử có khóa nhỏ nhất trong dãy từ phần tử thứ 1 đến phần tử thứ *n* để đổi chỗ cho phần tử thứ 1.

- Lượt 2: Tìm phần tử có khóa nhỏ nhất trong dãy từ phần tử thứ 2 đến phần tử thứ *n* để đổi chỗ cho phần tử thứ 2.

- Lượt *i*: Tìm phần tử có khóa nhỏ nhất trong dãy từ phần tử thứ *i* đến phần tử thứ *n* để đổi chỗ cho phần tử thứ i.

Xong lượt thứ *n*-1 thì dãy được sắp xếp xong.

```
Procedure SelectionSort;
Var  T:Obj;
     i,j:longint;
Begin
     For i:=1 to n-1 do
          For j:=i+1 to n do
               If A[i].key > A[j].key then
               Begin
                    T:=A[i];
                    A[i]:=A[j];
                    A[j]:=T;
               End;
End;
```

1.4. Sắp xếp chèn (Insert Sort)

Ý tưởng thuật toán: Chia dãy ban đầu thành hai dãy con, dãy đầu gồm các phần tử từ 1 đến *k* (*k*=1..*n*-1) là dãy đã được sắp xếp và dãy gồm các phần tử từ *k*+1 đến *n* là dãy

chưa sắp xếp. Với mỗi giá trị của *k*, ta lấy *X*=*A*[*k*+1] ra khỏi dãy rồi dịch chuyển các phần tử trong dãy đã sắp xếp (các phần tử từ 1 đến *k*) có khóa lớn hơn *X*.key sang phải một vị trí sau đó chèn *X* vào vị trí rỗng trong dãy đã sắp xếp.

Cụ thể:

- Lượt 1: Lấy phần tử thứ 2 (*X*:=A[2]) ra khỏi dãy. Dịch chuyển các phần tử trong đoạn từ 1 đến 1 có khóa lớn hơn khóa của *X* sang phải một vị trí. Chèn *X* vào vị trí trống.

- Lượt 2: Lấy phần tử thứ 3 (X:=A[3]) ra khỏi dãy. Dịch chuyển các phần tử trong đoạn từ 1 đến 2 có khóa lớn hơn khóa của *X* sang phải một vị trí. Chèn *X* vào vị trí trống.

- Lượt *i*: Lấy phần tử thứ *i*+1 (X:=A[*i*+1]) ra khỏi dãy. Dịch chuyển các phần tử trong đoạn từ 1 đến *i* có khóa lớn hơn khóa của *X* sang phải một vị trí. Chèn *X* vào vị trí trống.

Sau *n*-1 lượt, dãy sắp xếp xong.

```
Procedure InsertSort;
Var  X:Obj;
     k,j:longint;
Begin
     For k:=1 to n-1 do
     Begin
       j:=k;
       X:=A[k+1];
       while (j >= 1) and (A[j].key > X.key)do
        Begin
          A[j+1]:=A[j] ;
          j:=j-1;
       End;
       A[j+1]:=X;
     End;
End;
```

1.5. Sắp xếp nổi bọt (Bubble Sort)

Ý tưởng cơ bản của thuật toán nổi bọt là tìm và đổi chỗ các cặp phần tử kề nhau sai thứ tự (phần tử đứng trước có khoá lớn hơn khoá của phần tử đứng sau) cho đến khi không tồn tại cặp nào sai thứ tự (dãy được sắp xếp).

Cụ thể:

- Lượt 1: ta xét từ cuối dãy, nếu gặp 2 phần tử kề nhau mà sai thứ tự thì đổi chỗ chúng cho nhau. Sau lượt 1, phần tử có khoá nhỏ thứ nhất được đưa về vị trí 1.

- Lượt 2: ta xét từ cuối dãy (chỉ đến phần tử thứ 2), nếu gặp 2 phần tử kề nhau mà sai thứ tự thì đổi chỗ chúng cho nhau. Sau lượt 2, phần tử có khoá nhỏ thứ hai được đưa về vị trí 2.

- Lượt i: ta xét từ cuối dãy về (chỉ đến phần tử thứ i, vì phần đầu dãy từ 1 đến i-1 đã được xếp đúng thứ tự), nếu gặp 2 phần tử kề nhau mà sai thứ tự thì đổi chỗ chúng cho nhau. Sau lượt i, phần tử có khoá nhỏ thứ i được đưa về vị trí i.

Xong lượt thứ *n*-1 thì dãy được sắp xếp xong.

```
Procedure BubbleSort;
Var  T:Obj;
     i,j:longint;
Begin
     For i:=1 to n-1 do
          For j:=n downto i+1 do
               If A[j-1].key > A[j].key then
               Begin
                    T:=A[j-1];
                    A[j-1]:=A[j];
                    A[j]:=T;
               End;
End;
```

Đánh giá độ phức tạp

Số phép toán so sánh trong thuật toán SelectionSort và BubbleSort cũng như số phép toán dịch chuyển trong thuật toán InsertSort được dùng để đánh giá hiệu suất về mặt thời gian. Tại lượt thứ *i* ta cần *n-i* phép toán.

Như vậy tổng số phép toán là: $(n\text{-}1) + (n\text{-}2) + \ldots + 1 = n(n\text{-}1)/2$.

Vì vậy, ba thuật toán trên đều có độ phức tạp là $O(n^2)$.

1.6. Sắp xếp bằng đếm phân phối (Distribution Counting)

Trong trường hợp khoá các phần tử $A[1], A[2], \ldots, A[n]$ là các số nguyên nằm trong khoảng từ 0 tới k ta có thuật toán đơn giản và hiệu quả như sau: Xây dựng dãy $C[1], C[2], \ldots, C[k]$, trong đó $C[v]$ $(1 \le v \le k)$ là số lần xuất hiện khoá v trong dãy.

Như vậy, mảng *C* được xây dựng như sau:

```
For i:=1 to k do C[i]:=0 {khởi tạo mảng C}
For i:=1 to n do C[A[i].key]:=C[A[i].key]+1;
```

Sau khi sắp xếp:

- Các phần tử có khóa bằng 0 đứng trong đoạn từ vị trí 1 đến vị trí *C*[0];

- Các phần tử có khóa bằng 1 đứng trong đoạn từ vị trí *C*[0]+1 đến vị trí *C*[1];

- Các phần tử có khóa bằng 2 đứng trong đoạn từ vị trí *C*[0]+*C*[1]+1 đến vị trí *C*[2];

- Các phần tử có khóa bằng 3 đứng trong đoạn từ vị trí *C*[0]+*C*[1]+*C*[2]+1 đến vị trí *C*[3];

…

- Các phần tử có khóa bằng *k* đứng trong đoạn từ vị trí *C*[0]+*C*[1]+…+*C*[*k*-1]+1 đến vị trí *C*[*k*];

Ví dụ với dãy có 10 phần tử với dãy khóa: 2, 1, 2, 4, 3, 5, 1, 2, 4, 3

Ta có mảng *C* như sau:

C[0]	*C*[1]	*C*[2]	*C*[3]	*C*[4]	*C*[5]
0	2	3	2	2	1

Dãy khóa sau khi sắp xếp là: 1, 1, 2, 2, 2, 3, 3, 4, 4, 5

Độ phức tạp của thuật toán là O(*max*(*n*,*k*)).

1.7. Sắp xếp nhanh (Quick Sort)

Ý tưởng của thuật toán như sau: Để sắp xếp dãy coi như là sắp xếp đoạn từ chỉ số 1 đến chỉ số *n*. Để sắp xếp một đoạn trong dãy, nếu đoạn chỉ có một phần tử thì dãy đã được sắp xếp, ngược lại ta chọn một phần tử *x* trong đoạn đó làm "chốt", mọi phần tử có khoá nhỏ hơn khoá của “chốt” được xếp vào vị trí đứng trước chốt, mọi phần tử có khoá lớn hơn khoá của “chốt” được xếp vào vị trí đứng sau chốt. Sau phép hoán chuyển như vậy thì đoạn đang xét được chia làm hai đoạn mà mọi phần tử trong đoạn đầu đều có khoá nhỏ hơn khoá của “chốt” và mọi phần tử trong đoạn sau đều có khoá lớn hơn khoá của “chốt”. Tiếp tục sắp xếp kiểu như vậy với hai đoạn con, ta sẽ được đoạn đã cho được sắp xếp theo chiều tăng dần của khoá.

Cụ thể:

Giả sử cần sắp xếp dãy các phần tử có chỉ số từ *l* đến *r*.

+ Chọn *X* là phần tử ngẫu nhiên trong đoạn *l…r* (có thể chọn *X* là phần tử ở giữa đoạn, *X*=*A*[(*l*+*r*) div 2]).

+ Cho *i* chạy từ *l* sang phải, *j* chạy từ *r* sang trái; nếu phát hiện một cặp ngược thứ tự (*i* ≤ *j* và *A*[*i*].key ≥ *A*[*j*].key) thì đổi chỗ hai phần tử này; cho đến khi *i* > *j*. Lúc đó dãy ở tình trạng: khóa của các phần tử từ *l* đến *j* nhỏ hơn *X*.key và khóa của các phần tử từ *i* đến *r* lớn hơn *X*.key. Tiếp tục thực hiện thao tác sắp xếp như vậy với hai đoạn *l…j* và *i…r*.

Thủ tục QuickSort(l, r) sau, sắp xếp đoạn từ *l* tới *r*, để sắp xếp dãy số ta gọi QuickSort(1, n).

```
Procedure QuickSort(l,r:longint);
Var  i,j:longint;
     T,X:Obj;
Begin
     i:=l; j:=r;
     X:= A[(l+r) div 2]; //X:= A[random(r-l+1)+l];
     repeat
          While A[i].key < X.key do inc(i);
          While A[j].key > X.key do dec(j);
          If i<=j then
          Begin
               T:=A[i]; A[i]:=A[j]; A[j]:=T;
               i:=i+1; j:=j-1;
          End;
     Until i>j;
     If j>l then QuickSort(l,j);
     If i<r then QuickSort(i,r);
End;
```

Đánh giá độ phức tạp

Việc chọn chốt để phân đoạn quyết dịnh hiệu quả của thuật toán, nếu việc chọn chốt không tốt rất có thể việc phân đoạn bị suy biến thành trường hợp xấu (phân thành hai đoạn mà số phần tử của hai đoạn chênh lệch nhiều) khiến Quick Sort hoạt động chậm. Các tính toán dộ phức tạp chi tiết cho thấy thuật toán Quick sort:

- Có thời gian thực thi cỡ O(nlgn) trong trường hợp trung bình.

- Có thời gian thực thi cỡ $O(n^2)$ trong trường hợp xấu nhất (2 đoạn dược chia thành một đoạn n-1 và một doạn 1 phần tử). Khả năng để xảy ra trường hợp này là rất ít, còn nếu chọn chốt ngẫu nhiên, hầu như sẽ không xảy ra.

1.8. Một số ví dụ về bài toán sắp xếp

Ví dụ 1: Giá trị nhỏ thứ *k*

Cho dãy số nguyên $A[1], A[2], \ldots, A[n]$, các số đôi một khác nhau và số nguyên dương k ($1 \le k \le n$). Hãy đưa ra gá trị nhỏ thứ k trong dãy.

Dữ liệu vào: Tệp văn bản MINK.INP gồm:

+ Dòng đầu ghi hai số nguyên n và k ($1 \le k \le n \le 10^5$) cách nhau một dấu cách.

+ Dòng thứ hai ghi n số nguyên $A[1], A[2], \ldots, A[n]$ có giá trị tuyệt đối không vượt quá 10^6, giữa các số cách nhau một dấu cách.

Kết quả: Đưa ra tệp văn bản MINK.OUT, chỉ ghi một số duy nhất là số nhỏ thứ k trong dãy.

Ví dụ dãy: 6, 7,8, 4, 3, 2 và $k = 4$ thì giá trị nhỏ thứ k là 6.

MINK.INP	MINK.OUT
6 4 6 7 8 4 3 2	6

Giải:

+ Sắp xếp các số trong dãy theo thứ tự tăng dần. Với $n \le 5000$, có thể sử dụng một trong ba thuật toán sắp xếp đơn giản như sắp xếp chọn, chèn hoặc nổi bọt, nhưng nếu $n > 5000$ thì cần phải sử dụng thuật toán sắp xếp nhanh Quick Sort.

+ Sau khi sắp xếp, số nhỏ thứ là $A[k]$.

Trường hợp 1 ($n \le 5000$) ta có thể sử dụng thuật toán sắp xếp chọn và chỉ sắp xếp đến phần tử thứ k.

Trường hợp 2 ($n > 5000$) ta phải sử dụng thuật toán sắp xếp nhanh Quick Sort. Lưu ý: chúng ta chỉ quan tâm đến đoạn từ l đến r chứa k, nếu k nằm ngoài đoạn l, r thì không cần phải sắp xếp.

Sau đây là chương trình được cài đặt bằng Pascal

```
const     fi =  'MINK.INP';
          fo = 'MINK1.OUT';
          maxn = 100000;
type tarray = array[1..maxn] of longint;
var  n,k:longint;
     A:tarray;
procedure Input;
var  f:text;
     i:longint;
begin
     assign(f,fi); reset(f);
     readln(f,n,k);
     for i:=1 to n do read(f,A[i]);
     close(f);
end;
procedure SelectionSort;
var i,j,X:longint;
begin
     for i:=1 to k do
       for j:=i+1 to n do
          if A[i]>A[j] then
          begin
               X:=A[i];
               A[i]:=A[j];
               A[j]:=x;
          end;
end;
procedure QuickSort(l,r:longint);
var i,j,X,T:longint;
begin
     if (k < l) or (k > r) then exit;
     i:=l; j:=r; X:=A[(l+r) div 2];
     repeat
          while A[i]<X do inc(i);
          while A[j]>X do dec(j);
          if i<=j then
          begin
               T:=A[i]; A[i]:=A[j]; A[j]:=T;
               inc(i); dec(j);
          end;
     until i>j;
     if l<j then QuickSort(l,j);
     if r>i then QuickSort(i,r);
```

```
end;

procedure Output;
var f:text;
begin
     assign(f,fo); rewrite(f);
     writeln(f,A[k]);
     close(f);
end;
BEGIN
     Input;
     //SelectionSort;
     QuickSort(1,n);
     Output;
END.
```

Ví dụ 2: Số tự nhiên nhỏ nhất không xuất hiện trong dãy

Cho dãy gồm n ($0 < n \leq 30000$) số tự nhiên có giá trị không vượt quá 10^9. Hãy tìm số tự nhiên nhỏ nhất không xuất hiện trong dãy.

Dữ liệu vào: Tệp văn bản STNNN.INP gồm:

+ Dòng đầu ghi số số n.

+ Dòng thứ hai ghi n số trong dãy.

Kết quả: Ghi ra tệp văn bản STNNN.OUT ghi số tự nhiên nhỏ nhất không xuất hiện trong dãy.

Ví dụ

STNNN.INP	STNNN.INP
5 5 0 3 1 4	2

Giải:

Ta có nhận xét rằng, số tự nhiên nhỏ nhất không xuất hiện trong dãy sẽ nằm trong đoạn [0, n]. Do đó ta có thể sử dụng mảng *C:Array*[0..30000] *of longint*; với $C[x]$ là số lần xuất hiện của số x trong dãy, nếu $C[x]=0$ nghĩa là x không xuất hiện trong dãy.

Chương trình có thể cài đặt như sau:

```
const
     fi = 'STNNN.INP';
```

```
        fo = 'STNNN.OUT';
        maxn = 30000;
    var  n:longint;
        C:array[0..maxn] of longint;

    procedure Input;
    var f:text;
        x,i:longint;
    begin
        fillbyte(C,sizeof(C),0);
        assign(f,fi); reset(f);
        readln(f,n);
        for i:=1 to n do
        begin
            read(f,x);
            if x<=n then C[x]:=C[x]+1;
        end;
        close(f);
    end;
    procedure Output;
    var f:text;
       i:longint;
    begin
        assign(f,fo); rewrite(f);
        i:=0;
        while C[i]>0 do i:=i+1;
        write(f,i);
        close(f);
    end;
    BEGIN
        Input;
        Output;
    END.
```

2. Tìm kiếm

Tìm kiếm là công việc quan trọng trong đời sống cũng như trong các hệ thống tin học. Ví dụ như tìm đường đi trên bản đồ, tìm họ tên trong danh sách, tra một từ nào đó trong từ điển… Trong các hệ thống thông tin thường phải lưu trữ một khối lượng dữ liệu lớn nên việc xây dựng các giải thuật cho phép tìm kiếm nhanh sẽ có ý nghĩa rất lớn. Nếu dữ liệu trong hệ thống đã được tổ chức theo một trật tự nào đó, thì việc tìm kiếm sẽ tiến hành nhanh chóng và hiệu quả hơn.

Tìm kiếm tức là chỉ ra đối tượng thỏa mãn một điều kiện nào đó nằm trong không gian tìm kiếm gồm nhiều đối tượng. Trong tin học, không gian tìm kiếm có rất nhiều dạng cấu trúc như danh sách, đồ thị, cây… Ở đây, chúng tôi chỉ đề cập đến một số giải thuật tìm kiếm đơn giản trên danh sách hay mảng.

2.1. Phát biểu bài toán

Giả sử các đối tượng được biểu diễn bởi bản ghi gồm một số trường. Một trong các trường đó được gọi là khoá tìm kếm. Cấu trúc của đối tượng được định nghĩa bằng Pascal như đã trình bày trong phần 1.2. của tài liệu này.

Bài toán tìm kiếm được phát biểu như sau: Cho một dãy các đối tượng $A[1]$, $A[2]$,… $A[n]$. Hãy tìm một phần tử có khóa bằng x cho trước.

Kết quả của bài toán tìm kiếm có thể xảy ra một trong hai trường hợp: Nếu tìm được đối tượng thỏa điều kiện bài toán (có khóa bằng x) thì chỉ ra vị trí của đối tượng trong dãy, ngược lại nếu không tìm thấy đối tượng trong dãy thì trả về giá trị 0.

2.2. Tìm kiếm tuần tự (Sequence Searching)

Ý tưởng của thuật toán tìm kiếm tuần tự là duyệt từ đầu dãy đến cuối dãy cho đến khi tìm được phần tử thỏa điều kiện bài toán hoặc duyệt hết dãy mà không tìm thấy phần tử nào thỏa điều kiện.

```
Function SeqSearch:longint;
Var i:longint;
Begin
     For i:= 1 to n do
        If A[i].key = x then exit(i);
      exit(0);
End;
```

2.3. Tìm kiếm nhị phân (Binary Searching)

Điều kiện để áp dụng được thuật toán tìm kiếm nhị phân là các phần tử trong dãy đã được sắp xếp theo chiều không giảm của khóa tìm kiếm.

Tức là: $A[1].key \le A[2].key \le \ldots \le A[n].key$

Ý tưởng của thuật toán tìm kiếm nhị phân: Chọn một phần tử giữa dãy, nếu phần tử này có khóa bằng *x* thì phép tìm kiếm thỏa mãn và dừng lại. Ngược lại nếu khóa của phần tử được chọn lớn hơn *x* thì lặp lại phép tìm kiếm với dãy từ phần tử đầu đến phần tử kế trước phần tử được chọn. Còn nếu khóa phần tử được chọn nhỏ hơn *x* thì lặp lại phép tìm kiếm với dãy từ phần tử kế sau phần tử được chọn đến phần tử cuối dãy. Trong trường hợp này, phép tìm kiếm chỉ dừng lại khi dãy tìm kiếm không còn phần tử nào.

Giải thuật tìm kiếm nhị phân được mô tả như sau:

```
function BinaSearch(x: keytype):longint;
var l, r, m: longint;
begin
    l:=1; r:=n;
    while l ≤ r do
    begin
        m:= (l + r) div 2;
        if a[m].key = x then exit(mid);
        if a[m].key < x then l:= m + 1
        else r:= m-1;
    end;
    exit(0);
end;
```

Thuật toán tìm kiếm nhị phân có thể mô tả bằng kỹ thuật đệ quy với dãy gồm các phần tử có vị trí đầu là *l* và vị trí cuối là *r* được mô tả như sau:

```
Function BinaSearch(l,r:longint): longint;
Var m:longint;
Begin
     If l>r then exit(0);
     m:= (l+r) div 2;
     If A[m].key = x then return(m)
     Else
       if A[m].key > x then exit(BinaSearch(l,m-1))
       Else exit(BinaSearch(m+1,r));
End;
```

Độ phức tạp của thuật toán tìm kiếm nhị phân là O(logn)

Ví dụ 3: Cho dãy số $A[1] \le A[2] \le \ldots \le A[n]$ ($1 < n \le 10^7$), các số trong dãy là số nguyên có giá trị tuyệt đối không vượt quá 10^9. Hãy đưa ra chỉ số i mà $A[i] = x$ cho trước hoặc đưa ra i=0 nếu không có phần tử nào có giá trị bằng x.

Dữ liệu vào: Tệp văn bản Bsearch.INP gồm:

+ Dòng đầu ghi số n và số x, hai số cách nhau một dấu cách.

+ Dòng thứ hai ghi n số nguyên $A[1], A[2], \ldots, A[n]$. Giữa các số cách nhau một dấu cách.

Kết quả: Ghi ra tệp văn bản Bsearch.OUT chỉ một số nguyên duy nhất là vị trí của phần tử có giá trị bằng x.

Ví dụ:

Bsearch.INP	Bsearch.OUT
8 10 2 4 6 7 9 11 15 20	0
8 15 2 4 6 7 9 11 15 20	7

Giải:

Do dãy đã cho đã được sắp xếp theo thứ tự không giảm nên ta có thể áp dụng thuật toán tìm kiếm nhị phân để giải quyết bài toán này như sau:

```
const     fi = 'BSearch.INP';
          fo = 'BSearch.OUT';
          maxn = 10000000;
var       n,x:longint;
          A:array[0..maxn] of longint;
procedure Input;
var f:text;
    i:longint;
begin
     assign(f,fi); reset(f);
     readln(f,n,x);
     for i:=1 to n do read(f,A[i]);
     close(f);
end;
Function BinaSearch(l,r:longint): longint;
Var m:longint;
Begin
     if l>r then exit(0);
```

```
        m:= (l+r) div 2;
        if A[m] = x then exit(m)
        if A[m] > x then exit(BinaSearch(l,m-1))
        else exit(BinaSearch(m+1,r));
    End;
    procedure Output;
    var f:text;
    begin
        assign(f,fo); rewrite(f);
        write(f,BinaSearch(1,n));
        close(f);
    end;
    BEGIN
        Input;
        Output;
    END.
```

3. Bài tập

Bài 1: Quà sinh nhật

Công ty Alpha có N nhân viên được đánh số thứ tự từ 1 đến N. Mỗi tháng trong năm công ty tổ chức tặng quà sinh nhật cho tất cả các nhân viên của mình sinh trong tháng đó.

Yêu cầu: Cho biết nhân viên thứ i (i=1..N) của công ty được sinh vào tháng T_i. Bạn hãy giúp công ty tính số lượng món quà cần mua cho mỗi tháng trong một năm.

Dữ liệu vào: Tệp văn bản SINHNHAT.INP gồm:

+ Dòng đầu ghi số nguyên dương N ($0 < N \leq 10^6$).

+ Dòng thứ hai ghi N số nguyên, với số thứ i là T_i (i = 1..N) có giá trị từ 1 đến 12 là tháng sinh của nhân viên thứ i. Các số trên cùng dòng được ghi cách nhau một dấu cách.

Kết quả: Ghi ra tệp văn bản SINHNHAT.OUT theo từng dòng, với mỗi dòng gồm hai số nguyên, số đầu là tháng trong năm, số thứ hai là số món quà cần mua cho tháng đó, hai số cách nhau một dấu cách. Chỉ số tháng trên các dòng theo thứ tự tăng dần, chỉ số tháng nhỏ ghi ở dòng trước, chỉ số tháng lớn hơn ghi ở dòng sau.

Ví dụ:

SINHNHAT.INP	SINHNHAT.OUT
10 1 3 1 3 4 3 11 9 11 4	1 2 3 3 4 2 9 1 11 2

Bài 2: Số lặp

Cho một dãy gồm N số nguyên không âm A_1, A_2,…, A_N. Người ta cần loại bỏ một số các số bị lặp lại trong dãy A sao cho các số còn lại trong dãy đôi một khác nhau (không có số nào bị lặp lại). Nếu một số xuất hiện K lần (K>1) trong dãy thì phải loại bỏ đi K-1 số bị lặp và chỉ giữ lại đúng một số. Hỏi sau khi loại bỏ hết tất cả các số bị lặp trong dãy thì số lượng các số trong dãy A còn lại là bao nhiêu?

Dữ liệu vào: tệp văn bản SOLAP.INP gồm:

+ Dòng đầu ghi số nguyên dương N ($2 \leq N \leq 10000$);

+ Dòng thứ 2 ghi N số nguyên A1, A2,…, AN ($0 \leq$ Ai ≤ 1000 với i chạy từ 1 đến N), mỗi số cách nhau đúng một dấu cách.

Dữ liệu ra: tệp văn bản SOLAP.OUT chỉ ghi một số duy nhất là số lượng các số còn lại của dãy A sau khi đã loại bỏ các số bị lặp.

Ví dụ

SOLAP.INP	SOLAP.OUT
7 1 2 3 3 4 10 2	5

Bài 3: Cặp số có tích lớn nhất

Cho dãy gồm N số nguyên $A_1, A_2, \ldots, A_N$. Hãy tìm cặp số A_i, A_j ($1 \leq i \neq j \leq N$) sao cho tích của chúng là lớn nhất. Ví dụ với dãy số: 3, 4, 1, 2, 3, 9, 1, 3 thì hai số được chọn là 4 và 9 vì tích của chúng $4 \times 9 = 36$ là lớn nhất.

Dữ liệu vào: Từ tệp văn bản CAPSO.INP gồm:

+ Dòng đầu tiên ghi số nguyên N ($2 \leq N \leq 10^7$).

+ Dòng thứ hai ghi N số nguyên, $A_1, A_2, \ldots, A_N$ có giá trị tuyệt đối không vượt quá 10^9.

Kết quả: ghi vào tệp văn bản CAPSO.OUT một số nguyên duy nhất là tích lớn nhất tìm được.

Ví dụ:

CAPSO.INP	CAPSO.OUT
8 3 4 1 2 3 9 1 3	36

Bài 4: Biểu thức lớn nhất

Một dãy gồm N số nguyên không âm $A_1, A_2, \ldots, A_N$ được viết thành một hàng ngang, giữa hai số liên tiếp có một khoảng trắng, như vậy có tất cả N - 1khoảng trắng. Người ta muốn đặt k dấu cộng và N - k - 1 dấu trừ vào N - 1 khoảng trắng đó để nhận được một biểu thức có giá trị lớn nhất.

Ví dụ, với dãy gồm 5 số nguyên 28, 9, 5, 1, 69 và k = 2 thì cách đặt $28+9-5-1+69$ là biểu thức có giá trị lớn nhất.

Yêu cầu: Cho dãy gồm N số nguyên không âm $A_1, A_2, \ldots, A_N$ và số nguyên dương k, hãy tìm cách đặt k dấu cộng và $N - k - 1$ dấu trừ vào $N - 1$ khoảng trắng để nhận được một biểu thức có giá trị lớn nhất.

Dữ liệu vào: từ tệp văn bản PTIT016D.INP gồm:

+ Dòng đầu chứa hai số nguyên dương N và k ($0 < k < N \leq 10^6$) cách nhau một dấu cách.

+ Dòng thứ hai ghi N số nguyên không âm $A_1, A_2, \ldots, A_N$ ($0 \leq A_i \leq 10^6$; $i = 1 \div N$), giữa các số cách nhau một dấu cách.

Kết quả: Ghi vào tệp văn bản PTIT016D.OUT chỉ ghi một số nguyên là giá trị của biểu thức đạt được.

Ví dụ:

PTIT016D.INP	PTIT016D.OUT
5 2 28 9 5 1 69	100

Bài 5: Hành tinh Hough

Hành tinh Hough xa xôi được chia thành hai nửa bán cầu có trình độ khoa học kỹ thuật khác hẳn nhau. Ở Hough A, ngành khoa học máy tính và tự động hóa phát triển mạnh mẽ, mọi công việc của họ đều được thực hiện một cách tự động. Trái ngược với Hough A, ở Hough B trình độ của họ vẫn đang ở mức thời kỳ đồ sắt bởi lẽ họ đề cao lao động chân tay.

Mặc dù trình độ khoa học kỹ thuật ở hai bán cầu rất khác nhau nhưng họ vẫn chung sống hòa bình với nhau từ thế kỷ này qua thế kỷ khác. Hằng năm họ còn tổ chức một cuộc thi lớn dành cho tất cả người dân trên hành tinh. Ở Hough A có ***n*** người tham dự còn Hough B có ***m*** người tham dự. Nhờ chương trình đo sức khỏe nên Hough A biết trước được chỉ số sức khỏe của tất cả người dân trên hành tinh. Trong cuộc thi, mỗi lượt sẽ có hai người ở hai bán cầu lên thi đấu, người có chỉ số sức khỏe tốt hơn sẽ dành chiến thắng, mỗi người chỉ được thi đấu tối đa một lần.

Yêu cầu: Hãy cho biết mỗi người tham dự Hough B có chỉ số sức khỏe lớn hơn bao nhiêu người tham dự của Hough A.

Dữ liệu vào: Từ tệp văn bản **HOUGH.INP** gồm:

+ Dòng đầu tiên ghi hai số nguyên dương ***n*** và ***m***.

+ Dòng thứ hai ghi ***n*** số nguyên dương $a_1, a_2, \ldots, a_n$ cho biết chỉ số sức khỏe của ***n*** người tham dự cuộc thi ở Hough A.

+ Dòng thứ ba ghi ***m*** số nguyên dương $b_1, b_2, \ldots, b_m$ cho biết chỉ số sức khỏe của ***m*** người tham dự cuộc thi ở Hough B.

Kết quả: Ghi vào tệp văn bản **HOUGH.OUT** gồm ***m*** dòng, dòng thứ ***i*** cho biết người thứ ***i*** ở Hough B có chỉ số sức khỏe lớn hơn bao nhiêu người ở Hough A.

Ví dụ:

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

Gới hạn: Trong tất cả các test các số nguyên không vượt quá 10^9

+ Có 50% số test tương ứng với 50% số điểm có n, m ≤ 10^3

+ 50% số test còn lại tương ứng với 50% số điểm có n, m ≤10^5

TÀI LIỆU THAM KHẢO

[1] *Hồ Sỹ Đàm, Đỗ Đức Đông, Lê Minh Hoàng, Nguyễn Thanh Hùng* - Tài liệu giáo khoa chuyên tin quyển 1 – Nhà xuất bản giáo dục.

[2] Đề thi tuyển sinh lớp 10 chuyên Lê Quý Đôn – Khánh Hòa năm 2016-2017, năm 2017 - 2018 .