Lab 2、Stacks

1000. Bracket Matching

Time Limit: 1sec Memory Limit:256MB

Description

检查输入字符串中的括号是否匹配。括号包括：{, }, (,), [,].

Input

第一行是一整数，即测试样例个数 n.以下 n 行，每一行是一个长度不超过 100 个字符的字符串。

Output

第一行是一整数，即测试样例个数 n.以下 n 行，每一行是一个长度不超过 100 的字符串。

Sample Input

Copy sample input to clipboard

```
3   (换行)  a
2-[(1+2)*2]
(a+b])
```

Sample Output

```
Yes  (换行)  Yes
```

```
//检查输入字符串中的括号是否匹配。括号包括：{, }, (, ), [, ].
//Input 第一行是一整数，即测试样例个数 n.以下 n 行，每一行是一个长度不超过 100 个字符的字符串。
//Output 第一行是一整数，即测试样例个数 n.以下 n 行，每一行是一个长度不超过 100 的字符串。
#include<iostream>
#include<stack>
#include<cstdio>
#include<algorithm>
using namespace std;
int main(){
    stack<char> openings;
    char symbol;
    bool matched= true;
    bool go_on=true;
    int n;
    cin>>n;
```

```
    cin.get();
    while(n--)
    {

        go_on=true;
        while(go_on){
            symbol=cin.get();

            if(symbol=='{'||symbol=='('||symbol=='['
            openings.push(symbol);
            if(symbol=='}'||symbol==')'||symbol==']')
            {
                if(openings.empty())

                    matched=false;

                else {
                    char match;
                    match=openings.top();
                    openings.pop();
                        if(matched){
                        matched = ((symbol == '}'&&match == '{')||(symbol == ')'&&match == '('))||(symbol
== ']'&&match == '['));
                    }
                    }

             }
            if(symbol=='\n'){
                if(!openings.empty()||!matched) {
                    cout<<"No"<<endl;
                     go_on=false;
                    while(!openings.empty()){
                        openings.pop();
                    }
                }
                else{
                    cout<<"Yes"<<endl;
                    go_on=false;
                }
                matched = true;
            }
        }
    }
    return 0;}
```

1001. 后缀表达式计算

Description

来自 USA 的 Mr.Allison 是某班的英语老师，学期开始之前他要求自己班的同学阅读一本英文小说，并写一份 50000 字的读书报告。该班的同学很苦恼，并想方设法让 Mr.Allison 放弃读书笔记这个念头，于是该班的大牛 PYY 想到一个借口：看那么多份读书笔记会花费很多时间的！把这个理由告诉 Mr.Allison 之后，Mr.Allison 也觉得挺有道理，但一共要阅读多少文字呢？于是 PYY 就给出一条后缀表达式，并告诉 Mr.Allison 说，这条表达式的结果就是您要阅读的文字。Mr.Allison 的数学不咋地，于是就找你来帮他计算这条后缀表达式的值。

Input

第一行是一整数，即测试样例个数 T.
以下 T 行，每一行是一个长度不超过 100 的字符串，代表一条后缀表达式。表达式中只含有+、-、*、/四种运算符和 26 个小写英文字母，不含其它字符。每一个英文字母代表一个正整数：a = 1,b = 2,c = 3...y = 25,z = 26。

Output

每一个输入样例，单独一行输出结果：后缀表达式的值，一个正实数 S，保留两位小数。

Sample Input

Copy sample input to clipboard

```
2
ab+c*
int**py++
```

Sample Output

```
9.00
```

```
//给出后缀表达式计算其结果
//Input 第一行是一整数，即测试样例个数 T.
//以下 T 行，每一行是一个长度不超过 100 的字符串，代表一条后缀表达式。
//表达式中只含有+、-、*、/四种运算符和 26 个小写英文字母，不含其它字符。
//每一个英文字母代表一个正整数
//a = 1,b = 2,c = 3...y = 25,z = 26。
//Output 每一个输入样例，单独一行输出结果：后缀表达式的值，一个正实数 S，保留两位小数。

#include<iostream>
#include<cstring>
#include<stack>
#include<iomanip>

using namespace std;
```

```
int main()
{
	int T;
	cin>>T;
	string str;

	while(T--)
	{
		stack<double> s;
		while(!s.empty())
		{
			s.pop();//clear the stack
		}

		cin>>str;
		for(int j=0;j<str.length();j++)
		{
			if(str[j]>='a' && str[j]<='z')
			s.push(str[j]-'a'+1);
			else
			{
				double a=s.top();
				s.pop();
				//	cout<<a<<endl;
				double b=s.top();
				//	cout<<b<<endl;
				s.pop();

				if(str[j]=='+')
				s.push(a+b);
				else if(str[j]=='-')
				s.push(b-a);
				else if(str[j]=='*')
				s.push(a*b);
				else if(str[j]=='/')
				s.push(1.00*b/a);
			}
		}
		double result=s.top();
		cout<<fixed<<setprecision(2)<<result<<endl;
	}
	return 0;
}
```

1002. 中缀表达式转后缀表达式（带括号）

Description

将中缀表达式（infix expression）转换为后缀表达式（postfix expression）。假设中缀表达式中的操作数均以单个英文字母表示，且其中只包含左括号'('，右括号')'和双目算术操作符+，-，*，/。

Input

第一行是测试样例个数n。以下n行，每行是表示中缀表达式的一个字符串（其中只包含操作数和操作符和左右括号，不包含任何其他字符），长度不超过100个字符。

Output

为每一个测试样例单独一行输出对应后缀表达式字符串（其中只包含操作数和操作符，不包含任何其他字符）

Sample Input

Copy sample input to clipboard

```
2
A+B*C-D-E/F
a+(b-c)*d+e/f
```

Sample Output

```
ABC*+D-EF/-
```

```
//1002. 中缀表达式转后缀表达式（带括号）
//将中缀表达式（infix expression）转换为后缀表达式（postfix expression）。
//假设中缀表达式中的操作数均以单个英文字母表示，
//且其中只包含左括号'('，右括号‘)’和双目算术操作符+，-，*，/。
//第一行是测试样例个数n。
//以下n行，每行是表示中缀表达式的一个字符串
//其中只包含操作数和操作符和左右括号，不包含任何其他字符），长度不超过100个字符。
//Output 为每一个测试样例单独一行输出对应后缀表达式字符串（其中只包含操作数和操作符，不包含任何其他字符）
#include<iostream>
#include<stack>
using namespace std;

int main() {
	int N;
	cin >> N;
```

```
cin.get();
char sym;
char ori[100];
stack <char> sta;
for (int i = 0; i < N; i++) {
	cin >> ori;
	int j = 0;
	while(!sta.empty()){
		sta.pop();
	}// clear stack.
	while (ori[j] != '\0') {
		sym = ori[j];
		if (sym != '+' && sym != '*' && sym != '/' && sym != '-' && sym != '(' && sym != ')') {
			cout << sym;
		}
		else {
			if (!sta.empty()) {
				if (sym == '*' || sym == '/' || sym == '(') sta.push(sym);
				else if (sym == ')') {
					while (sta.top() != '(') {
						cout << sta.top();
						sta.pop();
					}
					sta.pop();
				}
				else if (sym == '+' || sym == '-') {
					while (!sta.empty()) {
						if (sta.top() != '(') {
							cout << sta.top();
							sta.pop();
						}
						else break;
					}
					sta.push(sym);
				}
			}

			else sta.push(sym);
		}
		j++;
	}
	while (!sta.empty()) {
		cout << sta.top();
		sta.pop();
```

```
            }
            cout << endl;
        }
        return 0;
}
```

Lab 4、Queues

1000. 买礼物

Description

公元前 XXX 年，有一位鼎鼎大名的大魔王，大魔王不仅拥有万里江山，而且后宫有 n 个美丽的老婆。为讨好他的大小老婆们，大魔王决定给她们各送一分礼物。于是他就把买礼物的任务交给他的好心（ji）腹（you）CC。为了防止老婆们明争暗斗争风吃醋，大魔王认为使礼物价格的差值尽量的小，那必是极好的。说人话：如果 A 表示这 n 份物品中价格的最大值，B 表示这 n 份物品中价格的最小值，要使 A-B 尽可能小。当听话的 CC 去到商店时，店员给 CC 展示了 m 份礼物，CC 要在这 m 份礼物中选择 n 份，并使它们符合 SB 大魔王的要求。假如你是 CC，请你编写一个程序解决这个问题。

Input

题目包含多组测试数据，第一行为数据组数 k。每个测试数据有两行，首行为两个整数 n,m，$2 \le n \le m \le 50$，n 为大王的老婆数，次行包含 m 个整数，用空格分开，表示每份礼物的价格。当输入一个 0 时表示结束输入

Output

一个整数，表示 A-B 的最小值。

Sample Input

Copy sample input to clipboard

```
2
4 6
10 12 10 7 5 22
2 10
4 5 6 7 8 9 10 11 12 12
0
```

Sample Output

```
5
```

```
//Input 题目包含多组测试数据，第一行为数据组数 k。
//每个测试数据有两行，首行为两个整数 n,m，2≤n≤m≤50，
```

```
//n 为大王的老婆数，次行包含 m 个整数，用空格分开，表示每份礼物的价格。当输入一个 0 时表示结束输入
//Output 一个整数，表示 A-B 的最小值。
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;
int main(){
	int t;
	int n,m,min;
	int x[100];
	queue<int> gift;
	cin>>t;
	while(1){
		cin>>n;
		if(n==0) break;
		else cin>>m;
		for(int i=0;i<m;i++){
			cin>>x[i];
		}
		sort(x,x+m);
		for(int i=0;i<m;i++){
			int temp;
			gift.push(x[i]);
			if(gift.size()==n){
				if(i==n-1){
					min=temp=gift.back()-gift.front();
				}
				else{
					temp=gift.back()-gift.front();
					if(temp<min){
					min=temp;
				}
				}
				gift.pop();
			}
		}
		cout<<min<<endl;
		while(!gift.empty()){
			gift.pop();
		}
	}
	return 0;
}
```

1001. Boring

Description

After the make-up examine,Tiantian is bored because he needn't review *digital circuit* now.So he decides to calculate the number of people in the queue.What a boring boy!

There are three operations of the queue:

In x: push x(x <= 10000) in the end of the queue.

Out: write out the first element of the queue and erase it.If the queue is empty,write out -1.

Count: return the length of the queue.

Input

The first number T(T <= 20) means the number of tests.

Each test contains a single number N(N <= 1000) means the number of operations.Following n lines describe the operation of the queue.

Output

For each Out and Count operation,write out its expected answer.

Sample Input

```
2
10
Count
Count
In 9035
Out
Out
In 9809
In 4983
Out
Count
Out
10
```

In 1589
Count
Count
Count
Count
In 7173
Count
In 4567
In 120
Out

Sample Output

0
0
9035
−1
9809
1
4983
1
1
1
1
2

```
   //There are three operations of the queue:
//In x: push x(x <= 10000) in the end of the queue.
//Out:write out the first element of the queue and erase it.If the queue is empty,write out -1.
//Count: return the length of the queue.The first number T(T <= 20) means the number of tests.
//Input~Each test contains a single number N(N <= 1000) means the number of operations.
//Following n lines describe the operation of the queue.
//Output~For each Out and Count operation,write out its expected answer.
#include<iostream>
#include<queue>
using namespace std;
int main(){
	int t,n,x;
	char a[10];
	queue<int> num;
	cin>>t;
	while(t--){
		cin>>n;
		for(int i=0;i<n;i++){
			cin>>a;
```

```
            if(a[0]=='C'){
                cout<<num.size()<<endl;
            }
            else if(a[0]=='I'){
                cin>>x;
                num.push(x);

            }
            else if(a[0]=='O'){
                if(num.empty()){
                    cout<<"-1"<<endl;
                }
                else {
                    cout<<num.front()<<endl;
                    num.pop();
                }
            }
        }
        while(!num.empty()){
            num.pop();
        }
    }
    return 0;
}
```

1002. Linked list

Description

There is a sequence with n(<=10^5) elements number from 1~n. There are two types of operations:
Operation one : (1,x), means inserting an integer x immediately after the current position p (initially p = 0).
Operation two: (2,k), means moving after the current position p by k (k >= 0) elements. (For example , p = 0. after operation(2,2), p = 2)
Now, give you m(<=10^5) operations, please print the sequence after m operations.

Input

The first line contains two integer n,m, followed by n lines, one integer per line, showing the original sequ
The next n+2 to n+m+1 lines, each has two integers, the operations described above.

Output

One number per line, the sequence after m operations.

Sample Input

Copy sample input to clipboard

```
5 6
1
2
3
4
5
1 6
2 2
1 7
1 8
2 3
1 9
```

Sample Output

```
6
1
8
7
2
9
3
4
5
```

Hint :注意本题使用 O(n^2)的算法会超时

```
#include <iostream>
#include <list>
using namespace std;

int main(){
	list<int> mylist;
	int n,m,x,y,choose;
	cin>>n>>m;
	for(int i=0;i<n;i++){
		cin>>x;
		mylist.push_back(x);
	}
	list <int>::iterator it=mylist.begin();
	for(int i=0;i<m;i++){
```

```
        cin>>choose;
        switch(choose){
        case 1:
            cin>>y;
            if(it==mylist.begin())
                mylist.push_front(y);
            else
            {
                it--;//     inserting an integer x immediately after the current position p
                mylist.insert(it,y);
            }
            break;
        case 2:
            cin>>y;
            for(int j=0;j<y;j++)
            {
                it++;//    moving after the current position p by k
            }
            break;
        }
    }
    while(!mylist.empty()){
        cout<<mylist.front()<<endl;
        mylist.pop_front();

    }


    return 0;
}
```

1003. 猴子选大王

Description

猴子选大王，有 N 只猴子，从 1～N 进行编号。它们按照编号的顺时针方向，排成一个圆圈，然后从第一只猴子开始报数。第一只猴子报 1，以后每只猴子报的数字都是它前面猴子所报数字加 1。如果一只猴子报的数字是 M，则该猴子出列，下一只猴子重新从 1 开始报数。剩下的猴子继续排成一个圆圈报数，直到全部的猴子都出列为止。最后一个出列的猴子胜出。请用链式结构来实现。

Input

The first line is an integer t, indicating the number of test cases. Then there are t lines and each line contains two positive integer N(0<N<=100) and M(0<M<=100).

Output

For each test case, print out the number of the Monkey King.

Sample Input

Copy sample input to clipboard

```
2
5 2
4 3
```

Sample Output

```
3
```

```
#include<iostream>
using namespace std;
struct node
{
	int num;
	node *next;
};
int main(){
	int t,n,m;
	cin>>t;
	while(t--){
		cin>>n>>m;
		if(m==1){
			cout<<n<<endl;
			continue;
		}

			node *head=new node;//储存头指针的位置
			node *p=head;
			p->num=1;
			p->next=NULL;
			for(int i=1;i<n;i++){
				node *temp=new node;
				temp->num=i+1;
				temp->next=NULL;
				p->next=temp;
```

```
                p=temp;
            }
            p->next=head;
            int temp=1;
        node *circle=head;
        while(circle->next!=circle)
        {
            if((temp+1)==m)
            {
                circle->next=circle->next->next;
                circle=circle->next;
                temp=1;
            }
            else
            {
                circle=circle->next;
                temp=temp+1;
            }
        }
        cout<<circle->num<<endl;
    }
    return 0;
}
```

Hw5 、Lists and strings

1000. Insert for single link list with head node

Description

带虚拟头结点的单链表结点结构如下：

```
struct ListNode
{
  int data;
  ListNode *next;
};
```

链表类接口如下：

```
class List
{
public:
  List()
  {
    head = new ListNode;
    head->next = NULL;
  }
```

```
~List()
{
  ListNode* curNode;
  while( head->next )
  {
    curNode = head->next;
    head->next = curNode->next;
    delete curNode;
  }
  delete head;
}
```

```
//在链表第 pos(pos>0)个结点之前插入新结点，新结点的值为 toadd
   //链表实际结点从 1 开始计数。
   //调用时需保证 pos 小等于链表实际结点数
    void List::insert(int toadd, int pos);
  // Data field
  ListNode *head; //head 指向虚拟头结点，head-next 指向第一个实际结点
};

 //删除链表的第 pos(pos>0)个结点
        //链表实际结点从 1 开始计数。
        //调用时需保证 pos 小等于链表实际结点数
        void remove(int pos);

      // Data field
      ListNode *head; //head 指向虚拟头结点，head-next 指向第一个实际结点
};
```

请实现如下函数：

void List::remove(int pos)

void List::insert(int toadd, int pos)

只提交 insert 函数实现和 remove 函数实现，不要提交类定义及 main 函数。

```
void List::insert(int toadd, int pos){
        ListNode* newNode;
        ListNode* curNode=head;
        ListNode* tempNode;
        newNode=new ListNode;
        for (int i=1;i<pos;i++){
           curNode=curNode->next;
        }
        newNode->data=toadd;
        tempNode = curNode->next;
        curNode->next=newNode;
        newNode->next=tempNode;
}
```

```
void List::remove(int pos){
    ListNode *curNode;
    ListNode *tempNode;
    curNode=head;
    for (int i=1;i<pos;i++){
        curNode=curNode->next;
        }
    tempNode=curNode->next;
    curNode->next=tempNode->next;
}
```

1002. 双栈排序

Description

Tom 最近在研究一个有趣的排序问题。如图所示，通过 2 个栈 S1 和 S2，Tom 希望借助以下 4 种操作实现将输入序列升序排序。

操作 a

如果输入序列不为空，将第一个元素压入栈 S1

操作 b

如果栈 S1 不为空，将 S1 栈顶元素弹出至输出序列

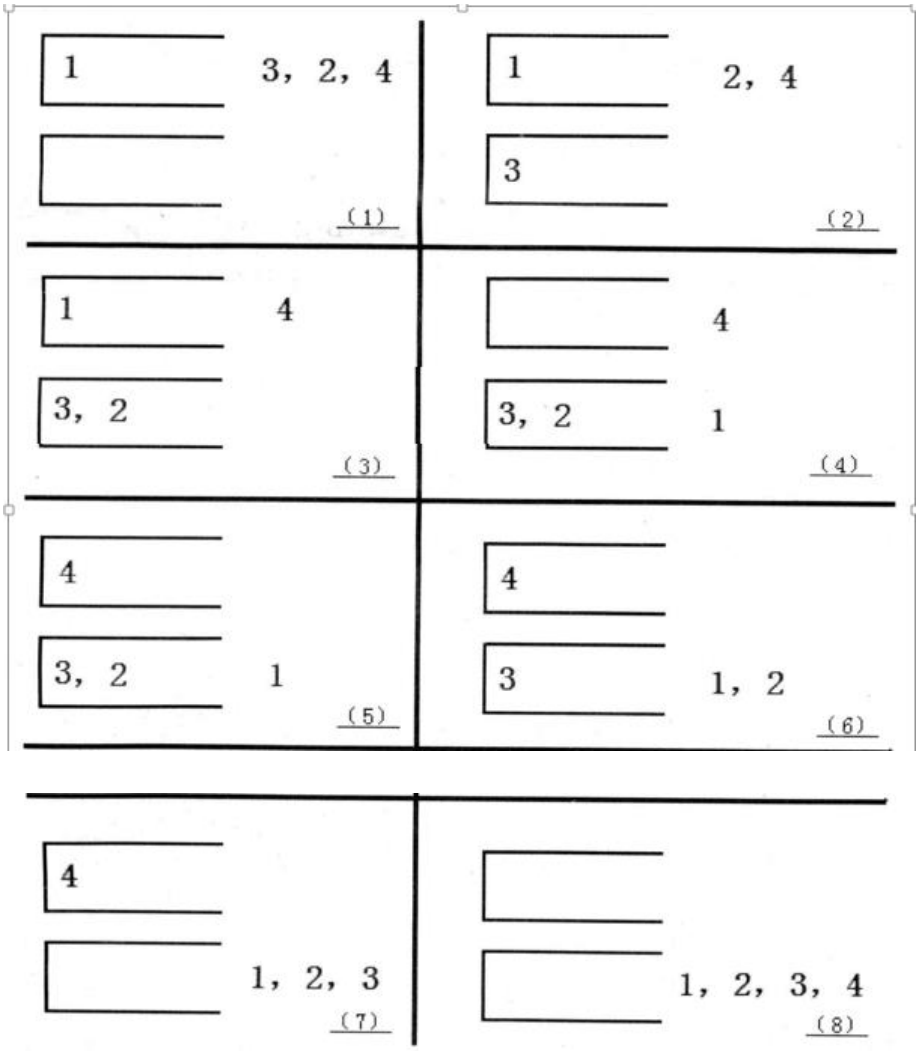
操作 c

如果输入序列不为空，将第一个元素压入栈 S2

操作 d

如果栈 S2 不为空，将 S2 栈顶元素弹出至输出序列

如果一个 1~n 的排列 P 可以通过一系列操作使得输出序列为 1，2，…，(n-1)，n，Tom 就称 P 是一个“可双栈排序排列”。例如(1,3,2,4)就是一个“可双栈排序序列”，而(2,3,4,1)不是。下图描述了一个将(1,3,2,4)排序的操作序列：<a,c,c,b,a,d,d,b>



当然，这样的操作序列有可能有几个，对于上例(1,3,2,4)，<a,c,c,b,a,d,d,b>是另外一个可行的操作序列。Tom 希望知道其中字典序最小的操作序列是什么。

Input

输入有多组 Case,每个 Case 第一行是一个整数 n(n<=1000)。
第二行有 n 个用空格隔开的正整数，构成一个 1~n 的排列。

Output

每组 Case 输出一行，如果输入的排列不是“可双栈排序排列”，输出数字 0；否则输出字典序最小的操作序列，每两个操作之间用空格隔开，行尾没有空格。

Sample Input

Copy sample input to clipboard

```
4
1 3 2 4
4
2 3 4 1
```

Sample Output

```
a b a a b b a b
```

```cpp
#include<iostream>
#include<cstring>
#include<algorithm>
#include<stack>
#include<cstdio>
#include<cstdlib>  // for the exit() function:关闭所有文件，终止正在执行的程序
using namespace std;

int n; //序列元素数目
bool edge[1001][1001] ;//  由于n<=1000，用于构边
int s[1001],F[1001];
int color[1001];//1和2两种颜色,还有一种是没染色，共三种情况
stack<int> s1,s2;
bool ans;
```

```cpp
void dfs(int x,int c)  //对一个有向图进行染色 ,相邻两个点的颜色不同  x是遍历到的节点， c是这个节点的颜色
{
    color[x] = c;
    for (int j=1;j<=n;j++)
    {
        //  edge[x][j]是指从到j节点连成的边 ,节点是一个数据f
        if(edge[x][j])//  两个节点之间若存在边
        {
            //按编号递增的顺序从每个未染色的顶点开始染色,相邻的顶点染上不同的色,如果发生冲突,则是无解的
            if(color[j]==c)
            {
                ans=false;
                break;
            }
            if(!color[j])//节点未被染色时，要染成跟相邻节点不同的颜色，即为3-c
                dfs(j,3-c);
        }
    }
}
```

```
int main()
{
    while(cin>>n)
    {
        ans=true;
        //void *memset(void *s,int c,size_t n)   总的作用：将已开辟内存空间 s 的首 n 个字节的值设为值 c
        memset(F,0,sizeof(F));//每次循环开始时清零
        memset(color,0,sizeof(color));
        memset(edge,0,sizeof(edge));

        for(int i=1;i<=n;i++)
            cin>>s[i];
        F[n+1]=0x7fffffff;// 边界

        for(int i=n;i>=1;i--)//状态转移方程
        {
            F[i]=min(s[i],F[i+1]);
        }

        for(int i=1;i<=n;i++)//结论P: S[i],S[j]两个元素不能进入同一个栈 <=> 存在k,满足i<j<k,使得S[k]<S[i]<S[j]
        {                          //判断数对(i,j)是否满足P,只需判断(S[i]<S[j] 并且 F[j+1]<S[i])即可
            for(int j=i+1;j<=n;j++)
            {
                if(s[i]<s[j] && F[j+1]<s[i])
                    edge[i][j]=edge[j][i]=true;
            }
        }
```

```
         for(int i=1;i<=n;i++)//     深度染色
         {
             if(!color[i])//对未染色的点进行染色
                 dfs(i,1);
         }

         if(!ans)
             cout<<0<<endl;
         else
         {
             int aim = 1;
             int count=0;
             for (int i = 1; i <= n;++i)
             {
                 if (color[i]==1)
                 {
                     s1.push(s[i]);
                     cout<<"a ";
                 }
                 else
                 {
                     s2.push(s[i]);
                     cout<<"c ";
                 }
```

```
                while (!s1.empty() && s1.top() == aim || !s2.empty() && s2.top()== aim)
                {
                    if (!s1.empty() && s1.top()== aim)
                    {
                        s1.pop();
                        count++;
                    //  cout<<count<<endl;
                        if(count==n)
                            cout<<"b"<<endl;
                        else
                            cout<<"b ";
                    }
                    else
                    {
                        s2.pop();
                        count++;
                        //cout<<count<<endl;
                        if(count==n)
                            cout<<"d"<<endl;
                        else
                            cout<<"d ";
                    }
                    aim ++;
                }
            }
        }

    }

    return 0;
}
```

```
/*
    分析条件，我们把问题抽象为数学模型。设输入序列为S，考虑S[i],S[j]两个元素不能进入同一个栈的条件.
    注意,这里所说的"S[i],S[j]两个元素不能进入同一个栈",不是说仅仅不能同时在一个栈中,而是自始至终不
    能进入一个栈,即如果有解,那么S[i],S[j]一定进入过的栈不同.

    结论P: S[i],S[j]两个元素不能进入同一个栈 <=> 存在k,满足i<j<k,使得S[k]<S[i]<S[j].
    证明略过,请参考sqybi.尝试后可以发现结论P是正确的.

    把每个元素按照输入序列中的顺序编号,看作一个图中的每个顶点.这时,我们对所有的(i,j)满足i<j,
    判断是否满足结论P,即S[i],S[j]两个元素能否进入同一个栈.如果满足P,则在i,j之间连接一条边.

    我们对图染色,由于只有两个栈,我们得到的图必须是二分图才能满足条件.由于要求字典序最小,即
    尽量要进入栈1,我们按编号递增的顺序从每个未染色的顶点开始染色,相邻的顶点染上不同的色,如
    果发生冲突,则是无解的.否则我们可以得到每个顶点颜色,即应该进入的栈.

    接下来就是输出序列了,知道了每个元素的决策,直接模拟了.

    在判断数对(i,j)是否满足P时,枚举检查是否存在k的时间复杂度是O(n),则总的时间复杂度是O(n^3),
    对于n=1000是太大了.这原因在于过多得枚举了k,我们可以用动态规划把枚举k变为O(1)的算法.

    设F[i]为Min{S[i],S[i+1],S[i+2]..S[n-1],S[n]},状态转移方程为F[i]=Min{ S[i] , F[i+1] }.
    边界为F[N+1]=极大的值.

    判断数对(i,j)是否满足P,只需判断(S[i]<S[j] 并且 F[j+1]<S[i])即可.时间复杂度为O(n^2).

*/
```

1003. Radix Sort

Description

In this exercise, you need to implement a radix sort to sort *n* elements, where each element is no greater than 100000.

You should not use sort() or qsort() function in your program.

Input

There are multiple cases.
For each case, the first line is an integer *n*. The next line contains *a[i]* (*1<=i<=n, 1<=a[i]<=100000*) to be sorted.

Output

For each case, output the sorted array.

Sample Input

Copy sample input to clipboard

```
3
100 300 200
```

Sample Output ： 100 200 300

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <queue>
using namespace std;
//vec用来存放 待排序的数字，n为待排序数字的个数，max为最大值
void radixSort(vector <int> &vec,int n,int max){
    int x;
    int k=0,p=0;
    queue <int> bucket[10];
    for (int i=1;i<=max;i*=10){//根据最大数字判断需要求多少位数

        for(int j=0;j<n;j++){
            x=(vec[j]/(int)pow(10,p))%10;//求某位数上的数字
            bucket[x].push(vec[j]);
        }
        p++;
        k=0;

        for (int j=0;j<10;j++){

            while(!bucket[j].empty()){
                vec[k]=bucket[j].front();
                bucket[j].pop();
                k++;
            }
        }
    }
}
int main(){
    int n,x,max;
    vector <int> num;
    while(cin>>n){
        for(int i=0;i<n;i++){
        cin>>x;
        num.push_back(x);
        if(i==0) max=x;
        else {
            if(max<x){
                max=x;
            }
        }
    }
    radixSort(num,n,max);
    for (int i=0;i<n-1;i++){
        cout<<num[i]<<" ";
    }
    cout<<num[n-1]<<endl;
    num.clear();
    }

    return 0;
}
```

1004. Message Flood

Description

Well, how do you feel about mobile phone? Your answer would probably be something like that “It’s so convenient and benefits people a lot”. However, if you ask Merlin this question on the New Year’s Eve, he will definitely answer “What a trouble! I have to keep my fingers moving on the phone the whole night, because I have so many greeting messages to send! ”. Yes, Merlin has such a long name list of his friends, and he would like to send a greeting message to each of them. What’s worse, Merlin has another long name list of senders that have sent message to him, and he doesn’t want to send another message to bother them (Merlin is so polite that he always replies each message he receives immediately). So, before he begins to send messages, he

needs to figure to how many friends are left to be sent. Please write a program to help him.
Here is something that you should note. First, Merlin's friend list is not ordered, and each name is alphabetic strings and case insensitive. These names are guaranteed to be not duplicated. Second, some senders may send more than one message to Merlin, therefore the sender list may be duplicated. Third, Merlin is known by so many people, that's why some message senders are even not included in his friend list.

Input

There are multiple test cases. In each case, at the first line there are two numbers n and m (1<=n, m<=20000), which is the number of friends and the number of messages he has received. And then there are n lines of alphabetic strings (the length of each will be less than 10), indicating the names of Merlin's friends, one per line. After that there are m lines of alphabetic strings, which are the names of message senders.

The input is terminated by n=0.

Output

For each case, print one integer in one line which indicates the number of left friends he must send.

Sample Input

Copy sample input to clipboard

```
5 3
Inkfish
Henry
Carp
Max
Jericho
Carp
Max
Carp
0
```

Sample Output : 3

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <cstring>
using namespace std;

char* list[40000][9]={0};
char flag[40000]={0};

int main(){
    int i,j,num,m,n,index;
    char *p;
    while(1){
        scanf("%d",&m);
        if(m==0) break;
        scanf("%d\n",&n);
        char  f[m][11], sender[n][11] ;

        for(i=0; i<m; i++){
            index=0;
            gets( f[i] );

            p = f[i];
            while(*p){
                *p = tolower(*p);
                index += (*p-89) * (*p-2);
                p++;
            }

            list[index][ flag[index] ] = f[i];
            ++flag[index];
        }

        for(i=0; i<n; i++){
            scanf("%s", sender[i]);
        }
        num=0;
        for(i=0; i<n; i++){
            index = 0;
            p = sender[i];
            while(*p){
                *p = tolower(*p);
                index += (*p-89) * (*p-2);
                p++;
            }
            if(flag[index] == 0) continue;
            for(j=0; j<flag[index]; j++){
                if( strcmp(sender[i], list[index][j])==0 ){
                    num++;
                    list[index][j][0] = '\0';
                }
            }
        }

        printf("%d\n", m-num);

        memset(flag, 0, sizeof(flag));

    }

    return 0;
}
```

Lab5 、 Lists and strings

1000. Loops in the Linked List

Description

给定链表结点类型定义，要求实现 check 函数返回对于给定链表是否存在环，注意链表的边界情况。

```
struct node{
    node *next;
    int val;
};

bool check(node *head){

//here

}
```

Hint

只需要提交 check 函数实现，不需要提交 main 函数

```
//判断给定链表是否存在环
bool check(node *head){
      node *fast=head;
      node *slow=head;
      while (fast && fast->next)
      {
            slow=slow->next;
            fast=fast->next->next;
            if(fast==slow)
            return true;
      }
      return false;

}
```

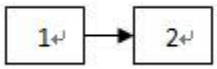
1001. Delete Duplicate

Description

已知线性表中的元素以递增有序排列，并以单链表作存储结构。试写一个高效的算法，删除表中所有冗余的结点，即数据域相同的结点只保留一个。

例如对链表：

进行操作，将得到：



链表结点定义如下：

```
struct LinkNode {

        int data;

        LinkNode *next;

      LinkNode(int d, LinkNode *add_on = NULL) {

                  data = d;

                  next = add_on;
```

```
    }
};
typedef LinkNode *LinkList;
```

请实现函数：

```
void delete_duplicate(LinkList &head);
```

注意内存的回收。

Hint

只需提交 delete_duplicate() 函数

```
void delete_duplicate(LinkList &head){
    LinkNode *curNode=head;
    LinkNode *tempNode;
    while (curNode!=NULL&&curNode->next!=NULL){
        tempNode=curNode->next;
        if(tempNode->data==curNode->data){
            curNode->next=tempNode->next;
            delete tempNode;
        }
        else {
            curNode=tempNode;
        }

    }
}
```

1002. Candy Sharing Game

Description

A number of students sit in a circle facing their teacher in the center. Each student initially has an even number of pieces of candy. When the teacher blows a whistle, each student simultaneously gives half of his or her candy to the neighbor on the right. Any student, who ends up with an odd number of pieces of candy, is given another piece by the teacher. The game ends when all students have the same number of pieces of candy.

Write a program which determines the number of times the teacher blows the whistle and the final number of pieces of candy for each student from the amount of candy each child starts with.

Input

The input may describe more than one game. For each game, the input begins with the number N of students, followed by N (even) candy counts for the children counter-clockwise

around the circle. The input ends with a student count of 0. Each input number is on a line by itself.

Output

For each game, output the number of rounds of the game followed by the amount of candy each child ends up with, both on one line.

Sample Input

Copy sample input to clipboard

```
6
36
2
2
2
2
2
11
22
20
18
16
14
12
10
8
6
4
2
4
2
4
6
8
0
```

Sample Output

```
15 14
17 22
4 8
Notes:
The game ends in a finite number of steps because:
```

```
1. The maximum candy count can never increase.
2. The minimum candy count can never decrease.
3. No one with more than the minimum amount will ever decrease to the minimum.
```

```cpp
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;

int stu[100000], rest[100000];

bool sharing(int n, int sum, int a[]) {
    int average = sum / n;
    for (int i = 0; i < n; i++) {
        if (a[i] != average) return false;
    }
    return true;
}

int main() {
    int n;

    while (cin >> n) {
        stu[100000] = {0};
        rest[100000] = {0};
        int count = 0, sum = 0;
        bool continue_ = true;

        if (n == 0) break;

        else {
            for (int i = 0; i < n; i++) {
                cin >> stu[i];
                sum += stu[i];
            }

            while (continue_) {
                for (int i = 0; i < n; i++) rest[i] = stu[i] / 2;

                stu[0] = rest[0] + stu[n-1] / 2;
                for (int i = 1; i < n; i++) {
                    stu[i] = stu[i] / 2 + rest[i-1];
                }

                for (int i = 0; i < n; i++) {
                    if (stu[i] % 2 != 0) {
                        stu[i]++;
                        sum++;
                    }
                }

                count++;

                if (sharing (n, sum, stu) == true) continue_ = false;
                else continue_ = true;
            }

            cout << count << " " << stu[0] << endl;
        }
    }

    return 0;

}
```

Lab6、Searching

1000. 范围统计

Description

Given a list of N numbers and Q queries,for each query given two numbers a,b.You are asked to count how many numbers in the list that are in the range [a,b].

Input

Input contains multiple testcases.

The first line of input is an Integer T(T <= 10), the number of testcase.
The second line is an integer N(N <= 100000),the number in the list.
Then N integers followed.
The next line is an integer Q(Q <= 10000),the number of query.

For each query there are two interger a,b(a <= b) -- the lower bound and the upper bound accordingly

Output

For each query ,output one integer that represent how many numbers are in the range [a,b] .

Sample Input

Copy sample input to clipboard

```
1
10
5 2 4 1 3 9 7 6 8 10
5
4 6
2 7
0 5
-100 100
-100 0
```

Sample Output

```
3
6
5
10
0
```

Hint

Use std::lower_bound() and std::upper_bound()

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    int n,t,q,x,a,b;
    vector<int>vec;
    cin>>t;
    while(t--){
        cin>>n;
        for(int i=0;i<n;i++){
            cin>>x;
            vec.push_back(x);
        }
        sort(vec.begin(),vec.end());
        cin>>q;
        for(int i=0;i<q;i++){
            cin>>a>>b;
            int count=0;
            vector<int>::iterator iter,low,up;
            low=std::lower_bound (vec.begin(), vec.end(), a);
            up = std::upper_bound (vec.begin(), vec.end(), b);
            /*for(iter=vec.begin();iter!=vec.end();iter++){
                if(iter>=low&&iter<up){
                    count++;
                }
            }*/
            cout<<up-low<<endl;
        }
      vec.clear();
    }
    return 0;
}
```

1001. Binary Search

Description

实现二分查找函数，函数接口如下。

```
  /* size 为数组 s 的实际大小。
假定 s 非递减有序，如果 s 中存在值为 target 的元素，
则返回最后一次出现的位序号，否则返回-1 表示不存在。
位序号从 0 开始计。*/
int binSearch(const int s[], const int size, const int target)
{
// 请将实现代码添加在这里
}
```

提交时只需提交上述函数，不要提交 main()函数。

调用例子：

```
int s[8] = {0,1,1,3,3,3,6,6};

cout << binSearch(s,8,3) << endl;    //输出 5

cout << binSearch(s,8,4) << endl;    //输出-1
```

Hint

不允许使用 STL 库里面的相关函数和库（否则可能会出现编译错误）

包括 iostream, map, vector, set, algorithm, queue 等

```
int binSearch(const int s[], const int size, const int target) {
    int bottom = 0, top = size - 1;
    while (bottom < top) {
        int mid = (bottom + top) / 2;
        if (s[mid] < target) bottom = mid + 1;
        else top = mid;
    }

    if (top < bottom) return -1;

    else {
        if (s[bottom] == target) {
            while (s[bottom] == s[bottom+1]) bottom++;
            return bottom;
        }
        else return -1;
    }
}
```

1002. 最大值最小化

Description

把一个包含 n 个正整数的序列划分成 m 个连续的子序列（每个正整数恰好属于一个序列）。设第 i 个序列的各数之和为 S(i)，如何让所有 S(i)的最大值尽量小?

例如序列 1 2 3 2 5 4，划分成 3 个序列的最优方案为 1 2 3 | 2 5 | 4，其中 S(1)=6, S(2)=7, S(3)=4，最大值为 7；如果划分成 1 2 | 3 2 | 5 4，则最大值为 9，不如刚才的好。

$n \le 10^6$，所有数之和不超过 10^9。

Input

There may be multiple cases.

The first line of the input gives two integer n and m. The next line contains n integers.

Output

For each case, output the minimum value.

Sample Input

Copy sample input to clipboard

```
6 3
1 2 3 2 5 4
```

Sample Output

```
7
```

```cpp
#include <iostream>
#include <cstdio>
using namespace std;

const int N = 100010;

int a[N];
int n, m;
long long sum;

bool check (long long key) {
    int cnt = 0;//记录划分线个数。
    long long s = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] > key) return false;
        if (s + a[i] > key) {
            cnt++;
            s = 0;
        }
        s += a[i];
    }
    if (cnt <= m-1) return true;
    else return false;
}

void init () {
    sum = 0;
    for (int i = 0; i < n; i++) {
        scanf ("%d", &a[i]);
        sum += a[i];
    }
}

void solve () {
    long long left = 0, right = sum;

    while (left < right) {
        long long mid = (left + right) >> 1;
        if (check (mid)) right = mid;
        else left = mid + 1;
    }

    printf ("%lld\n", left);
}

int main () {
    while (~scanf("%d%d", &n, &m)) {
        init();
        solve();
    }

    return 0;

}
```

1003. 方程求解

Description

已知函数 y = e^x+ln(x)-1，实现函数

long double solve(long double y)

{

// here

}

对于传入的 y，返回 x 值

要求 f(x)与 y 的误差小于 1e-6，其中 0 < y < 1e10

Hint

只需要提交 solve 函数实现

```cpp
#include <cstdio>
#include <cmath>
using namespace std;

const long double e = 2.718281828459;

long double f(long double x, const long double y) {
    return pow(e, x) + log(x) / log(e) - 1 - y;
}

long double solve (long double y) {
    long double l = 0, r = pow (10, 20), mid = (l+r) / 2;
    while (abs(f(mid, y)) >= pow(10, -10)) {
        if (f(mid, y) < 0) l = mid;
        else r = mid;
        mid = (l+r) / 2;
    }

    if (abs(f(mid, y)) < pow (10, -10)) return mid;
    else return -1;
}
```

Hw7、Sorting

1000. mergesort for list

Description

Please use mergesort to sort a linked list of data.

The linked list is defined as follows.

struct linkedlist{

    int data;

    linkedlist *next;

};

Please implement the following function.

//sort the list by mergesort and return the head of it

void mergesort(linkedlist *&head, int len){

  //add your code here

}

Hint

  1->3->2->4->NULL

  ^

  |

Head

```cpp
#include<iostream>
#include<cstdio>
using namespace std;
struct linkedlist {
    int data;
    linkedlist *next;
};

//sort the list by mergesort and return the head of it
linkedlist *merge(linkedlist *first, linkedlist *second) {
    linkedlist *last_sorted;
    linkedlist combined;
    last_sorted=&combined;
    while (first != NULL && second != NULL) {
        if (first->data <= second->data) {
            last_sorted->next = first;
            last_sorted = first;
            first = first->next;
        } else {
            last_sorted->next = second;
            last_sorted = second;
            second = second->next;
        }
    }
    if(first == NULL)
        last_sorted->next = second;

    if (second == NULL)
        last_sorted->next = first;
    return combined.next;
}
```

```cpp
linkedlist *devide(linkedlist *sublist){
    linkedlist *pos,*mid,*second;
    if((mid=sublist)==NULL)return NULL;
    pos=mid->next;
    while(pos!=NULL){
        pos=pos->next;
        if(pos!=NULL){
            mid=mid->next;
            pos=pos->next;
        }
    }
    second=mid->next;
    mid->next=NULL;
    return second;
}
void mergesort(linkedlist *&head, int len) {
    if(head!=NULL&&head->next!=NULL){
        linkedlist *second=devide(head);
        mergesort(head,1);
        mergesort(second,1);
        head=merge(head,second);
    }
}
```

1001. 明明的随机数

Description

明明想在学校中请一些同学一起做一项问卷调查，为了实验的客观性，他先用计算机生成了 N 个 1 到 1000 之间的随机整数（N≤100），对于其中重复的数字，只保留一个，把其余相同的数去掉，不同的数对应着不同的学生的学号。然后再把这些数从小到大排序，按照排好的顺序去找同学做调查。请你协助明明完成“去重”与“排序”的工作。

Input

输入包含多个测试数据。

每个测试数据有 2 行，第 1 行为 1 个正整数，表示所生成的随机数的个数 N，第 2 行有 N 个用空格隔开的正整数，为所产生的随机数。

Output

对每个测试数据输出 2 行。第 1 行为 1 个正整数 M，表示不相同的随机数的个数。第 2 行为 M 个用空格隔开的正整数，为从小到大排好序的不相同的随机数。

Sample Input

Copy sample input to clipboard

```
10
20 40 32 67 40 20 89 300 400 15
3
3 2 1
```

Sample Output

```
8
15 20 32 40 67 89 300 400
3
1 2 3
```

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cstdio>
#include <vector>
#include <algorithm>
//利用vector进行排序和去重
using namespace std;
int main(){
    int n,x;
    //srand(unsigned(time(0)));
    vector <int> vec;
    while (cin>>n){
        for(int i=0;i<n;i++){
            cin>>x;
            vec.push_back(x);
        }
        sort(vec.begin(),vec.end());
        int now = 1;
        for(int i = 1; i < vec.size(); i++){
            if (vec[i] == vec[i - 1]) continue;
            vec[now++] = vec[i];
        }
        vec.resize(now);
        int len=vec.size();
        cout<<len<<endl;
        for (int i=0;i<len-1;i++){
            cout<<vec[i]<<" ";
        }
        cout<<vec[len-1]<<endl;
        vec.clear();
    }
    return 0;
}
```

1002. 奖学金

Description

某小学最近得到了一笔赞助，打算拿出其中一部分为学习成绩优秀的前 5 名学生发奖学金。期末，每个学生都有 3 门课的成绩：语文、数学、英语。先按总分从高到低排序，如果两个同学总分相同，再按语文成绩从高到低排序，如果两个同学总分和语文成绩都相同，那么规定学号小的同学排在前面，这样，每个学生的排序是唯一确定的。

任务：先根据输入的 3 门课的成绩计算总分，然后按上述规则排序，最后按排名顺序输出前 5 名学生的学号和总分。注意，在前 5 名同学中，每个人的奖学金都不相同，因此，你必须严格按上述规则排序。例如，在某个正确答案中，如果前两行的输出数据（每行输出两个数：学号、总分）是：

7 279

5 279

这两行数据的含义是：总分最高的两个同学的学号依次是 7 号、5 号。这两名同学的总分都是 279（总分等于输入的语文、数学、英语三科成绩之和），但学号为 7 的学生语文成绩更高一些。如果你的前两名的输出数据是：

5 279

7 279

则按输出错误处理，不能得分。

Input

输入包含多组测试数据，每个测试数据有 n+1 行。
第 1 行为一个正整数 n，表示该校参加评选的学生人数。
第 2 到 n+1 行，每行有 3 个用空格隔开的数字，每个数字都在 0 到 100 之间。第 j 行的 3 个数字依次表示学号为 j-1 的学生的语文、数学、英语的成绩。每个学生的学号按照输入顺序编号为 1~n（恰好是输入数据的行号减 1）。
所给的数据都是正确的，不必检验。

Output

对于每个测试数据输出 5 行，每行是两个用空格隔开的正整数，依次表示前 5 名学生的学号和总分。两个相邻测试数据间用一个空行隔开。

Sample Input

Copy sample input to clipboard

```
6
90 67 80
87 66 91
78 89 91
88 99 77
67 89 64
78 89 98
8
80 89 89
88 98 78
90 67 80
87 66 91
78 89 91
88 99 77
67 89 64
78 89 98
```

Sample Output

```
6 265
4 264
3 258
2 244
1 237

8 265
2 264
```

```
6 264
1 258
5 258
```

```cpp
#include <iostream>
#include <algorithm>

using namespace std;
struct Student{
    int chinese;
    int math;
    int english;
    int grade;
    int id;
};

Student student[10000];

bool cmp(Student a, Student b){
    if (a.grade > b.grade) return true;
    if (a.grade < b.grade) return false;
    if (a.chinese > b.chinese) return true;
    if (a.chinese < b.chinese) return false;
    return a.id < b.id;
}
```

```cpp
int main(){
    int n;
    bool first = true;
    while (cin>>n){
        if (first) first = false;
        else cout << endl;
        for(int i=0;i<n;i++){
            cin>>student[i].chinese>>student[i].math>>student[i].english;
            student[i].id=i+1;
            student[i].grade=student[i].chinese+student[i].math+student[i].english;
        }
        sort(student, student + n, cmp);
        for(int i=0;i<5;i++){
            cout<<student[i].id<<" "<<student[i].grade<<endl;
        }
    }
    return 0;
}
```

Lab7、Sorting

1000. Inversion Number

Description

Let A(1), ..., A(n) be a sequence of n numbers. If i<j and A(i)>A(j), then the pair (i,j) is called an inversion pair.

The inversion number of a sequence is one common measure of its sortedness. Given the sequence A, calculate its inversion number.

Input

There are multiple cases.

Each case contains an integer n (n<=100,000) followed by A(1) , ..., A(n).

Output

For each case, output the inversion number.

Sample Input

Copy sample input to clipboard

```
5
3 1 4 5 2
```

Sample Output

```
4
```

Hint

The answer may exceed 2^31.

```cpp
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <string.h>
using namespace std;

    /* 归并求逆序对数，arr 存储最终有序结果
     * 在函数外申请一个临时数组作为参数传入
     * 避免递归不断创建临时数组的开销
     */
int arra[100001];

long long Merge(int * arr, int beg, int mid, int end, int * tmp_arr)  {
    memcpy(tmp_arr+beg,arr+beg,sizeof(int)*(end-beg+1));
    int i = beg;
    int j = mid + 1;
    int k = beg;
    long long inversion = 0;  // 合并过程中的逆序数
    while(i <= mid && j <= end)  {
        if(tmp_arr[i] <= tmp_arr[j])  {
            arr[k++] = tmp_arr[i++];
        }
        else  {
            arr[k++] = tmp_arr[j++];
            inversion += (mid - i + 1);
        }
    }
    while(i <= mid)  {
        arr[k++] = tmp_arr[i++];
    }
    while(j <= end) {
        arr[k++] = tmp_arr[j++];
    }
    return inversion;
}
```

```cpp

long long MergeInversion(int * arr, int beg, int end, int * tmp_arr)  {
    long long inversions = 0;    // 记录倒序数
    if(beg < end)  {
        int mid = (beg + end) >> 1;
        inversions += MergeInversion(arr, beg, mid, tmp_arr);
        inversions += MergeInversion(arr, mid+1, end, tmp_arr);
        inversions += Merge(arr, beg, mid, end, tmp_arr);
    }
    return inversions;
}

int main() {
    int array[100001];
    int N;

    while (cin >> N) {
        for (int i = 0; i < N; i++) cin >> arra[i];
        memcpy(array, arra, sizeof arra);
        printf("%lld\n", MergeInversion(arra,0,N-1,array));
    }

    return 0;
}
```

1001. Multi-key Sorting

Description

Consider a table with rows and columns. The columns are numbered from ***1*** to ***C***. For simplicity's sake, the items in the table are strings consisting of lower case letters.

1	2	3
apple	red	sweet
apple	green	sour
pear	green	sweet
banana	yellow	sweet
banana	brown	rotten

Table 1

1	2	3
banana	brown	rotten
apple	green	sour
pear	green	sweet
apple	red	sweet
banana	yellow	sweet

Table 2

1	2	3
apple	green	sour
apple	red	sweet
banana	brown	rotten
banana	yellow	sweet
pear	green	sweet

Table 3

You are given the operation Sort(k) on such tables: Sort(k) sorts the rows of a table in the order of the values in column ***k*** (while the order of the columns does not change). The sort is stable, that is, rows that have equal values in column ***k***, remain in their original order. For example, applying

Sort(2) to Table 1 yields Table 2.
We are interested in sequences of such sort operations. These operations are successively applied to the same table. For example, applying the sequence Sort(2); Sort(1) to Table 1 yields Table 3.
Two sequences of sort operations are called equivalent if, for any table, they have the same effect. For example, Sort(2); Sort(2); Sort(1) is equivalent to Sort(2); Sort(1). However, it is not equivalent to Sort(1); Sort(2), because the effect on Table 1 is different.

Task

Given a sequence of sort operations, determine a shortest equivalent sequence.

Input

注意：输入包含多个测试数据。

The first line of the input contains two integers, ***C*** and ***N***. ***C*** (***1 ≤ C ≤ 1 000 000***) is the number of columns and ***N*** (***1 ≤ N ≤ 3 000 000***) is the number of sort operations. The second line contains ***N*** integers, k_i ***(1 ≤ k_i ≤ C)***. It defines the sequence of sort operations Sort(k_1); ...; Sort(k_N).

Output

The first line of the output contains one integer, ***M***, the length of the shortest sequence of sort operations equivalent to the input sequence (Subtask A). The second line contains exactly ***M*** integers, representing a shortest sequence (Subtask B). You can omit the second line if you solve only Subtask A.

Sample Input

Copy sample input to clipboard

```
4 6
1 2 1 2 3 3
```

Sample Output

```
3
```

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

using namespace std ;

int a[3000005] , b[3000005] ;
int visit[1000005] ;
int main ()
{
  int c , n , k ;
  while(scanf("%d %d" , &c , &n) != EOF)
  {
    for(int i=0 ; i<n ; i++)
      scanf("%d" , &a[i]) ;
    k = 0 ;
    for(int i=n-1 ; i>=0 ; i--)
    {
      if(!visit[a[i]])
      {
        b[k++] = a[i] ;
        visit[a[i]] = 1 ;
      }
    }
    printf("%d\n" , k) ;
    printf("%d" , b[k-1]) ;
    for(int i = k-2 ; i >= 0 ; i--)
      printf(" %d" , b[i]) ;
    printf("\n") ;
    memset(visit , 0 , sizeof(visit)) ;
  }
  return 0 ;
}
```

Lab 8、Sorting

1000. Find the k-th smallest element

Description

Given a sequence A=(a_1,...,a_n), your job is to find the (k+1)-th smallest element of A.

For example, A=(3, 2, 3, 4, 5) and k=1, then 3 is returned, since 3 is the second smallest element of A.

Please implement and submit the following function:

int select(int a[], int n, int k) {

  // Your code will be here. You may invoke other functions.

}

Note: 0<=k<n<=1000000

Hint

算法思想：采用类似快速排序的方法随机选择一个枢纽元进行划分，数组以 p 为界分成两个部分。如果左边部分的数目 i（包括 A[p]）等于 k，则返回 A[p]。否则如果左边部分元素数目小于 k，则递归调用该函数从右边即[p+1, u]选择第 k-i 小元素。如果左边元素数目大于 k，则从范围[l, p-1]中选择第 k 小元素。

事实上，C++ STL 提供 nth_element()的函数实现第 k 小元素查找，但作为练习，希望同学们不要使用该函数。

```
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;
//算法思想：采用类似快速排序的方法随机选择一个枢纽元进行划分，数组以p为界分成两个部分。
//如果左边部分的数目i（包括A[p]）等于k，则返回A[p]。
//否则如果左边部分元素数目小于k，则递归调用该函数从右边即[p+1, u]选择第k-i小元素。
//如果左边元素数目大于k，则从范围[l, p-1]中选择第k小元素.
int partition(int a[],int low, int high) {
    int pivot;
    int i, last_small=low;
    int mid=(low+high)/2;
    swap(a[low],a[mid]);
    pivot = a[low];

    for (i = low+1 ; i <=high; i++) {
        if (a[i] < pivot) {
            last_small = last_small + 1;
            swap(a[last_small],a[i]);
        }
    }
    //cout<<1<<endl;
    swap(a[low], a[last_small]);
    return last_small;
}
void recursive_select(int a[],int low,int high,int k){
    int pivot_pos;

    pivot_pos=partition(a,low,high);
    if(pivot_pos==k) return ;
    else if(pivot_pos<k) recursive_select(a,pivot_pos+1,high,k);
    else recursive_select(a,low,pivot_pos-1,k);


}
int select(int a[], int n, int k) {
    recursive_select(a,0,n-1,k);
    return a[k];
}
```

1001. heap

Description

实现一个小根堆
给定如下数据类型的定义：

```
class array {
    private:
    int elem[MAXN];

    public:
    int &operator[](int i) { return elem[i]; }
};
```

```
class heap {
    private:
    int n;
    array h;

    public:
    void clear() { n = 0; }
    int top() { return h[1]; }
    int size() { return n; }
    void push(int);
    void pop();
};
```

要求实现：

```
void heap::push(int x) {
    // your code
}

void heap::pop() {
    // your code
}
```

Hint

只提交heap::push和heap::pop

```
06. void heap::push(int x)
07. {
08.     h[++n]=x;//把新的数据当做最后一个叶子节点
09.     int k=n;
10.     while(k!=1)
11.     {
12.         if(h[k]<h[k/2])
13.         {
14.             swap(h[k],h[k/2]);
15.             k/=2;
16.         }
17.         else
18.             break;
19.     }
20. }
```

```cpp
void heap::pop()
{
    h[1]=h[n--];//pop是指把root（即h[1]）去掉 ，然后用最后一个叶子节点代替它，再对堆进行调整
    int i=1;
    int ok=0;
    int flag;
    while(i*2<=n && ok==0)
    {
        if(h[i]>h[2*i])
        {
            flag=2*i;
        }
        else
            flag=i;

        if(h[flag]>h[2*i+1] && 2*i+1<=n)
        {
            flag=2*i+1;
        }

        if(flag!=i)
        {
            swap(h[i],h[flag]);
            i=flag;
        }
        else
            ok=1;
    }
}
```

Lab9、Recursion

1000. Permutation Generation[Special judge]

Description

Enumerate all the permutations with size *n* and output them. You can output these permutations in any order.

Input

An integer *n* (*n*<=9).

Output

Output all the permutations with size *n*.

Sample Input

Copy sample input to clipboard

```
3
```

Sample Output

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Hint

Read Section 6.6 of the textbook for your reference.

题目可以用 **printf** 输出，并且注意行末不输出空格

```cpp
#include <iostream>
#include <cstdio>
using namespace std;
//new_entry表示最后插入的数
//degree表示待排列数的总数
void process_permutation(int *permutation){
    int cur=0;
    while(permutation[cur]!=0){
        if(permutation[permutation[cur]]==0){
        printf("%d",permutation[cur]);
        cur=permutation[cur];
        }
        else{
        printf("%d ",permutation[cur]);
        cur=permutation[cur];
        }
    }
    printf("\n");
}
void permute(int new_entry,int degree,int *permutation){
    int cur=0;
    do{
        permutation[new_entry]=permutation[cur];
        permutation[cur]=new_entry;
        if(new_entry==degree)
            process_permutation(permutation);
        else
            permute(new_entry+1,degree,permutation);

        permutation[cur]=permutation[new_entry];
        cur=permutation[cur];
    }while(cur!=0);
}
int main(){
    int degree;
    int max_degree=9;
    int permutation[max_degree+1];
    cin>>degree;
    permutation[0]=0;
    permute(1,degree,permutation);
    return 0;
}
```

1001. Subset Generation I[Special judge]

Description

Generate all the subsets of {A, B, ..., } of *n* elements. There are 2^n subsets in total. You can output them in any order.

Input

An integer *n* (*n*<=9).

Output

Output all the subsets of {A, B, ..., } of *n* elements. In each subset, the elements should be printed from smallest to largest.

Sample Input

Copy sample input to clipboard

```
3
```

Sample Output

```

A
B
AB
C
AC
BC
ABC
```

Hint

The first line in the sample output is a blank line, which corresponds to an empty set.

```cpp
#include<iostream>
#include<cstdio>
using namespace std;

void combination(char *s, int p, int q, bool *flag)
{
    if (p == q)
    {
        for (int i=0; i<q; i++)
        {
            if(flag[i])
                putchar(s[i]);
        }
        puts("");
        return;
    }

    flag[p] = false;
    combination(s,p+1,q, flag);
    flag[p] = true;
    combination(s,p+1,q, flag);
}

int main()
{   int n;
    scanf("%d",&n);
    bool flag[27]={0};
    char str[27];
    for(int i=0;i<n;i++){
        str[i]='A'+i;
    }
    combination(str, 0, n, flag);
    return 0;
}
```

1002. (选做) Subset Generation II

Description

Generate all the subsets of {A, B, ... } of *n* elements. You should guarantee that the output follows the **lexicographical** order.

Input

An integer n. (n<=19)

Output

Output all the subsets in the lexicographical order. In each subset, the elements should be printed from smallest to largest.

Sample Input

Copy sample input to clipboard

```
3
```

Sample Output

```
A
AB
ABC
AC
B
BC
C
```

Hint

The first line in the sample output is a blank line, which corresponds to an empty set.

""<"A"<"AB"<"ABC"<"AC"<"B"<"BC"<"C"

You'd better not invoke a *sort* function.

```cpp
#include <iostream>
#include <string>
#include <cstdio>
using namespace std;

char set[20] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
                'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S'};

void subset(char output[], int character, int current, int max)
{
    output[current] = '\0';
    printf("%s\n", output);
    if(character > max || current > max){
        return;
    }
    for(int i = character; i <= max; i++){
        output[current] = set[i];
        //cout << output << endl;
        subset(output, i + 1, current + 1, max);
    }
}

int main()
{
    int n;
    char output[20];
    cin >> n;
    subset(output, 0, 0, n - 1);
    return 0;
}
```