Lab 10. Hashing

1000. Query

We have a set of strings (set A) and a set of queries (set B). Each query is to check whether a string exists in set A or not. If yes, output this string.

Your task is to implement the following function:

void query(string A[], int n, string B[], int m);

Here, n is the number of strings in set A, m is the number of strings in set B. 1<=n,m<=500,000.

Submit the function only.

Input

No input.

Output

Output the strings in set B which exist in set A. The output should follow the original order of the strings in set B.

Hint

For example,

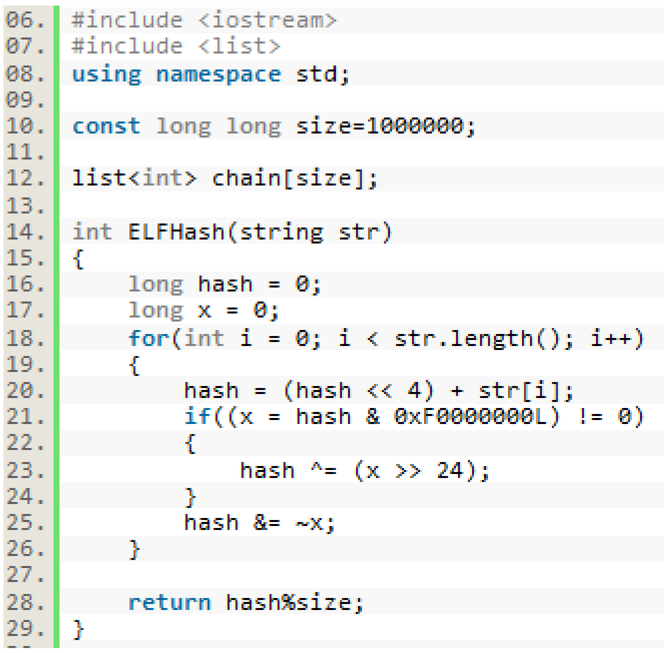
A[0]="ABC", A[1]="CD", A[2]="D"

B[0] = "A", B[1] ="CD", B[2]="BC", B[3]="ABC",

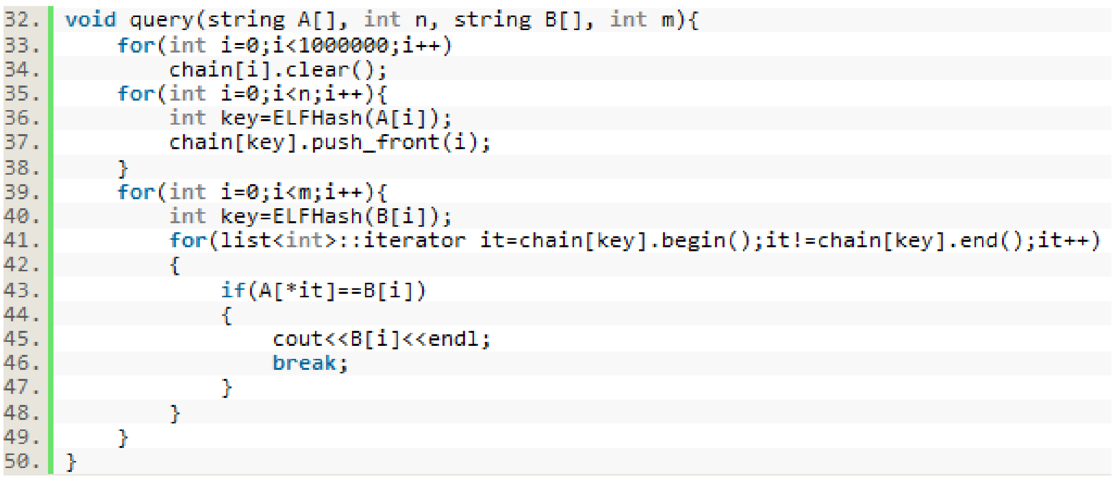
then you should output:

CD

ABC



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

const long long size=1000000;

list<int> chain[size];

int ELFHash(string str)
{
    long hash = 0;
    long x = 0;
    for(int i = 0; i < str.length(); i++)
    {
        hash = (hash << 4) + str[i];
        if((x = hash & 0xF0000000L) != 0)
        {
            hash ^= (x >> 24);
        }
        hash &= ~x;
    }

    return hash%size;
}
```



```cpp
void query(string A[], int n, string B[], int m){
    for(int i=0;i<1000000;i++)
        chain[i].clear();
    for(int i=0;i<n;i++){
        int key=ELFHash(A[i]);
        chain[key].push_front(i);
    }
    for(int i=0;i<m;i++){
        int key=ELFHash(B[i]);
        for(list<int>::iterator it=chain[key].begin();it!=chain[key].end();it++)
        {
            if(A[*it]==B[i])
            {
                cout<<B[i]<<endl;
                break;
            }
        }
    }
}
```

1001. MJ, Nowhere to Hide

Description

On BBS, there is a familiar term called MJ (short for MaJia), which means another BBS ID of one person besides his/her main ID.
These days, a lot of ACMers pour water on the ACMICPC Board of argo. Mr. Guo is very angry about that and he wants to punish these guys. ACMers are all smart boys/girls, right? They usually use their MJs while pouring water, so Mr. Guo can not tell all the IDs apart. Unfortunately, the IP can not be changed, i.e, the posts of main ID and MJ of the same person has the same IP address, meanwhile, the IP addresses of different person is different. Assuming that each person has exactly one main ID and one MJ, by reading their posts on BBS, you then tell Mr. Guo whom each MJ belongs to.

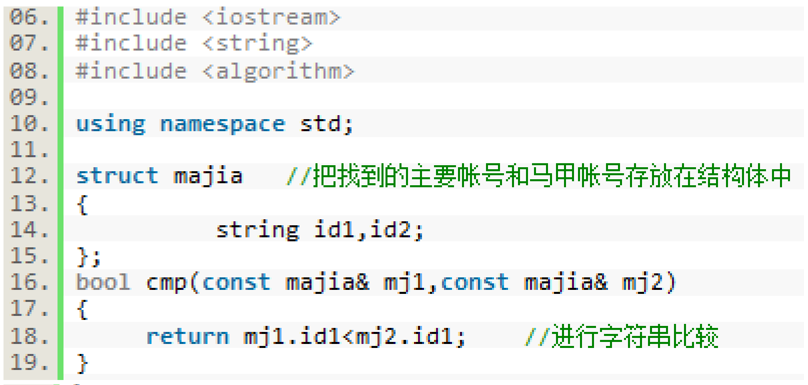
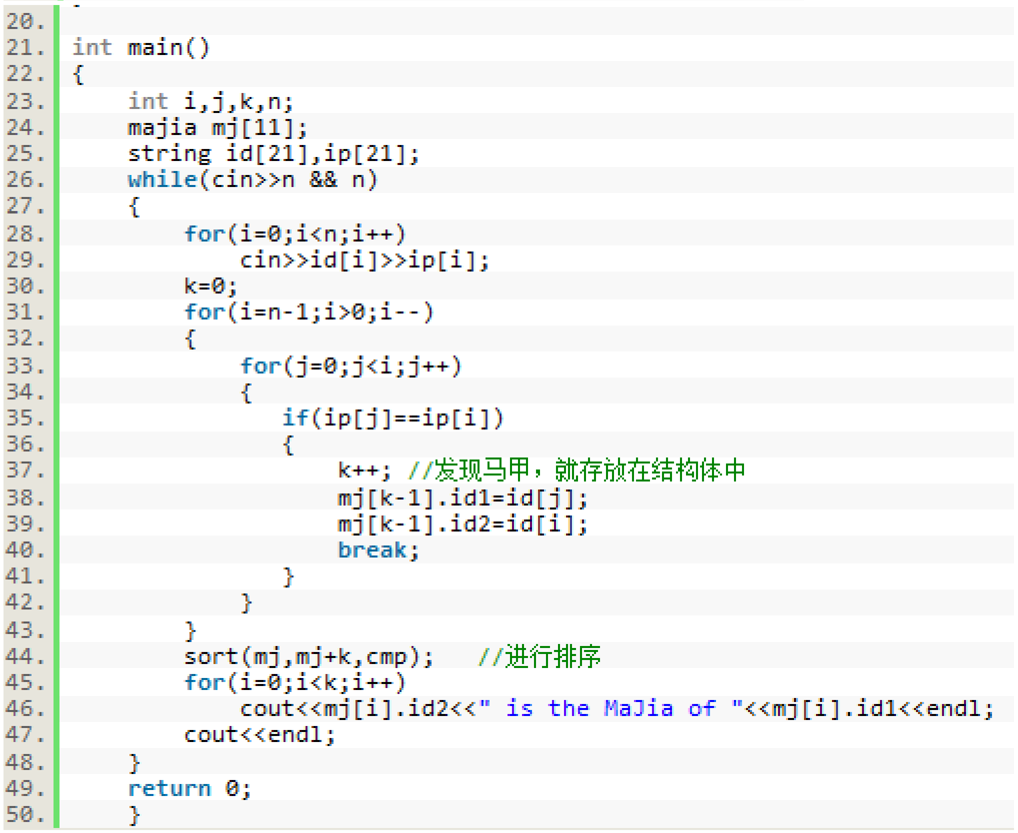
Input

The first line of each test cases is an even integer **n** (0<=n<=20), the number of posts on BBS.
Then n lines follow, each line consists of two strings:
BBS_ID IP_Address
BBS_ID means the ID who posts this post. BBS_ID is a string contains only lower case alphabetical characters and its length is not greater than 12. Each BBS ID appears only once in each test cases.
IP_Address is the IP address of that person. The IP address is formatted as "A.B.C.D", where A, B, C, D are integers ranging from 0 to 255.
It is sure that there are exactly 2 different BBS IDs with the same IP address. The first ID appears in the input is the main ID while the other is the MJ of that person.
Your program should be terminated by n = 0.

Output

For each test case, output n/2 lines of the following format: "MJ_ID is the MaJia of main_ID"
They should be displayed in the lexicographical order of the main_ID.
Print a blank line after each test cases.
See the sample output for more details.

Sample Input
```
8
inkfish 192.168.29.24
zhi 192.168.29.235
magicpig 192.168.50.170
pegasus 192.168.29.235
iamcs 202.116.77.131
finalBob 192.168.29.24
tomek 202.116.77.131
magicduck 192.168.50.170
4
mmmmmm 172.16.72.126
kkkkkk 192.168.49.161
llllll 192.168.49.161
nnnnnn 172.16.72.126
0
```
Sample Output
```
tomek is the MaJia of iamcs
finalBob is the MaJia of inkfish
magicduck is the MaJia of magicpig
pegasus is the MaJia of zhi

llllll is the MaJia of kkkkkk
nnnnnn is the MaJia of mmmmmm
```

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

using namespace std;

struct majia    //把找到的主要帐号和马甲帐号存放在结构体中
{
          string id1,id2;
};
bool cmp(const majia& mj1,const majia& mj2)
{
     return mj1.id1<mj2.id1;     //进行字符串比较
}

int main()
{
    int i,j,k,n;
    majia mj[11];
    string id[21],ip[21];
    while(cin>>n && n)
    {
        for(i=0;i<n;i++)
            cin>>id[i]>>ip[i];
        k=0;
        for(i=n-1;i>0;i--)
        {
            for(j=0;j<i;j++)
            {
               if(ip[j]==ip[i])
               {
                   k++; //发现马甲，就存放在结构体中
                   mj[k-1].id1=id[j];
                   mj[k-1].id2=id[i];
                   break;
               }
            }
        }
        sort(mj,mj+k,cmp);   //进行排序
        for(i=0;i<k;i++)
            cout<<mj[i].id2<<" is the MaJia of "<<mj[i].id1<<endl;
        cout<<endl;
    }
    return 0;
    }
```

Sicily

Hw 11. Binary Trees

1000. Play with Tree VI

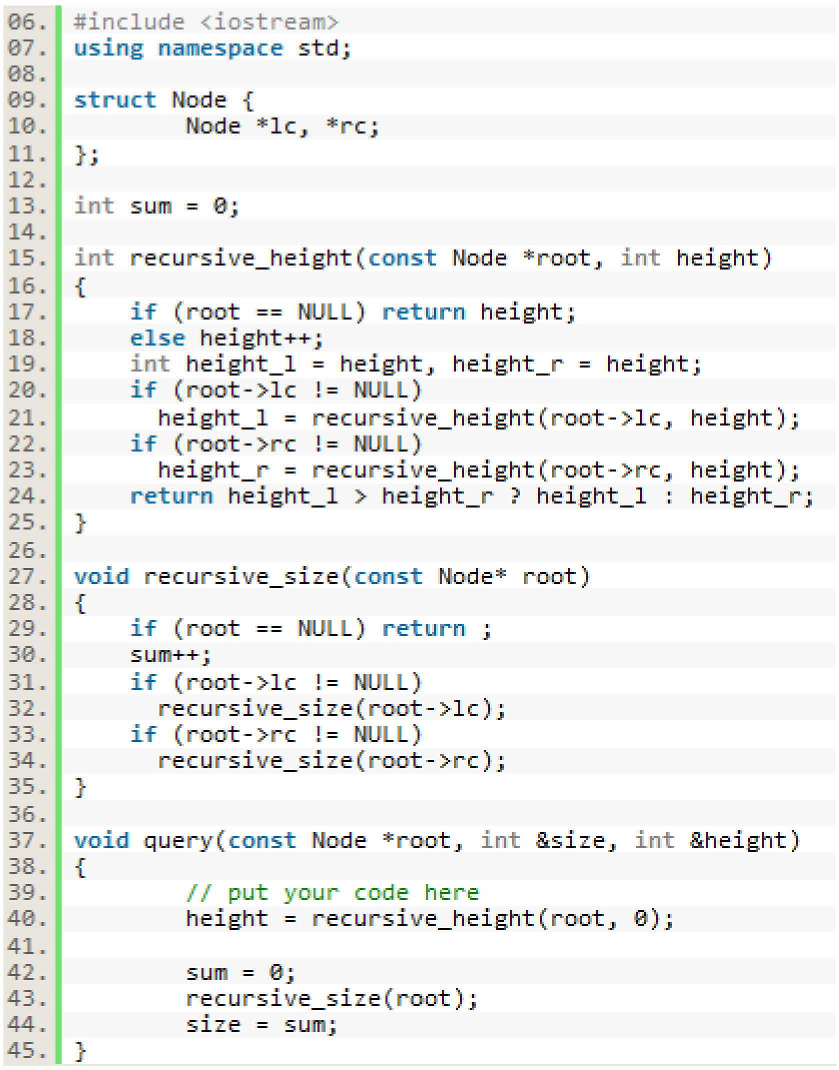
Description

Given a tree, your task is to get the height and size(number of nodes) of the tree.

```
struct Node {
        Node *lc, *rc;
};
```

```
void query(const Node *root, int &size, int &height)
{
        // put your code here
}
```

Submit the struct and the function only.



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

struct Node {
        Node *lc, *rc;
};

int sum = 0;

int recursive_height(const Node *root, int height)
{
    if (root == NULL) return height;
    else height++;
    int height_l = height, height_r = height;
    if (root->lc != NULL)
      height_l = recursive_height(root->lc, height);
    if (root->rc != NULL)
      height_r = recursive_height(root->rc, height);
    return height_l > height_r ? height_l : height_r;
}

void recursive_size(const Node* root)
{
    if (root == NULL) return ;
    sum++;
    if (root->lc != NULL)
      recursive_size(root->lc);
    if (root->rc != NULL)
      recursive_size(root->rc);
}

void query(const Node *root, int &size, int &height)
{
        // put your code here
        height = recursive_height(root, 0);

        sum = 0;
        recursive_size(root);
        size = sum;
}
```

1001. Postorder

Description

给出树的前序遍历 A[]与中序遍历 B[]，求后序遍历。

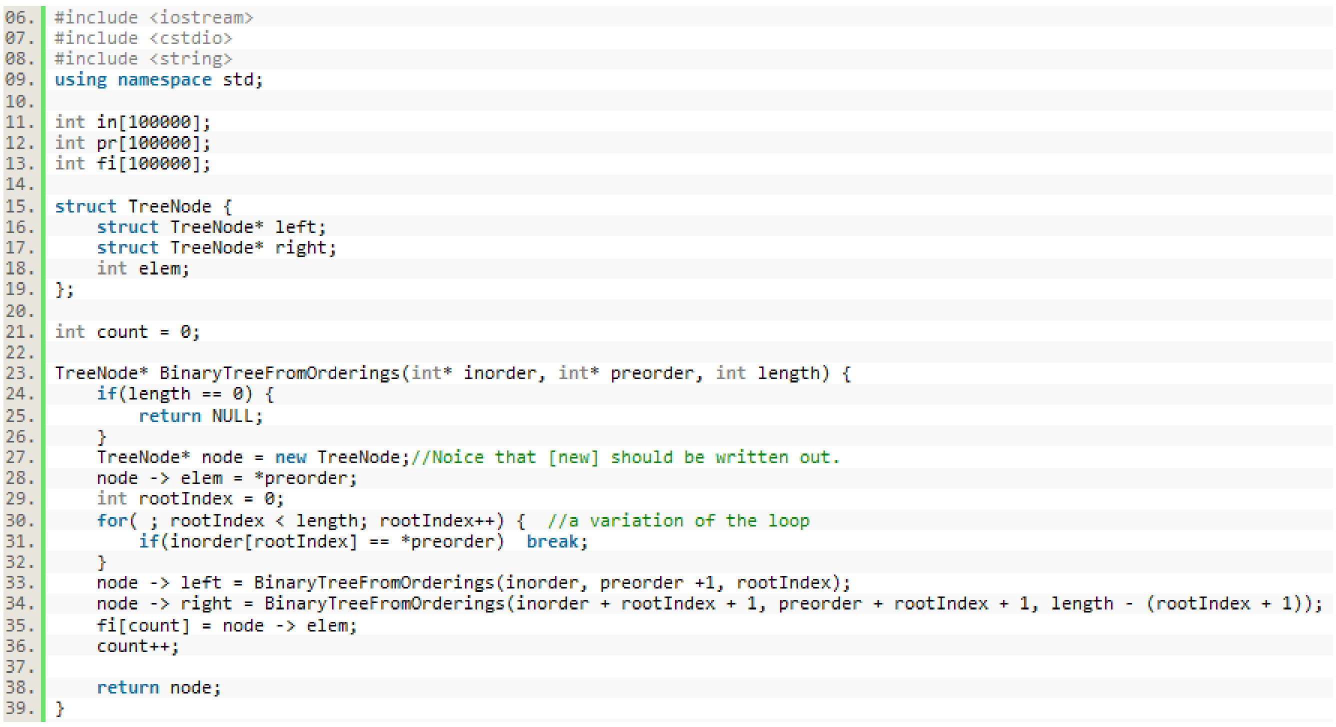
Input

Input contain 3 lines;
The first line contain an integer n(n <= 100000), the number of node in the tree.
The second line contain n distinct intergers, which is the sequence A[](0 <= a0,a2,...,an-1<= n-1).
The third line contain n distinct intergers, which is the sequence B[](0 <= b0,b2,...,bn-1<= n-1).

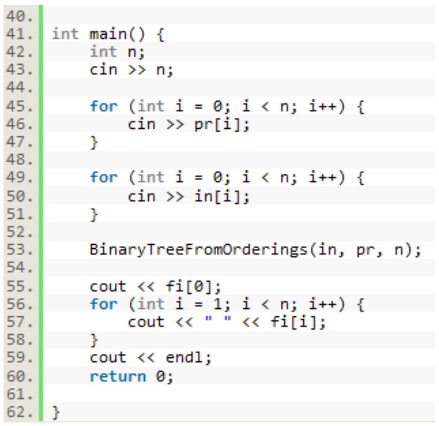
Output

The postorder of that tree.

Sample Input
```
10
7 2 0 5 8 4 9 6 3 1
7 5 8 0 4 2 6 3 9 1
```
Sample Output
```
8 5 4 0 3 6 1 9 2 7
```



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

int in[100000];
int pr[100000];
int fi[100000];

struct TreeNode {
    struct TreeNode* left;
    struct TreeNode* right;
    int elem;
};

int count = 0;

TreeNode* BinaryTreeFromOrderings(int* inorder, int* preorder, int length) {
    if(length == 0) {
        return NULL;
    }
    TreeNode* node = new TreeNode;//Noice that [new] should be written out.
    node -> elem = *preorder;
    int rootIndex = 0;
    for( ; rootIndex < length; rootIndex++) {  //a variation of the loop
        if(inorder[rootIndex] == *preorder)  break;
    }
    node -> left = BinaryTreeFromOrderings(inorder, preorder +1, rootIndex);
    node -> right = BinaryTreeFromOrderings(inorder + rootIndex + 1, preorder + rootIndex + 1, length - (rootIndex + 1));
    fi[count] = node -> elem;
    count++;

    return node;
}
```



```cpp

int main() {
    int n;
    cin >> n;

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

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

    BinaryTreeFromOrderings(in, pr, n);

    cout << fi[0];
    for (int i = 1; i < n; i++) {
        cout << " " << fi[i];
    }
    cout << endl;
    return 0;

}
```

Lab 12. Binary Trees

1000. 二叉搜索树的遍历

Description

给定一组无序整数，以第一个元素为根节点，生成一棵二叉搜索树，对其进行中序遍历和先序遍历。

Input

输入包括多组数据，每组数据包含两行：第一行为整数 m(1<=m<=3000)，表示该组数据中整数的数目，第二行给出 m 个整数，相邻整数间用一个空格间隔。最后一组数据后紧跟着包含 0 的一行输入，标识输入的结束。

Output

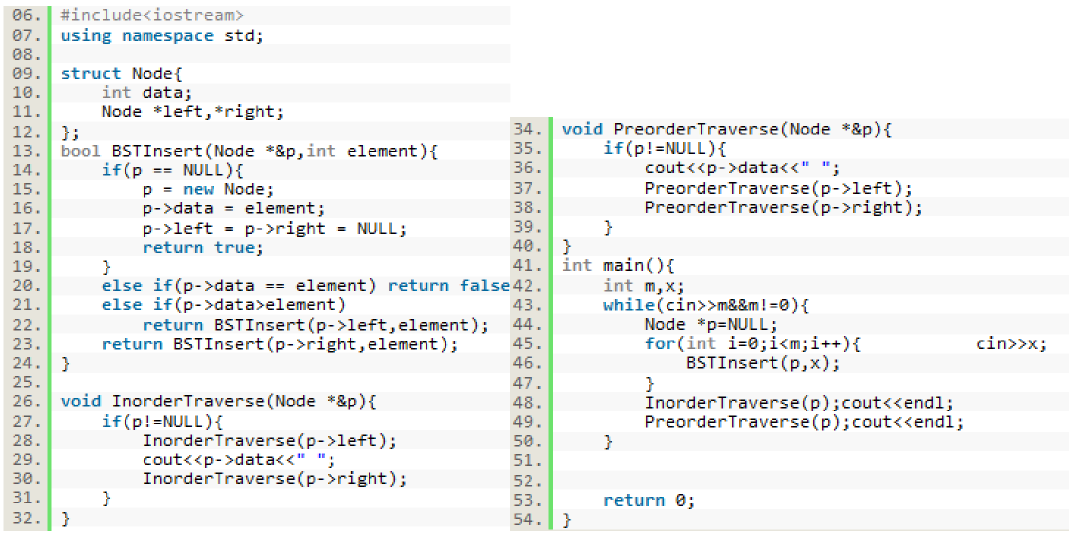
每组输入产生两行输出，第一行是中序遍历结果，第二行是先序遍历结果，每个整数后面带一个空格，每行中第一个整数前无空格。

Sample Input

```
9
10 4 16 9 8 15 21 3 12
6
20 19 16 15 45 48
0
```

Sample Output

```
3 4 8 9 10 12 15 16 21
10 4 3 9 8 16 15 12 21
15 16 19 20 45 48
20 19 16 15 45 48
```



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

struct Node{
    int data;
    Node *left,*right;
};
bool BSTInsert(Node *&p,int element){
    if(p == NULL){
        p = new Node;
        p->data = element;
        p->left = p->right = NULL;
        return true;
    }
    else if(p->data == element) return false
    else if(p->data>element)
        return BSTInsert(p->left,element);
    return BSTInsert(p->right,element);
}

void InorderTraverse(Node *&p){
    if(p!=NULL){
        InorderTraverse(p->left);
        cout<<p->data<<" ";
        InorderTraverse(p->right);
    }
}
```

```
void PreorderTraverse(Node *&p){
    if(p!=NULL){
        cout<<p->data<<" ";
        PreorderTraverse(p->left);
        PreorderTraverse(p->right);
    }
}
int main(){
    int m,x;
    while(cin>>m&&m!=0){
        Node *p=NULL;
        for(int i=0;i<m;i++){           cin>>x;
            BSTInsert(p,x);
        }
        InorderTraverse(p);cout<<endl;
        PreorderTraverse(p);cout<<endl;
    }


    return 0;
}
```

1001. Binary tree

Description

Your task is very simple: Given a binary tree, every node of which contains one upper case character (‘A’ to ‘Z’); you just need to print all characters of this tree in pre-order.

Input

Input may contain several test data sets.

For each test data set, first comes one integer n (1 <= n <= 1000) in one line representing the number of nodes in the tree. Then n lines follow, each of them contains information of one tree node. One line consist of four members in order: i (integer, represents the identifier of this node, 1 <= i <= 1000, unique in this test data set), c (char, represents the content of this node described as above, ‘A’ <= c <= ‘Z’), l (integer, represents the identifier of the left child of this node, 0 <= l <= 1000, note that when l is 0 it means that there is no left child of this node), r (integer, represents the identifier of the right child of this node, 0 <= r <= 1000, note that when r is 0 it means that there is no right child of this node). These four members are separated by one space.

Input is ended by EOF.

You can assume that all inputs are valid. All nodes can form only one valid binary tree in every test data set.

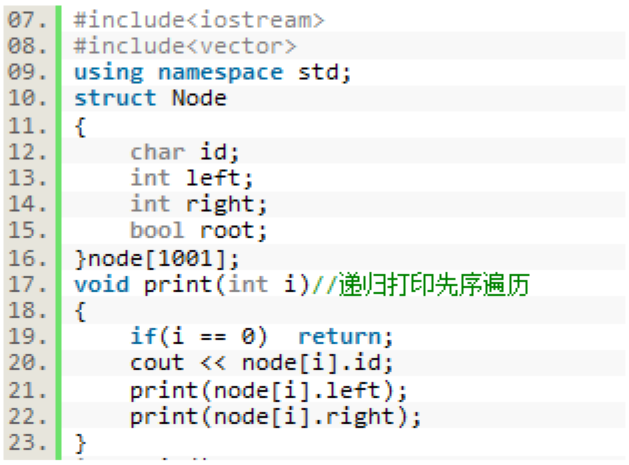
Output

For every test data set, please traverse the given tree and print the content of each node in pre-order. Characters should be printed in one line without any separating space.

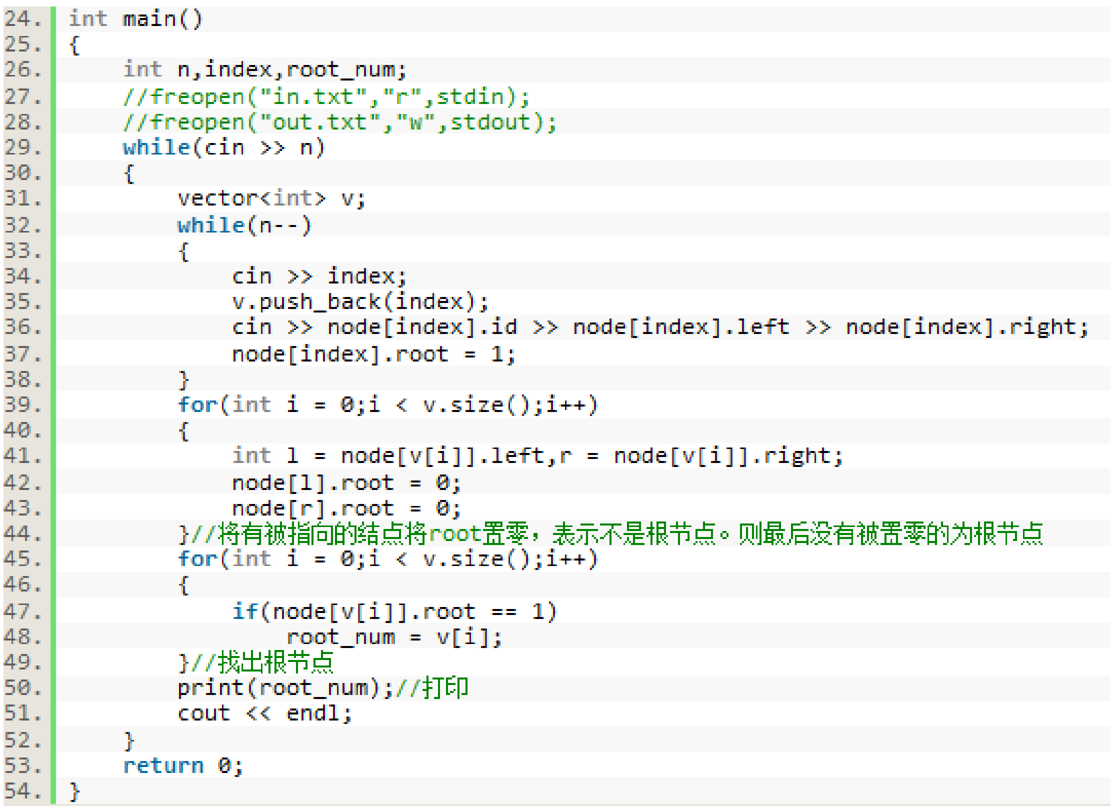
Sample Input

```
3
4 C 1 3
1 A 0 0
3 B 0 0
1
```

1000 Z 0 0
3
1 Q 0 2
2 W 3 0
3 Q 0 0
Sample Output
CAB
Z
QWQ



```cpp
#include<iostream>
#include<vector>
using namespace std;
struct Node
{
    char id;
    int left;
    int right;
    bool root;
}node[1001];
void print(int i)//递归打印先序遍历
{
    if(i == 0)  return;
    cout << node[i].id;
    print(node[i].left);
    print(node[i].right);
}
```



```cpp
int main()
{
    int n,index,root_num;
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    while(cin >> n)
    {
        vector<int> v;
        while(n--)
        {
            cin >> index;
            v.push_back(index);
            cin >> node[index].id >> node[index].left >> node[index].right;
            node[index].root = 1;
        }
        for(int i = 0;i < v.size();i++)
        {
            int l = node[v[i]].left,r = node[v[i]].right;
            node[l].root = 0;
            node[r].root = 0;
        }//将有被指向的结点将root置零，表示不是根节点。则最后没有被置零的为根节点
        for(int i = 0;i < v.size();i++)
        {
            if(node[v[i]].root == 1)
                root_num = v[i];
        }//找出根节点
        print(root_num);//打印
        cout << endl;
    }
    return 0;
}
```

1002. 完全二叉树

Description

如下图，由正整数 1，2，3，...组成一棵无限大的满二叉树。从某一个结点到根结点（编号是 1 的结点）都有
一条唯一的路径，比如 10 到根节点的路径是(10,5,2,1)，由 4 到根节点的路径是(4,2,1)，从根结点 1 到根结点
的路径上只包含一个结点 1，因此路径是(1)。
对于两个结点 X 和 Y,假设它们到根结点的路径分别是(X1,X2,...,1)和(Y1,Y2,...,1)(这里显然有 X=X1,Y=Y1)，
那么必然存在两个正整数 i 和 j,使得从 Xi 和 Yj 开始,有 Xi=Yj,Xi+1=Yj+1,...，现在的问题就是，给定 X 和 Y，
要求 Xi(也就是 Yj)。

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

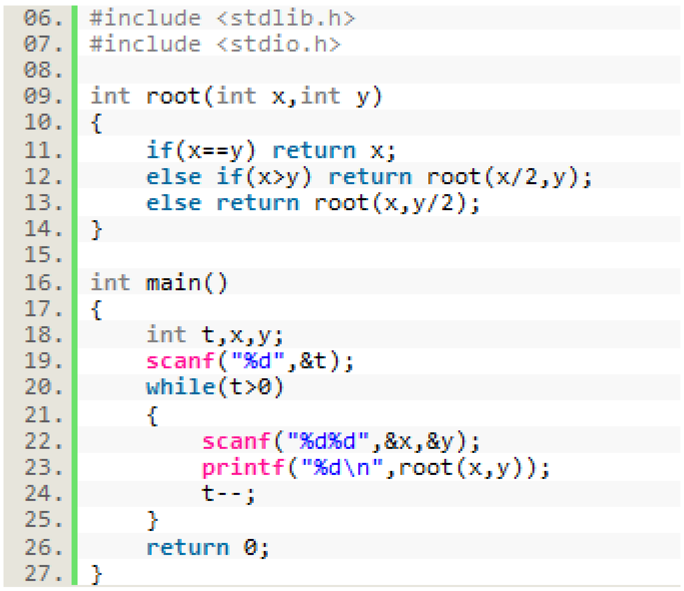
Input

输入的第一行是一个整数 T，表示测试用例个数。以下 T 行，每行对应一个测试用例。
每个测试用例包括两个整数 X 和 Y，这两个整数都不大于 1000。

Output

对每个测试用例，单独一行输出一个整数 Xi。

Sample Input
```
2
10 4
7 13
```
Sample Output
```
2
3
```



```c
#include <stdlib.h>
#include <stdio.h>

int root(int x,int y)
{
    if(x==y) return x;
    else if(x>y) return root(x/2,y);
    else return root(x,y/2);
}

int main()
{
    int t,x,y;
    scanf("%d",&t);
    while(t>0)
    {
        scanf("%d%d",&x,&y);
        printf("%d\n",root(x,y));
        t--;
    }
    return 0;
}
```

Lab 13. Multiway Trees

1000. 家族查询

Description

某个家族人员过于庞大，要判断两个是否是亲戚，确实还很不容易，现在给出某个亲戚关系图，求任意给出的两个人是否具有亲戚关系。
规定：x 和 y 是亲戚，y 和 z 是亲戚，那么 x 和 z 也是亲戚。如果 x,y 是亲戚，那么 x 的亲戚都是 y 的亲戚，y 的亲戚也都是 x 的亲戚。（人数≤5000，询问亲戚关系次数≤5000）。

Input

第一行为数据组数 T(T <= 20)

对于每组数据，第一行有两个整数 n、m(1 <= n, m <= 5000)，表示有 n 个人，编号 1~n，其中存在 m 个亲戚关系

接下来 m 行，每行有两个整数 u、v(u != v, 1 <= u, v <= n)，表示 u 和 v 之间有亲戚关系

然后是询问组数 q(1 <= q <= 5000)

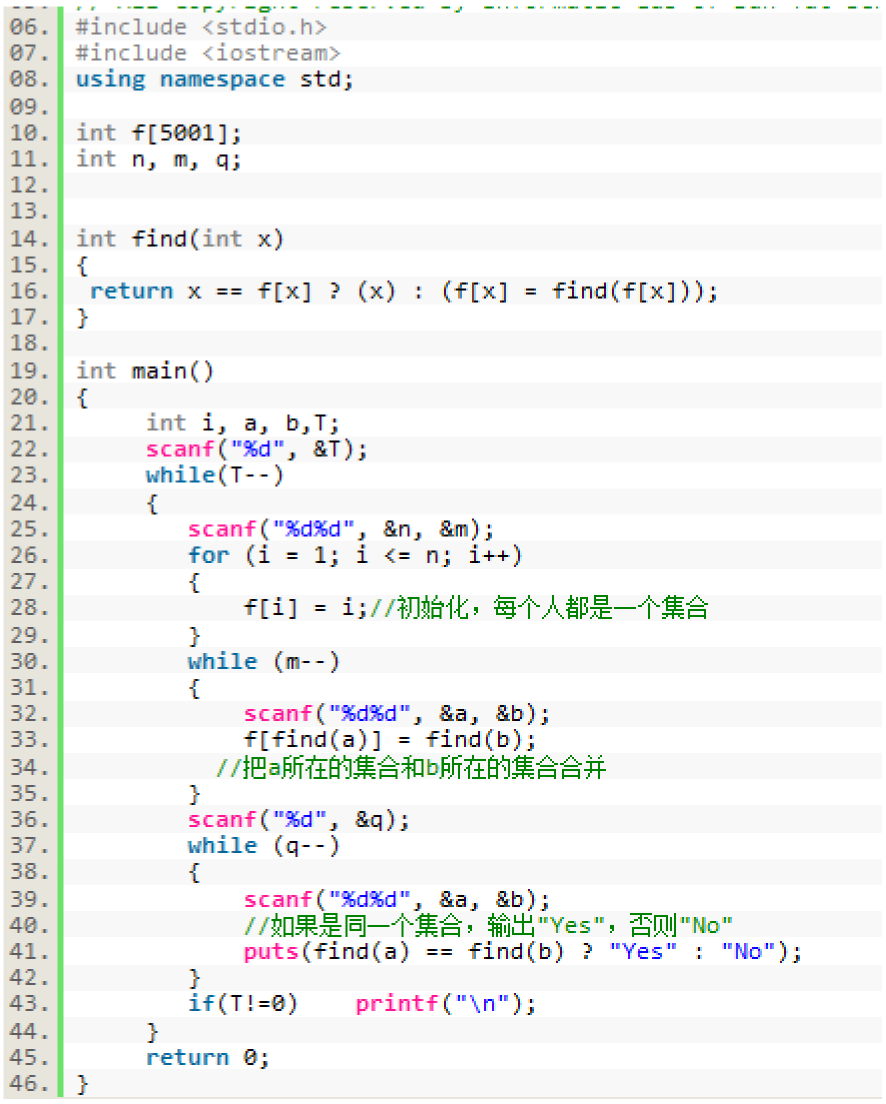
接下来 q 行，每行有两个整数 u、v(u != v, 1 <= u, v <= n)，询问 u 和 v 之间是否有亲戚关系

Output

对于每组数据，输出 q 行，每行为"Yes"或"No"，表示是否存在亲戚关系

每组数据的输出用空行隔开

Sample Input
```
2
3 1
2 3
2
1 2
2 3
4 2
1 2
1 4
3
1 2
1 3
2 4
```
Sample Output
```
No
Yes

Yes
No
Yes
```



```cpp
#include <stdio.h>
#include <iostream>
using namespace std;

int f[5001];
int n, m, q;


int find(int x)
{
 return x == f[x] ? (x) : (f[x] = find(f[x]));
}

int main()
{
     int i, a, b,T;
     scanf("%d", &T);
     while(T--)
     {
        scanf("%d%d", &n, &m);
        for (i = 1; i <= n; i++)
        {
            f[i] = i;//初始化，每个人都是一个集合
        }
        while (m--)
        {
            scanf("%d%d", &a, &b);
            f[find(a)] = find(b);
          //把a所在的集合和b所在的集合合并
        }
        scanf("%d", &q);
        while (q--)
        {
            scanf("%d%d", &a, &b);
            //如果是同一个集合，输出"Yes"，否则"No"
            puts(find(a) == find(b) ? "Yes" : "No");
        }
        if(T!=0)    printf("\n");
     }
     return 0;
}
```

1001. Phone number

Description

Given a list of phone numbers, determine if it is consistent in the sense that no number is the prefix of another.

Let’s say the phone catalogue listed these numbers:

Emergency 911

Alice 97625999

Bob 91125426

In this case, it’s not possible to call Bob, because the central would direct your call to the emergency line as soon as you had dialled the first three digits of Bob’s phone number. So this list would not be consistent.

Input

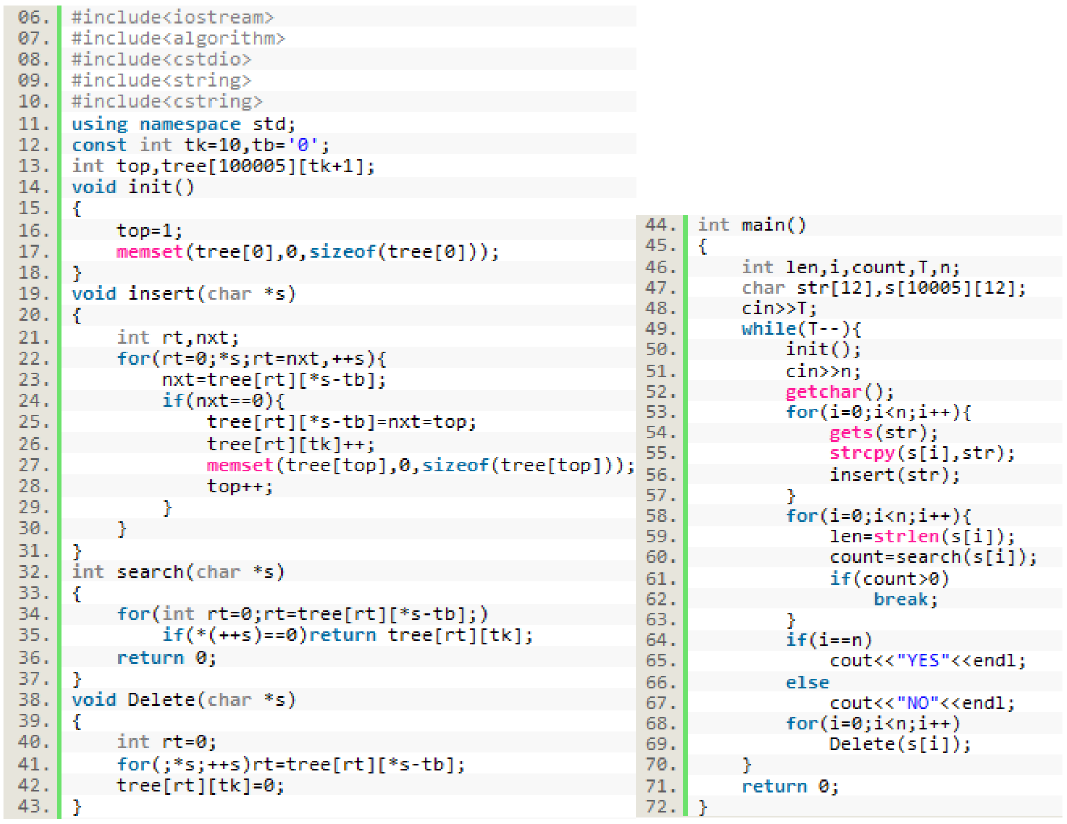
The first line of input gives a single integer, $1 \le t \le 160$, the number of test cases. Each test case starts with n, the number of phone numbers, on a separate line, $1 \le n \le 10000$.

Then follows n lines with one unique phone number on each line. A phone number is a sequence of at most ten digits.

Output

One line for each test case, output “YES” if the list is consistent, or “NO” otherwise.

Sample Input
```
1
3
911
97625999
91125426
```
Sample Output
```
NO
```



```cpp
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<string>
#include<cstring>
using namespace std;
const int tk=10,tb='0';
int top,tree[100005][tk+1];
void init()
{
    top=1;
    memset(tree[0],0,sizeof(tree[0]));
}
void insert(char *s)
{
    int rt,nxt;
    for(rt=0;*s;rt=nxt,++s){
        nxt=tree[rt][*s-tb];
        if(nxt==0){
            tree[rt][*s-tb]=nxt=top;
            tree[rt][tk]++;
            memset(tree[top],0,sizeof(tree[top]));
            top++;
        }
    }
}
int search(char *s)
{
    for(int rt=0;rt=tree[rt][*s-tb];)
        if(*(++s)==0)return tree[rt][tk];
    return 0;
}
void Delete(char *s)
{
    int rt=0;
    for(;*s;++s)rt=tree[rt][*s-tb];
    tree[rt][tk]=0;
}
int main()
{
    int len,i,count,T,n;
    char str[12],s[10005][12];
    cin>>T;
    while(T--){
        init();
        cin>>n;
        getchar();
        for(i=0;i<n;i++){
            gets(str);
            strcpy(s[i],str);
            insert(str);
        }
        for(i=0;i<n;i++){
            len=strlen(s[i]);
            count=search(s[i]);
            if(count>0)
                break;
        }
        if(i==n)
            cout<<"YES"<<endl;
        else
            cout<<"NO"<<endl;
        for(i=0;i<n;i++)
            Delete(s[i]);
    }
    return 0;
}
```

Hw 14. Graphs

1000. Can I Post the letter

Description

I am a traveler. I want to post a letter to Merlin. But because there are so many roads I can walk through, and maybe I can't go to Merlin's house following these roads, I must judge whether I can post the letter to Merlin before starting my travel.

Suppose the cities are numbered from 0 to N-1, I am at city 0, and Merlin is at city N-1. And there are M roads I can walk through, each of which connects two cities. Please note that each road is direct, i.e. a road from A to B does not indicate a road from B to A.

Please help me to find out whether I could go to Merlin's house or not.

Input

There are multiple input cases. For one case, first are two lines of two integers N and M, (N<=200, M<=N*N/2), that means the number of citys and the number of roads. And Merlin stands at city N-1. After that, there are M lines. Each line contains two integers i and j, what means that there is a road from city i to city j.

The input is terminated by N=0.

Output

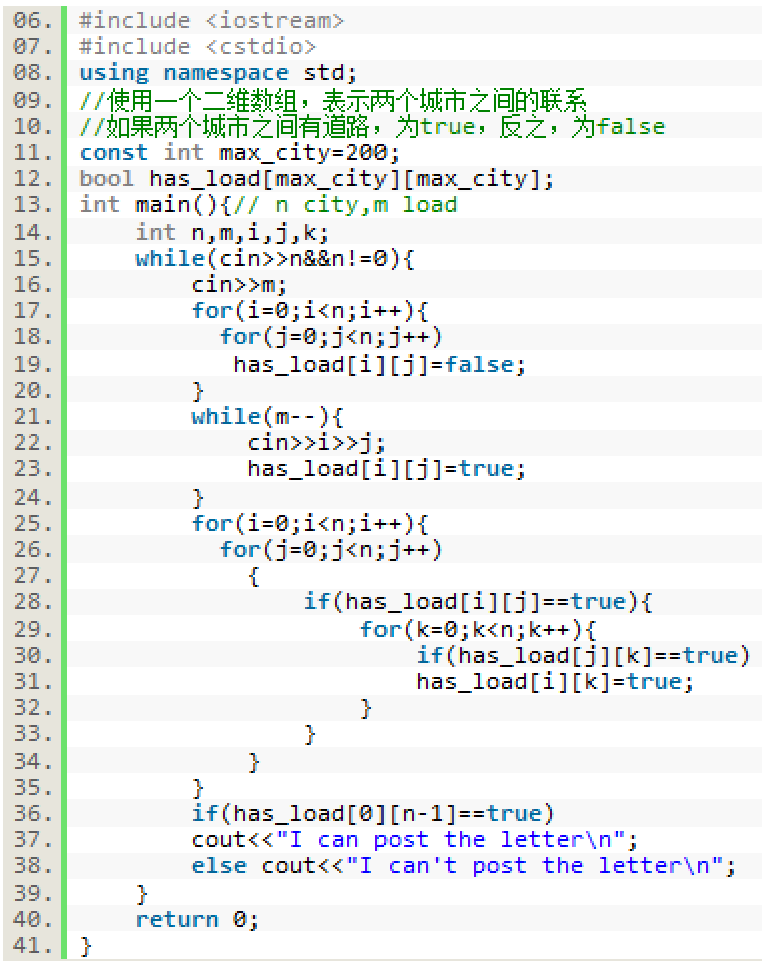
For each test case, if I can post the letter print “I can post the letter” in one line, otherwise print “I can't post the letter”.

Sample Input

```
3
2
0 1
1 2
3
1
0 1
0
```

Sample Output

```
I can post the letter
I can’t post the letter
```



```cpp
#include <iostream>
#include <cstdio>
using namespace std;
//使用一个二维数组，表示两个城市之间的联系
//如果两个城市之间有道路，为true，反之，为false
const int max_city=200;
bool has_load[max_city][max_city];
int main(){// n city,m load
    int n,m,i,j,k;
    while(cin>>n&&n!=0){
        cin>>m;
        for(i=0;i<n;i++){
          for(j=0;j<n;j++)
           has_load[i][j]=false;
        }
        while(m--){
            cin>>i>>j;
            has_load[i][j]=true;
        }
        for(i=0;i<n;i++){
          for(j=0;j<n;j++)
            {
                if(has_load[i][j]==true){
                    for(k=0;k<n;k++){
                        if(has_load[j][k]==true)
                        has_load[i][k]=true;
                    }
                }
            }
        }
        if(has_load[0][n-1]==true)
        cout<<"I can post the letter\n";
        else cout<<"I can't post the letter\n";
    }
    return 0;
}
```

1001. Connect components in undirected graph

Description

输入一个简单无向图，求出图中连通块的数目。

Input

输入的第一行包含两个整数 n 和 m，n 是图的顶点数，m 是边数。

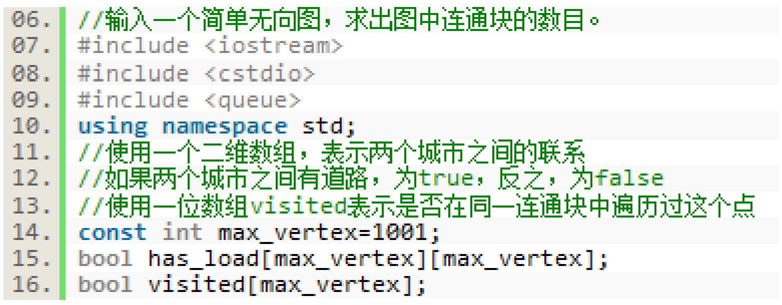
1<=n<=1000，0<=m<=10000。

以下 m 行，每行是一个数对 v y，表示存在边(v,y)。顶点编号从 1 开始。

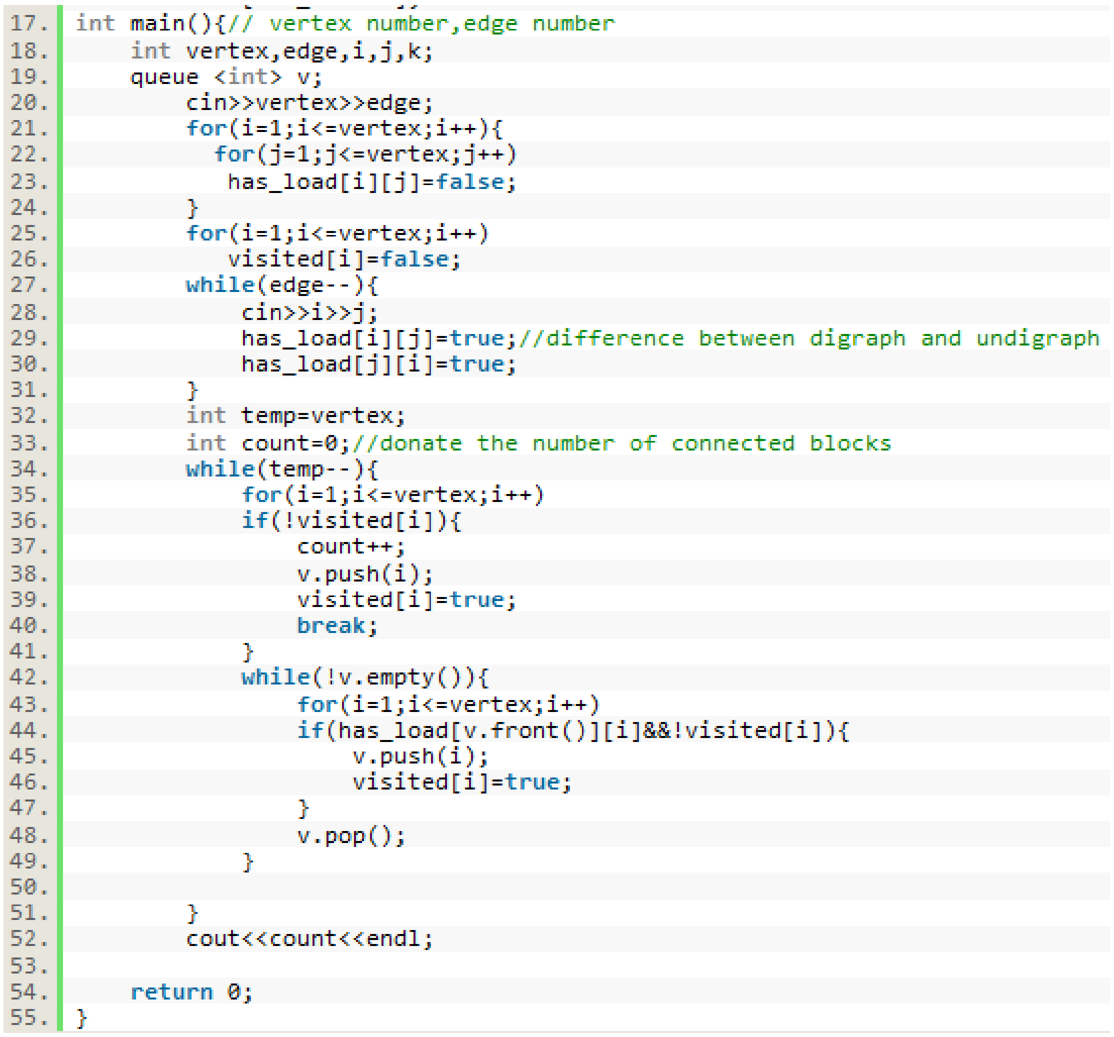
Output

单独一行输出连通块的数目。

Sample Input
5 3
1 2
1 3
2 4
Sample Output
2



```
//输入一个简单无向图，求出图中连通块的数目。
#include <iostream>
#include <cstdio>
#include <queue>
using namespace std;
//使用一个二维数组，表示两个城市之间的联系
//如果两个城市之间有道路，为true，反之，为false
//使用一位数组visited表示是否在同一连通块中遍历过这个点
const int max_vertex=1001;
bool has_load[max_vertex][max_vertex];
bool visited[max_vertex];
```



```
int main(){// vertex number,edge number
    int vertex,edge,i,j,k;
    queue <int> v;
        cin>>vertex>>edge;
        for(i=1;i<=vertex;i++){
          for(j=1;j<=vertex;j++)
           has_load[i][j]=false;
        }
        for(i=1;i<=vertex;i++)
           visited[i]=false;
        while(edge--){
            cin>>i>>j;
            has_load[i][j]=true;//difference between digraph and undigraph
            has_load[j][i]=true;
        }
        int temp=vertex;
        int count=0;//donate the number of connected blocks
        while(temp--){
            for(i=1;i<=vertex;i++)
            if(!visited[i]){
                count++;
                v.push(i);
                visited[i]=true;
                break;
            }
            while(!v.empty()){
                for(i=1;i<=vertex;i++)
                if(has_load[v.front()][i]&&!visited[i]){
                    v.push(i);
                    visited[i]=true;
                }
                v.pop();
            }

        }
        cout<<count<<endl;

    return 0;
}
```

1002. Bicoloring

Description

输入一个简单（无多重边和自环）的连通无向图，判断该图是否能用黑白两种颜色对顶点染色，使得每条边的两个端点为不同颜色。

Input

输入的第一行包含两个整数 n 和 m，n 是图的顶点数，m 是边数。

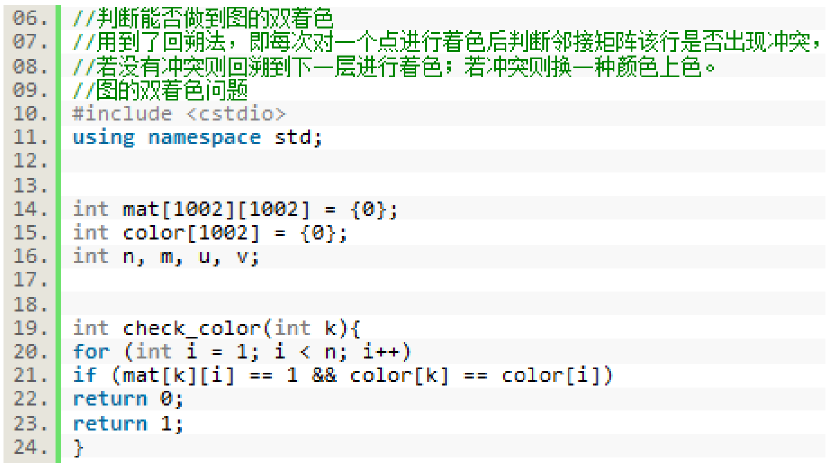
1<=n<=1000，0<=m<=10000。

以下 m 行，每行是一个数对 u v，表示存在边(u,v)。顶点编号从 1 开始。

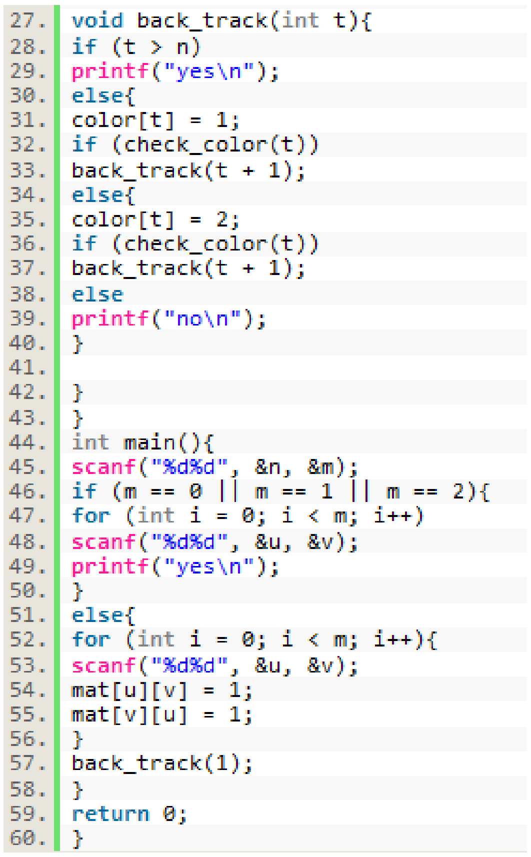
Output

如果能做到双着色，输出"yes"，否则输出"no"。

Sample Input
3 3
1 2
2 3
3 1
Sample Output
No



```
//判断能否做到图的双着色
//用到了回朔法，即每次对一个点进行着色后判断邻接矩阵该行是否出现冲突，
//若没有冲突则回溯到下一层进行着色；若冲突则换一种颜色上色。
//图的双着色问题
#include <cstdio>
using namespace std;


int mat[1002][1002] = {0};
int color[1002] = {0};
int n, m, u, v;


int check_color(int k){
for (int i = 1; i < n; i++)
if (mat[k][i] == 1 && color[k] == color[i])
return 0;
return 1;
}
```



```
void back_track(int t){
if (t > n)
printf("yes\n");
else{
color[t] = 1;
if (check_color(t))
back_track(t + 1);
else{
color[t] = 2;
if (check_color(t))
back_track(t + 1);
else
printf("no\n");
}

}
}
int main(){
scanf("%d%d", &n, &m);
if (m == 0 || m == 1 || m == 2){
for (int i = 0; i < m; i++)
scanf("%d%d", &u, &v);
printf("yes\n");
}
else{
for (int i = 0; i < m; i++){
scanf("%d%d", &u, &v);
mat[u][v] = 1;
mat[v][u] = 1;
}
back_track(1);
}
return 0;
}
```

Lab 14. Graphs

1000. Ordering Tasks

Description

John has n tasks to do. Unfortunately, the tasks are not independent and the execution of one task is only possible if other tasks have already been executed.

Input

There are multiple test cases. The first line contains an integer T, indicating the number of test cases. Each test case begins with a line containing two integers, 1 <= n <= 100000 and 1 <= m <= 100000. n is the number of tasks (numbered from 1 to n) and m is the number of direct precedence relations between tasks. After this, there will be m lines with two integers i and j, representing the fact that **task i must be executed before task j**. It is guaranteed that no task needs to be executed before itself either directly or indirectly.

Output

For each test case, print a line with n integers representing the tasks in a possible order of execution. To separate them, print exactly one space after each integer. If there are multiple solutions, output the smallest one by lexical order.

Sample Input
```
1
5 5
3 4
4 1
3 2
2 4
5 3
```
Sample Output
```
5 3 2 4 1
```

```cpp
#include<iostream>
#include<cstdio>
#include<vector>
#include<queue>//优先队列的头文件
#include<cstring>//memset头文件
using namespace std;

struct cmp{
    bool operator ()(int &a,int &b){
        return a>b;//最小值优先
    }
};
const int max_tasks=100001;
int indegree[max_tasks]={0};
priority_queue <int,vector<int>,cmp> q;
//int has_load[100][100];
vector<int> v[max_tasks];//has load between two points
int main(){
    int test,tasks,precedence;
    int i,j;
    cin>>test;
    while(test--){
        cin>>tasks>>precedence;
        memset(indegree,0,sizeof(indegree));
        for(i=0;i<=max_tasks;i++){
            v[i].clear();
        }
        while(!q.empty()){
            q.pop();
        }
```

```cpp
        while(precedence--){
            cin>>i>>j;
            //has_load[i][j]=1;
            v[i].push_back(j);
            indegree[j]++;
        }
        for(i=1;i<=tasks;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }
        int temp;
        while(!q.empty()){
            temp=q.top();
            printf("%d ",temp);
            q.pop();
            //for(i=1;i<=tasks;i++){
                //if(has_load[temp][i])
            vector<int>:: iterator it;
            for(it=v[temp].begin();it!=v[temp].end();it++){
                    indegree[*it]--;
                    if(indegree[*it]==0) q.push(*it);
            }
            v[temp].clear();
    }
        printf("\n");
    }

    return 0;
}
```

1001. DAG?

Description

输入一个有向图，判断该图是否是有向无环图(Directed Acyclic Graph)。

Input

输入的第一行包含两个整数 n 和 m，n 是图的顶点数，m 是边数。

1<=n<=100，0<=m<=10000。

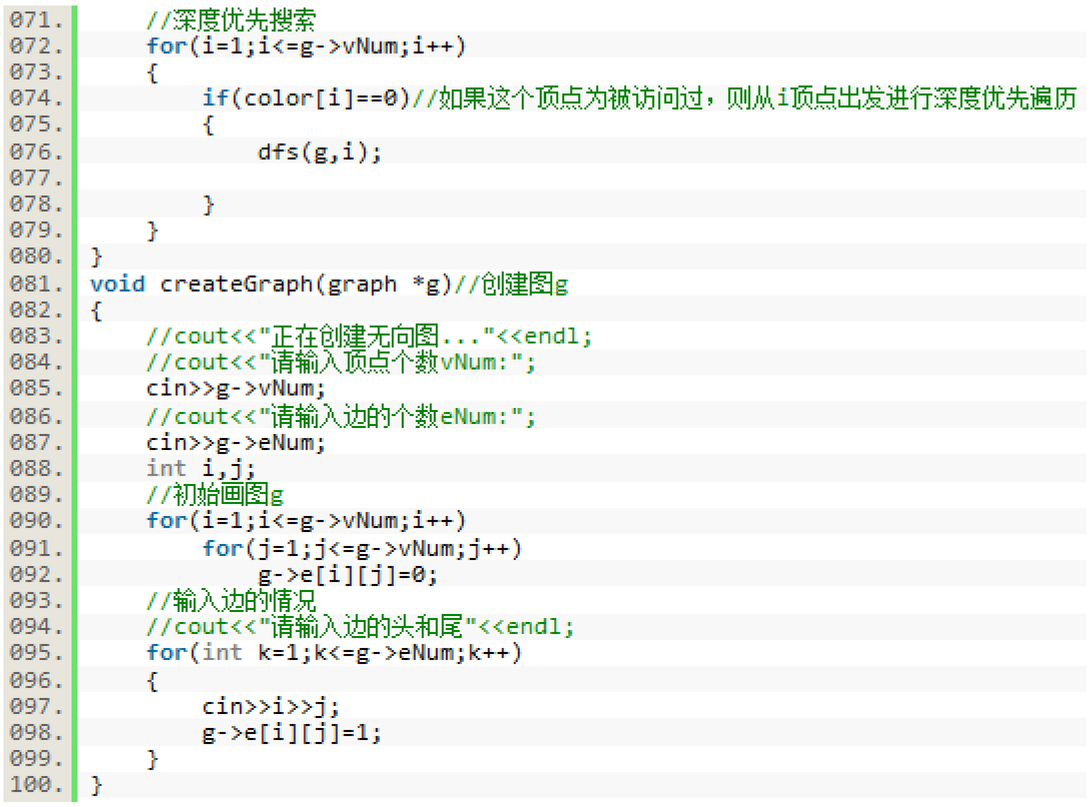
接下来的 m 行，每行是一个数对 u v，表示存在有向边(u,v)。顶点编号从 1 开始。

Output

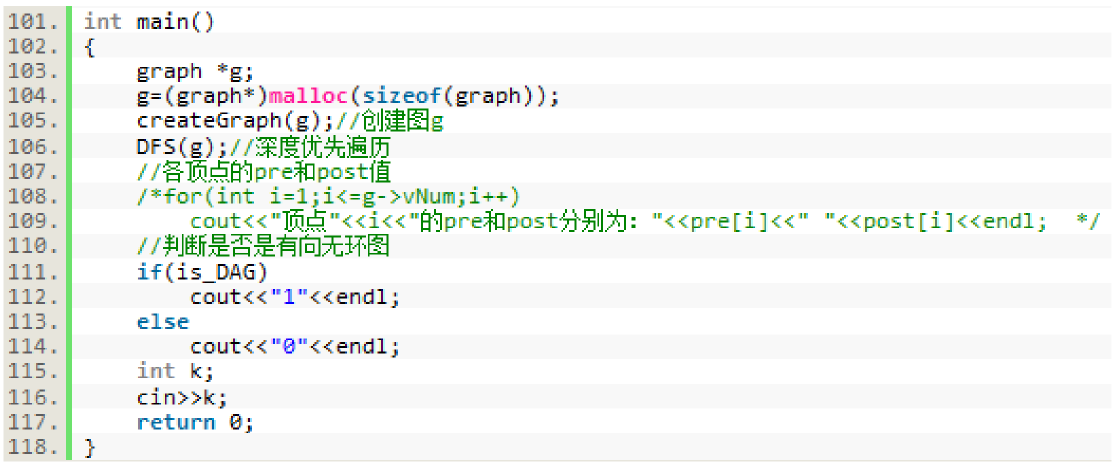
如果图是 DAG，输出 1，否则输出 0

Sample Input
```
3 3
1 2
2 3
3 1
```
Sample Output
```
0
```

```
#include<iostream>
#include<malloc.h>
using namespace std;
#define maxNum 100 //定义邻接举证的最大定点数
int pre[maxNum];
int post[maxNum];
int point=0;//pre和post的值
bool is_DAG=true;//标识位，表示有向无环图
/*
顶点颜色表 color[u]
    0 白色，未被访问过的节点标白色
    -1 灰色，已经被访问过一次的节点标灰色
    1 黑色，当该节点的所有后代都被访问过标黑色
反向边:
    如果第一次访问(u,v)时v为灰色，则(u,v)为反向边。在对图的深度优先搜索中没有发现
    反向边，则该图没有回路
程序判断依据:
    仍然是按图的节点深度遍历，访问到V时，V若被访问过，那么有2种状态:
    color[u]=-1，程序跳出，存在环
    color[u]=1，程序继续，这不是环
时间复杂度: O(n+e)
*/
int color[maxNum];//顶点颜色表 color[u]
//图的邻接矩阵表示结构
typedef struct
{
    char v[maxNum];//图的顶点信息
    int e[maxNum][maxNum];//图的顶点信息
    int vNum;//顶点个数
    int eNum;//边的个数
}graph;
void createGraph(graph *g);//创建图g
void DFS(graph *g);//深度优先遍历图g
void dfs(graph *g,int i);//从顶点i开始深度优先遍历与其相邻的点
void dfs(graph *g,int i)
{
    //cout<<"顶点"<<g->v[i]<<"已经被访问"<<endl;
    //cout<<"顶点"<<i<<"已经被访问"<<endl;
    color[i]=-1;
    pre[i]=++point;
    for(int j=1;j<=g->vNum;j++)
    {
        if(g->e[i][j]!=0)
        {
            if(color[j]==-1)//探索到回边,存在环
            {
                is_DAG=false;//不是有向无环图
            }
            else if(color[j]==0)
                dfs(g,j);
        }
    }
    post[i]=++point;
    color[i]=1;//表示i的后裔节点都被访问过
}
void DFS(graph *g)
{
    int i;
    //初始化color数组，表示一开始所有顶点都未被访问过，//初始化pre和post
    for(i=1;i<=g->vNum;i++)
    {
        color[i]=0;
        pre[i]=0;
        post[i]=0;
    }
```



```
    //深度优先搜索
    for(i=1;i<=g->vNum;i++)
    {
        if(color[i]==0)//如果这个顶点为被访问过，则从i顶点出发进行深度优先遍历
        {
            dfs(g,i);

        }
    }
}
void createGraph(graph *g)//创建图g
{
    //cout<<"正在创建无向图..."<<endl;
    //cout<<"请输入顶点个数vNum:";
    cin>>g->vNum;
    //cout<<"请输入边的个数eNum:";
    cin>>g->eNum;
    int i,j;
    //初始画图g
    for(i=1;i<=g->vNum;i++)
        for(j=1;j<=g->vNum;j++)
            g->e[i][j]=0;
    //输入边的情况
    //cout<<"请输入边的头和尾"<<endl;
    for(int k=1;k<=g->eNum;k++)
    {
        cin>>i>>j;
        g->e[i][j]=1;
    }
}
```



```
int main()
{
    graph *g;
    g=(graph*)malloc(sizeof(graph));
    createGraph(g);//创建图g
    DFS(g);//深度优先遍历
    //各顶点的pre和post值
    /*for(int i=1;i<=g->vNum;i++)
        cout<<"顶点"<<i<<"的pre和post分别为: "<<pre[i]<<" "<<post[i]<<endl;  */
    //判断是否是有向无环图
    if(is_DAG)
        cout<<"1"<<endl;
    else
        cout<<"0"<<endl;
    int k;
    cin>>k;
    return 0;
}
```

1002. 无路可逃？

Description

唐僧被妖怪关在迷宫中。孙悟空好不容易找到一张迷宫地图，并通过一个魔法门来到来到迷宫某个位置。假设迷宫是一个 n*m 的矩阵，它有两种地形，1 表示平地，0 表示沼泽，孙悟空只能停留在平地上。孙悟空目前的位置在坐标(sx,sy)处，他可以向上下左右四个方向移动。

请你帮助孙悟空计算一下，迷宫中是否存在一条路从他所在位置(sx,sy)到达唐僧所在位置(tx,ty)？

Input

输入第一行为一个整数 t（0<t<=10），表示测试用例个数。

每个测试样例的第 1 行是 2 个正整数 n （1≤n≤100)，m （1≤n≤100)，表示迷宫是 n*m 的矩阵。接下来的 n 行输入迷宫，每行有 m 个数字（0 或者 1），数字之间用一个空格间隔。

接下来的 1 行是 4 个整数 sx,sy,tx,ty，1≤sx,tx≤n，1≤sy,ty≤m，表示孙悟空所在位置为第 sx 行第 sy 列，唐僧所在位置为第 tx 行第 tx 列。迷宫左上角编号是(1,1)，右下角编号是(n, m)。

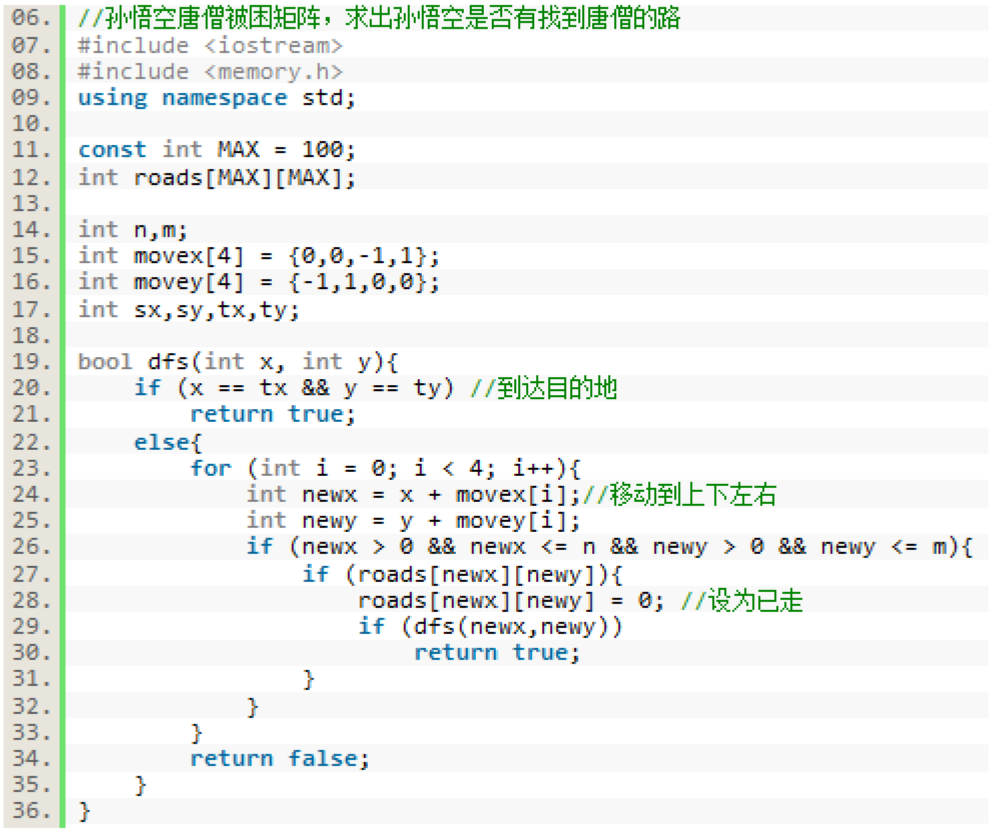
数据保证(sx,sy)格和(tx,ty)必定为平地。

Output

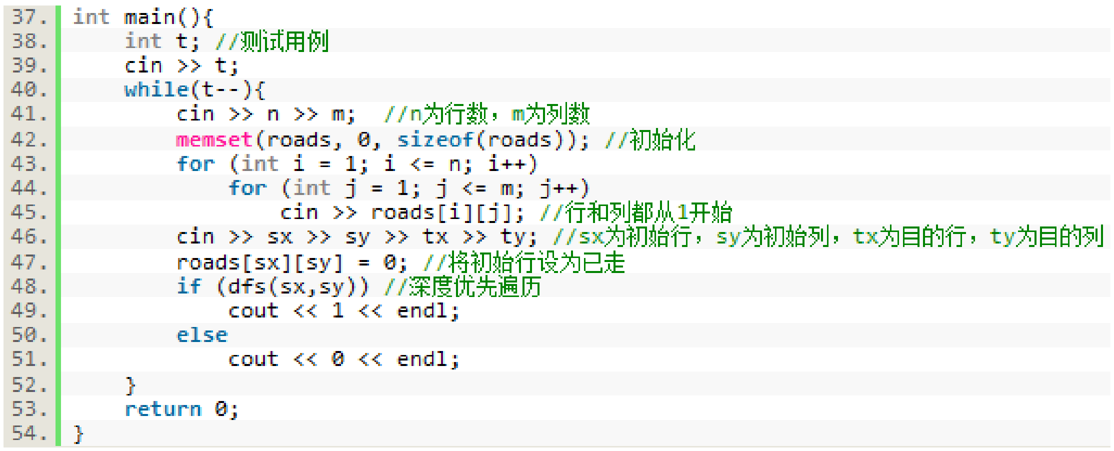
每个样例单独输出一行：1 表示路径存在，0 表示路径不存在。

```
Sample Input
2
2 2
1 0
0 1
1 1 2 2
4 4
1 1 1 0
1 0 1 1
1 0 1 1
1 1 1 0
1 1 3 4
Sample Output
```

0
1



```
//孙悟空唐僧被困矩阵，求出孙悟空是否有找到唐僧的路
#include <iostream>
#include <memory.h>
using namespace std;

const int MAX = 100;
int roads[MAX][MAX];

int n,m;
int movex[4] = {0,0,-1,1};
int movey[4] = {-1,1,0,0};
int sx,sy,tx,ty;

bool dfs(int x, int y){
    if (x == tx && y == ty) //到达目的地
        return true;
    else{
        for (int i = 0; i < 4; i++){
            int newx = x + movex[i];//移动到上下左右
            int newy = y + movey[i];
            if (newx > 0 && newx <= n && newy > 0 && newy <= m){
                if (roads[newx][newy]){
                    roads[newx][newy] = 0; //设为已走
                    if (dfs(newx,newy))
                        return true;
                }
            }
        }
        return false;
    }
}
```



```
int main(){
    int t; //测试用例
    cin >> t;
    while(t--){
        cin >> n >> m;  //n为行数，m为列数
        memset(roads, 0, sizeof(roads)); //初始化
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
                cin >> roads[i][j]; //行和列都从1开始
        cin >> sx >> sy >> tx >> ty; //sx为初始行，sy为初始列，tx为目的行，ty为目的列
        roads[sx][sy] = 0; //将初始行设为已走
        if (dfs(sx,sy)) //深度优先遍历
            cout << 1 << endl;
        else
            cout << 0 << endl;
    }
    return 0;
}
```

Lab 15. Graphs2

1000. Highways

Description

The island nation of Flatopia is perfectly flat. Unfortunately, Flatopia has no public highways. So the traffic is difficult in Flatopia. The Flatopian government is aware of this problem. They're planning to build some highways so that it will be possible to drive between any pair of towns without leaving the highway system.

Flatopian towns are numbered from 1 to N. Each highway connects exactly two towns. All highways follow straight lines. All highways can be used in both directions. Highways can freely cross each other, but a driver can only switch between highways at a town that is located at the end of both highways.

The Flatopian government wants to minimize the length of the longest highway to be built. However, they want to guarantee that every town is highway-reachable from every other town.
Input
The first line is an integer N (3 <= N <= 500), which is the number of villages. Then come N lines, the i-th of which contains N integers, and the j-th of these N integers is the distance (the distance should be an integer within [1, 65536]) between village i and village j.
Output
You should output a line contains an integer, which is the length of the longest road to be built such that all the villages are connected, and this value is minimum.

This problem contains multiple test cases!

The first line of a multiple input is an integer T, then a blank line followed by T input blocks. Each input block is in the format indicated in the problem description. There is a blank line between input blocks.
The output format consists of T output blocks. There is a blank line between output blocks.
Sample Input
```
1

3
0 990 692
990 0 179
692 179 0
```
Sample Output
```
692
```

```
//相当于求最小生成树最长的一条边，对于kruskal  algorithm为最后加进去的一条边
//kruskal: 就是构造 最小生成树 在构造最小生成树的时候 先对所有的边按权重排序 选择最小的边
// 然后在遍历剩下的边里最小的 每次找的要判断他是否已经和已选择过的边连通了
// 最后所有点都包括进来了 就完成了

#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;

const int max_villages=501;
int dis[501][501];
int a[501];//用于并查集的数组，先把每一个顶点看做一个单独的树
struct Edge{
    int u,v;
    int w;
    Edge(int x,int y,int z){
        u=x;
        v=y;
        w=z;
    }
    /*bool operator < (const Edge a){
        return w<a.w;
    };*/
};
bool cmp(Edge aa,Edge bb)
{
    return aa.w < bb.w;
}
```

```
//并查集
int find (int x){
    if(x==a[x]) return x;
    else {
        a[x]=find(a[x]);
        return a[x];
    }
}
int main(){
    int cases,villages,i,j;
    vector<Edge>edge;
    cin>>cases;
    while(cases--){
        cin>>villages;
        edge.clear();
        for(i=1;i<=villages;i++)
          for(j=1;j<=villages;j++)
          {
            cin>>dis[i][j];
            if(i>j) edge.push_back(Edge(i,j,dis[i][j]));
          }
          sort(edge.begin(),edge.end(),cmp);
          for(i=1;i<=villages;i++) a[i]=i;
          int max=0,num=0;
          for(i=0;i<edge.size();i++){
            if(find(edge[i].u)!=find(edge[i].v)){//排除了起始点是一个点的情况,同时也是判断是否联通的条件
              num++;
              max=edge[i].w;//kruskal排序中最后添加进来的一条边即为最小生成树的最长边
              a[find(edge[i].u)] =a[find(edge[i].v)];//当这条边已被添加让这两点连通
            }
            if(num==villages-1) break;
          }
          printf("%d\n",max);
          if(cases!=0)printf("\n");
    }
    return 0;
}
```

1001. Campus

Description

At present, Zhongshan University has 4 campuses with a total area of 6.17 square kilometers sitting respectively on both sides of the Pearl River or facing the South China Sea. The Guangzhou South Campus covers an area of 1.17

square kilometers, the North Campus covers an area of 0.39 square kilometers, the Guangzhou East Campus has an area of 1.13 square kilometers and the Zhuhai Campus covers an area of 3.48 square kilometers. All campuses have exuberance of green trees, abundance of lawns and beautiful sceneries, and are ideal for molding the temperaments, studying and doing research.

Sometime, the professors and students have to go from one place to another place in one campus or between campuses. They want to find the shortest path between their source place S and target place T. Can you help them?

Input

The first line of the input is a positive integer C. C is the number of test cases followed. In each test case, the first line is a positive integer N (0<N<=100) that represents the number of roads. After that, N lines follow. The i-th(1<=i<=N) line contains two strings Si, Ti and one integer Di (0<=Di<=100). It means that there is a road whose length is Di between Si and Ti. Finally, there are two strings S and T, you have to find the shortest path between S and T. S, T, Si(1<=i<=N) and Ti(1<=i<=N) are all given in the following format: str_Campus.str_Place. str_Campus represents the name of the campus, and str_Place represents the place in str_Campus. str_Campus is "North", "South", "East" or "Zhuhai". str_Place is a string which has less than one hundred lowercase characters from "a-z". You can assume that there is at most one road directly between any two places.

Output

The output of the program should consist of C lines, one line for each test case. For each test case, the output is a single line containing one integer. If there is a path between S and T, output the length of the shortest path between them. Otherwise just output "-1" (without quotation mark). No redundant spaces are needed.

Sample Input

```
1
2
South.xiaolitang South.xiongdelong 2
South.xiongdelong Zhuhai.liyuan 100
South.xiongdelong South.xiaolitang
```

Sample Output

```
2
```

```
#include<iostream>
#include<stdio.h>
#include<cmath>
#include<iomanip>
#include <map>
#include <vector>
#include <string>
#include <algorithm>
#include <sstream>
#include <stack>
using namespace std;
#define MAX 65536

typedef struct DES
{
    string des;
    int distance;
}node;



int main()
{
    int n;
    cin>>n;
    for(int i=0;i<n;i++)
    {
        map<string,vector<node> > data;
        map<string,int> flag;
        int m;
```

```
cin>>m;
for(int j=0;j<m;j++)
{
    string source,destination;
    int distance;
    cin>>source>>destination>>distance;
    flag[source]=MAX;
    flag[destination]=MAX;
    map<string,vector<node> >::iterator ite;
    //把两个节点分别作为键存储在 map 里面
    if((ite=data.find(source))!=data.end())
    {
        node tmp={destination,distance};
        ite->second.push_back(tmp);
    }
    else
    {
        node tmp={destination,distance};
        vector<node> tmpVec;
        tmpVec.push_back(tmp);
        data.insert(make_pair(source,tmpVec));
    }

    if((ite=data.find(destination))!=data.end())
    {
        node tmp={source,distance};
        ite->second.push_back(tmp);
    }
    else
    {
        node tmp={source,distance};
        vector<node> tmpVec;
        tmpVec.push_back(tmp);
        data.insert(make_pair(destination,tmpVec));
    }
}
string source,destination;
cin>>source>>destination;

stack<string> search;
search.push(source);
flag[source]=0;
//int min=-1;
while(!search.empty())
```

```
        {
            string tmpStackNode=search.top();
            vector<node> tmpVecNode=data[tmpStackNode];
            search.pop();
            for(vector<node>::iterator
ite=tmpVecNode.begin();ite!=tmpVecNode.end();ite++)
            {
                if(flag[tmpStackNode]+ite->distance<flag[ite->des])
                {
                    search.push(ite->des);
                    flag[ite->des]=flag[tmpStackNode]+ite->distance;
                }
            }
        }
        map<string,int>::iterator it=flag.find(destination);
        if(it==flag.end()||it->second==MAX)
            cout<<-1<<endl;
        else cout<<it->second<<endl;
    }
}
```

1002. Robot

Description

Karell Incorporated has designed a new exploration robot that has the ability to explore new terrains, this new robot can move in all kinds of terrain, it only needs more fuel to move in rough terrains, and less fuel in plain terrains. The only problem this robot has is that it can only move orthogonally, the robot can only move to the grids that are at the North, East, South or West of its position.

The Karell`s robot can communicate to a satellite dish to have a picture of the terrain that is going to explore, so it can select the best route to the ending point, The robot always choose the path that needs the minimum fuel to complete its exploration, however the scientist that are experimenting with the robot, need a program that computes the path that would need the minimum amount of fuel. The maximum amount of fuel that the robot can handle is 9999 units

The Terrain that the robot receives from the satellite is divided into a grid, where each cell of the grid is assigned to the amount of fuel the robot would need to pass thought that cell. The robot also receives the starting and ending coordinates of the exploration area.

Path Example

```
From (1,1) to (5,5)
Fuel needed 10
```

Input

The first line of the input file is the number of tests that must be examined.

The first line of the test is the number of rows and columns that the grid will contain.

The rows and columns will be 0 < *row*   100 , 0 < *column*   100

The next lines are the data of the terrain grid

The last line of the test has the starting and ending coordinates.

Output

One line, for each test will have the amount of fuel needed by the robot

Sample Input

```
3
5 5
1 1 5 3 2
4 1 4 2 6
3 1 1 3 3
5 2 3 1 2
2 1 1 1 1
1 1 5 5
5 4
2 2 15 1
5 1 15 1
5 3 10 1
5 2 1 1
8 13 2 15
1 1 1 4
10 10
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
```

```
1 1 10 10
```

Sample Output

```
10
15
19
```

```
#include <iostream>
#include <stdio.h>
#include <string>
#include <cstring>
#include <queue>
#include <set>
#include <vector>
using namespace std;

int dis[109][109];

struct Gra
{
    int r;
    int c;
    friend bool operator <(Gra g1, Gra g2)
    {
        if(dis[g1.r][g1.c] < dis[g2.r][g2.c])
            return true;
    }
};

int main()
{
    int t;
    cin >> t;
    while (t --)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        int s[109][109];
        for (int i = 1; i <= a; ++ i)
        {
            for (int j = 1; j <= b; ++ j)
            {
                cin >> s[i][j];
            }
        }
```

```
int ar, ac, br, bc;
cin >> ar >> ac >> br >> bc;
set<Gra> q;
int know[109][109];
memset(know, 0, sizeof(know));
memset(dis, 9999999, sizeof(dis));
Gra tu[10009];
tu[1].r = ar;
tu[1].c = ac;
q.insert(tu[1]);
Gra temp;
int i = 2;
//know[ar][ac] = 1;
dis[ar][ac] = s[ar][ac];
set<Gra>::iterator it;
set<Gra>::iterator itt;
while (!q.empty())
{
    it = q.begin();
    temp = *it;
    itt = it;
    ++ it;
    Gra tem;
    for (; it != q.end(); ++ it)
    {
        tem = *it;
        if (dis[temp.r][temp.c] > dis[tem.r][tem.c])
        {
            temp = tem;
            itt = it;
        }
    }
    q.erase(itt);
    know[temp.r][temp.c] = 1;
    if (temp.r == br && temp.c == bc)
    {
        cout << dis[temp.r][temp.c] << endl;
        break;
    }
    if (temp.c - 1 >= 1)
    {
        tu[i].c = temp.c - 1;
        tu[i].r = temp.r;
        if (know[tu[i].r][tu[i].c] == 0)
```

```
                {
                    if (dis[tu[i].r][tu[i].c] > dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c])
                    {
                        dis[tu[i].r][tu[i].c] = dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c];
                        q.insert(tu[i]);
                        ++ i;
                    }
                }
            }

            if (temp.r - 1 >= 1)
            {
                tu[i].r = temp.r - 1;
                tu[i].c = temp.c;
                if (know[tu[i].r][tu[i].c] == 0)
                {
                    if (dis[tu[i].r][tu[i].c] > dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c])
                    {
                        dis[tu[i].r][tu[i].c] = dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c];
                        q.insert(tu[i]);
                        ++ i;
                    }
                }
            }

            if (temp.c + 1 <= b)
            {
                tu[i].c = temp.c + 1;
                tu[i].r = temp.r;
                if (know[tu[i].r][tu[i].c] == 0)
                {
                    if (dis[tu[i].r][tu[i].c] > dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c])
                    {
                        dis[tu[i].r][tu[i].c] = dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c];
                        q.insert(tu[i]);
                        ++ i;
                    }
                }
            }

            if (temp.r + 1 <= a)
            {
                tu[i].r = temp.r + 1;
                tu[i].c = temp.c;
                if (know[tu[i].r][tu[i].c] == 0)
                {
                    if (dis[tu[i].r][tu[i].c] > dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c])
                    {
                        dis[tu[i].r][tu[i].c] = dis[temp.r][temp.c] + 
s[tu[i].r][tu[i].c];
                        q.insert(tu[i]);
                        ++ i;
                    }
                }
            }
        }
    }
}
```

Sicily

Lab 16. Mock Test 1

1000. 简单哈希

Description

使用线性探测法(Linear Probing)可以解决哈希中的冲突问题，其基本思想是：设哈希函数为 h(key) = d, 并且假定哈希的存储结构是循环数组, 则当冲突发生时, 继续探测 d+1, d+2…, 直到冲突得到解决.

例如, 现有关键码集为 {47，7，29，11，16，92，22，8，3}，

设：哈希表表长为 m=11；哈希函数为 Hash(key)=key mod 11；采用线性探测法处理冲突。建哈希表如下：

0	1	2	3	4	5	6	7	8	9	10
11	22		47	92	16	3	7	29	8	

现在给定哈希函数为 **Hash(key)= key mod m**，要求按照上述规则，使用线性探测法处理冲突.要求建立起相应哈希表，并按要求打印。

Input
仅有一个测试用例，第 1 行为整数 n 与 m（1 <= n, m <= 10000），n 代表 key 的总数，m 代表哈希表的长度，并且令哈希函数为：**Hash(key) = key mod m.**
接下来 n 行，每行一个整数，代表一个 key。Key 与 key 两两不相同（0 <= key <= 10，000）。
Output

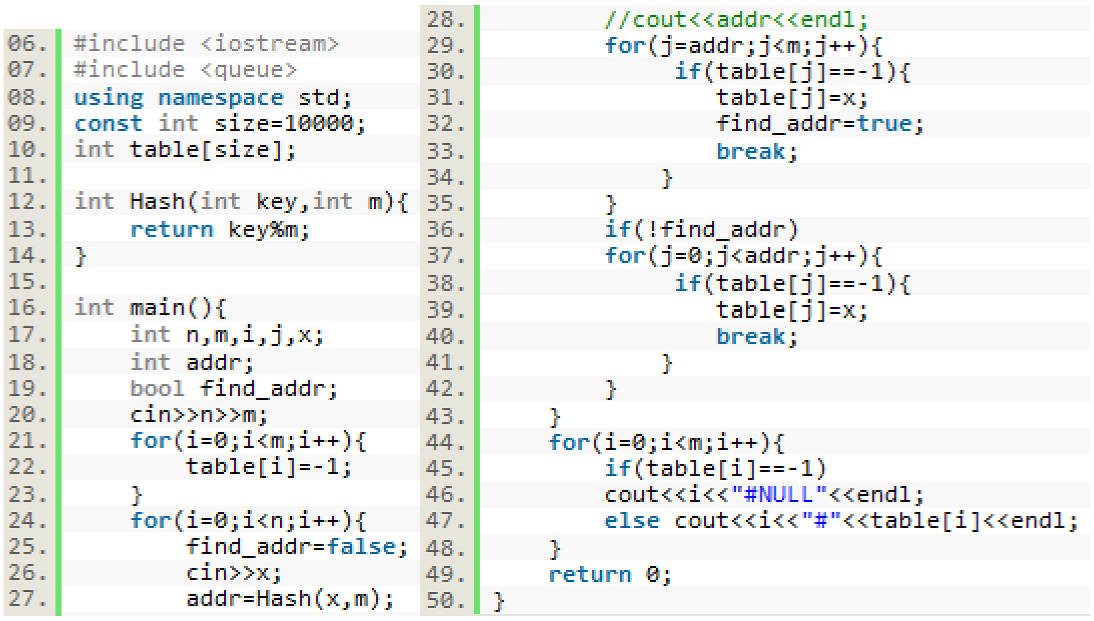
输出建立好的 hash 表，比如下表

0	1	2	3	4	5	6	7	8	9	10
11	22		47	92	16	3	7	29	8	

应输出

```
0#11
1#22
2#NULL
3#47
4#92
5#16
6#3
7#7
8#29
9#8
10#NULL
```

Sample Input
```
3 5
1
5
6
```
Sample Output
```
0#5
1#1
2#6
3#NULL
4#NULL
```



```cpp
#include <iostream>
#include <queue>
using namespace std;
const int size=10000;
int table[size];

int Hash(int key,int m){
    return key%m;
}

int main(){
    int n,m,i,j,x;
    int addr;
    bool find_addr;
    cin>>n>>m;
    for(i=0;i<m;i++){
        table[i]=-1;
    }
    for(i=0;i<n;i++){
        find_addr=false;
        cin>>x;
        addr=Hash(x,m);
        //cout<<addr<<endl;
        for(j=addr;j<m;j++){
            if(table[j]==-1){
                table[j]=x;
                find_addr=true;
                break;
            }
        }
        if(!find_addr)
        for(j=0;j<addr;j++){
            if(table[j]==-1){
                table[j]=x;
                break;
            }
        }
    }
    for(i=0;i<m;i++){
        if(table[i]==-1)
        cout<<i<<"#NULL"<<endl;
        else cout<<i<<"#"<<table[i]<<endl;
    }
    return 0;
}
```

1001. 插入排序

Description

插入排序是一种十分常见的排序方法。其基本操作就是将一个数据插入到已经排好序的有序数据中,从而得到一个新的、个数加一的有序数据，算法适用于少量数据的排序，是一种稳定的排序方法。请你结合数据结构所学知识，实现这种排序方法，并将每一轮的排序结果输出。

Input

第一行输入整数 N（1<=n<=100）代表待排序的数据个数。

接下来一行中有 N 个整数，代表待排序的数据。

Output

第一行输出原始的数据顺序，接下来每一行输出一趟排序的结果（如果顺序没有改变则不输出），直至排序完成，从而实现数据的升序排列。

如果一开始的数据已经有序，则只输出原始数据。

注意，输出的每一个数字后面均带有一个空格。

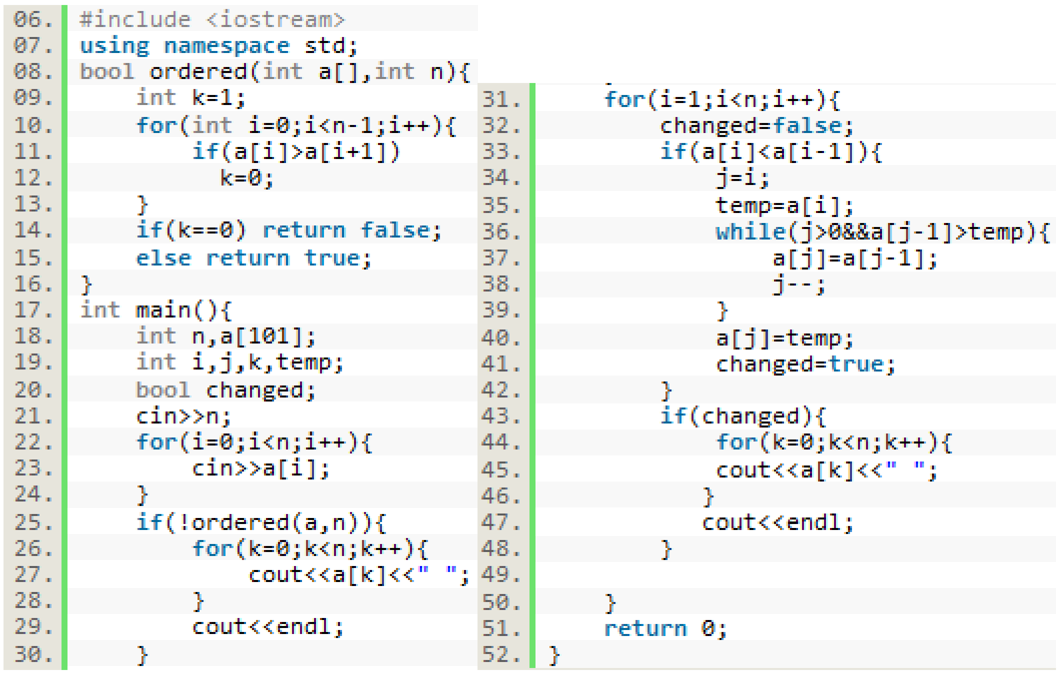
Sample Input

```
5
78 24 13 2 99
```

Sample Output

```
78 24 13 2 99
24 78 13 2 99
```

```
13 24 78 2 99
2 13 24 78 99
```

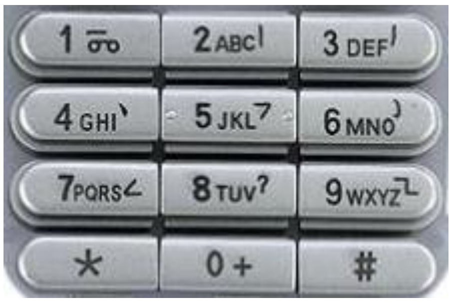


```
#include <iostream>
using namespace std;
bool ordered(int a[],int n){
    int k=1;
    for(int i=0;i<n-1;i++){
        if(a[i]>a[i+1])
          k=0;
    }
    if(k==0) return false;
    else return true;
}
int main(){
    int n,a[101];
    int i,j,k,temp;
    bool changed;
    cin>>n;
    for(i=0;i<n;i++){
        cin>>a[i];
    }
    if(!ordered(a,n)){
        for(k=0;k<n;k++){
            cout<<a[k]<<" ";
        }
        cout<<endl;
    }
    for(i=1;i<n;i++){
        changed=false;
        if(a[i]<a[i-1]){
            j=i;
            temp=a[i];
            while(j>0&&a[j-1]>temp){
                a[j]=a[j-1];
                j--;
            }
            a[j]=temp;
            changed=true;
        }
        if(changed){
            for(k=0;k<n;k++){
            cout<<a[k]<<" ";
           }
           cout<<endl;
        }

    }
    return 0;
}
```

1002. 单词数值

Description

小明喜欢发短信，一天他突然想出一种定义短信中单词的值的方法，也许能够揭示某种规律. 小明的手机键盘如下图所示:



输入法为字母输入，即每次输入一个字母，例如需要输入字母 x，由于 x 在数字键 9 上，排在第二位，因此需要按下 9 键两次.

小明定义单词的值如下：单词中每个字母(不区分大小写)的值等于**按键数值**与**按键次数**之积，单词的值等于所有字母的值之和. 如单词 word 的值为 9 + 6*3 + 7*3 + 3 = 51.

Input

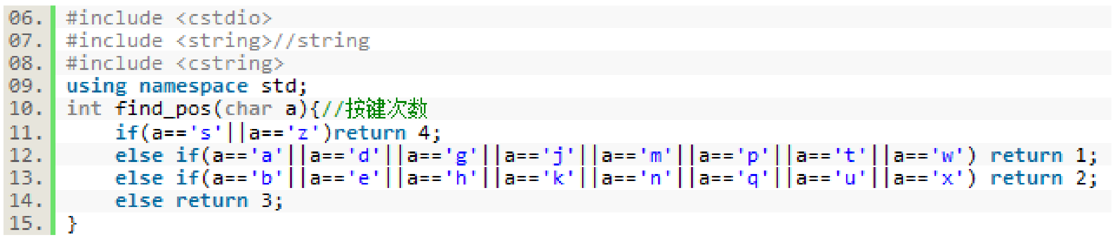
 第一行给出单词数 T(1 <= T <= 100, 000).
接下来 T 行，每行一个单词，单词中所有字母均为小写，单词最大长度不超过 100.

Output

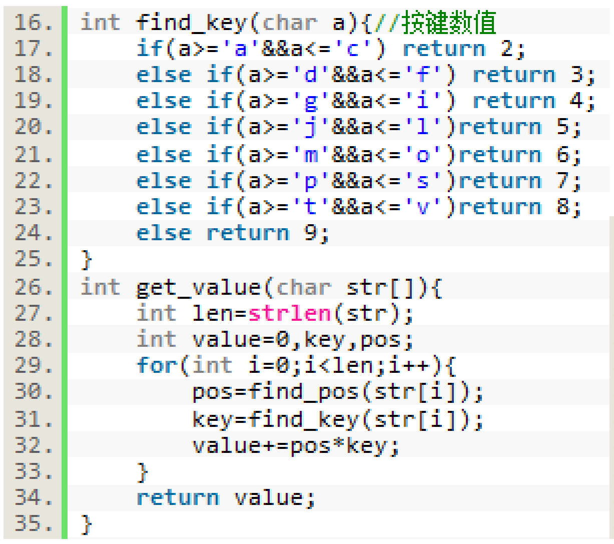
 输出 T 行，每行为单词的对应值.

Sample Input
```
3
word
i
day
```
Sample Output
```
51
12
32
```
Hint

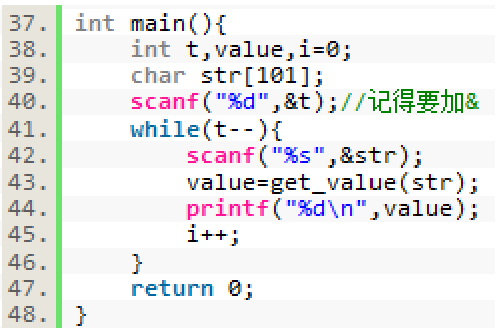
请使用 scanf 和 printf 来输入输出数据。使用 cin 和 cout 将有可能导致超时。



```
#include <cstdio>
#include <string>//string
#include <cstring>
using namespace std;
int find_pos(char a){//按键次数
    if(a=='s'||a=='z')return 4;
    else if(a=='a'||a=='d'||a=='g'||a=='j'||a=='m'||a=='p'||a=='t'||a=='w') return 1;
    else if(a=='b'||a=='e'||a=='h'||a=='k'||a=='n'||a=='q'||a=='u'||a=='x') return 2;
    else return 3;
}
```



```
int find_key(char a){//按键数值
    if(a>='a'&&a<='c') return 2;
    else if(a>='d'&&a<='f') return 3;
    else if(a>='g'&&a<='i') return 4;
    else if(a>='j'&&a<='l')return 5;
    else if(a>='m'&&a<='o')return 6;
    else if(a>='p'&&a<='s')return 7;
    else if(a>='t'&&a<='v')return 8;
    else return 9;
}
int get_value(char str[]){
    int len=strlen(str);
    int value=0,key,pos;
    for(int i=0;i<len;i++){
        pos=find_pos(str[i]);
        key=find_key(str[i]);
        value+=pos*key;
    }
    return value;
}
```



```
int main(){
    int t,value,i=0;
    char str[101];
    scanf("%d",&t);//记得要加&
    while(t--){
        scanf("%s",&str);
        value=get_value(str);
        printf("%d\n",value);
        i++;
    }
    return 0;
}
```

1003. 中值滤波

Description

中值滤波是基于排序统计理论的一种能有效抑制噪声的非线性信号处理技术。其在数字图像处理中也有广泛应用。在一维形式下，一维中值滤波具有大小为奇数的滑动模板，模板中的数据由小到大排序，取排在中间位置上的数据作为最终的结果进行处理。

中值滤波的核心运算时将模板中的数据进行排序，这样，如果一个亮点（暗点）的噪声，就会在排序的过程中被排在数据序列的最左边或者最右边，因此选择的数据序列中间位置上的值一般不是噪声点的值，由此便可达到抑制噪声的目的。

小明在学习中值滤波的过程中，根据滑动模板大小有可能是偶数的现象对其进行了改进。如果模板大小为奇数则保持原有算法，如果是偶数则取中间两位的平均值作为最后的结果。

Input

第一行为整数 n（3<=n<=10）,代表模板的大小

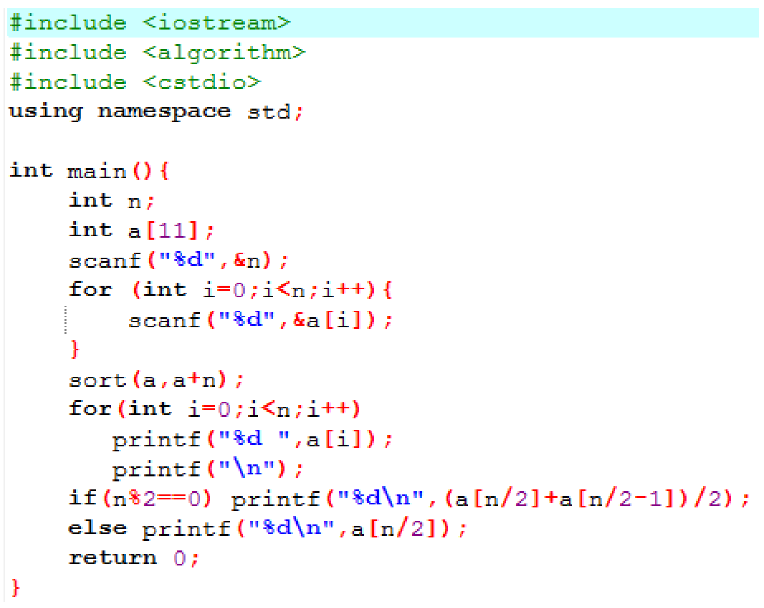
接下来 n 行，每一行一个整数，代表一个信号值 k（0<=k<=225）

Output

第一行输出排好序的信号值，每个信号值紧接一个空格。

第二行输出信号的中值（如果结果为小数则向下取整）

Sample Input
```
6
45 56 34 5 23 77
```
Sample Output
```
5 23 34 45 56 77
39
```



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

int main(){
    int n;
    int a[11];
    scanf("%d",&n);
    for (int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    sort(a,a+n);
    for(int i=0;i<n;i++)
       printf("%d ",a[i]);
       printf("\n");
    if(n%2==0) printf("%d\n",(a[n/2]+a[n/2-1])/2);
    else printf("%d\n",a[n/2]);
    return 0;
}
```

1004. 打印机调度

Description

沃森是一家打印机生产公司。该公司的打印机会根据所要打印文件的紧急程度进行打印。其基本工作原理是，接受一批打印机文件，将其放入文件队列中，文件的紧急程度划分成 9 个等级（9 级紧急程度最高，1 级紧急程度最低）。

● 队列中的第一个文件 H 将被打印。

● 如果队列中有优先级大于 H 的文件，则文件 H 在不打印的条件下移动到队列尾部（称为置尾操作）。

● 否则，打印文件 H（并不再将其放回队列中）。

现在你需要打印一系列文件。请编写程序模拟打印机的工作原理，并确定一共需要多少次置尾操作。

Input

第一行输入一个正整数代表测试的用例个数（不大于 50），对于每一个用例：

一行输入一个整数 n，其中 n 代表队列中文件的份数（1<=n<=100）。

另一行的 n 个整数代表队列中文件的优先等级（1 到 9），第一个整数是第一份文件的优先级，以此类推。

Output

对于每一个测试用例，输出只含一个整数 s 的一行，代表打印所有文件所需要的置尾操作操作数。假设中间不再有其他文件加入打印队列。

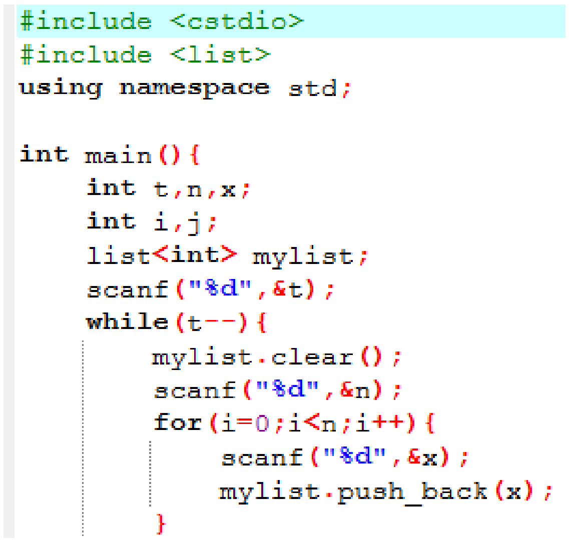
Sample Input

    Copy sample input to clipboard

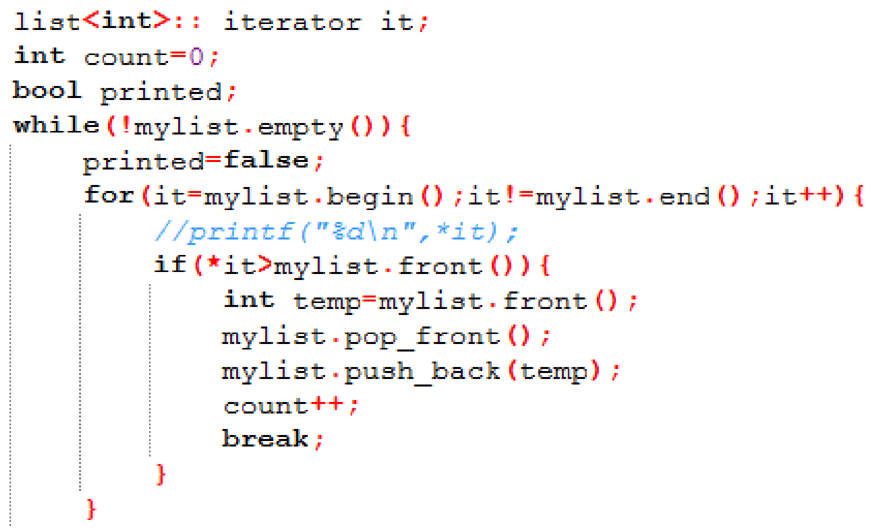
```
3
5
4 4 4 4 9
3
6 5 3
3
7 9 1
```

Sample Output

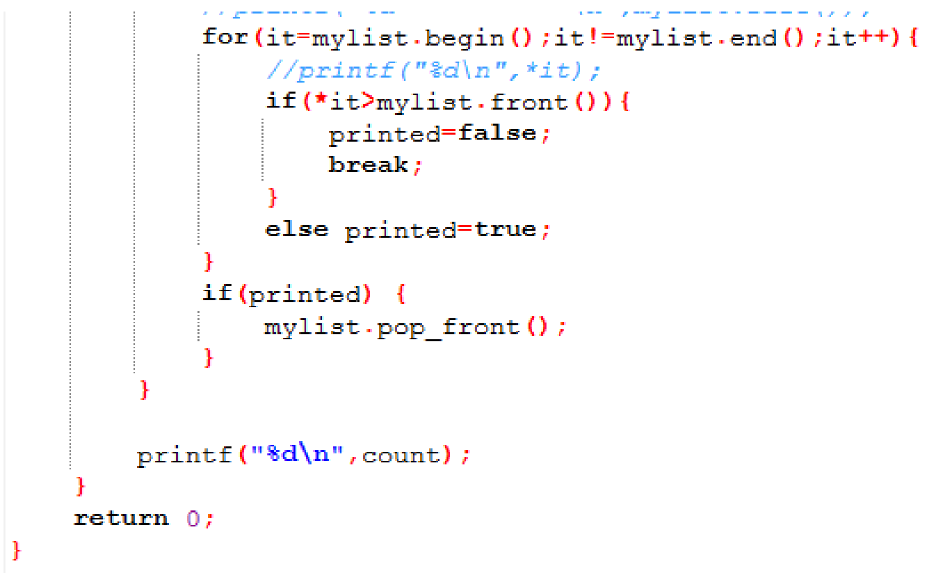
```
4
0
2
```



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

int main(){
    int t,n,x;
    int i,j;
    list<int> mylist;
    scanf("%d",&t);
    while(t--){
        mylist.clear();
        scanf("%d",&n);
        for(i=0;i<n;i++){
            scanf("%d",&x);
            mylist.push_back(x);
        }
```



```
list<int>:: iterator it;
int count=0;
bool printed;
while(!mylist.empty()){
    printed=false;
    for(it=mylist.begin();it!=mylist.end();it++){
        //printf("%d\n",*it);
        if(*it>mylist.front()){
            int temp=mylist.front();
            mylist.pop_front();
            mylist.push_back(temp);
            count++;
            break;
        }
    }
```



```
            for(it=mylist.begin();it!=mylist.end();it++){
                //printf("%d\n",*it);
                if(*it>mylist.front()){
                    printed=false;
                    break;
                }
                else printed=true;
            }
            if(printed) {
                mylist.pop_front();
            }
        }

        printf("%d\n",count);
    }
    return 0;
}
```

1005. 射击游戏

Description

小明很喜欢玩射击游戏。他刚考完了数据结构期末考试，感觉不错，于是又来到了射击娱乐场放松一下。和上次一样，先从老板那租了一把步枪和装有 N 发子弹的弹夹。这里，再把规则说一遍。在射击的过程中，小明每次都有两种选择：从弹夹中取出一颗子弹上膛，或者打一发子弹出去。**注意：所有的子弹都从枪口上膛。**小明感觉这有点像《数据结构》课程中的“栈”的特点。因此在打完了这 N 发子弹之后，他想验证一下这些子弹打出来的顺序是不是真的满足“栈”的特性。假设 N 颗子弹的编号为 1,2,…,N。子弹从弹夹中取出的顺序也是从 1 到 N，这里 **N = 10**。给定一个子弹被打出的顺序，你可以帮小明验证它满不满足“栈”的打出顺序吗？

Input

可能有多个测试输入，第一行给出总共的测试输入的个数。

每个测试输入只有一行：用空格隔开的十个数，表示子弹打出的编号顺序。

Output

对每个测试输入，输出只有一行：

“Yes”，如果打出顺序满足“栈”的特点；

“No”，如果打出顺序不满足“栈”的特点。

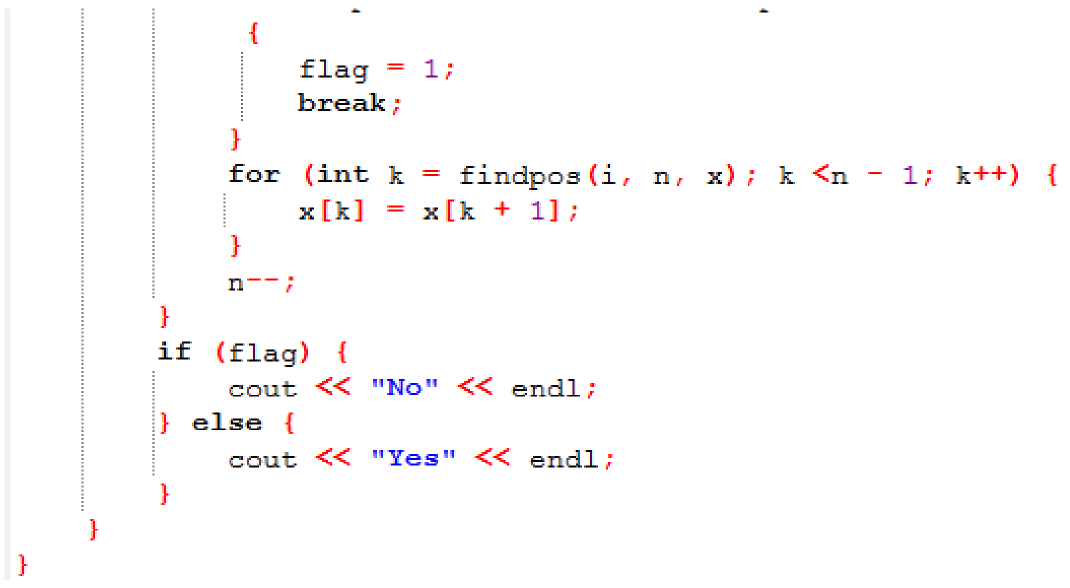
Sample Input

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

```
10 9 8 7 6 5 4 3 2 1
3 1 2 4 5 6 7 8 9 10
```
Sample Output
```
Yes
Yes
No
```

```
//在放第N+1颗子弹的时候，我们必须把它放在第N颗子弹的后面或者前面一位
//
//这样才能满足栈的输出过程。这样我们只要检测每递减一位的合法性就能很容易判断YESorNo了
//（9次判断即可），而不必去模拟栈的过程。

//每次检测只需检测本位的下一位的位置合法性即可，注意一点，每次判断完后记得删除这一位，
//至于原因大家模拟插入时的原理去理解就行了；
```

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

int findpos(int i, int n, int a[]) {
    int j;
    for (j = 0; j <n; j++) {
        if (a[j] == i) {
            break;
        }
    }
    return j;
}
int main() {
    int n;
    int t,  j;
    int x[10];
    cin >> t;
    while(t--) {
        n=10;
        int flag = 0;
        for (int i = 0; i<n; i++) {
            cin >> x[i];
        }
        for (int i = 10; i >= 2; i--) {
            if (findpos(i, n, x) < findpos(i - 1, n, x) &&
             x[findpos(i, n, x)] != x[findpos(i, n, x) + 1] + 1)
```



```
                {
                    flag = 1;
                    break;
                }
                for (int k = findpos(i, n, x); k <n - 1; k++) {
                    x[k] = x[k + 1];
                }
                n--;
            }
            if (flag) {
                cout << "No" << endl;
            } else {
                cout << "Yes" << endl;
            }
        }
}
```

1006. Maze

Description

为了测试某种药物对小白鼠方向感的影响，生物学家在实验室做了一个矩形迷宫，入口和出口都确定为唯一的，且分布在矩形的不同边上。现在让你算出小白鼠最短需要走多少步，才可以从入口走到出口。

Input

共 N+1 行，第一行为 N（N = 0，表示输入结束），以下 N 行 N 列 0-1 矩阵，1 表示不能通过,0 表示可以通过(左上角和右下角为 0，即入口和出口). N<30.

Output

只有一个数，为最少要走的格子数。0 表示没有路径。

Sample Input

```
5
0 1 1 1 1
0 0 1 1 1
1 0 0 0 1
1 1 1 0 1
1 1 1 0 0
4
0 0 1 1
0 0 0 1
1 1 1 1
1 1 1 0
0
```

Sample Output

9
0

```
//思路就是用结构体来记录迷宫中的每一点坐标和走到该点所需要的最短格子数
//用队列来装这个结构体，当队列为空时，即所有能走的点都已经走过一遍了。判断是否到达出口
//若是已经到达出口，就输出当前的最短路径

#include <cstdio>
#include <iostream>
#include <queue>
using namespace std;

struct point{
        int x;//横坐标
        int y;//纵坐标
        int step;//走到该点的最短路径
};

int main(){
        int n,i,j;
        int maze[101][101];
        point current,next;
        queue<point> q;
        while(cin>>n&&n!=0){
                while(!q.empty()) q.pop();
                for(i=0;i<n;i++)
                   for(j=0;j<n;j++){
                          cin>>maze[i][j];
                   }
                current.x=current.y=0;
                current.step=1;
                maze[0][0]=1;//为了避免走回头路
                q.push(current);
                while(!q.empty()){
                        current=q.front();
                        if(current.x==n-1&&current.y==n-1)break;

        if(current.x-1>=0&&maze[current.x-1][current.y]==0){//向上走
                                next.x=current.x-1;
                                next.y=current.y;
                                next.step=current.step+1;
                                maze[current.x-1][current.y]=1;
                                q.push(next);
```

```
                    }

        if(current.y-1>=0&&maze[current.x][current.y-1]==0){//向左走
                            next.x=current.x;
                            next.y=current.y-1;
                            next.step=current.step+1;
                            maze[current.x][current.y-1]=1;
                            q.push(next);
                    }

        if(current.x+1<n&&maze[current.x+1][current.y]==0){//向下走
                            next.x=current.x+1;
                            next.y=current.y;
                            next.step=current.step+1;
                            maze[current.x+1][current.y]=1;
                            q.push(next);
                    }

        if(current.y+1<n&&maze[current.x][current.y+1]==0){//向右走
                            next.x=current.x;
                            next.y=current.y+1;
                            next.step=current.step+1;
                            maze[current.x][current.y+1]=1;
                            q.push(next);
                    }
                    q.pop();
       }

if(current.x==n-1&&current.y==n-1)printf("%d\n",q.front().step);
       else printf("%d\n",0);
        }

        return 0;
}
```

1007. Translation

Description

You have just moved from Waterloo to a big city. The people here speak an incomprehensible dialect of a foreign language. Fortunately, you have a dictionary to help you understand them.

Input

Input consists of up to 100,000 dictionary entries, followed by a blank line, followed by a message of up to 100,000 words. Each dictionary entry is a line containing an English word, followed by a space and a foreign language word. No foreign word appears more than once in the dictionary. The message is a sequence of words in the foreign language, one word on each line. Each word in the input is a sequence of at most 10 lowercase letters.

Output

Output is the message translated to English, one word per line. Foreign words not in the dictionary should be translated as "eh".

Sample Input

```
dog ogday
cat atcay
pig igpay
froot ootfray
loops oopslay

atcay
ittenkay
oopslay
```

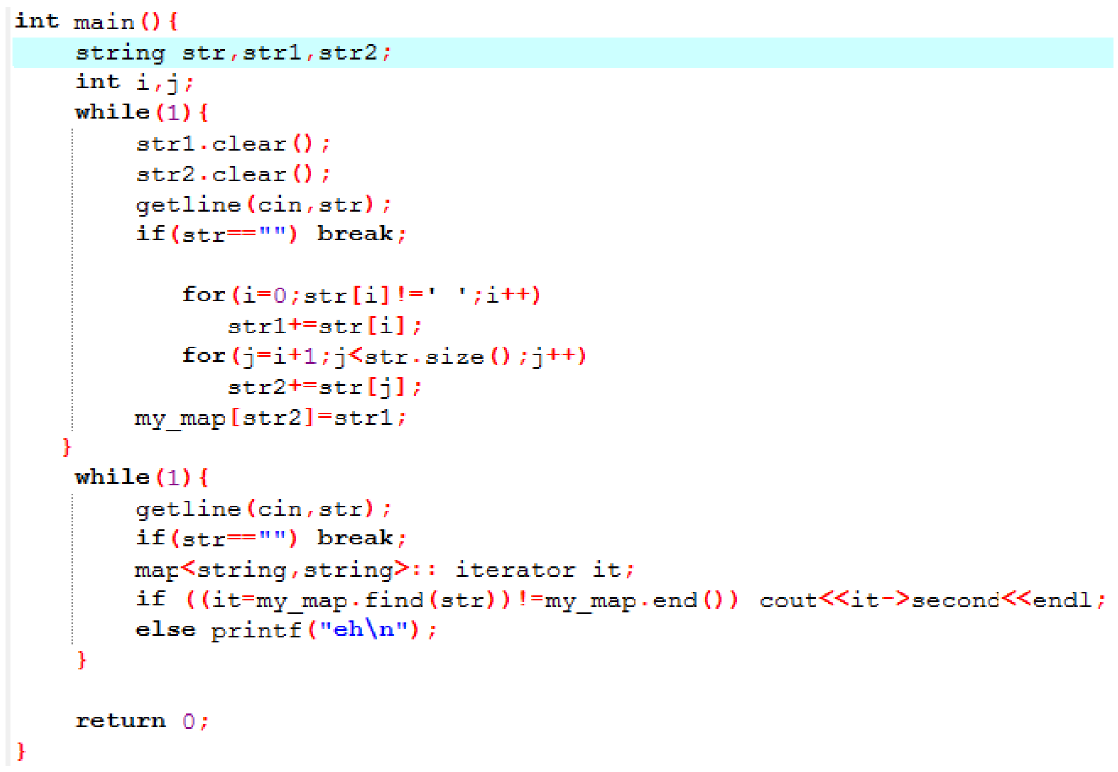
Sample Output

```
cat
eh
loops
```

Hint

Huge input and output, scanf and printf are recommended.

```
#include <map>
#include <cstring>
using namespace std;
//gets可以无限读取，不会判断结束，以回车结束读取
//string中对+进行了重载a+b=ab
//map<key,value>中key值不可以被修改，value可以被修改,即map[key]=value
```



```
int main(){
    string str,str1,str2;
    int i,j;
    while(1){
        str1.clear();
        str2.clear();
        getline(cin,str);
        if(str=="") break;

           for(i=0;str[i]!=' ';i++)
              str1+=str[i];
           for(j=i+1;j<str.size();j++)
              str2+=str[j];
        my_map[str2]=str1;
   }
    while(1){
        getline(cin,str);
        if(str=="") break;
        map<string,string>:: iterator it;
        if ((it=my_map.find(str))!=my_map.end()) cout<<it->second<<endl;
        else printf("eh\n");
    }

    return 0;
}
```