图算法例题

图算法例题

1000. sicily 1155. Can I Post the lette

Time Limit: 1sec Memory Limit:32MB

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
```

Source Code

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

int n,m;
vector<int> vout[200];
bool visited[200];

bool flood(int u)
{
    visited[u]=1;
    if (u==n-1) return 1;
    for (int x=0; x<vout[u].size(); x++)
    {
        int &v=vout[u][x];
        if (!visited[v] && flood(v)) return 1;
    }
    return 0;
}

int main()
{
    while (cin>>n, n)
    {
        fill(vout,vout+n,vector<int>());
        fill(visited,visited+n,0);

        cin>>m;
        while (m--)
        {
            int u,v;
            cin>>u>>v;
            vout[u].push_back(v);
        }

        cout<<(flood(0)?"I can post the letter\n":"I can't post the
letter\n");
    }

    return 0;
}
```

1001. 无路可逃？

Time Limit: 1sec Memory Limit:256MB

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
```

Source Code

```
#include <cstdio>
#include <cstring>
#include <queue>
```

```
using namespace std;
const int MAXN = 110;
int maze[MAXN][MAXN];
struct point {
    int x, y;
    point(int a=0, int b=0) : x(a), y(b) {}
};
int dx[4] = {1, 0, -1, 0};
int dy[4] = {0, 1, 0, -1};
int main()
{
    int i, j, t, sx, sy, tx, ty, head, tail, n, m;
    scanf("%d", &t);
    while (t--)
    {
        queue<point> cq;
        scanf("%d%d", &n, &m);
        for (i=0;i<n;++i) {
            for (j=0;j<m;++j) scanf("%d", &maze[i][j]);
        }
        scanf("%d%d%d%d", &sx, &sy, &tx, &ty);
        --sx; --sy; --tx; --ty;
        maze[sx][sy] = -1;
        cq.push(point(sx,sy));
        while (!cq.empty())
        {
            point cur = cq.front();
            cq.pop();
            for (i=0;i<4;++i) {
                point tar = cur;
                tar.x += dx[i];
                tar.y += dy[i];
                if (tar.x < 0 || tar.x >= n || tar.y < 0 || tar.y >= m)
continue;
                if (maze[tar.x][tar.y] <= 0) continue;
                maze[tar.x][tar.y] = -1;
                cq.push(tar);
            }
        }
        if (maze[tx][ty] >= 0) puts("0"); else puts("1");
    }
    return 0;
}
```

1002. connect components in undirected graph

Time Limit: 1sec Memory Limit:256MB

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

Source Code

```
#include <queue>
#include <vector>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
using namespace std;
const int MAXN = 110;
vector<int> graph[MAXN];
int vis[MAXN];
int n, m, u, v;
//int dist[MAXN];
int cc;


void bfs(int i)
{
   queue<int> cq;
   vis[i] = 1;
```

```
    cq.push(i); //从顶点 i 开始遍历
    while(!cq.empty())
    {
        u = cq.front(); cq.pop();
        for(int i=0;i<graph[u].size();i++)
        {
            v = graph[u][i];
            if (vis[v]==1) continue;
            vis[v] = 1;
            cq.push(v);
        }
    }
}

void bfs()
{
    memset(vis,0,sizeof(vis));
    cc = 0; //初始设置连通块数目 cc=0
    for (int i=1; i<=n; i++)
        if (vis[i]==0)
        {
            ++cc;
            bfs(i);
        }
}


int main()
{
    cin >> n >> m;
    for (int i=1; i<=m; i++)
    {
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    bfs();
    cout << cc << endl;
    return 0;
}
```

1003. bicoloring

Time Limit: 1sec Memory Limit:256MB

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

Source Code

```
#include <queue>
#include <vector>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
using namespace std;
const int MAXN = 1010;
vector<int> graph[MAXN];
int vis[MAXN];     // 0 表示未访问，1 表示访问过
int n, m, u, v;
int color[MAXN]; // 0 表示未染色，1 表示染为红色，-1 表示染为黑色


bool bfs(int i)
{
    queue<int> cq;
```

```
    vis[i] = 1;
    color[i] = 1;
    cq.push(i); //从顶点 i 开始遍历
    while(!cq.empty())
    {
        u = cq.front(); cq.pop();
        for(int i=0;i<graph[u].size();i++)
        {
            v = graph[u][i];
            if (vis[v]==1)
            {
                if (color[v] == -color[u]) continue;
                else return false;
            }
            vis[v] = 1;
            color[v] = -color[u];
            cq.push(v);
        }
    }
    return true;
}

int main()
{
    memset(vis,0,sizeof(vis));
    memset(color,0,sizeof(color));

    cin >> n >> m;
    for (int i=1; i<=m; i++)
    {
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    cout << (bfs(1) ? "yes" : "no") << endl;
    return 0;
}
```

1004. Knight Moves

Time Limit: 1sec Memory Limit:32MB

Description

A friend of you is doing research on the Traveling Knight Problem (TKP) where you are to find the shortest closed tour of knight moves that visits each square of a given set of n squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy. Of course you know that it is vice versa. So you offer him to write a program that solves the "difficult" part. Your job is to write a program that takes two squares a and b as input and then determines the number of knight moves on a shortest route from a to b.

Input

There are multiple test cases. The first line contains an integer T, indicating the number of test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter (a-h) representing the column and a digit (1-8) representing the row on the chessboard.

Output

For each test case, print one line saying "To get from xx to yy takes n knight moves.".

Sample Input

```
8
e2 e4
a1 b2
b2 c3
a1 h8
a1 h7
h8 a1
b1 c3
f6 f6
```

Sample Output

```
To get from e2 to e4 takes 2 knight moves.
To get from a1 to b2 takes 4 knight moves.
To get from b2 to c3 takes 2 knight moves.
To get from a1 to h8 takes 6 knight moves.
To get from a1 to h7 takes 5 knight moves.
To get from h8 to a1 takes 6 knight moves.
To get from b1 to c3 takes 1 knight moves.
To get from f6 to f6 takes 0 knight moves.
```

Source Code

```
#include <stdio.h>
#include <memory.h>
#include <vector>
```

```
using namespace std;

const int MAXN = 8;
int front,rear,minstep;
int color[9][9];
const int searchTable[8][2] = {{-1,-2},{-2,-1},{-2,1},{-1,2},{1,2},{2,1},{2,-1},{1,-2}};

struct node
{
	int x,y,step;
}que[100],temp;

int bfs(int sx,int sy,int ex,int ey,int step)
{
	if(sx == ex && sy == ey)
	{
		minstep = step;
		return 0;
	}
	struct node q;
	color[sx][sy] = 1;
	temp.x = sx;
	temp.y = sy;
	temp.step = step;
	que[rear++] = temp;
	while(front < rear)
	{
		temp = que[front++];
		for(int i = 0;i < 8;i++)
		{
			q.x = temp.x + searchTable[i][0];
			q.y = temp.y + searchTable[i][1];
			q.step = temp.step + 1;
			if(q.x < 1 || q.y < 1 || q.x > 8 || q.y > 8 || color[q.x][q.y]) continue;
			if(q.x == ex && q.y == ey)
			{
				minstep = q.step;
				return 0;
			}
			que[rear++] = q;
			color[q.x][q.y] = 1;
		}
	}
}
```

```
int main()
{
	char ch1[3],ch2[3];
	int sx,sy,ex,ey,t;
	scanf("%d",&t);
	while(t--)
	{
		minstep = 0;
		scanf("%s%s",&ch1,&ch2);
		memset(color,0,sizeof(color));
		memset(que,0,sizeof(que));
		front = rear = 0;
		sx = ch1[0] - 'a' + 1;
		sy = ch1[1] - '0';
		ex = ch2[0] - 'a' + 1;
		ey = ch2[1] - '0';
		bfs(sx,sy,ex,ey,0);
		printf("To get from %s to %s takes %d knight moves.\n",ch1,ch2,minstep);
	}
  //    system("pause");
	return 0;
}
```

1005. 有向图边的分类

Time Limit: 1sec Memory Limit:256MB

Description

输入一个有向图，从顶点 1 开始做 dfs 对边进行分类。

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。
接下来的 1 行，包含一个整数 k，表示会查询 k 条边的类型。
接下来的 k 行，每行是一个数对 u v，表示查询边 u v 的类型。

Output

对每条查询的边，单独一行输出边的类型，参见输出样例。

Sample Input

4 6

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

Sample Output

edge (1,2) is Tree Edge
edge (3,1) is Back Edge
edge (1,3) is Down Edge
edge (4,2) is Cross Edge

Source Code

```
#include <cstdio>
#include <cstring>
using namespace std;
const int V = 1110;
int edge[V][V], parent[V], pre[V], post[V], tag;

int n, m, k, u, v;
void read()
{
  memset(edge,0,sizeof(edge));
  scanf("%d%d",&n,&m);
  for(int i=0;i<m;i++)
  {
    scanf("%d%d",&u,&v);
    edge[u][v] = 1;
  }
}

/*
DAG 的深度有限搜索标记
INIT: edge[][]邻接矩阵；pre[], post[], tag 全置 0;
CALL: dfstag(i,n); pre/post: 开始/结束时间
*/
void dfstag(int cur, int n)
{
```

```
  // vertex 1 ~ n
  pre[cur] = ++tag;
  for (int i=1; i<=n; ++i) if (edge[cur][i]==1) {
    if (0 == pre[i]) {
      parent[i] = cur;
      edge[cur][i]=-1;//printf("edge (%d,%d) is Tree Edge\n",cur,i);
      dfstag(i,n);
    } else {
      if (0 == post[i]) edge[cur][i]=-2;//printf("edge (%d,%d) is Back
Edge\n",cur,i);
      else if (pre[i] > pre[cur]) edge[cur][i]=-3;//printf("edge
(%d,%d) is Down Edge\n",cur,i);
      else   edge[cur][i]=-4;//printf("edge   (%d,%d)   is   Cross
Edge\n",cur,i);
    }
  }
  post[cur] = ++tag;
}

int dfs(int n)
{
  memset(parent,-1,sizeof(pre));
  memset(pre,0,sizeof(pre));
  memset(post,0,sizeof(post));
  tag = 0;
  for(int i=1; i<=n; i++)
    if (parent[i]==-1) dfstag(i,n);
}
//char* edgetype[] = {"Tree Edge","Back Edge","Down Edge","Cross
Edge"};


void print(int a[])
{
  for(int i=1; i<=n; i++)
    printf("%d ",a[i]);
  printf("\n");
}
int main()
{
  memset(edge,0,sizeof(edge));
  scanf("%d%d",&n,&m);
  for(int i=0;i<m;i++)
  {
```

```
    scanf("%d%d",&u,&v);
    edge[u][v] = 1;
  }
  dfs(n);
  scanf("%d",&k);
  for(int i=0; i<k; i++)
  {
    scanf("%d%d",&u,&v);
    if (edge[u][v]==-1) printf("edge (%d,%d) is Tree Edge\n",u,v);
    else if (edge[u][v]==-2) printf("edge (%d,%d) is Back
Edge\n",u,v);
    else if (edge[u][v]==-3) printf("edge (%d,%d) is Down
Edge\n",u,v);
    else if (edge[u][v]==-4) printf("edge (%d,%d) is Cross
Edge\n",u,v);
  }
  return 0;
}
```

1006. DAG?

Time Limit: 1sec Memory Limit:256MB

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

Source Code

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int V = 1110;
int edge[V][V], parent[V], pre[V], post[V], tag;
bool isDAG;
int n, m, u, v;
void read()
{
  memset(edge,0,sizeof(edge));
  scanf("%d%d",&n,&m);
  for(int i=0;i<m;i++)
  {
    scanf("%d%d",&u,&v);
    edge[u][v] = 1;
  }
}

/*
DAG 的深度有限搜索标记
INIT: edge[][]邻接矩阵；pre[], post[], tag 全置 0;
CALL: dfstag(i,n); pre/post: 开始/结束时间
*/
void dfstag(int cur, int n)
{
  // vertex 1 ~ n
  pre[cur] = ++tag;
  for (int i=1; i<=n; ++i) if (edge[cur][i]) {
    if (0 == pre[i]) {
      parent[i] = cur;
      //printf("edge (%d,%d) is Tree Edge\n",cur,i);
      dfstag(i,n);
    } else {
      if (0 == post[i]) isDAG=false;//printf("edge (%d,%d) is Back
Edge\n",cur,i);
      else if (pre[i] > pre[cur]) ;//printf("edge (%d,%d) is Down
Edge\n",cur,i);
      else ;//printf("edge (%d,%d) is Cross Edge\n",cur,i);
    }
  }
  post[cur] = ++tag;
}
```

```
int dfs(int n)
{
  memset(parent,-1,sizeof(pre));
  memset(pre,0,sizeof(pre));
  memset(post,0,sizeof(post));
  tag = 0;
  for(int i=1; i<=n; i++)
    if (parent[i]==-1) dfstag(i,n);
}

int main()
{
  read();
  isDAG=true;
  dfs(n);
  cout <<isDAG << endl;
}
```

1007. sicily . Magic Island

Time Limit: 1sec Memory Limit:32MB

Description

There are N cities and N-1 roads in Magic-Island. You can go from one city to any other. One road only connects two cities. One day, The king of magic-island want to visit the island from the capital. No road is visited twice. Do you know the longest distance the king can go.

Input

There are several test cases in the input

A test case starts with two numbers N and K. (1<=N<=10000, 1<=K<=N). The cities is denoted from 1 to N. K is the capital.

The next N-1 lines each contain three numbers X, Y, D, meaning that there is a road between city-X and city-Y and the distance of the road is D. D is a positive integer which is not bigger than 1000.

Input will be ended by the end of file.

Output

One number per line for each test case, the longest distance the king can go.

Sample Input

3 1

1 2 10

1 3 20

Sample Output

20

Source Code

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

const int inf=1000000000;
const int N=10001;

int n,s;
int d[N];
vector<int> vout[N];
vector<int> eout[N];

int best[N];
queue<int> q;

int main()
{
    //freopen("1024.in","r",stdin);

    while (scanf("%d%d",&n,&s)!=EOF)
    {
        s--;
        fill(vout,vout+n,vector<int>());
        fill(eout,eout+n,vector<int>());

        for (int e=0; e<n-1; e++)
        {
            int u,v;
            scanf("%d%d%d",&u,&v,&d[e]);
            u--; v--;
            vout[u].push_back(v);
            vout[v].push_back(u);
            eout[u].push_back(e);
            eout[v].push_back(e);
        }

        fill(best,best+n,inf);
```

```
        q.push(s);
        best[s]=0;
        while (!q.empty())
        {
            int u=q.front();
            for (int x=0; x<vout[u].size(); x++)
            {
                int v=vout[u][x];
                int e=eout[u][x];
                if (best[v] > best[u]+d[e])
                {
                    best[v] = best[u]+d[e];
                    q.push(v);
                }
            }
            q.pop();
        }

        printf("%d\n",*max_element(best,best+n));
    }

    //while(1);
    return 0;
}
```

1008. Forest

Time Limit: 1sec Memory Limit:32MB

Description

In the field of computer science, forest is important and deeply researched , it is a model for many data structures . Now it's your job here to calculate the depth and width of given forests.

Precisely, a forest here is a directed graph with neither loop nor two edges pointing to the same node. Nodes with no edge pointing to are roots, we define that roots are at level 0 . If there's an edge points from node A to node B , then node B is called a child of node A , and we define that B is at level (k+1) if and only if A is at level k .

We define the depth of a forest is the maximum level number of all the nodes , the width of a forest is the maximum number of nodes at the same level.

Input

There're several test cases. For each case, in the first line there are two integer numbers n and m ($1 \leqslant n \leqslant 100$, $0 \leqslant m \leqslant 100$, $m \leqslant n*n$) indicating the number of nodes and edges respectively , then

m lines followed , for each line of these m lines there are two integer numbers a and b (1⩽a,b⩽ n)indicating there’s an edge pointing from a to b. Nodes are represented by numbers between 1 and n .n=0 indicates end of input.

Output

For each case output one line of answer , if it’s not a forest , i.e. there’s at least one loop or two edges pointing to the same node, output “INVALID”(without quotation mark), otherwise output the depth and width of the forest, separated by a white space.

Sample Input

```
1 0
1 1
1 1
3 1
1 3
2 2
1 2
2 1
0 88
```

Sample Output

```
0 1
INVALID
1 2
INVALID
```

Source Code

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <vector>
#include <queue>
#include <algorithm>
#include <utility>
using namespace std;

const int inf = 1000000000;
const int N = 100;

int n, m;

int deg[N];
vector<int> vout[N];
```

```
int d[N];
int w[N];

bool run(int& depth, int& width)
{
    //check degree
    for (int v=0; v<n; v++)
        if (deg[v]>1) return 0;

    //init
    fill(d, d+n, inf);
    fill(w, w+n, 0);
    depth = 0;

    //bfs
    for (int s=0; s<n; s++)
    {
        if (deg[s]==0)
        {
            d[s] = 0;
            w[0]++;
            queue<int> q;
            q.push(s);
            while (!q.empty())
            {
                int u = q.front();
                for (int x=0; x<vout[u].size(); x++)
                {
                    int v = vout[u][x];
                    d[v] = d[u] + 1;
                    w[d[v]]++;
                    depth = max(depth, d[v]);
                    q.push(v);
                }
                q.pop();
            }
        }
    }

    //check loop
    for (int v=0; v<n; v++)
        if (d[v] >= inf) return 0;
```

```
    //find width
    width = *max_element(w, w+depth+1);

    return 1;
}

int main()
{
    while (scanf("%d%d", &n, &m), n)
    {
        //init
        memset(deg, 0, n*sizeof(int));
        fill(vout, vout+n, vector<int>());

        //add edges
        while (m--)
        {
            int u, v;
            scanf("%d%d", &u, &v);
            u--;    v--;
            vout[u].push_back(v);
            deg[v]++;
        }

        //run
        int depth, width;
        if (run(depth, width))
        {
            printf("%d %d\n", depth, width);
        }
        else
        {
            printf("INVALID\n");
        }
    }

    return 0;
}
```

1009. sicily 1424. 奖金

Time Limit: 1sec Memory Limit:32MB

Description

由于无敌的凡凡在2005年世界英俊帅气男总决选中胜出，Yali Company总经理Mr.Z心情好，决定给每位员工发奖金。公司决定以每个人本年在公司的贡献为标准来计算他们得到奖金的多少。

于是Mr.Z下令召开m方会谈。每位参加会谈的代表提出了自己的意见：“我认为员工a的奖金应该比b高！”Mr.Z决定要找出一种奖金方案，满足各位代表的意见，且同时使得总奖金数最少。每位员工奖金最少为100元。

数据满足n<=10000，m<=20000。

Input

第一行两个整数n,m，表示员工总数和代表数；

以下m行，每行2个整数a,b，表示某个代表认为第a号员工奖金应该比第b号员工高。

Output

若无法找到合法方案，则输出“Poor Xed”；否则输出一个数表示最少总奖金。

Sample Input

2 1

1 2

Sample Output

201

Source Code

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

const int N=10000;
int n,m;
vector<int> vout[N];
int vin_n[N];
bool visited[N],checked[N];
int f[N];
bool inQ[N];
queue<int> q;

bool loop(int i)
{
    if (visited[i]) return 1;
    if (checked[i]) return 0;
```

```
    visited[i]=1;
    for (int x=0; x<vout[i].size(); x++)
    {
        int& j=vout[i][x];
        if (loop(j)) return 1;
    }
    visited[i]=0;
    checked[i]=1;
    return 0;
}

bool valid()
{
    fill(checked,checked+N,0);
    fill(visited,visited+N,0);
    for (int i=0; i<n; i++)
    {
        if (loop(i)) return 0;
    }
    return 1;
}

void flood()
{
    fill(f,f+n,0);
    fill(inQ,inQ+n,0);

    for (int i=0; i<n; i++)
    {
        if (vin_n[i]==0)
        {
            q.push(i);
            inQ[i]=1;
        }
    }
    while (!q.empty())
    {
        int u=q.front();
        for (int x=0; x<vout[u].size(); x++)
        {
            int v=vout[u][x];
            if (f[v] < f[u]+1)
            {
```

```
                f[v] = f[u]+1;
                if (!inQ[v])
                {
                    q.push(v);
                    inQ[v]=1;
                }
            }
        }
        q.pop();
        inQ[u]=0;
    }
}

int main()
{
    scanf("%d%d",&n,&m);
    fill(vin_n,vin_n+n,0);
    for (int i=0; i<m; i++)
    {
        int a,b;
        scanf("%d%d",&a,&b);
        a--;
        b--;
        vout[b].push_back(a);
        vin_n[a]++;
    }

    if (valid())
    {
        flood();
        int sum=n*100;
        for (int i=0; i<n; i++)
            sum+=f[i];
        printf("%d\n", sum);
    }
    else
    {
        printf("Poor Xed\n");
    }

    return 0;
}
```

1011. sicily 1090. Highways

Time Limit: 1sec Memory Limit:32MB

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

Source Code

```
#include <iostream>
```

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

const int N = 500;

int n;
int d[N][N];

class edge
{
public:
    int u, v;
    edge(int u, int v)
    {
        this->u = u;
        this->v = v;
    }
};

bool operator<(const edge& a, const edge& b)
{
    return d[a.u][a.v] > d[b.u][b.v];
}

bool t[N];
priority_queue<edge> q;

void take(int u)
{
    t[u] = 1;
    for (int v=0; v<n; v++)
    {
        if (!t[v]) q.push(edge(u, v));
    }
}

int main()
{
    int T;  scanf("%d", &T);
    for (int c=0; c<T; c++)
    {
        //input
        scanf("%d", &n);
        for (int u=0; u<n; u++)
```

```
        for (int v=0; v<n; v++)
            scanf("%d", &d[u][v]);

        //initialize
        memset(t, 0, sizeof(t));
        q = priority_queue<edge>();

        //prim
        take(0);
        int dmax = 0;
        for (int e=0; e<n-1; e++)
        {
            int u, v;
            do
            {
                u = q.top().u;
                v = q.top().v;
                q.pop();
            } while (t[u] && t[v]);

            take(!t[u] ? u : v);
            dmax = max(dmax, d[u][v]);
        }

        //output
        if (c) printf("\n");
        printf("%d\n", dmax);
    }

    return 0;
}
```

```
//Kruskal for MST
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class edge
{
public:
    int u, v, w;
    edge(int u, int v, int w)
    {
```

```
        this->u = u;
        this->v = v;
        this->w = w;
    }
};

bool operator<(const edge& a, const edge& b)
{
    return a.w < b.w;
}

const int N = 500;

int n;
vector<edge> e;

int p[N], r[N];

int root(int x)
{
    if (p[x] == x) return x;

    return p[x] = root(p[x]);
}

void merge(int x, int y)
{
    x = root(x);
    y = root(y);

    if (r[x] < r[y])
    {
        p[x] = y;
    }
    else if (r[x] > r[y])
    {
        p[y] = x;
    }
    else
    {
        p[y] = x;
        r[x]++;
    }
}
```

```
int main()
{
    int T;  scanf("%d", &T);
    for (int c=0; c<T; c++)
    {
        //clear
        e.clear();

        //input
        scanf("%d", &n);
        for (int u=0; u<n; u++)
        for (int v=0; v<n; v++)
        {
            int w;
            scanf("%d", &w);
            if (u < v) e.push_back(edge(u, v, w));
        }

        //sort edges
        sort(e.begin(), e.end());

        //init forest
        for (int x=0; x<n; x++)
        {
            p[x] = x;
            r[x] = 0;
        }

        //kruskal
        int wmax = 0;
        int k = 0;
        for (int i=0; k<n-1 && i<e.size(); i++)
        {
            int u = e[i].u;
            int v = e[i].v;
            int w = e[i].w;
            if (root(u) != root(v))
            {
                merge(u, v);
                k++;
                wmax = w;
            }
        }
```

```
        //output
        if (c) printf("\n");
        printf("%d\n", wmax);
    }

    return 0;
}
```

1012. Robot

Time Limit: 1sec Memory Limit:32MB

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.

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

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 < row100 , 0 < column100

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

Source Code

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

```
const int inf = 1000000000;
const int N = 100;
const int dy[4]={0,1,0,-1};
const int dx[4]={1,0,-1,0};

int n,m,a[N][N];
int sx,sy,tx,ty;
queue<int> q;
int best[N][N];

bool inCourt(int y, int x)
{
    return 0<=y && y<n && 0<=x && x<m;
}

int main()
{



    int T;
    cin>>T;
    while (T--)
    {
        cin>>n>>m;
        for (int i=0; i<n; i++)
            for (int j=0; j<m; j++)
            {
                cin>>a[i][j];
                best[i][j]=inf;
            }
        //cin>>sx>>sy>>tx>>ty;
        cin>>sy>>sx>>ty>>tx;
        sx--; sy--; tx--; ty--;

        q.push(sy*m+sx);
        best[sy][sx]=a[sy][sx];
        while (!q.empty())
        {
            int u=q.front();
            int y=u/m;
            int x=u%m;
            for (int d=0; d<4; d++)
            {
```

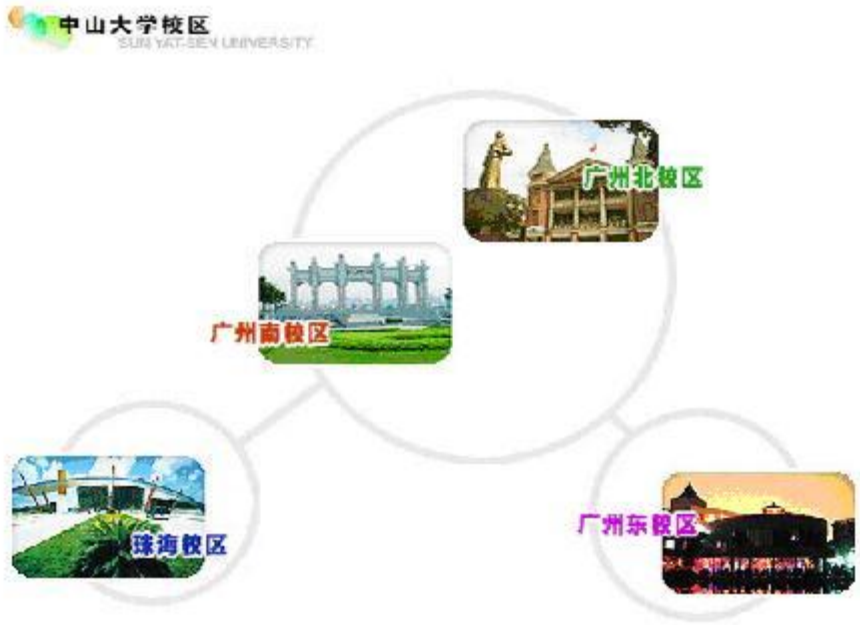
```
            int y1=y+dy[d];
            int x1=x+dx[d];
            if     (inCourt(y1,x1)     &&     best[y1][x1]     >
best[y][x]+a[y1][x1])
            {
                best[y1][x1] = best[y][x]+a[y1][x1];
                q.push(y1*m+x1);
            }
        }
        q.pop();
    }
    cout<<best[ty][tx]<<endl;
}

    return 0;
}
```

1013. sicily 1031. Campus

Time Limit: 1sec Memory Limit:32MB

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

Source Code

```
#include <iostream>
#include <string>
#include <map>
#include <queue>
#include <vector>
#include <algorithm>
using namespace std;

const int N=202, M=100;
int n,m,s,t;
map<string,int> f;
int d[N][N];

vector<int> vout[N];

int best[N];
queue<int> q;

int main()
{
    int T;
    cin>>T;
    while (T--)
    {
        //initialize
        f.clear();
        fill(vout,vout+N,vector<int>());

        //input
        string a,b;
        cin>>m;
        while (m--)
        {
            cin>>a>>b;
            if (f.find(a)==f.end()) f[a]=f.size()-1;
            if (f.find(b)==f.end()) f[b]=f.size()-1;
            int u,v;
            u=f[a];
            v=f[b];
```

```
            vout[u].push_back(v);
            vout[v].push_back(u);
            cin>>d[u][v];
            d[v][u]=d[u][v];
        }

        cin>>a>>b;
        if (f.find(a)==f.end()) f[a]=f.size()-1;
        if (f.find(b)==f.end()) f[b]=f.size()-1;
        s=f[a];
        t=f[b];

        n=f.size();

        //initialize
        fill(best,best+n,INT_MAX);
        best[s]=0;

        //bfs
        q.push(s);
        while (!q.empty())
        {
            int u=q.front();
            for (int x=0; x<vout[u].size(); x++)
            {
                int v=vout[u][x];
                if (best[v] > best[u]+d[u][v])
                {
                    best[v] = best[u]+d[u][v];
                    q.push(v);
                }
            }
            q.pop();
        }

        cout<<(best[t]!=INT_MAX ? best[t] : -1)<<endl;
    }

    return 0;
}
```