Chapter 5 Graph Theory



Keshuai

2015-11-07

Copyright ：1.0

目录

一．基础图论

（一）拓扑排序

（二）欧拉路

二．深度优先遍历

（一）无向图割顶和桥

（二）无向图双联通分量

（三）有向图强联通分量

（四）2-SAT

例题：uva 12273（2sat变形+限制条件）

三．最短路X

四．生成树

（一）最小生成树

（二）次小生成树

（三）最小有向生成树

五．二分图X

（一）二分图判定X

（二）最大匹配X

（三）最佳完美匹配X

（四）稳定婚姻问题X

例题：uva 1175（稳定婚姻问题）

六．网络流X

一、基础图论

（一）拓扑排序

对一个有向无环图(Directed Acyclic Graph简称DAG)G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)，则u在线性序列中出现在v之前。通常，这样的线性序列称为满足拓扑次序(Topological Order)的序列，简称拓扑序列。简单的说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为拓扑排序。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TopSort.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <queue>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

int in[maxn];

void topSort(int\* a, int n, vector<int>\* g) {

memset(in, 0, sizeof(in));

for (int i = 0; i < n; i++) {

for (int j = 0; j < g[i].size(); j++) {

int v = g[i][j];

in[v]++;

}

}

queue<int> que;

for (int i = 0; i < n; i++) if (!in[i])

que.push(i);

int mv = n-1;

while (!que.empty()) {

int u = que.front();

que.pop();

a[mv--] = u;

for (int i = 0; i < g[u].size(); i++) {

int v = g[u][i];

in[v]--;

if (in[v] == 0) que.push(v);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（二）欧拉路

如果给定无孤立结点图G，若存在一条路，经过图中每边一次且仅一次，这条路称为欧拉路；

如果给定无孤立结点图G，若存在一条回路，经过图中每边一次且仅一次，那么该回路称为欧拉回路。

欧拉路的判定：图联通，并且至多有两个点的度数为奇数，其它点的度数为偶数。

欧拉回路的判定：图联通，点的度数为偶数。

路径输出：Feurly算法。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Feurly.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

typedef pair<int,int> pii;

const int maxn = 1005;

int top = 0, vis[maxn], path[maxn];

vector<pii> G[maxn];

void feurly (int u) {

for (int i = 0; i < G[u].size(); i++) {

int e = G[u][i].first, v = G[u][i].second;

if (vis[e]) continue;

vis[e] = 1;

feurly(v);

path[top++] = e;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

二、深度优先遍历

（一）无向图割顶和桥

1、割顶：

对于无向图G，如果删除某个点u后，连通分量数目增加，称u为图的关节点（articulation vertex）或割顶（cut vertex）。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CutPoint.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5; // The number of Node

int dfsclock, pre[maxn];

bool iscut[maxn];

vector<int> G[maxn];

int dfs (int u, int f) {

int lowu = pre[u] = ++dfsclock, child = 0;

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if (!pre[v]) {

child++;

int lowv = dfs(v, u);

lowu = min(lowu, lowv);

if (lowv >= pre[u]) iscut[u] = true;

} else if (pre[v] < pre[u] && v != f)

lowu = min(lowu, pre[v]);

}

if (f < 0 && child == 1) iscut[u] = false;

return lowu;

}

void findCutPoint(int n) {

dfsclock = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

for (int i = 1; i <= n; i++)

if (!pre[i]) dfs(i, -1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2、桥：

对于无向图G，如果删除某条边e后，连通分量数目增加，称e为图的桥（bridge）或割顶（cut edge）。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CutPoint.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5; // The number of Node

const int maxm = 1e6 + 5; // The number of Edge

int dfsclock, pre[maxn];

bool iscut[maxm \* 2];

int first[maxn], jump[maxm \* 2], linker[maxm \* 2];

int dfs (int u, int fa) {

int lowu = pre[u] = ++dfsclock;

for (int i = first[u]; i != -1; i = jump[i]) {

int v = linker[i];

if (!pre[v]) {

int lowv = dfs(v, u);

lowu = min(lowu, lowv);

if (lowv > pre[u]) iscut[i] = iscut[i^1] = true; // 正反向边

} else if (pre[v] < pre[u] && v != fa)

lowu = min(lowu, pre[v]);

}

return lowu;

}

void findEdge (int n) {

dfsclock = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

for (int i = 1; i <= n; i++)

if (!pre[i]) dfs(i, -1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（二）无向图双连通分量

1、点双连通：

对于一个连通图，如果任意两点至少存在两条点不重复的路径，则称这个图是点-双连通的。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Point-biconnected.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

struct Edge {

int from, to, val;

Edge(int from = 0, int to = 0, int val = 0): from(from), to(to), val(val) {}

};

int dfsclock, cntbcc, pre[maxn], bccno[maxn];

bool iscut[maxn];

vector<int> G[maxn], BCC[maxn];

stack<Edge> S;

int dfs (int u, int fa) { // 割顶的bccno无意义

int lowu = pre[u] = ++dfsclock, child = 0;

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

Edge e = Edge(u, v);

if (!pre[v]) {

child++;

S.push(e);

int lowv = min(lowu, lowv);

if (lowv >= pre[u]) {

iscut[u] = 1;

BCC[++cntbcc].clear();

while (true) {

Edge x = S.top();

S.pop();

if (bccno[x.from] != cntbcc) {

BCC[cntbcc].push\_back(x.from);

bccno[x.from] = cntbcc;

}

if (bccno[x.to] != cntbcc) {

BCC[cntbcc].push\_back(x.to);

bccno[x.to] = cntbcc;

}

if (x.from == u && x.to == v) break;

}

}

} else if (pre[v] < pre[u] && v != fa) {

S.push(e);

lowu = min(lowu, pre[v]);

}

}

if (fa < 0 && child == 1) iscut[u] = 0;

return lowu;

}

void findBCC(int n) {

dfsclock = cntbcc = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

memset(bccno, 0, sizeof(bccno));

for (int i = 0; i < n; i++)

if (!pre[i]) dfs(i, -1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2、边双连通：

对于一个连通图，如果任意两点至少存在两条边不重复的路径，则称这个图是边-双连通的。

除了桥不属于任何边-双连通分量之外，其他每条边恰好属于一个边-双连通分量。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Edge-biconnected.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5; // The number of Node

const int maxm = 1e6 + 5; // The number of Edge

/\* 如果要求边双联通分量，只需在做一遍dfs，保证不经过割边即可。\*/

int dfsclock, pre[maxn], cntbcc, bccno[maxn];

bool iscut[maxm \* 2];

int first[maxn], jump[maxm \* 2], linker[maxm \* 2];

int dfs (int u, int fa) {

int lowu = pre[u] = ++dfsclock;

for (int i = first[u]; i != -1; i = jump[i]) {

int v = linker[i];

if (!pre[v]) {

int lowv = dfs(v, u);

lowu = min(lowu, lowv);

if (lowv > pre[u])

iscut[i] = iscut[i^1] = 1;

} else if (pre[v] < pre[u] && v != fa)

lowu = min(lowu, pre[v]);

}

return lowu;

}

void dfs (int u) {

pre[u] = ++dfsclock;

bccno[u] = cntbcc;

for (int i = first[u]; i + 1; i = jump[i]) {

if (iscut[i]) continue;

int v = linker[i];

if (!pre[v]) dfs(v);

}

}

void findEdge (int n) {

dfsclock = 0;

memset(pre, 0, sizeof(pre));

memset(iscut, 0, sizeof(iscut));

for (int i = 1; i <= n; i++)

if (!pre[i]) dfs(i, -1);

dfsclock = cntbcc = 0;

memset(bccno, 0, sizeof(bccno));

for (int i = 1; i <= n; i++)

if (!pre[i]) { cntbcc++; dfs(i); }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（三）有向图强连通分量

在有向图中，任意两点之间存在相互可达的路径，则称这个图为强连通的。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Strong-onnected.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

stack<int> S;

vector<int> G[maxn];

int dfsclock, cntscc, sccno[maxn], pre[maxn];

int dfs(int u) {

int lowu = pre[u] = ++dfsclock;

S.push(u);

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if (!pre[v]) {

int lowv = dfs(v);

lowu = min(lowu, lowv);

} else if (!sccno[v])

lowu = min(lowu, pre[v]);

}

if (lowu == pre[u]) {

cntscc++;

while (true) {

int x = S.top();

S.pop();

sccno[x] = cntscc;

if (x == u) break;

}

}

return lowu;

}

void findSCC(int n) {

dfsclock = cntscc = 0;

memset(pre, 0, sizeof(pre));

memset(sccno, 0, sizeof(sccno));

for (int i = 0; i < n; i++)

if (!pre[i]) dfs(i);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（四）2-SAT

有n个布尔变量xi，另有m个需要满足的条件，每个条件的形式都是“xi为真/假或者xj为真/假”

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TwoSAT.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

struct TwoSAT {

int n, s[maxn \* 2], c;

bool mark[maxn \* 2];

vector<int> g[maxn \* 2];

void init (int n) {

this->n = n;

memset(mark, false, sizeof(mark));

for (int i = 0; i < n\*2; i++) g[i].clear();

}

void addClause(int x, int xval, int y, int yval) { // (sx || sy)

x = x \* 2 + xval;

y = y \* 2 + yval;

g[x^1].push\_back(y);

g[y^1].push\_back(x);

}

bool dfs (int x) {

if (mark[x^1]) return false;

if (mark[x]) return true;

mark[x] = true;

s[c++] = x;

for (int i = 0; i < g[x].size(); i++)

if (!dfs(g[x][i])) return false;

return true;

}

bool solve () {

for (int i = 0; i < n\*2; i += 2) {

if (!mark[i] && !mark[i+1]) {

c = 0;

if (!dfs(i)) {

while (c) mark[s[--c]] = false;

if (!dfs(i+1)) return false;

}

}

}

return true;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

例题：uva 12273（2sat变形+限制条件）

题目大意：给定一个DNA序列，然后给定一些回文位置的集合，判断是否可以通过-1，0，+1的变换，使得序列满足。

解题思路： 每个位置拆成4个状态考虑，建立关系，并且将每个位置不可转变的形态预先标记。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva12273.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5;

struct TwoSAT {

int n, s[maxn \* 4], c;

bool mark[maxn \* 4], must[maxn \* 4];

vector<int> g[maxn \* 4];

void init (int n) {

this->n = n;

memset(mark, 0, sizeof(mark));

memset(must, 0, sizeof(must));

for (int i = 0; i < 4 \* n; i++) g[i].clear();

}

void addLink(int x, int y) { g[x].push\_back(y); }

bool dfs (int u) {

for (int i = 1; i <= 3; i++)

if (mark[u^i]) return false;

if (must[u]) return false;

if (mark[u]) return true;

mark[u] = true;

s[c++] = u;

for (int i = 0; i < g[u].size(); i++)

if (!dfs(g[u][i])) return false;

return true;

}

void draw(int u) {

if (must[u]) return;

must[u] = true;

for (int i = 0; i < g[u].size(); i++)

draw(g[u][i]);

}

bool solve () {

for (int i = 0; i < 4 \* n; i += 4) {

if (!mark[i] && !mark[i+1] && !mark[i+2] && !mark[i+3]) {

bool flag = true;

c = 0;

for (int k = 0; k < 4 && flag; k++) {

if (must[i+k]) continue;

while (c) mark[s[--c]] = false;

if (dfs(i+k)) flag = false;

}

if (flag) return false;

}

}

return true;

}

}solver;

bool flag;

int N, M, a[maxn];

char S[maxn];

inline int idx(char c) {

if (c == 'A') return 0;

else if (c == 'G') return 1;

else if (c == 'T') return 2;

else return 3;

}

void addClause(int p, int q) {

for (int i = 0; i < 4; i++) {

solver.addLink(p \* 4 + i, q \* 4 + i);

solver.addLink(q \* 4 + i, p \* 4 + i);

}

}

void init () {

solver.init(N);

scanf("%s", S);

int k, x;

while (M--) {

scanf("%d%\*c", &k);

for (int i = 0; i < k; i++) scanf("%d", &a[i]);

for (int i = 0; i < k/2; i++) addClause(a[i], a[k-i-1]);

}

for (int i = 0; i < N; i++) { // 每个位置不能连续变换两次

int v = (idx(S[i]) + 2) % 4;

solver.draw(i\*4 + v);

}

for (int i = 1; i < N; i++) { // 相邻位置只能变化一个

int u = idx(S[i-1]), v = idx(S[i]);

for (int j = 1; j < 4; j++) {

int tu = (u + j) % 4, tv = (v + j) % 4;

solver.addLink((i-1)\*4+tu, i\*4+v);

solver.addLink(i\*4+tv, (i-1)\*4+u);

}

}

}

int main () {

while (scanf("%d%d", &N, &M) == 2 && N + M) {

init();

printf("%s\n", solver.solve() ? "YES" : "NO");

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

三、最短路

四、生成树

（一）最小生成树

最小生成树：给出加权无向图，求一棵生成树，使得树的权值和最小。

最小瓶颈生成树：给出加权无向图，求一棵生成树，使得最大边权值尽量小。

最小瓶颈路：给定加权无向图的两个节点u和v，求从u到v的一条路径，使得路径上的最长边尽量短。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MST.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 1e4 + 5; // The number of Node;

const int maxm = 1e6 + 5; // The number of Edge;

struct Edge {

int u, v, w;

Edge(int u = 0, int v = 0, int w = 0): u(u), v(v), w(w) {}

bool operator < (const Edge& a) const { return w < a.w; }

} E[maxm];

int F[maxn];

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

int Kruskal(int n, int m, Edge\* e) {

int ret = 0;

sort(e, e + m);

for (int i = 0; i <= n; i++) F[i] = i;

for (int i = 0; i < m; i++) {

int u = e[i].u, v = e[i].v, w = e[i].w;

if (find(u) != find(v)) {

F[find(u)] = find(v);

n--;

ret += w;

}

}

if (n != 1) {} // Can not build a MST

return ret;

}

/\* 最小瓶颈树：从一个空图开始，按照权值从小到大加入，图第一次完全联通时，该图的最小生成树即为原图的最小瓶颈生成树\*/

/\* 最小瓶颈路：求出图的最小生成树，则起点和终点间的唯一路径上权值最大的边即为要求的瓶颈\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（二）次小生成树

次小生成树：先对给定无向图做最小生成树，然后枚举一条非最小生成树的树边，在加入这条边后，图上会有回路，因此删除回路上权值最大的边。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SMST.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef pair<int,int> pii;

const int maxn = 1e4 + 5; // The number of Node;

const int maxm = 1e6 + 5; // The number of Edge;

const int inf = 0x3f3f3f3f;

struct Edge {

int u, v, w;

Edge(int u = 0, int v = 0, int w = 0): u(u), v(v), w(w) {}

bool operator < (const Edge& a) const { return w < a.w; }

}E[maxm];

int N, M, F[maxn], D[maxn][maxn], use[maxm];

vector<pii> G[maxn];

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

void dfs (int u, int fa, int w, int\* d) {

d[u] = w;

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i].first;

if (v == fa) continue;

dfs(v, u, max(w, G[u][i].second), d);

}

}

int Kruskal(int n, int m, Edge\* e) {

int ret = 0;

sort(e, e + m);

for (int i = 0; i <= n; i++) F[i] = i;

for (int i = 0; i < m; i++) {

int u = e[i].u, v = e[i].v, w = e[i].w;

if (find(u) != find(v)) {

n--;

use[i] = 1;

ret += w;

F[find(u)] = find(v);

G[u].push\_back(make\_pair(v, w));

G[v].push\_back(make\_pair(u, w));

}

}

if (n != 1) {} // Can not build a MST

return ret;

}

int SMST (int n, int m, Edge\* e) {

for (int i = 0; i <= n; i++) {

F[i] = i; G[i].clear();

}

memset(use, 0, sizeof(use));

int ans = Kruskal(n, m, e), ret = inf;

for (int i = 1; i <= N; i++) dfs(i, 0, 0, D[i]);

for (int i = 0; i < M; i++) {

if (use[i]) continue;

int u = e[i].u, v = e[i].v, w = e[i].w;

ret = min(ret, ans + w - D[u][v]);

}

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（三）最小有向生成树

有向生成树（directed spanning tree）：恰好有一个入度为0的节点（根节点），其它点入度均为1，可以从根节点到达所有其它节点。

最小有向生成树：给定一个有向带权图G和其中一个节点u，找出一个以u为根节点，权值和最小的有向生成树。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DMST.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

typedef long long ll;

const int inf = 0x3f3f3f3f;

const int maxn = 1005;

struct DMST {

struct Edge {

int u, v, d;

Edge(int u = 0, int v = 0, int d = 0): u(u), v(v), d(d) {}

bool operator < (const Edge& u) const { return d < u.d; }

};

int in[maxn], pre[maxn], id[maxn], vis[maxn];

vector<Edge> edges;

void init () { edges.clear(); }

void addEdge(int u, int v, int d) { edges.push\_back(Edge(u, v, d)); }

bool directedMST(int root, int n, int& ans) {

ans = 0;

while (true) {

// 删除自环+找最小入边

memset(in, inf, sizeof(in));

for (int i = 0; i < edges.size(); i++) {

int u = edges[i].u, v = edges[i].v;

if (edges[i].d < in[v] && u != v) {

pre[v] = u;

in[v] = edges[i].d;

}

}

// 判断是否可以到达所有节点

for (int i = 0; i < n; i++) {

if (i == root) continue;

if (in[i] == inf) return false;

}

int cntnode = 0;

//memset(id, -1, sizeof(int) \* n);

//memset(vis, -1, sizeof(int) \* n);

memset(id, -1, sizeof(id));

memset(vis, -1, sizeof(vis));

in[root] = 0;

for (int i = 0; i < n; i++) {

ans += in[i];

int v = i;

while (vis[v] != i && id[v] == -1 && v != root) {

vis[v] = i;

v = pre[v];

}

if (v != root && id[v] == -1) {

for (int u = pre[v]; u != v; u = pre[u])

id[u] = cntnode;

id[v] = cntnode++;

}

}

// 没有找到环, 终止

if (cntnode == 0) break;

// 缩点，重新标记

for (int i = 0; i < n; i++) {

if (id[i] == -1)

id[i] = cntnode++;

}

for (int i = 0; i < edges.size(); i++) {

int v = edges[i].v;

edges[i].u = id[edges[i].u];

edges[i].v = id[edges[i].v];

if (edges[i].u != edges[i].v)

edges[i].d -= in[v];

}

n = cntnode;

root = id[root];

}

return true;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

五、二分图

（一）二分图判定

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bipartite.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e6 + 5;

struct Bipartite {

int n, color[maxn];

vector<int> g[maxn];

void init (int n) {

this->n = n;

memset(color, -1, sizeof(color));

for (int i = 0; i <= n; i++) g[i].clear();

}

void addEdge(int u, int v) {

g[u].push\_back(v);

g[v].push\_back(u);

}

bool dfs(int u) {

for (int i = 0; i < g[u].size(); i++) {

int v = g[u][i];

if (color[u] == color[v]) return false;

if (!color[v]) {

color[v] = 3 - color[u];

if (!dfs(v)) return false;

}

}

return true;

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（二）二分图匹配

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KM.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

const int maxn = 1e5 + 5;

int N, L[maxn];

bool T[maxn];

vector<int> G[maxn];

bool match(int u) {

for (int i = 0; i < G[u].size(); i++) {

int v = G[u][i];

if (!T[v]) {

T[v] = true;

if (!L[v] || match(L[v])) {

L[v] = u;

return true;

}

}

}

return false;

}

int KM () {

int ret = 0;

memset(L, 0, sizeof(L));

for (int i = 1; i <= N; i++) {

memset(T, false, sizeof(T));

if (match(i)) ret++;

}

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（三）最佳完美匹配

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PKM.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const int maxn = 105;

const int inf = 0x3f3f3f3f;

int N, L[maxn], Lx[maxn], Ly[maxn], W[maxn][maxn], slack[maxn];

bool S[maxn], T[maxn];

int match (int u) {

S[u] = true;

for (int i = 1; i <= N; i++) if (!T[i]) {

if (Lx[u] + Ly[i] == W[u][i]) {

T[i] = true;

if (!L[i] || match(L[i])) {

L[i] = u;

return true;

}

} else

slack[i] = min(slack[i], Lx[u]+Ly[i]-W[u][i]);

}

return false;

}

void update () {

int a = inf;

for (int i = 1; i <= N; i++) if (!T[i])

a = min(a, slack[i]);

for (int i = 1; i <= N; i++) {

if (S[i]) Lx[i] -= a;

if (T[i]) Ly[i] += a;

}

}

void KM () {

for (int i = 1; i <= N; i++) {

L[i] = Lx[i] = Ly[i] = 0;

for (int j = 1; j <= N; j++)

Lx[i] = max(Lx[i], W[i][j]);

}

for (int i = 1; i <= N; i++) {

for (int j = 1; j <= N; j++) slack[j] = inf;

while (true) {

for (int j = 1; j <= N; j++) S[j] = T[j] = false;

if (match(i)) break;

else update();

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PKM.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（四）稳定婚姻问题

例题：uva 1175（稳定婚姻问题）

题目大意：n个男生，n个女生，每个人都对异性有一个排序，代表对他们的喜欢程度。你的任务是将男生和女生一一配对，使得男生u和女生v不存在以下情况：（1）男生u和女生v不是一对；（2）他们喜欢对方的程度都大于各自当前的舞伴。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*uva1175.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

using namespace std;

const int maxn = 1005;

int N, pref[maxn][maxn], order[maxn][maxn], jump[maxn];

int future\_husband[maxn], future\_wife[maxn];

queue<int> Q;

void engage (int man, int woman) {

int m = future\_husband[woman];

if (m) {

future\_wife[m] = 0;

Q.push(m);

}

future\_wife[man] = woman;

future\_husband[woman] = man;

}

int main () {

int cas;

scanf("%d", &cas);

while (cas--) {

scanf("%d", &N);

for (int i = 1; i <= N; i++) {

for (int j = 1; j <= N; j++) scanf("%d", &pref[i][j]);

jump[i] = 1;

future\_wife[i] = 0;

Q.push(i);

}

for (int i = 1; i <= N; i++) {

for (int j = 1; j <= N; j++) {

int x;

scanf("%d", &x);

order[i][x] = j;

}

future\_husband[i] = 0;

}

while (!Q.empty()) {

int man = Q.front();

Q.pop();

int woman = pref[man][jump[man]++];

if (!future\_husband[woman]) engage(man, woman);

else if (order[woman][man] < order[woman][future\_husband[woman]])

engage(man, woman);

else

Q.push(man);

}

for (int i = 1; i <= N; i++)

printf("%d\n", future\_wife[i]);

if (cas) printf("\n");

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/