## 最小树形图\_朱刘算法(poj 3164)

#include <climits>

#include <cstring>

#include <cstdio>

#include <cmath>

#define N 1005

#define type double

#define MAX INT\_MAX

#define MAXN 105

#define MAXM 1005

using namespace std;

struct Point

{

double x, y;

}p[MAXN];

struct Edge{

int u , v;

type cost;

}E[40005];

int pre[N],ID[N],vis[N];

type In[N];

int n,m;

double dis(Point a, Point b)

{

return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y));

}

type zhuliu(int root,int NV,int NE) {//根、结点数、边数

type ret = 0;

while(true) {

//1.找最小入边

for(int i=0;i<NV;i++) In[i] = MAX;

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

int u = E[i].u;

int v = E[i].v;

if(E[i].cost < In[v] && u != v) { //这一步可以把自环切掉，找出每个点的最小入边

pre[v] = u;

In[v] = E[i].cost;

}

}

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

if(i == root) continue;

if(In[i] == MAX) return -1;//除了跟以外有点没有入边,则根无法到达它

}

//2.找环

int cntnode = 0;

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

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

In[root] = 0;

for(int i=0;i<NV;i++) {//标记每个环

ret += In[i];

int v = i;

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

vis[v] = i; //vis的作用就是把环中各节点都标记为有入边的那个点，如下图，把1,2,3,4,5,6都标记为1

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 ++;

}

}

if(cntnode == 0) break;//无环

for(int i=0;i<NV;i++) if(ID[i] == -1) {

ID[i] = cntnode ++;

}

//3.缩点,重新标记，调整进入环的边的权值，以便下一轮循环找出最小入边

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

int v = E[i].v;

E[i].u = ID[E[i].u];

E[i].v = ID[E[i].v];

if(E[i].u != E[i].v) {

E[i].cost -= In[v];

}

}

NV = cntnode;

root = ID[root];

}

return ret;

}

int main()

{

while(scanf("%d%d", &n, &m) != EOF)

{

for(int i = 0; i < n; i++) scanf("%lf%lf", &p[i].x, &p[i].y);

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

{

scanf("%d%d", &E[i].u, &E[i].v);

E[i].u--;

E[i].v--;

if(E[i].u != E[i].v) E[i].cost = dis(p[E[i].u], p[E[i].v]);

else E[i].cost = MAX; //去除自环

}

type ans = zhuliu(0, n, m);

if(ans == -1) printf("poor snoopy\n");

else printf("%.2f\n", ans);

}

return 0;

}

## 斯坦纳树

题目:

斯坦纳树   
Time Limit: 1 Sec Memory Limit: 128 MB   
Description   
现在有一个n\*m的矩阵,某些元素为0,剩下的元素大于0.   
现在你要选择一些元素,使得任意两个为0的元素都能够通过选中的元素四连通.   
(注意,若想达到要求,所有的0自身必须被选中.)   
那么请问选中元素的和的最小值是多少?

Input   
第一行两个整数n,m,表示矩阵的长和宽.   
接下来n行,每行m个整数,第i行第j个整数A[i,j]表示矩阵中第i行第j列的元素大小.

Output   
输出一行一个整数表示选中元素的最小和.

Sample Input   
4 4   
0 1 1 0   
1 9 9 1   
1 9 9 1   
0 1 1 0   
Sample Output   
6

HINT   
1<=n,m<=10,2<=num(0)<=10,0<=A[i,j]<=2^16.

题解：

斯坦纳树模板   
流程：

枚举状态集S   
{   
     枚举S的子集s   
     {   
         更新f[S][1~n]   
     }   
     将 f[S][x]<inf 的x入队   
     spfa(S)   
}

代码：

#include <queue>

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#define N 105

#define P 12

#define inf 0x3f3f3f3f

using namespace std;

const int dx[]={0,0,1,-1};

const int dy[]={1,-1,0,0};

struct Eli

{

int v,n;

}e[N<<2];

int head[N],cnt;

inline void add(int u,int v)

{

e[++cnt].v=v;

e[cnt].n=head[u];

head[u]=cnt;

}

int n,m;

int id[P][P],val[N];

void build()

{

int i,j,k;

int x,y;

scanf("%d%d",&n,&m);

for(i=1;i<=n;i++)for(j=1;j<=m;j++)

scanf("%d",&val[id[i][j]=++cnt]);

cnt=0;

for(i=1;i<=n;i++)for(j=1;j<=m;j++)for(k=0;k<4;k++)

if(id[x=i+dx[k]][y=j+dy[k]])add(id[i][j],id[x][y]);

n\*=m,cnt=0;

}

int f[1<<P][N];

queue<int>q;

bool in[N];

void spfa(int S)

{

int i,u,v;

while(!q.empty())

{

u=q.front(),q.pop(),in[u]=0;

for(i=head[u];i;i=e[i].n)

{

if(f[S][v=e[i].v]>f[S][u]+val[v])

{

f[S][v]=f[S][u]+val[v];

if(!in[v])q.push(v);

}

}

}

}

void work()

{

int i,j,k;

int S,s;

memset(f,0x3f3f3f3f,sizeof f);

for(i=1;i<=n;i++)if(!val[i])f[1<<cnt++][i]=0;

for(S=1;S<(1<<cnt);S++)

{

for(s=(S-1)&S;s;s=(s-1)&S)for(i=1;i<=n;i++)

f[S][i]=min(f[S][i],f[s][i]+f[S^s][i]-val[i]);

for(i=1;i<=n;i++)if(f[S][i]<inf&&!in[i])q.push(i),in[i]=1;

spfa(S);

}

int ans=inf;

for(i=1;i<=n;i++)ans=min(ans,f[(1<<cnt)-1][i]);

printf("%d\n",ans);

}

int main()

{

freopen("test.in","r",stdin);

build();

work();

return 0;

}

## 最小乘积生成树(bzoj 2359)

题解：

裸最小乘积生成树。

最小乘积生成树定义：

有一张n个点m条边的无向图，每条边有k个权值。

现在要取一个边集M使得其将所有点连通，并使

∏ki=1(∑j∈Mjcost(j,vali)) 最小

即个边集的每一种边权的总和的乘积最小。

比如：

k=1时，就是裸最小生成树。

k=2时，就是要使 [边集的权值1的和]\*[边集的权值2的和] 最小。

最小乘积生成树的一种求法：

广义上的说法（没必要看，或者看完下面的再来看这个就好）

首先我们可以把每种生成树想成一个k维的点，第i维的坐标即那一维上权值的和。

然后我们可以先求出每一维坐标最小的一棵生成树（裸上最小生成树就好），

然后得到一个k-1维的面，然后我们来求一下离这个面最远的点，然后分治下去……据说期望很快……

二维最小乘积生成树的求法：

给每一棵生成树都定义两个权值X、Y，其中X为其包含的所有边的权值x的和，Y为其包含的所有边的权值y的和，那么我们可以把每一种生成树看成一个坐标。

我们先求出坐标x最小的一棵生成树，再求出坐标y最小的一棵生成树。

然后我们可以考虑，最优的点一定在下凸包上【证明一】，然后我们要进行一个不断向左下拓展点的过程：对于两个点A、B形成的直线，我们可以找出在这条直线左下的最远的点C，然后对AC、CB递归做同样的过程，直到找不到一个在左下的点C为止。

然后如何找一个最远的点C呢？

我们可以发现既然有一条边固定，那么不妨把“最远”转化成三角形面积最大，这个用叉积搞一搞，，然后会推出公式面积跟点C有关的部分= 常数A∗x+常数B∗y ，，那么我们把所有边的权值 x 乘上 A ，权值 y 乘上 B 就好了，，，A、B 是啥自己求去！

【题外】：三维的，就是离一个平面最远，转化成体积最大，递归分成三层而非两层，，，，，然后四维，甚至更高维，感觉应该是同理的吧？

关于上文中【证明一】

每个点（xi,yi） 都对应一条函数曲线 ki=xi∗yi，而任意两不同 ki ，它们的函数曲线是不交的（有交的话则存在一点 (xj,yj) 使得 ki=xj∗yj=kj而ki!=kj 成立，显然这是悖论），那么显然最优点肯定不会在凸包内，否则必有凸包上一点比它优。

那么会不会求出这个某种意义上的凸包后，最优点在凸包外，却没被找到呢？

不会。

若有这种情况，此点必然在凸包上某边的左下方，，然后一定会被找出来。。

代码：

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#define N 205

#define M 10100

#define inf 0x3f3f3f3f

using namespace std;

struct Eli

{

int u,v,a,b,c;

void read()

{

scanf("%d%d%d%d",&u,&v,&a,&b);

u++,v++;

}

}e[M];

inline bool cmpa(const Eli &a,const Eli &b){return a.a<b.a;}

inline bool cmpb(const Eli &a,const Eli &b){return a.b<b.b;}

inline bool cmpc(const Eli &a,const Eli &b){return a.c<b.c;}

struct Point

{

int x,y;

void print(){printf("%d %d\n",x,y);}

Point(int \_x=0,int \_y=0):x(\_x),y(\_y){}

bool operator < (const Point &A)const

{

unsigned int p= x;p\*= y;

unsigned int q=A.x;q\*=A.y;

return p==q?x<A.x:p<q;

}

}ans,now,mina,minb;

int f[N],n,m;

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

Point Kruscal()

{

int i,fa,fb;

now=Point(0,0);

for(i=1;i<=n;i++)f[i]=i;

for(i=1;i<=m;i++)

{

fa=find(e[i].u),fb=find(e[i].v);

if(fa!=fb)

{

f[fb]=fa;

now.x+=e[i].a;

now.y+=e[i].b;

}

}

if(now<ans)ans=now;

return now;

}

inline int xmul(const Point &A,const Point &B,const Point &C)

{return (C.y-A.y)\*(B.x-A.x)-(C.x-A.x)\*(B.y-A.y);}

void work(const Point &A,const Point &B)

{

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

e[i].c=e[i].b\*(A.x-B.x)+e[i].a\*(B.y-A.y);

sort(e+1,e+m+1,cmpc);

Point C=Kruscal();

if(xmul(A,B,C)<=0)return ;

work(A,C),work(C,B);

}

int main()

{

// freopen("test.in","r",stdin);

int i,j,k;

int a,b,c;

ans=Point(inf,inf);

scanf("%d%d",&n,&m);

for(i=1;i<=m;i++)e[i].read();

sort(e+1,e+m+1,cmpa),mina=Kruscal();

sort(e+1,e+m+1,cmpb),minb=Kruscal();

work(minb,mina),ans.print();

fclose(stdin);

fclose(stdout);

return 0;

}

## fleury算法

#include <iostream>

#include <stdio.h>

#include <string.h>

#define MAXN 200

using namespace std;

struct stack{

int top,node[MAXN];

}s;

int Edge[MAXN][MAXN];

int n;

void dfs(int x){

int i;

s.top++;

s.node[s.top]=x;

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

if(Edge[i][x]>0){

Edge[i][x]=0;

Edge[x][i]=0;

dfs(i);

break;

}

}

}

void Fleury(int x){

int i,b;

s.top=0;s.node[s.top]=x;

while(s.top>=0){

b=0;

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

if(Edge[s.node[s.top]][i]>0){

b=1;break;

}

}

if(b==0){

printf("%d ",s.node[s.top]+1);

s.top--;

}

else{

s.top--;

dfs(s.node[s.top+1]);

}

}

printf("\n");

}

int main()

{

//freopen("in.txt","r",stdin);

int i,j;

int m,s,t;

int degree,num,start;

while(scanf("%d%d",&n,&m)!=EOF){

memset(Edge,0,sizeof(Edge));

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

scanf("%d%d",&s,&t);

Edge[s-1][t-1]=1;

Edge[t-1][s-1]=1;

}

num=0;start=0;

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

degree=0;

for(j=0;j<n;j++)

degree+=Edge[i][j];

if(degree%2==1){

start=i;

num++;

}

}

if(num==0 || num==2) Fleury(start);

else printf("No Euler path\n");

}

return 0;

}

## ISAP最大流

#include <stdio.h>

#include <string.h>

#include <algorithm>

#define clear(A, X) memset (A, X, sizeof A)

#define copy(A, B) memcpy (A, B, sizeof A)

using namespace std;

const int maxE = 1000000;

const int maxN = 100000;

const int maxQ = 1000000;

const int oo = 0x3f3f3f3f;

struct Edge {

int v;//弧尾

int c;//容量

int n;//指向下一条从同一个弧头出发的弧

} edge[maxE];//边组

int adj[maxN], cntE;//前向星的表头

int Q[maxQ], head, tail;//队列

int d[maxN], cur[maxN], pre[maxN], num[maxN];

int sourse, sink, nv;//sourse：源点，sink：汇点，nv：编号修改的上限

int n, m;

void addedge (int u, int v, int c) {//添加边

//正向边

edge[cntE].v = v;

edge[cntE].c = c;//正向弧的容量为c

edge[cntE].n = adj[u];

adj[u] = cntE++;

//反向边

edge[cntE].v = u;

edge[cntE].c = 0;//反向弧的容量为0

edge[cntE].n = adj[v];

adj[v] = cntE++;

}

void rev\_bfs () {//反向BFS标号

clear (num, 0);

clear (d, -1);//没标过号则为-1

d[sink] = 0;//汇点默认为标过号

num[0] = 1;

head = tail = 0;

Q[tail++] = sink;

while (head != tail) {

int u = Q[head++];

for (int i = adj[u]; ~i; i = edge[i].n) {

int v = edge[i].v;

if (~d[v]) continue;//已经标过号

d[v] = d[u] + 1;//标号

Q[tail++] = v;

num[d[v]]++;

}

}

}

int ISAP() {

copy (cur, adj);//复制，当前弧优化

rev\_bfs ();//只用标号一次就够了，重标号在ISAP主函数中进行就行了

int flow = 0, u = pre[sourse] = sourse, i;

while (d[sink] < nv) {//最长也就是一条链，其中最大的标号只会是nv - 1，如果大于等于nv了说明中间已经断层了。

if (u == sink) {//如果已经找到了一条增广路，则沿着增广路修改流量

int f = oo, neck;

for (i = sourse; i != sink; i = edge[cur[i]].v) {

if (f > edge[cur[i]].c){

f = edge[cur[i]].c;//不断更新需要减少的流量

neck = i;//记录回退点，目的是为了不用再回到起点重新找

}

}

for (i = sourse; i != sink; i = edge[cur[i]].v) {//修改流量

edge[cur[i]].c -= f;

edge[cur[i] ^ 1].c += f;

}

flow += f;//更新

u = neck;//回退

}

for (i = cur[u]; ~i; i = edge[i].n) if (d[edge[i].v] + 1 == d[u] && edge[i].c) break;

if (~i) {//如果存在可行增广路，更新

cur[u] = i;//修改当前弧

pre[edge[i].v] = u;

u = edge[i].v;

}

else {//否则回退，重新找增广路

if (0 == (--num[d[u]])) break;//GAP间隙优化，如果出现断层，可以知道一定不会再有增广路了

int mind = nv;

for (i = adj[u]; ~i; i = edge[i].n) {

if (edge[i].c && mind > d[edge[i].v]) {//寻找可以增广的最小标号

cur[u] = i;//修改当前弧

mind = d[edge[i].v];

}

}

d[u] = mind + 1;

num[d[u]]++;

u = pre[u];//回退

}

}

return flow;

}

void init () {//初始化

clear (adj, -1);

cntE = 0;

}

void work () {

int u, v, c;

init ();

for (int i = 0; i < m; ++ i) scanf ("%d%d%d", &u, &v, &c), addedge (u, v, c);

sourse = 1; sink = n; nv = sink + 1;

printf ("%d\n", ISAP ());

}

int main() {

while (~scanf("%d%d", &m, &n)) work ();

return 0;

}

## Dinic费用流(poj2112)

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <algorithm>

#include <cstring>

using namespace std;

const int MAXPT = 255;

const int MAXED = 100005;

const int INF = 0x3fffffff;

struct node{

int u, v, flow;

int opp;

int next;

};

struct Dinic{

node arc[MAXED];

int vn, en, head[MAXPT]; //vn点个数(包括源点汇点),en边个数

int cur[MAXPT]; //当前弧

int q[MAXPT]; //bfs建层次图时的队列

int path[MAXED], top; //存dfs当前最短路径的栈

int dep[MAXPT]; //各节点层次

void init(int n){

vn = n;

en = 0;

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

}

void insert\_flow(int u, int v, int flow){

arc[en].u = u;

arc[en].v = v;

arc[en].flow = flow;

arc[en].opp = en + 1;

arc[en].next = head[u];

head[u] = en ++;

arc[en].u = v;

arc[en].v = u;

arc[en].flow = 0; //反向弧

arc[en].opp = en - 1;

arc[en].next = head[v];

head[v] = en ++;

}

bool bfs(int s, int t){

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

int lq = 0, rq = 1;

dep[s] = 0;

q[lq] = s;

while(lq < rq){

int u = q[lq ++];

if (u == t){

return true;

}

for (int i = head[u]; i != -1; i = arc[i].next){

int v = arc[i].v;

if (dep[v] == -1 && arc[i].flow > 0){

dep[v] = dep[u] + 1;

q[rq ++] = v;

}

}

}

return false;

}

int solve(int s, int t){

int maxflow = 0;

while(bfs(s, t)){

int i,j,k;

for (i = 1; i <= vn; i ++) cur[i] = head[i];

for (i = s, top = 0;;){

if (i == t){

int mink;

int minflow = 0x3fffffff;

for (k = 0; k < top; k ++)

if (minflow > arc[path[k]].flow){

minflow = arc[path[k]].flow;

mink = k;

}

for (k = 0; k < top; k ++)

arc[path[k]].flow -= minflow, arc[arc[path[k]].opp].flow += minflow;

maxflow += minflow;

top = mink; //arc[mink]这条边流量变为0, 则直接回溯到该边的起点即可(这条边将不再包含在增广路内).

i = arc[path[top]].u;

}

for (j = cur[i]; j != -1; cur[i] = j = arc[j].next){

int v = arc[j].v;

if (arc[j].flow && dep[v] == dep[i] + 1)

break;

}

if (j != -1){

path[top ++] = j;

i = arc[j].v;

}

else{

if (top == 0) break;

dep[i] = -1;

i = arc[path[-- top]].u;

}

}

}

return maxflow;

}

}dinic;

int map[250][250];

void floyd(int n){

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

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

if(map[i][k] == INF) continue;

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

if (map[j][k] == INF) continue;

if (map[i][j] > map[i][k] + map[k][j])

map[i][j] = map[i][k] + map[k][j];

}

}

}

return ;

}

int go(int mid, int k, int c, int m)

{

int i;

dinic.init(k+c+2);

for (i = 1; i <= k; i ++){

dinic.insert\_flow(k+c+1, i, m);

}

for (i = k+1; i <= k+c; i ++){

dinic.insert\_flow(i, k+c+2, 1);

}

for (i = 0; i < k; i ++){

for (int j = k; j < k+c; j ++){

if (map[i][j] <= mid){

dinic.insert\_flow(i+1, j+1, 1);

}

}

}

return dinic.solve(k+c+1, k+c+2);

}

int BS(int k, int c, int m){

int l = 0, r = 10000000;

while(l < r){

int mid = (l+r)>>1;

if (go(mid, k, c, m) == c){

r = mid;

}

else{

l = mid + 1;

}

}

return r;

}

int main ()

{

int k, c, m;

scanf("%d %d %d", &k, &c, &m);

for (int i = 0; i < k+c; i ++){

for (int j = 0; j < k+c; j ++){

scanf("%d", &map[i][j]);

if (map[i][j] == 0)

map[i][j] = INF;

}

}

floyd(k+c);

printf("%d\n", BS(k, c, m));

return 0;

}

## KM算法

#include <iostream>

#include <cstring>

#include <cstdio>

using namespace std;

const int MAXN = 305;

const int INF = 0x3f3f3f3f;

int love[MAXN][MAXN]; // 记录每个妹子和每个男生的好感度

int ex\_girl[MAXN]; // 每个妹子的期望值

int ex\_boy[MAXN]; // 每个男生的期望值

bool vis\_girl[MAXN]; // 记录每一轮匹配匹配过的女生

bool vis\_boy[MAXN]; // 记录每一轮匹配匹配过的男生

int match[MAXN]; // 记录每个男生匹配到的妹子 如果没有则为-1

int slack[MAXN]; // 记录每个汉子如果能被妹子倾心最少还需要多少期望值

int N;

bool dfs(int girl)

{

vis\_girl[girl] = true;

for (int boy = 0; boy < N; ++boy) {

if (vis\_boy[boy]) continue; // 每一轮匹配 每个男生只尝试一次

int gap = ex\_girl[girl] + ex\_boy[boy] - love[girl][boy];

if (gap == 0) { // 如果符合要求

vis\_boy[boy] = true;

if (match[boy] == -1 || dfs( match[boy] )) { // 找到一个没有匹配的男生 或者该男生的妹子可以找到其他人

match[boy] = girl;

return true;

}

} else {

slack[boy] = min(slack[boy], gap); // slack 可以理解为该男生要得到女生的倾心 还需多少期望值 取最小值 备胎的样子【捂脸

}

}

return false;

}

int KM()

{

memset(match, -1, sizeof match); // 初始每个男生都没有匹配的女生

memset(ex\_boy, 0, sizeof ex\_boy); // 初始每个男生的期望值为0

// 每个女生的初始期望值是与她相连的男生最大的好感度

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

ex\_girl[i] = love[i][0];

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

ex\_girl[i] = max(ex\_girl[i], love[i][j]);

}

}

// 尝试为每一个女生解决归宿问题

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

fill(slack, slack + N, INF); // 因为要取最小值 初始化为无穷大

while (1) {

// 为每个女生解决归宿问题的方法是 ：如果找不到就降低期望值，直到找到为止

// 记录每轮匹配中男生女生是否被尝试匹配过

memset(vis\_girl, false, sizeof vis\_girl);

memset(vis\_boy, false, sizeof vis\_boy);

if (dfs(i)) break; // 找到归宿 退出

// 如果不能找到 就降低期望值

// 最小可降低的期望值

int d = INF;

for (int j = 0; j < N; ++j)

if (!vis\_boy[j]) d = min(d, slack[j]);

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

// 所有访问过的女生降低期望值

if (vis\_girl[j]) ex\_girl[j] -= d;

// 所有访问过的男生增加期望值

if (vis\_boy[j]) ex\_boy[j] += d;

// 没有访问过的boy 因为girl们的期望值降低，距离得到女生倾心又进了一步！

else slack[j] -= d;

}

}

}

// 匹配完成 求出所有配对的好感度的和

int res = 0;

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

res += love[ match[i] ][i];

return res;

}

int main()

{

while (~scanf("%d", &N)) {

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

for (int j = 0; j < N; ++j)

scanf("%d", &love[i][j]);

printf("%d\n", KM());

}

return 0;

}

## Tarjan(求桥割点，点从1-n)

#include<iostream>

using namespace std;

#include<cstdio>

#include<cstring>

#include<vector>

#define N 201

vector<int>G[N];

int n,m,low[N],dfn[N];

bool is\_cut[N];

int father[N];

int tim=0;

void input()

{

scanf("%d%d",&n,&m);

int a,b;

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

{

scanf("%d%d",&a,&b);

G[a].push\_back(b);/\*邻接表储存无向边\*/

G[b].push\_back(a);

}

}

void Tarjan(int i,int Father)

{

father[i]=Father;/\*记录每一个点的父亲\*/

dfn[i]=low[i]=tim++;

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

{

int k=G[i][j];

if(dfn[k]==-1)

{

Tarjan(k,i);

low[i]=min(low[i],low[k]);

}

else if(Father!=k)/\*假如k是i的父亲的话，那么这就是无向边中的重边，有重边那么一定不是桥\*/

low[i]=min(low[i],low[k]);

}

}

void count()

{

int rootson=0;

Tarjan(1,0);

for(int i=2;i<=n;++i)

{

int v=father[i];

if(v==1)

rootson++;/\*统计根节点子树的个数，根节点的子树个数>=2,就是割点\*/

else{

if(low[i]>=dfn[v])/\*割点的条件\*/

is\_cut[v]=true;

}

}

if(rootson>1)

is\_cut[1]=true;

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

if(is\_cut[i])

printf("%d\n",i);

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

{

int v=father[i];

if(v>0&&low[i]>dfn[v])/\*桥的条件\*/

printf("%d,%d\n",v,i);

}

}

int main()

{

input();

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

memset(father,0,sizeof(father));

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

memset(is\_cut,false,sizeof(is\_cut));

count();

return 0;

}

## Tarjan(SCC)

#include <queue>

#include <stack>

#include <stdio.h>

#include <string.h>

#include <algorithm>

using namespace std;

const int N = 2e3+5;

const int M = 1e4+5;

struct Vertex

{

int head;

}V[N];

struct Edge

{

int v,next;

}E[M];

int top,scc,id,dfn[N],low[N],belong[N];

bool in[N];

stack<int> S;

void init()

{

top = 0;

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

}

void add\_edge(int u,int v)

{

E[top].v = v;

E[top].next = V[u].head;

V[u].head = top++;

}

void tarjan(int u)

{

dfn[u] = low[u] = ++id;

S.push(u);

in[u] = true;

for(int i=V[u].head;~i;i=E[i].next)

{

int v = E[i].v;

if(!dfn[v])

{

tarjan(v);

low[u] = min(low[u],low[v]);

}

else if(in[v])

low[u] = min(low[u],dfn[v]);

}

if(dfn[u] == low[u])

{

int v;

do

{

v = S.top();

S.pop();

in[v] = false;

belong[v] = scc;

}while(u != v);

scc++;

}

}

void get\_scc(int n)

{

scc = id = 0;

memset(dfn,0,sizeof(dfn));

memset(in,false,sizeof(in));

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

if(!dfn[i])

tarjan(i);

}

## Tarjan(BCC)

#include <cstdio>

#include <cstring>

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;

const int MAXNODE = 10005;

const int MAXEDGE = 100010;

const int INF = 0x3f3f3f3f;

struct Edge{

int v, id, next;

bool bridge;

Edge() {}

Edge(int v, int id, int next): v(v), id(id), next(next), bridge(false){}

}E[MAXEDGE \* 2], cut[MAXEDGE];

struct Node {

int u, v;

Node() {}

Node(int u, int v): u(u), v(v) {}

};

stack<Node> Stack;

//bcc表示的是一个bcc里面的点

vector<int> bcc[MAXNODE];

//pre纪录的是时间戳,lowlink纪录的是该点及其该子孙节点所能返回的最早时间戳是多少,bccno纪录的是该点当前是属于哪个bcc的

int head[MAXNODE], pre[MAXNODE], lowlink[MAXNODE], bccno[MAXNODE];

int n, m, tot, bcc\_cnt, dfs\_clock, cut\_cnt;

bool iscut[MAXNODE];

//双向边的添加

void AddEdge(int u, int v, int id) {

E[tot] = Edge(v, id, head[u]);

head[u] = tot++;

E[tot] = Edge(u, id, head[v]);

head[v] = tot++;

}

void init() {

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

tot = 0;

}

//点双连通

void dfs(int u, int fa) {

pre[u] = lowlink[u] = ++dfs\_clock;

int child = 0;//纪录当前节点有多少个子节点

for (int i = head[u]; ~i; i = E[i].next) {

int v = E[i].v;

if (!pre[v]) {

Stack.push(Node(u, v));

child++;

dfs(v, u);

lowlink[u] = min(lowlink[u], lowlink[v]);//更新

//子节点最多返回到该点

if (lowlink[v] >= pre[u]) {

//该边为桥

if (lowlink[v] > pre[u]) {

E[i].bridge = E[i ^ 1].bridge = true;

cut[cut\_cnt++] = E[i];

}

iscut[u] = true;

bcc\_cnt++; bcc[bcc\_cnt].clear();

while (1) {

Node x = Stack.top(); Stack.pop();

if (bccno[x.u] != bcc\_cnt) {

bcc[bcc\_cnt].push\_back(x.u);

bccno[x.u] = bcc\_cnt;

}

if (bccno[x.v] != bcc\_cnt) {

bcc[bcc\_cnt].push\_back(x.v);

bccno[x.v] = bcc\_cnt;

}

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

}

}

}

else if (v != fa && pre[v] < pre[u]) {//反向边

Stack.push(Node(u, v));

lowlink[u] = min(lowlink[u], pre[v]);

}

}

//u是根结点，且只有一个孩子，那就不是割点了

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

}

void find\_bcc() {

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

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

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

dfs\_clock = bcc\_cnt = cut\_cnt = 0;

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

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

}

int main() {

init();

return 0;

}

## 2-sat算法(hdu 3062)

#include<stdio.h>

#include<string.h>

const int maxn = 3000;

struct node

{

int v,next;

}eg[maxn\*maxn];

int head[maxn];

int dfn[maxn],low[maxn],sta[maxn],insta[maxn],belong[maxn];

int tot,color,top,Index,n;

void add(int a,int b)

{

eg[tot].v= b;

eg[tot].next = head[a];

head[a] = tot++;

}

void tarjan(int u)

{

dfn[u] = low[u] = ++Index;

sta[top++] = u;

insta[u] = 1;

for(int i=head[u];i+1;i= eg[i].next)

{

int v= eg[i].v;

if(!dfn[v])

{

tarjan(v);

low[u] = low[u] < low[v] ? low[u] : low[v];

}else if(insta[v])

low[u] = low[u] < dfn[v] ? low[u] : dfn[v];

}

if(low[u] == dfn[u])

{

int v;

color++;

do

{

v= sta[--top];

insta[v] = 0;

belong[v] =color;

}while(v != u);

}

}

void init()

{

tot = color = top = Index =0;

memset(dfn,0,sizeof(dfn));

memset(insta,0,sizeof(insta));

memset(belong,0,sizeof(belong));

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

}

void work()

{

int i;

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

{

if(!dfn[i])tarjan(i);

}

int flag = 0;

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

{

if(belong[i\*2] == belong[i\*2+1])

{

break;

}

}

if(i == n) printf("YES\n");

else printf("NO\n");

}

int main()

{

int m,a1,a2,b1,b2;

while(~scanf("%d",&n))

{

init();

scanf("%d",&m);

while(m--)

{

scanf("%d%d%d%d",&a1,&a2,&b1,&b2);

// printf("%d %d\n",a1\*2+a2,b1\*2 + !b2);

// printf("%d %d\n",b1\*2 + b2,a1\*2+!a2);

add(a1\*2 + b1,a2\*2 + 1 -b2);

add(a2\*2 + b2,a1\*2 + 1 -b1);

}

work();

}

return 0;

}

## 最大团(hdu 1530)

#include<iostream>

#include<cstring>

#include<cstdio>

using namespace std;

int n;

int map[55][55],cn,maxt,mark[500];

void dfs(int x)

{

if(x>n)

{

maxt=cn;

return;

}

bool flag=true;

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

{

if(mark[i]&&!map[x][i])

{

flag=false;

break;

}

}

if(flag)

{

cn++;

mark[x]=true;

dfs(x+1);

cn--;

}

if(cn+n-x+1>maxt)

{

mark[x]=false;

dfs(x+1);

}

}

int main()

{

while(~scanf("%d",&n),n)

{

memset(map,0,sizeof(map));

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

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

{

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

{

scanf("%d",&map[i][j]);

}

}

cn=maxt=0;

dfs(1);

printf("%d\n",maxt);

}

return 0;

}