最小树形图

int n, m;

struct Edge {

int u, v, c;

} e[M];

int realRoot, pre[N], in[N], id[N], vis[N];

int directedMST(int root, int n, int m) {

int ret = 0;

while(true) {

memset(in, 0x3f, sizeof in);

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

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

if(u != v && e[i].c < in[v]) {

if(u == root) realRoot = i;

pre[v] = u;

in[v] = e[i].c;

}

}

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

if(i == root) continue;

if(in[i] == INF) return -1;

}

int cnt = 0;

memset(id, -1, sizeof id);

memset(vis, -1, sizeof vis);

in[root] = 0;

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

ret += in[i];

int u = i;

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

vis[u] = i;

u = pre[u];

}

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

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

id[v] = cnt;

}

id[u] = cnt++;

}

}

if(cnt == 0) break;

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

if(id[i] == -1) id[i] = cnt++;

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

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

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

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

if(u != v) e[i].c -= in[v];

}

n = cnt;

root = id[root];

}

return ret;

}

## K短路

int n,m,s,t,k;

struct qnode{

int v,c;

qnode(int \_v=0,int \_c=0):v(\_v),c(\_c){}

bool operator <(const qnode &r)const{

return c>r.c;

}

};

struct Edge{

int v,cost;

Edge(int \_v=0,int \_cost=0):v(\_v),cost(\_cost){}

};

vector<Edge>E[maxn],E2[maxn];

bool vis[maxn];

int dist[maxn];

void Dijkstra(int n,int start){//点的编号从1开始

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

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

dist[i]=inf;

priority\_queue<qnode>que;

while(!que.empty())

que.pop();

dist[start]=0;

que.push(qnode(start,0));

qnode tmp;

while(!que.empty()){

tmp=que.top();

que.pop();

int u=tmp.v;

if(vis[u])

continue;

vis[u]=true;

for(int i=0; i<E2[u].size(); i++){

int v=E2[tmp.v][i].v;

int cost=E2[u][i].cost;

if(!vis[v]&&dist[v]>dist[u]+cost){

dist[v]=dist[u]+cost;

que.push(qnode(v,dist[v]));

}

}

}

}

void addedge(int u,int v,int w){

E[u].push\_back(Edge(v,w));

E2[v].push\_back(Edge(u,w));

}

struct Node{

int u;

int h,g;

Node(int u\_,int h\_,int g\_){u=u\_,h=h\_,g=g\_;}

bool operator<(Node a)const{

return h+g>a.h+a.g;

}

};

int cnt[maxn];

int Astar(int s,int t,int k){

priority\_queue<Node>que;

while(!que.empty())

que.pop();

memset(cnt,0,sizeof cnt);

if(s==t)

k++;

que.push(Node(s,0,dist[s]));

while(!que.empty()){

Node u = que.top(); que.pop();

cnt[u.u]++;

if(cnt[u.u]==k&&u.u==t)

return u.h;

if(cnt[u.u]>k)

continue;

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

int v=E[u.u][i].v;int w=E[u.u][i].cost;

que.push(Node(v,u.h+w,dist[v]));

}

}

return -1;

}

int main(){

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

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

E[i].clear();

E2[i].clear();

}

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

int u,v,w; scanf("%d%d%d",&u,&v,&w);

addedge(u,v,w);

}

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

Dijkstra(n,t);

printf("%d\n",Astar(s,t,k));

}

return 0;

}

int head[MAXN],head1[MAXN];

LL dis[MAXN];

bool vis[MAXN];

int n,m,tot,st,en,k;

struct Edge{

int u,v,nxt,nxt1;

LL c;

Edge(){}

Edge(int \_u,int \_v,LL \_c):u(\_u),v(\_v),c(\_c){}

}e[MAXN\*2];

struct qnode{

int v;

LL c;

qnode(){}

qnode(int \_v,LL \_c):v(\_v),c(\_c){}

bool operator < (const qnode& rhs) const{

return c+dis[v]>rhs.c+dis[rhs.v];

}

};

void addedge(int u,int v,LL c){

e[tot]=Edge(u,v,c);

e[tot].nxt=head[u];head[u]=tot;

e[tot].nxt1=head1[v];head1[v]=tot++;

}

void dij(int src){

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

for(int i=1;i<=n;i++) dis[i]=INF;

dis[src]=0;

priority\_queue<qnode> que;

que.push(qnode(src,0));

while(!que.empty()){

qnode pre=que.top(); que.pop();

vis[pre.v]=true;

for(int i=head1[pre.v];i!=-1;i=e[i].nxt1){

if(dis[e[i].u]>dis[pre.v]+e[i].c){

dis[e[i].u]=dis[pre.v]+e[i].c;

que.push(qnode(e[i].u,0));

}

}

while(!que.empty()&&vis[que.top().v]) que.pop();

}

}

LL a\_star(int src){

priority\_queue<qnode> que;

que.push(qnode(src,0));

k--;

while(!que.empty()){

qnode pre=que.top();que.pop();

if(pre.v==en){

if(k) k--;

else return pre.c;

}

for(int i=head[pre.v];i!=-1;i=e[i].nxt)

que.push(qnode(e[i].v,pre.c+e[i].c));

}

return -1;

}

int main(){

int T; scanf("%d",&T);

while(T--){

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

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

memset(head1,-1,sizeof(head1));

tot=0;

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

int u,v;LL c;

scanf("%d%d%lld",&u,&v,&c);

addedge(u,v,c); addedge(v,u,c);

}

st=1,en=n;k=2;

dij(en);

if(st==en) k++;

printf("%lld\n",a\_star(st));

}

return 0;

}

## 4.LCA st表

/\*

HDU 5296

\*/

struct Edge{

int u,v,w,next;

};

struct LCA{

int dp[2\*maxn][50];

int tot,head[maxn];

Edge edge[maxn\*2];

void addedge(int u,int v,int w,int &cnt){

edge[cnt].u=u; edge[cnt].v=v; edge[cnt].w=w;

edge[cnt].next=head[u]; head[u]=cnt++;

}

//深度 距离

int d[maxn],dis[maxn];

//欧拉序列，就是dfs遍历的顺序，长度为2\*n-1,下标从1开始

//表示点i在ver中第一次出现的位置(节点i的dfs序)

int ver[2\*maxn],first[maxn];

bool vis[maxn];

void init(){

tot=0; dis[1]=0;

memset(head,-1,sizeof head);

memset(vis,false,sizeof vis);

}

void dfs(int u,int dep,int pre){

vis[u]=true;

ver[++tot]=u;

first[u]=tot;

d[tot]=dep;

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

int v = edge[i].v,w = edge[i].w;

if(vis[v])

continue;

dis[v]=dis[u]+w;

dfs(v,dep+1,u);

ver[++tot]=u;

d[tot]=dep;

}

}

void ST(int n){

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

dp[i][0]=i;

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

for(int i=1; i+(1<<j)-1<=n; i++){

int a=dp[i][j - 1],b=dp[i+(1<<(j-1))][j-1];

dp[i][j]=d[a]<d[b]?a:b;

}

}

int RMQ(int l,int r){

int k=0;

while((1<<(k+1))<=r-l+1)

k++;

int a=dp[l][k],b=dp[r-(1<<k)+1][k];

return d[a]<d[b]?a:b;

}

int Lca(int u,int v){

int x = first[u], y = first[v];

if(x>y)

swap(x,y);

return ver[RMQ(x,y)];

}

}lca;

set<int> s; set<int>::iterator it;

bool vis[maxn];//判断点是否在集合中

int calc(int u){

if(s.empty())

return 0;

it=s.upper\_bound(u);

int x,y;//x,y分别是比dfs序为u的点大的最小点和比它小的最大点

if(it==s.end()||it==s.begin()){

x = lca.ver[\*s.rbegin()];

y = lca.ver[\*s.begin()];

}

else{

x = lca.ver[\*it];

it--;

y = lca.ver[\*it];

}

u = lca.ver[u];

return lca.dis[u] - lca.dis[lca.Lca(x,u)] - lca.dis[lca.Lca(y,u)] + lca.dis[lca.Lca(x,y)];

}

int main(){

int T; scanf("%d",&T);

for(int kase=1; kase<=T; kase++){

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

memset(vis,false,sizeof vis);

s.clear(); lca.init(); int cnt=0;

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

int u,v,w; scanf("%d%d%d",&u,&v,&w);

lca.addedge(u,v,w,cnt);

lca.addedge(v,u,w,cnt);

}

lca.dfs(1,-1,-1); lca.ST(2\*n-1);

printf("Case #%d:\n",kase);

int op,u,ans=0;

while(m--){

scanf("%d%d",&op,&u);

if(op==1){

if(!vis[u]){

vis[u]=true;

ans += calc(lca.first[u]);

s.insert(lca.first[u]);

}

}

else{

if(vis[u]){

vis[u]=false;

s.erase(lca.first[u]);

ans-=calc(lca.first[u]);

}

}

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

}

}

return 0;

}

## 4.全局最小割 Stoer-Wagner

//全局最小割

**割：**在一个图G（V，E）中V是点集，E是边集。在E中去掉一个边集C使得G（V，E-C）不连通，C就是图G（V，E）的一个割；

**最小割：**在G（V，E）的所有割中，边权总和最小的割就是最小割。

int n,m;

int vis[maxn];

int wet[maxn];

int combine[maxn];

int e[maxn][maxn];

int s,t,mincut;

void Search(){

int i,j,Max,tmp;

memset(vis,0,sizeof(vis));

memset(wet,0,sizeof(wet));

s=t=-1;

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

Max=-inf;

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

if (!combine[j]&&!vis[j]&&wet[j]>Max){

tmp=j;

Max=wet[j];

}

}

if(t==tmp)

return;

s=t; t=tmp;

mincut=Max;

vis[tmp]=1;

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

if(!combine[j]&&!vis[j]){

wet[j]+=e[tmp][j];

}

}

}

}

int Stoer\_Wagner(){

int i,j;

memset(combine,0,sizeof(combine));

int ans=inf;

for (i=0; i<n-1; i++){

Search();

if(mincut<ans)

ans=mincut;

if(ans==0)

return 0;

combine[t] = 1;

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

if(!combine[j]){

e[s][j]+=e[t][j];

e[j][s]+=e[j][t];

}

}

}

return ans;

}

int main(){

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

memset(e,0,sizeof(e));

while(m--){

int u,v,w;

scanf("%d%d%d", &u, &v, &w);

e[u][v]+=w;

e[v][u]+=w;

}

printf("%d\n",Stoer\_Wagner());

}

return 0;

}

## 6.判断弦图

弦：连接环中不相邻的两点的边

弦图：当图中任意长度大于3的环都至少有一个弦

int n,m;

bool g[maxn][maxn],vis[maxn];

int lable[maxn],set[maxn];

void Relable(){

memset(vis,false,sizeof vis);

vis[1]=true;

for(int num=n-1; num>0; num--){

memset(lable,0,sizeof lable);

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

if(!vis[i]){

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

if(g[i][set[n-j+1]])

lable[i]++;

}

}

int maxv=0,mx;

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

if(lable[i]>maxv){

maxv=lable[i];

mx=i;

}

}

set[num]=mx;

vis[mx]=true;

}

}

bool check(){

int tmp[maxn];

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

memset(tmp,0,sizeof tmp);

int t=0;

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

if(g[set[i]][set[j]]){

t++;

tmp[t]=set[j];

}

}

for(int j=2; j<=t; j++)

if(!g[tmp[j]][tmp[1]])

return false;

}

return true;

}

int main(){

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

if(n==0)

break;

memset(g,false,sizeof g);

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

int u,v; scanf("%d%d",&u,&v);

g[u][v]=g[v][u]=true;

}

set[n]=1;

Relable();

if(check())

printf("Perfect\n");

else

printf("Imperfect\n");

printf("\n");

}

return 0;

}

## 7.判断仙人掌图

/\*

判断是不是仙人掌图 HDU3594

性质1: 仙人掌图的DFS树没有横向边

性质2: low(v) <= dfn(u) v是u的儿子

性质3: 设某个点 u 有 au 个儿子的low值小于dfn(u),同时u自己有bu条逆向边 则au+bu<2

\*/

int n,index;

bool flag;

struct Edge{

int to,next;

}edge[maxn];

int tot,head[maxn];

int vis[maxn];

int cnt[maxn],dfn[maxn],low[maxn];

//cnt表示经过的环数目

//dfn表示的是其在树中的深度，low表示其经过移动可达到的最大高度（最小深度）

void addedge(int u,int v){

edge[tot].to=v; edge[tot].next=head[u]; head[u]=tot++;

}

void dfs(int u){

int v;

if(!flag)

return;

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

vis[u]=1;

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

int v = edge[i].to;

if(vis[v]==2){//出现了横向边

flag=false;

return;

}

if(!vis[v]){

dfs(v);

//如果其子节点所能达到的最大高度小于父节点，那么子节点一定不能达到父节点，所以不是连通图

if(low[v]>dfn[u]){

flag=false;

return;

}

if(low[v]<dfn[u]){

++cnt[u];//经过u的环+1

if(low[v]<low[u])//更新所能达到的最大高度。

low[u]=low[v];

}

}

else if(low[v]<dfn[u]){

++cnt[u];//经过u的环+1

if(low[v]<low[u])//更新所能达到的最大高度。

low[u]=low[v];

}

if(cnt[u]>1){//性质3

flag=false;

return;

}

}

vis[u]=2;//以u为根节点 的子树访问完毕，如果再有访问的话就是横向边.

}

void init(){

tot=0;

memset(head,-1,sizeof head);

memset(cnt,0,sizeof cnt);

memset(vis,false,sizeof vis);

}

int main(){

int T; scanf("%d",&T);

while(T--){

init();

scanf("%d",&n);

int u,v;

while(scanf("%d%d",&u,&v)){

if(u==0&&v==0)

break;

addedge(u+1,v+1);

}

flag=true;

index=0;

dfs(1);

if(!flag||index<n)

puts("NO");

else

puts("YES");

}

return 0;

}

## 8.Dinic

//Dinic

struct Dinic{

struct Edge{

int to,next,cap;

}edge[maxn\*maxn];

int tot,head[maxn],dist[maxn],s,t;

//bool vis[maxn];//判断是否选中

void init(int ss,int tt){

s=ss; t=tt;

tot=0;

memset(head,-1,sizeof head);

}

void addedge(int u,int v,int w){

edge[tot]=(Edge){v,head[u],w};

head[u]=tot++;

edge[tot]=(Edge){u,head[v],0};

head[v]=tot++;

}

bool bfs(){

queue<int> que;

que.push(s);

memset(dist,-1,sizeof dist);

// memset(vis,false,sizeof vis);

dist[s]=0;

while(!que.empty()){

int u=que.front(); que.pop();

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

if(dist[edge[i].to]==-1&&edge[i].cap>0){

dist[edge[i].to]=dist[u]+1;

//vis[edge[i].to]=true;

que.push(edge[i].to);

}

}

}

return dist[t]!=-1;

}

int dfs(int start, int curFlow){

if(start==t)

return curFlow;

int i,minFlow=0,v,temp;

for(i=head[start]; i!=-1; i=edge[i].next){

v=edge[i].to;

if(dist[start]==dist[v]-1&&edge[i].cap > 0){

temp=dfs(v,min(edge[i].cap,curFlow));

edge[i].cap-=temp;

edge[i^1].cap+=temp;

curFlow-=temp;

minFlow+=temp;

if(0==curFlow)

break;

}

}

if(0==minFlow)

dist[start]=-2;

return minFlow;

}

int max\_flow(){

int res=0;

while(bfs()){

res+=dfs(s,inf);

}

return res;

}

}dinic;

## 9.最大密度子图

// 给定一个无向图，要求它的一个子图，使得子图中边数|E|与点数|V|的比值最大

int n,m,s,t,cnt,sum,degree[maxn];

bool vis[maxn];

struct Node{

int x,y;

}p[maxn];

//Dinic

//用于表示边的结构体(终点、容量、反向边)

struct edge{

int to,rev;

double cap;

edge(int to,double cap,int rev) :to(to), cap(cap), rev(rev){}

};

vector<edge> G[maxn];//图的领接表表示

int level[maxn];//顶点到源点的距离标号

int iter[maxn];//当前弧，在其之前的边已经没有用了

void add\_edge(int from,int to,double cap){

G[from].push\_back(edge(to, cap, G[to].size()));

G[to].push\_back(edge(from, 0, G[from].size()-1));

}

//通过BFS计算从源点出发的距离标号

void bfs(int s){

memset(level,-1,sizeof level);

queue<int> que;

level[s]=0;

que.push(s);

while(!que.empty()){

int v=que.front(); que.pop();

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

edge&e=G[v][i];

if(e.cap>0&&level[e.to]<0){

level[e.to]=level[v]+1;

que.push(e.to);

}

}

}

}

//通过DFS寻找增广路

double dfs(int v,int t,double f){

if(v==t)

return f;

for(int &i=iter[v]; i<G[v].size(); i++){

edge&e=G[v][i];

if(e.cap>0&&level[v]<level[e.to]){

double d=dfs(e.to,t,min(f,e.cap));

if(d>eps){

e.cap-=d;

G[e.to][e.rev].cap+=d;

return d;

}

}

}

return 0;

}

//求解从s到t的最大流

double max\_flow(int s,int t){

double flow=0;

for(;;){

bfs(s);

if(level[t]<0)

return flow;

memset(iter,0,sizeof iter);

double f;

while((f=dfs(s,t,inf))>0){

flow+=f;

}

}

}

void init(double mid){

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

G[i].clear();

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

add\_edge(s,i,m);

add\_edge(i,t,m+2\*mid-degree[i]);

}

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

add\_edge(p[i].x,p[i].y,1.0);

add\_edge(p[i].y,p[i].x,1.0);

}

}

void dfs(int v){

cnt++;

vis[v]=true;

vector<edge> gv=G[v];

for (vector<edge>::iterator it = gv.begin(); it != gv.end(); ++it){

const edge &e = \*it;

if (e.cap > eps && !vis[e.to])

dfs(e.to);

}

}

int main(){

memset(vis,false,sizeof vis);

memset(degree,0,sizeof degree);

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

if(m==0){

printf("1.00000\n");

return 0;

}

s=0,t=n+1;

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

scanf("%d%d",&p[i].x,&p[i].y);

degree[p[i].x]++; degree[p[i].y]++;

}

double l=0,r=m,mid,hg;

const double epss=(1.0/n/n);

while(r-l>=epss){

// PRI(hg)

mid=(l+r)/2;

init(mid);

hg=(n\*m-max\_flow(s,t))/2;

if(hg>eps)

l=mid;

else

r=mid;

}

// printf("%lf\n",l);

init(l);

max\_flow(s,t);

cnt=0;

dfs(0);

int sum=0;

// printf("%d\n",cnt-1);

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

if(vis[p[i].x]&&vis[p[i].y])

sum++;

}

// PRI(sum)

double ans=double(sum)/(cnt-1);

printf("%.5f\n",ans);

return 0;

}

## 10.最大团

/\*

1、最大团点的数量=补图中最大独立集点的数量

2、二分图中，最大独立集点的数量+最小覆盖点的数量=整个图点的数量

3、二分图中，最小覆盖点的数量=最大匹配的数量

4、图的染色问题中，最少需要的颜色的数量=最大团点的数量

\*/

// 团：完全图

// 1. 最大团点的数量 = 补图中最大独立集点的数量

// 2. 图的染色问题中，最少需要的颜色的数量 = 最大团点的数量

struct MaxClique {

static const int V = 60;

bool g[V][V];

int n, ans, max[V], adj[V][V];

int path[V], clique[V]; //for record

//max[i]:= [i, n]'s maximum clique

//adj[dep][i]:= available vertices

void init(int \_n) { n = \_n; }

bool dfs(int cur, int dep) {

if(cur == 0) {

if(dep > ans) {

ans = dep;

swap(clique, path);

return 1;

}

return 0;

}

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

if(dep + cur - i <= ans) return 0; //dep + left <= ans

int u = adj[dep][i], nxt = 0;

if(dep + max[u] <= ans) return 0; //same as above

path[dep] = u;

for(int j = i + 1; j < cur; ++j) {

int v = adj[dep][j];

if(g[u][v]) adj[dep + 1][nxt++] = v;

}

if(dfs(nxt, dep + 1)) return 1;

}

return 0;

}

int maxClique() {

ans = 0;

memset(max, 0, sizeof max);

for(int i = n - 1; ~i; --i) {

int cur = 0;

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

if(g[i][j]) adj[1][cur++] = j;

path[0] = i;

dfs(cur, 1);

max[i] = ans;

}

return ans;

}

} solver;

## 11.最小费用最大流

//最小费用最大流，求最大费用只需要取相反数，结果取相反数即可。

//点的总数为 N，点的编号 0~N-1

struct Edge{

int to,next,cap,flow,cost;

}edge[maxm];

int head[maxn],tol;

int pre[maxn];

int dis[maxn];//double???

bool vis[maxn];

int N;

void init(int n){

N=n;

tol=0;

memset(head,-1,sizeof head);

}

void addedge(int u,int v,int cap,int cost){

edge[tol].to=v; edge[tol].cap=cap; edge[tol].cost=cost; edge[tol].flow=0;

edge[tol].next=head[u]; head[u]=tol++;

edge[tol].to=u; edge[tol].cap=0; edge[tol].cost=-cost; edge[tol].flow=0;

edge[tol].next=head[v]; head[v]=tol++;

}

bool spfa(int s,int t){

queue<int> q;

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

dis[i]=inf;

vis[i]=false;

pre[i]=-1;

}

dis[s]=0;

vis[s]=true;

q.push(s);

while(!q.empty()){

int u=q.front(); q.pop();

vis[u]=false;

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

int v=edge[i].to;

if(edge[i].cap>edge[i].flow&&dis[v]>dis[u]+edge[i].cost){

dis[v]=dis[u]+edge[i].cost;

pre[v]=i;

if(!vis[v]){

vis[v]=true;

q.push(v);

}

}

}

}

if(pre[t]==-1)

return false;

else

return true;

}

//返回的是最大流，cost存的是最小费

int MincostMaxflow(int s,int t,int &cost){

int flow=0;

cost=0;

while(spfa(s,t)){

int Min=inf;

for(int i=pre[t]; i!=-1; i=pre[edge[i^1].to]){

if(Min>edge[i].cap-edge[i].flow)

Min=edge[i].cap-edge[i].flow;

}

for(int i=pre[t]; i!=-1; i=pre[edge[i^1].to]){

edge[i].flow+=Min;

edge[i^1].flow-=Min;

cost+=edge[i].cost\*Min;

}

flow+=Min;

}

return flow;

}

## 12.有向图欧拉回路计数

/\*

Trees可以通过基尔霍夫矩阵求出。那什么是基尔霍夫矩阵呢？假设D(G)为图G的度数矩阵，在这个矩阵中，当i=j时

有d[i][j]为点i的度数，当i≠j时，d[i][j]=0；再假设矩阵A(G)为图G的邻接矩阵，在这个矩阵中，若i点和j点

可达，则有a[i][j]=1，否则a[i][j]为0。那么这个时候就可以令基尔霍夫矩阵为C(G)=D(G)-A(G)。然后通过求基尔霍

夫矩阵的的行列值就可以了。由于要求以1号结点为根节点的生成树的个数，那么计算2~n结点的n-1阶基尔霍夫矩阵

的行列值就行了。Deg[1]表示1号结点的入度。

资料：

求有向图的欧拉回路个数，是BEST定理

ec(G)=ts(G)?deg(s)!?∏v∈V, v≠s(deg(v)?1)!, ts(G):=以s为根的外向树的个数

如何计算有向图的外向树个数，这要用到MatrixTree定理

基尔霍夫矩阵K=度数矩阵D?邻接矩阵A

无向图的度数矩阵就是每个点自己的度数

有向图的度数矩阵就是每个点自己的入度

邻接矩阵是表示u?>v边的个数的矩阵

重边：按照边数计算，自环：不计入度数

无向图生成树计数：c=|K的任意1个n?1阶主子式|

有向图外向树计数：c=|去掉根所在的那阶得到的主子式|

\*/

const LL MOD=998244353;

const int MAXV=400+1;

int V;

LL D[MAXV][MAXV];//从i，到j的边的数目

LL in[MAXV],out[MAXV];//每个结点的入度，出度

struct Matrix{

LL a[MAXV][MAXV];

Matrix(){

memset(a,0,sizeof(a));

}

LL det(int n){//求前n行n列的行列式的值

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

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

a[i][j]=(a[i][j]%MOD+MOD)%MOD;

LL ret=1;

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

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

while(a[j][i]){

LL t=a[i][i]/a[j][i];

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

a[i][k]=((a[i][k]-a[j][k]\*t)%MOD+MOD)%MOD;

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

swap(a[i][k],a[j][k]);

ret=-ret;

}

if(!a[i][i])

return 0;

ret=ret\*a[i][i]%MOD;

}

ret=(ret%MOD+MOD)%MOD;

return ret;

}

};

LL get\_fac(LL x){//计算阶乘

LL res=1;

for(LL i=2;i<=x;++i)

res=(res\*i)%MOD;

return res;

}

LL exgcd(LL a, LL b, LL &x, LL &y){

LL d=a;

if(b){

d=exgcd(b, a%b, y, x);

y-=(a/b)\*x;

}

else{

x=1; y=0;

}

return d;

}

LL inv(LL a){//计算逆元

LL x, y;

exgcd(a, MOD, x, y);

return (MOD+x%MOD)%MOD;

}

void init(){

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

in[i]=out[i]=0;

}

int main(){

int cas=1;

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

init();

Matrix mat;

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

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

scanf("%lld", &D[i][j]);

mat.a[i][j]-=D[i][j];

mat.a[j][j]+=D[i][j];

in[j]+=D[i][j];

out[i]+=D[i][j];

}

//如果存在点入度不等于出度，则不存在欧拉回路直接输出0

bool ok=true;

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

if(in[i]!=out[i]){

ok=false;

break;

}

if(!ok){

printf("Case #%d: 0\n", cas++);

continue;

}

//把根节点移到最后，方便去掉它求行列式

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

swap(mat.a[0][i], mat.a[V-1][i]);

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

swap(mat.a[i][0], mat.a[i][V-1]);

LL ans=mat.det(V-1);

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

ans=(ans\*get\_fac(in[i]-(i!=0)))%MOD;

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

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

ans=(ans\*inv(get\_fac(D[i][j])))%MOD;

printf("Case #%d: %lld\n", cas++, ans);

}

return 0;

}

## 12．Tarjan+集合合并

//HDU6041

int n,m,k;

int tot,head[maxn];

struct Edge{

int to,next,cost;

}edge[maxn];

void addedge(int u,int v,int cost){

edge[tot].to=v; edge[tot].cost=cost;

edge[tot].next=head[u]; head[u]=tot++;

}

struct Node{

int val,id1,id2;

Node(int val,int id1,int id2):val(val),id1(id1),id2(id2){}

bool operator < ( const Node & a ) const{

return val<a.val;

}

};

int dfn[maxn],low[maxn],index;

bool vis[maxn];

stack<int> st;

int a[maxn],b[maxn],vec[maxn];

void init(){

index=0,tot=0; a[0]=0;

memset(vis,false,sizeof vis);

memset(head,-1,sizeof head);

}

void unite(int \*a,int \*b){

priority\_queue<Node> que;

for(int i=1; i<=b[0]; i++)

que.push(Node(a[1]+b[i],1,i));

vec[0]=0;

while(vec[0]<k&&!que.empty()){

Node now=que.top(); que.pop();

vec[++vec[0]]=now.val;

if(now.id1+1<=a[0]){

now.id1++;

que.push(Node(a[now.id1]+b[now.id2],now.id1,now.id2));

}

}

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

a[i]=vec[i];

}

void Tarjan(int u,int pre){

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

vis[u]=true;

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

int v = edge[i].to;

if(v==pre) continue;

if(!vis[v]){

st.push(i);

Tarjan(v,u);

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

if(low[v]>=dfn[u]){//u是割点

b[0]=0;

int tmp;

do{

tmp=st.top(); st.pop();

b[++b[0]]=edge[tmp].cost;

}while(tmp!=i);

if(b[0]>1)//找到环

unite(a,b);

}

}

if(vis[v]&&dfn[v]<dfn[u]){//防止一个边被加入多次

st.push(i);

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

}

}

}

int main(){

int kase=1;

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

init();

int sum=0;

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

int u,v,w; scanf("%d%d%d",&u,&v,&w);

addedge(u,v,w); addedge(v,u,w);

sum+=w;

}

scanf("%d",&k);

a[++a[0]]=0;

Tarjan(1,-1);

unsigned ans=0;

for(int i=1; i<=a[0]; i++)

ans+=(i\*(sum-a[i]));

printf("Case #%d: %u\n",kase++,ans);

}

return 0;

}

## 13.去吧floyd

1.

最大权闭合子图：

最大权闭合子图指选择u，则u以下关系的都要选，一定要选到底，不能跳过u选它以下的。增设一个超级源点和一个超级汇点，（1->n）的点中，当点权为正时，从源点向该点连一条权值为点权大小的边，当点权为负时，从该点连一条权值大小为它的绝对值的边连向汇点。对于（u，v），如果选择u必须选择v，对（u，v）连一条容量为 ∞ 的边。

结论：

最小割为简单割  闭合图是简单割  简单割是闭合图

最小割所产生的两个集合中，其源点S所在集合(除去S)为最大权闭合图。 答案数等于靠近源点最小割一边的点数，最大利益==所有点正权值之和-最小割。

2.

上下界网络流

**1：无源汇上下界可行流（循环流）：**

对于每根管子有一个上界容量up和一个下界容量low，我们让这根管子的容量下界变为0，上界为up-low。

可是这样做了的话流量就不守恒了，为了再次满足流量守恒，即每个节点"入流=出流”，我们增设一个超级源点st和一个超级终点sd。我们开设一个数组du[]来记录每个节点的流量情况。

du[i]=in[i]（i节点所有入流下界之和）-out[i]（i节点所有出流下界之和）。

当du[i]大于0的时候，st到i连一条流量为du[i]的边。

当du[i]小于0的时候，i到sd连一条流量为-du[i]的边。

最后对（st，sd）求一次最大流即可，当所有附加边全部满流时（即maxflow==所有du[]>0之和），有**可行解**。

**2：有源汇上下界可行流：**

模型:现在的网络有一个**源点s和汇点t**,求出一个流使得源点的总流出量等于汇点的总流入量,其他的点满足流量守恒,而且每条边的流量满足上界和下界限制.

源点s的流出量等于汇点t的流入量,我们就可以从汇点t向源点s连一条下界为0上界为无穷大的边,相当于把从源点s流出的流量再流回来.在这样的图中套用上面的算法求出一个可行的循环流,拆掉从汇点t到源点s的边就得到一个可行的有源汇流.

**3：有源汇上下界最大流：**

模型:现在的网络有一个源点s和汇点t,求出一个流使得源点的总流出量等于汇点的总流入量,其他的点满足流量守恒,而且每条边的流量满足上界和下界限制.在这些前提下要求**总流量最大**.

套用上面的算法求出一个有源汇有上下界可行流.此时的流不一定最大.

接下来在残量网络上跑s-t最大流即可.(删除与超源超汇相连的边)

最终的最大流流量=可行流流量(即t到s的无穷边上跑出的流量)( head[t]^1 )+新增广出的s-t流量

**4：有源汇上下界最小流：**

模型:现在的网络有一个源点s和汇点t,求出一个流使得源点的总流出量等于汇点的总流入量,其他的点满足流量守恒,而且每条边的流量满足上界和下界限制.在这些前提下要求**总流量最小**.

理解一下dinic的反向边.反向边的流量增加等价于正向边的的流量减少.因此我们在残量网络上找出t到s的流就相当于减小了s到t的流,因此我们在跑出可行流的残量网络上跑t-s最大流,用可行流的大小减去这一次t-s最大流的大小就是最小流的大小.(t-s最大流其实是尽量缩减s-t方向的流).

ans+=dinic.max\_flow();

dinic.addedge(t,s,inf,0);

ans+=dinic.max\_flow();