附录：

SPFA：

判负环：

虫洞是可以回到过去的有向边。John的每个农场有M条小路（无向边)连接着N（从1..N标号）块地，并有W个虫洞。现在John想借助这些虫洞来回到过去，请你告诉他能办到吗。

思路：即判负环。

代码：

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <set>

#include <climits>

#include <cmath>

#include <queue>

using namespace std;

const int N = 510, INF = 0x3f3f3f3f, MOD = 100000000;

struct edge

{

int to, c, next;

}g[N\*N\*2];

int cnt, head[N];

int d[N], num[N];

bool vis[N];

inline void add(int v, int u, int c)

{

g[cnt].to = u, g[cnt].c = c, g[cnt].next = head[v], head[v] = cnt++;

}

bool spfa(int s, int t)

{

queue<int> que;

memset(d, 0x3f, sizeof d);

memset(num, 0, sizeof num);

memset(vis, 0, sizeof vis);

que.push(s), d[s] = 0, vis[s] = 1, num[s]++;

while(! que.empty())

{

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

vis[v] = 0;

for(int i = head[v]; i != -1; i = g[i].next)

{

int u = g[i].to;

if(d[u] > d[v] + g[i].c)

{

d[u] = d[v] + g[i].c;

if(! vis[u])

{

que.push(u), vis[u] = 1, num[u]++;

if(num[u] > t) return 1;//入队次数超过点数，存在负环

}

}

}

}

return 0;

}

int main()

{

int t, n, m, k;

scanf("%d", &t);

while(t--)

{

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

cnt = 0;

memset(head, -1, sizeof head);

int v, u, c;

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

{

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

add(v, u, c), add(u, v, c);

}

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

{

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

add(v, u, -c);

}

if(spfa(1, n)) printf("YES\n");

else printf("NO\n");

}

return 0;

}

差分约束：

POJ3169:

一共有n头牛，有ml对关系好的牛，md对关系不好的牛，接下来ml行，每行三个元素A,B,D，表示A牛和B牛相距不希望超过D，接下来md行，每行三个元素A,B,D表示A牛和B牛的相距至少要有D才行。求1号牛和n号牛的最大距离。

代码：

const int N = 1010;

const int E = 20020;

const int INF = 0x3f3f3f3f;

int head[N];//每个结点的头指针

int vis[N];//在队列标志

int num[N];//每个点的入队列次数

int que[N];//spfa循环指针

int dis[N];

struct P {

int to;

int v;

int nxt;

} e[E];

int tot;

void add(int a, int b, int v)//加边

{

e[tot].to = b;

e[tot].v = v;

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

head[a] = tot++;

}

int n;

int spfa(int s) {

int front = 0, rear = 0;

for (int v = 1; v <= n; v++)//初始化

{

if (v == s) {

que[rear++] = v;

vis[v] = 1;

num[v] = 1;

dis[v] = 0;

} else {

vis[v] = 0;

num[v] = 0;

dis[v] = INF;

}

}

while (front != rear) {

int u = que[front++];

vis[u] = 0;

if (front >= N)front = 0;

if (++num[u] > n) return -1;

//num[i]为入队列次数，用来判断是否存在负环回来

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

int v = e[i].to;

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

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

if (!vis[v]) {

vis[v] = 1;

que[rear++] = v;

if (rear >= N)rear = 0;

}

}

}

}

if (dis[n] == INF) return -2;

return dis[n];

}

void init(){

tot = 0;

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

}

int main() {

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

// freopen("out.txt", "w", stdout);

int ML, MD;

int a, b, c;

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

init();

while (ML--) {

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

if (a > b) swap(a, b);//注意加边顺序

add(a, b, c);

//大-小<=c ,有向边(小，大）：c

}

while (MD--) {

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

if (a < b)swap(a, b);

add(a, b, -c);

//大-小>=c,小-大<=-c,有向边(大，小):-c

}

printf("%d\n",spfa(1));

}

return 0;

}

差分约束：

最长路求最小值：

using namespace std;

const int N = 5e4 + 5;

const int inf = 0x3f3f3f3f;

int n,cnt=0;

struct {

int to,nxt,w;

}e[4\*N];

int head[N];

int dis[N];

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

e[cnt].to=v;

e[cnt].w=w;

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

head[u]=cnt++;

}

bool ins[N];

int des;

int spfa(int s){//最长路

memset(ins,0, sizeof(ins));

memset(dis,-inf, sizeof(dis));//初始化

dis[s]=0;

queue<int> q;

q.push(s),ins[s]=1;

while(!q.empty()){

int u=q.front();

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

for(int i=head[u];~i;i=e[i].nxt){

int v=e[i].to;

int w=e[i].w;

if(dis[u]+w>dis[v]){

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

if(!ins[v]) q.push(v),ins[v]=1;

}

}

}

return dis[des];

}

int main() {

ci(n);

des=0;

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

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

int x,y,z;

ci(x),ci(y),ci(z);

add(x-1,y,z);//s[y]-s[x-1]>=z

des=max(des,y);

}

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

add(i-1,i,0);//s[i]-s[i-1]>=0

add(i,i-1,-1);//s[i-1]-s[i]>=-1

}

int ans=spfa(0);

pi(ans);

return 0;

}

次短路：

思路：次短路肯定是最短路中某一条边不走，而走了其他边再回到最短路上，而且只可能绕一个地方，我们枚举每条边<s,t>

有d[s]—>源点到s的最短距离

有dr[t]—>t到汇点的最短距离，这样就需要从t到s求一次最短路得到了

然后我们枚举每一条边有：tmp=d[s]+dr[t]+len<s,t> 取最小且大于最短路的结果

#include <iostream>

#include <queue>

using namespace std;

#define MAXV 5010

#define MAXE 200100

#define INF 1<<29

typedef struct{

int t,w,next;

}Edge;

Edge edge[MAXE];

int d[MAXV],dr[MAXV];

int n,m,edge\_sum;

int head[MAXV];

bool vis[MAXV];

void init(){

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

edge\_sum=0;

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

}

void addedge(int s,int t,int w){

edge[edge\_sum].t=t;

edge[edge\_sum].w=w;

edge[edge\_sum].next=head[s];

head[s]=edge\_sum++;

}

void spfa(int source,int dt[]){

int i,v,u;

queue <int>q;

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

dt[source]=0;

vis[source]=1;

q.push(source);

while(!q.empty()){

v=q.front();q.pop();

vis[v]=0;

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

u=edge[i].t;

if(dt[v]+edge[i].w<dt[u]){

dt[u]=dt[v]+edge[i].w;

if(!vis[u]){

vis[u]=1;

q.push(u);

}

}

}

}

}

int main(){

int a,b,c;

int ans,tmp,i;

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

init();

while(m--){

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

addedge(a,b,c);//双向边

addedge(b,a,c);

}

spfa(1,d);//1->s的最短路

spfa(n,dr);//t->n的最短路

ans=INF;

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

for(int j=head[i];j!=-1;j=edge[j].next){//枚举换掉的边

b=edge[j].t;

c=edge[j].w;

tmp=d[i]+dr[b]+c;

if(tmp>d[n] && ans>tmp){//更新最小值

ans=tmp;

}

}

}

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

}

return 0;

}

K短路：spfa+A\*

#include<iostream>

#include<cstring>

#include<queue>

using namespace std;

const int N=100010;

int n,m,dis[N];

int tot,head1[N],head2[N];

bool flag[N];

struct edge

{

int to;

int w;

int next;

}e[N\*2],e2[N\*2];

struct node

{

int f;

int g;

int from;

bool operator < (node a)const

{

if(a.f==f)

return g>a.g;

return f>a.f;

}

};

void add\_edge(int u,int v,int w)

{

tot++;

e[tot].to=v;

e[tot].w=w;

e[tot].next=head1[u];

head1[u]=tot;

e2[tot].to=u;

e2[tot].w=w;

e2[tot].next=head2[v];

head2[v]=tot;

}

void prepare()

{

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

dis[i]=N;tot=0;

memset(head1,0,sizeof(head1));

memset(head2,0,sizeof(head2));

}

void spfa(int t)

{

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

dis[i]=N;

dis[t]=0;

queue<int> q;

q.push(t);

flag[t]=1;

while(!q.empty())

{

int v=q.front();

q.pop();flag[v]=0;

for(int i=head2[v];i;i=e2[i].next)

if(dis[e2[i].to]>dis[v]+e2[i].w)

{

dis[e2[i].to]=dis[v]+e2[i].w;

if(!flag[e2[i].to])

{

q.push(e2[i].to);

flag[e2[i].to]=1;

}

}

}

}

int a\_star(int s,int t,int k)

{

if(s==t) k++;

if(dis[s]==N) return -1;

priority\_queue<node> q;

int cnt=0;

node tmp,to;

tmp.from=s;

tmp.g=0;

tmp.f=tmp.g+dis[tmp.from];

q.push(tmp);

while(!q.empty())

{

tmp=q.top();

q.pop();

if(tmp.from==t) cnt++;

if(cnt==k) return tmp.g;

for(int i=head1[tmp.from];i;i=e[i].next)

{

to.from=e[i].to;

to.g=tmp.g+e[i].w;

to.f=to.g+dis[to.from];

q.push(to);

}

}

return -1;

}

int main()

{

int x,y,z,s,t,k;

while(cin>>n>>m)

{

prepare();

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

{

cin>>x>>y>>z;

add\_edge(x,y,z);

}

cin>>s>>t>>k;

spfa(t);

int ans=a\_star(s,t,k);

cout<<ans<<endl;

}

return 0;

}

最小生成树算法：

1.prim(稠密图）:

#include "cstdio"

#include "iostream"

#include "algorithm"

#include "string"

#include "cstring"

#include "queue"

#include "cmath"

#include "vector"

#define inf 0x3f3f3f

#define mj

#define db double

#define ll long long

using namespace std;

const int mod=1e9+7;

int d[105][105],f[105],n,mi,v;

bool vis[105];

int prim()

{

for(int i=0;i<n;i++) vis[i]=0,f[i]=mod;

f[0]=0;

int res=0;

while(1)

{

int v=-1;

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

if(!vis[u]&&(v==-1||f[u]<f[v])) v=u;

}

if(v==-1) break;

vis[v]=1;

res+=f[v];

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

f[u]=min(f[u],d[v][u]);

}

}

return res;

}

int main()

{

while(scanf("%d",&n)==1&&n){

memset(f,0,sizeof(f));

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

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

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

}

}

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

}

return 0;

}

2.kru算法：

挑战109:

#include "cstdio"

#include "iostream"

#include "algorithm"

#include "string"

#include "cstring"

#include "queue"

#include "cmath"

#include "vector"

#define inf 0x3f3f3f

#define mj

#define db double

#define ll long long

using namespace std;

const int N=5.5e4+5;

const int mod=1e9+7;

int u[N],v[N],w[N];

struct e{

int u,v,cost;

};

e a[N];

int cmp(e a, e b){

return a.cost>b.cost;

}

int fa[N],p[N];

int find(int x){

return fa[x]==x?x:fa[x]=find(fa[x]);

}

int kru(int n,int m){

int ans=0;

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

sort(a,a+m,cmp);

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

int x,y;

x=find(a[i].u),y=find(a[i].v);

if(x!=y) ans+=a[i].cost,fa[x]=y;

}

return ans;

}

int main()

{

int n,m,r,t;

scanf("%d",&t);

while (t--)

{

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

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

int x;

scanf("%d%d%d",&a[i].u,&x,&a[i].cost);

a[i].v=x+n;

}

int ans=kru(m+n,r);

printf("%d\n",(n+m)\*10000-ans);

}

return 0;

}

拓扑排序

#include <bits/stdc++.h> //拓扑排序dfs

using namespace std;

int G[MAXN][MAXN],c[MAXN],t,topo[MAXN];

int n;

bool dfs(int u)

{

c[u]=-1;

for(int v=1;v<=MAXN;v++)

if(G[u][v])

{

if(c[v]<0)

return false;

else if(!c[v]&&!dfs(v))

return false;

}

c[u]=1;topo[--t]=u;

return true;

}

bool toposort()

{

memset(c,0,sizeof(c));

for(int u=1;u<=MAXN;u++)

if(!c[u])

if(!dfs(u))

return false;

return true;

}

int main()

{

cin>>n;

t=MAXN;

while(n--)

{

int u,v;

cin>>u>>v;

G[u][v]=1;

}

n=MAXN-1;

if(toposort())

{

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

cout<<topo[i]<<endl;

}

else

cout<<"wrong"<<endl;

return 0;

}

#include<iostream>弱连通分量判断

#include<cstdio>

#include<vector>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn=1000+10;

int instack[maxn],dfn[maxn],low[maxn],belong[maxn],in[maxn],out[maxn];

vector<vector<int>> G(maxn);

vector<int> s;

int n,m;

int cnt=0,idex=0;

void Tarjan(int u)

{

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

s.push\_back(u);

instack[u]=1;

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

{

int v=G[u][i];

if(!dfn[v])

{

Tarjan(v);

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

}

else if(instack[v])

{

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

}

}

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

{

cnt++;

int p;

do{

p=s[s.size()-1];

instack[p]=0;

belong[p]=cnt;

s.pop\_back();

}while(u!=p);

}

}

bool Count()

{

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

memset(out,0,sizeof(out));

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

{

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

{

int v=G[i][j];

if(belong[i]!=belong[v])

{

out[belong[i]]++;

in[belong[v]]++;

}

}

}

int ans=0;

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

{

if(!out[i]) ans++;

if(!in[i]) ans++;

}

if(ans>2) return false;

return true;

}

int main()

{

int t;

scanf("%d",&t);

while(t--)

{

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

cnt=0;idex=0;

s.clear();

for(int i=1;i<=n;i++) G[i].clear();

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

memset(low,0,sizeof(low));

memset(instack,0,sizeof(instack));

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

int a,b;

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

{

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

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

}

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

{

if(!dfn[i])

{

Tarjan(i);

}

}

if(Count()) printf("Yes\n");

else printf("No\n");

}

return 0;

}

#include <stdio.h> //强联通分量判断

#include <string.h>

#include <vector>

#include <stack>

using namespace std;

#define MIN(a,b) ((a)<(b)?(a):(b))

#define N 10005 // 题目中可能的最大点数

stack<int>sta; // 存储已遍历的结点

vector<int>gra[N]; // 邻接表表示图

int dfn[N]; // 深度优先搜索访问次序

int low[N]; // 能追溯到的最早的次序

int InStack[N]; // 检查是否在栈中(2为在栈中，1为已访问，且不在栈中，0为不在)

vector<int> Component[N]; // 获得强连通分量结果

int InComponent[N]; // 记录每个点在第几号强连通分量里

int index,Cont; // 索引号，强连通分量个数

int n, m; // 点数，边数

void init()

{

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

memset(low, 0, sizeof(low));

memset(InStack, 0, sizeof(InStack));

index = Cont = 0;

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

{

gra[i].clear();

Component[i].clear();

}

while(!sta.empty())

sta.pop();

}

void tarjan(int u)

{

InStack[u] = 2;

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

sta.push(u);

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

{

int t = gra[u][i];

if (dfn[t] == 0)

{

tarjan(t);

low[u] = MIN(low[u], low[t]);

}

else if (InStack[t] == 2)

{

low[u] = MIN(low[u], dfn[t]);

}

}

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

{

++ Cont; //；连通分量数目

while (!sta.empty())

{

int j = sta.top();

sta.pop();

InStack[j] = 1;

Component[Cont].push\_back(j);

InComponent[j]=Cont;

if (j == u)

break;

}

}

}

void input(void)

{

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

{

int a,b;

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

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

}

}

void solve()

{

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

if(!dfn[i])

tarjan(i);

if(Cont>1)

puts("No");

else

puts("Yes");

}

int main()

{

while(scanf("%d%d",&n,&m),n+m)

{

init();

input();

solve();

}

}

树的重心:

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

vector<int>tree[1005];

const int inf=10000000;

int num[1005];

int n;

int minNode,minbalance;

void bfs(int u,int v){

int minson=0;

num[u]=1;

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

int son=tree[u][i];

if(son!=v){

bfs(son,u);

num[u]+=num[son];

minson=max(minson,num[son]);

}

}

minson=max(minson,n-num[u]);

if(minson<minbalance){

minbalance=minson;

minNode=u;

}

else if(minson==minbalance)

minNode=min(minNode,u);

}

int main()

{

while(cin>>n){

for(int i=0;i<1005;i++) tree[i].clear();

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

int u,v;

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

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

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

}

minNode=0;

minbalance=inf;

bfs(1,0);

cout<<minNode<<' '<<minbalance<<endl;

}

return 0;

}

二分图匹配:

const ll MAX=1005;

int mp[MAX][MAX];

int n;

int linker[MAX];

bool used[MAX];

bool dfs(int a){

for(int i=0;i<n;i++)if(mp[a][i]&&!used[i]){

used[i]=true;

if(linker[i]==-1||dfs(linker[i])){

linker[i]=a;

return true;

}

}

return 0;

}

int hungary(){ //返回最大匹配数

int res=0;

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

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

memset(used,0,sizeof(used));

if(dfs(i)) res++;

}

return res;

}

int main()

{

int i,j,a,b,num;

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

memset(mp,0,sizeof(mp));

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

scanf("%d: (%d)",&a,&num);

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

scanf("%d",&b);

mp[a][b]=1;

}

}

int cnt=hungary();

printf("%d\n",n-cnt/2); //n-cnt==最大独立集

}

return 0;

}

KM算法:

求带权二分图匹配的最小/大 值。如果要求最小权匹配，则将权值取相反数，再把结果取相反数，那么最小权匹配就求出来了。

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

二分图最佳匹配 （kuhn munkras 算法 O(m\*m\*n)).

邻接矩阵形式 。 返回最佳匹配值，传入二分图大小m,n

邻接矩阵 mat ，表示权，match1,match2返回一个最佳匹配,为匹配顶点的match值为-1，

一定注意m<=n，否则循环无法终止，最小权匹配可将全职取相反数。

初始化： for(i=0;i<N;i++)

for(j=0;j<N;j++) mat[i][j]=-inf;

对于存在的边：mat[i][j]=val;//注意不能负值

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

#include<string.h>

#define N 310

#define inf 1000000000

#define \_clr(x) memset(x,-1,sizeof(int)\*N)

int KM(int m,int n,int mat[][N],int \*match1,int \*match2)

{

int s[N],t[N],l1[N],l2[N];

int p,q,i,j,k,ret=0;

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

{

l1[i]=-inf;

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

l1[i]=mat[i][j]>l1[i]?mat[i][j]:l1[i];

if(l1[i]==-inf) return -1;

}

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

l2[i]=0;

\_clr(match1);

\_clr(match2);

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

{

\_clr(t);

p=0;q=0;

for(s[0]=i;p<=q&&match1[i]<0;p++)

{

for(k=s[p],j=0;j<n&&match1[i]<0;j++)

{

if(l1[k]+l2[j]==mat[k][j]&&t[j]<0)

{

s[++q]=match2[j];

t[j]=k;

if(s[q]<0)

{

for(p=j;p>=0;j=p)

{

match2[j]=k=t[j];

p=match1[k];

match1[k]=j;

}

}

}

}

}

if(match1[i]<0)

{

i--;

p=inf;

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

{

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

{

if(t[j]<0&&l1[s[k]]+l2[j]-mat[s[k]][j]<p)

p=l1[s[k]]+l2[j]-mat[s[k]][j];

}

}

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

l2[j]+=t[j]<0?0:p;

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

l1[s[k]]-=p;

}

}

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

ret+=mat[i][match1[i]];

return ret;

}

KM算法的运行要求是必须存在一个完备匹配，如果求一个最大权匹配(不一定完备)该如何办？依然很简单，把不存在的边权值赋为0。

KM算法求得的最大权匹配是边权值和最大，如果我想要边权之积最大，又怎样转化？还是不难办到，每条边权取自然对数，然后求最大和权匹配，求得的结果a再算出e^a就是最大积匹配。至于精度问题则没有更好的办法了。

补充定义和定理：

最大匹配数：最大匹配的匹配边的数目

最小点覆盖数：选取最少的点，使任意一条边至少有一个端点被选择

最大独立数：选取最多的点，使任意所选两点均不相连

最小路径覆盖数：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以为 0（即单个点）。

定理1：最大匹配数 = 最小点覆盖数（这是 Konig 定理）

定理2：最大匹配数 = 最大独立数

定理3：最小路径覆盖数 = 顶点数 - 最大匹配数

强连通分量：

void tarjan(int u)

{

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

ins[u]=1;

St[top++]=u;

for(int i=head[u];~i;i=e[i].nxt){

int v=e[i].to;

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

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

}

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

int v;

do{

v=St[—top];

beg[v]=num;

ins[v]=0;

cnt[num]++;

}while(u!=v);

num++;

}

}

注释版：

void tarjan(int u)

{

low[u]=dfn[u]=++id;//次序从1开始，初始时将dfn[u]=low[u]都置为次序号

ins[u]=1;

St[top++]=u;

//取u节点的下一节点v,当没有v可取时说明搜索已经到达当前最底部，这时我们函数返回寻找另一条路径。

for(int i=head[u];~i;i=e[i].nxt){

int v=e[i].to;

if(!dfn[v]) tarjan(v),low[u]=min(low[u],low[v]); // 在深度搜索返回时，如果v节点下存在子树，要将u节点的low[u]更新。

else if(ins[v]) low[u]=min(low[u],dfn[v]); // v节点已经被访问，并且在栈中，说明在当前路径上存在环，此处只是赋值，但并不代表在u子树的底下的多个节点没有比当前环更大的环,无法作为深度终止条件。

}

// 搜索完结返回时，判断dfn[u]==low[u]，相等说明找到了一个环，将栈中节点弹出,单个节点也为环。

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

// 将栈中节点弹出，并计数

int v;

do{

v=St[—top];

beg[v]=num;//缩点

ins[v]=0;

cnt[num]++;//当前强连通分量内的点数

}while(u!=v);

num++;//强连通分量的个数

}

}

边双连通分量：

问题：将一个无向图加多少条边可以变为双连通图。

//可处理重边

void tarjan(int u, int pre) {

int v;

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

St[top++] = u;

ins[u] = 1;

int pre\_cnt = 0;

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

v = e[i].to;

if (v == pre && pre\_cnt == 0) {

pre\_cnt++;

continue;

}

if (!dfn[v]) {

tarjan(v, u);

if (low[u] > low[v])low[u] = low[v];

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

bridge++;

e[i].cut = 1;

e[i ^ 1].cut = 1;

}

} else if (ins[v] && low[u] > dfn[v])

low[u] = dfn[v];

}

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

num++;

do {

v = St[--top];

ins[v] = 0;

beg[v] = num;

} while (v != u);

}

}

void init() {

id = top = num = ans = tot = 0;

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

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

memset(ins, 0, sizeof(ins));

memset(du, 0, sizeof(du));

}

int n,m,u,v;

int main() {

while (scanf("%d%d", &n, &m) == 2) {

init();

while (m--) {

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

add(u, v), add(v, u);

}

tarjan(1, 0);

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

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

if (e[j].cut) du[beg[i]]++;

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

if (du[i] == 1)

ans++;

//叶子个数ans, 边双连通图需要加边 (ans+1)/2

printf("%d\n", (ans + 1) / 2);

}

return 0;

}

点双连通分量：

问题：求无向图中桥数和出现在多个环中的边数。

struct P{

int to,nxt;

}e[M<<1];int head[N],tot;

void add(int u,int v){

e[tot].to = v;

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

head[u] = tot++;

}

void cal(){

int sum=0;

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

int u=a[i];

for(int j=head[u];j!=-1;j=e[j].nxt){

int v=e[j].to;

if(vis[v]) sum++;//统计双连通分量内的边数

}

}

sum/=2;

if(sum>cc) ans2+=sum;//若边数大于点数，则为多个环

}

void dfs(int u,int pre){

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

st[++top]=u;

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

int v=e[i].to;

if(v==pre) continue;

if(!dfn[v]){

dfs(v,u);

if(low[u]>low[v]) low[u]=low[v];

if(low[v]>dfn[u]) ans1++;//桥

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

cc=0;

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

int x;

do{

x=st[top--];

a[cc++]=x;//双连通分量中的点存起来

vis[x]=1;//标记

}while(x!=v);

a[cc++]=u;

vis[u]=1;

cal();

}

}

else if(low[u]>dfn[v]) low[u]=dfn[v];

}

}

void init(){

tot=id=top=0;

ans1=ans2=0;

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

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

memset(low,0,sizeof(low));

}

int main()

{

int u,v;

while(scanf("%d%d",&n,&m)==2&&n||m){

init();

while(m--){

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

add(u,v);

add(v,u);

}

for(int i=0;i<n;i++) if(!dfn[i]) dfs(i,-1);

printf("%d %d\n",ans1,ans2);

}

return 0;

}

LCA：

（1）：

* dfs+st，在线

题意：无向树，求LCA。

const int N = 10010;

int rmq[2 \* N];//rmq 数组，就是欧拉序列对应的深度序列

struct ST {

int mm[2 \* N];

int dp[2 \* N][20];//最小值对应的下标

void init(int n) {

mm[0] = -1;

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

mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];

dp[i][0] = i;

}

for (int j = 1; j <= mm[n]; j++)

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

dp[i][j] = rmq[dp[i][j - 1]] < rmq[dp[i + (1 << (j - 1))][j- 1]] ? dp[i][j - 1] : dp[i + (1 << (j - 1))][j - 1];

}

//查询 [a,b] 之间最小值的下标

int query(int a, int b)

{

if (a > b)swap(a, b);

int k = mm[b - a + 1];

return rmq[dp[a][k]] <= rmq[dp[b - (1 << k) + 1][k]] ? dp[a][k] : dp[b - (1 << k) + 1][k];

}

};

//边的结构体定义

struct P {

int to, nxt;

};

P e[N \* 2];

int tot, head[N];

int F[N \* 2];//欧拉序列，就是 dfs 遍历的顺序，长度为 2\*n-1, 下标从 1 开始

int P[N];//P[i] 表示点 i 在 F 中第一次出现的位置

bool ok[N];

int cnt;

ST st;

void init() {

tot = 0;

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

}

//加边，无向边需要加两次

void add(int u, int v) {

e[tot].to = v;

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

head[u] = tot++;

}

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

F[++cnt] = u;

rmq[cnt] = dep;

P[u] = cnt;

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

int v = e[i].to;

if (v == pre)continue;

dfs(v, u, dep + 1);

F[++cnt] = u;

rmq[cnt] = dep;

}

}

//查询 LCA 前的初始化

void LCA\_init(int root, int num) {

cnt = 0;

dfs(root, root, 0);

st.init(2 \* num - 1);

}

//查询 u,v 的 lca 编号

int query\_lca(int u, int v) {

return F[st.query(P[u], P[v])];

}

int T,n,u,v,root;

int main() {

scanf("%d", &T);

while (T--) {

scanf("%d", &n);

init();

memset(ok, 0, sizeof(ok));

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

scanf("%d%d", &u, &v);//u,v间有边

add(u, v);

add(v, u);

ok[v] = 1;

}

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

if (!ok[i]) {

root = i;

break;

}

LCA\_init(root, n);

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

printf("%d\n", query\_lca(u, v));

}

return 0;

}

（2）：

* 离线，LCATarjan
* 复杂度O(n+Q);

题意：给出一颗有向树，Q 个LCA查询，输出每个点作为结果的次数。

int fa[N];//需要初始化为-1

int find(int x)

{

if(fa[x] == -1)return x;

return fa[x] = find(fa[x]);

}

//并查集

void unit(int u,int v)

{

int t1 = find(u);

int t2 = find(v);

if(t1 != t2) fa[t1] = t2;

}

bool vis[N];//访问标记

int anc[N];//祖先

int ans[Q];//每次的查询结果，下标0~q-1

int h[Q],cnt[N];

bool ok[N];

int tt,n,u,v,k,q,root;

struct P

{

int to,nxt;

}e[N\*2];

int head[N],tot;

void add(int u,int v)

{

e[tot].to = v;

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

head[u] = tot++;

}

struct B

{

int q,nxt;

int id;//查询编号

}qry[Q\*2];

void add\_qry(int u,int v,int id)

{

qry[tt].q = v;

qry[tt].nxt = h[u];

qry[tt].id = id;

h[u] = tt++;

qry[tt].q = u;

qry[tt].nxt = h[v];

qry[tt].id = id;

h[v] = tt++;

}

void init()

{

tot = 0;

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

tt = 0;

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

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

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

memset(anc,0,sizeof(anc));

}

void LCA(int u)

{

anc[u] = u;

vis[u] = 1;

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

{

int v = e[i].to;

if(vis[v])continue;

LCA(v);

unit(u,v);

anc[find(u)] = u;

}

for(int i = h[u];i != -1;i = qry[i].nxt)

{

int v = qry[i].q;

if(vis[v])

ans[qry[i].id] = anc[find(v)];

}

}

int main()

{

while(scanf("%d",&n) == 1)

{

init();

memset(ok,false,sizeof(ok));

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

{

scanf("%d:(%d)",&u,&k);

while(k--)

{

scanf("%d",&v);

ok[v] = 1;

add(u,v),add(v,u);

}

}

scanf("%d",&q);

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

{

char e; cin>>e;

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

add\_qry(u,v,i);

}

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

if(!ok[i])

{

root = i;

break;

}

LCA(root);

memset(cnt,0,sizeof(cnt));

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

cnt[ans[i]]++;

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

if(cnt[i] > 0)

printf("%d:%d\n",i,cnt[i]);

}

return 0;

}

（3）：

\* LCA 倍增。

const int N = 10010;

const int D = 20;

struct P

{

int to,nxt;

}e[N\*2];

int head[N],tot;

void add(int u,int v)

{

e[tot].to = v;

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

head[u] = tot++;

}

void init()

{

tot = 0;

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

}

int fa[N][D];//fa[i][j]表示结点i的第2^j个祖先

int d[N];//深度数组

void BFS(int root)

{

queue<int>q;

d[root] = 0;

fa[root][0] = root;

q.push(root);

while(!q.empty())

{

int tmp = q.front();

q.pop();

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

fa[tmp][i] = fa[fa[tmp][i-1]][i-1];

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

{

int v = e[i].to;

if(v == fa[tmp][0])continue;

d[v] = d[tmp] + 1;

fa[v][0] = tmp;

q.push(v);

}

}

}

int LCA(int u,int v)

{

if(d[u] > d[v]) swap(u,v);

int hu = d[u], hv = d[v];

int tu = u, tv = v;

for(int det = hv-hu, i = 0; det ;det>>=1, i++)

if(det&1)

tv = fa[tv][i];

if(tu == tv)return tu;

for(int i = D-1; i >= 0; i--)

{

if(fa[tu][i] == fa[tv][i])

continue;

tu = fa[tu][i];

tv = fa[tv][i];

}

return fa[tu][0];

}

bool ok[N];

int T,n,u,v,root;

int main()

{

scanf("%d",&T);

while(T--)

{

scanf("%d",&n);

init();

memset(ok,0,sizeof(ok));

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

{

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

add(u,v),add(v,u);

ok[v] = true;

}

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

if(!ok[i])

{

root = i;

break;

}

BFS(root);

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

printf("%d\n",LCA(u,v));

}

return 0;

}

求树上两点距离，带边权修改。

#define lowbit(x) (x&(-x))

using namespace std;

const int N = 1e5 + 111;

struct P {

int to, id, w, nxt;

}edge[N \* 2];

int head[N], tol;

void add(int u, int v, int id, int w) {

edge[tol].to = v; edge[tol].w = w; edge[tol].id = id;

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

}

int s, bitn;

int w[N];

vector<P> G[N];

int id[N], es[N \* 2], dep[N \* 2], vs[N \* 2];

int bit[N \* 2], dp[N \* 2][20];

void add(int i, int x) {

while (i <= bitn) {

bit[i] += x;

i += lowbit(i);

}

}

int sum(int i) {

int ret = 0;

while (i > 0) {

ret += bit[i];

i -= lowbit(i);

}

return ret;

}

void dfs(int u, int p, int sum, int &k) {

id[u] = k;

vs[k] = u;

dep[k++] = sum;

for (int i = head[u]; ~i; i = edge[i].nxt) {

P &e = edge[i];

if (e.to != p) {

add(k, e.w);

es[e.id \* 2] = k;

dfs(e.to, u, sum + e.w, k);

vs[k] = u;

dep[k++] = sum;

add(k, -e.w);

es[e.id \* 2 + 1] = k;

}

}

}

int my\_min(int a, int b) {

return dep[a] <= dep[b] ? a : b;

}

void init\_rmq(int n) {

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

for (int j = 1; j < 20; ++j) {

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

dp[i][j] = my\_min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);

}

}

}

void init(int n) {

bitn = n \* 2 - 1;

memset(bit, 0, sizeof bit);

int k = 0;

dfs(0, -1, 0, k);

init\_rmq(2 \* n - 1);

}

int query(int l, int r) {

int k = log2(r - l + 1);

return my\_min(dp[l][k], dp[r - (1 << k) + 1][k]);

}

int lca(int a, int b) {

if (id[a] > id[b]) swap(a, b);

return vs[query(id[a], id[b])];

}

int main()

{

int n, q, u, v;

while (~scanf("%d%d%d", &n, &q, &s)) {

for (int i = 0; i < n; ++i) G[i].clear(), head[i] = -1;

tol = 0;

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

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

--u, --v;

add(u, v, i, w[i]);

add(v, u, i, w[i]);

}

init(n);

int op, a, b;

—s;//起始节点

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

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

--a;

if (op == 0) {

int p = lca(s, a);//a为目标节点

printf("%d\n", sum(id[s]) + sum(id[a]) - 2 \* sum(id[p]));//s,a距离

s = a;

}

else {

scanf("%d", &b);//第a条边权改为b

add(es[2 \* a], b - w[a]);

add(es[2 \* a + 1], w[a] - b);

w[a] = b;

}

}

}

return 0;

}

2-SAT:

const int N = 1 << 12;

const int M = 1 << 13;

struct P {

int fm, to, nxt;

} e[M];

int head[N];

int tot;

void add(int u, int v) {

e[tot].fm = u;

e[tot].to = v;

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

head[u] = tot++;

}

int dfn[N], low[N], id;

int St[N], top;

bool ins[N];

int cmp[N], num;

void tarjan(int u) {

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

ins[u] = true;

St[++top] = u;

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

int v = e[i].to;

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

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

}

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

++num;

while (1) {

int v = St[top--];

cmp[v] = num;

ins[v] = false;

if (v == u) break;

}

}

}

bool sat(int n) // 序号从0开始

{

for (int i = 0; i < 2 \* n; ++i) if (!dfn[i]) tarjan(i);

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

if (cmp[i] == cmp[i ^ 1]) return false;

}

return true;

}

void init() {

memset(dfn, 0, sizeof dfn);

memset(ins, false, sizeof ins);

id = top = num = tot = 0;

memset(head, -1, sizeof head);

}

int n,m;

int key[M];

int a[N], b[N];

bool check(int x){

init();

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

if (a[i] == b[i]) add(key[a[i]] ^ 1, key[a[i]]);

else {

add(key[a[i]] ^ 1, key[b[i]]);//一组中只选一个

add(key[b[i]] ^ 1, key[a[i]]);

}

}

return sat(n);

}

int main() {

int u, v;

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

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

scanf("%d%d", &u, &v);//每组两把钥匙中只能用一把

key[u] = 2 \* i;//编号

key[v] = 2 \* i + 1;

}

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

scanf("%d%d", a + i, b + i);//门上有两个锁(可能相同)，需要至少开一个

}

int l=0,r=m;

int ans=0;

while(l<=r){//二分答案，能开多少个门

int mid=l+r>>1;

if(check(mid)) ans=mid,l=mid+1;

else r=mid-1;

}

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

}

return 0;

}

网络流：

(1)dinic：

const int N = 2010;//点数的最大值

const int M = 1200010;//边数的最大值

const int INF = 0x3f3f3f3f;

struct Edge {

int to, next, cap, flow;

} edge[M];//注意是 M

int tol;

int head[N];

void init() {

tol = 2;

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

}

void add(int u, int v, int w, int rw = 0) {

edge[tol].to = v;

edge[tol].cap = w;

edge[tol].flow = 0;

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

head[u] = tol++;

edge[tol].to = u;

edge[tol].cap = rw;

edge[tol].flow = 0;

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

head[v] = tol++;

}

int Q[N];

int dep[N], cur[N], sta[N];

bool bfs(int s, int t, int n) {

int front = 0, tail = 0;

memset(dep, -1, sizeof(dep[0]) \* (n + 1));

dep[s] = 0;

Q[tail++] = s;

while (front < tail) {

int u = Q[front++];

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

int v = edge[i].to;

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

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

if (v == t)return true;

Q[tail++] = v;

}

}

}

return false;

}

int dinic(int s, int t, int n) {

int maxflow = 0;

while (bfs(s, t, n)) {

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

int u = s, tail = 0;

while (cur[s] != -1) {

if (u == t) {

int tp = INF;

for (int i = tail - 1; i >= 0; i--)

tp = min(tp, edge[sta[i]].cap - edge[sta[i]].flow);

maxflow += tp;

for (int i = tail - 1; i >= 0; i--) {

edge[sta[i]].flow += tp;

edge[sta[i] ^ 1].flow -= tp;

if (edge[sta[i]].cap - edge[sta[i]].flow == 0)

tail = i;

}

u = edge[sta[tail] ^ 1].to;

} else if (cur[u] != -1 && edge[cur[u]].cap > edge[cur[u]].flow && dep[u] + 1 == dep[edge[cur[u]].to]) {

sta[tail++] = cur[u];

u = edge[cur[u]].to;

} else {

while (u != s && cur[u] == -1)

u = edge[sta[--tail] ^ 1].to;

cur[u] = edge[cur[u]].next;

}

}

}

return maxflow;

}

int main()

{

int n,m;

while(scanf("%d%d",&m,&n)==2)

{

init();

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

int x,y,z;

ci(x),ci(y),ci(z);

add(x,y,z);

}

pi(dinic(1,n,n));

}

}

4.19 曼哈顿最小生成树

POJ 3241 求曼哈顿最小生成树上第 k 大的边

const int N = 1e5 + 5;

const int INF = 0x3f3f3f3f;

struct P {

int x, y, id;

} p[N];

bool cmp(P a, P b) {

if (a.x != b.x) return a.x < b.x;

else return a.y < b.y;

}

//树状数组，找 y-x 大于当前的，但是 y+x 最小的

struct BIT {

int min\_val, pos;

void init() {

min\_val = INF;

pos = -1;

}

} bit[N];

//所有有效边

struct Edge {

int u, v, d;

} e[N << 2];

bool cmpedge(Edge a, Edge b) {

return a.d < b.d;

}

int tot;

int n;

int F[N];

int find(int x) {

if (F[x] == -1) return x;

else return F[x] = find(F[x]);

}

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

e[tot].u = u;

e[tot].v = v;

e[tot++].d = d;

}

int lowbit(int x) {

return x & (-x);

}

void update(int i, int val, int pos) {

while (i > 0) {

if (val < bit[i].min\_val) {

bit[i].min\_val = val;

bit[i].pos = pos;

}

i -= lowbit(i);

}

}

//查询 [i,m] 的最小值位置

int ask(int i, int m) {

int min\_val = INF, pos = -1;

while (i <= m) {

if (bit[i].min\_val < min\_val) {

min\_val = bit[i].min\_val;

pos = bit[i].pos;

}

i += lowbit(i);

}

return pos;

}

int dist(P a, P b) {

return abs(a.x - b.x) + abs(a.y - b.y);

}

void Manhattan\_minimum\_spanning\_tree(int n, P p[]) {

int a[N], b[N];

tot = 0;

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

// 种坐标变换

if (dir == 1 || dir == 3) {

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

swap(p[i].x, p[i].y);

} else if (dir == 2) {

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

p[i].x = -p[i].x;

}

sort(p, p + n, cmp);

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

a[i] = b[i] = p[i].y - p[i].x;

sort(b, b + n);

int m = unique(b, b + n) - b;

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

bit[i].init();

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

int pos = lower\_bound(b, b + m, a[i]) - b + 1;

int ans = ask(pos, m);

if (ans != -1)

addedge(p[i].id, p[ans].id, dist(p[i], p[ans]));

update(pos, p[i].x + p[i].y, i);

}

}

}

int solve(int k) {

Manhattan\_minimum\_spanning\_tree(n, p);

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

sort(e, e + tot, cmpedge);

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

int u = e[i].u;

int v = e[i].v;

int t1 = find(u), t2 = find(v);

if (t1 != t2) {

F[t1] = t2;

k--;

if (k == 0)return e[i].d;

}

}

}

int main() {

int k;

while (scanf("%d%d", &n, &k) == 2 && n) {

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

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

p[i].id = i;

}

printf("%d\n", solve(n - k));

}

return 0;

}