2019 CCPC哈尔滨赛站



目录

[Kruskal 求最小生成树 3](#_Toc21618111)

[次小生成树 4](#_Toc21618112)

[严格次小生成树 7](#_Toc21618113)

[Havel-Hakimi定理 10](#_Toc21618114)

[ST表 12](#_Toc21618115)

[LCA 13](#_Toc21618116)

[tarjan求割点 15](#_Toc21618117)

[tarjan求割边(除函数部分外没变化) 16](#_Toc21618118)

[双联通分量 17](#_Toc21618119)

[tarjan求SCC 18](#_Toc21618120)

[有向图缩点 20](#_Toc21618121)

[SPFA求最短路/判负环 21](#_Toc21618122)

[Dijkstra算法求最短路 23](#_Toc21618123)

[归并排序 24](#_Toc21618124)

[拓扑排序 25](#_Toc21618125)

[树的哈希 27](#_Toc21618126)

[树的直径 30](#_Toc21618127)

[差分约束 31](#_Toc21618128)

[一般图最大匹配 34](#_Toc21618129)

[动态树 37](#_Toc21618130)

[分块 41](#_Toc21618131)

[树链剖分 43](#_Toc21618132)

[约瑟夫问题 49](#_Toc21618133)

[完美消除序列&&弦图最小染色 50](#_Toc21618134)

[欧拉回路 52](#_Toc21618135)

[哈密顿回路 53](#_Toc21618136)

[树论 56](#_Toc21618137)

[选出尽可能多的路径且路径间不能有重叠 56](#_Toc21618138)

[树上点分治 59](#_Toc21618139)

[求一条简单路径，权值和等于 K，且边的数量最小 59](#_Toc21618140)

[Q个询问，询问长度为k的路径是否存在 62](#_Toc21618141)

[求存在多少条长度为3的倍数的路径 65](#_Toc21618142)

[快速幂 & 费马小定理 68](#_Toc21618143)

# Kruskal 求最小生成树

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

**using** **namespace** std;

**const** **int** sz = **200010**;

**int** f[sz];//并查集

**int** n,m;

**int** **find**(**int** x)

{

**if**(f[x] == x)

**return** x;

**return** f[x] = find(f[x]);

}

**struct** gtnd

{

**int** f,t,d;

}l[sz];//图的边存到 l[] 中

**bool** **cmp**(gtnd swc,gtnd dc)

{

**return** swc.d > dc.d;

}

**int** **Kruskal**()

{

**int** ans = **0**;// ans 为最小生成树权值的总和

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

f[i] = i;

sort(l+**1**,l+m+**1**,cmp);

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

{

**int** u = l[i].f , v = l[i].t;

**int** fu = find(u) , fv = find(v);

**if**(fu != fv)

{

f[fu] = fv;

ans += l[i].d;

}

}

**return** ans;

}

# 次小生成树

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int sz = 201000;

int hed[sz],nxt[sz],dist[sz][32];

int par[sz][32],deep[sz];

int tot = 1;

struct edge

{

int f,t,d;

bool pick;

}l[sz],g[sz];

bool cmp(edge a,edge b)

{

return a.d < b.d;

}

void build(int f,int t,int d)

{

l[tot].t = t;

l[tot].d = d;

nxt[tot] = hed[f];

hed[f] = tot ++;

}

int n;

void dfs(int u,int fa,int dep,int dis)

{

par[u][0] = fa;

dist[u][0] = dis;

deep[u] = dep;

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i].t;

if(v != fa)

dfs(v,u,dep+1,l[i].d);

}

}

int lca(int u,int v)

{

int ans = 0;

if(deep[u] < deep[v])

swap(u,v);

for(int i = 31 ; i >= 0 ; i --)

if(deep[par[u][i]] >= deep[v])

ans = max(dist[u][i],ans) , u = par[u][i];

for(int i = 31 ; i >= 0 ; i --)

if(par[u][i] != par[v][i])

ans = max(dist[u][i],max(dist[v][i],ans)) , u = par[u][i] , v = par[v][i];

if(u != v)

ans = max(dist[u][0],max(dist[v][0],ans)) , u = par[u][0] , v = par[v][0];

return ans;

}

int f[sz];

int find(int x)

{

if(f[x] == x)

return x;

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

}

void init()

{

memset(l,0,sizeof(l));

memset(hed,0,sizeof(hed));

memset(nxt,0,sizeof(nxt));

tot = 1;

}

int main()

{

// int T;

int n,m;

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

// int K = 1;

// while(T --)

{

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

init();

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

f[i] = i;

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

{

scanf("%d%d%d",&g[i].f,&g[i].t,&g[i].d);

g[i].pick = 0;

}

// printf("Case #%d : ",K ++);

sort(g+1,g+m+1,cmp);

int mst = 0 , mmst = 2147483641;

int cnt = 0;

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

{

int ff = find(g[i].f);

int tt = find(g[i].t);

if(ff != tt)

{

build(g[i].f,g[i].t,g[i].d);

build(g[i].t,g[i].f,g[i].d);

g[i].pick = 1;

f[tt] = ff;

mst += g[i].d;

cnt ++;

}

}

dfs(1,0,1,0);

for(int i = 1 ; i <= 31 ; i ++)

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

par[j][i] = par[par[j][i-1]][i-1],

dist[j][i] = max(dist[j][i-1],dist[par[j][i-1]][i-1]);//倍增思想

bool is\_unique = 1;

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

{

if(g[i].pick)

continue;

int dis = lca(g[i].f,g[i].t);

// printf("#%d %d\n",dis,g[i].d);

if(dis == g[i].d)

continue;

mmst = min(mmst,mst-dis+g[i].d);

}

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

}

return 0;

}

# 严格次小生成树

#include<cctype>

#include<cstdio>

#include<cstring>

#include<algorithm>

#define add(x,y,z) (e[++p]=(edg){x,y,z})

#define build(x,y,z) (ed[++k]=(edge){h[x],y,z},h[x]=k)

#define N 400005

#define M 400005

#define ll long long

using namespace std;

struct edg

{

int x,y,z;

}e[M];

struct edge

{

int next,to,dis;

}ed[N<<1];

int n,m,k,p,h[N],dep[N],ffa[N],fa[N][20],mx1[N][20],mx2[N][20];

bool f[M];

ll ans,ret=0x7fffffff;

char readc()

{

static char buf[100000],\*l=buf,\*r=buf;

if (l==r)

r=(l=buf)+fread(buf,1,100000,stdin);

if (l==r)

return EOF; return \*l++;

}

int read()

{

int x = 0;

char ch = readc();

while(!isdigit(ch))

ch=readc();

while(isdigit(ch))

x=(x<<3)+(x<<1)+(ch^48),ch=readc();

return x;

}

int findfa(int x)

{

if(ffa[x] == x)

return x;

return ffa[x]=findfa(ffa[x]);

}

bool cmp(edg a,edg b)

{

return a.z<b.z;

}

void dfs(int x)

{

for(int i = h[x]; i ; i = ed[i].next)

{

int v = ed[i].to;

if(v != fa[x][0])

{

fa[v][0] = x , mx1[v][0] = ed[i].dis;

dep[v] = dep[x] + 1 , dfs(v);

}

}

}

void Make()

{

for(int j = 1 ; j <= 16 ; j ++)

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

{

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

mx1[i][j] = max(mx1[i][j-1] , mx1[fa[i][j-1]][j-1]);

if(mx1[i][j-1] == mx1[fa[i][j-1]][j-1])

mx2[i][j] = max(mx2[i][j-1],mx2[fa[i][j-1]][j-1]);

else

{

mx2[i][j] = min(mx1[i][j-1],mx1[fa[i][j-1]][j-1]);

mx2[i][j] = max(mx2[i][j],mx2[fa[i][j-1]][j-1]);

mx2[i][j] = max(mx2[i][j],mx2[i][j-1]);

}

}

}

void calc(int &x,int &m1,int &m2,int j)

{

if(mx1[x][j]>m1)

m2 = m1 , m1 = mx1[x][j];

else if(mx1[x][j] < m1)

m2 = max(m2,mx1[x][j]);

m2 = max(m2,mx2[x][j]) , x = fa[x][j];

}

void LCA(int x,int y,int z)

{

int m1 = 0 , m2 = 0;

if(dep[x] < dep[y])

swap(x,y);

for(int j = 16 ; ~j ; j --)

if(dep[fa[x][j]] >= dep[y])

calc(x,m1,m2,j);

if(x == y)

{

ret = min(ret,1ll\*(z==m1?z-m2:z-m1));

return;

}

for(int j = 16 ; ~j ; j --)

if(fa[x][j] != fa[y][j])

calc(x,m1,m2,j) , calc(y,m1,m2,j);

calc(x,m1,m2,0) , calc(y,m1,m2,0);

ret = min(ret,1ll\*(z==m1?z-m2:z-m1));

}

int main()

{

n = read(), m = read();

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

ffa[i] = i;

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

{

int f,t,d;

f = read() , t = read() , d = read();

add(f,t,d);

}

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

int cnt = 0;

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

{

int x = e[i].x , y = e[i].y , z = e[i].z;

int fx = findfa(x) , fy = findfa(y);

if(fx != fy)

{

ffa[fx] = fy;

ans += z;

cnt ++;

build(x,y,z) , build(y,x,z);

f[i] = true;

if(cnt == n-1)

break;

}

}

dep[1] = 1;

dfs(1);

Make();

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

if(!f[i])

LCA(e[i].x,e[i].y,e[i].z);

printf("%lld\n",ret+ans);

return 0;

}

# Havel-Hakimi定理

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int sz = 200010;

int n;

struct node

{

int x,id;

}num[sz];

bool cmp(node a,node b)

{

return a.x >= b.x;

}

int mp[21][21];

//给定 n 个点的度，求复合条件的无向图

int main()

{

int T;

scanf("%d",&T);

while(T --)

{

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

scanf("%d",&n);

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

{

scanf("%d",&num[i].x);//n个点的度

num[i].id = i;

}

int k = 1;

while(k <= n)

{

sort(num+k,num+n+1,cmp);

if(num[k].x > n - k)

break;

int pos = 0;

for(int i = 1 ; i <= num[k].x ; i ++)

{

if(num[k+i].x <= 0)

break;

pos = i;

num[k+i].x --;

mp[num[k].id][num[k+i].id] = 1 , mp[num[k+i].id][num[k].id] = 1;

}

if(pos < num[k].x)

break;

k ++;

}

if(k <= n)

puts("NO");

else

{

puts("YES");

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

{

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

{

printf("%d",mp[i][j]);

}

puts("");

}

}

}

return 0;

}

# ST表

#include<bits/stdc++.h>

using namespace std;

const int MAX = 100000;

int a[500];

int stTable[2][MAX][32];

int preLog2[MAX];

void st\_prepare(int n,int \*array)

{

preLog2[1] = 0;

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

{

preLog2[i] = preLog2[i-1];

if((1 << preLog2[i] + 1) == i)

{

++ preLog2[i];

}

}

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

{

stTable[0][i][0] = array[i];

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

{

stTable[0][i][j] = min(stTable[0][i][j-1],stTable[0][i+(1 << j-1)][j-1]);

}

}

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

{

stTable[1][i][0] = array[i];

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

{

stTable[1][i][j] = max(stTable[1][i][j-1],stTable[1][i+(1 << j-1)][j-1]);

}

}

}

int query\_min(int l,int r)

{

int len = r - l + 1 , k = preLog2[len];

return min(stTable[0][l][k],stTable[0][r-(1<<k)+1][k]);

}

int query\_max(int l,int r)

{

int len = r - l + 1 , k = preLog2[len];

return max(stTable[1][l][k],stTable[1][r-(1<<k)+1][k]);

}

int main()

{

int n, l, r;

cin >> n;

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

cin >> a[i];

st\_prepare(n, a);

while(true)

{

cin >> l >> r;

cout << query\_min(l - 1, r - 1) << " " << query\_max(l - 1, r - 1) << endl;

}

return 0;

}

# LCA

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

**using** **namespace** std;

**const** **int** size = **200010**;

**int** head[size],next[size],dist[size][**32**];//dist[i][j]记录了第 i 个节点向上走 2^j 步走的边的权值和

**int** par[size][**32**],deep[size];//par[i][j]记录了第 i 个节点向上走 2^j 步可以到达的节点，deep记录了深度

**int** tot = **1**;//边的数量

**struct** dc

{

**int** t,d;

}l[size];

**void** **build**(**int** f,**int** t,**int** d) // 建一条 f点到 t 点权值为 d 的有向边

{

l[tot].t = t;

l[tot].d = d;

next[tot] = head[f];

head[f] = tot ++;

}

**int** n;

**void** **dfs**(**int** u,**int** fa,**int** dep,**int** dis)//预处理深度、每个节点的父亲节点、边权

{

par[u][**0**] = fa;

dist[u][**0**] = dis;

deep[u] = dep;

**for**(**int** i = head[u] ; i ; i = next[i])

{

**int** v = l[i].t;

**if**(v != fa)

dfs(v,u,dep+**1**,l[i].d);

}

}

**int** **lca**(**int** u,**int** v)//求树上 u v 两点的距离

{

**int** ans = **0**;

**if**(deep[u] < deep[v])

swap(u,v);

**for**(**int** i = **31** ; i >= **0** ; i --)

**if**(deep[par[u][i]] >= deep[v])

ans += dist[u][i] , u = par[u][i];

**for**(**int** i = **31** ; i >= **0** ; i --)

**if**(par[u][i] != par[v][i])

ans += dist[u][i] + dist[v][i] , u = par[u][i] , v = par[v][i];

**if**(u != v)

ans += dist[u][**0**] + dist[v][**0**] , u = par[u][**0**] , v = par[v][**0**];

**return** ans; // 此时,若 u v 两点联通(即树不是森林) u == v ，且此时的 u v 为询问时 u v 的最近公共祖先

}

**int** **main**()

{

scanf("%d",&n);

**for**(**int** i = **1** ; i < n ; i ++)//点编号从 1 开始

{

**int** f,t,d;

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

build(f,t,d);

build(t,f,d);//建边

}

dfs(**1**,**0**,**1**,**0**);//预处理

**for**(**int** i = **1** ; i <= **31** ; i ++)

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

par[j][i] = par[par[j][i-**1**]][i-**1**] , dist[j][i] = dist[j][i-**1**] + dist[par[j][i-**1**]][i-**1**];//倍增思想

**int** m;

scanf("%d",&m);

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

{

**int** u,v;

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

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

}

**return** **0**;

}

# tarjan求割点

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stack>

#include<algorithm>

**using** **namespace** std;

**const** **int** sz = **200010**;

**int** head[sz],nxt[sz],l[sz];

**int** tot = **1**;

**int** n,m;

**void** **build**(**int** f,**int** t)

{

l[tot] = t;

nxt[tot] = head[f];

head[f] = tot ++;

}

**int** low[sz],dfn[sz],dfs\_clock;

**bool** is\_cut[sz];//第 i 点是否为割点

**int** **tarjan**(**int** u,**int** fa)//当前节点 u 和他的上一个节点 fa

{

dfn[u] = low[u] = ++ dfs\_clock;

**int** child = **0**;

**for**(**int** i = head[u] ; i ; i = nxt[i])

{

**int** v = l[i];

**if**(!dfn[v])

{

child ++;

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

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

**if**(dfn[u] <= low[v])

is\_cut[u] = **1**;//

}

**else** **if**(dfn[v] < dfn[u] && v != fa)

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

}

**if**(child == **1** && fa == **0**)

is\_cut[u] = **0**;

**return** low[u];

}

**int** **main**()

{

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

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

{

**int** f,t;

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

build(f,t);

build(t,f);

}

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

**if**(!dfn[i])

tarjan(i,**0**);//防止图不连通

**return** **0**;

}

# tarjan求割边(除函数部分外没变化)

**int** low[sz],dfn[sz],dfs\_clock;

**struct** gtnd

{

**int** f,t;

}cut[sz];//cut[] 存割边连接的两个点

**int** conut;

**int** **tarjan**(**int** u,**int** fa)

{

dfn[u] = low[u] = ++ dfs\_clock;

**int** child = **0**;

**for**(**int** i = head[u] ; i ; i = nxt[i])

{

**int** v = l[i];

**if**(!dfn[v])

{

child ++;

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

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

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

cut[++conut].f = u , cut[count].t = v;

}

**else** **if**(dfn[v] < dfn[u] && v != fa)

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

}

**return** low[u];

}

# 双联通分量

**int** root, cnt;

**int** vis[maxn], dfn[maxn], low[maxn];

**bool** cut[maxn];

//vector<pair<int, int>>bridge;

**void** **dfs**(**int** u, **int** fa)

{

**int** son=**0**;

vis[u]=**1**;

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

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

{

**int** v=G[u][i];

**if** (v==fa) **continue**;

**if** (vis[v]==**1**) low[u]=min(low[u], dfn[v]); //返祖边

**if** (vis[v]==**0**)

{

dfs(v, u);

son++;

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

**if** ( (u==root && son>**1**) || (u!=root && low[v]>=dfn[u]))

{

cut[u]=true;

//if(low[v] > dfn[u]) bridge.push\_back({u, v}); //(u, v) 是桥

}

}

}

vis[u]=**2**;

}

**void** **tarjan\_init**()

{

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

memset(cut, **0**, **sizeof**(cut));

cnt=**0**; root=**1**;

//bridge.clear();

}

# tarjan求SCC

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stack>

#include<algorithm>

using namespace std;

const int sz = 200010;

int head[sz],nxt[sz],l[sz];

int low[sz],dfn[sz],scc\_num,dfs\_clock;

bool in\_stack[sz];

stack < int > s;

struct gtnd

{

int p,num;

}scc[sz];

int tarjan(int u)

{

dfn[u] = low[u] = ++ dfs\_clock ;

s.push(u);

in\_stack[u] = 1;

for(int i = head[u] ; i ; i = nxt[i])

{

int v = l[i];

if(!dfn[v])

{

low[v] = tarjan(v);

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

}

else if(!scc[v].num)

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

}

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

{

scc\_num ++;

int v;

do

{

v = s.top();

s.pop();

in\_stack[v] = 0;

scc[v].num = scc\_num;//记录当前点属于第几个scc

scc[v].p = v;//记录对应的点

}while(u != v)

}

return low[u];

}

# 有向图缩点

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

if(!dfn[i])

tarjan(i);

// printf("sccnum %d\n",scc\_num);

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

{

for(int j = hed[i] ; j ; j = nxt[j])

{

int f = i , t = l[j];

if(scc[f].num != scc[t].num)

{

edge nxt;

nxt.f = scc[f].num;

nxt.t = scc[t].num;

g.push\_back(nxt);

}

}

}

memset(hed,0,sizeof(hed));

tot = 1;

sort(g.begin(),g.end());

g.erase(unique(g.begin(),g.end()),g.end());

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

{

build(g[i].f,g[i].t);

}

# SPFA求最短路/判负环

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<vector>

#include<queue>

using namespace std;

const int sz = 20001;

queue < int > q;

int n,m;

int dist[sz];

struct edge

{

int t,d;

};

vector < edge > l[sz];

void build(int f,int t,int d)

{

if(d >= 0)

l[f].push\_back({t,d}) , l[t].push\_back({f,d});

else

l[f].push\_back({t,d});

}

bool use[sz];

int tim[sz];

bool spfa(int s)

{

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

dist[i] = 2147483641;

dist[s] = 0;

use[s] = 1;

q.push(s);

while(!q.empty())

{

int f = q.front();

q.pop();

use[f] = 0;

for(int i = 0 ; i < l[f].size() ; i ++)

{

int t = l[f][i].t;

int d = l[f][i].d;

if(dist[t] > dist[f] + d)

{

dist[t] = dist[f] + d;

tim[t] = tim[f] + 1;

if(tim[t] > n)

return false;

if(!use[t])

{

use[t] = 1;

q.push(t);

}

}

}

}

return true;

}

int main()

{

int T;

scanf("%d",&T);

while(T --)

{

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

while(!q.empty())

q.pop();

for(int i = 1 ; i <= n + 5 ; i ++)

l[i].clear() , use[i] = 0 , tim[i] = 0;

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

{

int f,t,d;

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

build(f,t,d);

}

bool ans = spfa(1);

if(!ans)

puts("YE5");

else

puts("N0");

}

return 0;

}

# Dijkstra算法求最短路

int dij(int s,int e)

{

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

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

dist[i] = 2147483641;

dist[s] = 0;

node st;

st.p = s;

st.d = 0;

q.push({s,0});

while(!q.empty())

{

node f = q.top();

q.pop();

if(use[f.p])

continue;

use[f.p] = 1;

for(int i = head[f.p] ; i ; i = nxt[i])

{

int t = l[i].t;

if(use[t])

continue;

if(dist[t] > dist[f.p] + l[i].d)

{

dist[t] = dist[f.p] + l[i].d;

q.push({t,dist[t]});

}

}

}

if(dist[e] == 2147483641)

return -1;

return dist[e];

}

# 归并排序

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#define ll long long

**using** **namespace** std;

**const** **int** sz = **2000100**;

ll ans;

**int** n;

**int** num[sz];

**int** temp[sz];

**void** **merge\_sort**(**int** l,**int** r)//将区间 l - r 排序

{

**if**(l == r)

**return** ;

**int** mid = l + r >> **1**;

merge\_sort(l,mid) , merge\_sort(mid+**1**,r);

**int** p = l , pl = l , pr = mid + **1**;

**while**(pl <= mid || pr <= r)

{

**if**(pr > r || (pl <= mid && num[pl] <= num[pr]))

temp[p ++] = num[pl ++];

**else**

temp[p ++] = num[pr ++] , ans += mid - pl + **1**;

}

**for**(**int** i = l ; i <= r ; i ++)

num[i] = temp[i];

**return** ;

}

**int** **main**()

{

**int** n;

scanf("%d",&n);

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

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

merge\_sort(**1**,n);

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

printf("%d ",num[i]);

**return** **0**;

}

# 拓扑排序

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<queue>

**using** **namespace** std;

**const** **int** sz = **200010**;

**int** head[sz],nxt[sz],l[sz];

**int** ru[sz];//点 i 的入度

**int** tot = **1**;

**int** n,m;

queue < **int** > q;

**void** **build**(**int** f,**int** t)

{

l[tot] = t;

nxt[tot] = head[f];

head[f] = tot ++;

}

**void** **top\_sort**()

{

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

**if**(!ru[i])

{

q.push(i);//每次查看入队元素是否只有1个来确定拓扑序是否唯一

printf("%d ",i);

}

**while**(!q.empty())

{

**int** f = q.front();

q.pop();

**for**(**int** i = head[f] ; i ; i = nxt[i])

{

ru[l[i]] --;

**if**(!ru[l[i]])

{

printf("%d ",l[i]);

q.push(l[i]);

}

}

}

}

**int** **main**()

{

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

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

{

**int** f,t;

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

ru[t] ++;

build(f,t);

}

top\_sort();

**return** **0**;

# 树的哈希

#include<iostream>

#include<cstring>

#include<cstdio>

#include<algorithm>

#include<queue>

#include<ctime>

#include<cmath>

#include<set>

#include<map>

#define ull unsigned long long

#define ll long long

#define INF 10000

using namespace std;

int n,m;

const int N = 2010;

int max\_g,g[17];

vector <int> edge[N];

int size[N];

ull Hash[N],RD[N];

void build(int f,int t)

{

edge[f].push\_back(t);

}

void dfs(int u,int fa)

{

size[u] = 1;

int tmp = 0;

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

{

int v = edge[u][i];

if(v == fa) continue;

dfs(v,u);

tmp = max(tmp,size[v]+1);

size[u] += size[v];

}

tmp = max(tmp,n-size[u]+1);

if(tmp == max\_g){

g[++g[0]] = u;

}

else if(tmp < max\_g)

{

g[0] = 1;

g[g[0]] = u;

max\_g = tmp;

}

return ;

}

void dfs\_hash(int u,int fa)

{

Hash[u] = 1;

size[u] = 1;

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

{

int v = edge[u][i];

if(v == fa) continue;

dfs\_hash(v,u);

Hash[u] += Hash[v] ^ RD[size[v]];

size[u] += size[v];

}

Hash[u] \*= RD[n-size[u]+1];

}

map < ull,int > MP;

int main()

{

srand(time(0));

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

RD[i] = (ull)rand()\*rand();

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

{

MP.clear();

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

{

scanf("%d",&n);

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

edge[j].clear();

int root = 0;//注意要任意找一个点作为根节点

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

{

int x;

scanf("%d",&x);

if(!x)

{

root = j;

continue;

}

build(x,j);

build(j,x);

}

max\_g = INF;

g[0] = 0;

dfs(root,0);

ull hs = 1;

for(int j = 1 ; j <= g[0] ; j ++)

{

dfs\_hash(g[j],0);

hs \*= Hash[g[j]];

}

if(!MP[hs])

MP[hs] = i;

printf("%d\n",MP[hs]);//hs为这颗树的hash值

}

}

return 0;

}

# 树的直径

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

**using** **namespace** std;

**int** n;

**const** **int** size = **200010**;

**int** head[size],next[size],l[size];

**int** tot = **1**;

**void** **build**(**int** f,**int** t)

{

l[tot] = t;

next[tot] = head[f];

head[f] = tot ++;

}

**int** pos,dist;

**void** **dfs**(**int** u,**int** p,**int** dis)

{

**if**(dis > dist)

dist = dis , pos = u;

**for**(**int** i = head[u] ; i ; i = next[i])

{

**int** v = l[i];

**if**(v != p)

dfs(v,u,dis+**1**);

}

}

**int** **get\_length**()

{

pos = **0** , dist = **0**;

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

dist = **0**;

dfs(pos,-**1**,**0**);

**return** dist;

}

**int** **main**()

{

scanf("%d",&n);

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

{

**int** ll,rr;

scanf("%d%d",&ll,&rr);

**if**(ll)

build(ll,i) , build(i,ll);

**if**(rr)

build(rr,i) , build(i,rr);

}

**int** ans = get\_length();

**return** **0**;

}

# 差分约束

将每个式子转化为Xi<=Xj+W(i,j),那么就建一条边，Xj -> Xi = W(i,j),然后利用最短路径解决问题，在x0定死的情况下，求得最小值

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<queue>

using namespace std;

const int size = 2000010;

int head[size],nxt[size],dist[size];

struct dc

{

int t,d;

}l[size];

int tot = 1;

void build(int f,int t,int d)

{

l[tot].t = t;

l[tot].d = d;

nxt[tot] = head[f];

head[f] = tot ++;

}

int n,m;

queue < int > q;

int ti[size];

bool use[size];

bool spfa()

{

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

dist[i] = 214748364;

q.push(0);

use[0] = 1;

dist[0] = 0;

while(!q.empty())

{

int f = q.front();

q.pop();

use[f] = 0;

for(int i = head[f] ; i ; i = nxt[i])

{

int t = l[i].t;

if(dist[t] > dist[f] + l[i].d)

{

dist[t] = dist[f] + l[i].d;

ti[t] = ti[f] + 1;

if(ti[t] > n)

return false;

if(!use[t])

{

use[t] = 1;

q.push(t);

}

}

}

}

return true;

}

int main()

{

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

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

{

int f,t,d;

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

build(t,f,d);

}

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

build(0,i,0);

//建图部分根据题目来

int minn = 2147483641;

bool flag = spfa();

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

minn = min(minn,dist[i]);

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

if(dist[i] == 214748364)

dist[i] = 0;

//求任意解

if(minn < 0)

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

dist[i] += -minn;

else

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

dist[i] -= minn;

if(flag)

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

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

else

puts("NO SOLUTION");

return 0;

}

/\*

5 8

1 2 0

1 5 -1

2 5 1

3 1 5

4 1 4

4 3 -1

5 3 -1

5 4 -3

\*/

# 一般图最大匹配

#include<iostream>

#include<cstdio>

#include<cstring>

#include<vector>

#include<queue>

using namespace std;

const int sz = 505;

int n,m,cnt,Num[105][7],fa[sz],pre[sz];

int Mark[sz],mat[sz],vis[sz];

queue < int > q;

const int M = 300010;

int hed[M],nxt[M],l[M];

int tot = 1;

void build(int f,int t)

{

l[tot] = t;

nxt[tot] = hed[f];

hed[f] = tot ++;

}

int find(int x)

{

if(fa[x] == x)

return x;

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

}

int getint()

{

int x = 0;

char ch = getchar();

while(ch < '0' || '9' < ch)

ch = getchar();

while('0' <= ch && ch <= '9')

{

x = x \* 10 + ch - '0';

ch = getchar();

}

return x;

}

void Build()

{

n = getint();

m = getint();

while(m --)

{

int x = getint() , y = getint();

build(x,y);

build(y,x);

}

}

void Augment(int p)

{

while(p != -1)

{

int tmp = mat[pre[p]];

mat[p] = pre[p];

mat[pre[p]] = p;

p = tmp;

}

}

void Rebuild(int x,int y,int lca)

{

while(find(x) != lca)

{

pre[x] = y;

if(fa[x] == x)

fa[x] = lca;

if(fa[mat[x]] == mat[x])

fa[mat[x]] = lca;

if(Mark[mat[x]] == 1)

{

q.push(mat[x]);

Mark[mat[x]] = 0;

}

y = mat[x];

x = pre[y];

}

}

void Link(int x,int y)

{

cnt ++;

int p = find(x);

for(;;)

{

vis[p] = cnt;

p = mat[p];

if(p == -1)

break;

p = find(pre[p]);

}

int lca;

p = find(y);

for(;; p = find(pre[mat[p]]))

if(vis[p] == cnt)

{

lca = p;

break;

}

Rebuild(x,y,lca);

Rebuild(y,x,lca);

}

int BFS(int s)

{

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

fa[i] = i , Mark[i] = pre[i] = -1;

while(!q.empty())

q.pop();

Mark[s] = 0;

q.push(s);

while(!q.empty())

{

int f = q.front();

q.pop();

for(int i = hed[f] ; i ; i = nxt[i])

{

int t = l[i];

if(find(t) == find(f))

continue;

if(Mark[t] == -1)

{

Mark[t] = 1;

pre[t] = f;

if(mat[t] == -1)

{

Augment(t);

return 1;

}

Mark[mat[t]] = 0;

q.push(mat[t]);

}

else if(Mark[t] == 0)

Link(t,f);

}

}

return 0;

}

int main()

{

int ans = 0;

Build();

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

mat[i] = -1;

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

if(mat[i] == -1)

ans += BFS(i);

cout << ans << endl;

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

printf("%d%c",mat[i] == -1 ? 0 : mat[i],i == n ? '\n' : ' ');

return 0;

}

# 动态树

#include <stdio.h>

#include <cstring>

#include <cmath>

#include <set>

#include <map>

#include <stack>

#include <queue>

#include <bitset>

#include <utility>

#include <sstream>

#include <iostream>

#include <algorithm>

//#include <unordered\_map>

#define rep(i,a,n) for(int i=a;i<n;++i)

#define per(i,a,n) for(int i=(n)-1;i>=a;--i)

#define foredge(u,i) for(int i=head[u];~i;i=l[i].nxt)

#define mem(a,x) memset(a,x,sizeof(a))

#define LOG(a) cout<<'#'<<a<<endl

void print\_array(int \*a,int n){printf("#");for(int i=0;i<n;i++) printf("%d%c",a[i],i==n-1?'\n':' ');}

typedef long long ll;

using namespace std;

typedef vector<int> vec;

typedef vector<vec> mat;

typedef pair<int,int> pii;

const double EPS=1e-8;

const ll INF=0x3f3f3f3f3f3f3f3f;

const int IINF=0x3f3f3f3f;

inline ll readll(){

ll x(0),op(1);char ch=getchar();

while(ch<'0'||ch>'9'){if(ch=='-')op=-1;ch=getchar();}

while(ch>='0'&&ch<='9')x=x\*10+ch-'0',ch=getchar();

return x\*op;

}

const int sz=300009;

int v[sz];//点权，修改点权时需要把这个点splay到根节点再修改，否则会影响正确性

int maxx[sz];//子树维护的信息:异或和

int add[sz];

int xr[sz];

int n;

struct LCT{

#define ls c[x][0]

#define rs c[x][1]

int f[sz],c[sz][2];

int size[sz];

bool rev[sz];//反转标记

void init(int n){

rep(i,0,n+1) f[i]=c[i][0]=c[i][1]=rev[i]=0,size[i] = 1;

}

bool son(int x){return c[f[x]][1]==x;}//0左儿子，1右儿子

bool nroot(int x){return c[f[x]][0]==x||c[f[x]][1]==x;}//判断节点是否为一个Splay的根

void reverse(int x){swap(ls,rs);rev[x]^=1;}//翻转操作

void pushup(int x){//上传信息

size[x] = 1;

if(ls) size[x] += size[ls];

if(rs) size[x] += size[rs];

xr[x]=xr[ls]^xr[rs]^v[x];

}

void pushdown(int x){//判断并释放懒标记

if(rev[x]){if(ls)reverse(ls);if(rs)reverse(rs);rev[x]=0;}

if(add[x]){

if(ls)v[ls]+=add[x],maxx[ls]+=add[x],add[ls]+=add[x];

if(rs)v[rs]+=add[x],maxx[rs]+=add[x],add[rs]+=add[x];

add[x]=0;

}

}

void rotate(int x){//一次旋转

int y=f[x],z=f[y],k=son(x),w=c[x][!k];

if(nroot(y)) c[z][son(y)]=x; //注意if(nroot(y))语句，此处不判断会引起致命错误

c[x][!k]=y;c[y][k]=w; if(w)f[w]=y;

f[y]=x;f[x]=z;pushup(y);

}

void splay(int x){//只传了一个参数，因为所有操作的目标都是该Splay的根（与普通Splay的区别3）

static int st[sz];

int y=x,z=0;

st[++z]=y;//st为栈，暂存当前点到根的整条路径，pushdown时一定要从上往下放标记（与普通Splay的区别4）

while(nroot(y)) st[++z]=y=f[y];

while(z) pushdown(st[z--]);

while(nroot(x)){

y=f[x];z=f[y];

if(nroot(y)) rotate((c[y][0]==x)^(c[z][0]==y)?x:y);

rotate(x);

}

pushup(x);

}

void access(int x){for(int y=0;x;x=f[y=x]) splay(x),rs=y,pushup(x);}//访问

void makeroot(int x){access(x);splay(x);reverse(x);}//换根

int findroot(int x){access(x);splay(x);while(ls)pushdown(x),x=ls;splay(x);return x;}//找根（在真实的树中的）

int get\_fa(int x){access(x),splay(x);if(!ls)return -1;pushdown(x);x=ls;while(rs)pushdown(x),x=rs;splay(x);return x;}

void split(int x,int y){makeroot(x);access(y);splay(y);}//提取路径，得到一棵独立的[x,y]路径的splay，splay根节点为y

void link(int x,int y){makeroot(x);if(findroot(y)!=x)f[x]=y;}//连边

void cut(int x,int y){makeroot(x);if(findroot(y)==x&&f[y]==x&&!c[y][0]){f[y]=c[x][1]=0;pushup(x);}}//断边

}lct;

int query(int x,int y)

{

lct.makeroot(x);

lct.access(y);

lct.splay(y);

return lct.size[y] - 1;

}

int main()

{

int n,m;

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

lct.init(n);

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

{

lct.splay(i);

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

}

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

{

int op,x,y;

scanf("%d%d%d",&op,&x,&y);

if(op == 0)

{//询问x到y的异或和

lct.split(x,y);

printf("%d\n",xr[y]);

}

else if(op == 1)

{//连接x到y

if(lct.findroot(x) == lct.findroot(y))

continue;

lct.link(x,y);

}

else if(op == 2)

{//删除边xy

if(lct.findroot(x) != lct.findroot(y))

continue;

lct.cut(x,y);

}

else

{//点x上权值变成y

lct.splay(x);

v[x] = y;

}

}

return 0;

}

# 分块

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<cmath>

#include<vector>

#define ll long long

**using** **namespace** std;

**const** **int** sz = **200010**;

**int** n;//数组长度

ll num[sz];//初始数组

**int** find\_block[sz];//第i个数属于第find\_block[i]块

**struct** block\_

{

ll num;

}block[sz];//记录块内信息

vector < **int** > ve[sz];

**int** block\_num;//整块的数量

**void** **block\_init**()

{

block\_num = sqrt(n);

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

{

find\_block[i] = (i-**1**) / block\_num + **1**;

ve[find\_block[i]].push\_back(num[i]);

}

**for**(**int** i = **1** ; i <= find\_block[n] ; i ++)

sort(ve[i].begin(),ve[i].end());

**return** ;

}

**void** **reset**(**int** x)//第x个块

{

ve[x].clear();

**int** l = (x-**1**) \* block\_num + **1** , r = min(x\*block\_num,n);

**for**(**int** i = l ; i <= r ; i ++)

ve[x].push\_back(num[i]);

sort(ve[x].begin(),ve[x].end());

}

//找到x所属块的左右端点

**int** **block\_l**(**int** x)

{

**return** (find\_block[x]-**1**) \* block\_num + **1**;

}

**int** **block\_r**(**int** x)

{

**return** find\_block[x] \* block\_num;

}

**void** **change**(**int** l,**int** r,**int** c)

{

**for**(**int** i = l ; i <= min(r,block\_r(l)) ; i ++) //块内暴力

num[i] += c;

reset(find\_block[l]);

**if**(find\_block[l] != find\_block[r]) //不重合的话将右端点的块也暴力处理

{

**for**(**int** i = block\_l(r) ; i <= r ; i ++)

num[i] += c;

reset(find\_block[r]);

}

**for**(**int** i = find\_block[l] + **1** ; i <= find\_block[r] - **1** ; i ++)//每个块 O(1)

block[i].num += c;

**return** ;

}

**int** **ask**(**int** l,**int** r,**int** c)//查询lr中小于c的个数

{

**int** ans = **0**;

**for**(**int** i = l ; i <= min(r,block\_r(l)) ; i ++) //块内暴力

**if**(num[i] + block[find\_block[l]].num < c)

ans ++;

**if**(find\_block[l] != find\_block[r]) //不重合的话将右端点的块也暴力处理

{

**for**(**int** i = block\_l(r) ; i <= r ; i ++)

**if**(num[i] + block[find\_block[r]].num < c)

ans ++;

}

**for**(**int** i = find\_block[l] + **1** ; i <= find\_block[r] - **1** ; i ++)//每个块 O(1)

{

**int** x = c - block[i].num;

ans += lower\_bound(ve[i].begin(),ve[i].end(),x) - ve[i].begin();

}

**return** ans;

}

**int** **main**()

{

scanf("%d",&n);

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

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

block\_init();

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

{

**int** opt,l,r;

ll c;

scanf("%d%d%d%lld",&opt,&l,&r,&c);

**if**(opt == **0**)

change(l,r,c);

**if**(opt == **1**)

printf("%d**\n**",ask(l,r,c\*c));

}

**return** **0**;

}

# 树链剖分

#include<iostream>

#include<cstdio>

#include<cstring>

#define ll long long

using namespace std;

const int sz = 200010;

ll num[sz];

int n,m;

int f[sz],d[sz],top[sz],id[sz],rk[sz],size[sz],son[sz],cnt;

char ins[11];

struct Tree

{

int l,r;

ll sum,max,add;

}tree[sz<<2];

void update(int p)

{

tree[p].sum = tree[p<<1].sum + tree[p<<1|1].sum;

tree[p].max = max(tree[p<<1].max,tree[p<<1|1].max);

}

void spread(int p)

{

if(tree[p].add)

{

tree[p<<1].add += tree[p].add;

tree[p<<1|1].add += tree[p].add;

tree[p<<1].sum += tree[p].add \* (tree[p<<1].r - tree[p<<1].l + 1);

tree[p<<1|1].sum += tree[p].add \* (tree[p<<1|1].r - tree[p<<1|1].l + 1);

tree[p<<1].max += tree[p].add;

tree[p<<1|1].max += tree[p].add;

tree[p].add = 0;

}

}

void build\_tree(int p,int l,int r)

{

tree[p].l = l , tree[p].r = r;

if(l == r)

{

tree[p].sum = tree[p].max = num[rk[l]];

return ;

}

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

build\_tree(p<<1,l,mid);

build\_tree(p<<1|1,mid+1,r);

update(p);

}

void change(int p,int l,int r,int x)

{

if(tree[p].l >= l && tree[p].r <= r)

{

tree[p].add += x;

tree[p].sum += x \* (tree[p].r - tree[p].l + 1);

tree[p].max += x;

return ;

}

spread(p);

int mid = (tree[p].l + tree[p].r) >> 1;

if(mid >= l)

change(p<<1,l,r,x);

if(mid < r)

change(p<<1|1,l,r,x);

update(p);

}

ll ask\_sum(int p,int l,int r)

{

if(tree[p].l >= l && tree[p].r <= r)

return tree[p].sum;

spread(p);

ll ans = 0;

int mid = (tree[p].l + tree[p].r) >> 1;

if(mid >= l)

ans += ask\_sum(p<<1,l,r);

if(mid < r)

ans += ask\_sum(p<<1|1,l,r);

return ans;

}

ll ask\_max(int p,int l,int r)

{

if(tree[p].l >= l && tree[p].r <= r)

return tree[p].max;

spread(p);

ll ans = -2147483641;

int mid = (tree[p].l + tree[p].r) >> 1;

if(mid >= l)

ans = max(ask\_max(p<<1,l,r),ans);

if(mid < r)

ans = max(ask\_max(p<<1|1,l,r),ans);

return ans;

}

int hed[sz],nxt[sz],l[sz];

int tot = 1;

void build(int f,int t)

{

l[tot] = t;

nxt[tot] = hed[f];

hed[f] = tot ++;

}

void dfs1(int u,int fa,int dep)

{

d[u] = dep;

f[u] = fa;

size[u] = 1;

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i];

if(v != fa)

{

dfs1(v,u,dep+1);

size[u] += size[v];

if(size[v] > size[son[u]])

son[u] = v;

}

}

}

void dfs2(int u,int t)

{

top[u] = t;

id[u] = ++ cnt;

rk[cnt] = u;

if(!son[u])

return ;

dfs2(son[u],t);

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i];

if(v != f[u] && v != son[u])

{

dfs2(v,v);

}

}

}

ll sum(int x,int y)

{

ll ans = 0;

while(top[x] != top[y])

{

if(d[top[x]] < d[top[y]])

swap(x,y);

ans += ask\_sum(1,id[top[x]],id[x]);

x = f[top[x]];

}

if(id[x] > id[y])

swap(x,y);

ans += ask\_sum(1,id[x],id[y]);

return ans;

}

ll maxs(int x,int y)

{

ll ans = -2147483641;

while(top[x] != top[y])

{

if(d[top[x]] < d[top[y]])

swap(x,y);

ans = max(ask\_max(1,id[top[x]],id[x]),ans);

x = f[top[x]];

}

if(id[x] > id[y])

swap(x,y);

ans = max(ask\_max(1,id[x],id[y]),ans);

return ans;

}

void changes(int x,int y,ll z)

{

while(top[x] != top[y])

{

if(d[top[x]] < d[top[y]])

swap(x,y);

change(1,id[top[x]],id[x],z);

x = f[top[x]];

}

if(id[x] > id[y])

swap(x,y);

change(1,id[x],id[y],z);

}

int read()

{

int x = 0 , f = 1;

char ins = getchar();

while(ins < '0' || ins > '9')

{

if(ins == '-')

f = -1;

ins = getchar();

}

while(ins >= '0' && ins <= '9')

{

x = (x << 3) + (x << 1) + ins - '0';

ins = getchar();

}

return x \* f;

}

int main()

{

n = read();

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

{

int f,t;

f = read() , t = read();

build(f,t) , build(t,f);

}

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

num[i] = read();

dfs1(1,0,1);

dfs2(1,1);

build\_tree(1,1,n);

m = read();

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

{

int u,v;

scanf("%s",ins);

u = read() , v = read();

if(ins[0] == 'C')

{//点修改

ll temp = sum(u,u);

changes(u,u,-temp);

changes(u,u,v);

}

else if(ins[1] == 'M')

{//区间最大

printf("%lld\n",maxs(u,v));

}

else

{//区间求和

printf("%lld\n",sum(u,v));

}

}

return 0;

}

# 约瑟夫问题

#include<bits/stdc++.h>

#define ll long long

using namespace std;

ll jos\_On(ll n,ll m,ll k)// O(m)求n个人走k步第m个出局的玩家

{

ll ans = (k-1) % (n-m+1) + 1;

if(k == 1)

return m;

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

{

ans = (ans + k - 1) % (i + n - m) + 1;

}

return ans;

}

ll jos\_Klogn(ll n,ll m,ll k)//O(klogn)求n个人走k步第m个出局的玩家

{

ll ans = (k-1) % (n-m+1) + 1;

if(k == 1)

return m;

ll now = 1 , a = n - m;

while(now < m)

{

ll d = (ll)ceil((now+a-ans)\*1.0/(k-1));

if(d == 0)

d ++;

if(now + d >= m)

d = m - now;

now += d;

ll mod = (now + a);

ans = (ans + k \* d % mod - 1 + mod) % mod + 1;

}

return ans;

}

inline ll read()

{

ll x = 0;

char in = getchar();

while(in < '0' || in > '9')

in = getchar();

while(in >= '0' && in <= '9')

{

x = (x << 3) + (x << 1) + in - '0';

in = getchar();

}

return x;

}

int main()

{

int T;

scanf("%d",&T);

int cas = 0;

while(T --)

{

long long n = read(),m = read(),k = read();

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

if(m <= k)

printf("%I64d\n",jos\_On(n,m,k));

else

printf("%I64d\n",jos\_Klogn(n,m,k));

}

return 0;

}

# 完美消除序列&&弦图最小染色

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<cstring>

#include<queue>

#include<vector>

using namespace std;

const int sz = 200010;

vector < int > l[sz];

void build(int f,int t)

{

l[f].push\_back(t);

l[t].push\_back(f);

}

int R[sz],SA[sz],lab[sz],col[sz];

priority\_queue <pair<int,int> > q;

int n,m;

void color(int u)

{

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

{

int v = l[u][i];

if(lab[v] == -1)

continue;

R[lab[v]] = u;

}

for(int i = 1 ; lab[u] == -1 ; i ++)

{

if(R[i] != u)

lab[u] = i;

}

}

int main()

{

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

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

{

int f,t;

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

build(f,t);

}

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

memset(lab,0,sizeof(lab));

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

{

q.push(make\_pair(0,i));

}

for(int i = n ; i >= 1;)

{

int now = q.top().second;

q.pop();

if(R[now] != -1)

continue;

SA[i] = now;

R[now] = i --;

for(int j = 0 ; j < l[now].size() ; j ++)

{

int v = l[now][j];

if(R[v] != -1)

continue;

lab[v] ++;

q.push(make\_pair(lab[v],v));

}

}

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

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

//SA为弦图的完美消除序列

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

color(SA[i]);

int ans = 0;

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

ans = max(ans,lab[i]);

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

return 0;

}

# 欧拉回路

void dfs(int u)

{

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

{

int v = l[u][i].t;

int w = l[u][i].d;//w为边号

if(!vis[w])

{

vis[w]=true;

dfs(v);

ans[++ans[0]]=w;

}

}

}

# 哈密顿回路

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<vector>

using namespace std;

const int sz = 510;

int n,m;

bool l[sz][sz];

int read()

{

int x = 0;

char in = getchar();

while(in < '0' || in > '9')

{

in = getchar();

}

while(in >= '0' && in <= '9')

{

x = (x << 3) + (x << 1) + in - '0';

in = getchar();

}

return x;

}

int ans[sz];

bool vis[sz];

int s,e;

void dfs(int u)

{

vis[u] = 1;

e = u;

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

{

if(l[u][i])

{

int v = i;

if(vis[v])

continue;

ans[++ans[0]] = v;

dfs(v);

break;

}

}

}

void reverse(int l,int r)

{

while(l < r)

{

swap(ans[l],ans[r]);

l ++;

r --;

}

}

void solve()

{

s = 1;

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

{

if(l[s][i])

{

e = i;

break;

}

}

vis[s] = 1 , vis[e] = 1;

ans[++ans[0]] = s;

ans[++ans[0]] = e;

while(12 < 450)

{

dfs(e);

reverse(1,ans[0]);

swap(s,e);

dfs(e);

int mid = 0;

if(!l[s][e])

{

for(int i = 2 ; i < ans[0] ; i ++)//

{

if(l[ans[i]][e] && l[ans[i+1]][s])

{

mid = i + 1;

break;

}

}

reverse(mid,ans[0]);

e = ans[ans[0]];

}

if(ans[0] == n)

break;

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

{

if(!vis[i])

{

int j;

for( j = 2 ; j < ans[0] ; j ++)

{

if(l[ans[j]][i])

{

mid = j;

break;

}

}

if(l[ans[mid]][i])

{

e = i;

mid = j;

break;

}

}

}

s = ans[mid-1];

reverse(1,mid-1);

reverse(mid,ans[0]);

ans[++ans[0]] = e;

vis[e] = 1;

}

}

int main()

{

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

{

n \*= 2;

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

memset(ans,0,sizeof(ans));

//哈密顿回路 例题题意需要建补图(不用在意建图)

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

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

if(i == j)

l[i][j] = 0;

else

l[i][j] = 1;

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

{

int f = read() , t = read();

l[f][t] = 0;

l[t][f] = 0;

}

solve();

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

printf("%d%c",ans[i],i == (ans[0]) ? '\n' : ' ');

puts("");

}

return 0;

}

# 树论

## 选出尽可能多的路径且路径间不能有重叠

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int sz = 200010;

int hed[sz],nxt[sz];//dist[i][j]记录了第 i 个节点向上走 2^j 步走的边的权值和

int par[sz][32],deep[sz];//par[i][j]记录了第 i 个节点向上走 2^j 步可以到达的节点，deep记录了深度

int tot = 1;//边的数量

int l[sz];

int n,m;

bool vis[sz];

struct node

{

int u,v,lcauv;

}ins[sz];

void init()

{

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

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

memset(l,0,sizeof(l));

memset(par,0,sizeof(par));

memset(deep,0,sizeof(deep));

memset(hed,0,sizeof(hed));

memset(nxt,0,sizeof(nxt));

tot = 1;

}

bool cmp(node a,node b)

{

return deep[a.lcauv] > deep[b.lcauv];

}

void build(int f,int t) // 建一条 f点到 t 点权值为 d 的有向边

{

l[tot] = t;

nxt[tot] = hed[f];

hed[f] = tot ++;

}

void dfs(int u,int fa,int dep)//预处理深度、每个节点的父亲节点、边权

{

par[u][0] = fa;

deep[u] = dep;

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i];

if(v != fa)

dfs(v,u,dep+1);

}

}

int lca(int u,int v)//求树上 u v 两点的距离

{

if(deep[u] < deep[v])

swap(u,v);

for(int i = 31 ; i >= 0 ; i --)

if(deep[par[u][i]] >= deep[v])

u = par[u][i];

for(int i = 31 ; i >= 0 ; i --)

if(par[u][i] != par[v][i])

u = par[u][i] , v = par[v][i];

if(u != v)

u = par[u][0] , v = par[v][0];

return u; // 此时,若 u v 两点联通(即树不是森林) u == v ，且此时的 u v 为询问时 u v 的最近公共祖先

}

void dfs1(int u)

{

vis[u] = 1;

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i];

if(vis[v] || deep[v] < deep[u])

continue;

dfs1(v);

}

}

//n个点的树，给定m条路径，选出尽可能多的路径且路径间不能有重叠

int main()

{

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

{

init();

for(int i = 1 ; i < n ; i ++)//点编号从 1 开始

{

int f,t;

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

build(f,t);

build(t,f);//建边

}

dfs(1,0,1);//预处理

for(int i = 1 ; i <= 31 ; i ++)

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

par[j][i] = par[par[j][i-1]][i-1];//倍增思想

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

{

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

ins[i].lcauv = lca(ins[i].u,ins[i].v);

}

sort(ins+1,ins+m+1,cmp);

int ans = 0;

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

{

if(vis[ins[i].u] || vis[ins[i].v])

continue;

ans ++;

dfs1(ins[i].lcauv);

}

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

}

return 0;

}

## 树上点分治

### 求一条简单路径，权值和等于 K，且边的数量最小

#include <bits/stdc++.h>

using namespace std;

const int sz = 2000100;

const int INF = 2e9;

int n,k;

struct node

{

int t,d;

};

vector <node> l[sz];

int q[sz];

int root,maxx[sz],size[sz],sum;

bool vis[sz];

void build(int f,int t,int d)

{

l[f].push\_back({t,d});

l[t].push\_back({f,d});

}

void calcsiz(int u,int fa)

{

size[u] = 1;

maxx[u] = 0;

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

{

int v = l[u][i].t;

int w = l[u][i].d;

if(v == fa || vis[v])

continue;

calcsiz(v,u);

maxx[u] = max(maxx[u],size[v]);

size[u] += size[v];

}

maxx[u] = max(maxx[u],sum-size[u]);

if(maxx[u] < maxx[root])

root = u;

}

struct P

{

int dis,tim;

}dd[sz];

int cnt;

int dist[sz];

void calcdis(int u,int fa,int dep)

{

dd[++cnt].dis = dist[u];

dd[cnt].tim = dep;

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

{

int v = l[u][i].t;

int w = l[u][i].d;

if(vis[v] || v == fa)

continue;

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

calcdis(v,u,dep+1);

}

}

int tf[10000010];

int ans = 214748364;

queue < int > tag;

void dfz(int u)

{

tf[0] = 0;

tag.push(0);

vis[u] = true;

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

{

int v = l[u][i].t;

int w = l[u][i].d;

if(vis[v])

continue;

dist[v] = w;

calcdis(v,u,1);

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

if(dd[j].dis <= k)

if(tf[k - dd[j].dis] != -1)

{

ans = min(ans,tf[k-dd[j].dis] + dd[j].tim);

}

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

{

if(tf[dd[j].dis] == -1)

tf[dd[j].dis] = dd[j].tim;

else

tf[dd[j].dis] = min(tf[dd[j].dis],dd[j].tim);

tag.push(dd[j].dis);

}

cnt = 0;

}

while(!tag.empty())

{

tf[tag.front()] = -1;

tag.pop();

}

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

{

int v = l[u][i].t;

int w = l[u][i].d;

if(vis[v])

continue;

sum = size[v];

root = 0;

maxx[root] = INF;

calcsiz(v,u);

calcsiz(root,-1);

dfz(root);

}

}

int main()

{

for(int i = 0 ; i <= 1000000 ; i ++)

tf[i] = -1;

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

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

{

int a,b,c;

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

a ++ , b ++;

build(a,b,c);

}

root = 0;

maxx[root] = INF;

sum = n;

calcsiz(1, -1);

calcsiz(root, -1);

dfz(root);

if(ans == 214748364)

puts("-1");

else

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

return 0;

}

### Q个询问，询问长度为k的路径是否存在

#include <bits/stdc++.h>

#define ll long long

using namespace std;

const int sz = 20010;

const int INF = 2e9;

int n,m;

struct node

{

int t;

int d;

};

vector <node> l[sz];

int q[sz];

int root,maxx[sz],size[sz],sum;

bool vis[sz];

void build(int f,int t,int d)

{

l[f].push\_back({t,d});

l[t].push\_back({f,d});

}

void calcsiz(int u,int fa)

{

size[u] = 1;

maxx[u] = 0;

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

{

int v = l[u][i].t;

int w = l[u][i].d;

if(v == fa || vis[v])

continue;

calcsiz(v,u);

maxx[u] = max(maxx[u],size[v]);

size[u] += size[v];

}

maxx[u] = max(maxx[u],sum-size[u]);

if(maxx[u] < maxx[root])

root = u;

}

int dd[sz],cnt;

int dist[sz];

void calcdis(int u,int fa)

{

dd[++cnt] = dist[u];

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

{

int v = l[u][i].t;

ll w = l[u][i].d;

if(vis[v] || v == fa)

continue;

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

calcdis(v,u);

}

}

bool tf[10000010];

int ans[sz];

queue < int > tag;

void dfz(int u,int fa)

{

tf[0] = true;

tag.push(0);

vis[u] = true;

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

{

int v = l[u][i].t;

int w = l[u][i].d;

if(vis[v] || v == fa)

continue;

dist[v] = w;

calcdis(v,u);

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

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

if(q[k] >= dd[j])

ans[k] |= tf[q[k] - dd[j]];

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

{

tf[dd[j]] = true;

tag.push(dd[j]);

}

cnt = 0;

}

while(!tag.empty())

{

tf[tag.front()] = false;

tag.pop();

}

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

{

int v = l[u][i].t;

int w = l[u][i].d;

if(vis[v] || v == fa)

continue;

sum = size[v];

root = 0;

maxx[root] = INF;

calcsiz(v,u);

calcsiz(root,-1);

dfz(root,u);

}

}

int main()

{

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

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

{

int a,b,c;

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

build(a,b,c);

}

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

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

root = 0;

maxx[root] = INF;

sum = n;

calcsiz(1, -1);

calcsiz(root, -1);

dfz(root, -1);

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

{

if(q[i] == 0)

{

puts("Yes");

continue;

}

if(ans[i])

printf("Yes\n");

else

printf("No\n");

}

return 0;

}

### 求存在多少条长度为3的倍数的路径

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<vector>

#include<cstring>

#define ll long long

using namespace std;

const int sz = 200100;

const int INF = 2e9;

int n,k;

struct node

{

int t,d;

}l[sz];

int hed[sz],nxt[sz];

int tot = 1;

int root,maxx[sz],size[sz],sum;

bool vis[sz];

void build(int f,int t,int d)

{

l[tot].t = t;

l[tot].d = d;

nxt[tot] = hed[f];

hed[f] = tot ++;

}

void calcsiz(int u,int fa)

{

size[u] = 1;

maxx[u] = 0;

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i].t;

int w = l[i].d;

if(v == fa || vis[v])

continue;

calcsiz(v,u);

maxx[u] = max(maxx[u],size[v]);

size[u] += size[v];

}

maxx[u] = max(maxx[u],sum-size[u]);

if(maxx[u] < maxx[root])

root = u;

}

ll dis[11];

ll dist[sz];

void calcdis(int u,int fa)

{

dis[dist[u]%3] ++;

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i].t;

int w = l[i].d % 3;

if(vis[v] || v == fa)

continue;

dist[v] = (dist[u] + w) % 3;

calcdis(v,u);

}

}

ll solve(int u,int d,int fa)

{

dis[0] = 0 , dis[1] = 0, dis[2] = 0;

dist[u] = d % 3;

calcdis(u,fa);

ll anss = dis[1] \* dis[2] \* 2ll + dis[0] \* dis[0];

/\* int i = 1 , j = cnt;

while(i < j)

{

while(i < j && dd[i] + dd[j] > k)

j --;

anss += j - i;

i ++;

}\*/

return anss;

}

ll ans;

void dfz(int u,int fa)

{

ans += solve(u,0,fa);

vis[u] = true;

for(int i = hed[u] ; i ; i = nxt[i])

{

int v = l[i].t;

int w = l[i].d;

if(vis[v])

continue;

ans -= solve(v,w,u);

sum = size[v];

root = 0;

maxx[root] = INF;

calcsiz(v,u);

calcsiz(root,-1);

dfz(root,u);

}

}

void init()

{

memset(hed,0,sizeof(hed));

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

ans = 0 , tot = 1;

}

ll gcd(ll a,ll b)

{

if(b == 0)

return a;

return gcd(b,a%b);

}

int main()

{

scanf("%d",&n);

{

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

{

int a,b,c;

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

build(a,b,c%3);

build(b,a,c%3);

}

root = 0;

maxx[root] = INF;

sum = n;

calcsiz(1, -1);

calcsiz(root, -1);

dfz(root, -1);

ll gcd\_ = gcd(ans,1ll\*n\*n);

printf("%lld/%lld\n", ans/gcd\_,n\*n/gcd\_);

init();

}

return 0;

}

# 快速幂 & 费马小定理

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#define ll long long

**using** **namespace** std;

**const** **int** mod = **1000000007**;

ll **ksm**(ll x,ll p)

{

**if**(p == **0**)

**return** **1**;

**if**(p == **1**)

**return** x % mod;

**if**(p == **2**)

**return** ((x%mod) \* (x%mod))%mod;

**int** temp = ksm(x,p/**2**) % mod;

**if**(p % **2** == **1**)

**return** (((temp \* temp) % mod) \* (x%mod));

**if**(p % **2** == **0**)

**return** (temp \* temp) % mod;

}

**int** **get**(**int** a)

{

**return** ksm(a,mod-**2**);

}