**图论**

2020.09.05

目录

[图论 4](#_Toc50555600)

[单源最短路 4](#_Toc50555601)

[dijkstra 4](#_Toc50555602)

[K短路 4](#_Toc50555603)

[多元最短路 6](#_Toc50555604)

[Floyd 6](#_Toc50555605)

[Johnson 7](#_Toc50555606)

[树的直径 9](#_Toc50555607)

[最小生成树 9](#_Toc50555608)

[Prim 9](#_Toc50555609)

[Kruskal 10](#_Toc50555610)

[Boruvka 11](#_Toc50555611)

[网络流 12](#_Toc50555612)

[最大流 12](#_Toc50555613)

[费用流 19](#_Toc50555614)

[网络流模型 22](#_Toc50555615)

[最近公共祖先（LCA） 22](#_Toc50555616)

[倍增法 22](#_Toc50555617)

[rmq法 23](#_Toc50555618)

[Tarjan法 24](#_Toc50555619)

[最小斯坦纳树 25](#_Toc50555620)

[匹配问题 27](#_Toc50555621)

[二分图最大匹配 27](#_Toc50555622)

[二分图最优匹配 28](#_Toc50555623)

[一般图最大匹配 30](#_Toc50555624)

[一般图最优匹配 30](#_Toc50555625)

[差分约束系统 30](#_Toc50555626)

[线性规划 30](#_Toc50555627)

[最小树形图 30](#_Toc50555628)

[度数序列生成图 30](#_Toc50555629)

[判定 30](#_Toc50555630)

[构造 31](#_Toc50555631)

[欧拉回路 31](#_Toc50555632)

[有向图欧拉回路 31](#_Toc50555633)

[无向图欧拉回路 32](#_Toc50555634)

[一笔画问题 34](#_Toc50555635)

[fluery算法 34](#_Toc50555636)

[计数问题 34](#_Toc50555637)

[虚树 34](#_Toc50555638)

[适用范围 34](#_Toc50555639)

[简单做法 34](#_Toc50555640)

[构建方法 34](#_Toc50555641)

[代码 35](#_Toc50555642)

[从编码到各种树的计数 36](#_Toc50555643)

[编码 36](#_Toc50555644)

[有标号无根树的计数(公式) 36](#_Toc50555645)

[有标号有根树的计数 37](#_Toc50555646)

[无标号有根树的计数 37](#_Toc50555647)

[无标号无根树的计数 38](#_Toc50555648)

[从哈密顿路到竞赛图(tournament)相关 39](#_Toc50555649)

[哈密顿路 39](#_Toc50555650)

[竞赛图 40](#_Toc50555651)

[联通分量 40](#_Toc50555652)

[强连通分量 40](#_Toc50555653)

[点双联通分量 41](#_Toc50555654)

[dsu on tree 42](#_Toc50555655)

[拓扑排序 45](#_Toc50555656)

[一般情况 45](#_Toc50555657)

[求字典序最小解 46](#_Toc50555658)

[求小的数尽量靠前的解 46](#_Toc50555659)

[2-sat 46](#_Toc50555660)

[1. 具体实现 46](#_Toc50555661)

[2. 输出一组方案 46](#_Toc50555662)

[3. 输出字典序最小解 50](#_Toc50555663)

# 图论

## 单源最短路

### dijkstra

const int N = 100101; **//N是点数最大值**

typedef long long ll;

typedef pair<ll , int> pli;

ll dis[N]; bool vis[N];

vector <pli> o[N];

priority\_queue <pli , vector<pli> , greater<pli> > Q;

int n , m; **//n是总点数，m是总边数**

int S = 0 , T = 0; **//S是起点，T是终点**

void add(int s , int t , int v){

o[s].push\_back(make\_pair(v , t));

}

void dij(){

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

dis[i] = -1;

dis[S] = 0; Q.push(make\_pair(0 , S));

while(!Q.empty()){

ll d = Q.top().first;

int now = Q.top().second; Q.pop();

if(vis[now]) continue;

vis[now] = 1;

for(int i = 0 ; i < o[now].size() ; ++ i){

int to = o[now][i].second; ll nl = d + o[now][i].first;

if(dis[to] == -1 || nl < dis[to]){

dis[to] = nl;

Q.push(make\_pair(dis[to] , to));

}

}

}

}

### K短路

#include<iostream>

#include<stdio.h>

#include<string.h>

#include<algorithm>

#include<queue>

#define maxn 200005

using namespace std;

struct nod

{

int now;

double dis;

nod(int a,double b)

{

now=a;

dis=b;

}

nod(){}

};

int n,m,tot[2];

double dis[maxn],sum;

int head[2][maxn],nex[2][maxn],to[2][maxn];

double val[2][maxn];

void add(int f,int x,int y,double z)

{

to[f][++tot[f]]=y; val[f][tot[f]]=z; nex[f][tot[f]]=head[f][x]; head[f][x]=tot[f];

}

int vis[maxn];

queue<int> qq;

void spfa(int start)

{

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

dis[i]=123456789.0;

dis[start]=0;

qq.push(start);

while(!qq.empty())

{

int now=qq.front(); qq.pop(); vis[now]=0;

for(int i=head[0][now];i;i=nex[0][i])

{

if(dis[to[0][i]]>dis[now]+val[0][i])

{

dis[to[0][i]]=dis[now]+val[0][i];

vis[to[0][i]]=1;

qq.push(to[0][i]);

}

}

}

}

bool operator<(nod a,nod b)

{

return dis[a.now]+a.dis>dis[b.now]+b.dis;

}

priority\_queue<nod> q;

int cnt[maxn],ans;

void astar(int start,int maxx)

{

q.push(nod(start,0));

while(!q.empty())

{

nod now=q.top(); q.pop();

if(now.dis+dis[now.now]>sum)

return;

cnt[now.now]++;

if(cnt[now.now]>maxx)

continue;

if(now.now==n)

{

if(sum>=now.dis)

sum-=now.dis,ans++;

else

return;

}

else

{

for(int i=head[1][now.now];i;i=nex[1][i])

{

q.push(nod(to[1][i],now.dis+val[1][i]));

}

}

}

}

int main()

{

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

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

{

int x,y;

double z;

scanf("%d%d%lf",&x,&y,&z);

add(1,x,y,z);

add(0,y,x,z);

}

spfa(n);

astar(1,sum/dis[1]);

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

}

## 多元最短路

### Floyd

**在时间复杂度下求出不含负环的图每对点之间的最短路**

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

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

f[i][j] = INF;

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

int u , v , w;

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

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

}

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

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

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

f[i][j] = min(f[i][k] + f[k][j] , f[i][j]);

### Johnson

**//可以在的时间内计算出含负边权的点对最短路**

#include <iostream>

#include <cstdio>

#include <queue>

using namespace std;

typedef long long ll;

typedef pair<ll , int> pli;

const int N = 3030 , M = 10101 , INF = 1e9;

class Edges{

public:

int next , to;

int v;

} edge[M];

int first[N] , ct[N];

ll dis[N] , h[N]; bool vis[N];

ll f[N][N];

priority\_queue <pli , vector<pli> , greater<pli> > pQ;

queue <int> Q;

int n , m , te;

void add(int s , int t , int v){

++ te; edge[te].to = t; edge[te].next = first[s]; edge[te].v = v; first[s] = te;

}

bool spfa(){ **//spfa判断有无负环**

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

dis[i] = 0; vis[i] = 1; Q.push(i);

}

while(!Q.empty()){

int v = Q.front(); Q.pop(); vis[v] = 0;

for(int e = first[v] ; e ; e = edge[e].next){

int to = edge[e].to;

if(dis[to] > dis[v] + edge[e].v){

dis[to] = dis[v] + edge[e].v;

if(!vis[to]){

ct[to] ++; if(ct[to] == n) return 0;

Q.push(to);

}

}

}

}

return 1;

}

void dij(int S){

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

dis[i] = -1; vis[i] = 0;

}

dis[S] = 0; pQ.push(make\_pair(0 , S));

while(!pQ.empty()){

ll d = pQ.top().first;

int now = pQ.top().second; pQ.pop();

if(vis[now]) continue;

vis[now] = 1;

for(int e = first[now] ; e ; e = edge[e].next){

int to = edge[e].to;

if(dis[to] == -1 || d + edge[e].v < dis[to]){

dis[to] = d + edge[e].v;

pQ.push(make\_pair(dis[to] , to));

}

}

}

}

int main(){

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

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

int u , v , w;

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

add(u , v , w);

}

if(!spfa()){

printf("-1\n");

return 0;

}

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

h[i] = dis[i];

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

for(int e = first[i] ; e ; e = edge[e].next)

edge[e].v += h[i] - h[edge[e].to];

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

dij(i);

ll ans = 0;

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

f[i][j] = (dis[j] == -1 ? INF : dis[j] + h[j] - h[i]);

ans += f[i][j] \* j;

}

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

}

return 0;

}

## 树的直径

## 最小生成树

### Prim

typedef long long ll;

typedef pair<int,int> pii;

typedef vector<pii> vpii;

const int N = 5050; **//点数**

priority\_queue <pii , vector<pii> , greater<pii> > Q;

int n , m;

vpii e[N];

bool vis[N];

int main(){

ios::sync\_with\_stdio(0);

cin >> n >> m;

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

int s , t , v;

cin >> s >> t >> v;

e[s].push\_back(make\_pair(v , t));

e[t].push\_back(make\_pair(v , s));

}

vis[1] = 1;

for(int i = 0 ; i < e[1].size() ; ++ i)

Q.push(e[1][i]);

ll ans = 0;

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

while(!Q.empty()){

pii top = Q.top();

if(vis[top.second])

Q.pop();

else

break;

}

if(Q.empty()){

cout << "orz" << endl; **//不连通**

return 0;

}

pii top = Q.top(); Q.pop();

int pt = top.second; vis[pt] = 1;

ans += top.first;

for(int i = 0 ; i < e[pt].size() ; ++ i)

Q.push(e[pt][i]);

}

cout << ans << endl;

return 0;

}

### Kruskal

#include <iostream>

#include <algorithm>

using namespace std;

const int N = 5050 , M = 2e5 + 10; **//N点数 M边数**

class Edges{

public:

int s , t , v;

} e[M];

int root[N];

int getroot(int x){

return root[x] = (x == root[x] ? x : getroot(root[x]));

}

int n , m;

bool cmp(Edges e1 , Edges e2){

return e1.v < e2.v;

}

int main(){

ios::sync\_with\_stdio(0);

cin >> n >> m;

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

cin >> e[i].s >> e[i].t >> e[i].v;

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

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

root[i] = i;

long long ans = 0;

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

int S = getroot(e[i].s) , T = getroot(e[i].t);

if(S != T){

ans += e[i].v;

root[S] = T;

}

}

cout << ans << endl;

return 0;

}

### Boruvka

**这是一种基于合并的最小生成树算法**

#include <iostream>

using namespace std;

const int N = 5050 , M = 2e5 + 10;

int n , m;

int s[M] , t[M] , v[M];

int root[N];

int Mie[N] , flag[N] , to[N];

int getroot(int x){

return root[x] = (x == root[x] ? x : getroot(root[x]));

}

bool cmp(int e1 , int e2){

return v[e1] < v[e2] || v[e1] == v[e2] && e1 < e2;

}

int main(){

ios::sync\_with\_stdio(0);

cin >> n >> m;

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

cin >> s[i] >> t[i] >> v[i];

int Bcnt = n;

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

root[i] = i;

long long ans = 0;

while(Bcnt > 1){

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

int S = getroot(s[i]) , T = getroot(t[i]);

if(S != T){

if(flag[S] != Bcnt || cmp(i , Mie[S])){

Mie[S] = i;

flag[S] = Bcnt;

to[S] = T;

}

if(flag[T] != Bcnt || cmp(i , Mie[T])){

Mie[T] = i;

flag[T] = Bcnt;

to[T] = S;

}

}

}

int merge = 0;

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

if(getroot(i) == i){

if(flag[i] != Bcnt){

cout << "orz" << endl; **//不连通**

return 0;

}

if(getroot(to[i]) != i){

merge ++;

ans += v[Mie[i]];

root[i] = root[to[i]];

}

}

Bcnt -= merge;

}

cout << ans << endl;

return 0;

}

## 网络流

### 最大流

#### 普通网络流

**//CJY的dinic**

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int MAXN = 1010 + 10, MAXM = 1e5 + 10;

const int INF = 1e9 + 7;

class Edges{

public:

int v, next, to;

};

Edges edge[MAXM];

int te = 1;

int ans = 0 , dis[MAXN];

int first[MAXN], sec[MAXN];

bool vis[MAXN];

int S, T; **//point:[S,T]**

int q[MAXN];

int head = 0, tail = 1;

void add(int s , int t , int v){

++ te;

edge[te].to = t; edge[te].next = first[s];

edge[te].v = v; first[s] = te;

}

void bfs(){

head = 0, tail = 1;

for(int i = S + 1; i <= T; ++i)

dis[i] = INF;

memset(vis, 0, T+2);

q[++head] = S;

dis[S] = 0;

vis[S] = 1;

while(head <= tail){

for(int e = first[q[head]] ; e ; e = edge[e].next){

int to = edge[e].to;

if(edge[e].v && !vis[to]){

dis[to] = dis[q[head]] + 1;

vis[to] = 1;

q[++tail] = to;

}

}

++head;

}

}

int dfs(int p, int flow){

if(p == T)

return flow;

int ret = 0;

for(int &e = sec[p] ; e ; e = edge[e].next)

if(edge[e].v && dis[edge[e].to] == dis[p] + 1){

int f = dfs(edge[e].to , min(flow , edge[e].v));

edge[e].v -= f;

edge[e^1].v += f;

flow -= f;

ret += f;

if(!flow) break;

}

return ret;

}

void cal(){

while(1){

bfs();

if(dis[T] == INF)

break;

for(int i = S; i <= T; ++ i)

sec[i] = first[i];

ans += dfs(S , INF);

}

}

int main(){

cal();

cout << ans << endl;

return 0;

}

**//薛欣dinic**

void addedge(int x,int y,int v)

{

tot++;

edge[tot]=EDGE(y, t[x], v, tot+1);

t[x]=tot;

tot++;

edge[tot]=EDGE(x, t[y], 0, tot-1);

t[y]=tot;

}

int bfs()

{

for(int i=s;i<=e;i++) vist[i]=0;

que[1]=s; vist[s]=1;

int head=0,tail=1;

while(head<tail)

{

head++;

int x=que[head];

for(int p=t[x];p>0;p=edge[p].nxt)

{

int y=edge[p].to;

if(edge[p].v>0&&!vist[y])

{

vist[y]=vist[x]+1;

que[++tail]=y;

}

}

}

return vist[e];

}

int dfs(int x,int f)

{

if(x==e||!f) return f;

int sumf=0;

for(int p=t[x];p>0;p=edge[p].nxt)

{

int y=edge[p].to;

if(vist[y]==vist[x]+1&&edge[p].v>0)

{

int flow=dfs(y,min(f,edge[p].v));

edge[p].v-=flow;

edge[edge[p].re].v+=flow;

f-=flow;

sumf+=flow;

if(!f) break;

}

}

return sumf;

}

int dinic()

{

int ans=0;

while(bfs()) { ans+=dfs(s,inf); }

return ans;

}

#### 有上下界的网络流

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int N = 2020 + 10, M = 4e5 + 10;

const int INF = 1e9 + 7;

class Edges{

public:

int v, next, to;

};

Edges edge[M];

int te = 1;

int ans = 0 , dis[N];

int first[N], sec[N] , V[N];

bool vis[N];

int S, T , n;

int q[N];

int head = 0, tail = 1;

void add(int s , int t , int v){

++ te;

edge[te].to = t; edge[te].next = first[s];

edge[te].v = v; first[s] = te;

}

void link(int s , int t , int L , int R){

**//从s到t加入流量下界L，上界R的边**

V[s] -= L; V[t] += L;

add(s , t , R - L);

add(t , s , 0);

}

void bfs(){

head = 0, tail = 1;

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

dis[i] = INF;

memset(vis, 0, n+10);

q[++head] = S;

dis[S] = 0;

vis[S] = 1;

while(head <= tail){

for(int e = first[q[head]] ; e ; e = edge[e].next){

int to = edge[e].to;

if(edge[e].v && !vis[to]){

dis[to] = dis[q[head]] + 1;

vis[to] = 1;

q[++tail] = to;

}

}

++head;

}

}

int dfs(int p, int flow){

if(p == T)

return flow;

int ret = 0;

for(int &e = sec[p] ; e ; e = edge[e].next)

if(edge[e].v && dis[edge[e].to] == dis[p] + 1){

int f = dfs(edge[e].to , min(flow , edge[e].v));

edge[e].v -= f;

edge[e^1].v += f;

flow -= f;

ret += f;

if(!flow) break;

}

return ret;

}

void cal(){

while(1){

bfs();

if(dis[T] == INF)

break;

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

sec[i] = first[i];

ans += dfs(S , INF);

}

}

void init(){

memset(first , 0 , 4 \* (n + 10));

ans = 0;

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

V[i] = 0;

te = 1;

}

int main(){

int nn , mm;

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

n = nn + mm + 3;

init();

int g;

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

scanf("%d" , &g);

link(nn + i + 1 , nn + mm + 3 , g , INF);

}

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

int c , d , t , l , r; scanf("%d%d" , &c , &d);

link(0 , i + 1 , 0 , d);

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

scanf("%d%d%d" , &t , &l , &r);

link(i + 1 , nn + t + 2 , l , r);

}

}

S = 1 , T = n - 1; **//辅助源和辅助汇**

for(int i = 0 ; i <= n ; ++ i) **//对每个点积攒/缺乏的流量建边**

if(V[i] > 0){

add(S , i , V[i]);

add(i , S , 0);

ans -= V[i];

}

else if(V[i] < 0){

add(i , T , -V[i]);

add(T , i , 0);

}

link(n , 0 , 0 , INF);

cal();

if(ans < 0){

cout << -1 << endl;

cout << endl;

continue;

}

ans += edge[first[0]].v; **//加入最后那条循环边的流量**

S = 0; T = n; **//超级源和超级汇**

te -= 2; first[S] = edge[first[S]].next;

first[T] = edge[first[T]].next;

**//去掉循环边**

cal();

cout << ans << endl;

cout << endl;

}

return 0;

}

### 费用流

#### 最小费用最大流

**//XX**

int spfa(int S, int E)

{

for(int i=1;i<=E;i++) dist[i]=inf;

dist[0]=0; que.push(0); vist[0]=1;

while(que.empty() == 0)

{

int x=que.front();

que.pop();

for(int p=t[x];p>0;p=edge[p].nxt)

{

int y=edge[p].to;

if(edge[p].v>edge[p].u&&dist[y]>dist[x]+edge[p].c)

{

dist[y]=dist[x]+edge[p].c;

pree[y]=p; pre[y]=x;

if(vist[y]==0)

{

vist[y]=1;

que.push(y);

}

}

}

vist[x]=0;

}

return (dist[E]<inf);

}

void update(int S, int E)

{

LL maxa = inf;

int x = E;

while(x != S)

{

int p = pree[x];

maxa = min(maxa,(LL)edge[p].v - edge[p].u);

x = pre[x];

}

x = E;

while(x != S)

{

int p = pree[x];

cost += maxa \* edge[p].c;

edge[p].u += maxa; edge[edge[p].re].u -= maxa;

x = pre[x];

}

}

void solve(int S, int E, int n)

{

cost = 0;

while(spfa(S, E)) update(S, E);

cout<<cost<<endl;

}

#### 最大费用可行流

**//将边权取负值，跑最小费用流，但要注意判断可行流而非最大流**

**//如果看你出现负环要注意判负环**

**//判负环依据：经过的点数>n或经过的边数>m**

int spfa(int S, int E)

{

for(int i = 1;i <= E;i++) dist[i] = inf;

dist[S] = 0; que.push(0); vist[0] = 1; cnt[S] = 1;

while(que.empty() == 0)

{

int x = que.front();

que.pop();

for(int p = t[x]; p > 0; p = edge[p].nxt)

{

int y = edge[p].to;

if(edge[p].v > edge[p].u && dist[y] > dist[x] + edge[p].c)

{

· /\*

此处为判负环操作

cnt[y] = cnt[x] + 1;

if(cnt[y] > E) return 0;

\*/

dist[y] = dist[x] + edge[p].c;

pree[y] = p; pre[y] = x;

if(vist[y] == 0)

{

vist[y] = 1;

que.push(y);

}

}

}

vist[x] = 0;

}

**return (dist[E] <= 0); //判断可行流**

}

void update(int S, int E)

{

LL maxa = inf;

int x = E;

while(x != S)

{

int p = pree[x];

maxa = min(maxa,(LL)edge[p].v - edge[p].u);

x = pre[x];

}

x = E;

while(x != S)

{

int p = pree[x];

cost += maxa \* edge[p].c;

edge[p].u += maxa; edge[edge[p].re].u -= maxa;

x = pre[x];

}

}

void solve()

{

while(spfa(S, E)) update(S, E);

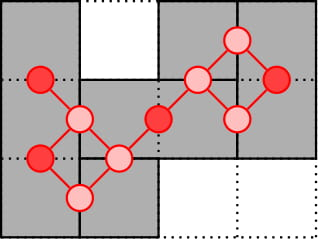
cout<<-cost<<endl;

}

### 网络流模型

#### CF 1404E bricks

给一个的网格，每一个网格或是#或是\*,#表示这个点你必须涂黑一次，\*表示这个点你一定不能涂黑。每次你可以涂一个或是的网格，一开始网格全白，求最后网格的满足要求的最少次数。



点拨：要善于把题目转换成最大独立集、最小点覆盖、最小路径覆盖。

思路：首先先把问题转化成最多去多少条边（方块个数-去掉边数=答案+1）

然后我们可以发现任意一个格点的四条边的相邻两条边不能同时去掉，这样我们把有限制的点连边，就变成了最大独立集问题，可以发现这是一个二分图，因此可以用dinic。

## 最近公共祖先（LCA）

### 倍增法

void dfs(int x, int fa) **//先求直接祖先**

{

for(int p = t[x]; p; p = edge[p].nxt)

{

int y = edge[p].to;

if(y != fa) { f[0][y] = x; dfs(y, x); }

}

}

void pre() //倍增

{

for(int i = 1; i <= 20; i++)

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

}

int get\_lca(int x, int y)//求lca

{

if(dep[y] < dep[x]) swap(x, y);

for(int i = 20; i >= 0; i--)

if(dep[y] - dep[x] >= (1 << i)) y = f[i][y];

**if(x == y) return x;//注意特判**

for(int i = 20; i >= 0; i--)

if(f[i][x] != f[i][y]){ x = f[i][x]; y = f[i][y];}

return f[0][x];

}

### rmq法

void dfs(int x)

{

def[x]=++now; **//时间戳**

rmin[++cont][0] = now; **//记入列表**

arrive[x] = cont; **//记录到达时刻在列表中的位置**

wh[now] = x; **//记录该时间进入的元素**

vist[x] = 1;

for(int p = t[x]; p; p = edge[p].nxt)

{

int y = edge[p].to;

if(vist[xx] == 0)

{

dfs(xx);

rmin[++cont][0] = def[x]; **//每次访问子节点后，将父节点计入列表**

}

}

}

void rmq() **//预处理区间最小值**

{

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

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

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

}

int find(int l, int r)

{

int k,tp;

l = arrive[l]; r = arrive[r];

**//l与r在列表中位置所形成的区间中最小值即为LCA**

if(l > r) swap(l, r);

k = (int)(log((r - l + 1)\*1.0) / log(2.0));

**//这里可以预处理log**

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

}

### Tarjan法

int find(int x)

{

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

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

return f[x];

}

void dfs(int x,int fa)

{

for(int i = t[x]; i; i = edge[i].nxt)

{

int y = edge[i].to;

if(y != fa){ dfs(y,x); f[y] = x;}**//访问子节点后做合并操作**

}

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

{

int y = node[x][i].to;

if(f[y]) ans[node[x][i].id] = find(y);**//更新有关该点的答案**

}

}

int main()

{

int n, m, i, x, y, rt;

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

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

{

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

addedge(x, y);

addedge(y, x);

}

for(int i = 1; i <= m; i++)**//将询问离线**

{

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

node[x].push\_back(NODE(y,i));

node[y].push\_back(NODE(x,i));

}

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

dfs(rt, 0);

for(int i = 1; i <= m; i++) printf("%d\n", ans[i]);

return 0;

}

## 最小斯坦纳树

给定一张图，给k个关键点，问使这k个点联通所需要的最小边权和是多少？

时间复杂度

#include <iostream>

#include <cstdio>

#include <queue>

#include <cstring>

using namespace std;

typedef long long ll;

typedef pair<ll , int> pli;

const int N = 121 , M = 1010 , SS = 1 << 10 , INF = 1e9;

class Edges{

public:

int next , to;

int v;

} edge[M];

int first[N];

ll dis[N]; bool vis[N];

ll dp[N][SS];

queue <int> Q;

int n , m , te , k;

void add(int s , int t , int v){

++ te; edge[te].to = t; edge[te].next = first[s]; edge[te].v = v; first[s] = te;

}

inline ll Min(ll x1, ll x2){

if(x1 == -1) return x2;

else if(x2 == -1) return x1;

return min(x1 , x2);

}

void spfa(int S){

for(int G = ((S - 1) & S) ; G ; G = ((G - 1) & S)){

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

for(int e = first[i] ; e ; e = edge[e].next){

int to = edge[e].to;

if(dp[i][G] != -1 && dp[to][S ^ G] != -1)

dp[i][S] = Min(dp[i][S] , dp[i][G] + dp[to][S^G] + edge[e].v);

}

}

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

dis[i] = dp[i][S];

if(dis[i] != -1){

vis[i] = 1; Q.push(i); }

}

while(!Q.empty()){

int v = Q.front(); Q.pop(); vis[v] = 0;

for(int e = first[v] ; e ; e = edge[e].next){

int to = edge[e].to;

if(dis[to] == -1 || dis[to] > dis[v] + edge[e].v){

dis[to] = dis[v] + edge[e].v;

if(!vis[to])

Q.push(to);

}

}

}

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

dp[i][S] = dis[i];

}

int main(){

memset(dp , -1 , sizeof dp);

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

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

int u , v , w;

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

add(u , v , w);

add(v , u , w);

}

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

int x;

scanf("%d" , &x);

dp[x][1 << i] = 0;

}

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

spfa(i);

ll ans = -1;

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

ans = Min(ans , dp[i][(1 << k) - 1]);

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

return 0;

}

## 匹配问题

### 二分图最大匹配

int vispipei[maxn],match[maxn];

int dfspipei(int now)

{

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

if(dis[now][i])

{

int nex=i;

if(!vispipei[nex])

{

vispipei[nex]=1;

if(match[nex]==0 || dfspipei(match[nex]))

{

match[nex]=now;

return true;

}

}

}

return false;

}

int solve()

{

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

match[i]=0;

int cnt=0;

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

{

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

vispipei[j]=0;

if(!dfspipei(i))

cnt++;

}

return cnt;

}

**使用的时候千万要注意把return写够！以及match和vis数组的清空！**

#### 性质

**最小点覆盖=最大匹配**

**最小路径覆盖=∣V∣-最大匹配**

**最大独立集=∣V∣-最大匹配。最大独立集S 与 最小覆盖集T 互补**

#### HK

BFS

**首先把二分图染成黑色的点加到队列里。**

**对于每一个队列里的点u，与它相连的点v,如果v没被访问过，首先深度设为dep[u]+1**

**如果没被匹配，则找到了一条增广路，对vv就什么都不做最后return true**

**否则将与vv匹配的点，加到队列里。**

DFS

**与匈牙利的类似，只不过要求dep[v] 必须是dep[u]+1**

### 二分图最优匹配

int nn;

int wx[maxn],wy[maxn],weight[maxn][maxn],st[maxn];

int bel[maxn],visx[maxn],visy[maxn];

int dfs(int u)

{

visx[u]=1;

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

{

if(visy[v]) continue;

int t=wx[u]+wy[v]-weight[u][v];

if(!t)

{

visy[v]=1;

if(bel[v]==-1||dfs(bel[v]))

{

bel[v]=u;

return 1;

}

}

else if(st[v]>t)

st[v]=t;

}

return 0;

}

int km()

{

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

wx[i]=-inf,bel[i]=-1,wy[i]=0;

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

{

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

{

wx[i]=max(wx[i],weight[i][j]);

}

}

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

{

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

{

st[j]=inf;

}

while(1)

{

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

visx[j]=visy[j]=0;

if(dfs(i)) break;

int ret=inf;

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

if(!visy[j]&&ret>st[j])

ret=st[j];

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

if(visx[j])

wx[j]-=ret;

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

{

if(visy[j])

wy[j]+=ret;

else

st[j]-=ret;

}

}

}

int ans=0;

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

if(~bel[i])

ans+=weight[bel[i]][i];

return ans;

}

int solve()

{

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

{

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

{

weight[i][j]=-inf;

}

}

getweight();

return km();

}

**两个模板如果多次使用都要注意清空！二分图有最优匹配首先得有最大匹配!**

### 一般图最大匹配

### 一般图最优匹配

## 差分约束系统

## 线性规划

## 最小树形图

## 度数序列生成图

### 判定

一个度数序列可以生成一张图当且仅当

其中是度数序列，时间复杂度

### 构造

构造采用贪心法，每次选度数最大的点，向度数较大的前个点连边，若不能连则一定不能构造，时间复杂度

## 欧拉回路

### 有向图欧拉回路

欧拉回路判定：连通，所有点出度等于入度

欧拉路径判定：连通，除两个点度数为+1和-1，其余所有点出度等于入度

欧拉路径需要将两个特殊点连接在一起，转化为欧拉回路来做。

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int N = 1e5 + 10 , M = 4e5 + 10;

class Edges{

public:

int next , to;

int id;

};

Edges edge[M];

int first[N] , sum = 0;

int degree[N] , ans[M];

bool visit[M];

int point[M] , edges[M]; **//point是答案点序列，edges是答案边序列**

int te = 1;

int n , m;

void dfs(int p , int num){

for(int &e = first[p] ; e ; e = edge[e].next)

if(!visit[e]){

visit[e] = 1;

dfs(edge[e].to , e);

}

++ sum;

point[sum] = p; edges[sum] = edge[num].id;

}

void add(int s , int t , int id){

++ te;

edge[te].to = t;

edge[te].next = first[s];

edge[te].id = id;

first[s] = te;

degree[s] ++; degree[t] --;

}

int main(){

ios::sync\_with\_stdio(false);

cin >> n >> m;

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

int s , t;

cin >> s >> t;

add(s , t , i);

}

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

if(degree[i] != 0){

cout << "NO" << endl;

return 0;

}

dfs(edge[2].to , 0); **//边从2开始编号**

sum --;

if(sum != m){

cout << "NO" << endl;

return 0;

}

cout << "YES" << endl;

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

cout << edges[i] << " ";

cout << endl;

return 0;

}

### 无向图欧拉回路

欧拉回路判定：连通，度数都是偶数

欧拉路径判定：连通，除两个点度数是奇数

欧拉路径需要将两个特殊点连接在一起，转化为欧拉回路来做。

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int N = 1e5 + 10 , M = 4e5 + 10;

class Edges{

public:

int next , to;

int id;

};

Edges edge[M];

int first[N] , sum = 0;

int degree[N] , ans[M];

bool visit[M];

int point[M] , edges[M]; **//point是答案点序列，edges是答案边序列**

int te = 1;

int n , m;

void dfs(int p , int num){

for(int &e = first[p] ; e ; e = edge[e].next)

if(!visit[e]){

visit[e] = visit[e ^ 1] = 1;

dfs(edge[e].to , e);

}

++ sum;

point[sum] = p; edges[sum] = edge[num].id;

}

void add(int s , int t , int id){

++ te;

edge[te].to = t;

edge[te].next = first[s];

edge[te].id = id;

first[s] = te;

}

int main(){

ios::sync\_with\_stdio(false);

int x;

cin >> n >> m;

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

int s , t;

cin >> s >> t;

degree[s] ++; degree[t] ++;

add(s , t , i); **//正向边编号为正**

add(t , s , -i); **//负向边编号为负**

}

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

if(degree[i] & 1){

cout << "NO" << endl;

return 0;

}

dfs(edge[2].to , 0);

sum --;

if(sum != m){

cout << "NO" << endl;

return 0;

}

cout << "YES" << endl;

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

cout << edges[i] << " ";

cout << endl;

return 0;

}

### 一笔画问题

一笔画问题，需要先判定度数。

对于每个连通块，需要把笔画加起来，每个连通块内部答案是max(1,奇数点个数/2)

构造时，需要先把所有奇数点两两随机组合，之后每个连通块跑无向图的欧拉回路，最后要把你强制加的边删除。

以上算法时间复杂度都是的。

### fluery算法

### 计数问题

## 虚树

### 适用范围

虚树常常用来处理**询问的点数**远小于处理询问要经过的**树上的点数**的一类问题。

### 简单做法

把所有点和他们任意两个的LCA求出来，将它们按照父子关系构建一棵树，

**还要想好两两之间的边权如何处理**

### 构建方法

首先把所有询问的点按照在原树的序排序，我们还需要一个栈来维护从**根到当前节点**的链。

把号点加进去，为了保证3不会把栈弹空

每枚举到一个点，如果栈为空，直接入栈，否则求它和栈顶的。 对于这个和栈顶元素的序有下面三种关系：

，说明栈顶就是，直接把当前元素入栈。

，由于我们维护的是一个链，就可以让栈顶出栈，入栈

*C*.否则，一直到出现上面的情况

加完加最后再加枚举到的点。

至于连边，显然一个点出去的时候，它只会和上面的那个点连边，所以弹出的时候要连下边，最后如果栈不为空也要连边。

在实践中其实不需要判断序，只需要判断深度就可以辣。

### 代码

sort(a+1,a+1+m,cmp);   
**if**(!bel[1]) st[++top]=1;  
int temp;  
**for**(int i=1;i<=m;i++)  
{  
 **if**(!top)  
 {  
 st[++top]=a[i];  
 **continue**;  
 }  
 temp=lca(a[i],st[top]);  
 **while**(1)  
 {  
 **if**(dep[st[top-1]]<=dep[temp])  
 {  
 **if**(temp!=st[top])  
 add(temp,st[top]);   
 top--;  
 **if**(st[top]!=temp) st[++top]=temp;  
 **break**;  
 }  
 add(st[top-1],st[top]);  
 top--;  
 }  
 **if**(st[top]!=a[i]) st[++top]=a[i];  
}  
**while**(top>1) add(st[top-1],st[top]),top--;

虚树上的重要结论：**每次求本质上是求和上一个要添加的元素的，即不会出现求和的**

## 从编码到各种树的计数

### 编码

将**无根树**进行编码的一种方式，无根树就是边没有方向的树，即以树上的每一个点做根，只算一种方案。

#### 无根树转序列

每次找到编号最小的叶子，输出它父亲的编号，然后删除，直到只剩2个节点。

#### 序列转无根树

设集合

第次，找到在中没有在编码第到位出现的最小的数，将它与编码的第位连边，并把它从S中删除。

最后剩下的两个数，它们两个连一条边即可。

代码实现的时候，可以将每个点的度数初始化为序列中出现的次数+1，每次找度数为1最小的那一个与序列当前位连边。

### 有标号无根树的计数(公式)

由上面的做法显然可以得出，每一个长度为n-2的序列，唯一的对应了一棵无根树，且一颗无根树唯一的对应了一个序列。

因此可以得到n个点的有标号无根树总共有个。

#### 性质与应用

在编码中，每个点出现的次数为。

如果限制n个点的读数为，那么这样的无根树共有

如果只限制了k个呢，编码剩了m个位置呢？那么假装限制了，再乘以随便填的方案数,即

### 有标号有根树的计数

编码是无根树，但是树形均不同，那么每个生成的树以每个点做根均可，因此共有

### 无标号有根树的计数

**方法1朴素形式**，设为树大小为n时的答案。设对于根，大小为k的子树共有个，而每个子树都可以选择的一种方案，即,该方程非负整数解个数为()。

**方法1高级形式**经过复杂的母函数推导，我们得到

$$令s\_{n,j}=\sum\_{1\leq i \leq \frac n j}a\_{n+1-i\*j},则有s\_{n,j}=s\_{n-j,j}+a\_{n+1-j}\\ 则有a\_{n+1}=\frac {\sum\_{1\leq j \leq n}ja\_js\_{n,j}} {n}$$

**方法2朴素形式**如果我们把定义改为表示树大小为n，根节点度数为j的方案数，那么,就有

**方法2高级形式**我们考虑对进行，可以通过限制最大子树进行背包，即从小到大枚举，则有

这样有一个好处就是可以限制最大度数，如构造烷烃。

### 无标号无根树的计数

利用一下上题的

**方法1**，设为本题的答案，根据质心的关系，可以得到

$$n为奇数时\\ b\_n=a\_n-\sum\_{0<i\leq \lfloor\frac n 2\rfloor}a\_ia\_{n-i}\\ n为偶数时\\ b\_n=a\_n-\sum\_{0<i\leq n}a\_ia\_{n-i}+\frac 1 2 a\_{\frac n 2}(a\_{\frac n 2}+1)$$

**方法2**，可以通过枚举重心的方法来解决，强制，则保证是是以重心为根的无标号有根树计数，但如果n为偶数，可能会有两个重心，这两个重心子树大小都是，因此最终结果为

烷烃例题 [BZOJ4271]chemistry，方法2(没有加高精度)代码如下

**//无高精度版本**  
#include<iostream>  
#include<stdio.h>  
#include<string.h>  
#include<algorithm>  
#define maxn 505  
**using** **namespace** std;  
**typedef** long long ll;  
int n,m;  
ll dp[maxn][maxn];  
ll c(ll n,ll m)  
{  
 ll ans=1;  
 **for**(int i=m;i>=1;i--)  
 {  
 ans=ans\*n;  
 n--;  
 }  
 **for**(int i=m;i>1;i--)  
 {  
 ans/=i;  
 }  
 **return** ans;  
}  
int main()  
{  
 dp[1][0]=1;  
 scanf("%d",&n); m=4;  
 **for**(int maxson=1;maxson<((n+1)/2);maxson++)  
 {  
 ll a=0;  
 **for**(int j=0;j<m;j++)   
 a+=dp[maxson][j];  
 **for**(int i=n;i>maxson;i--)  
 {  
 **for**(int j=1;j<=m;j++)  
 {  
 **for**(int k=1;k<=j&&maxson\*k<i;k++)  
 {  
 dp[i][j]+=dp[i-maxson\*k][j-k]\*c(a+k-1,k);  
 }  
 }  
 }  
 }  
 ll ans=0;  
 **for**(int j=0;j<=m;j++)  
 ans+=dp[n][j];  
 **if**(!(n&1))  
 {  
 ll a=0;  
 **for**(int j=0;j<m;j++)  
 a+=dp[n/2][j];  
 ans+=c(a+1,2);  
 }  
 printf("%lld\n",ans);  
}

## 从哈密顿路到竞赛图(tournament)相关

### 哈密顿路

设一无向图有 n 个顶点，u、v 为图中任意不相邻的两点，deg(x) 代表 x 的度数 ，若成立，则存在哈密顿通路

#### 判定（Dirac定理）

设一个无向图中有 N 个节点，若所有节点的度数都大于等于 N/2，则汉密尔顿回路一定存在。注意，“N/2” 中的除法不是整除，而是实数除法。如果 N 是偶数，当然没有歧义；如果 N 是奇数，则该条件中的 “N/2” 等价于 “⌈N/2⌉”，可以根据此来构造哈密顿回路。

### 竞赛图

完全图给每个边确定了方向

#### 判定

竞赛图每一个点的出度从小到大排序之后，称为比分序列

对于一个长度为n的序列S是一个合法的比分序列，当且仅当

#### 性质

要么没环，要么有环，有环则必然有一个三元环，且一定不会有二元环。（三个点三个点分析即可）

任何竞赛图都有哈密顿路径

竞赛图的强连通块存在一个哈密顿回路

一个竞赛图缩点后，是一个类似链状的东西，由第一个点连向它每一个后面的点

竞赛图里, 大小为 n>1的强连通块中, 大小为[3,n]的简单环均存在

注意到竞赛图缩点之后是条链，那么链的最后一个点是没有出边的，而且这些点是按出度排好的，所以我们如果反过来，按出度排序，如果有一个x满足就说明这些点是最后一个SCC，如果x≠n就说明这个图不强连通，否则一定强连通。

## 联通分量

### 强连通分量

void tarjan(int x)

{

dfn[x] = ++cnt;

low[x] = cnt;

in[x] = 1; //这个很重要

stak[++stop] = x;

for(int p = t[x]; p; p = edge[p].nxt)

{

int y = edge[p].to;

if(!dfn[y])

{

tarjan(y);

low[x] = min(low[y], low[x]);

}

else

**if(in[y])//一定要有这个判断**

low[x] = min(dfn[y], low[x]);

}

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

{

int y;

do{

y = st[stop--];

col[y] = x;

in[y] = 0;

}while(y != x);

}

}

### 点双联通分量

void tarjan(int x, int from)

{

dfn[x] = ++cnt;

low[x] = cnt;

in[x] = 1;

st[++stop] = x;

for(int p = t[0][x]; p; p = edge[0][p].nxt)

{

int y = edge[0][p].to;

if(!dfn[y])

{

tarjan(y, p);

low[x] = min(low[x], low[y]);

}

else

if(p != (from ^ 1))

low[x] = min(low[x], dfn[y]);

//注意加边的时候从1开始加以保证反边为^1得到的结果

}

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

{

int y;

do{

y = st[stop--];

col[y] = x;

in[y] = 0;

}while(y != x);

}

}

## dsu on tree

1. dsu on tree解决的是什么问题呢？
   1. **不带修改**
   2. 暴力做法是通过枚举一个点，然后枚举这个点的子树来**统计信息**来解决
   3. 例如求每个点的子树里出现了多少种颜色，出现次数最多的颜色等。
2. 复杂度证明？
   1. 由树链剖分的性质，任意一个节点到根节点的路径上轻边的数量不超过 条 ，每个节点的信息也就会被统计那么 次，因此为。
3. 代码
   1. *//cf600E*  
      #include<iostream>  
      #include<stdio.h>  
      #include<algorithm>  
      #include<vector>  
      #include<stdlib.h>  
      #include<limits.h>  
      #include<queue>  
      #include<string.h>  
      #include<map>  
      #include<set>  
      #define maxn 100005  
      #define lson (now<<1)  
      #define rson ((now<<1)|1)  
      #define mid ((nl+nr)>>1)  
      using namespace std;  
      typedef long long ll;  
      int n,tot;  
      int c[maxn];  
      int head[maxn],nex[maxn<<1],to[maxn<<1];  
      void add(int x,int y)  
      {  
       to[++tot]=y; nex[tot]=head[x]; head[x]=tot;  
      }  
      int sz[maxn],son[maxn];  
      void dfs1(int now,int fa)  
      {  
       sz[now]=1;  
       for(int i=head[now];i;i=nex[i])  
       {  
       if(to[i]!=fa)  
       {  
       dfs1(to[i],now);  
       if(sz[to[i]]>sz[son[now]])  
       son[now]=to[i];  
       sz[now]+=sz[to[i]];  
       }  
       }  
      }  
      ll sum=0;   
      int maxx=0;  
      ll vis[maxn],ans[maxn],cnt[maxn];  
      void dfs(int now,int fa,int val)  
      {  
       cnt[c[now]]+=val;  
       if(val>0 && cnt[c[now]]>=maxx)  
       {  
       if(cnt[c[now]]>maxx)  
       sum=0,maxx=cnt[c[now]];  
       sum+=c[now];  
       }  
       for(int i=head[now];i;i=nex[i])  
       {  
       if(to[i]!=fa && !vis[to[i]])  
       {  
       dfs(to[i],now,val);  
       }  
       }  
      }  
      void dfs2(int now,int fa,int isson)  
      {  
       for(int i=head[now];i;i=nex[i])  
       {  
       if(to[i]!=fa && to[i]!=son[now])  
       {  
       dfs2(to[i],now,0);  
       }  
       }  
       if(son[now])  
       {  
       dfs2(son[now],now,1);  
       vis[son[now]]=1;  
       }  
       dfs(now,fa,1); ans[now]=sum;  
       if(son[now])  
       vis[son[now]]=0;  
       if(!isson)  
       {  
       dfs(now,fa,-1);  
       maxx=sum=0;  
       }  
      }  
      int main()  
      {  
       scanf("%d",&n);  
       for(int i=1;i<=n;i++)  
       scanf("%d",&c[i]);  
       for(int i=1;i<n;i++)  
       {  
       int x,y;  
       scanf("%d%d",&x,&y);  
       add(x,y);  
       add(y,x);  
       }  
       dfs1(1,-1);  
       dfs2(1,-1,0);  
       for(int i=1;i<=n;i++)  
       {  
       printf("%lld%c",ans[i],(i!=n ? ' ' : '\n'));  
       }  
      }

## 拓扑排序

### 一般情况

#include<iostream>  
#include<stdio.h>  
#include<string.h>  
#include<algorithm>  
#define maxn 100005  
using namespace std;  
int n,m,tot;  
int deg[maxn];  
int q[maxn];  
int head[maxn],nex[maxn],to[maxn];  
void add(int x,int y)  
{  
 to[++tot]=y;nex[tot]=head[x];head[x]=tot;  
}  
int main()  
{  
 scanf("%d%d",&n,&m);  
 for(int i=1;i<=m;i++)  
 {  
 int x,y;  
 scanf("%d%d",&x,&y);  
 add(x,y);  
 deg[y]++;  
 }  
 int hd=0,tl=-1;  
 for(int i=1;i<=n;i++)  
 if(!deg[i])  
 q[++tl]=i;  
 while(hd<=tl)  
 {  
 int now=q[hd]; hd++;  
 for(int i=head[now];i;i=nex[i])  
 {  
 deg[to[i]]--;  
 if(!deg[to[i]])  
 q[++tl]=to[i];  
 }  
 }  
 for(int i=0;i<n;i++)  
 printf("%d ",q[i]);  
}

### 求字典序最小解

正向建图，直接贪心取最小即可。

### 求小的数尽量靠前的解

**反向建图，**让**编号大**的尽量靠**前，**然后序列倒过来

## 2-sat

### 1. 具体实现

**1. a和b至少得选一个为例**，

我们对每个点拆成两个点 a1,a2，b1,b2表示对应的点选或者不选，那么要是我们a没有选，那么b就必须选了，所以从a2向b1连边即可。 b同理。

**2. a必选**

那么我们从a2向a1连边，由于一个点选和不选不能在同一个集合，所以我们肯定不会选a2,这样就可以满足了。

### 2. 输出一组方案

1. 显然这是一个依赖关系，中间可能存在环，所以我们先要缩点，如果一个点选和不选都缩到一个点去了，说明不存在可行解，否则我们按照原图反向建图，这样依赖关系底层的点是随便选，我们随便一选，然后根据一个点选和不选只能选一个，在拓扑排序的时候递推即可。

2. 还有一种很妙的方法，由于我们缩点的顺序就是拓扑序，那么我们对于每个点，一个点代表选和不选的那个节点所在强联通分量集合字典序较小的即为原图拓扑序在后的那一个。贪心的选择这个即可。

3. 代码

#include<iostream>   
 #include<stdio.h>   
 #include<string.h>   
 #include<algorithm>   
 #include<vector>   
 #include<stack>   
 #include<queue>   
 #define maxn 200005   
 using namespace std;   
 stack<int>s;   
 int n,m,cnt,tot,num,ans,sum;   
 priority\_queue<int,vector<int>,greater<int> > q;   
 vector<int>edge1[maxn],edge2[maxn];   
 int pri[maxn],pos[maxn],deg[maxn];   
 int dfn[maxn],low[maxn],bel[maxn],vis[maxn];   
 int getnum()   
 {   
 int now=0;char ch=getchar();   
 while(ch<'0' || ch>'9') ch=getchar();   
 while('0'<=ch && ch<='9') {now=now\*10+ch-'0';ch=getchar();}now\*=2;   
 if(ch=='h') now++;   
 return now;   
 }   
 int myabs(int now)   
 {   
 return now>0 ? now : -now;   
 }   
 void add(int u,int v)   
 {   
 edge1[u].push\_back(v);   
 }   
 void tarjan(int now)   
 {   
 vis[now]=1;   
 s.push(now);   
 dfn[now]=low[now]=++cnt;   
 int len=edge1[now].size();   
 for(int i=0;i<len;i++)   
 {   
 int nex=edge1[now][i];   
 if(!dfn[nex])   
 {   
 tarjan(nex);   
 low[now]=min(low[now],low[nex]);   
 }   
 else if(vis[nex])   
 {   
 low[now]=min(low[now],dfn[nex]);   
 }   
 }   
 if(dfn[now]==low[now])   
 {   
 int nex;   
 num++;   
 do   
 {   
 nex=s.top();s.pop();   
 vis[nex]=0;   
 bel[nex]=num;   
 }while(nex!=now);   
 }   
 }   
 void toposort()   
 {   
 for(int i=1;i<=num;i++)   
 if(deg[i]==0)   
 q.push(i);   
 while(!q.empty())   
 {   
 int now=q.top();q.pop();   
 if(!pri[now]) pri[now]=1,pri[pos[now]]=-1;   
 int len=edge2[now].size();   
 for(int i=0;i<len;i++)   
 {   
 int nex=edge2[now][i];   
 deg[nex]--;   
 if(!deg[nex])   
 q.push(nex);   
 }   
 }   
 }   
 int check()   
 {   
 for(int i=0;i<2\*n;i+=2)   
 {   
 if(bel[i]==bel[i^1]) return false;   
 pos[bel[i]]=bel[i^1];   
 pos[bel[i^1]]=bel[i];   
 }   
 for(int i=0;i<2\*n;i++)   
 {   
 int len=edge1[i].size();   
 for(int j=0;j<len;j++)   
 {   
 int nex=edge1[i][j];   
 if(bel[i]!=bel[nex])   
 {   
 deg[bel[i]]++;   
 edge2[bel[nex]].push\_back(bel[i]);   
 }   
 }   
 }   
 toposort();   
 return true;   
 }   
 int main()   
 {   
 while(scanf("%d%d",&n,&m) && (n||m))   
 {   
 while(!s.empty()) s.pop();   
 while(!q.empty()) q.pop();   
 cnt=tot=num=ans=sum=0;   
 for(int i=0;i<=2\*n;i++)   
 {   
 edge1[i].clear();edge2[i].clear();   
 pri[i]=dfn[i]=low[i]=bel[i]=vis[i]=deg[i]=pos[i]=0;   
 }   
   
 for(int i=1;i<=m;i++)   
 {   
 int x,y;   
 x=getnum();y=getnum();   
 add(x,y^1);add(y,x^1);   
 }   
   
 add(0,1);   
   
 for(int i=0;i<2\*n;i++)   
 if(!dfn[i])   
 tarjan(i);   
   
 if(!check())   
 {   
 printf("bad luck\n");   
 continue;   
 }   
   
 for(int i=1;i<n;i++)   
 {   
 if(pri[bel[i\*2+1]]==-1) printf("%dh",i);   
 else printf("%dw",i);   
 if (i<n-1) printf(" ");else printf("\n");   
 }   
 }   
 }

### 3. 输出字典序最小解

* + - 1. 从小到大枚举每个点是否可行，如果它存在可行解，就把它钦定必选，后面跟着判断即可。
      2. 代码

#include<iostream>  
#include<stdio.h>  
#include<string.h>  
#include<algorithm>  
#include<vector>  
#define maxn 200005  
using namespace std;  
int n,m,cnt;  
vector<int>edge[maxn];  
int s[maxn],match[maxn];  
void add(int x,int y)  
{  
 edge[x].push\_back(y);  
}  
int dfs(int now)  
{  
 if(match[now^1]) return false;  
 if(match[now]) return true;  
 s[++cnt]=now;match[now]=true;  
 int len=edge[now].size();  
 for(int i=0;i<len;i++)  
 {  
 int nex=edge[now][i];  
 if(!dfs(nex)) return false;  
 }  
 return true;  
}  
int can()  
{  
 memset(match,0,sizeof(match));  
 for(int i=0;i<2\*n;i+=2)  
 {  
 int now=i,nex=i^1;  
 if(!match[now] && !match[nex])  
 {  
 cnt=0;  
 if(!dfs(now))  
 {  
 for(int j=1;j<=cnt;j++)  
 match[s[j]]=false;  
 if(!dfs(nex))  
 return false;  
 }   
 }  
 }  
 return true;  
}  
int main()  
{  
 while(~scanf("%d%d",&n,&m))  
 {  
 for(int i=0;i<2\*n;i++)  
 edge[i].clear();  
 for(int i=1;i<=m;i++)  
 {  
 int x,y;  
 scanf("%d%d",&x,&y);  
 x--;y--;  
 add(x,y^1);  
 add(y,x^1);  
 }  
 if(can())  
 {  
 for(int i=0;i<2\*n;i+=2)  
 {  
 if(match[i]) printf("%d\n",i+1);  
 else printf("%d\n",i+2);  
 }  
 }  
 else printf("NIE\n");   
 }  
}