图论

# 单源最短路的建图方式

* 边权非负 朴素Dijkstra(稠密图的时候效率高，边多)，堆优化版Dijkstr(稀疏图的时候效率高，点相对多)
* 有负权边 Bellman-ford(基本不用)，Spfa(主要用)

## 例题1 1129 热浪

除了起点和终点之外，其他的点都由 双向道路 连向至少两个其他的点，每条道路有路费，C 条边(a,b,w),由起点到终点的最短单源最短距离

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 2510,M = 6200 \* 10 + 10;  
int n,m,S,T;  
int h[N],e[M],ne[M],w[M],idx;  
int dist[N],q[N];//距离数组和循环队列,一个点可能进出多次,会爆N所以循环一下队列即可  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,ne[idx] = h[a],w[idx] = c,h[a] = idx ++;  
}  
void spfa()  
{  
 memset(dist,0x3f,sizeof dist);  
 dist[S] = 0;  
 int hh = 0,tt = 1;  
 q[0] = S,st[S] = 1;  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == tt) hh = 0;//循环队列  
 st[t] = 0;//出队清除  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] > dist[t] + w[i])  
 {  
 dist[j] = dist[t] + w[i];  
 if(!st[j])  
 {  
 q[tt++] = j;  
 if(tt == N)   
 tt = 0;  
 st[j] = 1;  
 }  
 }  
 }  
 }  
}  
int main()  
{  
 \_;  
 cin >> n >> m >> S >> T;  
 memset(h,-1,sizeof h);  
 for(int i=0;i<m;i++)  
 {  
 int a,b,c;  
 add(a,b,c),add(b,a,c);  
 }  
 spfa();  
 cout << dist[T] << endl;  
 return 0;  
}  
复制

## 例题2 1128 信使

一个哨所同时向相连的哨所一起传递消息,信件在哨所内停留的时间忽略不计,直至所有哨所都收到消息(由指挥部开始(在第一个哨所)传递) 核心:对于每个点来说,他接受到信的时间等于它到指挥部的最短距离

单源最短路问题,但是因为数据足够小,所以直接使用多源最短路floyd算法即可(n^3)，floyd一般用作多源上，但是他也可以用在单源上，只要数据范围较小便可以用在单源

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 110,INF = 0x3f3f3f3f;  
int n,m;  
int d[N][N];  
int main()  
{  
 \_;  
 cin >> n >> m;  
 memset(d,0x3f,sizeof d);  
 for(int i=0;i<m;i++)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 d[a][b] = d[b][a] = min(d[a][b],c);//重边保留最小边即可  
 }  
 for(int k=1;k<=n;k++)  
 for(int i=1;i<=n;i++)  
 for(int j=1;j<=n;j++)  
 d[i][j] = min(d[i][j],d[i][k] + d[k][j]);  
 int res = 0;  
 for(int i=1;i<=n;i++)  
 {  
 if(d[1][i] == INF)  
 {  
 res = -1;  
 break;//有点无法到达  
 }  
 else res = max(res,d[1][i]);  
 }  
 cout << res << endl;  
 return 0;  
}//4 4 1 2 4 2 3 7 2 4 1 3 4 6 -- > 11  
复制

## 例题3 1127 香甜的黄油

所有牧场到一个牧场距离和的最小值,答案一定存在,多源汇最短路问题(从多个节点到一个节点)floyd(n^3)超时,堆优化的Dijkstra算法(mlogn),spfa(m,最差mn)

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 810,M = 3000,INF = 0x3f3f3f3f ;//点数和边数  
int n,p,m;  
int id[N];//奶牛的编号  
int h[N],e[M],ne[M],w[M],idx;  
int dist[N],q[N];  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,ne[idx] = h[a],w[idx] = c,h[a] = idx ++;  
}  
int spfa(int start)  
{  
 memset(dist,0x3f,sizeof dist);  
 dist[start] = 0;  
 int hh = 0,tt = 1;//循环队列tt初始值为1  
 q[0] = start,st[start] = 1;  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == N) hh = 0;//一个点可能进出多次,防止爆N  
 st[t] = false;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] > dist[t] + w[i])  
 {  
 dist[j] = dist[t] + w[i];  
 if(!st[j])  
 {  
 q[tt++] = j;  
 if(tt == N) tt = 0;  
 st[j] = 1;  
 }  
 }  
 }  
 }  
 int res = 0;  
 for(int i=0;i<n;i++)  
 {  
 int j = id[i];  
 if(dist[j] == INF)//这个点位根节点到不了这个点,返回无解  
 return INF;  
 res += dist[j];  
 }  
 return res;  
}  
int main()  
{  
 \_;  
 cin >> n >> p >> m;//牛的个数,牧场个数,边的个数  
 for(int i=0;i<n;i++) cin >> id[i];//每头牛的编号  
 memset(h,-1,sizeof h);  
 for(int i=0;i<m;i++)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 add(a,b,c),add(b,a,c);  
 }  
 int res = INF;  
 for(int i=1;i<=p;i++)//p是牧场数  
 res = min(res,spfa(i));//spfa返回的是所有奶牛到他(i)的距离之和  
 cout << res << endl;  
 return 0;  
}//3 4 5 2 3 4 1 2 1 1 3 5 2 3 7 2 4 3 3 4 5 -- > 8  
复制

## 例题4 1126 最小花费

一个无向图,相连的人可以相互转账,手续费是权值z,求从A最少转多少钱可以使得B收到100,将手续费(百分数)转换为0到1之间的一个数,求最小乘积最大即可,每次取最大的数据相乘即可log(积) = log(a) + log(b),因为转换为log(x)的和,且0<=x<1,所以log一定是负数,,边权\*-1,从而转换为Dij最短路,此时边权非负,如果边权有正有负,只能用spfa来做,存在负权边只能使用spfa,Dijkstra只能处理正数,写的时候没必要按照分析的时候取log,直接取乘积的最大值即可,一般乘积最小边权都是正的,权重>=1则dijkstra或spfa都可,如果权重>=0(有负权边)spfa写

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 2010;  
int n,m,S,T;  
double g[N][N];//不必初始化,因为不管多少钱传过去都是0,最大值是0,等价于没有边  
double dist[N];//g存的是最大值  
bool st[N];  
void dijkstra()  
{  
 dist[S] = 1;  
 for(int i=1;i<=n;i++)  
 {  
 int t = -1;  
 for(int j=1;j<=n;j++)  
 {  
 if(!st[j] && (t == -1 || dist[t] < dist[j]))  
 t = j;  
 }  
 st[t] = 1;  
 for(int j=1;j<=n;j++)  
 dist[j] = max(dist[j],dist[t] \* g[t][j]);  
 }  
}  
int main()  
{  
 scanf("%d%d",&n,&m);  
 while(m --)  
 {  
 int a,b,c;  
 scanf("%d%d%d",&a,&b,&c);  
 double z = (100.0 - c) / 100.0;  
 g[a][b] = g[b][a] = max(g[a][b],z);  
 }  
 cin >> S >> T;  
 dijkstra();//dist求得是乘积的最大值,结果是100/dist  
 printf("%.8lf\n",100.0 / dist[T]);  
 return 0;  
}//3 3 1 2 1 2 3 2 1 3 3 1 3 -- > 103.07153164  
复制

## 例题5 920 最优乘车

巴士只能按照巴士路线依次经过若干个巴士站，最终到达终点站，一名旅客想去一个地点玩，但是如果没有直达的巴士，他只能换乘，求这个旅客换乘次数最小是多少，样例输入中第一行是数据范围，下面是一个完整的巴士路线 ---> 求坐车的最小次数 - 1 ，将每一条路线抽象成边，边内的所有点两两之间(注意方向)的距离都为 1 ，转化为最短路问题，即是最少乘车数量，边内是单向边，答案是 最少乘车数量 - 1 ， bfs即可，因为所有边的权重一样，且数据不大

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 510;//使用邻接矩阵来存边  
int n,m;  
bool g[N][N];//表示邻接矩阵有没有边  
int dsit[N];  
int stop[N];  
int q[N];//bfs 队列  
bool st[N];//判重  
void bfs()  
{  
 int hh = 0,tt = 0;  
 memset(dist,0x3f,sizeof dist);  
 q[0] = 1;  
 dist[1] = 0;  
  
 while(hh <= tt)  
 {  
 int t = q[hh ++];  
 for(int i=1;i<=n;i++)  
 {  
 if(g[t][i] && dist[i] > dist[t] + 1)  
 {  
 dist[i] = dist[t] + 1;  
 q[++ tt] = i;  
 }  
 }  
 }  
  
}  
int main()  
{  
 \_;  
 cin >> m >> n;  
 string line;  
 getline(cin,line);//因为不知道路线具体有多少边，所以使用字符串，且使用getline防止空格  
 while(m --)  
 {  
 getline(cin,line);  
 stringstream ssin(line);//相当于s\_scanf头文件sstream  
 int cnt = 0,p;//cnt表示站台数量，p 表示站台编号  
 while(ssin >> p) stop[cnt ++] = p;  
 for(int j=0;j<cnt;j++)  
 {  
 for(int k=j+1;k<cnt;k++)  
 {  
 g[stop[j]][stop[k]] = true;//边内从前到后两两点之间建单向边  
 }  
 }  
 }  
 bfs();  
 if(dist[n] = 0x3f3f3f3f) cout << "NO" << endl;  
 else cout << max(0,dist[n] - 1) << endl;  
 return 0;  
}//3 7 6 7 4 7 3 6 2 1 3 5 ---> 2  
复制

## 例题6 903 昂贵的聘礼

一个人要向一个目标人物换一个东西，但是这个东西太贵了，他可以通过部分钱另一个人的其他物品来获得，得到另一个人的物品同样有这两种情况，另外，这些人等级制度森严，如果他和某个地位较低的人进行了交易，地位较高的人就不会再和他交易，反之同理，且这个差距是有一定的限制的，不超过这个限制就能相互交易。这个人不会用多样物品换一样物品因为不会得到更低的价格，也即用物品加更低的价格换另一个人的物品，优惠并不会叠加，只能使用物加钱的一种。物品都有编号，价格 P，等级 L，替代品 T，优惠 Vi，等级差距限制是 M，计算最少需要多少金币才能娶到酋长的女儿。输入第一行数据范围，后面 P L n(代表多少种替代方式)下面 n 行 输入替代物品的信息,输入顺序即是物品编号

建图的时候加一个虚拟点，表示直接买，其他的可以间接买的时候，连接一条由物品交换到这个物品的单向边，权重是通过下一个物品交换时买那个物品的价格

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 110,INF = 0x3f3f3f3f;  
int n,m;  
int w[N][N],level[N];//边的距离(权重)和等级   
int dist[N];  
bool st[N];  
  
int dijkstra(int down,int up)  
{  
 memset(dist,0x3f,sizeof dist);  
 memset(st,0,sizeof st);  
 dist[0] = 0;  
 for(int i=1;i<=n+1;i++)//加了一个 0 号点，共 n + 1 次循环  
 {  
 int t = -1;  
 for(int j=0;j<=n;j++)//j 需要从0开始循环，因为虚拟点 0 号点应该也是包含在里面的  
 {  
 if(!st[j] && (t == -1 || dist[t] > dist[j]))  
 t = j;  
 }  
 st[t] = true;  
 for(int j=1;j<=n;j++)  
 {  
 if(level[j] >= down && level[j] <= up)//等级在范围内  
 dist[j] = min(dist[j],dist[t] + w[t][j]);  
 }   
 }  
 return dist[1];  
}  
  
int main()  
{  
 \_;  
 cin >> m >> n;  
 memset(w,0x3f,sizeof w);  
 for(int i=1;i<=n;i++)  
 w[i][i] = 0;  
 //虚拟原点定义为 0 号点，表示直接购买  
 for(int i=1;i<=n;i++)  
 {  
 int price,cnt;//价格和可替换的数量  
 cin >> price >> level[i] >> cnt;  
 w[0][i] = min(price,w[0][i]);//由虚拟 0 点向当前点连一条边，代表直接购买，并且判重  
 while(cnt --)//读入替代品，结构是， 编号 价格  
 {  
 int id,cost;  
 cin >> id >> cost;  
 w[id][i] = min(cost,w[id][i]);  
 }  
 }  
 int res = INF;  
 for(int i=max(0,level[1]-m);i<=level[1];i++)//枚举等级区间，因为有差距限制，所以一定要把 1 包含进去，获得下端点  
 {  
 res = min(res,dijkstra(i,i+m));//传入的是区间的范围  
 }  
 cout << res << endl;  
 return 0;  
}//1 4 10000 3 2 2 8000 3 5000 1000 2 1 4 200 3000 2 1 4 200 50 2 0 ---> 5250  
复制

# 单源最短路的综合应用

主要是最短路算法和其他算法的结合

## 例题 1 1135 新年好

最短路 + DFS，n 个车站，m 个双向公路连接，无重边，一个人按顺序拜访一些人，求时间最短是多少，拜访的人分别在若干个车站，、先爆搜在求每个点到其他五个点的单源最短路，可能会超时，优化：引入一个额外点 1 ，先做6次 spfa (O(n+m)) 或者Dijkstra算法(O(mlogn))，求分别到其他点的最短距离，然后在 DFS 所有拜访顺序 5！，对于每一种拜访顺序，可以通过查表的方式算出最短距离 6 \* O(m) + 5!

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 50010,M = 200010,INF = 0x3f3f3f3f;  
int n,m;  
int source[6];  
int h[N], e[M],w[M],ne[M],idx;  
int q[N],dist[6][N];//6 个点到其他的点的不同最短路  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx ++;  
}   
void spfa(int start,int dist[])//spfa 或者 dijkstra 算法都可以  
{  
 memset(dist,0x3f,N\*4);//初始化 N \* 4 是因为队列中存的点不止 N 个  
 dist[start] = 0;  
 //循环队列的方式写  
 int hh = 0,tt = 1;//循环队列 hh=0,tt=1,正常队列hh=0,tt=0 代表有一个元素  
 q[0] = start;  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == N) hh = 0;  
 st[t] = false;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] > dist[t] + w[i])  
 {  
 dist[j] = dist[t] + w[i];  
 if(!st[j])  
 {  
 q[tt ++] = j;  
 if(tt == N) tt = 0;  
 st[j] = true;  
 }  
 }  
 }  
 }   
}  
int dfs(int u,int start,int distance)  
{  
 if(u == 6) return distance;  
 int res = INF;  
 for(int i=1;i<=5;i++)  
 {  
 if(!st[i])  
 {  
 int next = source[i];  
 st[i] = true;  
 res = min(res,dfs(u + 1,i,distance + dist[start][next]));//当前距离加上走到下一个点的距离  
 st[i] = false;//回溯，恢复现场，表示下一次搜索的时候这个点未搜过  
 }  
 }  
 return res;  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 source[0] = 1;//额外加的点  
 memset(h,-1,sizeof h);  
 for(int i=1;i<=5;i++)  
 cin >> source[i];  
 while(m --)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 add(a,b,c),add(b,a,c);  
 }  
 for(int i=0;i<6;i++)  
 spfa(source[i],dist[i]);  
 cout << dfs(1,0,0) << endl;//拜访的第几个亲戚，当前起点，当前总费用（距离）  
}//6 6 2 3 4 5 6 1 2 8 2 3 3 3 4 4 4 5 5 5 6 2 1 6 7 ---> 21  
复制

## 例题 2 340通信线路

单源最短路 + 二分，N 座通信基站，P 条 双向 电缆，第 i 条连接Ai，Bi，其中 1 号是公司的总站，N 号是农场内，每条升级是 Li，农场主可以指定一条从1 到 N 的路径，在不超过 k 条电缆的前提下，这条中的前k 最大值免费更新，这条路剩余的电缆中，农场主只需要支付最贵的电缆即可，求总花费最少，若不存在通路，输出 -1，第 k + 1 大值，选中一条路径更新其中的电缆即可，不需要全部更新所有路径 二分思路：[0,1000001]区间内的某个点 x ，从 1 走到 N，最少经过的长度大于 x 的边的数量是否小于等于 k ，左端点 0 是为了防止所有的边都被删了，右端点是因为100000不能确定是结果是1000000，还是不连通 将所有的边分类，边长大于 x ，边权看做 1 ，否则边权看做0，双端队列BFS求1到N的最短路(权重为1 或 0)

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 1010,M = 20010;  
int n,m,k;  
int h[N],e[M],w[M],ne[M],idx;  
deque<int> q;  
int dist[N];//双端队列起的作用类似于堆，优先队列，模拟时间短  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,ne[idx] = h[a],w[idx] = c,h[a] = idx ++;  
}  
bool check(int bound)  
{  
 memset(st,0,sizeof st);  
 memset(dist,0x3f,sizeof dist);  
 dist[1] = 0;  
 q.push\_back(1);  
 while(q.size())  
 {  
 int t = q.front();  
 q.pop\_front();  
 if(st[t]) continue;  
 st[t] = true;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 int v = w[i] > bound;//大于边界是 1 ，小于边界是 0  
 if(dist[j] > dsit[t] + v)  
 {  
 dist[j] = dist[t] + v;  
 if(!v)   
 q.push\_front(j);//是 0 加到队头  
 else  
 q.push\_back(j);  
 }  
 }  
 }  
 return dist[n] <= k;  
}  
int main()  
{  
 \_;  
 cin >> n >> m >> k;  
 memset(h,-1,sizeof h);  
 while(m --)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 add(a,b,c),add(b,a,c);  
 }  
 int l = 0,r = 1e6 + 1;  
 while(l < r)  
 {  
 int mid = l + r >> 1;  
 if(check(mid)) r = mid;  
 else l = mid + 1;  
 }  
 if(r == 1e6 + 1) r = -1;  
 cout << r << endl;  
 return 0;  
}//5 7 1 1 2 5 3 1 4 2 4 8 3 2 3 5 2 9 3 4 7 4 5 6 ---> 4  
复制

# 最小生成树

* Prim 算法 ( O(n^2)邻接矩阵 )
* kruskal算法（常用） ( O(mlogn) 直接存边，结构体)

# 次小生成树

## 定义

给一个带权的图，把图的所有的生成树按权值从小到大排序，第二小的次小生成树

* 做法1：先求最小生成树，在枚举删去最小生成树中的边求解，再求一遍最小生成树 O(mlogn+nm)
* 做法2：先求最小生成树，然后依次枚举非树边，然后将该边加入树中，同时从树中去掉一条边，使得最终的图仍是一棵树。则一定可以求出次小生成树（设T为G的一棵生成树，对于非树边a和树边b，插入a，并删除b的操作记作(+a,-b)，如果T+a-b之后，仍是一棵生成树，称(+a,-b)是T的一个可行交换，则称T进行一次可行交换所得到的新的生成树集合称为T的 邻集） -----> 次小生成树一定在最小生成树的邻集中。

## 理论基础

1. 任意一课最小生成树一定 可以 包含无向图中权值最小的边
2. 给定一张无向图G=(V,E)，n = |V|，m = |E| (绝对值表示集合里的元素数量)，从 E 中选出 k < n-1 条边构成 G 的一个生成森林。若在剩余的 m - k 条边中选 n-1-k 条边添加到生成森林中，使其成为 G 的生成树，并且选出的边的权值之和最小。则该生成树一定 可以 包含 m - k 条边中连接生成森林的两个不连通节点的权值最小的边

### 思路

如何证明当前的这条边一定可以被选：假设不选当前边，最终得到了一棵树。然后将这棵边加上，那么必然会出现一个环，在这个环上一定可以找出一条长度不小于当前边的边，那么把当前边替换上去，结果一定不会变差。

# 典型应用

## 例题 1 1140 最短网络

将光纤在各个农场间连接起来，是的光纤的长度最短，输入为邻接矩阵，使用Prim算法较为方便

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long  
const int N = 110;  
int n;  
int w[N][N];  
int dist[N];  
bool st[N];  
  
int prim()  
{  
 int res = 0;  
 memset(dist,0x3f,sizeof dist);  
 dist[1] = 0;  
 for(int i=0;i<n;i++)  
 {  
 int t = -1;  
 for(int j=1;j<=n;j++)  
 {  
 if(!st[j] && (t == -1 || dist[t] > dist[j]))  
 t = j;  
 }  
 res += dist[t];  
 st[t] = true;  
 for(int j=1;j<=n;j++)  
 {  
 dist[j] = min(dist[j],w[t][j]);  
 }  
 }  
 return res;  
}  
  
int main()  
{  
 \_;  
 for(int i=1;i<=n;i++)  
 {  
 for(int j=1;j<=n;j++)  
 {  
 cin >> w[i][j];  
 }  
 }  
 cout << prim() << endl;  
 return 0;  
}  
复制

## 例题 2 1141 局域网

?无重边和自环，f(i,j)表示 i 和 j 之间的畅通程度，越小越畅通，为 0 表示未连通，求删除一些网线使得网络中没有回路，并且 f 的总和最大，并且保证不影响连通性，即是图仍然要是联通的 ---> 相当于在这个图的每一个连通块内求一棵生成树。多连通块使用Kruskal算法比较方便，输出的是删除的边的 f 和

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 110,M = 210;  
int n,m;  
struct Edge  
{  
 int a,b,w;  
 bool operator<(const Edge &t)const  
 {  
 return w < t.w;  
 }  
}e[M];  
int p[N];  
int find(int x)  
{  
 if(x != p[x])  
 p[x] = find(p[x]);  
 return p[x];  
}  
int main()  
{  
 cin >> n >> m;  
 for(int i=1;i<=n;i++)   
 p[i] = i;  
 for(int i=0;i<m;i++)  
 {  
 int a,b,w;  
 cin >> a >> b >> w;  
 e[i] = {a,b,w};  
 }  
 sort(e,e + m);  
 int res = 0;  
 for(int i=0;i<m;i++)  
 {  
 int a = find(e[i].a),b = find(e[i].b),w = e[i].w;  
 if(a != b)  
 {  
 p[a] = b;  
 }  
 else res += w;  
 }  
 cout << res << endl;  
 return 0;  
}//5 5 1 2 8 1 3 1 1 5 3 2 4 5 3 4 2 ----> 8  
复制

## 例题 3 1142 繁忙的都市

有 n 哥交叉路口，有些交叉口之间有路相连，双向边，无重边，道路的分值越小表示道路越繁忙，要求：全联通，改造数尽可能小，改造分值的最大值尽可能小，依次满足，输出改造了几条道路和最大的分值是多少，本题之中的最小生成树是最大的边权最小

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 310,M = 10010;  
int n,m;  
struct Edge  
{  
 int a,b,w;  
 bool operator<(const Edge &t) const  
 {  
 return w < t.w;  
 }  
}e[M];  
int p[N];  
int find(int x)  
{  
 if(x != p[x])  
 p[x] = find(p[x]);  
 return p[x];  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 for(int i=1;i<=n;i++) p[i] = i;  
 for(int i=0;i<m;i++)  
 {  
 int a,b,w;  
 cin >> a >> b >> w;  
 e[i] = {a,b,w};  
 }  
 sort(e,e + m);  
 int res = 0;//res存最后一条边  
 for(int i=0;i<m;i++)  
 {  
 int a = find(e[i].a),b = find(e[i].b),w = e[i].w;  
 if(a != b)  
 {  
 p[a] = b;  
 res = w;//因为是sort后的结果所以最后一个进去的边一定是最大分值  
 }  
 }  
 cout << n - 1 << " " << res << endl;  
}//4 5 1 2 3 1 4 5 2 4 7 2 3 6 3 4 8 ---> 3 6  
复制

## 例题 4 1143 联络员

tyvj 之间存在无向边，边分为两大类一类为必选边，一类为可选择性，求使得图连通的花费最小，注意：u 和 v 之间可能存在多条通信道路，计算的结果要累加所有。n 个管理员，m 个通信渠道，直接先把必选边加到并查集里面，将剩余的边进行kruskal算法,缩点本题直接将祖宗节点看做连通块的代表点即可

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 2010,M = 10010;  
int n,m;  
struct Edge  
{  
 int a,b,w;  
 cin >> a >> b >> w;  
 bool operator<(const Edge &t)const  
 {  
 return w < t.w;  
 }  
}e[M];  
int p[N];  
int find(int x)  
{  
 if(x != p[x])  
 p[x] = find(p[x]);  
 return p[x];  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 for(int i=1;i<=n;i++)  
 p[i] = i;  
 int res = 0;  
 int k =0;  
 for(int i=0;i<m;i++)  
 {  
 int a,b,w,t;  
 cin >> t >> a >> b >> w;  
 if(t == 1)//必选边  
 {  
 res += w;  
 p[find(a)] = find(b);  
 }  
 else   
 e[k ++] = {a,b,w};  
 }  
 sort(e,e+k);  
 for(int i=0;i<k;i++)  
 {  
 int a = find(e[i].a),b = find(e[i].b),w = e[i].w;  
 if(a != b)  
 {  
 p[a] = b;  
 res += w;  
 }  
 }  
 cout << res << endl;  
 return 0;  
}//5 6 1 1 2 1 1 2 3 1 1 3 4 1 1 4 1 1 2 2 5 10 2 2 5 5 ---> 9  
复制

## 例题 5 1144 连接格点

m 行 n 列的点阵，相邻两点可以相连，纵向的每条花费一个单位，横向的花费两个单位，输入x1，y1，x2，y2表示第 x1 行第 y1 列的点和第 x2 行第 y2 列的点已经连线了，且有|x1-x2| + |y1-y2| = 1,求使得联通的最小花费

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 1010,M = N \* N,K = 2 \* N \* N;//M表示点数 K表示边数  
int n,m;  
int ids[N][N];  
struct Edge  
{  
 int a,b,w;  
}e[K];//不需要排序，因为只有1 或 2 直接分开存储即可，优化时间  
int p[M];  
int find(int x)  
{  
 if(x != p[x])  
 p[x] = find(p[x]);  
 return p[x];  
}  
int k;  
void get\_edges()  
{  
 int dx[4] = {-1,0,1,0},dy[4] = {0,1,0,-1},dw[4] = {1,2,1,2};  
 for(int z=0;z<2;z++)//两种边  
 {  
 for(int i=1;i<=n;i++)  
 {  
 for(int j=1;j<=m;j++)  
 {  
 for(int u=0;u<4;u++)  
 {  
 if(u % 2 == z)//横向边或者纵向变，实现先枚举上下再枚举左右，直接实现了排序的功能，纵向权重为 1 ，横向为 2 先纵后横直接实现排序  
 {  
 int x = i + dx[u],y = j + dy[u],w = dw[u];  
 if(x && x <= n && y && y <= m)  
 {  
 int a = ids[i][j],b = ids[x][y];  
 if(a < b)//因为将二维坐标映射为一维，从前往后都是递增，所以当a<b的时候，加一次边即可，防止重复  
 e[k ++] = {a,b,w};  
 }  
 }  
 }  
 }  
 }   
 }  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 for(int i=1;i<=n \* m;i++)  
 p[i] = i;  
 for(int i=1,t=1;i<=n;i++)//t = 1和 i = 1一样只会初始化一次，莫要混淆  
 for(int j=1;j<=m;j++,t++)  
 ids[i][j] = t;  
 int x1,y1,x2,y2;  
 while(cin >> x1 >> y1 >> x2 >> y2)  
 {  
 int a = ids[x1][y1],b = ids[x2][y2];//将二维坐标映射为一维坐标   
 p[find(a)] = find(b);  
 }  
 get\_edges();  
 //Kruskal  
 int res = 0;  
 for(int i=0;i<k;i++)  
 {  
 int a = find(e[i].a),b = find(e[i].b),w = e[i].w;  
 if(a != b)  
 {  
 p[a] = b;  
 res += w;  
 }  
 }  
 cout << res << endl;  
 return 0;  
}//2 2 1 1 2 1 ---> 3  
复制

## 最小生成树的扩展应用

## 例题 1 1146 新的开始

两种方式，一种在矿井 i 上建立一个发电站，费用是 Vi，或者将其与 j(j点有电力供应) 相连，费用是Pij，求最小花费

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 310;  
int n;  
int w[N][N];  
int dist[N];//最大边的长度,Dijk表示路径长度  
bool st[N];  
  
int prim()  
{  
 memset(dist,0x3f,sizeof dist);  
 dist[0] = 0;  
 int res = 0;  
 for(int i=0;i<n+1;i++)  
 {  
 int t = -1;  
 for(int j=0;j<=n;j++)  
 {  
 if(!st[j] && (t == -1 || dist[t] > dist[j]))  
 t = j;  
 }  
 st[t] = true;  
 res += dist[t];  
 for(int j=0;j<=n;j++)  
 dist[j] = min(dist[j],w[t][j]);  
 }  
 return res ;  
}  
  
int main()  
{  
 \_;  
 cin >> n;  
 for(int i=1;i<=n;i++)  
 {  
 cin >> w[0][i];  
 w[i][0] = w[0][i];  
 }  
 for(int i=1;i<=n;i++)  
 {  
 for(int j=1;j<=n;j++)  
 {  
 cin >> w[i][j];  
 }  
 }  
 cout << prim() << endl;  
 return 0;  
}//4 5 4 4 3 0 2 2 2 2 0 3 3 2 3 0 4 2 3 4 0 ---> 9  
复制

## 例题 2 1145 北极通讯网络

有两种通讯工具，一种是无线电收发机，不同的有不同的参数d ，表示建立通讯的村落之间距离不超过 d ，且d 越大价格越贵，选择时数量不限但是型号相同，第二种卫星设备，数量有限只能给一部分村庄配备，一共 k 台卫星设备，求如何分配使得无线电型号的d 最小 ---> 找一个最小的 d 值，使得将所有权值小于 d 的边删去后，整个图形的连通块的个数不超过 k，由于并查集本身的结果就是单调的，不必使用二分

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
#define PII pair<int,int>  
#define x first  
#define y second  
const int N = 510,M = N \* N / 2;//点数和边数  
int n,k,m;  
struct Edge   
{  
 int a,b;  
 double w;  
 bool operator<(const Edge &t)const  
 {  
 return w < t.w;  
 }  
}e[M];  
int p[N];  
PII q[N];  
double get\_dist(PII a,PII b)  
{  
 int dx = a.x - b.x;  
 int dy = a.y - b.y;  
 return sqrt(dx \* dx + dy \* dy);  
}  
int find(int x)  
{  
 if(x != p[x])  
 p[x] = find(p[x]);  
 return p[x];  
}  
int main()  
{  
 \_;  
 cin >> n >> k ;  
 for(int i=0;i<n;i++) cin >> q[i].x >> q[i].y;  
 for(int i=0;i<n;i++)  
 {  
 for(int j=0;j<i;j++)  
 {  
 e[m ++] = {i,j,get\_dist(q[i],q[j])};  
 }  
 }  
 sort(e,e + m);  
 for(int i=0;i<n;i++) p[i] = i;  
 int cnt = n;//连通块个数，如果d=0，需要 n 台卫星设备  
 double res = 0;//答案,即是 d  
 for(int i=0;i<m;i++)  
 {  
 if(cnt <= k) break;  
 int a = find(e[i].a),b = find(e[i].b);  
 double w = e[i].w;  
 if(a != b)  
 {//因为本身就是排序后的，每次操作后得到的res一定是最小的，当cnt<=k时，表示剩下的 d 较大的可以直接使用卫星设备操作，结束循环，输出res即可  
 p[a] = b;  
 cnt --;  
 res = w;  
 }  
 }  
 cout << res << endl;  
}  
复制

## 例题 3 346 走廊泼水节

有一棵N节点的树，要求增加若干条边扩充为完全图，并且最小生成树仍然是这棵树，求增加的边的权值总和是多少,所有的新边的权重取 Wi + 1 即可,读入的是 n-1 条边，完全图的定义是每对顶点之间都有且只有一条边

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 6010;  
int n;  
struct Edge  
{  
 int a,b,w;  
 bool operator<(const Edge &t)const  
 {  
 return w < t.w;  
 }  
}e[N];  
int p[N],size[N];  
int find(int x)  
{  
 if(x != p[x])  
 p[x] = find(p[x]);  
 return p[x];  
}  
int main()  
{  
 \_;  
 int t;  
 cin >> t;  
 while(t --)  
 {  
 cin >> n;  
 for(int i=0;i<n-1;i++)  
 {  
 int a,b,w;  
 cin >> a >> b >> w;  
 e[i] = {a,b,w};  
 }  
 sort(e,e + n - 1);  
 for(int i=1;i<=n;i++) p[i] = i,size[i] = 1;  
 int res = 0;  
 for(int i=0;i<n-1;i++)  
 {  
 int a = find(e[i].a),b = find(e[i].b),w = e[i].w;  
 if(a != b)  
 {  
 res += (size[a] \* size[b]- 1) \* (w + 1);//由于是完全图，所以要求每对顶点之间有且只有一条边  
 size[b] += size[a];  
 p[a] = b;  
 }  
 }  
 cout << res << endl;  
 }  
}  
复制

## 例题 4

先求最小生成树，同时标记每条边是树边还是非树边，建立最小生成树，然后通过DFSBFS预处理出来任意两点之间的边权最大值，然后枚举每条非树边，将非树边加入树中，去掉对应的环上最大值的树边得到新树，sum + w - dist[a][b]，加较小减最大得到大于最小生成树的最小树此题是严格次小，必须严格大于最小生成树

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 510,M = 10010;  
int n,m;  
struct Edge  
{  
 int a,b,w;  
 bool f;//表示是不是非树边  
 bool operator<(const Edge &t)const  
 {  
 return w < t.w;  
 }  
}edge[M];  
int p[N];  
int dist[N][N];  
int h[N],e[N\*2],w[N\*2],ne[N\*2],idx;//用来建树  
void add(int a,itn b,int w)  
{  
 e[dix] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx ++;  
}  
int find(int x)  
{  
 if(x != p[x])  
 p[x] = find(p[x]);  
 return p[x];  
}  
void dfs(int u,int fa,int maxd,int d[])//maxd存的是从根节点到当前点的边权最大值  
{  
 d[u] = maxd;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(j != fa)  
 {  
 dfs(j,u,max(maxd,w[i]),d);//维护的时候更新一下最大值即可  
 }  
 }  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 memset(h,-1,sizeof h);  
 for(int i=0;i<m;i++)  
 {  
 int a,b,w;  
 cin >> a >> b >> w;  
 edge[i] = {a,b,w};  
 }  
 sort(edge,edge + m);  
 for(int i=1;i<=n;i++) p[i] = i;  
 ll sum = 0;  
 for(int i=0;i<m;i++)  
 {  
 int a = edge[i].a,b = edge[i].b,w = edge[i].w;  
 int pa = find(a),pb = find(b);  
 if(pa != pb)  
 {  
 p[pa] = pb;  
 sum += w;  
 add(a,b,w),add(b,a,w);//建树，无向边  
 edge[i].f = 1;//标记是树边  
 }  
 }  
 for(int i=1;i<=n;i++)  
 {  
 dfs(i,-1,0,dist[i]);//传入父节点是因为是双向边防止死循环  
 }  
 for(int i=0;i<m;i++)  
 {  
 if(!edge[i].f)//是非树边  
 {  
 int a = edge[i].a,b = edge[i].b,w = edge[i].w;  
 if(w > dist[a][b])  
 {  
 res = min(res,sum + w - dist[a][b]);  
 }  
 }  
 }  
 cout << res << endl;  
 return 0;  
}//4 4 1 2 100 2 4 200 2 3 250 3 4 100 ----> 450  
复制

# 求负环的常用方法：基于SPFA

1. 统计每个点入队的次数，如果某个点入队 n 次，则说明存在负环
2. 统计当前每个点的最短路中所包含的边数，如果某点的最短路所包含的边数大于等于 n ，则说明存在负环。
3. 当所有点的入队次数超过 2n 时，我们就认为图中有很大可能是存在负环的。(一直超时再试)

## 例题1 904 虫洞

有 w 个单向的虫洞和 M 条双向的路径，N 片田地，农夫能够从某片田地出发，由路径和虫洞回到过去，并在出发前赶到出发地，问是否可以？ ------> 是否存在负环，存在负环会让时间减短，只有存在负环才会使得在出发前重新回到出发地。

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 510,M = 5210;  
int n,m1,m2;  
int h[N],e[M],w[M],ne[M],idx;  
int dist[N];  
int q[N],cnt[N];  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx ++;  
}  
bool spfa()  
{  
 int hh = 0,tt = 0;  
 memset(dist,0,sizeof dist);  
 memset(st,0,sizeof st);  
 memset(cnt,0,sizeof cnt);  
 for(int i=1;i<=n;i++)  
 {  
 q[tt ++] = i;//每个点都需要入队，因为不知道负环边经过那一个点，都需要判断下。  
 st[i] = true;  
 }  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == N)//防止多次进队导致越界循环队列  
 hh = 0 ;  
 st[t] = false;//出队  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] > dist[t] + w[i])  
 {  
 dist[j] = dist[t] + w[i];  
 cnt[j] = cnt[t] + 1;  
 if(cnt[j] >= n)  
 return true;  
 if(!st[j])  
 {  
 q[tt ++] = j;  
 if(tt == N) tt = 0;  
 st[j] = true;  
 }  
 }  
 }  
 }  
 return false;  
}  
int main()  
{  
 \_;  
 int t;  
 cin >> t;  
 while(t --)  
 {  
 cin >> n >> m1 >> m2;  
 memset(h,-1,sizeof h);  
 idx = 0;  
 while(m1 --)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 add(a,b,c),add(b,a,c);  
 }  
 while(m2 --)//虫洞  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 add(a,b,-c);  
 }  
 if(spfa())  
 cout << "YES" << endl;  
 else cout << "NO" << endl;  
 }   
}//2 3 3 1 1 2 2 1 3 4 2 3 1 3 1 3 3 2 1 1 2 3 2 3 4 3 1 8 ----> NO YES  
复制

# 求正环类问题

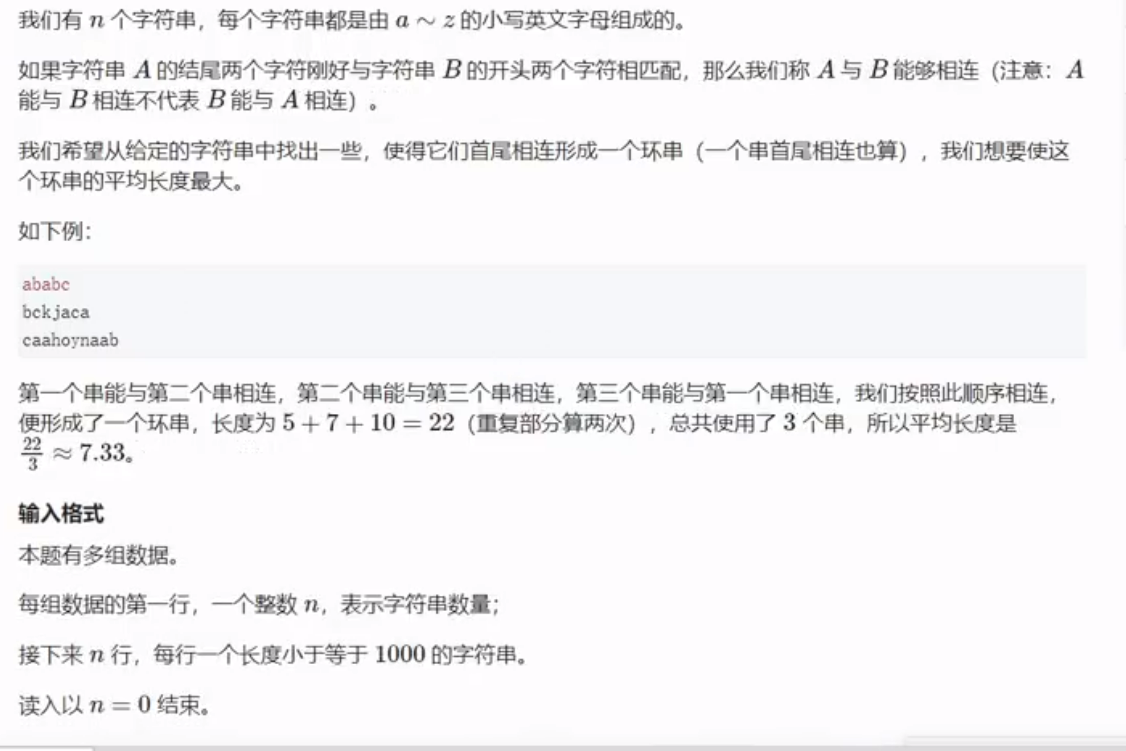
求一个图中是否存在正环可以转换为相反的求负环问题 ，只是边点权重的分配有点不同，但是也可以直接转换为求某点的最长路，加上一定的经验，例如当点入队次数**超过最多点的2倍时**，我们认为图中存在正环，部分题目不加此优化会超时，并且求负环的最短路和求正环的最长路的主要区别在于在更新边的权值的时候，最短路求得是小于（边更新变小)，最长路求的是大于（边更新变大）。

## 例题 2 361 观光奶牛

01分数规划  
L个点，P条边的有向图，每个点都有一个权值 f[i],每条边都有一个权值 t[i]，求图中的一个环，使环上各点的权值之和 / 环上各边的权值之和 最大，----> 01分数规划，点权和边权都存在，可以将点权放到边权上，出边入边都可以，由此转换为 fi - mid \* ti > 0 (fi - mid ti) > 0 图中是否存在一个正环，二分查找

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
const int N = 1010,M = 5010;  
int n,m;  
int wf[N];  
int h[N],e[M],wt[M],ne[M],idx;  
double dist[N];  
int q[N],cnt[N];  
bool st[N];  
  
void add(int a,int b,int c)  
{  
 e[idx] = b,ne[idx] = h[a],wt[idx] = c,h[a] = idx ++;  
}  
  
bool check(double mid)  
{  
 //距离不必初始化，距离是多少不会影响求负环  
 memset(st,0,sizeof st);  
 memset(cnt,0,sizeof cnt);  
 int hh = 0,tt = 0;//循环队列  
 for(int i=1;i<=n;i++)  
 {  
 q[tt ++] = i;  
 st[i] = true;  
 }  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == N) hh = 0;  
 st[t] = false;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] < dist[t] + wf[t] - mid \* wt[i])  
 {//最长路：边更新变大，与最短路相反  
 dist[j] = dist[t] + wf[t] - mid \* wt[i];  
 cnt[j] = cnt[t] + 1;  
 if(cnt[j] >= n) return true;  
 if(!st[j])  
 {  
 q[tt ++] = j;  
 if(tt == N) tt = 0;  
 st[j] = true;  
 }  
 }  
 }  
 }  
 return false;  
}  
  
int main()  
{  
 \_;  
 cin >> n >> m;  
 memset(h,-1,sizeof h);  
 for(int i=1;i<=n;i++)  
 cin >> wf[i];  
 while(m --)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 add(a,b,c);  
 }  
 double l = 0,r = 1010;  
 while(r - l > 1e-4)  
 {  
 double mid = (l + r) / 2;  
 if(check(mid)) l = mid;  
 else r = mid;  
 }  
 cout << r << endl;  
 return 0;  
}  
复制

## 例题 2 1165 单词环

 wi / si 最大， wi / 1 > M , (Wi - M 1) > 0 图中存在正环，判断无解的情况只需要判断M = 0即可，若此时 < 0 则代表无解. 由26个小写字母构成的两个字符最多有 26 26 = 676种可能，也即676个点，两两之间最多676 \* 676 < 100000条边，

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 700,M = 100010;//点的数量，边的数量  
int n;  
int h[N],e[M],w[M],ne[M],idx;  
double dist[N];  
int q[N],cnt[N];  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx ++;  
}  
bool check(double mid)  
{  
 memset(st,0,sizeof st);  
 memset(cnt,0,sizeof cnt);  
 int hh = 0,tt = 0;  
 for(int i=0;i<676;i++)  
 {  
 q[tt ++] = i;  
 st[i] = true;  
 }  
 int count = 0;  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == N) hh = 0;  
 st[t] = false;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] < dist[t] + w[i] - mid \* 1)  
 {//最长路：边更新变大，与最短路相反  
 dist[j] = dist[t] + w[i] - mid;  
 cnt[j] = cnt[t] + 1;  
 if(++count > 2 \* n) return true;//点更新的次数大于2\*n经验上就是存在负环  
 if(cnt[j] >= N) return true;  
 if(!st[j])  
 {  
 q[tt ++] =j;  
 if(tt == N) tt = 0;  
 st[j] = true;  
 }  
 }  
 }  
 }  
 return false;  
}  
  
int main()  
{  
 \_;  
 char str[1010];  
 while(cin >> n,n)  
 {  
 memset(h,-1,sizeof h);  
 idx = 0 ;  
 for(int i=0;i<n;i++)  
 {  
 cin >> str;  
 int len = strlen(str);  
 if(len >= 2)  
 {  
 int left = (str[0] - 'a') \* 26 + str[1] - 'a';  
 int right = (str[len - 2] - 'a') \* 26 + str[len - 1] - 'a';  
 add(left,right,len);  
 }  
 }  
 if(!check(0)) cout << "NO solution" << endl;  
 else   
 {  
 double l = 0,r = 1000;  
 while(r - l > 1e-4)  
 {  
 double mid = (l + r) / 2;  
 if(check(mid)) l = mid ;  
 else r = mid;  
 }  
 cout << r << endl;  
 }  
 }  
 return 0;  
}//3 intercommunicational alkylbenzenesulfonate tetraiodophenolphthalein ---> 0  
复制

# 差分约束

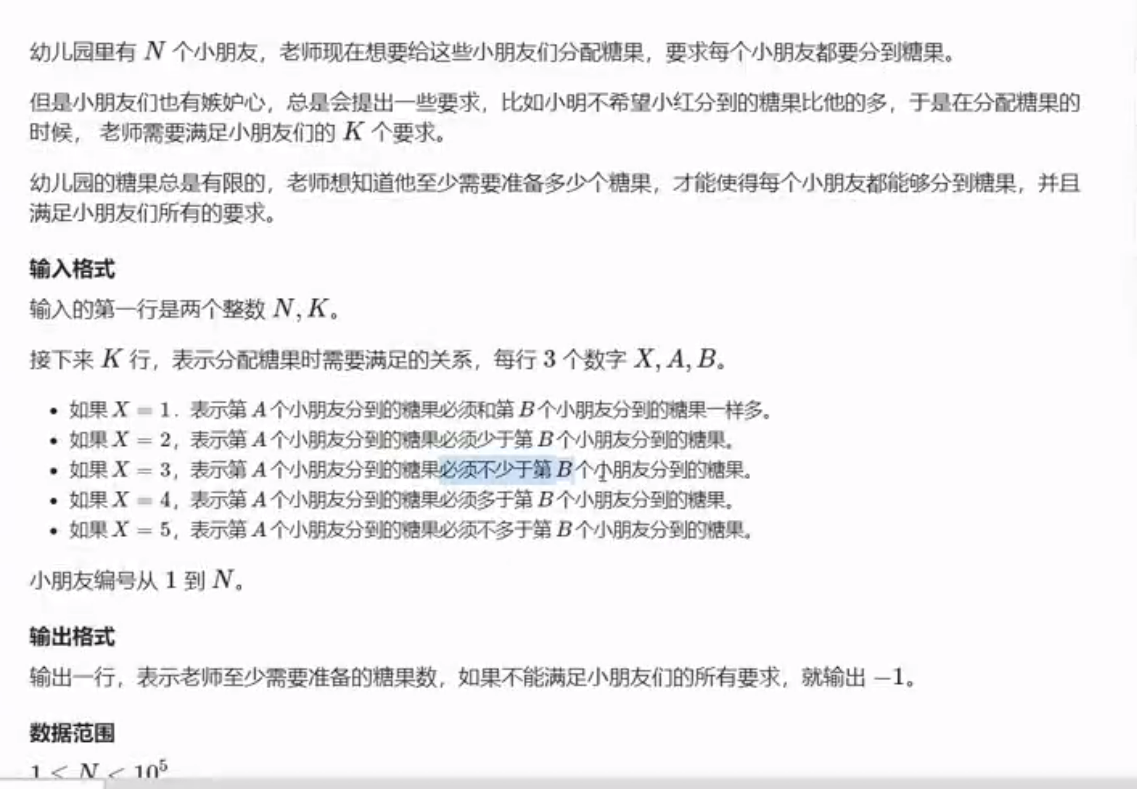
## 简介

对于最短路问题，队列的作用就是将遍历到的点存起来，栈也可以实现这个功能，不过队列是先进先出，栈是先进后出，如果存在一个环，由于栈是先进后出，栈相对于队列来说一般更快的找到环，因为先进后出,找到环之后会在一个点里不停地循环，每次都更新一次点的数量，对于当前点不断更新，而不是队列中将环中的点依次更新，实现线性。

1. 求不等式组的可行解 形如： + 这种**方程组**的一组可行解，源点需要满足的条件：**从源点出发，一定可以走到所有的边**。如果无解一定是推到了 < 的情况 存在负环（1.先将每个不等式 + 转换为一条从 走到 长度为 的一条边，2.找到一个超级源点，使得该源点一定可以遍历到所有边，3.从源点求一遍单源最短路，①如果存在负环，则原不等式组一定无解，②如果，没有负环，则dist[i]就是原不等式的组的一个可行解）
2. 如何求最大值或者最小值（这里的最大值是每一个变量的最值）

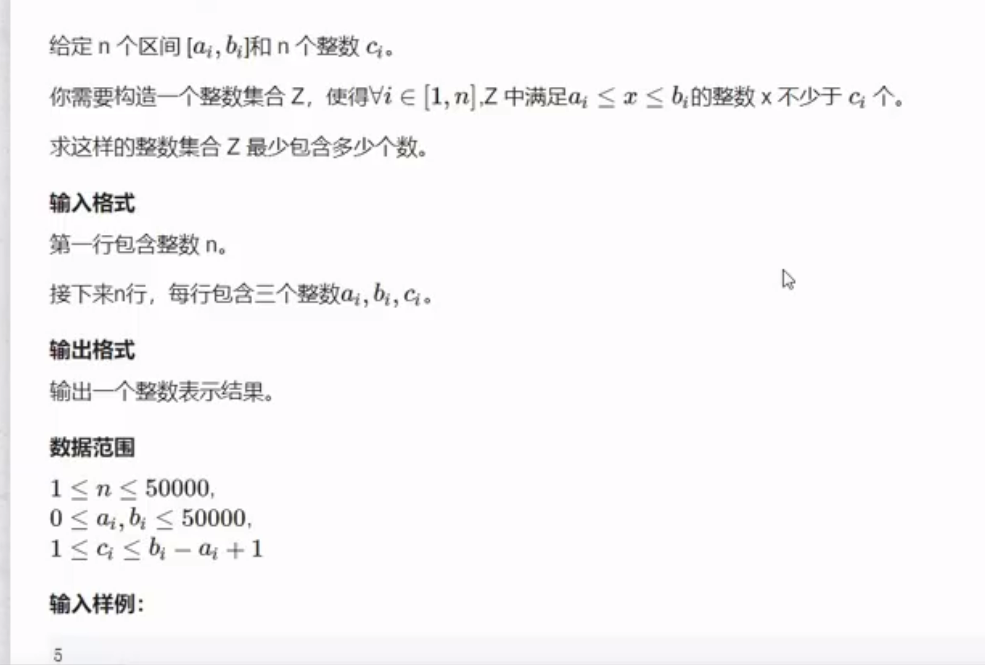
* 结论1 ： 如果求得是**最小值**，则应该求**最长路**；如果求得是**最大值**，则应该求**最短路** （如何转换 c,其中c是一个常数，这类的不等式，方法：建立一个超级源点 0，然后建立 0 -> i，长度是 c 的边即可。以求 的最大值为例：求所有从 出发，构成的不等式链 + + + ... + + ... ,所计算出的上界,最终 的最大值等于所有上界的**最小值。** 每一条链都转换为一条最长路，求最长路的最小值

## 例题 1 1169 糖果

五个条件分别转换为 A B,B A; B A + 1; A B ; A B +1 ; B A; 至少准备数等于最多需要数，求最大值 -> 最长路 -> 大于号，注意每个点都要 > 1,所以建立一个超级源点 = 0, 则 X > + 1  
优先使用队列，如果确定算法正确，并且超时的时候可以将队列改为栈尝试一次。

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 100010,M = 300010;//最坏情况是第一种，每个点建两条边，且源点要与每个点建边，故边是点的三倍  
int n,m;  
int h[N],e[M],w[M],ne[M],idx;  
int dist[N];  
int q[N],cnt[N];//cnt用来求正环  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx++;  
}  
bool spfa()  
{  
 int hh = 0,tt = 1;  
 memset(dist,-0x3f,sizeof dist);//最长路，需要初始化为负无穷  
 dist[0] = 0;//超级源点  
 q[0] = 0;  
 st[0] = true;  
 while(hh != tt)//虽然栈存求负环spfa会更快，但是在求一般的问题时，栈会十分么慢，最长路这种无环的时候更新从后往前会造成许多浪费，本题队列超时，需改为栈尝试  
 {  
 //int t = q[hh ++];  
 int t = q[-- tt];  
 //if(hh == N) hh = 0;  
 st[t] = false;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] < dist[t] + w[i])  
 {  
 dist[j] = dist[t] + w[i];  
 cnt[j] = cnt[t] + 1;  
 if(cnt[j] >= n + 1)//包括 0 号点，一共 n + 1 个点  
 return false;  
 if(!st[j])  
 {  
 q[tt ++] = j;  
 //if(tt == N) tt = 0;  
 st[j] = true;  
 }  
 }  
 }  
 }  
 return true;  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 memset(h,-1,sizeof h);  
 while(m --)  
 {  
 int x,a,b;  
 cin >> x >> a >> b;  
 if(x == 1)  
 add(b,a,0),add(a,b,0);  
 else if(x == 2)//从 a 向 b 连一条长度为 1 的边  
 add(a,b,1);  
 else if(x == 3)  
 add(b,a,0);//从 b 向 a 连一条长度为 0 的边  
 else if(x == 4)   
 add(b,a,1);  
 else add(a,b,0);  
 }  
 for(int i=1;i<=n;i++) add(0,i,1);//源点到每个点的距离为 1 ，源点的值为0，每个人大于1  
 if(!spfa())  
 cout << -1 << endl;//有负环无解  
 else  
 {  
 ll res = 0;  
 for(int i=1;i<=n;i++) res += dist[i];  
 cout << res << endl;  
 }  
   
}//5 7 1 1 2 2 3 2 4 4 1 3 4 5 5 4 5 2 3 5 4 5 1 ---> 11  
复制

## 例题 2 362 区间

 由于 c 的范围小于区间长度，所以本题一定有解， = 0， 表示 1 ~ i 中被选出的的个数， ,并且有 - 1 -> , 得到 [a,b],c --> - c.

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 50010,M = 150010;//因为三种情况都可能有一条边，所以边是点的三倍  
int n;  
int h[N],e[M],w[M],ne[M],idx;  
int dist[N];//dist即是前缀和数组  
int q[N],cnt[N];//cnt数组是判环的时候用的，这里保证有解不必判断  
bool st[N];  
viod add(int a,int b,int c)  
{   
 e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx ++;  
}  
void spfa()  
{  
 int hh = 0,tt = 1;  
 memset(dist,-0x3f,sizeof dist);  
 dist[0] = 0;//表示前 0 个数里面学几个数  
 q[0] = 0;  
 st[0] = true;  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == N) hh = 0;  
 st[t] = false;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] < dist[t] + w[i])  
 {  
 dist[j] = dist[t] + w[i];  
 if(!st[j])  
 {  
 q[tt ++] = j;  
 if(tt == N) tt = 0;  
 st[t] = true;  
 }  
 }  
 }  
 }  
}   
int main()  
{  
 \_;  
 cin >> n;  
 memset(h,-1,sizeof h);  
 for(int i=1;i<=50001;i++)  
 {  
 add(i-1,i,0);//第一种情况，由 i-1 向 i 连一条  
 add(i,i-1,-1);//由 i 到 i - 1 建一条权重为 -1 的边  
 }  
 while(n --)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 a ++,b ++;//此步操作是因为区间左端点可以取到 0，对于前缀和来说不方便，所以++，抵消掉 0  
 add(a - 1,b,c);  
 }  
 spfa();  
 cout << dist[50001] << endl;  
 return 0;  
}//5 3 7 3 8 10 3 6 8 1 1 3 1 10 11 1 ---> 6  
复制

## 例题 3 1170 排队布局

1. ,按编号顺序排列，2. + L(最短路是小于号的形式) 3. - D;源点假设所有的点都在0点的左半边，0->i=0, 第一种没有正负环，最后一个是最短路，中间的情况是最短路是否是正无穷

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 1010,M = 21010,INF = 0x3f3f3f3f;//第一种1000，后两种各10000种  
int n,m1,m2;  
int h[N],e[M],ne[M],w[M],idx;  
int dist[N];  
int q[N],cnt[N];  
bool st[N];  
void add(int a,int b,int c)  
{  
 e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx ++;  
}  
bool spfa(int size)  
{  
 memset(dist,0x3f,sizeof dist);  
 memset(cnt,0,sizeof cnt);  
 memset(st,0,sizeof st);  
 int hh = 0,tt = 0;  
 for(int i=1;i<=size;i++)  
 {  
 dist[i] = 0;  
 q[tt ++] = i;  
 st[i] = true;  
 }  
 while(hh != tt)  
 {  
 int t = q[hh ++];  
 if(hh == N) hh = 0;  
 st[t] = false ;  
 for(int i=h[t];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(dist[j] > dist[t] + w[i])  
 {  
 dist[j] = dist[t] + w[i];  
 cnt[j] = cnt[t] + 1;  
 if(cnt[j] >= n) return false;  
 if(!st[j])  
 {  
 q[tt ++] = j;  
 if(tt == N) tt = 0;  
 st[j] = true;  
 }  
 }  
 }  
 }  
 return true;  
}  
int main()  
{  
 \_;  
 cin >> n >> m1 >> m2;  
 memset(h,-1,sizeof h);  
 for(int i=1;i<n;i++)  
 add(i + 1,i,0);  
 while(m1 --)//第一种边  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 if(b < a) swap(a,b);   
 add(a,b,c);  
 }  
 while(m2 --)  
 {  
 int a,b,c;  
 cin >> a >> b >> c;  
 if(b < a) swap(a,b);  
 add(b,a,-c);  
 }  
 if(!spfa(n))//第一问需要把所有的点加到队列里面去  
 cout << -1 << endl;  
 else   
 {  
 spfa(1);//第二种只需要将第一个点放进队列中即可  
 if(dist[n] == INF) cout << -2 << endl;  
 else cout << dist[n] << endl;  
 }  
 return 0;  
}  
复制

## 例题 4 393 雇佣收银员

最小值 --> 最长路

# 有向图的强连通分量

对于一个有向图。连通分量：对于分量中任意两点u，v，必然可以从u走到v，且从v走到u。一般按照DFS的顺序来求解  
强连通分量 ：极大连通分量

### 作用：

令一个有向图缩点成为有向无环图（PAG），也即是拓扑图（缩点指的是将所有的连通分量缩成一个点,环成点），这样求最短路或者最长路可以直接递推来做 O(n + m)

### 分类 ：

1.树枝边（x 是 y 的一个父节点）  
2.前向边 （x 是 y 的祖先节点）  
3.后向边（x 是 y 的字节点，不必直接字节点）  
4.横叉边（x 和 y 不在同一个分支，由x搜到另一个分支的边，这个 y 此时已经被搜过了）

### 解释：

如果按照DFS从左到右的顺序搜索，由 x 搜到左边的点这时左边的点被搜过是横叉边，如果搜到右边的点，未搜过，是树枝边

### 判断：

如果一个点在强连通分量（SCC）中：  
1.存在后向边指向祖先节点（后向边一定向上走）  
2.它走到一个横叉边上，且横叉边的点走到了祖先节点

# Tarjan算法求SCC：

### 实现方式：

引入一个时间戳的概念，按照DFS的搜索顺序给每一个节点打上时间戳，即是第几个搜索到的点，对于每个点定义两个时间戳，dfn[u]表示遍历到 u 的时间戳，low[u] 从 u 开始走所能遍历到的最小时间戳，如果 u 是它所在的强连通分量的最高点，等价于 dfn[u] == low[u]

void tarjan(int u)  
{  
 dfn[u] = low[u] = ++ timestamp;//第一次先打上时间戳  
 stk[++ top] = u,in\_stk[u] = true;//入栈，并且标记为true  
 for(int i=h[u];~i;i=ne[i])//遍历 u 能到的点  
 {  
 int j = e[i];  
 if(!dfn[j])//当前的点没有遍历过  
 {  
 trajan(j);//遍历 j   
 low[u] = min(low[u],low[j])//这两个点相连通，最高点一致  
 }  
 else if(in\_stk[j])//这个情况下 j 要不是祖先要不是横叉点  
 low[u] = min(low[u],dfn[j]);//这个点遍历过，更新两个点能到的最高点  
 }  
 if(dfn[u] == low[u])//代表满足SCC  
 {  
 int y ;  
 ++ scc\_cnt;  
 do   
 {  
 y = stk[top --];  
 in\_stk[y] = false;  
 id[y] = scc\_cnt;  
 }while(y != u)  
 }  
}  
//这个栈中所有点是当前还没有搜完的强连通分量中的所有点   
复制

# 缩点：

遍历一下所有点，遍历所有 i 的领点，如果 i 和 j 不在同一个连通块加一条新边 id[i] -> id[y] , 连通分量编号递减的顺序一定是拓扑排序

## 例题 1 -- 1174受欢迎的牛

A,B)代表A认为B牛受欢迎。且具有传递性，求有多少牛被除自己之外的所有牛认为是受欢迎的,如果是拓扑图的话，则最多只有一个且这个点的出度为 0，强连通分量出度为 0，这个块中的所有点。

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 10010,M = 50010;  
int n,m;  
int h[N],e[M],ne[M],idx;  
int dfn[N],low[N],timestamp;  
int stk[N],top;  
bool in\_stk[N];  
int id[N],scc\_cnt,si[N];//每个点属于强连通块的编号，强连通块的数量,每个强连通块中点的数量  
int dout[N];//出度  
void add(int a,int b)  
{  
 e[idx]=b,ne[idx]=h[a],h[a] = idx ++;  
}  
void tarjan(int u)  
{  
 dfn[u] = low[u] = ++ timestamp;  
 stk[++ top] = u,in\_stk[u] = 1;  
 for(int i=h[u];~i;i=ne[i])//遍历当前点所能到的所有点  
 {  
 int j = e[i];  
 if(!dfn[j])  
 {  
 tarjan(j);  
 low[u] = min(low[u],low[j]);  
 }  
 else if(in\_stk[j])  
 {  
 low[u] = min(low[u],dfn[j]);  
 }  
 }  
 if(dfn[u] == low[u])  
 {  
 ++ scc\_cnt;  
 int y;  
 do  
 {  
 y = stk[top --];  
 in\_stk[y] = false;  
 id[y] = scc\_cnt;  
 si[scc\_cnt] ++;  
 }while(y != u);  
 }  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 memset(h,-1,sizeof h);  
 while(m --)  
 {  
 int a,b;  
 cin >> a >> b;  
 add(a,b);  
 }  
 for(int i=1;i<=n;i++)  
 {  
 if(!dfn[i])  
 tarjan(i);  
 }  
 for(int i=1;i<=n;i++)//统计新图中出度为 0 的点块  
 {  
 for(int j=h[i];~j;j=ne[j])  
 {  
 int k = e[j];  
 int a = id[i],b = id[k];// a 表示 i 所在的连通块，b 表示 k 所在的连通块  
 if(a != b)//不在一个连通块  
 dout[a] ++;//从 a 走到 b ，a的块出度加 1  
 }  
 }  
 int zeros = 0,sum = 0;  
 for(int i=1;i<=scc\_cnt;i++)  
 {  
 if(!dout)  
 {  
 zeros ++ ;  
 sum += si[i];  
 if(zeros > 1)  
 {  
 sum = 0;  
 break;  
 }   
 }  
 }  
 cout << sum << endl;  
}  
// 3 3 1 2 2 1 2 3 ---> 1  
复制

## 例题2 -- 367学校网络

一些学校之间相互链接，可以理解为数据共享，但不是双向的，有向边，问最少提供给几个学校使得所有的学校都可以获得这个软件 和 最少需要提供多少支援关系使得将软件提供给任何一个学校使得所有的学校都可以获得这个新软件。  
第一问的答案是多少个入度为 0 的点，即是起点，第二问的答案是max(p,q),p，q 分别代表起点和终点的数量，抽象理解，将不相连的块的终点和起点相连 (假如p < q，如果p=1，易得结果是 q，如果p > 1则一定可以找到两个起点到达的终点不同（反证法p < q） = p - p2, = q - q1,递推可以得到只有一个起点，总和为 q，即是max(p,q),q < p 同理)

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 110,M = 10010;  
int n;  
int h[N],e[M],ne[M],idx;  
int dfn[N],low[N],timestamp;  
int stk[N],top;  
bool in\_stk[N];  
int id[N],scc\_cnt;  
int din[N],dout[N];  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx ++;  
}  
void tarjan(int u)  
{  
 dfn[u] = low[u] = ++ timestamp;  
 stk[++ top] = u,in\_stk[u] = true;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(!dfn[j])  
 {  
 tarjan(j);  
 low[u] = min(low[u],low[j]);  
 }  
 else if(in\_stk[j])   
 low[u] = min(low[u],dfn[j]);  
 }  
 if(dfn[u] == low[u])  
 {  
 ++ scc\_cnt;  
 int y;  
 do   
 {  
 y = stk[top --];  
 in\_stk[y] = false;  
 id[y] = scc\_cnt;   
 }while(y != u);  
 }  
}  
int main()  
{  
 \_;  
 cin >> n;  
 memset(h,-1,sizeof h);  
 for(int i=1;i<=n;i++)  
 {  
 int t;  
 while(cin >> t,t) add(i,t);  
 }  
 for(int i=1;i<=n;i++)  
 {  
 if(!dfn[i])  
 tarjan(i);  
 }  
 for(int i=1;i<=n;i++)  
 {  
 for(int j=h[i];~j;j=ne[j])  
 {  
 int k = e[j];  
 int a = id[i],b = id[k];  
 if(a != b)  
 {  
 dout[a] ++;  
 din[b] ++;  
 }  
   
 }  
 }  
 int a = 0,b = 0;  
 for(int i=1;i<=scc\_cnt;i++)  
 {  
 if(!din[i]) a ++;  
 if(!dout[i]) b ++;  
 }  
 cout << a << endl;  
 if(scc\_cnt == 1)//只有一个块，  
 cout << 0 << endl;  
 else cout << max(a,b) << endl;  
}  
//5 2 4 3 0 4 5 0 0 0 1 0 ----> 1 2  
复制

## 例题4 -- 1175最大半连通子图

一个有向图 G = (V,E)如果任意 u,v 属于V，满足 u -> v 或 v -> u 即对于图中任意两点u,v存在一条 u 到 v 的 有向路径或者从 v 到 u 的有向路径（至少有一个成立)，若 = ( , )满足 是E中所有和 有关的边，则称 是G的一个导出子图，若 半连通则称为G的半连通子图，包含点数最多则是最大半连通子图，要求给定一个有向图G，求出最大半连通子图的节点数K，以及不同的最大半连通子图的数目C ---> 第一问是求一条最长的链，不分叉的链，第二问是最长链的方案数 注意去重

#include<bits/stdc++.h>//用哈希表判重  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 1e5 + 10,M = 2e6 + 10;  
int n,m,mod;  
int h[N],hs[N],e[M],ne[M],idx;// h 是原图的表头 hs 是缩点后的表头  
int dfn[N],low[N],timestamp;  
int stk[N],top;  
bool in\_stk[N];  
int id[N],scc\_cnt,size[N];  
int f[N],g[N];  
void add(int h[],int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx ++;  
}  
void tarjan(int u)  
{  
 dfn[u] = low[u] = ++ timestamp;  
 stk[++ top] = u,in\_stk[u] = true;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(!dfn[j])  
 {  
 tarjan(j);  
 low[u] = min(low[u],low[j]);  
 }  
 else if(in\_stk[j])  
 low[u] = min(low[u],dfn[j]);  
 }  
 if(dfn[u] == low[u])  
 {  
 ++ scc\_cnt;  
 int y;  
 do  
 {  
 y = stk[top --];  
 in\_stk[y] = false;  
 id[y] = scc\_cnt;  
 size[scc\_cnt] ++;//连通块中的点数一定满足半连通，满足点之间相连就行可以使单双向  
 }while(y != u);  
 }  
}  
int main()  
{  
 \_;  
 memset(h,-1,sizeof h);  
 memset(hs,-1,sizeof hs);  
 cin >> n >> m >> mod;  
 while(m --)  
 {  
 int a,b;  
 cin >> a >> b;  
 add(h,a,b);  
 }  
 for(int i=1;i<=n;i++)  
 {  
 if(!dfn[i])  
 tarjan(i);  
 }  
 //建新图  
 unordered\_set<ll> S;//去重 (u,v) ---> 哈希为 u \* 1000000 + v  
 for(int i=1;i<=n;i++)  
 {  
 for(int j=h[i];~j;j=ne[j])  
 {  
 int k = e[j];  
 int a = id[i],b = id[k];  
 ll hash = a \* 1000000ll + b;  
 if(a != b && !S.count(hash))  
 {   
 add(hs,a,b);  
 S.insert(hash);  
 }  
 }  
 }  
 for(int i=scc\_cnt;i;i--)//递减情况下缩点后的图是拓扑图  
 {  
 if(!f[i])//先更新联通快内的点数  
 {  
 f[i] = size[i];  
 g[i] = 1;//第一次更新个数一定是 1   
 }  
 for(int j=hs[i];~j;j=ne[j])  
 {  
 int k = e[j];  
 if(f[k] < f[i] + size[k])//将这个点块和下一个相连的点块结合点数的情况比较  
 {  
 f[k] = f[i] + size[k];  
 g[k] = g[i];  
 }  
 else if(f[k] == f[i] + size[k])  
 g[k] = (g[k] + g[i]) % mod;  
   
 }  
 }  
 int maxf = 0,sum = 0;//maxf最多子节点，sum是方案数  
 for(int i=1;i<=scc\_cnt;i++)  
 {  
 if(f[i] > maxf)  
 {  
 maxf = f[i];  
 sum = g[i];  
 }  
 else if(f[i] == maxf)  
 sum = (sum + g[i]) % mod;  
 }  
 cout << maxf << endl << sum << endl;  
}  
复制

## 例题5 -- 368 银河

每行输入 T A B 代表（A,B)之间的亮度关系，T=1，A=B;T=2,A<B;T=3,A>=B;T=4,A>B,T=5,A<=B，大小关系均是亮度的相对强弱，输出，这些星系亮度值总和最小是多少，每个点的亮度大于等于 1 ，spfa算法也可做但可能被卡，Tarjan算法稳定 O(n),判断是否有正环，环即是每两个点之间一定是不大于或者不小于关系，这样才等价于一个点 一、tarjan 二、缩点建图 三、依据拓扑序递推

#include<bits/stdc++.h>  
using namespace std;  
#define \_ ios::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);  
#define endl '\n'  
#define ll long long   
const int N = 1e5 + 10,M = 6e5 + 10;  
int n,m;  
int h[N],hs[N],e[M],ne[M],w[N],idx;  
int dfn[N],low[N],timestamp;  
int stk[N],top;  
bool in\_stk[N];  
int id[N],scc\_cnt,size[N];  
int dist[N];  
void add(int h[],int a,int b,int c)  
{  
 e[idx] = b,w[idx] = c,ne[idx] = h[a],h[a] = idx ++;  
}  
void tarjan(int u)  
{  
 dfn[u] = low[u] = ++ timestamp;  
 stk[++ top] = u,in\_stk[u] = true;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(!dfn[j])  
 {  
 tarjan(j);  
 low[u] = min(low[u],low[j]);  
 }  
 else if(in\_stk[j])  
 low[u] = min(low[u],dfn[j]);  
 }  
 if(dfn[u] == low[u])  
 {  
 ++ scc\_cnt;  
 int y;  
 do   
 {  
 y = stk[top --];  
 in\_stk[y] = false;  
 id[y] = scc\_cnt;  
 size[scc\_cnt] ++;  
 }while(y != u);  
 }  
}  
int main()  
{  
 \_;  
 cin >> n >> m;  
 memset(h,-1,sizeof h);  
 memset(hs,-1,sizeof hs);  
 for(int i=1;i<=n;i++)  
 add(h,0,i,1);//从0号点开始向其他点建一条长度为 1 的边  
 while(m --)  
 {  
 int t,a,b;  
 cin >> t >> a >> b;  
 if(t == 1) add(h,b,a,0),add(h,a,b,0);//亮度相等  
 else if(t == 2)  
 add(h,a,b,1);//第4个变量是长度  
 else if(t == 3)  
 add(h,b,a,0);  
 else if(t == 4)  
 add(h,b,a,1);  
 else add(h,a,b,0);  
 }  
 tarjan(0);//从0 号点开始tarjan  
 bool success = true;  
 for(int i=0;i<=n;i++)  
 {  
 for(int j=h[i];~j;j=ne[j])  
 {  
 int k = e[j];  
 int a = id[i],b = id[k];  
 if(a == b)  
 {  
 if(w[j] > 0)  
 {  
 success = false;  
 break;  
 }  
 }  
 else add(hs,a,b,w[j]);  
 }  
 if(!success) break;  
 }  
 if(!success) cout << -1 <<endl;  
 else   
 {  
 for(int i=scc\_cnt;i;i--)  
 {  
 for(int j=hs[i];~j;j=ne[j])  
 {  
 int k = e[j];  
 dist[k] = max(dist[k],dist[i] + w[j]);  
 }  
 }  
 ll res = 0;  
 for(int i=1;i<=scc\_cnt;i++)  
 {  
 res += dist[i] \* size[i];  
 }  
 cout << res << endl;  
 }  
}  
  
复制

## 例题

* 暑假题单6 A ，判断子节点中是否有low[v] > dfn[u] ， 或者一个节点是否有两个以上的儿子，这些点一定是割点 (割点)删除这个点会是图一分为二，两个不相连通的图。

### 分析

如果是割点，则把这个点删除后分为两个不连通的图，如果low[v] >= dfn[u],即是子节点的low小于父节点的dfn，这个时候代表子节点一定和图上的上面的点(已经遍历过得点)不相连，则将这个点割开会分开

#include <bits/stdc++.h>  
using namespace std;  
#define int long long  
#define endl "\n"  
#define ull unsigned long long  
#define ms(x,y) memset(x,y,sizeof x);  
#define debug(x) cout << #x << " = " << x << endl;  
#define ios ios::sync\_with\_stdio(0),cin.tie(0),cout.tie(0);  
#define fre freopen("input.txt", "r", stdin);freopen("output.txt", "w", stdout);  
const int mod=998244353;  
const int inf=0x3f3f3f3f3f3f3f3f;  
const double esp=1e-6;  
const int N = 1e6 + 10;  
int h[N],e[N \* 2],ne[N\*2],idx;  
int low[N],dfn[N];  
int id[N];  
bool in\_stk[N];  
int stk[N],top;  
int scc\_cnt;  
int timestamp;  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx ++;  
}  
int n,m;  
bool st[N];  
void tarjan(int u,int fa)  
{  
 int child = 0;  
 low[u] = dfn[u] = ++ timestamp;  
 stk[++ top] = u,in\_stk[u] = true;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(!dfn[j])  
 {  
 tarjan(j,fa);  
 low[u] = min(low[u],low[j]);  
 if(low[j] >= dfn[u] && u != fa) st[u] = true;//不是父节点的情况下  
 if(u == fa) child ++;  
  
 }  
 else if(in\_stk[j])  
 low[u] = min(low[u],dfn[j]);  
 }  
 if(child >= 2 && u == fa) st[u] = 1;//有两个以上子节点  
}  
  
void slove()  
{  
 cin >> n >> m;  
 memset(h,-1,sizeof h);  
 while(m --)  
 {  
 int a,b;  
 cin >> a >> b;  
 add(a,b);  
 add(b,a);  
 }  
 for(int i=1;i<=n;i++)  
 {  
 if(!dfn[i])  
 tarjan(i,i);  
 }  
 int ans = 0;  
 for(int i=1;i<=n;i++)  
 if(st[i]) ans ++;  
 cout << ans << endl;  
 for(int i=1;i<=n;i++)  
 if(st[i]) cout << i << " ";  
}  
// #define LOCAL  
signed main()  
{  
 ios  
 //fre  
 #ifdef LOCAL  
 freopen("in.txt", "r", stdin);  
 freopen("out.txt", "w", stdout);  
 auto start = std::chrono::high\_resolution\_clock::now();  
 #endif  
   
 int t=1;  
 // cin >> t;  
 while(t--)  
 slove();  
  
 #ifdef LOCAL  
 auto end = std::chrono::high\_resolution\_clock::now();  
 cout << "Execution time: "  
 << std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count()  
 << " ms" << '\n';  
 #endif  
 return 0;  
}  
复制

* 暑假题单 6 J Tarjan + 割点

// #pragma GCC optimize(2)  
// #pragma GCC optimize(3,"Ofrrst","inline")  
#include <bits/stdc++.h>  
using namespace std;  
#define int long long  
#define endl "\n"  
#define ull unsigned long long  
#define ms(x,y) memset(x,y,sizeof x);  
#define debug(x) cout << #x << " = " << x << endl;  
#define ios ios::sync\_with\_stdio(0),cin.tie(0),cout.tie(0);  
#define fre freopen("input.txt", "r", stdin);freopen("output.txt", "w", stdout);  
const int mod=998244353;  
const int inf=0x3f3f3f3f3f3f3f3f;  
// const int N = 3e5+10;  
const double esp=1e-6;  
const int N = 1e6 + 10;  
int h[N],e[N \* 2],ne[N\*2],idx;  
int low[N],dfn[N];  
int id[N];  
bool in\_stk[N];  
int stk[N],top;  
int scc\_cnt;  
int timestamp;  
void add(int a,int b)  
{  
 e[idx] = b,ne[idx] = h[a],h[a] = idx ++;  
}  
int n,m;  
int A,B;  
int check(int u,int v)  
{  
 if(dfn[v]<=dfn[A]&&dfn[v]>dfn[B]) return 1;  
 if(dfn[v]<=dfn[B]&&dfn[v]>dfn[A]) return 1;  
 return 0;  
}  
bool st[N];  
int ans = 1e9;  
void tarjan(int u,int fa)  
{  
 int child = 0;  
 low[u] = dfn[u] = ++ timestamp;  
 stk[++ top] = u,in\_stk[u] = true;  
 for(int i=h[u];~i;i=ne[i])  
 {  
 int j = e[i];  
 if(!dfn[j])  
 {  
 tarjan(j,u);  
 low[u] = min(low[u],low[j]);  
 if(low[j] >= dfn[u] && check(u,j) && u!=A&&u!=B) ans = min(ans,u);//不是父节点的前提之下  
 }   
 else if(in\_stk[j])  
 low[u] = min(low[u],dfn[j]);  
 }  
 // if(child >= 2 && u == fa) st[u] = 1;//有两个以上的子节点  
}   
  
void slove()  
{  
 cin >> n;  
 memset(h,-1,sizeof h);  
 int a,b;  
 while(cin >> a >> b,a&&b)  
 {  
 add(a,b);  
 add(b,a);  
 }  
 cin >> A >> B;  
 tarjan(1,0);  
 if(ans == 1e9) cout << "No solution" << endl;  
 else cout << ans << endl;  
   
}  
// #define LOCAL  
signed main()  
{  
 ios  
 //fre  
 #ifdef LOCAL  
 freopen("in.txt", "r", stdin);  
 freopen("out.txt", "w", stdout);  
 auto start = std::chrono::high\_resolution\_clock::now();  
 #endif  
   
 int t=1;  
 // cin >> t;  
 while(t--)  
 slove();  
  
 #ifdef LOCAL  
 auto end = std::chrono::high\_resolution\_clock::now();  
 cout << "Execution time: "  
 << std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count()  
 << " ms" << '\n';  
 #endif  
 return 0;  
}  
复制

位运算

* (a ^ b) ^ (b ^ c) = c
* a & b + a ^ b = a | b
* a & b + a | b = a + b
* a + b = (a ⊕ b) + 2(a ∧ b)
* (a + b) − (a ⊕ b) = 2(a ∧ b)