目录

[TOC]

1 图论

1.1 1. 最短路理解

1.1.1 1.1. 经典求法 dijkstra

从 bfs 演化而来，是最短路最好的算法，（是优先队列的 bfs，我们每次寻找的点就是当下最好的，贪心的扩大图中的点

每次找到的 num 就是在 vis 集合中能到达的且离 s（源点）最近的点，他的最短路就可以确认。

一个点可以进队多次，但是取出来只有剩下的在队伍中作废（因为 vis 置为一了

PS：1. 可以用 P 数组记录路径，p[i] 就是记录到扩展到 i 点的顶点 2. 不可以有负权边，也不能有负圈

邻接矩阵版

```
int a[1003][1003];//存图
int  d[maxn];//存距离
int p[maxn];//可以用来求一个路径
int vis[maxn];//维护我们已经知道最短路的点集
void dijkstra(int s){
    memset(d,INF,sizeof(d));
    d[s] = 0;
    for(int i = 0;i < n;i++){//总共循环 n 次，因为我们求 n 个点的最短路，每次循环必求出一个点的最短路
        int num = -1,minn = INF;
        for(int j=1;j<=n;j++){
            if(!vis[j]&&d[j]<minn){
                num = j;
                minn = d[j];
            }
        }
        vis[num] = 1;//每次选出的点肯定是最短路
        for(int j=1;j<=n;j++){
            if(!vis[j]){
                d[j] = min(d[j],d[num]+a[num][j]);
            }
        }
    }
}
```

链式前向星版

```
struct dis{
  int id,d;
  bool operator <(const dis &x)const{//优先列对这个结构体的排队规则，注意内部如果是 > 号是最小的在队头
      return d>x.d;
```

```
  }
};
int vis[maxn];
int d[maxn];//也要一个 d 数组
struct edge{
    int v,w,next;
}e[maxn<<1];
int cnt,head[maxn];
void add(int u,int v,int w){
    e[cnt].v = v;
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
void dijkstra(int s){
    memset(d,INF,sizeof(d));
    priority_queue<dis>q;
    dis ss = {s,0};
    q.push(ss);
    d[s] = 0;
    while(!q.empty()){
        dis now = q.top();
        q.pop();
        if(vis[now.id])continue;//如果已经出过队了就不用在看了
        vis[now.id] = 1;
        for(int i = head[now.id];~i;i = e[i].next){
            int v = e[i].v;
            if(d[v]>now.d+e[i].w){
                d[v] = now.d + e[i].w;
                dis nx = {v,d[v]};
                q.push(nx);
            }
        }
    }
}
```

1.1.2 1.2. Bellman-Ford 与 SPFA

先说说 Bellman-Ford，其实大多数书上就说他是迭代扩展，标号修正，其实很难理解，其实大致流程就是从 u 出发先找距离为一的点我们把这些点的距离更新，再找距离为二条边的点，从距离为一条边的点推过来，然后距离为三的。。。从其他的点推过来，其实就是类似动态规划的 flody 思想。蓝书上有另一个说法，就是所有点满足三件不等式，所以我们就从关于源点最近的点开始推，然后推到其他所有边 SPFA 其实是 Bellman-Ford 的队列优化方法，因为我们找下一层边时，队

列中的点就是全部带扩展的边，（如果我们不这样的话我我们要扫到其他所有边

1.1.3 1.3. SPFA

```
int dis[N],vis[N],ct[N];
queue<int>q;//队列和数组记得初始化
int SPFA(int s,int h){//h 其实也不需要，我写的这道题需要
    q.push(s);
    vis[s] = 1;ct[s] = 1;dis[s] = 0;
    mxh[s] = h;
    while(!q.empty()){
        int now = q.front();
        q.pop();
        vis[now] = 0;
        for(int i = head[now];~i;i=e[i].next){
            int v = e[i].v;
            int w = e[i].w;
            int limt = e[i].limt;
            if(mxh[v] < min(mxh[now],limt)){//按照最短路的化这里要修改转移方程
                mxh[v]  = min(mxh[now],limt);
                if(!vis[v]){
                    ct[v]++;
                    vis[v] = 1;
                    q.push(v);
                    if(ct[v]>n)return 1;
                }
            }
        }
    }
    return 0;
}
```

1.1.4 1.4. Floyd

```
vector<int>path;
int pos[N][N];//记录路径表示 i 与 j 经过需要经过的编号最大点 (k)
void get_path(int x,int y){//递归找路，加上 x 与 y 就能获得整个路径
    if(pos[x][y]==0)return ;
    get_path(x,pos[x][y]);
    path.push_back(pos[x][y]);
    get_path(pos[x][y],y);
```

```
}
for(int k = 1;k <= n;k++){
        for(int i = 1;i <= n;i++){
            for(int j = 1;j <= n;j++){
                if(d[i][j] > d[i][k]+d[k][j]){
                    d[i][j] = d[i][k] + d[k][j];
                    pos[i][j] = k;
                }
            }
        }
    }
```

1.1.5 1.5. 计算无向图中的最小环

如果出现环套环，这种情况 dfs 标记深度，判环是不可取的，并查集维护集合大小也不可续，dfs 搜索到自身也不可取

我么给出了一种复杂度比较大的方法 Floyd 方法：

众所周知，Floyd 算法求最短路的过程是三重循环。当最外层恰好循环到 k 时，代表着目前所求出的最短路所含的点集为 $[1,k]$ 在第 k 次循环时 $dp[i][j]$ 是 i 到 j 的最短路，并且不经过 k, 我们看 k 这个点，他经过了两个点，然后这两个点的最短路是 $dp[i][j]$，那说明经过至少有 k,i,j 三个点的最小环就可以求出来了，

注意初始化 dp 数组的值例题：

- Shortest Cycle 首先我们看出来了，应该是抽屉原理（n 个抽屉，如果我们放 n+1 个小球，必定会有两个小球在一个抽屉），一共就位，如果一位存在三个数都为 1，那么答案就是 3，也就是说如果给定的数个数大于 64*2，那么必行会有某一位存在三个 1，那么也就是把问题的规模放到的 100 左右，然后就是刚好可以利用 floyd 算法求一个最小环了

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
using namespace std;
typedef long long ll;
const int maxn = 1e5+5;
ll a[maxn];
ll g[200][200];
ll dp[200][200];
ll ans = INF;
int main(){
    int n;
    int cnt = 0;
    scanf("%d",&n);
    for(int i = 0;i<n;i++){
        ll x;scanf("%lld",&x);
        if(x!=0)a[cnt++]=x;
```

```
        }
        if(cnt>64*2){
            printf("3\n");
            return 0;
        }
        for(int i =0;i < cnt;i++){
            for(int j = 0 ;j<cnt;j++){
                dp[i][j] = 100;
                g[i][j] = 100;
            }
        }
        for(int i = 0;i<cnt;i++){
            dp[i][i] = 0;
            for(int j = 0;j < i;j++){
                if((a[i]&a[j])!=0){
                    g[i][j] = g[j][i] = 1;
                    dp[i][j] = dp[j][i] = 1;
                }
            }
        }
        for(int k = 0;k < cnt;k++){
            for(int i = 0;i < k; i++){
                for(int j = i+1;j<cnt;j++){
                    ans = min(ans,dp[i][j] + g[i][k] + g[k][j]);
                }
            }

            for(int i=0;i<cnt;i++){
                for(int j = 0;j<cnt;j++){
                    dp[i][j] = min(dp[i][j],dp[i][k] + dp[k][j]);
                }
            }
        }

        if(ans>=100){
            cout<<-1<<endl;
        }
        else cout<<ans<<endl;
    }
```

1.1.6 1.6. 最长路

- [P1807 最长路]https://www.luogu.com.cn/problem/P1807

  SPFA 跑带有负权的最长路（邻接矩阵版）一定要判断是否要有重边!!!!!! 此题数据范围较小，数据不强，如果数据变大我们需要用链式前向星

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
using namespace std;
int dis[1505];
int g[1505][1505];
int vis[1505],cnt[1509];
queue<int >q;
int n,m;
int spfa(int s){
    q.push(s);
    vis[s] = 1;cnt[s] =1 ,dis[s]=0;
    while(!q.empty()){
        int  now = q.front();
        q.pop();
        vis[now] = 0;
        for(int i=1;i<=n;i++){
            if(g[now][i]!=INF)
                if(dis[i]<dis[now]+g[now][i]){
                    dis[i]=dis[now]+g[now][i];
                    if(!vis[i]){
                        cnt[i]++;
                        vis[i] = 1;
                        q.push(i);
                        if(cnt[i]>n)return 1;
                    }
                }
        }
    }
    return 0;
}
int main(){
    cin>>n>>m;
    memset(g,INF,sizeof(g));//邻接表需要设置 INF 表示两个点没有变相连
    for(int i=0;i<m;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        if(g[u][v]!=INF)g[u][v] = max(w,g[u][v]);//判断是否有重边
```

```
        else g[u][v] = w;
    }
    spfa(1);
    if(dis[n]>0)cout<<dis[n];
    else cout<<"-1";
}
```

1.1.7 1.7. Floyd 记录路径

```
#include <iostream>
#include <cstring>
#include <vector>
#include <algorithm>
#include <cstdio>
#define INF 0x3f3f3f3f
using namespace std;
const int N = 105;
const int maxn = 10005;
int n,m;
int d[N][N];
int a[N][N];
int ans = INF;
vector<int>path;
int pos[N][N];
void get_path(int x,int y){
    //cout<<x<<" "<<y<<"#"<<pos[x][y]<<endl;
    if(pos[x][y]==0)return ;
    get_path(x,pos[x][y]);
    path.push_back(pos[x][y]);
    get_path(pos[x][y],y);
}
int main(){
    cin>>n>>m;
    memset(a,INF,sizeof(a));
    memset(d,INF,sizeof(d));
    for(int i = 1;i<=n;i++){
        a[i][i] = d[i][i] = 0;
    }
    for(int i = 0;i < m;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
```

```
        a[u][v] = a[v][u] = min(a[u][v],w);
        d[u][v] = d[v][u] = min(d[u][v],w);
    }
    for(int k = 1;k <= n;k++){
        for(int i = 1;i < k;i++){
            for(int j = i+1;j < k ;j++){
                if((long long )d[i][j]+a[j][k]+a[k][i] < ans){
                    ans = d[i][j]+a[j][k]+a[k][i];
                    //cout<<ans<<"##"<<endl;
                    path.clear();
                    path.push_back(i);
                    get_path(i,j);
                    path.push_back(j);
                    path.push_back(k);

                }
            }
        }

        for(int i = 1;i <= n;i++){
            for(int j = 1;j <= n;j++){
                if(d[i][j] > d[i][k]+d[k][j]){
                    d[i][j] = d[i][k] + d[k][j];
                    pos[i][j] = k;
                }
            }
        }
    }
    if(ans == INF){
        cout<<"No solution.";return 0;
    }
    for(int i = 0;i < path.size();i++){
        cout<<path[i]<<" ";
    }
}
```

1.2 2. 最小生成树

1.2.1 2.1. Kruskal

思想：并查集 + 贪心

Kruskal 板子

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
using namespace std;
const int maxn  = 5005;
int cnt = 0;
int head[maxn];
struct nod{
    int u,v,w;
}e[200005];//这里不是前向行的存边方法，只是正长的存边
bool cmp(nod a ,nod b){
    return a.w<b.w;
}
int fa[maxn];
int get(int x){
    if(fa[x]==x)return x;
    else return fa[x]=get(fa[x]);
}
bool merge(int x,int y){
    int p = get(x);
    int q = get(y);
    if(p!=q){
        fa[p] = q;
        return true;
    }
    return false ;
}
int main(){
    int n,m;

    scanf("%d%d",&n,&m);
    for(int i=0;i<=n;i++)fa[i] = i;
    for(int i=0;i<m;i++){
        scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);

    }
    //板子开始部分
    sort(e,e+m,cmp);//贪心的排序
    int ans=0,cnt=0;
    for(int i=0;i<m;i++){
        if(cnt==n-1)break;
        else if(merge(e[i].u,e[i].v)){
```

```
            cnt++;
            ans+=e[i].w;

        }
    }

    //
    cout<<ans;
}
```

1.3 3. 拓扑排序

1.3.1 3.1. 关键词

有向无环图，入度，出度拓扑序的用途

概念：对于一张有向无环图 (DAG) 而言，该图的拓扑排序是一个由该图所有顶点组成的线性序列。使得图中任意一对顶点 u 和 v，若存在边从 u 指向 v，则 u 在线性序列中出现在 v 之前。(对一个有向无环图 (Directed Acyclic Graph 简称 DAG)G 进行拓扑排序，是将 G 中所有顶点排成一个线性序列，使得图中任意一对顶点 u 和 v，若边 (u,v) 属于 E(G)，则 u 在线性序列中出现在 v 之前。

完全是拓扑序的板子

```
// 预先处理 in 数组，ans 是存储拓扑序的数组，如果要方便查找，再加一个 pos 数组
vector<int>g[maxn];//这里我用的是 vector 存的图
    queue<int> q;
    for(int i=1;i<=n;i++){
        if(in[i]==0){
            q.push(i);
        }
    }
    int ct = 0;
    while(!q.empty()){
        int f = q.front();
        q.pop();
        ans[++ct] = f;
        pos[f] = ct;
        for(int i=0;i<g[f].size();i++){
            int v = g[f][i];
            in[v]--;
            if(in[v]==0){
                q.push(v);
            }
```

```
        }
    }
    if(ct<n)这里如果成立说明图是有环的所以无法排序
```

网上找的链式前向行的拓扑排序版本

```
    struct node{
        int to,next;
    }e[maxn<<1];
    void init(){
        vec.clear();
        cnt=tot=0;
        for(int i=1;i<=n;i++){
            pos[i]=deg[i]=head[i]=0;
        }
    }
    void add(int u,int v){
        e[++cnt].to=v;
        e[cnt].next=head[u];
        head[u]=cnt;
    }
    //



        while(!que.empty()){
            int x=que.front();
            que.pop();
            pos[x]=++tot;
            for(int i=head[x];i;i=e[i].next){
                deg[e[i].to]--;
                if(deg[e[i].to]==0){
                    que.push(e[i].to);
                }
            }
        }
        if(tot!=n){//有环
            printf("NO\n");
        }else{
```

- P4017 最大食物链计数

  拓扑排序加 dp

- Directing Edges

拓扑排序 + 并查集 - B-Rank of Tetris > 为啥要用并查集是因为有个‘=’非常讨厌, 拓扑排序时我们无法处理这种情况，我们需要用并查集合点因为相等的点可以算一个点，用并查集通过 get 就可以找到代表这个点的点

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
using namespace std;
const int maxn = 10005;
int in[maxn];
vector<int> g[maxn];
int fa[maxn];
int n,m;
int a[maxn<<1],b[maxn<<1];
char ch[maxn<<1];
int get(int x){
    if(fa[x]==x)return x;
    else return fa[x] = get(fa[x]);//路径压缩
}
void merge(int x,int y){
    int p = get(x);
    int q = get(y);
    if(p!=q)fa[p] = q;
}
int ans[maxn];//其实这个题不大需要
void inits(){
    memset(ans,0,sizeof(ans));
    memset(in,0,sizeof(in));
    for(int i=0;i<=n;i++){
        g[i].clear();
    }
}
int main(){
    while(~scanf("%d%d",&n,&m)){
        inits();
        for(int i=0;i<n;i++)fa[i]=i;
        for(int i=0;i<m;i++){
            scanf("%d %c %d",&a[i],&ch[i],&b[i]);
            //cout<<endl<<a<<ch<<b;
            if(ch[i]=='='){
                merge(a[i],b[i]);//我们得记得提前合并然后在处理
            }
        }
        for(int i=0;i<m;i++){
            if(ch[i]=='<'){
```

```
            g[get(b[i])].push_back(get(a[i]));
            in[get(a[i])]++;
        }
        else if(ch[i]=='>'){
            g[get(a[i])].push_back(get(b[i]));
            in[get(b[i])]++;
        }
    }
    int nn=0;//记录合并之后的点数
    for(int i=0;i<n;i++){
        if(fa[i]==i)nn++;
    }
    //cout<<nn<<endl;
    queue<int>q;
    for(int i=0;i<n;i++){
        if(in[get(i)]==0&&i==get(i)){//这个点没有合并才可以算
            q.push(get(i));
        }
    }
    int ct=0,flag=0;
    while(!q.empty()){
        int f = q.front();
        //cout<<"k";
        int cnt = q.size();
        if(cnt>1){
            flag=1;
        }
        q.pop();
        ans[++ct] = f;
        for(int i=0;i<g[f].size();i++){
            int v = g[f][i];
            in[v]--;
            if(!in[v]){
                q.push(v);
            }
        }
    }
    if(ct<nn){
        printf("CONFLICT\n");//有环是矛盾
    }
    else {
        if(flag)printf("UNCERTAIN\n");//对内有两个是信息不足
```

```
            else printf("OK\n");
        }
    }
}
```

1.4 4. 欧拉通路，欧拉回路

欧拉通路: 通过图中每条边且只通过一次，并且经过每一顶点的通路

欧拉回路: 通过图中每条边且只通过一次，并且经过每一顶点的回路

有向图的基图：忽略有向图所有边的方向，得到的无向图称为该有向图的基图。

1.4.0.1 4.1. 无向图

> 设 G 是连通无向图，则称经过 G 的每条边一次并且仅一次的路径为欧拉通路；如果欧拉通路是回路（起点和终点是同一个顶点），则称此回路是欧拉回路具有欧拉回路的无向图 G 成为欧拉图

定理

无向图 G 存在欧拉通路的充要条件是：G 为连通图，并且 G 仅有两个奇度结点（度数为奇数的顶点）或者无奇度结点。

推论

(1) 当 G 是仅有两个奇度结点的连通图时，G 的欧拉通路必以此两个结点为端点；

(2) 当 G 是无奇度结点的连通图时，G 必有欧拉回路

(3) G 为欧拉图（存在欧拉回路）的充分必要条件是 G 为无奇度结点的连通图

1.4.0.2 4.2. 有向图

(1) 设 D 是有向图，D 的基图连通，则称经过 D 的每条边一次并且仅有一次的有向路径为有向欧拉通路

(2) 如果有向欧拉通路是有向回路，则称此有向回路为有向欧拉回路

(3) 具有有向欧拉回路的图 D 称为有向欧拉图

(有向图) 定理

> 有向图 D 存在欧拉通路的充要条件是：D 为有向图，D 的基图连通，并且所有顶点的出度与入度相等；或者除两个顶点外，其余顶点的出度与入度都相等，而这两个顶点中一个顶点的出度与入度之差为 1，另一个顶点的出度与入度之差为-1.

推论

(1) 当 D 除出、入度之差为 1，-1 的两个顶点之外，其余顶点的出度与入度相等时，D 的有向欧拉通路必以出、入度之差为 1 的顶点作为始点，以出、入度之差为-1 的顶点作为终点。

(2) 当 D 的所有顶点的出、入度都相等时，D 中存在有向欧拉回路。

(3) 有向图 D 为有向欧拉图的充要条件是 D 的基图为连通图，并且所有顶点的出、入度都相等。

1.4.1 4.1. 求解欧拉回路通过 dfs 或并查集求解

- [UVA-10054]https://vjudge.net/contest/399288#problem

  题意大致是这样：有一堆珠子，每个珠子有两个颜色，我们需要从中找出来一些珠子把它穿成一串，每两个珠子相对的颜色相同我们把每个珠子看整一个边，两个颜色看成节点，就是看看给定的边又没有欧拉回路既然问你有无欧拉回路那就欧拉定理判断有没有度数为奇数的点（如果是寻找欧拉通路的话需要两个或没有奇数度的点）如果全为偶数度的点，就肯定有欧拉回路，直接包搜就可以了，注意会又重边的情况，然后因为你也不知道 1-50 之内的那个点在图内使用，所以我们直接全搜一下，搜完一定是所有的点都用完了所以不会再搜了。(最后 for 循环的解释因为是一条路一路搜下去就可以了所以不用考虑回溯的 vis 问题，然后也不要考虑存路径，回溯是输出就好了

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
using namespace std;
const int maxn = 105;
int n;
int g[maxn][maxn];

int d[maxn];

int st,flag=0;
void dfs(int u){
    for(int v=1;v<=50;v++){
        if(g[u][v]){
            g[u][v]--;
            g[v][u]--;

            dfs(v);
            printf("%d %d\n",v,u);//注意顺序是倒过来的

        }
    }
}
void inits(){
    memset(g,0,sizeof(g));
    memset(vis,0,sizeof(vis));
    memset(d,0,sizeof(d));
    ans.clear();
    flag=0;
}
int main(){
```

```
    int t;
    cin>>t;
    for(int o=1;o<=t;o++){
        scanf("%d",&n);
        inits();
        printf("Case #%d\n",o);
        for(int i=0;i<n;i++){
            int u,v;
            scanf("%d%d",&u,&v);
            g[u][v]++;
            g[v][u]++;
            d[u]++;d[v]++;
        }
        for(int i=1;i<=50;i++){
            if(d[i]&1){
                flag=1;
                break;
            }
        }
        if(flag){
            printf("some beads may be lost\n\n");
        }
        else {
            for(int i=1;i<=50;i++){
                dfs(i);
            }
            cout<<endl;
        }

    }
}
```

1.4.1.0.1 并查集 + 欧拉回路

题目大意：这个是要我们找出存在几个欧拉回路寻找有几个欧拉回路设计并查集的操作，就是每个连通图上的点的特点并到根节点处理

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100005;
int fa[maxn];
int get(int x){
```

```
    if(x==fa[x])return x;
    else return get(fa[x]);
}
void merge(int x,int y){
    int p = get(x);
    int q = get(y);
    if(p!=q)fa[p] = q;
}
int in[maxn],vis[maxn],num[maxn];//num 存每个联通图中的奇数度的个数
vector<int>rot;
int main(){
    int n,m;
    while(~scanf("%d%d",&n,&m)){
        rot.clear();
        memset(num,0,sizeof(num));
        memset(vis,0,sizeof(vis));
        memset(in,0,sizeof(in));
        for(int i=1;i<=n;i++)fa[i]=i;
        for(int i=1;i<=m;i++){
            int x,y;
            scanf("%d%d",&x,&y);
            merge(x,y);
            in[x]++;in[y]++;
        }
        int ans = 0;
        for(int i=1;i<=n;i++){
            int k = get(i);
            if(k==i&&!vis[k]){
                vis[k] = 1;
                rot.push_back(k);//存跟根节点
            }
            if(in[i]&1)num[k]++;
        }
        for(int i=0;i<rot.size();i++){
            int k = rot[i];
            if(in[k]==0)continue;//如果这个节点是孤立的，就把他忽略掉，因为题目中说不存在
            if(num[k]==0){
                ans++;//欧拉图就派一队就好
            }
            else {
                ans+=num[k]/2;//如果不是一个欧拉图，那需要的就是节点的度为奇数的 1/2
            }
```

```
        }
        cout<<ans<<endl;
    }
}
```

1.5 5. 关于 LCA

1.5.1 5.1. 介绍有关算法的叫法

- 离线算法其实就是将多个询问一次性解决。离线算法往往是与在线算法相对的。例如求 LCA 的算法中，树上倍增属于在线算法，在对树进行 O(n) 预处理后，每个询问用 O(log_2n) 复杂度回答。而离线的 Tarjan 算法则是用 O(n+q) 时间将询问一次性全部回答。

1.5.2 5.2. 朴素算法

向上标记法

1.5.3 5.3. Tarjan

Tarjan 的基本思想其实的就是想上标记法的优化，使用并查集优化，就是把回溯的点的合并到父节点，难点其实在于记录闻讯和 dfs 函数的结构复杂度 $O(n+m)$

- How far away ?

```
#include <bits/stdc++.h>
#define ms(x,y) memset(x,y,sizeof(x))
using namespace std;
const int N = 40005;
int n,m;
struct edge{
    int v,next,w;
}e[N<<1];
int head[N],cnt;
int vis[N],dis[N];
int ans[N];
void add(int u,int v,int w){
    e[cnt].v = v;
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
```

```
int fa[N];
int get(int x){
    if(x==fa[x])return x;
    else return fa[x] = get(fa[x]);
}
vector<int>qu[N];
vector<int>qu_id[N];
void inits(){
    cnt = 0;
    for(int i = 0;i<=n;i++){
        fa[i] = i;ans[i] = 0;
        head[i] = -1;vis[i] = 0;dis[i] = 0;
        qu[i].clear();
        qu_id[i].clear();
    }
}
void add_query(int x,int y,int id){
    qu[x].push_back(y);qu_id[x].push_back(id);
    qu[y].push_back(x);qu_id[y].push_back(id);
}

void Tarjan(int x,int father){
    for(int i = head[x];~i;i = e[i].next){
        int v = e[i].v;
        int w = e[i].w;
        if(v==father)continue;
        dis[v] = dis[x]+w;
        Tarjan(v,x);
        fa[v] = x;
    }
    for(int i =0;i < qu[x].size();i++){
        int y = qu[x][i];
        if(vis[y] == 1){
            int lca = get(y);
            ans[qu_id[x][i]] = dis[x]+dis[y]-2*dis[lca];
        }
    }
    vis[x] = 1;//回溯的时候要标记这个点已经遍历过并且回溯了
}
int main(){
    int T;
    cin>>T;
```

```
    while(T--){
        scanf("%d%d",&n,&m);
        inits();
        for(int i = 0;i < n-1; 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 < m;i++){
            int x,y;
            scanf("%d%d",&x,&y);
            add_query(x,y,i);
        }
        Tarjan(1,0);
        for(int i = 0;i<m;i++){
            printf("%d\n",ans[i]);
        }
    }

}
```

- P3379【模板】最近公共祖先 (LCA)

  Tarjan(离线) 算法

因为完全是板子题, 题解基本就是直接的模板

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 500005;
int head[maxn],cnt=0;
int fa[maxn],vis[maxn],ans[maxn];
struct edge{
    int u,v,next;
}e[maxn<<1];
struct note { int node, id; }; //询问以结构体形式保存
vector<note> ques[maxn];
void add(int u,int v){
    e[cnt].u = u;
    e[cnt].v = v;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
int get(int x){
```

```
    if(fa[x]==x)return x;
    else return fa[x]=get(fa[x]);//路径压缩需要
}
void dfs(int u,int from){
    for(int i=head[u];~i;i=e[i].next){
        int v = e[i].v;
        if(v==from)continue;
        dfs(v,u);
        fa[v]=u;
    }
    int len = ques[u].size();
     for (int i = 0; i < len; i++)
        if (vis[ques[u][i].node])
            ans[ques[u][i].id] = get(ques[u][i].node);

    //访问完毕回溯
    vis[u] = 1;
}
int main(){
    memset(head,-1,sizeof(head));
    int n,m,s;
    cin>>n>>m>>s;
    for(int i=0;i<=n;i++)fa[i]=i;
    for(int i=0;i<n-1;i++){
        int x,y;
        scanf("%d%d",&x,&y);
        add(x,y);
        add(y,x);
    }
    for(int i=0;i<m;i++){
        int x,y;
        scanf("%d%d",&x,&y);
        note no;                    //说说这里，id 记录了第几个被询问
        no.node=y;                  //node 表示第 x 点与那个点有关联
        no.id=i;
        ques[x].push_back(no);
        no.node=x;                  //要入队两次
        no.id = i;
        ques[y].push_back(no);
    }
    dfs(s,0);
    for(int i=0;i<m;i++){
```

```
        cout<<ans[i]<<endl;
    }
}
```

1.5.4 5.4. 在线倍增 lca

复杂度：初始化 $O(n)$ 询讯 $O(logn)$

```
int n,m;
int dis[N];//这个是记录到根节点的距离（可以不要）
int d[N];//这个是记录层数（深度），用于 lca 向上倍增
int f[N][30];//倍增数组
int t ;
void bfs(int x){//遍历一遍预处理
    queue<int >q;
    q.push(x);
    d[x] = 1;
    while(!q.empty()){
        int now = q.front();
        q.pop();
        for(int i = head[now];~i;i = e[i].next){
            int v = e[i].v;
            int w = e[i].w;
            if(d[v])continue;
            d[v] = d[now]+1;
            dis[v] = dis[now] + w;
            f[v][0] = now;
            for(int i = 1;i < t; i++){
                f[v][i] = f[f[v][i-1]][i-1];
            }
            q.push(v);
        }
    }
}
int lca(int x, int y){
    if(d[x] > d[y])swap(x,y);
    for(int i = t;i >= 0;i--){
        if(d[f[y][i]] >= d[x])y = f[y][i];
    }
    if(x==y)return x;
    for(int i = t;i >= 0;i--){
        if(f[x][i]!=f[y][i]){
```

```
            x = f[x][i];
            y = f[y][i];
        }
    }
    return f[x][0];
}
```

1.6 6. 双连通分量与 Tarjan

1.6.1 6.1. 相关概念

1.6.1.1 6.1.1. 无向图

- 在无向图中，如果顶点 Vi 到顶点 Vj 有路径，则称顶点 Vi 和 Vj 连通。
- 如果无向图中任意两个顶点之间都连通，则称为连通图。
- 如果不是连通图，则图中的极大连通子图称为连通分量。

  重点区分：极大连通子图和极小连通子图
- 极大连通子图是无向图的连通分量，极大要求该连通子图包含其所有的边。
- 极小连通子图是既要保持图连通，又要使得边数最少的子图。

  进一步，到有向图中，概念就变为强连通，强连通图，强连通分量 #### 6.1.2. 有向图 * 在有向图中，如果从 Vi 到 Vj 和从 Vj 到 Vi 之间都有路径，则称这两个顶点是强连通的
- 若图中任何一对顶点都是强连通的，则称此图为强连通图
- 有向图中的极大强连通子图称为有向图的强连通分量

1.6.1.2 6.1.3. 基本的知识 判断图的连通性方法（DFS，BFS，并查集）

推荐并查集（其他的我也不会了呜呜 >.<）

- 还有强连通分量的一些代码讲解在师哥 ppt 上

1.6.2 6.2. 题解

- 牛客第 8 场 i 题

这个题对我来说好难呜呜呜，我甚至看不出来这是个图论题

官方的题解就是简单明了，强有力的说明了我是个傻子，就是把图一画，只要一个连通分量里有环，就可以全部都取到这些点，如果没有只能取连通分量点数-1 个点。

然而知道了这些的我依然不会做

那就讲一讲底下的代码的功能：* 用并查集去存图（并查集根节点可以代表这个连通分量发性质）* 维护了每一个连通分量的点个数，和边个数 * 解决了点开到了 1e9 普通数组存不下的问题，进行了特殊处理（我觉得是一种压缩方法）

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5+5;
int fa[maxn<<1],a[maxn],b[maxn],c[maxn<<1],sz[maxn<<1],edge[maxn<<1];
int cnt = 1;
int get(int x){
    if(fa[x]==x)return x;
    else return fa[x]=get(fa[x]);//要路径压缩
}
void merge(int x,int y){
    int p,q;
    p = get(x);
    q = get(y);
    if(p!=q){
        fa[p]=q;
    }
}
int main(){
    int t;
    scanf("%d",&t);
    for(int ii=1;ii<=t;ii++){
//      memset(sz,0,sizeof(sz));
//      memset(edge,0,sizeof(edge));
        int n;
        scanf("%d",&n);
        cnt=1;
        for(int i=1;i<=n;i++){
            scanf("%d%d",&a[i],&b[i]);
            c[cnt++]=a[i],c[cnt++]=b[i];
        }
        int nn;
        sort(c+1,c+cnt+1);
        nn = unique(c+1,c+cnt+1)-c-1;//去重后返回新的大小，这时 c 数组存储了所有出现的点对，并且排好了序去
 ↪  了重
        for(int i=1;i<=nn;i++){
            fa[i]=i;edge[i]=sz[i]=0;
        }
        for(int i=1;i<=n;i++){//二分查找
            a[i] = lower_bound(c+1,c+nn,a[i])-c;//这是 a[i] 变成了 c 数组中 a[i] 这个元素的位置
            b[i] = lower_bound(c+1,c+nn,b[i])-c;//同理
```

```
            merge(a[i],b[i]);//连接的其实是位置
        }

        for(int i=1;i<=nn;i++)sz[get(i)]++;//就是每一个点祖先节点大小加一这里还是下标的操作，每个点换成了它
        ↪ 的下标因为点大小 1e9 下标 2e5
        for(int i=1;i<=n;i++)edge[get(a[i])]++;//记录边，边就不能从每一个点记录，因为会有边相连成环，在并查
        ↪ 集存不了环，我们从开始每一次给出一个边就给这个点祖先节点的边数加一，注意 a[i] 是可以重复出现某一点
        ↪ 的这里可以改成 edge[get(b[i])]++，因为 a[i],b[i] 祖先节点是一样的

        int ans = 0;
        for(int i=1;i<=nn;i++){
            if(get(i)!=i)continue;
            if(sz[i]==edge[i]+1)ans+=(sz[i]-1);
            else ans+=sz[i];
        }
        printf("Case #%d: %d\n",ii,ans);
    }
}
```

1.6.3 6.3. 有向图 Tarjan 与强连通分量

求强连通分量，应该是不能用并查集去搞这个

变量说明

```
#define ms(a,v)  memset(a,v,sizeof(a))
int n,m;
const int maxn = 10005; //点数
int head[maxn],cnt = 0;
struct {
    int u,v,next;
}e[100005];
void add(int u,int v){
    e[cnt].u = u;
    e[cnt].v = v;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
int low[maxn],dfn[maxn],vis[maxn];//vis 数组是记录点是否在栈内  dfn 是记录每个点 dfs 序
stack<int> s;
int num = 0;//dfs 序计数，或者理解为时间戳
int lis_num = 0;//强连通分量的个数
int tag[maxn];//tag 是记录每个点的属于几号连通分量
```

初始化代码

```
void inits(){
    lis_num = 0;num = 0;cnt = 0;
    ms(head,-1);
    ms(vis,0);
    ms(tag,0);
    ms(dfn,0);
    ms(low,0);
}
```

Tarjan

```
void Tarjan(int now){
    s.push(now);//栈可以数组代替
    vis[now] = 1;
    dfn[now] = low[now]= ++num;
    for(int i=head[now];~i;i=e[i].next){
        int v = e[i].v;
        if(!dfn[v]){
            Tarjan(v);
            low[now]  = min(low[now],low[v]);
        }
        else if(vis[v]){
            low[now] = min(low[now],dfn[v]);
        }
    }
    if(dfn[now]==low[now]){//出栈
        lis_num++;
        int t;
        do{
            t = s.top();
            vis[t] = 0;
            tag[t] = lis_num;//这个可以没有如果不需要记录联通分量的序号
            s.pop();
        }while(t!=now);
    }
}
```

这个还可用于 DAG 的缩点（有别于并查集的缩点）例如：

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

```
        int sz=g[i].size();
        for(int j=0; j<sz; j++)
        {
            int v=g[i][j];
            if(color[v]!=color[i])
            {
                du[color[i]]++;
                //在这里可以建一个新的图
            }
        }
        cnt[color[i]]++;//统计每一个分量的点数
    }
```

1.6.4 6.4. 无向图求连通分量，割点（关节点），割边（桥）

```
vector<int>g[maxn];
int dfn[maxn],low[maxn];
int dep=0,child=0;
int cut[maxn];
int n,m;
void Tarjan(int u,int fa)
{
    dfn[u]=low[u]=++dep;
    for(int i=0;i<g[u].size();i++)
    {
        int v=g[u][i];
        if(!dfn[v])
        {
            Tarjan(v,u);
            low[u]=min(low[u],low[v]);
            //if(u==root) child++;    //对于根结点是否为割点的判定：记录子树个数
            if(low[v]>=dfn[u]){     //这里改为  low[v]>dfn[u] ，则 (u,v) 是一条割边
                cut[u]++;//或者用 iscut 可以判断是否为割点但是根节点得特判            //其他结点 u 若符合
 ↪ 该条件，u 就是割点

            }
        }
        else if(v!=fa) low[u]=min(low[u],dfn[v]);
    }
}
```

1.6.5 无向图割边判定（Tarjan）

注意成对变换，还有初始化

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
using namespace std;
const int N = 1005;
const int maxn = N*N+5;
struct edge{
    int v,next,w;
}e[maxn<<1];
int n,m;
int head[N],cnt;
int ans = INF;
void add(int u,int v,int w){
    e[cnt].v = v;
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
int dfn[N],low[N];
int num = 0;
void inits(){
    // memset(head,-1,sizeof head);
    for(int i = 0; i <= n;i++){
        dfn[i] = low[i] = 0;
        head[i] = -1;
    }
    cnt = 0;
    num = 0;
    ans = INF;
}

void Tarjan(int x,int in_edge){
    dfn[x] = low[x] = ++num;
    for(int i = head[x];~i;i=e[i].next){
        int v = e[i].v;
        if(!dfn[v]){
            Tarjan(v,i);
            low[x] = min(low[v],low[x]);
            if(low[v]>dfn[x]){
                ans = min(e[i].w,ans);
```

```
            }
        }
        else if(i!=(in_edge ^ 1))low[x] = min(low[x],dfn[v]);
    }
}
int main(){
    int tt = 1;
    while(1){
        scanf("%d%d",&n,&m);
        if(n==0&&m==0)break;
        if(tt!=1)printf("\n");
        tt++;
        inits();
        for(int i = 0;i < m; i++){
            int u,v, w;
            scanf("%d%d%d",&u,&v,&w);
            add(u,v,w);
            add(v,u,w);
        }
        Tarjan(1,-1);
        if(num!=n)printf("0");
        else if(ans==INF)printf("-1");
        else printf("%d",max(ans,1));

    }

}
```

```
#include<bits/stdc++.h>
#define ms(a,v)  memset(a,v,sizeof(a))
using namespace std;
const int maxn = 3e5+5;
vector<int>g[maxn];
int dfn[maxn],low[maxn];
int dep=0,child=0;
int cut[maxn];
int n,m;
void Tarjan(int u,int fa)
{
    dfn[u]=low[u]=++dep;
    for(int i=0;i<g[u].size();i++)
    {
```

```
        int v=g[u][i];
        if(!dfn[v])
        {
            Tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]){
                cut[u]++;//这里可以记录他有多少次成为割点的情况就是说去点这个点回多多少连通分量
            }
        }
        else if(v!=fa) low[u]=min(low[u],dfn[v]);
    }
}
void solve(){
    int ans = 0;
    for(int i=1;i<=n;i++){
        if(!dfn[i]){
            Tarjan(i,-1);
            ans++;//一共有多少连通分量
            cut[i]--;//根节点子树如果大于二才是割点
        }
    }
    for(int i=1;i<=n;i++){
        if(i!=n)printf("%d ",ans+cut[i]);
        else printf("%d ",ans+cut[i]);
    }

}

int main(){
    //inits();
    scanf("%d%d",&n,&m);
    for(int i=0;i<m;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    solve();
}
```

1.7 7. 基础的树上 dp 问题

- P1395 会议

为啥能联想到树上 dp 的问题，因为想到找树重心的事情（虽然树重心不是这么定义的），找发使用了树上 dp ### 7.1. 树的重心 * 树的重心是指，当前节点的子树最大节点数在所有节点的子树的最大节点数中最小。子树的意思：把这个节点删掉，下面会出现几棵树，他们都是这个节点的子树。

核心代码 (链式前行星版)：

```
//这个代码的核心思路是树上 dfs+dp
//深搜会首先找到树的叶子节点，并在开始更新了叶子节点为根树的 sz 是 1 叶子节点没有子树，然后利用回溯的时候更新
 ↪ son (dp) 数组

int sz[maxn]; //指 i 节点为根的子树大小是多少
int son[maxn];//指 i 节点子树的最大节点数
int rt=0;//记录树的重心
int sum;//总结点的个数   初始化为 n 就好了
void findrt(int u,int fa){//开始传随便一个点作为根结点然后传 0 也好-1 也好作为它不存在的父节点
    sz[u] = 1;
    son[u] = 0;
    for(int i=head[u];~i;i=e[i].next){
        int v = e[i].v;
        if(v==fa)continue;
        findrt(v,u);
        sz[u]+=sz[v];//回溯更新了 sz
        son[u] = max(son[u],sz[v]);
    }
    son[u] = max(son[u],sum-sz[u]);//因为我们求的是向下的子树，子树还包括它父亲节点那边的树，所以只需要
 ↪ sum-sz[u] 就能求得上面的子树了
//  cout<<son[u]<<endl;
    if(son[u]<=son[rt]){
        if(son[u]==son[rt]){
            rt = min(rt,u);//这个其实是题目需要因为重心有多个，这里我要的是节点最小的那个
        }
        else rt=u;
    }
}
```

1.8 8. 树直径问题

1.8.1 8.1. 树上 dfs

1.8.1.1 8.1.1. ——用于求解树直径

说明：对于树的直径问题，如果要求数直径的左右端点和树直径的中点用树 dp 不易求，解法是两次 dfs 或者两次 bfs 复杂度 $O(n)$

注意：不能在存在负边的情况下求直径

大神解释：因为在第一次遍历后，有的边变为了 -1，然后你第一次 bfs 或 dfs 会因为选取的起点 s 不同，而导致求出不同的最远点，那么你在用这个不一定正确的最远点求出的直径也可能是错误的

- [P5536【XR-3】核心城市] https://www.luogu.com.cn/problem/P5536 此题我认为不是树直径的板子题，需要一些思考
- [题解]https://www.luogu.com.cn/problem/solution/P5536 里面对于 dfs 求树直径的原理有解释

下面是模板 (需要理解)

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5+5;
int len,lzj,rzj,zd;
int deep[maxn],father[maxn],maxdeep[maxn],ans[maxn];
struct {
    int u,v,next;
}e[maxn<<1];
int head[maxn],cnt=0;
void add(int u,int v){
    e[cnt].u = u;
    e[cnt].v = v;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
void dfs1(int cur,int fa){
    if(deep[cur]>len){
        len = deep[cur];
        lzj = cur;
    }
    for(int i=head[cur];~i;i=e[i].next){
        int v = e[i].v;
        if(v==fa)continue;
        deep[v]=deep[cur]+1;
        dfs1(v,cur);
    }
}
void dfs2(int cur,int fa){
    if(deep[cur]>len){
        len = deep[cur];
        lzj = cur;
```

```
    }
    for(int i=head[cur];~i;i=e[i].next){
        int v = e[i].v;
        if(v==fa)continue;
        deep[v]=deep[cur]+1;
        father[v]=cur;
        dfs2(v,cur);
    }
}
void dfs3(int cur,int fa){
    maxdeep[cur]=deep[cur];
    for(int i=head[cur];~i;i=e[i].next){
    //  cout<<"kk"<<endl;
        int v = e[i].v;
        if(v==fa)continue;
        deep[v]=deep[cur]+1;
        dfs3(v,cur);
        maxdeep[cur]=max(maxdeep[cur],maxdeep[v]);
    }
}
bool cmp(int x,int y){
    return x>y;
}
int main(){
    memset(head,-1,sizeof(head));
    int n,k;
    scanf("%d%d",&n,&k);
    for(int i=0;i<n-1;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        add(u,v);
        add(v,u);
    }
    dfs1(1,0);
    len = 0;
    memset(deep,0,sizeof(deep));
    dfs2(lzj,0);
    for(int i=0;i<(len+1)/2;i++){
        lzj=father[lzj];
    }
    memset(deep,0,sizeof(deep));
    dfs3(lzj,0);
```

```
    for(int i=1;i<=n;i++){
        ans[i]=maxdeep[i]-deep[i];
    //  cout<<ans[i]<<endl;
    }
    sort(ans+1,ans+n+1,cmp);

    cout<<ans[k+1]+1;
}
```

1.8.2 8.2. 树形 dp

我们从如下角度考虑：遍历树上每一个点（从叶子节点开始，并把它看作根节点）维护数组 D，数组 D[x] 表示 x 到子树最远节点的距离，我们一定是从 v（x 的子节点）转移过来的，有 D[x] = max(D[x],D[v]+w)，然后就好办了，如果我们把 x 看作根节点，直径就是 D[x] 加一个次长路（到最子节点第二远的距离）然后有 ans = max(ans,D[x]+D[v]+w) 复杂度 $O(n)$ **可用于有负权边求解直径的问题**板子

```
int D[N];
int ans = 0;
void dp(int x,int fa){
    for(int i  = head[x];~i;i = e[i].next){
        int v = e[i].v;
        int w = e[i].w;
        if(v==fa)continue;
        dp(v,x);
        ans = max(ans,D[x]+D[v]+w);
        D[x] = max(D[x],D[v]+w);
    }
}
```

1.9 9. 匹配

1.9.1 9.1. 二分图

判断二分图的方法 - dfs

```
bool dfs(int u,int col){//col 初始化 1 或者 2 都可
    color[u] = col;
    for(int i = head[u];~i;i = e[i].next){
        int v = e[i].v;
        if(color[u]==color[v])return false ;
        if(!color[v]){
```

```
            if(!dfs(v,3-col))return false ;
        }
    }
    return true;
}
```

```
for(int i = 0;i < n;i++){
    if(!color[i])dfs(i,1);
}
```

1.10 10. 匈牙利算法

匈牙利算法的复杂度是 $O(VE)$

```
bool dfs(int x){
    if(a[x])return false ;
    for(int i =0;i<g[x].size();i++){
        int v = g[x][i];
        if(a[v])continue;
        if(!vis[v]){
            vis[v] = 1;
            if(match[v]==-1||dfs(match[v])){
                match[x] = v;
                match[v] = x;
                return true;
            }
        }
    }
    return false ;
}

//搭配主函数内部的
for(int i = 1;i <= n*n;i++){//穿进来所有的顶点
        if(match[i]!=-1)continue;//记得初始化
        memset(vis,0,sizeof(vis));
        if(dfs(i)){
            ans++;
        }
    }
```

第二种板子，用法稍有不同，此方法邻接矩阵比较好用，并且要注意建图

```
bool dfs(int x){
    for(int i = 1;i <= n*m; i++){
        if(!mp[x][i])continue;
        if(!vis[i]){
            vis[i] = 1;
            if(match[i]==0||dfs(match[i])){
                //match[x] = i;
                match[i] = x;
                return true;
            }
        }
    }
    return false ;
}

//主函数内部
memset(match,0,sizeof(match));//这边来说是只传左部
for(int i = 1;i <= m ; i++){
    //if(match[i]!=0)continue;
    memset(vis,0,sizeof(vis));
    if(dfs(i))ans++;
}
```

1.10.1 10.1. HK 算法

用于求解二分图最大匹配问题，复杂度 $O(\sqrt{n}m)$ - 要比匈牙利快 - 关键是建图，如何找到左部右部，然后左部向右部连有向边

- Prime Independence

要注意几点：1. 建图是左部连向右部 2. 注意初始化左部右部的节点数

```
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
using namespace std;
const int maxn = 5e5+5;
const int N = 4e4+5;
typedef long long ll;
int lt[N],rt[N];
struct edge{
    int v,next;
}e[maxn<<1];
int head[N],cnt = 0;
```

```
void add(int u,int v){
    e[cnt].v = v;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
const int MAXN = 500005;

// int match[N];
bool used[N];
int p, n;
int nx, ny, dis; //nx,ny 分别是左点集和右点集的点数
int dx[N], dy[N], cx[N], cy[N]; //dx,dy 分别维护左点集和右点集的标号
//cx 表示左点集中的点匹配的右点集中的点，cy 正好相反
bool bfs() //寻找增广路径集，每次只寻找当前最短的增广路
{
    queue<int> que;
    dis = INF;
    memset(dx, -1, sizeof dx);
    memset(dy, -1, sizeof dy);
    for(int i = 1; i <= nx; i++)//将未遍历的节点入队，并初始化次节点距离为 0
    if(cx[i] == -1)
    {
        que.push(i);
        dx[i] = 0;
    }
    while(!que.empty())
    {
        int u = que.front();
        que.pop();
        if(dx[u] > dis) break;
        for(int i = head[u]; ~i; i = e[i].next)
        {
            int v = e[i].v;
            if(dy[v] == -1)
            {
                dy[v] = dx[u] + 1;
                if(cy[v] == -1) dis = dy[v]; //找到了一条增广路，dis 为增广路终点的标号
                else dx[cy[v]] = dy[v] + 1,que.push(cy[v]);
            }
        }
    }
    return dis != INF;
```

```
}
int dfs(int u)
{
    for(int i = head[u]; ~i ; i = e[i].next)
    {
        int v = e[i].v;
        if(! used[v] && dy[v] == dx[u] + 1) //如果该点没有被遍历过并且距离为上一节点 +1
        {
            used[v] = true;
            if(cy[v] != -1 && dy[v] == dis) continue; //u 已被匹配且已到所有存在的增广路终点的标号，再递
             ↪ 归寻找也必无增广路，直接跳过
            if(cy[v] == -1 || dfs(cy[v]) )
            {
                cy[v] = u,cx[u] = v;
                return 1;
            }
        }
    }
    return 0;
}
int hopcroft_karp()
{
    int res = 0;
    memset(cx, -1, sizeof cx);
    memset(cy, -1, sizeof cy);
    while(bfs())
    {
        memset(used, 0, sizeof used);
        for(int i = 1; i <= nx; i++)
        if(cx[i] == -1)res += dfs(i);
    }
    return res;
}

int pos[MAXN];
int num[N];
int prime[MAXN];
void getprime(){
    memset(prime,0,sizeof prime);
    for(int i = 2; i <= MAXN; i++){
        if(!prime[i]) prime[++prime[0]] = i;
        for(int j = 1 ;j <= prime[0] && prime[j] <= MAXN / i; j++){
```

```
            prime[prime[j] * i] = 1;
            if(i % prime[j] == 0) break;
        }
    }
}

int factor[105][2];
int fatcnt;
int sum;
int getFactors(int x) {
    fatcnt = 0;
    int tmp = x;
    for (int i = 1; prime[i] <= tmp / prime[i]; i++) {
        factor[fatcnt][1] = 0;
        if (tmp % prime[i] == 0) {
            factor[fatcnt][0] = prime[i];
            while (tmp % prime[i] == 0) {
                factor[fatcnt][1] ++;
                tmp /= prime[i];
                sum++;
            }
            fatcnt++;
        }
    }
    if(tmp != 1) {
        factor[fatcnt][0] = tmp;
        factor[fatcnt++][1] = 1;
        sum++;
    }
    return fatcnt;
}


void inits(){
    for(int i = 0;i<=n;i++){
        head[i] = -1;
    }
    memset(pos,0,sizeof pos);
    cnt = 0;
}

int main(){
```

```
    int t;
    scanf("%d",&t);
    int tt = 0;
    getprime();
    while(t--){
        scanf("%d",&n);
        inits();
        for(int i = 1;i <= n; i++){
            scanf("%d",&num[i]);
            pos[num[i]] = i;
        }

        for (int i = 1; i <= n; i++) {
           sum = 0;
           int pnum = getFactors(num[i]);
           for(int k = 0; k < pnum; k++) {
               if(pos[num[i] / factor[k][0]] != 0) {
                   if(sum & 1)
                        add(pos[num[i]],pos[num[i] / factor[k][0]]);
                   else
                       add(pos[num[i] / factor[k][0]], pos[num[i]]);
               }
           }
        }
        nx = ny = n;
        printf("Case %d: %d\n",++tt,n - hopcroft_karp());
    }

}
```

1.10.2 10.2. 二分图最优匹配（KM 算法）

- 适用于边权最大的匹配就是完美匹配的情况
- 思想就是顶点标号的思想，不断扩展可行的增广路
- 复杂度 $O(n^2 * m)$

```
const int N = 105;
double w[N][N];//我写的这个题是 double 的如果是 int 类型也可以写 int
double la[N],lb[N];//标号值
bool va[N],vb[N];//是否在增广路上出现过，做标记
int match[N];
int n;
```

```
double delta;
double upd[N];//递归是记录更新值
bool dfs(int x){
    va[x] = 1;
    for(int y = 1;y <= n; y++){
        if(!vb[y]){
            if(fabs(la[x] + lb[y] - w[x][y] ) <= eps ){
                vb[y] = 1;
                if(!match[y]||dfs(match[y])){
                    match[y] = x;
                    return true;
                }
            }
            else upd[y] = min(upd[y],la[x] + lb[y] - w[x][y]);

        }
    }
    return false ;
}


void KM(){
    for(int i = 1; i <= n; i++){
        la[i] = -(1<<30);//-inf
        lb[i] = 0;
        for(int j = 1;j <= n; j++){
            la[i] = max(la[i],w[i][j]);
        }
    }

    for(int i = 1 ;i <= n; i++){
        while(true){
            memset(va,0,sizeof(va));
            memset(vb,0,sizeof(vb));
            for(int j = 1; j <= n; j++)upd[j] = 1e9;
            if(dfs(i))break;
            delta = 1e10;

            for(int j = 1;j <= n; j++){
                if(!vb[j])delta = min(delta,upd[j]);
            }
            for(int j = 1;j <= n; j++){
```

```
                if(va[j])la[j] -= delta;
                if(vb[j])lb[j] += delta;
            }
        }
    }
    //一下是求左部映射到右部，如果不想这么做就要反向建左右部
    for(int i = 1;i <= n; i++){
        rm[match[i]] = i;
    }
    for(int i = 1;i <= n; i++){
        printf("%d\n",rm[i]);
    }
}
```

- KM 优化的详细讲解

板子

```
double w[N][N]; // 边权
double la[N], lb[N], upd[N]; // 左、右部点的顶标
bool va[N], vb[N]; // 访问标记：是否在交错树中
int match[N]; // 右部点匹配了哪一个左部点
int last[N]; // 右部点在交错树中的上一个右部点，用于倒推得到交错路
int n;

bool dfs(int x, int fa) {
    va[x] = 1;
    for (int y = 1; y <= n; y++)
        if (!vb[y])
            if (fabs(la[x] + lb[y] - w[x][y]) < eps) { // 相等子图
                vb[y] = 1; last[y] = fa;
                if (!match[y] || dfs(match[y], y)) {
                    match[y] = x;
                    return true;
                }
            }
            else if (upd[y] > la[x] + lb[y] - w[x][y] + eps) {
                upd[y] = la[x] + lb[y] - w[x][y];
                last[y] = fa;
            }
            return false;
}
```

```
void KM() {
    for (int i = 1; i <= n; i++) {
        la[i] = -1e100;
        lb[i] = 0;
        for (int j = 1; j <= n; j++)
            la[i] = max(la[i], w[i][j]);
    }
    for (int i = 1; i <= n; i++) {
        memset(va, 0, sizeof(va));
        memset(vb, 0, sizeof(vb));
        for (int j = 1; j <= n; j++) upd[j] = 1e10;
        // 从右部点 st 匹配的左部点 match[st] 开始 dfs, 一开始假设有一条 0-i 的匹配
        int st = 0; match[0] = i;
        while (match[st]) { // 当到达一个非匹配点 st 时停止
            double delta = 1e10;
            if (dfs(match[st], st)) break;
            for (int j = 1; j <= n; j++)
                if (!vb[j] && delta > upd[j]) {
                    delta = upd[j];
                    st = j; // 下一次直接从最小边开始 DFS
                }
            for (int j = 1; j <= n; j++) { // 修改顶标
                if (va[j]) la[j] -= delta;
                if (vb[j]) lb[j] += delta; else upd[j] -= delta;
            }
            vb[st] = true;
        }
        while (st) { // 倒推更新增广路
            match[st] = match[last[st]];
            st = last[st];
        }
    }
}
```

1.11 11. 最优带权匹配

- Jewels

KM 板子题，是最优带权匹配，匹配最大，然后转换一下转成最小就可以了，板子是在 oiwiki 上嫖来的

注意这个模版定点标号是从 0 开始的

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll maxn = 1e12;//这个是一个上限，用来求最小的时候减去的
template <typename T>
struct hungarian {  // km
  int n;
  vector<int> matchx;  // 左集合对应的匹配点
  vector<int> matchy;  // 右集合对应的匹配点
  vector<int> pre;     // 连接右集合的左点
  vector<bool> visx;   // 拜访数组 左
  vector<bool> visy;   // 拜访数组 右
  vector<T> lx;
  vector<T> ly;
  vector<vector<T> > g;
  vector<T> slack;
  T inf;               //极大值
  T res;
  queue<ll> q;
  int org_n;
  int org_m;

  hungarian(int _n, int _m) {
    org_n = _n;
    org_m = _m;
    n = max(_n, _m);
    inf = numeric_limits<T>::max();
    // inf = 1LL<<62;                  //这边需要修改我们的极大值
    res = 0;
    g = vector<vector<T> >(n, vector<T>(n));
    matchx = vector<int>(n, -1);
    matchy = vector<int>(n, -1);
    pre = vector<ll>(n);
    visx = vector<bool>(n);
    visy = vector<bool>(n);
    lx = vector<T>(n, -inf);
    ly = vector<T>(n);
    slack = vector<T>(n);
  }

  void addEdge(int u, int v, ll w) {    //构造函数
    g[u][v] = max(w, 0LL);  // 负值还不如不匹配 因此设为 0 不影响
```

```
}

bool check(int v) {
  visy[v] = true;
  if (matchy[v] != -1) {
    q.push(matchy[v]);
    visx[matchy[v]] = true;  // in S
    return false;
  }
  // 找到新的未匹配点 更新匹配点 pre 数组记录着" 非匹配边" 上与之相连的点
  while (v != -1) {
    matchy[v] = pre[v];
    swap(v, matchx[pre[v]]);
  }
  return true;
}

void bfs(int i) {              //匈牙利
  while (!q.empty()) {
    q.pop();
  }
  q.push(i);
  visx[i] = true;
  while (true) {
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int v = 0; v < n; v++) {
        if (!visy[v]) {
          T delta = lx[u] + ly[v] - g[u][v];
          if (slack[v] >= delta) {
            pre[v] = u;
            if (delta) {
              slack[v] = delta;
            } else if (check(v)) {  // delta=0 代表有机会加入相等子图 找增广路
                                    // 找到就 return 重建交错树
              return;
            }
          }
        }
      }
    }
```

```
    // 没有增广路 修改顶标
    T a = inf;
    for (int j = 0; j < n; j++) {
      if (!visy[j]) {
        a = min(a, slack[j]);
      }
    }
    for (int j = 0; j < n; j++) {
      if (visx[j]) {  // S
        lx[j] -= a;
      }
      if (visy[j]) {  // T
        ly[j] += a;
      } else {  // T'
        slack[j] -= a;
      }
    }
    for (int j = 0; j < n; j++) {
      if (!visy[j] && slack[j] == 0 && check(j)) {
        return;
      }
    }
  }
}

void solve() {
  // 初始顶标
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      lx[i] = max(lx[i], g[i][j]);
    }
  }

  for (int i = 0; i < n; i++) {
    fill(slack.begin(), slack.end(), inf);
    fill(visx.begin(), visx.end(), false);
    fill(visy.begin(), visy.end(), false);
    bfs(i);
  }

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

```
      if (g[i][matchx[i]] > 0) {
        res += g[i][matchx[i]];
      } else {
        matchx[i] = -1;
      }
    }
    cout << maxn*n - 1LL*res << "\n";
    // for (int i = 0; i < org_n; i++) {
    //   cout << matchx[i] + 1 << " ";
    // }
    // cout << "\n";
  }
};
int main(){
    int n;
    scanf("%d",&n);
    hungarian<ll> km(n,n);
    for(int i = 0 ;i < n;i++){
        long long x,y,z,v;
        scanf("%lld%lld%lld%lld",&x,&y,&z,&v);
        for(int j = 0;j < n; j++){
            km.addEdge(i,j ,maxn-(x*x + y*y + 1LL*(z+j*v)*(z+j*v)));
        }
    }
    km.solve();

}
```

1.11.1 11.1. 最大团

求一个图的最大图团模版，邻接矩阵存图，复杂度 $O(n^3)$ 当然可以求补图的最大独立集但是时间复杂度太高了对于二分图而言，最大独立集 = 定点数 - 最大匹配数二分图顶点的个数如到达 4e4，用匈牙利算法会严重超时，可以使用优化的算法——Hopcroft-Krap 算法，邻接表存图时，时间复杂度是 $O(V\sqrt{E})$

下面是对与普通图来讲的

```
const int maxn = 105;

int mx;//最大团数 (要初始化为 0)
int vis[maxn], tuan[maxn];
int can[maxn][maxn];//can[i] 表示在已经确定了经选定的 i 个点必须在最大团内的前提下还有可能被加进最大团的结点
↪ 集合
```

```
int num[maxn];//num[i] 表示由结点 i 到结点 n 构成的最大团的结点数
bool g[maxn][maxn];//邻接矩阵 (从 1 开始)
int n, m;

bool dfs(int tot, int cnt) {
    if(tot == 0) {
        if(cnt > mx) {
            mx = cnt;
            for(int i = 0; i < mx; i++) {
                tuan[i] = vis[i];
            }
            return true;
        }
        return false;
    }
    for(int i = 0; i < tot; i++) {
        if(cnt + (tot - i) <= mx) return false;
        if(cnt + num[can[cnt][i]] <= mx) return false;
        int k = 0;
        vis[cnt] = can[cnt][i];
        for(int j = i + 1; j < tot; j++) {
            if(g[can[cnt][i]][can[cnt][j]]) {
                can[cnt + 1][k++] = can[cnt][j];
            }
        }
        if(dfs(k, cnt + 1)) return false;
    }
    return false;
}

void maxclique() {
    mx = 1;
    for(int i = n; i >= 1; i--) {
        int k = 0;
        vis[0] = i;
        for(int j = i + 1; j <= n; j++) {
            if(g[i][j]) {
                can[1][k++] = j;
            }
        }
        dfs(k, 1);
        num[i] = mx;
```

```
    }
}
```

1.12 12. 最大流问题

- P3376 【模板】网络最大流

题解：模板直接过，把 int 改 long long

```
#include <bits/stdc++.h>
const long long inf=1e18;//找一个数据范围大的数
typedef long long ll;
using namespace std;
int n,m;
const int maxM = 120005<<1;//边数注意要乘二
const int maxN = 1205;//点数
class Graph
{
private:
    int cnt;
    int Head[maxN];
    int Next[maxM];
    ll W[maxM];
    int V[maxM];
    ll Depth[maxN];
    int cur[maxN];//cur 就是记录当前点 u 循环到了哪一条边
public:
    int s,t;
    void init()
        {
            cnt=-1;
            memset(Head,-1,sizeof(Head));
            memset(Next,-1,sizeof(Next));
        }
    void _Add(int u,int v,ll w)//链式前向星
        {
            cnt++;
            Next[cnt]=Head[u];
            Head[u]=cnt;
            V[cnt]=v;
            W[cnt]=w;
```

```
    }
void Add_Edge(int u,int v,ll w)
    {
        _Add(u,v,w);
        _Add(v,u,0);
    }
ll dfs(int u,ll flow)
    {
        if (u==t)
            return flow;
        for (int& i=cur[u];i!=-1;i=Next[i])//注意这里的 & 符号，这样 i 增加的同时也能改变 cur[u] 的
        ↪ 值，达到记录当前弧的目的
        {
            if ((Depth[V[i]]==Depth[u]+1)&&(W[i]!=0))
            {
                ll di=dfs(V[i],min(flow,W[i]));
                if (di>0)
                {
                    W[i]-=di;
                    W[i^1]+=di;
                    return di;
                }
            }
        }
        return 0;
    }
int bfs()
    {
        queue<int> Q;
        while (!Q.empty())
            Q.pop();
        memset(Depth,0,sizeof(Depth));
        Depth[s]=1;
        Q.push(s);
        do
        {
            int u=Q.front();
            Q.pop();
            for (int i=Head[u];i!=-1;i=Next[i])
                if ((Depth[V[i]]==0)&&(W[i]>0))
                {
                    Depth[V[i]]=Depth[u]+1;
```

```
                    Q.push(V[i]);
                }
            }
            while (!Q.empty());
            if (Depth[t]>0)
                return 1;
            return 0;
        }
    ll Dinic()
        {
            ll Ans=0;
            while (bfs())
            {
                for (int i=1;i<=n;i++)//每一次建立完分层图后都要把 cur 置为每一个点的第一条边
                    cur[i] = Head[i];
                while (ll d=dfs(s,inf))
                {
                    Ans+=d;
                }
            }
            return Ans;
        }
};

int main(){
    Graph g;
    scanf("%d%d%d%d",&n,&m,&g.s,&g.t);
    g.init();
    for(int i=0;i<m;i++){
        int u,v;
        ll w;
        scanf("%d%d%lld",&u,&v,&w);
        g.Add_Edge(u,v,w);
    }
    cout<<g.Dinic();

}
```

1.12.1 12.1. 网络流初步

蓝书 P440 知识点汇总

- 网络流的定义
  - 三个性质
- 最大流
  - 如何映射到匹配问题（二分图）
  - 解决多重匹配问题
- E-K 算法
  - 注意这个反向边更新策略
  - $O(nm^2)$ 处理 $1e4$ 以下
- D-k 算法
  - 残量网络
  - bfs
  - dfs
  - $O(m\sqrt{n})$ 大约处理到 $1e5$
- 必须边和可行边
  - 定义（任何匹配方案都包括的边是必须边，至少一个包括的边是可行边）
  - 加源点汇点，源点到左部，汇点到右部，设边权，跑一边最大流
  - 注意跑完了之后剩下残联网络，残量网络的匹配边正向 0 反向 1，要会判断匹配边和非匹配边
  - Tarjan 在残量网络上跑，判断可行边和必须边
- 最小割
  - 等于最大流
  - 点边转化，（有向无向图）
  - 问题模型 有 n 个物品和两个集合，如果将一个物品放入 a 集合会花费 a_i，放入 b 集合会花费 b_i；还有若干个形如 u_i, v_i, w_i 限制条件，表示如果 u_i 和 v_i 同时不在一个集合会花费 w_i 。每个物品必须且只能属于一个集合，求最小的代价。
- 费用流
  - 关系转化
    - 最大匹配——最大带权匹配
    - 最大流——最小费用最大流
  - E-K 算法转化
    - 将 bfs 换成 SPFA

详细内容请看oiwiki-网络流部分

1.12.2 12.2. 最小割

- 例题Path

不能说是裸的板子把，也算是半裸的板子。题意：就是有有一个人想从 1 走到 n, 然后又有一个人前来阻拦，然后就是要截住第一个人的去路，注意：第一个人走的是最短路，但是最短路可能有很多条，第二个人不知道走那一条，所以只能把所有的最短路都截住。开始想错了，想用 dijskra 直接转移状态，但是突然又想到了我们单单把所有的最短路经过的边和点提出来，结果图就成了一个新图。我们就是要把新图所有的路全部堵死——最小割

```
const int N = 1e4 + 5,M = 2e4+5;//N 表示点数，M 表示边数，记得要开两倍，注意加边
void add(int u,int v,ll w){
```

```
    e[cnt].v = v;
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
int  s, t, dep[N], cur[N];
int bfs(int s, int t) {
    memset(dep, 0, sizeof(dep));
    memcpy(cur, head, sizeof(head));
    queue<int> q;
    q.push(s), dep[s] = 1;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i; i = e[i].next) {
            int v = e[i].v;
            if (e[i].w && !dep[v]) q.push(v), dep[v] = dep[u] + 1;
        }
  }
  return dep[t];
}
ll dfs(int u, int t, ll flow) {
    if (u == t) return flow;
    ll ans = 0;
    for (int &i = cur[u]; ~i && ans < flow; i = e[i].next) {
        int v = e[i].v;
        if (e[i].w && dep[v] == dep[u] + 1) {
            ll x = dfs(v, t, min(e[i].w, flow - ans));
            if (x)e[i].w -= x, e[i ^ 1].w += x, ans += x;
        }
    }
    if (ans < flow) dep[u] = -1;
    return ans;
}
ll dinic(int s, int t) {
    ll ans = 0;
    while (bfs(s, t)) {
        ll x;
        while ((x = dfs(s, t, 1LL << 60))) ans += x;
    }
    return ans;
}
```

AC 代码:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 2e4+5;
const int N = 1e4 + 5,M = 2e4+5;
struct edge{
    int v,next;
    ll w;
}e[maxn];
int head[N],cnt;
void add(int u,int v,ll w){
    e[cnt].v = v;
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
ll dis[maxn];
struct Node {
    int id;
    ll d;
    bool operator < (const Node x)const{
        return d>x.d;
    }
};
priority_queue<Node>q;
int n,m;
int vis[N];
ll ans ;
struct Edge{
    int u,v;
    ll w;
}edge[maxn];
vector<Edge>ne[N];
void dijkstra(int x){
    while(!q.empty())q.pop();
    for(int i = 0;i<=n;i++){
        ne[i].clear();
        dis[i] = (1LL<<60);
        vis[i] = 0;
    }
    Node s = Node{x,0};
```

```
    dis[x] = 0;
    q.push(s);
    while(!q.empty()){
        Node now = q.top();q.pop();
        int u = now.id;
        if(vis[u])continue;
        vis[u] = 1;
        for(int i = head[u];~i;i=e[i].next){
            int v = e[i].v;
            ll w = e[i].w;
            if(dis[u] + w < dis[v]){
int  s, t, dep[N], cur[N];

int bfs(int s, int t) {
    memset(dep, 0, sizeof(dep));
    memcpy(cur, head, sizeof(head));
    queue<int> q;
    q.push(s), dep[s] = 1;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i; i = e[i].next) {
            int v = e[i].v;
            if (e[i].w && !dep[v]) q.push(v), dep[v] = dep[u] + 1;
        }
  }
  return dep[t];const int N = 1e4 + 5,M = 2e4+5;
}
ll dfs(int u, int t, ll flow) {
    if (u == t) return flow;
    ll ans = 0;
    for (int &i = cur[u]; ~i && ans < flow; i = e[i].next) {
        int v = e[i].v;
        if (e[i].w && dep[v] == dep[u] + 1) {
            ll x = dfs(v, t, min(e[i].w, flow - ans));
            if (x)e[i].w -= x, e[i ^ 1].w += x, ans += x;
        }
    }
    if (ans < flow) dep[u] = -1;
    return ans;
}
ll dinic(int s, int t) {
```

```
    ll ans = 0;
    while (bfs(s, t)) {
        ll x;
        while ((x = dfs(s, t, 1LL << 60))) ans += x;
    }
    return ans;
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        ans = (1LL<<60);
        memset(head,-1,sizeof head);cnt = 0;
        scanf("%d%d",&n,&m);
        for(int i = 0;i < m;i++){
            int u,v;ll w;
            scanf("%d%d%lld",&u,&v,&w);
            add(u,v,w);
        }
        dijkstra(1);

        memset(head,-1,sizeof head);cnt = 0;
        for(int j = 1;j <= n;j++){
            for(int i = 0;i < (int)ne[j].size();i++){
                Edge tmp  = ne[j][i];
                add(tmp.u,tmp.v,tmp.w);
                add(tmp.v,tmp.u,0);//注意加反向边，虽然是有向图
            }
        }
        printf("%lld\n",dinic(1,n));
    }
}
```