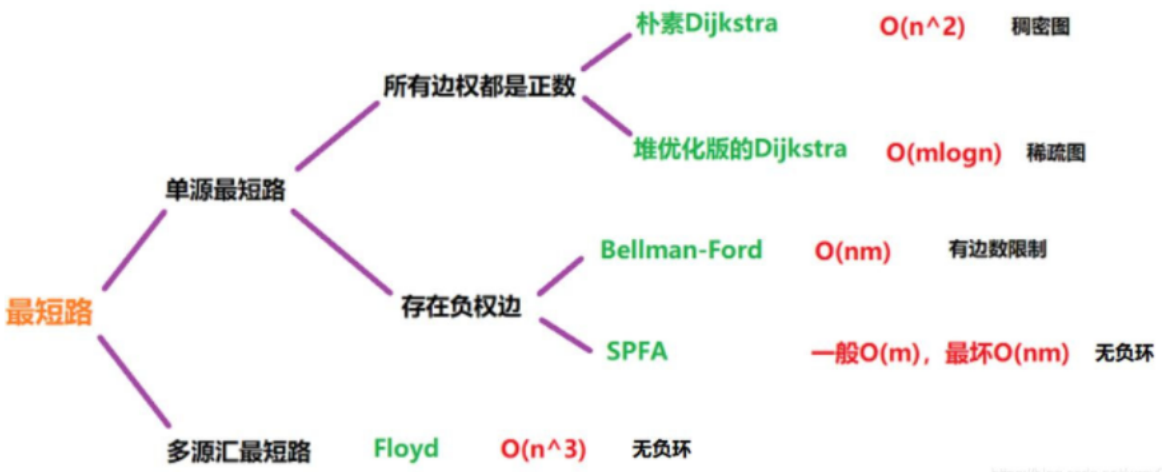
最短路
dijistra
spfa
floyed
差分约束
k短路
最小生成树
prim
kruskal
朱刘算法(有向图最小生成树)
kruskal重构树
匹配
霍尔定理
二分匹配
二分图最大权匹配
一般图最大匹配
一般图最大权匹配
最大团
网络流
dinic算法
预流推进hlpp
费用流
树链
LCA
树链剖分
Tarjan
强连通分量
点双连通分量(割点)
边双连通分量(割边)
2-sat问题
Cayley公式和prufer序列

最短路



以下算法使用时要改inf

dijistra

朴素版$O(n^2)$

```
const int inf=0x3f3f3f3f;
int c[N],vis[N];
vector<pair<int,int>>v[N];
void dijkstra(int x){
    fill(c,c+n+1,inf);
    fill(vis,vis+n+1,0);
    c[x]=0;
    for (int k=1;k<=n;k++){
        int p,mi=inf;
        for (int i=1;i<=n;i++){
            if (!vis[i]&&c[i]<mi){
                mi=c[i];p=i;
            }
        }
        vis[p]=1;
        for (auto i:v[p]){
            int to=i.first,d=i.second;
            if (!vis[to]&&c[p]+d<c[to]){
                c[to]=c[p]+d;
            }
        }
    }
}
```

堆优化$O(mlogn+nlogn)$

```
const int inf=0x3f3f3f3f;
int n,m;
int c[N];
vector<pair<int,int>>v[N];
void dijkstra(int x){
    priority_queue< pair<int,int> , vector< pair<int,int>> , greater<
pair<int,int>> > q;
    fill(c,c+n+1,inf);
    c[x]=0;
    q.push({0,x});
    while(!q.empty()){
        pair<int,int> r=q.top();q.pop();
        int u=r.second;
        if (c[u]<r.first)continue;
        for (auto i:v[u]){
            int to=i.first,d=i.second;
            if(c[u]+d<c[to]){
                c[to]=c[u]+d;
                q.push({c[to],to});
            }
        }
    }
}
```

spfa

一般$O(m)$，最坏$O(nm)$，会被卡

```
const int inf=0x3f3f3f3f;
int c[N],vis[N];
vector<pair<int,int>>v[N];
void spfa(int x){
    fill(c,c+n+1,inf);
    fill(vis,vis+n+1,0);
    queue<int>q;
    q.push(x);
    c[x]=0;vis[x]=1;
    while(!q.empty()){
        int u=q.front();q.pop();
        for(auto i:v[u]){
            int v=i.first,d=i.second;
            if(c[v]>c[u]+d){
                c[v]=c[u]+d;
                if(!vis[v]){
                    vis[v]=1;q.push(v);
                    //in[v]=in[u]+1;if (in[v]>=n)return false;含有负环
                }
            }
        }
        vis[u]=0;
    }
}
```

```
const int inf=0x3f3f3f3f;
int c[N],vis[N];
vector<pair<int,int>>v[N];
void spfa(int x){
    fill(c,c+n+1,inf);
    fill(vis,vis+n+1,0);
    deque<int>q;
    q.push_back(x);
    c[x]=0;vis[x]=1;
    int sum=c[x],num=1;
    while(!q.empty()){
        int u=q.front();
        while(c[u]*num>sum){//LLL优化,Large Label Last
            q.pop_front();
            q.push_back(u);
            u=q.front();
        }
        q.pop_front();
        sum-=c[u];
        num--;
        for(auto i:v[u]){
            int v=i.first,d=i.second;
            if(c[v]>c[u]+d){
                c[v]=c[u]+d;
                if(!vis[v]){
                    vis[v]=1;
                    if(!q.empty()&&c[v]<c[q.front()])//SLF优化,Small Label First
                        q.push_front(v);
```

```
                    else q.push_back(v);
                    sum+=c[v];
                    num++;
                }
            }
        }
        vis[u]=0;
    }
}
```

floyed

$O(n^3)$

```
for(int k=1;k<=n;k++) //中转点
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            mp[i][j]=min(mp[i][j],mp[i][k]+mp[k][j]);
```

01传递闭包$O(n^3/32)$

```
bitset<N>a[N];
for (int i=0;i<n;i++){
    for (int j=0;j<n;j++){
        if(a[j][i])a[j]|=a[i];
    }
}
```

差分约束

n个未知量x_i，m个不等式$x_a - x_b <= c$，求其一组解

其他带等号不等式都可转换为这种形式

分析：

$x_a - x_b <= c$变形为$x_a <= x_b + c$同最短路的不等式，可建立b到a一条长为c的有向边。

若$x_1, \ldots, x_n$为一组解，则$x_1 + d, \ldots, x_n + d$也是其解

设$c[0] = 0$，并向每一个点连一条权重为0的边，跑单源最短路，若图中存在负环，则给定的差分约束系统无解；否则，$c[i]$为该差分约束系统的一组解。

因为存在负边权，采用spfa，和spfa复杂度一样，最坏$O(nm)$

```
#include<bits/stdc++.h>
using namespace std;
const int inf=0x3f3f3f3f,N=5e3+10;//设为数据范围上限
int n,m;
int c[N],vis[N],in[N];
vector<pair<int,int>>v[N];
bool spfa(int x){
    fill(c,c+n+1,inf);
    fill(vis,vis+n+1,0);
    fill(in,in+n+1,0);
    queue<int>q;
    q.push(x);
    c[x]=0;vis[x]=1;
```

```
    while(!q.empty()){
        int u=q.front();q.pop();
        for(auto i:v[u]){
            int v=i.first,d=i.second;
            if(c[v]>c[u]+d){
                c[v]=c[u]+d;
                if(!vis[v]){
                    vis[v]=1;q.push(v);
                    in[v]=in[u]+1;
                    if(in[v]>=n+1)return false;
                }
            }
        }
        vis[u]=0;
    }
    return true;
}
void add(int x,int y,int z){
    v[x].push_back({y,z});
}
signed main(){
    int x,y,z;
    cin>>n>>m;
    for (int i=1;i<=m;i++){
        scanf("%d%d%d",&x,&y,&z);
        add(y,x,z);
    }
    for (int i=1;i<=n;i++)add(0,i,0);
    if (!spfa(0))puts("NO");
    else for(int i=1;i<=n;i++)printf("%d ",c[i]);
    return 0;
}
```

k短路

最小生成树

定理：同一个图的每个最小生成树中，边权相等的边数量相等

prim

$O(n^2)$

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10,inf=0x3f3f3f3f;
int n,m,k;
vector<pair<int,int>>v[N];
int c[N],vis[N];
signed main(){
    cin>>n>>m;
    int x,y,z;
    for (int i=1;i<=m;i++){
        scanf("%d%d%d",&x,&y,&z);
        v[x].push_back({y,z});v[y].push_back({x,z});
    }
    int ans=0;
    fill(c,c+1+n,inf);
```

```
    c[1]=0;
    for (int k=1;k<=n;k++){
        int mi=inf,p=0;
        for (int i=1;i<=n;i++){
            if (!vis[i]&&mi>c[i]){
                mi=c[i];p=i;
            }
        }
        if (p==0)return puts("orz"),0;
        vis[p]=1;ans+=mi;
        for (auto i:v[p]){
            if(!vis[i.first])c[i.first]=min(c[i.first],i.second);
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

kruskal

$O(mlogm)$

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
int n,m,k;
struct node{
    int x,y,z;
}e[N];
bool cmp(node a,node b){
    return a.z<b.z;
}
int f[N];
int ff(int x){
    if (f[x]==x)return x;
    return f[x]=ff(f[x]);
}
bool uni(int x,int y){
    int xx=ff(x),yy=ff(y);
    if (xx!=yy){
        f[xx]=yy;return 1;
    }
    return 0;
}
signed main(){
    cin>>n>>m;
    for (int i=1;i<=n;i++)f[i]=i;
    int x,y,z;
    for (int i=1;i<=m;i++)scanf("%d%d%d",&e[i].x,&e[i].y,&e[i].z);
    sort(e+1,e+1+m,cmp);
    int ans=0,cnt=1;
    for (int i=1;i<=m;i++){
        if (uni(e[i].x,e[i].y)){
            ans+=e[i].z;cnt++;
        }
    }
    if (cnt!=n)puts("orz");
```

```
    else cout<<ans<<endl;
    return 0;
}
```

朱刘算法(有向图最小生成树)

$O(nm)$

```
#include<bits/stdc++.h>
using namespace std;
const int inf=2000000000;
const int N=1000+10;
typedef long long ll;
struct edge{
    int u,v;
    ll w;
}e[10005];
int m,n,pre[N],id[N],visit[N],xroot;
ll in[N],sum;
//eCnt为图中的边数
//n为图中的顶点数
//pre[i]为顶点i的前驱节点
//id[i]为缩环，形成新图的中间量
//in[i]为点i的最小入边
//visit[i]遍历图时记录顶点是否被访问过
ll directedMST(int root,int nv,int ne){
    ll ans=0;
    while(1){
        //1.找最小入边
        for(int i=0;i<nv;i++) in[i]=inf;
        for(int i=0;i<ne;i++){
            int u=e[i].u;
            int v=e[i].v;
            if(u!=v&&e[i].w<in[v]){
                if(u==root)//此处标记与源点相连的最小边
                    xroot=i;
                in[v]=e[i].w;
                pre[v]=u;
            }
        }
        for(int i=0;i<nv;i++)//判断图是否连通
            if(i!=root&&in[i]==inf)return -1;//除了跟以外有点没有入边,则根无法到达它
        //2.找环
        int nodeCnt=0;//图中环的数目
        memset(id,-1,sizeof(id));
        memset(visit,-1,sizeof(visit));
        in[root]=0;
        for(int i=0;i<nv;i++){
            ans+=in[i];
            int v=i;
            while(visit[v]!=i&&id[v]==-1&&v!=root){//每个点寻找其前序点，要么最终寻找至根部，要么找到一个环
                visit[v]=i;
                v=pre[v];
            }
            if(v!=root&&id[v]==-1){//缩点
                for(int u=pre[v];u!=v;u=pre[u])
```

```
                    id[u]=nodeCnt;
                id[v]=nodeCnt++;
            }
        }
        if(nodeCnt==0) break;//如果无环，跳出循环
        for(int i=0; i<nv; i++)
            if(id[i]==-1)
                id[i]=nodeCnt++;
        //3.缩点，重新标记
        for(int i=0;i<ne;i++){
            int v=e[i].v;
            e[i].u=id[e[i].u];
            e[i].v=id[e[i].v];
            if(e[i].u!=e[i].v)
                e[i].w-=in[v];
        }
        nv=nodeCnt;
        root=id[root];
    }
    return ans;
}
int main(){
    int m;
    while(scanf("%d%d",&n,&m)!=EOF){
        sum=0;
        for(int i=0;i<m;i++){
            scanf("%d%d%lld",&e[i].u,&e[i].v,&e[i].w);
            e[i].u++;e[i].v++;//都++之后，把0设为超级源点，联通各点
            sum+=e[i].w;
            if(e[i].u==e[i].v)e[i].w=inf;//消除自环
        }
        sum++;//此处必须++，因为需要权值比总权值大，因为这个w几次，，，
        for(int i=m;i<n+m;i++){
            e[i].u=0;
            e[i].v=i-m+1;
            e[i].w=sum;
        }
        ll ans=directedMST(0,n+1,m+n);//根 点数 边数
        if(ans==-1||ans-sum>=sum) printf("impossible\n");//ans-sum是除去虚根的最小树形图的最短路径，如果这个距离比所有的边权值和sum还大，说明还有另外的边由虚点发出，故说明此图不连通
        else printf("%lld %d\n",ans-sum,xroot-m);
        printf("\n");
    }
    return 0;
}
```

kruskal重构树

例如：求最小生成树x到y的最长边。可以在kruskal选取未连接的两个点的过程中，新增一个点作为两点的根节点，其点权为边权值，在查询x到y的最长边时，就可以直接t=lca(x,y),t的权值就是最长边。

```
void uni(int x,int y,int z){
    int xx=ff(x),yy=ff(y);
    if (xx!=yy){
        val[++cnt]=z;
        f[xx]=f[yy]=f[cnt]=cnt;
        v[xx].push_back(cnt);v[cnt].push_back(xx);
        v[yy].push_back(cnt);v[cnt].push_back(yy);
    }
}
```

匹配

最大匹配：二分图中边集的数目最大的那个匹配；

最小顶点覆盖：用最少的点，让每条边都至少和其中一个点关联；

最小边覆盖：用尽量少的不相交的边覆盖所有顶点；

最小链覆盖：用尽量少的不相交简单路径覆盖有向无环图(DAG)G的所有顶点；

最长反链：DAG的一个点集，任意两点都不能从一个走到另一个。

最大独立集：在N个点的图G中选出m个点，使这m个点两两之间没有边的点中，m的最大值。

二分图的最小点覆盖 = 二分图的最大匹配（证明见König定理）

二分图的最少边覆盖 = 点数 - 二分图的最大匹配

最小链覆盖 = 最长反链 = 点数 - 二分图的最大匹配（建图就是拆点i为i和i+n，按照二分图最大匹配的形式建图）

二分图的最大独立集 = 点数 - 二分图的最大匹配

无向图的最大团 = 无向图补图的最大独立集

霍尔定理

霍尔定理是判断二分图是否存在完美匹配的充要条件：

对于X,Y的完美匹配，要求$|X| \leq |Y|$；对于任意X的子集a，b是a能到达的并集，$|a| \leq |b|$

1.2021牛客多校10 C

$n=30000,k<=100$，每个女生都有k个不喜欢的男生，询问匹配方案，若没有，输出-1。

考虑霍尔定理，对于度大的男生，必然会产生完美匹配，也就是把男生点按度数从小到大排序，然后找到最小的t满足deg[t]>=k+(t-1)

可考虑，n-k个点暴力匹配(用set维护)，后面不超过k个点进行二分匹配，复杂度为$O(nklogn+nk^2)$

```
#include<bits/stdc++.h>
using namespace std;
const int N=3e4+5;
int n,m,k,p,t,x;
int used[N],match[N];
bitset<N>a[N];
int flag;
bool dfs(int x){
    for (int i=1;i<=n;i++){
        if (used[i]!=flag&&a[x][i]){
            used[i]=flag;
```

```
            if (!match[i]||dfs(match[i])){
                match[i]=x;
                return 1;
            }
        }
    }
    return 0;
}
set<int>s,e;
struct node{
    int id,d;
}d[N];
bool cmp(node a,node b){
    return a.d>b.d;
}
int main(){
    int T;
    cin>>n;
    int ans=0;
    for (int i=1;i<=n;i++)a[i].set();
    for (int i=1;i<=n;i++){
        d[i].id=i;
        scanf("%d",&k);
        for (int j=1;j<=k;j++){
            scanf("%d",&x);
            d[x].d++;
            a[x][i]=0;
        }
        s.insert(i);
    }
    sort(d+1,d+1+n,cmp);
    for (int r=1;r<=n;r++){
        int i=d[r].id,p=0;
        for (int j:s){
            if (a[i][j]){
                p=j;
                match[j]=i;ans++;
                break;
            }
        }
        if (p)s.erase(s.find(p));
        else e.insert(i);
    }
    for (int i:e){
        flag++;
        if (dfs(i))ans++;
    }
    if (ans==n){
        for (int i=1;i<=n;i++)printf("%d ",match[i]);puts("");
    }
    else puts("-1");
    return 0;
}
```

二分匹配

稳定婚姻算法，$O(nm)$

```
//p个和n个匹配，a为地图
#include<bits/stdc++.h>
using namespace std;
const int N=1e3+5;
int n,m,k,p,t;
int a[N][N],used[N],match[N];
bool dfs(int x){
    for (int i=1;i<=n;i++){
        if (!used[i]&&a[x][i]){
            used[i]=1;
            if (!match[i]||dfs(match[i])){
                match[i]=x;
                return 1;
            }
        }
    }
    return 0;
}
int main(){
    int T;
    cin>>T;
    while(T--){
        cin>>p>>n;
        memset(a,0,sizeof a);
        memset(match,0,sizeof match);
        for (int i=1;i<=p;i++){
            for (int j=1;j<=n;j++){
                scanf("%d",&a[i][j]);
            }
        }
        int ans=0;
        for (int i=1;i<=p;i++){
            memset(used,0,sizeof used);//可以换成时间戳优化
            if (dfs(i))ans++;
        }
        if (ans==p)puts("YES");
        else puts("NO");
    }
    return 0;
}
```

二分图最大权匹配

KM算法，$O(n^3)$

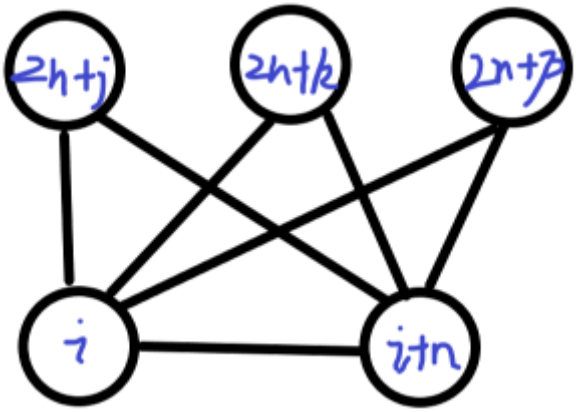
```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define int ll
constexpr int inf=0x3f3f3f3f3f3f3f3f;
constexpr int N = 500;
int n, m, e[N + 7][N + 7];
struct km{
```

```
    int mb[N + 7], vb[N + 7], ka[N + 7], kb[N + 7], p[N + 7], c[N + 7];
    void Bfs(int u) {
        int a, v = 0, vl = 0, d;
        for (int i = 1; i <= n; i++) p[i] = 0, c[i] = inf;
        mb[v] = u;
        do {
            a = mb[v], d = inf, vb[v] = 1;
            for (int b = 1; b <= n; b++)
                if (!vb[b]){
                    if (c[b] > ka[a] + kb[b] - e[a][b])
                        c[b] = ka[a] + kb[b] - e[a][b], p[b] = v;
                    if (c[b] < d) d = c[b], vl = b;
                }
            for (int b = 0; b <= n; b++)
                if (vb[b]) ka[mb[b]] -= d, kb[b] += d;
                else c[b] -= d;
            v = vl;
        } while (mb[v]);
        while (v) mb[v] = mb[p[v]], v = p[v];
    }
    ll solve() {
        for (int i = 1; i <= n; i++) mb[i] = ka[i] = kb[i] = 0;
        for (int a = 1; a <= n; a++) {
            for (int b = 1; b <= n; b++) vb[b] = 0;
            Bfs(a);
        }
        ll res = 0;
        for (int b = 1; b <= n; b++) res += e[mb[b]][b];
        return res;
    }
}KM;
signed main(){
    int m;
    scanf("%lld%lld", &n, &m);
    for (int a = 1; a <= n; a++)
        for (int b = 1; b <= n; b++)
            e[a][b] = -inf;
    while (m--){
        int u, v, w;
        scanf("%lld%lld%lld", &u, &v, &w);
        e[u][v] = w;
    }
    printf("%lld\n", KM.solve());
    for (int u = 1; u <= n; u++) printf("%d ", KM.mb[u]); puts("");
}
```

一般图最大匹配

带花树$O(n^3)$，但是能过1000的点

建图技巧：1.n个人匹配m个蝴蝶结，若一个人能匹配2个蝴蝶结，则答案+1



如图1—$2n$为人，$2n+1$—$2n+m$为蝴蝶结，每个人拆为i与i+n，i与i+n相连，人与其可匹配的蝴蝶结相连

设带花树匹配的边数为x，因为匹配一个蝴蝶结不如匹配其拆点，则
$2*ans+(n-ans)=x,\therefore ans=x-n$

```
#include<bits/stdc++.h>
const int N=1000+10;
using namespace std;
namespace FlowerTree{
    int FlowerTree_n;
    //每个节点只能有一个匹配对象
    struct Edge{
        int to,next;
    }e[1000010];//边数
    int head[N],tot=1;
    int fa[N];//fa[i] 记录i点属于哪一个点为根的花
    int pre[N];//pre[i] i的前驱节点
    queue<int> Q;//int Q[N * N * 2], h, t;//bfs队列 头尾
    int Match[N];//Match[i]标记匹配的对象
    int tim;//lca时间戳
    int Type[N];//Type[i]标记i点的类型 奇节点：1 偶节点：2 没有标记过时为0
    int dfn[N];//每个节点的lca时间戳
    void addEdge(int u,int v){//建双边
        e[++tot]={v,head[u]};
        head[u]=tot;
        e[++tot]={u,head[v]};
        head[v]=tot;
    }
    int find(int x) {
        return fa[x]==x?x:fa[x]=find(fa[x]);
    }
    int lca(int x, int y) {
        ++tim;
        x=find(x);y=find(y);
        while (dfn[x]!=tim) {
            dfn[x]=tim;
            x=find(pre[Match[x]]);
            if(y)swap(x,y);
        }return x;
    }
    //开花操作
    //将奇环缩成一个点并将原来奇点的点变为偶点并加入队列中
    void blossom(int x,int y,int p) {
        while (find(x)!=p){
```

```
            pre[x]=y;
            y=Match[x];
            if (Type[y]==2)Type[y]=1,Q.push(y);
            fa[x]=p;
            fa[y]=p;
            x=pre[y];
        }
    }
    bool Aug(int st){ //从没有匹配的u点开始bfs找增广路 将其标记为A类点
        memset(Type,0,sizeof Type);
        memset(pre,0,sizeof pre);
        for (int i=1;i<=FlowerTree_n;i++)fa[i]=i;
        while (!Q.empty())Q.pop();
        //------------------
        Q.push(st);
        Type[st]=1;//标记为奇点
        while (!Q.empty()) {
            int u=Q.front();
            Q.pop();
            for (int i=head[u];i;i=e[i].next){
                int v=e[i].to;
                //如果遇到了偶节点 就忽略不管
                if (find(u)==find(v)||Type[v]==2)continue;
                //如果这个点没有被匹配过 说明找到了一条增广路
                //因为队列里存的都是奇点 所以找到的那个没有匹配过的点肯定是偶点
                if (!Type[v]){
                    Type[v]=2;
                    pre[v]=u;
                    if (!Match[v]) {
                        //找到now之前的那个节点x 将原先匹配过的x匹配现在的节点
                        //让原来匹配的那个节点重新找别人匹配
                        for (int now=v,x;now;now=x) {
                            x=Match[pre[now]];
                            Match[now]=pre[now];
                            Match[pre[now]]=now;
                        }
                        return true;
                    }
                    //如果有匹配 则v是偶点 将v匹配的对象放入队列中
                    Type[Match[v]]=1;
                    //Q[t++]=Match[v];
                    Q.push(Match[v]);
                }
                else if (Type[v]==1){
                    // 如果找到的点是奇点 即 找到环了
                    // 就进行开花操作
                    int p=lca(u,v);
                    blossom(u,v,p);
                    blossom(v,u,p);
                }
            }
        }
        return false;
    }
    void init() {
        memset(Match,0,sizeof Match);
        memset(head,0,sizeof head);
        memset(dfn,0,sizeof dfn);
```

```
        tim=0;
        tot=1;
    }
    int solve(int _n){
        FlowerTree_n=_n;
        int res=0;
        for (int i=1;i<=FlowerTree_n; i++){
            if (!Match[i]&&Aug(i))res++;
        }return res;
    }
}
using namespace FlowerTree;
signed main(){
    int T,x,y,n,m;
    scanf("%d",&T);
    while (T--){
        init();
        scanf("%d%d",&n,&m);
        for (int i=1;i<=m;i++){
            scanf("%d%d",&x,&y);
            FlowerTree::addEdge(x,y);
        }
        int ans=solve(n);
        printf("%d\n",ans);
    }
    return 0;
}
```

俞天瑞模板

对某些情况相对较慢

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
#include<queue>
using namespace std;
const int N = 510;
int n, m, x, y; vector<int>g[N];
namespace Blossom {
    int mate[N], n, ret, nxt[N], f[N], mark[N], vis[N], t; queue<int>Q;
    int F(int x) { return x == f[x] ? x : f[x] = F(f[x]); }
    void merge(int a, int b) { f[F(a)] = F(b); }
    int lca(int x, int y) {
        for (t++;; swap(x, y))if (~x) {
            if (vis[x = F(x)] == t)return x; vis[x] = t;
            x = mate[x] != -1 ? nxt[mate[x]] : -1;
        }
    }
    void group(int a, int p) {
        for (int b, c; a != p; merge(a, b), merge(b, c), a = c) {
            b = mate[a], c = nxt[b];
            if (F(c) != p)nxt[c] = b;
            if (mark[b] == 2)mark[b] = 1, Q.push(b);
            if (mark[c] == 2)mark[c] = 1, Q.push(c);
        }
```

```
    }
    void aug(int s, const vector<int>G[]) {
        for (int i = 0; i < n; ++i)nxt[i] = vis[i] = -1, f[i] = i, mark[i] = 0;
        while (!Q.empty())Q.pop(); Q.push(s); mark[s] = 1;
        while (mate[s] == -1 && !Q.empty()) {
            int x = Q.front(); Q.pop();
            for (int i = 0, y; i < (int)G[x].size(); ++i) {
                if ((y = G[x][i]) != mate[x] && F(x) != F(y) && mark[y] != 2) {
                    if (mark[y] == 1) {
                        int p = lca(x, y);
                        if (F(x) != p)nxt[x] = y;
                        if (F(y) != p)nxt[y] = x;
                        group(x, p), group(y, p);
                    }
                    else if (mate[y] == -1) {
                        nxt[y] = x;
                        for (int j = y, k, l; ~j; j = l)k = nxt[j], l = mate[k],
mate[j] = k, mate[k] = j;
                        break;
                    }
                    else nxt[y] = x, Q.push(mate[y]), mark[mate[y]] = 1, mark[y]
= 2;

                }
            }
        }
    }
    int solve(int _n, const vector<int>G[]) {
        n = _n; memset(mate, -1, sizeof mate);
        for (int i = t = 0; i < n; ++i)if (mate[i] == -1)aug(i, G);
        for (int i = ret = 0; i < n; ++i)ret += mate[i] > i;
        printf("%d\n", ret);
        for (int i = 0; i < n; i++)printf("%d", mate[i] + 1);
        return ret;
    }
}
signed main() {
    scanf("%d%d", &n, &m);
    while (m--){
        scanf("%d%d",&x,&y),x--,y--,g[x].push_back(y), g[y].push_back(x);
    }
    Blossom::solve(n, g);
}
```

一般图最大权匹配

带权带花树$O(n^3)$

```
#include <bits/stdc++.h>
#define N 810
using namespace std;
typedef long long ll;
inline int read(){
    int x=0,f=1; char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1; ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0'; ch=getchar();}
    return x*f;
```

```
}
struct edge{int u,v,w;}mps[N][N];
int n,m,mat[N],pre[N],bl[N],fa[N],tim;ll totw=0;
int sign[N],lab[N],slacku[N],blofm[N][N],tot,mx;
vector <int> leaves[N];
int q[N],hd;
inline int calc_e(const edge &e){
    return lab[e.u]+lab[e.v]-mps[e.u][e.v].w*2;
}
inline void updata(int u,int x){
    if(!slacku[x]||calc_e(mps[u][x])<calc_e(mps[slacku[x]][x]))
        slacku[x]=u;
}
inline void calc_slack(int x){
    slacku[x]=0;
    for(int i=1;i<=n;i++)
        if(mps[i][x].w>0&&fa[i]!=x&&sign[fa[i]]==0)
            updata(i,x);
}
inline void q_push(int x){
    if(x<=n) q[++hd]=x;
    else for(int i=0;i<(int)leaves[x].size();i++)
        q_push(leaves[x][i]);
}
inline int get_lca(int x, int y){
    if(tim==100000000)
        memset(bl,0,sizeof bl),tim=0;
    for(++tim;x||y;swap(x,y)) if(x){
        if(bl[x]==tim) return x;
        bl[x]=tim; x=fa[mat[x]];
        if(x) x=fa[pre[x]];
    }
    return 0;
}
inline void set_fa(int x,int y){
    fa[x]=y; if(x>n)
    for(int i=0;i<(int)leaves[x].size();i++)
        set_fa(leaves[x][i],y);
}
inline void set_mat(int x,int y){
    mat[x]=mps[x][y].v;
    if(x<=n) return ;
    int xr=blofm[x][mps[x][y].u];
    int pr=find(leaves[x].begin(),leaves[x].end(),xr)-leaves[x].begin();
    if(pr%2==1)
        reverse(leaves[x].begin()+1, leaves[x].end()),
        pr=(int)leaves[x].size()-pr;
    for(int i=0;i<pr;i++)
        set_mat(leaves[x][i],leaves[x][i^1]);
    set_mat(xr,y);
    rotate(leaves[x].begin(),leaves[x].begin()+pr,leaves[x].end());
}
inline void blossom_blooms(int x){
    for(int i=0;i<(int)leaves[x].size();i++){
        if(leaves[x][i]>n&&!lab[leaves[x][i]])
            blossom_blooms(leaves[x][i]);
        else set_fa(leaves[x][i],leaves[x][i]);
    }
```

```
    fa[x]=0;
}
inline void blossom_make(int u,int lca,int v){
    int x=n+1; while(x<=tot&&fa[x]) x++;
    if(x>tot) tot++;
    lab[x]=sign[x]=0;
    mat[x]=mat[lca]; leaves[x].clear();
    leaves[x].push_back(lca);
    for(int i=u;i!=lca;i=fa[pre[fa[mat[i]]]])

leaves[x].push_back(i),leaves[x].push_back(fa[mat[i]]),q_push(fa[mat[i]]);
    reverse(leaves[x].begin()+1, leaves[x].end());
    for(int i=v;i!=lca;i=fa[pre[fa[mat[i]]]])

leaves[x].push_back(i),leaves[x].push_back(fa[mat[i]]),q_push(fa[mat[i]]);
    set_fa(x,x);
    for(int i=1;i<=tot;i++)
        mps[x][i].w=mps[i][x].w=0, blofm[x][i]=0;
    for(int i=0;i<(int)leaves[x].size();i++){
        int xs=leaves[x][i];
        for(int j=1;j<=tot;j++)
            if(!mps[x][j].w||calc_e(mps[xs][j])<calc_e(mps[x][j]))
                mps[x][j]=mps[xs][j],mps[j][x]=mps[j][xs];
        for(int j=1;j<=tot;j++)
            if(blofm[xs][j]) blofm[x][j]=xs;
    }
    calc_slack(x);
}
inline void link(int x,int y){
    while(1){
        int xx=fa[mat[x]];
        set_mat(x,y);
        if(!xx) return ;
        set_mat(xx,fa[pre[xx]]);
        x=fa[pre[xx]]; y=xx;
    }
}
inline int deal_edge(const edge &e){
    int u=fa[e.u],v=fa[e.v];
    if(sign[v]==-1){//unsigned
        pre[v]=e.u; // cause we bfs all vertices tegother,we dont' need to
discuss two situation
        sign[v]=1; sign[fa[mat[v]]]=0;
        slacku[v]=slacku[fa[mat[v]]]=0;
        q_push(fa[mat[v]]);
    }
    else if(!sign[v]){//S signed vertex
        int lca=get_lca(u,v);
        if(!lca){
            link(u,v); link(v,u); //connected! new argument.
            for(int i=n+1;i<=tot;i++)
                if(fa[i]==i&&lab[i]==0)
                    blossom_blooms(i); // flower may not be a flower any more so
we blossom blooms!
            return 1;
        }
        else blossom_make(u,lca,v); // form a new flower!
    }
```

```
    return 0;
}
inline void blossom_bloom_1(int x){
    for(int i=0;i<(int)leaves[x].size();i++)
        set_fa(leaves[x][i],leaves[x][i]);
    int xr=blofm[x][mps[x][pre[x]].u];
    int pr=find(leaves[x].begin(), leaves[x].end(),xr)-leaves[x].begin();
    if(pr%2==1)
        reverse(leaves[x].begin()+1, leaves[x].end()),
        pr=(int)leaves[x].size()-pr;
    for(int i=0;i<pr;i+=2){
        int u=leaves[x][i],v=leaves[x][i+1];
        pre[u]=mps[v][u].u;
        sign[u]=1; sign[v]=0;
        slacku[u]=0; calc_slack(v); q_push(v);
    }
    sign[xr]=1; pre[xr]=pre[x];
    for(int i=pr+1;i<(int)leaves[x].size();i++){
        int u=leaves[x][i];
        sign[u]=-1; calc_slack(u);
    }
    fa[x]=0;
}
inline int match(){
    for(int i=1;i<=tot;i++) slacku[i]=0,sign[i]=-1;
    hd=0; for(int i=1;i<=tot;i++)
        if(fa[i]==i&&!mat[i])
            slacku[i]=pre[i]=sign[i]=0,q_push(i);
    if(!hd) return 0;
    while(1){
        for(int i=1;i<=hd;i++){
            int lx=q[i]; for(int j=1;j<=n;j++)
                if(mps[lx][j].w>0&&fa[lx]!=fa[j]){
                    if(!calc_e(mps[lx][j])){
                        if(deal_edge(mps[lx][j]))
                            return 1;
                    }
                    else if(sign[fa[j]]!=1) updata(lx,fa[j]);
                }
        }
        int d=0x3fffffff;
        for(int i=1;i<=n;i++) if(!sign[fa[i]])
            d=min(d,lab[i]);
        for(int i=n+1;i<=tot;i++)
            if(fa[i]==i&&sign[i]==1)
                d=min(lab[i]/2,d);
        for(int i=1;i<=tot;i++) if(fa[i]==i&&slacku[i]){
            if(sign[i]==-1) d=min(calc_e(mps[slacku[i]][i]),d);
            else if(sign[i]==0) d=min(calc_e(mps[slacku[i]][i])/2,d);
        }
        for(int i=1;i<=n;i++)
            if(sign[fa[i]]==0) lab[i]-=d;
            else if (sign[fa[i]]==1) lab[i]+=d;
        for(int i=n+1;i<=tot;i++)
            if(fa[i]==i){
                if(sign[i]==0) lab[i]+=d*2;
                else if(sign[i]==1) lab[i]-=d*2;
            }
```

```
        hd=0;
        for(int i=1;i<=n;i++) if(!lab[i]) return 0; //all vetices matched,single
vetices's label = 0
        for(int i=1;i<=tot;i++)
            if(fa[i]==i&&slacku[i]&&fa[slacku[i]]!=i&&calc_e(mps[slacku[i]]
[i])==0)
                /*new edge*/ if(deal_edge(mps[slacku[i]][i])) return 1;
        for(int i=n+1;i<=tot;i++)
            if(fa[i]==i&&sign[i]==1&&!lab[i])
                blossom_bloom_1(i);
    }
    return 0;
}
inline void solve(){
    for(int i=1;i<=n;i++) mat[i]=0;
    tot=n; hd=totw=0;
    for(int i=0;i<=n;i++) fa[i]=i,leaves[i].clear();
    for(int i=1;i<=n;i++) for(int j=1;j<=n;j++)
        blofm[i][j]=(i==j? i:0);
    for(int i =1;i<=n;i++) lab[i]=mx; //init label
    while(match());
    for(int i=1;i<=n;i++) if(mat[i]&&mat[i]<i)
        totw+=mps[i][mat[i]].w;
}
signed main(){
    n=read(); m=read();
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            mps[i][j]=(edge){i,j,0};
    for(int i=1;i<=m;i++){
        int u=read(),v=read(),w=read();
        mps[u][v].w=mps[v][u].w=w;mx=max(mx,w);
    }
    solve();printf("%lld\n",totw);
    for(int i=1;i<=n;i++)
        printf("%d ",mat[i]);
    puts("");
    return 0;
}
```

最大团

n<=50

```
int g[N][N],cnum[N],ans;//g为图
bool dfs(int step,int *R,int Rlen){
    int R2[N],R2len;
    if(step > ans){ ans = step; return true;} //剪枝3
    for(int i = 1; i <= Rlen; ++i){
        if(step + Rlen - i + 1 <= ans){ break;} //剪枝1
        if(step + cnum[R[i]] <= ans){ break;} //剪枝2
        R2len = 0;
        for(int j = i + 1; j <= Rlen; ++j){
            if(g[R[i]][R[j]]){
                R2[++R2len] = R[j];
            }
        }
```

```
        if(dfs(step + 1,R2,R2len))return true;
    }
    return false;
}
int maxclique(){
    int R[N],Rlen = 0;
    ans = 1; cnum[n] = 1;
    for(int i = n - 1; i >= 1; --i){
        Rlen = 0;
        for(int j = i + 1; j <= n; ++j){
            if(g[i][j])R[++Rlen] = j;
        }
        dfs(1,R,Rlen);
        cnum[i] = ans;
    }
    return ans;
}
```

网络流

dinic算法

$O(n^2m)$，一般1000可过

最大流=最小割

1.一张图，每个点有正负数，选择了一个点，必须选择它能连到的所有点，求选择的点的权值和最大值。

方法：原图间连inf边，正值连超级源点，负值连超级汇点。答案为max(0,正数和-最大流)

```
struct Dinic{
    int dep[N],head[N],cur[N];
    int step,n;
    struct node{
        int to,n;
        ll cap;
    }e[N<<1];
    void init(int x){
        n=x;step=0;
        for (int i=0;i<=n;i++)head[i]=-1;
    }
    void add(int x,int y,int v){
        e[step].to=y;
        e[step].cap=v;
        e[step].n=head[x];
        head[x]=step++;
        e[step].to=x;
        e[step].cap=0;
        e[step].n=head[y];
        head[y]=step++;
    }
    bool bfs(int s,int t){
        for (int i=0;i<=n;i++)dep[i]=0;
        queue<int>q;
        q.push(s);
        dep[s]=1;
        cur[s]=head[s];
```

```
        while(!q.empty()){
            s=q.front();
            q.pop();
            for(int i=head[s];i!=-1;i=e[i].n){
                int y=e[i].to;
                if(e[i].cap>0&&dep[y]==0){
                    dep[y]=dep[s]+1;
                    cur[y]=head[y];
                    if(y==t) return true;
                    q.push(y);
                }
            }
        }
        return false;
    }
    ll dfs(int s,ll flow,int t){
        if(s==t||flow<=0) return flow;
        ll rest=flow;
        for(int i=cur[s];i!=-1;i=e[i].n){
            cur[s]=i;
            int y=e[i].to;
            if(e[i].cap>0&&dep[y]==dep[s]+1){
                ll tmp=dfs(y,min(rest,e[i].cap),t);
                if(tmp<=0) dep[y]=0;
                rest-=tmp;
                e[i].cap-=tmp;
                e[i^1].cap+=tmp;
                if(rest<=0) break;
            }
        }
        return flow-rest;
    }
    ll solve(int s,int t){
        ll ans=0;
        while(bfs(s,t))
            ans+=dfs(s,inf,t);
        return ans;
    }
}dinic;
```

预流推进hlpp

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 100005 + 5;
const int MAXM = 1.2e6 + 5;

struct edge { int u, v, w; };
template <class T = int>
struct HLPP {
    const T INF = numeric_limits<T>::max();
    struct edge {
        int to, rev;
        T f;
    };
```

```
    int maxn, s, t;
    edge edges[2 * MAXM];
    int first_edge[MAXN + 1];
    int _cur_edge[MAXN];
    int nxt[MAXN];
    int lst[MAXN];
    T excess[MAXN];
    int arc[MAXN];

    int gapNxt[2 * MAXN], gapPrv[2 * MAXN];

    int height[MAXN];
    int highest, highestGap, work;
    int q[2 * MAXM];

    vector<int> degs;
    void clear()
    {
        highest = highestGap = work = 0;
        memset(arc, 0, sizeof(arc));
        memset(nxt, 0, sizeof(nxt));
        memset(lst, 0, sizeof(lst));
        memset(height, 0, sizeof(height));
        memset(q, 0, sizeof(q));
        memset(gapNxt, 0, sizeof(gapNxt));
        memset(gapPrv, 0, sizeof(gapPrv));
    }
    void init(vector<int> degrees, int s, int t)
    {
        clear();
        this->s = s;
        this->t = t;
        maxn = degrees.size();
        assert(maxn <= MAXN);
        int cnt = 0;
        for (int i = 0; i < maxn; ++i) {
            first_edge[i] = cnt;
            cnt += degrees[i];
        }
        first_edge[maxn] = cnt;
        copy(first_edge, first_edge + maxn + 1, _cur_edge);
        degs.swap(degrees);
    }

    void addEdge(int from, int to, int f, bool isDirected = true)
    {
        edges[_cur_edge[from]++] = { to, _cur_edge[to], f };
        edges[_cur_edge[to]++] = { from, _cur_edge[from] - 1, isDirected ? 0 : f
};
    }

    void pushLst(int h, int v)
    {
        nxt[v] = lst[h];
        lst[h] = v;
    }

    void updHeight(int v, int nh)
```

```
    {
        if (height[v] != maxn) {
            gapNxt[gapPrv[v]] = gapNxt[v];
            gapPrv[gapNxt[v]] = gapPrv[v];
        }

        height[v] = nh;
        if (nh == maxn)
            return;

        highestGap = max(highestGap, nh);
        if (excess[v] > 0) {
            highest = max(highest, nh);
            pushLst(nh, v);
        }

        nh += maxn;
        gapNxt[v] = gapNxt[nh];
        gapPrv[v] = nh;
        gapNxt[nh] = v;
        gapPrv[gapNxt[v]] = v;
    }

    void globalRelabel()
    {
        work = 0;
        fill(height, height + maxn, maxn);
        fill(lst, lst + maxn, -1);
        iota(gapNxt, gapNxt + maxn, 0);
        iota(gapPrv, gapPrv + maxn, 0);
        height[t] = 0;
        q[0] = t;
        int sz = 1;
        for (size_t i = 0; i < sz; ++i) {
            int v = q[i];
            for (int ie = first_edge[v]; ie < first_edge[v + 1]; ++ie) {
                auto& e = edges[ie];
                if (height[e.to] == maxn && edges[e.rev].f > 0)
                    q[sz++] = e.to, updHeight(e.to, height[v] + 1);
            }
            highest = highestGap = height[v];
        }
    }

    void push(int v, edge& e)
    {
        T df = min(excess[v], e.f);
        if (df > 0) {
            if (excess[e.to] == 0)
                pushLst(height[e.to], e.to);
            e.f -= df, edges[e.rev].f += df;
            excess[v] -= df, excess[e.to] += df;
        }
    }

    void discharge(int v)
    {
        int nh = maxn;
```

```
        for (int i = arc[v]; i < first_edge[v + 1]; i++) {
            auto& e = edges[i];
            if (e.f > 0) {
                if (height[v] == height[e.to] + 1) {
                    push(v, e);
                    if (excess[v] <= 0) {
                        arc[v] = i;
                        return;
                    }
                }
                else
                    nh = min(nh, height[e.to] + 1);
            }
        }

        for (int i = first_edge[v]; i < arc[v]; i++) {
            auto& e = edges[i];
            if (e.f > 0) {
                if (height[v] == height[e.to] + 1) {
                    push(v, e);
                    if (excess[v] <= 0) {
                        arc[v] = i;
                        return;
                    }
                }
                else
                    nh = min(nh, height[e.to] + 1);
            }
        }

        work++;

        if (gapNxt[gapNxt[height[v] + maxn]] != height[v] + maxn) {
            updHeight(v, nh);
        }
        else {
            int oldH = height[v];
            for (int h = oldH; h < highestGap + 1; h++) {
                for (int i = gapNxt[h + maxn]; i < maxn; i = gapNxt[i]) {
                    height[i] = maxn;
                }
                gapNxt[h + maxn] = gapPrv[h + maxn] = h + maxn;
            }
            highestGap = oldH - 1;
        }
    }

    T calc()
    {
        for (int v = 0; v < maxn; ++v) {
            sort(edges + first_edge[v], edges + first_edge[v + 1],
                [](edge& l, edge& r) { return l.to < r.to; });
            for (int i = first_edge[v]; i < first_edge[v + 1]; i++) {
                auto& e = edges[i];
                edges[e.rev].rev = i;
            }
        }
```

```
        copy(first_edge, first_edge + maxn, arc);
        fill(excess, excess + maxn, 0);
        excess[s] = INF, excess[t] = -INF;
        globalRelabel();

        for (int ie = first_edge[s]; ie < first_edge[s + 1]; ie++)
            push(s, edges[ie]);

        for (; highest >= 0; highest--) {
            while (lst[highest] != -1) {
                int v = lst[highest];
                lst[highest] = nxt[v];
                if (height[v] == highest) {
                    discharge(v);
                    if (work > 4 * maxn)
                        globalRelabel();
                }
            }
        }

        return excess[t] + INF;
    }
};

HLPP<> hlpp;
vector<edge> v;
vector<int> degs;
int solve(int n,vector<edge>& v,int s,int t)
{
    degs.clear(); degs.resize(n + 1);
    for (auto& x : v)
    {
        ++degs[x.u], ++degs[x.v];
    }
    hlpp.init(degs, s, t);
    for (auto& x : v)
        hlpp.addEdge(x.u, x.v, x.w);

    return hlpp.calc();
}

signed main()
{
    //int T; cin >> T;
    //while (T--)
    {
        int n, m, s, t;
        scanf("%d%d%d%d", &n, &m, &s, &t);
        v.clear(); v.resize(m);
        for (auto& x : v)
            scanf("%d%d%d", &x.u, &x.v, &x.w);

        cout<<solve(n, v, s, t);
    }
}
```

费用流

```
struct zkw{
    int Cost=0,Flow=0;
    int vis[N],dis[N],head[N];
    int step,n;
    struct node{
        int to,cap,cost,n;
    }e[N<<1];
    void add(int x,int y,int v,int c){
        e[step].to=y;
        e[step].cap=v;
        e[step].cost=c;
        e[step].n=head[x];
        head[x]=step++;
        e[step].to=x;
        e[step].cap=0;
        e[step].cost=-c;
        e[step].n=head[y];
        head[y]=step++;
    }
    bool spfa(int s,int t){
        for(int i=0;i<=n;i++) vis[i]=0,dis[i]=inf;//要改
        deque<int>q;
        q.push_back(t);
        dis[t]=0;
        vis[t]=1;
        while(!q.empty()){
            int u=q.front();
            q.pop_front();
            for(int i=head[u];~i;i=e[i].n){
                int v=e[i].to;
                if(e[i^1].cap&&dis[v]>dis[u]+e[i^1].cost){
                    dis[v]=dis[u]+e[i^1].cost;
                    if(!vis[v]){
                        vis[v]=1;
                        if(!q.empty()&&dis[v]<dis[q.front()])//SLF优化
                            q.push_front(v);
                        else q.push_back(v);
                    }
                }
            }
            vis[u]=0;
        }
        return dis[s]<inf;
    }
    int dfs(int s,int t,int flow){
        vis[s]=1;
        if(s==t||flow<=0) return flow;
        int res,used=0;
        for(int i=head[s];~i;i=e[i].n){
            int v=e[i].to;
            if(!vis[v]&&e[i].cap&&dis[s]-dis[v]==e[i].cost){
                res=dfs(v,t,min(e[i].cap,flow-used));
                if(res) {
                    e[i].cap-=res;
                    e[i^1].cap+=res;
```

```
                    Cost+=res*e[i].cost;
                    used+=res;
                }
                if(used==flow)break;
            }
        }
        return used;
    }
    void solve(int s,int t){
        while(spfa(s,t)){
            vis[t]=1;
            while(vis[t]){
                for(int i=0;i<=n;i++) vis[i]=0;//要改
                Flow+=dfs(s,t,inf);
            }
        }
    }
    void init(int n){
        step=0;
        Flow=Cost=0;
        memset(head,-1,sizeof head);
        this->n=n;
    }
}zkw;
```

树链

LCA

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int inf=0x3f3f3f3f;
const int N=5e5+10;
const int mod=998244353;
int n,m,k;
int step,head[N];
int f[N][21],dep[N];
struct node{
    int to,n;
}e[N<<1];
void add(int x,int y){
    e[step].to=y;
    e[step].n=head[x];
    head[x]=step++;
}
inline void dfs(int u,int fa){
    dep[u]=dep[fa]+1;
    for(int i=0;i<=19;i++)//2^0 ~ 2^19
        f[u][i+1]=f[f[u][i]][i];//递推公式，上面讲过了。
    for(int i=head[u];i!=-1;i=e[i].n){
        int v=e[i].to;
        if(v==fa) continue;
        f[v][0]=u;
        dfs(v,u);
    }
}
```

```
inline int LCA(int x,int y){
    if(dep[x]<dep[y]) swap(x,y);//让x深度较大
    //用“暴力”的思想：先让x,y跳到同一深度，然后一起往上跳
    for(int i=20;i>=0;i--){//倒着for，x能多跳尽量多跳 ，才能优化时间
        if(dep[f[x][i]]>=dep[y]) x=f[x][i];//先跳到同一层
        if(x==y) return y;
    }
    for(int i=20;i>=0;i--){//此时x,y已跳到同一层
        if(f[x][i]!=f[y][i]){//如果 f[x][i]和f[y][i]不同才跳
            x=f[x][i];
            y=f[y][i];
        }
    }
    return f[x][0];//跳完上述步骤后，两点离LCA仅一步之遥，让x(或y)再向上跳一步就是LCA。
}
int main(){
    int s,t;
    step=0;memset(head,-1,sizeof head);
    cin>>n>>m>>s;
    int x,y;
    for (int i=1;i<=n-1;i++){
        scanf("%d%d",&x,&y);
        add(x,y);add(y,x);
    }
    dfs(s,-1);
    for (int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        printf("%d\n",LCA(x,y));
    }
    return 0;
}
```

欧拉序st表，建立$O(nlogn)$，查询$O(1)$

```
#include <bits/stdc++.h>
using namespace std;
const int N=5e5+10;
int n,m,k;
vector<int>v[N];
int st[21][N*2];
int dfn[N],tms=0,o[N*2],dep[N];
void dfs(int x,int fa) {
    dep[x]=dep[fa]+1;st[0][++tms]=x;dfn[x]=tms;
    for (int i:v[x]){
        if(i==fa)continue;
        dfs(i,x);st[0][++tms]=x;
    }
}
void build(){
    for (int i=1;i<=tms;i++)o[i]=log(i)/log(2)+1e-7;
    for (int i=1;i<=o[tms];i++){
        for (int j=1,u,v;j+(1<<i)-1<=tms;j++) {
            u=st[i-1][j];v=st[i-1][j+(1<<i-1)];
            st[i][j]=dep[u]<dep[v]?u:v;
        }
    }
}
```

```
int LCA(int u,int v) {
    if (!u||!v)return 0;
    u=dfn[u];v=dfn[v];
    if (u>v)std::swap(u,v);
    int d=o[v-u+1];u=st[d][u];v=st[d][v-(1<<d)+1];
    return dep[u]<dep[v]?u:v;
}
signed main(){
    int x,y;
    scanf("%d%d%d",&n,&m,&k);
    for (int i=1;i<n;i++){
        scanf("%d%d",&x,&y);v[x].push_back(y);v[y].push_back(x);
    }
    dfs(k,0);build();
    while (m--){
        scanf("%d%d",&x,&y);
        printf("%d\n",LCA(x,y));
    }
    return 0;
}
```

树链剖分

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int N=2e5+10;
const int inf=0x3f3f3f3f;
const int mod=1e9+7;
int n,m,k,p;
int a[N];
int si[N],dep[N],fa[N],son[N];//si:子树大小,dep:深度,fa:父亲,son:重儿子
int id[N],top[N],anew[N],cnt=0;
//id:带重链的dfs序,top:重链的父亲,anew:id[x]的权值(build时用)
int head[N],step=0;
int res=0;//临时变量
struct edge{
    int to,n;
}e[N<<2];
void add(int x,int y){
    e[step].to=y;
    e[step].n=head[x];
    head[x]=step++;
}
void dfs1(int x,int ff,int depth){//处理dep,si,fa,son
    dep[x]=depth;
    si[x]=1;
    fa[x]=ff;
    int mx=-1;
    for (int i=head[x];~i;i=e[i].n){
        int v=e[i].to;
        if (v==ff)continue;
        dfs1(v,x,depth+1);
        si[x]+=si[v];
        if (si[v]>mx)mx=si[v],son[x]=v;
    }
}
```

```
void dfs2(int x,int topf){//x当前节点，topf当前链的最顶端的节点
    id[x]=++cnt;//标记每个点的新编号
    anew[cnt]=a[x];//把每个点的初始值赋到新编号上来
    top[x]=topf;
    if(!son[x])return;//如果没有儿子则返回
    dfs2(son[x],topf);//按先处理重儿子，再处理轻儿子的顺序递归处理
    for(int i=head[x];~i;i=e[i].n){
        int v=e[i].to;
        if(v==fa[x]||v==son[x])continue;
        dfs2(v,v);
    }
}
struct node{
    int val,lazy;
    int l,r;
}tr[N<<2];
void pushup(int rt){
    tr[rt].val=(tr[rt<<1].val+tr[rt<<1|1].val)%p;//
}
void putdown(int rt){
    if(tr[rt].lazy){
        tr[rt<<1].lazy+=tr[rt].lazy;
        tr[rt<<1].val+=(tr[rt<<1].r-tr[rt<<1].l+1)*tr[rt].lazy;
        tr[rt<<1|1].lazy+=tr[rt].lazy;
        tr[rt<<1|1].val+=(tr[rt<<1|1].r-tr[rt<<1|1].l+1)*tr[rt].lazy;
        tr[rt<<1].val%=p;tr[rt<<1|1].val%=p;//
        tr[rt].lazy=0;
    }
}
void build(int l,int r,int rt){
    tr[rt].l=l;
    tr[rt].r=r;
    tr[rt].lazy=0;
    if (l==r){
        tr[rt].val=anew[l]%p;//
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,rt<<1);
    build(mid+1,r,rt<<1|1);
    pushup(rt);
}
void update(int l,int r,int add,int rt){
    if (l<=tr[rt].l&&r>=tr[rt].r){
        tr[rt].lazy+=add;
        tr[rt].val+=(tr[rt].r-tr[rt].l+1)*add;
        return;
    }
    putdown(rt);
    if(l<=tr[rt<<1].r)
        update(l,r,add,rt<<1);
    if(r>=tr[rt<<1|1].l)
        update(l,r,add,rt<<1|1);
    pushup(rt);
}
int query(int l,int r,int rt){
    if(l==tr[rt].l&&r==tr[rt].r)
        return tr[rt].val%p;//
```

```
    putdown(rt);
    if(l>=tr[rt<<1|1].l)
        return query(l,r,rt<<1|1);
    else if(r<=tr[rt<<1].r)
        return query(l,r,rt<<1);
    else
        return (query(l,tr[rt<<1].r,rt<<1)+query(tr[rt<<1|1].l,r,rt<<1|1))%p;//
}
void updateRange(int x,int y,int k){
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        update(id[top[x]],id[x],k,1);
        x=fa[top[x]];
    }
    if(dep[x]>dep[y])swap(x,y);
    update(id[x],id[y],k,1);
}
void updateSon(int x,int k){
    update(id[x],id[x]+si[x]-1,k,1);
}
int qRange(int x,int y){
    int ans=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        res=query(id[top[x]],id[x],1);
        ans=(ans+res)%p;//
        x=fa[top[x]];
    }
    //在同一链上时
    if(dep[x]>dep[y])swap(x,y);
    res=query(id[x],id[y],1);
    ans=(ans+res)%p;//
    return ans;
}
int qSon(int x){
    res=query(id[x],id[x]+si[x]-1,1);
    return res;
}
int main(){
    int x,y;
    step=cnt=0;
    memset(head,-1,sizeof head);
    memset(son,0,sizeof son);
    cin>>n>>m>>k>>p;//点数、询问数、根节点、取模数
    for (int i=1;i<=n;i++)scanf("%d",&a[i]);
    for (int i=1;i<n;i++){
        scanf("%d%d",&x,&y);
        add(x,y);add(y,x);
    }
    dfs1(k,0,1);
    dfs2(k,k);
    build(1,n,1);
    while(m--){
        int c,x,y,z;
        scanf("%d",&c);
        if(c==1){//x到y的最短路径+z
            scanf("%d%d%d",&x,&y,&z);
            updateRange(x,y,z%p);//
```

```
        }
        else if(c==2){//询问x到y路径的权值和
            scanf("%d%d",&x,&y);
            printf("%d\n",qRange(x,y));
        }
        else if(c==3){//x的子树全+y
            scanf("%d%d",&x,&y);
            updateSon(x,y);
        }
        else{//询问x的子树权值和
            scanf("%d",&x);
            printf("%d\n",qSon(x));
        }
    }
    return 0;
}
```

Tarjan

强连通分量

每个分量内，两点互相可以到达

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=5e4+10;
int n,m,k;
int head[N];
int step,tot,gg;
int dfn[N],low[N];//dfn[x]标记x是在dfs中第几个遍历到的点
int color[N],cnt[N];
bool ff[N];
stack<int>s;
struct node{
    int to,n;
}e[N<<1];
void add(int x,int y){
    e[step].to=y;
    e[step].n=head[x];
    head[x]=step++;
}
void tarjan(int x){
    dfn[x]=low[x]=++tot;//表示点i是第几个被遍历到的，把low[i]的值先赋值为dfn[i]的值
    s.push(x);
    ff[x]=1;
    for(int i=head[x];i!=-1;i=e[i].n){
        int v=e[i].to;
        if(!dfn[v]){
            tarjan(v);
            low[x]=min(low[x],low[v]);
        }
        else if(ff[v])low[x]=min(low[x],dfn[v]);
    }
    int k;
    if(low[x]==dfn[x]){
        ++gg;
```

```
        do{
            k=s.top();s.pop();
            ff[k]=0;
            color[k]=gg;cnt[gg]++;//将一个分量中的元素染成一色
        }while(x!=k);
    }
}
int main(){
    cin>>n>>m;
    int x,y;
    step=0;
    memset(head,-1,sizeof head);
    for (int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        add(x,y);
    }
    tot=0,gg=0;
    for (int i=1;i<=n;i++){
        if (!dfn[i])tarjan(i);
    }
    for (int i=1;i<=gg;i++){
        printf("%d ",cnt[i]);
    }puts("");
    return 0;
}
```

点双连通分量(割点)

每个分量内，两点有两条及以上路径可以到达，即不存在割点

$O(N+M)$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+10;
int n,m,k;
int tot,dfn[N],low[N];
int top,sta[N];
int gg;vector<int>S[N];//点双连通分量
int ans[N];//ans[i]为去除i后新形成的块数,ans[i]>0,i为割点
vector<int>v[N];
void tarjan(int x,int fa){
    low[x]=dfn[x]=++tot;
    for(int i:v[x]){
        int v=i;
        if(!dfn[v]){
            sta[++top]=v;
            tarjan(v,x);
            low[x]=min(low[v],low[x]);
            if(low[v]>=dfn[x]){
                ans[x]++;
                gg++;
                while(sta[top]!=v)S[gg].push_back(sta[top--]);//将点出栈直到目标点
                S[gg].push_back(sta[top--]);//目标点出栈
                S[gg].push_back(x);//不要忘了将当前点存入S
            }
        }
```

```
        else if(v!=fa)low[x]=min(low[x],dfn[v]);
    }
}
int main(){
    int T=1;
    while (T--){
        cin>>n>>m;
        int x,y;
        for (int i=1;i<=m;i++){
            scanf("%d%d",&x,&y);
            v[x].push_back(y);v[y].push_back(x);
        }
        int num=0;
        for (int i=1;i<=n;i++){
            if (!dfn[i]){
                tarjan(i,-1);
                ans[i]--;
            }
        }
        for (int i=1;i<=n;i++)if (ans[i]>0)num++;//割点数
        cout<<num<<endl;
        for (int i=1;i<=n;i++){
            if (ans[i]>0){//ans[i]为去除i后新形成的块数
                cout<<i<<" ";
            }
        }puts("");
        for (int i=1;i<=n;i++)v[i].clear();
        tot=0;for (int i=1;i<=n;i++)dfn[i]=low[i]=0;
        for (int i=1;i<=n;i++)ans[i]=0;
        top=0;
        for (int i=1;i<=gg;i++)S[i].clear();gg=0;
    }
    return 0;
}
```

边双连通分量(割边)

每个分量内，不存在割边

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+10;
int n,m,k;
int tot,dfn[N],low[N];
int top,sta[N];
int gg,color[N];#边双连通分量
vector<int>v[N];
int ans=0;
void tarjan(int x,int fa){//和强连通分量差不多
    dfn[x]=low[x]=++tot;
    sta[++top]=x;
    for(int i:v[x]){
        int v=i;
        if(!dfn[v]){
            tarjan(v,x);
            low[x]=min(low[x],low[v]);
```

```
            if(low[v]>dfn[x])ans++;//割边数 v-x为割边
        }
        else if(v!=fa)low[x]=min(low[x],dfn[v]);
    }
    if(dfn[x]==low[x]){
        ++gg;
        int h=0;
        do{
            h=sta[top];
            color[h]=gg;
            top--;
        }while(h!=x);
    }
}
int main(){
    int T=1;
    while (T--){
        cin>>n>>m;
        int x,y;
        for (int i=1;i<=m;i++){
            scanf("%d%d",&x,&y);
            v[x].push_back(y);v[y].push_back(x);
        }
        for (int i=1;i<=n;i++){
            if (!dfn[i]){
                tarjan(i,-1);
            }
        }
        cout<<ans<<endl;
        for (int i=1;i<=n;i++)v[i].clear();
        tot=0;for (int i=1;i<=n;i++)dfn[i]=low[i]=0;
        top=0;
        gg=0;ans=0;
    }
    return 0;
}
```

2-sat问题

2-sat意味着布尔变量的可满足性(Satisfiability)，NP 完全

给定n个布尔变量x_i,m个条件：$i, j, a, b, x_i = a|x_j = b$

原式	建图
$\neg a \vee b$	$a \to b \wedge \neg b \to \neg a$
$a \vee b$	$\neg a \to b \wedge \neg b \to a$
$\neg a \vee \neg b$	$a \to \neg b \wedge b \to \neg a$

ab相同，$\neg a \vee b, \neg b \vee a$

ab不同，$a \vee b, \neg a \vee \neg b$

$O(N + M)$

图连接起来，同一强连通分量的值相同

输出一组解

```
#include<bits/stdc++.h>
using namespace std;
//#define int long long
const int N=2e6+10;
int n,m;
vector<int>v[N];
int color[N],dfn[N],cnt[N],low[N],gg=0,tot=0;
bool ff[N];
stack<int>s;
void tarjan(int x){
    dfn[x]=low[x]=++tot;//表示点i是第几个被遍历到的，把low[i]的值先赋值为dfn[i]的值
    s.push(x);
    ff[x]=1;
    for(int i:v[x]){
        int v=i;
        if(!dfn[v]){
            tarjan(v);
            low[x]=min(low[x],low[v]);
        }
        else if(ff[v])low[x]=min(low[x],dfn[v]);
    }
    int k;
    if(low[x]==dfn[x]){
        ++gg;
        do{
            k=s.top();s.pop();
            ff[k]=0;
            color[k]=gg;cnt[gg]++;//将一个分量中的元素染成一色
        }while(x!=k);
    }
}
signed main(){
    cin>>n>>m;
    for (int i=1;i<=m;i++){//对x点标号为x,-x标号为x+n
        int a,b,va,vb;
        scanf("%d%d%d%d",&a,&va,&b,&vb);
        if (va&&vb){// a, b 都真，-a -> b, -b -> a
            v[a+n].push_back(b);
            v[b+n].push_back(a);
        }
        else if(!va&&vb){// a 假 b 真，a -> b, -b -> -a
            v[a].push_back(b);
            v[b+n].push_back(a+n);
        }
        else if(va&&!vb){// a 真 b 假，-a -> -b, b -> a
            v[a+n].push_back(b+n);
            v[b].push_back(a);
        }
        else if(!va&&!vb){// a, b 都假，a -> -b, b -> -a
            v[a].push_back(b+n);
            v[b].push_back(a+n);
        }
        /*
        v[a+n*(va&1)].push_back(b+n*(vb^1));
        v[b+n*(vb&1)].push_back(a+n*(va^1));
```

```
        */
    }
    tot=0,gg=0;
    for (int i=1;i<=2*n;i++){
        if (!dfn[i])tarjan(i);
    }
    for (int i=1;i<=n;i++){
        if (color[i]==color[i+n]){//x 与 -x 在同一强连通分量内，一定无解
            puts("IMPOSSIBLE");return 0;
        }
    }
    puts("POSSIBLE");
    for (int i=1;i<=n;i++)printf("%d ",color[i]<color[i+n]);//如果不使用Tarjan找
环，请改成大于号
    puts("");
    return 0;
}
```

Cayley公式和prufer序列

Cayley公式是一个完全图K_n有n^(n-2)棵生成树，换句话说n个节点的带标号的无根树

有n^(n-2)个。

无根树转化为Prufer序列：找到编号最小的叶节点，删除这个节点，然后与这个叶节点相连

的点计入序列，直到这棵树只剩下两个节点、一条边

Prufer序列转化为无根树：设点集V={1,2,3,...,n}，每次取出Prufer序列中最前面的元素u，

在V中找到编号最小的没有在Prufer序列中出现的元素v，给u，v连边然后从序列中删除v，

最后在V中剩下两个节点，给它们连边。最终得到的就是无根树。

具体实现也可以用一个set，维护Prufer序列中没有出现的编号。复杂度O(nlogn)。

Prufer序列

一个 n 个节点的树可以 n-2 个 [1,n] 之间的数表示，这些数形成的序列叫作 Prufer 序列，树与 Prufer 序列一一对应

无根树转化为 Prufer 序列：

找到编号最小的度数为1的点，删除该节点并在序列中添加与该节点相连的节点的编号，重复这样的过程直到整个树只剩下两个节点

Prufer 序列转化为无根树：

取出 Prufer 序列中最前面的元素 u ，在点集中找到编号最小的没有在 Prufer 序列中出现的元素v，给 u , v 连边，然后在点集中删除 v ，重复这样的过程直到点集中只剩两个点，最后给这两个点连边

Prufer序列的性质：

1. Cayley 公式：n个点的无向完全图的生成树有 n^{n-2} 种
2. Prufer序列中某个编号出现的次数就等于这个编号的节点在无根树中的度数-1
3. n 个节点的度依次为 $d_1, d_2, \ldots, d_n$ 的树共有 $\frac{(n-2)!}{\prod_{i=1}^{n}(d_i-1)!}$ 个
4. n 个点的有根树有 n^{n-1} 种
5. n 个点 m 条边的无向图，有k个连通块，加 k-1 条边使整个图连通，方案数为 $n^{k-2}\prod_{i=1}^{k} s_i$ （其中 s_i 为每个连通块的大小）