最小树形图

给定包含 n个结点， m 条有向边的一个图。试求一棵以结点 r为根的最小树形图，并输出最小树形图每条边的权值之和

```
int n,m;
int rt;
struct node{
   int u,v;
   long long w;
}p[500005];
long long  in[500005],pre[500005],vis[500005],id[500005];
inline long long solve(){
     long long ans=0;
     int cnt=0,u,v;
     long long lazy;
    while(1){
        F(i,1,n){
            in[i]=1e15;
            id[i]=vis[i]=0;
        }
        F(i,1,m){
            if(p[i].u^p[i].v&&chkmin(in[p[i].v],p[i].w)){
            pre[p[i].v]=p[i].u;
            }
        }
        in[rt]=0;
        F(i,1,n){
            if(in[i]==1e15)return -1;
            ans+=in[i];
            for(u=i;u!=rt&&vis[u]!=i&&!id[u];u=pre[u])vis[u]=i;
            if(u!=rt&&!id[u]){
                id[u]=++cnt;
                for(v=pre[u];v!=u;v=pre[v])id[v]=cnt;
            }
        }
        if(!cnt)return ans;
        F(i,1,n)if(!id[i])id[i]=++cnt;
        F(i,1,m){
            lazy=in[p[i].v];
            if((p[i].u=id[p[i].u])!=(p[i].v=id[p[i].v]))
                p[i].w-=lazy;
        }
        n=cnt;rt=id[rt];cnt=0;
    }
}
int main () {
#ifndef ONLINE_JUDGE
file("4716");
#endif
    n=read();
    m=read();
    rt=read();
//  long long l=0;
```

```
    F(i,1,m){
       p[i].u=read(),p[i].v=read(),p[i].w=read();
//     l+=p[i].w;
    }
  //  rt=n+1;
/*  F(i,1,n){
     p[i+m].u=rt;
     p[i+m].v=i;
     p[i+m].w=l;
    }
    m=n+m;
    n=n+1;*/
//  long long ans=solve();
//  if(ans>=l)ans=-1;
    printf("%lld\n",solve());
    return 0;
}
//tarjan
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
#define maxn 102
#define INF 0x3f3f3f3f

struct UnionFind {
  int fa[maxn << 1];
  UnionFind() { memset(fa, 0, sizeof(fa)); }
  void clear(int n) { memset(fa + 1, 0, sizeof(int) * n); }
  int find(int x) { return fa[x] ? fa[x] = find(fa[x]) : x; }
  int operator[](int x) { return find(x); }
};
struct Edge {
  int u, v, w, w0;
};
struct Heap {
  Edge *e;
  int rk, constant;
  Heap *lch, *rch;
  Heap(Edge *_e) : e(_e), rk(1), constant(0), lch(NULL), rch(NULL) {}
  void push() {
    if (lch) lch->constant += constant;
    if (rch) rch->constant += constant;
    e->w += constant;
    constant = 0;
  }
};
Heap *merge(Heap *x, Heap *y) {
  if (!x) return y;
  if (!y) return x;
  if (x->e->w + x->constant > y->e->w + y->constant) swap(x, y);
  x->push();
  x->rch = merge(x->rch, y);
  if (!x->lch || x->lch->rk < x->rch->rk) swap(x->lch, x->rch);
  if (x->rch)
    x->rk = x->rch->rk + 1;
  else
```

```
    x->rk = 1;
  return x;
}
Edge *extract(Heap *&x) {
  Edge *r = x->e;
  x->push();
  x = merge(x->lch, x->rch);
  return r;
}

vector<Edge> in[maxn];
int n, m, fa[maxn << 1], nxt[maxn << 1];
Edge *ed[maxn << 1];
Heap *Q[maxn << 1];
UnionFind id;

void contract() {
  bool mark[maxn << 1];
  // 将图上的每一个结点与其相连的那些结点进行记录。
  for (int i = 1; i <= n; i++) {
    queue<Heap *> q;
    for (int j = 0; j < in[i].size(); j++) q.push(new Heap(&in[i][j]));
    while (q.size() > 1) {
      Heap *u = q.front();
      q.pop();
      Heap *v = q.front();
      q.pop();
      q.push(merge(u, v));
    }
    Q[i] = q.front();
  }
  mark[1] = true;
  for (int a = 1, b = 1, p; Q[a]; b = a, mark[b] = true) {
    //寻找最小入边以及其端点，保证无环。
    do {
      ed[a] = extract(Q[a]);
      a = id[ed[a]->u];
    } while (a == b && Q[a]);
    if (a == b) break;
    if (!mark[a]) continue;
    // 对发现的环进行收缩，以及环内的结点重新编号，总权值更新。
    for (a = b, n++; a != n; a = p) {
      id.fa[a] = fa[a] = n;
      if (Q[a]) Q[a]->constant -= ed[a]->w;
      Q[n] = merge(Q[n], Q[a]);
      p = id[ed[a]->u];
      nxt[p == n ? b : p] = a;
    }
  }
}

ll expand(int x, int r);
ll expand_iter(int x) {
  ll r = 0;
  for (int u = nxt[x]; u != x; u = nxt[u]) {
    if (ed[u]->w0 >= INF)
      return INF;
    else
```

```
      r += expand(ed[u]->v, u) + ed[u]->w0;
  }
  return r;
}
ll expand(int x, int t) {
  ll r = 0;
  for (; x != t; x = fa[x]) {
    r += expand_iter(x);
    if (r >= INF) return INF;
  }
  return r;
}
void link(int u, int v, int w) { in[v].push_back({u, v, w, w}); }

int main() {
  int rt;
  scanf("%d %d %d", &n, &m, &rt);
  for (int i = 0; i < m; i++) {
    int u, v, w;
    scanf("%d %d %d", &u, &v, &w);
    link(u, v, w);
  }
  // 保证强连通
  for (int i = 1; i <= n; i++) link(i > 1 ? i - 1 : n, i, INF);
  contract();
  ll ans = expand(rt, n);
  if (ans >= INF)
    puts("-1");
  else
    printf("%lld\n", ans);
  return 0;
}
```

负环

```
int n,m;
int he[2005],to[6005],ne[6005],cost[6005],e;
void add(int x,int y,int z){
     to[++e]=y;
     ne[e]=he[x];
     he[x]=e;
     cost[e]=z;
}
int dis[100005];
int sum[100005];
queue<int>q;
int vis[100005];
bool spfa(int x){
    Set(vis,0);
    Set(dis,0x3f3f3f3f);
    while(!q.empty())q.pop();
    q.push(x);
    dis[x]=0;
    vis[x]=1;
    sum[x]=0;
    while(!q.empty()){
       int u=q.front();
       q.pop();
       for(int i=he[u];i;i=ne[i]){
          int v=to[i];
          if(chkmin(dis[v],dis[u]+cost[i])){
              sum[v]=sum[u]+1;
              if(sum[v]>=n)return 1;
              if(!vis[v])q.push(v),vis[v]=1;
          }
       }
       vis[u]=0;
    }
    return 0;
}
int main () {
#ifndef ONLINE_JUDGE
file("a");
#endif
    int T=read();
    while(T--){
        Set(he,0);
        e=0;
       n=read(),m=read();
       F(i,1,m){
          int x=read(),y=read(),z=read();
          add(x,y,z);
          if(z>=0)add(y,x,z);
       }
       if(spfa(1))puts("YES");
       else puts("NO");
    }
    return 0;
}
```

tarjan缩点

给定一个 n个点 m 条边有向图，每个点有一个权值，求一条路径，使路径经过的点权值之和最大。你只需要求出这个权值和。

```
int he[20005],to[200005],ne[200005],e;
int w[20005];
int n,m;
int dfn[20005],low[20005],col[20005],colcnt,dfnclock,stk[20005],top;
void add(int x,int y){
  to[++e]=y;
  ne[e]=he[x];
  he[x]=e;
}
void tarjan(int x){
    dfn[x]=low[x]=++dfnclock;
    stk[++top]=x;
    for(int i=he[x];i;i=ne[i]){
      int v=to[i];
      if(!dfn[v]){
        tarjan(v);
        chkmin(low[x],low[v]);
      }
      else if(!col[v])chkmin(low[x],dfn[v]);
    }
    if(dfn[x]==low[x]){
       colcnt++;
       while(stk[top]!=x){
          col[stk[top]]=colcnt;
          top--;
       }
       col[stk[top]]=colcnt;
       top--;
    }
}
int dp[200005];
int st[100005],ed[100005];
int DP(int u){
    if(dp[u])return dp[u];
    dp[u]=w[u];
    for(int i=he[u];i;i=ne[i]){
        int v=to[i];
        chkmax(dp[u],DP(v)+w[u]);
    }
    return dp[u];
}
int main () {
#ifndef ONLINE_JUDGE
file("3387");
#endif
    n=read();
    m=read();
    F(i,1,n){
       w[i]=read();
    }
    F(i,1,m){
      int x=read(),y=read();
      add(x,y);
      st[i]=x,ed[i]=y;
    }
    colcnt=n;
    F(i,1,n)if(!col[i])tarjan(i);
    F(i,1,n){
       w[col[i]]+=w[i];
    }
    F(i,1,m){
      if(col[st[i]]!=col[ed[i]]){
         add(col[st[i]],col[ed[i]]);
      }
    }
    int ans=0;
    F(i,n+1,colcnt)chkmax(ans,DP(i));

    printf("%d\n",ans);
    return 0;
}
```

割点

```
int n,m;
int iscut[100005],dfn[100005],low[100005],dfs_clock;
int he[100005],to[200005],ne[200005],e;
void add(int x,int y){
   to[++e]=y;
   ne[e]=he[x];
   he[x]=e;
}
int tarjan(int x,int ff){
    int son=0;
    int lowu=dfn[x]=++dfs_clock;
     for(int i=he[x];i;i=ne[i]){
        int v=to[i];
        if(!dfn[v]){
           son++;
            int lowv=tarjan(v,x);
            chkmin(lowu,lowv);
            if(lowv>=dfn[x])iscut[x]=1;
        }
        else if(v!=ff&&dfn[v]<dfn[x]){
           chkmin(lowu,dfn[v]);
        }
     }
     if(son==1&&ff==0)iscut[x]=0;
     low[x]=lowu;
     return lowu;
}
int main () {
#ifndef ONLINE_JUDGE
file("3388");
#endif
    n=read();
    m=read();
    F(i,1,m){
      int x=read(),y=read();
      add(x,y);
      add(y,x);
    }
    F(i,1,n){
      if(!dfn[i])tarjan(i,0);
    }
    int ans=0;
    F(i,1,n)if(iscut[i])ans++;
    printf("%d\n",ans);
    F(i,1,n){
      if(iscut[i])printf("%d ",i);
    }
    return 0;
}
```

dij

```
int n,m,s;
int he[100005],to[200005],ne[200005],cost[200005],e;
int dis[100005];
void add(int x,int y,int z){
     to[++e]=y;
     ne[e]=he[x];
     he[x]=e;
     cost[e]=z;
}
#define PII pair<int,int>
int flag[100005];
priority_queue<PII,vector<PII> ,greater<PII> >Q;
void spfa(){
    F(i,1,n)dis[i]=2147483647;
   dis[s]=0;
   Q.push(make_pair(dis[s],s));
   while(!Q.empty()){
    PII now=Q.top();
    Q.pop();
     if(flag[now.second])continue;
    flag[now.second]=1;
    for(int i=he[now.second];i;i=ne[i]){
         int v=to[i];
        if(dis[now.second]+cost[i]<dis[v]){
            dis[v]=dis[now.second]+cost[i];
            if(!flag[v])
              Q.push(make_pair(dis[v],v));
        }
    }
   }
}
int main () {
#ifndef ONLINE_JUDGE
file("4779");
#endif
    n=read();
    m=read();
    s=read();
    F(i,1,m){
       int x=read(),y=read(),z=read();
       add(x,y,z);
    }
    spfa();
    F(i,1,n)printf("%d ",dis[i]);
    return 0;
}
```

支配树

给定一张有向图，求从1号点出发，每个点能支配的点的个数（包括自己）

对于一张有向图，我们确定一个起点 S。

对于一个点 k，称 x 为其支配点当且仅当，删去点 x后，S无法到达 k，即 S到达 k 的所有路径都必须通过 x。

容易发现，一个点的支配点不止一个。

同时，也很容易发现，如果我们将一个点 k与其最近的支配点连边，那么会形成一个树形结构，这个树形结构即支配树。

```
const int N=3e5+10;
int n,m,ans[N],dfn[N],id[N],tot,dep[N];
int anc[N],fa[N],semi[N],mn[N],in[N],ff[20][N];
int q[N],top;
queue<int> Q;
vector<int> E[N];
struct Map{
    struct edge {int next,to;}a[N<<1];
    int head[N],cnt;
    void reset() {cnt=0;memset(head,0,sizeof(head));}
    void link(int x,int y) {a[++cnt]=(edge){head[x],y};head[x]=cnt;}
}A,rA,B,C;
void reset(){
    A.reset();rA.reset();B.reset();C.reset();
    tot=top=0;
    for(int i=1;i<=n;i++){
        dfn[i]=id[i]=ans[i]=in[i]=anc[i]=dep[i]=0;
        fa[i]=mn[i]=semi[i]=i;
        E[i].clear();
        for(int j=0;j<=18;j++) ff[j][i]=0;
    }
}
void dfs(int x,int fr){
    dfn[x]=++tot;id[tot]=x;
    anc[x]=fr;B.link(fr,x);
    for(int i=A.head[x];i;i=A.a[i].next)
        if(!dfn[A.a[i].to]) dfs(A.a[i].to,x);
}
void dfscalc(int x,int fr){
    ans[x]=1;
    for(int i=C.head[x];i;i=C.a[i].next)
        dfscalc(C.a[i].to,x),ans[x]+=ans[C.a[i].to];
}
int find(int x){
    if(x==fa[x]) return x;
    int tt=fa[x];fa[x]=find(fa[x]);
    if(dfn[semi[mn[tt]]]<dfn[semi[mn[x]]]) mn[x]=mn[tt];
    return fa[x];
}
int LCA(int x,int y){
    if(dep[x]<dep[y]) swap(x,y);
    int d=dep[x]-dep[y];
    for(int i=18;i>=0;i--) if(d&(1<<i)) x=ff[i][x];
    for(int i=18;i>=0;i--)
        if(ff[i][x]^ff[i][y])
            x=ff[i][x],y=ff[i][y];
    return x==y?x:ff[0][x];
}
void Work(){
    dfs(n,0);
    for(int w=n;w>=2;w--){
        int x=id[w],res=n;
        if(!x) continue;
        for(int i=rA.head[x];i;i=rA.a[i].next){
            int R=rA.a[i].to;
            if(!dfn[R]) continue;
            if(dfn[R]<dfn[x]) res=min(res,dfn[R]);
            else find(R),res=min(res,dfn[semi[mn[R]]]);
        }
        semi[x]=id[res];fa[x]=anc[x];B.link(semi[x],x);
    }
    for(int x=1;x<=n;x++)
        for(int i=B.head[x];i;i=B.a[i].next)
            in[B.a[i].to]++,E[B.a[i].to].pb(x);
    for(int x=1;x<=n;x++) if(!in[x]) Q.push(x);
    while(!Q.empty()){
        int x=Q.front();Q.pop();q[++top]=x;
        for(int i=B.head[x];i;i=B.a[i].next)
            if(!--in[B.a[i].to]) Q.push(B.a[i].to);
    }
    for(int i=1;i<=top;i++){
        int x=q[i],f=0,l=E[x].size();
        if(l) f=E[x][0];
        for(int j=1;j<l;j++) f=LCA(f,E[x][j]);
        ff[0][x]=f;dep[x]=dep[f]+1;C.link(f,x);
        for(int p=1;p<=18;p++) ff[p][x]=ff[p-1][ff[p-1][x]];
    }
    ans[0]=0;dfscalc(n,0);
}
int main(){
    cin>>n>>m;
    reset();
    for(int i=1,x,y;i<=m;i++){
        scanf("%d%d",&x,&y);
        x=n-x+1;y=n-y+1;
        A.link(x,y),rA.link(y,x);
    }
    Work();
    for(int i=n;i>=1;i--) printf("%d ",ans[i]);
    return 0;
}
```

2-sat

```
int n,m;
int he[3000005],to[3000005],ne[3000005],e;
void add(int x,int y){
   to[++e]=y;
   ne[e]=he[x];
   he[x]=e;
}
int top,low[2500005],pre[2500005],tim,stk[2500005],scc[2500005],cnt;
void tarjan(int x){
    low[x]=pre[x]=++tim;
    stk[++top]=x;
    for(int i=he[x];i;i=ne[i]){
        int v=to[i];
        if(!pre[v]){
          tarjan(v);
          chkmin(low[x],low[v]);
        }
        else if(!scc[v])chkmin(low[x],pre[v]);
   }
   if(low[x]==pre[x])for(++cnt;stk[top+1]^x;--top)
        scc[stk[top]]=cnt;
}
int main () {
#ifndef ONLINE_JUDGE
file("4782");
#endif
   n=read();
   m=read();
   F(i,1,m){
      int x=read(),a=read(),y=read(),b=read();
      if(x==y){
            if(a==b)add(x<<1|a^1,(x<<1|a));
            continue;
        }
      add(x*2+(a^1),y*2+b);
      add(y*2+(b^1),x*2+a);
   }
   F(i,2,2*n+1){
     if(!scc[i])tarjan(i);
     if(scc[i]&&scc[i^1]&&scc[i]==scc[i^1]){
       puts("IMPOSSIBLE");
       return 0;
     }
   }
   puts("POSSIBLE");
   F(i,1,n){
         printf("%d ",scc[i<<1]>scc[i<<1|1]);
   }
    return 0;
}
```

最小割树

```
int n,m;
int ans[505][505];
int he[505],to[300005],ne[300005],w[300005],e=1;
int cur[505];
int dis[505];
int vis[505];
void add(int x,int y,int z){
    to[++e]=y;
    ne[e]=he[x];
    he[x]=e;
    w[e]=z;
}
int A[505],B[505];
void clear(){
   for(int i=2;i<=e;i+=2)w[i]=w[i^1]=(w[i]+w[i^1])/2;
}
int S,T;
int fa[505];
bool bfs(){
   static queue<int>q;
   Set(dis,-1);
   dis[S]=1;
   q.push(S);
   while(!q.empty()){
       int x=q.front();
       q.pop();
       for(int i=he[x];i;i=ne[i]){
          int v=to[i];
          if(dis[v]==-1&&w[i]){
             dis[v]=dis[x]+1;
              q.push(v);
          }
       }
   }
   return dis[T]!=-1;
}
int dfs(int x,int flow){
   if(x==T||!flow)return flow;
   int res=0,f;
   for(int &i=cur[x];i;i=ne[i]){
      int v=to[i];
      if(dis[v]==dis[x]+1&&(f=dfs(v,min(flow,w[i])))){
           res+=f;
           flow-=f;
           w[i]-=f;
           w[i^1]+=f;
           if(flow==0)break;
      }
   }
   return res;
}
int dinic(){
    int res=0;
    while(bfs()){
```

```
    //  cerr<<"done"<<endl;
       memcpy(cur,he,sizeof(cur));
       res+=dfs(S,0x3f3f3f3f);
    }
    return res;
}
void dfs(int x){
     vis[x]=1;
    for(int i=he[x];i;i=ne[i]){
         int v=to[i];
         if(w[i]&&!vis[v])dfs(v);
    }
}
void solve(int l,int r){
   if(l==r)return ;
   clear();
   S=A[l],T=A[r];
   int cnt=dinic();
   Set(vis,0);
   dfs(S);
   F(i,1,n){
     if(vis[i]){
       F(j,1,n){
         if(!vis[j]){
            chkmin(ans[i][j],cnt);
            ans[j][i]=ans[i][j];
         }
       }
     }
   }
   int L=l-1,R=r+1;
     F(i,l,r){
        if(!vis[A[i]])B[++L]=A[i];
        else B[--R]=A[i];
     }
     F(i,l,r)A[i]=B[i];
     solve(l,L),solve(R,r);
}
int main () {
#ifndef ONLINE_JUDGE
file("4897");
#endif
    n=read();
    m=read();
    F(i,1,m){
       int u=read(),v=read(),z=read();
       add(u,v,z);
       add(v,u,z);
    }
    F(i,1,n)A[i]=i,fa[i]=1;
    Set(ans,0x3f3f3f3f);
    //solve(1,n);
    F(i,2,n){
      clear();
      int v=fa[i];
      S=i;
      T=v;
//    cerr<<S<<" "<<T<<endl;
```

```
      int w=dinic();
//    cerr<<i<<endl;
      Set(vis,0);
      F(j,1,i-1)ans[i][j]=ans[j][i]=min(ans[j][fa[i]],w);
      dfs(i);
      F(j,i+1,n)if(vis[j]&&fa[i]==fa[j])fa[j]=i;
    }
    int q=read();
    while(q--){
       int x=read(),y=read();
       printf("%d\n",ans[x][y]);
    }
    return 0;
}
```

（严格）次小生成树

先跑一边Kruskal，并在LCT上跑出最小生成树的样子

然后对于每一条非树边，加入后会产生一个环，我们删去环内最大值（若最大值和改变相同则插入次大值）即可

```
#include<bits/stdc++.h>
using namespace std;
#define il inline
#define re register
#define debug printf("Now is Line : %d\n",__LINE__)
#define file(a) freopen(#a".in","r",stdin);//freopen(#a".out","w",stdout)
#define inf 1234567890000000000
#define ll long long
il int read() {
    re int x = 0, f = 1; re char c = getchar();
    while(c < '0' || c > '9') { if(c == '-') f = -1; c = getchar();}
    while(c >= '0' && c <= '9') x = x * 10 + c - 48, c = getchar();
    return x * f;
}
#define rep(i, s, t) for(re int i = s; i <= t; ++ i)
#define get_fa(x) ch[1][fa[x]] == x
#define isroot(x) ch[1][fa[x]] == x || ch[0][fa[x]] == x
#define updown(x) tag[x] ^= 1, swap(ch[0][x], ch[1][x])
#define maxn 600005
struct edge {
    int u, v, w;
}e[maxn];
int n, m, is[maxn],ch[2][maxn], fa[maxn], tag[maxn], st[maxn], Fa[maxn];
int mx1[maxn], mx2[maxn], val[maxn];
ll ans, Ans = inf;
il int find(int x) {
    while(Fa[x] != x) x = Fa[x] = Fa[Fa[x]];
    return x;
}
il bool cmp(edge a, edge b) {return a.w < b.w;}
il void pushdown(int x) {
    if(!tag[x]) return;
    if(ch[0][x]) updown(ch[0][x]);
    if(ch[1][x]) updown(ch[1][x]);
    tag[x] = 0;
}
il void pushup(int x) {
    mx1[x] = val[x];
    if(mx1[x] < mx1[ch[0][x]]) mx2[x] = mx1[x], mx1[x] = mx1[ch[0][x]];
    else if(mx1[x] > mx1[ch[0][x]]) mx2[x] = max(mx2[x], mx1[ch[0][x]]);
    if(mx1[x] < mx1[ch[1][x]]) mx2[x] = mx1[x], mx1[x] = mx1[ch[1][x]];
    else if(mx1[x] > mx1[ch[1][x]]) mx2[x] = max(mx2[x], mx1[ch[1][x]]);
    mx2[x] = max(max(mx2[x], mx2[ch[1][x]]), mx2[ch[0][x]]);
}
il void rotate(int x) {
    int y = fa[x], z = fa[y], w = get_fa(x), k = get_fa(y);
    if(isroot(y)) ch[k][z] = x; fa[x] = z;
    ch[w][y] = ch[w ^ 1][x], fa[ch[w ^ 1][x]] = y;
    ch[w ^ 1][x] = y, fa[y] = x;
    pushup(y), pushup(x);
}
il void Splay(int x) {
    int top = 0, y = x;
    st[++ top] = y;
    while(isroot(y)) st[++ top] = y = fa[y];
    while(top) pushdown(st[top --]);
    while(isroot(x)) {
        if(isroot(fa[x])) rotate(get_fa(x) == get_fa(fa[x]) ? fa[x] : x);
        rotate(x);
    }
}
il void access(int x) {for(re int y = 0; x; x = fa[y = x]) Splay(x), ch[1][x] =
y, pushup(x);}
il void makeroot(int x) {access(x), Splay(x), updown(x);}
il int findroot(int x) {
    access(x), Splay(x);
    while(ch[0][x]) x = ch[0][x];
    return Splay(x), x;
}
il void spilt(int x, int y) {makeroot(x), access(y), Splay(y);}
il void link(int x, int y) {
    makeroot(x);
    if(findroot(y) != x) fa[x] = y;
}
int main() {
    n = read(), m = read();
    rep(i, 1, m) e[i].u = read(), e[i].v = read(), e[i].w = read();
    rep(i, 1, n) Fa[i] = i;
    sort(e + 1, e + 1 + m, cmp);
    rep(i, 1, m) {
        val[i + n] = e[i].w;
        int u = e[i].u, v = e[i].v, a = find(u), b = find(v);
        if(a != b) ans += e[i].w, link(u, i + n), link(i + n, v), is[i] = 1,
Fa[a] = b;
    }
    rep(i, 1, m) {
        if(is[i]) continue;
        int u = e[i].u, v = e[i].v;
        spilt(u, v);
        if(e[i].w > mx1[v]) Ans = min(Ans, (ll)e[i].w - mx1[v]);
        else Ans = min(Ans, (ll)e[i].w - mx2[v]);
    }
    printf("%lld", Ans + ans);
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
#define re register
#define il inline
#define int long long //把所有int转成longlong
#define debug printf("Now is line %d\n",__LINE__);
il int read(){
    re int x=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-') f=-1;c=getchar();}
    while(c>='0'&&c<='9') x=(x<<3)+(x<<1)+(c^48),c=getchar();
    return x*f;
}//快读
#define maxm 300005
#define inf 12345678900000000
#define maxn 100005
struct Edge{int u,v,w,next;}e[maxm<<1];
struct qj{int ma,ma2;}q[maxn<<2];
struct Edge1{
    int u,v,w;
    bool operator <(const Edge1 &x) const{return w<x.w;}//按照边权排序
}edge[maxm];
int n,m,vis[maxm],ans=inf,head[maxn],cnt,fa[maxn],mtree;
il void add(int u,int v,int w){
    e[++cnt].v=v;
    e[cnt].w=w;
    e[cnt].next=head[u];
    head[u]=cnt;
}//前向星加边
namespace smallesttree{
    il int find(int x){
        while(x!=fa[x]) x=fa[x]=fa[fa[x]];
        return x;
    }//并查集找祖先
    il void init(){
        for(re int i=1;i<=n;i++) fa[i]=i; //预处理并查集
        for(re int i=0;i<m;i++)
edge[i].u=read(),edge[i].v=read(),edge[i].w=read();
    }
    il void kruskal(){
        init();
        sort(edge,edge+m);
        re int T=0;
        for(re int i=0;i<m;++i){
            re int eu=find(edge[i].u),ev=find(edge[i].v);//寻找祖先
            if(eu!=ev){

add(edge[i].u,edge[i].v,edge[i].w),add(edge[i].v,edge[i].u,edge[i].w);
                mtree+=edge[i].w;//记录子树大小
                fa[ev]=eu;//合并
                vis[i]=1;//标记该边为树边
                if(++T==n-1) break;//边数等于节点数+1即为一颗树
            }
        }
    }
}
//求出最小生成树
namespace treecut{
    int
dep[maxn],father[maxn],top[maxn],W[maxn],a[maxn],size[maxn],son[maxn],seg[maxn],
col;
    //dep:深度 father:父亲节点 top:重链的顶端 W:到根节点的距离 a:点的权值 size:子树大小
son:重儿子 seg:在线段树中的序号（dfs序）
    il void dfs1(int u,int fr){
        dep[u]=dep[fr]+1;
        size[u]=1;
        father[u]=fr;
        for(re int i=head[u];i;i=e[i].next){
            re int v=e[i].v;
            if(v!=fr){
                W[v]=W[u]+e[i].w;//W为每一个点到根节点的距离
                dfs1(v,u);
                size[u]+=size[v];
                if(size[v]>size[son[u]]) son[u]=v;
            }
        }
    }//预处理出dep、size、father以及son
    il void dfs2(int now,int fi){
        top[now]=fi;
        seg[now]=++col;
        a[col]=W[now]-W[father[now]];//a为点的权值（它与之父亲节点边的权值）（相当于前缀
和）
        if(!son[now]) return;
        dfs2(son[now],fi);
        for(re int i=head[now];i;i=e[i].next){
            re int v=e[i].v;
            if(v!=son[now]&&v!=father[now]) dfs2(v,v);
        }
    }//预处理出每个节点的top、seg以及权值
    //树剖模板就不解释了
    #define ls k<<1
    #define rs k<<1|1
    il bool CMP(int a,int b){return a>b;}
    il int getse(int x,int g,int z,int c){
        re int a[5]={x,g,z,c};
        sort(a,a+4,CMP);
        for(re int i=1;i<3;++i){
            if(a[i]!=a[0]) return a[i];
        }
    }//找到两个区间的最大值和严格次大值（四个数）的最大值与严格次大值
    // 就是合并两个区间的最大值和严格次大值
    il void build(int k,int l,int r){
        if(l==r){
            q[k].ma=a[l];
            return;
        }
        re int mid=(l+r)>>1;
        build(ls,l,mid),build(rs,mid+1,r);
        q[k].ma=max(q[ls].ma,q[rs].ma);
        q[k].ma2=getse(q[ls].ma,q[rs].ma,q[ls].ma2,q[rs].ma2);
    }//预处理出区间最大值与次大值
    il qj query(int k,int l,int r,int ll,int rr){
        if(ll>r||rr<l) return (qj){-inf,-inf};
        if(ll<=l&&rr>=r) return (qj){q[k].ma,q[k].ma2};
        re int mid=(l+r)>>1;
        re qj t1=query(ls,l,mid,ll,rr),t2=query(rs,mid+1,r,ll,rr);
        return (qj){max(t1.ma,t2.ma),getse(t1.ma,t2.ma,t1.ma2,t2.ma2)};
    }//查询区间的区间的最大值与次小值
    il int LCA(int u,int v,int d){
        re int need=-inf;
        while(top[u]!=top[v]){
            if(dep[top[u]]<dep[top[v]]) swap(u,v);
            qj temp=query(1,1,n,seg[top[u]],seg[u]);
            u=father[top[u]];
            need=max(need,(temp.ma==d)?temp.ma2:temp.ma);//严格次小边（如果
temp.ma==k就是非严格次小）
        }
        if(dep[u]<dep[v]) swap(u,v);//找到LCA
        qj temp=query(1,1,n,seg[v]+1,seg[u]);
        return max(need,(temp.ma==d)?temp.ma2:temp.ma);//同上
    }
    il void init(){
        dfs1(1,0),dfs2(1,1),build(1,1,n);
    }
}
//树链剖分
signed main(){
    n=read(),m=read();
    smallesttree::kruskal();//求出最小生成树
    treecut::init();//预处理
    for(re int i=0;i<m;++i){
        if(vis[i]) continue;//枚举所有非树边（没有在最小生成树的边）
        re int temp=mtree/*最小生成树边权和*/+edge[i].w/*本来的树边的边权*/-
treecut::LCA(edge[i].u,edge[i].v,edge[i].w)/*找到严格次小边的边权*/;
        if(ans>temp&&temp!=mtree+e[i].w/*其实就是严格此小边不为0（没有找到严格次小边）
*/&&temp>mtree) ans=temp;
    }
    printf("%lld",ans);
    return 0;
}
```

边双和点双

```
#include<bits/stdc++.h>
using namespace std;
#define N 5000+5
#define E 10000+5
int head[N],cnt=1;
struct Node{
    int to,nxt;
}e[E];
void add(int u,int v){
    ++cnt;
    e[cnt].to=v;
    e[cnt].nxt=head[u];
    head[u]=cnt;
}
int n,r;
int dfn[N],low[N],bridge[N],t;
int c[N],dcc,du[N],leaf;
void tarjan(int u,int edge){//同求桥
    dfn[u]=low[u]=++t;
    for(int i=head[u];i!=-1;i=e[i].nxt){
        int v=e[i].to;
        if(!dfn[v]){
            tarjan(v,i);
            low[u]=min(low[u],low[v]);
            if(dfn[u]<low[v])bridge[i]=bridge[i^1]=1;
        }
        else if(i!=(edge^1))low[u]=min(low[u],dfn[v]);
    }
}
void dfs(int u){//求边双联通分量
    c[u]=dcc;
    for(int i=head[u];i!=-1;i=e[i].nxt){
        int v=e[i].to;
        if(c[v]||bridge[i])continue;//不能经过桥
        dfs(v);
    }
}
int main(){
    memset(head,-1,sizeof head);
    scanf("%d%d",&n,&r);
    for(int i=1;i<=r;i++){
        int x,y;scanf("%d%d",&x,&y);
        add(x,y);add(y,x);
    }
    for(int i=2;i<=cnt;i++)//从2开始哦!
        if(!dfn[e[i].to])tarjan(e[i].to,i);
    for(int i=1;i<=n;i++)
        if(!c[i]){dcc++;dfs(i);}//图不一定连通
    for(int i=1;i<=n;i++)
        for(int j=head[i];j!=-1;j=e[j].nxt)
        if(c[i]!=c[e[j].to])//不是同一个边双连通分量
            du[c[e[j].to]]++;//入度++
    for(int i=1;i<=dcc;i++)
    if(du[i]==1)leaf++;
    printf("%d",(leaf+1)/2);//考虑多余的，要向上取整
}
#include<cstdio>
#include<cstring>
#include<vector>
using namespace std;
#define pb(x) push_back(x)
typedef vector<int>::iterator pt;
const int maxn=1005;
int n,m;
vector<int> bcc[maxn];int bccN;
vector<int> to[maxn];
int E[maxn][maxn],eflag=0;
int dfn[maxn],low[maxn],prt[maxn],T;
int stk[maxn],top;
void dfs(int x){//这个函数是关键
  low[x]=dfn[x]=++T;
  for(pt i=to[x].begin();i!=to[x].end();++i){
    if(*i==prt[x])continue;//恰好沿着树边往回走？不可以的
    if(dfn[*i]==0){
      prt[*i]=x;
      stk[++top]=*i;//一个点对应一条树边,代表x到*i所连的树边
      dfs(*i);//递归搜索
      if(low[*i]>=dfn[x]){
    bcc[++bccN].clear();
    do{
      bcc[bccN].pb(stk[top]);
    }while(stk[top--]!=*i);
    bcc[bccN].pb(x);//注意点x还要单独加到这个BCC中
      }
      if(low[*i]<low[x])low[x]=low[*i];//更新low[x]
    }
    if(dfn[*i]<low[x])low[x]=dfn[*i];//更新low[x]
  }
}
int good[maxn],gflag;
int ufs[maxn],col[maxn];
int find(int x){
  if(x==ufs[x])return x;
  int rt=find(ufs[x]);
  col[x]^=col[ufs[x]];
  return ufs[x]=rt;
}
int bccbel[maxn],belT=0;
bool link(int x,int y){
  int rx=find(x),ry=find(y);
  if(rx!=ry){
    ufs[rx]=ry;
    col[rx]=col[x]^col[y]^1;
    return false;
  }else{
    return col[x]==col[y];
  }
}
bool check(int x){
  ++belT;
  for(pt i=bcc[x].begin();i!=bcc[x].end();++i){
    ufs[*i]=*i;col[*i]=0;bccbel[*i]=belT;
  }
  for(pt i=bcc[x].begin();i!=bcc[x].end();++i){
    for(pt j=to[*i].begin();j!=to[*i].end();++j){
      if(bccbel[*j]==belT){
    if(link(*i,*j))return true;
      }
    }
  }
  return false;
}
int main(){
  while(scanf("%d%d",&n,&m),n!=0){
    for(int i=1;i<=n;++i)to[i].clear();
    ++eflag;
    for(int i=1,a,b;i<=m;++i){
      scanf("%d%d",&a,&b);
      E[a][b]=E[b][a]=eflag;
    }

    for(int i=1;i<=n;++i)
      for(int j=1;j<=n;++j)
    if(i!=j&&E[i][j]!=eflag)to[i].pb(j);

    memset(dfn,0,sizeof(dfn));
    memset(prt,0,sizeof(prt));
    memset(low,0,sizeof(low));
    top=0;T=0;bccN=0;
    for(int i=1;i<=n;++i){
      if(dfn[i]==0)dfs(i);
    }
    ++gflag;
    for(int i=1;i<=bccN;++i){
      if(check(i)){
    for(pt j=bcc[i].begin();j!=bcc[i].end();++j){
      good[*j]=gflag;
    }
      }
    }
    int ans=0;
    for(int i=1;i<=n;++i){
      if(good[i]!=gflag)++ans;
    }
    printf("%d\n",ans);
  }
  return 0;
}
```

带花树

```
int n,m;
queue<int>q;
int match[505],vis[505],ans,cnt,tim[505],pre[505],fa[505];
int find(int x){return fa[x]==x?fa[x]:fa[x]=find(fa[x]);}
int he[505],to[320005],ne[320005],e;
void add(int x,int y){
   to[++e]=y;
   ne[e]=he[x];
   he[x]=e;
}
int LCA(int u,int v){
   for(++cnt;;swap(u,v)){
       if(u){
         u=find(u);
         if(tim[u]==cnt)return u;
         tim[u]=cnt;
         u=pre[match[u]];
       }
   }
}
void bloss(int u,int v,int lca){
    while(find(u)!=lca){
        pre[u]=v;
        v=match[u];
        if(vis[v]==2){
            vis[v]=1;
            q.push(v);
        }
        if(find(u)==u)fa[u]=lca;
        if(find(v)==v)fa[v]=lca;
        u=pre[v];
    }
}
int bfs(int x){
   F(i,1,n){
     fa[i]=i;
   }
     Set(vis,0);
     Set(pre,0);
     while(!q.empty())q.pop();
   q.push(x);
   vis[x]=1;
   while(!q.empty()){
       int u=q.front();
//     cerr<<u<<endl;
      q.pop();
      for(int i=he[u];i;i=ne[i]){
          int v=to[i];
        if(find(u)==find(v)||vis[v]==2)continue;
        if(!vis[v]){
           vis[v]=2;
           pre[v]=u;
           if(!match[v]){
               for(int t=v,lst;t;t=lst){
                  lst=match[pre[t]];
                  match[t]=pre[t];
                  match[pre[t]]=t;
               }
               return 1;
           }
           vis[match[v]]=1;
           q.push(match[v]);
        }
        else{
           int lca=LCA(u,v);
           bloss(u,v,lca);
           bloss(v,u,lca);
        }
      }
   }
   return 0;
}
int main () {
#ifndef ONLINE_JUDGE
file("79");
#endif
    n=read();
    m=read();
    F(i,1,m){
       int x=read(),y=read();
       add(x,y);
       add(y,x);
    }
    F(i,1,n){
      if(!match[i])ans+=bfs(i);
    }
    printf("%d\n",ans);
    F(i,1,n){
      printf("%d%c",match[i],i==n?'\n':' ');
    }
    return 0;
}
```

有向图欧拉路

一个有向图存在欧拉路径当且仅当每个点的出入度之差的绝对值小于等于1且之多有一个出度为1的点。

字典序最小。

```
int eulerian_path(int* r, int w) {
    static int s[N], t; s[++t] = w; int m = 0;
    while(t) {
        int u = s[t];
        if (g[u].empty()) r[++m] = u, --t;
        else s[++t] = g[u].back(), g[u].pop_back();
    }
    reverse(r + 1, r + m + 1); return m;
}
//  Example: 按顺序输出欧拉回路中的每条边
int ans[N];
int cnt = eulerian_path(ans, s);
printf("%d\n", cnt - 1);
for (int i = 1; i <= cnt - 1; ++i) {
    printf("%d %d\n", ans[i], ans[i + 1]);
```

无向图欧拉路

非递归，不保证字典序。

```
typedef pair<int, int> pii;
vector<pii> g[N];

void adde(int u, int v) {
    if (u != v) {
        g[u].push_back({ v, g[v].size() });
        g[v].push_back({ u, g[u].size() - 1 });
    }
    else {
        g[u].push_back({ u, g[u].size() });
    }
}

int eulerian_path(int* r, int w) {
    static int s[N], t; int m = 0; s[++t] = w;
    while (t) {
        int u = s[t];
        if (g[u].empty())
            r[++m] = u, --t;
        else {
            pii p = g[u].back();
            int v = p.first, i = p.second;
            if (v != u) {
                if (i + 1 != g[v].size()) {
                    pii& q = g[v].back();
                    g[q.first][q.second].second = i;
                    swap(g[v][i], q);
                }
                g[u].pop_back();
                g[v].pop_back();
                s[++t] = v;
            }
            else {
                g[u].pop_back();
                s[++t] = u;
            }
        }
    }
    return m;
}
//  Example: 按顺序输出欧拉回路中的每条边
int ans[N];
int cnt = eulerian_path(ans, 1);
printf("%d\n", cnt - 1);
for (int i = 1; i <= cnt - 1; ++i) {
    printf("%d %d\n", ans[i], ans[i + 1]);
```

极大团

```
#define N 80
#define ffirst(s) (s)._Find_first()
#define fnext(s, i) (s)._Find_next(i)
#define for_(i, n, s) for (int i = ffirst((s)); i < n; i = fnext((s), i))
typedef bitset<N> bs;

bs g[N];
bs msk(int n) {
    bs b;
    for (int i = 0; i != n; ++i)
        b[i] = 1;
    return b;
}

int max_clique(bs c, bs t, bs d, int n) {
    if (t.none() && d.none())
        return c.count; //  Clique c found!
    int v = -1, w = N; bs s = t | d;
    for_(u, n, s) if ((t & ~g[u]).count() < w) v = u;
    bs e = t & ~g[v];
    if (e.none()) return 0;
    int res = 0;
    for_(v, n, e) {
        bs b = 1ll << v;
        t[v] = 0;
        res = max(res, max_clique(c | b, t & g[v], d & g[v], n));
        d[v] = 1;
    }
    return res;
}
//  Example:
cout << max_clique(0, msk(n), 0) << endl;
```

最大独立集

```
#define N 80
#define ffirst(s) (s)._Find_first()
#define fnext(s, i) (s)._Find_next(i)
#define for_(i, n, s) for (int i = ffirst((s)); i < n; i = fnext((s), i))
using namespace std;

typedef bitset<N> bs;

bs g[N];

bs msk(int n) {
    bs b;
    for (int i = 0; i != n; ++i)
        b[i] = 1;
    return b;
}

bs max(bs b1, bs b2) { return b1.count() < b2.count() ? b2 : b1; }

bs mis(bs s, bs c, int n) {
    if (c.none()) return s;
    int dm = -1, v;
    for_(u, n, c) {
        int d = (g[u] & c).count() - 1;
        if (d == 0) return mis(s.set(u), c.reset(u), n);
        if (d == 1) return mis(s.set(u), c & ~g[u], n);
        if (d > dm)
            dm = d, v = u;
    }
    if (dm == 2) {
        for_(u, n, c) {
            int sz = 0;
            for (v = u; v < n; ++sz) {
                if (sz & 1) s[v] = 1;
                c[v] = 0;
                v = ffirst(g[v] & c);
            }
        }
        return s;
    }
    else return max(mis(s.set(v), c & ~g[v], n), mis(s, c.reset(v), n));
}
//  Example:
cout << mis(bs(), msk(n), n).count() << endl;
```

点覆盖计数

```
#define W 40

typedef bitset<W> bs;
bs g[W];
typedef long long ll;
ll f[1 << (W / 2)];
ll indset_cnt(int n) {
    int n1 = n / 2, n2 = n - n1;
    fill_n(f, 1 << n1, 0);
    for (int i = 0; i != (1 << n1); ++i) {
        f[i] = 1;
        for (int j = 0; j != n1; ++j)
            if (((1 << j) & i) && (g[j] & bs(i)).any())
                f[i] = 0;
    }
    for (int i = 0; i != n1; ++i)
        for (int j = 0; j != (1 << n1); ++j)
            if (j & (1 << i)) f[j] += f[j ^ (1 << i)];
    ll ans = 0;
    bs msk((1 << n1) - 1);
    for (int i = 0; i != (1 << n2); ++i) {
        bool flg = 0; bs b;
        for (int j = 0; j != n2; ++j) {
            if ((1 << j) & i) {
                if ((g[n1 + j] & (bs(i) << n1)).any()) {
                    flg = 1;
                    break;
                }
                b |= g[n1 + j];
            }
        }
        if (flg) continue;
        ans += f[(~b & msk).to_ulong()];
    }
    return ans;
}
```

k短路

```
typedef long long ll;

namespace kth_shortest_path {

const ll inf = LLONG_MAX;
typedef pair<ll, int> pli;
struct edge { int v, i; ll w; };

//  Persistent Leftist Tree
int ls[M], rs[M], dep[M], nc; pli val[M];
int gn(pli v, int q = 0) {
    int p = ++nc;
    ls[p] = ls[q]; rs[p] = rs[q];
    dep[p] = q ? dep[q] : 1;
    val[p] = q ? val[q] : v;
    return p;
}

int merge(int x, int y) {
    if (!x || !y) return x | y;
    if (val[x] > val[y]) swap(x, y);
    int z = gn({}, x); rs[z] = merge(rs[z], y);
    if (dep[ls[z]] < dep[rs[z]]) swap(ls[z], rs[z]);
    dep[z] = dep[rs[z]] + 1; return z;
}


vector<edge> g[N], gr[N]; vector<int> gt[N];
ll dis[N]; int pre[N], rt[N];

ll solve(int n, int m, int s, int t, int k,
          const vector<pair<pair<int, int>, ll>>& es) {
    for (int i = 1; i <= n; ++i) {
        g[i].resize(0);
        gr[i].resize(0);
        gt[i].resize(0);
    }
    for (int i = 0; i != es.size(); ++i) {
        pair<pair<int, int>, ll> p = es[i];
        int u = p.first.first, v = p.first.second;
        ll w = p.second;
        g[u].push_back({ v, i, w });
        gr[v].push_back({ u, i, w });
    }

    //  Dijkstra
    fill_n(dis + 1, n, inf); fill_n(pre + 1, n, -1);
    priority_queue<pli, vector<pli>, greater<pli>> pq;
    pq.push({ dis[t] = 0, t });
    while (!pq.empty()) {
        pli p = pq.top(); pq.pop();
        int u = p.second; ll du = p.first;
        if (du > dis[u]) continue;
        for (edge e : gr[u]) if (dis[e.v] > du + e.w)
```

```
            pq.push({ dis[e.v] = du + e.w, e.v }), pre[e.v] = e.i;
    }

    if (dis[s] == inf) return -1;

    //  Building heaps
    nc = 0; fill_n(rt + 1, n, 0);
    for (int u = 1; u <= n; ++u) {
        if (dis[u] == inf) continue;
        for (edge e : g[u]) {
            if (e.i == pre[u])
                gt[e.v].push_back(u);
            else if (dis[e.v] != inf)
                rt[u] = merge(rt[u], gn({ dis[e.v] + e.w - dis[u], e.v }));
        }
    }

    //  Merging heaps
    queue<int> q; q.push(t);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : gt[u])
            rt[v] = merge(rt[v], rt[u]), q.push(v);
    }

    //  Iterate k times
    if (k == 1) return dis[s]; int cnt = 0;
    if (rt[s]) pq.push({ dis[s] + val[rt[s]].first, rt[s] });
    while (!pq.empty()) {
        pli p = pq.top(); pq.pop();
        int x = p.second, u = val[x].second;
        ll res = p.first; if (++cnt == k - 1) return res;
        if (rt[u]) pq.push({ res + val[rt[u]].first, rt[u] });
        if (ls[x]) pq.push({ res - val[x].first + val[ls[x]].first, ls[x] });
        if (rs[x]) pq.push({ res - val[x].first + val[rs[x]].first, rs[x] });
    }
    return -1;
}

}   //  namespace kth_shortest_path
```

prufer序

树toprufer序

```
int deg[N]; bool del[N];
void encode_tree(int* p, int n) {
    priority_queue<int, vector<int>, greater<int>> pq;
    fill(del + 1, del + n + 1, 0);
    for (int i = 1; i <= n; ++i) {
        deg[i] = g[i].size();
        if (deg[i] == 1)
            pq.push(i);
    }
    for (int i = 1; i <= n - 2; ++i) {
        int u = pq.top(), v; pq.pop(); del[u] = 1;
        for (int w : g[u]) if (!del[w]) v = w;
        p[i] = v; if (--deg[v] == 1) pq.push(v);
    }
}
```

prufer序to树

```
int cnt[N];
void decode_tree(int* p, int n) {
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int i = 1; i <= n - 2; ++i) cnt[p[i]]++;
    for (int i = 1; i <= n; ++i) if (!cnt[i]) pq.push(i);
    for (int i = 1; i <= n - 2; ++i) {
        int u = p[i], v = pq.top(); pq.pop();
        g[u].push_back(v);
        g[v].push_back(u);
        if (!--cnt[u]) pq.push(u);
    }
    int u = pq.top(); pq.pop();
    int v = pq.top(); pq.pop();
    g[u].push_back(v);
    g[v].push_back(u);
}
```

树的性质

直径

```
const int N = 10000 + 10;

int n, c, d[N];
vector<int> E[N];

void dfs(int u, int fa) {
  for (int v : E[u]) {
    if (v == fa) continue;
    d[v] = d[u] + 1;
    if (d[v] > d[c]) c = v;
    dfs(v, u);
  }
}

int main() {
  scanf("%d", &n);
  for (int i = 1; i < n; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    E[u].push_back(v), E[v].push_back(u);
  }
  dfs(1, 0);
  d[c] = 0, dfs(c, 0);
  printf("%d\n", d[c]);
  return 0;
}//dfs

const int N = 10000 + 10;

int n, d = 0;
int d1[N], d2[N];
vector<int> E[N];

void dfs(int u, int fa) {
  d1[u] = d2[u] = 0;
  for (int v : E[u]) {
    if (v == fa) continue;
    dfs(v, u);
    int t = d1[v] + 1;
    if (t > d1[u])
      d2[u] = d1[u], d1[u] = t;
    else if (t > d2[u])
      d2[u] = t;
  }
  d = max(d, d1[u] + d2[u]);
}

int main() {
  scanf("%d", &n);
  for (int i = 1; i < n; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    E[u].push_back(v), E[v].push_back(u);
  }
  dfs(1, 0);
  printf("%d\n", d);
  return 0;
}//dp
```

重心

```
// 这份代码默认节点编号从 1 开始，即 i ∈ [1,n]
int size[MAXN],  // 这个节点的“大小”（所有子树上节点数 + 该节点）
    weight[MAXN],  // 这个节点的“重量”
    centroid[2];   // 用于记录树的重心（存的是节点编号）
void GetCentroid(int cur, int fa) {  // cur 表示当前节点 (current)
  size[cur] = 1;
  weight[cur] = 0;
  for (int i = head[cur]; i != -1; i = e[i].nxt) {
    if (e[i].to != fa) {  // e[i].to 表示这条有向边所通向的节点。
      GetCentroid(e[i].to, cur);
      size[cur] += size[e[i].to];
      weight[cur] = max(weight[cur], size[e[i].to]);
    }
  }
  weight[cur] = max(weight[cur], n - size[cur]);
  if (weight[cur] <= n / 2) {  // 依照树的重心的定义统计
    centroid[centroid[0] != 0] = cur;
  }
}
```

dsu on tree

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

const int N = 2e5 + 5;

int n;

// g[u]: 存储与 u 相邻的结点
vector<int> g[N];

// sz: 子树大小
// big: 重儿子
// col: 结点颜色
// L[u]: 结点 u 的 DFS 序
// R[u]: 结点 u 子树中结点的 DFS 序的最大值
// Node[i]: DFS 序为 i 的结点
// ans: 存答案
// cnt[i]: 颜色为 i 的结点个数
// totColor: 目前出现过的颜色个数
int sz[N], big[N], col[N], L[N], R[N], Node[N], totdfn;
int ans[N], cnt[N], totColor;

void add(int u) {
  if (cnt[col[u]] == 0) ++totColor;
  cnt[col[u]]++;
}
void del(int u) {
  cnt[col[u]]--;
  if (cnt[col[u]] == 0) --totColor;
}
int getAns() { return totColor; }

void dfs0(int u, int p) {
  L[u] = ++totdfn;
  Node[totdfn] = u;
  sz[u] = 1;
  for (int v : g[u])
    if (v != p) {
      dfs0(v, u);
      sz[u] += sz[v];
      if (!big[u] || sz[big[u]] < sz[v]) big[u] = v;
    }
  R[u] = totdfn;
}

void dfs1(int u, int p, bool keep) {
  // 计算轻儿子的答案
  for (int v : g[u])
    if (v != p && v != big[u]) {
      dfs1(v, u, false);
    }
  // 计算重儿子答案并保留计算过程中的数据（用于继承）
  if (big[u]) {
    dfs1(big[u], u, true);
  }
  for (int v : g[u])
    if (v != p && v != big[u]) {
      // 子树结点的 DFS 序构成一段连续区间，可以直接遍历
      for (int i = L[v]; i <= R[v]; i++) {
        add(Node[i]);
      }
    }
  add(u);
  ans[u] = getAns();
  if (keep == false) {
    for (int i = L[u]; i <= R[u]; i++) {
      del(Node[i]);
    }
  }
}

int main() {
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) scanf("%d", &col[i]);
  for (int i = 1; i < n; i++) {
    int u, v;
    scanf("%d%d", &u, &v);
    g[u].push_back(v);
    g[v].push_back(u);
  }
  dfs0(1, 0);
  dfs1(1, 0, false);
  for (int i = 1; i <= n; i++) printf("%d%c", ans[i], " \n"[i == n]);
  return 0;
}
```

虚树

```
inline bool cmp(const int x, const int y) { return id[x] < id[y]; }

void build() {
  sort(h + 1, h + k + 1, cmp);
  sta[top = 1] = 1, g.sz = 0, g.head[1] = -1;
  // 1 号节点入栈，清空 1 号节点对应的邻接表，设置邻接表边数为 1
  for (int i = 1, l; i <= k; ++i)
    if (h[i] != 1) {
      //如果 1 号节点是关键节点就不要重复添加
      l = lca(h[i], sta[top]);
      //计算当前节点与栈顶节点的 LCA
      if (l != sta[top]) {
        //如果 LCA 和栈顶元素不同，则说明当前节点不再当前栈所存的链上
        while (id[l] < id[sta[top - 1]])
          //当次大节点的 Dfs 序大于 LCA 的 Dfs 序
          g.push(sta[top - 1], sta[top]), top--;
        //把与当前节点所在的链不重合的链连接掉并且弹出
        if (id[l] > id[sta[top - 1]])
          //如果 LCA 不等于次大节点（这里的大于其实和不等于没有区别）
          g.head[l] = -1, g.push(l, sta[top]), sta[top] = l;
        //说明 LCA 是第一次入栈，清空其邻接表，连边后弹出栈顶元素，并将 LCA 入栈
        else
          g.push(l, sta[top--]);
        //说明 LCA 就是次大节点，直接弹出栈顶元素
      }
      g.head[h[i]] = -1, sta[++top] = h[i];
      //当前节点必然是第一次入栈，清空邻接表并入栈
    }
  for (int i = 1; i < top; ++i)
    g.push(sta[i], sta[i + 1]);  //剩余的最后一条链连接一下
  return;
}
```

斯坦纳树

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 510;
const int INF = 0x3f3f3f3f;
typedef pair<int, int> P;
int n, m, k;

struct edge {
  int to, next, w;
} e[maxn << 1];

int head[maxn << 1], tree[maxn << 1], tot;
int dp[maxn][5000], vis[maxn];
int key[maxn];
priority_queue<P, vector<P>, greater<P> > q;

void add(int u, int v, int w) {
  e[++tot] = edge{v, head[u], w};
  head[u] = tot;
}

void dijkstra(int s) {  // 求解最短路
  memset(vis, 0, sizeof(vis));
  while (!q.empty()) {
    P item = q.top();
    q.pop();
    if (vis[item.second]) continue;
    vis[item.second] = 1;
    for (int i = head[item.second]; i; i = e[i].next) {
      if (dp[tree[i]][s] > dp[item.second][s] + e[i].w) {
        dp[tree[i]][s] = dp[item.second][s] + e[i].w;
        q.push(P(dp[tree[i]][s], tree[i]));
      }
    }
  }
}

int main() {
  memset(dp, INF, sizeof(dp));
  scanf("%d %d %d", &n, &m, &k);
  int u, v, w;
  for (int i = 1; i <= m; i++) {
    scanf("%d %d %d", &u, &v, &w);
    add(u, v, w);
    tree[tot] = v;
    add(v, u, w);
    tree[tot] = u;
  }
  for (int i = 1; i <= k; i++) {
    scanf("%d", &key[i]);
    dp[key[i]][1 << (i - 1)] = 0;
  }
  for (int s = 1; s < (1 << k); s++) {
    for (int i = 1; i <= n; i++) {
      for (int subs = s & (s - 1); subs;
           subs = s & (subs - 1))  // 状压 dp 可以看下题解里写的比较详细
        dp[i][s] = min(dp[i][s], dp[i][subs] + dp[i][s ^ subs]);
      if (dp[i][s] != INF) q.push(P(dp[i][s], i));
    }
    dijkstra(s);
  }
  printf("%d\n", dp[key[1]][(1 << k) - 1]);
  return 0;
}
```

DMST

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 502;
typedef long long ll;
typedef pair<int, int> pii;
ll d[MAXN][MAXN], dd[MAXN][MAXN], rk[MAXN][MAXN], val[MAXN];
const ll INF = 1e17;
int n, m;
bool cmp(int a, int b) { return val[a] < val[b]; }
void floyd() {
  for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++) d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
struct node {
  ll u, v, w;
} a[MAXN * (MAXN - 1) / 2];
void solve() {
  //求图的绝对中心
  floyd();
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
      rk[i][j] = j;
      val[j] = d[i][j];
    }
    sort(rk[i] + 1, rk[i] + 1 + n, cmp);
  }
  ll P = 0, ansP = INF;
  //在点上
  for (int i = 1; i <= n; i++) {
    if (d[i][rk[i][n]] * 2 < ansP) {
      ansP = d[i][rk[i][n]] * 2;
      P = i;
    }
  }
  //在边上
  int f1 = 0, f2 = 0;
  ll disu = INT_MIN, disv = INT_MIN, ansL = INF;
  for (int i = 1; i <= m; i++) {
    ll u = a[i].u, v = a[i].v, w = a[i].w;
    for (int p = n, i = n - 1; i >= 1; i--) {
      if (d[v][rk[u][i]] > d[v][rk[u][p]]) {
        if (d[u][rk[u][i]] + d[v][rk[u][p]] + w < ansL) {
          ansL = d[u][rk[u][i]] + d[v][rk[u][p]] + w;
          f1 = u, f2 = v;
          disu = (d[u][rk[u][i]] + d[v][rk[u][p]] + w) / 2 - d[u][rk[u][i]];
          disv = w - disu;
        }
        p = i;
      }
    }
  }
  cout << min(ansP, ansL) / 2 << '\n';
  //最小路径生成树
```

```cpp
  vector<pii> pp;
  for (int i = 1; i <= 501; ++i)
    for (int j = 1; j <= 501; ++j) dd[i][j] = INF;
  for (int i = 1; i <= 501; ++i) dd[i][i] = 0;
  if (ansP <= ansL) {
    for (int j = 1; j <= n; j++) {
      for (int i = 1; i <= m; ++i) {
        ll u = a[i].u, v = a[i].v, w = a[i].w;
        if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
          dd[P][v] = dd[P][u] + w;
          pp.push_back({u, v});
        }
        u = a[i].v, v = a[i].u, w = a[i].w;
        if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
          dd[P][v] = dd[P][u] + w;
          pp.push_back({u, v});
        }
      }
    }
    for (auto [x, y] : pp) cout << x << ' ' << y << '\n';
  } else {
    d[n + 1][f1] = disu;
    d[f1][n + 1] = disu;
    d[n + 1][f2] = disv;
    d[f2][n + 1] = disv;
    a[m + 1].u = n + 1, a[m + 1].v = f1, a[m + 1].w = disu;
    a[m + 2].u = n + 1, a[m + 2].v = f2, a[m + 2].w = disv;
    n += 1;
    m += 2;
    floyd();
    P = n;
    for (int j = 1; j <= n; j++) {
      for (int i = 1; i <= m; ++i) {
        ll u = a[i].u, v = a[i].v, w = a[i].w;
        if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
          dd[P][v] = dd[P][u] + w;
          pp.push_back({u, v});
        }
        u = a[i].v, v = a[i].u, w = a[i].w;
        if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {
          dd[P][v] = dd[P][u] + w;
          pp.push_back({u, v});
        }
      }
    }
    cout << f1 << ' ' << f2 << '\n';
    for (auto [x, y] : pp)
      if (x != n && y != n) cout << x << ' ' << y << '\n';
  }
}
void init() {
  for (int i = 1; i <= 501; ++i)
    for (int j = 1; j <= 501; ++j) d[i][j] = INF;
  for (int i = 1; i <= 501; ++i) d[i][i] = 0;
}
int main() {
  init();
  cin >> n >> m;
  for (int i = 1; i <= m; ++i) {
    ll u, v, w;
    cin >> u >> v >> w;
    w *= 2;
    d[u][v] = w, d[v][u] = w;
    a[i].u = u, a[i].v = v, a[i].w = w;
  }
  solve();
  return 0;
}
```

圆方树

```
#include <algorithm>
#include <cstdio>
#include <vector>

const int MN = 100005;

int N, M, cnt;
std::vector<int> G[MN], T[MN * 2];

int dfn[MN], low[MN], dfc;
int stk[MN], tp;

void Tarjan(int u) {
  printf("  Enter : #%d\n", u);
  low[u] = dfn[u] = ++dfc;                // low 初始化为当前节点 dfn
  stk[++tp] = u;                          // 加入栈中
  for (int v : G[u]) {                    // 遍历 u 的相邻节点
    if (!dfn[v]) {                        // 如果未访问过
      Tarjan(v);                          // 递归
      low[u] = std::min(low[u], low[v]);  // 未访问的和 low 取 min
      if (low[v] == dfn[u]) {  // 标志着找到一个以 u 为根的点双连通分量
        ++cnt;                 // 增加方点个数
        printf("  Found a New BCC #%d.\n", cnt - N);
        // 将点双中除了 u 的点退栈，并在圆方树中连边
        for (int x = 0; x != v; --tp) {
          x = stk[tp];
          T[cnt].push_back(x);
          T[x].push_back(cnt);
          printf("    BCC #%d has vertex #%d\n", cnt - N, x);
        }
        // 注意 u 自身也要连边（但不退栈）
        T[cnt].push_back(u);
        T[u].push_back(cnt);
        printf("    BCC #%d has vertex #%d\n", cnt - N, u);
      }
    } else
      low[u] = std::min(low[u], dfn[v]);  // 已访问的和 dfn 取 min
  }
  printf("  Exit : #%d : low = %d\n", u, low[u]);
  printf("  Stack:\n    ");
  for (int i = 1; i <= tp; ++i) printf("%d, ", stk[i]);
  puts("");
}

int main() {
  scanf("%d%d", &N, &M);
  cnt = N;  // 点双 / 方点标号从 N 开始
  for (int i = 1; i <= M; ++i) {
    int u, v;
    scanf("%d%d", &u, &v);
    G[u].push_back(v);  // 加双向边
    G[v].push_back(u);
  }
  // 处理非连通图
  for (int u = 1; u <= N; ++u)
    if (!dfn[u]) Tarjan(u), --tp;
  // 注意到退出 Tarjan 时栈中还有一个元素即根，将其退栈
  return 0;
}
```