支配树

算法简介

给定一个有向图，定义一个超级源点，连向所有入度为 0 的点。

对于点对 (u,v)，如果删除 u 将导致超级源点不可以到达 v，则称 u 支配 v。

易知支配关系构成树，其中每个子树的根节点支配子树的所有结点。称这样的树为支配树。

算法模板

有向无环图

洛谷p2597

对每个结点 v，设原图中有 $u_1,u_2,u_3\cdots u_k\to v$，易知 v 在支配树上的父结点为 $u_1,u_2\cdots u_k$ 在支配树上的 LCA。

考虑对原图的点进行拓扑，边拓扑边建立支配树，动态维护每个点 u 的当前父结点 $p(u)$。

每次拓扑到 u 时直接在支配树上连边 $p(u)\to u$，然后动态更新原图中 $u\to v$ 的每个 $p(v)\to \text{LCA}(p(v),u)$。

注意 $p(u)$ 初值为 0 且 $\text{LCA}(p(u),0)=p(u)$，至于 LCA 可以考虑用倍增维护。时间复杂度 $O(m\log n)$。

```
const int MAXN=1e5+5,MAXM=1e6+5,MAXV=18;
struct Edge{
    int to,next;
}edge[MAXM+MAXN];
int head1[MAXN],head2[MAXN],edge_cnt;
int deg[MAXN],f[MAXN],dep[MAXN],anc[MAXN][MAXV],lg2[MAXN];
void Insert1(int u,int v){
    edge[++edge_cnt]=Edge{v,head1[u]};
    head1[u]=edge_cnt;
    deg[v]++;
}
void Insert2(int u,int v){
    edge[++edge_cnt]=Edge{v,head2[u]};
    head2[u]=edge_cnt;
}
int LCA(int u,int v){
    if(dep[u]<dep[v])
    swap(u,v);
    while(dep[u]>dep[v])u=anc[u][lg2[dep[u]-dep[v]]];
    if(u==v)
```

```
    return u;
    for(int i=MAXV-1;i>=0;i--){
        if(anc[u][i]!=anc[v][i])
        u=anc[u][i],v=anc[v][i];
    }
    return anc[u][0];
}
int build(int n){
    lg2[1]=0;
    _for(i,2,MAXN)lg2[i]=lg2[i>>1]+1;
    int rt=n+1;
    queue<int> q;
    _rep(i,1,n){
        if(deg[i]==0){
            f[i]=rt;
            q.push(i);
        }
    }
    while(!q.empty()){
        int u=q.front();q.pop();
        dep[u]=dep[f[u]]+1;
        Insert2(f[u],u);
        anc[u][0]=f[u];
        for(int i=1;i<MAXV;i++)
        anc[u][i]=anc[anc[u][i-1]][i-1];
        for(int i=head1[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(f[v]==0)
            f[v]=u;
            else
            f[v]=LCA(u,f[v]);
            deg[v]--;
            if(deg[v]==0)
            q.push(v);
        }
    }
    return rt;
}
int sz[MAXN];
void dfs(int u){
    sz[u]=1;
    for(int i=head2[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs(v);
        sz[u]+=sz[v];
    }
}
int main(){
    int n=read_int();
    _rep(u,1,n){
```

```
        int v=read_int();
        while(v){
            Insert1(v,u);
            v=read_int();
        }
    }
    int rt=build(n);
    dfs(rt);
    _rep(i,1,n)
    enter(sz[i]-1);
    return 0;
}
```

一般有向图

~~挖坑待填~~

算法例题

例题一

gym 101741 L

题意

给定一个无向连通图，定义源点为 1 号点。对每条边，询问删除该边会导致源点到多少个点的最短路改变。

首先跑最短路，然后保留所有在最短路树上的边，同时规定每条边方向由距离近的点指向距离远的点，易知构成有向无环图。

问题转化为求有向无环图的支配边。

对任意一个点，如果该点有至少两条入边，易知所有入边支配点集均为空，否则该边的支配点集等价于该点的支配子树。

```
const int MAXN=2e5+5,MAXM=2e5+5,MAXV=22;
namespace Tree{
    struct Edge{
        int to,id,next;
    }edge[MAXN+MAXM];
    int head1[MAXN],head2[MAXN],edge_cnt;
    int deg[MAXN],f[MAXN],dep[MAXN],anc[MAXN][MAXV],lg2[MAXN];
    int deg0[MAXN];
    void Insert1(int u,int v,int id){
        edge[++edge_cnt]=Edge{v,id,head1[u]};
        head1[u]=edge_cnt;
```

```
        deg[v]++;
        deg0[v]++;
    }
    void Insert2(int u,int v){
        edge[++edge_cnt]=Edge{v,0,head2[u]};
        head2[u]=edge_cnt;
    }
    int LCA(int u,int v){
        if(dep[u]<dep[v])
        swap(u,v);
        while(dep[u]>dep[v])u=anc[u][lg2[dep[u]-dep[v]]];
        if(u==v)
        return u;
        for(int i=MAXV-1;i>=0;i--){
            if(anc[u][i]!=anc[v][i])
            u=anc[u][i],v=anc[v][i];
        }
        return anc[u][0];
    }
    int build(int n){
        lg2[1]=0;
        _for(i,2,MAXN)lg2[i]=lg2[i>>1]+1;
        int rt=n+1;
        queue<int> q;
        _rep(i,1,n){
            if(deg[i]==0){
                f[i]=rt;
                q.push(i);
            }
        }
        while(!q.empty()){
            int u=q.front();q.pop();
            dep[u]=dep[f[u]]+1;
            Insert2(f[u],u);
            anc[u][0]=f[u];
            for(int i=1;i<MAXV;i++)
            anc[u][i]=anc[anc[u][i-1]][i-1];
            for(int i=head1[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(f[v]==0)
                f[v]=u;
                else
                f[v]=LCA(u,f[v]);
                deg[v]--;
                if(deg[v]==0)
                q.push(v);
            }
        }
        return rt;
    }
```

```
    int sz[MAXN],ans[MAXM];
    void dfs(int u){
        sz[u]=1;
        for(int i=head2[u];i;i=edge[i].next){
            int v=edge[i].to;
            dfs(v);
            sz[u]+=sz[v];
        }
    }
    void solve(int n,int m){
        int rt=build(n);
        dfs(rt);
        _rep(u,1,n){
            for(int i=head1[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(deg0[v]==1)
                ans[edge[i].id]=sz[v];
            }
        }
        _for(i,0,m)
        enter(ans[i]);
    }
}
struct Edge{
    int to,w,id,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w,int id){
    edge[++edge_cnt]=Edge{v,w,id,head[u]};
    head[u]=edge_cnt;
}
LL dis[MAXN];
bool vis[MAXN];
int main(){
    int n=read_int(),m=read_int();
    _for(i,0,m){
        int u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w,i);
        Insert(v,u,w,i);
    }
    priority_queue<pair<LL,int> >q;
    mem(dis,127);
    dis[1]=0;
    q.push(make_pair(-dis[1],1));
    while(!q.empty()){
        int u=q.top().second;
        q.pop();
        if(vis[u])continue;
        vis[u]=true;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
```

```
            if(dis[v]>dis[u]+edge[i].w){
                dis[v]=dis[u]+edge[i].w;
                q.push(make_pair(-dis[v],v));
            }
        }
    }
    _rep(u,1,n){
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(dis[v]==dis[u]+edge[i].w)
            Tree::Insert1(u,v,edge[i].id);
        }
    }
    Tree::solve(n,m);
    return 0;
}
```