虚树

算法简介

一种用于加速树上 dp 算法，时间复杂度 $O(k\log k)$ 其中 k 为树上的关键节点数。

算法思想

树上关键点数量较少时很多节点不需要进行 dp 考虑重建一棵树，只保留必要的节点。

发现所有关键节点的 LCA 必须保留，但如果暴力枚举每个关键节点的 LCA 并考虑是否保留时间复杂度至少为 $O(k^2)$

事实上只需要将关键点序列dfs 序排序，然后枚举序列中相邻节点的 LCA

可以证明该方法枚举得到的 LCA 不会有遗漏。假设 $p=\text{LCA}(u,v)$ 且 u,v 不是序列中相邻的节点。

于是必有 u,v 属于 p 的不同子树，考虑取序列中与 v 相邻且 dfs 序小于 v 的节点，记为 v_2

首先易知 v_2 也位于 p 的子树。于是如果 v_2 与 v 不在同一棵子树，那么 $\text{LCA}(v_2,v)=p$

否则把 v 换成 v_2 继续重复上述操作，最后总有 v_{k-1},v_k 不在同一棵子树(最差的情况是 $v_k=u$) 证毕。

然后为了保证最终选取的点一定会构成树而不是森林，考虑强制将根节点加入虚树或者构造一个虚根。

接下来是建边过程，考虑用单调栈维护从根节点出发的一条树链。

每新加入一个节点时，先计算新节点与栈顶节点(事实上栈顶节点一定是上一次加入的节点，即与新元素相邻的节点)的 LCA 记为 p

如果 p 恰好为栈顶节点，则直接将新节点入栈。

否则，将栈顶节点弹出直到栈顶栈顶节点的下一个节点的 dfs 序不大于 p 的dfs 序。

注意到每当栈顶节点被弹出时他在虚树中的位置已经确定，可以直接将他与下一个栈顶节点连一条边。

然后如果 p 恰为栈顶栈顶节点的下一个节点，则将栈顶节点弹出并将他与下一个栈顶节点连一条边。

否则将栈顶节点弹出并将他与 p 连一条边，再将 p 加入栈。这一系列操作结束后再将新节点入栈。

所有关键节点都访问结束后将栈的剩余元素出栈并连边。

算法模板

洛谷p2495

题意

给定一棵以 1 为根的边权树，设切断一条边的费用为这条边的边权。接下来 q 次询问。

每次询问给定 k_i 个关键节点(保证不含根节点)，要求切断若干条边使得所有关键节点节点与根节点不连通，问该操作的最小费用为多少。

题解

首先考虑 dp 过程，有状态转移方程

\begin{equation}v \text{ 是 }u \text{的儿子节点且} v \text{ 不是关键节点}\text{dp}(u)\gets\min(\text{dp}(v),w(u,v))\end{equation} \begin{equation}v \text{ 是 }u \text{的儿子节点且} v \text{ 是关键节点}\text{dp}(u)\gets w(u,v)\end{equation}

接下来建立虚树，并令 $w(u,v)$ 为 $u\to v$ 路径上的最短边。事实上令 $w(u,v)$ 为 $1\to v$ 的最短边不影响最终答案。

最后暴力 dp 即可，时间复杂度 $O(\sum_{i=1}^q k_i\log k_i)$注意每次新建树时不能暴力清零 head 数组，需要在建树过程中清零。

```
const int MAXN=3e5+5;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int d[MAXN],dis[MAXN],sz[MAXN],f[MAXN],dfn[MAXN],dfs_t;
int h_son[MAXN],mson[MAXN],p[MAXN];
void dfs_1(int u,int fa,int depth,int Dis){
    sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;dfn[u]=++dfs_t;dis[u]=Dis;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        dfs_1(v,u,depth+1,min(Dis,edge[i].w));
        sz[u]+=sz[v];
        if(sz[v]>mson[u])
        h_son[u]=v,mson[u]=sz[v];
    }
}
void dfs_2(int u,int top){
```

```
    p[u]=top;
    if(mson[u])dfs_2(h_son[u],top);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==f[u]||v==h_son[u])
        continue;
        dfs_2(v,v);
    }
}
int LCA(int u,int v){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])swap(u,v);
        u=f[p[u]];
    }
    return d[u]<d[v]?u:v;
}
struct cmp{
    bool operator () (const int a,const int b)const{
        return dfn[a]<dfn[b];
    }
};
int Stack[MAXN],top,node[MAXN];
void build(int k){
    edge_cnt=0;
    sort(node,node+k,cmp());
    Stack[top=1]=1;head[1]=0;
    _for(i,0,k){
        int p=LCA(Stack[top],node[i]);
        if(p!=Stack[top]){
while(dfn[p]<dfn[Stack[top-1]])Insert(Stack[top-1],Stack[top],dis[Stack[top
]]),top--;
            if(p!=Stack[top-1])
            head[p]=0,Insert(p,Stack[top],dis[Stack[top]]),Stack[top]=p;
            else
            Insert(Stack[top-1],Stack[top],dis[Stack[top]]),top--;
        }
        head[node[i]]=0,Stack[++top]=node[i];
    }
    while(top>1)Insert(Stack[top-1],Stack[top],dis[Stack[top]]),top--;
}
bool is_key[MAXN];
LL dp(int u){
    LL ans=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(is_key[v])
        ans+=edge[i].w;
        else
        ans+=min(dp(v),1LL*edge[i].w);
    }
    return ans;
```

```
}
int main()
{
    int n=read_int(),root=1,u,v,w;
    _for(i,1,n){
        u=read_int(),v=read_int(),w=read_int();
        Insert(u,v,w);Insert(v,u,w);
    }
    dfs_1(root,0,0,1e9);
    dfs_2(root,root);
    int q=read_int();
    while(q--){
        int k=read_int();
        _for(i,0,k){
            node[i]=read_int();
            is_key[node[i]]=true;
        }
        build(k);
        enter(dp(root));
        _for(i,0,k)
        is_key[node[i]]=false;
    }
    return 0;
}
```

算法练习

洛谷p3233

习题一

题意

给定一棵 n 个节点的无根树，q 次询问。

每次询问给定 k_i 个关键点。树上每个节点属于离它最近的关键点(如果有多个距离最近点则选择编号最小的)。

每次询问要求输出 k_i 个数，代表属于每个给定关键点的节点数。

题解

强制以 1 为根，转化为有根树。先考虑暴力 dp 的做法。

在原树上，先一次 dfs 找到每个节点子树方向上距离最小的关键节点，同时记录最小节点的距离和编号。

然后再一次 dfs 更新每个节点祖先方向的距离最小的关键节点，即可得到答案。

暴力 dp 对每次询问时间复杂度 $O(n)$，接下来考虑虚树优化 dp。

对虚树上的点，直接使用上述暴力 dp 即可。关键在于如何计算不在虚树上的点对关键节点答案的贡献。

对于不在虚树上的点，要么位于虚树上的某个节点在原树上的儿子节点的子树中且该子树中没有关键节点，记为第一类点。

要么位于虚树上的某两个节点在原树上的路径上的某个节点的非路径方向的子树中，记为第二类点。

先考虑第一类点的贡献，发现第一类点可以全部计入它的第一个属于虚树的祖先节点的贡献中。

对于第二类点的贡献。取虚树上两相邻点在原树上的路径，可以根据到哪个关键节点近将路径分成两段。

每段路径上的节点及其非路径方向的子树对距离该节点近的关键节点做贡献。

具体实现过程与上述讨论存在少量差异，详细见代码。

```
const int MAXN=3e5+5,MAXM=20;
struct Edge{
    int to,w,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w=1){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
int d[MAXN],anc[MAXN][MAXM],lg2[MAXN],sz[MAXN],dfn[MAXN],dfs_t;
void dfs(int u,int fa){
    anc[u][0]=fa,d[u]=d[fa]+1,sz[u]=1,dfn[u]=++dfs_t;
    for(int i=1;(1<<i)<d[u];i++)
    anc[u][i]=anc[anc[u][i-1]][i-1];
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)
        continue;
        dfs(v,u);
        sz[u]+=sz[v];
    }
}
void init(int root){
    lg2[0]=-1;
    _for(i,1,MAXN)
    lg2[i]=lg2[i>>1]+1;
    d[root]=1;
    dfs(root,0);
}
int LCA(int p,int q){
    if(d[p]<d[q])
    swap(p,q);
```

```
    while(d[p]>d[q])p=anc[p][lg2[d[p]-d[q]]];
    if(p==q)
    return p;
    for(int i=lg2[d[p]-1];i>=0;i--){
        if(anc[p][i]!=anc[q][i])
        p=anc[p][i],q=anc[q][i];
    }
    return anc[p][0];
}
struct cmp{
    bool operator () (const int a,const int b)const{
        return dfn[a]<dfn[b];
    }
};
int Stack[MAXN],top,node[MAXN],temp[MAXN];
void build(int k){
    edge_cnt=0;
    sort(node,node+k,cmp());
    Stack[top=1]=1;head[1]=0;
    _for(i,0,k){
        if(node[i]==1)continue;
        int p=LCA(Stack[top],node[i]);
        if(p!=Stack[top]){
while(dfn[p]<dfn[Stack[top-1]])Insert(Stack[top-1],Stack[top],d[Stack[top]]
-d[Stack[top-1]]),top--;
            if(p!=Stack[top-1])
            head[p]=0,Insert(p,Stack[top],d[Stack[top]]-d[p]),Stack[top]=p;
            else
            Insert(Stack[top-1],Stack[top],d[Stack[top]]-
d[Stack[top-1]]),top--;
        }
        head[node[i]]=0,Stack[++top]=node[i];
    }
    while(top>1)Insert(Stack[top-1],Stack[top],d[Stack[top]]-
d[Stack[top-1]]),top--;
}
bool is_key[MAXN];
int min_dis[MAXN],min_node[MAXN],ans[MAXN];
void dfs_2(int u){
    if(is_key[u])min_dis[u]=0,min_node[u]=u;
    else min_dis[u]=MAXN+1,min_node[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs_2(v);
        if(min_dis[u]>min_dis[v]+edge[i].w)
        min_dis[u]=min_dis[v]+edge[i].w,min_node[u]=min_node[v];
        else
if((min_dis[u]==min_dis[v]+edge[i].w)&&min_node[u]>min_node[v])
        min_node[u]=min_node[v];
    }
```

```
}
void cal(int p,int q){
    int son=q,cut=q;
    while(d[son]>d[p]+1)son=anc[son][lg2[d[son]-d[p]-1]];
    ans[min_node[p]]-=sz[son];
    for(int i=lg2[d[q]-d[p]];i>=0;i--){
        if(d[anc[cut][i]]<d[p])continue;
        int dis1=d[anc[cut][i]]-d[p]+min_dis[p],dis2=d[q]-
d[anc[cut][i]]+min_dis[q];
        if(dis1>dis2||(dis1==dis2&&min_node[p]>min_node[q]))
        cut=anc[cut][i];
    }
    ans[min_node[q]]+=sz[cut]-sz[q];ans[min_node[p]]+=sz[son]-sz[cut];
}
void dfs_3(int u){
    ans[min_node[u]]+=sz[u];
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(min_dis[v]>min_dis[u]+edge[i].w)
        min_dis[v]=min_dis[u]+edge[i].w,min_node[v]=min_node[u];
        else
if((min_dis[v]==min_dis[u]+edge[i].w)&&min_node[v]>min_node[u])
        min_node[v]=min_node[u];
        cal(u,v);
        dfs_3(v);
    }
}
int main()
{
    int n=read_int(),root=1,u,v;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);Insert(v,u);
    }
    init(root);
    int q=read_int();
    while(q--){
        int k=read_int();
        _for(i,0,k){
            node[i]=temp[i]=read_int();
            is_key[node[i]]=true;ans[node[i]]=0;
        }
        build(k);
        dfs_2(root);
        dfs_3(root);
        _for(i,0,k){
            space(ans[temp[i]]);
            is_key[node[i]]=false;
        }
        puts("");
    }
```

```
    return 0;
}
```