

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 个节点(保证不含根节点)，问使得这 k_i 个节点与根节点不连通的最小费用和为多少。

题解

首先考虑 dp 过程，有状态转移方程

$$\begin{aligned} \text{dp}(u) &= \min(\text{dp}(v), w(u,v)) \quad \text{if } u \text{ is not a key node} \\ \text{dp}(u) &= w(u,v) \quad \text{if } u \text{ is a key node} \end{aligned}$$

接下来建立虚树，并令 $w(u,v)$ 为 u 到 v 路径上的最短边。事实上令 $w(u,v)$ 为 1 到 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;  
}
```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E8%99%9A%E6%A0%91&rev=1595766174 

Last update: 2020/07/26 20:22