树上启发式合并

算法简介

可离线处理子树相关信息统计的算法，时间复杂度 $O(n\log n)$空间复杂度 $O(n)$

算法思想

首先发现对每棵子树进行信息统计时，只需要合并所有儿子节点所在子树信息，最后加上该节点本身信息即可。

考虑信息合并的过程，类比并查集的启发式合并，发现把节点数少的子树信息合并到节点数多的子树将得到更优复杂度。

于是考虑保留重儿子子树的统计信息，然后暴力遍历轻儿子子树，合并结果。

具体实现为先 dfs 处理轻儿子子树的询问，再删除轻儿子子树的统计信息(否则空间复杂度难以承受)。

然后处理重儿子子树的询问，但保留重儿子子树的统计信息，最后再次遍历轻儿子子树，然后把信息暴力合并。

关于时间复杂度，发现某个节点每被额外统计一次当且仅当他每有一个祖先节点为轻儿子。

由于每个节点到根节点最多有 $O(\log n)$ 条轻边，于是每个节点最多被统计 $O(\log n)$ 次，于是总时间复杂度为 $O(n\log n)$

算法模板

CF570D

题意

给定一个以 1 为根的 n 个节点的树，每个点上有一个小写英文字母。每个点的深度定义为该节点到 1 号节点路径上的点数。

每次询问 a,b查询以 a 为根的子树内深度为 b 的节点上的字母重新排列之后是否能构成回文串。

题解

考虑字符串排序后能构成回文串的条件，发现只要出现字符串中出现奇数次的字符不超过一种就行。

于是只需要维护每个节点子树的每个深度的每种字符个数就行了。

```
const int MAXN=5e5+5;
```

```
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
char buf[MAXN];
int cnt[MAXN][30];
bool ans[MAXN];
vector<pair<int,int> > query[MAXN];
bool check(int d){
    int temp=0;
    _for(i,0,26)temp+=cnt[d][i]&1;
    return temp<=1;
}
int d[MAXN],sz[MAXN],dfs_id[MAXN],node_id[MAXN],dfs_t;
int h_son[MAXN],mson[MAXN];
void dfs_1(int u,int depth){
    sz[u]=1;d[u]=depth;dfs_id[u]=++dfs_t;node_id[dfs_t]=u;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs_1(v,depth+1);
        sz[u]+=sz[v];
        if(sz[v]>mson[u]){
            h_son[u]=v;
            mson[u]=sz[v];
        }
    }
}
void update_node(int u,int v){cnt[d[u]][buf[u]-'a']+=v;}
void update_tree(int u,int v){
    _for(i,0,sz[u])
    update_node(node_id[dfs_id[u]+i],v);
}
void dfs_2(int u,bool del){
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==h_son[u])continue;
        dfs_2(v,true);
    }
    if(h_son[u])dfs_2(h_son[u],false);
    update_node(u,1);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==h_son[u])continue;
        update_tree(v,1);
    }
```

```
    _for(i,0,query[u].size())
    ans[query[u][i].first]=check(query[u][i].second);
    if(del)update_tree(u,-1);
}
int main()
{
    int n=read_int(),q=read_int(),root=1,u,d;
    _rep(i,2,n)
    Insert(read_int(),i);
    scanf("%s",buf+1);
    _for(i,0,q){
        u=read_int(),d=read_int();
        query[u].push_back(make_pair(i,d));
    }
    dfs_1(root,root);
    dfs_2(root,false);
    _for(i,0,q){
        if(ans[i])
        puts("Yes");
        else
        puts("No");
    }
    return 0;
}
```