LCA

考虑让一个非叶结点成为一个**公共祖先**，需要满足他至少 2 个儿子的子树中有已经被选定的 **叶子节点** 那么，我们在选的时候可以考虑**贪心**处理，一个结点只选他的两个儿子，而两个儿子还需要继续贪心，选择他价值最大的两个儿子，此处可以发现，选择儿子这个过程是递归的，每次都是迭代的最优值，可以考虑树上贪心。

整体答案变化 => $[+1+2+2+2+2+2]$、$[+1+2+2+2]$、$[+1+2]$、$[+1]$、$[+1]\dots$
即整合出的子树的值为降序

```
#pragma GCC optimize(2)
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 2e6 + 7;
const int N = 2e5+7;
const ll INF = 1e18;
ll n, m, k, x, y, z;
#define Pa pair<ll, ll>
struct node
{
    ll to, next, w;
    node() {}
    node(ll w, ll to) : to(to), next(0), w(w) {}
    friend bool operator<(node a, node b)
    {
        return a.w > b.w;
    }
};
ll nxt, rnxt;
node e[maxn * 2];
ll h[N];
bool vis[N];
ll dis[N];
void add(ll u,ll v,ll w = 0){
    e[++nxt].next=h[u];
    e[nxt].w=w;
    e[nxt].to=v;
    h[u] = nxt;
}
void init(){
    nxt=1;
    for (int i = 0; i <= n; i++)//网络流使用2*n+7
    {
        h[i] = -1;
    }
}
void solve()
```

```
{
    string ss;
    cin >>n ;
    priority_queue<Pa> res;
    ll cnt =0;
    init();
    for(int i =2;i<=n;i++){
        cin >>x;
        add(x,i,1);//建树
    }
    vector<Pa >dp(n+1);
    vector<ll> dir(n+1);
    auto dfs = [&](auto self,ll u,ll f)->void{
        ll son = 0;
        vector<Pa> psi;//记录儿子的值
        for(int i =h[u];~i;i=e[i].next){
            ll v=e[i].to;
            if (v!=f){
                self(self,v,u);
                son++;
                psi.push_back(dp[v]);
            }
        }
        if (son==0){
            dp[u] = {1,1};
            cnt++;
        }else {
            sort(psi.begin(),psi.end(),greater<Pa>());
            for(int i =0;i<psi.size();i++){
                if (i<2){//拿最大的两个儿子
                    dp[u].first+=psi[i].first;
                    dp[u].second+=psi[i].second;
                }else {//剩下的儿子从树中割出
                    res.push(psi[i]);
                }
            }
            if (psi.size()>1){
                dp[u].first++;
            }
        }
    };
    dfs(dfs,1,0);
    res.push(dp[1]);
    ll sum =0;
    ll ans =0 ;
    for(int i =1;i<cnt;i++){
        sum++;
        ll add = 0;
        add = 2*sum-1;
```

```
        if (sum==res.top().second){
            ans+=res.top().first;
            sum=0;
            add=0;
            res.pop();
        }
        cout << ans+add<<" \n"[i==cnt];
    }
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    ll t = 1;
    // cin >>t;
    while (t--)
        solve();
    return 0;
}
```