树上启发式合并

树上启发式合并（DSU on tree）是一种处理树上不带修改的离线子树问题的算法，时间复杂度$O(nlogn)$。

算法流程

DSU on tree的流程很简洁，对于结点u：

①处理好所有轻儿子及其子树的答案，算好答案后删除刚刚算出来的轻儿子子树信息对于u结点答案的贡献。

②处理好重儿子及其子树的答案，算好答案后不删除贡献。

③暴力统计所有轻儿子及其子树以及u结点本身的贡献，与上步算出来的重儿子的贡献合并，就可以得到u结点的答案。

例题

CF600E Lomsat gelral / 求子树众数和

树的节点有颜色，一种颜色占领了一个子树，当且仅当没有其他颜色在这个子树中出现得比它多。求**占领每个子树的所有颜色出现次数之和。**

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N=2e5+7;
const int mod=1e9+7;

int n,flag,sum,maxc;
vector<int>G[N];
int sz[N],fa[N],son[N],col[N],cnt[N],ans[N];

inline void dfs(int x){  //处理基本信息
    sz[x]=1;
    for(auto to:G[x]){
        if(to==fa[x]) continue;
        fa[to]=x;
        dfs(to);
        sz[x]+=sz[to];
        if(sz[son[x]]<sz[to]) son[x]=to;
    }
```

```
}

inline void count(int x,int v){ //贡献某结点及其所有轻儿子的贡献
    cnt[col[x]]+=v;
    if(cnt[col[x]]>maxc){ //更新子树颜色最大编号以及颜色数
        maxc=cnt[col[x]];
        sum=col[x];
    }else if(cnt[col[x]]==maxc) sum+=col[x]; //两个颜色数量相同都得算
    for(auto to:G[x]){  //排除被标记的重儿子，统计其他儿子子树信息
        if(to==fa[x]||to==flag) continue;
        count(to,v);
    }
}

inline void solve(int x,bool kp){
    for(auto to:G[x]){  //轻儿子算答案删贡献
        if(to==son[x]||to==fa[x]) continue;
        solve(to,0);
    }
    if(son[x]) solve(son[x],1),flag=son[x]; //重儿子算答案不删贡献
    count(x,1); //暴力统计当前子树轻儿子的贡献合并到重儿子信息中
    flag=0;
    ans[x]=sum;
    if(!kp){  //如果当前是轻儿子，删除贡献
        count(x,-1);
        sum=maxc=0; //count会改变这两个值，需要初始化
    }
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n;
    for(int i=1;i<=n;i++) cin>>col[i];
    for(int i=1,u,v;i<n;i++){  //建图
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1);
    solve(1,0);  //树上启发式合并
    for(int i=1;i<=n;i++) cout<<ans[i]<<" ";
    return 0;
}
```

树的果实

给定一棵带边权的树，每次查询摘下一颗子树的贡献，即**(边权出现次数×边权)的平方**

```
//#pragma GCC optimize("Ofast", "inline", "-ffast-math")
//#pragma GCC target("avx,sse2,sse3,sse4,mmx")
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
#define int long long
```

```cpp
#define fi first
#define se second
#define pii pair<int,int>
#define mk make_pair
using namespace std;
const int N=2e5+7;
const int mod=1e9+7;
typedef long long ll;

pii P;
int n,m,c;
int a[N],cnt[N],ans[N],sz[N],son[N],now=0;
vector<pii>G[N];

inline void dfs(int x,int fa){  //处理树链
    sz[x]=1;
    for(auto y:G[x]){
        if(y.fi==fa)continue;
        dfs(y.fi,x);
        sz[x]+=sz[y.fi];
        if(sz[son[x]]<sz[y.fi])son[x]=y.fi;
    }
    return;
}

inline void ch(ll p,int w){
    now-=(p*cnt[p])*(p*cnt[p]);
    cnt[p]+=w;  //更新p数字的出现次数
    now+=(p*cnt[p])*(p*cnt[p]); //更新答案
}

inline void del(int x,int fa,ll p){  //删除对答案的贡献
    ch(p,-1);
    for(auto y:G[x]){
        if(y.fi!=fa){
            del(y.fi,x,y.se);
        }
    }
    return;
}

inline void add(int x,int fa,ll p){
    ch(p,1);
    for(auto y:G[x]){
        if(y.fi!=fa){
            add(y.fi,x,y.se);
        }
    }
    return;
}

inline void solve(int x,int fa,int kp){  //kp表示是否删除贡献
    for(auto y:G[x]){
        if(y.fi==fa||y.fi==son[x])continue;
        solve(y.fi,x,0);  //首先处理非重儿子
    }
    if(son[x]) solve(son[x],x,1); //处理重儿子
    for(auto y:G[x]){
```

```
        if(y.fi==fa)continue;
        else if(y.fi==son[x]) ch(y.se,1);
        else add(y.fi,x,y.se);
    }
    ans[x]=now;
    if(!kp){
        for(auto y:G[x]){
            if(y.fi==fa)continue;
            del(y.fi,x,y.se);
        }
    }
    return;
}

signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>m>>c;
    for(int i=1,u,v,w; i<n; i++) {
        cin>>u>>v>>w;
        G[u].push_back(mk(v,w));
        G[v].push_back(mk(u,w));
    }
    dfs(1,0);
    solve(1,0,0);
    for(int i=1,w; i<=m; i++) {
        cin>>w;
        cout<<ans[w]<<"\n";
    }
    return 0;
}
```

2021哈尔滨-Colorful Tree

给你一个树，所有叶子节点有一个颜色，每次你可以选择一个点进行操作，操作后会使得以这个节点代表的子树中所有的叶子节点的颜色被染成某种色。问你最少多少次操作，可以使得叶子节点满足题目给的颜色。

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+7;

int n,col[N],dp[N],fa[N];
vector<int>G[N];
multiset<int>st[N];

void dfs(int x){
    if(!G[x].size()){ //叶子结点，可以减少的次数为0
        st[x].insert(col[x]);
        dp[x]=1;
        return;
    }
    int mx=1;
    for(auto to:G[x]){
        if(to==fa[x]) continue;
```

```
        dfs(to); //算出儿子结点的答案
        dp[x]+=dp[to]; //
        if(st[x].size()<st[to].size()) swap(st[to],st[x]);
        for(auto it:st[to]){  //小的集合向大的集合合并
            st[x].insert(it);
            mx=max(mx,(int)st[x].count(it)); //记录出现最多的颜色次数
        }
    }
    if(mx>1){
        multiset<int>ss;
        for(auto it:st[x]){  //合并
            if(!ss.count(it)&&st[x].count(it)==mx) ss.insert(it);
        }
        swap(st[x],ss);
    }
    //总和-儿子需要的颜色次数+这个结点染色可以减少maxn次染色
    dp[x]=dp[x]-G[x].size()+mx;
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n;
    for(int i=2,x;i<=n;++i){ //建树
        cin>>fa[x];
        G[fa[x]].emplace_back(i);
    }
    int ans=0;
    for(int i=1;i<=n;++i){
        cin>>col[i];
        if(col[i]) ++ans;
    }
    dfs(1);
    cout<<ans-dp[1]+1; //总数-可以减少的+第一个节点染色
    return 0;
}
```

参考资料

OI-wiki

https://www.bilibili.com/video/BV1JE411d7tD

https://blog.csdn.net/m0_52398496/article/details/124402563