树链剖分_重链启发式合并

应用场景 多次询问子树中有多少节点满足某种性质，要求复杂度上界 $NlogN$。

具体场景

给出一棵树，每个节点有颜色，询问一些子树中不同的颜色数量。

考虑一个问题，能不能朴素地直接用一个数组把子树的每种颜色数量返回？

因为对于一个点，进入以它儿子为根的各个子树之前需要清空。同时，清空之后，在父节点时，又会面临需要使用全部的数据，因此我们又需要搜索一次所有的子树。

若使用1个数组，每个节点被遍历的次数是祖先链的长度（复杂度瓶颈）。

多个数组，没有清空带来的影响，但是合并复杂度、空间复杂度较高。

我们可以发现，只要我们可以保证，每个点被清空、再次搜索（访问）的次数，就可以保证复杂度。

我们看看这个过程是否有优化空间：先访问所有子树，然后清空。

不是所有子树答案必须清空。**最后一棵子树可以不清空。**

我们考虑让最后一棵子树尽可能大试试。即选择最后遍历**重儿子**。

模拟过程。

那，我们此时，不防引入所有的重链剖分的概念（重儿子会实际使用，其他概念用于辅助证明）。

观察访问次数，5 6为例：每个节点，除了自己被访问的那一次以外，还会额外祖先链上轻节点访问。

因此，访问次数复杂度上界 $NlogN$ 。

核心思想：对询问离线处理，在树上合并时，调整了合并的顺序，用到了类似于并查集启发式合并的优化，故名dsu on tree。合并的逻辑与重链相关，因此，也称为重链启发式合并。

注意此时 对dfs2，对答案相关的更新函数的实现。

```
void update(int u,int flag){
    cnt[c[u]]+=flag;
    if(cnt[c[u]]>max_){
        max_=cnt[c[u]];
        sum=c[u];
    }
    else if(cnt[c[u]]==max_){
        sum+=c[u];
    }

    for(int i=head[u];i;i=graph[i].next){
        int v=graph[i].to;
        if(v==father[u]||v==Son) continue;
        update(v,flag);
    }
}
void dfs2(int u,int tag){
    for(int i=head[u];i;i=graph[i].next){
        int v=graph[i].to;
        if(v==father[u]||v==son[u]) continue;
        dfs2(v,0);
    }
    if(son[u]){
        dfs2(son[u],1);
        Son=son[u];
    }

    update(u,1);
    ans[u]=sum;
    Son=0;

    if(!tag){
        update(u,-1);
        sum=max_=0;
    }
}
```