

树链剖分

树链剖分的思想及能解决的问题

树链剖分用于将树分割成若干条链的形式，以维护树上路径的信息。

具体来说，将整棵树剖分为若干条链，使它组合成线性结构，然后用其他的数据结构维护信息。

树链剖分（树剖/链剖）有多种形式，如重链剖分、长链剖分和用于 Link/cut Tree 的剖分（有时被称作“实链剖分”），大多数情况下（没有特别说明时），“树链剖分”都指“重链剖分”。

重链剖分可以将树上的任意一条路径划分成不超过 $O(\log n)$ 条连续的链，每条链上的点深度互不相同（即是自底向上的一条链，链上所有点的 LCA 为链的一个端点）。

重链剖分还能保证划分出的每条链上的结点 DFS 序连续，因此可以方便地用一些维护序列的数据结构（如线段树）来维护树上路径的信息。

如：

1. 修改树上两点之间的路径上所有点的值。
2. 查询树上两点之间的路径上结点权值的和/极值/其它（在序列上可以用数据结构维护，便于合并的信息）

除了配合数据结构来维护树上路径信息，树剖还可以用来 $O(\log n)$ 地求 LCA，在某些题目中，还可以利用其性质来灵活地运用树剖。

重链剖分

我们给出一些定义：

定义重子结点表示其子结点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子结点，就无重子结点。

定义轻子结点表示剩余的所有子结点。

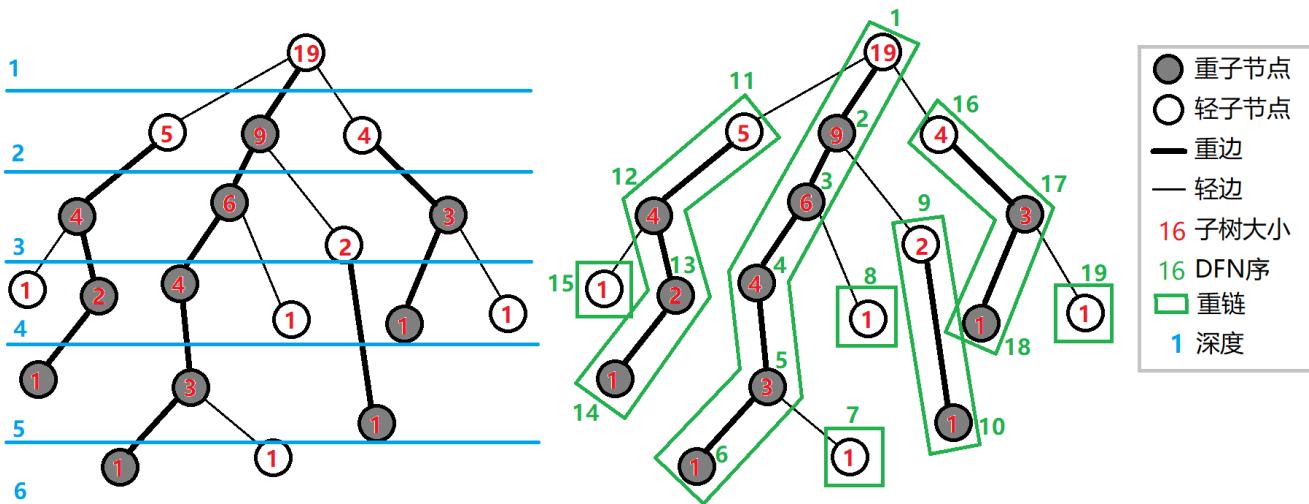
从这个结点到重子结点的边为重边

到其他轻子结点的边为轻边

若干条首尾衔接的重边构成重链

把落单的结点也当作重链，那么整棵树就被剖分成若干条重链。

如图：



实现

树剖的实现分两个 DFS 的过程。伪代码如下：

```

第一个 DFS 记录每个结点的父节点 father[] 深度 deep[] 子树大小 size[] 重子结点 hson[] $$
\begin{array}{l}\text{TREE-BUILD}(u,dep)\|\begin{array}{l}1 \& u.hson\gets 0\\2 \& u.hson.size\gets 0\\3 \& u.deep\gets dep\\4 \& u.size\gets 1\\5 \& \textbf{for } \text{each } u\text{ 's son } v\\6 \& \text{quad } u.size\gets u.size+v.size\\7 \& \text{quad } v.father\gets u\\8 \& \text{quad } \textbf{if } v.size>u.hson.size\\9 \& \text{quad } u.hson\gets v\\10 \& \textbf{return } u.size\end{array}\end{array} $$ 第二个 DFS 记录所在链的链顶 top 应初始化为结点本身 ) 、重边优先遍历时的 DFS 序 ( dfn[] DFS 序对应的结点编号 rank[] ) $$
\begin{array}{l}\text{TREE-DECOMPOSITION}(u,top)\|\begin{array}{l}1 \& u.top\gets top\\2 \& tot\gets tot+1\\3 \& u.dfn\gets tot\\4 \& rank(tot)\gets u\\5 \& \textbf{if } u.hson\neq 0\\6 \& \text{quad } \text{TREE-}\\ \text{DECOMPOSITION}(u.hson,top)\\7 \& \text{quad } \textbf{for } \text{each } u\text{ 's son } v\\8 \& \text{quad } \text{quad } \textbf{if } v\neq u.hson\\9 \& \text{quad } \text{quad } \text{TREE-DECOMPOSITION }(v,v)\end{array}\end{array} $$ 以下为代码实现。

```

我们先给出一些定义：

- $\text{fa}(x)$ 表示结点 x 在树上的父亲。
 - $\text{dep}(x)$ 表示结点 x 在树上的深度。
 - $\text{siz}(x)$ 表示结点 x 的子树的结点个数。
 - $\text{son}(x)$ 表示结点 x 的重儿子 \sqcup
 - $\text{top}(x)$ 表示结点 x 所在重链的顶部结点（深度最小）。
 - $\text{dfn}(x)$ 表示结点 x 的**DFS序**，也是其在线段树中的编号。
 - $\text{rnk}(x)$ 表示 DFS 序所对应的结点编号，有 $\text{rnk}(\text{dfn}(x)) = x \sqcup$

我们进行两遍 DFS 预处理出这些值，其中第一次 DFS 求出 $\text{fa}(x), \text{dep}(x), \text{siz}(x), \text{son}(x)$ ；第二次 DFS 求出 $\text{top}(x), \text{dfn}(x), \text{rnk}(x)$ 。

```
void dfs1(int o){  
    son[o]=-1;  
    siz[o]=1;  
    for(int i=h[o];i;i=nxt[i]) {
```

```

        if(!dep[p[j]]){
            dep[p[j]]=dep[o]+1;
            fa[p[j]]=o;
            dfs1(p[j]);
            siz[o]+=siz[p[j]];
            if(son[o]==-1||siz[p[j]]>siz[son[o]]) son[o]=p[j];
        }
    }
}

void dfs2(int o,int t){
    top[o]=t;
    cnt++;
    dfn[o]=cnt;
    rnk[cnt]=o;
    if(son[o]==-1) return;
    dfs2(son[o],t); //优先对重儿子进行 DFS 可以保证同一条重链上的点 DFS 序连续
    for(int j=h[o];j;j=nxt[j]){
        if(p[j]!=son[o]&&p[j]!=fa[o]) dfs2(p[j],p[j]);
    }
}

```

重链剖分的性质

树上每个结点都属于且仅属于一条重链

重链开头的结点不一定是重子结点（因为重边是对于每一个结点都有定义的）。

所有的重链将整棵树完全剖分

在剖分时优先遍历重儿子，最后重链的 DFS 序就会是连续的。

在剖分时重边优先遍历，最后树的 DFN 序上，重链内的 DFN 序是连续的。按 DFN 排序后的序列即为剖分后的链。

一棵子树内的 DFN 序是连续的。

可以发现，当我们向下经过一条轻边时，所在子树的大小至少会除以二。

因此，对于树上的任意一条路径，把它拆分成从 LCA 分别向两边往下走，分别最多走 $O(\log n)$ 次，因此，树上的每条路径都可以被拆分成不超过 $O(\log n)$ 条重链。

常见应用

路径上维护

用树链剖分求树上两点路径权值和，伪代码如下

```

$ \begin{array}{l} \text{TREE-PATH-SUM } (u,v) \\ \begin{array}{l} 1 \& \text{tot}\gets 0 \\ 2 \& \text{while } u.\text{top} \text{ is not } v.\text{top} \\ 3 \& \quad \text{if } u.\text{top}.\text{deep} < v.\text{top}.\text{deep} \\ 4 \& \quad \quad \quad \text{SWAP}(u,v) \\ 5 \& \quad \quad \quad \text{tot}\gets \text{tot}+\text{sum of values between } u \text{ and } u.\text{top} \\ 6 \& \quad \quad \quad u \text{ }\backslash\text{ gets } u.\text{top}.\text{father} \\ 7 \& \quad \quad \quad \text{tot}\gets \text{tot}+\text{sum of } \end{array} \end{array}

```

values between }u{text{ and }v\\ 8 & \textbf{return }tot \end{array}\end{array} \$\$ 链上的 DFS 序是连续的，可以使用线段树、树状数组维护。

每次选择深度较大的链往上跳，直到两点在同一条链上。

同样的跳链结构适用于维护、统计路径上的其他信息。

子树维护

有时会要求维护子树上的信息，譬如将以 \$x\$ 为根的子树的所有结点的权值增加 \$v\$。

在 DFS 搜索的时候，子树中的结点的 DFS 序是连续的。

每一个结点记录 bottom 表示所在子树连续区间末端的结点。

这样就把子树信息转化为连续的一段区间信息。

求最近公共祖先

不断向上跳重链，当跳到同一条重链上时，深度较小的结点即为 LCA。

向上跳重链时需要先跳所在重链顶端深度较大的那个。

参考代码：

```
int lca(int u,int v){  
    while(top[u]!=top[v]) {  
        if(dep[top[u]]>dep[top[v]])  
            u=fa[top[u]];  
        else  
            v=fa[top[v]];  
    }  
    return dep[u]>dep[v]?v:u;  
}
```

参考资料

<https://oi-wiki.org/graph/hld/>

