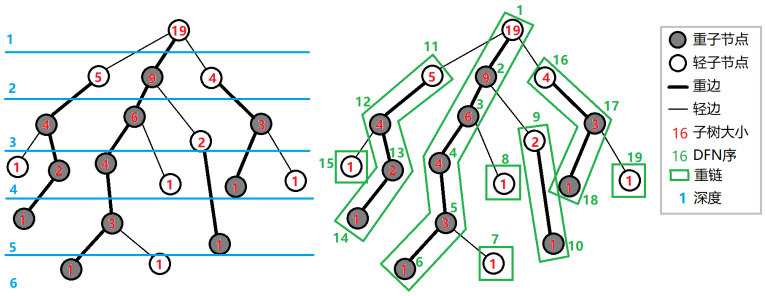
树链剖分

Heavy-light Decomposition

Idea：将树按照子树大小分为若干重链和轻链，每一条重链对应线段树或其他数据结构上连续的一段，而轻链则散落在重链对应的连续段落之间。询问或更改链上的信息时，两个端点不断沿着重链向上跳并统计或修改，直至跳到同一条重链上，也即跳到数据结构对应的连续一段上，统计或修改后完成操作。



Complexity：一次操作 $O(\lg n)$

Code（以线段树为例）：

```
int fa[N], dep[N], son[N], sz[N];
void dfs(int x, int f, int depth){
    fa[x] = f; dep[x] = depth; sz[x] = 1; son[x] = 0;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(edge[i].to == f) continue;
        dfs(edge[i].to, x, depth + 1);
        sz[x] += sz[edge[i].to];
        if(!son[x] || sz[edge[i].to] > sz[son[x]])
            son[x] = edge[i].to;
    }
}
int top[N], st[N], ed[N], dfsClock, func[N];
void dfs(int x, int tp){
    st[x] = ++dfsClock; func[dfsClock] = x; top[x] = tp;
    if(son[x])  dfs(son[x], tp);
    for(int i = head[x]; i; i = edge[i].nxt){
        if(edge[i].to == fa[x] || edge[i].to == son[x]) continue;
        dfs(edge[i].to, edge[i].to);
    }
    ed[x] = dfsClock;
}

#define lid id<<1
#define rid id<<1|1
#define mid ((tr[id].l + tr[id].r) >> 1)
#define len(id) (tr[id].r - tr[id].l + 1)
struct segTree{
    int l, r;
    LL sum, lazyAdd;
}tr[N<<2];
void pushup(int id){
    tr[id].sum = (tr[lid].sum + tr[rid].sum) % MOD;
}
void pushdown(int id){
    if(tr[id].lazyAdd && tr[id].l != tr[id].r){
        (tr[lid].lazyAdd += tr[id].lazyAdd) %= MOD;
        (tr[lid].sum += tr[id].lazyAdd * len(lid) % MOD) %= MOD;
        (tr[rid].lazyAdd += tr[id].lazyAdd) %= MOD;
        (tr[rid].sum += tr[id].lazyAdd * len(rid) % MOD) %= MOD;
        tr[id].lazyAdd = 0;
    }
}
void build(int id, int l, int r){
    tr[id].l = l; tr[id].r = r;
    tr[id].sum = tr[id].lazyAdd = 0;
    if(tr[id].l == tr[id].r){
```

```
        tr[id].sum = a[func[l]];
        return;
    }
    build(lid, l, mid);
    build(rid, mid+1, r);
    pushup(id);
}
void add(int id, int l, int r, LL val){
    pushdown(id);
    if(tr[id].l == l && tr[id].r == r){
        (tr[id].lazyAdd += val) %= MOD;
        (tr[id].sum += val * len(id) % MOD) %= MOD;
        return;
    }
    if(r <= mid)    add(lid, l, r, val);
    else if(l > mid)    add(rid, l, r, val);
    else{
        add(lid, l, mid, val);
        add(rid, mid+1, r, val);
    }
    pushup(id);
}
LL query(int id, int l, int r){
    pushdown(id);
    if(tr[id].l == l && tr[id].r == r)
        return tr[id].sum % MOD;
    if(r <= mid)    return query(lid, l, r);
    else if(l > mid)    return query(rid, l, r);
    else    return (query(lid, l, mid) + query(rid, mid+1, r)) % MOD;
}

void addPath(int u, int v, LL val){
    while(top[u] != top[v]){
        if(dep[top[u]] < dep[top[v]])   swap(u, v);
        add(1, st[top[u]], st[u], val);
        u = fa[top[u]];
    }
    if(dep[u] < dep[v]) swap(u, v);
    add(1, st[v], st[u], val);
}
LL queryPath(int u, int v){
    LL res = 0;
    while(top[u] != top[v]){
        if(dep[top[u]] < dep[top[v]])   swap(u, v);
        (res += query(1, st[top[u]], st[u])) %= MOD;
        u = fa[top[u]];
    }
    if(dep[u] < dep[v]) swap(u, v);
    (res += query(1, st[v], st[u])) %= MOD;
    return res;
}
void addSubtree(int u, LL val){ add(1, st[u], ed[u], val); }
LL querySubtree(int u){ return query(1, st[u], ed[u]); }
```