Heavy Light Decomposition

Pranav Mathur

January 17th, 2020

1 The Problem

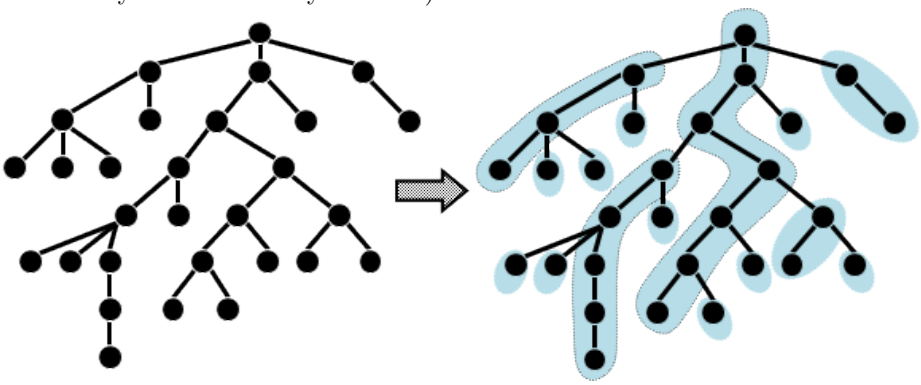
Suppose you have a rooted tree with weights at each node. Answer a series of queries that either update the value at a node, or query the maximum value along a path from a node to the root.

The naive solution to this problem is to simply walk up that path from a node to the root and calculate the maximum value along the path. This solution has runs in $O(1)$ time for updates and $O(n)$ time for queries, which is too slow for multiple queries.

The problem of finding the maximum value along a path is a signal that segment trees might be useful, but how do we make a segment tree from a tree? We will do this using a technique called heavy light decomposition.

2 Heavy Light Decomposition

The idea of HLD is to label every edge in the tree as heavy or light. For each vertex, label the edge leading to the largest subtree as heavy, and label all other edges as light. This gives us the interesting property that if we consider any subtree of size n, all of its "light" subtrees have a size of at most $n/2$ (if they didn't they would be heavy subtrees).



Labeling edges in this way gives us some useful results. First, note that since every vertex has exactly one heavy edge, **we can decompose the tree into disjoint paths consisting of heavy edges**. Second, note that since each light subtree has size $n/2$ and heavy paths are connected by light edges, **any path from a node to the root of the tree will pass through at most** $\log N$ **heavy paths**. If we maintain a segment tree for each heavy path that allows us to find the maximum value of the nodes in that path), we no longer have to traverse every node on the path to the root. Instead we can just travel along the heavy paths from the node to the root. Since there are at most $\log N$ heavy paths from the node to the root querying each path takes $\log N$ time, total runtime for a query is $\log^2 N$!

3 A Trickier Problem

Now let's consider a slightly trickier problem: find the maximum value along the path from node a to node b in the tree. This problem is different, but can still be solved using HLD. The process we will use is similar to what we do to find the least common ancestor of two nodes. While nodes a and b are not in the same heavy path, move the deeper of the two nodes up its heavy path and do a segment tree query on that path. When they are on the same path, do a segment tree query from a to b on that path. Then, return the max value found on all segment tree queries.

4 Implementation

First, we will do a DFS to calculate each node's depth, its parent, and the heavy edge emanating from that node. This allows us to label heavy edges in the tree, while also recording information we will need when performing queries (parent and depth).

Algorithm 4.1 Label Heavy and Light Edges

```
for all vertices v do
    parent[v] ← −1
    depth[v] ← 0
    heavy[v] ← −1
function LABEL(v)
    size ← 1
    maxChildSize ← 0
    for all neighbors c of v do
        if c ≠ parent[v] then
            parent[c] ← v
            depth[c] ← depth[v] + 1
            childSize ← label(c)
            size += childSize
            if childSize > maxChildSize then
                maxChildSize ← childSize
                heavy[v] ← c
    return size
```

Next, we will decompose the tree by recording the head of each heavy path and building the segment tree. Note that instead of maintaining a segment tree for each heavy path, we can maintain a single segment tree by processing every node in a single heavy path consecutively. Thus, the segment tree consists of a series of heavy paths.

Algorithm 4.2 Find the Decomposition

```
curPos ← 0
function DECOMPOSE(v, h)
    head[v] ← h
    pos[v] ← curPos + +
    segTreeArr[curPos] ← arr[v]
    if heavy ≠ −1 then
        decompose(heavy[v], h)
    for all neighbors c of v do
        if c ≠ parent[v] and c ≠ heavy[v] then
            decompose(c, c)
```

Finally, we will perform queries from node A to node B using an algorithm similar to an LCA query.

Algorithm 4.3 Query from A to B

```
function QUERY(a, b)
    ans ← 0
    while head(a) ≠ head(b) do
        if depth[head[a]] ≤ depth[head[b]] then
            heavyPathMax ← segTreeQuery(pos[head[a]], pos[a])
            ans ← max(ans, heavyPathMax)
            a ← parent[head[a]]
        else
            heavyPathMax ← segTreeQuery(pos[head[b]], pos[b])
            ans ← max(ans, heavyPathMax)
            b ← parent[head[b]]
    if depth[a] ≤ depth[b] then
        lastPathMax ← segTreeQuery(pos[a], pos[b])
    else
        lastPathMax ← segTreeQuery(pos[b], pos[a])
    ans ← max(ans, lastPathMax)
    return ans
```

5 Applications and Problems

HLD is a very useful technique that doesn't just apply to finding the maximum value along the path between two nodes. Since the underlying data structure is a segment tree, HLD can be used to perform any query on the path between two nodes that segment trees can do for an array. The following problems all use HLD or its motivating ideas.

USACO December 2011, Gold, Grass Planting

USACO February 2019, Gold, Cowland

USACO December 2019, Gold, Milk Visits

This dude put a bunch of problems at the bottom of his HLD article:
https://blog.anudeep2011.com/heavy-light-decomposition/