Heavy Light Decomposition

Ian

Problem

Given a tree, the i-th node contains a value, a_i. There are Q queries. Each query could be in two types:

1. u, v, delta. Add delta to all nodes between path(u, v).
2. u, v. Find sum of a_i for all node i that are between path(u, v).

N, Q <= 1e5

Naive Solution

For type 1 query, start dfs in node u and find the path(u, v). Then just add the delta.

For type 2 query, similiar approach as query 1.

Naive Solution

Tip: for path problems, we could decompose path(v, u) to path(v, w) and path(u, w) where w is the lca of node v, u.

So for both query, we could see them as modifying/querying in path(v, w) and path(u, w). Be careful that we don’t want to work on node w twice. One method is to -val(w) when query and -delta to w when modify. Second method is to consider path(v, w) and path(u, r), where r is the second node in path(w, v) (so r is the child of w and subtree of w contains u).

Naive Solution

Still, the algorithm works in O(QN), which is bad.

Consider a full binary tree, this naive algorithm could work in O(Q log N), as the depth of the binary tree is only log(n).

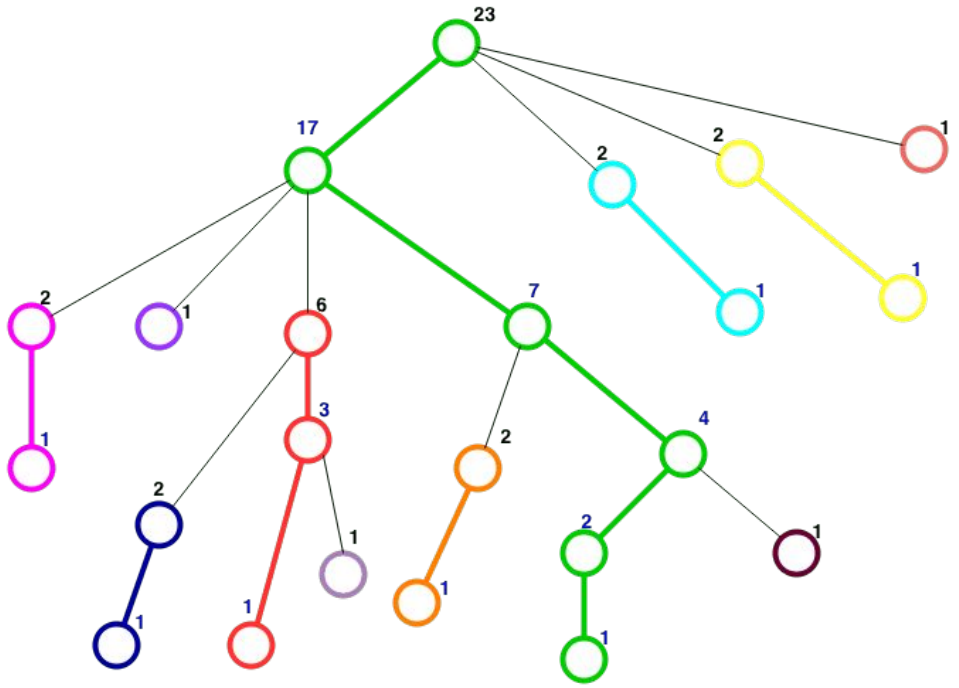
If we could turn the tree to some full binary tree-like thing, then we could have a good time complexity.

Heavy Light Decomposition

We can actually split the tree into several chains (paths), such that for every node v, it only pass through O(log n) chains (paths) to reach the root (and any ancestor). (Also the chain would only go from parent to one child)

Since the chains are some consecutive nodes in the tree (in sequence), we could use some data structure to query the information fast.

So we could achieve O(log^2 n) per query! (pass through O(log n) chains, each chain use O(log n) time to query data)



Each Chain is represented with different color.
Thin Black lines represent the connecting edges. They connect 2 chains.

Source: https://blog.anudeep2011.com/heavy-light-decomposition/

Heavy Light Decomposition

For every node, the child with the largest subtree size would be called “heavy”, “light” otherwise.

Heavy light decomposition works like this:

> For every node v, extend its chain to its heavy child.
>
> Then, for every light child, start a new chain.

Heavy Light Decomposition

To query from v to w(some parent of v).

We could do this:

```
    if v and w is in the same chain
        add information from interval(v, w) to answer
    else
        add information from interval(chain_head, v) to answer
        set v as parent(chain_head) and do this all over again
```

Heavy Light Decomposition

Implementaion:

https://codeforces.com/blog/entry/53170

Heavy Light Decomposition

HLD is actually a special dfs order (and we need to store the chain head), there is nothing magic.

Heavy Light Decomposition

let f(n) be the maximum chains the one node need to pass through when it goes to the root, for any tree with n nodes.

Consider using one node as root, then we have some subtree T1, T2… (assuming T1 has the largest size).

f(n) = max(f(T1), max(f(T2), …, f(Tk)) + 1)

f(1) = 1

We could see to maximize f(n), we need to have a binary tree, which has log n depth.

Practice Problem

---

https://www.spoj.com/problems/QTREE/

`Hint: to handle edge problem, we could think of node v storing value of edge(parent(v), v)`

Some Tricks Using HLD

Sometimes the modification of the data is hard to maintain, e.g. applying some non-associative functions like mod, and we can’t really use segment tree.

Still, one could try to use heavy light decomposition. Instead of using segment tree to do query, we simply have a naive for loop to update them.

Some Tricks Using HLD

The time complexity is O(QN), but since the data is consecutive, we would only access O(q log n) interval and have O(q log n) cache miss.

We could sometimes solve problems by just rearranging the dfs order (heavy light decomposition).

Source: https://codeforces.com/blog/entry/67001

Some Tricks Using HLD

Another trick is related to smaller to larger (merging smaller data structure to larger one).

It is used to solve subtree problem, probably all nodes have a value and we want to ask some information about the subtrees (offline).

Sometimes when problems have tight ML/TL or we need to merge arrays, then we could try to use some HLD trick.

Some Tricks Using HLD

We dfs to all light children first, and removing all the records. Then we dfs to the heavy child, without removing all the records. After that, we dfs again to all light children. We then would get all the data from subtree.

```
solve(v)
    for all light child u, solve(u) and remove(u) #remove all records in subtree u
    solve(heavy child)
    for all light child u, add(u) #add all records in subtree u
    add v’s information into some data structure
```

Some Tricks Using HLD

The previous tricks works in O(n log n).

We could first see that the solve function works in O(N) (just like normal dfs). For the add and remove function, considering a node v, it would be accessed one time when its ancestor is some light child. Since there is O(log n) chain from root to v, there is O(log n) ancestor that is light child.