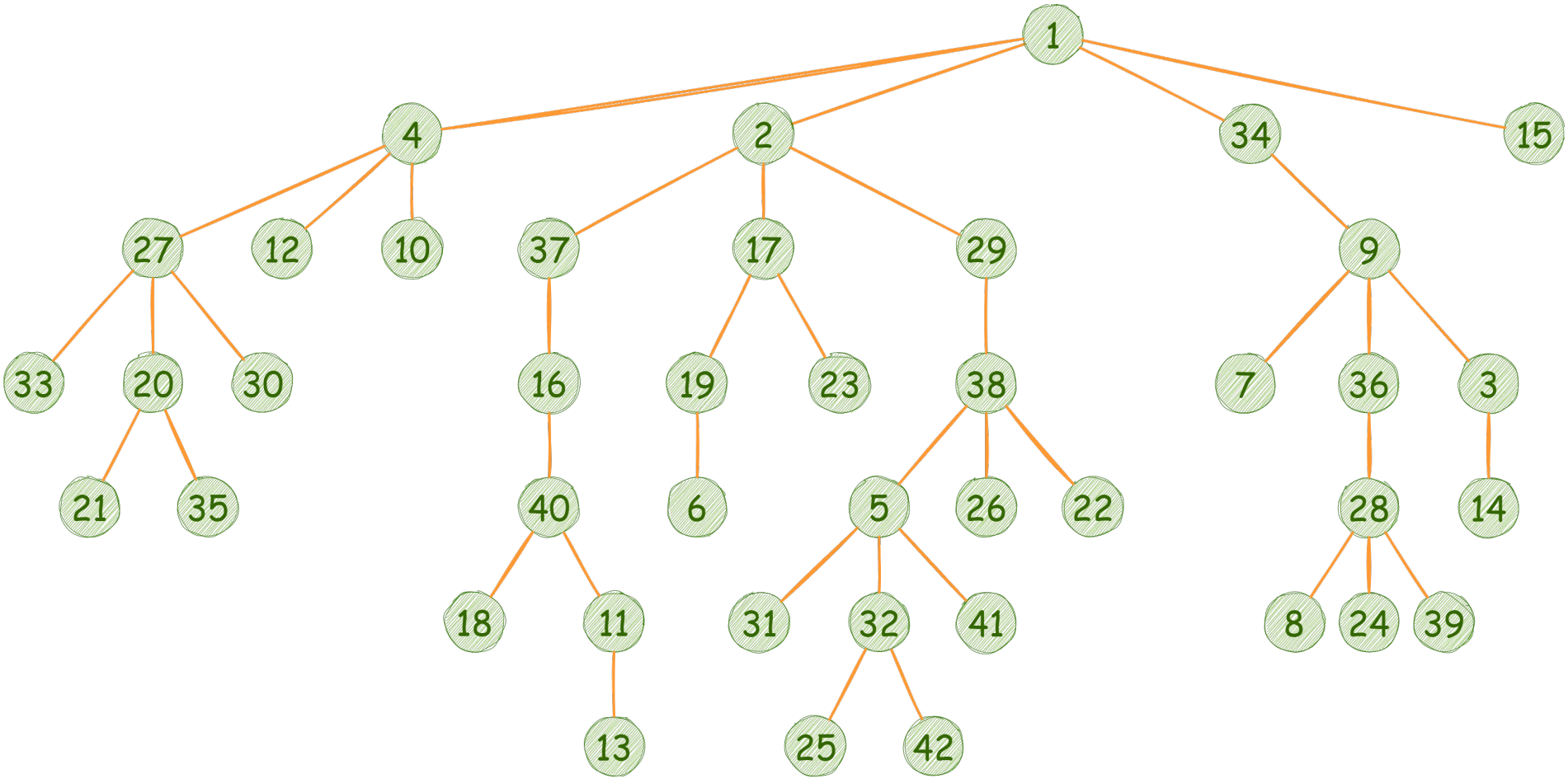
inzva ACWC'23 | Tree Notes

Prepared with ♥, just for you...

inzva Team

June 29, 2024

Tree


1
4
2
34
15
27
12
10
37
17
29
9
33
20
30
16
19
23
38
7
36
3
21
35
40
6
5
26
22
28
14
18
11
31
32
41
8
24
39
13
25
42

Binary Lifting

Ancestor

[Parent's parent's...] parent, and also itself.

Motivation: Finding the k-th ancestor in a few $(\log(k))$ operations instead of k. "Lifting" in binary!

1. Store parents with DFS/BFS
2. Knowing the first ancestors (parent), store the second ancestors (parent's parent)
3. Knowing the second ancestors (parent's parent), store the fourth ancestors (second ancestor's second ancestor)
4. Knowing the fourth ancestors, store the eighth ancestors
5. ...and so on. Guess where it's going...

Binary Lifting — Data Structure

`parent[i][j]`: Node `i`'s 2^{j}-th ancestor (`parent[n][log(n)]`)

- `parent[i][0]`: Parent (first ancestor)
- `parent[i][1]`: Second ancestor
- `parent[i][2]`: Fourth ancestor

How to?

```
parent[i][j] = parent[parent[i][j-1]][j-1]
```

Binary Lifting — Nice to Know

- Binary lifting is not specific to trees. It works on any graph where each node's out degree (number of outgoing edges) is at most 1.
- Notice that every tree satisfies this condition. Binary lifting uses the "child → parent" direction.
- Yes, cycles are allowed!

Common Ancestor

- "Common" requires (at least) two nodes (Let's stick with two for convenience)
- Common: Such a node that is an ancestor of both
- There might be multiple common ancestors!

Lowest Common Ancestor

- LCA(u, v) — The **lowest common ancestor** of nodes u and v
- Of all common ancestors, the one closest to the nodes (furthest from the root)
- Unique, of course

Lowest Common Ancestor — How?

1. Find and store heights (levels) with DFS/BFS
2. Binary lift the lower node to the other's level
   - If they become the same node, the one higher is the LCA of the other
   - Equivalent of finding the k-th ancestor of the lower node if the height difference is k
3. Afterward, binary lift both nodes simultaneously **unless** they become the same
   - Equivalent of finding the $(k-1)$-th ancestor if the height difference between them is k
4. In the end, the LCA is the parent of both!

About the third step

During lifting, we can't know whether we've found the LCA or got above it once we get to the same node. This is why we always want to stay below the LCA, i.e., keep them different.

Lowest Common Ancestor — Use Cases

- The distance between u and v in $O(\log n)$
  - We know how to find LCA, and also heights. That's everything we need!

Euler Tour

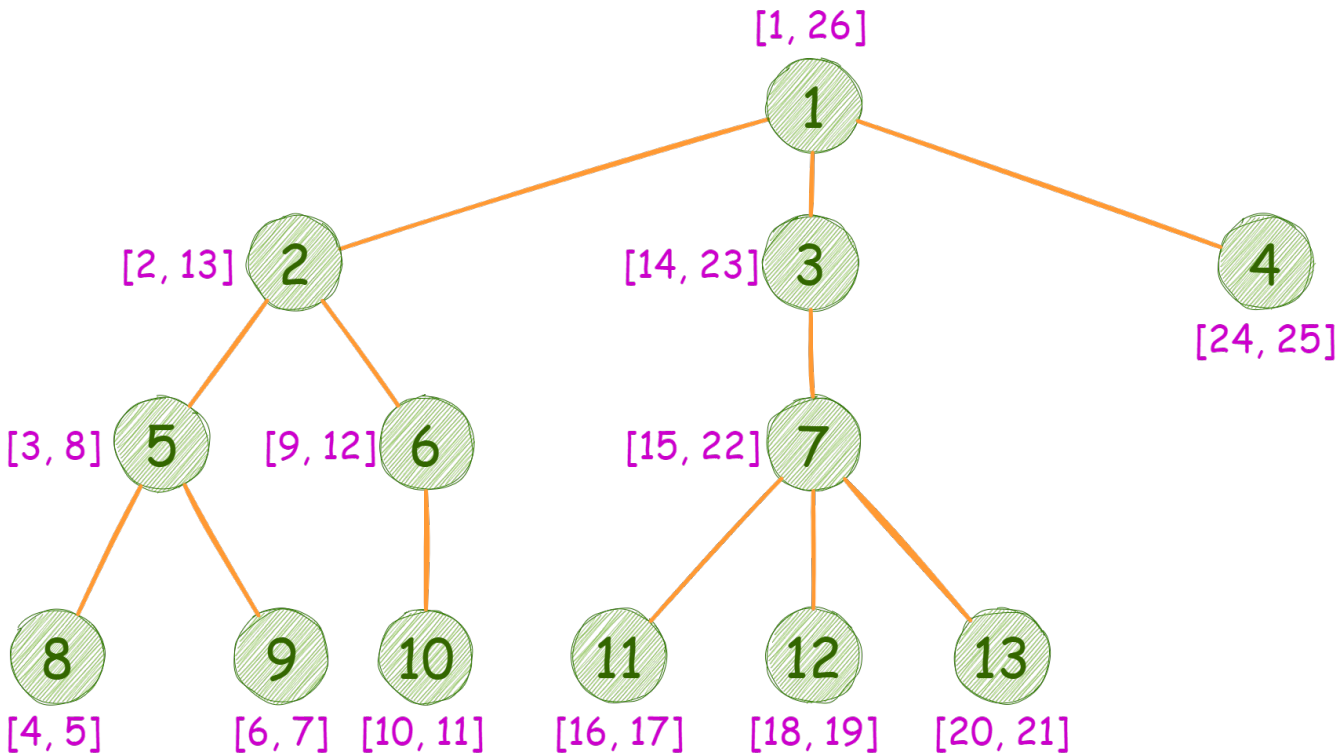
What if we store the `in` and `out` times while DFS-ing?

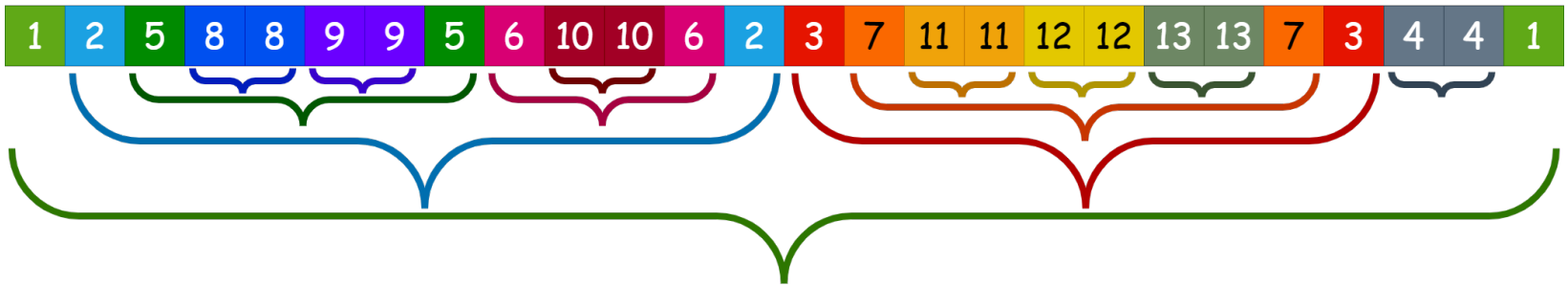
- `in[i]` — The time `dfs(i)` is called
- `out[i]` — The time `dfs(i)` returns
- The interval `[in[i], out[i]]` represents the subtree of node `i`
- `u` is `v`'s ancestor $\iff$ `in[u]` $>$ `in[v]` $\&$ `out[v]` $<$ `out[u]`

Seems familiar...

It's nothing different than the preorder traversal order.

Euler Tour


[1, 26]
1
[2, 13]
2
[14, 23]
3
4
[24, 25]
[3, 8]
5
[9, 12]
6
[15, 22]
7
8
9
10
11
12
13
[4, 5]
[6, 7]
[10, 11]
[16, 17]
[18, 19]
[20, 21]


1 2 5 8 8 9 9 5 6 10 10 6 2 3 7 11 11 12 12 13 13 7 3 4 4 1

Euler Tour — `in` & `out`

A `timer` is required. Different strategies exist:

- `timer++` on both call and return
- `timer++` on only call
- `timer++` on only return

Regardless:

- `in[i] = timer` at the beginning
- `out[i] = timer` in the end

Holy Sources

- https://cp-algorithms.com/graph/lca_binary_lifting.html
- https://usaco.guide/gold/tree-euler?lang=cpp
- https://usaco.guide/plat/binary-jump?lang=cpp