Queries on Trees

Special class

Tanuj Khattar • Nov 22, 2020

Queries on Trees

Course: https://unacademy.com/a/i-p-c-intermediate-track

tanujkhattar@

Objective

- Types of Query / Update Problems on Trees
  - Path Query / Update
  - Subtree Query / Update
- Heavy Light Decomposition (HLD)
  - Theory
  - Implementation
- Euler Tour Technique
  - Theory
  - Implementation
- Other techniques for Queries on Trees
  - Centroid Decomposition
  - Auxiliary Tree
- Conclusion

Path Query and Update Problems

- **Path Query:** Given two nodes x, y - compute some function f(x, y) that depends on the path between nodes x & y.
  - Eg: sum, min, max, number of distinct elements etc.
- **Point Update:** Change the value of any one edge / node in the tree
- **Path Range Update:** Change the value of all nodes/edges on a path.
  - Eg: Add x to all nodes in a path, take mod x for all nodes in a path etc.

Subtree Query and Update Problems

- **Subtree Query:** Given a node x - compute some function f(x) that depends on values of nodes/edges in the subtree of x
    - Eg: sum, min, max, number of distinct elements etc.
- **Point Update:** Change the value of any one edge / node in the tree
- **Subtree Range Update:** Change the value of all nodes/edges in the subtree of a node x.
    - Eg: Add “val” to all nodes in the subtree of node x etc.

How to support Updates & Queries on a Tree?

- **Step-1: Find a way to "Linearize" the tree into an array.**
    - **Heavy Light Decomposition:** Any path between (x, y) can be represented as concatenation of at-most logN different [L, R] ranges in the linearised array.
    - **Euler Tour Traversal:** Any subtree of a node x corresponds to a single range [L, R] in the linearised array.
- **Step-2: Use one of the "standard" techniques to solve the update/query problem on the linearised tree.**
    - Eg: Segment Trees, Square Root Decomposition etc.

Heavy Light Decomposition

- Break the tree into vertex-disjoint “chains” going from “top” to “bottom”
- For every node, the edge b/w max. size subtree child (**special child**) will be a “heavy” edge, rest will be “light” edges (**normal child**).
- Every light edge connects two different chains / a new chain starts after every light edge.

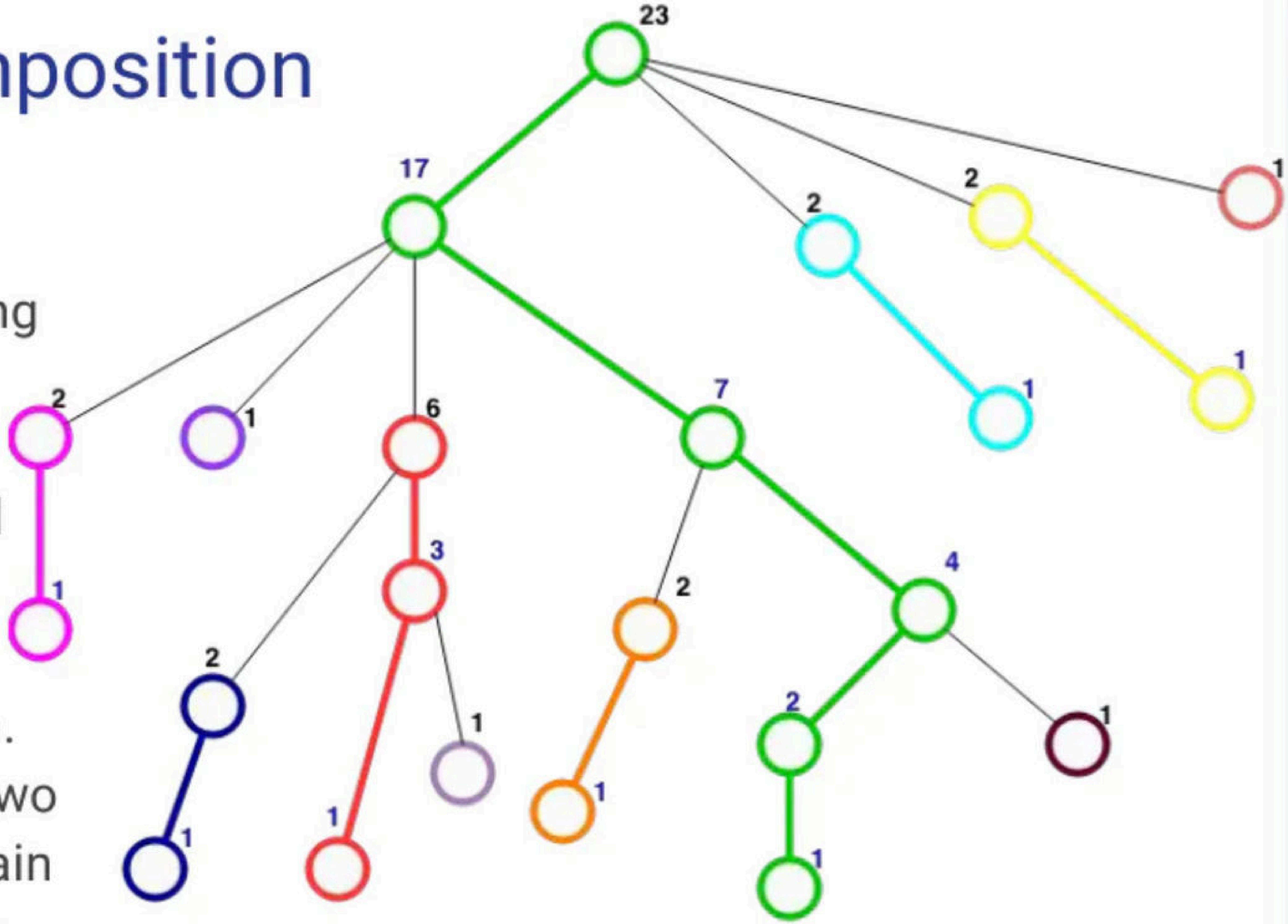

Image Source: https://blog.anudeep2011.com/heavy-light-decomposition/

HLD Visualization

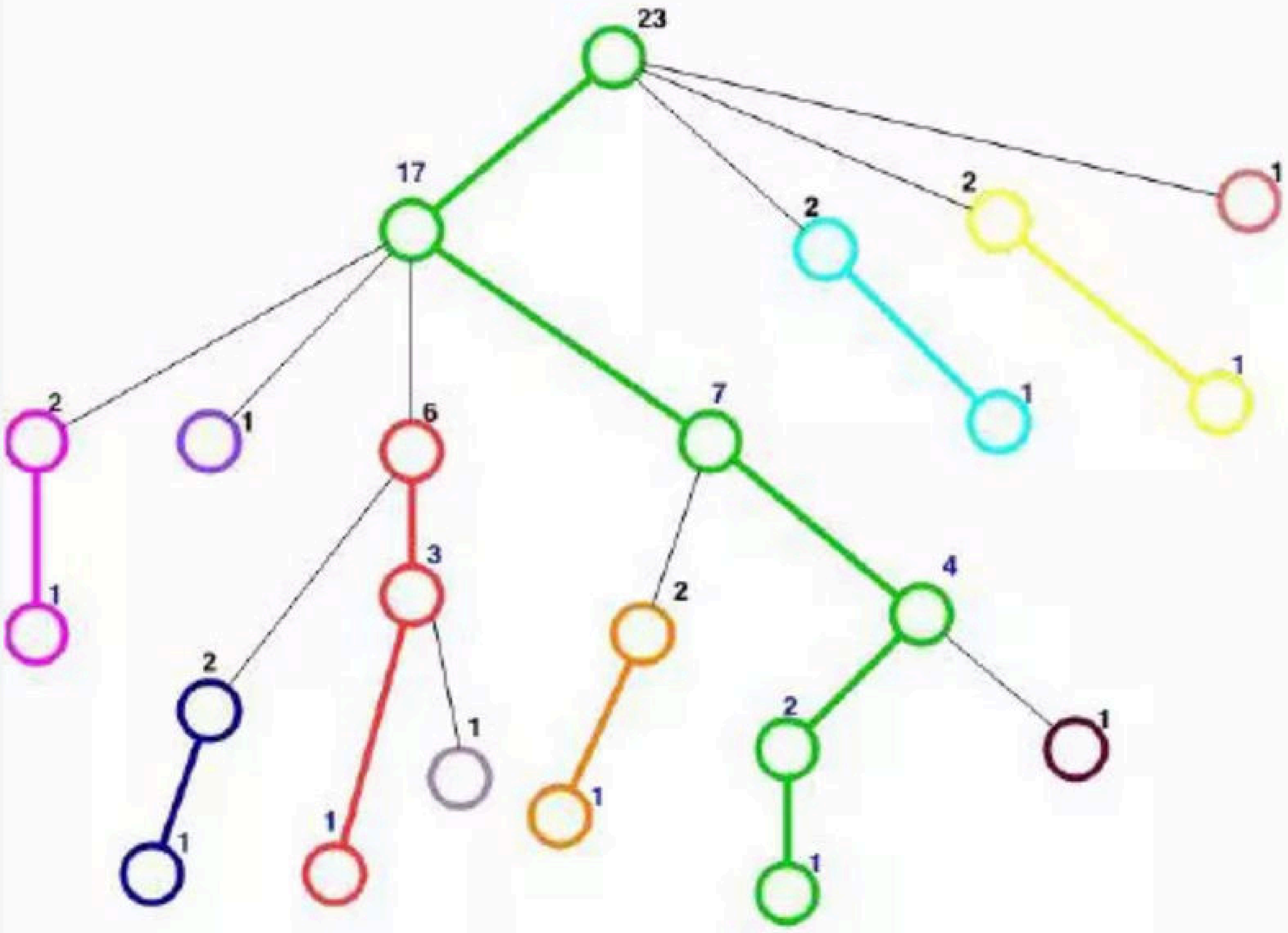

Image Source: https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.

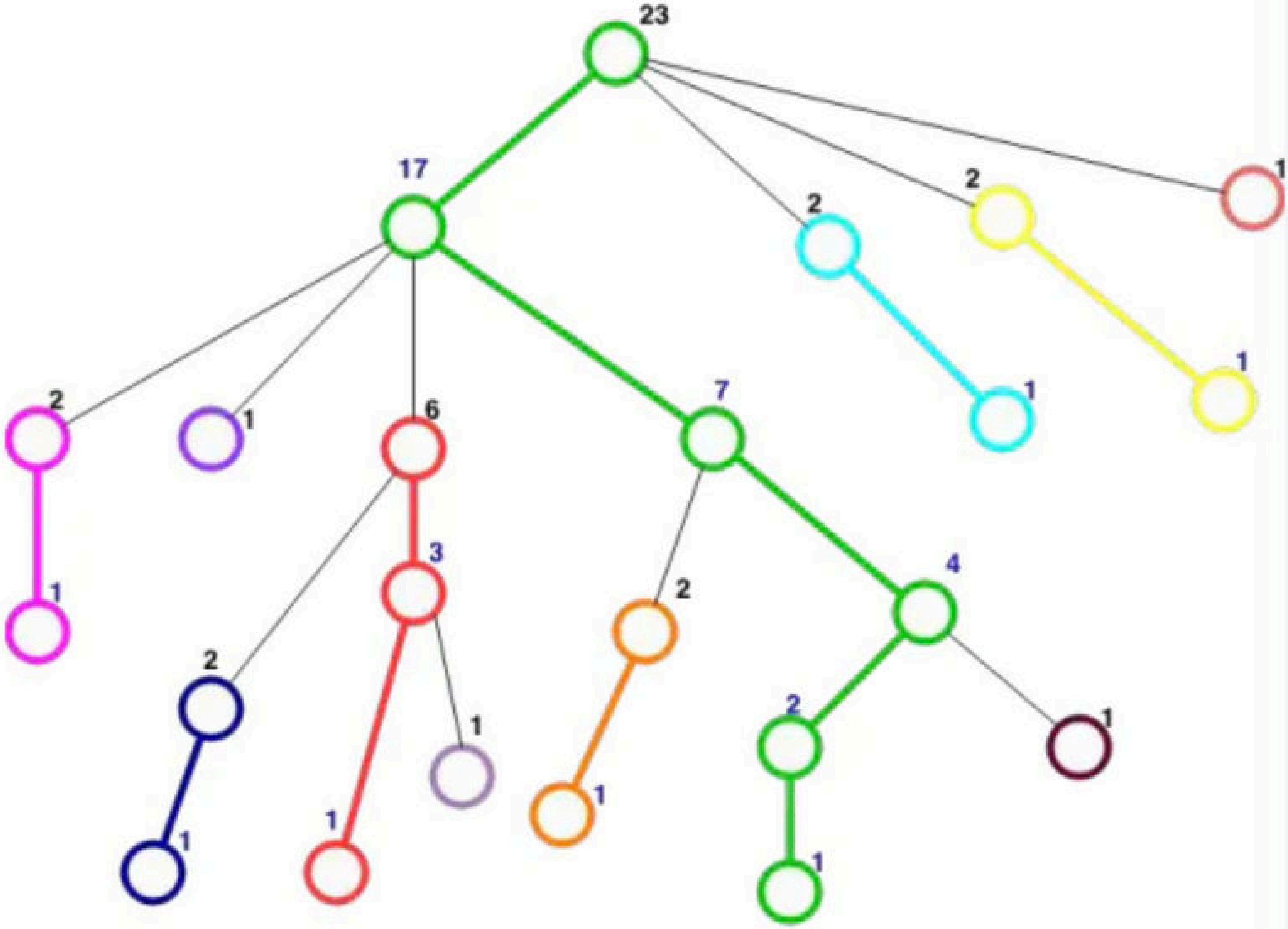

Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.
- Subtree size reduces by at-least half on traversing a "light" edge.

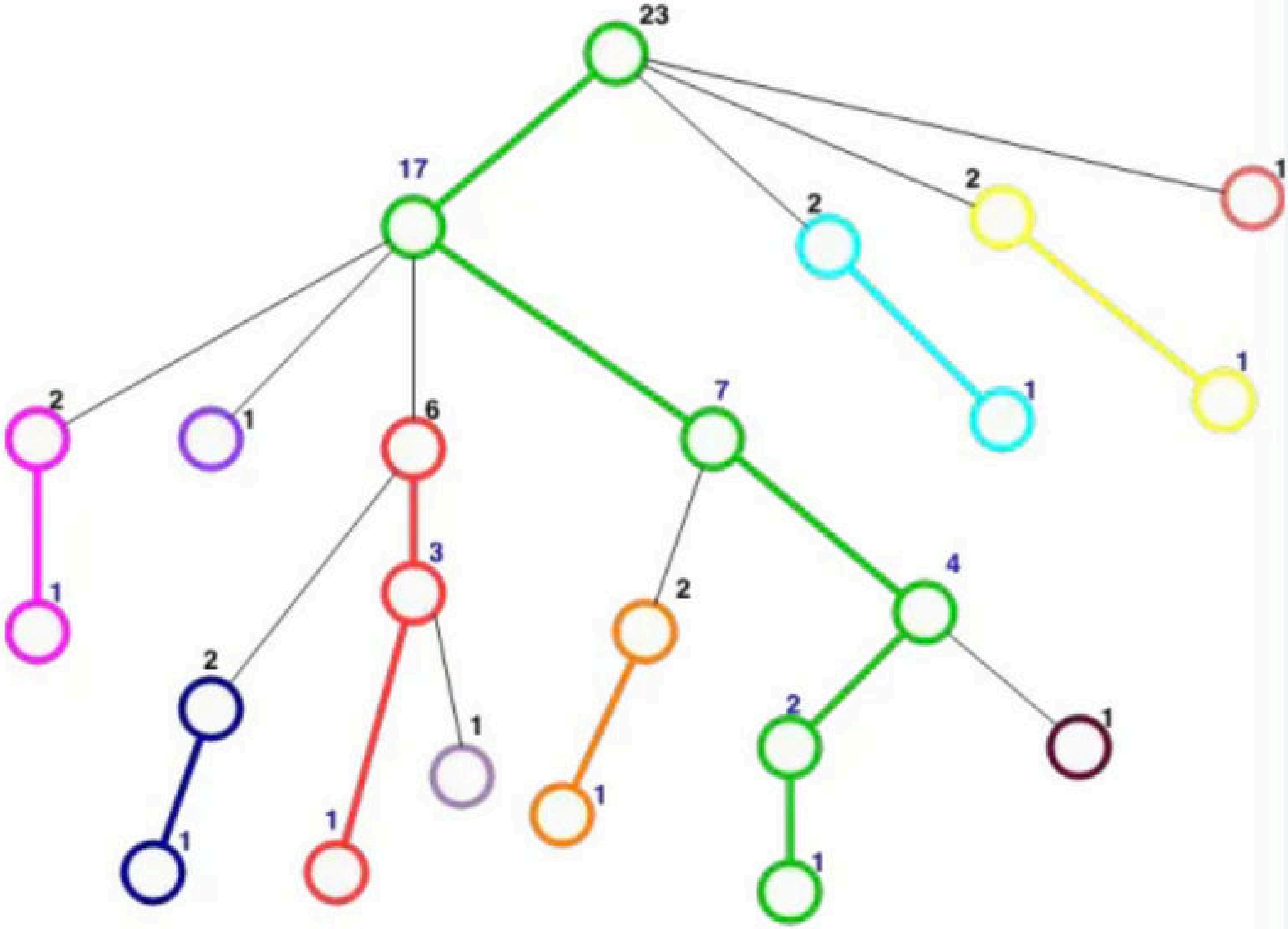

Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.
- Subtree size reduces by at-least half on traversing a "light" edge.
- Therefore, we can go up from any node **x** to it's ancestor node **p** by changing at-most logN chains.

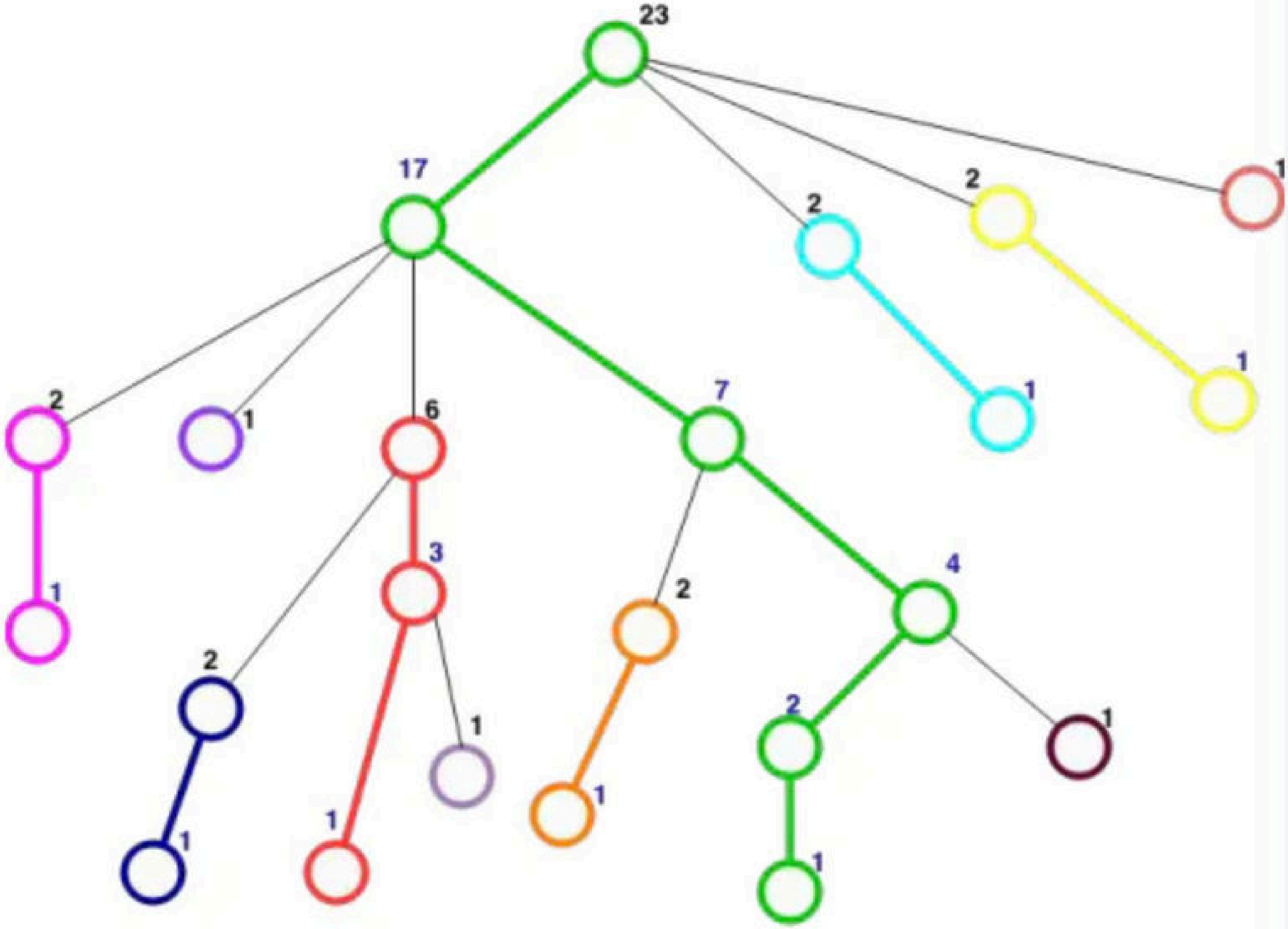

Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Properties

- Every vertex is part of exactly 1 chain.
- Every chain forms a subarray in the "linearised" tree.
- Subtree size reduces by at-least half on traversing a "light" edge.
- Therefore, we can go up from any node **x** to it's ancestor node **p** by changing at-most logN chains.
- Any path A -- B can be written as A -- LCA + LCA -- B; and hence can be traversed by changing at-most 2 * logN chains.

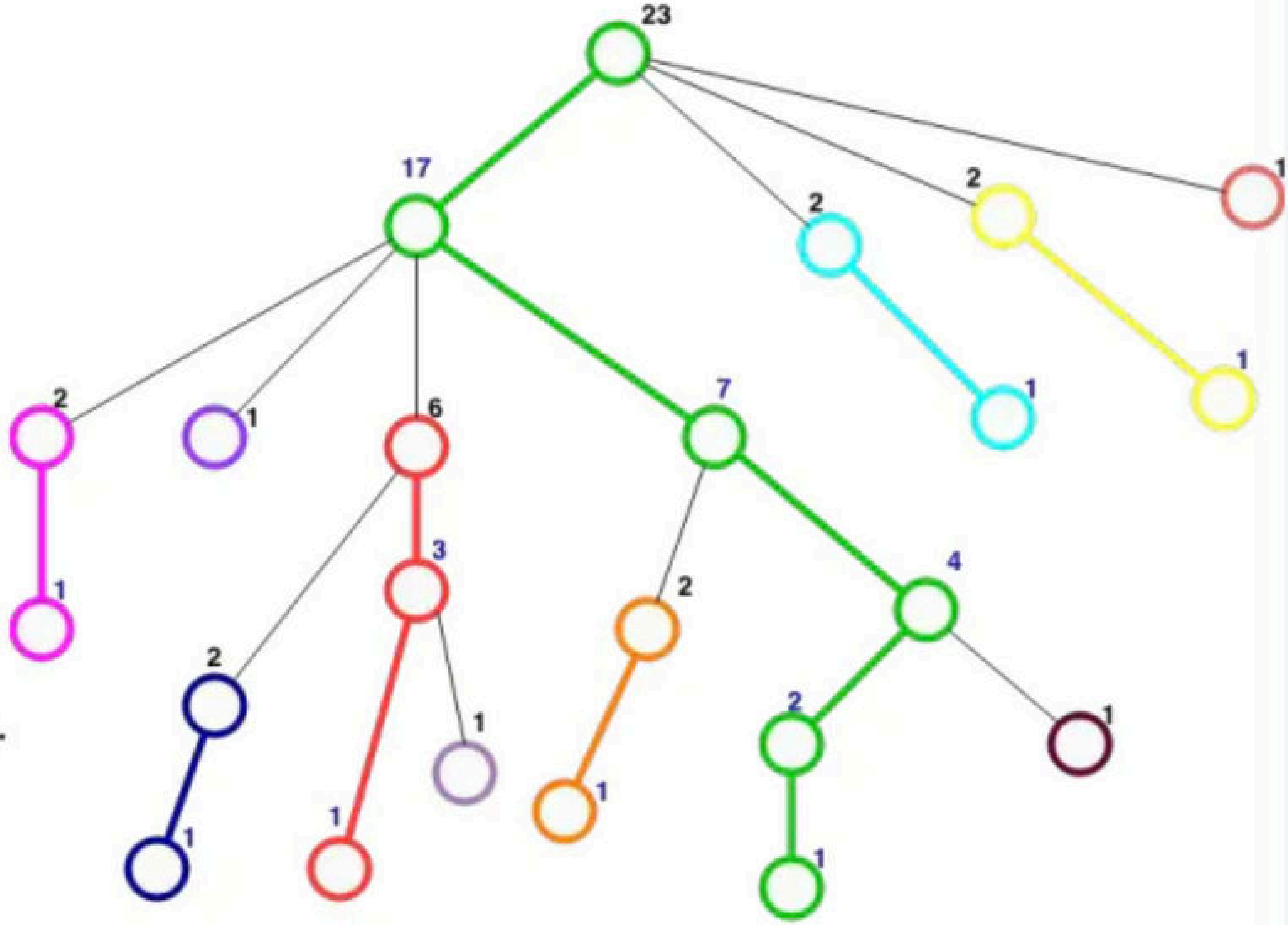

Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

HLD - Steps to support path updates / queries

- Decompose the tree into chains via HLD.
- Linearise the chains into an array and build a Data Structure on the array that supports range queries / updates.
- For any path query/update b/w nodes A & B; process it as a query/update on O(logN) different ranges in the linearised array -- corresponding to O(logN) chains that we need to traverse while going from A -- LCA -- B in the original tree.
- Therefore, total time taken will be O(logN * TimeTakenByLinearDS)

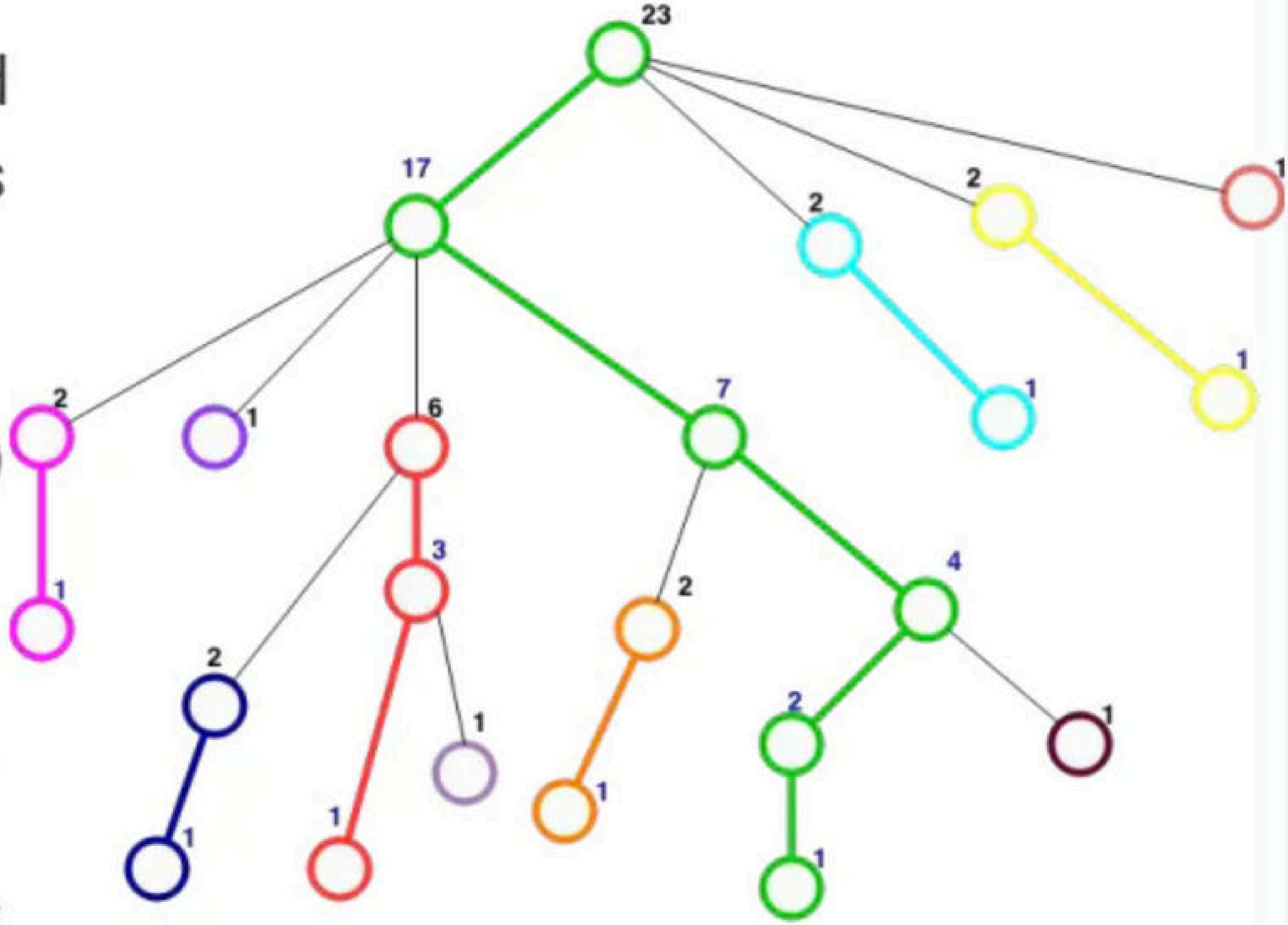

Image Source:
https://blog.anudeep2011.com/heavy-light-decomposition/

Heavy Light Decomposition - Implementation

Euler Tour Technique (ETT)

Way-1: Insert every node twice

- Insert every node/edge in the euler tour array whenever you enter/exit the node.
- Therefore, every node/edge of the tree will occur twice in the euler tour array -- at indices start[x] and end[x] for a given node/edge x.

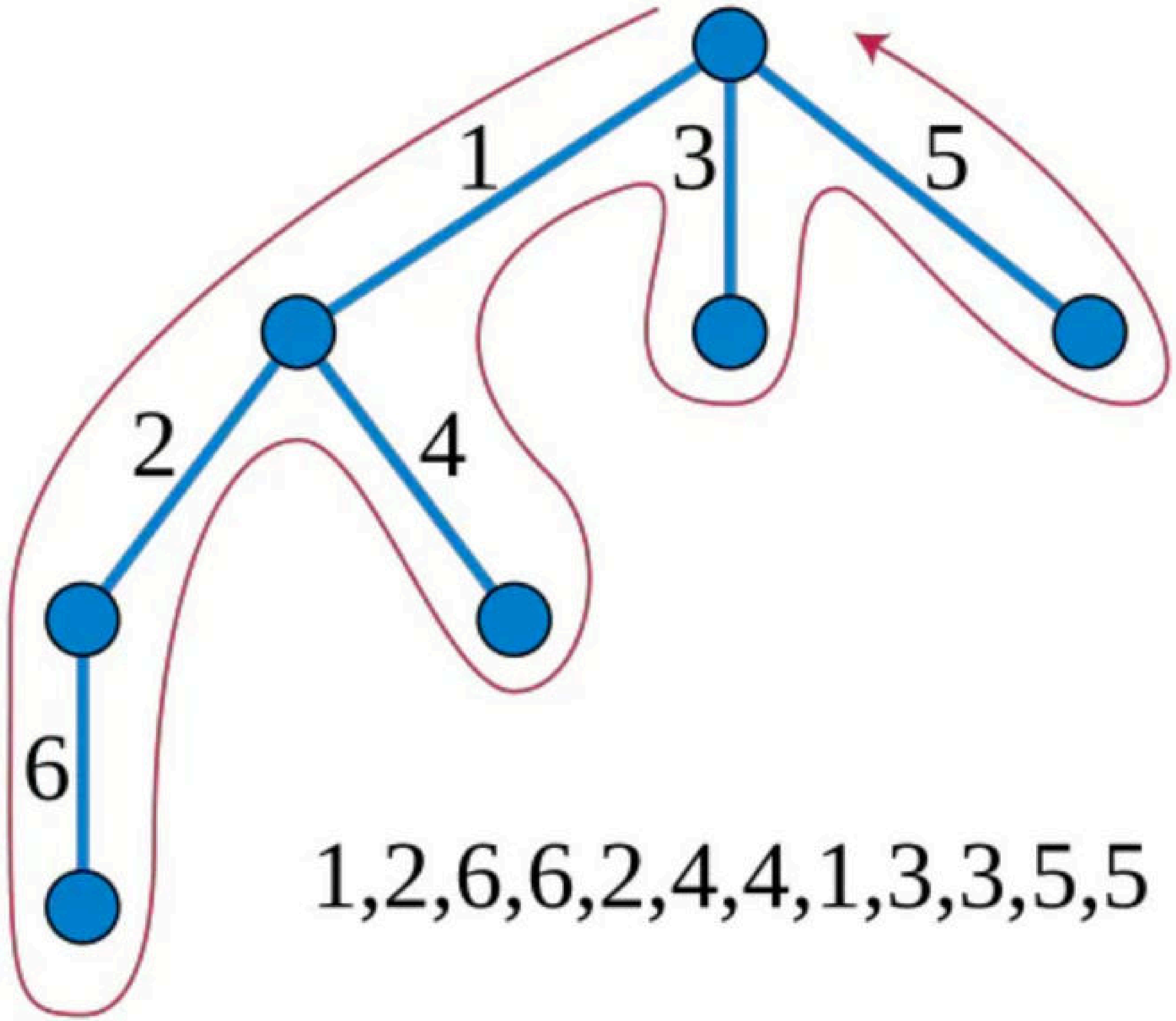

Image Source: https://en.wikipedia.org/wiki/Euler_tour_technique

Euler Tour Technique (ETT)

Way-1: Insert every node twice

- A subtree of node x is represented by the continuous range [start[x], end[x]]
- A path between two nodes A & B contains nodes which occur exactly once in the continuous range [End[A], Start[B]] -- Useful for applying MO's on Trees where we can ignore an element **y** if it occurs twice in the range [L, R].

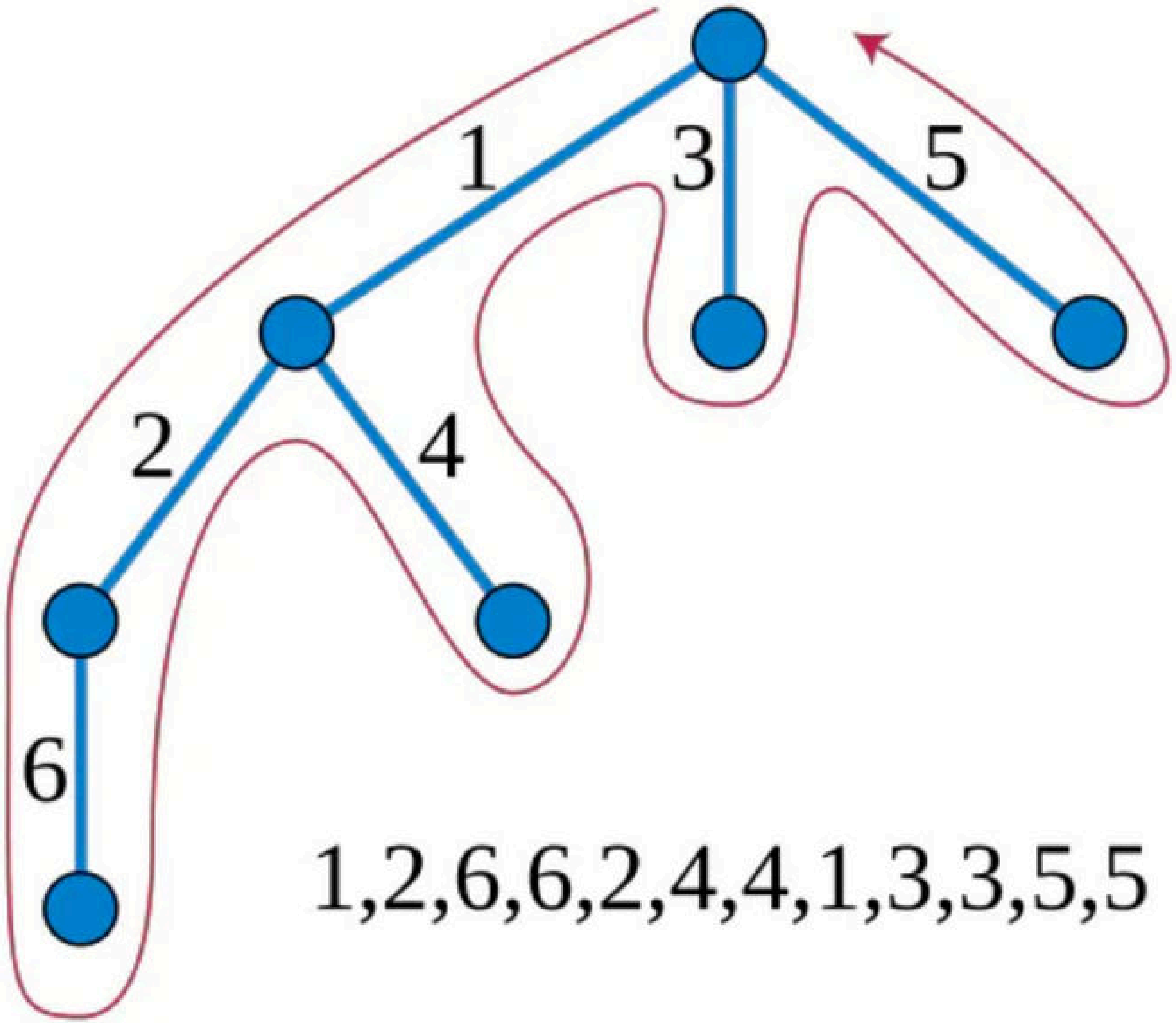

Image Source: https://en.wikipedia.org/wiki/Euler_tour_technique

Euler Tour Technique (ETT)

Way-2: Insert every node only once

- Insert every node/edge in the euler tour array whenever you enter the node and increment the timer.
- Start[x] = Time at which you enter the node x.
- End[x] = Time at which you exit the node x.
- All nodes in subtree of x occur exactly once in the range [Start[x], End[x]].

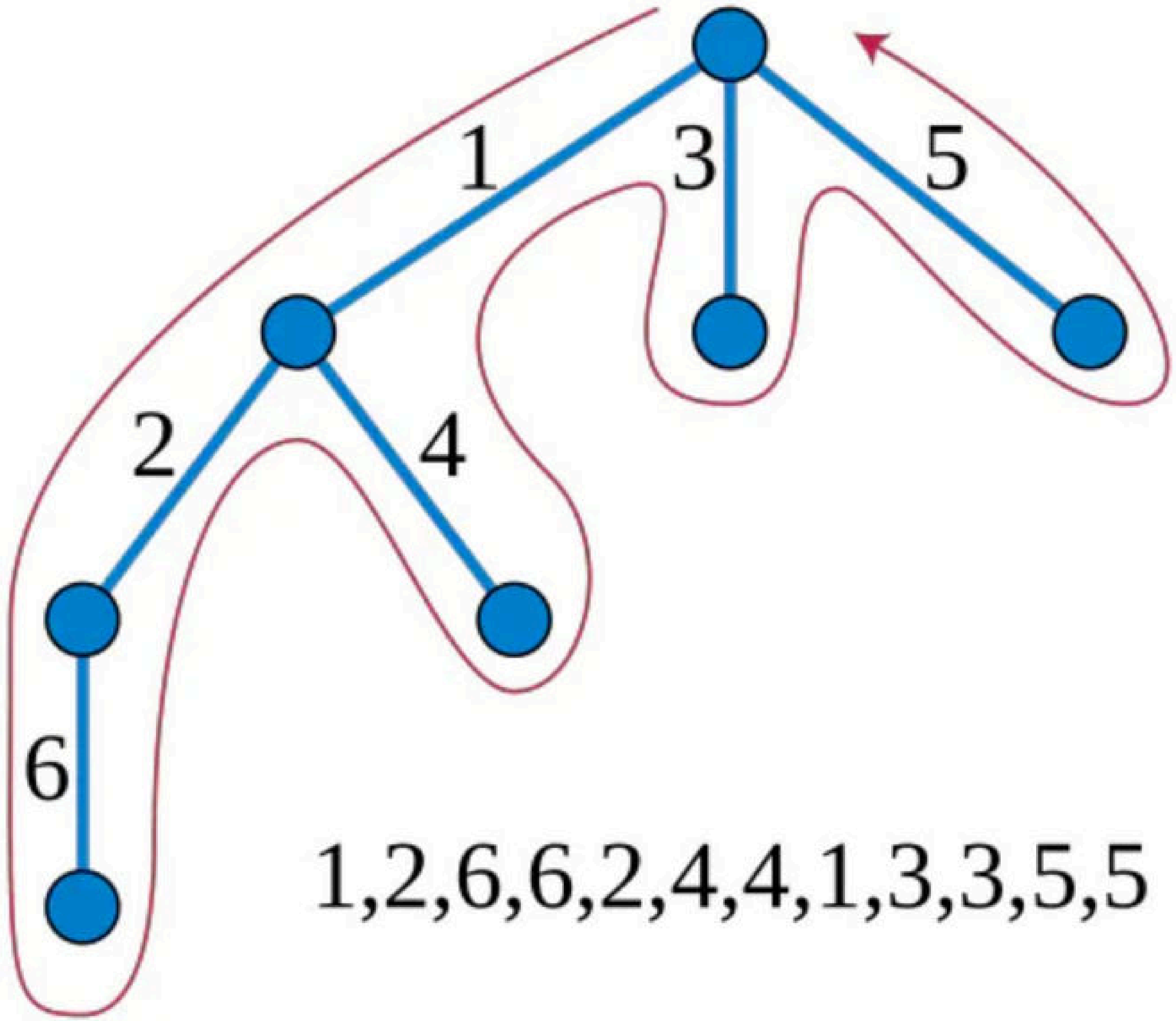

Image Source: https://en.wikipedia.org/wiki/Euler_tour_technique

ETT - Steps to support subtree updates / queries

- Build the Euler Tour array for the given tree by doing a DFS.
- Maintain a Data Structure on the Euler Tour array that supports range queries / updates
- For any query/update on all nodes in the subtree of node x, process it as a query/update on range [Start[x], End[x]] in the linearised array.

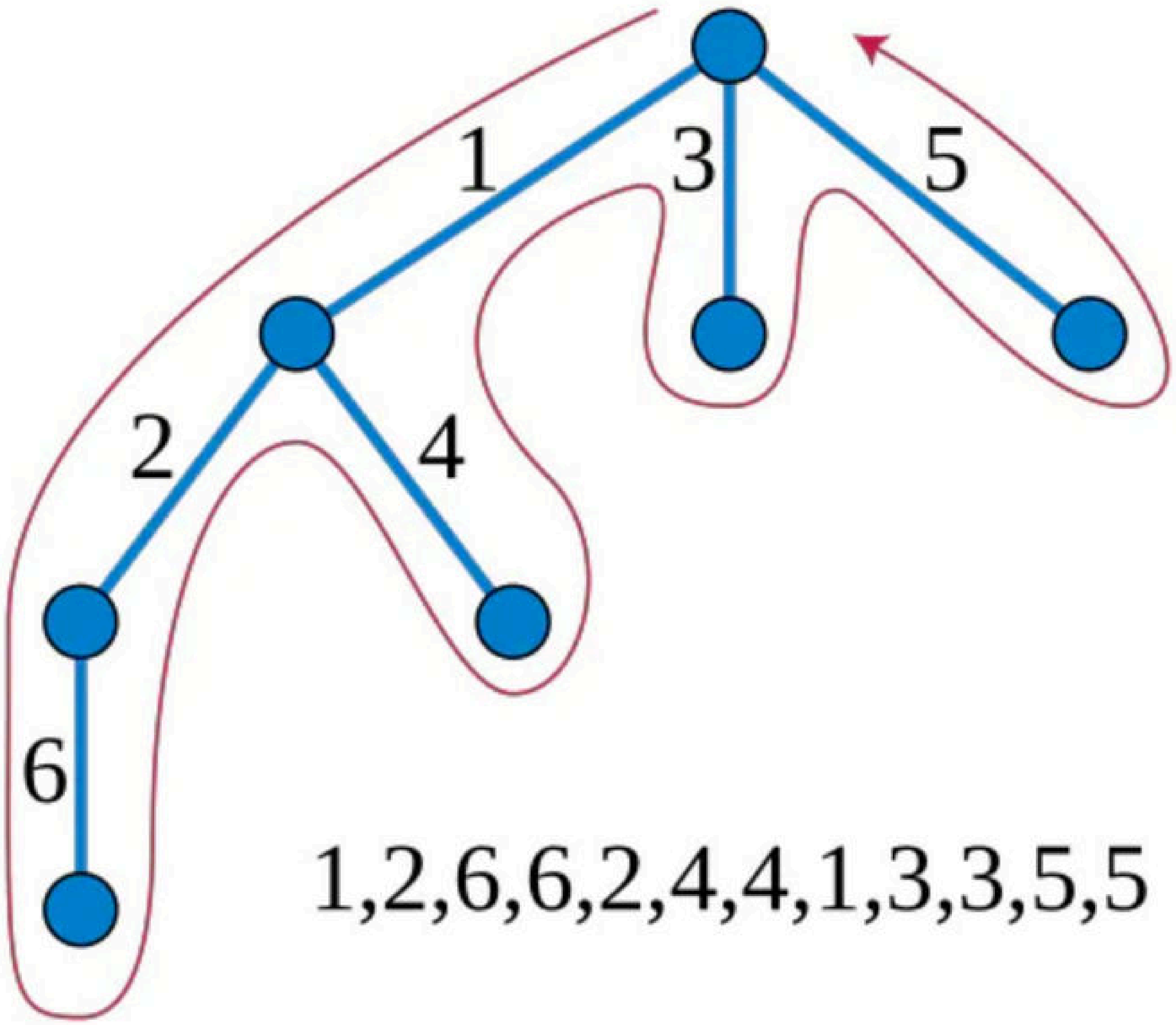

Image Source: https://en.wikipedia.org/wiki/Euler_tour_technique

Simultaneously maintaining Euler Tour & HLD

- **What if you wish to support path and subtree updates & queries together?**
- **Tl;Dr Simply add start[x]/end[x] computation to HLD DFS.**
- HLD is also a Way-2 Euler Tour Ordering
- In HLD, the "order" in which we call DFS on the children is decided by subtree sizes.
- But once that is done, we "add our node to the array" and "increase the timer" whenever we enter a node for the first time.
- Therefore, we can simply maintain the start[x] and end[x] times for every node x, similar to Way-2 of Euler Tour Technique.
- Now, any subtree **x** will be present as a linear range [start[x], end[x]]
- Any path from vertex A to chainHead[chainNo[A]] will be present as a linear range [start[chainHead[chainNo[A]]], start[A]] -- **exactly same as the usual HLD**

Simultaneously maintaining Euler Tour & HLD

You can now support simultaneous Queries and Updates of the form

- **Range Subtree Update:** Add val to all nodes in the subtree of node x
- **Range Path Update:** Add val to all nodes on path from node x to y
- **Range Subtree Query:** Return sum of all nodes in the subtree of node x
- **Range Path Query:** Return sum of all nodes on the path from node x to y

In O(log^2N) via HLD + Euler Tour + Segment Tree. :)

Sample Problem: https://www.hackerrank.com/challenges/subtrees-and-paths/problem

Other techniques for Queries on Trees

- Centroid Decomposition
  - https://tanujkhattar.wordpress.com/2016/01/10/centroid-decomposition-of-a-tree/
- Auxiliary Tree
  - See https://codeforces.com/problemset/problem/613/D + Editorial

Conclusion

- Queries on Trees is a vast topic and there are many tricks / data structures to support this.
- HLD & Euler Tour traversals + Segment Trees / Square Root Decomposition provide us a very powerful toolkit to answer different types of query on tree problems.