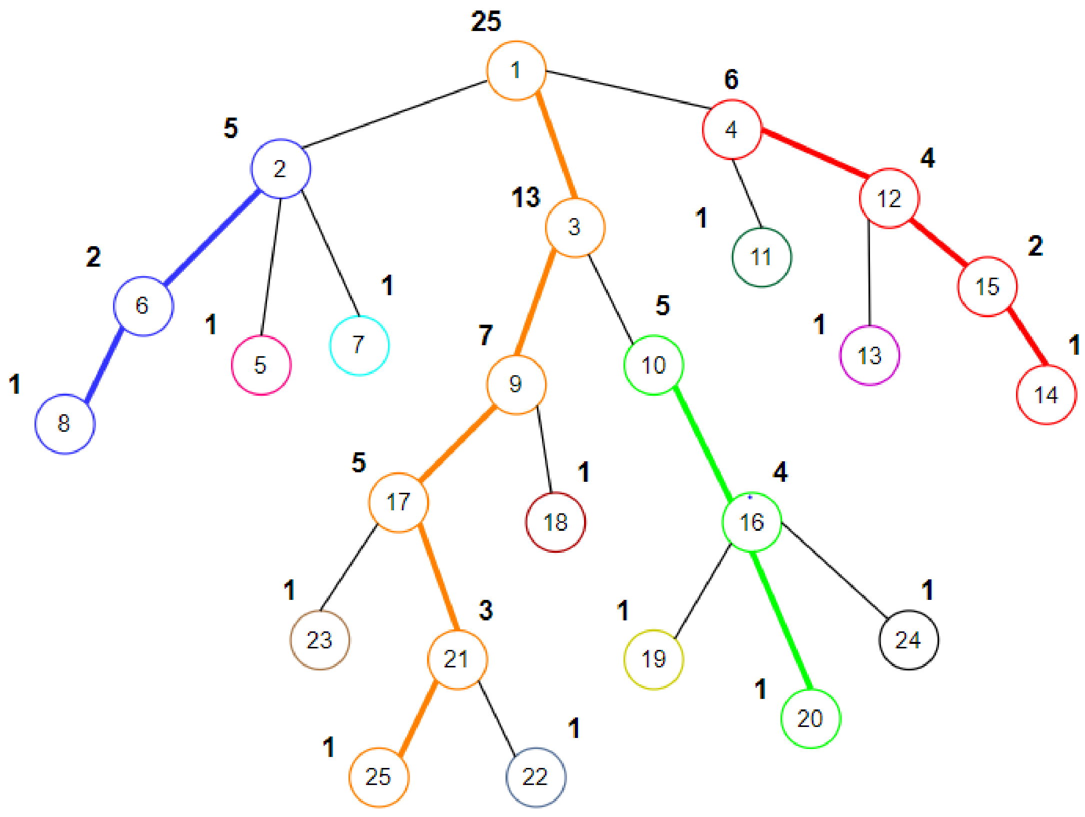
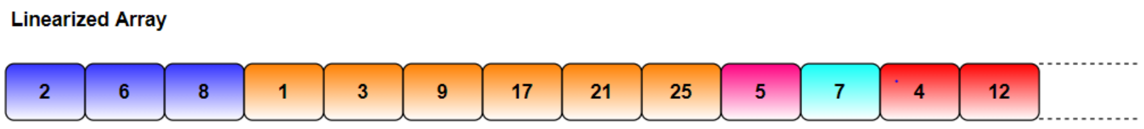
Handling Queries using Heavy-Light Decomposition

Now since we know the terminology, we have the same tree where each colored path basically represents a heavy path, where we have taken into consideration that every such heavy path is the longest path consisting of a collection of heavy edges.
We call each such colored path a **chain**.
Heavy-Light decomposition, as the name suggests, breaks the whole tree into **vertex-disjoint chains** from **top** to **bottom.** The leaf nodes which are not a part of any chain(colored path) form a chain of a single node(Each such leaf node is of a different color).



How many different chains can we have for a tree?
We can have N - 1 different chains for a tree consisting of N nodes in the worst case. **Example:** A star graph of N nodes.

These chains are useful for **linearizing** our tree and represent our tree in the form of an array. We pick every chain and represent it as a subarray of the array representing the tree, the order in which chains are picked does not matter.



Let us discuss some important properties regarding this decomposition of the tree into vertex-disjoint chains:

- Every light edge connects two different chains. In other words, a new chain starts after every light edge.
- Every vertex is a part of exactly one chain.
- Each chain forms a subarray in the **"linearized"** representation of the tree.

Now we know a way to linearize our tree into an array but why use this technique and what are its advantages?

The decomposition of a tree into vertex-disjoint chains ensures a very important property that the tree follows:
The subtree size reduces by at least half when a light edge is traversed.

We already know that a light edge connects two different chains, traversing a light edge i.e moving from one chain to another reduces the search space(the number of nodes) by at least half from top to bottom. Let us try to prove this claim by contradiction:

Consider a node having N nodes in its subtree rooted at this node, having a heavy child denoted by $N_{HC.}$
Let us assume there exists a light child denoted by N_{LC} of this node having > N/2 number of nodes in its subtree, so the number of nodes in the heavy child will be:
$N_{HC} = N - N_{LC}$
$N_{HC} < N / 2$ (As $N_{LC} > N / 2$)
But from the definition, the heavy child is the child with the largest subtree rooted at this child, hence by contradiction, we can say that traversing any light edge from a node will reduce the subtree size by at least half of the subtree size rooted at that node.

Hence from the above claim, we can say that **we can go from a given node x to any of its ancestors in the tree by changing at most $\log_2 N$ chains.** This is a result of the previous claim because changing a chain means traversing through a light edge and we know that as we traverse a light edge it reduces the subtree size by at least half on going from top to bottom, hence doubling the subtree size on going from bottom to top.

Let us say we change **'c'** chains as we move up from a given node **x** to one of it's ancestors, then each such change in the chain **doubles** the subtree size. So in the worst case, we will move up until the subtree size becomes equal to the total number of nodes in the tree, hence:
$2^c <= N$, where N is the total number of nodes in the tree

$c <= \log_2 N$

So if we put the above statement for the linearized array representing the tree, we can go from a given node x to any of its ancestors by changing at most **$\log_2 N$ disjoint [L, R] segments** representing different chains.

Path in a Tree
A path between two nodes **'a'** and **'b'** can be represented as the path from **'a'** to **LCA(a, b)** and from **LCA(a, b)** to **b**, where LCA(a, b) represents the lowest common ancestor of nodes 'a' and 'b'. Hence a path between any pair of nodes can be traversed by changing at most **$2*O(\log_2 N)$ chains**.

Handling Queries

Since we can represent any path into at most $2*\log_2 N$ different chains, we can build an efficient data structure over each such chain to calculate the answer to the query, like segment trees or some other data structure.
For Example:
If we need to find the maximum value on the path between nodes a and b, we can build a segment tree over every chain in our heavy-light decomposed tree. In the worst case, we will traverse at most $2*\log_2 n$ different chains.
Then we can find the maximum element for each of the chains representing the path from a to b with the help of a segment tree in $\log_2 n$ time, where n is the number of elements present in a chain. The maximum of all the chains will be the answer to our query.
Hence the overall complexity to find the **maximum** will be **$O(\log_2 N * \log_2 N)$** in the worst case.

Let us generalize the technique of heavy-light decomposition:

- Decompose the given tree into vertex-disjoint chains using heavy-light decomposition.
- Linearize the chains into an array, with each chain representing a subarray of the array representing the tree and build a data structure(For Example: Segment trees) that supports range updates and queries.
- For any given query involving some computation(sum, min, max etc) or some updates on the nodes on the path between a pair of nodes 'a' and 'b', process the query as a combination of different chains / different [L, R] segments in the array, which will take $O(\log_2 N)$ time as we go from 'a' -> LCA(a, b) -> 'b'.
- Hence the total time taken to process a query will be $O(\log_2 N$ * Time taken by DS built over the chain to answer the query).