DFS on Trees

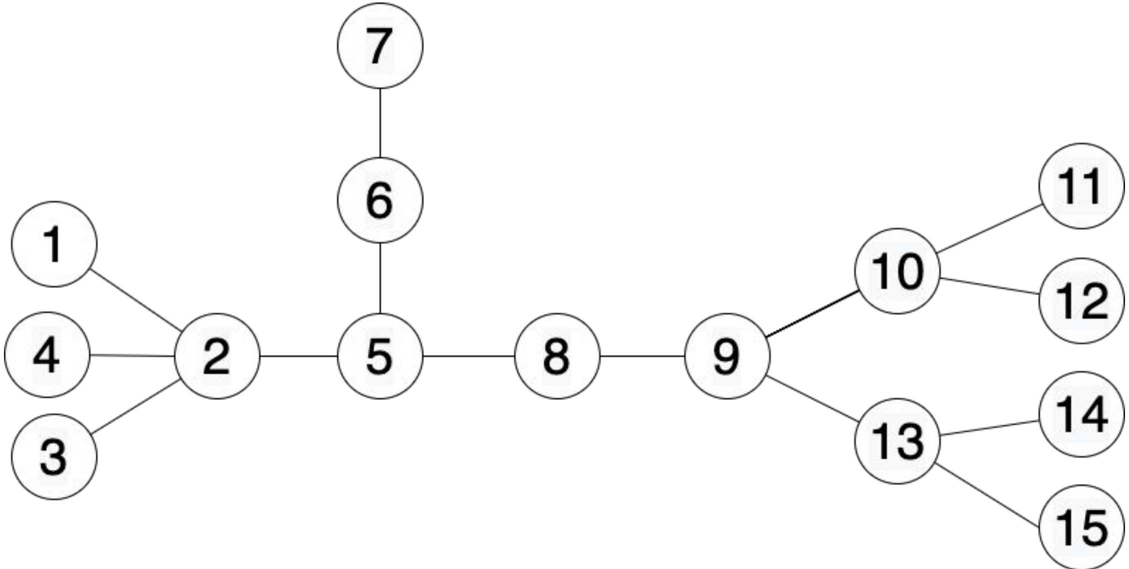
tanujkhattar@

Objective

- Introduction to Trees
  - Definition & Examples
  - Properties of Trees
- DFS on Trees
  - DFS on Rooted Trees
  - DFS on Unrooted Trees
- Path Queries on Trees
  - LCA + Prefix Sums from root → node.
- Path Updates on Trees
  - LCA + Subtree Sum after lazy updates.
- Conclusion

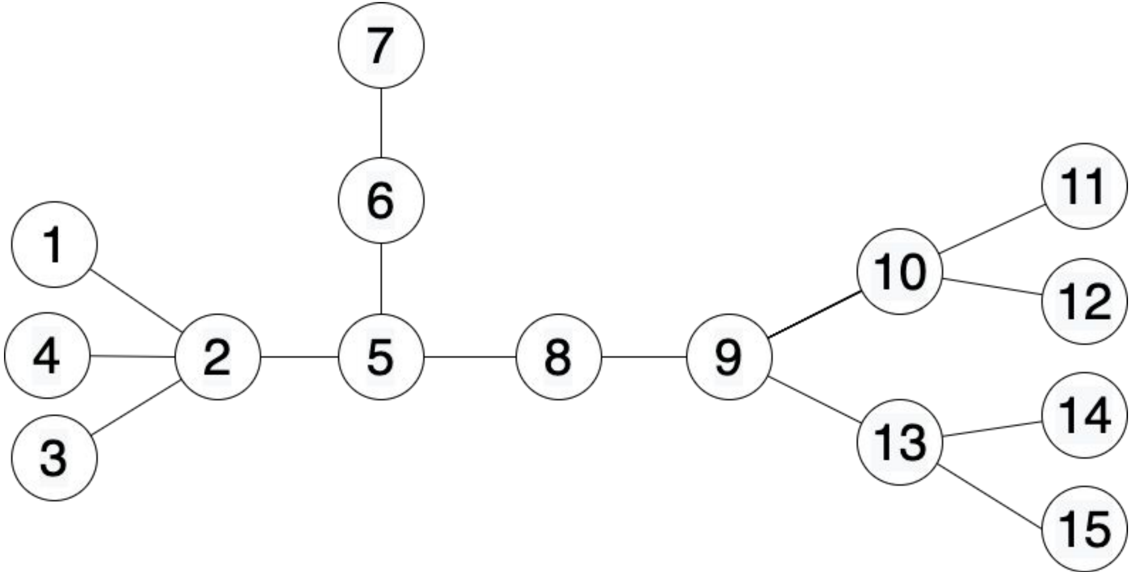
Introduction to Trees

- **Tree** is an undirected connected graph without cycles.
- **Forest** is a collection of many independent trees.



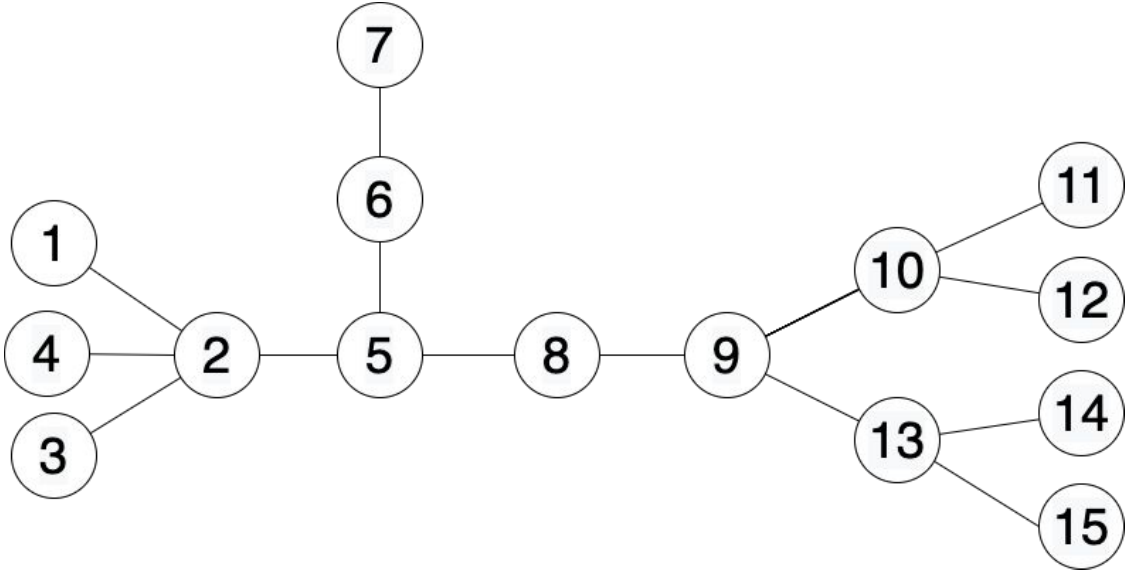
Properties of a Tree

- For any two nodes x, y in the tree, there exists a unique simple path connecting x & y in the tree.
  - Since the tree is connected, there should exist at least one simple path.
  - If more than one path exists, then there will be a cycle -- contradiction.



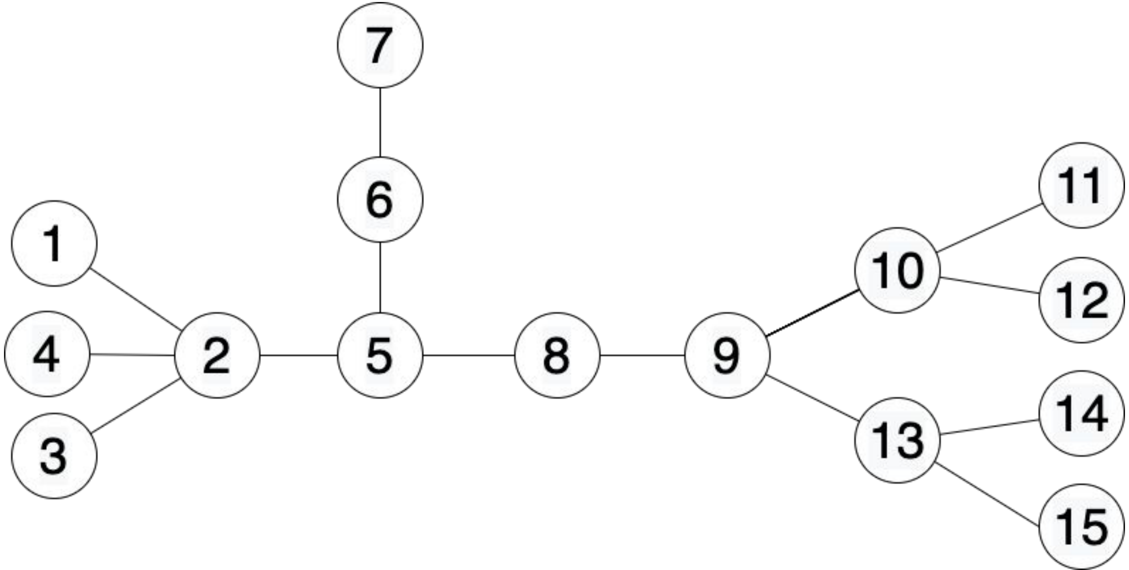
Properties of a Tree

- A tree with N nodes has N - 1 edges.
  - Removing any edge (u, v) disconnects the tree into two parts M1 & M2 of size < N.
  - By induction, total edges = (M1 - 1) + (M2 - 1) + 1 = N - 1.



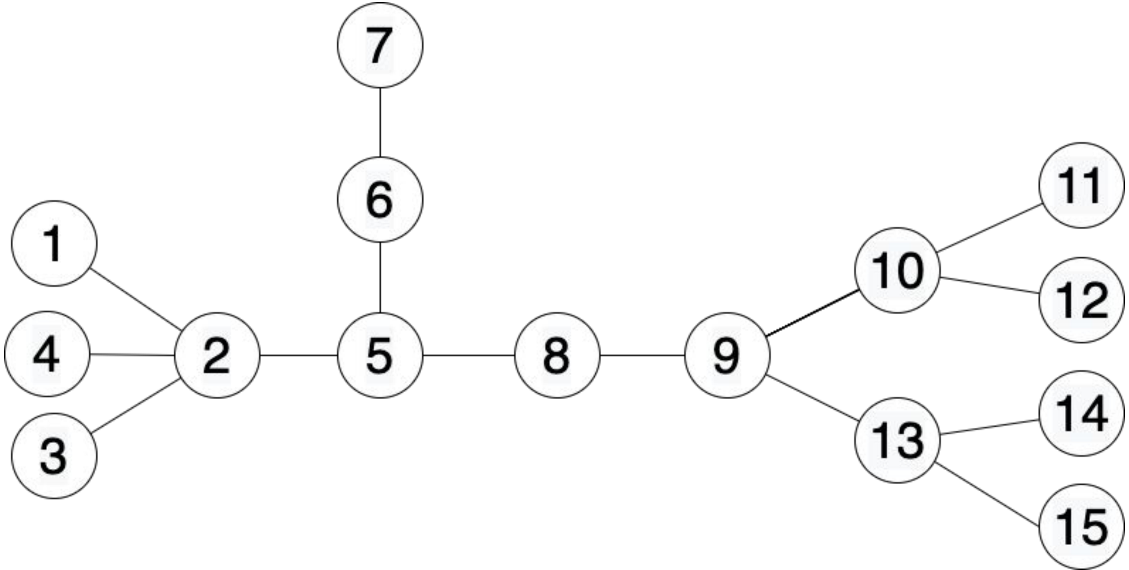
Properties of a Tree

- **Internal Nodes:** Nodes with degree > 1.
- **Leaf Nodes:** Nodes with degree = 1.



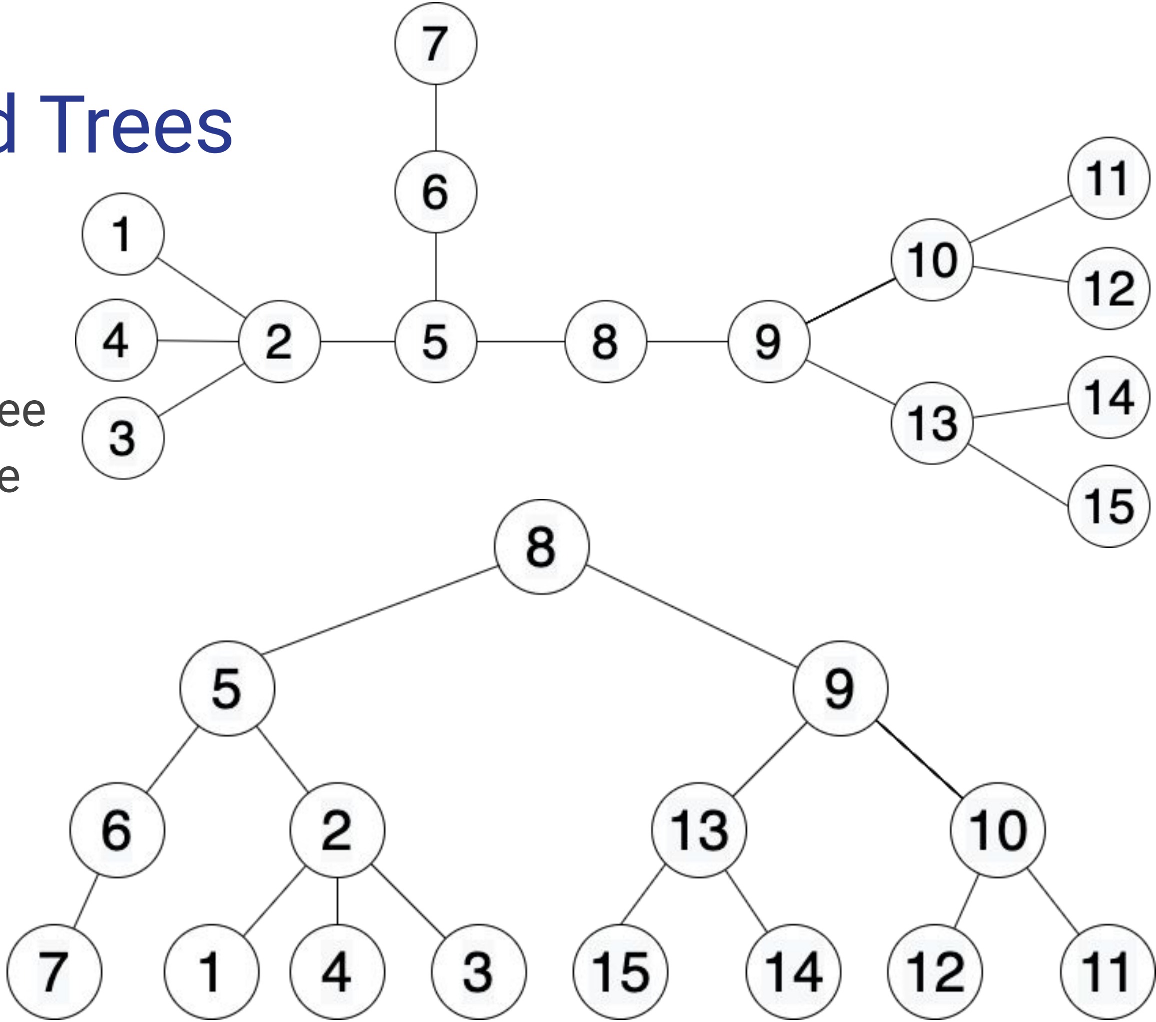
Properties of a Tree

- **Weighted Tree:** A tree in which edges have weights
- **Unweighted Tree:** A tree in which don’t have weight.
    - Assume edge weights are 1 to compute distance etc.



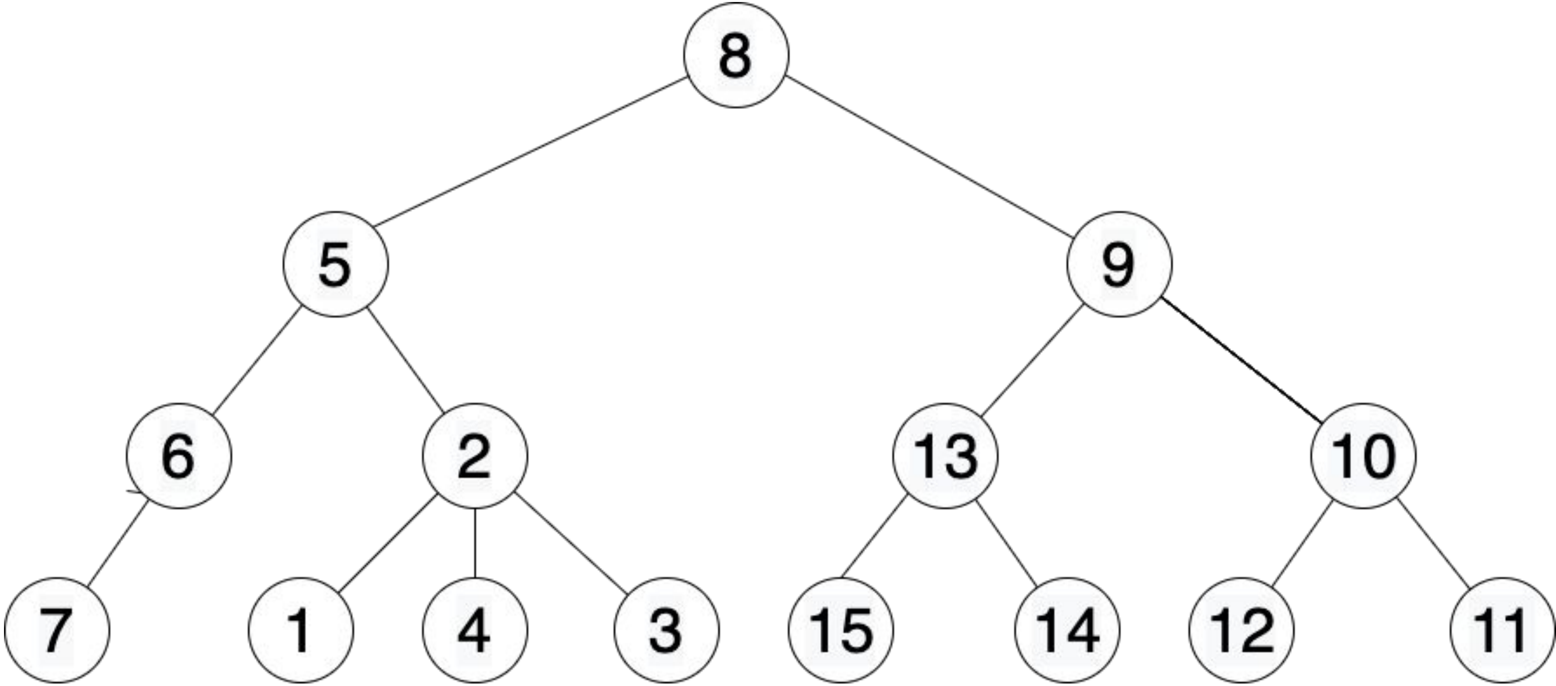
Rooted vs Unrooted Trees

- **Unrooted Tree** is like a tree lying flat on the table.
- The process of **Rooting** a tree is like picking up the flat tree from the **root** node and hanging it on the wall.



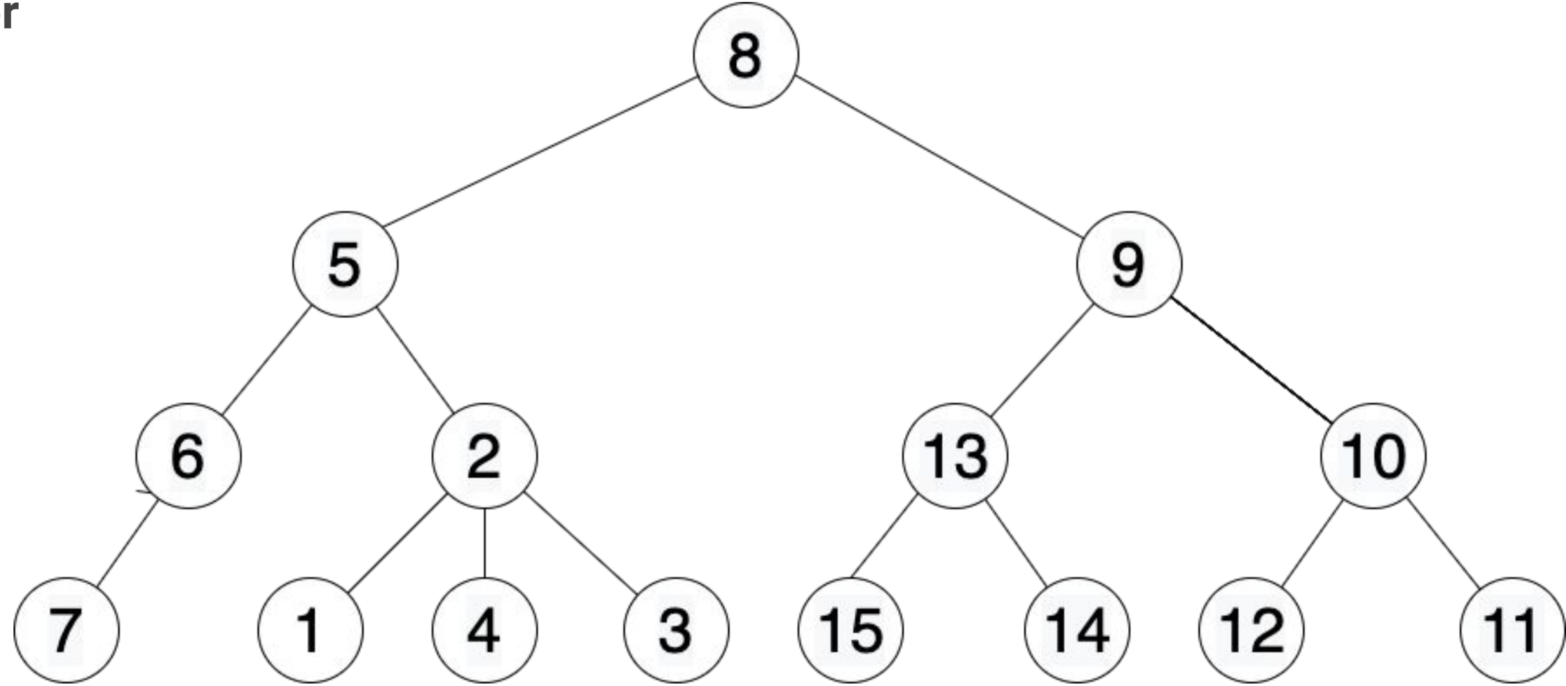
Properties of a Rooted Tree

- **Parent**



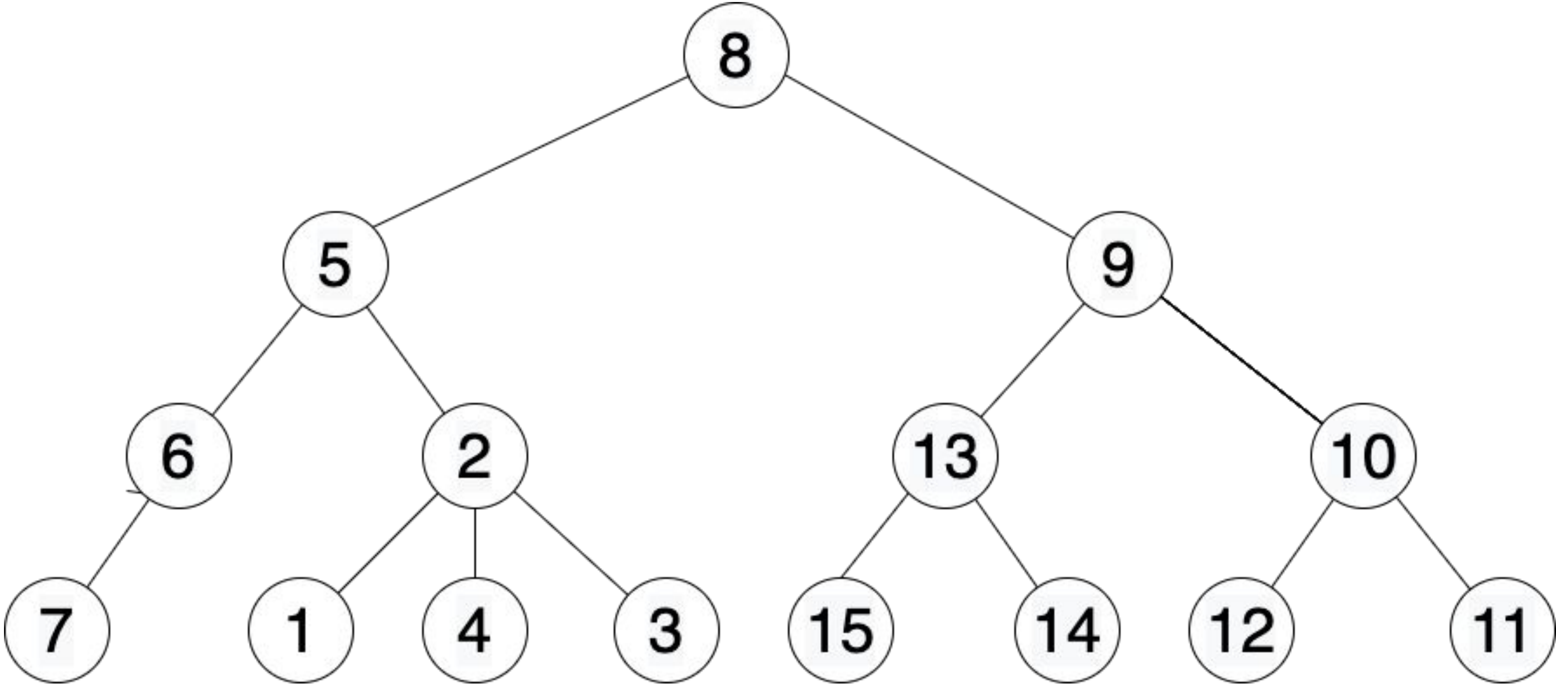
Properties of a Rooted Tree

- **Ancestor**



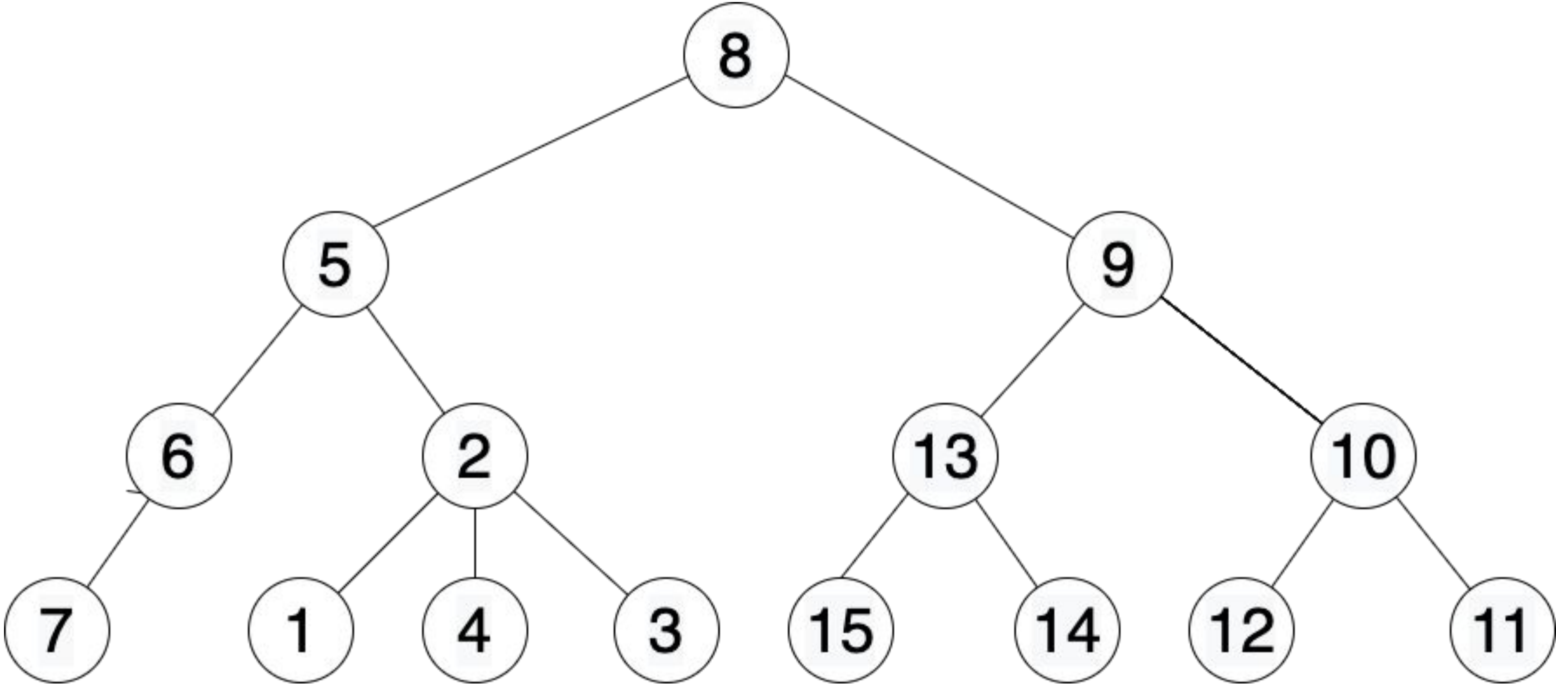
Properties of a Rooted Tree

- **Subtree**



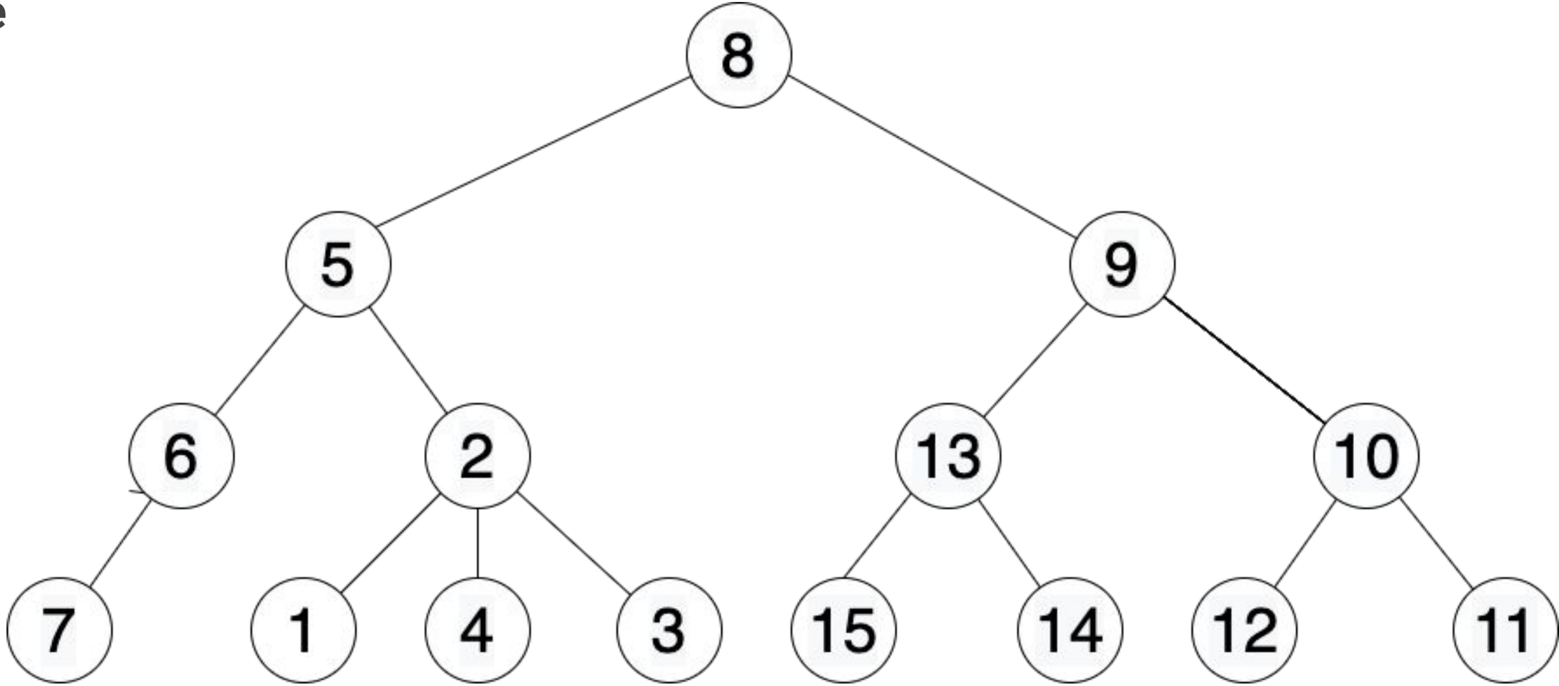
Properties of a Rooted Tree

- **Level**



Properties of a Rooted Tree

- **Distance**



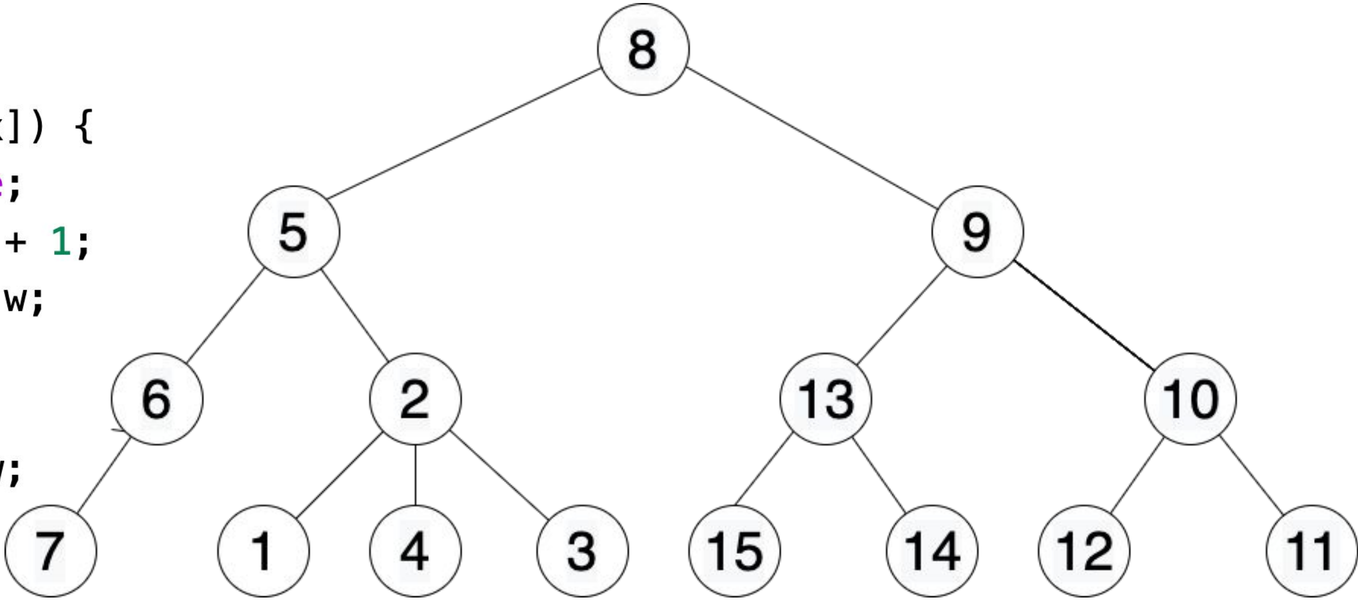
Problem - 1 : DFS on Rooted Trees.

- **Given a weighted tree with N nodes and node “root”, hang the tree on the root node and for every other node, compute**
  - **Parent**
  - **Level**
  - **Distance from root**
  - **Subtree Size**
  - **Subtree Sum**
- Easier version: https://cses.fi/problemset/task/1674

Solution - 1: DFS on Rooted Trees

- Assume edges are directed from top to bottom.
- Therefore, while exploring any node, you visit all its children and then return.

```
void dfs(int x, int p) {
  par[x] = p;
  sz[x] = 1;
  for (auto [y, w] : g[x]) {
    if (y == p) continue;
    level[y] = level[x] + 1;
    dist[y] = dist[x] + w;
    dfs(y, x);
    sz[x] += sz[y];
    sum[x] += sum[y] + w;
  }
}
```



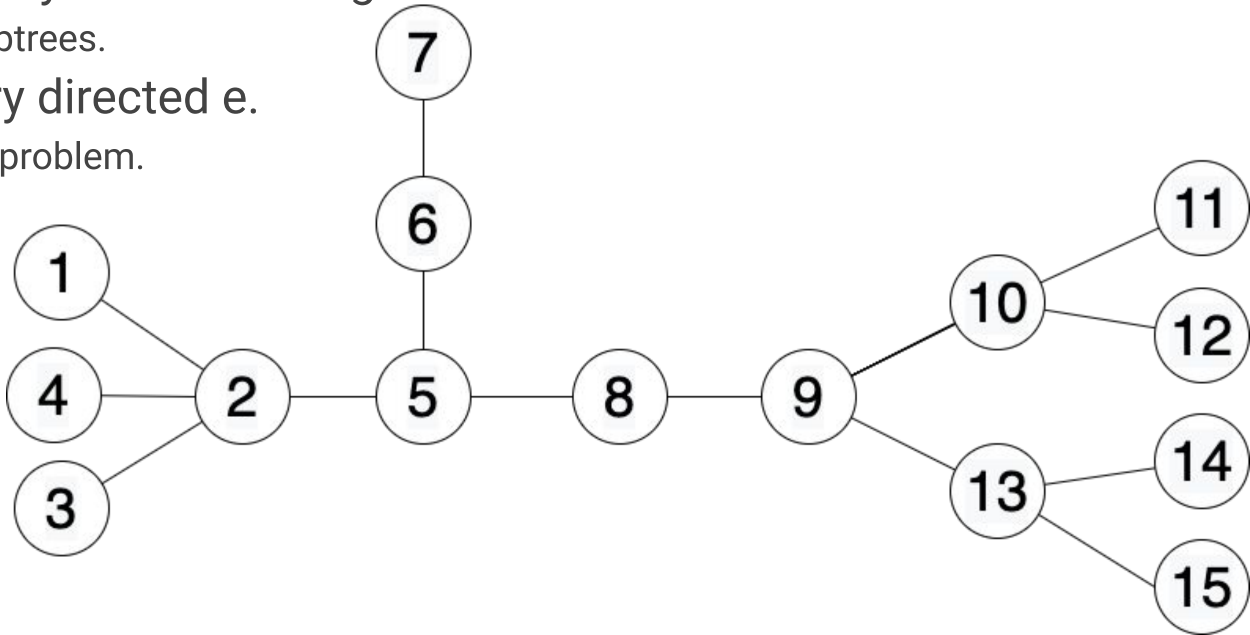
Problem - 2 : DFS on Unrooted Trees

- **Given an unrooted weighted tree, for every node find the maximum distance to another node in the tree.**

- Easier Version: https://cses.fi/problemset/task/1132

Solution - 2: DFS on Unrooted Trees

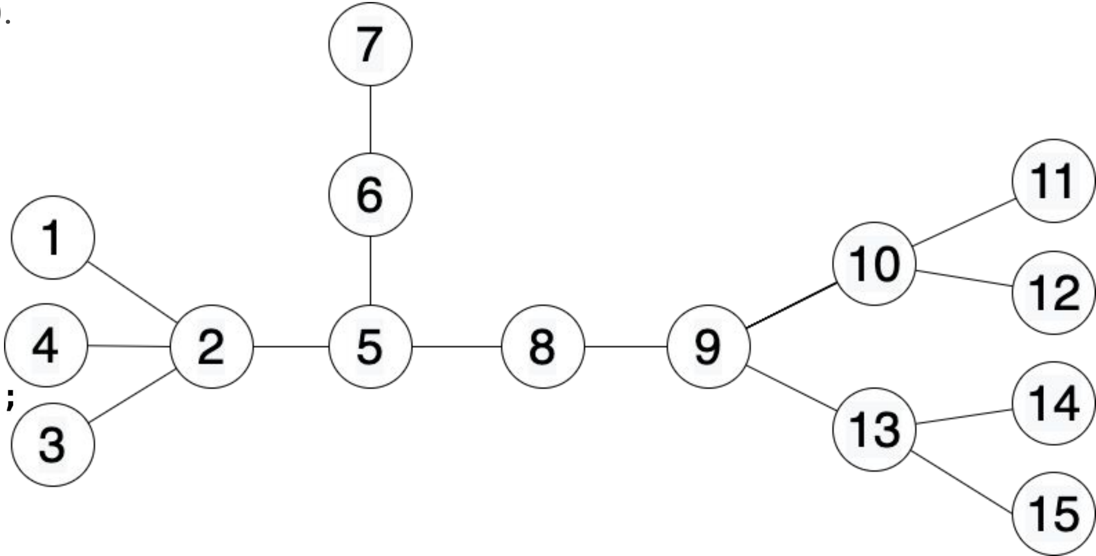
- Replace every edge e with two directed edges (2 * e) & (2 * e + 1);
  - Edge opposite to e $\rightarrow$ e ^ 1.
- A “subtree” is identified by a directed edge.
  - Total 2 * (N - 1) such subtrees.
- Compute DP[e] for every directed e.
  - DP[e] = FAR[e] for given problem.



Solution - 2: DFS on Unrooted Trees (Way-1)

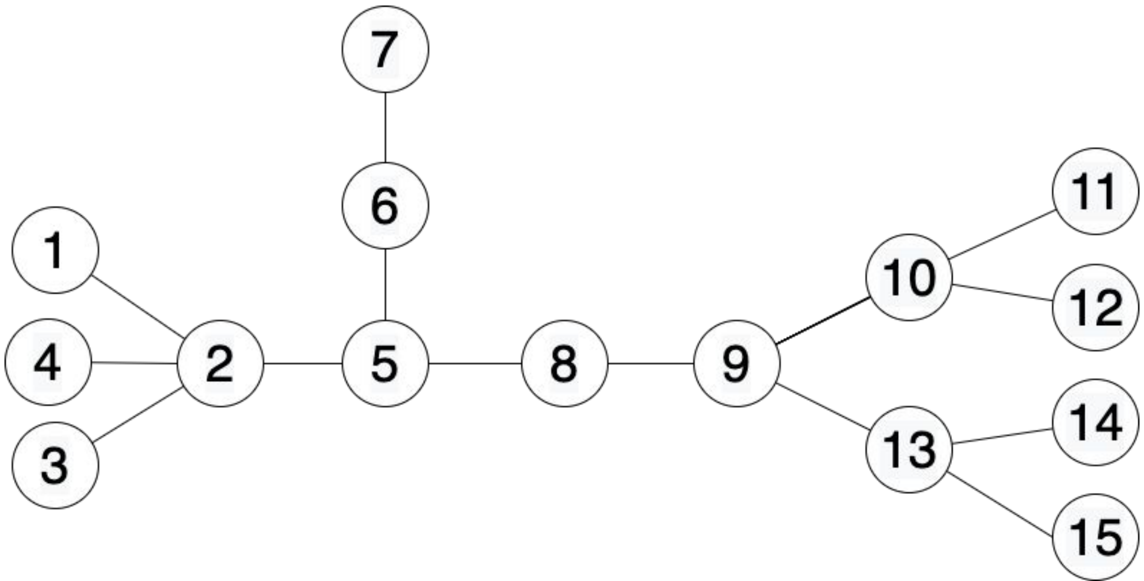
- How to compute DP[e] for every e?
- **Way-1 (Naive):**
  - Visit every edge e once. -- O(N)
  - Iterate on adjacency list of V[e] and compute DP[e] -- O(deg[V[e]])
  - Total?? -- O(N ^ 2) (eg: star graph).

```
void dfs_brute_force(int xe) {
  if (vis[xe]) return;
  vis[xe] = 1;
  for (auto ye : g[V[xe]]) {
    if (ye != (xe ^ 1)) {
      dfs(ye);
      far[xe] = max(far[xe], far[ye] + W[ye]);
    }
  }
}
```



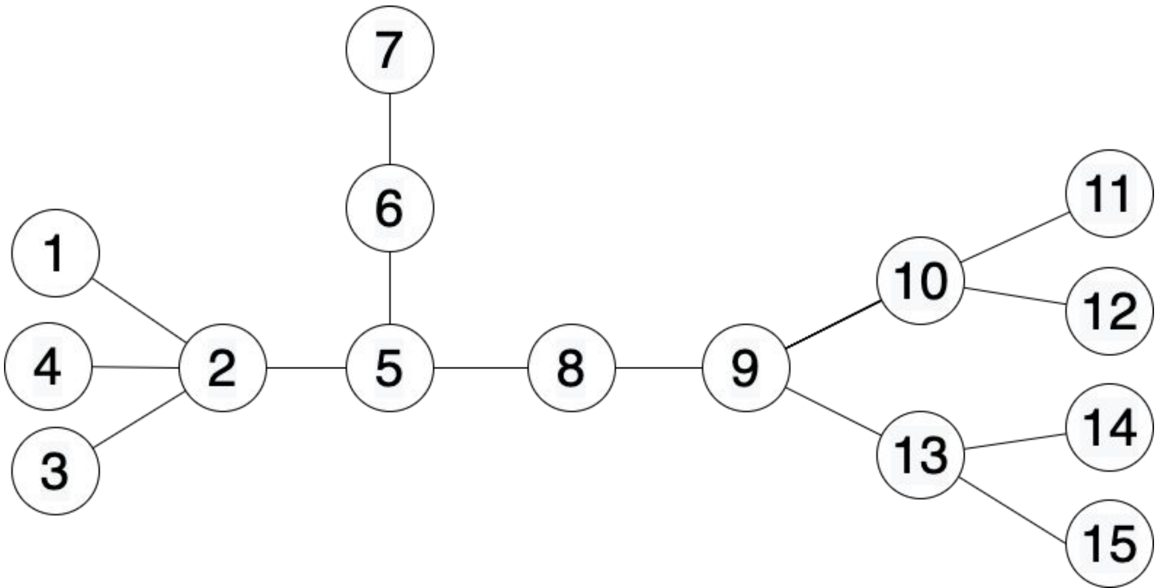
Solution - 2: DFS on Unrooted Trees (Way-2)

- For every vertex v, maintain some DS s.t. you can efficiently combine DP[e] for all e except \`pe\`.
  - Eg: Prefix/Suffix max for far[e].



Solution - 2: DFS on Unrooted Trees (Way-2)

- When you visit a vertex for the first time, all except 1 outgoing edge would get explored!
  - Next time you visit the vertex, remember this and only explore the 1 unexplored edge.



Solution - 2: DFS on Unrooted Trees (Way-2)

- For every vertex v, maintain some DS s.t. you can efficiently combine DP[e] for all e except `pe`.
  - Eg: Prefix/Suffix max for far[e].
- When you visit a vertex for the first time, all except 1 outgoing edge would get explored!
  - Next time you visit the vertex, remember this and only explore the 1 unexplored edge.
- Once all adjacent edges have been explored, add all values to DS and use them to compute DP[pe] for every subsequent DFS call to (V[pe], pe).
  - The DS should work in sublinear time s.t. you don't spend O(deg[V[pe]]) for every pe.
  - For majority cases, computing prefix & suffix functions should be enough! -- works in O(1).

Solution - 2: DFS on Unrooted Trees (Way-2) Code

```
// Representing an unrooted weighted tree
// using an edge list, we 2 directed edges
// for every edge. pos[e] represents
// position of e in the edge list of U[e].
int U[M], V[M], W[M], pos[M];
vector<int> g[N];
void add_edge(int e, int x, int y, int w) {
  U[e] = x;
  V[e] = y;
  W[e] = w;
  pos[e] = g[x].size();
  g[x].push_back(e);
}
```

```
// DFS on UnRooted Tree.
void dfs(int x, int pe) {
  if (!vis[x]) {
    // Visiting x for the first time.
    // Explore all edges except parent edge.
    vis[x] = (pe ^ 1);
    for (auto e : g[x])
      if (e != vis[x]) dfs(V[e], e);
    recompute_prefix_suffix_max(x);
  } else if (vis[x] > 1) {
    // Visiting x for the second time.
    // Only 1 outgoing edge would be unexplored.
    dfs(V[vis[x]], vis[x]);
    vis[x] = 1;
    recompute_prefix_suffix_max(x);
  }
  if (pe) {
    int idx = pos[pe ^ 1];
    far[pe] = max(val_or_zero(prefix_max[x], idx - 1),
                  val_or_zero(suffix_max[x], idx + 1)) +
              W[pe];
  }
}
```

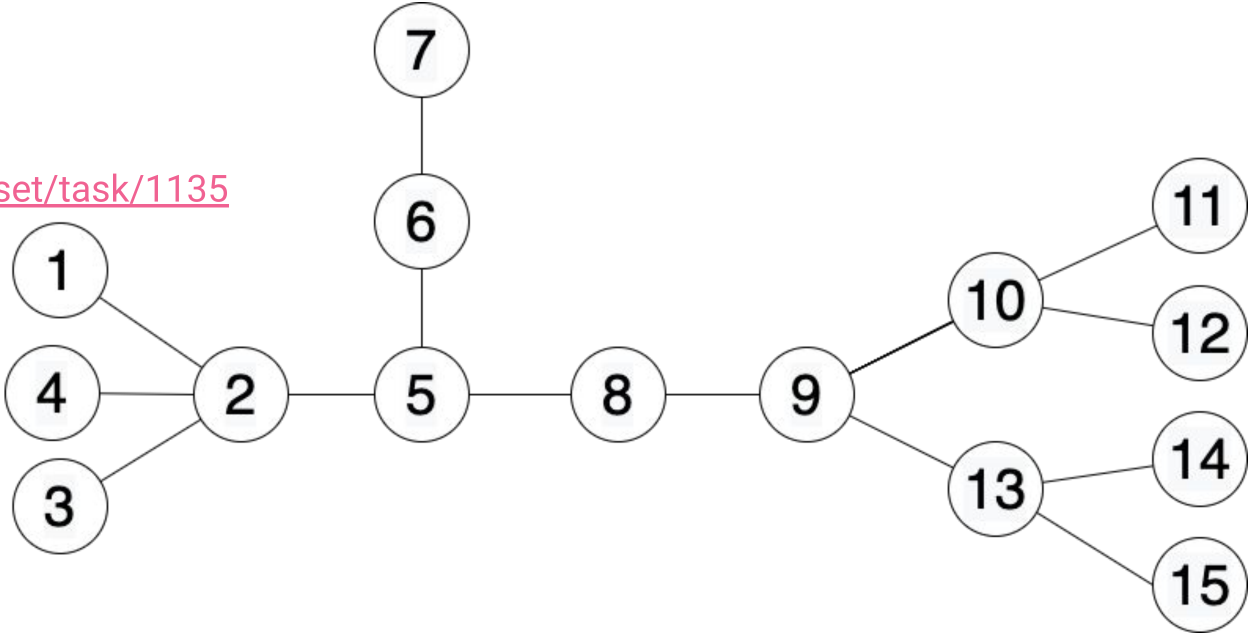
DFS on Unrooted Trees

- This is also called “**Rerooting Technique**”
    - You can choose the optimal root / answer later once you have computed the function (DP) value across every edge.
- More practice problems:
    - https://codeforces.com/contest/219/problem/D
    - https://codeforces.com/contest/1187/problem/E

Path Queries on a Trees

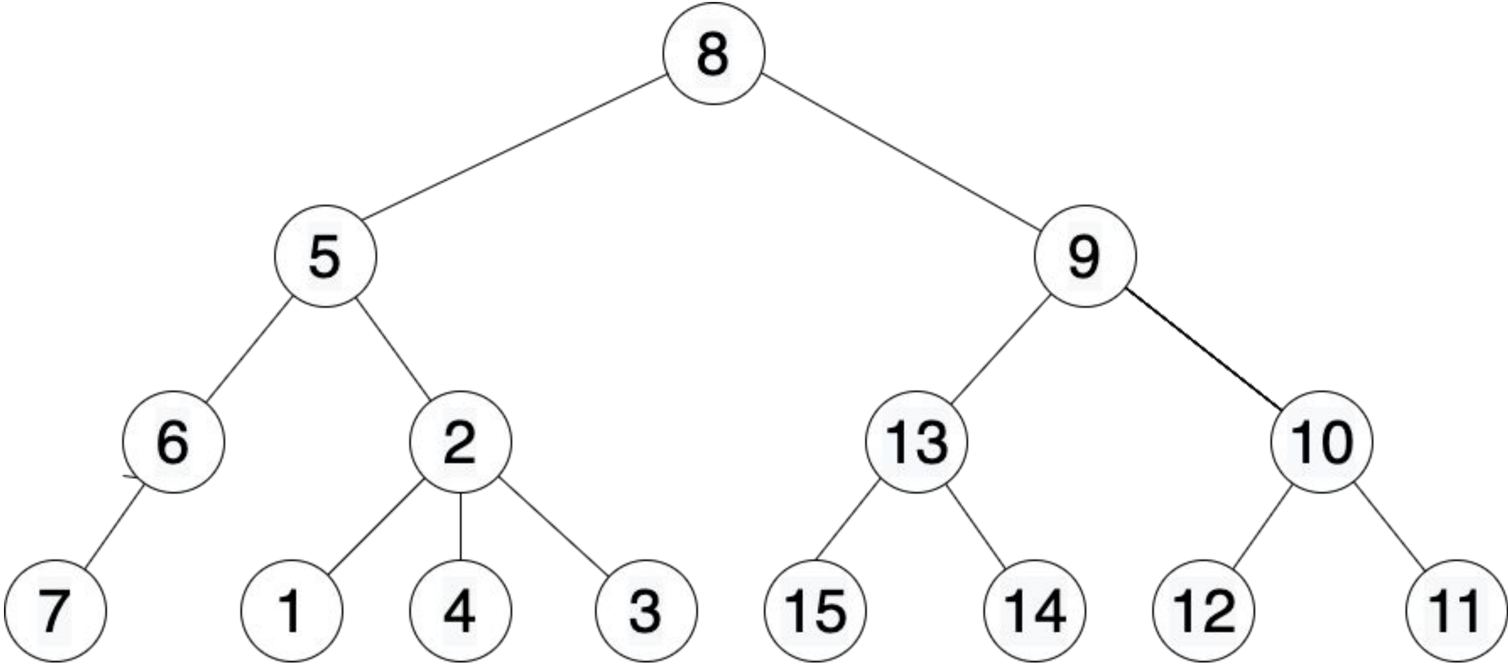
- Given a tree with N nodes and Q queries of the form:
  - x, y -- Find sum of edge weights between x & y.

- Practice Problem:
  - https://cses.fi/problemset/task/1135



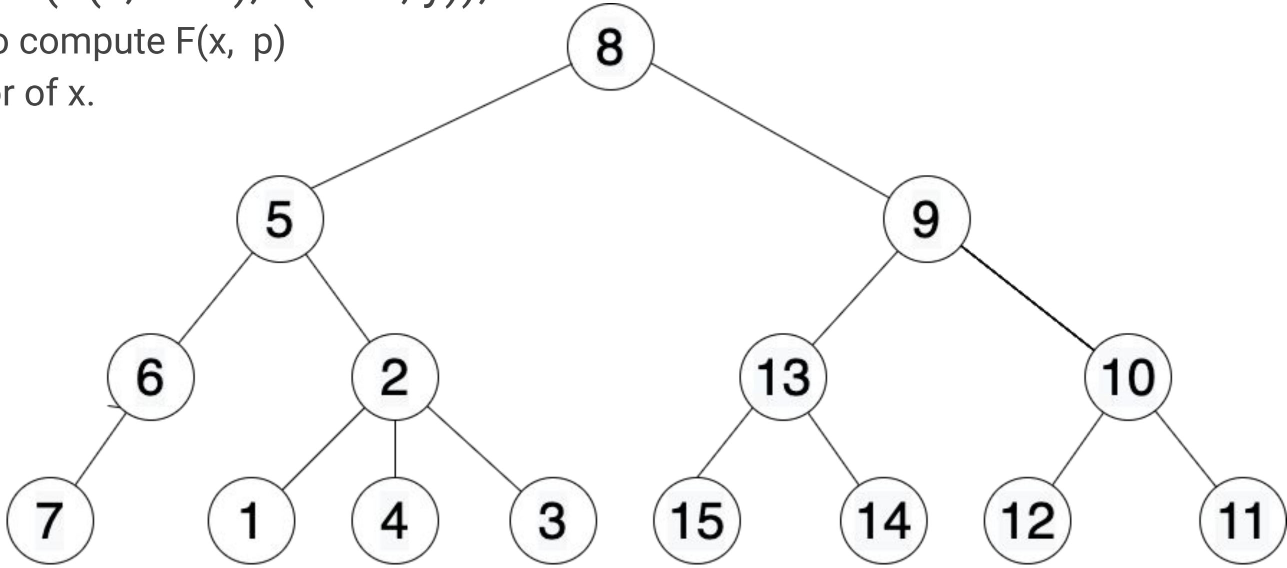
Path Queries -- Brute Force Solution

- For every query (x, y); do a DFS from x and return the dist[y].
- Eg: https://acm.timus.ru/problem.aspx?space=1&num=1471



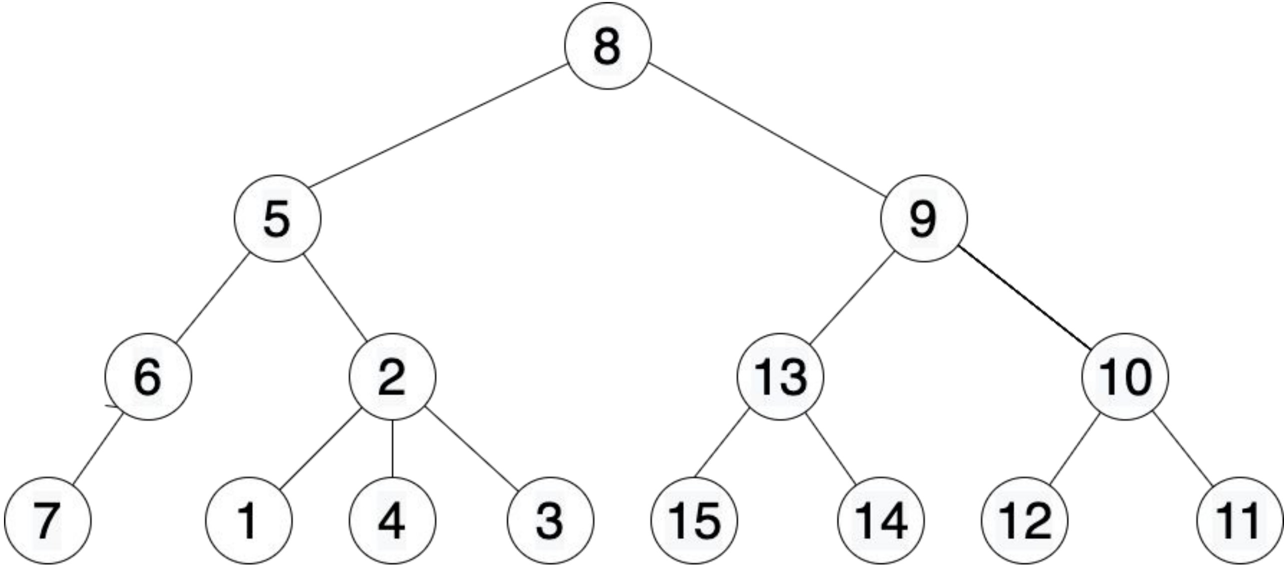
Lowest Common Ancestor (LCA)

- Hang the tree on any arbitrary node.
- Any path x → y can be reduced into union of two paths:
  - (x → LCA) && (LCA → y).
- Thus, F(x, y) = Combine(F(x, LCA), F(LCA, y));
  - We only need a way to compute F(x, p) where p is an ancestor of x.



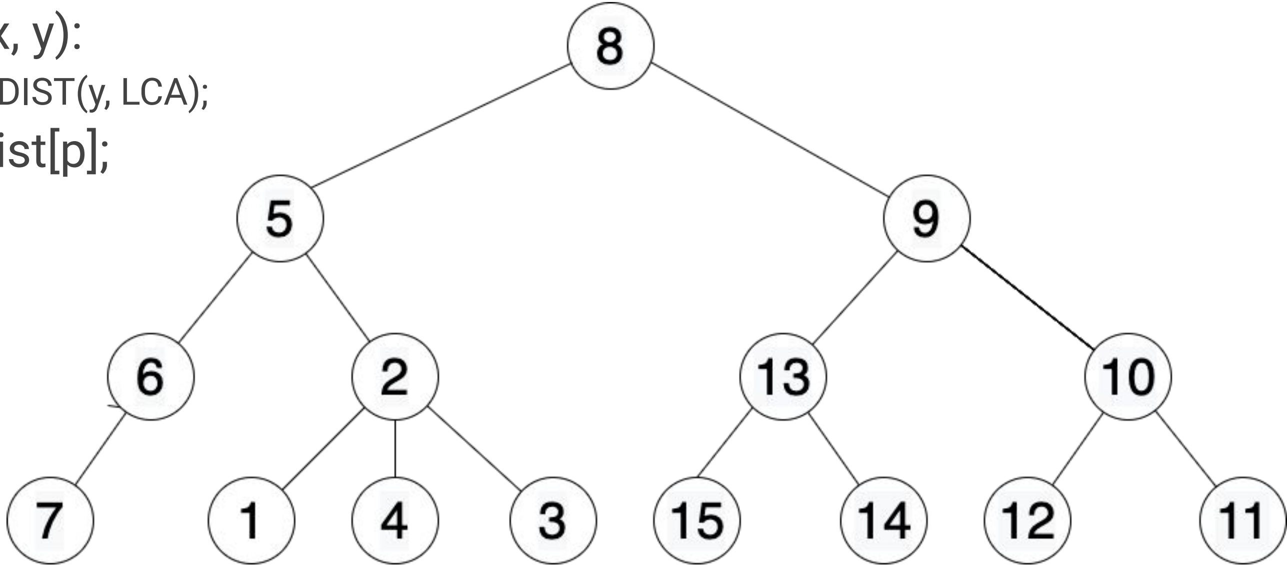
How to find LCA ?

- Brute Force Approach
  - Keep going up from x & y till they meet at a common point.
  - Always go up from the deeper node.
- We will discuss more efficient ways in the next class!



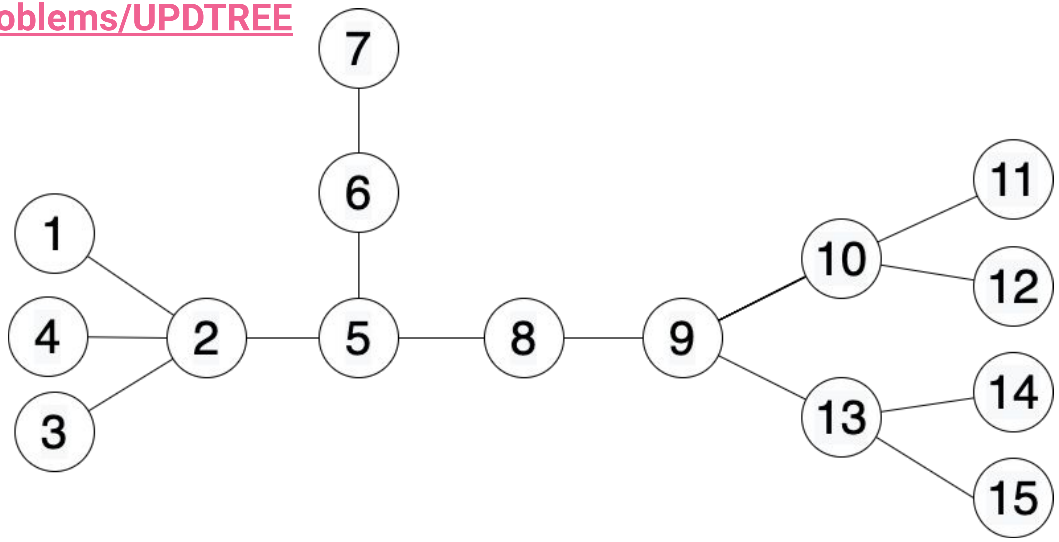
Path Queries via LCA & Prefix Sums on Trees

- **Question:** Given a tree with N nodes and Q queries of the form:
  - x, y -- Find sum of edge weights between x & y.
- For every node x, compute dist[x] = distance of node x from the root.
  - This is done using DFS on rooted trees as seen earlier.
- For any given query (x, y):
  - return DIST(x, LCA) + DIST(y, LCA);
- DIST(x, p) = dist[x] - dist[p];



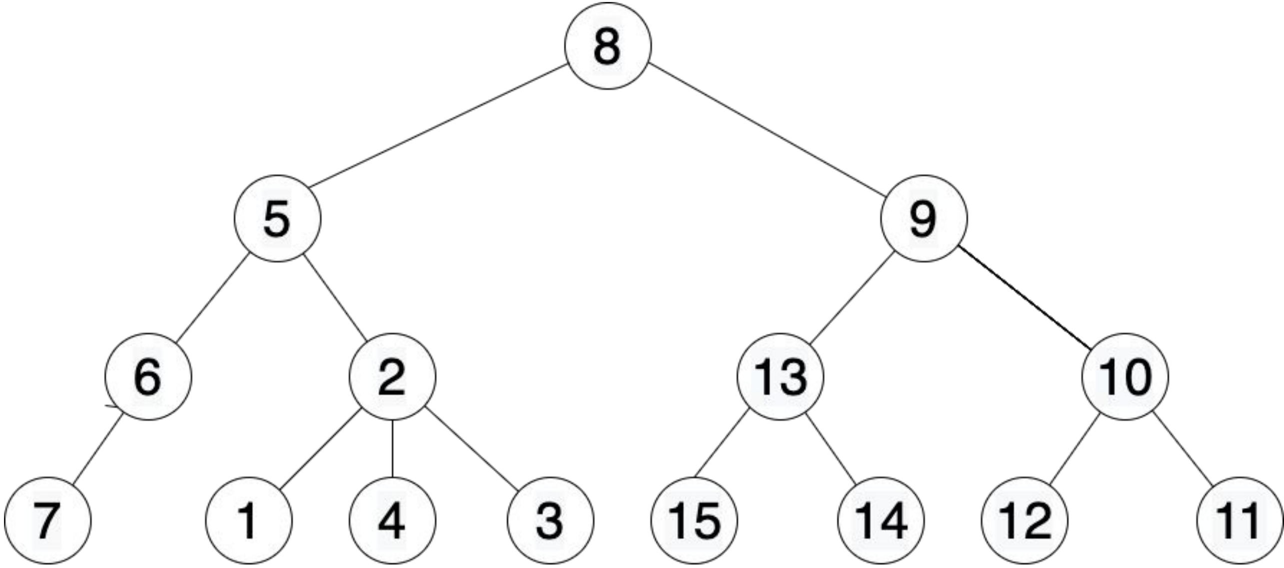
Path Updates on a Tree

- Given a tree with N nodes and Q updates of the form:
  - x, y, val -- Add val to every edge on the path from x to y.
- Print the final weights of all edges.
- Practice Problem:
  - https://www.codechef.com/problems/UPDTREE



Path Updates on a Tree

- Lazily process all updates by marking on the appropriate nodes.
  - val[x] += add; val[y] += add; val[LCA] -= 2 * add;
- Compute the Subtree Sum for every node via DFS on Unrooted Tree.
  - We have already seen out to do this.



Conclusion

- Today, we learnt about Properties of Trees, DFS on Rooted & Unrooted Trees, (Either of) Path Queries or Updates on a Tree.
- In the next class, we will learn how to find LCA and solve some more interesting problems on answering Path Queries (without updates).