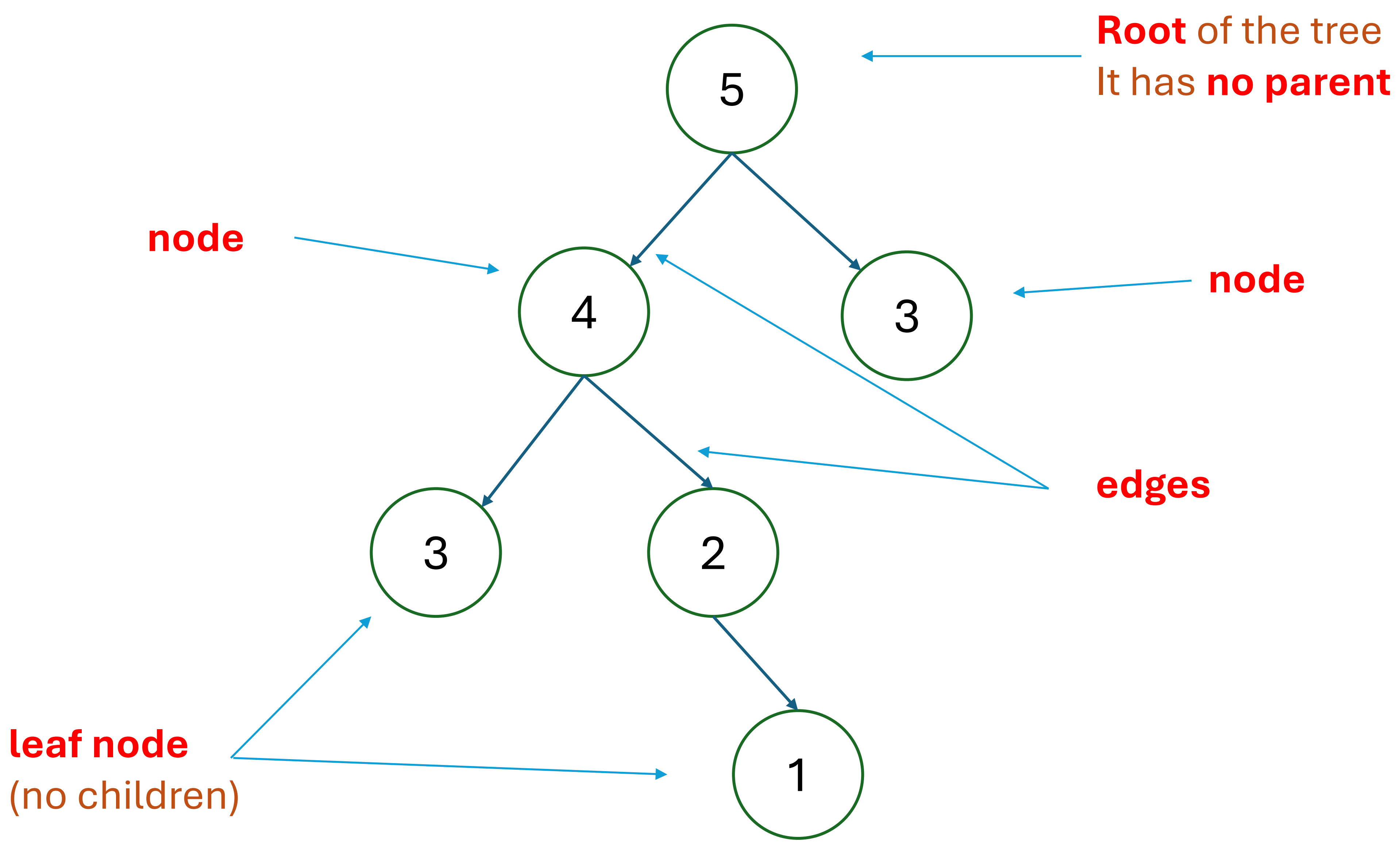
Binary Tree

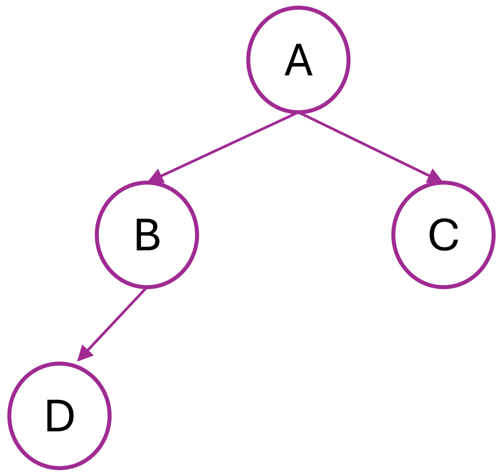
Tree

- A tree is an **abstract data type**
- A **linked node-based data structure**
  - A **hierarchical ordering** of the data which conveys **parent-child relationship**
- A tree is **a collection of nodes**, which can be empty
- If not empty, there is a single root node r, and zero or more subtrees $T_1, T_2, \ldots, T_k$ whose roots are connected by a directed edge from r.

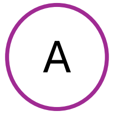
- One entry point, the **root**
  - Only access point to the tree
- Each other node is either a **leaf** or an **internal node**
- An internal node has **1 or more children**, nodes that can be reached directly from that internal node.
- The internal node is said to be the **parent** of its **child** nodes
  - All nodes, except the root, have one parent


Root of the tree
It has no parent
5
node
4
3
node
edges
3
2
leaf node
(no children)
1

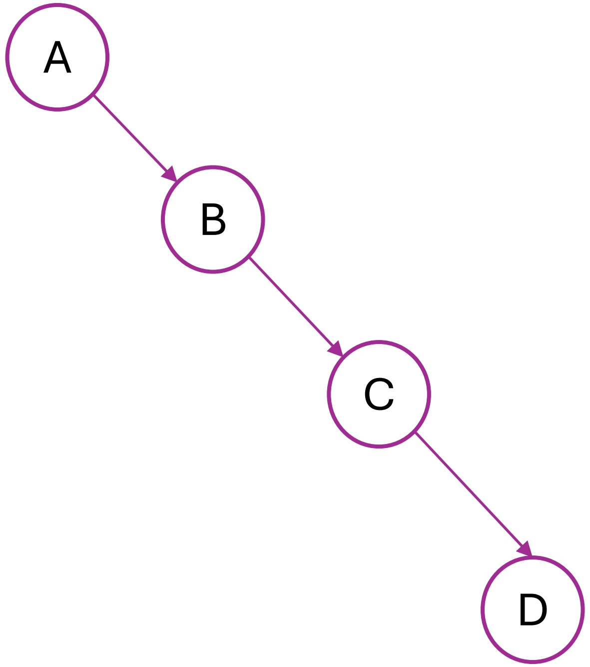
- **Siblings**: two nodes that have the same parent
- **Edge**: link from one node to another
- **Path length**: number of edges that must be traversed to get from one node to another
- **Depth**: number of edges from the root node to a particular node
- **Height of a node in a tree**: number of edges in the longest path from the node to a leaf node.
  - The height of the root node is the height of the tree
  - *Height of a tree containing only root is 0*
  - *Height of an empty tree is -1*
- **Descendants**: any nodes that can be reached via 1 or more edges from this node
- **Ancestors**: any nodes for which this node is a descendant



- B and C are siblings
- Path length from node A to node D is 2
  - Depth of the tree from root to node D
- Height of the node B is 1
- Height of the tree is 2
- B is the descendant of A and B is the ancestor for D

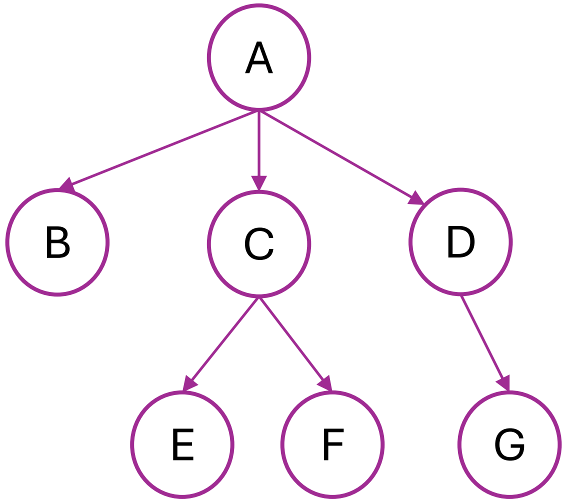


height = 0



height = 3

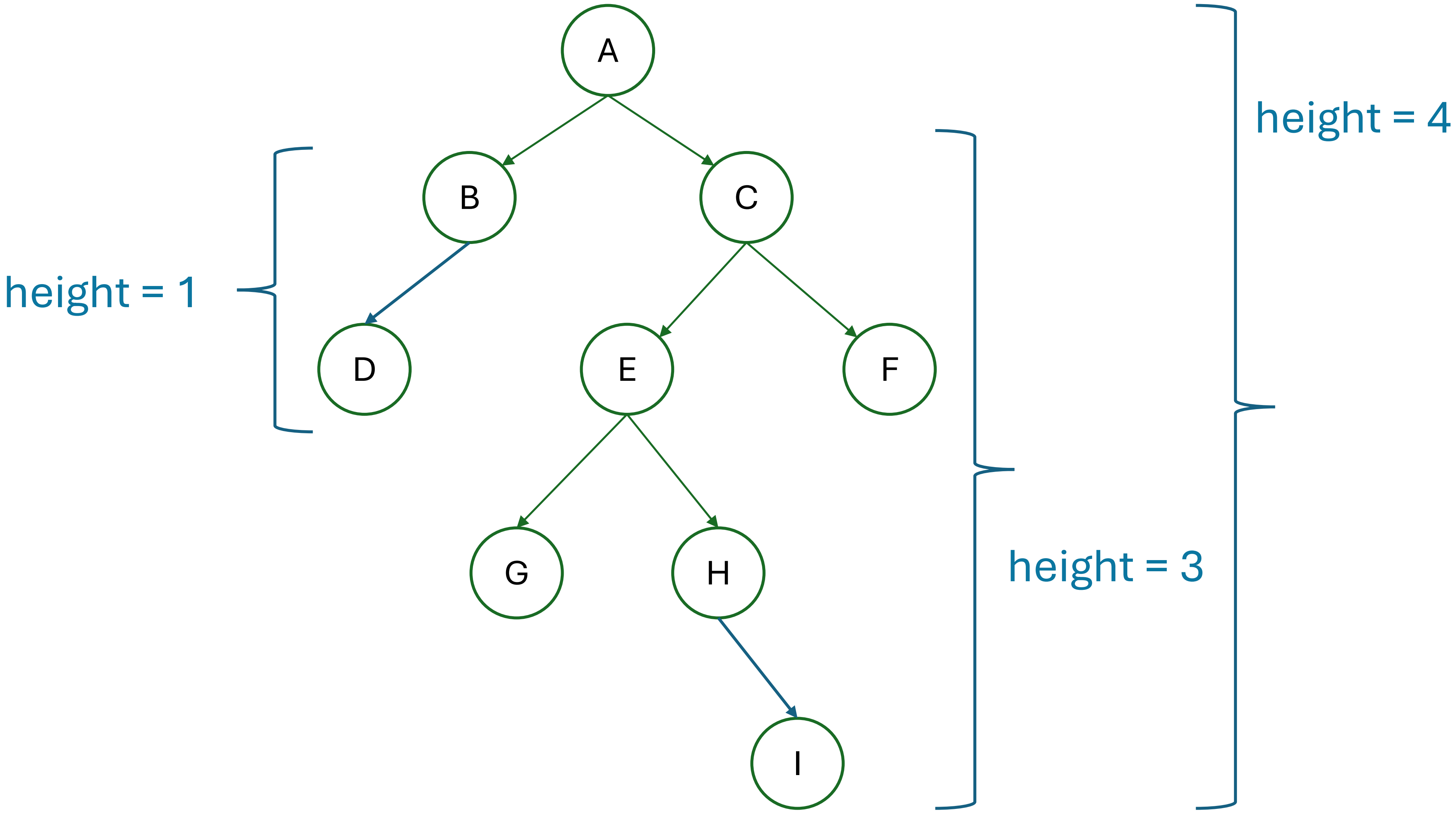
(from node A to node D)



height = 2

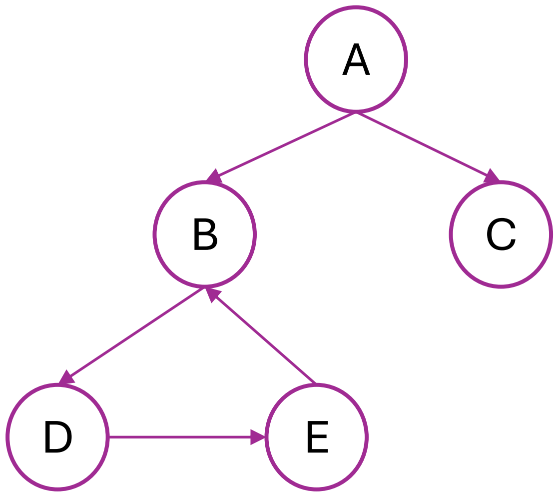
(from node A to node E)

height = -1

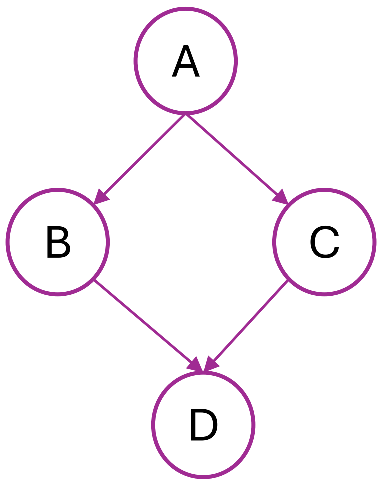


A
B
C
D
E
F
G
H
I
height = 1
height = 3
height = 4

- A **graph** is a collection of nodes and edges
- A **tree** is a type of graph
- A tree cannot have **cycles** – *a non-empty path from some node to itself*
  - A node cannot be its ancestor, and a node cannot have multiple parents

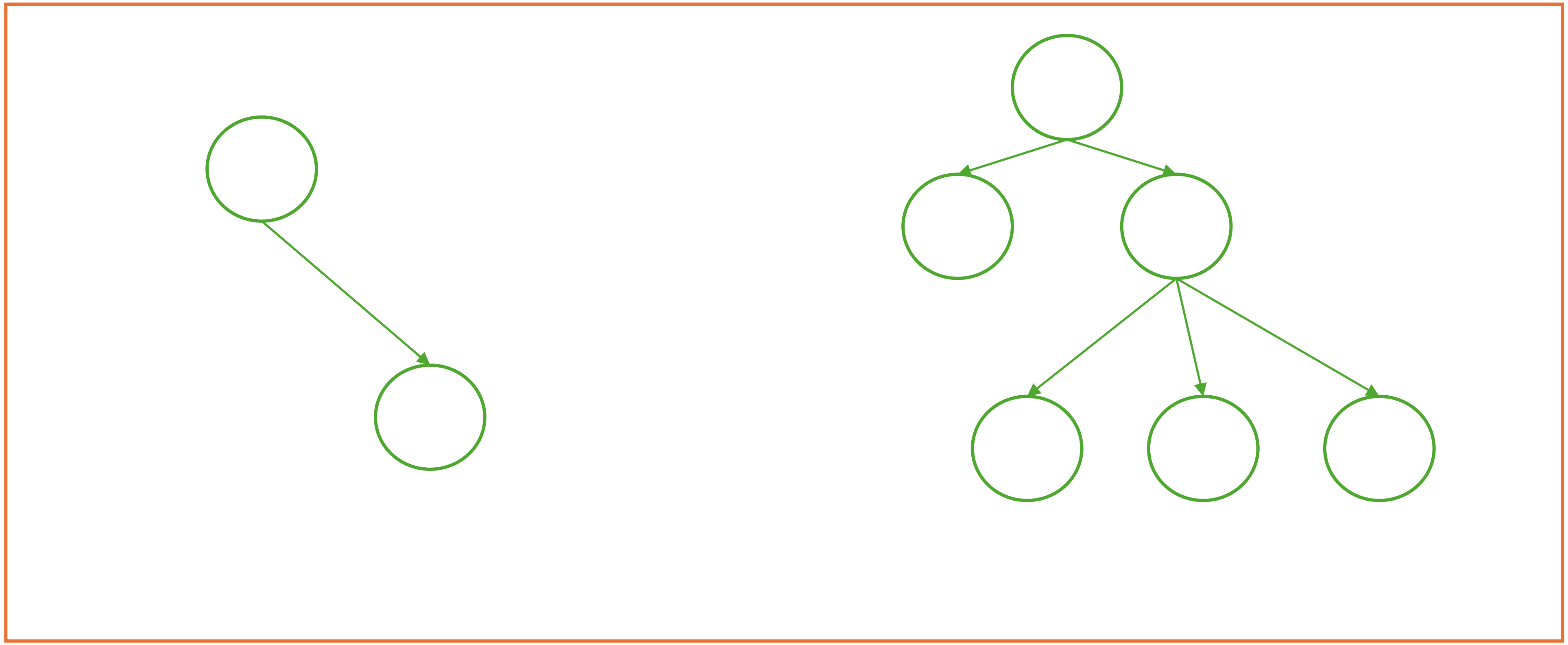


- Not conveying parent-child relationship
- Has a cycle

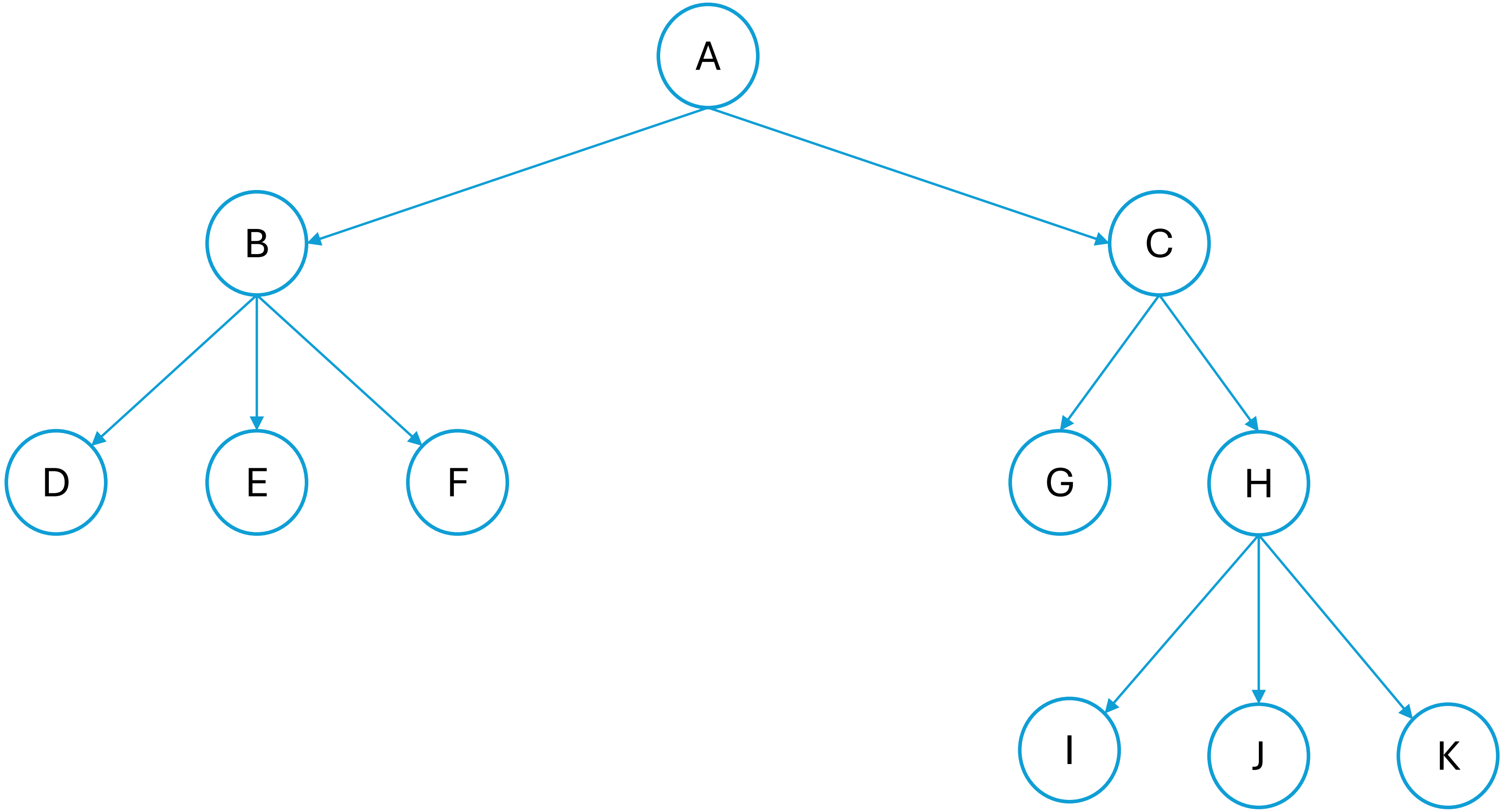


- Cannot have two parents

- The following is not a tree; It’s a **forest**



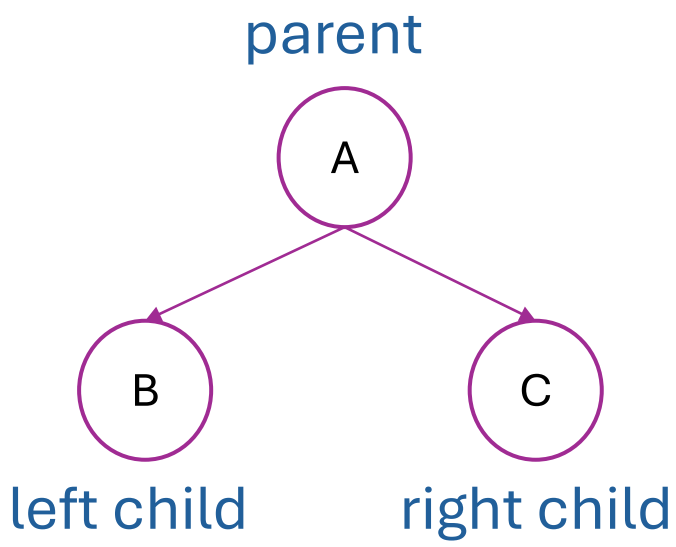
- How many edges must there be in a tree with ***n*** nodes?



n-1 edges

Binary Tree

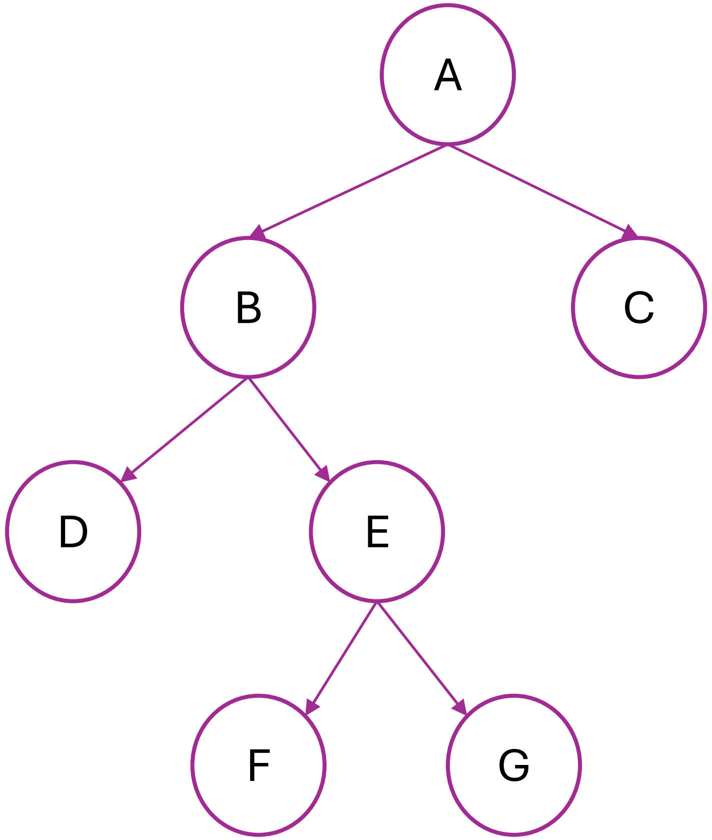
- A tree in which every node has **at most two children**
  - The possible children are usually referred to as the **left child** and the **right child**




```
treeNode{
      int data //or whatever data type suits our need
      treeNode *left
      treeNode *right
}
```

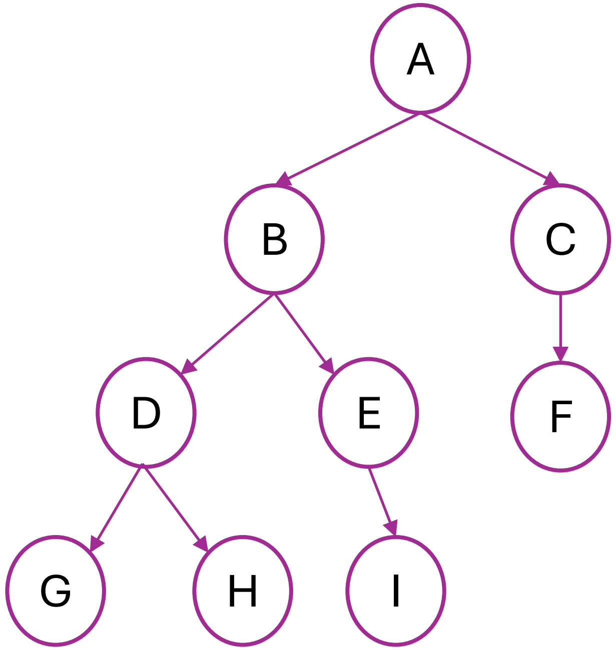
Full Binary Tree

- A binary tree in which each node has **2 or 0 children**

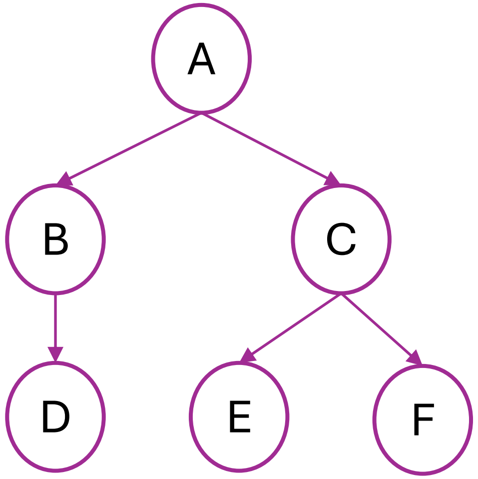


Complete Binary Tree

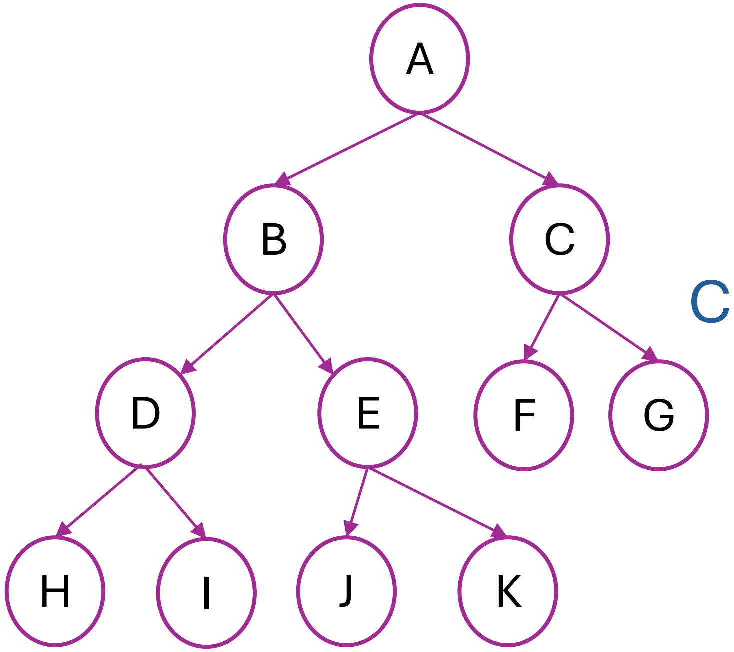
- All level of the tree are **completely filled up**, except perhaps the last level, whose nodes **must all be as far left as possible**



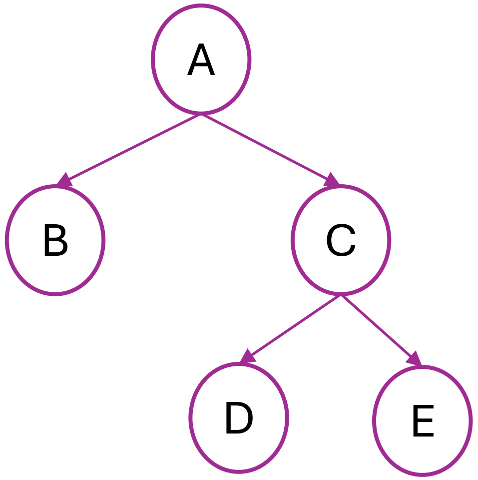
Not complete



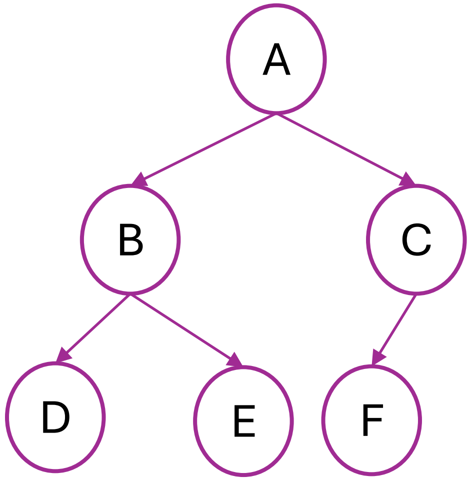
Not complete



Complete

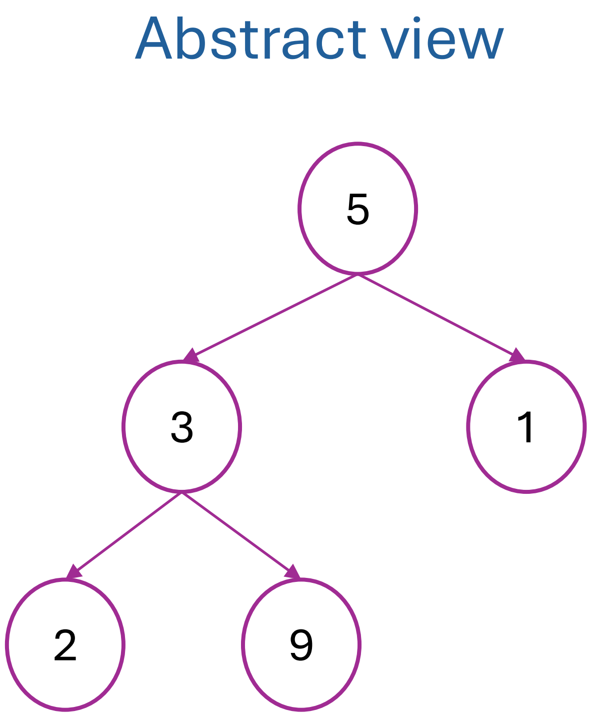


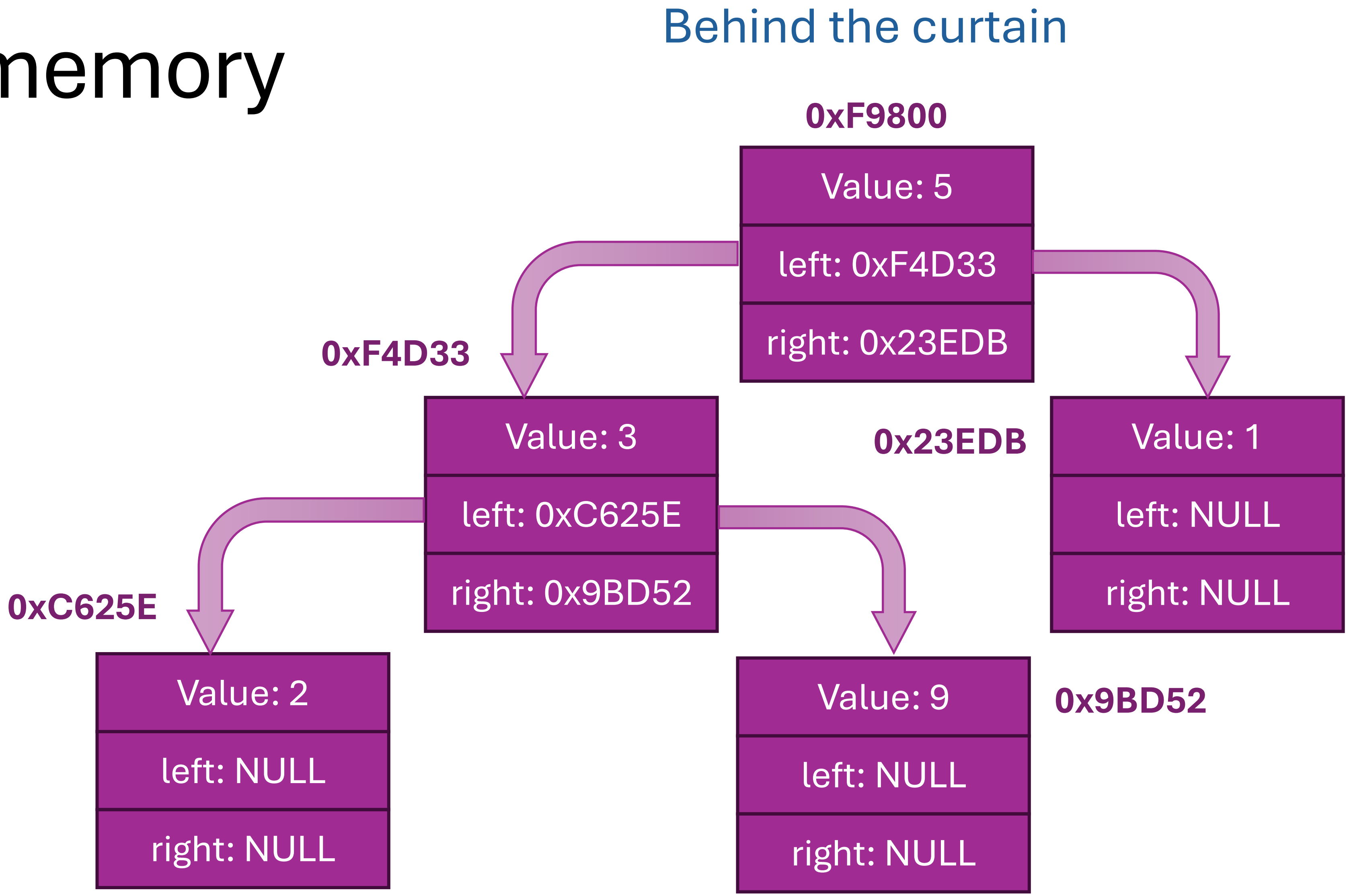
Not complete



Complete

A binary tree in memory





We would also have a dedicated variable to store the address of the root node

Perfect Binary Tree

- A binary tree with **all leaf nodes at the same depth**.
- **All internal nodes have exactly two children**.

- A perfect binary tree has the maximum number of nodes for a given height

- A perfect binary tree has **$(2^{(n+1)} - 1)$ nodes** where n is the height of the tree
  - height = 0 -> 1 node
  - height = 1 -> 3 nodes
  - height = 2 -> 7 nodes
  - height = 3 -> 15 nodes

Balanced Binary Tree

- A balanced binary tree, also referred to as a **height-balanced binary tree**, is defined as a binary tree in which the **height of the left and right subtree of any node differ by not more than 1**.