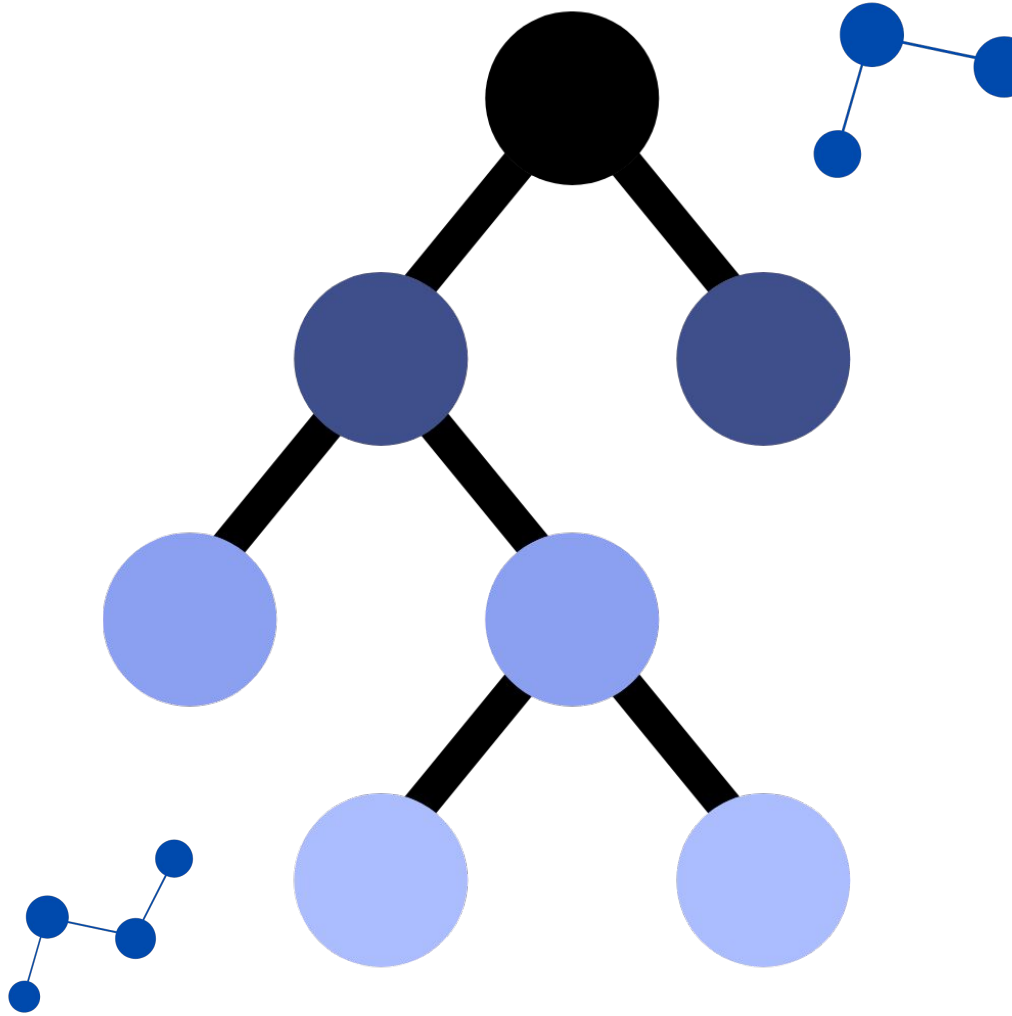
Trees

Binary Trees and Binary Search Trees

Part I



Lecture Flow

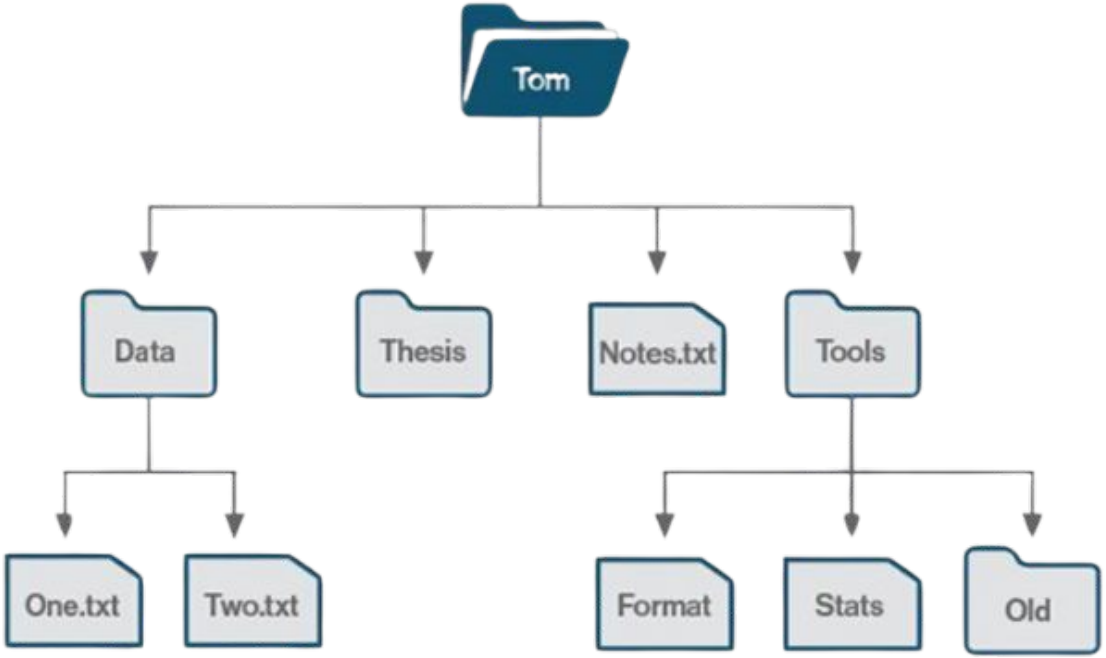
1. Prerequisites
2. Real life problem
3. Definition
4. Tree Terminologies
5. Types of Trees
6. Tree Traversal
7. Checkpoint
8. Basic BST Operations
9. Time and space complexity analysis
10. Things to pay attention to(common pitfalls)
11. Applications of a Tree
12. Practice Questions

Pre-requisites

- Basic Recursion
- Linked List
- Stacks
- Queues

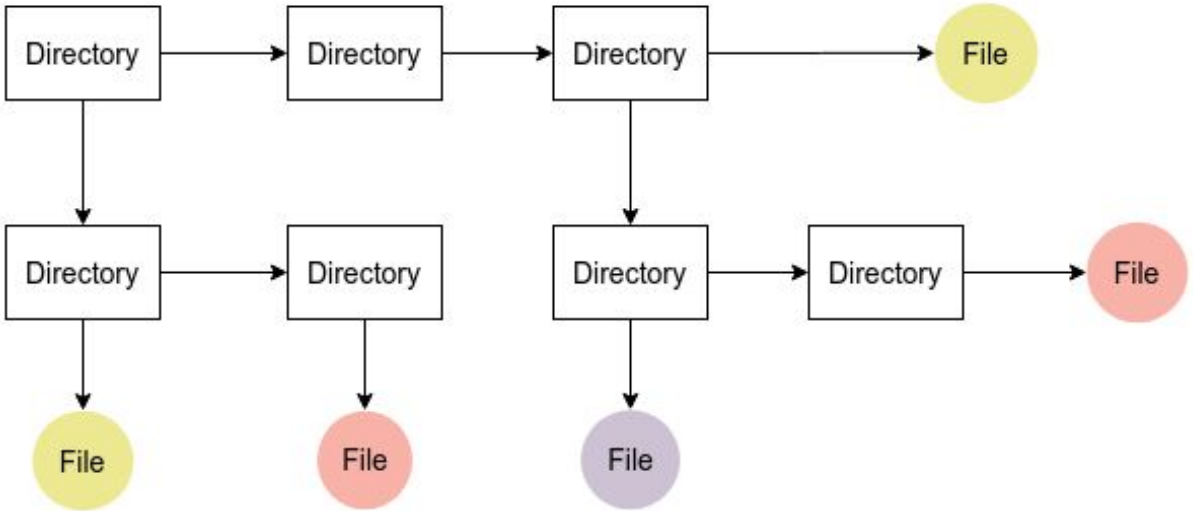
Real-Life Problem

What data structure will be best to implement a file management system ?



Real-Life Problem

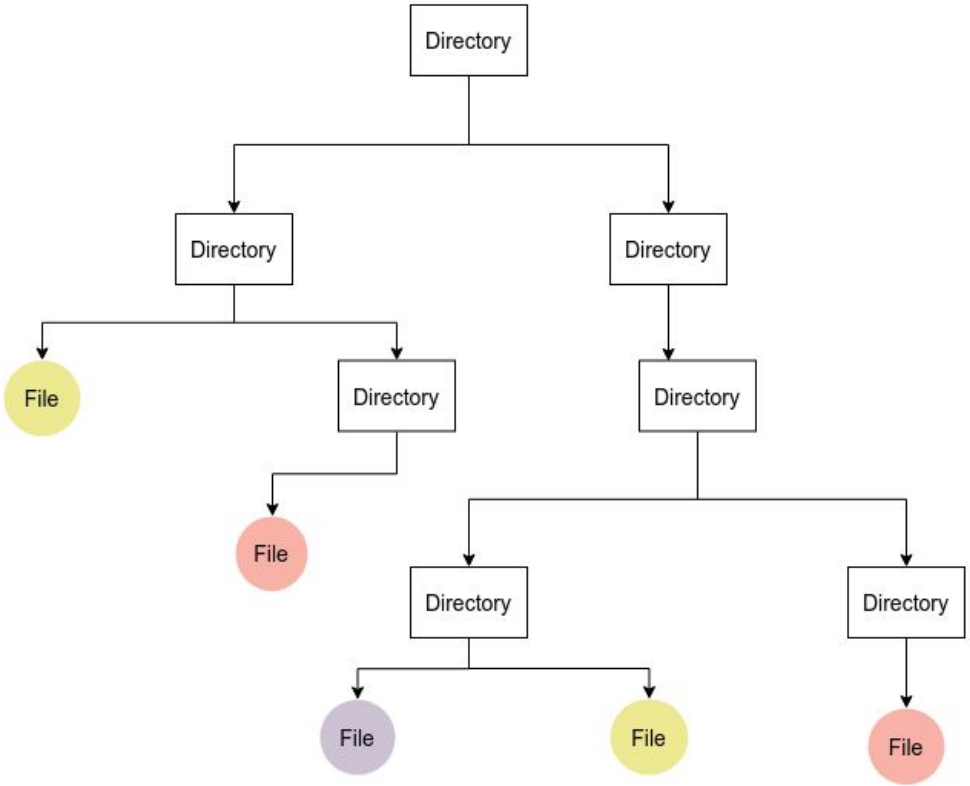
What about linked-lists? Why?



But a directory can contain multiple directories and/or files.

Real-Life Problem

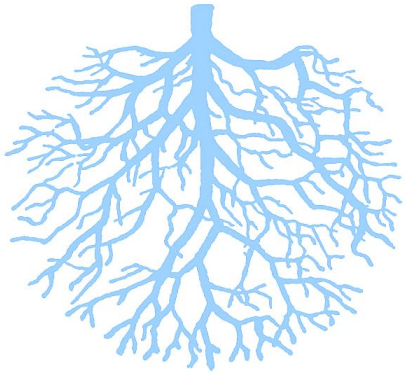
So a linked list with multiple ‘next nodes’?

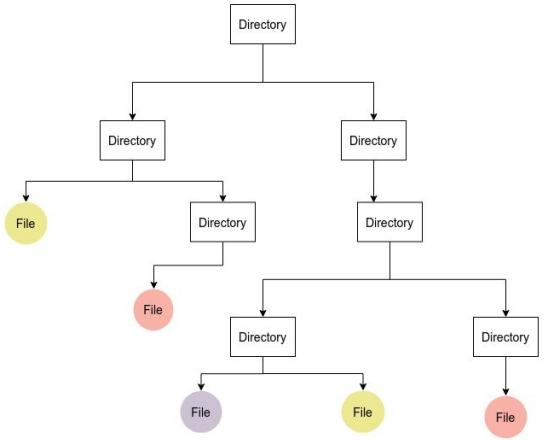


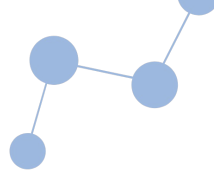
What are Trees?

Definition

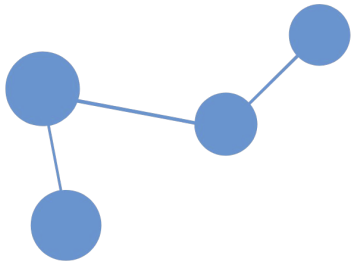
- A hierarchical, non-linear data structure composed of zero or more nodes.
- Why the name tree? Such data structure branches out starting from the root, pretty much like the trees around us but upside down.





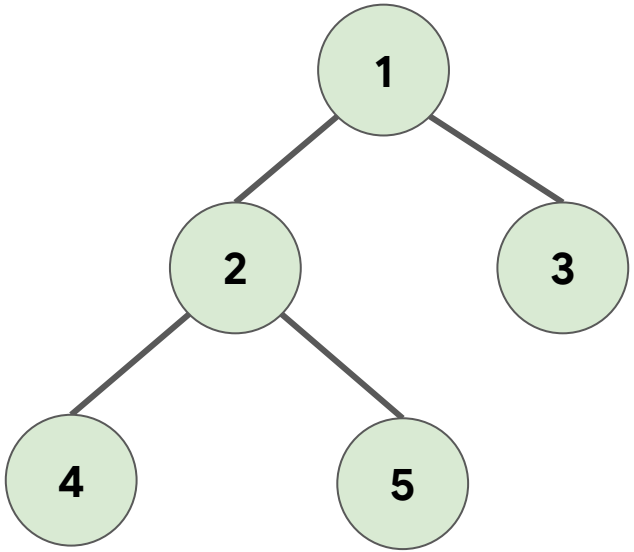


Terminologies in Trees



Terminology - Node

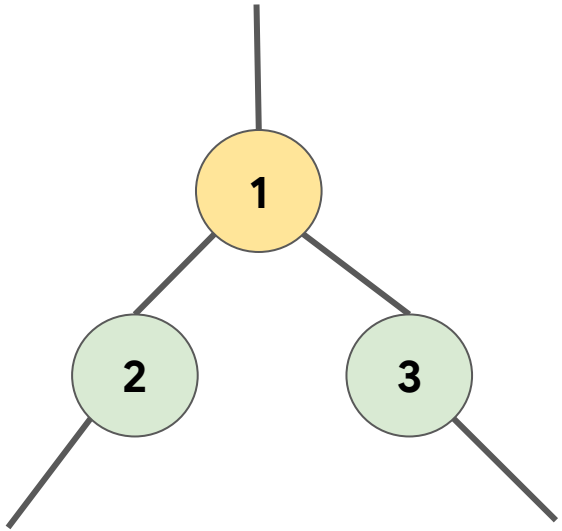
- A data structure that contains a value, a condition or a data structure (yes even trees)
- In trees, a node can have 0 or more children but at most one parent.




```
class Node:
    def __init__(self, key:int):
        self.left = None
        self.right = None
        self.val = key
```

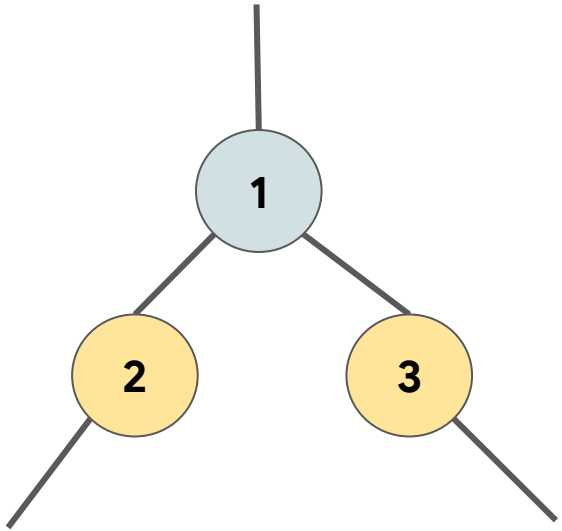
Terminology - Parent

- A node is called parent node to the nodes it's pointers point to.



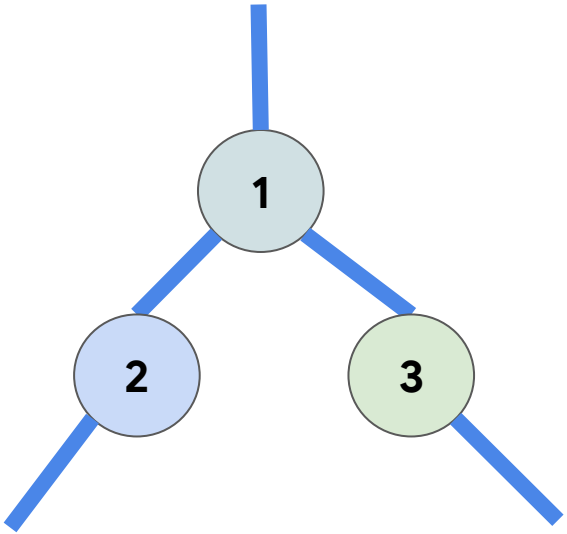
Terminologies - Child, Siblings

- The nodes a node's pointers point to are called child nodes of that node.
- Siblings: nodes that have the same parent node.



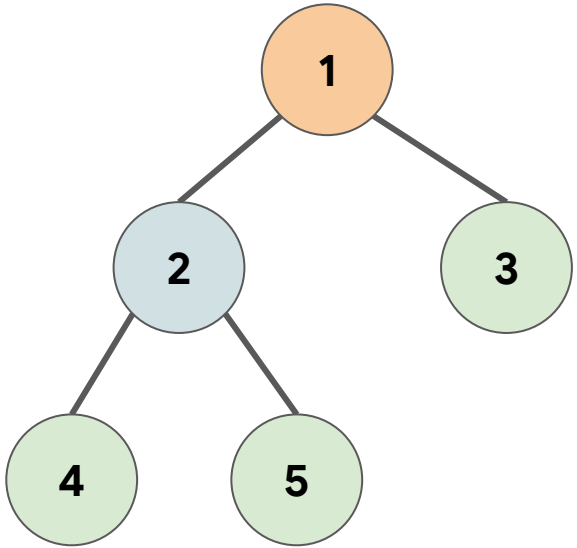
Terminology - Edge

- Edge: a connection between a child and parent node.



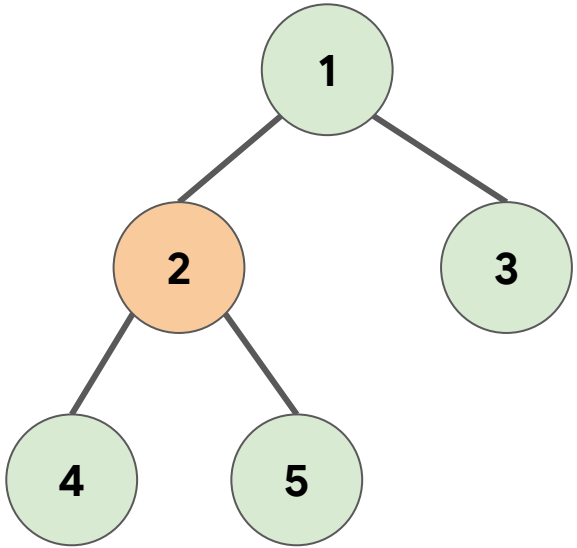
Terminology - Root Node

- A node with no parent is called a root node.



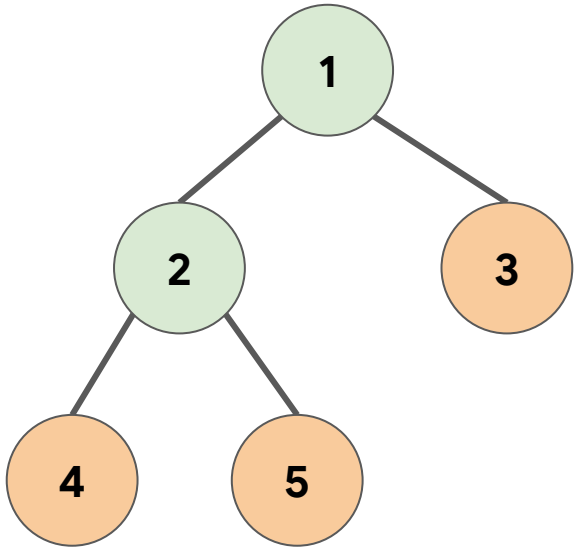
Terminology - Inner Node

- A node with the parent and the child is called an inner node or internal node



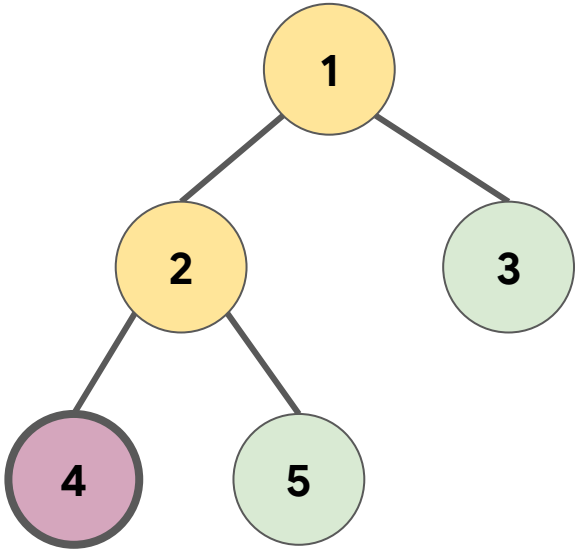
Terminology - Leaf Node

- A node with no children



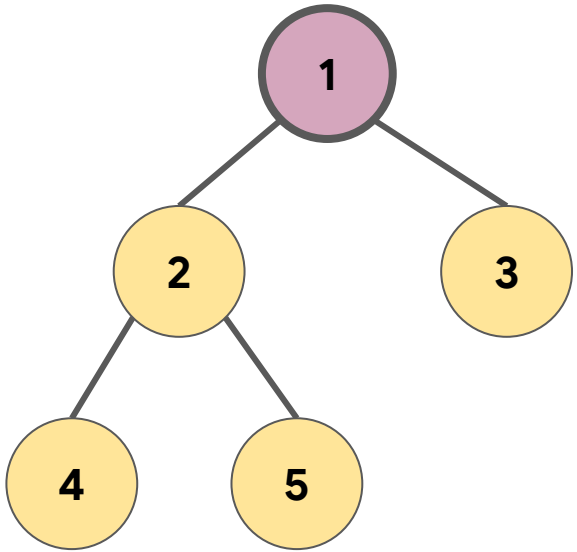
Terminology - Ancestor

- A node is called the ancestor of another node if it is the parent of the node or the ancestor of its parent node.
- In simpler terms, A is an ancestor of B if it is B's parent node, or the parent of B's parent node or the parent of the parent of B's parent node and so on.



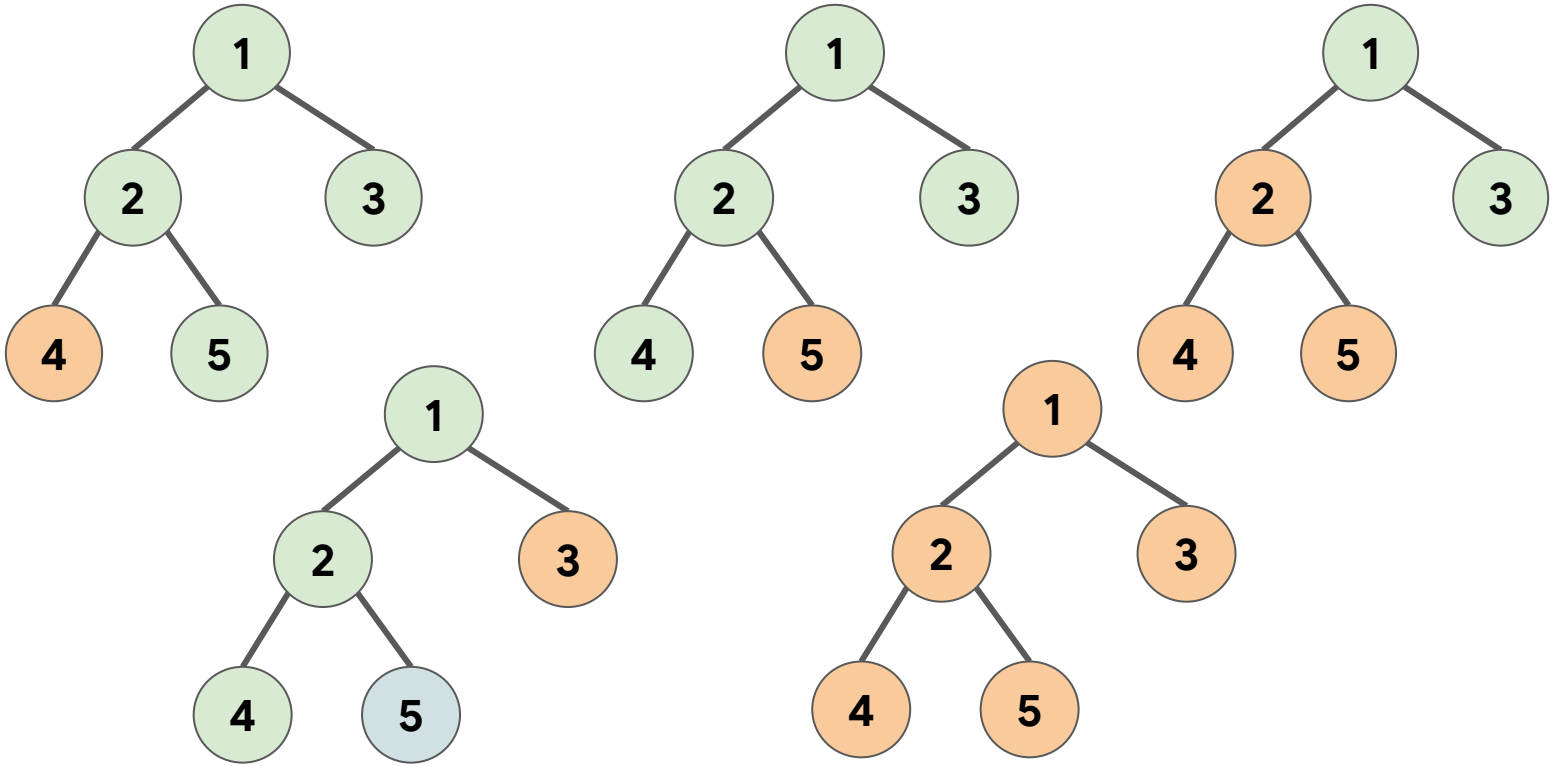
Terminology - Descendant

- A node A is called the descendant of another node B if B is the ancestor of A.
- In simpler terms, A is a descendant of B if A is the child node of B, or the child of the child node of B or the child of child of the child node of B and so on.



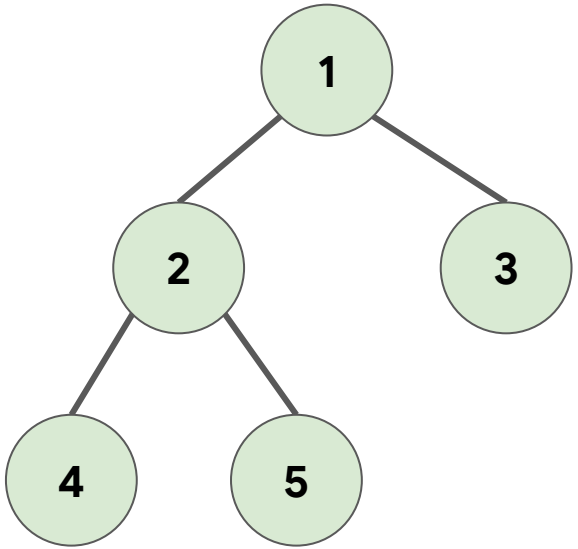
Terminology - Sub-tree

- A subtree of a tree consists of a node **n** and all of the descendants of node **n**.



Terminology - Level, Height and Depth

- The level of a tree indicates how far you are from the root
- The height of a tree indicates how far you are from the farthest leaf
- The depth of the node is the total number of edges from the root to the current node. Level = depth + 1

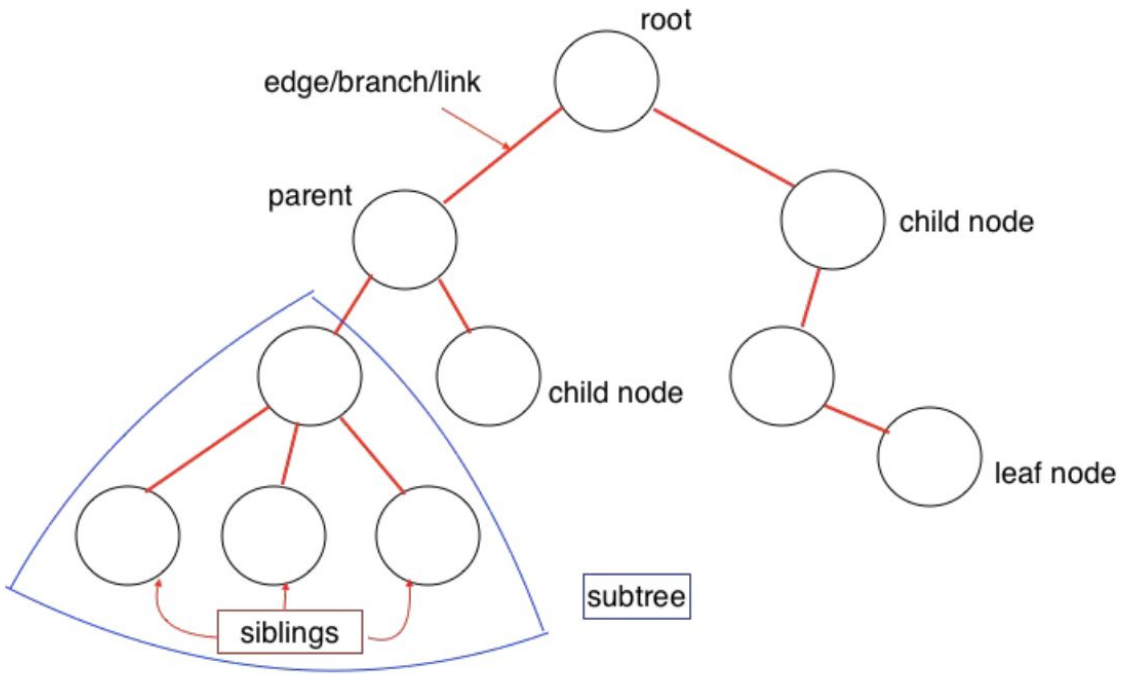


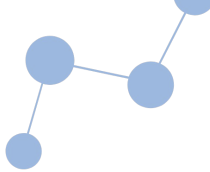
Level 1 Height 2 Depth 0

Level 2 Height 1 Depth 1

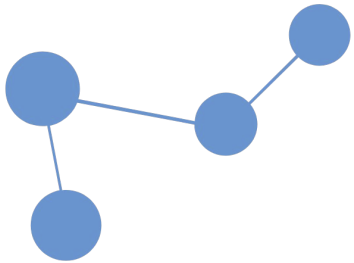
Level 3 Height 0 Depth 2

Terminologies - Summary





Types of Trees

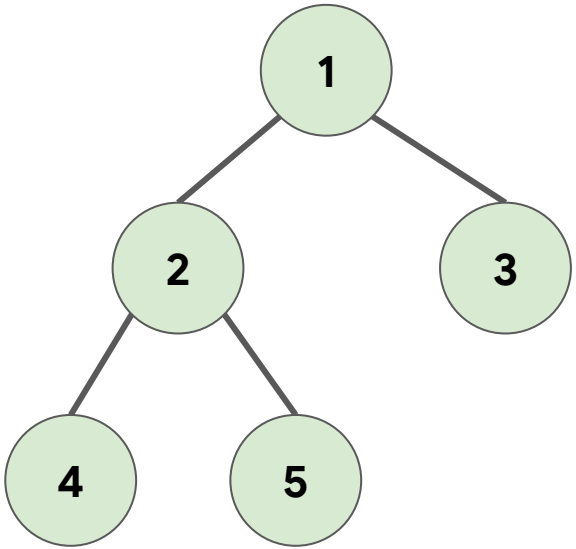


Types

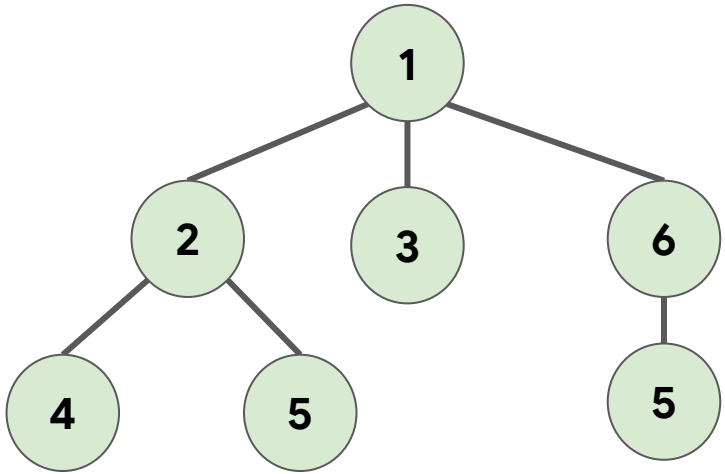
Depending on the number of children of every node, trees are generally classified as

1. Binary tree: Every node has at most two children
2. n-ary tree: Every node has at most n children

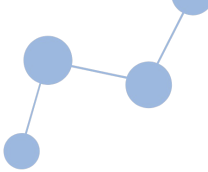
Binary trees



3-ary trees (a.k.a, Ternary Trees)



Types - Implementation



Binary tree

```
class Node:
    def __init__(self, key:int):
        self.left = None
        self.right = None
        self.val = key
```

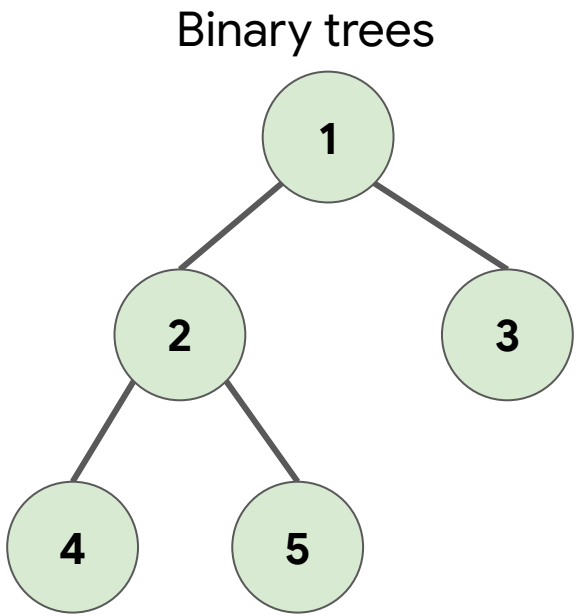
N-ary tree

```
class Node:
    def __init__(self, key:int):
        self.val = key
        self.children = []
        # len(children) <= N
```

Note: the N-ary tree implementation can be used for binary trees if `len(self.children) <= 2`

Types - Binary Trees

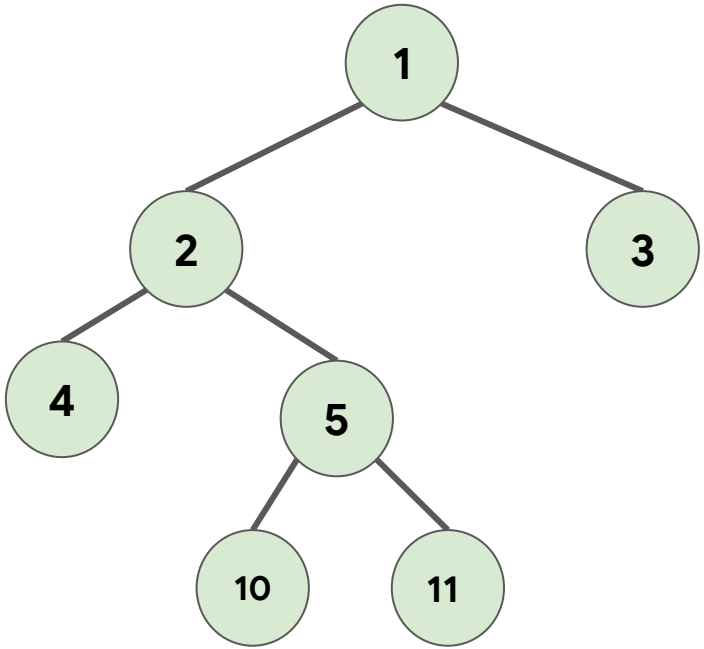
A binary tree is a tree in which every internal node and root node has at most two children. These two child nodes are often called the left child node or right child node of the node.



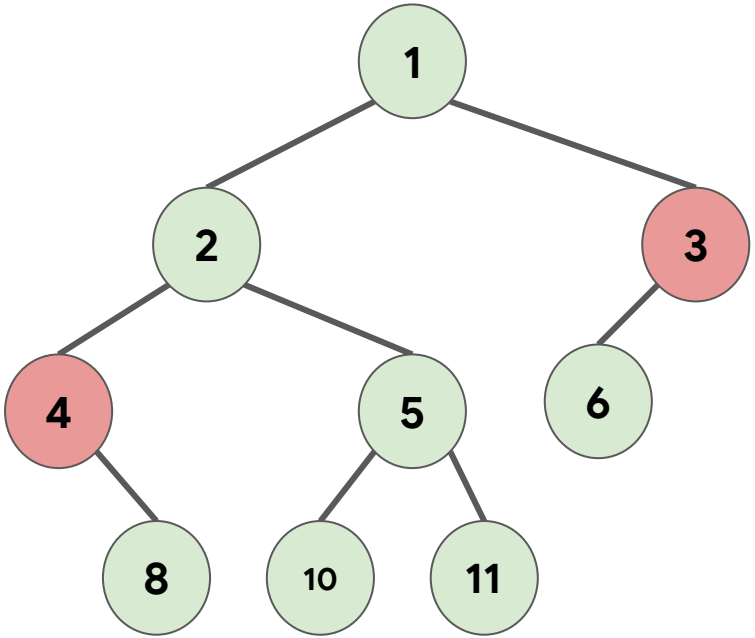
Types - Full Binary Trees

- A full binary tree is a special type of binary tree where each node has 0 or 2 children

✔ Full Binary tree



✘ Not a Full Binary tree



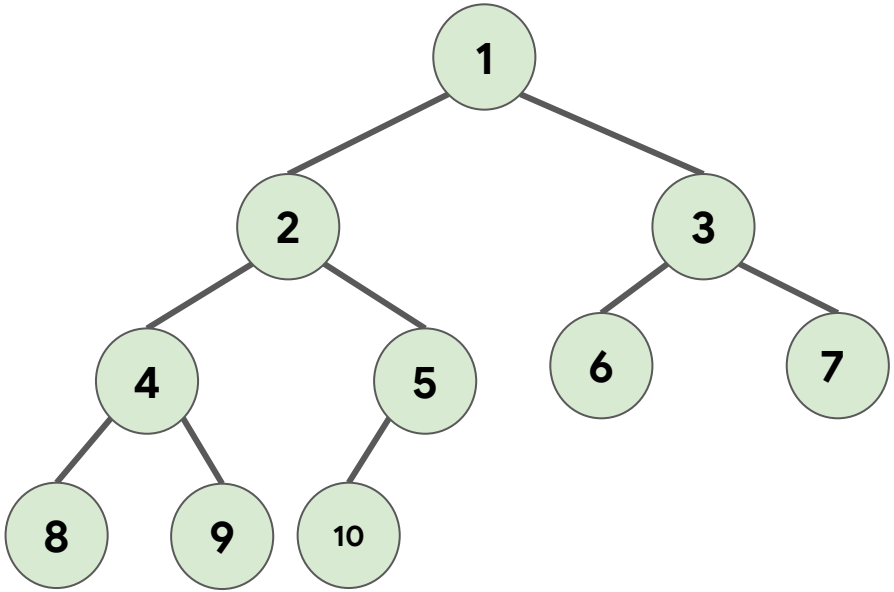
Types - Complete Binary Trees

- A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the “lowest” level nodes which are filled from as left as possible.

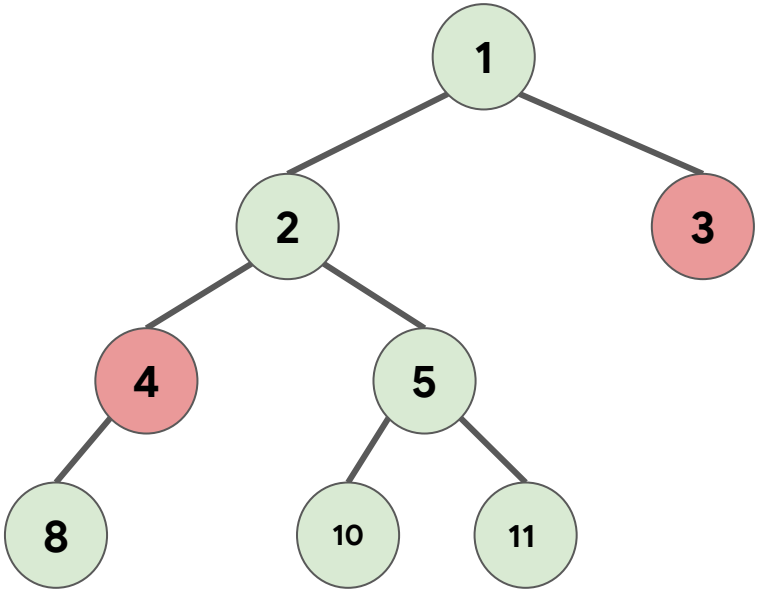
- A complete binary tree is just like a full binary tree, but with two major differences
  - All the leaf elements must lean towards the left.
  - The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.

Types - Complete Binary Trees

✔ Complete Binary tree

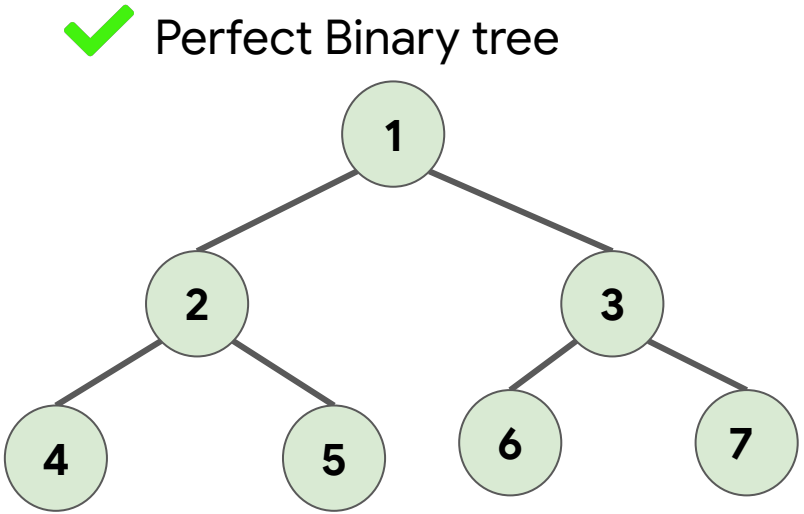


✘ Not a Complete Binary tree



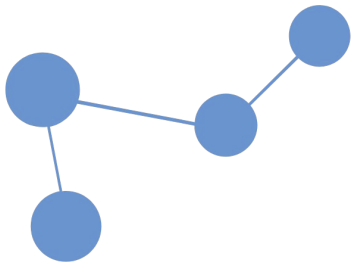
Types - Perfect Binary Trees

- A perfect binary tree is a special type of binary tree in which all the leaf nodes are at the same depth, and all non-leaf nodes have two children.



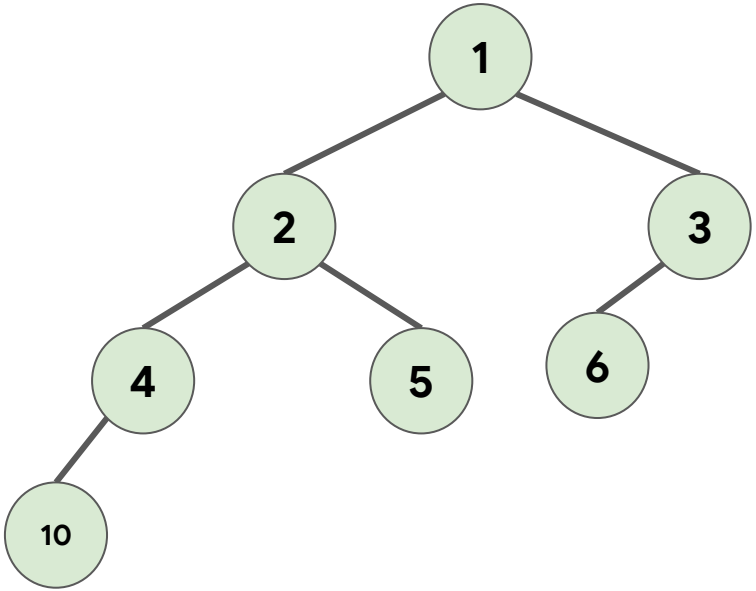
Types - Balanced Binary Trees

- A 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.

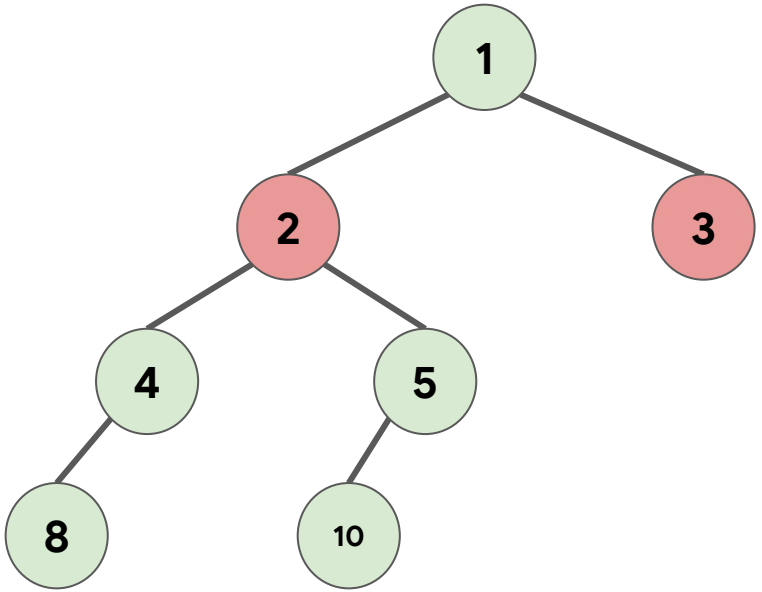


Types - Balanced Binary Trees

✔ Balanced Binary tree

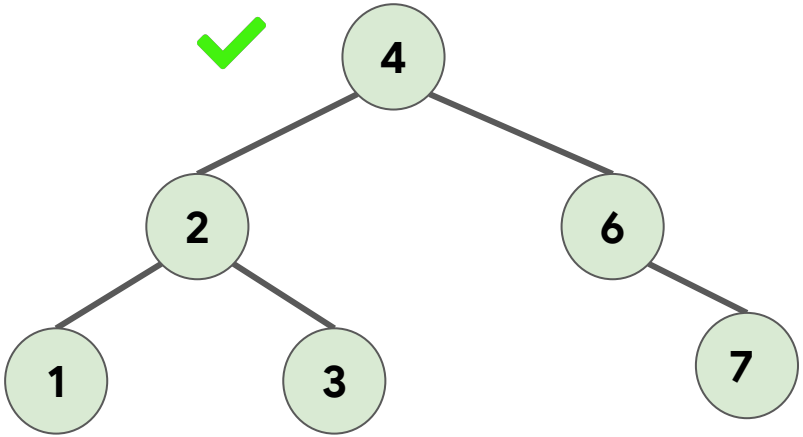


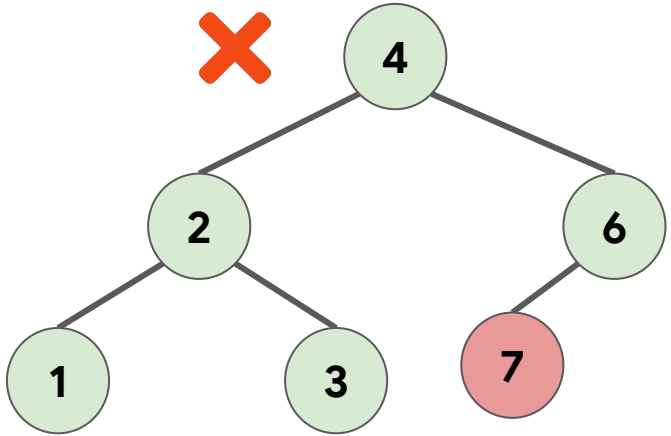
✘ Not a Balanced Binary tree



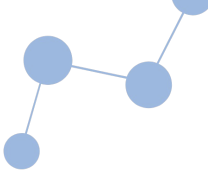
Types - Binary Search Trees

- A binary search tree is a binary tree that has the following properties:
  - The left subtree of the node only contains values less than the value of the node.
  - The right subtree of the node only contains values greater than or equal to the value of the node.
  - The left and right subtrees of the nodes should also be the binary search trees.

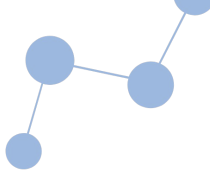




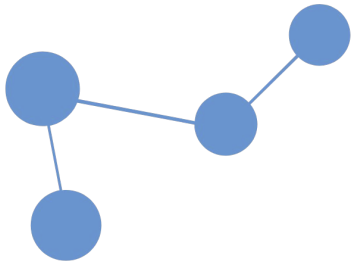
Types - Binary Search Trees



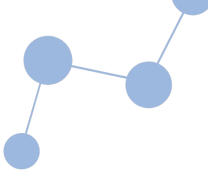
- Why binary search trees? Efficient search, insert and delete.
- Applications
  - Sorting large datasets
  - Maintaining sorted stream
  - Implementing dictionaries and priority queues



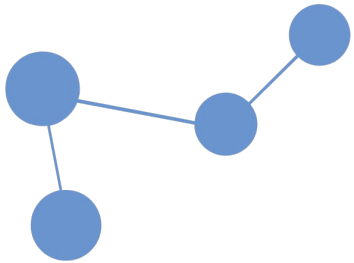
Tree Traversal



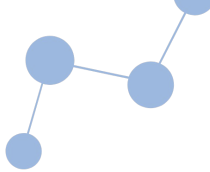
Tree Traversal



- Traversing a tree means visiting every value. Why would you need to do that?
  - To determine the a certain statistic, such as extrememum value or average over all values
  - To sort the values

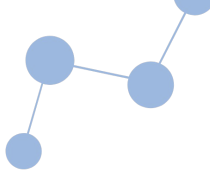


Tree Traversal - Depth First Search



- Depth-first search (DFS) is a method for exploring a tree or graph.
- In a DFS, you go as deep as possible down one path before backing up and trying a different one. You explore one path, hit a dead end, and go back and try a different one.
- There are basically three ways of traversing a binary tree:
    1. Preorder Traversal
    2. Inorder Traversal
    3. Postorder Traversal

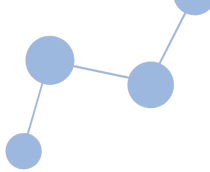
Depth First Search - Preorder



- In preorder traversal, we recursively traverse the parent node first, then the left subtree of the node, and finally, the node's right subtree.

- Problem Link

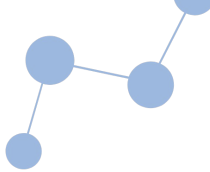
Depth First Search - Inorder



- In inorder traversal, we traverse the left subtree first, then the the parent node and finally, the node's right subtree.
- Inorder traversal in BST results in a sorted order of the values

Problem Link

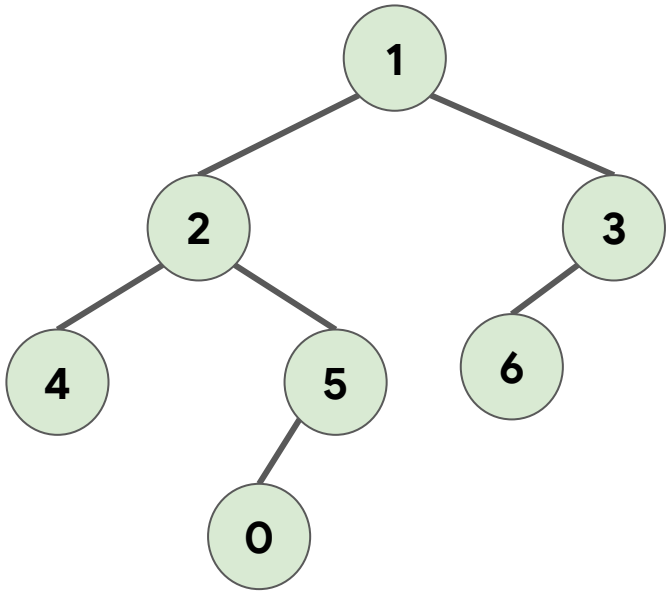
Depth First Search - Postorder



- In inorder traversal, we traverse the left subtree first, then the the right subtree of, and finally, the parent node.

Problem Link

Depth First Search - Example

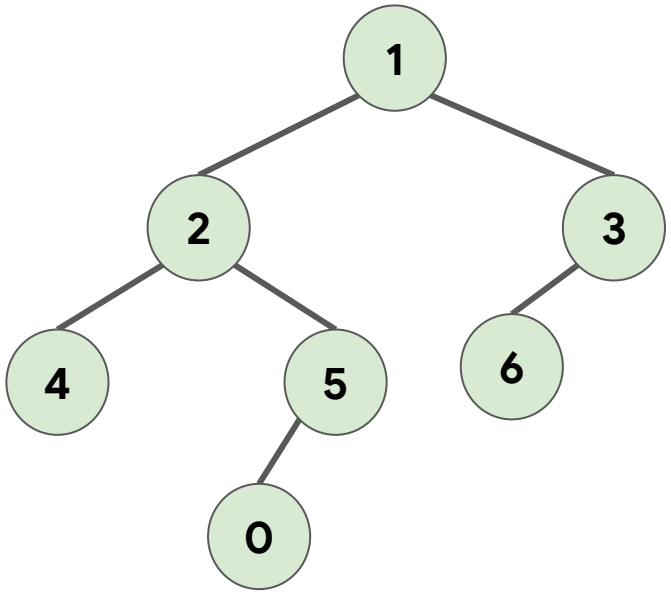


Preorder:

Inorder:

Postorder:

Depth First Search - Example



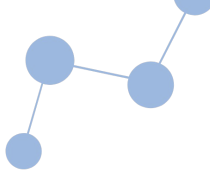
Preorder: 1 2 4 5 0 3 6

Inorder: 4 2 0 5 1 6 3

Postorder: 4 0 5 2 6 3 1

Checkpoint

link



Practice Problem

Link