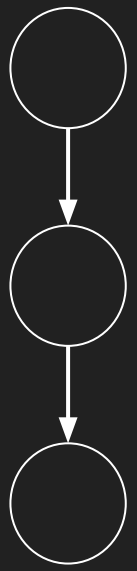
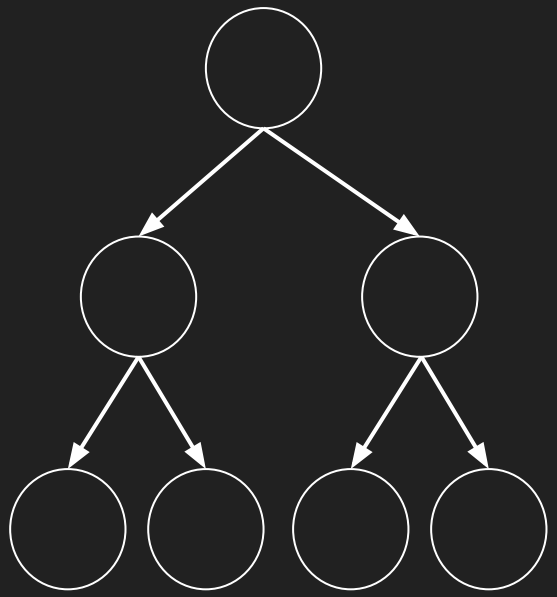
Binary Trees

Linked List

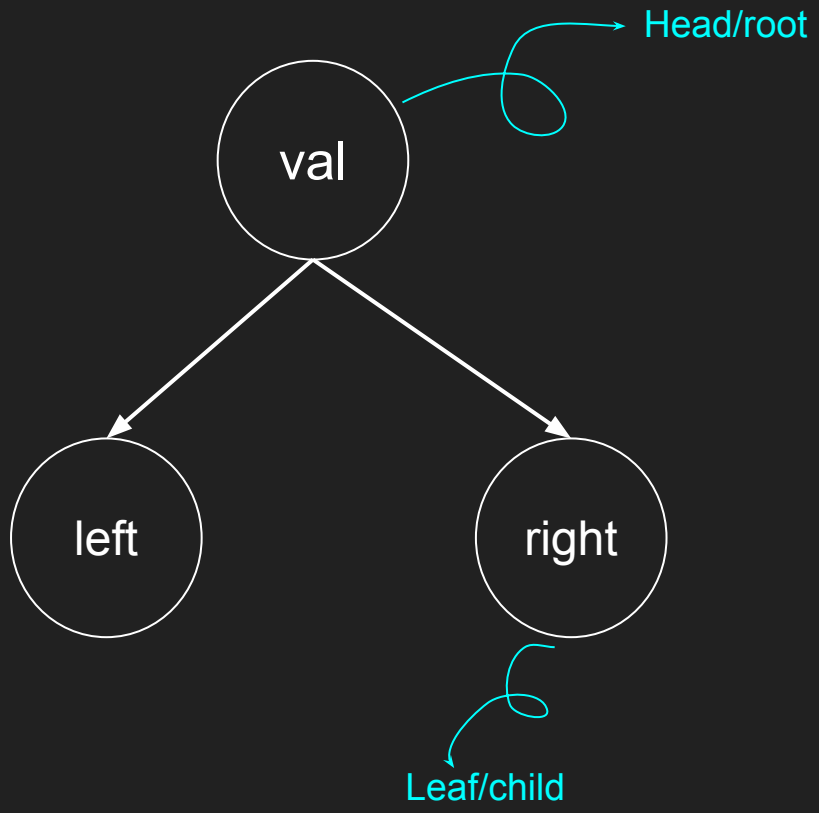


Binary Tree



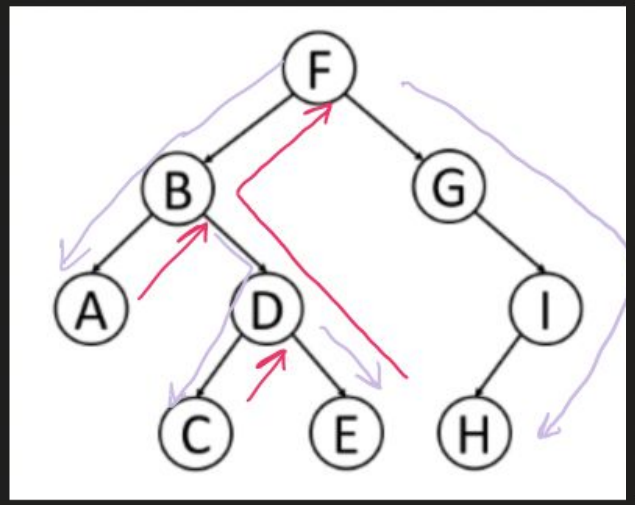
Tree Node

```
public class TreeNode<T> {
 T val;
 TreeNode left;
 TreeNode right;
}
```



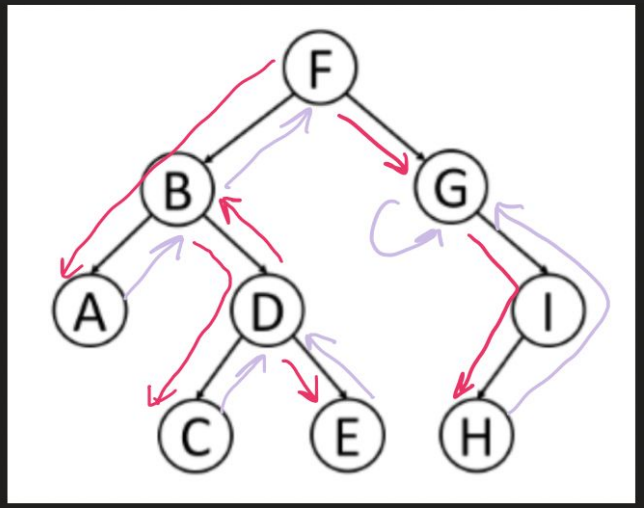
✨Traversals✨

Pre-order [root][left][right]



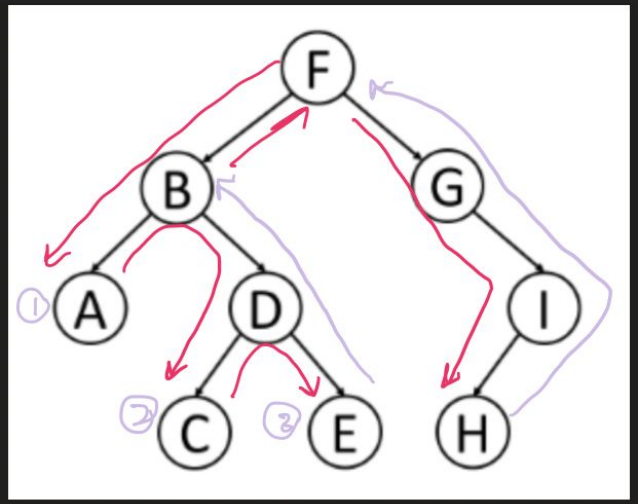
[F, B, A, D, C, E, G, I, H]

In-Order [left][root][right]



[A, B, C, D, E, F, G, I, H]

Post-Order [left][right][root]



[A, C E, D, B, H, I, G, F]

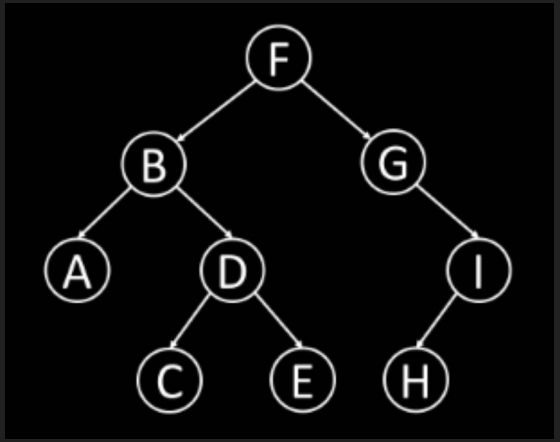
```
private void traversePreorder (TreeNode<T>,
List<T> traversal) {

  if(subtree == null) {

    return;

  }

  traversal.add(subtree.val);

  traversePreorder(subtree.left, traversal);

  traversePreorder(subtree.right, traversal);

}
```

```
private void traversePostorder (Node<T>
subtree, List<T> traversal) {

  if (subtree == null) {

    return;

  }

  traversePostorder(subtree.left, traversal);

  traversePostorder(subtree.right, traversal);

  traversal.add(subtree.element);

}
```

```
private void traverseInorder(Node<T> subtree, List<T> traversal) {
  if(subtree == null) {
    return;
  }
  traverseInorder(subtree.left, traversal);
  traversal.add(subtree.element);
  traverseInorder(subtree.right, traversal);
}
```

BFS

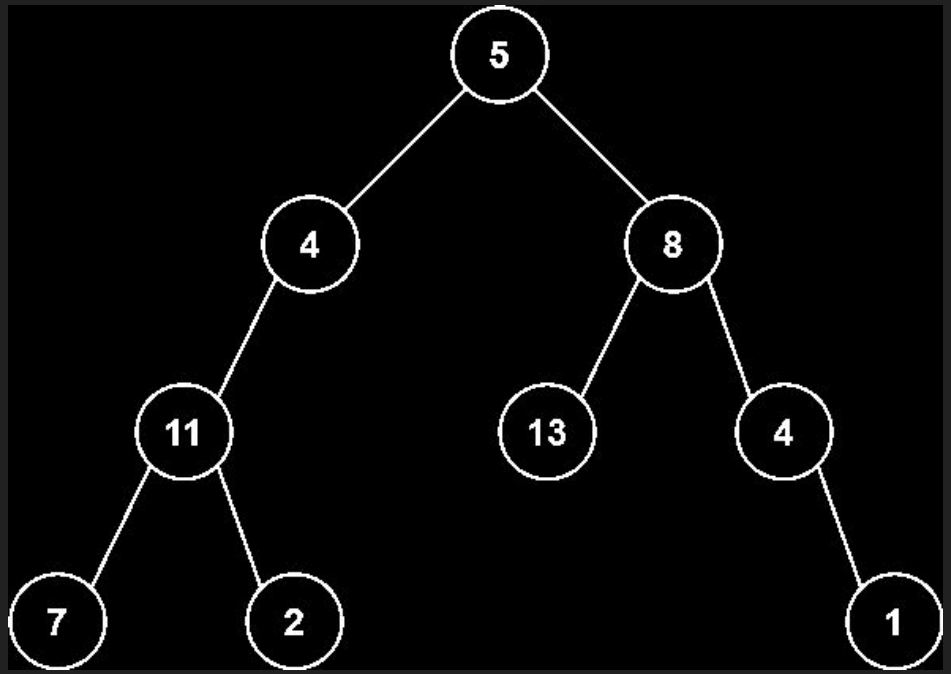
```
private void traverseBreadthFirst(Node<E> subtree, List<E> traversal) {
    Deque<Node<E>> queue = new ArrayDeque<>();
    queue.addLast(subtree);
    while (!queue.isEmpty()) {
      Node<E> node = queue.pollFirst();
      traversal.add(node.element);
      if (node.left != null) {
        queue.addLast(node.left);
      }
      if (node.right != null) {
        queue.addLast(node.right);
      }
    }
 }
```



F, B, G, A, D, I, C, E, H

Easy Problem
-- just need to traverse

Given the root of a binary tree and an integer `targetSum`, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals `targetSum`.



E.g. `targetSum` = 22 → return true
As Path 5-4-11-2 = 22

1. Think about the intended behaviour (which traversal order do we want?)
2. Base case
3. Recursive case (left child & right child)

Pre-order traversal Algorithm

```
private void traversePreorder(TreeNode<T> , List<T> traversal) {
  if(subtree == null) {
    return;
}
  traversal.add(subtree.val);
  traversePreorder(subtree.left, traversal);
  traversePreorder(subtree.right, traversal);
}
```

Let’s think

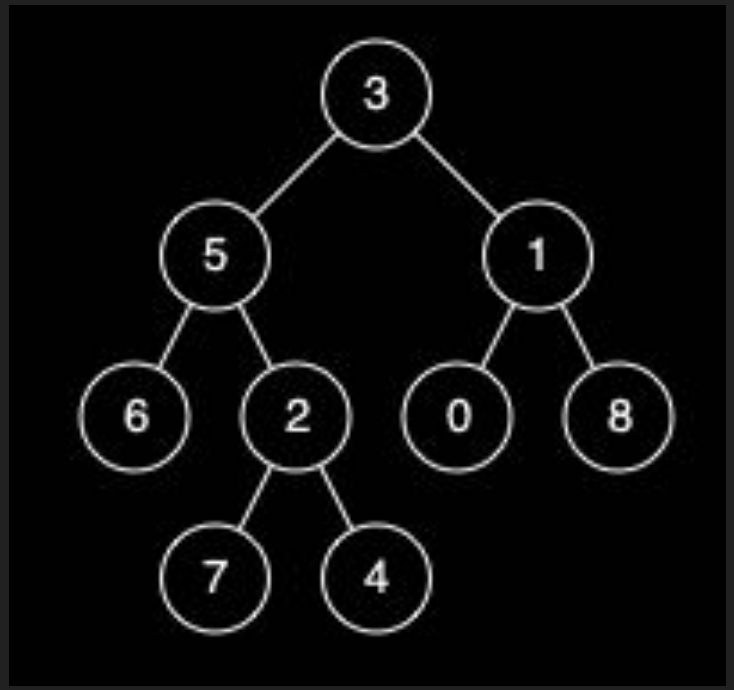
Solution

```java
public boolean pathSum(TreeNode root, int sum) {
    if (root == null) return false;
    if (root.left == null && root.right == null &&
        sum - root.val == 0) {
        return true;
      }
    return pathSum(root.left, sum - root.val) ||
              pathSum(root.right, sum - root.val)
}
```

Harder Problem -- Backtracking required

Find the lowest common ancestor of two given nodes in the tree.

The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself)



Between 6 & 7 → 5
Between 3 & 5 → 3

Post-order traversal Algorithm

```
private void traversePostorder(Node<T> subtree, List<T> traversal) {
  if (subtree == null) {
    return;
  }
  traversePostorder(subtree.left, traversal);
  traversePostorder(subtree.right, traversal);
  traversal.add(subtree.element);
}
```

Let’s think!

```java
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
  if (root == null || root == p || root == q) return root;
  TreeNode left = lowestCommonAncestor(root.left, p, q);
  TreeNode right = lowestCommonAncestor(root.right, p, q);
  if (left == null && right == null) {
    return null;
  }
  if (left != null && right != null) {
    return root;
  }
  return left == null ? right : left;
}
```