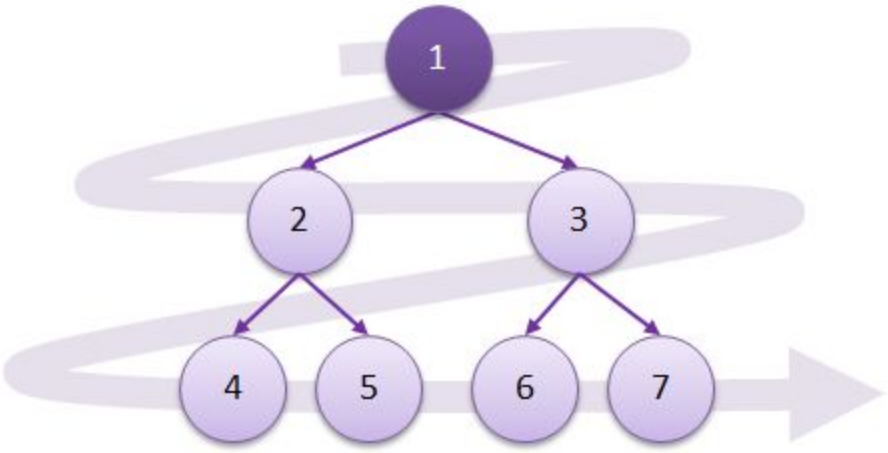
Chapter 5 Tree

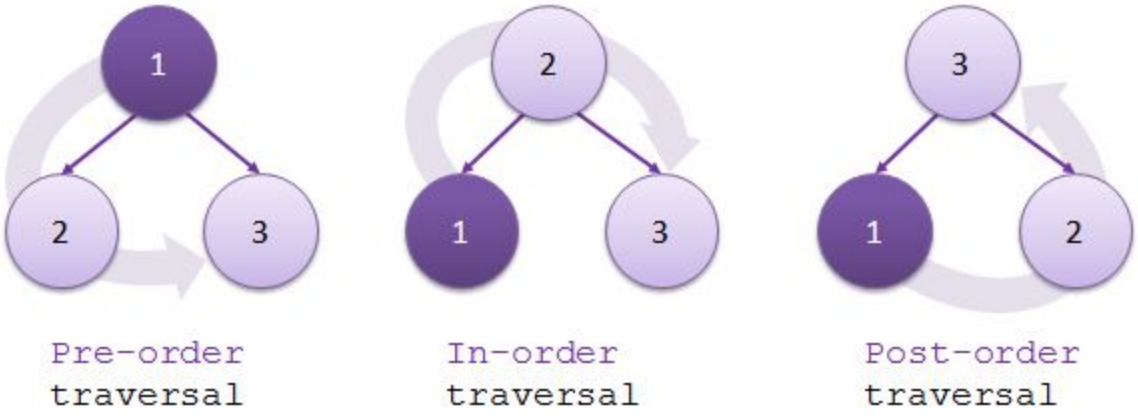
https://leetcode.com/tag/tree/

BFS

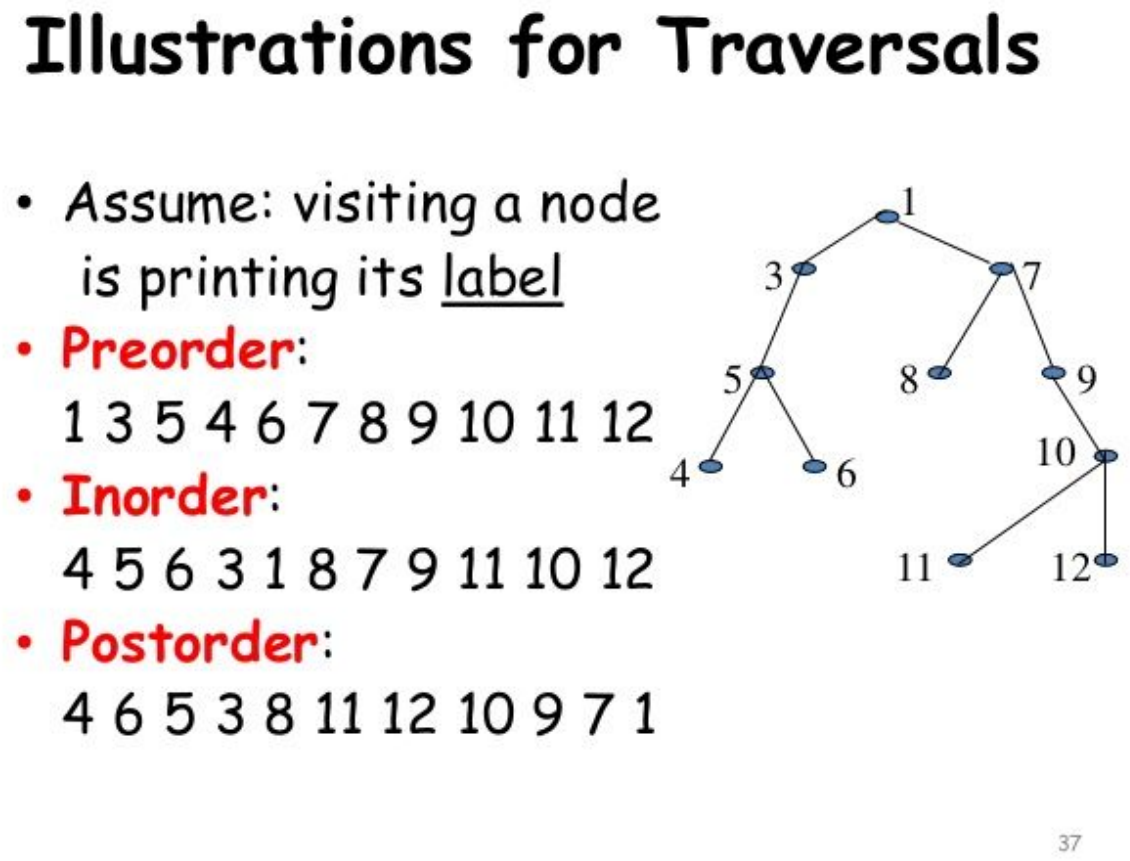


https://www.101computing.net/traversal-of-a-binary-tree/

DFS (中序 vs. 先序 vs. 后序)



https://www.101computing.net/traversal-of-a-binary-tree/



来源：https://image.slidesharecdn.com/7-160527095739/95/7tree-37-638.jpg?cb=1464343078

先序遍历（pre-order traversal）

Maximum Depth of Binary Tree

Given a binary tree, find its maximum depth. The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

```
Given binary tree [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
return its depth = 3.
```

递归遍历

- Time Complexity: O(n)
- Space Complexity: O(1)

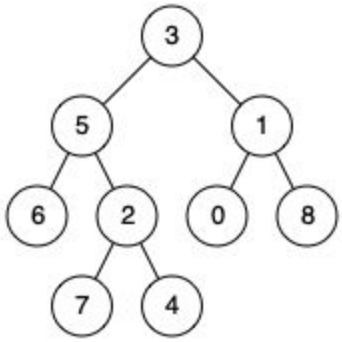
maxDepth(3) -> 1 + Math.max(maxDepth(9), maxDepth(20));
maxDepth(9) -> 1 + Math.max(maxDepth(null), maxDepth(null));
maxDepth(20) -> 1 + Math.max(maxDepth(15), maxDepth(7));
……

```
public int maxDepth(TreeNode root) {
  if (root==null) return 0;
  return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));
}
```

[Lowest Common Ancestor of a Binary Tree]
Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: “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).”

Given the following binary tree: root = [3,5,1,6,2,0,8,null,null,7,4]



Examples:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of of nodes 5 and 1 is 3.

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of
itself according to the LCA definition.
```

递归遍历

- Time Complexity: O(n)
- Space Complexity: O(1)

```
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 root;
    return left != null ? left : right;
}
```

辅助数据结构

- 通过遍历树，我们可以使用HashMap来创建父子节点关系
- 从而获得节点p和节点q的父节点链，再在其中找LCA
- Time Complexity: O(n)
- Space Complexity: O(n)

相似题目：

Flatten Binary Tree to Linked List
Binary Tree Preorder Traversal
Closest Binary Search Tree Value

中序遍历（in-order traversal）

Kth Smallest Element in a BST
Given a **binary search tree**, write a function kthSmallest to find the kth smallest element in it.

Example:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1

Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
```

中序遍历 BST，会按照从小到大遍历树的各个节点

这段代码中，count和result都是state，我们的function没有state

- 全局变量
- 数组

traverse(5) -> traverse(3), count++, traverse(6)
traverse(3) -> traverse(2), count++, traverse(4)
traverse(2) -> traverse(1), count++, **traverse(null)**
traverse(1) -> **traverse(null)**, count++, **traverse(null)**

```
int count = 0;
int result = Integer.MIN_VALUE;

public int kthSmallest(TreeNode root, int k) {
    traverse(root, k);
    Int[] xx = new
    return xx[0];
}

public void traverse(TreeNode root, int k, int[] ) {
    if(root == null) return;
    traverse(root.left, k);
    count ++;
    if(count == k) result = root.val;
    traverse(root.right, k);
}
```

[Convert BST to Greater Tree]

Given a Binary Search Tree (BST), convert it to a Greater Tree such that **every key of the original BST is changed to the original key plus sum of all keys greater than the original key in BST**.

Example:

```
Input: The root of a Binary Search Tree like this:
              5
            /   \
           2     13

Output: The root of a Greater Tree like this:
             18
            /   \
          20     13
```

递归**反向中序**遍历

```
int sum = 0;
public TreeNode convertBST(TreeNode root) {
```

```
    if (root != null) {
        convertBST(root.right);
        sum += root.val;
        root.val = sum;
        convertBST(root.left);
    }
    return root;
}
```

相似题目:
Binary Tree Inorder Traversal
Binary Search Tree Iterator

后序遍历（post-order traversal）

Binary Tree Tilt
Given a binary tree, return the tilt of the whole tree.

The tilt of a tree node is defined as the absolute difference between **the sum of all left subtree node values** and **the sum of all right subtree node values**. Null node has tilt 0.

The tilt of the whole tree is defined as the sum of all nodes' tilt.

Example:

```
Input:
         1
       /   \
      2     3
Output: 1
Explanation:
Tilt of node 2 : 0
Tilt of node 3 : 0
Tilt of node 1 : |2-3| = 1
Tilt of binary tree : 0 + 0 + 1 = 1
```

后序递归遍历

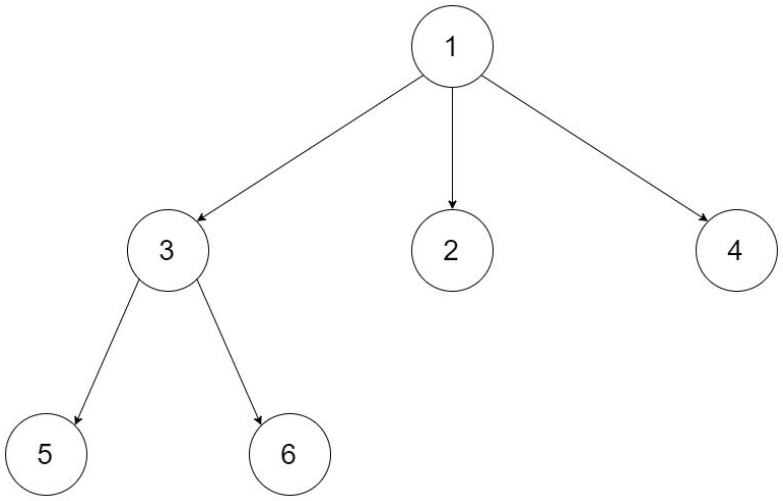
```
int tilt = 0;

public int findTilt(TreeNode root) {
    traverse(root);
    return tilt;
}

public int traverse(TreeNode root)
{
```

```
    if(root==null ) return 0;
    int left = traverse(root.left);
    int right = traverse(root.right);
    tilt += Math.abs(left-right);
    return left + right + root.val;
}
```

[N-ary Tree Postorder Traversal]

Given an n-ary tree, return the postorder traversal of its nodes' values.

For example, given a 3-ary tree:



Return its postorder traversal as: [5,6,3,2,4,1].

```
List<Integer> list = new ArrayList<>();

public List<Integer> postorder(Node root) {
    if (root == null)
        return list;

    for(Node node: root.children)
        postorder(node);

    list.add(root.val);

    return list;
}
```

[Binary Tree Maximum Path Sum]

Given a non-empty binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some **starting node** to **any node** in the tree **along the parent-child connections**. The path must contain at least one node and does not need to go through the root.

Example:

```
Input: [1,2,3]
       1
      / \
     2   3
Output: 6

Input: [-10,9,20,null,null,15,7]
   -10
   / \
  9  20
    /  \
   15   7
Output: 42
```

后序递归遍历

- path只能是一个方向，即向下

```
int maxValue;

public int maxPathSum(TreeNode root) {
    maxValue = Integer.MIN_VALUE;
    maxPathDown(root);
    return maxValue;
}

private int maxPathDown(TreeNode node) {
    if (node == null) return 0;
    int left = Math.max(0, maxPathDown(node.left));
    int right = Math.max(0, maxPathDown(node.right));

    maxValue = Math.max(maxValue, left + right + node.val);

    return Math.max(left, right) + node.val;
}
```

相似题目：

Binary Tree Postorder Traversal

Amazon OA:

Find distance between two nodes of a Binary Tree

Find the distance between two keys in a binary tree, no parent pointers are given. Distance between two nodes is the minimum number of edges to be traversed to reach one node from other.

层序遍历（Level order traversal）

[Binary Tree Level Order Traversal]
Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

Example:

```
Given binary tree [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
return its level order traversal as:
[
  [3],
  [9,20],
  [15,7]
]
```

先序递归遍历

```
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> res = new ArrayList<List<Integer>>();
    levelHelper(res, root, 0);
    return res;
}

public void levelHelper(List<List<Integer>> res, TreeNode root, int height) {
    if (root == null) return;
    if (height >= res.size()) {
        res.add(new LinkedList<Integer>());
    }
    res.get(height).add(root.val);
    levelHelper(res, root.left, height+1);
    levelHelper(res, root.right, height+1);
}
```

循环遍历

- 需要使用辅助数据结构（Queue/Stack）
- Time Complexity: O(n)
- Space Complexity: O(n)

```
public List<List<Integer>> levelOrder(TreeNode root) {
    Queue<TreeNode> queue = new LinkedList<TreeNode>();
```

```
    List<List<Integer>> wrapList = new LinkedList<List<Integer>>();

    if(root == null) return wrapList;

    queue.offer(root);
    while(!queue.isEmpty()){
        int levelSize = queue.size();
        List<Integer> subList = new LinkedList<Integer>();
        for(int i=0; i<levelSize; i++) {
            if(queue.peek().left != null) queue.offer(queue.peek().left);
            if(queue.peek().right != null) queue.offer(queue.peek().right);
            subList.add(queue.poll().val);
        }
        wrapList.add(subList);
    }
    return wrapList;
}
```

[Populating Next Right Pointers in Each Node II]
Given a binary tree

```
struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.
Example:

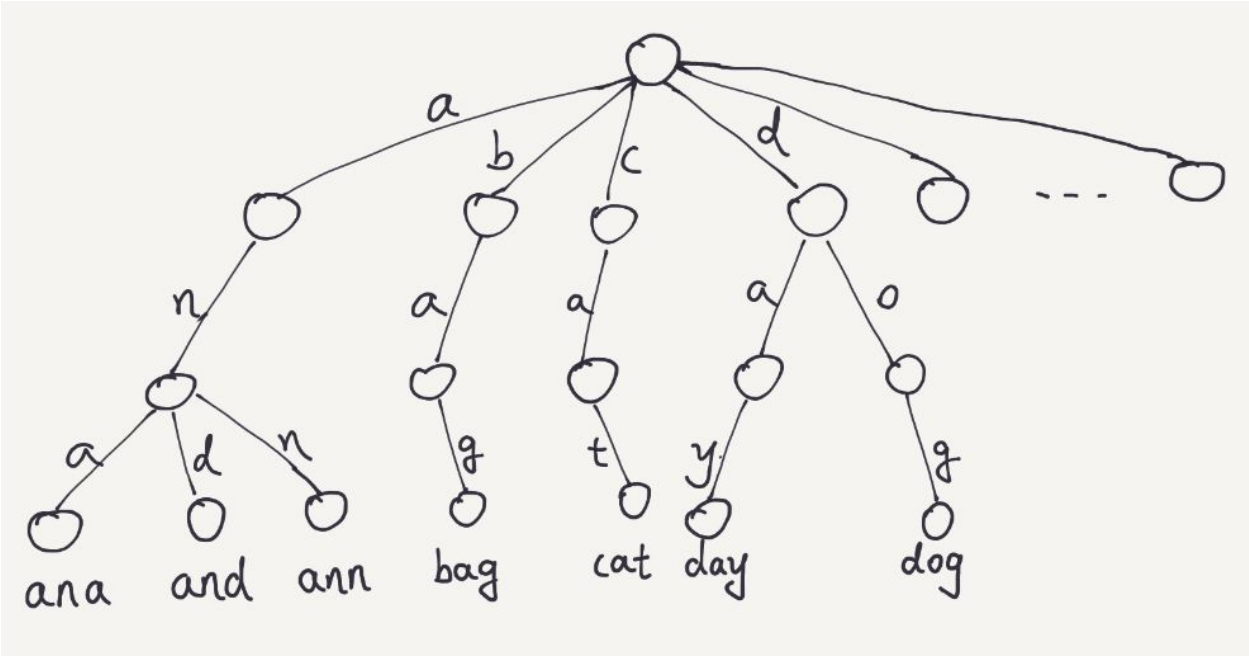
```
Given the following binary tree,
     1
   /  \
  2    3
 / \    \
4   5    7
After calling your function, the tree should look like:
     1 -> NULL
   /  \
  2 -> 3 -> NULL
 / \    \
4-> 5 -> 7 -> NULL
```

改变Input数据的结构，从而优化space complexity

```java
public void connect(TreeLinkNode root) {
    TreeLinkNode dummyHead = new TreeLinkNode(0);
    TreeLinkNode pre = dummyHead;

    while (root != null) {
           if (root.left != null) {
                  pre.next = root.left;
                  pre = pre.next;
           }
           if (root.right != null) {
                  pre.next = root.right;
                  pre = pre.next;
           }

           root = root.next;
           if (root == null) {
                  pre = dummyHead;
                  root = dummyHead.next;
                  dummyHead.next = null;
           }
    }
}
```

相似题目：
Populating Next Right Pointers in Each Node
Find Bottom Left Tree Value
Add One Row to Tree

字典树（Trie）



https://www.programcreek.com/2014/05/leetcode-implement-trie-prefix-tree-java/

Longest Word in Dictionary

Given a list of strings words representing an English Dictionary, **find the longest word in words that can be built one character at a time by other words in words**. If there is more than one possible answer, return the longest word with the smallest lexicographical order.

If there is no answer, return the empty string.

Example:

```
Input:
words = ["w","wo","wor","worl", "world"]
Output: "world"
Explanation:
The word "world" can be built one character at a time by "w", "wo", "wor", and
"worl".

Input:
words = ["a", "banana", "app", "appl", "ap", "apply", "apple"]
Output: "apple"
Explanation:
Both "apply" and "apple" can be built from other words in the dictionary. However,
"apple" is lexicographically smaller than "apply".
```

Brute Force

- 双重循环（遍历每一个string可以衍生出的string，并判断是否在dictionary）

- Time Complexity:
- Space Complexity:

Tire解法

- 通过dictionary建立tire，遍历tire找到高度
- 如何找树的高度？DFS vs. BFS
- Time Complexity:
- Space Complexity: