Tuan Nhu Dinh Follow

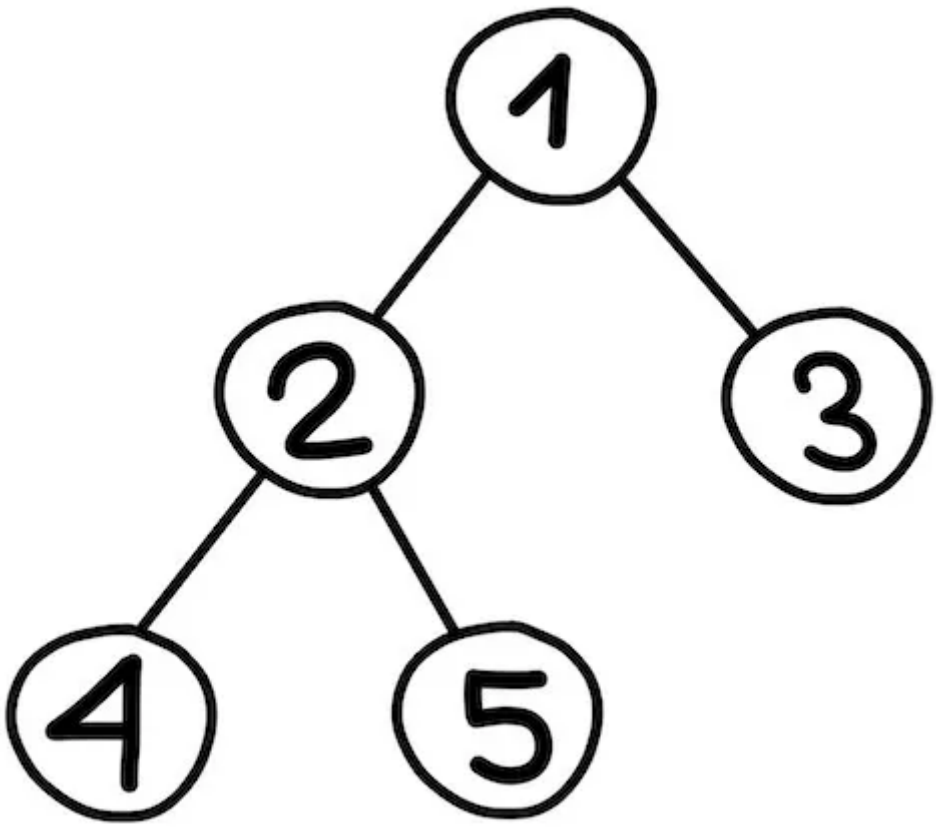
Apr 4, 2020 · 2 min read · ✦ · Listen

Binary tree traversals cheat sheet for coding interviews

Image from Daniel Start

This blog is a part of my “15 days cheat sheet for hacking technical interviews at big tech companies”. In this blog, we discuss different binary tree traversal implementations which play an important role to solve coding interview questions related to tree data structure.

Unlike array, linked list, queues or stacks …, tree data structure can be traversed in different ways. Following are the generally used ways for traversing trees.



1. Inorder traversal: left, node, right (LNR)

```
Inorder traversal for the above tree: 4 2 5 1 3
```

Recursive implementation:

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None

# recursive version
def inorder_traversal_recursive(root):
    result = []

    def recur(node):
        # base case
        if not node:
            return

        # traverse the left subtree
        recur(node.left)
        # visit node
        result.append(node.val)
        # traverse the right subtree
        recur(node.right)

    recur(root)
    return result
```

tree_traversal_inorder.py hosted with ❤️ by GitHub view raw

Iterative implementation using stack:

```python
def inorder_traversal_iterating(root):
    result = []
    # using stack
    stack = []
    current = root
    while stack or current:
        if current:
            # traverse the left subtree
            stack.append(current)
            current = current.left
            continue
        # visit node
        current = stack.pop()
        result.append(current.val)
        # traverse the right subtree
        current = current.right
    return result
```

inorder_traversal_iterating.py hosted with ❤️ by GitHub view raw

2. Preorder traversal: node, left, right (NLR)

Preorder traversal for the above tree: **1 2 4 5 3**

Recursive implementation:

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None

def preorder_traversal_recursive(root):
    result = []

    def recur(node):
        # base case
        if not node:
            return

        # visit node
        result.append(node.val)
        # traverse the left subtree
        recur(node.left)
        # traverse the right subtree
        recur(node.right)

    recur(root)
    return result
```

tree_traversal_preorder.py hosted with ❤️ by GitHub view raw

Iterative implementation using stack:

```python
def preorder_traversal_iterating(root):
    if not root:
        return []

    result = []
    # using stack
    stack = [root]
    while stack:
        current = stack.pop()
        # visit node
        result.append(current.val)
        # put right to stack first as we want to visit right after left!!
        if current.right:
            stack.append(current.right)
        if current.left:
            stack.append(current.left)
    return result
```

tree_traversal_preorder_interative.py hosted with ❤️ by GitHub view raw

3. Postorder traversal: left, right, node (LRN)

Postorder traversal for the above tree: **4 5 2 3 1**

Recursive implementation:

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None

def postorder_traversal_recursive(root):
    result = []

    def recur(node):
        # base case
        if not node:
            return

        # traverse the left subtree
        recur(node.left)
        # traverse the right subtree
        recur(node.right)
        # visit node
        result.append(node.val)

    recur(root)
    return result
```

tree_traversal_postorder.py hosted with ❤️ by GitHub view raw

Iterative implementation using stack. *In this implementation, we first get the items in order of (node, right, left) and reverse the list to get the target output.*

4. Level order traversal

While all the above traversals are kind of depth first traversals (starts at the top node and goes as far as it can down a given branch, then backtracks), the level order traversal is a kind of breadth first traversals.

```
Level order traversal for the above tree:
[
  [1],
  [2,3],
```

```
    [4,5]
]
```

Recursive implementation:

Iterative implementation using queue:

Use cases for different traversals

The time complexity is **O(n)** for all traversal techniques, and each of them may be preferred in different scenarios:

1. Inorder traversal (LNR): in a binary search trees (BST), we can travel the nodes in ascending sorted order by using inorder traversal.

2. Preorder traversal (NLR): it is a topologically sorted order (because a parent node is processed before any of its child nodes is done) and it is usually used to create a

copy of tree or compare two binary trees.

3. Postorder traversal (LRN): it is used to delete the tree as before deleting the parent node we should delete its children nodes first.
4. Level order traversal: it is useful for the questions required to deal with the tree in layer by layer.

Recommended questions

You can practice the tree traversal technique with the following questions:

1. Binary Tree Inorder Traversal
2. Binary Tree Preorder Traversal
3. Binary Tree Postorder Traversal
4. Binary Tree Level Order Traversal
5. Populating Next Right Pointers in Each Node
6. Binary Search Tree Iterator
7. Binary Tree Right Side View
8. Delete Nodes And Return Forest
9. Serialize and Deserialize Binary Tree
10. Boundary of Binary Tree

Binary Tree Coding Interviews In Order Traversal Computer Science

Tree Data Structures