Trees: DFS and BFS

Depth First Search + Breadth First Search

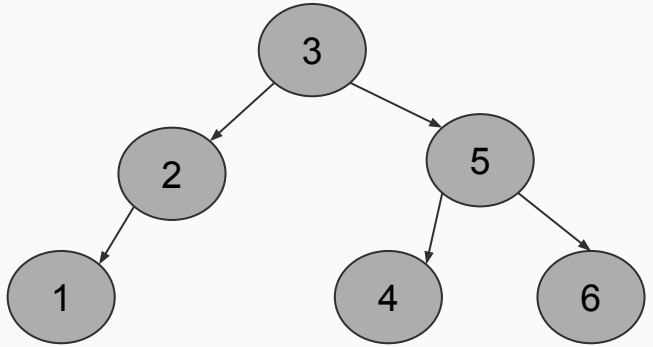
- Some of THE most important and common algorithms that you can learn
- DFS+BFS can solve large majority of tree and graph problems by themselves
- Is the foundation for many more complicated algorithms such as Djikstra or topological sorting if you choose to (not necessary for 99% of interviews!!!)

Depth First Search

- Searches as deep into one path as possible before searching a different path
- 3 types:
  - In-order traversal
  - Pre-order traversal
  - Post-order traversal
- Utilizes a stack
  - This makes recursion perfect since recursion inherently uses a stack via the call stack!

In-Order DFS

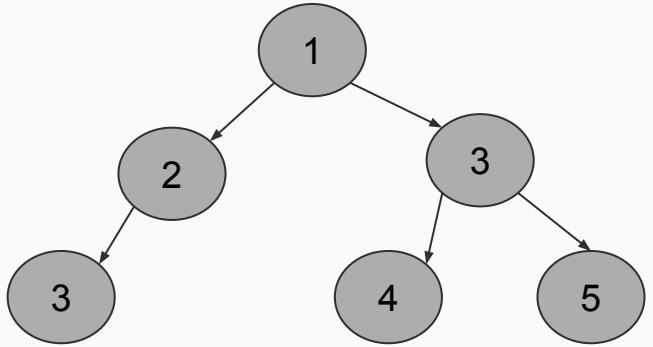
- Steps
  - Traverse left sub tree
  - Process node
  - Traverse right sub tree
- Demo
  - Binary Tree Inorder Traversal




```
def inorderTraversal(root):
    if not root: return
    inorderTraversal(root.left)
    print(root.val)
    inorderTraversal(root.right)
```

Pre-Order DFS

- Steps
  - Process node
  - Traverse left sub tree
  - Traverse right sub tree
- Demo
  - Binary Tree Preorder Traversal

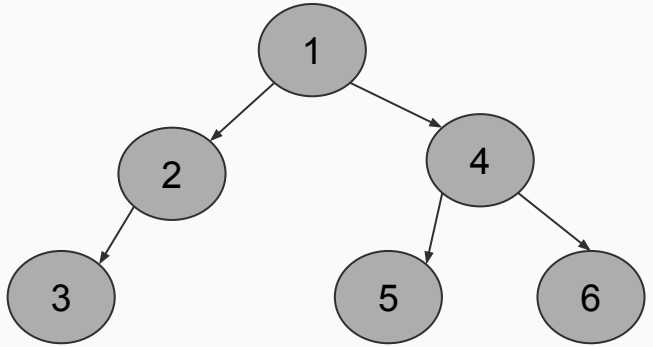



```
def preorderTraversal(root):
    if not root) return
    print(root.val)
    inorderTraversal(root.left)
    inorderTraversal(root.right)
```

Post-Order DFS

- Steps
  - Traverse left sub tree
  - Traverse right sub tree
  - Process node
- Demo
  - Binary Tree Postorder Traversal

- Follow up
  - Can we reverse the order of our traversal?




```
def postorderTraversal(root):
    if not root: return
    inorderTraversal(root.left)
    inorderTraversal(root.right)
    print(root.val)
```

DFS Runtime

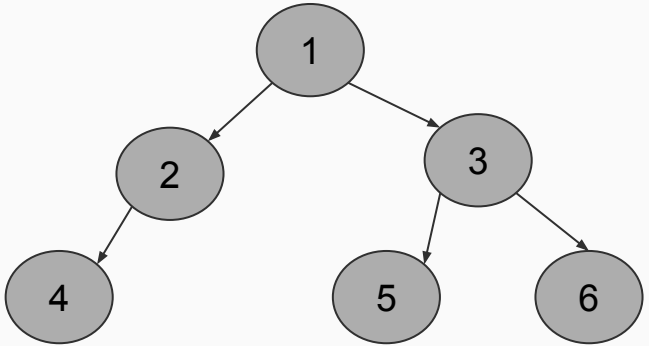
Operations	Big-O Time
Search tree	O(n)

Breadth First Search (BFS)

- Searches through nodes level by level
- We process nodes closest to the root first
- This algorithm is very useful when finding shortest paths or when dealing with layers
- Utilizes a queue
    - Because recursion uses a stack, we always want to just run BFS iteratively

Breadth First Search (cont.)

- Steps
  - Add the root to the queue
  - Shift out a “current node” from the queue
  - Process current node
  - Push current node’s children into the queue
  - Repeat process until queue is empty
- Demo
  - Binary Tree Level Order Traversal



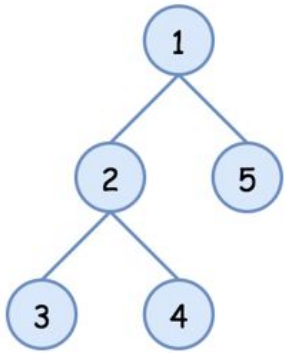
BFS Runtime

Operations	Big-O Time
Search tree	O(n)

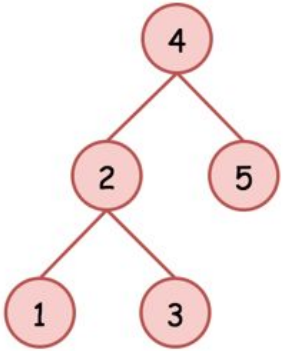
Traversals

DFS Preorder	DFS Inorder	DFS Postorder	BFS
Node -> Left -> Right	Left -> Node -> Right	Left -> Right -> Node	Node -> Left -> Right

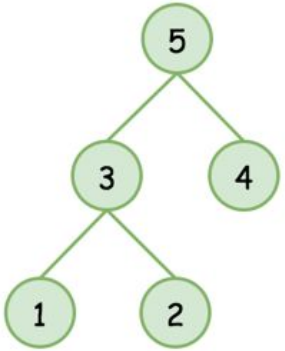
Traversal = [1, 2, 3, 4, 5]



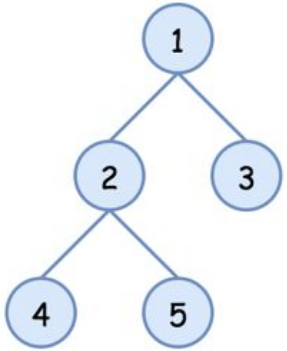
[root.val] +
preorder(root.left) +
preorder(root.right)
if root else []



inorder(root.left) +
[root.val] +
inorder(root.right)
if root else []



postorder(root.left) +
postorder(root.right) +
[root.val]
if root else []



iterations
with the queue

Questions?

Let’s practice!

- Review
  - Same Tree
  - Maximum Depth of Binary Tree
  - Count Complete Tree Nodes
- Bonus
  - Path Sum
  - Invert Binary Tree

Iterative Depth First Search

- Though recursion is works very naturally for DFS, we can do it iteratively too!
- Instead of using the call stack, we would simply create our own stack and run it similarly as how we run our queue for BFS
- It is not recommended to use iterative DFS most of the time because there is a lot more code and has the same time and space complexities. It can also overcomplicate some problems.
- Demo:
    - Binary Tree Level Order Traversal

Continued Practice!

- Review
  - Path Sum
  - Count Good Nodes in Binary Tree
  - Binary Tree Right Side View
  - Invert Binary Tree
- Bonus
  - Subtree of Another Tree
  - Binary Tree Level Order Traversal II - do this one without using .reverse()!