**N-ary Tree Postorder Traversal**

* [Solution](https://leetcode.com/explore/learn/card/n-ary-tree/130/traversal/1787/#solution)
  + [How to traverse the tree](https://leetcode.com/explore/learn/card/n-ary-tree/130/traversal/1787/#how-to-traverse-the-tree)
  + [Approach 1: Iterations](https://leetcode.com/explore/learn/card/n-ary-tree/130/traversal/1787/#approach-1-iterations)

Solution

How to traverse the tree

There are two general strategies to traverse a tree:

* *Breadth First Search* (BFS)

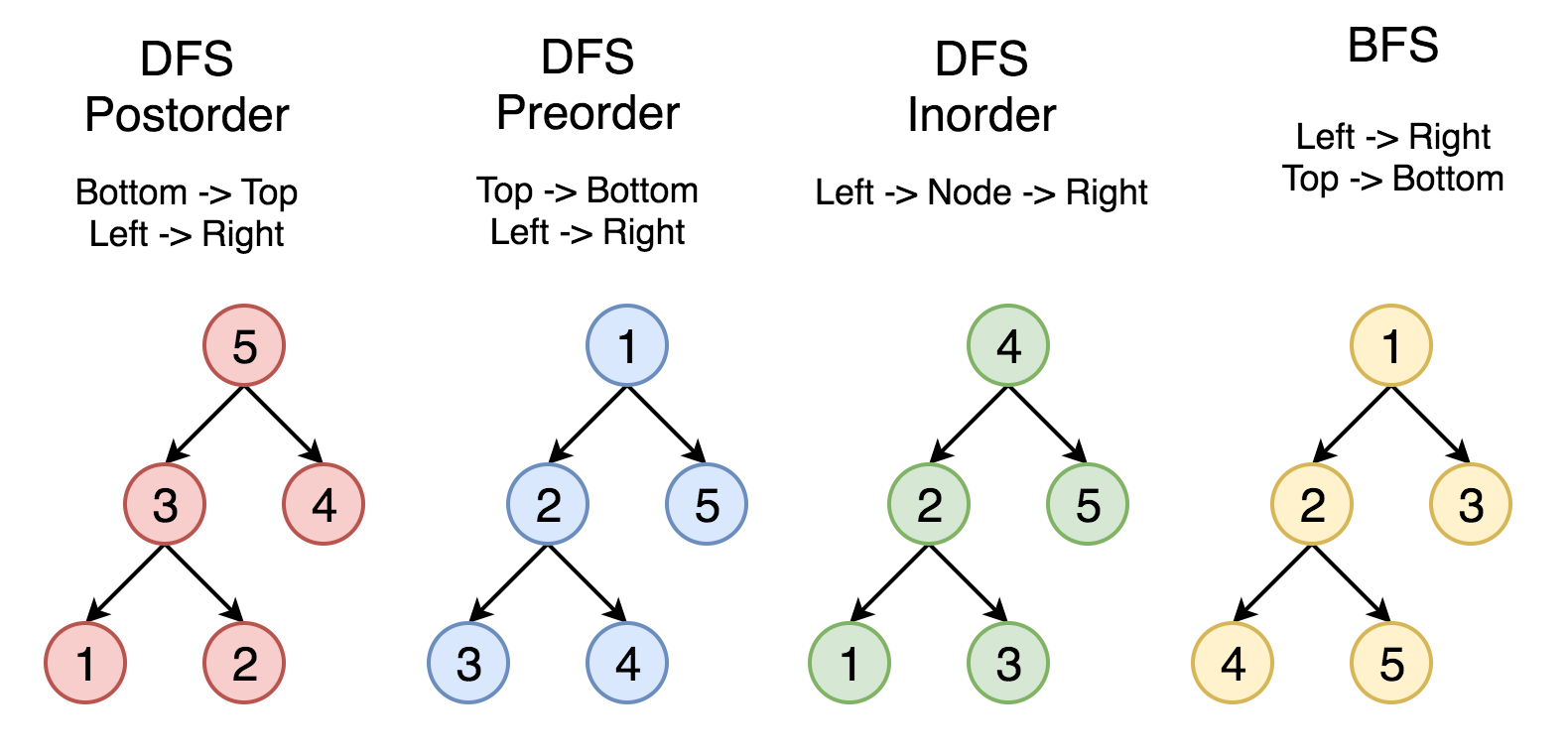
We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.

* *Depth First Search* (DFS)

In this strategy, we adopt the depth as the priority, so that one would start from a root and reach all the way down to certain leaf, and then back to root to reach another branch.

The DFS strategy can further be distinguished as preorder, inorder, and postorder depending on the relative order among the root node, left node and right node.

On the following figure the nodes are numerated in the order you visit them, please follow 1-2-3-4-5 to compare different strategies.



Here the problem is to implement postorder traversal using iterations.

Approach 1: Iterations

**Algorithm**

First of all, here is the definition of the TreeNode which we would use in the following implementation.

Let's start from the root and then at each iteration pop the current node out of the stack and push its child nodes. In the implemented strategy we push nodes into stack following the order Top->Bottom and Left->Right. Since DFS postorder traversal is Bottom->Top and Left->Right the output list should be reverted after the end of loop.



Note: with this iterative approach, we aren't technically traversing the tree in a postorder manner. However, we are still obtaining the values in a postorder ordering, which is what the problem wants.

**Complexity Analysis**

* Time complexity : we visit each node exactly once, thus the time complexity is O(*N*), where *N* is the number of nodes, *i.e.* the size of tree.
* Space complexity : depending on the tree structure, we could keep up to the entire tree, therefore, the space complexity is O(*N*).