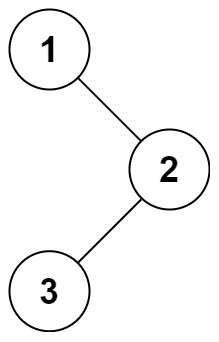
<https://leetcode.com/problems/binary-tree-preorder-traversal/>

Given the root of a binary tree, return *the preorder traversal of its nodes' values*.

**Example 1:**



Input: root = [1,null,2,3]

Output: [1,2,3]

**Example 2:**

Input: root = []

Output: []

**Example 3:**

Input: root = [1]

Output: [1]

**Constraints:**

* The number of nodes in the tree is in the range [0, 100].
* -100 <= Node.val <= 100

**Follow up:** Recursive solution is trivial, could you do it iteratively ?

**Attempt 1: 2022-10-23**

**Solution 1:  Recursive traversal (10min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<Integer> preorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

if(root == null) {

return result;

}

// No modification on tree structure, can use original object 'root' to traverse

helper(root, result);

return result;

}

private void helper(TreeNode root, List<Integer> result) {

if(root == null) {

return;

}

result.add(root.val);

helper(root.left, result);

helper(root.right, result);

}

}

**Solution 2: Iterative traversal with Stack style 1 (10min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<Integer> preorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

if(root == null) {

return result;

}

Stack<TreeNode> stack = new Stack<TreeNode>();

stack.push(root);

while(!stack.isEmpty()) {

TreeNode node = stack.pop();

result.add(node.val);

// Must push right first then push left as we need stack pop out left first

if(node.right != null) {

stack.push(node.right);

}

if(node.left != null) {

stack.push(node.left);

}

}

return result;

}

}

**Solution 3: Iterative traversal with Stack style 2 (10min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<Integer> preorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

if(root == null) {

return result;

}

// Note that in this solution only right children are stored to stack.

Stack<TreeNode> stack = new Stack<TreeNode>();

// No modification on tree structure, can use original object 'root' to traverse

while(root != null || !stack.isEmpty()) {

if(root != null) {

result.add(root.val);

stack.push(root.right);

root = root.left;

} else {

root = stack.pop();

}

}

return result;

}

}

**Refer to**

<https://leetcode.com/problems/binary-tree-preorder-traversal/discuss/45266/Accepted-iterative-solution-in-Java-using-stack./44818>