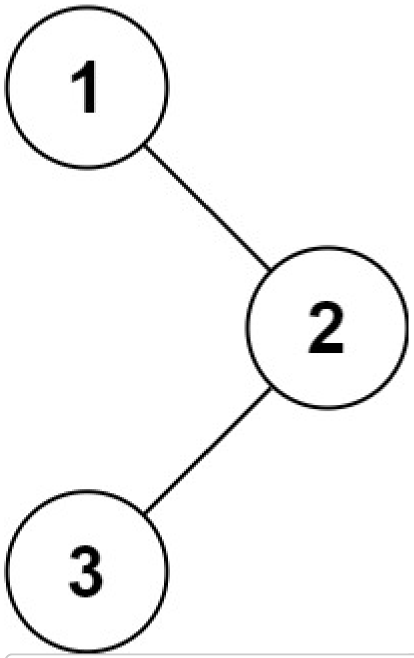
94. Binary Tree Inorder Traversal

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

Example 1:



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

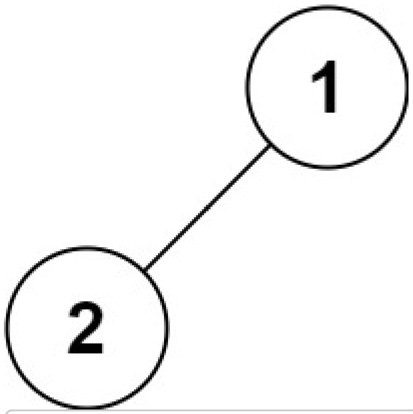
Example 2:

```
Input: root = []
Output: []
```

Example 3:

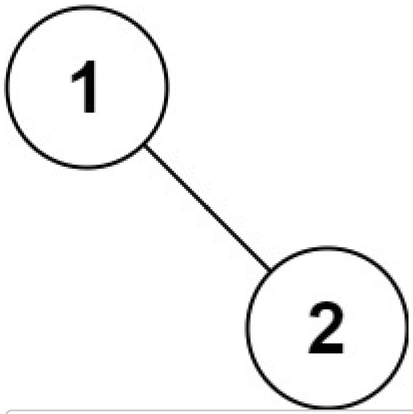
```
Input: root = [1]
Output: [1]
```

Example 4:



```
Input: root = [1,2]
Output: [2,1]
```

Example 5:



```
Input: root = [1,null,2]
Output: [1,2]
```

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?

中序遍历二叉树

iteratively traversal (stack)

1. `while (!stack.isEmpty() || root != null)` push root.left until it reaches null

2. pop one node then add it to inOrder, finally `node = node.right`
3. return inOrder

---

105. Construct Binary Tree from Preorder and Inorder Traversal

Given preorder and inorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

For example, given

```
preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]
```

Return the following binary tree:

```
    3
   / \
  9  20
    /  \
   15   7
```

前序和中序数组构建二叉树

DFS, Array

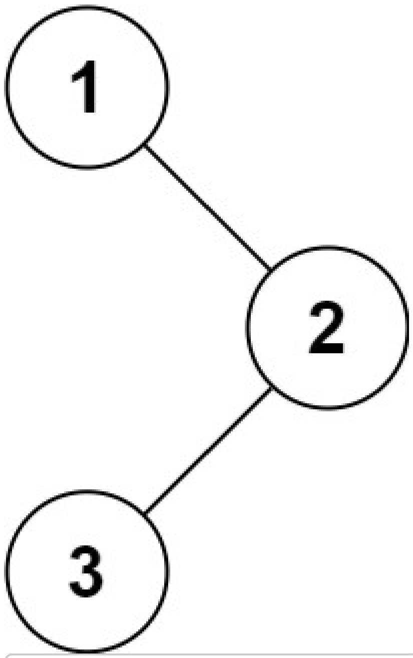
1. store inorder into a hashmap with (element, index)
2. dfs(preorder, hashmap, preStart(0), inStart(0), inEnd(length-1)), locate the positions of three indexes to build the left and right subtree
3. **root = preorder[preStart]**
4. find its index in inorder array, its left part is the left subtree and right part is the right subtree
5. dfs(preStart+1, inStart, index-1) and dfs(preStart+1+index-inStart, index+1, inEnd), quit the dfs until **preStart > length** and **inStart > inEnd**
6. return the node

---

144. 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]
```

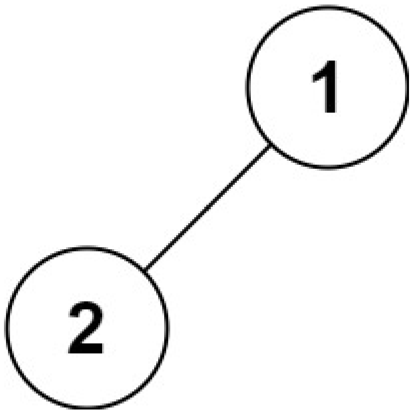
Example 2:

```
Input: root = []
Output: []
```

Example 3:

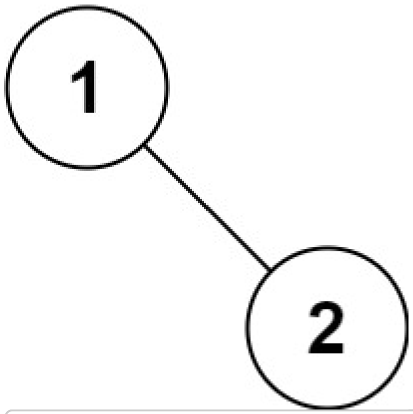
```
Input: root = [1]
Output: [1]
```

Example 4:



```
Input: root = [1,2]
Output: [1,2]
```

Example 5:



```
Input: root = [1,null,2]
Output: [1,2]
```

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?

前序遍历二叉树

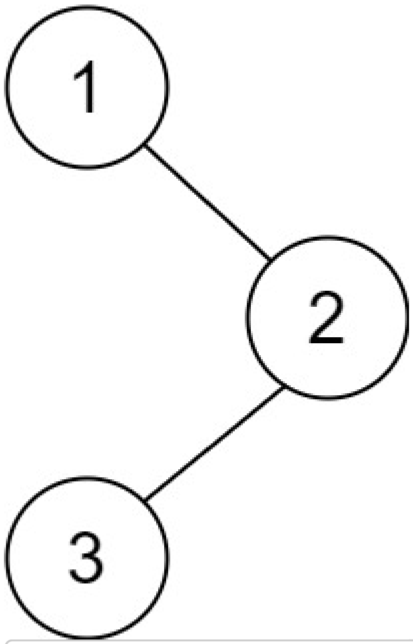
iteratively traversal (stack)

1. push root into stack
2. while stack not empty, then pop one node add its right and left into stack finally add its value to preOrder
3. return preOrder

145. Binary Tree Postorder Traversal

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

Example 1:



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

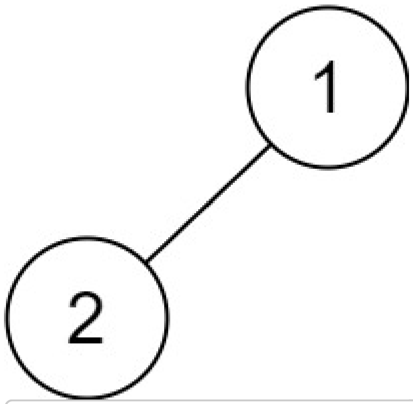
Example 2:

```
Input: root = []
Output: []
```

Example 3:

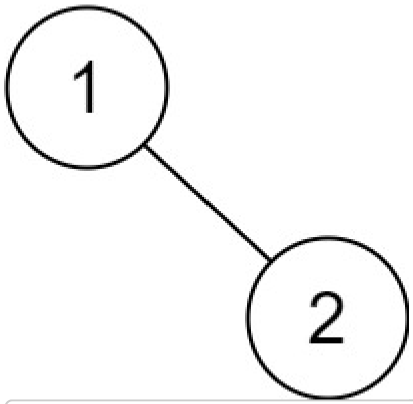
```
Input: root = [1]
Output: [1]
```

Example 4:



```
Input: root = [1,2]
Output: [2,1]
```

Example 5:



```
Input: root = [1,null,2]
Output: [2,1]
```

Constraints:

- The number of the 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?

后序遍历二叉树

iteratively traversal (stack)

1. push root into stack
2. loop while stack **isNotEmpty**, pop one node then add its left and right into stack, finally **addFirst** its value to the front of postOrder
3. return postOrder