

094. Binary Tree Inorder Traversal

094 Binary Tree Inorder Traversal

- Hash Table + tree
- Stack + tree

Description

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

For example:

Given binary tree `[1,null,2,3]`,

```
  1
   \
    2
   /
  3
```

return `[1,3,2]`.

1. Thought line

(1) Stack

inorder: left \rightarrow root \rightarrow right.

Stack 思路:

1. 对 \forall node

↳ 找到最左点, 并将一路经过的点加入 toVisit

↳ 当最左点为空时, 取 toVisit.top(), 压入, 去其右子树

\Rightarrow Cur: starts from Root

rootJumped: starts from empty.

\Rightarrow To each node: Cur

(1) If Cur = nullptr:

- visit rootJumped.top() // Go to Cur's parent node
- Replace "Cur = rootJumped.top()"
- Push Cur \rightarrow val into Result
- Go to Right Child to start over // Inorder: left child \rightarrow node \rightarrow right child

(2) If Cur != NULL

- Push Cur into rootJumped // Cur is exist as a root node
- Go to Cur' left child to start over

2. Stack+tree

```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8  * };
9  */
10 class Solution {
11 public:
12     vector<int> inorderTraversal(TreeNode* root) {
13         stack<TreeNode*> rootJumped;
14         vector<int> result;
15         TreeNode* cur = root;
16         while(cur || !rootJumped.empty()){
17             if (cur){
18                 rootJumped.push(cur);
19                 cur = cur->left;
20             }
21             else{
22                 cur = rootJumped.top();
23                 rootJumped.pop();
24                 result.push_back(cur->val);
25                 cur = cur->right;
26             }
27         }
28     }
29     return result;
30 }
```

