144. Binary Tree Preorder Traversal

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

For example: Given binary tree `{1,#,2,3}`,

```
   1
    \
     2
    /
   3
```

return `[1,2,3]`.

Note: Recursive solution is trivial, could you do it iteratively?

Tags: Tree Stack

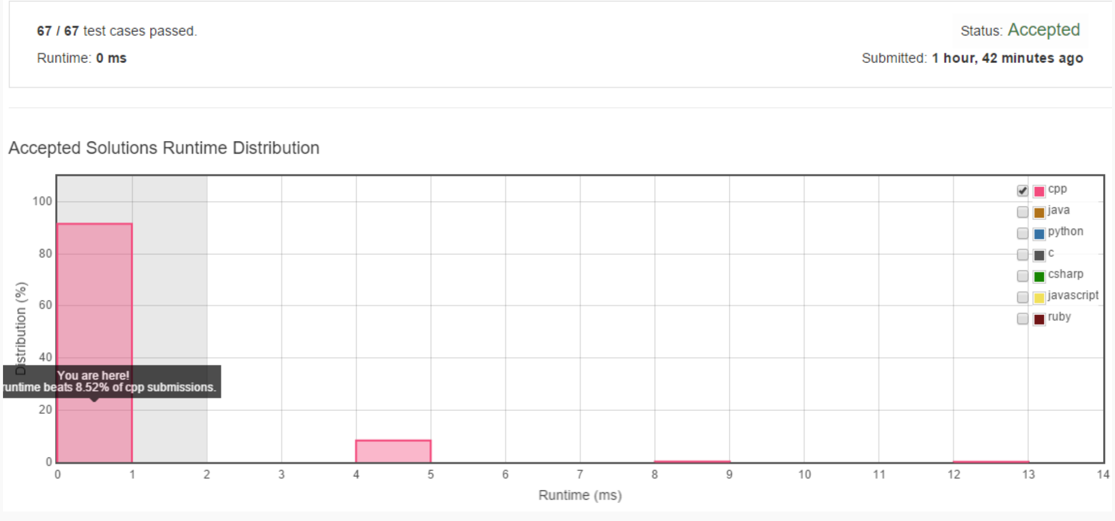
```
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> re;
        if (!root) return re;
        stack<TreeNode*> myStack;
        myStack.push(root);
        TreeNode* currNode = NULL;
        while (!myStack.empty()) {
            currNode = myStack.top();
            re.push_back(currNode->val);
            myStack.pop();
            if (currNode->right) myStack.push(currNode->righ
t);
            if (currNode->left) myStack.push(currNode->lef
t);
```

```
        }
        return re;
    }
};
```

思路:

该题目已标注不能使用递归的方法做。因此可以很自然的想到用栈`(stack)`去实现。

因为前序遍历的顺序是`根--左--右`，这也是我们出栈的顺序。那么想让出栈顺序为`根--左--右`，可以先让根入栈，然后根先出栈，再让根的孩子入栈。这就构成了整个算法的思路。其终止条件为，栈为空，栈空则意味着我们所有的节点已经遍历完。



67 / 67 test cases passed.

Runtime: **0 ms**

Status: Accepted

Submitted: **1 hour, 42 minutes ago**

Accepted Solutions Runtime Distribution

You are here!
untime beats 8.52% of cpp submissions.

145. Binary Tree Postorder Traversal

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

For example:

Given binary tree `{1,#,2,3}`,

```
   1
    \
     2
    /
   3
```

return `[3,2,1]`.

Note: Recursive solution is trivial, could you do it iteratively?

Tags: Tree Stack

思路 1:

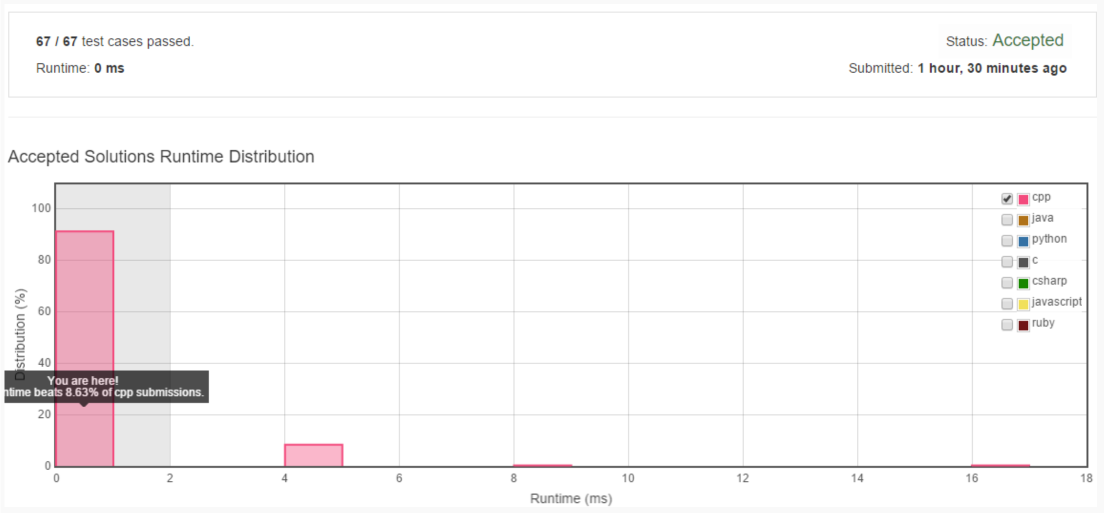
后序遍历的顺序是`左--右--根`，如果我们更改一下前序遍历，把它变成`根--右--左`，然后再将结果反向，就成了`左--右--根`，得到最终结果。

```
class Solution {
public:
    vector<int> postorderTraversal(TreeNode *root) {
        stack<TreeNode*> nodeStack;
        vector<int> re;
        if(root==NULL) return re;
        nodeStack.push(root);
        while(!nodeStack.empty()) {
            TreeNode* node= nodeStack.top();
            re.push_back(node->val);
            nodeStack.pop();
            if(node->left)
            nodeStack.push(node->left);
            if(node->right)
            nodeStack.push(node->right);
        }
        reverse(re.begin(),re.end());
        return re;
    }
};
```

思路 2:

对于后序遍历，当从栈中 pop 节点的时候进行访问。Last_pop 代表最后一次 pop 的节点。如果 lastpop != top->left，意味着该节点的左子树还没有被 push 进栈中，因此要将其左孩子入栈；如果 last_pop == top->left，就要将右孩子入栈；否则，直接 pop 栈顶节点。

```
class Solution {
public:
    vector<int> postorderTraversal(TreeNode *root) {
        vector<int> re;
        if (!root) return re;
        stack<TreeNode*> s;
        s.push(root);
        TreeNode* last_pop = root;
        TreeNode* top = NULL;
        while (!s.empty()) {
            top = s.top();
            if (top->left && top->left != last_pop && top->r
ight != last_pop) { // push_left
                s.push(top->left);
            } else if (top->right && top->right != last_pop && (top->left == NULL || top->left == last_pop)) { // push_right
                s.push(top->right);
            } else { // pop
                s.pop();
                last_pop = top;
                re.push_back(top->val); // visit top
            }
        }
        return re;
    }
};
```



67 / 67 test cases passed.

Runtime: 0 ms

Status: Accepted

Submitted: 1 hour, 30 minutes ago

Accepted Solutions Runtime Distribution

199. Binary Tree Right Side View

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

For example:

Given the following binary tree,

```
   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
```

You should return `[1, 3, 4]`.

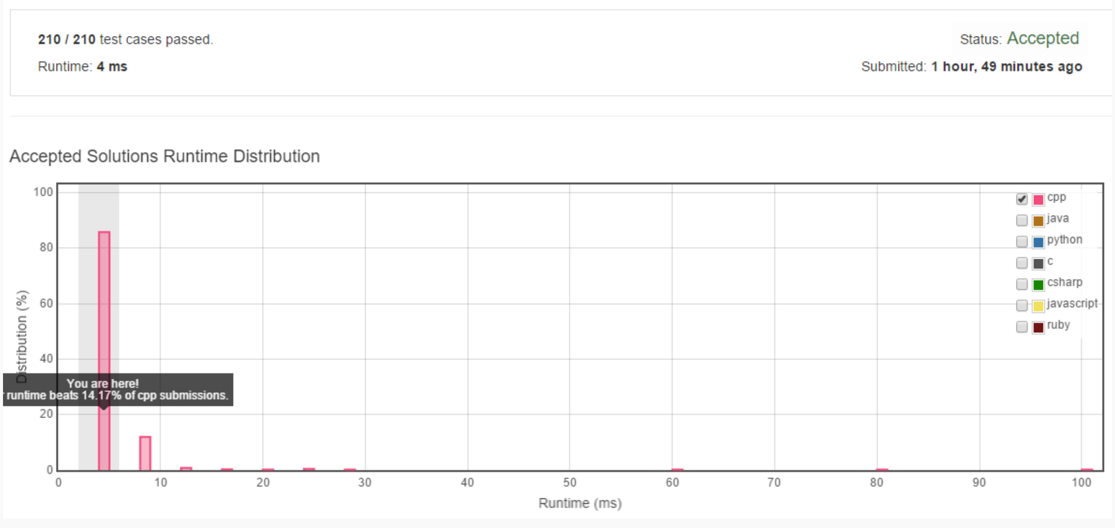
Tags: Tree, Depth-first Search, Breadth-first Search

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) { //BFS
        vector<int> re;
        if (!root) return re;
        queue<TreeNode*> myQueue;
        myQueue.push(root);
        TreeNode* currNode = NULL;
        TreeNode* endNode = root; // the last node addr of o
ne level
        while (!myQueue.empty()) {
            currNode = myQueue.front();
            myQueue.pop();
            if (currNode->left) myQueue.push(currNode->lef
t);
```

```
            if (currNode->right) myQueue.push(currNode->right);
            if (currNode == endNode) {
                re.push_back(currNode->val);
                endNode = myQueue.empty() ? endNode : myQueue.back();
            }
        }
        return re;
    }
};
```

思路:

此题的意思是要找到树每层的最后一个节点。直观的想法是，把每层的节点依次遍历，找到最后一个节点，然后放到结果列表里。因此可以采用 BFS。BFS 是用队列实现的，显然根节点是第一层的最后一个节点。关键在于知道队列里哪个节点是该层的最后一个节点。根据 BFS 的实现方法，当上一层的最后一个节点出栈时，首先会将该节点的左孩子、右孩子入栈，如果其左孩子和右孩子存在的话。那么完成之后，就意味着上一层的节点已全部出队，下一层的节点已全部入队。因此现在队列中的最后一个元素则是该层的最后一个元素。当 BFS 运行完毕以后，结果已经全部存入 vector 中。



210 / 210 test cases passed.
Runtime: 4 ms

Status: Accepted
Submitted: 1 hour, 49 minutes ago

Accepted Solutions Runtime Distribution