Inorder tree traversal without recursion or stack (Morris Traversal)

binary trees BST Morris threading tree traversal

Inorder tree traversal without recursion or stack (fast traversal) is typically called the Morris traversal and this is based on the idea of threaded binary trees.

The problem with typical inorder traversal is that they are implemented using recursion or an explicit stack. If the tree is fairly balanced the stack space needed is `O(log n)` but for skewed trees this can incur massive stack space requirements, in the order of `O(n)`. Fast traversal eliminates this problem by using thread pointers that point to the successor of the node.

A threaded binary tree is defined as follows --
A binary tree is threaded by making all right child pointers that would normally be null point to the inorder successor of the node (if it exists), and all left child pointers that would normally be null point to the inorder predecessor of the node.

It is also possible to identify the parent of a node using a threaded tree without the parent pointers or a stack albeit slowly. The following code demonstrates a possible implementation.

```
def parent(node):
    if node is node.tree.root:
        return None
    else:
        x = node
        y = node
        while True:
            if is_thread(y):
                p = y.right
                if p is None or p.left is not node:
                    p = x
                    while not is_thread(p.left):
                        p = p.left
                    p = p.left
                return p
            elif is_thread(x):
                p = x.left
                if p is None or p.right is not node:
                    p = y
                    while not is_thread(p.right):
                        p = p.right
                    p = p.right
                return p
            x = x.left
            y = y.right
```

Algorithm for morris traversal (Inorder).

```
vector<int> Solution::inorderTraversal(TreeNode* A) {
```

```cpp
    vector<int> traversal;
    TreeNode* curr = A;
    while(curr != NULL){
        if(curr->left == NULL){
            traversal.push_back(curr->val);
            curr = curr->right;
        }
        else{
            TreeNode* pre = curr->left;
            while(pre->right != NULL && pre->right != curr){
                pre = pre->right;
            }
            if(pre->right == NULL){
                pre->right = curr;
                curr = curr->left;
            }
            else{
                pre->right = NULL;
                traversal.push_back(curr->val);
                curr = curr->right;
            }
        }
    }
    return traversal;
}
```

Algorithm for morris traversal (Preorder).

```cpp
vector<int> Solution::preorderTraversal(TreeNode* A) {
    vector<int> traversal;
    TreeNode* curr = A;
    while(curr != NULL){
        if(curr->left == NULL){
            traversal.push_back(curr->val);
            curr = curr->right;
        }
        else{
            TreeNode* pre = curr->left;
            while(pre->right != NULL && pre->right != curr)
                pre = pre->right;
            if(pre->right == NULL){
                traversal.push_back(curr->val);
                pre->right = curr;
                curr = curr->left;
            }
            else{
                pre->right = NULL;
                curr = curr->right;
            }
        }
    }
    return traversal;
}
```