PostOrder Traversal in a Binary Tree (With C Code)

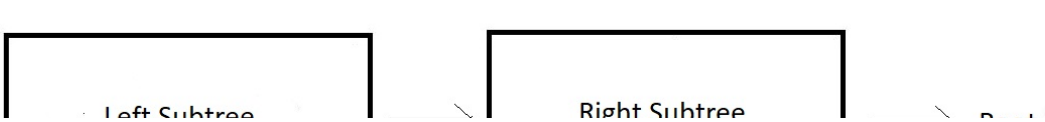
Show Course Contents

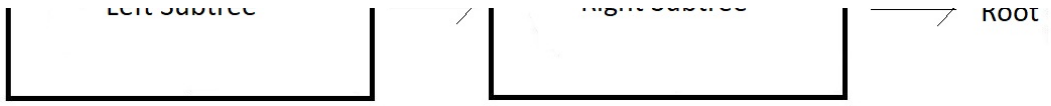
Overview Q&A Downloads Announcements

PostOrder Traversal in a Binary Tree (With C Code)

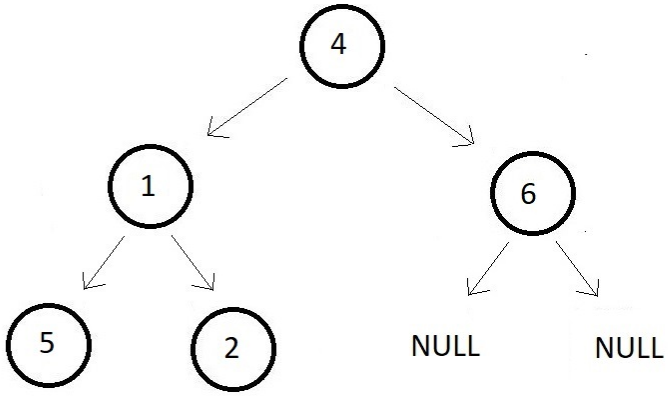
In the last lecture, we discussed in detail our first traversal technique, the PreOrder Traversal. We saw its programming implementation in C. If you haven't checked it out already, I would suggest doing so before proceeding. You can then get an idea of the general approach we are taking. Today, we'll see the PostOrder Traversal in detail, and also cover its programming part.

We clearly have a basic idea of how PostOrder traversal works. I had asked you all previously to explore the flow of this technique yourselves. Today we will see that in detail. So, here you first start with the left subtree and consider that as another tree in itself. You then apply PostOrder on this subtree. You then swiftly move to the right subtree and repeat the whole thing again considering this as another individual tree. At the end, you visit the root node of the main tree and here you finish visiting the whole tree.

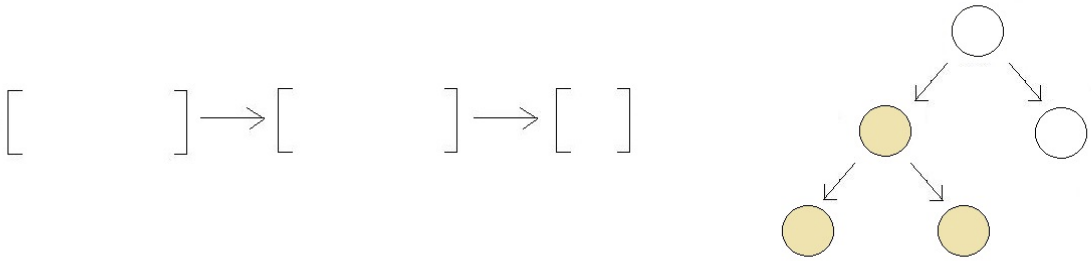




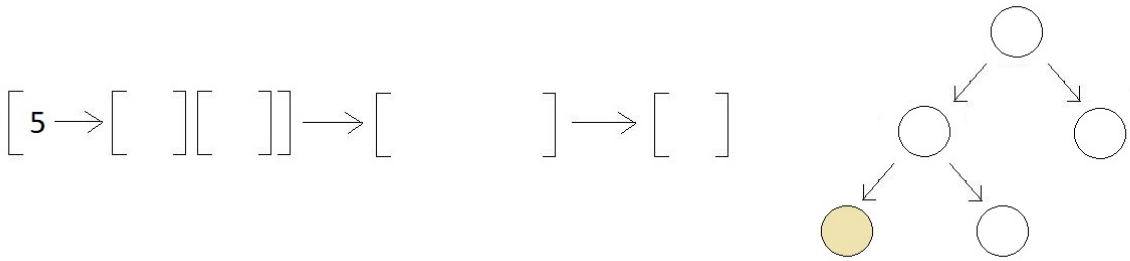
I would like to follow the same approach as I did in my last lecture. First, we will take an example binary tree, and apply PostOrder Traversal on the same.



In the first step, you visit the left subtree considering it as a totally different tree. But before we start with the left subtree, we mark the presence of the right subtree and the root as following:

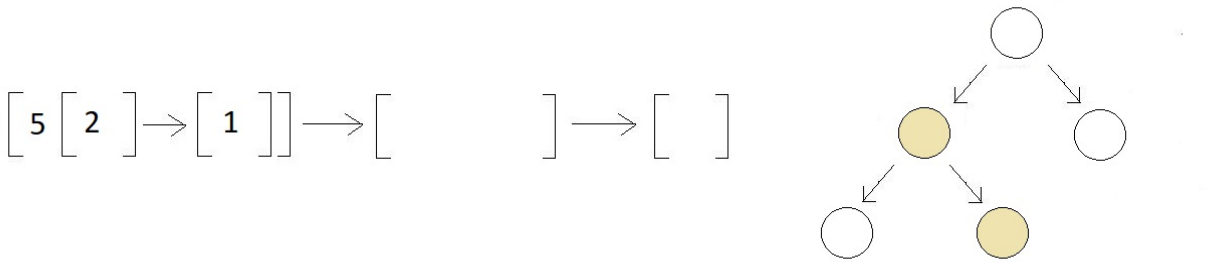


Now, you go through the left subtree, and visit its left node first.

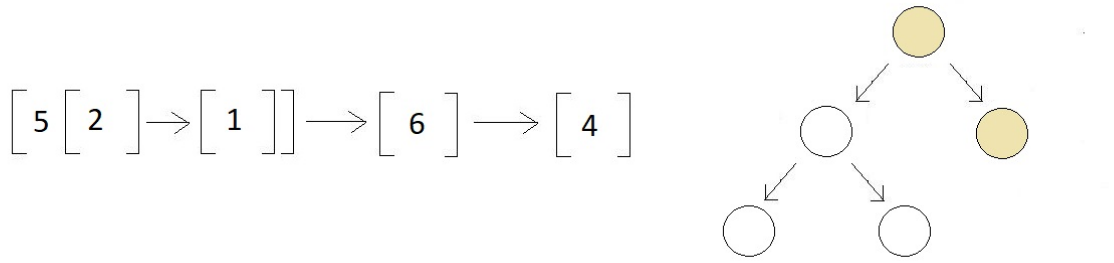


And then you proceed further with the right subtree of this new tree we considered. And since the right subtree of this tree has just a single element, you finish

visiting it and then the root of the node, and return back to our original tree.



And then, we visit the right subtree, and since it contains no left or right subtree further, we finish visiting our right subtree and then move to the root. And there we mark our completion of PostOrder traversal.



And our final order of postorder traversal is: 5 → 2 → 1 → 6 → 4.

Following a thorough study of its flow, we are now ready to implement its programming. I have attached the source code below. Follow it as we proceed.

Understanding the code snippet below:

1. Following what we did before, we wouldn't start from scratch creating the struct Node and the createNode function and everything. We would just copy the whole thing we did in the PreOrder Traversal programming part and paste them here. This would save us a lot of time and help us compare both Pre and Post Order traversals.
2. Create all the five nodes, using the createNode function, and link them using the arrow operator and altering their left and right pointer element. This creates our tree. The next thing would be to create the postOrder function.

```
// Constructing the root node - Using Function (Recommended)
struct node *p = createNode(4);
struct node *p1 = createNode(1);
struct node *p2 = createNode(6);
struct node *p3 = createNode(5);
struct node *p4 = createNode(2);
```

```
    // Finally The tree looks like this:
    //       4
    //      / \
    //     1   6
    //    / \
    //   5   2

    // Linking the root node with left and right children
    p->left = p1;
    p->right = p2;
    p1->left = p3;
    p1->right = p4;
```

Code Snippet 1: Creating the Binary tree

Creating the postOrder function:

3. Create a void function postOrder and pass the pointer to the root node of the tree you want to traverse as the only parameter. Inside the function, check if the pointer is not NULL, otherwise we wouldn't do anything. If it is not NULL, we would not directly print the data of the root since this time it's the last one to get visited.
4. So first, you simply call the same function recursively on the left subtree and then the right subtree using the left and the right elements of the root struct. Once called, recursively, the function now considers the left subtree as an individual tree and applies postorder on it, and the same goes for the right subtree.
5. After visiting them both, you just print the data element of the root node marking it visited. And you are done.

```
void postOrder(struct  node* root){
    if(root!=NULL){
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}
```

Code Snippet 2: Creating the postOrder function

Here is the whole source code:

```c
#include<stdio.h>
#include<malloc.h>

struct node{
    int data;
    struct node* left;
    struct node* right;
};

struct node* createNode(int data){
    struct node *n; // creating a node pointer
    n = (struct node *) malloc(sizeof(struct node)); // Allocating m
    n->data = data; // Setting the data
    n->left = NULL; // Setting the left and right children to NULL
    n->right = NULL; // Setting the left and right children to NULL
    return n; // Finally returning the created node
}

void preOrder(struct  node* root){
    if(root!=NULL){
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct  node* root){
    if(root!=NULL){
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
```

```
    }
}

int main(){

    // Constructing the root node - Using Function (Recommended)
    struct node *p = createNode(4);
    struct node *p1 = createNode(1);
    struct node *p2 = createNode(6);
    struct node *p3 = createNode(5);
    struct node *p4 = createNode(2);
    // Finally The tree looks like this:
    //      4
    //     / \
    //    1   6
    //   / \
    //  5   2

    // Linking the root node with left and right children
    p->left = p1;
    p->right = p2;
    p1->left = p3;
    p1->right = p4;

    preOrder(p);
    printf("\n");
    postOrder(p);
    return 0;
}
```

Code Snippet 3: Implementing the postOrder function

Now simply call both preOrder and the postOrder function passing the pointer to the root node as their parameter and see how their results vary.

```
    preOrder(p);
```

```
    printf("\n");
    postOrder(p);
```

Code Snippet 4: Using the preOrder and the postOrder function

And the output we received was:

```
4 1 5 2 6
5 2 1 6 4
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above code

And it worked. You can clearly observe the differences in both the traversals. Things were all consistent with our discussion above. You must check the function yourself by creating your own binary tree and by applying both techniques. And here we finish learning our second method as well. Let's move on to the last one.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll learn to implement our third and the last traversal technique in C, **InOrder traversal**. Till then keep coding.

Previous Next

CodeWithHarry | Copyright © 2022 CodeWithHarry.com