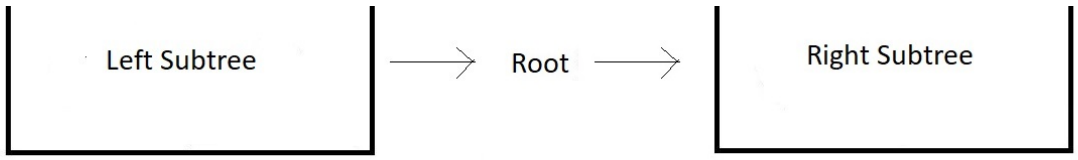
Show Course Contents

Overview Q&A Downloads Announcements

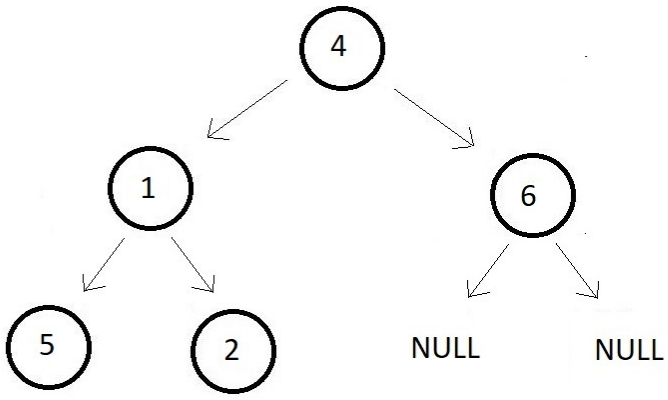
InOrder Traversal in a Binary Tree (With C Code)

In the last lecture, we discussed in detail our second traversal technique, the PostOrder Traversal. We saw its programming implementation in C. If you have not seen the last two lectures already, I suggest checking them out before continuing. Since today we are going to see our last traversal technique, it is a must to see them. Today, we'll see the InOrder Traversal in detail, and also cover its programming part.

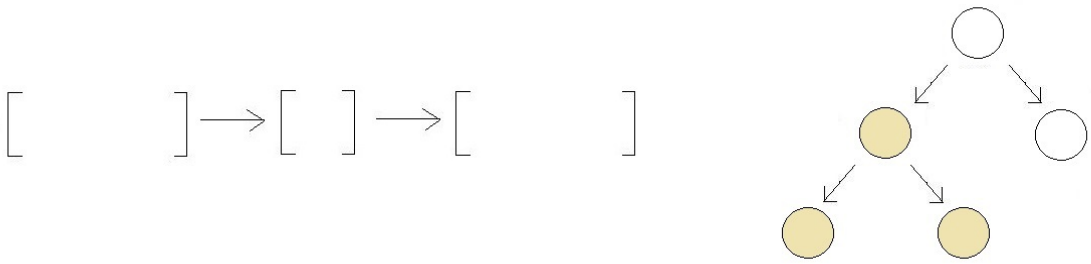
We did study the basic idea of how InOrder traversal works. I had even asked you all previously to explore the flow of this technique yourselves. Today we will see that in detail. So, here you first start with the left subtree and consider that as another tree in itself. You then apply InOrder on this subtree. You then come back to the root node. And then swiftly move to the right subtree and repeat the whole thing again considering this as another individual tree. And this would finish your job of visiting each node. Since the root is in between here, hence the name InOrder.



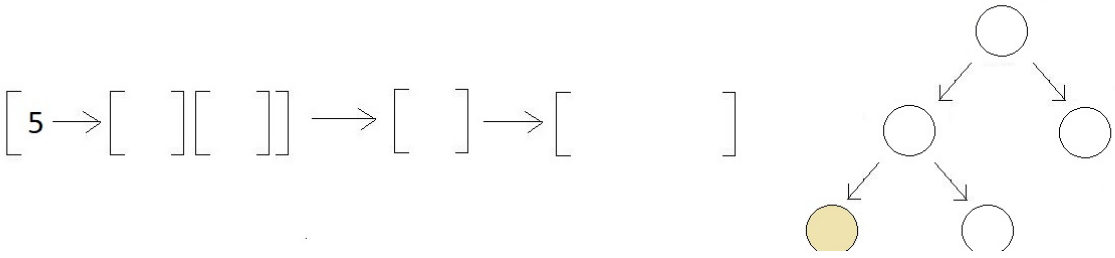
Our flow would remain unchanged, as we follow the same approach as the last two lectures. First, we will take an example binary tree, and apply InOrder Traversal on the same.



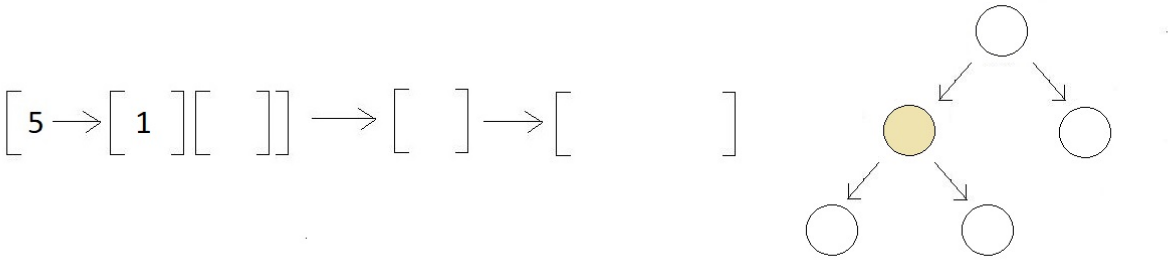
In the first step, you consider the left subtree as a completely different one and apply InOrder separately to it. But before you start with the left subtree, make sure to mark the presence of the root node and the right subtree as following:



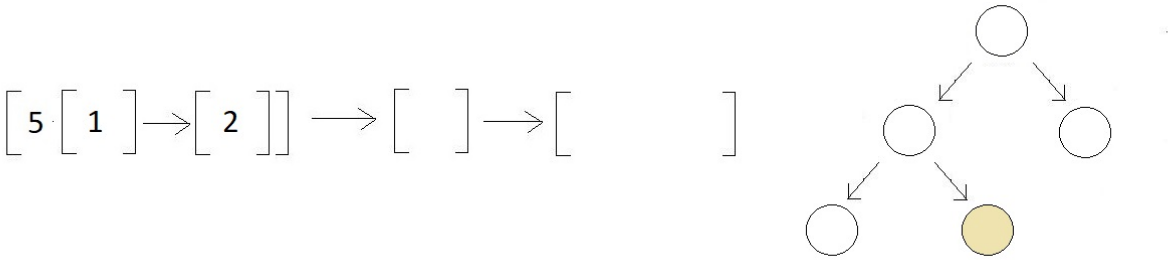
Now, you go through the left subtree, and further visit the left subtree of this new tree first.



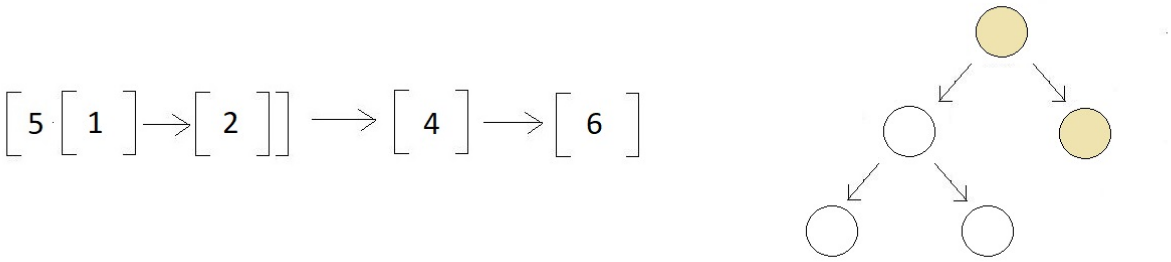
And then you proceed with the root of this new tree we considered.



And after that, you visit the right subtree of this tree which has just a single element.



With that being visited you move back to the original tree. And here, we visit the root node first and then the right subtree, and since it contains no left or right subtree further, we finish visiting our right subtree. There we mark the end of our InOrder traversal.



And our final order of inorder traversal is: 5 → 1 → 2 → 4 → 6.
Having completed the flow of the InOrder Traversal, 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. You know what to do first, right? We avoid doing repetitions, so we wouldn't start from scratch creating the whole struct Node and the createNode thing again rather we would just copy the whole thing we did in the PostOrder Traversal programming part and paste them here. This would get us even the codes for

PreOrder and PostOrder.

2. You should now be able to create Binary Tree yourselves by now. So, just create the same binary tree we observed above using the createNode function. And if you are still not sure about creating a binary tree, follow the previous lectures. Let's now move to create the InOrder function.

Creating the inOrder function:

3. Create a void function inOrder 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, as we are doing every time, since this is the base case for the recursion to stop. If it is NULL, we wouldn't do anything but if it isn't we would call the same function recursively on the left subtree using the left element of the root struct.
4. After we finish visiting the left subtree, we print the data element of the root node indicating it as visited.
5. Having visited both the left subtree and the root, we now move to the right subtree and call it recursively. This completes our flow. And we are done visiting all the nodes.

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

Code Snippet 1: Creating the InOrder function

Here is the whole source code:

```
#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);
    }
}

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

```
    }
}

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);
    // printf("\n");
    inOrder(p);
    return 0;
}
```

Code Snippet 2: Implementing the inOrder function

Now simply call all the traversal functions, preOrder, postOrder, and inOrder

passing the pointer to the root node as their parameter and see how they work.

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

Code Snippet 3: Using the preOrder, the postOrder, and the inOrder functions

And the output we received was:

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

Figure 1: Output of the above code

And it worked accurately to our knowledge. You should feel and observe the differences in all the traversal techniques. Things matched with our manual traversal above. You must verify the functions yourself by creating your own binary tree and by applying all the techniques. And here we finish all our traversal methods.

The binary tree we took as an example could be traversed manually very easily. I wonder, however, whether you can handle a large binary tree, or if you should use the same approach in an assessment with a time constraint. Obviously, no. Three are ways faster than this. And I'll make sure you learn that in the next lecture.

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 a quick method to write the order of traversal regardless of the technique we follow. Till then keep coding.

Previous

Next

CodeWithHarry