Show Course Contents

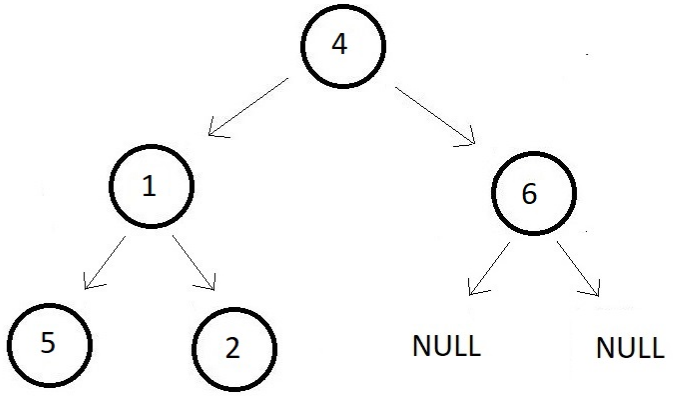
Overview Q&A Downloads Announcements

Preorder Traversal in a Binary Tree (With C Code)

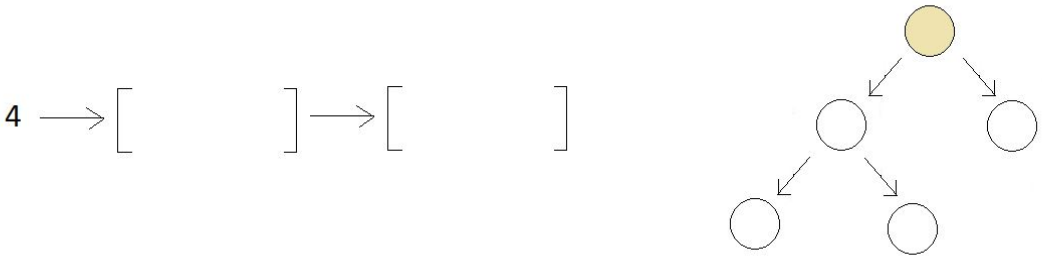
In the last lecture, we saw why we traverse through trees for different purposes. We primarily discussed, in brief, all the three types of traversal techniques. I discussed the flow of the PreOrder Traversal there itself and asked you to cover the last two yourselves. We will surely cover them in detail starting today with the first one, PostOrder Traversal, and see its programming implementation in C.

We basically have a basic idea of how PreOrder traversal works. Today we will see that in great detail. So, here you first start with the root node of the main tree and then get the hold of the left subtree. Now consider this left subtree as a new tree, and apply PreOrder on this. Recursively doing this with all the further subtrees, you will visit each node of this tree. Once you finish with the left subtree, you go to the right subtree and consider this as another tree and repeat the whole thing again.

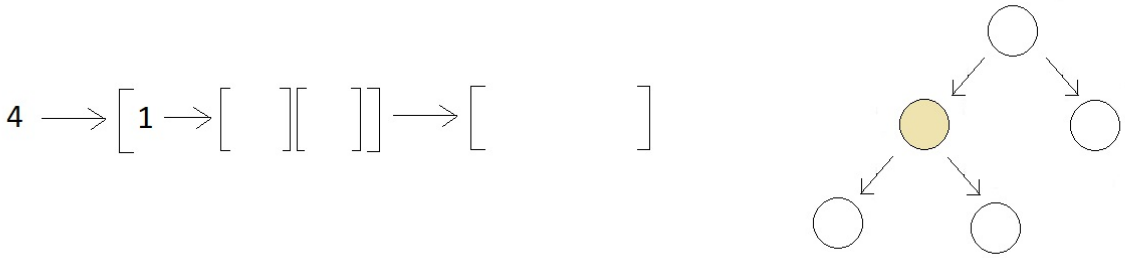
Let's first take an example binary tree, and apply PreOrder Traversal on the same.



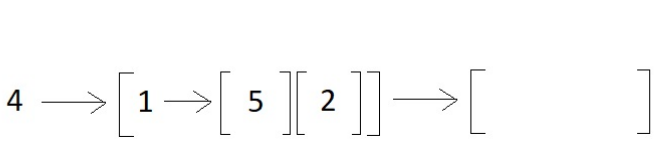
In the first step, you visit the root node and mark the presence of the left and the right subtree as separate individual trees.

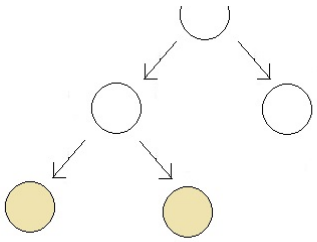


After you visit the root node, you move to the left subtree considering it as a different tree, and start with its root node.

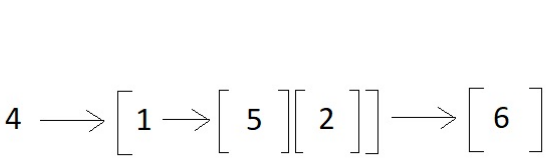


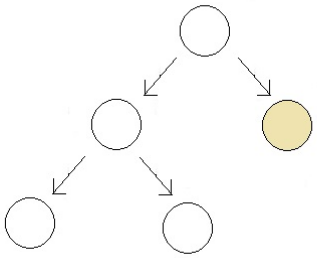
And then you proceed further with the left and right subtrees of this new tree we considered. And since both the left and right subtrees of this tree have just a single element in them, you finish visiting them, and return back to our original tree.





And finally, we visit the right subtree, and since it contains no left or right subtree further, we finish our preorder traversal here itself.





And our final order of preorder traversal is: 4 → 1 → 5 → 2 → 6.
Now we are ready to implement its programming having studied the flow in detail. I have attached the source code below. Follow it as we proceed.

Understanding the code snippet below:

1. First of all, we wouldn't start from scratch creating the struct Node and the createNode function and everything. So just copy the whole thing we did in our previous programming lecture and paste them here. This would save us a lot of time.
2. Create all the five nodes, using the createNode function, and link them using the arrow operator, and altering their left and right pointer elements. This creates our tree. The next thing would be to create the preOrder 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 preOrder function:

3. Create a void function preOrder 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. So, if it is not NULL, print the data element of the root struct node by using the arrow operator.
4. After you finish visiting the root node, simply call the same function recursively on the left and the right subtrees and you're done. Applying recursion does your job in its own subtle ways. It considers the left subtree as an individual tree and applies preorder on it, and the same goes for the right subtree.

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

Code Snippet 2: Creating the preOrder function

Here is the whole source code:

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

```c
#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);
    }
}

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);
    return 0;
}
```

Code Snippet 3: Implementing the preOrder function

Now simply call the preOrder function passing the pointer to the root node as its parameter and see if it actually visits each node.

```
    preOrder(p);
```

Code Snippet 4: Using the preOrder function

And the output we received was:

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

Figure 1: Output of the above code

So, it worked. It did visit each node, and it accurately followed the flow we discussed above. You can check the function yourself by creating your own binary tree. I don't think you need any more assistance there. And here we finish learning our first method. Let's make our way up to the next 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 second traversal technique in C, **PostOrder traversal**. Till then keep coding.

Previous Next

CodeWithHarry

Copyright © 2022 CodeWithHarry.com