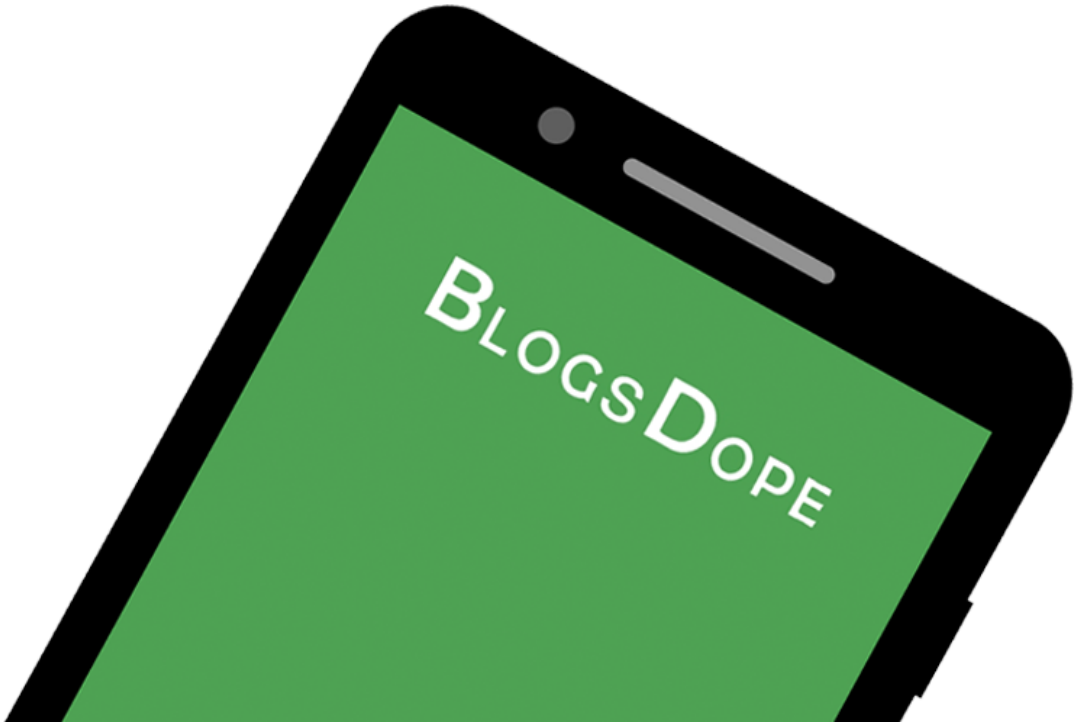
Binary Tree in Java: Traversals, Finding Height of Node

Sept. 14, 2018 LINKED LIST (/blog/tag/linked-list/?tag=linked-list) JAVA (/blog/tag/java/?tag=java) TREE (/blog/tag/tree/?tag=tree) BINARY TREE (/blog/tag/binary-tree/?tag=binary-tree) BINARY SEARCH TREE (/blog/tag/binary-search-tree/?tag=binary-search-tree) DATA STRUCUTRE (/blog/tag/data-strucutre/?tag=data-strucutre) 1390



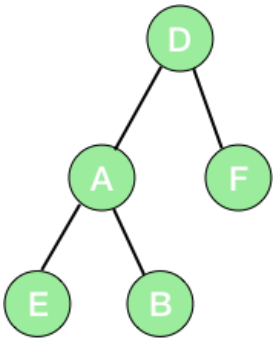
(https://play.google.com/store/apps/details?id=com.blogsdope&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1)

Previous:

1. Trees in Computer Science (https://www.codesdope.com/blog/article/trees-in-computer-science)
2. Binary Trees (https://www.codesdope.com/blog/article/binary-trees)

This post is about implementing a binary tree in Java. You can visit Binary Trees (https://www.codesdope.com/blog/article/binary-trees) for the concepts behind binary trees. We will implement **inorder**, **preorder** and **postorder** traversals and then finish this post by **making a function to calculate the height of the tree**.

The binary tree (https://www.codesdope.com/blog/article/binary-trees) we will be using in this post is:



So, let's make a node Java using class.

```
class Node{
    private String data;
    private Node left;
    private Node right;

    public Node(String element){
        data = element;
        left = null;
        right = null;
    }

    public void setRightChild(Node n)
    {
        right = n;
    }

    public void setLeftChild(Node n){
        left = n;
    }

    public Node getRightChild(){
        return right;
    }

    public Node getLeftChild(){
        return left;
    }

    public String getData(){
        return data;
    }
}
```

`private String data` – The data which we are going to store in this node is of string type.

`private Node right` – Our node also contains two other nodes i.e., its right child and its left child. 'right' is the right child of the current node.

`private Node right` – 'left' is the left child of the current node.

Now, we have a node and we need methods to set and get children and the data and a constructor.

`public Node(String element)` – It is the constructor of the 'Node' class. It is setting the data of the node to the string passed to it and making the left and right children null.

`setRightChild(Node n)` and `Node getRightChild()` – These are the methods to set the right child of a node and to return the right child of the node.

`setLeftChild(Node n)` and `Node getLeftChild()` – Similarly, methods to get and set the left child of a node.

`getData()` – Method to return the data of the node.

Traversals in a Binary Tree

Now, we have made our node. Thus, the next task is to make the tree described in the above picture and implement **inorder**, **postorder** and **preorder** traversals to it. So, let's first make the tree in the main function.

```
class Tree{
    public static void main(String[] args){
        Node root = new Node("D");

        /*
             ____
            |  D |
            |____|

        */

        root.setLeftChild(new Node("A")); //left child of root

        /*
                 ____
                |  D |
               /|____|
         ____ /
        |  A |
        |____|

        */

        root.setRightChild(new Node("F")); //right child of root

        /*
                 ____
                |  D |
               /|____|\
         ____ /       _\___
        |  A |         |  F |
        |____|         |____|

        */

        root.getLeftChild().setLeftChild(new Node("E")); // new node

        /*
                        ____
                       |  D |
                      /|____|\
                ____ /       _\___
               |  A |         |  F |
              /|____|         |____|
           ___/
          |  E |
          |____|

        */

        root.getLeftChild().setRightChild(new Node("B")); // new node

            /*
                        ____
                       |  D |
                      /|____|\
                ____ /       _\___
               |  A |         |  F |
              /|____|\        |____|
           ___/       _\___
          |  E |      |   B |
          |____|      |_____|

        */
```

```
    }
}
```

You can learn the concepts behind the traversals from the post Binary Trees (https://www.codesdope.com/blog/article/binary-trees).

Preorder Traversal

```
// method for preorder
public static void preorder(Node root){
    if(root!=null){ // checking if the root is not null
        System.out.print(" "+root.getData()+" "); // printing data at roo
        preorder(root.getLeftChild()); // visiting left child
        preorder(root.getRightChild()); // visiting right child
    }
}
```

In preorder traversal, we first visit the root and then the left subtree and lastly the right subtree. We are doing the same here.

`System.out.print(" "+root.getData()+" ")` – We are first visiting the root (of the main tree or subtree) or the current node then we will visit its left subtree and then the right subtree.

`preorder(root.getLeftChild())` – Then we are visiting the left subtree.

`preorder(root.getRightChild())` – And lastly the right subtree.

Postorder Traversal

```
public static void postorder(Node root){
    if(root!=null){ // checking if the root is not null
        postorder(root.getLeftChild()); // visiting left child
        postorder(root.getRightChild()); // visiting right child
        System.out.print(" "+root.getData()+" "); // printing data at roo
    }
}
```

In postorder traversal, we first visit the left subtree and then the right and lastly the node.

Inorder Traversal

```
public static void inorder(Node root){
    if(root!=null){ // checking if the root is not null
        inorder(root.getLeftChild()); // visiting left child
        System.out.print(" "+root.getData()+" "); // printing data at roo
        inorder(root.getRightChild()); // visiting right child
    }
}
```

We first visit the left subtree and then root and lastly the right subtree in inorder traversal.

Height of a Node or Binary Tree

Height of a node is 1+ height greater among the heights of the left subtree and the right subtree. Also, the height of a leaf node or a null node is 0. Thus, we will first write a method to identify a leaf node.

Function to Identify Leaves in Binary Tree

```
public static boolean isLeaf(Node a){
    if(a.getRightChild()==null && a.getLeftChild()==null)
        return true;
    return false;
}
```

Checking for a leaf node is simple. If both the children of a node are null then it is a leaf node. We are checking the same with – `if(a.getRightChild()==null && a.getLeftChild()==null)`.

Now, we are ready to write a function to get the height of any node of a tree.

```
// function to return maximum of two numbers
public static int getMax(int a, int b){
    return (a>b) ? a : b;
}

//function to get the height of a tree or node
public static int getHeight(Node a){
    if(a==null || isLeaf(a)) // height will be 0 if the node is leaf or n
        return 0;
    //height of a node will be 1+ greater among height of right subtree a
    return(getMax(getHeight(a.getLeftChild()), getHeight(a.getRightChild()
}
```

'getMax' is a function to determine the greater number of the two numbers passed to it.

'getHeight' is the function to calculate the height of the tree. We are first checking for a null node or leaf node with `if(a==NULL || isLeaf(a))`. In both cases, the height will be 0. Else, the height will be 1+maximum among the heights of left and the right subtrees – `get_max(get_height(a->left_child), get_height(a->right_child)) + 1`.

Let's implement the above concepts and see the result.

```
class Node{
    private String data;
    private Node left;
    private Node right;

    public Node(String element){
        data = element;
        left = null;
        right = null;
    }

    public void setRightChild(Node n)
    {
        right = n;
    }

    public void setLeftChild(Node n){
        left = n;
    }

    public Node getRightChild(){
        return right;
    }

    public Node getLeftChild(){
        return left;
    }

    public String getData(){
        return data;
    }
}

// main class
class Tree{

    // method for preorder
    public static void preorder(Node root){
        if(root!=null){ // checking if the root is not null
            System.out.print(" "+root.getData()+" "); // printing data at
            preorder(root.getLeftChild()); // visiting left child
            preorder(root.getRightChild()); // visiting right child
        }
    }

    //method for postorder
    public static void postorder(Node root){
        if(root!=null){ // checking if the root is not null
            postorder(root.getLeftChild()); // visiting left child
            postorder(root.getRightChild()); // visiting right child
            System.out.print(" "+root.getData()+" "); // printing data at
        }
    }

    //method for inorder
    public static void inorder(Node root){
        if(root!=null){ // checking if the root is not null
            inorder(root.getLeftChild()); // visiting left child
            System.out.print(" "+root.getData()+" "); // printing data at
            inorder(root.getRightChild()); // visiting right child
        }
    }

    // method to check if a node is leaf or not
```

```
public static boolean isLeaf(Node a){
    if(a.getRightChild()==null && a.getLeftChild()==null)
        return true;
    return false;
}

// function to return maximum of two numbers
public static int getMax(int a, int b){
    return (a>b) ? a : b;
}

//function to get the height of a tree or node
public static int getHeight(Node a){
    if(a==null || isLeaf(a)) // height will be 0 if the node is leaf
        return 0;
    //height of a node will be 1+ greater among height of right subtr
    return(getMax(getHeight(a.getLeftChild()), getHeight(a.getRightChi
}

public static void main(String[] args){
    Node root = new Node("D");

    /*
         ____
        |  D |
        |____|

    */

    root.setLeftChild(new Node("A")); //left child of root

    /*
              ____
             |  D |
            /|____|
       ____ /
      |  A |
      |____|

    */

    root.setRightChild(new Node("F")); //right child of root

    /*
              ____
             |  D |
            /|____|\
       ____ /      _\___
      |  A |       |  F |
      |____|       |____|

    */

    root.getLeftChild().setLeftChild(new Node("E")); // new node

    /*
                        ____
                       |  D |
                      /|____|\
                ____ /      _\___
               |  A |       |  F |
              /|____|       |____|
         ____/
        |  E |
        |____|
```

```
        */

        root.getLeftChild().setRightChild(new Node("B")); // new node

            /*
                            ____
                          |  D |
                         /|____|\
                    ____ /       _\___
                  |  A |          |  F |
                 /|____|\         |____|
           _____/       _\___
          |  E |       |   B |
          |____|       |_____|

        */

        preorder(root);
        System.out.println("");
        postorder(root);
        System.out.println("");
        inorder(root);
        System.out.println("");

        System.out.println(getHeight(root));
    }
}
```

Output:

```
 D   A   E   B   F

 E   B   A   F   D

 E   A   B   D   F

2
```

Next:

1. Binary Tree in Java: Traversals, Finding Height of Node (https://www.codesdope.com/blog/article/binary-tree-in-java-traversals-finding-height-of-n)
2. Binary Search Tree (https://www.codesdope.com/blog/article/binary-search-tree)
3. Binary Search Tree in Java (https://www.codesdope.com/blog/article/binary-search-tree-in-java)
