Binary Tree | Set 1 (Introduction)

Trees: Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.

Tree Vocabulary: The topmost node is called root of the tree. The elements that are directly under an element are called its children. The element directly above something is called its parent. For example, 'a' is a child of 'f', and 'f' is the parent of 'a'. Finally, elements with no children are called leaves.

```
      tree
      ----
       j    <-- root
     /   \
    f      k
  /   \      \
 a     h      z    <-- leaves
```

Why Trees?
1. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

```
file system
-----------
     /    <-- root
  /      \
...       home
      /          \
   ugrad        course
    /       /      |     \
  ...      cs101  cs112  cs113
```

2. Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linl List and slower than arrays).
3. Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered

Main applications of trees include:
1. Manipulate hierarchical data.
2. Make information easy to search (see tree traversal).
3. Manipulate sorted lists of data.
4. As a workflow for compositing digital images for visual effects.
5. Router algorithms
6. Form of a multi-stage decision-making (see business chess).

Binary Tree: A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

Binary Tree Representation in C: A tree is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL.
A Tree node contains following parts.
1. Data
2. Pointer to left child
3. Pointer to right child

In C, we can represent a tree node using structures. Below is an example of a tree node with an integer data.

C

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

Python

```
# A Python class that represents an individual node
# in a Binary Tree
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.val = key
```

```
   node and key value*/
class Node
{
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}
```

First Simple Tree in C

Let us create a simple tree with 4 nodes in C. The created tree would be as following.

```
      tree
      ----
       1    <-- root
     /   \
    2     3
   /
  4
```

C

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

/* newNode() allocates a new node with the given data and NULL left and
   right pointers. */
struct node* newNode(int data)
{
  // Allocate memory for new node
  struct node* node = (struct node*)malloc(sizeof(struct node));

  // Assign data to this node
  node->data = data;

  // Initialize left and right children as NULL
  node->left = NULL;
  node->right = NULL;
  return(node);
}
```

```
  struct node *root = newNode(1);
  /* following is the tree after above statement

        1
      /   \
     NULL  NULL
  */


  root->left        = newNode(2);
  root->right       = newNode(3);
  /* 2 and 3 become left and right children of 1
           1
         /   \
        2      3
     /    \    /  \
    NULL NULL NULL NULL
  */


  root->left->left  = newNode(4);
  /* 4 becomes left child of 2
           1
       /       \
      2          3
    /   \       /  \
   4    NULL  NULL  NULL
  /  \
NULL NULL
*/

  getchar();
  return 0;
}
```

Python

```python
# Python program to introduce Binary Tree

# A class that represents an individual node in a
# Binary Tree
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.val = key


# create root
root = Node(1)
''' following is the tree after above statement
        1
      /   \
     None  None'''
```

```
''' 2 and 3 become left and right children of 1
              1
            /   \
          2       3
        /   \    /  \
      None None None None'''


root.left.left  = Node(4);
'''4 becomes left child of 2
              1
          /       \
         2          3
       /   \       /  \
      4    None  None  None
     /  \
   None None'''
```

Java

```
/* Class containing left and right child of current
   node and key value*/
class Node
{
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

// A Java program to introduce Binary Tree
class BinaryTree
{
    // Root of Binary Tree
    Node root;

    // Constructors
    BinaryTree(int key)
    {
        root = new Node(key);
    }

    BinaryTree()
    {
        root = null;
    }

    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        /* following is the tree after above statement

              1
            /   \
          null  null     */

        tree.root.left = new Node(2);
        tree.root.right = new Node(3);

        /* 2 and 3 become left and right children of 1
               1
             /   \
            2      3
          /    \    /  \
        null null null null  */


        tree.root.left.left = new Node(4);
        /* 4 becomes left child of 2
                    1
                /       \
               2          3
             /   \       /  \
            4    null  null  null
           /   \
          null null
         */
    }
}
```

Summary: Tree is a hierarchical data structure. Main uses of trees include maintaining hierarchical data, providing moderate access and insert/delete operations. Binary trees are special cases of tree where every node has at most two children.

Below are set 2 and set 3 of this post.
Properties of Binary Tree
Types of Binary Tree

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Tree Traversals (Inorder, Preorder and Postorder)
Find the node with minimum value in a Binary Search Tree
Write a program to Calculate Size of a tree | Recursion
Write a Program to Find the Maximum Depth or Height of a Tree