Implementing a Binary Search Tree

Emmanuel Ameh

Binary Search Tree Implementation

In the following project, the Java programming language within Visual Studio Code program will be used in implementing a binary search tree, equivalent to a concordance implementation. This tree parses a text file and outputs each word within the file and concurrently exhibits the count for each individual word, alphabetically. Words in the text include “a” “is” “tree” “fruit” “fruitless” “without”. When an exception is thrown “Binary tree implementation is complete” is the output. In this code the exception is overridden.

**Java Code for Binary Search Tree Implementation (Concordance)**

import java.io.\*;

class BinaryTrees {

/\* Class containing left and right child of current node and key value\*/

class Node {

String key;

int count;

Node left, right;

public Node(String item) {

key = item;

count=1;

left = right = null;

}

}

// Root of the Binary Search Tree

Node root;

// Constructor

BinaryTrees() {

root = null;

}

// This method mainly calls insertRec()

void insert(String key) {

root = insertRec(root, key);

}

/\* A recursive function to insert a new key in Binary Search Tree \*/

Node insertRec(Node root, String key) {

/\* If the tree is empty, return a new node \*/

if (root == null) {

root = new Node(key);

return root;

}

/\* Otherwise, recur down the tree \*/

if (key.compareTo(root.key)<0)

root.left = insertRec(root.left, key);

else if (key.compareTo(root.key)>0)

root.right = insertRec(root.right, key);

else

root.count++;

/\* return the (unchanged) node pointer \*/

return root;

}

// This method mainly calls InorderRec()

void rightnodeleft() {

orderRec(root);

}

// A utility function to do order traversal of Binary Search Tree

void orderRec(Node root) {

if (root != null) {

orderRec(root.right);

System.out.println(root.key+"\t\t"+root.count);

orderRec(root.left);

}

}

// Driver Program to test the above functions

public static void main(String[] args) {

String str;

BinaryTrees tree = new BinaryTrees();

System.out.println("Word"+"\t\t"+"Count");

tree.insert("fruit");

tree.insert("a");

tree.insert("a");

tree.insert("without");

tree.insert("fruitless");

tree.insert("tree");

tree.insert("is");

tree.insert("without");

tree.insert("fruit");

tree.insert("fruitless");

tree.insert("fruit");

tree.insert("without");

tree.insert("tree");

tree.insert("fruitless");

tree.insert("tree");

tree.insert("fruit");

tree.insert("tree");

tree.insert("tree");

tree.insert("is");

tree.insert("fruitless");

tree.insert("tree");

tree.insert("fruitless");

tree.insert("without");

tree.insert("fruitless");

tree.insert("fruit");

tree.insert("tree");

tree.insert("a");

// print order traversal of the Binary Search Trees

tree.rightnodeleft();

try {

FileReader fr=new FileReader("example.txt");

BufferedReader br= new BufferedReader(fr);

while((str=br.readLine())!=null)

tree.insert(str);

fr.close();

tree.rightnodeleft();

}

catch(Exception e) {

System.out.println("Binary tree application complete");

return;

}

}

}



