Vaishak Menon

CS303 Lab 8 – Binary Search Tree

Problem Specification:

The problem given to us is to create the code for a binary search tree and run it against the given input and data files. We had to specifically test out two different functions. The first was the inorder traversal and the second being the searching function.

Program Design:

I have three classes: Node.java, BST.java, and Driver.java. The driver class just reads in all the files and creates nodes for the BST file to use. The BST file has three functions: TreeInsert(), inOrderTreeWalk(), and search(). The TreeInsert() function essentially creates the tree by appending a node to the previous node starting from the root each time. The first node to be added to the tree instantly becomes the root. The inOrderTreeWalk function starts at the root of the function and goes down the left and right side of each node and prints out the key. Finally, the search function goes through the entire tree and finds every element that is listed within the input.dat or KEYs.dat file. After finding the element, it prints out the time it took to find it and the key and description of the node. The final class I have is the Node class, which is essentially a class that holds the data for anything that is deemed to be a Node object. It holds the key and description of each node as well as its left and right node information.

Testing Plan:

In order to implement this, I created a system that allows the user to choose which file they would like to create a binary search tree out of. Then I run the inOrderTreeWalk and print out all the keys of the nodes within the tree. I have set inOrderTreeWalk to only run if you are using Data2.csv or Data3.csv due to Data1.csv and UPC.csv causing a stack overflow due to how many things it is printing. After it is done printing out all the keys, I ask the user if which input file they would like to use. The choices are the input.dat or the KEYs.csv. After calling the search on each of the keys within either file the program prints out how long it took to search for that key and prints out the key and its description.

Test Cases:

```
Choose a new file or hit 5 to exit
1: Data1.csv
2: Data2.csv
3: Data3.csv
4: UPC.csv
1
WILL NOT RUN INORDER TRAVERSAL DUE TO STACK OVERFLOW ERROR
Choose key file
1: input.dat
2: KEYs.csv
2
Time take to search for the key 79.0: 80100
Key: 79.0 Description: INDIANA
Time take to search for the key 93.0: 5000
Key: 93.0 Description: treo
Time take to search for the key 123.0: 7600
Key: 123.0 Description: Wrsi
Time take to search for the key 161.0: 7700
Key: 161.0 Description: Dillons/Kroger
Time take to search for the key 2.14000007E9: 771600
Key: 2.14000007E9 Description: Rhinestone
Time take to search for the key 2.140118461E9: 491500
Key: 2.140118461E9 Description: """V"": Breakout/The Deception  VHS Tape"
Time take to search for the key 2.144209103E9: 667000
Key: 2.144209103E9 Description: Tintorera
Time take to search for the key 2.144622711E9: 646200
Key: 2.144622711E9 Description: Taxi
Time take to search for the key 2.147483647E9: 219200
Key: 2.147483647E9 Description: Toshiba
Time take to search for the key 2.158242769E9: 223200
Key: 2.158242769E9 Description: GREEN
Time take to search for the key 2.158561631E9: 280900
Key: 2.158561631E9 Description: HOT
Time take to search for the key 2.158769549E9: 241300
Key: 2.158769549E9 Description: gjfhjbgkj
Time take to search for the key 2.160500567E9: 275000
Key: 2.160500567E9 Description: oz
Time take to search for the key 2.172307284E9: 261100
Key: 2.172307284E9 Description: Mixed
Time take to search for the key 2.177000074E9: 269500
Key: 2.177000074E9 Description: 4
Time take to search for the key 2.184000098E9: 177300
Key: 2.184000098E9 Description: oz
Time take to search for the key 2.187682888E9: 209300
Key: 2.187682888E9 Description: fairway
Key 2.8776900003E10 has no description in the file you searched for!
Key 2.8776901406E10 has no description in the file you searched for!
Key 2.8784014907E10 has no description in the file you searched for!
Key 2.8785102009E10 has no description in the file you searched for!
Key 2.8785103105E10 has no description in the file you searched for!
Key 2.8785201054E10 has no description in the file you searched for!
Key 2.8785203508E10 has no description in the file you searched for!
Key 2.8785207452E10 has no description in the file you searched for!
Key 2.87852552E10 has no description in the file you searched for!
Key 2.8785300405E10 has no description in the file you searched for!
Key 2.8785300603E10 has no description in the file you searched for!
Key 2.8785313146E10 has no description in the file you searched for!
Key 2.8785343457E10 has no description in the file you searched for!
Key 2.878540002E10 has no description in the file you searched for!
Key 2.8785509112E10 has no description in the file you searched for!
Key 2.8785509327E10 has no description in the file you searched for!
Key 2.8785509778E10 has no description in the file you searched for!
Key 2.8785601106E10 has no description in the file you searched for!
```

```
Choose which file you would like to use:
1: Data1.csv
2: Data2.csv
3: Data3.csv
4: UPC.csv
1
WILL NOT RUN INORDER TRAVERSAL DUE TO STACK OVERFLOW ERROR
Choose key file
1: input.dat
2: KEYs.csv
1
Time take to search for the key 79.0: 35000
Key: 79.0 Description: INDIANA
Time take to search for the key 93.0: 16100
Key: 93.0 Description: treo
Time take to search for the key 123.0: 8100
Key: 123.0 Description: Wrsi
Time take to search for the key 161.0: 4800
Key: 161.0 Description: Dillons/Kroger
Time take to search for the key 2.14000007E9: 2646500
Key: 2.14000007E9 Description: Rhinestone
Time take to search for the key 2.140118461E9: 2582200
Key: 2.140118461E9 Description: """V"": Breakout/The Deception  VHS Tape"
Time take to search for the key 2.144209103E9: 2545700
Key: 2.144209103E9 Description: Tintorera
Time take to search for the key 2.144622711E9: 2390800
Key: 2.144622711E9 Description: Taxi
Time take to search for the key 2.147483647E9: 2378500
Key: 2.147483647E9 Description: Toshiba
Time take to search for the key 2.158242769E9: 2668100
Key: 2.158242769E9 Description: GREEN
Time take to search for the key 2.158561631E9: 3147800
Key: 2.158561631E9 Description: HOT
Time take to search for the key 2.158769549E9: 2886600
Key: 2.158769549E9 Description: gjfhjbgkj
Time take to search for the key 2.160500567E9: 2434500
Key: 2.160500567E9 Description: oz
Time take to search for the key 2.172307284E9: 1999200
Key: 2.172307284E9 Description: Mixed
Time take to search for the key 2.177000074E9: 2247100
Key: 2.177000074E9 Description: 4
Time take to search for the key 2.184000098E9: 3390900
Key: 2.184000098E9 Description: oz
Time take to search for the key 2.187682888E9: 2746300
Key: 2.187682888E9 Description: fairway
```

Both test cases show data set 1 being inputted into the tree and then both input file and key file used as search keys.

Analysis/Conclusion:

Based on the results I got for the first data file and the search keys, it seems logical for how long it took to get the results. The closer to the root, the smaller time it took to find and print out the key and its description. The higher the number, the longer time it took to find that key, due to the distance and number of checks that had to be completed to get to that node.

The time complexity of TreeInsert is only O(n) since there is only one loop. The time complexity is O(h), h being the height of the tree. The time complexity of search is O(n). The overall time complexity should be O(n) because it cannot take longer than all of the elements in the data files.

References:

The only references I must make are to the pseudocode and my previous lab in which I used a similar format for writing this report. Below I have added images of all my code for this lab.

```java
public class BST extends Node
{

    BST(Double key, String s)
    {
        super(key, s);
    }

    public static Node TreeInsert(Node T, Node z)
    {
        Node y = null;
        Node x = T;

        while(x != null)
        {
            y = x;
            if(z.key < x.key)
            {
                x = x.left;
            }
            else
            {
                x = x.right;
            }
        }

        if(y == null)
        {
            T = z;
        }
        else if(z.key < y.key)
        {
            y.left = z;
        }
        else
        {
            y.right = z;
        }

        return T;
    }

    public static void inOrderTreeWalk(Node x)
    {
        if(x != null)
        {
            inOrderTreeWalk(x.left);
            System.out.println(x.key);
            inOrderTreeWalk(x.right);
        }
    }
```

```java
    public static Node search(Node x, double k)
    {
        while((x != null) && (k != x.key))
        {
            if(k < x.key)
            {
                x = x.left;
            }
            else
            {
                x = x.right;
            }
        }
        return x;
    }
}
```

```java
public static void main(String[] args)
{
    ArrayList<Double> A = new ArrayList<Double>();
    ArrayList<String> S = new ArrayList<String>();


    File f = new File("");
    Scanner userInput = new Scanner(System.in);


    System.out.println("Choose which file you would like to use: ");
    String[] choices = new String[]{"Data1.csv", "Data2.csv", "Data3.csv", "UPC.csv"};
    for(int t = 0; t < choices.length; t++)
    {
        System.out.println((t + 1) + ": " + choices[t]);
    }

    int choice = userInput.nextInt();
    do
    {
        switch (choice)
        {
        case 1:
            f = new File("Data1.csv");
            break;

        case 2:
            f = new File("Data2.csv");
            break;

        case 3:
            f = new File("Data3.csv");
            break;

        case 4:
            f = new File("UPC.csv");
            break;

        default:
            f = new File("Data1.csv");
            break;
        }
```

```
try
{
    Scanner sc = new Scanner(f);
    String[] s = new String[2];
    int count = 0;
    while (sc.hasNextLine()) {
        s = sc.nextLine().trim().split(",");
        A.add(Double.parseDouble(s[0]));
        if (s.length != 1) {
            S.add(s[1]);
        } else {
            s = new String[2];
            s[0] = String.valueOf(A.get(count));
            s[1] = "";
            S.add(s[1]);
        }
        count++;
    }

    sc.close();
}
catch (FileNotFoundException e)
{
    e.printStackTrace();
}

Node[] keys = new Node[A.size()];

for (int i = 0; i < keys.length; i++)
{
    keys[i] = new Node(A.get(i), S.get(i));
}

Node root = null;
for (Node key : keys)
{
    if (key != null)
    {
        root = BST.TreeInsert(root, key);
    }
}
```

```
switch(choice2)
{
    case 1:
        f2 = new File("input.dat");
        try
        {
            String[] s = new String[3];
            Scanner sc2 = new Scanner(f2);
            while (sc2.hasNextLine())
            {
                s = sc2.nextLine().trim().split(",");
                testInputNum.add(Double.parseDouble(s[0]));
                testInputS.add(s[2]);
            }

            sc2.close();
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        break;

    case 2:
        f2 = new File("KEYs.csv");
        try
        {
            String[] s = new String[3];
            Scanner sc2 = new Scanner(f2);
            while (sc2.hasNextLine())
            {
                s = sc2.nextLine().trim().split(",");
                testInputNum.add(Double.parseDouble(s[0]));
            }

            sc2.close();
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        break;

    default:
        f2 = new File("input.dat");
        break;
}
```

```
if (choice == 2 || choice == 3)
{
    BST.inOrderTreeWalk(root);
}
else if(choice == 1 || choice == 4)
{
    System.out.println("WILL NOT RUN INORDER TRAVERSAL DUE TO STACK OVERFLOW ERROR");
}
else
{
    System.out.println("Default file chosen, Data1.csv is being used");
}

ArrayList<Double> testInputNum = new ArrayList<Double>();
ArrayList<String> testInputS = new ArrayList<String>();
File f2 = new File("");

System.out.println("Choose key file");
System.out.println("1: input.dat\n2: KEYs.csv");
int choice2 = userInput.nextInt();
```

```
    for (int i = 0; i < testInputNum.size(); i++)
    {
        long hTime = System.nanoTime();
        Node n = BST.search(root, testInputNum.get(i));
        long ehTime = System.nanoTime() - hTime;
        if(n != null)
        {
            System.out.println("Time take to search for the key " + testInputNum.get(i) + ": " + ehTime);
            System.out.println("Key: " + n.key + " Description: " + n.description);
        }
        if(n == null)
        {
            System.out.println("Key " + testInputNum.get(i) + " has no description in the file you searched for!");
        }
    }

    System.out.println("Choose next action");
    System.out.println("Choose a new file or hit 5 to exit");
    for(int t = 0; t < choices.length; t++)
    {
        System.out.println((t + 1) + ": " + choices[t]);
    }

    choice = userInput.nextInt();
}
while(choice != 5);
userInput.close();
```

```
public class Node
{
    Double key;
    String description;
    Node left, right;

    Node(Double inputKey, String s)
    {
        key = inputKey;
        description = s;
        left = null;
        right = null;
    }
}
```