Vaishak Menon

CS303 Lab 10 – Red-Black Tree

Problem Specification:

The problem given to us is to create the code for a red-black 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.

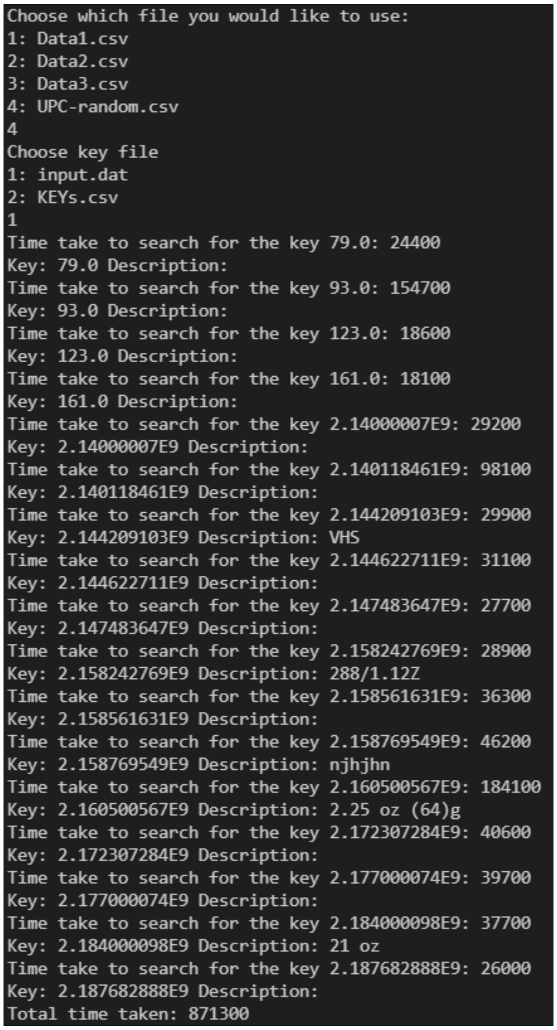
Program Design:

I have five classes: BSTNode.java, BST.java, RBNode.java, RBTree.java, and Driver.java The driver class just reads in all the files and creates nodes for the RBTree file to use. The RBTree file has five functions: RBInsert(), RBInsertFix(), LeftRotate(), RightRotate(), inOrderTreeWalk(), and search(). The RBTree() 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 RBInsertFix function fixes any issues that come from insertion by rotating left and right. It uses conditions to check to see if a left or right rotate is necessary and then calls either LeftRotate() or RightRotate(). 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 created for this lab is the RBNode class, which is essentially a class that holds the data for anything that is deemed to be a RBNode object. It holds the key and description of each node as well as its left, right, and parent node information. It also holds a Boolean variable called color that is false when red and true when black.

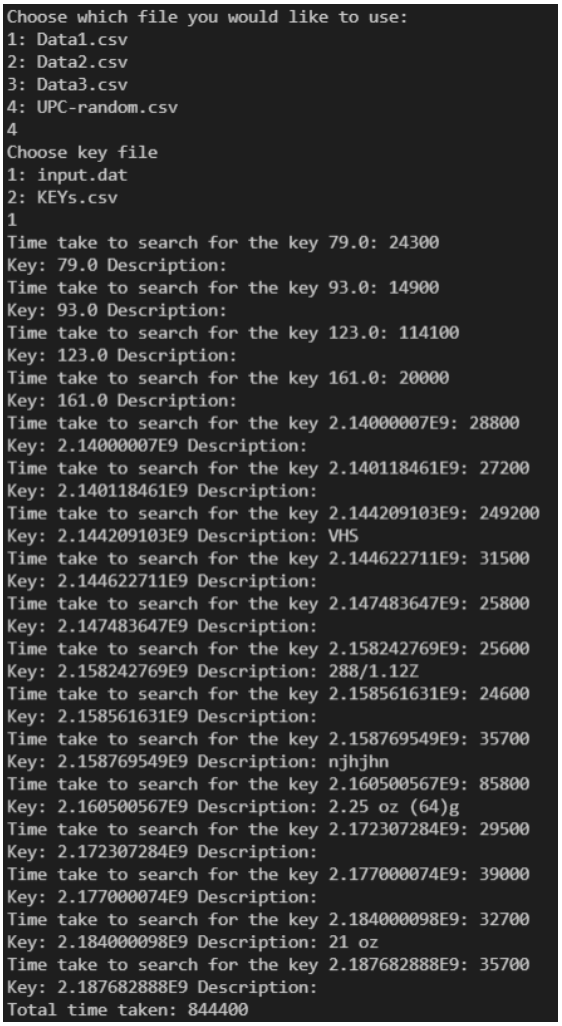
Testing Plan:

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. For this lab, I commented out the line that prints out the key because the UPC-random.txt file has so many keys that it takes a while to print them all out. Afterwards, 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.

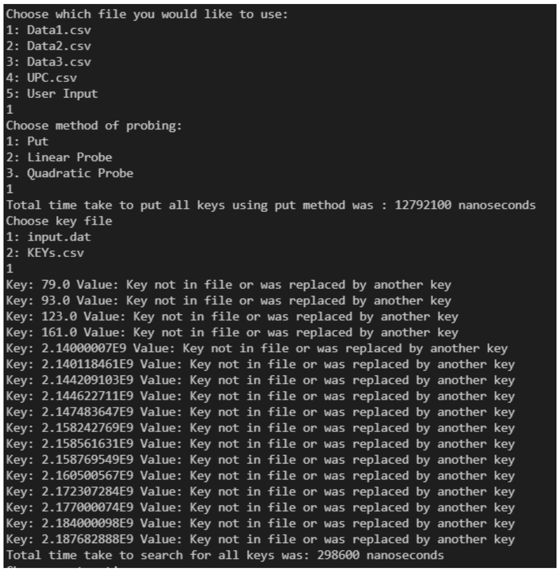
Test Cases:



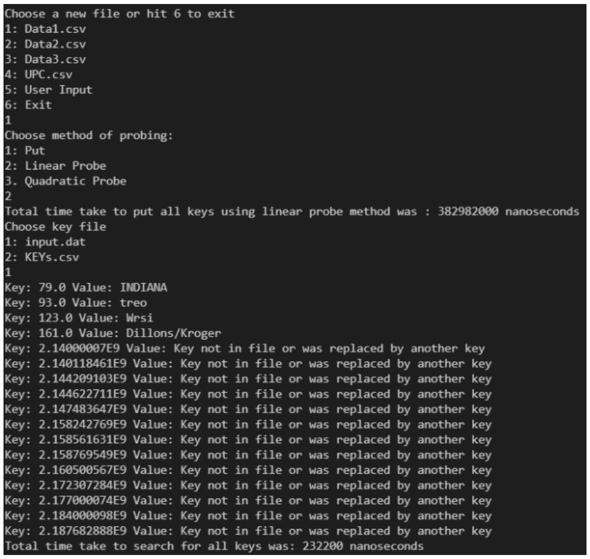
```
Choose which file you would like to use:
1: Data1.csv
2: Data2.csv
3: Data3.csv
4: UPC-random.csv
4
Choose key file
1: input.dat
2: KEYs.csv
1
Time take to search for the key 79.0: 24400
Key: 79.0 Description:
Time take to search for the key 93.0: 154700
Key: 93.0 Description:
Time take to search for the key 123.0: 18600
Key: 123.0 Description:
Time take to search for the key 161.0: 18100
Key: 161.0 Description:
Time take to search for the key 2.14000007E9: 29200
Key: 2.14000007E9 Description:
Time take to search for the key 2.140118461E9: 98100
Key: 2.140118461E9 Description:
Time take to search for the key 2.144209103E9: 29900
Key: 2.144209103E9 Description: VHS
Time take to search for the key 2.144622711E9: 31100
Key: 2.144622711E9 Description:
Time take to search for the key 2.147483647E9: 27700
Key: 2.147483647E9 Description:
Time take to search for the key 2.158242769E9: 28900
Key: 2.158242769E9 Description: 288/1.12Z
Time take to search for the key 2.158561631E9: 36300
Key: 2.158561631E9 Description:
Time take to search for the key 2.158769549E9: 46200
Key: 2.158769549E9 Description: njhjhn
Time take to search for the key 2.160500567E9: 184100
Key: 2.160500567E9 Description: 2.25 oz (64)g
Time take to search for the key 2.172307284E9: 40600
Key: 2.172307284E9 Description:
Time take to search for the key 2.177000074E9: 39700
Key: 2.177000074E9 Description:
Time take to search for the key 2.184000098E9: 37700
Key: 2.184000098E9 Description: 21 oz
Time take to search for the key 2.187682888E9: 26000
Key: 2.187682888E9 Description:
Total time taken: 871300
```



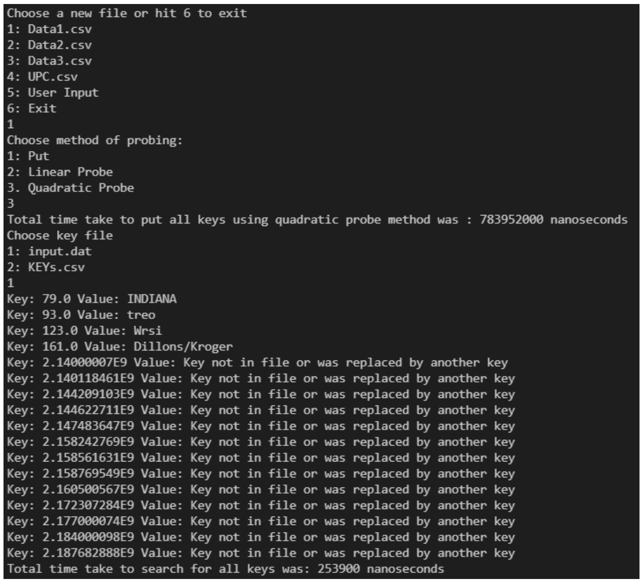
```
Choose which file you would like to use:
1: Data1.csv
2: Data2.csv
3: Data3.csv
4: UPC-random.csv
4
Choose key file
1: input.dat
2: KEYs.csv
1
Time take to search for the key 79.0: 24300
Key: 79.0 Description:
Time take to search for the key 93.0: 14900
Key: 93.0 Description:
Time take to search for the key 123.0: 114100
Key: 123.0 Description:
Time take to search for the key 161.0: 20000
Key: 161.0 Description:
Time take to search for the key 2.14000007E9: 28800
Key: 2.14000007E9 Description:
Time take to search for the key 2.140118461E9: 27200
Key: 2.140118461E9 Description:
Time take to search for the key 2.144209103E9: 249200
Key: 2.144209103E9 Description: VHS
Time take to search for the key 2.144622711E9: 31500
Key: 2.144622711E9 Description:
Time take to search for the key 2.147483647E9: 25800
Key: 2.147483647E9 Description:
Time take to search for the key 2.158242769E9: 25600
Key: 2.158242769E9 Description: 288/1.12Z
Time take to search for the key 2.158561631E9: 24600
Key: 2.158561631E9 Description:
Time take to search for the key 2.158769549E9: 35700
Key: 2.158769549E9 Description: njhjhn
Time take to search for the key 2.160500567E9: 85800
Key: 2.160500567E9 Description: 2.25 oz (64)g
Time take to search for the key 2.172307284E9: 29500
Key: 2.172307284E9 Description:
Time take to search for the key 2.177000074E9: 39000
Key: 2.177000074E9 Description:
Time take to search for the key 2.184000098E9: 32700
Key: 2.184000098E9 Description: 21 oz
Time take to search for the key 2.187682888E9: 35700
Key: 2.187682888E9 Description:
Total time taken: 844400
```



```
Choose which file you would like to use:
1: Data1.csv
2: Data2.csv
3: Data3.csv
4: UPC.csv
5: User Input
1
Choose method of probing:
1: Put
2: Linear Probe
3. Quadratic Probe
1
Total time take to put all keys using put method was : 12792100 nanoseconds
Choose key file
1: input.dat
2: KEYs.csv
1
Key: 79.0 Value: Key not in file or was replaced by another key
Key: 93.0 Value: Key not in file or was replaced by another key
Key: 123.0 Value: Key not in file or was replaced by another key
Key: 161.0 Value: Key not in file or was replaced by another key
Key: 2.14000007E9 Value: Key not in file or was replaced by another key
Key: 2.140118461E9 Value: Key not in file or was replaced by another key
Key: 2.144209103E9 Value: Key not in file or was replaced by another key
Key: 2.144622711E9 Value: Key not in file or was replaced by another key
Key: 2.147483647E9 Value: Key not in file or was replaced by another key
Key: 2.158242769E9 Value: Key not in file or was replaced by another key
Key: 2.158561631E9 Value: Key not in file or was replaced by another key
Key: 2.158769549E9 Value: Key not in file or was replaced by another key
Key: 2.160500567E9 Value: Key not in file or was replaced by another key
Key: 2.172307284E9 Value: Key not in file or was replaced by another key
Key: 2.177000074E9 Value: Key not in file or was replaced by another key
Key: 2.184000098E9 Value: Key not in file or was replaced by another key
Key: 2.187682888E9 Value: Key not in file or was replaced by another key
Total time take to search for all keys was: 298600 nanoseconds
```



```
Choose a new file or hit 6 to exit
1: Data1.csv
2: Data2.csv
3: Data3.csv
4: UPC.csv
5: User Input
6: Exit
1
Choose method of probing:
1: Put
2: Linear Probe
3. Quadratic Probe
2
Total time take to put all keys using linear probe method was : 382982000 nanoseconds
Choose key file
1: input.dat
2: KEYs.csv
1
Key: 79.0 Value: INDIANA
Key: 93.0 Value: treo
Key: 123.0 Value: Wrsi
Key: 161.0 Value: Dillons/Kroger
Key: 2.14000007E9 Value: Key not in file or was replaced by another key
Key: 2.140118461E9 Value: Key not in file or was replaced by another key
Key: 2.144209103E9 Value: Key not in file or was replaced by another key
Key: 2.144622711E9 Value: Key not in file or was replaced by another key
Key: 2.147483647E9 Value: Key not in file or was replaced by another key
Key: 2.158242769E9 Value: Key not in file or was replaced by another key
Key: 2.158561631E9 Value: Key not in file or was replaced by another key
Key: 2.158769549E9 Value: Key not in file or was replaced by another key
Key: 2.160500567E9 Value: Key not in file or was replaced by another key
Key: 2.172307284E9 Value: Key not in file or was replaced by another key
Key: 2.177000074E9 Value: Key not in file or was replaced by another key
Key: 2.184000098E9 Value: Key not in file or was replaced by another key
Key: 2.187682888E9 Value: Key not in file or was replaced by another key
Total time take to search for all keys was: 232200 nanoseconds
```



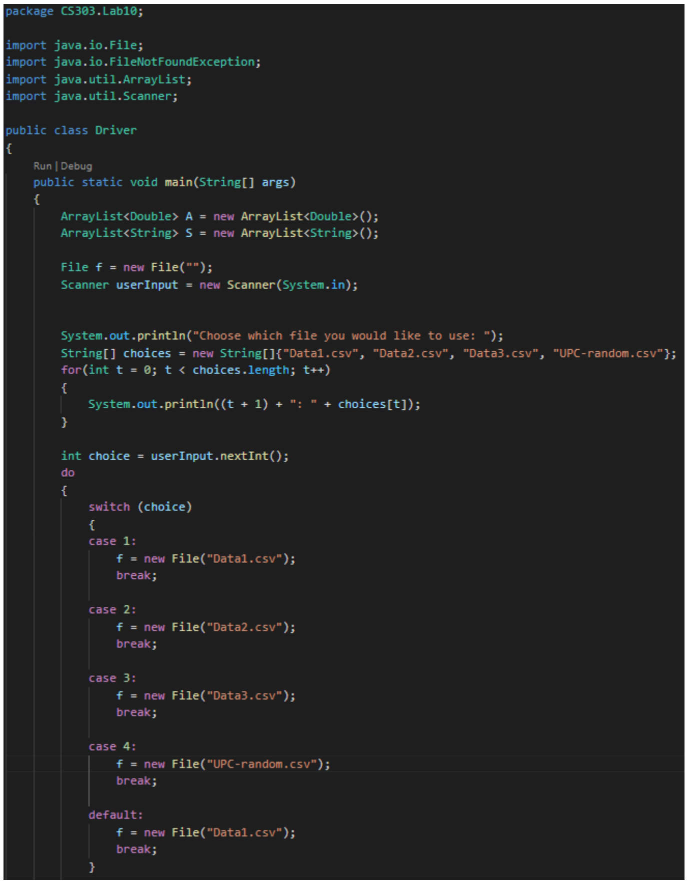
```
Choose a new file or hit 6 to exit
1: Data1.csv
2: Data2.csv
3: Data3.csv
4: UPC.csv
5: User Input
6: Exit
1
Choose method of probing:
1: Put
2: Linear Probe
3. Quadratic Probe
3
Total time take to put all keys using quadratic probe method was : 783952000 nanoseconds
Choose key file
1: input.dat
2: KEYs.csv
1
Key: 79.0 Value: INDIANA
Key: 93.0 Value: treo
Key: 123.0 Value: Wrsi
Key: 161.0 Value: Dillons/Kroger
Key: 2.14000007E9 Value: Key not in file or was replaced by another key
Key: 2.140118461E9 Value: Key not in file or was replaced by another key
Key: 2.144209103E9 Value: Key not in file or was replaced by another key
Key: 2.144622711E9 Value: Key not in file or was replaced by another key
Key: 2.147483647E9 Value: Key not in file or was replaced by another key
Key: 2.158242769E9 Value: Key not in file or was replaced by another key
Key: 2.158561631E9 Value: Key not in file or was replaced by another key
Key: 2.158769549E9 Value: Key not in file or was replaced by another key
Key: 2.160500567E9 Value: Key not in file or was replaced by another key
Key: 2.172307284E9 Value: Key not in file or was replaced by another key
Key: 2.177000074E9 Value: Key not in file or was replaced by another key
Key: 2.184000098E9 Value: Key not in file or was replaced by another key
Key: 2.187682888E9 Value: Key not in file or was replaced by another key
Total time take to search for all keys was: 253900 nanoseconds
```

Analysis/Conclusion:

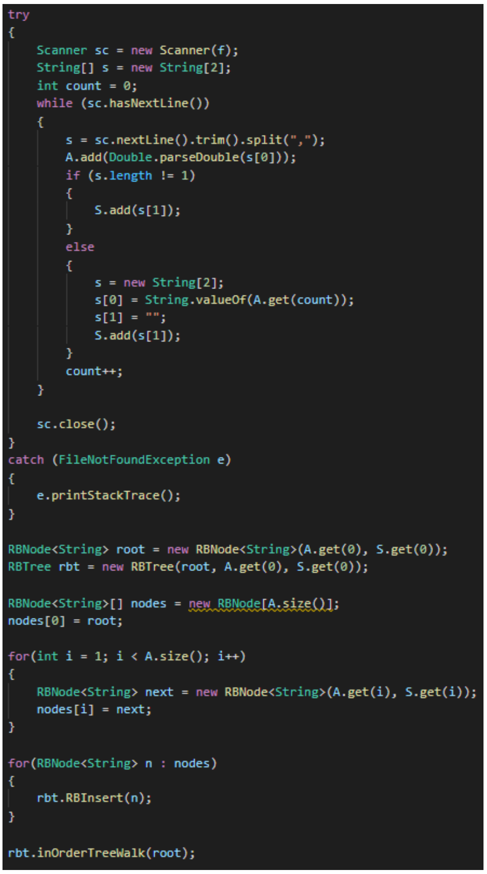
We have two different results to look at, the first being the red-black tree search total time and the HashMap put, linear put, and quadratic put searching times. At first glance, the red-black tree takes a lot more time to search, almost 4 times as long. This is excluding whether the key itself was found and if the description was printed. When comparing the actual results of the search, the red-black tree is able to find the key and description for every key within the input.dat file, while the HashMap search finds very few to none of the keys that input.dat holds. So technically, the red-black tree is better because we are able to find all of the keys from the input.dat file even if the time taken is longer we get better searching results.

References:

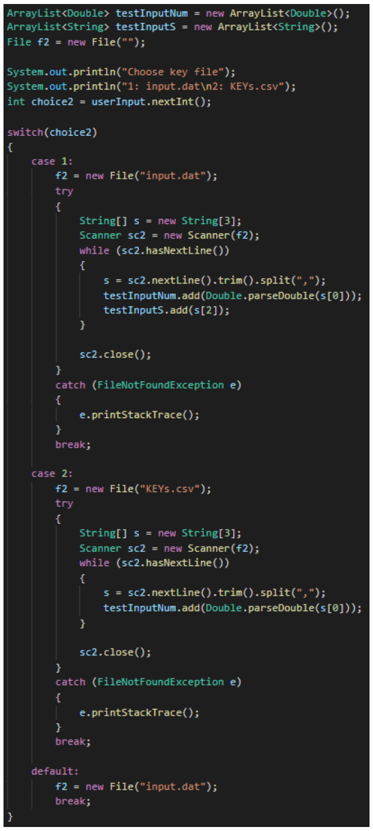
The only references I must make are to the pseudocode and my previous lab in which I used a similar format for writing this report. Below I have added images of all my code for this lab. I am only adding the RBTree, RBNode, and Driver code because the rest is almost identical to the BST code from lab 8.



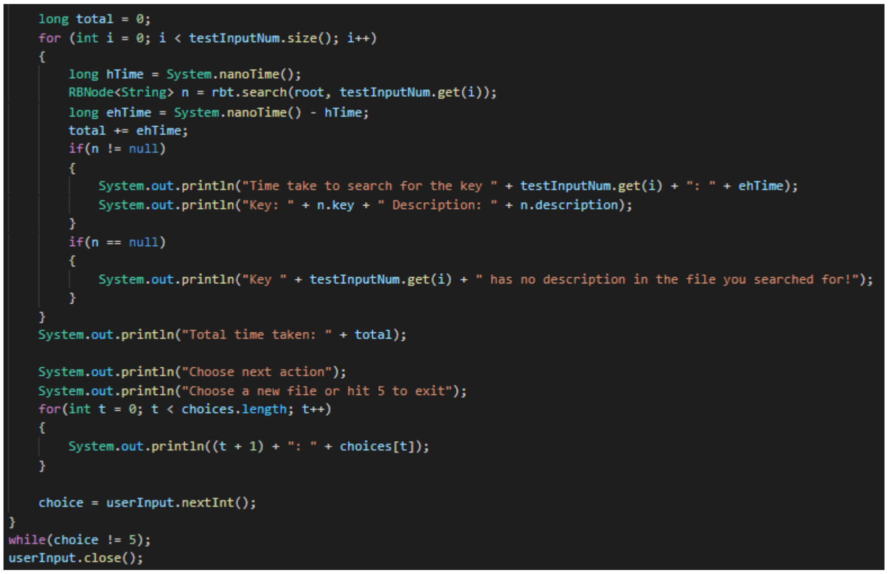
```java
package CS303.Lab10;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;

public class Driver
{
    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-random.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-random.csv");
                break;

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



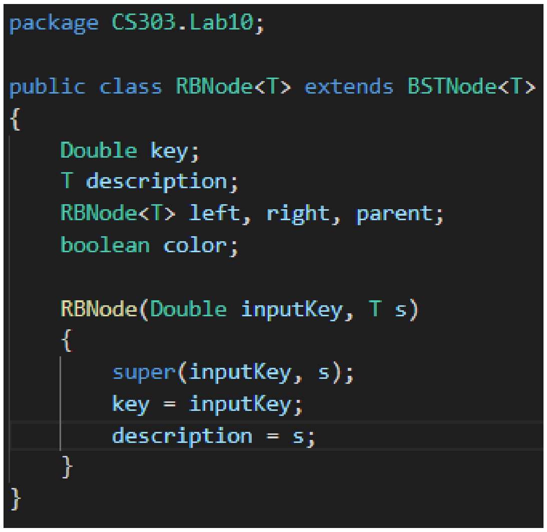
```java
try
{
    Scanner sc = new Scanner(f);
    String[] s = new String[2];
    int count = 0;
    while (sc.hasNextLine())
    {
        s = sc.nextLine().trim().split(",");
        A.add(Double.parseDouble(s[0]));
        if (s.length != 1)
        {
            S.add(s[1]);
        }
        else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

RBNode<String> root = new RBNode<String>(A.get(0), S.get(0));
RBTree rbt = new RBTree(root, A.get(0), S.get(0));

RBNode<String>[] nodes = new RBNode[A.size()];
nodes[0] = root;

for(int i = 1; i < A.size(); i++)
{
    RBNode<String> next = new RBNode<String>(A.get(i), S.get(i));
    nodes[i] = next;
}

for(RBNode<String> n : nodes)
{
    rbt.RBInsert(n);
}

rbt.inOrderTreeWalk(root);
```



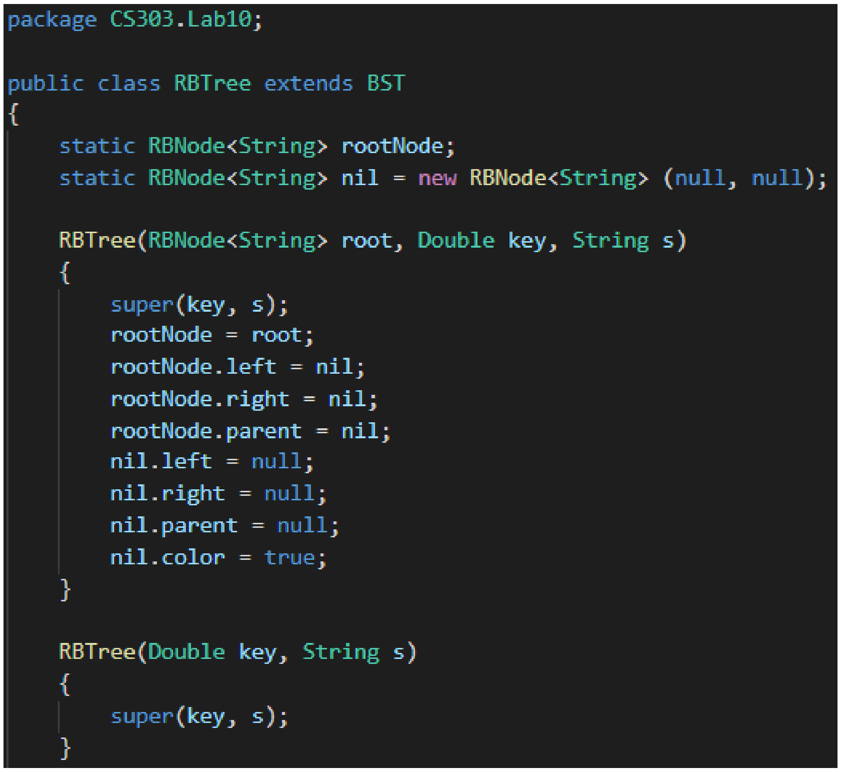
```java
ArrayList<Double> testInputNum = new ArrayList<Double>();
ArrayList<String> testInputS = new ArrayList<String>();
File f2 = new File("");

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

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;
}
```



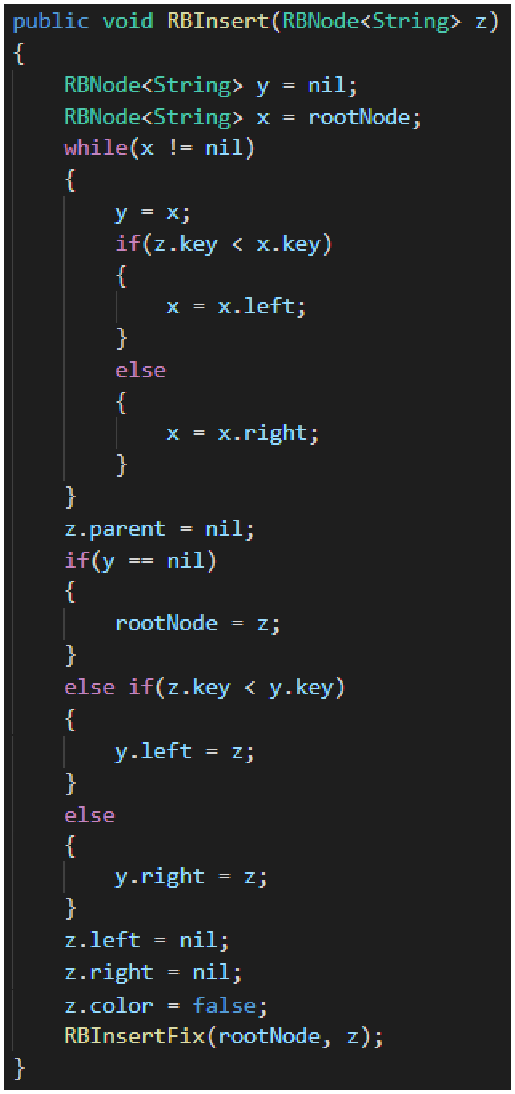
```java
    long total = 0;
    for (int i = 0; i < testInputNum.size(); i++)
    {
        long hTime = System.nanoTime();
        RBNode<String> n = rbt.search(root, testInputNum.get(i));
        long ehTime = System.nanoTime() - hTime;
        total += ehTime;
        if(n != null)
        {
            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("Total time taken: " + total);

    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();
```



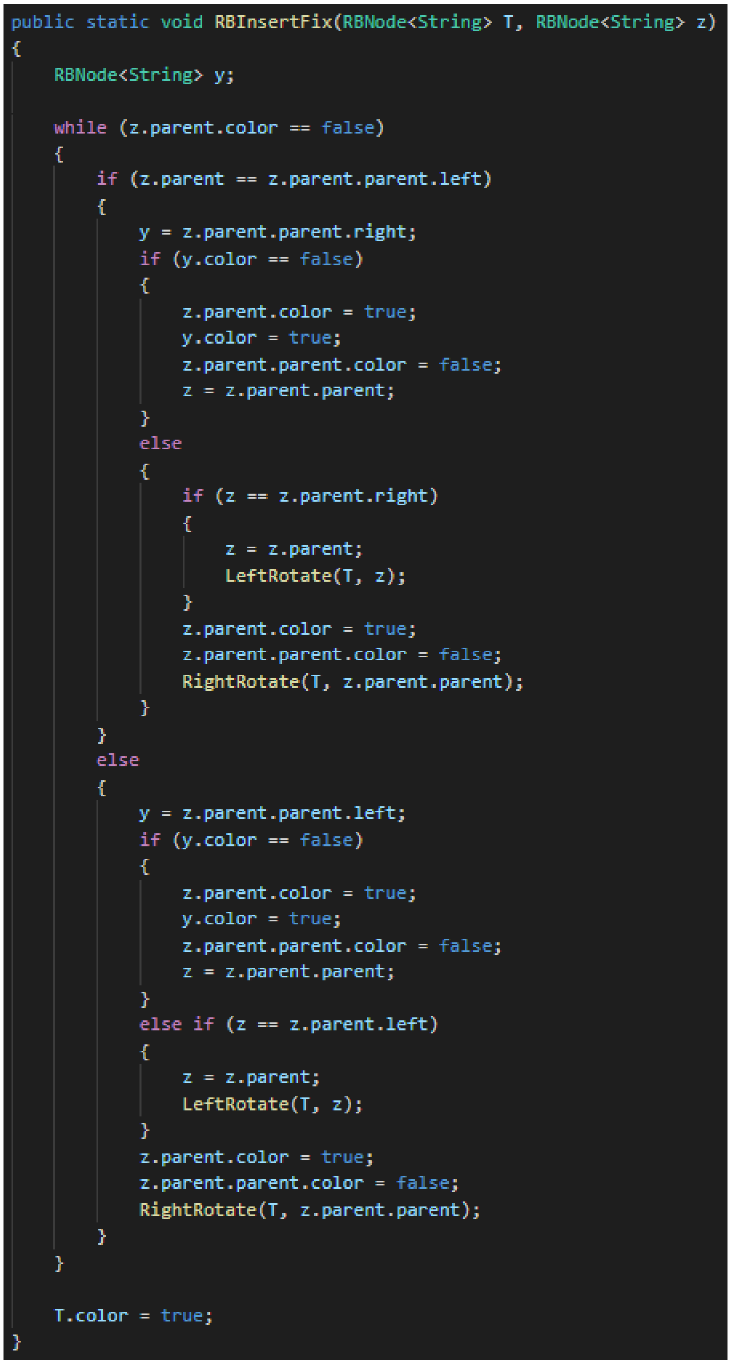
```java
package CS303.Lab10;

public class RBNode<T> extends BSTNode<T>
{
    Double key;
    T description;
    RBNode<T> left, right, parent;
    boolean color;

    RBNode(Double inputKey, T s)
    {
        super(inputKey, s);
        key = inputKey;
        description = s;
    }
}
```



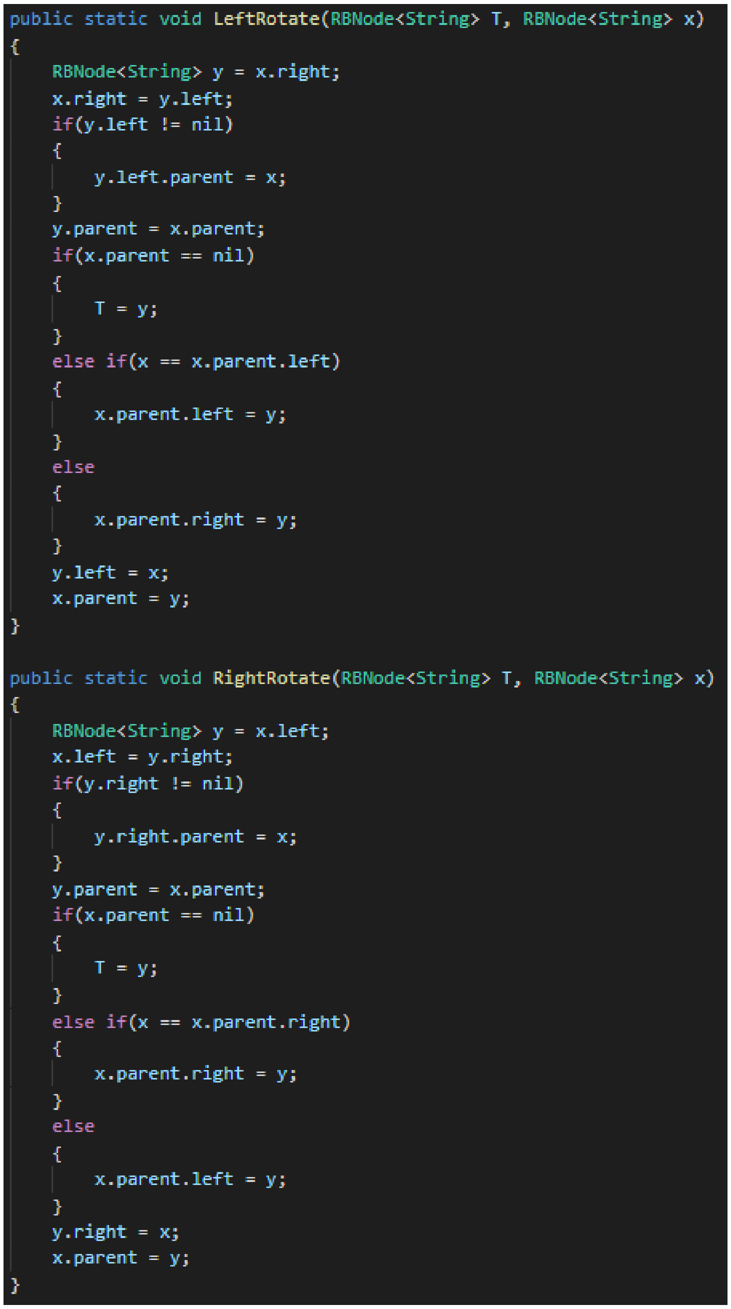
```java
package CS303.Lab10;

public class RBTree extends BST
{
    static RBNode<String> rootNode;
    static RBNode<String> nil = new RBNode<String> (null, null);

    RBTree(RBNode<String> root, Double key, String s)
    {
        super(key, s);
        rootNode = root;
        rootNode.left = nil;
        rootNode.right = nil;
        rootNode.parent = nil;
        nil.left = null;
        nil.right = null;
        nil.parent = null;
        nil.color = true;
    }

    RBTree(Double key, String s)
    {
        super(key, s);
    }
```



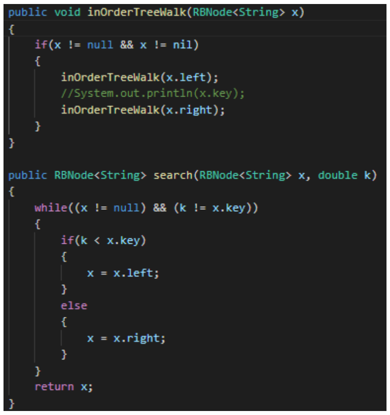
```java
public void RBInsert(RBNode<String> z)
{
    RBNode<String> y = nil;
    RBNode<String> x = rootNode;
    while(x != nil)
    {
        y = x;
        if(z.key < x.key)
        {
            x = x.left;
        }
        else
        {
            x = x.right;
        }
    }
    z.parent = nil;
    if(y == nil)
    {
        rootNode = z;
    }
    else if(z.key < y.key)
    {
        y.left = z;
    }
    else
    {
        y.right = z;
    }
    z.left = nil;
    z.right = nil;
    z.color = false;
    RBInsertFix(rootNode, z);
}
```



```
public static void RBInsertFix(RBNode<String> T, RBNode<String> z)
{
    RBNode<String> y;

    while (z.parent.color == false)
    {
        if (z.parent == z.parent.parent.left)
        {
            y = z.parent.parent.right;
            if (y.color == false)
            {
                z.parent.color = true;
                y.color = true;
                z.parent.parent.color = false;
                z = z.parent.parent;
            }
            else
            {
                if (z == z.parent.right)
                {
                    z = z.parent;
                    LeftRotate(T, z);
                }
                z.parent.color = true;
                z.parent.parent.color = false;
                RightRotate(T, z.parent.parent);
            }
        }
        else
        {
            y = z.parent.parent.left;
            if (y.color == false)
            {
                z.parent.color = true;
                y.color = true;
                z.parent.parent.color = false;
                z = z.parent.parent;
            }
            else if (z == z.parent.left)
            {
                z = z.parent;
                LeftRotate(T, z);
            }
            z.parent.color = true;
            z.parent.parent.color = false;
            RightRotate(T, z.parent.parent);
        }
    }

    T.color = true;
}
```



```
public static void LeftRotate(RBNode<String> T, RBNode<String> x)
{
    RBNode<String> y = x.right;
    x.right = y.left;
    if(y.left != nil)
    {
        y.left.parent = x;
    }
    y.parent = x.parent;
    if(x.parent == nil)
    {
        T = y;
    }
    else if(x == x.parent.left)
    {
        x.parent.left = y;
    }
    else
    {
        x.parent.right = y;
    }
    y.left = x;
    x.parent = y;
}

public static void RightRotate(RBNode<String> T, RBNode<String> x)
{
    RBNode<String> y = x.left;
    x.left = y.right;
    if(y.right != nil)
    {
        y.right.parent = x;
    }
    y.parent = x.parent;
    if(x.parent == nil)
    {
        T = y;
    }
    else if(x == x.parent.right)
    {
        x.parent.right = y;
    }
    else
    {
        x.parent.left = y;
    }
    y.right = x;
    x.parent = y;
}
```



```
public void inOrderTreeWalk(RBNode<String> x)
{
    if(x != null && x != nil)
    {
        inOrderTreeWalk(x.left);
        //System.out.println(x.key);
        inOrderTreeWalk(x.right);
    }
}

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