Noah Buchanan
Problem Set 5
Algorithms

October 6, 2020

1. Red Black Trees ensure that no path is more than twice as long as any other simple path in the tree. Either side of the tree could be represented as lg(n+1) as Binary Search trees are logarithmic in nature. Since we previously stated that no path is more than twice as long as any other path, the longest possible path (or height) could be at the very most 2lg(n+1). $2lg(1025)$ 20.something... take the ceiling of that we get 21.
2. There would be 2 red nodes.
3. The tree would not have the same black node count for all paths and thus would always break the last rule stating that all paths must have the same black node count, It would have to have $2^n - 1$ nodes if all were black for it to satisfy all conditions of Red Black Trees.

UARedBlackTree

note I slightly modified countRedNodes() and search() method signatures so that they could traverse recursively instead of iteratively.

```
import java.io.BufferedReader;
import java.io.FileReader;

public class UARedBlackTree {

        public final static UAEmployee nil = new UAEmployee(Color.BLACK);
        public UAEmployee root = nil;
        public int size;

        public static void main(String[] args) {

                UARedBlackTree T = new UARedBlackTree();
                T.readFile(args[0]);
```

```
        System.out.println("Red: " + T.countRedNodes(T.root) + "  Total: " + T.s

}

public void insert(UAEmployee z) {
        UAEmployee y = nil;
        UAEmployee x = root;
        while (x != nil) {
                y = x;
                if (z.id < x.id) {
                        x = x.left;
                } else {
                        x = x.right;
                }
        }
        z.parent = y;
        if (y == nil) {
                root = z;
        } else if (z.id < y.id) {
                y.left = z;
        } else {
                y.right = z;
        }
        z.left = nil;
        z.right = nil;
        z.color = Color.RED;
        size++;
        insertFixup(z);
}

public void insertFixup(UAEmployee z) {
        while (z.parent.color == Color.RED) {
                if (z.parent == z.parent.parent.left) {
                        UAEmployee y = z.parent.parent.right;
                        if (y.color == Color.RED) {
                                z.parent.color = Color.BLACK;
                                y.color = Color.BLACK;
                                z.parent.parent.color = Color.RED;
                                z = z.parent.parent;
                        } else {
                                if (z == z.parent.right) {
                                        z = z.parent;
                                        leftRotate(z);
                                }
                                z.parent.color = Color.BLACK;
                                z.parent.parent.color = Color.RED;
```

```
                                rightRotate(z.parent.parent);
                        }
                } else {
                        UAEmployee y = z.parent.parent.left;
                        if (y.color == Color.RED) {
                                z.parent.color = Color.BLACK;
                                y.color = Color.BLACK;
                                z.parent.parent.color = Color.RED;
                                z = z.parent.parent;
                        } else {
                                if (z == z.parent.left) {
                                        z = z.parent;
                                        rightRotate(z);
                                } else {
                                        z.parent.color = Color.BLACK;
                                        z.parent.parent.color = Color.RED;
                                        leftRotate(z.parent.parent);
                                }
                        }
                }
        }
        root.color = Color.BLACK;
}

public UAEmployee search(UAEmployee node, int id) {

        if (node == nil || node.id == id) {
                return node;
        } else if (node.id > id) {
                return search(node.left, id);
        }
        return search(node.right, id);
}

public void readFile(String filename) {

        try {
                BufferedReader br = new BufferedReader(new FileReader(filename))

                String line = "";

                while ((line = br.readLine()) != null) {
                        String[] x = line.split(",");
                        insert(new UAEmployee(Integer.parseInt(x[0]), x[1], x[2]
                }
        } catch (Exception e) {
```

```
                System.out.println("Failed to load");
        }
}

public int countRedNodes(UAEmployee node) {

        int count = 0;
        if (node == nil) {

                return 0;
        } else if (node.color == Color.RED) {
                count++;
        }
        count += countRedNodes(node.left);
        count += countRedNodes(node.right);

        return count;
}

public int size() {

        return size;
}

public void leftRotate(UAEmployee x) {
        UAEmployee y = x.right;
        x.right = y.left;
        if (y.left != nil) {
                y.left.parent = x;
        }
        y.parent = x.parent;
        if (x.parent == nil) {
                root = y;
        } else if (x == x.parent.left) {
                x.parent.left = y;
        } else {
                x.parent.right = y;
        }
        y.left = x;
        x.parent = y;
}

public void rightRotate(UAEmployee y) {
        UAEmployee x = y.left;
        y.left = x.right;
        if (x.right != nil) {
```

```
                x.right.parent = y;
        }
        x.parent = y.parent;
        if (y.parent == nil) {
                root = x;
        } else if (y == y.parent.right) {
                y.parent.right = x;
        } else {
                y.parent.left = x;
        }
        x.right = y;
        y.parent = x;
}

enum Color {
        RED, BLACK;
}

public static class UAEmployee {

        public int id;
        public String firstName;
        public String lastName;
        public Color color;
        public UAEmployee left;
        public UAEmployee right;
        public UAEmployee parent;

        public UAEmployee() {

        }

        public UAEmployee(int id, String firstName, String lastName) {

                this.id = id;
                this.firstName = firstName;
                this.lastName = lastName;
        }

        public UAEmployee(Color color) {

                this.color = color;
        }

        public String toString() {
                return id + ", " + firstName + ", " + lastName;
```

```
                    }
                }
            }
```