CIS 303 Analysis of Algorithms
Spring 2020

Christopher Rose
Assignment 5b

05/01/2020

Problem

For this assignment we were asked to implement a balance method and rotate methods. We were given the client code and the implementation for a Binary Search Tree. We are tasked with writing these new methods so that they may be used to implement an AVL Tree. We then need to observe differences in the run time for different amounts of nodes in the trees.

Hypothesis

My hypothesis for this assignment is that the worst case for the AVL Tree will be $\Omega(logn)$, because to insert elements into an AVL Tree takes $\Omega(logn)$ where n is the depth of the tree since the tree is always balanced. I also hypothesize that the worst case for the Binary Search Tree is $\Omega(n)$. My hypothesis will be supported if the experiments where we insert n elements have a depth of O(logn) and a run time of at most $\Omega(logn)$ for AVL Trees and $\Omega(n)$ for Binary Search Trees. If a significant number of trials do not have an run time approximate to this then it will not support my hypothesis.

Methods

Explain the details experiments you did to test your hypothesis. This explanation must include (where applicable, depending on the details of the problem):

1. My implementation:

```
private AVLNode balance (AVLNode rt) {
    if (rt == null) {
        return rt;
    }
    if (rt.right == null) {
        if (rt.left.right != null) {
            return balance(doubleRotateLeftRight(rt));
        }
```

```
        if (rt.left.left != null) {
                return balance(rotateRight(rt));
        }
    }
        if (rt.left == null) {
            if (rt.right.left != null) {
                return balance(doubleRotateRightLeft(rt));
            }
            if (rt.right.right != null) {
                return balance(rotateLeft(rt));
            }
        }
        rt.height = Math.max(height(rt.left), height(rt.right)) + 1;
        return rt;
    }

    private AVLNode rotateRight(AVLNode rt) {
        AVLNode temp = rt.left;
        rt.left = temp.right;
        temp.right = rt;
        rt = temp;
        return rt;
    }

    private AVLNode rotateLeft(AVLNode rt) {
        AVLNode temp = rt.right;
        rt.right = temp.left;
        temp.left = rt;
        rt = temp;
        return rt;
    }

    private AVLNode doubleRotateLeftRight(AVLNode rt) {
        rt.left = rotateLeft(rt.left);
        rt = rotateRight(rt);
        return rt;
    }

    private AVLNode doubleRotateRightLeft(AVLNode rt) {
        rt.right = rotateRight(rt.right);
        rt = rotateLeft(rt);
        return rt;
    }
```

2. To implement the balance method I put nested if statements for the cases where either the

left child or the right child is null and the grand-children of opposite tree were not.

3. My client code used an array of non-duplicating integers that get shuffled and inserted into both trees in the same order.

4. I ran and timed both experiments with the following values of N: 100, 500, 1000, 2500, 5000, 7500, 10000, 15000, 20000, 25000. The program prints the trees height, the trees, and the time it takes in milliseconds to insert the nodes into the tree.

Results

number of nodes	BST Height	BST Time(ms)	AVL Height	AVL Time(ms)
100	11	1	10	1
500	16	2	14	2
1000	21	3	17	2
2500	24	3	22	4
5000	27	4	21	6
7500	28	5	24	6
10000	27	6	24	7
15000	29	9	22	10
20000	32	12	25	11
25000	33	14	26	13

Discussion

According to the data shown, the worst case for AVL and BST run time is $\Omega(logn)$ as shown in the experiment with 25000 elements. $\log_2 25000$ is approximately 14 and in the experiment the run time is shown to be 13 for AVL and 14 for BST. Every other test yields a run time that is less than $\Omega(logn)$ meaning that they didn't surpass the worst case scenario.

Conclusion

The results did support my hypothesis, because the run times only were as long as $\Omega(logn)$. For more trials I would run more experiments with the same numbers of nodes to see if the run times were consistent.