Tushara Vepakomma

Student ID: 801044941

Date: 02/20/2018

Report for Algorithms and Data Structures Project 1

# Problem Statement in brief:

Build binary search trees and red black trees for the two input sequences given and compare the heights and average key comparisons of the resulting trees.

# Overview of Red black trees and binary search trees

**Binary search tree:**

A binary search tree is a tree in which all the nodes follow the properties as mentioned below:

1.The left sub-tree of a node has a key less than or equal to its parent node's key.

2.The right sub-tree of a node has a key greater than to its parent node's key.

The worst-case time complexity of search and insert operations is O(h) where h is height of Binary Search Tree. In worst case, we will have to traverse from root to the deepest leaf node. The height of a skewed tree may become n and the time complexity of search and insert operation will become O(n).

The efficient way to keep the height of a tree low is by balancing the trees. These are Red black trees.

**Red Black Tree:**

Red-black trees are self-balancing trees without affecting the complexity of the primitive operations. This is done by coloring each node in the tree with either red or black and preserving a set of properties that guarantee that the deepest path in the tree is not longer than twice the shortest one. The red black tree follows the following properties:

1.Every node is colored with either red or black.

2.All leaf (nil) nodes are colored with black; if a node’s child is missing then we will assume that it has a nil child in that place and this nil child is always colored black.

Both children of a red node must be black nodes.

Every path from a node n to a descendent leaf has the same number of black nodes.

Using these properties, we can say that a red-black tree which contains n nodes has a height of O (log n), thus all primitive operations on the tree will be of O (log n) since their order is a function of tree height.

Approach:

First the sequence of 500 random numbers are generated and then fed to binary search tree and red black tree. For binary search tree, for the insertion of keys, a new key is inserted at the leaf node. Once the leaf node is found, the new node is added as a child of the leaf node. Here we used a counter variable for the node. So, whenever the number coming as input is equal to the key value already present at the node, then the counter gets incremented by 1 at the node This counter logic is same for red black tree too. Later the height of the binary search tree is calculated which is the maximum depth. Average number of key comparisons is done by multiplying relative frequency of each key with its depth, summing all them and divide by 500.

The insertion for red black trees logic is inserting new node as a red node. The node at root should be black. If new node is red and and its parent is red, color of the sibling is checked. If sibling of parent is black, a rotation is performed. If the sibling of the parent is red, a color swap with grandparent will be performed. The height of the tree is calculated which is the maximum depth of a red black tree.

# Findings:

|  |  |  |  |
| --- | --- | --- | --- |
| Tree | Input | Height | Average no. of key comparisons |
| Binary search tree | Sequence of 500 random numbers | 14 | 6.65 |
| Red Black tree | Sequence of 500 random numbers | 9 | 4.088 |
| Binary search tree | Sequence of even, odd, random numbers | 51 | 31.408 |
| Red Black tree | Sequence of even, odd, random numbers | 10 | 3.896 |
|  |  |  |  |
|  |  |  |  |

# Conclusion:

From the results it can be inferred that the height of the tree can be kept low in red black trees by balancing the trees. Whereas if we use binary search trees, the height increases and the worst-case time complexity becomes O(h). The time complexity of red black tree is logarithmic function of n that is O(log n) which is better when compared to time complexity of binary search tree. So, it is highly preferred to build Red Black Trees keeping the height low and maintaining constant running time.

Note: The source code can be found in Random folder inside the zip file submitted. It has 3 classes Binary search tree, Node and RBTree