John Gill

Dr. Tala Talaei Khoei

CS 5008

25 July 2024

Question 3

The time complexity of Red-Black trees is O(log n) for the three main operations, those being insertion, deletion, and search. There are important properties of Red-Black Trees to consider before diving into the time complexity of their operations. All nodes are either red or black, the root is always black, red nodes cannot have red children, new nodes are always inserted as red, and every path from a node to its descendant leaves has the same number of black nodes.

So, for insertion, it works by inserting the node like it’s a standard BST since the tree is balanced. Some Red-Black Tree properties might be violated in that time, meaning some recoloring or rotations might be necessary. The insertion alone takes O(log n), and any fixing of the tree takes no longer than 2\*log(n+1).

Deletion in a Red-Black Tree also has a time complexity of O(log n), just like a standard BST. All the same statements about reordering said in regard to insertion apply here, too.

The search operation follows the same standard BST search procedure. The tree’s height is no more than 2\*log(n+1) as well, so the search operation will take O(log n) time.

Compared to theoretical bounds, an unbalanced BST can, in the worst case, have time complexity of O(n) for insertion, deletion, or search, because an unbalanced BST can degrade to become a linked list with linear time complexities. The balancing properties of Red-Black Trees ensure that the worst time complexity of any function for them is O(log n). It is for that reason that they are efficient for dynamic set operations. They balance complexity and performance quite well in practice, making them useful for applications like associative arrays, and priority queues, where dynamic data structures are necessary.