Jack Cobb

CS 3114

Project 1 Report

1) My overall design for this binary search tree relied on the comparators to correctly navigate the tree. I used a bst class that pulled the input and interpreted when commands to call from the database. The database had methods that were called based upon instructions from the bst class, and it further broke down those commands to correctly call upon the BinarySearchTree's methods, which handled all the actual work. The tree was made up of BSTNodes, which held city elements that had all the information; name, population, and payload.

2) I modified the bst so that whenever there was a node inserted to the left of the node, it would increase that node's left node count. It was placed in the insert method, so it got called recursively to all the nodes where it was inserted to the left of. When searching for the kth node, it was simple when moving left, however, if k was greater than the current nodes k value, findkth was called recursively to the right with a new value of (k – 1 – left node count). This allowed for only having to keep track of a left node count.

3) insert: Although insert calls a helper method, the helper method is the one that is recursively navigating through the tree, which using the comparators, it was able to determine which direction to go (halving the locations to insert), which results in O(logn)
find: Find traverses through tree and returns all elements that match the desired search term. This results in O(logn) time complexity because the traveling down the children eliminates half the nodes at each level because you can go left or right.
findKth: Due to the addition of a left node count for each node, findkth was implemented in O(logn) to find the node it requested.
findRange: Find Range finds everything equal to the bounds, or within the range. Since it has to check all possible nodes within the range, it is O(n) time complexity.
find, it also runs in O(logn), because the actual method that removes the node runs in constant time.
sort: Sort traverses the tree using an in order traversal, and prints out every toString for every city in the tree, so it runs in O(n) time.
tree: Tree is similar to sort in that is an in order traversal and it prints out the nodes, plus the spacing of the depth, so it also runs in O(n) time.
makenull: Since the makenull command just sets the root to null, thus clearing the tree, it runs in O(1) time.