Programming Assignment 4

Syed Abeer Hasan Zaidi

April 3, 2019

Program Description and Purpose of Assignment

The assignment was to build a Binary Search Tree using inputs from text files, and then calculating the average cost of finding a node in the BST that we made. The program was compiled using a makefile that I made, and it ran using the command **./run-tree** after compiling it. The fucntions involved in building this binary search tree are: **copy()**, **clear()**, **insert()**, **search()**, **get_size()**, **get_root()**, **update_search_cost()**, **inorder()**, **print_level_by_level()**, and **total_search_cost()**. The copy function works with the copy assignment operator and copy constructor to build a new Tree using an old one. The clear function works with the destructor, to deallocate all the memory allocated when constructing the tree. The insert function works on its own, as well as with the input operator, to add elements to the tree. The size function works on its own to find specific nodes in the tree. The get_size() and total_search_cost() functions work together to calculate the average search cost of the tree's made. The inorder() and print_level_by_level() functions print out the tree in two different configurations. I also made a height() function, to calculate the height of the tree, which I had intended to use inprinting out the tree level by level, but I never ended up using it.

The Data Structure

A Binary Search Tree is a Binary tree that organizes its nodes by the value inside of them. Each node has a left subtree and a right subtree; The left subtree consists of elements less than it, and the right subtree consists of elements greater than it. The Tree is defined recursively since each tree can be broken down into various subtrees, which are also made up of subtrees. In this assignment, the Binary Search Tree was defined as a class, and the nodes were described as a struct. The BST has a private data member, Node* root, that is a pointer to the root node of the tree. Each time an element is inserted into the tree, a pointer to the node is connected as a subtree.

Calculating Node Search Cost and Average Search Cost

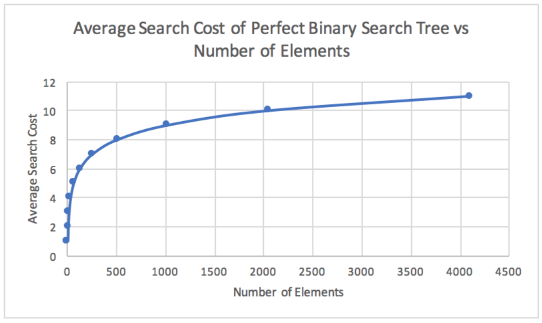
I calculated the individual search cost for the nodes by counting the number of comparisons performed when they were being inserted. I then double checked this calculation by traversing through the tree in the update_search_cost() function. Average search cost was calculated by first calculating the total search cost, by traversing through the tree and adding the individual search cost of every node, and then dividing that number by the return value from the get_size() function. The private data member, size, was also updated as elements were inserted into the tree.

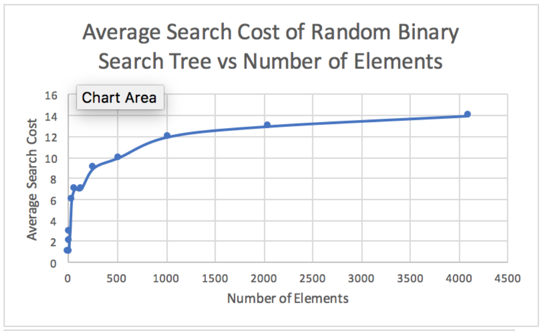
Average Search Cost Table

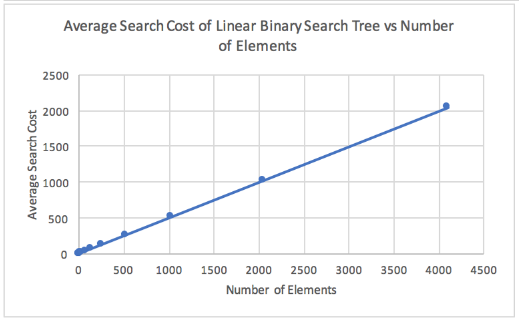
Average Search Cost of each binary tree

21		Type of Binary Tree		
# of Binary Trees	Number of elements	perfect	random	linear
1	1	1	1	1
2	3	1	1	2
3	7	2	2	4
4	15	3	3	8
5	31	4	6	16
6	63	5	7	32
7	127	6	7	64
8	255	7	9	128
9	511	8	10	256
10	1023	9	12	512
11	2047	10	13	1024
12	4095	11	14	2048

Graphing Average Search Cost







From the three graphs we can tell that the linear binary search tree has a average search cost of O(n), while the other two have average search costs of O(logn), which matches up with what is expected based on what we've been taught in the lectures.