Roslyn Davis

Computer Science

Project 3

13 November 2017

A Binary Way to Search

For project 3, I created an optimized and vetted program that uses a Binary Search Tree that compares two lists of items. In this case, I created 2 files that contain what courses I need to take and courses I’ve already taken. This can be used in hospitals for giving patients data, schools, or making financial decisions because this structure will lay everything out for you.

When coding this project, I first thought about the problem I was trying to solve. I wanted to find the best and most efficient way to give 5 outputs of information: items in both lists, items only in the first list, items only in the second list, items in the first list that is not in the second list and items in the second list that’s not in the first list.

The first thing I thought of creating was an AVL tree. AVL trees are self-balancing binary search trees that are balanced and have difference and some AVL trees have a deletion even though there are ways to get around it. I did recognize that an AVL tree could be useful and efficient. I also noticed that AVL trees have slower insertion and removal with an additional height constraint which I would also have to satisfy. However, I chose to do a binary search tree because my lists of information were already sorted. Unless the data is coming in a relatively random order, the tree can easily go down to be a linked list, which is its worst-case and can reduce the search time to O(n) which could take however long. This is also because my binary search tree is not balanced. However, I did create a “find” member function that handles my unsorted data.

Binary trees become truly useful when you balance them. This includes rotating sub-trees (LL rotation, RR rotation, LR rotation, RL rotation) through their root node so that the height difference between any 2 sub-trees is less than or equal to.

Binary search tree is used in various search applications where data is constantly entering/leaving, such as the map (which represents a data structure in which groups of unique key and collections of values are stored and each key relates to one value) and set objects (sets store values without a specific order) in programming languages' libraries. Using binary search trees is the best way to insert and remove data. For example, if I’m consistently trying to change courses to take in the summer verses courses to take in the fall or spring, I’d use a binary search tree because I just want to generate a quick answer. Or even to generate my GPA. However, if I’m creating a database where I have a bunch of random information that hasn’t been sorted, I need a program that will sort that information and give me those values.

I used a binary search tree because in this scenario, balancing wasn’t necessary. Normally, if a tree is unbalanced, then the time complexity to access an element of the binary tree increases. However, because we’re searching information that’s sorted through a system, teachers can go according to organized information and it makes it easier and decreases the time to search the information to relay to the student.

I implemented an in-order traversal because an in-order traversal uses the minimum amount of memory and calls the nodes in their expected order. I also used a vector in my standard library. To get outside information, I used “eof”. This means end of file. I also thought it was good practice for me to open and close my files which helped keep information safer. In average cases, my program works in Big O of: O(logn).