**Final Project**

James Dunaway

CS-250-T3201 Software Development Lifecycle 21EW3

Professor Hodde

06/16/2021

Data Structures

Vectors are similar to arrays, the main difference lies in the constant size of the array versus the dynamic size of the vector. For this reason, I have found myself working with vectors whenever possible, therefore, am very comfortable with their implementation. For example, the second lab required that a CSV file be loaded into a vector of bids. This program does not need to sort the data, but it efficiently stores each ‘bid’ item from the parsed CSV into the vector using simple, readable, and modular code that does no more than it needs. It can be easily changed to take in a different struct or load/display more information as needed such as when the display of clock ticks was added.

The program created for the fifth lab includes the use of vectors and a hash table, further encapsulating my knowledge of vectors and displaying quick learning of hash table functions. The hash table uses a simple key calculated by the modulo of the hash table’s size but was implemented in a modular fashion; the hash table size can be easily changed and doing so will automatically reflect in the calculation of the key. Using a vector of node objects, this hash table is organized using an efficient method that allows for fast searches/inserts/removals.

Lastly, the program created for the sixth lab required the implementation of a binary search tree and is an example of my ability to create one that is efficient, readable, and modular. This binary search tree implements a node vector allowing for efficient storage and access with the advantage of easily sorting the bids due to the binary search tree's inherent organization. For better modularity, this program also implements the use of public/private methods throughout splitting the responsibility of a larger function while making it more secure.

Algorithms

The program created in the sixth lab includes the simplest, most efficient, and technically sound search method. It takes a string object, bid’s id number, sets the current node to the root of the binary search tree then compares the given id to each id in the tree. This is coupled with a simple while loop containing a single if, if-else, else statement, each of which compares the given bid id to the one attached to the current node. This reliably returns the specified bid or reaches the bottom of the tree and returns nothing. This is efficient due to the design of a binary search tree, which allows the programmer to deduce which node to check next based on a simple comparison

The program created for the fourth lab includes a quicksort method utilized on a vector of bids, intending to quickly sort the bid vectors of varying in size. As with any efficient quicksort method, this recursively calls both itself and a separate partitioning method; wherein the quicksort method sorts the partitioned data sets using recursive calls on both itself and the partition method until it has been sorted. This is done efficiently with an average performance of O(N log(N)) while also being self-contained and human-readable.

Hash tables are a great way to store/sort a larger data set or if you are looking for a higher level of security. The program written for the fifth lab showcases my proficiency with hash table algorithms. The insert method first calculates the key, checks for that node and only moves forward if it is not already found; in which case, it will make multiple checks while moving through the hash table. It efficiently takes action in a logical order, eg. first checking if the node is empty and if it is, simply adding it there instead of continuously running the algorithm making unnecessary comparisons.

Students Choice

The final program written for this class was tasked to implement a working binary search tree to search/remove/insert data in a large data set quickly and efficiently. The program I wrote did just that, using search, insert, and remove methods on a vector of nodes. The node data structure is implemented using best practices such as multiple constructors, concise language, and shorthand where applicable. As stated previously the search method takes in only a bid id (which could be changed easily) and runs a simple while loop making comparisons along the way to return the requested bid or nothing. The insert/addNode method, split the functionality across two methods, the first solving the scenario where the given tree is empty and merely inserts the new bid as the root. In contrast to the addNode method which moves to the left or right of the current node based on the compared values, again, by using concise language, built-in C++ functions, and simple if/else statements.

The remove/removeNode functions are much more complex but still human-readable and efficient. The removeNode function is where the removal happens; it is done by first checking if the tree is empty and if so exiting the function otherwise it follows an if/else-if/else branch. This branch is used to recursively call itself to move along the tree until the provided bid/node is found. Once it is found another if/else-if/else branch is run to determine if there are or are not children which nodes need to be moved if at all and takes those necessary actions.

The functions in this program are written to be as modular as possible while not deviating from the needs of the program; each function stands on its own except for a public/private function combo for the insert and removal functions. Though the functions are not as reusable, for example, each function takes a string object, therefore, each function is specifically designed with a string in mind. The search method is using the C++ .compare method that compares the value of two strings. Granted, it shouldn't be too difficult to adapt these functions to another type such as a float though it may be just as easy to simply write a new function. The use of the compare method is also an example of using the most logical comparison based on the problem being solved. Each function is self-explanatory but well commented and follows a logical sequential order whenever possible. The removeNode method first checks to see if the tree is null, this is an example of a simple, chronological design that also helps avoid hard-to-find bugs.

Conclusion

A data structure is a form in which various objects/data are stored in a computer or program and is an essential part of any program. There are many ways to store data with varying strengths and weaknesses so they must be heavily considered during development. I tend to use vectors due to their flexibility, but they are not going to be the best choice in every scenario; if you’re working with a smaller, fixed-size, data set an array works whereas it may not work well with a larger data set that needs to change in size.

Algorithms are a way to take repeatable actions in a program, such as searching through as a list. A programmer's implementation of various algorithms could make or break the efficiency and speed of the program. If a developer needs to search through a 50 element list, using a selection sort algorithm will result in an average runtime of O(N^2) or O(2500) which will execute quickly on most modern machines but the same algorithm used on a 10,000 element list will take O(100,000,000). This is an example of how much influence an algorithm has on the performance of a program, for comparison, a binary search tree may require as little as 14 comparisons depending on the height of the tree.

In the past, I would purely work with what made sense in the moment or what I was comfortable with and this would usually get the job done but when working on a larger project small decisions can have a huge impact. I develop mostly in my personal life at this point but plan to hold a career in the field soon so I will be utilizing this knowledge most if not every day on various projects in a multitude of scenarios. I have already done so on a personal C# project, I was struggling in the past with an efficient way to search through a list of objects, and due to the quicker runtime and smaller list, I used a quicksort algorithm. This class has opened my eyes to the importance of data structures and algorithms with specific examples of how much time can be saved and what to look for when you are deciding how to tackle a problem.