Stephen Frueh

CS-260

Final Project

Portfolio Reflection

**Portfolio Reflection**

Throughout this course, we were introduced to a variety of data structures and algorithms used on them for many different purposes. Each of these is useful in their own situations, and knowing when and where to use each one is just as important as knowing they exist at all. While none of the structures and algorithms are exclusive to the C++ language, they make it possible to make progress efficiently without having to reinvent the wheel each time you need one.

Some examples of data structures I worked with during this course are vectors, hash tables, and tree structures. Each of these data structures are different internally in how they handle and store information, but each is essentially nothing more than a collection of information. They are each built to store data differently, which depending on how much data is stored, how often it’s changed, and how often it’s searched, can be incredibly more or less efficient depending on each situation. Vectors are simply long lists of data stored one after the other. Adding elements can be done easily, usually at the end, like a queue. A hash table is a table of data values, mapped by some sort of identifying key. How that key is generated can be many different ways, but basically acts as an ID for the data it’s representing. Lastly, trees are used to store values in nodes, with each node having one parent and two children.

All of the data structures wouldn’t be very useful without the algorithms that work on them. Most of the algorithms do the same thing on each structure ( such as insert, delete, or search ), but how they do it is likely very different because of the structure and how it’s uniquely set up. Searching through a vector unless it’s sorted is just done linearly, starting at the first node and moving onto each node after it in order. Sorting can be done as you add values, with a variety of sorting algorithms. Generally this is done with the lowest values at the start of the list, with the higher values at the end, but they can be sorted to any requirements. Hashing can also be done through any means depending on the requirements. Whether using a simple modulo operator on a value, or something much more complicated, as long as the values being stored have some identifier to represent them as a key, any method will work. If you want no conflicting keys to make every key unique, it can be much more difficult, but you can also store lists of values that share keys under that spot in the hash table. Trees are filled with nodes chained together, which can be a little more complicated but is a simple idea. A root node starts the tree, and values are added as child nodes to that root, which becomes a parent. When removing values that aren’t leaf nodes ( nodes with no children ), it’s important to make sure the tree is reconnected to the sub-tree below the node removed.

My favorite program that I worked on during this course was the binary search tree program. This one was my favorite because I like working with recursion, as it’s a powerful and tricky thing to use, especially correctly. It can solve many confusing problems that require the same thing to be done over and over, differently than how a loop repeats tasks.

Overall, I enjoyed this course because it finally started using powerful programming languages I enjoy. I’ve worked with C++ for a while personally, and I enjoy it. The data structures and algorithms we learned in this course are important tools for any programmer using this language to know. Without them, it would be incredibly difficult and time consuming to get any work done. The real tricky issue with these structures and algorithms is knowing what each of their respective strengths and weaknesses are. For example, using a sorted hash table and frequently inserting values could be tough for the hash table to do quickly and efficiently. Each of these structures and algorithms are like tools in a tool box, and the mastery comes from knowing exactly what each does, how it does it, and therefore when each would be the most useful. One of the more important things when programming is not re-inventing the wheel with every project or every task. These collections and algorithms can be used and re-used across many projects in this language, and even adapted to other languages just by making some simple changes and adapting the syntax. Instead of storing bids, the data can be made a template to store any kind of data, and since the comparison operators used in the algorithms such as greater than, less than, or equal to can be overloaded by each type of data, they can be made to work on anything that defines how those operations work.