Renaldo Musto

Final Reflection

I think that the program that best exemplifies the data structure of the vector is the vector sorting program we had to complete. The use of a vector allowed me to store all of the data I needed in just a few variables making it much easier to work with. For example we had to apply a sorting algorithm over the bids which required me to iterate through every single bid ID. If I did not use a vector to store the bid information I would have had to create a variable for each separate bid which would have been extremely time consuming and messy to sort through. In a vector the bids could all easily be accessed from one place allowing me to simply call the sorting algorithm on the data once and deal with all the necessary data at the same time. I think that the hash tables program is the best example of the use of the hash table data structure I have. I was required to implement a search algorithm over the .csv data and the hash table made this a relatively simple process. In a hash table data can only ever be in one possible bucket and that bucket is very easily found using one simple modulo calculation. Then from there you only have to perform a linked list walk over a few potential nodes to get the data you need. So unlike other data structures we can pretty much immediately get to where we need to be in the hash table after just one line of code making searching very simple. The program I have that is the best example of the tree data structure is the binary search tree program that I completed for this course. In that program I was required to implement a search algorithm over the data in the .csv file. Having this data loaded into a tree before searching made the process much easier than if we were to use something like a simple linked list. For example, because the data in the tree is somewhat organized based on which direction down the tree we traversed when entering the data we do not have to search through much of the tree to get to where we need to be. Checking how the bid we are looking for compares to the bid at the node we are at let’s us know which way to continue to look down the tree, speeding up the process dramatically.

The best example I have of the search algorithm is in the hash table program simply because of how simple it was to make and understand. In this algorithm there are only a few basic steps. First we take the bid we want and run it through the hashing algorithm. Now that we have the key where the data would be in the hash table we check there for a match. If data is found but no match is there then we walk the list to see if it is there if no data is found at the key then the search terminates. I think that this is effective because of how unambiguous it is. The data could only ever be where the key was and from there it is only a matter of checking the list for a match. I think that this technique of searching offers very little room for errors and is very quick and efficient using the most limited amount of resources since we are searching so little of the actual data we have. I think that the best example I have of the sort algorithm is the vector sorting program I completed. In this project we implemented the quicksort algorithm which I believe was the fastest one I ended up creating. This algorithm is so fast because it is able to ignore most of the data through a process of elimination. The data is first partitioned so that it is organized from high to low values. Then the midpoint is found and the amount of available data to search is constantly cut in half depending on how the searched value relates to the midpoint. This results in extremely fast sort times hence the name. The best example I have of the hash/chaining algorithm is in the hash table program. This algorithm is technically sound and functional because while it may take some extra time to write it is very effective at what it does, which is hashing the data. The algorithm is able to first find the key for the data using the modulo function and then now that we know the correct location of the data we can check to see if the location is empty or not and decide whether to simply place the data or chain it from there. When we chain we simply walk the linked list and check each value, while this is a slow process we only have to go over a few nodes so it is not a big deal.

My overall favorite program would have to be the hash table program. I think that this is the most effective program because it makes the best use of the data structures and algorithms that are employed. In this program I had to first create the empty hash table structure which involves setting up the keys, nodes and their pointers to default values so the program can have a clean start. The hash table is interesting because the algorithm it employs is very much a part of the structure and gives it some of its functionality. The algorithm for inserting, removing or searching a value from the hash table requires us to calculate the key for the item being searched which is called hashing and then we need to potentially use chaining from there which is essentially linked list walking. The program makes the hash table and hashing function work well together because the hash table gets its effectiveness from the hashing algorithm. Since the hashing algorithm usually only produces one key per value, searching a hash table becomes very simple as we only ever have to check a few nodes to get our potential match or confirm the entry does not exist. I coded this program modularly by keeping all the functionality organized to just a few functions and having the functions being as complete in and of themselves as possible. For example in the printall function the function doesn’t simply contain the ability to print but it is a complete package and therefore fully modular because it can also iterate through the data and make sure it printed everything. My use of functions also helped to keep the program reusable. It would not be difficult to take the functions in this program and apply them to a different program. For example the basic structure of a hash table will not change from application to application so we can use basically the exact same insertion and remove function we have here. We may just need to change the variables names and how the key is hashed out, but the overall logic would stay the same. I think that my annotations should keep my logic throughout the program clear to follow. I always made sure to make clear what my intentions were with each aspect of the code I wrote. An example of this would be the search algorithm starting on line 200 which I feel is the most well annotated part of my program. Each line and step is thoroughly walked through, I feel how the search is conducted is very well explained.

In conclusion, data structures appear to be an important and at times crucial aspect of developing a program. For example in the case of a vector we could theoretically make a different variable for every aspect of a bid but it is much easier to put all the ID’s and titles, etc into one vector that can be accessed all at once. Algorithms are also an essential aspect of developing effective code. A well written algorithm has the power to dramatically change the speed and efficiency of a program. I think the best example of this in my code is the vector sorting program. The difference in time between the selective sort and quicksort algorithm was rather dramatic especially on the larger .csv file. Since we are dealing with such small datasets and such a big difference is still apparent I think it would be easy to say that an effective algorithm could make or break a larger program. We could be talking about adding potential hours to a program's runtime. I learned quite a bit from my time in this course. I think that if I were to apply my knowledge to another aspect of my life I would mod my games better. Modding is something of a hobby I have and understanding basic data structures and algorithms has fundamentally changed my understanding of what can be done with code. Something like an item inventory once seemed unapproachable but now I can see how I could easily organize all my items in something like a vector and search through them using an algorithm. Inventory sorting is also something I've always wanted in some games and I can now see how that could be easily implemented using a sort algorithm.