Southern New Hampshire University

7-1 Final Project Reflection Paper

Ryan DeBraal
Southern New Hampshire University

CS-260-Q3396 Data Structures and Algorithms 20EW3

Southern New Hampshire University

For this final project I chose to consolidate Module Three, Five and Six into a single project and allow the user to select which *mode* they would like to utilize in order to load, read, search and delete bids from a data structure. This was a valuable experience for me because as I revisited each methodology and directly compared them against each other I became much more familiar with the pros and cons of each data structure type.

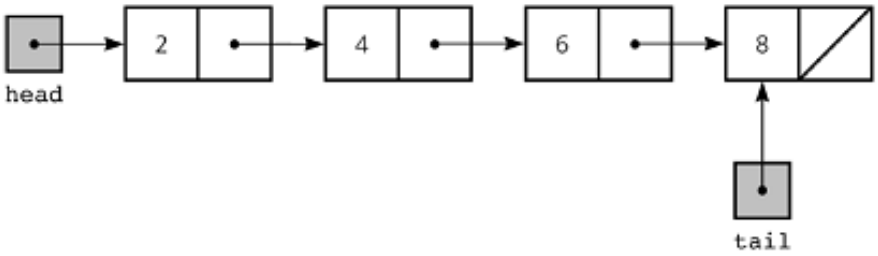
Data Structures

Main.cpp in the FinalProject solution best exemplifies my knowledge and understanding of the data structure types we learned about in this course.

The **Linked List** mode pulls the data from the CSV and stores it in a data structure with a dynamic size. However, this methodology has several drawbacks:

- They can only be iterated across in one direction, from the beginning or "head" to the end or "tail", this means there is no "random access" allowed.
- A pointer to the head node must be constantly maintained.

Fig 1. Linked Lists

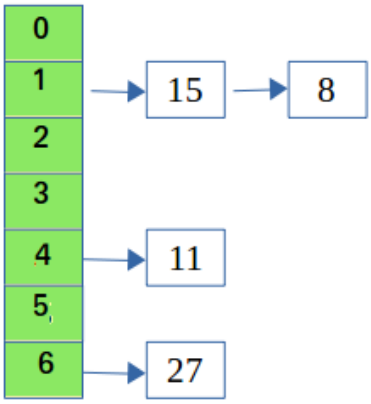


The **Hash Table** mode pulls the data into a data structure which utilizes a vector to map a value to a "hash" and then later retrieve that value via its hash. This allows data to be retrieved very quickly. A hash value is often derived using this simple formula: hash = key % noOfBuckets

By being able to simply "lookup" a specific data point in a collection, the ability to retrieve information becomes straightforward. This is a significant improvement over having to iterate across a collection until a match is found.

To avoid collisions, each "cell" of a hash table is *actually* a linked list of records so that data can be retrieved from a certain hash index in the same manner as the aforementioned linked list.

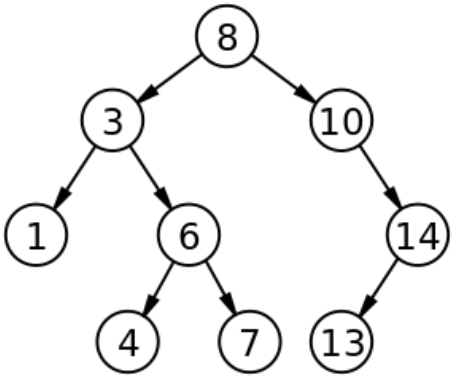
Fig 2. Hash Table with Chaining



The **Binary Search Tree** mode stores data in a more complex way and is therefore a little more unwieldy but search capabilities are significantly improved because of the way the data is stored and retrieved.

In a Binary Search Tree, each node contains a value from a *well-ordered set*. Every node in the left subtree contains a value which is smaller than the value in the current node, and every node in the right subtree contains a value which is larger than the value in the current node. This allows for simple boolean logic to quickly search for and retrieve data because the data is being *navigated through* rather than *iterated across*.

Fig 3. Binary Search Tree



Algorithms

Main.cpp in the FinalProject solution best exemplifies my knowledge and understanding of the algorithms we learned about in this course.

Search

All three data structures (linked lists, hash tables and binary search trees) can be adapted to search for and retrieve values contained within them. The **Linked List** mode is straight forward

but limited in its scope, this kind of data storage and retrieval is only really viable for a collection of perhaps a few hundred entries. The **Hash Table** mode has significant advantages over the linked list method; however, it also inherits the linked lists disadvantages because the chained hash tables still utilize linked lists, and therefore the space overhead of the "next" pointer can be significant. The **Binary Search Tree** mode does away with the limitations of the linked-list but it also far more complex to work with, it also requires a "well-ordered set", meaning the performance of the search algorithm is directly related to how well the data being stored is organized.

Sort

Sorting of a linked list requires a rather convoluted methodology, because a *singly* linked list can only be iterated across in one direction, this would require a separate object to be instantiated and maintained in order to capture values that are "above" or "below" the current pointer. (GeeksforGeeks, 2020)

Fig 4. Pseudo Code for sorting a linked list

```
1) Create an empty sorted (or result) list
2) Traverse the given list, do following for every node.
      a) Insert current node in sorted way in sorted or result list.
3) Change head of given linked list to head of sorted (or result) list.
```

Sorting of a hash table isn't necessary because as mentioned previously the data structure depends on "hash lookups" rather than iterating across the data.

Sorting of a Binary Search Tree would be antithetical to its very nature. A binary search trees is dependent upon the data being organized by branching logic.

Hash/Chaining

Hash Chaining is a clever workaround to the limitations of linked lists and the complications of binary search trees. While "under the hood" it seems somewhat *clunky* in its implementation, ultimately the ability to lookup data values by hash is a worthwhile endeavor that can make an application run faster and easier to develop.

> "Search time for hash table lookups are reduced significantly, searching (or inserting / removing) an item may require only O(1), in contrast to O(N) for searching a list or to O(log N) for binary search." (zyBooks)

Effectiveness

Main.cpp in the FinalProject solution was my favorite program to write, because it allowed me to "take off the training wheels" and code something that wasn't prescribed. I liked creating the "modes" for each data structure type and organizing the different functionalities into discrete

methods. I wish I had had more time to make the code mode modular. I ended up sticking all the classes I created into a single .cpp file which made it quite long and harder and harder to maintain as time went on. I didn't take the time to learn about the relationship between .cpp files and .h files, so "importing" classes was out of the question. I ended up using regions to better organize code, but if I were to continue development on this project I would move the LinkedList, HashTable, and BinarySearchTree classes into individual files.

I think the code is highly reusable, all the data structure implementations can be adapted to another project simply be swapping our the Bid struct for another object that needs to be maintained in a collection.

I am pleased with the annotations I placed in the code base, I try to code based on a motto I heard several years ago: "verbosity over brevity", To me this means it does not serve anyone better to write code that isn't cleanly organized and well documented. I think software developers are often tempted to try to condense their code into the smallest amount of space as possible, especially with the advent of LINQ. Coding something that works is the most important, but coding something that can be easily interpreted by another developer is a close second.

Conclusion

When building something, either in code or in the real world, it is important to use the right tool for the job. Knowing which kind of data structure to use will save development time and result in cleaner, faster code. In this case, I think the eBid Search application that we created was best organized using the Binary Search Tree data structure because each entry has a gaurenteed BidID, I would modify the BidID to be an integer value rather than a string value because that would allow the BST to more quickly determine which branch (left or right) to go down, I imagine the string manipulation of the unique identifier would eventually have a significant impact on search speed.

I was routinely surprised by the amount of effort put into creating things that I routinely take for granted in my day-to-day work developing applications in C#. Dictionarys, LINQ, NO Pointers. These are just a handful of examples of the kinds of tools that I routinely us and never give a second thought.

This course was definetly one of the most educational for me personally. I have been a professional developer for over a decade now and cut my teeth on JavaScript and Visual Basic 6, so I never took the oppurtunity to go back to something like C++ and learn about the underpinnings of computer programming languages.

Southern New Hampshire University

Citations

Insertion Sort for Singly Linked List. GeeksforGeeks, 3 Feb. 2020, www.geeksforgeeks.org/insertion-sort-for-singly-linked-list/

InterviewBit. “Linked Lists.” InterviewBit, www.interviewbit.com/courses/programming/topics/linked-lists/

Nilsson, Stefan. “Binary Search Trees Explained.” · YourBasic, Yourbasic.org, www.yourbasic.org/algorithms/binary-search-tree/

ZyBooks, https://learn.zybooks.com/zybook/SNHUCS260AY16-17/chapter/5/section/1