Paul Kenaga

Professor Ostrowski

CS 300

19 Nov. 2021

Hash Tables Code Reflection and Flowchart

The code for this assignment demonstrates the time complexity for the insert, delete, and search functions of a hash table that uses chaining to handle collisions. After calling the load bids method, a CSV file is loaded using the CSV parser and objects for the data structure are created from the strings within each row. The insert function is called and the objects are assigned a key that maps to a bucket where the object can be found. If the bucket is empty, a node is created and the new object becomes the first node at the index. If the bucket is not empty, the next pointer for the last node in the bucket points to the new object and the new object’s next pointer is set to empty. This is a quick method for inserting elements with an average time complexity of O(1). There is no need to traverse the whole structure to locate the index where an item will be placed. The delete and search operation perform similarly in the sense that the use a key to locate the index of an element and do not move from one index to another like an array. It is possible, however, for elements to all be at the same index and for the time complexity to exceed O(1). This can be mitigated by using a different hash function.

While developing this code, I came into an issue with building and running the program. My builds failed and my code would not run. I received an error message stating that a file could not be located. After researching the problem, I found that the uninitialized hash table was triggering a potential uninitialized variable warning and my Visual Studio settings had Security Development Lifecyle checks enabled. Disabling this check allowed my code to finally run. Another issue I ran into was the print method. At first, calling the print method displayed duplicate bids under keys that had mapped to more than one element. The first bid would be printed a number of times equal to how many bids were assigned to that key. A quick change from node->bid to node->next->bid in the output statement corrected the issue.

Another issue I encounter involved the remove function. Ideally I wanted to remove only the specified bid when the function is called, however, I can only figure out how to remove all bids and they key. I researched for hours over the weekend to try to find a resource to help resolve this issue but could not find one. If there are two objects at an index and I want to remove the first one, I need to make the second object the head somehow. If I want to make the remove the second object, I need to change the first object’s next pointer to null. I tried using calls to delete and erase, but they did not execute properly. Using “head” did not work either. I am not as confident with hash tables as I was with linked lists and would appreciate any material that can help me grow my understanding of them.