CS300 Data Structures

Kemal Sarper Yücel

21031

Homework 3-Hashtable

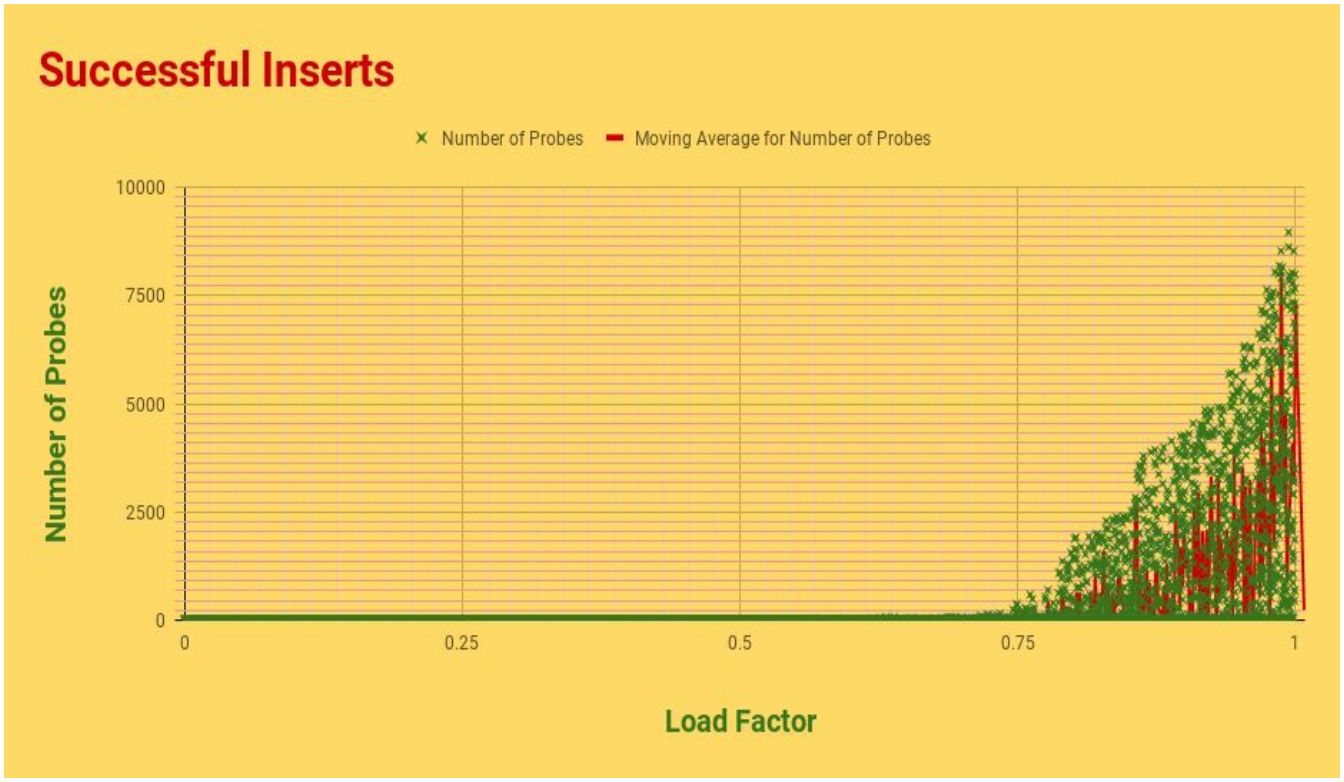
I used the hash function hash(x) = x mod M where x is the entry to the hash table and M is the capacity of the array in the hash table. I choose M to be 10007, because it is what nextPrime(10000) returned. I added 3 new private data items:int currProbes(The latest number of probes used for the given transaction), int numEntries(The number of entries in the hash table) and int size(to keep track of the capacity of the hash table). The methods I added were int Hash(const int&x) as mentioned above and bool IsFull(). I removed the resizing attribute of the hash table as well as findpos for the purposes of this assignment and changed insert, remove and find accordingly. My insert function does not skip over the entries which are DELETED, my find and remove member functions do. I implemented linear probing with the code “current = Hash(current +1);” where current is the current index the code is probing.

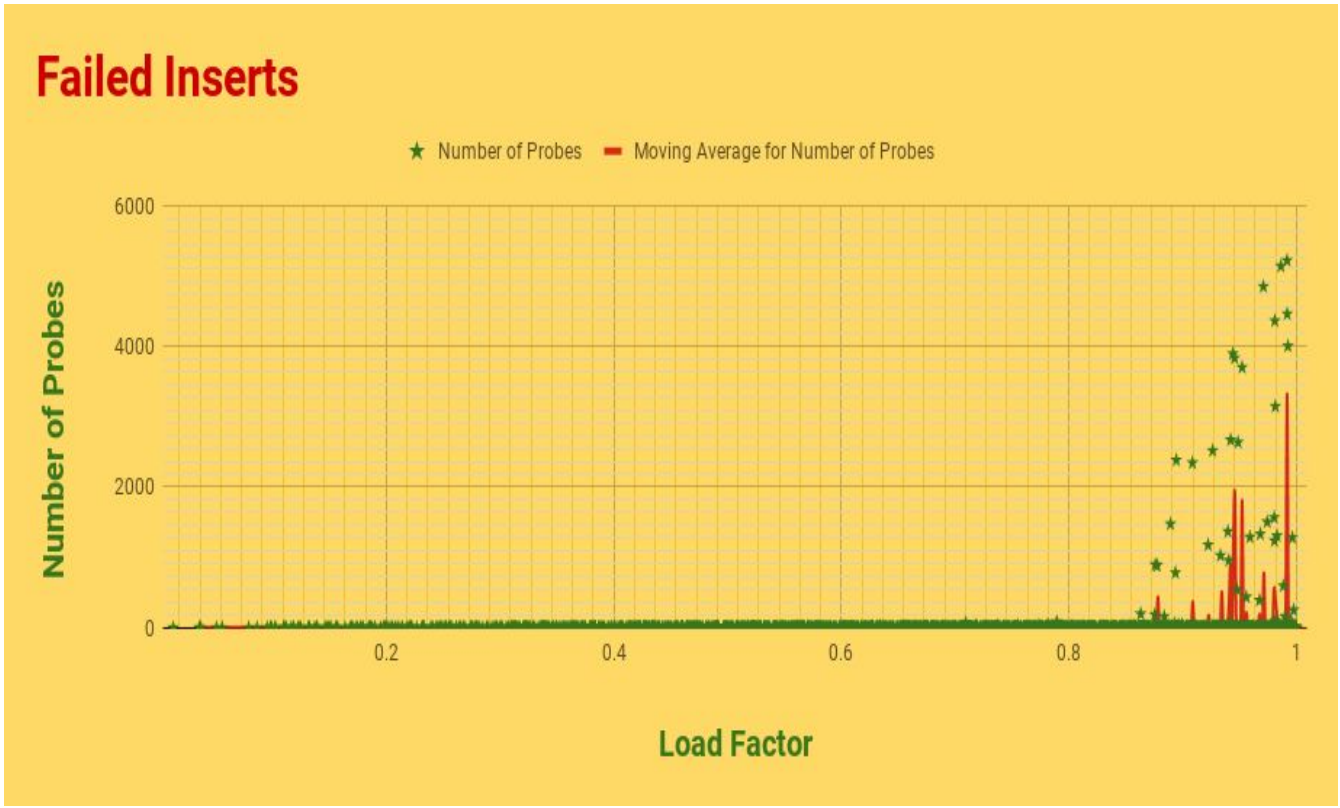
After each transaction I printed the total number of entries in the hash table along with the total number of probes for that specific transaction to the relevant table(Successful Inserts, Failed Inserts, Successful Deletes, Failed Deletes, Successful Finds, Failed Finds). From there I used Google Sheets and divide the column containing the total number of entries to M to compute the load factor. Then I plotted the total number of probes versus the load factor.

From the graphs in later pages, i concluded that the number of probes required for transactions starts rising at an alarming rate when the load factor reaches 0.75. The conclusion that can be reached is that it is probably best to resize the hash table whenever the load factor reaches 0.75. Waiting for the hash table to be full to resize would be very inefficient in terms of time complexity compared to this proposition of an earlier resizing.

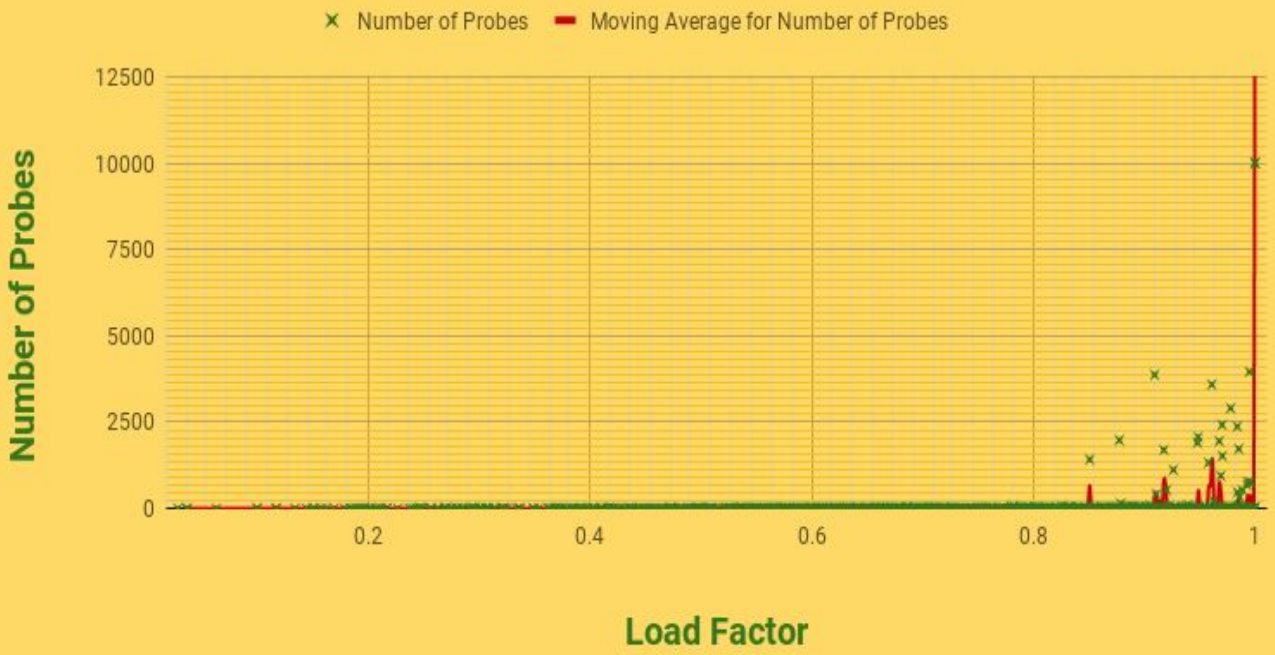
Also form the graphs, it is clear that successful inserts require more linear probing than failed inserts, failed deletes require more linear probing than successful deletes and failed finds require more linear probing than successful finds. This is clearly due to the method of traversing through the array. An insert transaction will always be successful unless the entry is already in the hash table for this homework, because linear probing ensures that a place will be found for each insert transaction until the table is filled. So, the failed inserts here are a result of the entry already being present in the table. Since the remove method calls the find method inside its code, it is to be expected for both failed finds and failed deletes to require more linear probing than their successful counterparts. They require an EMPTY entry or a traversal of the entire hash table to fail, so it is logical that failed instances of them require more linear probing.

Below are all the aforementioned graphs:





Successul Deletes



Failed Deletes

Successful Finds

Failed Finds