Report

*Comments on running: enter a single parameter in quotes with words separated by a single space in the form (“ <size> <probing type> <file name> <number of keys>”). Probing type is case sensitive and must be written as either “linear” or “quadratic” or “chaining”. E.g. HashMain “691 linear cleaned_data.csv 4”

Object-oriented design

I created the HashTable class for all the methods needed for implementing hash table. It contains the Hash method for the hash function. It contains two HashTable constructors. The first is for linear and quadratic probing and the second is for chaining. It also contains the three different insert and search methods for linear probing, quadratic probing and chaining. Quadratic probing uses the updated method.

I created the HashBank class to contain helper methods needed for the hash table implementation or the HashTable class itself. It contains the loadFactor method to return the load factor. It contains the IsPrime method which checks if an integer is a prime number (source: https://www.mkyong.com/java/how-to-determine-a-prime-number-in-java/). It also contains a toInt method which converts a string in the dateTime key format, as per the cleaned_data.csv file, to a six digit integer.

I created the node class to construct a node that stores a key and a value.

I created the Implementation class to implement the hash table. It contains 3 methods. One to run a linear probing hash table, one for quadratic probing and one for chaining. Each method in Implementations takes the table size (integer), a File object and a key integer as parameters.

All three methods in Implementation work the same:

1. Creates a HashTable (table) object
2. Creates an ArrayList (keyBase) to store keys from a shuffled key array
3. Create a scanner object using the data File. Then iterate over this scanner to insert every line of data to table as a key-value pair and also append the keyBase ArrayList.
4. Shuffles keyBase ArrayList.
5. Lastly iterates to print the first k outputs from the search method for the type of probing.

I created the HashMain class to accept user input in the form of parameters and then execute its main method. It contains only its main method. It accepts one line of string of input as a parameter in terminal. It then checks if the size entered is a prime number and then if the probing type entered is correct. It then runs the method in the Implementation class corresponding to the type entered with parameters as entered.

I created the InsertTest and SearchTest classes to test the search and insert function of the hash table. They are similar to the Hashmain class but they output the loadfactor and probe numbers instead of the values returned by search.

Experiment

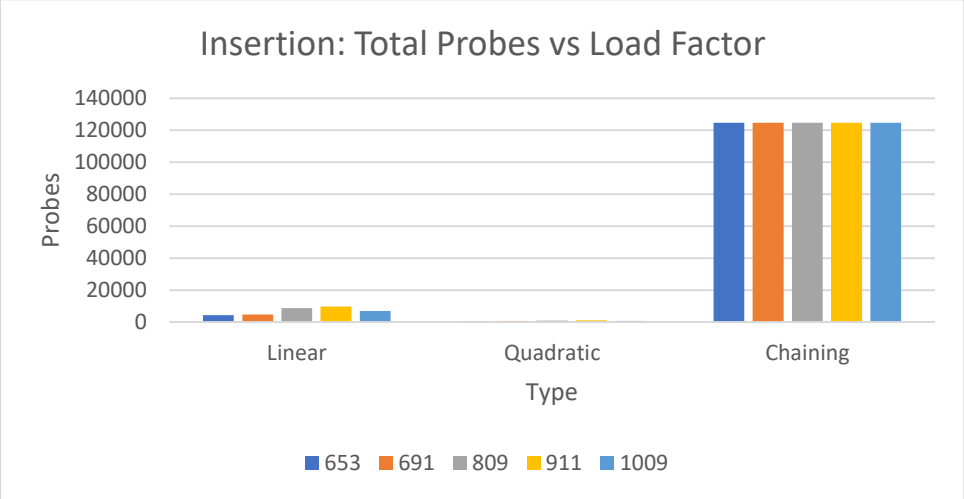
The goal of the experiment was to compare the load factor and probe numbers for the hash table between linear probing, quadratic probing and chaining.

To execute the experiment, I created two classes. Both are very similar to the HashMain class which implements the hash table. The first is the HashInsertTest class. This class outputs the loadfactor and number of probes required for making a hashtable of a given size. The second is the HashSearchTest This class outputs the loadfactor, total probes, average probes for searching for a given data subset of keys of size k in the hash table.

I then ran each of these for the test value set (653,691,809,911,1009) while redirecting output from the terminal to text files.

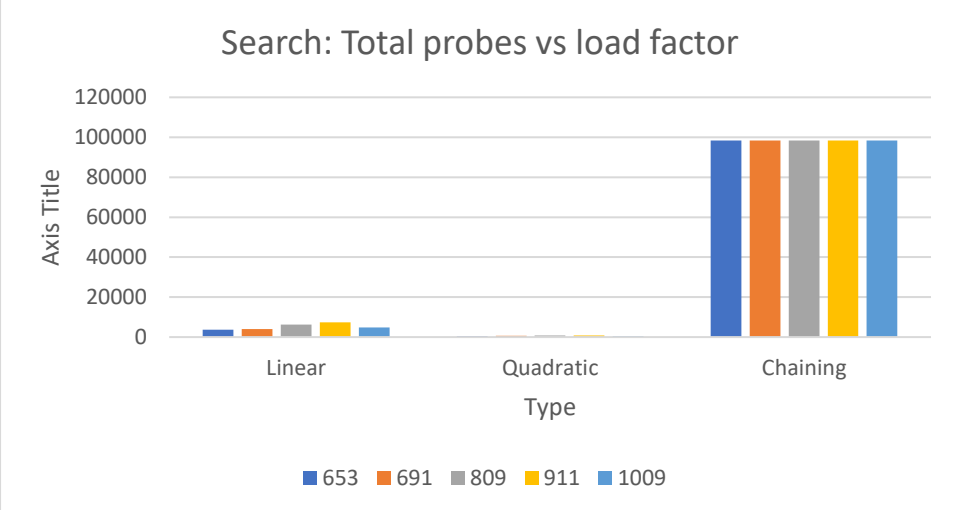
I copied from these text files to excels file to prepare my experiment data to be graphed

Insertion results

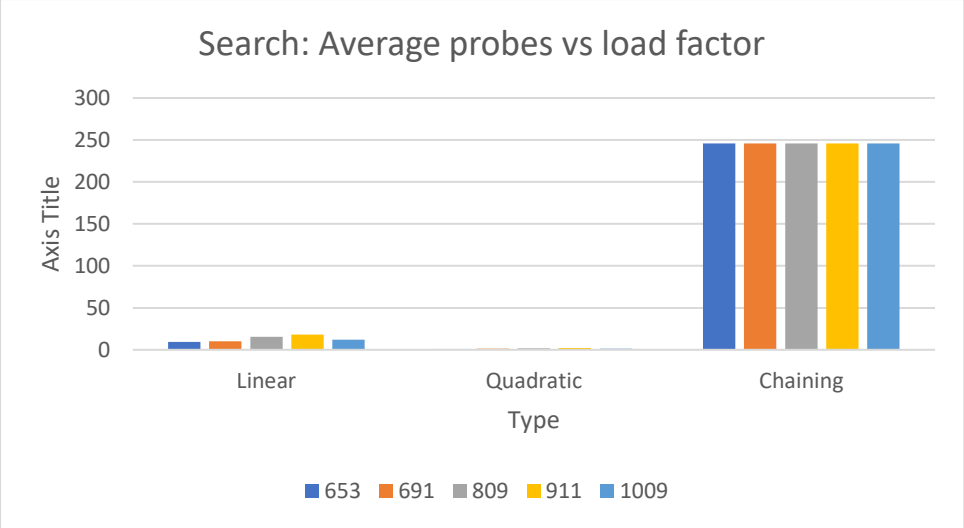


Quadratic probing requires the fewest insertion probes for every load size. Linear requires substantially more than quadratic. Chaining is constant at a extremely large amount

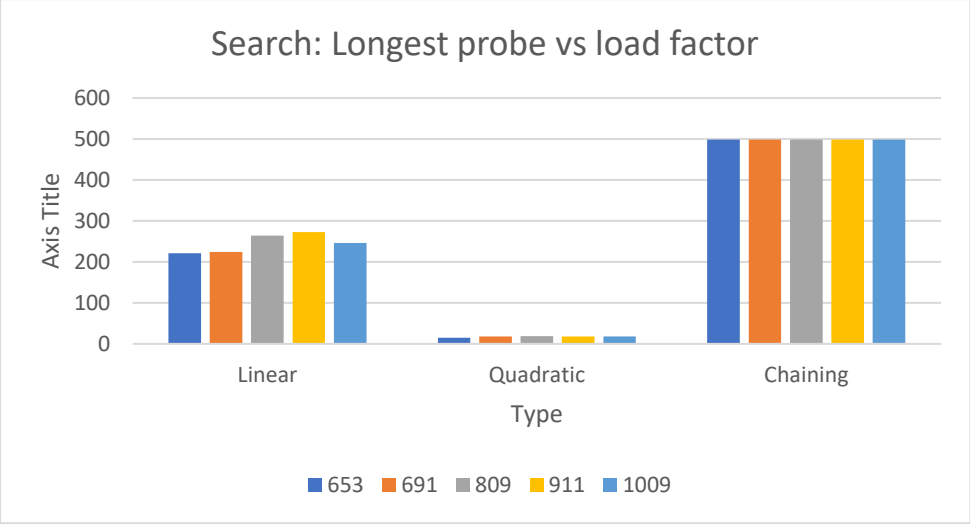
Searching results



Quadratic probing requires the fewest search probes for every load size. Linear requires substantially more than quadratic. Chaining is constant at a extremely large amount.



Quadratic probing requires the fewest probes for every load size. Linear requires substantially more than quadratic. Chaining is constant at a extremely large amount.



Quadratic probing requires the fewest probes for every load size. Linear requires much more than quadratic. Chaining is constant at more than double the amount that linear.

Conclusion

Given that the table size is always fairly larger than the number of items, quadratic will always be fastest. Linear will generally be slower than quadratic either way.

Chaining needs a small table size to present an advantage over the other two methods.

Creativity

1. I have made and included a perfect hash function in the HashTable class which will run the given dataset without any collisions (to be substituted into the relevant insert and search methods).

Git usage log

1: commit 78d1c5c46d9e0e27b9211cc555520d5f45d821bb

2: Author: Mahmood-Ali Parker <ali@HP.localdomain>

3: Date: Tue Apr 2 21:52:59 2019 +0200

4:

5: CHAINING IS FINALLY WORKING!

6:

7: commit 449093b1a8423fdabbde4ded8333ecb0c3bd477e

8: Author: Mahmood-Ali Parker <ali@HP.localdomain>

9: Date: Tue Apr 2 20:55:06 2019 +0200

10:

...

26: Author: Mahmood-Ali Parker <ali@HP.localdomain>

27: Date: Wed Mar 27 11:28:18 2019 +0200

28:

29: untested finished set of classes. testing not been done yet.

30:

31: commit 9e7a639a6d4e3b31a482a69d365ff60bb4c69b30

32: Author: Mahmood-Ali Parker <ali@HP.localdomain>

33: Date: Tue Mar 26 19:21:41 2019 +0200

34:

35: Created HashMain class with main method, created HashTable for hashtable methods, created HashBank for abitrary methods, created Node for data object