Hashing Analysis Experiment

By Karan Aulakh

Introduction

This report will discuss the likelihood of collisions in hash maps based off three factors: table size, how the key was generated and collision resolutions. For my experiment the table sizes compared were 10, 25 and 100. The two methods of generating indexes were key modulo table size and mid square. The two resolutions methods were separate chaining and open addressing. The language used to create and implement this experiment was C++ and all the keys were randomly generated between 0 and 3 times the table size for each trial. The exact implementation of my methods can be seen in the attached source code file where they are explained in greater detail.

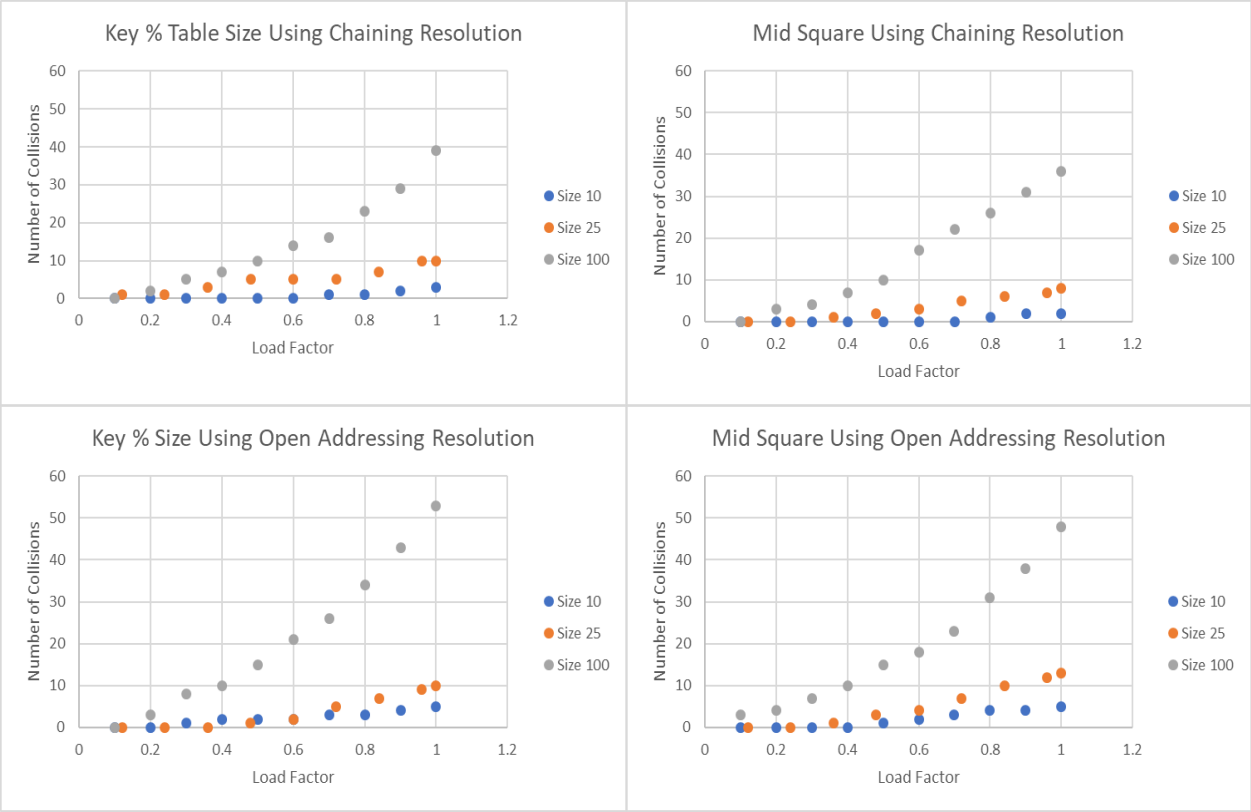
Implementation of Hash Tables

For this experiment, I created two separate classes to implement my hash tables. The first class "hash_chaining" is used to create all six hash maps that used separate chaining as it's collision resolution method. This hash map was created by creating an array of nodes that all have a value and a next pointer, essentially creating an array of linked lists where the array contained the head node. A value of -1 was used to represent an empty node and a value of 1 was used to represent a filled node.

My second class, called "hash_open_addressing" was used to implement all hash tables that used open addressing as their collision resolution method. This class is much simpler to understand since it is no more than an integer array, created dynamically. Once initialized the class saves each integer as a value of -1 to represent an empty index and uses 1 to represent a filled index.

Both classes use counters to keep track of how many collisions occurred, how many elements were added and the load factor, which is calculated every time an element is added. For my experiment collisions were incremented only once when the original desired index was full. They were not incremented multiple times in the case that the next possible insertion point was also filled. For example, under chaining, if the desired index is already filled it will only increment collisions once regardless of how many nodes are in that given chain and for open addressing it would increment collisions once regardless of how many spots it would check before finding the next available opening.

Results



As observed above, some of the more obvious conclusions are that as the table size increases the number of collisions increases. This makes perfect sense because as the table size is increasing the number of keys added to the element is also increasing, yielding more collisions.

The percentage of collisions over table size for all cases is between 20% and 53%. Narrowing the results to only tables of size one hundred, makes the data more accurate because it further randomizes the possibilities and narrows the range of collisions to between 36% and 53%. This data supports the claim that the number of collisions is generally close to or below 50% of the table size. Still considering tables of size 100, but of load factor 0.75 we can observe that all four experiments have a percentage of collisions over table size of lower than 30%. Proving why it is very beneficial to resize hash tables once they've reached a load factor of 0.75 to minimize collisions.

Discussion

When observing the percentage of collisions over table size there is no significant pattern in any of the four experiments as the table size increases. This is because all the experiments should be similar regardless of table size, except the larger the table size, the more accurate the data should be. This is like how if you flip a coin a few times the probability could even be 0% or 100%, however the more coin flips you perform, theoretically the closer it should get to a probability of 50%. The only pattern between collisions and table sizes is that the larger the table size the more overall collisions there will be. This is, however the obvious outcome since the larger the table size is the more elements are added into the hash map.

Collisions are less frequent when using the chaining resolution method as opposed to using open addressing. Which is a very logical outcome. This is because every time a collision occurs within chaining the number of filled indexes does not increase, the chain is added to the specific index that was desired in the first place. In open addressing, empty indexes go down by one on every addition, even additions that result in collisions. This results in a hash table that has one fewer open indexes, further increasing the chances of collisions. For example, consider a hash table with a total size of 50 that has 40 elements in it and 10 collisions. If this table used separate chaining it would have 30 of its 50 nodes filled up and the resulting elements would be chained onto those 30 nodes. This means the next element has a 30/50 chance or a 60% chance of colliding. On the other hand, if this scenario were to be performed on a hash table using open addressing, the table will have 40 nodes filled and only 10 free, yielding an 80% chance of collisions.

Maybe the least obvious observation was that collisions are higher using key modulo table size to determine the indexes as opposed to using mid square. This may be the case because using the mid square method randomizes the numbers more than key modulo table size does. Although, the fact that C++ random generator doesn't "truly" generate random numbers might have influenced this experiment. Nevertheless, mid square randomizes the key to a further extent because it performs even more operations on the seed, which means even numbers that started of similar would end up with a completely different key. For example, given 2 and 12 within a table size of 10, the mid square key will result in 4 and 6 whereas the key modulo table size would result in both keys being a value of 2.