Hashing Analysis

Hash table is a data structure that maps the keys to values through a well-designed hash function. A hash function computes an index into an array of buckets so that the desired values can be found efficiently. The basic functions such as insert, search, and delete are done in constant time (O(1)).

Two keys mapping to the same location in the hash table is called Collision. Number of collisions can be decreased by implementing a good hash function and resolution schema. In the project, we choose to implement MidSquare, and Key mod Table size hash functions.

In the project, we are given two resolution schemas: separate chaining, and open addressing.

1. Chaining Resolution:

In separate chaining, I use an array of linked list and keep track of the load factor. Load factor is total number of keys divided by size of the table and can be above one due to the fact that I am using linked lists in addition to the hash table. Load factor is recalculated after every insertion regardless of the number of collisions. All keys that map to the same hash value are kept in a list of bucket. The number of collisions in chaining resolution schema is equal to the number of items in the hash table minus the number of filled buckets. One of the important advantages of chaining resolution is that one can always add more elements to it, however, usage of extra space should be kept in mind.

1. Open addressing:

There are many different ways of implementing the open addressing resolution schema but I chose to implement the linear probing. When the hash function causes a collision by mapping to a new key to a cell of hash table that is already occupied by another key, my implementation of linear probing looks for the closest following free cell and inserts the new key. This traversal is known as linear probing since it goes by one element at a time. Load factor is recalculated after every insertion. The way that my implementation counts the number of collisions as follows: whenever the hash function generates an index value that is already filled, count that as one collision and search for another available index even if the next searched index is full, the number of collisions is still one. The process finishes once the key inserted in an available spot. The schema ends when the hash table size is full. Thus, my implementation does not resize the hash table.

Discussion of Plot

The plot above shows the Number of collisions vs load factor for separate chaining with hash function key mod table size. Note that the number of collisions increases as the load factor increases. The reason for that is as we add more random key to the hash table the probability of getting same index through the hash function increases and therefore the probability of the having a collision increases. Also note that adding keys increase the load factor. That is, it can be concluded that as the load factor increases the number of collision increases. Now note that the number of collisions for larger tables are higher than the smaller tables. The reasoning behind that is as we have a larger hash table, we are more likely to have collisions due to the principles of direct proportion. There are small exceptions as can be seen from the plot above. For example, focus on the table size 30 and the number of collisions for load factor between 0.4 and 0.6. The number of collisions are the same even though the load factors are different. The reason for that is key values are randomly generated and it might become possible that in some cases even though the number of keys are smaller, the key values can be mapped to the same index through a hash function more times. In fact, focus on the area where the load factor is between 0.2 and 0.4 for a table size 30. The number of collisions are higher for a hash table which has less keys in it. This is due to the randomness of keys and the implementation of the hash function. The discussed points here apply to other graphs as well.

Effect of Collision resolution scheme on collisions versus load factor

Separate Chaining with key mod table size vs Open Addressing with key mod table size

Regardless of the table size, separate chaining gives less number of collisions when the table is close to be full. The reasoning behind that is the usage of linked lists. Since we insert the same mapped index into a bucket when we use separate chaining, it reduces the total number of collisions. In cases that we have a high load factor (> 0.7), due to collisions happening more often, open addressing will check more hash table index before it finds an available spot to insert. Hence, chaining will become more efficient once the load factor is higher than 0.6 – 0.7. Rehashing should be done to resize the hash table and keep the load factor lower than 0.5 if linear probing is to be used. However, since I did not resize my hash table, linear probing performs worse compared to separate chaining.

Separate Chaining with MidSquare vs Open Addressing with key mod table size

Regardless of the hash function, the plots suggest that as the load factor gets higher than (0.5-0.6), the open addressing (linear probing) resolution method tends to do worse compared to separate chaining resolution method. It should also be noted that separate chaining results in less collisions when the load factor is slightly less than 0.4 which suggests that separate chaining is a bit better than open addressing in terms of number of collisions.

Discuss effect of hash function on collisions versus load factor

Based on the plots above, it can be shown that key mod table size hash function slightly gives a less number of collisions regardless of the table difference in table sizes. However, it can be due to the randomness of the key values. There is no strong evidence to show that one hash function comparatively is better than the other one. They both indicate similar results in number of collisions. There seems to be no correlation between the load factor and hash function, at least, for the three table sized that are used in the experiment. In terms of efficiency, both functions perform in constant time. Based on the evidence that is gathered from this experiment, both functions show the same performance.

Again, the number of collisions are not significantly different and load factors seems to be consistent. There is a slight difference between them but the difference may be caused by the randomness of the key values or the table sizes. Both hash function perform about the same.