Broday Walker
Hashing Experiment
March 24, 2020

I. Introduction

A. Hashing is a technique for inserting, deleting, and finding values in a data structure in an expected time of O(1). Hash tables are typically implemented as an array of size *m*, with *n* keys inserted into the array. Insertions into the hash table are accomplished by mapping the key value to some location in the table using a hash function. The hashing experiment presented in this report is implemented as a dynamically-allocated array of size 311.

B. A simple hashing function h(key) = key % *m* was used in the experiment. This function is responsible for locating the slot in the table in which the key is to be inserted. In reality, any arbitrary hash function that maps the key to a location in the table could be used. Ideally, the hash function selected for use in a hash table has the characteristic that all keys hashed with that function are equally likely to be mapped to any of the locations in the table. Even if this condition is satisfied, there is no guarantee the keys will be evenly distributed across the table.

C. Due to finite memory resources available to hold hash tables and the imperfect nature of hashing functions, hash tables must be equipped with a strategy or course of action to be taken when collisions occur. Collisions arise when more than one key hashes to the same table location. The strategy used to determine a solution for this conundrum is referred to as the collision resolution policy. Two widely-studied collision resolution policies include linear probing and double hashing. The linear probing collision resolution policy is simple to understand and implement. When a key hashes to an already occupied location in the hash table, the location is incremented by some number (typically 1), and then inserted into the new location if it is open. The cycle continues until an open location is found and the key is successfully inserted. Double hashing applies another hash function or operation to the key when the original location to which the key hashed is already occupied. This second hash must never evaluate to 0 and it should be able to probe all cells in the table. Both collision resolution policies were tested with two different load factors.

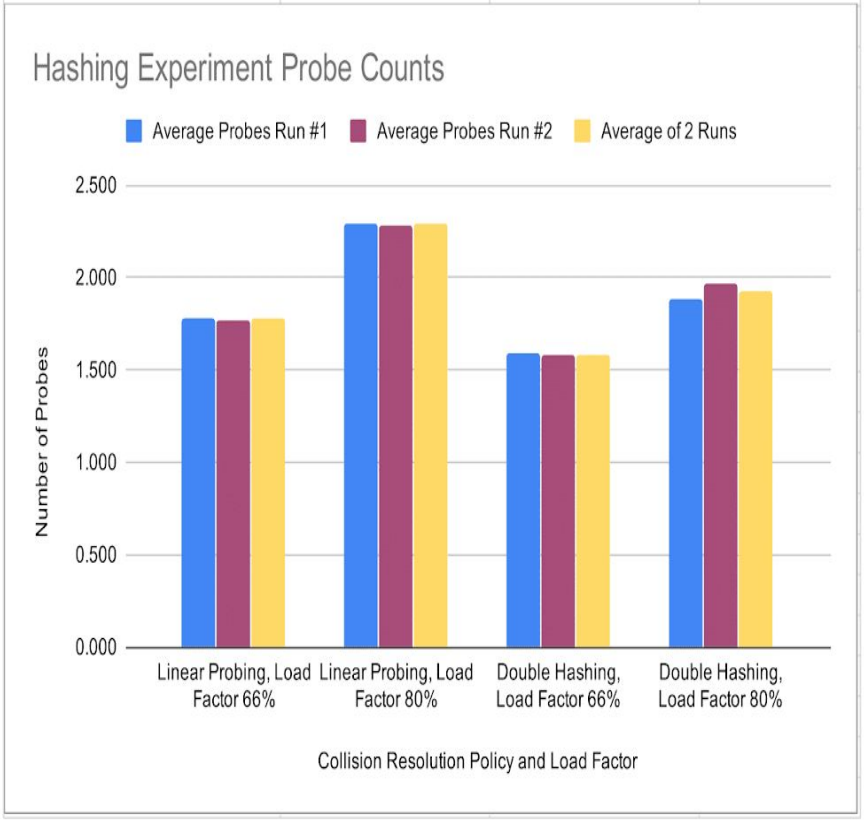
D. The load factor of a hash table can be found with a simple calculation: divide the number of keys in the table by the size of the table. This calculation provides a measure of how full a hash table is. In this experiment, load factors of 66% and 80% were tested. The load factor is important to track and understand because as the load factor rises, the likelihood that an insertion will result in a collision rises as well.

E. The function rand() was used to generate pseudo-random numbers between 0 and INT_MAX. A modulus 5000 operation was performed on the result, guaranteeing numbers between 0 to 4999. The random number generator was seeded to time(NULL). To ensure unique pseudo-random numbers were generated, all numbers were inserted into an unordered set data structure.

II. Actual Results

A.

Description	Average Probes Run #1	Average Probes Run #2	Average of 2 Runs
Linear Probing, Load Factor 66%	1.776	1.771	1.774
Linear Probing, Load Factor 80%	2.292	2.280	2.286
Double Hashing, Load Factor 66%	**1.585**	**1.580**	**1.583**
Double Hashing, Load Factor 80%	1.880	1.968	1.924



B. The use of double hashing as a collision resolution policy coupled with a hash table load factor of 66% resulted in the lowest average number of probes. The highest average number of hashes belonged to the table using linear probing as a collision resolution policy in a hash table with a load factor of 80%. Double hashing resulted in a lower average number of probes per key insertion for both table load factors when compared to linear probing. Each collision resolution policy and load factor resulted in a similar average number of probes between the two experiment runs.

The performance difference between linear probing and double hashing is significant. The higher average number of probes in linear probing can be attributed primarily to an effect known as primary clustering. As collisions occur in a hash table using linear probing, more collisions are likely to occur in the future as this form of collision resolution typically results in contiguous blocks of keys which are placed in locations they did not originally hash to. The benefit of double hashing is its ability to reduce the likelihood of the formation of a contiguous block of keys by moving one or more locations away from the initial collision. Once this "hop" is made, the key is inserted.

III. Conclusions

In conclusion, many factors, including collision resolution policy and table load factor, play an important role in the number of probes required to perform the operations of inserting, searching, or deleting in an open-addressed hash table. A finely-tuned hash table that has a low average number of probes is the result of many choices. The chosen hash function, collision resolution policy, load factor, the distribution from which the keys are drawn, and the table size are all key considerations in building a table. For the chosen collision resolution policies and load factors, the experiments ultimately showed that a higher load factor leads to more collisions while linear probing leads to higher clustering, and as a result, a higher average probe count.