1. Introduction

**Hashing** – Of all available structures for data storage, hashing proves as one of the fastest. Easily implemented and treated as an array / list, a hash table allows programmers the basic operations of insertion, deletion, and searching in, on average, near-constant time. The ideal hash table, is of prime-number size, maps all data values to unique spots (i.e. no collisions), offers constant time insertions, deletions, and searches, and lastly is no more than 80% full. Now, a hash table can be broken down into these parts: the hash function, and the collision resolution policy. Data is mapped to a specific spot or location in the table with a hash function, a mathematical function of the data itself. For example, hash\_spot = data\_value (mod table\_size) would be a very simplistic hash function. To be a bit more specific, the term “data\_value” will be replaced with “key”. The perfect hash function will map every key to a unique spot in the hash table, however, this is nearly impossible. Some keys will map to the same location, and a hash table’s collision resolution policy (CRP) established collision handling, either by chaining or open addressing.

**Program hash function –** In our program, we use the easy hash function,

where k is the key

m is the table size.

By modding the key by the table size, we produce the uniform hashing scheme, which means all spots in the table are equally likely to be hashed to by any given set of keys.

**Program CRPs** – Our program implements a hash table as a dynamically-allocated array in C++11. The table’s size can be determined by the programmer individually, however, we chose 311, a prime number. We pit two CRPs against each other, linear probing and double hashing.

* 1. Linear probing – a form of open addressing, linear probing has a probe sequence increment of one. This means if a new key collides with an old key, the adjacent spot following the collision spot is checked. If no key resides in that spot, the new key is placed there. However, if that spot is also occupied, the following spots are checked one by one until an empty spot is found or the entire hash table has been checked / probed. The equation is:

where h’(k) is the initial spot to which a key is mapped,

i is the increment,

m is the table size.

Linear probing, although simple to implement, promotes primary clustering. This occurs because placing keys in adjacent spots leads to long “lines” of data collecting in one section in the table. This leads to longer search times, in the end, so more complicated probing techniques are preferred.

* 1. Double hashing – one of the best methods for open addressing. Instead of checking the next spot in the table, the key is placed into another hash function. The equation is

where h1(k) is the first hash function,

h2(k) is the second hash function,

i is the increment,

m is the table size.

Double hashing acts like a random spot selector, which is why it’s one of the best CRP methods. Now, the second hash function needs to be relatively prime to the table size for the whole table to be searched (uniform hashing).

**Program load factors** **–** we employ two load factors in our experiment, 0.66 and 0.8. In brief, this means we will stop inserting keys into the hash table once they are 66% and 80% full, respectively. For each key insert, we will count the number of probes – or number of spots checked if empty. For the tables with load factors of 0.66, we can expect the average number of probes to be smaller than tables with load factors of 0.8, given the latter will be more full.

**Program random number generation** – For our experiment, 1000 keys will be generated randomly from the set of all positive integers between zero and 10,000, and dumped into a file. The file will be checked for duplicates, all of which will be removed. Regardless, the set range should be large enough to prevent duplicates.