Hashing Implementation and Analysis

Ben Massik
Tim Bauer

Purpose:

The purpose of this project is to evaluate the performance characteristics of Hash Tables based on how collisions are avoided. The specific characteristics are inserting, searching, and deleting in the Hash Table, and the specific collision resolution methods are linear probing, chaining with a linked list, chaining with a binary search tree, and cuckoo hashing.

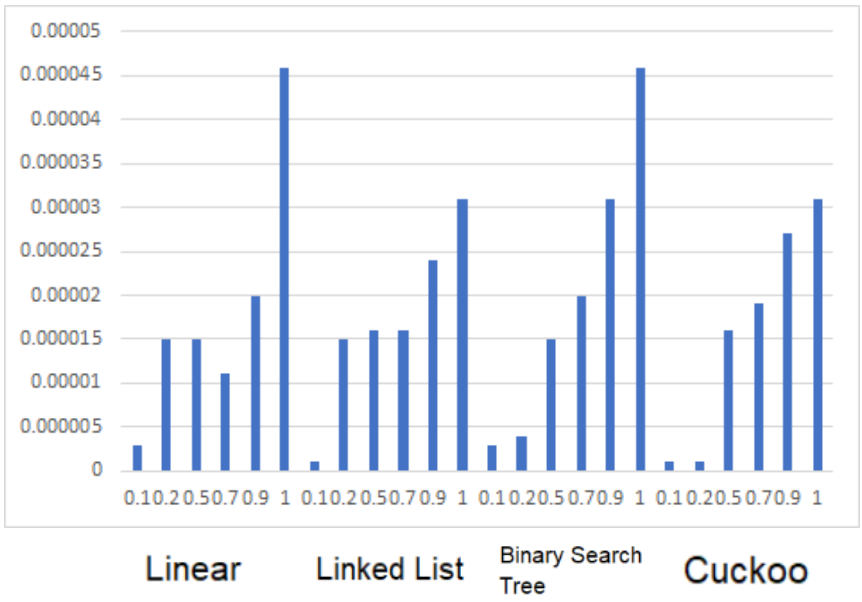
Procedure:

The data structures that were implemented are Hash Tables, Binary Search Trees, and Linked lists. The hash table is implemented in four different ways, one for each collision resolution. The table size for the Hash table is 10009 because prime numbers are used to make the size to avoid overlap in the functions. Data is assigned to the hash tables using one or both of the hash functions: index = key% TABLE_SIZE and index = Floor(key/TABLE_SIZE)%TABLE_SIZE.

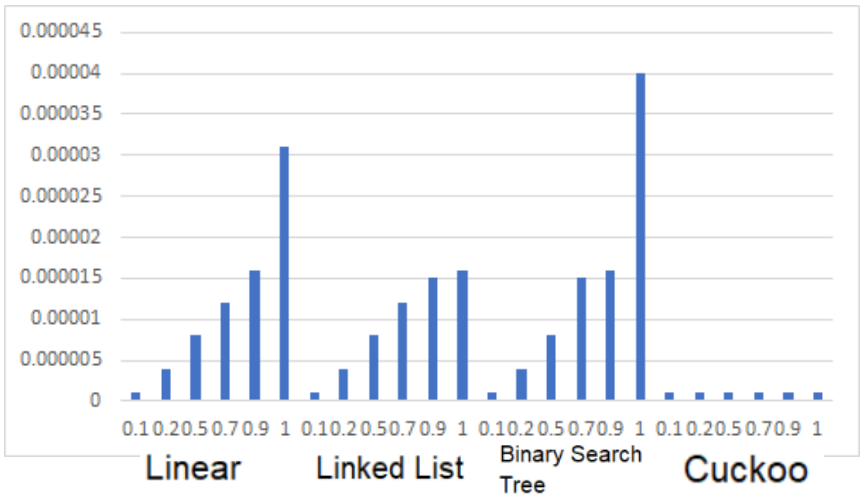
If a slot is full in the hash table, there are four different ways to store the data in a way that it is still retrievable depending on the collision method. The first method is linear probing, meaning that probe for the next available spot in the table. This can be inefficient at higher load factors for all commands once data is not at its index. The next way to avoid collision is chaining, which is where the linked lists and binary search trees are used. If data is already at the index, then it will be added to either the linked list or the Binary search tree. This method makes the data more organized, but can still be inefficient at higher load factors. The last method is Cuckoo hashing, which involves reorganizing two hash tables, each with their own hash functions, to find open spaces for the data by looking at both tables and indexes. This method has the best lookup time with a worst case complexity of O(1), but insertion can be time consuming and the data can get caught in a loop. The metric that we used to evaluate the most efficient method was running insertions, deletions, and searches on the different tables, and timing how long they took, on average, to run.

Data:

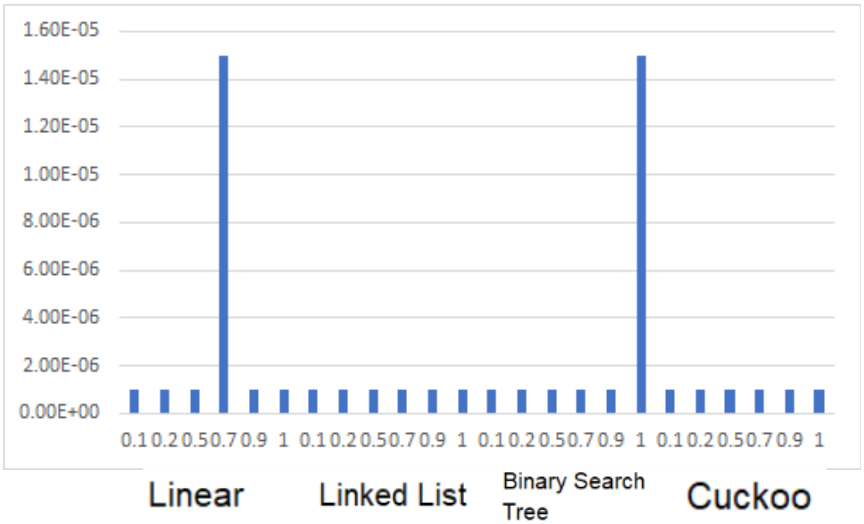
Insertion



Deletion



Searching



Results:

The program was too fast for just 100 instances, so it was run 1000 times. The results performed as expected. Cuckoo performed the best in all categories. It had the fastest average run time for every load factor on insert, delete, and search. Besides Cuckoo Hashing being the fastest, each function performed roughly the same in each category. It performs at roughly the same speed because there were not enough collisions and because computers are fast enough now. In order to be able to produce any noticeable differences, there would have to be a lot more tests performed.