Comparing Collision-Resolution Techniques for Hash Tables

Sam Schwartz
sschwartz222@uchicago.edu
CS152 Spring 2018

Abstract:

Hash tables are a data structure designed to be accessed in approximately constant time. However, this time is not a guarantee, and the collision-resolution technique is critical in influencing this time.

In this study, we evaluate three collision-resolution techniques: linear probing, quadratic probing, and linked list collision resolution. Our results are: The average number of operations performed with linear probing is the greatest at all hash table sizes for all trials performed except one (Vocab3.txt, size 10000). The average number of operations performed with quadratic probing is significantly fewer than the number performed with linear probing at smaller hash table sizes while about equal at larger sizes. The average number of operations performed by linked list collision resolution is the least in all cases tested.

Background and Motivation:

Hash Tables:

Hash tables are a type of data structure that store key-value pairs in buckets in an array. Hash tables use a hash function to compute the appropriate index of the array associated with a particular key. However, a problem may arise during the process of hashing known as a collision—where the bucket of the computed index is already occupied. In such a case, a collision-resolution technique must be used to determine the appropriate bucket. The advantage of a hash table lies in its efficiency independent of the number of elements stored in the table—in well-designed hash tables, the average time complexity of "search," "insert," and "delete" functions are O(1).

Collision-Resolution Techniques

As hash collisions are all but unavoidable when working with large datasets, functionalities must be implemented to handle such occurrences. We are testing three different collision-resolution techniques: linear probing, quadratic probing, and linked-list chaining.

Linear probing: All entries are stored in the buckets of the array themselves, but when an insertion is attempted and a bucket of a particular index is already occupied, the index is incremented by one (wrapping around the beginning until reaching the original index) until an empty bucket is found. If no empty bucket is found, then the hash table is full and the insertion fails.

Quadratic probing: Like linear probing, all entries are stored in the buckets of the array themselves. When an insertion is attempted and a bucket of a particular index is already occupied, the next index checked is given by ((h + j^2) % table_size), where j = 0 for the original index and is incremented by 1 for every check thereafter, until j = table_size. Quadratic probing has the advantage of spacing out colliding entries and preventing clusters, but has the disadvantage of not necessarily checking every bucket (quadratic probing can fail even if the table is not full).

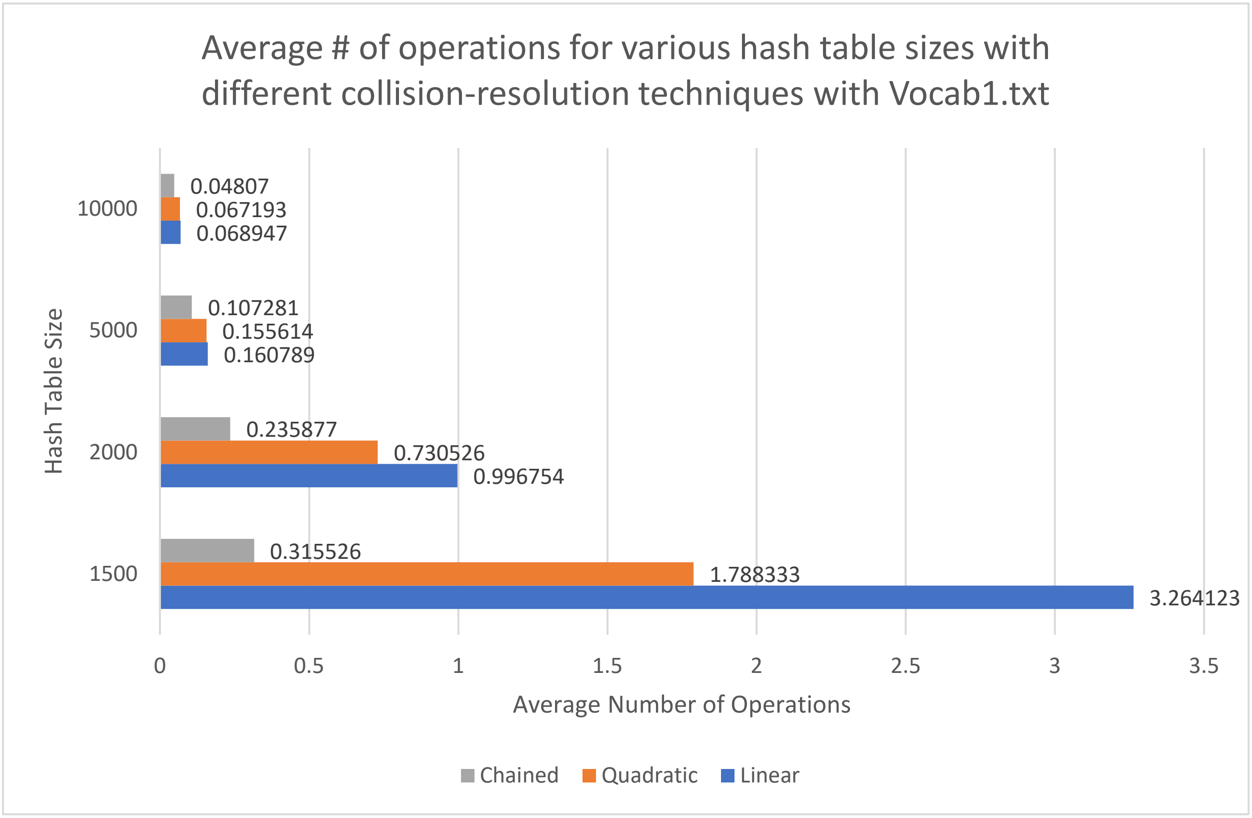
Linked-list chaining: When a collision occurs, a separate index is not computed. Rather, each bucket consists of a linked list containing all entries associated with the particular index. Insertion into a table with linked-list chaining will never fail if no limit is placed on the linked list size, but entries are prone to clustering if a particular index is calculated more often than others.

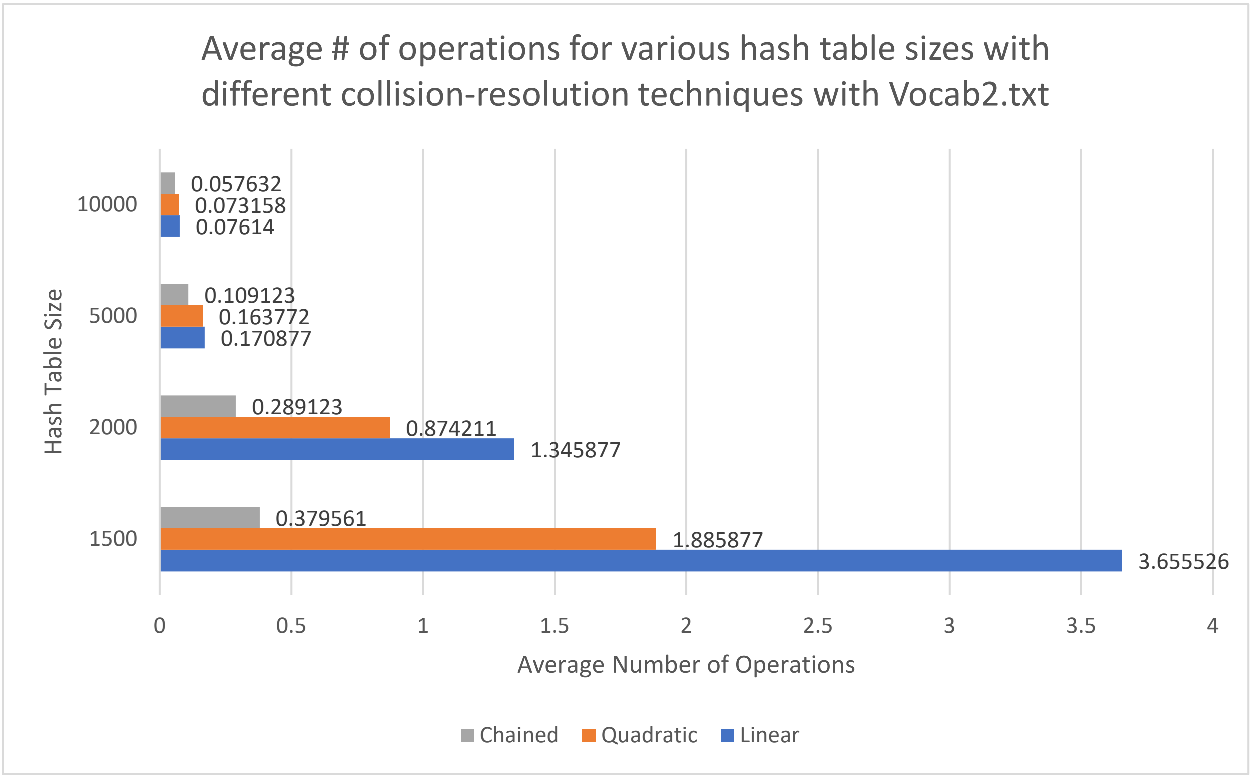
Importance:
Given the importance of hash tables as highly efficient lookup structures used in various kinds of computer software, caches, etc., it is in our best interest to seek the most efficient collision-resolution technique to preserve the efficiency of the structure and prevent them from becoming useless in the wake of poorly-resolved collisions.

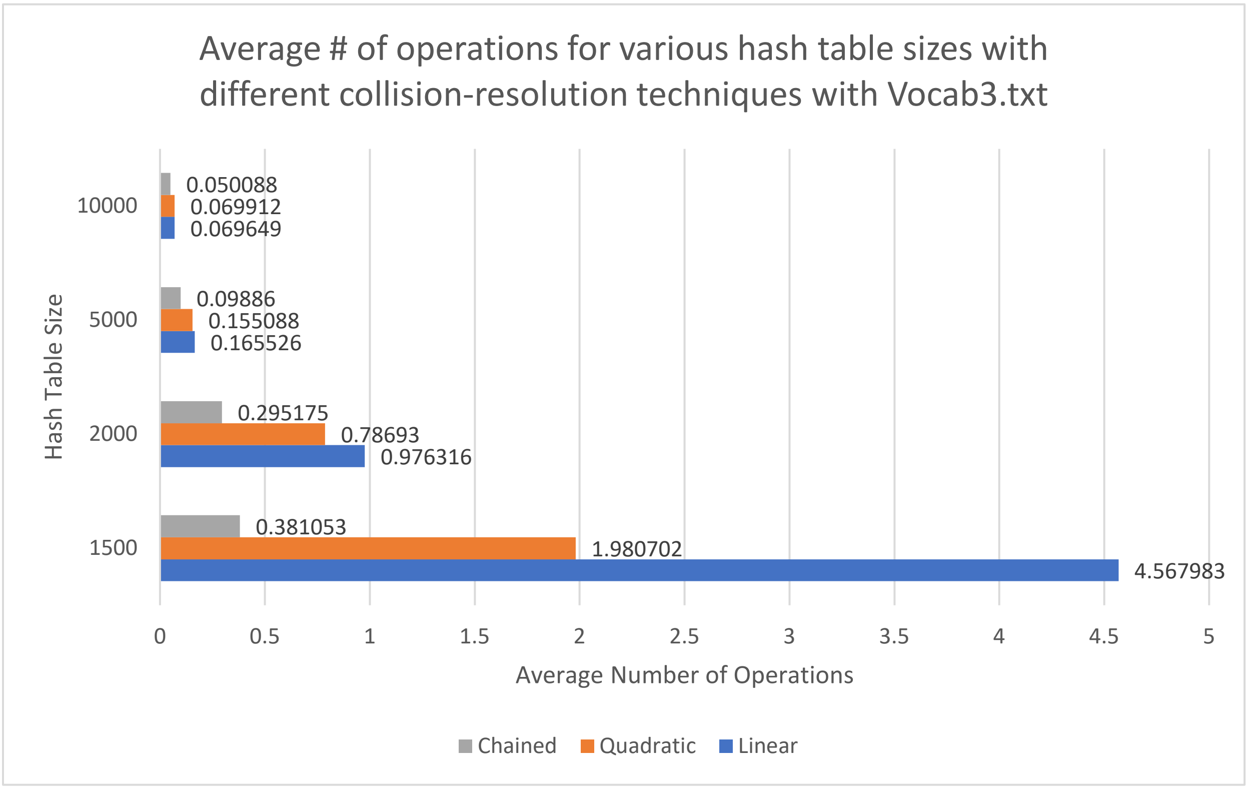
Methods:

A series of inserts and searches were attempted with provided test files “Vocab1.txt,” “Vocab2.txt,” and “Vocab3.txt.” Each time a collision occurred, an int pointer (initialized to 0) “opcount” was incremented by 1. Opcount was recorded for each operation and the average opcount per operation was computed for each type of collision-resolution technique at four different hash table sizes (1500, 2000, 5000, 10000) for all three test files.

Results:


Average # of operations for various hash table sizes with different collision-resolution techniques with Vocab1.txt
Hash Table Size
10000
5000
2000
1500
0.04807
0.067193
0.068947
0.107281
0.155614
0.160789
0.235877
0.730526
0.996754
0.315526
1.788333
3.264123
0
0.5
1
1.5
2
2.5
3
3.5
Average Number of Operations
Chained
Quadratic
Linear


Average # of operations for various hash table sizes with different collision-resolution techniques with Vocab2.txt
Hash Table Size
10000
5000
2000
1500
0.057632
0.073158
0.07614
0.109123
0.163772
0.170877
0.289123
0.874211
1.345877
0.379561
1.885877
3.655526
0
0.5
1
1.5
2
2.5
3
3.5
4
Average Number of Operations
Chained
Quadratic
Linear



Discussion:

Linear probing performed the worst in all test cases except (Vocab3.txt, size 10000), and performed especially poorly at smaller table sizes. This performance discrepancy at smaller table sizes may arise from the presence of more entries than number of buckets, as a failing search or insert necessarily checks each bucket in the table first before determining failure. Although the same concept applies to quadratic probing (as j is incremented from 0 to table_size), entries tend to be less clustered after insertions, and therefore quicker to probe/tend to have fewer further collisions, an aspect that contributes to quadratic probing's significantly more efficient performance at smaller table sizes.

Part of chaining's efficiency lies in its constant insert efficiency—by inserting to the head of each linked list, the insert operation never fails and is time-constant. This technique works well when entries are not clustered into a few, extremely large linked lists, which appears to be the case for all three of the tested files. However, the search function, which is reliant on the length of the linked list at a particular index owing to the looping needed to traverse a list, is on average O(n), with *n* being the number of items in the list.

In general, all three test files saw significantly decreased average numbers of operations at larger hash table sizes as the probability of collisions is much smaller (assuming no set deliberately created to cause collisions is used) when the number of available spots increases as the number of entries remains constant.

Conclusions:

Hash tables are a type of data structure that presents opportunities for highly efficient search, insert, and deletion operations. Maintaining high levels of efficiency is dependent on implementing effective collision-resolution techniques to resolve near-inevitable collisions of keys that map to the same index. Testing the techniques of linear probing, quadratic probing, and linked-list chaining, I found that chaining was the most efficient collision-resolution technique over all tested files and hash table sizes.