# Hash Tables - Hash Table Analysis

Timofey Maslyukov

University of Virginia, Charlottesville VA 22903, USA

gaf4sw@virginia.edu

**Abstract.** The purpose of this paper was to test and compare different collision resolution strategies on different grid sizes of characters. These two methods were conducted on 3 data grids of sizes 50x50, 140x70, and 250x250. The time for the insertion of a dictionary of words and the solution of the grid was recorded on different methods and the times for this process was recorded. A distinct advantage from the results was shown to be had by the linear probing resolution strategy as it had faster results. With the linear probing strategy an optimized method was created which decreased the number of runs of the program on unneeded characters. This optimization allowed for again smaller runtimes.

## 1 Introduction

### 1.1 Problem

The goal of this experiment was to analyze the runtimes of different collision resolution strategies as well as creating an optimized version of the most effective resolution strategy. The two collision resolution strategies used included the linear probing method as well as the Seperate Chaining method. The linear probing method would change the index of the collided data into a different index with a standardized hashing method. The other collision resolution strategy Seperate Chaining used bucket arrays which when some data had the same hash as a different data would be placed in a bucket array.

## 2. Method of analysis

### 2.2 Seperate Chaining Method

In order to measure the speed of each type of collision resolution strategy two different collision resolution strategies were used. The first was the Seperate Chaining Method. The method was used on 3 different grid types all with defining characteristics. The 50x50 grid with a small grid size, the 140x70 grid with uneven rows and columns and the 250x250 grid with a high amount of characters. The Seperate Chaninng Word Puzzle Method was performed on all 3 of these grids. The word puzzle method worked by using a scanner to import a dictionary of 25143 words into the Hash Table. The method would then use a scanner to import the grid desired for testing. The method would create a grid with columns and rows determined by the grid.txt file. Placing each character in the file into the grid would then create the grid suitable for testing. The method would then look at all the characters from size 3 to 22 in 8 cardinal directions and compare it to the hashtable. This method was performed 5 times for each grid. A timestamp was used at the beggining and end of the method to record the time it took for the program to complete. The number of collisions was also recorded in order to see how many times the collison would occur. This was done by setting up a counter for each time the data would be added into the non head of the bucket array. The average was computed based off of the division of number of trials and the sum of all the times. The standard deviation was computed based off a simple population standard deviation equation.

### 2.3 Linear Probing Method

The same method of testing was performed on the Linear Probing Method. 5 tests were performed on 3 different grid sizes using the Word Puzzle Solver. The collision strategy was implemented in a similar way with the count inceasing for every index shifted when a collision would occur. The method performed in the same manner and the same timing, avreging, and standardizing was performed to the data.

### 2.4 Optimized Linear Probing Method

After the results for the collision resolution strategies was performed the most efficient collision resolution strategy was chosen. An optimized version of the collision strategy was created. In this optimized version of the word puzzle finder method unneeded characters were not searched. This included the first and last two rows and columns since at those locations the north, east, south, and west were not always needed. This meant that a lot of unneeded characters were not searched saving vast amounts of time. Also due to the size of the dictionary of words being known the initial capacity was increased so the resize method was not used as much. With these two optimization the same exact testing method was performed. 5 tests on 3 grids using the Linear Probing resolution strategy was done. The method was perfomed in the same manner with the same timing, avreging, and standardizing as the other methods.

**Table 1.** This table shows the number of trials and grid type used on each method.

| Hash Table Type | HashTable (Seperate Chaining) | HashTable (Linear Probing) | HashTable (Optimization) | HashTable (Seperate Chaining) | HashTable (Linear Probing) | HashTable (Optimization) | HashTable (Seperate Chaining) | HashTable (Linear Probing) | HashTable (Optimization) |
|---|---|---|---|---|---|---|---|---|---|
| Number of Trials | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| Grid Type | 50x50 | 50x50 | 50x50 | 140x70 | 140x70 | 140x70 | 250x250 | 250x250 | 250x250 |

## 3 Results

### 3.1 Results of different methods

**The results of testing were able to show the meanigful difference between the collision strategies.**

The results were able to give meaningful data on the difference between different types of collision resolution strategies as well as different grid types and the optimization of the Word Searcher Method. The results showed that the Linear Probing Method was able to perform at a faster rate than the Seperate Chaning Method. This was at one exeption of the 140x70 grid though for this test the Linear Probing method produced a high standard of deviation. After this observation was made the optimized method was run using the Linear Probing Method. This produced the fastest results across all 3 grids. The

250x250 grid was able to produce the largest variance of results from previous methods especially with the optimized method. The time from the 50x50 to 250x250 method incereased drastically due to the high amount of characters introduced with the 250x250 grid. The collision of the linear probing strategy was lower than the Seperate Chaining method and the Optimized Method had the least out of all 3.

**Table 2.** This table shows the results of the HashTable Seperate Training Method on different grid sizes.

| | | Time (ms) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Trials | 1 | 2 | 3 | 4 | 5 | Average (ms) | Std. Dev (ms) | Collisions |
| HashTable (Seperate Chaining) | 50x50 | 741 | 568 | 613 | 463 | 643 | 605.6 | 91.14 | 4261 |
| | 140x70 | 1748 | 1819 | 2037 | 1907 | 1468 | 1795.8 | 190.17 | |
| | 250x250 | 7654 | 7912 | 7976 | 7576 | 7092 | 7642 | 313.51 | |

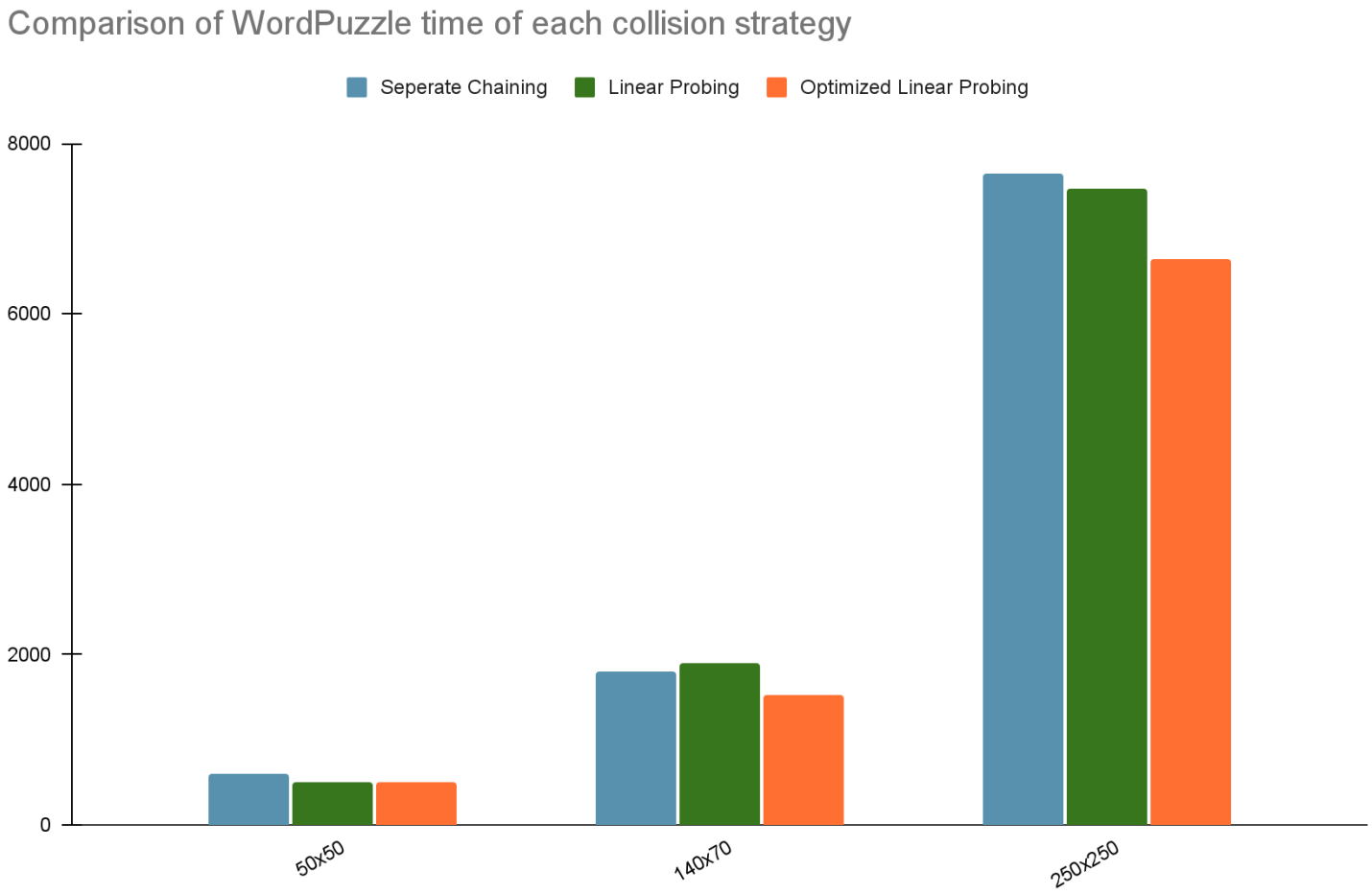
**Table 3.** This table shows the results of the HashTable Linear Probing Method on different grid sizes.

| | | Time (ms) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Trials | 1 | 2 | 3 | 4 | 5 | Average (ms) | Std. Dev (ms) | Collisions |
| HashTable (Linear Probing) | 50x50 | 630 | 466 | 503 | 486 | 458 | 508.6 | 62.70 | 2405 |
| | 140x70 | 1676 | 1767 | 1884 | 1437 | 2735 | 1899.8 | 442.65 | |
| | 250x250 | 7515 | 7624 | 7542 | 7436 | 7223 | 7468 | 136.43 | |

**Table 3.** This table shows the results of the Optimized HashTable Linear Probing Method on different grid sizes.

| | | Time (ms) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Trials | 1 | 2 | 3 | 4 | 5 | Average (ms) | Std. Dev (ms) | Collisions |
| HashTable (Linear Probing) Optimized | 50x50 | 448 | 474 | 573 | 491 | 484 | 494 | 42.11 | 2011 |
| | 140x70 | 1209 | 1806 | 1329 | 1455 | 1796 | 1519 | 243.06 | |
| | 250x250 | 6702 | 6537 | 6836 | 6773 | 6350 | 6639.6 | 175.87 | |

**Table 4.** This graph shows the average time for each of the methods compared to the grid size operated on.




## 4 Conclusion

### 4.1 Which method was better

After the experiment was concluded based on the results, a clear advatnage is shown is present for the linear probing method for dealing with collisions. It was faster in the 50x50 grid as well as the 250x250 grid. The one expection was the 140x70 grid. Removing the last test on the 140x70 grid would reveal faster average for the linear probing collision resolution strategy. As a result it can be concluded that the linear probing strategy is a more viable faster solution. Due to the small amount of collisions with the hash strategy the linear probing had smaller amount of collisions which resulted in a faster runtime. Also the bucketlist was not needed to be instanuated every hash table and meant that there was a lower runtime

for the linear probing method which does not need to instantiate bubble arrays. When instituting the optimized version of the linear probing method it was evident that the optimizations made were able to effectively decrease the run time of the program. When comparing to the proposed runtime the HashTables which were constructed alligned closely to the proposed runtime for the data which was being insorted. The reason why the linear probing method was effective was due to the smaller amount of collisions as well as the uneeded addition of bucket arrays which were not that useful for the hash function which was created.