Empirical Analysis of Open and Closed Hashing

Matthew Kay
Mkay0961@gmail.com
CSCI 230

Introduction

Hashing are very useful in coding, and especially helpful in dictionary's. They have an ADT that consists of search, insert, delete. Hashing works in a very unique way. Words are added to a hash. They are stored based on the values they are assigned from the hashing function. There are many different hashing functions and two types, open and closed. In this experiment, our goal is to determine which hashing is faster or more efficient.

I predict the search times for a closed hashing will not be as fast or efficient. The reason for this slower search time is because closed hashing, while searching, has to loop through all the words in the same key. This will take more time than if searching in a closed hash. Additionally, the results will also depend on the hashing functions. Some hash functions could group more words together in the same keys or could offset the words key, due to linear probing in closed hash.

Methods

To test my hypothesis for the time efficiency, I created two java classes called OpenHashing.java and ClosedHashing.java. These files implemented both the open and closed hash. Since the time efficiency is related to the hash functions and the hash values they produce, I used two hash functions for this experiment. Using two hash functions allowed me to compare the test time and efficiency's to determine which hash is better.

For the first hashing function I began by looping through the characters of the word getting each ASCII value for the letters. I then added each of these to a sum. Once I had the sum of all the ASCII values for each letter I then modded it by an arbitrary number. In this case, the arbitrary number chosen was 3. The number each word gave me became the key for that word. Below is the Java code for this first hash function along with it expressed as an equation.

```
//Hash Function 1
int key=0;
for(int i =0; i<word.length(); i++){
    char c = word.charAt(i);
    key += (int)c;
}
key = key%3;
return key;
```

$$h(k) = \left(\sum_{j=0}^{l} k_j\right) mod\ b$$

In this summation the l represents the length of the word. While kj represents the jth letter of the word. And b represents the arbitrary number.

For my second hashing function I did something similar to the first one. I started off looping through the characters of the word. This time at each letter in the word I received the ASCII value of that letter and modded that value by some arbitrary number. In this case, the number I chose was again 3. I then added each of the values to a sum. This sum I used as the key for that word. Below is the Java code for this second hash function along with it expressed as an equation.

```
//Hash Function 2
int key=0;
for(int i =0; i<word.length(); i++){
    char c = word.charAt(i);
    key += (int)c%3;
}
return key;
```

$$h(k) = \sum_{j=0}^{l} (k_j\ mod\ b)$$

In this summation the l represents the length of the word. While kj represents the jth letter of the word. And b represents the arbitrary number.

To test the search time of these two hashing function for open and closed hashing I created a java class called TestHasing.java. To start in this java file I took in a random text file I found online. Then I created methods that processed that text in the file. These methods stripped the text file off all unnecessary characters, data and repeated word. After processing the file, I was left with a string array of 546 words from the text file with no repeats. Next I took the first half in the string array and, in a loop, added them to both an open hash and closed hash for both functions. Next, in another loop, I would search both hashes for each word in the first half of the string array. Within my loop I would record the time each word took to search. This would represent the successful searches. To represent the unsuccessful searches, I would use the second half of the string array, the half that was not added to each hash. Doing this in a loop, I again recorded the search times for each word for both hashes. This gave me the data for my unsuccessful searches.

Results

After completing all tests, I had the necessary data needed to analyze the successful and unsuccessful search times for both open and closed hashing. In the end, I found that for function 1 closed hash resulted in a faster search time for both successful and unsuccessful. For function 2, open hashing had a faster search time for both successful and unsuccessful. Below is the evidence for these results.

Open Hashing

When creating the constructor for open hashing an array list of array lists must be created. When words are added to an open hashing there can be multiple words to the same key. We try to determine if this is more efficient than closed hashing. In open hashing to find a word, you have to find its key then iterate through the keys list of words to locate the one you are looking for. This could be very time consuming and take longer if there are a lot of words in the key.

By simply comparing the load factors of both functions 1 and 2 in table 1 we can tell that function 2 is more efficient; Even though neither function is fully efficient.

	Function 1	Function 2
Load Factor	182.0	39.0
Successful	92.0	20.5
Unsuccessful	182.0	39.0

Table 1. The calculated load factors, Successful, Unsuccessful for each function.

The average search time for open hash in function 1 was 11692.0127 nanoseconds successful and 5507.8315 unsuccessful. In function 2 the average search time was 844.9413 nanosecond successful and 1480.88645 nanosecond unsuccessful. So we know that function 2 had a much better search efficiency. We can clearly tell this from looking at the graphs. While comparing both functions for successful searches, you can see that in figure 1 most of the times are below 30000 nanoseconds. While in figure 2 they are all below 2000 nanoseconds. This difference is significant. Looking at unsuccessful searches, we see in figure 3 most times are below 5000 nanoseconds except for that one cluster. While in figure 4 all the times are below 2000 nanoseconds. These results make sense when looking at the table 1. We see that the average probes in function 2 for successful is about 70 probes lower than function 1. We can also see that the average probes in function 2 for unsuccessful is about 140 probes lower than function 1. We can tell that for open hashing function 2 was more efficient.

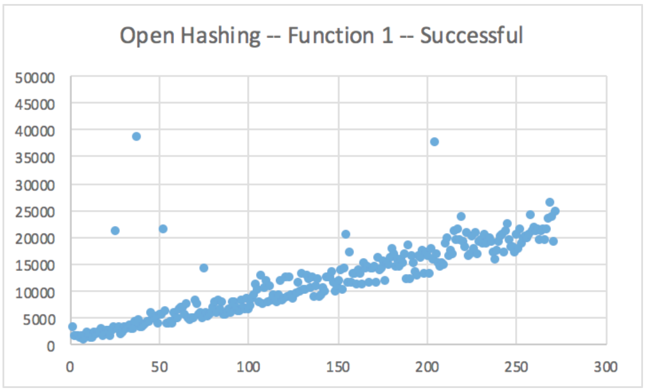

Figure 1.

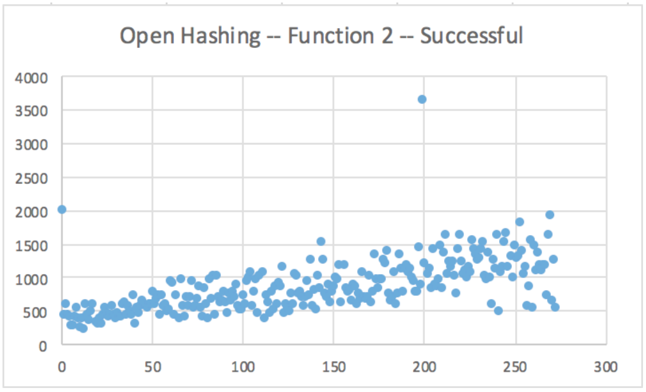

Figure 2.

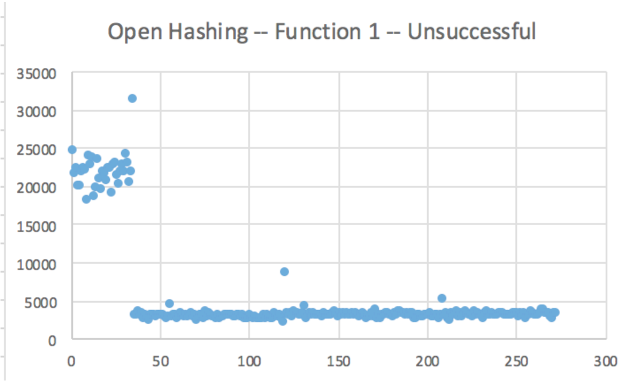

Figure 3.

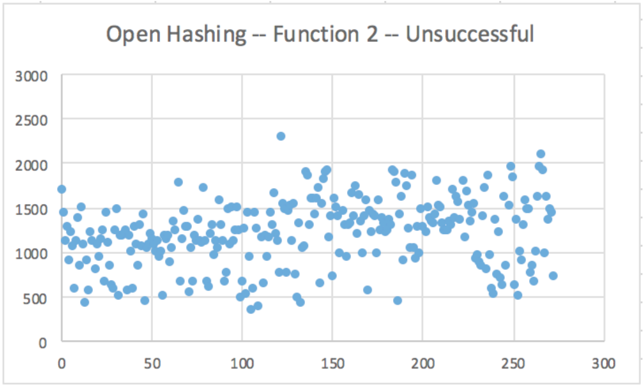

Figure 4.

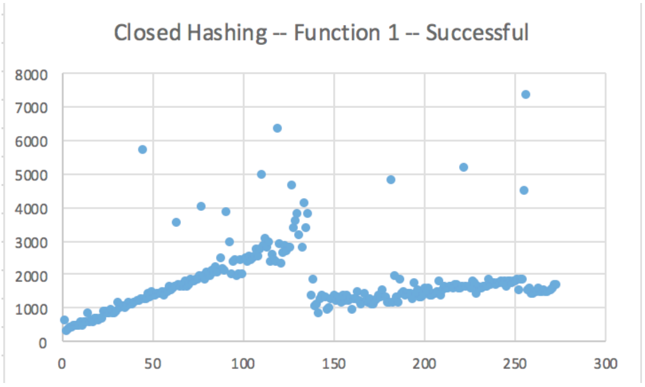

Figure 5.

Closed Hashing

When creating the constructor for open hashing an array list of words was created. When adding words to a closed hashing there can be one word to a key. If words have the same key when passed through the hash function in closed hashing, linear probing must be used. Linear probing is when you keep going down the array list until finding a key that's empty. The word is inserted there. This can cause a cluster problem that can also affect time efficiency.

At first, by simply comparing the load factors of both functions 1 and 2 in table 2 we can tell that they have equal number of average probes when it comes to closed hashing; Even though neither functions are fully efficient. We have to look at search times to try and tell if one is more efficient.

	Function 1	**Function 2**
Load Factor	0.5	0.5
Successful	1.5	1.5
Unsuccessful	2.5	2.5

Table 2. The calculated load factors, Successful, Unsuccessful for each function.

The average search time for closed hash in function 1 was 2488.11355 nanoseconds successful and 3294.55678 nanoseconds unsuccessful. For function 2 the average search time was 1756.75824 nanosecond successful and 8280.06227 nanosecond unsuccessful. In figure 5, for successful, function 1 we see that most of the times stayed under 4000 nanoseconds while in figure 6 for function 2, most of the times were also under 2000 nanoseconds. From the graphs and the averages we can tell function 2 had a better successful search time. In figure 7 for function 1 for unsuccessful we see that most times stayed under 4000 nanoseconds. While in figure 8 for function 2 most times also stayed under 4000 nanoseconds. So it is hard to clearly tell from the graphs which function is better, however, from the unsuccessful search averages we can tell that function 1 is faster. Since function 1 is better for unsuccessful searches while function 2 is better for successful searches I think more test would have to be done to truly tell which is better in closed hashing.

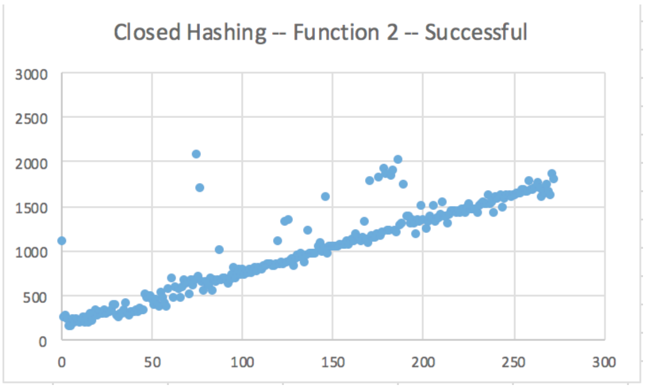

Figure 6.

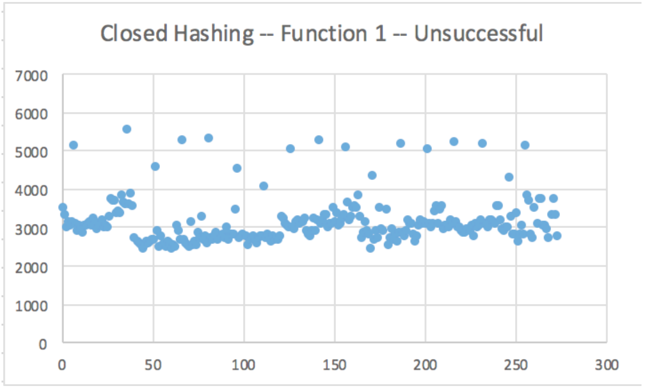

Figure 7.

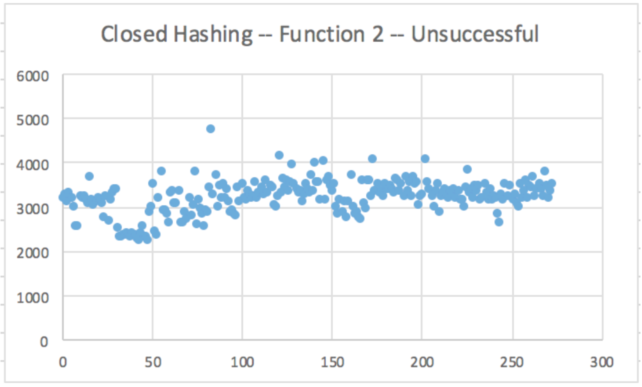

Figure 8.

Conclusion

In conclusion, I found my hypothesis to be mostly wrong. I predicted that closed hashing will have worse search times for every test. I found out that function 1 closed hash had a faster search time for both successful and unsuccessful. For function 2 open hashing had a faster search time for both successful and unsuccessful. At the same time my hypothesis was partially correct because I predicted that it depends on the hash function being used. There wasn't one clear winner for both functions. One hash was better for one of the functions while the other hash was better for the other function.

REFERENCES

[1] Levitin, A. *Introduction to the Design and Analysis of Algorithms*. Pearson, 2011.

[2] Morcinek N. *Herbal Wines*. Available from http://www.textfiles.com/drugs/winemake.txt; Accessed 11 December 2016