Average Search Time Efficiency of Open Hashing versus Closed Hashing

Bryce Barrett
Computer Science Department at the
College of Charleston
360 Concord Street
Charleston, SC, 29401
barrettbs@g.cofc.edu

1. Introduction

The proceeding report will focus on the analysis of the average case search time efficiency for both open hashing using chaining and closed hashing using quadratic probing. In this report, I hypothesize that to find a string successfully using open hashing and closed hashing, the following formulas will give an accurate representation of the average number of probes needed to find a string in a hash table. Open hashing will follow the formula $S = 1+\alpha/2$ and closed hashing will follow the formula $S = (1/\alpha)\ln(1/(1-\alpha))$. In terms of unsuccessful attempts, I hypothesize open hashing will follow the formula $U = \alpha$ and closed hashing will follow the formula $U = 1/(1- \alpha)$.

Throughout this report I will be focusing on proving that these formulas give the correct average number of probes in relation to my open and closed hashing classes that were created for the purpose of testing these formulas. In the process of testing my hypothesis, there were only a few constraints that occurred, one of these constraints was only being able to use data structures that have been provided for my computer science class throughout the year of 2017. If I was able to use the java API some of the methods in the source code would be much more readable, but also may require less time to execute. In the following sections of this report, I will present the methodology behind the experiments conducted to obtain results as well as present those results in the form of graphs with an analysis of what the data actually represents along with the comparison between the data and the formulas provided in my hypothesis.

2. Methods

2.1 Open Hashing

In the process of testing the average number of successful and unsuccessful searches for open hashing, there were a few key factors to take into consideration. The first factor was the load factor or α. I decided to test my hypothesis on load factors that were only above 1 because of the fact that I wanted the array lists of the hash tables to have some length that was greater than one. I tested a total of 11 different load factors ranging from 1 to 5 with an interval between the load factors of 0.4. The next factor that needed to be considered was the size of the hash map itself. The size of the hash map needed to be large enough to get an accurate representation of data and to avoid small errors or miscalculations that could be made in terms of the formula if the hash map had a very low size. I decided to create a hash map of size 1009 after carefully taking these facts into consideration. One of the last factors that needed to be decided was the amount of strings that I would be searching for in the successful and unsuccessful tests. I tested 400 different strings for each load factor. I did this to allow for a large enough sample size that would be able to give an accurate representation of all possible strings. The next factor is the randomness in which strings are selected to be placed and searched for in the hash map. The txt file that was used for this experiment was one that had all words in alphabetic order, so this could cause some problems when hashing the strings to values because many hash functions are based on the letters of a word. If the letters were in alphabetic order, this would mean that they could share many of the same letters which would lead to the string being hashed very closely with other strings from the txt file. I decided to use random strings from throughout the file to handle this problem. I also decided to choose random strings from throughout the strings already in the hash table to test for success. In terms of the unsuccessful tests, I randomly chose words from the txt file that were not in the hash map. In the end, I averaged the amount of probes it took to successfully find a string or unsuccessfully search for a string. I also averaged the search time (nanoseconds) to test the amount of time it takes for my search algorithm to find strings successfully and unsuccessfully. One last factor to mention, was the use of two separate hashing algorithms throughout the tests of this experiment. We used two hashing to allow us to run more tests and gather more data, to compare with each formula. This allowed us to nullify even more possible outliers in the data.

2.2 Closed Hashing

In the process of testing closed hashing for the average number of probes on successful attempts and unsuccessful attempts, there are also many factors to take into consideration. The first factor I will mention is the load factor of the hash map. I decided to test 10 different load factors throughout this experiment. They ranged from 0.1 to 0.55 with an interval of 0.05. My reasoning behind picking smaller load factors was to allow for a more thorough test of each formula stated in my hypothesis. If the load factor was above 0.5 there is a chance that some strings will not be able to be hashed into the hash map. Also, it is important to understand that as the load factor approaches 1 the amount of probes will greatly increase because there will be many more collisions in the hash map. Another important fact to note about the load factor is that it is useless to test closed hashing on anything with a load factor greater than 1 because once the load factor reaches 1, then nothing else will be able to be added to the hash map. The next factor I took into account was the size of the hash map. Again for closed hashing, I needed a hash map that was large enough to get an accurate representation of data as a whole. In choosing my hash map size, I used a similar approach as I did with open hashing which included choosing a size which allowed for outliers in my data to not affect the final result. I decided to choose 3001 as the size of my hash map because it allows me to test more strings that are actually in the hash map to offset the small load factors that I was using. One of the final factors was the amount of strings that I would be testing. As I mentioned above, since my load factors were very low I tried to offset this with a large hash map size to allow me to test more strings in the hash map. In the end, I

decided to test 200 strings that were in the hash map for the successful trials and 200 strings that were not in the hash map for the unsuccessful trials. I chose 200 new strings every time I changed the load factor to allow for a more thorough test. The next factor I took into account was the randomness of selected strings to be put into the hash map as well as the randomness of strings to be tested. To assume that I was truly getting different strings every time I inserted them into the hash map, I set up a random number generator using the Random class provided by java. I allowed this generator to select a random number up to 80,000 and insert the string at that location in my txt file into my hash map. Once I had all the strings in my hash map, I then used the same method to choose strings to test for successful searches. In terms of the unsuccessful searches, I chose random strings from my txt file greater than 80,000 to assure that none of those words were in my hash map. The last factor is the same hashing algorithms that we used to test open hashing. Again with closed hashing, we will be using the same two hashing algorithms as to help analyze our data.

3. Results

3.1 Open Hashing Successful Searches

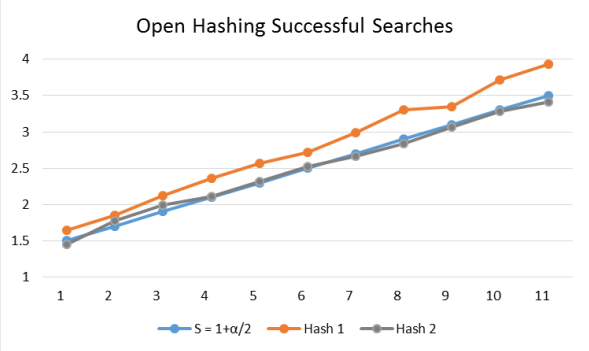

Figure 1. 11 different test load factors were used starting with 1 and going to 5 with an interval of 0.4

After collecting and analyzing the data from my tests that I previously described I have come to the conclusion that the formula for open hashing on successful searches is correct and does accurately describe the relationship between the load factor and the number of probes for a successful search. In the above graph it can be seen that each hash function does mimic the behavior of the formula very well. One discrepancy in the data however is the fact that the hash 1 function always takes slightly more probes than the hash 2 function and the formula for open hashing ($S = 1+\alpha/2$). Throughout the rest of my tests the hash one function does always seem to run a bit higher than the hash 2 function. Although this is a possible problem the hash 2 function does prove that the formula for open hashing is correct. The last thing to comment on about successfully searching for strings using open hashing is the timing of finding the said string. In my tests I found that no matter what the load factor was, the strings were always found in around 8,000 to 9,000 nanoseconds. If we analyze this number it makes sense, since we are using an array of array lists to implement our open hashing class it would make sense that the average time to find a string in an array list would always be around the same time.

3.2 Open Hashing Unsuccessful Searches

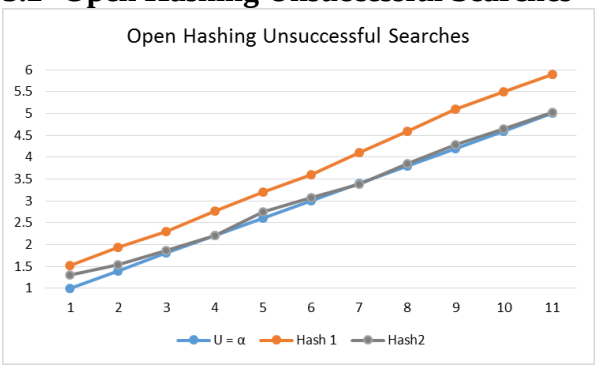

Figure 2. 11 different test load factors were used starting with 1 and going to 5 with an interval of 0.4

After analyzing data collected from the unsuccessful search attempts using open hashing, I am able to substantiate my hypothesis with this data. Upon changing the load factor of the unsuccessful attempts, I see that the average number of probes does follow the formula for unsuccessful searches using open hashing ($U = \alpha$). The data gained from testing hash 2 seemed to almost exactly mimic the formula in my hypothesis. The data that hash 1 output again follows the same trend of the formula but with a slightly increased number of probes than hash 2. This leads me to believe that hash 2 may be a slightly better hashing algorithm than hash 1. In terms of timing, I collected times that were again very similar to the timing collected for successful searches.

3.3 Closed Hashing Successful Searches

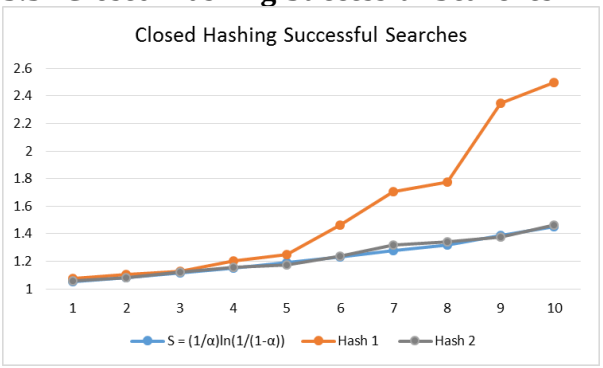

Figure 3. 10 different test load factors were used starting with 0.1 and going to 0.55 with an interval of 0.05

My tests on successful searches for closed hashing have given me enough data to substantiate my hypothesis in terms of the closed hashing successful search formula ($S = (1/\alpha)\ln(1/(1-\alpha))$). The hash 2 function follows the formula almost exactly yet again, but something interesting to note is the fact that the hash 2 function

began to stray away from the formula for successful searches as the load factor began to rise. A reason for this could possibly be because of the fact that the hash 1 function causes more secondary clustering in the hash table whereas hash 2 function does a much better job of hashing the strings as to avoid secondary clustering. Aside from the fact that hash 1 does stray form the formula slightly, the general trend of both hash 1 and hash 2 is following the path of the formula. So, we can conclude that the formula for successful searches in closed hashing is correct. Timing the successful searches in closed hashing gave similar results to open hashing in the fact that they were only a few thousand nanoseconds apart. It took on average 9,876 nanoseconds to search for a string that was in the hash map.

3.4 Closed Hashing Unsuccessful Searches

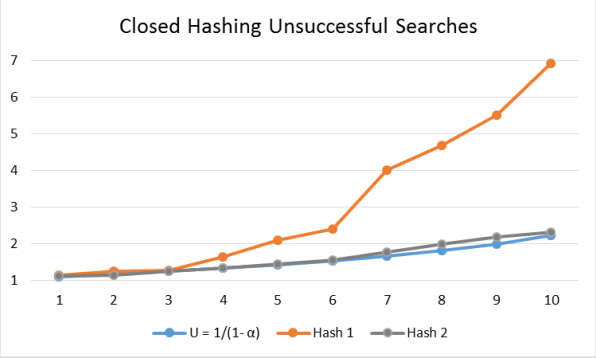

Figure 4. 10 different test load factors were used starting with 0.1 and going to 0.55 with an interval of 0.05

After close analysis of the unsuccessful searches in closed hashing, the formula in my hypothesis (U = 1/(1- α)) is again substantiated by data collected from these tests. The data from the hash 2 function closely follows the general trend of the formula for unsuccessful searches in closed hashing, but something to note is that the hash 1 function follows the formula closely for load factor values from 0.1 to 0.4, but then takes a turn for the worse and has a significantly greater average probe count than hash 2 function or the formula. This could be caused because of the amount of secondary clustering caused by the hash function. In terms of timing, the unsuccessful searches again took on average 10,123 nanoseconds to determine that the string was not in the list. This is also very similar to the time to search to see if a string is in the list. This can be attributed to the array data structure that was used to create the hash map for the closed hashing class as it takes constant time to retrieve any one item from an array.

4. Conclusion

The formula for successful searches (S = 1+α/2) in open hashing can be used to determine greater load factors as I have proven that the formula holds true for two separate hashing functions that tested 11 different load factors with the formula. In terms of the unsuccessful formula for open hashing (U = α), I can also conclude that this formula is also viable because of the extensive testing using two different hash functions and 11 different load factors. The formula for successful searches in closed hashing (S = (1/α)ln(1/(1-α))) also holds true as I tested this formula against 10 different load factors of very low values. Lastly, the formula for unsuccessful searches in closed hashing (U = 1/(1- α)) has been proven as well based on my tests with 10 different load factors. In conclusion, my hypothesis about the successful and unsuccessful searches through open and closed hashing has been substantiated by my findings through experimentation.

5. REFERENCES

[1] Buell, D.A. *Data Structures using Java.* Jones and Bartlett Learning. Burlington. 2013

[2] Preiss B.R. *Data Structures and Algorithms with Object-oriented Design Patterns in C++*. John Wiley and Sons Inc. New York. 1999

[3] "Collision Resolution." *SpringerReference* (n.d.): n. pag. Web.

[4] "Hashing Tutorial." *Hashing Tutorial: Section 2.4 - Hash Functions for Strings*. N.p., n.d. Web. 26 Apr. 2017.