6.851 Final Project
Tabulation Hashing Performance Benchmark

Maksim Stephenako
Yuzhi Zheng

May 2012

1 Introduction

Hashing is one of the most basic computer science concept. It allows elements to be reliably stored and retrieved from a limited number of slots, without dedicated slot of every possible variation of the element. While basic, hashing is used everywhere. Hashing is used in associative arrays, sometimes also known as dictionaries, in languages like PHP, Perl, and Python. Hashing can event be used for database indexing. Even lower level computer architectural components like processor caches use ideas from hashing to figure out which line to store value from a particular memory address. Hashing can also be used to keep track of sets or make sure certain data representations are unique. Even the famous MapReduce framework uses hashing to help shard inputs to be processed on different machines.

From a theoretical standpoint, hashing takes $O(1)$ time, which means it takes a constant amount of time. That is essentially as fast as it gets. However, big-O notations can not accurately depict the size of the constant factor. These constant factors sometimes have a significant but real influence on the performance of any algorithm. Since hashing is used so often, it is important to keep that constant factor as low as possible, and finding improvements whenever possible.

One of the most basic hashing function is the multiplicative hashing. Thorup and Zhang showed that a different type of hashing, tabulation hashing, could potentially be a good alternative to the more basic multiplicative hashing in their paper from 2010. More specifically, they looked at the performance of tabulation hashing used in conjunction with linear probing and found the performance to be competitive with other hash functions on dense tables.

This report takes a closer look at tabulation hashing and it's performance against the basic multiplicative hashing. Instead of only looking at linear probing, we expanded our collision resolution techniques to quadratic probing and also chaining. We plan to do some benchmark testing as well as analyzing the possible pros and cons of each type of hash functions as well as the different collision resolutions.

2 Tabulation Hashing

overall idea of tabulation hashing

make table
look stuff up etc
- 3 independence
- 4 independence
- 5 independence

3 Implementation

We implemented this project in C, hoping the result will be fast and efficiently. We enjoyed knowing exactly where certain arrays and variables are going to be laid out in memory. In the end, we have approximately 1.5k lines of code, including the hash functions, table generation, collision detection, and test code.

Fortunately for us, Thorup and Zhang included the code for tabulation hashing in their 2010 paper on 5-independent tabulation hashing. We were able to model most of our code based on what was included in the paper. We kept the logic behind how the hashes are generated, but made some changes on how the structures are stored in the code. Storing fewer pointers, hoping that will use less memory space and have higher performance.

overall view
about each function and mention the difference between them
talk about table sizes and how much space they take
problems we ran into
counting collisions instead of absolute time (or maybe we can do both)

4 Benchmark Results

Compare pure hashing vs tabulation hashing
compare linear probing
compare quadratic probing
compare chaining
compare between the three
some analysis on memory access and mention pros ad cons of each

5 Conclusion

summrize what we wanted to find out
what we did
and the results we found

References

[1] M. Thorup, Y. Zhang *Tabulation Based 5-Universal Hashing and Linear Probing*, 2010

[2] M. Thorup and Y. Zhang *Tabulation Based 4-Universal Hashing with Applications to Second Moment Estimation*, Proc. 15th SODA:608-617 2004.