A Table, A Chair ...[1]

Robin Dawes

November 1, 2021

[1] "A table, a chair, a bowl of fruit and a violin; what else does a man need to be happy?" — Albert Einstein

You will implement a variety of hashing functions and compare their effectiveness for hashing strings.

.

Preamble

A certain Canadian university has decided to get in on the super-hero film game by creating a set of interconnected movies which will be collectively called the HOTNCU (Harvard of the North Cinematic Universe). Each movie will focus on the thrilling adventures of one or more super-heroes who all happen to be students at the mysterious Institute Q, situated in a mythical far-south land called Tonario. The projected number of movies in the series is 4000. Teams of writers are already at work scripting all these movies.

The University Administration spent millions of dollars coming up with this clever disguise for the actual university.

Each movie under development has been assigned a project name to preserve secrecy. Each project name is a sequence of three 5-letter English words. A sample set of project names is provided in the file HOTNCU_project_names.txt.

You have been assigned the task of creating a data structure that can

- support insert and search operations
- provide access to each item with an average of ≤ 2 steps. For more information on this, see the section titled "Computing the Average Search Sequence Length" below.

The delete operation is optional. You are not required to implement it (but it's really easy and would be good practice).

Your hard-earned data structures expertise has convinced you that neither a sorted array nor a binary tree can meet this requirement, so you have settled on using a hash table.

The HOTNCU Project Director was previously a Computer Science professor and she has taken an interest in your project. She has already decided that you are required to use some form of open addressing. She is aware that your table will need to be > 4000 in size but she wants you to keep it small. (It's easy to store all the values in a table that is very close in size to the number of keys. The problem is that with a smaller table, the number of collisions will grow. It's also easy to get very good performance by using a huge table. The challenge is to get good performance with a small table.)

She wants you to explore three forms of open addressing: linear probing, quadratic probing and double hashing. For each method she wants you to experiment with different hashing functions and different combinations of values to determine a table size that lets you achieve the required performance standard.

Part 1:

Decide how you will convert the project names into usable key values. This may involve converting each letter in a project name to an integer, or simply treating each project name as a bit string and converting that entire bit string into an integer. You will find a wealth of ideas on the Internet. Whatever method you decide on, explain why you chose it and remember to cite your source if it is not your own creation.

Part 2:

Implement a hash table where collisions are resolved by linear probing.

Use an $h'(k)$ hashing function of your choice.

It is perfectly acceptable to just use the "string-to-integer conversion" method that you designed for Part 1. It is also acceptable to take the output of your string-to-integer conversion and apply another hashing function to it (such as "mid-square" or "multiplication method" - as explained in the course notes). As always, I encourage you to make use of available sources to learn more about hashing functions. However, using downloaded code from external sources is not acceptable – but writing your own code based on a published algorithm is fine (remember to cite your sources). You can also create your own hashing function from scratch – feel free to be creative. You

may wish to experiment with different $h'(k)$ functions to ry to reduce the number of collisions in your table.

Now try to find the smallest table size that lets you insert all the project names with an average search sequence length that is ≤ 2

Part 3:

Repeat Part 2 but use quadratic probing instead of linear probing.

Try at least two different combinations of c_1 and c_2 such as $(1, 1)$ and $\left(\frac{3}{2}, \frac{1}{2}\right)$

or other combinations

A table size must be rejected if your insertion method fails to insert one or more of the project names into the table.

Hint: start with a small table and try bigger and bigger tables until you find one that holds all the keys and satisfies the average search path length requirement.

Part 4:

Repeat Part 2 but with Double Hashing. You may want to try different combinations of $h_1(k)$ and $h_2(k)$.

You are free to choose any hashing functions you like for $h_1(k)$ and $h_2(k)$, but as with Parts 2 and 3 you must implement them yourself. You can reuse functions that you used in the earlier parts if you wish.

One approach to creating and trying different hashing functions is to start with one that involves a constant number in some way – perhaps as a multiplier, or an exponent. If changing the constant value produces different results, this gives a way to easily create new hashing functions. Here's an example:

$$
\begin{aligned}
&h_1(k): \\
&\qquad x = \log_2(k) * c_1 \\
&\qquad y = x - \lfloor x \rfloor \\
&\qquad \text{return } \lfloor m * y \rfloor
\end{aligned}
$$

This function involves the constant c_1 . We can easily create a new hashing function using a different constant c_2.

You can certainly use this function in your experiments ... but I strongly encourage you to do some reading and experiment with some others. This function doesn't seem to perform very well on the project name set – or perhaps I just haven't found a good c_1 value.

Part 5:

Let LP be the size of table required to store all the project names with an average search sequence length $\leq$ 2 , using Linear Probing to resolve collisions.

Let QP be the size of table required to store all the project names with an average search sequence length ≤ 2 , using Quadratic Probing to resolve collisions.

Let DH(d) be the size of table required to store all the project names with an average search sequence length $\leq$ 2 , using Double Hashing to resolve collisions.

Hypothesis: DH < QP < LP

Consider the results of your experiments.

Do they support the hypothesis?

Note: Since there are infinitely many variations of quadratic probing and double hashing, a small experiment such as this one cannot give conclusive evidence either way. A more comprehensive comparison would be much too time-consuming for this assignment ... but if you have some free time I think it might be interesting to do a deep dive into comparing Quadratic Probing with Double Hashing. Theory predicts that DH should be better, but maybe in practice there's not much difference ...

Second note: In this assignment I have asked you to measure the

average search sequence length. The maximum search sequence length is also of interest. In my own experiments on this data set, I usually find that when the average search sequence length is ≤ 2 , the maximum search sequence length is over 20 ! This means that most keys don't collide at all, but a few keys collide a lot.

Computing the Average Search Sequence Length

Every time your program looks at the contents of a table address, that counts as a step in the current operation. So if you are inserting a value and you try addresses 17, 5, and 83 before finally inserting the value in address 36, that insertion sequence has four steps.

Since we don't know which keys are most likely to be searched for, we can assume that each key is equally likely to be the target of a search. This means that the average number of steps in a search sequence will be exactly the same as the average number of steps in an insertion sequence. To compute that average we can add up the number of steps made during all the insertions and divide by the number of values that were inserted.

Thus you can compute the length of the search sequence for a key when the key is inserted. You don't have to insert all the keys and then search for them.

What Do I Submit?

Please submit your source code for the experiment, a summary of your experimental results, and your answer to the question regarding the stated hypothesis.

Due Date

The assignment was originally due on 20211105. This date has been changed. The assignment is now due at 11:59 PM on 20211112.