CISC-235
Winter 2020
Assignment 4

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 the far-away mythical land of Ratooni. (The University Administration is very proud of having come up with this clever disguise for the actual university.) The projected number of movies in the series will be slightly less than 2300. Teams are already at work scripting all these movies.

Each movie under development has been assigned a project name to preserve secrecy. Each project name is a 5-letter English word. A sample set of project names is provided in the file HOTNCU_project_names_2020.txt. There is **no way** this is the actual set of movie project names – those are secret!

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 $\leq$ d steps, where d is a small constant. 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.

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 > 2300 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 at two forms of open addressing: 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 **quadratic probing**.

Use an $h'(k)$ hashing function of your own 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). You must implement the algorithm yourself. 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 minimize the number of collisions in your table.

Now do this:

for $\mathrm{d} \in \{2, 3, 4, 5\}$:

try to find the smallest table size that lets you insert all the project names with an **average search sequence length**[1] that is $\leq \mathrm{d}$

You may want to try different combinations of c_1 and c_2 such as $c_1 = 1, c_2 = 1,$

$c_1 = \frac{1}{2}, c_2 = \frac{3}{2}$ 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.

---

1 See the section titled "**Computing the Average Search Sequence Length"**

Part 3:

Repeat Part 2 but with Double Hashing instead of Quadratic Probing. You may want to try different combinations of $h'(k)$ and $h''(k)$.

You are free to choose any hashing functions you like for $h'(k)$ and $h''(k)$, but as with Part 2 you must implement them yourself. You can reuse functions that you used in Part 2 if you want to.

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:

$h_1(k)$:

$x = \log_2(k) * c_1$	# c_1 is a constant
$y = x - \lfloor x \rfloor$	# y is the fractional part of x
return $\lfloor m * y \rfloor$	# m is the table size

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

$h_2(k)$:

$x = \log_2(k) * c_2$	# c_2 is a different constant
$y = x - \lfloor x \rfloor$	# y is the fractional part of x
return $\lfloor m * y \rfloor$	# m is the table size

You can certainly use these functions in your experiments ... but I strongly encourage you to do some research and find some others. These sample functions don't seem to perform very well on the project name set – or perhaps I just haven't found good c_1 and c_2 values for them.

Part 4:

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

Hypothesis: for d $\geq$ 2, DH(d) < QP(d)

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 strong 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 around 24 ! 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.

$$\text{average search length} = \frac{\text{sum of lengths of insertion sequences}}{\text{number of keys inserted}}$$

Logistics:

You may complete the programming part of this assignment in Python, Java, C or C++.

You must submit your source code, properly formatted and documented You must also submit a PDF file summarizing the results of your experiments and containing your conclusions. All files must contain your name and student number, and must contain the following statement: “I confirm that this submission is my own work and is consistent with the Queen's regulations on Academic Integrity.” Combine your files into a .zip archive for uploading.

You are required to work individually on this assignment. You may discuss the problem in general terms with others and brainstorm ideas, but you may not share code. This includes letting others read your code or your written conclusions. The course TAs will be available to advise and assist you regarding this assignment.

The due date for this assignment is 11:59 PM, March 29, 2020.