Homework 4

Due Friday, June 11 at 11:59 PM ET

Fun Sea Creature Fact: A blue whale's tongue is heavier than an elephant!

Installing and Handing In

Accept the GitHub assignment **here**. Homework is handed in through Gradescope. For written portion, submit a PDF and match the questions accordingly. For the coding portion, submit ALL of the .py files.

1 Written Problems

Problem 4.1

Mid-semester feedback form

Please fill out this mid-semester feedback form here! Make sure to use your **anonymous** Gradescope email. If you do not input the correct anonymous Gradescope email, we will not be able to grade this part of the homework for you.

Problem 4.2

Hashing with unknown table size

You are hashing items into a hash table with a universal hash function, but you're not quite sure how many items you need to hash. You arbitrarily make a hash table with 13 buckets. You would like for your hash table to have space for each item to be in a different bucket; that way, finding an element in the hash table has an expected run time of O(1). Unfortunately, it turns out you have to hash 60 items, so you must expand your hash table in order to keep the expected O(1) lookup run time. One way to handle this expansion is as follows: whenever the number of elements exceeds the number of buckets, replace the table with a table with one additional bucket. Using this approach, for *every element you insert* (after the 13th element) you are creating a new hash table, with an additional bucket. You can probably come up with a better way!

Describe in a brief paragraph what your improved solution would be. While you do not need to prove your answers, you must justify them. Make sure your explanation addresses these key areas:

1. How would you expand (no need to address shrinking) your hash table?
2. Remember that universal hash functions, as described in lecture, require that you know the size of the hash table in advance. Now you have a scenario where the hash table size is changing–how do you deal with this?

3. The costs of your solution in terms of running time, including worst case and either amortized or expected running times (whichever is appropriate), for each of your table's methods: insert, find, and delete.

Problem 4.3

Best of Two Balls and Bins

We saw in class that in universal hashing 150 Brown IDs into a hashtable of size $n = 151$, the probability of each other item being in the same bucket as item x was $1/n$. The expected number of other items in x's bucket was close to 1 so looking for x on average required looking at *two* elements. But there was always the possibility that some bucket could be *quite* full, and we'd like to avoid that.

1. Write a program for yourself that initializes an array $B[0 \dots n-1]$ to all zeroes, and then generates n random integers between 0 and $n-1$; if you generate the number i, then $B[i]$ is incremented. You can imagine that the random integer is the result of your hash function and that $B[i]$ is the corresponding bucket. When you're done, print out n and the largest entry in B (i.e., how full the fullest bucket is, if we distribute elements randomly). Run your program for n equal to powers of 2 ($n = 4, 8, 16, 32, 64, 128, 256, 512, 1024, ..., 2^{15}$) multiple times, and try to detect a pattern.

   Give an approximation for a formula (including a rough coefficient) for the average size of the fullest bucket, as a function of n. You don't need to hand in your code, but you may if you want to. However, be sure to explain your guess of the pattern. We will be grading your explanation, so including code is "showing your work" in this case and may help us assign partial credit. If you're having trouble figuring out a pattern, you should plot a graph. You don't need to explain mathematically why this formula is correct, just to identify it from your results.

2. Write a second program that's like the first, except that it generates *two* random numbers i and j, and increments the *smaller* of $B[i]$ and $B[j]$. Repeat the experiment from part 1 and come up with a new approximation for a formula. Compare your results to the results from part 1. Does the maximum bucket size grow faster or slower than in part 1 (with respect to n)? Explain why you think this is. You may need to try large values of n to see a difference.

3. Describe how you might improve on universal hashing using the idea from part 2. Your solution should involve picking TWO hash functions from the universal set (assume they're already written). Be careful to describe how to do insertion, deletion, and contains (returns true/false depending on whether a given value is in the table) in pseudocode. Your pseudocode

for each of these can be just a few lines long. Pseudocode for insertion in universal hashing looks like this:

```
function insert(val):
  // input: a value to insert to hash set H using hash function h
  // output: nothing
  index := h(val)
 H[index].append(val)
```

Note: := means "defined as". The above approach is called "best of two balls and bins", and it's been generalized to best-of-k, etc.

2 Python Problems

Problem 4.4

Queue

Overview

In the file `my_queue.py` implement a queue data structure that grows and shrinks.

Input/Output

Each of the queue's methods and their inputs and output are outlined in the stencil comments of `my_queue.py`. You can assume your queue's functions will be tested on anything that fits within the stencil comment's outlined definitions for Input, but will **not** be tested on inputs that don't fit within the outlined input definitions.

Details

- Your queue will be array based. We **strongly** recommend going back to the Expanding Stacks and Queue lecture for a refresher on how your queue should function.
- The following methods are the methods that you must implement: `init(self, int capacity)`, `size(self)`, `is_empty(self)`, `enqueue(self, item)`, `dequeue(self)`, `capacity(self)`
- **Not Permitted Functions** You are NOT allowed to use:
  - list-slicing in this assignment. An example of list-slicing is the code `myAwesomelist[4:38]` to fetch items $4, \ldots, 37$ in `myAwesomeList`.

  - Any of the following functions (the first five are list-methods and the last is a global function): `append()`, `extend()`, `remove()`, `pop()`, `insert()`, `del()`, `len()`

- **Growing and Shrinking**
  - **Growing:** To prevent your queue from having a maximum capacity, your queue will double its capacity everytime an element is to be added to a full queue. This means that if your queue has a capacity of 13, the queue should be able to have 13 elements enqueued before it grows.
  - **Shrinking:** Similarly, to prevent needless memory waste your queue will reduce its capacity by a factor of two when the queue is a quarter full. For example if your queue has a capacity of 17, it should shrink to capacity 8 when the fifth element is dequeued (i.e., when the number of items in the queue goes from five down to four). Finally, if your queue has a capacity of 3 or below, it should not shrink anymore. That being said, it CAN shrink to 3 and below (i.e, from a larger size to 3 or 2), but cannot shrink after that.
- **Maximizing Space** You **must** construct your queue so that all available space is maximized. This means you should only expand your queue when there is absolutely **no** empty space for additional elements. This means you'll have to implement the wrapping technique discussed in lecture, tracking head and tail indices.
- **Runtime** As discussed in lecture, one of the most important properties of a queue is that the enqueue and dequeue operations are amortized O(1). Since this implementation is array-based, the front of the queue will not always match with the front of the array when items are dequeued. Shifting all the elements of the queue will make the runtime greater than O(1).
- **Exceptions** If someone attempts to `dequeue` from an empty queue you should tell the user that an error occurred by raising an exception! In this assignment, you only need to raise an `EmptyQueueException` in a couple of spots, all specified in the stencil code. In the To-Do in Python, simply call `raise EmptyQueueException`. You should also check for validity of input and raise `InvalidInputException` where specified in the stencil code. You must pass some sort of message as a string into exceptions when you raise them. For example, `raise EmptyQueueException("message")`

Testing

Write all your tests in `my_queue_test.py`

3 Optional Readings

Problem 4.5

Machine learning models often use very large datasets—the popular ImageNet set contains 14 million images, meaning differences in Big O have an extreme impact on the number of computations performed and, consequently, on energy consumption associated with certain algorithms or models.

1. **Training a single AI model can emit as much carbon as five cars in their lifetimes**. - MIT Technology Review
2. **AI's Carbon Footprint Problem**. - Stanford Human-Centered Computing Initiative.
   This article also discusses how researchers are measuring and tracking carbon emissions, and what can be done to move towards "Greener AI."
3. **Interactive site for simulating the carbon footprint of a machine learning algorithm**. Click "Compute Your ML Carbon Impact" to utilize the carbon footprint calculator. Hardware type refers to various processors used, Provider and Region of Compute refer to what company is performing the computations and in what data center they are occurring.