Homework 4

Due Friday, February 22nd at 5:00 PM

"The only rule is don't be boring and dress cute wherever you go. Life is too short to blend in."
-Paris Hilton

Handing In

To hand in a homework, go to the directory where your work is saved and run `cs0160_handin hwX` where X is the number of the homework. Make sure that your written work is saved as a .pdf file, and any Python problems are completed in the same directory or a subdirectory. You can re-handin any work by running the handin script again. We'll only grade your most recent submission. To install stencil Python files for a homework, run `cs0160_install hwX`. **You will lose points if you do not hand in your written work as a .pdf file.**

1 Written Problems

Problem 4.1

Hashing with unknown table size

Silly premise: Everyone knows Paris Hilton is the ultimate it girl. From her multiple bedazzled phones, to her collection of tiaras and trucker hats, Paris always dresses like a princess. [HOT(PINK) TIP: Always dress like a princess. If you do, you'll be treated like one.] With a wardrobe of this size, she needs to have many closets. Since her tracksuit collection is the largest, she has asked her assistant, Kim Kardashian, to split up her tracksuits among her closets. Kim doesn't know how many tracksuits Paris has, especially with new ones coming in every day. She desperately needs your help to organize them. Can you help her find a way to organize an ever-increasing collection?

Universal hash functions, as described in lecture, require that you know the size of the hash table in advance. But what if you **don't** know the size beforehand? For example, the user may want to insert 60 items into a table that started out with 13 buckets (i.e., a target capacity of 13). When the number of elements exceeds the number of buckets, we could replace the table with one whose capacity is one bucket greater, but you can probably come up with a better way.

How would you handle increasing the size of the hash table? Does anything besides the size change? Describe how you should solve this problem, and discuss the costs of your solution in terms of running time, including worst case and either amortized or expected (whichever is appropriate), for each of your table's

methods: insert, find, and delete. You can assume that you're using a universal hash function. A brief paragraph is sufficient and you do not need to prove your answers, you can just justify them.

Problem 4.2

Best of Two Balls and Bins

Silly premise: Kim has started to organize Paris's tracksuits by color. [HOT(PINK) TIP: Always wear colorful tracksuits. Or else you'll look like you're actually going to the gym. Ew.] The problem is, most of Paris's are millennial pink, so they've all ended up in the same closet. Help her come up with a way to organize the tracksuits so that they'll be more evenly dispersed throughout her many closets.

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 $B[0 \ldots n-1]$ 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 4.3

Queue

Silly premise: Paris is a generous princess. She's always giving her many BFF's the gifts they deserve. With the recent tracksuit trend, she thought it would be perfect to give away some of the tracksuits that she's already worn. She wants to give them away in the order she bought them: first in, first out. The old ones don't have enough rhinestones on them, anyways. [HOT(PINK) TIP: There's no such thing as too many rhinestones. Because it's hot to bling as much as possible.] Help her create a way to give away her tracksuits in the right order!

You will be implementing the classic queue data structure. Your queue will be array based, and must exhibit growing and shrinking behavior based on how many elements are currently in the queue. Check out the Expanding Stacks and Queue lecture for more information.

Methods

You need to implement the following methods:

- `init(self, int capacity)` - initializes the queue with an initial capacity taken in as a parameter
- `size(self)` - returns an integer representing the number of items in the queue

- `is_empty(self)` - returns a boolean value for whether the queue is empty
- `enqueue(self, item)` - puts the given item in the back of the queue
- `dequeue(self)` - removes and returns the item at the front of the queue
- `front(self)` - returns but does not remove the first item in the queue
- `capacity(self)` - returns an integer representing the capacity of the queue

What you can't use

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. Some other functions that you are NOT allowed to use are the following (the first five are list-methods and the last is a global function):

- `append()`
- `extend()`
- `remove()`
- `pop()`
- `insert()`
- `del()`
- `len()`

A Note on Growing and Shrinking

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. 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, but cannot shrink after that.

A Note on 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 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). Think about how you can correctly dequeue the first element and enqueue after the last element without having to keep array and queue aligned.

Exceptions

Exceptions handle special conditions that change the normal flow of your program's execution. What if someone attempts to `dequeue` from an empty queue? You should tell the user that an error occurred by raising an exception! In this small assignment, you only need to raise an `EmptyQueueException` in a couple of spots. To do so in Python, simply call `raise EmptyQueueException`. You should also check for validity of input and raise `InvalidInputException` where specified in the stencil code.

Testing

Testing is important, and by important, we mean it's part of your grade. So make sure you think of all the different cases that need to be tested. For example, testing that None can be used as a value. Your test functions should be written in the file `queue_test.py` in the stencil code.

How To Test Your Code

To test your code, you'll need only run the `queue_test.py` file. You will need to make and hand in your own test cases. You can use built-ins for testing. Don't forget to put a comment at the top of **every** test so it's clear what the test is testing for.