Lab 3

This lab provides an introduction to queue implementations. In particular, you are asked to implement both a link-based queue and an circular array-based queue.

1 Queue Abstract Data Type

Details of each operation are given below; you will implement these operations for a link-based implementation and for a circular array implementation. You must verify, via test cases, that your implementations behave as expected (i.e., that they "work").

- `empty_queue` This function takes no arguments and returns an empty queue.
- `enqueue` This function takes a queue and a value as arguments and adds the value to the "end" of the queue.

  Because both of our implementations will mutate the data structure, this function need not (and should not) return anything.
- `dequeue` This function takes a queue as an argument and removes (and returns) the value at the "front" of the queue. If the queue is empty, then this operation should raise an `IndexError`.

  Again, because we will be mutating the data structure itself with the removal, we will only return the value being dequeued.
- `peek` This function takes a queue as an argument and returns (without removing) the value at the "front" of the queue. If the queue is empty, then this operation should raise an `IndexError`.
- `is_empty` This function takes a queue as an argument and returns whether or not the queue is empty.
- `size` This function takes a queue as an argument and returns the number of items in the queue.

For both implementations, these should *all* be $O(1)$ (i.e., constant time) operations.

2 Linked Queue

In a file named `linked_queue.py`, provide a data definition for a `LinkedQueue`. As discussed in class, there must be a way to add items to the back of the list and remove items from the front of the list.

Implement the functions listed above. Be sure that *all* of them are $O(1)$. This may require modifying the supplied `LinkedQueue` class.

Place your test cases in a file named `linked_queue_tests.py`.

3 Circular Array Queue

In a file named `circular_queue.py`, define the `CircularQueue` class for a circular array-based queue implementation and implement the aforementioned queue operations. Be sure that *all* of them are $O(1)$.

The circular buffer allows for tracking the head of the queue as an index into the array. Values are not shifted as the result of a dequeue operation; instead, the index for the head of the queue is simply updated. Similarly, the number

of elements currently in the queue is tracked and updated as values are enqueued. If we left it at that, then the queue would waste the space at the beginning of the array after each dequeue operations. Instead, the queue will "wrap around" to the beginning of the array when we read the end. Hence the name "circular" queue.

Place your test cases in a file named `circular_queue_tests.py`

Note: similar to our lab implementing a list structure, you are prohibited from using almost all of Python's list operations in your implementation. The only list operations you may use are:

- initializing with a specific size (through the `*` operator, e.g., `[None] * 100`), which will act as "allocating a new array", and
- indexing (e.g., `my_queue[4]`)

Every other builtin Python operation dealing with lists is expressly forbidden.

4 Testing

As mentioned in the syllabus, your code is periodically graded prior to the deadline and you will receive automated feedback based on the results of my tests. *But*, in order to receive *any* feedback, your tests much provide 100% test coverage of the code you are submitting. If you do not have 100% coverage, this is the only feedback you will receive.

100% test coverage means that every line of your code is run at some point in some test. Imagine if that weren't the case. That means that you have a line of code (or multiple lines of code) that could do anything, and you'd never know. You never tested them!

Your tests will go in files named:

- `linked_queue_tests.py`, and
- `circular_queue_tests.py`

5 GitHub Submission

Push your finished code back to GitHub. Refer to Lab 0, as needed, to remember how to push your local code.