Project 4: Circular Queues

Due: Thursday, February 28th 8:00 pm
This is not a team project, do not copy someone else's work.

Description

In this project, you will be implementing a circular queue. A circular queue is a data structure where items are added and removed using the FIFO (First In, First Out) principle, and the head and tail of the queue wrap around the array, creating a circular structure. Your assignment is to complete the remaining functions to implement the functionality of a Circular Queue, and use those functions to solve an application problem.

Turning It In

Your completed project must be submitted as a folder named "**Project4**" and must include:

- queue.py, a python3 file.
- readme.txt, a text file that includes:
  - Your name
  - Feedback on the project
  - How long it took to complete
  - A list of any external resources that you used, especially websites (make sure to include the URLs) and which function you used this information for. Please note that you can not use outside resources to solve the entire project, or use outside sources for multiple functions that leads to the solution of the project, then you are submitting someone else's work not yours. Outside sources use should not be used for more than one function, and it should not be more than few lines of code to solve that function. Please note that there is no penalty for using the programming codes shared by Onsay in Lectures and posted on D2L.

Assignment Specifications

The point of this project is to implement the functionality of a queue, which would be trivial using Python list methods. **Do not use Python list methods outside of the grow or shrink functions. You will lose points for any function you use Python list methods in (besides grow or shrink).**
Your task will be to complete the methods listed below:
The **Queue class** is included in the starter file with the following functions and **should not be edited.**

__init__
This function initializes an empty queue. Optional argument is capacity, which defaults to four if no argument is passed in. The five member variables of the function are:

- capacity: the number of items the queue can hold

- data: the data in the queue
- head: the index of the **first** item in the queue
- tail: the index where the **next** item should be added to the queue
- size: the number of items in the queue

__str__
This function provides a string representation of the Queue. Use this for debugging.

__eq__
This function checks if two queues are equivalent to each other. If they are equal, returns True, otherwise returns False.

The rest of the functions listed are your responsibility to complete and test. Take note of specified parameters and return values, as well as the listed time and space complexities we expect from your functions. **Do not change the function signatures.**

- **__len__(self)**
  - Returns the size of the queue
  - O(1) time complexity, O(1) space complexity

- **head_element(self)**
  - Returns the front element of the queue
  - O(1) time complexity, O(1) space complexity
- **tail_element(self)**
  - Returns the last element of the queue
  - O(1) time complexity, O(1) space complexity
- **is_empty(self)**
  - Returns whether or not the queue is empty (bool)
  - O(1) time complexity, O(1) space complexity
- **enqueue(self, data)**
  - Add an element to the back of the queue
  - Return None
  - O(1)* time complexity, O(1)* space complexity
  - Do not use python list methods (ie: append)
- **dequeue(self)**
  - Remove an element from the front of a queue.
  - Return element popped. If empty, return None.
  - O(1)* time complexity, O(1)* space complexity
  - Do not use python list methods (ie: remove, pop)
- **tail_dequeue(self)**
  - Remove an element from the back of a queue.
  - Return element popped. If empty, return None
  - O(1)* time complexity, O(1)* space complexity
  - Do not use python list methods (ie: remove, pop)
- **grow(self)**

- Doubles the capacity of the queue **immediately** when capacity is reached to make room for new elements
- Moves the head to the front of the newly allocated list
- O(n) time complexity, O(n) space complexity

- **shrink(self)**
- Halves the capacity of the queue **immediately** if the size is 1/4 or less of the capacity
- Capacity should never go below 4
- Moves the head to the front of the newly allocated list
- O(n) time complexity, O(n) space complexity

* Refers to an **amortized** time complexity.

Application Problem

When you have completed the CircularQueue class, you will be using it to complete the **greatest_val(w, values)**function. The function signature is provided in the skeleton file, outside of the other class declarations. Your task is to find the greatest value in the list **values** at each possible array of size **w.** You will append that greatest value to a new list, and return the list of all greatest values at the end of the function. You **MUST** use the CircularQueue data structure and methods to solve this problem. Any solution that doesn't use a CircularQueue will lose all points relating to this function.
Example:
input: values = [1,2,3,4] , w = 2
output: [2,3,4]
[1 2] 3 4 – greatest value is 2
1 [2 3] 4 – greatest value is 3
1 2 [3 4] – greatest value is 4

Parameter Constraints:

- If **values** is not empty, it will only contain valid numbers. It will never contain strings or Booleans.
- **w** will never be larger than the length of list **values**

Additional Requirements:

- You may **NOT** access any of the CircularQueue object member variables directly, for the LinkedList or Node classes. You are only allowed to use the class functions. (This means that if you use .head, .tail, .size, .data, or .capacity anywhere in the greatest_val function, you will lose all points relating to this function.)
- You are **NOT** allowed to use any other data structures besides a CircularQueue to solve this problem. You may declare an additional list to store and return the greatest values, but it may not be used for any other purpose.
- Time Complexity O(n), Space Complexity O(n)

Assignment Notes

- You are required to add and complete the docstring for each function. Make sure to include a description of the function, parameters AND what the function returns, otherwise your docstrings will be considered incomplete.
- Do not use Python List methods outside of the grow and shrink functions, and the max_value function ONLY to add items to the list you are returning. Indexing into a list is allowed. If you are confused on what a Python List method is, python.org is a great place to start for any general Python related concept.

Rubric

Mimir Testcases ___ /55
Additional Fully Hidden Testcases ___ /20
Manual Grading (complexity) ____ / 25

1. Accessors ___ / 2
2. Enqueue ___ / 3
3. Dequeue ___ / 3
4. Tail_dequeue ___ / 3
5. Grow ___ / 4
6. Shrink ___ / 4
7. Greatest_val ___ / 6

TOTAL ___ / 100

Project written by Sarah Johanknecht