CS-UY 1134 - Spring 2018 Midterm 2

Kiersten Page

TOTAL POINTS

78 / 100

QUESTION 1

1 Question 1 4 / 15

✓ **- 6 pts** a) Operation performed to incorrect token in the stack

✓ **- 5 pts** b) Incorrect / Missing

QUESTION 2

2 Question 2 15 / 20

✓ **- 2.5 pts** Incorrect value after enqueue('D')

✓ **- 2.5 pts** Incorrect value after enqueue('E')

QUESTION 3

3 Question 3 20 / 20

✓ **- 0 pts** Correct

QUESTION 4

4 Question 4 18 / 20

✓ **- 2 pts** Missing edge case when elem should be added at the end

QUESTION 5

5 Question 5a 7 / 7

✓ **- 0 pts** Correct

QUESTION 6

6 Question 5b-c 14 / 18

+ 18 pts correct

+ 0 pts No answer to (b) or misses the main idea

+ 2 pts No answer or misses the main idea, but shows understanding of linked list implementatioin of queue

+ 2 pts Keeps reference to the first bronze item, instead of the last gold item. In this case, would also need to modify bronze_enqueue() to deal with corner case of first bronze insertion, so taking off one point.

✓ **+ 3 pts** keeps reference to last gold item or to additional node separating golds from bronzes

✓ **+ 4 pts** gold_enqueue inserts gold item after last_gold or before first bronze or before sentinel

✓ **+ 3 pts** gold_enqueue updates last_gold on insertion (or leaves first_bronze as is)

✓ **+ 2 pts** dequeue deletes first element

✓ **+ 1 pts** dequeue returns first element or first node

+ 2 pts dequeue checks whether first node was the last gold node and, if so, updates reference to last gold (OR checks whether it's first bronze and updates that reference)

+ 2 pts memory picture has right linked list and additional data

+ 1 pts memory picture is correct Python representation of the data structures (or pretty close to it)

✓ **+ 1 pts** memory picture has correct linked list but omits other important members of gbq (e.g. reference to the boundary between gold and bronze) or they're incorrect

- 1 pts small error like: missing "self", self.data, wrong method names, mixing up prev and next, small initialization error, ...

- 2 pts incorrect access to members of the doubly linked linked list, self.data or missing initializations, etc

+ 5 pts linear time solution by tracking number of gold nodes with correct updates of this count (or by marking nodes as G or B or by searching for sentinel node dividing G from B.)

+ 3 pts linear time solution by tracking number of gold nodes without correct updates of this count (or by marking nodes as G or B without correct corner cases or searching for sentinel incorrectly or ...

+ 0 pts Check later

+ 5 pts on the right track for enqueue, but with

last_gold as integer instead of reference to node and incrementing instead of advancing to next node

+ **0 pts** check hard copy. Missing page ??

Question 1 (15 points)

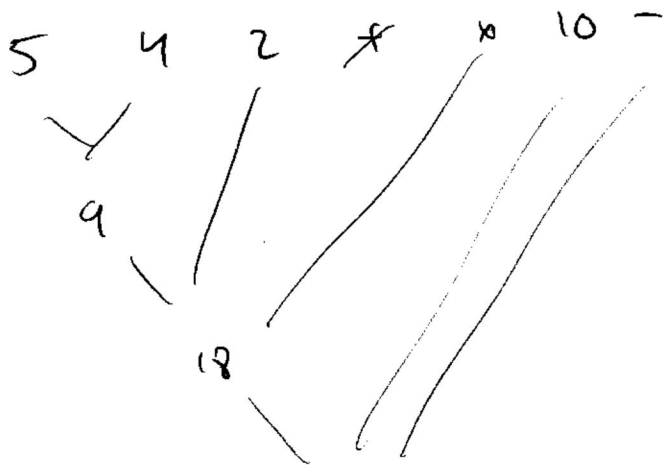
a. Consider the algorithm we studied for evaluating postfix expressions. Under each token (number or operator symbol) in the postfix expression: "5 4 2 + * 10 -", show what the stack looks like, **after** that token is processed:

5	4	2	+	*	10	-
5	4	2	9	18	10	8

b. Write an infix expression that is equivalent to "5 4 2 + * 10 -"

5 4 + 2 * 10 -

5 + 4 * 2 - 10



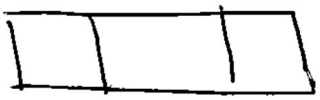
Question 2 (20 points)
Consider the "circular" array implementation of a queue that we studied in class:

```
class ArrayQueue:
    INITIAL_CAPACITY = 3

    def __init__(self):
        self.data = [None] * ArrayQueue.INITIAL_CAPACITY
        self.num_of_elems = 0
        self.front_ind = 0

    def __len__(self): ...

    def is_empty(self): ...

    def enqueue(self, elem): ...

    def dequeue(self): ...

    def first(self): ...

    def resize(self, new_cap): ...
```

first in first out

Show the values of the data members: `front_ind`, `num_of_elems`, and the contents of each `data[i]` after each of the following operations.
If you need to increase the capacity of `data`, add extra slots as described in class.

Note: the initial capacity is **3** (`INITIAL_CAPACITY=3`).



operation	front_ind	num_of_elems	data
q=ArrayQueue()	0	0	[None, None, None]
q.enqueue('A')	0	1	[A, None, None]
q.enqueue('B)	0	2	[A, B, None]
q.dequeue()	1	1	[None, B, None]
q.enqueue('C')	1	2	[None, B, C]
q.dequeue()	2	1	[None, None, C]
q.enqueue('D')	2	2	[None, None, C, D, None, None]
q.enqueue('E')	2	3	[None, None, C, D, E, None]
q.enqueue('F')	0	4	[A, B, E, F, None, None]

[C, D, E, F, None, None]

Question 3 (20 points)

Consider the following definition of the ***right circular-shift*** operation:
If *seq* is the sequence $\langle a_1, a_2, \ldots, a_{n-1}, a_n \rangle$, the ***right circular-shift*** of *seq* is $\langle a_n, a_1, a_2, \ldots, a_{n-1} \rangle$, that is the sequence we get when moving the last entry to the first position, while shifting all other entries to the next position.

For example, the right circular-shift of: $\langle 2, 4, 6, 8, 10 \rangle$, is: $\langle 10, 2, 4, 6, 8 \rangle$.

Implement the function:

```
def right_circular_shift(lnk_lst)
```

Assume the argument `lnk_lst` is a `DoublyLinkedList` object.
When called, the function should apply the right circular-shift operation on `lnk_lst` in place. That is, it **mutates** the list object, so that after the execution, it would contain the shifted sequence.

For example, if `lnk_lst` is `[2<-->4<-->6<-->8<-->10]`,
after calling `right_circular_shift(lnk_lst)`,
`lnk_lst` should be: `[10<-->2<-->4<-->6<-->8]`

Implementation requirements:

1. Your implementation must run in **worst-case constant time**.
2. In this implementation, you are **not** allowed to use the `delete_node`, `delete_first`, `delete_last` `add_after`, `add_before`, `add_first` and the `add_last` methods of the `DoublyLinkedList` class. You **should** change the values of `prev` and/or `next` for some node(s).

Write your answer on the next page

```
def right_circular_shift(lnk_lst):
    if (len(lnk_lst) <= 1):
        return
    else:
        first Node = trailer.prev
        last_node = trailer.prev.prev
        second_node = header.next
        header.next = first Node
        first Node.prev = header
        first Node.next = second_node
        second_node.prev = first Node
        trailer.prev = last_node
        last_node.next = trailer
```

ANSWER ON LAST PAGE

Question 4 (20 points)

Implement the following function:

```
def insert_to_sorted_queue(srt_q, elem)
```

This function is called with:

1. `srt_q` – a *Queue* object containing integers, appearing in an ascending order (the first element in the queue is the minimum)
2. `elem` – an integer

When called, it should add `elem` into its sorted place in `srt_q`. That is, it mutates the queue object, so that after the execution, it would also include `elem`, and remain sorted.

For example, if `srt_q` contains the elements: `<1, 3, 5, 7, 12>`,
after calling `insert_to_sorted_queue(srt_q, 6)`,
`srt_q` should be: `<1, 3, 5, 6, 7, 12>`

Implementation requirement:

1. For the queue object, you may **only** use the following operations: *q=Queue()*, *q.enqueue(item)*, *q.dequeue()*, *q.first()*, *q.is_empty()* and *len(q)* as defined for the *Queue ADT*.
   You should use the interface of the *Queue* as a black box. That is, you may not assume anything about the implementation of the *Queue*.

2. For the memory, **in addition to `srt_q`, you are allowed to use only $\theta(1)$ memory.**
   For example, you may use a constant number of integers. But you may **not** use another data structure (such as a list, stack, another queue, etc.) to store non-constant number of elements.

3. Your function should **run in linear time**. That is, if there are n items in `srt_q`, calling `insert_to_sorted_queue(srt_q, elem)` will run in $\theta(n)$.

Write your answer on the next page

Name: Kirsten Page Net ID: Kep394 -10-

elem = 5

```
def insert_to_sorted_queue(srt_q, elem):
    flag = False
    if srt_q.is_empty():
        raise (Exception ("Queue is Empty")
    else:
        for i in range (len (srt_q)):
            if srt_q.first() < elem:
                temp = srt_q.dequeue()
                srt_q.enqueue(temp)
            else:
                if flag == False:
                    srt_q.enqueue(elem)
                    temp = srt_q.dequeue()
                    srt_q.enqueue(temp)
                    flag = True
                else:
                    temp = srt_q.dequeue()
                    srt_q.enqueue(temp)
```

< 1, 2, 3, 4, 6 >

< 6, 1, 2, 3, 4 >

Question 5 (25 points)

A ***Gold-Bronze Queue*** *(GBQueue)* is an abstract data type that is similar to a Queue, but allows distinction between two categories of items: *Gold* items, and *Bronze* items. When a Bronze item is added to the *GBQueue*, it goes at the end, just like in a normal queue.
When a Gold item is added to the *GBQueue*, it goes after any Gold items that have already been added, but before all the Bronze items.

This abstract data type is useful in situations where some items (such as first-class passengers) have priority over others (economy passengers).

A *GBQueue* has the following operations:

- ***GBQueue()***: create a *GBQueue* object with no elements in it
- ***is_empty()***: returns *false* if there are one or more items in the *GBQueue*; *true* if there are no items in it
- ***dequeue()***: returns the item that is at the front of the *GBQueue*
- ***bronze_enqueue(item)***: insert a new item at the back of the *GBQueue*
- ***gold_enqueue(item)***: insert a new item after any items that have previously been added by *gold_enqueue*, but before any items that have been *bronze_enqueue*.

For example, given the following code:

```
gbq = GBQueue()
gbq.bronze_enqueue('A')
gbq.bronze_enqueue('B')
gbq.gold_enqueue('X')
gbq.gold_enqueue('Y')
gbq.bronze_enqueue('C')

while (not gbq.is_empty()):
      print(gbq.dequeue(), end=' ')
print()
```

If we evaluate the code above, it should print: **X Y A B C**

Note, this question has 3 sections.

a. Write a function:

```
def list_to_GBQueue(passengers_lst)
```

This function takes `passengers_lst`, a list whose items are pairs of strings: `(name, status)`, where there are two possible statuses: "first-class" and "economy". When called, the function should create and return a `GBQueue`, whose items are names. The returned `GBQueue`, should have all the people with status "first-class" in front of all the people with status "economy"; within each category, the people should be in the same relative order as they were in the `passengers_lst`.

```
def list_to_GBQueue(passengers_lst):
    pass Queue = GBQueue()
    for person in passengers_lst:
        if person[1] == "first-class":
            passQueue.gold_enqueue(person[0])
        else:
            passQueue.bronze_enqueue(person[0])
```

b. Complete the implementation of the GBQueue class.

Implementation requirements:

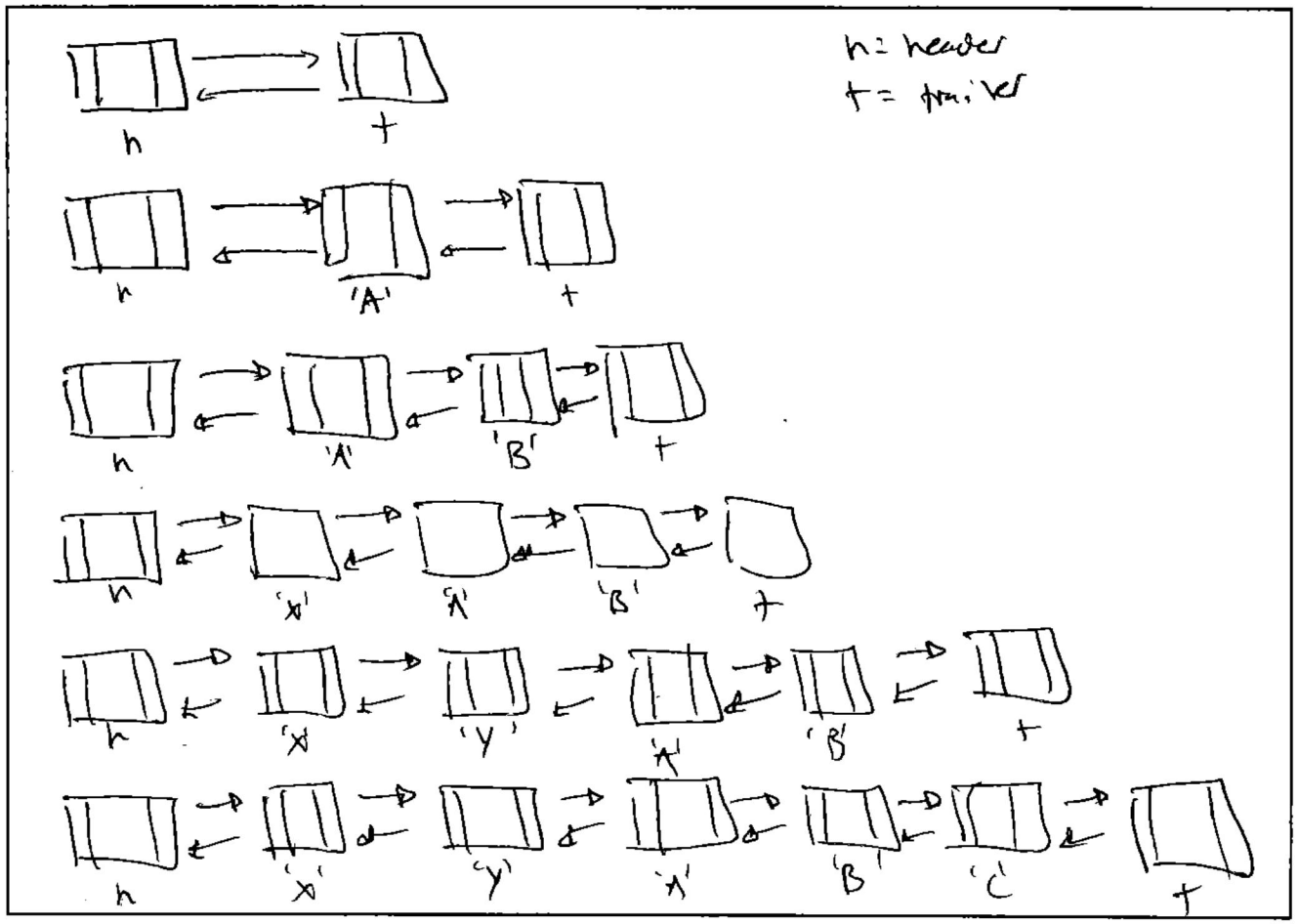
1. Data members requirement:
   A GBQueue object should have the following data-members:
   - **One DoublyLinkedList object** – in which the sequence of data items is stored. **Do not** use separate lists for the Gold and Bronze items.
   - **Constant additional space**. That is, you may not use any additional data structures that take $\theta(n)$ space, but you may use constant number of additional data members that store an integer, a double, a reference to some node in the list, etc.
2. Runtime requirement:
   ALL GBQueue operations should run in $\theta(1)$ worst-case.

Name: Kristen Page Net ID: Kep394

```
import DoublyLinkedList

class GBQueue:

    def __init__(self):

        self.data = DoublyLinkedList.DoublyLinkedList()
        self.gold_last = self.data.header.next
        [illegible, struck through]


    def __len__(self):
        return len(self.data)

    def is_empty(self):
        return (len(self) == 0)

    def bronze_enqueue(self, item):
        self.data.add_last(item)


    def gold_enqueue(self, item):
            self.data.add_after(self.gold_last, item)
            self.gold_last = self.gold_last.next


    def dequeue(self):
        if(self.is_empty()):
            raise Exception("GBQueue is empty")
        temp = self.data.delete_first( )
        return temp
```

c. draw the memory of showing what the gbq object, as you suggested in section (b), would look like, after executing the following code:

```
gbq = GBQueue()
gbq.bronze_enqueue('A')
gbq.bronze_enqueue('B')
gbq.gold_enqueue('X')
gbq.gold_enqueue('Y')
gbq.bronze_enqueue('C')
```



EXTRA PAGE IF NEEDED

Note question numbers of any questions or part of questions that you are answering here.
Also, write "ANSWER IS ON LAST PAGE" near the space provided for the answer.

#3

```
def right_circular_shift(lnk_lst):
    if (len(lnk_lst) <= 1):
        return
    else:
        newFirst = lnk_lst.trailer.prev
        newLast = lnk_lst.trailer.prev.prev
        newSecond = lnk_lst.header.next
        lnk_lst.header.next = newFirst
        newFirst.prev = lnk_lst.header
        newFirst.next = newSecond
        newSecond.prev = newFirst
        lnk_lst.trailer.prev = newLast
        newLast.next = lnk_lst.trailer
```