Name: SAHIL SINGH Net ID: SKS9379

Assigned Seat: N1

NYU, Tandon School of Engineering
CS-1134: Data Structures and Algorithms — Spring 2023

CS-1134 – Midterm Exam

Tuesday, April 18, 2023

- You have one hour and 15 minutes.
- There are 4 questions all together, with 100 points total.
- Do not unstaple the exam or remove any pages.
- Write your Name and NetID at the top of **each page**.
- If you write with a pencil, press hard enough so that the writing will show up when scanned.
- Write your answers clearly and concisely, in the spaces on the exam. Try to avoid writing near the edge of the page. **YOU MAY NOT USE THE BACKSIDE OF THE EXAM PAPERS**, as they will not be looked at. If you need extra space for an answer, use the **extra page at the end of the exam** and **mark it clearly**, so we can find it when we're grading.
- Calculators are not allowed.
- Read every question completely before answering it.
- For any questions about runtime, show the worst-case asymptotic runtime, using Theta notation.
- You do not have to do error checking. Assume all inputs to your functions are as described
- Cell phones, and any other electronic gadgets must be turned **off**.
- Do not talk to any students during the exam. If you truly do not understand what a question is asking, you may raise your hand when one of the CS1134 instructors is in the room.

```
class ArrayStack:

    def __init__(self):
        """initializes an empty ArrayStack object. A stack object has:
        data – an ArrayList, storing the elements currently in the
        stack in the order they entered the stack"""

    def __len__(self):
        """returns the number of elements stored in the stack"""

    def is_empty(self):
        """returns True if and only if the stack is empty"""

    def push(self, elem):
        """inserts elem to the stack"""

    def pop(self):
        """removes and returns the item that entered the stack last
        (out of all the items currently in the stack),
        or raises an Exception, if the stack is empty"""

    def top(self):
        """returns (without removing) the item that entered the stack
        last (out of all the items currently in the stack),
        or raises an Exception, if the stack is empty"""
```

```
class ArrayQueue:

    def __init__(self):
        """initializes an empty ArrayQueue object.
        A queue object has the following data members:
        1. data – an array, holding the elements currently in the
           queue in the order they entered the queue. The elements
           are stored in the array in a "circular" way (not necessarily
           starting at index 0)
        2. capacity – holds the physical size of the data array
        3. n – holds the number of elements that are
           currently stored in the queue
        4. front_ind – holds the index, where the (cyclic) sequence
           starts, or None if the queue is empty"""

    def __len__(self):
        """returns the number of elements stored in the queue"""

    def is_empty(self):
        """returns True if and only if the queue is empty"""

    def enqueue(self, elem):
        """inserts elem to the queue"""

    def dequeue(self):
        """removes and returns the item that entered the queue first
        (out of all the items currently in the queue),
        or raises an Exception, if the queue is empty"""

    def first(self):
        """returns (without removing) the item that entered the queue
        first (out of all the items currently in the queue),
        or raises an Exception, if the queue is empty"""

    def resize(self, new_cap):
       """resizes the capacity of the self.data array to be new_cap,
       while preserving the current contents of the queue"""
```

```
class DoublyLinkedList:
    class Node:
        def __init__(self, data=None):
            """initializes a new Node object containing the
            following attributes:
            1. data - to store the current element
            2. next - a reference to None
            3. prev - a reference to None """

        def disconnect(self):
            """detaches the node by setting all its attributes to None"""

    def __init__(self):
        """initializes an empty DoublyLinkedList object.
        A list object holds references to two "dummy" nodes:
        1. header - a node before the primary sequence
        2. trailer - a node after the primary sequence
        also a size count attribute is maintained
        3. n - the number of elements in the list"""

    def __len__(self):
        """returns the number of elements stored in the list"""

    def is_empty(self):
        """returns True if and only if the list is empty"""

    def add_after(self, node, data):
        """adds data to the list, after the element stored in node.
        returns a reference to the new node (containing data)"""

    def add_first(self, data):
        """adds data as the first element of the list. returns a reference to the
        new (first) node """

    def add_last(self, data):
        """adds data as the last element of the list. returns a reference to the new
        (last) node """

    def add_before(self, node, data):
        """adds data to the list, before the element stored in node.
        returns a reference to the new node (containing data)"""

    def delete_node(self, node):
        """removes node from the list, and returns the data stored in it"""

    def delete_first(self):
        """removes the first element from the list, and returns its value"""

    def delete_last(self):
        """removes the last element from the list, and returns its value"""

    def __iter__(self):
        """an iterator that allows iteration over the
        elements of the list from start to end"""

    def __repr__(self):
        """returns a string representation of the list, showing
        data values separated by <--> """
```

```
ass LinkedBinaryTree:
  class Node:
      def __init__(self, data, left=None, right=None, parent=None):
          """initializes a new Node object with the following attributes:
          1. data - to store the current element
          2. left - a reference to the left child of the node
          3. right - a reference to the right child of the node
          4. parent - a reference to the parent of the node"""

    def __init__(self, root=None):
        """initializes a LinkedBinaryTree object with the structure
        given in root (or empty if root is None). A tree object holds:
        1. root - a reference to the root node or None if tree is empty
        2. n - a node count"""

    def __len__(self):
        """returns the number of nodes in the tree"""

    def is_empty(self):
        """returns True if"f the tree is empty"""

    def count_nodes(self):
        """returns the number of nodes in the tree"""

    def sum(self):
        """returns the sum of all values stored in the tree"""

    def height(self):
        """returns the height of the tree"""

    def preorder(self):
        """generator allowing to iterate over the nodes of
        the (entire) tree in a preorder order"""

    def postorder(self):
        """generator allowing to iterate over the nodes of
        the (entire) tree in a postorder order"""

    def inorder(self):
        """generator allowing to iterate over the nodes of
        the (entire) tree in an inorder order"""

    def breadth_first(self):
        """generator allowing to iterate over the nodes of the
        (entire) tree level by level, each level from left to right"""

    def __iter__(self):
        """generator allowing to iterate over the data stored in the
        tree level by level, each level from left to right"""
```

Question 1 (25 points)

Consider the following definition of the ***left circular-shift*** operation:
If *seq* is the sequence $<a_1, a_2, \ldots, a_{n-1}, a_n>$, the ***left circular-shift*** of *seq* is $<a_2, a_3, \ldots, a_n, a_1>$, that is the sequence we get when moving the first entry to the last position, while shifting all other entries to the previous position.

For example, the left circular-shift of: *<2, 4, 6, 8, 10>*, is: *<4, 6, 8, 10, 2>*.

Add the following method to the `DoublyLinkedList` class:

```
def left_circular_shift(self)
```

When called, the method should apply a left circular-shift operation in place. That is, it should **mutate** the calling object, so that after the execution, it would contain the resulted sequence.

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

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/or the `add_last` methods of the `DoublyLinkedList` class.
   You **are expected** to change the values of `prev` and/or `next` attributes for some node(s) to reflect the left circular-shift operation.

Write your answer on the next page.

2 4 6 8 10 2

mfd 2 4 8 track

```
def left_circular_shift(self):
    if (len(self) <= 1):
        return
    else:
        top_node = self.header.next
        node = self.header.next.next
        node_prev = self.header.next.next.prev
        last_node = trailer.prev
        top_node_prev = top_node.prev

        self.header.next = node
        node.prev = self.header
        top_node.next = self.trailer
        self.trailer.prev = top_node

        last_node.next = top_node
        top_node.prev = last_node
```

Question 2 (30 points)

Implement the following function:

```
def remove_all(q, val)
```

This function is called with:

1. `q` – an `ArrayQueue` object containing integers
2. `val` – an integer

When called, it should remove all occurrences of `val` from q, so that when the function is done, `val` would no longer be in q, and all the other numbers will remain there, **keeping the relative order they had before** the function was called.

For example, if q contains the elements: `<4, 2, 15, 2, 2, 5, 1, 10, -2, 3>` (4 is first in line, and 3 is last in line),
after calling `remove_all(q, 2)`, `q` should be: `<4, 15, 5, 1, 10, -2, 3>`.

Implementation requirement:

1. Your function should **run in linear time**. That is, if there are n items in q, calling `remove_all(q, val)` will run in $\theta(n)$.

2. For the memory: In addition to q (the `ArrayQueue` that was given as input), your function may use:
   - one `ArrayStack` object
   - $\theta(1)$ additional memory.

   That is, in addition to q and the `ArrayStack`, you may not use another data structure (such as a list, another stack, another queue, etc.) to store non-constant number of elements.

3. **You should use the interfaces of `ArrayStack` and of `ArrayQueue` as a black box. That is, you may not assume anything about their implementation.**

Write your answer on the next page

```
def remove_all(q, val):
    stack = ArrayStack()
    if len(q) == 0
        return "Empty Queue"
    else:
        x = len(q)
        n = 0
        self.size = x
        for i in range(x):
            elem = q.dequeue()
            if elem != val:
                q.enqueue(elem)

            else:
                stack.push(elem)
                n += 1

        self.size -= n
```

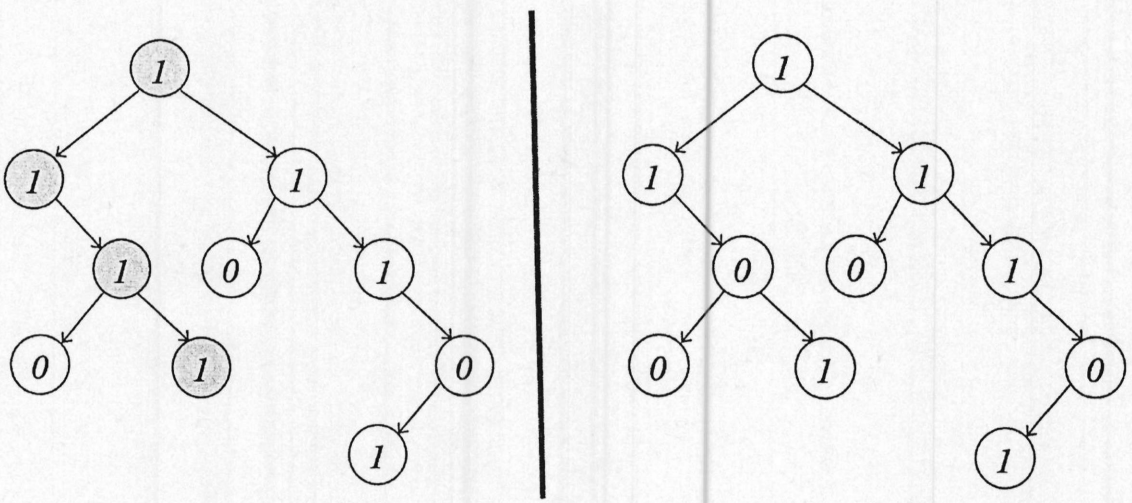
Question 3 (20 points)

Give a **recursive** implementation of the following function:

```
def has_full_path_of_ones(root)
```

This function is given `root`, a reference to a `LinkedBinaryTree.Node` that is the root of a non-empty binary tree **containing only 0s and 1s** as data (in each node).
When calling `has_path_of_ones(root)`, it should return `True` if the tree rooted by `root` **has a path that starts at the root and ends at a leaf, and that the data in all the nodes along it is 1**.

For example, if we call the function on the trees below, then, for the tree on the left, the function should return `True`. However, for the tree on the right, the function should return `False`.



Implementation requirements:

1. Your function has to run in **worst case linear time**. That is, if there are n nodes in the tree, your function should run in $\theta(n)$ worst-case.
2. You must give a **recursive** implementation.
3. You are **not allowed** to:
   - Define a helper function.
   - Add parameters to the function's header line
   - Set default values to any parameter.
   - Use global variables

Name: Sal [illegible] Net ID: [illegible] 11-

```
def has_full_path_of_ones(root):
    if root is None:
        return False

    left = has_full_path_of_ones(root.left)
    right = has_full_path_of_ones(root.right)
    if root.==1:
    return True
    if root.right is None and root.left ==1:
        return True
    if root.left is None and root.right ==1:
        return True
    if. root.left is not None and root.right is not None
        return left and right

    return False
```

Question 4 (25 points)

A ***Parity-Queue*** is an abstract data type that stores a collection of integer numbers. It still provide a FIFO behavior, like a regular queue. But, in addition to allowing to access (read or delete) the number that entered first, it also allows to access (read or delete) the even number that was entered first, and the odd number that was entered first.

A *Parity-Queue* has the following operations:

- ***pq = ParityQueue()*:** creates a new *ParityQueue* object, with no numbers in it.
- ***pq.is_empty()*:** returns *true* if there are no numbers in *pq*, or *false* otherwise.
- ***pq.enqueue(num)*:** inserts *num* to *pq*.
- ***pq.dequeue()*:** removes and returns the number that was first to enter into *pq*, or raises an *Exception* if *pq* is empty.
- ***pq.dequeue_even()*:** removes and returns the even number that was first to enter into *pq*, or raises an *Exception* if there are no even numbers in *pq*.
- ***pq.dequeue_odd()*:** removes and returns the odd number that was first to enter into *pq*, or raises an *Exception* if there are no odd numbers in *pq*.
- ***pq.first()*:** returns (without removing) the number that was first to enter into *pq*, or raises an *Exception* if *pq* is empty.
- ***pq.first_even()*:** returns (without removing) the even number that was first to enter into *pq*, or raises an *Exception* if there are no even numbers in *pq*.
- ***pq.first_odd()*:** returns (without removing) the odd number that was first to enter into *pq*, or raises an *Exception* if there are no odd numbers in *pq*.

For example, you should expect the following interaction:

```
>>> pq = ParityQueue()
>>> pq.enqueue(1)
>>> pq.enqueue(3)
>>> pq.enqueue(2)
>>> pq.enqueue(5)
>>> pq.enqueue(4)
>>> pq.enqueue(6)
>>> pq.enqueue(7)
>>> pq.dequeue()
1
>>> pq.dequeue_even()
2
>>> pq.dequeue_even()
4
>>> pq.dequeue_odd()
3
>>> pq.dequeue()
5
>>> pq.dequeue_odd()
7
>>> pq.dequeue()
6
>>> pq.is_empty()
True
```

Complete the implementation of the `ParityQueue` class below, so it would implement the *Parity-Queue ADT* defined above.
For simplicity, you are <u>not required</u> to implement the retrieve operations. That is, there is no need to implement the `first`, `first_even` and `first_odd` methods.

<u>Runtime requirement</u>: <u>Each</u> `ParityQueue` operation should run in $\theta(1)$ **<u>worst-case</u>**.

<u>Notes</u>:

1. You may use data types we implemented in class (such as `DoublyLinkedList`, `ArrayStack`, `ArrayQueue`), as data members in your implementation.
   a. Make sure to choose the most suitable data type, so you could satisfy the runtime requirement.
   b. You can't change the implementation of any of these data types. You may only use them.
2. You may want to attach an additional information to each number the user enters to the parity-queue (store them as a tuple).

<u>Hint</u>: First plan what data members each `ParityQueue` object should maintain, and how you would use them to support each operation (in constant time). After you have all that figured out, start coding.

```
class ParityQueue:

    def __init__(self):
        pq = DoublyLinkedList()
        pq_odd = ArrayQueue()
        pq_even = ArrayQueue()
```

```
    def __len__(self):
        return len(pq)
```

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

```
def enqueue(self, num):
    if len(pq) >= 0:
        pq.add_after(num)
    if num % 2 == 0:
        pq_even.enqueue(num)
    if num % 2 == 1:
        pq_odd.enqueue(num)

def dequeue(self):
    if( len(pq) == 0 ):
        raise Exception("ParityQueue is empty")
    else:
        rem = pq.delete_first()
        if rem % 2 == 0:
            pq_even.dequeue()
        else:
            pq_odd.dequeue()
    return rem
def dequeue_even(self):
    if( len(pq_even) == 0 ):
        raise Exception("No even numbers in ParityQueue")
    else:
        dele = pq_even.dequeue()
        eve = pq.delete_node(dele)
    return eve
```

```
def dequeue_odd(self):
    if( len(pq-odd)==0 ):
        raise Exception("No odd numbers in ParityQueue")
    else:
        dele = pq-odd.dequeue()
        odd = pq.delete_node(dele)

        return odd
```

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.