NYU, Tandon School of Engineering
CS-1134: Data Structures and Algorithms — Fall 2017

CS-1134 – Midterm Exam

Tuesday, November 14, 2017

- You have one hour and 20 minutes.

- There are 5 questions all together, with 100 points total.

- The exam has **TWO Parts**:
  1. The first part of the exam contains:
     - This cover page.
     - Documentation of the interface of some of the classes we implemented in the lectures. **You may use these classes and methods** without implementing them, unless explicitly stated otherwise**.**
     - A couple of pages for scratch work. **What you write in those pages will not be graded**, but you must hand it in with your exam.
  2. The second part of the exam contains the questions you need to answer, and a space for you to write your answers at. Write your answers clearly and concisely, in those spaces.
     YOU MAY NOT USE THE BACKSIDE OF THE EXAM PAPERS, as they will not be looked at. Also, try to avoid writing near the edge of the page.
     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.

- **Don't use pencils**, as they don't show up well when scanned.

- Write your Name and NetID at the head of each page.

- Calculators are not allowed.

- Read every question completely before answering it.

- For any questions about runtime, give an asymptotic analysis.

- 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.

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

        def disconnect(self):
            """deprecates 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"""

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

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

    def first_node(self):
        """returns a reference to the node storing the
        first element in the list"""

    def last_node(self):
        """returns a reference to the node storing the
        last element in the list"""

    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"""

    def add_last(self, data):
        """adds data as the last element of the list"""

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

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

    def __str__(self):
        """returns a string representation of the list"""
```

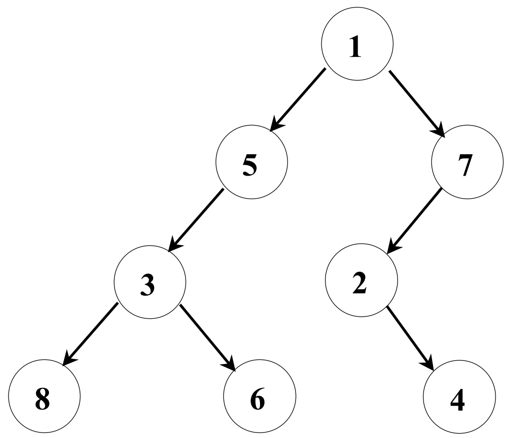
```python
class LinkedBinaryTree:
    class Node:
        def __init__(self, data, left=None, right=None, parent=None):
            """initializes a new Node object with the following attributes:
            1. data - to store the current element
            2. left - a reference to the left child of 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. size - 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 subtree_count(self, curr_root):
        """returns the number of nodes in the subtree rooted by curr_root"""

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

    def subtree_preorder(self, curr_root):
        """generator allowing to iterate  in a preorder order
        over the nodes of the subtree rooted with curr_root"""

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

    def subtree_postorder(self, curr_root):
        """generator allowing to iterate  in a postorder order
        over the nodes of the subtree rooted with curr_root"""

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

    def subtree_inorder(self, curr_root):
        """generator allowing to iterate  in an inorder order
        over the nodes of the subtree rooted with curr_root"""

    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"""
```

Scratch
(This paper will not be graded)

Scratch
(This paper will not be graded)

Question 1 (18 points)

Give the preorder, postorder and inorder traversal sequences, for the binary tree given below.



Preorder: ____

Postorder: ____

Inorder: ____

Question 2 (12 points)
Let T be a binary tree.

You are given the postorder and inorder sequences of T:
Postorder(T): 5, 6, 1, 4, 7, 2, 3
Inorder(T): 1, 5, 6, 3, 4, 2, 7

Draw T.

Question 3 (20 points)

Implement the following function:

```
def insert_sorted(srt_lnk_lst, elem)
```

This function is called with:

1. `srt_lnk_lst` – a `DoublyLinkedList` object containing integers, appearing in an ascending order.
2. `elem` – an integer

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

For example, if `srt_lnk_lst` is `[1<-->3<-->5<-->7<-->12]`,

after calling `insert_sorted(srt_lnk_lst, 6)`,

`srt_lnk_lst` should be: `[1<-->3<-->5<-->6<-->7<-->12]`

Implementation requirement: In this question, you are not allowed to use the `add_after`, `add_before`, `add_first` and the `add_last` methods of the `DoublyLinkedList` class.

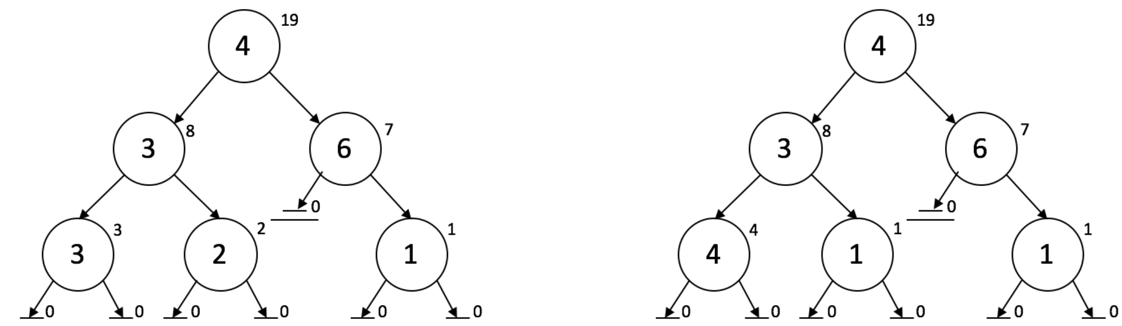
Write your answer on the next page

```
def insert_sorted(srt_lnk_lst, elem):
```

Question 4 (20 points)

Consider the following definition, of when is a binary tree considered to be sum-balanced. We say that a binary tree T satisfies the ***Sum-Balance Property*** if for every node p of T, the sum of all values in the subtrees rooted by the children of p, differ by at most 1.

For example, consider the following two trees. Note that in these figures we showed the sum of each subtree in a small font, to the right of each such root:



The tree on the left satisfies the sum-balance property, while the tree on the right does not (since the subtree rooted by the node containing 3 has one child with sum 4 and the second child with sum 1).

Notes:
1. An empty tree is sum-balanced.
2. A tree with a single node is always sum-balanced (since both its children are empty, hence their sum is 0).

In this question, we will implement the following function:

```
def is_sum_balanced(bin_tree)
```

The function is given `bin_tree`, a `LinkedBinaryTree` object, it will return `True` if the tree satisfies the sum-balance property, or `False` otherwise.

`is_sum_balanced` will call a **recursive** helper function:

```
def is_subtree_sum_balanced(subtree_root)
```

This function is given `subtree_root`, a reference to a node, that indicates the root of the subtree that this function operates on.

On the following page:

a. Complete the implementation of `is_sum_balanced`.
b. Implement the `is_subtree_sum_balanced` helper function

Implementation requirement: Your functions should run in **linear time**.

Hint: To meet the runtime requirement, you may want `is_subtree_sum_balanced` to return more than one value (multiple values could be collected as a tuple).

a.

```
def is_sum_balanced(bin_tree):

    ____________ = is_subtree_sum_balanced(bin_tree.root)

    return _________
```

b.

```
def is_subtree_sum_balanced(subtree_root):
```

Question 5 (30 points)

Give a Python implementation for the *Duplicates Stack ADT*. A duplicates stack supports operations that look at consecutive elements with the same value.

The *Duplicates Stack ADT* supports the following operations:

- **`DupStack`**(): initializes an empty `DupStack` object
- `dupS.`**`is_empty`**(): returns `True` if `dupS` does not contain any elements, or `False` otherwise.
- **`len`**`(dupS)`: returns the number of elements in `dupS`
- `dupS.`**`push`**`(e)`: adds an integer element `e`, to the top of `dupS`.
- `dupS.`**`top`**`()`: returns the top most element from the top of `dupS`, without removing it; an exception is raised if `dupS` is empty.
- `dupS.`**`top_dups_count`**(): returns the number of consecutive times the top most element appears at the top of `dupS`; an exception is raised if `dupS` is empty.
- `dupS.`**`pop`**`()`: removes and returns the top element from the top of `dupS`; an exception is raised if `dupS` is empty.
- `dupS.`**`pop_dups`**(): removes all consecutive appearances of the top most element from the top of `dupS`. This method would return the common value, that was removed; an exception is raised if `dupS` is empty.

For example, your implementation should follow the behavior below:

```
>>> dupS = DupStack()
>>> dupS.push(4)
>>> dupS.push(5)
>>> dupS.push(5)
>>> dupS.push(5)
>>> dupS.push(4)
>>> dupS.push(4)
>>> len(dupS)
6
>>> dupS.top()
4
>>> dupS.top_dups_count()
2
```

```
>>> dupS.pop()
4
>>> dupS.pop()
4
>>> dupS.top()
5
>>> dupS.top_dups_count()
3
>>> dupS.pop_dups()
5
>>> dupS.top()
4
```

Implementation requirements:

1. Data members requirement:
   A `DupStack` object should have the following data-members:
   - A `Stack` object – You may use such object without implementing the `Stack` class. Assume that `Stack` supports the *Stack ADT* (*s=Stack()*, *len(s)*, *s.is_empty()*, *s.push(e)*, *s.pop()*, *s.top()*).
   - Constant number of additional data members, if needed
2. Runtime requirement:
   Assuming that all `Stack` operation run in $\theta(1)$ worst-case, in your implementation **ALL** `DupStack` **operations should run in** $\boldsymbol{\theta(1)}$ **worst-case**.

Notes:

1. You should not assume anything about the inner implementation of the `Stack` objects. That is, you should use this class as a black box.
2. Make sure that your implementation of **`pop_dups`** **runs in constant time**.

Hint: You may want to store a tuple, as elements in the `Stack` data-member.

```
class EmptyCollection(Exception):
    pass

class DupStack:
    def __init__(self):
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________

    def __len__(self):
        ________________________________________________________
        ________________________________________________________

    def is_empty(self):
        ________________________________________________________
        ________________________________________________________

    def push(self, e):
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
```

```
def top(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________


def top_dups_count(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________


def pop(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
```

```
def pop_dups(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
```

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.