CMPSCI 187 (Fall 2018) Lab 10: Midterm 2 Preparation

The lab is due by 5:00 pm today. Please make sure that you complete your lab assignment in time and submit it to Gradescope by the deadline time.

- Go to ***File -> Make a Copy*** to make an editable copy of this Google Doc for your account
- Follow the instructions below to complete the lab
- When you are done, go to ***File -> Download As -> PDF Document***
- Log in to Gradescope and submit your PDF. Remember to submit to **Lab 10**, **NOT Project**!

Section 1. List

In this section, you will first implement a link-hopping method (the use of counters is prohibited, must follow node.next/node.prev only) for finding the middle node of a doubly linked list with head and tail pointers assuming an odd number of nodes of generic type T. If the list contains the elements [“a”, “b”, “c”], “b” is the middle element. For this method, return *null* if the list has an even number of elements.

Next, you will implement a findMedian() method to find the median of a DoublyLinkedList of type Float. Recall that the median of two real numbers is the “middle number” in the list if it is of odd length, or the arithmetic mean of the middle *two* numbers if of even length.

```
public class DLNode<T> {

	public T data;
	public DLNode<T> next;
	public DLNode<T> prev;
}
```

Given the above definition of a doubly linked node, complete the *findMiddle()* and *findMedian(DoublyLinkedList<Float> a)* methods. To help you get started, part of the code is already provided. You just need to complete the *TODO* parts.

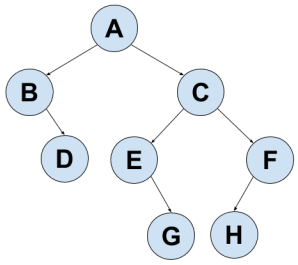
```
	public T findMiddle() {
	    if (size % 2 == 0)
	        return null;
	    DLNode<T> left = head;
	    DLNode<T> right = tail;
	    while (left != right) {
	        left = left.next;
	        right = right.prev;
	    }
	    return left.data;
	}


	public static float findMedian(DoublyLinkedList<Float> a){
	    DLNode<Float> left = a.head;
```

```
        DLNode<Float> right = a.tail;
        while (left != right && left != right.prev && right != left.prev) {
            left = left.next;
            right = right.prev;
        }
        return (left.data + right.data) / 2;
    }
```

Section 2. Tree Traversal - Recursive (20 points)

I. Warm-up: Given the following binary tree (note: this is NOT a BST), what would be the result of <u>in-order</u> traversal and <u>pre-order</u> traversal?



in-order traversal is:

B	D	A	E	G	C	H	F

pre-order traversal is:

A	B	D	C	E	G	F	H

II. Now, onwards to the main problem:

```
class Node {
    int data;
    Node left, right;
}
```

Given the above definition of a node, complete the printInOrder, printPreOrder, and printPostOrder methods of the BinaryTree class. To help you get started, part of the code is already provided. You just need to complete the *TODO* parts.

```
public class BinaryTree {
    Node root;
```

```
    BinaryTree() {
        root = null;
    }

    /**
     * TODO: Given a binary tree, print its nodes according to the
     *       "bottom-up" postorder traversal.
     */
    void printPostOrder(Node n) {
      if (n != null) {
          printPostOrder(n.left);
          printPostOrder(n.right);
          System.out.print(n.data + " ");
      }
    }

    /**
     * TODO: Given a binary tree, print its nodes in inorder
     */
    void printInOrder(Node n) {
      if (n != null) {
          printInOrder(n.left);
          System.out.print(n.data + " ");
          printInOrder(n.right);
      }
    }

    /**
     * TODO: Given a binary tree, print its nodes in preorder
     */
    void printPreOrder(Node node) {
      if (n != null) {
          System.out.print(n.data + " ");
          printPreOrder(n.left);
          printPreOrder(n.right);
      }
    }
}
```

III. Concept Question:

Is it possible for a preorder traversal to visit the nodes of T, a tree with more than one node, in the same order as a postorder traversal? If so, provide an example, otherwise argue why this cannot occur.

This cannot occur because if T has more than one node, then there must be at least a left or right node from the root that post-order will visit before the root and pre-order will visit after the root.

Section 3: Tree Traversal (Iterative)

So far, we have covered how to traverse a Binary Tree recursively. However, there's a way to traverse a Binary Tree iteratively using a stack.

Using the same BinaryTree class as above, implement in-order and post-order traversal using a stack. The functions should print the data in the nodes in the order of traversal. We are assuming that a Stack class with push(), peek(), pop(),and empty() already exists.

(a) In-order traversal

```
public void in_order_iterative(Node root) {
    if (root == null)
       return;
    Stack<Node> nodes = new Stack<Node>();
    nodes.push(root);

    while (!nodes.isEmpty()) {
       Node n = nodes.peek();
       if (n.left != null)
             nodes.push(n.left);
       else {
             while (n.right == null && !nodes.isEmpty()) {
                   n = nodes.pop();
                   System.out.print(n.data + " ");
             }
             if (n.right != null)
                   nodes.push(n.right);
       }
    }
}
```

(b) Post-order traversal. For an elegant solution to this one, use two stacks: one to manage traversal and one for printing the post-order traversal in the end.

```
public void post_order_iterative(Node root) {
    if (root == null)
       return;
    Stack<Node> nodes = new Stack<Node>();
```

```
    Stack<Node> printing = new Stack<Node>();
    nodes.push(root);

    while (!nodes.isEmpty()) {
       Node n = nodes.pop();
       printing.push(n);
       if (n.left != null)
             nodes.push(n.left);
       if (n.right != null)
             nodes.push(n.right);
    }
    while (!printing.isEmpty())
       System.out.print(printing.pop().data + " ");
}
```