

CS272 Lab Assignment #6: Doubly Linked List with Dummy Head and Tail

Learning objectives: Objective 1 (list), Objective 4 (Big-O), Objective 5, Objective 6, Objective 7

Requirements

Design and implement the doubly linked list class (with dummy nodes) that we discussed during class;
Implement it in *DoublyLinkedListDummy.java*.

Note:

- You are given the following file to use: [DIntNode.java](#).
- You should NOT add any new instance variables to the existing class file.
- You could use any methods that are already in the given file or that are discussed during the class.
- **Specifications** for all your classes and methods:
Please properly explain (1) the functionality of the methods, (2) the parameters, (3) the return values, (4) the pre-conditions if there is any; Please use inline comments, meaningful variable names, indentation, formatting, and whitespace throughout your program to improve its readability.
- You can (but are not required to) design and implement other facilitating methods (E.g., other get and set methods, toString method) to finish the implementation of the required methods.

Implement *DoublyLinkedListDummy.java* with the following detailed requirements.

1. (5 pts) This class has two instance variables:
(1) the first instance variable is the head of this doubly linked List, which should be a dummy node (instead of null reference);
(2) the second instance variable is the tail of this doubly linked List, which should be a dummy node (instead of null reference);
You should NOT add any new instance variables.
2. (10 pts) Please implement corresponding **getter, setter** methods for **head, tail** of this doubly linked list.

```
public DIntNode getHead()  
public DIntNode getTail()  
public void setHead(DIntNode node)  
public void setTail(DIntNode node)
```
3. (10 pts) The no-argument constructor which creates the dummy head and tail and link them together.
4. (10 pts) A method to add an element from the end of the list.

```
public void addEnd(int element)
```
5. (10 pts) A method to remove the first actual node (i.e., the node that the

dummy head points to).

```
public void removeFromHead()
```

Please denote (not analyze) the time complexity of your method in Big-O.

Grading note: 8 pts for method implementation; 2 pts for correctly denoting complexity.

6. (10 pts) A method to convert the list to a string.

```
public String toString()
```

This method should return a String with two lines where the first line lists all the nodes starting from the first node, and the second line lists all the nodes starting from the last node. E.g., given the doubly linked list at page 51 of the lecture notes, the output should be

```
(Forward) 12<->28<->12<->34
```

```
(Backward) 34<->12<->28<->12
```

7. (15 pts) A method to compute the number of times that the given value e occurs in nodes in this linked list.

```
public int countOccurrence(int e)
```

This method will count the times that e occurs in this doubly linked list. If no node in this list contains data e , return 0. E.g., given the following list:

```
1<->3<->7<->1
```

countOccurrence(1) should return 2;

countOccurrence(2) should return 0;

countOccurrence(3) should return 1;

Please denote (not analyze) the time complexity of your method in Big-O.

Grading note: 13 pts for method implementation; 2 pts for correctly denoting complexity.

8. (15 pts) A method to remove all the nodes that contains element e from the list.

```
public boolean removeAll(int e)
```

Please denote (not analyze) the time complexity of your method in Big-O.

Grading note: 13 pts for method implementation; 2 pts for correctly denoting complexity.

9. (15 pts) A method to extract a sublist of this list.

```
public DoublyLinkedListDummy subList(int beginIdx, int endIdx)
```

Returns a new doubly linked list which copies the portion of this list between the specified *beginIdx*, inclusive, and *endIdx*, exclusive. Suppose we have a list as below:

```
1<->2<->3<->4<->5<->6<->7
```

list.subList(0,5) would return a list

```
1<->2<->3<->4<->5
```

The precondition is

1. beginIdx is no less than 0
2. beginIdx is no greater than the size of this list
3. beginIdx is no greater than endIdx

Please denote (not analyze) the time complexity of your method in Big-O.

Grading note: 13 pts for method implementation; 2 pts for correctly denoting complexity.

10. (10 points) Bonus question.

```
public void printStatistics()
```

This method would print a brief summary of this list. For example we have a list as below:

```
1<->2<->3<->2<->1<->3<->3<->7<->10
```

printStatistics() should print the following:

number occurrence

1	2
2	2
3	3
7	1
10	1

11. Write test cases to thoroughly **test** your code.

Submission

A zipped file *your-bannerid-lab6.zip* containing your java file(s).

Grading Criteria

- The score allocation has already been put beside the questions.
- Please make sure that you test your code thoroughly by considering all possible test cases.
Your code may be tested using more test cases.