Institute of Pervasive Computing

Algorithms and Data Structures 1
Summer term 2023
Jäger, Beck, Zafar

Deadline CS: **Thu. 13.04.2023, 08:00**
Deadline AI: **Thu. 13.04.2023, 08:00**
Submission via: **Moodle**

Assignment 2

Elaboration time

Remember the time you need for the elaboration of this assignment and document it in Moodle.

Linked Lists

For this assignment, please submit a zip containing:

- the solution of the pen & paper exercise
- the source code of your **`my_sorted_doubly_linked_list.py`**

Note: **Don't change the code given in the code skeleton** (e.g., constructors, etc.), as it is necessary for the automated testing of your submission!

1. Array vs. Single Linked List — 8 points

Assume an array and a Single Linked List containing a **sorted** list of **digits** comprised from your **student ID**. The **array** sorts the digits in **descending order**, the **list** stores them in **ascending** order. For instance, for student ID 02166125 the list contains the following digits:

```
Array: [6,6,5,2,2,1,1,0]
List:  [0]->[1]->[1]->[2]->[2]->[5]->[6]->[6]
```

Use pen & paper to insert the number **4** to the array and the list. Show and comment the steps necessary to insert the digit in the array and list. Show important states and pointer variables before and after changes (as shown below). Describe in a few sentences how you would implement the insert method for an array and the list when you have reached the insertion point in the respective data structure. Show also the resulting array and list and highlight the newly introduced element.

Example pen & paper entries for the array:

```
Insert 4 into an array:
Step 0:
Array:                  [6,6,5,2,2,1,1,0]
Current element:         ^
Current element index:  0
Comment:                6 != 4 -> next element
------------------------------------------------------------------------------------------------------------
Step 1:
Array:                  [6,6,5,2,2,1,1,0]
Current element:           ^
Current element index:  1
Comment:                6 != 4 -> next element
```

Example pen & paper entries for the list:

```
Insert 4 into a list:
Step 0:
List:                   [0]->[1]->[1]->[2]->[2]->[5]->[6]->[6]
Current element:         ^
Comment:                0 != 4 -> next element by following the pointer
------------------------------------------------------------------------------------------------------------
Step 1:
List:                   [0]->[1]->[1]->[2]->[2]->[5]->[6]->[6]
Current element:              ^
Comment:                1 != 4 -> next element by following the pointer
```

Institute of
Pervasive Computing

Algorithms and Data Structures 1
Summer term 2023
Jäger, Beck, Zafar

Assignment 2

Deadline CS: **Thu. 13.04.2023, 08:00**
Deadline AI: **Thu. 13.04.2023, 08:00**
Submission via: **Moodle**

2. Sorted Doubly Linked List — 16 points

Implement a Sorted Doubly Linked List in the class **`MySortedDoublyLinkedList`** based on the class **`MyListNode`**. The list should store objects of type **`int`**. The list sorts its elements in ascending order. For your implementation use the provided code skeleton **`my_sorted_doubly_linked_list.py`** and implement the methods as described below:

`get_value``(self, index)`
Returns the value of the element at specific list index position (`0 ≤ index < size`). If the index position is not an `int` or out of range, a `ValueError` is raised.

`search_value``(self, val)`
Returns the index of the first occurrence of an element at `val`, or -1 if `val` is not contained in the list. If `val` is not an `int`, a `ValueError` is raised.

`insert``(self, val)`
Adds a node containing `val` to the list, keeping the list sorted in ascending order. In case of duplicates the new element is inserted either before or after the existing duplicates. If `val` is not an `int`, a `ValueError` is raised.

`remove_first``(self, val)`
Removes the first occurrence of the value `val` from the list and returns True if successful, otherwise returns False. If `val` is not an `int`, a `ValueError` is raised.

`remove_all``(self, val)`
Removes all occurrences of the value `val` from the list and returns True if successful, otherwise returns false. If `val` is not an `int`, a `ValueError` is raised.

`remove_duplicates``(self)`
Removes all duplicate occurrences of values from the list.

`filter_n_max``(self, n)`
Filters the list to the n highest values (`0 < n ≤ size`). If n is not an `int` or out of range, a `ValueError` is raised.

`filter_odd``(self)`
Removes all even values from the list, so that the resulting list only contains odd values.

Consider:

- Stick to the given interface.
- Make sure that **`head`** and **`tail`** references always point to the correct node.
- Keep the **`size`** variable of the list up to date after all operations.