JOHANNES KEPLER
UNIVERSITÄT LINZ

Algorithms and Data Structures 1
Summer term 2021
Dari Trendafilov, Stefan Grünberger

Assignment 2

Deadline: **Thu. 29.04.2021, 23:59**
Submission via: **Moodle**

Elaboration time

Remember the time you need for the elaboration of this assignment and document it in the file ***time.txt*** *according to the structure illustrated in the right box. Please do not pack this file into an archive but upload it as a* ***separate file****.*

```
#Student ID
K12345678
#Assignment number
02
#Time in minutes
190
```

Linked Lists & Unit Testing

For this assignment, please submit the source code of your `My_DoublyLinkedList.py` and `test_My_DoublyLinkedList.py` as specified in Task 1 and `My_SingleLinkedList.py` and `test_lists.py` as specified in Task 2.

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. Doubly Linked List 12 + 6 points

A) Implement a Doubly Linked List in the class **My_DoublyLinkedList** based on the class **My_ListNode.py.** The list should store objects of type `Integer` in descending order. For your implementation use the provided code skeleton **My_DoublyLinkedList**.py and implement the methods as described below:

```
insert_ordered(self, integer_val)
# Adds the element integer_val to the list (keeping list sorted in descending order). In case of
# duplicate(s) the new element is inserted either before or after the existing duplicate(s). If
# integer_val is not a valid Integer it shall raise a ValueError.

get_integer_value(self, index)
# Returns the value of the element at specific list index position (no removal). The first list
# element has index position 0. If the index position is out of range a ValueError is raised.

_remove(self, integer_val)
# Removes all occurrences of elements with value integer_val and returns true if
# successful, otherwise returns false. If integer_val is invalid a ValueError is raised.

remove_duplicates(self)
# Removes all duplicate values from the list, e.g. [7,6,4,4,4,1,1] -> [7,6,4,1].

reorder_list(self)
# Reorders the list, so that at the beginning there are all odd values (sorted), followed by all even
# values (sorted), e.g. [7,6,4,1] -> [7,1,6,4]. The index position of the first element with an even
# value is returned. In case there are only odd values return -1. Descending order within the sequences
# of odd and even values must be retained.
```

Consider:

- Verify the input parameter, e.g. inserting `None` objects is not allowed.
- Stick to the given interface.
- Make sure that `head` and `tail` references always point to the correct position.

B) Using the class `TestList` provided in `test_My_DoublyLinkedList.py` implement 3 unit tests to verify the methods you implemented in Task 1A. Below you can find some ideas of what could be tested. Select 3 of the methods and for each one implement one test case of your choice (you could also invent one on your own).

a) test_insert_ordered()

- insert 1 item and check list
- insert several items (not descending ordered) -> check list (correct order, links, …)
- …

b) test_remove()

- remove any element from an empty list
- generate a list and remove any element (not the first and not the last)
- generate a list and remove an element that is not in list
- generate a list and remove all elements -> especially check head/tail
- generate a list and remove the first item -> especially check head
- generate list and remove last item -> especially check tail
- …

c) test_remove_duplicates()

- generate list with duplicates (2 equal values) and call `removeDuplicates()` -> check for correct result
- generate list with duplicates (>2 equal values) and call `removeDuplicates()` -> check for correct result
- generate list with more than 1 duplicate (2 equal values) and call `removeDuplicates()` -> check for correct result
- generate list with more than 1 duplicate (>2 equal values) and call `removeDuplicates()` -> check for correct result
- …

d) test_reorder_list()

- call **reorderList** method on empty list
- generate a list with several nodes (even and odd values) and call `reorderList()` -> check result
- generate a list with even nodes' values and call `reorderList()` -> check result
- generate a list with odd nodes' values and call `reorderList()` -> check result
- …

e) test_get_integer_value ()

- call `get()` on an empty list
- generate a list and use `get()` on an index out of range
- …

Consider testing the functionality extensively (are all references pointing to the correct destination, order of elements, input parameter constraints, etc.).

2. List Performance Benchmarking 6 points

Extend the provided class **My_SingleLinkedList** with the new method **`prepend`**`(self, integer_val)`, for inserting an element at the beginning of the list (**NOTE**: this list is **NOT sorted)**. In the provided **`test_lists.py`** implement a method **`compare_lists`**`(num)`, which compares the performance of subsequent insertions of **`num`** number of elements into your **single linked list** and into the **built-in list** of Python's standard library. Make sure you **always insert at the beginning** of the lists in both cases!
For the comparison you have to measure the execution time needed for inserting the given number of elements. This can be achieved using the method **time.time_ns**() which returns the current timestamp. In order to get a meaningful result, perform the test 3x and calculate the average. The results shall be printed in the terminal.

Implement one unittest **`test_compare_lists`**`()` in **`test_lists.py`** that tests the performance of inserting 1000, 10.000, 100.000, 200.000, and 300.000 elements.

An example output of **`compare_lists`**`(1000)` could be:

```
  1000:  My_SingleLinkedList is     252.00us faster
```