PG4200 Exam November 2020

The exam should **NOT** be done in group: each student has to do the exercises on his/her own. During the exam period, you are not allowed to discuss any part of this exam with any other student or person, not even the lecturer (i.e., do not ask questions or clarification on the exam). In case of ambiguities in these instructions, do your best effort to address them (and possibly explain your decisions in some code comments). Failure to comply to these rules will result in a failed exam and possible further disciplinary actions.

This exam for **PG4200** is composed of 5 exercises. You have **24** hours to complete these tasks.

This exam does not have a grade in the *A-F* range, but rather a *Pass/Fail*. To pass the exam, **ALL** the exercises **MUST** be completed.

Each exercise must be in its own file (e.g., *.txt* and *.java*), with name specified in the exercise text. All these files need to be in a single folder, which then needs to be zipped in a *zip* file with name *pg4200_<id>.zip*, where you should replace <id> with the unique id you received with these instructions. If for any reason you did not get such id, use your own student id, e.g. *pg4200_701234.zip*. No "*rar*", no "*tar.gz*", etc. If you submit a *rar* or *a tar.gz* format (or any other format different from *zip*), then an examiner will mark your exam as failed without even opening such file.

You need to submit all source codes (eg., *.java*), and no compiled code (*.class*) or libraries (*.jar*, *.war*).

Note: there is **NO** requirement about writing test cases. However, keep in mind that, if your implementations are wrong, you will fail the exam. As such, it is recommended to write test cases for your implementations, to make sure your code works correctly. However, if you do so, do **NOT** submit them as part of your delivery (as the writing of the test cases will not be evaluated).

1) File: *Ex01.txt*

Write a regular expression that does match the following sentence:

Is this an out of season april fools joke?

However, the regex should be general enough to also match for further string variants following these properties:

a) there can be any number of spaces between the words
b) each letter can be either in lower or upper case

To clarify, the following sentence should be matched as well by the regex:

iS THIS AN oUT of sEason APRIL FOOLs joKe?

On the other hand, the following would NOT match the regex (notice the space in the middle of the first word):

i S THIS AN oUT of sEason APRIL FOOLs joKe?

2) File: *Ex02.java*

Write a new class that extends *org.pg4200.les02.list.MyList*, using *MyBidirectionalLinkedList* as starting point. However, besides having a link to the *head* and the *tail* of the list, you need to have a 3rd link, pointing to the *middle* of the list.

When you need to access an element (e.g., with a *get*), then you need to minimize the number of nodes to traverse. For example, assume a list with 100 elements, and you need to delete the element at position 40. Then, the most efficient way would be to start from the *middle* link, and go backwards following the *previous* links till position 40. On the other hand, if accessing at position 20, it would be more efficient to start from *head* and follow the *next* links.

Pay particular attention at the fact that, at each *delete* and *add* operation, the value of *middle* will likely need to be updated. Note: in case of list with an even number of elements, the middle could have 2 values, choose the lower. For example: length=1 and length=2 would have *middle* at position 0, whereas length=3 and length=4 would have it at position 1, length=5 at 2, and so on.

3) File: *Ex03.txt*

Consider the implementation in *Ex02* compared to *MyBidirectionalLinkedList*. Which one is better? Is one of them always better than the other? If yes, explain why. If not, explain in which cases one is better and in which cases it would be worse.

4) File: *Ex04.java*

Consider modifying the interface *MyStream* by adding the following method:

Optional<T> reduce(BinaryOperator<T> accumulator);

This method will need to have the following semantics (description taken from the JDK JavaDocs):

Performs a reduction on the elements of this stream, using an associative accumulation function, and returns an Optional describing the reduced value, if any. This is equivalent to:

```
boolean foundAny = false;
T result = null;
for (T element : this stream) {
  if (!foundAny) {
    foundAny = true;
    result = element;
  }
  else
    result = accumulator.apply(result, element);
}
return foundAny ? Optional.of(result) : Optional.empty();
```

but is not constrained to execute sequentially. The accumulator function must be an associative function. This is a terminal operation.

You can assume to add such method to *MyStreamSupport*. However, when you submit your *Ex04.java* file for this exam, you only need to provide that method implementation, you do **NOT** need to copy&paste the whole *MyStreamSupport*.

5) File: *Ex05.java*

Consider a string representation for exam grades, in the format of *<id><grade>*, where *id* is a number starting from 0 (can assume **NO MORE** than **500** students in an exam, and there can be exams with just a couple of students), and *grade*s in the range *A-F*. A valid string would be for example: *0A1F2F3C12F13B14B27A201B497A*

You can assume that the ids are sorted, but there can be holes in the sequence (e.g., representing students that did not submit).

Write a class with the following methods:

public static byte[] compress(String idsAndGrades)

public static String decompress(byte[] data)

You **MUST** come up with an **efficient** compression algorithm which is specialized and **customized** for this problem domain (i.e., do not use neither Huffman nor LZW, but rather use *DnaCompressor* as inspiration). If your compression-ratio is not efficient (i.e., you need to minimize the number of bits required for the compression), you will fail this exercise. For example, if there is just 1 student, you should not use more than 2 bytes in total in the compression. If there are 100 students, should not use more than 150 bytes.

In your implementation, you can rely on and use the classes *BitWriter* and *BitReader*.