PG4200 Exam June 2022

This exam for **PG4200** is composed of 5 exercises. You have **24** hours to complete these tasks. This exam will be graded according to the Norwegian system, using the grades **Pass** or **Fail**. Each exercise will be scored between 0 and 10 points. In order to **pass**, you **must** have **at least** 25 of the available 50 points, and you **must** provide an answer to all the exercises (i.e. **do not leave any exercise empty**).

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 code (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).

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.

You are free to use the code in the course repository (https://github.com/bogdanmarculescu/algorithms) as inspiration, but you will be assessed on your contribution to the solution and on the degree to which the solution answers the exercises. It is important that your solution is contained in the files named as described, and that you highlight with comments the code you have added.

Exercise 1:
File: Ex01.java

a. Assume that there is a catalog containing all the names of files included in a java project (filename and path).
   Write a regular expression that matches the names of automated test files.
   The path is assumed to be composed of modules/directory names, separated by the "/" character. You can assume that a path starts with a "/" character.
   You can assume that automated test files are source code files (so they would be .java, .cpp or .kt), and that their names contain the word "Test". You would also expect such files to be in some sort of folder/module marked test.

   Examples:
   Your expression
   Should match:
   "/pg4200algorithms/lessons/src/test/java/org/pg4200/les10/regex/MatcherTest.java"

"/lessons/src/test/java/org/pg4200/les09/UndirectedGraphTest.java"
"/solutions/src/test/java/org/pg4200/sol10/RegexExampleImpTest.java"

Should **NOT** match:
"/pg4200algorithms/lessons/src/main/java/org/pg4200/les10/regex/MatcherTest.java" – not in a test folder
"/lessons/src/test/java/org/pg4200/les09/UndirectedGraph.java" – does not contain the "Test" keyword.
"/solutions/src/test/java/org/pg4200/sol10/RegexExampleImpTest.html" – is not an executable test.

Your code will be in a .java file, in a method with the following signature:
public String regexPartA()
which will return your regular expression.

b. Write a regular expression that matches questions that involve the exam of a programming course. For example, this could be a filter on mattermost that allows someone to quickly see all the questions that are asked regarding the exam for PG4200 (or indeed other programming courses).

   You can assume that your method will receive as input an ArrayList<String>, with each string containing a particular line. The structure of each line is the same:
   you can assume that each sentence starts with the name of the user that has typed it followed by a colon ('**:**'), and that there is only one sentence per line.

   @Username: Text

   A valid course code starts with 2 or 3 letters and is followed by 3 or 4 digits. (for example, PG4200 or PGR112). The letters can be upper case OR lower case (folks don't always capitalize words in chat). For this exercise, you can assume most programming course codes start with the letters "PG".

   Your regular expression should match a line if the following conditions are all true:
   - If the line is valid (lines that do not conform to the assumption stated above are not to be matched)
   - If the line is a question
   - If the line contains a reference to programming (for example, the words "programming" or "programmering") **OR** a valid course code.
   - If the line contains the words "exam" or "eksamen"

   Examples:
   Should match:
   "@Bogdan: Is everyone ready for the pg4200 exam?"
   "@Bogdan: Are the exercises for the PGR112 exam ready?"
   "@Sven: When exactly is the next programming exam?"
   "@Harald: Where can I find more exercises for the programmering eksamen?"

Should **NOT** match:
"@Harald Where can I find more exercises for the programmering eksamen?" – not valid (missing ":")
"@Hardrada: Where can I find more ships to invade England?" – not about programming
"@Napoleon: I don't even know what programmering means, let alone be ready for the exam!" – not a question
"@Armfeld: Do we have to conquer Bergen for the HIS4230 exam?" – the course code does not start with PG, thus signaling it is not a programming course.

Your code will be in a method with the following signature:
public String regexPartA()
which will return your regular expression.

NOTE: The examples above are a subset of tests. Your solution is expected to be general enough to allow other valid examples to be added.

Exercise 2:
File Ex02.java
Consider a city that has 2 main means of public transport: metro and tram.
The tram and the metro network are different and can be thought of as two different graphs. Both graphs will have the same nodes (meaning the same stops), but will have different edges. You can use the UndirectedGraph example from the course github repository as an example. You can also extend that class to add new functionality (as was done, for example, with the AllPathsGraph).

Below you have a method that selects a node, randomly, from a graph. You can use that to randomly select two nodes for the task. Since both graphs should have the same nodes, you can apply this method on either graph.

Write a method that computes the path between two nodes. The path may begin in one of the graphs and end in the other (for example, someone takes a metro for the first stops and a tram for the last ones).

The method should return:
1 – a path with no changes of transport type, if one exists
2 – A path with one change of transport type
3 – an empty path if no path exists that fulfills the previous two requirements.
(You don't need to change back and forth between graphs, one change would be enough).

Your solution can be **inspired** from the code in the course, but it **must** be specialized to deal with the situation in this particular example.

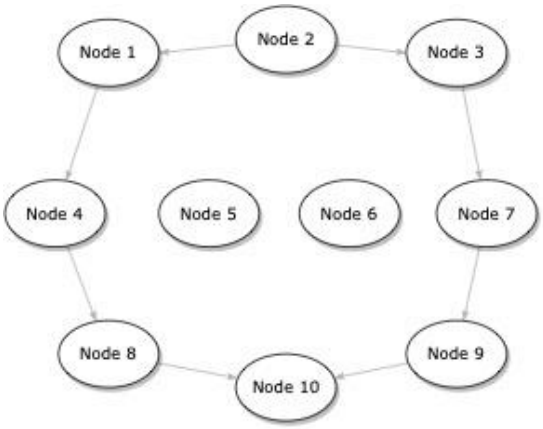
A method for randomly selecting a node in a graph:

```
public V selectNodeRandomly(){
   List<V> keys;
   Random rand = new Random();
   keys = List.copyOf(graph.keySet()); // convert the key set
```

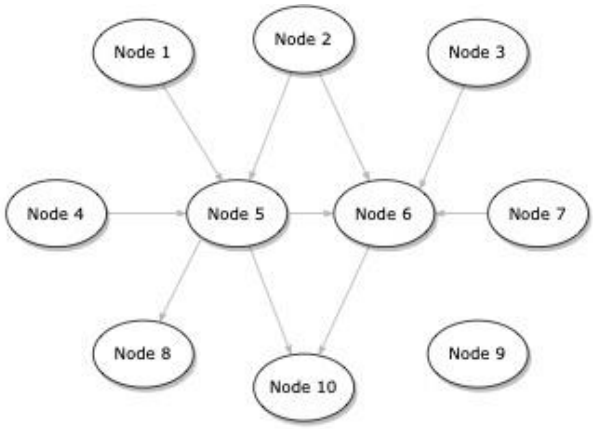
```
into a list
    int sel = rand.nextInt(keys.size()); // select a random
number between 0 and the max size of the list
    return keys.get(sel); // return the entry for the randomly
selected number
}
```

Example:

Metro network:



Tram network:



The method will have the signature:

```
public List<V> findPath(V start, V end)
```

For example,
findPath between Node5 and Node 9 would return:
{Node5, Node10, Node9}

NOTE: The name of the file will be Ex02.java. You will have to make some decisions about how to implement your method, and how to use the two graphs describing the two networks: metro and tram. You should write the assumptions you made, the reasoning, and describe your solution in comments or in a separate file called README.md.

Exercise 3:
File: Ex03.txt

Consider your implementation of the graphs from exercise 2. The graph would be based on the UndirectedGraph in the repository. That class uses a hashmap to keep track of nodes and edges.

Discuss the runtime performance of a hashmap. Could implementing the graph in another way be more efficient (and if so, how)? In particular, focus on the runtime performance of possible methods of finding a path between two nodes.

Discuss the runtime performance of the method finding a path between two different nodes, if those nodes are on different graphs (for example, between node 6, which is on the metro line, and node 9 which is on the tram line).

Note: In this exercise you are expected to support your conclusions with analysis and arguments.

Exercise 4:
File: Ex04.java

Consider a Program, composed of various courses at a university.

```
public class Program {
    public String programName;
    public ArrayList<Course> courses;

    public ArrayList<Course> getCourses(){
        return courses;
    }
}
```

Each course has a name, a course code, and a HashMap of students attending.

```
public class Course {
    public String courseName;
    public String courseCode;
    public HashMap<Integer, Student> students;
}
```

Each author is a Student, that has a firstName, lastName, and a studentId. The HashMap in the course class uses the studentid as a key (since it has to be unique).

```
public class Student {
    public String firstName;
    public String lastName;

    public Integer studentId;

    public Student(String firstName, String lastName, Integer
studentId){
        this.firstName = firstName;
        this.lastName = lastName;
        this.studentId = studentId;
    }
}
```

The task is, starting from a **program** and using streams:
Go through all the courses in a program, obtain a list of all the students that attend each course (each student should only appear once), and then create a new email address for each of the students (and return the list of student addresses).

An address will be composed of:

- The first letter of the first name
- The first letter of the last name
- The last 2 digits of the student id.
- "@hk.no" – to mark it out as an email address.

For example, a program that has a course with 2 students:

```
Student("Harald", "Hadrada", 1066)
Student("Nicolas", "Davout", 1823)
```

Will return a list with the following 2 generated addresses:
{"HH66@hk.no", "ND23@hk.no"}

Your solution will be implemented in a method with the signature:

```
public ArrayList<String> ex04(Program program)
```

that takes an object Program as parameter, and returns the list of email addresses.

Exercise 5:
File: Ex05.java

Write a compression algorithm for keeping track of course feedback.

For each entry in the course feedback list, you have:
<program> - the Program that the student is a part of. This can be (for this exercise): Programming, Artificial Intelligence, Front-end Programming, Cybersecurity, Data Science.
<course-code> - The code of the course receiving the feedback. The course code is composed of a pair of characters denoting the program that offers it, and a numerical value between 100 and 9999.
Examples:
PG4200 – a course in the Programming program
AI1701 – a course in the Artificial Intelligence program
FP1453 – a course in the Front-end programming program
SC1007 – a course in the CyberSecurity program
DS0112 – a course in the Data Science program
<unique-id> - an anonymous code, with a value between 100,000 and 900,000 (so, always 6 digits).
<assessment> - an overall grade, from A (best) to F (worst).
< date-of-completion > - the date at which the feedback was given.
<filename> - the pdf file where the full feedback can be found. The name of the file is always of the form: feedback-<**course-code**>-<**unique-id**>.pdf

Each item of feedback is described by a separate line with the format:
<program>-<course-code>:<assessment> / <date-of-completion>. File: <filename>;

Example inputs:
Programming-PG4200:456987 / 2022-JUN-06. File: feedback- PG4200-456987.pdf;
ArtificialIntelligence-AI1701:987456 / 2021-AUG-13. File: feedback-AI1701-987456.pdf;
FrontendProgramming-FP1453:963258 / 2022-OCT-30. File: feedback-FP1453-963258.pdf;
Cybersecurity-SC1007:741654 / 2022-JAN-05. File: feedback-SC1007-741654.pdf;

DataScience-DS0112:159753 / 2020-MAR-08. File: feedback-DS0112-159753.pdf;

You **MUST** come up with a compression algorithm that is **customized** for this problem (i.e. do not use Huffman or LZW, but rather use *DnaCompressor* and the exercises in class as inspiration).

In your implementation, you can use the classes BitWriter and BitReader in package *org.pg4200.les11*;

NOTE: You must provide both a **compression** and a **decompression** function. If one applies the compression function to some data, followed by the decompression function, the results should be identical to the original data (i.e. the compression is to be lossless).

The algorithm you propose should be **efficient**. You should strive to compress the data as much as you can. At a minimum, the size of the compressed data should be less than the size of the decompressed data.