EECS 2011: Assignment 4

Due: as set in *eClass.*

May be done in groups of up to three students (from either of the sections M and Z)

Motivation

The purpose of this assignment is to implement two graph algorithms: the minimum spanning tree and the shortest path.

Introduction

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of vertices (also called nodes or points) which are connected by edges (also called links or lines).

The included skeleton implementation provides several methods to build a graph data structure from a file, etc. It also provides an iterator that visits the elements. Note that not all of the operations are implemented.

Description

In this assignment, you will have to write two implementations of graph algorithms. For your convenience, some starting code is provided. Note that the starter code may either not implement some of the required features, or might implement them improperly – e.g., via stubs. Note that the speed of your algorithms may strongly depend on the implementation of the missing methods and on the choice of auxiliary data structures you choose to utilize for your algorithms. Unlike in the previous assignments, in this one you are permitted to use any `java.util` classes like `ArrayList`, `PriorityQueue`, or `HashMap`.

The graph is implemented a class called `Graph` (currently *abstract*), with the `DistanceGraph` extending it and implementing the loading of the graph content from a file. Take a look at the associated classed to confirm you understand the overall structure of the classes.

Part 1

Finish the implementation of the following two methods in the `Graphs` class (note the -s):

```
public static Path shortestPath(Graph graph, String from, String to)
public static Graph MST(Graph graph)
```

The first method should return a `Path`, containing inside a list of edges connecting the from and to vertices with one another. E.g., for a pair of vertices `c` and `f`, the path may contain the following edges in the list, in this specific order:

```
(c) -----4----- (d), (d) -----5----- (a), (a) -----3----- (e), (e) -----7----- (f)
```

The second method should return another graph, containing only the vertices and the edges that are a part of the MST. You may use any of the described MST algorithms in your implementation.

Of course, you are free to implement any private or protected methods and classes as you see fit. However, you should not modify the signatures of the existing methods if you choose to modify the method bodies. The main method in the `A4demo` class should be able to print the results of your two algorithms.

Part 2

Nothing needs to be submitted for this part.

Explore your implementation, by redesigning some methods or by modifying the input, to see if the graphs based on distances and those based on times can produce different outputs.

NOTES:

1. Do not use *package*-s in your project. Using packages will cost you a 10 % deduction from the assignment mark.
2. Some aspects of your code will be marked automatically (e.g., how it handles boundary cases and error conditions), using a custom tester class and/or custom unit tests. It is also imperative you test your classes. If any of the java files that you submit do not compile, the whole submission will be given a grade of zero, regardless of how trivial the compiler error is. The code you submit should compile using

   ```
   javac *.java
   ```

   command in either Windows, Linux, or macOS.
3. Your code should include Javadoc comments.

Submission

Find all the `java` files in your project and submit them electronically via eClass (**do not** zip, tar, rar, or 7z them). Submit only the file(s) you modified. You may create other classes, if you deem them necessary.

If working in a group, **make only one submission** per group and include a **`group.txt`** file containing the names and the student numbers of the group members. The deadline is firm. Contact your instructor *in advance* if you cannot meet the deadline explaining your circumstances.

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[1]. We look at whether the code passes the unit tests, satisfies the requirements of this document, and whether it conforms to the code style rules.

---

[1] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/

Academic Honesty

Academic honesty will be strictly enforced in this course. Specifically, direct collaboration (e.g., sharing code or answers) is not permitted, and plagiarism detection software will be employed. You are, however, allowed to discuss the questions, ideas, or approaches you take.

Cite all sources you use (online sources – including web sites, old solutions, books, etc.) in your code comments; using textbook examples is allowed, but these must be cited as well.

E.g.,

```
1) /**
     * Constructs an empty list with the specified initial capacity.
     *
     * @param   initialCapacity   the initial capacity of the list
     * @exception IllegalArgumentException if the specified initial
capacity
     *            is negative
     * uses code from www.xxx.yyy.edu/123.html for initialization
     *
     */


2) //formula based on Jane Smart's thesis, page XX, available at
https://...
        a = b + c;
```

Although using outside sources is allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced. You might find the following article useful to have an idea what kind of collaboration is appropriate: https://en.wikipedia.org/wiki/Clean_room_design

You may not post the contents of this assignment, nor solicit solutions on any external resources.