EECS 2011: Assignment 4

Due: as set in *eClass.*

May be done in groups of up to three students

> *TL;DR: implement one static method in one class. Submit that class. Cheating not allowed but some external code use is officially permitted.*

Motivation

The purpose of this assignment is to implement a commonly used graph algorithm and use it to solve a related problem.

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 functionality is implemented.

You are encouraged to review the textbook chapters on graphs. You may also use the Java examples from the Algorithms textbook[1] (please cite them if you do). A starter code is also supplied.

Description

In this assignment, you will have to write an implementation of a graph algorithm. 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. You are permitted to use any `java.util` classes: `ArrayList`, `PriorityQueue`, `HashMap`, etc.

The graph is implemented a class called `Graph`, implementing 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 method in the `Graphs` class (note the **-s** ending of the class name):

```
public static Set<String> nearby(Graph graph, String origin, int hops)
```

[1] https://algs4.cs.princeton.edu/40graphs/

The method allows one to determine which cities are *nearby* a specified city, within a specified number of edges, and returns a set of such cities as `String`s, with the corresponding distances (edge counts) appended to them after a comma. E.g., calling the method with Oshawa as origin, and 2 as the number of edges should produce a set with the following items:

"Port Perry, 1", "Ajax, 1", "Bowmanville, 1", "Uxbridge, 2", and "Scarborough, 2".

You will notice that the graph files contain the actual distances and possible travel times. For your algorithm, these will merely indicate that there is an edge present between a pair of vertices.

As usual, 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 in the classes supplied if you choose to modify the method bodies. The main method in the `A4demo` class should be able to print the mock results of your algorithm, to illustrate the API.

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. Three `csv` files are supplied with this assignment, one based on an Ontario map, and one more based on a graph example in the SW textbook.

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 are expected to change only the **`Graphs`** class. 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. The extension may or may not be granted, at instructor's discretion.

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[2]. 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.

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

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