UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Practical 7

Release: Monday 31 May 2021, 18:00
Deadline: Friday 4 June 2021, 18:00

PLAGIARISM POLICY

UNIVERSITY OF PRETORIA

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.library.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the *Library* quick link, and then choose the *Plagiarism* option under the *Services* menu). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

Objectives

The aim of this practical is to learn how to create Graphs and how to find shortest paths and articulation points.

Instructions

Complete the task below. Certain classes have been provided for you alongside this specification in the *Student files* folder. You have been given a main file which will test some code functionality, but it is by no means intended to provide extensive test coverage. You are encouraged to edit this file and test your code more thoroughly. Remember to test boundary cases. Submission instructions are given at the end of this document.

Graphs - Shortest Path & Articulation Points

A Graph is a collection of vertices and the connections between them. The connections are called edges. Each edge connects a pair of vertices. If the edges are not bi-directional, but directed from the start vertex to the end vertex, the graph is a directional graph or digraph. Each edge can be assigned a number that can represent values such as cost, distance, length or weight. Such a graph is then called a weighted digraph. You are required to implement a Shortest Path algorithm for a weighted digraph with positive and negative weights. Your algorithm must be able to deal with unreachable nodes. Your algorithm must also be able to handle cycles and self-cycles. Negative cycles should be detected. You have been given functional `Vertex` and `Edge` classes and a partially implemented `WeightedDirectedGraph` class to use.

In addition, your second task requires you to find articulation points in an undirected graph. Articulation points are vertices in a graph which, when removed, cause the graph to separate into two subgraphs with no paths between the two subgraphs. A graph may have no articulation points at all, depending on its degree of connectivity. You have been given an `UnweightedUndirectedGraph` class for this task, which will also make use of the `Vertex` and `Edge` class.

Task 1: Shortest Path [20]

```
List<Vertex> getShortestPath(Vertex sourceVertex, Vertex targetVertex)
```

This function should return the shortest path to the given `targetVertex`. The returned list should contain all the vertices from the source vertex to the target vertex in the order that describes the shortest path. Should the target vertex not be reachable, an empty list should be returned (**not null**). The predecessors can be stored in the provided field in the Vertex class. If a negative cycle is detected, `null` should be returned. The pre-condition is that the graph is a **weighted digraph**. This task makes use of the **Edge**, **Vertex** and **WeightedDirectedGraph** classes.

Task 2: Articulation Points [20]

`List<Vertex> getArticulationPoints()`

This function should return a **sorted** list of the vertices which are articulation points in an **undirected graph** . Your function should add all the vertices which are articulation points to the list called `articulationPoints` provided for you in the `UnweightedUndirectedGraph` class. A helper function has been provided called `sortArticulationPoints()` which will then sort your articulation points. After you have sorted the articulation points you have found, you can return the `articulationPoints` list. Refer to **Section 8.6.1** in the textbook for a description of articulation points in an undirected graph. Figure 1 shows the articulation points on a graph highlighted in red. If there are no articulation points or the graph is empty, return an empty list. This task makes use of the **Edge**, **Vertex**, **UnweightedUndirectedGraph** and **VertexNameSorter** classes.

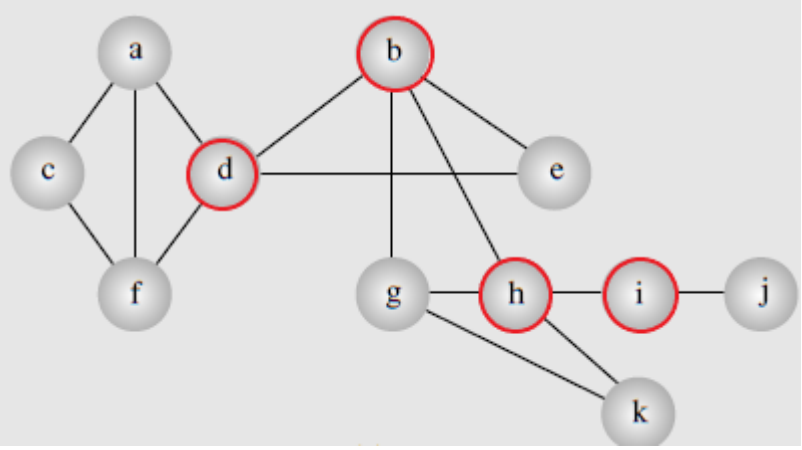

Figure 1: Graph from figure 8.16 in the textbook with articulation points highlighted in red.

You should use your own helper functions to assist in implementing these methods as per specification. However, you may not modify any of the given method signatures. You may add fields to the provided classes, but you should not add additional classes. Do not add any additional imports.

Submission

You need to submit your source files on the Fitch Fork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. The following files: **Edge.java**, **Vertex.java**, **WeightedDirectedGraph.java** and **UnweightedUndirectedGraph.java**, should be placed in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files in your submission. You have 4 days to complete this practical, regardless of which practical session you attend. You have 5 submissions and your best mark will be your final mark.

Upload your archive to the *Practical 7* slot on the Fitch Fork website. Submit your work before the deadline. **No late submissions will be accepted!**