UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Faculty of Engineering, Built Environment & IT

University of Pretoria

COS212 - Data structures and algorithms

Practical 7 Specifications: Shortest Path, Cycle Detection, Depth First Traversal of Graph Datastructure

Release Date: 09-05-2022 at 06:00

Due Date: 13-05-2022 at 23:59

Total Marks: 32

Contents

1 General instructions:

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of 1-dimensional native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire specification before you start coding.
- You will be afforded five upload opportunities.

2 Plagiarism

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.

3 Outcomes

The aim of this practical is to implement a shortest path algorithm, a cycle detection algorithm and a depth first traversal algorithm for a directed graph with both positive and negative weights.

4 Introduction

Complete the task below. Certain classes have been provided for you alongside this specification in the Student files folder. A very basic main has been provided. **Please note this main is not extensive and you will need to expand on it**. Remember to test boundary cases. Submission instructions are given at the end of this document.

5 Task 1: Graph

A Graph is a collection of vertices and connections between them. The connections are known as edges. Each edge connects to a pain 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 should be able to handle unreachable nodes. **It can be assumed that the shortest path function will not be called on a graph that contains a cycle.** You will also be required to implement a cycle detection algorithm and a depth first traversal algorithm. You have been provided with a Vertex.java, Edge.java, Graph.java and main.java files. You may add functions to these files but do not change any of the provided function declarations. Please note that [] is square brackets.

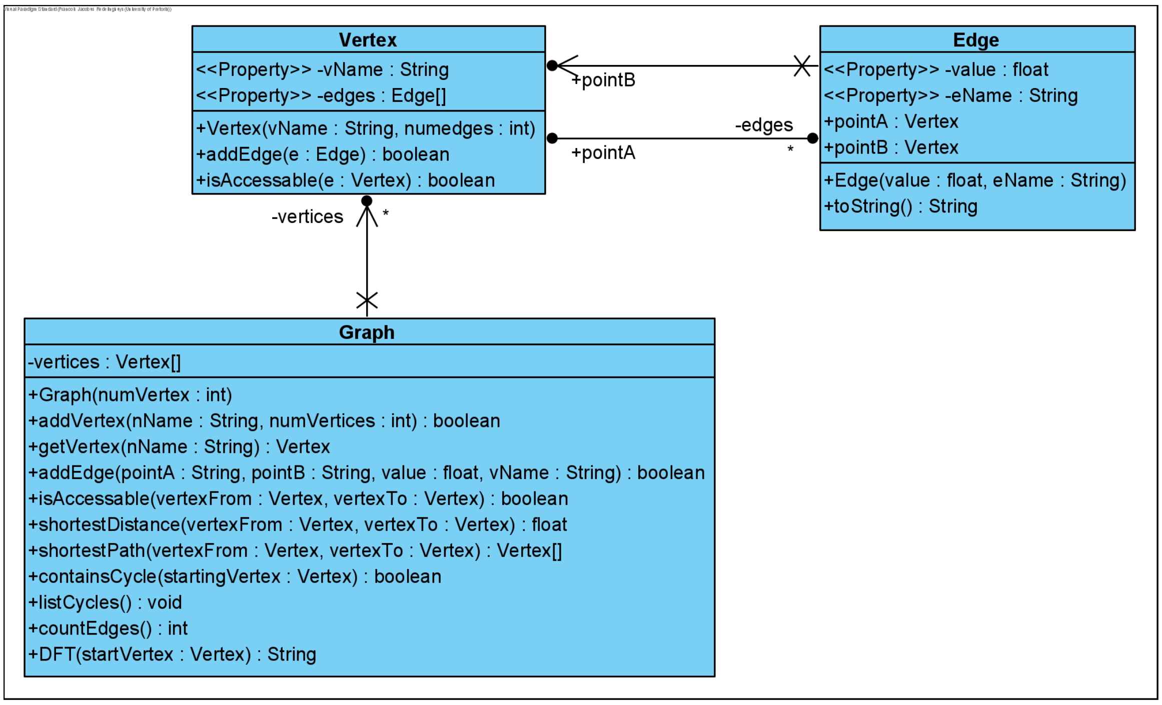
Vertex
<<Property>> -vName : String
<<Property>> -edges : Edge[]
+Vertex(vName : String, numedges : int)
+addEdge(e : Edge) : boolean
+isAccessable(e : Vertex) : boolean
Edge
<<Property>> -value : float
<<Property>> -eName : String
+pointA : Vertex
+pointB : Vertex
+Edge(value : float, eName : String)
+toString() : String
+pointB
-edges
+pointA
*
-vertices
*
Graph
-vertices : Vertex[]
+Graph(numVertex : int)
+addVertex(nName : String, numVertices : int) : boolean
+getVertex(nName : String) : Vertex
+addEdge(pointA : String, pointB : String, value : float, vName : String) : boolean
+isAccessable(vertexFrom : Vertex, vertexTo : Vertex) : boolean
+shortestDistance(vertexFrom : Vertex, vertexTo : Vertex) : float
+shortestPath(vertexFrom : Vertex, vertexTo : Vertex) : Vertex[]
+containsCycle(startingVertex : Vertex) : boolean
+listCycles() : void
+countEdges() : int
+DFT(startVertex : Vertex) : String

Figure 1: Complete class diagram

5.1 Graph

- Members:
  - vertices: Vertex[]
    - ∗ This is the array containing all the vertices in the graph and may contain nulls.
- Functions:
  - Graph(numVertex: int)
    - ∗ Please note this function has been provided.
    - ∗ **DO NOT CHANGE THIS FUNCTION**
    - ∗ This is the constructor for the graph.
  - addVertex(nName: String, numVertices: int): boolean
    - ∗ Please note this function has been provided.
    - ∗ **DO NOT CHANGE THIS FUNCTION**
    - ∗ This function will add a vertex to the first open position in the array.
    - ∗ If the vertex cannot be added the function will return false else it will return true.
  - getVertex(nName: String): Vertex
    - ∗ Please note this function has been provided.
    - ∗ **DO NOT CHANGE THIS FUNCTION**
    - ∗ This function will return the vertex with the same name as the passed in parameter.
    - ∗ If no vertex is found the function will return null;

- addEdge(pointA: String, pointB: String, value: float, vName: String): boolean
  - Please note this function has been provided.
  - **DO NOT CHANGE THIS FUNCTION**
  - This function will add an edge to the passed in vertexes.
  - Please note the direction of the edge is from pointA to pointB.
  - If the edge cannot be added the function will return false else it will return true.
- isAccessable(vertexFrom: Vertex, vertexTo: Vertex): Boolean
  - This function should determine if there is a connecting path from vertexFrom to vertexTo.
  - Note this path does not need to be the shortest path. Just any path.
  - If there is no path from vertexFrom to vertexTo then the function should return false else the function should return true.
  - Please note that the path connecting vertexFrom and vertexTo should start at vertexFrom and end at vertextTo.
- shortestDistance(vertexFrom: Vertex, vertexTo: Vertex): float
  - This function should determine the shortest path distance from vertexFrom to vertexTo.
  - If no path exists from vertexFrom to vertexTo then the function should return positive infinity.
  - The order of edges used in the shortest path should be the order the edges are placed in the vertex's edges array.
  - Given Example A and a vertexFrom of vertex 0 and vertexTo of vertex 4 the result will be the following: 3.0
- shortestPath(vertexFrom: Vertex, vertexTo: Vertex): Vertex[]
  - This function should determine the shortest path from vertexFrom to vertexTo.
  - If no path exists from vertexFrom to vertexTo then the function should return null.
  - The order of edges used in the shortest path should be the order the edges are places in the vertex's edges array.
  - The order of the vertices places in the vertex array should be the path from vertexFrom to vertexTo.
  - Given Example A and a vertexFrom of vertex 0 and vertexTo of vertex 4 the result will contain the following:
    {0,2,3,4}
- containsCycle(startingVertex: Vertex): boolean
  - This function should determine if there is a cycle starting at the passed in parameter.
  - If a cycle is detected then the function should return true else the function should return false.

- listCycles(): void
  - This function should print out all the cycles in the graph.
  - The order of edges used in the cycle detection algorithm should be the order of the edges that are places in the vertex's edges array.
  - The format of the cycle print out should be a dash(-) delimited list.
  - Each cycle list should be printed on a newline.
  - Given Example B the output will be as follows (n indicates newline):
    0-2-3-1-0n
- countEdges(): int
  - This function should return the number of edges in graph.
- DFT(startVertex: Vertex): String
  - This function should return the vertices as a semi-comma (;) delimited list in the order of a depth first traversal.
  - The order of edges used should be the order of edges placed in the vertex's edges array.
  - Once all the vertices reachable from the startingVertex has been visited, the remaining unvisited vertices should be visited in the order of the vertices array.
  - Given Example B and startingVertex of 0 the result will be as follows:
    0;2;3;1;4

5.2 Vertex

- Members:
  - vName: String
    - This is the name of the vertex.
  - edges: Edge[]
    - This is the array containing all the edges of the vertex.
    - The array can have elements that are null.
- Functions:
  - Vertex(vName: String, numedges: int)
    - Please note this function has been provided.
    - **DO NOT CHANGE THIS FUNCTION**
  - addEdge(e: Edge): boolean
    - Please note this function has been provided.
    - **DO NOT CHANGE THIS FUNCTION**
    - This function will add an edge to the first open position in the array.
    - If the edge is not able to be added the function will return true else the function will return false

  - isAccessable(e: Vertex): boolean
    * This function should determine if an path can be found from the current vertex to the passed in vertex
    * If a path cannot be established the function should return false else the function will return true.
    * *Hint: use this function as a helper function for Graph.isAccessable*

5.3 Edge

- Members:
  - value: float
    * This is the value of the edge connecting the two vertices
  - eName: String
    * This is the name of the edge
  - pointA: Vertex
    * This is the vertex from which the edge starts
  - pointB: Vertex
    * This is the vertex at which the edge ends.
  - pointA goes to pointB
- Functions:
  - Edge(value: float, eName: String)
    * Please note this function has been provided.
    * **DO NOT CHANGE THIS FUNCTION**
  - toString()
    * Please note this function has been provided.
    * **DO NOT CHANGE THIS FUNCTION**

6 Example figure:

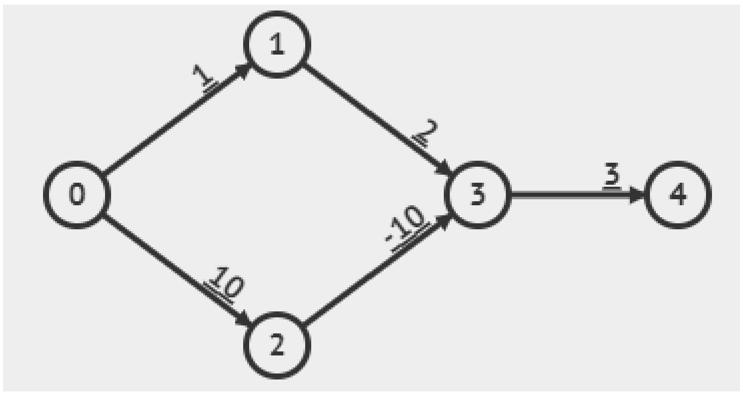

Figure 2: Example A

7 Example figure:

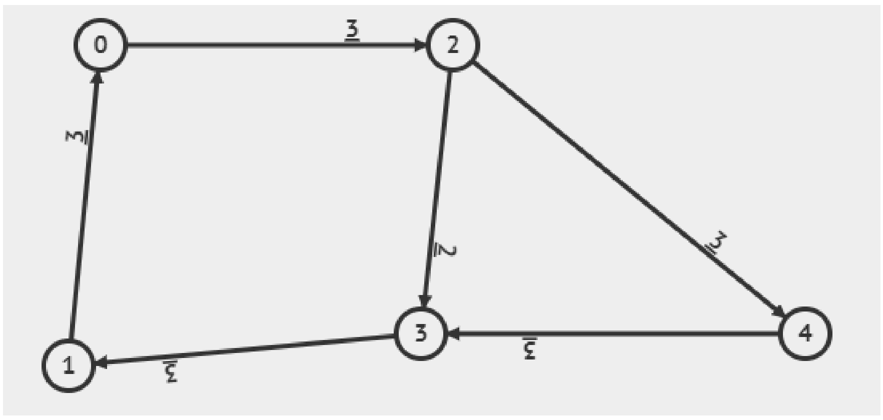

Figure 3: Example B

8 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. Place your Graph.java, Vertex.java, Edge.java file, as well as any custom classes you created, 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. Your code should be able to be compiled with the following command:
*make *.java*
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!**