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 8 Specifications: Topological Sorting, Elemental Sorting Algorithms

Release Date: 16-05-2022 at 06:00

Due Date: 20-05-2022 at 23:59

Total Marks: 40

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 topological sorting algorithm on a graph data structure as well as several elementary sorting algorithms.

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: Sorting algorithms

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 topological sorting algorithm on a directed graph data structure. Your algorithm should be able to handle unreachable nodes and cycles. You will also be required to implement several elementary sorting algorithms namely Insertion, Selection, Bubble and Comb sorting. You have been provided with a Vertex.java, Edge.java, Graph.java, BubbleSort.java, Sorting.java, InsertSort.java, CombSort.java, SelectionSort.java, TopologicalSort.java and Main.java files. You may only add functions to the specified files but do not change any of the provided function declarations. Please note that [] is square brackets.
Sort according to vertex names

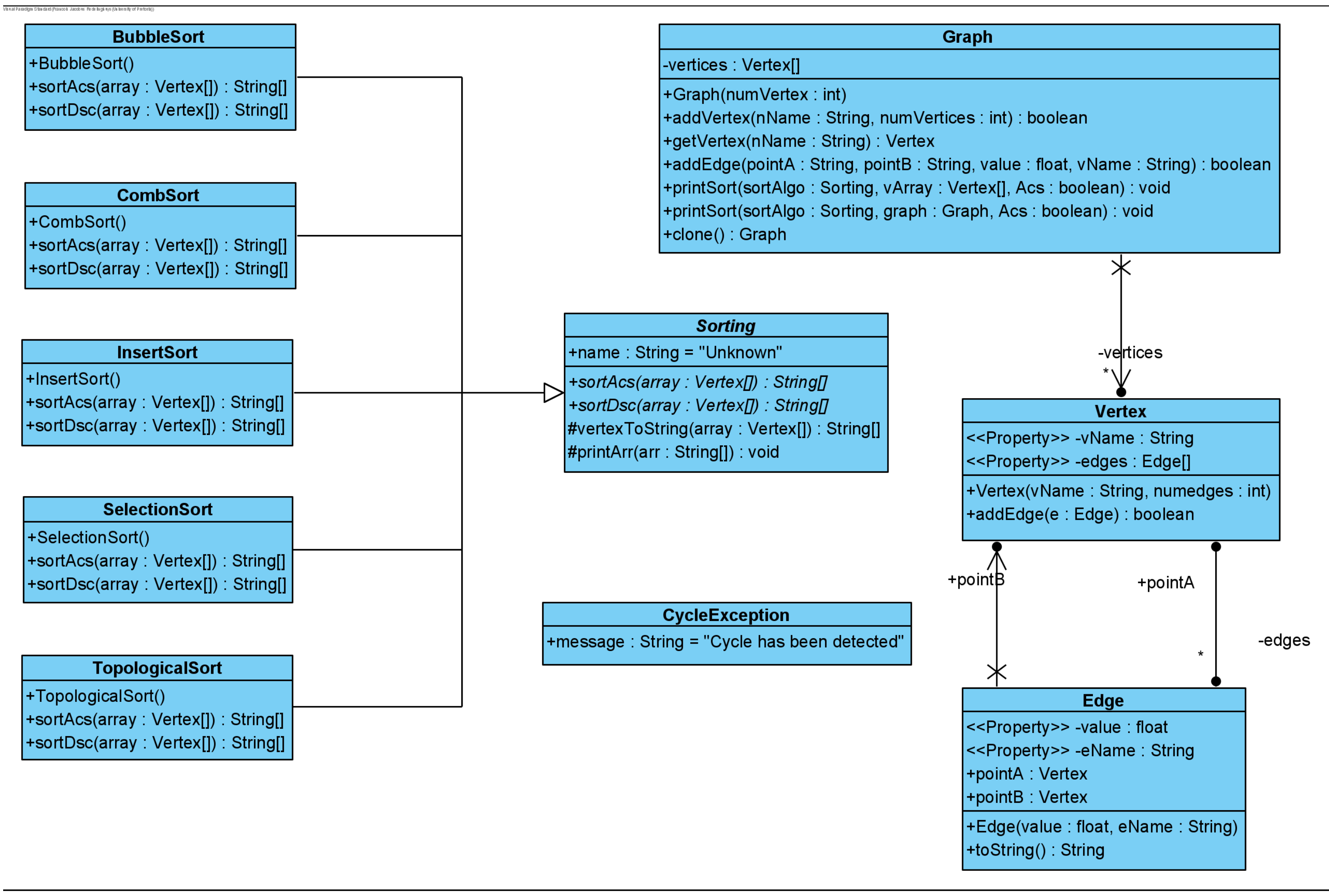
Visual Paradigm Standard(Francois Jacobus Redelinghuys(University of Pretoria))
BubbleSort
+BubbleSort()
+sortAcs(array : Vertex[]) : String[]
+sortDsc(array : Vertex[]) : String[]
CombSort
+CombSort()
+sortAcs(array : Vertex[]) : String[]
+sortDsc(array : Vertex[]) : String[]
InsertSort
+InsertSort()
+sortAcs(array : Vertex[]) : String[]
+sortDsc(array : Vertex[]) : String[]
SelectionSort
+SelectionSort()
+sortAcs(array : Vertex[]) : String[]
+sortDsc(array : Vertex[]) : String[]
TopologicalSort
+TopologicalSort()
+sortAcs(array : Vertex[]) : String[]
+sortDsc(array : Vertex[]) : String[]
Sorting
+name : String = "Unknown"
+sortAcs(array : Vertex[]) : String[]
+sortDsc(array : Vertex[]) : String[]
#vertexToString(array : Vertex[]) : String[]
#printArr(arr : String[]) : void
CycleException
+message : String = "Cycle has been detected"
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
+printSort(sortAlgo : Sorting, vArray : Vertex[], Acs : boolean) : void
+printSort(sortAlgo : Sorting, graph : Graph, Acs : boolean) : void
+clone() : Graph
-vertices
*
Vertex
<<Property>> -vName : String
<<Property>> -edges : Edge[]
+Vertex(vName : String, numedges : int)
+addEdge(e : Edge) : boolean
+pointB
+pointA
-edges
*
Edge
<<Property>> -value : float
<<Property>> -eName : String
+pointA : Vertex
+pointB : Vertex
+Edge(value : float, eName : String)
+toString() : String

Figure 1: Complete class diagram

5.1 Graph

- DO NOT CHANGE THIS CLASS AS IT WILL BE OVERWRITTEN

5.2 Vertex

- DO NOT CHANGE ANY OF THE PROVIDED FUNCTIONS
- You may add any helper members or functions that you need.

5.3 Edge

- DO NOT CHANGE THIS CLASS AS IT WILL BE OVERWRITTEN

5.4 Sorting

- vertexToString(array: Vertex[]): String[]
  - This is a helper function provided for you.
  - It takes in an array of type Vertex and returns an array of type String containing the vertices' names.
- printArr(arr: String[]): void
  - This is a helper function provided for you.
  - It will print the passed in array as a semi colon delimited list ending on a new line.
- sortAcs(array: Vertex[]): String[]
  - This is an abstracted function.
  - The implementations are described below.
- sortDsc(array: Vertex[]): String[]
  - This is an abstracted function.
  - The implementations are described below.

5.5 BubbleSort

- BubbleSort()
  - The constructor should initialize the name member to the following: BubbleSort
- sortAcs(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in ascending order using the bubble sort algorithm.
  - The function should also print out the partially sorted array.
  - Please see the pseudo code in the attached pdf document describing where the print should be placed.
- sortDsc(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in descending order using the bubble sort algorithm.
  - The function should also print out the partially sorted array.
  - Only the ascending pseudo code is given. You will need to modify this algorithm such that it will sort in descending order. Please note that the printing of the partially sorted array should still happen at the same place in the algorithm.

5.6 InsertSort

- InsertSort()
  - The constructor should initialize the name member to the following: InsertSort
- sortAcs(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in ascending order using the insertion sort algorithm.
  - The function should also print out the partially sorted array.
  - Please see the pseudo code in the attached pdf document describing where the print should be placed.
- sortDsc(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in descending order using the insertion sort algorithm.
  - The function should also print out the partially sorted array.
  - Only the ascending pseudo code is given. You will need to modify this algorithm such that it will sort in descending order. Please note that the printing of the partially sorted array should still happen at the same place in the algorithm.

5.7 SelectionSort

- SelectionSort()
  - The constructor should initialize the name member to the following: SelectionSort
- sortAcs(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in ascending order using the selection sort algorithm.
  - The function should also print out the partially sorted array.
  - Please see the pseudo code in the attached pdf document describing where the print should be placed.
- sortDsc(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in descending order using the selection sort algorithm.
  - The function should also print out the partially sorted array.
  - Only the ascending pseudo code is given. You will need to modify this algorithm such that it will sort in descending order. Please note that the printing of the partially sorted array should still happen at the same place in the algorithm.

5.8 CombSort

- CombSort()
  - The constructor should initialize the name member to the following: CombSort
- sortAcs(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in ascending order using the comb sort algorithm.
  - The function should also print out the partially sorted array and the gap size.
  - Please see the pseudo code in the attached pdf document describing where the print should be placed.
- sortDsc(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices in descending order using the comb sort algorithm.
  - The function should also print out the partially sorted array and the gap size.
  - Only the ascending pseudo code is given. You will need to modify this algorithm such that it will sort in descending order. Please note that the printing of the partially sorted array should still happen at the same place in the algorithm.

5.9 Topological sort

- TopologicalSort()
  - The constructor should initialize the name member to the following: TopologicalSort
- sortAcs(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices using the Topological sort algorithm.
  - The order of the names in the string array should be from lowest topological order to highest topological order.
  - If a cycle has been detected the function should throw the CycleDetection exception.
- sortDsc(array: Vertex[]): String[]
  - This function should return an array of string containing the names of the passed in vertices using the Topological sort algorithm.
  - The order of the names in the string array should be from highest topological order to lowest topological order.
  - If a cycle has been detected the function should throw the CycleDetection exception.

6 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. Only the following java files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- Graph.java
- Edge.java
- Vertex.java
- Sorting.java
- BubbleSort.java
- InsertSort.java
- CombSort.java
- SelectionSort.java
- TopologicalSort.java

. 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 8 slot on the Fitch Fork website. Submit your work before the deadline. **No late submissions will be accepted!**