Practical 7

COS212

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

Department of Computer Science

Deadline: 19/05/2023 at 23:59

Marks: 230

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 native arrays where applicable. If you require additional data structures, you must implement them yourself.
- If your code does not compile, you will be awarded a zero mark. 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 assignment before you start coding.
- You will be afforded five upload opportunities.
- Make sure your IDE did not create any packages or import any libraries which are not allowed.

2 Plagiarism

The Department of Computer Science considers plagiarism 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 permission) and copying material 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. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.** Also, note that the OOP principle of code reuse does not mean you should copy and adapt code to suit your solution.

3 Outcomes

Upon completing this assignment, you will have implemented the following:

- an undirected graph using Vertex and Edge classes.
- cycle detection using the union-find algorithm.
- a method for creating a minimum spanning tree from a graph.
- a method for graph colouring.

4 Background

Graphs are datastructures used to store data which is neither hierarchical nor ordered. It is used when the data has links which are not easily represented by other datastructures. Graphs are usually implemented using Vertex and Edge classes and then storing these in arrays. This practical will make use of this technique to implement a few graph algorithms.

5 Tasks

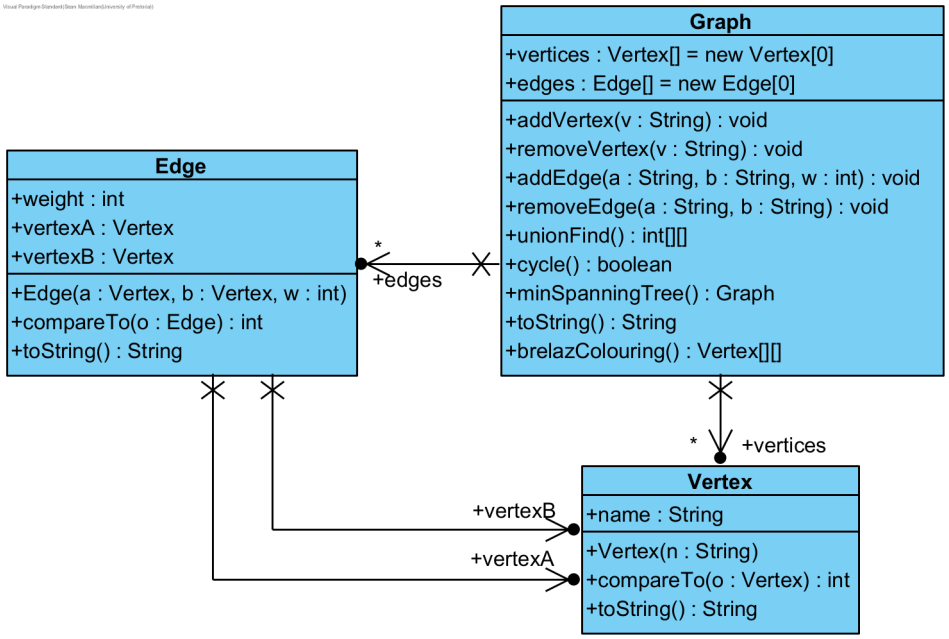

Figure 1: UML

5.1 Vertex

This class is given to you. You are allowed to add extra functions or members but don't change the given functions or members. **The compareTo() function has been overwritten. You can use this to compare the names of vertices with each other.**

5.2 Edge

This class is given to you. You are allowed to add extra functions or members but don't change the given functions or members. **A few important things to note about this class:**

- The vertices are sorted such that the "smaller" vertex is always stored in vertexA and the "larger" vertex is stored in vertexB.
- The compareTo() method is overwritten. This will first compare on vertexA, but if the vertexA's are equal it is compared on vertexB.

5.3 Graph

- Members
  - Vertex[] vertices
    - This is an array used to store the vertices.
    - **The array will always be sorted alphabetically.**
    - **The array will always be the correct size (i.e. there are no empty spaces)**
  - Edge[] edges
    - This is an array used to store the edges.
    - **The array will always be sorted using Edge.compareTo().**
    - **The array will always be the correct size (i.e. there are no empty spaces)**
- Functions
  - addVertex(v:String):void
    - This function attempts to add a vertex to the graph.
    - Duplicate values are not allowed in the graph. If a duplicate value is passed in, do nothing.
    - Ensure the rules described in the Members section are adhered to.
  - removeVertex(v:String):void
    - If the passed-in parameter is inside the graph, remove it.
    - Ensure the rules described in the Members section are adhered to.
  - addEdge(a:String,b:String,w:int):void
    - This function attempts to add an edge to the graph.
    - Duplicate values are not allowed in the graph. If a duplicate value is passed in, do nothing.
    - If either of the passed-in parameters doesn't correspond to a Vertex in the Graph, then do nothing.
    - *Note that the Edge constructor might swap vertexA and vertexB around. This is the intended behaviour.*
    - Ensure the rules described in the Members section are adhered to.
  - removeEdge(a:String,b:String):void
    - If the passed-in parameter is inside the graph, remove it.
    - Ensure the rules described in the Members section are adhered to.

- unionFind():int[][]
  - This function will implement the union-find cycle detection algorithm.
  - The function returns a 2D int array which is size $4 \times |V|$
  - You must use the algorithm in Section 10.1, which is a slight modification of the algorithm in the textbook.
  - The array that is returned is structured as follows: $\begin{bmatrix} \text{root} \\ \text{next} \\ \text{length} \\ \text{result} \end{bmatrix}$
  - root, next and length are the arrays from the Pseudocode.
  - Result is an int array which is the same size as the other arrays. If a cycle were detected, this would be an array filled with 1's. If no cycles were detected, this is an array filled with 0's.
  - Use the provided Main and Section 6 to see an example.
- cycle():boolean
  - Returns true if a cycle was found in the graph and returns false otherwise.
- minSpanningTree():Graph
  - This function must use the Kruskal algorithm for generating a minimum spanning tree.
  - The original graph should not be altered.
  - The function must return a graph which is the minimum spanning tree.
  - Use the provided Main and Section 6 to see an example.
- brelazColouring():Vertex[][]
  - This function will perform graph colouring on the current graph.
  - The Brelaz colouring algorithm should be used to achieve this.
  - This function will return a 2D vertex array. Every row will represent a different colour.
  - The resulting array must be perfectly sized. Thus the number of rows should be the number of colours. And the length of each row should be the number of nodes for that specific colour.
  - The rows should be sorted by the order in which colours were assigned. Thus if the colours created are $[c_0, \cdots, c_n]$, the 2D array should look like $\begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix}$, where c_j is the array of nodes with colour c_j.
  - Each colour should then be sorted in lexicographical order. Thus every row in the resulting array should be sorted in alphabetical order.
  - Use the provided Main and Section 6 to see an example.

- toString():String
  - This returns a String representation of the graph.
  - Use the provided Main and Section 6 to see how this should look.
  - The general format is shown below:

	vertices[0]	...	vertices[n-1]
vertices[0]	adjacencyMatrix[0][0]	...	adjacencyMatrix[0][n-1]
⋮	⋮	⋱	⋮
vertices[n-1]	adjacencyMatrix[n-1][0]	...	adjacencyMatrix[n-1][n-1]

Note that columns are separated by tabs.

6 Example

The provided Main creates a graph which looks as follows :

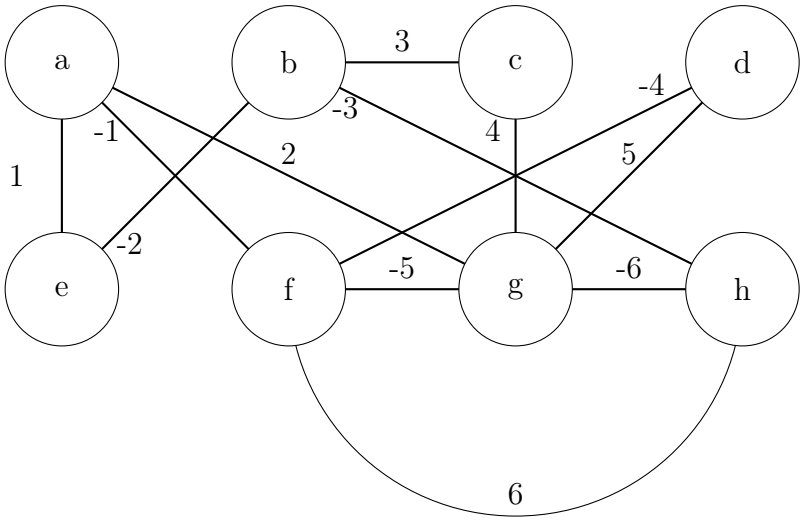

Figure 2: Graph from provided Main

The following subsections show the output when calling the functions on this graph.

6.1 toString

	a	b	c	d	e	f	g	h
a	0	0	0	0	1	-1	2	0
b	0	0	3	0	-2	0	0	-3
c	0	3	0	0	0	0	4	0
d	0	0	0	0	0	-4	5	0
e	1	-2	0	0	0	0	0	0
f	-1	0	0	-4	0	0	-5	6
g	2	0	4	5	0	-5	0	-6
h	0	-3	0	0	0	6	-6	0

6.2 unionFind

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 6 & 1 & 7 & 0 & 4 & 5 & 2 \\ 8 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

6.3 minSpanningTree

	a	b	c	d	e	f	g	h
a	0	0	0	0	0	-1	0	0
b	0	0	3	0	-2	0	0	-3
c	0	3	0	0	0	0	0	0
d	0	0	0	0	0	-4	0	0
e	0	-2	0	0	0	0	0	0
f	-1	0	0	-4	0	0	-5	0
g	0	0	0	0	0	-5	0	-6
h	0	-3	0	0	0	0	-6	0

This corresponds to the following graph:

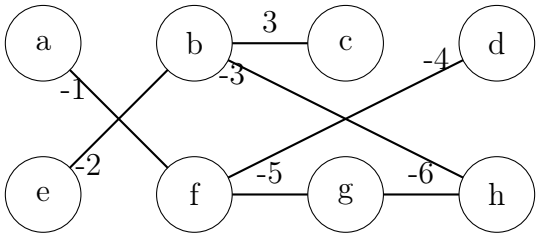

Figure 3: Minimum spanning tree

6.4 brelazColouring

$$\begin{bmatrix} b & g & \\ c & e & f \\ a & d & h \end{bmatrix}$$

This corresponds to the following graph:

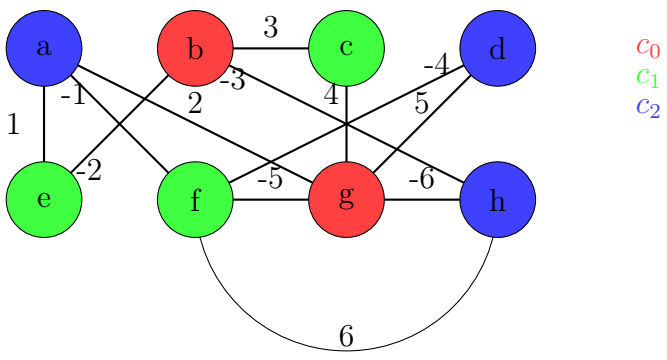

Figure 4: Graph Colouring

7 Submission instructions

Do not submit any custom function classes.
Your code should be able to be compiled and run with the following commands:
javac *.java
java Main

Once you are satisfied that everything is working, you must create an archive of all your Java code into one archive called uXXXXXXXX.{tar.gz/tar/zip} where X is your student number. Submit your code for marking under the appropriate link before the deadline. **Make sure your archive consists of only java files and no folders or subfolders.**

Please ensure that you thoroughly test your code before submitting it. Just because your code runs with local testing does not mean that your code works for every situation. **DO NOT USE FITCH FORK AS A DEBUGGER**

8 Submission checklist

- Vertex.java
- Edge.java
- Graph.java

9 Allowed imports

- No imports allowed

10 Psuedocode

10.1 Union-find

This algorithm uses three int arrays called root, next and length. These are the same length as the Vertices array, and the indexes correspond to each other. (i.e. vertices[3] 's root is saved in root[3], next is saved in next[3] and length is saved in length[3])

```
for (i = 0 to |V| - 1)
    root[i] = i
    next[i] = i
    length[i] = 1

for (Every edge in the graph)
    v = position of vertexA in Vertices array
    u = position of vertexB in Vertices array

    if (root[u] == root[v])
        A cycle was found.
        DO NOT RETURN
    else if (length[root[v]] < length[root[u]])
        rt = root[v]
        length[root[u]] += length[rt]
        root[rt] = root[u]
        for(j = next[rt];j != rt;j=next[j])
            root[j] = root[u]
        swap(next[rt],next[root[u]])
    else
        The same as the else if part, but with u and v reversed
```