ALGORITHMS & DATA STRUCTURES
Lahcen

TOPIC : GRAPHS
LAB. SUBMISSION

In this activity, you will work **individually** implementing a graph data structure, its main operations and the algorithms to find the MST and the shortest path between a pair of nodes. Remember: you can discuss ideas with your classmates, but you cannot see other's work neither others can see your individual work.

After finishing the implementation of your lab work, you must upload **a single file** with your code using the "Graphs Lab Submission" link available at learn.gold, Your code will be graded immediately.

The deadline for this lab submission is Friday 18th **of March 2022, 5pm**
(Please see VLE for updates on the submission dates).

PART 1: YOUR TASK

In this lab, your task is to implement a graph, its operations and the MST and Dijkstra's algorithm using the **adjacency list** representation.

Java programmers are provided with 3 files:

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

C++ programmers are provided with 6 files:

- Vertex.cpp and Vertex.hpp
- Edge.cpp and Edge.hpp
- Graph.cpp and Graph.hpp

Your task is completing the code that makes the methods in the skeleton code files operative. **You must not change the prototypes of the methods**. You can add other methods and variables if you want.

PART 2a: JAVA PROGRAMMERS

Please, look at the content of the skeleton file Vertex.java. There you will find only 1 method you need to implement:

- Vertex (name): This constructor creates a new vertex by just assigning it the name 'name' and creating a new instance of the adjacency list.

Please, look at the content of the skeleton file Edge.java. There you will find only 1 method you need to implement:

- Edge(from, to, weight): This constructor creates a new edge by assigning it an origin vertex (from), a destination vertex (to) and a weight.

Finally, please look at the content of the skeleton file Graph.java. You can see there are 9 methods you must implement in the .cpp file:

- Graph( ): This constructor creates a new object vlist (allocates memory to it).

- addVertex(v): This method creates a new vertex (using the constructor Vertex) named 'v' and appends it to the list of vertices (vlist).

- getVertex(v): This methods performs a linear search on vlist. If an element with a name equal to 'v' is found, the element (a vertex) is returned. If no vertex with name equal to 'v' is found in the list, NULL is returned.

- addEdge(v1, v2, weight): This method uses the method getVertex() to obtain the vertices identified by the names v1 and v2. Next, it uses the constructor Edge to create the new edge and adds it to the corresponding adjacency list. If vertices v1 and v2 are different, a second edge originating at v2 and ending at v1 must be added as well.

- getEdge(v1,v2): This method uses the method getVertex() to obtain the vertices identified by the names v1 and v2. Next, it performs a linear search on the corresponding adjacency list to check whether the edge exists. If so, it returns it. Otherwise, it returns NULL

- MST(): This method returns a graph, corresponding to the minimum spanning tree of the original graph. You can use Prim or Kruskal algorithms to solve this problem. Bear in mind that Kruskal requires you to implement a disjoint-set data structure as well.

- MSTCost(): This method returns the cost of the minimum spanning tree

- SP(v1,v2): This method returns a graph containing the route (sequence of vertices) of the shortest path from v1 to v2

- SPCost(v1,v2): This method returns the cost of the shortest path between v1 and v2

PART 2b: C++ PROGRAMMERS

Please, look at the content of the skeleton files Vertex.cpp and Vertex.hpp. In Vertex.cpp you will find only 1 method you need to implement:

- Vertex (v): This constructor creates a new vertex by just assigning it the name v and creating a new instance of the adjacency list.

Please, look at the content of the skeleton files Edge.cpp and Edge.hpp. In Edge.cpp you will find only 1 method you need to implement:

- Edge(from, to, weight): This constructor creates a new edge by assigning it an origin vertex (from), a destination vertex (to) and a weight.

Finally, please look at the content of the skeleton files Graph.cpp and Graph.hpp. You can see there are 8 methods you must implement in the .cpp file:

- Graph( ): This constructor creates a new object vlist.

- addVertex(v): This method creates a new vertex (using the constructor Vertex) and appends it to the list of vertices (vlist).

- getVertex(v): This methods performs a linear search on vlist. If an element with a name equal to v is found, the element (a vertex) is returned. If no vertex with name equal to v is found in the list, NULL is returned.

- addEdge(v1, v2, weight): This method uses the method getVertex() to obtain the vertices identified by the names v1 and v2. Next, it uses the constructor Edge to create the new edge and adds it to the corresponding adjacency list. If vertices v1 and v2 are different, a second edge originating at v2 and ending at v1 must be added as well.

- getEdge(v1,v2): This method uses the method getVertex() to obtain the vertices identified by the names v1 and v2. Next, it performs a linear search on the corresponding adjacency list to check whether the edge exists. If so, it returns it. Otherwise, it returns NULL

- MST(): This method returns the minimum spanning tree of the original graph

- MSTCost(): This method returns the cost of the minimum spanning tree

- SP(v1,v2): This method returns a graph containing the sequence of vertices of the shortest path from v1 to v2

- SPCost(v1,v2): This method returns the cost of the shortest path between v1 and v2

PART 3: SUBMISSION

For this lab submission your code will automatically graded. As soon as you upload your submission, your code will be analysed and a report will be displayed on the screen. The report includes your mark in the assignment as well as any errors you still might need to correct.

You can upload your submission as many times as you want, without any block window. Your highest score will be considered.

You can get familiar with the system simply uploading the skeleton code provided to you. In that case, you will get a mark of 39. To do so, you must:

- select the language of your submission (Question 1)
- upload the file (Question 2). In the case of C++ programmers, you must compress the .cpp and .hpp files and upload the resulting .zip file
- answer the last personalised question (Questions 3). The code you must insert here is only made of addVertex and addEdge operations.

As soon as you press the button "Submit" you will see the message "Verifying your answers….".

Because none of the methods is implemented in the skeleton code yet, you will be next shown a red screen (signalising a mark under 40) with the following message: "There are some errors in your answer. Your score is 39%".

Below the red screen, you will receive a report with the tests that your code failed.

After implementing each method, submit again and check that the number of failing tests has decreased.