ALGORITHMS AND DATA STRUCTURES 2

TOPIC 10: GRAPHS
FORMATIVE EXERCISE

In this activity, you will work **individually** to implement a graph.

PART 1: YOUR TASK

In this formative exercise, your task is to implement a graph using the adjacency list representation.

This exercise will be automatically graded. Thus, you are provided with six files:

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

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

PART 2: THE METHODS

Please look at the content of the skeleton files Vertex.cpp and Vertex.hpp. In Vertex.cpp you will find only one 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 one 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 nine methods you must implement in the .cpp file:

- 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 a 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

When you want to submit your work, please compress all files (.cpp and .hpp) into a zip file. The compressed file can have any name.

In the My submission tab, press the blue 'Create Submission' button. Below the text 'Upload Files and Submit', press the blue 'Graphs' button, select the file and click 'Submit'.

Upon submitting, you will see the date of the submission in grey. It might take a while for the system to grade your code. You can wait for the grade or log out and come back later.

Once the grade is ready, you can click on the submission tab and then on 'Show grader output' to check on eventual errors.

You can upload your submission as many times as you want. Your highest score will be recorded.

You can get familiar with the system by simply uploading the skeleton code provided for you. Naturally, all tests will fail in this case. Next, attempt to implement the methods, one by one. After each step, run the tests once more, and check that the number of failing tests has decreased.