EECS 560 Lab 10: Graph (undirected and without multi-edges)

Objective

- Get familiar with the adjacency list implementation of graph using C++.
- Get familiar with graph traversal algorithms such as BFS and DFS.
- Integrate the previously implemented data structures.

Specification of the ADT

1. Implement a graph class called `MyGraph`. The data structure should use adjacency list as internal implementation.
2. The first-dimensional of the adjacency list should be implemented using `MyVector`; the second-dimensional of the adjacency list should be implemented using `MyLinkedList`.
3. The `MyGraph` class should allow vertex insertion using function `VertexIDType addVertex(const VertexDataType& v_data)`, which returns an ID assigned to the vertex and allows for the retrieval of the vertex (and its data) in the future. The `MyGraph` class should also allow the retrieval of a vertex using its assigned ID with the function `Vertex* getVertex(const VertexIDType vid)`. Note that the ID should be tied to the vertex throughout its lifespan and should not be duplicated or modified.
4. The `MyGraph` class should further allow the deletion of a vertex with the function void `deleteVertex(const VertexIDType vid)`. When deleted, the data of the vertex and its assigned ID should both be destructed. Swap the vertex to be deleted with the last vertex in the adjacency list, in this case its size is exactly the number of remaining vertices in the graph rather than all vertices that have ever been inserted. Note that you should also delete all edges associated with the vertex.
5. Do the same for edges. The only exception is that when deleting an edge, you don't need to delete its associated vertices.
6. Implement the BFS algorithm with the function `void breadthFirstSearch(const VertexIDType v_src, MyVector<VertexIDType>& path)`. We recommend that you use the `MyQueue` class the facilitate the implementation.
7. Implement the DFS algorithm with the function `void depthFirstSearch(const VertexIDType v_src, MyVector<VertexIDType>& path)`. We recommend that you use the `MyStack` class the facilitate the implementation.
8. We recommend that you change the private members in `MyVector` class to protected, which can make your implementation easier.

Hint

All edge weights are assumed to be positive.

Testing and Grading

We will test your implementation using the tester main function posted online. The posted input and output examples should be used for a testing purpose, while we will use another set of inputs for grading. Your code will be compiled under Ubuntu 20.04 LTS using g++ version 9.3.0 (default) with C++11 standard.

Your final score will be the determined by the success percentage of your program when fed with many random inputs. Note that if your code does not compile (together with our tester main function), you will receive 0. Therefore, it is very important that you ensure your implementation can be successfully compiled before submission.

Submission and Deadline

Please submit your implementation as six .h files, with names "MyVector_[YourKUID].h" (Lab01), "MyLinkedList_[YourKUID].h" (Lab 02), "MyQueue_[YourKUID].h" (Lab 03), "MyStack_[YourKUID].h" (Lab 03), "MyHashTable_[YourKUID].h" (Lab 05), and "MyGraph_[YourKUID].h". For example, if my KU ID is c123z456, my submission will be three files named "MyVector_c124z456.h", "MyLinkedList_c124z456.h", "MyQueue_c124z456.h", "MyStack_c124z456.h", "MyHashTable_c124z456.h", and "MyGraph_c124z456.h". Submissions that do not comply with the naming specification will not be graded. All submission will go through Blackboard. The deadline is Friday Dec 8th, 2022, 11:59PM.

!!! Important note!!!
Because of the stop day for the Fall 2022 semester is Dec 9th, the deadline set on Dec 8th is a hard deadline. It means no extension is possible; and that Blackboard submission will close exactly on Dec 8th end of day.