EECS 560 Lab 10: Graph 1

Due date
11/8/2020 Sunday, 11:59 pm

Objective
Get familiar with adjacency list representation of graph with C++. Work on graph traversal using **B**readth-**F**irst **S**earch (BFS) and **D**epth-**F**irst **S**earch (DFS).

Requirements to completed from text from template

1. Create the graph class with name "**myGraph**".
2. Choose your backend data structure to store the necessary data for random access the vertices.
3. Choose your backend data structure to store the adjacency list. **Note that** you must choose the backend data structure wisely so that the actual running time and memory usage doesn't exceed the allowed limit. See **Testing and Grading** for more details.
4. Implement default constructor **myGraph(std::string filename)** that create the graph object using the provided instance file with name **filename**. The instance file has format: $s_1,t_{11},s_1,t_{12},...,s_2,t_{21},s_2,t_{22},...,$ (separated by comma), where each tuple (**s**,**t**) stands for one directed edge from source vertex **s** to target vertex **t**.
5. Implement default destructor **~myGraph()** that frees any allocated memory if there is any.
6. Implement public member functions **void print()** that print the adjacency list using **cout** in the format:
   <s_1>: <t_{11}> <t_{12}> ...
   <s_2>: <t_{21}> <t_{22}> ...
   ...
   where targets are separated by space ' ', don't leave space after the last target. Note that there is a string '**:** ' after the source.
7. Implement public member functions **void BFS(int source)** that traverses the graph starting from vertex with index **source** using BFS algorithm and prints all reached vertex index using **cout** (one index per line). Refer to Figure 9.16 to implement your function. Note you only print each vertex once.
8. Implement public member functions **void DFS(int source)** that traverses the graph starting from vertex with index **source** using recursive DFS algorithm and prints all reached vertex index using **cout** (one index per line). Follow Figure 9.61 to implement your function. Note you only print each vertex once.
9. Any other internal (private) functions you need to implement the previous functions.

Testing and Grading

We will test your implementation using a tester main function, on a number of instances that are randomly generated. We will release the tester main function, several instances (will be different from the grading instances but will have the same format), and expected output together with the lab instruction via Blackboard. You code will be compiled under Ubuntu 20.04 LTS using g++ version 9.3.0 (default) with C++11 standard.

The command line we are going to use for compiling your code is: "g++ -std=c++11 main.cpp" (note that main.cpp will try to include the .hpp file you submit, and your .hpp file needs to be property implemented to compile successfully).

Your final score will be the percentage your program passes the grading instances. 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.

Your program will be considered "fail" for the cases if any of the running time or maximum resident set size exceeds 5 times of the provided benchmark file.

Submission and Deadline

Please submit your implementation as one .hpp file, with file name "myGraph_[YourKUID].hpp". For example, if my KU ID is c123z456, my submission will be a single file named "myGraph_c124z456.hpp". Submissions that do not comply with the naming specification will not be graded. Please submit through Blackboard.