CSCI 3100, Fall 2018

Homework 6 [100 points]

Overview

This homework focuses on the Shortest Path algorithms. We talked about three 'single source' shortest path algorithms: brute force (check all possible paths and pick the shortest one), Bellman-Ford, and Dijkstra's algorithms. I coded up one of these algorithms and created a shared library called `shortest_path`. Your task is to determine, which algorithm I coded up.

Since we know that Dijkstra's algorithm only works on non-negative edge weights, I constrained my `shortest_path` library to only work on graphs with non-negative edge weights. That way, you can't use 'edge weights' as the differentiating factor. This leaves you with using 'run time' to determine which algorithm is contained in the `shortest_path` library.

In your 'git' repositories, you will find a directory named `'runShortestPath'`. This directory has the following structure and files:

```
build/
include/
lib/
CMakeLists.txt
GraphGenerator.cpp
RunShortestPath.cpp
```

The `'lib'` directory contains `libshortest_path.so`, a C++ library with the shortest path algorithm.

The 'include' directory contains the following header files that you will need:

```
Edge.h
Graph.h
ShortestPath.h
```

Using these files along with `libshortest_path.so`, you will be able to construct various graphs and run shortest path algorithms on them.

As an example of how to use `Graph.h` and `ShortestPath.h`, I added `RunShortestPath.cpp` file. It uses a graph generator that I wrote (`GraphGenerator.h/GraphGenerator.cpp`). You are welcome to use that code if you wish. However, if you wish to make your own graph generator, it is up to you.

Finally, `CMakeLists.txt` file is a file used by `'cmake'` to generate makefiles. Again, you are welcome to use it or you can create your own `Makefile`. Here is how you would use `CMakeLists.txt`:

```
cd build
cmake ..
make
```

This will generate an executable called `run_shortest_path` in the `'bin'` subdirectory of `runShortestPath` directory.

To determine which algorithm I coded up, you will need to run this algorithm on graphs of various sizes (varying in the number of vertices and edges), time how long it takes to run it and make an educated guess based on the run-time complexity you observed for the input sizes you provided.

Your Submission

Submit a report with your analysis on this problem and your conclusion about which algorithm is contained in the shortest_path library.

Overview – brief overview of what the report is about.

Approach – describe how you determined which algorithm it is. Include relevant data: graph sizes (number of vertices and edges) and how long it took for the algorithm to find the shortest path. Include any relevant plots you created for this data.

Conclusion – state your conclusion based on the data you observed.

Submit a typed-up report via email or in class.