Project 2C: Experimenting with Graph Algorithms

## General Notes

* Submit all source code, program output, and project report electronically on Canvas as a Zip file.
* Allowed programming languages: Java, C++, Python, or others *with prior approval from the instructor*.

## Academic Honesty Policy: For this project, you are not allowed to discuss with others. Also, you are not allowed to copy things from online resources, e.g., source code. We will use a plagiarism detection tool on all submitted source code. It is capable of finding portions of copied code from other students in the course, other sections of the course, previous offerings of the course, as well as Internet sources.

## Instructions

In this project you will implement one graph algorithm and then analyze the performance of the algorithm on a bunch of random graphs. You will then attempt to infer the complexity of the algorithm and provide a report describing the algorithm’s observed performance and a comparison to the theoretical/expected **performance**.

This project focuses on the ***min-hop path finding problem*** where a directed graph G containing multiple airports and their direct-flight-links is given, and our task is to find the min-hop path between two particular airports, ***s*** (start) and ***t*** (destination). To solve this problem, you need to use the BFS algorithm (refer to Chapter 22 lecture ppt on Canvas) on G taking ***s*** as the start node.

***Min-hop path finding problem***: A **directed** graph ***G*** is given which represents a network of airports and the **direct** flight links. Like the real-life situation, there may or may not be direct flight available between a pair of airports. Now, for two particular airports, ***s*** (start) and ***t*** (destination) we need to find the min-hop path. The output will be a sequence of airports (if any), starting with ***s*** and ending with ***t***. Note that in this problem we need to optimize **only the hop count**, not the actual flight distance or travel time.

To test your algorithm, you need to write a helper program to generate four random graphs (with increasing size). ***G1***: 50 nodes (i.e., airports) and 100 edges (i.e., direct flight links), ***G2***: 100 nodes and 400 edges, ***G3***: 200 nodes and 1600 edges, ***G4***: 400 nodes and 3200 edges. In each of the graphs, we assume that the source ***s*** and terminal ***t*** are present. For each of the graphs, you find the path (**if a path exists**) between **s** and **t.**

Note that you cannot use any library function to implement the graph, i.e., you need to implement the *Adj list* representation of a graph on your own. Similarly, you cannot use any BFS or min-hop or min-distance algorithm implementation from any library.

The emphasis of the project is on experimental analysis of running time of the solution. Track the running time and try to record it in units that are fine-grained enough to allow meaningful comparison. For example, the unit for time should be micro- or possibly even nano-seconds on fast machines. Moreover, your algorithms will be tested for correctness, so please ensure they give correct output!

## Expected Input Format and Test data

To test your algorithm, you need to write a helper program to generate four random graphs (with increasing size). ***G1***, ***G2***, ***G3***, ***G4***. **Make sure that the edges are generated randomly**. Use the ***adjacency list*** representation to implement/realize the graphs. In each of the graphs, you randomly choose the source s and destination t from the set of nodes of the graph.

**Expected Output Format**

For each of the above graphs, your code will find and output the min-hop path between ***s*** and ***t***. For each of the four graphs, the output should contain the following: (1) min-hop path between ***s*** and ***t***, and (2) time taken by the algorithm. If there are multiple min-hop paths between s and t, then your program just outputs one of them. In some case, there may not be any path between **s** and ***t****,* and in that case your program will indicate that by outputting “none” or so. To compute “time taken” by the algorithm, make sure your script randomly chooses many (say 25) pairs of s and t and report the average "time taken".

## What to Submit

You should submit an archive (ZIP, TAR/GZip, etc) of all files, including the following:

1. Source code of the solutions.
2. Makefiles (or simple scripts) to compile the source code and run on all data.
3. A README.txt file including instructions on how to compile and run your source code.
4. A project report (PDF or Word) containing (at a minimum) the following:
   1. Runs showing output (the results, including the running time) for each input data.
   2. Theoretical performance (i.e., order of running time) for the algorithm.
   3. Are the theoretical and actual performance results consistent?
   4. Analysis and Discussion of ***your results*** vis-à-vis expectations.