CS590 homework 6 – Graphs, and Shortest Paths

The due date for this assignment is Wednesday, May 8th, at 11.59pm. This assignment is worth 10% of your final grade.

Any sign of collaboration will result in a 0 and being reported to the Graduate Academic Integrity Board. Late submission policy described in the syllabus will be applied.

Develop a data structure for directed, weighted graphs $G = (V, E)$ using an adjacency matrix representation. The datatype `int` is used to store the weight of edges. `int` does not allow one to represent $\pm\infty$. Use the values `INT_MIN` and `INT_MAX` (defined in `limits.h`) instead.

```
#include <limits.h>

int d, e;

d = INT_MAX;
e = INT_MIN;

if (e == INT_MIN) { ... }

if (d != INT_MAX) { ... }
```

1. Develop a function `random_graph` that generates a random graph $G = (V, E)$ with n vertices and m edges taking the weights (integers values) at random out of the interval $[-w, w]$. In order to have a more natural graph we generate a series of random paths $v_0, v_1, \ldots, v_k$ through the graph. Each of the individual paths has to be non-cyclic. The combination of the random paths might contain a cycle. The edges used in the series of random paths has to add up to the desired m edges.

   Notes:

   - Determine the number of maximal allowed edges for a non-cyclic random path $v_0, v_1, \ldots, v_k$.
   - How do you ensure that the path is random? A random permutation of the vertices might be useful.
   - How do two random path that cross each other (share one or more edges) effect the overall edge count? Shared edges should be counted only once.

2. Describe (do not implement) how you would update the above implemented `random_graph` method to generate a graph $G = (V, E)$ that does not contain a negative-weight cycle. You are given a function that can determine whether or not an edge completes a negatice-weight cycle.

3. Implement the Bellman-Ford algorithm. What is the running time for Bellman-Ford using an adjacency matrix representation?

4. Implement the Floyd-Warshall algorithm. Your implementation should produce the shortest weight matrix $D^{(n)}$ and the predecessor matrix $\Pi^{(n)}$. Limit the number of newly allocated intermediate result matrices.

(1. 30pts, 2. 20pts, 3. 25pts, 4. 25pts)

Remarks:

- You are not allowed to use code from online resources. Your submission will be tested against that, and will receive a 0, and a report to the Graduate Academic Integrity Board if it is detected.
- A Makefile is provided to build the code in LinuxLab.
- The programming, and testing will take some time. Start early.
- Feel free to use the provided source code for your implementation. You have to document your code.
- A pdf report is required for the solution of problem 2.