CS590 homework 6 – Graphs, and Shortest Paths

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:

Solution to Problem 2:

To ensure that the generated graph does not contain a negative-weight cycle, you can modify the random_graph method by checking for potential negative cycles after adding each random edge. If adding a new edge creates a negative cycle, you can simply remove the edge and try another random edge until a valid graph is formed.

Here's a high-level description of the modified approach:

1. Generate a random edge as usual.
2. Check if adding the edge creates a negative-weight cycle using the provided function.
3. If a negative cycle is detected, remove the edge and go back to step 1.
4. Repeat steps 1-3 until the desired number of edges m is reached.

This modification ensures that the generated graph is free from negative-weight cycles. Keep in mind that this approach may result in a longer execution time as it involves additional checks and removals.

Solution to Problem 3

Running Time of Bellman-Ford using an adjacency matrix:

The running time of the Bellman-Ford algorithm using an adjacency matrix representation is O(V * E), where V is the number of vertices and E is the number of edges in the graph. This is because the algorithm performs V-1 passes over all E edges, relaxing each edge in each pass. The negative cycle detection step may require additional passes in case of a negative-weight cycle.