Code Challenge
Notes about the Solution

Task 1

- **Running**:
  - To use the program, please run the following in the terminal in the task1 directory:
    ./task1.sh [adj-matrix] [root-node]
    Example: ./task1.sh "0,1,1,0,0,0|0,0,1,1,0,0|0,0,0,1,1,1|0,0,0,0,1,0|0,0,0,0,0,1|0,0,0,0,0,0" 0
    This passes an adjacency matrix for a graph of 6 nodes where 0 is the root node.
  - You could need to add executable permission to the file using: **chmod +x task1.sh**.
  - To run the tests, please run the following in the terminal in the task1 directory:
  - **./test_graph.sh**
  - You could need to add executable permission to the file using: **chmod +x test_graph.sh**.
  - Of course we can modify the program to use options like ./task1.sh – adj-matrix [adj-matrix] –root-node [root-node] but I wanted to provide a minimum viable product. However I used options in my bash script to show help about the program using:
    ./task1.sh -h
- **Used algorithm**: There are many researchers who solved this problem using different algorithms aiming for better resources optimized solutions. The algorithms also differ in their support for negative weighted edges (e.g. among the algorithms listed in the table below, Bellman–Ford algorithm is the best for this purpose).
- Some algorithms:

Algorithm	Time Complexity
Bellman–Ford Algorithm	O(VE)
Dijkstra's Algorithm	O(E + VLogV)
Topological Sorting algorithm	O(V + E)

  Where V and E represent the number of vertices and edges in the graph, consecutively.
  Reference: Lecture notes from the University of California (https://www.ics.uci.edu/~eppstein/161/960208.html).
- Since the sheet is not asking for providing support for negative weights parameters, I went for solving the question using the **Topological Sorting** algorithm to reduce the time complexity of my solution. Otherwise, I would have implemented my solution to support multiple algorithms and checked the input parameters to determine if the weights contain negative values. In this case, it's important to notify the user about the used algorithm.
- The graph class supports weighted edges for future development. However, the input adjacency matrix can contain only ones and zeros.
- Since the input could represent mistakenly a non-DAG, the solution also checks if the provided adjacency matrix represents a DAG (directed acyclic graph) using DFS algorithm. If it's not, the program shows an error message and terminates. If the question is not strictly mentioning DAGs we can also generalize the solution to use the convenient algorithm according to the graph type, but the current solution is enough as a minimum viable product.
- **Python3** is used to solve this task.

- One of the most powerful libraries to solve graph problems is the **networkx** library and it includes its built-in functions to check if the graph is a DAG and to find the shortest path. It's also a very easy and useful tool to visualize the graph. In all circumstances, it's always a **very good practice to check our solution by comparing** the results with ones from existing common libraries.

Task 2

- **Running**:
  - To use the program, please run the following in the terminal in the task2 directory: **./script.sh**
  - You could need to add executable permission to the file using: **chmod +x script.sh**.
- In case of production, we should replace "RUN npm install" in the Dockerfile with "RUN npm ci –only=production". In all circumstances, we separate the composition files of local, staging and development in real life applications to apply best practices for CI / CD.

Task 3

- **Running**:
  - To use the program, please run the following in the terminal in the task3 directory: **./script.sh**
  - You could need to add executable permission to the file using: **chmod +x script.sh**.
- The solution provided is for a small to med-size project. For a state-of-art solution and for a robust and secure connection between containers / microservices, I would go for using Istio on the top of RedHat OpenShift for containers orchestration using Kubernetes.
- For pinging Server-A from Server-B, we can get more details about the status of Server-A using its "/ping" route which I added as extra work.
- We must use SSL in production while connecting to the DB from Server-A.
- I used setTimeout to allow servers to wait for 3 seconds before pinging and checking connection. There are other ways to force our servers to wait. E.g. using a bash script.

Thank you so much.
Best regards,
Omar