Documentation - Practical Work No. 3

Problem Statement:

(6) Write a program that, given a graph with costs and two vertices, finds a lowest cost walk between the given vertices, or prints a message if there are negative cost cycles in the graph. The program shall use the matrix multiplication algorithm.

The function implementing the solution is:

```
matrix_multiplication_shortest_path(self)
```

The function was implemented in the Graph class, and it allows the user to input a pair of vertices in order to find the shortest path, raising an exception in the case of negative cycles.

Inside the {body} of the function, there are two methods:

1. `find_all_paths(weights, previous_next_vertex)`

Parameters:

- `weights` :: matrix containing the weights
- `previous_next_vertex` :: matrix used for determining the path between the previous vertex and the current one; is used in computing the next vertex on the desired path from (start) to (destination)

Modus Operandi:

- Using the matrix multiplication algorithm in order to raise the weights matrix to the power of $2^{log(n-1)}$, this is much faster than the normal linear multiplication in order to get to the power of n-1; that value is also much bigger, though there should be no issue with it, due to the fact that distance matrices do not register any changes past the power of n, unless there are negative cycles (which is going to raise an error in the parent function)

Returns:

- `intermediary_matrices` :: list of intermediary matrices
- `new_next_vertex` :: matrix that indicates the next vertex on the desired path

2. `extend_matrix(previous_distance, weights, previous_next_vertex)`

Parameters:

- `previous_distance` :: matrix that contains the old distance
- `weights` :: matrix that contains the weights
- `previous_next_vertex` :: matrix used for determining the path between the previous vertex and the current one; is used in computing the next vertex on the desired path from (start) to (destination)

Modus Operandi:

- In “multiplying” matrices, this method basically extends the previous distances, using the weights matrix. It computes the next vertex on the path.

Return:

- `new_distance` :: matrix containing the resulting distance matrix
- `new_next_vertex` :: matrix indicating the next vertex

The method `matrix_multiplication_shortest_path`() computes the adjacency matrix of the graph, and computes the possibility of the graph having negative cycles by multiplying the matrix of distances by itself, and in the eventuality that the result is different than what we started with, it raises an exception (ergo, we could use logarithmic exponentiation).

Additionally, the method allows the user to save the resulting path, the intermediary matrices, as well as the initial graph.