Floyd-Warshall Algorithm

Given a directed or an undirected graph with V vertices, the task is to find d_{ij}, the length of the shortest path between each pair of vertices i and j. The algorithm also detects negative cycle. If there exists a vertex u such that $d_{uu} < 0$, i.e., the distance from u to itself is negative, then the graph has a negative cycle.

We number the vertices as 1 to V and maintain a matrix, $d[][]$ for storing the value of d_{ij}.

The algorithm has V-phases. After the n-phase, $d[i][j]$ for any vertices i and j will have the length of the shortest path between i and j, considering only the set of vertices $\{1, 2, \ldots, n\}$.

For the 0-th phase, $d[i][j] = weight_{ij}$, if there exists an edge between i and j. Otherwise, $d[i][j] = \infty$, which is some large number.

In every phase, for every pair of vertices i and j, we minimise the value of $d[i][j]$ as a minimum over $(d[i][n] + d[n][j])$. There are V phases and since there are V^2 pairs of vertices, every phase requires $O(V^2)$ time.

Below is a implementation of the algorithm in `floydWarshall.cpp` which uses a adjacency matrix instead of a list. It takes the `weightedGraph` class object as a parameter defined in the `graph.hpp`.

```cpp
// floydWarshall.cpp

#include "graph.hpp"
using ll = long long int;
const ll INF = 0x3f3f3f3f3f3f3f3f;

/**
 * @brief Finds the lengths of shortest paths between
 * all pair of vertices using Floyd-Warshall's algorithm.
 * Prints "Graph has a negative cycle" if a negative cycle
 * exists.
 *
 * @param G The weighted Graph object G.
 * @return std::vector<std::vector<ll>> A vector of vectors
 * container storing the result. d[0][0] = 0 if graph has a
 * negative cycle, otherwise 1.
 */
std::vector<std::vector<ll>>
floydWarshall(const weightedGraph &G) {

    // Number of vertices
    int V = G.getV();

    // initialize d[i][j] to INF
    std::vector<std::vector<ll>>
        d(V + 1, std::vector<ll> (V + 1, INF));

    for (Vertex i = 1; i <= V; ++i)
```

```
        d[i][i] = 0LL;

    // If there is an edge between i and j,
    // set d[i][j] = weight
    for (Vertex u = 1; u <= V; ++u) {
        const std::vector<Edge>& adj = G.getAdj(u);
        for (Edge e : adj) {
            d[u][e.first] = std::min(d[u][e.first],
                                     (ll)e.second);
        }
    }

    for (Vertex n = 1; n <= V; ++n) {
        for (Vertex i = 1; i <= V; ++i) {
            for (Vertex j = 1; j <= V; ++j) {
                d[i][j] = std::min(d[i][j],
                                   (d[i][n] + d[n][j]));
            }
         }
    }

    for (Vertex u = 1; u <= V; ++u) {
        if (d[u][u] < 0) {
            std::cout << "Graph has a negative cycle\n";
            d[0][0] = 0;
            return d;
        }
    }
    d[0][0] = 1;
    return d;
}
```

Therefore, for this algorithm,

- Time complexity: $O(V^3)$.
- Space complexity: $O(V^2)$.