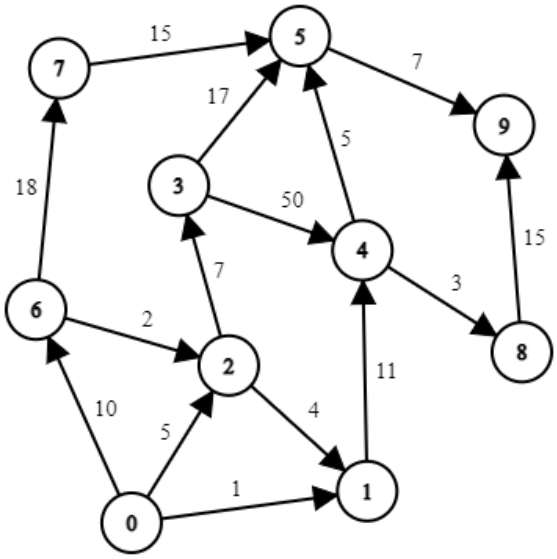
Laboratory 3 – Zsok Alina-Valentina

**Problem 3 -** 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 accessible from the starting vertex. The program will use a matrix defined as *d [x, k] = the cost of the lowest cost walk from s to x and of length at most k, where s is the starting vertex*.

| Dout – Dictionary Value | |
|---|---|
| Key | Value(s) |
| 0 | [1, 2, 7] |
| 1 | [4] |
| 2 | [1, 3] |
| 3 | [4, 5] |
| 4 | [5, 8] |
| 5 | [9] |
| 6 | [2, 7] |
| 7 | [5] |
| 8 | [9] |
| 9 | [] |




In order to find the lowest cost walk between two given vertices I used the Bellman Ford Algorithm because we also want to check if there are negative cost cycles. The Algorithm is done in three main steps. In **Step 1** we initialize the *distance list*, by putting on each position ∞ as the "*initial weight*" and the *predecessor list*, by putting "***null***" on each position as the "*initial predecessor*". Also, in this step we set the *source position* in the *distance list* with 0. In **Step 2** we relax the edges repeatedly (as explained below, there we want to find the *lowest cost walk* between the vertices 0 – **source** and 9 **- target**).

1. **After First Iteration (*source – 0 and target - 9*)**

| distance | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | weight | 0 | 1 | 5 | 12 | 12 | ∞ | 10 | 28 | ∞ | ∞ |
| predecessor | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | vertex | null | 0 | 0 | 2 | 1 | null | 0 | 6 | null | null |

2. **After Second Iteration**

| distance | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | weight | 0 | 1 | 5 | 12 | 12 | 17 | 10 | 28 | 15 | 24 |
| predecessor | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | vertex | null | 0 | 0 | 2 | 1 | 4 | 0 | 6 | 4 | 5 |

3. **After Third Iteration**

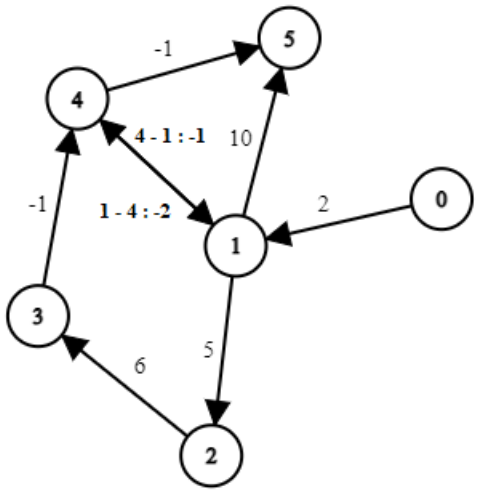
The "table remains the same" for the next iterations (until iteration 8).

In **Step 3** we check for negative-weight cycles by doing one more iteration, if one weight changes that means that there is a negative cycle, otherwise I generated the walk from the source to the target and return the String result that contains the full weight walk and the cost.

In this case the result is: ***Lowest Path = {0 → 1 → 4 → 5 → 9} cost = 24.0***

---

| Dout – Dictionary Value | |
|---|---|
| Key | Value(s) |
| 0 | [0, 1] |
| 1 | [2, 4, 5] |
| 2 | [3] |
| 3 | [4] |
| 4 | [1, 5] |
| 5 | [] |




Now we chose a directed graph that contains a negative cycle. We want to find the lowest cost walk between 0 (source) and 5 (target).

1. **After First Iteration**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| distance | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
| | weight | -1 | 1 | ∞ | ∞ | -1 | ∞ |
| predecessor | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
| | vertex | 0 | 0 | null | null | 1 | null |

2. **After Second Iteration**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| distance | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
| | weight | -2 | -2 | 6 | 12 | -4 | -2 |
| predecessor | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
| | vertex | 0 | 4 | 1 | 2 | 1 | 4 |

3. **After Third Iteration**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| distance | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
| | weight | -3 | -5 | 3 | 9 | -7 | -5 |
| predecessor | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
| | vertex | 0 | 4 | 1 | 2 | 1 | 4 |

4. **After Fourth Iteration**

| distance | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| | weight | -4 | -8 | 0 | 6 | -10 | -8 |
| predecessor | index/vertex | 0 | 1 | 2 | 3 | 4 | 5 |
| | vertex | 0 | 4 | 1 | 2 | 1 | 4 |

The String result it would be: "***Negative Cycle!***".

---

| FILE | SOURCE TO TARGET | PATH | COST |
|---|---|---|---|
| **graph1k** | **1** to **100** | **1** → 5 → 487 → 175 →714 →799 →222 →561 →**100** | 141 |
| | **100** to **1** | **100** → 259 → 229 → 641 → 538 → 854 → **1** | 196 |
| **graph10k** | **1** to **100** | **1** → 7317 → 460 → 6010 → 5295 → 4560 → 5513 → 8467 → 3517 → 99 → 9159 → 6840 → 5177 → 7133 → 288 → **100** | 344 |
| | **100** to **1** | **100** → 4442 → 3980 → 1974 → 407 → 4489 → 5162 → 2008 → 3631 → 2305 → 8336 → **1** | 238 |
| **graph100k** | 1 to 100 | took over 30 min | |
| | 100 to 1 | | |