**Lab: Graphs Bellman Ford, Longest Path in DAG**

This document defines the lab for ["Algorithms – Advanced (Java)" course @ Software University](https://softuni.bg/trainings/2992/algorithms-advanced-with-java-june-2020). Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/Contests/2488/Graphs-DijkstraI-and-MST-Lab).

# Bellman-Ford Shortes paths

You will be given **graph** from the console, your **task** is to find the **shortest** **path** and print it as a sequence **from S source vertex to D destination vertex** and then on the **second** line the **weight** of that path, or if there is **no** **such,** in case of **negative** cycle print a message **"Negative Cycle Detected".**

**Input**

* The input comes from the console. First is an integer the number of **nodes**, then the number of **edges**, after that each **edge** on a new line in the following format **"{source} {destination} {weight}".** Then you will read **two** integers on a **separate** **lines** the **source** and **destination** nodes.

**Output**

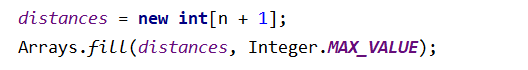
* Print on a single line the **path** **found** **separated** by **spaces and** on the second line the **weight** of that path, or if there is no path message **"Negative Cycle Detected".**

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  8  1 2 8  1 3 10  2 4 1  3 6 2  4 3 -4  4 6 -1  6 5 -2  5 3 1  1  6 | 1 2 4 3 6  7 |
| 4  4  1 2 1  2 3 -1  3 4 -1  4 1 -1  1  4 | Negative Cycle Detected |

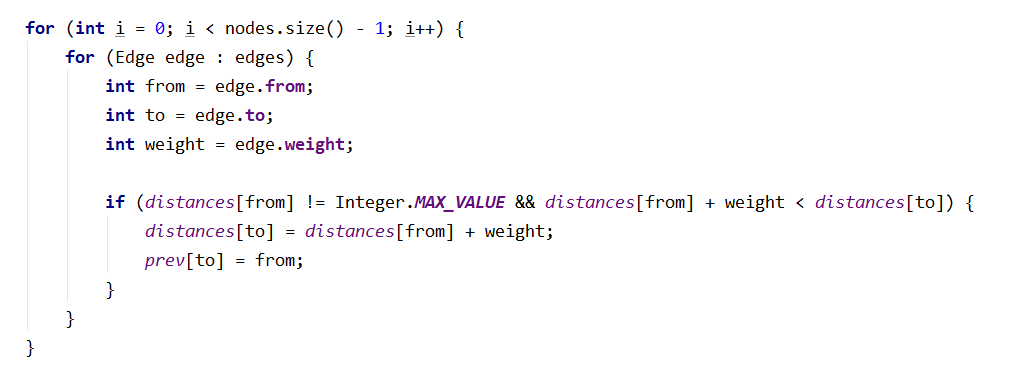
## Initialize Distances

We need an array to hold the minimum distance to each node. Initially, the distance to the source node is 0 and the distance to all other nodes is set to infinity (or, in our case, the maximal value for the int type):



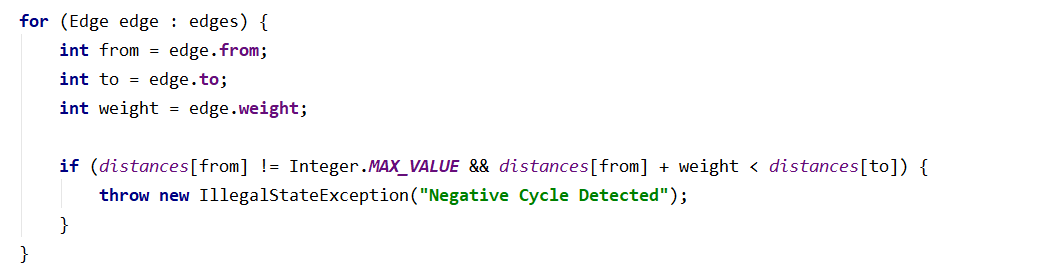
We also need to keep track of the nodes we’ve visited and, in order to reconstruct the path later, the previous node. The source node has no previous, the value for it will be **-1**.

## Run Bellman-Ford



**Add Handling for Negative Cycles**

Simply rerun one more time the algorithm for all edges.



## Reconstruct Shortest Path

You are familiar with this part it is indeed pretty easy.

# Longest Path

You will be given **graph** from the console, your **task** is to find the **longest** **path to D destination vertex** and then print the **weight** of that path.

**Input**

* The input comes from the console. First is an integer the number of **nodes**, then the number of **edges**, after that each **edge** on a new line in the following format **"{source} {destination} {weight}".** Then you will read **two** integers on a **separate** **lines** the **source** and **destination** nodes.

**Output**

* Print on a single line the **weight** of that path.

**Example**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  11  1 2 5  1 3 3  2 4 6  2 3 2  3 5 4  3 6 2  3 4 7  4 6 1  4 5 3  5 6 4  6 2 1  1  6 | 21 |
| 4  4  1 2 5  1 3 3  3 4 6  4 2 4  1  2 | 13 |