3. Exploration of Bellman-Ford Algorithm

Bellman-Ford algorithm is an algorithm that finds the shortest distances between r (given) and all other vertices in weighted directed graph G(V, E) with no negative cycle.

BellmanFord(G, r)

```
for each u∈V
        d_u ← ∞
d_r ← 0
for i ←1 to |V|-1
        for each (u, v) ∈ E    ---------- ❶
                if (d_u + w_u,v < d_v )  d_v ←d_u + w_u,v ;
```

You will have to write 2 different versions of Bellman-ford algorithm.

Version 1) Write your code for the Bellman-Ford pseudo code above.

Version 2) At ❶ it is inefficient to check if edge relaxation is possible for all edges.

Try to improve the efficiency of the algorithm with regard to edge relaxation at ❶.

Note :

1) For the comparison of version 1 and version 2, you are not allowed to use optimization options at compilation.

2) You should make your algorithm as efficient as possible as your version 2 will be graded on a **curve**.

[Constraints]

The starting vertex is 1.(r=1)

The number of vertices : 1 <= N <= 1,000

The number of edges : 1<= E <= 100,000

[File]

Write your implementations in “Solution3.java” file with “Solution3” class.

When you run the code with the following command, the output file “output3.txt” should be generated.
> javac Solution3.java -encoding UTF8 && java Solution3

[Input]
An input file “input3.txt” will be given, which as 10 test cases.
Each case consists of 2 lines; first has the number of vertices of G N, the number of edges of G E, and in the second line, E pieces of information about the edges as source, vertex, weight with space in each.

The indices of V starts from 1 and weights will be between -1000 and 1000 inclusively with no case of 0.

[Output]
For each case, you should print the case number as #x where x is the index of the case.
In the next line, print all distances mod 100,000,000 with splitting by space from version 1. Also, in the next line, print the execution time of version 1. Consequently print the all distances, and the execution time of version 2
You must produce your results as “output3.txt”
NOTE: use ‘System.currentTimeMillis()’ to measure each test case. You don’t have to include FILE I/O time.

[Example]
Input(input3.txt)

```
5 10                                                    ← 1st case
1 2 100 2 1 -50
10 30                                                   ← 2nd case
1 2 100 2 3 -50 3 1 30
…
```

Output(output3.txt)

```
#1
0 100                                   ← Shortest Paths of Version 1
0.0                                     ← Execution time of Version 1
0 100                                   ← Shortest Paths of Version 2
0.0                                     ← Execution time of Version 2
#2
0 100 50
0.0
0 100 50
0.0
…
```