3. Bellman-Ford Algorithm 탐구

Bellman-Ford algorithm은 Negative Cycle이 없는 Weighted Directed Graph G(V, E)와 시작 정점 r이 주어질 때. r에서 출발하여 모든 다른 정점들로 가는 최단경로를 계산하는 알고리즘이다.

```
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 ;
```

여기서 ❶ 부분에서 매번 모든 edge에 대해서 relaxation 가능 여부를 위해 시간을 쓰는 것은 좀 낭비적일 수 있다. 이 부분의 잠재적 비효율성을 개선하는 프로그램을 작성하라.

버전 1. 먼저 주어진 버전의 Bellman-Ford 알고리즘을 구현하라.
버전 2. 위의 지시대로 개선한 버전의 Bellman-Ford 알고리즘을 구현하라.

수행 시간 비교를 위해 컴파일 시에 optimization option은 사용하지 않는다. 위 개선 버전(버전 2)의 수행 시간은 최대한 단축시켜야 함. 이 부분은 상대평가가 됨.

[제약사항]

시작 정점은 1로 한다.
정점의 개수 1≤N≤1000, 간선의 개수 1≤E≤100,000
* 채점을 위한 컴파일 시에 optimization option은 사용하지 않는다.

[파일]

파일 이름은 “Solution3.java”, 클래스 이름은 “Solution3”으로 작성한다.
> javac Solution3.java -encoding UTF8 && java Solution3
명령어를 입력했을 때 output3.txt 파일이 생성되어야 한다.

[입력]

입력 파일에는 10 개의 테스트 케이스가 주어진다. 각 케이스는 2 줄로 이루어진다. 첫 줄에는 정점의 개수 N이 주어지고 공백을 두고 간선의 개수 E가 주어진다. 다음 줄에는 E개의 간선 정보가 공백을 두고 주어지는데, 각각의 간선 정보는 각각 출발 정점, 도착 정점, 가중치로 이루어진 3개의 값이 공백을 두고 주어진다.

이 때, 정점의 번호는 1부터 N까지로 매긴다. 간선의 가중치는 0을 제외한 -1000부터 +1000까지 주어진다. 입력파일의 이름은 “input3.txt”이다.

[출력]

각 테스트 케이스에 대해서, 케이스의 번호를 "#x" 의 형식으로 출력한 후(여기서 x 는 테스트 케이스 번호), 줄을 바꾼 다음 위 버전 1을 수행한 결과의 최단 경로 길이와 수행 시간을 기록한다. 줄을 바꾸어 위 버전 2에서 수행한 결과의 최단 경로 길이와 수행 시간을 기록한다. 출력 결과물을 “output3.txt”로 저장한다. 시간 측정은 java의 ‘System.currentTimeMillis()’를 통해 측정하도록 한다. 시간 측정에 FILE I/O는 포함하지 않아도 된다.

[예제]

입력 (input3.txt)

```
2 2                                                    ← 1번 케이스
1 2 100 2 1 -50
3 3                                                    ← 2번 케이스
1 2 100 2 3 -50 3 1 30
…
```

출력 (output3.txt)

```
#1
0 100                                          ← 버전 1로부터 구한 최단경로
0.0                                            ← 버전 1로부터 구한 시간
0 100                                          ← 버전 2로부터 구한 최단경로
0.0                                            ← 버전 2로부터 구한 시간
#2
0 100 50
0.0
0 100 50
0.0
…
```