Algorithm HW3

1. Problem Set

1) Bellman_Ford Algorithm
2) 최대신장트리
3) All pairs Shortest Paths

각각의 폴더에는 문제설명, 뼈대코드, 샘플입력, 샘플정답이 들어있습니다.

뼈대코드의 이름과 클래스명을 바꾸지 않고 그대로 그 위에 구현하도록 합니다.

작성한 알고리즘의 **시간 복잡도에 대한 분석**을 주석으로 뼈대코드마다 서술합니다.

2. 유의사항

시작 전에 반드시 최신 버전의 Java를 설치하고 **사용하는 편집기에서 설치한 최신 Java 빌드를 사용하도록 세팅해야 합니다(공지 사항의 Java 환경 설정 참고). 소스코드의 인코딩은 utf8**으로 설정하도록 합니다. (Eclipse의 경우 지원이 안 될 수 있어서 한글 깨짐 발생)

멀티 쓰레딩 및 기타 외부 라이브러리 사용을 금지합니다.

3. 제출 양식 및 기한

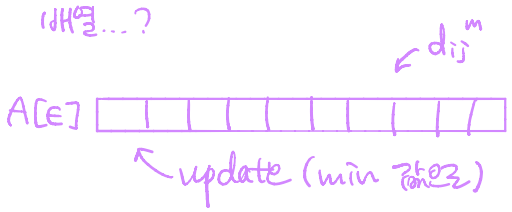
각각 문제에 해당하는 java 파일명(ex. Solution1.java) 그대로 제출합니다. 파일명이 일치하지 않아 채점 코드가 작동하지 않을 시 감점 5%가 있습니다. etl.snu.ac.kr 과목페이지에 문제번호 별로 각각 제출하도록 합니다. 이는 압축했을 시 다양한 파일들(특히나 이미 컴파일 된 파일)이 같이 첨부되는 것을 방지하기 위함입니다. 딜레이 패널티는 1일마다 20% 입니다.

- 제출 기한 : 2020-06-01 월 11:59 pm

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 정점들로 가는 최단경로를 계산하는 알고리즘이다. (이 문제에선 시작 정점 r을 1번 정점으로 하여 풀이를 진행한다.)

```
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 ) then 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 한다. 이 부분은 상대평가가 된다.

[제약사항]

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

[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 “input1.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 결과를 통해 얻은 각 정점들과의 최단 경로 길이를 1억으로 나눈 나머지 각각을 공백을 두고 출력하고, 줄을 바꾸어 수행 시간을 기록한다. 줄을 바꾸어 위 버전 2에서도 버전 1과 동일한 형식으로 최단 경로 길이를 1억으로 나눈 나머지와 수행 시간을 출력한다. 출력 결과물을 “output1.txt”로 저장한다.

[예제]

입력 (input1.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 (output1.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
…
```

2. 최대 신장 트리

N개의 정점과 E개의 간선으로 이루어진 weighted, undirected graph G가 아래의 조건을 만족하도록 주어진다. 각 간선의 가중치는 1 이상 1,000 이하의 정수값을 갖는다.

1) G의 임의의 두 정점에 대해, 이를 잇는 경로(path)가 항상 존재한다. 즉, G는 connected. 따라서 G에는 모든 정점을 잇는 신장 트리(spanning tree)가 존재한다.
2) G는 단순 그래프(simple graph)이다. 즉, 한 정점에서 동일한 정점으로 향하는 간선(self-loop)는 존재하지 않으며, 한 쌍의 정점을 두 개 이상의 간선이 연결하는 경우도 존재하지 않는다.

G의 신장 트리가 가질 수 있는 최대 가중치의 합을 구하는 알고리즘을 작성하라. 채점을 위한 컴파일 시에 최적화 옵션은 쓰지 않는다.

[제약사항]

정점의 개수 $5 \leq N \leq 20,000$, 간선의 개수 $5 \leq E \leq 80,000$
총 수행 시간이 3초를 넘지 않아야 한다.

[입력]

입력 파일에는 10개의 테스트 케이스가 주어진다. 각 케이스는 두 줄로 이루어진다. 첫 줄에는 그래프 G의 정점 수 N과 간선 수 E가 주어진다. 다음 줄에는 E개 간선의 정보를 표현하는 정수들이 나열된다. 각 간선은 양 끝 정점의 번호들과 가중치, 총 세 개의 정수로 표현된다. 정점의 번호는 1번부터 N번까지로 할당된다. 예를 들어 “3 9 17 4 3 5 …”은 3번 정점과 9번 정점을 잇는 가중치 17의 간선이 있고, 4번 정점과 3번 정점을 잇는 가중치 5의 간선이 있음을 의미한다. 즉, 총 3E 개의 정수가 공백을 사이에 두고 나열된다. 입력 파일의 이름은 “input2.txt”이다.

[출력]

각 테스트 케이스에 대해서, 케이스의 번호를 "#x" 의 형식으로 출력한 후(여기서 x는 테스트 케이스 번호), 공백을 하나 둔 다음 G의 신장 트리가 가질 수 있는 최대 가중치 합을 출력한다. 출력 결과물을 “output2.txt”로 저장한다.

[예제]

입력 (input2.txt)

```
5 10                                                                    ← 1번 케이스
4 1 3 1 5 1 2 1 1 5 3 5 2 3 6 3 4 6 4 2 6 4 5 2 3 1 5 2 5 1
7 12                                                                    ← 2번 케이스
1 2 7 6 4 1 7 2 1 1 3 1 4 2 10 4 3 3 4 1 5 2 5 4 7 1 9 7 4 7 6 3 7 5 4 6
…
```

출력 (output2.txt)

```
#1 22
#2 42
…
```

3. All Pairs Shortest Paths

N개의 정점으로 이루어진 weighted directed graph G가 다음 조건을 만족하도록 주어진다.

1) 가중치 합이 음이 되는 사이클이 존재하지 않는다.
2) 강연결요소(strongly connected component)의 개수가 1개다. 즉, 임의의 한 점에서 임의의 다른 점으로 가는 경로가 항상 존재한다.

이 때, 모든 정점 쌍 사이의 최단 거리를 구하는 알고리즘을 구현하라. 각 간선의 가중치는 -1,000 이상 1,000 이하의 정수로 주어진다. 제출하는 프로그램에서는 모든 정점 쌍 (전체 N*(N-1)개) 사이의 최단 거리 총합을 출력하도록 한다. 알고리즘은 최대한 효율적으로 작성하라. 채점을 위한 컴파일 시에 최적화 옵션은 쓰지 않는다.

[제약사항]

정점의 개수 $1 \leq N \leq 200$, 간선의 개수 $1 \leq E \leq 10,000$
총 수행 시간이 3초를 넘지 않아야 한다.

[입력]

입력 파일에는 10 개의 테스트 케이스가 주어진다. 각 케이스는 두 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 출발 정점, 도착 정점, 가중치로 이루어진 3개의 값이 공백을 두고 주어진다. 이 때, 정점의 번호는 1부터 N까지로 매긴다. 입력파일의 이름은 “input3.txt”이다.

[출력]

각 테스트 케이스에 대해서, 케이스의 번호를 "#x" 의 형식으로 출력한 후(여기서 x 는 테스트 케이스 번호), 공백을 하나 둔 다음 주어진 케이스에서 모든 정점 쌍 사이의 최단 거리의 총합을 출력한다. 총합 계산 중 overflow는 일어나지 않도록 입력이 주어질 것이니 고려하지 않아도 된다. 출력 결과물을 “output3.txt”로 저장한다.

[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 50
#2 240
...
```