

Floyd-Warshall Algorithm

Daeung Kim

Contents

1. Floyd-Warshall Algorithm

2. Q2 미래 도시

3. Q3 전보

1. Floyd-Warshall Algorithm

- 모든 노드에서 다른 모든 노드까지의 최단 경로를 모두 계산
- **다익스트라 알고리즘과 마찬가지로 단계별로 ‘해당 노드를 거쳐가는’ 경로를 확인해 최단 거리를 갱신하는 방식으로 알고리즘을 수행**
 - 다만 매 단계마다 방문하지 않은 노드 중에 최단 거리를 갖는 노드를 찾는 과정이 필요 없음
- 2차원 배열에 최단 거리 정보를 저장
- 2차원 배열의 값을 점화식에 따라 갱신한다는 점에서 다이나믹 프로그래밍 유형

1. Floyd-Warshall Algorithm

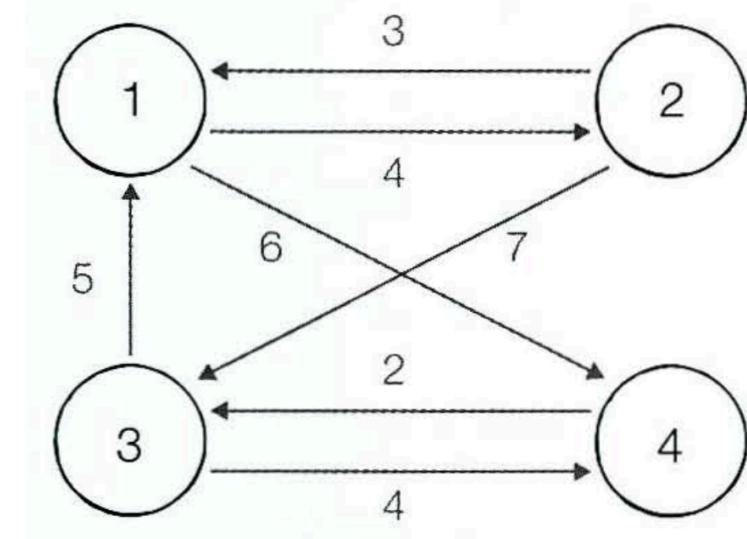
- 각 단계마다 특정한 노드 k 를 거쳐가는 경우를 확인
 - $a \rightarrow b$ 의 최단 거리보다 $a \rightarrow k \rightarrow b$ 가 더 짧은지 검사
- 점화식

$$D_{ab} = \min(D_{ab}, D_{ak} + D_{kb})$$

1. Floyd-Warshall Algorithm

[동작 과정 예시]

- [Step 0] 그래프 준비 & 최단 거리 테이블 초기화



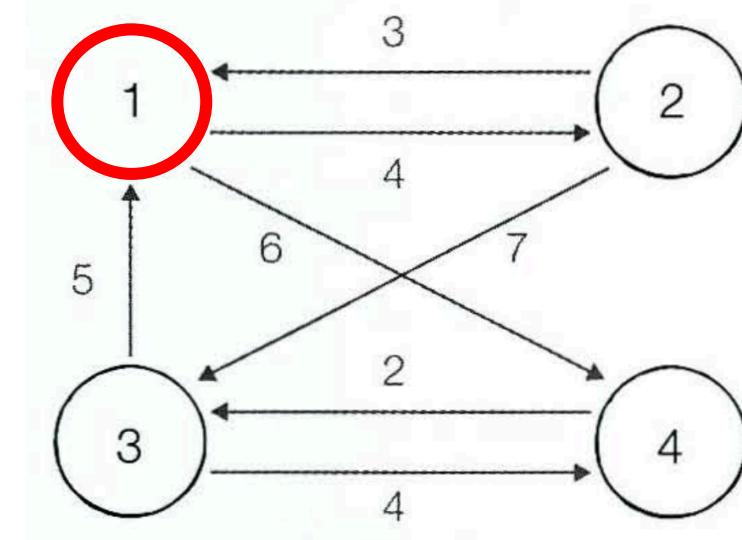
		도착	1번	2번	3번	4번
출발	1번	0	4	무한	6	
	2번	3	0	7	무한	
3번	5	무한	0	4		
4번	무한	무한	2	0		

1. Floyd-Warshall Algorithm

[동작 과정 예시]

$$D_{ab} = \min(D_{ab}, D_{ak} + D_{kb})$$

- [Step 1] 1번 노드를 거쳐 가는 경우를 고려해 테이블을 생성



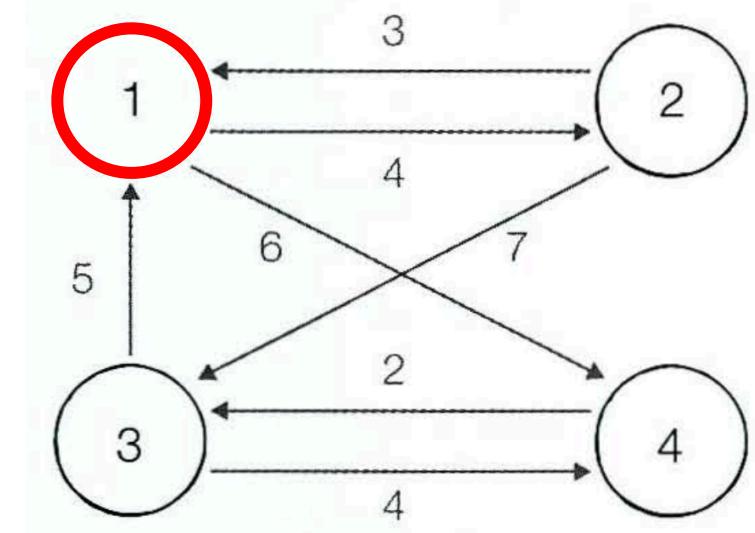
		도착	1번	2번	3번	4번
출발	1번	0	4	무한	6	
	2번	3	0	7	무한	
3번	5	무한	0	4		
4번	무한	무한	2	0		

0	4	무한	6
3	0		
5		0	
무한			0

1. Floyd-Warshall Algorithm

[동작 과정 예시]

- [Step 1] 1번 노드를 거쳐 가는 경우를 고려해 테이블을 갱신



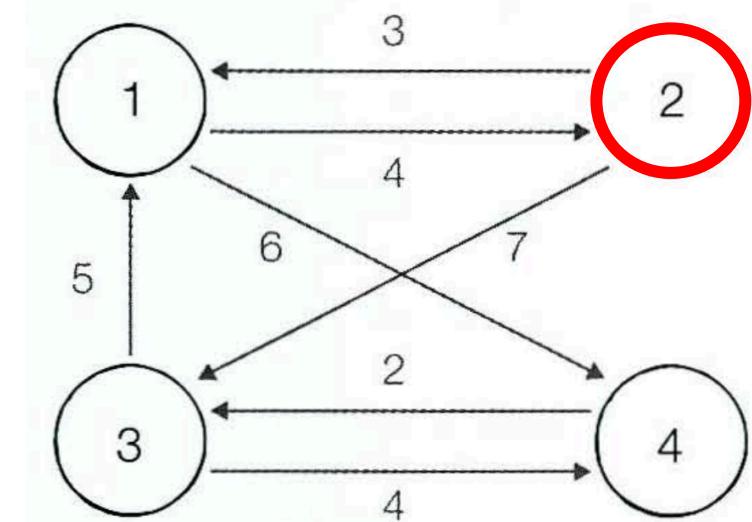
$$\begin{aligned}D_{23} &= \min(D_{23}, D_{21} + D_{13}) \\D_{24} &= \min(D_{24}, D_{21} + D_{14}) \\D_{32} &= \min(D_{32}, D_{31} + D_{12}) \\D_{34} &= \min(D_{34}, D_{31} + D_{14}) \\D_{42} &= \min(D_{42}, D_{41} + D_{12}) \\D_{43} &= \min(D_{43}, D_{41} + D_{13})\end{aligned}$$

0	4	무한	6
3	0	7	9
5	9	0	4
무한	무한	2	0

1. Floyd-Warshall Algorithm

[동작 과정 예시]

- [Step 2] 2번 노드를 거쳐 가는 경우를 고려해 테이블을 갱신



$$D_{13} = \min(D_{13}, D_{12} + D_{23})$$

$$D_{14} = \min(D_{14}, D_{12} + D_{24})$$

$$D_{31} = \min(D_{31}, D_{32} + D_{21})$$

$$D_{34} = \min(D_{34}, D_{32} + D_{24})$$

$$D_{41} = \min(D_{41}, D_{42} + D_{21})$$

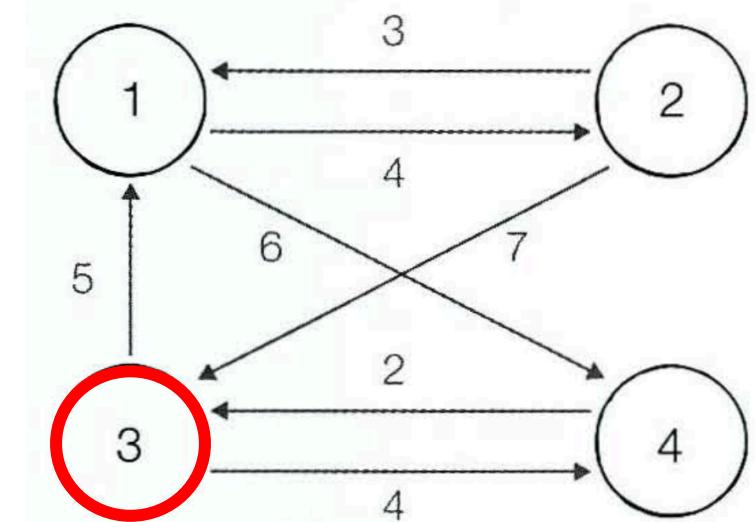
$$D_{43} = \min(D_{43}, D_{42} + D_{23})$$

0	4	11	6
3	0	7	9
5	9	0	4
무한	무한	2	0

1. Floyd-Warshall Algorithm

[동작 과정 예시]

- [Step 3] 3번 노드를 거쳐 가는 경우를 고려해 테이블을 갱신



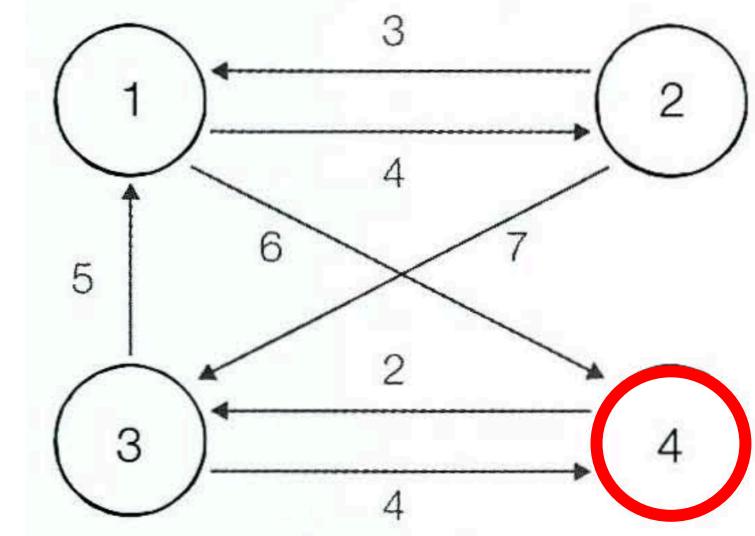
$$\begin{aligned}D_{12} &= \min(D_{12}, D_{13} + D_{32}) \\D_{14} &= \min(D_{14}, D_{13} + D_{34}) \\D_{21} &= \min(D_{21}, D_{23} + D_{31}) \\D_{24} &= \min(D_{24}, D_{23} + D_{34}) \\D_{41} &= \min(D_{41}, D_{43} + D_{31}) \\D_{42} &= \min(D_{42}, D_{43} + D_{32})\end{aligned}$$

0	4	11	6
3	0	7	9
5	9	0	4
7	11	2	0

1. Floyd-Warshall Algorithm

[동작 과정 예시]

- [Step 4] 4번 노드를 거쳐 가는 경우를 고려해 테이블을 갱신



$$D_{12} = \min(D_{12}, D_{14} + D_{42})$$

$$D_{13} = \min(D_{13}, D_{14} + D_{43})$$

$$D_{21} = \min(D_{21}, D_{24} + D_{41})$$

$$D_{23} = \min(D_{23}, D_{24} + D_{43})$$

$$D_{31} = \min(D_{31}, D_{34} + D_{41})$$

$$D_{32} = \min(D_{32}, D_{34} + D_{42})$$

0	4	8	6
3	0	7	9
5	9	0	4
7	11	2	0

1. Floyd-Warshall Algorithm

- 소스 코드

```
1  INF = int(1e9) # 무한을 의미하는 값으로 10억 설정
2  # 노드 개수, 간선 개수 입력받기
3  n = int(input())
4  m = int(input())
5  # 2차원 리스트(그래프 표현)을 만들고, 무한으로 초기화
6  graph = [[INF] * (n + 1) for _ in range(n + 1)]
7
8  # 자기 자신에서 자기 자신으로 가는 비용은 0으로 초기화
9  for a in range(1, n + 1):
10     for b in range(1, n + 1):
11         if a == b:
12             graph[a][b] = 0
13
14  # 각 간선에 대한 정보를 입력 받아, 그 값으로 초기화
15  for _ in range(m):
16      # A에서 B로 가는 비용은 C라고 설정
17      a, b, c = map(int, input().split())
18      graph[a][b] = c
19
20  # 점화식에 따라 플로이드 워셜 알고리즘 수행
21  for k in range(1, n + 1):
22      for a in range(1, n + 1):
23          for b in range(1, n + 1):
24              graph[a][b] = min(graph[a][b], graph[a][k] + graph[k][b])
25
26  # 수행된 결과를 출력
27  for a in range(1, n + 1):
28      for b in range(1, n + 1):
29          # 도달할 수 없는 경우, 무한(INFINITY)이라고 출력
30          if graph[a][b] == INF:
31              print("INFINITY", end=" ")
32          # 도달할 수 있는 경우 거리를 출력
33          else:
34              print(graph[a][b], end=" ")
```

1. Floyd-Warshall Algorithm

[플로이드 워셜 알고리즘 성능 분석]

- 노드 개수 N 일 때 알고리즘상으로 N 번의 단계 수행
 - 각 단계마다 $O(N^2)$ 의 연산을 통해 현재 노드를 거쳐 가는 모든 경로 고려
 - 경유노드 K 를 제외, $N - 1$ 개 노드에 대해 $n-1P_2$ 개 쌍 확인 : $O(n-1P_2) = O(N^2)$
 - 따라서 플로이드 워셜 알고리즘의 총 시간 복잡도는 $O(N^3)$
-
- 일반적으로 노드 개수 $N \leq 500$ 정도일 때 시도해볼 수 있다.
 - 그 이상일 경우 각 노드에 대해 다익스트라 알고리즘을 수행하여 시간 단축 가능

1. Floyd-Warshall Algorithm

[플로이드 워셜 vs N번의 다익스트라]

- 프로그래머스-합승택시요금의 효율성 테스트 결과
- 노드 개수 $3 \leq N \leq 200$
- 플로이드 워셜(좌) / N번의 다익스트라(우)

효율성 테스트
테스트 1 > 통과 (285.46ms, 10.5MB)
테스트 2 > 통과 (1036.04ms, 11.1MB)
테스트 3 > 통과 (2275.04ms, 11.5MB)
테스트 4 > 통과 (2378.47ms, 11.5MB)
테스트 5 > 통과 (2377.10ms, 11.5MB)
테스트 6 > 통과 (2247.87ms, 11.4MB)
테스트 7 > 통과 (2345.04ms, 13.9MB)
테스트 8 > 통과 (2258.22ms, 14.2MB)
테스트 9 > 통과 (2453.24ms, 13MB)
테스트 10 > 통과 (2393.26ms, 13MB)

효율성 테스트
테스트 1 > 통과 (18.50ms, 10.6MB)
테스트 2 > 통과 (98.74ms, 11.7MB)
테스트 3 > 통과 (40.16ms, 11.4MB)
테스트 4 > 통과 (39.05ms, 11.5MB)
테스트 5 > 통과 (37.06ms, 11.6MB)
테스트 6 > 통과 (35.60ms, 11.5MB)
테스트 7 > 통과 (911.53ms, 16.6MB)
테스트 8 > 통과 (943.52ms, 17.1MB)
테스트 9 > 통과 (767.54ms, 17.1MB)
테스트 10 > 통과 (760.60ms, 17MB)

1. Floyd-Warshall Algorithm

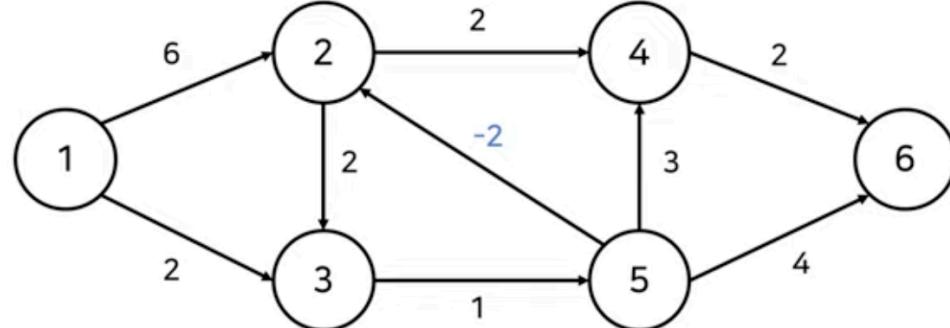
[음수 간선이 존재할 경우]

- 다익스트라와 마찬가지로 음수인 사이클은 알고리즘의 정당성을 보장 못함
- 다만, 벨만-포드 알고리즘과 동일하게 음수 사이클이 아닌 음수 간선은 처리할 수 있다.
- **가중치의 합이 음수인 사이클의 존재 여부 파악**
 - 자기 자신에게 가는 간선들을 검사함으로써 확인
 - 자기 자신에게 가는 간선 $\text{graph}[i][i]$ 는 0으로 초기화된다.
 - 플로이드 워셜 수행 후 $\text{graph}[i][i] < 0$ 이면 그래프에 음수 사이클이 존재한다.
 - Ex) i에서 출발해 다른 정점 k를 거쳐 다시 i로 돌아오면 $i \rightarrow k \rightarrow i$ 사이클을 이룬다.
이때 이 사이클의 가중치 합이 음수일 경우,
초기값이 0이었던 최단 경로 $\text{graph}[i][i]$ 가 음수값을 갖게 된다.

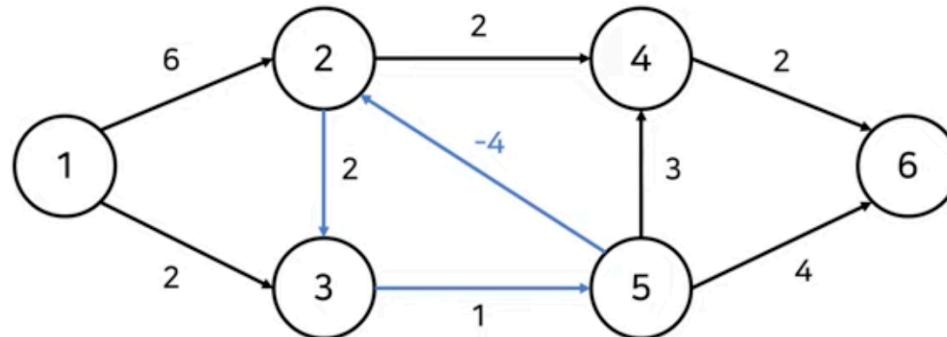
1. Floyd-Warshall Algorithm

[참고 – 음수 사이클(negative cycle) 예시]

- 1번 노드에서 시작하여 각 노드의 최단 거리를 구하는 예시
- 음수 간선이 포함되어 있어도 음수 사이클이 없다면 최단 거리를 계산할 수 있음
- 음수 사이클이 포함된 경우 최단 거리가 음의 무한인 노드가 발생



도착 노드	최소 비용
1번	0
2번	1
3번	2
4번	3
5번	3
6번	5



도착 노드	최소 비용
1번	0
2번	$-\infty$
3번	$-\infty$
4번	$-\infty$
5번	$-\infty$
6번	$-\infty$

최단 경로 알고리즘 비교

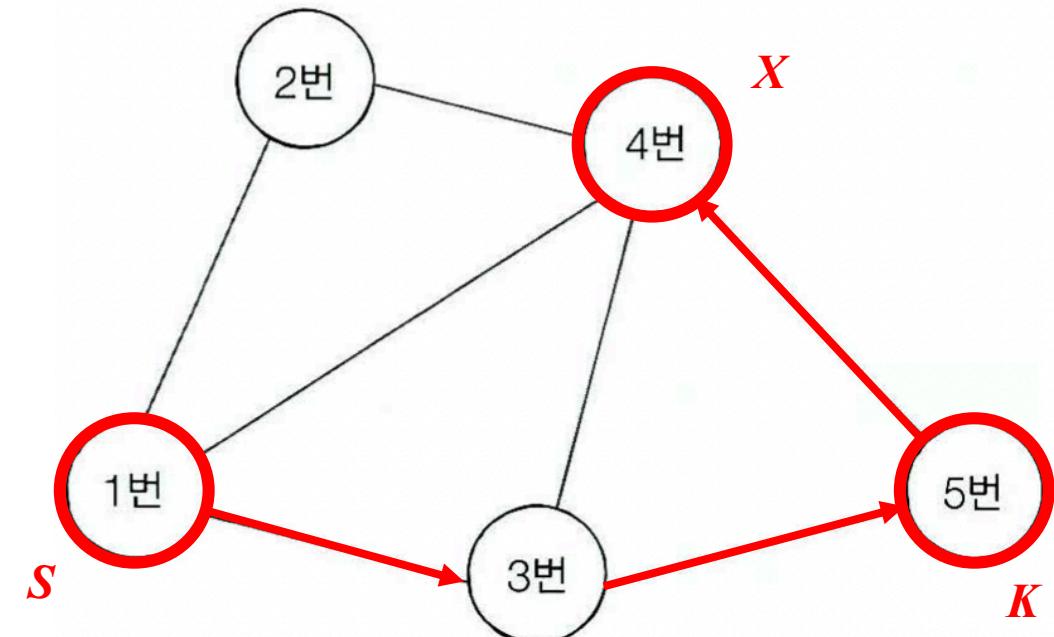
	BFS	Dijkstra	Bellman-Ford	Floyd-Warshall
특징	한 노드에서 다른 노드들까지의 최단 경로	한 노드에서 다른 노드들까지의 최단 경로	한 노드에서 다른 노드들까지의 최단 경로	모든 노드 쌍에 대해 최단 경로
가중치의 값	모두 동일한 가중치	양의 정수	음의 정수 가능 (음수 사이클 X)	음의 정수 가능 (음수 사이클 X)
자료구조	큐	우선순위 큐	DP 1D array $\text{cost}[n] = \min(\text{cost}[n], \text{cost}[k] + E(k, n))$	DP 2D array $\text{cost}[i][j] = \min(\text{cost}[i][j], \text{cost}[i][k] + \text{cost}[k][j])$
시간복잡도	$O(E)$	- 우선순위 큐 : $O(E \log V)$ - 선형탐색 : $O(V^2)$	$O(EV)$	$O(V^3)$

2. Q2 – 미래 도시

- 방문 판매원 A는 많은 회사가 모여 있는 공중 미래 도시에 있다. 공중 미래 도시에는 **1번부터 N번 까지의 회사**가 있는데 특정 회사끼리는 서로 도로를 통해 연결되어 있다. 방문 판매원 A는 현재 1번 회사에 위치해 있으며, X번 회사에 방문해 물건을 판매하고자 한다.
- 공중 미래 도시에서 특정 회사에 도착하기 위한 방법은 회사끼리 연결되어 있는 도로를 이용하는 방법이 유일하다. 또한 **연결된 2개의 회사는 양방향으로 이동할 수 있다**. 공중 미래 도시에서의 도로는 마하의 속도로 사람을 이동시켜주기 때문에 **특정 회사와 다른 회사가 도로로 연결되어 있다면, 정확히 1만큼의 시간으로 이동할 수 있다**.
- 또한 오늘 방문 판매원 A는 기대하던 소개팅에도 참석하고자 한다. 소개팅의 상대는 K번 회사에 존재한다. 방문 판매원 A는 X번 회사에 가서 물건을 판매하기 전에 먼저 소개팅 상대의 회사에 찾아가서 함께 커피를 마실 예정이다. 따라서 방문 판매원 A는 **1번 회사에서 출발하여 K번 회사를 방문 한 뒤에 X번 회사로 가는 것이 목표다**. 이때 방문 판매원 A는 가능한 한 빠르게 이동하고자 한다. **방문 판매원이 회사 사이를 이동하게 되는 최소 시간을 계산하는 프로그램을 작성하시오** 이 때 소개팅의 상대방과 커피를 마시는 시간 등은 고려하지 않는다고 가정한다.

2. Q2 – 미래 도시

- 예를 들어 $N = 5$, $X = 4$, $K = 5$ 이고 회사 간 도로가 7개면서 각 도로가 다음과 같이 연결되어 있을 때를 가정할 수 있다.
 - (1번, 2번), (1번, 3번), (1번, 4번),
(2번, 4번), (3번, 4번), (3번, 5번), (4번, 5번)
- 이때 방문 판매원 A가 최종적으로 4번 회사에 가는 경로를 (1번-3번-5번-4번)으로 설정하면, 소개팅에도 참석할 수 있으면서 총 3만큼의 시간으로 이동할 수 있다. 따라서 이 경우 최소 이동 시간은 3이다.



2. Q2 – 미래 도시

- 문제 조건

| 시간 제한 1초 | 메모리 제한 128MB |

입력 조건

- 첫째 줄에 전체 회사의 개수 N과 경로의 개수 M이 공백으로 구분되어 차례대로 주어진다.
 $(1 \leq N, M \leq 100)$
- 둘째 줄부터 M + 1번째 줄에는 연결된 두 회사의 번호가 공백으로 구분되어 주어진다.
- $M + 2$ 번째 줄에는 X와 K가 공백으로 구분되어 차례대로 주어진다. $(1 \leq K \leq 100)$

출력 조건

- 첫째 줄에 방문 판매원 A가 K번 회사를 거쳐 X번 회사로 가는 최소 이동 시간을 출력한다.
- 만약 X번 회사에 도달할 수 없다면 -1을 출력한다.

2. Q2 – 미래 도시

- 입출력 예시

입력 예시 1

```
5 7  
1 2  
1 3  
1 4  
2 4  
3 4  
3 5  
4 5  
4 5
```

출력 예시 1

```
3
```

입력 예시 2

```
4 2  
1 3  
2 4  
3 4
```

출력 예시 2

```
-1
```

2. Q2 – 미래 도시

- **핵심 아이디어**

- 출발지, 경유지, 도착지가 주어진 전형적인 플로이드 워셜 알고리즘 문제
- $1 \leq N, M \leq 100 \rightarrow N^3 \leq 1,000,000 \rightarrow$ 플로이드 워셜 적용 가능
- 1번 노드에서 K를 거쳐 X로 가는 최단 거리
 $= (1\text{번 노드에서 } K\text{까지 최단 거리}) + (K\text{에서 } X\text{까지 최단 거리})$

2. Q2 – 미래 도시

- 소스 코드

```
1 # Solution
2 INF = int(1e9) # 무한을 의미하는 값으로 10억 설정
3
4 # 노드 개수, 간선 개수 입력받기
5 n, m = map(int, input().split())
6 # 2차원 리스트(그래프 표현)을 만들고, 무한으로 초기화
7 graph = [[INF] * (n + 1) for _ in range(n + 1)]
8
9 # 자기 자신에서 자기 자신으로 가는 비용은 0으로 초기화
10 for a in range(1, n + 1):
11     for b in range(1, n + 1):
12         if a == b:
13             graph[a][b] = 0
14
15 # 각 간선에 대한 정보를 입력 받아, 그 값으로 초기화
16 for _ in range(m):
17     # A에서 B로 가는 비용은 1이라고 설정
18     a, b, c = map(int, input().split())
19     graph[a][b] = 1
20     graph[b][a] = 1
21
22 # 거쳐 갈 노드 X와 최종 목적지 노드 K를 입력받기
23 x, k = map(int, input().split())
```

```
25 # 점화식에 따라 플로이드 워셜 알고리즘 수행
26 for k in range(1, n + 1):
27     for a in range(1, n + 1):
28         for b in range(1, n + 1):
29             graph[a][b] = min(graph[a][b], graph[a][k] + graph[k][b])
30
31 # 수행된 결과를 출력
32 distance = graph[1][k] + graph[k][x]
33
34 # 도달할 수 없는 경우, -1 출력
35 if distance >= INF:
36     print('-1')
37 # 도달할 수 있다면, 최단 거리를 출력
38 else:
39     print(distance)
```

2. Q3 – 전보

- 어떤 나라에는 **N개의 도시**가 있다. 그리고 각 도시는 보내고자 하는 메시지가 있는 경우, 다른 도시로 전보를 보내서 다른 도시로 해당 메시지를 전송할 수 있다.
- 하지만 X라는 도시에서 Y라는 도시로 전보를 보내고자 한다면, 도시 X에서 Y로 향하는 통로가 설치되어 있어야 한다. 예를 들어 X에서 Y로 향하는 통로는 있지만, Y에서 X로 향하는 통로가 없다면 Y는 X로 메시지를 보낼 수 없다. 또한 통로를 거쳐 메시지를 보낼 때는 일정 시간이 소요된다.
- 어느 날 C라는 도시에서 위급 상황이 발생했다. 그래서 최대한 많은 도시로 메시지를 보내고자 한다. 메시지는 **도시 C에서 출발하여 각 도시 사이에 설치된 통로를 거쳐 최대한 많이 퍼져나갈 것이다.**
- 각 도시의 번호와 통로가 설치되어 있는 정보가 주어졌을 때, 도시 C에서 보낸 메시지를 받게 되는 도시의 개수는 총 몇 개이며 도시들이 모두 메시지를 받는 데까지 걸리는 시간은 얼마인지 계산하는 프로그램을 작성하시오.

2. Q3 – 전보

- 문제 조건

| 시간 제한 1초 | 메모리 제한 128MB |

입력 조건

- 첫째 줄에 도시의 개수 N, 통로의 개수 M, 메시지를 보내고자 하는 도시 C가 주어진다.
 $(1 \leq N \leq 30,000, 1 \leq M \leq 200,000, 1 \leq C \leq N)$
- 둘째 줄부터 M + 1번째 줄에 걸쳐서 통로에 대한 정보 X, Y, Z가 주어진다. 이는 특정 도시 X에서 다른 특정 도시 Y로 이어지는 통로가 있으며, 메시지가 전달되는 시간이 Z라는 의미다.
 $(1 \leq X, Y \leq N, 1 \leq Z \leq 1,000)$

출력 조건

- 첫째 줄에 도시 C에서 보낸 메시지를 받는 도시의 총 개수와 총 걸리는 시간을 공백으로 구분하여 출력한다.

2. Q3 – 전보

- 입출력 예시

입력 예시

```
3 2 1  
1 2 4  
1 3 2
```

출력 예시

```
2 4
```

2. Q3 – 전보

- **핵심 아이디어**

- 한 도시에서 다른 도시까지의 최단 거리 문제
- 노드의 개수 N, 간선의 개수 M : $1 \leq N \leq 30,000, 1 \leq M \leq 200,000$ 으로 충분히 크기 때문에 우선순위큐를 활용한 다익스트라 알고리즘 구현
- 최단거리 테이블을 구한 뒤, 연결된 노드의 개수와 최댓값을 출력

```

1 import heapq
2 import sys
3
4 input = sys.stdin.readline
5 INF = int(1e9) # 무한을 의미하는 값으로 10억을 설정
6
7 # 노드 개수, 간선의 개수, 시작 노드를 입력받기
8 n, m, start = map(int, input().split())
9 # 각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트 만들기
10 graph = [[] for i in range(n + 1)]
11 # 최단 거리 테이블을 모두 무한으로 초기화
12 distance = [INF] * (n + 1)
13
14 # 모든 간선 정보를 입력받기
15 for _ in range(m):
16     x, y, z = map(int, input().split())
17     # x번 노드에서 y번 노드로 가는 비용이 z라는 의미
18     graph[x].append((y, z))
19
20 def dijkstra(start):
21     q = []
22     # 시작 노드로 가기 위한 최단 경로는 0으로 설정하여, 큐에 삽입
23     heapq.heappush(q, (0, start))
24     distance[start] = 0
25     while q: # 큐가 비어있지 않다면
26         # 가장 최단 거리가 짧은 노드에 대한 정보를 꺼내기
27         dist, now = heapq.heappop(q)
28         if distance[now] < dist:
29             continue
30         # 현재 노드와 연결된 다른 인접한 노드들을 확인
31         for i in graph[now]:
32             cost = dist + i[1]
33             # 현재 노드를 거쳐서, 다른 노드로 이동하는 거리가 더 짧은 경우
34             if cost < distance[i[0]]:
35                 distance[i[0]] = cost
36                 heapq.heappush(q, (cost, i[0]))

```

```

38     # 다익스트라 알고리즘 수행
39     dijkstra(start)
40
41     # 도달할 수 있는 노드의 개수
42     count = 0
43     # 도달할 수 있는 노드 중, 가장 멀리 있는 노드와의 최단 거리
44     max_distance = 0
45     for d in distance:
46         # 도달할 수 있는 노드인 경우
47         if d != INF:
48             count += 1
49             max_distance = max(max_distance, d)
50     # 더미 노드는 제외해야 하므로 count - 1을 출력
51     print(count - 1, max_distance)

```

[EOS]
