최단 경로 알고리즘

최단 경로 문제

최단 경로 문제 : **가중 그래프**에서 간선의 가중치의 합이 최소가 되는 경로를 찾는 문제

- **단일 출발 (single-source) 최단 경로** : 어떤 하나의 정점에서 출발하여 나머지 모든 정점까지의 최단 경로를 찾는다.
- **단일 도착 (single-destination) 최단 경로** : 모든 정점에서 출발하여 어떤 하나의 정점까지의 최단 경로를 찾는다.

  -> 그래프 내의 간선들을 뒤집으면 단일 출발 최단 경로 문제로 바뀔 수 있다.
- **단일 쌍 (single-pair) 최단 경로** : 어떤 정점 v에서 v'로 가는 최단 경로를 찾는다.
- **전체 쌍 (all-pair) 최단 경로** : 모든 정점 쌍들 사이의 최단 경로를 찾는다.

<최단 경로 문제를 해결하는 알고리즘>

- **다익스트라 알고리즘** : 음이 아닌 가중 그래프에서의 단일 출발, 단일 도착, 단일 쌍 최단 경로 문제
- **벨만-포드 알고리즘** : 가중 그래프에서의 단일 출발, 단일 도착, 단일 쌍 최단 경로 문제
- **플로이드-워셜 알고리즘** : 전체 쌍 최단 경로 문제

** BFS : 가중치가 없거나 가중치가 동일한 그래프에서 최단 경로를 찾는 경우 가장 빠르다.

다익스트라

다익스트라 알고리즘 (Dijkstra Algorithm)

그래프 G = (V, E) 에서 특정 출발 정점(S)에서 다른 모든 정점까지의 최단 경로를 구하는 알고리즘이다.
음의 가중치를 가지는 간선이 없을 때 정상적으로 동작한다.

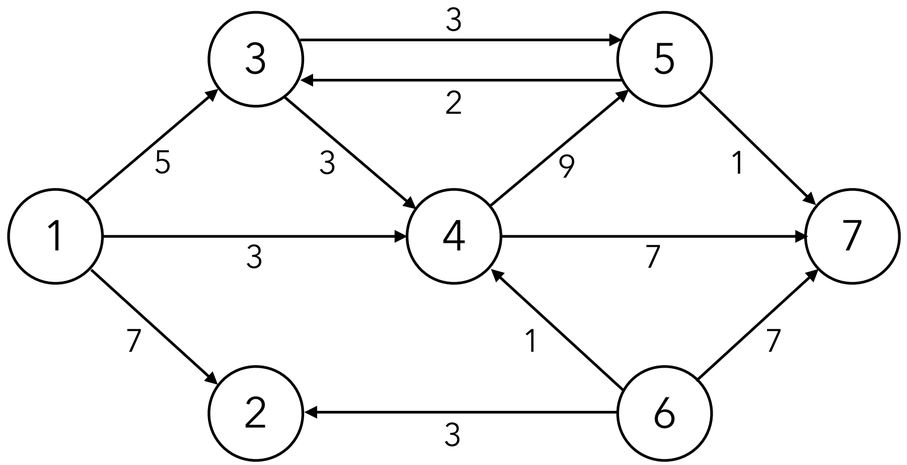
<알고리즘>

1. 출발 노드 S를 설정한다.
2. 출발 노드 S에서 모든 노드들까지의 최단 거리를 저장하는 배열 D를 초기화한다.
3. 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드를 선택한다. (D 배열 검사)
4. 해당 노드를 거쳐 다른 노드로 가는 비용을 계산하여 최단 거리 배열 D를 갱신한다.
5. 모든 노드를 방문할 때까지 3, 4 과정을 반복한다.

<특징>

- 각 정점을 최대 한 번씩만 방문하여 최단 거리를 확정한다.
- 아직 방문하지 않은 정점들 중 최단 거리인 정점을 찾아 방문하는 식으로 진행된다.
  - 이 때, 최단 거리가 최소인 정점을 찾는 과정에서 PriorityQueue 또는 Heap 자료구조를 이용하면 더욱 개선된 알고리즘이 가능하다.
- 매 순간마다 최단 거리의 정점을 선택하는 과정을 반복하므로 그리디 알고리즘으로 분류된다.
- 총 V x V 번 연산이 필요하므로 O(V^2)의 시간복잡도를 가진다.

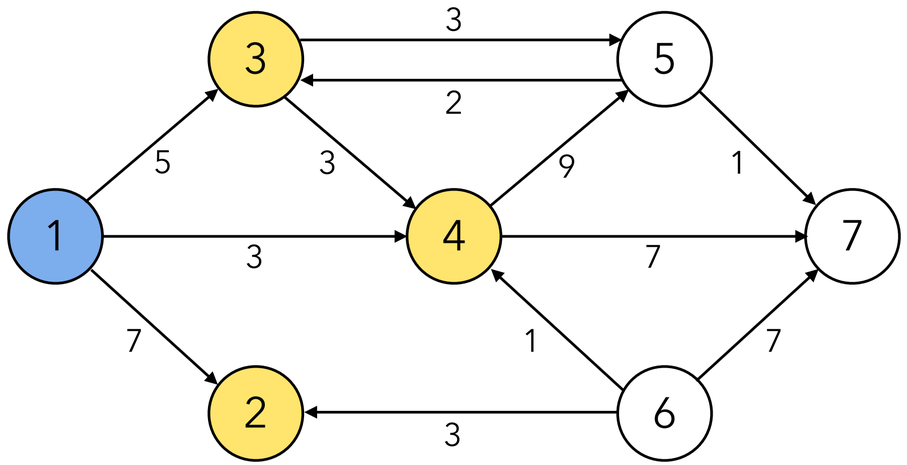
7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

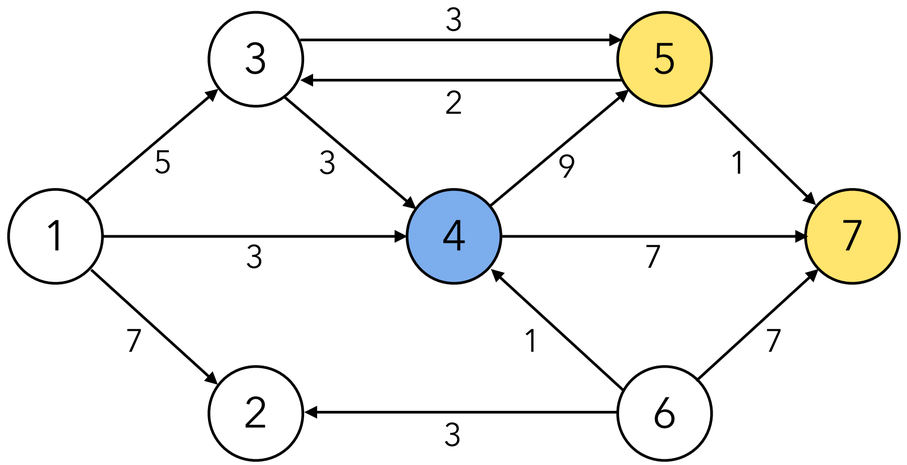
7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

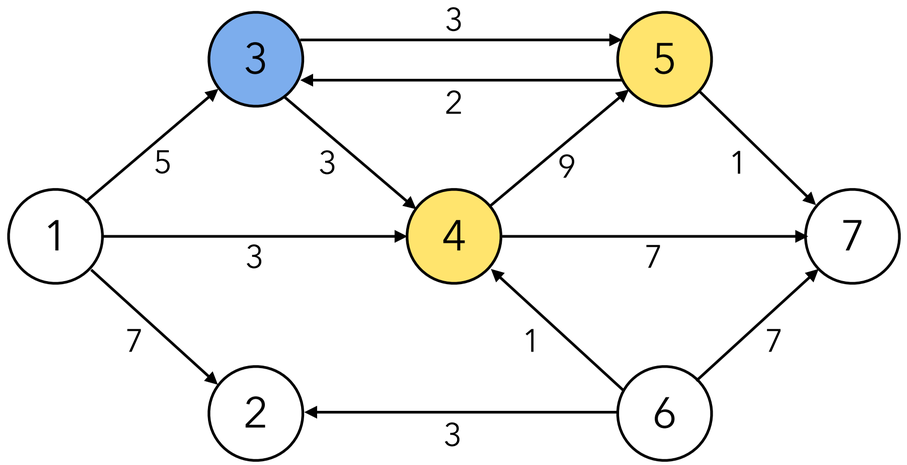
7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞
4	0	7	5	3	3 + 9 = 12	∞	3 + 7 = 10

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.

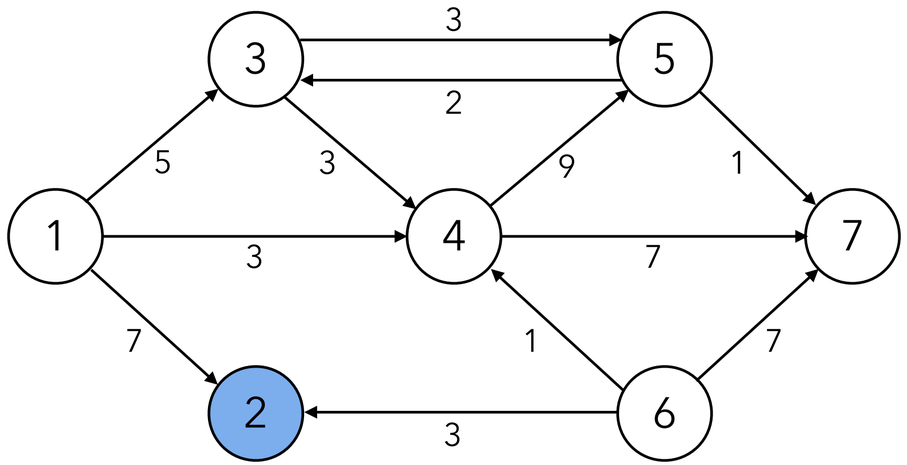


방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞
4	0	7	5	3	12	∞	10
3	0	7	5	3	5 + 3 = 8	∞	10

← 기존에 저장된 값과 비교하여
더 작은 값을 넣음

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

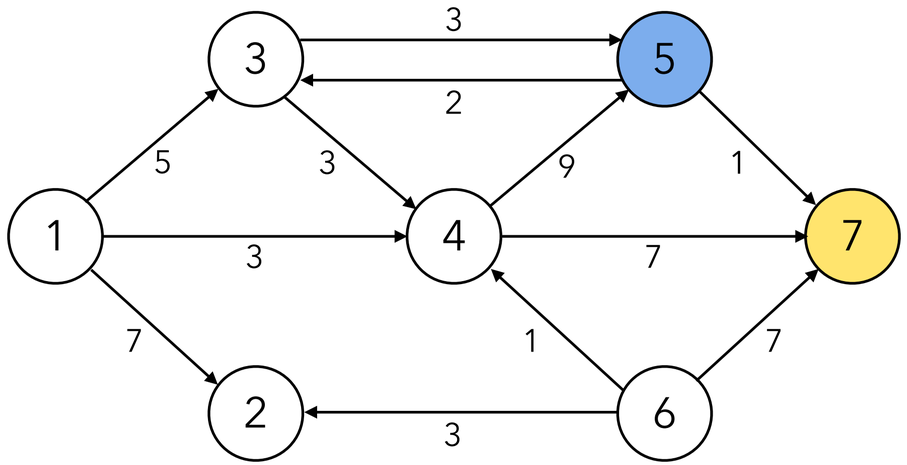
7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞
4	0	7	5	3	12	∞	10
3	0	7	5	3	8	∞	10
2	0	7	5	3	8	∞	10

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

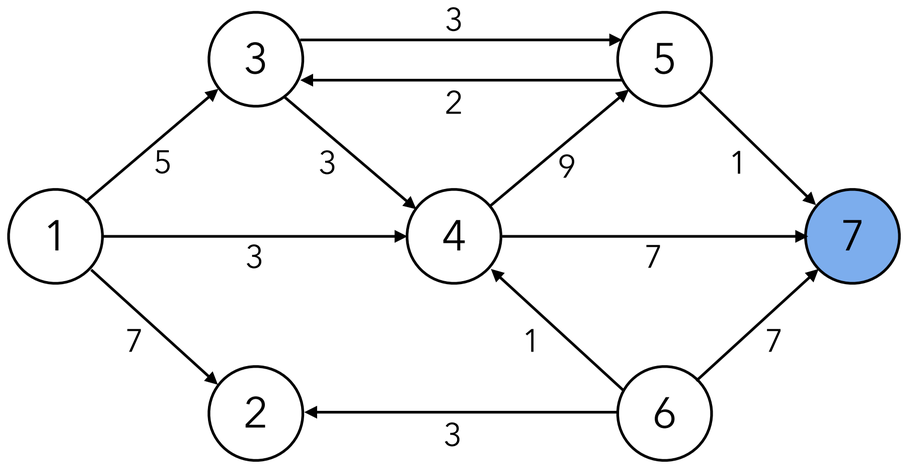
7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞
4	0	7	5	3	12	∞	10
3	0	7	5	3	8	∞	10
2	0	7	5	3	8	∞	10
5	0	7	5	3	8	∞	8 + 1 = 9

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

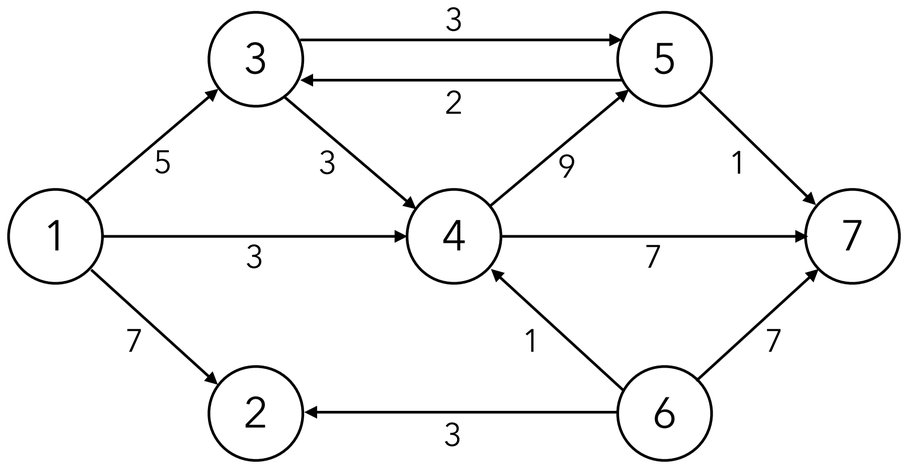
7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞
4	0	7	5	3	12	∞	10
3	0	7	5	3	8	∞	10
2	0	7	5	3	8	∞	10
5	0	7	5	3	8	∞	9
7	0	7	5	3	8	∞	9

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

7개의 정점과 음의 가중치를 갖지 않는 12개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



방문 정점 \ K	1	2	3	4	5	6	7
초기상태	0	∞	∞	∞	∞	∞	∞
1	0	7	5	3	∞	∞	∞
4	0	7	5	3	12	∞	10
3	0	7	5	3	8	∞	10
2	0	7	5	3	8	∞	10
5	0	7	5	3	8	∞	9
7	0	7	5	3	8	∞	9

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

벨만 - 포드

벨만-포드 알고리즘 (Bellman-Ford-Moore Algorithm)

그래프 G = (V, E) 에서 특정 출발 정점(S)에서 다른 모든 정점까지의 최단 경로를 구하는 알고리즘이다.
다익스트라 알고리즘과 달리, 음의 가중치를 가지는 간선도 가능하다.

<알고리즘>

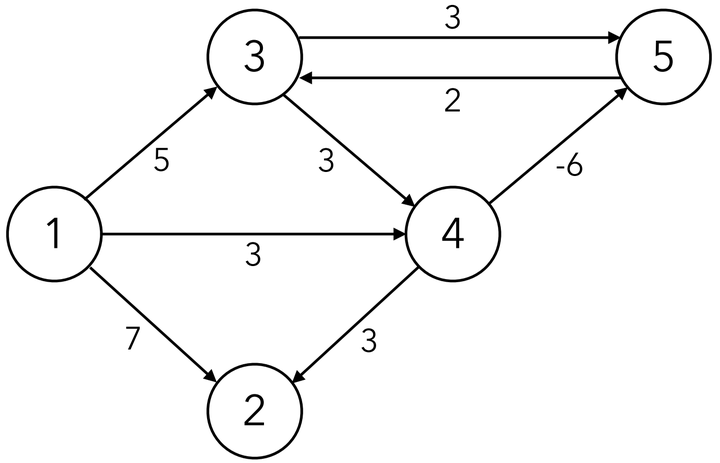
가중 그래프 (V, E)에서 **어떤 정점 A에서 정점 B까지의 최단 거리는 최대 V - 1개의 간선을 사용**한다.
(= 시작 정점 A를 포함하여 최대 V개의 정점을 지난다.)

1. 출발 노드 S를 설정한다.
2. 출발 노드 S에서 모든 노드들까지의 최단 거리를 저장하는 배열 D를 초기화한다.
3. 그래프의 모든 간선을 돌면서 각 노드로 가는 비용을 계산하여 최단 거리 배열 D를 갱신한다.
4. 3 과정을 (노드의 개수 - 1)번, 즉 V-1번 반복한다.
5. 3 과정을 한 번 더 반복하였을 때, 배열 D가 갱신되면 음의 사이클이 있는 것으로 판단한다.

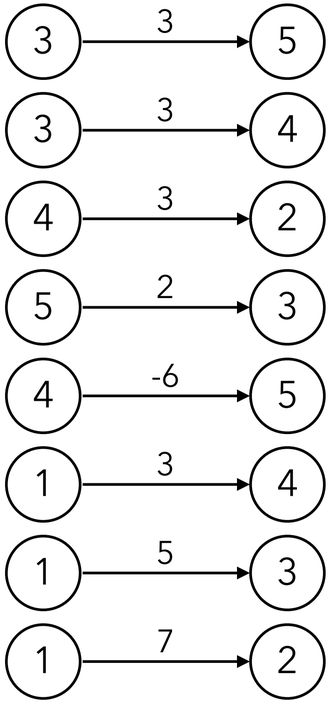
<특징>

- 음의 가중치를 가지는 간선도 가능하므로, 음의 사이클의 존재 여부를 따져야 한다.
- 최단 거리를 구하기 위해서 V - 1번 E개의 모든 간선을 확인한다.
- 음의 사이클 존재 여부를 확인하기 위해서 한 번 더 (V번째) E개의 간선을 확인한다.
  이 때 거리 배열이 갱신되었다면, 그래프 G는 음의 사이클을 가진다.
- 따라서 총 V x E 번 연산하므로 O(VE)의 시간복잡도를 가진다.

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.

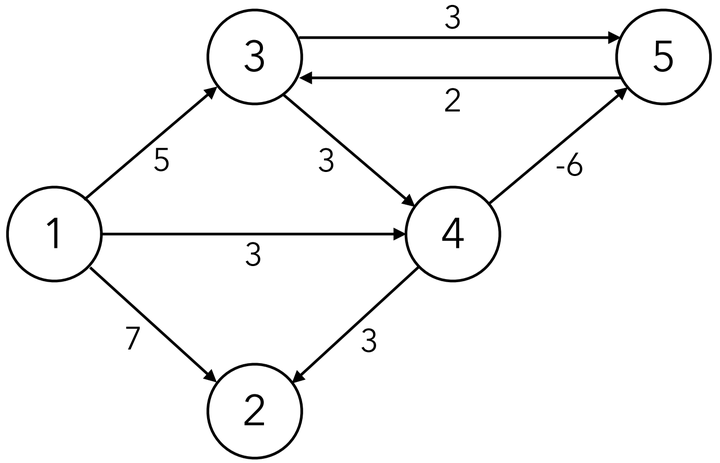


간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞

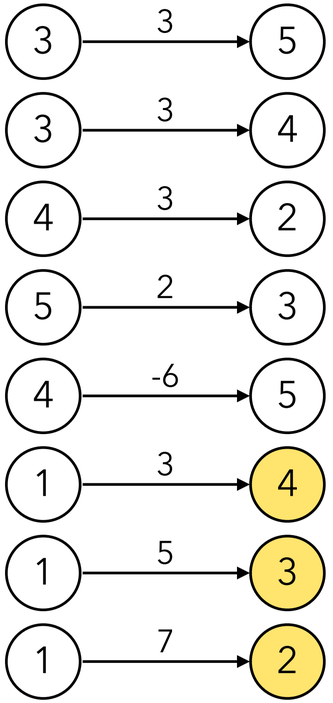


D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.

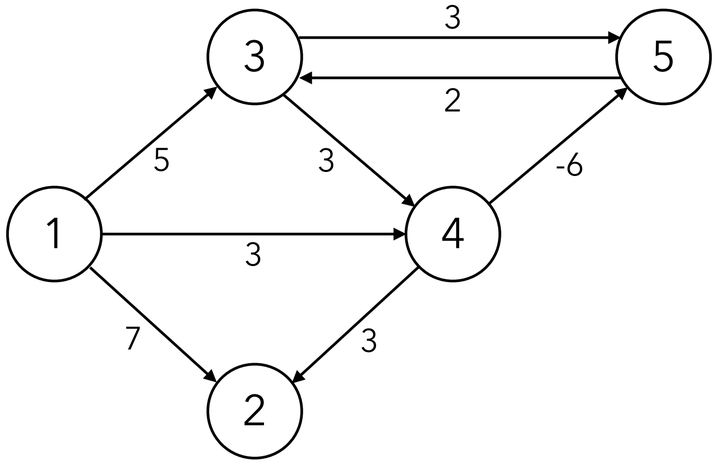


간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞
1	0	7	5	3	∞

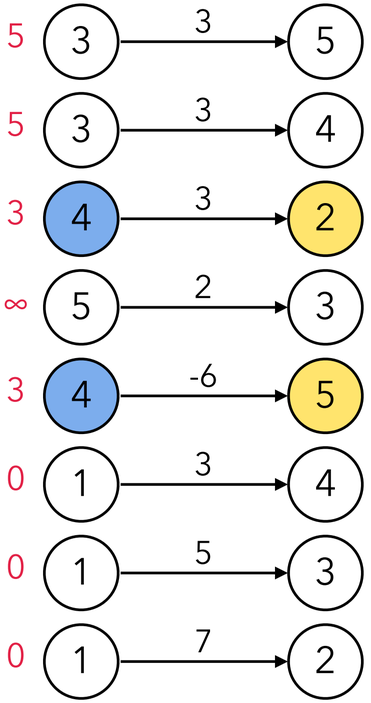


D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.

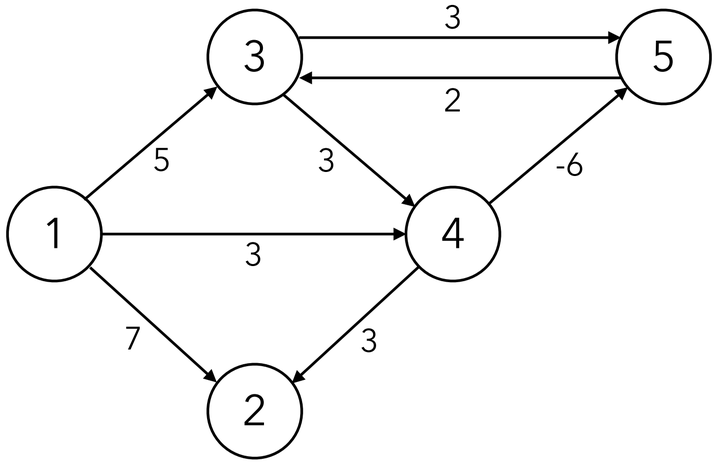


간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞
1	0	7	5	3	∞
2	0	3 + 3 = 6	5	3	3 + (-6) = -3

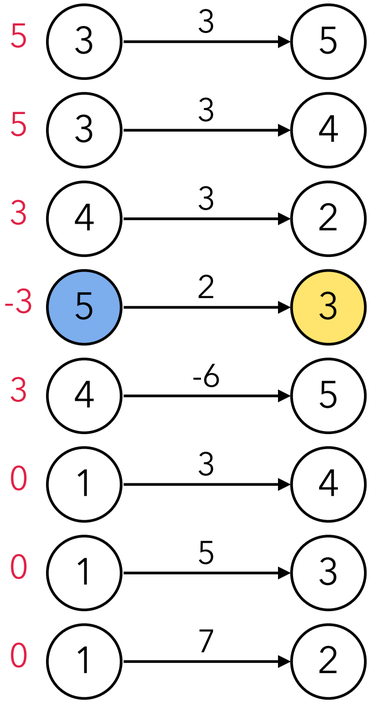


D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.

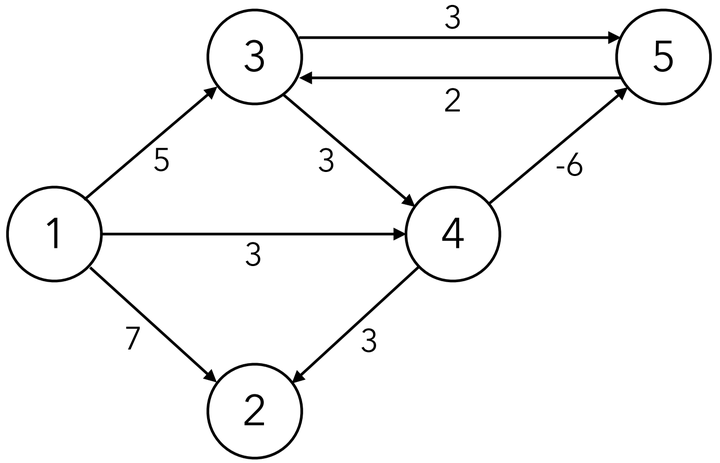


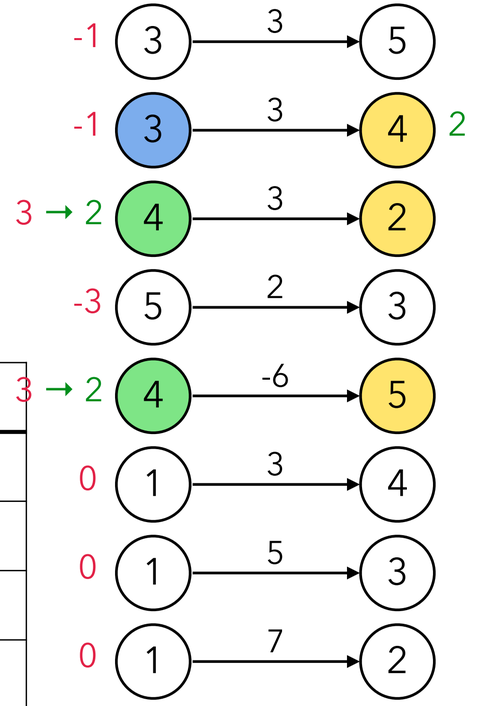
간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞
1	0	7	5	3	∞
2	0	6	5	3	-3
3	0	6	(-3) + 2 = -1	3	-3



D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.

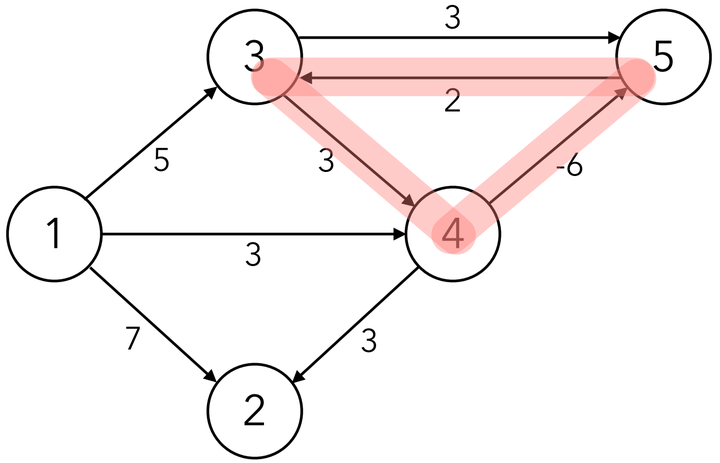




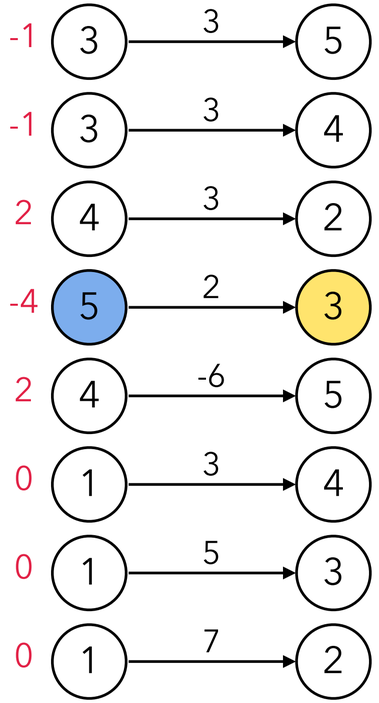
간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞
1	0	7	5	3	∞
2	0	6	5	3	-3
3	0	6	-1	3	-3
4	0	2 + 3 = 5	-1	(-1) + 3 = 2	2 + (-6) = -4

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



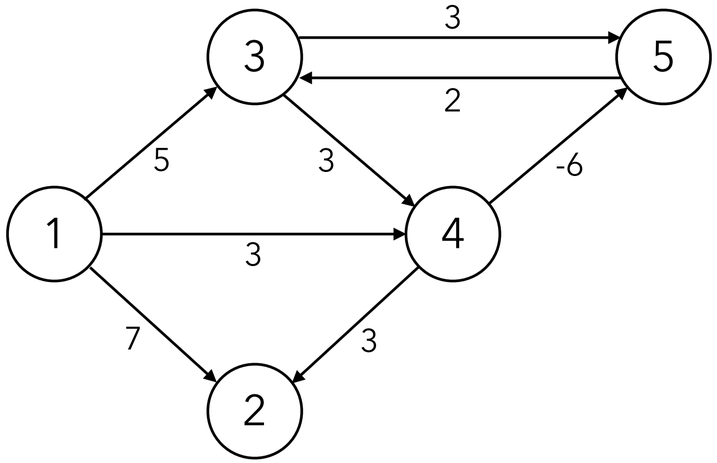
간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞
1	0	7	5	3	∞
2	0	6	5	3	-3
3	0	6	-1	3	-3
4	0	5	-1	2	-4
5	0	5	(-4) + 2 = -2	2	-4



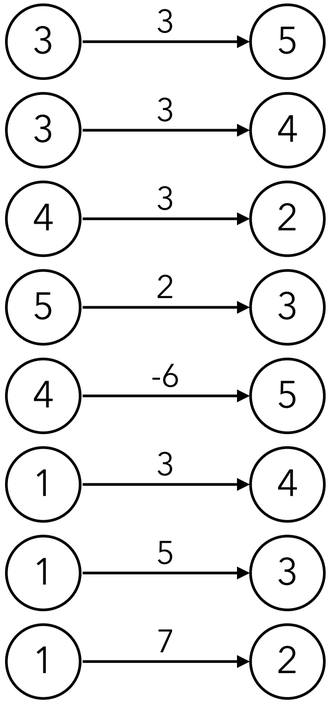
→ 음의 사이클을 가짐

D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
정점 1에서 나머지 정점까지의 최단 거리를 구하여라.



간선 사용 횟수 \ K	1	2	3	4	5
초기상태	0	∞	∞	∞	∞
1	0	7	5	3	∞
2	0	6	5	3	-3
3	0	6	-1	3	-3
4	0	5	-1	2	-4
5	0	5	-2	2	-4



D[K] : 출발 정점 S에서 정점 K까지의 최단 거리를 저장하는 배열

플로이드-워셜

플로이드-워셜 알고리즘 (Floyd-Warshall Algorithm)

그래프 G = (V, E) 에서 모든 정점 사이의 최단 경로를 구하는 알고리즘이다.

<알고리즘>

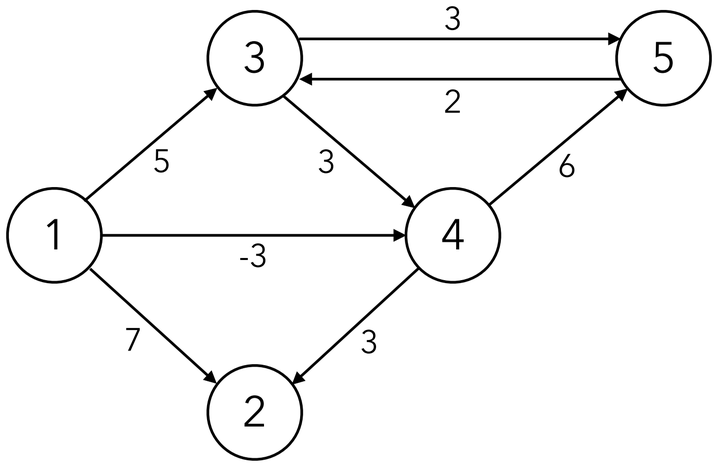
- 어떤 두 정점 사이의 최단 경로는 어떤 경유지(K)를 거치거나, 거치지 않는 경로 중 하나이다.
  즉 정점 A와 정점 B 사이의 최단 경로는 A-B 이거나 A-K-B 이다.
- 만약 경유지(K)를 거친다면 최단 경로를 이루는 부분 경로 역시 최단 경로이다.
  즉 A-B의 최단 경로가 A-K-B라면 A-K와 K-B도 각각 최단 경로이다.

1. 각 노드들 사이의 최단 거리를 저장하는 2차원 배열 D를 초기화한다.
2. 각 노드가 경유지 K를 지날 때마다 최단 경로를 계산하여 배열 D를 갱신한다.
3. 동적 계획법으로 해결하며, 점화식은 D[A][B] = min(D[A][B], D[A][K] + D[K][B] 이다.

<특징>

- 사이클이 없다면 음수 가중치를 가져도 적용 가능하다.
- 동적 계획법(Dynamic Programming)으로 접근한다.
- 모든 가능한 경유지에 대해서 모든 정점 -> 모든 정점으로 가는 최단 거리를 확인하므로 연산 횟수는 V^3이고,
  따라서 시간복잡도는 O(V^3)이다.

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
모든 정점 쌍에 대한 최단 거리를 구하여라.

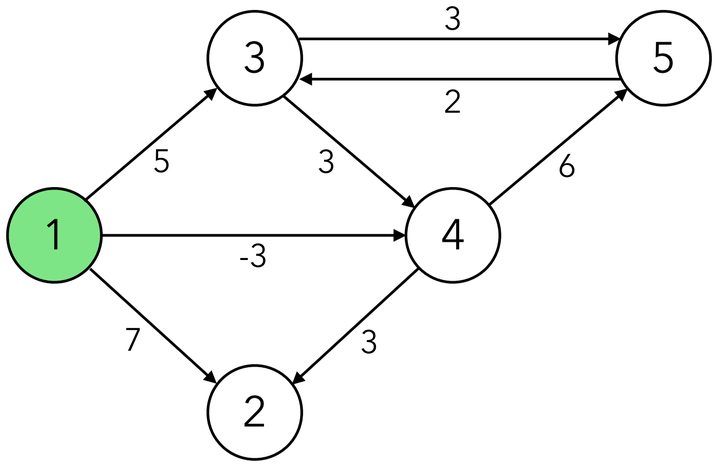


[Step 0. 초기 상태]

I \ J	**1**	**2**	**3**	**4**	**5**
1	0	7	5	-3	∞
2	∞	0	∞	∞	∞
3	∞	∞	0	3	3
4	∞	3	∞	0	6
5	∞	∞	2	∞	0

D[I][J] : 정점 I에서 정점 J까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
모든 정점 쌍에 대한 최단 거리를 구하여라.

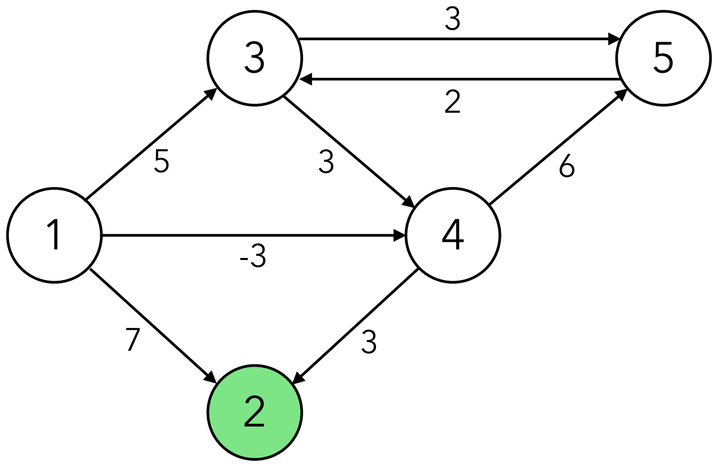


[Step 1. 경유지 K = 1] 점화식 : Dij = min(Dij, Di1 + D1j)

I \ J	**1**	**2**	**3**	**4**	**5**
1	0	7	5	-3	∞
2	∞	0	∞	∞	∞
3	∞	∞	0	3	3
4	∞	3	∞	0	6
5	∞	∞	2	∞	0

D[I][J] : 정점 I에서 정점 J까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
모든 정점 쌍에 대한 최단 거리를 구하여라.

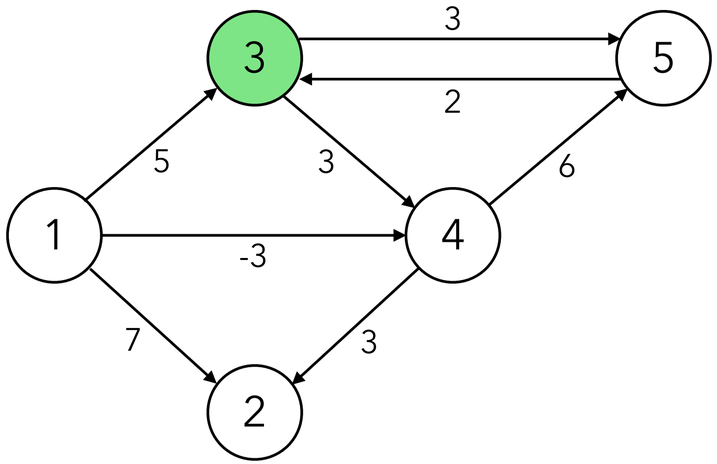


[Step 2. 경유지 K = 2] 점화식 : Dij = min(Dij, Di2 + D2j)

I \ J	**1**	**2**	**3**	**4**	**5**
1	0	7	5	-3	∞
2	∞	0	∞	∞	∞
3	∞	∞	0	3	3
4	∞	3	∞	0	6
5	∞	∞	2	∞	0

D[I][J] : 정점 I에서 정점 J까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
모든 정점 쌍에 대한 최단 거리를 구하여라.

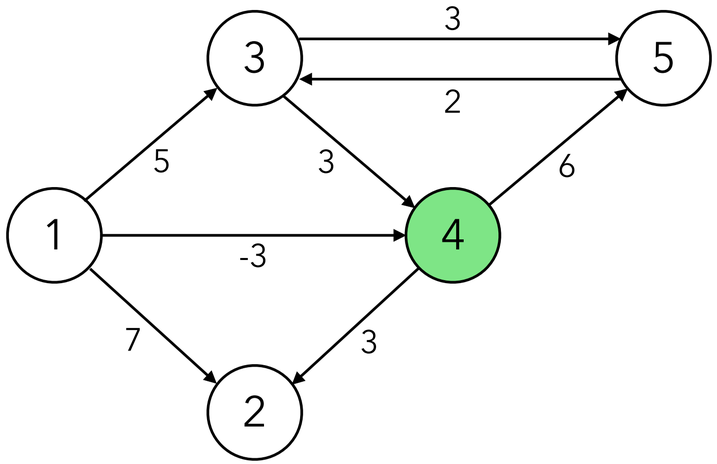


[Step 3. 경유지 K = 3] 점화식 : Dij = min(Dij, Di3 + D3j)

I \ J	**1**	**2**	**3**	**4**	**5**
1	0	7	5	-3	5 + 3 = 8
2	∞	0	∞	∞	∞
3	∞	∞	0	3	3
4	∞	3	∞	0	6
5	∞	∞	2	2 + 3 = 5	0

D[I][J] : 정점 I에서 정점 J까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
모든 정점 쌍에 대한 최단 거리를 구하여라.

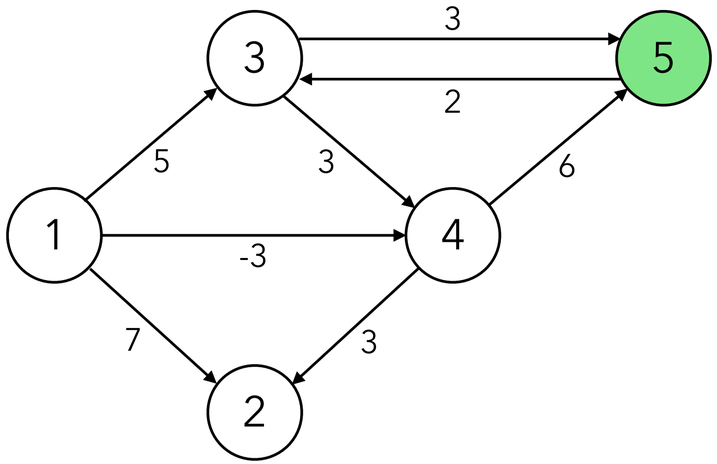


[Step 4. 경유지 K = 4] 점화식 : Dij = min(Dij, Di4 + D4j)

I \ J	1	2	3	4	5
1	0	-3 + 3 = 0	5	-3	-3 + 6 = 3
2	∞	0	∞	∞	∞
3	∞	3 + 3 = 6	0	3	3
4	∞	3	∞	0	6
5	∞	5 + 3 = 8	2	5	0

D[I][J] : 정점 I에서 정점 J까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
모든 정점 쌍에 대한 최단 거리를 구하여라.

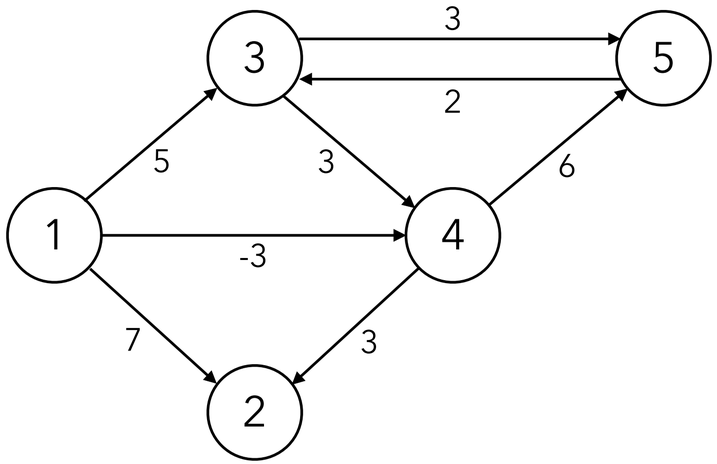


[Step 5. 경유지 K = 5] 점화식 : Dij = min(Dij, Di5 + D5j)

I \ J	1	2	3	4	5
1	0	0	5	-3	3
2	∞	0	∞	∞	∞
3	∞	6	0	3	3
4	∞	3	6 + 2 = 8	0	6
5	∞	8	2	5	0

D[I][J] : 정점 I에서 정점 J까지의 최단 거리를 저장하는 배열

5개의 정점과 8개의 간선으로 이루어진 가중 그래프 G
모든 정점 쌍에 대한 최단 거리를 구하여라.



[FINAL 최단 거리 배열]

I \ J	**1**	**2**	**3**	**4**	**5**
1	0	0	5	-3	3
2	∞	0	∞	∞	∞
3	∞	6	0	3	3
4	∞	3	8	0	6
5	∞	8	2	5	0

D[I][J] : 정점 I에서 정점 J까지의 최단 거리를 저장하는 배열