최단 경로 : 그래프의 시작점에서 다른 지점까지의 최단 거리

알고리즘

이름	간선의 가중치	시작점	도착점	시간 복잡도
BFS	모두 1	한 정점	모든 정점	$O(V+E)$
Dijkstra	≥ 0	한 정점	모든 정점	$O(E \log V)$
Floyd-Warshall	제약 없음	**모든 정점**	모든 정점	$O(V^3)$

- 여러 알고리즘은 각 장단점이 존재 -> 문제에 따라 선택

다익스트라(Dijkstra) 알고리즘

- 특정 노드에서 출발하여 다른 모든 노드로 가는 최단 경로 계산
- 그리디(탐욕적) 알고리즘으로 분류
  ⇒ 매 상황에서 가장 비용이 적은 노드를 선택

참고: 기본적으로 최단 경로 알고리즘은 DP의 원리가 적용된다.

- 플로이드 알고리즘은 DP

다익스트라 알고리즘의 동작 과정

1. 출발 노드 설정
2. **최단 거리 테이블 초기화**
   - 자기 자신까지의 거리 = 0
   - 다른 모든 노드까지 가는 거리 = 무한
3. **방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드 선택**
   - 매 상황에서 가장 비용이 적은 노드를 선택
   - 한 단계당 하나의 노드에 대한 최단 거리를 확실히 찾음
   - 따라서 한 번 처리 된 노드의 최단 거리는 고정, 더 이상 변경 X
4. 해당 노드를 거쳐 다른 노드로 가는 비용을 계산하여 **최단 거리 테이블 갱신**
5. 3,4번 반복

최단 거리 테이블

- 각 노드에 대한 현재까지의 최단 거리 정보
- 처리 과정에서 더 짧은 경로를 찾으면 최단 거리 정보 갱신
- 다익스트라 알고리즘을 수행한 뒤에 테이블에는 각 노드까지의 최단 거리 정보가 저장
- 단, 최단 거리만 알 수 있고,
  최단 경로는 다른 로직이 추가로 필요
  (경로 구하는 문제 빈도 낮음)

노드 번호	1	2	3	4	5	6
거리	0	무한	무한	무한	무한	무한

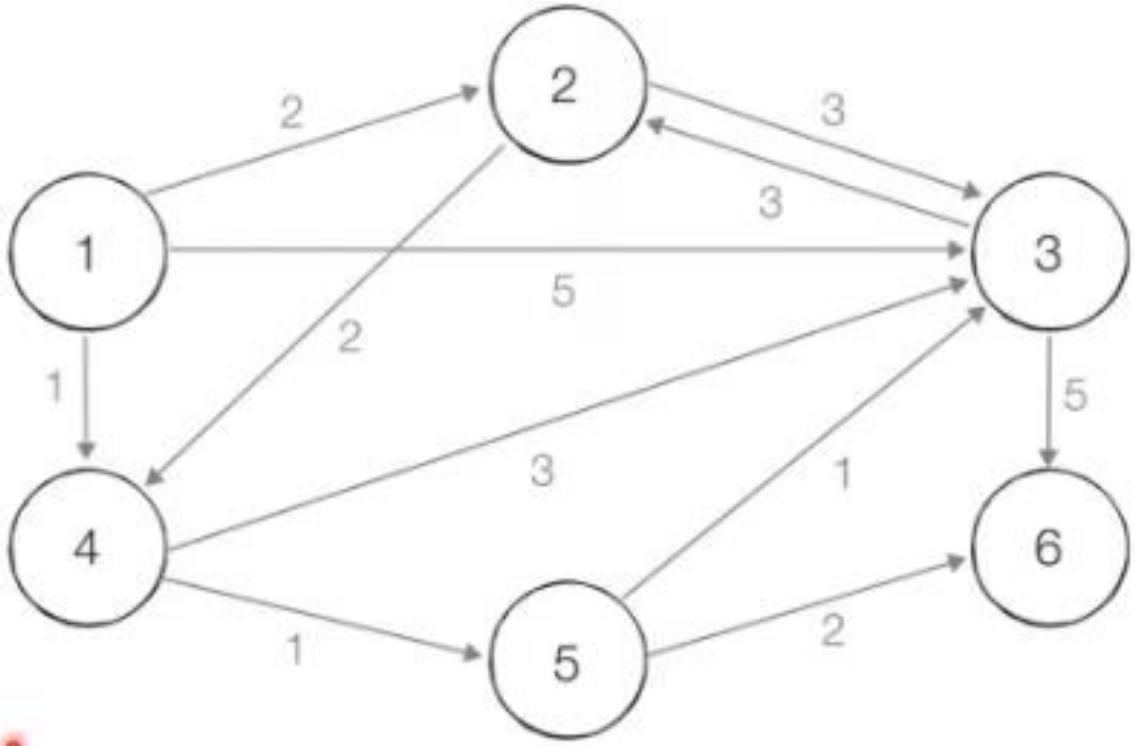
동작 과정 살펴보기

[초기 상태] 그래프를 준비하고 출발 노드를 설정합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드



노드 번호	1	2	3	4	5	6
거리	0	무한	무한	무한	무한	무한

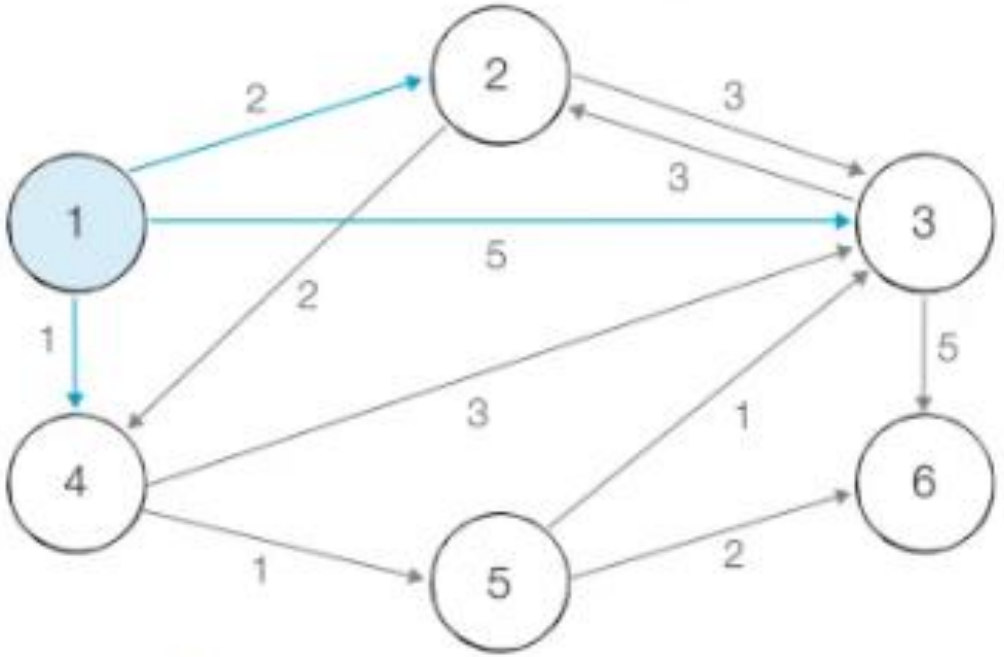
동작 과정 살펴보기

[Step 1] 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드인 **1번** 노드를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드



인접 노드	현재 값	거쳐갈 때	갱신 여부
2번	무한	0+2	True
3번	무한	0+5	True
4번	무한	0+1	True

노드 번호	1	2	3	4	5	6
거리	0	2	5	1	무한	무한

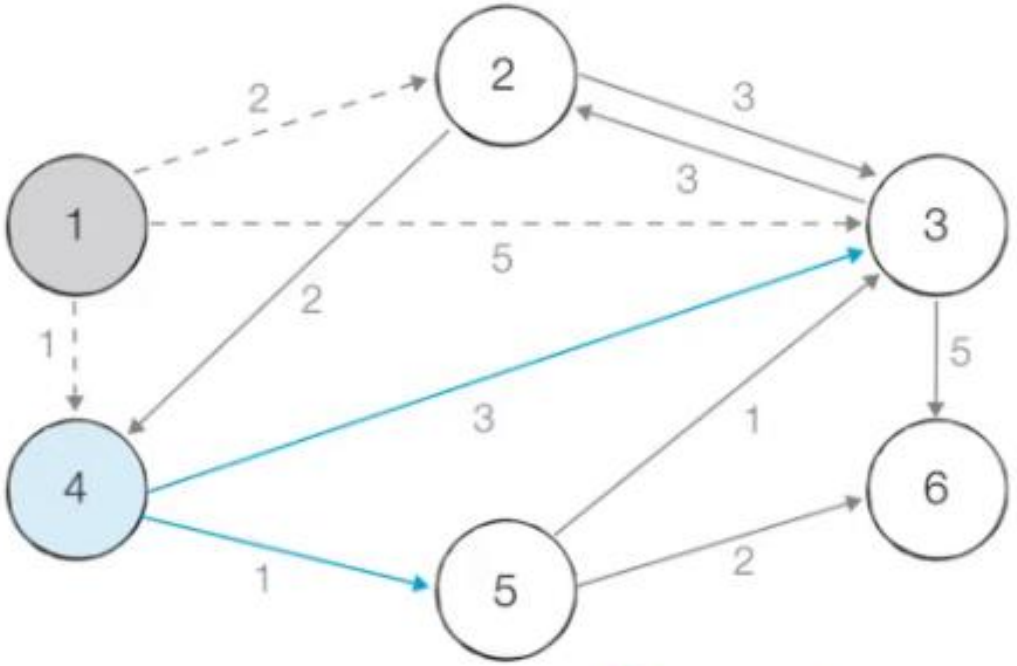
동작 과정 살펴보기

[Step 2] 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드인 **4번** 노드를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드



인접 노드	현재 값	거쳐갈 때	갱신 여부
3번	5	1+3	True
5번	무한	1+1	True

노드 번호	1	2	3	4	5	6
거리	0	2	4	1	2	무한

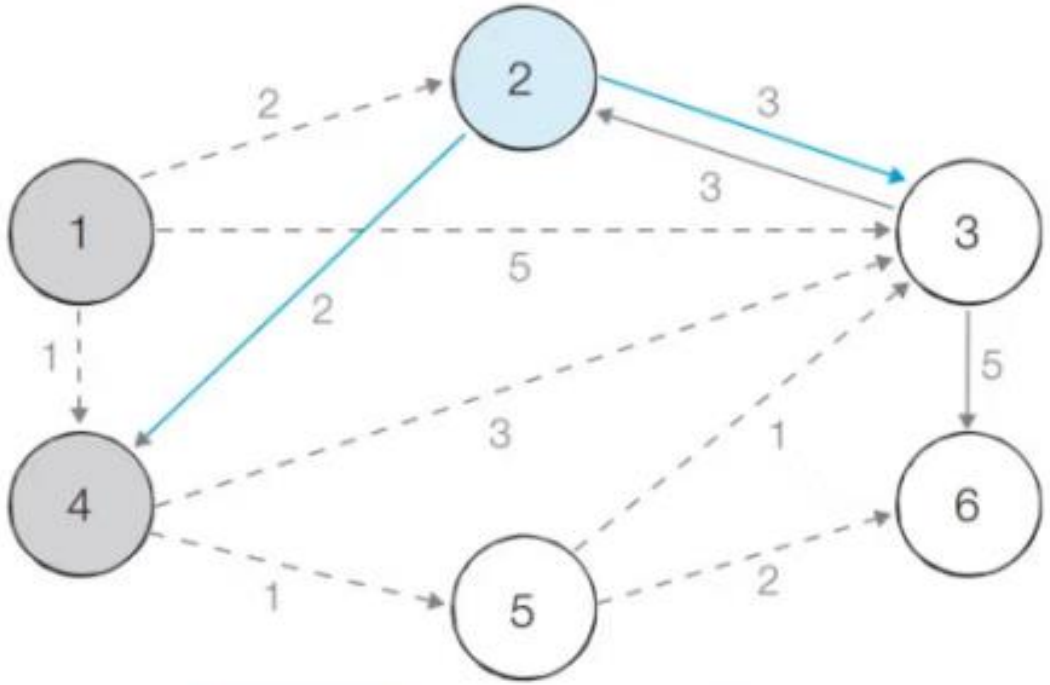
동작 과정 살펴보기

[Step 3] 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드인 **2번** 노드를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드



인접 노드	현재 값	거쳐갈 때	갱신 여부
3번	4	2+3	False
4번	1	2+2	False

노드 번호	1	2	3	4	5	6
거리	0	2	4	1	2	무한

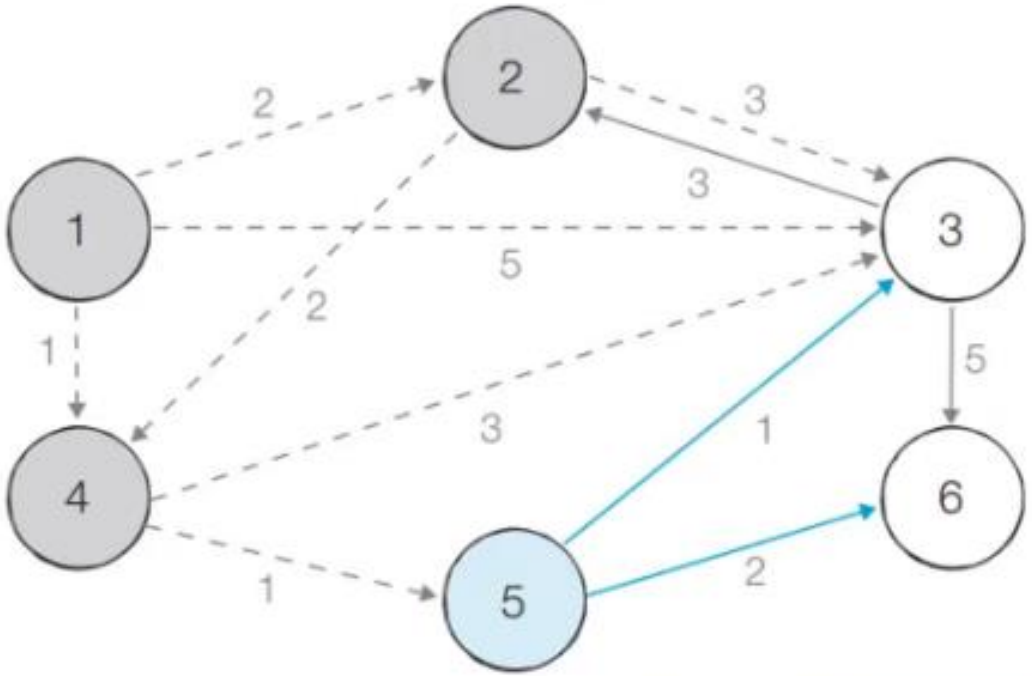
동작 과정 살펴보기

[Step 4] 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드인 **5번** 노드를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드



인접 노드	현재 값	거쳐갈 때	갱신 여부
3번	4	2+1	True
6번	무한	2+2	True

노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

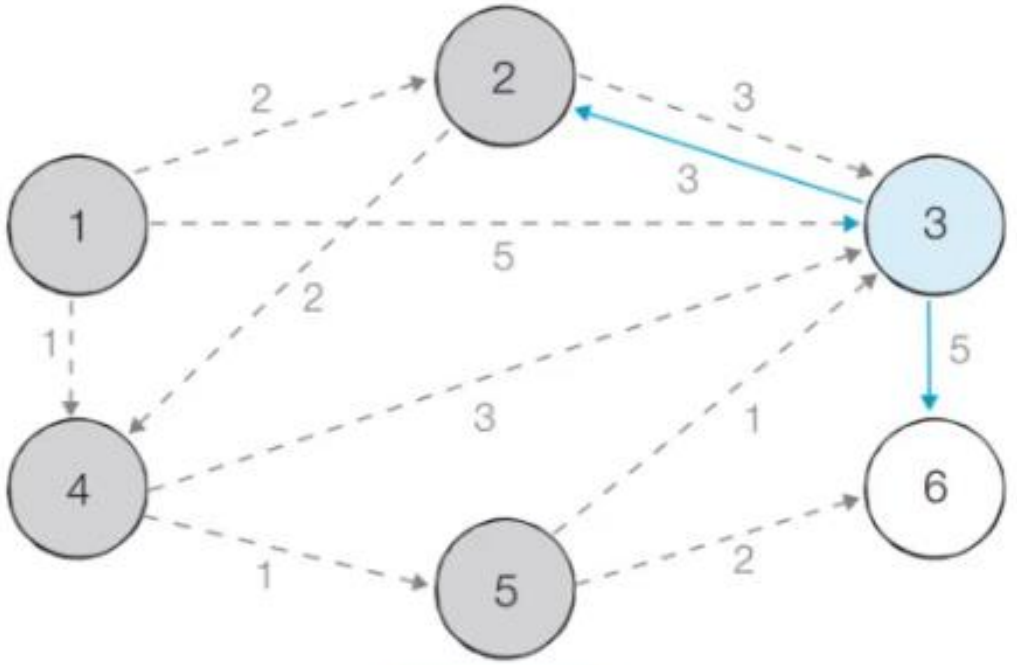
동작 과정 살펴보기

[Step 5] 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드인 **3번** 노드를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드



인접 노드	현재 값	거쳐갈 때	갱신 여부
2번	2	3+3	False
6번	4	3+5	False

노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

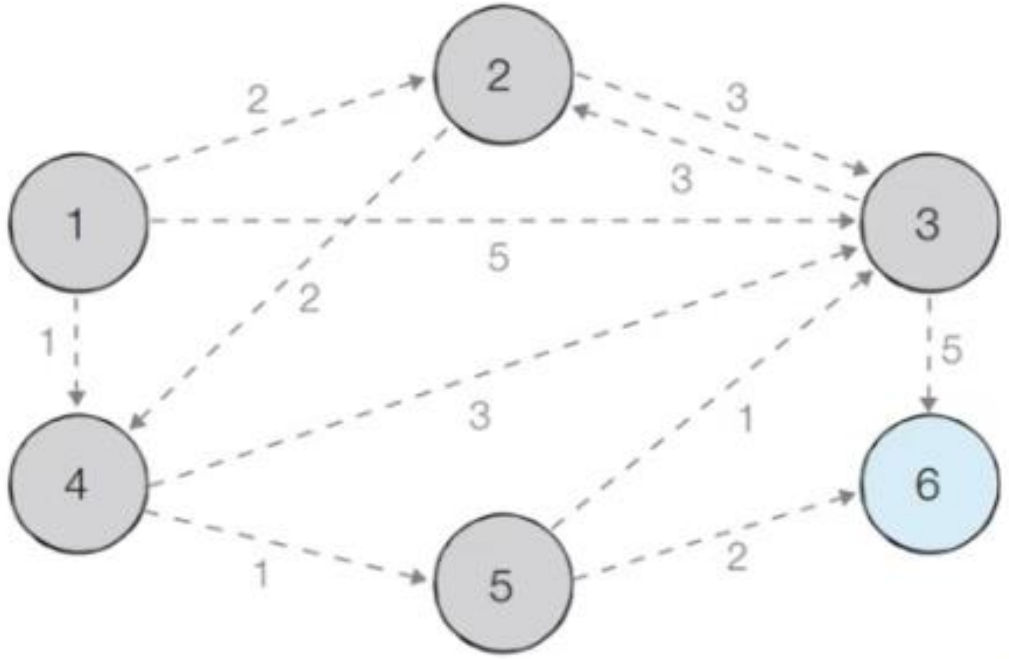
동작 과정 살펴보기

[Step 6] 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드인 **6번** 노드를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드



인접 노드	현재 값	거쳐갈 때	갱신 여부
이동할 수 있는 인접 노드가 없습니다.			

노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

다익스트라 알고리즘의 성능

- 노드의 개수 = V

- 총 O(V) 번에 걸쳐서 최단 거리가 가장 짧은 노드를 선형 탐색

⇒전체 시간 복잡도 $O(V^2)$

⇒V가 5,000개 이하일 때는 문제 X

⇒10,000개를 넘어간다면 시간 초과

따라서 우선순위 큐를 사용하여 성능을 개선할 수 있다.

다익스트라 알고리즘 : 우선순위 큐 활용

- Java에서는 Collection 클래스를 이용하여 PriorityQueue 사용
- 우선 순위 큐를 구현한 Heap의 삽입, 삭제 시간 복잡도는 O(logn) 이므로 선형 시간보다 작다.

동작과정

- 다익스트라 알고리즘이 동작하는 기본 원리는 동일
- 즉, 노드를 거쳐가면서 최단 거리를 갱신하는 방법 동일
- 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드를 선택하는 방법에 힙 자료구조를 이용한다는 점이 다름

다익스트라 알고리즘 : 우선순위 큐 활용

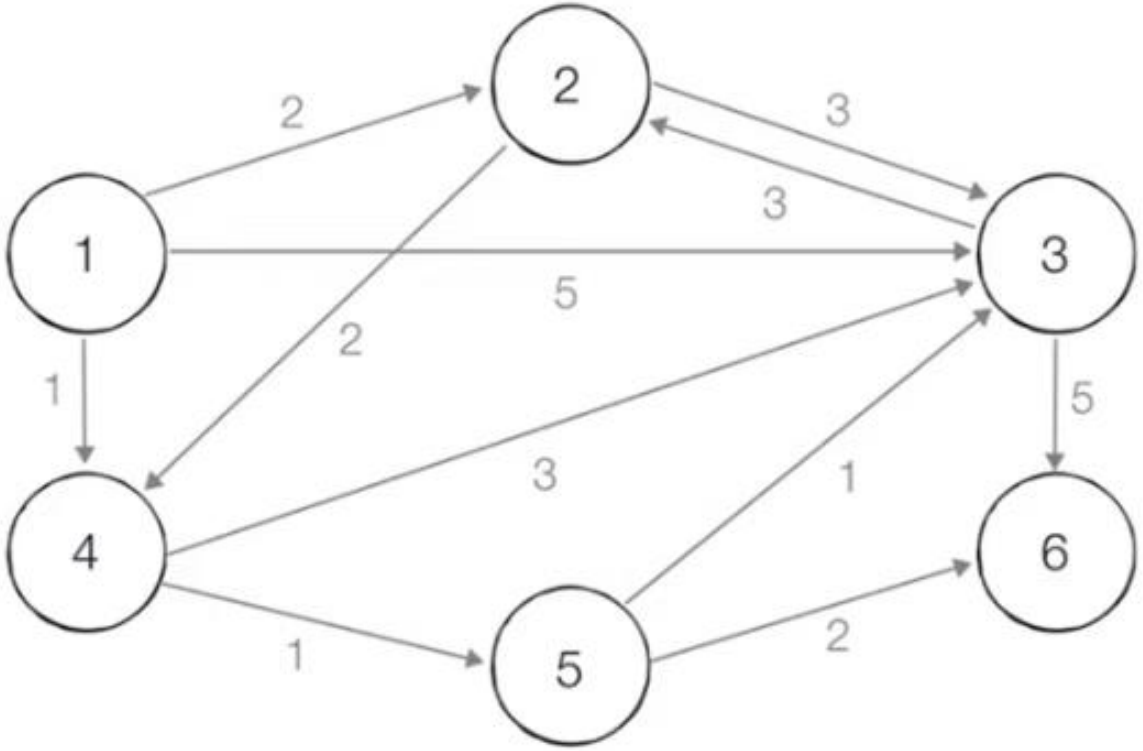
[초기 상태] 그래프를 준비하고 출발 노드를 설정하여 우선순위 큐에 삽입합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드

우선순위 큐
(거리: 0, 노드: 1)



노드 번호	1	2	3	4	5	6
거리	0	무한	무한	무한	무한	무한

다익스트라 알고리즘 : 우선순위 큐 활용

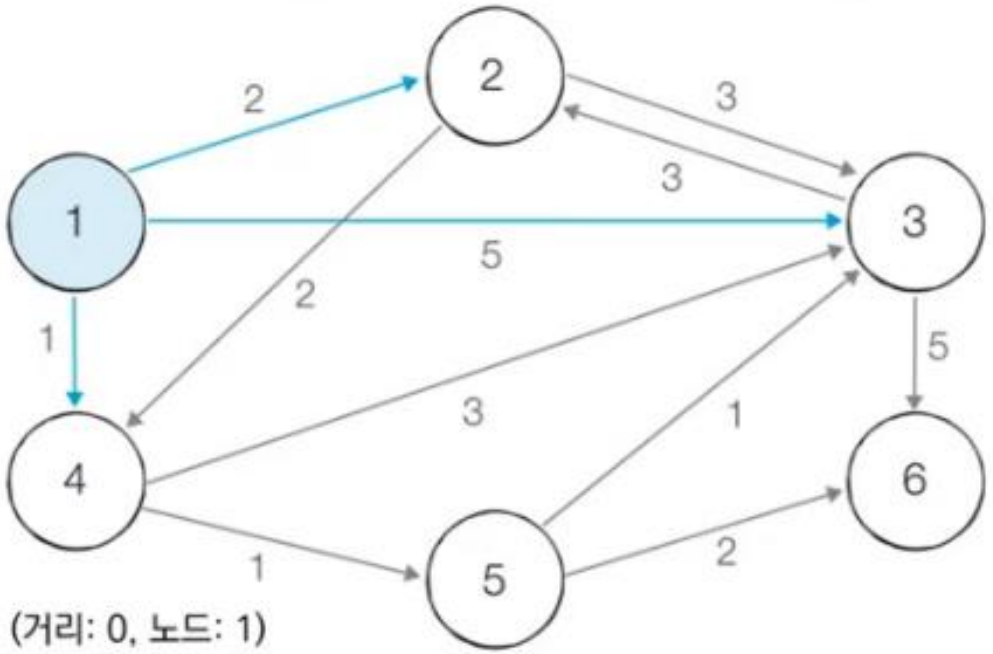
- [Step 1] 우선순위 큐에서 원소를 꺼냅니다. **1번** 노드는 아직 방문하지 않았으므로 이를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드

우선순위 큐
(거리: 1, 노드: 4)
(거리: 2, 노드: 2)
(거리: 5, 노드: 3)



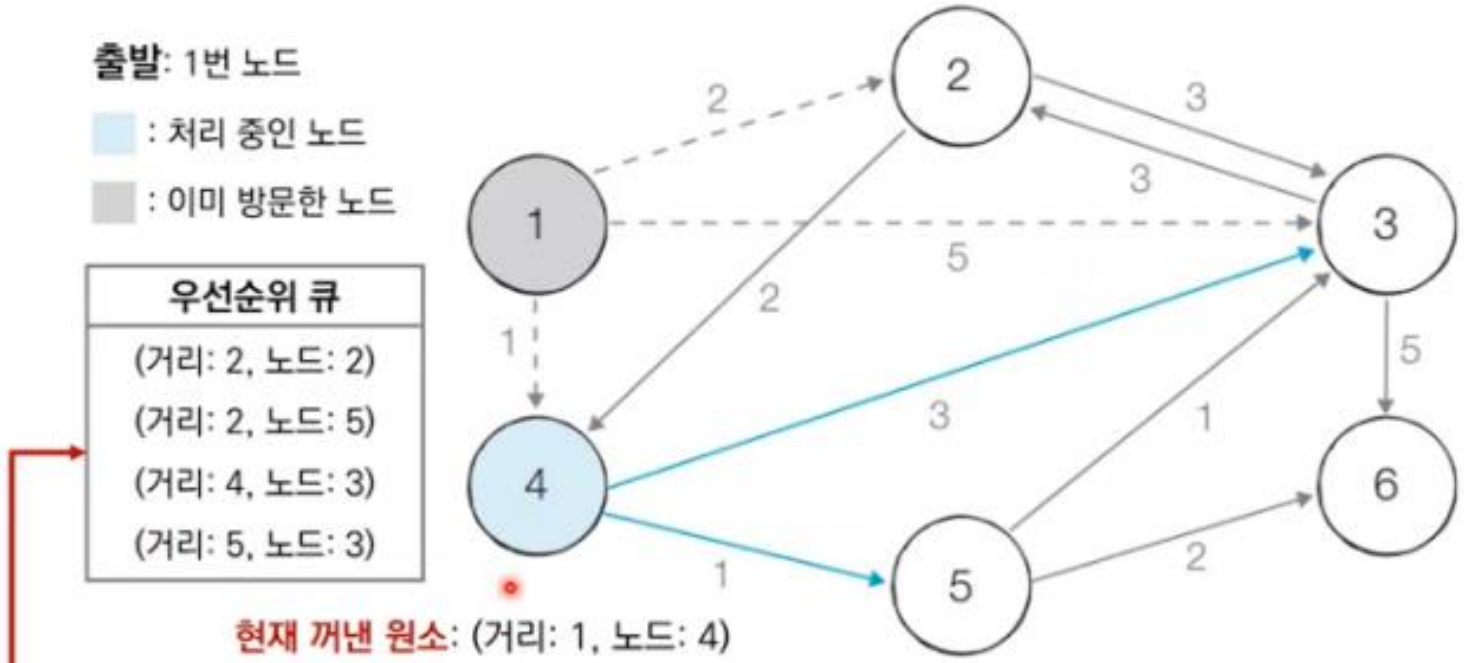
인접 노드	현재 값	거쳐갈 때	갱신 여부
2번	무한	0+2	True
3번	무한	0+5	True
4번	무한	0+1	True

현재 꺼낸 원소: (거리: 0, 노드: 1)

노드 번호	1	2	3	4	5	6
거리	0	2	5	1	무한	무한

다익스트라 알고리즘 : 우선순위 큐 활용

- **[Step 2]** 우선순위 큐에서 원소를 꺼냅니다. **4번** 노드는 아직 방문하지 않았으므로 이를 처리합니다.

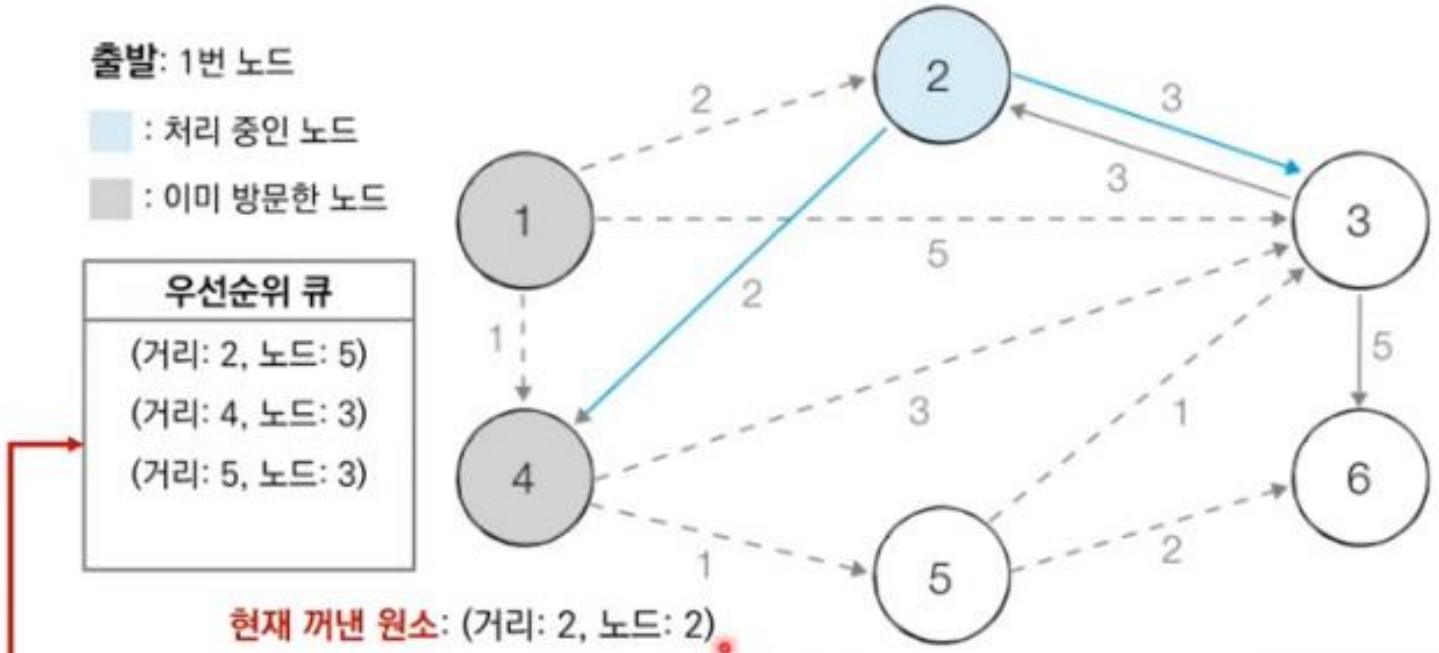


인접 노드	현재 값	거쳐갈 때	갱신 여부
3번	5	1+3	True
5번	무한	1+1	True

노드 번호	1	2	3	4	5	6
거리	0	2	4	1	2	무한

다익스트라 알고리즘 : 우선순위 큐 활용

- **[Step 3]** 우선순위 큐에서 원소를 꺼냅니다. **2번** 노드는 아직 방문하지 않았으므로 이를 처리합니다.



인접 노드	현재 값	거쳐갈 때	갱신 여부
3번	4	2+3	False
4번	1	2+2	False

노드 번호	1	2	3	4	5	6
거리	0	2	4	1	2	무한

다익스트라 알고리즘 : 우선순위 큐 활용

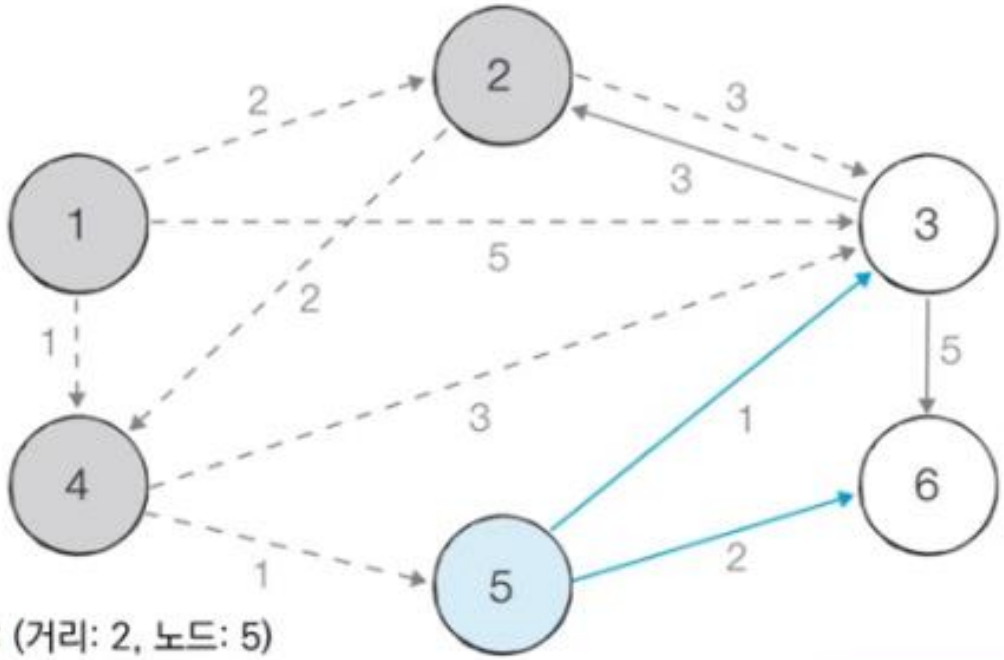
[Step 4] 우선순위 큐에서 원소를 꺼냅니다. **5번** 노드는 아직 방문하지 않았으므로 이를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드

우선순위 큐
(거리: 3, 노드: 3)
(거리: 4, 노드: 3)
(거리: 4, 노드: 6)
(거리: 5, 노드: 3)



인접 노드	현재 값	거쳐갈 때	갱신 여부
3번	4	2+1	True
6번	무한	2+2	True

현재 꺼낸 원소: (거리: 2, 노드: 5)

노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

다익스트라 알고리즘 : 우선순위 큐 활용

[Step 5] 우선순위 큐에서 원소를 꺼냅니다. **3번** 노드는 아직 방문하지 않았으므로 이를 처리합니다.

출발: 1번 노드

: 처리 중인 노드

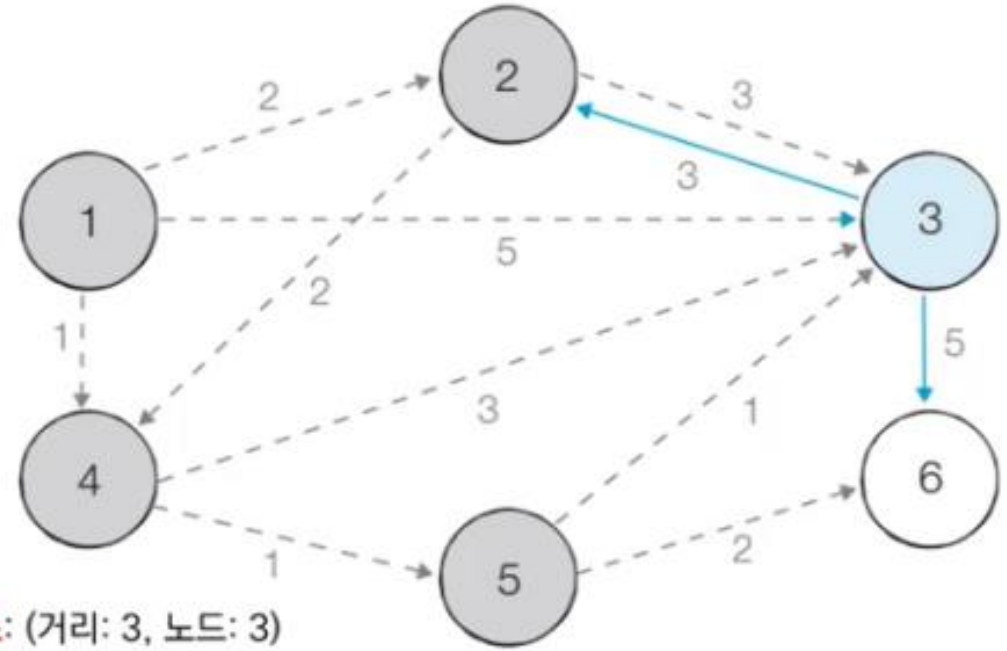
: 이미 방문한 노드

우선순위 큐

(거리: 4, 노드: 3)

(거리: 4, 노드: 6)

(거리: 5, 노드: 3)



인접 노드	현재 값	거쳐갈 때	갱신 여부
2번	2	3+3	False
6번	4	3+5	False

현재 꺼낸 원소: (거리: 3, 노드: 3)

노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

다익스트라 알고리즘 : 우선순위 큐 활용

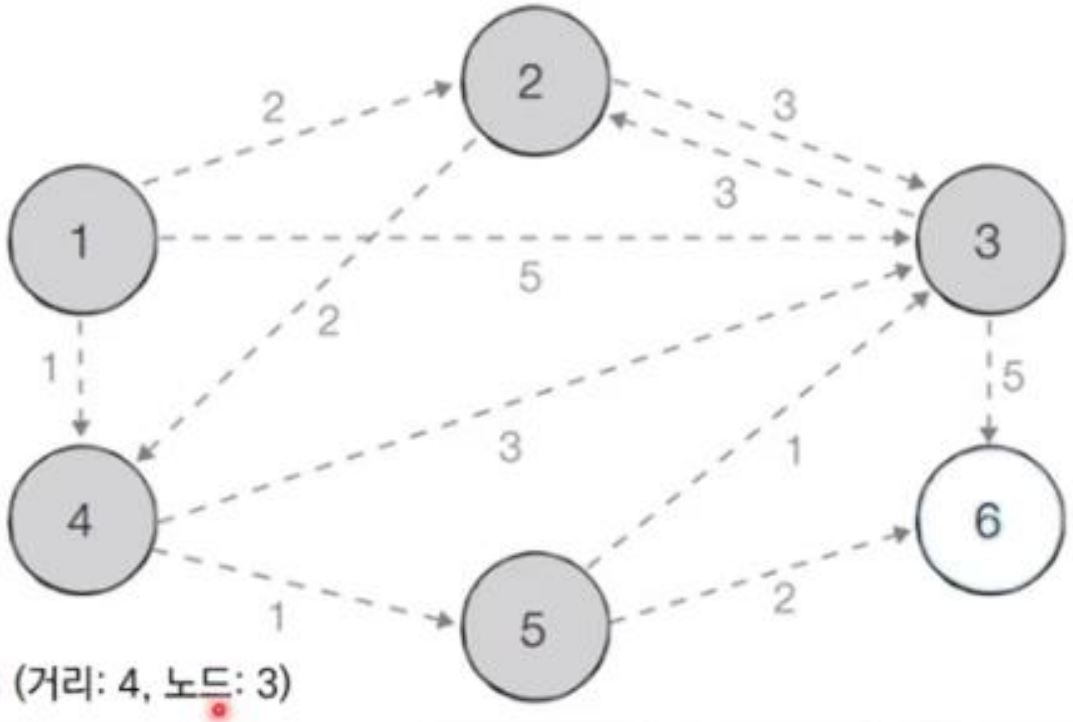
[Step 6] 우선순위 큐에서 원소를 꺼냅니다. **3번** 노드는 이미 방문했으므로 무시합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드

우선순위 큐
(거리: 4, 노드: 6)
(거리: 5, 노드: 3)

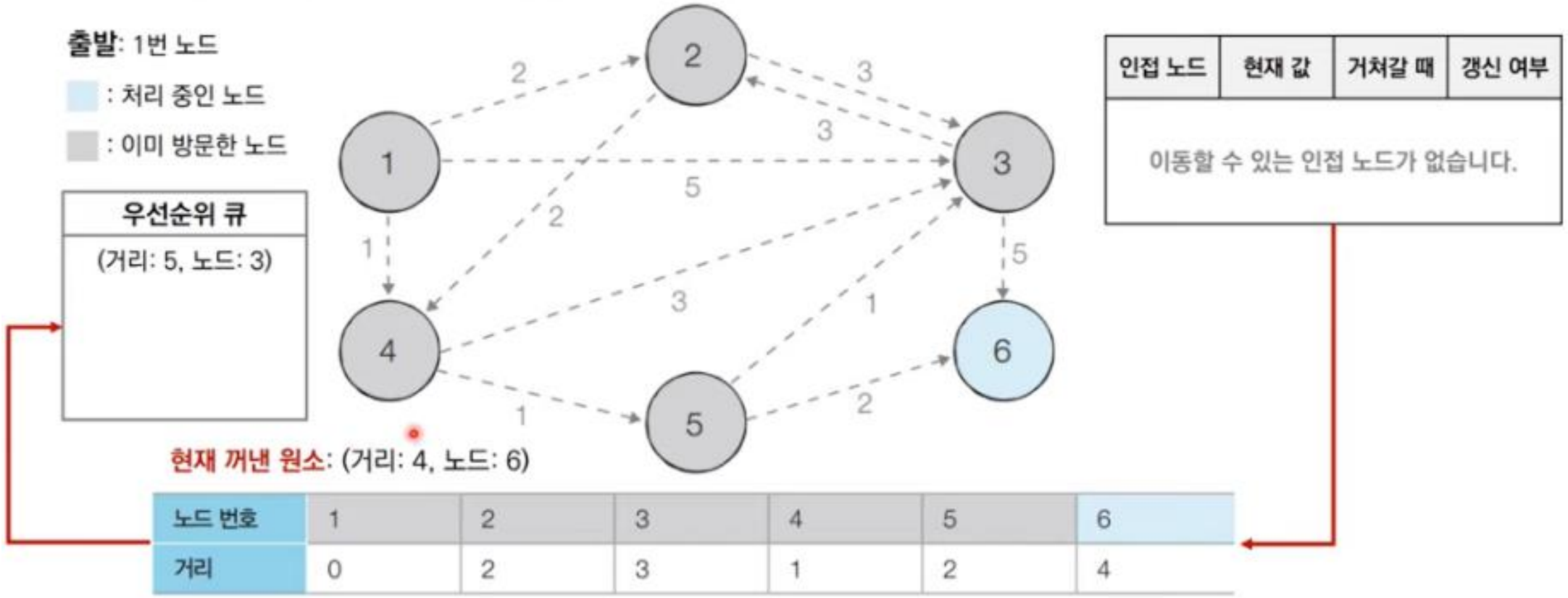


현재 꺼낸 원소: (거리: 4, 노드: 3)

노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

다익스트라 알고리즘 : 우선순위 큐 활용

[Step 7] 우선순위 큐에서 원소를 꺼냅니다. **6번** 노드는 아직 방문하지 않았으므로 이를 처리합니다.



노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

다익스트라 알고리즘 : 우선순위 큐 활용

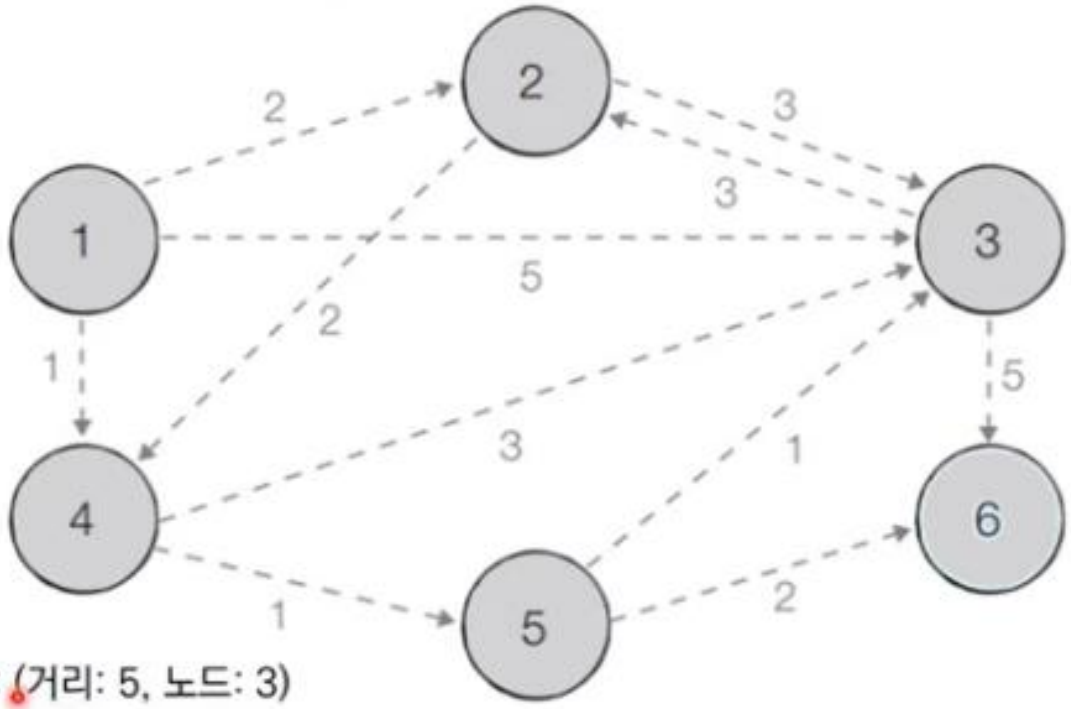
[Step 8] 우선순위 큐에서 원소를 꺼냅니다. **3번** 노드는 이미 방문했으므로 무시합니다.

출발: 1번 노드

: 처리 중인 노드

: 이미 방문한 노드

우선순위 큐



현재 꺼낸 원소: (거리: 5, 노드: 3)

노드 번호	1	2	3	4	5	6
거리	0	2	3	1	2	4

우선순위 큐 활용한 다익스트라 알고리즘 성능

- 노드의 개수 = V
- 최대 간선의 개수 = E

- 시간 복잡도는 O(ElogV)

플로이드 워셜(Floyd warshall) 알고리즘

- 모든 노드에서 다른 모든 노드까지의 최단 경로를 모두 계산
- 다익스트라 알고리즘과 마찬가지로 단계별로 거쳐 가는 노드를 기준으로 알고리즘을 수행
  - 다른 점 : 매 단계마다 방문하지 않은 노드 중에 최단 거리를 갖는 노드를 찾는 과정이 필요X
- 2차원 테이블에 최단 거리 정보 저장
- 다이나믹 프로그래밍 유형
- 시간 복잡도 $O(V^3)$

플로이드 워셜(Floyd warshall) 알고리즘

- 각 단계마다 특정한 노드 k를 거쳐 가는 경우를 확인
  - a에서 b로 가는 최단 거리보다
  - a에서 k를 거쳐 b로 가는 거리가 더 짧은지를 검사

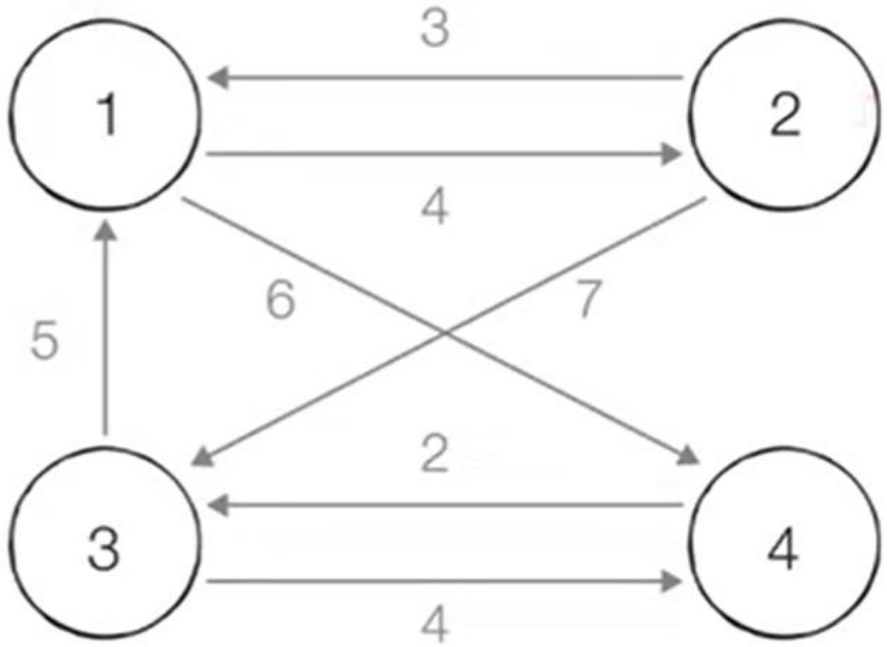
- 점화식

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

플로이드 워셜알고리즘 동작 과정

[초기 상태] 그래프를 준비하고 최단 거리 테이블을 초기화합니다.

- **기본 점화식**: $D_{ab} = \min(D_{ab}, D_{ak} + D_{kb})$

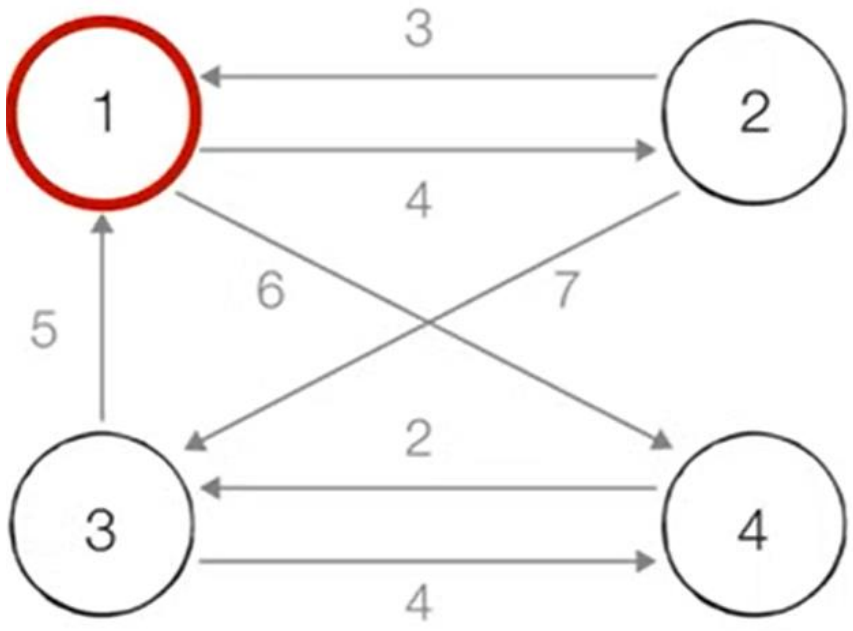


출발 \ 도착	1번	2번	3번	4번
1번	0	4	무한	6
2번	3	0	7	무한
3번	5	무한	0	4
4번	무한	무한	2	0

플로이드 워셜알고리즘 동작 과정

- K = 1

- [Step 1] 1번 노드를 거쳐 가는 경우를 고려하여 테이블을 갱신합니다.
  - **점화식**: $D_{ab} = \min(D_{ab}, D_{a1} + D_{1b})$



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

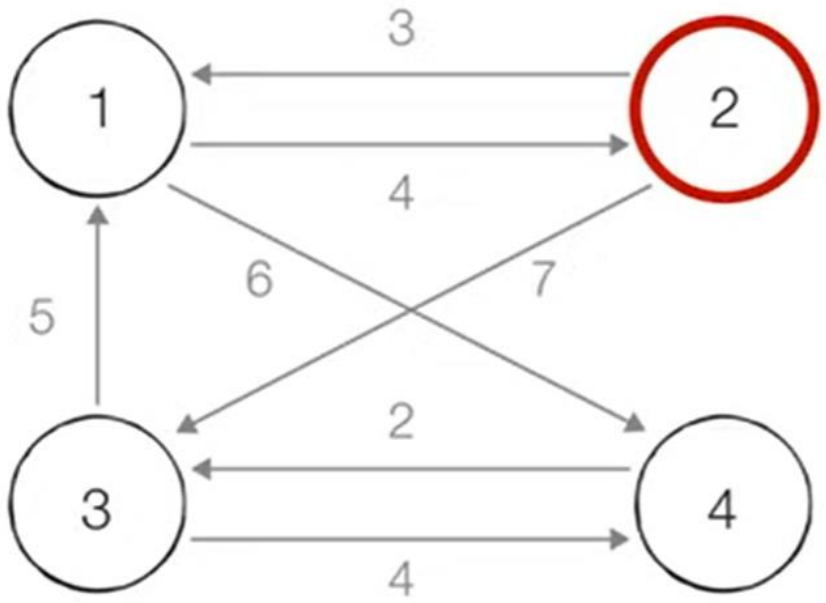
갱신된 최단 거리 테이블

0	4	무한	6
3	0	7	9
5	9	0	4
무한	무한	2	0

플로이드 워셜알고리즘 동작 과정

- K = 2

- [Step 2] 2번 노드를 거쳐 가는 경우를 고려하여 테이블을 갱신합니다.
  - 점화식: $D_{ab} = \min(D_{ab}, D_{a2} + D_{2b})$



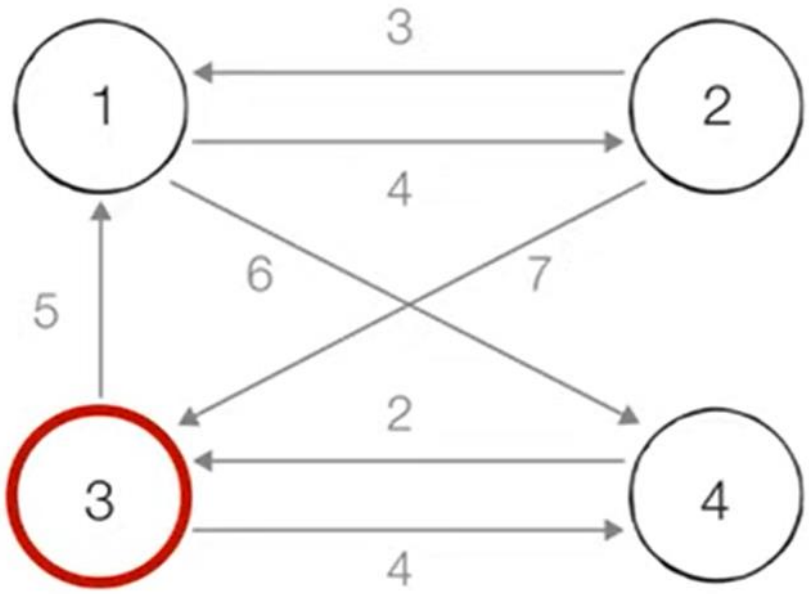
$$D_{13} = \min(D_{13}, D_{12} + D_{23})$$
$$D_{14} = \min(D_{14}, D_{12} + D_{24})$$
$$D_{31} = \min(D_{31}, D_{32} + D_{21})$$
$$D_{34} = \min(D_{34}, D_{32} + D_{24})$$
$$D_{41} = \min(D_{41}, D_{42} + D_{21})$$
$$D_{43} = \min(D_{43}, D_{42} + D_{23})$$

갱신된 최단 거리 테이블

0	4	11	6
3	0	7	9
5	9	0	4
무한	무한	2	0

플로이드 워셜알고리즘 동작 과정

- K = 3
- **[Step 3] 3번** 노드를 거쳐 가는 경우를 고려하여 테이블을 갱신합니다.
  - **점화식**: $D_{ab} = \min(D_{ab}, D_{a3} + D_{3b})$



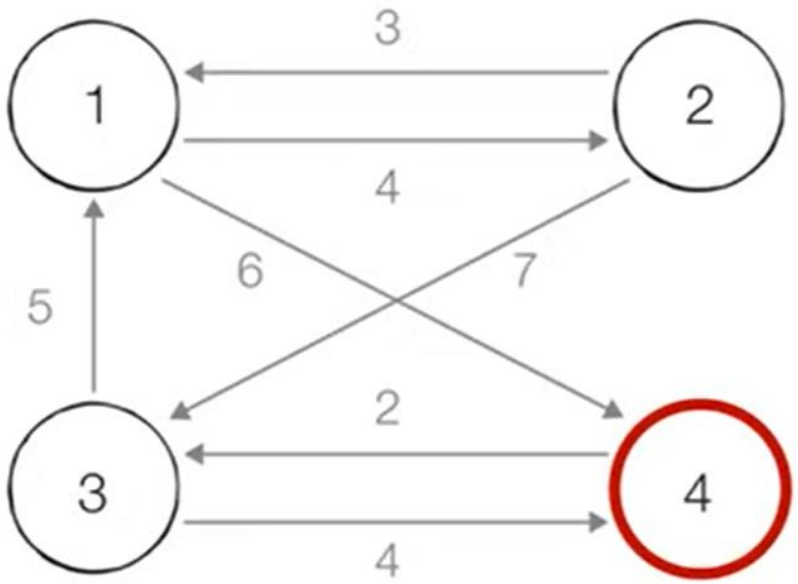
$$D_{12} = \min(D_{12}, D_{13} + D_{32})$$
$$D_{14} = \min(D_{14}, D_{13} + D_{34})$$
$$D_{21} = \min(D_{21}, D_{23} + D_{31})$$
$$D_{24} = \min(D_{24}, D_{23} + D_{34})$$
$$D_{41} = \min(D_{41}, D_{43} + D_{31})$$
$$D_{42} = \min(D_{42}, D_{43} + D_{32})$$

갱신된 최단 거리 테이블

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

플로이드 워셜알고리즘 동작 과정

- K = 4
- [Step 4] 4번 노드를 거쳐 가는 경우를 고려하여 테이블을 갱신합니다.
  - **점화식**: $D_{ab} = \min(D_{ab}, D_{a4} + D_{4b})$



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

갱신된 최단 거리 테이블

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

위상 정렬

위상

: 어떤 사물이 다른 사물과의 관계 속에서 가지는 위치나 상태

위상 정렬

: 각 노드들이 가지는 위상(간선)에 따라서 그래프를 구성하는 정점들을 순서대로 정렬

즉, 모든 노드를 방향성에 거스르지 않도록 순서대로 나열

위상 정렬 조건

- 사이클이 없는 방향 그래프 (DAG)
- 간선이 방향성을 가지는 그래프

위상 정렬

진입 차수(Indegree) : 특정한 노드로 들어오는 간선의 개수
진출 차수(Outdegree) : 특정한 노드에서 나가는 간선의 개수

진입차수 = 0
진출차수 = 2

진입차수 = 1
진출차수 = 1

진입차수 = 2
진출차수 = 0

위상 정렬 알고리즘

큐를 이용하는 위상 정렬 알고리즘 동작과정

1. 진입 차수가 0인 모든 노드를 큐에 넣는다.
2. 큐가 빌 때까지 다음의 과정을 반복한다.
    1) 큐에서 원소를 꺼내 해당 노드에서 나가는 간선을 그래프에서 제거한다.
    2) 새롭게 진입 차수가 0이 된 노드를 큐에 넣는다.

⇒ 각 노드가 큐에 들어온 순서가 위상 정렬을 수행한 결과

위상 정렬 알고리즘

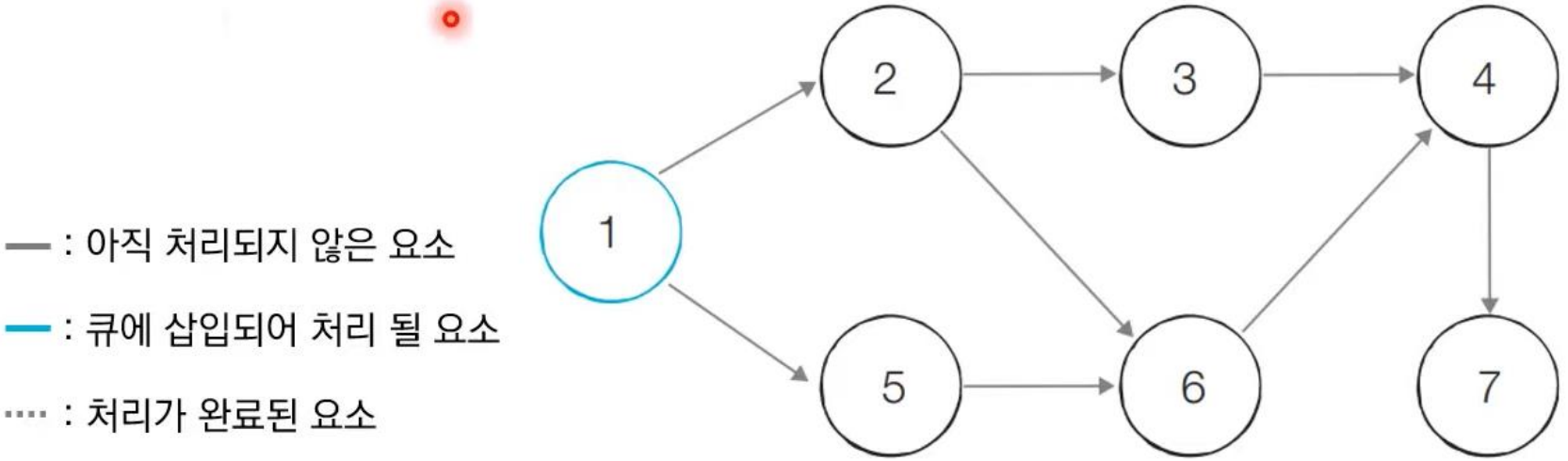
- 위상 정렬은 DAG에 대해서만 수행할 수 있다.
  - DAG (Direct Acyclic Graph) : 순환하지 않는 방향 그래프
- **순환하게 되면 진입 차수가 0인 노드가 존재하지 않기 때문이다**.
- **모든 원소를 방문하기 전에 큐가 빈다면 사이클이 존재**한다고 판단할 수 있다.

- 위상 정렬에서는 여러 가지 답이 존재
  - 큐에 새롭게 들어가는 원소가 한 단계에서 2개 이상 존재하는 경우

위상 정렬 알고리즘

[초기 단계] 초기 단계에서는 **진입차수가 0인 모든 노드**를 큐에 넣습니다.

- 처음에 **노드 1**이 큐에 삽입됩니다.



노드	1	2	3	4	5	6	7
진입차수	0	1	1	2	1	2	1

큐	노드 1

위상 정렬 알고리즘

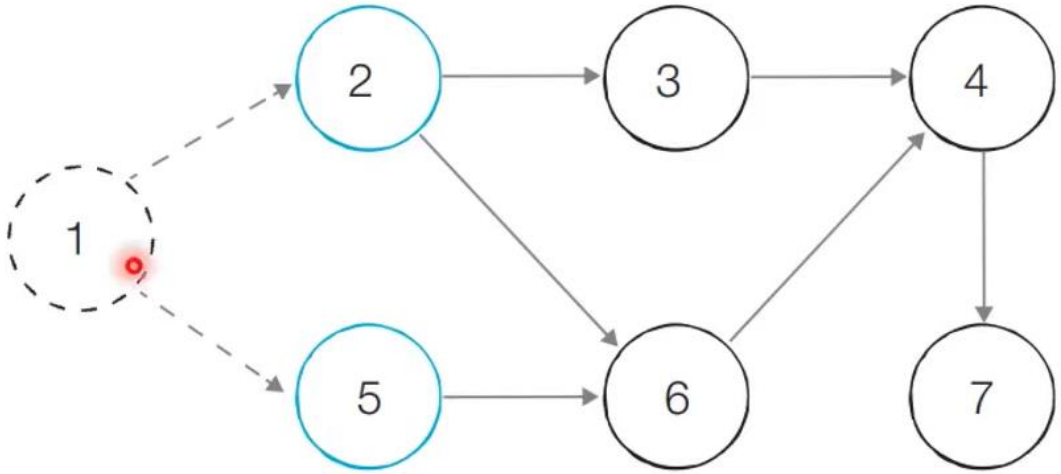
[Step 1] 큐에서 **노드 1**을 꺼낸 뒤에 **노드 1**에서 나가는 간선을 제거합니다.

- 새롭게 **진입차수가 0이 된 노드**들을 큐에 삽입합니다.

— : 아직 처리되지 않은 요소

— : 큐에 삽입되어 처리 될 요소

···· : 처리가 완료된 요소



노드	1	2	3	4	5	6	7
진입차수	0	0	1	2	0	2	1

큐	노드 2, 노드 5

위상 정렬 알고리즘

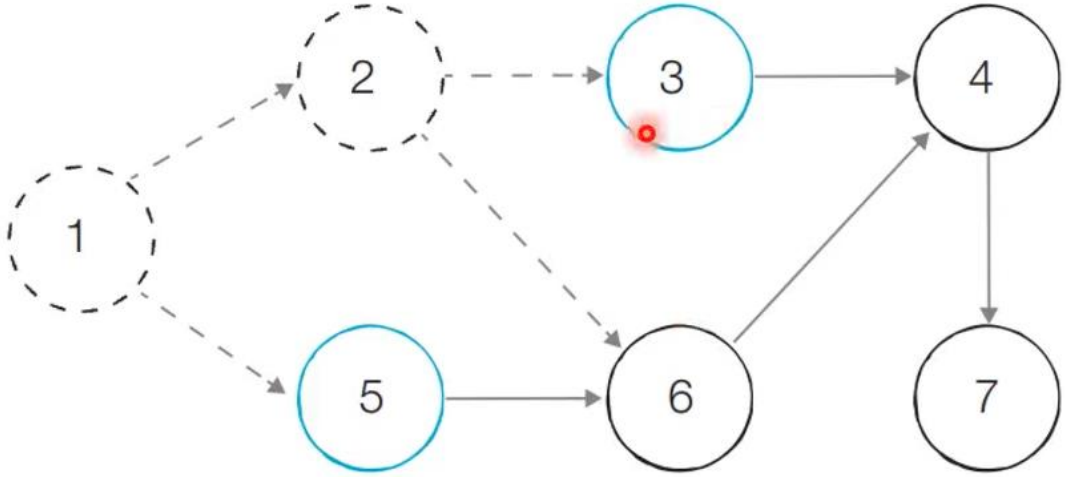
[Step 2] 큐에서 **노드 2**를 꺼낸 뒤에 **노드 2**에서 나가는 간선을 제거합니다.

- 새롭게 **진입차수가 0이 된 노드**를 큐에 삽입합니다.

— : 아직 처리되지 않은 요소

— : 큐에 삽입되어 처리 될 요소

···· : 처리가 완료된 요소



노드	1	2	3	4	5	6	7
진입차수	0	0	0	2	0	1	1

큐	노드 5, 노드 3

위상 정렬 알고리즘

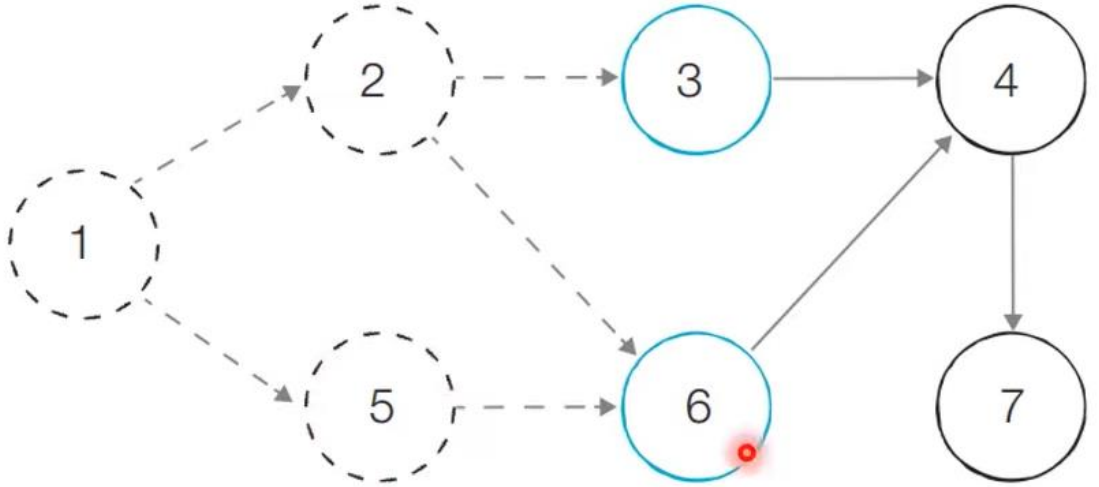
[Step 3] 큐에서 **노드 5**를 꺼낸 뒤에 **노드 5**에서 나가는 간선을 제거합니다.

- 새롭게 **진입차수가 0이 된 노드**를 큐에 삽입합니다.

— : 아직 처리되지 않은 요소

— : 큐에 삽입되어 처리 될 요소

···· : 처리가 완료된 요소



노드	1	2	3	4	5	6	7
진입차수	0	0	0	2	0	0	1

큐	노드 3, 노드 6

위상 정렬 알고리즘

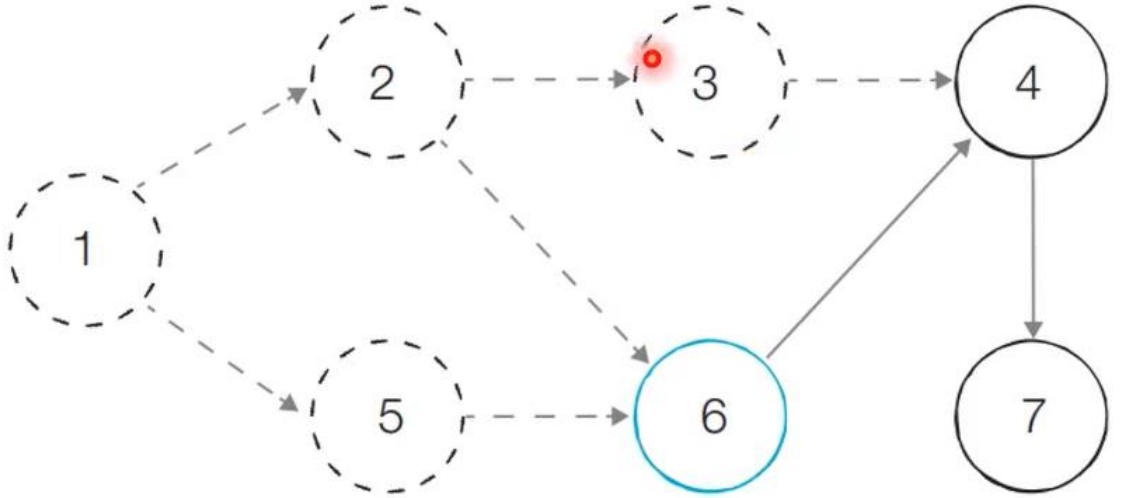
[Step 4] 큐에서 **노드 3**를 꺼낸 뒤에 **노드 3**에서 나가는 간선을 제거합니다.

- 새롭게 **진입차수가 0이 된 노드**가 없으므로 그냥 넘어갑니다.

— : 아직 처리되지 않은 요소

— : 큐에 삽입되어 처리 될 요소

···· : 처리가 완료된 요소



노드	1	2	3	4	5	6	7
진입차수	0	0	0	1	0	0	1

큐	노드 6

위상 정렬 알고리즘

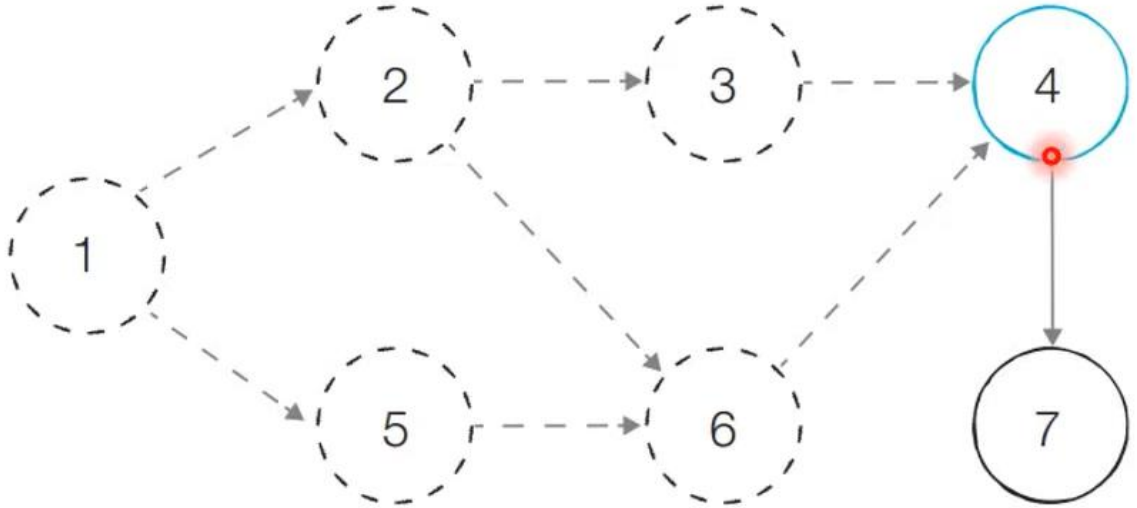
[Step 5] 큐에서 **노드 6**를 꺼낸 뒤에 **노드 6**에서 나가는 간선을 제거합니다.

- 새롭게 **진입차수가 0이 된 노드**를 큐에 삽입합니다.

— : 아직 처리되지 않은 요소

— : 큐에 삽입되어 처리 될 요소

···· : 처리가 완료된 요소



노드	1	2	3	4	5	6	7
진입차수	0	0	0	0	0	0	1

큐	노드 4

위상 정렬 알고리즘

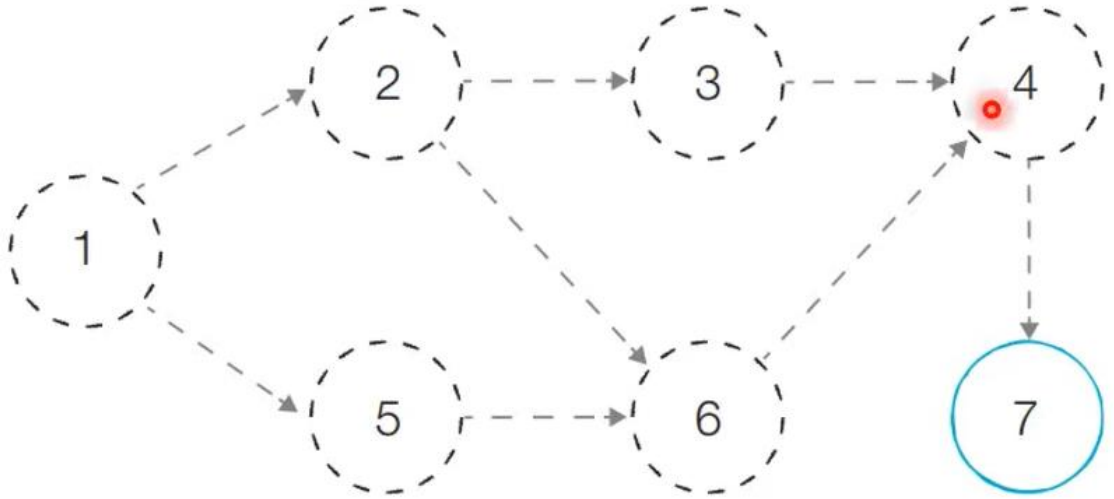
[Step 6] 큐에서 **노드 4**를 꺼낸 뒤에 **노드 4**에서 나가는 간선을 제거합니다.

- 새롭게 **진입차수가 0이 된 노드**를 큐에 삽입합니다.

— : 아직 처리되지 않은 요소

— : 큐에 삽입되어 처리 될 요소

···· : 처리가 완료된 요소



노드	1	2	3	4	5	6	7
진입차수	0	0	0	0	0	0	0

큐	노드 7

위상 정렬 알고리즘

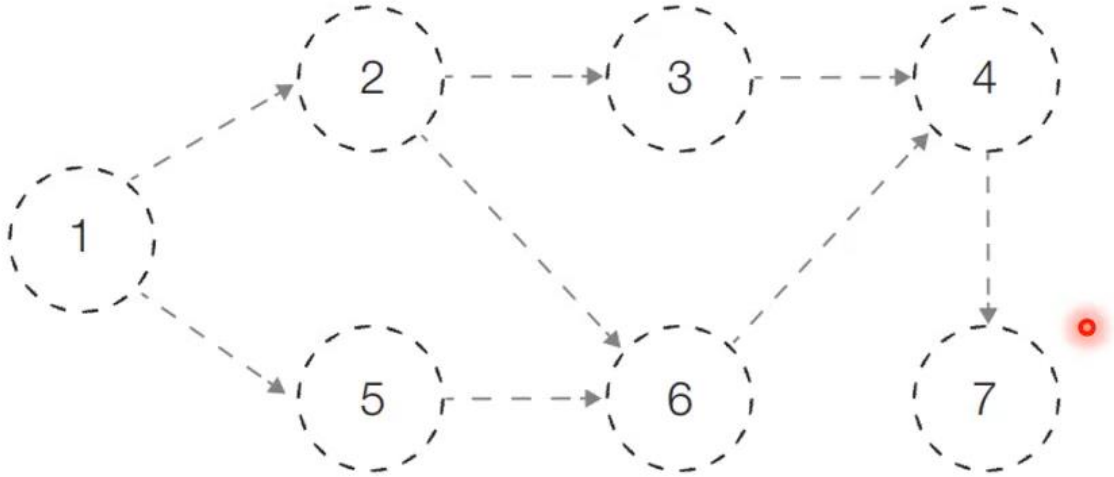
[Step 7] 큐에서 **노드 7**을 꺼낸 뒤에 **노드 7**에서 나가는 간선을 제거합니다.

- 새롭게 **진입차수가 0이 된 노드**가 없으므로 그냥 넘어갑니다.

— : 아직 처리되지 않은 요소

— : 큐에 삽입되어 처리 될 요소

···· : 처리가 완료된 요소



노드	1	2	3	4	5	6	7
진입차수	0	0	0	0	0	0	0

큐	

위상 정렬 알고리즘

[위상 정렬 결과]

- 큐에 삽입된 전체 노드 순서: **1 → 2 → 5 → 3 → 6 → 4 → 7**