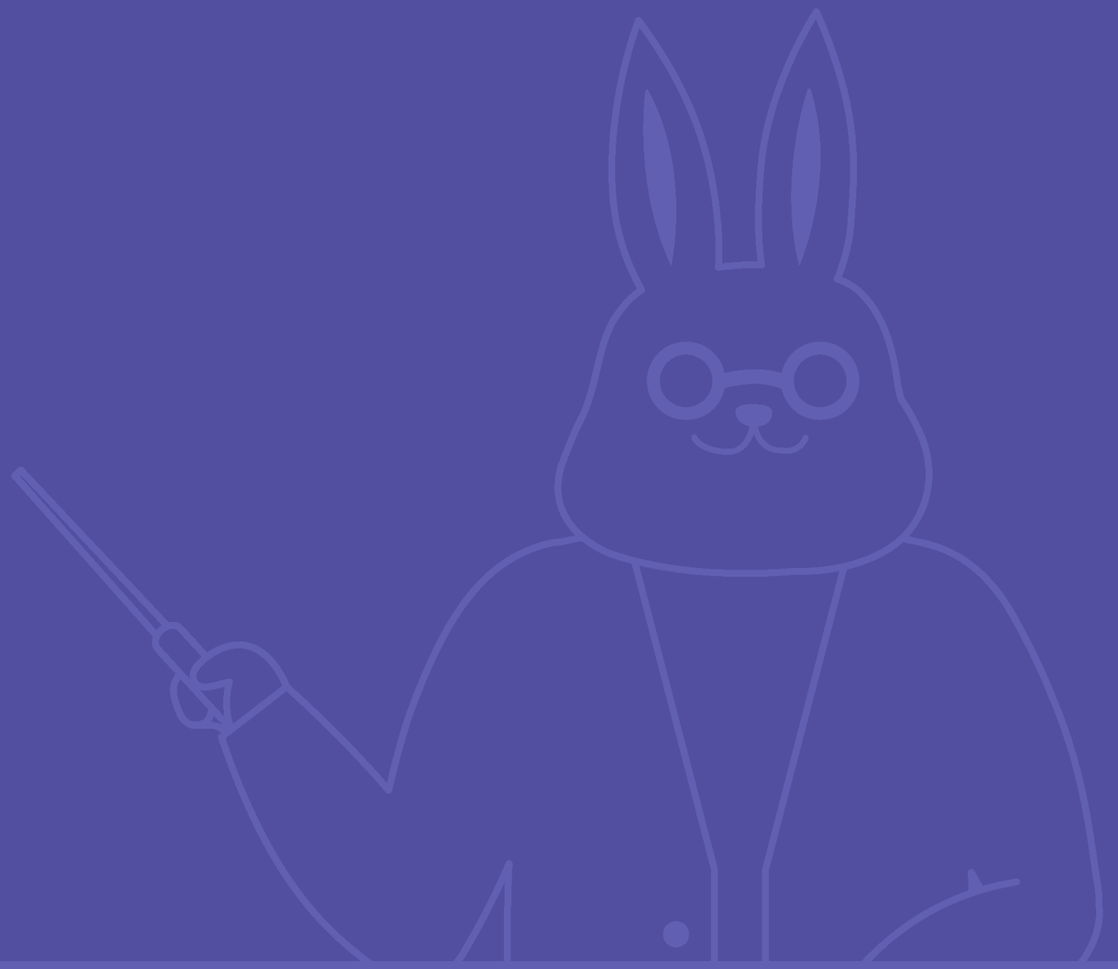
알고리즘의 정석 II

4장 그래프 알고리즘 심화



/* elice */

Contents

그래프 알고리즘의 활용

다양한 문제를 **그래프**로 만들어서 **동일한 문제**로 치환할 수 있습니다.

- 도시 간 최소이사비용, 최단등교시간 길 구하기

➔ 정점간의 최소거리를 구하는 문제로 바꾸어 풀 수 있습니다.

그래프 알고리즘의 활용

다양한 문제를 **그래프**로 만들어서 **동일한 문제**로 치환할 수 있습니다.

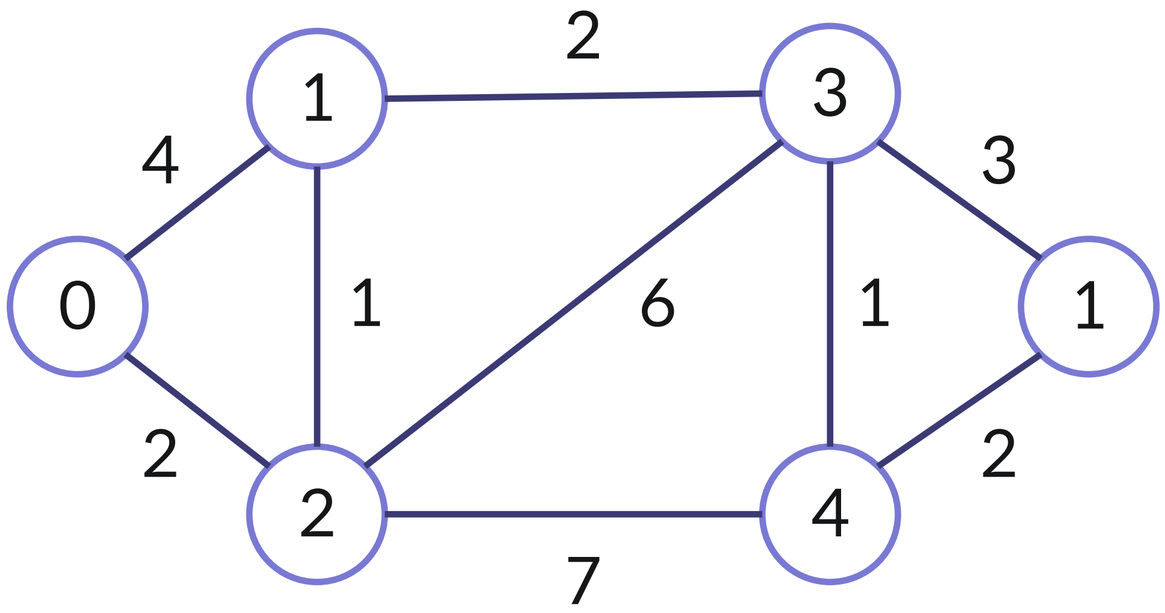
- 도시 간 최소이사비용, 최단등교시간 길 구하기

➔ 정점간의 최소거리를 구하는 문제로 바꾸어 풀 수 있습니다.

다양한 문제를 푸는 정형화된 대표 알고리즘으로는
최단거리알고리즘과 **최소신장트리**가 있습니다.

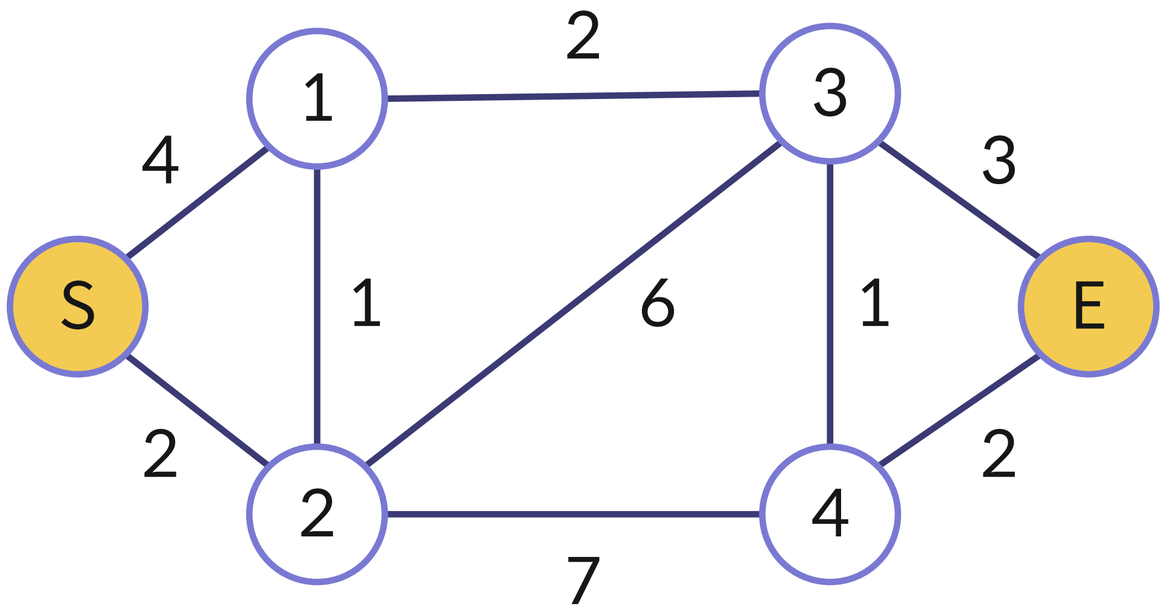
최단거리 알고리즘이란?

그래프의 **한 정점**에서 **다른정점**으로 가는 **최단거리**를 계산하는 알고리즘



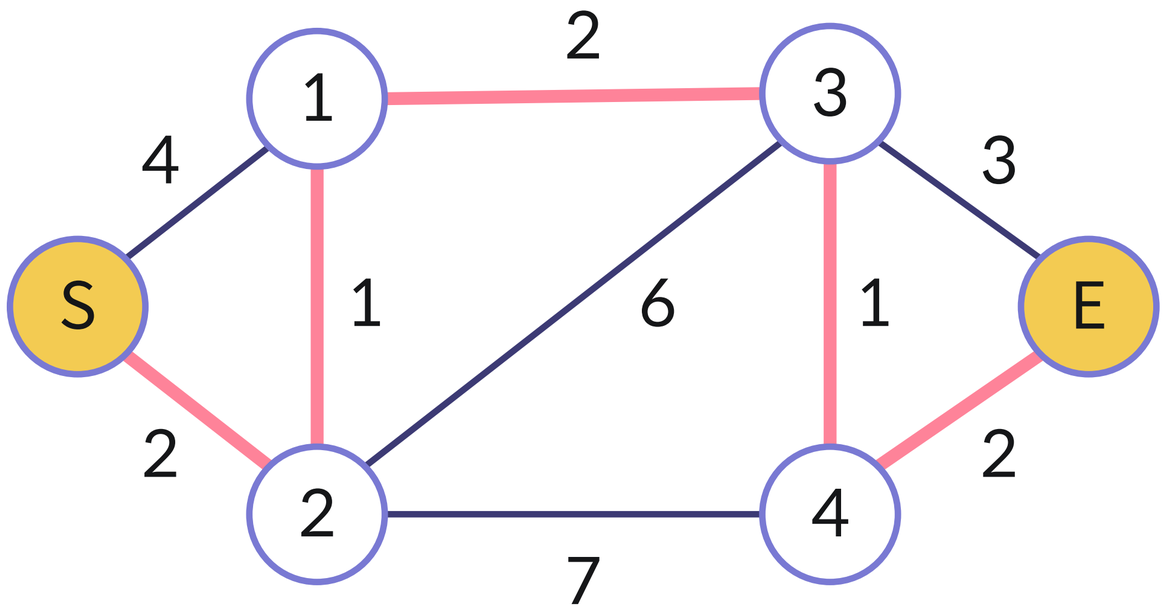
최단거리 알고리즘이란?

그래프의 **한 정점**에서 **다른정점**으로 가는 **최단거리**를 계산하는 알고리즘



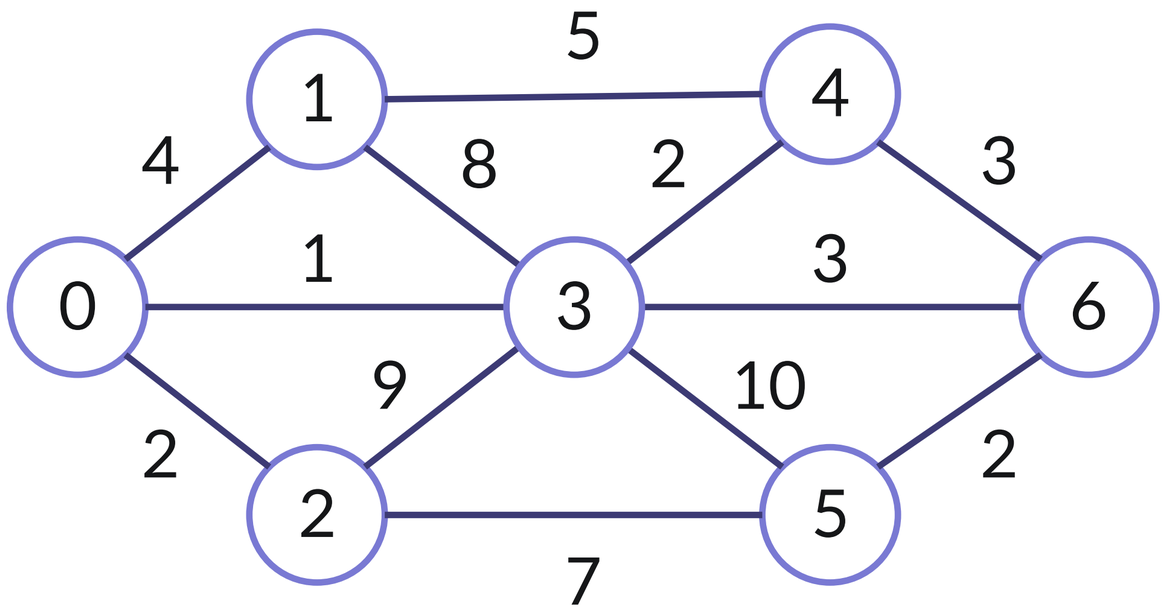
최단거리 알고리즘이란?

그래프의 **한 정점**에서 **다른정점**으로 가는 **최단거리**를 계산하는 알고리즘



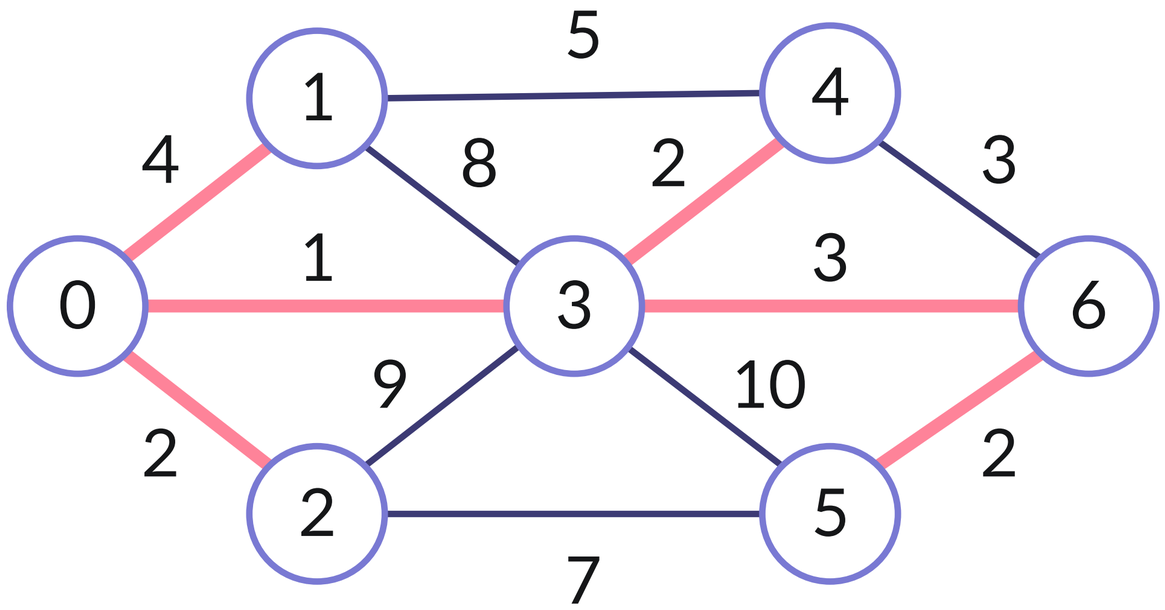
최소신장트리란?

주어진 그래프에서 **최소비용으로 트리**를 만드는 것입니다.
최소신장트리는 그래프내 **모든 정점을 연결하고 있는 트리**인데,
트리의 **간선의 가중치가 최소**가됩니다.



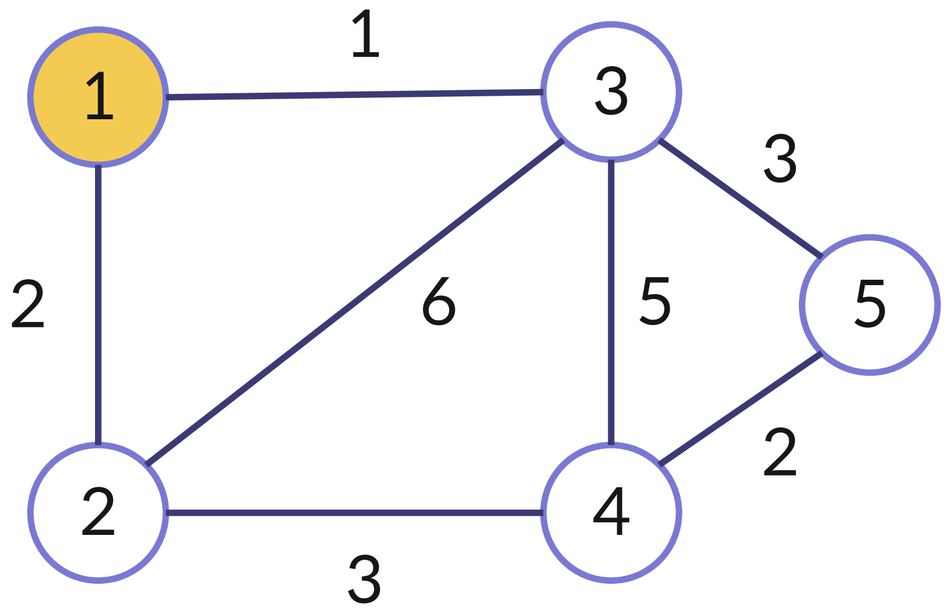
최소신장트리란?

주어진 그래프에서 **최소비용으로 트리**를 만드는 것입니다.
최소신장트리는 그래프내 **모든 정점을 연결하고 있는 트리**인데,
트리의 **간선의 가중치가 최소**가됩니다.



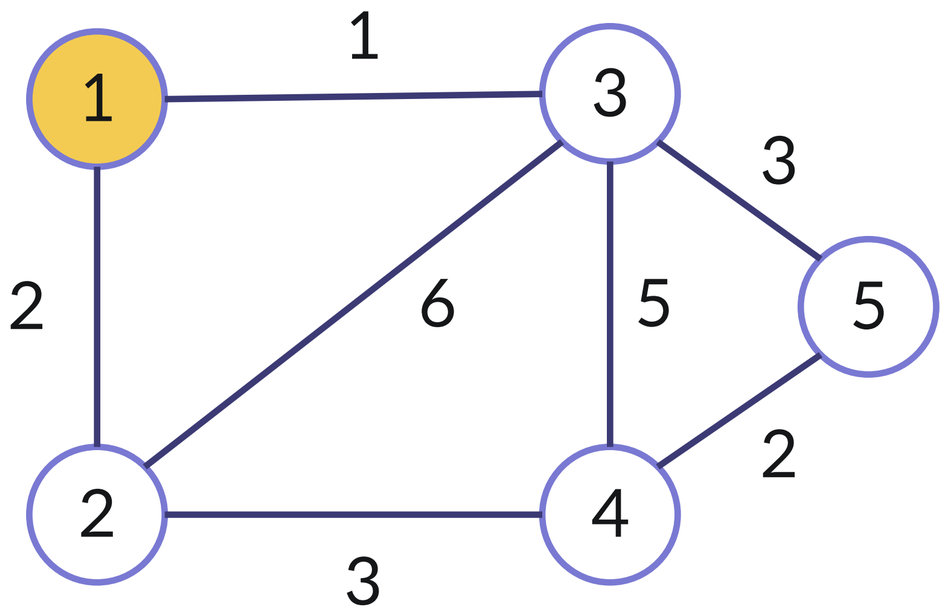
최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

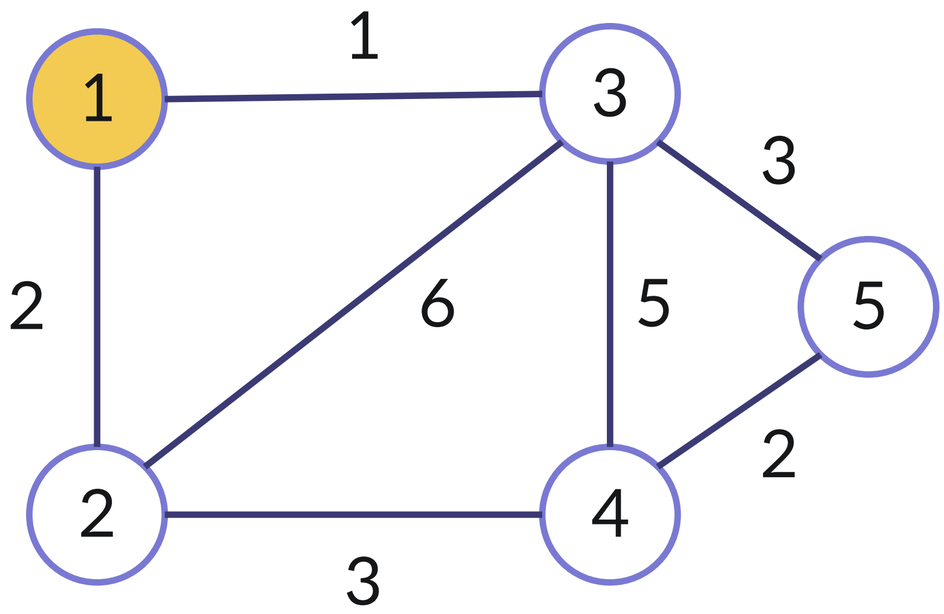
하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	∞	∞	∞	∞

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

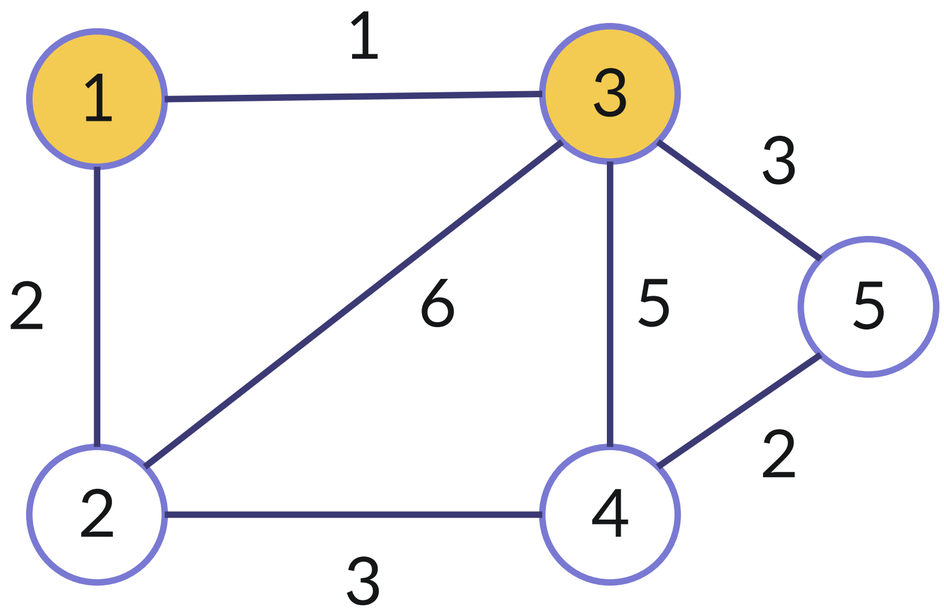
하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	2	1	∞	∞

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

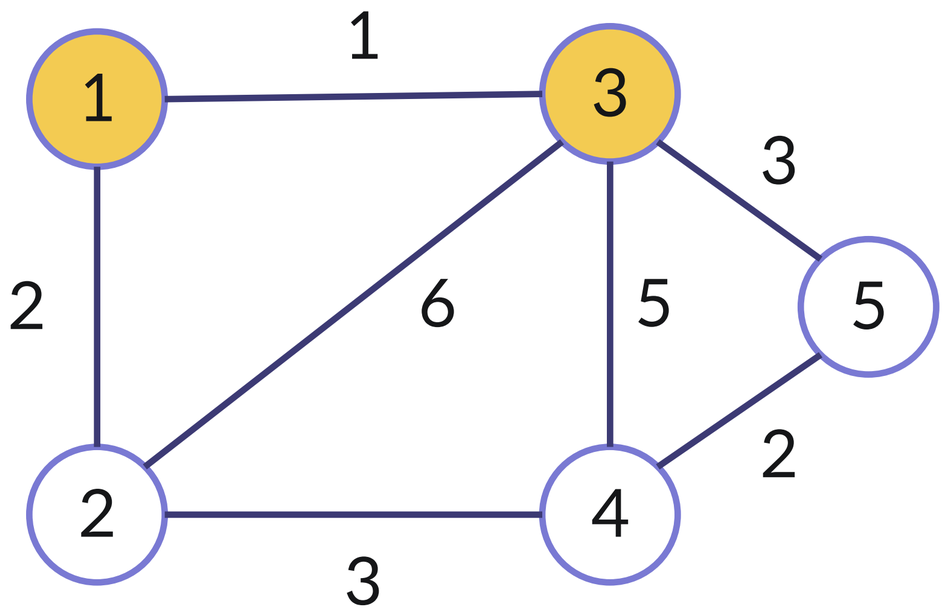
하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	2	1	∞	∞

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

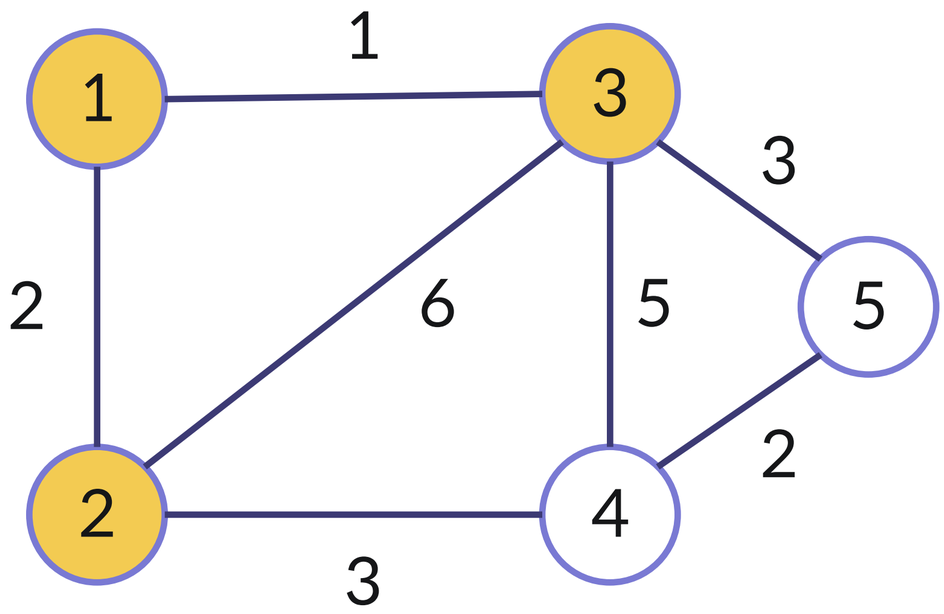
하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	2	1	6	4

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

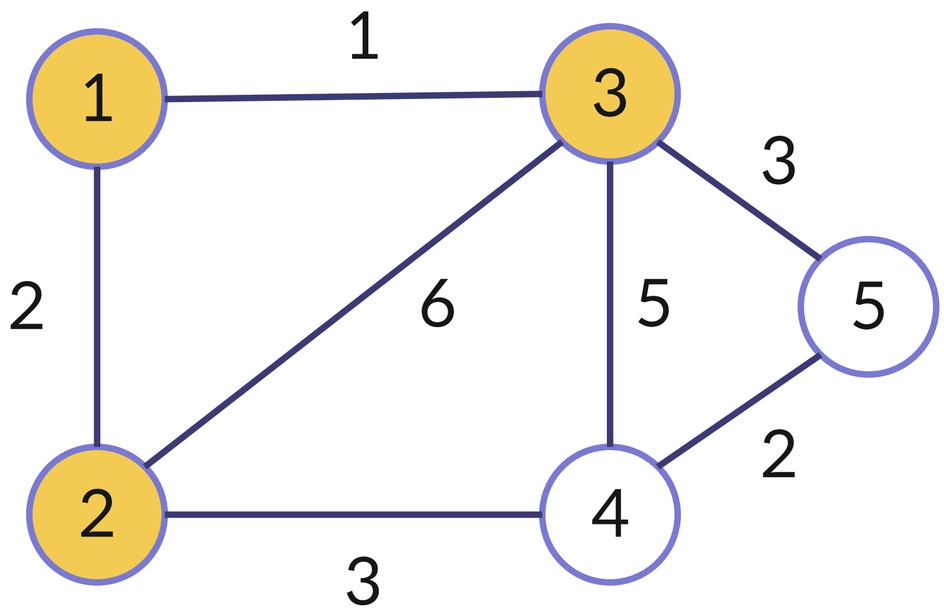
하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	2	1	6	4

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

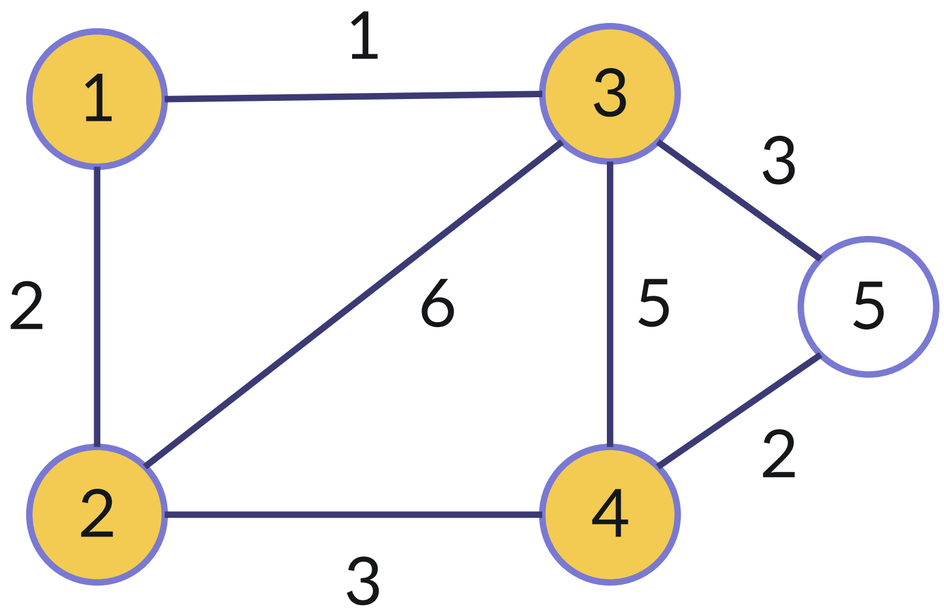
하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	2	1	5	4

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

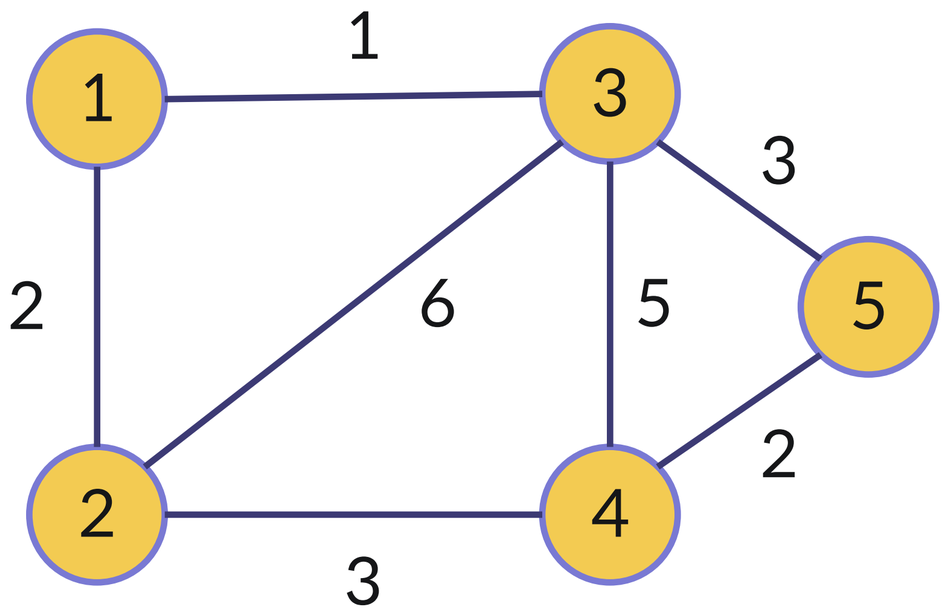
하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	2	1	5	4

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘



1	2	3	4	5
0	2	1	5	4

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

Example

```
def dijkstra(start, V, graph):
    visited = [False] * V

    dist = [float('inf')] * V
    dist[start] = 0

    while True:
        minimum = float('inf')
        node = -1
        for j in range(V):
            if not visited[j] and minimum > dist[j]:
                minimum = dist[j]
                node = j
        if minimum == float('inf'):
            break

        visited[node] = True

        for j in range(V):
            if visited[j]: continue
            via = dist[node] + graph[node][j]
            if dist[j] > via:
                dist[j] = via

    return dist
```

/* elice */

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

임의의 **정점 v에 다다랐을 때 드는 비용 dist[v]**값 중 **가장 작은 것**을 골라서 확정지어주기

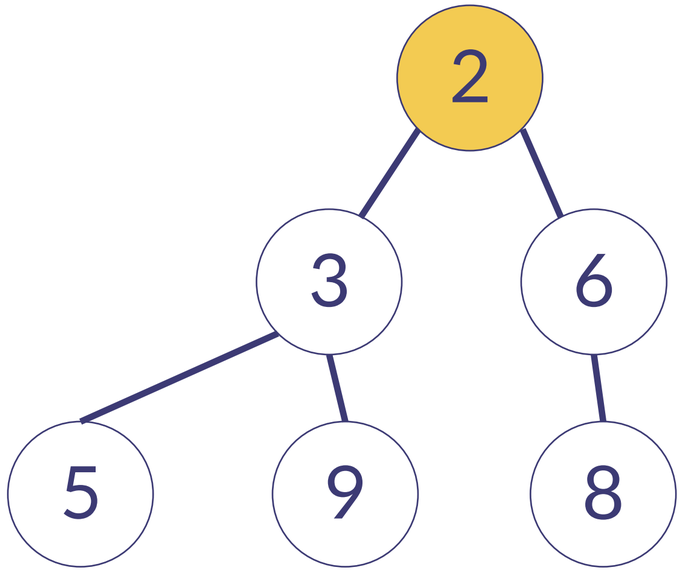
구현 핵심

1. 확정되지 않은 정점들 중 dist에 있는 가장 작은 값을 찾기

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

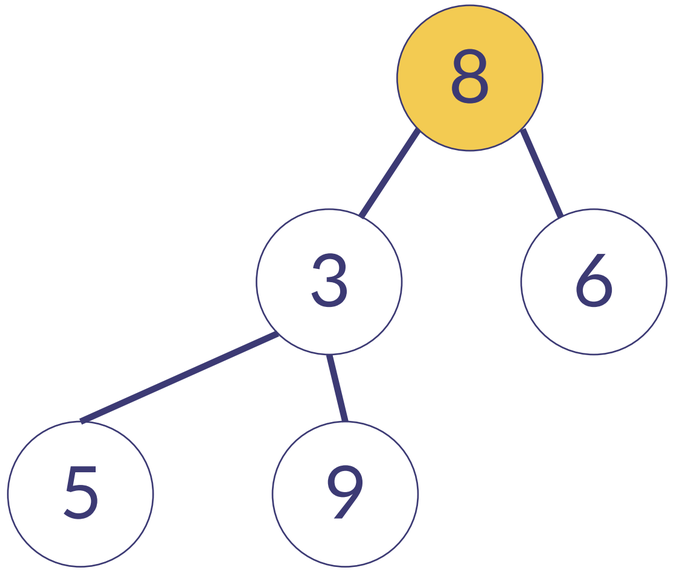
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

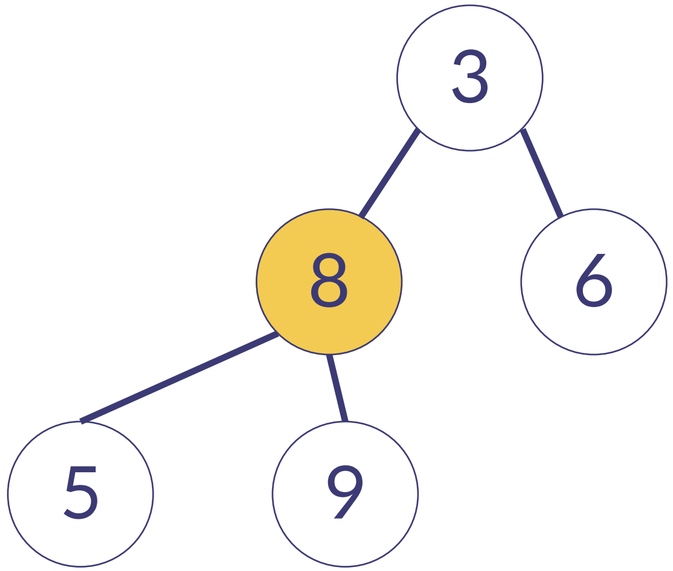
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

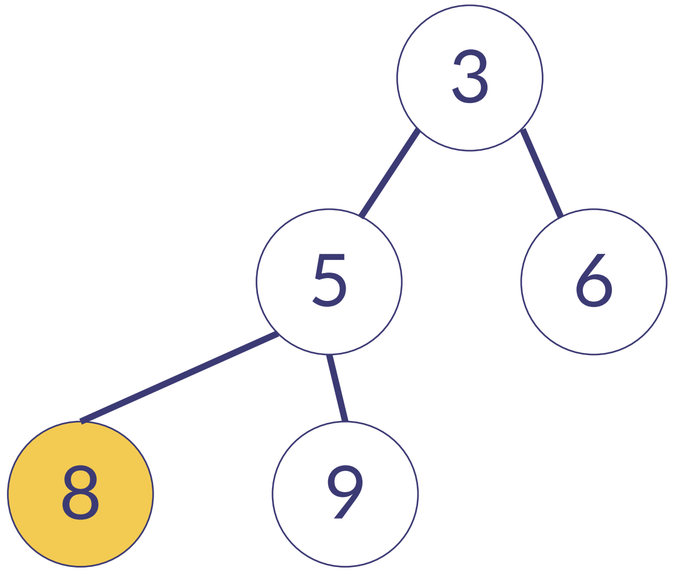
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

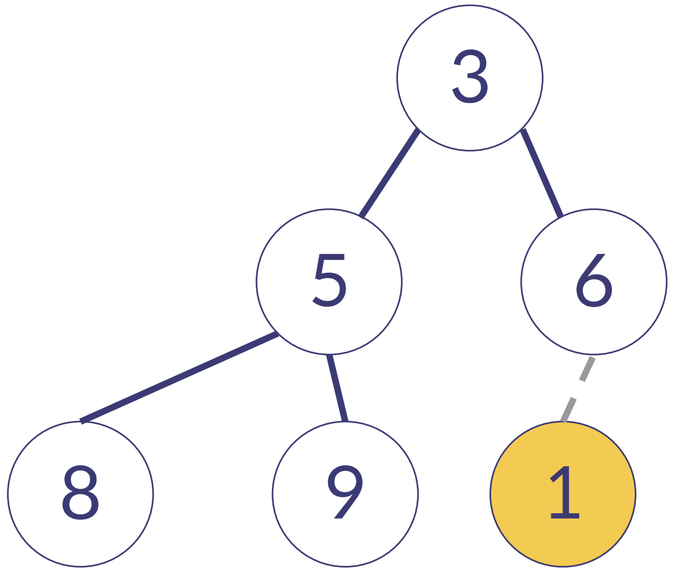
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

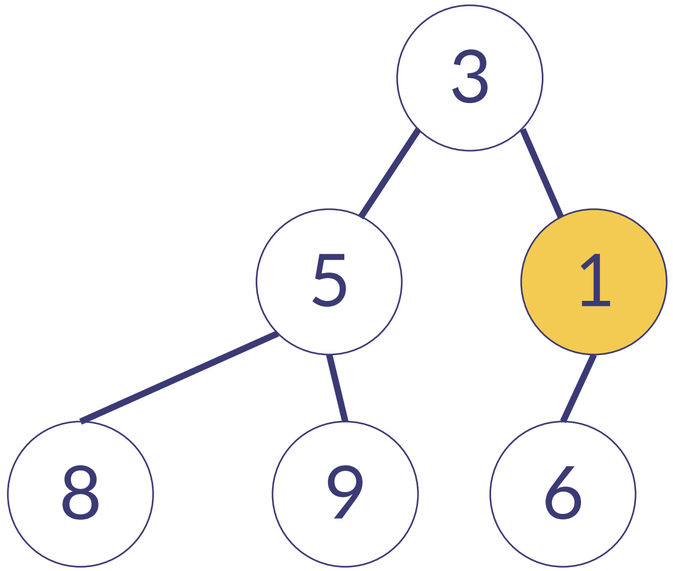
예) 3 9 8 5 6 에 1을 추가한다면?



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

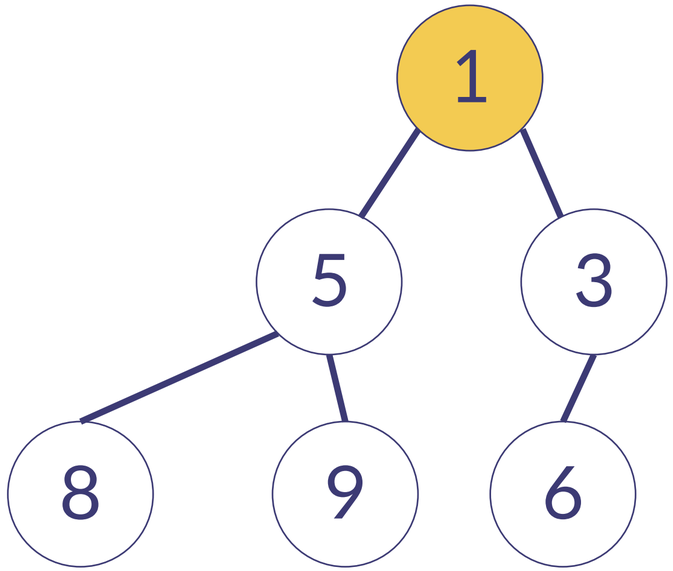
예) 3 9 8 5 6 에 1을 추가한다면?



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

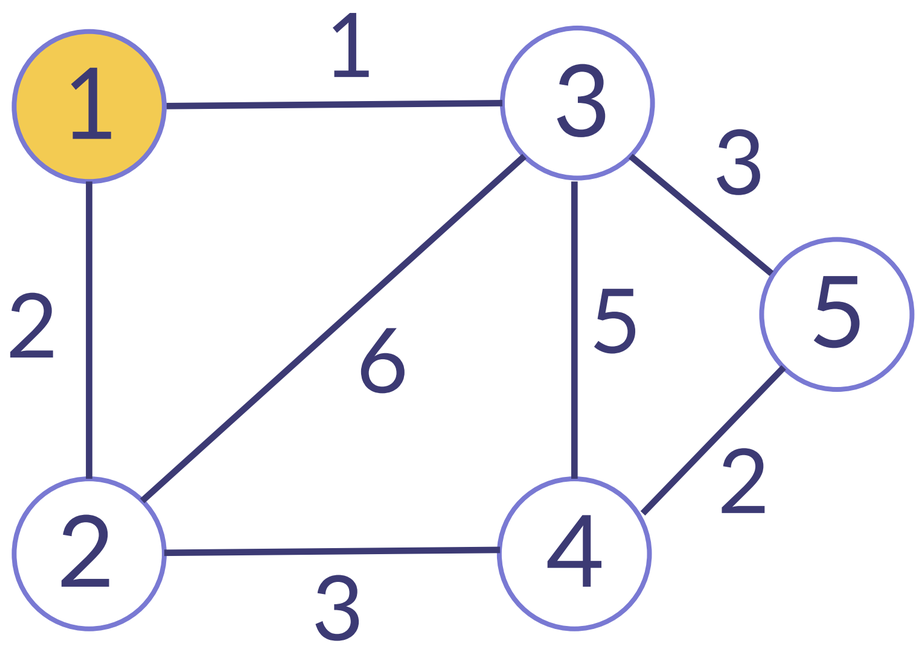
가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

예) 3 9 8 5 6 에 1을 추가한다면?



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

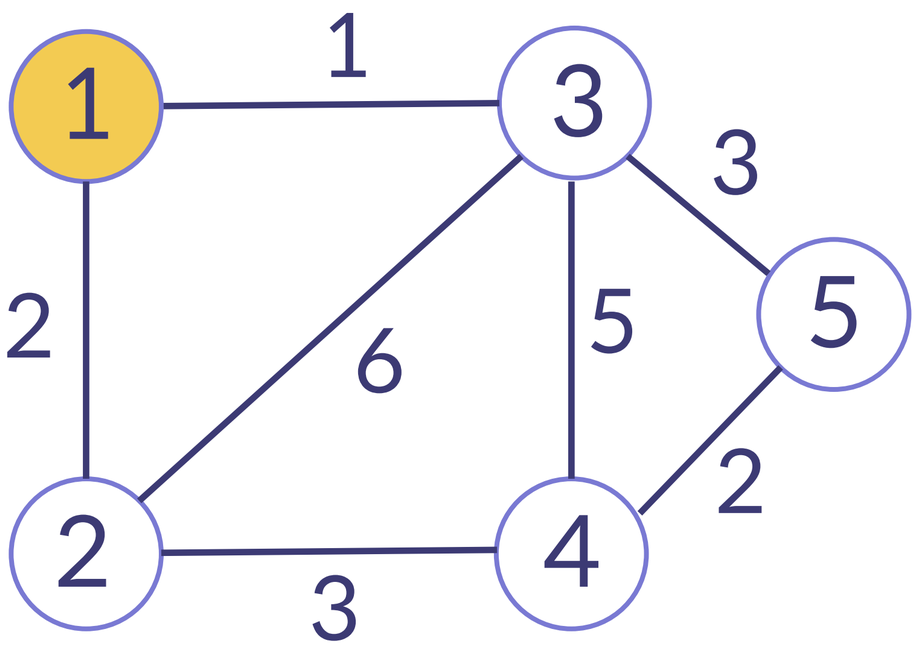
하나의 정점에서 다른 모든 정점까지의
최단경로를 구하는 알고리즘



1	2	3	4	5
0	∞	∞	∞	∞

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

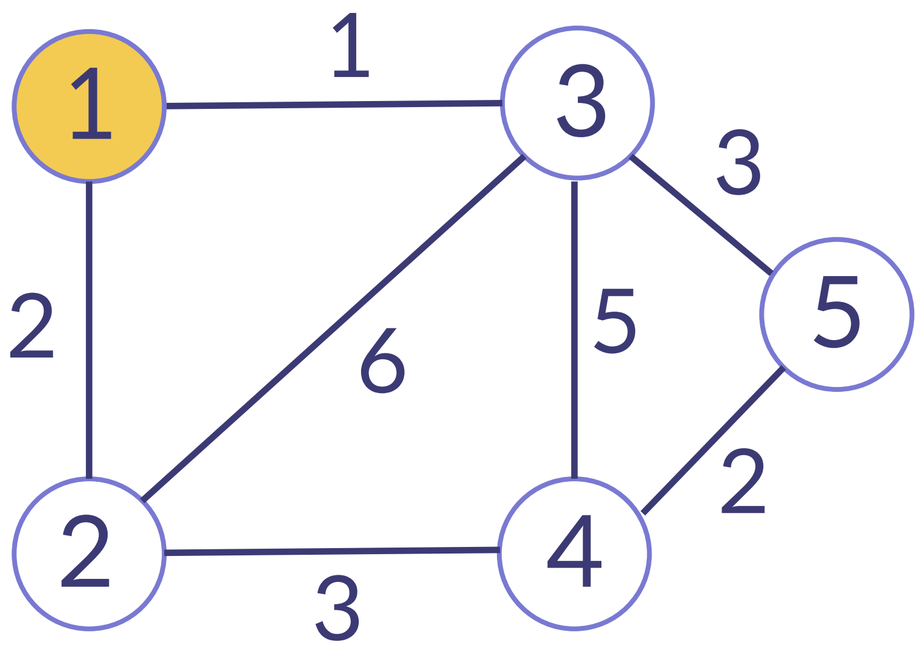
하나의 정점에서 다른 모든 정점까지의
최단경로를 구하는 알고리즘



1	2	3	4	5
0	∞	∞	∞	∞

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

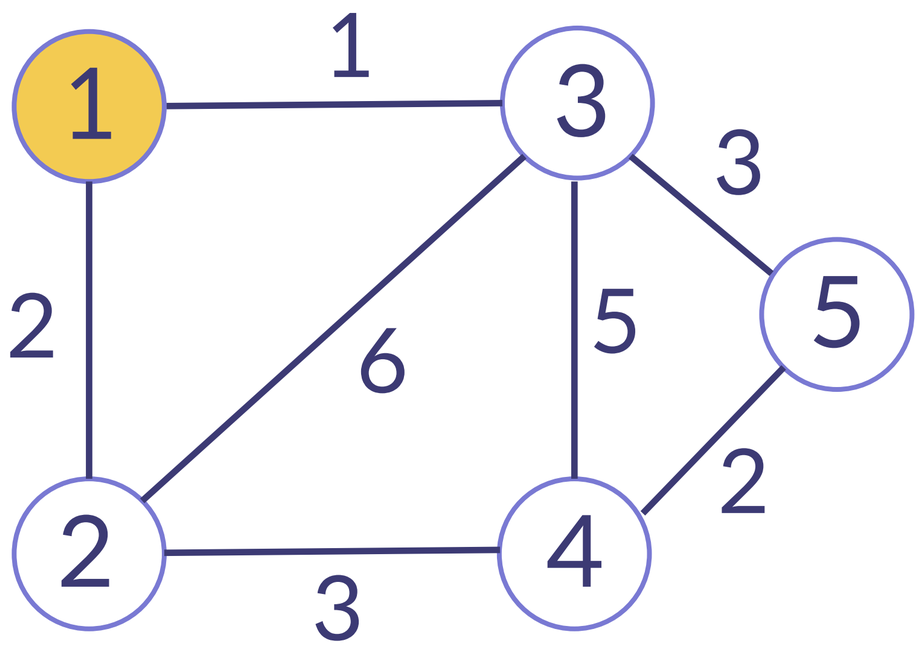
하나의 정점에서 다른 모든 정점까지의
최단경로를 구하는 알고리즘



1	2	3	4	5
0	∞	∞	∞	∞

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

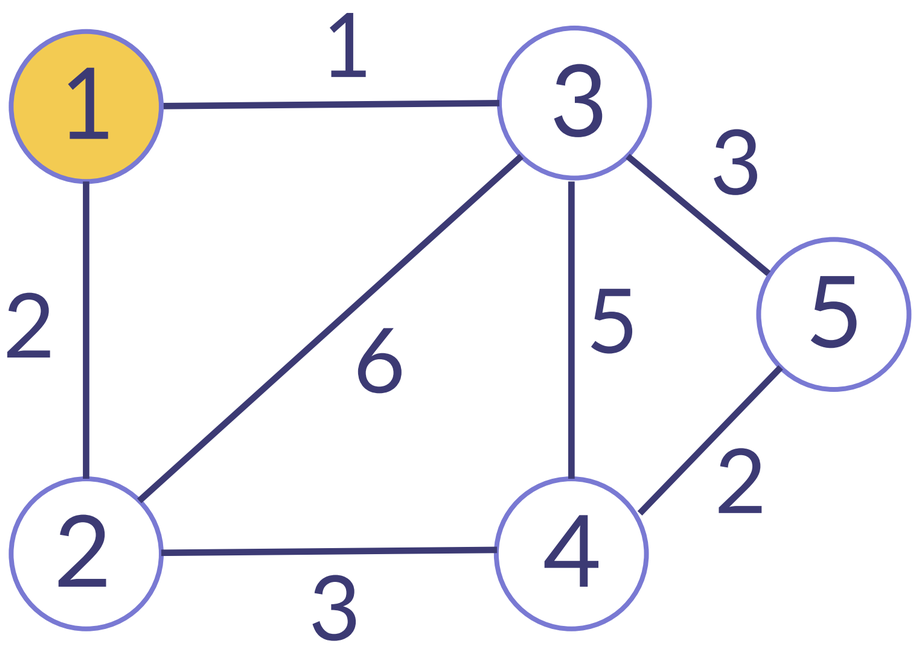
우선순위 큐에 dist[v]와 v를 함께 넣어서 dist에 남아있는 것 중 가장 작은 값을 찾아줍니다.



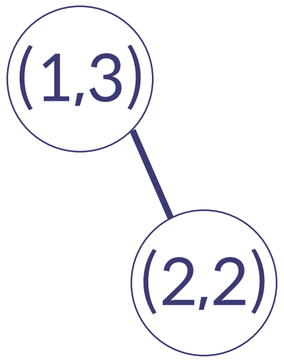
1	2	3	4	5
0	2	1	∞	∞

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

우선순위 큐에 dist[v]와 v를 함께 넣어서 dist에 남아있는 것 중 가장 작은 값을 찾아줍니다.

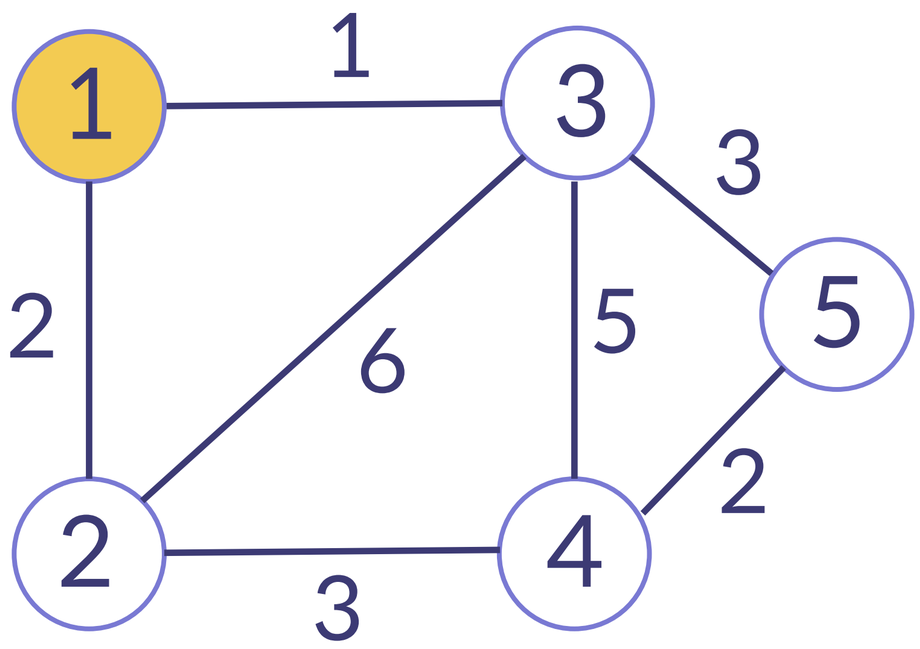


1	2	3	4	5
0	2	1	∞	∞



최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

우선순위 큐에 dist[v]와 v를 함께 넣어서 dist에 남아있는 것 중 가장 작은 값을 찾아줍니다.



1	2	3	4	5
0	2	1	∞	∞

(2, 2)

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

Example

```
def dijkstra(start, V, graph):
    visited = [False] * V

    dist = [float('inf')] * V
    dist[start] = 0

    while True:
        minimum = float('inf')
        node = -1
        for j in range(V):
            if not visited[j] and minimum > dist[j]:
                minimum = dist[j]
                node = j
        if minimum == float('inf'):
            break

        visited[node] = True

        for j in range(V):
            if visited[j]: continue
            via = dist[node] + graph[node][j]
            if dist[j] > via:
                dist[j] = via

    return dist
```

/* elice */

최단거리 알고리즘 - 다익스트라 (Dijkstra’s Algorithm)

Example

```
def dijkstra(start, V, graph):
    dist = [float('inf')] * V
    dist[start] = 0
    pq = [(0, start)]

    while pq:
        cur_dist, cur_vertex = heapq.heappop(pq)
        if cur_dist > dist[cur_vertex] : continue
```

```
    for neighbor, weight in
                        graph[cur_vertex].items():
        distance = cur_dist + weight
        if distance < dist[neighbor] :
            dist[neighbor] = distance
            heapq.heappush(pq,
                    (distance, neighbor))
return dist
```

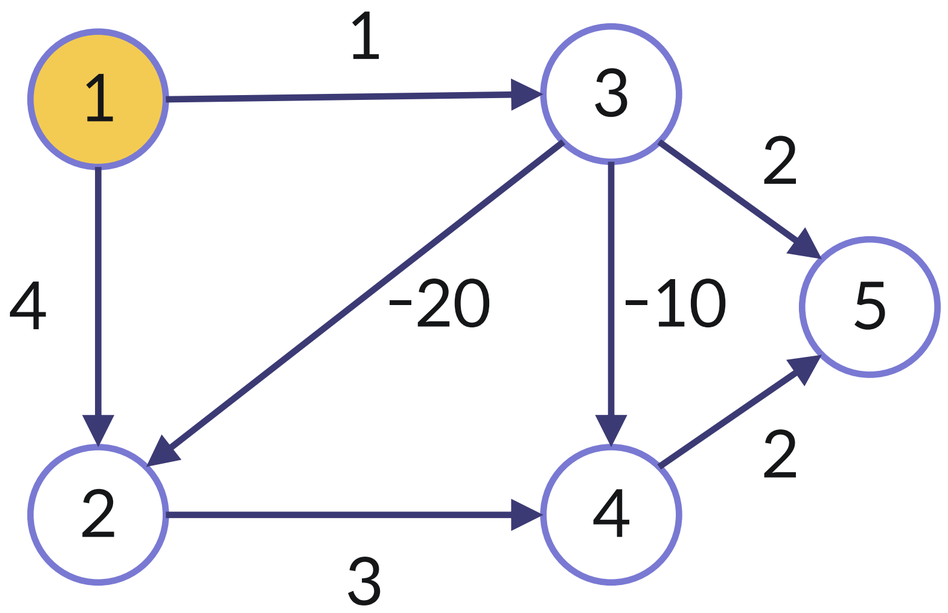
최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

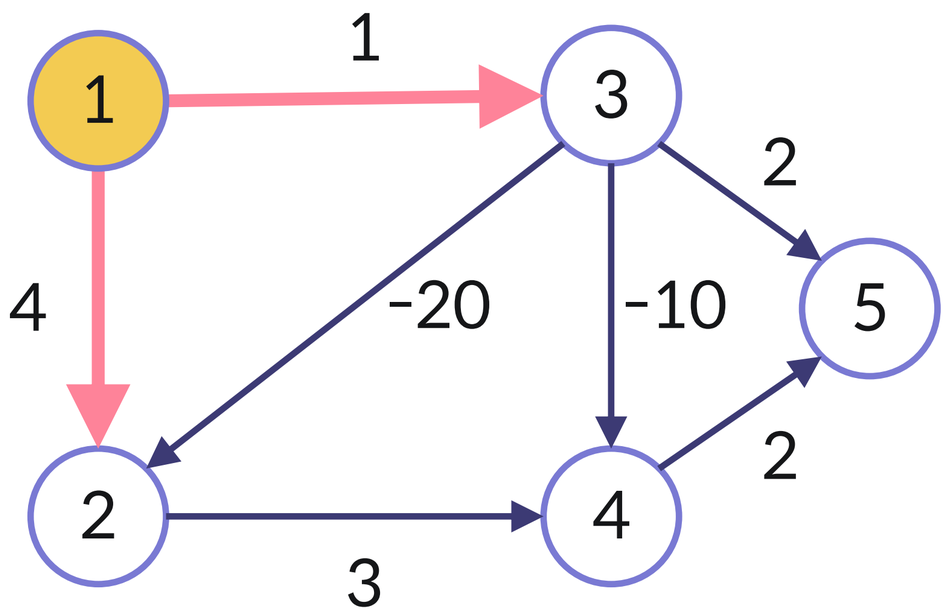
- 다익스트라와 다르게 음수 가중치도 사용 가능하다.



최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

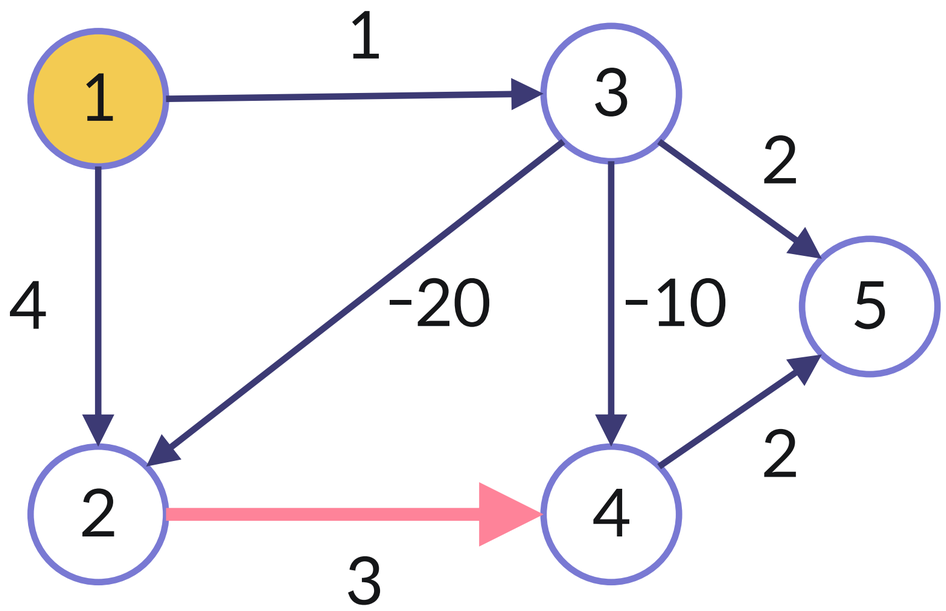


1	2	3	4	5
0	4	1	∞	∞

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

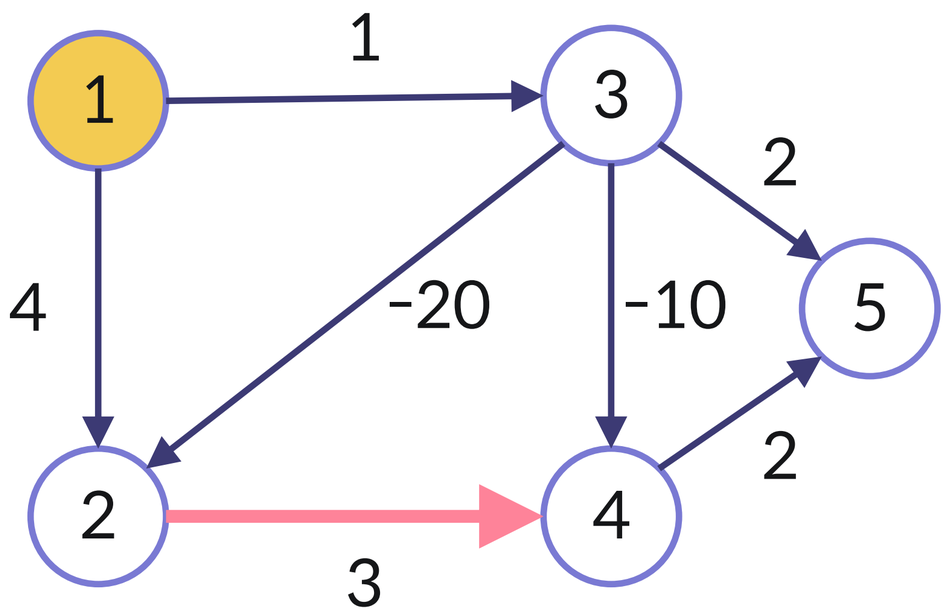


1	2	3	4	5
0	4	1	∞	∞

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

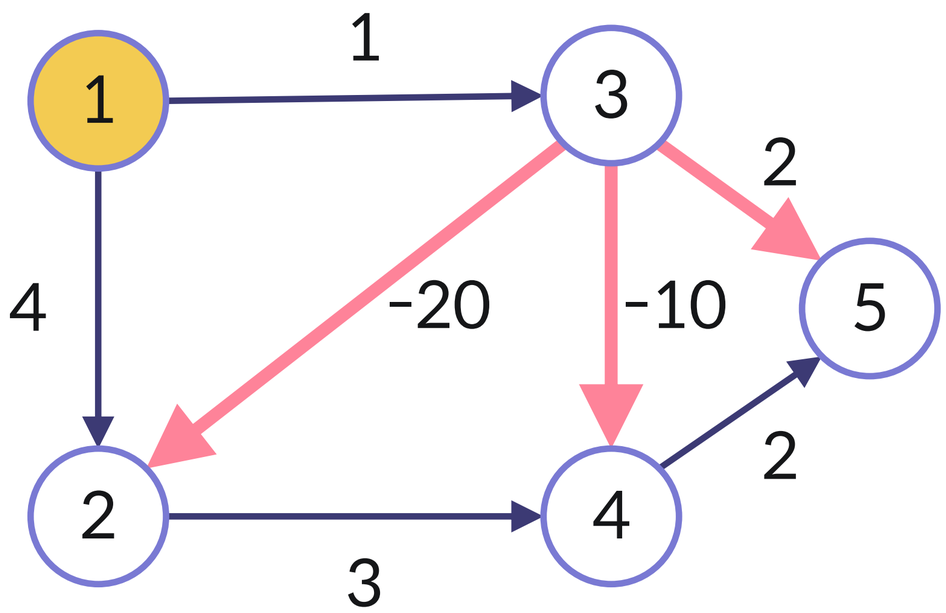


1	2	3	4	5
0	4	1	7	∞

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

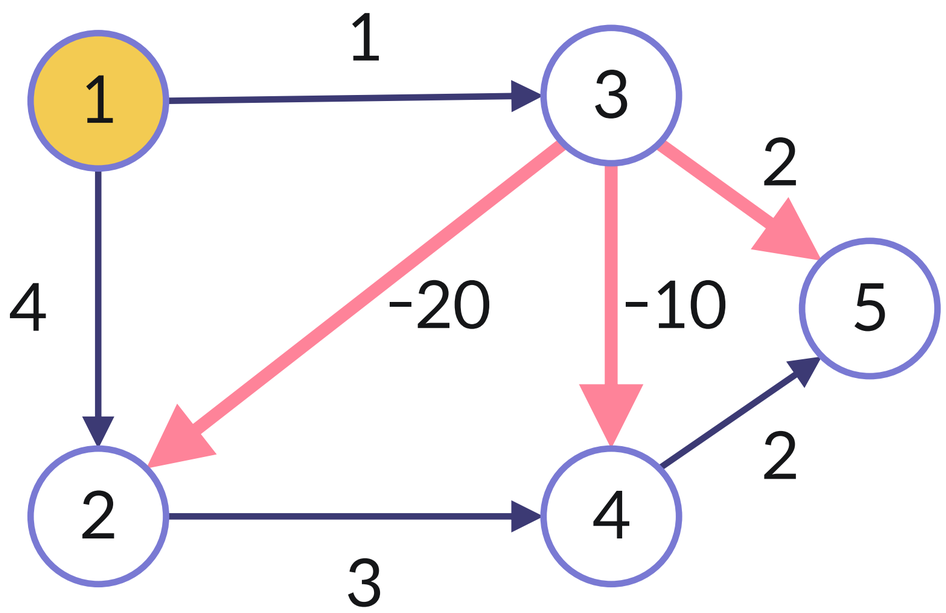


1	2	3	4	5
0	4	1	7	∞

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

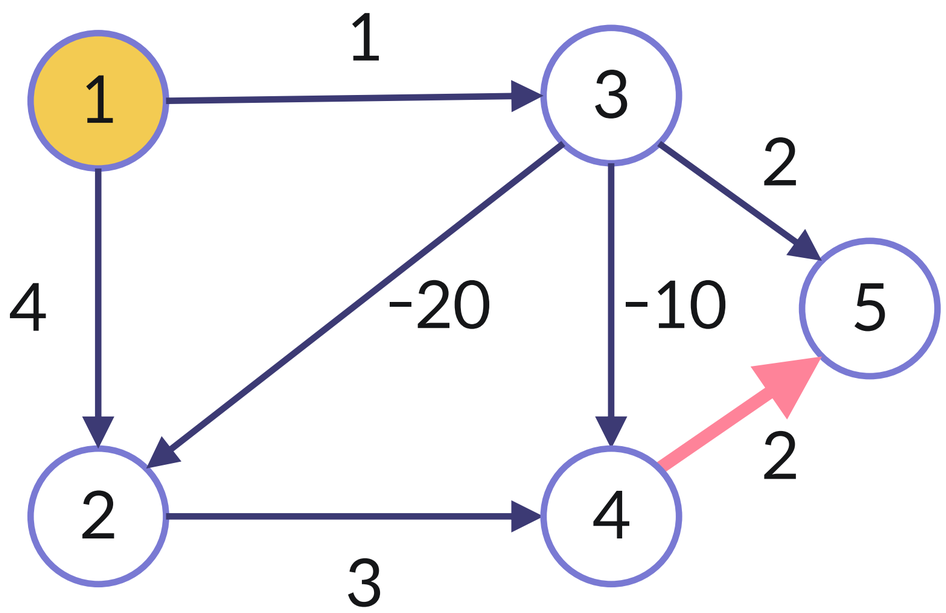


1	2	3	4	5
0	-19	1	-9	3

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

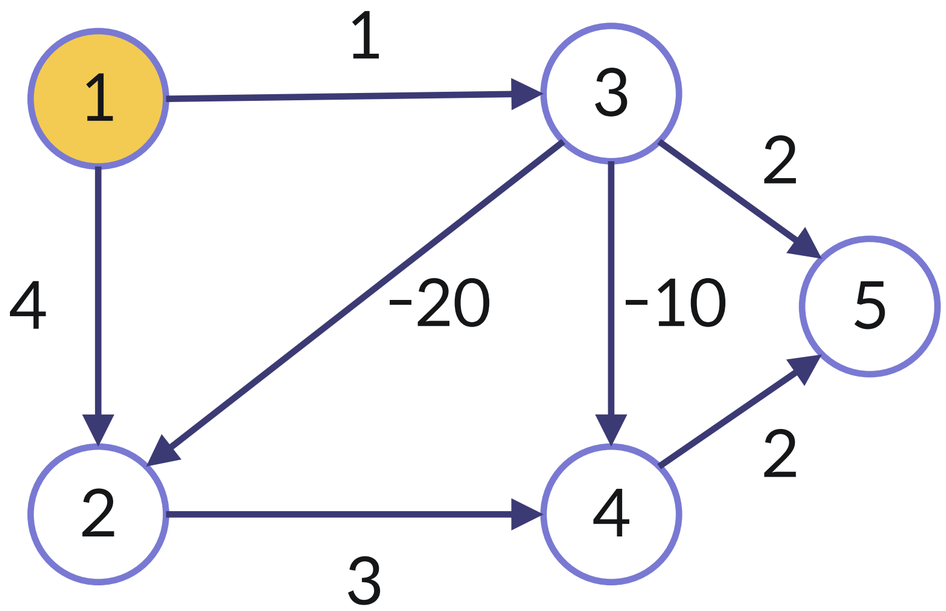


1	2	3	4	5
0	-19	1	-9	-7

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

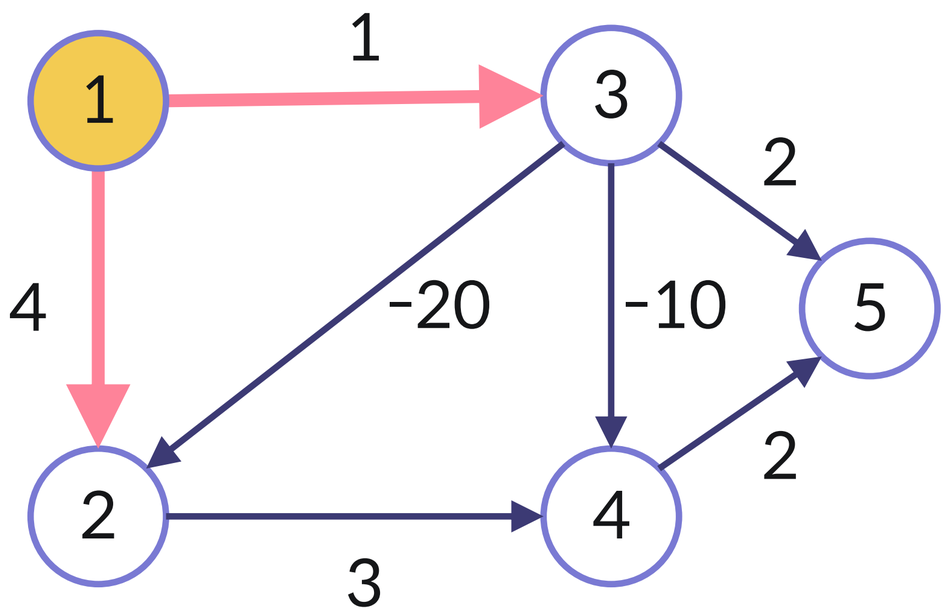


1	2	3	4	5
0	-19	1	-9	-7

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

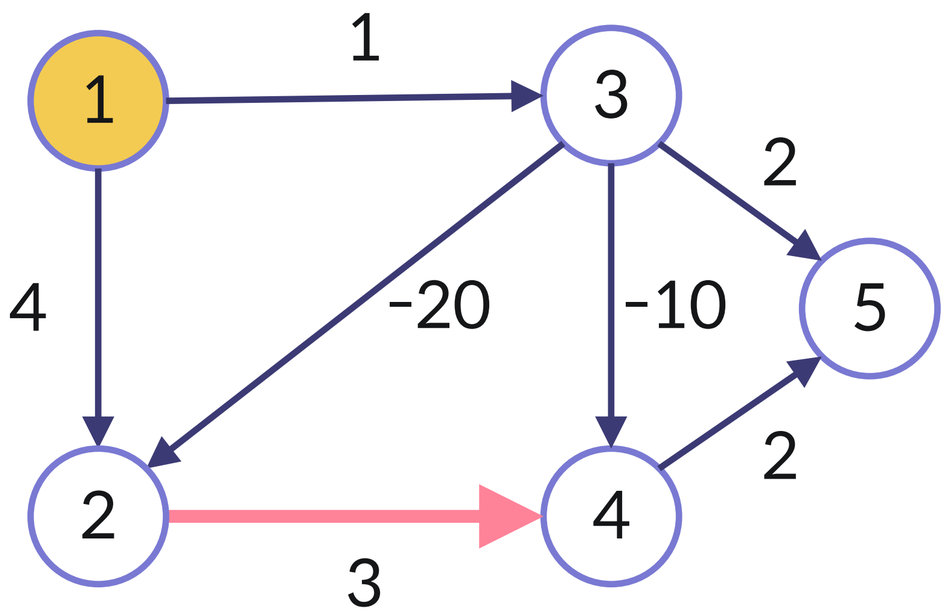


1	2	3	4	5
0	-19	1	-9	-7

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

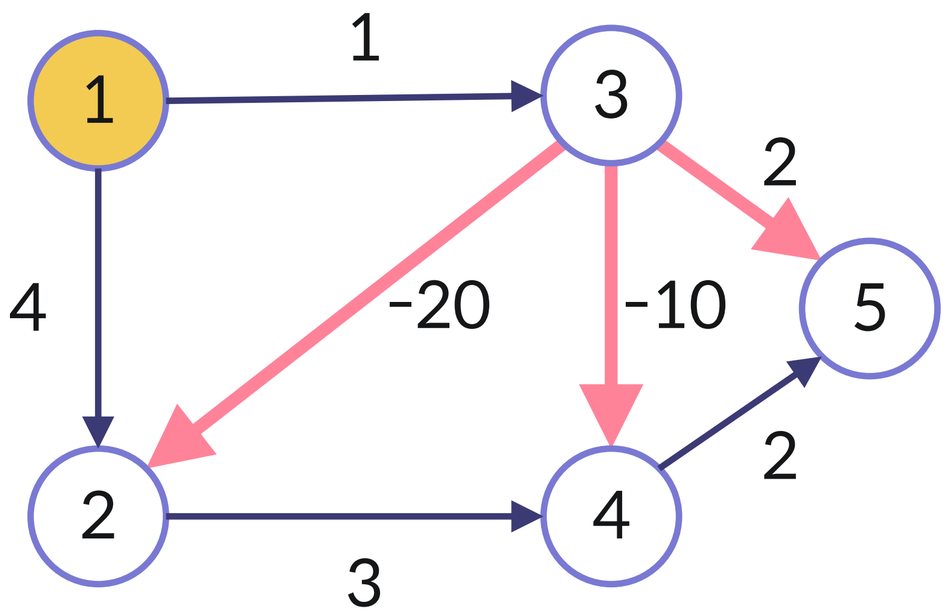


1	2	3	4	5
0	-19	1	**-16**	-7

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

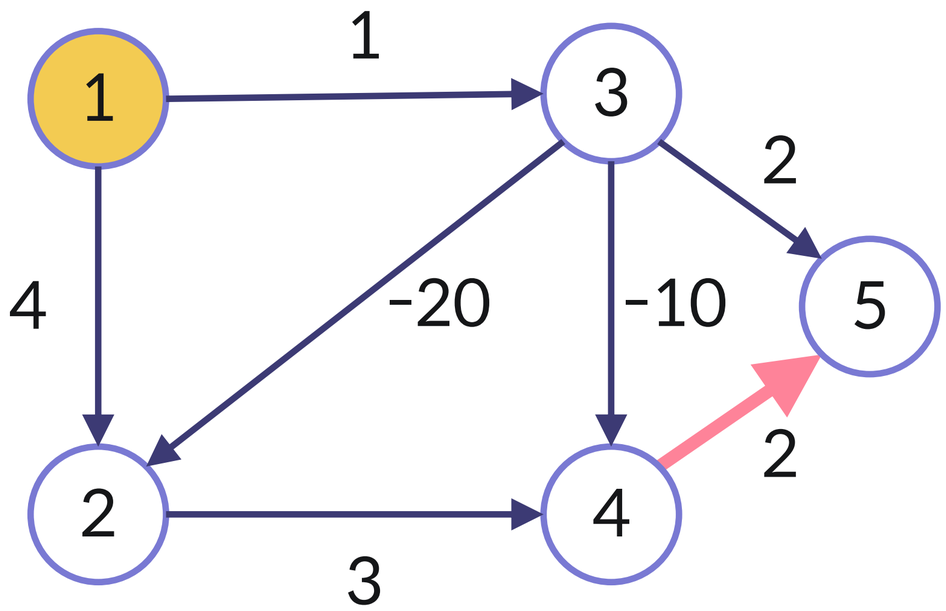


1	2	3	4	5
0	-19	1	-16	-7

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

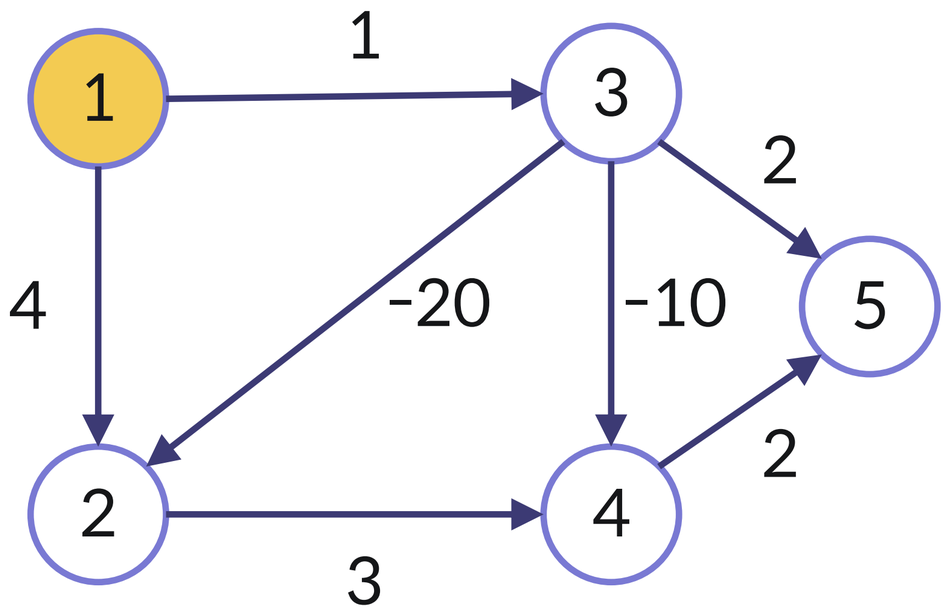


1	2	3	4	5
0	-19	1	-16	-14

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.



1	2	3	4	5
0	-19	1	-16	-14

정점갯수만큼 반복!

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

하나의 정점에서 다른 모든 정점까지의 **최단경로**를 구하는 알고리즘

- 다익스트라와 다르게 음수 가중치도 사용 가능하다.

최단거리 알고리즘 – 벨만포드 (Bellman–Ford algorithm)

Example

```
def bellman(start, V, E, edges):
    dist = [float('inf')] * V
    dist[start] = 0

    for i in range(V):
        for j in range(E):
            st = edges[j][0]
            ed = edges[j][1]
            weight = edges[j][2]

            if dist[st]+weight < dist[ed]:
                dist[ed] = dist[st]+weight
```

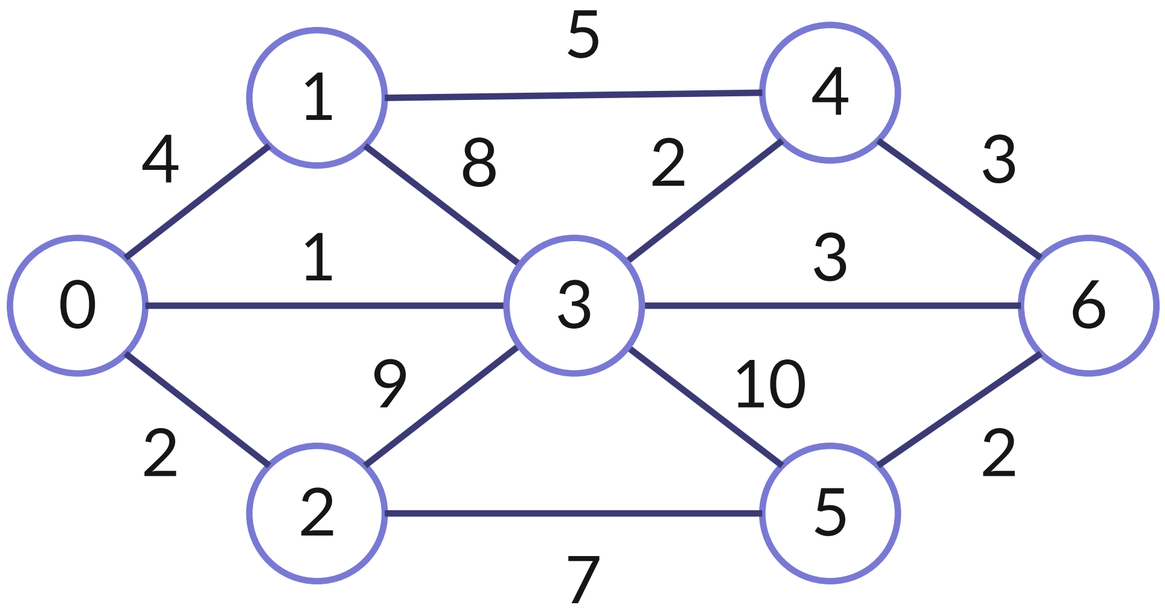
```
for i in range(E):
    st = edges[i][0]
    ed = edges[i][1]
    weight = edges[i][2]

    if dist[st]+weight < dist[ed]:
        return -1

return dist
```

/* elice */

최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

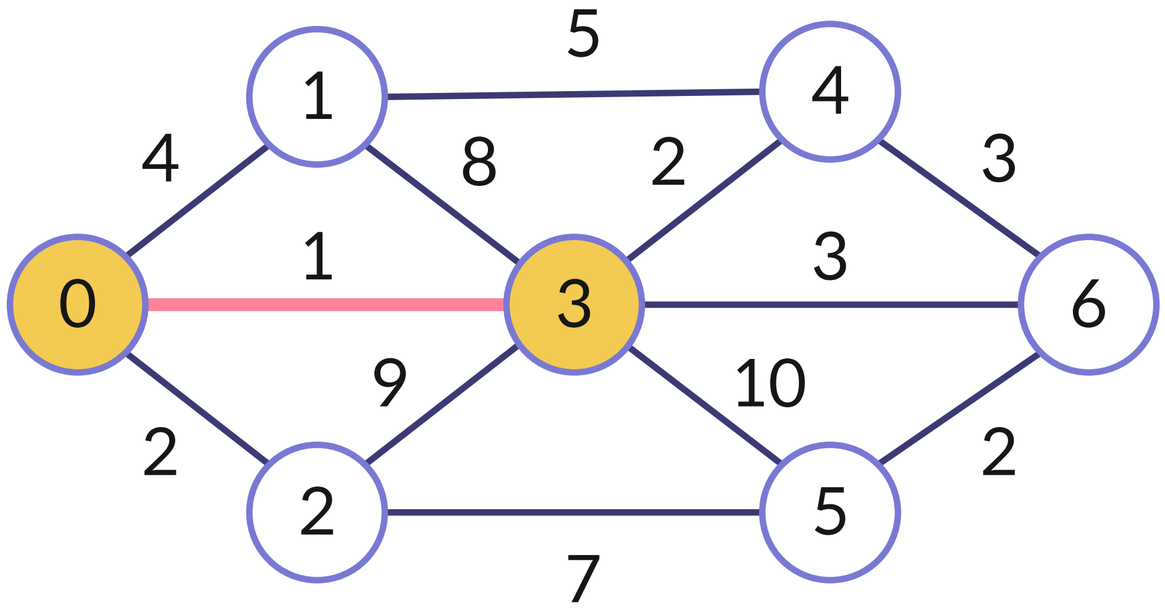
- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

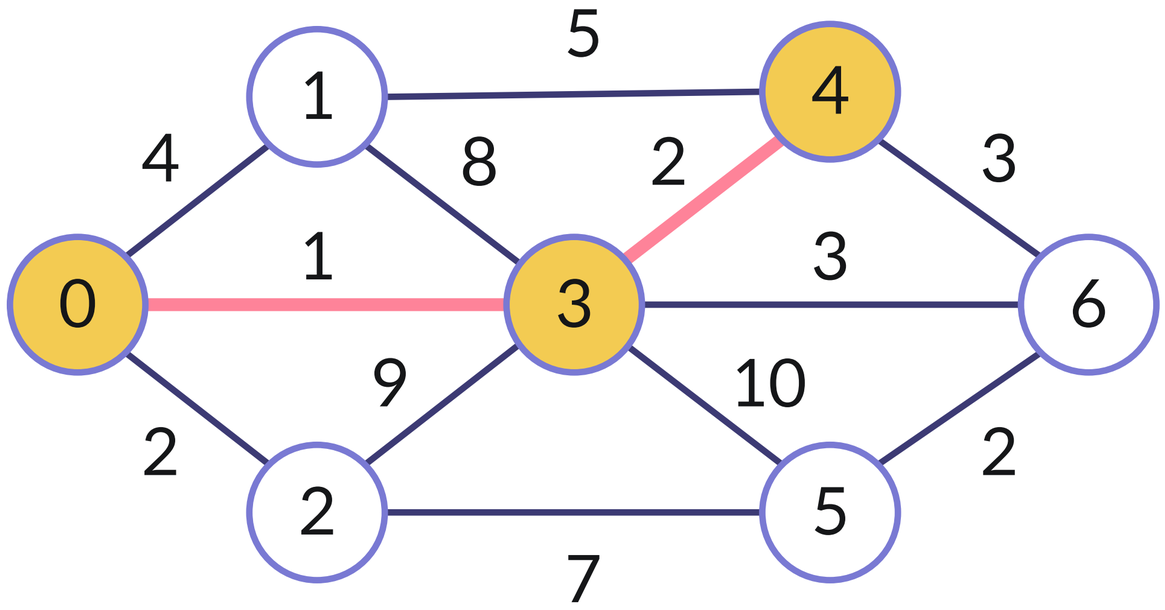
- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

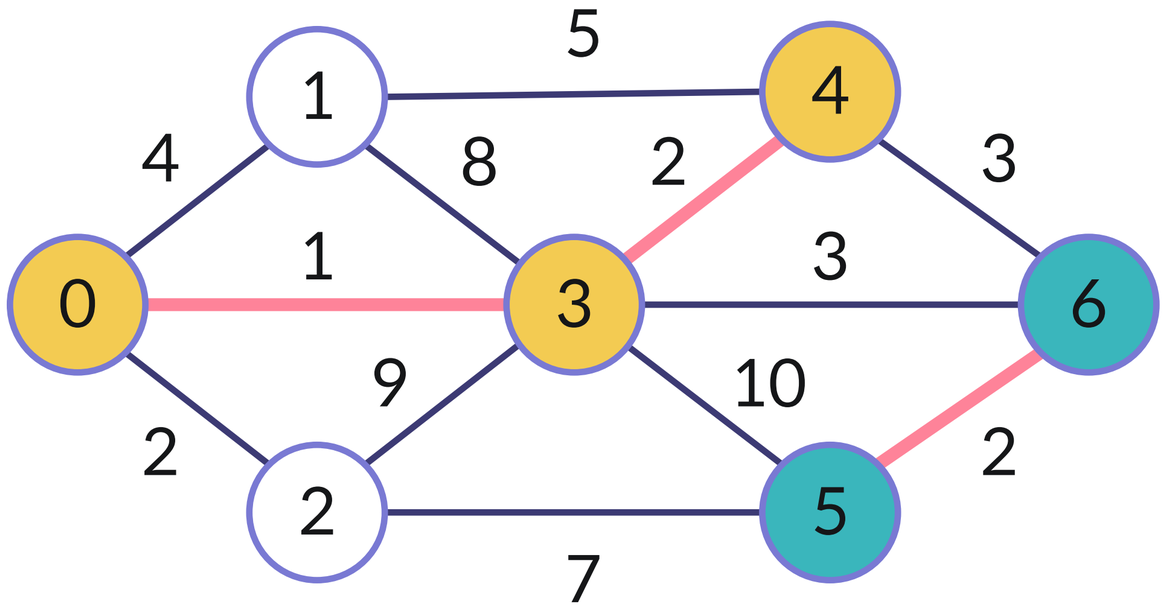
- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

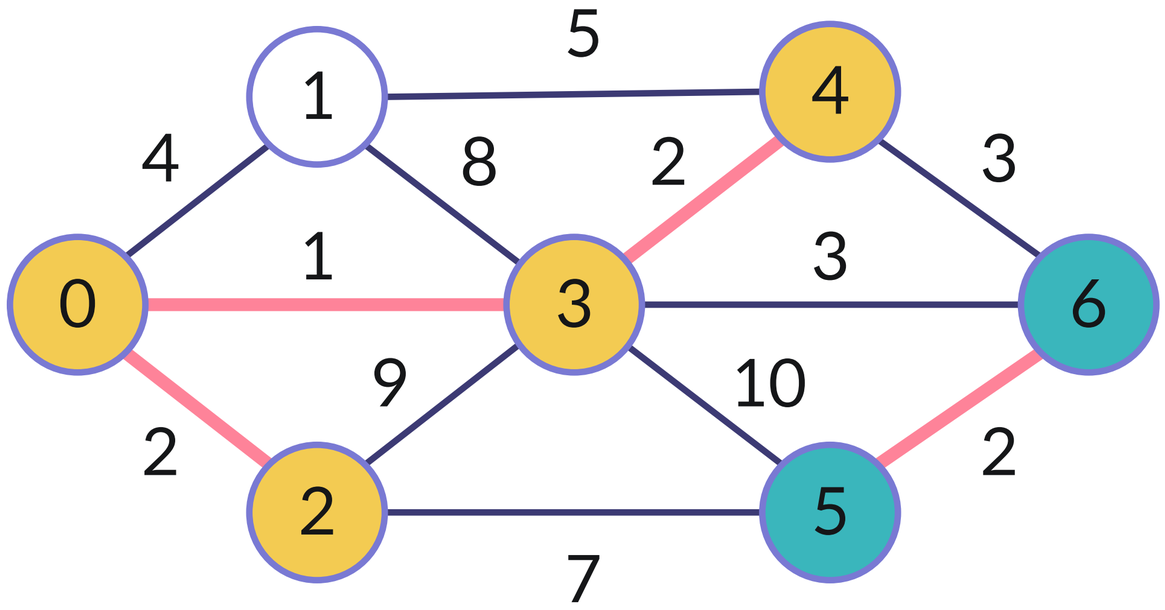
- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

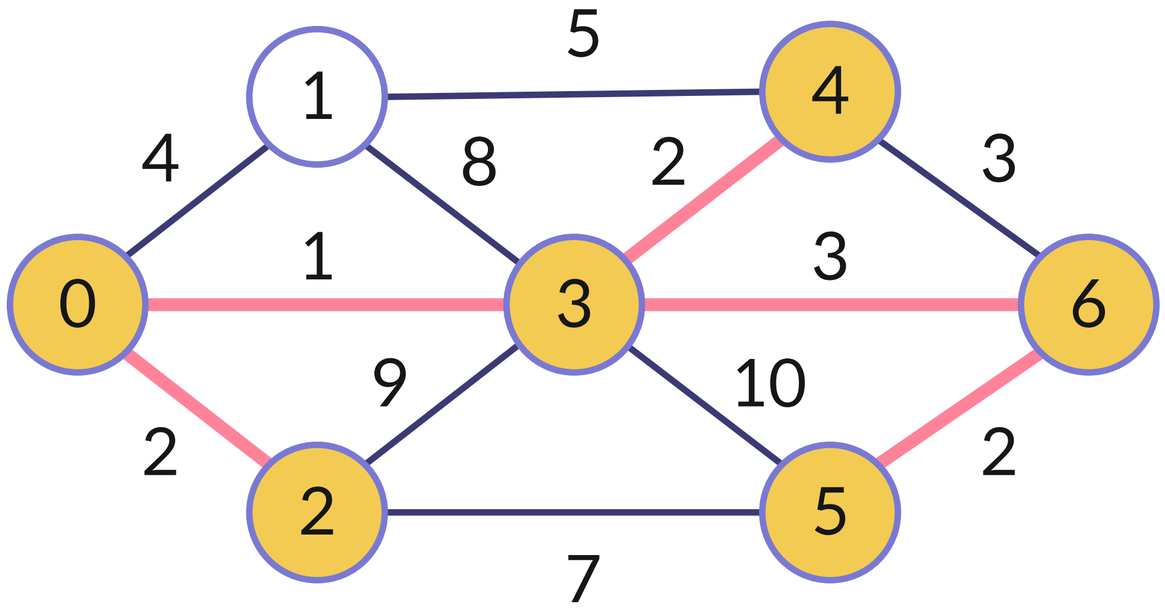
- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

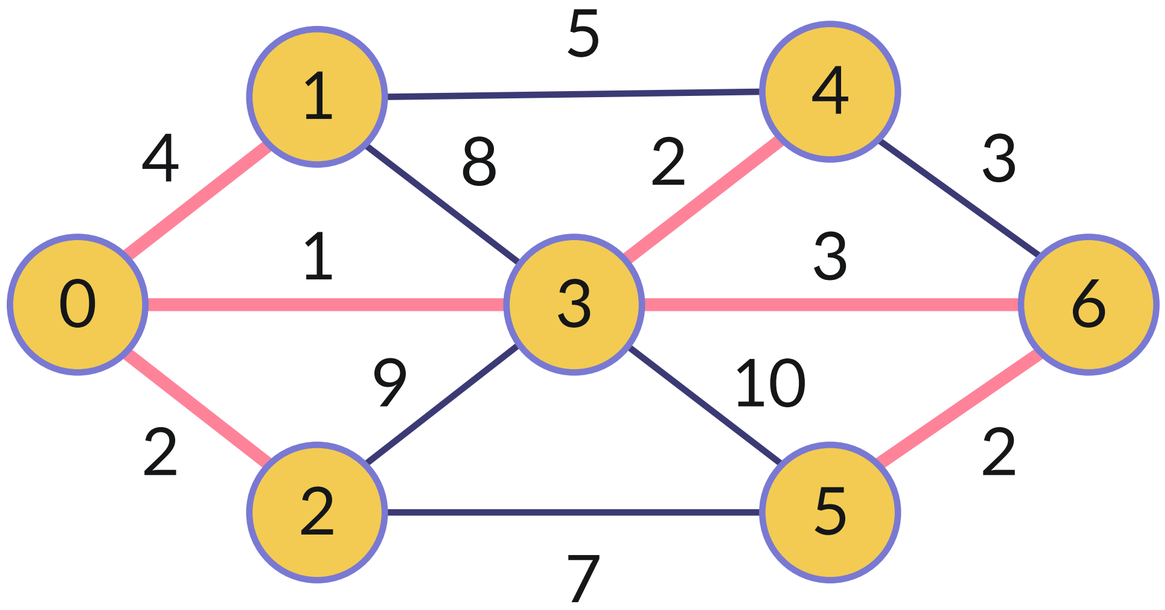
- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.

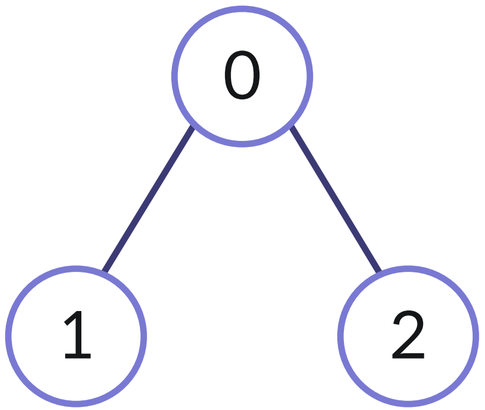
구현 핵심

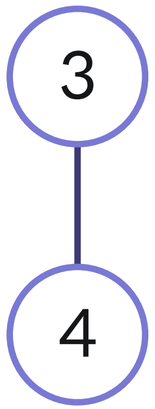
1. 간선을 **정렬**하여 짧은 간선부터 고려한다
2. 간선을 포함시킬 땐,
   사이클이 있는지 (이미 같은 트리에 포함된 정점이 있는지) **확인**한다.

최소신장트리 (minimum spanning tree) - 크루스칼

사이클을 확인하는 방법 – Union find

임의의 노드를 포함하는 트리 A와 트리 B가 동일한 트리인지 찾고 (**Find**)
두 트리를 한트리로 통합하는 과정 (**Union**)

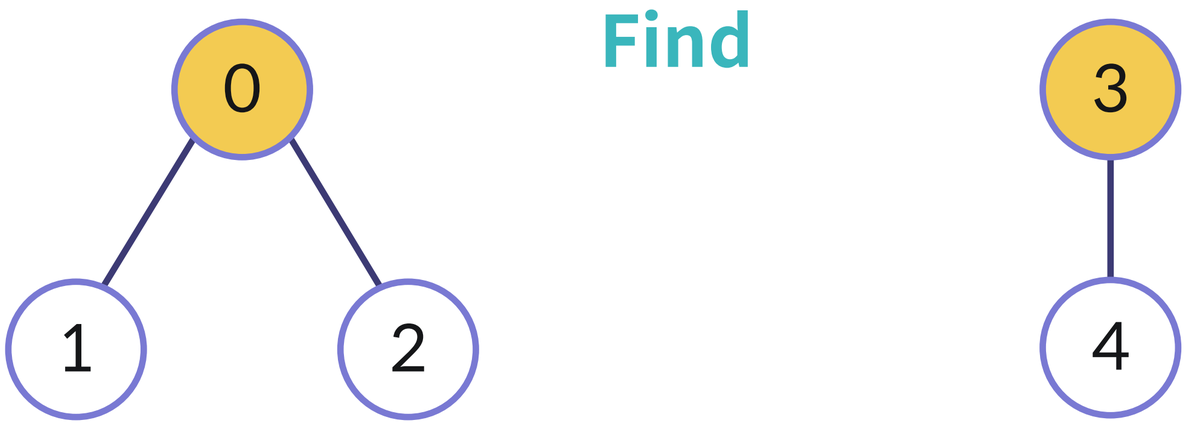




최소신장트리 (minimum spanning tree) - 크루스칼

사이클을 확인하는 방법 – Union find

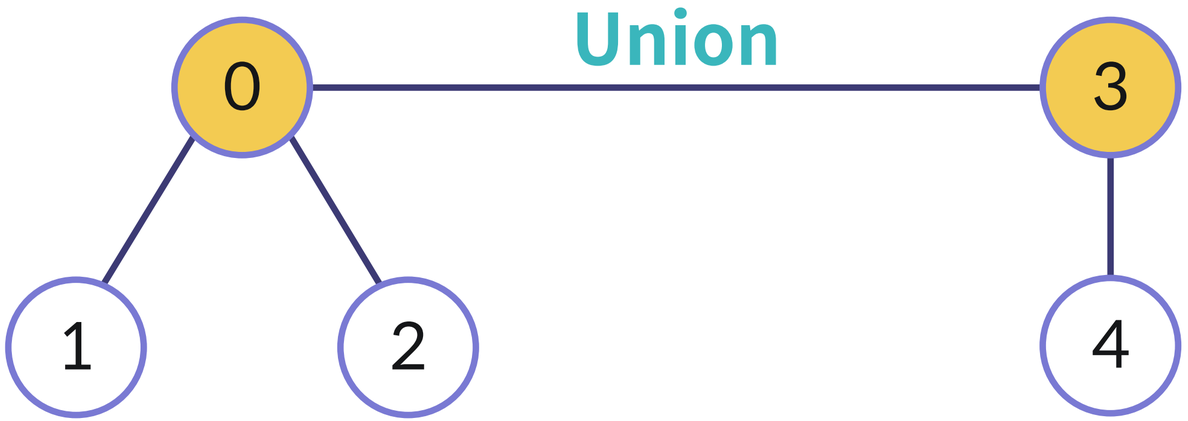
임의의 노드를 포함하는 트리 A와 트리 B가 동일한 트리인지 찾고 (**Find**)
두 트리를 한트리로 통합하는 과정 (**Union**)



최소신장트리 (minimum spanning tree) - 크루스칼

사이클을 확인하는 방법 – Union find

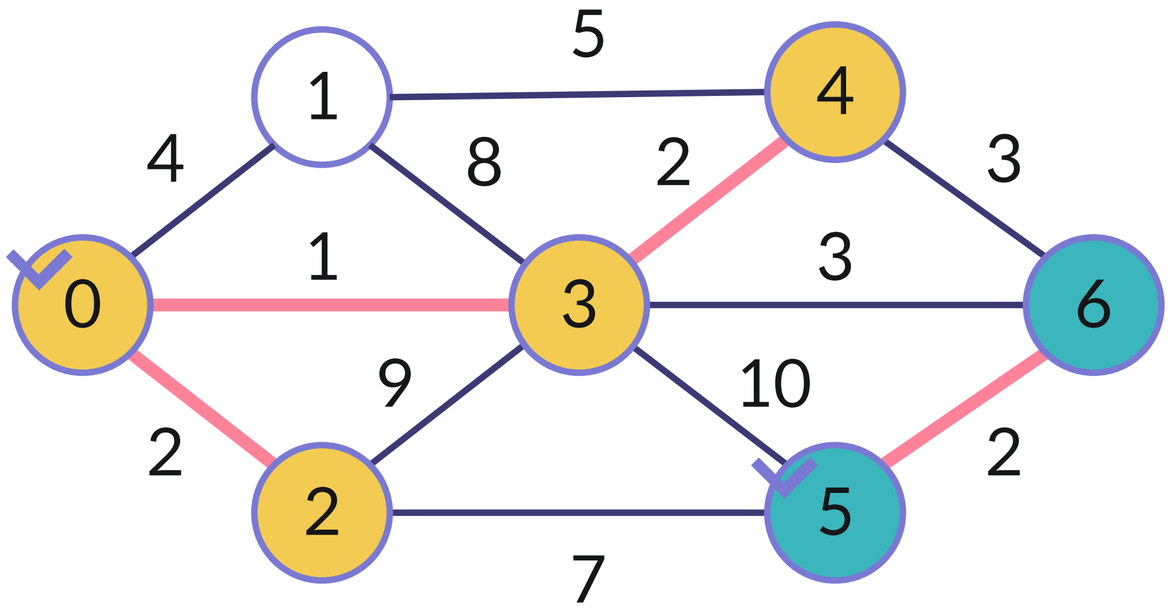
임의의 노드를 포함하는 트리 A와 트리 B가 동일한 트리인지 찾고 (**Find**)
두 트리를 한트리로 통합하는 과정 (**Union**)



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

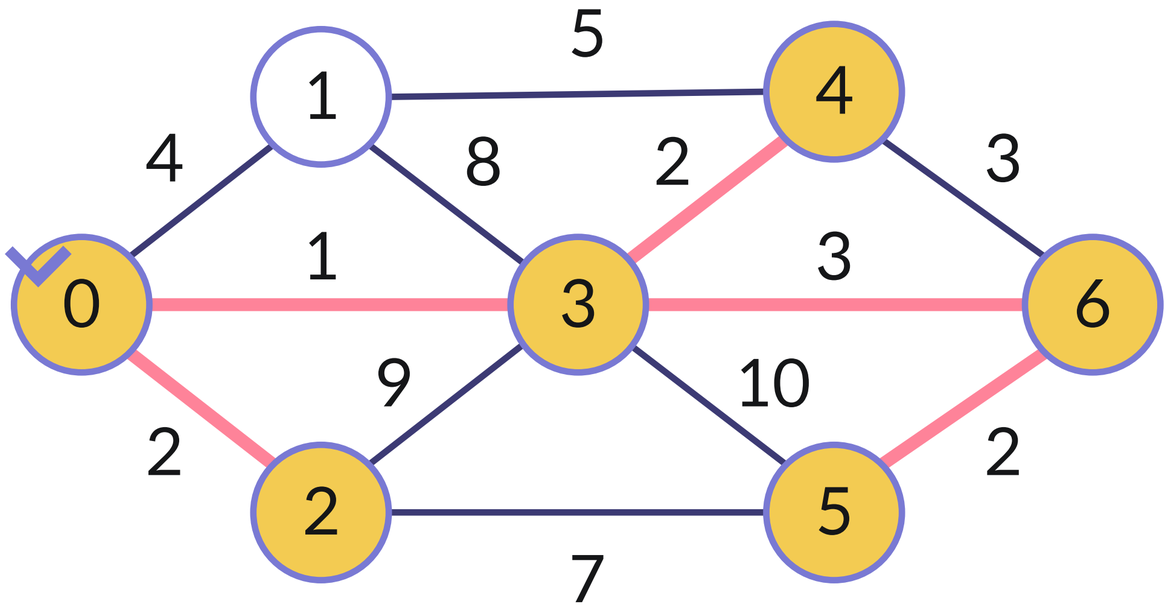
- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



최소신장트리 (minimum spanning tree) - 크루스칼

짧은 간선부터 골라 그래프에 포함시키는 방법

- 단, 트리에는 사이클이 없기 때문에, 사이클이 되는 경우는 건너뜁니다.



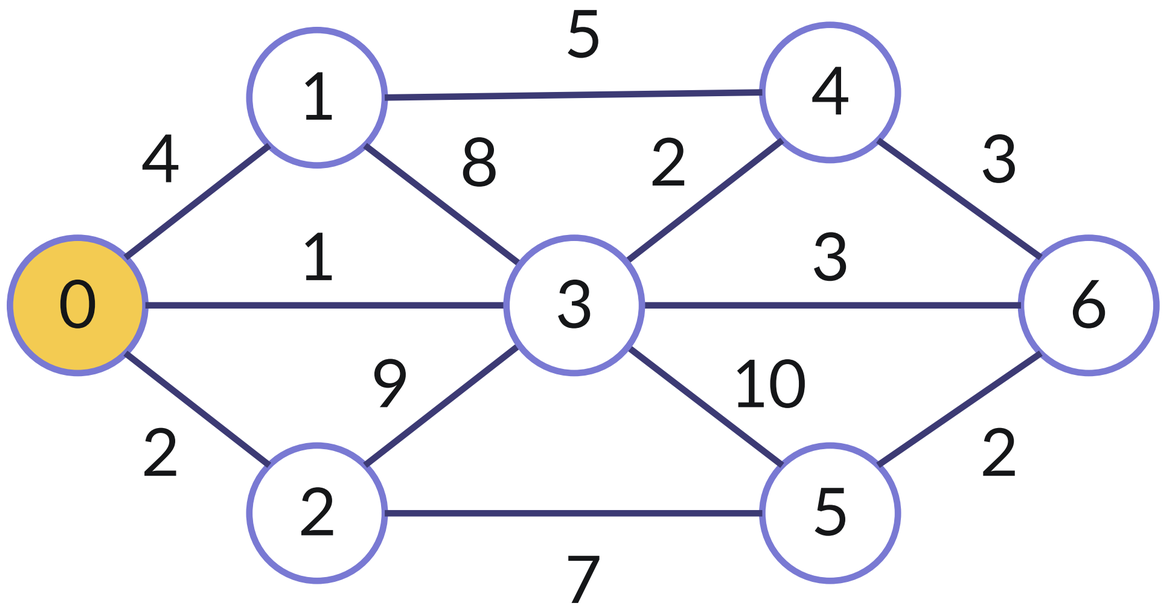
최소신장트리 (minimum spanning tree) - 크루스칼

Example

```
KRUSKAL(G):
    A = ∅
    For each vertex v ∈ G.V:
            MAKE-SET(v)
    For each edge (u, v) ∈ G.E ordered by
increasing order by weight(u, v):
            if FIND-SET(u) ≠ FIND-SET(v):
                A = A ∪ {(u, v)}
                UNION(u, v)
    return A
```

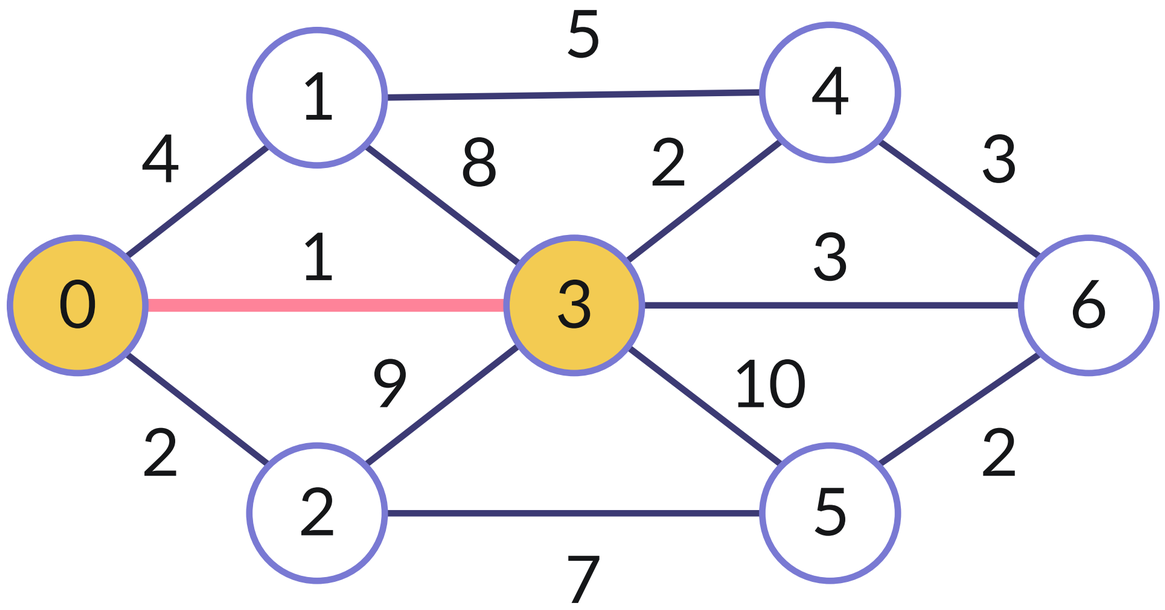
최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.



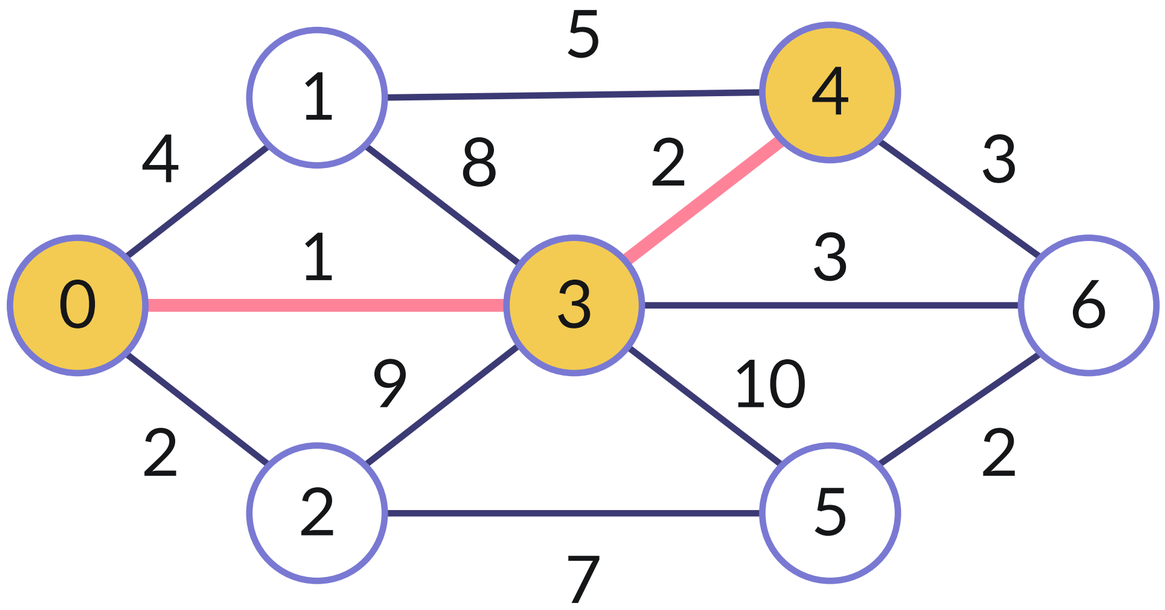
최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.



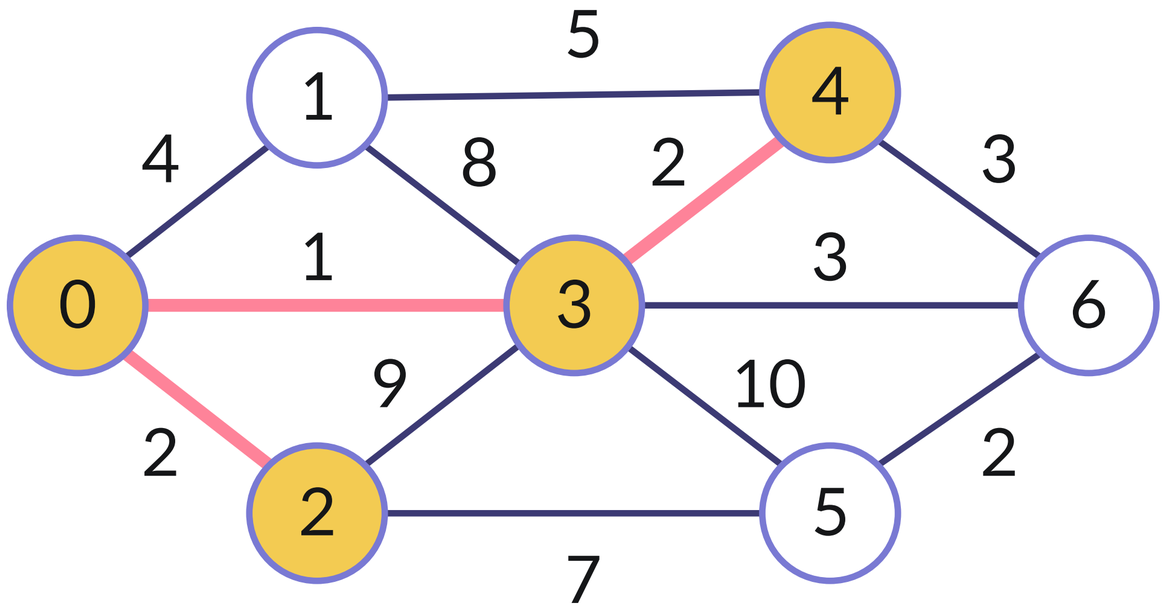
최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.



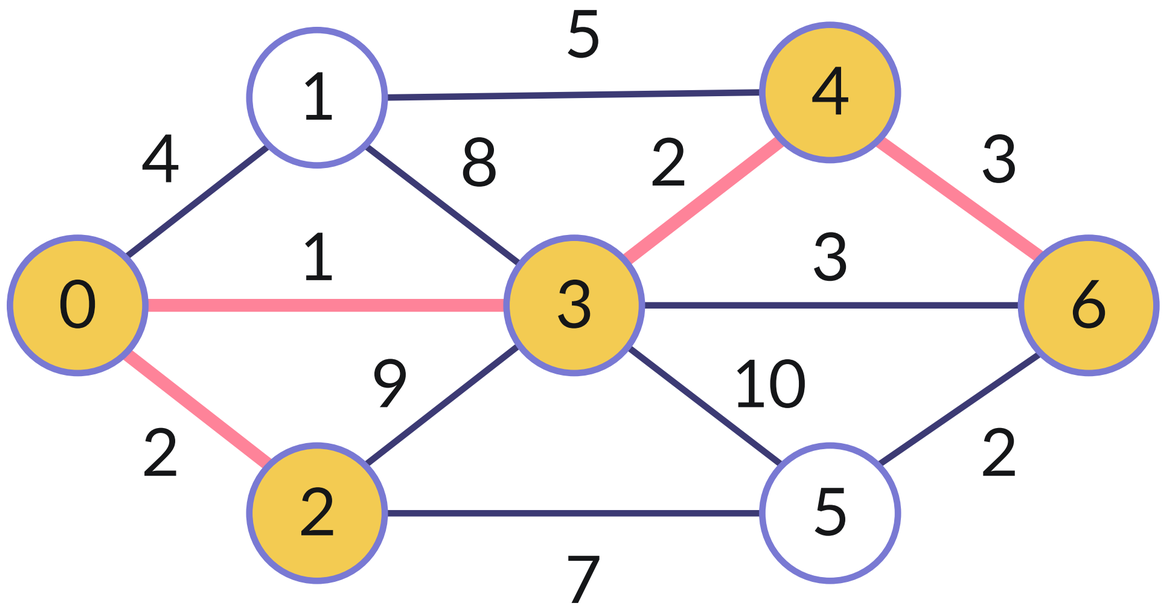
최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.



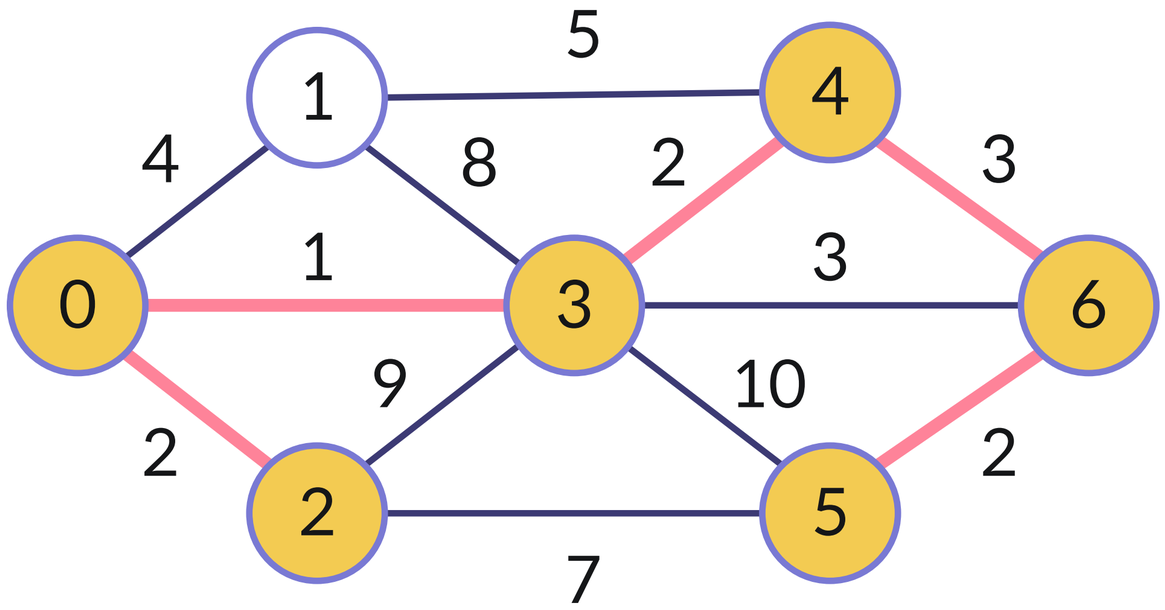
최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.



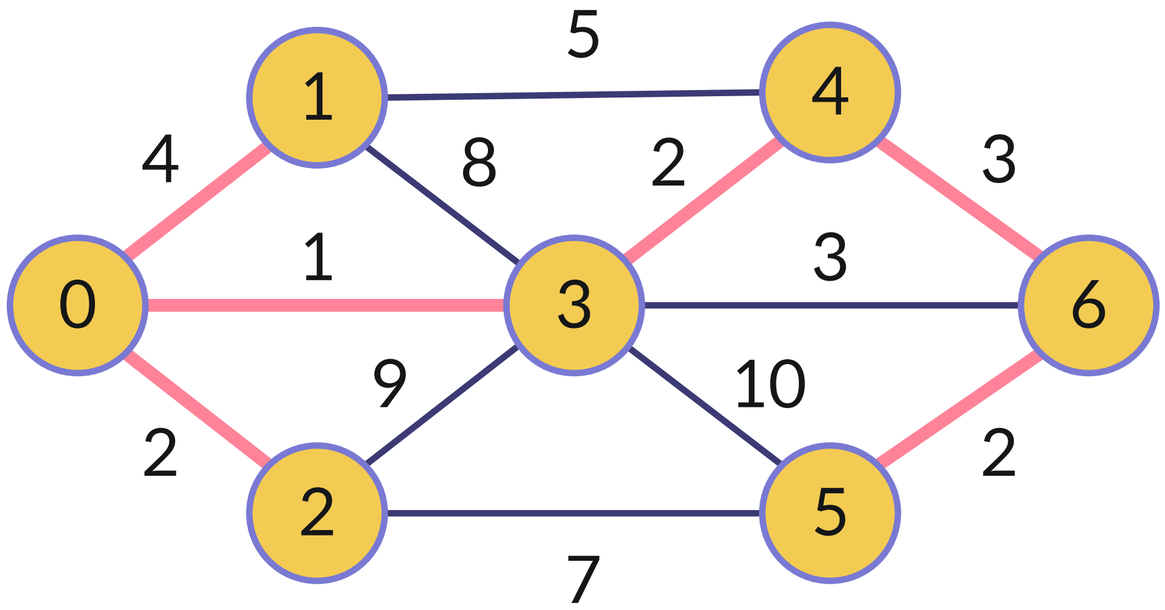
최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.



최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.



최소신장트리 (minimum spanning tree) - 프림

작은 트리에서 연결되어있는 **간선들을 하나씩 확장**하면서 트리를 만듭니다.

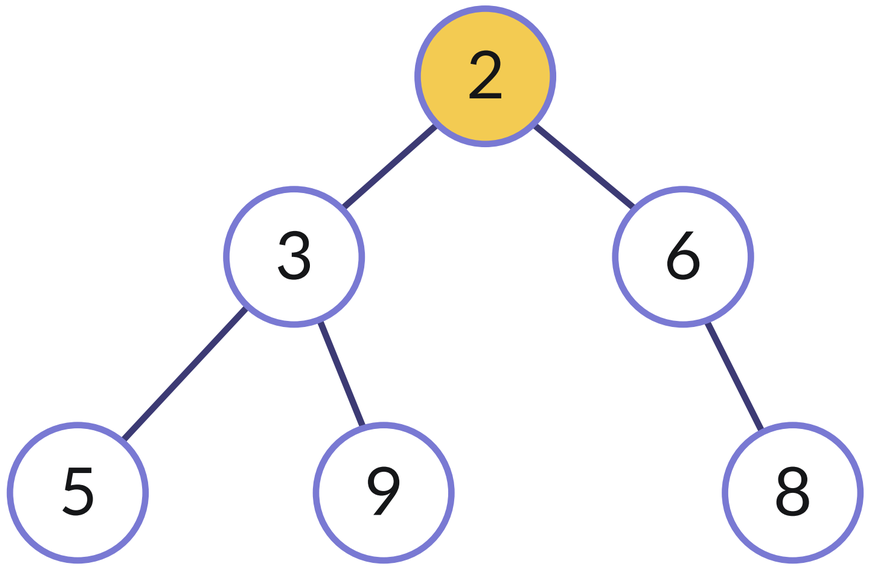
구현 핵심

1. 현재 만들어진 트리와 연결되어있는 가장 작은 가중치의 간선을 찾는 것

최소신장트리 (minimum spanning tree) - 프림

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

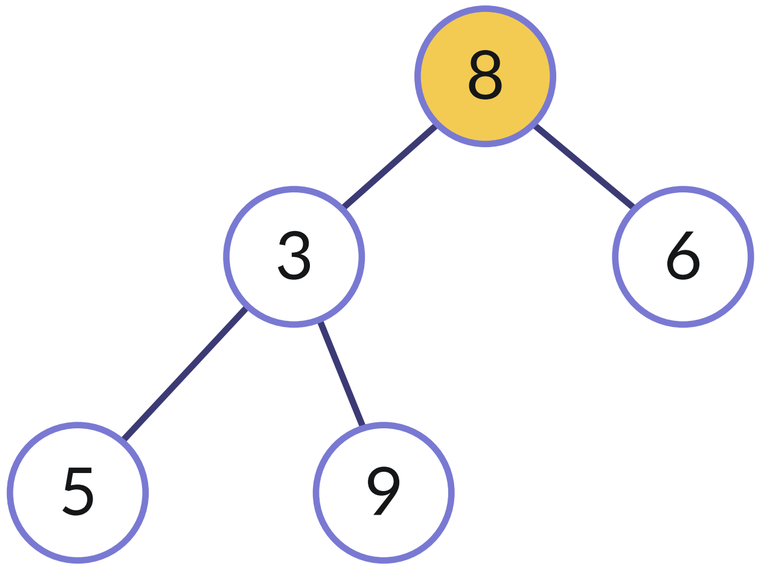
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최소신장트리 (minimum spanning tree) - 프림

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

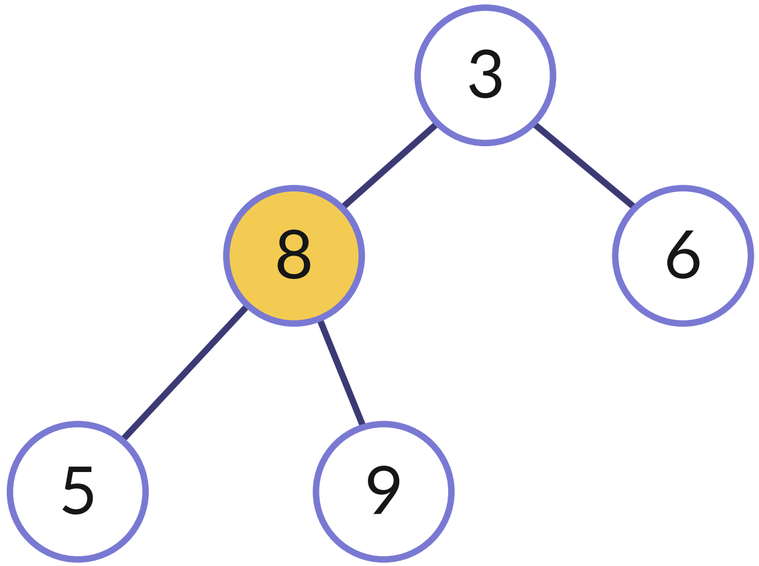
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최소신장트리 (minimum spanning tree) - 프림

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

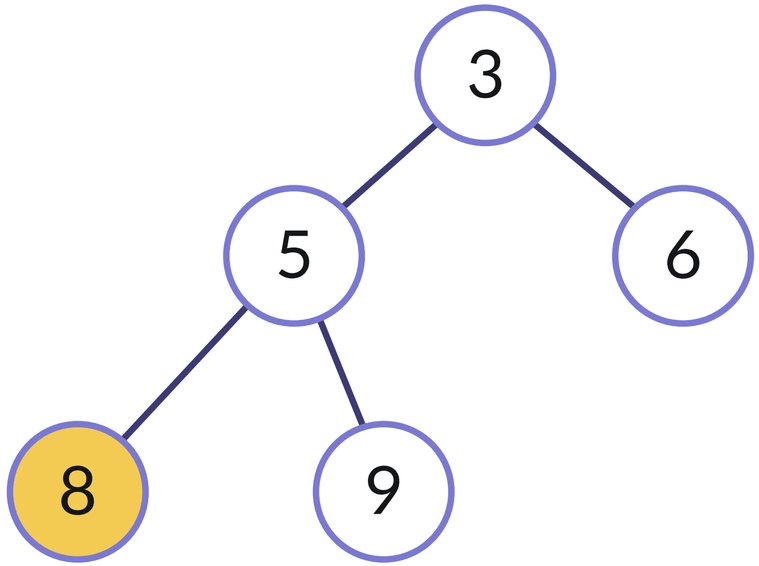
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최소신장트리 (minimum spanning tree) - 프림

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

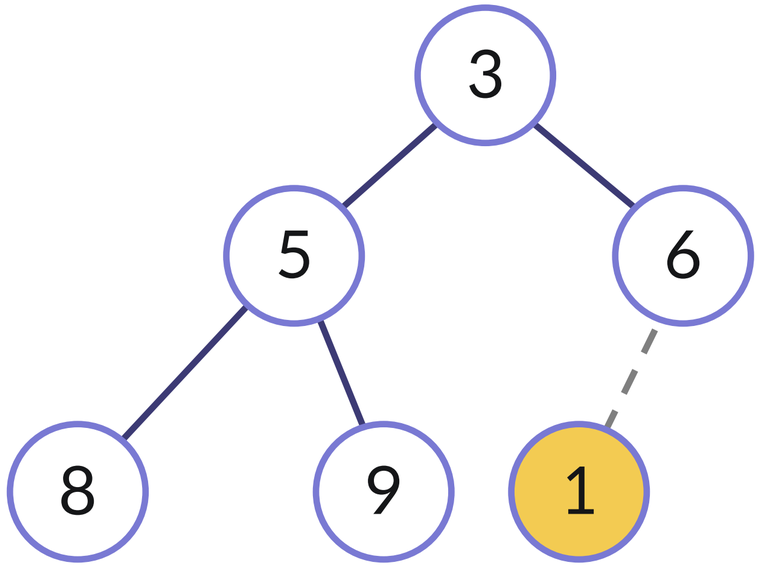
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최소신장트리 (minimum spanning tree) - 프림

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

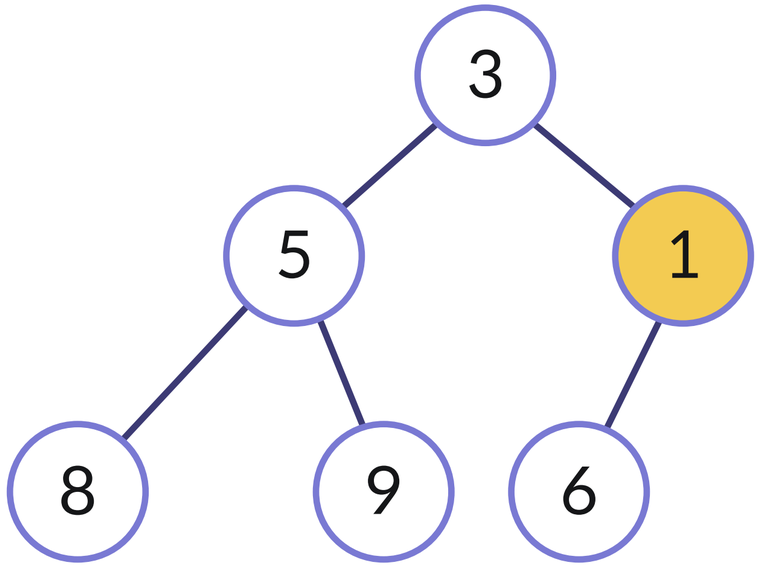
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최소신장트리 (minimum spanning tree) - 프림

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

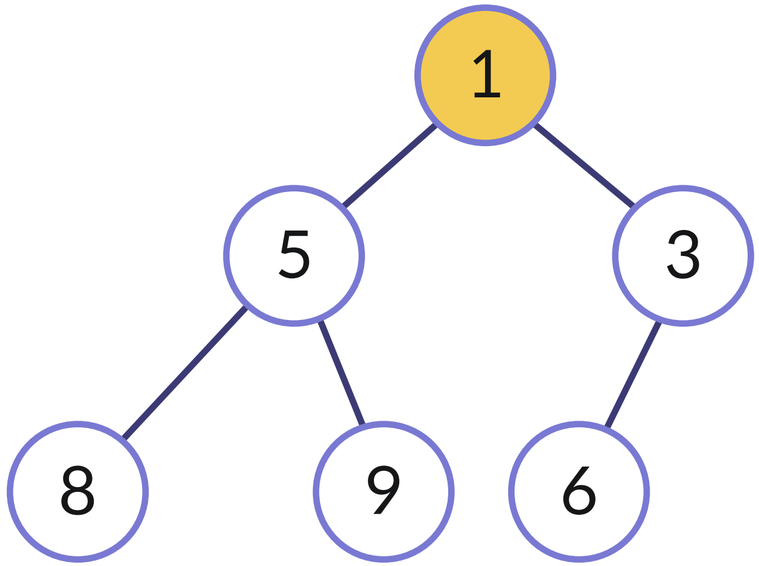
예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최소신장트리 (minimum spanning tree) - 프림

가지고 있는 값 중 가장 작은 값 찾기 – 우선순위큐

예) 3 9 2 8 5 6 중 가장 작은 수를 찾아서 사용하려면?



최소신장트리 (minimum spanning tree) - 프림

Example

```
PRIM(G):
    U = { 1 };
    while (U ≠ V)
    let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
        T = T ∪ {(u, v)}
        U = U ∪ {v}
```

05 정리

그래프 알고리즘에는 …

- 다양한 문제를 그래프로 표현한 후
  대표적인 그래프 문제 유형에 대응하여 풀 수 있습니다.

- 대표적인 그래프 알고리즘에는 **최단거리 알고리즘**과 **최소비용트리**가 있습니다.

- **최단거리 알고리즘** 중 하나의 정점에서 다른 모든 정점으로의 최단거리를 구하는데 **다익스트라**, **벨만포드** 등의 알고리즘이 있습니다.
  - 벨만포드는 음의 가중치를 허용하는 대신 느립니다.

- **최소비용트리**는 모든 정점을 최소비용으로 잇는 트리를 말하는데
  프림, **크루스칼** 등의 알고리즘으로 구현할 수 있습니다.

Contact

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io