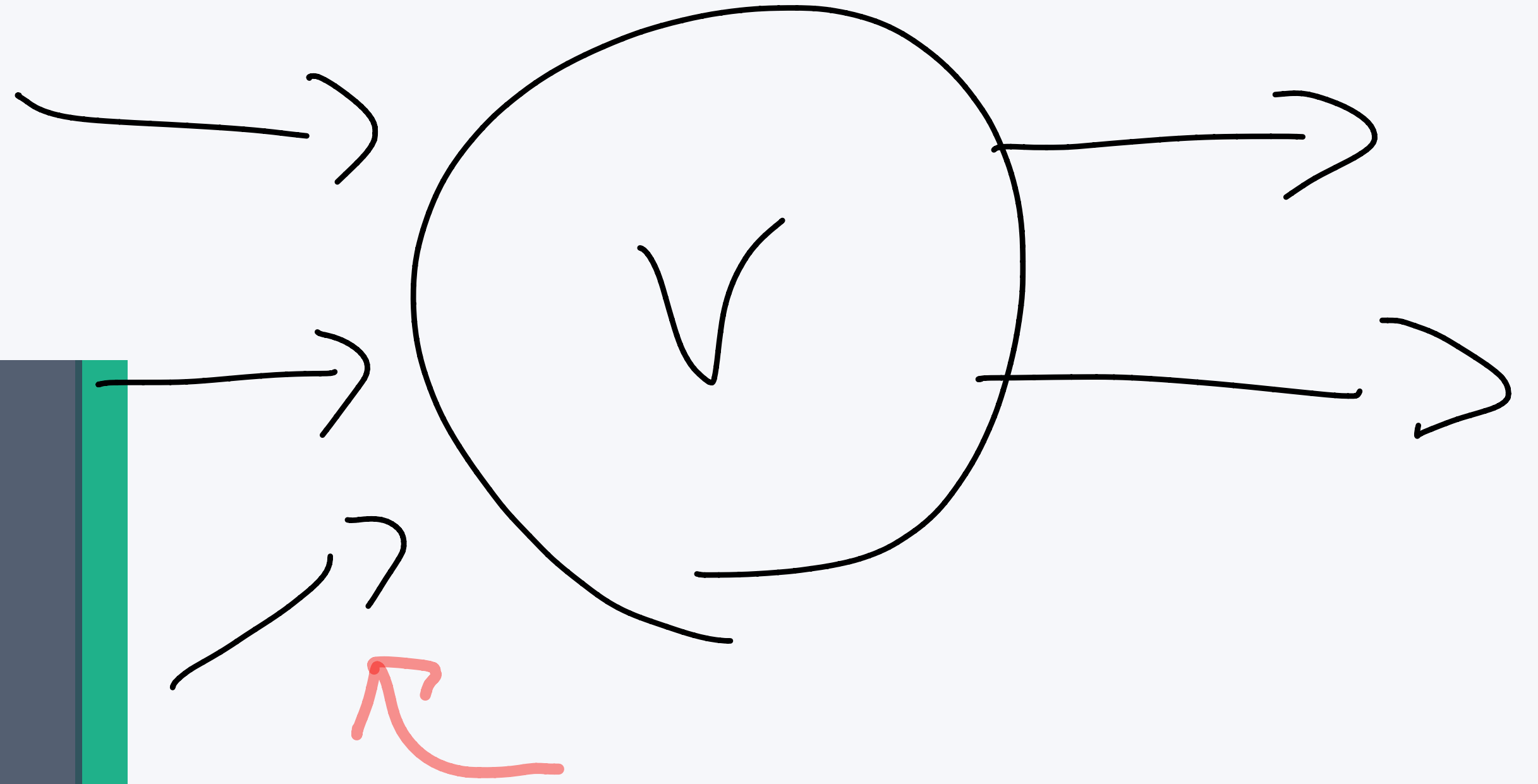


# 그래프 2

최백준 [choi@startlink.io](mailto:choi@startlink.io)

---

# DAG



- ① In degree
- ② Out degree

# DAG

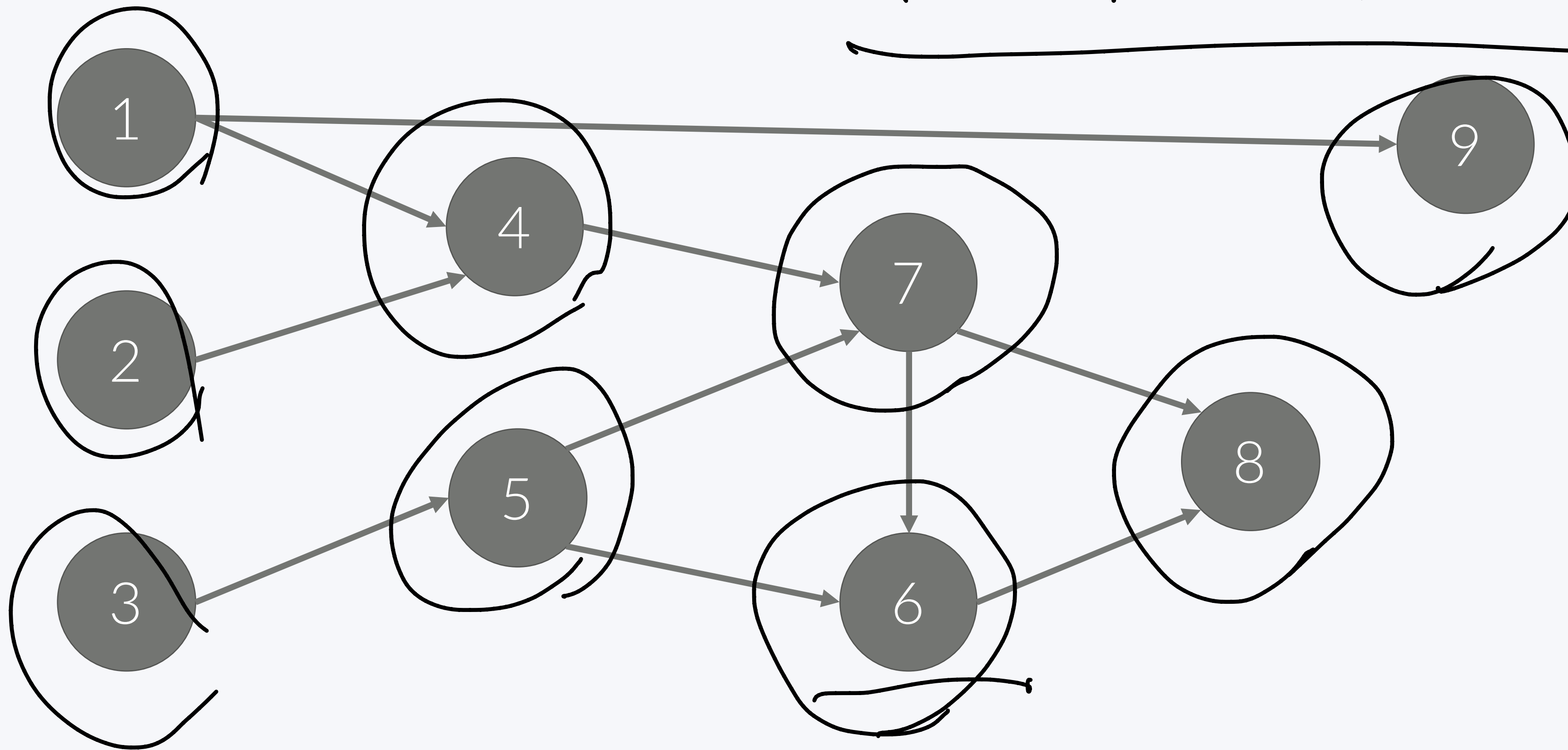
Directed Acyclic Graph

- 사이클이 없는 방향 있는 그래프를 DAG라고 한다.

위상정렬

3

1 2 3 4 5 6 7 8 9  
1 2 3 4 5 6 7 8 9



# 위상 정렬

---

# 위상 정렬

Topological Sort

5

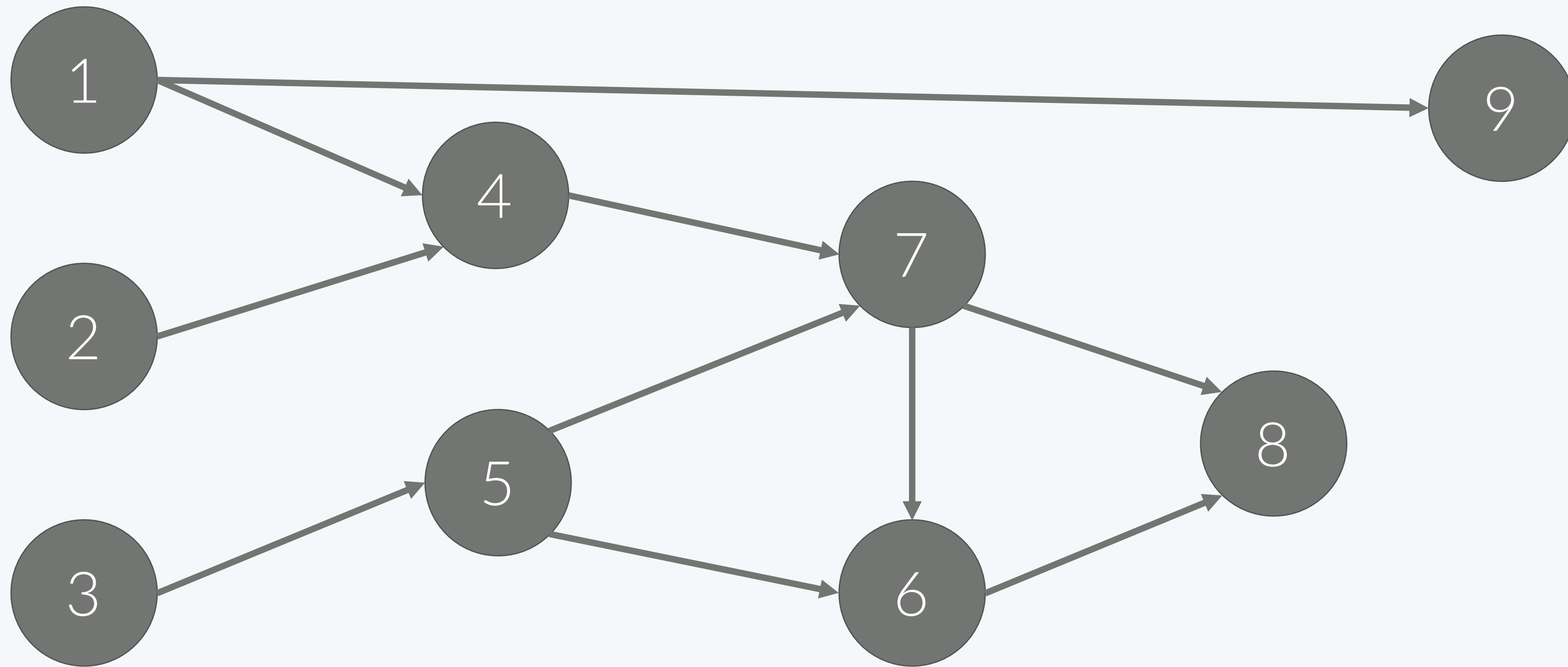
- 어떤 일을 하는 순서를 찾는 알고리즘이다
- 1 -> 2
- 2를 하기 전에 1을 먼저 해야 한다.

# 위상 정렬

## Topological Sort

6

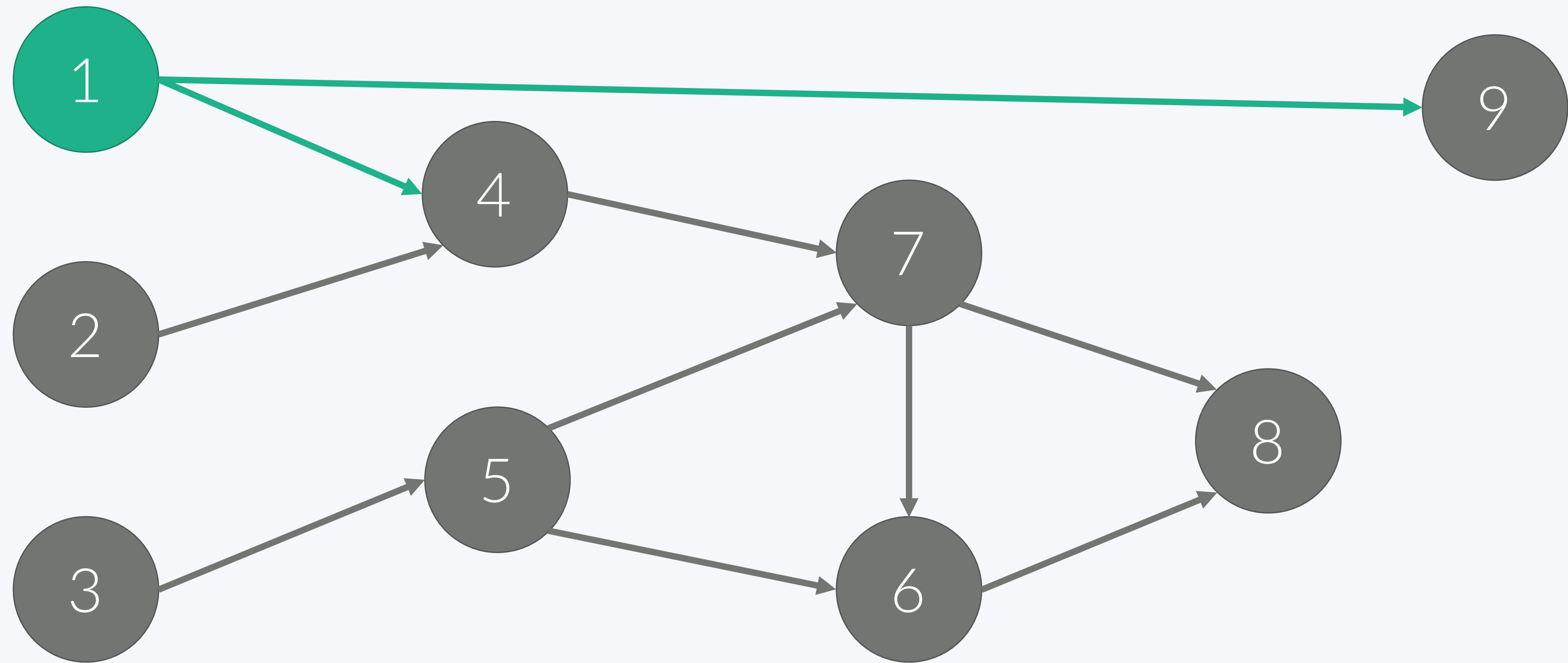
- 큐에 가장 들어있는 것은
- 들어오는 간선의 개수가 0인 것이다.
- 순서:
- 큐: 1 2 3



# 위상 정렬

Topological Sort

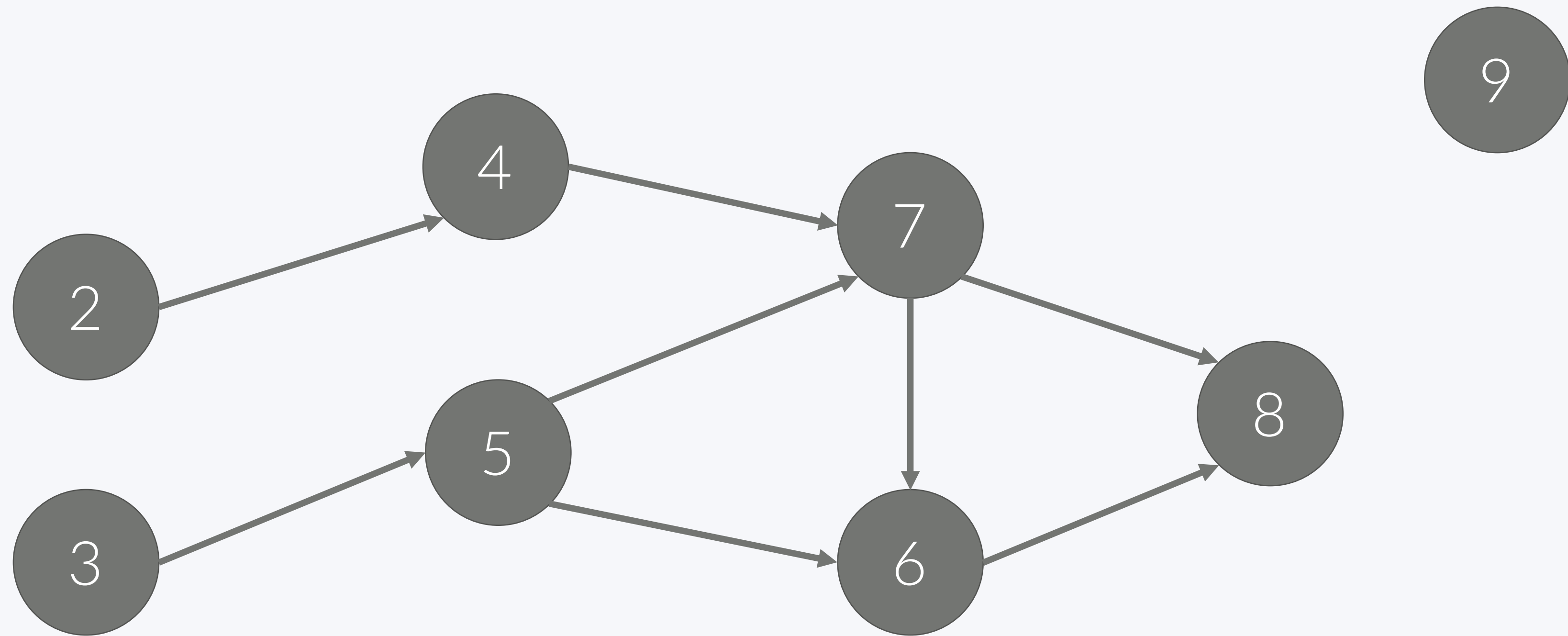
- 순서: 1
- 큐: 2 3



# 위상 정렬

Topological Sort

- 순서: 1
- 큐: 2 3 9

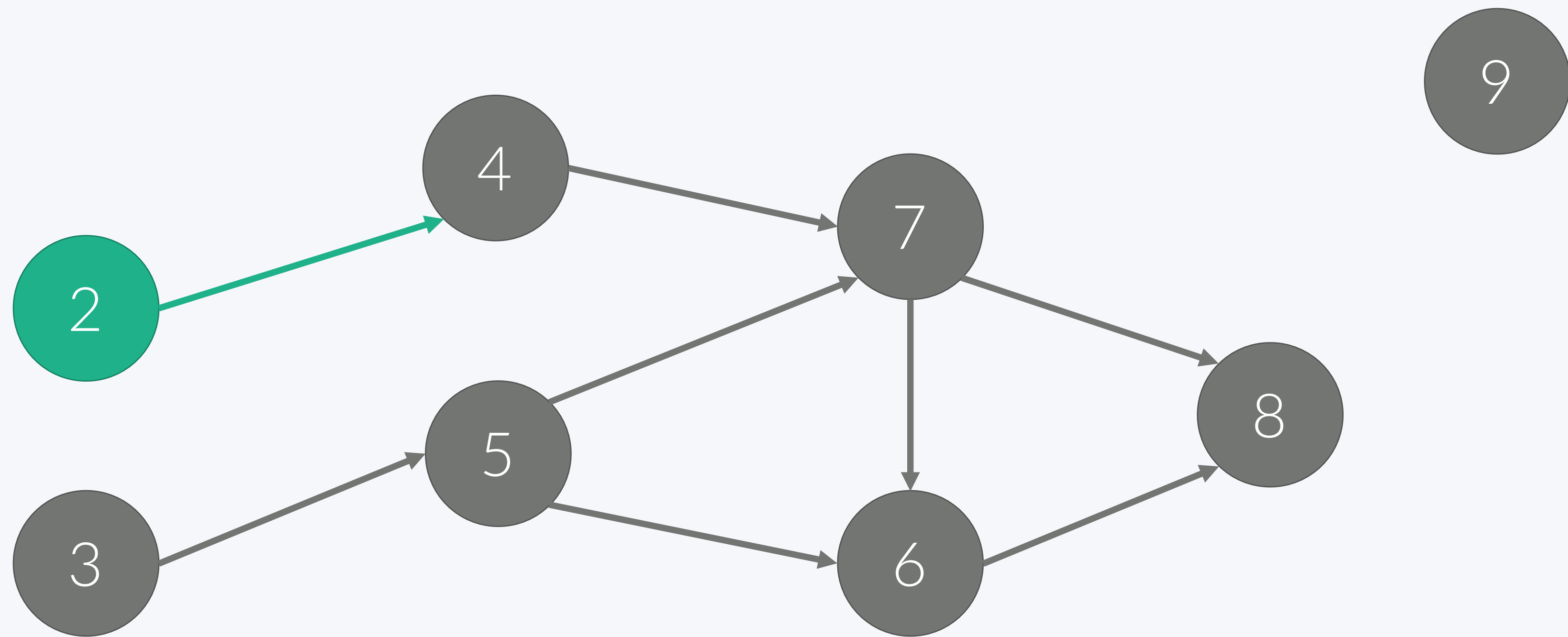




# 위상 정렬

Topological Sort

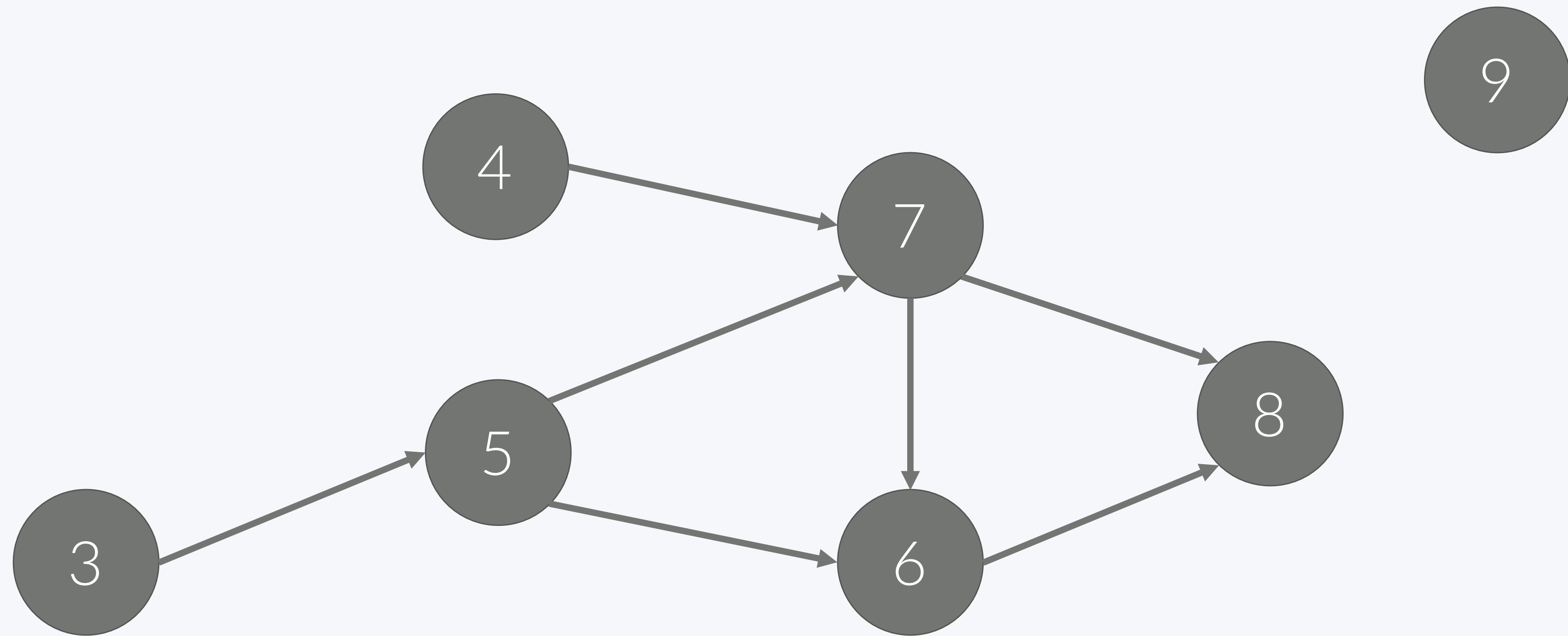
- 순서: 1 2
- 큐: 3 9



# 위상 정렬

Topological Sort

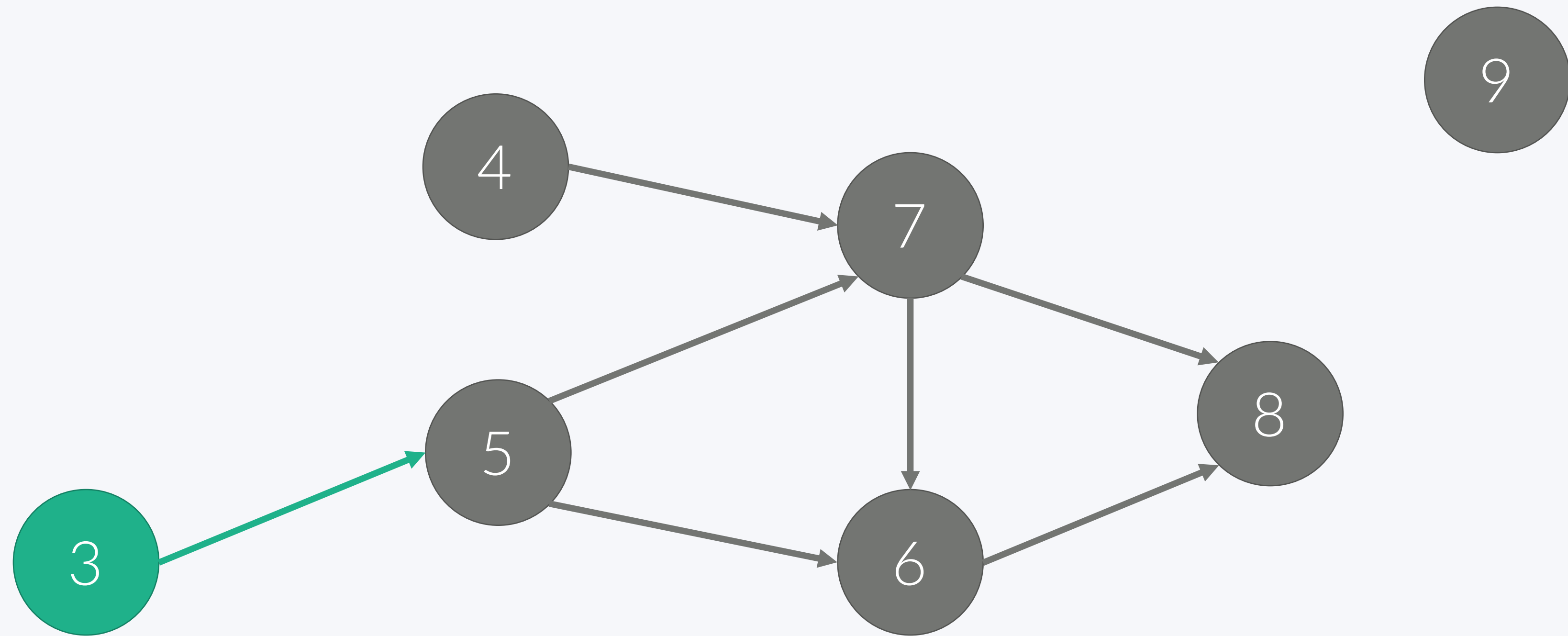
- 순서: 1 2
- 큐: 3 9 4



# 위상 정렬

Topological Sort

- 순서: 1 2 3
- 큐: 9 4



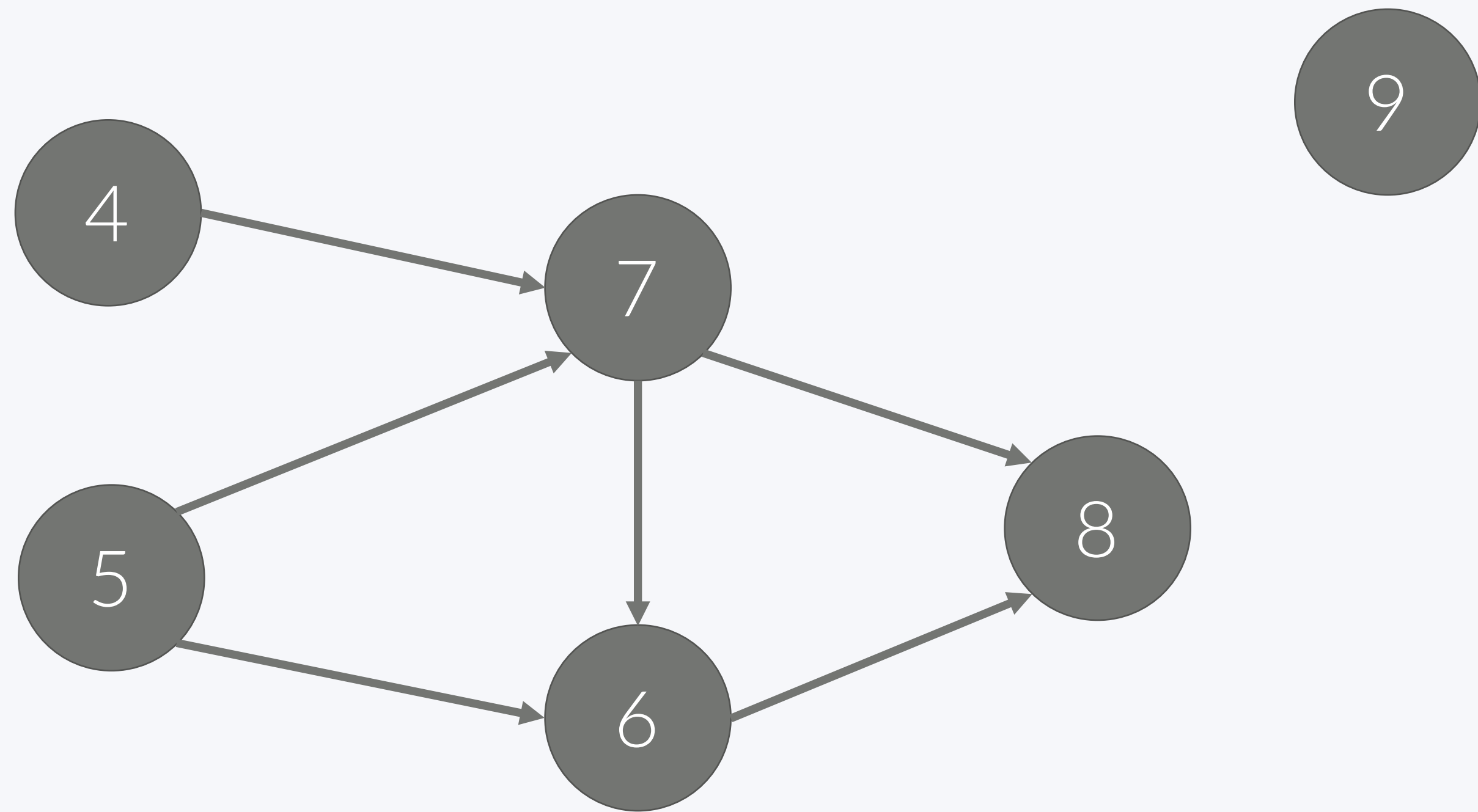
# 위상 정렬

Topological Sort

• 순서: 1 2 3

• 큐: 9 4 5

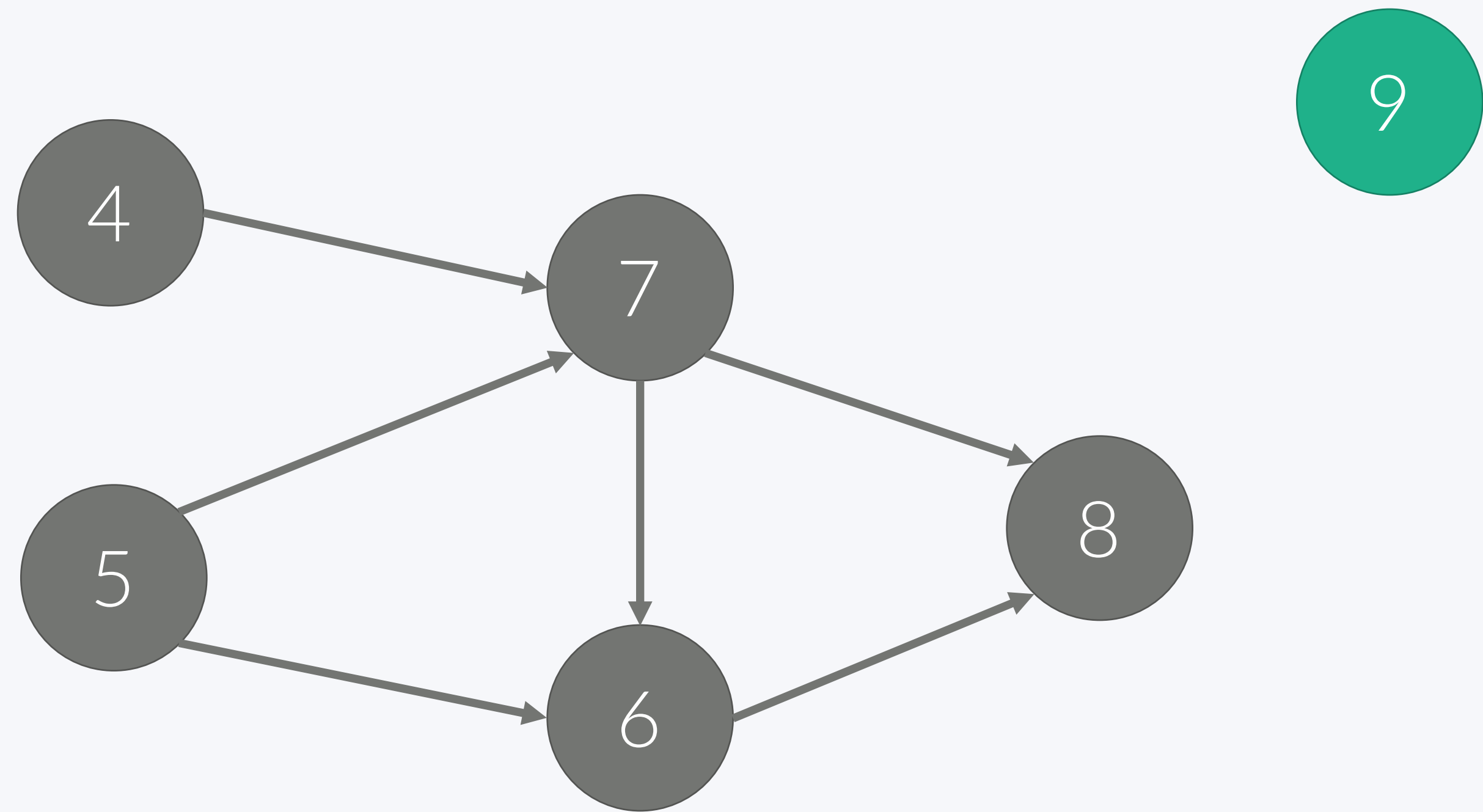
이러게  $\Rightarrow$  큐 :  $O(V+E)$   
시간  $\Rightarrow$  우선순위 큐 :  $O(V \lg V)$   
알



# 위상 정렬

Topological Sort

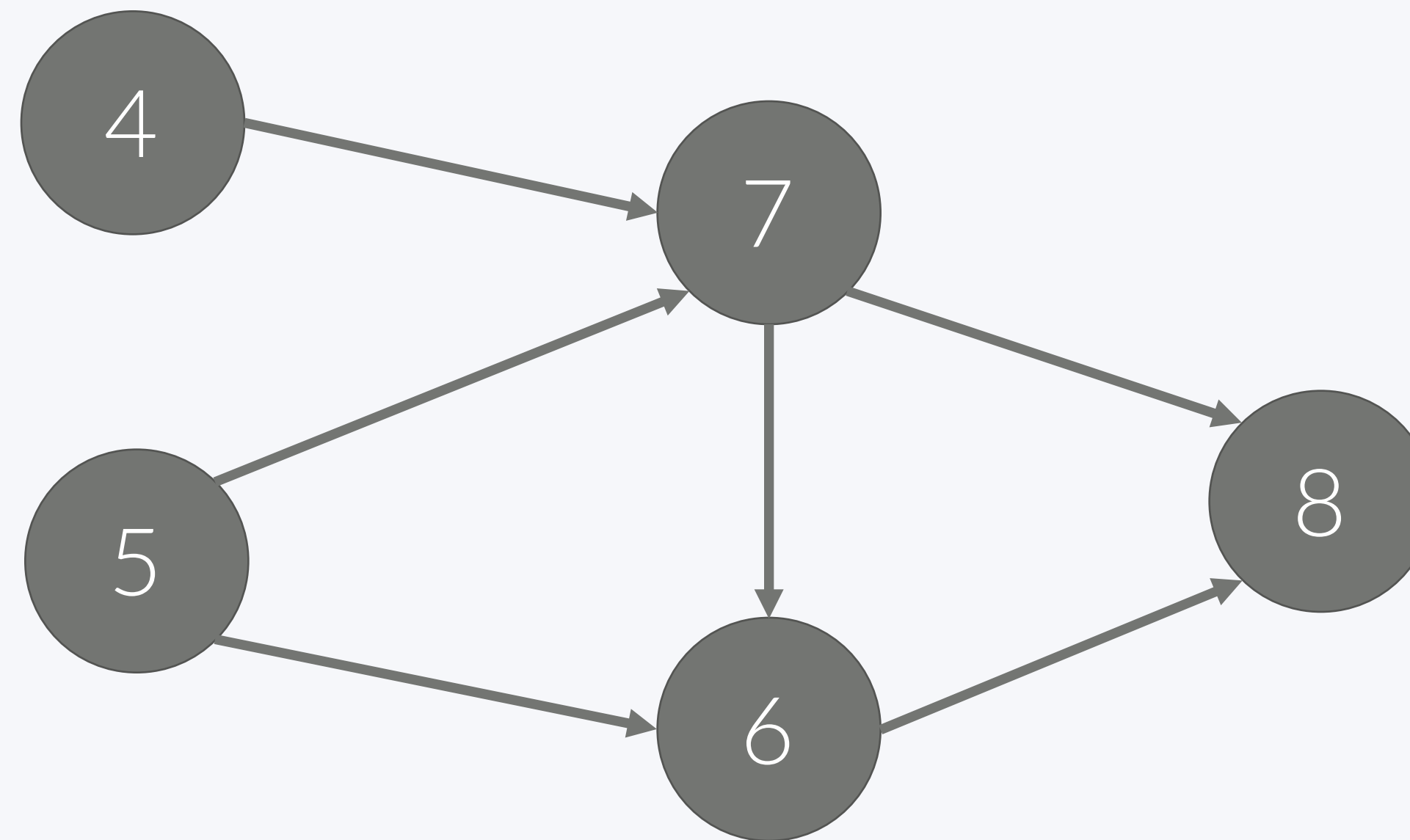
- 순서: 1 2 3 9
- 큐: 4 5



# 위상 정렬

Topological Sort

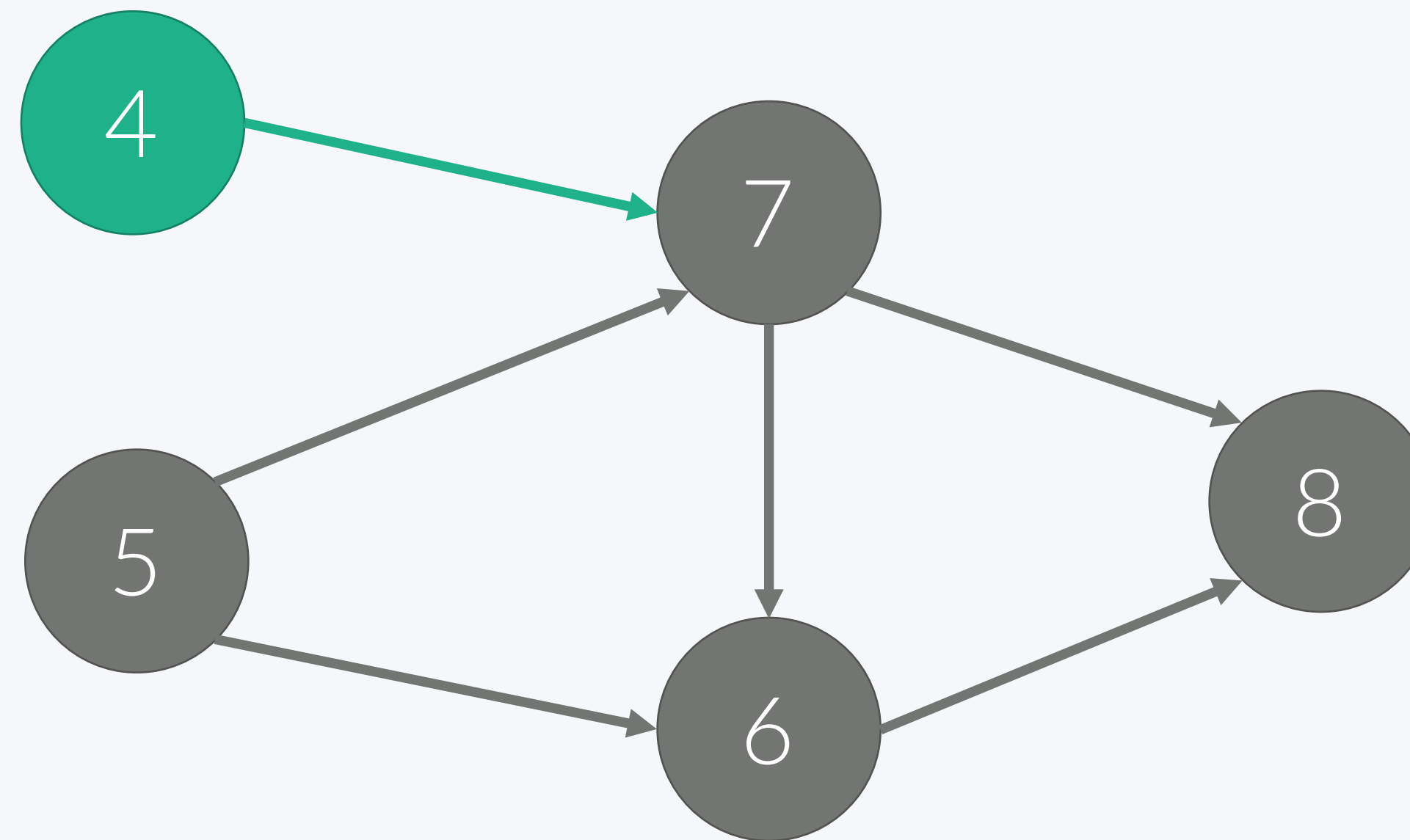
- 순서: 1 2 3 9
- 큐: 4 5



# 위상 정렬

Topological Sort

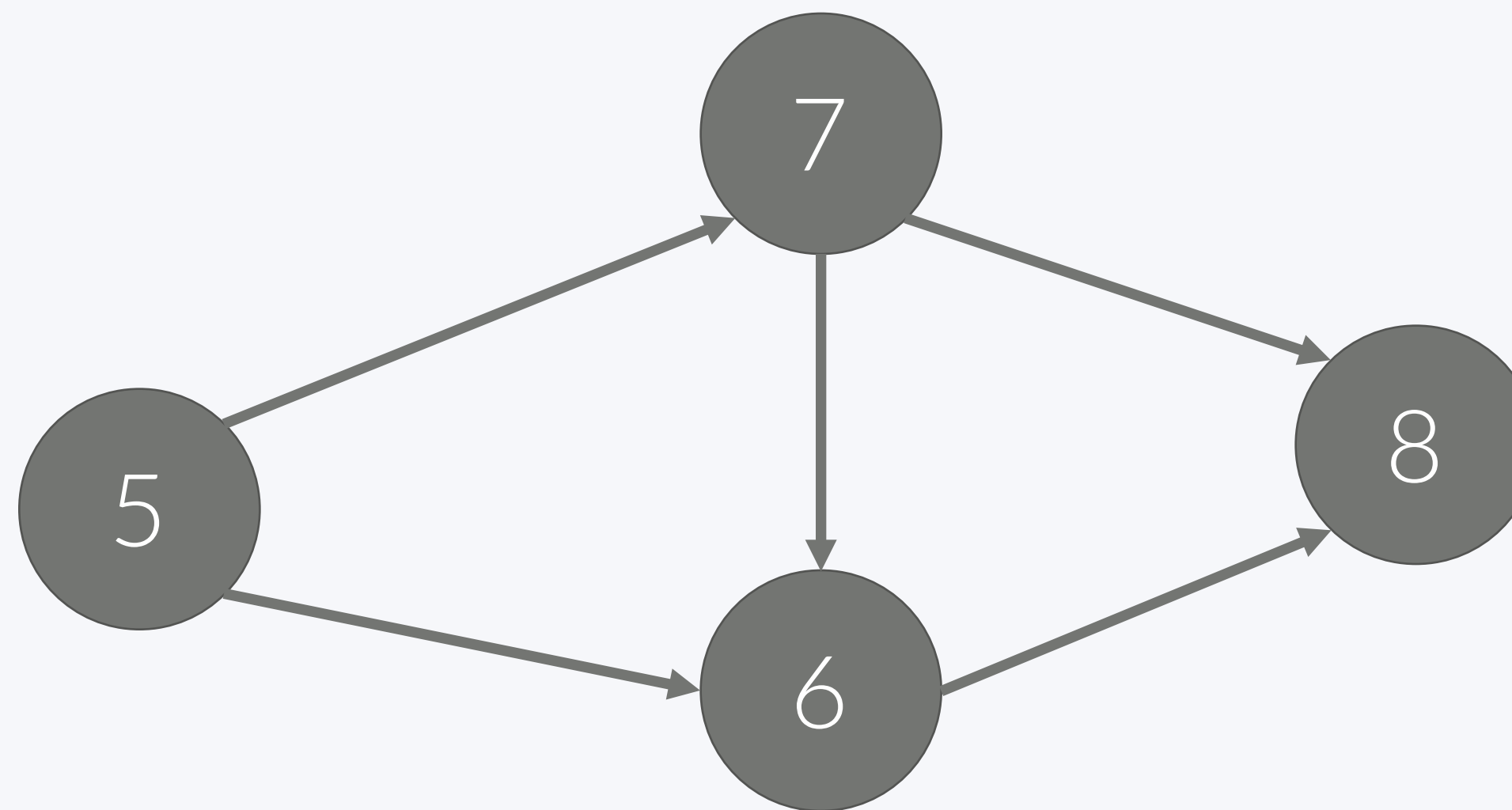
- 순서: 1 2 3 9 4
- 큐: 5



# 위상 정렬

Topological Sort

- 순서: 1 2 3 9 4
- 큐: 5

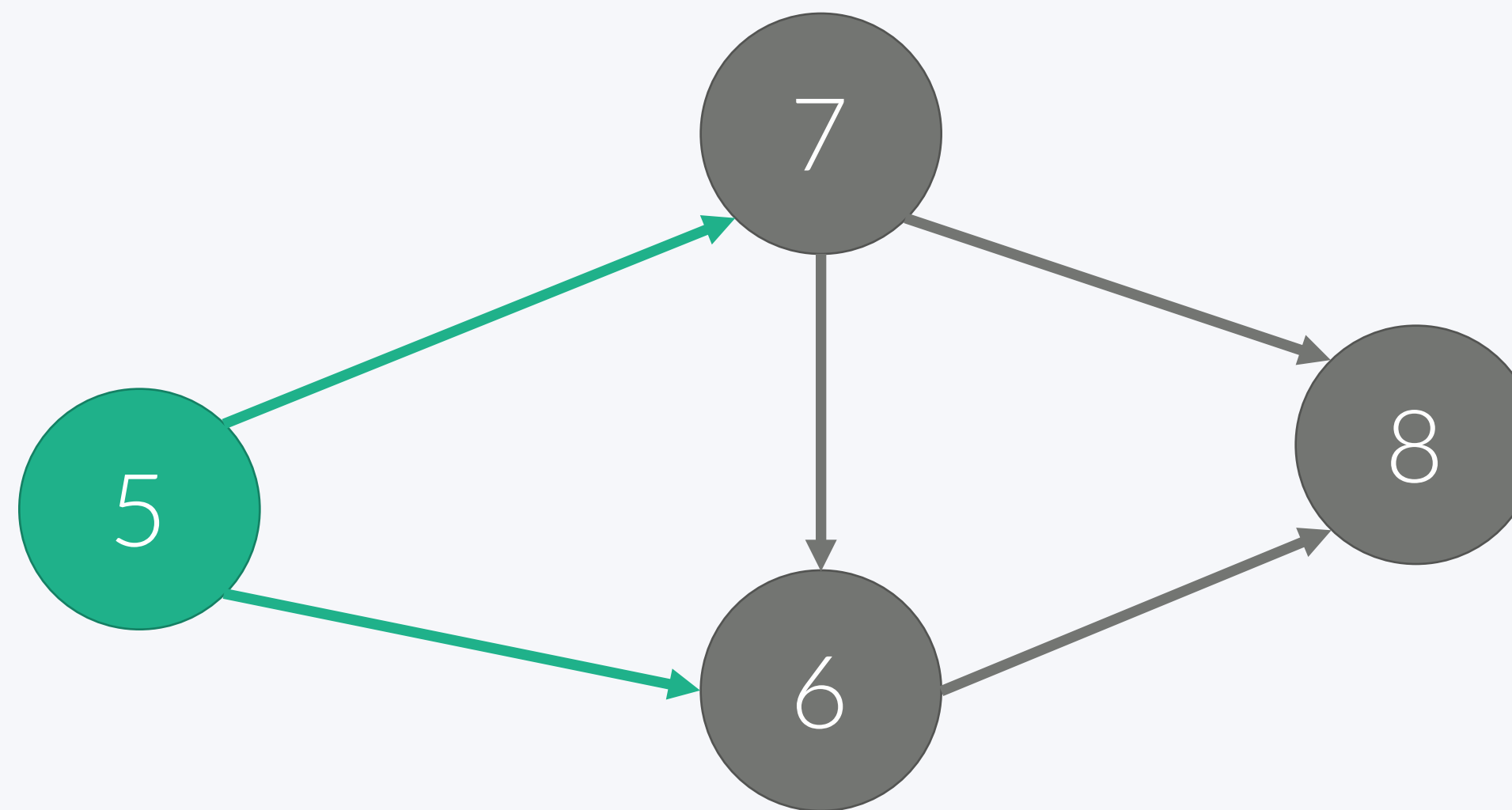




# 위상 정렬

Topological Sort

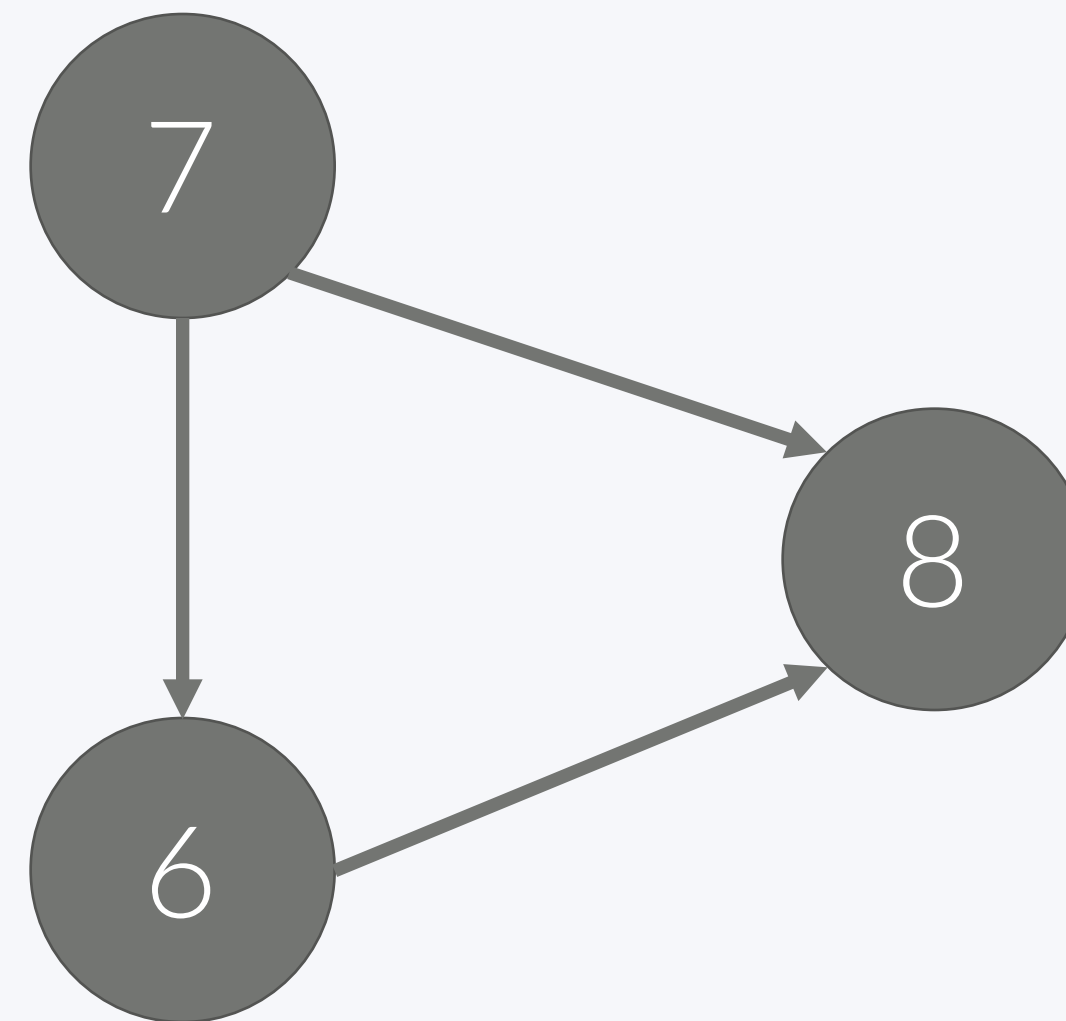
- 순서: 1 2 3 9 4 5
- 큐:



# 위상 정렬

Topological Sort

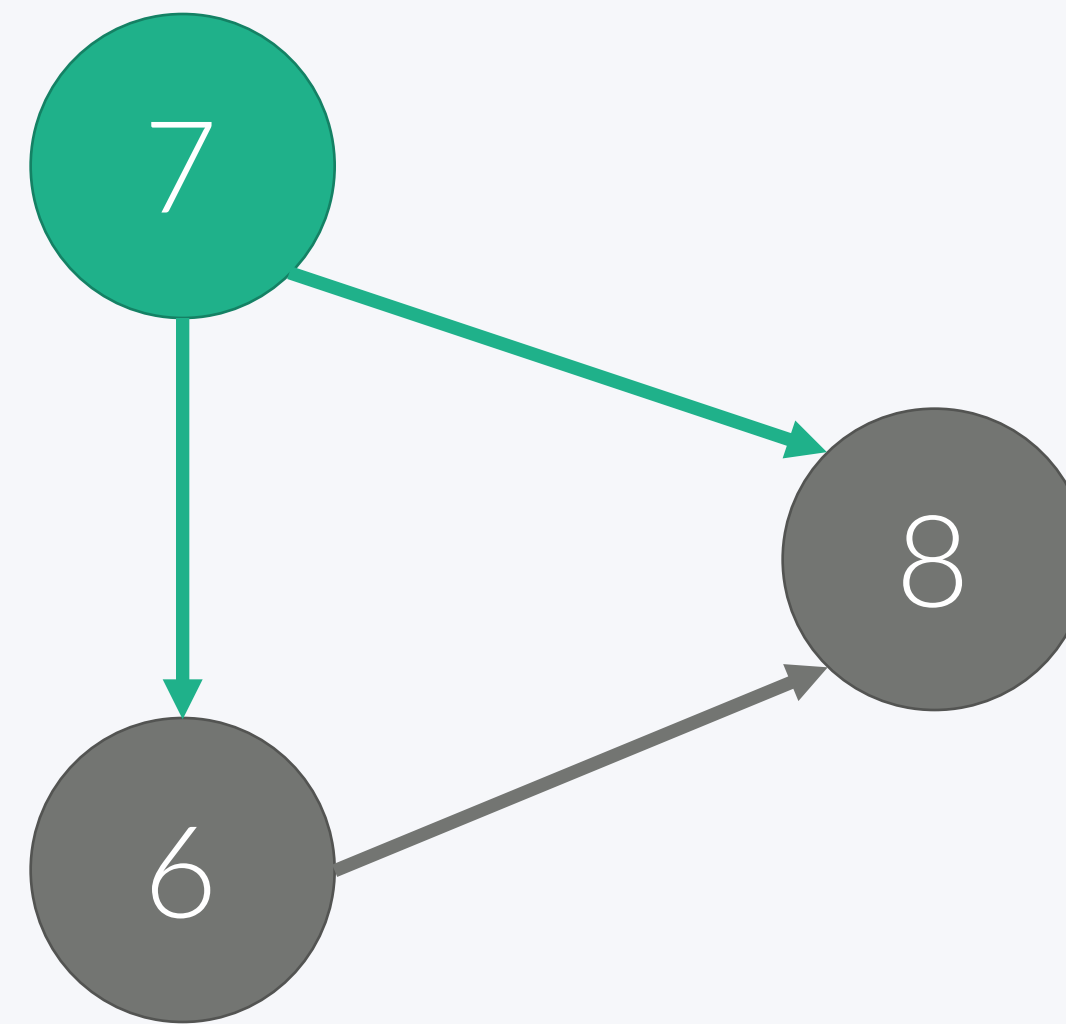
- 순서: 1 2 3 9 4 5
- 큐: 7



# 위상 정렬

Topological Sort

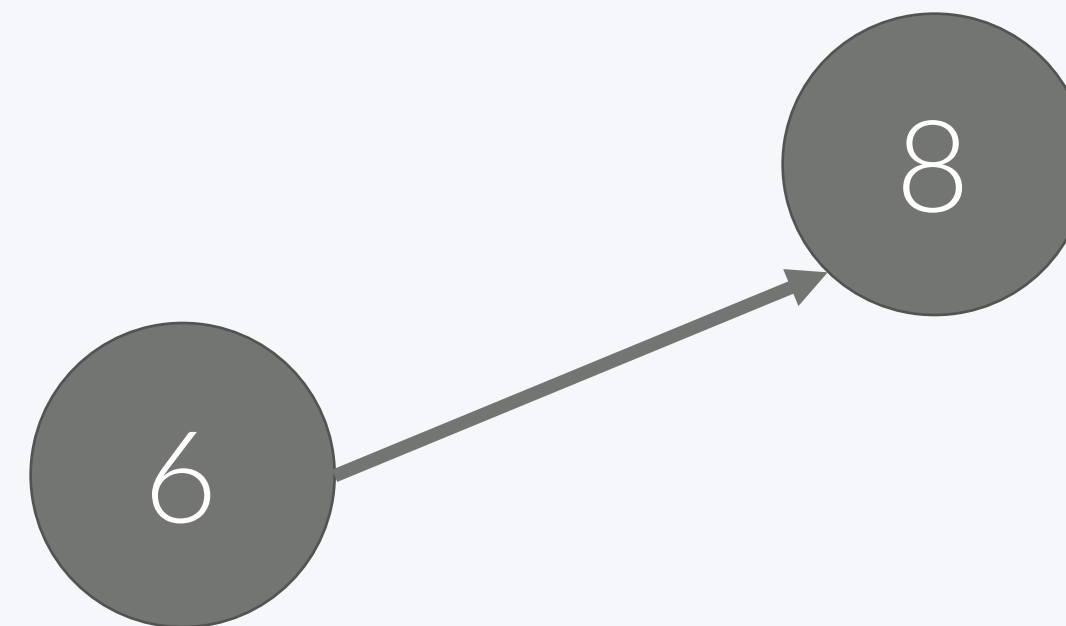
- 순서: 1 2 3 9 4 5 7
- 큐:



# 위상 정렬

Topological Sort

- 순서: 1 2 3 9 4 5 7
- 큐: 6

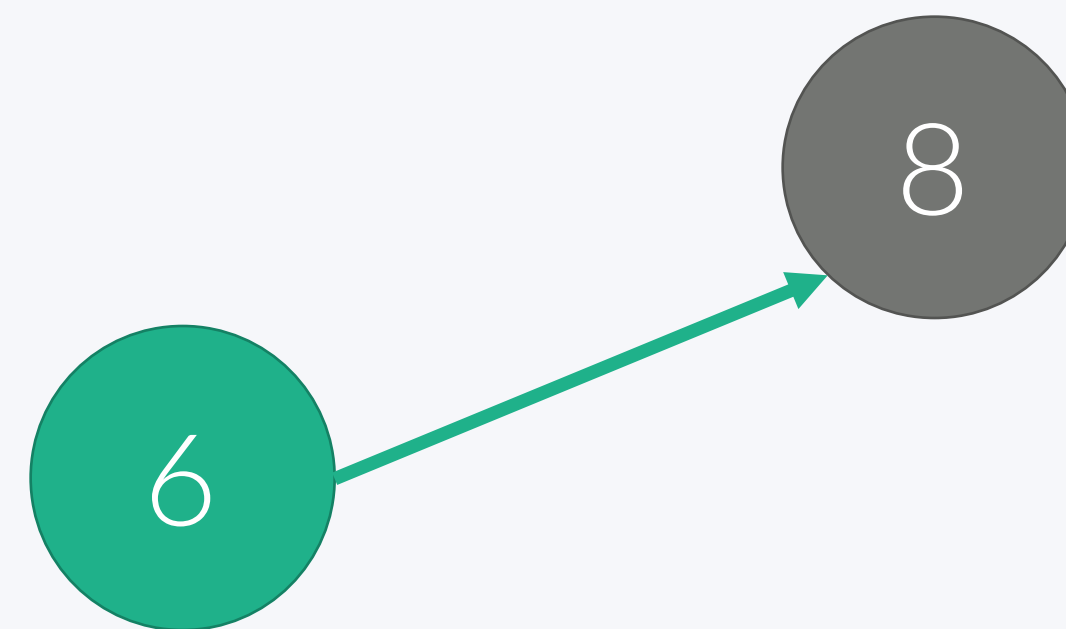


# 위상 정렬

Topological Sort

- 순서: 1 2 3 9 4 5 7 6 8
- 큐:

1 2 3 9 4 5 7 6 8  
1 1 1 1  
1 2 3 4 5 7 6 8 9



# 위상 정렬

Topological Sort

- 순서: 1 2 3 9 4 5 7 6
- 큐: 8

# 위상 정렬

Topological Sort

- 순서: 1 2 3 9 4 5 7 6 8
- 큐:

# 위상 정렬

Topological Sort

24

- 순서: 1 2 3 9 4 5 7 6 8
- 큐:



# 위상 정렬

## Topological Sort

25

- 가장 처음

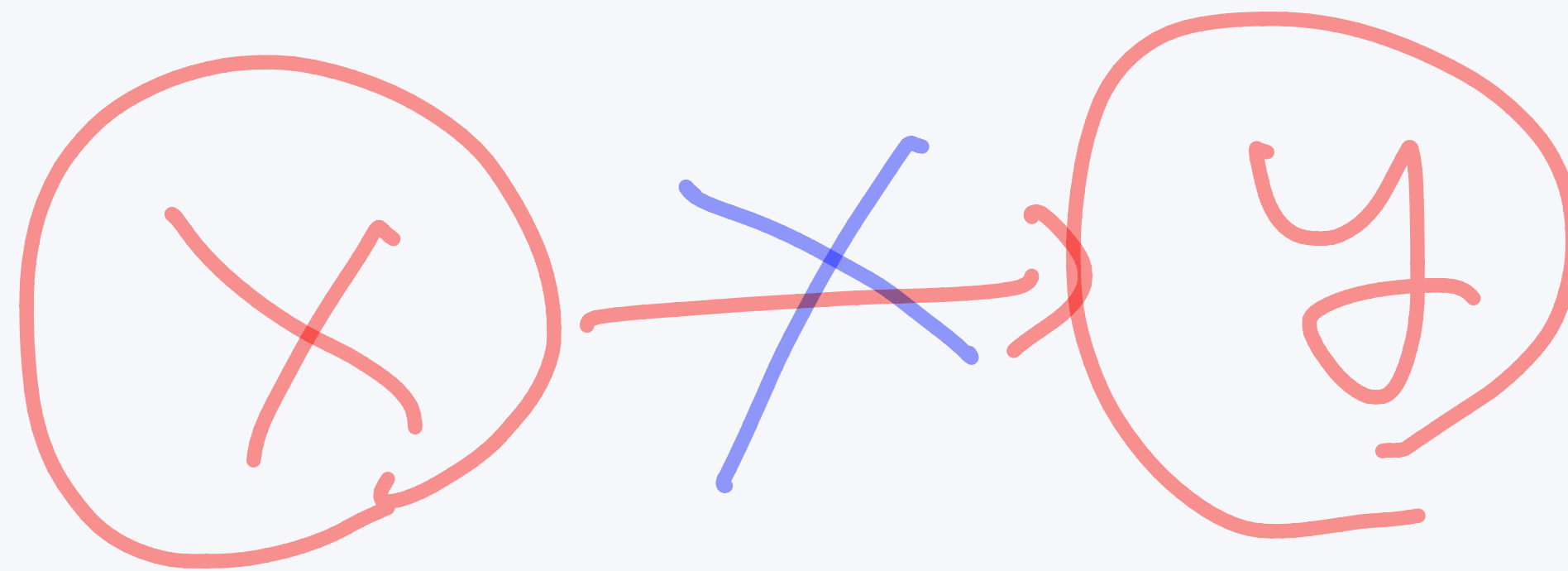
```
queue<int> q;  
for (int i=1; i<=n; i++) {  
    if (ind[i] == 0) {  
        q.push(i);  
    }  
}
```

# 위상 정렬

Topological Sort

```
for (int k=0; k<n; k++) {  
    int x = q.front();  
    q.pop();  
    printf("%d ", x);  
    for (int i=0; i<a[x].size(); i++) {  
        int y = a[x][i];  
        ind[y] -= 1;  
        if (ind[y] == 0) {  
            q.push(y);  
        }  
    }  
}
```

DFS



BFS

```
if (check[y] == false) {  
    check[y] = true;  
    q.push(y);  
}
```

# 줄 세우기

<https://www.acmicpc.net/problem/2252>

- N명의 학생들을 키 순서대로 줄을 세우려고 한다
- 각 학생의 키를 직접 재서 정렬하면 간단하겠지만, 마땅한 방법이 없어서 두 학생의 키를 비교하는 방법을 사용하기로 하였다
- 그나마도 모든 학생들을 다 비교해 본 것이 아니고, 일부 학생들의 키만을 비교해 보았다.
- 일부 학생들의 키를 비교한 결과가 주어졌을 때, 줄을 세우는 프로그램을 작성하시오.

# 줄 세우기

<https://www.acmicpc.net/problem/2252>

- C++: <https://gist.github.com/Baekjoon/17c688e9cc98568989c7>
- Java: <https://gist.github.com/Baekjoon/f62c5a1d4324f28e0c14>

# 문제집

<https://www.acmicpc.net/problem/1766>

- 위상 정렬에서 큐 대신에 우선 순위 큐를 사용해야 한다

# 문제집

<https://www.acmicpc.net/problem/1766>

- C++: <https://gist.github.com/Baekjoon/c6e346baef8b6d4b011c>
- Java: <https://gist.github.com/Baekjoon/5b5467aea67b63f4ec58>

# 작업

<https://www.acmicpc.net/problem/2056>

- 작업의 선행관계가 주어졌을 때, 모두 마치는 가장 빠른 시간
- 위상 정렬을 응용해서 풀 수 있다

DAG

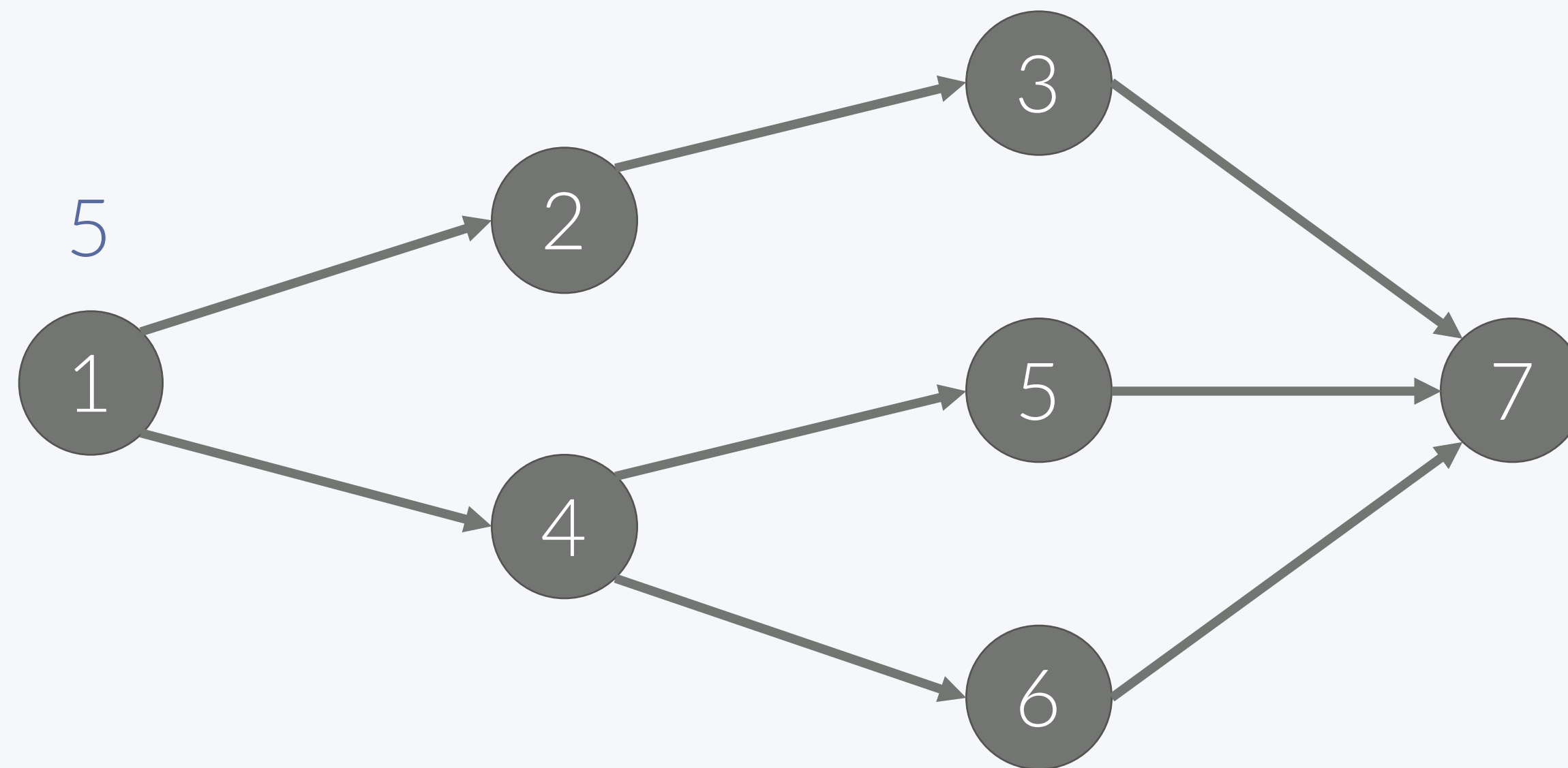
# 작업

32

<https://www.acmicpc.net/problem/2056>

- 큐: 1
- 현재 작업:

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4



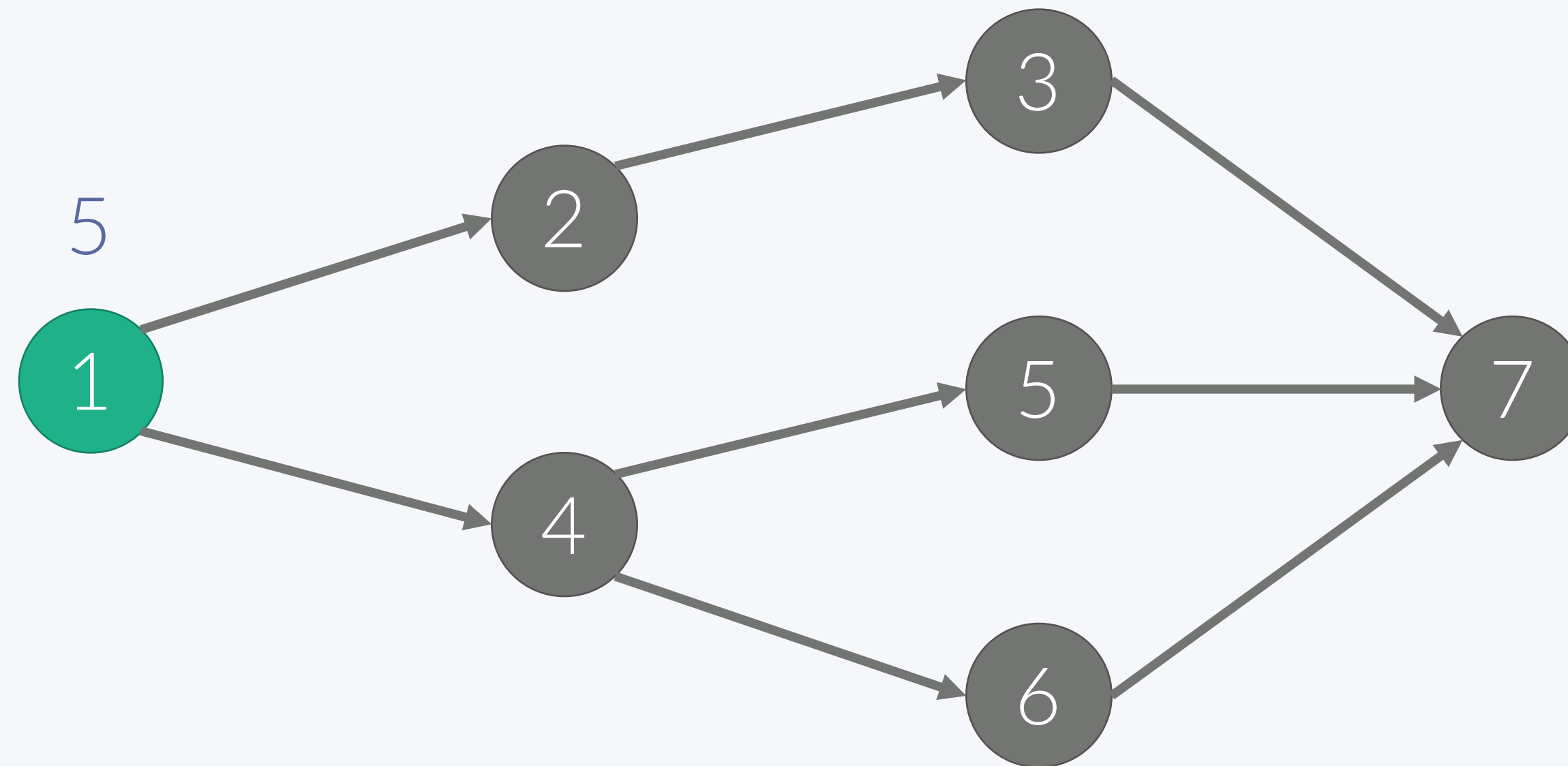


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐:
- 현재 작업: 1

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

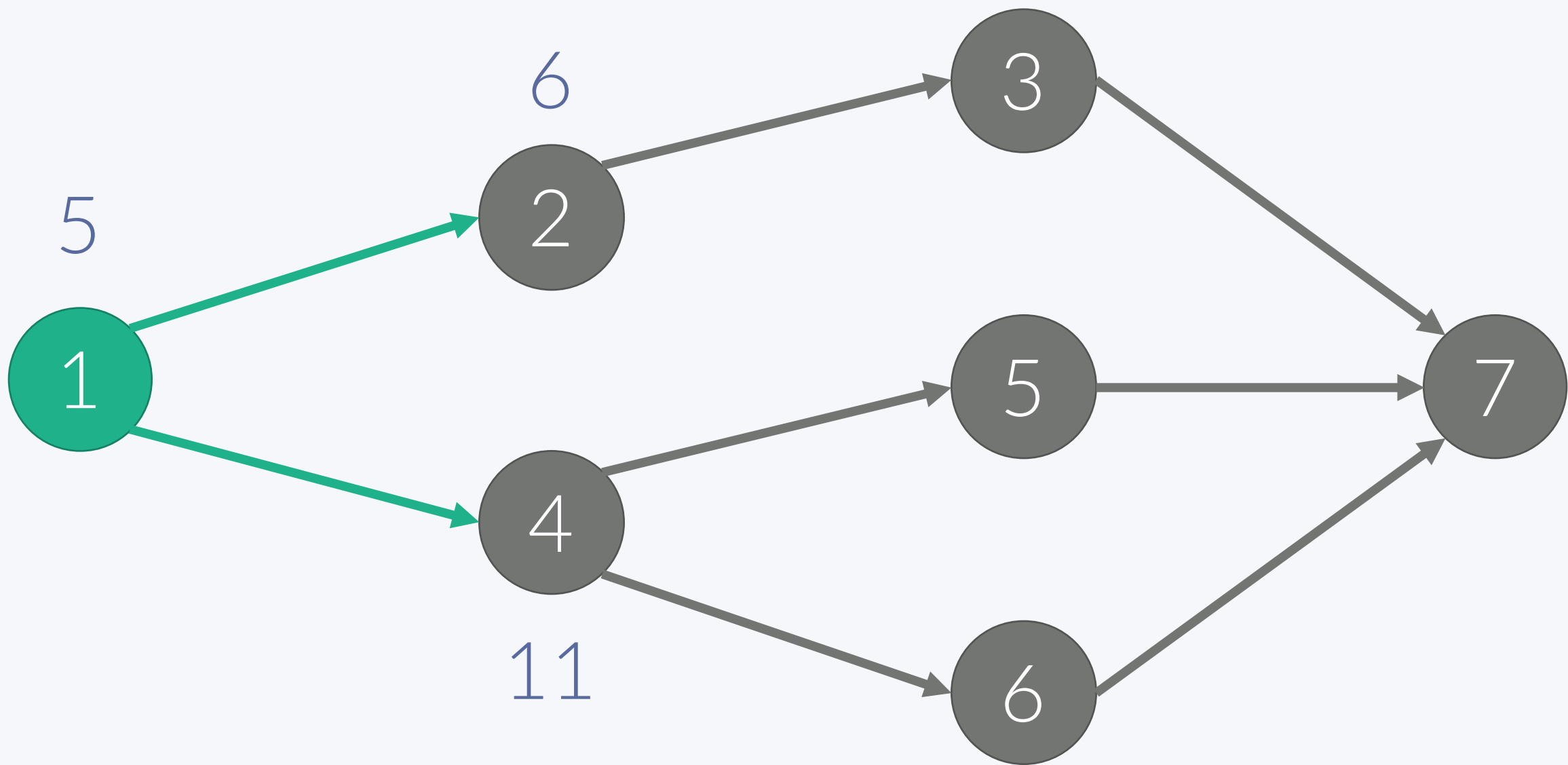


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐: 2 4
- 현재 작업: 1

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

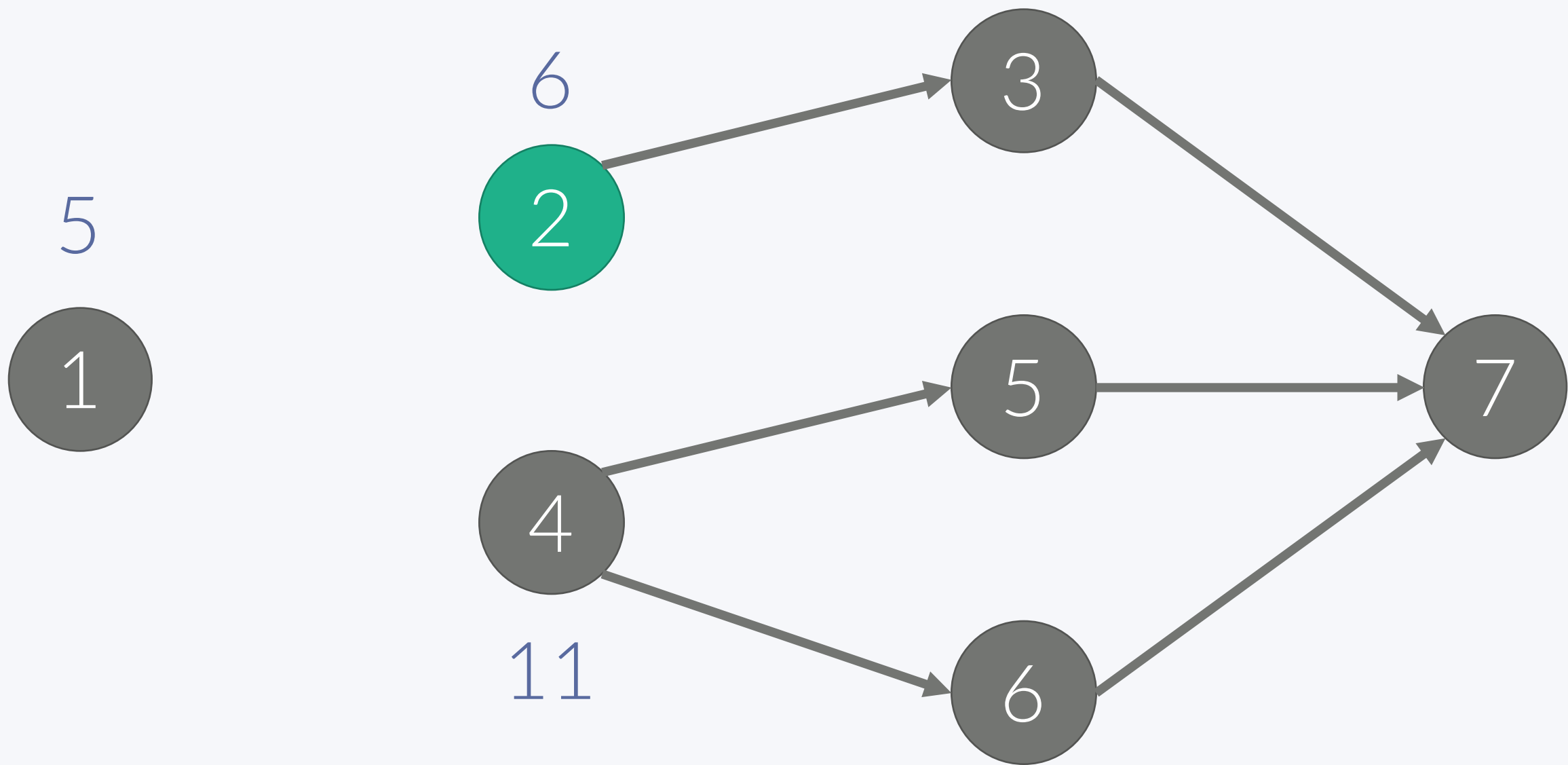


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐: 4
- 현재 작업: 2

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4



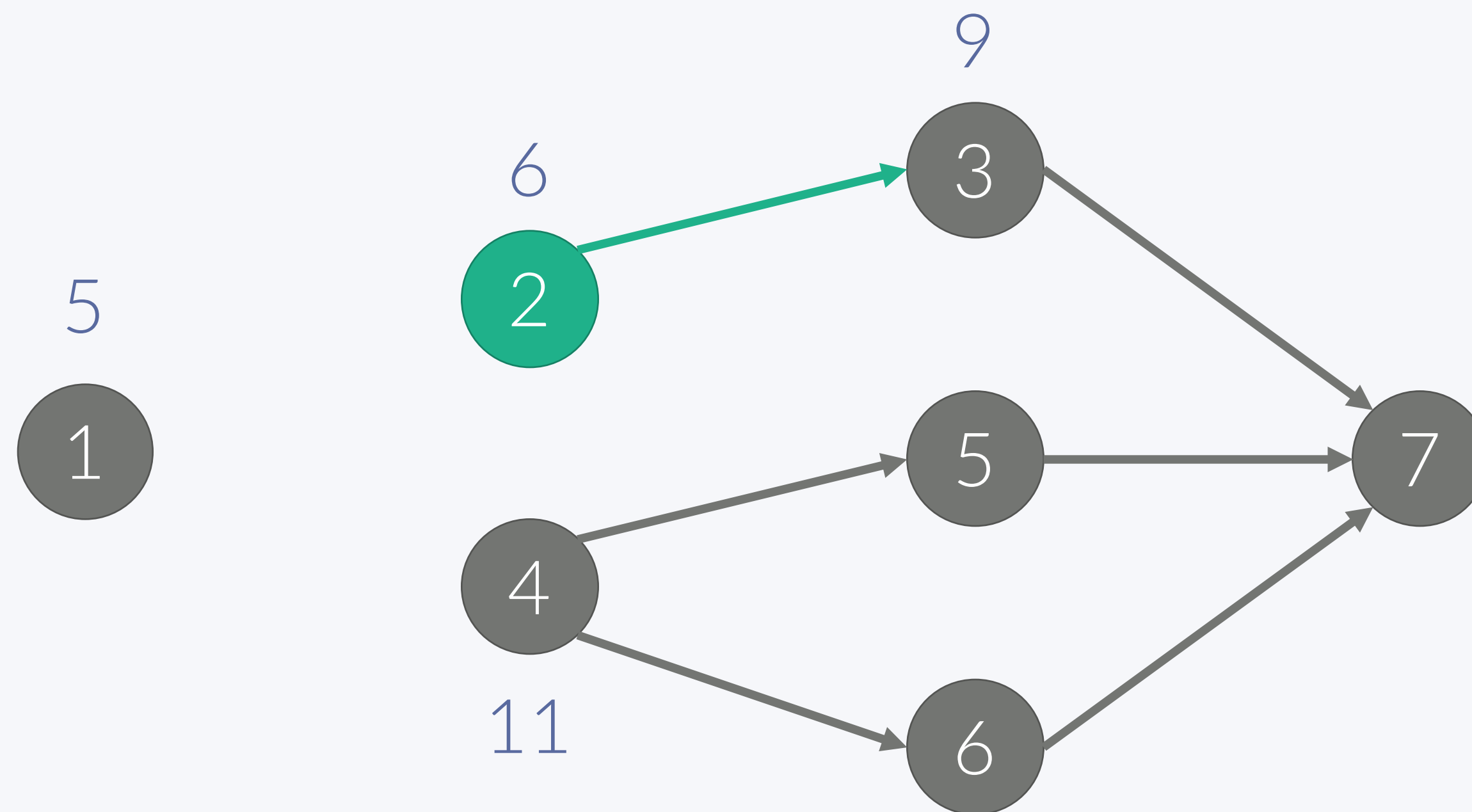
# 작업

36

<https://www.acmicpc.net/problem/2056>

- 큐: 4
- 현재 작업: 2

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

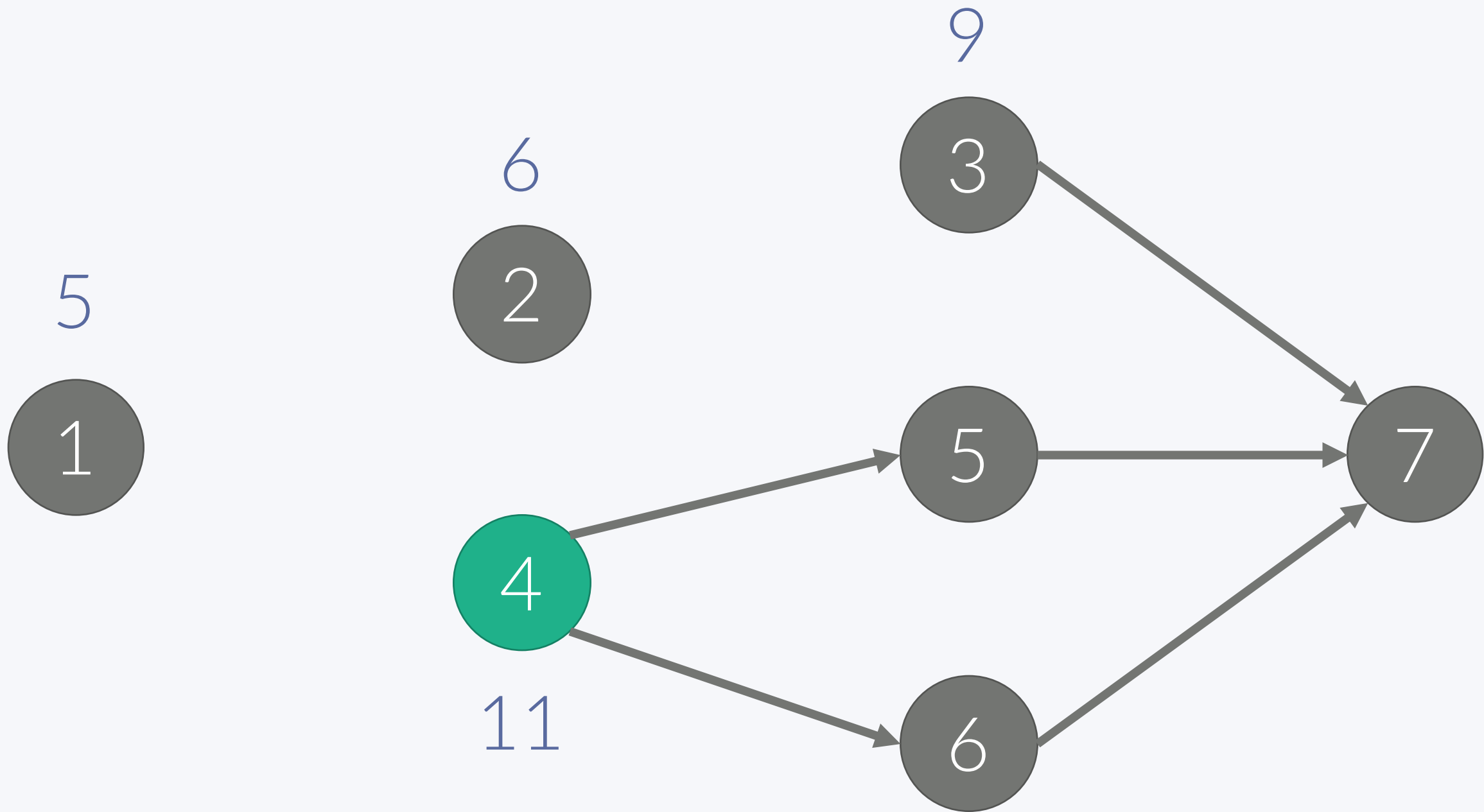


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐: 3
- 현재 작업: 4

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

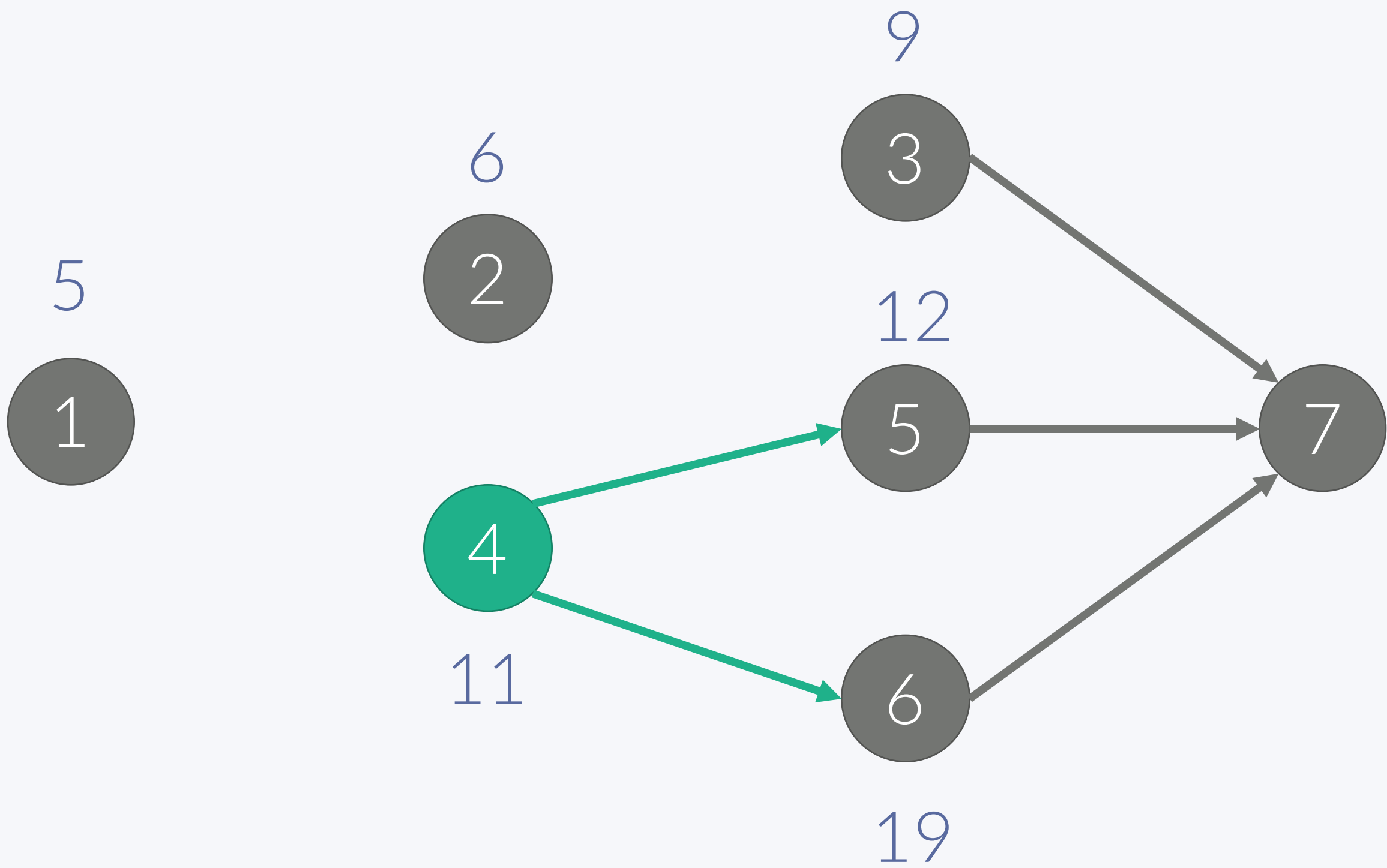


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐: 3
- 현재 작업: 4

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4



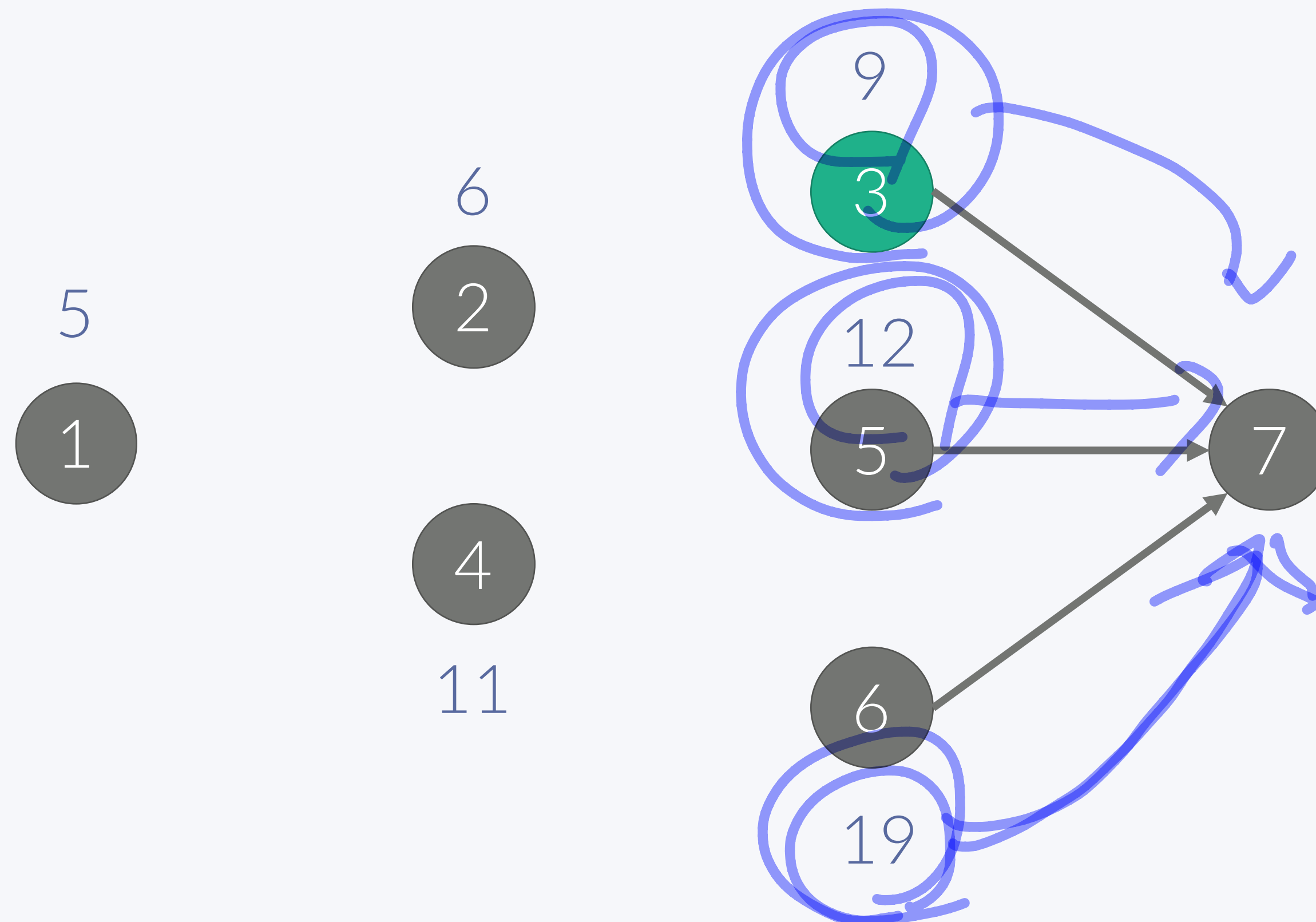
# 작업

39

<https://www.acmicpc.net/problem/2056>

- 큐: 5 6
- 현재 작업: 3

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

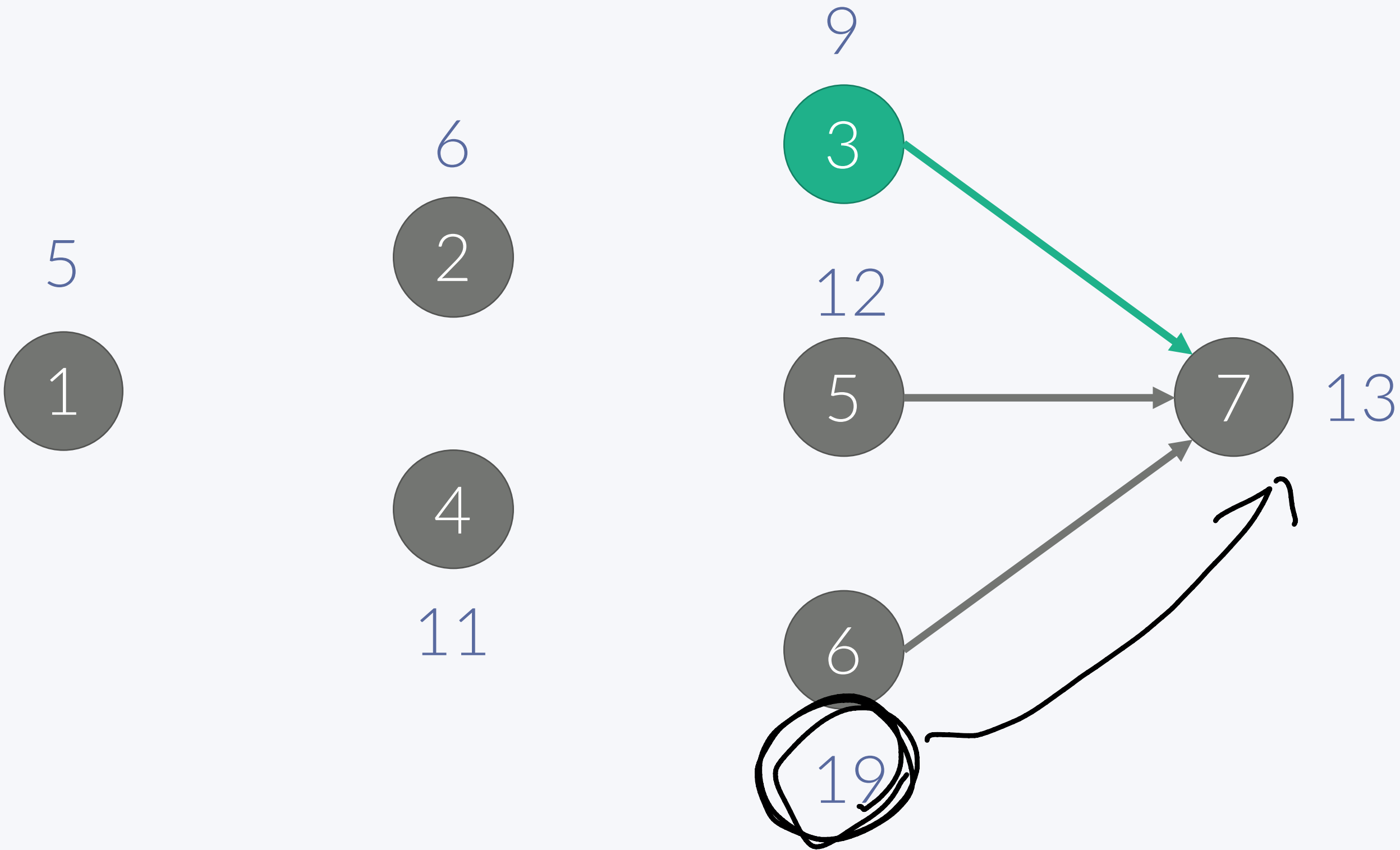


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐: 5 6
- 현재 작업: 3

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4



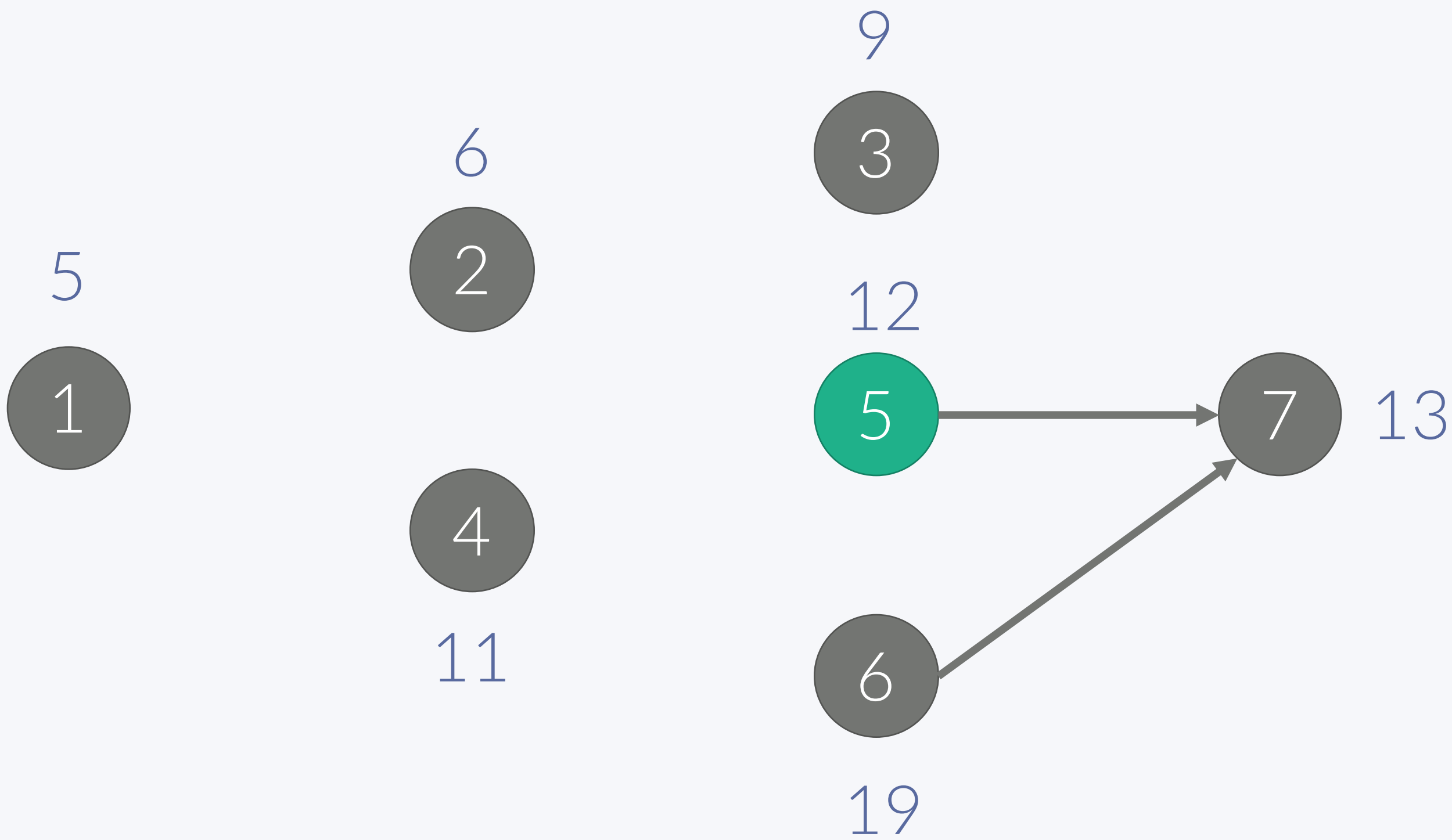


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐: 6
- 현재 작업: 5

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

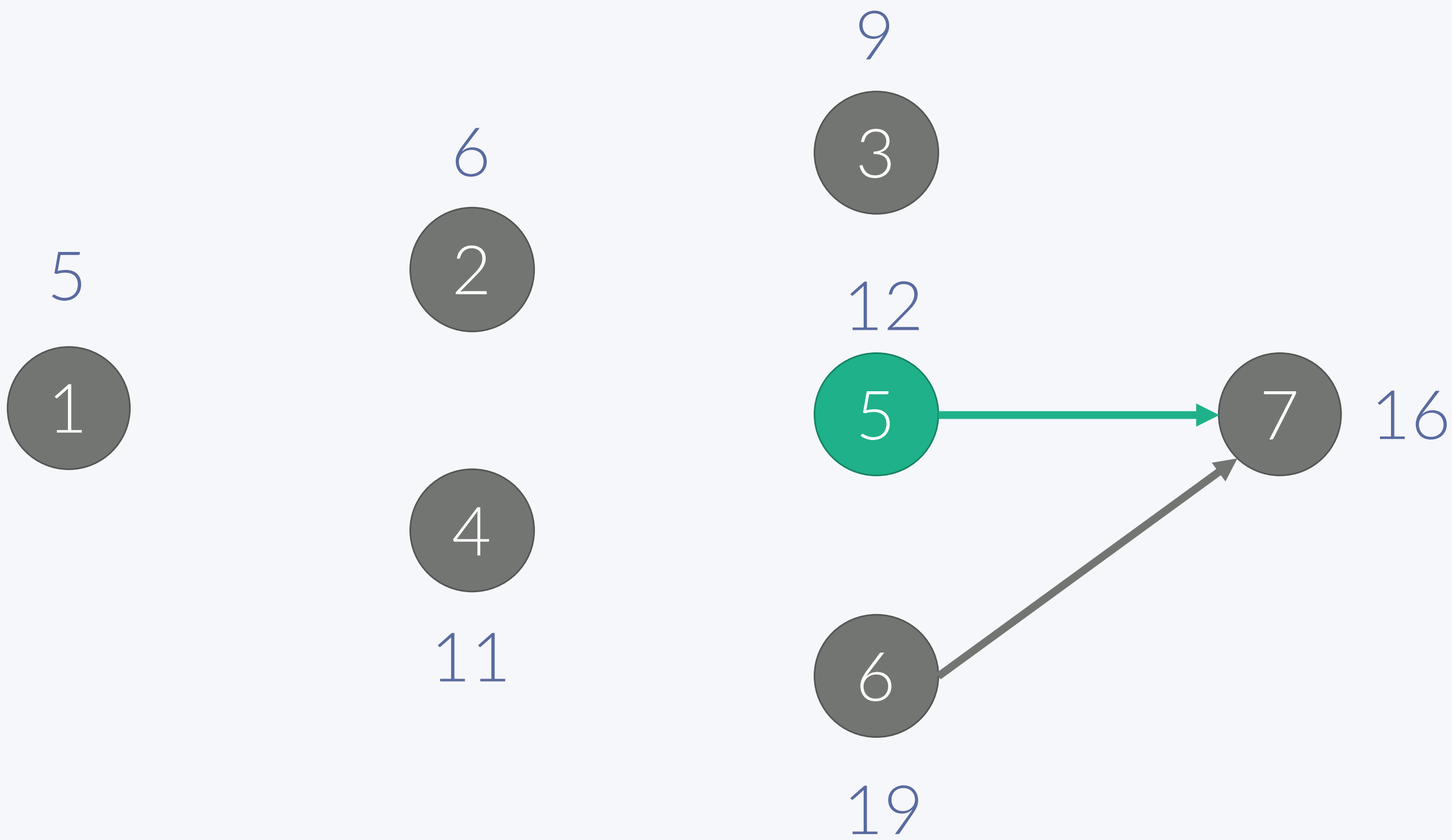


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐: 6
- 현재 작업: 5

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

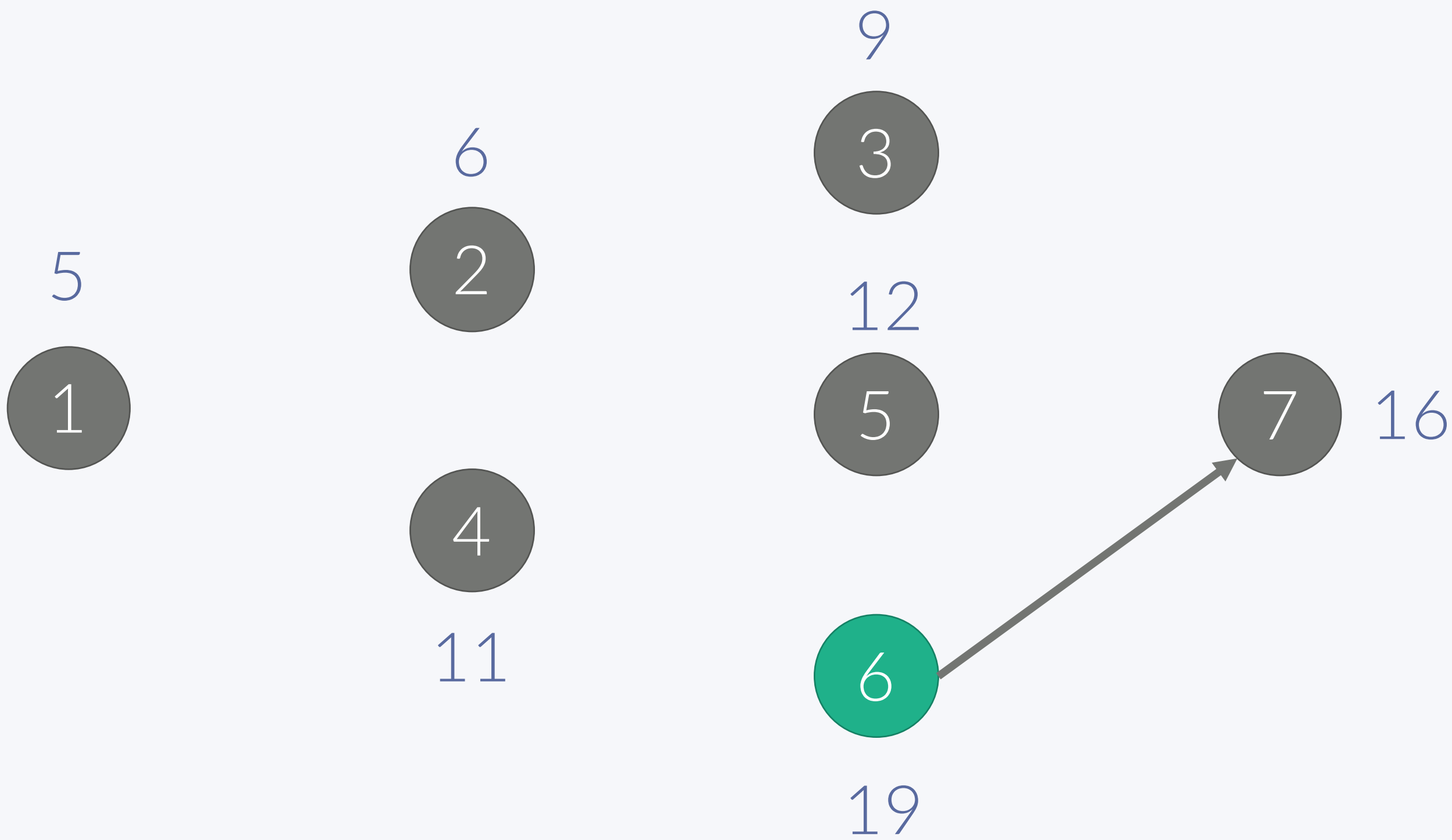


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐:
- 현재 작업: 6

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

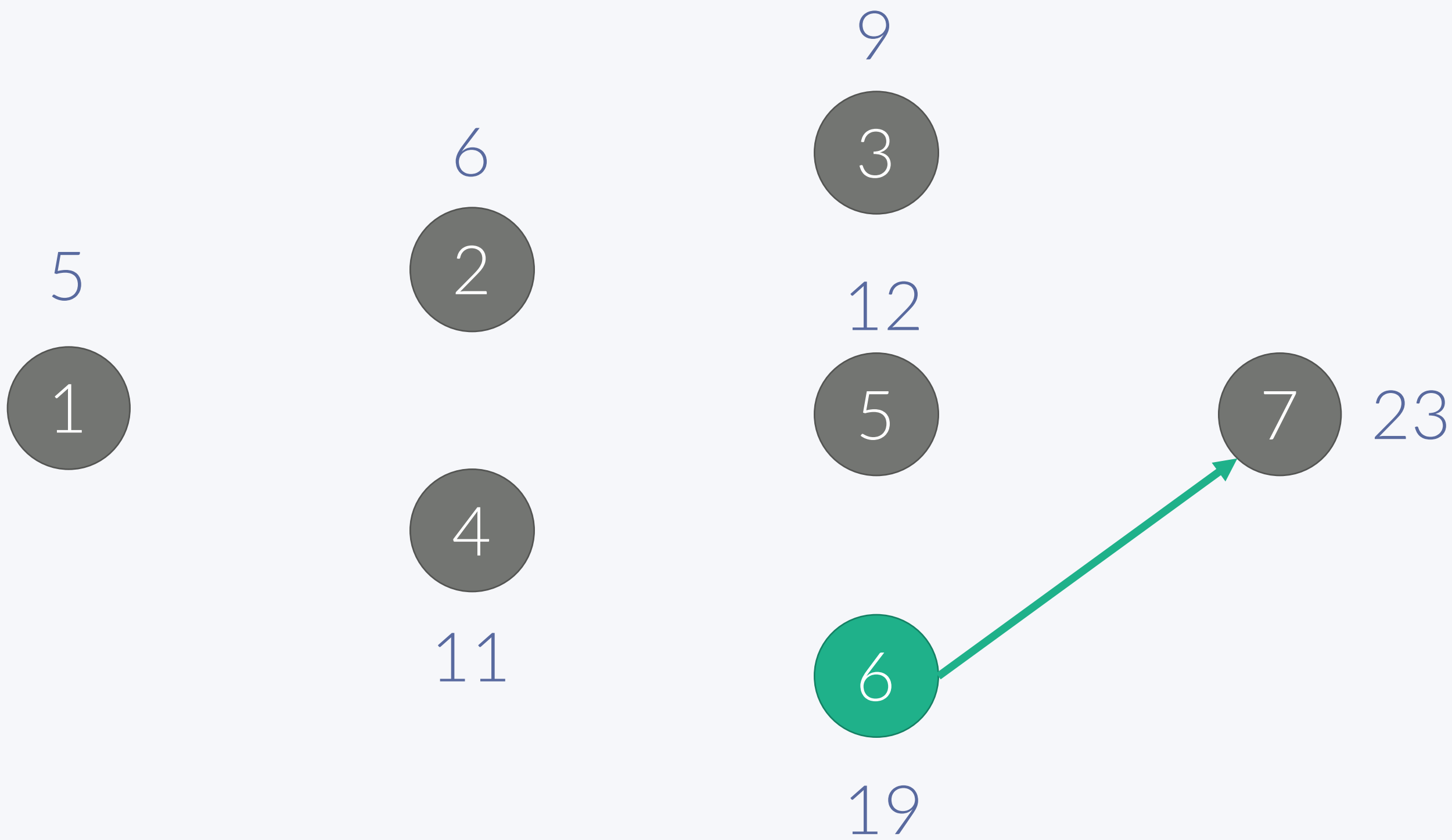


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐:
- 현재 작업: 6

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

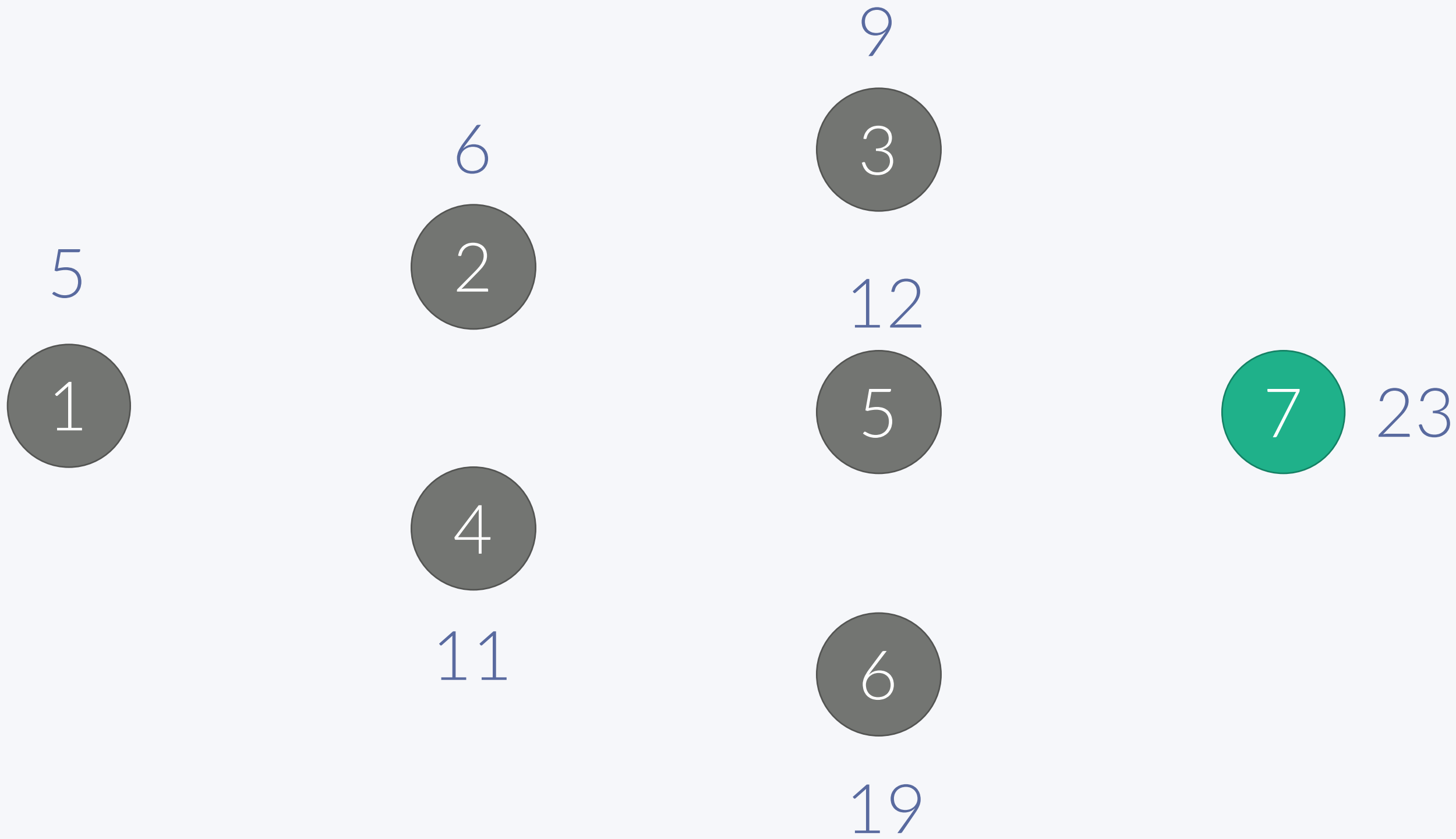


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐:
- 현재 작업: 7

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4

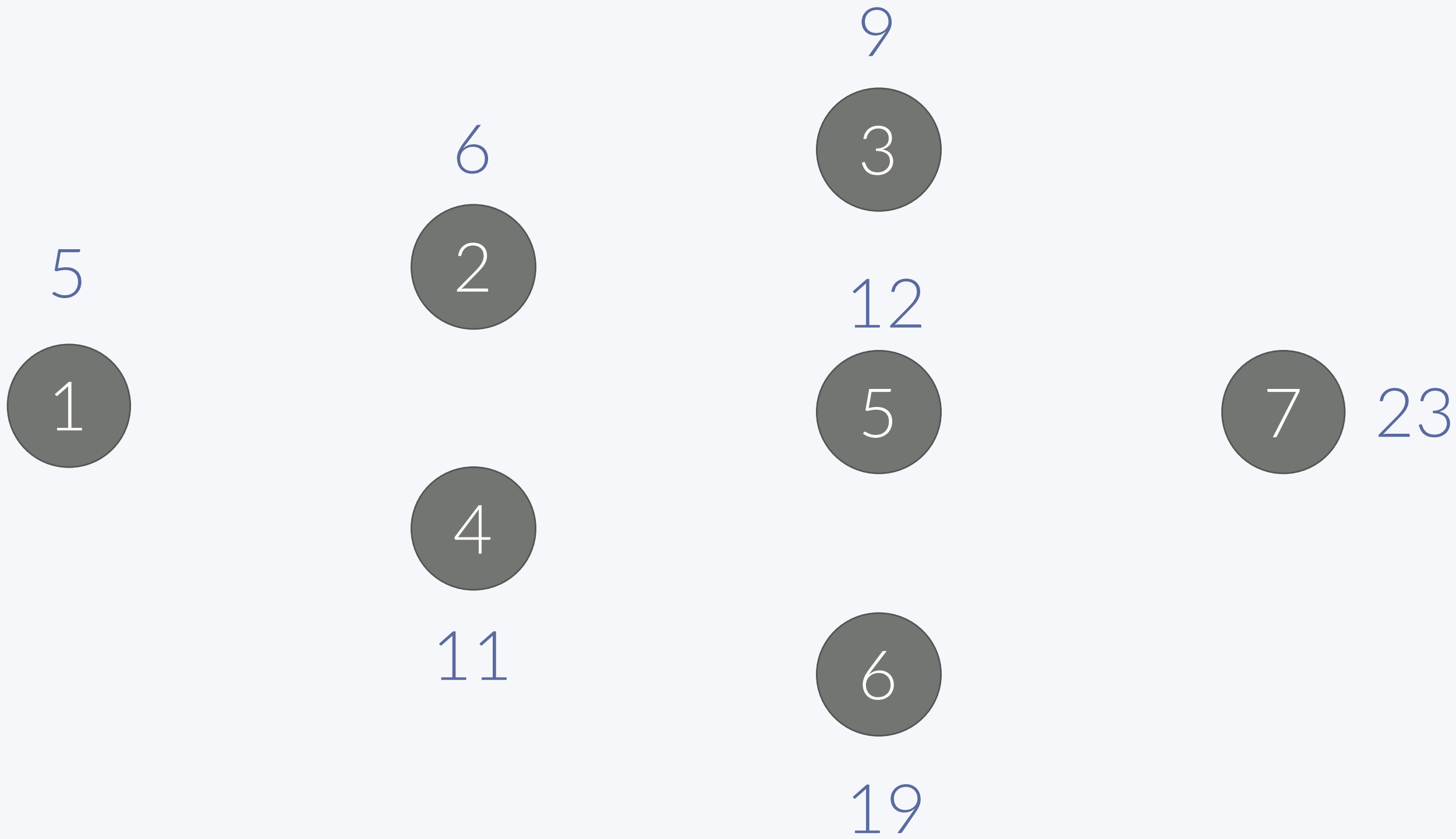


# 작업

<https://www.acmicpc.net/problem/2056>

- 큐:
- 현재 작업:

i	1	2	3	4	5	6	7
시간	5	1	3	6	1	8	4



# 작업

<https://www.acmicpc.net/problem/2056>

- C++: <https://gist.github.com/Baekjoon/fc6e0f974a4d65309e85>
- Java: <https://gist.github.com/Baekjoon/13769e8e0f06fa9ae383>

# 게임 개발

48

<https://www.acmicpc.net/problem/1516>

- N개의 각 건물이 완성되기까지 걸리는 최소 시간 구하기



# MST

---

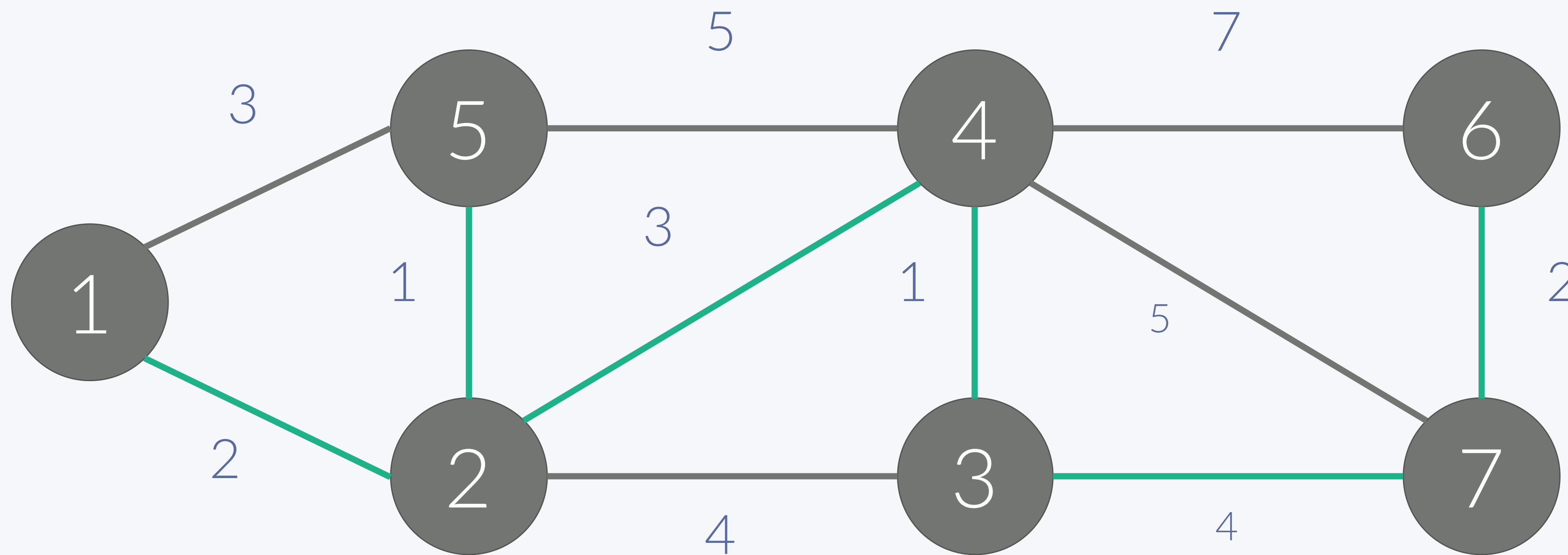
# 최소 스패닝 트리

크리히스트

50

Minimum Spanning Tree

- 스패닝 트리: 그래프에서 일부 간선을 선택해서 만든 트리
- 최소 스패닝 트리: 스패닝 트리 중에 선택한 간선의 가중치의 합이 최소인 트리



# 프림

---

# 프림

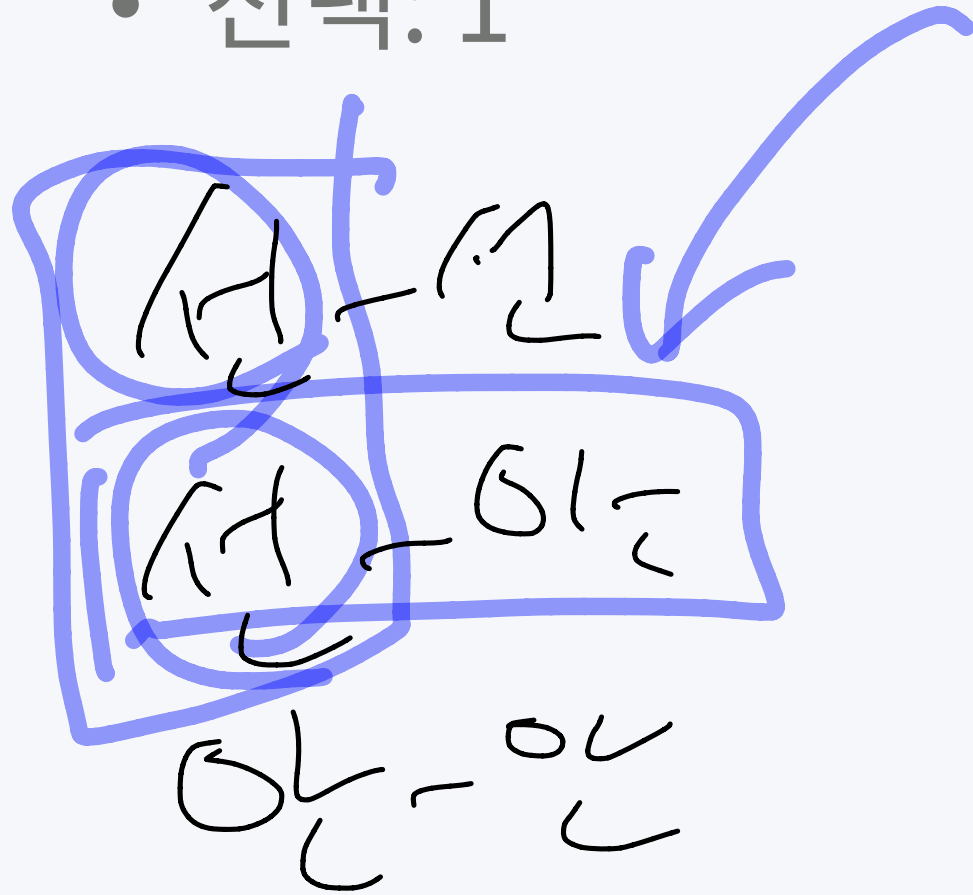
## Prim

1. 그래프에서 아무 정점이나 선택한다.
2. 선택한 정점과 선택하지 않은 정점을 연결하는 간선중에 최소값을 고른다. 이 간선을  $(u, v)$ 라고 한다. ( $u =$  선택,  $v =$  선택하지 않음)
3. 선택한 간선을 MST에 추가하고,  $v$ 를 선택한다.
4. 모든 정점선택하지 않았다면, 2번 단계로 돌아간다.

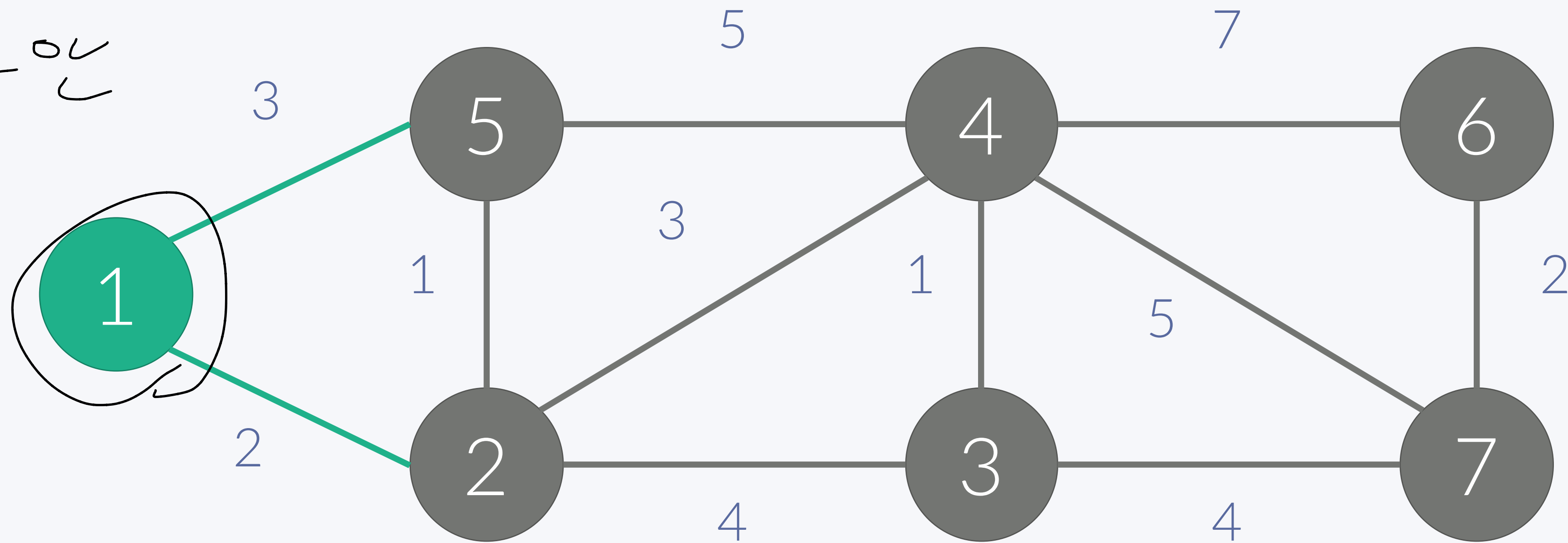
# 프림

Prim

- 선택: 1



Handwritten note in black ink:  $(2, 2)$  가 줄러



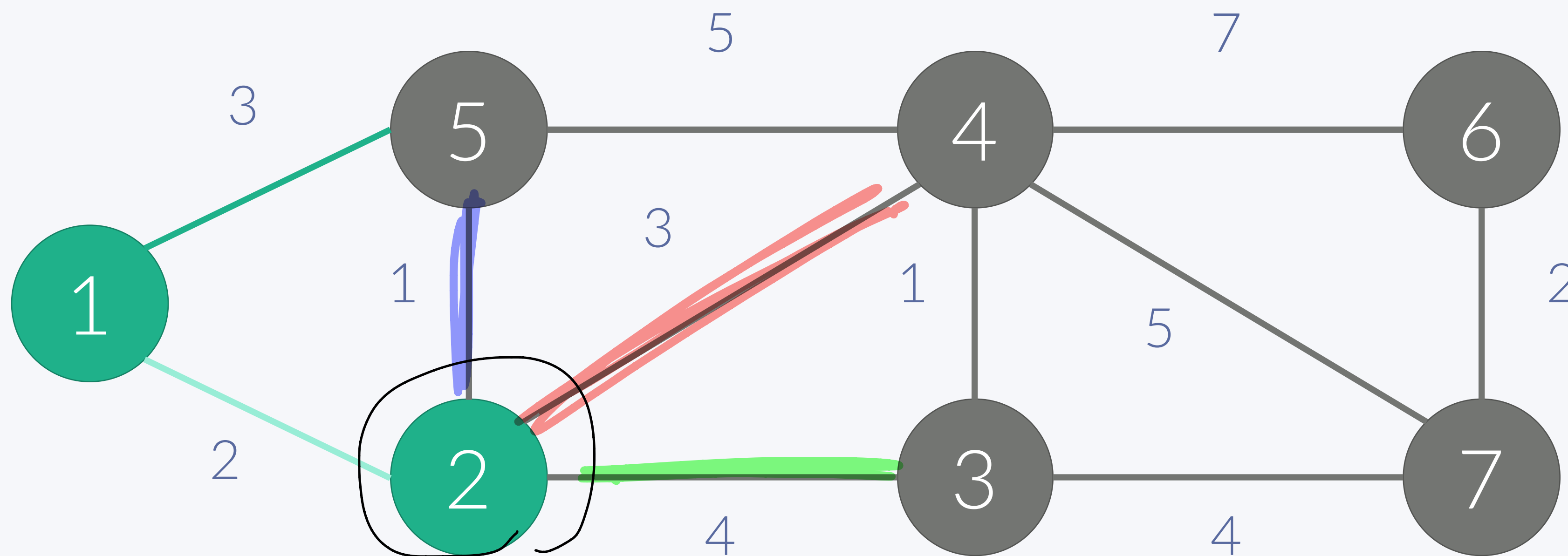
# 프림

Prim

- 선택: 1 2

$(2, 3)$ ,  $(2, 4)$

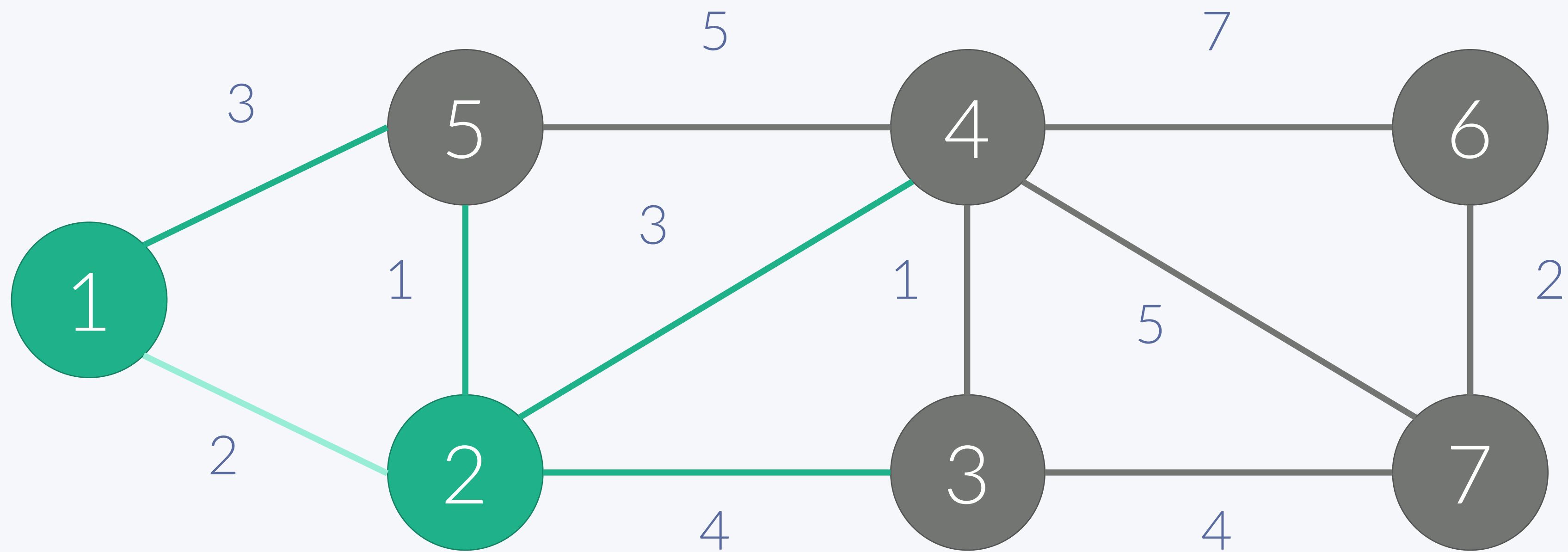
$(5, 3)$   
 $(5, 1)$   
 $(4, 3)$   
 $(3, 4)$



# 프림

Prim

- 선택: 1 2



# 프림

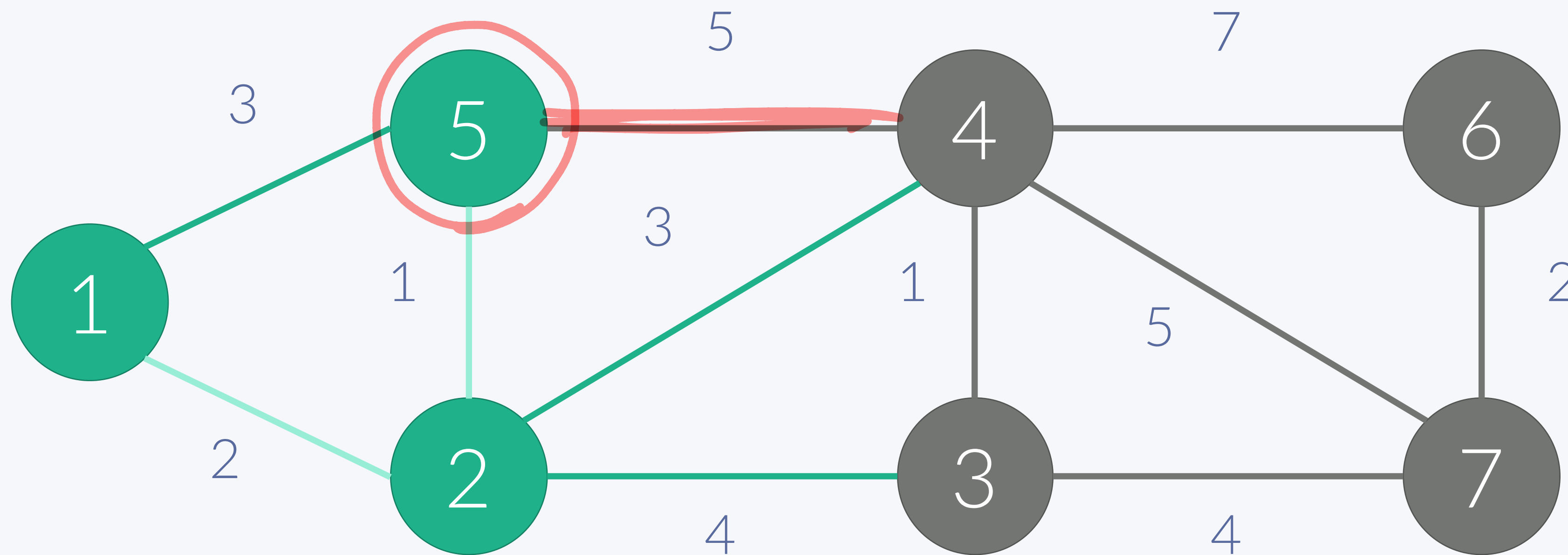
Prim

- 선택: 1 2 5

(정점, 가중치)

$O(N^2)$  (Edge)

~~(5, 3)~~  
(4, 3)  
(3, 4)  
(4, 5)

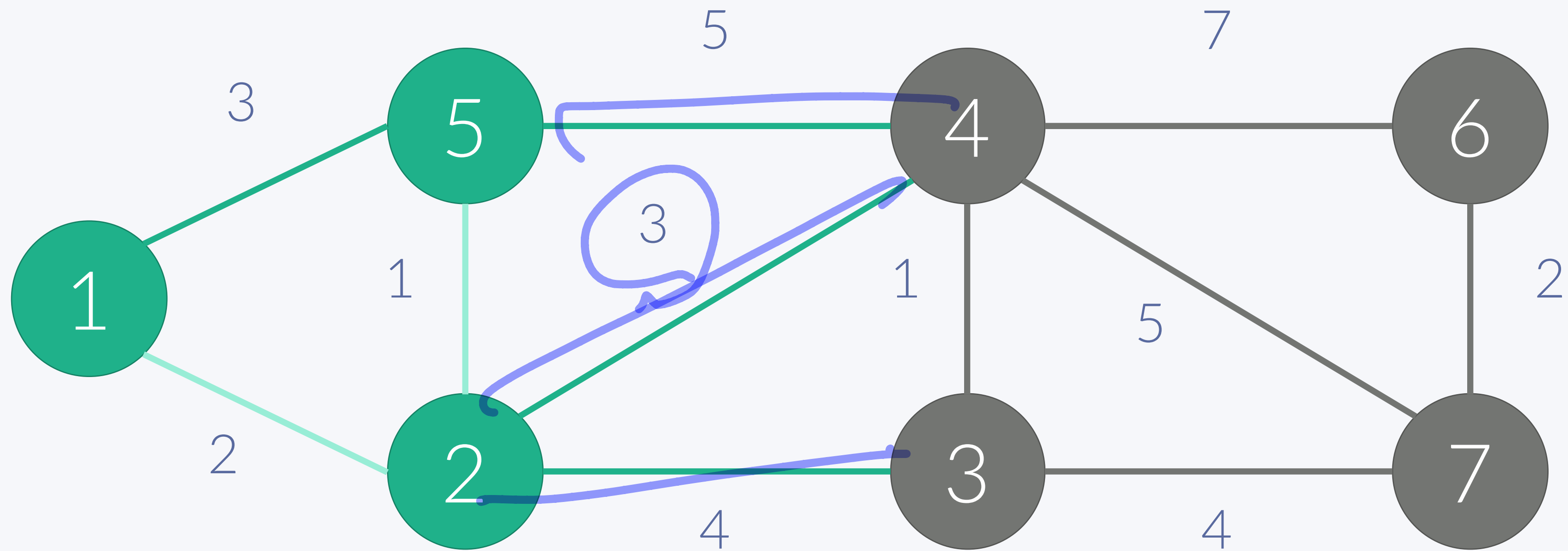




# 프림

Prim

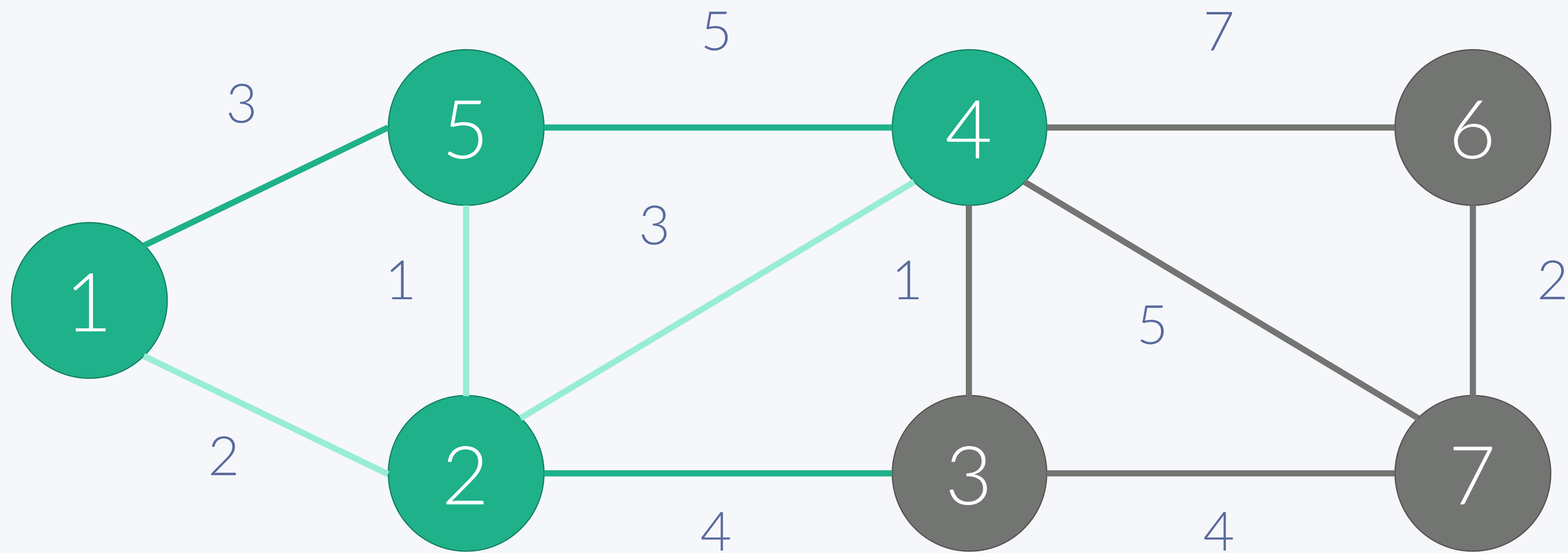
- 선택: 1 2 5



# 프림

Prim

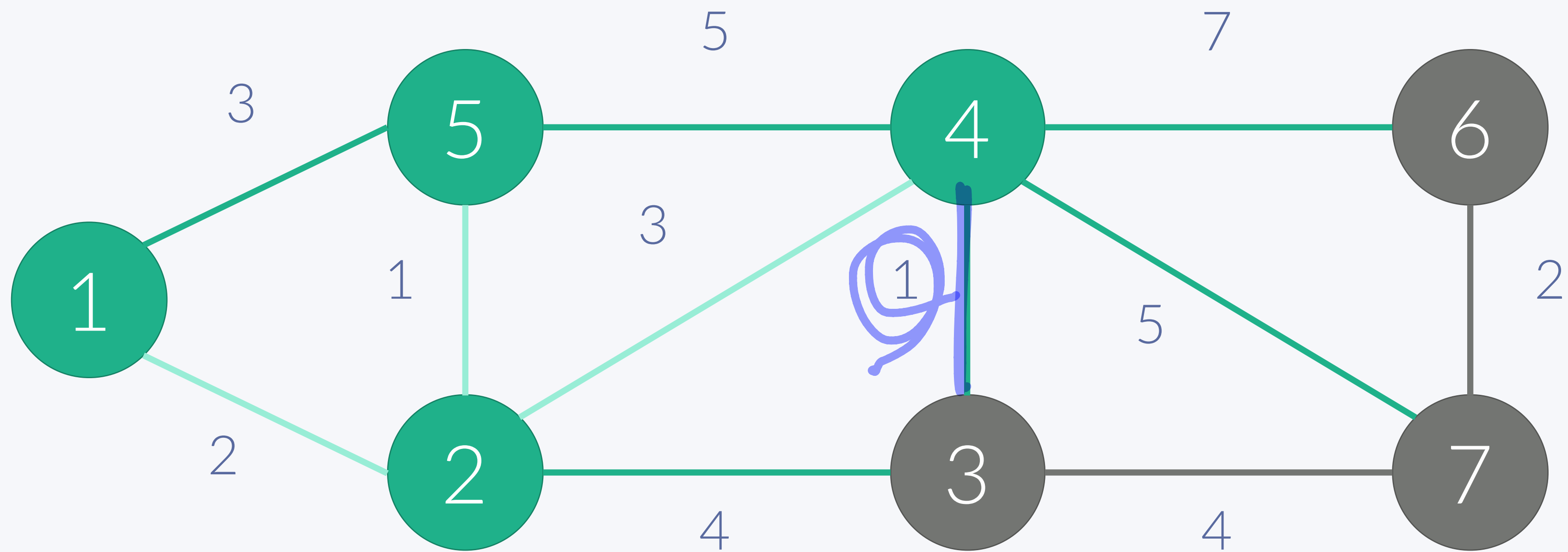
- 선택: 1 2 5 4



# 프림

Prim

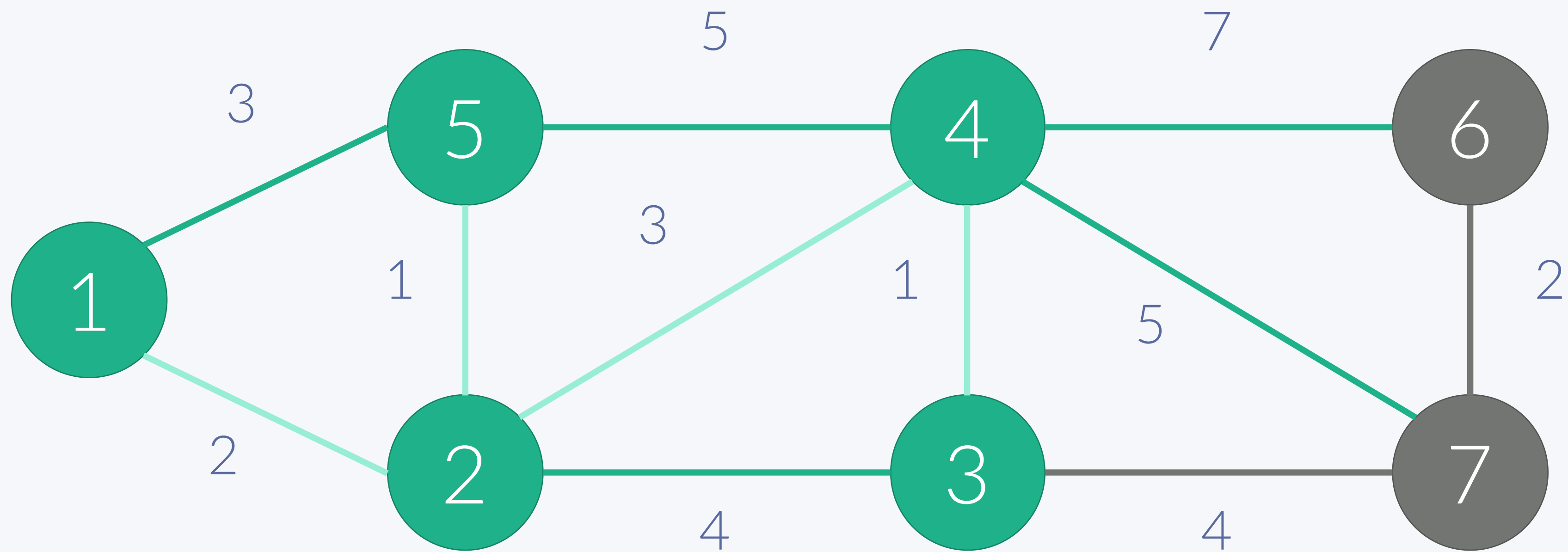
- 선택: 1 2 5 4



# 프림

Prim

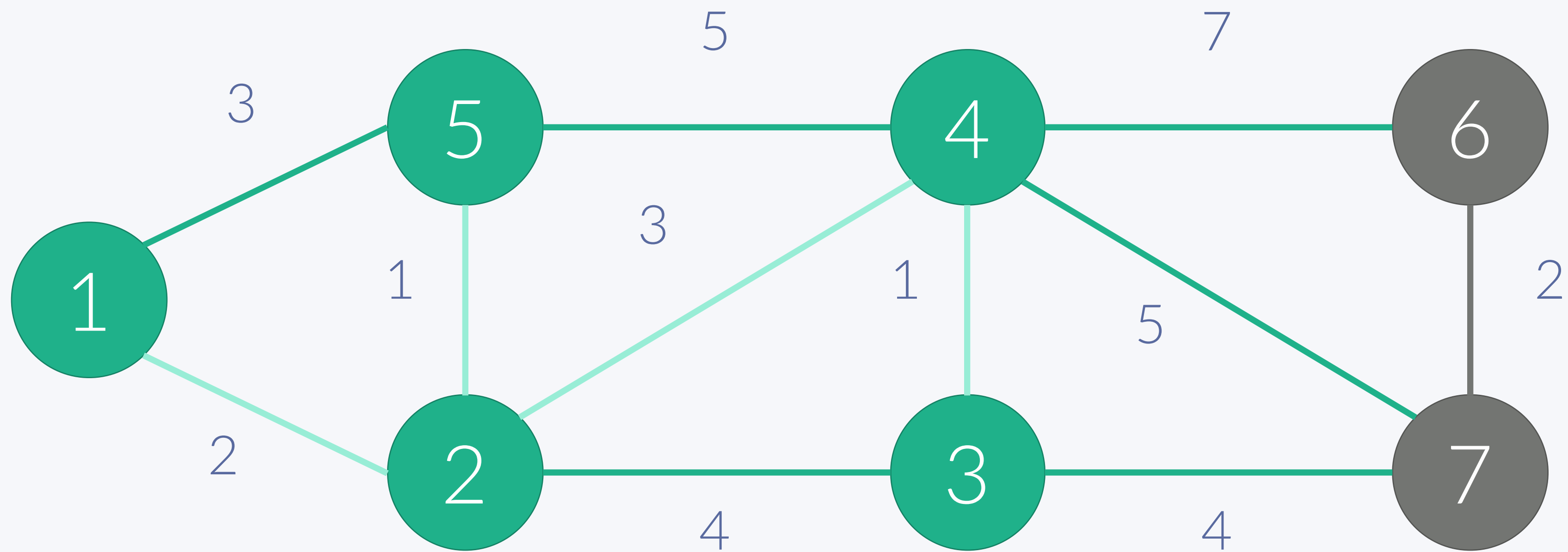
- 선택: 1 2 5 4 3



# 프림

Prim

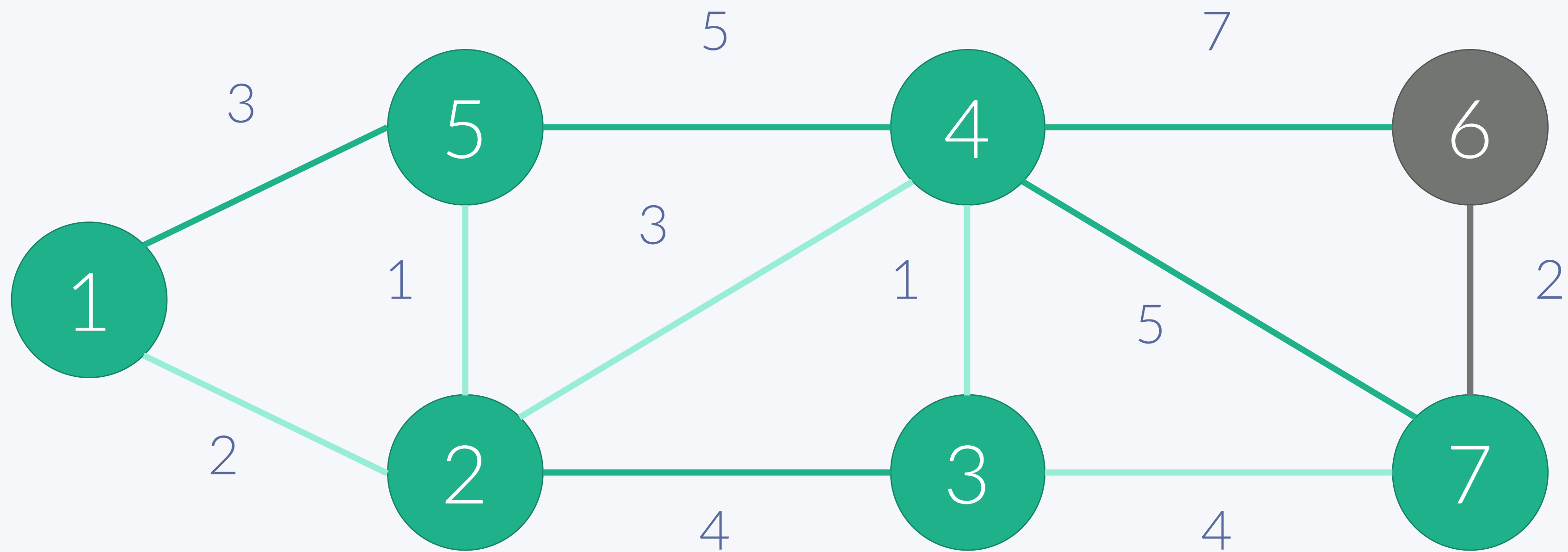
- 선택: 1 2 5 4 3



# 프림

Prim

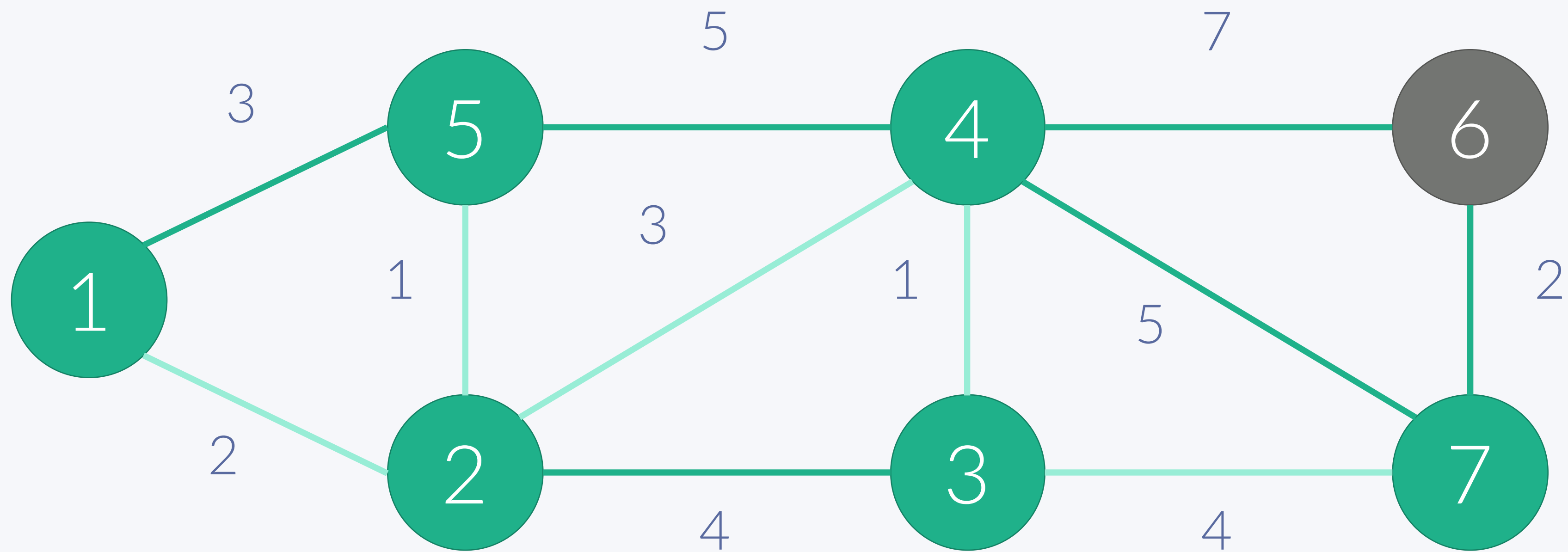
- 선택: 1 2 5 4 3 7



# 프림

Prim

- 선택: 1 2 5 4 3 7



# 프림

Prim

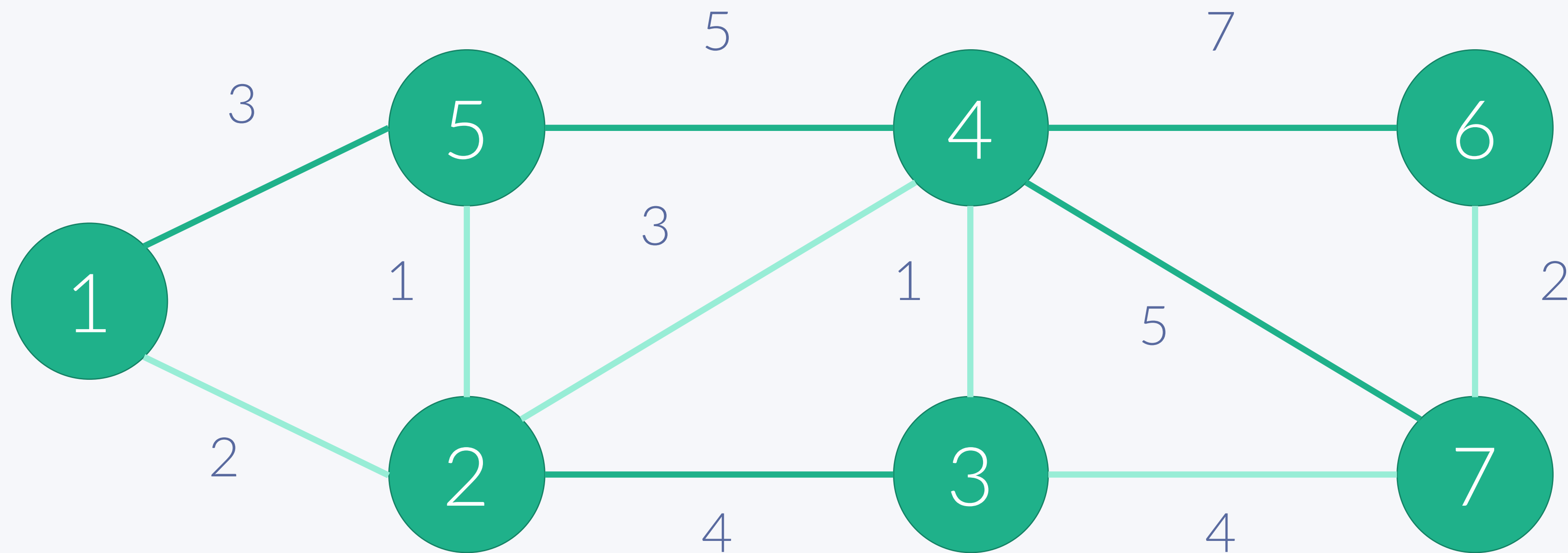
- 선택: 1 2 5 4 3 7 6

정점

가중치 E

64

$O(V^2)$





# 프림

## Prim

1. 그래프에서 아무 정점이나 선택한다.
  2. 선택한 정점과 선택하지 않은 정점을 연결하는 간선중에 최소값을 고른다. 이 간선을  $(u, v)$ 라고 한다. ( $u =$  선택,  $v =$  선택하지 않음)
  3. 선택한 간선을 MST에 추가하고,  $v$ 를 선택한다.
  4. 모든 정점선택하지 않았다면, 2번 단계로 돌아간다.
- 각각의 정점을 선택하고 모든 간선을 살펴봐야 한다.
  - 시간 복잡도:  $O(V \cdot E)$  최대  $O(V^3)$

# 프림

## Prim

1. 그래프에서 아무 정점이나 선택한다.
  2. 선택한 정점과 선택하지 않은 정점을 연결하는 간선중에 최소값을 고른다. 이 간선을  $(u, v)$ 라고 한다. ( $u =$  선택,  $v =$  선택하지 않음)
  3. 선택한 간선을 MST에 추가하고,  $v$ 를 선택한다.
  4. 모든 정점선택하지 않았다면, 2번 단계로 돌아간다.
- 최소값을 우선 순위 큐를 이용하면 최소값을  $\lg E$ 만에 찾을 수 있다
  - 시간 복잡도:  $O(E \lg E)$

# 네트워크 연결

Prim

- 그래프가 주어졌을 때, 그 그래프의 최소 스패닝 트리를 구하기

# 네트워크 연결

<https://www.acmicpc.net/problem/1922>

- C/C++: <https://gist.github.com/Baekjoon/1e2e79615fe4652e2944>
- C/C++: <https://gist.github.com/Baekjoon/each38feb11c3792ebbd>
- Java: <https://gist.github.com/Baekjoon/45634a257ef865fc6a7b>

$\frac{52}{2}$   
 $\frac{224}{2}$  222.

# 크루스칼

# 크루스칼

Kruskal

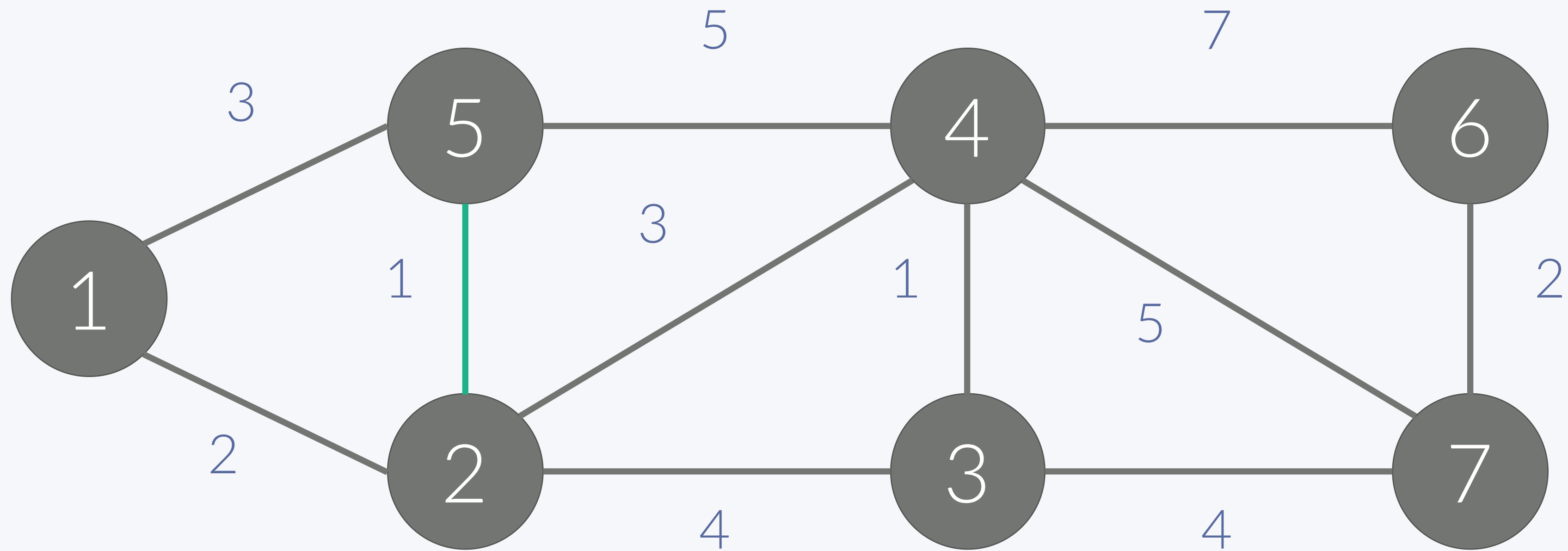
70

- 가중치가 작은 Edge부터 순서대로 살펴본다.
- Edge  $e$ 가  $(u, v, c)$  일 때
- $u$ 와  $v$ 가 다른 집합이면  $e$ 를 MST에 추가한다

# 크루스칼

Kruskal

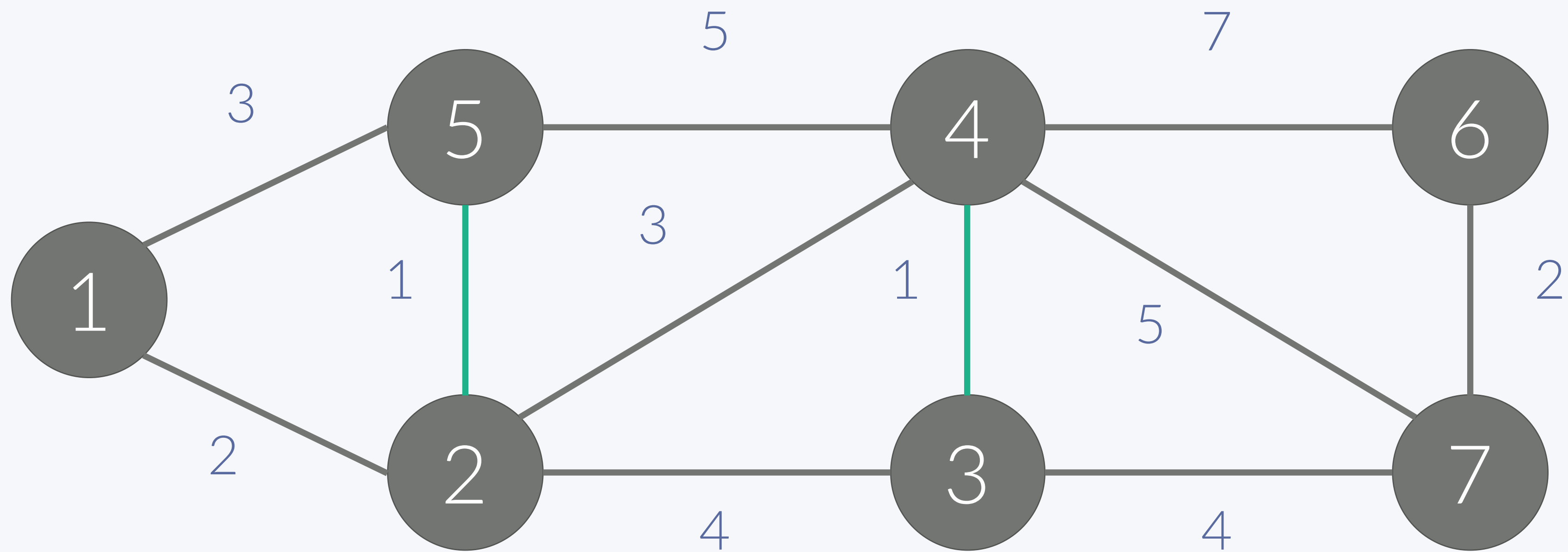
71



# 크루스칼

Kruskal

72

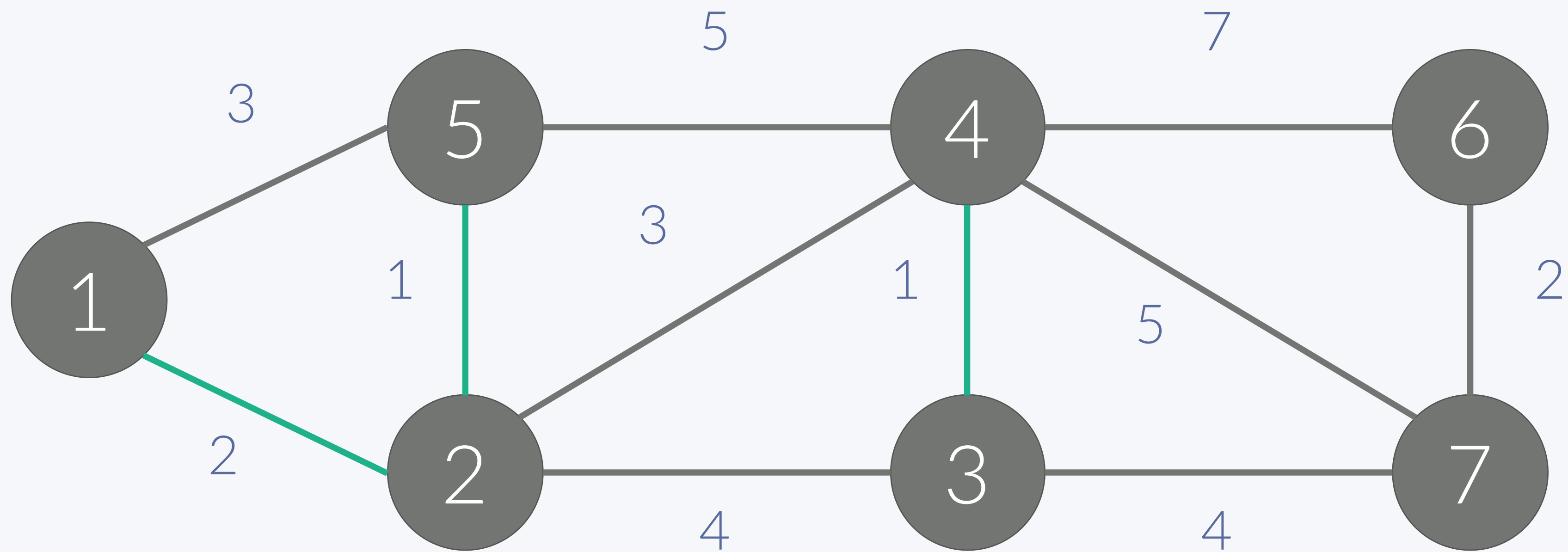




# 크루스칼

Kruskal

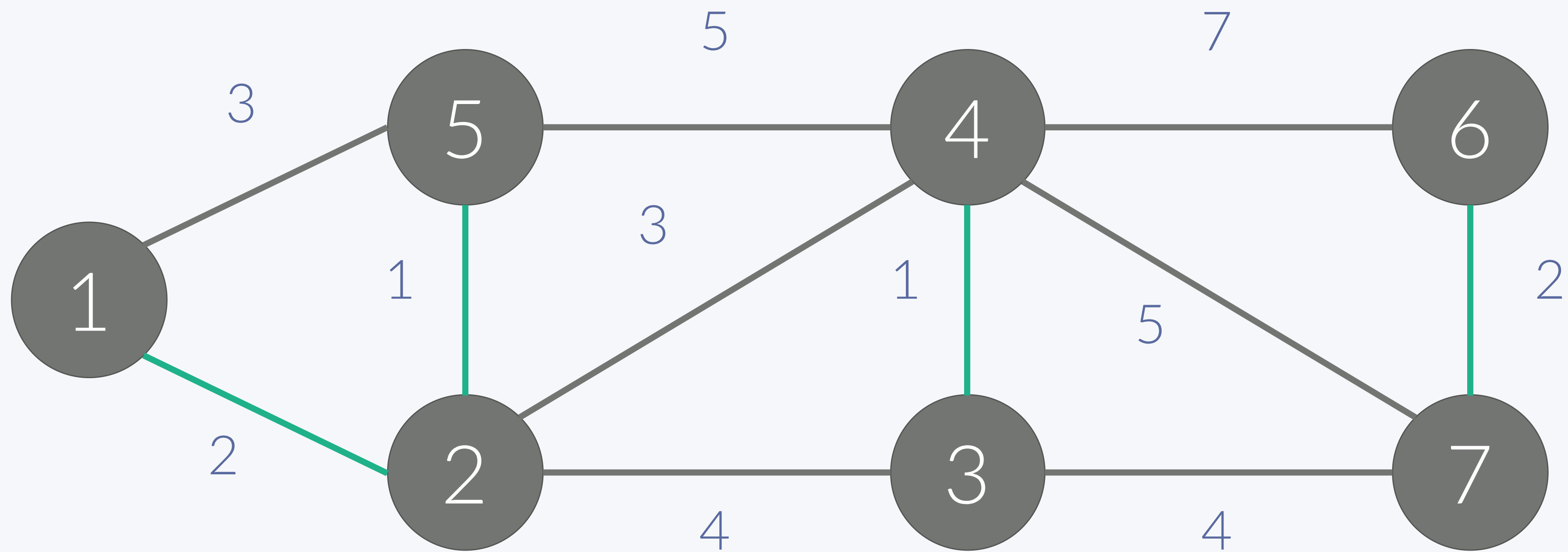
73



# 크루스칼

Kruskal

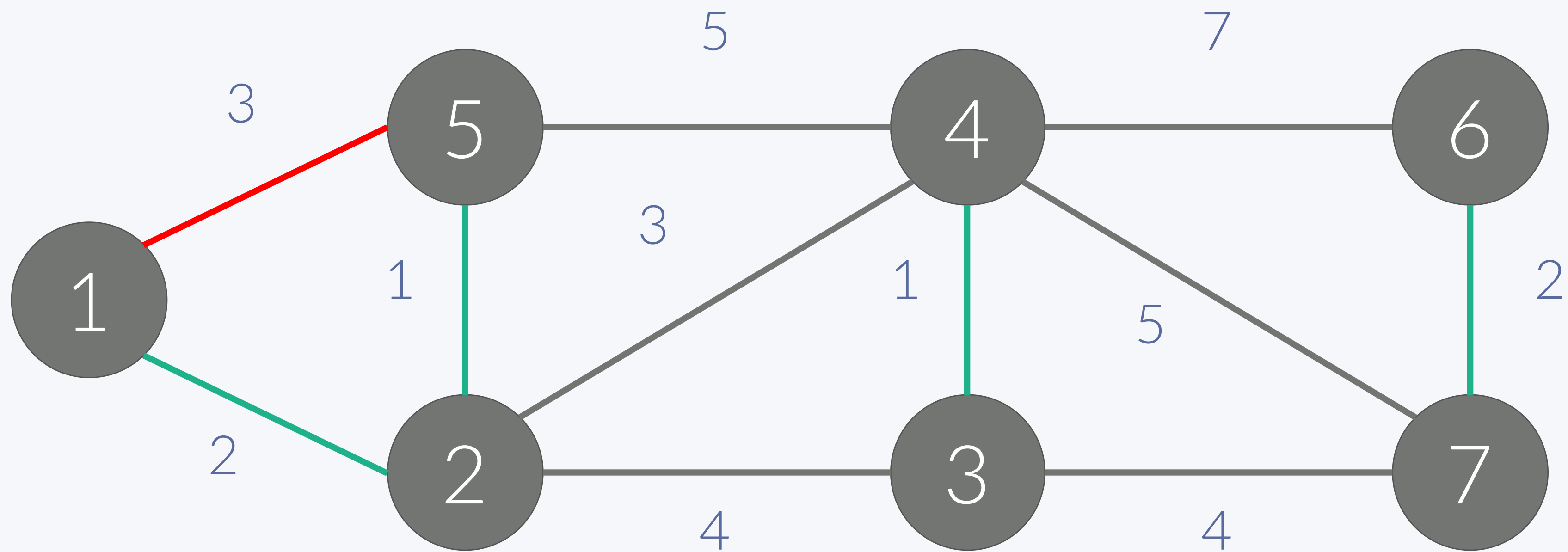
74



# 크루스칼

Kruskal

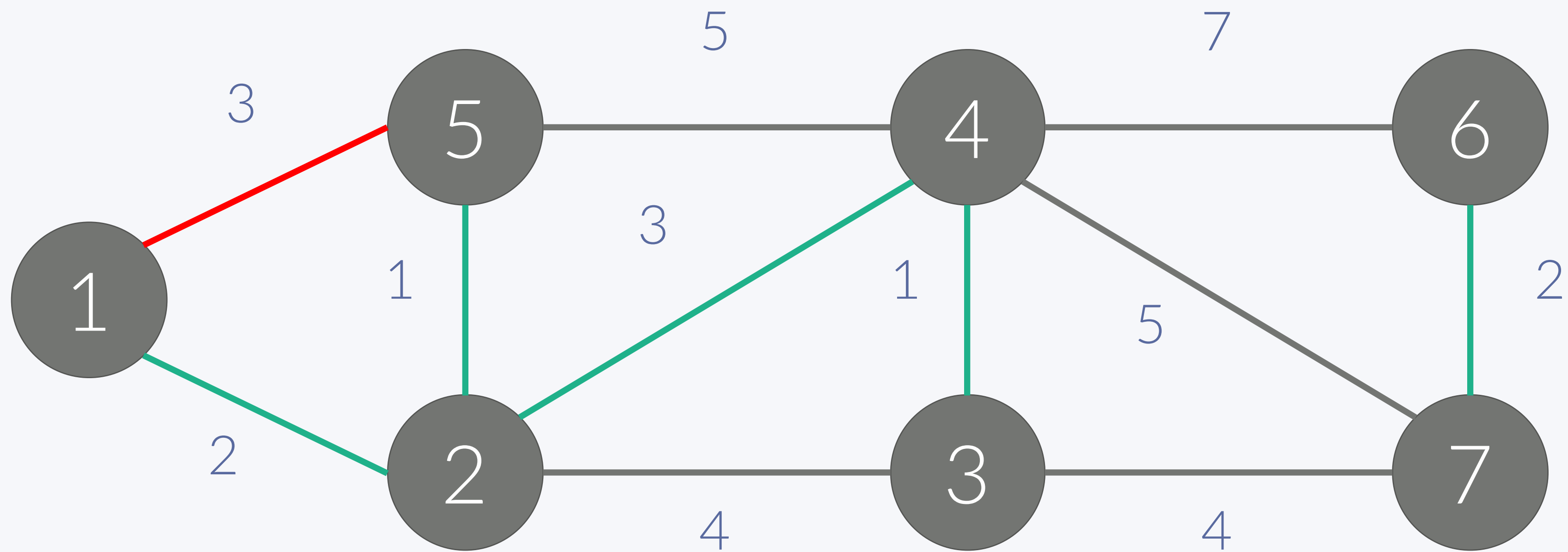
75



# 크루스칼

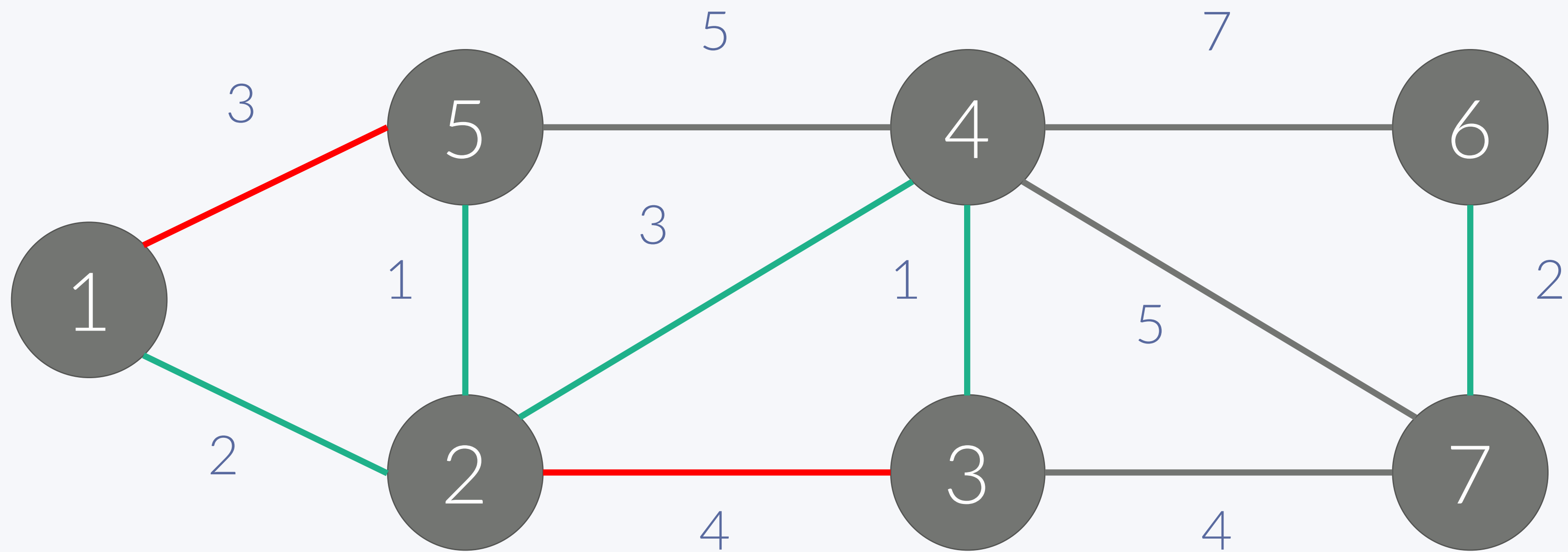
Kruskal

76



# 크루스칼

Kruskal

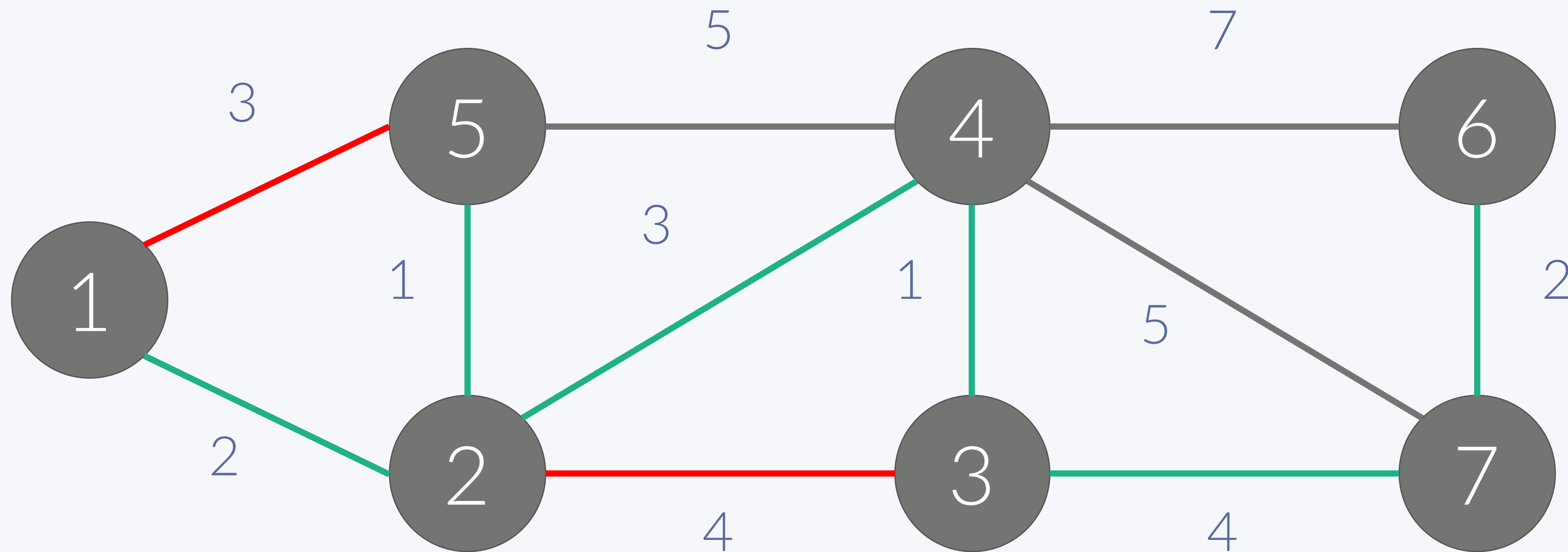


# 크루스칼

Kruskal

78

시간복잡도  
정렬  
ElogV  
크루스칼



# 최소 스패닝 트리

79

<https://www.acmicpc.net/problem/1197>

- 그래프가 주어졌을 때, 그 그래프의 최소 스패닝 트리를 구하기

# 최소 스패닝 트리

<https://www.acmicpc.net/problem/1197>

- C/C++: <https://gist.github.com/Baekjoon/47d844c037dedeab690a>
- Java: <https://gist.github.com/Baekjoon/22d20713a18608a2fbd2>



# 최단 경로

27.21

1(2522) 124

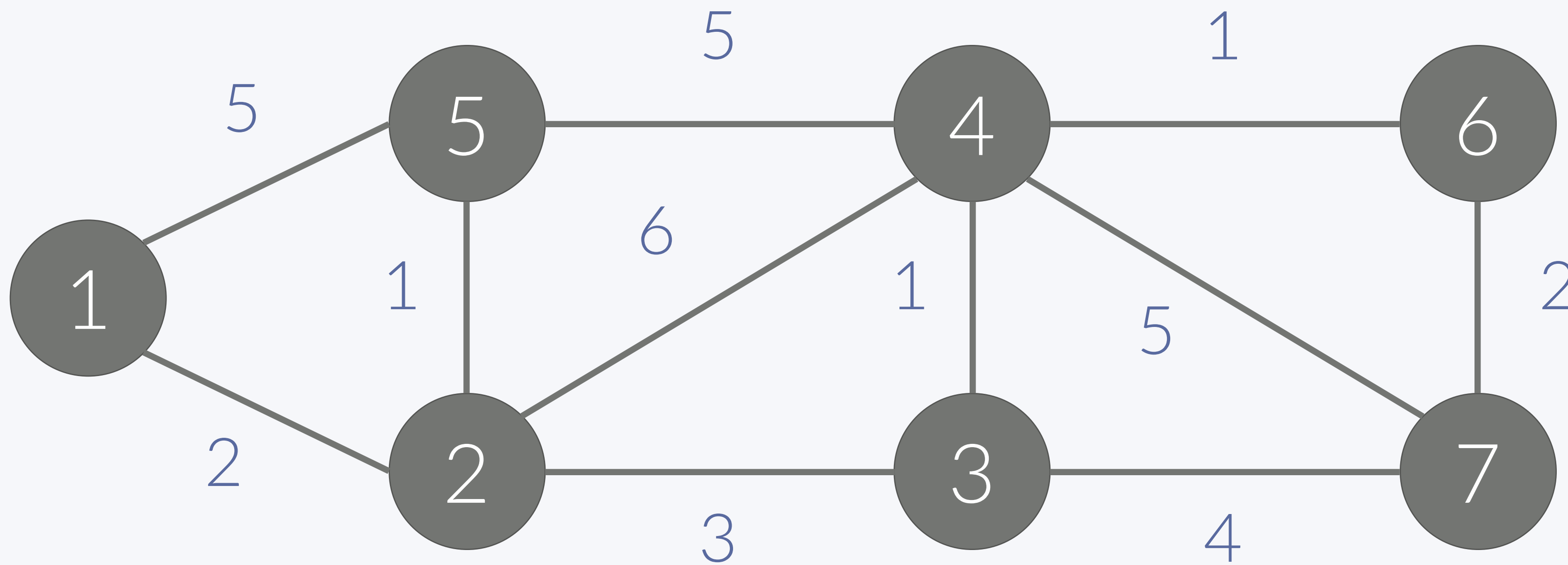
1(2520) 124

# 최단 경로

82

Single Source Shortest Path

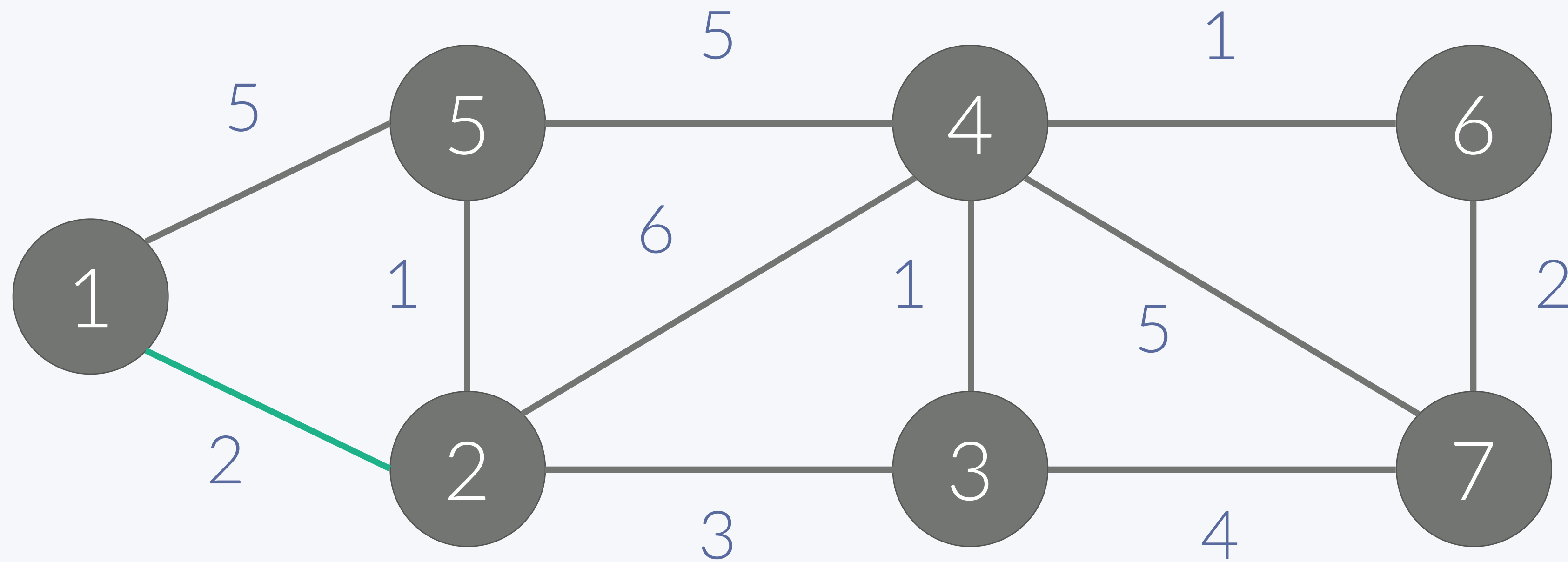
- 시작점이 1개일 때, 다른 모든 곳으로 가는 최단 경로 구하기



# 최단 경로

Single Source Shortest Path

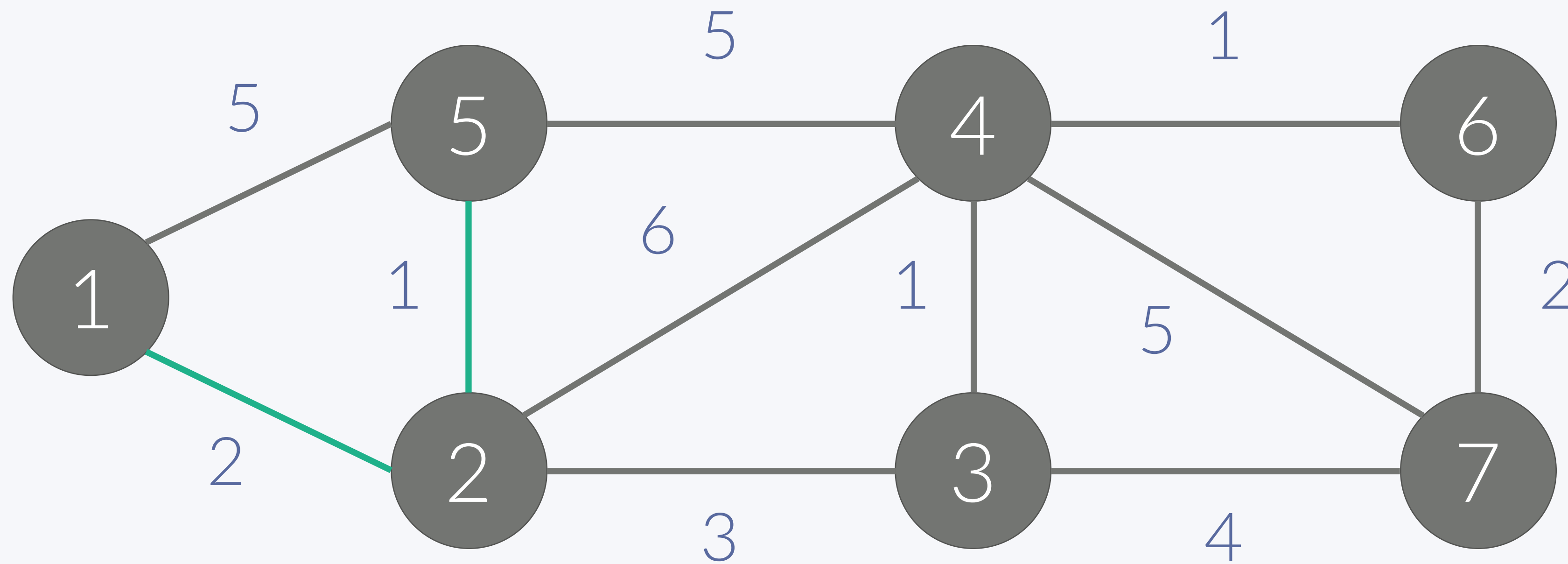
- 1에서 2까지 최단 경로



# 최단 경로

Single Source Shortest Path

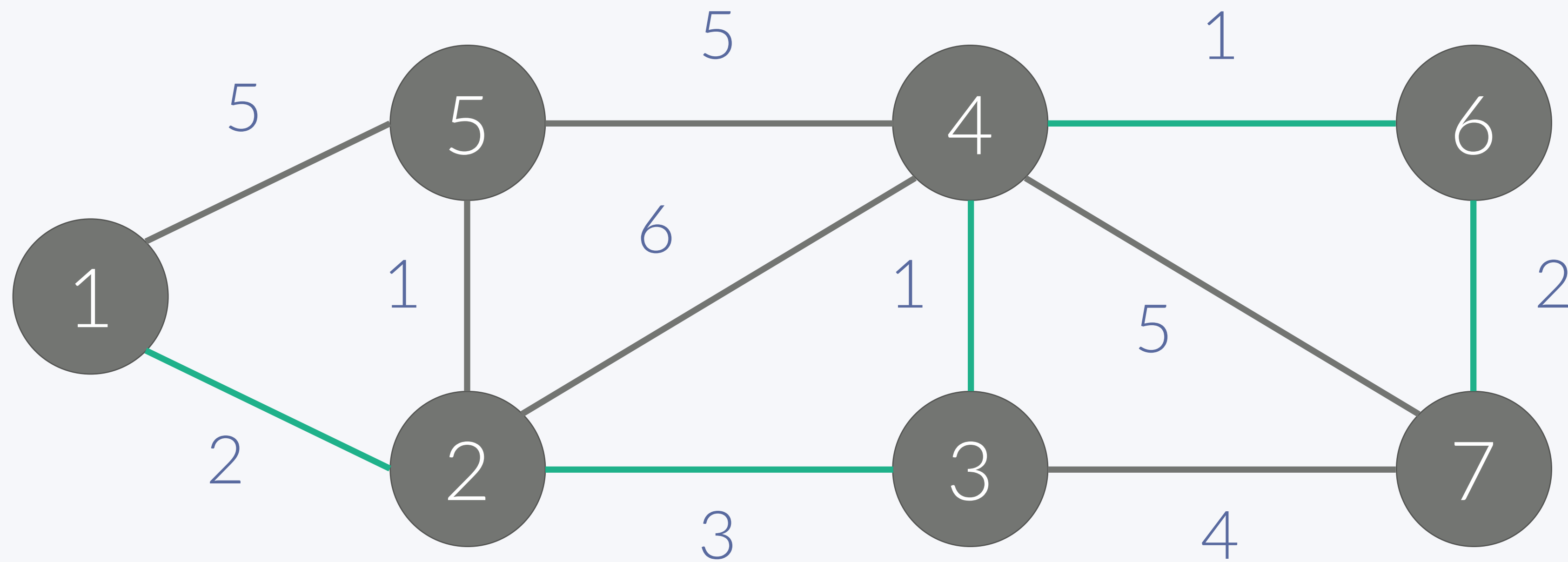
- 1에서 5까지 최단 경로



# 최단 경로

Single Source Shortest Path

- 1에서 7까지 최단 경로



# 최단 경로

Single Source Shortest Path

$A \rightarrow B$  최단경로는 최대  $V-1$ 개 간선

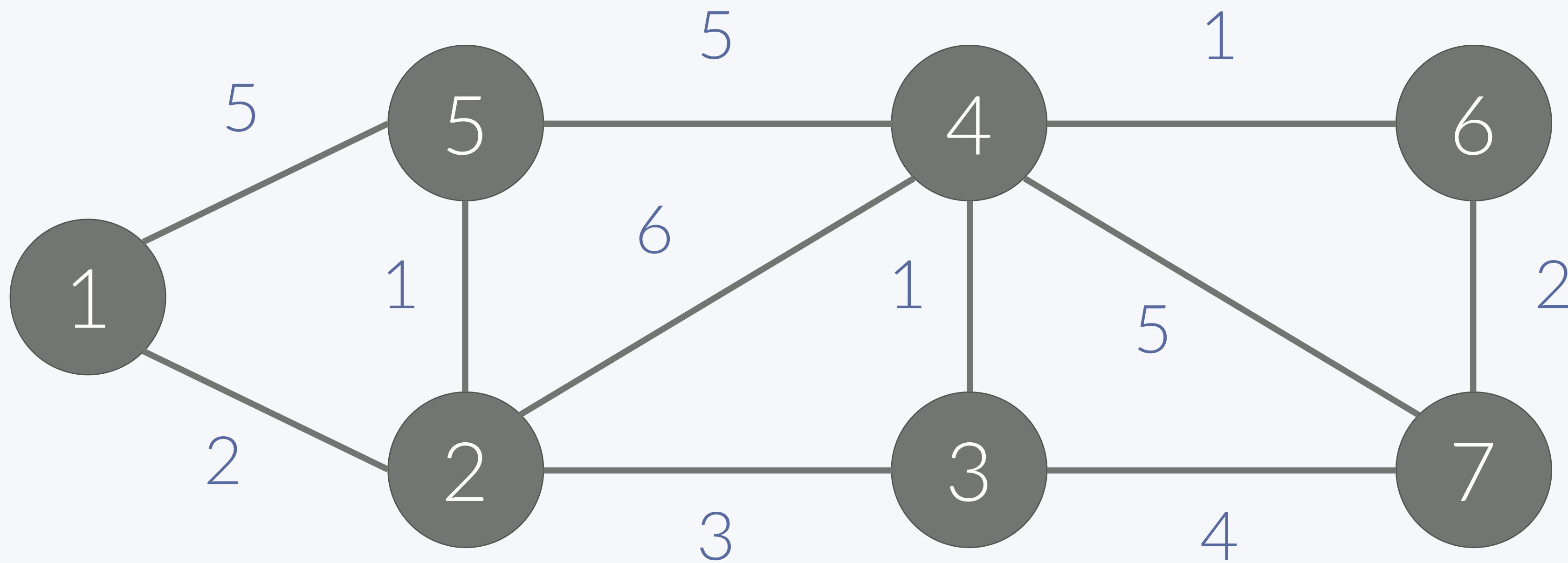
86

간선:  $V-1$ 개

- A  $\rightarrow$  B로 가는 최단 경로는 최대  $N-1$ 개의 간선으로 이루어져 있다

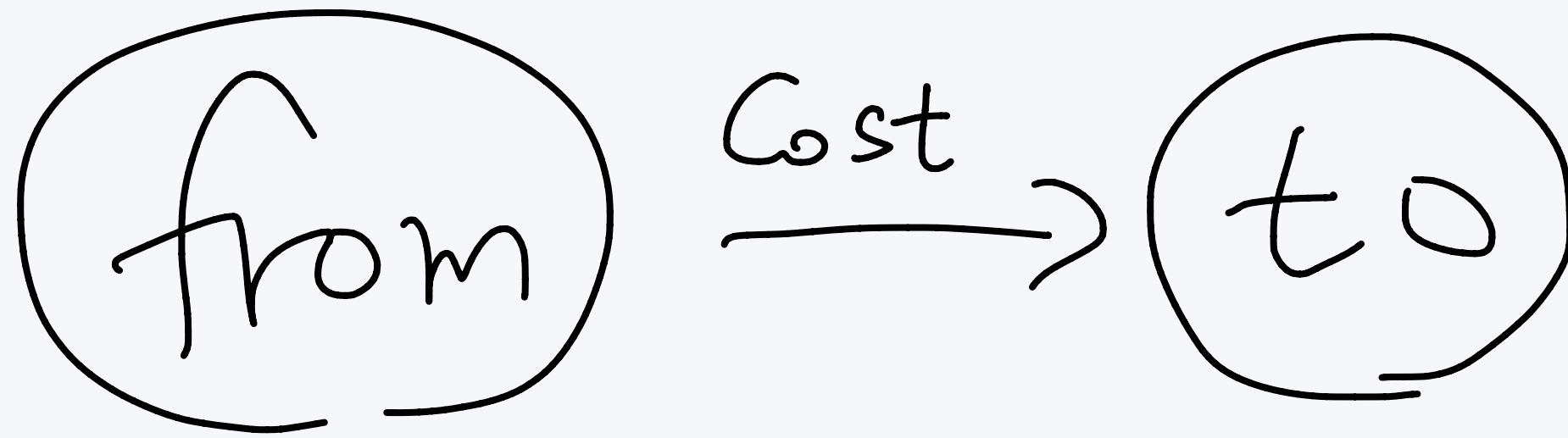
$A \rightarrow$   $(V-2)$   $\rightarrow B$

$A \rightarrow C \rightarrow B$   
↑      ↑



$dist[i] =$

시작점  $\rightarrow i$   
최단거리



if( $dist[to] > dist[from] + Cost$ )  
 $dist[to] = dist[from] + Cost$ ;

---

# 벨만포드

$O(VE)$

$(V-1) \times O(E)$

# 벨만포드

## Bellman-Ford Algorithm

88

- $D[i]$  = 시작점에서  $i$ 로 가는 최단경로
- 1. 모든 간선  $e(u, v, c)$ 에 대해서 다음을 검사한다.
  - $d[v] = \text{Min}(d[v], d[u] + c)$
- 1번 과정을 총  $N-1$ 번 반복한다.



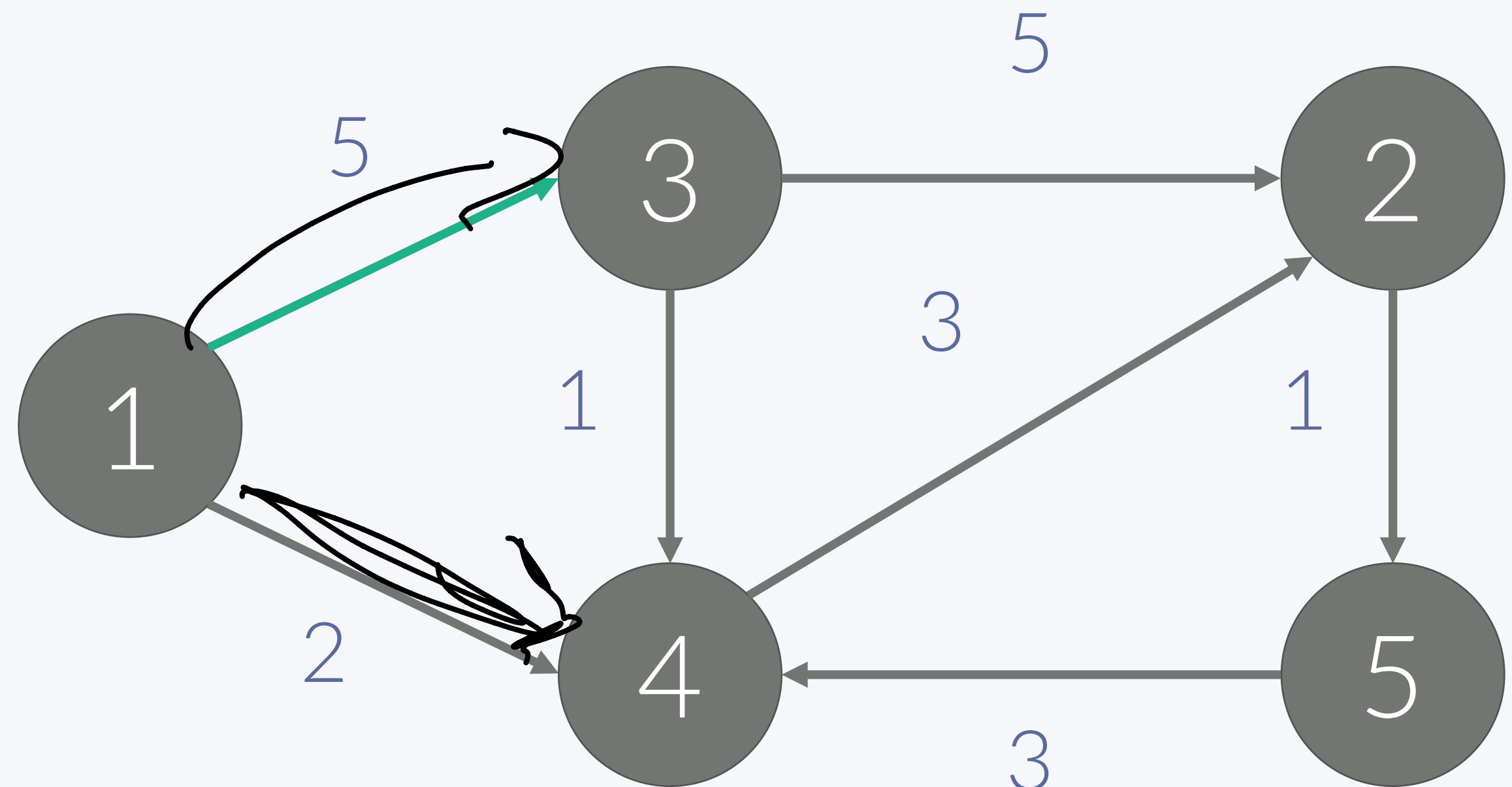
# 벨만포드

## Bellman-Ford Algorithm

- 1단계

i	1	2	3	4	5
D[i]	0	$\infty$	5	$\infty$	$\infty$

Handwritten annotations: A circle around node 1 in the D[i] row. A circle around node 3 in the D[i] row. A curved arrow from node 1 to node 4 with the number 2 written below it.

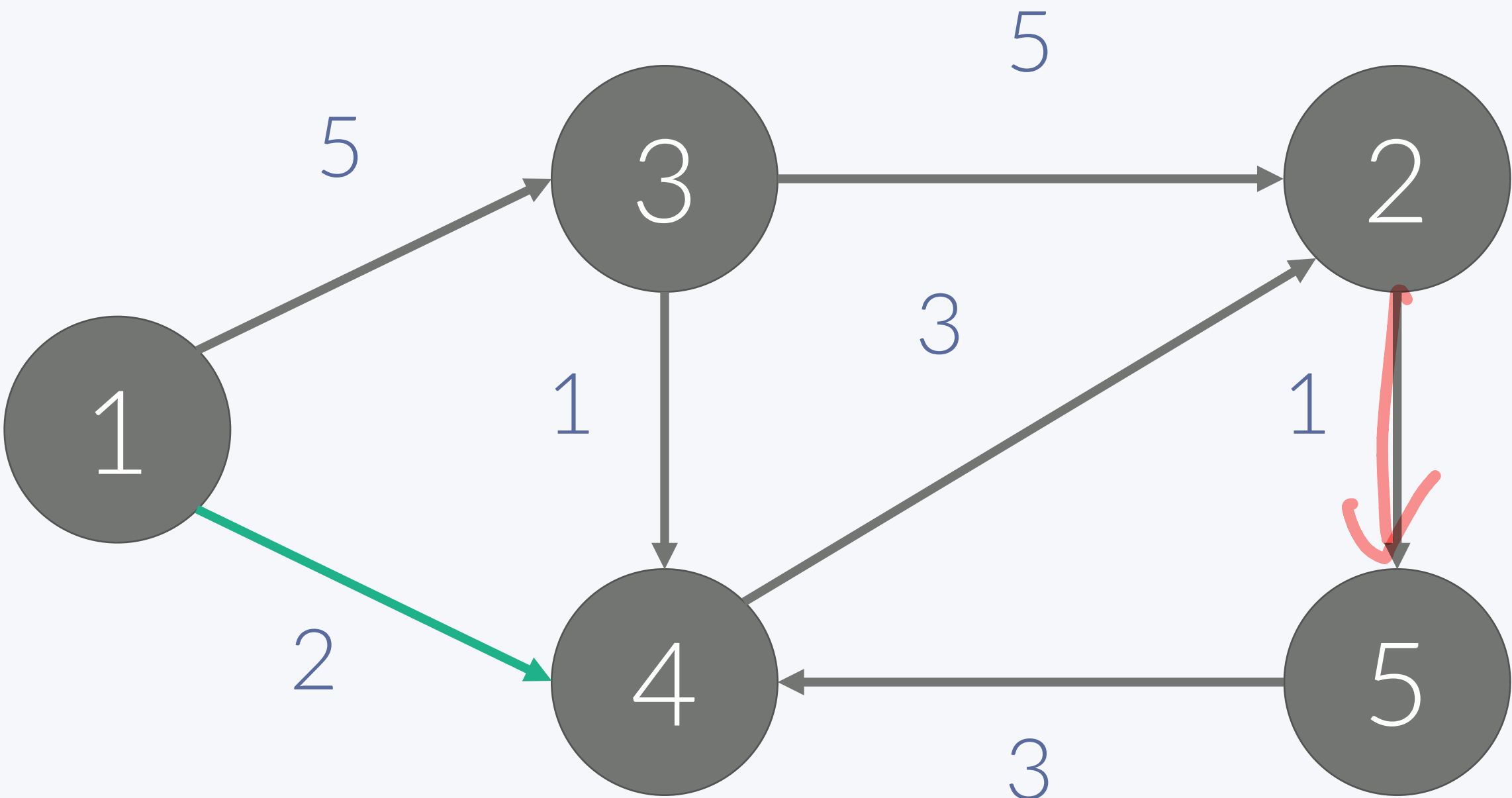


# 벨만포드

Bellman-Ford Algorithm

- 1단계

i	1	2	3	4	5
D[i]	0	$\infty$	5	2	$\infty$

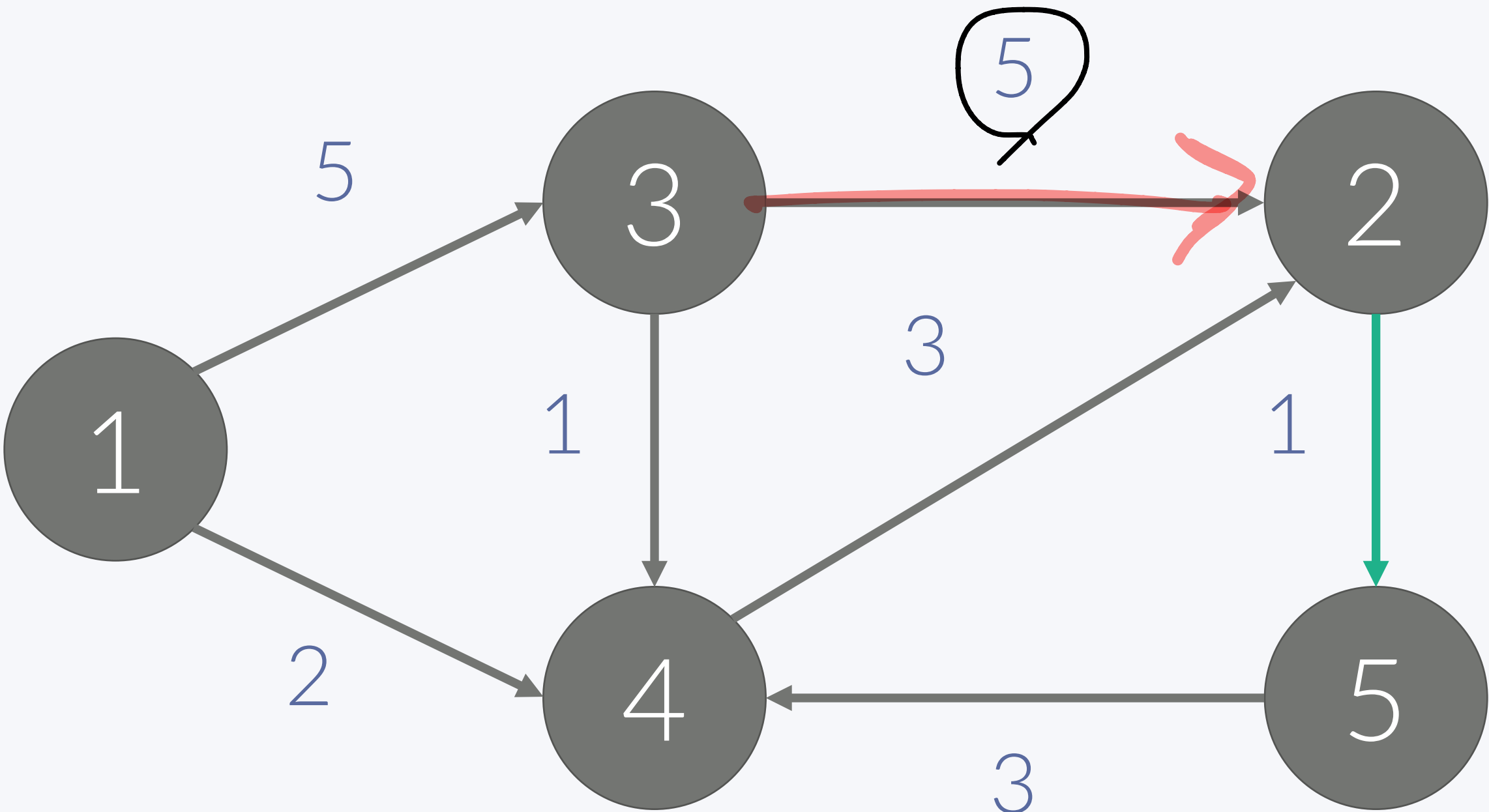


# 벨만포드

Bellman-Ford Algorithm

- 1단계

i	1	2	3	4	5
D[i]	0	<del>∞</del>	5	2	∞

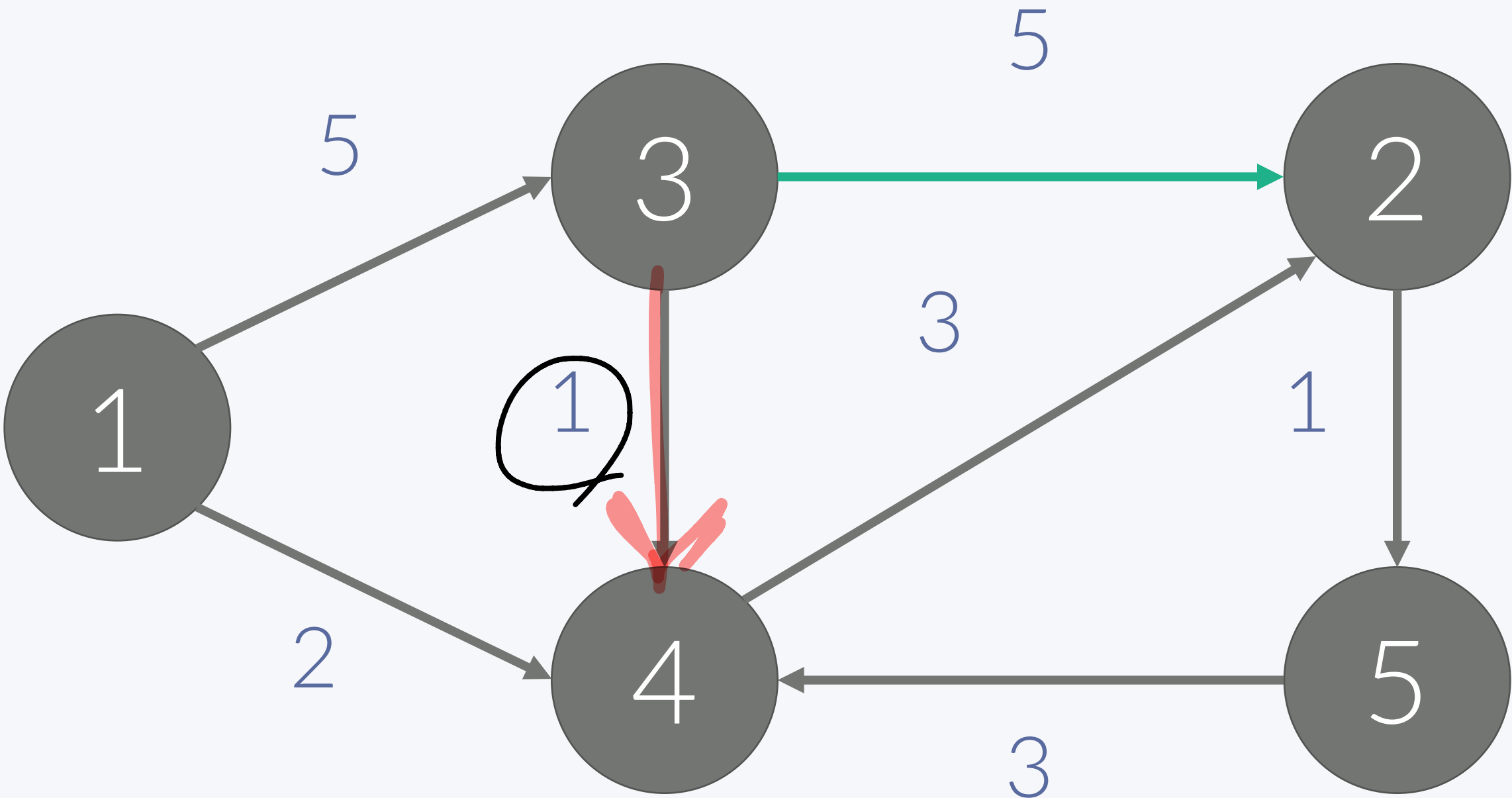


# 벨만포드

Bellman-Ford Algorithm

- 1단계

i	1	2	3	4	5
D[i]	0	10	5	<del>2</del>	$\infty$



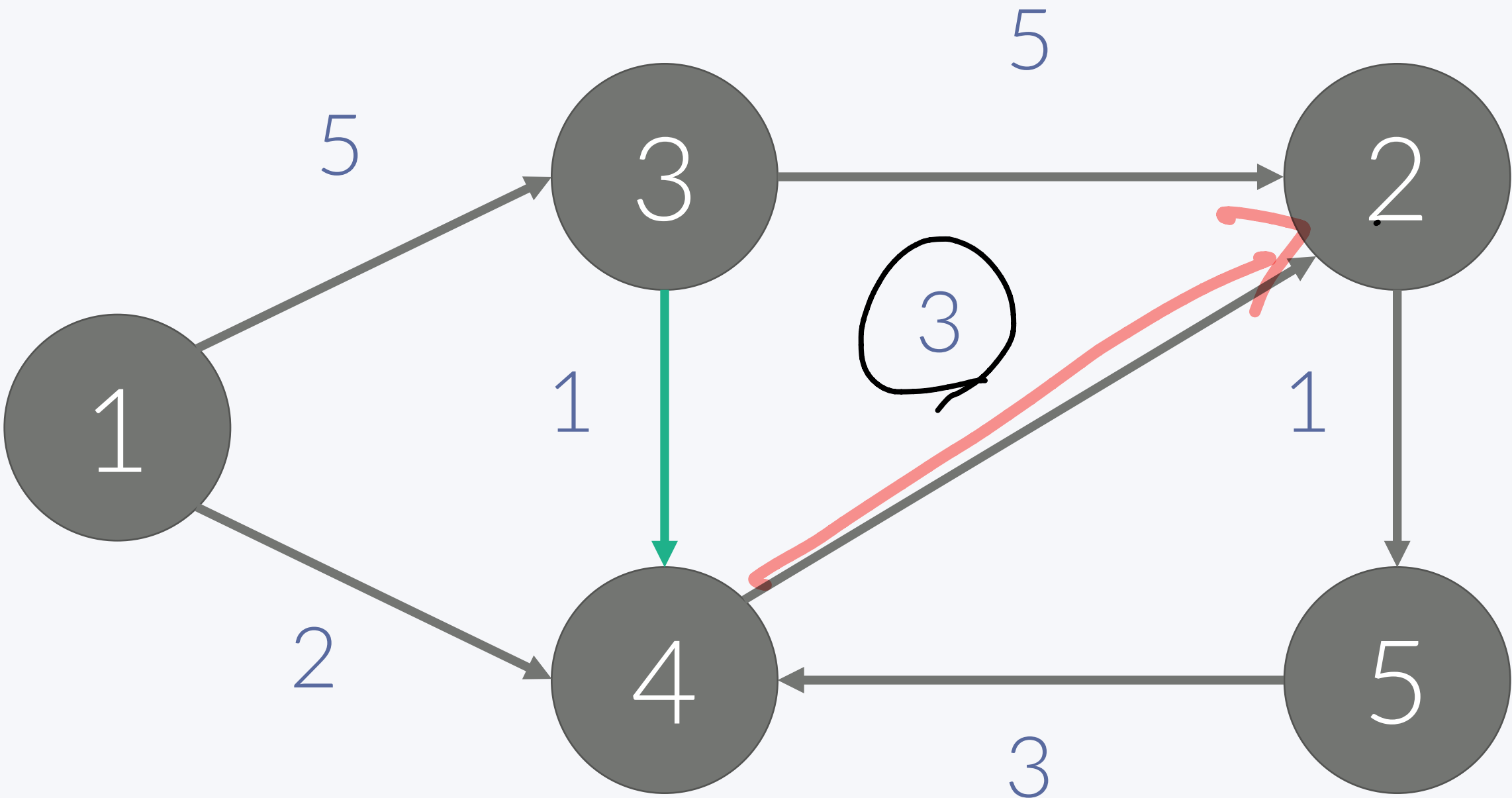
# 벨만포드

Bellman-Ford Algorithm

- 1단계

i	1	2	3	4	5
D[i]	0	5	5	2	$\infty$

*(Handwritten annotations: A circle around the value 2 in the D[4] cell, and an arrow pointing from this cell to the value 5 in the D[2] cell with the number 3 written below it.)*

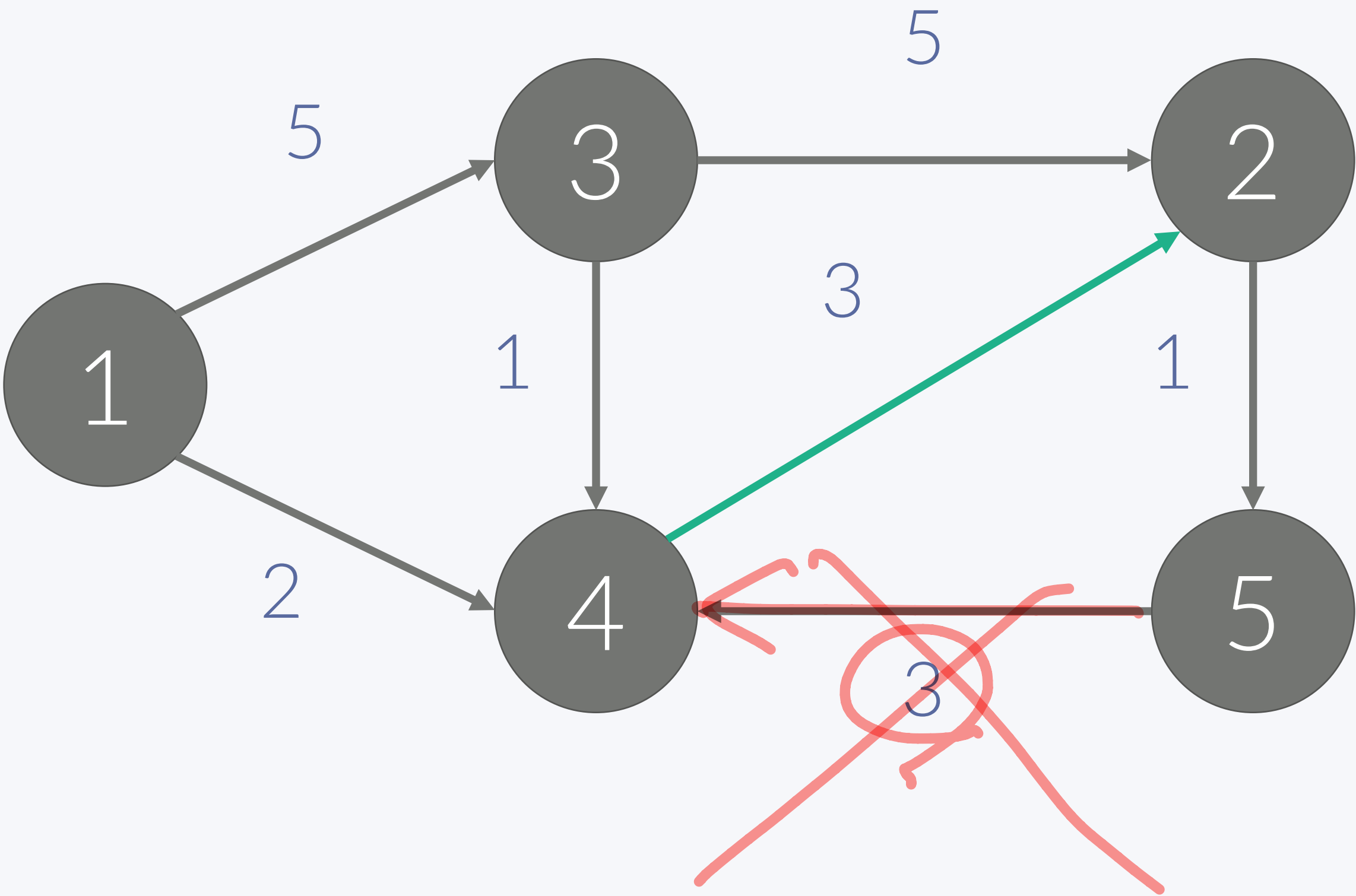


# 벨만포드

Bellman-Ford Algorithm

- 1단계

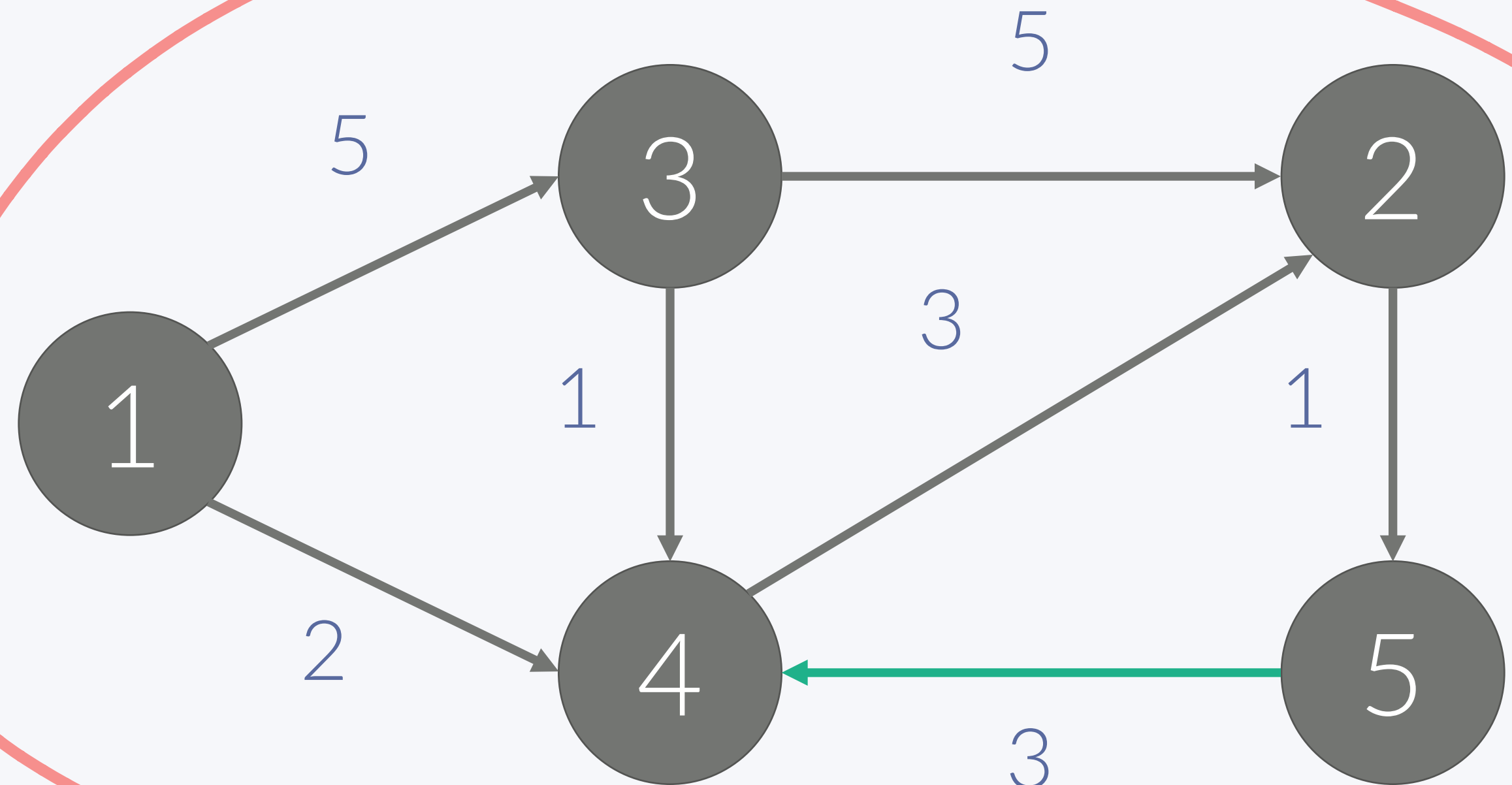
i	1	2	3	4	5
D[i]	0	5	5	2	<del><math>\infty</math></del>



# 벨만포드

# Bellman-Ford Algorithm

- 1단계



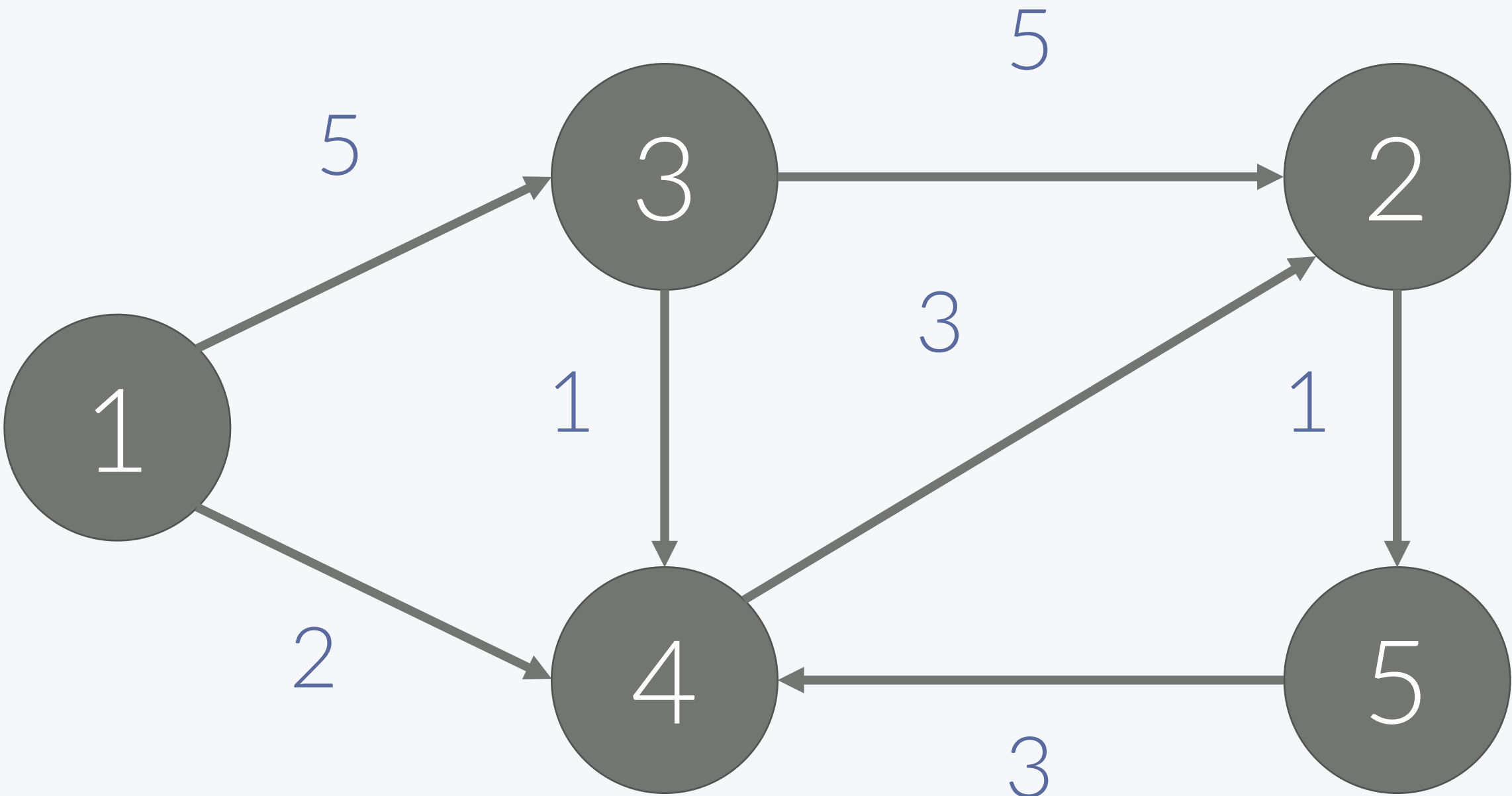
i	1	2	3	4	5
D[i]	0	5	5	2	$\infty$

# 벨만포드

Bellman-Ford Algorithm

- 이런 단계를 N-1번 반복한다

i	1	2	3	4	5
D[i]	0	5	5	2	$\infty$



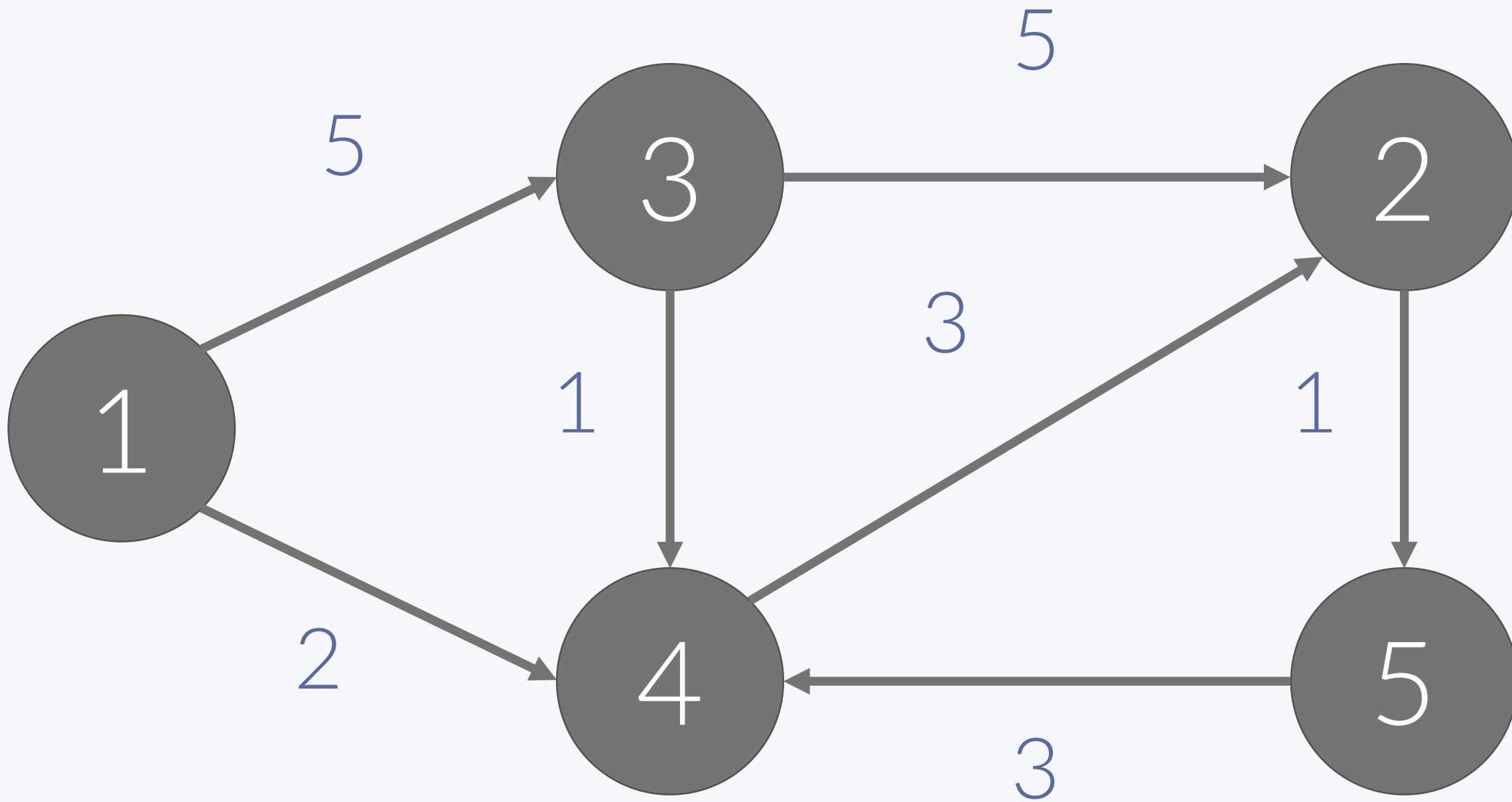


# 벨만포드

Bellman-Ford Algorithm

- 이런 단계를 N-1번 반복한다

$O(VE)$

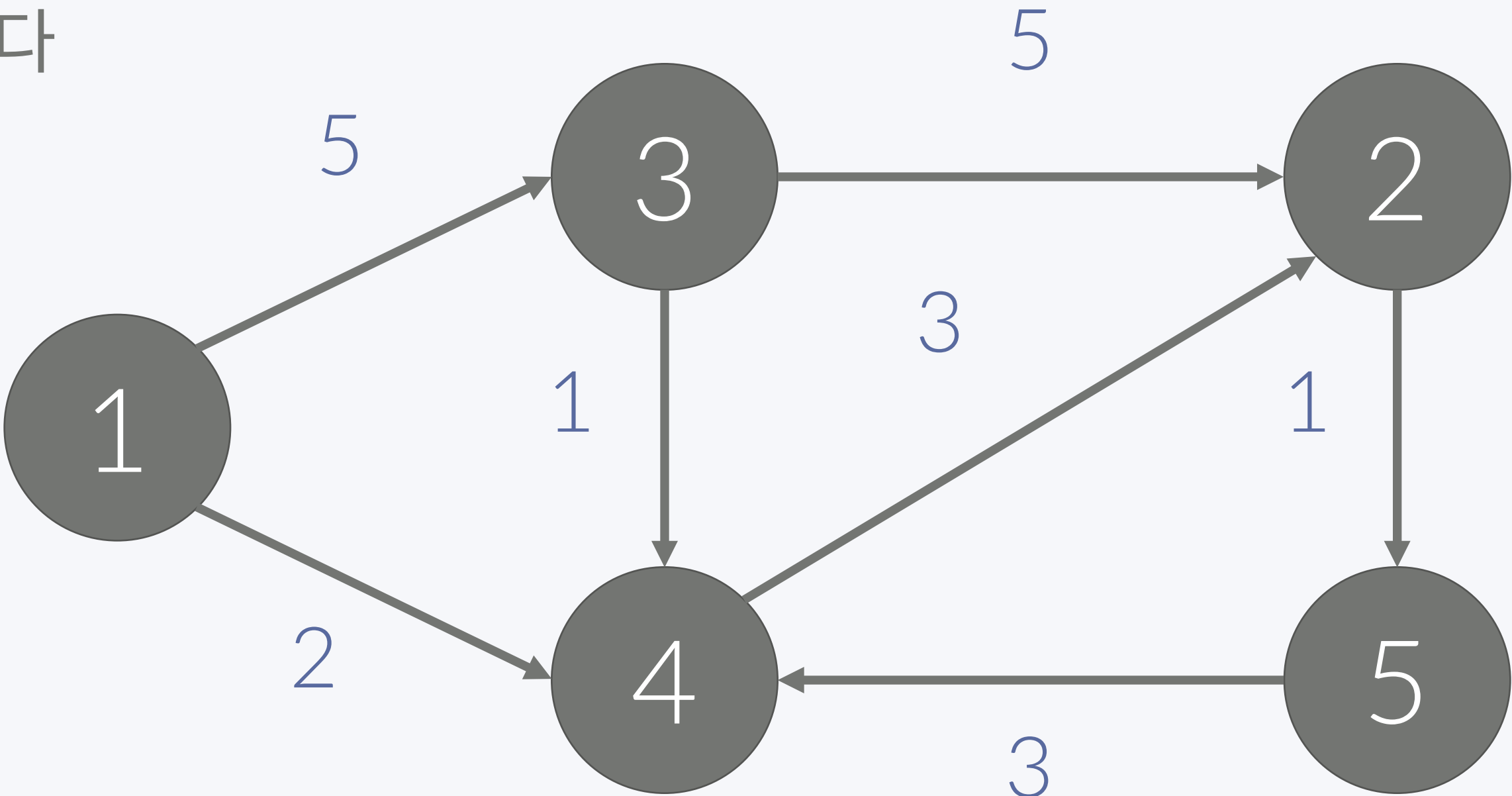
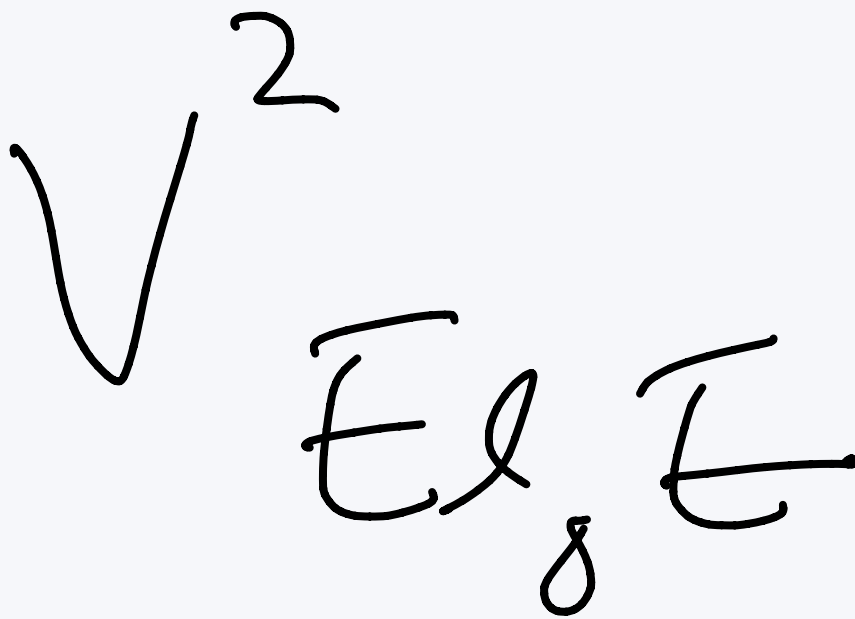


i	1	2	3	4	5
D[i]	0	5	5	2	6

# 벨만포드

Bellman-Ford Algorithm

- 시간 복잡도:  $O(VE)$
- $E \leq V^2$  이기 때문에  $O(V^3)$
- 가중치가 음수가 있는 경우에도 사용할 수 있다



i	1	2	3	4	5
D[i]	0	5	5	2	6

# 타임머신

<https://www.acmicpc.net/problem/11657>

- N개의 도시가 있다
- 한 도시에서 출발하여 다른 도시에 도착하는 버스가 M개 있다
- 각 버스는 A, B, C로 나타낼 수 있는데, A는 시작도시, B는 도착도시, C는 버스를 타고 이동하는데 걸리는 시간이다
- 시간 C가 양수가 아닌 경우가 있다.
- $C = 0$ 인 경우는 순간 이동을 하는 경우,  $C < 0$ 인 경우는 타임머신으로 시간을 되돌아가는 경우이다.
- 1번 도시에서 출발해서 나머지 도시로 가는 가장 빠른 시간을 구하는 프로그램을 작성하시오.

# 타임머신

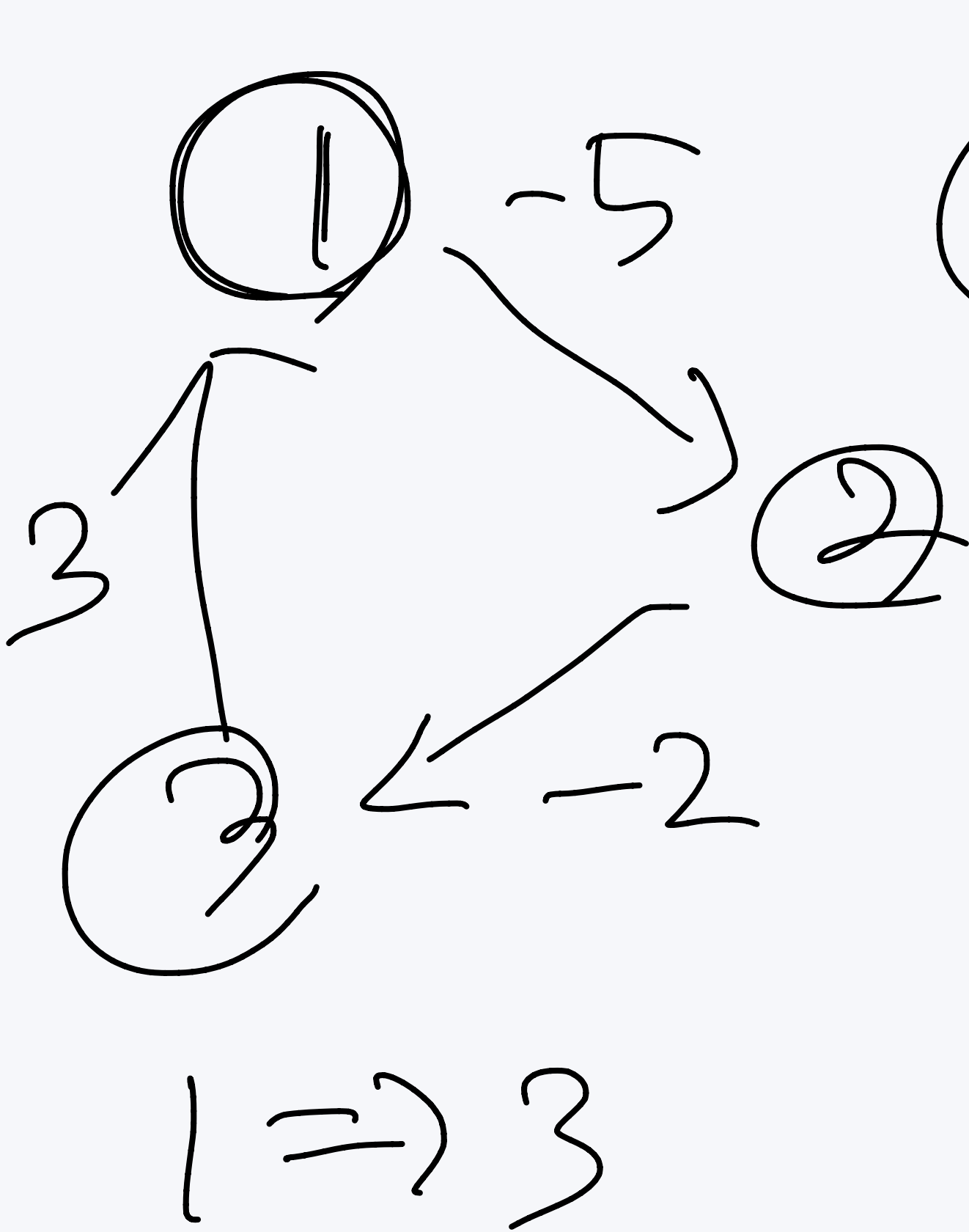
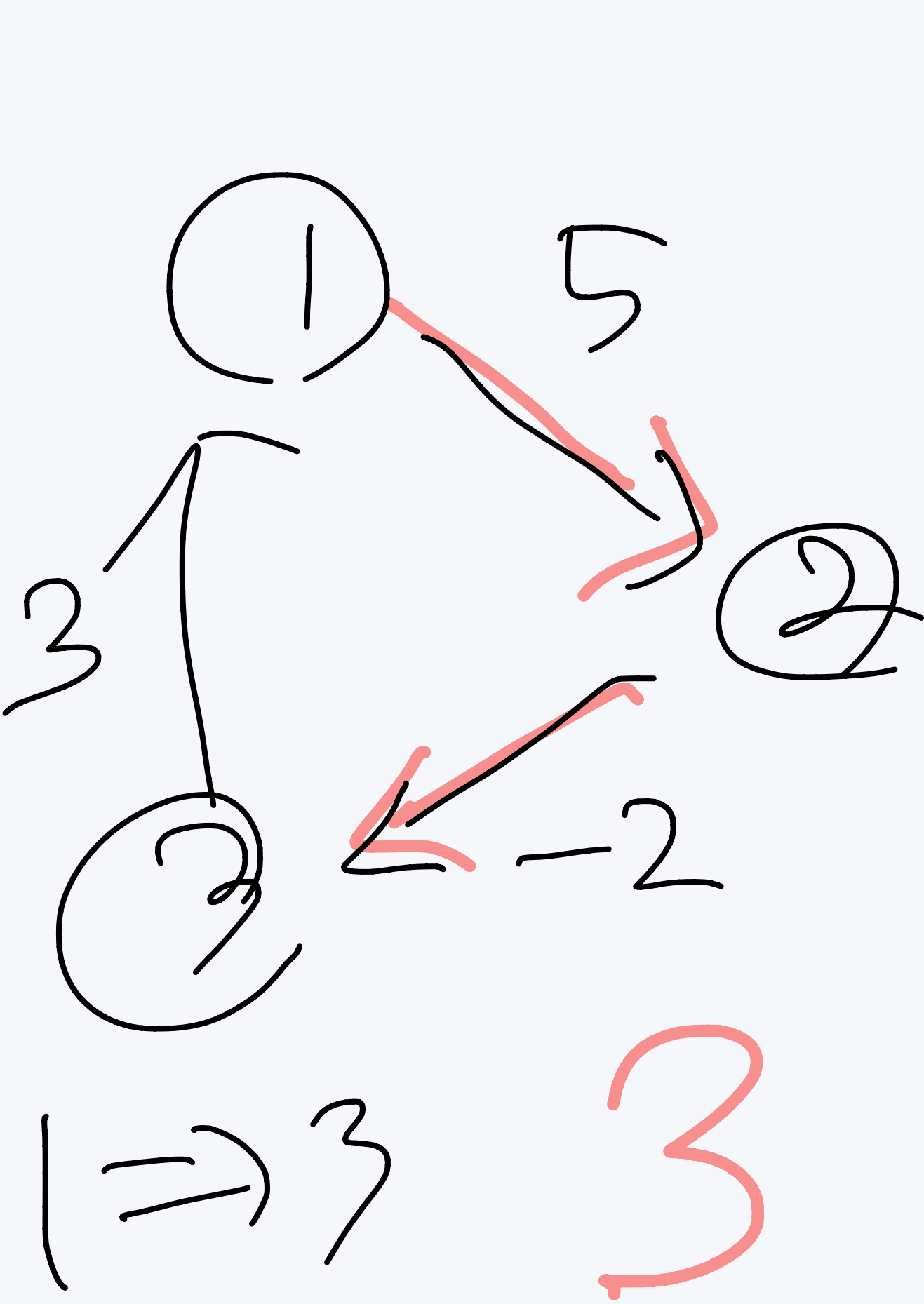
<https://www.acmicpc.net/problem/11657>

$1 \Rightarrow 3$

음수 사이클

- 음수가 있기 때문에, 벨만포드 알고리즘을 이용해서 최단거리를 구해야 한다.

최단  $\times$



$1 \rightarrow 2 \rightarrow 3 \rightarrow 1$   
 $\times \infty$

# 타임머신

101

<https://www.acmicpc.net/problem/11657>

- C++: <https://gist.github.com/Baekjoon/2376cd9ee0e01a284ad1>

10

2

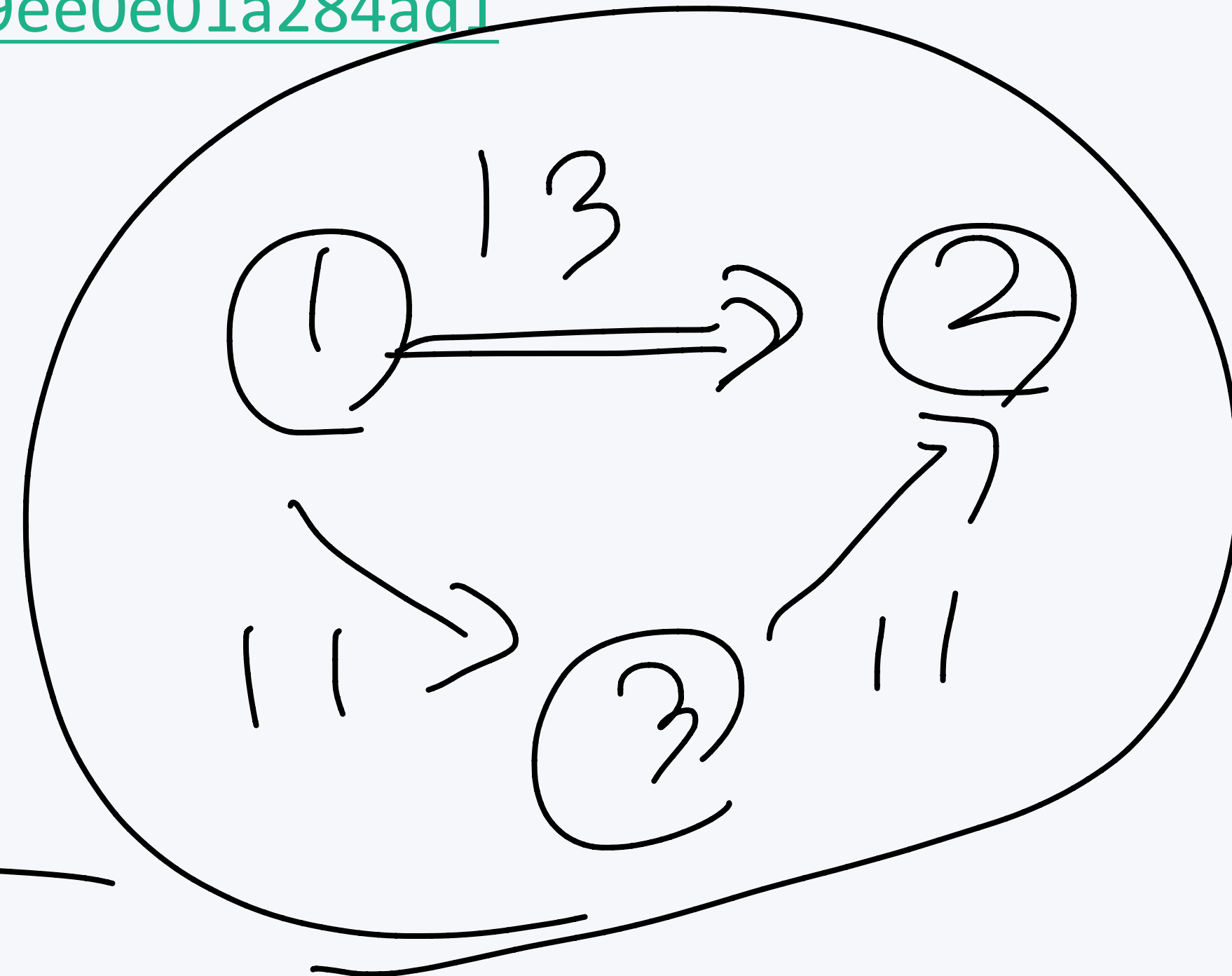
V-1000

결과: 2122

1000

2122

1000



13

22

# 웜홀

<https://www.acmicpc.net/problem/1865>

102

1441F

- N개의 지점 사이에는 M개의 도로와 W개의 웜홀이 있다
- 어떤 지점에서 출발을 하여서 시간여행을 하기 시작하여 다시 출발을 하였던 위치로 돌아왔을 때, 출발을 하였을 때 보다 시간이 되돌아가 있는 경우 찾기

$$d[y] > d[x] + 2$$

$\infty \quad \infty \quad -1$

$$d[y]$$

# 웜홀

103

<https://www.acmicpc.net/problem/1865>

- 음수로 되어있는 사이클을 찾는 문제

# 웜홀

104

<https://www.acmicpc.net/problem/1865>

- 최단 경로는 항상 최대  $N-1$ 개로 구성되어 있기 때문에
- $N$ 번째 단계에서 최단 경로가 갱신되면, 음수로 되어있는 사이클이 존재하는 것이다



<https://www.acmicpc.net/problem/1865>

- C++: <https://gist.github.com/Baekjoon/1b249392471b05478fbb>
- Java: <https://gist.github.com/Baekjoon/e0fdb841ab360f806bc6>

# 다익스트라

## Dijkstra

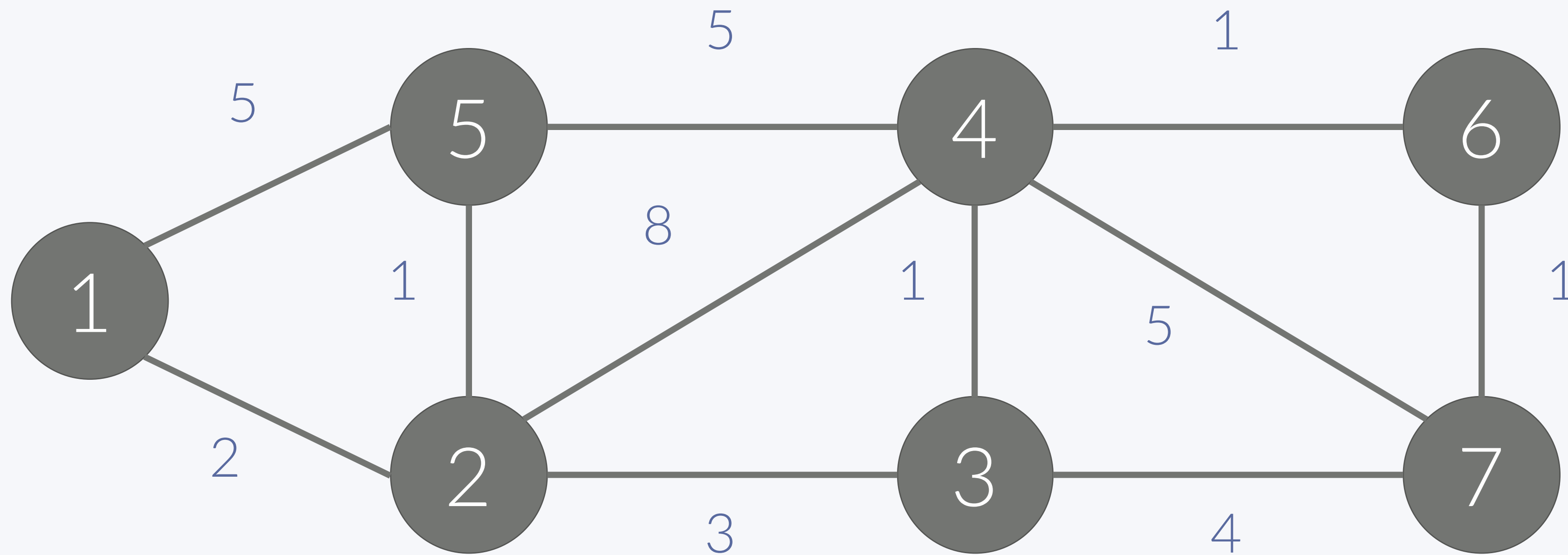
	Min Heap	BST
삽입	$O(1)$	$O(\lg N)$
삭제	$O(\lg N)$	$O(\lg N)$
탐색	X	$O(\lg N)$
구간	$O(\lg N)$	$O(\lg N)$
구간	$O(\lg N)$	$O(\lg N)$

# 다익스트라

## Dijkstra Algorithm

107

- $D[i]$  = 시작  $\rightarrow$   $i$ 로 가는 최단 거리
- $C[i]$  =  $i$ 가 체크되어 있으면 true, 아니면 false



# 다익스트라

## Dijkstra Algorithm

108

1. 체크되어 있지 않은 정점 중에서  $D$ 의 값이 가장 작은 정점  $x$ 를 선택한다.
  2.  $x$ 를 체크한다.
  3.  $x$ 와 연결된 모든 정점을 검사한다.
    - 간선을  $(x, y, z)$ 라고 했을 때
    - $d[y] > d[x] + z$  이면 갱신해준다.
- 1, 2, 3단계를 모든 정점을 체크할 때까지 계속한다.

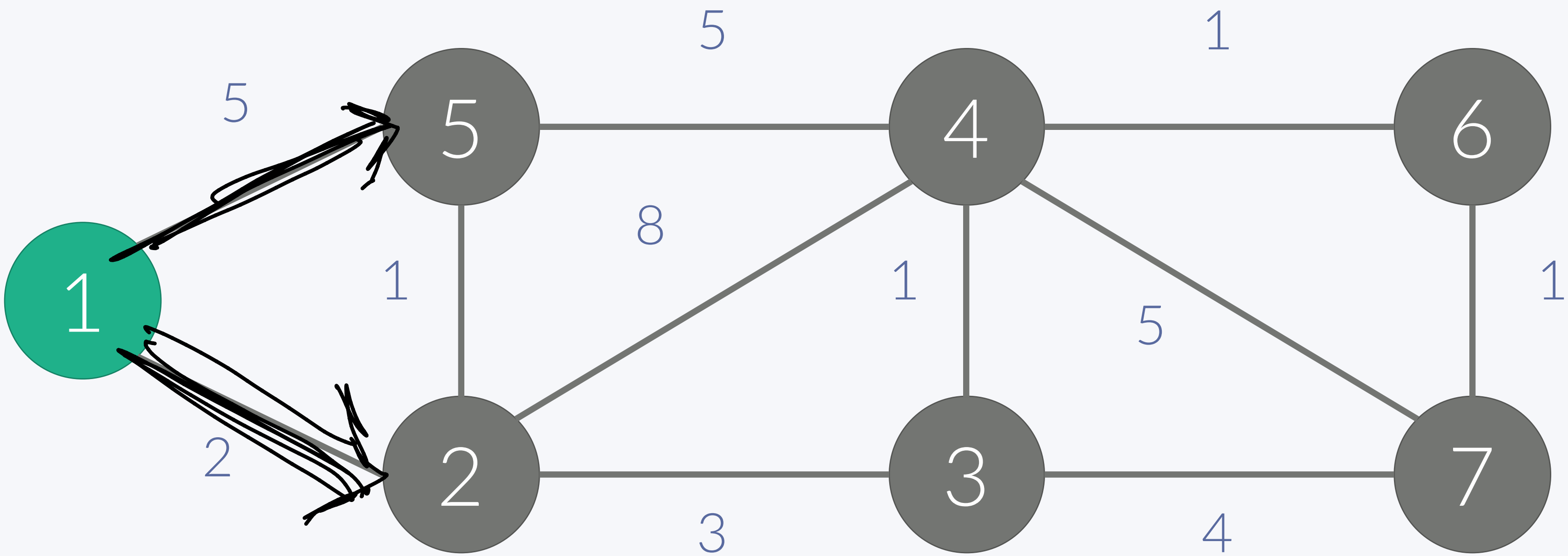
# 다익스트라

Dijkstra Algorithm

- 선택: 1

(정점, Dist)  
(5, 5)  
(2, 2)

i	1	2	3	4	5	6	7
D[i]	0	2	$\infty$	$\infty$	5	$\infty$	$\infty$
C[i]	1						

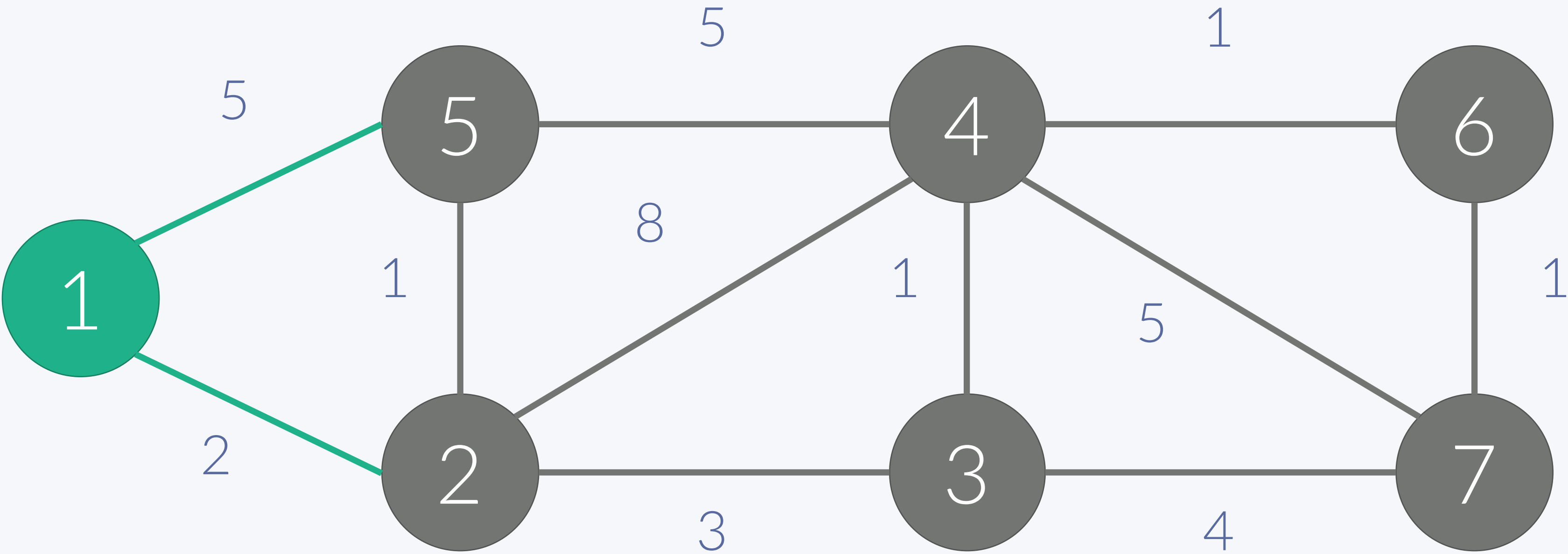


# 다익스트라

Dijkstra Algorithm

- 선택: 1

i	1	2	3	4	5	6	7
D[i]	0	2	$\infty$	$\infty$	5	$\infty$	$\infty$
C[i]	1						



# 다익스트라

Dijkstra Algorithm

- 선택: 2

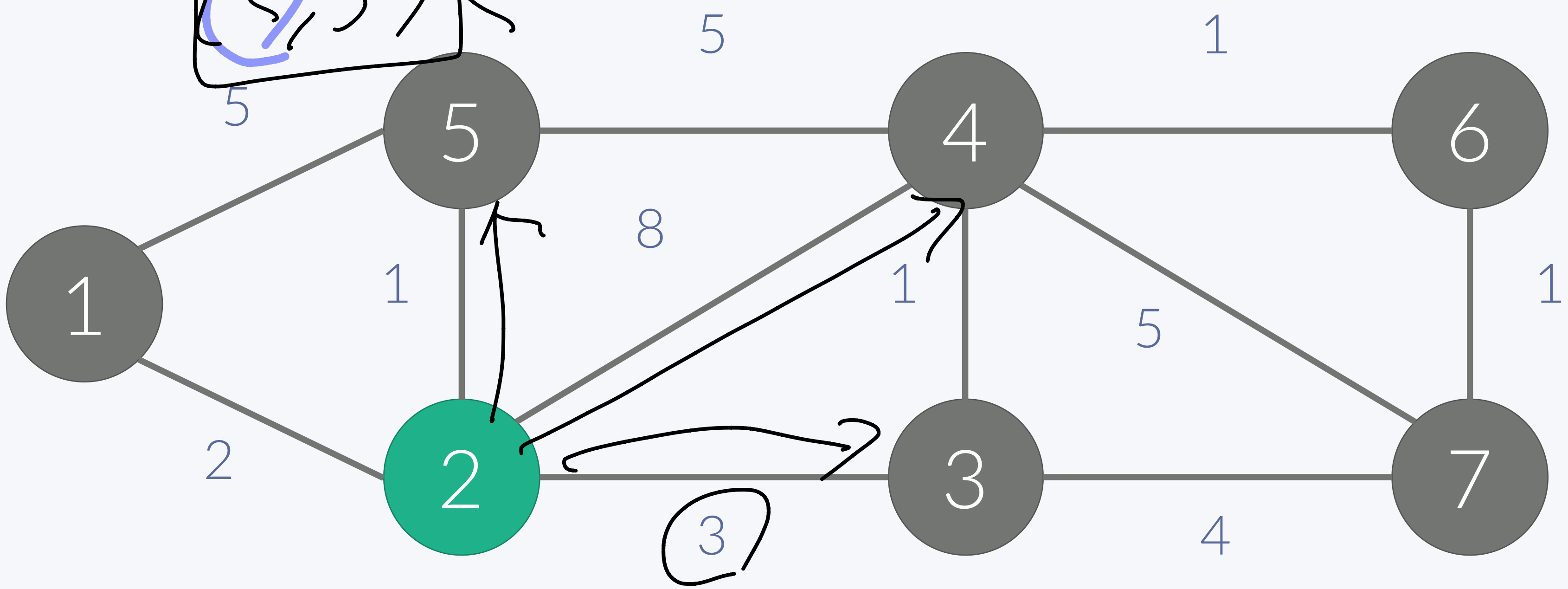
(정렬, Dist)

Edge

i	1	2	3	4	5	6	7
D[i]	0	2	5	10	3	∞	∞
C[i]	1	1					

(3, 5)  
(4, 10)  
(5, 3)

Edge V

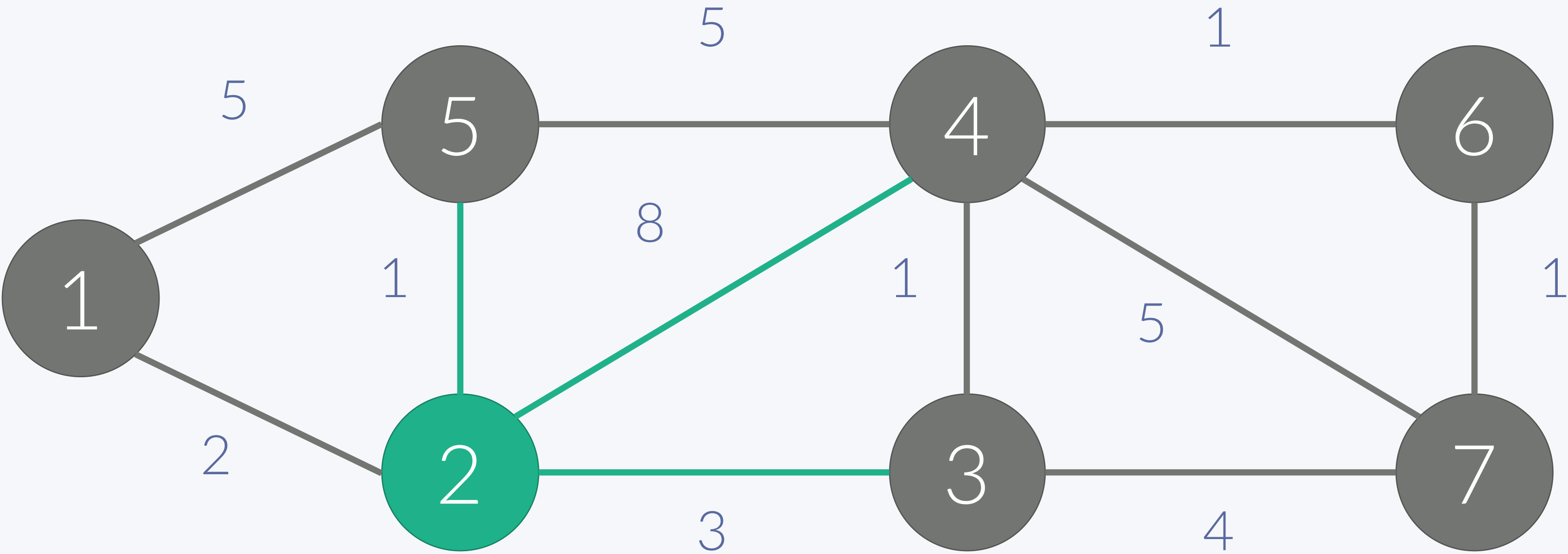


# 다익스트라

Dijkstra Algorithm

- 선택: 2

i	1	2	3	4	5	6	7
D[i]	0	2	5	10	3	$\infty$	$\infty$
C[i]	1	1					





# 다익스트라

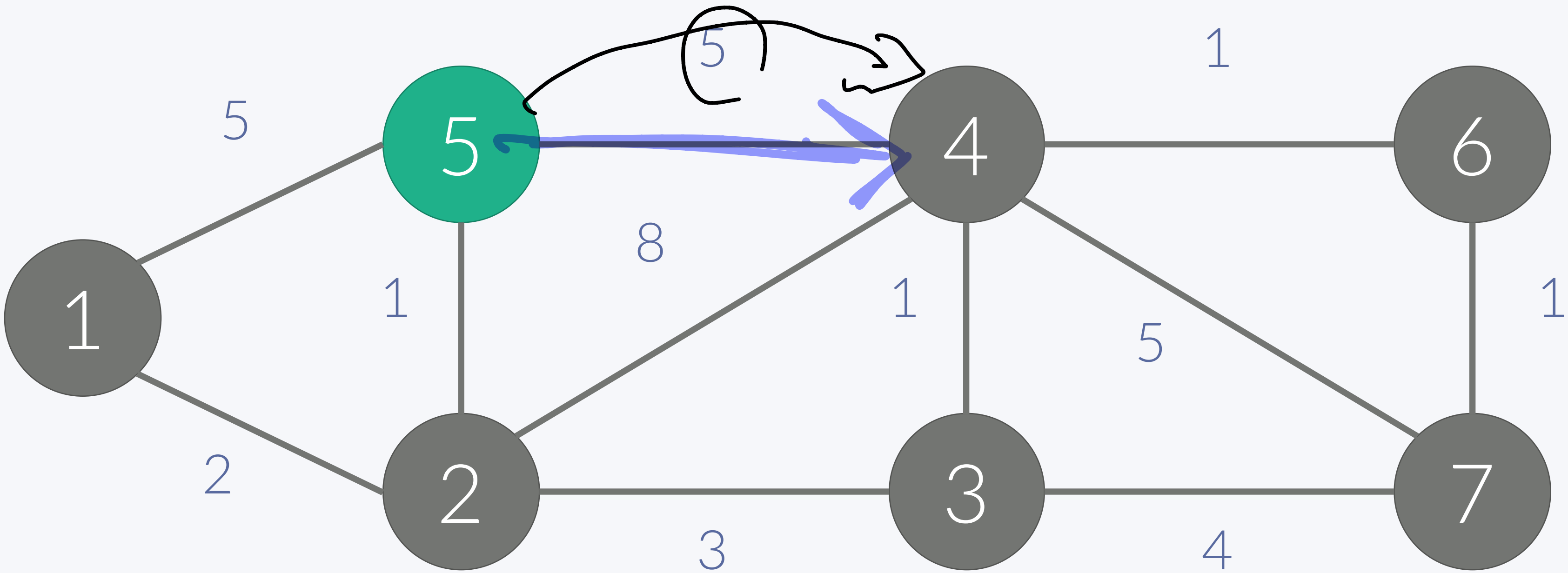
Dijkstra Algorithm

- 선택: 5

Edge

~~(5, 5)~~  
(3, 5)  
(4, 10)  
(4, 8)

i	1	2	3	4	5	6	7
D[i]	0	2	5	10	3	$\infty$	$\infty$
C[i]	1	1			1		

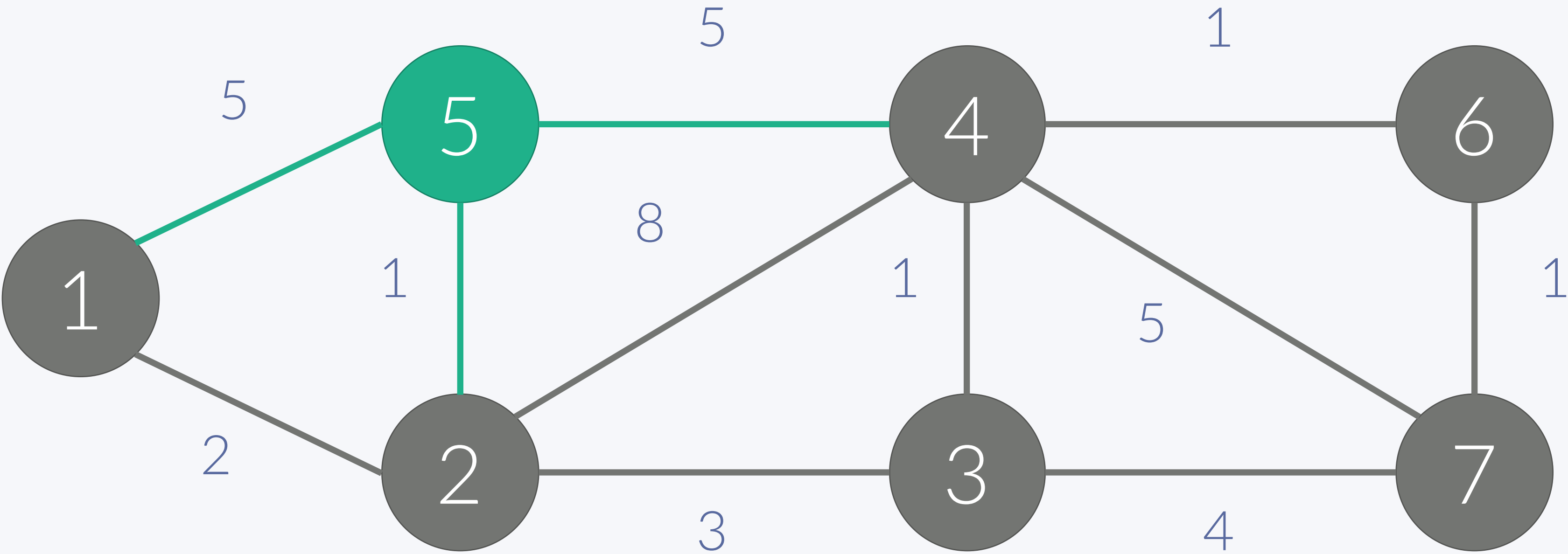


# 다익스트라

Dijkstra Algorithm

- 선택: 5

i	1	2	3	4	5	6	7
D[i]	0	2	5	8	3	$\infty$	$\infty$
C[i]	1	1			1		

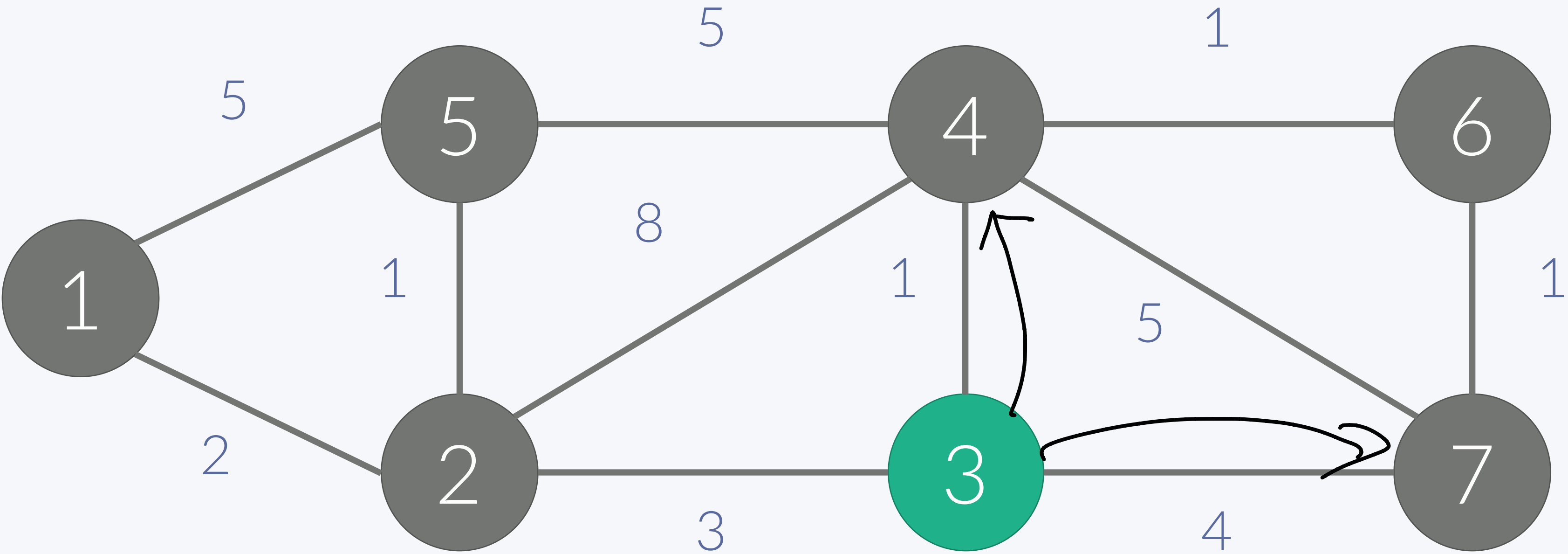


# 다익스트라

Dijkstra Algorithm

- 선택: 3

i	1	2	3	4	5	6	7
D[i]	0	2	5	8	3	$\infty$	$\infty$
C[i]	1	1	1		1		

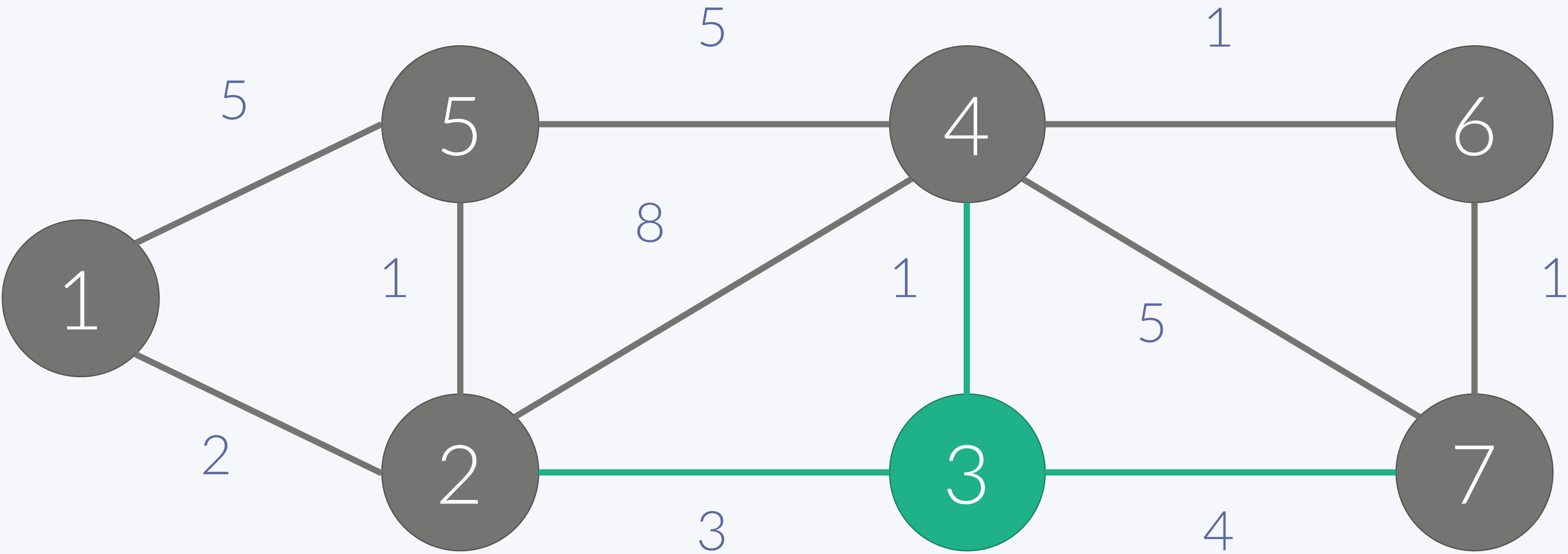


# 다익스트라

Dijkstra Algorithm

- 선택: 3

i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	$\infty$	9
C[i]	1	1	1		1		



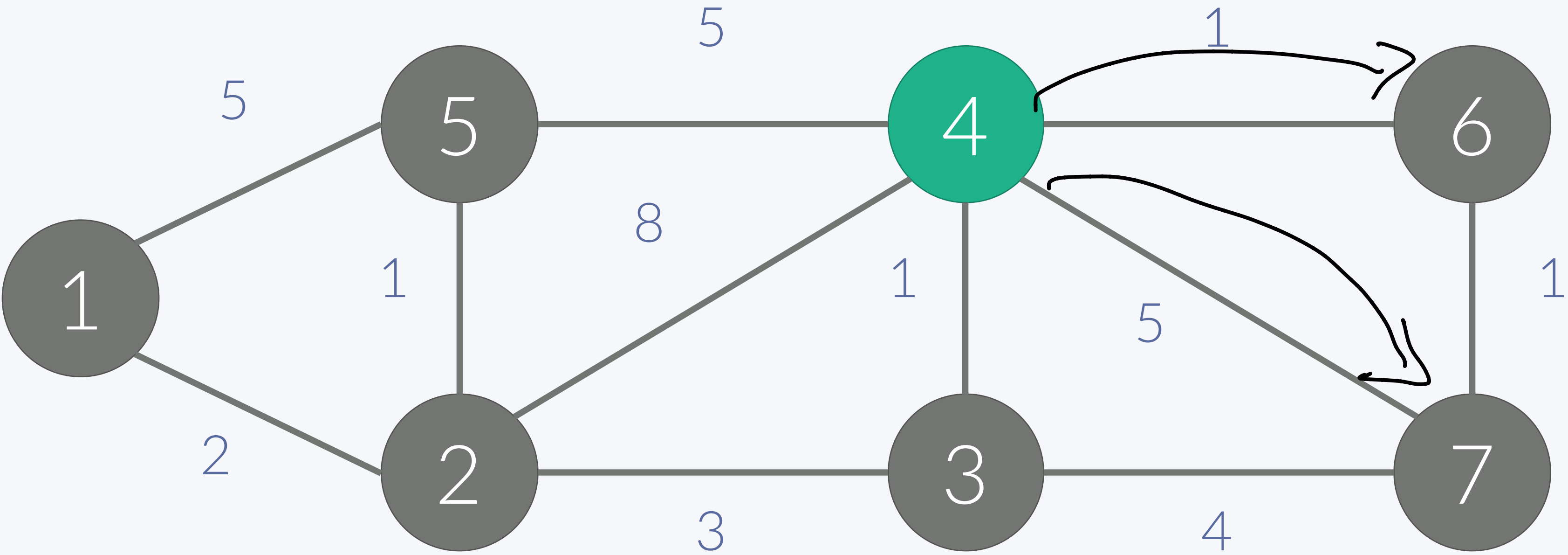
# 다익스트라

Dijkstra Algorithm

- 선택: 4

i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	<del>∞</del>	9
C[i]	1	1	1	1	1		

Handwritten annotations: An arrow from node 4 to node 5 is labeled '1'. An arrow from node 4 to node 6 is labeled '1'. An arrow from node 4 to node 7 is labeled '5'. A double vertical line is drawn next to node 6.

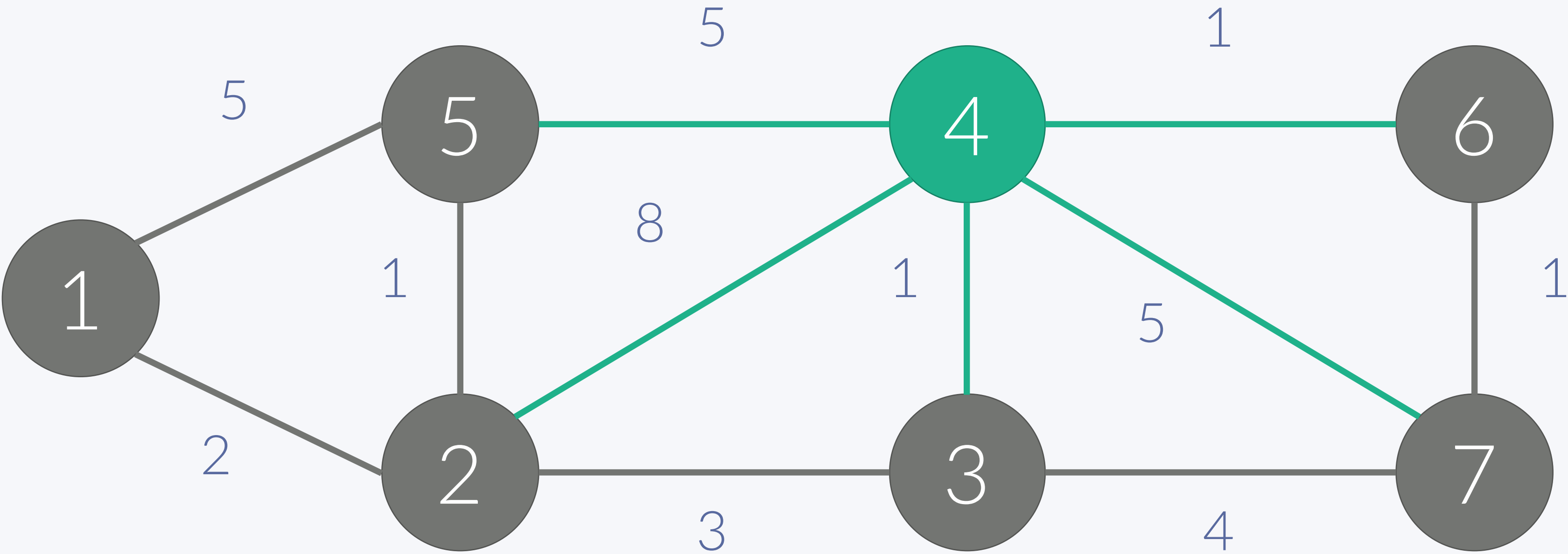


# 다익스트라

Dijkstra Algorithm

- 선택: 4

i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	9
C[i]	1	1	1	1	1		

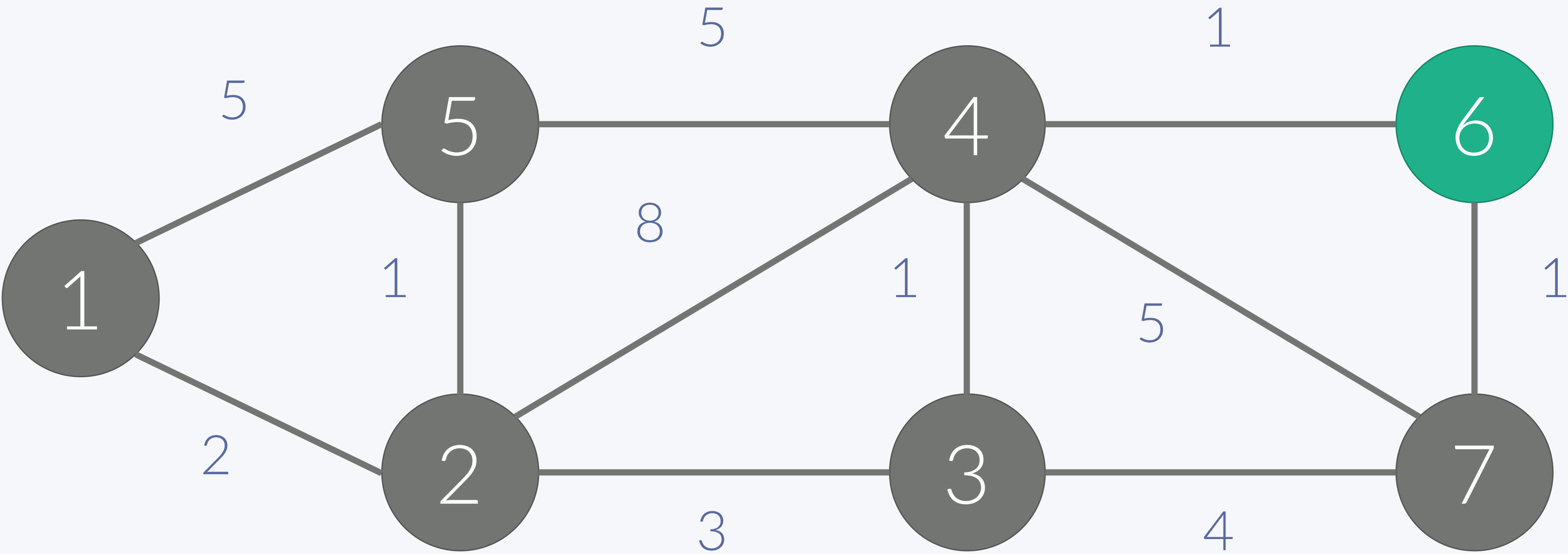


# 다익스트라

Dijkstra Algorithm

- 선택: 6

i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	9
C[i]	1	1	1	1	1	1	



# 다익스트라

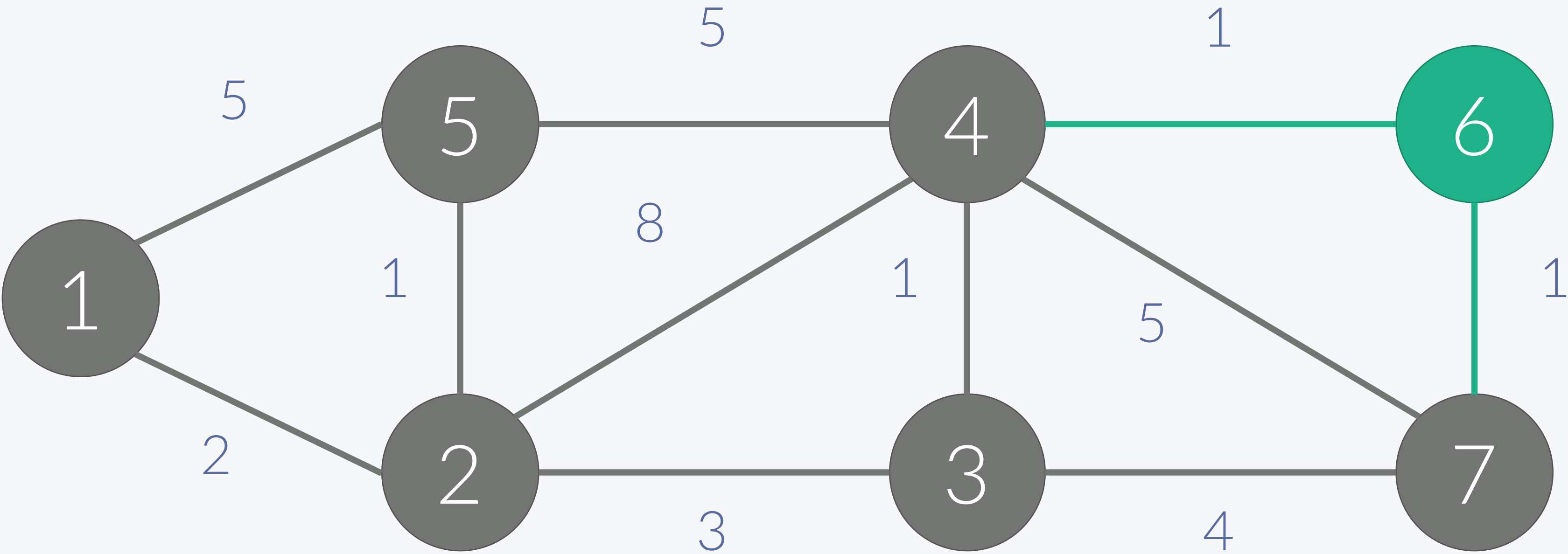
Dijkstra Algorithm

- 선택: 6

$V^2$

시간:  $VH$   
공간:  $O(V)$

i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	8
C[i]	1	1	1	1	1	1	



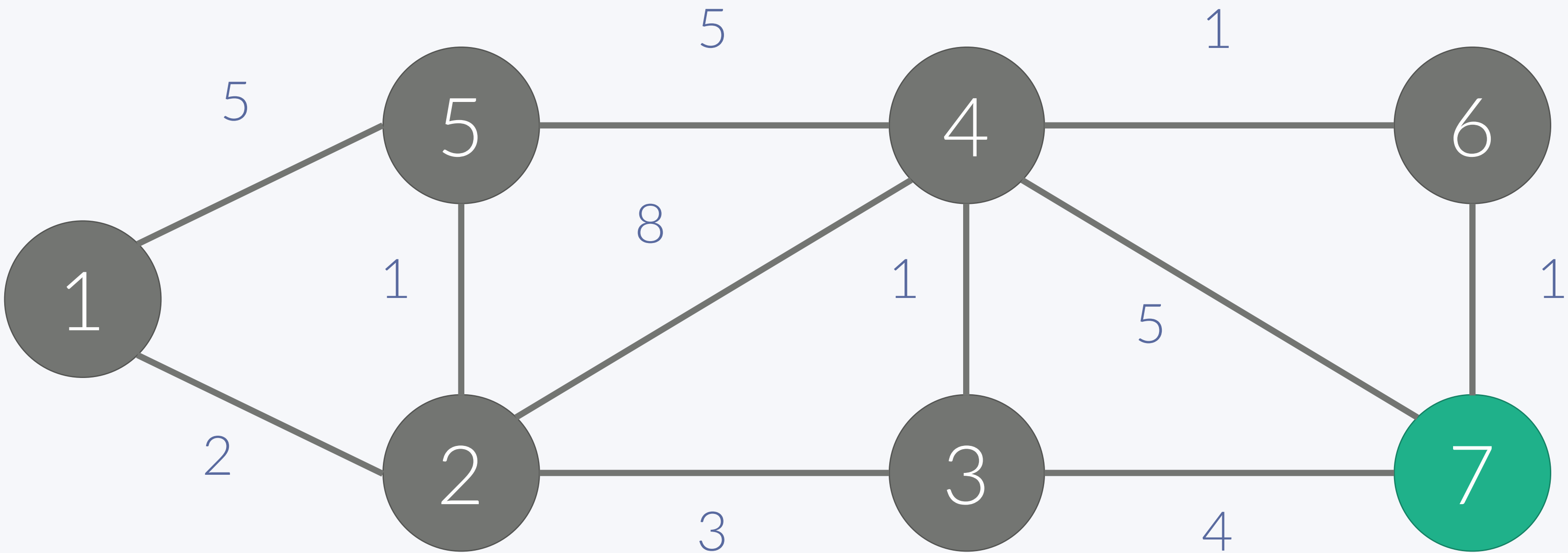


# 다익스트라

Dijkstra Algorithm

- 선택: 7

i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	8
C[i]	1	1	1	1	1	1	1

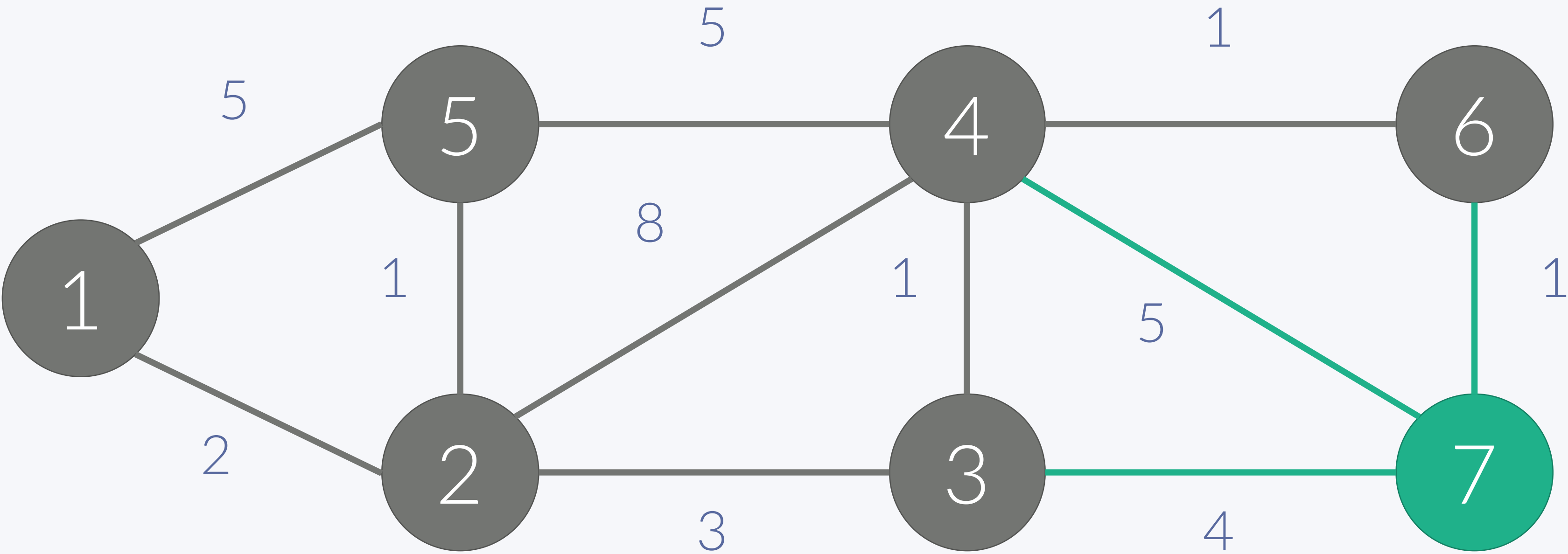


# 다익스트라

Dijkstra Algorithm

- 선택: 7

i	1	2	3	4	5	6	7
D[i]	0	2	5	6	3	7	8
C[i]	1	1	1	1	1	1	1



# 다익스트라

Dijkstra Algorithm

123

- 시간 복잡도
- 인접 행렬:  $O(V^2)$
- 인접 리스트:  $O(V^2)$

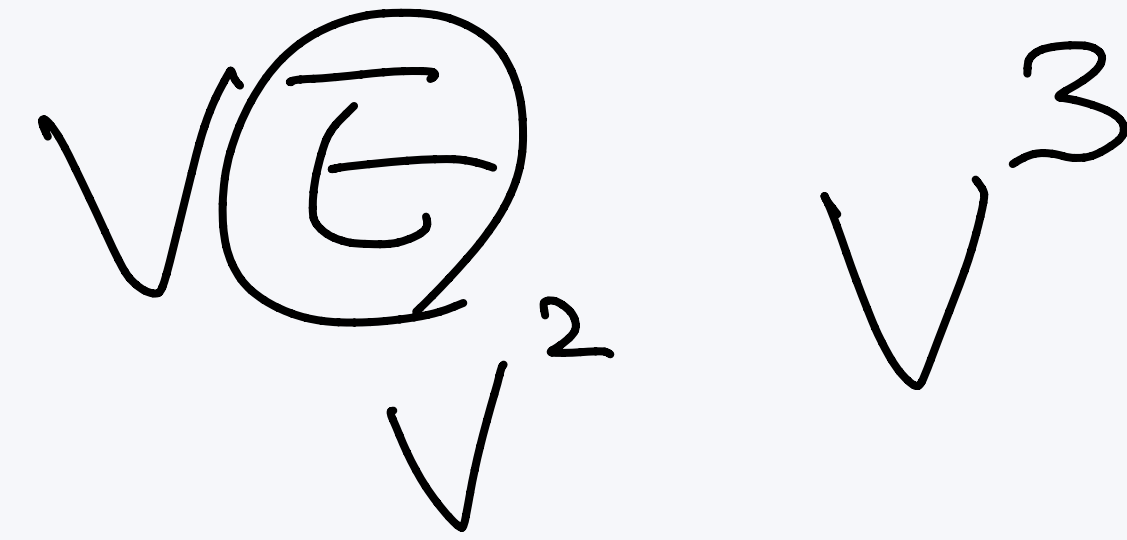
# 최소비용 구하기

<https://www.acmicpc.net/problem/1916>

- A에서 B로 가는 최단 경로

# 최소비용 구하기

<https://www.acmicpc.net/problem/1916>



Handwritten mathematical expressions:  $\sqrt{E}$ ,  $\sqrt{^2}$ , and  $\sqrt[3]{}$ .

- 인접 행렬
- C++: <https://gist.github.com/Baekjoon/910818e0a3aa4244f989>
- Java: <https://gist.github.com/Baekjoon/742dbb011c0a03fcaae4>
- 인접 리스트
- C++: <https://gist.github.com/Baekjoon/c19590f898e323683648>
- Java: <https://gist.github.com/Baekjoon/dead4e8e294f79c95e1c>

# 최소비용 구하기 2

<https://www.acmicpc.net/problem/11779>

- A에서 B로 가는 최단 경로
- 최단 경로도 구해야 함
- distance 값이 바뀔 때, 어디에서 왔는지를 저장해야 함

```
if (d[y] > d[x] + a[x][i].cost) {  
    d[y] = d[x] + a[x][i].cost;  
    v[y] = x;  
}
```

# 최소비용 구하기 2

<https://www.acmicpc.net/problem/11779>

- 다시 정답을 찾는 것은 재귀 호출이나 스택을 이용해서 구할 수 있다.
- 도착 -> 출발 과정을 거쳐서

```
stack<int> st;
int x = end;
while (x != -1) {
    st.push(x);
    x = v[x];
}
printf("%d\n",st.size());
while (!st.empty()) {
    printf("%d ",st.top());
    st.pop();
}
```

# 최소비용 구하기 2

128

<https://www.acmicpc.net/problem/11779>

- C/C++: <https://gist.github.com/Baekjoon/3094b9da5667d56e580a>



# 특정한 최단 경로

<https://www.acmicpc.net/problem/1504>

- 1 -> N으로 가는데, V1, V2를 꼭 거쳐서 가는 최단 경로
- 가능한 경우 2가지
- 1 -> V1 -> V2 -> N
- 1 -> V2 -> V1 -> N
- 다익스트라의 시작점을 1, V1, V2로 총 3번 알고리즘을 수행한 다음에 답을 구할 수 있다.

# 특정한 최단 경로

130

<https://www.acmicpc.net/problem/1504>

- C++: <https://gist.github.com/Baekjoon/1c8872f8eca44ee97e95>
- Java: <https://gist.github.com/Baekjoon/b8c102094d8c00da95c9>

# 최단 경로

<https://www.acmicpc.net/problem/1753>

- 다익스트라를 이용해서 최단 경로를 구해야 하는데
  - $1 \leq V \leq 20,000$  이기 때문에
  - $O(V^2)$ 이나  $O(VE)$ 는 시간이 너무 오래걸린다
  - 인접행렬도 만들 수가 없다.
- 
- 다익스트라에서 시간을 줄일 수 있는 부분은 어디일까?

# 최단 경로

132

<https://www.acmicpc.net/problem/1753>

1. 체크되어 있지 않은 정점 중에서 D의 값이 가장 작은 정점  $x$ 를 선택한다.
2.  $x$ 를 체크한다.
3.  $x$ 와 연결된 모든 정점을 검사한다.
  - 간선을  $(x, y, z)$ 라고 했을 때
  - $d[y] > d[x] + z$  이면 갱신해준다.

$O()$   
상상 X

$E \log E$

$E \log V$

- 1, 2, 3단계를 모든 정점을 체크할 때까지 계속한다.

정점 1000개      간선 10  
1000개      1000개

# 최단 경로

<https://www.acmicpc.net/problem/1753>

1. 체크되어 있지 않은 정점 중에서 D의 값이 가장 작은 정점  $x$ 를 선택한다.
  2.  $x$ 를 체크한다.
  3.  $x$ 와 연결된 모든 정점을 검사한다.
    - 간선을  $(x, y, z)$ 라고 했을 때
    - $d[y] > d[x] + z$  이면 갱신해준다.
- 1, 2, 3단계를 모든 정점을 체크할 때까지 계속한다.

# 최단 경로

134

<https://www.acmicpc.net/problem/1753>

- 힙을 사용해서 최소값을  $O(\lg E)$ 만에 찾을 수 있다

# 최단 경로

135

<https://www.acmicpc.net/problem/1753>

- C++: <https://gist.github.com/Baekjoon/653eceb179f8fdc284bf>

$$V > 4$$

$$E = V^2$$

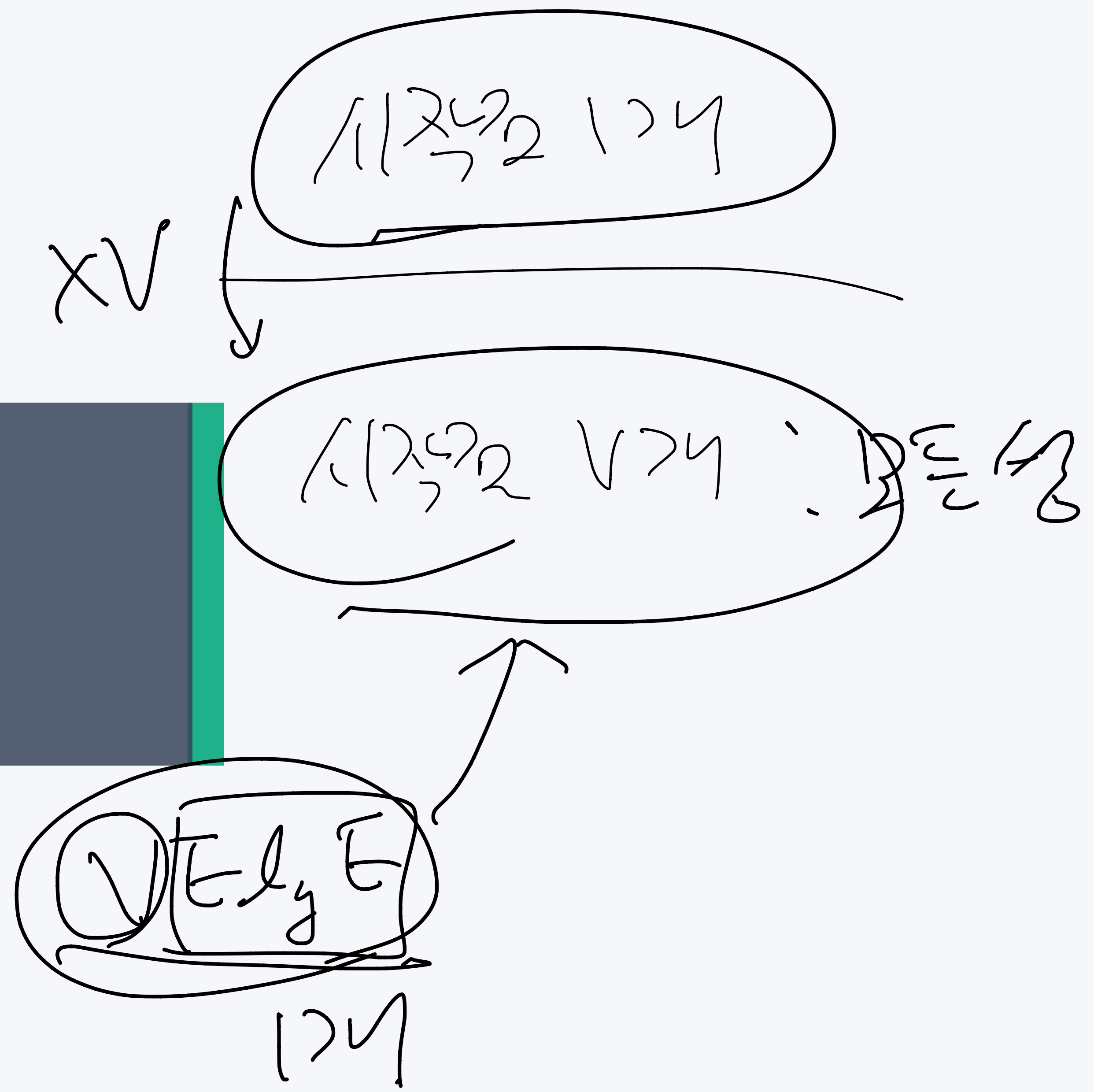
다른 스타일

$$O(V^2)$$

↓

$$O(E \lg t)$$
$$V^2 \lg V^2$$

# 플로이드



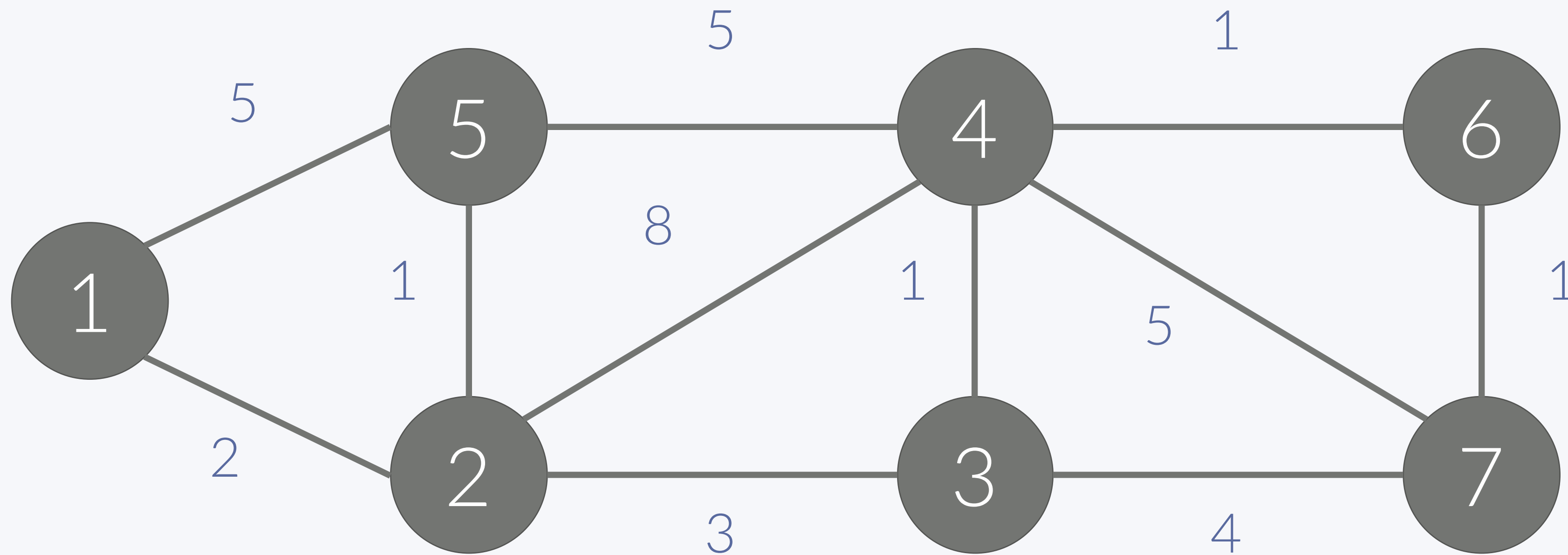


# 플로이드

## Floyd-Warshall Algorithm

137

- 모든 쌍의 최단 경로를 구하는 알고리즘

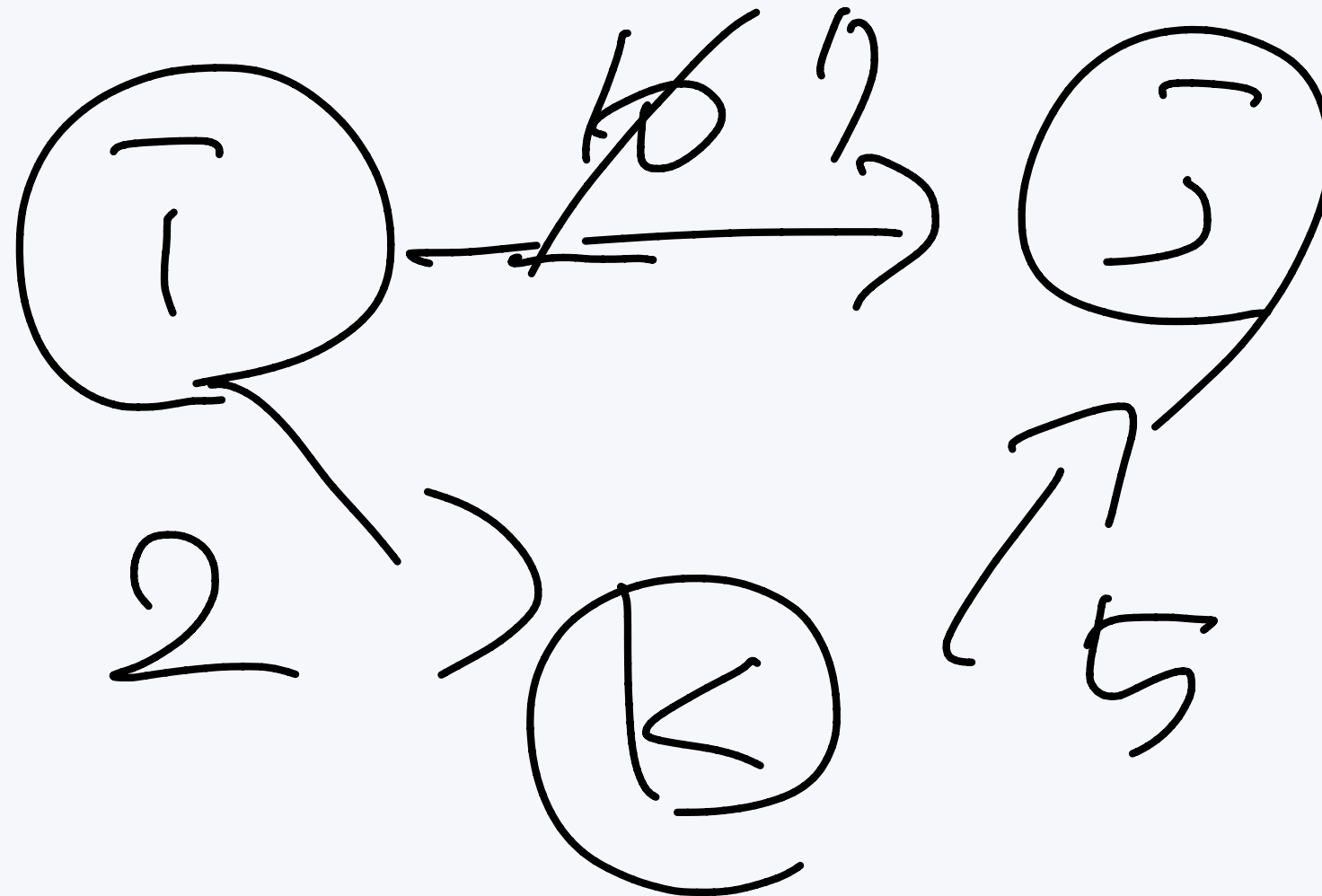


# 플로이드

Floyd-Warshall Algorithm ✓

```
for (int k=1; k<=n; k++) {  
    for (int i=1; i<=n; i++) {  
        for (int j=1; j<=n; j++) {  
            if (d[i][j] > d[i][k] + d[k][j]) {  
                d[i][j] = d[i][k] + d[k][j];  
            }  
        }  
    }  
}
```

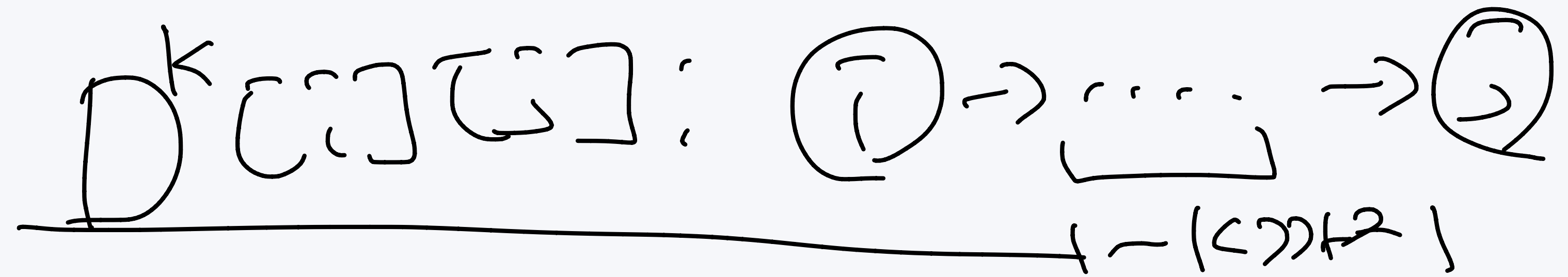
$n^3$   
 $\sqrt{3}$



# 플로이드

Floyd-Warshall Algorithm

139



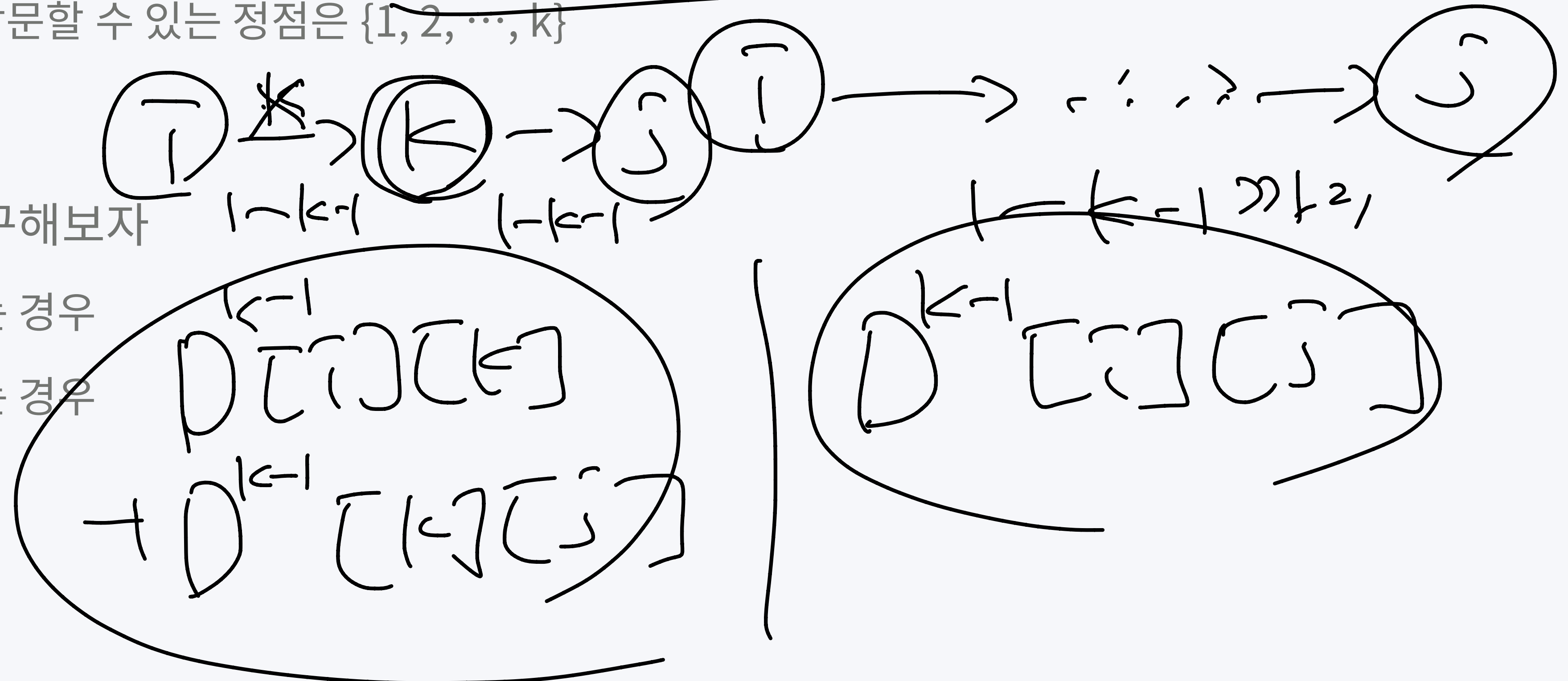
- 1~N 까지 정점이 있을 때
- $d[k][i][j]$ 를 다음과 같이 정의
  - $i \rightarrow j$ 로 이동하는 최단 경로
  - 이 때, 중간에 방문할 수 있는 정점은  $\{1, 2, \dots, k\}$

① 경로의  $k$ 가 있는 경우

② 경로의  $k$ 가 없는 경우

- 그럼  $d[k][i][j]$ 를 구해보자

- $k$ 가 경로에 없는 경우
- $k$ 가 경로에 있는 경우



# 플로이드

140

## Floyd-Warshall Algorithm

- 1~N 까지 정점이 있을 때
- $d[k][i][j]$ 를 다음과 같이 정의
  - $i \rightarrow j$ 로 이동하는 최단 경로
  - 이 때, 중간에 방문할 수 있는 정점은  $\{1, 2, \dots, k\}$
- 그럼  $d[k][i][j]$ 를 구해보자
  - k가 경로에 없는 경우
    - $d[k-1][i][j]$
  - k가 경로에 있는 경우
    - $d[k-1][i][k] + d[k-1][k][j]$

# 플로이드

## Floyd-Warshall Algorithm

141

- $d[k][i][j] = a[i][j]$  ( $k == 0$ )
- $\min(d[k-1][i][j], d[k-1][i][k] + d[k-1][k][j])$  ( $k \geq 1$ )

# 플로이드

142

## Floyd-Warshall Algorithm

- 구현할 때는 2차원 배열로 구현하면 된다

$$D^{k-1} \Rightarrow D^k$$

- `for (int k=1; k<=n; k++) { d[k-1]을 이용해 d[k]를 구한다`

# 플로이드

143

Floyd-Warshall Algorithm

```
for (int k=1; k<=n; k++) {  
    for (int i=1; i<=n; i++) {  
        for (int j=1; j<=n; j++) {  
            if (d[i][j] > d[i][k] + d[k][j]) {  
                d[i][j] = d[i][k] + d[k][j];  
            }  
        }  
    }  
}
```

$V \leq 100$

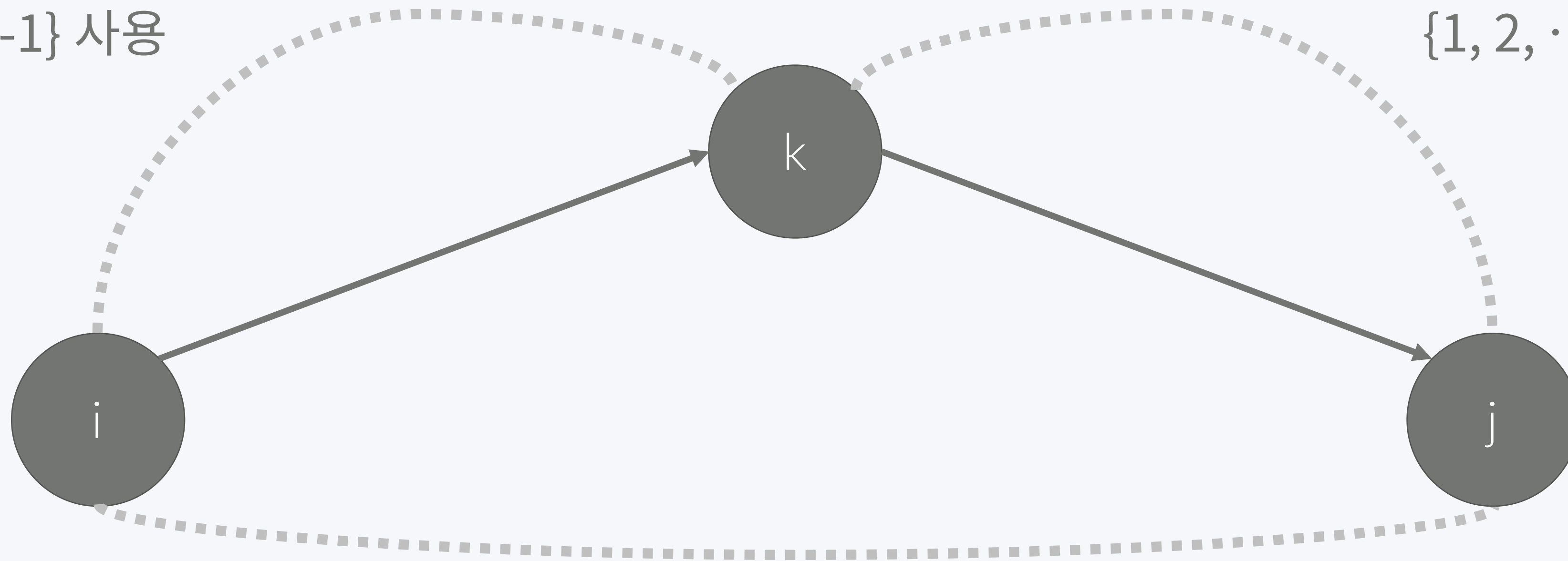
# 플로이드

144

## Floyd-Warshall Algorithm

- $d[k][i][j] = a[i][j]$  ( $k == 0$ )
- $\min(d[k-1][i][j], d[k-1][i][k] + d[k-1][k][j])$  ( $k \geq 1$ )

{1, 2, ..., k-1} 사용



{1, 2, ..., k-1} 사용



# 경로 찾기

145

<https://www.acmicpc.net/problem/11403>

- 가중치 없는 방향 그래프  $G$ 가 주어졌을 때, 모든 정점  $(i, j)$ 에 대해서,  $i$ 에서  $j$ 로 가는 경로가 있는지 없는지 구하는 프로그램을 작성하시오.

# 경로 찾기

146

<https://www.acmicpc.net/problem/11403>

- C/C++: <https://gist.github.com/Baekjoon/e25468e15712435e1dd7>
- Java: <https://gist.github.com/Baekjoon/d4d31e9ad6e13a731d50>

# 플로이드

147

<https://www.acmicpc.net/problem/11404>

- $n(1 \leq n \leq 100)$ 개의 도시가 있다
- 한 도시에서 출발하여 다른 도시에 도착하는  $m(1 \leq m \leq 100,000)$ 개의 버스가 있다
- 각 버스는 한 번 사용할 때 필요한 비용이 있다
- 모든 도시의 쌍  $(A, B)$ 에 대해서 도시 A에서 B로 가는데 필요한 비용의 최소값을 구하는 프로그램을 작성하시오

# 플로이드

148

<https://www.acmicpc.net/problem/11404>

- C/C++: <https://gist.github.com/Baekjoon/29e8d9d76246ccbffdcd>
- Java: <https://gist.github.com/Baekjoon/d4d31e9ad6e13a731d50>

# 플로이드 2

<https://www.acmicpc.net/problem/11780>

- 플로이드 구현을 보면
- $i \rightarrow k, k \rightarrow j$ 로 가는 간선을  $i \rightarrow j$ 로 바꿔줬기 때문에
- $i \rightarrow j$  이 원래 어떤 정점을 가르키고 있었는지를 알아야 한다.
- 따라서 다음과 같이 구현을 변경할 수 있다.

# 케빈 베이컨의 6단계 법칙

150

<https://www.acmicpc.net/problem/1389>

- 플로이드 알고리즘을 이용해 모든 쌍의 최단 경로를 구한 다음에 구할 수 있다
- C/C++: <https://gist.github.com/Baekjoon/499233fa40a691cecf0a>

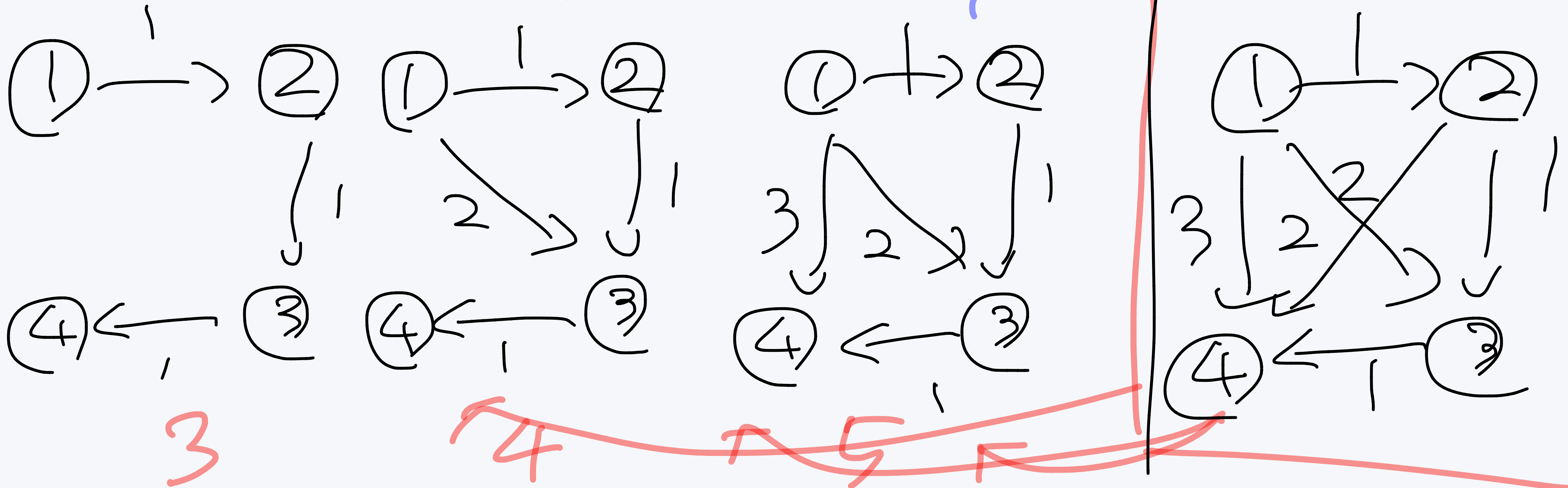
# 궁금한 민호

프로그래밍

151

<https://www.acmicpc.net/problem/1507>

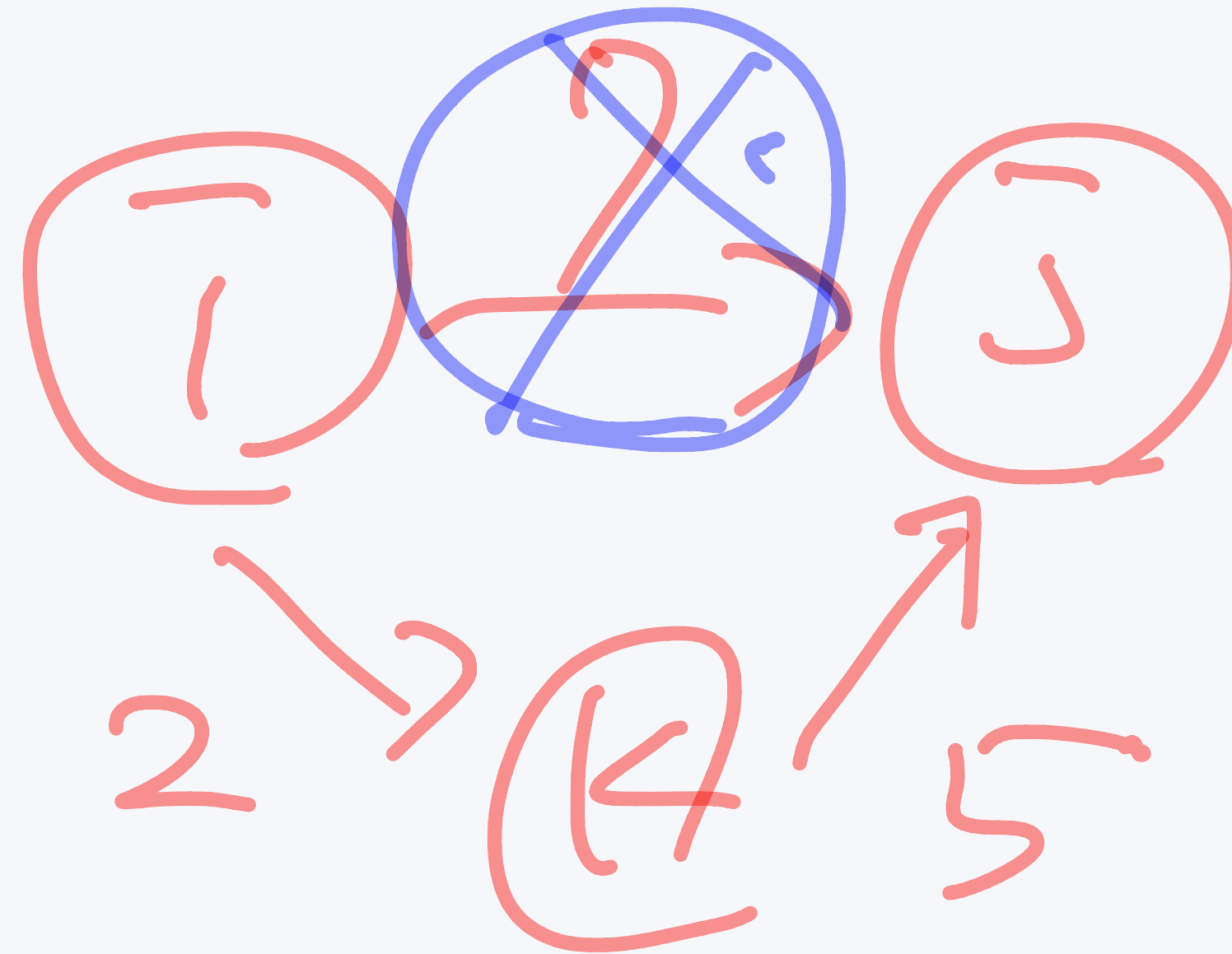
- 강호는 모든 쌍의 도시에 대해서 최소 이동 시간을 구해놓았다. 민호는 이 표를 보고 원래 도로가 몇 개 있는지를 구해보려고 한다.
- 모든 쌍의 도시 사이의 최소 이동 시간이 주어졌을 때, 이 나라에 존재할 수 있는 도로의 개수의 최소값과 그 때, 모든 도로의 시간의 합을 구하는 프로그램을 작성하시오.



# 궁금한 민호

<https://www.acmicpc.net/problem/1507>

- A에서 B로 가는 비용이  $x$  일 때
- A->C->B로 가는 비용이  $x$ 이면
- A->B로 가는 도로는 필요가 없다
- 이렇게 모든 도시의 쌍을 보면서 필요없는 도로를 제거





# 궁금한 민호

153

<https://www.acmicpc.net/problem/1507>

- C/C++: <https://gist.github.com/Baekjoon/2db040d34d91663a5c41>
- Java: <https://gist.github.com/Baekjoon/c30a5b1b9254e7dad0c>

# 운동

154

<https://www.acmicpc.net/problem/1956>

- 그래프에서 사이클 길이 중 최소길이를 찾는 문제
- 플로이드를 이용한 다음에  $d[i][i]$ 를 검사하면 된다.

<https://www.acmicpc.net/problem/1956>

- C/C++: <https://gist.github.com/Baekjoon/bda2e8572db7149bb9fd>
- Java: <https://gist.github.com/Baekjoon/3288251c4d2d98301be7>

# SPFA

---

# SPFA

## Shorteest Path Faster Algorithm

- 벨만포드의 성능을 향상시킨 알고리즘
- 최악의 경우에는 벨만 포드의 시간복잡도와 같지만 평균적으로  $O(E)$  이다
- 벨만포드의 아이디어와 같은 아이디어이다

```
for (int j = 0; j < n; j++) {  
    for (int k = 0; k < m; k++) {  
        int from = edges[k].from;  
        int to = edges[k].to;  
        int cost = edges[k].cost;  
        if (Distance[to] > Distance[from] + cost) {  
            Distance[to] = Distance[from] + cost;  
        }  
    }  
}
```

# SPFA

## Shorteest Path Faster Algorithm

- 벨만포드는 모든 간선에 대해서 업데이트를 진행하고
- SPFA는 아래 if문에 의해서 바뀐 정점과 연결된 간선에 대해서만 업데이트를 진행한다.
- 따라서, 인접 리스트의 구현이 필요하다.

```
for (int j = 0; j < n; j++) {  
    for (int k = 0; k < m; k++) {  
        int from = edges[k].from;  
        int to = edges[k].to;  
        int cost = edges[k].cost;  
        if (Distance[to] > Distance[from] + cost) {  
            Distance[to] = Distance[from] + cost;  
        }  
    }  
}
```

# SPFA

## Shorteest Path Faster Algorithm

- 바뀐 정점은 큐를 이용해서 관리하고
- 큐에 들어가있는지, 안 들어가있는지를 배열을 이용해서 체크한다.
- 초기화를 하고, 큐에 시작점을 넣어주고

```
for (int i=1; i<=n; i++) {  
    d[i] = inf;  
}  
d[1] = 0;  
queue<int> q;  
q.push(1);  
c[1] = true;
```

# SPFA

160

Shorteest Path Faster Algorithm

```
while (!q.empty()) {
    int from = q.front();
    c[from] = false; q.pop();
    for (Edge &e : a[from]) {
        int to = e.to, cost = e.cost;
        if (d[to] > d[from] + cost) {
            d[to] = d[from] + cost;
            if (c[to] == false) {
                q.push(to);
                c[to] = true;
            }
        }
    }
}
```



# 타임머신

161

<https://www.acmicpc.net/problem/11657>

- C++: <https://gist.github.com/Baekjoon/f50cd9501baec1064eec>