Shortest Path Problem

Dijkstra, Bellman-Ford and Floyd-Warshall

2021.01.31 10:00 KST

Shortest Path Problem

| Graph 상의 두 vertice 간 최단 거리 구하기

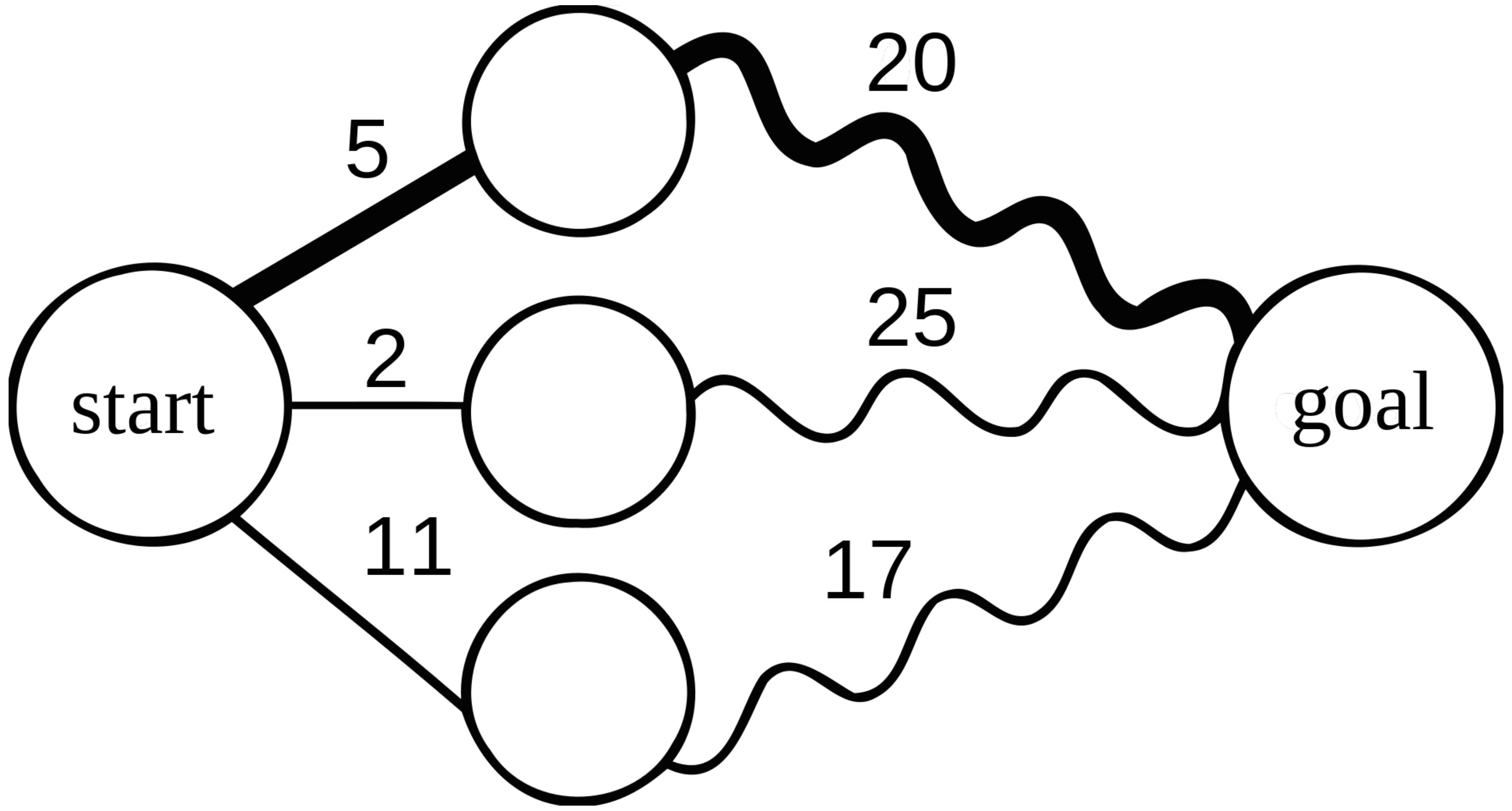
➡ Single-source Shortest Path

➡ Single-destination Shortest Path (= Single-source Shortest Path)

➡ All-pairs Shortest Path

Optimal Substructure

A problem is said to have **optimal substructure**
if an optimal solution can be constructed from optimal solutions of its subproblems



$$u \overset{p_{ux}}{\rightsquigarrow} x \overset{p_{xy}}{\rightsquigarrow} y \overset{p_{yv}}{\rightsquigarrow} v$$

Suppose this path p is a shortest path from u to v.

Then $\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.

Now suppose there exists a shorter path $x \overset{p'_{xy}}{\rightsquigarrow} y$.

Then $w(p'_{xy}) < w(p_{xy})$.

Construct p':

$$u \overset{p_{ux}}{\rightsquigarrow} x \overset{p'_{xy}}{\rightsquigarrow} y \overset{p_{yv}}{\rightsquigarrow} v$$

Then

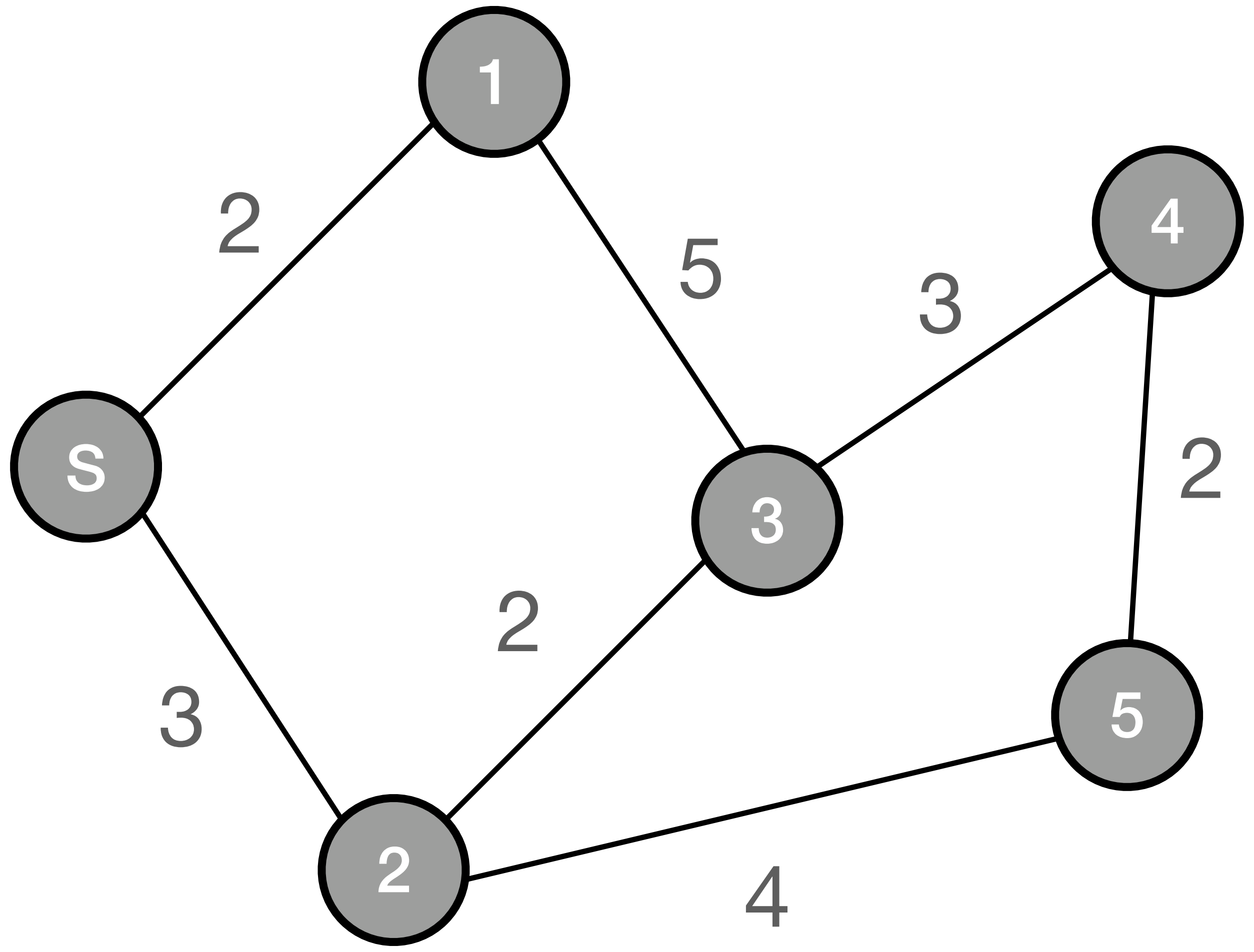
$$\begin{aligned} w(p') &= w(p_{ux}) + w(p'_{xy}) + w(p_{yv}) \\ &< w(p_{ux}) + w(p_{xy}) + w(p_{yv}) \\ &= w(p)\,. \end{aligned}$$

Contradicts the assumption that p is a shortest path.

Dijkstra Algorithm

Single-source shortest path

G



dist[ ]

S	1	2	3	4	5
0	INF	INF	INF	INF	INF

시작점 src로부터 각 node까지의 최단거리

각 단계마다

1) 도달 가능하고 방문하지 않은 정점 중 dist가 가장 작은 정점 u 선택
2) u에서 도달 가능한 정점 u’에 대해,

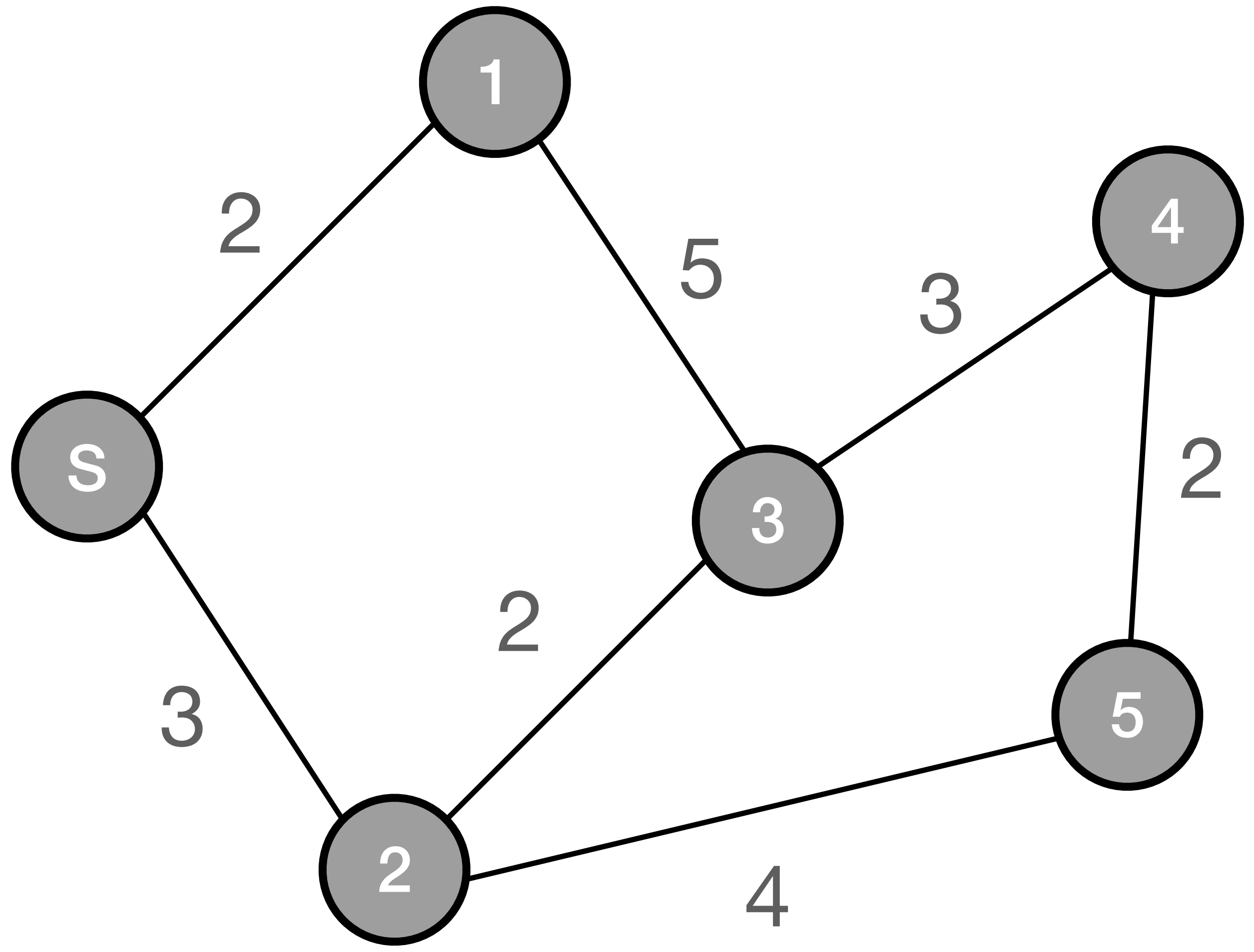
$$dist[u'] = \min(dist[u'], dist[u] + w(u, u'))$$ (edge relaxation)

Why?

➡ 최단거리임이 확정된 정점(Optimal substructure)에서 이어지는 경로만이 최단거리가 될 수 있음

Dijkstra Algorithm

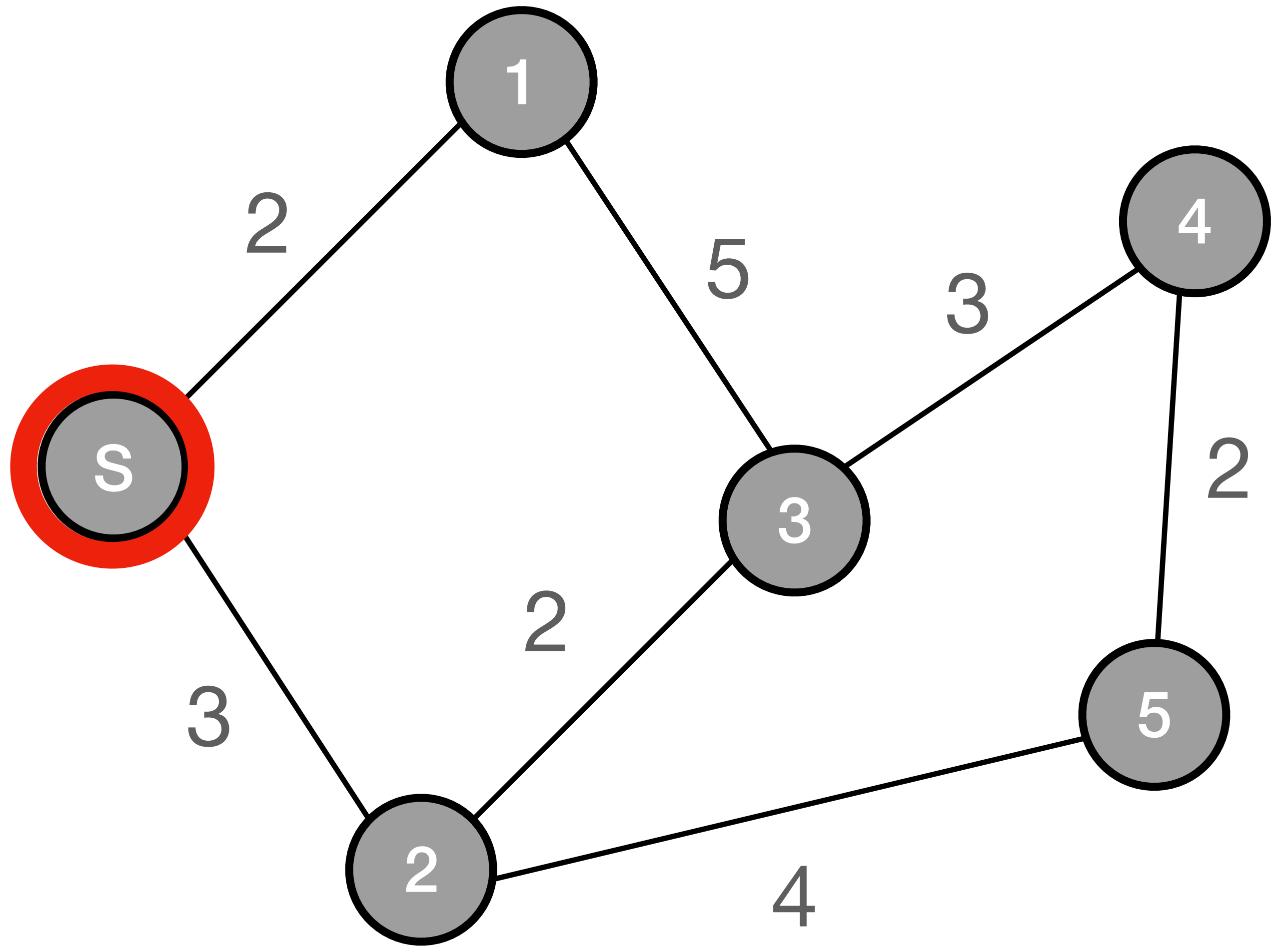
G



dist[ ]	S	1	2	3	4	5
	0	INF	INF	INF	INF	INF

Dijkstra Algorithm

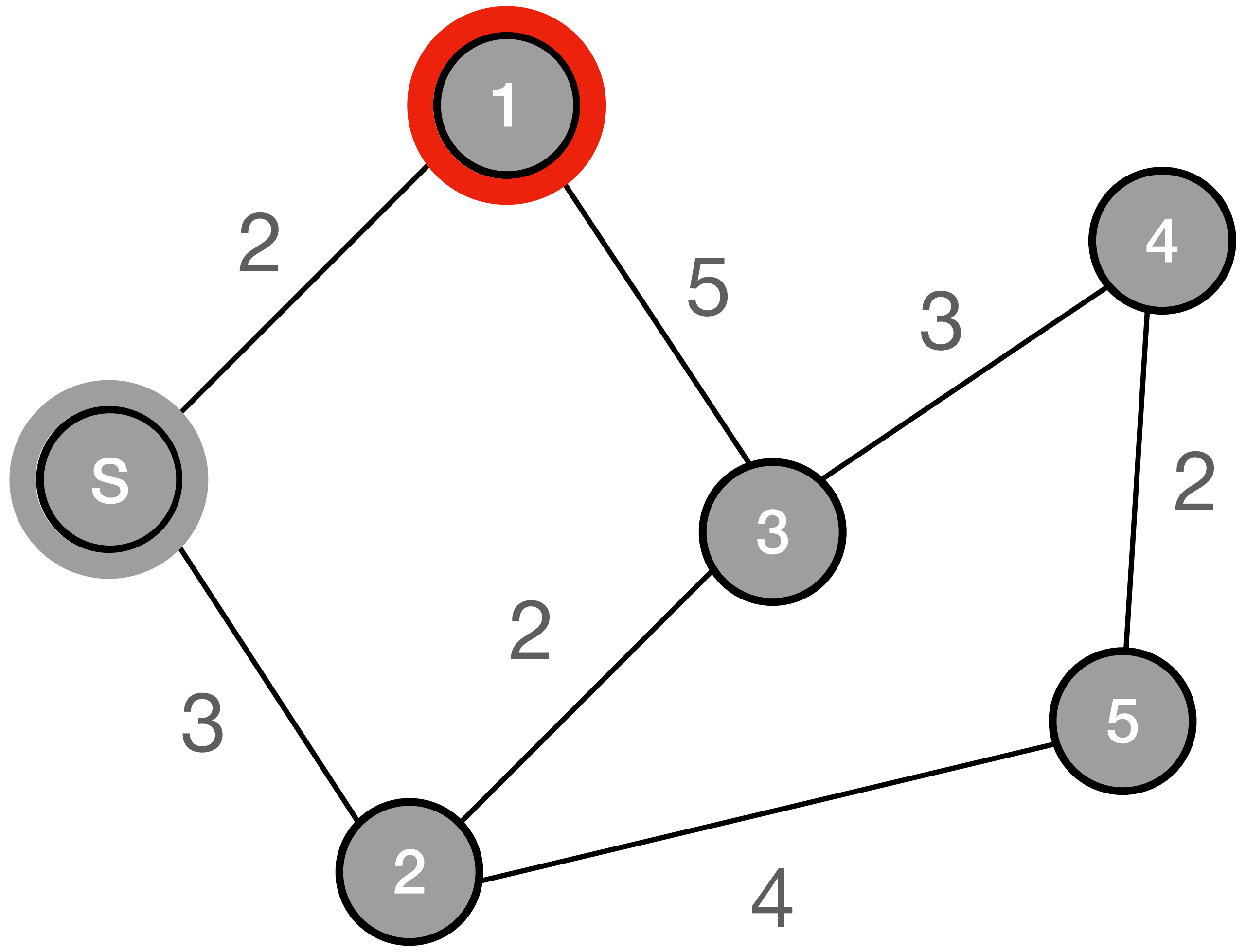
G



dist[ ]	S	1	2	3	4	5
	0	INF	INF	INF	INF	INF
		2	3			

Dijkstra Algorithm

G

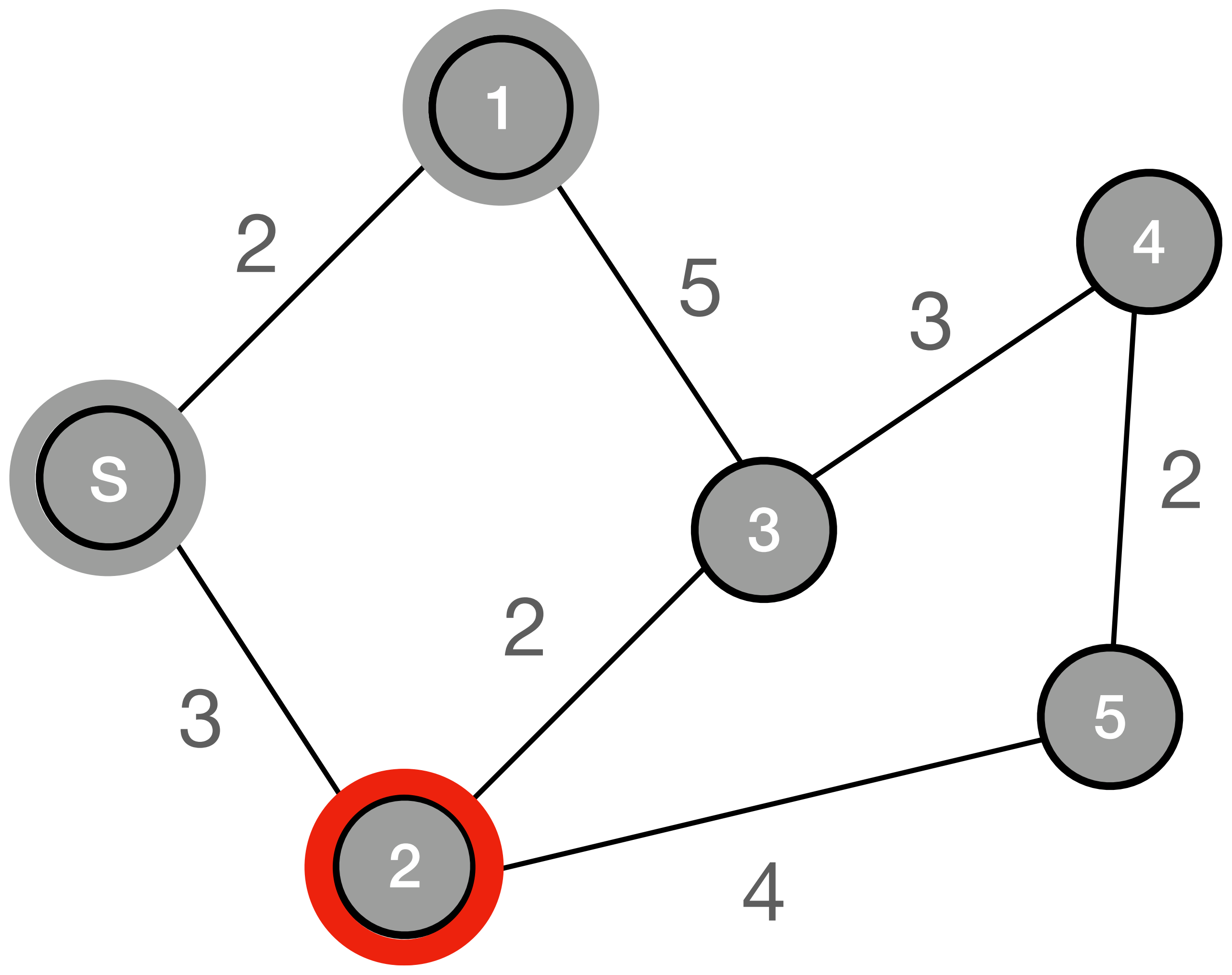


dist[ ]

S	1	2	3	4	5
0	2	3	INF	INF	INF
			7		

Dijkstra Algorithm

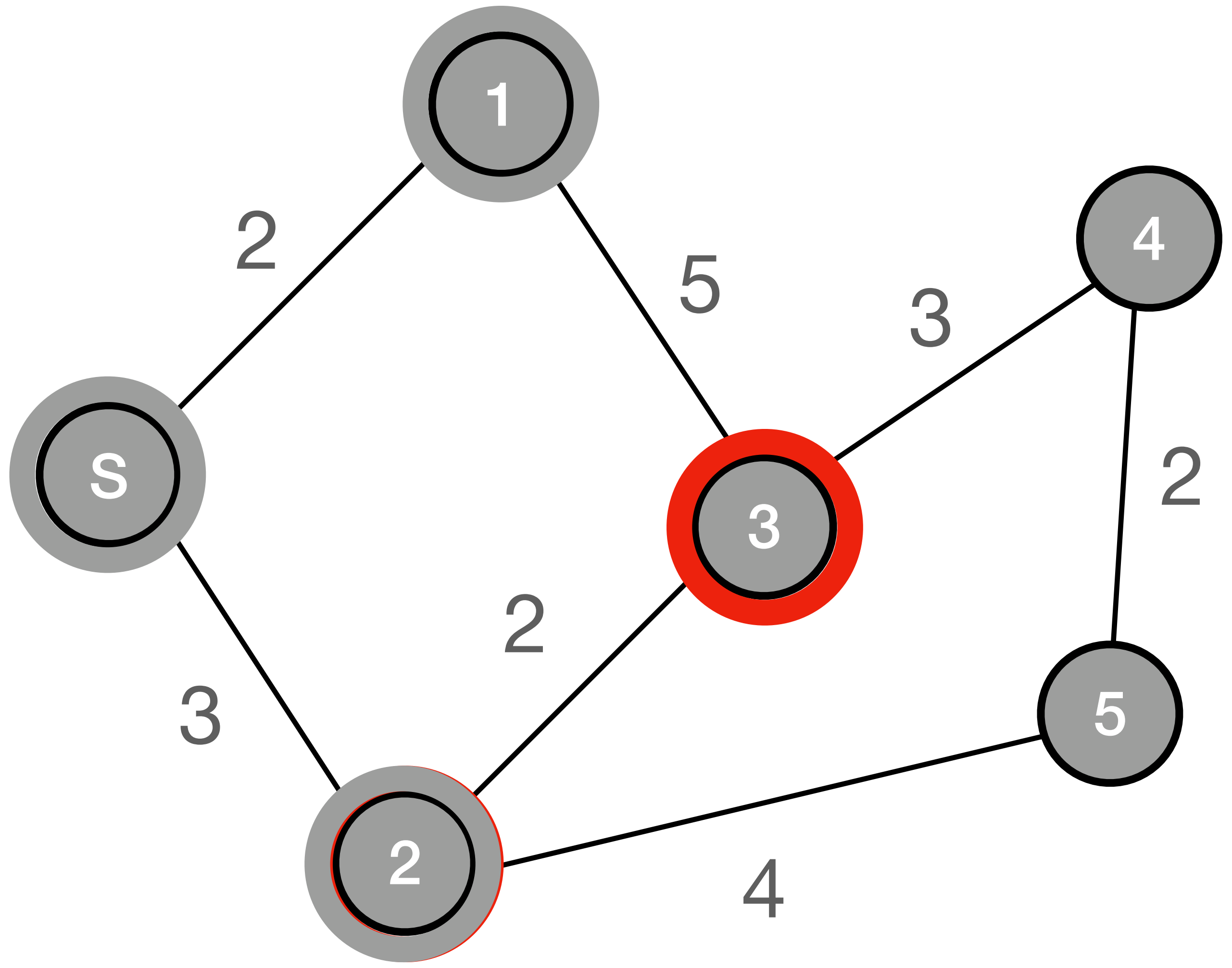
G



dist[ ]	S	1	2	3	4	5
	0	2	3	7	INF	INF
				5		7

Dijkstra Algorithm

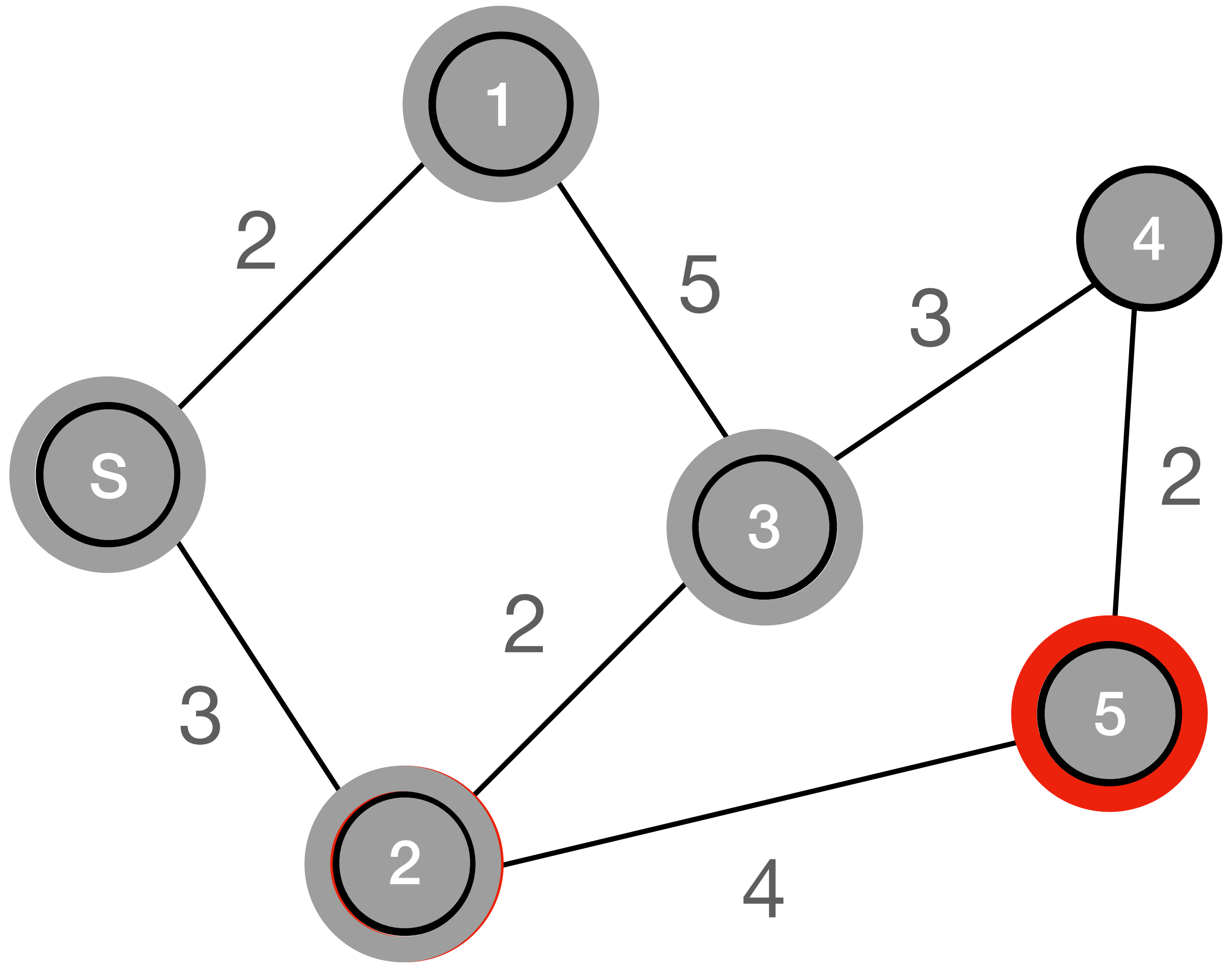
G



dist[ ]	S	1	2	3	4	5
	0	2	3	5	INF	7
					8	

Dijkstra Algorithm

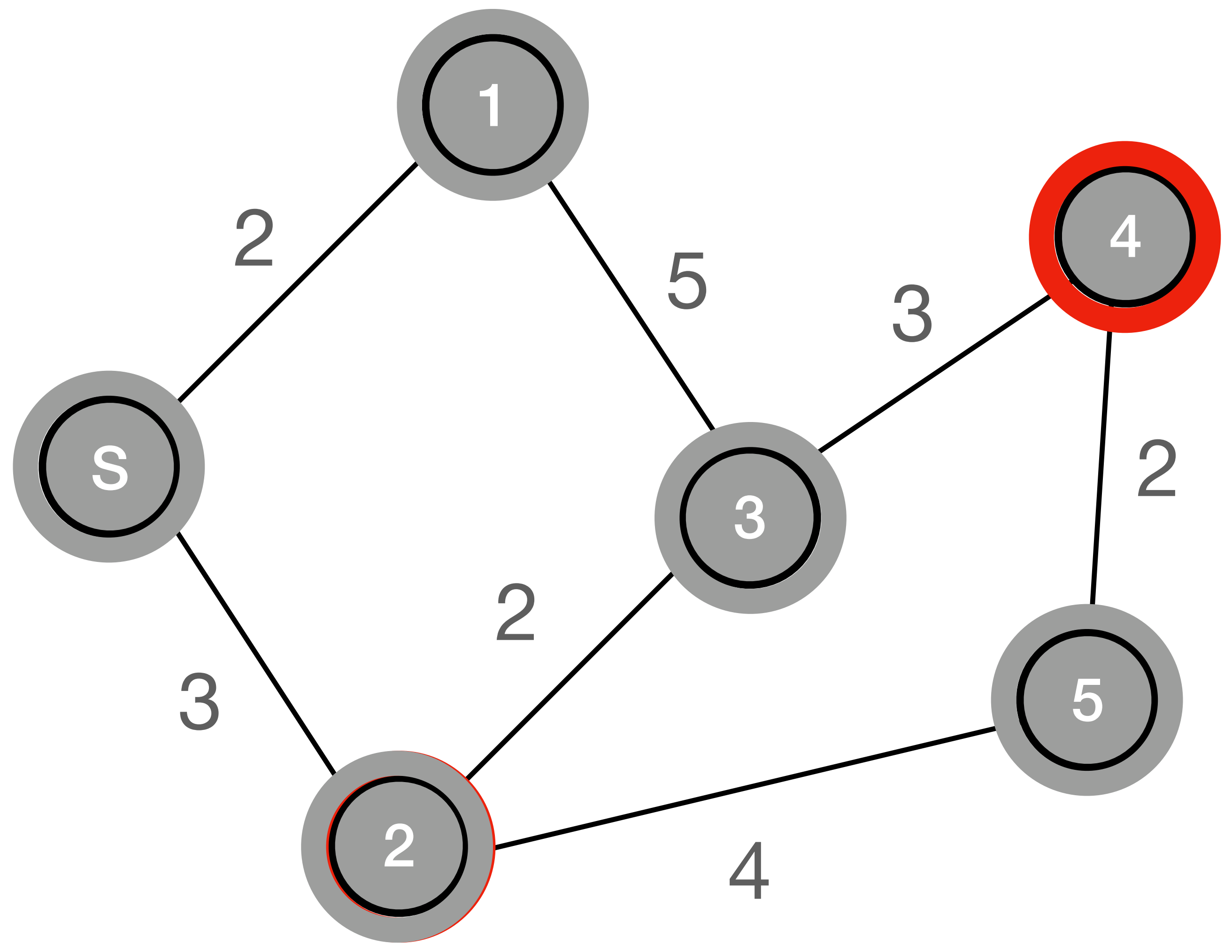
G



	S	1	2	3	4	5
dist[ ]	0	2	3	5	8	7

Dijkstra Algorithm

G



dist[ ]	S	1	2	3	4	5
	0	2	3	5	8	7

Dijkstra Algorithm

Time complexity

Naive approach

➡ 매 단계 dist가 가장 작은 곳 찾기: $O(|V|^2)$

edge relaxation: $O(|E|)$

총 $O(|V|^2 + |E|)$

Optimization

➡ 매 단계 dist가 가장 작은 곳 찾기: 적절한 자료구조를 사용해 $O(|V| log |V|)$

(피보나치 힙이라는 걸 들어는 봤지만 뭔진 잘 모름…그냥 우선순위 큐 씁시다)

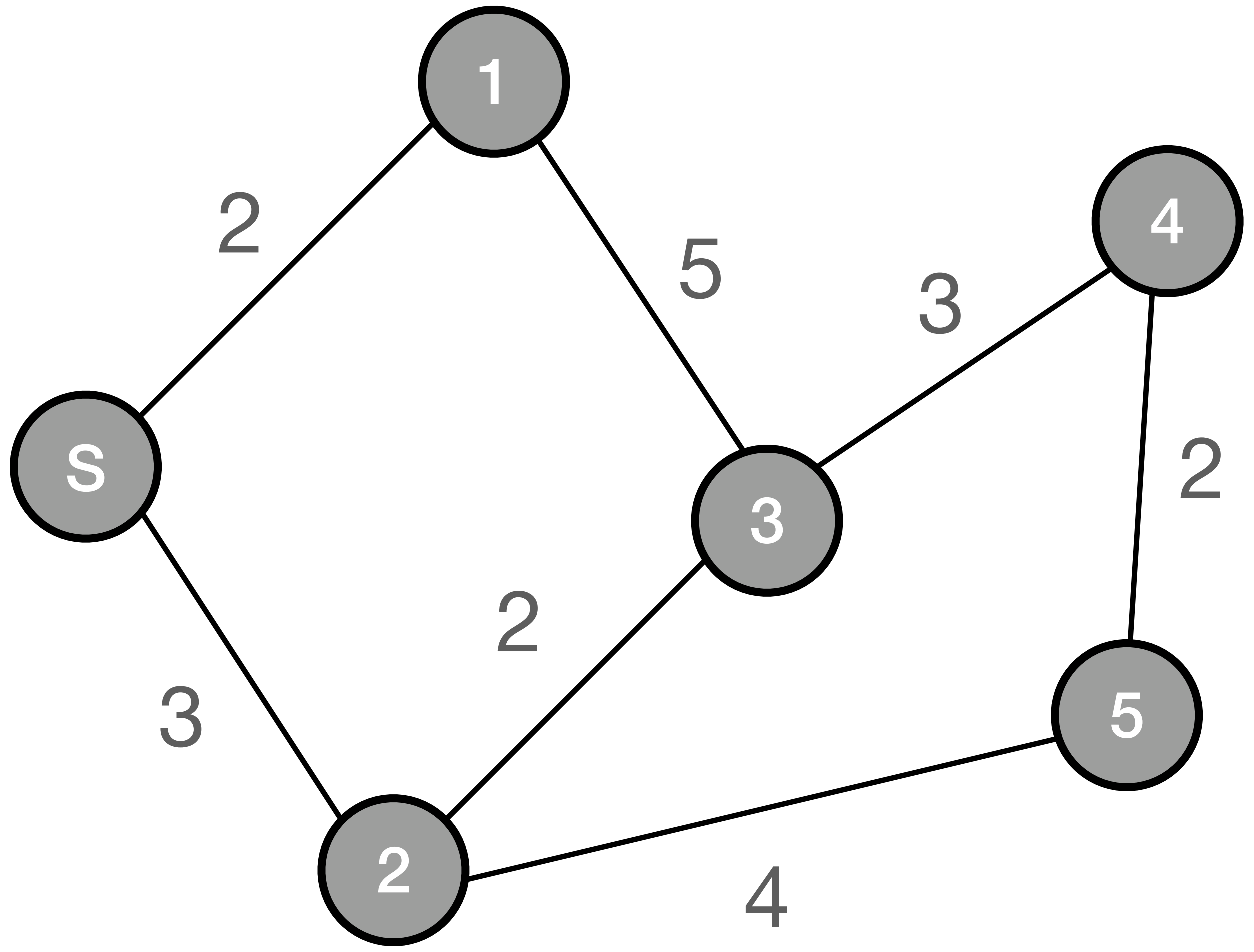
edge relaxation: $O(|E|)$

총 $O(|V| log |V| + |E|)$

Dijkstra Algorithm

Single-source shortest path

G



dist[ ]

S	1	2	3	4	5
0	INF	INF	INF	INF	INF

시작점 src로부터 각 node까지의 최단거리

각 단계마다

1) 도달 가능하고 방문하지 않은 정점 중 dist가 가장 작은 정점 u 선택
2) u에서 도달 가능한 정점 u'에 대해

$$dist[u'] = \min(dist[u'], dist[u] + w(u, u'))$$ (edge relaxation)

Why?

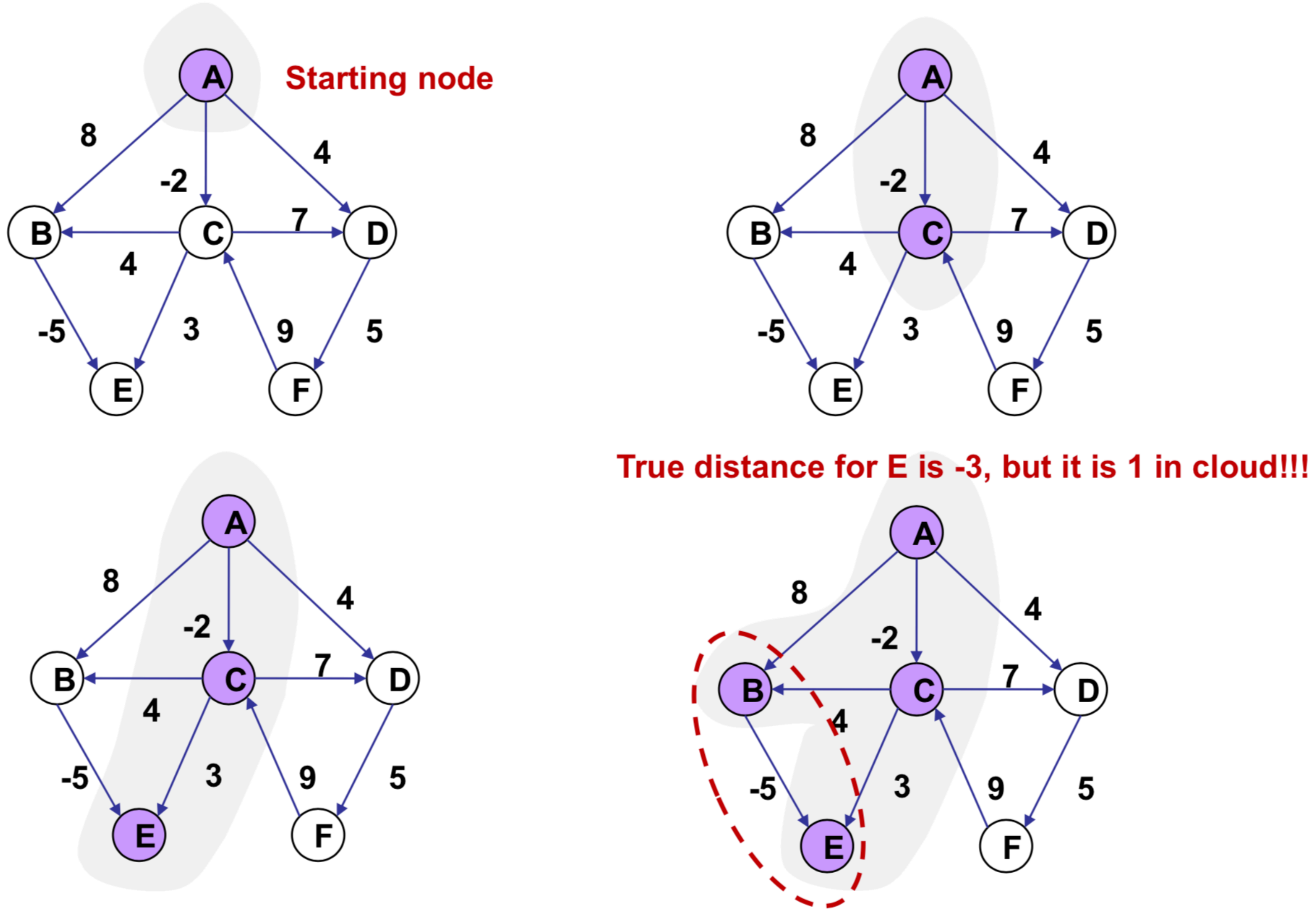
➡ **<u>최단거리임이 확정된 정점(Optimal substructure)</u>**에서 이어지는 경로만이 최단거리가 될 수 있음

Dijkstra Algorithm

단점

‘최단 경로의 부분 경로는 항상 최단 경로’라는 명제는 모든 간선이 음이 아닌 가중치를 가져서
간선이 늘어날 수록 가중치의 합이 항상 증가한다는 것을 전제로 해야 가능함

➡ <u>**음의 가중치를 갖는 간선이 존재할 경우 다익스트라 알고리즘 사용 불가**</u>

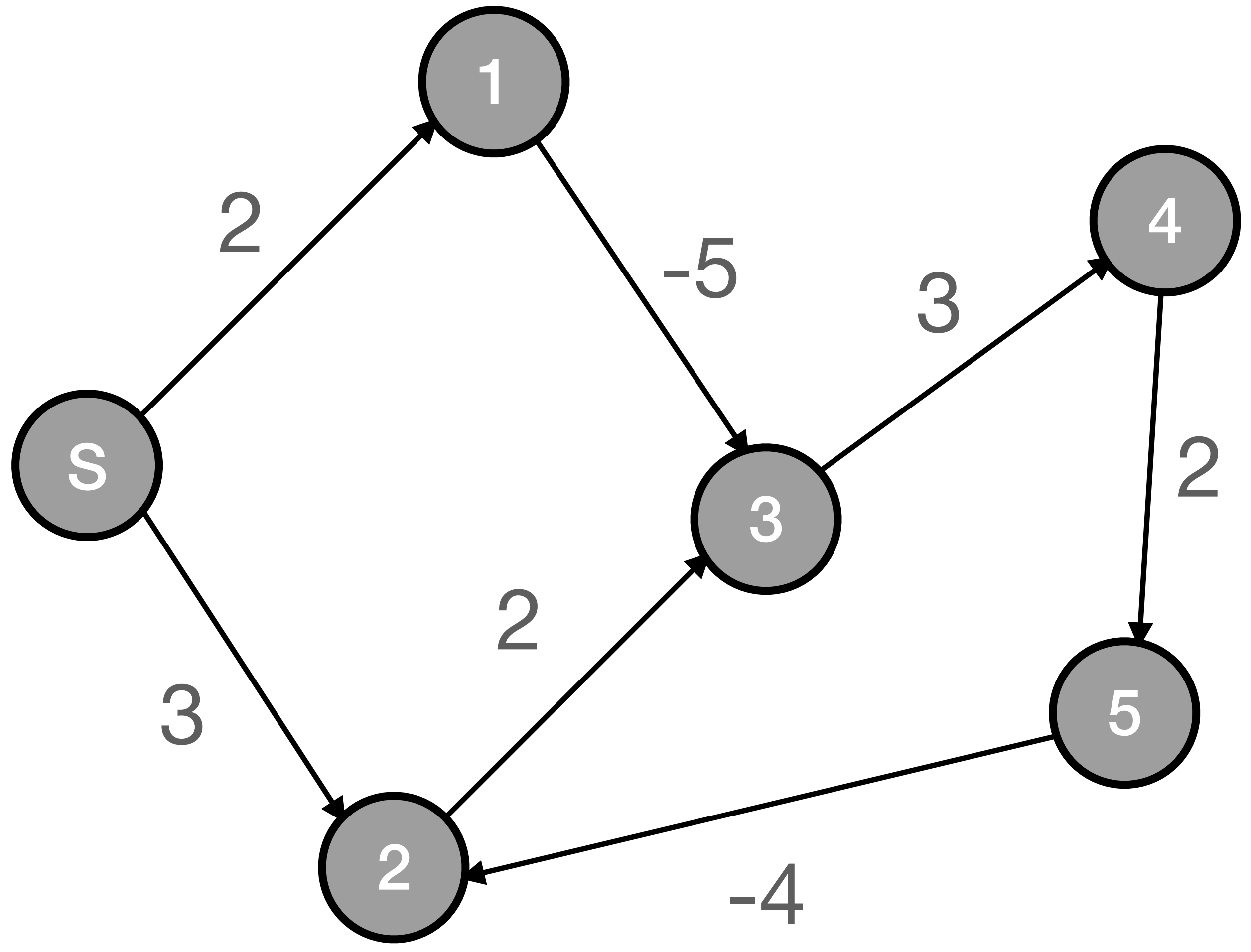


Bellman-Ford Algorithm

Single-source shortest path

\+ 음수 가중치를 갖는 edge도 허용!

G



	S	1	2	3	4	5
dist[ ]	0	INF	INF	INF	INF	INF

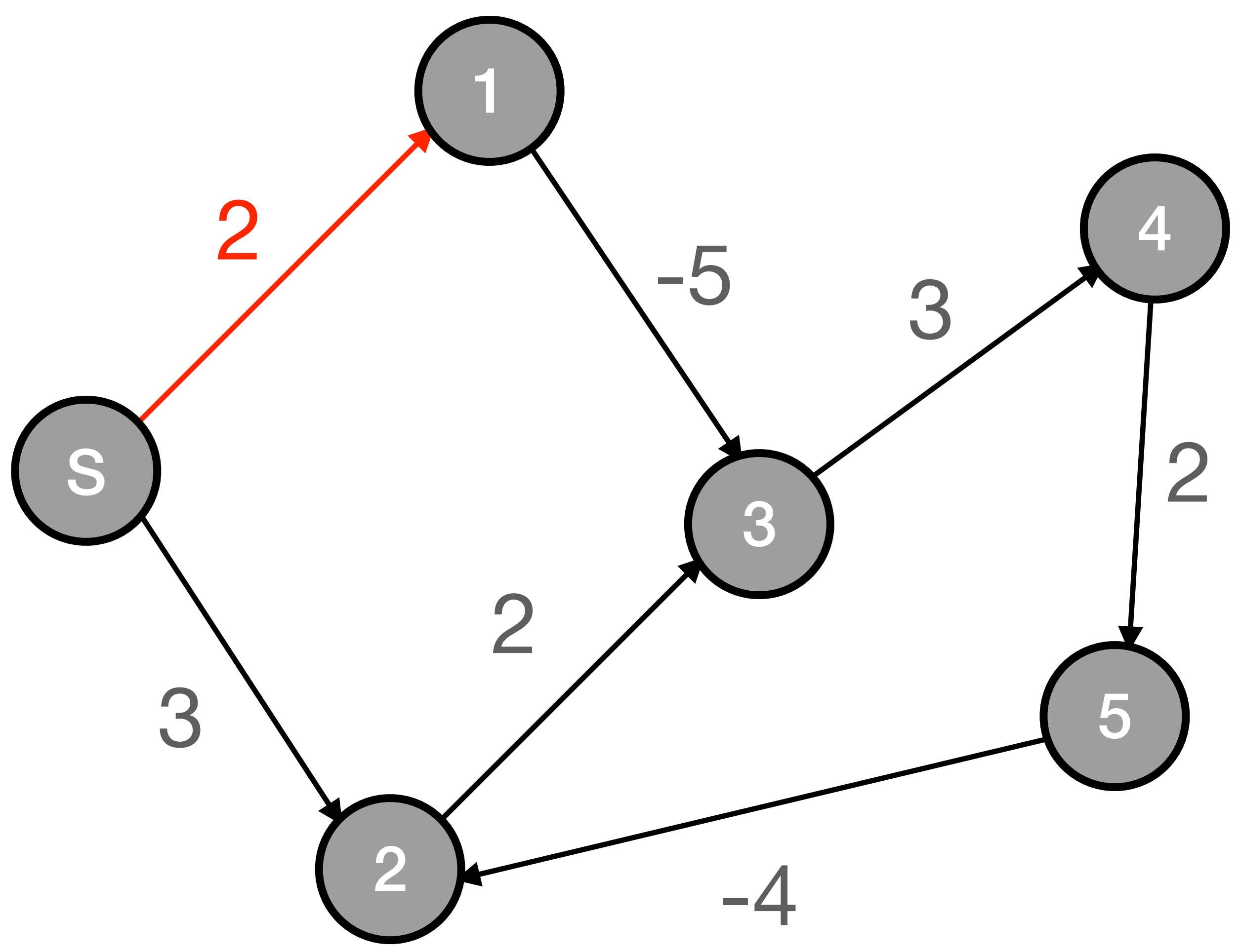
시작점 src로부터 각 node까지의 최단거리

최단 경로를 찾을 때 길이가 1, 2, …, |V|-1인 경로를 모두 체크함

➡ <u>모든 edge를 돌면서 dist를 업데이트</u> 하는 것을 <u>|V|-1</u>번 반복
(edge 순회 순서는 상관 없음)

Bellman-Ford Algorithm

G

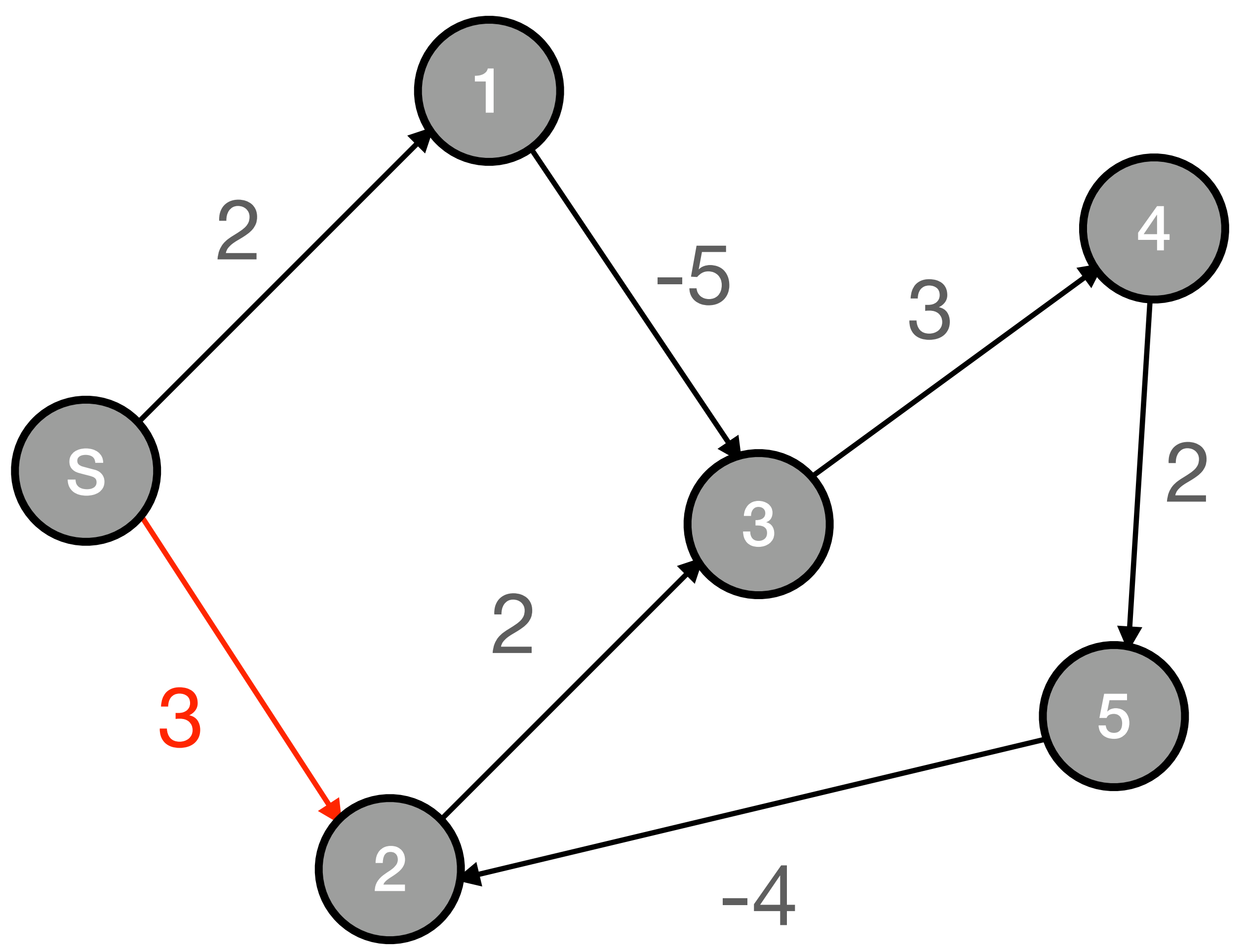


dist[ ]

S	1	2	3	4	5
0	2	INF	INF	INF	INF

Bellman-Ford Algorithm

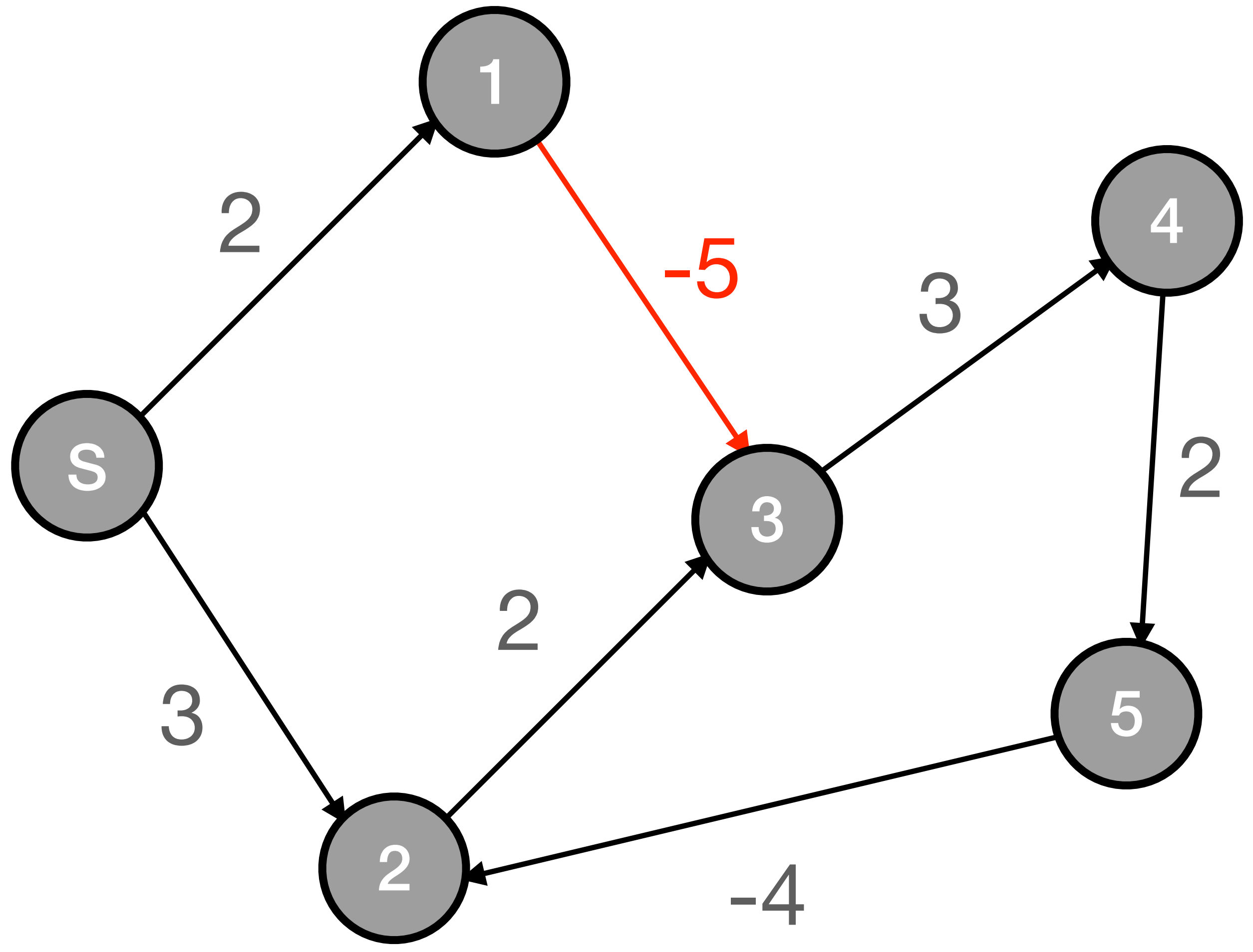
G



dist[ ]	S	1	2	3	4	5
	0	2	3	INF	INF	INF

Bellman-Ford Algorithm

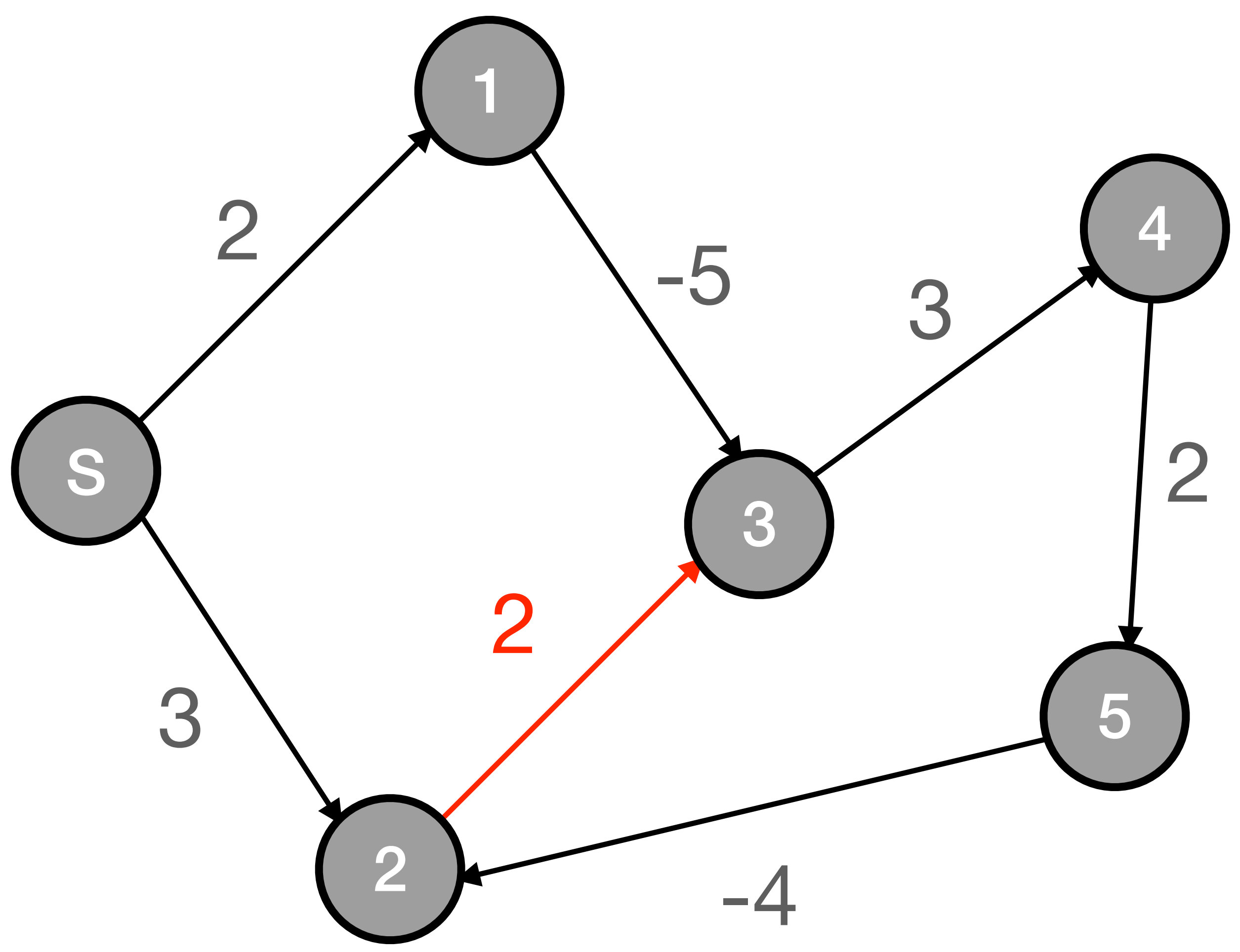
G



dist[ ]	S	1	2	3	4	5
	0	2	3	-3	INF	INF

Bellman-Ford Algorithm

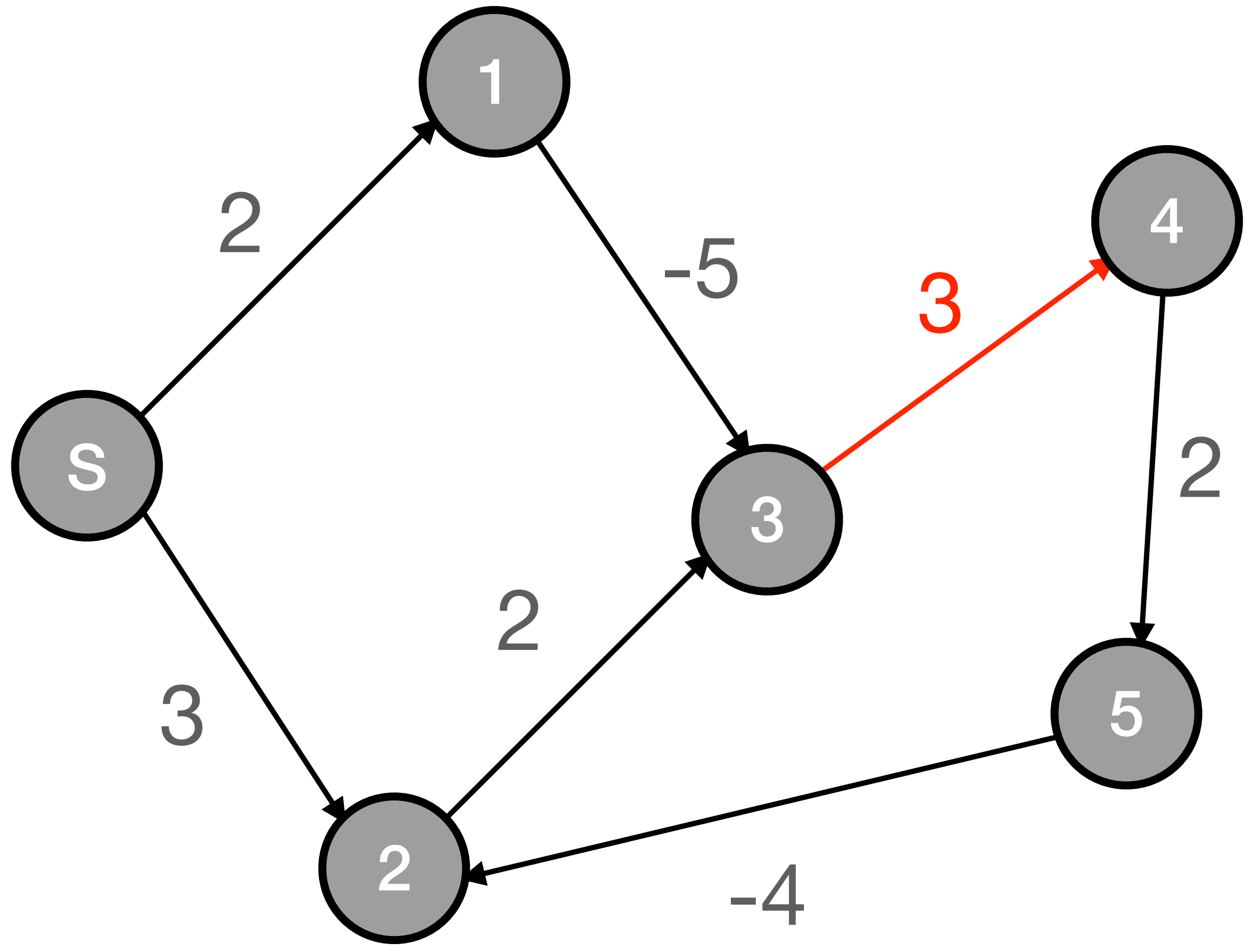
G



	S	1	2	3	4	5
dist[ ]	0	2	3	-3	INF	INF

Bellman-Ford Algorithm

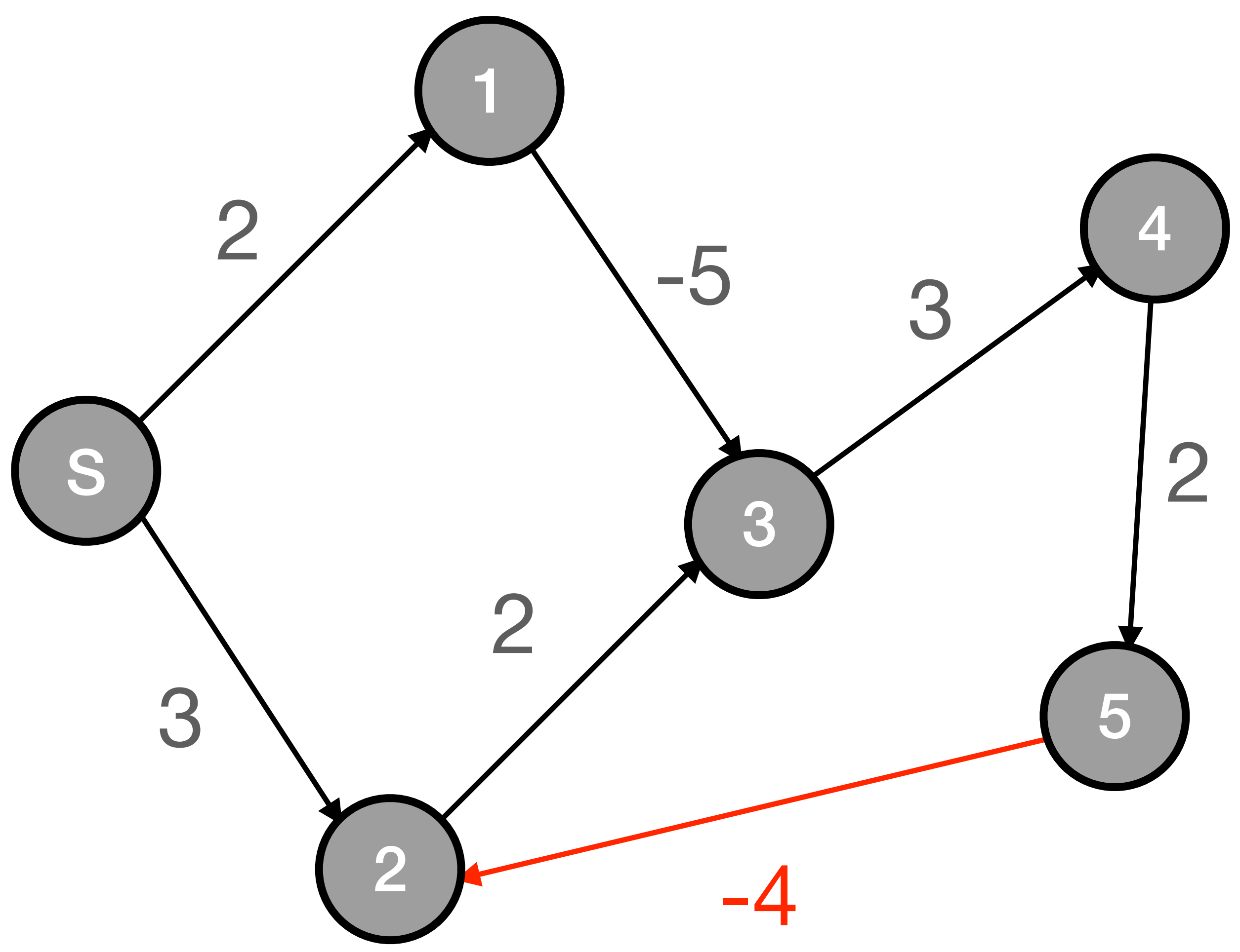
G



dist[ ]	S	1	2	3	4	5
	0	2	3	-3	0	INF

Bellman-Ford Algorithm

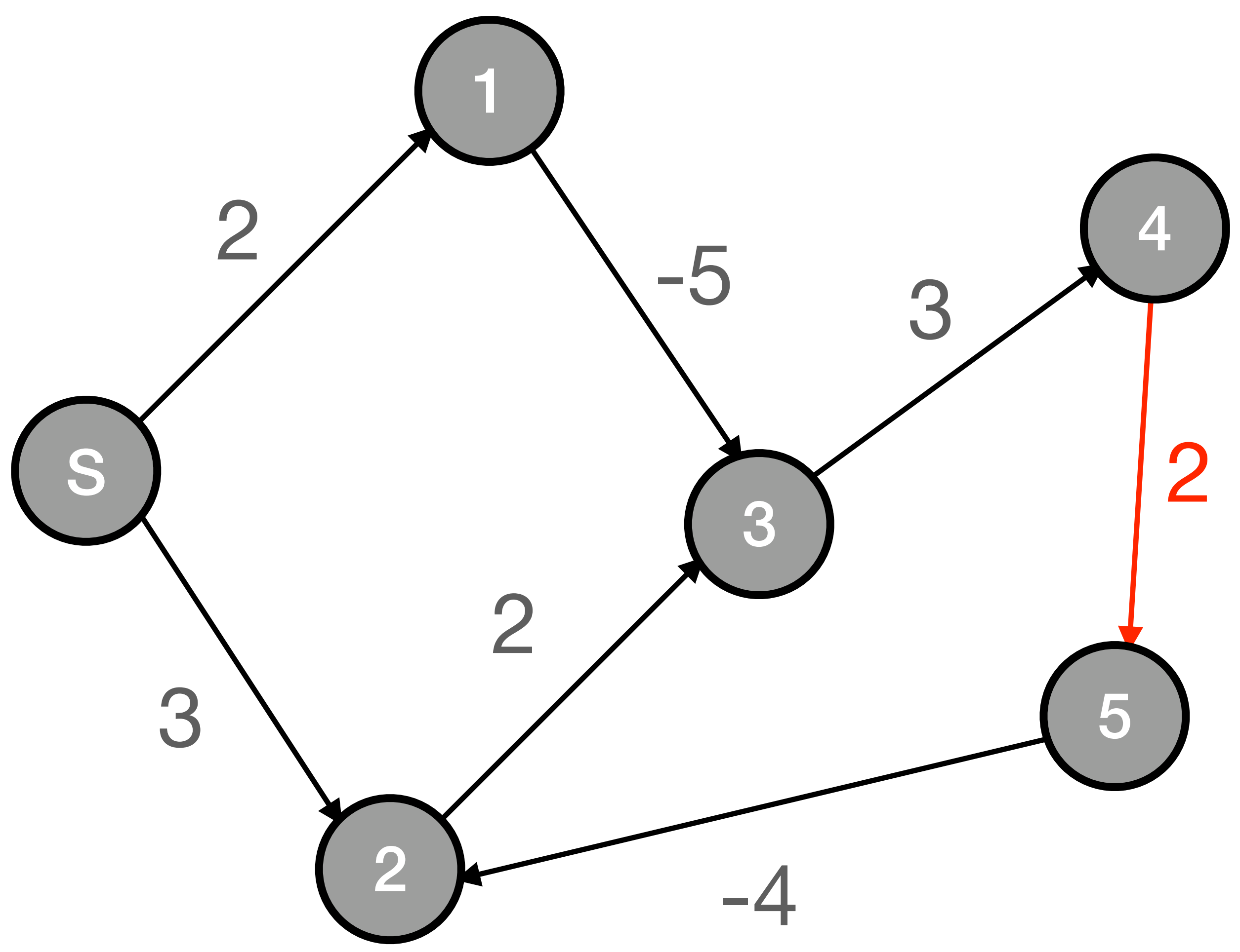
G



dist[ ]	S	1	2	3	4	5
	0	2	3	-3	0	INF

Bellman-Ford Algorithm

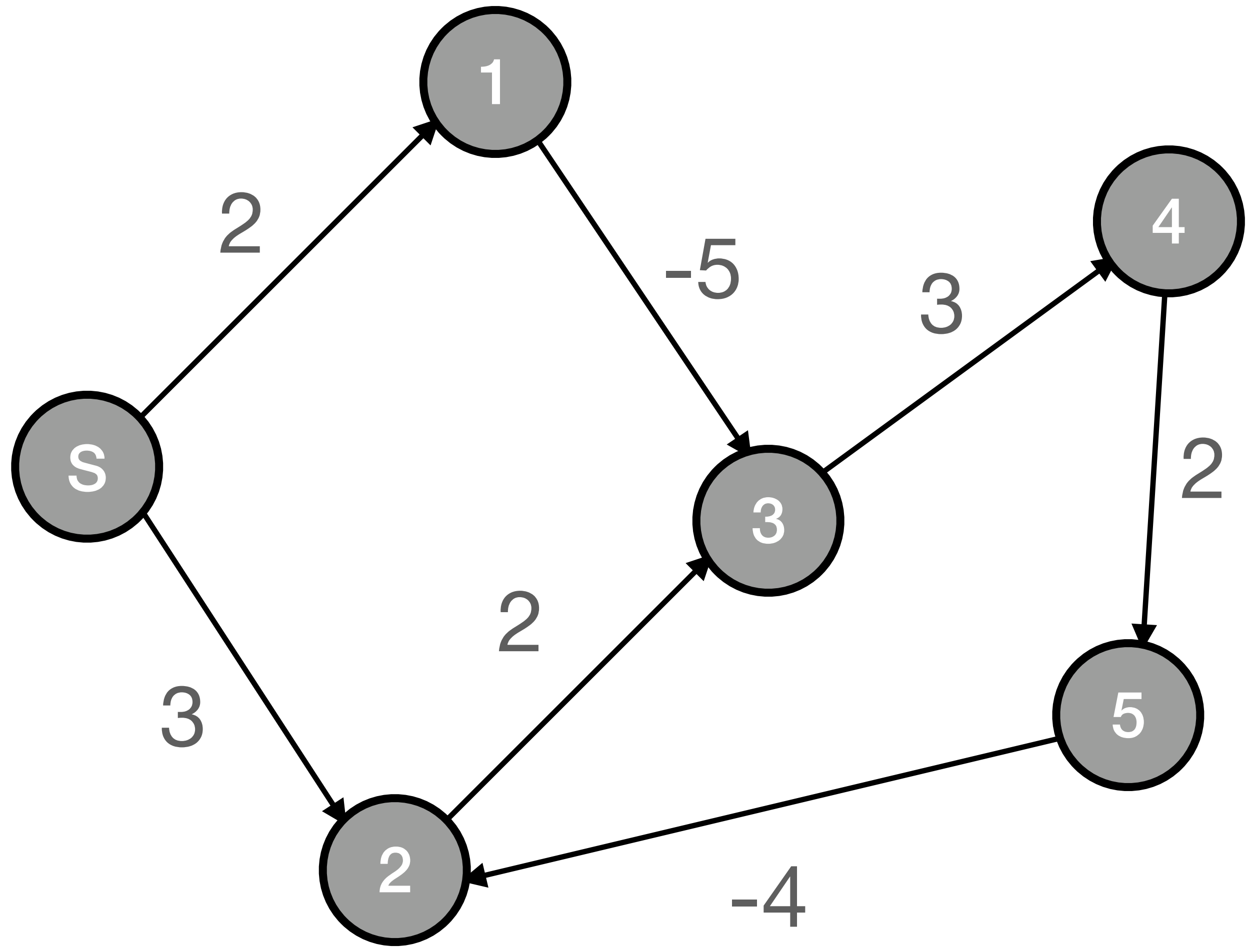
G



dist[ ]	S	1	2	3	4	5	
	0	2	3	-3	0	2	1회차 끝

Bellman-Ford Algorithm

G

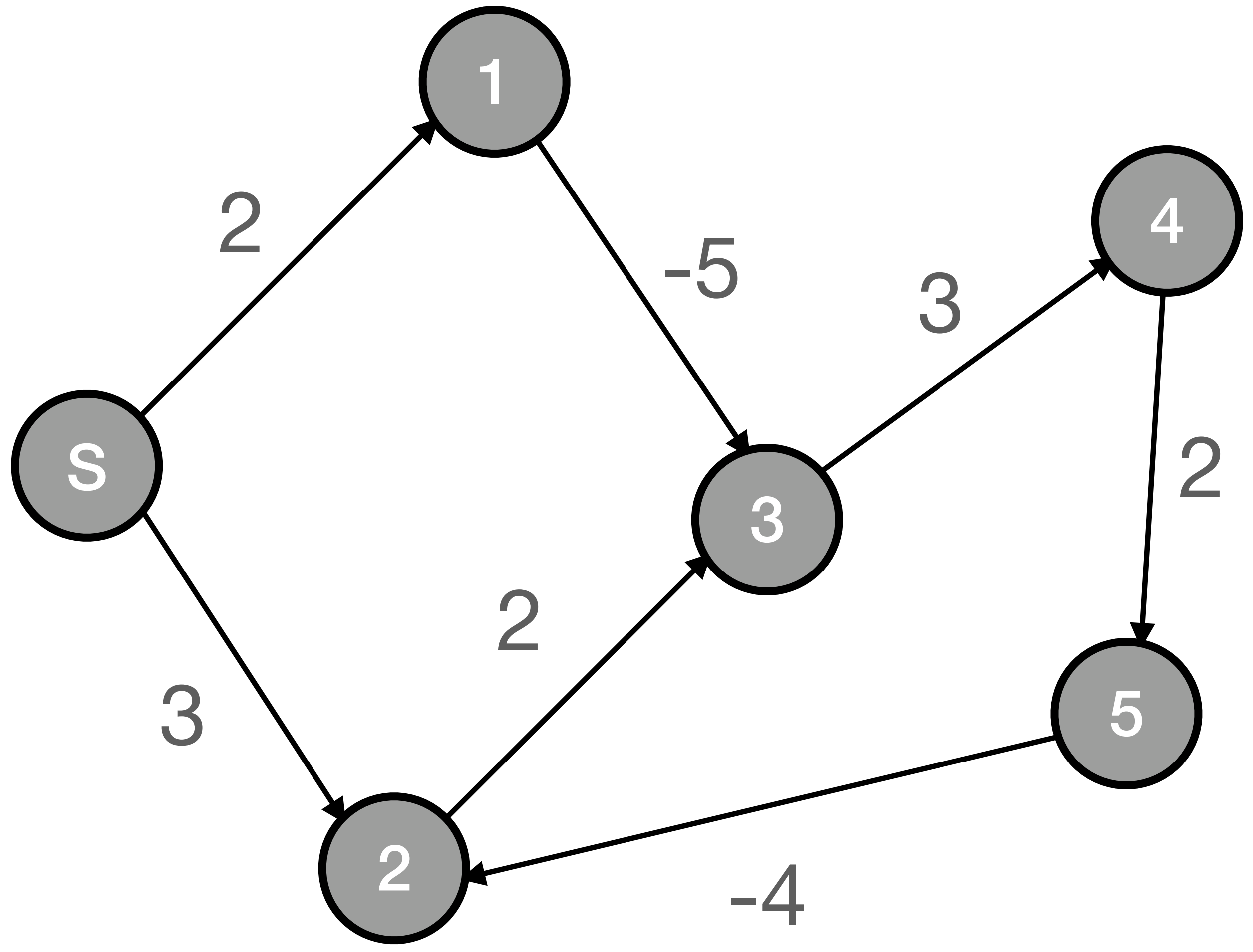


dist[ ]

	S	1	2	3	4	5	
	0	2	3	-3	0	2	
	0	2	-2	-3	0	2	2회차 끝

Bellman-Ford Algorithm

G



dist[ ]

S	1	2	3	4	5	
0	2	-2	-3	0	2	
0	2	-2	-3	0	2	3, 4, 5회차 끝: 각 정점까지의 최단거리

Bellman-Ford Algorithm

Time complexity

Naive approach

➡ 모든 edge 순회하기 $O(|E|)$ x |V|-1번

총 $O(|V||E|)$

Optimization (SPFA - Shortest Path Faster Algorithm)

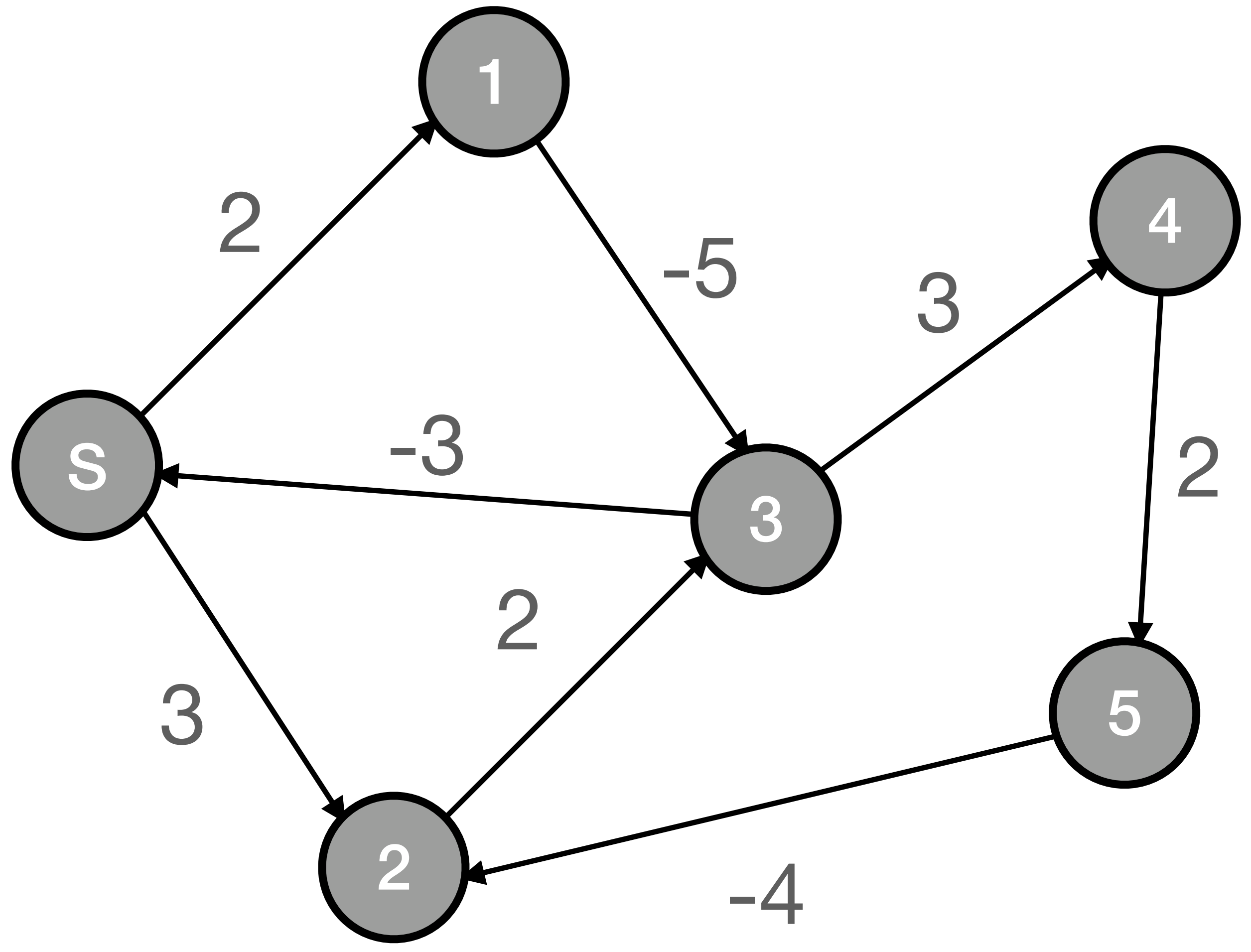
➡ 바뀐 정점과 연결된 edge에 대해서만 업데이트를 진행한다고 함 (자세히는 모름..)

시간복잡도는 $O(|V||E|)$이지만 평균적으로 $O(|V|+|E|)$ 또는 $O(|E|)$ 수준으로 돌아간다고 함

Bellman-Ford Algorithm

단점

G



dist[ ]

S	1	2	3	4	5
-INF	-INF	-INF	-INF	-INF	-INF

최단 경로를 찾을 때 길이가 1, 2, …, |V|-1인 경로를 모두 체크함

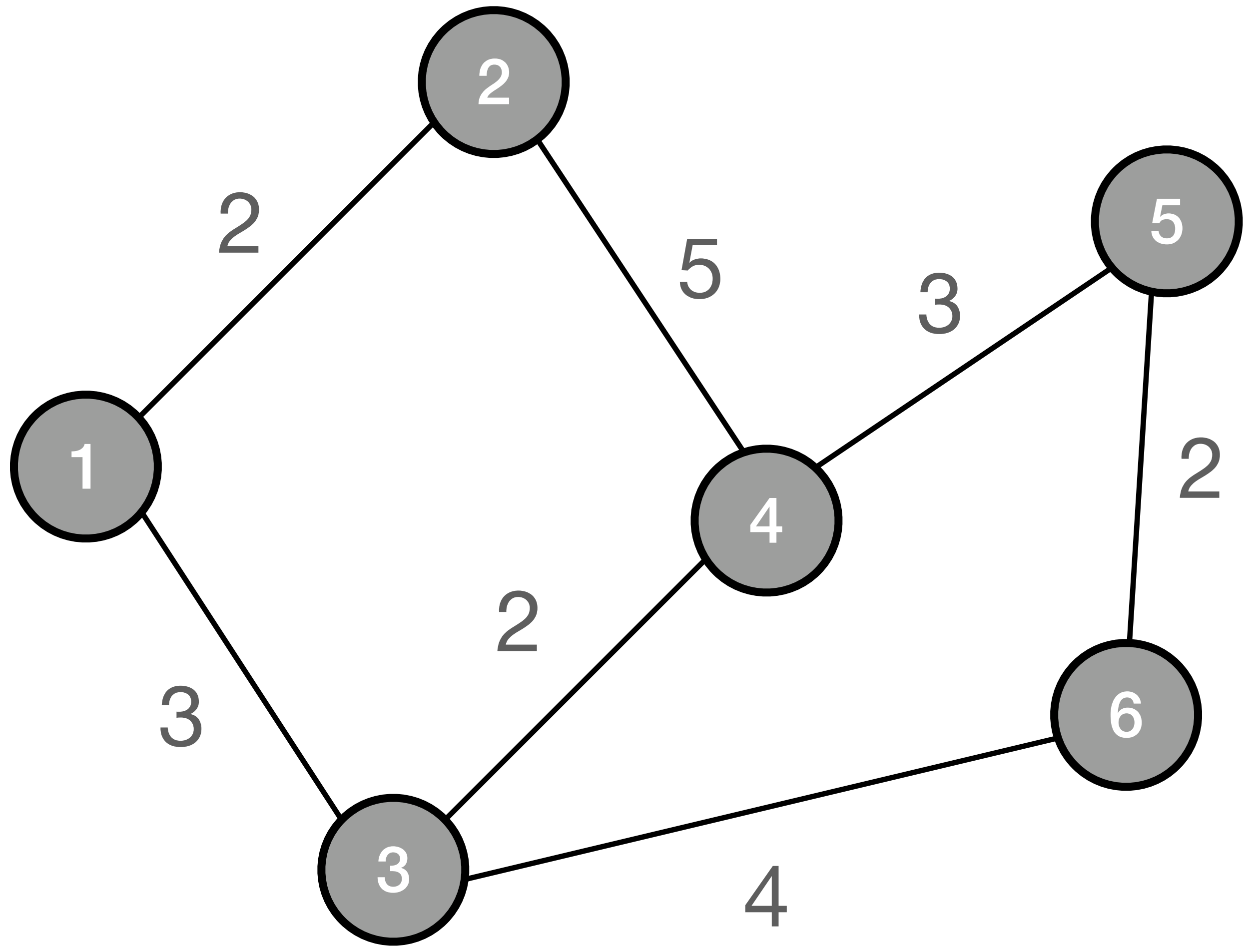
➡ 모든 최단경로는 정점을 중복하여 방문하지 않는다고 가정함

➡ 그래프에 음의 사이클이 존재할 경우 사용할 수 없음!
(가는 길에 음의 사이클이 있으면 거기만 계속 돌면 되니까)
|V|-1회차와 |V|회차의 dist가 같으면 올바른 최단거리이고,
다르면 음의 사이클이 존재함을 알 수 있음

Floyd-Warshall Algorithm

All-pair shortest path

G



dist[ ][ ]   dist[i][j]: i에서 j로 가는 최단경로의 거리

<u>$shortestPath(i, j, k)\ (1 \le k \le |V|)$</u>

i에서 j까지 $\{1,2,...,k\}$에 포함된 정점만을 거쳐 갈 수 있는 경로의 최단거리

단, 초기값인 $shortestPath(i, j, 0)$은

1) $if\ i = j,\ 0$
2) $else\ if\ edge\ (i, j)\ exists,\ w(i, j)$
3) $else,\ inf$

➡ $shortestPath(i, j, k-1)$을 알 때, $shortestPath(i, j, k)$의 경우의 수는

i) $shortestPath(i, j, k-1)$이 여전히 최단거리

ii) 중간 정점은 모두 $\{1,...,k\}$에 포함되는 i에서 k로, 그리고 k에서 j로 가는 새로운 최단 경로의 거리

➡ 따라서 $shortestPath(i, j, k)$는

i) $shortestPath(i, j, k-1)$

ii) $shortestPath(i, k, k-1) + shortestPath(k, j, k-1)$

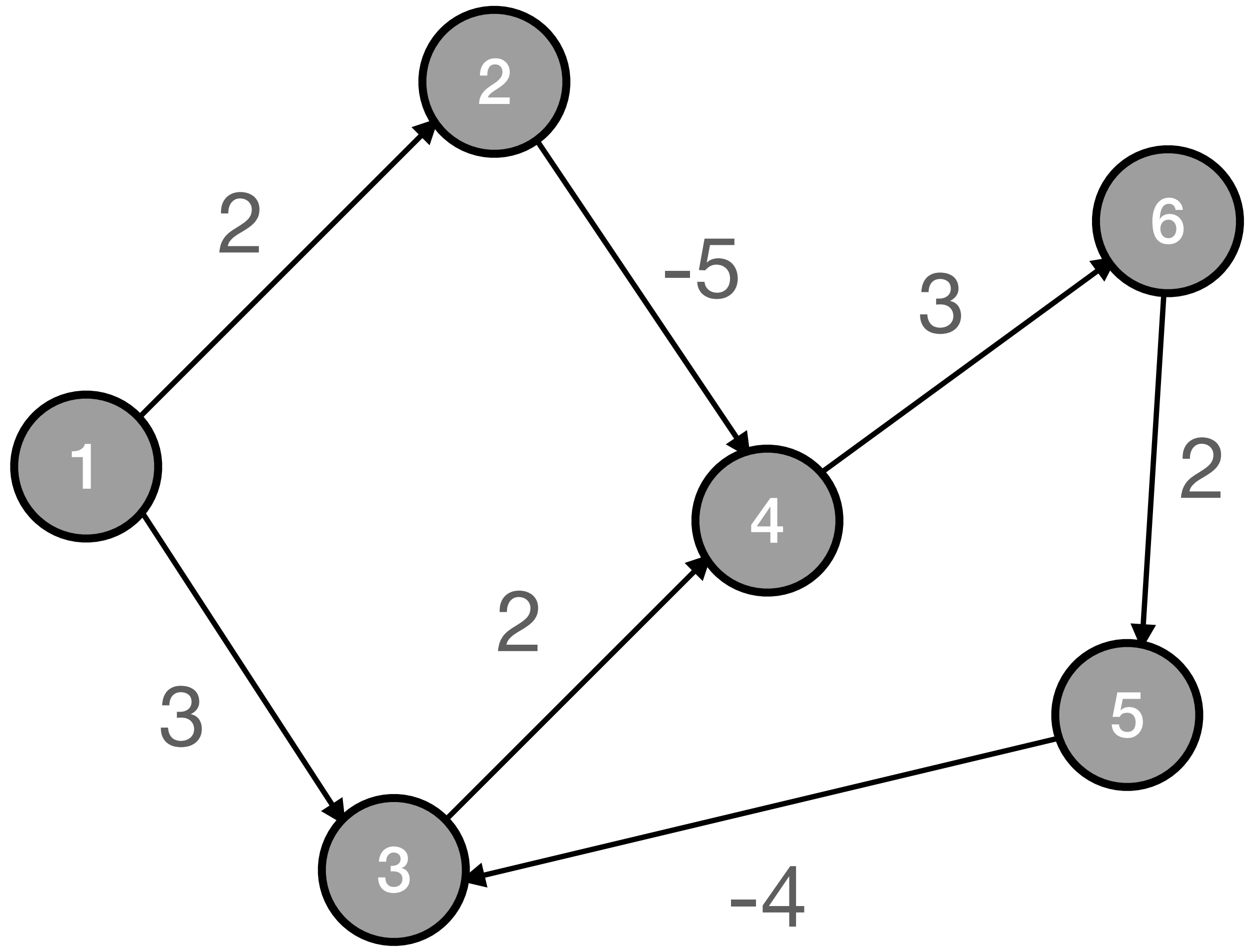
중 작은 값

Floyd-Warshall Algorithm

$shortestPath(i, j, k-1)$을 알 때, $shortestPath(i, j, k)$의 경우의 수는

i) $shortestPath(i, j, k-1)$이 여전히 최단거리

ii) 중간 정점은 모두 $\{1, ..., k\}$에 포함되는 i에서 k로, 그리고 k에서 j로 가는 새로운 최단 경로의 거리



initial

i \ j	1	2	3	4	5	6
1	0	2	3	INF	INF	INF
2	INF	0	INF	-5	INF	INF
3	INF	INF	0	2	INF	INF
4	INF	INF	INF	0	INF	3
5	INF	INF	-4	INF	0	INF
6	INF	INF	INF	INF	2	0

$$shortestPath(i, j, k) = \min(shortestPath(i, j, k-1), shortestPath(i, k, k-1) + shortestPath(k, j, k-1))$$

Floyd-Warshall Algorithm

$shortestPath(i, j, k-1)$을 알 때, $shortestPath(i, j, k)$의 경우의 수는

i) $shortestPath(i, j, k-1)$이 여전히 최단거리

ii) 중간 정점은 모두 $\{1,...,k\}$에 포함되는 i에서 k로, 그리고 k에서 j로 가는 새로운 최단 경로의 거리

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

$$shortestPath(i, j, k) = \min(shortestPath(i, j, k-1), shortestPath(i, k, k-1) + shortestPath(k, j, k-1))$$

Floyd-Warshall Algorithm

단점

다익스트라처럼 '최단경로에서 쌓아올린 게 최단경로다!' 하는 건 아니라서 음의 가중치를 갖는 edge가 있어도 사용 가능.

단, 마찬가지로 음의 사이클이 있으면 사용 불가능.

Time complexity

$shortestPath(i, j, k)$ 구하기: $O(|V|^2)$

$(1 \leq k \leq |V|)$: $O(|V|)$

총 $O(|V|^3)$