

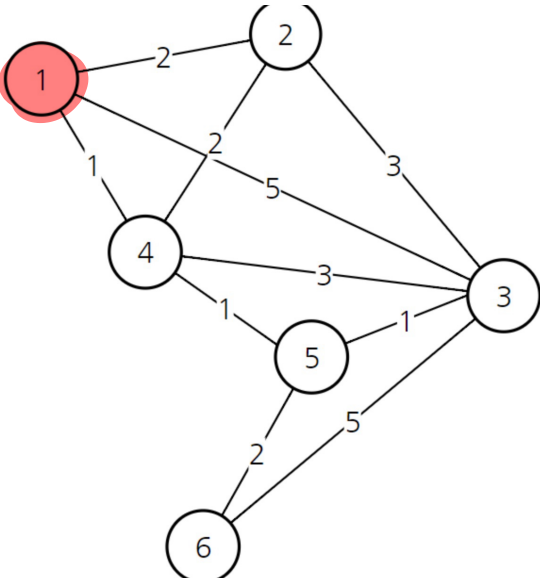
+

×

—

÷

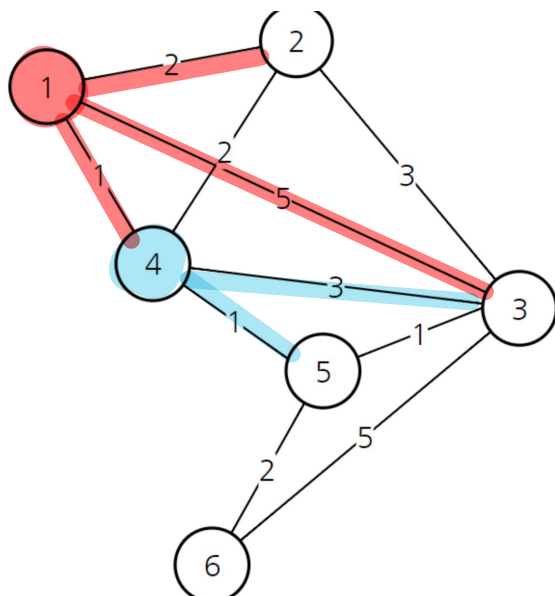
다익스트라 알고리즘



1	2	3	4	5	6
0	2	5	1	무한	무한
2	0	3	2	무한	무한
5	3	0	3	1	5
1	2	3	0	1	무한
무한	무한	5	1	0	2
무한	무한	5	무한	2	0

출발 노드는 노드 1번

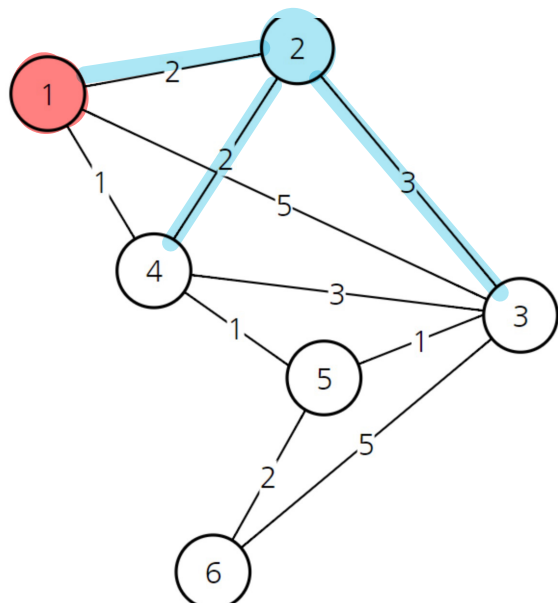
다익스트라 알고리즘



1	2	3	4	5	6
0	2	4	1	2	무한
2	0	3	2	무한	무한
5	3	0	3	1	5
1	2	3	0	1	무한
무한	무한	5	1	0	2
무한	무한	5	무한	2	0

1. 노드1을 선택
2. 노드1과 과 연결된 3개의 간선을 확인
3. 방문하지 않은 노드 중 가장 비용이 적은 노드는 4번
4. 4번 노드를 거쳐서 가는 경우를 모두 고려하여 최소비용 갱신

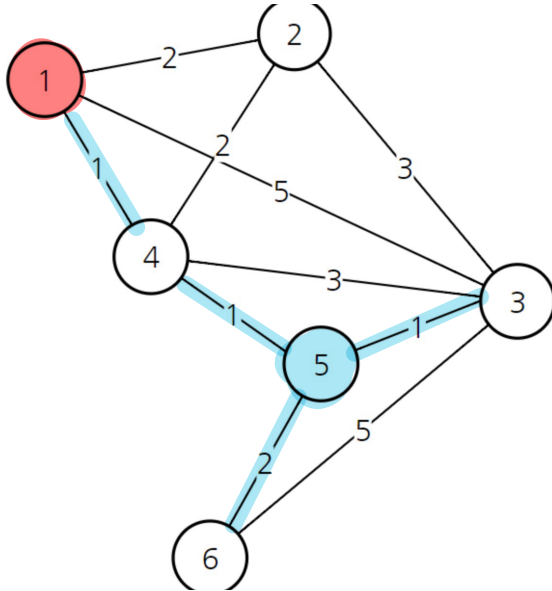
다익스트라 알고리즘



1	2	3	4	5	6
0	2	4	1	2	무한
2	0	3	2	무한	무한
5	3	0	3	1	5
1	2	3	0	1	무한
무한	무한	5	1	0	2
무한	무한	5	무한	2	0

1. 다음으로 방문하지 않은 노드 중 최소비용은 노드 2번
2. 2번을 거쳐서 가는 길 중 비용이 갱신되는 경우 없음

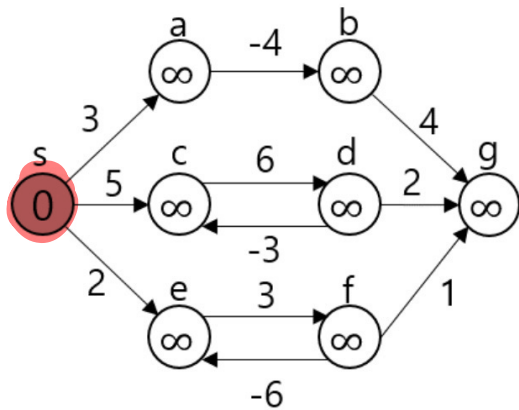
다익스트라 알고리즘



1	2	3	4	5	6
0	2	3	1	2	4
2	0	3	2	무한	무한
5	3	0	3	1	5
1	2	3	0	1	무한
무한	무한	5	1	0	2
무한	무한	5	무한	2	0

1. 그 다음 최소비용 노드 5번
2. 노드 5번을 거쳐서 갈 경우 비용 갱신 됨
3. 이후 방문하지 않은 노드를 탐색해도 최소비용은 갱신되지 않음

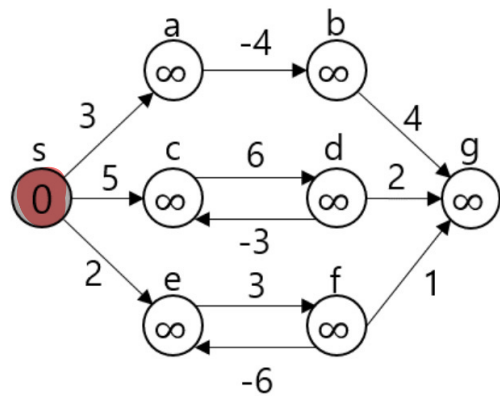
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	∞	∞	∞	∞	∞	∞	∞

시작노드(s) 노드 0번 지정

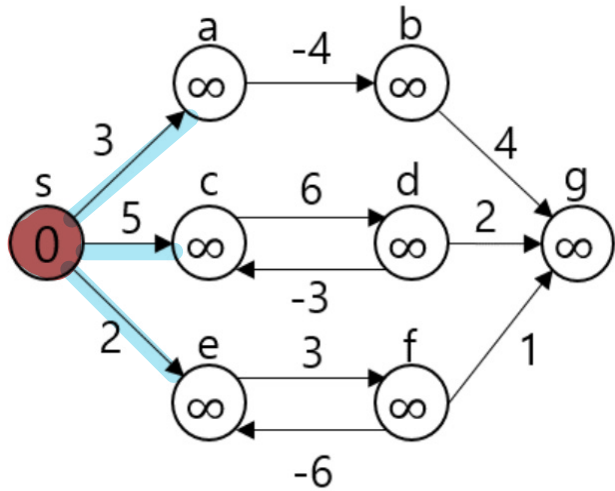
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	∞	∞	∞	∞	∞	∞	∞

- 1. 시작점 노드 제외 무한 기호로 설정

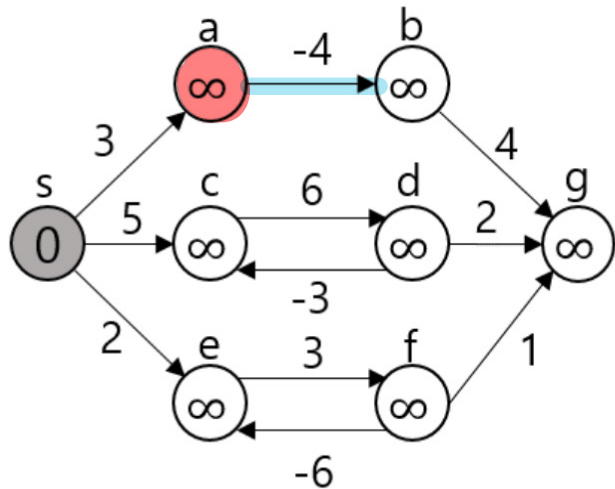
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	3	∞	5	∞	2	∞	∞

1. 노드 0번과 인접한 노드 a, c, e의 거리값은 무한기호보다 작으니깐 갱신

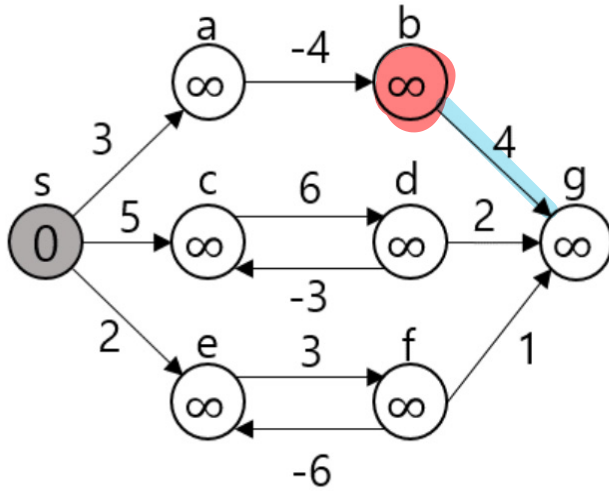
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	3	-1	5	∞	2	∞	∞

1. 노드 0의 인접한 노드를 다 탐색했으니 노드 a의 인접 노드 탐색 시작
2. 인접 노드 중 최소비용 거리 갱신이 가능

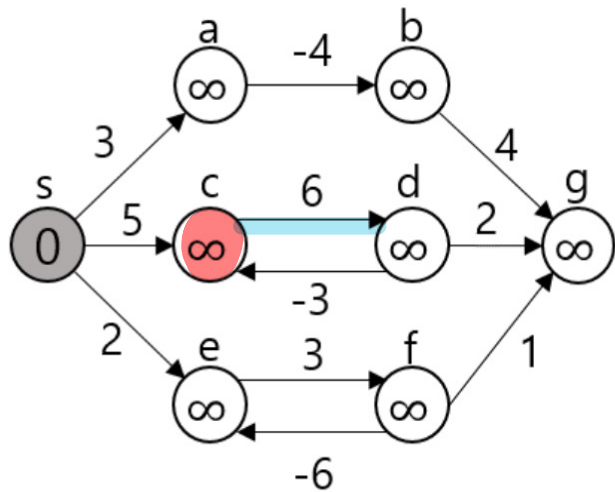
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	3	-1	5	∞	2	∞	3

1. 노드 a의 인접한 노드를 다 탐색했으니 노드 b의 인접 노드 탐색 시작
2. 인접 노드 중 최소비용 거리 갱신이 가능

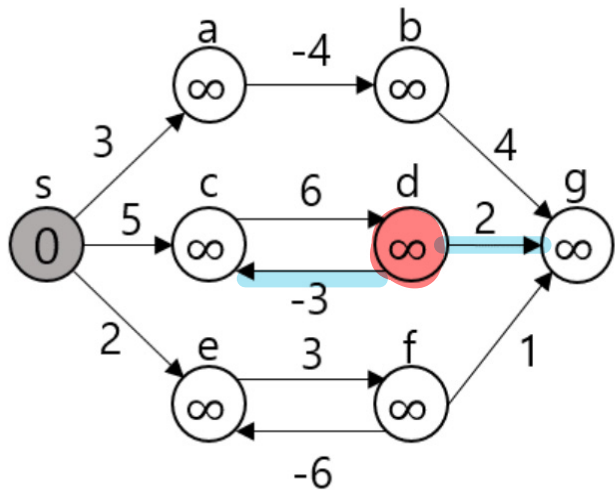
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	3	-1	5	1	2	∞	3

1. 노드 b의 인접한 노드를 다 탐색했으나 노드 c의 인접 노드 탐색 시작
2. 인접 노드 중 최소비용 거리 갱신이 가능

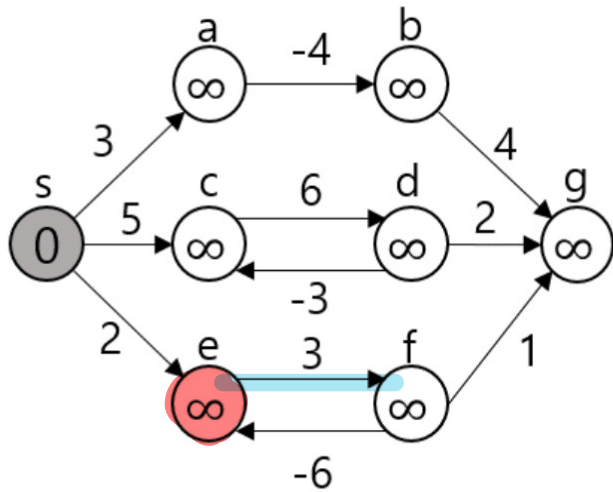
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	3	-1	5	11	2	∞	3

1. 노드 c의 인접한 노드를 다 탐색했으니 노드 d의 인접 노드 탐색 시작
2. 인접 노드 중 최소비용 거리 갱신이 없음

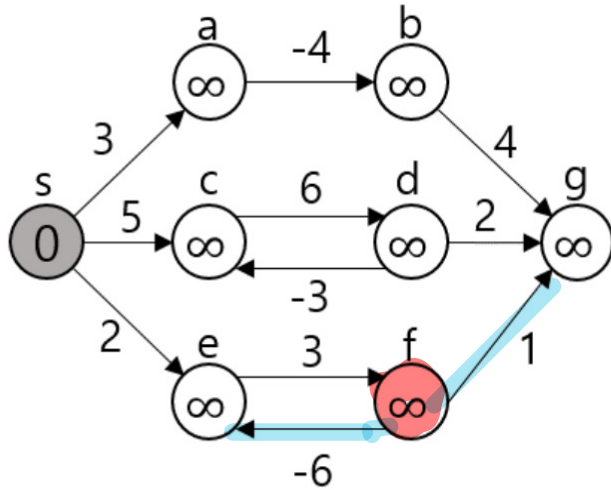
벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	3	-1	5	11	2	5	3

1. 노드 d의 인접한 노드를 다 탐색했으니 노드 e의 인접 노드 탐색 시작
2. 인접 노드 중 최소비용 거리 갱신이 가능

벨만포드 알고리즘



s	a	b	c	d	e	f	g
0	3	-1	5	11	-1	5	3

1. 노드 e의 인접한 노드를 다 탐색했으니 노드 f의 인접 노드 탐색 시작
2. 인접 노드 중 최소비용 거리 갱신이 가능
3. 노드 g의 인접노드는 존재하지 않으니 탐색 종료
4. 이 과정을 노드 전체 수 -1 만큼 반복하며 최소비용 갱신