2025 겨울방학 알고리즘 스터디

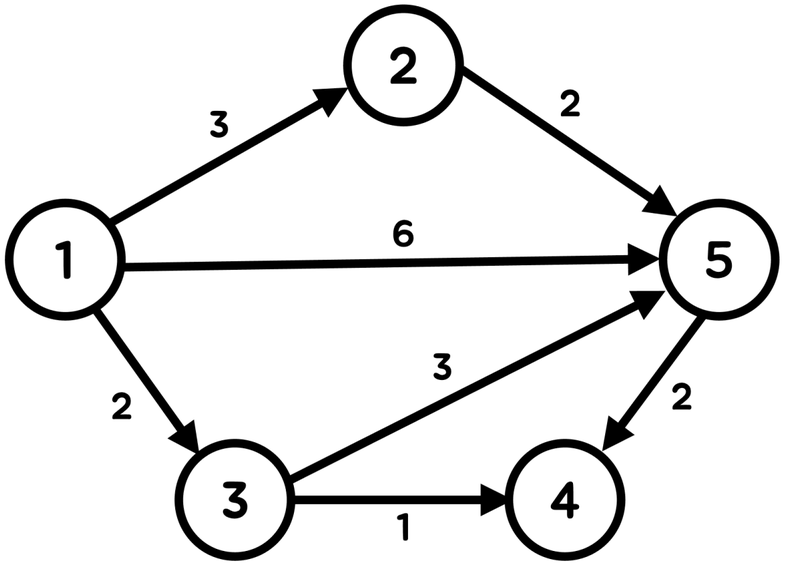
최단경로 알고리즘

컴퓨터 공학과 20230546 서보경

목차

다익스트라 알고리즘

다익스트라 알고리즘은 그래프에서 최단 경로를 효율적으로 찾는 알고리즘으로, 특히 가중치가 양수인 경우 매우 강력한 성능을 발휘한다.
이 알고리즘은 각 정점까지의 최단 거리를 실시간으로 업데이트하며, 우선순위 큐(힙)를 사용해 아직 방문하지 않은 정점 중 최단 거리를 가진 정점을 빠르게 선택할 수 있다.
특유의 탐욕적 선택 속성(그리디)으로 인해 시간 복잡도가 O(V + E log V)로 최적화되어, 큰 규모의 그래프에서도 효율적으로 작동한다.



다익스트라 알고리즘

문제

N개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 M개의 버스가 있다. 우리는 A번째 도시에서 B번째 도시까지 가는데 드는 버스 비용을 최소화 시키려고 한다. A번째 도시에서 B번째 도시까지 가는데 드는 최소비용을 출력하여라. 도시의 번호는 1부터 N까지이다.

입력

첫째 줄에 도시의 개수 N(1 ≤ N ≤ 1,000)이 주어지고 둘째 줄에는 버스의 개수 M(1 ≤ M ≤ 100,000)이 주어진다. 그리고 셋째 줄부터 M+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 그리고 그 다음에는 도착지의 도시 번호가 주어지고 또 그 버스 비용이 주어진다. 버스 비용은 0보다 크거나 같고, 100,000보다 작은 정수이다.

그리고 M+3째 줄에는 우리가 구하고자 하는 구간 출발점의 도시번호와 도착점의 도시번호가 주어진다. 출발점에서 도착점을 갈 수 있는 경우만 입력으로 주어진다.

출력

첫째 줄에 출발 도시에서 도착 도시까지 가는데 드는 최소 비용을 출력한다.

n개의 도시가 있으며, 특정 시작 도시로부터 도착 도시까지 가는데 드는 최소 버스 비용을 구해야하는 문제이다.

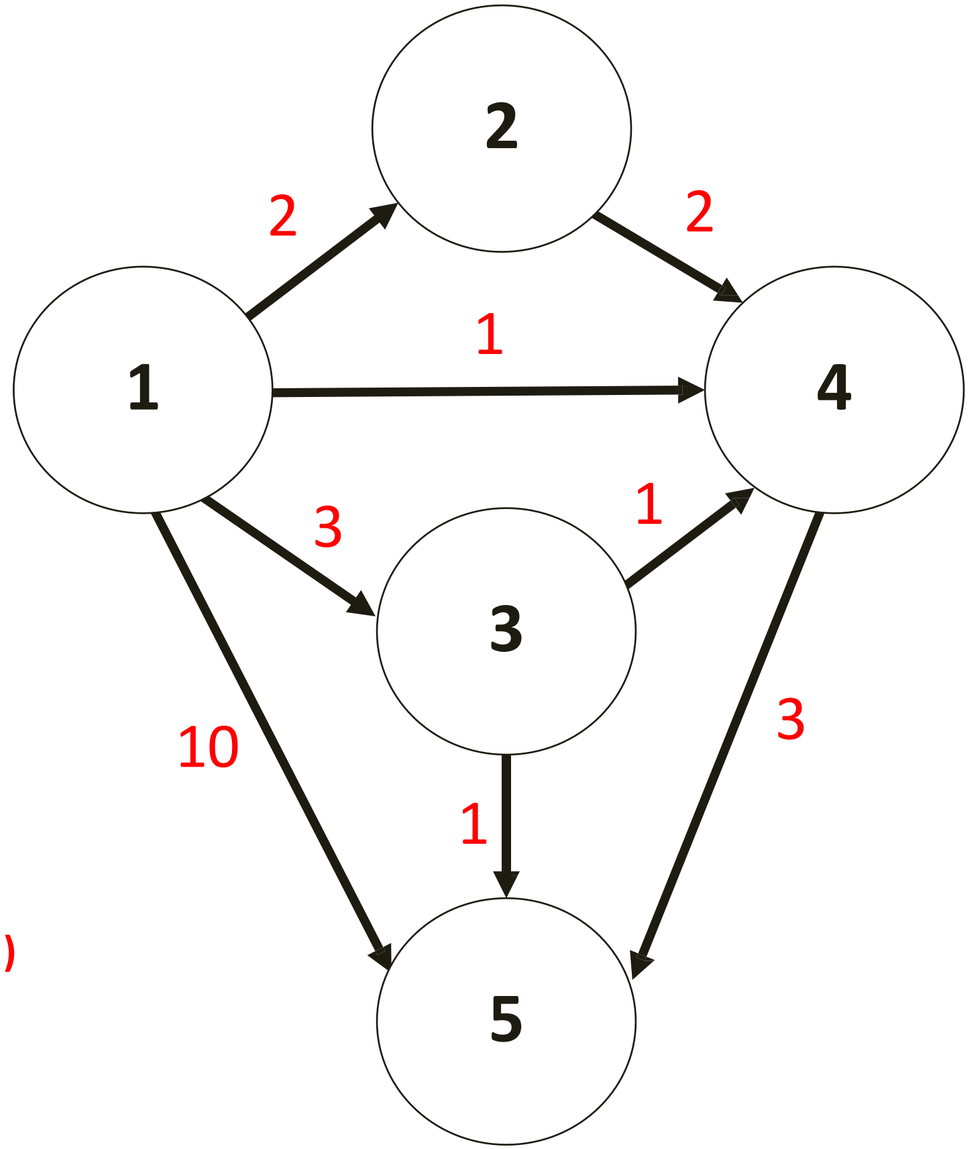
먼저 정점과 간선 정보를 입력 받은 뒤 그래프를 생성 한다.

이후, 다익스트라 알고리즘을 통해 시작 정점으로 부터 다른 정점까지의 최단 비용(가중치 합)을 구한다.

다익스트라 알고리즘

예제 입력 1 복사

```
5
8
1 2 2
1 3 3
1 4 1
1 5 10
2 4 2
3 4 1
3 5 1
4 5 3
1 5
```



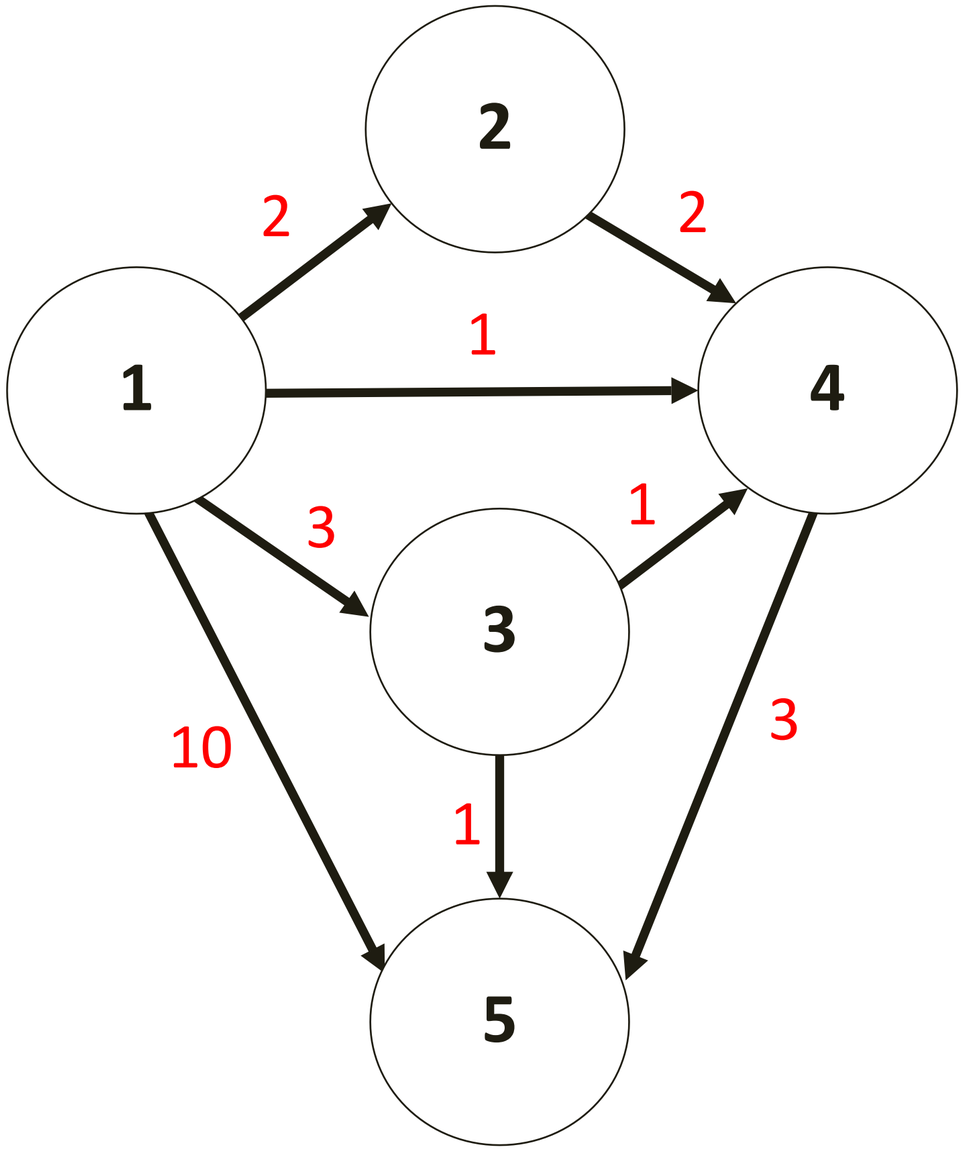
1 – 시작 정점 부터 최소 비용을 갱신해 나가기
(경로 추적할거면 비용 갱신될때 직전 정점도 같이 갱신 해주기)

2 – 가중치를 기준으로 하는 최소 힙을 만든다! (+ 방문 처리 없어도 됨)
-> 그러나 pdf에선 설명을 위해 방문처리를 한다고 가정한다.

3 – 최단경로가 확정된 정점은 보라색 표시

4 – 음수 간선 있을 땐 다익스트라를 사용하면 안됨!!

다익스트라 알고리즘



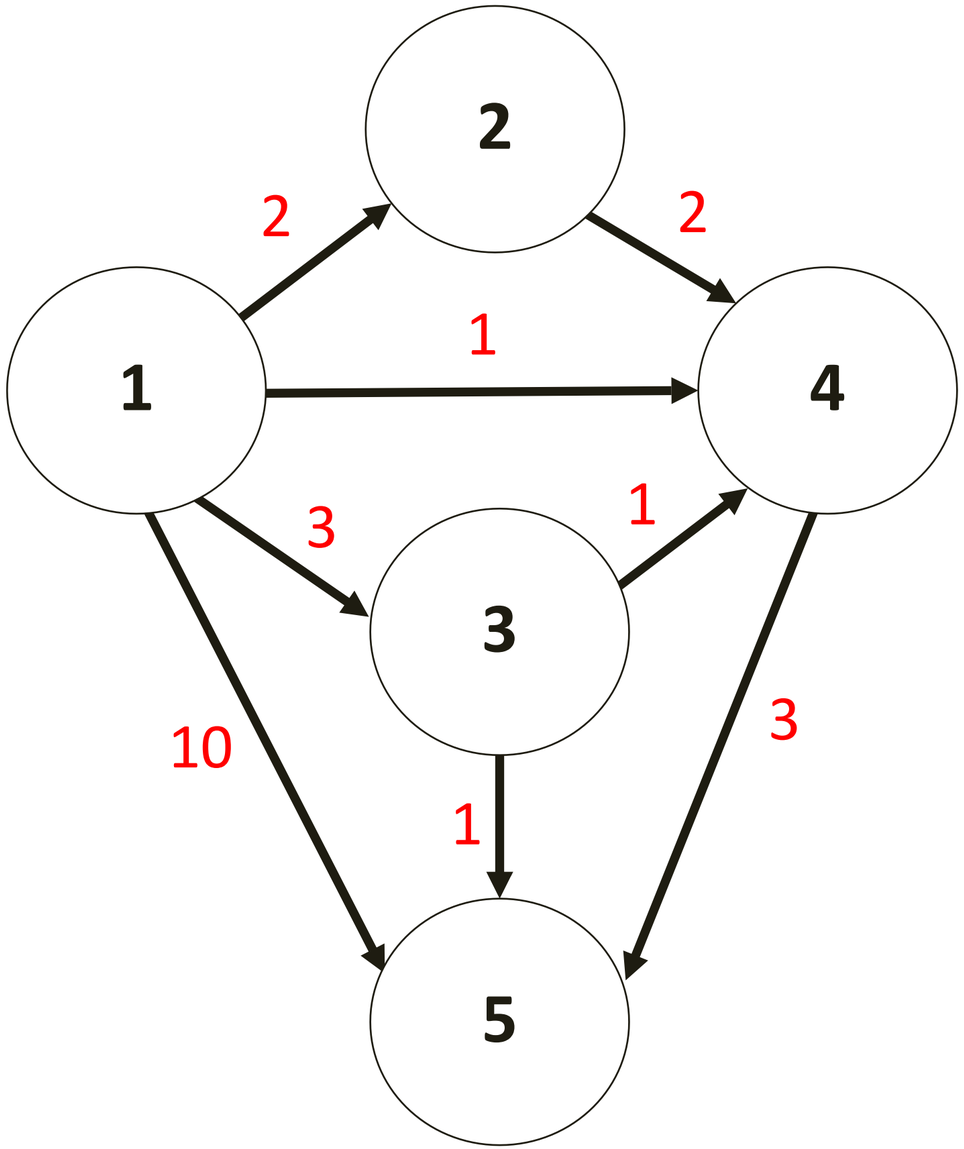
아직 탐색을 시작하기 전이다.

모든 정점에 대한 최소 비용은 모두 무한대로 초기화 해준다.

Heap(최소)

정점 번호	1	2	3	4	5
최소 비용	INF	INF	INF	INF	INF
직전 정점	-1	-1	-1	-1	-1

다익스트라 알고리



시작 정점은 1번 정점이다.

1번 정점의 최소 비용 0으로 초기화 해준다음에 정점 번호와 가중치 0을 힙에 넣는다.

1번 정점에서 1번 정점으로 가는 거리(가중치)는 0이기 때문이다! (자기 자신 -> 자기 자신)

다익스트라의 최단 경로 알고리즘은 특정 정점으로부터 출발하는 최단 경로를 구할 수 있다.

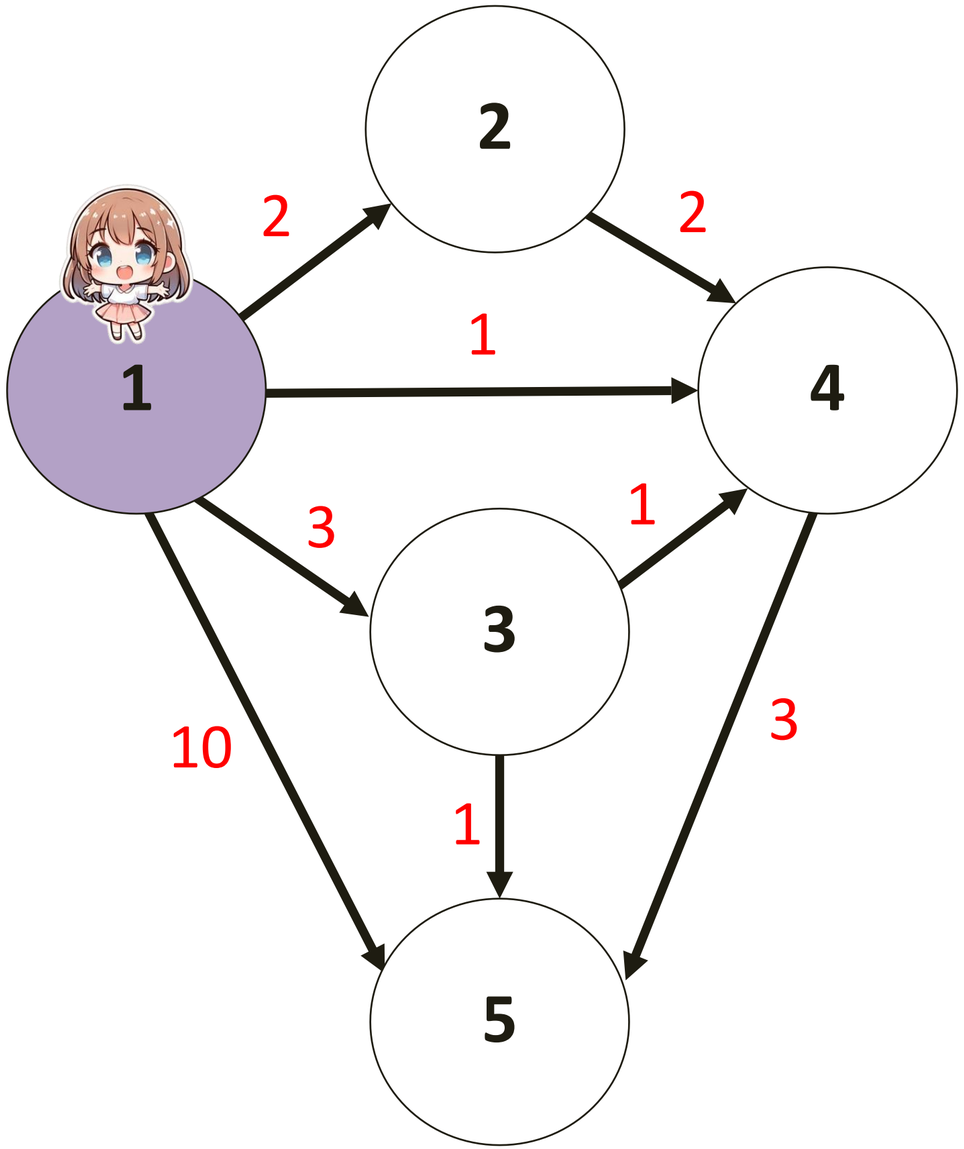
(모든 정점으로 부터의 최단 경로는 플로이드-워셜 에서 다룸)

Heap(최소)

1번 정점 (비용 0)

정점 번호	1	2	3	4	5
최소 비용	0	INF	INF	INF	INF
직전 정점	-1	-1	-1	-1	-1

다익스트라 알고리즘



힙에서 원소를 하나 빼온다.

정점 번호는 1이며, 이때 비용은 0이다.

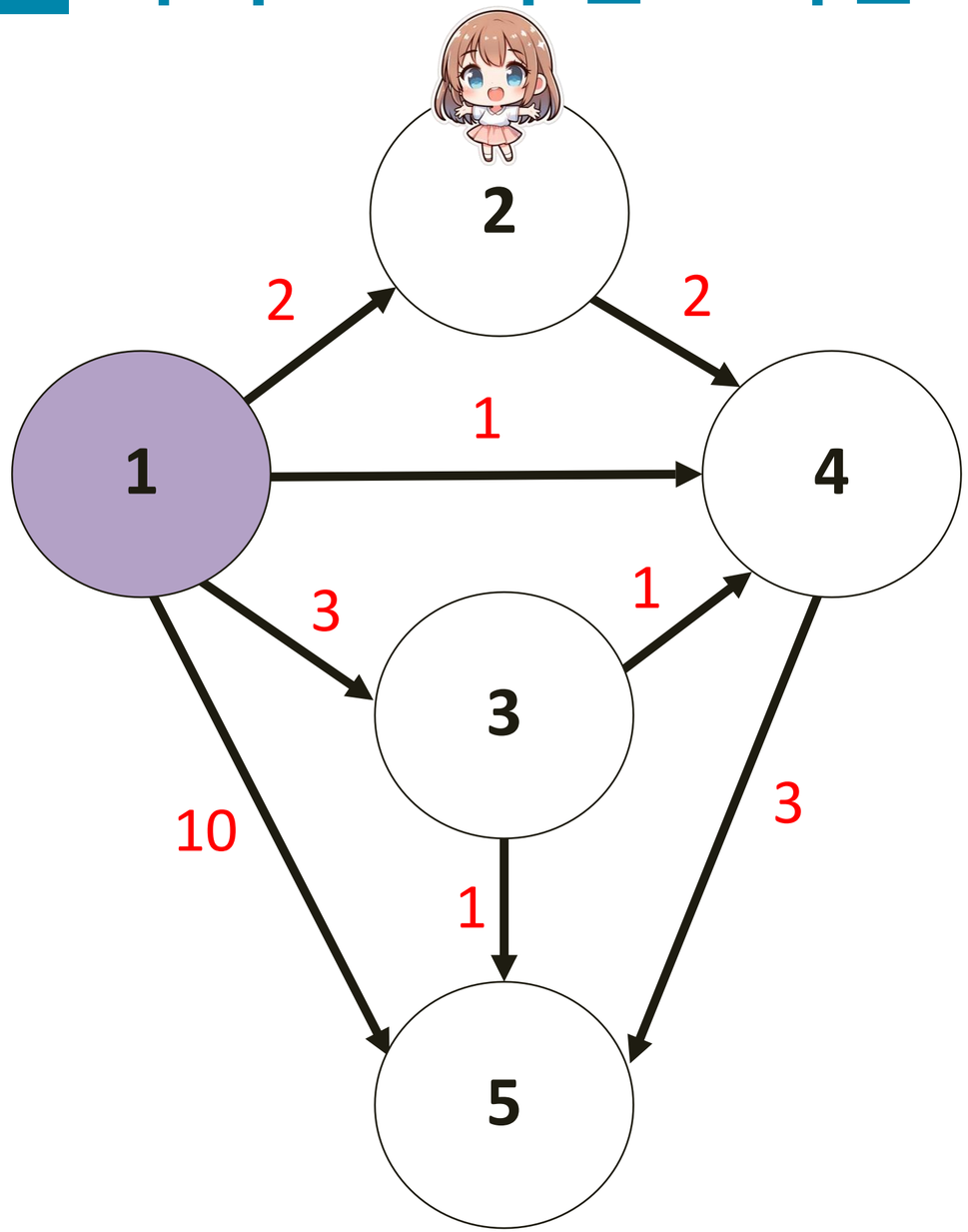
1번 정점을 방문 처리 해준다.
(탐욕적 선택 속성때문에 가능함)

이후 1번 정점에서 갈 수 있는 간선을 검사해준다.

Heap(최소)

정점 번호	1	2	3	4	5
최소 비용	0	INF	INF	INF	INF
직전 정점	-1	-1	-1	-1	-1

다익스트라 알고리즘



2번 정점은 1번 정점과 연결 되어있다.

간선의 가중치는 2이다.

갱신 비용 = 0 + 2 = 2

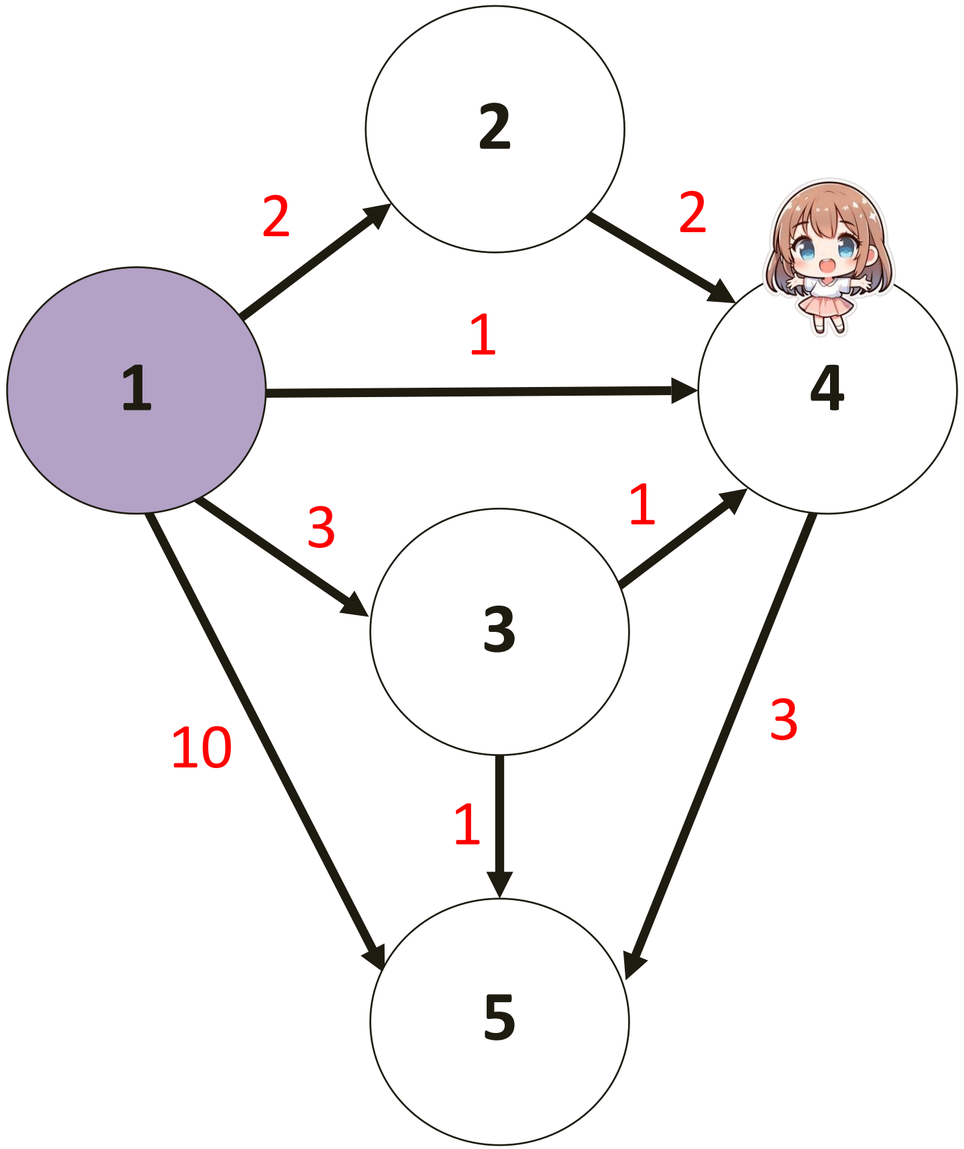
이때, 기존 비용 보다 갱신 비용이 더 작으므로
(INF vs 2) 비용을 갱신해주고 힙에 넣어준다.

Heap(최소)

2번 정점 (비용 2)

정점 번호	1	2	3	4	5
최소 비용	0	2	INF	INF	INF
직전 정점	-1	1	-1	-1	-1

다익스트라 알고리즘



4번 정점은 1번 정점과 연결 되어있다.

간선의 가중치는 1이다.

갱신 비용 = 0 + 1 = 1

이때, 기존 비용 보다 갱신 비용이 더 작으므로
(INF vs 1) 비용을 갱신해주고 힙에 넣어준다.

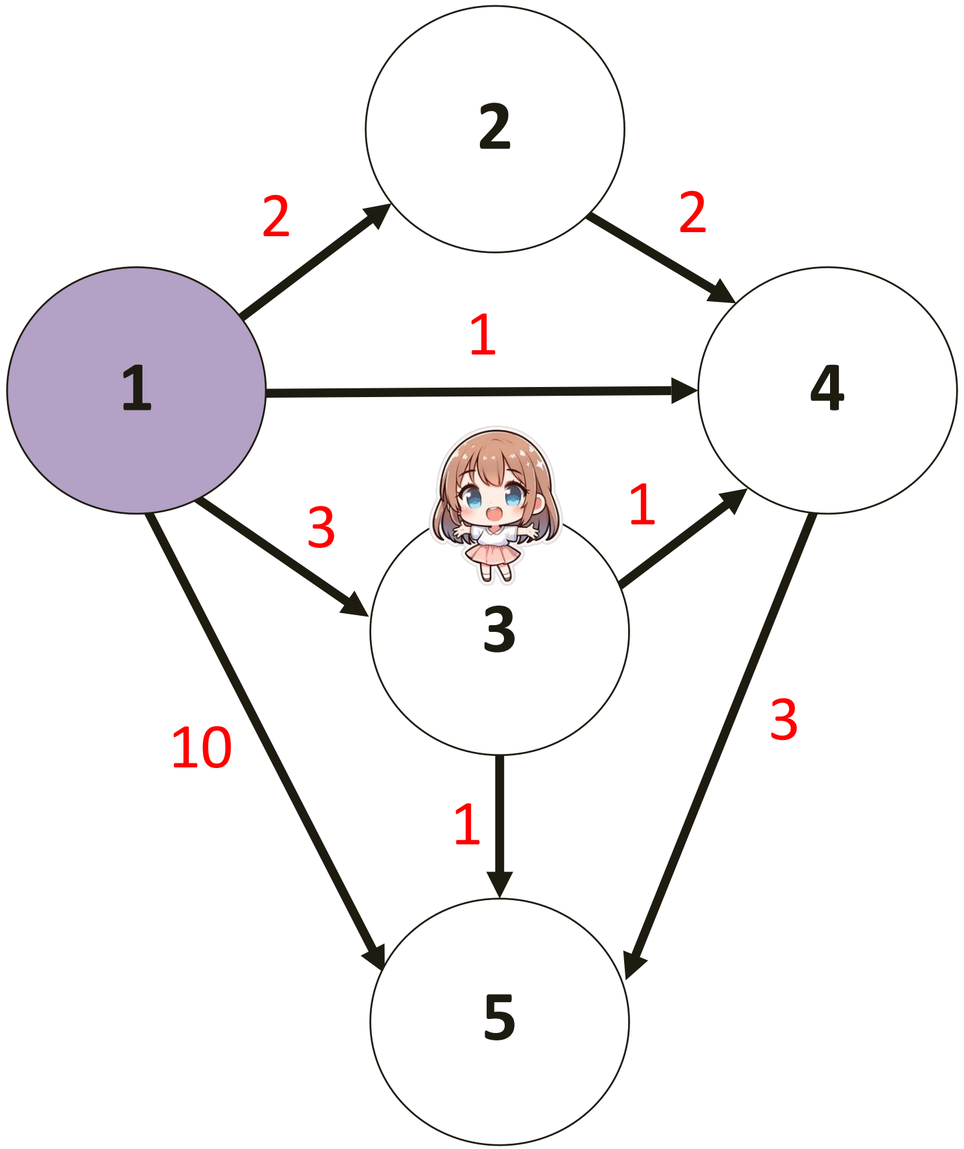
Heap(최소)

4번 정점 (비용 1)

2번 정점 (비용 2)

정점 번호	1	2	3	4	5
최소 비용	0	2	INF	1	INF
직전 정점	-1	1	-1	1	-1

다익스트라 알고리즘



3번 정점은 1번 정점과 연결 되어있다.

간선의 가중치는 3이다.

갱신 비용 = 0 + 3 = 3

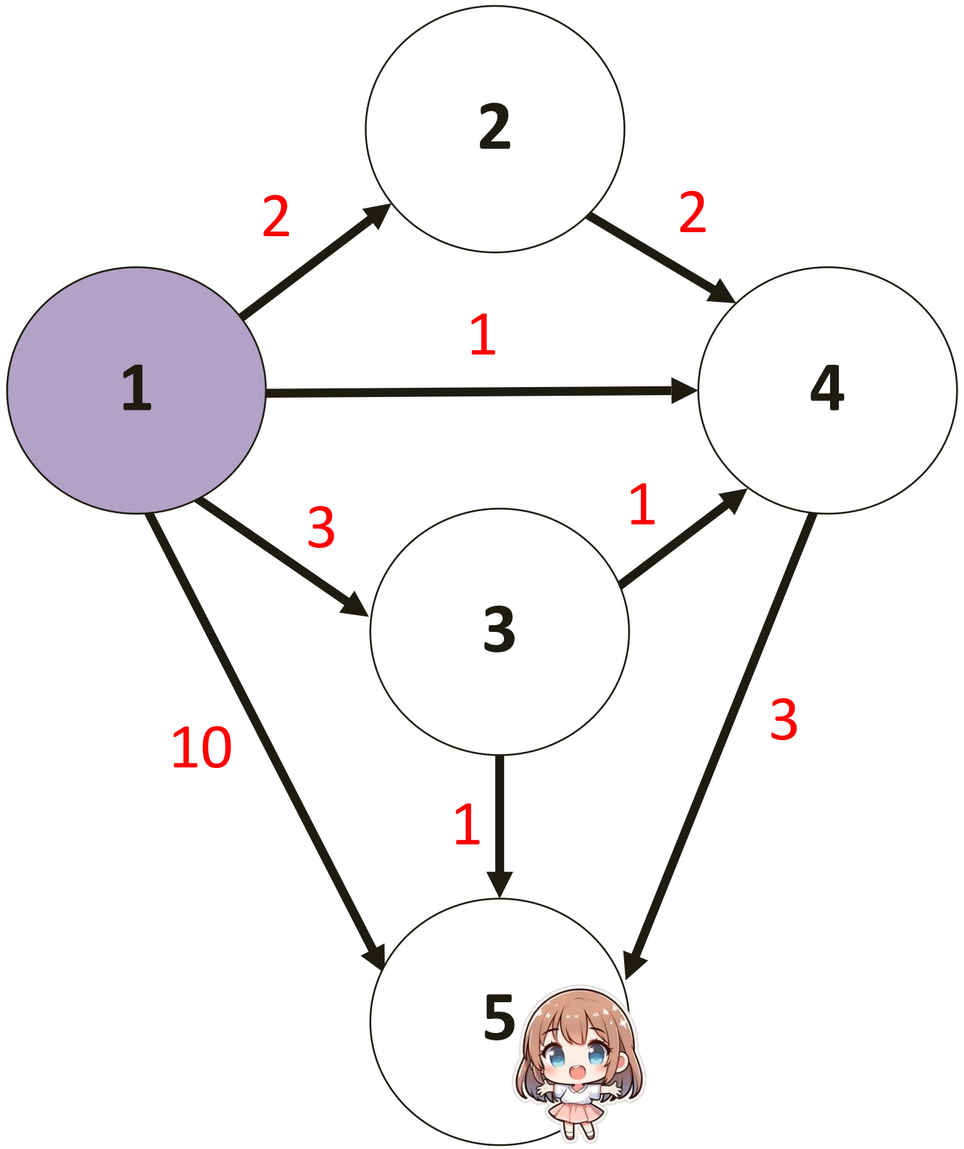
이때, 기존 비용보다 갱신 비용이 더 작으므로
(INF vs 3) 비용을 갱신해주고 힙에 넣어준다.

Heap(최소)

- 4번 정점 (비용 1)
- 2번 정점 (비용 2)
- 3번 정점 (비용 3)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	INF
직전 정점	-1	1	1	1	-1

다익스트라 알고리즘



5번 정점은 1번 정점과 연결 되어있다.

간선의 가중치는 10이다.

갱신 비용 = 0 + 10 = 10

이때, 기존 비용보다 갱신 비용이 더 작으므로
(INF vs 10) 비용을 갱신해주고 힙에 넣어준다.

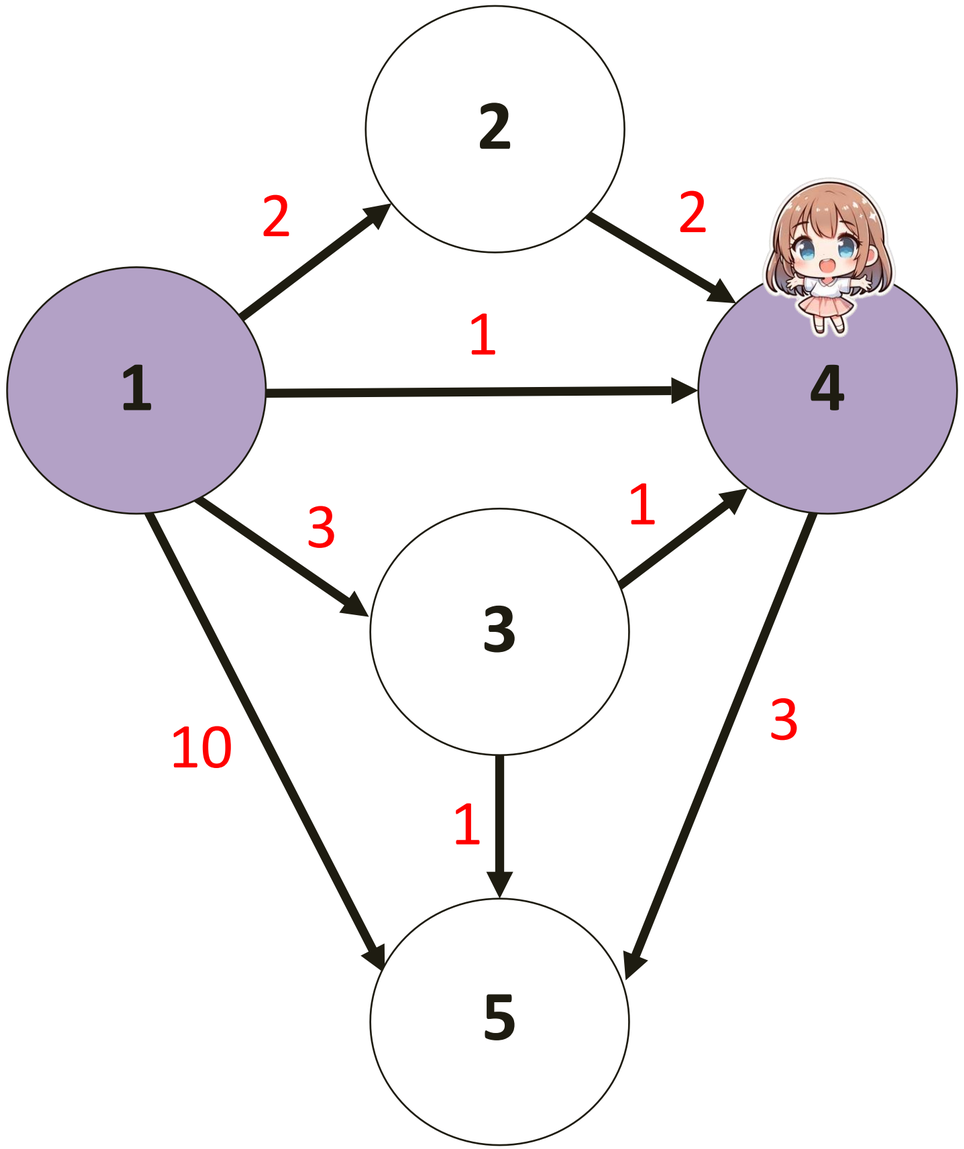
더 이상 방문할 정점이 없으므로
1번 정점에서 할 일을 끝낸다.

Heap(최소)

- 4번 정점 (비용 1)
- 2번 정점 (비용 2)
- 3번 정점 (비용 3)
- 5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	10
직전 정점	-1	1	1	1	1

다익스트라 알고리즘



힙에서 원소를 하나 빼온다.

정점 번호는 4이며, 이때 비용은 1이다.

4번 정점을 방문 처리 해준다.
(탐욕적 선택 속성때문에 가능함)

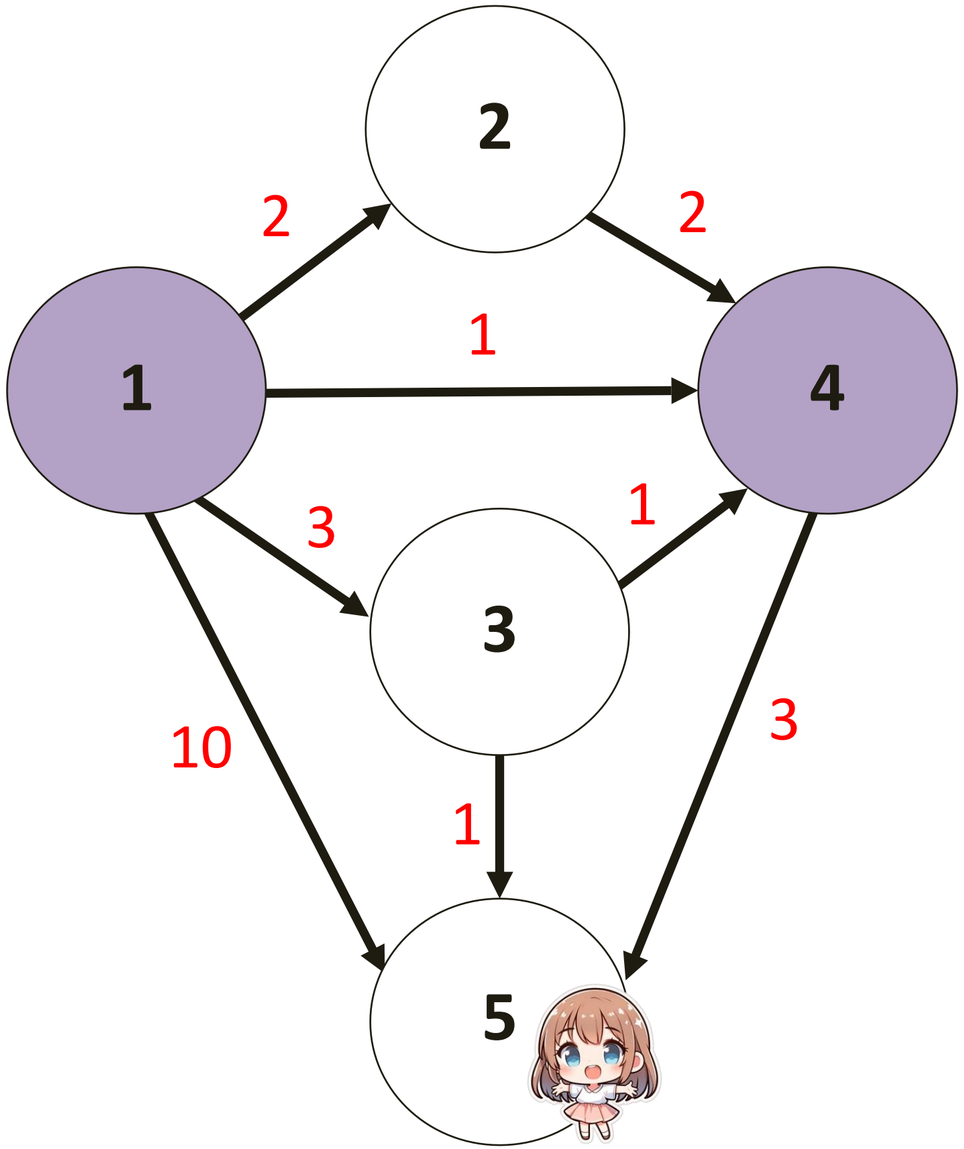
이후 4번 정점에서 갈 수 있는 간선을 검사해준다.

Heap(최소)

2번 정점 (비용 2)
3번 정점 (비용 3)
5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	10
직전 정점	-1	1	1	1	1

다익스트라 알고리즘



5번 정점은 4번 정점과 연결 되어있다.

간선의 가중치는 3이다.

갱신 비용 = 1 + 3 = 4

이때, 기존 비용보다 갱신 비용이 더 작으므로
(10 vs 4) 비용을 갱신해주고 힙에 넣어준다.

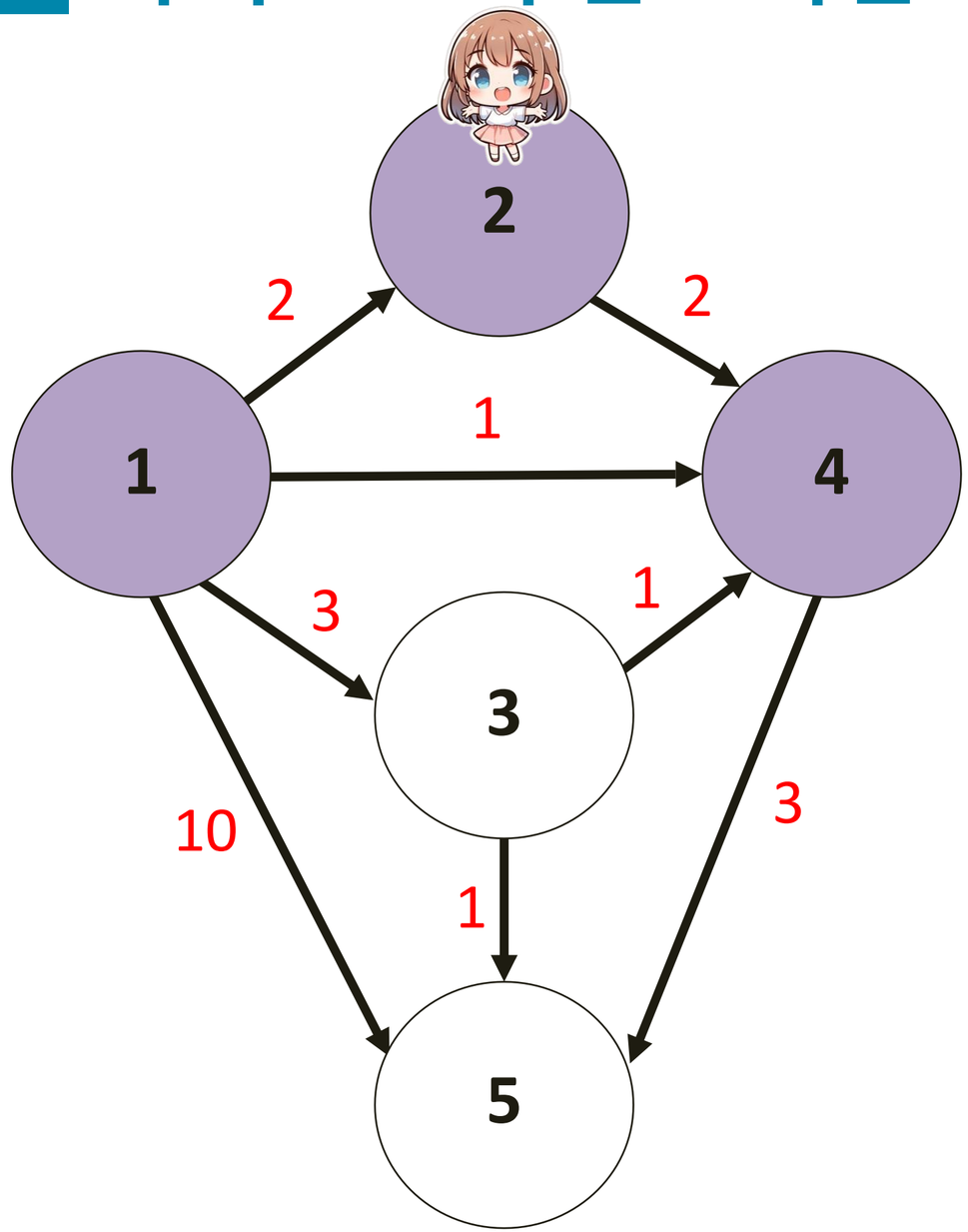
더 이상 방문할 정점이 없으므로
4번 정점에서 할 일을 끝낸다.

Heap(최소)

- 2번 정점 (비용 2)
- 3번 정점 (비용 3)
- 5번 정점 (비용 4)
- 5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



힙에서 원소를 하나 빼온다.

정점 번호는 2이며, 이때 비용은 2이다.

2번 정점을 방문 처리 해준다.
(탐욕적 선택 속성때문에 가능함)

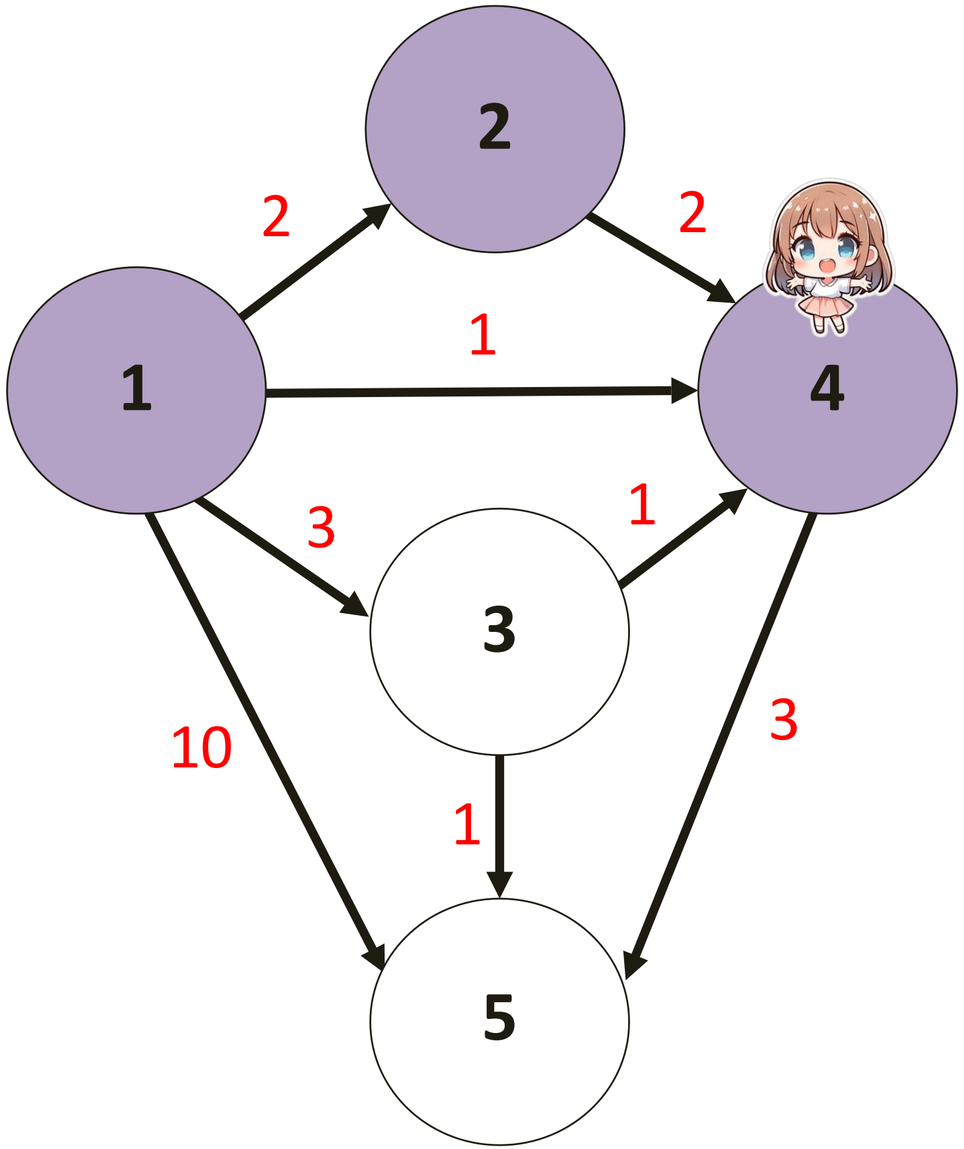
이후 2번 정점에서 갈 수 있는 간선을 검사해준다.

Heap(최소)

3번 정점 (비용 3)
5번 정점 (비용 4)
5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



4번 정점은 2번 정점과 연결 되어있다.

간선의 가중치는 2이다.

갱신 비용 = 2 + 2 = 4

그러나 기존 비용 보다 갱신 비용이 더 크므로
(1 vs 4) 아무것도 하지 않는다.

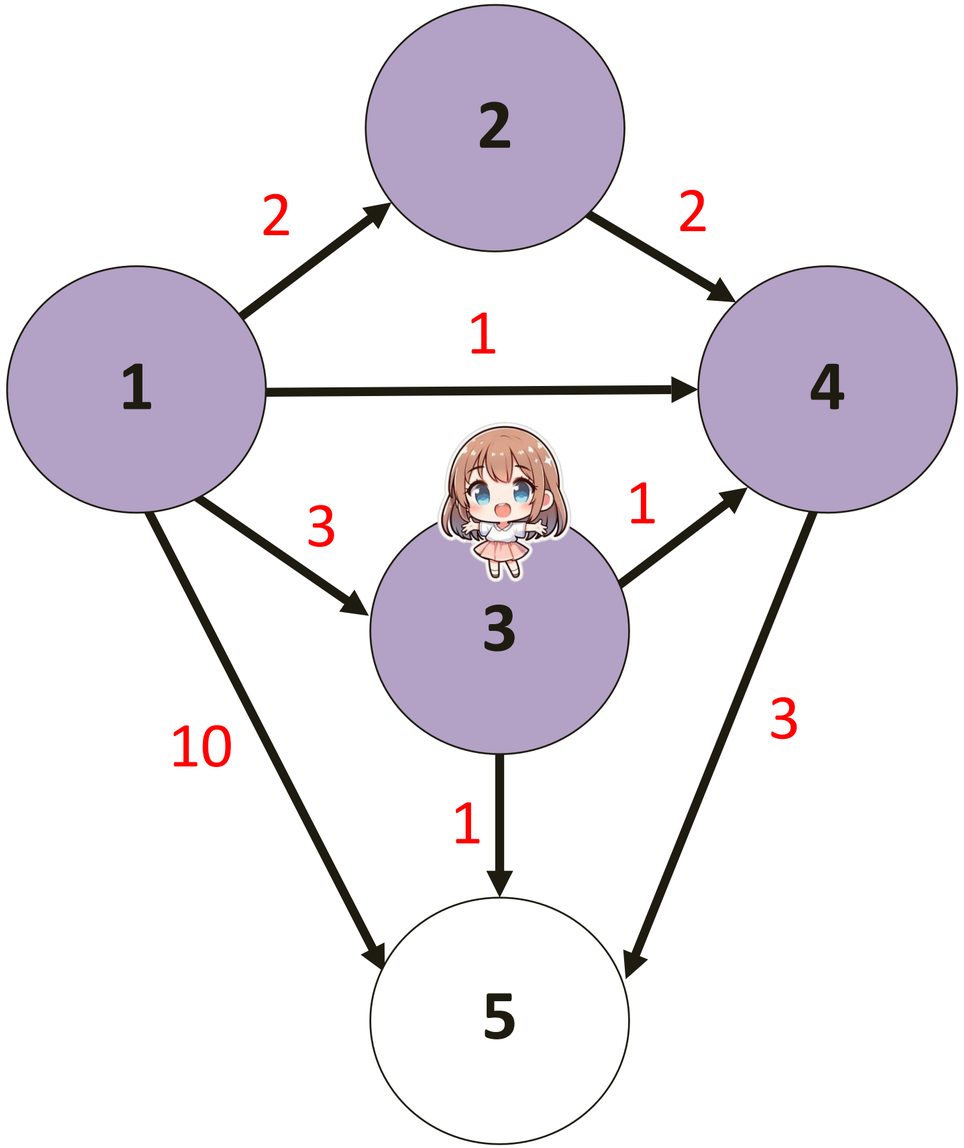
더 이상 방문할 정점이 없으므로
2번 정점에서 할 일을 끝낸다.

Heap(최소)

3번 정점 (비용 3)
5번 정점 (비용 4)
5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



힙에서 원소를 하나 빼온다.

정점 번호는 3이며, 이때 비용은 3이다.

3번 정점을 방문 처리 해준다.
(탐욕적 선택 속성때문에 가능함)

이후 3번 정점에서 갈 수 있는 간선을 검사해준다.

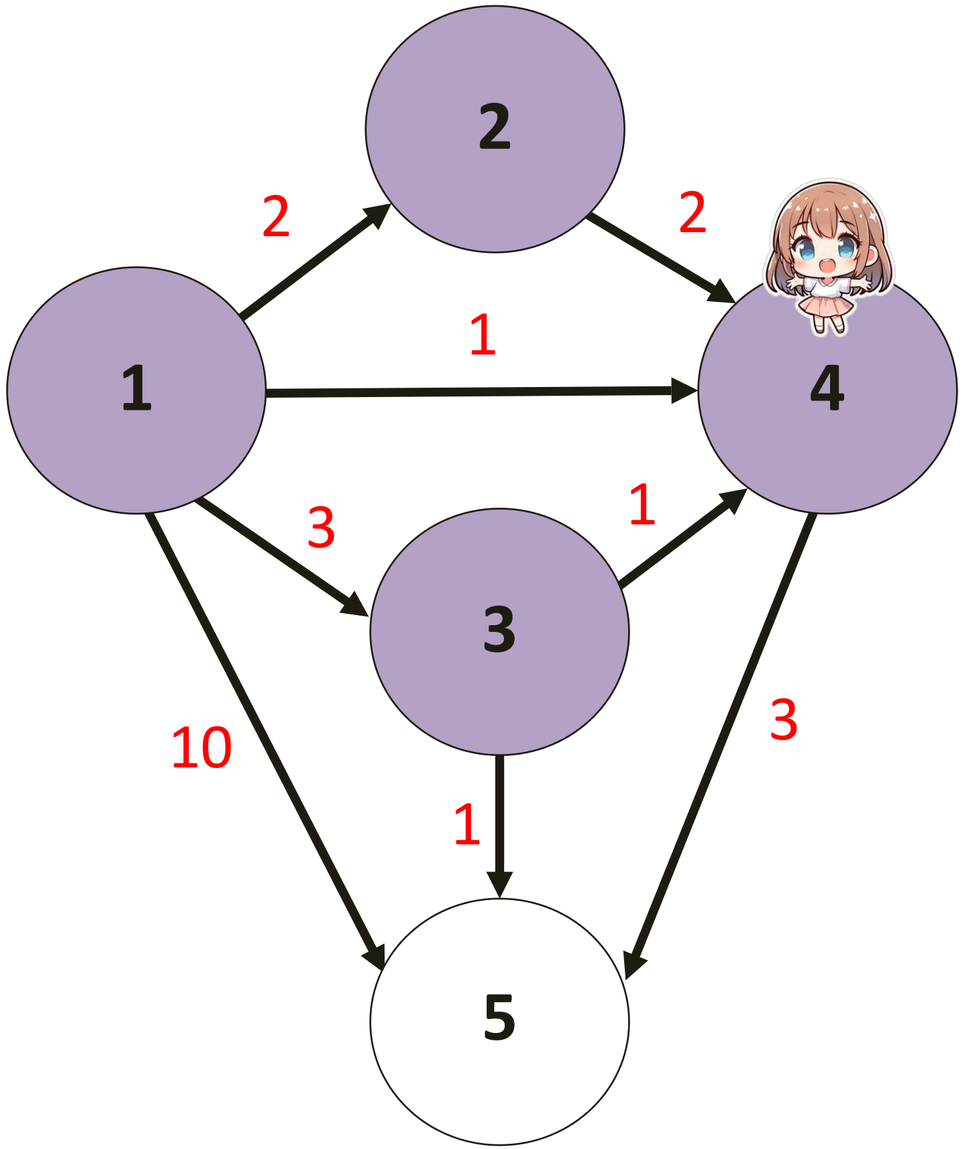
Heap(최소)

5번 정점 (비용 4)

5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



4번 정점은 3번 정점과 연결 되어있다.

간선의 가중치는 1이다.

갱신 비용 = 3 + 1 = 4

이때, 기존 비용보다 갱신 비용이 더 크므로
(1 vs 4) 아무것도 하지 않는다.

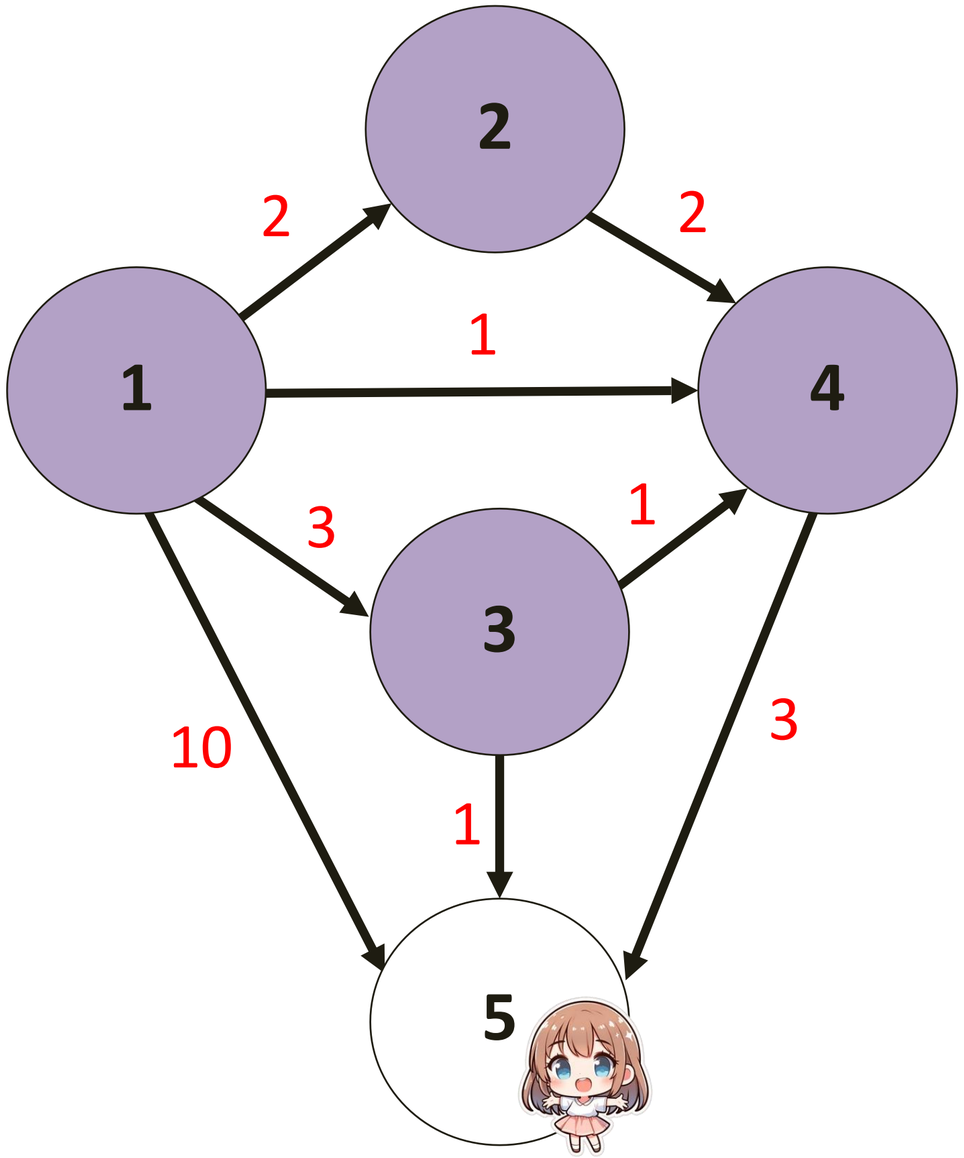
Heap(최소)

5번 정점 (비용 4)

5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



5번 정점은 3번 정점과 연결 되어있다.

간선의 가중치는 1이다.

갱신 비용 = 3 + 1 = 4

이때, 기존 비용보다 갱신 비용이 같으므로
(4 vs 4) 아무 짓도 하지 않는다.

넣어 줘도 괜찮지만 불필요한 작업을 늘리게 됨

더 이상 방문할 정점이 없으므로
3번 정점에서 할 일을 끝낸다.

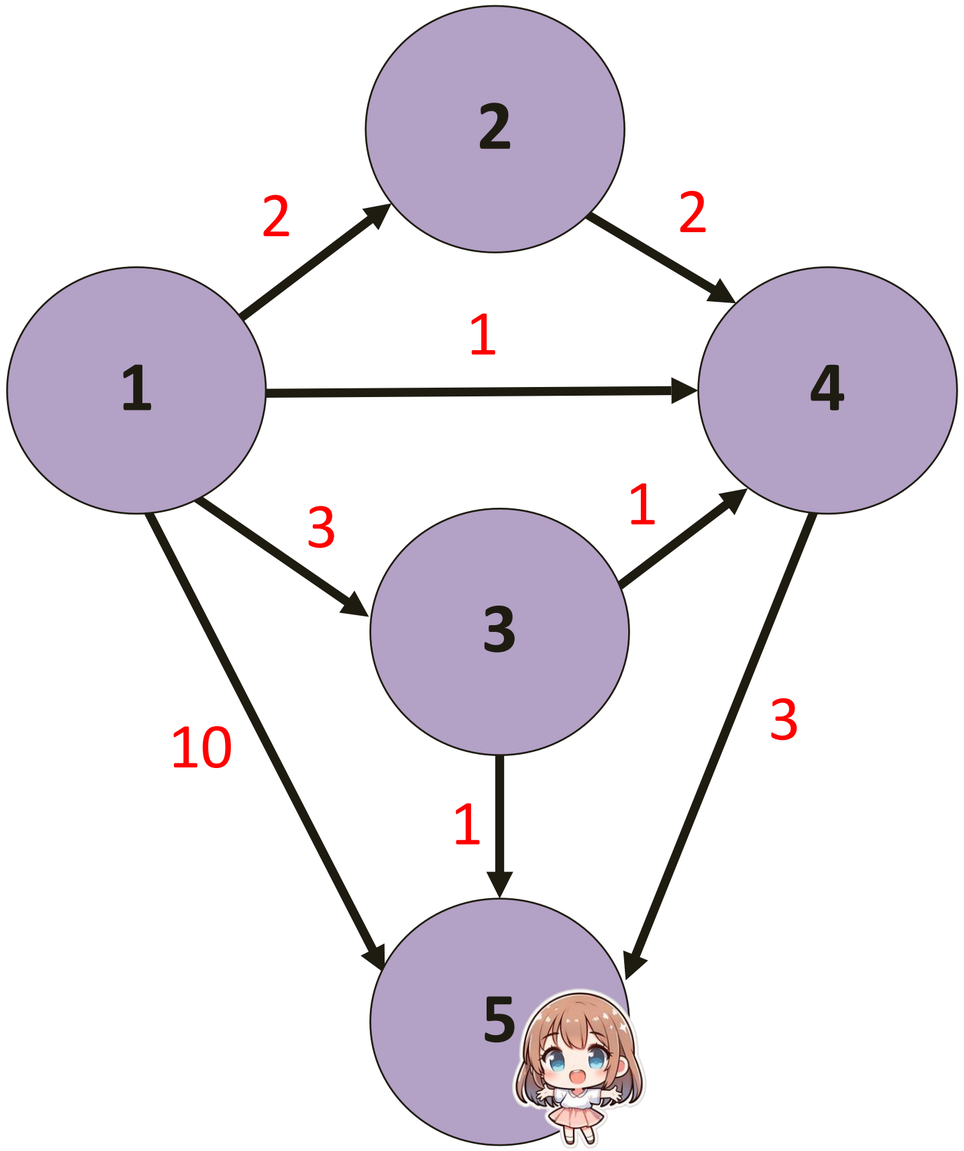
Heap(최소)

5번 정점 (비용 4)

5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



힙에서 원소를 하나 빼온다.

정점 번호는 5이며, 이때 비용은 4이다.

5번 정점을 방문 처리 해준다.
(탐욕적 선택 속성때문에 가능함)

이후 5번 정점에서 갈 수 있는 간선을 검사해준다.

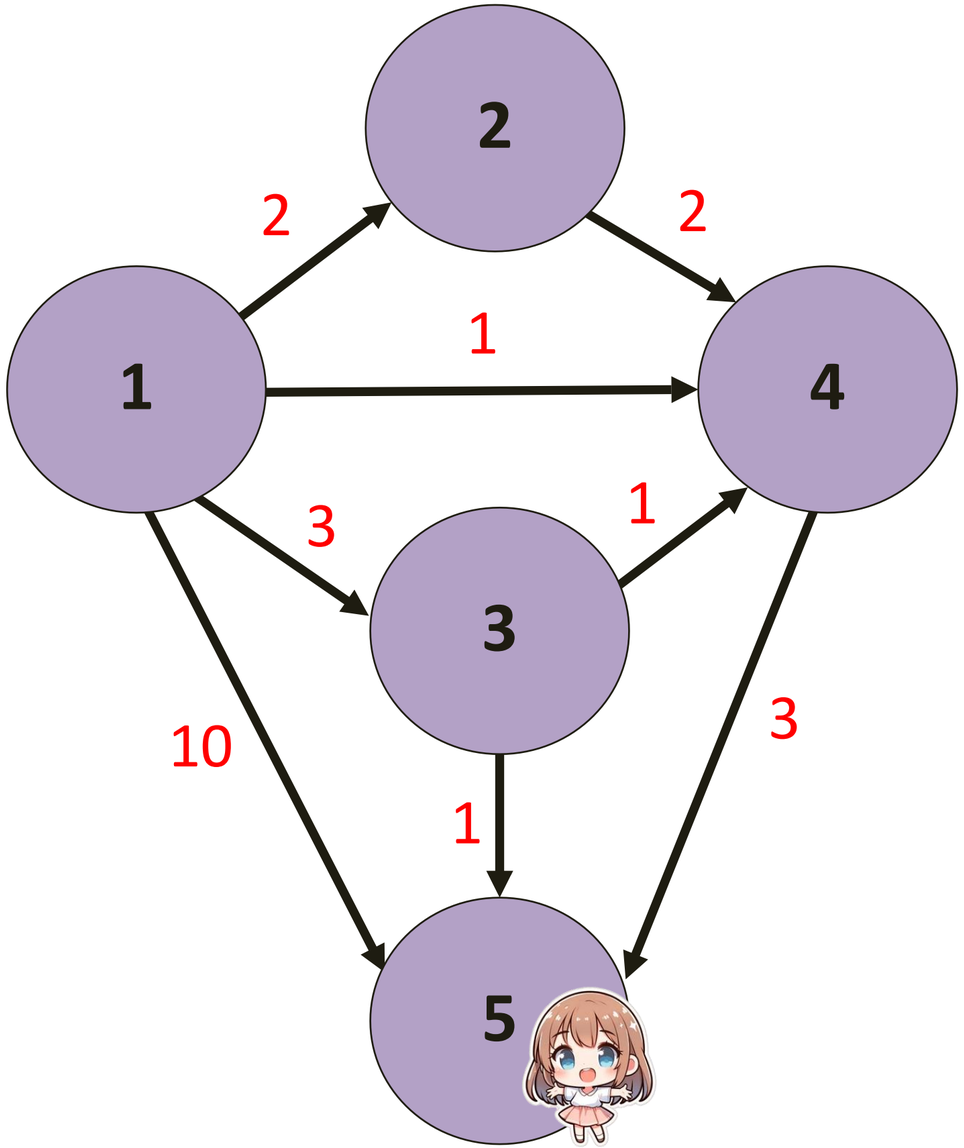
Heap(최소)

5번 정점 (비용 4)

5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



그러나 방문할 수 있는 정점이 없으므로(고립 정점)
5번 정점에서 할 일을 끝낸다.

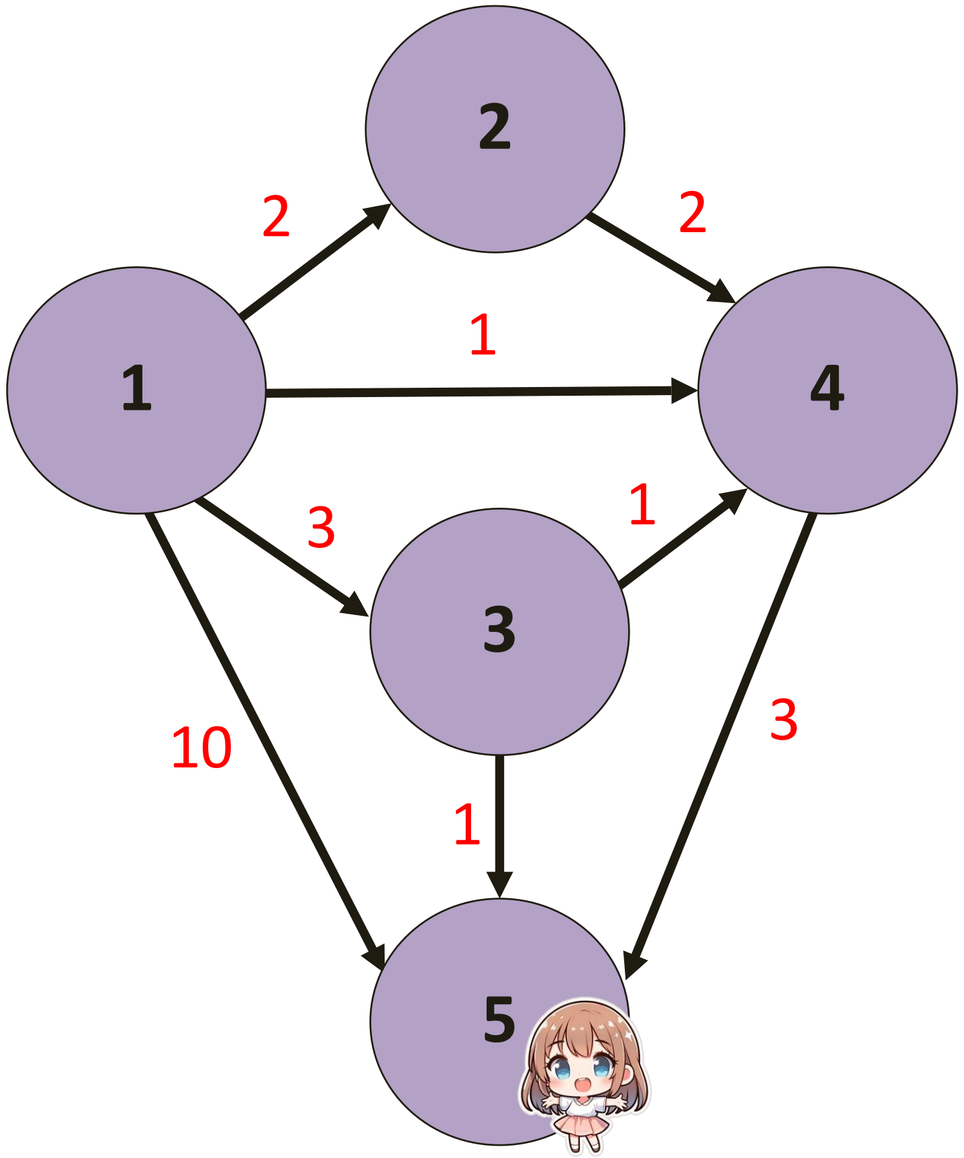
Heap(최소)

5번 정점 (비용 4)

5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

다익스트라 알고리즘



힙에서 원소를 하나 빼온다.

정점 번호는 5이며, 이때 비용은 10이다.

그러나, 이미 5번 정점에서의 최소 비용은 4 이다.
현재 노드가 가진 비용보다 작으므로 이 노드로
갱신을 할 필요가 없기 때문에 무시한다.

힙이 비었으므로 다익스트라 알고리즘이 종료된다.

결론적으로 1번 정점에서 5번 정점까지
이동하는 최소 비용은 '4' 이다.

* TIP!! *

시작 정점으로부터 임의 정점까지의 경로를 추적
하고싶다면?

->임의 정점의 직전 정점을 이용하여 백트래킹을
하면된다! (-1이 나올때 까지 타고 넘어가기)

Heap(최소)

5번 정점 (비용 4)

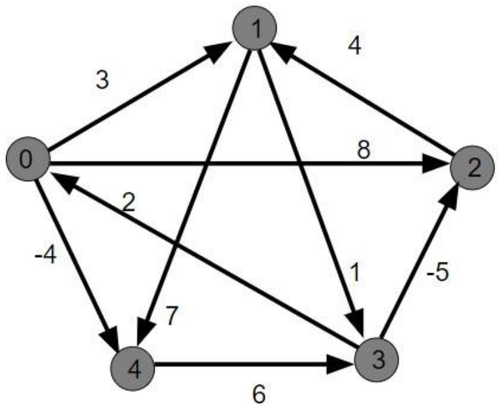
5번 정점 (비용 10)

정점 번호	1	2	3	4	5
최소 비용	0	2	3	1	4
직전 정점	-1	1	1	1	4

플로이드-워셜 알고리즘

플로이드-워셜 알고리즘은 그래프에서 모든 정점 쌍 간의 최단 경로를 찾는 알고리즘이다.
이 알고리즘은 동적 계획법(dp)을 사용하여 각 정점 쌍 간의 최단 경로를 점진적으로 계산하며, 그래프의 모든 경로를 고려해 최단 거리를 구한다.
시간 복잡도는 O(V^3)로, 그래프의 정점 수가 많을 경우 성능이 저하될 수 있지만, 간단한 구현과 모든 쌍의 최단 경로를 정확하게 계산할 수 있다는 장점을 가진다.
플로이드-워셜 알고리즘은 가중치가 음수인 경우에도 안정적으로 작동하는 특성이 있다.

Example: input



W

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

플로이드-워셜 알고리즘

문제

n(2 ≤ n ≤ 100)개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 m(1 ≤ m ≤ 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 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 n이 주어지고 둘째 줄에는 버스의 개수 m이 주어진다. 그리고 셋째 줄부터 m+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 버스의 정보는 버스의 시작 도시 a, 도착 도시 b, 한 번 타는데 필요한 비용 c로 이루어져 있다. 시작 도시와 도착 도시가 같은 경우는 없다. 비용은 100,000보다 작거나 같은 자연수이다.

시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐 수 있다.

출력

n개의 줄을 출력해야 한다. i번째 줄에 출력하는 j번째 숫자는 도시 i에서 j로 가는데 필요한 최소 비용이다. 만약, i에서 j로 갈 수 없는 경우에는 그 자리에 0을 출력한다.

n개의 도시가 있으며, 모든 시작 도시로부터 도착 도시까지 가는데 드는 최소 버스 비용을 구해야 하는 문제이다.

먼저 모든 칸을 INF로 초기화한 인접 행렬을 만들어 준다.

이후, 정점과 간선 정보를 입력 받고 인접 행렬을 채워준다.

플로이드-워셜 알고리즘을 통해 모든 시작 정점으로 부터 다른 정점까지의 최단 비용(가중치 합)을 구한다.

플로이드-워셜 알고리즘

	1번	2번	3번	4번	5번
1번	INF	INF	INF	INF	INF
2번	INF	INF	INF	INF	INF
3번	INF	INF	INF	INF	INF
4번	INF	INF	INF	INF	INF
5번	INF	INF	INF	INF	INF

정점 개수 * 정점 개수 만큼 인접 행렬로 만들어 주고,

모두 INF 값으로 초기화 해준다.

플로이드-워셜 알고리

	1번	2번	3번	4번	5번
1번	0	INF	INF	INF	INF
2번	INF	0	INF	INF	INF
3번	INF	INF	0	INF	INF
4번	INF	INF	INF	0	INF
5번	INF	INF	INF	INF	0

자기 자신으로 가는 비용은 당연히 '0'이다.

고로 주 대각선을 전부 0으로 초기화 해준다.

플로이드-워셜 알고리즘

	1번	2번	3번	4번	5번
1번	0	2	3	1	10
2번	INF	0	INF	2	INF
3번	8	INF	0	1	1
4번	INF	INF	INF	0	3
5번	7	4	INF	INF	0

입력에 맞춰서 인접 행렬을 갱신 해준다.

이때, 시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐수 있기 때문에 min 함수를 이용하여 갱신해준다.

```
5
14
1 2 2
1 3 3
1 4 1
1 5 10
2 4 2
3 4 1
3 5 1
4 5 3
3 5 10
3 1 8
1 4 2
5 1 7
3 4 2
5 2 4
```

플로이드-워셜 알고리즘

	1번	2번	3번	4번	5번
1번	0	2	3	1	10
2번	INF	0	INF	2	INF
3번	8	INF	0	1	1
4번	INF	INF	INF	0	3
5번	7	4	INF	INF	0

입력에 있는 표를 모두 갱신해 주었다.

이제 알고리즘을 써서 표를 갱신 할 것이다. 전략은 아래와 같다.

임의의 정점을 경유 지점으로 둔다. mid 라고 하자.

또한 시작 정점을 start, 도착 정점을 end로 두자.

그러면 adj_matrix[start][end] 와,

k를 거쳐가는 adj_matrix[start][mid] + adj_matrix[mid][end]의 값을 비교 하여 더 작은 것으로 갱신 해주면 된다.

어떤 정점을 mid로 설정해야 최적 경로가 나오는지 모르므로
1번 정점부터 n번 정점까지 한번씩 경유지로 설정한다.
즉, 경유지 한번씩 순회 = O(V), 인접 행렬 순회 = O(V^2) 이므로
총 시간 복잡도는 O(V^3)

주의점
adj_matrix[start][mid] + adj_matrix[mid][end] 오버플로우에 주의 한다!

플로이드-워셜 알고리즘

	1번	2번	3번	4번	5번
1번	0	2	3	1	10
2번	INF	0	INF	2	INF
3번	8	10	0	1	1
4번	INF	INF	INF	0	3
5번	7	4	10	8	0

mid : 1번 정점

플로이드-워셜 알고리즘

	1번	2번	3번	4번	5번
1번	0	2	3	1	4
2번	INF	0	INF	2	INF
3번	8	10	0	1	1
4번	INF	INF	INF	0	3
5번	7	4	10	6	0

mid : 2번 정점

플로이드-워셜 알고리즘

	1번	2번	3번	4번	5번
1번	0	2	3	1	4
2번	INF	0	INF	2	INF
3번	8	10	0	1	1
4번	INF	INF	INF	0	3
5번	7	4	10	6	0

mid : 3번 정점

플로이드-워셜 알고리즘

	1번	2번	3번	4번	5번
1번	0	2	3	1	4
2번	INF	0	INF	2	5
3번	8	10	0	1	1
4번	INF	INF	INF	0	3
5번	7	4	10	6	0

mid : 4번 정점

플로이드-워셜 알고리즘

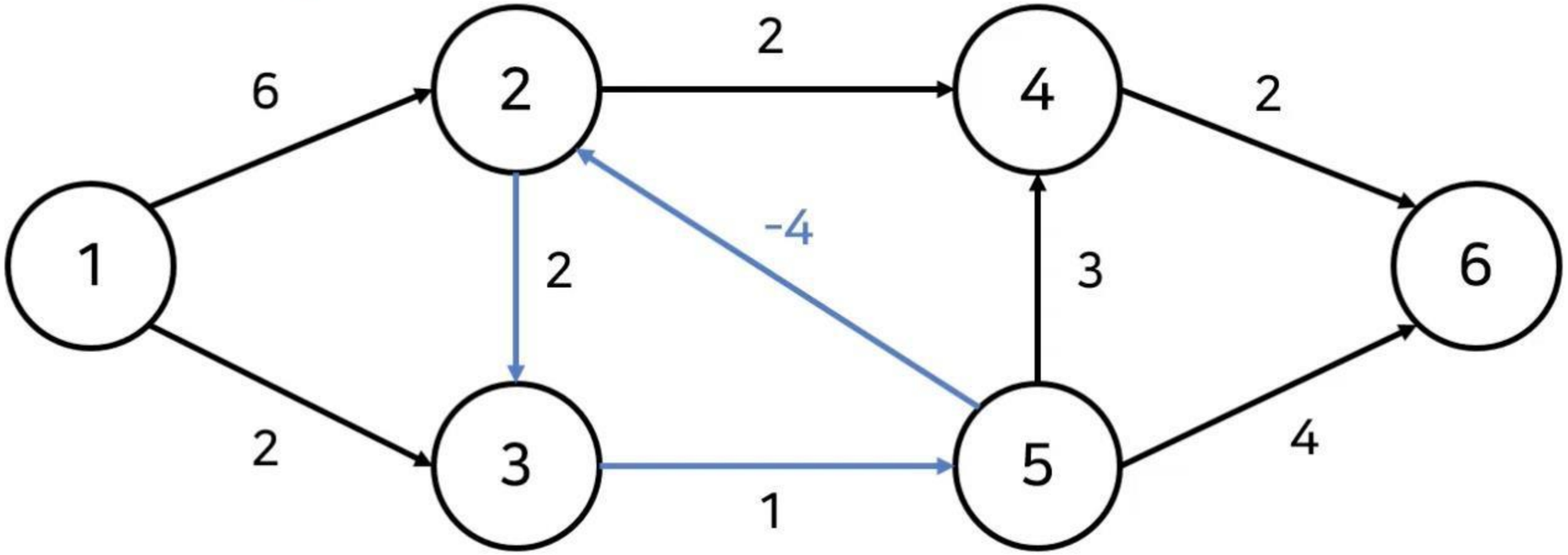
	1번	2번	3번	4번	5번
1번	0	2	3	1	4
2번	12	0	15	2	5
3번	8	5	0	1	1
4번	10	7	13	0	3
5번	7	4	10	6	0

mid : 5번 정점

플로이드-워셜은 정점과 정점과의 모든 경로를 고려하기 때문에 다소 느리지만 모든 정점으로부터 최단 거리를 구할 수 있다!
(정점에 비해 간선이 매우 많을 때 사용하면 유용하다)

벨만-포드 알고리즘

벨만-포드 알고리즘은 그래프에서 단일 출발점에서 모든 다른 정점까지의 최단 경로를 찾는 알고리즘이다.
다익스트라 알고리즘과 달리, 벨만-포드 알고리즘은 가중치가 음수인 그래프에서도 안정적으로 작동하며, 음수 사이클(음수 가중치를 가진 경로를 순환)도 감지할 수 있다는 장점이 있다. 이 알고리즘은 각 간선을 반복적으로 확인하여 최단 거리를 업데이트하는 방식으로 시간 복잡도는 O(VE)로 다소 느리지만, 다익스트라 알고리즘이 적용될 수 없는 음수 가중치 그래프에서 중요한 역할을 한다.



벨만-포드 알고리즘

N개의 도시가 있다. 그리고 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 = 0인 경우는 순간 이동을 하는 경우, C < 0인 경우는 타임머신으로 시간을 되돌아가는 경우이다.

1번 도시에서 출발해서 나머지 도시로 가는 가장 빠른 시간을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 N (1 ≤ N ≤ 500), 버스 노선의 개수 M (1 ≤ M ≤ 6,000)이 주어진다. 둘째 줄부터 M개의 줄에는 버스 노선의 정보 A, B, C (1 ≤ A, B ≤ N, -10,000 ≤ C ≤ 10,000)가 주어진다.

출력

만약 1번 도시에서 출발해 어떤 도시로 가는 과정에서 시간을 무한히 오래 전으로 되돌릴 수 있다면 첫째 줄에 -1을 출력한다. 그렇지 않다면 N-1개 줄에 걸쳐 각 줄에 1번 도시에서 출발해 2번 도시, 3번 도시, ..., N번 도시로 가는 가장 빠른 시간을 순서대로 출력한다. 만약 해당 도시로 가는 경로가 없다면 대신 -1을 출력한다.

n개의 도시가 있으며, 특정 시작 도시로부터
도착 도시까지 가는데 드는 최소 버스 비용을 구해야하는 문제이다.

그러나 이 문제는 간선의 가중치가 음수인 것이 있어서
다익스트라 알고리즘을 사용하지 못한다.

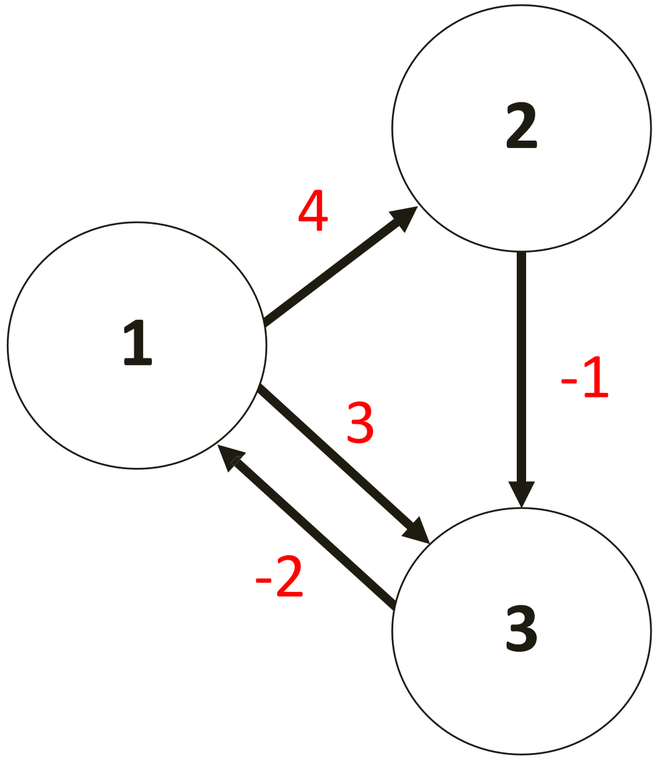
먼저 정점과 간선 정보를 입력 받은 뒤 그래프를 생성 합니다.

이후 벨만 포드 알고리즘을 통해 최단거리를 구해줍니다.

벨만-포드 알고리즘

예제 입력 1 복사

```
3 4
1 2 4
1 3 3
2 3 -1
3 1 -2
```

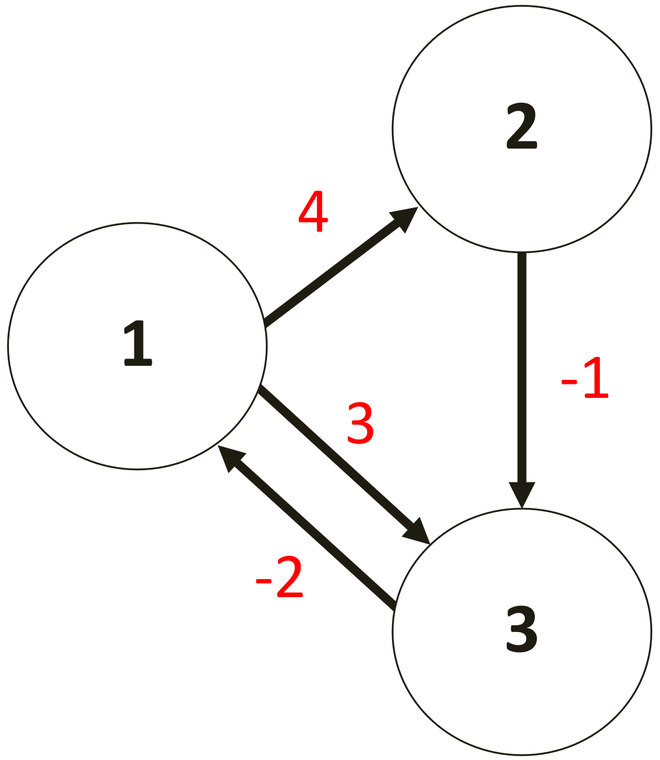


1 – 임의의 정점 부터 최소 비용을 점진적으로 갱신해 나가기

2 – 플로이드 워셜과 유사한 갱신 방식

3 – 음수 사이클을 검출할 수 있음! (사이클을 돌면 돌수록 가중치가 작아지는 경로)
-> 문제에서 나온 설명중 “시간을 무한정으로 돌릴 수 있다” == 음수 사이클이 존재

벨만-포드 알고리즘



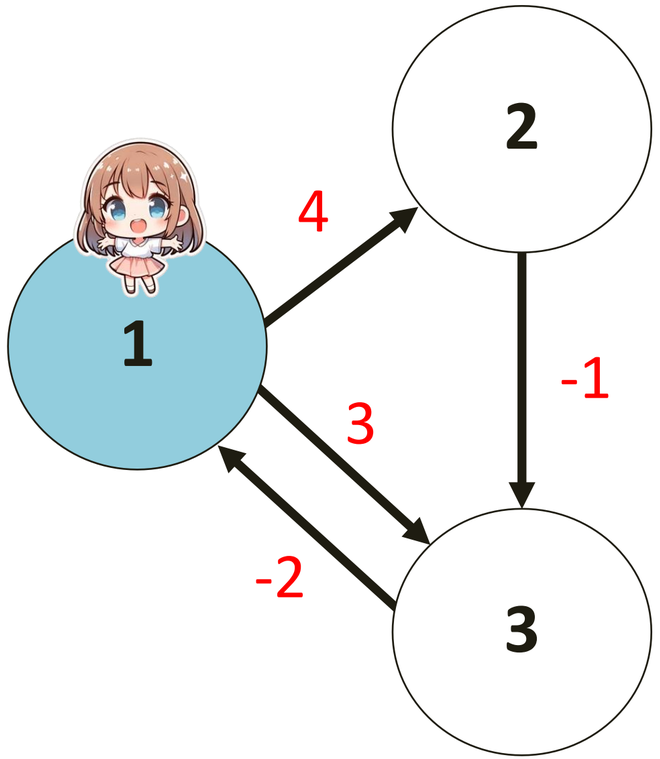
아직 탐색을 시작하기 전이다.

모든 정점에 대한 최소 비용은 모두 무한대로 초기화 해준다.
이때, 시작 지점인 1번 정점은 0으로 초기화 해준다.

플로이드-워셜 알고리즘과 유사하게 1번 정점 부터 n 번 정점까지
모든 간선을 검사하여 최소 비용을 점진적으로 갱신해준다.

정점 번호	1	2	3
최소 거리	0	INF	INF

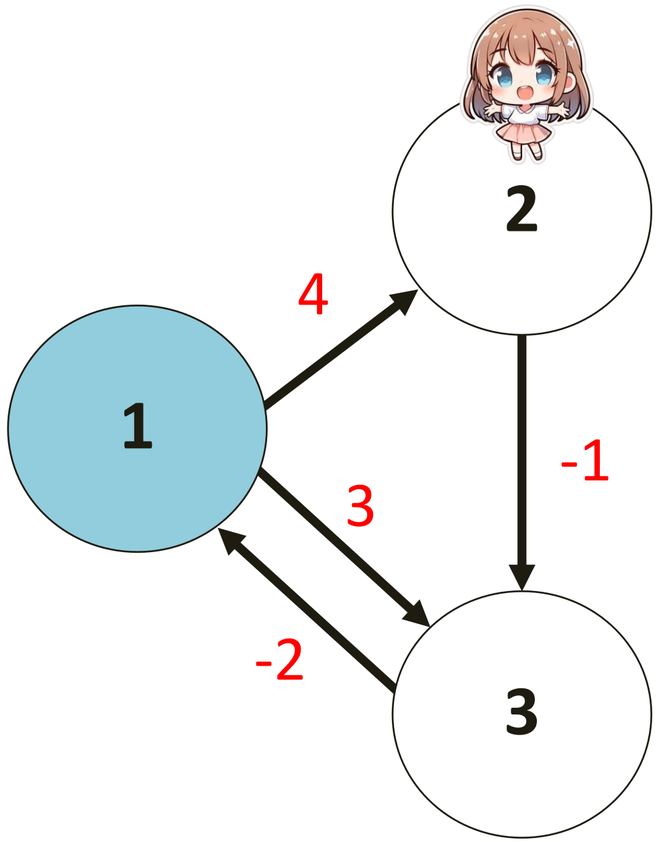
벨만-포드 알고리즘 문제 풀이 예시



1번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	0	INF	INF

벨만-포드 알고리



2번 정점은 1번 정점과 연결 되어있다.

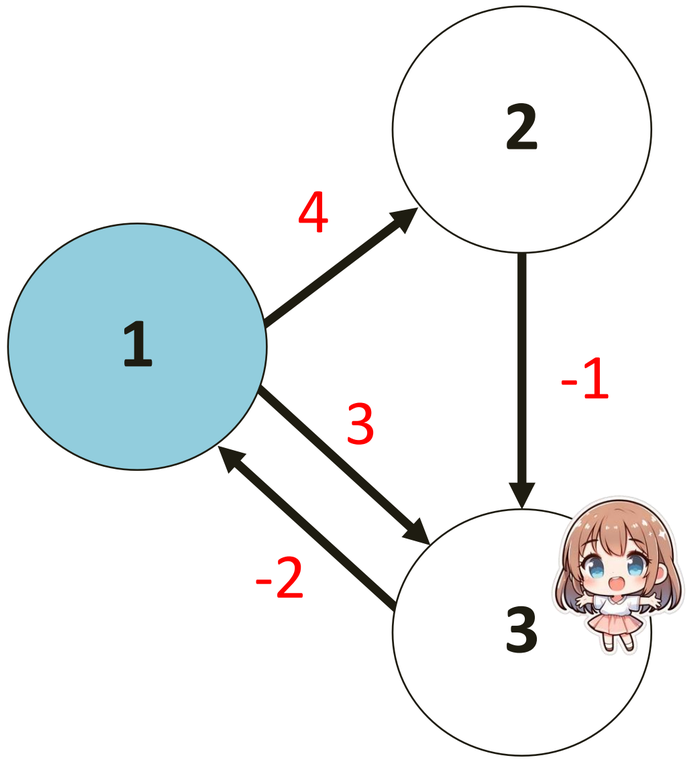
간선의 가중치는 4이다.

갱신 비용 = 0 + 4 = 4

이때, 기존 비용보다 갱신 비용이 작으므로
(INF vs 4) 갱신해준다.

정점 번호	1	2	3
최소 거리	0	4	INF

벨만-포드 알고리즘



3번 정점은 1번 정점과 연결 되어있다.

간선의 가중치는 3이다.

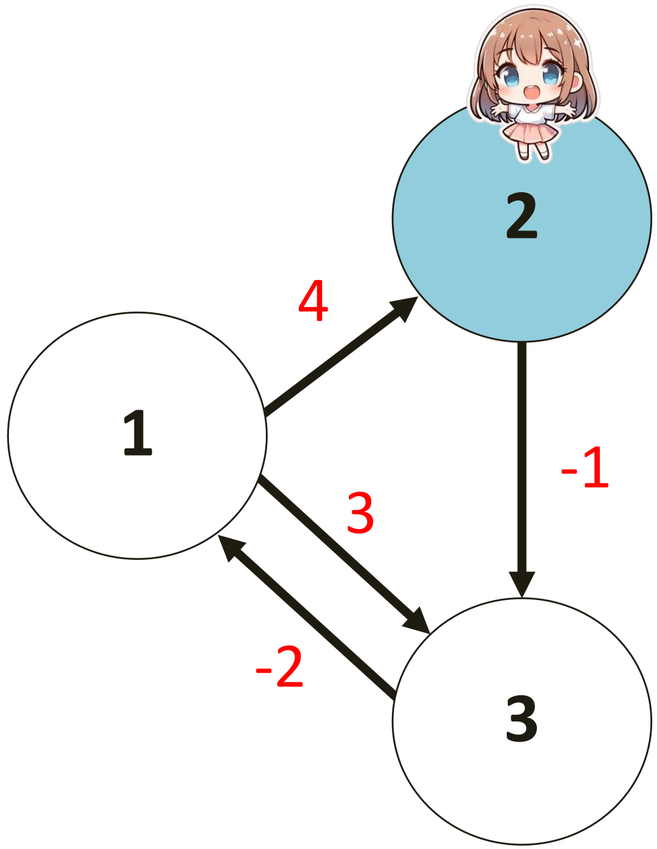
갱신 비용 = 0 + 3 = 3

이때, 기존 비용보다 갱신 비용이 작으므로
(INF vs 3) 갱신해준다.

더 이상 방문할 정점이 없으므로 1번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	0	4	3

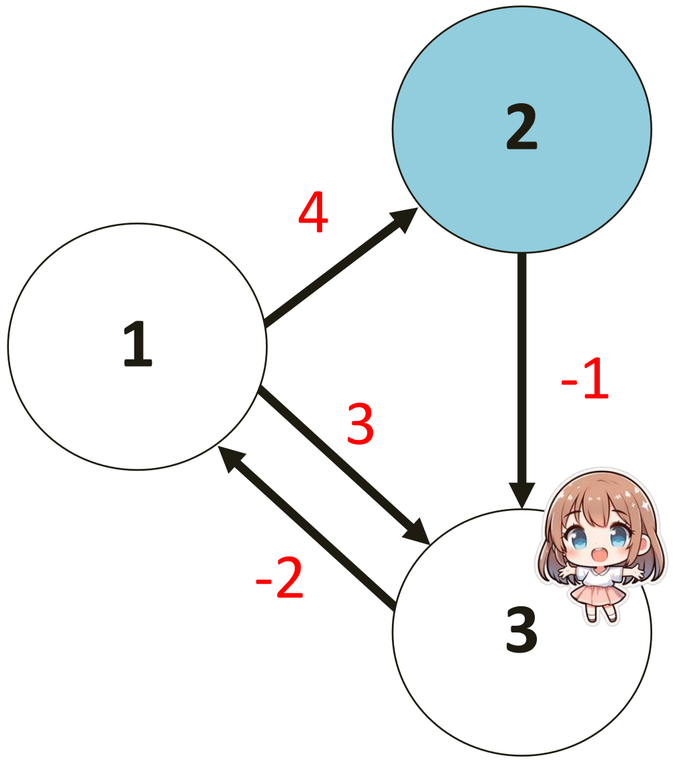
벨만-포드 알고리즘



2번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	0	4	3

벨만-포드 알고리즘



3번 정점은 2번 정점과 연결 되어있다.

간선의 가중치는 -1이다.

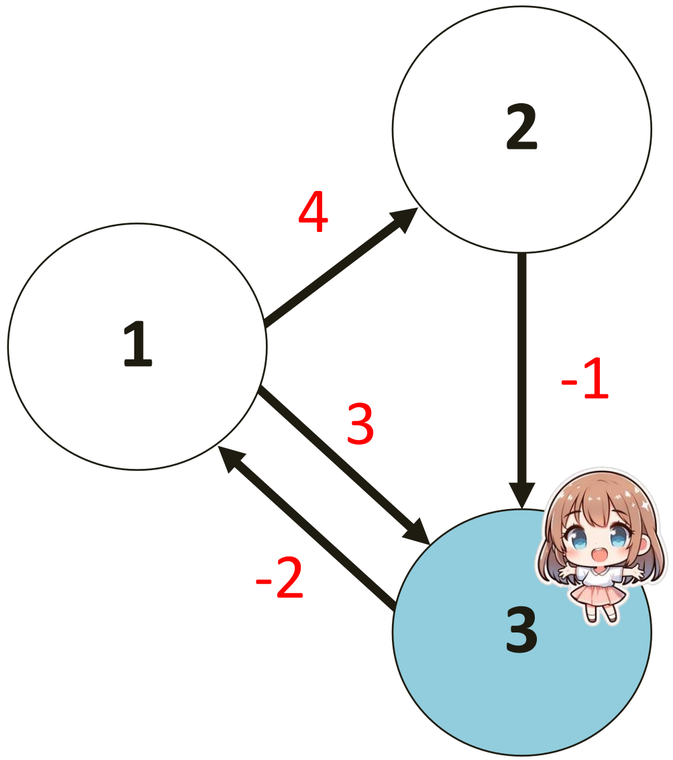
갱신 비용 = 4 - 1 = 3

이때, 기존 비용보다 갱신 비용이 같으므로
(3 vs 3) 갱신하지 않는다.

더 이상 방문할 정점이 없으므로 2번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	0	4	3

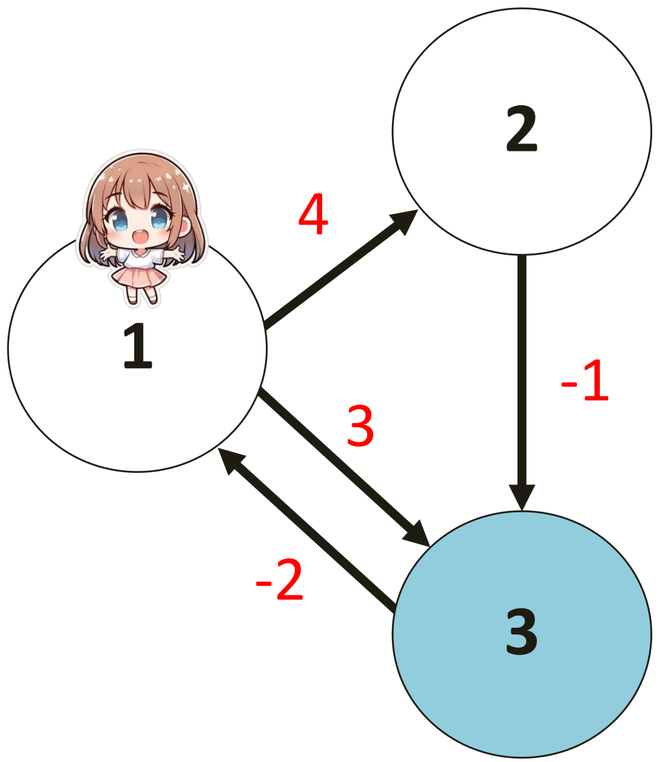
벨만-포드 알고리즘



3번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	0	4	3

벨만-포드 알고리즘



1번 정점은 3번 정점과 연결 되어있다.

간선의 가중치는 -2이다.

갱신 비용 = 3 - 2 = 1

이때, 기존 비용보다 갱신 비용이 크므로
(0 vs 1) 갱신하지 않는다.

더 이상 방문할 정점이 없으므로 3번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	0	4	3

이러한 경로 갱신작업을 총 V-1번 반복하면 된다!

왜 V-1번 반복해야하나?

가정

1 - 그래프 G는 V개의 정점과 E개의 간선으로 구성되어 있다고 하자.
2 – 최단 경로는 출발점 s 에서 다른 모든 정점 e로의 경로를 의미한다고 하자.
3 – 그래프에는 음수 가중치의 간선이 있을 수 있지만, 음수 사이클은 없다고 가정하자. 음수 사이클이 있을 경우, 최단경로는 무한히 작아질 수 있기 때문.

증명

1. : 최단 경로에서 한 정점 s 에서 다른 정점 e로 가는 경로는 최대 V-1개의 간선을 가질 수 있음.
   -> 왜냐하면 사이클이 없는 그래프에서 최대 경로 길이가 V-1이기 때문
2. : V-1개의 간선을 지나면 경로는 최대 V개의 정점을 포함하게 되며, 더 이상 정점을 추가할 수 없음

1. : 벨만-포드 알고리즘은 각 반복에서 모든 간선을 한 번씩 검사하고, 각 간선에 대해 최단 거리를 갱신함
   -> 첫 번째 반복에서는, s에서 한 간선으로 직접 연결된 정점들의 최단 거리가 초기화 된다.
   -> 두 번째 반복에서는, s에서 두개의 간선을 거쳐 도달할 수 있는 정점들의 최단 거리가 갱신됨
2. : 이러한 과정을 V-1번 반복하면, 최대 V-1개의 간선을 통해 연결된 모든 경로가 최단 경로로 갱신됨

1. : V-1번의 반복 후에는, s에서 e로가는 모든 가능한 최단 경로가 고려됨.
   -> 최단 경로가 최대 V-1개의 간선을 가질 수 있기 때문!
2. – 만약 V-1번 이상 반복이 필요하다면, 이는 음수 사이클이 존재하는 경우다.
   ->알고리즘은 반복되고, 계속해서 거리가 줄어들게 됨

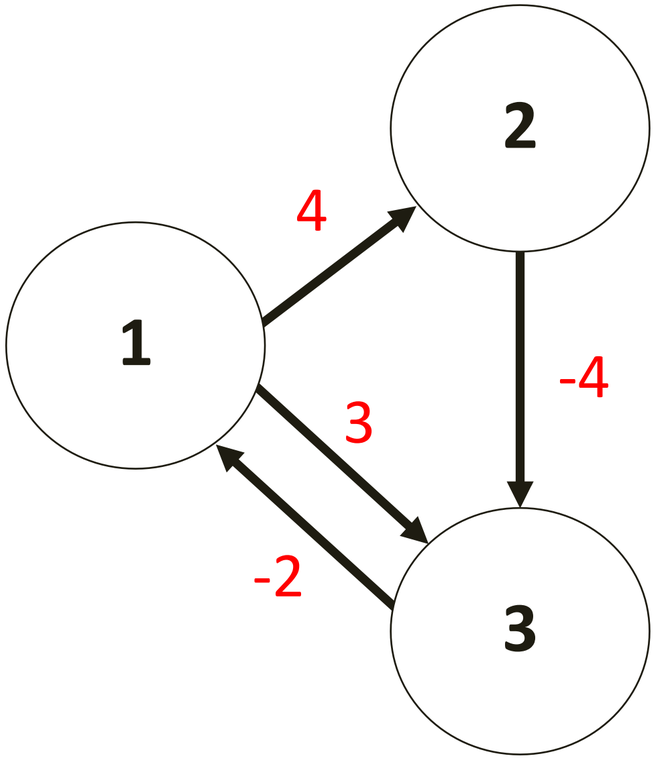
1. : 만약 V-1번의 반복이 충분하지 않다고 가정한다면?
   ->V-1번 반복 이후에도 일부 최단 경로가 갱신되지 않는 정점이 있다.
2. : 그러나, 이 정점이 s로부터 연결된 최단 경로를 갖는다면 ,그 경로는 최대 V-1개의 간선을 포함해야 하므로 이미 V-1번 반복 내에서 갱신 되었어야함.

고로, 귀류법에 의해 V-1번의 반복이 충분하지 않다는 가정은 모순이며, V-1번의 반복은 최단 경로를 계산하는 데 충분하다.

음수 사이클 검출하기

예제 입력 2 복사

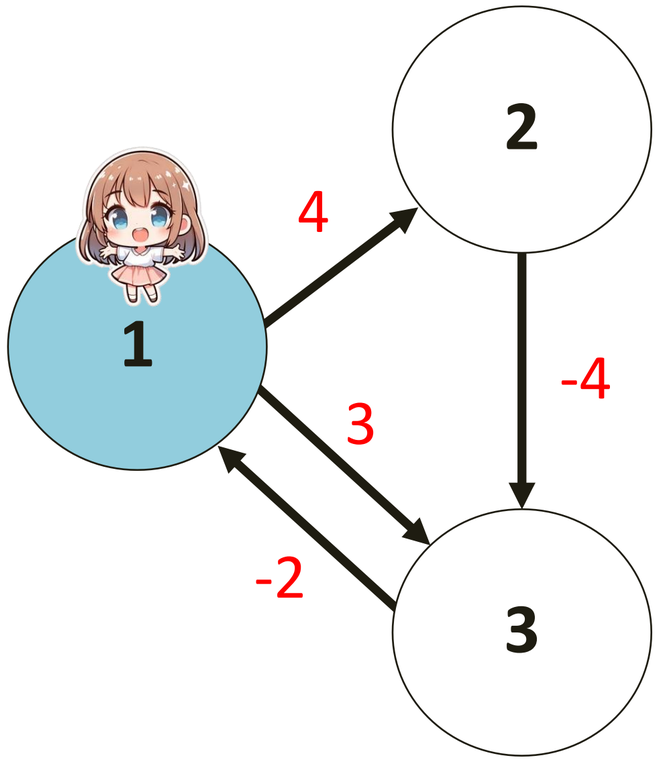
```
3 4
1 2 4
1 3 3
2 3 -4
3 1 -2
```



정점 번호	1	2	3
최소 거리	-4	2	-2

이미 V-1번 돌린 최소거리 표

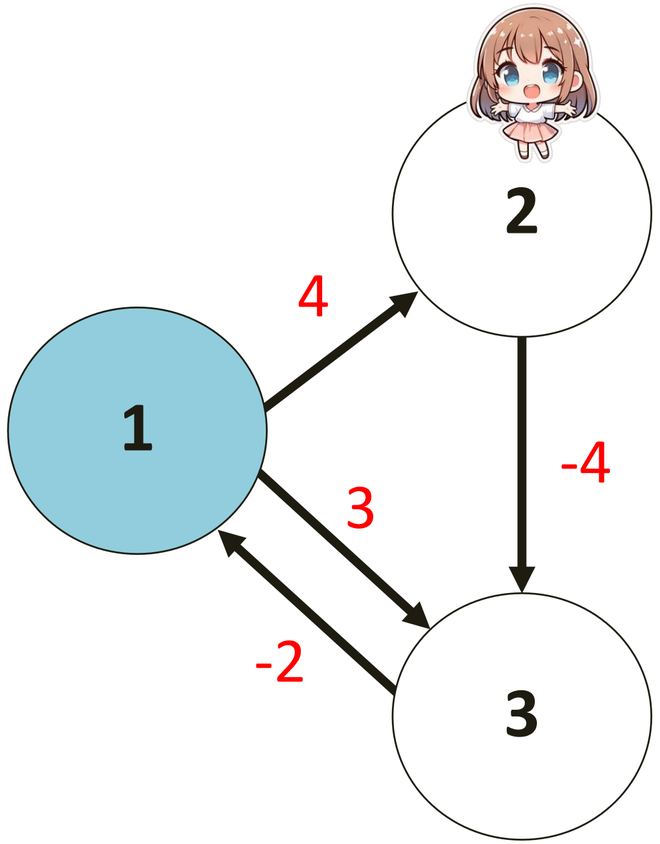
음수 사이클 검출하기 (v번째)



1번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	-4	2	-2

음수 사이클 검출하기 (v번째)



2번 정점은 1번 정점과 연결 되어있다.

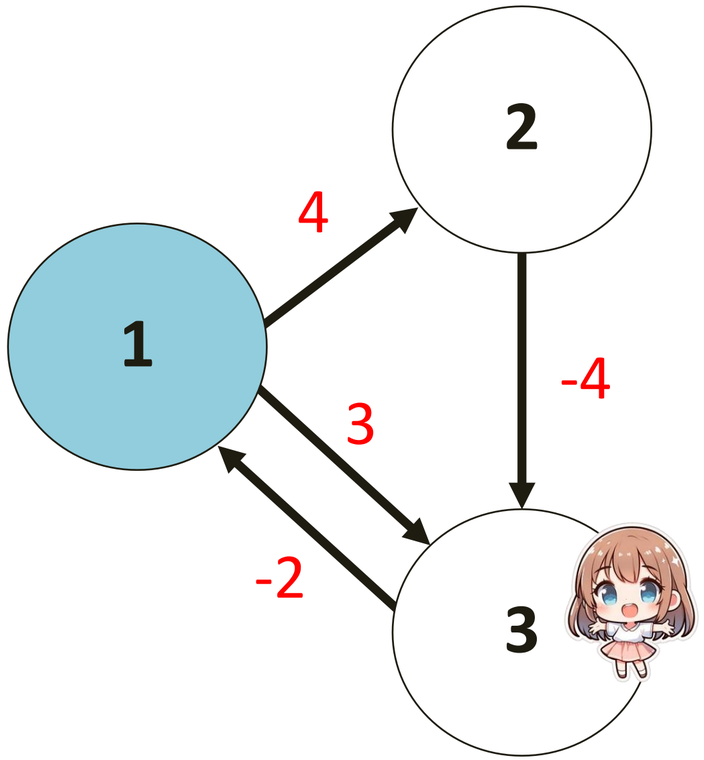
간선의 가중치는 4이다.

갱신 비용 = -4 + 4 = 0

이때, 기존 비용보다 갱신 비용이 작으므로
(2 vs 0) 갱신해준다.

정점 번호	1	2	3
최소 거리	-4	0	-2

음수 사이클 검출하기 (v번째)



3번 정점은 1번 정점과 연결 되어있다.

간선의 가중치는 3이다.

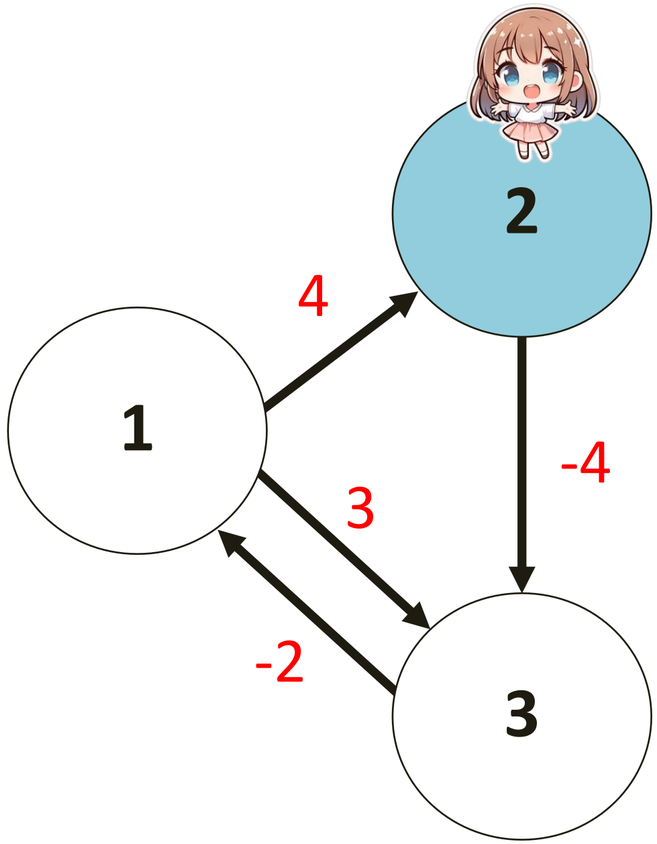
갱신 비용 = -4 + 3 = -1

이때, 기존 비용보다 갱신 비용이 크므로
(-2 vs -1) 갱신하지 않는다.

더 이상 방문할 정점이 없으므로 1번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	-4	0	-2

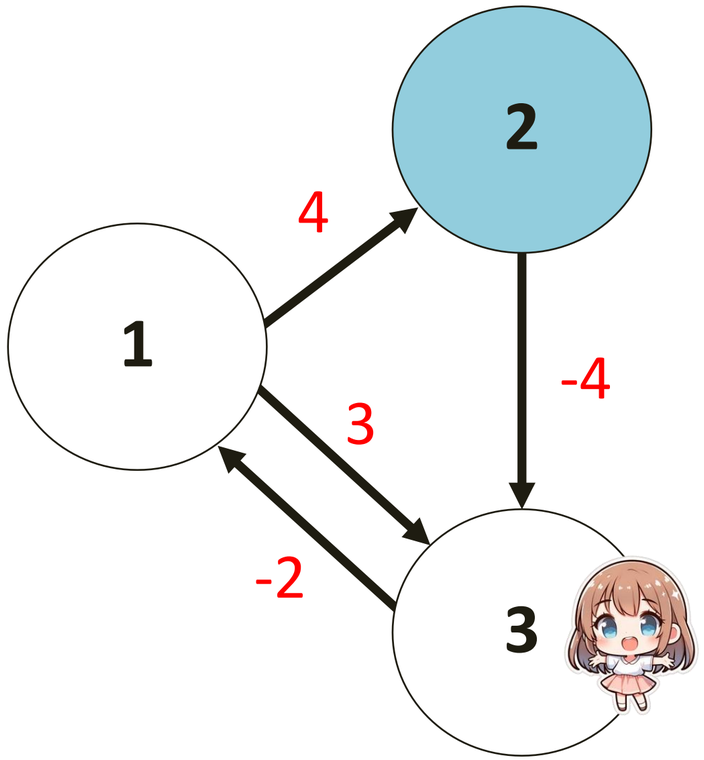
음수 사이클 검출하기 (v번째)



2번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	-4	0	-2

음수 사이클 검출하기 (v번째)



3번 정점은 2번 정점과 연결 되어있다.

간선의 가중치는 -4이다.

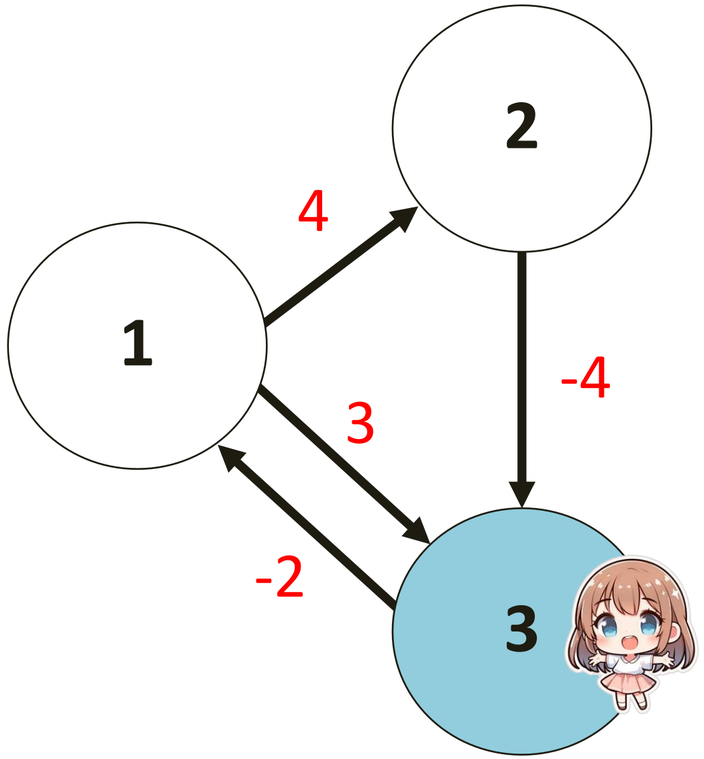
갱신 비용 = 0 - 4 = -4

이때, 기존 비용보다 갱신 비용이 작으므로
(-2 vs -4) 갱신한다.

더 이상 방문할 정점이 없으므로 2번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	-4	0	-4

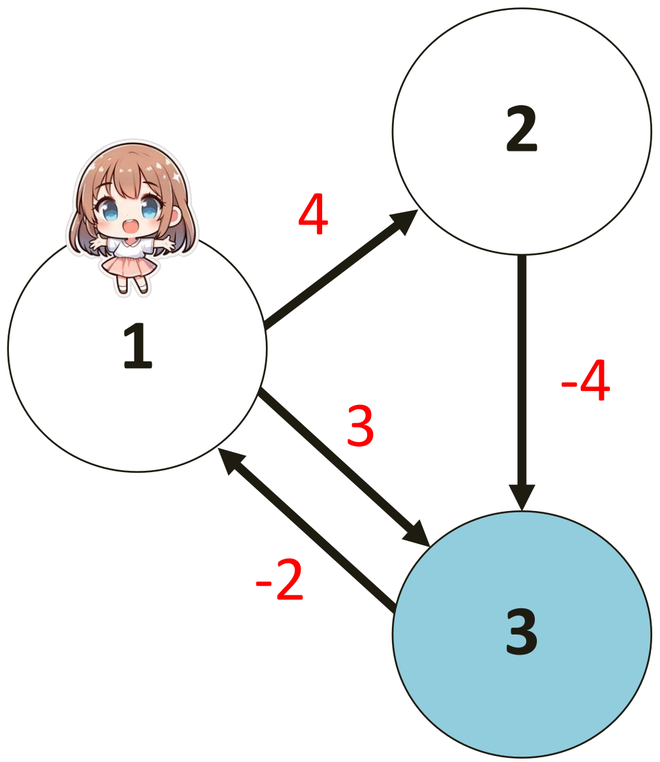
음수 사이클 검출하기 (v번째)



3번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	-4	0	-4

음수 사이클 검출하기 (v번째)



1번 정점은 3번 정점과 연결 되어있다.

간선의 가중치는 -2이다.

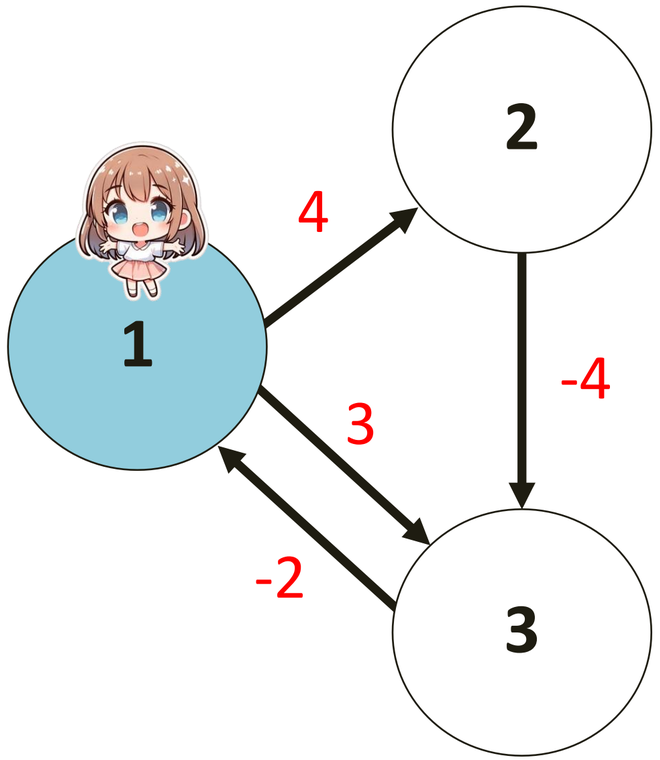
갱신 비용 = -4 - 2 = -6

이때, 기존 비용보다 갱신 비용이 작으므로
(-4 vs -6) 갱신한다.

더 이상 방문할 정점이 없으므로 3번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	-6	0	-4

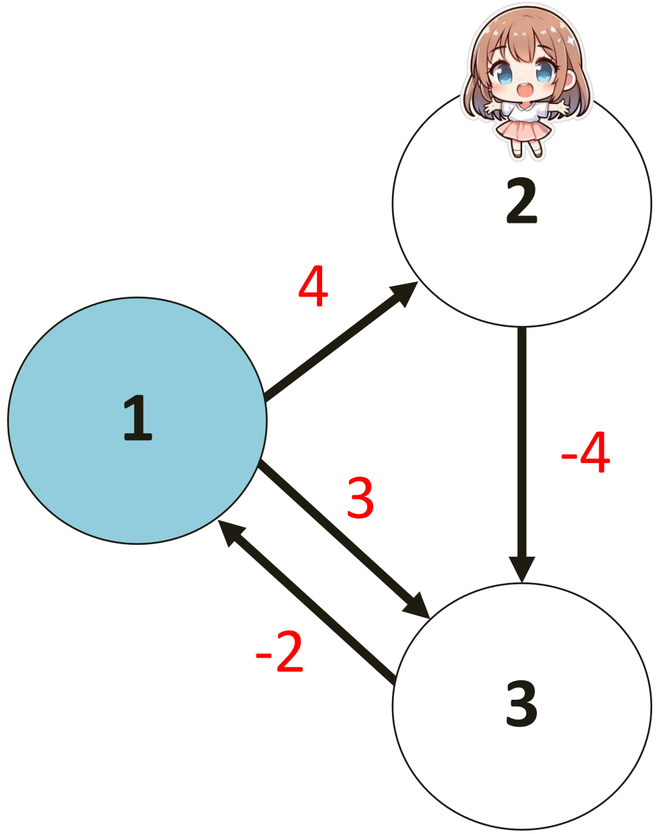
음수 사이클 검출하기 (V+1번째)



1번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	-6	0	-4

음수 사이클 검출하기 (V+1번째)



2번 정점은 1번 정점과 연결 되어있다.

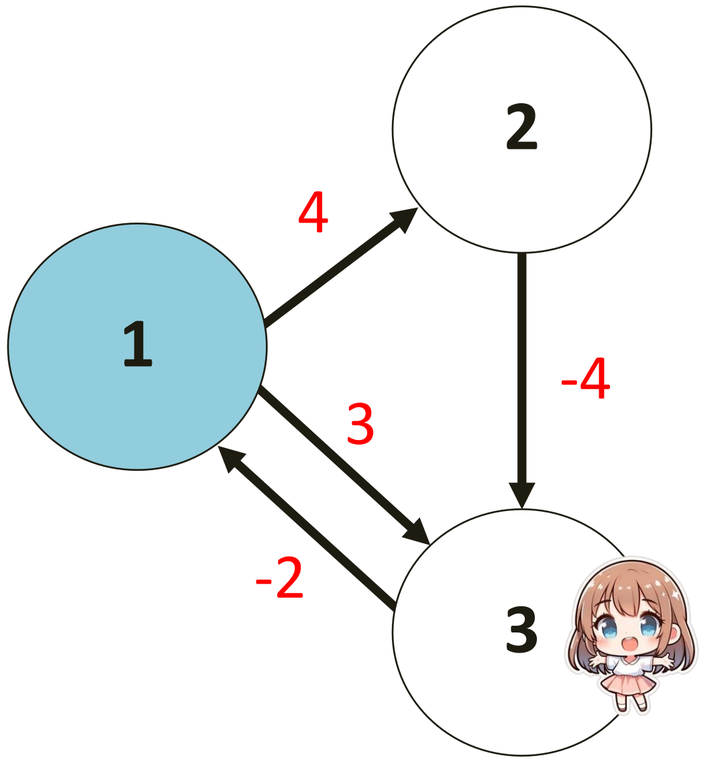
간선의 가중치는 4이다.

갱신 비용 = -6 + 4 = -2

이때, 기존 비용보다 갱신 비용이 작으므로
(0 vs -2) 갱신해준다.

정점 번호	1	2	3
최소 거리	-6	-2	-4

음수 사이클 검출하기 (V+1번째)



3번 정점은 1번 정점과 연결 되어있다.

간선의 가중치는 3이다.

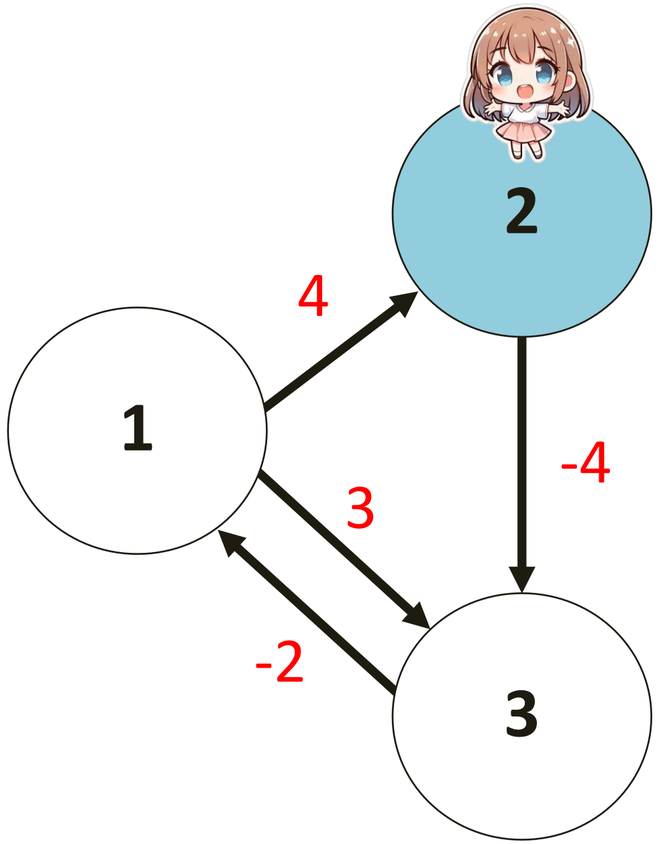
갱신 비용 = -6 + 3 = -3

이때, 기존 비용보다 갱신 비용이 크므로
(-4 vs -3) 갱신하지 않는다.

더 이상 방문할 정점이 없으므로 1번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	-6	-2	-4

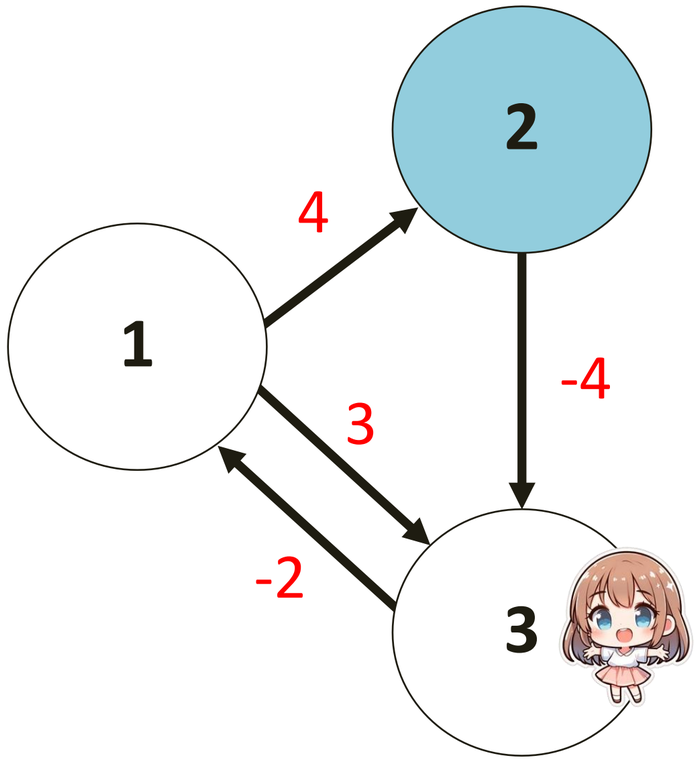
음수 사이클 검출하기 (V+1번째)



2번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	-6	-2	-4

음수 사이클 검출하기 (V+1번째)



3번 정점은 2번 정점과 연결 되어있다.

간선의 가중치는 -4이다.

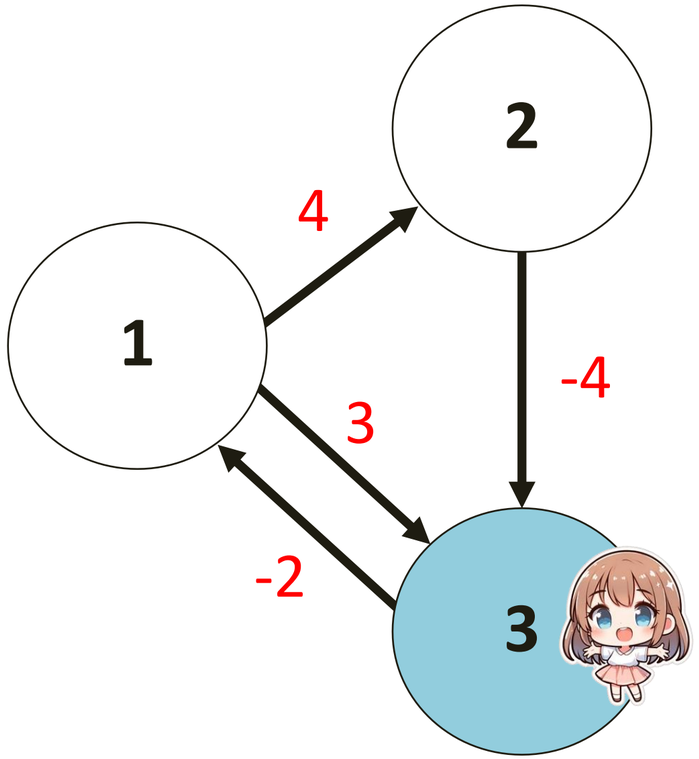
갱신 비용 = -2 - 4 = -6

이때, 기존 비용보다 갱신 비용이 작으므로
(-4 vs -6) 갱신한다.

더 이상 방문할 정점이 없으므로 2번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	-6	-2	-6

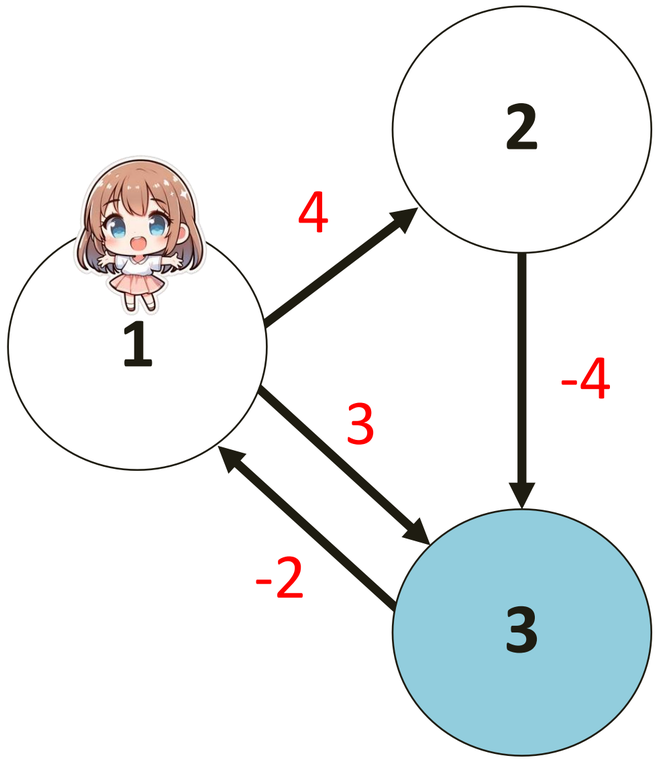
음수 사이클 검출하기 (V+1번째)



3번 정점의 갱신을 시작한다.

정점 번호	1	2	3
최소 거리	-6	-2	-6

음수 사이클 검출하기 (V+1번째)



1번 정점은 3번 정점과 연결 되어있다.

간선의 가중치는 -2이다.

갱신 비용 = -6 - 2 = -8

이때, 기존 비용보다 갱신 비용이 작으므로
(-6 vs -8) 갱신한다.

더 이상 방문할 정점이 없으므로 3번 정점에서 할 일을 끝낸다.

정점 번호	1	2	3
최소 거리	-8	-2	-6

음수 사이클 검출하기

k-1번째

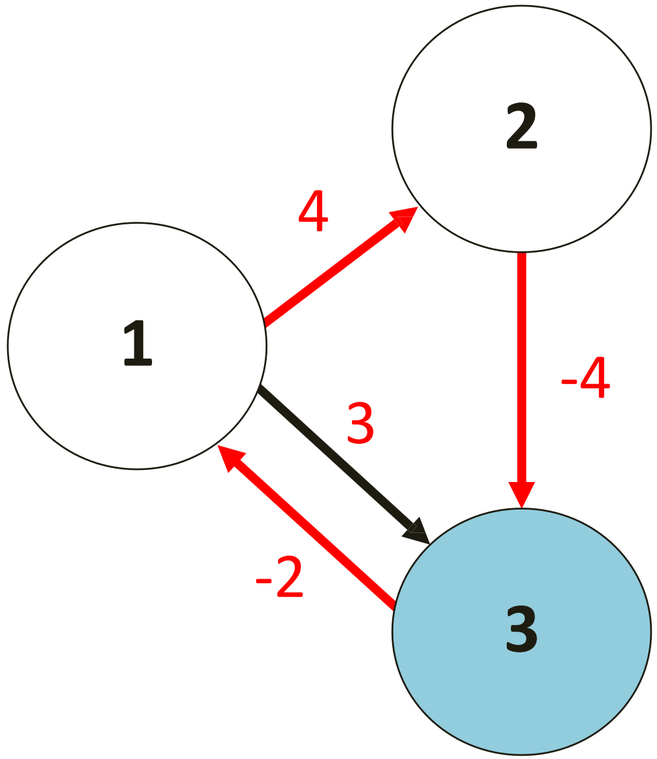
정점 번호	1	2	3
최소 거리	-4	2	-2

k번째

정점 번호	1	2	3
최소 거리	-6	0	-4

k+1번째

정점 번호	1	2	3
최소 거리	-8	-2	-6



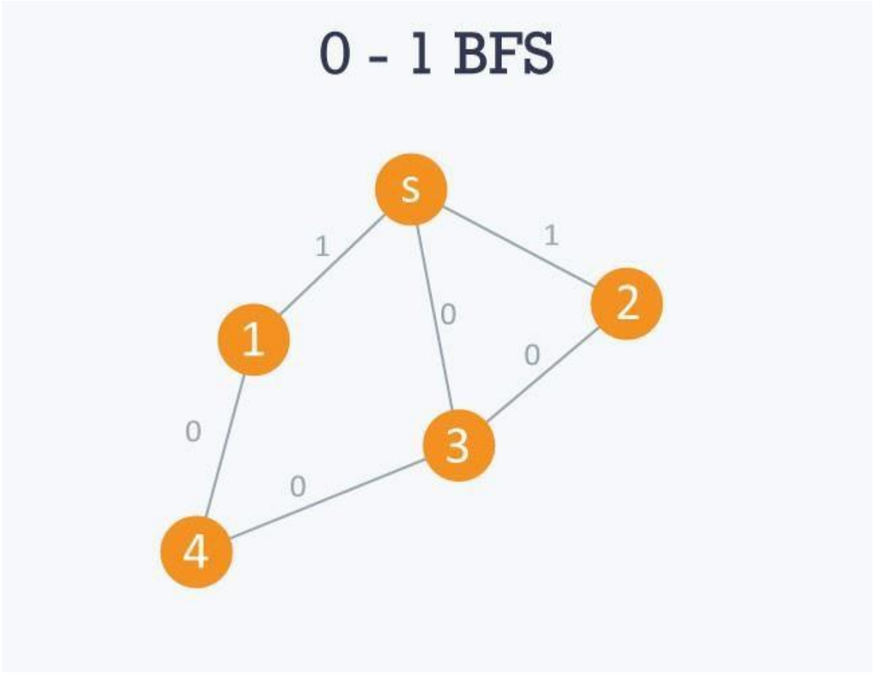
반복을 하면 할수록 최단거리가 줄어든다 == INF번 반복하면 -INF까지 간다.

고로 음수 사이클에서는 최소 거리 / 비용을 구할 수 없다.

이미 최단 경로를 구했다고 한들, 음수 사이클을 돌때마다 -2씩 최단 거리가 감소하므로 구할 수 없다.
(줄일려면 도는게 이득인데, N번째 돈것보다 N+1번째 돈게 더 작으니 계속 돌 수밖에 없게됨)

0 - 1 bfs

0-1 BFS(너비 우선 탐색)는 가중치가 0 또는 1인 간선으로 이루어진 그래프에서 최단 경로를 찾기 위해 사용되는 알고리즘이다.
일반적인 BFS와 유사하지만, 큐 혹은 우선순위 큐 대신 덱(deque)을 사용하여 최단 경로를 더욱 효율적으로 탐색한다. 가중치가 0인 간선은 덱의 앞쪽에, 가중치가 1인 간선은 덱의 뒤쪽에 추가하여, 비용이 적은 경로를 우선적으로 탐색한다. 이로 인해 시간 복잡도는 O(V + E)로 매우 효율적이며, 가중치가 0과 1로 제한된 그래프에서 다익스트라 알고리즘보다 빠르게 최단 경로를 구할 수 있다.



0 - 1 bfs

문제

알고스팟 운영진이 모두 미로에 갇혔다. 미로는 N*M 크기이며, 총 1*1크기의 방으로 이루어져 있다. 미로는 빈 방 또는 벽으로 이루어져 있고, 빈 방은 자유롭게 다닐 수 있지만, 벽은 부수지 않으면 이동할 수 없다.

알고스팟 운영진은 여러명이지만, 항상 모두 같은 방에 있어야 한다. 즉, 여러 명이 다른 방에 있을 수는 없다. 어떤 방에서 이동할 수 있는 방은 상하좌우로 인접한 빈 방이다. 즉, 현재 운영진이 (x, y)에 있을 때, 이동할 수 있는 방은 (x+1, y), (x, y+1), (x-1, y), (x, y-1) 이다. 단, 미로의 밖으로 이동 할 수는 없다.

벽은 평소에는 이동할 수 없지만, 알고스팟의 무기 AOJ를 이용해 벽을 부수어 버릴 수 있다. 벽을 부수면, 빈 방과 동일한 방으로 변한다.

만약 이 문제가 알고스팟에 있다면, 운영진들은 궁극의 무기 sudo를 이용해 벽을 한 번에 다 없애버릴 수 있지만, 안타깝게도 이 문제는 Baekjoon Online Judge에 수록되어 있기 때문에, sudo를 사용할 수 없다.

현재 (1, 1)에 있는 알고스팟 운영진이 (N, M)으로 이동하려면 벽을 최소 몇 개 부수어야 하는지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 미로의 크기를 나타내는 가로 크기 M, 세로 크기 N (1 ≤ N, M ≤ 100)이 주어진다. 다음 N개의 줄에는 미로의 상태를 나타내는 숫자 0과 1이 주어진다. 0은 빈 방을 의미하고, 1은 벽을 의미한다.

(1, 1)과 (N, M)은 항상 뚫려있다.

출력

첫째 줄에 알고스팟 운영진이 (N, M)으로 이동하기 위해 벽을 최소 몇 개 부수어야 하는지 출력한다.

N*M 크기의 미로가 있고 플레이어는 (1,1)에서 출발할 수 있다.

벽을 무한정으로 부술 수 있다고 가정할 때 (N,M)으로 도착 시 부순 벽의 최소 개수를 구해야 하는 문제이다.

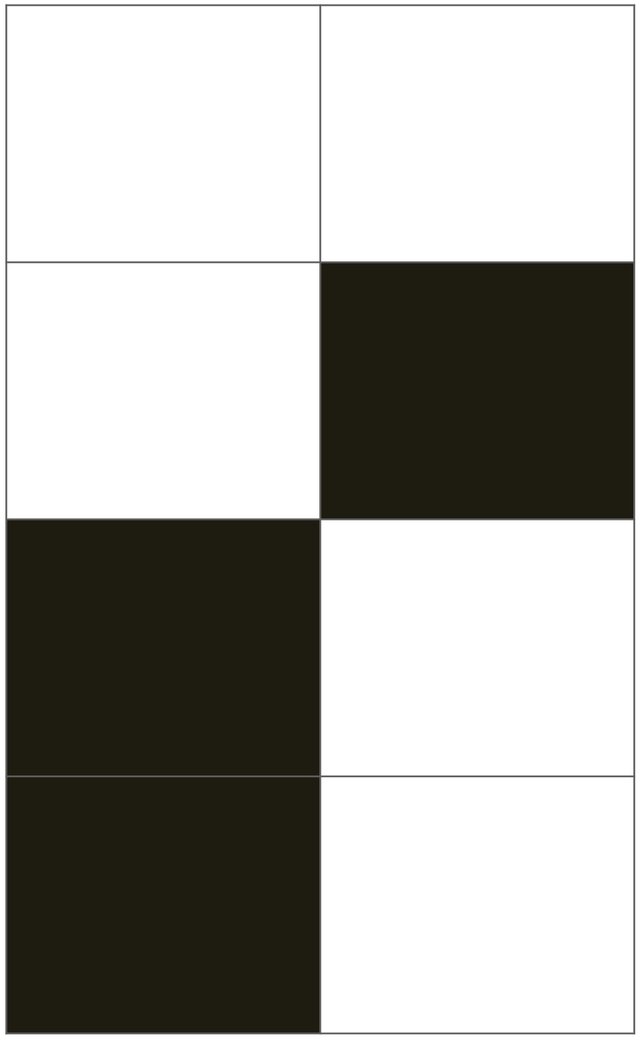
우리는 주로 이동의 가중치가 전부 같거나 없는 격자판에서 최단 경로를 구했다.
그러나 이 문제에서는 벽이 없는 곳으로 이동 할 때 가중치가 0, 벽이 있는 곳으로 이동할 때 가중치 1로 이동 비용이 다르다.

즉, 일반적인 bfs로는 최소비용을 구할 수 없다.

일반 bfs를 사용하면 벽을 부수는 비용을 고려하지 않고 단순히 경로의 길이만으로 탐색하게 되기 때문에 벽을 부수는 최소 경로를 탐색하지 못할 가능성이 높음.
+) 이 문제는 다익스트라 알고리즘으로도 풀 수 있다.

0-1 너비 우선 탐색 문제 풀이 예시

```
2 4
0 0
0 1
1 0
1 0
```



1 – 이미 선행 노드가 방문한 곳을 방문하는 것이 최적일수도 있다.
-> bool 타입이 아니라 int 타입으로 생성
(벽을 몇 개 부수고 이 위치에 왔는지 확인하고 갱신 해야 하기 때문)

2 – 덱을 이용한 방식
-> 일반적으로는 가중치가 낮은 노드가 front, 높은 노드가 back으로 들어감

3 – 일반 bfs와 동일하게 제일 먼저 목표지점에 도달한 노드가 곧 최단 경로

* 흰색은 빈방, 검은색은 벽*

0 – 1 bfs

INF	INF
INF	INF
INF	INF
INF	INF

Deque

아직 탐색을 시작하기 전이다.

모든 방문 배열에 대해 INF값으로 초기화 해준다.

*방문 처리를 해줄 때, A라는 노드가 먼저 임의의 칸에 도달 했다고 하자.

이 문제는 벽을 최소한으로 부수고 목표지점에 도달해야 하는 문제인데,

B라는 노드가 A보다 벽을 훨씬 덜 부순 채로 해당지점에 도착했다면 당연히 B가 우선이 되야 할 것이다.

고로 격자판에 있는 숫자보다 작다면 최적해의 여지가 있으므로 덱에 넣어주고 visited배열을 갱신 해준다.

(크거나 같다면 덱에 안 집어넣음 / 마치 visited[nx][ny] == true 같은 느낌)

0 – 1 bfs

0	INF
INF	INF
INF	INF
INF	INF

Deque

(1,1)
cnt : 0

(1,1) 에서 시작한다.

(1,1) 에는 벽이 없으므로 visited[1][1] = 0; 으로 처리해준다음
덱에 좌표와 벽의 개수를 집어넣어준다.

0 – 1 bfs

0 0 	INF
INF	INF
INF	INF
INF	INF

Deque

덱의 앞쪽에서 노드를 빼고 방문을 해준다.

좌표는 (1,1), 횟수는 0이였다.

상 하 좌 우 4 방향으로 탐색을 시작한다.

0 – 1 bfs

0	0 0 
INF	INF
INF	INF
INF	INF

Deque

(1,2)
cnt : 0

(1,2)에 도착했다.

벽이 존재하지 않는다.

갱신 횟수 : 0 + 0 = 0

이때, 기존 횟수 보다 갱신 횟수가 작으므로 (INF vs 0)
visited 배열을 갱신하고 덱 앞에 넣어준다.

0 – 1 bfs

0	0
0 0 	INF
INF	INF
INF	INF

Deque

(2,1) cnt : 0	(1,2) cnt : 0

(2,1)에 도착했다.

벽이 존재하지 않는다.

갱신 횟수 : 0 + 0 = 0

이때, 기존 횟수 보다 갱신 횟수가 작으므로 (INF vs 0)
visited 배열을 갱신하고 덱 앞에 넣어준다.

모든 방향을 이동했으므로 탐색을 종료한다.

0 – 1 bfs

0	0
0 0 	INF
INF	INF
INF	INF

Deque

(1,2)
cnt : 0

덱의 앞쪽에서 노드를 빼고 방문을 해준다.

좌표는 (2,1), 횟수는 0이였다.

상 하 좌 우 4 방향으로 탐색을 시작한다.

0 – 1 bfs

0  0	0
0	INF
INF	INF
INF	INF

Deque

(1,2)
cnt : 0

(1,1)에 도착했다.

벽이 존재하지 않는다.

갱신 횟수 : 0 + 0 = 0

이때, 기존 횟수와 갱신 횟수가 같으므로 (0 vs 0)
아무것도 하지 않는다.

0 – 1 bfs

0	0
0	INF
0 1 	INF
INF	INF

Deque

(1,2) cnt : 0	(3,1) cnt : 1

(3,1)에 도착했다.

벽이 존재한다.

갱신 횟수 : 0 + 1 = 1

이때, 기존 횟수 보다 갱신 횟수가 작으므로 (INF vs 1)
visited 배열을 갱신하고 덱 뒤에 넣어준다.

0 – 1 bfs

0	0
0	0 1 
1	INF
INF	INF

Deque

(1,2) cnt : 0	(3,1) cnt : 1	(2,2) cnt : 1

(2,2)에 도착했다.

벽이 존재한다.

갱신 횟수 : 0 + 1 = 1

이때, 기존 횟수 보다 갱신 횟수가 작으므로 (INF vs 1)
visited 배열을 갱신하고 덱 뒤에 넣어준다.

모든 방향을 이동했으므로 탐색을 종료한다.

0 – 1 bfs

0	0 0 
0	1
1	INF
INF	INF

Deque

(3,1) cnt : 1	(2,2) cnt : 1

덱의 앞쪽에서 노드를 빼고 방문을 해준다.

좌표는 (1,2), 횟수는 0이였다.

상 하 좌 우 4 방향으로 탐색을 시작한다.

0 – 1 bfs

0  0	0
0	1
1	INF
INF	INF

Deque

(3,1) cnt : 1	(2,2) cnt : 1

(1,1)에 도착했다.

벽이 존재하지 않는다.

갱신 횟수 : 0 + 0 = 0

이때, 기존 횟수와 갱신 횟수가 같으므로 (0 vs 0)
아무것도 하지 않는다.

0 – 1 bfs

0	0
0	0 1 
1	INF
INF	INF

Deque

(3,1) cnt : 1	(2,2) cnt : 1

(2,2)에 도착했다.

벽이 존재한다.

갱신 횟수 : 0 + 1 = 1

이때, 기존 횟수와 갱신 횟수가 같으므로 (1 vs 1)
아무것도 하지 않는다.

모든 방향을 이동했으므로 탐색을 종료한다.

0 – 1 bfs

0	0
0	1
1  1	INF
INF	INF

Deque

(2,2)
cnt : 1

덱의 앞쪽에서 노드를 빼고 방문을 해준다.

좌표는 (3,1), 횟수는 1이였다.

상 하 좌 우 4 방향으로 탐색을 시작한다.

0 – 1 bfs

0	0
1 0 	1
1	INF
INF	INF

Deque

(2,2)
cnt : 1

(2,1)에 도착했다.

벽이 존재하지 않는다.

갱신 횟수 : 1 + 0 = 1

이때, 기존 횟수 보다 갱신 횟수가 크므로 (0 vs 1)
아무것도 하지 않는다.

0 – 1 bfs

0	0
0	1
1	INF
1 2 	INF

Deque

(2,2) cnt : 1	(4,1) cnt : 2	

(4,1)에 도착했다.

벽이 존재 한다.

갱신 횟수 : 1 + 1 = 2

이때, 기존 횟수 보다 갱신 횟수가 작으므로 (INF vs 2)
visited 배열을 갱신하고 덱 뒤에 넣어준다.

0 – 1 bfs

0	0
0	1
1	1 
2	INF

Deque

(3,2) cnt : 1	(2,2) cnt : 1	(4,1) cnt : 2	

(3,2)에 도착했다.

벽이 존재하지 않는다.

갱신 횟수 : 1 + 0 = 1

이때, 기존 횟수 보다 갱신 횟수가 작으므로 (INF vs 1)
visited 배열을 갱신하고 덱 앞에 넣어준다.

모든 방향을 이동했으므로 탐색을 종료한다

0 – 1 bfs

0	0
0	1
1	1 
2	INF

Deque

(2,2) cnt : 1	(4,1) cnt : 2	

덱의 앞쪽에서 노드를 빼고 방문을 해준다.

좌표는 (3,2), 횟수는 1이였다.

상 하 좌 우 4 방향으로 탐색을 시작한다.

0 – 1 bfs

0	0
0	1 
1	1
2	INF

Deque

(2,2) cnt : 1	(4,1) cnt : 2

(2,2)에 도착했다.

벽이 존재한다.

갱신 횟수 : 1 + 1 = 2

이때, 기존 횟수 보다 갱신 횟수가 크므로 (1 vs 2)
아무것도 하지 않는다.

0 – 1 bfs

0	0
0	1
1	1
2	1 1 

Deque

(2,2) cnt : 1	(4,1) cnt : 2

(2,4)에 도착했다.

벽이 존재하지 않는다.

갱신 횟수 : 1 + 0 = 1

이때, 기존 횟수 보다 갱신 횟수가 작다. (INF vs 1)
또한, 목표 좌표인 (2,4)에 도착했다.

결론적으로 (1,1)에서 (2,4)까지 이동할 때 최소 1개의 벽을 부수고 도착할 수 있다.