JavaScript 최단 경로
벨만 포드 알고리즘 이해하기

벨만 포드 알고리즘 이해하기 | 최단 경로를 찾아주는 벨만 포드 알고리즘 이해하기

강사 나동빈

JavaScript
최단 경로
벨만 포드 알고리즘 이해하기

최단 경로 문제란?

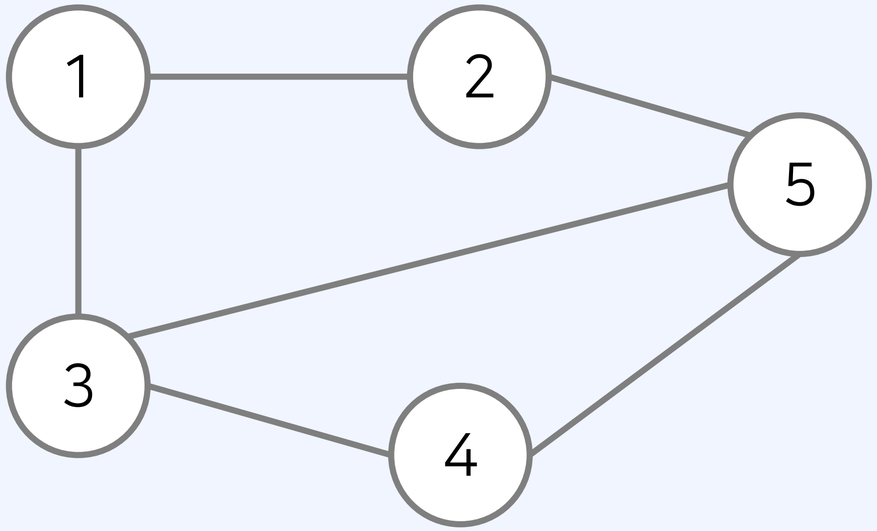
JavaScript
최단 경로
벨만 포드
이해하기

- 최단 경로 알고리즘은 **가장 짧은 경로를 찾는 알고리즘**을 의미한다.
- **다양한 문제 상황**
  - 한 지점에서 다른 한 지점까지의 최단 경로
  - 한 지점에서 다른 모든 지점까지의 최단 경로 → 다익스트라 알고리즘
  - 모든 지점에서 다른 모든 지점까지의 최단 경로 → 플로이드 워셜 알고리즘
- 각 지점은 그래프에서 **노드**로 표현
- 지점 간 연결된 도로는 그래프에서 **간선**으로 표현

그래프의 표현

JavaScript
최단 경로
벨만 포드
이해하기

- 일반적으로 JavaScript로 그래프 관련 문제를 해결할 때는?
- 2차원 배열(리스트)로 그래프를 표현한다.



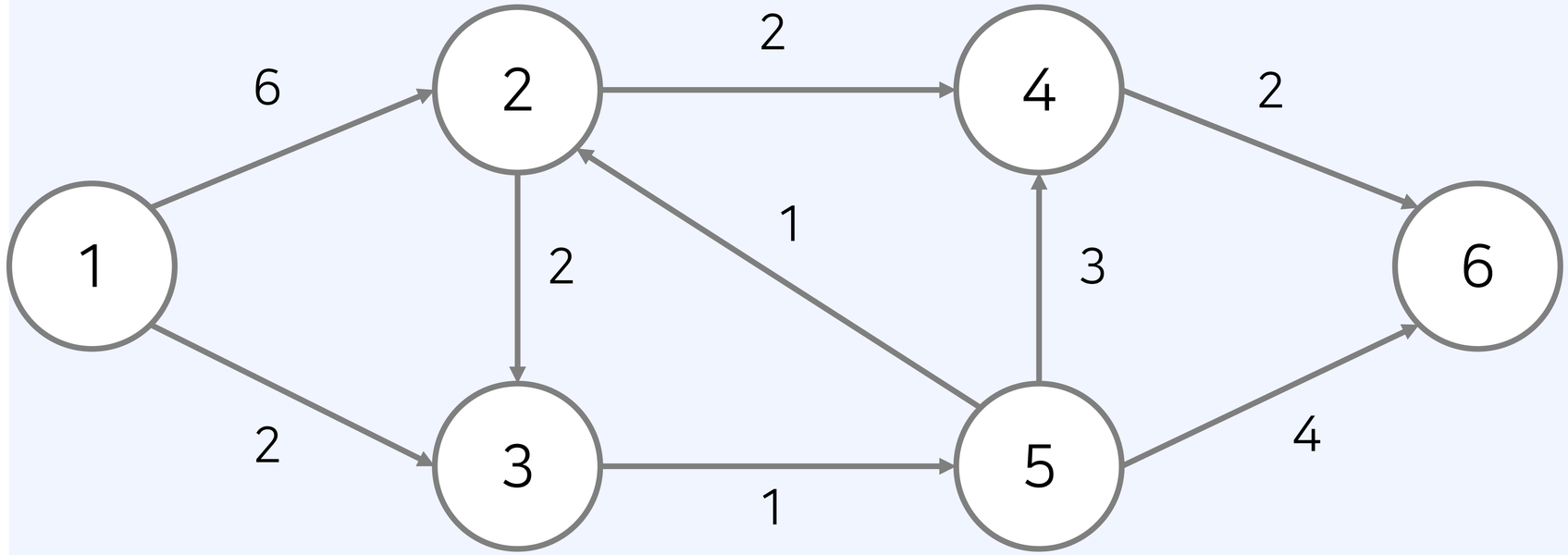
인접 리스트

1	2, 3
2	1, 5
3	1, 4, 5
4	3, 5
5	2, 3, 4

음수 간선이 포함된 상황에서의 최단 거리 문제

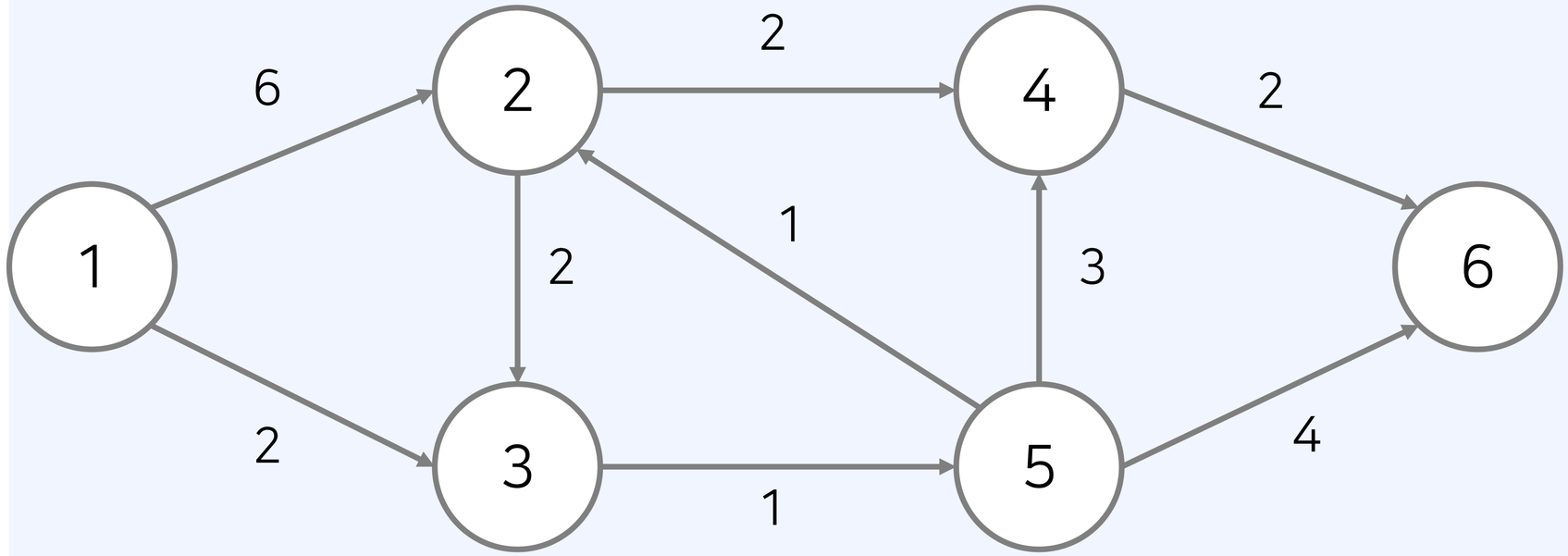
JavaScript
최단 경로
벨만 포드
이해하기

- 모든 간선의 비용이 **양수**일 때는 다익스트라 최단 경로 알고리즘을 사용하면 된다.
  - **1번 노드에서 다른 노드로 가기 위한 최소 비용**은 얼마일까?



음수 간선이 포함된 상황에서의 최단 거리 문제

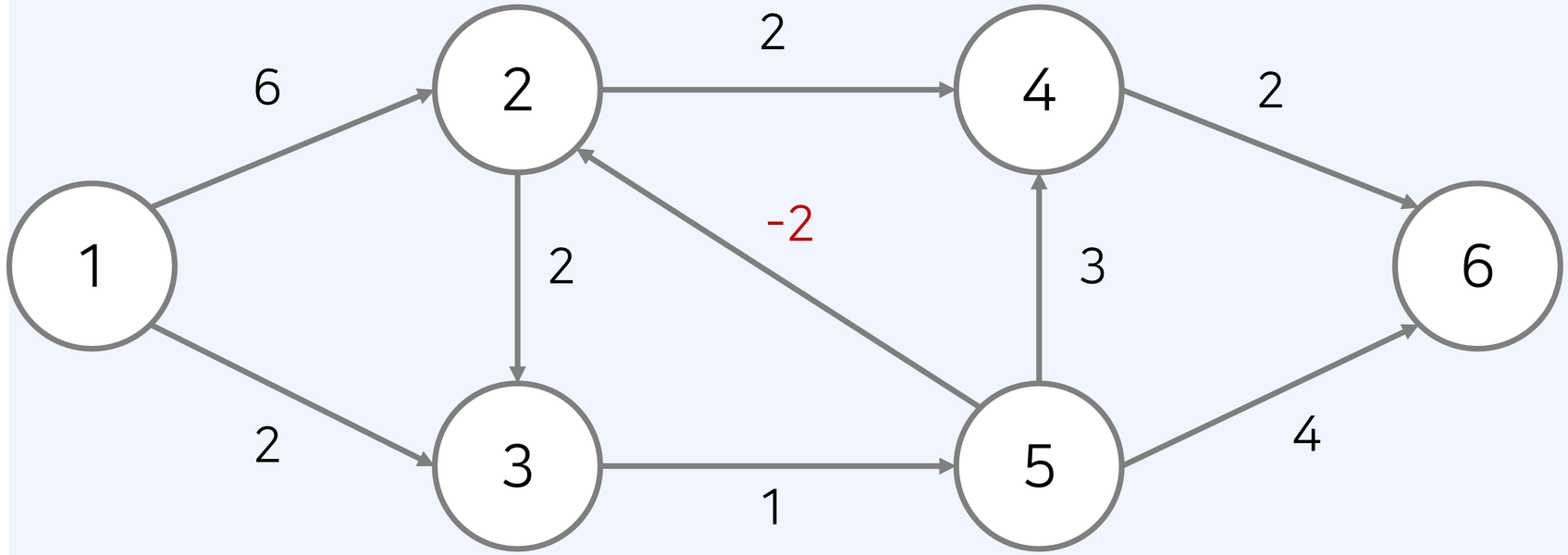
- 모든 간선의 비용이 **양수**일 때는 다익스트라 최단 경로 알고리즘을 사용하면 된다.
  - **1번 노드에서 다른 노드로 가기 위한 최소 비용**은 얼마일까?



도착 노드	최소 비용
1번	0
2번	4
3번	2
4번	6
5번	3
6번	7

음수 간선이 포함된 상황에서의 최단 거리 문제

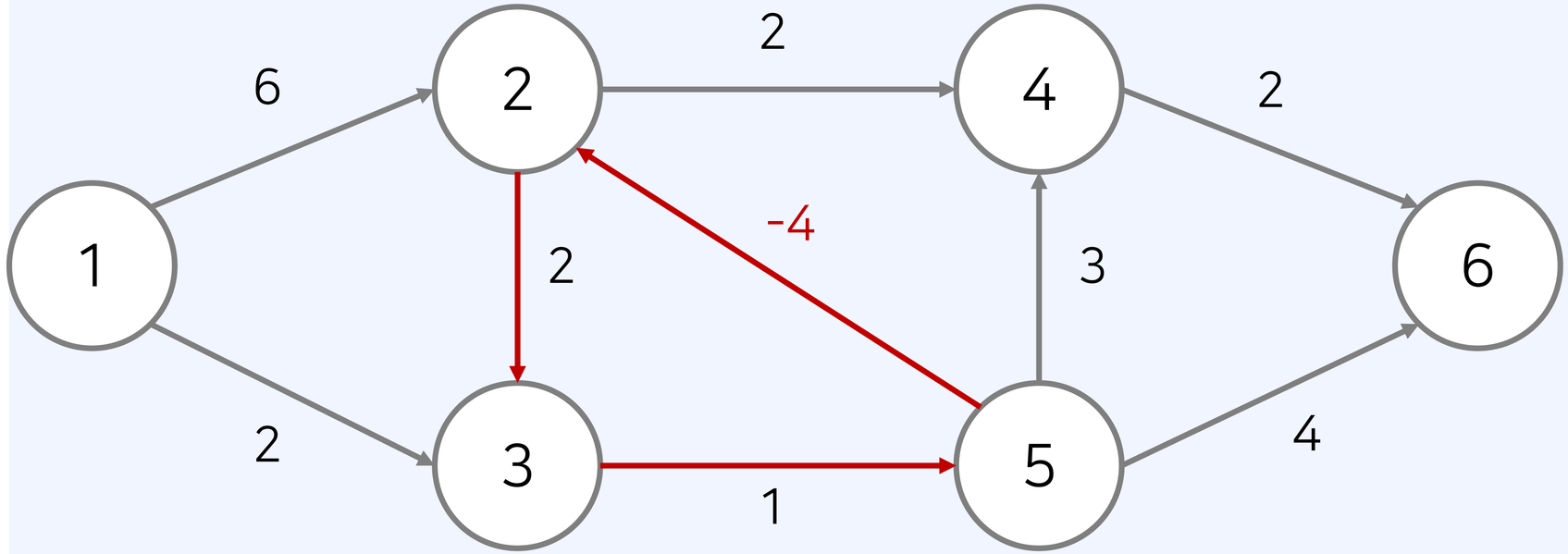
- 하지만 **음수 간선**이 포함된다면 어떻게 문제를 해결할 수 있을까?
  - 아래 그래프에서는 음수 간선이 포함되어 있다.
  - 하지만 여전히 최단 거리를 계산할 수 있다.



도착 노드	최소 비용
1번	0
2번	1
3번	2
4번	3
5번	3
6번	5

음수 간선이 포함된 상황에서의 최단 거리 문제

- 하지만 **음수 순환**이 포함된다면 어떻게 문제를 해결할 수 있을까?
  - 아래 그래프에서는 음수 순환이 포함되어 있다.
  - 이 경우 최단 거리가 음의 무한인 노드가 발생한다.



도착 노드	최소 비용
1번	0
2번	−∞
3번	−∞
4번	−∞
5번	−∞
6번	−∞

벨만 포드 최단 경로 알고리즘

- 음수 간선에 관하여 최단 경로 문제는 다음과 같이 분류할 수 있다.
  1) 모든 간선이 양수인 경우
  2) 음수 간선이 있는 경우
     1) 음수 순환은 없는 경우
     2) 음수 순환이 있는 경우
- **벨만 포드 최단 경로 알고리즘**은 음의 간선이 포함된 상황에서도 사용할 수 있다.
  - 또한 음수 순환을 감지할 수 있다.
  - 벨만 포드의 기본 시간 복잡도는 $O(VE)$로 다익스트라 알고리즘에 비해 느리다.

→ 기본적으로 동작 과정에서 다익스트라 알고리즘의 **"최적의 해"**를 항상 포함한다.

벨만 포드 최단 경로 알고리즘

- **벨만 포드 알고리즘**은 다음과 같다.
  1. 출발 노드를 설정한다.
  2. 최단 거리 테이블을 초기화한다.
  3. 다음의 과정을 $N-1$번 반복한다.
     1. 전체 간선 E개를 하나씩 확인한다.
     2. 각 간선을 거쳐 다른 노드로 가는 비용을 계산하여 최단 거리 테이블을 갱신한다.
- 만약 음수 순환이 발생하는지 체크하고 싶다면 **3번의 과정을 한 번 더 수행**한다.
  - 이때 최단 거리 테이블이 갱신된다면 음수 순환이 존재하는 것이다.

벨만 포드 알고리즘 VS 다익스트라 알고리즘

JavaScript
최단 경로
벨만 포드
이해하기

- 다익스트라 알고리즘
  - 매번 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드를 선택한다.
  - 음수 간선이 없다면 최적의 해를 찾을 수 있다.
- 벨만 포드 알고리즘
  - 매번 모든 간선을 전부 확인한다.
    - 따라서 **다익스트라 알고리즘에서의 최적의 해를 항상 포함**한다.
  - 다익스트라 알고리즘에 비해서 시간이 오래 걸리지만 음수 순환을 탐지할 수 있다.

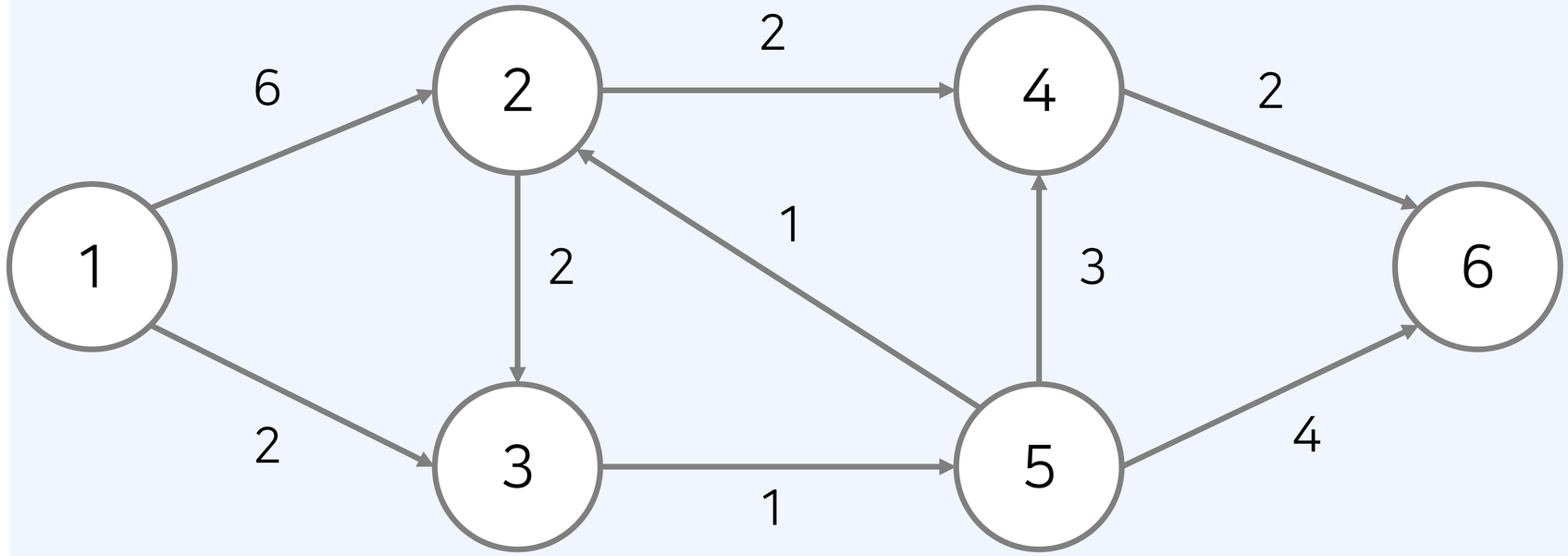
벨만 포드 알고리즘 소스코드

JavaScript 최단 경로
벨만 포드 이해하기

```
function bf(start) {
  // 시작 노드에 대해서 초기화
  dist[start] = 0;
  // 전체 n번의 라운드(round)를 반복
  for (let i = 0; i < n; i++) {
    // 매 반복마다 "모든 간선"을 확인하며
    for (let j = 0; j < m; j++) {
      let [cur, nextNode, cost] = edges[j];
      // 현재 간선을 거쳐서 다른 노드로 이동하는 거리가 더 짧은 경우
      if (dist[cur] != INF && dist[nextNode] > dist[cur] + cost) {
        dist[nextNode] = dist[cur] + cost;
        // n번째 라운드에서도 값이 갱신된다면 음수 순환이 존재
        if (i == n - 1) return true;
      }
    }
  }
  return false;
}
```

벨만 포드 알고리즘: 모든 간선이 양수인 경우

- **모든 간선이 양수**인 경우의 예시를 확인해 보자.



도착 노드	최소 비용
1번	0
2번	4
3번	2
4번	6
5번	3
6번	7

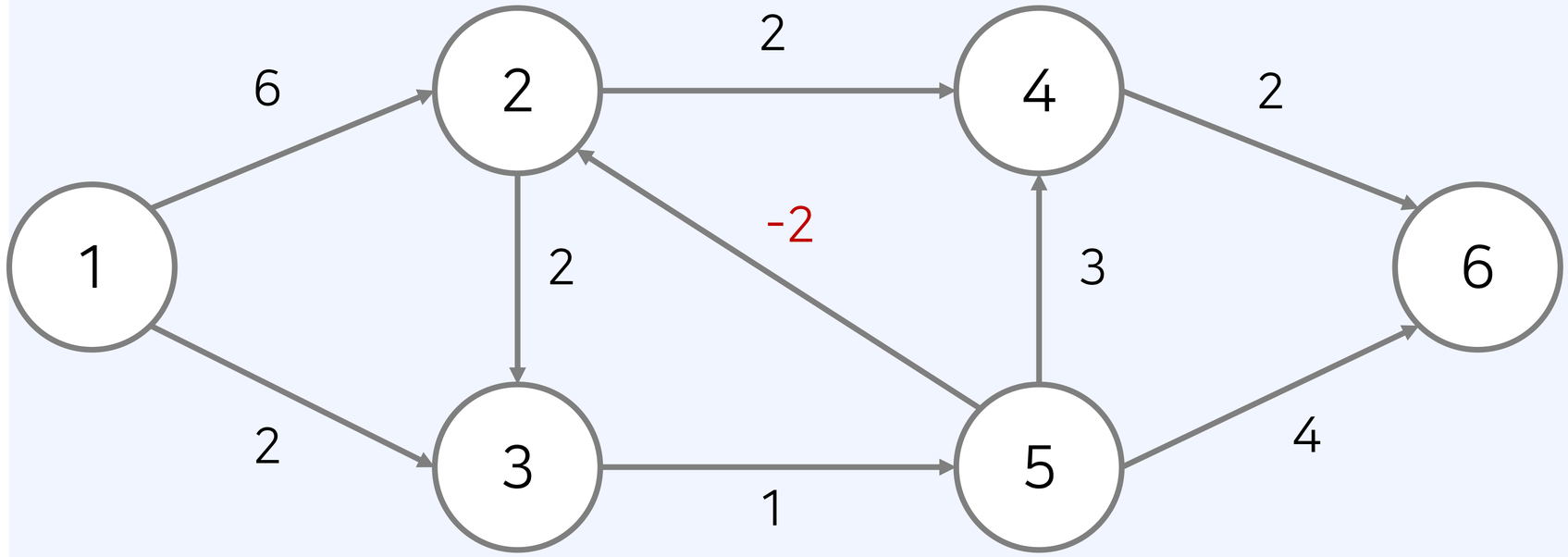
벨만 포드 알고리즘: 모든 간선이 양수인 경우

JavaScript
최단 경로
벨만 포드
이해하기

```
let INF = 1e9; // 무한을 의미하는 값으로 10억을 설정
let n = 6; // 노드의 개수
let m = 9; // 간선의 개수
// 모든 간선에 대한 정보를 담는 리스트 만들기
let edges = [ // [a, b, c]: a에서 b로 가는 비용이 c라는 의미
  [1, 2, 6],
  [1, 3, 2],
  [2, 3, 2],
  [2, 4, 2],
  [3, 5, 1],
  [4, 6, 2],
  [5, 2, 1],
  [5, 4, 3],
  [5, 6, 4]
];
let dist = new Array(n + 1).fill(INF); // 최단 거리를 모두 무한으로 초기화
// 벨만 포드 알고리즘 수행
let negativeCycle = bf(1); // 1번 노드가 시작 노드
if (negativeCycle) console.log(-1);
else {
  for (let i = 2; i <= n; i++) { // 1번 노드를 제외한 다른 노드들로 가기 위한 최단 거리 출력
    // 도달할 수 없는 경우 -1을 출력
    if (dist[i] == INF) console.log(-1);
    // 도달할 수 있는 경우 거리를 출력
    else console.log(dist[i]);
  }
}
```

벨만 포드 알고리즘: 음수 간선이 포함된 경우

- **음수 간선**이 포함된 예시를 확인해 보자.



도착 노드	최소 비용
1번	0
2번	1
3번	2
4번	3
5번	3
6번	5

벨만 포드 알고리즘: 음수 간선이 포함된 경우

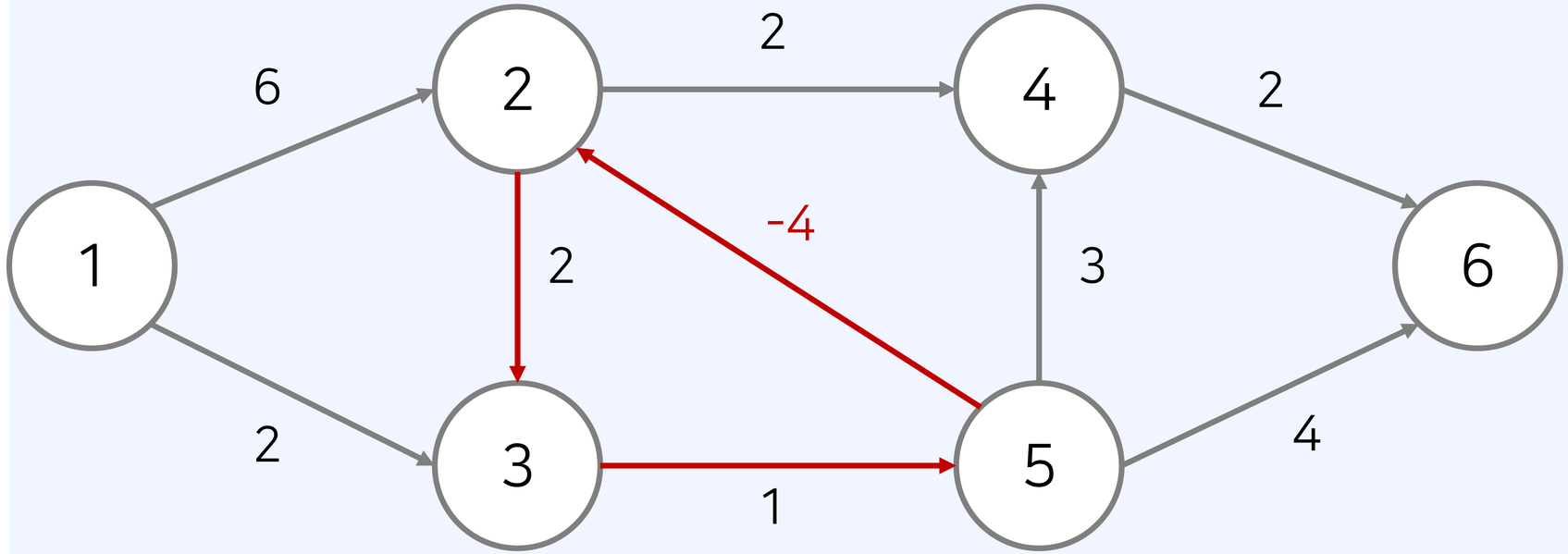
JavaScript
최단 경로
벨만 포드
이해하기

```
let INF = 1e9; // 무한을 의미하는 값으로 10억을 설정
let n = 6; // 노드의 개수
let m = 9; // 간선의 개수
// 모든 간선에 대한 정보를 담는 리스트 만들기
let edges = [ // [a, b, c]: a에서 b로 가는 비용이 c라는 의미
  [1, 2, 6],
  [1, 3, 2],
  [2, 3, 2],
  [2, 4, 2],
  [3, 5, 1],
  [4, 6, 2],
  [5, 2, -2],
  [5, 4, 3],
  [5, 6, 4]
];
let dist = new Array(n + 1).fill(INF); // 최단 거리를 모두 무한으로 초기화
// 벨만 포드 알고리즘 수행
let negativeCycle = bf(1); // 1번 노드가 시작 노드
if (negativeCycle) console.log(-1);
else {
  for (let i = 2; i <= n; i++) { // 1번 노드를 제외한 다른 노드들로 가기 위한 최단 거리 출력
    // 도달할 수 없는 경우 -1을 출력
    if (dist[i] == INF) console.log(-1);
    // 도달할 수 있는 경우 거리를 출력
    else console.log(dist[i]);
  }
}
```

벨만 포드 알고리즘: 음수 순환이 포함된 경우

JavaScript
최단 경로
벨만 포드
이해하기

- **음수 순환**이 포함된 예시를 확인해 보자.



도착 노드	최소 비용
1번	0
2번	−∞
3번	−∞
4번	−∞
5번	−∞
6번	−∞

벨만 포드 알고리즘: 음수 순환이 포함된 경우

```
let INF = 1e9; // 무한을 의미하는 값으로 10억을 설정
let n = 6; // 노드의 개수
let m = 9; // 간선의 개수
// 모든 간선에 대한 정보를 담는 리스트 만들기
let edges = [ // [a, b, c]: a에서 b로 가는 비용이 c라는 의미
  [1, 2, 6],
  [1, 3, 2],
  [2, 3, 2],
  [2, 4, 2],
  [3, 5, 1],
  [4, 6, 2],
  [5, 2, -4],
  [5, 4, 3],
  [5, 6, 4]
];
let dist = new Array(n + 1).fill(INF); // 최단 거리를 모두 무한으로 초기화
// 벨만 포드 알고리즘 수행
let negativeCycle = bf(1); // 1번 노드가 시작 노드
if (negativeCycle) console.log(-1);
else {
  for (let i = 2; i <= n; i++) { // 1번 노드를 제외한 다른 노드들로 가기 위한 최단 거리 출력
    // 도달할 수 없는 경우 -1을 출력
    if (dist[i] == INF) console.log(-1);
    // 도달할 수 있는 경우 거리를 출력
    else console.log(dist[i]);
  }
}
```