JavaScript 최단 경로
플로이드 워셜 알고리즘 이해하기

플로이드 워셜 알고리즘 이해하기 | 최단 경로를 찾아주는 플로이드 워셜 알고리즘 이해하기

강사 나동빈

JavaScript
최단 경로

플로이드 워셜 알고리즘 이해하기

최단 경로 문제란?

JavaScript
최단 경로
플로이드 워셜
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

플로이드 워셜 알고리즘 개요

- 모든 노드에서 다른 모든 노드까지의 최단 경로를 모두 계산한다.
- **플로이드 워셜(Floyd-Warshall)** 알고리즘은 다익스트라 알고리즘과 마찬가지로 단계별로 **거쳐 가는 노드를 기준으로 알고리즘을 수행**한다.

→ 매 단계마다 방문하지 않은 노드 중에 최단 거리를 갖는 노드를 찾는 과정이 필요하지 않는다.

- 플로이드 워셜은 2차원 테이블에 최단 거리 정보를 저장한다.
- 플로이드 워셜 알고리즘은 다이나믹 프로그래밍 유형에 속한다.

플로이드 워셜 알고리즘

JavaScript
최단 경로
플로이드 워셜
이해하기

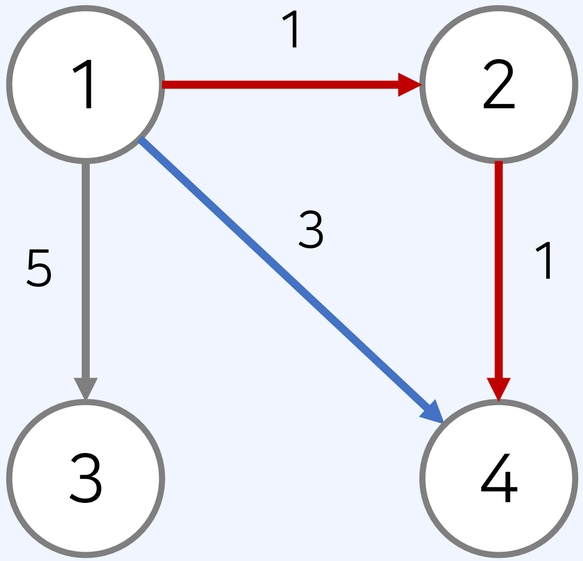
- 각 단계마다 **특정한 노드 K를 거쳐 가는 경우를 확인**한다.
  - A에서 B로 가는 최단 거리보다 A에서 K를 거쳐 B로 가는 거리가 더 짧은지 검사한다.
- 점화식은 다음과 같다.

$$D[A][B] = MIN(D[A][B], D[A][K] + D[K][B])$$

최단 경로 알고리즘의 특징

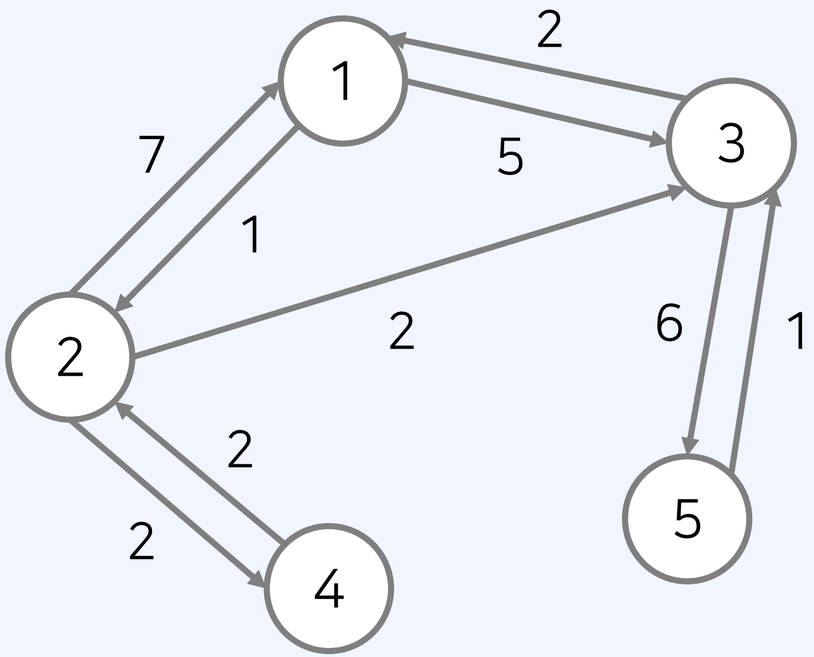
JavaScript
최단 경로
플로이드 워셜
이해하기

- 대부분의 최단 경로 알고리즘은 **다른 노드를 거쳐갈 때** 비용이 감소하는지 확인한다.
- 깊이 1: ① → ④로 가기 위하여 3원이 필요하다.
- **깊이 2**: ① → ② → ④로 가면 **2원**이 필요하다. (2번 노드를 거치며 확인 가능)



플로이드 워셜 알고리즘: 동작 과정 살펴보기

- 각 단계마다 A에서 **특정한 노드 K를 거쳐 B로 갈 때 더 짧아지는 경우를 확인**한다.
  - **기본 점화식**: $D[A][B] = MIN(D[A][B], D[A][K] + D[K][B])$

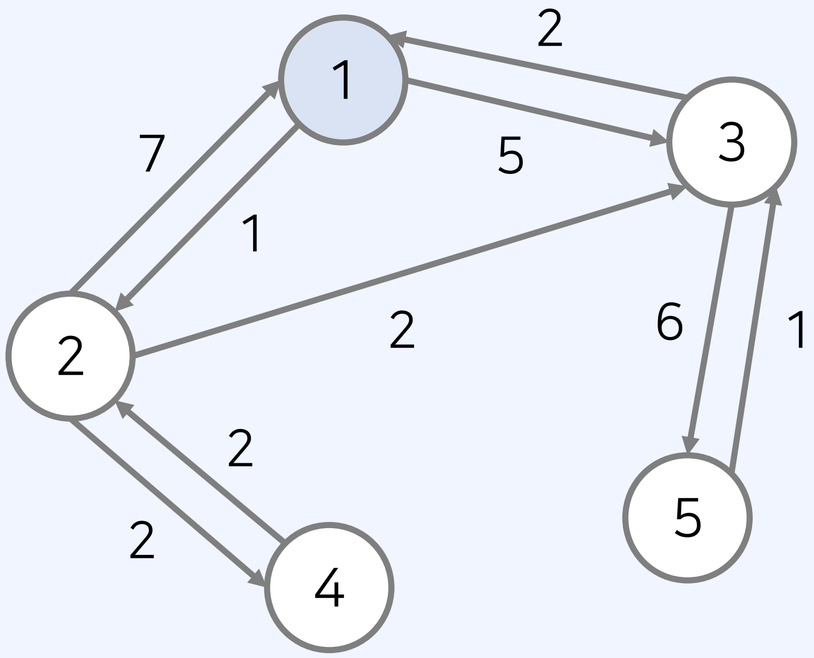


	1	2	3	4	5
1	0	1	5	INF	INF
2	7	0	2	2	INF
3	2	INF	0	INF	6
4	INF	2	INF	0	INF
5	INF	INF	1	INF	0

플로이드 워셜 알고리즘: 동작 과정 살펴보기

JavaScript
최단 경로
플로이드 워셜
이해하기

- 각 단계마다 A에서 **특정한 노드 K를 거쳐 B로 갈 때 더 짧아지는 경우를 확인**한다.
- **1단계**: $D[A][B] = MIN(D[A][B], D[A][1] + D[1][B])$



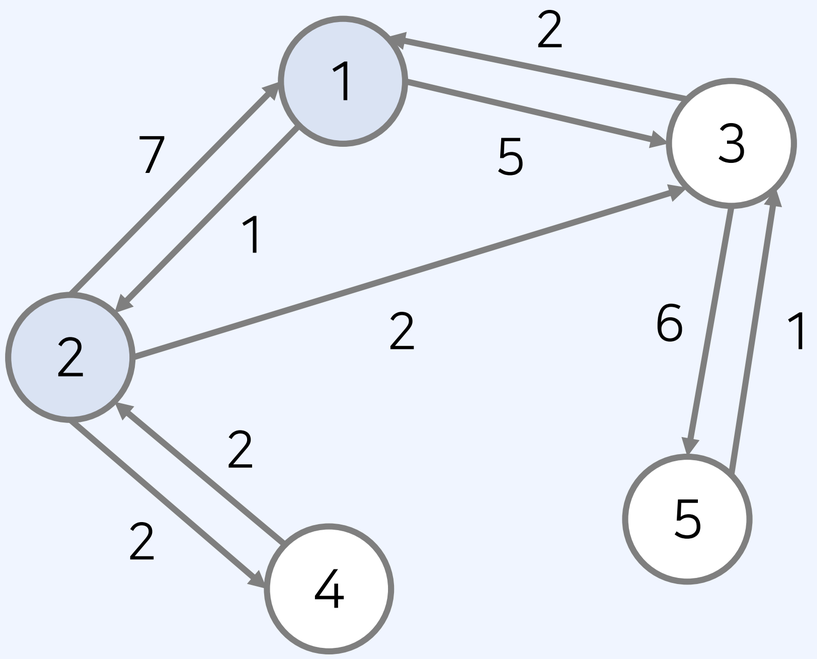
※ 파란색 선(거쳐가는 노드)상의 가까운 두 값을 더한다.

	1	2	3	4	5
1	0	1	5	INF	INF
2	7	0	2	2	INF
3	2	3	0	INF	6
4	INF	2	INF	0	INF
5	INF	INF	1	INF	0

플로이드 워셜 알고리즘: 동작 과정 살펴보기

JavaScript
최단 경로
플로이드 워셜
이해하기

- 각 단계마다 A에서 **특정한 노드 K를 거쳐 B로 갈 때 더 짧아지는 경우를 확인**한다.
- **2단계**: $D[A][B] = MIN(D[A][B], D[A][2] + D[2][B])$



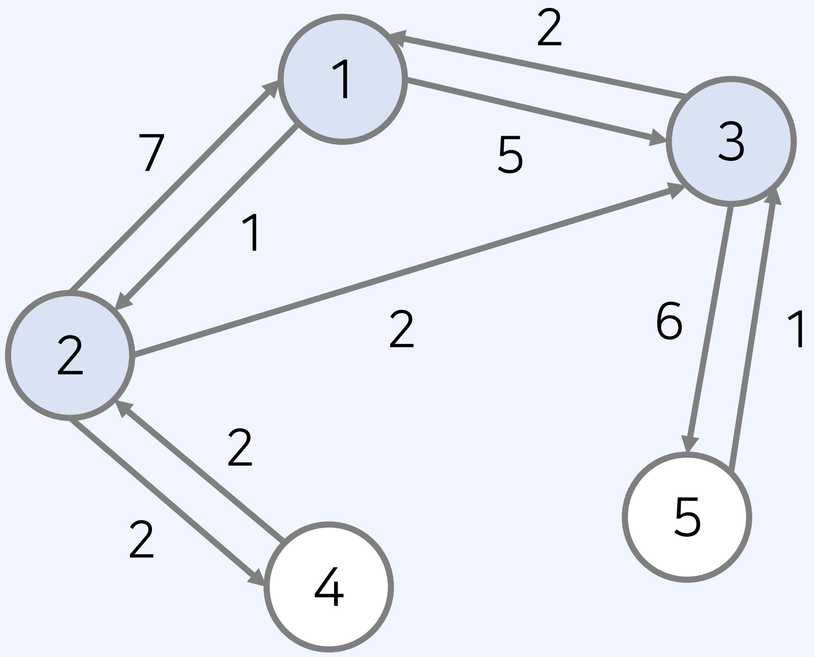
※ 파란색 선(거쳐가는 노드)상의 가까운 두 값을 더한다.

	1	2	3	4	5
1	0	1	3	3	INF
2	7	0	2	2	INF
3	2	3	0	5	6
4	9	2	4	0	INF
5	INF	INF	1	INF	0

플로이드 워셜 알고리즘: 동작 과정 살펴보기

JavaScript
최단 경로
플로이드 워셜
이해하기

- 각 단계마다 A에서 **특정한 노드 K를 거쳐 B로 갈 때 더 짧아지는 경우를 확인**한다.
- **3단계**: $D[A][B] = MIN(D[A][B], D[A][3] + D[3][B])$



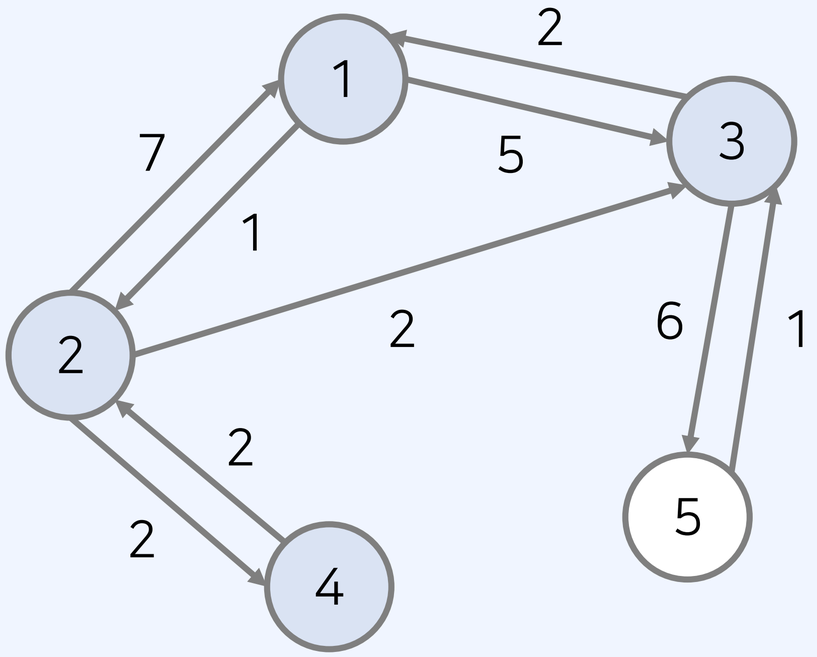
※ 파란색 선(거쳐가는 노드)상의 가까운 두 값을 더한다.

	1	2	3	4	5
1	0	1	3	3	9
2	4	0	2	2	8
3	2	3	0	5	6
4	6	2	4	0	10
5	3	4	1	6	0

플로이드 워셜 알고리즘: 동작 과정 살펴보기

JavaScript
최단 경로
플로이드 워셜
이해하기

- 각 단계마다 A에서 **특정한 노드 K를 거쳐 B로 갈 때 더 짧아지는 경우를 확인**한다.
- **4단계**: $D[A][B] = MIN(D[A][B], D[A][4] + D[4][B])$



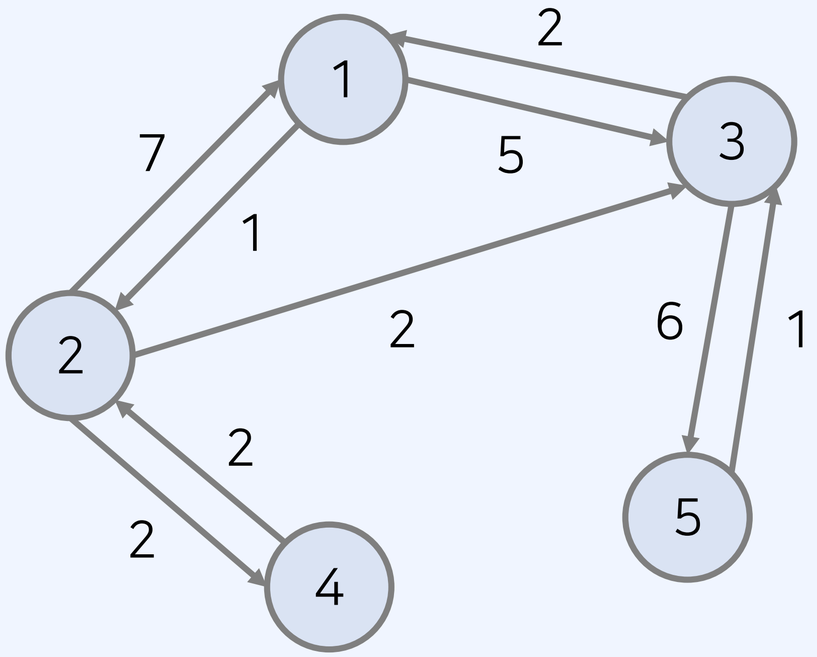
※ 파란색 선(거쳐가는 노드)상의 가까운 두 값을 더한다.

	1	2	3	4	5
1	0	1	3	3	9
2	4	0	2	2	8
3	2	3	0	5	6
4	6	2	4	0	10
5	3	4	1	6	0

플로이드 워셜 알고리즘: 동작 과정 살펴보기

JavaScript
최단 경로
플로이드 워셜
이해하기

- 각 단계마다 A에서 **특정한 노드 K를 거쳐 B로 갈 때 더 짧아지는 경우를 확인**한다.
- **5단계**: $D[A][B] = MIN(D[A][B], D[A][5] + D[5][B])$



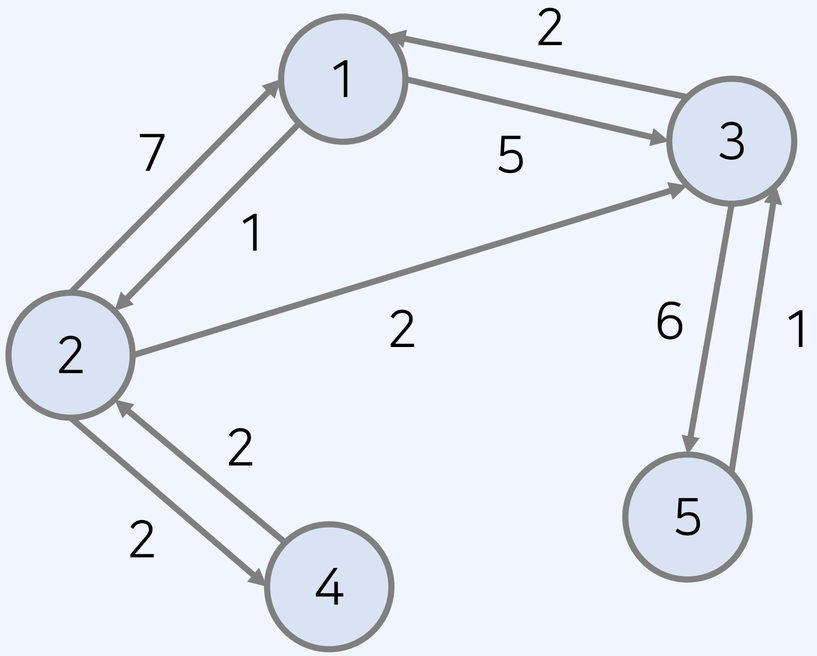
※ 파란색 선(거쳐가는 노드)상의 가까운 두 값을 더한다.

	1	2	3	4	5
1	0	1	3	3	9
2	4	0	2	2	8
3	2	3	0	5	6
4	6	2	4	0	10
5	3	4	1	6	0

플로이드 워셜 알고리즘: 동작 과정 살펴보기

JavaScript
최단 경로
플로이드 워셜
이해하기

- 최종적인 결과는 다음과 같다.



	1	2	3	4	5
1	0	1	3	3	9
2	4	0	2	2	8
3	2	3	0	5	6
4	6	2	4	0	10
5	3	4	1	6	0

플로이드 워셜 소스코드 예시

JavaScript
최단 경로
플로이드 워셜
이해하기

```
let INF = 1e9; // 무한을 의미하는 값으로 10억을 설정
let n = 5; // 노드의 개수

// graph[i][j]는 i에서 j로 가기 위한 초기 비용(간선 비용)
let graph = [ // 자기 자신으로 가는 비용은 0으로 초기화
  [INF, INF, INF, INF, INF, INF], // 인덱스 0은 사용하지 않음
  [INF, 0, 1, 5, INF, INF], // 1번 노드의 간선들
  [INF, 7, 0, 2, 2, INF], // 2번 노드의 간선들
  [INF, 2, INF, 0, INF, 6], // 3번 노드의 간선들
  [INF, INF, 2, INF, 0, INF], // 4번 노드의 간선들
  [INF, INF, INF, 1, INF, 0] // 5번 노드의 간선들
]

// 점화식에 따라 플로이드 워셜 알고리즘을 수행
for (let k = 1; k <= n; k++) {
  for (let a = 1; a <= n; a++) {
    for (let b = 1; b <= n; b++) {
      let cost = graph[a][k] + graph[k][b];
      graph[a][b] = Math.min(graph[a][b], cost);
    }
  }
}
```

플로이드 워셜 소스코드 예시

JavaScript
최단 경로
플로이드 워셜
이해하기

```
// 수행된 결과를 출력
for (let a = 1; a <= n; a++) {
  let line = '';
  for (let b = 1; b <= n; b++) {
    // 도달할 수 없는 경우, 무한(INFINITY)이라고 출력
    if (graph[a][b] == INF) line += 'INF ';
    // 도달할 수 있는 경우 거리를 출력
    else line += graph[a][b] + ' ';
  }
  console.log(line);
}
```

플로이드 워셜 알고리즘 성능 분석

JavaScript
최단 경로
플로이드 워셜
이해하기

- 노드의 개수가 N개일 때 알고리즘상으로 N번의 단계를 수행한다.
  - 각 단계마다 $O(N^2)$의 연산을 통해 현재 노드를 거쳐 가는 모든 경로를 고려한다.
- 따라서 플로이드 워셜 알고리즘의 총 시간 복잡도는 $O(N^3)$이다.