JavaScript 최단 경로
다익스트라 알고리즘 이해하기

다익스트라 알고리즘 이해하기 | 최단 경로를 찾아주는 다익스트라 알고리즘 이해하기

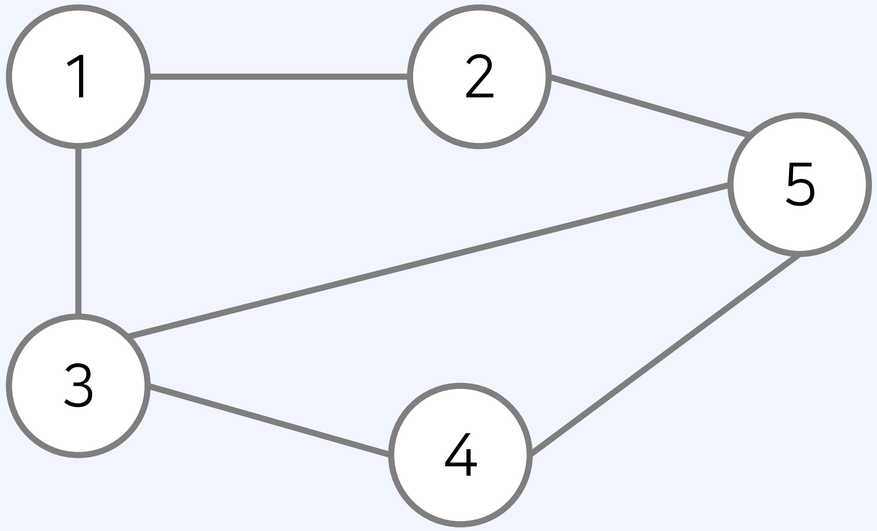
강사 나동빈

JavaScript
최단 경로
다익스트라 알고리즘 이해하기

그래프의 표현

JavaScript
최단 경로
다익스트라
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

최단 경로 문제란?

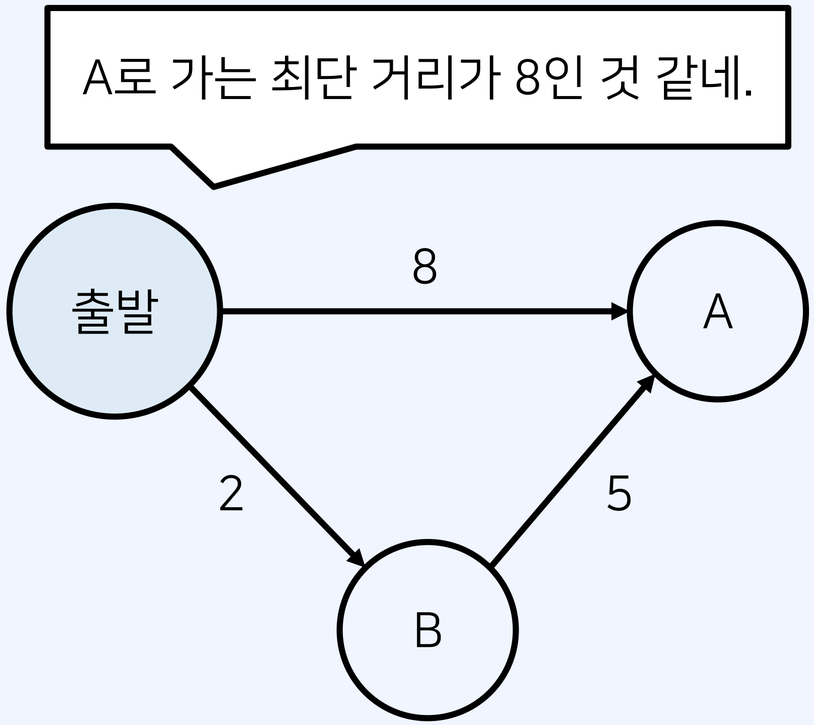
JavaScript
최단 경로
다익스트라
이해하기

- 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

최단 경로 알고리즘 동작 과정

JavaScript
최단 경로
다익스트라
이해하기

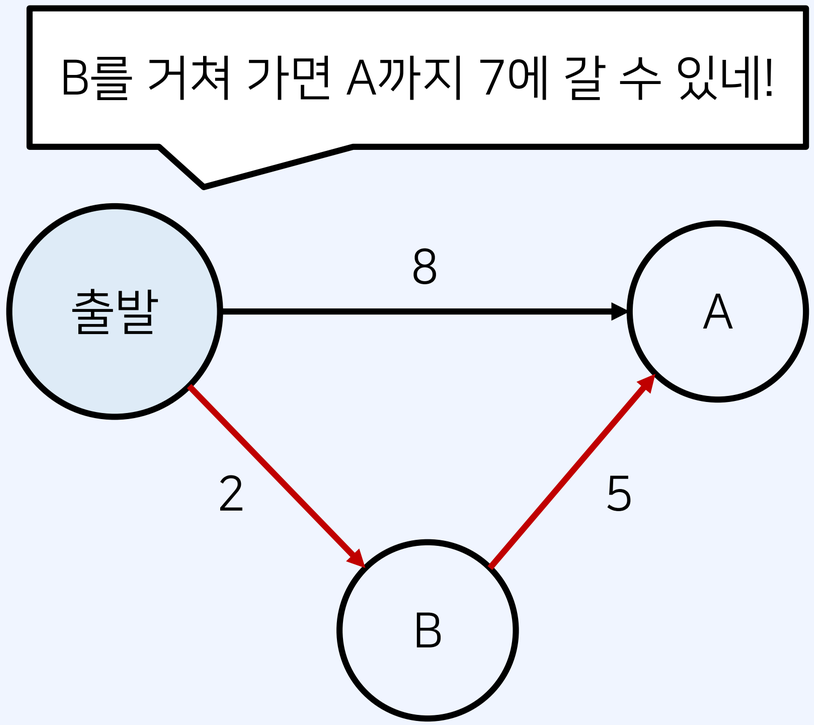
- 최단 거리 테이블은 각 노드에 대한 현재까지의 최단 거리 정보를 가진다.
- 처리 과정에서 더 짧은 경로를 찾으면 더 짧은 경로로 값을 갱신한다.



최단 경로 알고리즘 동작 과정

JavaScript
최단 경로
다익스트라
이해하기

- 최단 거리 테이블은 각 노드에 대한 현재까지의 최단 거리 정보를 가진다.
- 처리 과정에서 더 짧은 경로를 찾으면 더 짧은 경로로 값을 갱신한다.



다익스트라 최단 경로 알고리즘

JavaScript
최단 경로
다익스트라
이해하기

- **특정한 노드**에서 출발하여 **다른 모든 노드**로 가는 최단 경로를 계산한다.
- 다익스트라 최단 경로 알고리즘은 음의 간선이 없을 때 정상적으로 동작한다.
  - 현실 세계의 도로(간선)는 음의 간선으로 표현되지 않는다.
  - 음의 간선이 포함될 때는 벨만 포드(Bellman-Ford) 알고리즘을 사용할 수 있다.
- 다익스트라 최단 경로 알고리즘은 그리디 알고리즘으로 분류된다.
  - **매 상황에서 가장 비용이 적은 노드를 선택**해 임의의 과정을 반복한다.

다익스트라 최단 경로 알고리즘

- 알고리즘의 **동작 과정**은 다음과 같다.
  1. 출발 노드를 설정한다.
  2. 최단 거리 테이블을 초기화한다.
  3. 방문하지 않은 노드 중에서 **최단 거리가 가장 짧은 노드를 선택**한다.
  4. 해당 노드를 거쳐 다른 노드로 가는 비용을 계산하여 최단 거리 테이블을 갱신한다.

  → 최단 거리 테이블을 직접 갱신하지 않고, 우선순위 큐에 삽입하는 방식을 사용할 수도 있다.

  5. 위 과정에서 3번과 4번을 반복한다.

다익스트라 최단 경로 알고리즘

JavaScript
최단 경로
다익스트라
이해하기

- 다익스트라 최단 경로 알고리즘은 “그리디 알고리즘” 유형에 속한다.

→ **매 상황에서 방문하지 않은 가장 비용이 적은 노드를 선택**해 임의의 과정을 반복한다.

- 단계를 거치며 **한 번 처리된 노드의 최단 거리는 고정**되어 더 이상 바뀌지 않는다.
  - **한 단계당 하나의 노드에 대한 최단 거리를 확실히 찾는 것으로 이해**할 수 있다.
- 다익스트라 알고리즘을 수행한 뒤에 테이블에 각 노드까지의 최단 거리 정보가 저장된다.
  - 완전한 형태의 최단 경로를 구하려면 소스코드에 추가적인 기능을 더 넣어야 한다.

우선순위 큐(Priority Queue)

- 우선순위가 가장 높은 데이터를 가장 먼저 삭제하는 자료구조이다.
- 예를 들어 여러 개의 물건 데이터를 자료구조에 넣었다가 가치가 높은 물건 데이터부터 꺼내서 확인해야 하는 경우에 우선순위 큐를 이용할 수 있다.
- Python, C++, Java를 포함한 대부분의 프로그래밍 언어에서 **표준 라이브러리 형태로 지원**한다.

자료구조	추출되는 데이터
스택(Stack)	가장 나중에 삽입된 데이터
큐(Queue)	가장 먼저 삽입된 데이터
우선순위 큐(Priority Queue)	가장 우선순위가 높은 데이터

힙(Heap)

- 우선순위 큐(Priority Queue)를 구현하기 위해 사용하는 자료구조 중 하나다.
- **최소 힙(Min Heap)**과 **최대 힙(Max Heap)**이 있다.
- 다익스트라 최단 경로 알고리즘을 포함해 다양한 알고리즘에서 사용된다.

우선순위 큐 구현 방식	삽입 시간 복잡도	삭제 시간 복잡도
배열(리스트)	$O(1)$	$O(N)$
힙(Heap)	$O(logN)$	$O(logN)$

JavaScript의 힙(Heap) 라이브러리

JavaScript
최단 경로
다익스트라
이해하기

- JavaScript는 기본적으로 우선순위 큐를 라이브러리로 제공하지 않는다.
- 최단 경로 알고리즘 등에서 **힙(heap)**이 필요한 경우 **별도의 라이브러리를 사용**해야 한다.

JavaScript의 힙(Heap) 라이브러리

- https://github.com/janogonzalez/priorityqueuejs
- index.js 소스 코드를 가져온 뒤에 다음과 같이 사용할 수 있다.

```
// 최대힙(Max Heap)
let pq = new PriorityQueue(function(a, b) {
  return a.cash - b.cash;
});

pq.enq({cash: 250, name: 'Doohyun Kim'});
pq.enq({cash: 300, name: 'Gildong Hong'});
pq.enq({cash: 150, name: 'Minchul Han'});
console.log(pq.size()); // 3
console.log(pq.deq()); // {cash: 300, name: 'Gildong Hong'}
console.log(pq.peek()); // {cash: 250, name: 'Doohyun Kim'}
console.log(pq.size()); // 2
```

다익스트라 최단 경로 알고리즘의 특징

- 단계마다 방문하지 않은 노드 중에서 최단 거리가 가장 짧은 노드를 선택한다.

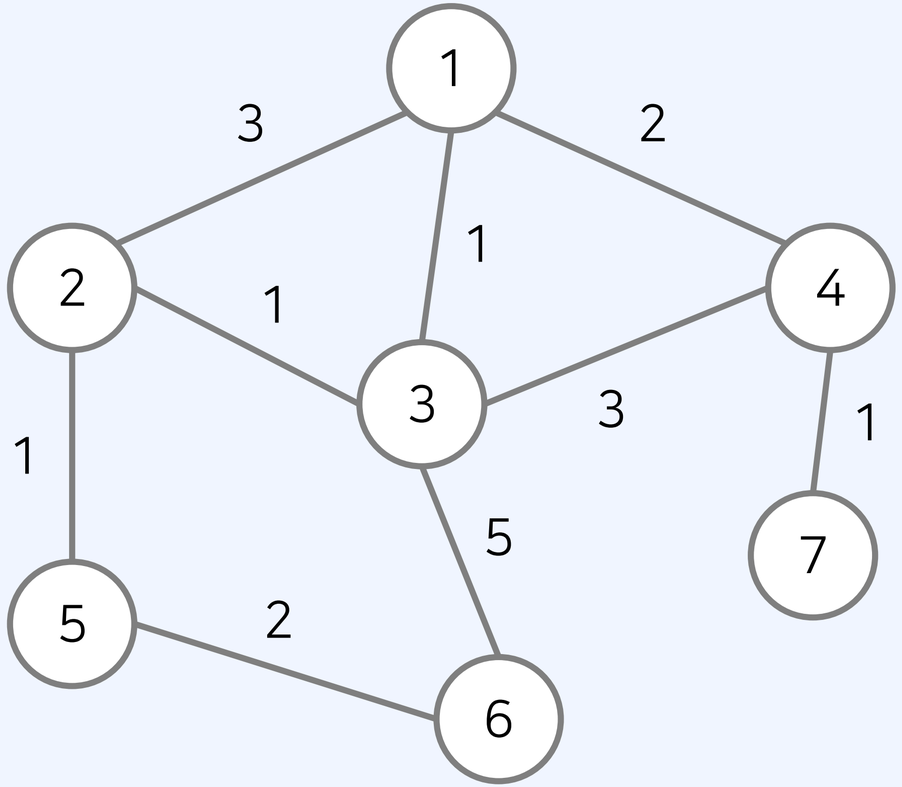
→ 이를 위해 **힙(Heap)** 자료구조를 이용한다.

→ 다시 말해, 현재 상황에서 가장 가까운 노드를 저장해 놓기 위해서 힙 자료구조를 이용한다.

- 현재의 최단 거리가 가장 짧은 노드를 선택해야 하므로 최소 힙을 사용한다.

다익스트라 알고리즘의 동작 과정

Distance	Node
0	1



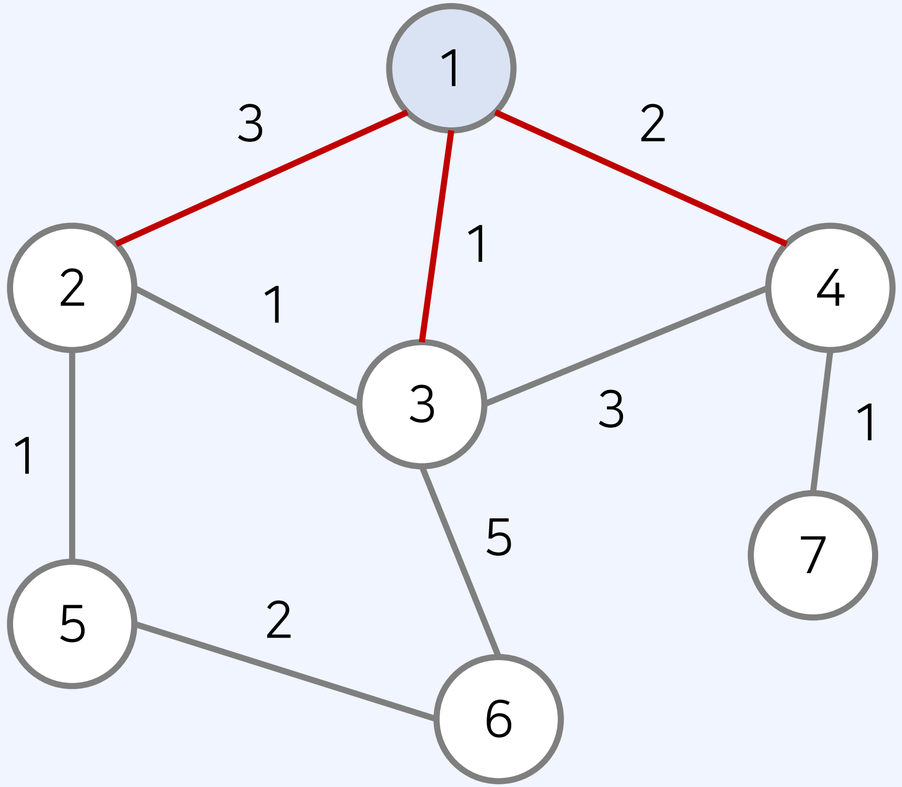
Node	1	2	3	4	5	6	7
Distance	INF	INF	INF	INF	INF	INF	INF

다익스트라 알고리즘의 동작 과정

Distance	Node
0 + 1 = 1	3

Distance	Node
0 + 2 = 2	4

Distance	Node
0 + 3 = 3	2



Distance	Node
0	1

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

다익스트라 알고리즘의 동작 과정

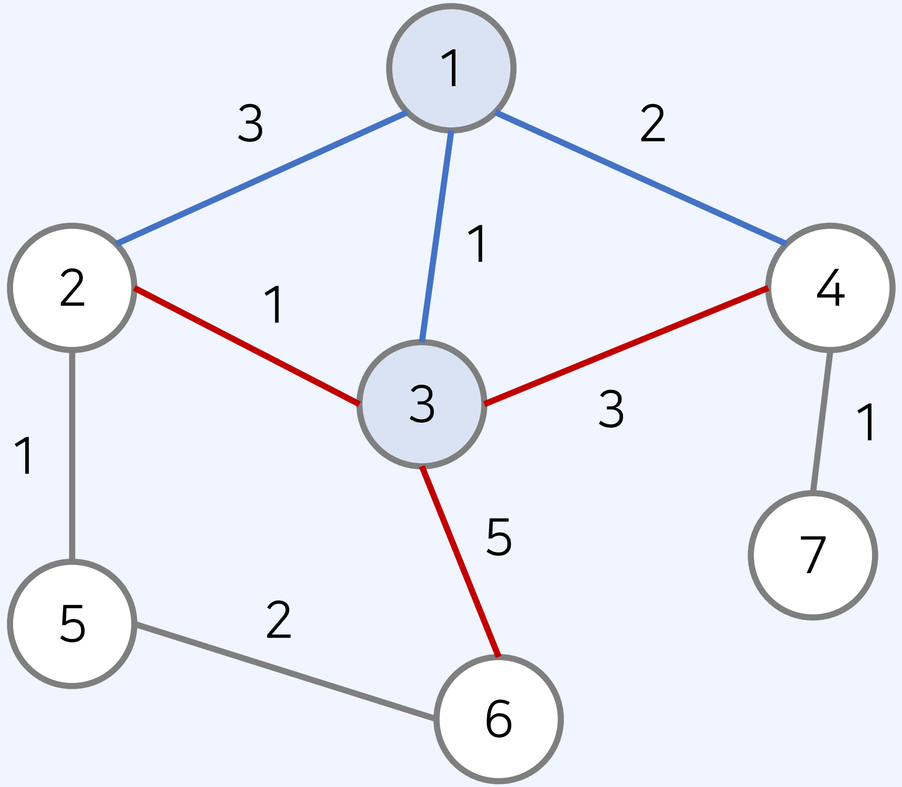
Distance	Node
2	4

Distance	Node
1 + 1 = 2	2

Distance	Node
3	2

Distance	Node
1 + 3 = 4	4

Distance	Node
1 + 5 = 6	6



Distance	Node
1	3

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

다익스트라 알고리즘의 동작 과정

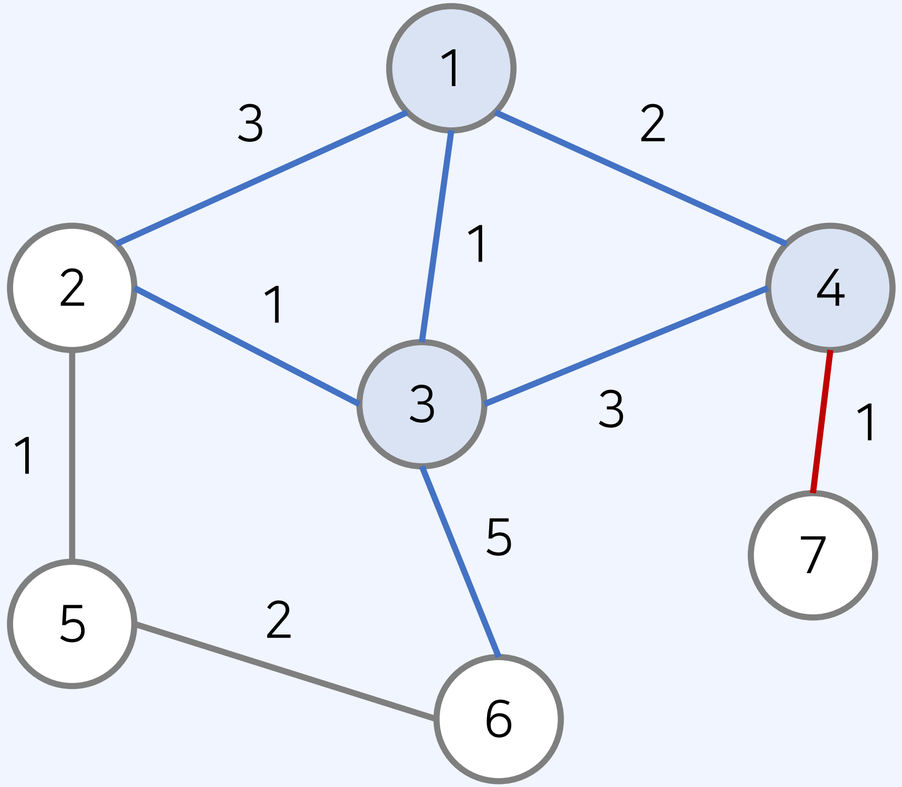
Distance	Node
2	2

Distance	Node
3	2

Distance	Node
2 + 1 = 3	7

Distance	Node
4	4

Distance	Node
6	6



Distance	Node
2	4

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

다익스트라 알고리즘의 동작 과정

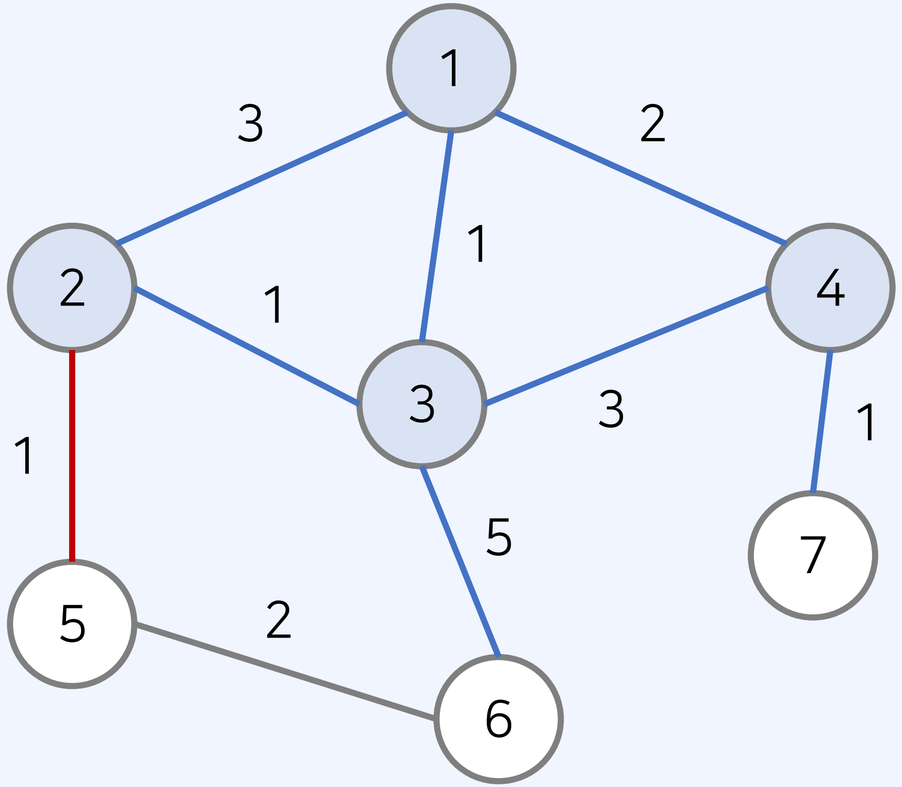
Distance	Node
3	2

Distance	Node
3	7

Distance	Node
2 + 1 = 3	5

Distance	Node
4	4

Distance	Node
6	6



Distance	Node
2	2

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

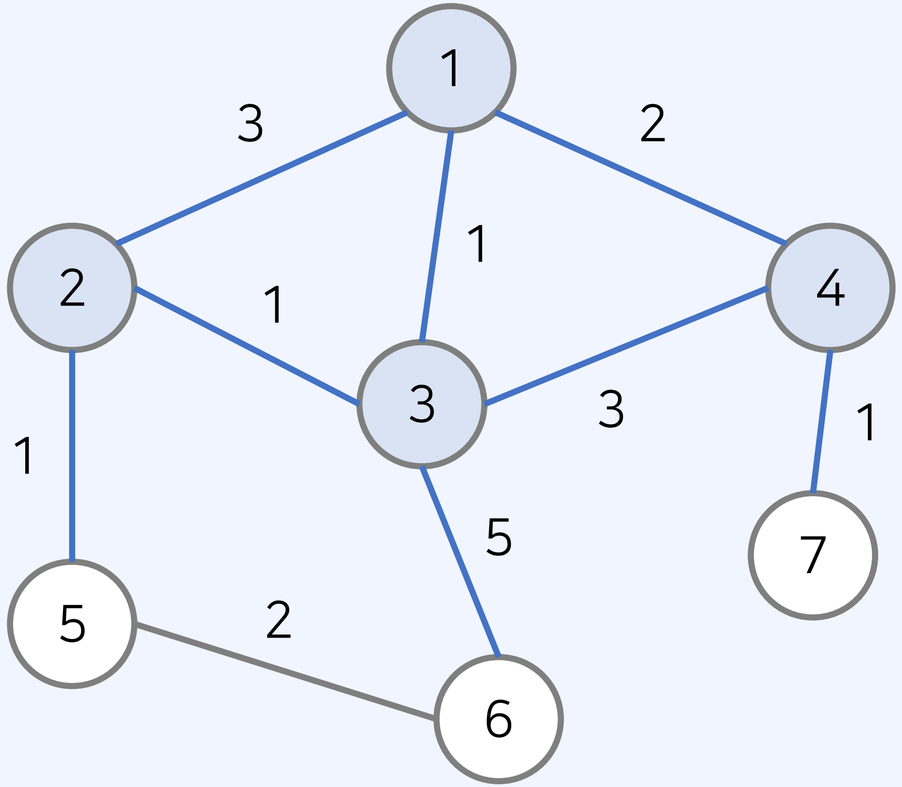
다익스트라 알고리즘의 동작 과정

Distance	Node
3	7

Distance	Node
3	5

Distance	Node
4	4

Distance	Node
6	6



Distance	Node
3	2

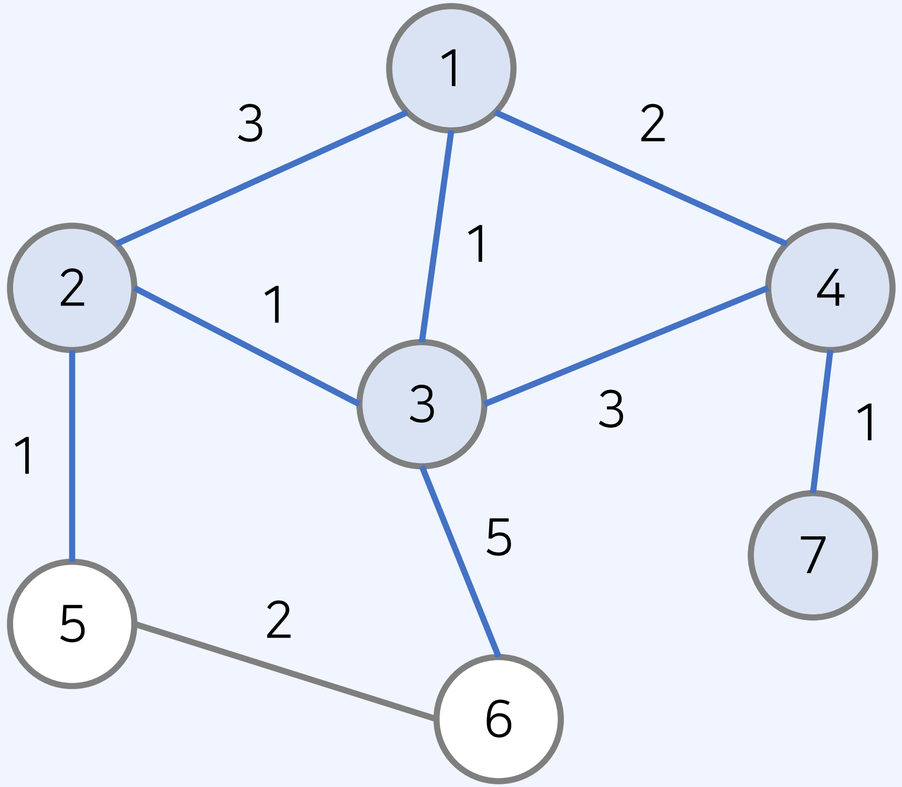
Node	1	2	3	4	5	6	7
Distance	0	2	1	2	INF	INF	INF

다익스트라 알고리즘의 동작 과정

Distance	Node
3	5

Distance	Node
4	4

Distance	Node
6	6



Distance	Node
3	7

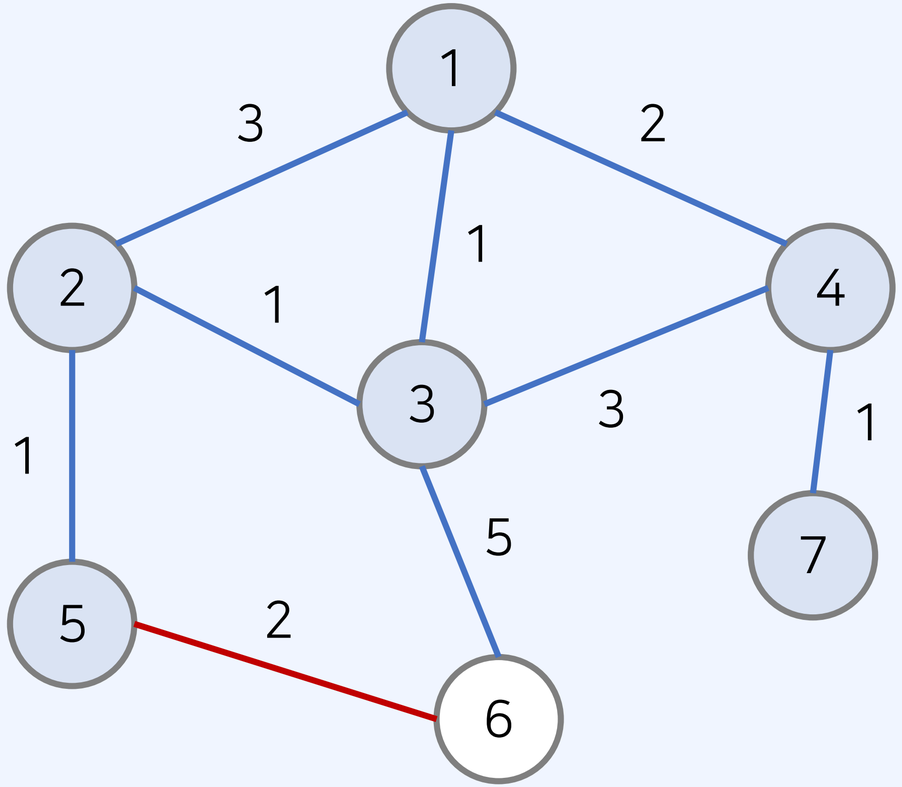
Node	1	2	3	4	5	6	7
Distance	0	2	1	2	INF	INF	3

다익스트라 알고리즘의 동작 과정

Distance	Node
4	4

Distance	Node
3 + 2 = 5	6

Distance	Node
6	6



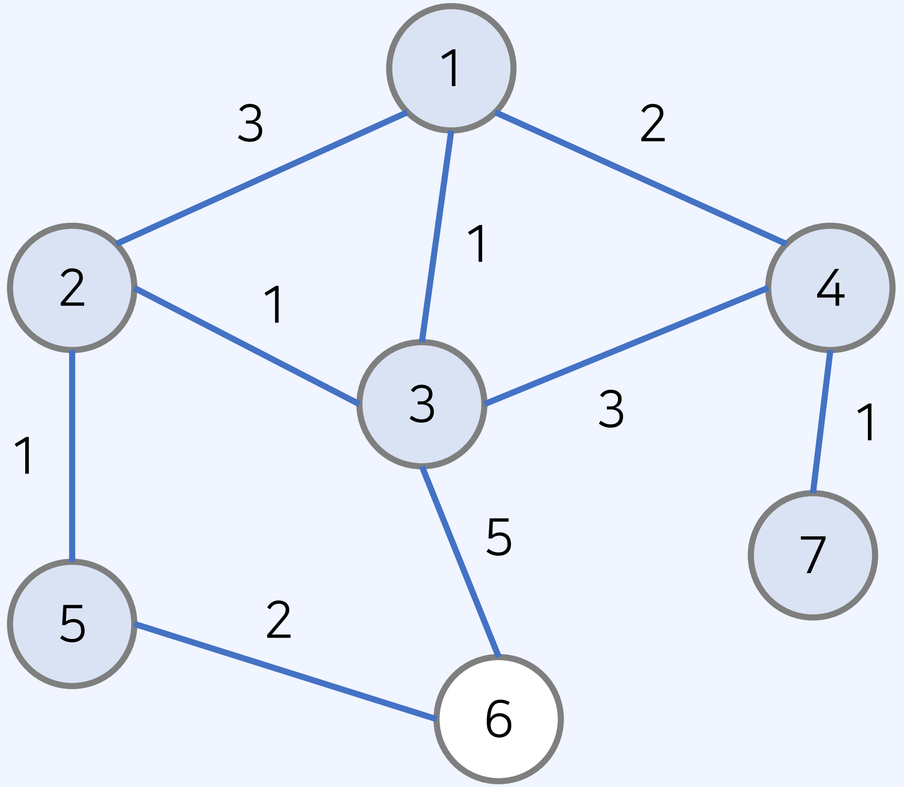
Distance	Node
3	5

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

다익스트라 알고리즘의 동작 과정

Distance	Node
5	6

Distance	Node
6	6



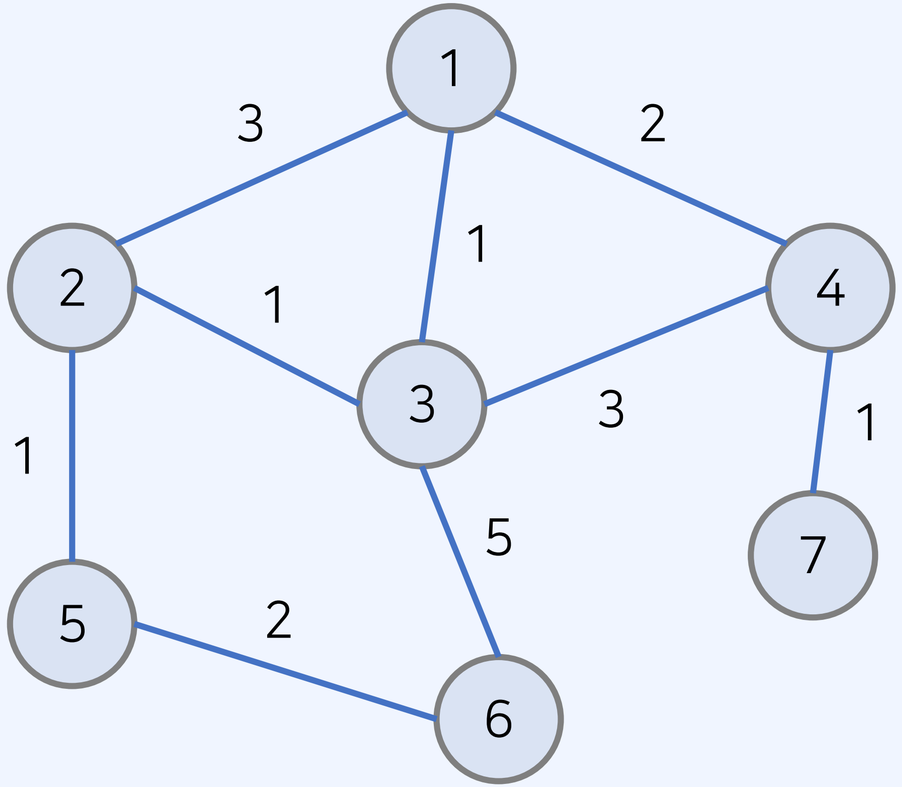
Distance	Node
4	4

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

JavaScript 최단 경로
다익스트라 이해하기

다익스트라 알고리즘의 동작 과정

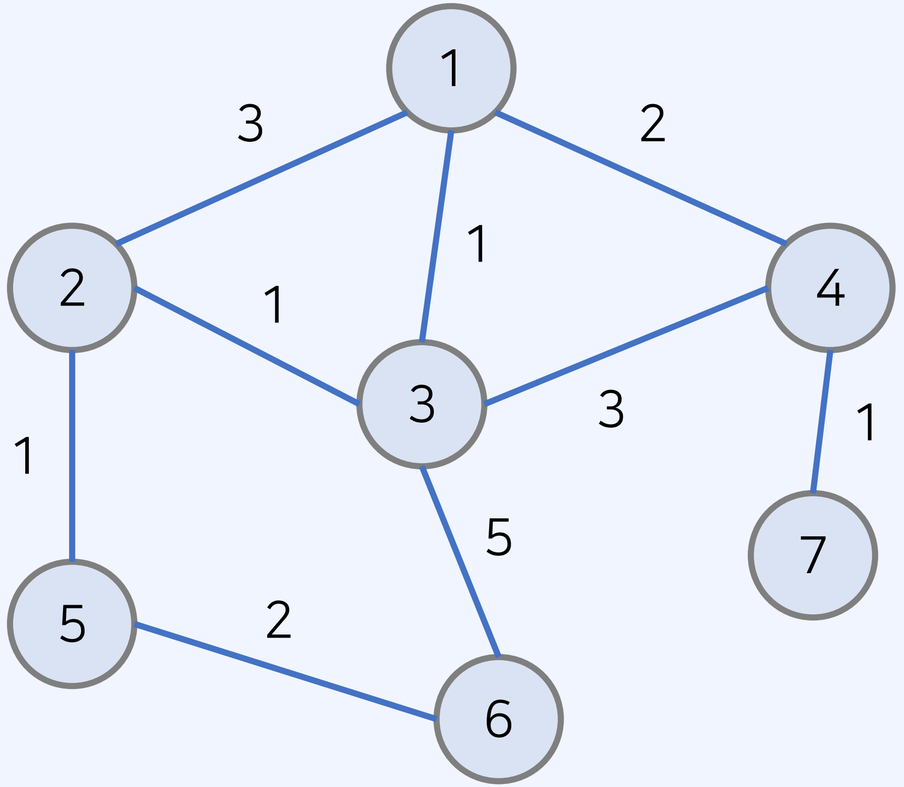
Distance	Node
6	6



Distance	Node
5	6

Node	1	2	3	4	5	6	7
Distance	0	2	1	2	3	5	3

다익스트라 알고리즘의 동작 과정



Distance	Node
6	6

Node	1	2	3	4	5	6	7
Distance	0	2	1	2	3	5	3

다익스트라 알고리즘의 소스코드 예시

JavaScript
최단 경로
다익스트라
이해하기

```
function dijkstra() { // 다익스트라(Dijkstra) 알고리즘 수행
  let pq = new PriorityQueue((a, b) => b[0] - a[0]); // 최소힙(Min Heap)
  // 시작 노드로 가기 위한 최단 거리는 0으로 우선순위 큐에 삽입
  pq.enq([0, start]);
  distance[start] = 0;
  while (pq.size() != 0) { // 우선순위 큐가 비어있지 않다면
    // 가장 최단 거리가 짧은 노드에 대한 정보 꺼내기
    let [dist, now] = pq.deq();
    // 현재 노드가 이미 처리된 적이 있는 노드라면 무시
    if (distance[now] < dist) continue;
    // 현재 노드와 연결된 다른 인접한 노드들을 확인
    for (let i of graph[now]) {
      let cost = dist + i[1];
      // 현재 노드를 거쳐서, 다른 노드로 이동하는 거리가 더 짧은 경우
      if (cost < distance[i[0]]) {
        distance[i[0]] = cost;
        pq.enq([cost, i[0]]);
      }
    }
  }
}
```

다익스트라 알고리즘의 소스코드 예시

JavaScript
최단 경로
다익스트라
이해하기

```
let INF = 1e9; // 무한을 의미하는 값으로 10억을 설정
let n = 7; // 노드의 개수
let start = 1; // 시작 노드 번호
// 각 노드에 연결되어 있는 노드에 대한 정보를 담는 리스트를 만들기
let graph = [ // 각 간선은 [노드, 비용] 형태
  [],
  [[2, 3], [3, 1], [4, 2]], // 1번 노드의 간선들
  [[1, 3], [3, 1], [5, 1]], // 2번 노드의 간선들
  [[1, 1], [2, 1], [4, 3], [6, 5]], // 3번 노드의 간선들
  [[1, 2], [3, 3], [7, 1]], // 4번 노드의 간선들
  [[2, 1], [6, 2]], // 5번 노드의 간선들
  [[3, 5], [5, 2]], // 6번 노드의 간선들
  [[4, 1]] // 7번 노드의 간선들
]
// 최단 거리 테이블을 모두 무한으로 초기화
let distance = new Array(n + 1).fill(INF);

// 다익스트라 알고리즘을 수행
dijkstra();
// 모든 노드로 가기 위한 최단 거리를 출력
for (let i = 1; i <= n; i++) {
  // 도달할 수 없는 경우 무한(INFINITY)이라고 출력
  if (distance[i] == INF) console.log('INFINITY');
  // 도달할 수 있는 경우 거리를 출력
  else console.log(distance[i]);
}
```

다익스트라 알고리즘의 시간 복잡도

JavaScript
최단 경로
다익스트라
이해하기

- 힙 자료구조를 이용하는 다익스트라 알고리즘의 시간 복잡도는 $\boldsymbol{O(ElogV)}$이다.
- 노드를 하나씩 꺼내 검사하는 반복문(while문)은 노드의 개수 V 이상의 횟수로는 처리되지 않다.
  - 결과적으로 현재 우선순위 큐에서 꺼낸 노드와 연결된 다른 노드들을 확인하는 총 횟수는 최대 간선의 개수(E)만큼 연산이 수행될 수 있다.
- 직관적으로 전체 과정은 E개의 원소를 우선순위 큐에 넣었다가 모두 빼내는 연산과 매우 유사하다.
  - 시간 복잡도를 $O(ElogE)$로 판단할 수 있다.
  - 중복 간선을 포함하지 않는 경우에 이를 $O(ElogV)$로 정리할 수 있다.
    - $O(ElogE) \rightarrow O(ElogV^2) \rightarrow O(2ElogV) \rightarrow O(ElogV)$