JavaScript 최단 경로

최단 경로 문제 풀이

최단 경로 문제 풀이 | 코딩 테스트에서 자주 등장하는 최단 경로 이해하기

강사 나동빈

JavaScript 최단 경로

최단 경로 문제 풀이

특정한 최단 경로: https://www.acmicpc.net/problem/1504

혼자 힘으로 풀어보기

문제 제목: 특정한 최단 경로

문제 난이도: ★★★☆☆

문제 유형: 최단 경로, 다익스트라

추천 풀이 시간: 50분

문제 풀이 핵심 아이디어

- 노드의 개수가 800개이므로, **다익스트라**를 활용해 최단 경로를 계산할 수 있다.
- 임의로 주어진 두 개의 정점 A와 B를 반드시 통과하는 최단 경로를 계산한다.
- 따라서 아래의 두 경우 중에서 더 짧은 경우를 계산하면 된다.

1. $1 \rightarrow A \rightarrow B \rightarrow N$
2. $1 \rightarrow B \rightarrow A \rightarrow N$

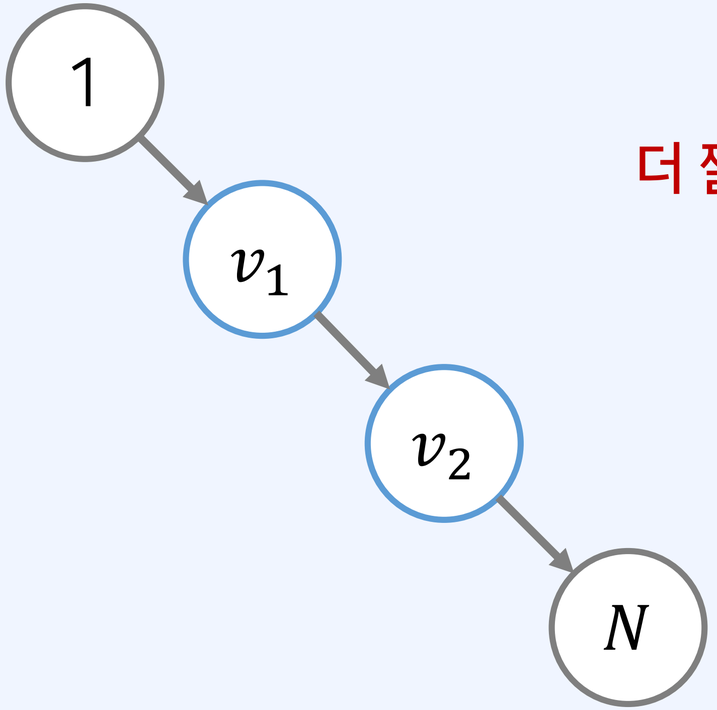
- 이를 위해 총 3번의 다익스트라 알고리즘을 수행하면 된다.

1. 노드 1에서 출발하여 A, B에 도착할 때
2. 노드 A에서 출발하여 B, N에 도착할 때
3. 노드 B에서 출발하여 A, N에 도착할 때

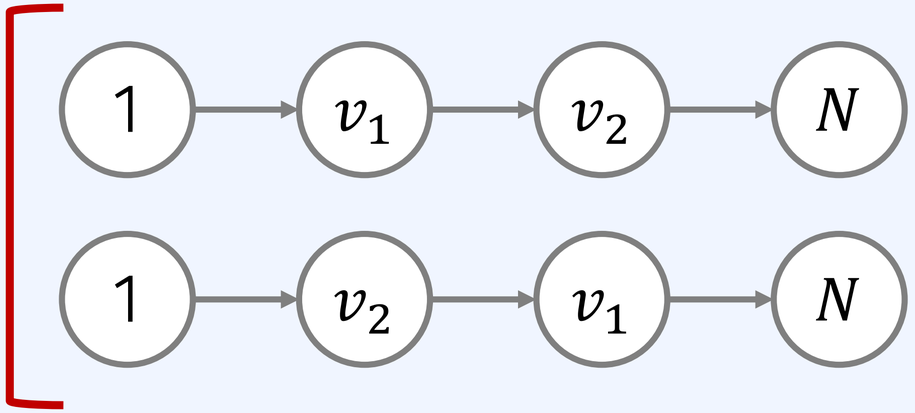
특정한 최단 경로: https://www.acmicpc.net/problem/1504

문제 풀이 핵심 아이디어

- 노드의 개수가 800개이므로, **다익스트라**를 활용해 최단 경로를 계산할 수 있다.
- **경로**: $X \rightarrow Y$로 갈 때 K를 거쳐야 한다. (**최단 경로**: $X \rightarrow K + K \rightarrow Y$)



더 짧은 것 선택



JavaScript 최단 경로
최단 경로 문제 풀이

정답 코드 예시

```
function dijkstra(start) { // 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]]);
      }
    }
  }
}
```

정답 코드 예시

```
let file = require('fs').readFileSync('/dev/stdin');
let input = file.toString().split('\n');

let INF = 1e9; // 무한을 의미하는 값으로 10억을 설정
// 노드의 개수, 간선의 개수를 입력받기
let [n, m] = input[0].split(' ').map(Number);
// 각 노드에 연결되어 있는 노드에 대한 정보를 담는 배열을 만들기
let graph = [];
for (let i = 0; i <= n + 1; i++) graph.push([]);
// 모든 간선 정보를 입력받기
for (let i = 1; i <= m; i++) {
  let [a, b, c] = input[i].split(' ').map(Number);
  // a번 노드에서 b번 노드로 가는 비용이 c라는 의미
  graph[a].push([b, c]);
  graph[b].push([a, c]);
}
// 꼭 거쳐야 하는 a와 b 노드 입력받기
let [a, b] = input[m + 1].split(' ').map(Number);
```

정답 코드 예시

```
let distance = new Array(n + 1).fill(INF); // 최단 거리 테이블 초기화
dijkstra(1); // 다익스트라 알고리즘 수행
let distance_1_to_a = distance[a];
let distance_1_to_b = distance[b];

distance = new Array(n + 1).fill(INF); // 최단 거리 테이블 초기화
dijkstra(a); // 다익스트라 알고리즘 수행
let distance_a_to_b = distance[b];
let distance_a_to_n = distance[n];

distance = new Array(n + 1).fill(INF); // 최단 거리 테이블 초기화
dijkstra(b); // 다익스트라 알고리즘 수행
let distance_b_to_a = distance[a];
let distance_b_to_n = distance[n];

let route1 = distance_1_to_a + distance_a_to_b + distance_b_to_n
let route2 = distance_1_to_b + distance_b_to_a + distance_a_to_n

let result = Math.min(route1, route2);
if (result >= INF) console.log(-1) // 경로가 없는 경우
else console.log(result);
```

거의 최단 경로: https://www.acmicpc.net/problem/5719

혼자 힘으로 풀어보기

문제 제목: 거의 최단 경로

문제 난이도: ★★★★☆

문제 유형: 최단 경로, 다익스트라

추천 풀이 시간: 60분

거의 최단 경로: https://www.acmicpc.net/problem/5719

문제 풀이 핵심 아이디어

- 본 문제는 **다익스트라** 알고리즘을 활용해 해결할 수 있는 문제다.
- **"최단 경로에 포함되지 않는 도로로만 이루어진 경로 중에서 가장 짧은 것"**을 찾자.

[문제 해결 방법]

1. 먼저 다익스트라 알고리즘을 이용해 최단 경로를 한 번 계산한다.
2. 이후에 BFS를 이용해 "최단 경로들"에 포함된 모든 간선을 찾는다.
   참고로 최단 경로는 하나가 아닐 수 있으며, 모든 간선을 다 찾아야 한다.
3. 결과적으로 그러한 간선들을 모두 지운 뒤에, 다시 다익스트라를 사용한다.

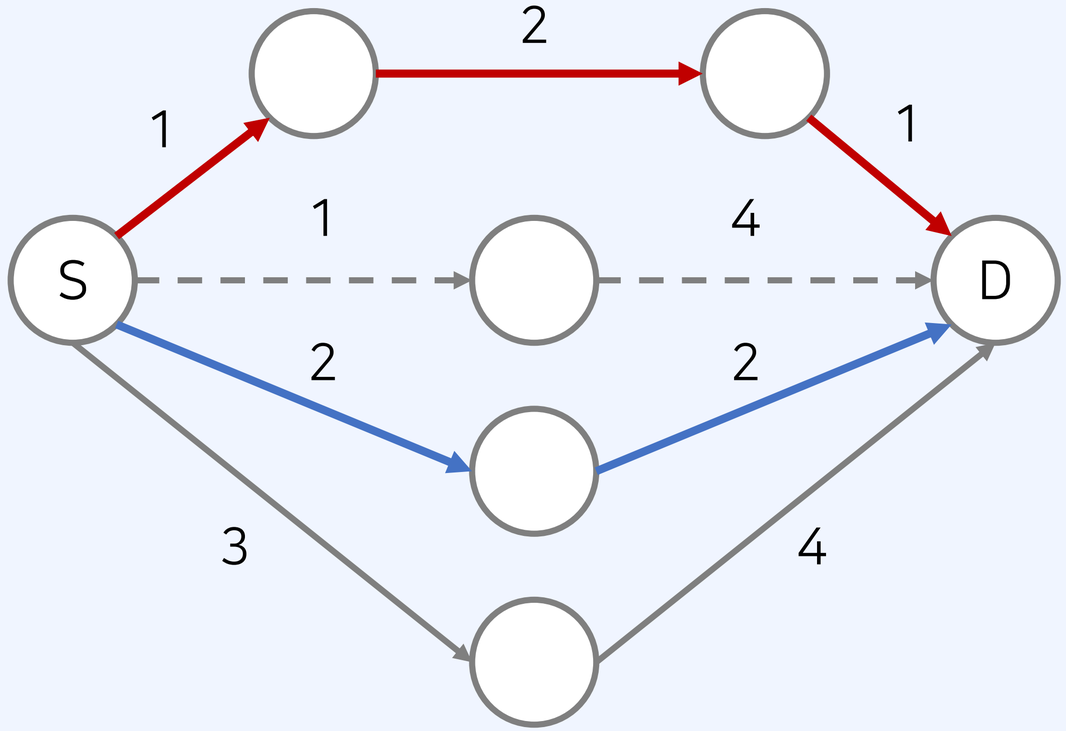
[참고]

- 기존의 다익스트라 함수를 전혀 변경하지 않고, 그대로 사용할 수 있다.
  다만, 경로 추적을 위한 $BFS()$ 함수를 구현해야 한다.

문제 풀이 핵심 아이디어

- 아래의 경우에서 ①과 ②에 포함된 간선들 제거하고, 최단 경로를 다시 계산한다.

→ 최단 경로에 포함된 간선은 $BFS(\cdot)$로 찾기 가능하다.



거의 최단 경로: https://www.acmicpc.net/problem/5719

JavaScript 최단 경로
최단 경로 문제 풀이

정답 코드 예시

```
function dijkstra() {
  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]]);
      }
    }
  }
}
```

거의 최단 경로: https://www.acmicpc.net/problem/5719

JavaScript 최단 경로
최단 경로 문제 풀이

정답 코드 예시

```
// 최단 경로 역추적 함수
function bfs() {
  let queue = new Queue();
  let visited = new Set(); // 특정한 노드 방문 여부
  queue.enqueue(end); // 도착 지점(end)을 큐에 삽입
  let removes = []; // 삭제할 간선들(결과)
  while (queue.getLength() != 0) { // 큐가 빌 때까지 반복하기
    let now = queue.dequeue();
    if (now == start) { // 시작점에 도착한 경우
      continue; // 모든 최단 경로를 확인하기 위해 break 대신 continue
    }
    for (let i of reversed_graph[now]) { // 현재 노드와 연결된 간선들 확인
      let cost = distance[i[0]] + i[1];
      // 최단 경로에 포함된 간선인 경우 삭제 목록에 추가
      if (cost == distance[now]) {
        removes.push([i[0], now]);
        // 각 "직전 노드"는 한 번씩만 방문
        if (!visited.has(i[0])) {
          queue.enqueue(i[0]);
          visited.add(i[0]);
        }
      }
    }
  }
  return removes;
}
```

JavaScript 최단 경로
최단 경로 문제 풀이

정답 코드 예시

```
// 새로운 그래프를 구성하는 함수
function getNewGraph() {
  // 최단 경로 역추적: 모든 최단 경로에 포함된 간선 쌍 (a, b)들을 계산
  removes = bfs();
  let newGraph = [];
  for (let i = 0; i < n; i++) newGraph.push([]);
  for (let a = 0; a < n; a++) {
    for (let [b, c] of graph[a]) {
      // 삭제 목록에 포함되지 않은 간선만 넣기
      let check = true;
      for (let [x, y] of removes) {
        if (a == x && b == y) check = false;
      }
      if (check) newGraph[a].push([b, c]);
    }
  }
  return newGraph;
}

let file = require('fs').readFileSync('/dev/stdin');
let input = file.toString().split('\n');

let INF = 1e9; // 무한을 의미하는 값으로 10억을 설정
let line = 0;
```

거의 최단 경로: https://www.acmicpc.net/problem/5719

JavaScript 최단 경로
최단 경로 문제 풀이

정답 코드 예시

```
while (true) {
  [n, m] = input[line].split(' ').map(Number); // 노드의 개수, 간선의 개수를 입력받기
  if (n == 0 && m == 0) break; // 테스트 케이스 종료
  // 시작 노드와 도착 노드 입력받기
  [start, end] = input[line + 1].split(' ').map(Number);
  graph = []; // 각 노드에 연결되어 있는 노드에 대한 정보를 담는 배열 만들기
  for (let i = 0; i < n; i++) graph.push([]);
  reversed_graph = []; // 경로 추적을 위한 역순 그래프
  for (let i = 0; i < n; i++) reversed_graph.push([]);
  // 모든 간선 정보를 입력받기
  for (let i = line + 2; i < line + 2 + m; i++) {
    let [a, b, c] = input[i].split(' ').map(Number);
    // a번 노드에서 b번 노드로 가는 비용이 c라는 의미
    graph[a].push([b, c]);
    reversed_graph[b].push([a, c]);
  }
  distance = new Array(n).fill(INF); // 최단 거리 테이블을 모두 무한으로 초기화
  dijkstra(); // 다익스트라 알고리즘을 수행

  graph = getNewGraph();

  distance = new Array(n).fill(INF); // 최단 거리 테이블을 모두 무한으로 초기화
  dijkstra(); // 다익스트라 알고리즘을 수행
  if (distance[end] == INF) console.log(-1); // 도달할 수 없는 경우, -1을 출력
  else console.log(distance[end]); // 도달할 수 있는 경우 거리를 출력
  line += m + 2;
}
```