JavaScript DFS 알고리즘
DFS 문제 풀이

DFS 문제 풀이 | 코딩 테스트에서 자주 등장하는 DFS 알고리즘 이해하기

강사 나동빈

JavaScript
DFS 알고리즘

DFS 문제 풀이

노드사이의 거리: https://www.acmicpc.net/problem/1240

혼자 힘으로 풀어보기

문제 제목: 노드사이의 거리

문제 난이도: ★★☆☆☆

문제 유형: 깊이 우선 탐색, 그래프 순회, 트리(Tree)

추천 풀이 시간: 40분

노드사이의 거리: https://www.acmicpc.net/problem/1240

문제 해결 아이디어

- 본 문제에서는 트리가 **양방향 간선**으로 구성되어 있다고 가정해보자.
- 문제에서 주어지는 노드의 개수 N은 최대 1000이다.

  주어지는 그래프는 항상 **트리(tree)** 형식이므로, 간선의 개수는 $N-1$개이다.
- **트리에서 노드 A와 B를 잇는 경로는 오직 1개만 존재**한다.

  따라서 트리 내에 존재하는 노드 A와 B의 거리를 구하기 위한 시간 복잡도는 $O(N)$이다.
- 쿼리의 개수 M은 최대 1000이다.

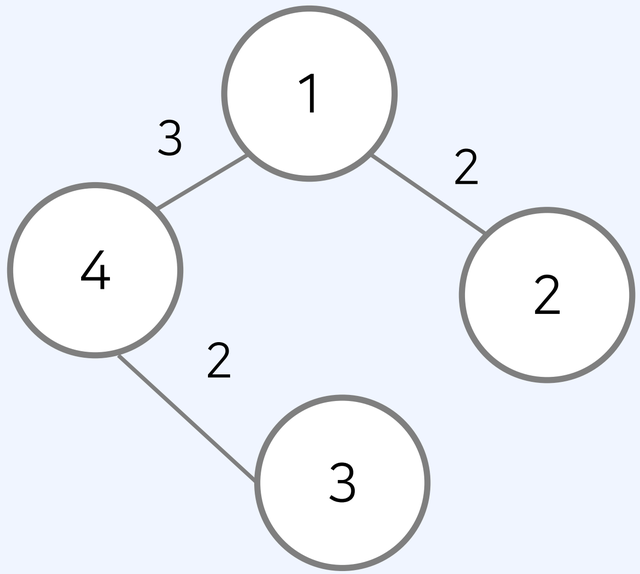
  따라서, 매 쿼리마다 노드 A와 B의 거리를 계산해도 요구되는 시간 복잡도는 $O(NM)$이다.

노드사이의 거리: https://www.acmicpc.net/problem/1240

문제 해결 아이디어

[핵심 아이디어 요약]

- 트리에서는 임의의 두 노드 간의 **경로**가 오직 1개이다.
- 따라서 트리에서는 BFS가 아닌 DFS로도 간단히 최단 거리를 계산할 수 있다.
- 단순히 매 쿼리(query)마다, 노드 A에서 B까지의 거리를 계산한다.



- 1번 노드부터 2번 노드까지의 거리: 2
- 3번 노드부터 2번 노드까지의 거리: 7

노드사이의 거리: https://www.acmicpc.net/problem/1240

정답 코드 예시


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

let [n, m] = input[0].split(' ').map(Number); // 노드의 개수(N), 쿼리의 개수(M)
let graph = []; // 트리 정보 입력받기
for (let i = 1; i <= n; i++) graph[i] = [];
for (let i = 1; i < n; i++) { // 노드 X와 노드 Y는 서로 연결
  let [x, y, cost] = input[i].split(' ').map(Number);
  graph[x].push([y, cost]);
  graph[y].push([x, cost]);
}

function dfs(x, dist) { // 깊이 우선 탐색(DFS) 함수 구현
  if (visited[x]) return; // 각 노드는 한 번만 방문
  visited[x] = true; // 방문 처리
  distance[x] = dist;
  for (let [y, cost] of graph[x]) dfs(y, dist + cost);
}

for (let i = 0; i < m; i++) { // 각 쿼리(query)마다 매번 DFS를 수행
  let [x, y] = input[n + i].split(' ').map(Number);
  visited = new Array(n + 1).fill(false);
  distance = new Array(n + 1).fill(-1);
  dfs(x, 0); // 노드 X에서 출발했을 때의 모든 노드까지의 거리 계산
  console.log(distance[y]); // Y까지의 최단 거리
}
```

트리: https://www.acmicpc.net/problem/4803

혼자 힘으로 풀어보기

문제 제목: 트리

문제 난이도: ★★★☆☆

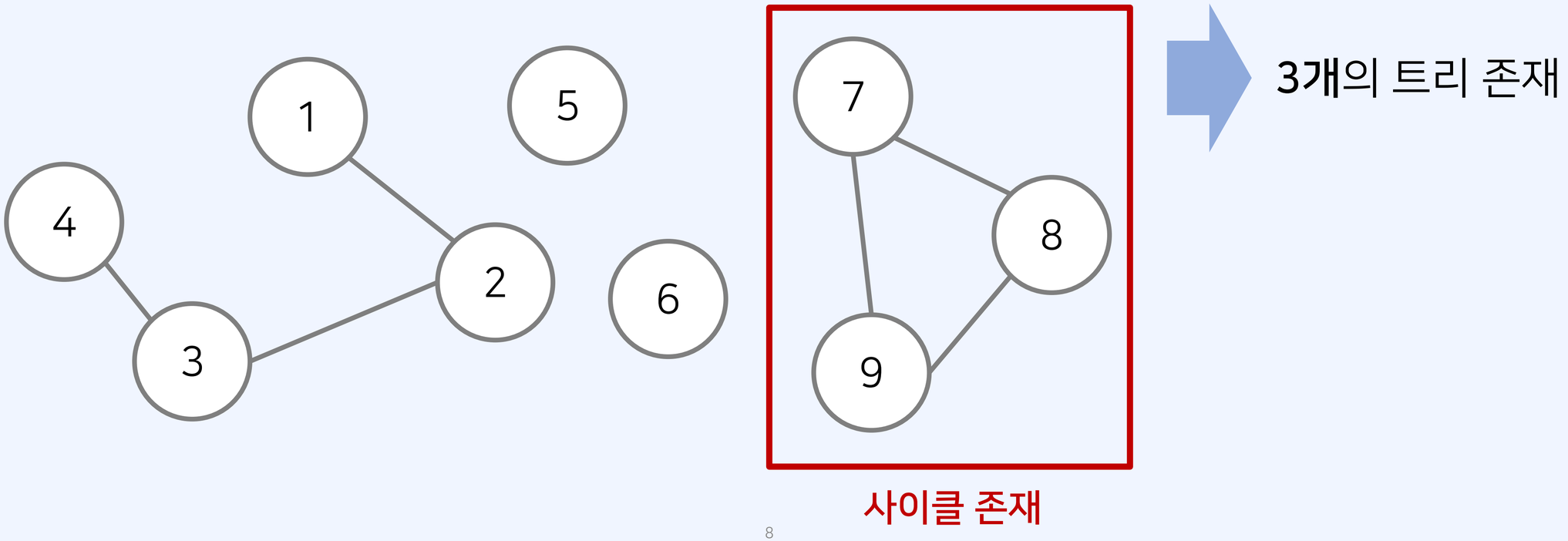
문제 유형: 깊이 우선 탐색, 그래프 내 사이클 판별, 트리(Tree)

추천 풀이 시간: 50분

트리: https://www.acmicpc.net/problem/4803

문제 해결 아이디어

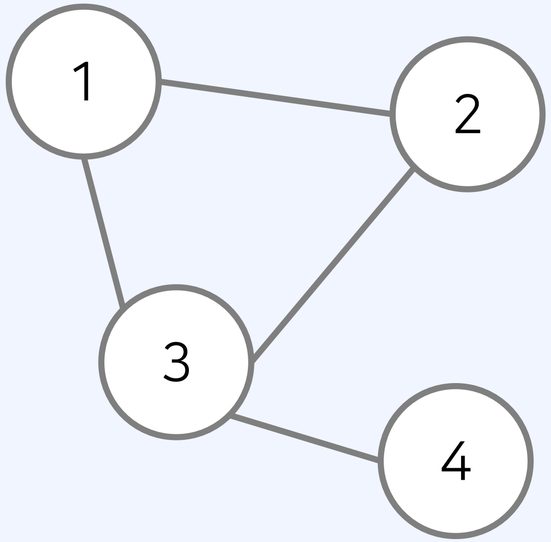
- 하나의 그래프 안에 포함된 **트리(tree)의 개수를 세는 문제**다.
- **트리**: 사이클이 없는 연결 요소
  - 트리의 정의에 따라서 **DFS를 이용해 트리의 개수를 계산**하여 문제를 해결할 수 있다.



트리: https://www.acmicpc.net/problem/4803

문제 해결 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클이다.
- 단, 무방향 그래프이므로 직전 노드는 제외한다.

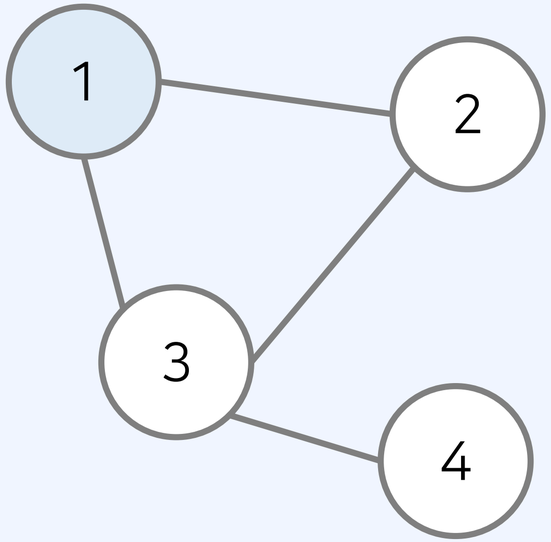


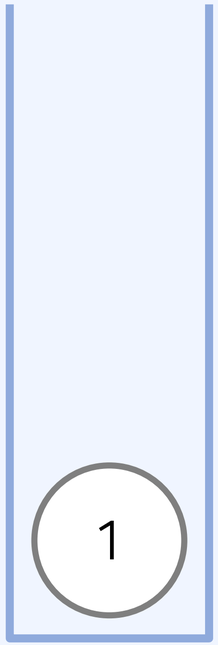

```
// 무방향 그래프에서 사이클 여부 확인
function isCycle(x, prev) {
  // 현재 노드 방문 처리
  visited[x] = true;
  // 다음 노드(인접 노드)를 하나씩 확인하며
  for (let y of graph[x]) {
    if (!visited[y]) { // 다음 노드를 아직 방문하지 않았다면
      // 다음 노드 기준으로 사이클이라면
      if (isCycle(y, x)) return true; // 사이클 발생
    }
    // 방문한 적 있는 노드인데, 직전 노드가 아니라면(무방향 그래프)
    else if (y != prev) return true;
  }
  return false;
}
```

트리: https://www.acmicpc.net/problem/4803

문제 해결 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클이다.
- 단, 무방향 그래프이므로 직전 노드는 제외한다.



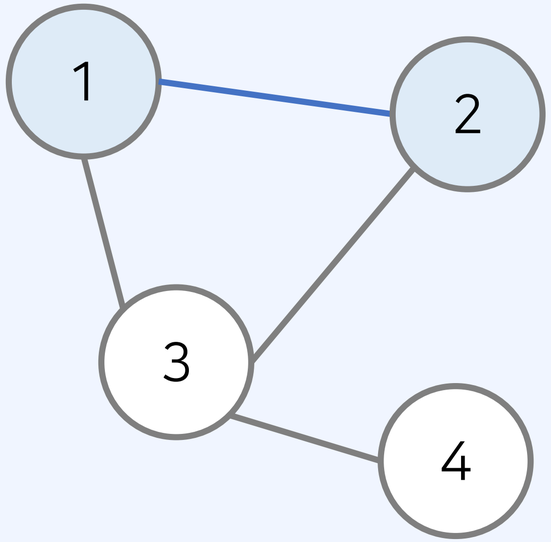


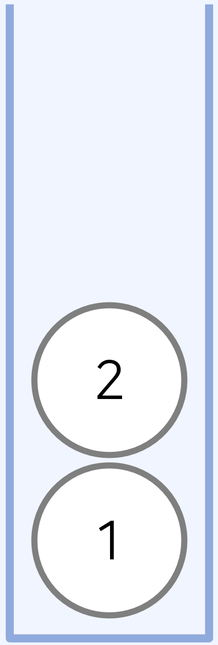

```
// 무방향 그래프에서 사이클 여부 확인
function isCycle(x, prev) {
  // 현재 노드 방문 처리
  visited[x] = true;
  // 다음 노드(인접 노드)를 하나씩 확인하며
  for (let y of graph[x]) {
    if (!visited[y]) { // 다음 노드를 아직 방문하지 않았다면
      // 다음 노드 기준으로 사이클이라면
      if (isCycle(y, x)) return true; // 사이클 발생
    }
    // 방문한 적 있는 노드인데, 직전 노드가 아니라면(무방향 그래프)
    else if (y != prev) return true;
  }
  return false;
}
```

트리: https://www.acmicpc.net/problem/4803

문제 해결 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클이다.
- 단, 무방향 그래프이므로 직전 노드는 제외한다.



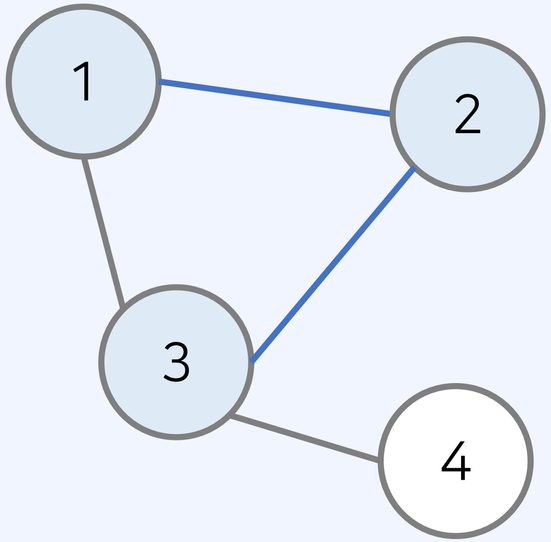


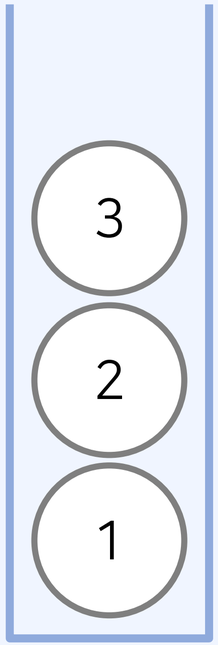

```
// 무방향 그래프에서 사이클 여부 확인
function isCycle(x, prev) {
  // 현재 노드 방문 처리
  visited[x] = true;
  // 다음 노드(인접 노드)를 하나씩 확인하며
  for (let y of graph[x]) {
    if (!visited[y]) { // 다음 노드를 아직 방문하지 않았다면
      // 다음 노드 기준으로 사이클이라면
      if (isCycle(y, x)) return true; // 사이클 발생
    }
    // 방문한 적 있는 노드인데, 직전 노드가 아니라면(무방향 그래프)
    else if (y != prev) return true;
  }
  return false;
}
```

트리: https://www.acmicpc.net/problem/4803

문제 해결 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클이다.
- 단, 무방향 그래프이므로 직전 노드는 제외한다.



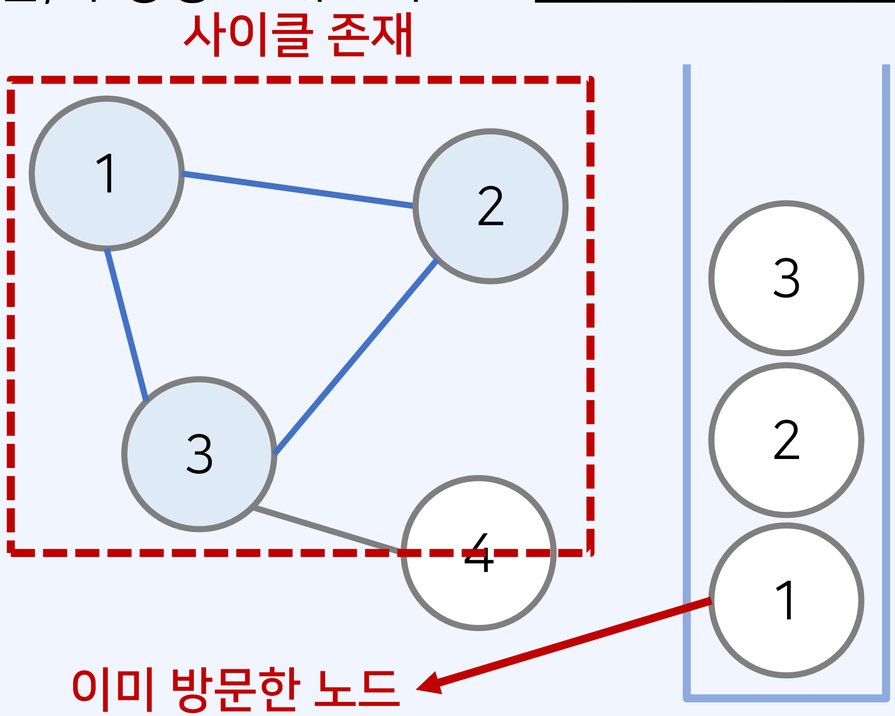



```
// 무방향 그래프에서 사이클 여부 확인
function isCycle(x, prev) {
  // 현재 노드 방문 처리
  visited[x] = true;
  // 다음 노드(인접 노드)를 하나씩 확인하며
  for (let y of graph[x]) {
    if (!visited[y]) { // 다음 노드를 아직 방문하지 않았다면
      // 다음 노드 기준으로 사이클이라면
      if (isCycle(y, x)) return true; // 사이클 발생
    }
    // 방문한 적 있는 노드인데, 직전 노드가 아니라면(무방향 그래프)
    else if (y != prev) return true;
  }
  return false;
}
```

트리: https://www.acmicpc.net/problem/4803

문제 해결 아이디어

- 무방향 그래프 내 사이클 판별 알고리즘은 다음과 같다.
- 특정 노드에서 DFS를 수행하는 과정에서 "인접 노드가 **이미 방문한 노드라면**" 사이클이다.
- 단, 무방향 그래프이므로 직전 노드는 제외한다.




```
// 무방향 그래프에서 사이클 여부 확인
function isCycle(x, prev) {
  // 현재 노드 방문 처리
  visited[x] = true;
  // 다음 노드(인접 노드)를 하나씩 확인하며
  for (let y of graph[x]) {
    if (!visited[y]) { // 다음 노드를 아직 방문하지 않았다면
      // 다음 노드 기준으로 사이클이라면
      if (isCycle(y, x)) return true; // 사이클 발생
    }
    // 방문한 적 있는 노드인데, 직전 노드가 아니라면(무방향 그래프)
    else if (y != prev) return true;
  }
  return false;
}
```

트리: https://www.acmicpc.net/problem/4803

정답 코드 예시


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

let line = 0;
let testCase = 1;
while (true) {
  let [n, m] = input[line].split(' ').map(Number);
  if (n == 0 && m == 0) break; // 노드의 개수(N)와 간선의 개수(M)
  graph = []; // 트리 정보 입력받기
  for (let i = 1; i <= n; i++) graph[i] = [];
  for (let i = 1; i <= m; i++) {
    let [x, y] = input[line + i].split(' ').map(Number);
    graph[x].push(y);
    graph[y].push(x);
  }
  visited = new Array(n + 1).fill(false); // 방문 처리 배열
  let cnt = 0; // 그래프 내 트리의 개수
  for (let i = 1; i <= n; i++) { // 하나씩 노드를 확인하며
    if (!visited[i]) { // 연결 요소이면서
      if (!isCycle(i, 0)) cnt++;// 사이클이 아니라면 트리이므로, 카운트하기
    }
  }
  if (cnt == 0) console.log(`Case ${testCase}: No trees.`);
  else if (cnt == 1) console.log(`Case ${testCase}: There is one tree.`);
  else console.log(`Case ${testCase}: A forest of ${cnt} trees.`);
  line += m + 1; // 다음 테스트 케이스로 이동
  testCase++;
}
```