JavaScript DFS 알고리즘
DFS 문제 풀이

DFS 문제 풀이 | 코딩 테스트에서 자주 등장하는 DFS 알고리즘 이해하기

강사 나동빈

JavaScript
DFS 알고리즘

DFS 문제 풀이

바이러스: https://www.acmicpc.net/problem/2606

혼자 힘으로 풀어보기

문제 제목: 바이러스

문제 난이도: ★★☆☆☆

문제 유형: 깊이 우선 탐색, 그래프 순회

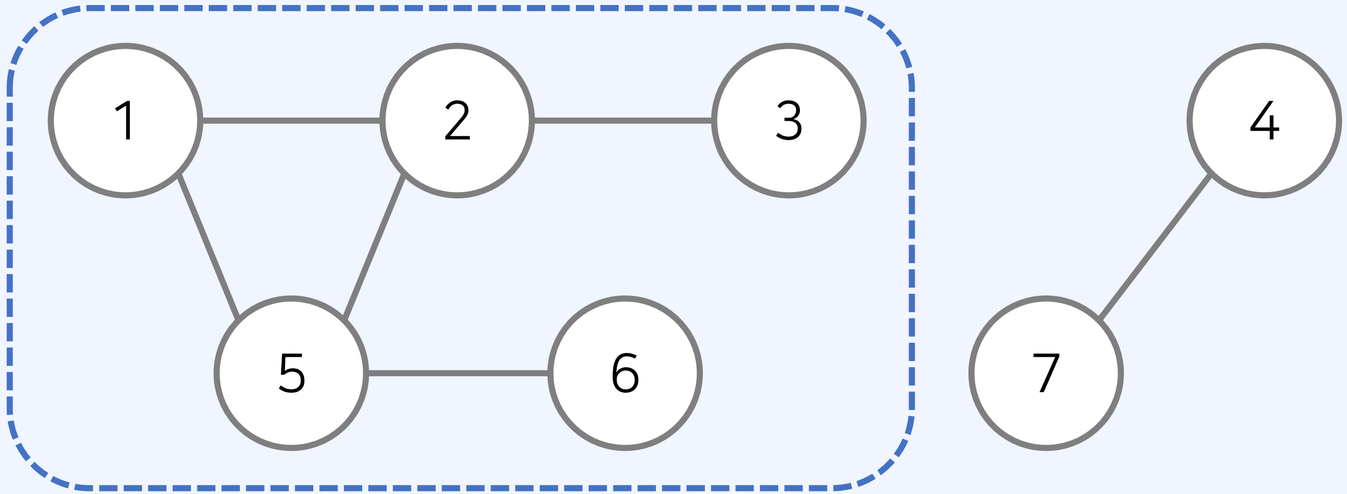
추천 풀이 시간: 40분

바이러스: https://www.acmicpc.net/problem/2606

문제 해결 아이디어

- 1번 노드에서 **도달할 수 있는** 다른 노드의 개수를 출력하는 문제다.
- DFS를 이용해 양방향 그래프에 대한 그래프 탐색을 진행할 수 있다.

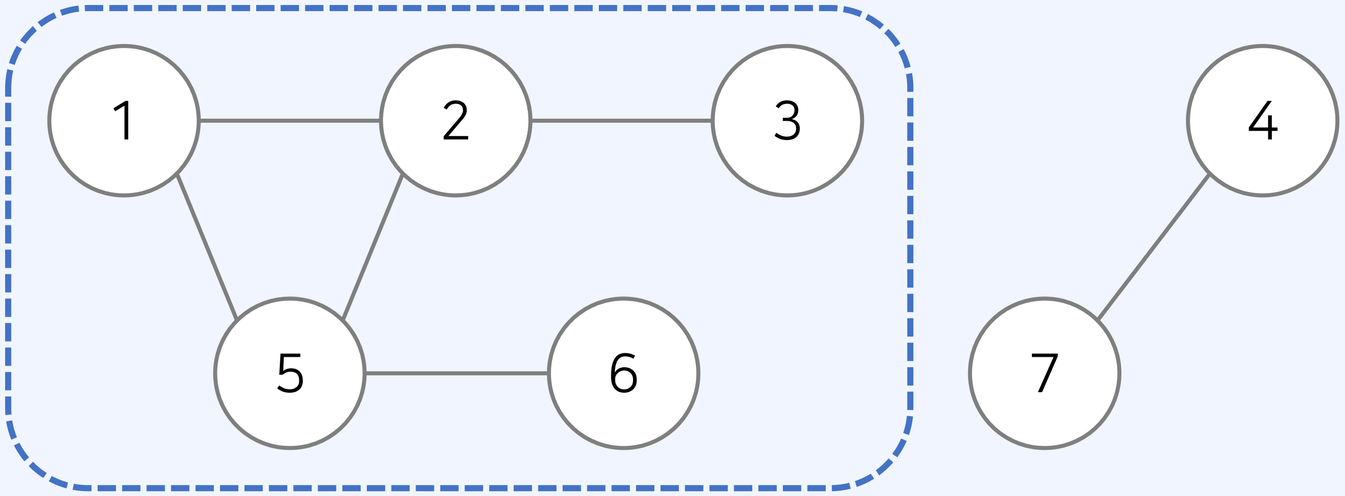
도달 가능한 다른 노드의 개수: 4개



바이러스: https://www.acmicpc.net/problem/2606

문제 해결 아이디어

- DFS를 이용해 1번 노드에서 출발해 그래프 탐색을 진행할 수 있다.
- 그래프를 인접 리스트로 표현할 때, **인덱스 0은 사용하지 않도록** 하면 직관적이다.



인접 리스트

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

바이러스: https://www.acmicpc.net/problem/2606

정답 코드 예시


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

let n = Number(input[0]); // 정점의 개수(N)
let m = Number(input[1]); // 간선의 개수(M)
let graph = []; // 그래프 정보 입력
for (let i = 1; i <= n; i++) graph[i] = [];
for (let i = 2; i <= m + 1; i++) {
  let [x, y] = input[i].split(' ').map(Number);
  graph[x].push(y);
  graph[y].push(x);
}

let cnt = 0;
let visited = new Array(n + 1).fill(false);
function dfs(x) { // 깊이 우선 탐색(DFS) 수행
  visited[x] = true; // 현재 노드를 방문 처리
  cnt++;
  for (y of graph[x]) { // 현재 노드와 연결된 다른 노드를 재귀적으로 방문
    if (!visited[y]) dfs(y);
  }
}
dfs(1);
console.log(cnt - 1);
```

유기농 배추: https://www.acmicpc.net/problem/1012

문제 해결 아이디어

- 본 문제는 연결 요소(connected component)의 개수를 계산하는 문제다.
- 각 위치에서 **[상, 하, 좌, 우]**의 위치로 간선이 연결되어 있는 그래프로 이해할 수 있다.

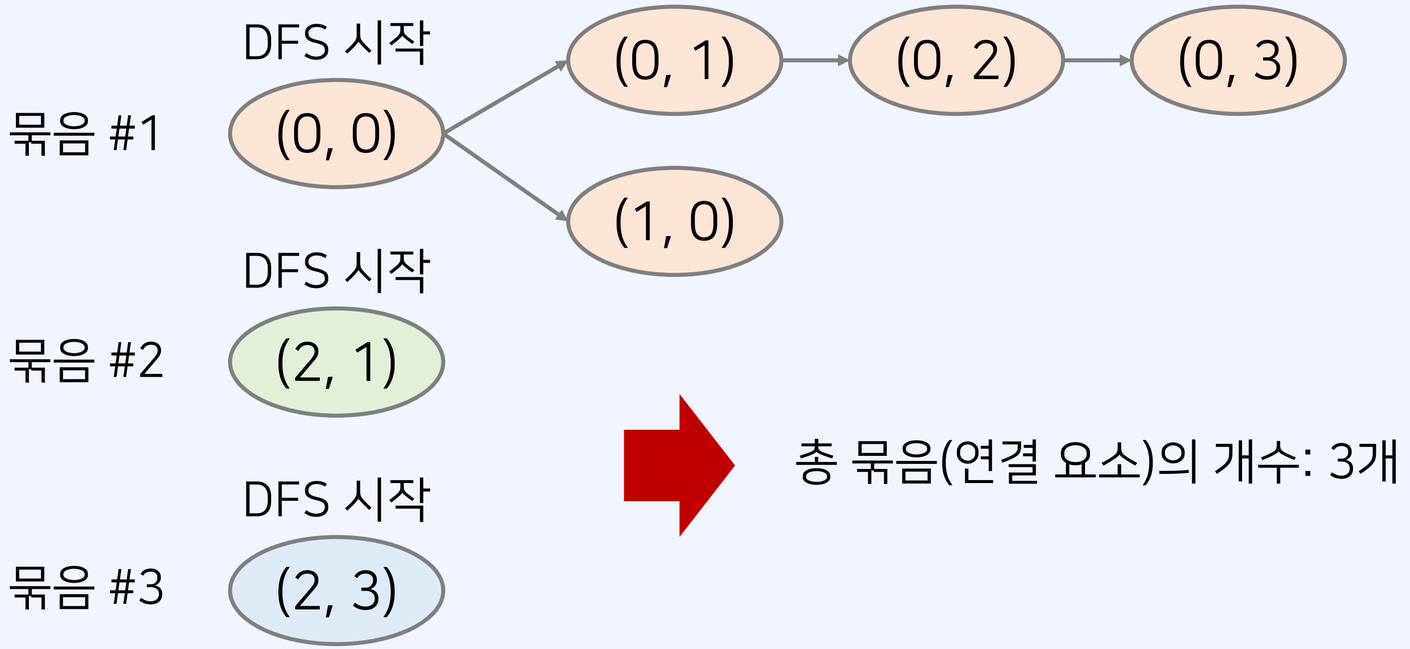
1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1
0	0	0	0	1	0	0	1	1	1

유기농 배추: https://www.acmicpc.net/problem/1012

문제 해결 아이디어

- 연결 요소는 일종의 **[묶음]**으로 이해하면 직관적이다.
- 왼쪽 위부터 오른쪽 아래까지 하나씩 확인하며, **[처리되지 않은]** 원소에 대하여 DFS를 호출한다.

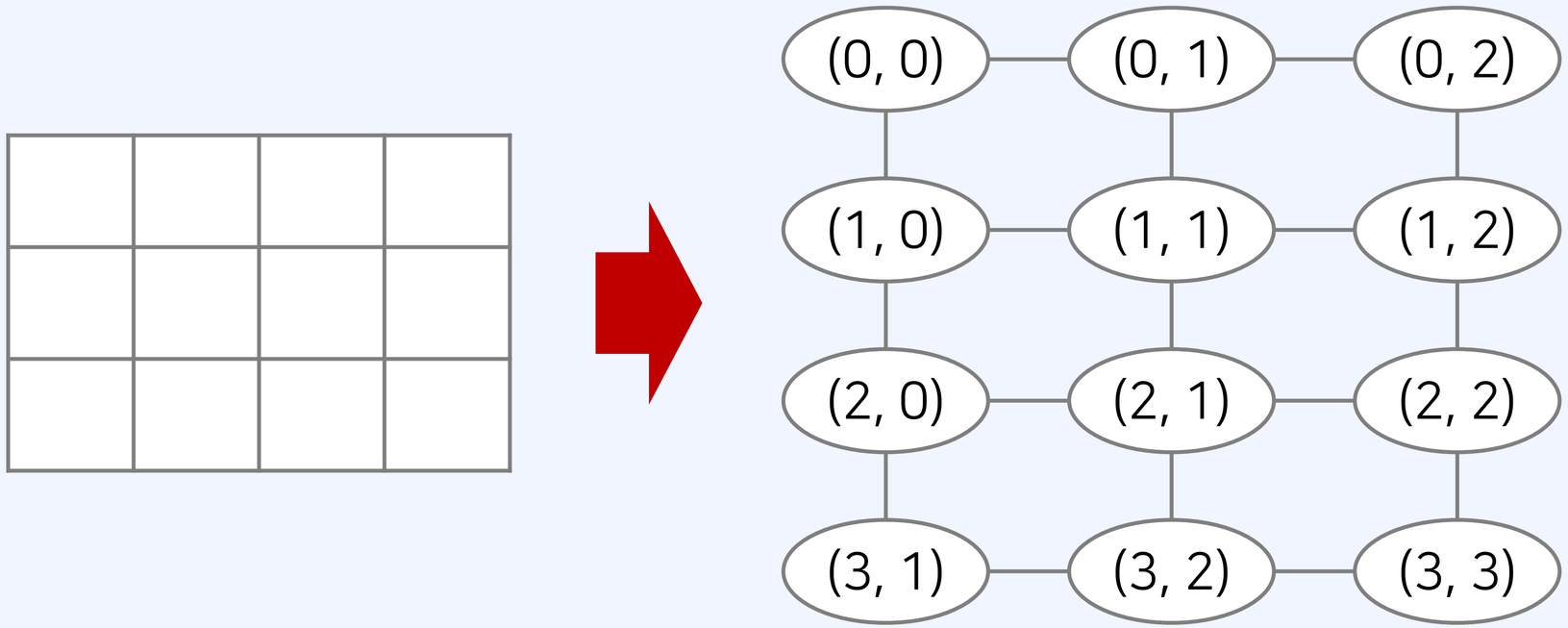
1	1	1	1
1	0	0	0
0	1	0	1



유기농 배추: https://www.acmicpc.net/problem/1012

문제 해결 아이디어

- 2차원 격자에서 **[상, 하, 좌, 우]**로 이동 가능하다면, 이는 그래프로 표현할 수 있다.
- 그래프로 표현한 뒤에 DFS를 수행하는 것으로 이해하면 간단하다.



유기농 배추: https://www.acmicpc.net/problem/1012

문제 해결 아이디어

- 왼쪽 위부터 시작해 **아직 처리되지 않은** 노드에 대하여 DFS를 수행한다.
- 아래 예시에서는 총 5번의 $dfs()$ 함수 호출이 발생한다.

1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	1	1
0	0	0	0	1	0	0	1	1	1

유기농 배추: https://www.acmicpc.net/problem/1012

정답 코드 예시


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

function dfs(graph, n, m, x, y) { // 깊이 우선 탐색(DFS) 수행
  // 주어진 범위를 벗어나는 경우에는 즉시 종료
  if (x <= -1 || x >= n || y <= -1 || y >= m)
    return false;
  // 현재 노드를 아직 처리하지 않았다면
  if (graph[x][y] == 1) {
    // 해당 노드 방문 처리
    graph[x][y] = -1;
    // 상, 하, 좌, 우의 위치들도 모두 재귀적으로 호출
    dfs(graph, n, m, x - 1, y);
    dfs(graph, n, m, x, y - 1);
    dfs(graph, n, m, x + 1, y);
    dfs(graph, n, m, x, y + 1);
    return true;
  }
  return false;
}
```

유기농 배추: https://www.acmicpc.net/problem/1012

정답 코드 예시


```
let testCases = Number(input[0]); // 테스트 케이스의 수
let line = 1;
while (testCases--) {
  // 가로 길이(M), 세로 길이(N), 배추가 심어져 있는 위치의 개수(K)
  let [m, n, k] = input[line].split(' ').map(Number);
  let graph = []; // 그래프 정보
  for (let i = 0; i < n; i++) {
    graph[i] = new Array(m);
  }
  for (let i = 1; i <= k; i++) {
    let [y, x] = input[line + i].split(' ').map(Number);
    graph[x][y] = 1;
  }
  let answer = 0; // 연결 요소(connected component)의 수 계산
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      if (dfs(graph, n, m, i, j)) answer++; // 현재 위치에서 DFS 수행
    }
  }
  line += k + 1; // 다음 테스트 케이스로 이동
  console.log(answer);
}
```