JavaScript DFS 알고리즘
DFS 문제 풀이

DFS 문제 풀이 | 코딩 테스트에서 자주 등장하는 DFS 알고리즘 이해하기

강사 나동빈

JavaScript
DFS 알고리즘

DFS 문제 풀이

치킨 배달: https://www.acmicpc.net/problem/15686

혼자 힘으로 풀어보기

문제 제목: 치킨 배달

문제 난이도: ★★★☆☆

문제 유형: 깊이 우선 탐색, 백트래킹

추천 풀이 시간: 50분

치킨 배달: https://www.acmicpc.net/problem/15686

문제 해결 아이디어

- 치킨 거리는 집과 가장 가까운 치킨집 사이의 거리이다.
- M개의 치킨 집만 남기는 모든 경우의 수 중에서 **[모든 집의 치킨 거리의 합]**의 최솟값을 계산한다.
- $N = 5, M = 3$인 경우를 고려해 보자. (정답은 5)

0	0	1	0	0
0	0	2	0	1
0	1	2	0	0
0	0	1	0	0
0	0	0	0	2

치킨 배달: https://www.acmicpc.net/problem/15686

정답 코드 예시


```
function dfs(depth, start) {
  if (depth == m) { // 각 조합을 확인하는 부분
    let result = []; // 조합(combination) 결과 저장 테이블
    for (let i of selected) result.push(chicken[i]);
    let sum = 0; // 치킨 거리의 합
    for (let [hx, hy] of house) { // 모든 집에 대하여
      let temp = 1e9; // 가장 가까운 치킨 집을 찾기
      for (let [cx, cy] of result) {
        temp = Math.min(temp, Math.abs(hx - cx) + Math.abs(hy - cy));
      }
      sum += temp; // 가장 가까운 치킨 집까지의 거리를 더하기
    }
    answer = Math.min(answer, sum); // 최솟값 계산
    return;
  }
  // start 지점부터 하나씩 원소 인덱스(index)를 확인하며
  for (let i = start; i < chicken.length; i++) {
    if (visited[i]) continue; // [중복을 허용하지 않으므로] 이미 처리 된 원소라면 무시
    selected.push(i); // 현재 원소 선택
    visited[i] = true; // 현재 원소 방문 처리
    dfs(depth + 1, i + 1); // 조합이므로, 재귀 함수 호출시 다음 인덱스(index)를 넣기
    selected.pop(); // 현재 원소 선택 취소
    visited[i] = false; // 현재 원소 방문 처리 취소
  }
}
```

치킨 배달: https://www.acmicpc.net/problem/15686

정답 코드 예시


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

let [n, m] = input[0].split(' ').map(Number); // 도시의 크기(N)와 치킨집의 개수(M)
let chicken = []; // 전체 치킨집 위치 배열
let house = []; // 전체 집 위치 배열
for (let i = 1; i <= n; i++) { // 전체 도시 정보 입력받기
  let line = input[i].split(' ').map(Number); // 빈 칸(0), 집(1), 치킨집(2)
  for (let j = 0; j < n; j++) {
    if (line[j] == 1) house.push([i, j]); // 집(1)
    if (line[j] == 2) chicken.push([i, j]); // 치킨집(2)
  }
}

let visited = new Array(chicken.length).fill(false); // 각 치킨집 방문 여부
let selected = []; // 현재 조합에 포함된 원소(element)

let answer = 1e9;
dfs(0, 0);
console.log(answer);
```

단지번호붙이기: https://www.acmicpc.net/problem/2667

혼자 힘으로 풀어보기

문제 제목: 단지번호붙이기

문제 난이도: ★★☆☆☆

문제 유형: 깊이 우선 탐색, 연결 요소(Connected Component)

추천 풀이 시간: 50분

단지번호붙이기: https://www.acmicpc.net/problem/2667

문제 해결 아이디어

- 맵의 크기는 $N \times N$ 크기의 **정사각형** 형태이다.
- N은 최대 25이므로, DFS 혹은 BFS를 이용할 수 있다.

단지번호붙이기: https://www.acmicpc.net/problem/2667

문제 해결 아이디어

- **연결 요소**란, 모든 두 꼭짓점 사이에 경로가 존재하는 그래프를 의미한다.
- 본 문제는 이러한 **연결 요소(connected component)**를 찾는 문제다.
- 구체적으로 각 연결 요소에 포함된 원소의 개수를 계산하는 것이 요구된다.

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

단지번호붙이기: https://www.acmicpc.net/problem/2667

문제 해결 아이디어

- **연결 요소**의 개수를 세는 방법은 다음과 같다.

1. 방문하지 않은 노드를 만날 때마다 카운트(count)하고, **DFS를 호출**한다.
2. DFS는 **해당 위치로부터 연결된(연결 요소에 포함된)** 모든 노드를 방문 처리한다.

단지번호붙이기: https://www.acmicpc.net/problem/2667

문제 해결 아이디어

1. 방문하지 않은 노드를 만날 때마다 카운트(count)하고, **DFS를 호출**한다.
2. DFS는 **해당 위치로부터 연결된(연결요소에 포함된)** 모든 노드를 방문 처리한다.

0	1	1	0	2	0	0
0	1	1	0	2	0	2
1	1	1	0	2	0	2
0	0	0	0	2	2	2
0	3	0	0	0	0	0
0	3	3	3	3	3	0
0	3	3	3	0	0	0

○ : DFS가 시작되는 위치

단지번호붙이기: https://www.acmicpc.net/problem/2667

정답 코드 예시


```
function dfs(x, y) {
  if (x <= -1 || x >= n || y <= -1 || y >= n) return 0;
  if (graph[x][y] >= 1) { // 아직 방문하지 않았다면
    graph[x][y] = -1;
    let result = 1;
    result += dfs(x - 1, y); // 4가지 방향 호출
    result += dfs(x, y - 1);
    result += dfs(x + 1, y);
    result += dfs(x, y + 1);
    return result;
  }
  return 0;
}
```

단지번호붙이기: https://www.acmicpc.net/problem/2667

정답 코드 예시


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

let n = Number(input[0]); // 지도의 크기(N)
let graph = []; // 그래프 정보 입력
for (let i = 1; i <= n; i++) {
  graph.push(input[i].split('').map(Number));
}

let answer = []; // 단지의 수 = 연결 요소(connected component)의 수 계산
for (let i = 0; i < n; i++) {
  for (let j = 0; j < n; j++) {
    let current = dfs(i, j); // 현재 위치에서 DFS 수행
    if (current > 0) answer.push(current); // 단지가 존재하는 경우
  }
}

answer.sort((a, b) => a - b); // 단지의 수와 오름차순 정렬된 각 단지내 집의 수 출력
console.log(answer.length + '\n' + answer.join('\n'));
```