JavaScript DFS 알고리즘
DFS 문제 풀이

DFS 문제 풀이 | 코딩 테스트에서 자주 등장하는 DFS 알고리즘 이해하기

강사 나동빈

JavaScript
DFS 알고리즘

DFS 문제 풀이

적록색약: https://www.acmicpc.net/problem/10026

혼자 힘으로 풀어보기

문제 제목: 적록색약

문제 난이도: ★★★☆☆

문제 유형: 깊이 우선 탐색, 그래프 순회

추천 풀이 시간: 50분

적록색약: https://www.acmicpc.net/problem/10026

문제 해결 아이디어

- 맵의 크기는 $N \times N$ 크기의 **정사각형** 형태다.
- N은 최대 100이므로, DFS 혹은 BFS를 이용할 수 있다.

연결 요소 4개

R	R	R	B	B
G	G	B	B	B
B	B	B	R	R
B	B	R	R	R
R	R	R	R	R

연결 요소 3개

G	G	G	B	B
G	G	B	B	B
B	B	B	G	G
B	B	G	G	G
G	G	G	G	G

적록색약: https://www.acmicpc.net/problem/10026

문제 해결 아이디어

- 연결 요소(connected component)의 개수를 세는 문제다.
- 초기에 연결 요소의 개수를 계산하고, 빨간색을 초록색으로 변경한 상태에서 다시 한 번 계산한다.

연결 요소 4개

R	R	R	B	B
G	G	B	B	B
B	B	B	R	R
B	B	R	R	R
R	R	R	R	R

연결 요소 3개

G	G	G	B	B
G	G	B	B	B
B	B	B	G	G
B	B	G	G	G
G	G	G	G	G

적록색약: https://www.acmicpc.net/problem/10026

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(연결요소에 포함된)** 모든 노드를 방문 처리한다.

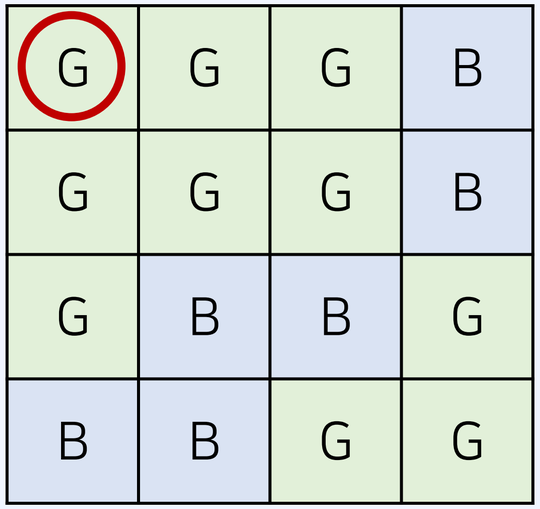
적록색약: https://www.acmicpc.net/problem/10026

문제 해결 아이디어

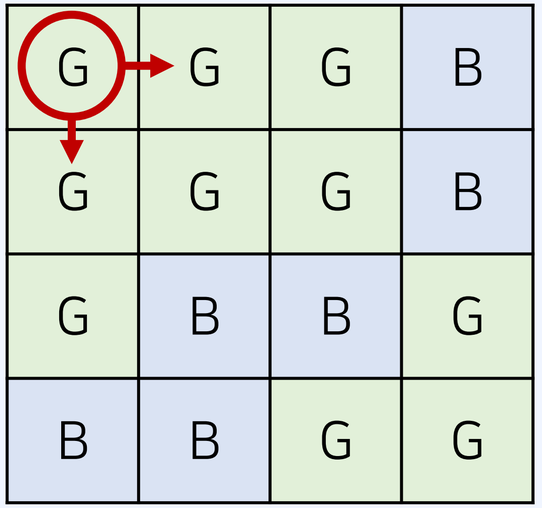
- **연결 요소**의 개수를 세는 방법은 다음과 같다.

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

1. 방문하지 않은 노드를 만나면 **DFS 호출** 시작



2. 인접한 노드에 대해서도 재귀적으로 방문 처리



적록색약: https://www.acmicpc.net/problem/10026

정답 코드 예시


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

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

// 상, 하, 좌, 우
let dx = [-1, 1, 0, 0];
let dy = [0, 0, -1, 1];

function dfs(x, y) {
  if (!visited[x][y]) { // 방문하지 않았다면
    visited[x][y] = true; // 방문 처리
    for (let i = 0; i < 4; i++) { // 인접한 영역을 하나씩 확인
      let nx = x + dx[i];
      let ny = y + dy[i];
      if (nx < 0 || ny < 0 || nx >= n || ny >= n) continue; // 공간을 벗어나는 경우 무시
      if (graph[x][y] == graph[nx][ny]) dfs(nx, ny); // 같은 색상이라면 재귀적으로 dfs() 호출
    }
    return true;
  }
  return false;
}
```

적록색약: https://www.acmicpc.net/problem/10026

정답 코드 예시


```
// DFS를 이용하여 연결 요소 세기
let result1 = 0;
let visited = [];
for (let i = 0; i < n; i++) visited.push(new Array(n).fill(false));
for (let i = 0; i < n; i++)
  for (let j = 0; j < n; j++)
    if (dfs(i, j, 0)) result1++;

// R → G 변환 이후에 다시 한 번 연결 요소 세기
for (let i = 0; i < n; i++)
  for (let j = 0; j < n; j++)
    if (graph[i][j] == 'R') graph[i][j] = 'G';

// DFS를 이용하여 연결 요소 세기
let result2 = 0;
visited = [];
for (let i = 0; i < n; i++) visited.push(new Array(n).fill(false));
for (let i = 0; i < n; i++)
  for (let j = 0; j < n; j++)
    if (dfs(i, j)) result2++;

console.log(result1 + ' ' + result2);
```

연구소: https://www.acmicpc.net/problem/14502

혼자 힘으로 풀어보기

문제 제목: 연구소

문제 난이도: ★★★☆☆

문제 유형: 깊이 우선 탐색, 그래프 순회, 트리

추천 풀이 시간: 50분

연구소: https://www.acmicpc.net/problem/14502

문제 해결 아이디어

- 벽을 3개 설치하는 모든 경우(조합)의 수를 고려해야 한다.
- 맵의 크기가 최대 8 X 8이므로, 벽을 설치할 수 있는 모든 조합의 수는 최악의 경우 $64C3$ 정도이다.
- 이는 100,000보다도 작은 수이므로, 모든 경우를 고려해도 제한 시간을 초과하지 않는다.
- 문제 해결 과정은 다음과 같다.
- ① DFS를 이용해 모든 조합의 수를 계산하기
- ② 각 조합마다 DFS를 이용해 안전 영역의 크기를 계산하기

연구소: https://www.acmicpc.net/problem/14502

문제 해결 아이디어

[핵심] 벽을 3개 설치하는 모든 경우(조합)의 수를 고려해야 한다.

입력 예시

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

예시

벽 설치하기

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

연구소: https://www.acmicpc.net/problem/14502

문제 해결 아이디어

- 벽을 설치하는 각 조합에 대하여 안전 영역의 크기를 계산한다.
- 벽을 설치한 뒤에는 DFS를 수행해 바이러스가 퍼지도록 한다.
- 이후에 '0'으로 표시된 위치의 수를 계산한다. (아래 예시에서 안전 영역의 크기는 27)

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

연구소: https://www.acmicpc.net/problem/14502

정답 코드 예시


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

let [n, m] = input[0].split(' ').map(Number);
let data = []; // 초기 맵 리스트
let temp = []; // 벽을 설치한 뒤의 맵 리스트
for (let i = 1; i <= n; i++) {
  let line = input[i].split(' ').map(Number);
  data.push(line);
  temp.push(new Array(m).fill(0));
}
let dx = [-1, 0, 1, 0]; // 4가지 이동 방향에 대한 리스트
let dy = [0, 1, 0, -1];
let result = 0;

// 깊이 우선 탐색(DFS)을 이용해 각 바이러스가 사방으로 퍼지도록 하기
function virus(x, y) {
  for (let i = 0; i < 4; i++) {
    let nx = x + dx[i];
    let ny = y + dy[i];
    if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue; // 맵을 벗어나는 경우 무시
    if (temp[nx][ny] == 0) {
      temp[nx][ny] = 2; // 해당 위치에 바이러스 배치하고, 다시 재귀적으로 수행
      virus(nx, ny);
    }
  }
}
```

연구소: https://www.acmicpc.net/problem/14502

정답 코드 예시


```
function getScore() { // 현재 맵에서 안전 영역의 크기 계산하는 메서드
  let score = 0;
  for (let i = 0; i < n; i++)
    for (let j = 0; j < m; j++)
      if (temp[i][j] == 0) score += 1;
  return score;
}

// 깊이 우선 탐색(DFS)을 이용해 울타리를 설치하면서, 매 번 안전 영역의 크기 계산
function dfs(count) {
  if (count == 3) { // 울타리가 3개 설치된 경우
    for (let i = 0; i < n; i++)
      for (let j = 0; j < m; j++)
        temp[i][j] = data[i][j]; // 임시 배열에 데이터 기록
    for (let i = 0; i < n; i++)
      for (let j = 0; j < m; j++)
        if (temp[i][j] == 2) virus(i, j); // 각 바이러스의 위치에서 전파 진행
    result = Math.max(result, getScore()); // 안전 영역의 최대값 계산
    return;
  }
  for (let i = 0; i < n; i++) // 빈 공간에 울타리를 설치
    for (let j = 0; j < m; j++)
      if (data[i][j] == 0) { // 울타리를 3개 설치하는 모든 조합 계산
        data[i][j] = 1;
        dfs(count + 1);
        data[i][j] = 0;
      }
}
dfs(0);
console.log(result);
```