0x0A DFS

상태	완료
담당자	㉮ 서연

DFS란?

💡 다차원 배열에서 각 칸 방문 시 깊이를 우선으로 방문하는 알고리즘임.

핵심원리

> 스택 top을 빼면서 인접한 좌표 탐색하며 Stack에 넣는 작업을 함.

DFS 코드 흐름 정리 — java ver

1. 전체 보드(=그래프) 선언
2. 그래프 2차원 배열과 같은 크기의 방문 여부를 저장할 boolean type의 배열 선언.
3. 현재 배열의 행의 수, 열의 수 저장함
4. 4방향을 탐색하기 위해서 dr, dc 1차원 배열 선언 (여기서 dr=y, dc=x와 대응)
5. 인접한 노드를 담고 탐색을 진행할 스택 선언
6. 처음 위치에서 시작하므로 0, 0좌표에 대해 방문 표시해주고 스택에 푸시

---

7. 변수에 현재 stack top 값을 담고 pop함. > c++과 다르게 java는 pop만으로 원소를 담음
8. dr 배열 길이만큼 돌면서 상하좌우 값 탐색
   8-1. 현재 좌표 위치에서 dr, dc 값을 더함
   8-2. 배열의 크기를 벗어나게 된다면 해당 좌표 탐색을 멈춤
   8-3. 이미 방문을 해서 true이거나, board의 값이 1이 아니면 탐색을 멈춤

8-4. 그 외에는 방문을 하는 경우이므로 인접 좌표 방문처리
8-5. 해당 좌표를 stack에 push함.

자바 코드로 써보면...?

```
import java.util.Arrays;
import java.util.Map;
import java.util.Stack;

public class dfs구현 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // 그래프 선언

        int[][] graph = {
                { 1, 1, 1, 0, 1, 0, 0, 0, 0, 0 },
                { 1, 0, 0, 0, 1, 0, 0, 0, 0, 0 },
                { 1, 1, 1, 0, 1, 0, 0, 0, 0, 0 },
                { 1, 1, 0, 0, 1, 0, 0, 0, 0, 0 },
                { 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

        // 방문 여부 저장할 배열 선언
        boolean[][] visited= new boolean[7][10];

        // 그래프의 행의 수 열의 수 할당
        int n= 7;
        int m=10;

        // 사방향 탐색을 위한 일차원 배열 선언
        int[] dr = { -1, 0, 0, 1 };
        int[] dc = { 0, -1, 1, 0 };

        // 인접한 노드를 담고 탐색을 진행할 좌표쌍을 담을 스택 선언
```

```
        Stack<int[]> stack= new Stack<>();

        // 처음 0,0에서부터 탐색 진행
        int nr=0; int nc=0;
        visited[nr][nc]=true;
        stack.push(new int[] {nr, nc});
//      System.out.println(Arrays.toString(stack.pop()));

        // 스택이 비게되면 탐색 종료
        while(!stack.empty()) {
            // 좌표를 담을 배열을 선언하고 pop함.
            int[] curr= stack.pop();
            System.out.println(Arrays.toString(curr)+"→");

            // 상하좌우 탐색
            for(int i=0; i<dr.length; i++) {
                // 현재 탐색하는 노드에 dr, dc를 더해줌
                nr= curr[0]+dr[i];
                nc= curr[1]+dc[i];

                // 배열 범위 초과하는지 확인
                if(nr<0 || nr >=n || nc<0 || nc>=m) {
                    continue;
                }

                // 이미 방문을 했거나 그래프가 인접한 노드가 없는지 확인
                if(visited[nr][nc] || graph[nr][nc]!=1) {
                    continue;
                }

                // 위 두가지 경우를 제외하면 방문 가능
                visited[nr][nc]= true;
                stack.push(new int[] {nr, nc});

//              System.out.println(nr+" "+nc);
            }

        }
```

```
    }
}
```

BFS vs DFS

사용 자료구조가 다르다! → 이로 인해 방문 순서에 직접적인 차이가 발생!

BFS는 현재 같은 거리에 있는 칸을 전부 확인하고 다음 거리에 있는 칸을 탐색하기 시작함.

DFS는 막힐 때까지 최대한 깊게 파고들면서 탐색함.

1. **BFS에서는 성립한 거리가 1만큼 더 떨어진다는 성질이 dfs에서는 성립하지 않는 이유**
    a. DFS는 깊이를 우선하기 때문에 방문 순서가 시작점으로부터의 거리와 일치하지 않게됨.