3기 / 3주차_구현

🌟 구현(시뮬레이션)

머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정이다.

내가 생각할 땐 모든 문제가 크게 보면 구현 문제에 속하는 것 같다.

빡 노가다 문제가 정말 많은 유형이기 때문에 백준 기준으로 보통 골드 이상의 난이도를 가지고 있으며 정답률 50%를 넘기 힘들다.

1. 백준 2573_빙산

🔑 풀이

로직 자체는 어렵지 않았지만 반례를 못 찾아서 거의 4시간을 붙잡고 있었다 … ^_^

덩어리 갯수 세고 빙산 녹이고(=1년이 지난다)를 반복하다가 덩어리가 2개 이상 될 때 몇 년이 지났는지를 구하면 된다.

아래는 내가 실수한 부분이다.

```
if (map[x][y] - countZero <= 0) {
        map[x][y] = -1;
} else {
    map[x][y] -= countZero;
}
```

같은 사이클 안에서 다 녹은 칸이 0으로 바뀌면 원래 0이었던 곳과 구분이 가지 않아서 -1로 바꿔주었다.

근데 다른 곳에서 얼음이 남아 있는지 판단할 때 0보다 큰 지를 확인한 게 아니라 0이 아닌지만 확인을 했기 때문에 -1인 부분까지 카운트가 됐다.

💻 코드

536 ms, 108528 KB

```
package week3;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
import java.util.Queue;
import java.util.StringTokenizer;

public class Baekjoon_2573 {
    private static int row, column;
    private static int[][] map;
    private static final int[] dx = {0, 1, 0, -1};
    public static final int[] dy = {1, 0, -1, 0};
    private static int answer = 0;

    public static void main(String[] args) throws IOException
        input();
        solve();
        output();
    }

    private static void input() throws IOException {
        BufferedReader br = new BufferedReader(new InputStrea
        StringTokenizer st = new StringTokenizer(br.readLine(
        row = Integer.parseInt(st.nextToken());
        column = Integer.parseInt(st.nextToken());

        map = new int[row][column];
        for (int i = 0; i < row; i++) {
            st = new StringTokenizer(br.readLine());
            for (int k = 0; k < column; k++) {
                map[i][k] = Integer.parseInt(st.nextToken());
            }
        }
    }

    private static void output() {
```

```
        System.out.println(answer);
    }

    private static void solve() {
        while (true) {
            int bundle = countBundle();

            if (bundle >= 2) {
                break;
            }

            if (bundle == 0) {
                answer = 0;
                break;
            }

            meltIce();
            answer++;
        }
    }

    private static int countBundle() {
        boolean[][] isVisited = new boolean[row][column];

        int bundle = 0;
        for (int i = 0; i < row; i++) {
            for (int k = 0; k < column; k++) {
                if (map[i][k] < 0) map[i][k] = 0;
                if (map[i][k] > 0 && !isVisited[i][k]) {
                    dfs(i, k, isVisited);
                    bundle++;
                }
            }
        }

        return bundle;
    }
```

```
    private static void dfs(int x, int y, boolean[][] isVisit
        isVisited[x][y] = true;

        for (int i = 0; i < 4; i++) {
            int nextX = x + dx[i];
            int nextY = y + dy[i];

            if (checkRange(nextX, nextY)) {
                if (!isVisited[nextX][nextY] && map[nextX][ne
                    dfs(nextX, nextY, isVisited);
                }
            }
        }
    }

    private static void meltIce() {
        Queue<int[]> queue = new LinkedList<>();

        for (int i = 0; i < row; i++) {
            for (int k = 0; k < column; k++) {
                if (map[i][k] > 0) {
                    queue.offer(new int[]{i, k});
                }
            }
        }

        while (!queue.isEmpty()) {
            int[] start = queue.poll();
            int x = start[0];
            int y = start[1];
            int countZero = 0;

            for (int i = 0; i < 4; i++) {
                int nextX = x + dx[i];
                int nextY = y + dy[i];

                if (checkRange(nextX, nextY)) {
                    if (map[nextX][nextY] == 0) countZero++;
```

```
                }
            }

            if (map[x][y] - countZero <= 0) {
                map[x][y] = -1;
            } else {
                map[x][y] -= countZero;
            }
        }
    }

    private static boolean checkRange(int x, int y) {
        return x >= 0 && y >= 0 && x < row && y < column;
    }
}
    public static void main(String[] args) throws IOException
        input();
        System.out.println(dfs(0, 0));
    }

    private static void input() throws IOException {
        BufferedReader br = new BufferedReader(new InputStrea
        StringTokenizer st = new StringTokenizer(br.readLine(
        row = Integer.parseInt(st.nextToken());
        column = Integer.parseInt(st.nextToken());
        map = new int[row][column];
        dp = new int[row][column];

        for (int i = 0; i < row; i++) {
            st = new StringTokenizer(br.readLine());
            for (int k = 0; k < column; k++) {
                map[i][k] = Integer.parseInt(st.nextToken());
                dp[i][k] = -1;
            }
        }
    }

    private static int dfs(int x, int y) {
```

```
        if (x == row - 1 && y == column - 1) {
            return 1;
        }

        if (dp[x][y] == -1) {
            dp[x][y] = 0;
            for (int i = 0; i < 4; i++) {
                int nextX = x + dx[i];
                int nextY = y + dy[i];

                if (nextX < 0 || nextY < 0 || nextX >= row ||

                if (map[x][y] > map[nextX][nextY]) {
                    dp[x][y] += dfs(nextX, nextY);
                }
            }
        }

        return dp[x][y];
    }
}
```