2468 - 안전 영역

난이도	실버 1

문제

문제

재난방재청에서는 많은 비가 내리는 장마철에 대비해서 다음과 같은 일을 계획하고 있다. 먼저 어떤 지역의 높이 정보를 파악한다. 그 다음에 그 지역에 많은 비가 내렸을 때 물에 잠기지 않는 안전한 영역이 최대로 몇 개가 만들어 지는 지를 조사하려고 한다. 이때, 문제를 간단하게 하기 위하여, 장마철에 내리는 비의 양에 따라 일정한 높이 이하의 모든 지점은 물에 잠긴다고 가정한다.

어떤 지역의 높이 정보는 행과 열의 크기가 각각 N인 2차원 배열 형태로 주어지며 배열의 각 원소는 해당 지점의 높이를 표시하는 자연수이다. 예를 들어, 다음은 N=5인 지역의 높이 정보이다.

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

이제 위와 같은 지역에 많은 비가 내려서 높이가 4 이하인 모든 지점이 물에 잠겼다고 하자. 이 경우에 물에 잠기는 지점을 회색으로 표시하면 다음과 같다.

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

물에 잠기지 않는 안전한 영역이라 함은 물에 잠기지 않는 지점들이 위, 아래, 오른쪽 혹은 왼쪽으로 인접해 있으며 그 크기가 최대인 영역을 말한다. 위의 경우에서 물에 잠기지 않는 안전한 영역은 5개가 된다(꼭짓점으로만 붙어 있는 두 지점은 인접하지 않는다고 취급한다).

또한 위와 같은 지역에서 높이가 6이하인 지점을 모두 잠기게 만드는 많은 비가 내리면 물에 잠기지 않는 안전한 영역은 아래 그림에서와 같이 네 개가 됨을 확인할 수 있다.

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

이와 같이 장마철에 내리는 비의 양에 따라서 물에 잠기지 않는 안전한 영역의 개수는 다르게 된다. 위의 예와 같은 지역에서 내리는 비의 양에 따른 모든 경우를 다 조사해 보면 물에 잠기지 않는 안전한 영역의 개수 중에서 최대인 경우는 5임을 알 수 있다.

어떤 지역의 높이 정보가 주어졌을 때, 장마철에 물에 잠기지 않는 안전한 영역의 최대 개수를 계산하는 프로그램을 작성하시오.

풀이과정

이 문제는 경로의 **특징을 저장**해야한다고 생각해서 dfs로 풀어봤다. (경로의 특징 저장이 맞는지는...모른다.)

처음에는 비의 양과 지역의 높이를 매번 비교해주며 영역을 카운트하려 했으나, 코드가 너무 복잡해져서 **특정 비의 양에 잠기지 않는 지역만 1로 표시해주는 배열**을 새로 만들었다.

코드는 다음과 같다.

```
package BOJ2468;

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

public class Main {

    static int[][] area;    // 주어진 지역 높이
    static int[][] dryArea;   // 잠기지 않는 지역 체크
    static boolean[][] visited;
    static int N, maxHeight, count, max;
    static int[] dx = { -1, 1, 0, 0 }; //x방향배열-상하
    static int[] dy = { 0, 0, -1, 1 }; //y방향배열-좌우
    public static void main(String[] args) throws IOException {
```

```
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st;
        N = Integer.parseInt(br.readLine());
        area = new int[N][N];

        maxHeight = 0;
        for(int i=0 ; i<N ; i++){
            st = new StringTokenizer(br.readLine());
            for(int j=0 ; j<N ; j++) {
                area[i][j] = Integer.parseInt(st.nextToken());
                if(maxHeight < area[i][j]) maxHeight = area[i][j];
            }
        }

        max = 1;
        for(int i=1 ; i<=maxHeight ; i++){
            count = 0;
            dryArea = new int[N][N];
            visited = new boolean[N][N];

            // 안 잠기는 지역 확인
            for(int j=0 ; j<N ; j++) {
                for(int k=0 ; k<N ; k++) {
                    if (area[j][k] > i) dryArea[j][k] = 1;
                }
            }

            for(int x=0 ; x<N ; x++){
                for(int y=0 ; y<N ; y++){
                    if(dryArea[x][y] == 1 && !visited[x][y]){
                        dfs(x,y);
                        count++;
                    }
                }
            }

            if(max < count) max = count;

        }

        System.out.println(max);
    }

    static void dfs(int x, int y) {
        visited[x][y] = true;
        int nextX, nextY;
        for(int i=0 ; i<4 ; i++){
            nextX = x + dx[i];
            nextY = y + dy[i];

            if(nextX < 0 || nextY < 0 || nextX >= N || nextY >= N)
                continue;
            if(visited[nextX][nextY] || dryArea[nextX][nextY] == 0)
                continue;

            dfs(nextX, nextY);
        }
```

```
    }
}
```

3중포문이 썩 맘에 들진 않았지만 이외의 방법이 생각나지 않아 일단 했다...ㅎㅎ

지역의 높이에 대한 배열을 먼저 입력받았다.

그리고 비의 양을 1부터 지역의 가장 높은 높이까지로 정해가면서 잠기지 않는 지역에 대한 정보를 담은 배열을 만들어주었다.

영역을 카운트하는 조건

- 방문하지 않은 곳
- 잠기지 않는 곳

이 두 가지 조건을 만족하면 해당 노드에서부터 dfs를 돌려 영역을 하나 세어주기로 했다.

이렇게 각 비의 양에 따른 잠기지 않는 영역의 개수를 세어준 뒤, 그 중 최대값을 출력하도록 하였다.

백준 2178번(최단 거리 탐색)이랑 2468번(영역 세어주기)이랑 코드 내용은 DFS와 BFS를 제외하고는 조금 비슷했던 것 같다.