농장 관리 풀이

	1	2	3	4	5	6	7
1	4	3	2	2	1	0	1
2	3	3	3	2	1	0	1
3	2	2	2	2	1	0	0
4	2	1	1	1	1	0	0
5	1	1	0	0	0	1	0
6	0	0	0	1	1	1	0
7	0	1	2	2	1	1	0
8	0	1	1	1	2	1	0

- N행 M열의 배열로 표현할 수 있는 그래프를 격자 그래프(Grid Graph)라고 합니다.
- 명시적으로 인접 리스트나 인접 행렬로 그래프를 구성하지 않고 암시적으로 그래프를 사용할 수 있습니다.
- *산봉우리*를 찾기 위해 그래프에서 간선을 정의합니다.

    - 격자에서 r번째 행 c번째 열을 표현하는 정점을 (r, c)라고 할 때, (r, c)는 8개의 정점 $(r-1, c-1)$, $(r-1, c)$, $(r-1, c+1)$, $(r, c-1)$, $(r, c+1)$, $(r+1, c-1)$, $(r+1, c)$, $(r+1, c+1)$ 중에서 높이가 같은 정점과 간선으로 연결되어 있습니다.
- 이와 같이 그래프를 정의하면 산봉우리를 찾는 문제를 그래프에서 연결 요소(Components)를 찾는 문제로 바꿀 수 있습니다.
- 단, 컴포넌트와 인접한 정점이 더 높은 높이를 가지고 있는지 확인해야 합니다. 기존의 격자 그래프 G에서 컴포넌트를 하나의 정점으로 모델링한 그래프 G_D를 가정해봅시다. G_D에서 모든 정점 u와 인접한 정점 v에 대해 두 정점의 높이를 비교하는 것으로 이를 구현할 수 있습니다.

소스 코드

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    const int dy[8] = { -1, 1, 0, 0, -1, -1, 1, 1 };
    const int dx[8] = { 0, 0, -1, 1, -1, 1, -1, 1 };

    int n, m;
    cin >> n >> m;

    vector<vector<int>> a(n, vector<int>(m));
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < m; j++) {
            cin >> a[i][j];
        }
    }

    auto in_range = [&](int y, int x) {
        return 0 <= y && y < n && 0 <= x && x < m;
        };

    vector<vector<int>> b(n, vector<int>(m));

    // 정점 (sy, sx)가 속한 컴포넌트의 정점을 모두 방문
    auto bfs = [&](int sy, int sx) {
        queue<pair<int, int>> q;
        q.emplace(sy, sx);
        b[sy][sx] = true;

        bool flag = true;
        while (!q.empty()) {
            auto [y, x] = q.front();
            q.pop();

            for (int d = 0; d < 8; d++) {
                int ny = y + dy[d], nx = x + dx[d];
                // 8방향으로 인접한 칸 중에서 높이가 같은 칸으로 간선이 이어져 있다.
```

```
                if (in_range(ny, nx) && a[ny][nx] == a[sy][sx] && !b[ny][nx]) {
                    q.emplace(ny, nx);
                    b[ny][nx] = true;
                }
            }

            for (int d = 0; d < 8; d++) {
                int ny = y + dy[d], nx = x + dx[d];
                // 8방향으로 인접한 칸 중에서 높이가 더 높은 칸이 있는지 확인.
                if (in_range(ny, nx) && a[ny][nx] > a[y][x]) {
                    flag = false;
                }
            }
        }

        return flag;
        };

    int c = 0;
    for (int y = 0; y < n; y++) {
        for (int x = 0; x < m; x++) {
            if (!b[y][x]) {
                c += bfs(y, x);
            }
        }
```

```
        }

        cout << c << '\n';
    }
```