백준 9879번 - Cross Country Skiing

백준 9879번 - Cross Country Skiing

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1초	512MB	171	85	74	49.333%

문제

The cross-country skiing course at the winter Moolympics is described by an M x N grid of elevations (1 <= M,N <= 500), each elevation being in the range 0 .. 1,000,000,000.

Some of the cells in this grid are designated as waypoints for the course. The organizers of the Moolympics want to assign a difficulty rating D to the entire course so that a cow can reach any waypoint from any other waypoint by repeatedly skiing from a cell to an adjacent cell with absolute elevation difference at most D. Two cells are adjacent if one is directly north, south, east, or west of the other. The difficulty rating of the course is the minimum value of D such that all waypoints are mutually reachable in this fashion.

입력

- Line 1: The integers M and N.
- Lines 2..1+M: Each of these M lines contains N integer elevations.
- Lines 2+M..1+2M: Each of these M lines contains N values that are either 0 or 1, with 1 indicating a cell that is a waypoint.

출력

- Line 1: The difficulty rating for the course (the minimum value of D such that all waypoints are still reachable from each-other).

예제 입력1

```
3 5
20 21 18 99 5
19 22 20 16 26
18 17 40 60 80
1 0 0 0 1
0 0 0 0 0
0 0 0 0 1
```

예제 출력1

```
21
```

출처

출처

알고리즘 분류

- 자료 구조
- 그래프 이론
- 이분 탐색
- 분리 집합
- 최소 스패닝 트리

접근 방법

문제의 조건에 따라, 모든 waypoints가 상호연결되게 만들어야한다.

문제의 최적해는 반드시 존재하며, waypoints를 모두 연결하는 spanning tree중 구성되는 edge의 max cost가 최솟값이되는 D를 찾아야한다.

먼저, 위 조건을 만족하는 Spanning Tree를 만들기 위해 총 연결되는 vertex의 개수를 알아야한다. waypoint들이 상호연결 되도록 할 때, waypoint가 아닌 중간 노드의 개수는 런타임 시간 이외에는 알 수가 없다. 즉, spanning tree를 직접적으로 구하기는 어렵다.

간접적으로 spanning tree를 구할 수 있는 방법이 있다. 모든 waypoint들(K개)이 상호 연결된 그래프에서 E 개의 edge 중 $E-K+1$ 개의 edge를 제거하면 된다.

그러면 이제 모든 waypoint들이 상호 연결된 그래프를 만들어보자. 문제에서 주어진 최적해를 구하기 위해, 상호 연결된 그래프 중 edge cost의 max가 최솟값이 되도록 구성해야한다.

완전 탐색하기에는 주어진 시간 내에 문제를 해결 할 수 없으므로, 결정 문제로 치환하여 풀어보자.

- $f(k)$:= waypoint를 상호 연결하는 그래프에서 모든 edge가 k이하가 되도록 구성할 수 있는가?

결정 함수 f는 단조함수의 형태를 가지기 때문에 이분 탐색으로 문제를 해결할 수 있다.

$f(k)$ 를 구현해보자.

waypoint를 상호 연결하는 그래프 중 하나라도 $f(k)=true$ 를 만족하는 그래프가 존재하면 k는 정답 후보이다.

그렇다면, k 이하의 모든 edge들을 선택하여 연결해준 그래프(`A`라고 칭하자)를 만든다면 $f(k)=true$ 를 만족하는 그래프가 존재한다면 `A`그래프의 부분집합이다. (`A`그래프에서 $E-K+1$ 개의 edge를 제거하여 만들 수 있음)

더 나은 방법

결정 문제로 치환하지 않고 최적해를 구할 수 있는 방법을 찾았다.

이전의 접근 방법에서 `Spanning Tree를 만들기 위해 총 연결되는 vertex의 개수를 알아야한다.` 라고 기술하였는데, 관점을 바꾸어 생각해보자. $M * N$ 배열에서 각각의 cell을 (i, j) 상태를 가지는 그래프로 표현하였을 때, 최적해는 MST의 부분 집합이다. (즉, MST에서 일부 edge를 제거하므로서 최적해를 구할 수 있다)

MST를 구성하는 과정에서 C = `현재까지 방문한 waypoint의 개수`를 기록한다.

그렇다면, MST를 구성하는 과정 중 C가 waypoint의 전체 개수가 되었을 때 해당 그래프가 최적해라는 것을 보장한다.

`union(u, v)` 연산에서, `u`라는 부분 그래프가 방문한 waypoint의 개수와 `v`라는 부분 그래프가 방문한 waypoint의 개수를 더해주는 과정에서 C 가 `waypoints.size()` 가 된다면 현재까지 구성한 그래프가 `waypoint를 상호연결하는 그래프`의 최적해가 되는 것이고 이것을 출력하면 된다.

(단순하게 `union(u,v)` 에서 중복 waypoint를 고려하지 않아도 되는 이유는 이미 counting된 waypoint는 서로소 집합의 특성에 따라서 cycle이 발생하지 않게 하기 위해 union연산에서 제외되기 때문이다.)

소스코드

```cpp
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
////////////////////////////////////////////////////////////////
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

int M, N, h[501][501];
vector<pair<int,int>> waypoints;
const int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
bool canMakeSpanningTree(const int D) {
    // D이하인 모든 Edge를 이어서 만든 그래프에서 일부 edge를 제거하는 것으로 spanning tree 완성 가능
    // D이하의 모든 Edge가 waypoints를 모두 경유하는가?
    vector<vector<bool>> visited(M, vector<bool>(N, false));
    queue<pair<int,int>> Q;
    Q.push({waypoints[0].first, waypoints[0].second});
    visited[waypoints[0].first][waypoints[0].second] = true;

    while (!Q.empty()) {
        auto [y, x] = Q.front();
        Q.pop();
        for (int d = 0; d < 4; d++) {
            int ny = y + dir[d][0], nx = x + dir[d][1];
            if (ny < 0 || nx < 0 || ny >= M || nx >= N || visited[ny][nx] || abs(h[ny][nx] - h[y][x]) > D)
                continue;

            visited[ny][nx] = true;
            Q.push({ny, nx});
        }
```

```
    }

    for (auto [y, x] : waypoints) {
        if (!visited[y][x])
            return false;
    }
    return true;
}
int main(void){
    FASTIO;
////////////////////////////////////////////////////////////////////

    cin >> M >> N;
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            cin >> h[i][j];
        }
    }

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            int temp;
            cin >> temp;
            if (temp == 1)
                waypoints.emplace_back(i, j);
        }
    }

    ll left = 0L, right = 1e9, ans;
    while (left <= right) {
        ll mid = (left + right) / 2;
        if (canMakeSpanningTree(mid)) {
            ans = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    cout << ans;

    return 0;
}

```