백준 1520번 - 내리막 길

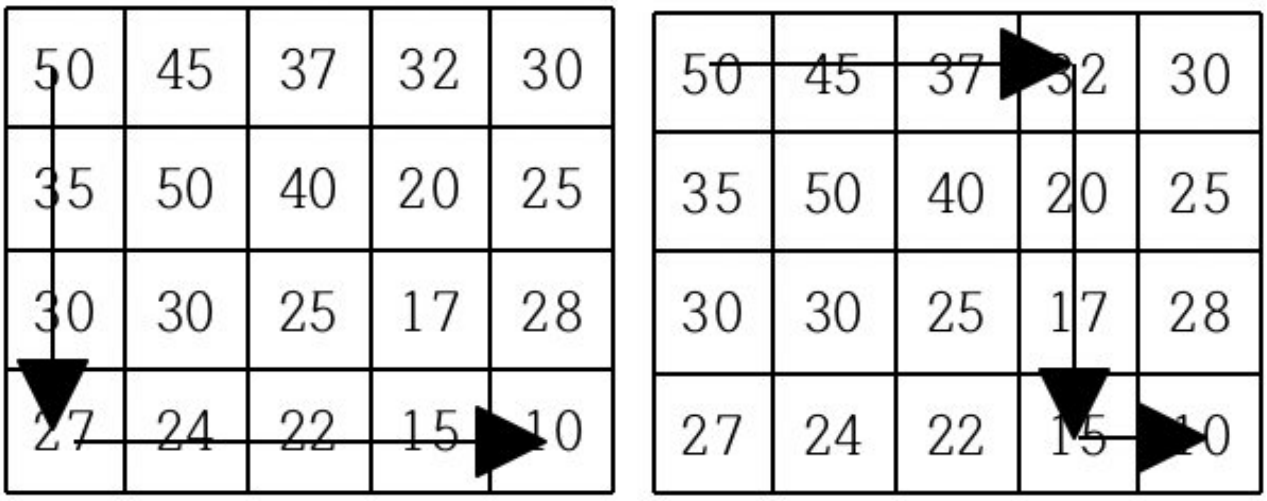
백준 1520번 - 내리막 길

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2초	128MB	56499	15516	11041	27.909%

문제

여행을 떠난 세준이는 지도를 하나 구하였다. 이 지도는 아래 그림과 같이 직사각형 모양이며 여러 칸으로 나뉘어져 있다. 한 칸은 한 지점을 나타내는데 각 칸에는 그 지점의 높이가 쓰여 있으며, 각 지점 사이의 이동은 지도에서 상하좌우 이웃한 곳끼리만 가능하다.

현재 제일 왼쪽 위 칸이 나타내는 지점에 있는 세준이는 제일 오른쪽 아래 칸이 나타내는 지점으로 가려고 한다. 그런데 가능한 힘을 적게 들이고 싶어 항상 높이가 더 낮은 지점으로만 이동하여 목표 지점까지 가고자 한다. 위와 같은 지도에서는 다음과 같은 세 가지 경로가 가능하다.



지도가 주어질 때 이와 같이 제일 왼쪽 위 지점에서 출발하여 제일 오른쪽 아래 지점까지 항상 내리막길로만 이동하는 경로의 개수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에는 지도의 세로의 크기 M과 가로의 크기 N이 빈칸을 사이에 두고 주어진다. 이어 다음 M개 줄에 걸쳐 한 줄에 N개씩 위에서부터 차례로 각 지점의 높이가 빈 칸을 사이에 두고 주어진다. M과 N은 각각 500이하의 자연수이고, 각 지점의 높이는 10000이하의 자연수이다.

출력

첫째 줄에 이동 가능한 경로의 수 H를 출력한다. 모든 입력에 대하여 H는 10억 이하의 음이 아닌 정수이다.

예제 입력1

```
4 5
50 45 37 32 30
35 50 40 20 25
30 30 25 17 28
27 24 22 15 10
```

예제 출력1

```
3
```

출처

출처

알고리즘 분류

- 다이나믹 프로그래밍
- 그래프 이론
- 그래프 탐색
- 깊이 우선 탐색

접근 방법

각각의 좌표를 (r, c)라는 pair 정보를 가진 정점으로 취급할 수 있다.

문제에서 주어진 조건을 잘 살펴보면 아래와 같은 사실을 도출할 수 있다.

1. `항상 내리막길로만 이동한다` 라는 조건이 존재하므로, a $\rightarrow$ b $\rightarrow$ a 로 가는 경우는 존재하지 않는다. 즉, 사이클이 발생하지 않는다.
    - 한번 방문한 좌표는 다시 방문할 수 없으므로 하나의 정점은 1개의 정점과 대응된다.
2. 가장 Naive한 백트래킹으로 접근할 때, a $\rightarrow$ b $\rightarrow$ c $\rightarrow$ $(M-1, N-1)$ 까지 `내리막길로 이동할 수 있는 모든 경우의 수` 를 구한 경우, u $\rightarrow$ (a ....)로 진행하는 경우의 수에 대해서는 이전의 탐색(a...)의 결과를 Retrieve하면 된다.

위 조건을 통해서 쉽게 문제를 해결할 수 있다.

$dp[i][j]$:= (i, j) 좌표에서 $(M-1, N-1)$ 좌표까지 내리막길로 이동할 수 있는 모든 경우의 수

로 가정해보자.

경우의 수는 항상 0이상의 정수이므로, 초기화 되지 않은 상태를 -1로 표현한다.

만약, 해당 좌표로 이전에 탐색을 진행할 경우, $dp[i][j]$는 -1이 아닌 정수로 표현되므로 그 경우는 이전 탐색의 결과를 retrieve해주자.

탐색의 끝에 도달할 경우 유의미한 탐색에 대해서는 1가지의 경우의 수로 상정하므로 이 경우는 1을 리턴한다.

DFS방식으로 탐색을 진행할 경우, Recursive하게 모든 경우의 수를 구할 수 있다. (Stack-frame의 동작 방식을 잘 생각해 보자)

소스코드

```cpp
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
////////////////////////////////////////////////////////////////
#include <bits/stdc++.h>
using namespace std;
int M, N, board[501][501], dp[501][501];
const int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
int dfs(int r, int c) {
    int &ret = dp[r][c];
    if (ret != -1) {
        return ret;
    }
    if (r == M-1 && c == N - 1)
        return 1;
    ret = 0;
    for (int d = 0; d < 4; d++) {
        int nr = r + dir[d][0], nc = c + dir[d][1];
        if (nr < 0 || nc < 0 || nr >= M || nc >= N || board[r][c] <= board[nr][nc])
            continue;
        ret += dfs(nr, nc);
    }
    return ret;
}

int main(void){
    FASTIO;
////////////////////////////////////////////////////////////////

    // dfs(i, j) := (i, j)부터 (M-1, N-1) 까지 도달할 수 있는 모든 경우의 수
    cin >> M >> N;
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            cin >> board[i][j];
        }
    }

    memset(dp, -1, sizeof(dp));
    cout << dfs(0, 0);

    return 0;
```

```
40 }
```