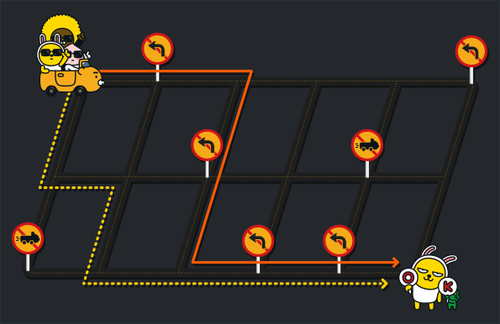
**보행자 천국**

카카오내비 개발자인 제이지는 시내 중심가의 경로 탐색 알고리즘 개발 업무를 담당하고 있다. 최근 들어 보행자가 자유롭고 편리하게 걸을 수 있도록 보행자 중심의 교통 체계가 도입되면서 도심의 일부 구역은 자동차 통행이 금지되고, 일부 교차로에서는 보행자 안전을 위해 좌회전이나 우회전이 금지되기도 했다. 복잡해진 도로 환경으로 인해 기존의 경로 탐색 알고리즘을 보완해야 할 필요가 생겼다.

도시 중심가의 지도는 m × n 크기의 격자 모양 배열 city\_map으로 주어진다. 자동차는 오른쪽 또는 아래 방향으로 한 칸씩 이동 가능하다.

city\_map[i][j]에는 도로의 상황을 나타내는 값이 저장되어 있다.

* 0인 경우에는 자동차가 자유롭게 지나갈 수 있다.
* 1인 경우에는 자동차 통행이 금지되어 지나갈 수 없다.
* 2인 경우는 보행자 안전을 위해 좌회전이나 우회전이 금지된다. (왼쪽에서 오던 차는 오른쪽으로만, 위에서 오던 차는 아래쪽으로만 진행 가능하다)



도시의 도로 상태가 입력으로 주어졌을 때, 왼쪽 위의 출발점에서 오른쪽 아래 도착점까지 자동차로 이동 가능한 전체 가능한 경로 수를 출력하는 프로그램을 작성하라. 이때 가능한 경로의 수는 컴퓨터가 표현할 수 있는 정수의 범위를 넘어설 수 있으므로, 가능한 경로 수를 20170805로 나눈 나머지 값을 출력하라.

**입력 형식**

입력은 도시의 크기를 나타내는 m과 n, 그리고 지도를 나타내는 2차원 배열 city\_map으로 주어진다. 제한조건은 아래와 같다.

* 1 <= m, n <= 500
* city\_map의 크기는 m × n이다.
* 배열의 모든 원소의 값은 0, 1, 2 중 하나이다.
* 출발점의 좌표는 (0, 0), 도착점의 좌표는 (m - 1, n - 1)이다.
* 출발점과 도착점의 city\_map[i][j] 값은 0이다.

**출력 형식**

출발점에서 도착점까지 이동 가능한 전체 경로의 수를 20170805로 나눈 나머지를 리턴한다.

**예제 입출력**

| **m** | **n** | **city\_map** | **answer** |
| --- | --- | --- | --- |
| 3 | 3 | [[0, 0, 0], [0, 0, 0], [0, 0, 0]] | 6 |
| 3 | 6 | [[0, 2, 0, 0, 0, 2], [0, 0, 2, 0, 1, 0], [1, 0, 0, 2, 2, 0]] | 2 |

**예제에 대한 설명**

첫 번째 예제는 모든 도로가 제한 없이 통행 가능한 경우로, 가능한 경우의 수는 6가지이다.  
두 번째 예제는 문제 설명에 있는 그림의 경우이다. 가능한 경우의 수는 빨간 실선과 노란 점선 2가지뿐이다.

**import** **java.util.LinkedList**;

**import** **java.util.Queue**;

**class** **Solution** {

**int** MOD = **20170805**;

**int** visit**[][]** = **new** **int**[**500**][**500**];

**public** **int** solution(**int** m, **int** n, **int[][]** cityMap) {

**int** answer = **0**;

visit[**0**][**0**] = **1**;

**int** dx**[]** = {**0**, -**1**};

**int** dy**[]** = {-**1**, **0**};

**for** (**int** i = **0**; i < m; i++) {

**for** (**int** j = **0**; j < n; j++) {

**if** (cityMap[i][j] != **1**) {

**for** (**int** k = **0**; k < **2**; k++) {

**int** nx = i + dx[k];

**int** ny = j + dy[k];

**if** (nx >= **0** && ny >= **0**) {

**while** ((nx >= **0** && ny >= **0**) && cityMap[nx][ny] == **2**) {

nx = nx + dx[k];

ny = ny + dy[k];

}

**if** (nx >= **0** && ny >= **0**) {

visit[i][j] = visit[i][j] + visit[nx][ny];

**if** (visit[i][j] >= MOD) visit[i][j] %= MOD;

}

}

}

}

}

}

***// for (int i = 0; i < m; i++) {***

***// for (int j = 0; j < n; j++) {***

***// System.out.print(visit[i][j] + " ");***

***// }***

***// System.out.println();***

***// }***

answer = visit[m - **1**][n - **1**];

**return** answer;

}

}