2178 - 미로 탐색

난이도	실버 1

문제

문제

N×M크기의 배열로 표현되는 미로가 있다.

1	0	1	1	1	1
1	0	1	0	1	0
1	0	1	0	1	1
1	1	1	0	1	1

미로에서 1은 이동할 수 있는 칸을 나타내고, 0은 이동할 수 없는 칸을 나타낸다. 이러한 미로가 주어졌을 때, (1, 1)에서 출발하여 (N, M)의 위치로 이동할 때 지나야 하는 최소의 칸 수를 구하는 프로그램을 작성하시오. 한 칸에서 다른 칸으로 이동할 때, 서로 인접한 칸으로만 이동할 수 있다.

위의 예에서는 15칸을 지나야 (N, M)의 위치로 이동할 수 있다. 칸을 셀 때에는 시작 위치와 도착 위치도 포함한다.

입력

첫째 줄에 두 정수 N, M(2 ≤ N, M ≤ 100)이 주어진다. 다음 N개의 줄에는 M개의 정수로 미로가 주어진다. 각각의 수들은 **붙어서** 입력으로 주어진다.

출력

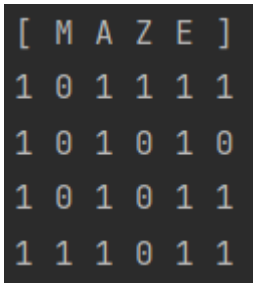
첫째 줄에 지나야 하는 최소의 칸 수를 출력한다. 항상 도착위치로 이동할 수 있는 경우만 입력으로 주어진다.

풀이과정

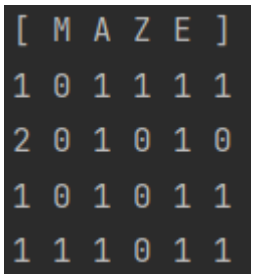
최단거리를 구하는 문제였으므로, BFS로 풀이했다. 알맞은 경로를 찾아가며 지나간 칸수는 누적해서 더해주었고, 마지막 칸에 도착했을 때 지나온 칸수가 나오도록 하였다.

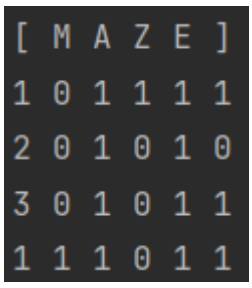
🎈 예시

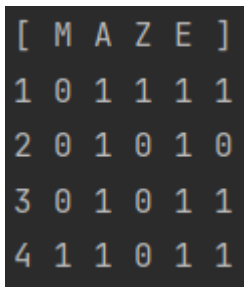
1. 초기상태

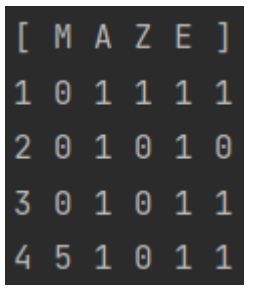


2. 진행과정

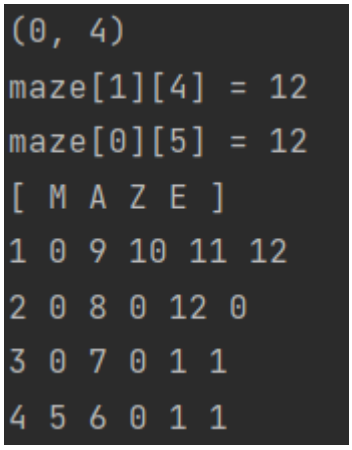




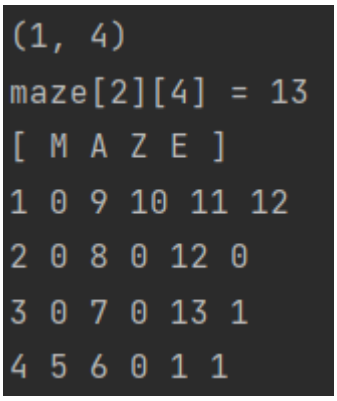




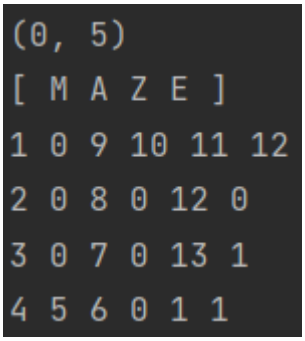
위와 같이, 지나온 칸수를 현재 칸까지 누적해서 더해주며 진행하였다. 만약 현재 칸에서 갈 수 있는 **선택지가 없는 경우, 그냥 스킵할 수 있도록** 하였다.



현재 위치 (0,4)에서 갈 수 있는 칸은 (1,4)와 (0,5)이다.



(1,4)에서는 (2,4)로 진행할 수 있다.



(0,5)에서는 진행할 수 있는 칸이 없으므로 그냥 넘어간다.

전체 코드를 보자.

```
package BOJ2178;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
```

```
import java.util.Queue;
import java.util.StringTokenizer;

public class Main {

    static int[][] maze;
    static boolean[][] visited;
    static int N, M;
    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));
        String str = br.readLine();
        StringTokenizer st = new StringTokenizer(str);
        N = Integer.parseInt(st.nextToken());
        M = Integer.parseInt(st.nextToken());

        maze = new int[N][M];
        for(int i=0 ; i<N ; i++){
            for(int j=0 ; j<M ; j++)
                maze[i][j] = br.read() - '0';
            br.readLine();
        }

        visited = new boolean[N][M];
        visited[0][0] = true;
        bfs(0,0);
        System.out.println(maze[N-1][M-1]);

    }

    static void bfs(int x, int y) {
        Queue<int[]> q = new LinkedList<>();

        q.add(new int[] {x,y});

        int curX, curY;
        int[] node;
        while(!q.isEmpty()){
            node = q.poll();
            curX = node[0];
            curY = node[1];
            for(int i=0; i<4; i++) {
                int nextX = curX + dx[i];
                int nextY = curY + dy[i];

                // 미로 밖으로 벗어날 경우 skip
                if (nextX < 0 || nextY < 0 || nextX >= N || nextY >= M)
                    continue;

                // 이미 방문했거나 방문할 수 없는 곳일 경우 skip
                if (visited[nextX][nextY] || maze[nextX][nextY] == 0)
                    continue;

                q.add(new int[] {nextX, nextY});
                maze[nextX][nextY] = maze[curX][curY] + 1;
                visited[nextX][nextY] = true;
```

```
            }

        }

    }
}
```

주로 봐야할 부분은 bfs 함수다.

먼저 인접한 노드들을 저장할 큐를 만들어주고, 첫 시작점을 큐에 넣어주었다.

그 이후 상하좌우로 인접한 노드들을 차례로 살펴보면서,

- 해당 노드에 대한 인덱스가 미로를 벗어나지 않는지
- 해당 노드 방문할 수 있는지

이 두 가지를 검사했을 때 모두 만족하는 경우 해당 노드로 진행한 뒤, 해당 노드의 인접한 노드들을 살펴보면서 위 과정을 반복한다.