말이 되고픈 원숭이(문제 _ 1600)

동물원에서 막 탈출한 원숭이 한 마리가 세상구경을 하고 있다. *그 녀석은 말(Horse)이 되기를 간절히 원했다*. 그래서 그는 말의 움직임을 유심히 살펴보고 그대로 따라 하기로 하였다. 말은 말이다.

말은 격자판에서 체스의 나이트와 같은 이동방식을 가진다. 다음 그림에 말의 이동방법이 나타나있다.

x표시한 곳으로 말이 갈 수 있다는 뜻이다. 참고로 말은 장애물을 뛰어넘을 수 있다.

	x		x	
x				x
		말		
x				x
	x		x	

근데 원숭이는 한 가지 착각하고 있는 것이 있다. 말은 저렇게 움직일 수 있지만 원숭이는 능력이 부족해서 **총 K번만 위와 같이 움직일 수 있고**, 그 외에는 그냥 인접한 칸으로만 움직일 수 있다. 대각선 방향은 인접한 칸에 포함되지 않는다.

이제 원숭이는 머나먼 여행길을 떠난다. 격자판의 **맨 왼쪽 위에서 시작해서 맨 오른쪽 아래**까지 가야한다. 인접한 네 방향으로 한 번 움직이는 것, 말의 움직임으로 한 번 움직이는 것, 모두 한 번의 동작으로 친다.

격자판이 주어졌을 때, 원숭이가 최소한의 동작으로 시작지점에서 도착지점까지 갈 수 있는 방법을 알아내는 프로그램을 작성하시오.

입력과 출력

입력
첫째 줄에 정수 K가 주어진다. 둘째 줄에 격자판의 가로길이 W, 세로길이 H가 주어진다. 그 다음 H줄에 걸쳐 W개의 숫자가 주어지는데, 0은 아무것도 없는 평지, 1은 장애물을

뜻한다. 장애물이 있는 곳으로는 이동할 수 없다. 시작점과 도착점은 항상 평지이다. W와 H는 1이상 200이하의 자연수이고, K는 0이상 30이하의 정수이다.

출력
첫째 줄에 원숭이의 동작수의 최솟값을 출력한다. 시작점에서 도착점까지 갈 수 없는 경우엔 -1을 출력한다.

예제

```
1
4 4
0 0 0 0
1 0 0 0
0 0 1 0
0 1 0 0


answer : 4
```

풀이

동작수(움직이는 횟수)의 최솟값을 구해야 하기 때문에 BFS를 사용해 최단거리를 구한다.

이때 말처럼 행동할 수 있는 횟수가 존재하기 때문에 visited 배열을 3차원 배열로 생성한다.

설명 참고 - https://yabmoons.tistory.com/48

코드

```
// 말이 움직일 수 있는 인덱스
int hx[8] = { -2,-2, -1,-1, 1,1, 2,2 };
int hy[8] = { -1,1, -2,2, -2,2, -1,1 };

// 원숭이가 움직일 수 있는 인덱스
int mx[4] = { 0,1,0,-1 };
int my[4] = { 1,0,-1,0 };
```

```
int arr[201][201]; // 지도 저장 배열
bool visited[201][201][31]; // x, y, k를 사용한 횟수 k이상 못씀

int main() {
    int k;
    cin >> k;

    int w, h;
    cin >> w >> h; //h가 행, w가 열

    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            cin >> arr[i][j];
        }
    }
    // 0,0 에서 n-1,m-1로 가야한다.

    // x, y, 사용 횟수, 이동 횟수
    queue <pair<pair<int, int>, pair<int, int>>> q;
    visited[0][0][0] = true;
    q.push({ {0,0},{0,0} });

    while (!q.empty()) {
        pair<int, int > cur = q.front().first;
        int kCnt = q.front().second.first;
        int distance = q.front().second.second;

        // 탈출 조건
        if (cur.first == h - 1 && cur.second == w - 1) {
            cout << distance;
            return 0;
        }

        q.pop();


        if (kCnt < k) { // 말의 움직임처럼 움직인다
            for (int i = 0; i < 8; i++) {
                int nx = cur.first + hx[i];
                int ny = cur.second + hy[i];

                // 범위 체크
                if (nx < 0 || nx >= h || ny < 0 || ny >= w) continue;

                // 벽이 아니고, 아직 방문한 곳이 아니라면
                if (arr[nx][ny] == 0 && !visited[nx][ny][kCnt + 1]) {
```

```
                    visited[nx][ny][kCnt + 1] = true;
                    q.push({ {nx,ny},{kCnt + 1, distance + 1} });
                }
            }
        }

        // 그냥 움직인다
        for (int i = 0; i < 4; i++) {
            int nx = cur.first + mx[i];
            int ny = cur.second + my[i];

            // 범위 체크
            if (nx < 0 || nx >= h || ny < 0 || ny >= w) continue;

            if (arr[nx][ny] == 0 && !visited[nx][ny][kCnt]) {
                visited[nx][ny][kCnt] = true;
                q.push({ {nx,ny},{kCnt, distance + 1} });
            }
        }
    }

    cout << -1; // 목적지로 가지 못하는 경우
}
```