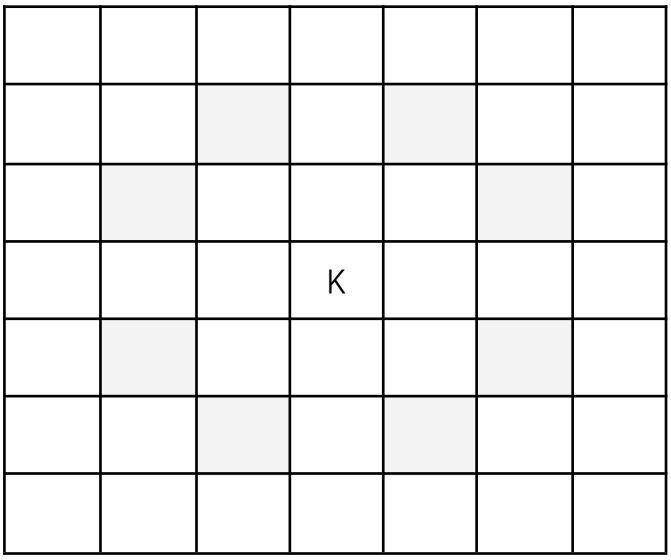
문제. 두 명의 기사

- 시간 제한: 1초
- 메모리 제한: 256MB
- 난이도: ★★★☆☆
- 문제 유형: 완전 탐색, 너비 우선 탐색(BFS)

$N \times N$ 크기의 보드 판에 주인공 1명, 기사 2명이 존재하며, 적군은 최소한 1명 이상 존재한다. 주인공의 위치는 "H", 기사의 위치는 "K", 적군의 위치는 "E"로 표시된다. 또한 빈 공간은 "B", 벽은 "X"로 표시된다. 한 위치에 여러 명이 존재하지 않는다.

주인공은 움직일 수 없으며, 기사는 원한다면 체스 말처럼 현재 위치에서 8가지 위치 중 하나로 이동할 수 있다. 기사가 $N \times N$ 격자 위치에서 이동 가능한 8가지 방향을 그림으로 표현하면 다음과 같다.

예를 들어 (행, 열)의 형태로 각 위치를 나타낸다고 가정하자. 행과 열의 인덱스가 0부터 시작한다고 하면, (3, 3)의 위치에서 이동이 가능한 위치는 (1, 2), (1, 4), (2, 1), (2, 5), (4, 1), (4, 5), (5, 2), (5, 4)이다. 아래 그림에서 음영으로 칠한 위치들과 같다.



2명의 기사는 처음 놓인 위치에 가만히 있는 것도 가능하며, 각자 빈 공간 혹은 적군이 있는 위치로 최대 1회 움직일 수 있다. 단, 벽이 있는 위치로는 이동할 수 없다. 적군이 있는 위치로 이동하게 되면, 적군은 제거되고, 그 위치를 기사가 차지한다. 특정한 기사는 최대 1회 이동을 수행하면 더 이상 움직일 수 없으며, 두 기사의 이동이 모두 끝나고 나면

적군의 이동이 시작된다. 기사 및 적군의 이동 과정에서 한 위치에 여러 명이 존재할 수 없다.

각 적군은 매 초마다 현재 위치에서 상, 하, 좌, 우로 이동할 수 있다. 기사 혹은 벽이 존재하는 위치로는 이동할 수 없으며, 주인공이 위치한 곳까지 도달하면 전체 과정이 종료된다.

예를 들어 $N = 5$인 예시를 확인해 보자. 현재 예시에서는 (1, 1) 위치에 있는 기사를 가만히 두고, (0, 1) 위치에 있는 기사를 (2, 2) 위치로 이동시켜 적군을 처리할 수 있다.

B	K	B	B	B
B	K	E	E	B
B	B	E	X	B
B	B	X	X	B
H	B	X	B	E

그 결과 다음과 같이 보드 판이 변경된다.

B	B	B	B	B
B	K	E	E	B
B	B	K	X	B
B	B	X	X	B
H	B	X	B	E

이 경우, 가장 가까운 적군이 주인공에 도달하기 위한 최소 시간은 7초이다.

결과적으로, 2명의 기사를 적절히 활용하여 **가장 가까운 적군이 주인공에 도달하기 위한 시간을 최대화**하는 것이 목표다. 목표에 맞게 2명의 기사를 적절히 활용했을 때, 가장

가까운 적군이 주인공에 도달하기 위한 최소 시간을 출력하는 프로그램을 작성하여라. 적군이 도달 불가능하게 만들 수 있다면 0을 출력한다.

입력 조건

가장 먼저 보드 판의 크기 N이 주어진다. N은 4 이상 100 이하의 자연수다.
이어서 행과 열의 길이가 각각 N인 2차원 배열 형태의 보드 판 정보 배열 $board$가 주어진다. 이때 주인공의 위치는 "H", 기사의 위치는 "K", 적군의 위치는 "E"로 표시된다. 또한 빈 공간은 "B", 벽은 "X"로 표시된다. 보드 판에는 항상 주인공 1명, 기사 2명이 존재하며, 적군은 최소한 1명 이상 존재한다.

출력 조건

목표에 맞게 2명의 기사를 적절히 활용했을 때, 가장 가까운 적군이 주인공에 도달하기 위한 최소 시간을 반환한다.

입출력 예시

N	$board$	정답
5	[["B", "K", "B", "B", "B"], ["B", "K", "E", "E", "B"], ["B", "B", "E", "X", "B"], ["B", "B", "X", "X", "B"], ["H", "B", "X", "B", "E"]]	7
5	[["E", "E", "E", "E", "E"], ["E", "E", "E", "E", "E"], ["E", "E", "E", "E", "E"], ["K", "E", "E", "E", "E"], ["H", "K", "E", "E", "E"]]	0
5	[["B", "B", "B", "B", "E"], ["B", "B", "B", "B", "B"],	8

	["K", "K", "B", "B", "B"], ["B", "B", "B", "B", "B"], ["H", "X", "E", "B", "B"]]	
5	[["B", "B", "B", "B", "B"], ["B", "B", "B", "B", "B"], ["K", "K", "B", "B", "B"], ["B", "B", "B", "B", "B"], ["H", "X", "E", "B", "B"]]	0

해설. 두 명의 기사

본 문제는 두 명의 기사를 적절히 활용하여, **가장 가까운 적군이 주인공에게 도달하기 위한 최소 시간을 최대화**하는 것이 목적이다.

한 명의 기사에 대하여, 초기의 위치에 가만히 있는 것을 포함하면 9가지 이동 경로 중에서 하나를 선택하는 것으로 이해할 수 있다. 본 문제에서는 두 명의 기사가 존재하므로, 총 9 X 9 = 81가지 경우가 존재한다.

결과적으로 총 81가지 경우에 대하여 매번 너비 우선 탐색(BFS)을 수행하여, 가장 가까운 적군이 주인공에게 도달하기 위한 최소 시간을 계산할 수 있다. 따라서 본 문제는 **너비 우선 탐색(BFS)** 및 **완전 탐색** 문제 유형에 속한다.

- **Python3 정답 코드 예시**

```
from collections import deque

dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def bfs(N, temp, enemies, hero_x, hero_y):
    graph = [[-1] * N for _ in range(N)]  # 최단 거리 맵
    q = deque()
    for (x, y) in enemies:
        if temp[x][y] == "E":  # 기사가 아닌 적군이 있는 경우
```

```
            q.append((x, y))
            graph[x][y] = 0
    while q:  # BFS 수행
        x, y = q.popleft()
        for i in range(4):
            nx = x + dx[i]
            ny = y + dy[i]
            # 공간을 벗어난 경우 무시
            if nx < 0 or ny < 0 or nx >= N or ny >= N:
                continue
            # 벽이거나 기사가 있는 경우 무시
            if temp[nx][ny] == "X" or temp[nx][ny] == "K":
                continue
            # 처음 방문하는 경우
            if graph[nx][ny] == -1:
                graph[nx][ny] = graph[x][y] + 1  # 최단 거리 기록
                q.append((nx, ny))
    return graph

# 가만히 있는 경우 및 이동 가능한 8가지 방향
knight_dx = [0, -2, -2, -1, -1, 1, 1, 2, 2]
knight_dy = [0, -1, 1, -2, 2, -2,d

# 전체 보드의 크기(N)와 각 보드 정보 배열(board)을 입력받기
def solution(N, board):
    answer = 0
    temp = [["B"] * N for _ in range(N)]  # 2차원 배열 생성
    hero_x = 0
    hero_y = 0
    enemies = []
    knights = []
    for i in range(N):
        for j in range(N):
            if board[i][j] == "H":
                hero_x = i
                hero_y = j
            elif board[i][j] == "E":
                enemies.append((i, j))
            elif board[i][j] == "K":
                knights.append((i, j))
    for k1 in range(9):  # 1번 기사의 이동 방향
        for k2 in range(9):  # 2번 기사의 이동 방향
```

```
            for x in range(N):
                for y in range(N):
                    temp[x][y] = board[x][y]
            # 기사는 항상 2명만 존재
            for i in range(2):
                kx, ky = knights[i]
                if i == 0:
                    nx = kx + knight_dx[k1]
                    ny = ky + knight_dy[k1]
                else:
                    nx = kx + knight_dx[k2]
                    ny = ky + knight_dy[k2]
                # 공간을 벗어난 경우 무시
                if nx < 0 or ny < 0 or nx >= N or ny >= N:
                    continue
                # 적군이 있거나 빈 공간일 때만 이동 가능
                if temp[nx][ny] == "E" or temp[nx][ny] == "B":
                    temp[kx][ky] = "B"
                    temp[nx][ny] = "K"
            graph = bfs(N, temp, enemies, hero_x, hero_y)
            if graph[hero_x][hero_y] != -1:  # 주인공에게 도달이 가능한 경우
                if answer < graph[hero_x][hero_y]:
                    answer = max(answer, graph[hero_x][hero_y])
            else:  # 주인공에게 도달이 불가능하도록 할 수 있다면
                return 0
    return answer
```

- **Java 정답 코드 예시**

```
import java.util.*;

class Node {
    public int x;
    public int y;

    public Node(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
class Solution {
    public static int[] dx = { -1, 1, 0, 0 };
    public static int[] dy = { 0, 0, -1, 1 };
    // 가만히 있는 경우 및 이동 가능한 8가지 방향
    public static int[] knight_dx = {0, -2, -2, -1, -1, 1, 1, 2, 2};
    public static int[] knight_dy = {0, -1, 1, -2, 2, -2, 2, -1, 1};

    public static int[][] bfs(int N, String[][] temp,
                              ArrayList<Node> enemies, int hero_x, int hero_y) {
        int[][] graph = new int[N][N]; // 최단 거리 맵
        for (int i = 0; i < N; i++)
            Arrays.fill(graph[i], -1);
        Queue<Node> q = new LinkedList<>();
        for (Node node: enemies) {
            int x = node.x;
            int y = node.y;
            if (temp[x][y].equals("E")) { // 기사가 아닌 적군이 있는 경우
                q.offer(new Node(x, y));
                graph[x][y] = 0;
            }
        }
        while (!q.isEmpty()) { // BFS 수행
            Node cur = q.poll();
            int x = cur.x;
            int y = cur.y;
            for (int i = 0; i < 4; i++) {
                int nx = x + dx[i];
                int ny = y + dy[i];
                // 공간을 벗어난 경우 무시
                if (nx < 0 || ny < 0 || nx >= N || ny >= N)
                    continue;
                // 벽인 경우 무시
                if (temp[nx][ny].equals("X") || temp[nx][ny].equals("K"))
                    continue;
                // 처음 방문하는 경우
                if (graph[nx][ny] == -1) {
                    // 최단 거리 기록
                    graph[nx][ny] = graph[x][y] + 1;
                    q.offer(new Node(nx, ny));
                }
            }
        }
```

```
    return graph;
}

// 전체 보드의 크기(N)와 각 보드 정보 배열(board)을 입력받기
public static int solution(int N, String[][] board) {
    int answer = 0;
    String[][] temp = new String[N][N]; // 2차원 배열 생성
    for (int i = 0; i < N; i++)
        Arrays.fill(temp[i], "B");
    int hero_x = 0;
    int hero_y = 0;
    ArrayList<Node> enemies = new ArrayList<Node>();
    ArrayList<Node> knights = new ArrayList<Node>();
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i][j].equals("H")) {
                hero_x = i;
                hero_y = j;
            }
            else if (board[i][j].equals("E")) {
                enemies.add(new Node(i, j));
            }
            else if (board[i][j].equals("K")) {
                knights.add(new Node(i, j));
            }
        }
    }
    for (int k1 = 0; k1 < 9; k1++) { // 1번 기사의 이동 방향
        for (int k2 = 0; k2 < 9; k2++) { // 2번 기사의 이동 방향
            for (int x = 0; x < N; x++) {
                for (int y = 0; y < N; y++) {
                    temp[x][y] = board[x][y];
                }
            }
            // 기사는 항상 2명만 존재
            for (int i = 0; i < 2; i++) {
                int kx = knights.get(i).x;
                int ky = knights.get(i).y;
                int nx = kx;
                int ny = ky;
                if (i == 0) {
                    nx += knight_dx[k1];
```

```
                    ny += knight_dy[k1];
                }
                else {
                    nx += knight_dx[k2];
                    ny += knight_dy[k2];
                }
                // 공간을 벗어난 경우 무시
                if (nx < 0 || ny < 0 || nx >= N || ny >= N) {
                    continue;
                }
                // 적군이 있거나 빈 공간일 때만 이동 가능
                if (temp[nx][ny].equals("E") || temp[nx][ny].equals("B")) {
                    temp[kx][ky] = "B";
                    temp[nx][ny] = "K";
                }
            }
            int[][] graph = bfs(N, temp, enemies, hero_x, hero_y);
            if (graph[hero_x][hero_y] != -1) { // 주인공에게 도달 가능한 경우
                if (answer < graph[hero_x][hero_y]) {
                    answer = Math.max(answer, graph[hero_x][hero_y]);
                }
            }
            else { // 주인공에게 도달 불가능하도록 할 수 있다면
                return 0;
            }
        }
    }
    return answer;
  }
}
```