문제. 봉술가

한 명의 봉술가는 $N \times N$ 크기의 보드 판에서 다수의 그래플러와 싸워야 한다. 보드 판에서 빈 공간은 "B", 그래플러의 위치는 "G"로 표시되며, 한 위치에 여러 명이 존재할 수 없다. 그래플러는 최소 2명 이상 주어지며, 초기 그래플러들은 서로 다른 칸에 위치해 있다. 그래플러가 존재하지 않는 빈 칸(빈 공간)은 1개 이상 주어진다.

봉술가가 빈 칸 중에서 자신의 시작 위치를 선택하고 나면, 이어서 그래플러들의 이동이 시작된다. 각 그래플러는 매 초마다 현재 위치에서 상, 하, 좌, 우로 이동할 수 있으며, 그래플러들의 이동 과정에서 한 위치에 여러 명이 존재할 수 없다. 봉술가는 싸움이 시작되기 전에 최대한 그래플러들부터 멀리 떨어지고 싶다.

결과적으로 봉술가가 시작 위치를 결정하기 위해, 빈 칸 중에서 **가장 가까운 그래플러와의 거리가 최대가 되는 빈 칸**을 계산하는 프로그램을 작성하여라. 가장 가까운 그래플러와의 거리가 최대가 되는 빈 칸이 여러 곳이라면, 가장 낮은(작은) 번호의 행을 갖는 위치를 반환한다. 만약 가장 낮은 번호의 행을 갖는 위치가 여러 개라면, 가장 낮은 번호의 열을 갖는 위치를 반환한다.

예를 들어 $N = 5$인 예시를 확인해 보자. 각 위치를 [행, 열]의 형태로 나타내고, 행과 열의 인덱스가 0부터 시작한다고 하면, 아래 예시에서 3명의 그래플러가 존재하는 위치는 차례대로 [1, 1], [2, 3], [4, 2]이다.

B	B	B	B	B
B	G	B	B	B
B	B	B	G	B
B	B	B	B	B
B	B	G	B	B

이 경우, 가장 가까운 그래플러와의 거리가 최대가 되는 지점들을 음영으로 표시하면 다음과 같다. 결과적으로 본 예시에서 **가장 가까운 그래플러와의 거리의 최댓값**은 3인 것을 알 수 있다.

B	B	B	B	B
B	G	B	B	B
B	B	B	G	B
B	B	B	B	B
B	B	G	B	B

따라서, 현재 예시에서는 본 문제에서 요구하는 [가장 낮은 행, 가장 낮은 열]의 위치는 [0, 4]이다.

입력 조건

가장 먼저 보드 판의 크기 N이 주어진다. N은 5 이상 500 이하의 자연수다.
이어서 행과 열의 길이가 각각 N인 2차원 배열 형태의 보드 판 정보 배열 $board$가 주어진다. 빈 공간은 "B", 그래플러의 위치는 "G"로 표시된다. 한 위치에 여러 명이 존재하지 않는다. 그래플러는 최소 2명 이상 주어지며, 빈 칸은 1개 이상 주어진다.

출력 조건

빈 칸 중에서 **가장 가까운 그래플러와의 거리가 최대가 되는 빈 칸**의 위치를 1차원 배열 [가장 낮은 행, 가장 낮은 열] 형태로 반환하여라. 가장 가까운 그래플러와의 거리가 최대가 되는 빈 칸이 여러 곳이라면, 가장 낮은 번호의 행을 갖는 위치를 반환한다. 만약 가장 낮은 번호의 행을 갖는 위치가 여러 개라면, 가장 낮은 번호의 열을 갖는 위치를 반환한다.

입출력 예시

N	$board$	정답
5	[["B", "B", "B", "B", "B"], ["B", "G", "B", "B", "B"], ["B", "B", "B", "G", "B"],	[0, 4]

	["B", "B", "B", "B", "B"], ["B", "B", "G", "B", "B"]]	
5	[["B", "B", "B", "B", "B"], ["B", "B", "B", "B", "B"], ["B", "B", "B", "B", "G"], ["B", "B", "B", "G", "B"], ["B", "B", "G", "B", "B"]]	[0, 0]

해설 2. 봉술가

본 문제는 **너비 우선 탐색(BFS)** 알고리즘을 이용하여 보드 판에 존재하는 N^2개의 모든 위치에 대하여 그래플러가 도달할 수 있는 최소 시간을 계산하는 방식으로 해결할 수 있다. 구체적으로, 가장 먼저 그래플러가 존재하는 모든 위치 값을 큐(queue) 자료구조에 삽입한다. 이후에 BFS 알고리즘을 수행하면, 모든 빈 칸에 대하여 가장 가까운 그래플러가 도달하기까지의 거리를 계산할 수 있다. 결과적으로 가장 거리 값이 큰 위치를 출력하는 것이 본 문제의 요구사항으로 이해할 수 있다. 이러한 알고리즘은 시간 복잡도 $O(N^2)$을 요구하며, 본 문제는 **너비 우선 탐색** 유형에 속한다.

- **Python3 정답 코드 예시**

```
from collections import deque

# 상, 하, 좌, 우
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def bfs(N, board):
    # 최단 거리 맵
    graph = [[-1] * N for _ in range(N)]
    # 너비 우선 탐색(BFS)을 위한 큐 생성
    q = deque()
    for i in range(N):
        for j in range(N):
            if board[i][j] == "G":
                # 각 위치를 큐에 삽입
                q.append((i, j))
```

```
                # 처음 그래플러가 존재하는 위치에 대한 최단 거리 설정
                graph[i][j] = 0
    # 너비 우선 탐색(BFS) 수행
    while q:
        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
            # 처음 방문하는 경우
            if graph[nx][ny] == -1:
                graph[nx][ny] = graph[x][y] + 1  # 최단 거리 기록
                q.append((nx, ny))
    return graph

# 전체 보드의 크기(N)와 보드 정보 배열(board)을 입력받기
def solution(N, board):
    graph = bfs(N, board)  # BFS를 활용한 최단 거리 탐색
    answer = [0, 0]  # 가장 가까운 그래플러와의 거리가 최대가 되는 위치
    max_value = 0  # 가장 가까운 그래플러와의 거리의 최댓값
    for i in range(N):
        for j in range(N):
            if max_value < graph[i][j]:
                answer = [i, j]
                max_value = graph[i][j]
    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 {
    // 상, 하, 좌, 우
    public static int[] dx = { -1, 1, 0, 0 };
    public static int[] dy = { 0, 0, -1, 1 };

    public static int[][] bfs(int N, String[][] board) {
        // 최단 거리 맵
        int[][] graph = new int[N][N];
        for (int i = 0; i < N; i++)
            Arrays.fill(graph[i], -1);
        // 너비 우선 탐색(BFS)을 위한 큐 생성
        Queue<Node> q = new LinkedList<>();
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (board[i][j].equals("G")) {
                    // 각 위치를 큐에 삽입
                    q.offer(new Node(i, j));
                    // 처음 그래플러가 존재하는 위치에 대한 최단 거리 설정
                    graph[i][j] = 0;
                }
            }
        }
        // 너비 우선 탐색(BFS) 수행
        while (!q.isEmpty()) {
            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;
                // 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[][] graph = bfs(N, board); // BFS를 활용한 최단 거리 탐색
        // 가장 가까운 그래플러와의 거리가 최대가 되는 위치
        int[] answer = new int[2];
        int maxValue = 0; // 가장 가까운 그래플러와의 거리의 최댓값
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (maxValue < graph[i][j]) {
                    answer[0] = i;
                    answer[1] = j;
                    maxValue = graph[i][j];
                }
            }
        }
        return answer;
    }
}
```