题目描述：

2XXX 年，人类通过对火星的大气进行宜居改造分析，使得火星已在理论上具备人类宜居的条件；由于技术原因，无法一次性将火星大气全部改造，只能通过局部处理形式；假设将火星待改造的区域为 row * column 的网格，每个网格有 3 个值，宜居区、可改造区、死亡区，使用 YES、NO、NA 代替，YES 表示该网格已经完成大气改造，NO 表示该网格未进行改造，后期可进行改造，NA 表示死亡区，不作为判断是否改造完成的宜居，无法穿过；

初始化下，该区域可能存在多个宜居区，并且每个宜居区能同时在每个太阳日单位向上下左右四个方向的相邻格子进行扩散，自动将 4 个方向相邻的真空区改造成宜居区；请计算这个待改造区域的网格中，可改造区是否能全部变成宜居区，如果可以，则返回改造的太阳日天数，不可以则返回 -1。

输入描述：

输入 row * column 个网格数据，每个网格值枚举值如下：YES，NO，NA；样例：

YES YES NO

NO NO NO

NA NO YES

输出描述：

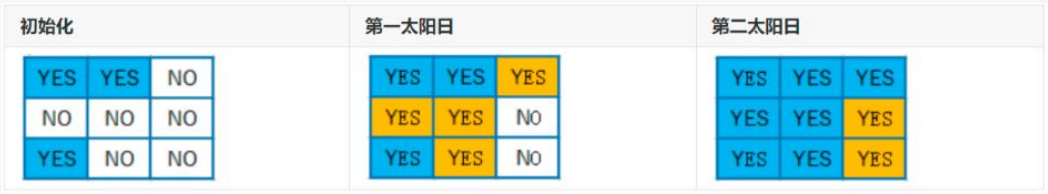
可改造区是否能全部变成宜居区，如果可以，则返回改造的太阳日天数，不可以则返回 -1。

补充说明：

grid[i][j]只有 3 种情况，YES、NO、NA

row == grid.length, column == grid[i].length, 1 <= row, column <= 8

示例1
输入：YES YES NO
NO NO NO
YES NO NO
输出：2
说明：经过2个太阳日，完成宜居改造。



示例 2

输入：

```
YES NO NO NO

NO NO NO NO

NO NO NO NO

NO NO NO NO
```

输出：

```
6
```

说明：

经过 6 个太阳日，可完成改造

示例 3

输入：

```
NO NA
```

输出：

```
-1
```

说明：

无改造初始条件，无法进行改造

示例 4

输入：

```
YES NO NO YES
NO NO YES NO
NO YES NA NA
YES NO NA NO
```

输出：

```
-1
```

说明：

-1 // 右下角的区域，被周边三个死亡区挡住，无法实现改造

```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        // 注意 hasNext 和 hasNextLine 的区别
        Map<Integer, String[]> map = new HashMap<>();
        int index = 0;
        while (in.hasNextLine()) { // 注意 while 处理多个 case
            String line = in.nextLine();
            if (line == null || "".equals(line)) {
                break;
            }
            String[] arrStr =  line.split(" ");
            map.put(index, arrStr);
            index++;
        }

        String[][] grid = new String[map.size()][map.get(0).length];
        for (int r = 0; r < map.size(); r++) {
            for (int c = 0; c < map.get(0).length ; c++) {
                grid[r][c] = map.get(r)[c];
            }
        }
        System.out.println(minDays(grid));
```

```
}

public static int minDays(String[][] grid) {
	int r = grid.length;
	int c = grid[0].length;

	Queue<int[]> queue = new LinkedList<>();
	int count = 0;

	for (int i = 0; i < r; i++) {
		for (int j = 0; j < c; j++) {
			if ("YES".equals(grid[i][j])) {
				queue.offer(new int[] {i, j});
			} else if ("NO".equals(grid[i][j])) {
				count++;
			}
		}
	}
	if (count == 0) {
		return 0;
	}
	int days = 0;
	int[][] directions = {{-1,0},{1,0},{0,-1},{0,1}};
	while (!queue.isEmpty()) {
		days++;
		int size = queue.size();
		for (int i = 0; i < size; i++) {
			int[] cell = queue.poll();
			for (int[] direction : directions) {
				int newR = cell[0] + direction[0];
				int newC= cell[1] + direction[1];
				if(newR>=0 && newR<r && newC >=0 && newC<c
				&& "NO".equals( grid[newR][newC])){
					queue.offer(new int[] {newR, newC});
					grid[newR][newC] = "YES";
					count --;
					if(count ==0){
						return days;
					}
				}
			}
		}
	}
```

```
            return -1;

        }
    }
```