Problem 361: Bomb Enemy

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given an

m x n

matrix

grid

where each cell is either a wall

'W'

, an enemy

'E'

or empty

'0'

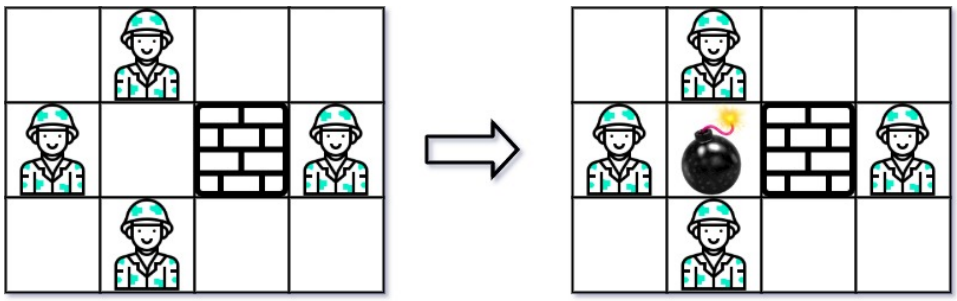
, return

the maximum enemies you can kill using one bomb

. You can only place the bomb in an empty cell.

The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since it is too strong to be destroyed.

Example 1:



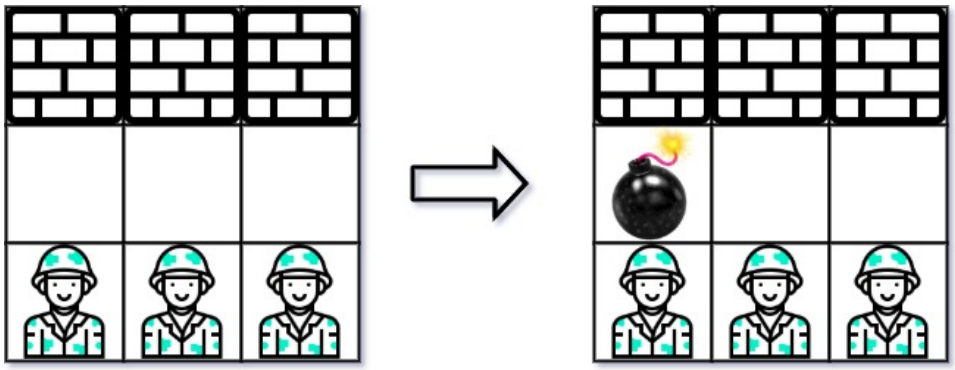
Input:

grid = [["0","E","0","0"],["E","0","W","E"],["0","E","0","0"]]

Output:

3

Example 2:



Input:

grid = [["W","W","W"],["0","0","0"],["E","E","E"]]

Output:

1

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 500

grid[i][j]

is either

'W'

,

'E'

, or

'0'

.

Code Snippets

C++:

```
class Solution {
public:
int maxKilledEnemies(vector<vector<char>>& grid) {

}
};
```

Java:

```
class Solution {
public int maxKilledEnemies(char[][] grid) {
```

```
}
}
```

Python3:

```
class Solution:
def maxKilledEnemies(self, grid: List[List[str]]) -> int:
```

Python:

```
class Solution(object):
def maxKilledEnemies(self, grid):
"""
:type grid: List[List[str]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {character[][]} grid
* @return {number}
*/
var maxKilledEnemies = function(grid) {

};
```

TypeScript:

```
function maxKilledEnemies(grid: string[][]): number {

};
```

C#:

```
public class Solution {
public int MaxKilledEnemies(char[][] grid) {

}
}
```

C:

```
int maxKilledEnemies(char** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func maxKilledEnemies(grid [][]byte) int {

}
```

Kotlin:

```
class Solution {
fun maxKilledEnemies(grid: Array<CharArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxKilledEnemies(_ grid: [[Character]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_killed_enemies(grid: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby:

```
# @param {Character[][]} grid
# @return {Integer}
def max_killed_enemies(grid)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $grid
* @return Integer
*/
function maxKilledEnemies($grid) {

}
}
```

Dart:

```
class Solution {
int maxKilledEnemies(List<List<String>> grid) {

}
}
```

Scala:

```
object Solution {
def maxKilledEnemies(grid: Array[Array[Char]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_killed_enemies(grid :: [[char]]) :: integer
def max_killed_enemies(grid) do

end
end
```

Erlang:

```
-spec max_killed_enemies(Grid :: [[char()]]) -> integer().
max_killed_enemies(Grid) ->
.
```

Racket:

```
(define/contract (max-killed-enemies grid)
(-> (listof (listof char?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Bomb Enemy
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxKilledEnemies(vector<vector<char>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Bomb Enemy
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxKilledEnemies(char[][] grid) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Bomb Enemy
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxKilledEnemies(self, grid: List[List[str]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxKilledEnemies(self, grid):
"""
:type grid: List[List[str]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Bomb Enemy
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
/**
* @param {character[][]} grid
* @return {number}
*/
var maxKilledEnemies = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Bomb Enemy
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxKilledEnemies(grid: string[][]): number {

};
```

C# Solution:

```
/*
* Problem: Bomb Enemy
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxKilledEnemies(char[][] grid) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Bomb Enemy
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxKilledEnemies(char** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Bomb Enemy
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxKilledEnemies(grid [][]byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxKilledEnemies(grid: Array<CharArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxKilledEnemies(_ grid: [[Character]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Bomb Enemy
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_killed_enemies(grid: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Character[][]} grid
# @return {Integer}
def max_killed_enemies(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $grid
* @return Integer
*/
function maxKilledEnemies($grid) {

}
}
```

Dart Solution:

```
class Solution {
int maxKilledEnemies(List<List<String>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def maxKilledEnemies(grid: Array[Array[Char]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_killed_enemies(grid :: [[char]]) :: integer
def max_killed_enemies(grid) do

end
end
```

Erlang Solution:

```
-spec max_killed_enemies(Grid :: [[char()]]) -> integer().
max_killed_enemies(Grid) ->
.
```

Racket Solution:

```
(define/contract (max-killed-enemies grid)
(-> (listof (listof char?)) exact-integer?)
)
```