Problem 749: Contain Virus

Problem Information

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

Problem Description

A virus is spreading rapidly, and your task is to quarantine the infected area by installing walls.

The world is modeled as an

m x n

binary grid

isInfected

, where

isInfected[i][j] == 0

represents uninfected cells, and

isInfected[i][j] == 1

represents cells contaminated with the virus. A wall (and only one wall) can be installed between any two

4-directionally

adjacent cells, on the shared boundary.

Every night, the virus spreads to all neighboring cells in all four directions unless blocked by a wall. Resources are limited. Each day, you can install walls around only one region (i.e., the affected area (continuous block of infected cells) that threatens the most uninfected cells the following night). There

will never be a tie

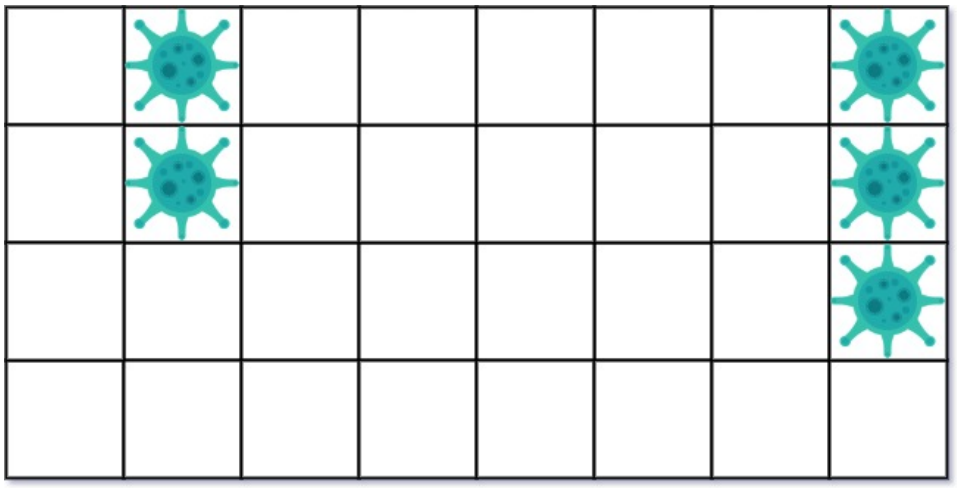
.

Return

the number of walls used to quarantine all the infected regions

. If the world will become fully infected, return the number of walls used.

Example 1:



Input:

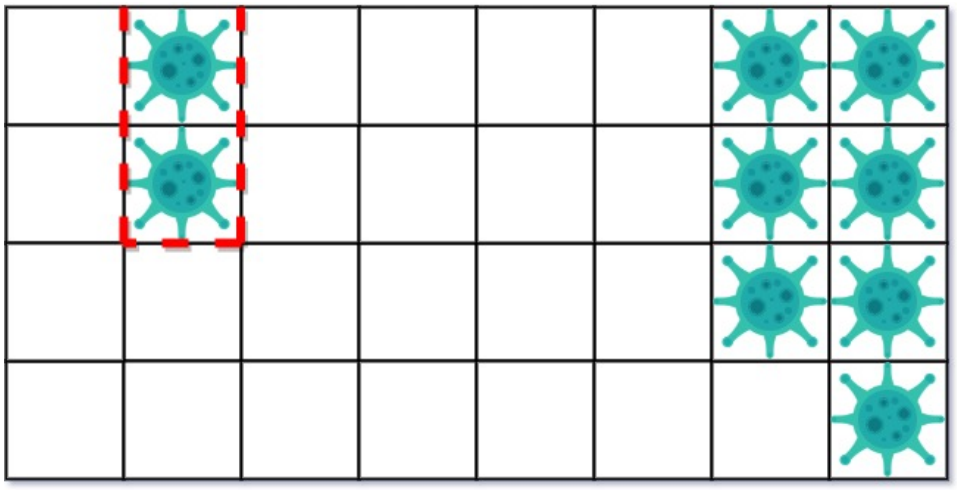
isInfected = [[0,1,0,0,0,0,0,1],[0,1,0,0,0,0,0,1],[0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,0]]

Output:

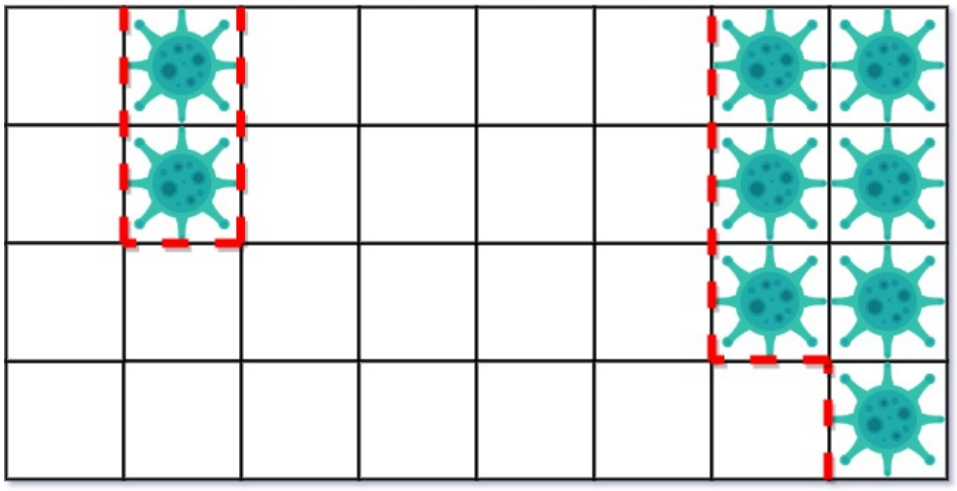
10

Explanation:

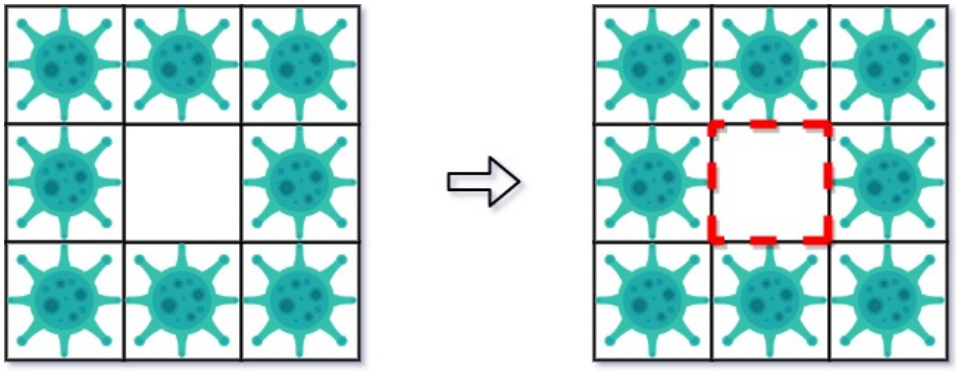
There are 2 contaminated regions. On the first day, add 5 walls to quarantine the viral region on the left. The board after the virus spreads is:



On the second day, add 5 walls to quarantine the viral region on the right. The virus is fully contained.



Example 2:



Input:

isInfected = [[1,1,1],[1,0,1],[1,1,1]]

Output:

4

Explanation:

Even though there is only one cell saved, there are 4 walls built. Notice that walls are only built on the shared boundary of two different cells.

Example 3:

Input:

isInfected = [[1,1,1,0,0,0,0,0,0],[1,0,1,0,1,1,1,1,1],[1,1,1,0,0,0,0,0,0]]

Output:

13

Explanation:

The region on the left only builds two new walls.

Constraints:

m == isInfected.length

n == isInfected[i].length

1 <= m, n <= 50

isInfected[i][j]

is either

0

or

1

.

There is always a contiguous viral region throughout the described process that will

infect strictly more uncontaminated squares

in the next round.

Code Snippets

C++:

```
class Solution {
public:
int containVirus(vector<vector<int>>& isInfected) {

}
};
```

Java:

```
class Solution {
public int containVirus(int[][] isInfected) {

}
}
```

Python3:

```
class Solution:
def containVirus(self, isInfected: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def containVirus(self, isInfected):
"""
:type isInfected: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} isInfected
* @return {number}
*/
var containVirus = function(isInfected) {

};
```

TypeScript:

```
function containVirus(isInfected: number[][]): number {

};
```

C#:

```
public class Solution {
public int ContainVirus(int[][] isInfected) {

}
}
```

C:

```
int containVirus(int** isInfected, int isInfectedSize, int*
isInfectedColSize) {

}
```

Go:

```
func containVirus(isInfected [][]int) int {

}
```

Kotlin:

```
class Solution {
fun containVirus(isInfected: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func containVirus(_ isInfected: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn contain_virus(is_infected: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} is_infected
# @return {Integer}
def contain_virus(is_infected)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $isInfected
* @return Integer
*/
function containVirus($isInfected) {

}
```

```
}
```

Dart:

```
class Solution {
int containVirus(List<List<int>> isInfected) {

}
}
```

Scala:

```
object Solution {
def containVirus(isInfected: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec contain_virus(is_infected :: [[integer]]) :: integer
def contain_virus(is_infected) do

end
end
```

Erlang:

```
-spec contain_virus(IsInfected :: [[integer()]]) -> integer().
contain_virus(IsInfected) ->
.
```

Racket:

```
(define/contract (contain-virus isInfected)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Contain Virus
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int containVirus(vector<vector<int>>& isInfected) {

}
};
```

Java Solution:

```
/**
* Problem: Contain Virus
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int containVirus(int[][] isInfected) {

}
}
```

Python3 Solution:

```
"""
Problem: Contain Virus
Difficulty: Hard
Tags: array, search
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

```
class Solution(object):
def containVirus(self, isInfected):
"""
:type isInfected: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Contain Virus
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} isInfected
* @return {number}
*/
var containVirus = function(isInfected) {

};
```

TypeScript Solution:

```
/**
* Problem: Contain Virus
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function containVirus(isInfected: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Contain Virus
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int ContainVirus(int[][] isInfected) {

}
}
```

C Solution:

```
/*
* Problem: Contain Virus
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

int containVirus(int** isInfected, int isInfectedSize, int*
isInfectedColSize) {

}
```

Go Solution:

```
// Problem: Contain Virus
// Difficulty: Hard
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func containVirus(isInfected [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun containVirus(isInfected: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func containVirus(_ isInfected: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Contain Virus
// Difficulty: Hard
```

```
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn contain_virus(is_infected: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} is_infected
# @return {Integer}
def contain_virus(is_infected)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $isInfected
* @return Integer
*/
function containVirus($isInfected) {

}
}
```

Dart Solution:

```
class Solution {
int containVirus(List<List<int>> isInfected) {

}
}
```

Scala Solution:

```
object Solution {
def containVirus(isInfected: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec contain_virus(is_infected :: [[integer]]) :: integer
def contain_virus(is_infected) do

end
end
```

Erlang Solution:

```
-spec contain_virus(IsInfected :: [[integer()]]) -> integer().
contain_virus(IsInfected) ->
.
```

Racket Solution:

```
(define/contract (contain-virus isInfected)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```