Problem 317: Shortest Distance from All Buildings

Problem Information

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

Problem Description

You are given an

m x n

grid

grid

of values

0

,

1

, or

2

, where:

each

0

marks

an empty land

that you can pass by freely,

each

1

marks

a building

that you cannot pass through, and

each

2

marks

an obstacle

that you cannot pass through.

You want to build a house on an empty land that reaches all buildings in the

shortest total travel

distance. You can only move up, down, left, and right.

Return

the

shortest travel distance

for such a house

. If it is not possible to build such a house according to the above rules, return

-1

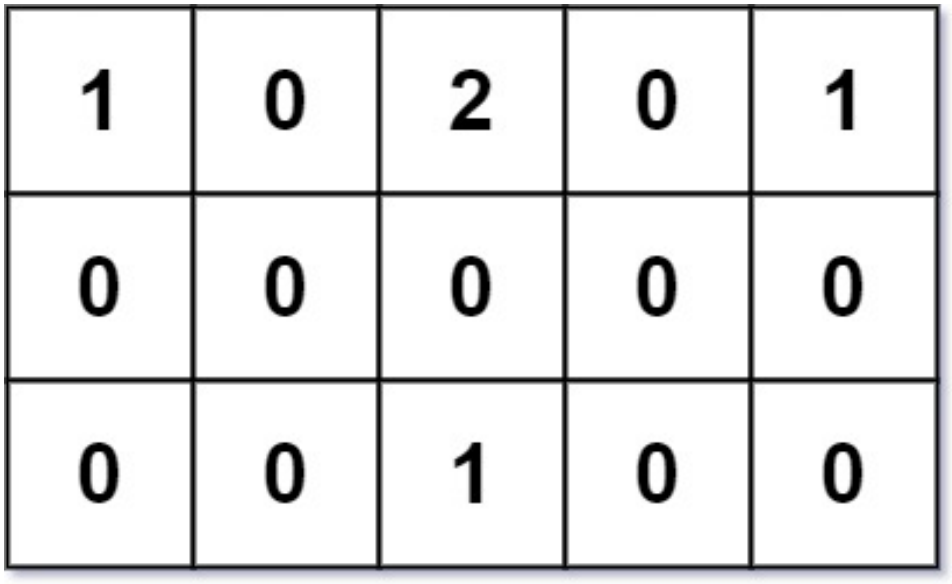
.

The

total travel distance

is the sum of the distances between the houses of the friends and the meeting point.

Example 1:



Input:

grid = [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]

Output:

7

Explanation:

Given three buildings at (0,0), (0,4), (2,2), and an obstacle at (0,2). The point (1,2) is an ideal empty land to build a house, as the total travel distance of 3+3+1=7 is minimal. So return 7.

Example 2:

Input:

grid = [[1,0]]

Output:

1

Example 3:

Input:

grid = [[1]]

Output:

-1

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 50

grid[i][j]

is either

0

,

1

, or

2

.

There will be

at least one

building in the

grid

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int shortestDistance(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shortestDistance(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int ShortestDistance(int[][] grid) {

}
}
```

C:

```
int shortestDistance(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func shortestDistance(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun shortestDistance(grid: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
```

```
*/
function shortestDistance($grid) {

}
}
```

Dart:

```
class Solution {
int shortestDistance(List<List<int>> grid) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (shortest-distance grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Distance from All Buildings
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Shortest Distance from All Buildings
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int shortestDistance(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Distance from All Buildings
Difficulty: Hard
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Distance from All Buildings
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: Shortest Distance from All Buildings
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function shortestDistance(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Distance from All Buildings
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int ShortestDistance(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Distance from All Buildings
* Difficulty: Hard
```

```
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int shortestDistance(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Shortest Distance from All Buildings
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func shortestDistance(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun shortestDistance(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Distance from All Buildings
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int shortestDistance(List<List<int>> grid) {
```

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (shortest-distance grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```