Problem 1091: Shortest Path in Binary Matrix

Problem Information

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

Problem Description

Given an

n x n

binary matrix

grid

, return

the length of the shortest

clear path

in the matrix

. If there is no clear path, return

-1

.

A

clear path

in a binary matrix is a path from the

top-left

cell (i.e.,

(0, 0)

) to the

bottom-right

cell (i.e.,

(n - 1, n - 1)

) such that:

All the visited cells of the path are

0

.

All the adjacent cells of the path are

8-directionally

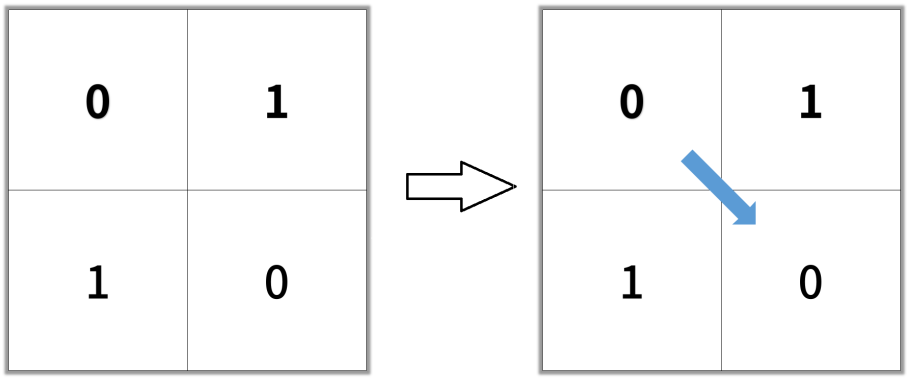
connected (i.e., they are different and they share an edge or a corner).

The

length of a clear path

is the number of visited cells of this path.

Example 1:



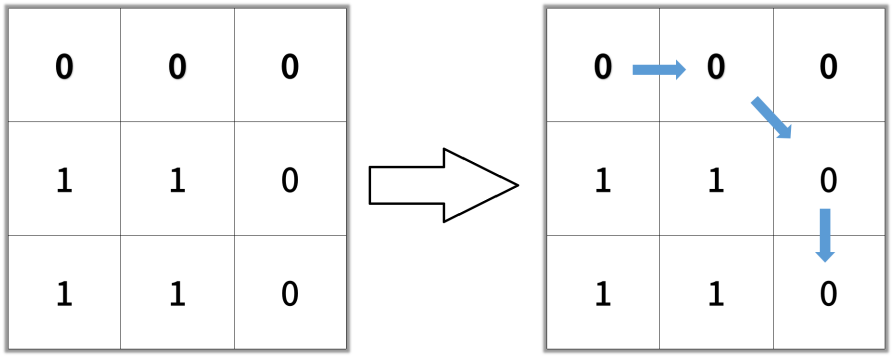
Input:

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

Output:

2

Example 2:



Input:

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

Output:

4

Example 3:

Input:

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

Output:

-1

Constraints:

n == grid.length

n == grid[i].length

1 <= n <= 100

grid[i][j] is 0 or 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (shortest-path-binary-matrix grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Path in Binary Matrix
* Difficulty: Medium
* 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 shortestPathBinaryMatrix(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Path in Binary Matrix
* Difficulty: Medium
* 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 shortestPathBinaryMatrix(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Path in Binary Matrix
Difficulty: Medium
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 shortestPathBinaryMatrix(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Path in Binary Matrix
* Difficulty: Medium
* 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[][]} grid
* @return {number}
*/
var shortestPathBinaryMatrix = function(grid) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Shortest Path in Binary Matrix
* Difficulty: Medium
* 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 shortestPathBinaryMatrix(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Path in Binary Matrix
* Difficulty: Medium
* 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 ShortestPathBinaryMatrix(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Path in Binary Matrix
* Difficulty: Medium
```

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

}
```

Go Solution:

```
// Problem: Shortest Path in Binary Matrix
// Difficulty: Medium
// 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 shortestPathBinaryMatrix(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Path in Binary Matrix
// Difficulty: Medium
// 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 shortest_path_binary_matrix(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (shortest-path-binary-matrix grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```