Problem 1293: Shortest Path in a Grid with Obstacles Elimination

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

grid

where each cell is either

0

(empty) or

1

(obstacle). You can move up, down, left, or right from and to an empty cell in

one step

.

Return

the minimum number of

steps

to walk from the upper left corner

(0, 0)

to the lower right corner

(m - 1, n - 1)

given that you can eliminate

at most

k

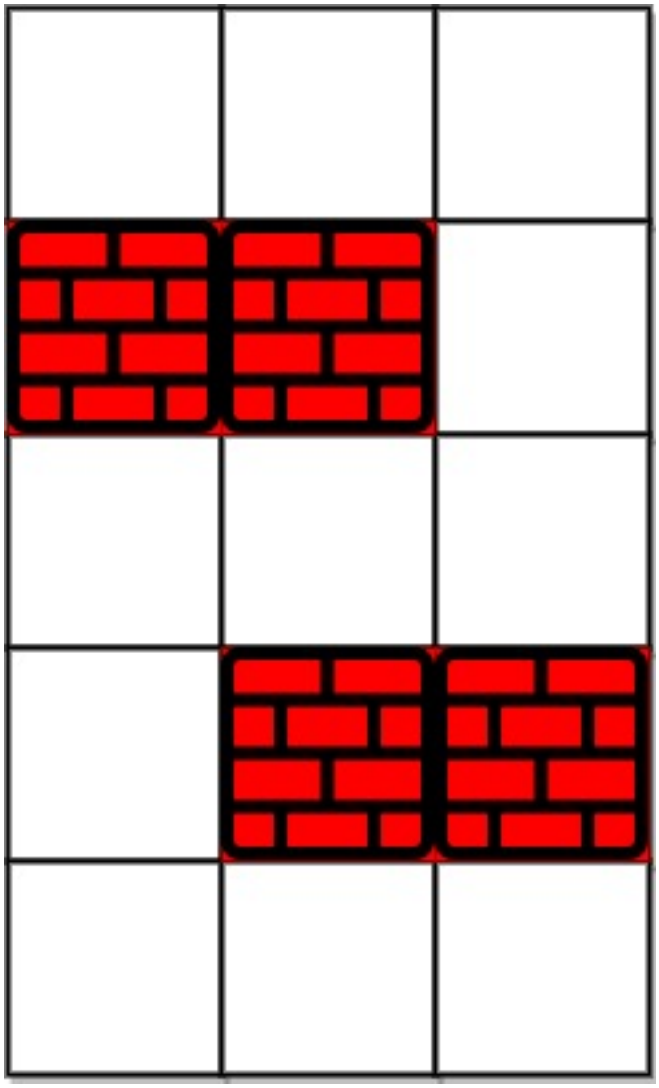
obstacles

. If it is not possible to find such walk return

-1

.

Example 1:



Input:

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

Output:

6

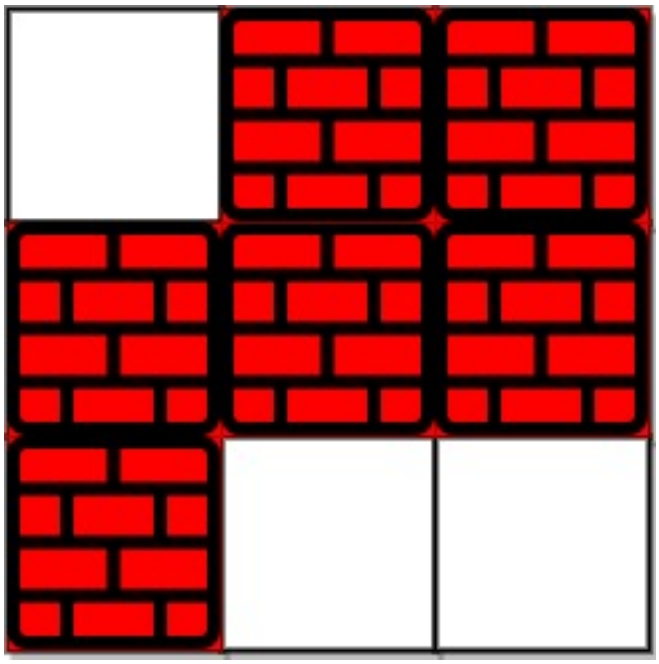
Explanation:

The shortest path without eliminating any obstacle is 10. The shortest path with one obstacle elimination at position (3,2) is 6. Such path is (0,0) -> (0,1) -> (0,2) -> (1,2) -> (2,2) ->

(3,2)

-> (4,2).

Example 2:



Input:

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

Output:

-1

Explanation:

We need to eliminate at least two obstacles to find such a walk.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 40

1 <= k <= m * n

grid[i][j]

is either

0

or

1

.

grid[0][0] == grid[m - 1][n - 1] == 0

Code Snippets

C++:

```
class Solution {
public:
int shortestPath(vector<vector<int>>& grid, int k) {

}
};
```

Java:

```
class Solution {
public int shortestPath(int[][] grid, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shortestPath(grid: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int ShortestPath(int[][] grid, int k) {

}
}
```

C:

```
int shortestPath(int** grid, int gridSize, int* gridColSize, int k) {

}
```

Go:

```
func shortestPath(grid [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun shortestPath(grid: Array<IntArray>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func shortestPath(_ grid: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn shortest_path(grid: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer}
def shortest_path(grid, k)

end
```

PHP:

```
class Solution {

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

```
* @param Integer $k
* @return Integer
*/
function shortestPath($grid, $k) {

}
}
```

Dart:

```
class Solution {
int shortestPath(List<List<int>> grid, int k) {

}
}
```

Scala:

```
object Solution {
def shortestPath(grid: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_path(grid :: [[integer]], k :: integer) :: integer
def shortest_path(grid, k) do

end
end
```

Erlang:

```
-spec shortest_path(Grid :: [[integer()]], K :: integer()) -> integer().
shortest_path(Grid, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Shortest Path in a Grid with Obstacles Elimination
* Difficulty: Hard
* Tags: array, tree, graph, 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 shortestPath(vector<vector<int>>& grid, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Path in a Grid with Obstacles Elimination
* Difficulty: Hard
* Tags: array, tree, graph, 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 shortestPath(int[][] grid, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Shortest Path in a Grid with Obstacles Elimination
Difficulty: Hard
Tags: array, tree, graph, 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 shortestPath(self, grid: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Path in a Grid with Obstacles Elimination
* Difficulty: Hard
* Tags: array, tree, graph, 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
* @param {number} k
* @return {number}
*/
var shortestPath = function(grid, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Path in a Grid with Obstacles Elimination
* Difficulty: Hard
* Tags: array, tree, graph, 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 shortestPath(grid: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Path in a Grid with Obstacles Elimination
* Difficulty: Hard
* Tags: array, tree, graph, 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 ShortestPath(int[][] grid, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Shortest Path in a Grid with Obstacles Elimination
* Difficulty: Hard
* Tags: array, tree, graph, 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 shortestPath(int** grid, int gridSize, int* gridColSize, int k) {

}
```

Go Solution:

```
// Problem: Shortest Path in a Grid with Obstacles Elimination
// Difficulty: Hard
// Tags: array, tree, graph, 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 shortestPath(grid [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun shortestPath(grid: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func shortestPath(_ grid: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Shortest Path in a Grid with Obstacles Elimination
// Difficulty: Hard
// Tags: array, tree, graph, 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_path(grid: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer} k
# @return {Integer}
def shortest_path(grid, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer $k
* @return Integer
*/
function shortestPath($grid, $k) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int shortestPath(List<List<int>> grid, int k) {

}
}
```

Scala Solution:

```
object Solution {
def shortestPath(grid: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_path(grid :: [[integer]], k :: integer) :: integer
def shortest_path(grid, k) do

end
end
```

Erlang Solution:

```
-spec shortest_path(Grid :: [[integer()]], K :: integer()) -> integer().
shortest_path(Grid, K) ->
.
```

Racket Solution:

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