Problem 3552: Grid Teleportation Traversal

Problem Information

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

Problem Description

You are given a 2D character grid

matrix

of size

m x n

, represented as an array of strings, where

matrix[i][j]

represents the cell at the intersection of the

i

th

row and

j

th

column. Each cell is one of the following:

'.'

representing an empty cell.

'#'

representing an obstacle.

An uppercase letter (

'A'

-

'Z'

) representing a teleportation portal.

You start at the top-left cell

(0, 0)

, and your goal is to reach the bottom-right cell

(m - 1, n - 1)

. You can move from the current cell to any adjacent cell (up, down, left, right) as long as the destination cell is within the grid bounds and is not an obstacle

.

If you step on a cell containing a portal letter and you haven't used that portal letter before, you may instantly teleport to any other cell in the grid with the same letter. This teleportation does not count as a move, but each portal letter can be used

at most

once during your journey.

Return the

minimum

number of moves required to reach the bottom-right cell. If it is not possible to reach the destination, return

-1

.

Example 1:

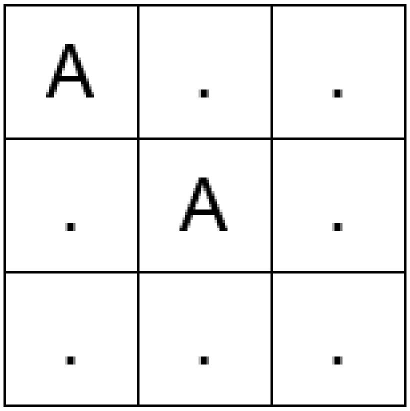
Input:

matrix = ["A..",".A.","..."]

Output:

2

Explanation:



A	.	.
.	A	.
.	.	.

Before the first move, teleport from

(0, 0)

to

(1, 1)

.

In the first move, move from

(1, 1)

to

(1, 2)

.

In the second move, move from

(1, 2)

to

(2, 2)

.

Example 2:

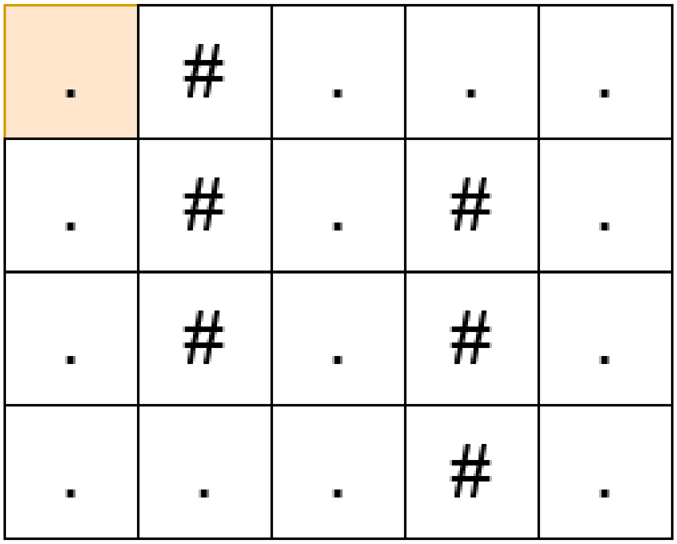
Input:

matrix = [".#...",".#.#.",".#.#.","...#."]

Output:

13

Explanation:



.	#	.	.	.
.	#	.	#	.
.	#	.	#	.
.	.	.	#	.

Constraints:

1 <= m == matrix.length <= 10

3

1 <= n == matrix[i].length <= 10

3

matrix[i][j]

is either

'#'

,

'.'

, or an uppercase English letter.

matrix[0][0]

is not an obstacle.

Code Snippets

C++:

```
class Solution {
public:
int minMoves(vector<string>& matrix) {

}
};
```

Java:

```
class Solution {
public int minMoves(String[] matrix) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} matrix
* @return {number}
*/
var minMoves = function(matrix) {

};
```

TypeScript:

```
function minMoves(matrix: string[]): number {

};
```

C#:

```
public class Solution {
public int MinMoves(string[] matrix) {

}
}
```

C:

```
int minMoves(char** matrix, int matrixSize) {

}
```

Go:

```
func minMoves(matrix []string) int {

}
```

Kotlin:

```
class Solution {
fun minMoves(matrix: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func minMoves(_ matrix: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_moves(matrix: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} matrix
# @return {Integer}
def min_moves(matrix)

end
```

PHP:

```
class Solution {

/**
* @param String[] $matrix
* @return Integer
*/
function minMoves($matrix) {

}
}
```

Dart:

```
class Solution {
int minMoves(List<String> matrix) {

}
}
```

Scala:

```
object Solution {
def minMoves(matrix: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_moves(matrix :: [String.t]) :: integer
def min_moves(matrix) do

end
end
```

Erlang:

```
-spec min_moves(Matrix :: [unicode:unicode_binary()]) -> integer().
min_moves(Matrix) ->
.
```

Racket:

```
(define/contract (min-moves matrix)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Grid Teleportation Traversal
* Difficulty: Medium
* Tags: array, string, tree, hash, 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 minMoves(vector<string>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Grid Teleportation Traversal
* Difficulty: Medium
* Tags: array, string, tree, hash, 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 minMoves(String[] matrix) {

}
}
```

Python3 Solution:

```
"""
Problem: Grid Teleportation Traversal
Difficulty: Medium
Tags: array, string, tree, hash, 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 minMoves(self, matrix: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Grid Teleportation Traversal
* Difficulty: Medium
* Tags: array, string, tree, hash, 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 {string[]} matrix
* @return {number}
*/
var minMoves = function(matrix) {

};
```

TypeScript Solution:

```
/**
* Problem: Grid Teleportation Traversal
* Difficulty: Medium
* Tags: array, string, tree, hash, 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 minMoves(matrix: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Grid Teleportation Traversal
* Difficulty: Medium
* Tags: array, string, tree, hash, 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 MinMoves(string[] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Grid Teleportation Traversal
* Difficulty: Medium
* Tags: array, string, tree, hash, 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 minMoves(char** matrix, int matrixSize) {

}
```

Go Solution:

```
// Problem: Grid Teleportation Traversal
// Difficulty: Medium
// Tags: array, string, tree, hash, 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 minMoves(matrix []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMoves(matrix: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minMoves(_ matrix: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Grid Teleportation Traversal
// Difficulty: Medium
// Tags: array, string, tree, hash, 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 min_moves(matrix: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} matrix
# @return {Integer}
def min_moves(matrix)
```

```


end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $matrix
* @return Integer
*/
function minMoves($matrix) {

}
}
```

Dart Solution:

```
class Solution {
int minMoves(List<String> matrix) {

}
}
```

Scala Solution:

```
object Solution {
def minMoves(matrix: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_moves(matrix :: [String.t]) :: integer
def min_moves(matrix) do

end
end
```

Erlang Solution:

```
-spec min_moves(Matrix :: [unicode:unicode_binary()]) -> integer().
min_moves(Matrix) ->
  .
```

Racket Solution:

```
(define/contract (min-moves matrix)
  (-> (listof string?) exact-integer?)
  )
```