Problem 3568: Minimum Moves to Clean the Classroom

Problem Information

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

Problem Description

You are given an

m x n

grid

classroom

where a student volunteer is tasked with cleaning up litter scattered around the room. Each cell in the grid is one of the following:

'S'

: Starting position of the student

'L'

: Litter that must be collected (once collected, the cell becomes empty)

'R'

: Reset area that restores the student's energy to full capacity, regardless of their current energy level (can be used multiple times)

'X'

: Obstacle the student cannot pass through

'.'

: Empty space

You are also given an integer

energy

, representing the student's maximum energy capacity. The student starts with this energy from the starting position

'S'

.

Each move to an adjacent cell (up, down, left, or right) costs 1 unit of energy. If the energy reaches 0, the student can only continue if they are on a reset area

'R'

, which resets the energy to its

maximum

capacity

energy

.

Return the

minimum

number of moves required to collect all litter items, or

-1

if it's impossible.

Example 1:

Input:

classroom = ["S.", "XL"], energy = 2

Output:

2

Explanation:

The student starts at cell

(0, 0)

with 2 units of energy.

Since cell

(1, 0)

contains an obstacle 'X', the student cannot move directly downward.

A valid sequence of moves to collect all litter is as follows:

Move 1: From

(0, 0)

→

(0, 1)

with 1 unit of energy and 1 unit remaining.

Move 2: From

(0, 1)

$\rightarrow$

(1, 1)

to collect the litter

'L'

.

The student collects all the litter using 2 moves. Thus, the output is 2.

Example 2:

Input:

classroom = ["LS", "RL"], energy = 4

Output:

3

Explanation:

The student starts at cell

(0, 1)

with 4 units of energy.

A valid sequence of moves to collect all litter is as follows:

Move 1: From

(0, 1)

→

(0, 0)

to collect the first litter

'L'

with 1 unit of energy used and 3 units remaining.

Move 2: From

(0, 0)

→

(1, 0)

to

'R'

to reset and restore energy back to 4.

Move 3: From

(1, 0)

→

(1, 1)

to collect the second litter

'L'

.

The student collects all the litter using 3 moves. Thus, the output is 3.

Example 3:

Input:

classroom = ["L.S", "RXL"], energy = 3

Output:

-1

Explanation:

No valid path collects all

'L'

.

Constraints:

1 <= m == classroom.length <= 20

1 <= n == classroom[i].length <= 20

classroom[i][j]

is one of

'S'

,

'L'

,

'R'

,

'X'

, or

'.'

1 <= energy <= 50

There is exactly

one

'S'

in the grid.

There are

at most

10

'L'

cells in the grid.

Code Snippets

C++:

```
class Solution {
public:
int minMoves(vector<string>& classroom, int energy) {

}
};
```

Java:

```
class Solution {
public int minMoves(String[] classroom, int energy) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minMoves(classroom: string[], energy: number): number {

};
```

C#:

```
public class Solution {
public int MinMoves(string[] classroom, int energy) {

}
}
```

C:

```
int minMoves(char** classroom, int classroomSize, int energy) {

}
```

Go:

```
func minMoves(classroom []string, energy int) int {

}
```

Kotlin:

```
class Solution {
fun minMoves(classroom: Array<String>, energy: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String[]} classroom
# @param {Integer} energy
# @return {Integer}
def min_moves(classroom, energy)

end
```

PHP:

```
class Solution {

/**
* @param String[] $classroom
* @param Integer $energy
* @return Integer
*/
function minMoves($classroom, $energy) {

}
}
```

Dart:

```
class Solution {
int minMoves(List<String> classroom, int energy) {

}
}
```

Scala:

```
object Solution {
def minMoves(classroom: Array[String], energy: Int): Int = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

```
-spec min_moves(Classroom :: [unicode:unicode_binary()], Energy :: integer())
-> integer().
min_moves(Classroom, Energy) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Clean the Classroom
* Difficulty: Medium
* Tags: array, 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>& classroom, int energy) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Clean the Classroom
* Difficulty: Medium
* Tags: array, 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[] classroom, int energy) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Clean the Classroom
Difficulty: Medium
Tags: array, 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, classroom: List[str], energy: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Clean the Classroom
* Difficulty: Medium
* Tags: array, 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[]} classroom
* @param {number} energy
* @return {number}
*/
var minMoves = function(classroom, energy) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Clean the Classroom
* Difficulty: Medium
* Tags: array, 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(classroom: string[], energy: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Clean the Classroom
* Difficulty: Medium
```

```
* Tags: array, 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[] classroom, int energy) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Clean the Classroom
* Difficulty: Medium
* Tags: array, 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** classroom, int classroomSize, int energy) {

}
```

Go Solution:

```
// Problem: Minimum Moves to Clean the Classroom
// Difficulty: Medium
// Tags: array, 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(classroom []string, energy int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minMoves(classroom: Array<String>, energy: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Moves to Clean the Classroom
// Difficulty: Medium
// Tags: array, 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(classroom: Vec<String>, energy: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} classroom
# @param {Integer} energy
# @return {Integer}
def min_moves(classroom, energy)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $classroom
* @param Integer $energy
* @return Integer
*/
function minMoves($classroom, $energy) {

}
}
```

Dart Solution:

```
class Solution {
int minMoves(List<String> classroom, int energy) {

}
}
```

Scala Solution:

```
object Solution {
def minMoves(classroom: Array[String], energy: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_moves(Classroom :: [unicode:unicode_binary()], Energy :: integer())
-> integer().
min_moves(Classroom, Energy) ->
.
```

Racket Solution:

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