Problem 1810: Minimum Path Cost in a Hidden Grid

Problem Information

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

Problem Description

This is an

interactive problem

.

There is a robot in a hidden grid, and you are trying to get it from its starting cell to the target cell in this grid. The grid is of size

m x n

, and each cell in the grid is either empty or blocked. It is

guaranteed

that the starting cell and the target cell are different, and neither of them is blocked.

Each cell has a

cost

that you need to pay each time you

move

to the cell. The starting cell's cost is

not

applied before the robot moves.

You want to find the minimum total cost to move the robot to the target cell. However, you

do not know

the grid's dimensions, the starting cell, nor the target cell. You are only allowed to ask queries to the

GridMaster

object.

The

GridMaster

class has the following functions:

boolean canMove(char direction)

Returns

true

if the robot can move in that direction. Otherwise, it returns

false

.

int move(char direction)

Moves the robot in that direction and returns the cost of moving to that cell. If this move would move the robot to a blocked cell or off the grid, the move will be ignored

, the robot will remain in the same position, and the function will return

-1

.

boolean isTarget()

Returns

true

if the robot is currently on the target cell. Otherwise, it returns

false

.

Note that

direction

in the above functions should be a character from

{'U','D','L','R'}

, representing the directions up, down, left, and right, respectively.

Return

the

minimum total cost

to get the robot from its initial starting cell to the target cell. If there is no valid path between the cells, return

-1

.

Custom testing:

The test input is read as a 2D matrix

grid

of size

m x n

and four integers

r1

,

c1

,

r2

, and

c2

where:

grid[i][j] == 0

indicates that the cell

(i, j)

is blocked.

grid[i][j] >= 1

indicates that the cell

(i, j)

is empty and

grid[i][j]

is the

cost

to move to that cell.

(r1, c1)

is the starting cell of the robot.

(r2, c2)

is the target cell of the robot.

Remember that you will

not

have this information in your code.

Example 1:

Input:

grid = [[2,3],[1,1]], r1 = 0, c1 = 1, r2 = 1, c2 = 0

Output:

2

Explanation:

One possible interaction is described below: The robot is initially standing on cell (0, 1), denoted by the 3. - master.canMove('U') returns false. - master.canMove('D') returns true. - master.canMove('L') returns true. - master.canMove('R') returns false. - master.move('L') moves the robot to the cell (0, 0) and returns 2. - master.isTarget() returns false. - master.canMove('U') returns false. - master.canMove('D') returns true. - master.canMove('L') returns false. - master.canMove('R') returns true. - master.move('D') moves the robot to the cell (1, 0) and returns 1. - master.isTarget() returns true. - master.move('L') doesn't move the robot and returns -1. - master.move('R') moves the robot to the cell (1, 1) and returns 1. We now know that the target is the cell (1, 0), and the minimum total cost to reach it is 2.

Example 2:

Input:

grid = [[0,3,1],[3,4,2],[1,2,0]], r1 = 2, c1 = 0, r2 = 0, c2 = 2

Output:

9

Explanation:

The minimum cost path is (2,0) -> (2,1) -> (1,1) -> (1,2) -> (0,2).

Example 3:

Input:

grid = [[1,0],[0,1]], r1 = 0, c1 = 0, r2 = 1, c2 = 1

Output:

-1

Explanation:

There is no path from the robot to the target cell.

Constraints:

1 <= n, m <= 100

m == grid.length

n == grid[i].length

0 <= grid[i][j] <= 100

Code Snippets

C++:

```
/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* class GridMaster {
* public:
* bool canMove(char direction);
* int move(char direction);
* boolean isTarget();
* };
*/

class Solution {
public:
int findShortestPath(GridMaster &master) {

}
};
```

Java:

```
/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* class GridMaster {
```

```
* boolean canMove(char direction);
* int move(char direction);
* boolean isTarget();
* }
*/

class Solution {
public int findShortestPath(GridMaster master) {

}
}
```

Python3:

```
# """
# This is GridMaster's API interface.
# You should not implement it, or speculate about its implementation
# """
#class GridMaster(object):
# def canMove(self, direction: str) -> bool:
#
#
# def move(self, direction: str) -> int:
#
#
# def isTarget(self) -> bool:
#
#

class Solution(object):
def findShortestPath(self, master: 'GridMaster') -> int:
```

Python:

```
# """
# This is GridMaster's API interface.
# You should not implement it, or speculate about it's implementation
# """
#class GridMaster(object):
# def canMove(self, direction):
# """
```

```
# :type direction: str
# :rtype bool
# """
#
# def move(self, direction):
# """
# :type direction: str
#. :rtype int
# """
#
# def isTarget(self):
# """
# :rtype bool
# """
#

class Solution(object):
def findShortestPath(self, master):
"""
:type master: GridMaster
:rtype: int
"""
```

JavaScript:

```
/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* function GridMaster() {
*
* @param {character} direction
* @return {boolean}
* this.canMove = function(direction) {
* ...
* };
* @param {character} direction
* @return {integer}
* this.move = function(direction) {
* ...
* };
* @return {boolean}
* this.isTarget = function() {
```

```
* ...
* };
* };
*/

/**
* @param {GridMaster} master
* @return {integer}
*/
var findShortestPath = function(master) {

};
```

C#:

```
/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* class GridMaster {
* bool canMove(char direction);
* int move(char direction);
* bool isTarget();
* };
*/

class Solution {
public int FindShortestPath(GridMaster master) {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Path Cost in a Hidden Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
```

```
* 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
*/

/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* class GridMaster {
* public:
* bool canMove(char direction);
* int move(char direction);
* boolean isTarget();
* };
*/

class Solution {
public:
int findShortestPath(GridMaster &master) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Path Cost in a Hidden Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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
*/

/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* class GridMaster {
* boolean canMove(char direction);
* int move(char direction);
```

```
* boolean isTarget();
* }
*/

class Solution {
public int findShortestPath(GridMaster master) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Path Cost in a Hidden Grid
Difficulty: Medium
Tags: array, graph, search, queue, heap

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
"""

# """
# This is GridMaster's API interface.
# You should not implement it, or speculate about its implementation
# """
#class GridMaster(object):
# def canMove(self, direction: str) -> bool:
#
#
# def move(self, direction: str) -> int:
#
#
# def isTarget(self) -> bool:
#
#

class Solution(object):
def findShortestPath(self, master: 'GridMaster') -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# """
# This is GridMaster's API interface.
# You should not implement it, or speculate about it's implementation
# """
#class GridMaster(object):
# def canMove(self, direction):
# """
# :type direction: str
# :rtype bool
# """
#
# def move(self, direction):
# """
# :type direction: str
#. :rtype int
# """
#
# def isTarget(self):
# """
# :rtype bool
# """
#

class Solution(object):
def findShortestPath(self, master):
"""
:type master: GridMaster
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Path Cost in a Hidden Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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
```

```
*/

/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* function GridMaster() {
*
* @param {character} direction
* @return {boolean}
* this.canMove = function(direction) {
* ...
* };
* @param {character} direction
* @return {integer}
* this.move = function(direction) {
* ...
* };
* @return {boolean}
* this.isTarget = function() {
* ...
* };
* };
*/

/**
* @param {GridMaster} master
* @return {integer}
*/
var findShortestPath = function(master) {

};
```

C# Solution:

```
/*
* Problem: Minimum Path Cost in a Hidden Grid
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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
*/

/**
* // This is the GridMaster's API interface.
* // You should not implement it, or speculate about its implementation
* class GridMaster {
* bool canMove(char direction);
* int move(char direction);
* bool isTarget();
* };
*/

class Solution {
public int FindShortestPath(GridMaster master) {

}
}
```