Problem 1391: Check if There is a Valid Path in a Grid

Problem Information

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

Problem Description

You are given an

m x n

grid

. Each cell of

grid

represents a street. The street of

grid[i][j]

can be:

1

which means a street connecting the left cell and the right cell.

2

which means a street connecting the upper cell and the lower cell.

3

which means a street connecting the left cell and the lower cell.

4

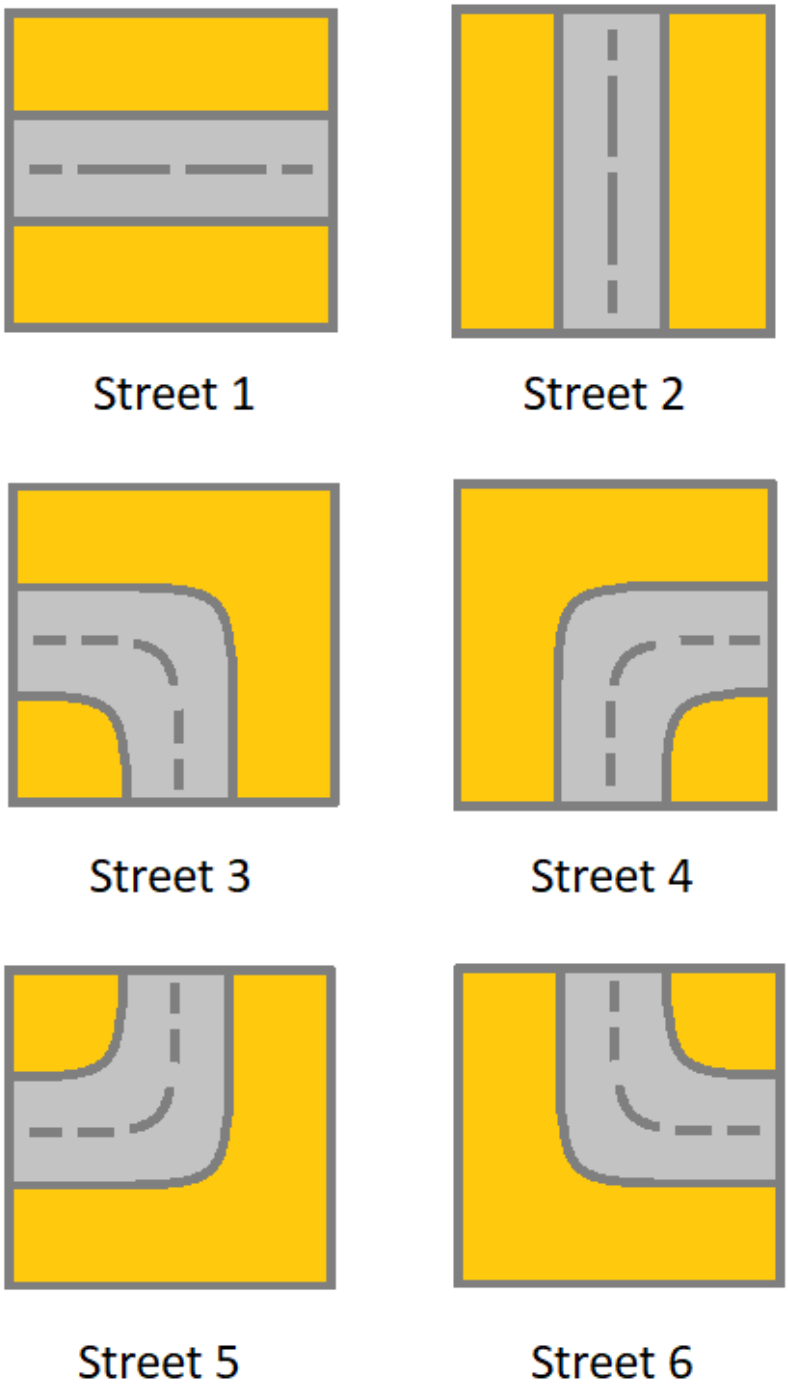
which means a street connecting the right cell and the lower cell.

5

which means a street connecting the left cell and the upper cell.

6

which means a street connecting the right cell and the upper cell.



You will initially start at the street of the upper-left cell

(0, 0)

. A valid path in the grid is a path that starts from the upper left cell

(0, 0)

and ends at the bottom-right cell

(m - 1, n - 1)

.

The path should only follow the streets

.

Notice

that you are

not allowed

to change any street.

Return

true

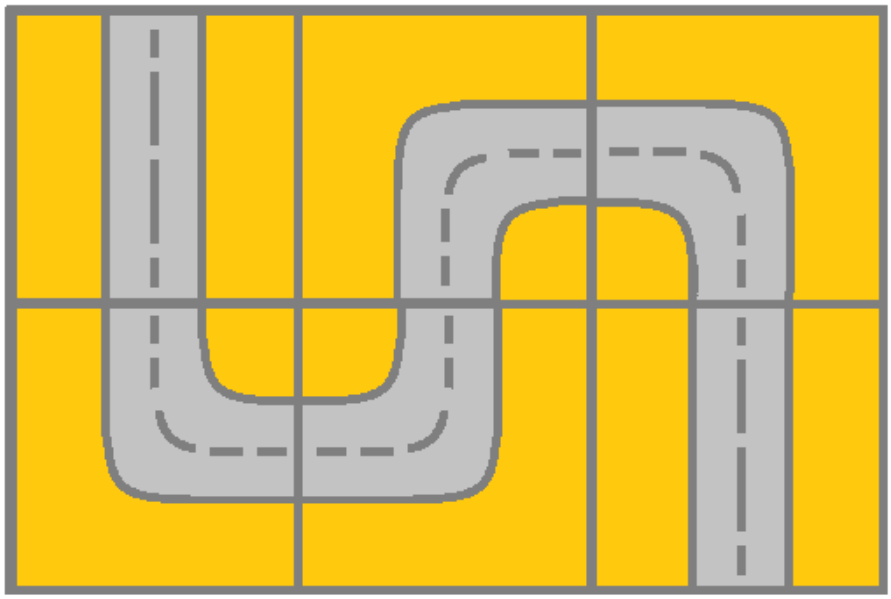
if there is a valid path in the grid or

false

otherwise

.

Example 1:



Input:

grid = [[2,4,3],[6,5,2]]

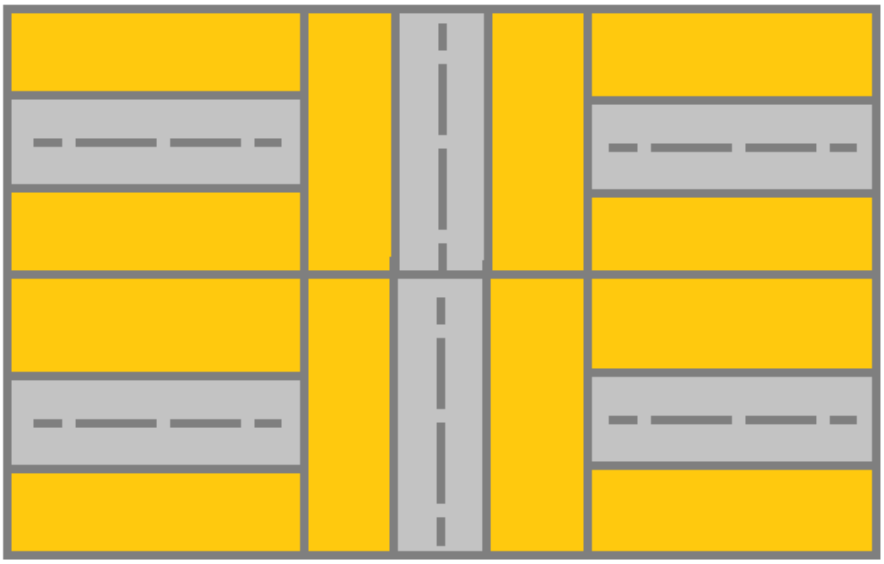
Output:

true

Explanation:

As shown you can start at cell (0, 0) and visit all the cells of the grid to reach (m - 1, n - 1).

Example 2:



Input:

grid = [[1,2,1],[1,2,1]]

Output:

false

Explanation:

As shown you the street at cell (0, 0) is not connected with any street of any other cell and you will get stuck at cell (0, 0)

Example 3:

Input:

grid = [[1,1,2]]

Output:

false

Explanation:

You will get stuck at cell (0, 1) and you cannot reach cell (0, 2).

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 300

1 <= grid[i][j] <= 6

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean hasValidPath(int[][] grid) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def hasValidPath(self, grid):
```

```
"""
:type grid: List[List[int]]
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function hasValidPath(grid: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool HasValidPath(int[][] grid) {

}
}
```

C:

```
bool hasValidPath(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func hasValidPath(grid [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun hasValidPath(grid: Array<IntArray>): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool hasValidPath(List<List<int>> grid) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (has-valid-path grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if There is a Valid Path in a Grid
* Difficulty: Medium
* 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:
bool hasValidPath(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Check if There is a Valid Path in a Grid
* Difficulty: Medium
* 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 boolean hasValidPath(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if There is a Valid Path in a Grid
Difficulty: Medium
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 hasValidPath(self, grid: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if There is a Valid Path in a Grid
* Difficulty: Medium
* 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
* @return {boolean}
*/
var hasValidPath = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if There is a Valid Path in a Grid
* Difficulty: Medium
* 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 hasValidPath(grid: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if There is a Valid Path in a Grid
* Difficulty: Medium
* 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 bool HasValidPath(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Check if There is a Valid Path in a Grid
* Difficulty: Medium
* 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
```

```
*/

bool hasValidPath(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Check if There is a Valid Path in a Grid
// Difficulty: Medium
// 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 hasValidPath(grid [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun hasValidPath(grid: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check if There is a Valid Path in a Grid
// Difficulty: Medium
// 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 has_valid_path(grid: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool hasValidPath(List<List<int>> grid) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (has-valid-path grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```