Problem 3565: Sequential Grid Path Cover

Problem Information

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

Problem Description

You are given a 2D array

grid

of size

m x n

, and an integer

k

. There are

k

cells in

grid

containing the values from 1 to

k

exactly once

, and the rest of the cells have a value 0.

You can start at any cell, and move from a cell to its neighbors (up, down, left, or right). You must find a path in

grid

which:

Visits each cell in

grid

exactly once

.

Visits the cells with values from 1 to

k

in order

.

Return a 2D array

result

of size

(m * n) x 2

, where

result[i] = [x

i

, y

i

]

represents the

i

th

cell visited in the path. If there are multiple such paths, you may return

any

one.

If no such path exists, return an

empty

array.

Example 1:

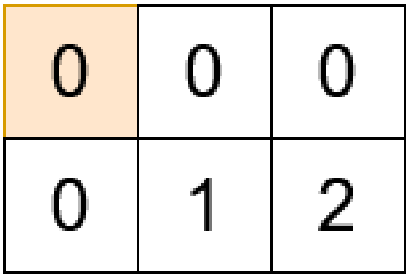
Input:

grid = [[0,0,0],[0,1,2]], k = 2

Output:

[[0,0],[1,0],[1,1],[1,2],[0,2],[0,1]]

Explanation:



0	0	0
0	1	2

Example 2:

Input:

grid = [[1,0,4],[3,0,2]], k = 4

Output:

[]

Explanation:

There is no possible path that satisfies the conditions.

Constraints:

1 <= m == grid.length <= 5

1 <= n == grid[i].length <= 5

1 <= k <= m * n

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

grid

contains all integers between 1 and

k

exactly

once.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<List<Integer>> findPath(int[][] grid, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<int>> FindPath(int[][] grid, int k) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findPath(int** grid, int gridSize, int* gridColSize, int k, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}

}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sequential Grid Path Cover
* Difficulty: Medium
* Tags: array
*
* 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
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Sequential Grid Path Cover
* Difficulty: Medium
* Tags: array
*
* 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
*/

class Solution {
public List<List<Integer>> findPath(int[][] grid, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sequential Grid Path Cover
```

```
Difficulty: Medium
Tags: array

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
"""

class Solution:
def findPath(self, grid: List[List[int]], k: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sequential Grid Path Cover
* Difficulty: Medium
* Tags: array
*
* 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
*/

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

TypeScript Solution:

```
/**
* Problem: Sequential Grid Path Cover
* Difficulty: Medium
* Tags: array
*
* 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
*/

function findPath(grid: number[][], k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Sequential Grid Path Cover
* Difficulty: Medium
* Tags: array
*
* 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
*/

public class Solution {
public IList<IList<int>> FindPath(int[][] grid, int k) {

}
}
```

C Solution:

```
/*
* Problem: Sequential Grid Path Cover
* Difficulty: Medium
* Tags: array
*
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume caller calls free().
*/
int** findPath(int** grid, int gridSize, int* gridColSize, int k, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Sequential Grid Path Cover
// Difficulty: Medium
// Tags: array
//
// 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

func findPath(grid [][]int, k int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sequential Grid Path Cover
// Difficulty: Medium
// Tags: array
//
// 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

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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