Problem 885: Spiral Matrix III

Problem Information

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

Problem Description

You start at the cell

(rStart, cStart)

of an

rows x cols

grid facing east. The northwest corner is at the first row and column in the grid, and the southeast corner is at the last row and column.

You will walk in a clockwise spiral shape to visit every position in this grid. Whenever you move outside the grid's boundary, we continue our walk outside the grid (but may return to the grid boundary later.). Eventually, we reach all

rows * cols

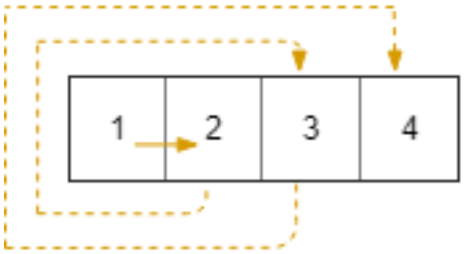
spaces of the grid.

Return

an array of coordinates representing the positions of the grid in the order you visited them

.

Example 1:



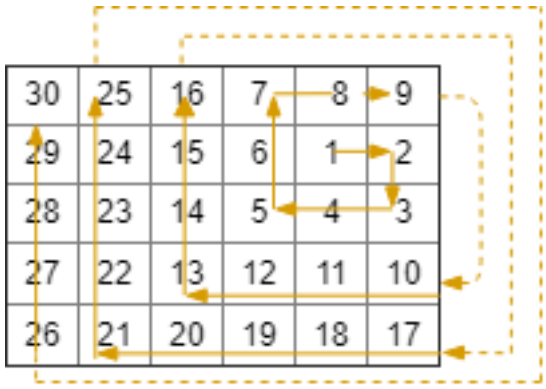
Input:

rows = 1, cols = 4, rStart = 0, cStart = 0

Output:

[[0,0],[0,1],[0,2],[0,3]]

Example 2:



Input:

rows = 5, cols = 6, rStart = 1, cStart = 4

Output:

[[1,4],[1,5],[2,5],[2,4],[2,3],[1,3],[0,3],[0,4],[0,5],[3,5],[3,4],[3,3],[3,2],[2,2],[1,2],[0,2],[4,5],[4,4],[4,3],[4,2],[4,1],[3,1],[2,1],[1,1],[0,1],[4,0],[3,0],[2,0],[1,0],[0,0]]

Constraints:

1 <= rows, cols <= 100

0 <= rStart < rows

0 <= cStart < cols

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> spiralMatrixIII(int rows, int cols, int rStart, int
cStart) {

}
};
```

Java:

```
class Solution {
public int[][] spiralMatrixIII(int rows, int cols, int rStart, int cStart) {

}
}
```

Python3:

```
class Solution:
def spiralMatrixIII(self, rows: int, cols: int, rStart: int, cStart: int) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def spiralMatrixIII(self, rows, cols, rStart, cStart):
"""
:type rows: int
:type cols: int
:type rStart: int
:type cStart: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number} rows
* @param {number} cols
* @param {number} rStart
* @param {number} cStart
* @return {number[][]}
*/
var spiralMatrixIII = function(rows, cols, rStart, cStart) {

};
```

TypeScript:

```
function spiralMatrixIII(rows: number, cols: number, rStart: number, cStart:
number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] SpiralMatrixIII(int rows, int cols, int rStart, int cStart) {

}
}
```

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** spiralMatrixIII(int rows, int cols, int rStart, int cStart, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func spiralMatrixIII(rows int, cols int, rStart int, cStart int) [][]int {

}
```

Kotlin:

```
class Solution {
fun spiralMatrixIII(rows: Int, cols: Int, rStart: Int, cStart: Int):
Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func spiralMatrixIII(_ rows: Int, _ cols: Int, _ rStart: Int, _ cStart: Int)
-> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn spiral_matrix_iii(rows: i32, cols: i32, r_start: i32, c_start: i32) ->
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} rows
# @param {Integer} cols
# @param {Integer} r_start
# @param {Integer} c_start
# @return {Integer[][]}
def spiral_matrix_iii(rows, cols, r_start, c_start)

end
```

PHP:

```
class Solution {

/**
* @param Integer $rows
* @param Integer $cols
* @param Integer $rStart
* @param Integer $cStart
* @return Integer[][]
*/
function spiralMatrixIII($rows, $cols, $rStart, $cStart) {

}
}
```

Dart:

```
class Solution {
List<List<int>> spiralMatrixIII(int rows, int cols, int rStart, int cStart) {

}
}
```

Scala:

```
object Solution {
def spiralMatrixIII(rows: Int, cols: Int, rStart: Int, cStart: Int):
Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec spiral_matrix_iii(rows :: integer, cols :: integer, r_start :: integer,
c_start :: integer) :: [[integer]]
def spiral_matrix_iii(rows, cols, r_start, c_start) do

end
end
```

Erlang:

```
-spec spiral_matrix_iii(Rows :: integer(), Cols :: integer(), RStart ::
integer(), CStart :: integer()) -> [[integer()]].
spiral_matrix_iii(Rows, Cols, RStart, CStart) ->
.
```

Racket:

```
(define/contract (spiral-matrix-iii rows cols rStart cStart)
(-> exact-integer? exact-integer? exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Spiral Matrix III
* 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>> spiralMatrixIII(int rows, int cols, int rStart, int
cStart) {

}
};
```

Java Solution:

```
/**
* Problem: Spiral Matrix III
* 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 int[][] spiralMatrixIII(int rows, int cols, int rStart, int cStart) {

}
}
```

Python3 Solution:

```
"""
Problem: Spiral Matrix III
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 spiralMatrixIII(self, rows: int, cols: int, rStart: int, cStart: int) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def spiralMatrixIII(self, rows, cols, rStart, cStart):
"""
:type rows: int
:type cols: int
:type rStart: int
:type cStart: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Spiral Matrix III
* 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} rows
* @param {number} cols
* @param {number} rStart
* @param {number} cStart
* @return {number[][]}
*/
var spiralMatrixIII = function(rows, cols, rStart, cStart) {

};
```

TypeScript Solution:

```
/**
* Problem: Spiral Matrix III
* 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 spiralMatrixIII(rows: number, cols: number, rStart: number, cStart:
number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Spiral Matrix III
* 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 int[][] SpiralMatrixIII(int rows, int cols, int rStart, int cStart) {

}
}
```

C Solution:

```
/*
* Problem: Spiral Matrix III
* 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** spiralMatrixIII(int rows, int cols, int rStart, int cStart, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Spiral Matrix III
// 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 spiralMatrixIII(rows int, cols int, rStart int, cStart int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun spiralMatrixIII(rows: Int, cols: Int, rStart: Int, cStart: Int):
Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func spiralMatrixIII(_ rows: Int, _ cols: Int, _ rStart: Int, _ cStart: Int)
-> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Spiral Matrix III
// 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 spiral_matrix_iii(rows: i32, cols: i32, r_start: i32, c_start: i32) ->
```

```
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} rows
# @param {Integer} cols
# @param {Integer} r_start
# @param {Integer} c_start
# @return {Integer[][]}
def spiral_matrix_iii(rows, cols, r_start, c_start)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $rows
* @param Integer $cols
* @param Integer $rStart
* @param Integer $cStart
* @return Integer[][]
*/
function spiralMatrixIII($rows, $cols, $rStart, $cStart) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> spiralMatrixIII(int rows, int cols, int rStart, int cStart) {

}
}
```

Scala Solution:

```
object Solution {
def spiralMatrixIII(rows: Int, cols: Int, rStart: Int, cStart: Int):
Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec spiral_matrix_iii(rows :: integer, cols :: integer, r_start :: integer,
c_start :: integer) :: [[integer]]
def spiral_matrix_iii(rows, cols, r_start, c_start) do

end
end
```

Erlang Solution:

```
-spec spiral_matrix_iii(Rows :: integer(), Cols :: integer(), RStart ::
integer(), CStart :: integer()) -> [[integer()]].
spiral_matrix_iii(Rows, Cols, RStart, CStart) ->
.
```

Racket Solution:

```
(define/contract (spiral-matrix-iii rows cols rStart cStart)
(-> exact-integer? exact-integer? exact-integer? exact-integer? (listof
(listof exact-integer?)))
)
```