Problem 54: Spiral Matrix

Problem Information

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

Problem Description

Given an

m x n

matrix

, return

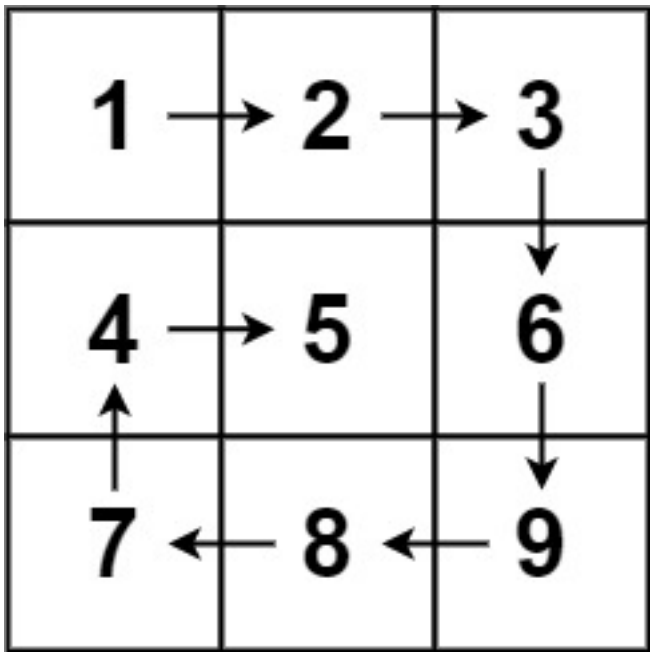
all elements of the

matrix

in spiral order

.

Example 1:



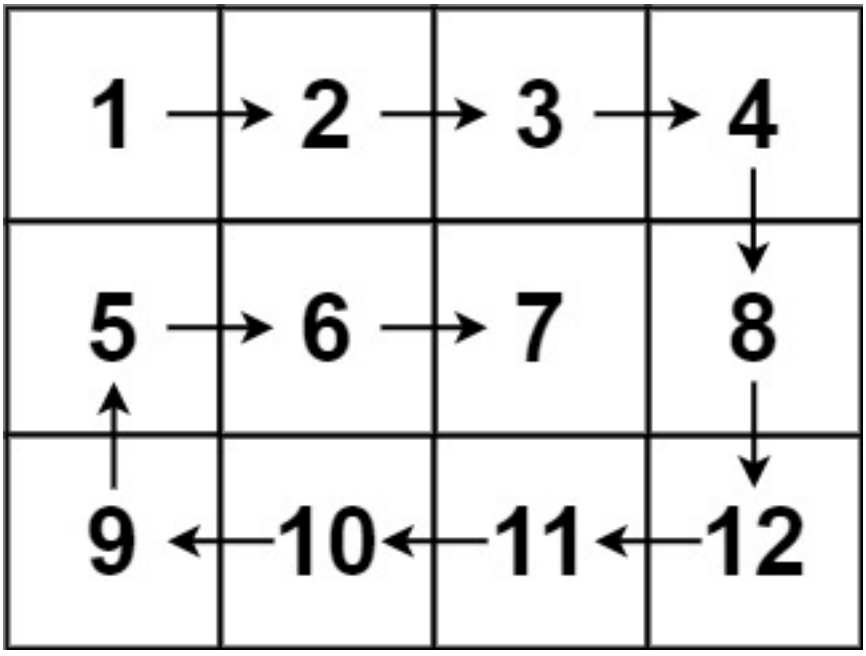
Input:

matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output:

[1,2,3,6,9,8,7,4,5]

Example 2:



Input:

matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

Output:

[1,2,3,4,8,12,11,10,9,5,6,7]

Constraints:

m == matrix.length

n == matrix[i].length

1 <= m, n <= 10

-100 <= matrix[i][j] <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<int> spiralOrder(vector<vector<int>>& matrix) {

}
};
```

Java:

```
class Solution {
public List<Integer> spiralOrder(int[][] matrix) {

}
}
```

Python3:

```
class Solution:
def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def spiralOrder(self, matrix):
"""
:type matrix: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} matrix
* @return {number[]}
*/
var spiralOrder = function(matrix) {

};
```

TypeScript:

```
function spiralOrder(matrix: number[][]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> SpiralOrder(int[][] matrix) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* spiralOrder(int** matrix, int matrixSize, int* matrixColSize, int*
returnSize) {
```

```
}
```

Go:

```
func spiralOrder(matrix [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun spiralOrder(matrix: Array<IntArray>): List<Int> {

}
}
```

Swift:

```
class Solution {
func spiralOrder(_ matrix: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn spiral_order(matrix: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} matrix
# @return {Integer[]}
def spiral_order(matrix)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer[]
*/
function spiralOrder($matrix) {

}
}
```

Dart:

```
class Solution {
List<int> spiralOrder(List<List<int>> matrix) {

}
}
```

Scala:

```
object Solution {
def spiralOrder(matrix: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec spiral_order(matrix :: [[integer]]) :: [integer]
def spiral_order(matrix) do

end
end
```

Erlang:

```
-spec spiral_order(Matrix :: [[integer()]]) -> [integer()].
spiral_order(Matrix) ->
.
```

Racket:

```
(define/contract (spiral-order matrix)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Spiral Matrix
* 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<int> spiralOrder(vector<vector<int>>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Spiral Matrix
* 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<Integer> spiralOrder(int[][] matrix) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Spiral Matrix
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 spiralOrder(self, matrix: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def spiralOrder(self, matrix):
"""
:type matrix: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Spiral Matrix
* 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[][]} matrix
* @return {number[]}
*/
var spiralOrder = function(matrix) {

};
```

TypeScript Solution:

```
/**
* Problem: Spiral Matrix
* 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 spiralOrder(matrix: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Spiral Matrix
* 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<int> SpiralOrder(int[][] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Spiral Matrix
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* spiralOrder(int** matrix, int matrixSize, int* matrixColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Spiral Matrix
// 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 spiralOrder(matrix [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun spiralOrder(matrix: Array<IntArray>): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func spiralOrder(_ matrix: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Spiral Matrix
// 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_order(matrix: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer[]}
def spiral_order(matrix)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer[]
*/
function spiralOrder($matrix) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> spiralOrder(List<List<int>> matrix) {

}
}
```

Scala Solution:

```
object Solution {
def spiralOrder(matrix: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec spiral_order(matrix :: [[integer]]) :: [integer]
def spiral_order(matrix) do

end
end
```

Erlang Solution:

```
-spec spiral_order(Matrix :: [[integer()]]) -> [integer()].
spiral_order(Matrix) ->
.
```

Racket Solution:

```
(define/contract (spiral-order matrix)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```