**QPREP4-Spiral matrix**

**Module Introduction**

Write a program to output the elements of a matrix in spiral order.

#### Objective

Given an input matrix of size m x n, output the elements of the matrix in spiral order. Leverage the functions you have created in the previous module if need be.

#### Examples

**Example 1**

Input:

1 2 3

4 5 6

7 8 9

Output:

1 2 3 6 9 8 7 4 5

***SOLUTION STEPS FROM NEXT PAGE:***

**Write down at least 3 examples in the following format. Kindly stick to the format.**

**Suggestion:**

EXAMPLE#1: m & n are not equal

INPUT:

2 4

1 2 3 4

5 6 7 8

OUTPUT:

1 2 3 4 8 7 6 5

EXAMPLE#2: Slightly larger m & n

INPUT:

4 4

3 4 5 6

6 5 4 2

1 2 9 8

6 3 8 7

OUTPUT:

3 4 5 6 2 8 7 8 3 6 1 6 5 4 9 2

EXAMPLE#3:

INPUT:

4 2

1 2

3 4

5 6

7 8

OUTPUT:

1 2 4 6 8 7 5 3

**Detail your problem understanding here**

**Suggestion:**

A matrix is provided as input. We have to traverse through the matrix in spiral manner i.e. clockwise and print the values that are seen along the way.

**Does this problem follow a known algorithmic pattern or standard application of a data structure? If there are multiple approaches, which one would you choose and why? Write down your chosen approach in 2-3 sentences like you would explain to a 10 year old.**

The problem involves manipulating indices in a 2D array and general matrix understanding.

Spiral traversal is achieved by first going RIGHT as far as possible, then DOWN as far as possible, then LEFT as far as possible and finally UP till you see a previously visited node. At this point, start going RIGHT again and repeat.

One can either store the visited nodes for a simpler solution using extra storage. Other option is to do some calculations based on the matrix dimensions, avoid revisiting a node.

Rows = 3; Column = 4

Except first time, reduce row or column by 1 and make steps in appropriate direction. Keep changing the direction in a cyclic manner

GoRight(4 times); // First time Move right (column) number of times => 4 times = newColumn Output: 1, 2, 3, 4

**1 2 3 4**

5 6 7 8

9 10 11 12

GoDown(2 times); // Move down (row-1) number of times => 2 times = newRow Output: 8, 12

1 2 3 4

5 6 7 **8**

9 10 11 **12**

GoLeft(3 times); // Move left (newColumn-1) number of times => 3 times = newColumn Output: 11, 10, 9

1 2 3 4

5 6 7 8

**9 10 11** 12

GoUp(1 time); // Move up (newRow-1) number of times => 1 time = newRow

Output: 5

1 2 3 4

**5** 6 7 8

9 10 11 12

TIP: Were you able to arrive at a similar logic? If you had taken an example, tried to solve the problem manually first, then you may have ended up with something similar

**Write the pseudocode here in plain English**

enum Direction {

RIGHT = 1,

DOWN = 2,

LEFT = 3,

UP = 4

}

Read the input matrix

spiralMatrix = []

Start from indices (-1, -1)

Start with currentDirection = RIGHT

horizontalSteps = columns

verticalSteps = rows - 1

while steps possible in currentDirection {

nodes = matrixMove(currentDirection, horizontalSteps or verticalSteps)

spiralMatrix.append(nodes)

Decrease horizontalSteps or verticalSteps appropriately.

currentDirection = changeDirectionClockwise()

}

return spiralMatrix

**Can you specify a few boundary or edge cases here?**

**Edge cases**

EXAMPLE#1: Matrix dimension is 0 \* 0

INPUT:

0 0

OUTPUT:

Empty (no output needed)

EXAMPLE#2: Zero rows in dimension

INPUT:

0 2

OUTPUT:

Empty (no output needed)

EXAMPLE#3: Matrix of 1 \* 1

INPUT:

1 1

10

OUTPUT:

10

**Write the functions you would create here**

matrixMove() from previous module

Direction changeDirectionClockwise(Direction)

#### Summary

Starting with a brief explanation of the problem statement followed by pseudocode and then implementing the solution helps you approach the problem in a systematic way. This methodology helps with easy as well as hard problems.

**Time Complexity: O(m \* n)**

We have to traverse through every element in the matrix which is m \* n

**Space Complexity: O(m \* n)**

If you are storing the visited nodes, you may require extra storage. You can also solve the problem without extra storage.

#### Concepts

Concepts covered in this Module

* 2D Array
* Matrix traversal

Similar problems

* <https://leetcode.com/problems/spiral-matrix-ii/>
* <https://leetcode.com/problems/spiral-matrix-iii/>

#### Good habits

Think about these for your solution:

* Comments - have you used comments in a way that others can understand this code?
* Test Cases - Are most of the scenarios/corner cases/boundary conditions handled in the solution?
* Naming Convention - Are the variables and functions named sensibly and with uniform convention?
* Modular Functions - Has the solution been addressed using concise functions? Will these functions work without any changes if they are to be used in another problem?
* Optimization - Analyze the Time Complexity and Space Complexity for your solution. Has the solution been optimized or did it use the brute force method? Is further optimization desirable/possible?
* Data Structures - Has the optimal/appropriate data structure been used?

SOLUTION:

APPROACH 1: solved. Can be more modular ofc.

import java.io.\*;

import java.util.\*;

class SpiralMatrix {

// Implement your solution by completing the below function

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

List<Integer> lst = new ArrayList<Integer>();

int i;

int rowStart = 0;

int colStart = 0;

int rowEnd = matrix.length;

int colEnd = matrix[0].length;

/\* k - starting row index

m - ending row index

l - starting column index

n - ending column index

i - iterator

\*/

while (rowStart < rowEnd && colStart < colEnd) {

// Print the first row from the remaining rows

for (i = colStart; i < colEnd; ++i) {

lst.add(matrix[rowStart][i]);

}

rowStart++;

// Print the last column from the remaining columns

for (i = rowStart; i < rowEnd; ++i) {

lst.add(matrix[i][colEnd - 1]);

}

colEnd--;

// Print the last row from the remaining rows \*/

if (rowStart < rowEnd) {

for (i = colEnd - 1; i >= colStart; --i) {

lst.add(matrix[rowEnd - 1][i]);

}

rowEnd--;

}

// Print the first column from the remaining columns \*/

if (colStart < colEnd) {

for (i = rowEnd - 1; i >= rowStart; --i) {

lst.add(matrix[i][colStart]);

}

colStart++;

}

}

return lst;

}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int m = scanner.nextInt();

int[][] matrix = new int[n][m];

for (int i = 0; i < n; ++i)

for (int j = 0; j < m; ++j)

matrix[i][j] = scanner.nextInt();

scanner.close();

List<Integer> result = new SpiralMatrix().spiralOrder(matrix);

for (int i = 0; i < result.size(); ++i)

System.out.printf("%d ", result.get(i));

}

}

**Complexity Analysis:**

* **Time Complexity:**
* **Space Complexity:**