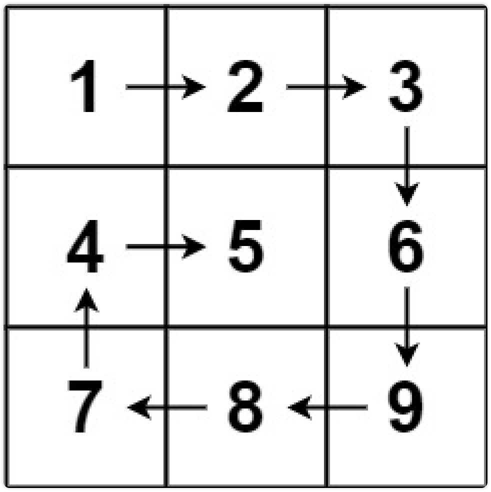
Day 16

Prepared By Azan Imtiaz

Spiral Matrix (Leatcode)

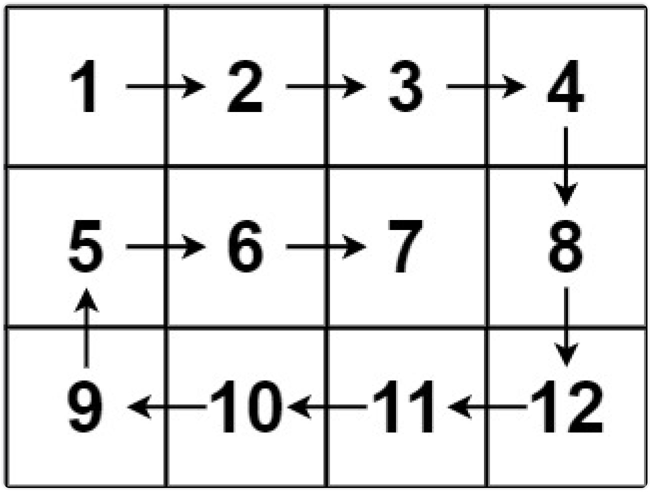
Given an `m x n` `matrix`, return *all elements of the* `matrix` *in spiral order*.

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`

Intuition

The goal is to traverse a 2D matrix in a spiral order, starting from the top-left corner and moving right, down, left, and up successively. The challenge lies in keeping track of the boundaries as we continuously peel off the outer layers of the matrix.

Approach

1. Initialize four pointers to keep track of the current boundaries (`rowBegin`, `rowEnd`, `colBegin`, `colEnd`).
2. Loop through the matrix while there are still elements within the boundaries.
3. Traverse right across the top row and increment `rowBegin`.
4. Traverse down the last column and decrement `colEnd`.
5. If there are rows remaining, traverse left across the bottom row and decrement `rowEnd`.
6. If there are columns remaining, traverse up the first column and increment `colBegin`.
7. Repeat steps 3-6 until all elements have been visited.

Solution

```
import java.util.List;
import java.util.ArrayList;

class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> arr = new ArrayList<>();
        int rowBegin = 0;
        int rowEnd = matrix.length - 1;
        int colBegin = 0;
        int colEnd = matrix[0].length - 1;
        while (rowBegin <= rowEnd && colBegin <= colEnd) {
            // Traverse Right
            for (int i = colBegin; i <= colEnd; i++) {
                arr.add(matrix[rowBegin][i]);
            }
            rowBegin++;

            // Traverse Down
            for (int i = rowBegin; i <= rowEnd; i++) {
                arr.add(matrix[i][colEnd]);
            }
            colEnd--;

            // Traverse Left
            if (rowBegin <= rowEnd) {
                for (int i = colEnd; i >= colBegin; i--) {
                    arr.add(matrix[rowEnd][i]);
                }
            }
            rowEnd--;

            // Traverse Up
            if (colBegin <= colEnd) {
                for (int i = rowEnd; i >= rowBegin; i--) {
                    arr.add(matrix[i][colBegin]);
                }
                colBegin++;
            }
        }
        return arr;
    }
}
```

Explanation

The code defines a method `spiralOrder` that takes a 2D integer array `matrix` and returns a list of integers representing the elements of the matrix in spiral order. It uses a while loop to iterate over the matrix within the defined boundaries, adjusting the boundaries after each traversal to ensure that each element is visited exactly once.

Time Complexity

The time complexity is O(N), where N is the total number of elements in the matrix. Each element is visited exactly once.

Dry Run

Test Case 1:

Matrix:

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

Dry Run Steps:

- Initialize `rowBegin = 0`, `rowEnd = 2`, `colBegin = 0`, `colEnd = 2`.
- Traverse Right from `colBegin` to `colEnd`: Add `1, 2, 3` to the list.
- Increment `rowBegin` to `1`.
- Traverse Down from `rowBegin` to `rowEnd`: Add `6, 9` to the list.
- Decrement `colEnd` to `1`.
- Check if `rowBegin <= rowEnd` (1 <= 1): True
  - Traverse Left from `colEnd` to `colBegin`: Add `8, 7` to the list.
- Decrement `rowEnd` to `0`.
- Check if `colBegin <= colEnd` (0 <= 1): True
  - Traverse Up from `rowEnd` to `rowBegin`: Add `4` to the list.
- Increment `colBegin` to `1`.
- Since `rowBegin <= rowEnd` and `colBegin <= colEnd` are no longer true, we stop.

Output:
`[1, 2, 3, 6, 9, 8, 7, 4]`

Test Case 2:

Matrix:

```
[
  [1],
  [2],
  [3]
]
```

Dry Run Steps:

- Initialize `rowBegin = 0`, `rowEnd = 2`, `colBegin = 0`, `colEnd = 0`.
- Traverse Right from `colBegin` to `colEnd`: Add `1` to the list.
- Increment `rowBegin` to `1`.
- Traverse Down from `rowBegin` to `rowEnd`: Add `2, 3` to the list.
- Decrement `colEnd` to `-1`.
- Since `colEnd` is now less than `colBegin`, the remaining steps are skipped.

Output:
`[1, 2, 3]`

Test Case 3:

Matrix:

```
[
  [1, 2, 3, 4]
]
```

Dry Run Steps:

- Initialize `rowBegin = 0`, `rowEnd = 0`, `colBegin = 0`, `colEnd = 3`.
- Traverse Right from `colBegin` to `colEnd`: Add `1, 2, 3, 4` to the list.
- Increment `rowBegin` to `1`.
- Since `rowBegin` is now greater than `rowEnd`, the remaining steps are skipped.

Output:

```
[1, 2, 3, 4]
```

In each test case, the algorithm follows a spiral path, adjusting the boundaries after each traversal. The conditions within the loop ensure that the traversal only occurs if there are elements left in the current direction.

Thank You Every one For Reading