# Array Traverse

Traverse the array can be one independent category in Array problem. Normally this happens in a 2D array, a matrix. The common trick is that we should remember current positions and traverse in an organized way.

## 48. Rotate Image

Medium

You are given an *n* x *n* 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

**Note:**

You have to rotate the image [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm), which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

**Example 1:**

Given **input matrix** =

[

[1,2,3],

[4,5,6],

[7,8,9]

],

rotate the input matrix **in-place** such that it becomes:

[

[7,4,1],

[8,5,2],

[9,6,3]

]

**Example 2:**

Given **input matrix** =

[

[ 5, 1, 9,11],

[ 2, 4, 8,10],

[13, 3, 6, 7],

[15,14,12,16]

],

rotate the input matrix **in-place** such that it becomes:

[

[15,13, 2, 5],

[14, 3, 4, 1],

[12, 6, 8, 9],

[16, 7,10,11]

]

### Analysis:

The difficulty part is how to traverse the matrix in order. Assume you start from four side with start row, end row, start column and end column and push the 4 side to the center until they all same. You always swap the start row to other 3 sides.

/// <summary>

/// Leet code #48. Rotate Image

///

/// You are given an n x n 2D matrix representing an image.

///

/// Rotate the image by 90 degrees (clockwise).

///

/// Note:

///

/// You have to rotate the image in-place, which means you have to

/// modify the input 2D matrix directly. DO NOT allocate another 2D

/// matrix and do the rotation.

///

/// Example 1:

///

/// Given input matrix =

/// [

/// [1,2,3],

/// [4,5,6],

/// [7,8,9]

/// ],

///

/// rotate the input matrix in-place such that it becomes:

/// [

/// [7,4,1],

/// [8,5,2],

/// [9,6,3]

/// ]

///

/// Example 2:

///

/// Given input matrix =

/// [

/// [ 5, 1, 9,11],

/// [ 2, 4, 8,10],

/// [13, 3, 6, 7],

/// [15,14,12,16]

/// ],

///

/// rotate the input matrix in-place such that it becomes:

/// [

/// [15,13, 2, 5],

/// [14, 3, 4, 1],

// [12, 6, 8, 9],

/// [16, 7,10,11]

/// ]

/// </summary>

void LeetCodeArray::rotate(vector<vector<int>>& matrix)

{

if (matrix.empty() || matrix[0].empty()) return;

int begin\_row = 0;

int end\_row = matrix.size() - 1;

int begin\_col = 0;

int end\_col = matrix[0].size() - 1;

while ((begin\_row <= end\_row) && (begin\_col <= end\_col))

{

for (int i = 0; i < (end\_col - begin\_col); i++)

{

swap(matrix[begin\_row][begin\_col + i], matrix[begin\_row + i][end\_col]);

swap(matrix[begin\_row][begin\_col + i], matrix[end\_row][end\_col - i]);

swap(matrix[begin\_row][begin\_col + i], matrix[end\_row - i][begin\_col]);

}

begin\_row++;

end\_row--;

begin\_col++;

end\_col--;

}

}

## 54. Spiral Matrix

Medium

Given a matrix of *m* x *n* elements (*m* rows, *n* columns), return all elements of the matrix in spiral order.

**Example 1:**

**Input:**

[

[ 1, 2, 3 ],

[ 4, 5, 6 ],

[ 7, 8, 9 ]

]

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

**Example 2:**

**Input:**

[

[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]

### Analysis:

We can use the same method as above by using start row, end row, start column and end column and push the 4 side to the center until they all same.

/// <summary>

/// LeetCode #54. Spiral Matrix

///

/// Given a matrix of m x n elements (m rows, n columns), return all elements

/// of the matrix in spiral order.

///

/// Example 1:

///

/// Input:

/// [

/// [ 1, 2, 3 ],

/// [ 4, 5, 6 ],

/// [ 7, 8, 9 ]

/// ]

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

///

/// Example 2:

///

/// Input:

/// [

/// [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]

/// You are given an n x n 2D matrix representing an image.

/// </summary>

vector<int> LeetCodeArray::spiralOrder(vector<vector<int>>& matrix)

{

vector<int> result;

if (matrix.empty() || matrix[0].empty()) return result;

int begin\_row = 0;

int end\_row = matrix.size() - 1;

int begin\_col = 0;

int end\_col = matrix[0].size() - 1;

int direction = 0;

while ((begin\_row <= end\_row) && (begin\_col <= end\_col))

{

switch (direction)

{

case 0:

for (int i = begin\_col; i <= end\_col; i++)

{

result.push\_back(matrix[begin\_row][i]);

}

begin\_row++;

break;

case 1:

for (int i = begin\_row; i <= end\_row; i++)

{

result.push\_back(matrix[i][end\_col]);

}

end\_col--;

break;

case 2:

for (int i = end\_col; i >= begin\_col; i--)

{

result.push\_back(matrix[end\_row][i]);

}

end\_row--;

break;

case 3:

for (int i = end\_row; i >= begin\_row; i--)

{

result.push\_back(matrix[i][begin\_col]);

}

begin\_col++;

break;

}

direction = (direction + 1) % 4;

}

return result;

}

## 59. Spiral Matrix II

Medium

Given a positive integer *n*, generate a square matrix filled with elements from 1 to *n*2 in spiral order.

**Example:**

**Input:** 3

**Output:**

[

[ 1, 2, 3 ],

[ 8, 9, 4 ],

[ 7, 6, 5 ]

]

### Analysis:

Almost same as LeetCode 54.

/// <summary>

/// Leet code #59. Spiral Matrix II

///

/// Medium

///

/// Given a positive integer n, generate a square matrix filled with elements

/// from 1 to n2 in spiral order.

///

/// Example:

///

/// Input: 3

/// Output:

/// [

/// [ 1, 2, 3 ],

/// [ 8, 9, 4 ],

/// [ 7, 6, 5 ]

/// ]

/// </summary>

vector<vector<int>> LeetCodeArray::generateMatrix(int n)

{

vector<vector<int>> result(n, vector<int>(n, 0));

if (n <= 0) return result;

int begin\_row = 0;

int end\_row = n - 1;

int begin\_col = 0;

int end\_col = n - 1;

int direction = 0;

int index = 0;

while ((begin\_row <= end\_row) && (begin\_col <= end\_col))

{

switch (direction)

{

case 0:

for (int i = begin\_col; i <= end\_col; i++)

{

index++;

result[begin\_row][i] = index;

}

begin\_row++;

break;

case 1:

for (int i = begin\_row; i <= end\_row; i++)

{

index++;

result[i][end\_col] = index;

}

end\_col--;

break;

case 2:

for (int i = end\_col; i >= begin\_col; i--)

{

index++;

result[end\_row][i] = index;

}

end\_row--;

break;

case 3:

for (int i = end\_row; i >= begin\_row; i--)

{

index++;

result[i][begin\_col] = index;

}

begin\_col++;

break;

}

direction = (direction + 1) % 4;

}

return result;

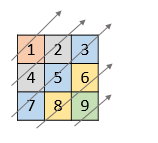
}

## 1424. Diagonal Traverse II

Medium

Given a list of lists of integers, nums, return all elements of nums in diagonal order as shown in the below images.

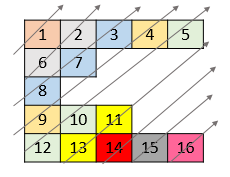
**Example 1:**

****

**Input:** nums = [[1,2,3],[4,5,6],[7,8,9]]

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

**Example 2:**

****

**Input:** nums = [[1,2,3,4,5],[6,7],[8],[9,10,11],[12,13,14,15,16]]

**Output:** [1,6,2,8,7,3,9,4,12,10,5,13,11,14,15,16]

**Example 3:**

**Input:** nums = [[1,2,3],[4],[5,6,7],[8],[9,10,11]]

**Output:** [1,4,2,5,3,8,6,9,7,10,11]

**Example 4:**

**Input:** nums = [[1,2,3,4,5,6]]

**Output:** [1,2,3,4,5,6]

**Constraints:**

* 1 <= nums.length <= 10^5
* 1 <= nums[i].length <= 10^5
* 1 <= nums[i][j] <= 10^9
* There at most 10^5 elements in nums.

### Analysis:

This is an continuous array, every time you move down one row, starts with column 0 and move up one row one step, adding column by one, if that row has enough columns on specific column then output otherwise skip it.

However there is an issue here, the matrix is very big, so we cannot waste our time on non-existing cells, in this case there are two ways to do so, one is a little bit easy, you iterate original array, and store all the cells by the row+col in first come last out manner in a new matrix, and output that matrix, but it will cause extra memory. Another way is that you record all rows with remaining columns not iterated and every time you revisit the rows.

/// <summary>

/// Leet code #1424. Diagonal Traverse II

///

/// Medium

///

/// Given a list of lists of integers, nums, return all elements of nums

/// in diagonal order as shown in the below images.

///

/// Example 1:

/// Input: nums = [[1,2,3],[4,5,6],[7,8,9]]

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

///

/// Example 2:

/// Input: nums = [[1,2,3,4,5],[6,7],[8],[9,10,11],[12,13,14,15,16]]

/// Output: [1,6,2,8,7,3,9,4,12,10,5,13,11,14,15,16]

///

/// Example 3:

/// Input: nums = [[1,2,3],[4],[5,6,7],[8],[9,10,11]]

/// Output: [1,4,2,5,3,8,6,9,7,10,11]

///

/// Example 4:

/// Input: nums = [[1,2,3,4,5,6]]

/// Output: [1,2,3,4,5,6]

///

/// Constraints:

/// 1. 1 <= nums.length <= 10^5

/// 2. 1 <= nums[i].length <= 10^5

/// 3. 1 <= nums[i][j] <= 10^9

/// 4. There at most 10^5 elements in nums.

/// </summary>

vector<int> LeetCodeArray::findDiagonalOrder(vector<vector<int>>& nums)

{

map<int, int> index\_map;

for (int i = 0; i < (int)nums.size(); i++) index\_map[0 - i] = 0;

int index = 0;

vector<int> result;

while (!index\_map.empty())

{

auto pos = index\_map.lower\_bound(index);

while (pos != index\_map.end())

{

int row = 0 - pos->first;

result.push\_back(nums[row][pos->second]);

pos->second++;

auto temp = pos;

pos++;

if (temp->second >= (int)nums[row].size()) index\_map.erase(temp);

}

index--;

}

return result;

}