# Array Manipulation

In this paper, I am going to talk about array manipulation, there may not be a common pattern for all of them, instead each one has its own characteristic, but it is quite fun to discuss them.

## 41. First Missing Positive

Hard

Given an unsorted integer array, find the smallest missing positive integer.

**Example 1:**

Input: [1,2,0]

Output: 3

**Example 2:**

Input: [3,4,-1,1]

Output: 2

**Example 3:**

Input: [7,8,9,11,12]

Output: 1

**Note:**

Your algorithm should run in *O*(*n*) time and uses constant extra space.

### Analysis:

Consider the array is row of room with index + 1 as room number, you can start from first room, if it is a valid number within 1 to n, drag it to its own room and take the number in that room out. You also need to remember the least unvisited room, so in case when we have a duplicate or loop, you can start a new search loop.

/// <summary>

/// Leet code #41. First Missing Positive

///

/// Given an unsorted integer array, find the smallest missing positive

/// integer.

///

/// Example 1:

///

/// Input: [1,2,0]

/// Output: 3

///

/// Example 2:

///

/// Input: [3,4,-1,1]

/// Output: 2

///

/// Example 3:

///

/// Input: [7,8,9,11,12]

/// Output: 1

///

/// Note:

///

/// Your algorithm should run in O(n) time and uses constant extra space.

/// </summary>

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

{

if (nums.size() == 0)

{

return 1;

}

size\_t index = 0;

while (index < nums.size())

{

// non-positive or out of range, skip it.

if ((nums[index] <= 0) || (nums[index] >= (int)nums.size()))

{

index++;

}

// already in order, skip it

else if (nums[index] == index + 1)

{

index++;

}

// already same data so no need to swap

else if (nums[index] == nums[nums[index] - 1])

{

index++;

}

else

{

swap(nums[index], nums[nums[index] - 1]);

}

}

for (size\_t i = 0; i < nums.size(); i++)

{

if (nums[i] != i + 1)

{

return i + 1;

}

}

return nums.size() + 1;

}

## 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;

}

## 169. Majority Element

Easy

Given an array of size *n*, find the majority element. The majority element is the element that appears **more than** ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

**Example 1:**

**Input:** [3,2,3]

**Output:** 3

**Example 2:**

**Input:** [2,2,1,1,1,2,2]

**Output:** 2

### Analysis:

This is a cancelling problem, each number should be cancelled by a different number and the one left is the answer.

/// <summary>

/// Leet code #169. Majority Element

///

/// Given an array of size n, find the majority element. The majority element

/// is the element that appears more than n/2 times.

///

/// You may assume that the array is non-empty and the majority element

/// always exist in the array.

///

/// Example 1:

///

/// Input: [3,2,3]

/// Output: 3

///

/// Example 2:

///

/// Input: [2,2,1,1,1,2,2]

/// Output: 2

/// </summary>

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

{

int count = 0;

int major\_number = 0;

for (size\_t i = 0; i < nums.size(); i++)

{

if (count == 0)

{

major\_number = nums[i];

count++;

}

else

{

if (major\_number == nums[i])

{

count++;

}

else

{

count--;

}

}

}

return major\_number;

}

## 229. Majority Element II

Medium

Given an integer array of size *n*, find all elements that appear more than ⌊ n/3 ⌋ times.

**Note:**The algorithm should run in linear time and in O(1) space.

**Example 1:**

**Input:** [3,2,3]

**Output:** [3]

**Example 2:**

**Input:** [1,1,1,3,3,2,2,2]

**Output:** [1,2]

### Analysis:

This is a cancelling problem, we pick any 3 numbers, if there is no duplication we discard all of them, if there is a duplication, we add the count and move the window ahead and pick a next one number. So after all the round, we end up with most 2 popular numbers, then we validate which one is the one over 1/3. The theory is that every number can be cancelled together with another two different numbers, if the number is over 1/3 it should survive in the final list.

/// <summary>

/// Leet code #229. Majority Element II

///

/// Given an integer array of size n, find all elements that appear more than

/// n/3 times.

///

/// Note: The algorithm should run in linear time and in O(1) space.

///

/// Example 1:

///

/// Input: [3,2,3]

/// Output: [3]

///

/// Example 2:

///

/// Input: [1,1,1,3,3,2,2,2]

/// Output: [1,2]

/// </summary>

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

{

vector<int> result;

if (nums.size() < 3) return nums;

int count1 = 1, count2 = 1;

int number1 = nums[0], number2 = nums[1];

for (size\_t i = 2; i < nums.size(); i++)

{

if (number1 == nums[i])

{

count1++;

}

else if (number2 == nums[i])

{

count2++;

}

else if (count1 == 0)

{

number1 = nums[i];

count1++;

}

else if (count2 == 0)

{

number2 = nums[i];

count2++;

}

else

{

count1--;

count2--;

}

}

count1 = count(nums.begin(), nums.end(), number1);

count2 = count(nums.begin(), nums.end(), number2);

if (count1 > (int)nums.size() / 3)

{

result.push\_back(number1);

}

if (count2 > (int)nums.size() / 3)

{

if (number2 != number1) result.push\_back(number2);

}

return result;

}

## 649. Dota2 Senate

Medium

In the world of Dota2, there are two parties: the Radiant and the Dire.

The Dota2 senate consists of senators coming from two parties. Now the senate wants to make a decision about a change in the Dota2 game. The voting for this change is a round-based procedure. In each round, each senator can exercise one of the two rights:

1. Ban one senator's right:  
   A senator can make another senator lose **all his rights** in this and all the following rounds.
2. Announce the victory:  
   If this senator found the senators who still have rights to vote are all from **the same party**, he can announce the victory and make the decision about the change in the game.

Given a string representing each senator's party belonging. The character 'R' and 'D' represent the Radiant party and the Dire party respectively. Then if there are n senators, the size of the given string will be n.

The round-based procedure starts from the first senator to the last senator in the given order. This procedure will last until the end of voting. All the senators who have lost their rights will be skipped during the procedure.

Suppose every senator is smart enough and will play the best strategy for his own party, you need to predict which party will finally announce the victory and make the change in the Dota2 game. The output should be Radiant or Dire.

**Example 1:**

**Input:** "RD"

**Output:** "Radiant"

**Explanation:** The first senator comes from Radiant and he can just ban the next senator's right in the round 1.

And the second senator can't exercise any rights any more since his right has been banned.

And in the round 2, the first senator can just announce the victory since he is the only guy in the senate who can vote.

**Example 2:**

**Input:** "RDD"

**Output:** "Dire"

**Explanation:**

The first senator comes from Radiant and he can just ban the next senator's right in the round 1.

And the second senator can't exercise any rights anymore since his right has been banned.

And the third senator comes from Dire and he can ban the first senator's right in the round 1.

And in the round 2, the third senator can just announce the victory since he is the only guy in the senate who can vote.

**Note:**

1. The length of the given string will in the range [1, 10,000].

### Analysis:

This is another cancelling problem, each senator, when come to speak, first he should check if he is banned on the current party, if yes, he walk down and deduct ban count, if he has the right then he will simply add the ban count for the opposite party. When no one is banned in one round we are final.

/// <summary>

/// Leet code #649. Dota2 Senate

///

/// In the world of Dota2, there are two parties: the Radiant and the Dire.

/// The Dota2 senate consists of senators coming from two parties. Now the

/// senate wants to make a decision about a change in the Dota2 game. The

/// voting for this change is a round-based procedure. In each round, each

/// senator can exercise one of the two rights:

/// 1. Ban one senator's right:

/// A senator can make another senator lose all his rights in this and

/// all the following rounds.

/// 2. Announce the victory:

/// If this senator found the senators who still have rights to vote are

/// all from the same party, he can announce the victory and make the

/// decision about the change in the game.

///

/// Given a string representing each senator's party belonging. The

/// character 'R' and 'D' represent the Radiant party and the Dire party

/// respectively. Then if there are n senators, the size of the given

/// string will be n.

/// The round-based procedure starts from the first senator to the last

/// senator in the given order. This procedure will last until the end of

/// voting. All the senators who have lost their rights will be skipped

/// during the procedure.

/// Suppose every senator is smart enough and will play the best strategy

/// for his own party, you need to predict which party will finally

/// announce the victory and make the change in the Dota2 game. The output

/// should be Radiant or Dire.

/// Example 1:

/// Input: "RD"

/// Output: "Radiant"

/// Explanation: The first senator comes from Radiant and he can just ban

/// the next senator's right in the round 1.

/// And the second senator can't exercise any rights any more since his

/// right has been banned.

/// And in the round 2, the first senator can just announce the victory

/// since he is the only guy in the senate who can vote.

///

/// Example 2:

/// Input: "RDD"

/// Output: "Dire"

/// Explanation:

/// The first senator comes from Radiant and he can just ban the next

/// senator's right in the round 1.

/// And the second senator can't exercise any rights anymore since his

/// right has been banned.

/// And the third senator comes from Dire and he can ban the first

/// senator's right in the round 1.

/// And in the round 2, the third senator can just announce the victory

/// since he is the only guy in the senate who can vote.

///

/// Note:

/// The length of the given string will in the range [1, 10,000].

/// </summary>

string LeetCode::predictPartyVictory(string senate)

{

vector<int> count\_map(2);

while (true)

{

string temp;

for (auto ch : senate)

{

int index = (ch == 'R' ? 0 : 1);

if (count\_map[index] > 0)

{

count\_map[index]--;

}

else

{

temp.push\_back(ch);

count\_map[1 - index]++;

}

}

if (temp.size() == senate.size()) break;

senate = temp;

}

if (senate[0] == 'D') return "Dire";

else return "Radiant";

}

## 287. Find the Duplicate Number

Medium

Given an array *nums* containing *n* + 1 integers where each integer is between 1 and *n* (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

**Example 1:**

**Input:** [1,3,4,2,2]

**Output:** 2

**Example 2:**

**Input:** [3,1,3,4,2]

**Output:** 3

**Note:**

1. You **must not** modify the array (assume the array is read only).
2. You must use only constant, *O*(1) extra space.
3. Your runtime complexity should be less than *O*(*n*2).
4. There is only one duplicate number in the array, but it could be repeated more than once.

### Analysis:

First this is not a medium problem. It is definitely a hard one if you have never seen it. This took Donald Knuth more than 24 hours to find the solution. The solution is this if you think the array as a linked list and every number is a address point to the next, a duplication means the loop in the whole traverse path. In this case you run the fast and slow runner will catch the duplicate (they meet at the duplicate).

/// <summary>

/// Leet code #287. Find the Duplicate Number

///

/// Given an array nums containing n + 1 integers where each integer is

/// between 1 and n (inclusive), prove that at least one duplicate number

/// must exist. Assume that there is only one duplicate number, find the

/// duplicate one.

///

/// Example 1:

///

/// Input: [1,3,4,2,2]

/// Output: 2

///

/// Example 2:

///

/// Input: [3,1,3,4,2]

/// Output: 3

///

/// Note:

///

/// 1. You must not modify the array (assume the array is read only).

/// 2. You must use only constant, O(1) extra space.

/// 3. Your runtime complexity should be less than O(n2).

/// 4. There is only one duplicate number in the array, but it could be

/// repeated more than once.

/// </summary>

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

{

// protect empty array

if (nums.size() == 0) return -1;

// tortoise

int slow = 0;

int fast = 0;

while (true)

{

slow = nums[slow];

fast = nums[nums[fast]];

if (slow == fast) break;

}

fast = 0;

while (true)

{

slow = nums[slow];

fast = nums[fast];

if (slow == fast) break;

}

return slow;

}