LeecodeNoteFor2023

drive link: https://drive.google.com/drive/folders/1e4Lboc4y1uaDwOWHhmAIXtQHzPPXAJVB

task link: https://ke.qq.com/course/package/31104?flowToken=1039500

Learning Time: 2:00-6:00 7:30-10:30 7h

数组字符串

- question: input/output基础 link: https://www.nowcoder.com/test/27976983/summary#question
- ACM模式下Java代码格式

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
      //从控制台输入
      Scanner scanner = new Scanner(System.in);
      String str = scanner.nextLine();
      String res = "";
      System.out.Println(res);
    }
  }
```

- ACM模式下JavaScript(V8)代码格式

```
//从控制台输入
const str = readline();
let res = str
print(res);//或使用console.log(res)
```

- ACM模式下JavaScript(Node)代码格式
- 单行

```
const readline = require('readline')

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
})

rl.on('line', function(line) {
      let res = line
      console.log(res)
})
```

- 多行输入

```
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});


const inputArr = [];
rl.on('line', function(line){
    inputArr.push(line);//将输入流转换为数字类型保存到inputArr中
}).on('close', function(){
    sortScore(inputArr);//调用解决函数并输出
})

//解决函数
function sortScore(inputArr) {

}
```

- question: HJ12 字符串反转 link: https://www.nowcoder.com/practice/e45e078701ab4e4cb49393ae30f1bb04?tpId=37&tqId=21235&ru=/exam/oj
    - answer:

```
# python code
import sys
# 逆序遍历数组 <=> 切片stride = -1 <=> list[start: end: stride]
for line in sys.stdin:
    a = line.split()
    # line.split() return a list
    print(a[0][::-1])
```

```
// Js code
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output:process.stdout
})

rl.on("line", function line(line) {
    let res = ""
    for (var i = line.length - 1; i >= 0; i--) {
        res += line[i];
    }
    console.log(res)
});
```

- question: HJ68 成绩排序 link: https://www.nowcoder.com/practice/8e400fd9905747e4acc2aeed7240978b?tpId=37&tqId=21291&ru=/exam/oj
    - answer:

```python
# python code
import sys
# read one line only
n = int(input())
# Python三目运算符
order = True if int(input()) == 0 else False

score = []
for line in sys.stdin:
    s = line.split()
    score.append([s[0], int(s[1])])

# 按照key排序, 默认排序为升序, reverse是否逆序.
score.sort(key=lambda x: x[1], reverse=order)

# print(*s) 可以直接将list打印出来且间隔为一个空格.
for s in score:
    # print(*s)
    print(s[0], s[1])
```

```javascript
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
//js遇到多行输入问题时，首先用rl.on('line', function(line){})逐行读取字符串存入inputArr中，随后在.on('close', function())中用特定函数对inputArr处理
const inputArr = [];
rl.on('line', function(line){
    inputArr.push(line);//将输入流转换为数字类型保存到inputArr中
}).on('close', function(){
    sortScore(inputArr);//调用解决函数并输出
})

//解决函数
function sortScore(inputArr) {
    let studentNum = inputArr[0];
    let flag = parseInt(inputArr[1]);
    let students = [];

    // push the students into the array
    // js 的数组可以存json类型的数据，即键值对
    for (let i = 0; i < studentNum; i++) {
        var line = inputArr[2+i].split(" ");
        var name = line[0];
        var score = parseInt(line[1]);
        students.push({"name":name, "score":score, "index":i});
    }

    // sort the array
    // js sort默认情况是按照从小到大排序，如果数据是json键值对，那么需要编写sort的回调函数来定义顺序规则，升序s1-s2，降序s2-s1。json数据可用.name的方法进行比较
    if (flag == 0) {
        students.sort((s1, s2) => {
            return s2.score - s1.score || s1.index - s2.index
        })
    } else if (flag == 1) {
        students.sort((s1, s2) => {
            return s1.score - s2.score
        })
    }
    for (var student of students) {
        console.log(student.name + " " + student.score);
    }
}
```

- question: LC442 数组中重复的数据 link: https://leetcode.cn/problems/find-all-duplicates-in-an-array/
    - answer:

```python
# python code
class Solution:
    def findDuplicates(self, nums: List[int]) -> List[int]:
        # store freq
        dic = {}
        # store ans
        ans = []

        for n in nums:
            # record freq, get value from dic according to key and default value is 0
            dic[n] = dic.get(n, 0) + 1
            if dic[n] > 1:
                ans.append(n)

        return ans
```

```javascript
/**
 * @param {number[]} nums
 * @return {number[]}
 */
var findDuplicates = function(nums) {
    let res =[];
    nums.sort();
    for (let i = 1; i < nums.length; i++) {
        if (nums[i] == nums[i-1]) {
            res.push(nums[i]);
        }
    }
    return res;
};
```

- question: LC448 找到所有数组中消失的数字 link: https://leetcode.cn/problems/find-all-numbers-disappeared-in-an-array/
    - answer:

```
# python code
# 如果遍历1—len(nums)+1，去寻找对应值在nums出现与否会超时。
# 不允许使用额外空间，利用nums充当字典，先遍历nums，nums中的值作为key，改变对应key的value为负值以此标记此值是否出现过，最后筛选value大于0的key则是没出现过的key。
class Solution:
    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
        # get n from nums as the key to reverse nums[n]
        for n in nums:
            # because the value appeared is negative, use abs to get positive value
            nums[abs(n)-1] = - abs(nums[abs(n)-1])

        # python一行赋值list [目标值 循环方式 条件]
        ans = [n+1 for n in range(len(nums)) if nums[n] > 0]

        return ans
```

```
/**
 * 使用Js set不能存储重复值的特性以及set.has()方法判断nums中的缺失值
 * @param {number[]} nums
 * @return {number[]}
 */
var findDisappearedNumbers = function(nums) {
    let set = new Set();
    let res = [];
    // add nums into the set
    for (let i = 0; i < nums.length; i++) {
        set.add(nums[i]);
    }
    // vertify every num from [1,n] if they are existed in the set
    for (let i = 1; i <= nums.length; i++) {
        if (!set.has(i)) {
            res.push(i);
        }
    }
    return res;
};
```

- question: LC1002 查找共用字符 link: https://leetcode.cn/problems/find-common-characters/
    - answer:

```
# python code
class Solution:
    def commonChars(self, words: List[str]) -> List[str]:
        # simple version
        minFreq = [float("inf")] * 26

        for word in words:
            freq = [0] * 26
            for latter in word:
                freq[ord(latter) - ord("a")] += 1

            for i in range(26):
                minFreq[i] = min(minFreq[i], freq[i])

        ans = []
        for i in range(26):
            ans.extend([chr(i + ord("a"))] * minFreq[i])

        return ans
```

```
# python code
class Solution:
    def commonChars(self, words: List[str]) -> List[str]:
        # enhanced version
        # Counter to create a dic about the string count the freq for every char
        freq = Counter(words[0])

        for word in words:
            freq &= Counter(word)

        # Counter.elements() returns value keys, if "a":3 then "a a a"
        return list(freq.elements())

        # more enhanced version
        # reduce叠加遍历 reduce(function, list)
        # return list(reduce(lambda x, y: x & y, map(collections.Counter, words)).elements())
```

```
// 自己的思路
//  * @param {string[]}
//  * @return {string[]}
//  */
// var commonChars = function(words) {
//     let res = [];
//     let firstMap = new Map()
//     for (let j = 0; j < words[0].length; j++) {
//         if (firstMap.has(words[0][j])) {
//             firstMap.set(words[0][j], firstMap.get(words[0][j]) + 1 );
//         } else {
//             firstMap.set(words[0][j], 1);
//         }
//     }
//     for (let i = 0; i < words.length; i++ ) {
//         let map = new Map();
//         let word = words[i];
//         for (let j = 0; j < word.length; j++) {
//             if (map.has(words[0][j])) {
//                 map.set(words[0][j], firstMap.get(words[0][j]) + 1 );
//             } else {
//                 map.set(words[0][j], 1);
//             }
//         }
//         firstSet = res = testSame(map, firstMap);
//     }
//     return res;
// };

// function testSame(map1, map2){
//     let res = new Map();
//     const iterator = map1.values();

//     for(let pair1 of map1) {
//         console.log("pair1 is ", pair1.key,  pair1.value)
//         for (let pair2 of map2) {
//             if (pair1.key == pair2.key) {
//                 res.set(pair1, (pair1.value > pair2.value)?pair2.value:pair2.value)
//             }
//         }
//     }
//     return res;
// }

/**
 * @param {string[]} words
 * @return {string[]}
 */
//主要思路为建立一个26个字母组成的arr来记录每单个单词的字母出现顺序，然后将arr放入arrlist当中，最后遍历arraylist26遍，每遍确定一个字母的数量，然后把对应数量的数组置于res结果数组中。
var commonChars = function(words) {
    let res = [];
    let arrayList = [];
    for (let word of words) {
        let array = new Array(26).fill(0); //.fill可以批量填充数组的值
        for (let char of word) {
            array[char.charCodeAt() - "a".charCodeAt()]++; //charCodeAt可以找对应字母deASCII码
        }
        arrayList.push(array);
    }

    for (let i = 0; i<26; i++) {
        let min = 100;
        for (let arr of arrayList) {
            if (arr[i] < min) {
                min = arr[i]
            }
        }
        for (let j = 0; j < min; j++) {
            res.push(String.fromCharCode(i + "a".charCodeAt()));
        }
    }
    return res;
};
```

- question: LC1370 上升下降字符串 link: https://leetcode.cn/problems/increasing-decreasing-string/
    - answer:

```
# python code
# 如果题目指出字符串只有a-z最好使用26长度的数组
# 字典也可以进行排序 sorted(dic, key=some key)
class Solution:
    def sortString(self, s: str) -> str:
        alpha = [0] * 26
        # count for s
        for l in s:
            alpha[ord(l)-ord("a")] += 1

        ans = []

        while len(ans) < len(s):
            # start from min
            for i in range(26):
                if alpha[i]:
                    ans.append(chr(i + ord("a")))
                    alpha[i] -= 1
            # start from max
            for i in range(26):
                if alpha[26-i-1]:
                    ans.append(chr(26-i-1 + ord("a")))
                    alpha[26-i-1] -= 1

        # "".join(list) can get a string combination
        return "".join(ans)
```

```
/**
 * @param {string} s
 * @return {string}
 */

//  main idea: 使用数组的index作为字母键值对的key，Value作为字母出现的次数，将字符串s中的字母进行录入到数组。录入后，使用do while 语句先将26个元素遍历一遍，搞出第一遍的顺序，遍历时减少对应数组的value值，同时如果value依然大于0，就代表还要继续遍历，unfinish就是变为true。然后在while中以unfinish作为条件继续遍历。要实现反向顺序，就要使用另外一个boolean变量 inorder进行操作，默认正序，每轮遍历自目前，都要将此值反转。
var sortString = function(s) {
    let res = [];
    let alphaList = new Array(26).fill(0);
    let unfinish;
    let inorder = true;
    // add the data into alphaList
    for(let char of s) {
        alphaList[char.charCodeAt() - "a".charCodeAt()]++;
    }
    //
    do {
        unfinish = false;
        if (inorder) {
            inorder = !inorder;
            for (let i = 0; i < 26; i ++) {
                if (alphaList[i]>0) {
                    res.push(String.fromCharCode("a".charCodeAt() + i));
                    alphaList[i]--;
                }
                if (alphaList[i]>0) {
                    unfinish = true;
                }
            }
        } else {
            inorder = !inorder;
            for (let i = 25; i >= 0; i --) {
                if (alphaList[i]>0) {
                    res.push(String.fromCharCode("a".charCodeAt() + i));
                    alphaList[i]--;
                }
                if (alphaList[i]>0) {
                    unfinish = true;
                }
            }
        }
    } while (unfinish)
    return res.join("");
};

// 借鉴
// unfinish 和 inorder不必要，可以使用(res.length=s.length)对循环进行判断
```

双指针

- question: LC283 移动零 link: https://leetcode.cn/problems/move-zeroes/
    - answer:

```
# python code
# like quick sort
# 双指针，右指针指向的位置不为0时与左之指针交换
# 可以保持相对顺序，如果两者都不为0时，同时右移一格，并不会交换不为0的值。
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)

        left, right = 0,0

        while right < n:
            # 不为零可以写作 if nums[right]:
            if nums[right] != 0:
                nums[right], nums[left] = nums[left], nums[right]
                left += 1
            right += 1

        return nums
```

```
// /**
//  * @param {number[]} nums
//  * @return {void} Do not return anything, modify nums in-place instead.
//  */
// // 方法1：使用双指针，两个指针全部从0到len, 左指针是代表按顺序的数组全部元素，思想是一位一位往里面填非零的数。右指针遍历数组查看非零的数，并将非零的数与左指针数值交换。
// var moveZeroes = function(nums) {
//     let res = [];
//     let zeros = [];
//     let left = 0;
//     for (let i = 0; i < nums.length; i ++) {
//         if (nums[i] !== 0) {
//             let temp = nums[i];
//             nums[i] = nums[left];
//             nums[left] = temp;
//             left++;
//         }
//     }
// };

// 方法2：使用slice删除0元素，同时在数组后面补0
var moveZeroes = function(nums) {
    for (let i = 0; i < nums.length; i++) {
        if (nums[i] === 0) {
            nums.slice(0, i).concat(nums.slice(i+1, nums.length));
            nums.push(0);
        }
    }
};
```

- question: LC26 删除有序数组中的重复元素 link: https://leetcode.cn/problems/remove-duplicates-from-sorted-array/
    - answer:

```
# python code
# 双指针，左指针记录唯一元素，如果重复，右指针右移直到不重复。
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:

        left, right = 1,1

        while right < len(nums):
            # 用加法不太好，容易超出范围，最好用减法
            # if not nums[right] == nums[left]:
            #     nums[left + 1] = nums[right]
            if nums[right] != nums[left - 1]
                nums[left] = nums[right]
                left += 1

            right += 1

        # left最终指向需要数组的后一格
        return left
```

```
/**
 * @param {number[]} nums
 * @return {number}
 */
// 方法一： 快慢指针（慢指针变换条件为快指针与快指针前一位比较），slow的作用是一位一位的将符合条件的数字填入数组。条件为通过fast和fast前一位比较，确定nums[slow]的值。fast与fast前一位一致，那么就不改变slow，如果不一致没那么就可以使nums[slow]=nums[fast]。此外条件还可以比较slow和fast位置是否相等，如果不相等的话，那么就用fast位置的数值填入slow位置
 var removeDuplicates = function(nums) {
    let slow = 0;
    let fast = 1;
    while (fast < nums.length) {
        if (nums[fast] !== nums[slow]) {
            nums[slow + 1] = nums[fast];
            slow++;
        }
        fast++;
    }
    return slow+1;
};

// 方法二：使用splice，对nums从第一位left进行遍历，判断left位置和left上一位的数值是否相等，相等的话就继续检测下一位是否依然相等，然后统计出相等树枝的数量len，然后对nums数组使用nums.splice(left, len)进行裁剪。
 var removeDuplicates = function(nums) {
    let left = 1;
    while (left < nums.length) {
        if (nums[left] != nums[left - 1]) {
            left++;
        } else {
            let len = 1;
            while (nums[left-1] == nums[left+len]) {
                len++;
            }
            nums.splice(left, len);
        }
    }
};
```

- question: LC80 删除排序数组中的重复元素二 link: https://leetcode.cn/problems/remove-duplicates-from-sorted-array-ii/
    - answer:

```python
# python code
# 有个重复的就去检查前第几个数，并且双指针初始化时要相应改变。同时，用减法可以避免out of index的问题。
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        n = len(nums)

        left, right = 2,2

        while right < n:

            if nums[right] != nums[left - 2]:
                nums[left] = nums[right]
                left += 1

            right += 1

        # left最终指向需要数组的后一格
        return left
```

```javascript
/**
 * @param {number[]} nums
 * @return {number}
 */
// 同样的快慢指针，快慢指针一同从第3个元素出发，检验慢指针的前第二位是否与fast值相等，如果不相等，那么就对慢指针位置用快指针赋值，若果相等，那么慢指针不动，继续用下一个fast比较，如果不同再换。

var removeDuplicates = function(nums) {
    let slow = fast = 2;
    if (nums.length <= 2) return 2;
    while (fast < nums.length) {
        if (nums[slow-2] != nums[fast]) {
            nums[slow] = nums[fast];
            slow++;
        }
        fast++;
    }
    return slow;
};

//方法二：splice
var removeDuplicates = function(nums) {
    let start = 2;
    while (start < nums.length) {
        if (nums[start] == nums[start-2]) {
            let len = 1;
            while (nums[start + len] == nums[start]) {
                len++;
            }
            nums.splice(start, len)
        }
        start++;
    }
};
```

- question: LC27 移除元素 link: https://leetcode.cn/problems/remove-element/
    - answer:

```python
# python code
# 所有删除元素且不占用额外空间，都可以使用交换的方式。
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:

        n = len(nums)

        left, right = 0,0

        while right < n:
            if nums[right] != val:
                nums[left], nums[right] = nums[right], nums[left]
                left += 1

            right += 1

        return left
```

```
/**
 * @param {number[]} nums
 * @param {number} val
 * @return {number}
 */
// 方法一：快慢指针
var removeElement = function(nums, val) {
    let slow = fast = 0;
    while (fast < nums.length) {
        if (nums[fast] != val) {
            nums[slow] = nums[fast]
            slow++;
        }
        fast++;
    }
    return slow;
};



// 方法二:暴力splice
// var removeElement = function(nums, val) {
//     let start = 0;
//     while (start < nums.length) {
//         if (nums[start] == val) {
//             let len = 1;
//             while (nums[start] == nums[start+len]) {
//                 len++;
//             }
//             nums.splice(start,len);
//         }
//         start++;
//     }
// };
```

- question: LC344 反转字符串 link: https://leetcode.cn/problems/reverse-string/
    - answer:

```
# easy to swap
# 双指针，一个开头， 一个结尾
class Solution:
    def reverseString(self, s: List[str]) -> None:
        """
        Do not return anything, modify s in-place instead.
        """

        n = len(s)

        right, left = 0, n-1

        while right < left:
            s[right], s[left] = s[left], s[right]
            right += 1
            left -= 1

        return s
```

```
/**
 * @param {character[]} s
 * @return {void} Do not return anything, modify s in-place instead.
 */
var reverseString = function(s) {
    let left = 0;
    let right = s.length - 1;
    while (left < right) {
        let temp = s[left];
        s[left] = s[right];
        s[right] = temp;
        left ++;
        right --;
    }
    return s;
};
```

- question: LC125 验证回文串 link: https://leetcode.cn/problems/valid-palindrome/
    - answer:

```
# python code
# 重点在于处理字符串，去除空格用str.strip()，去除别的字符用str.isalnum()，只包含数字和字母时返回TRUE。
class Solution:
    def isPalindrome(self, s: str) -> bool:

        s = "".join(ch.lower() for ch in s if ch.isalnum())

        left, right = 0, len(s)-1

        while left < right:
            if s[left] != s[right]:
                return False

            left += 1
            right -= 1


        return True
```

```
var isPalindrome = function(s) {
    s=s.replace(/[^a-zA-Z0-9]/g,"").replace(/\s/g,"").toLowerCase(); // replace 函数替换字符串中的非字母字符
    //toLowerCase()用于对字符串进行全小写处理
    let left = 0;
    let right = s.length - 1;
    while (left <= right) {
        if (s[left] !== s[right]) {
            return false;
        }
        left++;
        right--;
    }
    return true;
};
```

- question: LC11 盛最多水的容器 link: https://leetcode.cn/problems/container-with-most-water/
    - answer:

```
# python code
# 双指针，一头一尾，每次找最短边向内收缩直到双指针相遇。
class Solution:
    def maxArea(self, height: List[int]) -> int:
        n = len(height)

        left, right = 0, n-1
        v = 0

        while left < right:

            temp = min(height[left], height[right]) * (right - left)

            v = max(v, temp)

            if height[left] < height[right]:
                left += 1
            else:
                right -= 1

        return v
```

```
/**
 * @param {number[]} height
 * @return {number}
 */
// 收尾双指针向内收缩，短的一边收缩。
var maxArea = function(height) {
    let max = 0;
    let left = 0;
    let right = height.length - 1;
    while (left < right){
        let width = right - left;
        let shortest = height[left] < height[right] ? height[left++] : height[right--];
        let currentVolume =  width * shortest;
        if (currentVolume > max) {
            max = currentVolume;
        }
    }
    return max;
};
```

一维数组

- question: LC1480 一维数组的动态和（前缀和） link: https://leetcode.cn/problems/running-sum-of-1d-array/
    - answer:

```
# python code
# 当前项等于前一项加上自己。
class Solution:
    def runningSum(self, nums: List[int]) -> List[int]:

        for i in range(1, len(nums)):
            nums[i] += nums[i-1]

        return nums
```

```
/**
 * @param {number[]} nums
 * @return {number[]}
 */
var runningSum = function(nums) {
    let runningSum = [];
    let sum = 0;
    for (let i = 0; i < nums.length; i++) {
        sum += nums[i]
        runningSum[i] = sum;
    }
    return runningSum;
};
```

- question: LC238 除自身以外数组的乘积 link: https://leetcode.cn/problems/product-of-array-except-self/
    - answer:

```
# python code
# 除自身外的除数分两部分，一部分是左边所有数相乘，一部分是右边所有数相乘，最后两边相乘。
# 计算单边是，分开计算，叠加会更高效
# 只用一个数组来存储可以节省空间，但要考虑到最左边和最右边的点，初始化为1。
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        n = len(nums)
        ans = [0] * n

        ans[0] = 1

        for i in range(1, n):
            ans[i] = ans[i-1] * nums[i-1]

        r = 1

        # for i in reversed(range(length)):
        # reversed 可以使得 i 从n到0进行遍历
        for i in range(1, n+1):
            ans[n-i] = ans[n-i] * r
            r *= nums[n-i]

        return ans
```

```
// 左右积合一起
var productExceptSelf = function(nums) {
  let ans = [];
  let left = [];
  left.push(1);
  let right = [];
  right.push(1);
  let len = nums.length;
  for (let i = 1; i < len; i++) {
      left.push(left[i-1] * nums[i-1])
      right.push(right[i - 1] * nums[len - i])
      console.log("nums[len - i - 1]", nums[len - i - 1]);
  }
  right = right.reverse();
  for (let i = 0; i < len; i++) {
      console.log("i: ", i, "left: ", left[i], "right: ", right[i]);
      ans[i] = left[i] * right[i];
  }
  return ans;
};

console.log(productExceptSelf([1,2,3,4]));
```

- question: LC941 有效的山脉数组 link: https://leetcode.cn/problems/valid-mountain-array/
    - answer:

```
# python code
# 双循环检索，第一个循环检索上坡，第二个循环检测下坡，中间任何间断则报错。
class Solution:
    def validMountainArray(self, arr: List[int]) -> bool:
        N = len(arr)
        i = 0

        while i + 1 < N and arr[i] < arr[i + 1]:
            i += 1

        if i == 0 or i == N - 1:
            return False

        while i + 1 < N and arr[i] > arr[i + 1]:
            i += 1

        return i == N - 1
```

```

//顺序扫描数组，查看扫描一升一降(或)后的i的位置是否是原数组的长度。如果短了那么说明不是山脉数组。但是要注意山脉数组的最高点不能是第一位和最后一位
var validMountainArray = function(arr) {
    const N = arr.length;
    let i = 0;

    // 递增扫描
    while (i + 1 < N && arr[i] < arr[i + 1]) {
        i++;
    }

    // 最高点不能是数组的第一个位置或最后一个位置
    if (i === 0 || i === N - 1) {
        return false;
    }

    // 递减扫描
    while (i + 1 < N && arr[i] > arr[i + 1]) {
        i++;
    }

    return i === N - 1;
};
```

- question: LC189 旋转数组 link: https://leetcode.cn/problems/rotate-array/
    - answer:

```
# python code
# 关键点一：原地改动nums，任何的赋值操作都会导致nums的id变化，只对nums内的元素做操作才不会导致id变化。
# nums = nums[::-1] 会导致nums的id变化
# nums[:] = nums[::-1] 不会导致nums的id变化，因为nums[:]表示对nums内每个元素对应赋值。
# 关键点二：k值大于len(nums)时，需要对k做处理，当k等length时，相当于对数组没操作，所以对length取余则可以得到最终的k。
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        k = k % len(nums)

        nums[:] = nums[::-1]

        nums[k:len(nums)] = nums[k:len(nums)][::-1]

        nums[0:k] = nums[0:k][::-1]

        return nums
```

```
/**
 * @param {number[]} nums
 * @param {number} k
 * @return {void} Do not return anything, modify nums in-place instead.
 */
// 旋转数组，将数组切为两份，然后将后一份放置于前一份前。注意本题需要最后在原数组的地址上进行数组变更，另外要注意旋转次数大于数组长度的情况，使用取余将问题简化。
var rotate = function(nums, k) {
    let len = nums.length;
    k = k % len;
    let num1 = nums.slice(len - k, len);
    let num2 = nums.slice(0, len - k);
    let newNums = num1.concat(num2);
    for (let i = 0; i < len; i ++) {
        nums[i] = newNums[i]
    }
    return nums;
};
```

- question: LC665 非递减数列 link: https://leetcode.cn/problems/non-decreasing-array/
    - answer:

```
# python code
# 遇到突变时，检测突变时检测当前位置与前两位的大小，必须使得当前位置大于前两位的值，否则就需要更改不止一次，如果是小于前两位的值，则当前位置能选择的最小值则是前一位的值。
class Solution:
    def checkPossibility(self, nums: List[int]) -> bool:
        n = len(nums)

        flag = True

        for i in range(1, n):
            if nums[i] < nums[i-1]:
                if flag:
                    flag = False
                else:
                    return flag

                if i > 1 and nums[i] < nums[i-2]:
                    nums[i] = nums[i-1]

        return True
```

```
[3,4,2,3]
为了方便处理我前面和后面加入两个哨兵元素后数组变为
[-Infinity,3,4,2,3,Infinity]
当i等于3时，4>2，出现了一个减小的情况。
关键来了，这时我们有两种改变方法，满足一个即可。
将4改为2 ，这时必须满足 4左边的元素小于等于2
将2改为4，这时必须满足4小于等于2右边的元素。  如果上述两种都不能成立其中一个，那么就说不能修改，返回false即可。
关键就是上面的两个情况。相信各位大佬能简单看懂。希望各位大佬提出修改意见

var checkPossibility = function(nums) {
    let sum = 0;
    nums.push(Infinity)
    nums.unshift(-Infinity)
    for (let i = 2; i < nums.length - 1; i++) {
        if (nums[i-1]> nums[i]) {
            sum++;
            if (nums[i-1]>nums[i+1] && nums[i] < nums[i-2]) return false;
        }
        if (sum>1) {
            return false;
        }

    }
    return true;
};
```

- question: LC228 汇总区间 link: https://leetcode.cn/problems/summary-ranges/
    - answer:

```python
# python code
# 移动end，满足条件则end移动，否则更新start和end，最后跳出循环后还得再append。
class Solution:
    def summaryRanges(self, nums: List[int]) -> List[str]:

        n = len(nums)
        if not nums:
            return []

        start, end = nums[0],nums[0]

        ans = []

        for i in range(1, n):
            if nums[i] == end + 1:
                end = nums[i]
            else:
                if start != end:
                    ans.append("->".join((str(start), str(end))))
                else:
                    ans.append(str(start))

                start, end = nums[i], nums[i]

        if start != end:
            ans.append("->".join((str(start), str(end))))
        else:
            ans.append(str(start))

        return ans
```

```javascript
/**
 * @param {number[]} nums
 * @return {string[]}
 */
// var summaryRanges = function(nums) {
//     let end = 1;
//     let res = [];
//     while (end < nums.length) {
//         start = end - 1;
//         while (nums[end - 1] + 1 == nums[end]) {
//             end++;
//         }
//         if (start == end - 1) {
//             res.push(nums[start].toString());
//             end++
//         } else {
//             res.push(nums[start]+"->"+nums[end]);
//         }
//     }
//     return res;
// };

var summaryRanges = function(nums) {
    let piv = 0;
    let res = [];
    while (piv < nums.length) {
        start = piv;
        while (nums[piv] + 1 == nums[piv + 1]) {
            piv++;
        }
        if (piv == start) {
            res.push(nums[start].toString());
        } else {
            res.push(nums[start]+"->"+nums[piv]);
        }
        piv++;
    }
    return res;
};
```

- question: LC163 缺失的区间(vip) link: https://blog.csdn.net/ft_sunshine/article/details/103445054
    - answer:

```
# python code
def summaryRanges(nums, lower, upper):

    n = len(nums)
    if not nums:
        return []

    ans = []
    start = lower

    for i in range(1, n):
        if nums[i] == start + 1:
            start = nums[i]
        else:
            start += 1
            if nums[i] < upper:
                end = nums[i] - 1
                if start != end:
                    ans.append("->".join((str(start), str(end))))
                else:
                    ans.append(str(start))

            else:
                end = upper
                if start != end:
                    ans.append("->".join((str(start), str(end))))
                else:
                    ans.append(str(start))
                break

            start = nums[i]

    if upper > nums[n-1]:
        start += 1
        end = upper
        if start != end:
            ans.append("->".join((str(start), str(end))))
        else:
            ans.append(str(start))

    return ans
```

```
//简化版
var antiSummaryRanges = function(nums) {
  let res = [];
  let piv = 0;
  let start = 0;
  let end = 0;
  while (piv < nums.length - 1) {
    console.log("piv ", piv)
    if (nums[piv] + 1 != nums[piv+1]) {
      start = nums[piv] + 1;
      let len = 0;
      console.log("start " + start);
      end = nums[piv+1] - 1;
      console.log("end " + end);


      // print the array
      if (end == start) {
        res.push(start);
      } else {
        res.push(start+"->"+end);
      }
    }
    piv++;
  }
  return res;
}

console.log(antiSummaryRanges([0, 1, 3, 50, 75]));
```

- question: LC31 下一个排列 link: https://leetcode.cn/problems/next-permutation/
    - answer:

```python
# python code
# 先从尾部开始，找到下降点，下降点右边的点比较，找到大于下降点的最小值，与之交换，交换后，将下降点右边的点倒序，得到下一个值。
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        n = len(nums)

        i, k = n-2, n-1

        while i >= 0 and nums[i] >= nums[i+1]:
            i -= 1

        if i >= 0:
            while nums[i] >= nums[k]:
                k -= 1

            nums[i], nums[k] = nums[k], nums[i]

        i, k = i + 1, n - 1

        while nums[i] > nums[k] and i < k:
            nums[i], nums[k] = nums[k], nums[i]
            i += 1
            k -= 1

        return nums
```

不会啊，交给时间吧

```javascript
 var nextPermutation = function(nums, n = nums.length) {
  if (!n) return []
  let p = n - 1, p1 = n - 1, nearIndex = 0, minD = Infinity
  while(p > 0) {
    if (nums[p] > nums[p - 1]) break
    p -- // p为末尾开始算起的第一个非递减序列的起始值
  }
  if (p === 0) {
    nums.sort((a, b) => a - b)
    return nums
  }
  while (p1 > p - 1) {
    if (minD > nums[p1] - nums[p - 1] && nums[p1] > nums[p - 1]) {
      minD = nums[p1] - nums[p - 1]
      nearIndex = p1
    }
    p1 --
  }
  // 交换邻近大值
  let tmp = nums[nearIndex]
  nums[nearIndex] = nums[p - 1
  nums[p - 1] = tmp

  for (let i = p; i < n; i++) { // 在区间i ~ n - 1做插排
    let min = Infinity, k = i
    for (let j = i; j < n; j++) {
      if (min > nums[j]) {
        min = nums[j]
        k = j
      }
    }
    if (k !== i) {
      nums[k] = nums[i]
      nums[i] = min
    }
  }
  return nums
};
```

- question: LC135 分发糖果（困难） link: https://leetcode.cn/problems/candy/
    - answer:

```python
# python code
# 两次遍历，先从右边开始，找比右边大的，再从左边开始，找比左边大的。两者取最大值
class Solution:
    def candy(self, ratings: List[int]) -> int:
        n = len(ratings)
        left = [1] * n

        for i in range(1, n):
            if ratings[i] > ratings[i - 1]:
                left[i] = left[i - 1] + 1

        right = 1

        for i in reversed(range(n-1)):
            if ratings[i] > ratings[i + 1]:
                right += 1
            else:
                right = 1

            left[i] = max(right, left[i])

        return sum(left)
```

```
//我们遍历该数组两次，处理出每一个学生分别满足左规则或右规则时，最少需要被分得的糖果数量。每个人最终分得的糖果数量即为这两个数量的最大值。
var candy = function(ratings) {
    const n = ratings.length;
    const left = new Array(n).fill(0);
    for (let i = 0; i < n; i++) {
        if (i > 0 && ratings[i] > ratings[i - 1]) {
            left[i] = left[i - 1] + 1;
        } else {
            left[i] = 1;
        }
    }

    let right = 0, ret = 0;
    for (let i = n - 1; i > -1; i--) {
        if (i < n - 1 && ratings[i] > ratings[i + 1]) {
            right++;
        } else {
            right = 1;
        }
        ret += Math.max(left[i], right);
    }
    return ret;
};
```

- question: LC605 种花问题 link: https://leetcode.cn/problems/can-place-flowers/
    - answer:

```
# python code
# 找到1的位置，记录下来，每次找到1与之前的1的位置相减之后再减2并整除2，得到可以种花的位置。循环结束后要寻找n+1的位置与当前pre的位置之间是否可以再种。
class Solution:
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:

        m = len(flowerbed)

        pre, cnt = -1, 0

        for i in range(m):
            if flowerbed[i] == 1:
                if pre == -1:
                    cnt += i//2
                else:
                    cnt += (i - pre - 2) // 2
                pre = i

                if cnt >= n:
                    return True

        if pre == -1:
            cnt += (m + 1) // 2
        else:
            cnt += (m - pre - 1) // 2

        return True if cnt >= n else False
```

```
// /**
//  * @param {number[]} flowerbed
//  * @param {number} n
//  * @return {boolean}
//  */
// var canPlaceFlowers = function(flowerbed, n) {
//     let space = 0;
//     let pivot = 0;

//     if (n == 0) {
//         return true
//     }

//     if (flowerbed.length == 1) {
//         console.log("!", flowerbed[0])
//         return flowerbed[0] == 0;
//     }
//     if (flowerbed.length == 2) {

//         if (flowerbed[1] == 0 && flowerbed[0] == 0) {
//             space++
//         }
//         return space>=n;
//     }


//     if (flowerbed[0] == 0 && flowerbed[1] == 0) {
//         flowerbed[0] = 1;
//         space++;
//         console.log("pre", space);
//     }
//     while (pivot < flowerbed.length - 1) {
//         if (flowerbed[pivot] == 0) {
//             if (flowerbed[pivot - 1] != 1 && flowerbed[pivot + 1] != 1) {
//             flowerbed[pivot] = 1;
//             space++;
//             console.log("in", space);
//         }
//         }
//         pivot ++;
//     }

//     if (flowerbed[flowerbed.length - 1] == 0 && flowerbed[flowerbed.length - 2] == 0) {
//         flowerbed[0] = 1;
//         space++;
//         console.log("end", space);
//     }
//     console.log(space)
//     return space>=n
// };

var canPlaceFlowers = function(flowerbed, n) {
    let space = 0;
    let pivot = 0;


    // 当存在当前值需要前后比较时，将首元素和尾元素的特殊情况可以使用||与前后值的比较相结合，比如，如果前置比较||首值的特殊情况 。
    while (pivot < flowerbed.length) {
        if (flowerbed[pivot] == 0) {
            if ((pivot == 0 || flowerbed[pivot - 1] == 0) && (pivot == flowerbed.length - 1 || flowerbed[pivot + 1] == 0)) {
            flowerbed[pivot] = 1;
            space++;
            console.log("pivot is ", pivot)
            }
        }
        pivot ++;
    }

    console.log(space)
    return space>=n
};
```

- question: LC860 柠檬水找零 link: https://leetcode.cn/problems/lemonade-change/
    - answer:

```
# python code
# hard code, find all possible situations.
from collections import Counter
class Solution:
    def lemonadeChange(self, bills: List[int]) -> bool:
        five, ten = 0,0
        for i in bills:
            if i == 5:
                five += 1
```

```
/**
 * @param {number[]} bills
 * @return {boolean}
 */
var lemonadeChange = function(bills) {
    let len = bills.length;
    let five = 0;
    let ten = 0;
    for (let i= 0; i < len; i++) {
        let money = bills[i];
        if(money == 5) {
            five++;
        } else if (money == 10) {
            ten++
        }
         {
            while (money > 10 && ten > 0){
                money-=10;
                ten--;
            }
            while (money > 5) {
                console.log(money)
                money -= 5;
                five --;
            }
            if (five<0) {
                return false;
            }
        }
    }
    return true;
};
```

二维数组

- question: lc867 矩阵转置 link: https://leetcode.cn/problems/transpose-matrix/
    - answer:

```
# python code

        for i in range(n):
            dic1 = {}
            dic2 = {}
            for j in range(n):
                if board[i][j] != '.':
                    if board[i][j] in dic1:
                        return False
                    else:
                        dic1[board[i][j]] = 1

                if board[j][i] != '.':
                    if board[j][i] in dic2:
                        return False
                    else:
                        dic2[board[j][i]] = 1

        for i in range(0,n,3):
            for j in range(0,n,3):
                dic = {}
                for m in range(n):
                    if board[i+(m//3)][j+(m%3)] != '.':
                        if board[i+(m//3)][j+(m%3)] in dic:
                            return False
                        else:
                            dic[board[i+(m//3)][j+(m%3)]] = 1

        return True
```

```
/**
 * @param {number[][]} matrix
 * @return {number[][]}
 */
var transpose = function(matrix) {
    let res = [];
    let m = matrix.length;
    let n = matrix[0].length;
    for (let i = 0; i< n; i++) {
        let row = matrix[i];
        let resLine = []

        for (let j = 0; j < m; j++) {
            resLine.push(matrix[j][i]);
        }
        res.push(resLine);
    }
    return res;
};
```

- question: lc48 旋转图像 link: https://leetcode.cn/problems/rotate-image
    - answer:

```python
# python code
# 两种思路解决，第一是通过先转置，再reverse。第二种，找对应关系m[i][j] 对应 m[j][n-1-i]。
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix)

        for i in range(n):
            for j in range(i, n):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

        for i in range(n):
            matrix[i][:] = matrix[i][::-1]
```

```javascript
/**
 * @param {number[][]} matrix
 * @return {void} Do not return anything, modify matrix in-place instead.
 */

// js初始定义数组的时候，不要使用fill 要使用 Array.from(Array(n), item => new Array(n).fill(0)将数组变换
var rotate = function(matrix) {
    let n = matrix.length
    let arr = Array.from(Array(n), item => new Array(n).fill(0))
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++){
            arr[j][n - 1 - i] = matrix[i][j]
        }
    }
    console.log(arr);
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++){
            matrix[i][j] = arr[i][j]
        }
    }
};
```

- question: lc36 有效的数独 link: https://leetcode.cn/problems/valid-sudoku
    - answer:

```python
# python code
# 进行三次扫描，第一次找每一行，第二次找每一列，第三次找每个3x3。其中，最后一步中，m从0~8，对应3x3的行数为m//3，列数为m%3。由此得到3x3内的位置。
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        n = len(board)

        for i in range(n):
            dic1 = {}
            dic2 = {}
            for j in range(n):
                if board[i][j] != '.':
                    if board[i][j] in dic1:
                        return False
                    else:
                        dic1[board[i][j]] = 1

                if board[j][i] != '.':
                    if board[j][i] in dic2:
                        return False
                    else:
                        dic2[board[j][i]] = 1

        for i in range(0,n,3):
            for j in range(0,n,3):
                dic = {}
                for m in range(n):
                    if board[i+(m//3)][j+(m%3)] != '.':
                        if board[i+(m//3)][j+(m%3)] in dic:
                            return False
                        else:
                            dic[board[i+(m//3)][j+(m%3)]] = 1

        return True
```

```javascript
/**
 * @param {character[][]} board
 * @return {boolean}
 */
var isValidSudoku = function(board) {
    // check the line
    for (let i = 0; i<9; i++) {
        let arr = new Array(9).fill(false);
        for (let j = 0; j < 9; j++) {
            if (arr[board[i][j]] == true) {
                console.log(i, j, "return false by row")
                return false;
            }
            if (board[i][j] != ".") {
                arr[board[i][j]] = true;
            }
        }
    };

    for (let i = 0; i<9; i++) {
        let arr = new Array(9).fill(false);
        for (let j = 0; j < 9; j++) {
            if (arr[board[j][i]] == true) {
                console.log(i, j, arr[board[j][i]], board[j][i], "return false by column")
                return false;
            }
            if (board[j][i] != ".") {
                arr[board[j][i]] = true;
            }
        }
    };


    // 寻找起始点然后开启3*3小循环
    for (let a = 0; a < 9; a += 3){
        for (let b = 0; b < 9; b+= 3) {
            let arr = new Array(9).fill(false);
            for (let i = a; i < a+3; i++) {
                for (let j = b; j < b+3; j++) {
                    if (arr[board[i][j]] == true) {
                        console.log(i, j, "return false by group")
                        return false;
                    }
                    if (board[i][j] != ".") {
                        arr[board[i][j]] = true;
                    }
                }
            }
        }
    }
    return true;
};
```

- question: lc73 矩阵置零 link: https://leetcode.cn/problems/set-matrix-zeroes/
    - answer:

```python
# python code
# 利用自身数组的第一行与第一列记录是否需要变为零，同时需要两个变量来记录第一行与第一列是否需要变为全零。进阶版本可以只使用一个变量来记录但过程不是很清晰。
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        row = len(matrix)
        col = len(matrix[0])

        col_flag, row_flag = True, True

        for i in range(row):
            if matrix[i][0] == 0:
                col_flag = False
                break

        for i in range(col):
            if matrix[0][i] == 0:
                row_flag = False
                break

        for i in range(1, row):
            for j in range(1, col):
                if matrix[i][j] == 0:
                    matrix[i][0] = 0
                    matrix[0][j] = 0

        for i in range(1, row):
            for j in range(1, col):
                if matrix[i][0] == 0 or matrix[0][j] == 0:
                    matrix[i][j] = 0

        if not row_flag:
            for i in range(col):
                matrix[0][i] = 0

        if not col_flag:
            for i in range(row):
                matrix[i][0] = 0
```

```
/**
 * @param {number[][]} matrix
 * @return {void} Do not return anything, modify matrix in-place instead.
 */
var setZeroes = function(matrix) {
    let m = matrix.length;
    let n = matrix[0].length;
    let zerosIndexI = new Set();
    let zerosIndexJ = new Set();
    // let arr = Array.from(new Array(m), item => new Array(n).fill(false));
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            // arr[i][j] = true;
            if (matrix[i][j] == 0) {
               zerosIndexI.add(i)
               zerosIndexJ.add(j)
            }
        }
    }
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            if (zerosIndexI.has(i) || zerosIndexJ.has(j)) {
                matrix[i][j] = 0;
            }
        }
    }
};
```

- question: lc54 剑指29 螺旋矩阵 link: https://leetcode.cn/problems/spiral-matrix/
    - answer:

```
# python code
# 外层循环，大循环内有四次小循环，代表最外层。
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:

        n = len(matrix)
        m = len(matrix[0])

        ans = []
        start = 0

        while len(ans) < n*m:

            for i in range(start, m-start):
                ans.append(matrix[start][i])
            if len(ans) == n*m:
                break

            for i in range(1+start, n-start):
                ans.append(matrix[i][m-start-1])
            if len(ans) == n*m:
                break

            for i in reversed(range(start, m-start-1)):
                ans.append(matrix[n-start-1][i])
            if len(ans) == n*m:
                break

            for i in reversed(range(start+1, n-start-1)):
                ans.append(matrix[i][start])

            start += 1

        return ans
```

```
/**
 * @param {number[][]} matrix
 * @return {number[]}
 */
var spiralOrder = function(matrix) {
    let res =[];
    let m = matrix.length;
    let n = matrix[0].length;
    let row = 0;
    let column = 0;
    let total = m*n;
    console.log(total)
    directionIndex = 0;
    let visited = new Array(m).fill(0).map(() => new Array(n).fill(false));
    // 右下左上
    let directions = [[0,1], [1,0], [0,-1], [-1, 0]];
    for (let i = 0; i < total; i++) {
        visited[row][column] = true;
        console.log(row, column, matrix[row][column])
        res.push(matrix[row][column])
        let Nrow = row + (directions[directionIndex][0]);
        let Ncolumn = column +  (directions[directionIndex][1]);
        if (Ncolumn >= n || Ncolumn <0 || Nrow >= m || Nrow < 0 || visited[Nrow][Ncolumn]) {
            directionIndex = (directionIndex + 1) % 4;

        }
        row = row + (directions[directionIndex][0]);
        column = column +  (directions[directionIndex][1]);
    }

    return res;
};
```

- question: lc59 螺旋矩阵二 link: https://leetcode.cn/problems/spiral-matrix-ii/
    - answer:

```python
# python code
# 类似 lc54 但这个是方阵，所以不用做if检测。
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        matrix = [[0]*n for _ in range(n)]
        init = 1
        start = 0

        while init <= n*n:

            for i in range(start, n-start):
                matrix[start][i] = init
                init += 1

            for i in range(1+start, n-start):
                matrix[i][n-start-1] = init
                init += 1

            for i in reversed(range(start, n-start-1)):
                matrix[n-start-1][i] = init
                init += 1

            for i in reversed(range(start+1, n-start-1)):
                matrix[i][start] = init
                init += 1

            start += 1

        return matrix
```

```javascript
/**
 * @param {number} n
 * @return {number[][]}
 */
var generateMatrix = function(n) {
    let res = [];
    let matrix = [];
    let row = 0;
    let column = 0;
    let total =n*n;
    directionIndex = 0;
    for (let i = 0; i < n; i++) {
        matrix.push(new Array());
        for (let j = 0; j < n; j++) {
            matrix[i].push(0);
        }
    }
    let visited = new Array(n).fill(0).map(() => new Array(n).fill(false));
    // 右下左上
    let directions = [[0,1], [1,0], [0,-1], [-1, 0]];
    for (let i = 0; i < total; i++) {
        visited[row][column] = true;
        matrix[row][column] = i+1;
            console.log(matrix)
        let Nrow = row + (directions[directionIndex][0]);
        let Ncolumn = column +  (directions[directionIndex][1]);
        if (Nrow >= n || Ncolumn >= n || Ncolumn <0  || Nrow < 0 || visited[Nrow][Ncolumn]) {
            directionIndex = (directionIndex + 1) % 4;

        }
        row = row + (directions[directionIndex][0]);
        column = column +  (directions[directionIndex][1]);
    }

    for (let i = 0; i<n; i++) {
        res.push(new Array())
        for (let j = 0; j < n; j++) {
            res[i].push(matrix[i][j]);
        }
    }

    return res;
};
```

- question: lc498 对角线遍历 link: https://leetcode.cn/problems/diagonal-traverse/
    - answer:

```
# python code
# 对角线的数量为n + m - 1，当对角线为偶数时，从下往上遍历，当对角线为奇数时，从上往下遍历。当i为偶数且小于行数时，起始点为（i，0），当i大于行数时，起始点为（行数-1，i-(m-1)）。
class Solution:
    def findDiagonalOrder(self, mat: List[List[int]]) -> List[int]:
        m = len(mat)
        n = len(mat[0])

        ans = []

        for i in range(n + m - 1):
            if i % 2:
                if i < n:
                    x = 0
                    y = i
                else:
                    x = i - n + 1
                    y = n - 1
                while x < m and y >= 0:
                    ans.append(mat[x][y])
                    x += 1
                    y -= 1
            else:
                if i < m:
                    x = i
                    y = 0
                else:
                    y = i - m + 1
                    x = m - 1
                while x >= 0 and y < n:
                    ans.append(mat[x][y])
                    x -= 1
                    y += 1

        return ans
```

```
/**
 * @param {number[][]} mat
 * @return {number[]}
 */
//利用x, y 遍历数组。因为对角线只有两个方向，我们就用两个方向的情况讨论。方向1为右上方向，他的终止条件是碰到上墙壁和有墙壁。方向2为左下方向，他的终止条件为碰到下墙壁和做墙壁。

var findDiagonalOrder = function(mat) {
    let direction = 1;
    let m = mat.length;
    let n = mat[0].length;
    let total = m * n
    let x = 0;
    let y = 0;
    let res = [];
    for (let i = 0; i<total; i++) {
        res.push(mat[y][x]);
        if (direction == 1) {
            if (x + 1 == n) {
                y++;
                direction = 2;
            } else if (y - 1 < 0){
                x++;
                direction = 2;
            } else {
                x++;
                y--;
            }
        } else {
            if (y + 1 == m) {
                x++;
                direction = 1;
            } else if (x - 1 < 0){
                y++;
                direction = 1;
            } else {
                y++;
                x--;
            }
        }
    }
    return res;
};
```

- question:lc118 杨辉三角 link: https://leetcode.cn/problems/pascals-triangle/
    - answer:

```
# python code
# 每一行第0和最后一个元素都为1，其他元素都为上一行的第j-1与j项相加之和。第n行m个数用公式为Cmn = n! / m!(n-m)! = cm-1n * ((n-m+1) / m)
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        ans = []

        for i in range(numRows):
            temp = []
            for j in range(i + 1):
                if j == 0 or j == i:
                    temp.append(1)
                else:
                    temp.append(ans[i-1][j-1]+ans[i-1][j])
            ans.append(temp)

        return ans
```

```
/**
 * @param {number} numRows
 * @return {number[][]}
 */
var generate = function(numRows) {
    let res = new Array(numRows);
    for (let i = 0; i < numRows; i++) {

        res[i] = new Array(i+1);
        res[i][0] = 1;
        res[i][i] = 1;
        console.log(res);
        for (let j = 1; j < i; j++) {
            res[i][j] = res[i - 1][j] + res[i - 1][j - 1];
        }
    }
    return res;
};
```

- question: lc119 杨辉三角二 link: https://leetcode.cn/problems/pascals-triangle-ii/
    - answer:

```
# python code
# 代入公式: ans[i] = ans[i-1] * (n-i+1) / i
class Solution:
    def getRow(self, rowIndex: int) -> List[int]:
        ans = [1] * (rowIndex + 1)

        for i in range(1, rowIndex + 1):

            ans[i] = int(ans[i-1] * (rowIndex - i + 1) / i)

        return ans
```

```
/**
 * @param {number} rowIndex
 * @return {number[]}
 */
var getRow = function(rowIndex) {
    let res = [];
    for (let i = 0; i <= rowIndex; i++) {
        let row = new Array(i+1);
        row[0] = 1;
        row[row.length - 1] = 1;

        for (let j = 1; j < i; j++) {
            row[j] = res[i - 1][j] + res[i - 1][j - 1]
        }
        if (i == rowIndex) {
            return row
        }
        res.push(row);
    }
};
```

字符串操作

- question: lc28 实现 strStr() link: https://leetcode.cn/problems/find-the-index-of-the-first-occurrence-in-a-string/
    - answer:

```
# python code
# 匹配过程中，检测是否有头部，匹配失败后直接移动到头部位置而不是一步一步移动。
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        nh = len(haystack)
        nn = len(needle)

        if nn >= nh:
            return 0 if needle == haystack else -1

        start = 0
        nextstart = 0

        flag = False
        first = True

        while start < nh:
            if haystack[start] == needle[0]:
                j = 0
                while j < nn:
                    if j+start < nh and haystack[j+start] == needle[j]:
                        if haystack[j+start] == needle[0] and first:
                            first = False
                            nextstart = j+start
                        flag = True
                        j += 1
                    else:
                        if not first:
                            start = nextstart
                        else:
                            start = j+start-1
                            nextstart = start

                        first = True
                        flag = False
                        break
                if flag:
                    return start

            start += 1

        return -1
```

```
/**
 * @param {string} haystack
 * @param {string} needle
 * @return {number}
 */
var strStr = function(haystack, needle) {
    let hl = haystack.length;
    let nl = needle.length;
    let a = 0;
    let b = 0;
    while (a <= hl) {
        if (haystack[a] == needle[b]) {
            if (b == nl - 1) {
                return a - b;
            }
            b++;
        } else {
            a = a - b;
            b = 0
        }
        a++;
    }
    return -1;
};
```

- question: lc344 反转字符串 link: https://leetcode.cn/problems/reverse-string/
    - answer:

```
# python code
# 赋值的方式可以通过双指针的形式首尾进行交换，或者反向切片并逐个赋值。
class Solution:
    def reverseString(self, s: List[str]) -> None:
        """
        Do not return anything, modify s in-place instead.
        """

        n = len(s)

        right, left = 0, n-1

        while right < left:
            s[right], s[left] = s[left], s[right]
            right += 1
            left -= 1

        return s
        # s[:] = s[::-1]
        # return s
```

```
/**
 * @param {string} s
 * @return {number}
 */
var lengthOfLastWord = function(s) {
    let space = " ".charCodeAt();
    let res = 0
    let spaceNum = 0;
    let shouldReturn = s[s.length - 1] == " "? false : true;
    for (let i = s.length - 1; i >= 0; i--) {
        console.log(s[i
        if (s[i] == " ") {
            if ( shouldReturn) {
                 return  res;
            }
            spaceNum ++;
            continue;
        }
        shouldReturn = true;
        res++;
    }
    return s.length - spaceNum;
};


// 使用前后指针 后指针end记录空位，前指针start记录满足要求的单词，详见得到res
// var lengthOfLastWord = function(s) {
//     let end = s.length - 1;
//     while(end >= 0 && s[end] == ' ') end--;
//     if(end < 0) return 0;
//     let start = end;
//     while(start >= 0 && s[start] != ' ') start--;
//     return end - start;
// };
```

- question: lc345 反转字符串中的元音字母 link: https://leetcode.cn/problems/reverse-vowels-of-a-string/
    - answer:

```
# python code
# str无法直接更改，需要将str先变为list再去更改，str.lower()变小写，str.upper()变大写。in检查某个元素是否在list中。
class Solution:
    def reverseVowels(self, s: str) -> str:
        n = len(s)
        right, left = 0, n-1

        s = list(s)

        yuan = ['a', 'e', 'i', 'o', 'u']

        while right < left:
            if s[right].lower() in yuan and s[left].lower() in yuan:
                s[right], s[left] = s[left], s[right]
                right += 1
                left -= 1
            elif s[right].lower() in yuan:
                left -= 1
            elif s[left].lower() in yuan:
                right += 1
            else:
                right += 1
                left -= 1

        return "".join(s)
```

```
/**
 * @param {string} s
 * @return {string}
 */
var reverseVowels = function(s) {
    let set = new Set(['a','e','i','o','u','A','E','I','O','U']);
    let left = 0;
    let right = s.length - 1;
    let res = s.split("");
    while (left < right) {
        console.log(left, right)
        while ((!set.has(s[left]))){
            left++;
            if (left >= right) {
                return res.join("");
            }
        }
        while ((!set.has(s[right]))){
            right--;
            if (left >= right) {
                return res.join("");
            }
        }
        console.log("fin currect: ", left, right)
        console.log(res[left])
        res[left] = s.charAt(right);
        console.log(res[left])
        res[right] = s.charAt(left);
        console.log(res)
        left++;
        right--;
    }
    return res.join("");
};

// 使用 arr.indexOf()
var reverseVowels = function(s) {
    var result;
    var temp
    var arr = ['a','e','i','o','u','A','E','I','O','U']
    var t = s.split('')
    var i = 0; j = t.length


    while(i<j){
        while(i<j && arr.indexOf(t[i]) == -1){
            i++;
        }
        while(i<j && arr.indexOf(t[j]) == -1){
            j--
        }
        temp = t[i];
        t[i] = t[j];
        t[j] = temp

        i++;
        j--
    }
    result = t.join('');
    return result
};
```

- question: lc1119 删去字符串中的元音（vip） link:
    - answer:

```
# python code
```

```

```

- question: lc557 反转字符串中的单词 link: https://leetcode.cn/problems/reverse-words-in-a-string-iii/
    - answer:
    - For python, the split() method splits a string into a list. You can specify the separator, default separator is any whitespace.
    - For python, The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)
    - For python, string can not be changed directly.

```
# python code
class Solution:
    def reverseWords(self, s: str) -> str:
        # allstr = s.split()

        # ans = allstr[0][::-1]

        # for i in range(1, len(allstr)):
        #     ans += " " + allstr[i][::-1]

        # return ans

        return " ".join([word[::-1] for word in s.split(" ")])
```

```
/**
 * @param {string} s
 * @return {string}
 */
var reverseWords = function(s) {
    let arr = s.split(" ");
    for (let i = 0; i < arr.length; i++) {
        arr[i] = arr[i].split("").reverse().join("")
    }
    return arr.join(" ");
};
```

- question: lc541 反转字符串 link: https://leetcode.cn/problems/reverse-string-ii/
    - answer:

```
# python code
# 2k为步长，每次反转i: i+2k，python切片时，即使末尾超出数组，也不会报错，只会遍历到尾部。
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        t = list(s)
        for i in range(0, len(t), 2 * k):
            t[i: i + k] = reversed(t[i: i + k])
        return "".join(t)
```

```
/**
 * @param {string} s
 * @return {string}
 */
var reverseWords = function(s) {
    let arr = s.split(" ");
    for (let i = 0; i < arr.length; i++) {
        arr[i] = arr[i].split("").reverse().join("")
    }
    return arr.join(" ");
};
```

- question: lc58 最后一个单词的长度 link: https://leetcode.cn/problems/length-of-last-word/
    - answer:

```
# python code
# 按空格划分，然后返回最后一个元素，index为-1.
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        return len(s.split()[-1])
```

```
/**
 * @param {string} s
 * @return {number}
 */
var lengthOfLastWord = function(s) {
    let space = " ".charCodeAt();
    let res = 0
    let spaceNum = 0;
    let shouldReturn = s[s.length - 1] == " "? false : true;
    for (let i = s.length - 1; i >= 0; i--) {
        console.log(s[i])
        if (s[i] == " ") {
            if ( shouldReturn) {
                 return  res;
            }
            spaceNum ++;
            continue;
        }
        shouldReturn = true;
        res++;
    }
    return s.length - spaceNum;
};


// 使用前后指针 后指针end记录空位，前指针start记录满足要求的单词，详见得到res
// var lengthOfLastWord = function(s) {
//     let end = s.length - 1;
//     while(end >= 0 && s[end] == ' ') end--;
//     if(end < 0) return 0;
//     let start = end;
//     while(start >= 0 && s[start] != ' ') start--;
//     return end - start;
// };
```

- question: lc165 比较版本号 link: https://leetcode.cn/problems/compare-version-numbers/
    - answer:

```python
# python code
# 按"."划分后先比较相同长度的信息，再比较多出来的信息。
class Solution:
    def compareVersion(self, version1: str, version2: str) -> int:
        v1 = version1.split(".")
        v2 = version2.split(".")

        n1 = len(v1)
        n2 = len(v2)

        for i in range(min(n1,n2)):
            if int(v1[i]) > int(v2[i]):
                return 1
            elif int(v1[i]) < int(v2[i]):
                return -1

        if n1 > n2:
            for i in range(n2, n1):
                if int(v1[i]) != 0:
                    return 1
        elif n1 < n2:
            for i in range(n1, n2):
                if int(v2[i]) != 0:
                    return -1

        return 0
```

```javascript
/**
 * @param {string} version1
 * @param {string} version2
 * @return {number}
 */
var compareVersion = function(version1, version2) {
    let v1Arr = version1.split(".");
    let v2Arr = version2.split(".");
    let i = 0;
    while (parseInt(v1Arr[i]) == parseInt(v2Arr[i])) {
        i++;
    }
    let minLen = v1Arr.length > v2Arr.length ? v2Arr.length:v1Arr.length;
    let maxArr = v1Arr.length > v2Arr.length ? v1Arr:v2Arr;
    if (i == minLen) {
        if (v1Arr.length == v2Arr.length) return 0;
        while (i < maxArr.length) {
            if(parseInt(maxArr[i]) != 0) {
                return v1Arr.length > v2Arr.length ? 1:-1
            }
            i++
        }
        return 0
    }
    return parseInt(v1Arr[i]) > parseInt(v2Arr[i]) ? 1 : -1;
};
```

- question: lc12 整数转罗马数字 link: https://leetcode.cn/problems/integer-to-roman/
    - answer:

```python
# python code
# 拆除特殊字符进行特殊处理。
class Solution:
    def intToRoman(self, num: int) -> str:
        mapR = ['I', 'X', 'C', 'M']
        mapS = ['V', 'L', 'D']
        map4 = ['IV', 'XL', 'CD']
        map9 = ['IX', 'XC', 'CM']
        ans = []

        for i in reversed(range(4)):
            n = num//(10**i)
            num = num - n*(10**i)
            if n < 4:
                ans.append("".join([mapR[i]]*n))
            elif n == 4:
                ans.append(map4[i])
            elif n > 4 and n < 9:
                ans.append(mapS[i] + "".join([mapR[i]]*(n-5)))
            elif n == 9:
                ans.append(map9[i])

        return "".join(ans)
```

```
//方法一：列举全部大单位，由大到小逐位相减
/**
 * @param {number} num
 * @return {string}
 */
var intToRoman = function(num) {
    const valueSymbols = [[1000, "M"], [900, "CM"], [500, "D"], [400, "CD"], [100, "C"], [90, "XC"], [50, "L"], [40, "XL"], [10, "X"], [9, "IX"], [5, "V"], [4,
"IV"], [1, "I"]];
    const roman = [];
    for (const [value, symbol] of valueSymbols) {
        while (num >= value) {
            num -= value;
            roman.push(symbol);
        }
        if (num == 0) {
            break;
        }
    }
    return roman.join('');
};


//方法一：列举个十百千全部元素，由大到小逐位放置
var intToRoman = function(num) {
    const thousands = ["", "M", "MM", "MMM"];
    const hundreds = ["", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"];
    const tens     = ["", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"];
    const ones     = ["", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"];

    const roman = [];
    roman.push(thousands[Math.floor(num / 1000)]);
    roman.push(hundreds[Math.floor(num % 1000 / 100)]);
    roman.push(tens[Math.floor(num % 100 / 10)]);
    roman.push(ones[num % 10]);
    return roman.join('');
};
```

- question: lc13 罗马数字转整数 link: https://leetcode.cn/problems/roman-to-integer/
    - answer:

```
# python code
# 当前罗马字符如果大于等于下一个罗马字符，则结果加上对应数值，否则减去对应数值。
class Solution:
    def romanToInt(self, s: str) -> int:
        dic = {
            "M" : 1000,
            "D" : 500,
            "C" : 100,
            "L" : 50,
            "X" : 10,
            "V" : 5,
            "I" : 1
        }

        ans = 0

        for i in range(len(s) - 1):
            if dic[s[i]] >= dic[s[i+1]]:
                ans += dic[s[i]]
            else:
                ans -= dic[s[i]]

        ans += dic[s[-1]]

        return ans
```

```
/**
 * @param {string} s
 * @return {number}
 */
var romanToInt = function(s) {
    let keys = ["I", "V", "X", "L", "C", "D", "M"];
    let values = [1, 5, 10, 50, 100, 500, 1000];
    let res =0;
    for (let i = 0; i < s.length; i++) {
        if (keys.indexOf(s.charAt(i)) >= keys.indexOf(s.charAt(i+1))) {
            res+=values[keys.indexOf(s.charAt(i))];
        } else {
            res-=values[keys.indexOf(s.charAt(i))];
        }
    }
    return res;
};


//除此之外，还可以使用function进行对数组的存储
function getchar() {
        switch(ch) {
            case 'I': return 1;
            case 'V': return 5;
            case 'X': return 10;
            case 'L': return 50;
            case 'C': return 100;
            case 'D': return 500;
            case 'M': return 1000;
            default: return 0;
        }
    }
```

- question: lc38 外观数列 link: https://leetcode.cn/problems/count-and-say/
    - answer:

```
# python code
# 计数统计，循环结束后需要再插值一次。
class Solution:
    def countAndSay(self, n: int) -> str:
        num = "1"
        if n == 1:
            return num

        for j in range(n-1):
            count = 1
            cur = num[0]
            ans = ""

            for i in range(1, len(num)):
                if num[i] == cur:
                    count += 1
                else:
                    ans += str(count) + str(cur)
                    cur = num[i]
                    count = 1

            ans += str(count) + str(cur)
            num = ans

        return num
```

```
// 好题
/**
 * @param {number} n
 * @return {string}
 */
var countAndSay = function(n) {

    let pre = "1";
    for (let i = 1; i < n; i++) {
        let start = 0;
        let pos = 0;
        let line = ""
        while (pos < pre.length) {
            while (pos < pre.length && pre[pos] == pre[start]) {
                pos++;
            }
            line = line + (pos - start) + pre[start]
            start = pos;
        }
        pre = line;
    }
    return pre;
};
```

- question: lc6 Z字形变换 link: https://leetcode.cn/problems/zigzag-conversion/
    - answer:

```
# python code
# 初始化n行string，每次i到达numRows-1与0时，改变方向.
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        if numRows <= 1:
            return s

        ans = ["" for i in range(numRows)]
        # 初始化n个字符串

        i = 0
        flag = -1
        # 移动方向
        for c in s:
            ans[i] += c
            if i == numRows-1 or i == 0:
                flag = -flag
                # 改变方向
            i += flag

        return "".join(ans)
```

```
/**
 * @param {string} s
 * @param {number} numRows
 * @return {string}
 */
var convert = function(s, numRows) {
    let arr = new Array(numRows).fill("");
    let flag = 1;
    let index = 0;
    if (numRows == 1) {
        return s;
    }
    for (let i = 0; i < s.length; i++) {
        arr[index]+=s.charAt(i);
        index += flag;
        if (index == numRows - 1 || index == 0) {
            flag = -flag;
        }
    }
    return arr.join('')

};
```

数学相关

- question: lc7 整数反转 link: https://leetcode.cn/problems/reverse-integer/
    - answer:

```python
# mod 10 and reverse the numbers
# or convert to string and reversed
class Solution:
    def reverse(self, x: int) -> int:
        if x < -2**31 or x > 2**31 - 1:
            return 0

        flag = True if x > 0 else False

        x = abs(x)

        ans = 0

        while x > 0:
            ans = ans*10 + x%10
            x //= 10

        if ans < -2**31 or ans > 2**31 - 1:
            return 0

        return ans if flag else -ans
```

```javascript
/**
 * @param {number} x
 * @return {number}
 */
var reverse = function(x) {
    let res = 0;
    let absx = x >= 0 ? x : -x;
    while (absx != 0) {
        let temp = absx%10;
        absx = Math.floor(absx / 10)
        res = res * 10 + temp;
        if(res > Math.pow(2, 31) - 1 || res < Math.pow(-2, 31)) return 0;
    }
    return x >= 0 ? res : -res;
};
```

- question: lc9 回文数 link: https://leetcode.cn/problems/palindrome-number/
    - answer:

```python
# 双指针头尾同时检测并向内侧移动（允许使用string的时候）
# 不允许使用string时，需要用数学方法得到反转后的数字，检查是否相等。
class Solution:
    def isPalindrome(self, x: int) -> bool:
        # s = str(x)
        # right, left = len(s)-1, 0

        # while left < right:
        #     if s[left] != s[right]:
        #         return False

        #     left += 1
        #     right -= 1

        # return True

        if x < 0:
            return False

        n = x

        ans = 0

        while n > 0:
            ans = ans*10 + n%10
            n = n//10

        return True if ans == x else False
```

```javascript
/**
 * @param {number} x
 * @return {boolean}
 */
var isPalindrome = function(x) {
    if (x < 0) return false
    let x2 = x.toString();
    let left = 0;
    let right = x2.length - 1
    while (left < right) {
        if (x2.charAt(right--) != x2.charAt(left++)) return false;
    }
    return true
};
```

- question: lc989 数组形式的整数加法 link: https://leetcode.cn/problems/add-to-array-form-of-integer/
    - answer:

```
逐位相加解题思路
while ( A 没完 || B 没完)
    A 的当前位
    B 的当前位
    和 = A 的当前位 + B 的当前位 + 进位carry
    当前位 = 和 % 10;
    进位 = 和 / 10;
```

```python
# 不转换为字符串，则需要将k转为list形式，相加有进位就进位。
class Solution:
    def addToArrayForm(self, num: List[int], k: int) -> List[int]:
        n = len(num)

        lk = []
        num[:] = num[::-1]

        while k > 0:
            lk.append(k%10)
            k //= 10

        if len(lk) > n:
            num.extend([0]*(len(lk)-n))
            n = len(lk)
        else:
            lk.extend([0]*(n-len(lk)))

        ans = [0] * (n+1)

        for i in range(n):
            ans[i] += lk[i] + num[i]
            if ans[i] >= 10:
                ans[i] %= 10
                ans[i+1] += 1

        if ans[-1] == 0:
            ans.pop()

        return ans[::-1]

# 转换为字符串后，连接再转为int然后相加最后拆成list返回。
        # n = int("".join([str(c) for c in num])) + k

        # ans = []

        # while n > 0:
        #     ans.append(n%10)
        #     n //= 10

        # return ans[::-1]
```

```javascript
/**
 * @param {number[]} num
 * @param {number} k
 * @return {number[]}
 */
var addToArrayForm = function(num, k) {
    let karr = [];
    let res = []
    while (k != 0) {
        let temp = k % 10;
        karr.push(temp);
        k = Math.floor(k / 10);
    }
    karr.reverse();
    let i = num.length - 1;
    let j = karr.length - 1;
    let carry = 0;
    while (i >= 0 || j >= 0) {
        let x = i >= 0 ? num[i--] : 0;
        let y = j >= 0 ? karr[j--] : 0;
        let sum = carry + x + y
        res.push(sum%10);
        carry = Math.floor(sum/10);
    }
    if (carry == 1) {
        res.push(1);
    }
    return res.reverse();

};
```

- question: lc66 加1 link: https://leetcode.cn/problems/plus-one/
    - answer:

```python
# 翻转后先hardcode给第一项加1，之后遍历0~n-2项，大于等于10则进位。跳出循环后，如果最后一位大于等于10则在末尾append（1）。
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:

        digits[:] = digits[::-1]

        digits[0] += 1

        for i in range(len(digits) - 1):
            if digits[i] >= 10:
                digits[i] %= 10
                digits[i+1] += 1

        if digits[-1] >= 10:
            digits[-1] %= 10
            digits.append(1)

        return digits[::-1]
```

```
/**
 * @param {number[]} digits
 * @return {number[]}
 */
var plusOne = function(digits) {
    let i = digits.length - 1;
    let carry = 1;
    while (i >= 0) {
        let sum = carry + digits[i];
        digits[i] = sum % 10;
        carry = Math.floor(sum / 10);
        i--;
    }
    if (carry != 0) {
        digits.unshift(1);
    }
    return digits
};
```

- question: lc415 字符串相加 link: https://leetcode.cn/problems/add-strings/
    - answer:

```
# 无法转为int时，用ord(str) - ord('0')
class Solution:
    def addStrings(self, num1: str, num2: str) -> str:
        i = len(num1) - 1
        j = len(num2) - 1
        carry = 0

        ans = ""

        while i >= 0 or j >= 0 or carry != 0:
            x = ord(num1[i]) - ord('0') if i >= 0 else 0
            y = ord(num2[j]) - ord('0') if j >= 0 else 0

            ans += chr((x + y + carry)%10 + ord('0'))
            carry = (x + y + carry)//10

            i -= 1
            j -= 1

        return ans[::-1]
```

```
// 诸位相加思路
/**
 * @param {string} num1
 * @param {string} num2
 * @return {string}
 */
var addStrings = function(num1, num2) {
    let res = [];
    let carry = 0;
    let i = num1.length - 1;
    let j = num2.length - 1;
    while ( i >= 0 || j >= 0) {
        let x = i >= 0 ? parseInt(num1.charAt(i--)) % 10 : 0;
        let y = j >= 0 ? parseInt(num2.charAt(j--)) % 10 : 0;
        let sum = carry + x + y;
        carry = Math.floor(sum/10)
        res.push(sum%10)
    }
    if ( carry == 1) {
        res.push(1);
    }
    return res.reverse().join("");
};
```

- question: lc67 剑指002 二进制求和 link: https://leetcode.cn/problems/add-binary/
    - answer:

```
# 类似上一题,把进位改成2就行,当前位置为result%2, 进位为result//2.
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        i = len(a) - 1
        j = len(b) - 1

        carry = 0

        ans = ""

        while i >= 0 or j >= 0 or carry != 0:
            x = ord(a[i]) - ord('0') if i >= 0 else 0
            y = ord(b[j]) - ord('0') if j >= 0 else 0

            ans += chr((x + y + carry)%2 + ord('0'))
            carry = (x + y + carry)//2

            i -= 1
            j -= 1

        return ans[::-1]
```

```
//诸位相加细节：
    1、i j length要-1
    2、别忘了i-- j--
    3、别忘了特殊进位
/**
 * @param {string} a
 * @param {string} b
 * @return {string}
 */
var addBinary = function(a, b) {
    let res = [];
    let carry = 0;
    let i = a.length - 1;
    let j = b.length - 1;
    while (i >= 0 || j >= 0) {
        let x = i >= 0 ? Number(a.charAt(i--)) : 0;
        let y = j >= 0 ? Number(b.charAt(j--)) : 0;
        let sum = x + y + carry;
        carry = Math.floor(sum / 2);
        res.push(sum % 2);
    }
    if (carry != 0) {
        res.push(1);
    }
    return res.reverse().join("")
};
```

- question: lc2 两数相加 link: https://leetcode.cn/problems/add-two-numbers/
    - answer:

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:
        ans = head = ListNode(0)
        carry = 0
        while l1 or l2 or carry:
            x = l1.val if l1 else 0
            y = l2.val if l2 else 0
            result = x + y + carry
            head.next = ListNode(result%10)
            head = head.next
            carry = result//10
            l1 = l1.next if l1 else l1
            l2 = l2.next if l2 else l2

        return ans.next
```

```
初识链表：
    遍历链表使用l1 = l1.next
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} l1
 * @param {ListNode} l2
 * @return {ListNode}
 */
var addTwoNumbers = function(l1, l2) {
    let carry = 0;
    let res = new ListNode();
    let cur = res;
    while (l1 != null || l2 != null) {
        let x = l1 != null ? l1.val : 0
        let y = l2 != null ? l2.val : 0
        if (l1) l1 = l1.next
        if (l2) l2 = l2.next
        let sum = carry + x + y;
        carry =Math.floor(sum/10);
        cur.next = new ListNode(sum%10);
        cur = cur.next
    }
    if (carry == 1) {
        cur.next = new ListNode(1)
    }
    return res.next;
};
```

- question: lc43 字符串相乘 link: https://leetcode.cn/problems/multiply-strings/
    - answer:

```
# 利用竖式乘法规则, 小数每一位与大数相乘再求和.
class Solution:
    def multiply(self, num1: str, num2: str) -> str:
        if num1 == "0" or num2 == "0":
            return "0"

        if len(num1) < len(num2):
            num1, num2 = num2, num1

        n1 = len(num1)
        n2 = len(num2)

        result = 0

        for i in range(n2):
            temp = 0
            for j in range(n1):
                temp = temp + (ord(num2[n2-1-i]) - ord("0")) * (ord(num1[n1-1-j]) - ord("0"))*(10**j)

            result += temp*(10**i)

        ans = ""

        while result > 0:
            ans += chr(result%10 + ord("0"))
            result //= 10

        return ans[::-1]
```

```
```

- question: lc204 计数质数 link: https://leetcode.cn/problems/count-primes/
    - answer:

```
# 筛选法,先初始化一个长度为n,且都为1的数组,当当前数组值为1时,将所有i的倍数的值都赋值为0,最后剩下的1就都是质数.
class Solution:
    def countPrimes(self, n: int) -> int:
        if n < 2:
            return 0

        isPrime = [1] * n
        isPrime[0] = isPrime[1] = 0

        for i in range(2, int(n ** 0.5) + 1):
            if isPrime[i]:
                for j in range(i*i, n, i):
                    isPrime[j] = 0

        return sum(isPrime)
```

```
/**
 * @param {number} n
 * @return {number}
 */
var countPrimes = function(n) {
    let arr = new Array(n).fill(1);
    let res = 0;
    for (let i = 2; i * i <= n; ++i) {
        if (arr[i]) {
            res ++;
            for (let j = i * i; j < n; j += i) {
                arr[j] = 0
            }
        }
    }
    return res;
};
```

- question: lc233 剑指43 数字1的个数 link: https://leetcode.cn/problems/number-of-digit-one/
    - answer:

```
```

```
var countDigitOne = function(n) {
    // mulk 表示 10^k
    // 在下面的代码中, 可以发现 k 并没有被直接使用到 (都是使用 10^k)
    // 但为了让代码看起来更加直观, 这里保留了 k
    let mulk = 1;
    let ans = 0;
    for (let k = 0; n >= mulk; ++k) {
        ans += (Math.floor(n / (mulk * 10))) * mulk + Math.min(Math.max(n % (mulk * 10) - mulk + 1, 0), mulk);
        mulk *= 10;
    }
    return ans;
};
```

- question: lc1232 缀点成线 link: https://leetcode.cn/problems/check-if-it-is-a-straight-line/
    - answer:

```
# 通过计算斜率是否相同来得到, 但需要注意不要用除法, 可以换成乘法:
# (yi-1 - yi)*(xi - xi+1) ?= (yi - yi+1) * (xi-1 - xi)
class Solution:
    def checkStraightLine(self, coordinates: List[List[int]]) -> bool:
        if len(coordinates) < 2:
            return False

        for i in range(1,len(coordinates)-1):
            if (coordinates[i-1][1] - coordinates[i][1]) * (coordinates[i][0] - coordinates[i+1][0]) != \
            (coordinates[i][1] - coordinates[i+1][1]) * (coordinates[i-1][0] - coordinates[i][0]):
                return False

        return True
```

```
/**
 * @param {number[][]} coordinates
 * @return {boolean}
 */
var checkStraightLine = function(coordinates) {
    let x0 = coordinates[0][0];
    let y0 = coordinates[0][1];
    let k = (coordinates[1][1] - y0) / (coordinates[1][0] - x0)
    if (x0 == 0 && coordinates[1][0] == 0) {
        for (let i = 1; i < coordinates.length; i++) {
            if (coordinates[i][0] != 0) return false;
        }
        return true
    }
    if (y0 == 0 && coordinates[1][1] == 0) {
        for (let i = 1; i < coordinates.length; i++) {
            if (coordinates[i][1] != 0) return false;
        }
        return true
    }
    for (let i = 1; i < coordinates.length; i++) {
        let dertax = coordinates[i][0] - x0;
        let dertay = coordinates[i][1] - y0;
        x0 = coordinates[i][0];
        y0 = coordinates[i][1];
        let kc = dertay / dertax;
        if (kc != k) {
            return false;
        }
    }
    return true;
};
```

位运算

按位与 &

按位与表示将两个数字转换为二进制数字，然后从低到高对末位比较，如果两个位都是1那么结果就是1，否则为0

```
// 1的二进制表示为: 00000000 00000000 00000000 00000001
// 3的二进制表示为: 00000000 00000000 00000000 00000011
// -----------------------------
// 1的二进制表示为: 00000000 00000000 00000000 00000001
console.log(1 & 3)     // 1


特殊情况:  n = n & n - 1会去掉二进制中存在的的最后一个1
```

按位或 |

按位或表示将两个数字转换为二进制数字，然后从低到高对末位比较，如果两个位只要有一个1，那么结果就为1，否则为0；

```
// 1的二进制表示为: 00000000 00000000 00000000 00000001
// 3的二进制表示为: 00000000 00000000 00000000 00000011
// -----------------------------
// 3的二进制表示为: 00000000 00000000 00000000 00000011
console.log(1 | 3)     // 3
```

按位异或(XOR) ^

如果对应两个操作位有且仅有一个1时结果为1，其他都是0。

```
// 1的二进制表示为: 00000000 00000000 00000000 00000001
// 3的二进制表示为: 00000000 00000000 00000000 00000011
// -----------------------------
// 2的二进制表示为: 00000000 00000000 00000000 00000010
console.log(1 ^ 3)     // 2
```

按位非(NOT)

~ 运算符是对位求反，1变0, 0变1，也就是求二进制的反码。

```
// 1的二进制表示为: 00000000 00000000 00000000 00000001
// 3的二进制表示为: 00000000 00000000 00000000 00000011
// -----------------------------
// 1反码二进制表示: 11111111 11111111 11111111 11111110
// 由于第一位（符号位）是1，所以这个数是一个负数。JavaScript 内部采用补码形式表示负数，即需要将这个数减去1，再取一次反，然后加上负号，才能得到这个负数对应的10进制值。
// -----------------------------
// 1的反码减1:      11111111 11111111 11111111 11111101
// 反码取反:        00000000 00000000 00000000 00000010
// 表示为10进制加负号: -2
console.log(~ 1)     // -2
```

左移（Left shift）<<

<<运算符使指定值的二进制数所有位都左移指定次数，其移动规则：丢弃高位，低位补0即按二进制形式把所有的数字向左移动对应的位数，高位移出(舍弃)，低位的空位补零。

```
// 1的二进制表示为: 00000000 00000000 00000000 00000001
// -----------------------------
// 2的二进制表示为: 00000000 00000000 00000000 00000010
console.log(1 << 1)     // 2
```

6. 有符号右移>>

> 该操作符会将指定操作数的二进制位向右移动指定的位数。向右被移出的位被丢弃，拷贝最左侧的位以填充左侧。由于新的最左侧的位总是和以前相同，符号位没有被改变。所以被称作"符号传播"。

// 1的二进制表示为: 00000000 00000000 00000000 00000001
// -----------------------------
// 0的二进制表示为: 00000000 00000000 00000000 00000000
console.log(1 >> 1) // 0

```
// 1的二进制表示为: 00000000 00000000 00000000 00000001
// -----------------------------
// 0的二进制表示为: 00000000 00000000 00000000 00000000
console.log(1 >> 1)      // 0
```

7. 无符号右移>>>

`>>>`该操作符会将第一个操作数向右移动指定的位数。向右被移出的位被丢弃，左侧用0填充。因为符号位变成了 0，所以结果总是非负的。（译注：即便右移 0 个比特，结果也是非负的。）

对于非负数，有符号右移和无符号右移总是返回相同的结果。例如， `9 >>> 2` 得到 `2` 和 `9 >> 2` 相同。

位运算的妙用

1. 使用&运算符判断一个数的奇偶

```
// 偶数 & 1 = 0
// 奇数 & 1 = 1
console.log(2 & 1)    // 0
console.log(3 & 1)    // 1
复制代码
```

1. 使用`~, >>, <<, >>>, |`来取整

```
console.log(~~ 6.83)    // 6
console.log(6.83 >> 0)  // 6
console.log(6.83 << 0)  // 6
console.log(6.83 | 0)   // 6
// >>>不可对负数取整
console.log(6.83 >>> 0)   // 6
复制代码
```

1. 使用`^`来完成值交换

```
var a = 5
var b = 8
a ^= b
b ^= a
a ^= b
console.log(a)   // 8
console.log(b)   // 5
复制代码
```

1. 使用`&, >>, |`来完成rgb值和16进制颜色值之间的转换

```
/**
 * 16进制颜色值转RGB
 * @param  {String} hex 16进制颜色字符串
 * @return {String}     RGB颜色字符串
 */
  function hexToRGB(hex) {
    var hexx = hex.replace('#', '0x')
    var r = hexx >> 16
    var g = hexx >> 8 & 0xff
    var b = hexx & 0xff
    return `rgb(${r}, ${g}, ${b})`
}

/**
 * RGB颜色转16进制颜色
 * @param  {String} rgb RGB进制颜色字符串
 * @return {String}     16进制颜色字符串
 */
function RGBToHex(rgb) {
    var rgbArr = rgb.split(/[^\d]+/)
    var color = rgbArr[1]<<16 | rgbArr[2]<<8 | rgbArr[3]
    return '#'+ color.toString(16)
}
// -------------------------------------------------
hexToRGB('#ffffff')               // 'rgb(255,255,255)'
RGBToHex('rgb(255,255,255)')      // '#ffffff'
```

- question: lc191 位1的个数 link: https://leetcode.cn/problems/number-of-1-bits/
    - answer:

```
# 三种方法实现。
# 第一，利用bin转为二进制字符串，计数其中的1
# 第二，因为n<2^32，只需要与上32次，每个位上都检测一次是否是1
# 第三，n & n-1 得到 n最后一位变为0后的数，例如6（110），5（101），6&5 = 4（100）
# 每次n & n-1 可以减少一个1，当n为0时，结束循环，记录循环次数则是1的个数。
class Solution:
    def hammingWeight(self, n: int) -> int:
        # bi = bin(n).strip("0b")

        # ans = 0

        # for c in bi:
        #     if c == "1":
        #         ans += 1

        # return ans

        # ans = 0
        # for i in range(32):
        #     if n & (1 << i):
        #         ans += 1
        # return ans

        ans = 0
        while n:
            n &= n-1
            ans += 1
        return ans
```

```
//每次n & n-1 可以减少一个1，当n为0时，结束循环，记录循环次数则是1的个数
/**
 * @param {number} n - a positive integer
 * @return {number}
 */
var hammingWeight = function(n) {
    let count = 0;
    while (n != 0) {
        n = n & n -1;
        count ++;
    }
    return count;
};
```

- question: lc461 汉明距离 link: https://leetcode.cn/problems/hamming-distance/
    - answer:

```
# 首先将x与y异或，得到的数每一位为两者不相同的非0位，最后统计1的个数，则是汉明距离。
class Solution:
    def hammingDistance(self, x: int, y: int) -> int:
        # x y异或
        a = x ^ y

        ans = 0
        while a:
            a &= a-1
            ans += 1

        return ans
```

```
// 首先使用异或获得一个展示全部不同的数a,然后统计1的个数
/**
 * @param {number} x
 * @param {number} y
 * @return {number}
 */
var hammingDistance = function(x, y) {
    let a = x ^ y;
    let count = 0;
    while (a) {
        a &= (a - 1)
        count++;
    }
    return count
};
```

- question: lc477 汉明距离总和 link: https://leetcode.cn/problems/total-hamming-distance/
    - answer:

```
# 如果两两之间求距离再求和会超时。
# 因为n < 2^32，所以可以用32位内每位的1与上每个数，记录有多少个在当前位置上是1，总数减掉是1的个数则得到是0的个数，用是1的个数乘上是0的个数则得到当前位能得到多少的距离，把每一位能得到的距离和求和则得到总的距离和。
class Solution:
    def totalHammingDistance(self, nums: List[int]) -> int:
        ans = 0

        n = len(nums)

        for i in range(32):
            count1 = 0
            for j in range(n):
                if nums[j] & (1 << i):
                    count1 += 1

            ans += count1 * (n - count1)

        return ans
```

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var totalHammingDistance = function (nums) {
    // 由于二进制数字中每一位是相互独立的，所以求n个数字二进制位不同的数量
    // 就等于  n个数字中在不同的二进制位上不同的数量 的总和。
    // 这里解释一下[二进制位不同数字]：其实就是一部分是1，一部分是0
    // 想象一下，现在我们需要计算四个数字的二级制中第二位不同的数量
    // 如果四个数字中 有两个数字为1，其余两个为0，
    // 那么在2这个二进制位上不同的数就有2*2=4个
    // 同理求出32个二进制位上面不同的个数，最终求和即可,代码如下
    let ans = 0, n = nums.length
    for (let i = 0; i < 32; i++) {
        let c = 0 //  n个数字在第i位是1的个数
        for (let x of nums) {
            if ((x >> i) & 1) c++
        }
        ans += c * (n - c)
    }
    return ans

};
```

- question: lc231 2的幂 link: https://leetcode.cn/problems/power-of-two/
    - answer:

```
# 一个数是2的幂说明其二进制表示中只有一个1。
# 有两种方式检测是否只有一个1。
# 第一种是消去最低位1的方法，检测结果是否为0。消去最低位的1使用: n & (n-1)
# 第二种，因为负数在二进制表示中是用补码表示。补码：每一位取反加1。如果是2的幂，则负数的高位符号位与正数相反，低位1的位置与正数一样，相与后得到正数本身。n & -n == n。
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        # return n > 0 and n & -n == n
        return n > 0 and n & (n - 1) == 0
```

```
/**
 * @param {number} n
 * @return {boolean}
 */
var isPowerOfTwo = function(n) {
    return n>0 && (n & (n - 1)) == 0;
};
```

- question: lc371 两整数之和 link: https://leetcode.cn/problems/sum-of-two-integers/
    - answer:

```
# 未进位相加结果可以用异或得到，进位信息用与及左移一位得到，两者不停相异或及更新进位，直到进位为0。python中整数无大小限制，因此需要人为限制，并且如果结果为负则需要转为补码。
class Solution:
    def getSum(self, a: int, b: int) -> int:
        MASK = 4294967296
        MAX = 2147483647
        MIN = 2147483648
        a %= MASK
        b %= MASK

        while b:
            carry = ((a & b) << 1) % MASK
            a = (a ^ b) % MASK
            b = carry

        return a if a <= MAX else ~((a ^ MIN) ^ MAX)
```

```
// 如何实现位运算的加法？使用三个元素

/**
 * @param {number} a
 * @param {number} b
 * @return {number}
 */
var getSum = function(a, b) {
    while (b!=0) {
        let carry = (a & b) << 1;
        a = a ^ b;
        b = carry;
    }
    return a;
};
```

- question: lc29 两数相除 link: https://leetcode.cn/problems/divide-two-integers/
    - answer:

```

```

```
/**
 * @param {number} dividend
 * @param {number} divisor
 * @return {number}
 */
var divide = function(dividend, divisor) {
    const MAX = Math.pow(2, 31) - 1;
    const MIN = -Math.pow(2, 31)
    if (dividend == MIN && divisor == -1) {
        return MAX;
    }
    let sign = dividend > 0 ^ divisor > 0;
    let res = 0;
    dividend = dividend > 0 ? - dividend : dividend;
    divisor = divisor > 0 ? - divisor : divisor;
    while (dividend <= divisor) {
        dividend -= divisor;
        res++;
    }
    return sign?-res:res;
};
```

- question: lc136 只出现一次的数字 link: https://leetcode.cn/problems/single-number/
    - answer:

```
# 找到不同的数字，因为每个数要不出现两次，要么出现一次，异或操作中，自己异或自己则会得到0，于是用一个数不停异或数组内的所有数，最后留下的那个数就是只出现一次的。
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        if len(nums) < 2:
            return nums[0]

        ans = nums[0]

        for i in range(1, len(nums)):
            ans ^= nums[i]

        return ans
```

```
// 相同的数做异或操作会得到0，因为其他数都有两个个，异或掉可得到0。
var singleNumber = function(nums) {
    let n = 0;
    for(let num of nums) {
        n ^= num;
    }
    return n;
};
```

- question: lc137 只出现一次的数字II link: https://leetcode.cn/problems/single-number-ii/
    - answer:

```python
# 出现三次，不能用异或处理。此时需要统计每一位1出现的次数mod3得到特殊的那个在这个位是不是1，最终在第32位时，如果是1，说明此数字是负数，需要减去最大的负数得到对应的负数。
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        n = len(nums)

        ans = 0

        for i in range(32):
            count3 = 0
            for j in range(n):
                if (1 << i) & nums[j]:
                    count3 += 1

            if count3%3:
                if i == 31:
                    # 如果32位是1，说明此数是负数，需要转为负数
                    ans -= (1 << i)
                else:
                    ans |= (1 << i)

        return ans
```

```javascript
// 使用js的map, 注意遍历map的方式为let [key, value] of map
/**
 * @param {number[]} nums
 * @return {number}
 */
var singleNumber = function(nums) {
    let map = new Map();
    for (let num of nums) {
        if (map.has(num)) {
            map.set(num, map.get(num) + 1)
        } else {
            map.set(num, 1);
        }
    }

    for (let [key, value] of map) {
        if (value == 1) {
            return key
        }
    }
};
```

- question: lc1318 或运算的最小翻转次数 link: https://leetcode.cn/problems/minimum-flips-to-make-a-or-b-equal-to-c/
    - answer:

```python
# 枚举出三种情况，分别计数。当c的i位为1时，如果a，b的i位有1则加0，否则加1，如果c的i位为0时，如果a，b的i位都为1时，加2，只有一个为1则加1，否则加0。
# 第二种方法，也是枚举，但可以简化操作。
class Solution:
    def minFlips(self, a: int, b: int, c: int) -> int:
        # ans = 0
        # for i in range(32):
        #     if c & (1 << i) and (a & (1 << i) == 0 and b & (1 << i) == 0):
        #         ans += 1
        #     elif c & (1 << i) == 0 and (a & (1 << i) and b & (1 << i)):
        #         ans += 2
        #     elif c & (1 << i) == 0 and (a & (1 << i) or b & (1 << i)):
        #         ans += 1

        # return ans

        ans = 0
        for i in range(32):
            bit_a, bit_b, bit_c = (a >> i) & 1, (b >> i) & 1, (c >> i) & 1
            if bit_c == 0:
                ans += bit_a + bit_b
            else:
                ans += int(bit_a + bit_b == 0)
        return ans
```

```
// 获得二进制的最后一位let bit_a = a >> k & 1;
// 修改二进制的最后一位 a -= Math.pow(2, k);

/**
 * @param {number} a
 * @param {number} b
 * @param {number} c
 * @return {number}
 */
var minFlips = function(a, b, c) {
    let k = 0;
    let count = 0;
    while ((a | b) != c) {
        let bit_a = a >> k & 1;
        let bit_b = b >> k & 1;
        let bit_c = c >> k & 1;
        console.log("a", a, "b", b)
        if (bit_c !== (bit_a | bit_b)) {
            if (bit_c == 1) {
                count ++;
                console.log("k = ", k, "count = ", count)
                a += Math.pow(2, k);
            } else {
                if (bit_a == 1) {
                    a -= Math.pow(2, k);
                    count ++;
                    console.log("k = ", k, "count = ", count)
                }
                if (bit_b == 1) {
                    b -= Math.pow(2, k);
                    count++
                    console.log("k = ", k, "count = ", count)
                }
            }
        }
        k++;
    }
    return count;
};
```

- question: lc201 数字范围按位与 link: https://leetcode.cn/problems/bitwise-and-of-numbers-range/
    - answer:

```
# 两种方法，都基于一个共识，区间求与的结果就是两个端点的共同前缀加后续补0。前缀不同则得到0。
# 第一种方法，将左右同时右移一格并记录移动次数，当左右相等时，则将其中一个左移 移动次数。
# 第二种方法，大数不停消去最后一个1，直到小于小数，则得到求与结果。
class Solution:
    def rangeBitwiseAnd(self, left: int, right: int) -> int:
        # method 1
        # count = 0
        # while left < right:
        #     left = left >> 1
        #     right = right >> 1
        #     count += 1

        # return right << count

        # method 2
        while left < right:
            right &= right - 1

        return right
```

```
//方法1：暴力法，从left与到right
//方法二：利用n & n-1 消掉最后一个1的性质
/**
 * @param {number} left
 * @param {number} right
 * @return {number}
 */
var rangeBitwiseAnd = function(left, right) {
    let res = left;
    while (left < right) {
        left++;
        res &= left;
    }
    return res
};

var rangeBitwiseAnd = function(left, right) {
        while (left < right)
            right &= right - 1

        return right
};
```

- question: lc476 数字的补数 link: https://leetcode.cn/problems/number-complement/
    - answer:

```
# 先找到num是几位的数，然后1左移num的位数再减1得到一个与num位数相同且全1的数，与num相异或得到结果。
class Solution:
    def findComplement(self, num: int) -> int:
        k = 0
        temp = num
        while temp:
            temp = temp >> 1
            k += 1

        return num ^ ((1<<k)-1)
```

```
var findComplement = function(num) {
    let n = num;
    let k = 0;
    let res = 0;

    while (n != 0) {
        k_bit = num >> k & 1;
        n -= k_bit * Math.pow(2, k);
        res += (k_bit==0 ? Math.pow(2, k) : 0);
        console.log(k_bit, res);
        k++
    }
    return res;
};
```

- question: lc405 数字转换为十六进制数 link: https://leetcode.cn/problems/convert-a-number-to-hexadecimal/
    - answer:

```
# 每次让num与上1111，将结果转为16进制后，num右移4位直到num等于0，将结果反转则是转换结果。
class Solution:
    def toHex(self, num: int) -> str:
        if num == 0:
            return "0"

        f = lambda x: chr(x + ord('0')) if x < 10 else chr(x - 10 + ord('a'))

        result = []
        num = num & 0xFFFFFFFF

        while num > 0:
            result.append(f(num & 0XF))
            num = num >> 4

        return "".join(result[::-1])
```

```

//方法1调库，num若为负整数num + 2 ** 32来补码运算
//方法2逐4位转换
// 循环体中的代码就是解决此问题的核心思想 核心思路：

// 每次拿到num的最后4位二进制，通过num&0xf拿（也可以这样num&15）
// 根据num&0xf返回的值，从数组中找到对应元素，添加到ans中
// 然后直接无符号右移，因为2进制转16进制符号位也要参与运算（学过计组的应该很熟悉）
// 循环上面三个步骤，最后ans进行处理就能拿到最终结果
/**
 * @param {number} num
 * @return {string}
 */
// var toHex = function(num) {
//     return num >= 0 ? num.toString(16) : (num + 2 ** 32).toString(16);
// };

var toHex = function(num) {
    if(num === 0) return "0";
    let ans = '';
    const str = "0123456789abcdef";
    while(num){
        ans = str[num & 0xf] + ans;
        num >>>= 4;
    }
    return ans;
};
```

- question: lc190 颠倒二进制位 link: https://leetcode.cn/problems/reverse-bits/
    - answer:

```
# 总的有32位，反转后的数每次右移一位且或上（原数与1），原数右移一位。
class Solution:
    def reverseBits(self, n: int) -> int:
        ans = 0

        for i in range(31):
            ans |= n & 1
            ans = ans << 1
            n = n >> 1

        ans |= n & 1

        return ans
```

```
/**
 * @param {number} n - a positive integer
 * @return {number} - a positive integer
 */
var reverseBits = function(n) {
    k = 0;
    let n2 = n
    let res = 0;
    while (n!=0) {
        n_bit = n2 >> k & 1;
        res += n_bit * Math.pow(2, 31-k)
        n -= n_bit * Math.pow(2, k)
        k++
    }
    return res;
};
```

排序

阿里面试题 - 快速查找第二大数

- question: lc912 ：排序数组 link: https://leetcode.cn/problems/sort-an-array/
    - answer:

```python
# quick sort
# 快速排序的核心在于每次先得到一个pivit，比较pivit与当前值，如果当前值小于pivit，则左标记加一，与左标记交换，最终左标记加一是pivit的位置，
import random
class Solution:
    def randomPartition(self, nums, left, right):
        i = random.randint(left, right)
        nums[i], nums[right] = nums[right], nums[i]

        return self.partition(nums, left, right)

    def partition(self, nums, left, right):
        pivit = nums[right]
        i = left - 1
        for j in range(left, right):
            if nums[j] <= pivit:
                i += 1
                nums[j], nums[i] = nums[i], nums[j]

        nums[i + 1], nums[right] = nums[right], nums[i + 1]
        return i + 1

    def quickSort(self, nums, left, right):
        if left < right:
            pos = self.randomPartition(nums, left, right)
            self.quickSort(nums, left, pos - 1)
            self.quickSort(nums, pos + 1, right)

    def sortArray(self, nums: List[int]) -> List[int]:
        self.quickSort(nums, 0, len(nums)-1)
        return nums

# merge sort
# 归并排序的核心在于递归的去进行排序。最小单元则是两个长度为1的list合并为一个list，这时小的先排入list。
class Solution:
    def sortArray(self, nums: List[int]) -> List[int]:
        def mergeSort(nums, left, right):
            if left == right:
                return

            mid = left + (right - left)//2
            mergeSort(nums, left, mid)
            mergeSort(nums, mid + 1, right)

            temp = []
            i, j = left, mid + 1
            while i <= mid or right >= j:
                if i > mid or (j <= right and nums[j] < nums[i]):
                    temp.append(nums[j])
                    j += 1
                else:
                    temp.append(nums[i])
                    i += 1
            nums[left:right+1] = temp
        mergeSort(nums, 0, len(nums)-1)
        return nums

# heap sort
# 堆排序的核心在于构建堆的过程，先初始化堆，使得堆中每个元素num[i] > num[2*i + 1] 且 num[i] > num[2*i + 2]，此时，堆顶就是最大值。开始排序后，每次将堆顶交换到右边，同时左边剩下的堆重新构建一次。
class Solution:
    def sortArray(self, nums: List[int]) -> List[int]:
        def maxHeap(heap, root, length):
            p = root
            while 2*p + 1 < length:
                left = 2*p + 1
                right = 2*p + 2
                if right >= length or heap[right] < heap[left]:
                    cur = left
                else:
                    cur = right
                if heap[p] < heap[cur]:
                    heap[p], heap[cur] = heap[cur], heap[p]
                    p = cur
                else:
                    break

        def initHeap(heap):
            for i in reversed(range(len(heap))):
                maxHeap(heap, i, len(heap))

        initHeap(nums)
        for i in reversed(range(len(nums))):
            nums[0], nums[i] = nums[i], nums[0]
            maxHeap(nums, 0, i)

        return nums
```

```
/**
 * @param {number[]} nums
 * @return {number[]}
 */
//方法1 调库
// var sortArray = function(nums) {
//     nums.sort((a, b) => a-b)
//     return nums
// };

//方法2 冒泡
// var sortArray = function(nums) {
//     for (let i = 0; i < nums.length - 1; i++) {
//         for (let j = 0; j < nums.length - i - 1; j++) {
//             if (nums[j] > nums[j+1]) swap(nums, j, j+1);
//         }
//     }
//     return nums
// }
// function swap (nums, i, i2) {
//     let temp = nums[i];
//     nums[i] = nums[i2];
//     nums[i2] = temp;
// }


//方法3 快速排序
// var sortArray = function(nums) {
//     return quickSort(nums, 0, nums.length - 1)
// }

// var quickSort = function(nums, l, r) {
//     // console.log(nums, l, r)
//     if (l > r) {
//         return nums
//     }

//     let left = l;
//     let right = r;
//     let pivot = nums[left];
//     while (left < right) {
//         while (left < right && pivot <= nums[right]) {
//             right--;
//         }
//         if (pivot > nums[right]) {
//             nums[left] = nums[right];
//             left++;
//         }
//         while (left < right && pivot >= nums[left]) {
//             left++;
//         }
//         if (pivot < nums[left]) {
//             nums[right] = nums[left];
//             right--;
//         }
//         if (left >= right) {
//             nums[left] = pivot
//         }
//     }
//     quickSort(nums,l,left-1);
//     quickSort(nums,right+1, r);
//     return nums;
// }


// 方法4 merge sort
/**
 * @param {number[]} nums
 * @return {number[]}
 */
var sortArray = function(nums) {
    if(nums.length < 2) return nums;
    let midIndex = Math.floor(nums.length / 2);
    let left = nums.slice(0, midIndex);
    let right = nums.slice(midIndex);
    return merge(sortArray(left), sortArray(right));
};
const merge = (left, right) => {
    let res = [];
    let i = 0;
    let j = 0;
    let lenL = left.length;
    let lenR = right.length;
    while(i < lenL && j < lenR) {
        if(left[i] < right[j]) {
            res.push(left[i]);
            i++;
        }
        else {
            res.push(right[j]);
            j++;
        }
    }
    while(i < lenL) res.push(left[i++]);
    while(j < lenR) res.push(right[j++]);
    return res;
}
```

- question: lc628：三个数的最大乘积 link: https://leetcode.cn/problems/maximum-product-of-three-numbers/
    - answer:

```
# 排序后用最大的三个数相乘或最小的两个数乘上最大的一个数，两者的最大值就是结果，第二个是为了防止都是负数的情况。
class Solution:
    def maximumProduct(self, nums: List[int]) -> int:
        nums = sorted(nums)

        return max(nums[-1]*nums[-2]*nums[-3], nums[0]*nums[1]*nums[-1])
```

```
// 如果数组中全是非负数，则排序后最大的三个数相乘即为最大乘积；如果全是非正数，则最大的三个数相乘同样也为最大乘积。如果数组中有正数有负数，则最大乘积既可能是三个最大正数的乘积，也可能是两个最小负数（即绝对值最大）与最大正数的乘积。

//综上，我们在给数组排序后，分别求出三个最大正数的乘积，以及两个最小负数与最大正数的乘积，二者之间的最大值即为所求答案。

var maximumProduct = function(nums) {
    nums.sort((a, b) => a - b);
    const n = nums.length;
    return Math.max(nums[0] * nums[1] * nums[n - 1], nums[n - 1] * nums[n - 2] * nums[n - 3]);
};
```

- question: lc88 ：合并两个有序数组 link: https://leetcode.cn/problems/merge-sorted-array/
    - answer:

```
# 从尾部开始的双指针，分情况插值。
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        i = n + m - 1
        p1, p2 = m - 1, n - 1
        while p1 >= 0 or p2 >= 0:
            if p1 == -1:
                nums1[i] = nums2[p2]
                p2 -= 1
            elif p2 == -1:
                nums1[i] = nums1[p1]
                p1 -= 1
            elif nums1[p1] <= nums2[p2]:
                nums1[i] = nums2[p2]
                p2 -= 1
            elif nums1[p1] > nums2[p2]:
                nums1[i] = nums1[p1]
                p1 -= 1

            i -= 1
```

```
/**
 * @param {number[]} nums1
 * @param {number} m
 * @param {number[]} nums2
 * @param {number} n
 * @return {void} Do not return anything, modify nums1 in-place instead.
 */

//方法一：调库， 先merge 再sort
var merge = function(nums1, m, nums2, n) {
    nums1.splice(m, nums1.length - m, ...nums2);
    nums1.sort((a, b) => a - b);
};

//方法2：双指针
var merge = function(nums1, m, nums2, n) {
    let p1 = 0, p2 = 0;
    const sorted = new Array(m + n).fill(0);
    var cur;
    while (p1 < m || p2 < n) {
        if (p1 === m) {
            cur = nums2[p2++];
        } else if (p2 === n) {
            cur = nums1[p1++];
        } else if (nums1[p1] < nums2[p2]) {
            cur = nums1[p1++];
        } else {
            cur = nums2[p2++];
        }
        sorted[p1 + p2 - 1] = cur;
    }
    for (let i = 0; i != m + n; ++i) {
        nums1[i] = sorted[i];
    }
};
```

- question: 剑指51 ：数组中的逆序对 link: https://leetcode.cn/problems/shu-zu-zhong-de-ni-xu-dui-lcof/
    - answer:

```

```

```
var reversePairs = function(nums) {
  let res = 0
  mergeSort(0,nums.length)
  return res
  function mergeSort(l,r){
    // 区间长度小于等于2时
    if(r-l<=2){
      // 统计区间内部逆序对,同时排序
      if(r-l===2&&nums[l]>nums[l+1]){
        let temp = nums[l]
        nums[l] = nums[l+1]
        nums[l+1] = temp
        res++
      }
      return
    }
    // 区间长度大于二，可以继续分割
    let mid = Math.floor((l+r)/2)
    mergeSort(l,mid)
    mergeSort(mid,r)
    merge(l,mid,r)
  }

  function merge(l,mid,r){
    let i = 0
    let j = 0
    let k = l
    let arr1 = nums.slice(l,mid)
    let arr2 = nums.slice(mid,r)
    while(i<arr1.length&&j<arr2.length){
      // 出现逆序
      if(arr1[i]>arr2[j]){
        nums[k] = arr2[j]
        j++
        k++
        // 统计不同区间的逆序对个数，左边数组当前元素以及右边的几个元素都可以加进去
        res+=arr1.length-i
      } else{
        nums[k] = arr1[i]
        i++
        k++
      }
    }
    while(i<arr1.length){
      nums[k] = arr1[i]
      i++
      k++
    }
    while(j<arr2.length){
      nums[k] = arr2[j]
      j++
      k++
    }
  }
};

作者：11111111111-6
链接：https://leetcode.cn/problems/shu-zu-zhong-de-ni-xu-dui-lcof/solution/by-11111111111-6-tao9/
来源：力扣（LeetCode）
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

- question: lc315 ：计算右侧小于当前元素的个数 link: https://leetcode.cn/problems/count-of-smaller-numbers-after-self/
    - answer:

- question: lc327 ：区间和的个数 link: https://leetcode.cn/problems/count-of-range-sum/
    - answer:

- question: lc493 ：翻转对 link: https://leetcode.cn/problems/reverse-pairs/
    - answer:

- question: lc50剑指16 ：Pow(x, n) link: https://leetcode.cn/problems/powx-n/
    - answer:

- question: lc75 ：颜色分类 link: https://leetcode.cn/problems/sort-colors/
    - answer:

```
/**
 * @param {number[]} nums
 * @return {void} Do not return anything, modify nums in-place instead.
 */
var sortColors = function(nums) {
    quickSort(nums, 0, nums.length - 1);
};

let quickSort = (nums, left, right) => {
    if (left > right) {
        return nums;
    }
    let l = left;
    let r = right;
    let pivot = nums[l];
    while (l < r) {
        while (l < r && pivot <= nums[r]) {
            r--;
        }
        if (pivot > nums[r]) {
            nums[l] = nums[r]
            l++;
        }
        while (l < r && nums[l] <= pivot) {
            l++;
        }
        if (nums[l] > pivot ) {
            nums[r] = nums[l];
            r--;
        }
        if (l >= r) {
            nums[l] = pivot;
        }
    }
    quickSort(nums, left, l - 1);
    quickSort(nums, l + 1, right);

}
```

- question: lc179剑指45 ：最大数 link: https://leetcode.cn/problems/largest-number/
    - answer:

```
/**
 * @param {number[]} nums
 * @return {string}
 */
var largestNumber = function(nums) {
    nums.sort((a,b) => {
        var stra = b.toString() + a.toString(), strb = a.toString() + b.toString();
        if (stra > strb) {
            return 1
        } else {
            return -1
        }
    });
    if (nums[0] == 0) return '0'
    return nums.join('');
};
```

- question: lc56剑指74 ：合并区间 link: https://leetcode.cn/problems/merge-intervals/
    - answer:

```
/**
 * @param {number[][]} intervals
 * @return {number[][]}
 */
/**
 * @param {number[][]} intervals
 * @return {number[][]}
 */
var merge = function(intervals) {
    // 先进行排序 （区间第一个元素）从小到大
    const sortedIntervals = intervals.sort((a, b) => a[0] - b[0]);
    const result = [];
    // 取出第一个区间
    let current = sortedIntervals[0];
    for (let i = 1; i < sortedIntervals.length; i++) {
        // 循环比较后面的区间
        const interval = sortedIntervals[i];
        // 如果下一个区间的最小值 存在于当前比较的区间（小于当前区间的最大值） 则合并
        if (interval[0] <= current[1]) {
            // 合并取两个区间最大值的最大者
            current[1] = Math.max(current[1], interval[1]); // case 1
        } else {
            // 如果不在前一个区间内 说明当前区间与下一个区间不连续 则把当前区间添加到结果集
            result.push(current);
            // 更新比较的当前区间为下一个区间
            current = interval; // case 2
        }
    }
    // 遍历结束之后两种情况
    // case 1 最后一个区间被合并，则需要将current添加到结果集
    // case 2 最后一个区间没有被合并，也需要将current添加到结果集
    result.push(current);
    return result;
};
```

- question: lc57 ：插入区间 link: https://leetcode.cn/problems/insert-interval/

- answer:

```
/**
 * @param {number[][]} intervals
 * @param {number[]} newInterval
 * @return {number[][]}
 */
var insert = function(intervals, newInterval) {
    let len = intervals.length;
    let res = [];
    for (let i = 0; i < intervals.length; i++) {
        if (newInterval[0] < intervals[i][0]) {
            intervals.splice(i, 0, newInterval);
            break;
        }
    }
    if (intervals.length == len) {
        intervals.push(newInterval)
    }


    let cur = intervals[0];
    for (let i = 1; i < intervals.length; i++) {
        let interval = intervals[i]
        if (interval[0] <= cur[1] ) {
            cur[1] = Math.max(interval[1], cur[1]);
            console.log(cur)
        } else {
            res.push(cur);
            cur = interval
        }
    }
    res.push(cur);
    return res;


};
```

- question: lc905 ： 按奇偶排序数组 link: https://leetcode.cn/problems/sort-array-by-parity/
    - answer:

- question: lc922 ： 按奇偶排序数组 II link: https://leetcode.cn/problems/sort-array-by-parity-ii/
    - answer:

- question: lc1365 ： 有多少小于当前数字的数字 link: https://leetcode.cn/problems/how-many-numbers-are-smaller-than-the-current-number/
    - answer:

- question: lc164 ： 最大间距 link: https://leetcode.cn/problems/maximum-gap/
    - answer:

查找

- question: lc704 ： 二分查找 link: https://leetcode.cn/problems/binary-search/
    - answer:

```
# 二分查找，每次找中间值，mid = (high-low)//2 + low。更新时，如果target小于目标值，则上限等于mid - 1，如果target大于目标值，下线等于mid + 1。
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        n = len(nums)
        low, high = 0, n-1

        while low <= high:
            mid = (high-low)//2 + low

            if target == nums[mid]:
                return mid
            elif target < nums[mid]:
                high = mid - 1
            else:
                low = mid + 1

        return -1
```

- question: lc34 ： 排序数组中找元素的第一个和最后一个位置 link: https://leetcode.cn/problems/find-first-and-last-position-of-element-in-sorted-array/
    - answer:

```
# 把二分查找转换为找到小于等于target的数中最大的一个，同时找到小于等于target+1最大的一个数，两者位置之间的数就是target的空间。
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        n = len(nums)
        low, high = 0, n-1

        while low <= high:
            mid = (high-low)//2 + low

            if target > nums[mid]:
                low = mid + 1
            else:
                high = mid - 1

        return low

    def searchRange(self, nums: List[int], target: int) -> List[int]:

        start = self.search(nums, target)
        if start == len(nums) or nums[start] != target:
            return [-1, -1]
        end = self.search(nums, target + 1) - 1

        return [start, end]
```

- question: lc35 ： 搜索插入位置 link: https://leetcode.cn/problems/search-insert-position/
    - answer:

```
# 二分法查找第一个小于等于target的位置
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums)-1
        while left <= right:
            mid = left + (right - left)//2
            if nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1

        return left
```

- question: lc278 ： 第一个错误的版本 link: https://leetcode.cn/problems/first-bad-version/
    - answer:

```
# 二分查找，如果返回true，说明错误在mid左边。如果返回false，说明在mid右边。
# The isBadVersion API is already defined for you.
# def isBadVersion(version: int) -> bool:

class Solution:
    def firstBadVersion(self, n: int) -> int:
        left, right = 0, n-1
        while left <= right:
            mid = left + (right - left)//2
            if isBadVersion(mid):
                right = mid - 1
            else:
                left = mid + 1

        return left
```

- question: lc33 ： 搜索旋转排序数组 link: https://leetcode.cn/problems/search-in-rotated-sorted-array/
    - answer:

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = left + (right - left)//2

            if nums[mid] == target:
                return mid

            if nums[mid] >= nums[0]:
                if nums[mid] > target >= nums[0]:
                    right = mid - 1
                else:
                    left = mid + 1

            else:
                if nums[mid] < target <= nums[len(nums) - 1]:
                    left = mid + 1
                else:
                    right = mid - 1

        return -1
```

- question: lc153 ： 寻找旋转排序数组中的最小值 link: https://leetcode.cn/problems/find-minimum-in-rotated-sorted-array/
    - answer:

```python
class Solution:
    def findMin(self, nums: List[int]) -> int:
        if len(nums) < 2:
            return nums[0]
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = left + (right - left)//2
            if nums[mid] >= nums[0]:
                if nums[0] >= nums[len(nums) - 1]:
                    left = mid + 1
                else:
                    return nums[0]
            else:
                right = mid - 1

        return nums[left]
```

- question: lc154 ：寻找旋转排序数组中的最小值 II link: https://leetcode.cn/problems/find-minimum-in-rotated-sorted-array-ii/
    - answer:

```python
# 分三种情况讨论，第一种，当mid < high时，说明最小值肯定在mid左边，此时high更新为mid。第二种情况，当mid > high，此时最小值一定在mid右边，更新low到mid + 1，第三种情况，当mid == high时，无法确定最小值在哪，只能确定最小值在low和high之间，所以只能左移一位high，不能右移low，因为循环条件是不相等时跳出，而更新low会使得mid错过最小值。
class Solution:
    def findMin(self, nums: List[int]) -> int:
        low, high = 0, len(nums) - 1
        while low < high:
            mid = low + (high - low) // 2
            if nums[mid] < nums[high]:
                high = mid
            elif nums[mid] > nums[high]:
                low = mid + 1
            else:
                high -= 1

        return nums[low]
```

- question: lc852 ：山脉数组的峰顶索引 link: https://leetcode.cn/problems/peak-index-in-a-mountain-array/
    - answer:

```python
# 二分查找，找到mid之后比较mid和mid+1，如果mid < mid + 1，最大值在mid右边，left = mid + 1。
class Solution:
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        left, right = 0, len(arr) - 1
        while left <= right:
            mid = left + (right - left)//2
            if arr[mid] < arr[mid + 1]:
                left = mid + 1
            else:
                right = mid - 1

        return left
```

- question: lc1095 ：山脉数组中查找目标值 link: https://leetcode.cn/problems/find-in-mountain-array/
    - answer:

```python
class Solution:
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        left, right = 0, len(arr)-1
        while left <= right:
            mid = left + (right - left)//2
            if arr[mid] < arr[mid + 1]:
                left = mid + 1
            else:
                right = mid - 1

        return left
```

- question: lc162 ：寻找峰值 link: https://leetcode.cn/problems/find-peak-element/
    - answer:

```python
# 需要增加一个判断防止越界。
class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        left, right = 0, len(nums)-1
        while left <= right:
            mid = left + (right - left)//2
            f = lambda x: nums[x] if -1 < x < len(nums) else float(-inf)

            if f(mid) < f(mid + 1):
                left = mid + 1
            else:
                right = mid - 1

        return left
```

- question: lc74 ：搜索二维矩阵 link: https://leetcode.cn/problems/search-a-2d-matrix/
    - answer:

- question: lc240 ：搜索二维矩阵II link: https://leetcode.cn/problems/search-a-2d-matrix-ii/
    - answer:

- question: lc69 ：x 的平方根 link: https://leetcode.cn/problems/sqrtx/
    - answer:

- question: lc1539 ：第 k 个缺失的正整数 link: https://leetcode.cn/problems/kth-missing-positive-number/
    - answer:

- question: lc771 ：宝石与石头 link: https://leetcode.cn/problems/jewels-and-stones/
    - answer:

- question: lc888 ：公平的糖果棒交换 link: https://leetcode.cn/problems/fair-candy-swap/
    - answer:

- question: lc128 ：最长连续序列 link: https://leetcode.cn/problems/longest-consecutive-sequence/
    - answer:

- question: lc136 ：只出现一次的数字 link: https://leetcode.cn/problems/single-number/
    - answer:

- question: lc389 ：找不同 link: https://leetcode.cn/problems/find-the-difference/
    - answer:

- question: lc554 ：砖墙 link: https://leetcode.cn/problems/brick-wall/
    - answer:

- question: lc205 ：同构字符串 link: https://leetcode.cn/problems/isomorphic-strings/
    - answer:

- question: lc290 ：单词规律 link: https://leetcode.cn/problems/word-pattern/
    - answer:

- question: lc242 ：有效的字母异位词 link: https://leetcode.cn/problems/valid-anagram/
    - answer:

- question: lc49 ：字母异位词分组 link: https://leetcode.cn/problems/group-anagrams/
    - answer:

- question: lc560 ：和为K的子数组 link: https://leetcode.cn/problems/subarray-sum-equals-k/
    - answer:

- question: lc41 ：缺失的第一个正数 link: https://leetcode.cn/problems/first-missing-positive/
    - answer:

- question: lc1122 ：数组的相对排序 link: https://leetcode.cn/problems/relative-sort-array/
    - answer:

栈与队列

- question: lc20 ：有效的括号 link: https://leetcode.cn/problems/valid-parentheses/
    - answer:

```
# 构建一个stack和字典，字典存储左边对应的右边，当c in s在字典中时，压入stack，当不在字典中时，首先检查此时stack是否为空，为空说明之前没有右边进栈直接return False，如果不为空，则检查弹出的字符在字典中是否对应当前字符。循环结束后需要检查stack是否为空，如果一一匹配则为空。
class Solution:
    def isValid(self, s: str) -> bool:
        if len(s) % 2:
            return False

        dic = {'(':')', '{':'}', '[':']'}
        stack = []

        for c in s:
            if c in dic:
                stack.append(c)
            else:
                if not stack or dic[stack.pop()] != c:
                    return False

        return not stack
```

- question: lc71 剑指 017 ：简化路径 link: https://leetcode.cn/problems/simplify-path/
    - answer:

```
# 先用"/"分割path，当遇到".."时，表示返回上级路径，所以stack弹出一个列表，当遇到"."时，不用变动。当不为空时，进入stack。
class Solution:
    def simplifyPath(self, path: str) -> str:
        stand = path.split("/")
        stack = []

        for name in stand:
            if name == "..":
                if stack:
                    stack.pop()
            elif name and name != '.':
                stack.append(name)

        return "/"+"/".join(stack)
```

- question: lc394 ：字符串解码 link: https://leetcode.cn/problems/decode-string/
    - answer:

```
# 需要
class Solution:
    def decodeString(self, s: str) -> str:
        stack = []
        for c in s:
            if c != ']':
                if "0" <= c <= "9":
                    if stack:
                        c_ = stack.pop()
                        if type(c_) == int:
                            stack.append(c_ * 10 + ord(c) - ord("0"))
                        else:
                            stack.append(c_)
                            stack.append(ord(c) - ord("0"))
                    else:
                        stack.append(ord(c) - ord("0"))
                else:
                    stack.append(c)
            else:
                temp = ""
                c_ = ""
                while c_ != "[":
                    c_ = stack.pop()
                    if c_ != "[":
                        temp += c_[::-1]
                    elif c_ == "[":
                        stack.append("".join([temp[::-1] for _ in range(stack.pop())]))

        return "".join(stack)
```

- question: lc224 ：基本计算器 link: https://leetcode.cn/problems/basic-calculator/
    - answer:

```
# 本质上是将括号拆开然后直接进行计算，当前的计算符受括号前的计算符影响，如果当前为正，则当前计算符是括号前的计算符，如果当前为负，则当前计算符与括号前计算符相反。遇到数字，先得到完整数字，再乘上当前符号并让最终结果加上，扫描一遍返回结果。
class Solution:
    def calculate(self, s: str) -> int:
        ops = [1]
        sign = 1

        ret = 0
        n = len(s)
        i = 0
        while i < n:
            if s[i] == ' ':
                i += 1
            elif s[i] == '+':
                sign = ops[-1]
                i += 1
            elif s[i] == '-':
                sign = -ops[-1]
                i += 1
            elif s[i] == '(':
                ops.append(sign)
                i += 1
            elif s[i] == ')':
                ops.pop()
                i += 1
            else:
                num = 0
                while i < n and s[i].isdigit():
                    num = num * 10 + ord(s[i]) - ord('0')
                    i += 1
                ret += num * sign

        return ret
```

- question: lc227 : 基本计算器二 link: https://leetcode.cn/problems/basic-calculator-ii/
    - answer:

```
# 四种运算符，如果是加号，直接入栈，减号，取负值入栈，乘号，弹出栈顶相乘后入栈，除号，弹出栈顶相除入栈，最后把栈求和。
class Solution:
    def calculate(self, s: str) -> int:
        n = len(s)
        sign = "+"
        num = 0
        ans = []

        for i in range(n):
            if s[i] != " " and s[i].isdigit():
                num = num*10 + int(s[i])

            if i == n - 1 or s[i] in "+-*/":
                if sign == "+":
                    ans.append(num)
                elif sign == "-":
                    ans.append(-num)
                elif sign == "*":
                    ans.append(ans.pop()*num)
                elif sign == "/":
                    ans.append(int(ans.pop()/num if num != 0 else 0))

                sign = s[i]
                num = 0

        return sum(ans)
```

- question: lc946 剑指31 : 验证栈序列 link: https://leetcode.cn/problems/validate-stack-sequences/
    - answer:

```
# 模拟一个新的栈，根据pushed和popped来模拟。每次从pushed取出一个push到模拟栈中，如果此时模拟栈的栈顶与popped当前位相等，则将模拟栈pop并将popped的当前为后移一位，当pushed被遍历完后，检查模拟栈是否为空，如果为空，则可以根据pushed和popped来实现一个栈。
class Solution:
    def validateStackSequences(self, pushed: List[int], popped: List[int]) -> bool:
        if len(popped) != len(pushed):
            return False

        sim = []
        n = len(pushed)
        j = 0

        for i in range(n):
            sim.append(pushed[i])
            while sim and sim[-1] == popped[j]:
                sim.pop()
                j += 1

        return len(sim) == 0
```

- question: lc739 剑指038 : 每日温度 link: https://leetcode.cn/problems/daily-temperatures/
    - answer:

- question: lc42 : 接雨水 link: https://leetcode.cn/problems/trapping-rain-water/
    - answer:

- question: lc84 剑指039 ：柱状图中最大的矩形 link: https://leetcode.cn/problems/largest-rectangle-in-histogram/
    - answer:

- question: lc85 剑指040 ：最大矩形 link: https://leetcode.cn/problems/maximal-rectangle/
    - answer:

- question: lc321 ：拼接最大数 link: https://leetcode.cn/problems/create-maximum-number/
    - answer:

- question: lc456 : 132模式 link: https://leetcode.cn/problems/132-pattern/
    - answer:

- question: lc151 ：翻转字符串里的单词 link: https://leetcode.cn/problems/reverse-words-in-a-string/
    - answer:

- question: lc1046 : 最后一块石头的重量 link: https://leetcode.cn/problems/last-stone-weight/
    - answer:

- question: lc215 ：数组中的第 K 个最大元素 link: https://leetcode.cn/problems/kth-largest-element-in-an-array/
    - answer:

- question: lc347 ：前 K 个高频元素 link: https://leetcode.cn/problems/top-k-frequent-elements/
    - answer:

- question: lc973 : 最接近原点的 K 个点 link: https://leetcode.cn/problems/k-closest-points-to-origin/
    - answer:

- question: lc703 ：数据流中的第 K 大元素 link: https://leetcode.cn/problems/kth-largest-element-in-a-stream/
    - answer:

- question: lc295 ：数据流的中位数 link: https://leetcode.cn/problems/find-median-from-data-stream/
    - answer:

- question: lc4 ：寻找两个正序数组的中位数 link: https://leetcode.cn/problems/median-of-two-sorted-arrays/
    - answer:

滑动窗口

- question: lc239 ：滑动窗口最大值 link: https://leetcode.cn/problems/sliding-window-maximum/
    - answer:

```
# 创建堆来维护窗口最大值。python中import heap 可以使用内置堆操作。heapq.heapify(list)，将一个list转为堆，堆顶是最小值，最小堆，所以寻找最大值时，需要先将所有元素取负值。
heapq.heappush(heap, value)将value插入堆中并自动调整堆，heapq.pop(heap)弹出堆顶最小值，并自动调整堆。
# 此题中，需要存储的结构是（value, index），value用于排序生成堆，index用于判断是否在窗口内。每次插入一个值，同时判断此时堆顶是否在当前窗口内，如果不在则弹出，最后将堆顶插入返回数组中。
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        n = len(nums)
        q = [(-nums[i], i) for i in range(k)]
        heapq.heapify(q)

        ans = [-q[0][0]]

        for i in range(k, n):
            heapq.heappush(q, (-nums[i], i))
            while q[0][1] <= i - k:
                heapq.heappop(q)
            ans.append(-q[0][0])

        return ans
```

```
```

- question: lc643 ：子数组最大平均数 I link: https://leetcode.cn/problems/maximum-average-subarray-i/
    - answer:

```
# 滑动窗口来实现，每次计算窗口内之和，加上右边进位值，减去左边值，与当前最大值比较，维护最大值，返回最大值除k。
class Solution:
    def findMaxAverage(self, nums: List[int], k: int) -> float:
        maxSum = 0
        for i in range(k):
            maxSum += nums[i]

        left, right = 0, k
        temp = maxSum

        while right < len(nums):
            temp = temp - nums[left] + nums[right]
            maxSum = max(maxSum, temp)
            left += 1
            right += 1

        return maxSum / k
```

```
```

- question: lc209 剑指 008 ：长度最小的子数组 link: https://leetcode.cn/problems/minimum-size-subarray-sum/
    - answer:

```
# 两种方法，第一种，用前缀和另存一个数组，然后在遍历一次前缀和数组，每个s[i]，找到s[i+m] >= s[i] + target，此时m为最小区间。
# 第二种方法，滑动窗口，右边不停往右移动，当当前和大于target时左边左移直到刚好小于等于target。
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:
        if not nums:
            return 0

        left, right = 0, 0
        ans = len(nums) + 1
        temp = 0

        while right < len(nums):
            temp += nums[right]
            while temp >= target:
                ans = min(ans, right - left + 1)
                temp -= nums[left]
                left += 1
            right += 1

        return ans if ans != len(nums) + 1 else 0
```

```
```

- question: lc3 剑指 016 ：无重复字符的最长子串 link: https://leetcode.cn/problems/longest-substring-without-repeating-characters/
    - answer:

```
# 滑动窗口，检查是否新入的元素在窗口内，如果在窗口内，则缩小窗口直到元素不在窗口内。
# 维护left时，可以新建一个循环，也可以用一个字典存储之前的right出现位置，如果left小于它，则更新成上一个right的位置加一，并且更新right现在的位置。
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # n = len(s)
        # ans = 0
        # left, right = 0,0
        # temp = 0
        # while right < n:
        #     temp += 1
        #     while left < right and s[right] in s[left:right]:
        #         left += 1
        #         temp -= 1
        #     ans = max(ans, temp)
        #     right += 1
        # return ans

        n = len(s)
        dic = {}
        ans = 0
        left, right = 0,0
        temp = 0
        while right < n:
            temp += 1
            if s[right] in dic:
                if left < dic[s[right]] + 1:
                    left = dic[s[right]] + 1
                    temp = right - left + 1
                dic[s[right]] = right
            else:
                dic[s[right]] = right
            ans = max(ans, temp)
            right += 1
        return ans
```

```

```

- question: lc76 ：最小覆盖子串【top100】 link: https://leetcode.cn/problems/minimum-window-substring/
    - answer:

```
# 利用两个字典及滑动窗口的思想。首先写一个函数判断两个字典是否属于包含关系，dic1包含dic2表现为dic1与dic2key相同，且每个key的值都大于等于dic2。先初始化模板的字典，然后遍历s，如果s出现在模板中，则将其对应的s字典值加一。当两个字典属于包含关系时，移动左指针，使得两个字典刚好不包含。
class Solution:
    def eqDic(self, dic1, dic2):
        for i in dic1:
            if dic1.get(i, 0) > dic2.get(i, 0):
                return False
        return True

    def minWindow(self, s: str, t: str) -> str:
        dicT = {}
        dicS = {}

        for i in range(len(t)):
            dicT[t[i]] = dicT.get(t[i], 0) + 1

        left = 0
        ans = ""
        temp = float("inf")

        for right in range(len(s)):
            if s[right] in dicT:
                dicS[s[right]] = dicS.get(s[right], 0) + 1

            while self.eqDic(dicT, dicS):
                if s[left] in dicT:
                    dicS[s[left]] = dicS.get(s[left], 0) - 1
                    if dicS[s[left]] < 0:
                        del dicS[s[left]]

                if right - left + 1 < temp:
                    temp = right - left + 1
                    ans = s[left : right + 1]

                left += 1

        return ans
```

```

```

- question: lc485 ：最大连续 1 的个数 link: https://leetcode.cn/problems/max-consecutive-ones/
    - answer:

```
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        left = 0
        ans = 0
        for right in range(len(nums)):
            if nums[right] == 0:
                left = right + 1
            ans = max(right - left + 1, ans)
        return ans
```

```

```

- question: lc487 ：最大连续1的个数(vip) II link:
    - answer:

```

```

```

```

- question: lc1004 ：最大连续 1 的个数 III link: https://leetcode.cn/problems/max-consecutive-ones-iii/

- answer:

- question: lc1151 ：最少交换次数来组合所有的(vip) link:
  - answer:

- question: lc30 ：串联所有单词的子串 link: https://leetcode.cn/problems/substring-with-concatenation-of-all-words/
  - answer:

- question: lc567 & 剑指 014 ：字符串的排列 link: https://leetcode.cn/problems/permutation-in-string/
  - answer:

- question: lc763 ：划分字母区间 link: https://leetcode.cn/problems/partition-labels/
  - answer:

- question: lc845 ：数组中的最长山脉 link: https://leetcode.cn/problems/longest-mountain-in-array/
  - answer:

综合应用I

- question: lc1 ：两数之和 link: https://leetcode.cn/problems/two-sum/
  - answer:

- question: lc167 剑指 006 ：两数之和：输入有序数组 link: https://leetcode.cn/problems/two-sum-ii-input-array-is-sorted/
  - answer:

- question: lc170 ：两数之和：数据结构设计(vip) link:
  - answer:

- question: lc653 ：两数之和：输入 BST link: https://leetcode.cn/problems/two-sum-iv-input-is-a-bst/
  - answer:

- question: lc15 剑指 007 ：三数之和【top100】 link: https://leetcode.cn/problems/3sum/
  - answer:

- question: lc18 ：四数之和 link: https://leetcode.cn/problems/4sum/
  - answer:

- question: lc349 : 两个数组的交集 link: https://leetcode.cn/problems/intersection-of-two-arrays/
  - answer:

- question: lc350 ：两个数组的交集 II link: https://leetcode.cn/problems/intersection-of-two-arrays-ii/
  - answer:

- question: lc169 剑指39 ：多数元素 link: https://leetcode.cn/problems/majority-element/
  - answer:

- question: lc229：多数元素变形题 link: https://leetcode.cn/problems/majority-element-ii/
    - answer:

- question: lc844：比较含退格的字符串 link: https://leetcode.cn/problems/backspace-string-compare/
    - answer:

- question: lc318：最大单词长度乘积 link: https://leetcode.cn/problems/maximum-product-of-word-lengths/
    - answer:

链表

- question: lc203：移除链表元素 link: https://leetcode.cn/problems/remove-linked-list-elements/
    - answer:

- question: lc237 删除链表中的节点 link: https://leetcode.cn/problems/delete-node-in-a-linked-list/
    - answer:

```
# 删除某个节点，同时不给出head，只给当前要删除node的地址时，直接把当前node的val变成下一个node的val，同时node的下一个节点变成，下一个节点的下一个节点。相当于直接跳过当前要删除的node，实际上内存中并未删除。
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-place instead.
        """
        node.val = node.next.val
        node.next = node.next.next
```

- question: lc83：删除排序链表中的重复元素 link: https://leetcode.cn/problems/remove-duplicates-from-sorted-list/
    - answer:

```
# 因为已经排序，所以只需要比较当前值与下一个值，如果下一个值与当前值相同，则直接将下一个值跳过，直接指向下一个值的下一个值。
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head:
            return head

        p = head
        while p.next:
            if p.val == p.next.val:
                p.next = p.next.next
            else:
                p = p.next

        return head
```

- question: lc82：删除排序链表中的重复元素II link: https://leetcode.cn/problems/remove-duplicates-from-sorted-list-ii/
    - answer:

```python
# 与前一题的不同点在于本题要检查的值不是当前值与下一值，而是下一值与下下一值，并且需要在相同时不停向下遍历删除node。
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head:
            return head

        ans = ListNode(0, head)
        p = ans
        while p.next and p.next.next:
            if p.next.val == p.next.next.val:
                cur = p.next.val
                while p.next and p.next.val == cur:
                    p.next = p.next.next
            else:
                p = p.next

        return ans.next
```

- question: lc876：链表的中间结点 link: https://leetcode.cn/problems/middle-of-the-linked-list/
    - answer:

```python
# 利用快慢指针，快指针一次走两格，慢指针一次走一格，当快指针走完时，慢指针刚好指向中间位置。
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head:
            return head

        slow, fast = head, head

        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        return slow
```

- question: lc19：删除链表的倒数第 N 个结点 link: https://leetcode.cn/problems/remove-nth-node-from-end-of-list/
    - answer:

```python
# 速度相同的指针，但第一个指针比第二个指针的起始位置多n，那么当第一个指针走到头的时候，第二个指针就刚好差n个到头，也就是倒数第n项，但这样删除起来比较麻烦，所以可以使得第一个指针比第二个指针多n - 1，那么直接把第二个指针的下一个赋值成下下个。
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:
        if not head:
            return head

        f,temp = head, ListNode(0, head)
        s = temp
        for i in range(n):
            f = f.next

        while f:
            s = s.next
            f = f.next

        s.next = s.next.next

        return temp.next222
```

- question: lc141：环形链表 link: https://leetcode.cn/problems/linked-list-cycle/
    - answer:

```python
# 利用快慢指针，如果两个指针会相遇，则说明有环，如果两个指针不会相遇就走到头，说明没环。
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def hasCycle(self, head: Optional[ListNode]) -> bool:
        if not head:
            return False

        fast, slow = head, head
        while fast.next and fast.next.next:
            fast = fast.next.next
            slow = slow.next
            if fast == slow:
                return True

        return False
```

- question: lc142 ：环形链表 II link: https://leetcode.cn/problems/linked-list-cycle-ii/
    - answer:

```
# 快慢指针相遇时，慢指针只走了一部分的环，而快指针则走了n次环和一部分的环，说明n次环和入环前的距离加上这部分环的距离是相等的，此时再用一个慢指针从head开始出发，走到与慢指针相遇，因为速度相同，所以慢指针的新指针的路程相同，可以得出慢指针走了m次环加剩余部分的环，这部分刚好等于起点到环起点的距离。于是，当新指针与满指针相遇时，就是在环入口处。
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def detectCycle(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head:
            return head
        fast, slow = head, head
        while fast.next and fast.next.next:
            fast = fast.next.next
            slow = slow.next
            if fast == slow:
                second = head
                while second != slow:
                    second = second.next
                    slow = slow.next
                return second

        return None
```

- question: lc206 ：反转链表 link: https://leetcode.cn/problems/reverse-linked-list/
    - answer:

```
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head:
            return head

        prev = None
        cur = head
        while cur:
            curNext = cur.next
            cur.next = prev
            prev = cur
            cur = curNext

        return prev
```

- question: lc92 ：反转链表 II link: https://leetcode.cn/problems/reverse-linked-list-ii/
    - answer:

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseBetween(self, head: Optional[ListNode], left: int, right: int) -> Optional[ListNode]:
        if not head:
            return head

        tempNode = ListNode(0, head)
        p = tempNode

        for _ in range(left - 1):
            p = p.next

        # 记录需要断开点的前一位
        pre = p

        for _ in range(right-left+1):
            p = p.next

        # 记录需要终止反转的位置
        succ = p

        # leftNode记录开始反转的位置，righNode记录反转后需要接在尾部的位置
        leftNode = pre.next
        rightNode = succ.next

        # 原链表中，断开反转位置与接入位置
        pre.next = None
        succ.next = None

        # 反转链表， 反转后的尾部为leftNode，起始位置为原来的succ终止位置。
        preNode = None
        temp = leftNode
        while temp:
            curNext = temp.next
            temp.next = preNode
            preNode = temp
            temp = curNext

        # 将反转前一位接上反转的首部
        pre.next = succ
        # 将反转的最后一位接上原来断开的尾部
        leftNode.next = rightNode

        return tempNode.next
```

- question: lc61 ： 旋转链表 link: https://leetcode.cn/problems/rotate-list/
    - answer:

- question: lc328 ： 奇偶链表 link: https://leetcode.cn/problems/odd-even-linked-list/
    - answer:

- question: lc725 ： 分隔链表 link: https://leetcode.cn/problems/split-linked-list-in-parts/
    - answer:

- question: lc24 ： 两两交换链表中的节点 link: https://leetcode.cn/problems/swap-nodes-in-pairs/
    - answer:

- question: lc25 ： K 个一组翻转链表 link: https://leetcode.cn/problems/reverse-nodes-in-k-group/
    - answer:

- question: lc234 ： 回文链表 link: https://leetcode.cn/problems/palindrome-linked-list/
    - answer:

- question: lc138 ： 复制带随机指针的链表 link: https://leetcode.cn/problems/copy-list-with-random-pointer/
    - answer:

- question: lc86 ： 分隔链表 link: https://leetcode.cn/problems/partition-list/
    - answer:

- question: lc160 ：相交链表 link: https://leetcode.cn/problems/intersection-of-two-linked-lists/
  - answer:

- question: lc2 ：两数相加 link: https://leetcode.cn/problems/add-two-numbers/
  - answer:

- question: lc445 ：两数相加 II link: https://leetcode.cn/problems/add-two-numbers-ii/
  - answer:

- question: lc21 ：合并两个有序链表 link: https://leetcode.cn/problems/merge-two-sorted-lists/
  - answer:

- question: lc23 ：合并K个升序链表 link: https://leetcode.cn/problems/merge-k-sorted-lists/
  - answer:

- question: lc147 ：对链表进行插入排序 link: https://leetcode.cn/problems/insertion-sort-list/
  - answer:

- question: lc148 ：排序链表 link: https://leetcode.cn/problems/sort-list/
  - answer:

二叉树

- question: lc 144 ：二叉树的前序遍历 link: https://leetcode.cn/problems/binary-tree-preorder-traversal/
  - answer:

```
# 两种方法，第一种递归，构建一个方法，先把根节点加入答案中，再去迭代左右节点。
# 第二种方法，迭代的方法，需要利用stack，把当前节点值加入答案中后，需要将节点入栈，当前节点转换为左节点，当左节点为空时，则从stack中pop出一个点，将当前节点设置为此节点的右节点。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def preorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        # ans = []
        # def preSearch(node):
        #     if not node:
        #         return
        #     ans.append(node.val)
        #     preSearch(node.left)
        #     preSearch(node.right)
        # preSearch(root)
        # return ans

        ans = []
        node = root
        stack = []

        while node or stack:
            while node:
                ans.append(node.val)
                stack.append(node)
                node = node.left
            node = stack.pop().right

        return ans
```

- question: lc 94 ：二叉树的中序遍历 link: https://leetcode.cn/problems/binary-tree-inorder-traversal/
  - answer:

```
# 递归比较简单，只需要改变顺序就行，迭代需要注意插值的位置是跳出内层循环后插值。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        # ans = []
        # def inorder(node):
        #     if not node:
        #         return
        #     inorder(node.left)
        #     ans.append(node.val)
        #     inorder(node.right)

        # inorder(root)

        # return ans

        ans = []
        stack = []
        node = root

        while node or stack:
            while node:
                stack.append(node)
                node = node.left
            node = stack.pop()
            ans.append(node.val)
            node = node.right

        return ans
```

- question: lc 145 : 二叉树的后序遍历 link: https://leetcode.cn/problems/binary-tree-postorder-traversal/
    - answer:

```
# 递归比较简单，迭代的时候，需要用一个变量来记录此节点是否右节点已经遍历完了。当访问完一棵子树的时候，我们用prev指向该节点。这样，在回溯到父节点的时候，我们可以依据prev是指向左子节点，还是右子节点，来判断父节点的访问情况。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def postorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        # ans = []
        # def postorder(node):
        #     if not node:
        #         return
        #     postorder(node.left)
        #     postorder(node.right)
        #     ans.append(node.val)

        # postorder(root)
        # return ans
        ans = []
        stack = []

        node = root
        pre = None

        while node or stack:
            while node:
                stack.append(node)
                node = node.left
            node = stack.pop()
            if not node.right or node.right == pre:
                ans.append(node.val)
                pre = node
                node = None
            else:
                stack.append(node)
                node = node.right

        return ans
```

BFS与DFS

- question: lc 102 & 剑指 32-2 : 二叉树的层序遍历【top100】 link: https://leetcode.cn/problems/binary-tree-level-order-traversal/
    - answer:

```
# 利用队列，将不为空的左右节点同时插入队列中，每层都把que弄空一次。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
from queue import Queue
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []

        que = Queue()
        que.put(root)
        ans = []

        while que.qsize() != 0:
            level = []
            n = que.qsize()
            for _ in range(n):
                curNode = que.get()
                level.append(curNode.val)
                if curNode.left:
                    que.put(curNode.left)
                if curNode.right:
                    que.put(curNode.right)
            ans.append(level)

        return ans
```

- question: lc 107 ：二叉树的层序遍历 II link: https://leetcode.cn/problems/binary-tree-level-order-traversal-ii/
    - answer:

```
# 自上往下的层序遍历后反向切片输出。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
from queue import Queue
class Solution:
    def levelOrderBottom(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []

        que = Queue()
        que.put(root)
        ans = []

        while que.qsize() != 0:
            level = []
            n = que.qsize()
            for _ in range(n):
                curNode = que.get()
                level.append(curNode.val)
                if curNode.left:
                    que.put(curNode.left)
                if curNode.right:
                    que.put(curNode.right)
            ans.append(level)

        return ans[::-1]
```

- question: lc 104 & 剑指 55-1 ：二叉树的最大深度【top100】 link: https://leetcode.cn/problems/maximum-depth-of-binary-tree/
    - answer:

```
# 递归DFS，左右节点数最大值加一
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0

        left = self.maxDepth(root.left)
        right = self.maxDepth(root.right)

        return max(left, right) + 1
```

- question: lc 543 ：二叉树的直径【top100】 link: https://leetcode.cn/problems/diameter-of-binary-tree/
    - answer:

```
# 二叉树直径是两点之间最短距离的最大值。两点之间的最短距离可以理解为一个根节点的左右深度的最大值之和-1，-1是因为求和时会把根节点算两次，所以需要剪掉一次。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def diameterOfBinaryTree(self, root: Optional[TreeNode]) -> int:
        self.ans = 0
        def dfs(node):
            if not node:
                return 0

            left = dfs(node.left)
            right = dfs(node.right)

            self.ans = max(self.ans, left + right + 1)

            return max(left, right) + 1

        dfs(root)
        return self.ans - 1
```

- question: lc 110 & 剑指 55-2 ：平衡二叉树 link: https://leetcode.cn/problems/balanced-binary-tree/
    - answer:

```
# 自下往上遍历，如果有一个子树不是平衡的，那么整体一定不平衡。所以设置flag，当有不平衡的子树出现则flag=False，最后检查flag是否变换过。也可以不用flag，而当不平衡和字数深度为-1时返回-1.
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isBalanced(self, root: Optional[TreeNode]) -> bool:
        self.flag = True
        def dfs(node):
            if not node:
                return 0

            left = dfs(node.left)
            right = dfs(node.right)

            if abs(left - right) > 1 :
                self.flag = False

            return max(left, right) + 1

        dfs(root)

        return self.flag
```

- question: lc 111 ：二叉树的最小深度 link: https://leetcode.cn/problems/minimum-depth-of-binary-tree/
    - answer:

```
# 首先要考虑到叶子节点是左右节点都为空时才能算作叶子节点。当左右不为空时，则返回左右深度的最小值加一，当左右有空时，则返回不为空的深度，也就是左右相加再加一。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0
        left = self.minDepth(root.left)
        right = self.minDepth(root.right)

        return min(left, right) + 1 if root.left and root.right else left + right + 1
```

- question: lc 404 ：左叶子之和 link: https://leetcode.cn/problems/sum-of-left-leaves/
    - answer:

```
# 先写一个判断叶子节点的lambda，递归过程中，如果左节点是叶子节点，则直接加上，否则继续递归，如果右节点不是叶子节点则可以继续递归否则不继续递归。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def sumOfLeftLeaves(self, root: Optional[TreeNode]) -> int:
        isLeaf = lambda node: not node.left and not node.right
        def dfs(node):
            ans = 0
            if not node:
                return 0
            if node.left:
                ans += node.left.val if isLeaf(node.left) else dfs(node.left)
            if node.right and not isLeaf(node.right):
                ans += dfs(node.right)

            return ans
        return dfs(root)
```

- question: lc 103 & 剑指 32-3 ：二叉树的锯齿形层序遍历 link: https://leetcode.cn/problems/binary-tree-zigzag-level-order-traversal/
    - answer:

```
# 在层序遍历的基础上，调整每次插入的顺序以及弹出的顺序。当此时需要从左到右遍历时，按照层序遍历的方法弹出和插入，先弹出队首，插入队尾，先插入左边再插入右边。当从右往左遍历时，需要将弹出顺序及插入顺序进行相反的操作，并且此时从队尾弹出，队首插入，先插入右边再插入左边。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def zigzagLevelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []

        q = collections.deque([root])
        leftOrder = True
        ans = []

        while q:
            level = list()
            size = len(q)
            for _ in range(size):
                if leftOrder:
                    node = q.popleft()
                    if node.left:
                        q.append(node.left)
                    if node.right:
                        q.append(node.right)

                else:
                    node = q.pop()
                    if node.right:
                        q.appendleft(node.right)
                    if node.left:
                        q.appendleft(node.left)
                level.append(node.val)

            ans.append(level)
            leftOrder = not leftOrder

        return ans
```

- question: lc 515 & 剑指 044 ：在每个树行中找最大值 link: https://leetcode.cn/problems/find-largest-value-in-each-tree-row/
    - answer:

```
# 层序遍历的过程中寻找每层的最大值并插入到最终结果中。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def largestValues(self, root: Optional[TreeNode]) -> List[int]:
        if not root:
            return []

        q = collections.deque([root])
        ans = []

        while q:
            size = len(q)
            levelMax = -float("inf")
            for _ in range(size):
                node = q.popleft()
                levelMax = max(levelMax, node.val)
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)
            ans.append(levelMax)
        return ans
```

- question: lc 199 & 剑指 046 ：二叉树的右视图 link: https://leetcode.cn/problems/binary-tree-right-side-view/
    - answer:

```python
# 层序遍历时，每次只把每层最后一个加入到结果中。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def rightSideView(self, root: Optional[TreeNode]) -> List[int]:
        if not root:
            return []

        q = collections.deque([root])
        ans = []

        while q:
            size = len(q)
            node = None
            for _ in range(size):
                node = q.popleft()
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)
            ans.append(node.val)

        return ans
```

- question: lc 100 ： 相同的树 link: https://leetcode.cn/problems/same-tree/
  - answer:

```python
# 直接dfs判断值的同时递归
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:
        if not p or not q:
            return p == q
        else:
            return p.val == q.val and self.isSameTree(p.right, q.right) and self.isSameTree(p.left, q.left)
```

- question: lc 101 ： 对称二叉树 link: https://leetcode.cn/problems/symmetric-tree/
  - answer:

```python
# 类似于检查二叉树是否相等，将二叉树拆成两个二叉树进行比较，但不同点在于每次迭代时，左边的左子树与右边的右子树相比，左边的右子树与右边的右子树相比。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSymmetric(self, root: Optional[TreeNode]) -> bool:
        if not root:
            return True

        def dfs(node1, node2):
            if not node1 or not node2:
                return node1 == node2
            else:
                return node1.val == node2.val and dfs(node1.left, node2.right) and dfs(node1.right, node2.left)
        return dfs(root.left, root.right)
```

- question: lc 662 ： 二叉树最大宽度 link: https://leetcode.cn/problems/maximum-width-of-binary-tree/
  - answer:

```python
# 记录每个子节点的节点与节点位置，结果是每一行的最后一个节点的位置减掉第一个位置加一。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def widthOfBinaryTree(self, root: Optional[TreeNode]) -> int:
        q = [[root,1]]
        ans = 1
        while q:
            level = []
            for node, index in q:
                if node.left:
                    level.append([node.left, index*2])
                if node.right:
                    level.append([node.right, index*2+1])
            ans = max(ans, q[-1][1] - q[0][1] + 1)
            q = level

        return ans
```

- question: lc 222 ： 完全二叉树的节点个数 link: https://leetcode.cn/problems/count-complete-tree-nodes/
  - answer:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def countNodes(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0

        q = collections.deque([root])
        ans = 0
        while q:
            size = len(q)
            for _ in range(size):
                ans += 1
                node = q.popleft()
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)

        return ans
```

- question: lc 114 ：二叉树展开为链表【top100】 link: https://leetcode.cn/problems/flatten-binary-tree-to-linked-list/
  - answer:
- question: lc 236 & 剑指 68-2 ：二叉树的最近公共祖先【top100】 link: https://leetcode.cn/problems/lowest-common-ancestor-of-a-binary-tree/
  - answer:

回溯思想

- question: lc 112 ：路径总和 link: https://leetcode.cn/problems/path-sum/
  - answer:
- question: lc 113 & 剑指 34 ：路径总和 II link: https://leetcode.cn/problems/path-sum-ii/
  - answer:
- question: lc 257 ：二叉树的所有路径 link: https://leetcode.cn/problems/binary-tree-paths/
  - answer:
- question: lc 437 ：路径总和 III【top100】 link: https://leetcode.cn/problems/path-sum-iii/
  - answer:
- question: lc 124 ：二叉树中的最大路径和【top100】 link: https://leetcode.cn/problems/binary-tree-maximum-path-sum/
  - answer:
- question: lc 226 & 剑指 226 ：翻转二叉树【top100】 link: https://leetcode.cn/problems/invert-binary-tree/
  - answer:
- question: lc 105 & 剑指 7 ：从前序与中序遍历序列构造二叉树【top100】 link: https://leetcode.cn/problems/construct-binary-tree-from-preorder-and-inorder-traversal/
  - answer:
- question: lc 106 ：从中序与后序遍历序列构造二叉树 link: https://leetcode.cn/problems/construct-binary-tree-from-inorder-and-postorder-traversal/
  - answer:
- question: lc 116 ：填充每个节点的下一个右侧节点指针 link: https://leetcode.cn/problems/populating-next-right-pointers-in-each-node/
  - answer:

二叉搜索树

- question: lc 701 ：二叉搜索树中的插入操作 link: https://leetcode.cn/problems/insert-into-a-binary-search-tree/
  - answer:
- question: lc 108 ：将有序数组转换为二叉搜索树 link: https://leetcode.cn/problems/convert-sorted-array-to-binary-search-tree/
  - answer:
- question: lc 235 & 剑指 68-1 ：二叉搜索树的最近公共祖先 link: https://leetcode.cn/problems/lowest-common-ancestor-of-a-binary-search-tree/
  - answer:
- question: lc 98 ：验证二叉搜索树【top100】 link: https://leetcode.cn/problems/validate-binary-search-tree/
  - answer:
- question: lc 501 ：二叉搜索树中的众数 link: https://leetcode.cn/problems/find-mode-in-binary-search-tree/
  - answer:
- question: lc 99 ：恢复二叉搜索树 link: https://leetcode.cn/problems/recover-binary-search-tree/
  - answer:
- question: lc 538 & 剑指 054 ：把二叉搜索树转换为累加树【top100】 link: https://leetcode.cn/problems/convert-bst-to-greater-tree/
  - answer:
- question: lc 589 ：N 叉树的前序遍历 link: https://leetcode.cn/problems/n-ary-tree-preorder-traversal/
  - answer:
- question: lc 590 ：N 叉树的后序遍历 link: https://leetcode.cn/problems/n-ary-tree-postorder-traversal/
  - answer:
- question: lc 429 ：N 叉树的层序遍历 link: https://leetcode.cn/problems/n-ary-tree-level-order-traversal/
  - answer:
- question: lc 690 ：员工的重要性 link: https://leetcode.cn/problems/employee-importance/
  - answer:

图的 DFS 和 BFS

- question: lc 733 ：图像渲染 link: https://leetcode.cn/problems/flood-fill/
  - answer:
- question: lc 463 ：岛屿的周长 link: https://leetcode.cn/problems/island-perimeter/
  - answer:
- question: lc 200 ：岛屿数量 link: https://leetcode.cn/problems/number-of-islands/
  - answer:

```
```

```
```

- question: lc 695 ：岛屿的最大面积 link: https://leetcode.cn/problems/max-area-of-island/
  - answer:

```
```

```
```

- question: lc 130 ：被围绕的区域 link: https://leetcode.cn/problems/surrounded-regions/
  - answer:

```
```

```
```

- question: lc 1034 ：边框着色 link: https://leetcode.cn/problems/coloring-a-border/
  - answer:

```
```

```
```

- question: lc 529 ：扫雷游戏 link: https://leetcode.cn/problems/minesweeper/
  - answer:

```
```

```
```

- question: lc 994 ：腐烂的橘子 link: https://leetcode.cn/problems/rotting-oranges/
  - answer:

```
```

```
```

数据结构设计

- question: lc 155 ：最小栈 link: https://leetcode.cn/problems/min-stack/
  - answer:

```python
# 用一个辅助栈去存储每次入栈时的当前值与之前的最小值的最小值，弹出时，同时弹出主栈与辅助栈顶值。
class MinStack:

    def __init__(self):
        self.stack = []
        self.minStack = [math.inf]


    def push(self, val: int) -> None:
        self.stack.append(val)
        self.minStack.append(min(self.minStack[-1], val))


    def pop(self) -> None:
        self.stack.pop()
        self.minStack.pop()

    def top(self) -> int:
        return self.stack[-1]

    def getMin(self) -> int:
        return self.minStack[-1]

# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
# obj.push(val)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()
```

```
```

- question: lc 225 ：用队列实现栈 link: https://leetcode.cn/problems/implement-stack-using-queues/
  - answer:

```python
# 两个队列实现一个栈，每次入栈时，将入栈值插入空队中，然后将另一个队的值不停弹出且插入到空队中，直到全部插入，将两个队地址交换。
class MyStack:

    def __init__(self):
        self.q1 = collections.deque()
        self.q2 = collections.deque()

    def push(self, x: int) -> None:
        self.q2.append(x)
        while self.q1:
            self.q2.append(self.q1.popleft())
        self.q1, self.q2 = self.q2, self.q1

    def pop(self) -> int:
        return self.q1.popleft()

    def top(self) -> int:
        return self.q1[0]

    def empty(self) -> bool:
        return not self.q1

# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

- question: lc 622：设计循环队列 link: https://leetcode.cn/problems/design-circular-queue/
    - answer:

```python
# front记录队首，rear记录队尾，因为每次插值时插入rear的位置之后rear加一，所以初始化空间为k+1，且rear的位置一直为空，而队尾真正的位置是rear+1.
class MyCircularQueue:

    def __init__(self, k: int):
        self.q = [0] * (k + 1)
        self.front = 0
        self.rear = 0

    def enQueue(self, value: int) -> bool:
        if self.isFull():
            return False
        else:
            self.q[self.rear] = value
            self.rear = (self.rear + 1) % len(self.q)
            return True

    def deQueue(self) -> bool:
        if self.isEmpty():
            return False
        else:
            self.front = (self.front + 1) % len(self.q)
            return True

    def Front(self) -> int:
        return -1 if self.isEmpty() else self.q[self.front]

    def Rear(self) -> int:
        return -1 if self.isEmpty() else self.q[self.rear - 1]

    def isEmpty(self) -> bool:
        return self.rear == self.front

    def isFull(self) -> bool:
        return (self.rear + 1) % len(self.q) == self.front
```

- question: lc 380：O(1)时间插入、删除和获取随机元素 link: https://leetcode.cn/problems/insert-delete-getrandom-o1/
    - answer:

```python
# 用字典进行查询及删除，用list进行随机选择。字典里key是要存储的值，value是对应list的index。删除时，把list最后一个值与要删除的值交换并pop，且最后一个值的dic要更新。
import random
class RandomizedSet:

    def __init__(self):
        self.dic = {}
        self.lis = []


    def insert(self, val: int) -> bool:
        if val in self.dic:
            return False
        else:
            self.dic[val] = len(self.lis)
            self.lis.append(val)
            return True


    def remove(self, val: int) -> bool:
        if val in self.dic:
            self.lis[self.dic[val]] = self.lis[-1]
            self.dic[self.lis[-1]] = self.dic[val]
            del self.dic[val]
            self.lis.pop()
            return True
        else:
            return False


    def getRandom(self) -> int:
        # index = randint(0,len(self.lis)-1)
        # return self.lis[index]
        return choice(self.lis)
```

```
```

- question: lc 381 ：O(1) 时间插入，删除和获取随机元素，允许重复 link: https://leetcode.cn/problems/insert-delete-getrandom-o1-duplicates-allowed/
    - answer:

```python
class RandomizedCollection:

    def __init__(self):
        self.dic = {}
        self.lis = []

    def insert(self, val: int) -> bool:
        if val in self.dic:
            self.dic[val].append(len(self.lis))
            self.lis.append(val)
            return False
        else:
            self.dic[val] = [len(self.lis)]
            self.lis.append(val)
            return True

    def remove(self, val: int) -> bool:
        if val in self.dic:
            # 获得要删除元素的最后一个index

            index = self.dic[val][-1]
            # 在lis中，将要删除元素的值赋为lis最后一个的值
            self.lis[index] = self.lis[-1]
            # 在字典中将lis最后一个值的索引改为要删除元素在lis中的索引
            self.dic[self.lis[-1]][-1] = index
            # 插入后需要重新排序数组，否则会超出限制。
            self.dic[self.lis[-1]].sort()
            # 在字典中将要删除元素的索引弹出，在list中弹出最后一个元素
            self.dic[val].pop()
            self.lis.pop()
            # 检查此时要删除元素在字典中的list是否为空，如果为空则删除
            if not self.dic[val]:
                del self.dic[val]

            return True
        else:
            return False


    def getRandom(self) -> int:
        return choice(self.lis)
```

```
```

- question: lc 146 ：LRU 缓存机制 link: https://leetcode.cn/problems/lru-cache/
    - answer:

```
# 用双向链表存储数据，用字典存储节点信息。当查询信息和插入时，需要将节点移动到首部，当字典存满后，需要删除尾部节点。所以重点在于写一个插入首部节点的方法以及删除尾部节点的方法。要实现移动到首部，需要实现插入首部和删除节点，移动过程是先删除节点再在首部插入。插入首部是通过首部标记，将新节点插入其中。删除尾部节点是通过尾部标记得到尾部前一个节点，删除。同时如果是新加入的节点，则size++，如果是已经存在的，则只更新value。
class LinkedNode:
    def __init__(self, key = 0, value = 0):
        self.key = key
        self.value = value
        self.next = None
        self.pre = None

class LRUCache:

    def __init__(self, capacity: int):
        self.cache = dict()
        self.head = LinkedNode()
        self.tail = LinkedNode()
        self.capacity = capacity
        self.size = 0
        self.head.next = self.tail
        self.tail.pre = self.head

    def get(self, key: int) -> int:
        if key not in self.cache:
            return -1
        else:
            node = self.cache[key]
            self.moveToHead(node)
            return node.value

    def put(self, key: int, value: int) -> None:
        if key in self.cache:
            node = self.cache[key]
            node.value = value
            self.moveToHead(node)
        else:
            node = LinkedNode(key, value)
            self.cache[key] = node
            self.addToHead(node)
            self.size += 1
            if self.size > self.capacity:
                node = self.removeTail()
                self.cache.pop(node.key)
                self.size -= 1

    def addToHead(self, node):
        node.pre = self.head
        node.next = self.head.next
        self.head.next.pre = node
        self.head.next = node

    def moveToHead(self, node):
        self.removeNode(node)
        self.addToHead(node)

    def removeNode(self, node):
        node.pre.next = node.next
        node.next.pre = node.pre

    def removeTail(self):
        node = self.tail.pre
        self.removeNode(node)
        return node
```

- question: lc 460 ：LFU 缓存 link: https://leetcode.cn/problems/lfu-cache/
  - answer:

- question: lc 547 ：省份数量 link: https://leetcode.cn/problems/number-of-provinces/
  - answer:

```
# 利用dfs的方式去遍历所有省份。当前城市的连接城市继续寻找下一个连接城市，直到没有连接的城市，省份数加一，继续找一个没有访问过的城市作为起点，往下遍历。
class Solution:
    def findCircleNum(self, isConnected: List[List[int]]) -> int:
        n = len(isConnected)
        visited = set()
        ans = 0

        def dfs(start):
            for end in range(n):
                if isConnected[start][end] and end not in visited:
                    visited.add(end)
                    dfs(end)

        for start in range(n):
            if start not in visited:
                dfs(start)
                ans += 1

        return ans
```

- question: lc 200 ：岛屿数量 link: https://leetcode.cn/problems/number-of-islands/
  - answer:

```
# 用dfs的方式遍历图，因为岛屿只和上下左右有关。dfs时每次遍历点要进行标记，然后注意停止条件。
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        def dfs(grid, row, col):
            if row < 0 or row >= nr or col < 0 or col >= nc or grid[row][col] != "1":
                return
            else:
                grid[row][col] = "2"

            dfs(grid, row, col+1)
            dfs(grid, row, col-1)
            dfs(grid, row+1, col)
            dfs(grid, row-1, col)

        nr = len(grid)
        nc = len(grid[0])
        ans = 0

        for i in range(nr):
            for j in range(nc):
                if grid[i][j] == "1":
                    ans += 1
                    dfs(grid, i, j)

        return ans
```

- question: lc 463 ：岛屿的周长 link: https://leetcode.cn/problems/island-perimeter/
    - answer:

- question: lc 695 ：岛屿的最大面积 link: https://leetcode.cn/problems/max-area-of-island/
    - answer:

```
# dfs遍历图,每次加一,遇到海和不符合要求的停止并返回0.最终放回最大面积.
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        def dfs(grid, row, col):
            if row < 0 or row >= nr or col < 0 or col >= nc or grid[row][col] != 1:
                return 0
            else:
                grid[row][col] = 2
                return 1 + dfs(grid, row, col+1) + dfs(grid, row, col-1) + dfs(grid, row+1, col) + dfs(grid, row-1, col)

        nr = len(grid)
        nc = len(grid[0])
        ans = 0

        for i in range(nr):
            for j in range(nc):
                if grid[i][j] == 1:
                    ans = max(ans, dfs(grid, i, j))

        return ans
```

- question: lc 827 ：最大人工岛link:
    - answer:

- question: lc 721 ：账户合并 link: https://leetcode.cn/problems/accounts-merge/
    - answer:

```python
# 本题核心在于并查集的创建过程，首先是初始化过程，初始化时，每个邮件视为单独的一个集合，所以初始化长度则为邮件数量。之后是合并操作。每批邮件以同用户名下第一个邮件的index为目标进行合并，同时合并到同一个index下。之后遇到在此名字下的邮件但unionList[index] != index说明此email在之前的名字下就出现过，于是要将之前的与当前的合并为同一个index下。
class UnionFind:
    def __init__(self, n):
        self.unionList = list(range(n))

    def find(self, index):
        if self.unionList[index] != index:
            self.unionList[index] = self.find(self.unionList[index])
        return self.unionList[index]

    def merge(self, index1, index2):
        self.unionList[self.find(index2)] = self.find(index1)


class Solution:
    def accountsMerge(self, accounts: List[List[str]]) -> List[List[str]]:
        emailName = {}
        emailIndex = {}

        for acc in accounts:
            name = acc[0]
            for email in acc[1:]:
                if email not in emailIndex:
                    emailIndex[email] = len(emailIndex)
                    emailName[email] = name

        union = UnionFind(len(emailIndex))

        for acc in accounts:
            first = emailIndex[acc[1]]
            for email in acc[2:]:
                union.merge(first, emailIndex[email])

        ans = {}

        for email, index in emailIndex.items():
            index = union.find(index)
            if index in ans:
                ans[index].append(email)
            else:
                ans[index] = [email]

        return [[emailName[emails[0]]] + sorted(emails) for emails in ans.values()]
```

```
```

综合应用II

lc 217 ： 存在重复元素
lc 219 ： 存在重复元素 II
lc 220 & 剑指 057 ： 存在重复元素 III
lc 258 ： 各位相加
lc 202 ： 快乐数
lc 263 ： 丑数

字典树 - 前缀树 - Tire

lc 208 & 剑指 062 ： 实现 Trie (前缀树)【top100】
lc 642 ： 搜索自动补全系统
lc 421 & 剑指 067 ： 数组中两个数的最大异或值
lc 440 ： 字典序的第K小数字

- question: link:
    - answer:

```
```

```
```

回溯算法

- question: lc 112 ： 路径总和 link: https://leetcode.cn/problems/path-sum/submissions/389620646/
    - answer:

```python
# 用bfs去遍历二叉树的同时，用两个队列分别存储节点和当前路径长度。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def hasPathSum(self, root: Optional[TreeNode], targetSum: int) -> bool:
        if not root:
            return False

        qNode = collections.deque([root])
        qVal = collections.deque([root.val])

        while qNode:
            node = qNode.popleft()
            temp = qVal.popleft()
            if not node.right and not node.left:
                if temp == targetSum:
                    return True

            if node.left:
                qNode.append(node.left)
                qVal.append(temp + node.left.val)

            if node.right:
                qNode.append(node.right)
                qVal.append(temp + node.right.val)

        return False
```

- question: lc 113 ：路径总和 link: https://leetcode.cn/problems/path-sum-ii/
    - answer:

```python
# 类似上一题的方法，用bfs去遍历的同时存下路径，符合要求则加入放回结果中。
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def pathSum(self, root: Optional[TreeNode], targetSum: int) -> List[List[int]]:
        if not root:
            return []

        qNode = collections.deque([root])
        qRout = collections.deque()
        qRout.append([root.val])

        ans = []

        while qNode:
            node = qNode.popleft()
            rout = qRout.popleft()

            if not node.right and not node.left:
                if sum(rout) == targetSum:
                    ans.append(rout)

            if node.left:
                qNode.append(node.left)
                qRout.append(rout + [node.left.val])

            if node.right:
                qNode.append(node.right)
                qRout.append(rout + [node.right.val])

        return ans
```

- question: lc 46 全排列 link: https://leetcode.cn/problems/permutations/
    - answer:

```
# python itertools 提供permutations和combinations两个方法可以直接做排列组合。
# 用回溯的思想做全排列。
# from itertools import permutations
# class Solution:
#     def permute(self, nums: List[int]) -> List[List[int]]:
#         return list(permutations(nums))

class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        if not nums:
            return []

        ans = []
        n = len(nums)

        def process(start):
            if start == n:
                # print("1",nums)
                # print("2",nums[:])
                # 如果append(nums)是将nums地址传入，后面的任何操作都会导致之前已经append的nums改变。
                ans.append(nums[:])
                # print("3",ans)

            for i in range(start, n):
                nums[i], nums[start] = nums[start], nums[i]
                process(start + 1)
                nums[i], nums[start] = nums[start], nums[i]

        process(0)

        return ans
```

- question: lc 47 全排列 link:
    - answer:

```
# from itertools import permutations
# class Solution:
#     def permuteUnique(self, nums: List[int]) -> List[List[int]]:
#         return list(set(permutations(nums, len(nums))))

class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        if not nums:
            return []

        ans = []
        n = len(nums)

        def process(start):
            if start == n:
                if nums[:] not in ans:
                    ans.append(nums[:])

            for i in range(start, n):
                nums[i], nums[start] = nums[start], nums[i]
                process(start + 1)
                nums[i], nums[start] = nums[start], nums[i]

        process(0)

        return ans
```

- question: lc 77 ：组合 link: https://leetcode.cn/problems/combinations/description/
    - answer:

```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        ans = []

        def process(start, temp):
            if len(temp) == k:
                ans.append(temp[:])
                return
            if (n+1-start + len(temp)) < k:
                return

            for i in range(start, n + 1):
                # 增减枝
                temp.append(i)
                process(i+1, temp)
                temp.pop()

        process(1, [])

        return ans
```

- question: lc 39 ：组合总和 link: https://leetcode.cn/problems/combination-sum/description/
    - answer:

```python
# 当前点可选择点是当前位置到结尾的位置的所有值，递归结束后需要剪枝pop
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:

        def process(temp, target, start):
            if target == 0:
                ans.append(temp[:])
                return
            if target < 0:
                return
            if target > 0:
                for i in range(start, len(candidates)):
                    temp.append(candidates[i])
                    process(temp, target - candidates[i], i)
                    temp.pop()

        ans = []
        process([], target, 0)

        return ans
```

- question: lc 40 ：组合总和 link: https://leetcode.cn/problems/combination-sum-ii/
    - answer:

```python
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:

        def process(temp, target, start):
            if target == 0:
                ans.append(temp[:])
                return
            if target < 0:
                return

            for i in range(start, n):
                if candidates[i] > target:
                    return
                if i > start and candidates[i - 1] == candidates[i]:
                    continue

                temp.append(candidates[i])
                process(temp, target - candidates[i], i + 1)
                temp.pop()

        ans = []
        n = len(candidates)
        candidates.sort()
        process([], target, 0)

        return ans
```

- question: lc 78 ：子集 link: https://leetcode.cn/problems/subsets/description/
    - answer:

```python
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:

        def process(temp, start):
            ans.append(temp[:])
            for i in range(start, n):
                temp.append(nums[i])
                process(temp, i + 1)
                temp.pop()

        ans = []
        n = len(nums)
        process([], 0)

        return ans
```

- question: lc 90 ：子集 link: https://leetcode.cn/problems/subsets-ii/
    - answer:

```python
# 先用排序和筛选可以符合约束条件
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        def process(temp, start):
            if temp not in ans:
                ans.append(temp[:])
            for i in range(start, n):
                temp.append(nums[i])
                process(temp, i + 1)
                temp.pop()
        nums.sort()
        ans = []
        n = len(nums)
        process([], 0)

        return ans
```

- question: lc 17 ：电话号码的字母组合 link: https://leetcode.cn/problems/letter-combinations-of-a-phone-number/
    - answer:

```python
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:
            return []

        dic = {"2":"abc",
        "3":"def",
        "4":"ghi",
        "5":"jkl",
        "6":"mno",
        "7":"pqrs",
        "8":"tuv",
        "9":"wxyz"}

        k = len(digits)
        ans = []

        def process(i, temp):
            if len(temp) == k:
                ans.append("".join(temp))
            else:
                for c in dic[digits[i]]:
                    temp.append(c)
                    process(i + 1, temp)
                    temp.pop()

        process(0, [])

        return ans
```

- question: lc 93 ： 复原 IP 地址 link: https://leetcode.cn/problems/restore-ip-addresses/
    - answer:

- question: lc 22 ： 括号生成 link: https://leetcode.cn/problems/generate-parentheses/
    - answer:

- question: lc 51 ： N 皇后（经典）link: https://leetcode.cn/problems/n-queens/
    - answer:

- question: lc 37 ： 数独问题 link: https://leetcode.cn/problems/sudoku-solver/
    - answer:

- question: lc 401 ： 二进制手表 link: https://leetcode.cn/problems/binary-watch/
    - answer:

- question: lc 131 ： 分割回文串 link: https://leetcode.cn/problems/palindrome-partitioning/
    - answer:

- question: lc 842 ： 将数组拆分成斐波那契序列 link: https://leetcode.cn/problems/split-array-into-fibonacci-sequence/
    - answer:

- question: lc 79 ： 单词搜索 link: https://leetcode.cn/problems/word-search/
    - answer:

- question: lc 679 ： 24 点游戏 link: https://leetcode.cn/problems/24-game/
    - answer:

贪心算法

lc 455 ： 分发饼干 - 贪心思想
lc 322 & 剑指 103 ： 零钱兑换 - 贪心 + 回溯【top100】
贪心算法特点总结 25-4
lc 45 ： 跳跃游戏 II
lc 55 ： 跳跃游戏【top100】

lc 1578 ：避免重复字母的最小删除成本
lc 402 ：移掉K位数字
lc 409 ：最长回文串
lc 680 & 剑指 019 ：验证回文字符串 Ⅱ
lc 316 ：去除重复字母
lc 1047 ：删除字符串中的所有相邻重复项
lc 1209 ：删除字符串中的所有相邻重复项 II
lc 976 ：三角形的最大周长
lc 674 ：最长连续递增序列
lc 738 ：单调递增的数字
lc 134 ： 加油站
lc 767 ：重构字符串
lc 621 ： 任务调度器【top100】
lc 670 ：最大交换
lc 861 ：翻转矩阵后的得分
lc 1029 ：两地调度
lc 330 ：按要求补齐数组

- question: link:
    - answer:

- question: link:
    - answer:

- question: link:
    - answer:

- question: link:
    - answer:

动态规划

lc 509 & 剑指 10-1 ：斐波那契数列问题 - 动态规划入门
lc 322 & 剑指 103 ：零钱兑换
动态规划总结 27-4
lc 64 & 剑指 099 ：最小路径和【top100】
lc 53 & 剑指 42 ：最大子数组之和【top100】
lc 53 & 剑指 42 ：最大子数组之和【top100】
lc 647、5、131 & 剑指 086 、020 ：回文子串【top100】
lc 516 ：最长回文子序列
lc 300 ：最长上升子序列【top100】
lc 1143 & 剑指 095 ：最长公共子序列
lc 72 ：编辑距离【top100】
lc 44 ：通配符匹配
lc 486 ：预测赢家
lc 70 & 剑指 10 - 2： 爬楼梯【top100】
lc 746 & 剑指 088 ：使用最小花费爬楼梯
lc 198 & 剑指 089 ：打家劫舍【top100】
lc 213 & 剑指 090 ：打家劫舍 II
lc 337 ：打家劫舍 III【top100】

背包问题/完全背包问题

lc 322 & 剑指 103 ：零钱兑换
lc 518 ：零钱兑换 II
lc 377 & 剑指 104 ：组合总和 Ⅳ
lc 494 & 剑指 102 ：目标和【top100】
lc 416 & 剑指 101 ：分割等和子集【top100】
lc 279 ：完全平方数【top100】
lc 474 ：一和零
lc 139 ：单词拆分【top100】
lc 62 & 剑指 098 ：不同路径【top100】
lc 63 ：不同路径 II
lc 120 & 剑指 100 ：三角形最小路径和
lc 97 & 剑指 096 ：交错字符串
lc 221 ： 最大正方形【top100】

系列算法题：买卖股票的最佳时机

lc 121 & 剑指 63 ：买卖股票的最佳时机【top100】
lc 122 ：买卖股票的最佳时机 II
lc 123 ：买卖股票的最佳时机 III
lc 188 ：买卖股票的最佳时机 IV
lc 309 ：最佳买卖股票时机含冷冻期【top100】
lc 714 ：买卖股票的最佳时机含手续费
lc 139. 单词拆分【top100】"
lc 140. 单词拆分 II
lc 91. 解码方法
lc 32. 最长有效括号【top100】
lc 10 & 剑指 19. 正则表达式匹配【top100】
lc 718. 最长重复子数组
lc 354. 俄罗斯套娃信封问题
lc 152. 乘积最大子数组【top100】
lc 376. 摆动序列

- question: link:
    - answer:

- question: link:
    - answer: