第一周：

1，15，20，27，46，55，66，136，171，198

第二周：

2，11，26，47，48，53，73，137，344，345

第三周：

77，100，101，125，209，242，260，283，349，350

第四周：

19，21，24，61，83，92，147，203，206，237

第五周：

18，25，143，148，202，205，217，219，290，451

第六周：

3、75、76、80、88、167、215、496、503、739

第一周：

1.

给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那两个整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。

示例:

给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9

所以返回 [0, 1]

代码：

class Solution {

    public int[] twoSum(int[] nums, int target) {

        Map<Integer, Integer> targetMap = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {

            if (targetMap.containsKey(target - nums[i])) {

                return new int[] {targetMap.get(target- nums[i]), i};

            }else {

                targetMap.put(nums[i], i);

            }

        }

        return null;

    }

}

1. 三数之和

给定一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？找出所有满足条件且不重复的三元组。

注意：答案中不可以包含重复的三元组。

例如, 给定数组 nums = [-1, 0, 1, 2, -1, -4]，

满足要求的三元组集合为：

[

[-1, 0, 1],

[-1, -1, 2]

]

代码：

class Solution {

    public List<List<Integer>> threeSum(int[] nums) {

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

        Arrays.sort(nums);

        if (nums == null || nums.length < 2 || nums[0] > 0 || nums[nums.length - 1] < 0)

            return newList;

        for (int i = 0; i < nums.length - 2; i++) {

            if (nums[i] > 0)

                break;

            int first = i + 1;

            int last = nums.length - 1;

            while (first < last) {

                if (nums[i] + nums[first] + nums[last] > 0) {

                    last--;

                    continue;

                }

                if (nums[i] + nums[first] + nums[last] < 0) {

                    first++;

                    continue;

                }

                if (nums[i] + nums[first] + nums[last] == 0) {

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

                    list.add(nums[i]);

                    list.add(nums[first]);

                    list.add(nums[last]);

                    newList.add(list);

                    while (nums[first] == nums[++first]) {

                        if (first == last)

                            break;

                    };

                    last--;

                }

            }

            while (nums[i] == nums[++i]) {

                if (i == nums.length - 2)

                    break;

            };

            i = i - 1;

        }

        return newList;

    }

}

1. 有效的括号

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。

左括号必须以正确的顺序闭合。

注意空字符串可被认为是有效字符串。

示例 1:

输入: "()"

输出: true

示例 2:

输入: "()[]{}"

输出: true

示例 3:

输入: "(]"

输出: false

示例 4:

输入: "([)]"

输出: false

示例 5:

输入: "{[]}"

输出: true

代码：

class Solution {

    public boolean isValid(String s) {

        Stack<Character> leftStack = new Stack<>();

        for (char c : s.toCharArray()) {

            if ( c == '(' || c == '{' || c == '[') {

                leftStack.push(c);

            } else {

                if (leftStack.isEmpty()) {

                    return false;

                }

                char ch = leftStack.pop();

                boolean b1 = ch == '(' && c != ')';

                boolean b2 = ch == '[' && c != ']';

                boolean b3 = ch == '{' && c != '}';

                if (b1 || b2 || b3)

                    return false;

            }

        }

        return leftStack.isEmpty();

    }

}

1. 移除元素

给定一个数组 nums 和一个值 val，你需要原地移除所有数值等于 val 的元素，返回移除后数组的新长度。

不要使用额外的数组空间，你必须在原地修改输入数组并在使用 O(1) 额外空间的条件下完成。

元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。

示例 1:

给定 nums = [3,2,2,3], val = 3,

函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。

你不需要考虑数组中超出新长度后面的元素。

示例 2:

给定 nums = [0,1,2,2,3,0,4,2], val = 2,

函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。

注意这五个元素可为任意顺序。

代码：

class Solution {

    public int removeElement(int[] nums, int val) {

        int count = nums.length;

        int flag = 0;

        for (int n : nums) {

            if (n != val) {

                nums[flag++] = n;

                if (flag == count)

                    return count;

                // flag++;

            }else {

                count--;

            }

        }

        return count;

    }

}

1. 全排列

给定一个没有重复数字的序列，返回其所有可能的全排列。

示例:

输入: [1,2,3]

输出:

[

[1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1]

]

代码：

class Solution {

    public List<List<Integer>> permute(int[] nums) {

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

        permute(0, nums, newList);

        return newList;

    }

    public void permute(int k, int[] nums, List<List<Integer>>newList) {

        if (k == nums.length - 1) {

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

            for (int i = 0; i < nums.length; i++)

                list.add(nums[i]);

            newList.add(list);

            return;

        }

        for (int i = k; i < nums.length; i++) {

            swap(nums, i, k);

            permute(k + 1, nums, newList);

            swap(nums, k, i);

            }

    }

    public void swap(int[] nums, int i, int j) {

        int temp = nums[i];

        nums[i] = nums[j];

        nums[j] = temp;

}

}

1. 跳跃游戏

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

示例 1:

输入: [2,3,1,1,4]

输出: true

解释: 我们可以先跳 1 步，从位置 0 到达 位置 1, 然后再从位置 1 跳 3 步到达最后一个位置。

示例 2:

输入: [3,2,1,0,4]

输出: false

解释: 无论怎样，你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0 ， 所以你永远不可能到达最后一个位置。

**代码：**

66.加一

给定一个由整数组成的非空数组所表示的非负整数，在该数的基础上加一。

最高位数字存放在数组的首位， 数组中每个元素只存储单个数字。

你可以假设除了整数 0 之外，这个整数不会以零开头。

示例 1:

输入: [1,2,3]

输出: [1,2,4]

解释: 输入数组表示数字 123。

示例 2:

输入: [4,3,2,1]

输出: [4,3,2,2]

解释: 输入数组表示数字 4321。

代码：

class Solution {

    public int[] plusOne(int[] digits) {

        if (digits == null)

            return null;

        if (digits.length == 0)

            return new int[]{1};

        int binary = digits.length - 1;

        if (digits[binary] < 9) {

            digits[binary] += 1;

            return digits;

        }else {

            while (binary > 0) {

                digits[binary--] = 0;

                if (digits[binary] < 9) {

                    digits[binary] += 1;

                    return digits;

                }else {

                    digits[binary] = 0;

                }

            }

            if (binary == 0) {

                int[] newArr = new int[digits.length + 1];

                newArr[0] = 1;

                return newArr;

            }

        }

        return digits;

    }

}

1. 只出现一次的数字

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

示例 1:

输入: [2,2,1]

输出: 1

示例 2:

输入: [4,1,2,1,2]

输出: 4

代码：

class Solution {

    public int singleNumber(int[] nums) {

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

        for (int n : nums) {

            if (list.contains(n)) {

                list.remove(n);

            }else {

                list.add(n);

            }

        }

        return list.get(0);

    }

}

1. Excel表格

给定一个Excel表格中的列名称，返回其相应的列序号。

例如，

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

...

示例 1:

输入: "A"

输出: 1

示例 2:

输入: "AB"

输出: 28

示例 3:

输入: "ZY"

输出: 701

代码：

class Solution {

    public int titleToNumber(String s) {

        char[] ch = s.toCharArray();

        int total = 0;

        for(int i = ch.length - 1; i >= 0; i--) {

            int base = 1;

            for(int j = 1; j < ch.length - i; j++) {

                base \*= 26;

            }

            total += (ch[i] - 'A' + 1) \* base;

        }

        return total;

    }

}

1. 打家劫舍

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组，计算你在不触动警报装置的情况下，能够偷窃到的最高金额。

示例 1:

输入: [1,2,3,1]

输出: 4

解释: 偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3)。

  偷窃到的最高金额 = 1 + 3 = 4 。

示例 2:

输入: [2,7,9,3,1]

输出: 12

解释: 偷窃 1 号房屋 (金额 = 2), 偷窃 3 号房屋 (金额 = 9)，接着偷窃 5 号房屋 (金额 = 1)。

  偷窃到的最高金额 = 2 + 9 + 1 = 12 。

代码：

class Solution {

    public int rob(int[] nums) {

        int len = nums.length;

        if(len == 0)

            return 0;

        int[] dp = new int[len + 1];

        dp[1] = nums[0];

        for(int i = 1; i < len; i++) {

            dp[i + 1] = Math.max(dp[i - 1] + nums[i], dp[i]);

        }

        return dp[len];

    }

}