# leetcode每日一题

[leetcode每日一题 1](#_Toc20033)

[回文排列 1](#_Toc23336)

[搜索插入位置 3](#_Toc20823)

[删除素组元素 4](#_Toc27662)

[按奇偶排序数组 6](#_Toc26705)

[两数之和 10](#_Toc453)

[动态规划 上楼梯 12](#_Toc28768)

[动态规划 不同路径 13](#_Toc666)

# 回文排列



import java.util.ArrayList;

class Solution {

    public boolean canPermutePalindrome(String s) {

        int length = s.length();

        ArrayList<Character> charContainer = new ArrayList<>();

        if (length % 2 == 0) {

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

                int temp = 1;

                if(charContainer.contains(s.charAt(i))==true)

                    continue;

                charContainer.add(s.charAt(i));

                for (int j = 0; j < length; j++) {

                    if (s.charAt(i) == s.charAt(j) && j != i)

                        temp++;

                }

                if (temp%2 != 0 || temp < 2)

                    return false;

            }

        } else {

            int odd\_num = 0;

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

                int temp = 1;

                if(charContainer.contains(s.charAt(i))==true)

                    continue;

                charContainer.add(s.charAt(i));

                for (int j = 0; j < length; j++) {

                    if (s.charAt(i) == s.charAt(j) && j != i)

                        temp++;

                }

                if(odd\_num > 1)

                    return false;

                if (temp < 2){

                    odd\_num++;

                }

                if(temp>2 && temp%2 != 0)

                    odd\_num++;

                if(temp == length)

                    return true;

            }

        }

        return true;

    }

}

# 搜索插入位置



class Solution {

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

        int[] position = new int[nums.length];

        int count = 0;

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

            if(target == nums[i]){

                return i;

            }

            if(nums[i] > target){

                count++;

                position[i] = i;

            }

        }

        if(count == 0)

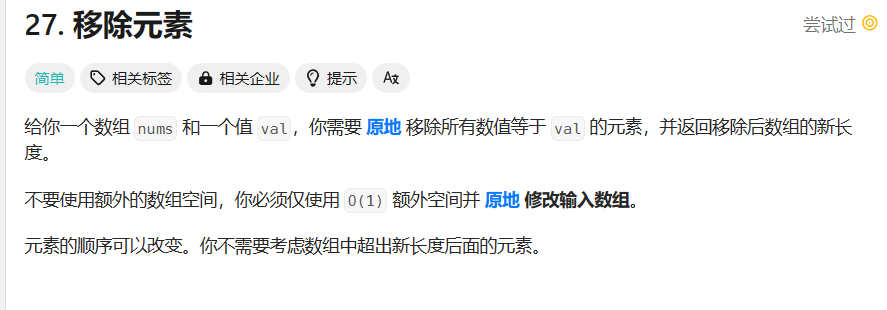
            return nums.length;

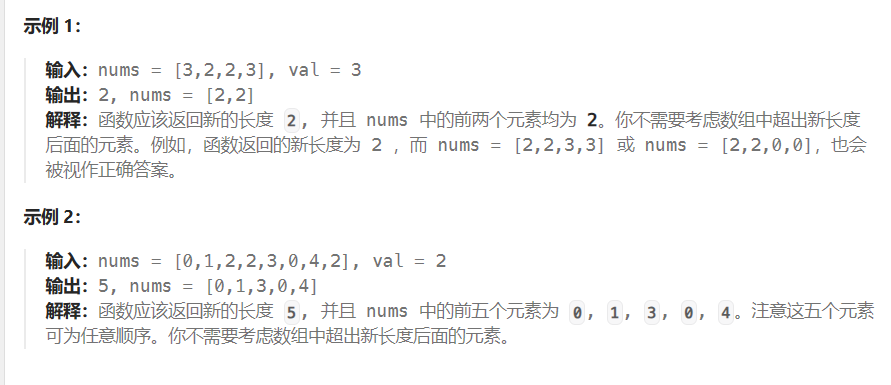
        return position[nums.length-count];

    }

}

# 删除素组元素





我的代码：

class Solution {

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

        int count = 0;

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

            if(nums[i] == val || nums[i] == -1){

                count++;

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

                    if(nums[j] != val && nums[j] != -1){

                        nums[i] = nums[j];

                        nums[j] = val;

                        break;

                    }

                }

            }

        }

        return nums.length - count;

    }

}

bing的代码：

class Solution {

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

// 判断数组是否为空

if (nums == null || nums.length == 0) {

return 0;

}

// 定义快慢指针

int fast = 0, slow = 0;

// 遍历数组

while (fast < nums.length) {

// 如果快指针指向的元素不等于val

if (nums[fast] != val) {

// 把它赋值给慢指针指向的元素

nums[slow] = nums[fast];

// 慢指针向前移动一步

slow++;

}

// 快指针向前移动一步

fast++;

}

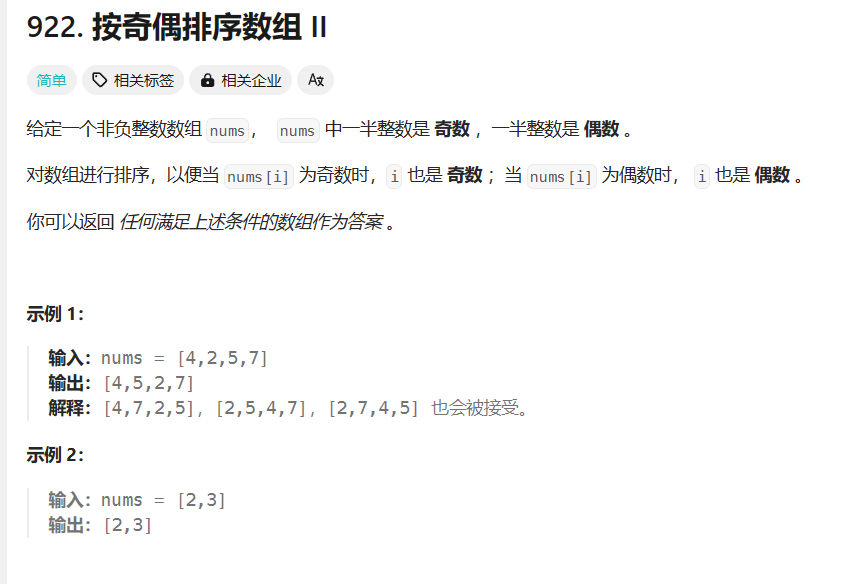
// 返回慢指针的位置，即新数组的长度

return slow;

}

}

# 按奇偶排序数组



1. 使用额外的数组空间

class Solution {

    public int[] sortArrayByParityII(int[] nums) {

        int len = nums.length;

        int odd = 1, even = 0;

        int[] newNums = new int[len];

        for(int i = 0; i < len; i++){

            if(nums[i]%2 == 0){

                newNums[even] = nums[i];

                even += 2;

            }

            if(nums[i]%2 != 0){

                newNums[odd] = nums[i];

                odd += 2;

            }

        }

        return newNums;

    }

}



1. 不使用额外的数组空间

class Solution {

    public int[] sortArrayByParityII(int[] nums) {

        int len = nums.length;

        // int odd = 1, even = 0;

        for(int i = 0; i < len; i++){

            // 下标为奇数但是值为偶数

            if(i%2 != 0 && nums[i]%2 == 0){

                for(int j = i +1; j < len; j += 2){

                    if(nums[j]%2 != 0 && j%2 == 0){

                        int temp;

                        temp = nums[i];

                        nums[i] = nums[j];

                        nums[j] = temp;

                        break;

                    }

                }

            }

            // 下标为偶数但是值为奇数

            if(i%2 == 0 && nums[i]%2 != 0){

                for(int j = i + 1; j < len; j += 2){

                    if(nums[j]%2 == 0 && j%2 != 0){

                        int temp;

                        temp = nums[i];

                        nums[i] = nums[j];

                        nums[j] = temp;

                        break;

                    }

                }

            }

        }

        return nums;

    }

}



1. 不使用额外的数组空间改进算法

class Solution {

    public int[] sortArrayByParityII(int[] nums) {

        int len = nums.length;

        int odd = 1, even = 0;

        while(odd < len && even < len){

            // 均不在位

            if(nums[odd]%2 == 0 && nums[even]%2 != 0){

                int temp;

                temp = nums[odd];

                nums[odd] = nums[even];

                nums[even] = temp;

            }else if(nums[odd]%2 != 0 && nums[even]%2 != 0){ // 偶数不在位

                while(nums[odd]%2 != 0){

                    odd += 2;

                }

                int temp;

                temp = nums[even];

                nums[even] = nums[odd];

                nums[odd] = temp;

            }else if(nums[odd]%2 == 0 && nums[even]%2 == 0){ // 奇数不在位

                while(nums[even]%2 == 0){

                    even += 2;

                }

                int temp;

                temp = nums[odd];

                nums[odd] = nums[even];

                nums[even] = temp;

            }

            else{

                odd += 2;

                even += 2;

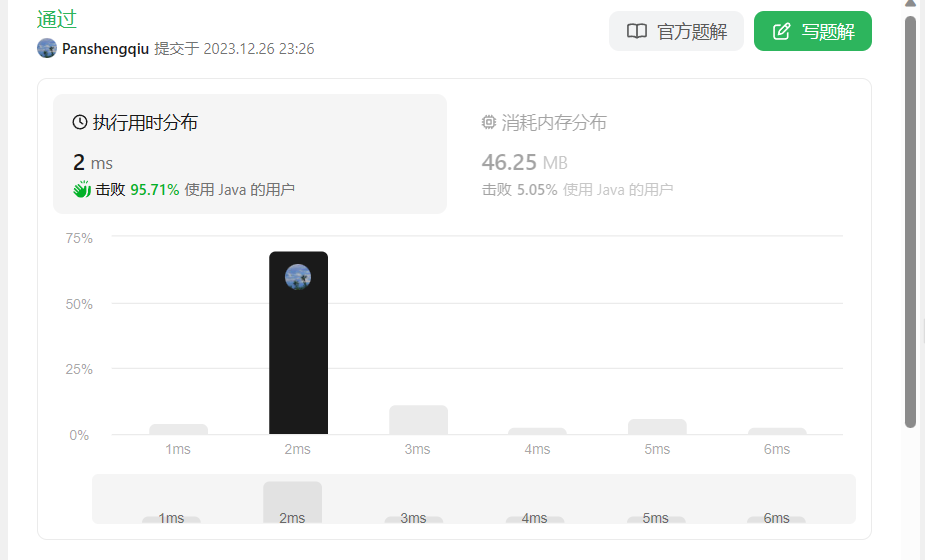
            }

        }

        return nums;

    }

}



## 两数之和



1. O(n2)算法

class Solution {

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

        int[] targetIdx = new int[2];

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

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

                if(nums[i] + nums[j] == target){

                    targetIdx[0] = i;

                    targetIdx[1] = j;

                    break;

                }

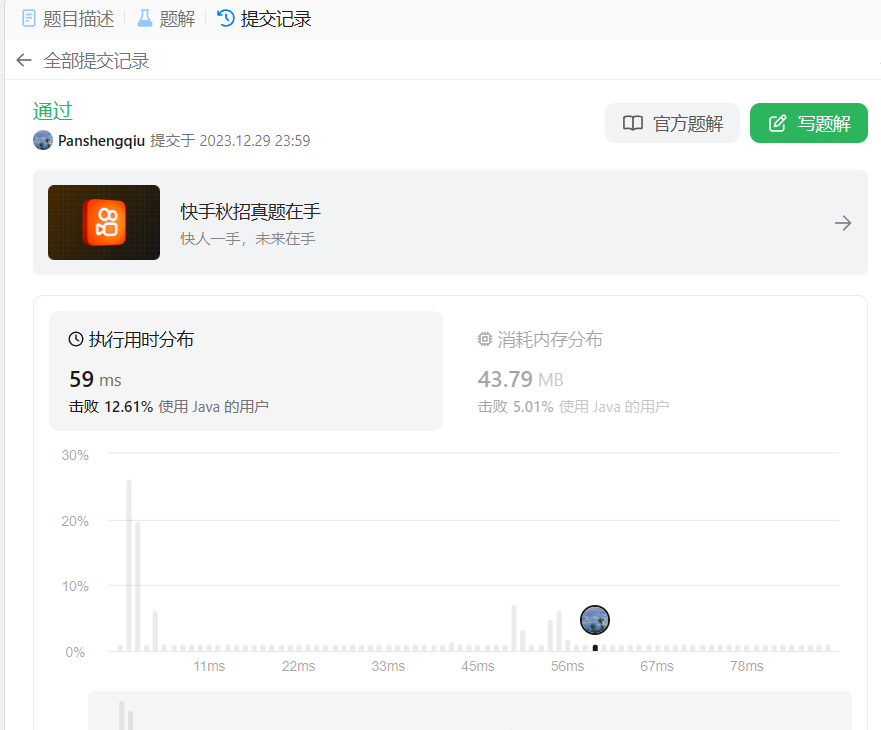
            }

        }

        return targetIdx;

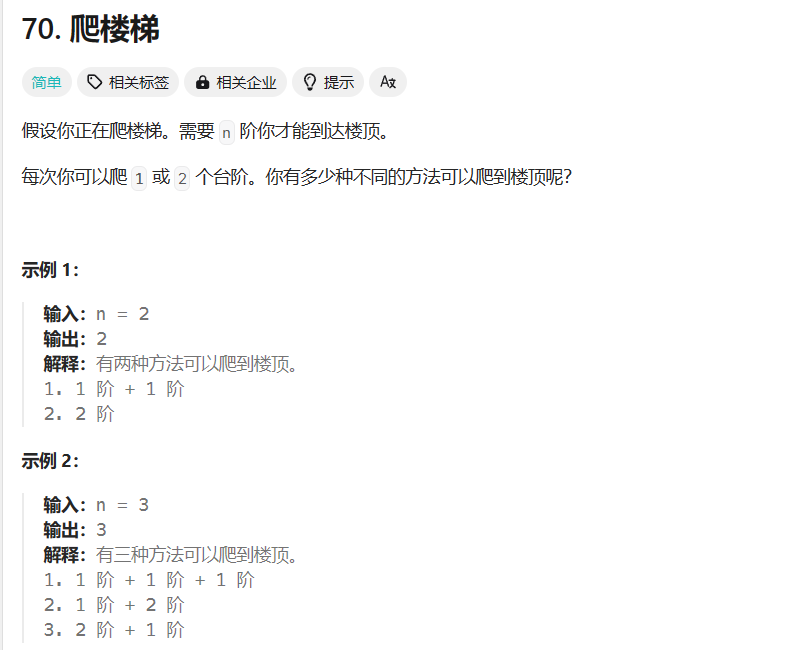
    }

}



1. O(n)算法

# 动态规划 上楼梯



代码：

class Solution {

    public int climbStairs(int n) {

        int[] dp = new int[n+1];

        dp[0] = 1;

        dp[1] = 1;

        for(int i = 2; i < n+1; i++){

            dp[i] = dp[i-1] + dp[i-2];

        }

        return dp[n];

    }

}

# 动态规划 不同路径



代码：

class Solution {

    public int uniquePaths(int m, int n) {

        int[][] dp = new int[m][n];

        for(int i = 0; i < m; i++){

            dp[i][0] = 1;

        }

        for(int j = 0; j < n; j++){

            dp[0][j] = 1;

        }

        for(int i = 1; i < m; i++){

            for(int j = 1; j < n; j++){

                dp[i][j] = dp[i-1][j] + dp[i][j-1];

            }

        }

        return dp[m-1][n-1];

    }

}

# 动态规划不同路径 II



代码：

class Solution {

    public int uniquePathsWithObstacles(int[][] obstacleGrid) {

        int m = obstacleGrid.length;

        int n = obstacleGrid[0].length;

        int[][] dp = new int[m][n];

        int x = 0, y = 0;

        for(int i = 0; i < m; i++){

            if(obstacleGrid[i][0] == 0){

                dp[i][0] = 1;

            }

            else{

                x = i;

                break;

            }

        }

        if(x != 0 && x < m)

            for(int i = x; i < m; i++)

                dp[i][0] = 0;

        for(int j = 0; j < n; j++){

            if(obstacleGrid[0][j] == 0){

                dp[0][j] = 1;

            }

            else{

                y = j;

                break;

            }

        }

        if(y != 0 && y < n)

            for(int j = y; j < n; j++)

                dp[0][j] = 0;

        for(int i = 1; i < m; i++){

            for(int j = 1; j < n; j++){

                if(obstacleGrid[i][j] == 1)

                    dp[i][j] = 0;

                else

                    dp[i][j] = dp[i-1][j] + dp[i][j-1];

            }

        }

        return dp[m-1][n-1];

    }

}