双指针算法

无隅

双指针

- 相遇型双指针
  - Two sum 类
  - Partition 类
  - 灌水类

- 同向型双指针
  - 窗口类
  - 快慢类

- 两双类(双指针with双数组)

相遇型双指针

一个数组，从两边往中间移动

Two Sum 类型题目

```
int sum = A[i] + A[j];
if (sum > targer) {
    j--;
    //write handle code
} else if (A[i] + A[j]) {
    i++;
    //write handle code
} else {
    //write handle code
    i++ or j--
}
```

四数之和

给定一个包含 n 个整数的数组 nums 和一个目标值 target，判断 nums 中是否存在四个元素 a，b，c 和 d ，使得 a + b + c + d 的值与 target 相等？找出所有满足条件且不重复的四元组。

注意：

答案中不可以包含重复的四元组。

示例：

给定数组 nums = [1, 0, -1, 0, -2, 2]，和 target = 0。

满足要求的四元组集合为： [[-1, 0, 0, 1], [-2, -1, 1, 2], [-2, 0, 0, 2]]

https://leetcode-cn.com/problems/4sum/description/

```java
public List<List<Integer>> fourSum(int[] nums, int target) {
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    HashSet<List<Integer>> set = new HashSet<>();
    Arrays.sort(nums);
    int start = 0;
    int end = nums.length - 1;

    for (int i = 0; i < nums.length - 3; i++) {
        for (int j = i + 1; j < nums.length - 2; j++) {
            start = j + 1;
            end = nums.length - 1;
            while (start < end) {
                if (nums[i] + nums[j] + nums[start] + nums[end] == target) {
                    ArrayList<Integer> list = new ArrayList<Integer>();
                    list.add(nums[i]);
                    list.add(nums[j]);
                    list.add(nums[start]);
                    list.add(nums[end]);
                    set.add(list);
                    start++;
                    end--;
                } else if (nums[i] + nums[j] + nums[start] + nums[end] < target) {
                    start++;
                } else {
                    end--;
                }
            }
        }
    }

    result.addAll(set);
    return result;

}
```

最接近的三数之和

给定一个包括 n 个整数的数组nums和一个目标值target。找出 nums中的三个整数，使得它们的和与 target 最接近。返回这三个数的和。假定每组输入只存在唯一答案

例如，给定数组 nums = [-1，2，1，-4]和target = 1. 与 target 最接近的三个数的和为 2. (-1 + 2 + 1 = 2).

https://leetcode.com/problems/3sum-closest/description/

```java
public int threeSumClosest(int[] nums, int target) {
    Arrays.sort(nums);
    int start = 0;
    int end = nums.length - 1;
    int min = Integer.MAX_VALUE /2;
    for (int i = 0; i < nums.length - 2; i++) {
        start = i + 1;
        end = nums.length - 1;
        while (start < end) {
            int sum = nums[i] + nums[start] + nums[end];
            if (sum == target) {
                return sum;
            } else if (sum < target) {
                start++;
            } else {
                end--;
            }
            min = Math.abs(sum - target) < Math.abs(min - target) ? sum : min;
        }
    }
    return min;
}
```

3Sum Smaller

给定一个n个整数的数组和一个目标整数target，

求满足下标为i、j、k的数组元素0 <= i < j < k < n，满足条件nums[i] + nums[j] + nums[k] < target这个的 (i, j, k) 的个数

```java
public int threeSumSmaller(int[] nums, int target) {
    if (nums == null || nums.length < 3) {
        return 0;
    }

    int count = 0;
    Arrays.sort(nums);

    for (int i = 2; i < nums.length; ++i) {
        for (int l = 0, r = i - 1; l < r; ++l) {
            while (l < r && nums[i] + nums[l] + nums[r] >= target) {
                r--;
            }
            if (l < r && nums[i] + nums[l] + nums[r] < target) {
                count += r - l;
            }
        }
    }
    return count;
}
```

灌水类

```
if (heights[left] > heights[left]) {
    right--;
} else if (heights[left] < heights[right]) {
    left++;
} else {
    left++ or right++;
}
```

接雨水

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水

https://leetcode-cn.com/problems/trapping-rain-water/description/

```java
public int trap(int[] height) {
    int left = 0, right = height.length - 1;
    int res = 0;
    if(left >= right)
        return res;
    int leftheight = height[left];
    int rightheight = height[right];
    while(left < right) {
        if(leftheight < rightheight) {
            left ++;
            if(leftheight > height[left]) {
                res += (leftheight - height[left]);
            } else {
                leftheight = height[left];
            }
        } else {
            right --;
            if(rightheight > height[right]) {
                res += (rightheight - height[right]);
            } else {
                rightheight = height[right];
            }
        }
    }
    return res;
}
```

```java
public int trap(int[] height) {
    if (height == null || height.length == 0) {
        return 0;
    }
    int ans = 0;
    int l = 0, r = height.length - 1;

    while (l < r) {
        int left = height[l];
        int right = height[r];
        if (height[l] <= height[r]) {
            while (l < r && left >= height[++l]) {
                ans += left - height[l];
            }
        } else {
            while (l < r && height[--r] <= right) {
                ans += right - height[r];
            }
        }
    }
    return ans;
}
```

盛最多水的容器

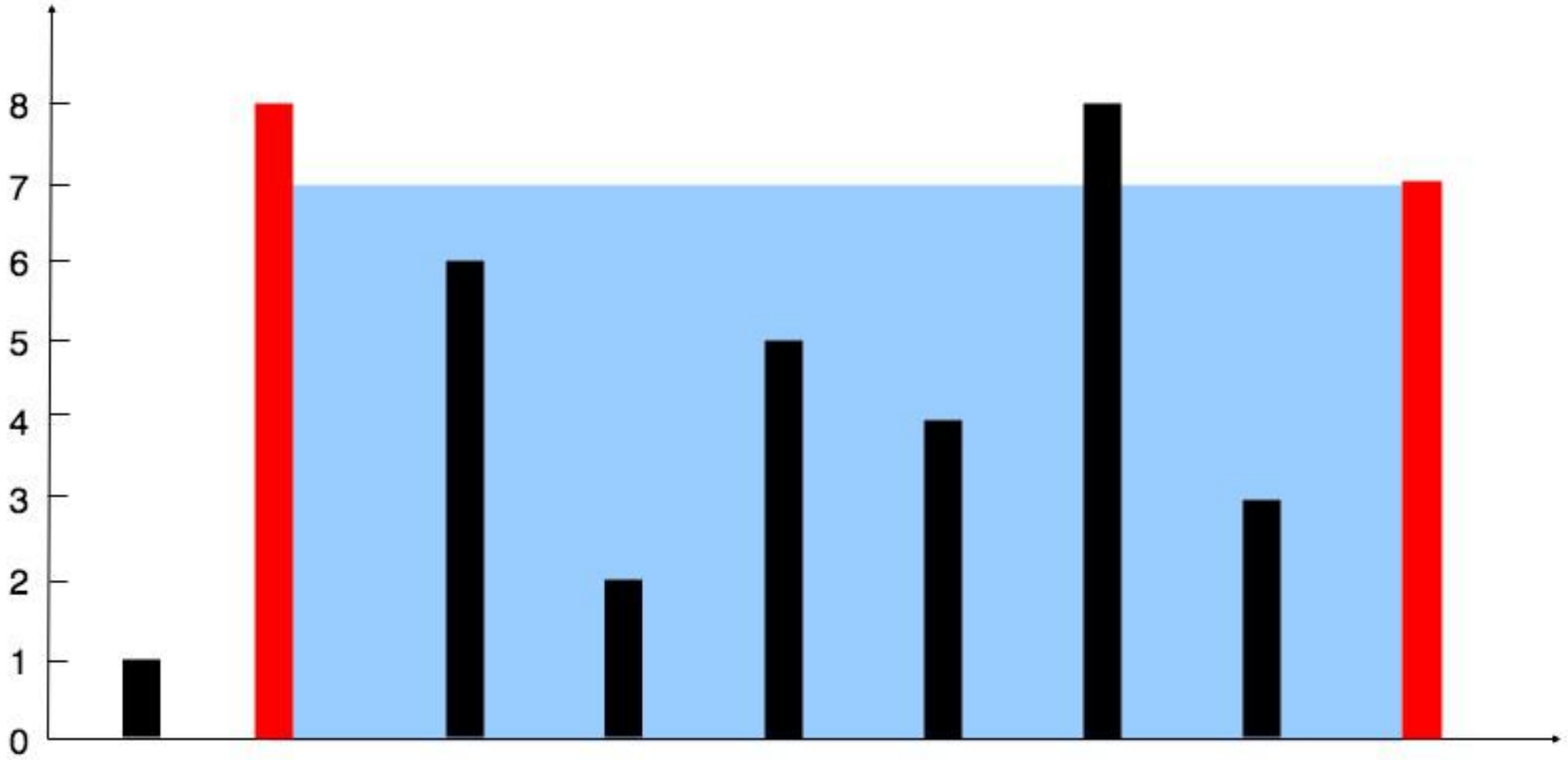
给定 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0)。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器，且 n 的值至少为 2

输入：[1,8,6,2,5,4,8,3,7] 输出：49

https://leetcode-cn.com/problems/container-with-most-water/description/

盛最多水的容器



```java
public int maxArea(int[] height) {
    if (height == null || height.length < 2) {
        return 0;
    }

    int left = 0;
    int right = height.length - 1;
    int area = 0;
    while (left <= right) {
        area = Math.max(area, computeArea(left, right, height));
        if (height[left] <= height[right]) {
            left++;
        }
        else {
            right--;
        }
    }
    return area;
}


public int computeArea(int left, int right, int[] height) {
    int h = Math.min(height[left], height[right]);
    return (right - left) * h;
}
```

Partition 类

Partition 类模板

```java
public int partition(int[] nums, int l, int r) {
    int left = l;
    int right = r;
    int pivot = nums[left];

    while (left < right) {
        while (left < right && nums[right] >= pivot) {
            right--;
        }
        nums[left] = nums[right];
        while (left < right && nums[left] <= pivot) {
            left++;
        }
        nums[right] = nums[left];
    }

    nums[left] = pivot;
    return left;
}
```

颜色分类

给定一个包含红色、白色和蓝色，一共 n 个元素的数组，原地对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。
此题中，我们使用整数 0、 1 和 2 分别表示红色、白色和蓝色。

注意:

不能使用代码库中的排序函数来解决这道题。
示例:
输入: [2,0,2,1,1,0] 输出: [0,0,1,1,2,2]
进阶：
一个直观的解决方案是使用计数排序的两趟扫描算法。

首先，迭代计算出0、1 和 2 元素的个数，然后按照0、1、2的排序，重写当前数组。
你能想出一个仅使用常数空间的一趟扫描算法吗？

```java
public void sortColors(int[] nums) {
    int i = 0;
    int j = nums.length - 1;
    while  (i <= j){
        while(i <= j && nums[i] < 1)    i++;
        while(i <= j && nums[j] >= 1)   j--;
        if (i <= j){
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
            i++;
            j--;
        }
    }
    j = nums.length - 1;
    while (i <= j){
        while(i <= j && nums[i] < 2)    i++;
        while(i <= j && nums[j] >= 2)   j--;
        if (i <= j){
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
            i++;
            j--;
        }
    }
}
```

Valid Palindrome

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Note: For the purpose of this problem, we define empty string as valid palindrome.

Example 1:

Input: "A man, a plan, a canal: Panama" Output: true

Example 2:

Input: "race a car" Output: false

```java
public boolean isPalindrome(String s) {
    s = s.toLowerCase();
    s = s.replaceAll("\\W", "");


    for (int i = 0; i < s.length() / 2; i++){
        if (s.charAt(i) != s.charAt(s.length() - 1- i)){
            return false;
        }
    }
    return true;
}
```

同向型窗口类

窗口类指针移动模板

通过两层for循环改进算法

```
for (i = 0; i < n; i++) {
    while(j < n) {
        if(满足条件) {
            j++;
            //更新j状态
        else(不满足条件) {
            break;
        }

    }
    //更新i状态
}
```

长度最小的子数组

给定一个含有 n 个正整数的数组和一个正整数 s ，找出该数组中满足其和 ≥ s 的长度最小的连续子数组。如果不存在符合条件的连续子数组，返回 0。

示例:

输入: s = 7, nums = [2,3,1,2,4,3] 输出: 2 解释: 子数组 [4,3] 是该条件下的长度最小的连续子数组。

https://leetcode-cn.com/problems/minimum-size-subarray-sum/description/

```java
public int minSubArrayLen(int s, int[] nums) {
    int i = 0;
    int j = 0;
    int sum = 0;
    int ans = Integer.MAX_VALUE;
    for(i = 0; i < nums.length; i++) {
        while(j < nums.length && sum < s) {
            sum += nums[j];
            j++;
        }
        if(sum >= s) {
            ans = Math.min(ans, j - i);
        }
        sum -= nums[i];
    }

    if (ans == Integer.MAX_VALUE) ans = 0;
    return ans;
}
```

无重复字符的最长子串

给定一个字符串，找出不含有重复字符的最长子串的长度。

示例 1:

输入: "abcabcbb" 输出: 3 解释: 无重复字符的最长子串是 "abc"，其长度为 3。 示例 2:

输入: "bbbbb" 输出: 1 解释: 无重复字符的最长子串是 "b"，其长度为 1。 示例 3:

输入: "pwwkew" 输出: 3 解释: 无重复字符的最长子串是 "wke"，其长度为 3。 请注意，答案必须是一个子串，"pwke" 是一个子序列而不是子串。

https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/description/

```java
public int lengthOfLongestSubstring(String s) {
    if (s == null || s.length() == 0) {
        return 0;
    }
    HashMap<Character, Integer> map = new HashMap<Character, Integer>();
    int i = 0;
    int start = -1;
    int length = 0;
    for (i = 0; i < s.length(); i++) {
        Character ch = s.charAt(i);
        if (map.containsKey(ch)) {
            int index = map.get(ch);
            start = Math.max(start, index);
        }
        length = Math.max(length, i - start);
        map.put(ch, i);
    }

    return length;
}
```

最小覆盖子串

给定一个字符串 S 和一个字符串 T，请在 S 中找出包含 T 所有字母的最小子串。

示例：

输入：S = "ADOBECODEBANC", T = "ABC" 输出："BANC"说明：

如果 S 中不存这样的子串，则返回空字符串 ""。

如果 S 中存在这样的子串，我们保证它是唯一的答案

https://leetcode-cn.com/problems/minimum-window-substring/description/

```java
public String minWindow(String s, String t) {
    if (s == null || s.length() == 0 ||
            t == null || t.length() == 0) {
        return "";
    }
    int[] sHash = new int[256];
    int[] tHash = new int[256];
    int ans = Integer.MAX_VALUE;
    String minStr = "";

    initTargetHash(tHash, t);

    int i = 0;
    int j = 0;

    for (i = 0; i < s.length(); i++) {
        while (j < s.length() && !isValid(sHash, tHash)) {
            sHash[s.charAt(j)]++;
            if (j < s.length()) {
                j++;
            } else {
                break;
            }
        }
        if (isValid(sHash, tHash)) {
            if (ans > j - i) {
                ans = Math.min(ans, j - i);
                minStr = s.substring(i, j);
            }
        }
        sHash[s.charAt(i)]--;
    }
    return minStr;
}
```

```java
boolean isValid(int[] sHash, int[] tHash) {
    for (int i = 0; i < sHash.length; i++) {
        if (tHash[i] > sHash[i]) {
            return false;
        }
    }
    return true;
}

public void initTargetHash(int[] tHash, String t) {
    for (int i = 0; i < t.length(); i++) {
        tHash[t.charAt(i)]++;
    }
}
```

两双类(双指针with双数组)

Merge Two Sorted Lists

最小差 The Smallest Difference

给定两个整数数组（第一个是数组 A，第二个是数组 B），在数组 A 中取 A[i]，数组 B 中取 B[j]，A[i] 和 B[j]两者的差越小越好(|A[i] - B[j]|)。返回最小差。

```java
public int smallestDifference(int[] A, int[] B) {
    if (A == null || A.length == 0 || B == null || B.length == 0) {
        return 0;
    }

    Arrays.sort(A);
    Arrays.sort(B);

    int ai = 0, bi = 0;
    int min = Integer.MAX_VALUE;
    while (ai < A.length && bi < B.length) {
        min = Math.min(min, Math.abs(A[ai] - B[bi]));
        if (A[ai] < B[bi]) {
            ai++;
        } else {
            bi++;
        }
    }
    return min;
}
```