双指针问题

双指针算法

通过使用两个指针，分别从数组或链表的头部和尾部开始向中间移动，常用来解决数组或链表问题

- 判断是否满足满足条件的两个数
- 寻找满足某条件的连续子序列
- 判断一个字符串是否是回文串
- 讲一个数组或链表按照某种方式重新排序

分类

- 快慢指针

  通常用于链表问题中，如判断链表是否有环或找到链表的中间节点等。通过使用两个指针，一个快指针和一个慢指针，快指针每次移动2步，慢指每次移动1步，当快指到达链表尾部时，慢指针就到达了链表中间位置。

- 左右指针

  常用于数组问题中，如在有序数组中查找目标元素等。通过使用两个指针，一个左指针和一个右指针，左指针从数组头部开始向右移动，右指针从数组尾部开始向左移动，根据具体问题来移动指针，最终得出结果

- 对撞指针

  通常用于有序数组或链表问题中，如判断回文字符串或找到两个数的平方和等。通过使用两个指针，一个指向数组或链表的头部，另一个指向尾部，根据具体问题来移动指针，最终得出结果。

模版

```
for(int i = 0, j = 0; i < n; i++)
{
    while(j < i && check(i, j)) j++;
    // 具体问题的逻辑
}
常见问题分类
    对于一个序列，用两个指针维护一段区间
    对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作
```

Acwing题目

1.最长连续不重复子序列

题目描述

给定一个长度为 n 的整数序列，请找出最长的不包含重复的数的连续区间，输出它的长度。

输入格式

第一行包含整数 n。

第二行包含 n 个整数（均在 $0 \sim 10^5$ 范围内），表示整数序列。

输出格式

共一行，包含一个整数，表示最长的不包含重复的数的连续区间的长度。

数据范围

$1 \leq n \leq 10^5$

输入样例：

```
5
1 2 2 3 5
```

输出样例：

```
3
```

代码

```cpp
#include<iostream>
using namespace std;
const int N = 1e5 + 10;

int n;
int a[N], s[N];

int main(void)
{
    int r = 0;
    cin >> n;

    for(int i = 0, j = 0; i < n; i++)
    {
        cin >> a[i];
        s[a[i]]++;

        while(s[a[i]] > 1) s[a[j++]] --;
        r = max(r, i - j + 1);
```

```
    }
    cout << r;
    return 0;
}
```

思路

遍历数组a中的每一个元素a[i], 对于每一个i，找到j使得双指针[j, i]维护的是以a[i]结尾的最长连续不重复子序列，长度为i - j + 1, 将这一长度与r的较大者更新给r。
对于每一个i，如何确定j的位置：由于[j, i - 1]是前一步得到的最长连续不重复子序列，所以如果[j, i]中有重复元素，一定是a[i]，因此右移j直到a[i]不重复为止（由于[j, i - 1]已经是前一步的最优解，此时j只可能右移以剔除重复元素a[i]，不可能左移增加元素，因此，j具有“单调性”、本题可用双指针降低复杂度）。
用数组s记录子序列a[j ~ i]中各元素出现次数，遍历过程中对于每一个i有四步操作：cin元素a[i] -> 将a[i]出现次数s[a[i]]加1 -> 若a[i]重复则右移j（s[a[j]]要减1） -> 确定j及更新当前长度i - j + 1给r。

2.数组元素的目标和

题目描述

给定两个升序排序的有序数组 A 和 B，以及一个目标值 x。

数组下标从 0 开始。

请你求出满足 $A[i]+B[j]=x$ 的数对 (i,j)。

数据保证有唯一解。

输入格式

第一行包含三个整数 n,m,x，分别表示 A 的长度，B 的长度以及目标值 x。

第二行包含 n 个整数，表示数组 A。

第三行包含 m 个整数，表示数组 B。

输出格式

共一行，包含两个整数 i 和 j。

数据范围

数组长度不超过 10^5。
同一数组内元素各不相同。
$1 \leq$ 数组元素 $\leq 10^9$

输入样例：

```
4 5 6
1 2 4 7
3 4 6 8 9
```

输出样例：

```
1 1
```

思路

i 从0开始遍历，从前往后遍历

j 从 m - 1开始遍历，从后往前遍历

j指针不后退，因为 如果 当前 a[i] + b[j] > x ，那么a[i+1] + b[j] 必然 > x，所以j没有后退的必要

代码

```cpp
#include<iostream>
#include<cstdio>
using namespace std;
const int N = 1e5 + 10;

int n, m, k;
int a[N], b[n];
int main(void)
{
    cin >> n >> m >> k;
    for(int i = 0; i < n; i++) cin >> a[i];
    for(int j = 0; j < m; j++) cin >> b[j];

    for(int i = 0, j = m - 1; i < n; i++)
    {
        while(j >= 0 && a[i] + b[j] > k) j--;
        if(j >= 0 && a[i] + b[j] == k)
            cout << i << " " << j << endl;
    }
    return 0;
}
```

3.判断子序列

题目描述

给定一个长度为 n 的整数序列 $a_1, a_2, \ldots, a_n$ 以及一个长度为 m 的整数序列 $b_1, b_2, \ldots, b_m$。

请你判断 a 序列是否为 b 序列的子序列。

子序列指序列的一部分项按**原有次序排列**而得的序列，例如序列 $\{a_1, a_3, a_5\}$ 是序列 $\{a_1, a_2, a_3, a_4, a_5\}$ 的一个子序列。

输入格式

第一行包含两个整数 n, m。

第二行包含 n 个整数，表示 $a_1, a_2, \ldots, a_n$。

第三行包含 m 个整数，表示 $b_1, b_2, \ldots, b_m$。

输出格式

如果 a 序列是 b 序列的子序列，输出一行 `Yes`。

否则，输出 `No`。

数据范围

$1 \le n \le m \le 10^5$,
$-10^9 \le a_i, b_i \le 10^9$

输入样例：

```
3 5
1 3 5
1 2 3 4 5
```

输出样例：

```
Yes
```

思路

1.j指针用来扫描整个b数组，i指针用来扫描a数组。若发现a[i] == b[j]，则让i指针后移一位。
2.整个过程中，j指针不断后移，而i指针只有当匹配成功时才后移一位，若最后若i == n，则说明匹配成功。

代码

```cpp
#include<iostream>
using namespace std;
const int N = 1e5 + 10;
int n, m;
int a[N], b[N];
int main(void)
{
    cin >> n >> m;
    for(int i = 0; i < n; i++) cin >> a[i];
    for(int i = 0; i < m; i++) cin >> b[i];

    int i = 0, j = 0;
    while(i < n && j < m)
    {
        if(a[i] == b[j]) i++;
        j++;
    }
    if(i == n) puts("Yes");
    else puts("No");
    return 0;
}
```

Leetcode 题目

1.验证回文串

题目描述

如果在将所有大写字符转换为小写字符、并移除所有非字母数字字符之后，短语正着读和反着读都一样。则可以认为该短语是一个 **回文串** 。

字母和数字都属于字母数字字符。

给你一个字符串 `s`，如果它是 **回文串** ，返回 `true` ；否则，返回 `false` 。

示例 1：

> **输入：** s = "A man, a plan, a canal: Panama"
> **输出：** true
> **解释：** "amanaplanacanalpanama" 是回文串。

示例 2：

> **输入：** s = "race a car"
> **输出：** false
> **解释：** "raceacar" 不是回文串。

示例 3：

> **输入：** s = " "
> **输出：** true
> **解释：** 在移除非字母数字字符之后，s 是一个空字符串 "" 。
> 由于空字符串正着反着读都一样，所以是回文串。

思路及代码

思路1：双指针

使用双指针。初始时，左右指针分别指向 sgood 的两侧，随后我们不断地将这两个指针相向移动，每次移动一步，并判断这两个指针指向的字符是否相同。当这两个指针相遇时，就说明 sgood 时回文串

```
class Solution{
    public:
        bool isPalindrome(string s){
            string s_good;
            for(auto ch : s)
            {
                if(isalnum(ch))
                    s_good += tolower(ch);
            }
            int n = s_good.size();
```

```
            int i = 0, j = n - 1;
            while(i < j)
            {
                if(s_good[i] != s_good[j])
                    return false;
                i++;
                j--;
            }
            return true;
        }
}
```

> `isalnum` 是判断 字符串中的每个字符是否是字母或数字
>
> `tolower` 函数将字符串中每个字符转换成小写
>
> 这两个函数都在 `cctype` 库中

2.判断子序列

题目描述

给定字符串 **s** 和 **t** ，判断 **s** 是否为 **t** 的子序列。

字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，`"ace"`是`"abcde"`的一个子序列，而`"aec"`不是）。

进阶:

如果有大量输入的 S，称作 S1, S2, ... , Sk 其中 k >= 10亿，你需要依次检查它们是否为 T 的子序列。在这种情况下，你会怎样改变代码?

致谢:

特别感谢 @pbrother 添加此问题并且创建所有测试用例。

示例 1:

> **输入:** `s = "abc", t = "ahbgdc"`
> **输出:** `true`

示例 2:

> **输入:** `s = "axc", t = "ahbgdc"`
> **输出:** `false`

提示:

- `0 <= s.length <= 100`
- `0 <= t.length <= 10^4`
- 两个字符串都只由小写字符组成。

思路及代码

思路1：双指针

本题询问的是，s 是否是 t 的子序列，因此只要能找到任意一种 s 在 t 中出现的方式，即可认为 s 是 t 的子序列。

而当我们从前往后匹配，可以发现每次贪心地匹配靠前的字符是最优决策。

这样，我们初始化两个指针 i 和 j，分别指向 s 和 t 的初始位置。每次贪心地匹配，匹配成功则 i 和 j 同时右移，匹配 s 的下一个位置，匹配失败则 j 右移，i 不变，尝试用 t 的下一个字符匹配 s。

最终如果 i 移动到 s 的末尾，就说明 s 是 t 的子序列。

```
class Solution {
    public:
        isSubsequence(string s, string t)
        {
            int n = s.length(), m = t.length();
            int i = 0, j = 0;
            while(i < n && j < m)
            {
                if(s[i] == s[j])
                    i++;
                j++;
            }
            if(i == n)
                return true;
            return false;
        }
}
```

3.两数之和

题目描述

给你一个下标从 **1** 开始的整数数组 `numbers` ，该数组已按 **非递减顺序排列** ，请你从数组中找出满足相加之和等于目标数 `target` 的两个数。如果设这两个数分别是 `numbers[index1]` 和 `numbers[index2]` ，则 `1 <= index1 < index2 <= numbers.length` 。

以长度为 2 的整数数组 `[index1, index2]` 的形式返回这两个整数的下标 `index1` 和 `index2`。

你可以假设每个输入 **只对应唯一的答案** ，而且你 **不可以** 重复使用相同的元素。

你所设计的解决方案必须只使用常量级的额外空间。

示例 1：

> **输入：** numbers = [***2***,***7***,11,15], target = 9
> **输出：** [1,2]
> **解释：** 2 与 7 之和等于目标数 9 。因此 index1 = 1, index2 = 2 。返回 [1, 2] 。

示例 2：

> **输入：** numbers = [***2***,3,***4***], target = 6
> **输出：** [1,3]
> **解释：** 2 与 4 之和等于目标数 6 。因此 index1 = 1, index2 = 3 。返回 [1, 3] 。

示例 3：

> **输入：** numbers = [***-1***,***0***], target = -1
> **输出：** [1,2]
> **解释：** -1 与 0 之和等于目标数 -1 。因此 index1 = 1, index2 = 2 。返回 [1, 2] 。

思路及代码

初始时两个指针分别指向第一个元素位置和最后一个元素的位置。每次计算两个指针指向的两个元素之和，并和目标值比较。如果两个元素之和等于目标值，则发现了唯一解。如果两个元素之和小于目标值，则将左侧指针右移一位。如果两个元素之和大于目标值，则将右侧指针左移一位。移动指针之后，重复上述操作，直到找到答案。

```
class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
```

```
        int low = 0, high = numbers.size() - 1;
        while (low < high) {
            int sum = numbers[low] + numbers[high];
            if (sum == target) {
                return {low + 1, high + 1};
            } else if (sum < target) {
                ++low;
            } else {
                --high;
            }
        }
        return {-1, -1};
    }
};
```

4.盛最多水的容器

题目描述

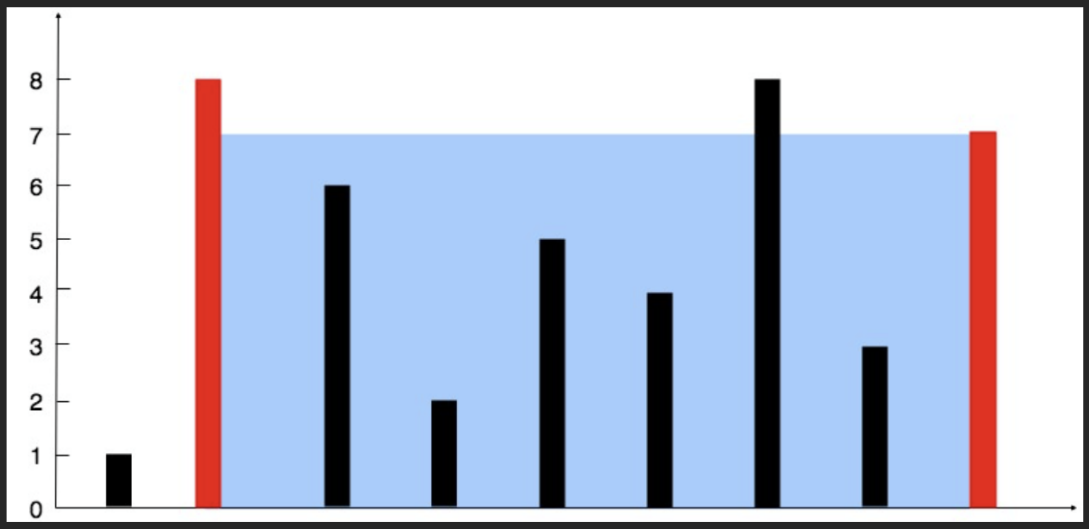
给定一个长度为 `n` 的整数数组 `height` 。有 `n` 条垂线，第 `i` 条线的两个端点是 `(i, 0)` 和 `(i, height[i])` 。

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。

返回容器可以储存的最大水量。

说明：你不能倾斜容器。

示例 1：



> **输入：** [1,8,6,2,5,4,8,3,7]
> **输出：** 49
> **解释：** 图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例 2：

> **输入：** height = [1,1]
> **输出：** 1

思路及代码

在初始时，左右指针分别指向数组的左右两端，它们可以容纳的水量为 $\min(1, 7) * 8 = 8$。

此时我们需要移动一个指针。移动哪一个呢？直觉告诉我们，应该移动对应数字较小的那个指针（即此时的左指针）。这是因为，由于容纳的水量是由

$$\text{两个指针指向的数字中较小值} * \text{指针之间的距离}$$

决定的。如果我们移动数字较大的那个指针，那么前者「两个指针指向的数字中较小值」不会增加，后者「指针之间的距离」会减小，那么这个乘积会减小。因此，我们移动数字较大的那个指针是不合理的。因此，我们移动 **数字较小的那个指针。**

> 有读者可能会产生疑问：**我们可不可以同时移动两个指针？** 先别急，我们先假设 **总是移动数字较小的那个指针** 的思路是正确的，在走完流程之后，我们再去进行证明。

```cpp
class Solution {
public:
    int maxArea(vector<int>& height) {
        int n = height.size();
        int l = 0, r = n - 1;
        int result = 0;
        while(l < r)
        {
            int len = min(height[l], height[r]) * (r - l);
            result = max(result, len);
            if(height[l] < height[r])
                l ++;
            else
                r --;
        }
        return result;
    }
};
```

5.三数之和

题目描述

给你一个整数数组 `nums` ，判断是否存在三元组 `[nums[i], nums[j], nums[k]]` 满足 `i != j`、`i != k` 且 `j != k` ，同时还满足 `nums[i] + nums[j] + nums[k] == 0` 。请

你返回所有和为 `0` 且不重复的三元组。

注意： 答案中不可以包含重复的三元组。

示例 1：

> **输入：** nums = [-1,0,1,2,-1,-4]
> **输出：** [[-1,-1,2],[-1,0,1]]
> **解释：**
> nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0 。
> nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0 。
> nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0 。
> 不同的三元组是 [-1,0,1] 和 [-1,-1,2] 。
> 注意，输出的顺序和三元组的顺序并不重要。

示例 2：

> **输入：** nums = [0,1,1]
> **输出：** []
> **解释：** 唯一可能的三元组和不为 0 。

示例 3：

> **输入：** nums = [0,0,0]
> **输出：** [[0,0,0]]
> **解释：** 唯一可能的三元组和为 0 。

思路及代码

唉，把第一个数num[i]的相反数当成target，对j 和k求两数之和就可以了

> md，为什么当年算法期末这都不会啊

```cpp
class Solution {
    public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        int n = nums.size();
        sort(nums.begin(), nums.end());
        vector<vector<int>> results;

        for(int i = 0; i < n; i ++)
        {
            // 去重
            if(nums[i] > 0)
                break;
            if(i > 0 && nums[i] == nums[i - 1])
                continue;
            int target = -nums[i];
            int j = i + 1, k = n - 1;
            while(j < k)
            {
                int sum = nums[j] + nums[k];
                if(j > i + 1 && nums[j] == nums[j - 1])
                {
                    j ++;
                    continue;
                }
                else if(k < n - 1 && nums[k] == nums[k + 1])
                {
```

```
                    k --;
                    continue;
                }
                if(sum == target)
                {
                    results.push_back({nums[i], nums[j], nums[k]});
                    j ++;
                    k --;
                }
                else if(sum < target)
                    j ++;
                else
                    k --;
            }
        }
        return results;
    }
};
```