双指针算法之同向双指针（上）

主讲人 令狐冲
课程版本 v7.0

同向双指针

两根指针都从头出发，朝着同一个方向移动

两数之差问题

https://www.lintcode.com/problem/two-sum-difference-equals-to-target

https://www.jiuzhang.com/solutions/two-sum-difference-equals-to-target

求两数之差等于给定的 target，不适用额外空间

Two Sum 的第 11 个变形题

当不能使用哈希表时

可以在排序数据集上进行二分来替代

不能使用哈希表的情况比如数据集很大

或者题目要求不适用额外空间

使用二分法优化到 O(nlogn)


```java
public int[] twoSum7(int[] nums, int target) {
    if (nums == null || nums.length < 2) {
        return new int[]{-1, -1};
    }

    target = Math.abs(target);
    for (int i = 0; i < nums.length; i++) {
        int j = binarySearch(nums, i + 1, nums.length - 1, target + nums[i]);
        if (j != -1) {
            return new int[]{nums[i], nums[j]};
        }

    }

    return new int[]{-1, -1};
}

private int binarySearch(int[] nums, int start, int end, int target) {
    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        if (nums[mid] == target) {
            return mid;
        }
        if (nums[mid] < target) {
            start = mid;
        } else {
            end = mid;
        }
    }

    if (nums[start] == target) {
        return start;
    }
    if (nums[end] == target) {
        return end;
    }

    return -1;
}
```

```python
def twoSum7(self, nums, target):
    if not nums:
        return [-1, -1]

    target = abs(target)
    for i in range(len(nums)):
        j = self.binary_search(nums, i + 1, len(nums) - 1, target + nums[i])
        if j != -1:
            return [nums[i], nums[j]]

    return [-1, -1]

def binary_search(self, nums, start, end, target):
    while start + 1 < end:
        mid = (start + end) // 2
        if nums[mid] < target:
            start = mid
        else:
            end = mid
    if nums[start] == target:
        return start
    if nums[end] == target:
        return end

    return -1
```

这个算法的可优化之处在哪儿？

当 i 向右移动时

满足 nums[j] - nums[i] = target 的 j 也会向右移动

没有必要在去 i + 1 ~ j - 1 之间寻找

使用同向双指针算法


```java
public int[] twoSum7(int[] nums, int target) {
    if (nums == null || nums.length < 2) {
        return new int[]{-1, -1};
    }

    target = Math.abs(target);

    int j = 1;
    for (int i = 0; i < nums.length; i++) {
        j = Math.max(j, i + 1);
        while (j < nums.length && nums[j] - nums[i] < target) {
            j++;
        }
        if (j >= nums.length) {
            break;
        }
        if (nums[j] - nums[i] == target) {
            return new int[]{nums[i], nums[j]};
        }
    }

    return new int[]{-1, -1};
}
```

```python
def twoSum7(self, nums, target):
    if not nums:
        return [-1, -1]

    target = abs(target)
    j = 1
    for i in range(len(nums)):
        j = max(j, i + 1)
        while j < len(nums) and nums[j] - nums[i] < target:
            j += 1
        if j >= len(nums):
            break
        if nums[j] - nums[i] == target:
            return [nums[i], nums[j]]

    return [-1, -1]
```

同向双指针的模板

```
j = 0 or j = 1
for i from 0 to (n - 1)
    while j < n and (i, j的搭配不满足条件)
        j += 1
    if (i, j的搭配满足条件)
        处理i, j的这次搭配
```

同向双指针复杂度 = O(n)

两根指针同向而行，都不会“回头”

每个指针访问数组中每个元素最多一次

全零子串问题

https://www.lintcode.com/problem/number-of-substrings-with-all-zeroes/

求出字符串中全0子串的个数

001000 有 5个0、3个00，1个000，共9个子串

套用模板


```python
def stringCount(self, inputStr):
    if not inputStr:
        return 0

    j, answer = 1, 0
    for i in range(len(inputStr)):
        if inputStr[i] != '0':
            continue
        j = max(j, i + 1)
        while j < len(inputStr) and inputStr[j] == '0':
            j += 1
        answer += j - i

    return answer
```

```java
public int stringCount(String str) {
    if (str == null) {
        return 0;
    }

    int j = 1, answer = 0;
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) != '0') {
            continue;
        }

        j = Math.max(j, i + 1);
        while (j < str.length() && str.charAt(j) == '0') {
            j++;
        }

        answer += j - i;
    }

    return answer;
}
```

数组去重

https://www.lintcode.com/problem/remove-duplicate-numbers-in-array/

去掉未排序数组中的重复元素

[1,3,1,2,0,2] => [1,2,3,0, ?, ?]

? 的位置放什么数无所谓

要求在 原数组上进行操作，也就是额外空间复杂度O(1)

继续套模板


```python
def deduplication(self, nums):
    if not nums:
        return 0

    nums.sort()
    j = 1
    for i in range(len(nums)):
        while j < len(nums) and nums[j] == nums[i]:
            j += 1
        if j >= len(nums):
            break
        nums[i + 1] = nums[j]

    return i + 1
```

```java
public int deduplication(int[] nums) {
    if (nums == null || nums.length == 0) {
        return 0;
    }

    Arrays.sort(nums);
    int i, j = 1;
    for (i = 0; i < nums.length; i++) {
        while (j < nums.length && nums[i] == nums[j]) {
            j++;
        }
        if (j >= nums.length) {
            break;
        }
        nums[i + 1] = nums[j];
    }

    return i + 1;
}
```