链滴

算法学习之路 | 双指针

作者：qq692310342
原文链接：https://ld246.com/article/1565767242621
来源网站：链滴

Question 1 两数之和 II - 输入有序数组

```
/**
 * 给定一个已按照升序排列 的有序数组，找到两个数使得它们相加之和等于目标数。
 *
 * 函数应该返回这两个下标值 index1 和 index2，其中 index1 必须小于 index2。
 *
 * 思路：
 * 使用双指针，一个指针指向值较小的元素，一个指针指向值较大的元素。
 * 指向较小元素的指针从头向尾遍历，指向较大元素的指针从尾向头遍历。
 * 如果两个指针指向元素的和 sum == target，那么得到要求的结果；
 * 如果 sum > target，移动较大的元素，使 sum 变小一些；
 * 如果 sum < target，移动较小的元素，使 sum 变大一些。
 */
public class Question1 {

    public int[] twoSum(int[] numbers, int target) {
//        设置两个指针
        int p1 = 0;
        int p2 = numbers.length - 1;
//        设置输出
        while (p1 < p2) {
            int result = numbers[p1] + numbers[p2];
            if (result == target) {
                return new int[]{p1+1,p2+1};
            }else if(result>target){
                p2--;
            }else if(result<target){
                p1++;
            }
        }
        return null;
    }
}
```

Question 2 平方数之和

```
/**
 * 给定一个非负整数 c ，你要判断是否存在两个整数 a 和 b，使得 a2 + b2 = c。
 */
public class Question2 {
    public boolean judgeSquareSum(int c) {
        int i = 0;
//        取c的开方  更高效
        int j = (int) Math.sqrt(c);
//        设置头尾两个指针
        while (i <= j) {
            int powSum = i * i + j * j;
            if (powSum == c) {
                return true;
            } else if (powSum > c) {
                j--;
```

```
            }else{
                i++;
            }
        }
        return false;
    }
}
```

Question 3 反转字符串中的元音字母

```
/**
 * 编写一个函数，以字符串作为输入，反转该字符串中的元音字母。
 */
public class Question3 {

    private final static HashSet<Character> vowels = new HashSet<>(
            Arrays.asList('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'));

    public String reverseVowels(String s) {
        int i = 0;
        int j = s.length() - 1;
        char[] result = new char[s.length()];
        while (i <= j) {
            char ci = s.charAt(i);
            char cj = s.charAt(j);
            if (!vowels.contains(ci)) {
                result[i] = ci;
                i++;
            } else if (!vowels.contains(cj)) {
                result[j] = cj;
                j--;
            } else {
//                到了找到了两个元音
                result[i] = cj;
                result[j] = ci;
                i++;
                j--;
            }
        }
        return new String(result);
    }

}
```

Question 4 验证回文字符串 II

```
/**
 * 给定一个非空字符串 s，最多删除一个字符。判断是否能成为回文字符串。
 */
public class Question4 {

    public boolean validPalindrome(String s) {
        for (int i = 0, j = s.length() - 1; i < j; i++, j--) {
```

```
            if (s.charAt(i) != s.charAt(j)) {
                return isPalindrome(s, i, j - 1) || isPalindrome(s, i + 1, j);
            }
        }
        return true;
    }

    private boolean isPalindrome(String s, int i, int j) {
        while (i < j) {
            if (s.charAt(i++) != s.charAt(j--)) {
                return false;
            }
        }
        return true;
    }
}
```

Question 5 合并两个有序数组

```
/**
 * 给定两个有序整数数组 nums1 和 nums2，将 nums2 合并到 nums1 中，使得 num1 成为一个有
数组。
 * 初始化 nums1 和 nums2 的元素数量分别为 m 和 n。
 * 你可以假设 nums1 有足够的空间（空间大小大于或等于 m + n）来保存 nums2 中的元素。
 */
public class Question5 {
    public  void merge(int[] nums1, int m, int[] nums2, int n) {
        int[] result = new int[m + n];
        int i = 0;
        int j = 0;
        int index = 0;

        while (i <= m  - 1 && j <= n - 1) {
            if (nums1[i] <= nums2[j]) {
                result[index++] = nums1[i++];
            } else {
                result[index++] = nums2[j++];
            }
        }
//      吧剩下的直接放进去
        if (i == m) {
//          nums2还有没放完
            for (; j < n; j++) {
                result[index++] = nums2[j];
            }
        } else {
//          nums1还没有放完
            for (; i < m; i++) {
                result[index++] = nums1[i];
            }
        }
//      放入元素
        for (int x = 0;x<result.length;x++){
            nums1[x] = result[x];
```

```
        }
    }
}
```

Question 6 环形链表

```
/**
 * 给定一个链表，判断链表中是否有环。
 * <p>
 * 为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始
。
 * 如果 pos 是 -1，则在该链表中没有环。
 *
 * 做法:想象一下，两名运动员以不同的速度在环形赛道上跑步会发生什么?
 * 快慢指针
 * 使用双指针，一个指针每次移动一个节点，一个指针每次移动两个节点，如果存在环，那么这两个
针一定会相遇。
 */
public class Question6 {

    public boolean hasCycle(ListNode head) {
        if(head==null||head.next ==null){
            return false;
        }
//        定义两个快慢指针
        ListNode slow = head;
        ListNode fast = head.next;

        while(slow!=fast){
            if(fast ==null || fast.next ==null){
                return false;
            }
            slow = slow.next;
            fast = fast.next.next;
        }
        return true;
    }
}

class ListNode {
    int val;
    ListNode next;

    ListNode(int x) {
        val = x;
        next = null;
    }
}
```

Question 7 通过删除字母匹配到字典里最长单词

```
/**
 * 给定一个字符串和一个字符串字典，找到字典里面最长的字符串，该字符串可以通过删除给定字符
```

```
的某些字符来得到。
 * 如果答案不止一个，返回长度最长且字典顺序最小的字符串。如果答案不存在，则返回空字符串。
 * <p>
 * 通过删除字符串 s 中的一个字符能得到字符串 t，可以认为 t 是 s 的子序列，
 * 我们可以使用双指针来判断一个字符串是否为另一个字符串的子序列。
 * 输入
 * s = "abpcplea", d = ["ale","apple","monkey","plea"]
 * <p>
 * 输出:
 * "apple"
 */
public class Question7 {
    public String findLongestWord(String s, List<String> d) {
        String longestWord = "";
//        循环字典
        for (String target : d) {
            boolean bool = isExistWord(target, s);
            if (bool) {
//                说明可以组成
//                判断长度
                if (longestWord.length() < target.length()) {
//                    判断字典里的字母 顺序最小的为结果
                    longestWord = target;
                } else if (longestWord.length() == target.length()) {
                    longestWord = findTheTarget(longestWord, target);
                }
            }
        }
        return longestWord;
    }

    private boolean isExistWord(String target, String s) {
        int i = 0;
        int j = 0;
        while (j < s.length()) {
//            找到一个匹配的 i++  j无论如何都+
            if (target.charAt(i) == s.charAt(j)) {
                i++;
//                如果提前找到了 就放回
                if (i == target.length()) {
                    return true;
                }
            }
            j++;
        }
        return i == target.length() ? true : false;
    }

    private String findTheTarget(String longestWord, String target) {
        for (int i = 0; i < longestWord.length(); i++) {
//      判断
            if (target.charAt(i) < longestWord.charAt(i)) {
                return target;
            }else if(target.charAt(i) == longestWord.charAt(i)){
```

```
            continue;
        }else {
            return longestWord;
        }
    }
    return longestWord;
}

//    public static void main(String[] args) {
//        String s ="wordgoodgoodgoodbestword";
//        List<String> d = new ArrayList<>();
//        d.add("word");
//        d.add("good");
//        d.add("best");
//        d.add("good");
//        String longestWord = findLongestWord(s, d);
//        System.out.println(longestWord);
//    }
}
```

END

2019年8月14日15:20:06