链滴

算法学习之路 | 贪心思想 02

作者：qq692310342
原文链接：https://ld246.com/article/1566007981288
来源网站：链滴

Question7 种花问题

```
/**
 * 假设你有一个很长的花坛，一部分地块种植了花，另一部分却没有。
 * 可是，花卉不能种植在相邻的地块上，它们会争夺水源，两者都会死去。
 * <p>
 * 给定一个花坛（表示为一个数组包含0和1，其中0表示没种植花，1表示种植了花），
 * 和一个数 n 。能否在不打破种植规则的情况下种入 n 朵花？能则返回True，不能则返回False。
 * <p>
 * 思路:
 * 先把第一个和最后一个给种上
 * 然后再循环找能种上的花盆，种上之后改成1
 */
public class Question7 {
    public boolean canPlaceFlowers(int[] flowerbed, int n) {
//        n=0判断
        if (n == 0) {
            return true;
        }
//        判断花盆数量只有一个的话
        if(flowerbed.length==1){
            if(flowerbed[0]==0&&n==1){
                return true;
            }
            return false;
        }
//        先排除掉第一和最后一盆
        if(flowerbed[0]==0&&flowerbed[1]==0){
            flowerbed[0]=1;
            n--;
        }
```

```
        if (flowerbed[flowerbed.length - 1] == 0 && flowerbed[flowerbed.length-2]==0) {
            flowerbed[flowerbed.length-1]=1;
            n--;
        }
//        遍历花盆
        for (int i = 1; i < flowerbed.length - 1; i++) {
            if (flowerbed[i] == 1) {
                continue;
            } else if (flowerbed[i - 1] == 0 && flowerbed[i + 1] == 0) {
                n--;
                flowerbed[i] = 1;
            }
        }
        return n <= 0 ? true : false;
    }
}
```

Question 8 判断子序列

```
/**
 * 给定字符串 s 和 t ，判断 s 是否为 t 的子序列
 *
 * 你可以认为 s 和 t 中仅包含英文小写字母。字符串 t 可能会很长（长度 ~= 500,000），而 s 是个字符串（长度 <=100）。
 * 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新字符串。
 * （例如，"ace"是"abcde"的一个子序列，而"aec"不是）。
 *
 * 解题思路:
 * 直接遍历目标数组，应用一个指针size
 *
 */
public class Question8 {
    public boolean isSubsequence(String s, String t) {
        int size = 0;
        for(int i = 0;i<t.length()&&size<s.length();i++){
            if(s.charAt(size)==t.charAt(i)){
//                一样了，size++
                size++;
            }
        }
        return s.length()==size?true:false;
    }
}
```

Question 9 非递减数列

```
/**
 * 给定一个长度为 n 的整数数组，你的任务是判断在最多改变 1 个元素的情况下，该数组能否变成个非递减数列。
 * 我们是这样定义一个非递减数列的： 对于数组中所有的 i (1 <= i < n)，满足 array[i] <= array[i + ]。
 * <p>
```

```
 * 解题思路:
 * 见注释
 */
public class Question9 {
    public boolean checkPossibility(int[] nums) {
//        非空判断
        if (nums.length == 0 || nums == null || nums.length == 1) {
            return true;
        }
//        定义删除次数
        int index = 0;
//        第一个数特殊处理
        if (nums[0] > nums[1]) {
            index++;
            nums[0] = nums[1];
        }
        for (int i = 1; i < nums.length - 1; i++) {
            if (nums[i] > nums[i + 1]) {
//                找到非递增的数
                index++;
//                修改数字 最大能承受的修改的数字
//                注意这里需要进行贪心判断
                if (nums[i - 1] <= nums[i + 1]) {
//                    此时上一个比下一个还小，那么就没必要直接改下一个了，直接修改nums[i]
                    nums[i] = nums[i - 1];
                } else {
//                    上一个比下一个要大，说明要实现递增就只能够修改nums[i+1]了
                    nums[i + 1] = nums[i];
                }
            }
        }
        return index <= 1 ? true : false;
    }
}
```

Question 10 最大子序和

```
/**
 * 给定一个整数数组 nums ， 找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回最大和。
 * 例如:
 * 输入: [-2,1,-3,4,-1,2,1,-5,4],
 * 输出: 6
 * 解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。
 * 运用到动态规划的小知识
 */
public class Question10 {
    public int maxSubArray(int[] nums) {
//        非空判断
        if (nums.length == 0 || nums == null) {
            return 0;
        }
//        定义初始化最大值
        int max = nums[0];
```

```
//        初始化sum
        int sum = 0;
        for (int num : nums) {
            if (sum > 0) {
//                有所增益
                sum += num;
            } else {
//                没有增益 舍弃前者
                sum = num;
            }
//            前一个最大值已经存储好了 ， 与新的最大值进行比对
            max = Math.max(sum, max);
        }
        return max;
    }
}
```

Question 11 划分字母区间

```
/**
 * 字符串 S 由小写字母组成。
 * 我们要把这个字符串划分为尽可能多的片段，同一个字母只会出现在其中的一个片段。
 * 返回一个表示每个字符串片段的长度的列表。
 */
public class Question11 {

    public List<Integer> partitionLabels(String S) {
//        非空判断
        if (S == null || S.length() == 0) {
            return new ArrayList<Integer>();
        }
//        统计每个字母最后出现的位置
        int[] last = new int[26];
        for (int i = 0; i < S.length(); i++) {
            last[S.charAt(i) - 'a'] = i;
        }
//        统计结果
        List<Integer> list = new ArrayList<>();
//        记录当前最大的目标字母出现的最后位置
        int maxIndex = 0;
//        记录当前目标字母出现的最后位置，初始化不能为0，防止第一个字母为a
        int index = -1;
//        记录最后i的值
        int lastI = -1;
//        遍历S
        for (int i = 0; i < S.length(); i++) {
//            获取当前字母的最后出现位置
            index = last[S.charAt(i) - 'a'];

//            判断是否是比maxIndex还要大
            if (index > maxIndex) {
//                修改maxIndex 继续找
                maxIndex = index;
            } else if (index < maxIndex) {
```

```
//              直接继续，说明该字母在范围内
                continue;
            } else if (index == maxIndex&& maxIndex==i) {
//                找到一个范围了，记录 初始化数据
                list.add(maxIndex-lastI);
//                初始化
                lastI = maxIndex;
//                maxIndex不能初始化为0，需要初始化为max++ 防止出现最后只剩下一个字母的情况
                maxIndex++;
                index = -1;
            }
        }
        return list;
    }
}
```