编程题_README

编程题 1

假设有一个很长的花坛，一部分地块种植了花，另一部分却没有。可是，花不能种植在相邻的地块上，它们会争夺水源，最终都会死去。

给你一个整数数组 flowerbed 表示花坛，由若干 0 和 1 组成，其中 0 表示没有种植花，1 表示种植了花。另外有一个数 n，能否在不打破种植规则的情况下种入 n 朵花？如果能则返回 true，不能则返回 false 。

1.我的思路

本题需要我们判断在给定的花坛下能否种植好给定数目的花朵，所以我们可以将该花坛还可以种植的花朵数量与需要种植的花朵数量进行比较即可。运用贪心算法，对每一个位置都进行判断，只要是可以种植花朵就尽量种植，这样就可以算出可以再种植的最大花朵数量。

对每一个位置都进行判断，可以种植的条件是该位置为空(0)而且两边都没有花朵(1)；主要还是一些边界条件的判断，即如果是第 0 个位置只要保证该位置的右边没有花朵即可；如果是最后一个位置只要保证该位置的左边没有花朵即可。这个条件我们可以使用一下判断条件：if ((i == 0 || flowerbed[i - 1] == 0) && (i == flowerbed.size() - 1 || flowerbed[i + 1] == 0))。最后当这个位置种植好花朵之后，记得将这个位置的标志置为 1 即可。

2.主要函数说明

```
bool canPlaceFlowers(vector<int>& flowerbed, int n)
{
    int sumFlowers = 0;
    for (unsigned int i = 0; i < flowerbed.size(); i++)
    {
        if (flowerbed[i] == 0)
        {
            if ((i == 0 || flowerbed[i - 1] == 0) && (i == flowerbed.size() - 1 || flowerbed[i + 1] == 0))
            {
                sumFlowers++;
                flowerbed[i] = 1;
            }
        }
    }
    return (sumFlowers >= n);
}
```

canPlaceFlowers 函数的功能是通过遍历计算来判断是否在花坛中可以种植 n 朵花。输入参数为花坛数组 flowerbed 和要种植的花数 n，输出的参数是最后的判断结果 true/false。通过遍历，首先判断给位置是否为空，如果为空，才有种植花朵的可能性；如何再判断该位置是否满足种植花朵的条件，即 if ((i == 0 || flowerbed[i - 1] == 0) && (i == flowerbed.size() - 1 || flowerbed[i + 1] == 0))，该判断也将第一个位置和最后一个位置特殊情况考虑进去。最后将所求的可以种植的最大花朵数量和需要种植的花朵数量进行比较，最后得出结果。

3.三个测试用例

用例 1：

输入：flowerbed = [1,0,0,0,1] , n = 1

输出：true

用例 2：

输入：flowerbed = [1,0,0,0,1] , n = 2

输出：false

用例 3：

输入：flowerbed = [1,0,0,0,1,0,0,0,0,1] , n = 2

输出：true

编程题 2

你有两个有序且数组内元素互不相同的数组 nums1 和 nums2。

一条合法路径定义如下：

选择数组 nums1 或者 nums2 开始遍历(从下标 0 出开始)。从左到右遍历当前数组。如果你遇到了 nums1 和 nums2 中都存在的值，那么你可以切换路径到另一个数组对应数字处继续遍历(但在合法路径中重复数字只会被统计一次)。得分定义为合法路径中不同数字的和。

请你返回所有可能合法路径中的最大得分。由于答案可能很大，请你将它对 10^9 +7 取余后返回。

1.我的思路

本题要求解路径的最大得分，由于有两个数组，使用我们将以 nums1 为终点的得分为 sum1，以 nums2 为终点的得分为 sum2；我们需要判断从哪一个为起点的最后得分高，使用我们要计算从每一个起点开始各自的最高得分；此时我们使用两个指针，一个为 i，另一个为 j，前者指向 nums1 的下标，后者指向 nums2 的下标；再不断遍历的过程中(双下标各自不超过各自的数组长度)，如果两个下标所指向各自数组的值不相等，则哪一个指针向右移，同时该数组的计数总和加上前面的部分；如果两个下标所指向各自数组的值相等，此时两个指针都要右移动，这时还要注意，更新 sum1 和 sum2 和当前的两者最大值！这一点非常重要！因为在两个数字相同时可以跑到另一条路径上！相当于前面的得分可以一样(取最大的)！当退出循环后，如果两个数组的长度不同，肯定会存在另一条路径后面还有剩余的情况，此时将后面的部分再加上即可；最后比较两种情况，取最高得分并且对 10^9 +7 取余后返回，即为最后的结果。

2.主要函数说明

```
int maxSum(vector<int>& nums1, vector<int>& nums2)
{
    int i = 0;
    int j = 0;
    long sum1 = 0;
    long sum2 = 0;
    int length1 = nums1.size();
    int length2 = nums2.size();
    while (i < length1 && j < length2)
    {
        if (nums1[i] < nums2[j])
        {
            sum1 += nums1[i];
            i++;
        }
        else if (nums1[i] > nums2[j])
        {
            sum2 += nums2[j];
            j++;
```

```
            }
            else
            {
                sum1 += nums1[i];
                sum2 += nums2[j];
                long tempMax = max(sum1, sum2);
                sum1 = sum2 = tempMax;
                i++;
                j++;
            }
        }
        while (i < length1)
            sum1 += nums1[i++];
        while (j < length2)
            sum2 += nums2[j++];
        return (max(sum1, sum2) % 1000000007);
    }
```

maxSum 函数的功能是通过双指针计算比较得出最大得分数，它的输入参数为两个数组 nums1 和 nums2，返回最后计算的最大得分数。两个指针开始分别指向两个数组的起点，如果当前两数组位置大小不同，则通过前几项相加，保存当前路径的值；如果当前两数组位置大小相同，则更新两个计数的值为两者最大的值加上当前节点的值；循环过后，将剩下的部分相加。最后比较两个结果的大小，取其最大值并且对 10^9 +7 取余后，得出最大得分。

3.三个测试用例

用例 1：

输入：nums1 = [2,4,5,8,10] , nums2 = [4,6,8,9]

输出：30

用例 2：

输入：nums1 = [1,3,5,7,9] , nums2 = [3,5,100]

输出：109

用例 3：

输入：nums1 = [1,2,3,4,5] , nums2 = [6,7,8,9,10]

输出：40