本文主要解析LeetCode在2023年7月8日22:30组织的第【108】场双周赛题目，竞赛链接。

有兴趣的同学可以关注一下我的LeetCode个人账号。

题目一、最长交替子序列【简单】

题目描述

给你一个下标从 **0** 开始的整数数组 `nums` 。如果 `nums` 中长度为 `m` 的子数组 `s` 满足以下条件，我们称它是一个交替子序列：

- `m` 大于 `1` 。
- `s1 = s0 + 1` 。
- 下标从 0 开始的子数组 `s` 与数组 `[s0, s1, s0, s1,...,s(m-1) % 2]` 一样。也就是说，`s1 - s0 = 1` ，`s2 - s1 = -1` ，`s3 - s2 = 1` ，`s4 - s3 = -1` ，以此类推，直到 `s[m - 1] - s[m - 2] = (-1)^m` 。

请你返回 `nums` 中所有 **交替** 子数组中，最长的长度，如果不存在交替子数组，请你返回 `-1` 。

子数组是一个数组中一段连续 **非空** 的元素序列。

示例 1：

```
输入：nums = [2,3,4,3,4]
输出：4
解释：交替子数组有 [3,4] ，[3,4,3] 和 [3,4,3,4] 。最长的子数组为 [3,4,3,4] ，长度为4 。
```

示例 2：

```
输入：nums = [4,5,6]
输出：2
解释：[4,5] 和 [5,6] 是仅有的两个交替子数组。它们长度都为 2 。
```

提示：

- `2 <= nums.length <= 100`
- `1 <= nums[i] <= 10^4`

分析

简单题，按题意模拟即可

代码

```python
class Solution:
    def alternatingSubarray(self, nums: List[int]) -> int:
        res = -1
        n = len(nums)

        for i in range(n):
            diff = 1
            count = 1
            for j in range(i+1, n):
                if nums[j] - nums[j-1] == diff:
                    count += 1
                    res = max(res, count)
                    diff *= -1
                else:
                    break
        return res
```

题目二、重新放置石块【中等】

题目描述

给你一个下标从 **0** 开始的整数数组 `nums` ，表示一些石块的初始位置。再给你两个长度 **相等** 下标从 **0** 开始的整数数组 `moveFrom` 和 `moveTo` 。

在 `moveFrom.length` 次操作内，你可以改变石块的位置。在第 `i` 次操作中，你将位置在 `moveFrom[i]` 的所有石块移到位置 `moveTo[i]` 。

完成这些操作后，请你按升序返回所有 **有** 石块的位置。

注意：

- 如果一个位置至少有一个石块，我们称这个位置 **有** 石块。
- 一个位置可能会有多个石块。

示例 1：

```
输入：nums = [1,6,7,8], moveFrom = [1,7,2], moveTo = [2,9,5]
输出：[5,6,8,9]
解释：一开始，石块在位置 1,6,7,8 。
第 i = 0 步操作中，我们将位置 1 处的石块移到位置 2 处，位置 2,6,7,8 有石块。
第 i = 1 步操作中，我们将位置 7 处的石块移到位置 9 处，位置 2,6,8,9 有石块。
第 i = 2 步操作中，我们将位置 2 处的石块移到位置 5 处，位置 5,6,8,9 有石块。
最后，至少有一个石块的位置为 [5,6,8,9] 。
```

示例 2：

```
输入：nums = [1,1,3,3], moveFrom = [1,3], moveTo = [2,2]
输出：[2]
解释：一开始，石块在位置 [1,1,3,3] 。
第 i = 0 步操作中，我们将位置 1 处的石块移到位置 2 处，有石块的位置为 [2,2,3,3] 。
第 i = 1 步操作中，我们将位置 3 处的石块移到位置 2 处，有石块的位置为 [2,2,2,2] 。
由于 2 是唯一有石块的位置，我们返回 [2] 。
```

提示：

- `1 <= nums.length <= 10^5`
- `1 <= moveFrom.length <= 10^5`
- `moveFrom.length == moveTo.length`
- `1 <= nums[i], moveFrom[i], moveTo[i] <= 10^9`
- 测试数据保证在进行第 `i` 步操作时，`moveFrom[i]` 处至少有一个石块。

分析

使用集合保存所有的石块位置，遍历moveFrom和moveTo，从集合中删除moveFrom[i],增加moveTo[i]，最后转换成有序数组返回

代码

```python
class Solution:
    def relocateMarbles(self, nums: List[int], moveFrom: List[int], moveTo: List[int])
-> List[int]:
        s = set(nums)

        n = len(moveFrom)
        for i in range(n):
            f,t = moveFrom[i], moveTo[i]
            s.remove(f)
            s.add(t)
        res = list(s)
        return sorted(res)
```

题目三、[将字符串分割为最少的美丽子字符串](#)【中等】

题目描述

给你一个二进制字符串 `s` ，你需要将字符串分割成一个或者多个 **子字符串** ，使每个子字符串都是 **美丽** 的。

如果一个字符串满足以下条件，我们称它是 **美丽** 的：

- 它不包含前导 0 。
- 它是 `5` 的幂的 **二进制** 表示。

请你返回分割后的子字符串的 **最少** 数目。如果无法将字符串 `s` 分割成美丽子字符串，请你返回 `-1` 。

子字符串是一个字符串中一段连续的字符序列。

示例 1：

```
输入：s = "1011"
输出：2
解释：我们可以将输入字符串分成 ["101", "1"] 。
- 字符串 "101" 不包含前导 0 ，且它是整数 51 = 5 的二进制表示。
- 字符串 "1" 不包含前导 0 ，且它是整数 50 = 1 的二进制表示。
最少可以将 s 分成 2 个美丽子字符串。
```

示例 2：

```
输入：s = "111"
输出：3
解释：我们可以将输入字符串分成 ["1", "1", "1"] 。
- 字符串 "1" 不包含前导 0 ，且它是整数 50 = 1 的二进制表示。
最少可以将 s 分成 3 个美丽子字符串。
```

示例 3：

```
输入：s = "0"
输出：-1
解释：无法将给定字符串分成任何美丽子字符串。
```

提示：

- `1 <= s.length <= 15`
- `s[i]` 要么是 `'0'` 要么是 `'1'` 。

分析

记忆化递归

定义dfs(index)，表示从index位置开始分割美丽子字符串得到的最少分割次数

递归起点：index = 0

递归结束：1.包含前导0， 2.index已经到字符串末尾

`5` 的幂的 **二进制**的处理：

先把在数据范围内的5的幂先加入到一个集合中，再将子字符串转换成数字，然后判断数字是否在集合中，以此来判断子字符串是否是5的幂的二进制

代码

```
val_set = set()
val = 1
while(val <= 2 ** 15):
    val_set.add(val)
    val *= 5


class Solution:
    def minimumBeautifulSubstrings(self, s: str) -> int:
        n = len(s)

        @cache
        def dfs(i):
            if i == n:
                return 0

            if s[i] == '0':
                return inf

            res = inf
            for j in range(i, n):
                binary_str = s[i:j + 1]
                num = int(binary_str, 2)
                if num not in val_set:
                    continue
                res = min(res, 1 + dfs(j + 1))

            return res

        return dfs(0) if dfs(0) != inf else -1
```

题目四、黑格子的数目【中等】

题目描述

给你两个整数 `m` 和 `n` ，表示一个下标从 **0** 开始的 `m x n` 的网格图。

给你一个下标从 **0** 开始的二维整数矩阵 `coordinates` ，其中 `coordinates[i] = [x, y]` 表示坐标为 `[x, y]` 的格子是 **黑色的** ，所有没出现在 `coordinates` 中的格子都是 **白色的**。

一个块定义为网格图中 `2 x 2` 的一个子矩阵。更正式的，对于左上角格子为 `[x, y]` 的块，其中 `0 <= x < m - 1` 且 `0 <= y < n - 1` ，包含坐标为 `[x, y]` ， `[x + 1, y]` ， `[x, y + 1]` 和 `[x + 1, y + 1]` 的格子。

请你返回一个下标从 **0** 开始长度为 `5` 的整数数组 `arr` ， `arr[i]` 表示恰好包含 `i` 个 **黑色** 格子的块的数目。

示例 1：

```
输入：m = 3, n = 3, coordinates = [[0,0]]
输出：[3,1,0,0,0]
解释：网格图如下：

只有 1 个块有一个黑色格子，这个块是左上角为 [0,0] 的块。
其他 3 个左上角分别为 [0,1] ，[1,0] 和 [1,1] 的块都有 0 个黑格子。
所以我们返回 [3,1,0,0,0] 。
```

示例 2：

```
输入：m = 3, n = 3, coordinates = [[0,0],[1,1],[0,2]]
输出：[0,2,2,0,0]
解释：网格图如下：

有 2 个块有 2 个黑色格子（左上角格子分别为 [0,0] 和 [0,1]）。
左上角为 [1,0] 和 [1,1] 的两个块，都有 1 个黑格子。
所以我们返回 [0,2,2,0,0] 。
```

提示：

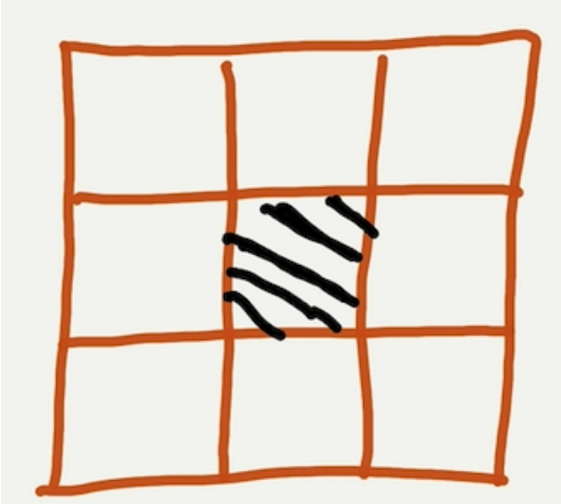
- `2 <= m <= 10^5`
- `2 <= n <= 10^5`
- `0 <= coordinates.length <= 10^4`
- `coordinates[i].length == 2`
- `0 <= coordinates[i][0] < m`
- `0 <= coordinates[i][1] < n`

- `coordinates` 中的坐标对两两互不相同。

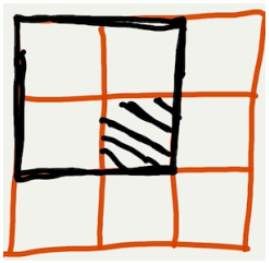
分析

考虑一个黑色格子对结果的影响：

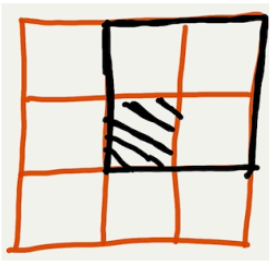
假设中间位置的格子是黑色的：



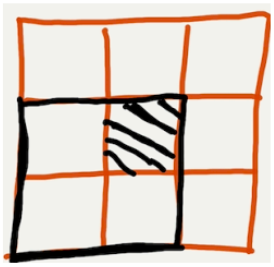
那么对2x2矩阵结果有影响的有下面四种情况：



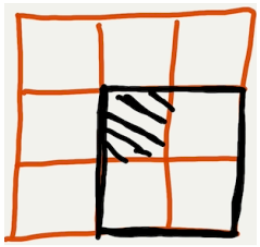
1.左上角为(x-1,y-1)的矩阵



2.左上角为(x-1,y)的矩阵



3.左上角为(x,y-1)的矩阵



4.左上角为(x,y)的矩阵

在代码处理的时候，先将所有的四个2x2矩阵的左上角位置加入键值对中，然后数据处理时再判断该位置能不能作为2x2矩阵的左上角，坐标满足2x2矩阵左上角的条件是：x >= 0 and y >= 0 and x + 1 < m and y + 1 < n

代码

```
class Solution:
    def countBlackBlocks(self, m: int, n: int, coordinates: List[List[int]]) ->
List[int]:
        map = dict()
        for arr in coordinates:
            x,y = arr[0],arr[1]
            key_list = [str(x-1) + ":" + str(y-1), str(x-1) + ":" + str(y), str(x) +
":" + str(y-1), str(x) + ":" + str(y)]
            for key in key_list:
                if key in map:
                    map[key] = map[key] + 1
```

```
                else:
                    map[key] = 1

        res = [(m-1) * (n-1), 0, 0, 0, 0]
        for k,v in map.items():
            nums = k.split(":")
            x, y = int(nums[0]), int(nums[1])
            if x >= 0 and y >= 0 and x + 1 < m and y + 1 < n:
                res[0] -= 1
                res[v] += 1
        return res
```