0718. 最长重复子数组

ITCharge 大约 4 分钟

- 标签：数组、二分查找、动态规划、滑动窗口、哈希函数、滚动哈希
- 难度：中等

题目链接

- 0718. 最长重复子数组 - 力扣

题目大意

描述：给定两个整数数组 $nums1$、$nums2$。

要求：计算两个数组中公共的、长度最长的子数组长度。

说明：

- $1 \leq nums1.length, nums2.lengt$ 1000。
- $0 \leq nums1[i], nums2[i] \leq 100$。

示例：

- 示例 1：

```py
输入：nums1 = [1,2,3,2,1], nums2 = [3,2,1,4,7]
输出：3
解释：长度最长的公共子数组是 [3,2,1] 。
```

- 示例 2：

```py
输入：nums1 = [0,0,0,0,0], nums2 = [0,0,0,0,0]
输出：5
```

解题思路

思路 1：暴力（超时）

1. 枚举数组 $nums1$ 和 $nums2$ 的子数组开始位置 i、j。
2. 如果遇到相同项，即 $nums1[i] == nums2[j]$，则以 $nums1[i]$、$nums2[j]$ 为前缀，同时向后遍历，计算当前的公共子数组长度 $subLen$ 最长为多少。
3. 直到遇到超出数组范围或者 $nums1[i + subLen] == nums2[j + subLen]$ 情况时，停止遍历，并更新答案。
4. 继续执行 $1 \sim 3$ 步，直到遍历完，输出答案。

思路 1：代码

```py
class Solution:
    def findLength(self, nums1: List[int], nums2: List[int]) -> int:
        size1, size2 = len(nums1), len(nums2)
        ans = 0
        for i in range(size1):
            for j in range(size2):
                if nums1[i] == nums2[j]:
                    subLen = 1
                    while i + subLen < size1 and j + subLen < size2 and nums1[i
+ subLen] == nums2[j + subLen]:
                        subLen += 1
                    ans = max(ans, subLen)

        return ans
```

思路 1：复杂度分析

- **时间复杂度**：$O(n \times m \times min(n, m))$。其中 n 是数组 $nums1$ 的长度，m 是数组 $nums2$ 的长度。
- **空间复杂度**：$O(1)$。

思路 2：滑动窗口

暴力方法中，因为子数组在两个数组中的位置不同，所以会导致子数组之间会进行多次比较。

我们可以将两个数组分别看做是两把直尺。然后将数组 $nums1$ 固定， 让 $nums2$ 的尾部与 $nums1$ 的头部对齐，如下所示。

```py
nums1 =             [1, 2, 3, 2, 1]
nums2 = [3, 2, 1, 4, 7]
```

然后逐渐向右移动直尺 $nums2$，比较 $nums1$ 与 $nums2$ 重叠部分中的公共子数组的长度，直到直尺 $nums2$ 的头部移动到 $nums1$ 的尾部。

```py
nums1 =             [1, 2, 3, 2, 1]
nums2 =    [3, 2, 1, 4, 7]

nums1 =             [1, 2, 3, 2, 1]
nums2 =       [3, 2, 1, 4, 7]

nums1 =             [1, 2, 3, 2, 1]
nums2 =          [3, 2, 1, 4, 7]

nums1 =             [1, 2, 3, 2, 1]
nums2 =             [3, 2, 1, 4, 7]

nums1 =             [1, 2, 3, 2, 1]
nums2 =                [3, 2, 1, 4, 7]

nums1 =             [1, 2, 3, 2, 1]
nums2 =                   [3, 2, 1, 4, 7]

nums1 =             [1, 2, 3, 2, 1]
nums2 =                      [3, 2, 1, 4, 7]

nums1 =             [1, 2, 3, 2, 1]
nums2 =                         [3, 2, 1, 4, 7]
```

在这个过程中求得的 $nums1$ 与 $nums2$ 重叠部分中的最大的公共子数组的长度就是 $nums1$ 与 $nums2$ 数组中公共的、长度最长的子数组长度。

思路 2：代码

```py
class Solution:
    def findMaxLength(self, nums1, nums2, i, j):
        size1, size2 = len(nums1), len(nums2)
        max_len = 0
        cur_len = 0
        while i < size1 and j < size2:
            if nums1[i] == nums2[j]:
                cur_len += 1
                max_len = max(max_len, cur_len)
            else:
                cur_len = 0
            i += 1
            j += 1
        return max_len

    def findLength(self, nums1: List[int], nums2: List[int]) -> int:
        size1, size2 = len(nums1    en(nums2)
        res = 0
        for i in range(size1):
            res = max(res, self.findMaxLength(nums1, nums2, i, 0))

        for i in range(size2):
            res = max(res, self.findMaxLength(nums1, nums2, 0, i))

        return res
```

思路 2：复杂度分析

- **时间复杂度**：$O(n + m) \times min(n, m)$。其中 n 是数组 $nums1$ 的长度，m 是数组 $nums2$ 的长度。
- **空间复杂度**：$O(1)$。

思路 3：动态规划

1. 划分阶段

按照子数组结尾位置进行阶段划分。

2. 定义状态

定义状态 $dp[i][j]$ 为：「以 $nums1$ 中前 i 个元素为子数组（$nums1[0]...nums2[i-1]$）」和「以 $nums2$ 中前 j 个元素为子数组（$nums2[0]...nums2[j-1]$）」的最长公共子数组长度。

3. 状态转移方程

1. 如果 $nums1[i-1] = nums2[j-1]$，则当前元素可以构成公共子数组，此时 $dp[i][j] = dp[i-1][j-1] + 1$。
2. 如果 $nums1[i-1] \neq nums2[j-1]$，则当前元素不能构成公共子数组，此时 $dp[i][j] = 0$。

4. 初始条件

- 当 $i = 0$ 时，$nums1[0]...nums1[i-1]$ 表示的是空数组，空数组与 $nums2[0]...nums2[j-1]$ 的最长公共子序列长度为 0，即 $dp[0][j] = 0$。
- 当 $j = 0$ 时，$nums2[0]...nums2[j-1]$ 表示的是空数组，空数组与 $nums1[0]...nums1[i-1]$ 的最长公共子序列长度为 0，即 $dp[i][0] = 0$。

5. 最终结果

- 根据状态定义， $dp[i][j]$ 为：「以 $nums1$ 中前 i 个元素为子数组（$nums1[0]...nums2[i-1]$）」和「以 $nums2$ 中前 j 个元素为子数组（$nums2[0]...nums2[j-1]$）」的最长公共子数组长度。在遍历过程中，我们可以使用 res 记录下所有 $dp[i][j]$ 中最大值即为答案。

思路 3：代码

```py
class Solution:
    def findLength(self, nums1: List[int], nums2: List[int]) -> int:
        size1 = len(nums1)
        size2 = len(nums2)
        dp = [[0 for _ in range(size2 + 1)] for _ in range(size1 + 1)]
        res = 0
        for i in range(1, size1 + 1):
            for j in range(1, size2 + 1):
                if nums1[i - 1] == nums2[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1] + 1
                if dp[i][j] > res:
                    res = dp[i][j]

        return res
```

思路 3：复杂度分析

- **时间复杂度**：$O(n \times m)$。其中 n 是数组 $nums1$ 的长度，m 是数组 $nums2$ 的长度。
- **空间复杂度**：$O(n \times m)$。