# 题目

给两个整数数组 A 和 B ，返回两个数组中公共的、长度最长的子数组的长度。

**示例：**

输入：

A: [1,2,3,2,1]

B: [3,2,1,4,7]

输出：3

解释：

长度最长的公共子数组是[3, 2, 1] 。

**提示：**

1 <= len(A), len(B) <= 1000

0 <= A[i], B[i] < 100

注：对比最长回文子串（都可以采用动态规划的算法）

# 分析

## 方法一：动态规划

要解决“寻找两个数组中公共且长度最长的子数组”问题，核心思路是动态规划（DP）。利用子数组“连续”的特性，通过定义DP状态记录以两个数组特定位置结尾的最长公共子数组长度，逐步推导出全局最大值。

解题思路

1、DP状态定义：

定义 dp[i][j] 表示以 nums1[i-1]（nums1第i个元素）和 nums2[j-1]（nums2第j个元素）为结尾的最长公共子数组的长度。

（注：索引从1开始定义，是为了避免处理i=0或j=0时的边界判断，此时dp[0][j]和dp[i][0]均为0，代表“空数组与任意子数组的公共长度为0”）。

2、状态转移方程：

- 若nums1[i-1] == nums2[j-1]：当前元素相同，说明公共子数组可延续，dp[i][j] = dp[i-1][j-1] + 1（前一个位置的公共长度+1）。

- 若nums1[i-1] != nums2[j-1]：当前元素不同，以这两个元素结尾的公共子数组断裂，dp[i][j] = 0。

3、初始化与结果：

- 初始时，dp[0][j] = 0（i=0，nums1为空）、dp[i][0] = 0（j=0，nums2为空）。

- 遍历过程中，实时记录dp[i][j]的最大值，即为最终的“最长公共子数组长度”。

代码：

class Solution {

public:

int findLength(vector<int>& nums1, vector<int>& nums2) {

int m = nums1.size();

int n = nums2.size();

// 优化空间：用一维数组替代二维数组（因dp[i][j]仅依赖dp[i-1][j-1]，可压缩为1D）

// 选择长度较小的数组作为内层循环，进一步减少空间开销

if (m < n) return findLength(nums2, nums1);

vector<int> dp(n + 1, 0); // dp[j] 对应原二维DP中的dp[i][j]

int max\_len = 0; // 记录最长公共子数组长度

// 遍历nums1（外层，长度较长的数组）

for (int i = 1; i <= m; ++i) {

// 遍历nums2（内层，从后往前，避免覆盖dp[j-1]的旧值）

for (int j = n; j >= 1; --j) {

if (nums1[i-1] == nums2[j-1]) {

dp[j] = dp[j-1] + 1; // 等价于原dp[i][j] = dp[i-1][j-1] + 1

max\_len = max(max\_len, dp[j]);

} else {

dp[j] = 0; // 元素不同，公共子数组断裂

}

}

}

return max\_len;

}

};

代码解释

1、空间优化逻辑：

原二维DP数组dp[m+1][n+1]存在冗余——计算 dp[i][j] 仅需 dp[i-1][j-1]（前一行前一列的值），因此可压缩为一维数组 dp[n+1]。

- 内层循环从后往前遍历（j从n到1）：避免更新 dp[j] 时，dp[j-1]已被当前行的计算覆盖（确保使用的是上一行的 dp[j-1]值）。

- 选择较长数组作为外层循环：当 m > n 时，一维数组长度为 n+1，比m+1更小，进一步节省空间。

2、核心计算流程：

以示例 nums1 = [1,2,3,2,1]、nums2 = [3,2,1,4,7] 为例：

- 当 i=3（nums1[2] = 3）、j=1（nums2[0] = 3）时，元素相同，dp[1] = dp[0] + 1 = 1，max\_len=1。

- 当 i=4（nums1[3] = 2）、j=2（nums2[1] = 2）时，元素相同，dp[2] = dp[1] + 1 = 2，max\_len=2。

- 当 i=5（nums1[4] = 1）、j=3（nums2[2] = 1）时，元素相同，dp[3] = dp[2] + 1 = 3，max\_len=3（最终结果）。

复杂度分析

- 时间复杂度：O(m×n)，其中 m、n分别为两个数组的长度。需遍历两个数组的所有位置组合，每个位置的计算为常数级。

- 空间复杂度：O(min(m,n))，优化后仅使用一维数组，长度为较短数组的长度+1。

**或：**

class Solution {

public:

int findLength(vector<int>& A, vector<int>& B) {

int n = A.size(), m = B.size();

vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

int ans = 0;

for (int i = n - 1; i >= 0; i--) {

for (int j = m - 1; j >= 0; j--) {

dp[i][j] = A[i] == B[j] ? dp[i + 1][j + 1] + 1 : 0;

ans = max(ans, dp[i][j]);

}

}

return ans;

}

};

**复杂度分析：**

时间复杂度：O(N×M)。

空间复杂度：O(N×M)。

N表示数组A的长度，M表示数组B的长度。

空间复杂度还可以再优化，利用滚动数组可以优化到O(min(N,M))。

**或（推荐）：**

class Solution {

public:

int findLength(vector<int>& A, vector<int>& B) {

int n = A.size(), m = B.size();

vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

int ret = 0;

for (int i = n - 1; i >= 0; i--) {

for (int j = m - 1; j >= 0; j--) {

if(A[i] == B[j])

dp[i][j] = dp[i + 1][j + 1] + 1;

else

dp[i][j] = 0;

ret = max(ret, dp[i][j]);

}

}

return ret;

}

};

## 方法二：滑动窗口

## 方法三：二分查找+哈希表