# 题目

如果一个数列 至少有三个元素 ，并且任意两个相邻元素之差相同，则称该数列为等差数列。

例如，[1,3,5,7,9]、[7,7,7,7] 和 [3,-1,-5,-9] 都是等差数列。

给你一个整数数组 nums ，返回数组 nums 中所有为等差数组的 子数组 个数。

子数组 是数组中的一个连续序列。

示例 1：

输入：nums = [1,2,3,4]

输出：3

解释：nums 中有三个子等差数组：[1, 2, 3]、[2, 3, 4] 和 [1,2,3,4] 自身。

示例 2：

输入：nums = [1]

输出：0

提示：

1 <= nums.length <= 5000

-1000 <= nums[i] <= 1000

# 分析

## 方法一：动态规划

思路：

为了解决这个问题，我们可以使用动态规划的方法。对于给定的数组nums，我们可以定义一个辅助数组dp，其中dp[i]表示以nums[i]结尾的等差子数组的个数。

我们遍历数组nums，对于每个位置i，我们检查是否存在一个位置j（j < i），使得nums[i] - nums[j]等于nums[j] - nums[j-1]（假设j > 0），即满足等差数列的条件。如果找到这样的位置j，那么以nums[i]结尾的等差子数组个数就等于以nums[j]结尾的等差子数组个数加上1（因为我们可以将nums[i]添加到任何以nums[j]结尾的等差子数组的末尾，形成一个新的等差子数组）。

同时，我们还需要一个变量count来记录总的等差子数组个数。每当我们找到一个满足等差条件的子数组时，我们将dp[i]加到count上。

代码：

class Solution {

public:

int numberOfArithmeticSlices(vector<int>& nums) {

if (nums.size() < 3) {

return 0;

}

int n = nums.size();

vector<int> dp(n, 0); // dp[i]表示以nums[i]结尾的等差子数组个数

int count = 0; // 总的等差子数组个数

for (int i = 2; i < n; i++) {

if (nums[i] - nums[i-1] == nums[i-1] - nums[i-2]) {

dp[i] = dp[i-1] + 1; // 以nums[i]结尾的等差子数组个数等于以nums[i-1]结尾的个数加1

count += dp[i]; // 将dp[i]加到总的等差子数组个数上

}

}

return count;

}

};

注意，在上面的代码中，我们从索引2开始遍历数组，因为等差数列至少需要三个元素。同时，我们只在找到满足等差条件的子数组时才更新dp[i]和count。

这个算法的时间复杂度是O(n)，其中n是数组nums的长度。我们只需要遍历一次数组即可找到所有的等差子数组。空间复杂度也是O(n)，因为我们需要一个大小为n的辅助数组dp来存储以每个位置结尾的等差子数组个数。

## 方法二：计数

思路：

1、初始化:

n 存储数组nums的长度。

如果n为1，则直接返回0，因为长度为1的数组不可能构成等差子数组。

diff 用来存储当前考虑的等差数列的公差。

t 用来存储当前公差下的等差子数组数量（注意，这里的t并不是子数组的长度，而是长度至少为3的等差子数组的数量）。

ans 用来存储总的等差子数组数量。

2、遍历数组:

从索引2开始遍历数组，因为等差数列至少要有3个元素。

对于每个索引i，检查当前元素nums[i]与前一个元素nums[i-1]的差是否等于diff（即是否维持了当前的公差）。

如果维持了公差，则t加1，表示在当前的公差下，我们又找到了一个新的等差子数组。

如果不维持了公差，则更新diff为新的公差，并将t重置为0。

无论是否维持公差，都将t加到ans上，因为t表示的是在当前公差下，以nums[i-1]结尾的长度至少为3的等差子数组的数量。

代码：

class Solution {

public:

int numberOfArithmeticSlices(vector<int>& nums) {

int n = nums.size(); // 获取数组长度

if (n == 1) {

return 0; // 如果数组只有一个元素，无法构成等差子数组

}

int diff = nums[0] - nums[1]; // 初始化公差为前两个元素的差

int t = 0; // 当前公差下的等差子数组数量

int ans = 0; // 总的等差子数组数量

// 从第三个元素开始遍历，因为等差数列至少要有3个元素

for (int i = 2; i < n; ++i) {

// 如果当前元素与前一个元素的差等于当前公差

if (nums[i - 1] - nums[i] == diff) {

++t; // 在当前公差下，等差子数组数量加1

}

else {

// 如果当前元素与前一个元素的差不等于当前公差

// 更新公差为新的差

diff = nums[i - 1] - nums[i];

// 重置t，因为公差变了，之前的t不再适用

t = 0;

}

// 将t加到ans上，因为t表示的是在当前公差下，以nums[i-1]结尾的长度至少为3的等差子数组的数量

ans += t;

}

return ans; // 返回总的等差子数组数量

}

};

这个实现的时间复杂度是O(n)，其中n是数组nums的长度。我们只需要遍历一次数组即可得到答案。这个算法比之前的O(n^3)的算法更高效，因为它利用了一个关键的事实：一旦我们确定了等差数列的公差，就可以通过一次遍历来找到所有使用该公差的等差子数组。