# 题目

给你一个整数数组nums和一个整数k，请你返回数对(i, j)的数目，满足i < j且|nums[i] - nums[j]| == k。

|x|的值定义为：

如果x >= 0，那么值为x。

如果x < 0，那么值为-x。

示例 1：

输入：nums = [1,2,2,1], k = 1

输出：4

解释：差的绝对值为1的数对为：

- [1,2,2,1]

- [1,2,2,1]

- [1,2,2,1]

- [1,2,2,1]

示例 2：

输入：nums = [1,3], k = 3

输出：0

解释：没有任何数对差的绝对值为3。

示例 3：

输入：nums = [3,2,1,5,4], k = 2

输出：3

解释：差的绝对值为2的数对为：

- [3,2,1,5,4]

- [3,2,1,5,4]

- [3,2,1,5,4]

提示：

1 <= nums.length <= 200

1 <= nums[i] <= 100

1 <= k <= 99

# 分析

## 方法一：暴力法

思路：

使用了两层循环来遍历所有可能的数对 (i, j)，其中 i 从数组的第一个元素开始，j 从 i 的下一个元素开始。对于每一对数 (nums[i], nums[j])，它检查差的绝对值是否等于 k，如果是，则增加计数器 count。

代码：

class Solution {

public:

int countKDifference(vector<int>& nums, int k) {

int count = 0;

for (int i = 0; i < nums.size(); ++i) {

for (int j = i + 1; j < nums.size(); ++j) {

if (abs(nums[i] - nums[j]) == k) {

++count;

}

}

}

return count;

}

};

请注意，这个解法的时间复杂度是 O(n^2)，其中 n 是数组 nums 的长度。对于较小的数组（如题目中给出的提示），这个解法是有效的。但如果数组非常大，这个解法可能会变得很慢。对于大型数组，可以考虑使用哈希表或其他更高效的算法来减少时间复杂度。

## 方法二：哈希表

思路：  
 我们进行一次遍历，遍历时下标代表j。对每一个j，我们需要知道在这个j之前的符合条件的i的个数，即满足∣nums[i]−nums[j]∣=k的i的个数，亦即满足nums[i]=nums[j]+k或nums[i]=nums[j]−k的i的个数。使用哈希表可以在O(1)的时间内统计出这样的个数，因此在遍历时我们可以使用一个哈希表来维护不同数值的频率，并统计符合条件的数对总数。

代码：

class Solution {

public:

int countKDifference(vector<int>& nums, int k) {

int res = 0, n = nums.size();

unordered\_map<int, int> cnt;

for (int j = 0; j < n; ++j) {

// 检查是否存在nums[j]-nums[i] == k

int num1 = nums[j] - k;

if (cnt.count(num1) > 0) {

res += cnt[num1];

}

int num2 = nums[j] + k;

// 检查是否存在nums[j]-nums[i] == -k

if (cnt.count(num2) > 0) {

res += cnt[num2];

}

++cnt[nums[j]];

}

return res;

}

};

或：

class Solution {

public:

int countKDifference(vector<int>& nums, int k) {

int res = 0, n = nums.size();

unordered\_map<int, int> cnt;

for (int j = 0; j < n; ++j) {

res += (cnt.count(nums[j] - k) ? cnt[nums[j] - k] : 0);

res += (cnt.count(nums[j] + k) ? cnt[nums[j] + k] : 0);

++cnt[nums[j]];

}

return res;

}

};

复杂度分析

时间复杂度：O(n)，其中n为数组nums的长度。我们仅使用了一次遍历来寻找所有符合条件的数对。

空间复杂度：O(n)。哈希表消耗了O(n)的空间。