# 0220. 存在重复元素 III

ITCharge 大约 5 分钟

- 标签：数组、桶排序、有序集合、排序、滑动窗口
- 难度：中等

## 题目链接

- 0220. 存在重复元素 III - 力扣

## 题目大意

**描述**：给定一个整数数组 $nums$，以及两个整数 $k$、$t$。

**要求**：判断数组中是否存在两个不同下标的 $i$ 和 $j$，其对应元素满足 $abs(nums[i] - nums[j]) \leq t$，同时满足 $abs(i - j) \leq k$。如果满足条件则返回 `True`，不满足条件返回 `False`。

**说明**：

- $0 \leq nums.length \leq 2 \times 10^4$。
- $-2^{31} \leq nums[i] \leq 2^{31} - 1$。
- $0 \leq k \leq 10^4$。
- $0 \leq t \leq 2^{31} - 1$。

**示例**：

- 示例 1：

```py
输入：nums = [1,2,3,1], k = 3, t = 0
输出：True
```

- 示例 2：

```py
输入：nums = [1,0,1,1], k = 1, t = 2
输出：True
```

# 解题思路

题目中需要满足两个要求，一个是元素值的要求（$abs(nums[i] - nums[j]) \leq t$），一个是下标范围的要求（$abs(i - j) \leq k$）。

对于任意一个位置 $i$ 来说，合适的 $j$ 应该在区间 $[i - k, i + k]$ 内，同时 $nums[j]$ 值应该在区间 $[nums[i] - t, nums[i] + t]$ 内。

最简单的做法是两重循环遍历数组，第一重循环遍历位置 $i$，第二重循环遍历 $[i - k, i + k]$ 的元素，判断是否满足 $abs(nums[i] - nums[j]) \leq t$。但是这样做的时间复杂度为 $O(n \times k)$，其中 $n$ 是数组 $nums$ 的长度。

我们需要优化一下检测相邻 $2 \times k$ 个元素是否满足 $abs(nums[i] - nums[j]) \leq t$ 的方法。有两种思路：「桶排序」和「滑动窗口（固定长度）」。

## 思路 1：桶排序

1. 利用桶排序的思想，将桶的大小设置为 $t + 1$。只需要使用一重循环遍历位置 $i$，然后根据 $\lfloor \frac{nums[i]}{t+1} \rfloor$，从而决定将 $nums[i]$ 　　 ，哪个桶中。
2. 这样在同一个桶内各个元素之间的差值绝对值都小于等于 $t$。而相邻桶之间的元素，只需要校验一下两个桶之间的差值是否不超过 $t$。这样就可以以 $O(1)$ 的时间复杂度检测相邻 $2 \times k$ 个元素是否满足 $abs(nums[i] - nums[j]) \leq t$。
3. 而 $abs(i - j) \leq k$ 条件则可以通过在一重循环遍历时，将超出范围的 $nums[i - k]$ 从对应桶中删除，从而保证桶中元素一定满足 $abs(i - j) \leq k$。

具体步骤如下：

1. 将每个桶的大小设置为 $t + 1$。我们将元素按照大小依次放入不同的桶中。
2. 遍历数组 $nums$ 中的元素，对于元素$ nums[i]$ ：
    1. 如果 $nums[i]$ 放入桶之前桶里已经有元素了，那么这两个元素必然满足 $abs(nums[i] - nums[j]) \leq t$，
    2. 如果之前桶里没有元素，那么就将 $nums[i]$ 放入对应桶中。
    3. 再判断左右桶的左右两侧桶中是否有元素满足 $abs(nums[i] - nums[j]) <= t$。
    4. 然后将 $nums[i - k]$ 之前的桶清空，因为这些桶中的元素与 $nums[i]$ 已经不满足 $abs(i - j) \leq k$ 了。
3. 最后上述满足条件的情况就返回 `True` ，最终遍历完仍不满足条件就返回 `False` 。

## 思路 1：代码

```py
class Solution:
    def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) ->
bool:
        bucket_dict = dict()
        for i in range(len(nums)):
            # 将 nums[i] 划分到大小为 t + 1 的不同桶中
            num = nums[i] // (t + 1)

            # 桶中已经有元素了
            if num in bucket_dict:
                return True

            # 把 nums[i] 放入桶中
            bucket_dict[num] = nums[i]

            # 判断左侧桶是否满足条件
            if (num - 1) in buck^+ ^ict and abs(bucket_dict[num - 1] - nums[i])
<= t:
                return True
            # 判断右侧桶是否满足条件
            if (num + 1) in bucket_dict and abs(bucket_dict[num + 1] - nums[i])
<= t:
                return True
            # 将 i - k 之前的旧桶清除，因为之前的桶已经不满足条件了
            if i >= k:
                bucket_dict.pop(nums[i - k] // (t + 1))

        return False
```

## 思路 1：复杂度分析

- **时间复杂度**：$O(n)$。$n$ 是给定数组长度。
- **空间复杂度**：$O(min(n, k))$。桶中最多包含 $min(n, k + 1)$ 个元素。

## 思路 2：滑动窗口（固定长度）

1. 使用一个长度为 $k$ 的滑动窗口，每次遍历到 $nums[right]$ 时，滑动窗口内最多包含 $nums[right]$ 之前最多 $k$ 个元素。只需要检查前 $k$ 个元素是否在 $[nums[right] - t, nums[right] + t]$ 区间内即可。
2. 检查 $k$ 个元素是否在 $[nums[right] - t, nums[right] + t]$ 区间，可以借助保证有序的数据结构（比如 `SortedList` ）+ 二分查找来解决，从而减少时间复杂度。

具体步骤如下：

1. 使用有序数组类 $window$ 维护一个长度为 $k$ 的窗口，满足数组内元素有序，且支持增加和删除操作。
2. $left$、$right$ 都指向序列的第一个元素。即：`left = 0` ，`right = 0` 。
3. 将当前元素填入窗口中，即 `window.add(nums[right])` 。
4. 当窗口元素大于 $k$ 个时，即当 $right - left > k$ 时，移除窗口最左侧元素，并向右移动 $left$。
5. 当窗口元素小于等于 $k$ 个时：
   1. 使用二分查找算法，查找 $nums[right]$ 在 $window$ 中的位置 $idx$。
   2. 判断 $window[idx]$ 与相邻位置上的元素差值绝对值，若果满足 $abs(window[idx] - window[idx - 1]) \leq t$ 或者 $abs(window[idx + 1] - window[idx]) \leq t$ 时返回 `True` 。
6. 向右移动 $right$。
7. 重复 $3 \sim 6$ 步，直到 $right$ 到达数组末尾，如果还没找到满足条件的情况，则返回 `False` 。

## 思路 2：代码

```py
from sortedcontainers import SortedList

class Solution:
    def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) ->
bool:
        size = len(nums)
        window = SortedList()
        left, right = 0, 0
        while right < size:
            window.add(nums[right])
```

```
            if right - left > k:
                window.remove(nums[left])
                left += 1

            idx = bisect.bisect_left(window, nums[right])

            if idx > 0 and abs(window[idx] - window[idx - 1]) <= t:
                return True
            if idx < len(window) - 1 and abs(window[idx + 1] - window[idx]) <=
t:
                return True

            right += 1

        return False
```

## 思路 2：复杂度分析

- **时间复杂度**：$O(n \times \log(min(n, k)))$。
- **空间复杂度**：$O(min(n, k))$。

# 参考资料

- 【题解】利用桶的原理O(n)，Python3 - 存在重复元素 III - 力扣