

宫水三叶的刷题日记

单调队列

Author : 宫水三叶

Date : 2021/10/07

QQ Group: 703311589

WeChat : oaoaya

宫水三叶

刷题日记

公众号: 宫水三叶的刷题日记

噔噔噔噔，这是公众号「[宫水三叶的刷题日记](#)」的原创专题「单调队列」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「单调队列」即可获取最新下载链接。

💡下面介绍使用本合集的最佳使用实践：

学习算法：

1. 打开在线目录（[Github 版](#) & [Gitee 版](#)）；
2. 从侧边栏的类别目录找到「单调队列」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题；
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流   

题目描述

这是 LeetCode 上的 [1438. 绝对差不超过限制的最长连续子数组](#)，难度为 中等。

Tag：「滑动窗口」、「单调队列」、「二分」

给你一个整数数组 `nums`，和一个表示限制的整数 `limit`，请你返回最长连续子数组的长度，该子数组中的任意两个元素之间的绝对差必须小于或者等于 `limit`。

如果不存在满足条件的子数组，则返回 0。

示例 1：

宫水三叶
の
刷题日记

公众号：宫水三叶的刷题日记

输入: `nums = [8,2,4,7]`, `limit = 4`

输出: 2

解释: 所有子数组如下:

[8] 最大绝对差 $|8-8| = 0 \leq 4$.

[8,2] 最大绝对差 $|8-2| = 6 > 4$.

[8,2,4] 最大绝对差 $|8-2| = 6 > 4$.

[8,2,4,7] 最大绝对差 $|8-2| = 6 > 4$.

[2] 最大绝对差 $|2-2| = 0 \leq 4$.

[2,4] 最大绝对差 $|2-4| = 2 \leq 4$.

[2,4,7] 最大绝对差 $|2-7| = 5 > 4$.

[4] 最大绝对差 $|4-4| = 0 \leq 4$.

[4,7] 最大绝对差 $|4-7| = 3 \leq 4$.

[7] 最大绝对差 $|7-7| = 0 \leq 4$.

因此, 满足题意的最长子数组的长度为 2。

示例 2:

输入: `nums = [10,1,2,4,7,2]`, `limit = 5`

输出: 4

解释: 满足题意的最长子数组是 `[2,4,7,2]`, 其最大绝对差 $|2-7| = 5 \leq 5$ 。

示例 3:

输入: `nums = [4,2,2,2,4,4,2,2]`, `limit = 0`

输出: 3

提示:

- $1 \leq \text{nums.length} \leq 10^5$
- $1 \leq \text{nums}[i] \leq 10^9$
- $0 \leq \text{limit} \leq 10^9$

宫水三叶
の
刷题日记

公众号: 宫水三叶的刷题日记

二分 + 滑动窗口

执行结果：通过 [显示详情 >](#)

执行用时：214 ms，在所有 Java 提交中击败了 14.41% 的用户

内存消耗：47.2 MB，在所有 Java 提交中击败了 85.85% 的用户

炫耀一下：



[写题解，分享我的解题思路](#)

数据范围是 10^5 ，因此只能考虑「对数解法」和「线性解法」。

对数解法很容易想到「二分」。

在给定 `limit` 的情况下，倘若有「恰好」满足条件的区间长度为 `len`，必然存在满足条件且长度小于等于 `len` 的区间，同时必然不存在长度大于 `len` 且满足条件的区间。

因此长度 `len` 在数轴中具有「二段性」。

问题转化为「如何判断 `nums` 中是否有长度 `len` 的区间满足绝对值不超过 `limit`」

我们可以枚举区间的右端点 `r`，那么对应的左端点为 `r - len + 1`，然后使用「单调队列」来保存区间的最大值和最小值。

宫水三叶
の
刷题日记

公众号：宫水三叶的刷题日记

```

class Solution {
    public int longestSubarray(int[] nums, int limit) {
        int n = nums.length;
        int l = 1, r = n;
        while (l < r) {
            int mid = l + r + 1 >> 1;
            if (check(nums, mid, limit)) {
                l = mid;
            } else {
                r = mid - 1;
            }
        }
        return r;
    }

    boolean check(int[] nums, int len, int limit) {
        int n = nums.length;
        Deque<Integer> max = new ArrayDeque<>(), min = new ArrayDeque<>();
        for (int r = 0, l = r - len + 1; r < n; r++, l = r - len + 1) {
            if (!max.isEmpty() && max.peekFirst() < l) max.pollFirst();
            while (!max.isEmpty() && nums[r] >= nums[max.peekLast()]) max.pollLast();
            max.addLast(r);
            if (!min.isEmpty() && min.peekFirst() < l) min.pollFirst();
            while (!min.isEmpty() && nums[r] <= nums[min.peekLast()]) min.pollLast();
            min.addLast(r);
            if (l >= 0 && Math.abs(nums[max.peekFirst()] - nums[min.peekFirst()]) <= limit)
                return true;
        }
        return false;
    }
}

```

- 时间复杂度：枚举长度的复杂度为 $O(\log n)$ ，对于每次 `check` 而言，每个元素最多入队和出队常数次数，复杂度为 $O(n)$ 。整体复杂度为 $O(n \log n)$
- 空间复杂度： $O(n)$

宫水三叶
の
刷题日记

公众号: 宫水三叶的刷题日记

双指针

执行结果：通过 显示详情 >

执行用时：33 ms，在所有 Java 提交中击败了 93.43% 的用户

内存消耗：47.7 MB，在所有 Java 提交中击败了 80.08% 的用户

炫耀一下：



写题解，分享我的解题思路

上述解法我们是在对 `len` 进行二分，而事实上我们可以直接使用「双指针」解法找到最大值。

始终让右端点 `r` 右移，当不满足条件时让 `l` 进行右移。

同时，还是使用「单调队列」保存我们的区间最值，这样我们只需要对数组进行一次扫描即可得到答案。

```
class Solution {
    public int longestSubarray(int[] nums, int limit) {
        int n = nums.length;
        int ans = 0;
        Deque<Integer> max = new ArrayDeque<>(), min = new ArrayDeque<>();
        for (int r = 0, l = 0; r < n; r++) {
            while (!max.isEmpty() && nums[r] >= nums[max.peekLast()]) max.pollLast();
            while (!min.isEmpty() && nums[r] <= nums[min.peekLast()]) min.pollLast();
            max.addLast(r);
            min.addLast(r);
            while (Math.abs(nums[max.peekFirst()] - nums[min.peekFirst()]) > limit) {
                l++;
                if (max.peekFirst() < l) max.pollFirst();
                if (min.peekFirst() < l) min.pollFirst();
            }
            ans = Math.max(ans, r - l + 1);
        }
        return ans;
    }
}
```

- 时间复杂度：每个元素最多入队和出队常数次数，复杂度为 $O(n)$

- 空间复杂度： $O(n)$

**🔍更多精彩内容，欢迎关注：[公众号](#) / [Github](#) / [LeetCode](#) / [知乎](#) **

💡更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。

最新专题合集资料下载，可关注公众号「[宫水三叶的刷题日记](#)」，后台回复「单调队列」获取下载链接。

觉得专题不错，可以请作者吃糖🍬🍬🍬：

宫水三叶
の
刷题日记

公众号: 宫水三叶的刷题日记



“给作者手机充个电”

YOLO 的赞赏码

版权声明：任何形式的转载请保留出处 [Wiki](#)。