

【二分查找】-平均像素值

题目描述与示例

题目描述

一个图像有 n 个像素点，存储在一个长度为 n 的数组 `img` 里，每个像素点的取值范围 `[0,255]` 的正整数。

请你给图像每个像素点值加上一个整数 k （可以是负数），得到新图 `newImg`，使得新图 `newImg` 的所有像素平均值最接近中位值 `128`。

请输出这个整数 k 。

输入描述

n 个整数，中间用空格分开

输出描述

一个整数 k

补充说明

- $1 \leq n \leq 100$
- 如有多个整数 k 都满足，输出小的那个 k ；
- 新图的像素值会自动截取到 `[0,255]` 范围。当新像素值 < 0 ，其值会更改为 `0`；当新像素值 > 255 ，其值会更改为 `255`；例如 `newImg="-1 -2 256"`，会自动更改为 `"0 0 255"`

示例一

输入

```
1 129 130 129 130
```

输出

```
1 -2
```

说明

-1 的均值 128.5，-2 的均值为 127.5，输出较小的数 -2

示例二

输入

```
1 0 0 0 0
```

输出

```
1 128
```

解题思路

如果题目描述中没有以下这句话，那么本题非常简单

- 新图的像素值会自动截取到 $[0, 255]$ 范围。当新像素值 < 0 ，其值会更改为 0；当新像素值 > 255 ，其值会更改为 255；例如 `newImg="-1 -2 256"`，会自动更改为 `"0 0 255"`

由于上述这个限制，当我们选取一个特定的值 k 的时候，原数组并不会整体增加 nk （其中 n 是原数组的长度）。

我们考虑这个问题的二段性。

- 当 k 取一个绝对值很大的负数时，譬如 -255 ，那么所有像素值均会被更改为 0 ，此时新数组平均数为 0
- 当 k 取一个绝对值很大的正数时，譬如 255 ，那么所有像素值均会被更改为 255 ，此时新数组平均数为 255
- 因此，存在一个特定的 $k = \text{ans}$ ，能够使得新数组的所有像素值的平均值恰好大于等于 128 。此时 $k = \text{ans}$ 和 $k = \text{ans} - 1$ 这两个数，都是有可能使得像素值最接近 128 的结果，再加上一个判断即可。

上述二段性问题显然可以直接使用二分查找来完成。

接下来考虑二分查找的子问题 `cal_new_average(nums, k, n)`。

在确定了某一个特定的 k 的情况下，我们可以非常方便地**计算出新数组的平均值**。其过程如下

- 遍历原数组中的每一个数字 `num`，将其 $+k$ 得到修改后的新元素 `new_num`
 - 在遍历中，若
 - `new_num` 的值小于 0 ，则将其设置为 0
 - `new_num` 的值大于 255 ，则将其设置为 255
 - 将 `new_num` 的值加入一个新的变量 `new_sum` 中，表示新数组的和
- 遍历结束后，计算 `new_sum / n` 作为函数返回值

```
1 def cal_new_average(nums, k, n):
2     # 新数组的和，初始化为0
3     new_sum = 0
4     # 遍历原数组中的所有数字num
5     for num in nums:
6         # 加k得到新数组new_num
7         new_num = num + k
8         # 若新数字小于0，则修改为0
9         if new_num < 0:
10            new_num = 0
11        # 若新数字大于255，则修改为255
12        if new_num > 255:
13            new_num = 255
```

```
14         # 将新数字加入new_sum中
15         new_sum += new_num
16     # 返回新数组的值的平均值，注意此处使用/而不是//
17     # 因为可能得到浮点数
18     return new_sum / n
```

剩余的二分主框架部分，就是直接套模板了。

```
1  # k的左闭右开区间，right最大值为255，闭区间取值256
2  left, right = -255, 256
3
4  # 二分查找，计算第一个使得新数组平均值小于128的k
5  while left < right:
6      mid = (left + right) // 2
7      # 若计算结果小于128，说明整体平均值还可以更大，left右移
8      if cal_new_average(nums, mid, n) < 128:
9          left = mid + 1
10     # 若计算结果不小于128，说明整体平均值需要变小，right左移
11     else:
12         right = mid
13
14 # 退出循环后，k = left是使得cal_new_average(nums, k, n)恰好不小于128的值
15 # left和left-1都有可能是答案，看哪一个更接近128
16 # 若left-1（使得新数组平均值小于128的临界点）更接近128，则输出left-1
17 # 若left（使得新数组平均值大于等于128的临界点）更接近128，则输出left
18 if 128 - cal_new_average(nums, left-1, n) <= cal_new_average(nums, left, n) -
19     128:
19     print(left-1)
20 else:
21     print(left)
```

代码

Python

```
1  # 题目：【二分查找】2024E-平均像素值
2  # 分值：100
```

```

3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：二分查找
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 # 二分查找子问题
9 def cal_new_average(nums, k, n):
10     # 新数组的和，初始化为0
11     new_sum = 0
12     # 遍历原数组中的所有数字num
13     for num in nums:
14         # 加k得到新数组new_num
15         new_num = num + k
16         # 若新数字小于0，则修改为0
17         if new_num < 0:
18             new_num = 0
19         # 若新数字大于255，则修改为255
20         if new_num > 255:
21             new_num = 255
22         # 将新数字加入new_sum中
23         new_sum += new_num
24     # 返回新数组的和的平均值，注意此处使用/而不是//
25     # 因为可能得到浮点数
26     return new_sum / n
27
28
29 # 输入原数组
30 nums = list(map(int, input().split()))
31 # 计算原数组的长度n
32 n = len(nums)
33
34
35 # k的左闭右开区间，right最大值为255，闭区间取值256
36 left, right = -255, 256
37
38 # 二分查找，计算第一个使得新数组平均值小于128的k
39 while left < right:
40     mid = (left + right) // 2
41     # 若计算结果小于128，说明整体平均值还可以更大，left右移
42     if cal_new_average(nums, mid, n) < 128:
43         left = mid + 1
44     # 若计算结果不小于128，说明整体平均值需要变小，right左移
45     else:
46         right = mid
47
48 # 退出循环后，k = left是使得cal_new_average(nums, k, n)恰好大于等于（不小于）128的值
49 # left和left-1都有可能是答案，看哪一个更接近128

```

```

50 # 若left-1（使得新数组平均值小于128的临界点）更接近128，则输出left-1
51 # 若left（使得新数组平均值大于等于128的临界点）更接近128，则输出left
52 if 128 - cal_new_average(nums, left-1, n) <= cal_new_average(nums, left, n) -
    128:
53     print(left-1)
54 else:
55     print(left)

```

Java

```

1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4
5 public class Main {
6
7     // 二分查找子问题
8     public static double calNewAverage(List<Integer> nums, int k, int n) {
9         // 新数组的和，初始化为0
10        double newSum = 0;
11        // 遍历原数组中的所有数字num
12        for (int num : nums) {
13            // 加k得到新数组newNum
14            int newNum = num + k;
15            // 若新数字小于0，则修改为0
16            if (newNum < 0) {
17                newNum = 0;
18            }
19            // 若新数字大于255，则修改为255
20            if (newNum > 255) {
21                newNum = 255;
22            }
23            // 将新数字加入newSum中
24            newSum += newNum;
25        }
26        // 返回新数组的和的平均值，注意此处使用/而不是//
27        // 因为可能得到浮点数
28        return newSum / n;
29    }
30
31    public static void main(String[] args) {
32        Scanner scanner = new Scanner(System.in);
33
34        // 输入原数组
35        List<Integer> nums = new ArrayList<>();

```

```

36     while (scanner.hasNextInt()) {
37         nums.add(scanner.nextInt());
38     }
39     // 计算原数组的长度n
40     int n = nums.size();
41
42     // k的左闭右开区间, right最大值为255, 闭区间取值256
43     int left = -255, right = 256;
44
45     // 二分查找, 计算第一个使得新数组平均值小于128的k
46     while (left < right) {
47         int mid = left + (right - left) / 2;
48         // 若计算结果小于128, 说明整体平均值还可以更大, left右移
49         if (calNewAverage(nums, mid, n) < 128) {
50             left = mid + 1;
51         }
52         // 若计算结果不小于128, 说明整体平均值需要变小, right左移
53         else {
54             right = mid;
55         }
56     }
57
58     // 退出循环后, k = left是使得calNewAverage(nums, k, n)恰好大于等于 (不小于)
128的值
59     // left和left-1都有可能是答案, 看哪一个更接近128
60     // 若left-1 (使得新数组平均值小于128的临界点) 更接近128, 则输出left-1
61     // 若left (使得新数组平均值大于等于128的临界点) 更接近128, 则输出left
62     if (128 - calNewAverage(nums, left - 1, n) <= calNewAverage(nums,
left, n) - 128) {
63         System.out.println(left - 1);
64     } else {
65         System.out.println(left);
66     }
67
68     scanner.close();
69 }
70 }
71

```

C++

```

1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4

```

```
5 using namespace std;
6
7 // 二分查找子问题，注意返回值是浮点数
8 double calNewAverage(const vector<int>& nums, int k, int n) {
9     // 新数组的和，初始化为0，注意类型是浮点数
10    double newSum = 0;
11    // 遍历原数组中的所有数字num
12    for (int num : nums) {
13        // 加k得到新数组newNum
14        int newNum = num + k;
15        // 若新数字小于0，则修改为0
16        if (newNum < 0) {
17            newNum = 0;
18        }
19        // 若新数字大于255，则修改为255
20        if (newNum > 255) {
21            newNum = 255;
22        }
23        // 将新数字加入newSum中
24        newSum += newNum;
25    }
26    // 返回新数组的值的平均值
27    return newSum / n;
28 }
29
30 int main() {
31     // 输入原数组
32     vector<int> nums;
33     int num;
34     while (cin >> num) {
35         nums.push_back(num);
36         if (cin.peek() == '\n') break; // 读取到换行符时结束输入
37     }
38     // 计算原数组的长度n
39     int n = nums.size();
40
41     // k的左闭右开区间，right最大值为255，闭区间取值256
42     int left = -255, right = 256;
43
44     // 二分查找，计算第一个使得新数组平均值小于128的k
45     while (left < right) {
46         int mid = left + (right - left) / 2;
47         // 若计算结果小于128，说明整体平均值还可以更大，left右移
48         if (calNewAverage(nums, mid, n) < 128) {
49             left = mid + 1;
50         }
51         // 若计算结果不小于128，说明整体平均值需要变小，right左移
```



```

52         else {
53             right = mid;
54         }
55     }
56
57     // 退出循环后, k = left是使得calNewAverage(nums, k, n)恰好不小于128的值
58     // left和left-1都有可能是答案, 看哪一个更接近128
59     // 若left-1 (使得新数组平均值小于128的临界点) 更接近128, 则输出left-1
60     // 若left (使得新数组平均值大于等于128的临界点) 更接近128, 则输出left
61     if (128 - calNewAverage(nums, left - 1, n) <= calNewAverage(nums, left, n)
    - 128) {
62         cout << left - 1 << endl;
63     } else {
64         cout << left << endl;
65     }
66
67     return 0;
68 }
69

```

时空复杂度

时间复杂度: $O(N \log M)$ 。 N 为子问题的时间复杂度, M 为二分查找的范围 $[-255, 255]$ 的大小。

空间复杂度: $O(1)$ 。 仅需维护若干常数变量。