【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 <= n <= 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 = 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 = ans` 和 `k = 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_num / n` 作为函数返回值

```python
def cal_new_average(nums, k, n):
    # 新数组的和，初始化为0
    new_sum = 0
    # 遍历原数组中的所有数字num
    for num in nums:
        # 加k得到新数组new_num
        new_num = num + k
        # 若新数字小于0，则修改为0
        if new_num < 0:
            new_num = 0
        # 若新数字大于255，则修改为255
        if new_num > 255:
            new_num = 255
```

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

剩余的二分主框架部分，就是直接套模板了。

```
# k的左闭右开区间，right最大值为255，闭区间取值256
left, right = -255, 256

# 二分查找，计算第一个使得新数组平均值小于128的k
while left < right:
    mid = (left + right) // 2
    # 若计算结果小于128，说明整体平均值还可以更大，left右移
    if cal_new_average(nums, mid, n) < 128:
        left = mid + 1
    # 若计算结果不小于128，说明整体平均值需要变小，right左移
    else:
        right = mid

# 退出循环后，k = left是使得cal_new_average(nums, k, n)恰好不小于128的值
# left和left-1都有可能是答案，看哪一个更接近128
# 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
# 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
if 128 - cal_new_average(nums, left-1, n) <= cal_new_average(nums, left, n) - 128:
    print(left-1)
else:
    print(left)
```

代码

Python

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

```python
# 作者：许老师-闭着眼睛学数理化
# 算法：二分查找
# 代码看不懂的地方，请直接在群上提问


# 二分查找子问题
def cal_new_average(nums, k, n):
    # 新数组的和，初始化为0
    new_sum = 0
    # 遍历原数组中的所有数字num
    for num in nums:
        # 加k得到新数组new_num
        new_num = num + k
        # 若新数字小于0，则修改为0
        if new_num < 0:
            new_num = 0
        # 若新数字大于255，则修改为255
        if new_num > 255:
            new_num = 255
        # 将新数字加入new_sum中
        new_sum += new_num
    # 返回新数组的和的平均值，注意此处使用/而不是//
    # 因为可能得到浮点数
    return new_sum / n


# 输入原数组
nums = list(map(int, input().split()))
# 计算原数组的长度n
n = len(nums)


# k的左闭右开区间，right最大值为255，闭区间取值256
left, right = -255, 256

# 二分查找，计算第一个使得新数组平均值小于128的k
while left < right:
    mid = (left + right) // 2
    # 若计算结果小于128，说明整体平均值还可以更大，left右移
    if cal_new_average(nums, mid, n) < 128:
        left = mid + 1
    # 若计算结果不小于128，说明整体平均值需要变小，right左移
    else:
        right = mid

# 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
# left和left-1都有可能是答案，看哪一个更接近128
```

```
# 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
# 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
if 128 - cal_new_average(nums, left-1, n) <= cal_new_average(nums, left, n) - 128:
    print(left-1)
else:
    print(left)
```

Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {

    // 二分查找子问题
    public static double calNewAverage(List<Integer> nums, int k, int n) {
        // 新数组的和，初始化为0
        double newSum = 0;
        // 遍历原数组中的所有数字num
        for (int num : nums) {
            // 加k得到新数组newNum
            int newNum = num + k;
            // 若新数字小于0，则修改为0
            if (newNum < 0) {
                newNum = 0;
            }
            // 若新数字大于255，则修改为255
            if (newNum > 255) {
                newNum = 255;
            }
            // 将新数字加入newSum中
            newSum += newNum;
        }
        // 返回新数组的和的平均值，注意此处使用/而不是//
        // 因为可能得到浮点数
        return newSum / n;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 输入原数组
        List<Integer> nums = new ArrayList<>();
```

```
        while (scanner.hasNextInt()) {
            nums.add(scanner.nextInt());
        }
        // 计算原数组的长度n
        int n = nums.size();

        // k的左闭右开区间，right最大值为255，闭区间取值256
        int left = -255, right = 256;

        // 二分查找，计算第一个使得新数组平均值小于128的k
        while (left < right) {
            int mid = left + (right - left) / 2;
            // 若计算结果小于128，说明整体平均值还可以更大，left右移
            if (calNewAverage(nums, mid, n) < 128) {
                left = mid + 1;
            }
            // 若计算结果不小于128，说明整体平均值需要变小，right左移
            else {
                right = mid;
            }
        }

        // 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
        // left和left-1都有可能是答案，看哪一个更接近128
        // 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
        // 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
        if (128 - calNewAverage(nums, left - 1, n) <= calNewAverage(nums, left, n) - 128) {
            System.out.println(left - 1);
        } else {
            System.out.println(left);
        }

        scanner.close();
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <cmath>

```

```cpp
using namespace std;

// 二分查找子问题，注意返回值是浮点数
double calNewAverage(const vector<int>& nums, int k, int n) {
    // 新数组的和，初始化为0，注意类型是浮点数
    double newSum = 0;
    // 遍历原数组中的所有数字num
    for (int num : nums) {
        // 加k得到新数组newNum
        int newNum = num + k;
        // 若新数字小于0，则修改为0
        if (newNum < 0) {
            newNum = 0;
        }
        // 若新数字大于255，则修改为255
        if (newNum > 255) {
            newNum = 255;
        }
        // 将新数字加入newSum中
        newSum += newNum;
    }
    // 返回新数组的和的平均值
    return newSum / n;
}

int main() {
    // 输入原数组
    vector<int> nums;
    int num;
    while (cin >> num) {
        nums.push_back(num);
        if (cin.peek() == '\n') break; // 读取到换行符时结束输入
    }
    // 计算原数组的长度n
    int n = nums.size();

    // k的左闭右开区间，right最大值为255，闭区间取值256
    int left = -255, right = 256;

    // 二分查找，计算第一个使得新数组平均值小于128的k
    while (left < right) {
        int mid = left + (right - left) / 2;
        // 若计算结果小于128，说明整体平均值还可以更大，left右移
        if (calNewAverage(nums, mid, n) < 128) {
            left = mid + 1;
        }
        // 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(NlogM)`。`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。