中位数问题

PPt report template of Hunan university

排序技术研讨

湖南大学
HUNAN UNIVERSITY

概述

概念

中位数是一种统计学上的概念，它是一组数据按照大小顺序排列后，位于中间位置的数。如果数据的数量是奇数，那么中位数就是正中间的那个数；如果数据的数量是偶数，那么中位数就是中间两个数的平均值。

例如，对于这组数据：1, 3, 3, 6, 7, 8, 9，中位数是6。

对于这组数据：1, 2, 3, 4，中位数是 (2 + 3) / 2 = 2.5。

中位数是一种衡量数据集中趋势的重要工具，它不受极端值的影响，因此在数据存在异常值时，中位数往往比平均数更能反映数据的真实情况。

标题一

副标题

查找中位数（分治策略） 01

02 动态中位数

中位数贪心 03

目录

01 算法思想

由在此输入详细介绍，以表达项目工作的详细资料和文字信息。

02 求解过程

由在此输入详细介绍，以表达项目工作的详细资料和文字信息。

03 算法具体步骤

由在此输入详细介绍，以表达项目工作的详细资料和文字信息。

04 性能分析

由在此输入详细介绍，以表达项目工作的详细资料和文字信息。

湖南大学
HUNAN UNIVERSITY

第一部分

查找中位数（分治策略）

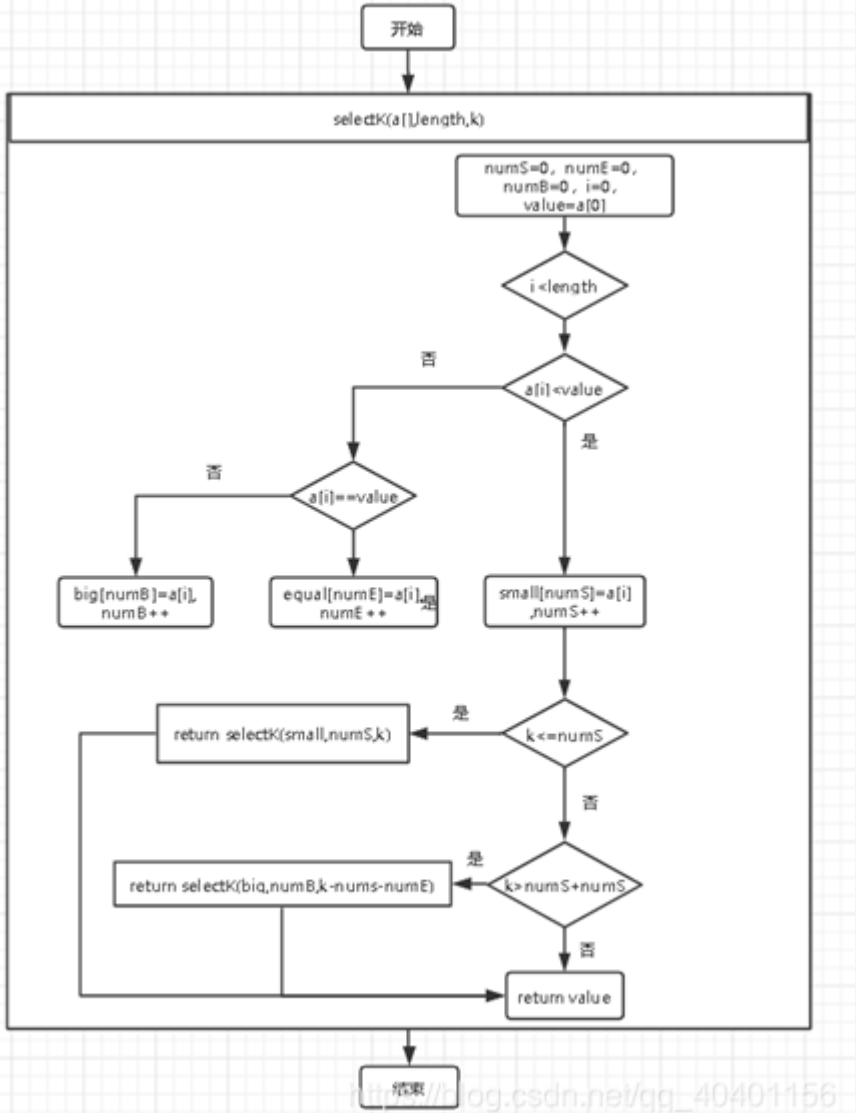
湖南大学
HUNAN UNIVERSITY

第一部分

算法思想

要找出一个数组的中位数，最简单的方法当然是将数组排序，但快速排序的时间复杂度也需要O(nlogn)，我们可以寻找更快的算法来解决。

首先对于一个长度为n的有序数组a[n]，若n为偶数，则中位数为(a[n/2]+a[n/2-1])/2，若n为奇数，则中位数为a[n/2]，那么问题的关键就是找到a[n/2]和a[n/2-1]，然而这是在有序数组中的，那么换到无序的数组中，我们可以把问题转换为求数组中第n/2大的和第n/2+1的数。



https://blog.csdn.net/qq_40401156

换到无序的数组中，我们可以把问题转换为求数组中第n/2大的和第n/2+1的数，再一般点就是求一个无序数组中第k大的数。那么如何求第k大的数呢，我们可以先在数组中取一个值value，将数组划分为小于value的small，等于value的equal，大于value的big三个部分，分别记三个部分的元素个数为numS、numE、numB，

若k<=numS，则说明我们要找的数就在small中，

若numS<k<=numS+numE，则说明我们要找的值在equal中，而又因为equal中的值都相等，因此我们要找的值就等于equal中元素的值，

若k>numS+numE，则我们要找的数就在big中；在一趟比较完成之后，若我们没有得到我们需要的值，只得到了我们需要的数所在的范围，那么我们可以再对得到的small或big再使用以上算法，直到得到需要的值。

湖南大学
HUNAN UNIVERSITY

第二部分

求解过程

过程

- 在数组中取一个值value，将数组划分为小于value的small，等于value的equal，大于value的big三个部分，分别记三个部分的元素个数为numS、numE、numB
- 比较k的值和各部分元素的个数
- 若k<=numS，则说明我们要找的数就在small中
- 若k>numS+numE，则我们要找的数就在big中，在这部分继续这个分治的过程直到找到对应的数字

湖南大学
HUNAN UNIVERSITY

第三部分

具体步骤讲解

选取与分区

```
int selectK(int a[], int length, int k)  //主体函数
{
    int *small = new int[length];
    int *equal = new int[length];
    int *big = new int[length];  //建立三个部分的数组
    int value = a[0];  //将第一个元素作为参考值
    int numS = 0, numE = 0, numB = 0;
for (int i = 0; i < length; i++)
{
    if (a[i] < value)  //找到small部分的元素个数
    {
        small[numS] = a[i];
        numS++;
    }
    else if (a[i] == value)  //找到equal部分的元素个数
    {
        equal[numE] = a[i];
        numE++;
    }
    else  //找到big部分的元素个数
    {
        big[numB] = a[i];
        numB++;
    }
```

递归的选择

```
if (k <= numS)return selectK(small, numS, k);  //如果答案在比k小的部分里面，继续递归
else if (k > numE + numS)return selectK(big, numB, k-numS-numE);  //如果答案在比k大的部分里，继续递归
else return value;  //如果恰好value值就是我们要找的第k大的数，则直接return答案
```

湖南大學 HUNAN UNIVERSITY

湖南大学
HUNAN UNIVERSITY

第四部分

性能分析

性能分析

我们记长度为 n 的数组查找中位数所需要的时间为 T(n)，有：

$$T(n)=\begin{cases}O(1), n=1\\ T(numS)|T(numE)+O(n), n>1\end{cases}$$

由于 numS, numE 都小于等于 n，我们可以将 numS 与 numE 记为 n 减去一个数，记为 n-m，那么当 n>1 时，有：

$$T(n)=T(n-m_1)+O(n)=T(n-m_1-m_2)+O(n)+O(n-m_1)$$

$$=T(n-m_1-m_2-m_2)+O(n)+O(n-m_1)+O(n-m_1-m_1)=\cdots$$

$$=T\left(n-\sum_{i=1}^{k}m_i\right)+O(n)+O(n-m_1)+\cdots+O(n-\sum_{i=1}^{k-1}m_i)$$

当 k 越来越大时，$n-\sum_{i=1}^{k}m_i$将趋近于 1，而$O(n-m_1)$、$O(n-m_1-m_1)$、$O(n-\sum_{i=1}^{k-1}m_i)$也均小于 O(n)，因此该算法的时间复杂度为O(n)。

https://blog.csdn.net/qq_40401156

最佳和平均情况：在最佳和平均情况下，每次都能将数组大小减少大约一半。这是因为选择的value能较好地将数组平均分割。因此，算法的时间复杂度是O(n)，其中n是数组的长度。这与快速排序的平均时间复杂度类似，但由于只需要递归进入一个分区，所以常数因子较小。

最坏情况：在最坏情况下，每次选取的value可能总是数组中的最小或最大元素，这样每次只能排除一个元素，使得算法复杂度退化为O(n^2)。但这种情况可以通过随机选择pivot来有效避免。

湖南大学 HUNAN UNIVERSITY

性能分析

本算法和快速排序的性能对比

数据规模n	本算法	快速排序
1000	1ms	1ms
10000	2ms	2ms
100000	7ms	28ms
1000000	60ms	1396ms
10000000	4537ms	14353ms

湖南大學 HUNAN UNIVERSITY

湖南大学
HUNAN UNIVERSITY

第二部分

动态中位数

动态中位数

问题描述

依次读入一个整数序列，每当已经读入的整数个数为奇数时，输出已读入的整数构成的序列的中位数。

输入格式

第一行输入一个整数 P，代表后面数据集的个数，接下来若干行输入各个数据集。

每个数据集的第一行首先输入一个代表数据集的编号的整数。

然后输入一个整数 M，代表数据集中包含数据的个数，M 一定为奇数，数据之间用空格隔开。

数据集的剩余行由数据集的数据构成，每行包含 10 个数据，最后一行数据量可能少于 10 个，数据之间用空格隔开。

湖南大学
HUNAN UNIVERSITY

第一部分

算法思想

最小堆的堆顶和最大堆的堆顶是大小顺序相邻的元素。维护两个堆，其实也就是在维护这种相邻关系。而保持两个堆中的元素个数相当（在已读入个数是奇数时相差1），其实也就是在维护中位数始终处于大顶堆的堆顶。不妨假设当前已经读入数字的个数为m个

1. 若m是奇数，那么大顶堆中维护了[m/2]个元素，而小顶堆中维护了[m/2]个元素。

2. 若m是偶数，那么两个堆各维护了m/2个元素。
考虑中位数是第[(m+1)/2]个元素，即第[(m)/2]个元素，不难看出中位数始终位于大顶堆的堆顶（请注意，我们在奇数时才更新中位数）。

湖南大学
HUNAN UNIVERSITY

第二部分

求解过程

求解过程

对顶堆算法：维护一个最大堆，一个最小堆。

每当读入数据时，将新读入的数据压入最大堆中。

当最大堆的元素个数大于最小堆元素个数+1时，将最大堆的堆顶元素弹出并压入最小堆（即已经读入的元素个数m是偶数时，维持最大堆和最小堆中的元素个数相当）。

如最大堆的堆顶元素大于最小堆的堆顶元素，则弹出最大堆堆顶元素压入最小堆，并弹出最小堆堆顶元素压入最大堆。

湖南大学
HUNAN UNIVERSITY

第三部分

具体步骤讲解

```
int m, n;
cin >> m >> n;
printf("%d %d\n", m, (n + 1) / 2);
priority_queue<int> max_heap;//建立大根堆
priority_queue<int, vector<int>, greater<int>> min_heap;//小根堆
for (int i = 0; i < n; i++)
{
    int x;
    cin >> x;
    max_heap.push(x);
    if (!min_heap.empty() && min_heap.top() < max_heap.top())//如果大根堆的堆顶元素大于小根堆堆顶元素
    {
        auto max_heap_elem = max_heap.top();
        auto min_heap_elem = min_heap.top();
        max_heap.pop();
        min_heap.pop();
        min_heap.push(max_heap_elem);
        max_heap.push(min_heap_elem);//大小根堆的堆顶元素弹出并弹进对方堆中
    }
    if (max_heap.size() > min_heap.size() + 1)//如果大根堆的元素数量比小根堆大2
    {
        min_heap.push(max_heap.top());//将大根堆的堆顶元素压入小根堆中以维持m为偶数时两者的元素数量一致
        max_heap.pop();
    }
    if (!(i & 1)) // 考虑到这里是从0开始的 所以其实是奇数更新中位
    {
        cnt++;
        printf("%d ", max_heap.top());
```

湖南大学
HUNAN UNIVERSITY

第四部分

性能分析

性能分析

1.**使用两个优先队列**:
- **大根堆** (max_heap): 存储较小的一半元素，堆顶是这部分的最大值。
- **小根堆** (min_heap): 存储较大的一半元素，堆顶是这部分的最小值。

2.**处理每个数值**:
- 对于每个测试用例中的 n 个数，每个数值的处理包括插入操作和可能的堆调整。
- 插入操作：每次读取一个数值 x 并插入到大根堆中，这个操作的时间复杂度是 O(log n).
- 堆调整:
  - 如果大根堆的顶部元素大于小根堆的顶部元素，交换这两个堆顶元素，每次交换的时间复杂度也是 O(log n).
  - 确保两个堆的大小平衡（大根堆的元素数量可以比小根堆多1或相等），这可能涉及将大根堆的顶部元素移动到小根堆，操作复杂度为 O(log n).

3.**输出中位数**:
- 对于每个测试用例，每隔一个数（即输出序列的奇数位置）输出大根堆的顶部元素作为中位数。
- 每10个中位数输出后会换行，这部分操作的复杂度是 O(1)。

4.**总体复杂度**:
- 每个数的处理涉及几个 O(log n) 的操作，因此整体复杂度对于每个测试用例是 O(n log n)。

湖南大学
HUNAN UNIVERSITY

第三部分

中位数贪心

中位数贪心

问题描述

462. 最小操作次数使数组元素相等 II

中等 | 相关标签 | 相关企业 | A文

给你一个长度为 `n` 的整数数组 `nums` ，返回使所有数组元素相等需要的最小操作数。

在一次操作中，你可以使数组中的一个元素加 `1` 或者减 `1` 。

示例 1：

> **输入：** `nums = [1,2,3]`
> **输出：** `2`
> **解释：**
> 只需要两次操作（每次操作指南使一个元素加 1 或减 1）：
> `[1,2,3] => [2,2,3] => [2,2,2]`

贪心，把每个数都变成其中位数就是该题的最优解

贪心的证明：

将$nums$升序排序，假设中位数为$nums[m]$，记将$nums[m]$左侧所有元素都变成$nums[m]$的代价为A，将$nums[t]$右侧所有元素都变成$nums$[m]的代价为B

假设将所有元素变成$nums[m]$左侧的某个$nums$[t]的代价更小，假设$nums[t]$左侧有x个元素(包括$nums[t]$)，则将这x个元素变为$nums[t]$即将这些元素变为$nums[m]$再变为$nums[t]$，变化量$-xd$(记$nums[t]-nums[k]=d$)。将$nums[t]$右侧的所有元素都变成$nums$[t]等价于将$nums[t]$右侧所有元素都变成$nums[m]$再变成$nums[t]$，变化量为$(n-x)*d$故将所有元素都变成$nums[t]$的花费可以表示成：

$$A+B-xd+(n-x)*d=A+B+nd-2xdA$$

由于变成$nums[t]$代价更小，所以

$A+B+nd-2xd< A+B \rightarrow x>n/2$

这与$nums[t]$在中位数$nums$[m]左侧矛盾，同理可证中位数右侧不存在代价更小的点，故将所有数变为中位数代价最小

综上只需将所有数排序后累加所有数与中位数的差的绝对值即可

性能分析

1.**排序操作**：对数组进行排序是时间复杂度最高的步骤，通常使用的排序算法（如快速排序、归并排序等）具有 O(n log n) 的时间复杂度。
2.**计算代价**：遍历数组并计算每个元素与中位数的差的绝对值。这个步骤的时间复杂度是 O(n)，因为你需要访问数组中的每一个元素一次。
因此，整个算法的总时间复杂度主要由排序步骤决定，即 O(n log n)。

谢 谢 观 看

thank you for watching

参考文献

作者：xperia2链接：https://www.acwing.com/file_system/file/content/whole/index/content/9658360/来源：AcWing

Csdn ： 查找中位数（分治策略）

$Note$-中位数贪心