剑指Offer（六十三）：数据流中的中位数

2018年2月1日 10:42:16 5 4,605 °C 编辑

一、前言

本系列文章为《剑指Offer》刷题笔记。

刷题平台：牛客网

书籍下载：共享资源

二、题目

如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。

1、思路

这道题的解法有很多，文本使用最大堆和最小堆实现。

主要思想：

最大堆 | 最小堆

我们将数据分为两部分，位于左边最大堆的数据比右边最小堆的数据要小，左、右两边内部的数据没有排序，也可以根据左边最大的数及右边最小的数得到中位数。

接下来考虑用最大堆和最小堆实现的一些细节。

首先要保证数据平均分配到两个堆中，因此两个堆中数据的数目之差不能超过1.为了实现平均分配，可以在数据的总数目是偶数时把新数据插入到最小堆中，否则插入到最大堆中。

此外，还要保证最大堆中所有数据小于最小堆中数据。所以，新传入的数据需要先和最大堆的最大值或者最小堆中的最小值进行比较。以总数目为偶数为例，按照我们制定的规则，新的数据会被插入到最小堆中，但是在这之前，我们需要判断这个数据和最大堆中的最大值谁更大，如果最大堆中的数据比较大，那么我们就需要把当前数据插入最大堆，然后弹出新的最大值，再插入到最小堆中。由于最终插入到最小堆的数字是原最大堆中最大的数字，这样就保证了最小堆中所有数字都大于最大堆的数字。

2、代码

下面代码中，我们基于stl中的函数push_heap、pop_heap以及vector实现堆。比较仿函数less和greater分别用来实现最大堆和最小堆。

C++：

```C++
class Solution {
public:
    void Insert(int num)
    {
        // 如果已有数据为偶数，则放入最小堆
        if(((max.size() + min.size()) & 1) == 0){
            // 如果插入的数字小于最大堆里的最大的数，则将数字插入最大堆
            // 并将最大堆中的最大的数字插入到最小堆
            if(max.size() > 0 && num < max[0]){
                // 插入数据插入到最大堆数组
```

```
                max.push_back(num);
                // 调整最大堆
                push_heap(max.begin(), max.end(), less<int>());
                // 拿出最大堆中的最大数
                num = max[0];
                // 删除最大堆的栈顶元素
                pop_heap(max.begin(), max.end(), less<int>());
                max.pop_back();
            }
            // 将数据插入最小堆数组
            min.push_back(num);
            // 调整最小堆
            push_heap(min.begin(), min.end(), greater<int>());
        }
        // 已有数据为奇数，则放入最大堆
        else{
            if(min.size() > 0 && num > min[0]){
                // 将数据插入最小堆
                min.push_back(num);
                // 调整最小堆
                push_heap(min.begin(), min.end(), greater<int>());
                // 拿出最小堆的最小数
                num = min[0];
                // 删除最小堆的栈顶元素
                pop_heap(min.begin(), min.end(), greater<int>());
                min.pop_back();
            }
            // 将数据插入最大堆
            max.push_back(num);
            push_heap(max.begin(), max.end(), less<int>());
        }
    }
    double GetMedian()
    {
        // 统计数据大小
        int size = min.size() + max.size();
        if(size == 0){
            return 0;
        }
        // 如果数据为偶数
        if((size & 1) == 0){
            return (min[0] + max[0]) / 2.0;
        }
        // 奇数
        else{
            return min[0];
        }
    }
private:
    // 使用vector建立最大堆和最小堆,min是最小堆数组,max是最大堆数组
    vector<int> min;
    vector<int> max;
};
```