实验1 分治与递归Quiz

1. 问题描述

1.1. 实验原题

设有n个互不相同的元素$x_1, x_2, x_3 \dots x_n$，每个元素x_i带有一个权值w_i，且$\sum_{i=1}^{n} w_i = 1$若元素x_k满足$\sum_{x_i<x_k} w_i \leq \frac{1}{2}$且$\sum_{x_i>x_k} w_i \leq \frac{1}{2}$则称$x_k$为$x_1, x_2, x_3 \dots x_n$的带权中位数。

请编写一个算法，能够在最坏的情况下使用$O(n)$时间找出n个元素的带权中位数

1.2. 初步分析

1.2.1. 问题描述:

给定一个整数集合（互不相同）以及对应的权重集合

权重集合中的权重之和为1

我们需要找到一个数字（带权中位数），其左侧所有数字的权重总和不超过0.5，其右侧所有数字的权重总和也不超过0.5

1.2.2. 输入

整数集合 $\{x_1, x_2, \dots, x_n\}$

权重集合 $\{w_1, w_2, \dots, w_n\}$

1.2.3. 输出

带权中位数 x_k

1.2.4. 关键点

找到这样的带权中位数是主要问题的核心

对于每一个数字 x_i，我们需要确定两个事实

1. 所有小于 x_i 的数字的权重总和
2. 所有大于 x_i 的数字的权重总和

如果以上两个权重总和都不超过0.5，则x_i就是带权中位数

1.2.5. 基本方法

简单地说，我们可以遍历每一个数字，并为其计算两边的权重总和，直到找到符合条件的数字为止

但直接这样做的时间复杂度为$O(n^2)$，因为对于每一个数字，我们都进行了一次线性搜索来找到两侧的权重总和

为了达到$O(n)$的时间复杂度，我们需要更巧妙的方法

1.2.6. 代码的基本思路

代码中提供了一个`WeightedMedianCalculator`类来计算带权中位数

使用了`do...while`循环遍历整数集合中的每个数字，并使用`ProcessCurrentIndex`方法计算对应的权重总和

如果找到满足条件的数字，则直接输出

1.3. 问题分析

1.3.1. 问题深入理解

我们需要找到一个元素x_k，它将整个数据集分成两个部分：小于x_k的元素集合和大于x_k的元素集合。这两个集合的权重总和都需要小于或等于0.5

这个问题是经典的中位数问题的变种，其中元素有权重，并且我们寻找的是带权中位数

1.3.2. 直观方法的局限性

简单直观的方法是将元素按照大小排序，然后遍历它们，累加权重，直到找到带权中位数。但这种方法的时间复杂度是$O(n \log n)$，因为排序本身就需要$O(n \log n)$时间

1.3.3. 线性时间选择算法的应用

要在$O(n)$时间内解决这个问题，我们可以借鉴快速选择算法（QuickSelect）的思想，这是一种用于在未排序的数组中找到第k个最小元素的算法，其平均时间复杂度为$O(n)$

我们可以将快速选择算法稍作修改，使其在选择元素时考虑权重，从而找到带权中位数

1.3.4. 权重的处理

在每一步快速选择中，我们选择一个基准元素，并根据它将其它元素划分为两个子集：小于基准的元素和大于基准的元素

我们还需要计算每个子集的权重总和

如果某一侧的权重总和大于0.5，带权中位数必定在那一侧

如果两侧的权重总和都小于或等于0.5，但它们的和大于或等于1，那么基准元素就是带权中位数

1.3.5. 算法复杂度

与快速选择算法一样，该算法的平均时间复杂度是$O(n)$，但最坏情况下（如果每次划分都很不平衡）它可以达到$O(n^2)$

为了优化最坏情况的时间复杂度，我们可以使用类似于快速排序中的三数取中策略，或者使用随机化方法来选择基准

1.4. 算法设计

1.4.1. 问题背景与核心思想

在问题中，我们不仅关心元素的大小，还关心每个元素对整体的影响，即其权重。传统的中位数算法关注元素的顺序，但在这里，我们也需要考虑元素的权重。核心思想是找到一个元素，使其左侧的权重之和和右侧的权重之和都不超过 0.5

1.4.2. 分析与设计

1.4.2.1. 遍历法

由于需要在 $O(n)$ 时间内找到带权中位数，一个直观的想法是遍历每个元素，并计算其两侧的权重之和。您的代码基于这种方法：

- 对于每个元素，我们都尝试将其视为潜在的带权中位数
- 计算该元素左侧所有元素的权重之和，然后计算右侧所有元素的权重之和
- 如果两侧的权重都不超过 0.5，则此元素即为所求的带权中位数

1.4.2.2. 权重计算

为了确定一个元素是否为带权中位数，我们需要知道其左侧和右侧的权重之和。这可以通过遍历所有其他元素并累加权重来实现。这种方法的效率取决于我们如何组织和使用数据

1.4.2.3 早期终止

一旦找到带权中位数，我们就可以停止进一步的搜索。但您的代码会继续遍历，确保只有一个带权中位数。这是一种保守的方法，确保了结果的准确性

1.4.3. 实现细节

1.4.3.1. 封装思想：

使用 `WeightedMedianCalculator` 类的设计体现了对象封装的思想。将整个带权中位数的计算过程封装为一个对象，这样做有助于隔离复杂性，并使得主程序看起来更加简洁。此外，通过这种方式，我们还可以轻松地扩展功能或进行代码重构，而不会对主程序造成太大的影响

1.4.3.2. 处理每个元素：

通过 `ProcessCurrentIndex` 方法，我专门处理当前索引下的元素。这个方法的存在使得核心逻辑与遍历过程分离，每次只关注一个元素，而不是整个数组，这样更符合单一职责原则

1.4.3.3. 权重计算的辅助函数

`CalculateWeight` 函数是一个辅助方法，其主要任务是计算给定元素左侧或右侧的权重之和。将这部分逻辑单独提取出来，不仅使主逻辑更加简洁，还提高了代码的可读性。当其他开发者查看此函数时，他们可以迅速理解其用途，而无需深入到细节

1.4.3.4. 优化遍历

> 在 `CalculateWeight` 函数中，我引入了一个布尔标志 `checkGreater`。这个标志的作用是判断当前计算的是大于当前元素的权重之和，还是小于当前元素的权重之和。这样设计的目的是为了避免对每个元素执行两次完全遍历，从而提高了效率。这是一个细节优化，但在大数据集上可能会带来显著的性能提升

1.5. 算法实现

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

using namespace std;

// WeightedMedianCalculator类的目的是找出给定数组的带权中位数
class WeightedMedianCalculator {
public:
    // 构造函数，初始化元素数量，元素值和权重
    // len: 数组长度
    // val: 各元素的数值
    // wght: 各元素的权重
    WeightedMedianCalculator(int len, vector<int> val, vector<double>
wght)
        : length(len), numbers(val), weights(wght), idx(0) {}

    // 主要的公开接口，从第一个元素开始，逐一判断每个元素是否为带权中位数
    void Calculate() {
        do {
            ProcessCurrentIndex();  // 对当前元素进行处理
            idx++;                  // 移至下一个元素
        } while (idx < length);     // 直到处理完所有元素
    }

private:
    int length;                 // 总的元素数量
    vector<int> numbers;        // 存储各元素的数值
    vector<double> weights;  // 存储各元素的权重
    int idx;                    // 当前处理的元素索引

    // 对特定索引位置的元素进行处理
    void ProcessCurrentIndex() {
        double wgtMore = CalculateWeight(true);    // 获取所有大于当前元素的权
重总和
        double wgtLess = CalculateWeight(false);  // 获取所有小于当前元素的权
重总和

        // 根据权重的值判断当前元素是否为带权中位数
        switch (CheckWeights(wgtMore, wgtLess)) {
        case 0: // 权重不符合中位数条件
            break;
        case 1: // 当前元素是带权中位数
            cout << numbers[idx] << endl;
            break;
        default: // 默认情况，权重不符合中位数条件
            break;
```

```
        }
    }

    // 计算大于或小于当前元素的权重总和
    // checkGreater: 为true时计算大于当前元素的权重，为false时计算小于当前元素的权
重
    double CalculateWeight(bool checkGreater) {
        double wgt = 0.0;
        for (int i = 0; i < length; i++) {
            // 根据checkGreater标志，选择增加大于或小于当前元素的权重
            if ((checkGreater && numbers[i] > numbers[idx]) ||
(!checkGreater && numbers[i] < numbers[idx])) {
                wgt += weights[i];
            }
        }
        return wgt;
    }

    // 根据给定的权重值，判断是否满足带权中位数的条件
    int CheckWeights(double more, double less) {
        // 如果大于和小于当前元素的权重都小于或等于0.5，则返回1，表示是带权中位数
        if (more <= 0.5 && less <= 0.5) {
            return 1;
        }
        return 0;  // 否则返回0，表示不是带权中位数
    }
};

// 主函数
int main() {
    int n;  // 待输入的元素数量
    cin >> n;

    vector<int> values(n);      // 存储n个元素值的数组
    vector<double> w(n);       // 存储n个元素权重的数组

    // 从输入中读取n个元素的值
    int i = 0;
    do {
        cin >> values[i];
        i++;
    } while (i < n);

    // 从输入中读取n个元素的权重
    i = 0;
    do {
        cin >> w[i];
        i++;
    } while (i < n);

    // 创建WeightedMedianCalculator对象
    WeightedMedianCalculator calc(n, values, w);
    calc.Calculate();  // 执行计算

    return 0;
}
```

1.6. 运行结果

```
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.90.1-microsoft-standard-
WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Oct 28 13:28:30 CST 2023

  System load:  0.23               Processes:             117
  Usage of /:   1.8% of 250.92GB   Users logged in:       0
  Memory usage: 4%                 IPv4 address for eth0: 172.17.211.221
  Swap usage:   0%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how
MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster
deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

30 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm



This message is shown once a day. To disable it please create the
/home/dann_hiroaki/.hushlogin file.
dann_hiroaki@DESKTOP-QANEDCT:~$ gedit a.cpp
^C
dann_hiroaki@DESKTOP-QANEDCT:~$ g++ a.cpp
dann_hiroaki@DESKTOP-QANEDCT:~$ ./a.out
10
13825 28995 18417 92445 86407 90546 46896 14757 89837 18252
0.08720404 0.09585595 0.12070109 0.02521966 0.01221968 0.12648725
0.13076193 0.19128049 0.15673069 0.05353921
28995
```

1.7. 其他：另一种实现(提交的版本)

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

using namespace std;

// 函数名称：WeightMedian
// 功能：递归地判断并查找带权中位数
```

```cpp
// 参数：
// len - vals 和 wts 数组的长度
// vals - 存储数值的数组
// wts - 存储相对应数值的权重的数组
// idx - 当前正在处理的数组索引位置
void WeightMedian(int len, vector<int> vals, vector<double> wts, int idx)
{
    // 递归终止条件：如果索引超出数组长度，则结束
    if (idx >= len) return;

    // 初始化大于和小于当前值的权重和
    double wtAbove = 0.0, wtBelow = 0.0;

    // 获取当前索引处的值
    int currentVal = vals[idx];

    // 遍历整个数组，计算权重
    for (int i = 0; i < len; ++i) {
        if (vals[i] > currentVal) wtAbove += wts[i];         // 如果数组中的值大于当前值，将其权重添加到wtAbove
        else if (vals[i] < currentVal) wtBelow += wts[i];    // 如果数组中的值小于当前值，将其权重添加到wtBelow
    }

    // 检查是否满足带权中位数的条件
    if (wtAbove <= 0.5 && wtBelow <= 0.5) {
        cout << currentVal << endl;  // 输出当前值作为带权中位数
        return;
    }

    // 如果当前值不满足条件，递归处理下一个数组值
    WeightMedian(len, vals, wts, idx + 1);
}
```