**暨南大学本科实验报告专用纸**

课程名称 算法分析与设计实验 成绩评定

实验项目名称 众数 指导教师 李展 实验地点 N116

实验项目编号 003 实验项目类型 综合型

学生姓名 张瑞鹏 学号 2020101124

学院 信息科学技术学院 系 专业

实验时间 2022 年 月 日 午～ 月 日 下 午

1. **问题描述**

★问题描述：给定含有n个元素的多重集合S，每个元素在S中出现的次数称为该元素的重数。多重集S中重数最大的元素称为众数。

例如，S={1，2，2，2，3，5}。多重集S的众数是2，其重数为3。

★算法设计：对于给定的有n个自然数组成的多重集S，计算S的众数及其重数。

★数据输入：输入数据由文件名为input.txt的文本文件提供。文件的第1行为多重集S中元素个数n；在接下来的n行中，每行有一个自然数。

★结果输出：将计算结果输出到文件output.txt。输出文件有2行，第1行是众数，第2行是重数。

输入文件示例

input.txt

6

1

2

2

2

3

5

输出文件示例

output.txt

2

3

**（二）算法思路**（用文字简单说明）

求众数（快速排序的思想）

本算法的功能是不排序，找出数组的众数。如果所有数字的格式全部是1，输出没有众数，如果有多个众数，全部输出。

**为此建立两种数据结构：  
1.mode ：调用partition分割函数时，返回的中枢，包括了中枢所在位置，中枢value的大小，中枢重数是多少。**

**2.modelist ：找众数函数findmode很有可能返回的不仅仅是一个众数，很有可能是几个众数，于是定义了这样一个结构体，其中包含了一个数组list，其中放了所有众数，listnums是数组的大小，times是众数的重数。**

快速排序是将枢纽放在一个位置，该位置往左全部是小于等于枢纽的，该位置往右是大于等于枢纽的。

首先：找众数函数先调用partition函数

通过改写快速排序算法的partiton部分，把等于的中枢的元素都删除，并且记录其个数，然后中枢往左的都是比中枢小的，中枢往右的都是比中枢大的。Return值为一个结构体，记录中枢最终位置，中枢大小，已经重数。这个部分写了两个版本：调用partition的数组的会改变大小的，因为与中枢的值相同的值全部被删除，这样的话在后面舍弃部分救恩那个更加精确，但是由于考虑到移位会让函数算法复杂度增加，于是后来取消了删除移位）

舍弃：

回到找众数函数，利用partition的结果，通过砍掉一些部分来提高算法的效率。因为中枢是记录了个数的，如果中枢的个数大于左边元素的总个数，那么左边元素的必然不可能存在众数，直接左边一整段全部舍弃。右边也一样。

递归：

将左部分与右部分通过舍弃步骤能满足上述条件的的序列继续递归求众数，这样的就会返回fromleft，fromright两组众数，也有可能是只返回一组，也可能一组的不返回。此时就有fromleft,fromright,midd三组元素，其中midd表示的是中枢，这里运用一下贪心的思想，先用比较fromleft和midd，哪个重数大，把fromleft更新成哪一个，然后比较fromright和fromleft，哪个重数大，把fromleft更新成哪一个。补充：若相同，表示有多个众数，添加到fromleft中。这样最后结果就能统一用return fromleft返回。

1. **算法实施步骤和流程**（伪代码/流程图等方式描述）

mode parttition(int\* arry, int &start, int &end) {//基于快排思想的集中排序//start end发生改动

int left = start; int right = end; mode result;//定义结构体用于返回

result.times = 1;

result.val = arry[start];//记录开头的元素 这个元素为本次排序的枢纽

while (left<right)

{

while (left < right && result.val < arry[right])right--;

if (arry[left] > arry[right]) arry[left] = arry[right];

else if (arry[left] == arry[right]&& left < right){

result.times += 1;left += 1;

}

while (left < right && result.val >arry[left])left++;

if (arry[left] > arry[right]) arry[right] = arry[left];

else if (arry[left] == arry[right]&& left < right) {

result.times += 1;right -= 1;

}

}

result.place = left;

arry[left] = result.val;

return result;//result 包含中枢的数值大小，且包含中枢的个数

}

modelist FandMode(int\* arry, int start, int end) {

mode midd= parttition(arry, start, end);//引用 开始与结束会被改变

modelist fromleft; modelist fromright;

if (((midd.place - start >= midd.times)&&(midd.place-start>1)) || ((end - midd.place >= midd.times)&&(end - midd.place>1)))

{

if (midd.place - start >= midd.times && midd.place - start != 1) {

fromleft = FandMode(arry, start, midd.place - 1);

if (fromleft.times == midd.times) {//归一化

fromleft.resultsize += 1;

fromleft.result = (int\*)realloc(fromleft.result, (fromleft.resultsize+1) \* sizeof(int));

fromleft.result[fromleft.resultsize - 1] = midd.val;//把它加到里面

}

else if (fromleft.times < midd.times) {

fromleft.resultsize = 1;

fromleft.times = midd.times;

fromleft.result = (int\*)realloc(fromleft.result, 1 \* sizeof(int));

fromleft.result[0] = midd.val;//把它加到里面

}

}

else {

fromleft.resultsize = 1;

fromleft.times = midd.times;

fromleft.result = (int\*)malloc(1\*sizeof(int));

fromleft.result[0] = midd.val;

}

if (end - midd.place >= midd.times && end - midd.place != 1) {

fromright = FandMode(arry, midd.place + 1, end);

if (fromleft.times == fromright.times) {//归一化

int newsize = fromleft.resultsize = fromleft.resultsize + fromright.resultsize;

fromleft.result = (int\*)realloc(fromleft.result, newsize \* sizeof(int));

for (int i = newsize - fromright.resultsize, j = 0; i < fromleft.resultsize; i++, j++) {

fromleft.result[i] = fromright.result[j];//把它加到里面

}

}

else if (fromleft.times < fromright.times) {

fromleft = fromright;//直接覆盖

}

}//最终都在数组fromleft对象之中

}

else {

fromleft.resultsize = 1;

fromleft.times = midd.times;

fromleft.result = (int\*)malloc(1\*sizeof(int));

fromleft.result[0] = midd.val;

}

return fromleft;

}

1. **源代码**（通过了编译运行的正确程序）

#include<stdio.h>

#include<stdlib.h>

#include<iostream>

using namespace std;

struct mode {

int val;//众数值得大小

int times;//众数出现的次数

int place;//众数所在的位置

};

struct modelist {//众数列表

int times;//他们的重数

int \*result;//存值的大小的数组

int resultsize;//数组本身的的大小

};

mode parttition(int\* arry, int &start, int &end) {//基于快排思想的集中排序//start end发生改动

int left = start; int right = end; mode result;//定义结构体用于返回

result.times = 1;

result.val = arry[start];//记录开头的元素 这个元素为本次排序的枢纽

while (left<right)

{

while (left < right && result.val < arry[right])right--;

if (arry[left] > arry[right]) arry[left] = arry[right];

else if (arry[left] == arry[right]&& left < right){

result.times += 1;left += 1;

}

while (left < right && result.val >arry[left])left++;

if (arry[left] > arry[right]) arry[right] = arry[left];

else if (arry[left] == arry[right]&& left < right) {

result.times += 1;right -= 1;

}

}

result.place = left;

arry[left] = result.val;

return result;//result 包含中枢的数值大小，且包含中枢的个数

}

modelist FandMode(int\* arry, int start, int end) {

mode midd= parttition(arry, start, end);//引用 开始与结束会被改变

modelist fromleft; modelist fromright;

if (((midd.place - start >= midd.times)&&(midd.place-start>1)) || ((end - midd.place >= midd.times)&&(end - midd.place>1)))

{

if (midd.place - start >= midd.times && midd.place - start != 1) {

fromleft = FandMode(arry, start, midd.place - 1);

if (fromleft.times == midd.times) {//归一化

fromleft.resultsize += 1;

fromleft.result = (int\*)realloc(fromleft.result, (fromleft.resultsize+1) \* sizeof(int));

fromleft.result[fromleft.resultsize - 1] = midd.val;//把它加到里面

}

else if (fromleft.times < midd.times) {

fromleft.resultsize = 1;

fromleft.times = midd.times;

fromleft.result = (int\*)realloc(fromleft.result, 1 \* sizeof(int));

fromleft.result[0] = midd.val;//把它加到里面

}

}

else {

fromleft.resultsize = 1;

fromleft.times = midd.times;

fromleft.result = (int\*)malloc(1\*sizeof(int));

fromleft.result[0] = midd.val;

}

if (end - midd.place >= midd.times && end - midd.place != 1) {

fromright = FandMode(arry, midd.place + 1, end);

if (fromleft.times == fromright.times) {//归一化

int newsize = fromleft.resultsize = fromleft.resultsize + fromright.resultsize;

fromleft.result = (int\*)realloc(fromleft.result, newsize \* sizeof(int));

for (int i = newsize - fromright.resultsize, j = 0; i < fromleft.resultsize; i++, j++) {

fromleft.result[i] = fromright.result[j];//把它加到里面

}

}

else if (fromleft.times < fromright.times) {

fromleft = fromright;//直接覆盖

}

}//最终都在数组fromleft对象之中

}

else {

fromleft.resultsize = 1;

fromleft.times = midd.times;

fromleft.result = (int\*)malloc(1\*sizeof(int));

fromleft.result[0] = midd.val;

}

return fromleft;

}

int main(void) {

int a[13] = {2,6,4,5,87,6,6,36,16,16,6,16,6};

int start = 0;

int end = 12;

modelist re = FandMode(a, start, end);

if (re.times > 1) {

printf("众数如下：");

for (int i = 0; i < re.resultsize; i++) {

printf("%d ", re.result[i]);

}

printf("\n");

printf("重数是：%d", re.times);

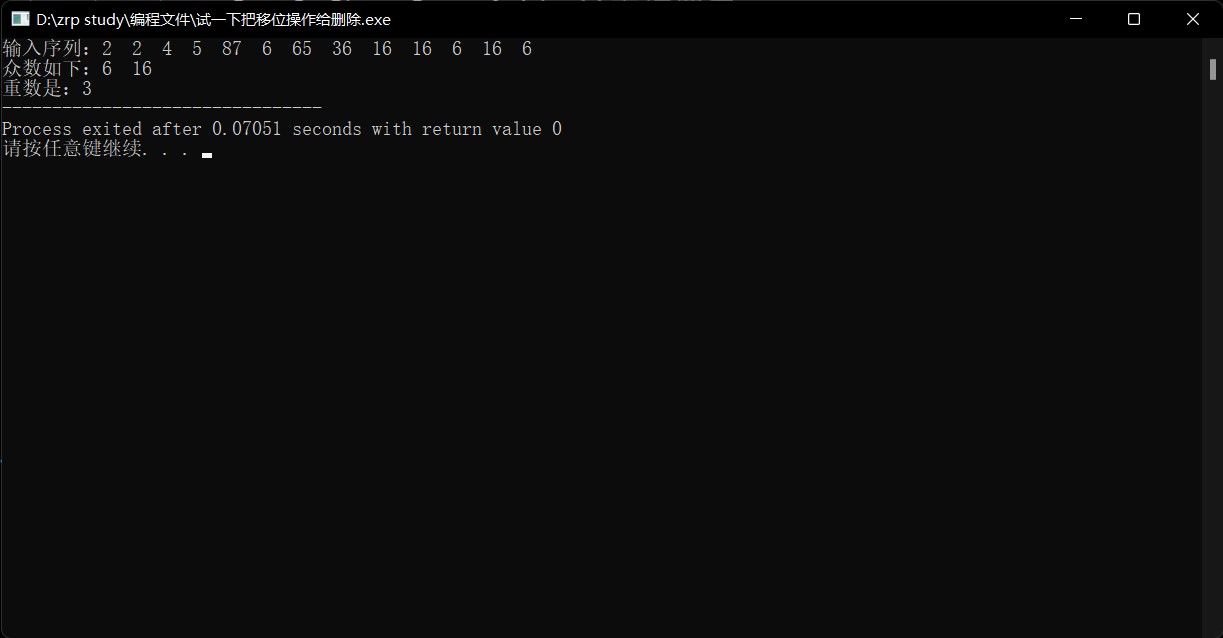
}

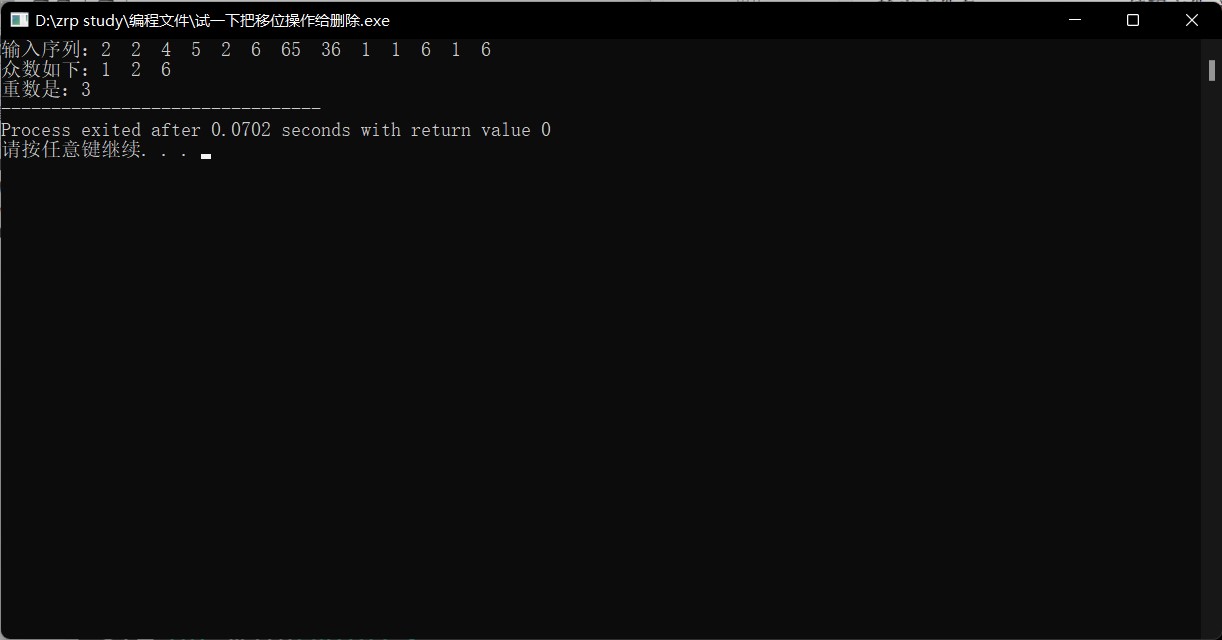
else printf("没有众数！！");

return 0;

}

1. **测试结果**（至少有两个以上算例及程序运行结果，截图贴进实验报告）



****

**（六）实验总结**（至少三句话，可以写复杂度分析、遇到问题、可能的改进措施、心得体会等）

1.如果使用分治法求解，算法复杂度最坏不会超过n^2。有可能第一个数就找到了，而且左边与右边的元素总个数还众数的个数，因此直接被舍弃。这种就是最好的情况，但是出现的概率较低。因为其中加入了舍弃的部分，不能完全按照一般的分治法来求得一个准确是时间复杂度。

2.由于代码使用c语言实现，很多操作不如c++那么简洁，但由于个人对c++不是太熟悉，很多操作运用的不好，暂时还是用c来完成。

3.一开始听到不排序就得到结果感觉不可思议，后来想到快速排序可以稍作更改，把相同的元素集中到中间，但是这似乎还需要移动很多的步数，使得时间复杂度变高。在到后来才思考到可以通过定义数据结构来完成内容的返回。

**暨南大学本科实验报告专用纸(附页)**