寻找多数元素

1. 问题重述

在一个整数序列A[1…n]中,如果整数a在A中出现的次数多于，那么称a为A中的多数元素。作业的任务就是要设计一种算法寻找多数元素。

1. 算法介绍

由题目可以知道，多数元素表示在一个数组中出现次数最多，并且出现次数大于数组元素总个数的一半的元素。解决这个问题可以有很多种方法。

（1）蛮力搜索法

这种方法把列表中的每个元素进行遍历，统计每个元素出现的次数。如果某个元素的计数大于，就可以断言它是多数元素；否则，在序列中就没有多数元素。但是利用这样的算法寻找需要进行的比较次数是

代价太过昂贵，如果先排序，再按上述方法进行计算，则最坏情况下的时间复杂度仍为。

（2）中间元素法

由多数元素的定义可以知道，在经过排序的序列中，若存在多数元素，则中间元素（即第个元素）一定是多数元素。选择中间元素后可以再次扫描序列，确定中间元素是否确实是多数元素。

在此算法中，用于排序的时间复杂度为，而且中间元素可以在时间内找到，故此算法将花费的时间。但这种算法也有其劣势，因为寻找中项算法的时间复杂性中的隐含常数太大，并且它要求先对数组进行排序，在数组长度较大的时候，其计算代价也是较大的。

（3）剔除元素法

这是一个很漂亮的解法，它用到的比较次数要少得多。这个算法主要依赖以下原理：

在原序列中去除两个不同的元素后，那么在原序列中的多数元素在新序列中仍为多数元素。

这样原问题就简单了，我们只需要将数组中不相同的元素两两剔除，那么剩下的数就自然是多数元素。下面介绍该算法的过程。

首先将数组的第一个元素设置为多数元素候选c，以及计数器count = 1，一开始令c=A[1]，从A[2]开始将后面的数与c比较，若相等则count+1，不相等count-1。如果所有元素都已经扫描完毕并且计数器大于0，那么返回c作为多数元素的候选者。如果count减到0，便表示该元素出现次数太少，将其和后面一个不相等的元素剔除，再次重置c和count，不断执行上述操作，直到比较到最后count都不为0，则c就为候选的多数元素。最后再统计序列中候选者c出现的次数，以最终确定c是否为多数元素。

选择此种方法进行编程，算法的输入为n个元素的数组A[1…n]，算法输出为数组中的多数元素（若多数元素存在），当不存在多数元素时，输出None。

下面展示其递归实现的伪代码：

1. 编程实现

本次作业采用C++作为编程语言。下面先给出 函数的源代码：

**int candidate(int m)**

**{**

**int j, c, count;**

**j = m; c = A[m]; count = 1;**

**while (j < n&&count>0)**

**{**

**j++;**

**if (A[j] == c)**

**count++;**

**else**

**count--;**

**}**

**if (j == n)**

**return c;**

**else**

**return candidate(j + 1);**

**}**

下面给出main函数对其进行调用的代码：

**int main()**

**{**

**cout << "请输入数组中的元素个数:";**

**cin >> n;**

**A = new int[n + 1];**

**cout << "下面请输入n个元素。" << endl;**

**for (int i = 1; i <= n; i++)**

**{**

**cout <<"第"<<i<<"个元素是:";**

**cin >> A[i];**

**}**

**int c, count = 0;**

**c = candidate(1);**

**for (int j = 1; j <= n; j++)**

**{**

**if (A[j] == c)**

**count++;**

**}**

**if (count > (n / 2))**

**cout << "数组中的多数元素是:"<<c<< endl;**

**else**

**cout << "该数组中没有多数元素！";**

**delete A;**

**return 0;}**

1. 案例分析与算法检验

1.案例分析

为了验证算法的正确性，采用一个案例介绍算法的执行过程，并查看最终的输出。

以课本上的例子A=[1,3,2,3,3,4,3]为例，详细分析算法的迭代过程。

在此例中，序列的长度n=7。算法运行时执行主函数，运行至第九条语句时跳转到candidate函数。

首先执行candidate(1)

j=1,c=1,count=1.

j=2,A[2]c,count=0(计数器为0，进行元素剔除),return candidate(3).

下面执行candidate(3)

j=3,c=2,count=1.

j=4,A[4]c,count=0(计数器为0，进行元素剔除),return candidate(5).

然后执行candidate(5)

j=5,c=3,count=1.

j=6,A[6]c,count=0(计数器为0，进行元素剔除),return candidate(7).

最后执行candidate(7)

j=7,c=3,count=1.

j=n=7,return 3.

程序从这退出candidate函数，进入主函数，在主函数里验证返回值在序列A中出现的次数，发现在序列长度为7的情况下，3这个数字出现了4次，符合多数元素的定义，程序最终成功输出。

下面给出程序运行截图。



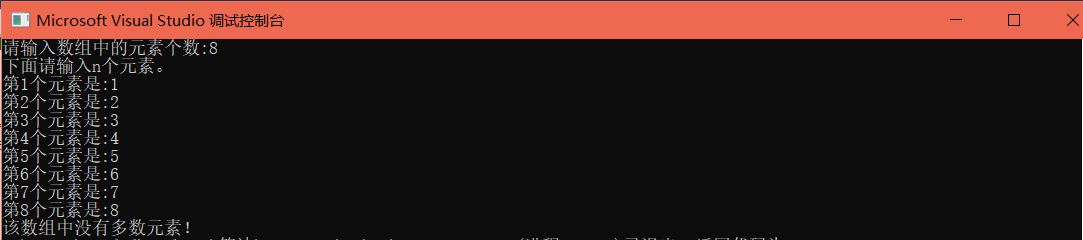
看到算法输出与预想相符合。

从以上的分析过程可以看到，算法如同介绍中一样，每次删除两个不相同的元素，最后把剩下的元素进行输出。在主函数中对candidate函数的输出进行验证，看其是否满足多数元素的条件，如果满足条件，则将其按照多数元素输出。如果不满足条件，则证明原序列不存在多数元素。

2.算法检验

上面以一个序列为例详细介绍了算法的执行过程，为了验证算法在不同长度序列、有无多数元素序列上的通用性，下面用不同的实例进行验证。

①以数列A=[1，2，3，4，5，6，7，8]为例，数组长度为8，不存在多数元素且每个元素各不相同，将其输入到所编写的程序。运行结果如下图所示。



可以看到程序返回了正确的结果。

②以数列A=[1，1，1，1，2，2，2，2]为例，数组长度为8，不存在多数元素但存在两个出现次数为的元素，将其输入到所编写的程序。运行结果如下图所示。

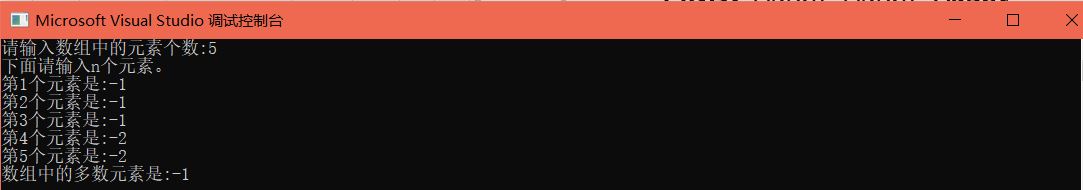
程序依然给出了正确的输出。

③以数列A=[1，1，1，1，2，2，2，2，2]为例，数组长度为9，存在多数元素但多数元素出现的次数仅比非多数元素多一次，将其输入到所编写的程序。运行结果如下图所示。



输出还是正确的。

④以数列A=[-1，-1，-1，-2，-2]为例，数组长度为5，存在多数元素且数组中元素全为负数，将其输入到所编写的程序。运行结果如下图所示。



可见，函数没有受数组内元素符号的干扰，给出了正确的结果。

这样一个数组一个数组不断输入的验证方式过于繁琐，为了验证程序在大量输入数据下的稳定性，采用随机的方法，每次随机产生数组长度，数组中的各个元素也为随机生成。为了计算准确率，我们把candidate算法计算的结果同蛮力搜索法计算的结果进行比较，如果结果相同，则判定为计算准确。对计算正确情况进行计数，最后把计算正确个数与总试验个数相比，如果比值等于1，则说明算法每次都得到了正确的结果，通过验证。

下面给出蛮力搜索法求多数元素的函数。

**int manli(int \*D, int nn) {**

**int \*B;**

**int maxnum = 0;**

**int count2 = 0;**

**B = new int[nn] {0};**

**for (int k = 0; k < nn; k++) {**

**int c = D[k];**

**for (int j = 0; j < nn; j++) {**

**if (c == D[j])**

**{**

**B[k]++;**

**}**

**}**

**}**

**for (int q = 0; q < nn; q++) {**

**if (B[q] > B[maxnum]) {**

**maxnum = q;**

**}**

**}**

**int cc = D[maxnum];**

**for (int j = 0; j < nn; j++)**

**{**

**if (A[j] == cc)**

**count2++;**

**}**

**if (count2 > (nn / 2))**

**{**

**return cc;**

**}**

**else {**

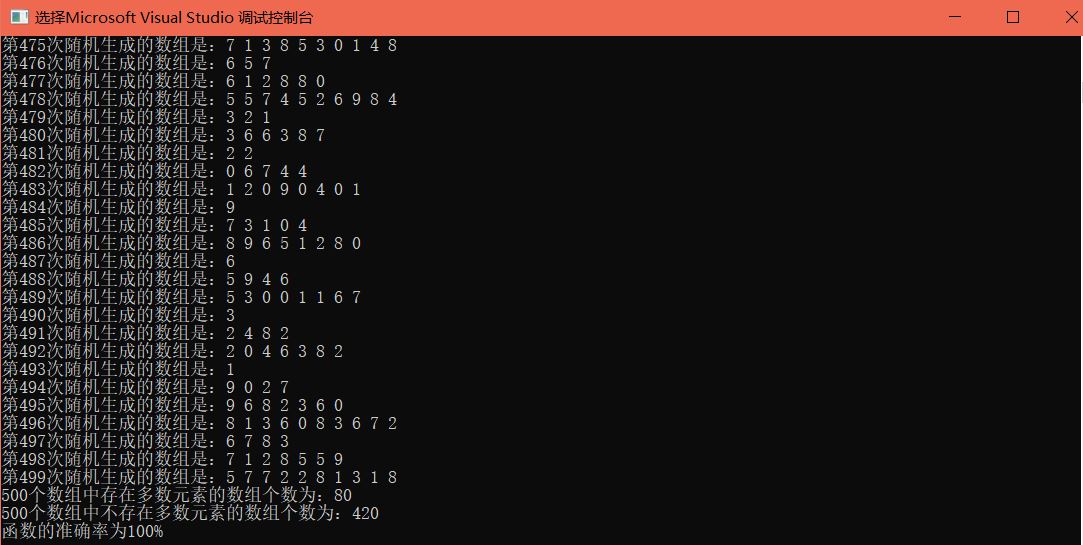
**return 99999999;**

**}**

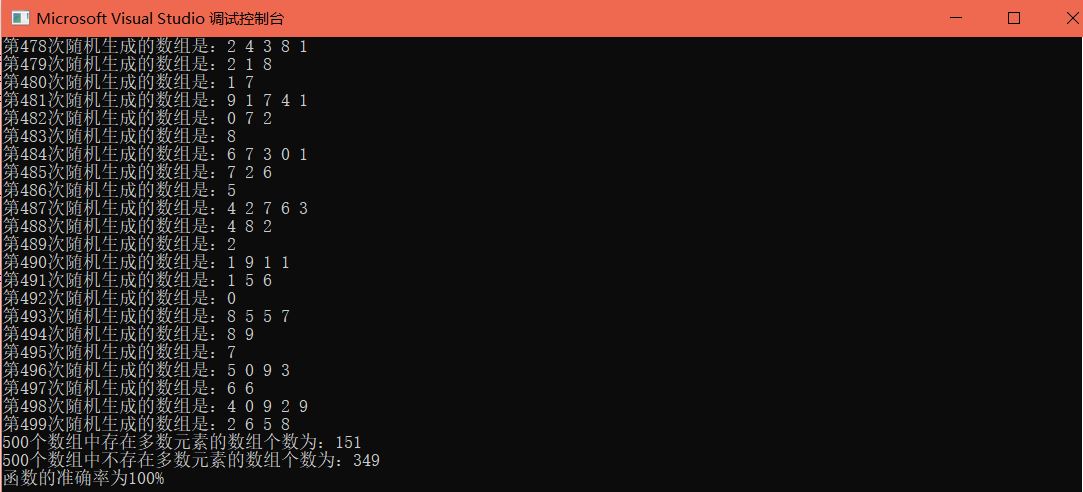
**}**

经过分析和验证，此函数可以准确求得一个数组中的多数元素。为了简便，当数组中不存在多数元素时，函数返回99999999。此时只要保证原数组中不含99999999，就可认为此函数返回正确结果。

下面需要对源代码进行更改。引入随机函数，每次随机产生数组长度n,数组中的每个元素也随机产生。设置数组长度n的随机取值范围是1到10，设置数组中的每个元素的取值范围是0到9，设定测试次数为500次。更改后的源代码在附录中给出。运行结果如下图所示。

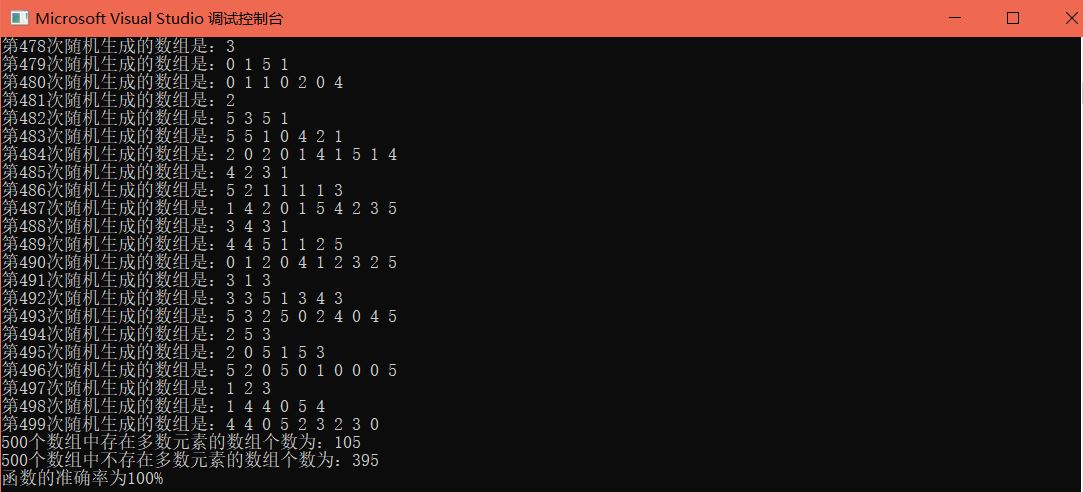


从上图看出，candidate函数计算多数元素的准确率不受输入数组长度和性质不同的影响，每次测试都成功返回了正确的输出。

由常识可知，当数组内各个元素的取值范围不变而数组长度变小时，该数组存在多数元素的可能性变大。下面进行验证，将数组长度n的随机取值范围缩小到1到5之间，再次查看输出结果，如下图所示。

从上图的运行结果可以看到，在500个数组中存在多数元素的数组个数较上一次具有较大幅度的提高,与常识相符。

又想到当数组长度的取值范围不变，数组内各个元素的取值范围缩小时，数组存在多数元素的可能性也变大。下面进行验证，数组长度n的取值范围与第一次一样设置为1到10，而数组内各个元素的取值范围从0到9缩小为0到5，再次查看程序输出，如下图所示。



可以看到500个随机生成的数组中存在多数元素的数组个数也有了一定程度的提高。

经过以上的验证过程，可以确定利用剔除元素法来求解数组所含多数元素的方法是正确的，且具有很好的鲁棒性。

五.复杂度分析

（一）时间复杂度

观察函数可以看到，此函数最终需要将数组内的每个数都遍历一遍，所用时间复杂度为。而在主程序中，循环执行了次，所用时间也为， 故总体的时间复杂度为：。

（二）空间复杂度

观察源代码可以看到，空间复杂度为 。对于查找多数元素这一问题来说已为最优情况。

随机生成数组进行验证的代码：

#include "pch.h"

#include <iostream>

#include <cstdlib>

#include <ctime>

#include<string>

#include <algorithm>

#define random(a,b) (rand() % (b - a + 1)) + a

using namespace std;

int manli(int\*A, int n);

int candidate(int);

int \*A;

int n;

int duile = 0;

int wuduoshu = 0;

int youduoshu = 0;

string outt = "该数组中没有多数元素！";

int main()

{

for (int k = 0; k < 500; k++) {

srand((int)time(0)+k);

n = random(1,10);

A = new int[n];

cout << "第" << k << "次随机生成的数组是：";

for (int kk = 0; kk < n; kk++) {

A[kk] = random(0, 5);

cout << A[kk] << ' ';

}

cout << "\n";

int man = manli(A, n);

int c, count = 0;

c = candidate(0);

for (int j = 0; j < n; j++)

{

if (A[j] == c)

count++;

}

if (count > (n / 2))

{

if (c == man) {

duile += 1;

youduoshu += 1;

}

}

else

if (man == 99999999) {

duile += 1;

wuduoshu += 1;

}

delete[] A;

}

double acc = duile / 500.0;

cout << "500个数组中存在多数元素的数组个数为："<<youduoshu<<endl;

cout << "500个数组中不存在多数元素的数组个数为：" << wuduoshu << endl;

cout <<"函数的准确率为"<< acc\*100 <<"%"<< endl;

return 0;

}

int manli(int \*D, int nn) {

int \*B;

int maxnum = 0;

int count2 = 0;

B = new int[nn] {0};

for (int k = 0; k < nn; k++) {

int c = D[k];

for (int j = 0; j < nn; j++) {

if (c == D[j])

{

B[k]++;

}

}

}

for (int q = 0; q < nn; q++) {

if (B[q] > B[maxnum]) {

maxnum = q;

}

}

int cc = D[maxnum];

for (int j = 0; j < nn; j++)

{

if (A[j] == cc)

count2++;

}

if (count2 > (nn/2))

{

return cc;

}

else {

return 99999999;

}

}

int candidate(int m)

{

int j, c, count;

j = m; c = A[m]; count = 1;

while (j < n&&count>0)

{

j++;

if (A[j] == c)

count++;

else

count--;

}

if (j == n)

return c;

else

return candidate(j + 1);

}