**数据结构实验报告**

姓名：郭宇瑶 学号：U201717031 班级：软工1704班

**实验二：power（x,n）求幂问题、教材2.19求主元问题的求解**

一、问题描述

用基于2和3的方式分别写出算法来求power（x,n）。分析两种算法的复杂程度，设计实验来验证想法。

基于教材（《数据结构与算法分析·c语言描述》）中的2.19题，设计并实现用分治求数组的主元的算法。如果不用分治，通过比较和计数，求复杂程度。

二、问题分析与算法设计

1.power（x,n）求幂问题：

若按常规方法计算x的n次方，需要进行n-1次乘法运算，效率低下。现在考虑将power(x,2)的结果保留到中间变量中(以2为底的方法)，则power(x,4)=power(x,2)\*power(x,2),只需两次乘法运算。再将power(x,4)的结果保留，则power(x,8)=power(x,4)\*power(x,4)······按此思路，用尾递归的方法解题，在int power(x,n)函数中，若n为奇数，则返回power(x,n/2)\*x,若n为偶数，则返回power(x,n/2)，这样每经过一次递归调用，问题规模减半，预计算法时间复杂度为O（log2n）。以3为底的方法同理，按n%3为0,1,2分类，每次递归调用问题规模缩小至1/3，预计算法时间复杂度为O（log3n）。

2.教材2.19求主元问题：

问题描述（教材原文）：

大小为N的数组A，其主要元素是一个出现次数超过N/2的元素（从而这样的元素最多有一个）。例如，数组3,3,4,2,4,4,2,4,4有一个主要元素4，而数组3,3,4,2,4,4,2,4没有主要元素。如果没有主要元素，那么你的程序应该指出来。下面是求解该问题的一个算法的概要：

首先，找出主要元素的一个候选元（这是难点）。这个候选元是唯一有可能是主要元素的元素。第二步确定是否该候选元实际上就是主要元素，这正好是对数组的顺序搜索。为找出数组A的一个候选元，构造第二个数组B。比较A[1]和A[2]，如果它们相等，则取其中之一加到数组B中；否则什么也不做。以该方式继续下去直到读完这个数组。然后，递归地寻找B中的候选元；它也是A的候选元。

非分治算法：设置两重循环，将数组内的每一个元素与所有元素比较，若相同，则count++,若count>n/2,则该元素为主元，返回该元素。若循环结束还未满足条件，则返回0，表示没有找到主元。预计算法时间复杂度为O（n2）.

分治算法：采用递归，构造第二个数组B。比较A1和A2，若他们相等，则取其一加入B中，否则什么也不做。以该方式继续下去直到读完整个数组。然后对B数组重复上述操作。预计算法时间复杂度为O（n）。

三、实验实现过程

1. power（x,n）求幂问题：

（1）定义方法int power(int x,int n) //利用递归方法求解；

（2）变量n取50,100,500，1000,5000，8000,10000,12000,20000，80000十组数据.将求值函数体循环1000000次，在函数执行前，定义time\_t类型的变量start=clock(); 函数执行后，加上代码：end=clock()；打印（int）(end-start) 记录函数结束时间；

（3）重复5 次实验，得到平均数据；

（4）记录相应10组实验结果并绘制相应Excel图表。

1. 求主元问题：
2. 分别定义分治法与非分治法的函数；
3. 给定五组测试数据：int a1[5] = { 1,1,2,2,1 };//1

int a2[8] = { 3,3,4,2,4,4,4,4 };//4

int a3[10] = { 1,4,4,2,3,4,4,3,3,4 };//无

int a4[12] = { 3,1,2,2,3,3,3,3,3,2,3,4};//3

int a5[15] = { 1,4,4,2,3,4,3,3,4,2,4,4,2,4,4 };//4

将求值函数体循环1000000次，在函数执行前，定义time\_t类型的变量start=clock(); 函数执行后，加上代码：end=clock()；打印（int）(end-start） 记录函数结束时间；利用非分治法的结果检验分治法是否正确，不正确返回-1.

（3）重复5 次实验，得到平均数据；

（4）记录相应3组实验结果并绘制相应Excel图表。

四、测试结果与分析

1.power（x,n）求幂问题：

（1）基于2的power（x,n）

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 50 | 100 | 500 | 1000 | 5000 | 8000 | 10000 | 12000 | 20000 | 80000 |
| 平均时间(clock) | 190.2 | 232.8 | 302.6 | 301.4 | 371.6 | 356.2 | 385.2 | 358.2 | 371.2 | 395.8 |

（2）基于3的power（x,n）

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 50 | 100 | 500 | 1000 | 5000 | 8000 | 10000 | 12000 | 20000 | 80000 |
| 平均(clock) | 125.6 | 149 | 180.2 | 214 | 265.8 | 295 | 291 | 276.4 | 306 | 321.8 |

2.求主元问题

（1）分治法

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 5 | 8 | 10 | 12 | 15 |
| 平均(clock) | 181.2 | 211 | 244.6 | 286.8 | 328.8 |

（2）非分治法

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 5 | 8 | 10 | 12 | 15 |
| 平均(clock) | 123.8 | 165.8 | 402.8 | 621.2 | 1173.6 |

1. 实验结论

power（x,n）求幂问题得出的n-t曲线近似为对数曲线，以2为底、以3为底的时间复杂度分别为O（log2N）与O（log3N），与预期相符。

教材2.19求主元问题所使用的五种测试用例均得出正确结果，非分治算法时间复杂度为O（N2）,分治算法时间复杂度为O（N）

1. 代码

//以2为底

int power(int x, int n)

{

if (0 == n)

return (1);

if (0==n%2)//n是偶数

{

return (power(x\*x, n /2) );

}

return (power(x\*x, n /2 )\*x);//n是奇数

}

//以3为底

int power3(int x, int n)

{

if (0 == n)

return 1;

if (1 == n)

return x;

if (2 == n)

return x\*x;

if (0 == n % 3)

{

return power3(x\*x\*x, n / 3);

}

if (1 == n % 3)

{

return power3(x\*x\*x, n / 3)\*x;

}

return power3(x\*x\*x, n / 3)\*x\*x;

}

//主元

int findMainElement(int \*array, int n) {

int length = 0;

if (1 == n)

return array[0];

else if (1 != n)

{

if (n & 1)//array长度为奇数

{

// const int size = n / 2 + 1;

int B[n / 2 + 1];

int j = 0;

for (int i = 0; i<n - 1; i += 2)

{

if (array[i] == array[i + 1])

B[j++] = array[i];

}

B[j] = array[n - 1];

return findMainElement(B, j + 1);

}

else

{

int B[n/2];

int j = 0;

for (int i = 0; i<n; i += 2)

{

if (array[i] == array[i + 1])

{

B[j++] = array[i];

}

return findMainElement(B, j + 1);

}

}

}else return 0;

}

//非分治法：比较和计数

int find(int \*array, int n)

{

int count = 0;

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

{

count = 0;

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

{

if (array[j] == array[i])

count++;

}

if (n & 1)//n为奇数

{

if (count>n / 2 + 1)

return array[i];

}

else if (0 == n % 2)//n为偶数

{

if (count>n / 2)

return array[i];

}

}

return 0;

}

//利用非分治法检查分治法结果正确与否

/\*\*

\*param n:分治法返回值

\*param res:非分治法返回值

\*/

int checkMainEle(int n,int res){

If(n==res)return n;

return -1;

}