**集美大学实验报告**

**班级 计算2114 学号 202121331104 姓名 庄佳强 成绩**

**课程 《算法设计与分析》 日期 2024 年 4 月 9 日**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验项目** | **实验二 分治法的应用** | **指导老师** | **杨艳华** |
| **实验目的** | 掌握分治法的使用 | | |
| **一、实验内容**  1.某石油公司有*n*口油井，为方便输送石油，计划修建输油管道。根据设计要求，水平方向有一条主管道，每口油井修一条垂直方向的支线管道通向主管道。请设计一种算法确定主管道的位置，使得所有油井到主管道之间的支线管道长度的总和最小。提示：复杂度为*O*(*n*)才能通过所有测试用例。  2.Karatsuba乘法用于实现两个大整数相乘。所谓大整数相乘是指整数比较大，相乘的结果超出了基本数据类型的表示范围，所以不能直接做乘法运算。算法的基本原理是将大整数拆分成两段后变成较小的整数，请设计分治算法说明Karatsuba乘法的原理，并实现该算法。 | | | |
| **二、问题分析**  1.要使的个个点到一条直线上的距离之和最短，当线段在点集的为中位数时可以得到。  问题被简化为求集合中的中位数，要用时间复杂度为*O*(*n*)，就不能用普通的排序法，至少为O(nlogn),只能使用计数排序才能为*O*(*n*)。  2.Karatsuba 乘法原理  既然当输入规模 n 增加时计算量会以 n^2 增加，那么考虑使用分治的方式把一个规模较大的问题分解成若干个规模较小的问题，这样解决这几个规模较小的问题就会比较容易了。  对于 x 和 y 的乘积 z = x \* y，可以把 x 和 y 都拆成两段：x 的左半段和右半段分别是 a 和 b，y 的左半段和右半段分别是 c 和 d。  现在考虑乘积 h(x) = f(x) \* g(x)：      h(x) = (a \* m + b) \* (c \* m + d)        = a \* c \* (m^2) + (a \* d + b \* c) \* m + b \* d  (a \* d + b \* c)=(a + b) \* (c + d) - a \* c - b \* d  所以用分治法求a\*c b\*d (a+b)\*(c+d)就可以求出x\*y | | | |
| **三、数据结构定义**  1.  arr[1000010]：这是一个大小为1000010的整数数组。这个数组用于存储输入的数据，特别是在后续的排序和计算中使用。该数组在程序的中间部分被填充。  a[20010]：这是一个大小为20010的整数数组。它用于统计输入数据中特定值出现的次数。具体来说，该数组被用于统计b + 10000这个值出现的次数，其中b是从输入读取的一个整数。这个数组的目的是为了对数据进行计数和排序。  2.  使用大量的string容器和int类型。 | | | |
| **四、算法伪代码描述**  **1>:**  ans -> 0  arr with size 1000010  a with size 20010  Int main()  Read n from input  For i from 0 to n-1 do i++:  Read c and b from input  Increment a[b + 10000]  Set h to 0  For j from 0 to 20009 do j++:  If a[j] is not 0:  For i from 0 to a[j]-1:  arr[h] = j - 10000  Increment h  ans -> 0  For i from 0 to n-1 do i++:  ans += absolute value of (arr[n / 2] - arr[i])  Output ans  End  **2>:**  Class Solution:  Function add(num1, num2):  carry -> 0  result ->null    设置i for num1.size - 1  设置j for num2.size - 1    While i >= 0 OR j >= 0 OR carry > 0:  设置digit1 to (i >= 0) ? (num1[i] - '0') : 0  设置digit2 to (j >= 0) ? (num2[j] - '0') : 0    设置sum to digit1 + digit2 + carry  设置carry to sum / 10  Append (sum % 10) to the beginning of result  --i  --j    Return result  Function subtract(num1, num2):  result ->null  borrow -> 0    设置i for num1.size - 1  设置j for num2.size - 1    While i >= 0:  设置digit1 to num1[i] - '0'  设置digit2 to (j >= 0) ? (num2[j] - '0') : 0    设置diff to digit1 - digit2 - borrow    If diff < 0:  Add 10 to diff  设置borrow to 1  Else:  设置borrow to 0    Append diff to the beginning of result  i--  j--    Remove leading zeros from result    If result is empty:  Return "0"  Else:  Return result  Function Karatsuba(x, y):  设置numx for num1.size  设置numy for num2.size    If numx == 0 OR numy == 0 OR x == "0" OR y == "0":  Return "0"    If x == "1":  Return y    If y == "1":  Return x    If numx <= 2 OR numy <= 2:  设置result to x \* y (using standard multiplication)  Return result    设置m to min(numx, numy) / 2  设置x1 to first part of x  设置x0 to second part of x  设置y1 to first part of y  设置y0 to second part of y    设置z2 to Karatsuba(x1, y1)  设置z0 to Karatsuba(x0, y0)  设置z1 to Karatsuba(add(x1, x0), add(y1, y0))  设置z11 to subtract(subtract(z1, z0), z2)    Return add(add(z2 + string(m \* 2, '0'), z0), z11 + string(m, '0')); | | | |
| **五、算法时间和空间复杂度分析（要求写出详细过程）**  **1>:**  **时间复杂度：**  读取统计操作中单层for循环 0（n）  对n个元素进行排序0(n)  计算操作中需要遍历nci O(n)  时间复杂度为T(n)=O(3n)=O(n)  **空间复杂度：**  Arr数组：O(n)  a数组：O(20010)=O（k）  所以，总的空间复杂度是O(n + k)   1. **:**   **时间复杂度：**  对于递归中：  T(n) = 3 \* T(n / 2) + O(n)    当 n > 1  T(n)=O(1)            当 n = 1  解递归方程得到 T(n) = O(n^log3)。  **空间复杂度：**  Karatsuba 函数中使用了栈空间，最多有logn层空间所以空间复杂度为：O(logn)。 | | | |
| **六、错误记录和解决**   1. 第一题中一开始使用快排来找中位数，但是快排为O(logn)，导致最后一个检测点一直超时。   解决：跟换排序方式，使用计数排序O(n)，使用统计找到中位数。这样可以避免超时。   1. 第二题中要计算的数字十分的大，使用long long 都没办法存下，导致没法计算。   解决：使用string 把数字转化为字符串，然后在通过对字符串进行操作，使得字符串转化回数字用于计算，最后再转化为字符串拼接。 | | | |
| **七、心得体会**  分治法是一种算法设计策略，它的核心思想是将一个大问题分解成若干个规模较小的子问题，逐个解决这些子问题，然后合并它们的解以得到原问题的解。  通过分解问题，我们可以更清晰地理解问题的结构和特性，从而设计出更优雅和高效的解决方案。  但是也需要对空间和时间进行合理的分配。 | | | |