**《算法分析与设计实验》**

**报告**

学 号： JB174007

姓 名： 程睿

年 级： 2017级

专 业： 计算机科学与技术

指导老师： 纪霞

完成日期： 2019 年 11 月

**安徽大学江淮学院**

**理工部**

## 实验名称：大整数的乘法

## 问题描述：

在某些情况下需要处理很大的整数，无法在计算机硬件能直接表示的整数范围内进行处理，若要精确地表示大整数并在计算结果中精确地得到所有位数上的数字，需要用到软件的方法来实现大整数乘积算法。

设X和Y都是n位二进制整数，现在要计算它们的乘积XY，可以用小学所学的方法来设计一个计算乘积XY的算法，但是这样做步骤太多，效率较低。如果将每两个一位数的乘法或加法看做一步运算，那么这种方法要进行O（n2）步才能求出乘积XY。

## 解决思想：

为了更好地处理大整数乘积算法，可以使用分治的方法，将两个大整数都分为2段，每段n/2位。

C:n/2位 D:n/2位

A:n/2位 B:n/2位

X = Y =

所以。所以X和Y的乘积为：



但是此时需要进行4次n/2为整数的乘法（AC、AD、BC和BD）、3次不超过2n位的整数加法（分别对应式子中的+），以及2次移位操作（分别对应式子的2n/2和2n）。所有这些加法和移位共用了O(n)步运算。T（n）可表示为：



计算出T（n） = O（n2），此时时间复杂度与直接相乘时相差不大，因此要减少乘法的次数可得：



这个式子需要做3次n/2位整数的乘法，分别是（AC、BD和（A-B）（D-C））、6次减法和2次移位操作，由此可得：



求得T（n） = O(nlog3) = O(n1.59)。

## 代码描述

#define SIGN(A) ((A > 0) ? 1 : -1)

//定义宏，判断该数是否大于0；

int divideConquer(int X, int Y, int n){

//X为第一个数，Y为第二个数，n为两整数长度。

int sign = SIGN(X) \* SIGN(Y);

//求出两数乘积项的符号；

int x = abs(X);//求X的绝对值；

int y = abs(Y);//求Y的绝对值；

if(x == 0 || y == 0){

//当两数为0时，相乘结果也为0；

return 0;

}else if(n == 1){

//当两数均为1位数，则两数绝对值相乘再与符号sign相乘；

return sign \* x \* y;

}else{

int A = (int) x / pow(10, (int)(n / 2));

//求式子中A的值，A = X/（10n/2），X的高n/2位；

int B = x - A \* pow(10, n / 2);

//求式子中B的值，B = X-A，B为X的低n/2位；

int C = (int) y / pow(10, (int)(n / 2));

//求式子中C的值，C = Y/（10n/2），Y的高n/2位；

int D = y - C \* pow(10, n / 2);

//求式子中D的值，D = Y-C，D为Y的低n/2位；

int AC = divideConquer(A, C, n / 2);

//将原问题划分为子问题，通过递归调用，对A\*C再进行大整数乘法，得到A\*C的值；

int BD = divideConquer(B, D, n / 2);

//通过划分子问题，求出B\*D的值，通过递归对B\*D进行大整数乘法；

int ABDC = divideConquer((A - B), (D - C), n / 2) + AC + BD;

//继续划分原问题为子问题，通过递归调用求出(A-B)\*(D-C)的值，则变成长度为n/2位长度的整数相乘；

return sign \* (AC \* pow(10 , n) + ABDC \* pow(10, (int)(n / 2)) + BD);

//通过得到已知的乘积项，对最后的结果进行计算，sign表示最后的乘积项的符号，

//(AC \* pow(10 , n) + ABDC \* pow(10, (int)(n / 2)) + BD)为：

//

}

}

## 总结：

通过分治方法，将原问题划分为3个规模为原问题一半的问题进行求解，通过递归进行求解，从而使这个算法的复杂度降到O（nlog3），该算法可以减少乘法的次数，提高算法效率。同样，可以将解决问题的规模划分为4段。在问题分析中，将（AD+BC）转化为（（A-B）(D-C)+AC+BD）,而不是（（A+B）(C+D)-AC-BD）是因为在计算机中加法可能会产生进位，而减法不会产生进位，从而保证了精度。

若该整数为二进制数，则代码描述如下：

#define SIGN(A) ((A > 0) ? 1 : -1)

int divideConquer(int X, int Y, int n){

int sign = SIGN(X) \* SIGN(Y);

int x = abs(X);//求X的绝对值；

int y = abs(Y);//求Y的绝对值；

if(x == 0 || y == 0){

//当两数为0时，相乘结果也为0；

return 0;

}else if(n == 1){

//当两数均为1位数，则两数绝对值相乘再与符号sign相乘；

return sign \* x \* y;

}else{

int A = (int) x / pow(2, (int)(n / 2));

//求式子中A的值，A = X/（2n/2），X的高n/2位；

int B = x - A \* pow(2, n / 2);

//求式子中B的值，B = X-A，B为X的低n/2位；

int C = (int) y / pow(2，(int)(n / 2));

//求式子中C的值，C = Y/（2n/2），Y的高n/2位；

int D = y - C \* pow(2, n / 2);

//求式子中D的值，D = Y-C，D为Y的低n/2位；

int AC = divideConquer(A, C, n / 2);

//将原问题划分为子问题，通过递归调用，对A\*C再进行大整数乘法，得到A\*C的值；

int BD = divideConquer(B, D, n / 2);

//通过划分子问题，求出B\*D的值，通过递归对B\*D进行大整数乘法；

int ABDC = divideConquer((A - B), (D - C), n / 2) + AC + BD;

//继续划分原问题为子问题，通过递归调用求出(A-B)\*(D-C)的值，则变成长度为n/2位长度的整数相乘；

return sign \* (AC \* pow(2 , n) + ABDC \* pow(2, (int)(n / 2)) + BD);

//通过得到已知的乘积项，对最后的结果进行计算，sign表示最后的乘积项的符号，

//(AC \* pow(2 , n) + ABDC \* pow(2, (int)(n / 2)) + BD)为：

//

}

}