大整数乘法

刘可扬 PB13210051

实验目的

实现十进制大数的 Karatsuba 乘法。
代码地址：https://github.com/LKY-stephen/LargeIntergerCalculating

实验原理

Karatsuba 乘法
输入：2 个 n 位二进制表示整数 A，B
输出： AxB 的值。
A 与 B 可表示为

$$A = A_0 + A_1 2^{\frac{n}{2}}$$

$$B = B_0 + B_1 2^{\frac{n}{2}}$$

设:

$$A(X) = A_0 + A_1 X; B(X) = B_0 + B_1 X$$

则有：

$$AB = A(X)B(X)|_{x=2^{\wedge}(n/2)}$$

$$A(X)B(X) = \left(A_0 + A_1 X\right)\left(B_0 + B_1 X\right)$$

$$= Z_0 + X\left(Z_1 - Z_0 - Z_2\right) + X^2 Z_2$$

其中

$$Z_0 = A_0 B_0, Z_1 = A_1 B_1, Z_2 = (A_0 + A_1)(B_0 + B_1)$$

时间复杂度为

$$T(n) = 3T(\frac{n}{2}) + O(n) = O(n^{\log_2 3})$$

代码实现

本程序使用 C++编写，使用字符数组存储大整数：
　　每一个数被当作一个一个大数类来进行存储，用户需要只需要依据提示完成输入即可获得结果：

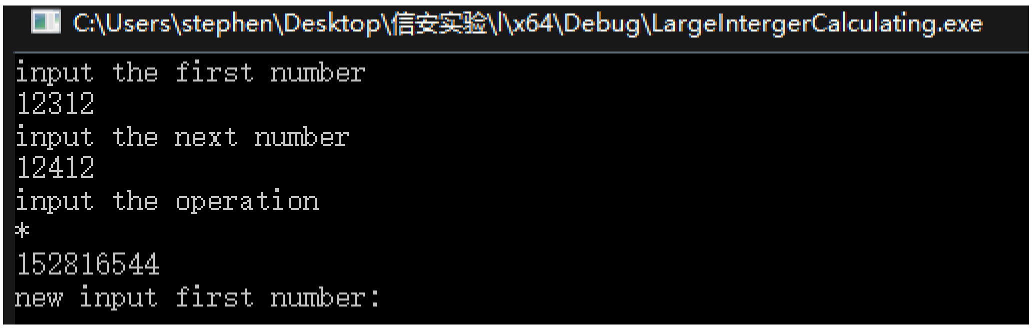

图 1 操作方法

历史记录 存储

12312 × 12412 =

152,816,544

图 2 对比结果

整数类结构：

```
class Interger{
public:
    vector<unsigned char> number;//save the number
    int length;//save the length
    bool signal;//save the signal of the number

    Interger(char head);//init function
    Interger();
    Interger(Interger & x);
    Interger& operator =(const Interger& old);
    bool operator ==(const Interger& old);

    Interger add(Interger number2);//add same signal number2
    vector<unsigned char> sub(Interger number2);//minus number2
    Interger time(Interger* number2);//time number2
    Interger singletime(int y);
    vector<unsigned char> divide(Interger number2);//divided by number2

};
```

图 3 整数类的方法

number 为存储数字绝对值的向量结构，利用字节存储可以降低对内存的需求，使用向量是为了方便递归调用时的操作。length 为绝对值的长度，signal 为数字的符号。由于 legth 的使用次数较多，所以单独建立一个变量来提高速度。

三个初始化方法为，开头字符方法，用于对第一个输入进行判断后建立一个数字；空方法，直接建立一个空的数字；引用方法，为了方便递归调用而使用的方法。

重载了两个符号，一个直接赋值的方法和一个判断相等的方法用于简化赋值。之后的四则运算其实都可以使用符号重载，但是由于初期设计不完善，所以还是使用函数方法来处理，未来可能会全部重载。

singletime 则是对于一个 n 位数和一个 1 位数的乘法提供一个快捷计算接口。

加法操作

```
while (iterator1 >= 0 && iterator2 >= 0)
{
    temp = carry+number[iterator1] + number2.number[iterator2]-'0'-'0';
    if (temp > 0x09) { ... }
    else { ... }
    answer.number.insert(answer.number.begin(), temp+'0');
    iterator1--;
    iterator2--;
}
//add the large part
if (carry == 0x0)
{   //no carrybit
    if (iterator1 >= 0) { ... }
    else if (iterator2 >= 0) { ... }
}
else
{   //carry bit
    if (iterator1 >= 0)
    {//number1 has left
        int i;
        for (i = iterator1; i >= 0; i--)
        {
            if (number[i] != '9') { ... }
            answer.number.insert(answer.number.begin(),'0');
```

图 4 加法的主体操作

减法操作

```
if (answersignal)
{
    iterator2 = number2.length-1;
    iterator1 = length-1;
    while (iterator2 >= 0)
    {
        if (number[iterator1] < number2.number[iterator2])
        {
            for (int j = iterator1-1; j >= 0; j--)
            {
                if (number[j] > '0')
                {
                    number[j]--;
                    break;//break for
                }
                number[j] = '9';
            }
            temp= (number[iterator1] - number2.number[iterator2] + 10)+'0';
        }//endif
        else
        {
            temp= (number[iterator1] -number2.number[iterator2])+'0';
        }
        answer.insert(answer.begin(),temp);
```

图 5 减法操作的主体循环

乘法操作

```
for (i = 0; i < 4; i++)
{
    tempstr[i]->length = tempstr[i]->number.size();
    tempstr[i]->signal = true;
}
tempstr[6]->number.swap(tempstr[0]->time(tempstr[2]).number);//z2=x0*y0
adjust(tempstr[6])
tempstr[4]->number.swap((tempstr[1]->time(tempstr[3]).number));//z0=x1*y1
adjust(tempstr[4])
tempstr[7]->number.swap(tempstr[3]->add(*tempstr[2]).number);//y1+x1
adjust(tempstr[7])
tempstr[5]->number.swap((tempstr[1]->add(*tempstr[0])).time(tempstr[7]).number);//z1
adjust(tempstr[5])
tempstr[7]->number .swap(tempstr[5]->sub(tempstr[4]->add(*tempstr[6])));//z1-z0-z2
adjust(tempstr[7])
for (int i = 0; i < middle; i++)
{
    tempstr[6]->number.insert(tempstr[6]->number.end(), '0');
    tempstr[7]->number.insert(tempstr[7]->number.end(), '0');
    tempstr[6]->number.insert(tempstr[6]->number.end(), '0');
}
tempstr[6]->length = tempstr[6]->number.size();
killzero(tempstr[6], tempstr[6]->length)
adjust(tempstr[6])
tempstr[7]->length = tempstr[7]->number.size();
```

图 6 karatsuba 的主体部分

```
Interger Interger::singletime(int y)
{
    int carryb = 0, leftb, opr, tempb = 0;
    char temp;
    Interger answer;
    answer.number.clear();
    if (y == 0) { ... }
    else if (y == 1) { ... }
    for (int i = length - 1; i >= 0; i--)
    {
        opr = number[i] - '0';
        tempb = opr*y + carryb;
        leftb = tempb % 10;
        carryb = (tempb - leftb) / 10;
        temp = leftb + '0';
        answer.number.insert(answer.number.begin(),temp);
    }
    if (carryb)
    {
        temp = carryb + '0';
        answer.number.insert(answer.number.begin(), temp);
    }
    answer.length = answer.number.size();
    return answer;
}
```

图 7 单步乘法

以上是四个本次实验要用到的功能部分。综合考虑了错误输入以及去掉 0 前缀和符号的处理，并且能够持续的给出操作提示。未来可以给出一个更加优化的版本。

结果分析

图 8 测试结果

结果正确，由于加入了较多鲁棒性测试，所以暂时速度较慢，未来会开始着手优化。

五、总结

本次实验很好地实现了 Karatsuba 乘法。