

# 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/LargeIntegerCalculating>

## 实验原理

Karatsuba 乘法

输入: 2 个  $n$  位二进制表示整数  $A, B$

输出:  $A \times B$  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:

$$\begin{aligned} AB &= A(X)B(X)|_{x=2^{(n/2)}} \\ A(X)B(X) &= (A_0 + A_1 X)(B_0 + B_1 X) \\ &= Z_0 + X(Z_1 - Z_0 - Z_2) + X^2 Z_2 \end{aligned}$$

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

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

## 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:

```
C:\Users\stephen\Desktop\信安实验\Debug\LargeIntegerCalculating.exe
input the first number
12312
input the next number
12412
input the operation
*
152816544
new input first number:
```

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

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

    Integer add(Integer number2);//add same signal number2
    vector<unsigned char> sub(Integer number2);//minus number2
    Integer time(Integer* number2);//time number2
    Integer singletime(int y);
    vector<unsigned char> divide(Integer 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 length 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);//a2=x0*y0
adjust(tempstr[6])
tempstr[4]->number.swap((tempstr[1]->time(tempstr[3]).number));//a0=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);//a1
adjust(tempstr[5])
tempstr[7]->number.swap(tempstr[5]->sub(tempstr[4]->add(*tempstr[6])));//a1-a0-a2
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

```

Integer Integer::singletime(int y)
{
    int carryb = 0, leftb, opr, tempb = 0;
    char temp;
    Integer 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

```
input the first number
107401690697069936506154496106610259106951101200466087656348437972756601353621433052950706348303454020669921106170748016
505437033294537918704503912218394282698170605822084655284010793373170170539394272610212999659369082899634235595298502088
807602904079440109505606958068921748870239529500021380435833045004997715500035964190102271097895007447170557407401516254
070800176053104026009911641393440200448125676117910058063058027709420630750873095700545969850397206532826579603029185405
664007007929266431001081089899208640598844960080001361811774103203911891089006504683110602984046822521907108732340090710
051110466695909399820703949280783284305010536020466450427210929197099347038349303604622147402114435626078706529024945600
359282038587342123507195822122518419058649442395091101420190833984089571370711600896548737740909175860779218228805608024
066180123780862930136582902906021057640073703665032782168048036754796795805389399395839049173848014279783876607044498595
400691055709050848372762438899720900540531811357084270383751232000
input the next number
107401690697069936506154496106610259106951101200466087656348437972756601353621433052950706348303454020669921106170748016
505437033294537918704503912218394282698170605822084655284010793373170170539394272610212999659369082899634235595298502088
807602904079440109505606958068921748870239529500021380435833045004997715500035964190102271097895007447170557407401516254
070800176053104026009911641393440200448125676117910058063058027709420630750873095700545969850397206532826579603029185405
664007007929266431001081089899208640598844960080001361811774103203911891089006504683110602984046822521907108732340090710
051110466695909399820703949280783284305010536020466450427210929197099347038349303604622147402114435626078706529024945600
359282038587342123507195822122518419058649442395091101420190833984089571370711600896548737740909175860779218228805608024
066180123780862930136582902906021057640073703665032782168048036754796795805389399395839049173848014279783876607044498595
400691055709050848372762438899720900540531811357084270383751232000
input the operation
*
115351231645890789438138729465909846693353769467703556866189975162159225264633555029984592367772894938903576323555403693
725569184785421584736687192954164945762563125786097767954679426251644936827225866776662454081690822469361440583075271963
997184549926034538506519401338562867455536293172867356810737127055487947020520945953431203346390623847574605473942664126
926572445441948743777979754486707196943155329097387083937738066213430936436759686188880474408380429150638934311202424577
224179112756253519450436007183646900568849596260173536153132751687037694159663234002935176272943222672893582852859256302
597485436459113466171907914197712098655405375412445017937723590983003208495580135648564513186538960317280517512541249449
141408356584809184029642239344605344229023177050599094324519464287078332736491258927336628534699881485058508185894039954
984431048283689879573487780380484622009467592718664980651233456064828398816256391391750127813989549703969854842087313742
606588329197921089426081717253950681694052459893081162453948416578524780437951859812729962493813258262600279589171764681
197941131357153799899577248272345790858309526908259182530443230171711969615153539597198796914102517553632810056970882426
489855291568341414338617949738038104583438043636720542395752023014652148255177250511454537431753377718632166732932881234
414377182937120255626572929787451006923306722481287492528222382476770904944518671122657701594926876372117784493706620160
724039691780101795749080470044300799103866990139015454075124082511643095215589224297760650189861872890324447323498170574
007828427311442966659445320607396902447736064585572356242408296580955833415450958640801967779003469381433263003155139345
51964733570327995704793344813939475898922418274099449080678708865661888468218849481426482163671195409565092885041610402
060127450492514759144722147934623025056218140500591511033931894325097138791524218331673198653219648299571902929025107154
304557603312302188193051377271239241530589923469195214019984279668908912185128829916417754900922129683782612279062880615
17824000000
```

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。