

电子科技大学信息与软件工程学院

实 验 报 告

学 号 2018091618008

姓 名 袁昊男

(实验) 课程名称 信息安全数学基础

理论教师 熊虎

实验教师 熊虎

电子科技大学

实验报告

学生姓名：袁昊男 学号：2018091618008 指导教师：熊虎

实验地点：信软楼 303 实验时间：2019.12.14

一、实验名称：多精度整数运算的实现

二、实验学时：2 学时

三、实验目的：

- 1、掌握一种 1024 比特整数的加减乘除运算原理；
- 2、依据所选择的算法，编程实现该算法；
- 3、提供该算法的源代码及测试用例，给出运行结果分析。

四、实验原理：

1、算法 1.5.1 多精度数加法。

输入：正整数 x 和 y ，长度为 $n+1$ ，基为 b 。

输出：和 $x+y$ 的 b 进制表示 $x+y=(w_{n+1}w_n\cdots w_1w_0)_b$ 。

(1) $c \leftarrow 0$ (c 是进位数)。

(2) i 从 0 到 n 执行：

如果 $(x_i + y_i + c) < b$ ，则 $w_i \leftarrow x_i + y_i + c$ ， $c \leftarrow 0$ ，否则 $w_i \leftarrow x_i + y_i + c - b$ ， $c \leftarrow 1$ 。

(3) $w_{n+1} \leftarrow c$ 。

(4) 返回 $(w_{n+1}w_n\cdots w_1w_0)$ 。

2、算法 1.5.2 多精度数减法。

输入：正整数 x 和 y ，长度为 $n+1$ ，基为 b ，并且 $x \geq y$ 。

输出：差 $x-y$ 的 b 进制表示 $x-y=(w_nw_{n-1}\cdots w_1w_0)_b$ 。

(1) $c \leftarrow 0$ 。

(2) i 从 0 到 n 执行：

若 $x_i - y_i + c \geq 0$ ，则 $w_i \leftarrow (x_i - y_i + c)$ ， $c \leftarrow 0$ ，否则 $w_i \leftarrow (x_i - y_i + c + b)$ ， $c \leftarrow -1$ 。

(3) 返回 $(w_nw_{n-1}\cdots w_1w_0)$ 。

3、算法 1.5.3 多精度数乘法

输入：正整数 x 和 y ，长度分别为 $n+1$ 和 $t+1$ ，基为 b 。

输出：乘积 $x \cdot y = (w_{n+t+1} \cdots w_1 w_0)_b$ 的 b 进制表示。

(1) i 从 0 到 $(n+t+1)$ 执行： $w_i \leftarrow 0$ 。

(2) i 从 0 到 t 执行：

(2.1) $c \leftarrow 0$ ；

(2.2) j 从 0 到 n 执行：

计算 $(uv)_b = w_{i+j} + x_i \cdot x_j + c$ ， $w_{i+j} \leftarrow v$ ， $c \leftarrow u$ ；

(2.3) $w_{i+n+1} \leftarrow u$ 。

(3) 返回 $(w_{n+t+1} w_n \cdots w_1 w_0)_b$ 。

4、算法 1.5.4 多精度数平方。

输入：正整数 $x = (x_{t-1} x_{t-2} \cdots x_1 x_0)_b$ 。

输出： $x \cdot x = x^2$ 的 b 进制表示。

(1) i 从 0 到 $(2t-1)$ 执行： $w_i \leftarrow 0$ 。

(2) i 从 0 到 $(t-1)$ 执行：

(2.1) $(uv)_b \leftarrow w_{2i} + x_i \cdot x_i$ ， $w_{2i} \leftarrow v$ ， $c \leftarrow u$ ；

(2.2) j 从 0 到 n 执行：

$(uv)_b \leftarrow w_{i+j} + 2x_j \cdot x_i + c$ ， $w_{i+j} \leftarrow v$ ， $c \leftarrow u$ ；

(2.3) $w_{i+t} \leftarrow u$ 。

(3) 返回 $(w_{2t-1} w_{2t-2} \cdots w_1 w_0)_b$ 。

5、算法 1.5.5 多精度数带余除法。

输入：正整数 $x = (x_n x_{n-1} \cdots x_1 x_0)_b$ ， $y = (y_t y_{t-1} \cdots y_1 y_0)_b$ ，其中 $n \geq t \geq 1$ ，
 $y_t \neq 0$ 。

输出：商 $q = (q_{n-t} \cdots q_1 q_0)_b$ 和余数 $r = (r_t \cdots r_1 r_0)_b$ ，满足 $x = qy + r$ ，
 $0 \leq r < y$ 。

(1) j 从 0 到 $(n-t)$ 执行 $q_j \leftarrow 0$ 。

(2) 当 $x \geq yb^{n-t}$ 时执行 $q_{n-t} \leftarrow q_{n-t} + 1$ ， $x \leftarrow x - yb^{n-t}$ 。

(3) i 从 n 到 $(t+1)$ 递减执行：

(3.1) 如果 $x_i = y_t$ ，则 $q_{i-t+1} \leftarrow b-1$ ；否则 $q_{i-t+1} \leftarrow \lfloor (x_i b + x_{i-1}) / y_t \rfloor$ ；

(3.2) 当 $q_{i-t+1}(y_t b + y_{t-1}) > x_i b^2 + x_{i-1} b + x_{i-2}$ 时执行： $q_{i-t+1} \leftarrow q_{i-t+1} - 1$ ；

(3.3) $x \leftarrow x - q_{i-t+1} y b^{i-t+1}$ ；

(3.4) 如果 $x < 0$ ，则 $x \leftarrow x + y b^{i-t+1}$ ， $q_{i-t+1} \leftarrow q_{i-t+1} - 1$ 。

(4) $r \leftarrow x$ 。

(5) 返回 (q, r) 。

五、 实验内容：

- 1、编程实现 1024 比特整数的加减乘除运算；
- 2、提供算法的源代码及测试用例，给出运行结果分析；
- 3、可将运行结果与标准大数库中的运算进行效率比较；
- 4、按照标准实验报告整理实验内容。

六、 实验器材（设备、元器件）：

电脑一台。

七、 实验步骤：

- 1、学习多精度数四则运算的算法，理解算法实现的过程；
- 2、分析题目需求，设计数据结构；
- 3、编码实现，并按测试用例进行输入输出测试；
- 4、分析实验结果，与标准大数库的运算效率作对比。

八、 实验结果与分析（含重要数据结果分析或核心代码流程分析）

1、大数加法

(1) 代码

```
1. /*
2. 大数加法
3. 参数:
4. num1 为第一个大数，用字符数组保存
5. num2 为第二个大数
6. sum 数组保存相加的结果 即: num1+num2=sum
7. 返回值: 返回数组 sum 的有效长度，即计算结果的位数
8. */
9. int Addition(char num1[], char num2[], int sum[])
10. {
11.     int i, j, len;
12.     int n2[MAX] = { 0 };
13.     int len1 = strlen(num1); // 计算数组 num1 的长度，即
        大数的位数
14.     int len2 = strlen(num2); // 计算数组 num2 的长度，即
        大数的位数
15.
16.     len = len1 > len2 ? len1 : len2; // 获取较大的位数
17.     //将 num1 字符数组的数字字符转换为整型数字，且逆向保存在
        整型数组 sum 中，即低位在前，高位在后
18.     for (i = len1 - 1, j = 0; i >= 0; i--, j++)
19.         sum[j] = num1[i] - '0';
20.     // 转换第二个数
```

```

21.     for (i = len2 - 1, j = 0; i >= 0; i--, j++)
22.         n2[j] = num2[i] - '0';
23.     // 将两个大数相加
24.     for (i = 0; i <= len; i++)
25.     {
26.         sum[i] += n2[i]; // 两个数从低位开始相加
27.         if (sum[i] > 9) // 判断是否有进位
28.         { // 进位
29.             sum[i] -= 10;
30.             sum[i + 1]++;
31.         }
32.     }
33.     if (sum[len] != 0) // 判断最高位是否有进位
34.         len++;
35.     return len; // 返回和的位数
36. }

```

说明：采用算法 1.5.1，实现多精度大数加法。实现的数据结构是以 char 型一维数组来存储大数，每一个数组单元存储大数的一位。由于 1024 位二进制数至多可以用 $\lg 2^{1024} = 1024 \cdot \lg 2 \approx 309$ 位十进制数表示，因此在进行大数加法运算时，被加数、加数及结果都是大小为 500 的数组，以保证不会产生溢出错误。函数的参数为两个 char 型待操作数 num1 和 num2，以及存放结果的 int 型数组 sum。函数返回数组 sum 的有效长度，即加法结果的位数。

(2) 测试用例

注：为便于说明算法原理及测试编码本身的有效性和正确性，本测试用例中未使用 309 位十进制数（即 1024 位二进制）

序号	输入 num1	输入 num2	预期输出 sum	实际输出 sum
1	0	0	0	0
2	123456789876	0	123456789876	123456789876
	543212345678		543212345678	543212345678
	987654321234		987654321234	987654321234
	567898765432		567898765432	567898765432
3	12	74	12	12
	537493856198		665077614802	665077614802
	765432128647		621817965913	621817965913
	582946543263		452221222238	452221222238
	846224479165		468659966138	468659966138
	43	74	17	17

(3) 运行截图

```

C:\袁昊男\学习\大二上\信息安全数学基础\实验\...
加法测试:
0
+
0
=
0
duration: 0.000000 seconds
请按任意键继续. . .

```

```

C:\袁昊男\学习\大二上\信息安全数学基础\实验\...
加法测试:
12345678987654321234567898765432123456789876543212
+
0
=
12345678987654321234567898765432123456789876543212
duration: 0.011000 seconds
请按任意键继续. . .

```

```

C:\袁昊男\学习\大二上\信息安全数学基础\实验\...
加法测试:
53749385619876543212864758294654326384622447916543
+
12758375860385638583726586927467897462243548697274
=
66507761480262181796591345222122223846865996613817
duration: 0.007000 seconds
请按任意键继续. . .

```

注：结果与手工计算验证正确。

(4) 效率对比

测试次数	本算法运行时间（秒）	标准库运行时间（秒）
1	0.007	0.001
2	0.009	0.002
3	0.013	0.001
4	0.007	0.001
5	0.006	0.001
平均时间（秒）	0.0084	0.0012
效率比	14.286%	

2、大数减法

(1) 代码

```

1.  /*
2.   大数减法
3.   参数:
4.   num1 为被减数, 用字符数组保存
5.   num2 为减数
6.   sum 数组保存相减的结果    即: num1-num2=sum
7.   返回值: 返回数组 sum 的有效长度, 即计算结果的位数
8.  */
9.  int Subtraction(char num1[], char num2[], int sum[])
10. {
11.     int i, j, len, flag;
12.     char *temp;
13.     int n2[MAX] = { 0 };
14.     int len1 = strlen(num1); // 计算数组 num1 的长度, 即
        大数的位数
15.     int len2 = strlen(num2); // 计算数组 num2 的长度, 即
        大数的位数
16.
17.     // 在进行减法之前要进行一些预处理
18.     flag = 0; // 为 0 表示结果是正整数, 为 1 表示结果是负整
        数
19.     if (len1 < len2) // 如果被减数位数小于减数
20.     {
21.         flag = 1; // 标记结果为负数
22.         // 交换两个数, 便于计算
23.         temp = num1;
24.         num1 = num2;
25.         num2 = temp;
26.         len = len1;
27.         len1 = len2;
28.         len2 = len;
29.     }
30.     else if (len1 == len2) // 如果被减数的位数等于减数的
        位数
31.     {
32.         // 判断哪个数大
33.         for (i = 0; i < len1; i++)
34.         {
35.             if (num1[i] == num2[i])
36.                 continue;
37.             if (num1[i] > num2[i])
38.             {
39.                 flag = 0; // 标记结果为正数
40.                 break;
41.             }
42.             else
43.             {
44.                 flag = 1; // 标记结果为负数
45.                 // 交换两个数, 便于计算
46.                 temp = num1;
47.                 num1 = num2;
48.                 num2 = temp;
49.                 break;
50.             }
51.         }
52.     }
53.     len = len1 > len2 ? len1 : len2; // 获取较大的位数

```

```

54. //将 num1 字符数组的数字转换为整型数且逆向保存在整型数组
    sum 中，即低位在前，高位在后
55. for (i = len1 - 1, j = 0; i >= 0; i--, j++)
56.     sum[j] = num1[i] - '0';
57. // 转换第二个数
58. for (i = len2 - 1, j = 0; i >= 0; i--, j++)
59.     n2[j] = num2[i] - '0';
60. // 将两个大数相减
61. for (i = 0; i <= len; i++)
62. {
63.     sum[i] = sum[i] - n2[i]; // 两个数从低位开始相
    减
64.     if (sum[i] < 0) // 判断是否有借位
65.     { // 借位
66.         sum[i] += 10;
67.         sum[i + 1]--;
68.     }
69. }
70. // 计算结果长度
71. for (i = len1 - 1; i >= 0 && sum[i] == 0; i--)
72.     ;
73. len = i + 1;
74. if (flag == 1)
75. {
76.     sum[len] = -1; // 在高位添加一个-1 表示负数
77.     len++;
78. }
79. return len; // 返回结果的位数
80. }

```

说明：采用算法 1.5.2，实现多精度大数减法。实现的数据结构是以 char 型一维数组来存储大数，每一个数组单元存储大数的一位。由于 1024 位二进制数至多可以用 $\lg 2^{1024} = 1024 \cdot \lg 2 \approx 309$ 位十进制数表示，因此在进行大数减法运算时，被减数、减数及结果都是大小为 500 的数组，以保证不会产生溢出错误。函数的参数为两个 char 型待操作数 num1 和 num2，以及存放结果的 int 型数组 sum。函数返回数组 sum 的有效长度，即减法结果的位数。

(2) 测试用例

注：为便于说明算法原理及测试编码本身的有效性和正确性，本测试用例中未使用 309 位十进制数（即 1024 位二进制）

序号	输入 num1	输入 num2	预期输出 sum	实际输出 sum
1	0	0	0	0
2	0	123456789876	-1234567898	-1234567898
		543212345678	765432123456	765432123456
		987654321234	789876543212	789876543212
		567898765432	345678987654	345678987654

		12	3212	3212
3	537493856198	127583758603	409910097594	409910097594
	765432128647	856385837265	909046291381	909046291381
	582946543263	869274678974	713671864289	713671864289
	846224479165	622435486972	223788992192	223788992192
	43	74	69	69

(3) 运行截图

```

C:\袁昊男\学习\大二上\信息安全数学基础\实验\...
减法测试:
0
-
0
=
-
duration: 0.001000 seconds
请按任意键继续...

```

```

C:\袁昊男\学习\大二上\信息安全数学基础\实验\...
减法测试:
12345678987654321234567898765432123456789876543212
-
12345678987654321234567898765432123456789876543212
=
-12345678987654321234567898765432123456789876543212
duration: 0.009000 seconds
请按任意键继续...

```

```

C:\袁昊男\学习\大二上\信息安全数学基础\实验\...
减法测试:
53749385619876543212864758294654326384622447916543
-
12758375860385638583726586927467897462243548697274
=
40991009759490904629138171367186428922378899219269
duration: 0.016000 seconds
请按任意键继续...

```

注：结果与手工计算验证正确。

(4) 效率对比

测试次数	本算法运行时间（秒）	标准库运行时间（秒）
1	0.016	0.001
2	0.006	0.001

3	0.006	0.001
4	0.011	0.002
5	0.005	0.001
平均时间（秒）	0.0088	0.0012
效率比	13.636%	

3、大数乘法

(1) 代码

```

1.  /*
2.   大数乘法
3.   参数:
4.   num1 为第一个因数, 用字符数组保存
5.   num2 为第二个因数
6.   sum 数组保存相乘的结果  即: num1*num2=sum
7.   返回值: 返回数组 sum 的有效长度, 即计算结果的位数
8.  */
9.  int Multiplication(char num1[], char num2[], int sum[])
10. {
11.     int i, j, len, len1, len2;
12.     int a[MAX + 10] = { 0 };
13.     int b[MAX + 10] = { 0 };
14.     int c[MAX * 2 + 10] = { 0 };
15.
16.     len1 = strlen(num1);
17.     for (j = 0, i = len1 - 1; i >= 0; i--) //把数字字符
        转换为整型数
18.         a[j++] = num1[i] - '0';
19.     len2 = strlen(num2);
20.     for (j = 0, i = len2 - 1; i >= 0; i--)
21.         b[j++] = num2[i] - '0';
22.
23.     for (i = 0; i < len2; i++)//用第二个数乘以第一个数, 每
        次一位
24.     {
25.         for (j = 0; j < len1; j++)
26.         {
27.             c[i + j] += b[i] * a[j]; //先乘起来, 后面统一
                进位
28.         }
29.     }
30.
31.     for (i = 0; i < MAX * 2; i++) //循环统一处理进位问
        题
32.     {
33.         if (c[i] >= 10)
34.         {
35.             c[i + 1] += c[i] / 10;
36.             c[i] %= 10;
37.         }
38.     }
39.

```

```
40.     for (i = MAX * 2; c[i] == 0 && i >= 0; i--); //跳过
      高位的 0
41.     len = i + 1; // 记录结果的长度
42.     for (; i >= 0; i--)
43.         sum[i] = c[i];
44.     return len;
45. }
```

说明：采用算法 1.5.3，实现多精度大数乘法。实现的数据结构是以 char 型一维数组来存储大数，每一个数组单元存储大数的一位。由于 1024 位二进制数至多可以用 $\lg 2^{1024} = 1024 \cdot \lg 2 \approx 309$ 位十进制数表示，因此在进行大数乘法运算时，被乘数、乘数及结果都是大小为 1000 的数组，以保证不会产生溢出错误。函数的参数为两个 char 型待操作数 num1 和 num2，以及存放结果的 int 型数组 sum。函数返回数组 sum 的有效长度，即乘法结果的位数。

(2) 测试用例

注：为便于说明算法原理及测试编码本身的有效性和正确性，本测试用例中未使用 309 位十进制数（即 1024 位二进制）

序号	输入 num1	输入 num2	预期输出 sum	实际输出 sum
1	0	0	0	0
2	0	123456789876 543212345678 987654321234 567898765432 12	0	0
3	537493856198 765432128647 582946543263 846224479165 43	127583758603 856385837265 869274678974 622435486972 74	685754864003 191862053883 941381633246 330624341379 857014407080 575108359957 070861460970 276963123603 782	685754864003 191862053883 941381633246 330624341379 857014407080 575108359957 070861460970 276963123603 782

(3) 运行截图



注：结果与手工计算验证正确。

(4) 效率对比

测试次数	本算法运行时间（秒）	标准库运行时间（秒）
1	0.008	0.001
2	0.011	0.001
3	0.015	0.001
4	0.011	0.001
5	0.012	0.001
平均时间（秒）	0.0114	0.0010
效率比	8.772%	

4、大数除法

(1) 代码

```

1.  /*
2.   函数 SubStract 功能:
3.   用长度为 len1 的大整数 p1 减去长度为 len2 的大整数 p2
4.   结果存在 p1 中, 返回值代表结果的长度
5.   不够减: 返回 -1 ,   正好够: 返回 0
6.  */
7.  int SubStract(int *p1, int len1, int *p2, int len2)
8.  {
9.      int i;
10.     if (len1 < len2)
11.         return -1;
12.     if (len1 == len2)
13.     {
14.         for (i = len1 - 1; i >= 0; i--)
15.         {
16.             if (p1[i] > p2[i]) // 若大, 则满足条件, 可
做减法
17.                 break;
18.             else if (p1[i] < p2[i]) // 否则返回 -1
19.                 return -1;
20.         }
21.     }
22.     for (i = 0; i <= len1 - 1; i++) // 从低位开始做减
法
23.     {
24.         p1[i] -= p2[i]; // 相减
25.         if (p1[i] < 0) // 若是否需要借位
26.         { // 借位
27.             p1[i] += 10;
28.             p1[i + 1]--;
29.         }
30.     }
31.     for (i = len1 - 1; i >= 0; i--) // 查找结果的最高
位
32.     {
33.         if (p1[i]) // 最高位第一个不为 0
34.             return (i + 1); // 得到位数并返回
35.     }
36.     return 0; // 两数相等的时候返回 0
37. }
38.
39. /*
40.  大数除法---结果不包括小数点
41.  num1 被除数
42.  num2 除数
43.  sum 商, 存放计算的结果, 即: num1/num2=sum
44.  返回数组 sum 的有效长度, 即商的位数
45.  */
46.  int Division(char num1[], char num2[], char sum[], int
r[])
47.  {
48.      int i, j;
49.      int len1, len2, len = 0; // 大数位数
50.      int dValue; // 两大数相差位数
51.      int nTemp; // Subtract 函数返回值
52.      int num_a[MAX] = { 0 }; // 被除数
53.      int num_b[MAX] = { 0 }; // 除数
54.      int num_c[MAX] = { 0 }; // 商

```

```

55.     int temp[MAX * 2 + 10] = { 0 };
56.     int temp1[MAX] = { 0 };
57.     char temp2[MAX];
58.     int temp3[MAX];
59.     int temp4[MAX];
60.
61.     len1 = strlen(num1);           //获得大数的位数
62.     len2 = strlen(num2);
63.
64.     //将数字字符转换成整型数，且翻转保存在整型数组中
65.     for (j = 0, i = len1 - 1; i >= 0; j++, i--)
66.         num_a[j] = num1[i] - '0';
67.     for (j = 0, i = len2 - 1; i >= 0; j++, i--)
68.         num_b[j] = num2[i] - '0';
69.
70.     if (len1 < len2)                //如果被除数小于除数，直接
        返回-1，表示结果为 0
71.     {
72.         return -1;
73.     }
74.     dValue = len1 - len2;           //相差位数
75.     for (i = len1 - 1; i >= 0; i--) //将除数扩大，使
        得除数和被除数位数相等
76.     {
77.         if (i >= dValue)
78.             num_b[i] = num_b[i - dValue];
79.         else //低位置 0
80.             num_b[i] = 0;
81.     }
82.     len2 = len1;
83.     for (j = 0; j <= dValue; j++) //重复调用，同时记
        录减成功的次数，即为商
84.     {
85.         while ((nTemp = SubStract(num_a, len1, num_b +
            j, len2 - j)) >= 0)
86.         {
87.             len1 = nTemp;           //结果长度
88.             num_c[dValue - j]++;    //每成功减一次，
            将商的相应位加 1
89.         }
90.     }
91.     // 计算商的位数，并将商放在 sum 字符数组中
92.     for (i = MAX - 1; num_c[i] == 0 && i >= 0; i-
        -); //跳过高位 0，获取商的位数
93.     if (i >= 0)
94.         len = i + 1; // 保存位数
95.     for (j = 0; i >= 0; i--, j++) // 将结果复制到
        sum 数组中
96.         sum[j] = num_c[i] + '0';
97.     sum[j] = '\0'; // sum 字符数组结尾置 0
98.
99.     int len_new= Multiplication(num2, sum, temp);
100.    for (i = 0; i < len_new; i++)
101.    {
102.        temp2[i] = (char)('0' + temp[i]);
103.    }
104.    temp2[i] = '\0';
105.

```

```

106.     for (i = len_new - 1, j = 0; i >= 0; i--, j++)
107.         temp3[j] = temp2[i] - '0';
108.
109.     for (i = 0; i < len_new; i++)
110.     {
111.         temp2[i] = (char)('0' + temp3[i]);
112.     }
113.
114.     len3=Subtraction(num1, temp2, temp4);
115.
116.     i = len3-1;
117.
118.     for (j = 0; i >= 0; i--, j++)    // 将结果复制到
sum 数组中
119.         r[j] = temp4[i];
120.
121.
122.     return len;    // 返回商的位数
123. }

```

说明：采用算法 1.5.5，实现多精度大数除法。实现的数据结构是以 char 型一维数组来存储大数，每一个数组单元存储大数的一位。由于 1024 位二进制数至多可以用 $\lg 2^{1024} = 1024 \cdot \lg 2 \approx 309$ 位十进制数表示，因此在进行大数减法运算时，被乘数、乘数及结果都是大小为 500 的数组，以保证不会产生溢出错误。函数的参数为两个 char 型待操作数 num1 和 num2、存放商的 char 型数组 sum，以及存放余数的 int 型数组 r。函数返回数组 sum 的有效长度，即商的位数。

(2) 测试用例

注：为便于说明算法原理及测试编码本身的有效性和正确性，本测试用例中未使用 309 位十进制数（即 1024 位二进制）

序号	输入 num1	输入 num2	预期输出 sum	实际输出 sum
1	123456789876 543212345678 987654321234 567898765432 12	0	Error	Error
2	0	123456789876 543212345678 987654321234 567898765432 12	0	0
3	537493856198	127583758603	685754864003	685754864003

	765432128647		191862053883	191862053883
	582946543263		941381633246	941381633246
	846224479165		330624341379	330624341379
	43		857014407080	857014407080
			575108359957	575108359957
			070861460970	070861460970
			276963123603	276963123603
			782	782

(3) 运行截图

```

除法测试:
12345678987654321234567898765432123456789876543212
÷
0
=
Error
duration: 0.003000 seconds
请按任意键继续...

```

```

除法测试:
0
÷
12345678987654321234567898765432123456789876543212
=
0
r=0
duration: 0.001000 seconds
请按任意键继续...

```

```

除法测试:
53749385619876543212864758294654326384622447916543
÷
127583758603
=
421287052587371274192126241937490213263
r=866964954
duration: 0.006000 seconds
请按任意键继续...

```

注：结果与手工计算验证正确。

(4) 效率对比

测试次数	本算法运行时间（秒）	标准库运行时间（秒）
1	0.006	0.000
2	0.009	0.001
3	0.014	0.001
4	0.005	0.001
5	0.015	0.001
平均时间（秒）	0.0098	0.0008
效率比	8.163%	

九、 总结及心得体会：

在密码学中使用的整数已经远远超过程序语言中的长整型表示范围，因此有必要对大整数的运算实现进行研究，而大整数的运算可以通过多精度数的算法来实现。通过本实验，实现了大数的加、减、乘、除一般效率的四则运算，与标准大数库的运行效率相比仍有较大差距。除了可以选用效率更高的算法外，还可以设计其他数据结构来实现算法，比如使用结构体变量，每个单元存储 64 位十进制数来减少单元数，进一步提升效率等等。

通过本次实验，我掌握了一般大数的算法与编程方法，对密码学的编码有了初步的认识，本实验所写的函数还可方便其他函数调用，如扩展的欧几里得算法、RSA 算法等，有较大的实用性。

十、 对本实验过程及方法、手段的改进建议：

本实验可以放在对应章节的学习中，统一安排在期末进行学生完成的时间压力较大。

报告评分：

指导教师签字：