大数相除的快速算法*

邢卫　　宋东平

（浙江大学工业控制技术国家重点实验室　　杭州 310027）

摘　要　本文提出了两种新的大整数相除的快速算法，证明了新算法的正确性，并与目前常用的 Knuth 算法进行了性能比较分析。特别是当公开钥密码体制 RSA 的公开密钥满足一定的条件时，新算法具有明显的优点。仿真结果验证了新算法的有效性。

关键词　大数除法，估商，RSA

1　引言

公开钥密码体制因其显著的特点正日益受到人们的青睐。RSA 是目前相对比较可靠且已走向实际应用的一种公开钥密码体制，用软件实现 RSA 要涉及到大整数的加、减、乘、除和模运算。其中最重要的是大数相除运算。它是提高 RSA 软件实现速度的关键，也是 RSA 走向实用的关键。因此，寻找快速的大数相除算法是很有意义的。

本文提出了两种新的大整数相除的快速算法，并与目前常用的 Knuth 算法进行了比较分析。

2　大数相除的算法框架

我们所谓的大数（十进制超过 100 位）早已超出了现有计算机的字长限制。在计算机内部一般采用"b 进制多位数"来表示大数。比如记 $x\equiv(x_0x_1\cdots x_k)_b$，其中 $0\leqslant x_i<b$，则有 $x=x_0b^k-x_1b^{k-1}+\cdots+x_{k-1}b-x_k$。在具体实现时，可采用数组来表示大数 x 的每一"位"，即在数组的元素中依次存放 x_i。在下文中，我们简记 $x=x_0x_1\cdots x_k$。

进制 b 的选择也很重要。自然，b 必须能在计算机字长范围内表示。可以选取 10，1000 或 10000 等，但较好的方法是选取 $b=2^w$，其中 w 为计算机的字长。在现行的计算机上，这意味着两个二进制 2w 位以内的数的乘、除法可以由硬件指令直接执行。而且，采用这种进制表示法，大数在计算机中的存放效率也可以达到最高（100%）。例如在 IBM PC 系列兼容机上，进制 b 可取 2^{16}（64K），我们称之为"64K 进制"。

大数相除的最简单的方法是将 b 进制大数展开化成二进制数，然后按位处理，再将商和余数化成 b 进制数。这种方法在作除法时操作简单，直观，但速度慢。合理的方法应按 b 进制直接处理，并利用计算机本身的乘除指令。这类算法比化成二进制的除法要快 4 到 10 倍。

不失一般性，考虑一个 m+n 位数 u 除以一个 n 位数 v，得到 m+1 位商 q 和 n 位余数 r。

* 本文为中科院科技资金资助课题。1996 年 1 月 29 日收到

记被除数 $u=u_1u_2\cdots u_{m+n}$，除数 $v=v_1v_2\cdots v_n$。

我们回顾一下普通的长除法，可以看出一次大数的除法实际上可以分解成许多简单的步骤，每一步都是 n+1 位数 u 除以 n 位数 v，且满足 $0\leqslant u/v<b$。每一步的余数 r 都比 v 小，并运用于下一步中。

$$\begin{array}{r} q \\ v_1v_2\cdots v_n \;\big/\!\!\overline{\;u_0u_1u_2\cdots u_n} \\ \underline{\leftarrow qv\rightarrow} \\ \leftarrow r\rightarrow \end{array}$$

例如十进制下，3095 除以 47，我们先用 309 除以 47，得商 6 余数 27；再以 275 除以 47，得商 5 余数 40。所以最后我们得到商 65 余数 40。根据这一思路，我们得到大数相除的算法框架如下：

【算法 B】

1、置循环变量 j=0

2、估计 $u_ju_{j+1}\cdots u_{j+n}$ 除以 $v_1v_2\cdots v_n$ 的商 $\hat{q}_j$（第一次循环中，令 $u_0=0$）

3、修正 $\hat{q}_j$ 得到准确的商 q_j

4、计算 $u_ju_{j+1}\cdots u_{j+n}-q_j=v_1v_2\cdots v_n$，并替代 $u_ju_{j+1}\cdots u_{j+n}$

5、j=j+1，如果 $j\leqslant m$ 则转 2

6、已求得商 $q=q_0q_1\cdots q_m$，余数 $r=u_{m+1}u_{m+2}\cdots u_{m+n}$。

显然，在整个算法中，本位商 q_j 的估计及其修正是关键问题。

3　大数相除的快速算法

对于本位商的估计与修正，早在 1969 年 Knuth[1]就指出，若令 $\hat{q}=\min(\lfloor u_0b-u_1)/v_1\rfloor, b-1)$，则当 $v_1\geqslant\lfloor b/2\rfloor$ 时，可以证明 $\hat{q}-2\leqslant q\leqslant\hat{q}$。由此可以得到大数相除的算法如下：

【算法 K】

0、规范化。将 $u=u_0u_1u_2\cdots u_{m+n}$ 和 $v=v_1v_2\cdots v_n$ 同时左移相同的次数 d，使得 $v_1\geqslant\lfloor b/2\rfloor$

1、置循环变量 j=0

2、估商。若 $u_j=v_1$，则 $\hat{q}_j=b-1$，否则 $\hat{q}_j=\lfloor(u_jb+u_{j+1})/v_1\rfloor$

3a、修正。计算部分积 $p=\hat{q}_j\times v_1v_2\cdots v_n$。若 $p\leqslant u_ju_{j+1}\cdots u_{j+n}$，则 $q_j=\hat{q}_j$，转 4

3b、p=p−v，若 $p\leqslant u_ju_{j+1}\cdots u_{j+n}$，则 $q_j=\hat{q}_j-1$，转 4

3c、p=p−v，$q_j=\hat{q}_j-2$

4、计算 $u_ju_{j+1}\cdots u_{j+n}-p$，并替代 $u_ju_{j+1}\cdots u_{j+n}$

5、j=j−1，如果 $j\leqslant m$ 则转 2

6、已求得商 $q=q_0q_1\cdots q_m$；将 $u_{m+1}u_{m+2}\cdots u_{m+n}$ 左移 d 次，得到余数 r。

相对于【算法 B】，可以看到该算法加入了第 0 步规范化操作（被除数、除数左移）和第 6 步中的还原操作（余数右移）。为分析移动次数 d 的数学期望，可假设 v_1 在 $[1,b-1]$ 区间内均匀分布，则平均移动的次数为 $E(d)=\sum_{k=1}^{w}(k-1)/2^k$，其中 w 为计算机的字长，且 $b=2^w$。当 w=16 时，E(d)=0.9997，即大约平均移动 1 次。注意到在第 2 步中估计商 $\hat{q}_j$ 时，正好可以利用计

算机中的除法指令，故平均移动次数只有 1 次的代价还是很理想的。在[2]文中，大数相除使用的就是 Knuth 算法。

但是在一些特定的应用场合如公开密钥算法 RSA 中，需要频繁地作大数的模运算（即取余数），而此时的除数 $v=v_1v_2\cdots v_n$ 是公开密钥中的一部分，是固定已知的。当 v_1 远小于 b 时，在规范化和还原操作中就需要作多次的左移和右移。能否找到直接基于 b 进制的、又不需要进行规范化操作和还原操作的算法呢？

估商算法的灵魂在于估计的准确性，使得只对 $\hat{q}_j$ 作微小的修正就可得到准确的商 q_j。考察上商的过程和 Knuth 估商算法，我们可以看出应使除数足够大，以保证所估计的商的准确性。所以，我们提出使用"3 位除以 2 位"的估商法，即根据 $u_ju_{j+1}u_{j+2}$ 和 v_1v_2 来求得 $\hat{q}_j$。在此实际上需要求得 $(u_jb^2+u_{j+1}b-u_{j+2})/(v_1b+v_2)$。

但是，被除数 $(u_jb^2+u_{j+1}b+u_{j+2})$ 将超出计算机的乘、除法指令操作长度 2w。对于这种情况，我们提出了两类算法：第一类是直接求得精确的 $\hat{q}_j=\lfloor(u_jb^2+u_{j+1}b+u_{j+2})/(v_1b+v_2)\rfloor$；第二类是间接估计算法。在实现直接求解算法时，一种方法是采用移位操作做二进制除法；另一种则是减小进制 b，使得 $(u_jb^2+u_{j+1}b+u_{j+2})$ 不超出计算机的乘除法指令操作长度 2w。前者的代价是用移位代替了除法指令；后者则由于减小了 b 而使得大数据的"位数"增多，从而使循环次数 j 变大。在下一节我们将证明 $\hat{q}_j-1\leqslant q_j\leqslant\hat{q}_j$。当用移位操作方法实现直接求解时，则可得大数相除算法如下：

【算法 X】

1、置循环变量 j=0

2、估商。若 $u_ju_{j+1}=v_1v_2$，则 $\hat{q}_j=b-1$，否则用二进制移位相除计算 $\hat{q}_j=\lfloor(u_jb^2+u_{j+1}b+u_{j+2})/(v_1b+v_2)\rfloor$

3a、修正。计算部分积 $p=\hat{q}_j\times v_1v_2\cdots v_n$。若 $p\leqslant u_ju_{j+1}\cdots u_{j+n}$，则 $q_j=\hat{q}_j$，转 4

3b、$p=p-v$，$q_j=\hat{q}_j-1$

4、计算 $u_ju_{j+1}\cdots u_{j+n}-p$，并替代 $u_ju_{j+1}\cdots u_{j+n}$

5、j=j+1，如果 $j\leqslant m$ 则转 2

6、已求得商 $q=q_0q_1\cdots q_m$，余数 $r=u_{m+1}u_{m+2}\cdots u_{m+n}$。

对比【算法 X】和【算法 K】，可以看出前者的估商精度更高，且不需要规范化操作和还原操作，但无法使用除法指令，需要一个专门的、用移位算法实现的"3 位除以 2 位"的函数。

若坚持使用计算机固有的乘、除法指令，或是因为所使用的高级语言不支持移位操作，同时又不以减小进制为代价，则可以采用第二类间接估计算法。在此引入两个辅助变量 $x=\lfloor(u_jb-u_{j+1})/v_1\rfloor$ 和 $y=\lfloor xv_2/(v_1b-v_2)\rfloor$，并令 $\hat{q}_j=\min(x-y,b-1)$。在下一节我们将证明 $\hat{q}_j-2\leqslant q_j\leqslant\hat{q}_j$。此时可得大数相除的算法如下：

【算法 S】

1、置循环变量 j=0

2、估商。计算 $x=\lfloor(u_jb-u_{j+1})/v_1\rfloor$ 和 $y=\lfloor xv_2/(v_1b+v_2)\rfloor$*，并令 $\hat{q}_j=\min(x-y,b-1)$

---

* 当且仅当 $u_j=v_1$ 时，$x\geqslant b$ 成立，从而可能使 $xv_2\geqslant b^2$ 成立，即 xv_2 超出二进制 2w 位。此时，需对 y 的计算采用修正的表达式，但仍可精确地算得 y 值。

3a、修正。计算部分积 $p=\hat{q}_j\times v_1v_2\cdots v_n$。若 $p\leqslant u_ju_{j+1}\cdots u_{j+n}$，则 $q_j=\hat{q}_j$，转 4

3b、$p=p-v$，若 $p\leqslant u_ju_{j+1}\cdots u_{j+n}$，则 $q_j=\hat{q}_j-1$，转 4

3c、$p=p-v$，$q_j=\hat{q}_j-2$

4、计算 $u_ju_{j+1}\cdots u_{j+n}-p$，并替代 $u_ju_{j+1}\cdots u_{j+n}$

5、$j=j+1$，如果 $j\leqslant m$ 则转 2

6、已求得商 $q=q_0q_1\cdots q_m$，余数 $r=u_{m+1}u_{m+2}\cdots u_{m+n}$。

对比【算法 S】和【算法 K】，可以看出其估商精度是一样的，前者的估商算法比后者复杂，但不需要规范化操作和还原操作。

4　算法的正确性证明

无论是哪一种估商及其修正算法，其估商及修正过程本身与循环变量 j 无关，因此事实上只需证明其中的任意一步是正确的即可。不失一般性，简记 $u=u_0u_1\cdots u_n$，$v=v_1v_2\cdots v_n$，估计所得的商为 $\hat{q}$，实际的商为 q。这里满足 $0\leqslant u/v<b$。

命题 1　对于【算法 X】，已知 $\hat{q}=\lfloor(u_0b^2-u_1b+u_2)/(v_1b+v_2)\rfloor$，求证 $\hat{q}-1\leqslant q\leqslant\hat{q}$。

命题 2　对于【算法 S】，已知 $x=\lfloor(u_0b+u_1)/v_1\rfloor$，$y=\lfloor xv_2/(v_1b+v_2)\rfloor$ 和 $\hat{q}=\min(x-y,b-1)$，求证 $\hat{q}-2\leqslant q\leqslant\hat{q}$。

命题 1 的证明

当使用移位或减小进制 b 精确地实现 $\hat{q}=\lfloor(u_0b^2+u_1b+u_2)/(v_1b+v_2)\rfloor$ 时，从 $\hat{q}$ 的定义我们得

$$(v_1b-v_2)\hat{q}\leqslant u_0b^2-u_1b+u_2 \tag{1}$$

$$u_0b^2+u_1b+u_2<(v_1b+v_2)(\hat{q}+1) \tag{2}$$

从(2)式，考虑到所有的表达式都是正整数，可得：

$$u_0b^2+u_1b+u_2-1\leqslant(v_1b+v_2)(\hat{q}+1) \tag{3}$$

对于被除数 u 和除数 v，我们有：

$$u=u_0b^n+u_1b^{n-1}+u_2b^{n-2}+\cdots+u_n<b^{n-2}(u_0b^2+u_1b+u_2+1) \tag{4}$$

$$b^{n-2}(v_1b+v_2)\leqslant v_1b^{n-1}+v_2b^{n-2}+\cdots+v_n=v \tag{5}$$

从(3)(4)(5)式，可得：

$$u<b^{n-2}(u_0b^2+u_1b+u_2+1)\leqslant b^{n-2}(v_1b+v_2)(\hat{q}+1)\leqslant v(\hat{q}+1)$$

即

$$u<v(\hat{q}-1) \tag{6}$$

换句话说，$\hat{q}-1$ 作为商“太大了”，所以，$q\leqslant\hat{q}$。

另一方面，我们有

$$v=v_1b^{n-1}+v_2b^{n-2}+\cdots+v_n<b^{n-2}(v_1b+v_2+1) \tag{7}$$

因为 $\hat{q}<b$，$1\leqslant v_1<b$，所以

$$\hat{q}-v_1b-v_2-1<0$$

从而

$$(v_1b+v_2+1)(\hat{q}-1)=(v_1b+v_2)\hat{q}+(\hat{q}-v_1b-v_2-1)<(v_1b+v_2)\hat{q} \tag{8}$$

从(1)(8)式，可得：

$$(v_1b+v_2+1)(\hat{q}-1)<u_0b^2-u_1b+u_2 \tag{9}$$

从(7)(9)式，得到：

$$v(\hat{q}-1)<b^{n-2}(v_1b+v_2+1)(\hat{q}-1)<b^{n-2}(u_0b^2-u_1b+u_2)\leqslant u$$

即　　$v(\hat{q}-1)<u$　　(10)

换句话说，$\hat{q}-1$ 作为商已经"够小了"。所以，$\hat{q}-1\leqslant q$。　　【证毕】

命题 2 的证明

在算法的每一步，满足 $0\leqslant u/v<b$，所以 $u_0\leqslant v_1$，当 $u_0<v_1$ 时，容易证明：

$$0\leqslant y\leqslant x<b \tag{11}$$

此时，$\hat{q}=x-y$。从 x 和 y 的定义，我们有

$$v_1x\leqslant u_0b+u_1 \tag{12}$$

$$u_0b+u_1<v_1(x+1) \tag{13}$$

$$(v_1b-v_2)y\leqslant xv_2 \tag{14}$$

$$xv_2<(v_1b+v_2)(y+1) \tag{15}$$

从(13)式，考虑到所有的表达式都是正整数，可得：

$$u_0b+u_1-1\leqslant v_1(x+1)$$

所以　　$u=u_0b^n+u_1b^{n-1}+\cdots+u_n<b^{n-1}(u_0b+u_1+1)\leqslant b^{n-1}v_1(x+1)$　　(16)

因为　　$v_1b(x+1)+(v_2x-(v_1b+v_2)y)+v_2=(v_1b+v_2)(x-y+1)$

考虑(14)式及 $v_2\geqslant 0$，从上式中可得到

$$bv_1(x+1)<(v_1b+v_2)(x-y+1) \tag{17}$$

注意到 $\hat{q}=x-y$ 可推得：

$$b^{n-2}(v_1b+v_2)(x-y-1)\leqslant(v_1b^{n-1}+v_2b^{n-2}+\cdots+v_n)(x-y+1)=v(\hat{q}+1) \tag{18}$$

从(16)(17)(18)式得到

$$u<b^{n-1}v_1(x+1)<b^{n-2}(v_1b+v_2)(x-y+1)<v(\hat{q}+1)$$

即　　$u<v(\hat{q}+1)$　　(19)

换句话说，$\hat{q}-1$ 作为商"太大了"。所以 $q\leqslant\hat{q}$。

另一方面，我们有

$$v=v_1b^{n-1}+v_2b^{n-2}+\cdots+v_n<b^{n-2}(v_1b+v_2-1) \tag{20}$$

从 $\hat{q}=x-y$ 可得：

$$(v_1b-v_2+1)(\hat{q}-2)=v_1bx+(v_2x-(v_1b+v_2)(y+1))+(x-y-v_1b-v_2-2)$$

由(11)(12)(15)式，从上式推出

$$(v_1b+v_2+1)(\hat{q}-2)<v_1bx\leqslant b(u_0b+u_1) \tag{21}$$

从(20)(21)式，得到：

$$v(\hat{q}-2)<b^{n-2}(v_1b+v_2-1)(\hat{q}-2)<b^{n-2}(u_0b^2+u_1b)\leqslant u$$

即　　$v(\hat{q}-2)<u$　　(22)

换句话说，$\hat{q}-2$ 作为商已经"够小了"，所以，$\hat{q}-2\leqslant q$。

当 $u_0=v_1$ 时，$x\geqslant b$ 成立，从而可能使 $xv_2\geqslant b^2$ 成立，即 xv_2 超出二进制 2w 位。此时，对 y 的计算在编程实现时稍作修改，仍可精确地求得 y 值。限于篇幅，本文不再介绍此修正算式及证明。

5　算法的复杂性分析与仿真结果

如上所述，三种算法的基本框架是完全一样的。不同之处在于：【算法 K】需要作初始时的

被除数、除数规范化操作和结束后的余数还原操作，但可充分利用计算机的字长和乘除指令，估商代价最低；【算法 X】或充分利用计算机字长，但以移位代替除法指令，估商代价增大；或利用除法指令，但以减小进制、增加循环次数为代价（后者未作仿真实现）；【算法 S】也可充分利用计算机字长和乘除指令，但估商算式复杂，代价增大。

三种算法的复杂度，都与被除数和除数的位数成正比，所以复杂度的阶是一样的。不同之处是不同的计算机字长和指令操作时间加在复杂度表达式上的系数。而这些是和具体的计算机及其字长、指令相关的，甚至和所使用的计算机语言及编译器相关。显然，当除数首位 v_1 远小于进制 b 时，【算法 K】在规范化和还原操作中就需要作多次的左移和右移，其速度将会受到明显的影响，而【算法 X】和【算法 S】则与除数的首位关系不大。

我们在(1)DEC 5900(CPU R4000)小型机，ULTRIX(BSD 4.3)操作系统所带 C 编译器；(2)DEC 5240(CPU R3000)工作站，ULTRIX(BSD 4.2)操作系统所带 C 编译器；(3)IBM PC 系列兼容机(486/33)，DOS 6.2 操作系统，BorlandC/C++3.1 编译器三种软硬件环境下进行了计算机仿真。仿真时，均采用 50000 对伪随机发生器生成的"64K 进制"的 40 位被除数和 20 位除数(相当于十进制 192 位左右的大数除以 96 位左右的大数)，并通过有意控制除数首位 v_1 的大小，结果都得到了与理论定性分析一致的仿真结果。下面所列的所有仿真结果都是在第(3)种环境下获得的。下表中，第一行显示的是【算法 K】规范化操作和还原操作移动的次数，表内的计时单位是秒。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
【算法 K】	104	109	113	118	123	126	130	134	137	142	145	150	153	157	161	165
【算法 X】	118	117	118	118	119	118	118	118	118	118	118	117	117	114	110	100
【算法 S】	106	121	114	116	118	117	118	118	118	118	117	118	117	116	114	109

从仿真结果看出，【算法 K】的时间随规范化和还原操作的复杂度单调递增，而【算法 X】和【算法 S】则基本保持不变。当移动次数较大时，后两者明显快于前者。在不对除数首位进行人为控制的情况下（这时除数首位服从均匀分布，平均移动次数为 1 次），三种算法在多次仿真下的平均时间依次为 109，118，110 秒。

6 结论

本文探讨了大整数相除的快速算法，提出了两种新的估商及修正算法，并与 Knuth 算法作了性能比较。在公开密钥体制 RSA 中，公开密钥(r，e)，其中 $r=p\times q$，p，q 为两个大素数。在加密、解密时，需要作大量的除数固定已知为 r 的大数除法。我们知道寻找合适的大素数 p，q 是很困难的，代价颇大。所以当 r 的首位较小时，采用本文所提出的新算法将有明显的优点。

参考文献

[1] Knuth，D. E. THE ART OF COMPUTER PROGRAMMING. Volumn 2/Seminumerical Algorithms，pp235～238. Addison-Wesley，1969.

[2] 吴永森、张芹，"公开密钥密码体制 RSA 算法的实现和应用"，计算机工程，Vol. 19，No. 2，pp28～32，1993.