

# 大数相除

2016年7月24日 15:57

## 大数除法

大数除法应该算是大数算术中比较复杂的一运算，通常都需要涉及到很多种大数运算，且比较高效的实现方式都非常的复杂。这里介绍一种容易理解但并非完美的实现方式。通常来说，这种方式有两个缺点：1. 比较耗费内存。2. 效率并不是很高。

这种方式是模拟我们手算除法的方式，先做一些规定。大数通常以字符串的方式保存。我们定义一个T来方便地表示char\* (即 typedef unsigned char\* T)，如后以 T x = "31429" 来表示92413。这样一个大数，我们约定低位在前，即个位是第1个字符。我们分析一下除法手工运算的过程。

$$\begin{array}{r} 1353 \\ 314 \overline{) 092413} \\ \underline{0628} \phantom{00} \\ 2961 \phantom{00} \\ \underline{2826} \phantom{00} \\ 1353 \phantom{00} \end{array}$$

$$\begin{array}{r} 1353 \\ 1256 \phantom{00} \\ \underline{\phantom{00}97} \phantom{00} \end{array}$$

这是被除数实际位数有5位，而除数的有效位数为3位，那么商最大有可能从  $5-3+1=3$  位，第3位开始。当第3位被除数比较小时，如12413，那么第3位即为0，一般来说不影响计算。(通过除数补0来避免+1，将被除数补成6位。) 根据手算除法的方式我们可以得到一种简单的计算除法的方式，那就是只取被除数与除数位数+1的数来做除法。如

除数与除数值加1的数来做除法。如  
 这是除数 314 为3位。那么每次取被除数  
 4位，即 0924, 2961, 1353, 来与除  
 数相除。那么不管被除数有多大，每一次  
 相除只需要4位。在一个INT类型的  
 数据中就能够存入，便可以计算。  
 用这种方法也解决了被除数是大数  
 的问题。但是我们如果遇到除数 314

也是一个大数时，那么就无法计算了。那么如何解决？  
 我们可以利用相同的思想，对除数也只取高几位。如  
 这是我们将高2位“31”，那么此时对应的除数为3位  
 即从092开始，则我们会有如下的计算过程。

$$\begin{array}{r}
 294 \\
 314 \overline{) 092413} \\
 \underline{0628} \phantom{3} \\
 2961 \phantom{3} \\
 \underline{2826} \phantom{3} \\
 1353 \phantom{3} \\
 \underline{1256} \phantom{3} \\
 97
 \end{array}$$

①  $092/31 = 2$   
 判断  $2 \times 314 < 0924$  ?  
 是那么就确定 2 为商  
 否则 2 大了，减去 1

② 从被除数中减去  $2 \times 314$

③ 接着求商 (相同方式)  
 即  $296/31 = 9$

$$\begin{array}{r}
 9 \times 314 = 2826 \\
 135/31 = 4 \\
 4 \times 314 = 1256
 \end{array}$$

在这种方法计算中，有两个问题  
 需要解决。第①在每一次求解  
 中需要做大数减法，314是  
 一个大数②要比较大数的大小。



还有一个问题就是我们用最高位来做除法，这是一种有损的除法，是有可能与实际不符合的。如我们按如上方式计算  $09000/109$  时，

$$090/10 = 9, \quad 9 \times 109 = 0981$$

$0981 > 0900$ ，说明 9 这个商大了。但是可以证明，这个商只可能比实际的商大 1。即  $9-1=8$  肯定是实际的商。所以  $8 \times 109 = 872 < 0900$ 。

x 为被除数，y 为除数，n 和 m 分别为被除数和除数字符串空间的总大小，（并非实际有几位数），q 为商的结果，r 为余数的结果，tmp 为存储被除数+临时商的空间的临时缓存，这些变量都需要自己申请空间，然后传入。这里会多耗费一个被除数+一个除数大小的空间（即 tmp）。

```
typedef unsigned char *T;
int XP_div(int n, T q, T x, int m, T y, T r, T tmp)
{
    int nx=n, my=m;
    n=XP_length(n, x); //求取被除数实际位数
    m=XP_length(m, y); //求取除数实际位数
    int k;
    unsigned char *rem=tmp, *dq=tmp+n+1; //用rem指向被除数第一个数，用dq指向结尾后两个数
    assert(2<=m&&m<=n);
    memcpy(rem, x, n); //将被除数赋值给rem
    rem[n]=0; //最高位补零，为方便计算
    //正式开始计算
    for(k=n-m; k>=0; k--) //商是从n-m位开始的，然后依此往低位移动 即k
    {
        int qk;
        int i;
        assert(2<=m&&m<=k+m&&k+m<=n);
        {
            int km=k+m;
            unsigned long y2=y[km-1]*BASE+y[km-2]; //除数最高两位的值 BASE此处为10，即10进制
            unsigned long r3=rem[km]*(BASE*BASE)+rem[km-1]*BASE+rem[km-2]; //从k+m开始一次后移1位的三位数的值，即被除数的
```

高三位

```
    qk=r3/y2; //被除数高三位/除数高二位
    if(qk>=BASE) //当出现大于10
        qk=BASE-1;
}
dq[m]=XP_product(m, dq, y, qk); //计算商为qk时 qk*y的大小存入dq 即商*除数y的大小 即dq=y*qk
for(i=m; i>0; i--) //判断dq是否比被除数小
    if(rem[i+k]!=dq[i]) //从最高位开始比较，当不相同时就退出
        break;
if(rem[i+k]<dq[i]) //退出后判断不相同的这一位是被除数rem大还是dq大
{
    dq[m]=XP_product(m, dq, y, -qk); //如果是dq大 说明商qk取大了，-qk，重新计算dq
}
q[k]=qk; //将商qk存下
{
    int borrow;
    assert(0<=k&&k<=k+m);
    borrow=XP_sub(m+1, &rem[k], &rem[k], dq, 0); //从除数中高k+m位减去dq 即目前的得到的余数的大小即rmk=rmk-dq
    assert(borrow==0);
}
//后移动一位重复这种计算。
}
```