PROGRAM AND ALGORITHM FOR LONG DIVISION

Falico Nicolae

Technical University of Moldova, Chisinau, Moldova

Abstract: In this paper effective and simple algorithm and program of long arithmetic are given. Program successfully finds residue and quotient for numbers containing more than thousand digits.

Key words: long arithmetic, algorithm.

Algorithms for long dividing are well known (see for example review in the fundamental work of Knut [1]). However, simple algorithms are not effective, they need much time for execution. The main difficulty in the effective algorithms is finding the correspondent digit of quotient. Some authors suggest finding this digit with the help of half diving [2] that simple for writing program, but the program becomes long and slow. Knut [1] very carefully proves simple formula that helps to find the good approximation for digits of quotient. This formula uses two digits of dividend and one of divisor. But in the following Knut [1] uses more complicated and more effective formula. His program written on MIX language contains more than hundred lines of code and not works when the divisor is one digit. In that case Knut suggests using another program. In this paper another formula is proved.

$$\textbf{*prez=}\frac{\textbf{(double)o*o* *pa1+o* *(pa1-1)+ *(pa1-2)}}{\textbf{o**pa2+*(pa2-1)+1}}$$

It is as effective as in Knut [1] but needs no normalization of dividend and divisor and renormalization of residue. The proof is given in the bottom of the paper and is not simple, but somebody can easily check the program using correspondent checking program and generating random inputs. We must mention that the close algorithms were suggested by another authors (see for example [3,4]). Program is written on C (the main function has only 20 operators) and is based on special structure for long numbers [2]. The long number is the array of integers, a[0] is the number of digits, a[1] is the lowest digit and a[a[0]] is the highest , and highest digit is followed by zero. For example, the number 123456789 with the base o=10000 is equal to a={3,6789,2345,1,0}. In compare with the programs from [2] written in Pascal and using the method of half dividing, the following code uses formula for finding digit. This makes program very fast. Moving pointers that is common for C makes program short and simple and in contrary to program Knut not needs no normalization of dividend and divisor and renormalization of residue. Program works when divisor contains one digit.

#define o 10000L

The base is defined. For 16-bits compiler maximum is 10000 because multiple of two 5 digit numbers is greater than MAXLONG.

void print_l(int *x,char *y){} void a_al(char *x,int *y){}

This functions converts numbers from the long mode to usual notation and backwards. They can be found anywhere.

```
int les(int *mid,int *a2){
        int i=*a2;
        if(mid[i+1])
return 0;
        while(i && mid[i]==a2[i])
i--;
        if(i==0)
return 0;
        else
return (mid[i]<a2[i]);
}
```

This function gets pointers to two long numbers and return 1 if the first number (*mid) is less than the second (*a2) and 0 in another case. Comparison begins from the highest ranks and works i=*a2 times that is equal to number of digits of (*a2). Function returns 0 if the number of digits of (*mid) is greater than (*a2).

```
void sub(int *a1,int q,int *a2){
        int i=1;
        while(i<=*a2 ){
                a1[i]-=(long)q * a2[i] % o;
                a1[i+1]-=(long)q * a2[i] / o;
                while(a1[i]<0){
                        a1[i]+=o;
                        a1[i+1]-=1;
                }
                i++;
        }
while(I<=*a1){
while(a1[i]<0){
                        a1[i]+=o;
                        a1[i+1]-=1;
                }
                i++;
}
}
```

This function gets pointers on two long numbers and digit. Number (*a2) is multiplied by q and is subtracted from (*a1) after working this program.

void d_l(int *a1,int *a2,int *rez){

```
/*1*/   int *prez,*pa1=a1 + *a1,*pa2=*a2+a2,*mid;
/*2*/   if(((*a1) – (*a2))>=0) {
/*3*/           mid=a1+*a1 - *a2;
*rez=*a1 - *a2 + 1;
}else{
/*4*/           *rez=1;rez[1]=0;
                return;
        }
/*5*/   prez=rez + *rez;
/*6*/   while(prez>rez){
/*7*/   if(*a2>pa1-mid || (*a2==pa1-mid)&&les(mid,a2)==1){
/*8*/   *prez--=0;
/*9*/                   mid--;                  //snosim sled cifru
/*10*/                  continue;
                }
/*11*/          if(*a2==pa1-mid ) {
                        (*a1)++;
/*12*/                  pa1++;                          //razriad
                }
/*13*/          *prez=( (double)  o * o * (*pa1) + o * (*(pa1-1)) +
               *(pa1-2) ) / ( o * (*pa2) + (*(pa2-1)) + 1 );
/*14*/          sub(mid,*prez,a2);
/*15*/          if (les(mid,a2)!=1){
/*16*/                  (*prez)++;
/*17*/                  sub(mid,1,a2);
                }
/*17*/          while(*pa1 == 0 && pa1>a1){
/*18*/                  pa1--;
              --*a1;    //nahozdenie chisla cifr v a1
                }
/*19*/ prez--;
       mid--;
        }
/*20*/ if (rez[*rez]==0)
            --*rez;
}
```

This is the main function. It gets three pointers on long digits. After it works in rez the quotient from dividing a1 on a2 will be written and in a1 will be residual. In line 1 pointers prez, pa1, pa2 on the highest ranks of numbers rez, a1, a2 are defined. Pointer mid is defined so that mid+1 is the position of digit in a1 from which the subtraction will begin. In operators 2-5 the initial values are written to that pointers and also initial number of digits of the result is calculated. This value may be changed in the operator 20. If comments operators 2-4 then when dividing, for example 1 on 1 cycle 6 will not work. In cycle 6 array rez is calculated from highest ranks. At first in 7-10 condition if the correspondent digit will be 0 is checked and than using shift mid—next digit of dividend is processed. Operator 13 is the core of suggested algorithm. In it three digits of highest ranks of dividend and two of divisor are used. In operator 11 the number of digits of dividend and divisor is checked and if needs the highest digit of a1 becomes 0 due to increasing number of digits on 1. In 12 the pointer to highest digit is changed. In 14 the subtracting from the position mid+1 in dividend is processed. In 15 the condition of continuing of dividing is checked and if yes than in 16 the result is increased on 1 and in 17 the subtracting is made. Because of that in a1 the number of digits is changed and in 17-18 the number of digits and pointer on the highest digit is changed. In 19 the pointer of the result is changed.

Let's give the proof of the correctness of algorithm. The following notations will be used α = Aa = A *10k+a   β = Bb=B*10k+b. Lets 0<=a,b<10k, A<10*B , B>=10 and A/(B+1)=Q.q. It is not difficult to show that Q* β< α. Lets proof that (Q+2)* β> α. This is equivalent to (A/(B+1)-0.q+2)* β> α. Or that is the same

(A+(1-0.q)*(B+1)+B+1) * β> α(B+1)   (1)

Let's proof the inequality From that will be followed (1).

(A+B+1)* β> α(B+1)   (2)

Inequality (2) is equivalent to(A+B+1)*(B*10k+b)> (A *10k+a)*(B+1) or

B2*10k+Bb+A*b+B*b<A*10k+a*B+a   (3)

Let's show that

B2*10k- A*10k-a>0   (4)

That is followed from a<10k и 10k*(B2-A)>= 10k*(10*B-A) >=10k, because 10*B-A is integer and A<10*B. Besides that

Bb = B*10k+b>a*B   (5)

Inequality (3) is followed from (4) and (5) .

REFERENCES:

1. Д.Э.Кнут., Искусство программирования т. 2 . Третье издание . Москва 2000 г.

2.С. Окулов , Программирование в алгоритмах, издательство "Бином. Лаборатория знаний" · 2002 г. · 344 стр.

3. George E. Collins, David R. Musser: Analysis of the Pope-Stein Division Algorithm. Information Processing Letters, 151-155 ,Volume 6, Number 5, October 1977.

4. E. V. Krishnamurthy, Salil K. Nandi:On the normalization requirement of divisor in divide-and-correct methods. Communications of the ACM (CACM), 809-813 Volume 10, Number 12, December 1967.