Рамиль Альварес Х.

Деление целых чисел в троичной симметричной системе

Алгоритмы деления в троичной симметричной системе описаны Брусенцовым Н.П. [1] в предположении, что значения делимого и делителя - нормализованные троичные числа, определенные в [2], т.е. равны нулю или их абсолютные величины принадлежат интервалу (0.5 ; 1.5). В работе [3] предложена модификация алгоритма Брусенцова и описан алгоритм извлечения квадратного корня. Описания алгоритмов в этих работах приведены на языке Алгол, т.е. не привязаны к конкретной системе счисления, и тем самым являются математическим описанием.

Брусенцов отмечает, что критерии, является ли очередная цифра частного нулем, в системах с неотрицательными цифрами и в троичной симметричной системе отличаются. В симметричной системе возможны два варианта критерия:

1) цифра частного есть нуль, если делимое не больше 1/2 делителя по абсолютной величине;
2) цифра частного есть нуль, если делимое меньше 1/2 делителя по абсолютной величине.

Автор считает первый вариант предпочтительным, так как чаще приводит к более компактной записи частного с требуемой точностью.

Отметим, что в алгоритме Брусенцова используется удвоенное значение делимого, из модуля которого на каждом шаге вычитается делитель и, если результат больше нуля, то цифра частного отлична от нуля и из результата снова вычитается делитель. Таким образом для выполнения деления требуется одно сложение (для получения удвоенного делимого), а число вычитаний равно числу разрядов частного плюс число ненулевых цифр в нем.

Предложенные в работе [3] модификации: отказ от удвоения делимого и использование половины делителя для сравнения с делимым, математические правильны, но они не работают при выполнении действий с ограниченным числом тритов. Если рассматривать делитель как целое, то точное значение его половины будет только при четном делителе. Этот алгоритм можно использовать при делении на 2, 8 и 10, как будет показано ниже.

Операция потритного сравнения

В работах [3,4] введена операция потритного сравнения значений, которая проще, чем сравнение с использованием вычитания с последующим определением знака результата. Вычитание выполняется, начиная с младших тритов с возможным переносом единиц (положительных или отрицательных) в старшие триты. Потритное сравнение производится слева направо до первых несовпадающих тритов, сравнение которых и определяет результат. В случае совпадающих значений результат сравнения равен нулю. Приведем пример выполнения потритного сравнения

```
+ - 0 - + + - - -
+ - 0 - - - + + +     >
```

Выделены первые несовпадающие триты + и – определяют результат >.

Описание алгоритма

Описание алгоритма приводится на языке C [5].

Целое троичное число представляется в виде структуры

```
struct trin
{ int n;
char trit [K];};
```

где n – число тритов, а массив trit – значение числа: младший трит хранится в элементе с индексом 0, а старшие – левее, элементы массива принимают значения –1, 0 и 1. Для числа нуль n равно нулю. K – параметр алгоритма.

Вспомогательные функции

Результатом функции sgn( ) является знак числа w. Если n равно нулю, то результат будет нуль, в противном случае – значение элемента массива trit с индексом n-1.

```
int sgn (struct trin w)
  {if (w.n == 0) return 0;
    else return w.trit[w.n-1];}
```

Результатом функции sgnk( ) с параметрами структура x и индексом k есть знак числа, представленного тритами структуры x с k-го по нулевой. При этом ищется первый слева направо ненулевой трит, который и определяет результат. В случае, когда все триты – нули, результат есть нуль.

```
int sgnk (struct trin x,int k)
{ int i=k, s=0;
     if (i >= 0)
       {while ( x.trit[i] == 0 && i-- >= 0);}
     if (i >= 0) s=(x.trit[i]);
return s;
}
```

Функция shift( ) с параметрами число w и тритом t определяет новое значение w. Если w не нуль, то производится сдвиг тритов влево, а триту с индексом нуль присваивается значение t. Если w равно нулю и t отлично от нуля, то триту с индексом нуль присваивается значение t. В случае, когда w и t равны нулю, w не меняется.

```
void shift (struct trin  *w, char t)
{ int j;
     if (w->n != 0)
      { for (j=w->n; j>0; j--)
           w->trit[j]=w->trit[j-1];
           w->trit[0]=t; w->n += 1;
      }
      else if (t != 0) {w->trit[0]=t; w->n=1;}
}
```

Функция cmp( ) с параметрами структуры x и y (при этом y>0) и sx – знак числа x производит потритное сравнение y и модуля x. Если длина структуры x больше длины y, то результат будет 1, а если длина x меньше длины y, то результат будет -1. При равенстве длин производится сравнение тритов числа x с учетом его знака (sx) и тритов числа y до первого несовпадения, которое и определяет результат. Если все триты совпали, то результат – нуль.

```
int cmp (struct trin x, struct trin y, int sx)
{ int i= x.n, s=-1;
     if (y.n<i) s=1;
     if (i == y.n)
     {i--;
     while (x.trit[i]*sx == y.trit[i] && i-- >= 0);
     if (i<0) s=0; else
     if (x.trit[i]*sx > y.trit[i]) s=1; else
     if (x.trit[i]*sx < y.trit[i]) s=-1;
     } ;
return s;
}
```

Функция red( ) производит удаление ведущих нулей структуры res, являющейся параметром функции. Она используется в функциях суммирования sum( ) и вычитания res( ) и в функции собственно деления dvs0( ).

```
void red ( struct trin * res)
{ int i;
for (i = res->n-1; i >= 0 && res->trit[i] == 0;
      i--);
res->n=i+1;
}
```

Описание функций суммирования sum( ) и вычитания res( ) не приводится из-за их очевидности. Параметрами функций являются структуры x и y - аргументы, а структура res – результат.

Основные функции

Функция dvs( ) с параметрами x – делимое, y – делитель и результатами: rs – частое и rs1 – остаток. В начале определяются удвоенное значение делимого (xx), знак (sy) и модуль (my) числа y и удвоенное значение модуля (myy), после чего производится обращение к функции собственно деления dvs0( ). В результате будет получено частное и удвоенный остаток (w1). Для получения самого остатка производится обращение к функции деления на 2 - dvs2( ).

```
void dvs (struct trin  x, struct trin y,
     struct trin * rs, struct trin *rs1)
// Деление x на y rs - частное, rs1 - остаток
{    struct trin xx, my,myy,w1;
     int i, k=y.n, sy=y.trit[k-1];
     sum (x,x,&xx);
     for (i=0; i<k; i++) my.trit[i]=sy*y.trit[i];
     my.n=y.n;
     sum (my,my,&myy);
     dvs0 (xx, my, myy,sy, rs,&w1);
     dvs2 (w1, rs1);
}
```

Аргументами функции собственно деления dvs0( ) являются структуры xx, my, myy и знак делимого sy. Результатом функции будет частное – структура rs и удвоенный остаток – структура rs1.

Возможно два вида обращения к функции dvs0( ):

1. xx - удвоенное делимое, тогда my – модуль делителя, myy - удвоенное my;

2. xx - делимое, тогда myy - делитель (> 0), my - половина myy. Это обращение может использоваться для деления на 2, 8 и 10, для которых половина делителя известна.

В основном цикле функции определяется значение очередного трита частного. Первоначально вычисляется новое значение числа w сдвигом влево удвоенного остатка и добавлением следующего трита удвоенного делимого. После этого производится сравнение модулей числа w и делителя my. Если модуль w больше my или если они равны и равны знаки w и необработанной части удвоенного делимого, то значение трита частного будет не нуль. В этом случае к числу w прибавляется или вычитается удвоенный делитель. После получения частного из его представления надо удалить ведущие нули.

```
void dvs0 (struct trin  xx, struct trin my,
  struct trin myy, int sy,struct trin *rs,
  struct trin *rs1)
{ int cwmy, i, sw, k;
  struct trin w ;
  rs->n=xx.n;
  for (i=xx.n-1; i>=0; i--) rs->trit[i]=0;
  w.n=0;
  for (i=xx.n-1 ; i>=0; i--)
    {shift (&w,xx.trit[i]);
    sw=sgn(w);
    cwmy = cmp(w,my,sw);
    if(cwmy == 1||(cwmy == 0&&sgnk(xx,i-1) == sw))
      {
    if (sw ==1) res(w,myy,&w); else sum(w,myy,&w);
    rs->trit[i]=sw*sy;
      }
    }
  red(rs);
  *rs1 = w;
}
```

Функция dvs2( ) является примером второго вида обращения к функции dvs0. В теле функции определены структуры m1, задающая число 1, и m2, задающая число 2.

```
void dvs2 (struct trin  xx,  struct trin *rs)
{struct trin m1, m2,w;
```

```
  m1.n=1; m1.trit[0]=1;
  m2.n=2; m2.trit[0]=-1;m2.trit[1]=1;
  dvs0 (xx, m1, m2,1, rs,&w);
}
```

Аналогично могут быть определены функции деления на 8 и 10.

Заключение

При выполнения предложенного алгоритма деления требуется одно сложение чисел порядка делимого (для вычисления удвоенного делимого) и суммирований или вычитаний чисел порядка удвоенного делителя. Число последних равно количеству ненулевых цифр частного плюс один (для вычисления удвоенного делителя).

Литература

1. Брусенцов Н.П. Алгоритмы деления для троичного кода с цифрами 0, 1, –1 // Вычислительная техника и вопросы кибернетики. Вып. 10. - Л.: Изд–во ЛГУ, 1974. С. 39–44.
2. Брусенцов Н.П., Маслов С.П. и др. Малая цифровая вычислительная машина "Сетунь". – М.: Изд–во МГУ, 1965.
3. Рамиль Альварес Х. Алгоритмы деления и извлечения квадратного корня в троичной симметричной системе // Вестн. Моск. ун-та. Сер. 15. Вычислительная математика и кибернетика. 2008. № 2. С. 42-45.
4. Рамиль Альварес Х. Троичный алгоритм извлечения квадратного корня // Программные системы и инструменты. Тематический сборник № 11. М.: Изд-во факультета ВМиК МГУ, 2010. С. 98-106.
5. Березин Б.И., Березин С.Б. Начальный курс С и С++. – М.: ДИАЛОГ МИФИ, 2005. – 288 с.

Исправления 11.01.12

```
// rs1->n=w.n;
// for (i=w.n-1; i>=0; i--)
// rs1->trit[i] = w.trit[i];
*rs1 = w;
```


Опубликовано: Программные системы и инструменты. Тематический сборник № 12. М.: Изд-во факультета ВМиК МГУ, 2011. С. 228-234.