УДК 519.6

Х. Рамиль Альварес

АЛГОРИТМЫ ДЕЛЕНИЯ И ИЗВЛЕЧЕНИЯ КВАДРАТНОГО КОРНЯ В ТРОИЧНОЙ СИММЕТРИЧНОЙ СИСТЕМЕ

(*лаборатория ЭВМ факультета ВМиК*, *e-mail*: *ramil@cs.msu.su*)

Предложена модификация алгоритма деления Брусенцова для троичной симметричной системы, основанный на использовании половины модуля делителя. Рассмотрен алгоритм извлечения квадратного корня в этой системе на базе известного метода в "столбик" и применения алгоритма деления.

В работах [1, 2], в которых описана созданная в 1840 г. английским изобретателем Томасом Фаулером троичная механическая вычислительная машина, приведены примеры деления в троичной симметричной системе. Однако в них не рассмотрены самые тонкие моменты процесса, вследствие чего их нельзя считать описанием алгоритма. Поэтому следует признать, что первые алгоритмы точного вычисления частного в троичной симметричной системе предложены Брусенцовым Н.П. в [3]. Алгоритмы деления предполагают, что значения делимого и делителя - нормализованные троичные числа, определенные в [4], т.е. равны нулю или их абсолютные величины принадлежат интервалу (0.5 ; 1.5).

1. Модификация алгоритма деления Брусенцова. Рассмотрим первый из предложенных алгоритмов [3] *division1*, результат которого выдается в виде пары чисел: нормализованной мантиссы e и порядка q, соответствующих представлению частного в виде $e \cdot 3^q$. Точность, с которой вычисляется мантисса e, задается целочисленным параметром k, указывающим количество троичных разрядов в представлении e.

procedure *division1* (*c*, *d*, *e*, *q*, *k*);
value *k*; **integer** *k*, *q*; **real** *c*, *d*; **integer array** *e*;
comment *c - делимое и d - делитель равны нулю или их абсолютные величины принадлежат интервалу* (0.5 ; 1.5). *Частное представляется посредством мантиссы e*[1 : *k*] *и порядка q в виде* $3^q \cdot \Sigma e_p \cdot 3^{1-p}$;
begin integer *p*; *se*, *sd*; **real** *g*;
prepare (*c*, *d*, *e*, *q*, *k*, *sd*);

```
comment Обнуление массива e. Удвоение делимого (c) и вычисление модуля (d) и
знака (sd) делителя d;
 for p := 1 step 1 until k
   begin if c > 0 then sc := 1 else sc := -1; g := c * sc - d ;
   if g > 0 then begin e[p] := sc * sd ; c := (g - d) * sc end;
   c := c * 3
   end
end division1;
```

Приведем описание процедуры *prepare*, которая заранее определяет значение порядка *q* и корректирует значение делимого *c* или делителя *d* так, что вычисляемая мантисса частного *e* получается нормализованной:

```
procedure prepare (c, d, e, q, k, sd);
value k; integer k, q, sd; integer array e; real c, d;
comment Оператор stop прерывает процедуру, если d=0;
  begin integer p;
  if d > 0 then sd := 1 else if d < 0 then sd := -1 else stop;
  d := abs(d); c := c + c;
  for p:= 1 step 1 until k do e[p] := 0;
  if abs(c) - d * 3 > 0 then begin q := 1; d := d * 3 end
     else if abs(c) - d > 0 then q:= 0
          else begin q := -1; c:= c * 3 end
  end prepare;
```

Отметим, что в алгоритме используется удвоенное значение делимого, из которого на каждом шаге вычитается делитель и результат сравнивается с нулем и при необходимости из результата снова вычитается делитель.

Предлагаемая модификация алгоритма (процедура *terndiv*) основана на том, что первоначально вычисляется половина делителя и в дальнейшем на каждом шаге

производится сравнение промежуточного делимого с этой половиной делителя и при необходимости из делимого производится вычитание полного делителя.

```
procedure terndiv (c, d, e, q, k);
value k; integer k, q; integer array e; real c, d;
comment c - делимое, d - делитель, равны нулю или их абсолютные величины
    принадлежат интервалу (0.5 ; 1.5). Частное представляется посредством
    мантиссы e[1 : k] и порядка q в виде 3^q·Σe_p·3^(1-p);
   begin integer i, sc, sd; real m;
    prepare1 (c, d, e, q, k, sd);
comment Обнуление массива e. Вычисление модуля (d) и знака (sd) делителя d.
   Вычисление m, равного половине модуля делителя d;
     for i := 1 step 1 until k do
       begin if c < 0 then sc := -1 else sc := 1;
           if compar(c, sc, m) > 0 then
comment compar - сравнение c и m с учетом знака sc, т.е. sign(c * sc - m);
           begin e[i] := sc * sd; c := c - sc * d end ;
           c := c * 3
       end
   end terndiv ;
```

Процедура *prepare*1 по сравнению с процедурой *prepare* включает вычисление половины модуля делителя. Следует отметить, что процедура *compar*(*c*, *sc*, *m*) – сравнение *c* и *m* с учетом знака *sc*, т.е. *sign*(*c* ∗ *sc* - *m*) сводится к поразрядному сравнению слева направо до первого несовпадающего, что существенно проще, чем вычитание.

```
procedure prepare1 (c, d, e, q, k, sd);
value k; integer k, q, sd; integer array e; real c, d;
comment Оператор stop прерывает процедуру, если d=0;
    begin integer p;
```

```
if d > 0 then sd := 1 else if d < 0 then sd := -1 else stop;
d := abs(d); media (d, m, k);
comment Вычисление m, равного d/2;
for p:= 1 step 1 until k do e[p] := 0;
if abs(c) - m * 3 > 0 then begin q := l; d := d * 3; m := m * 3 end
    else if abs(c) - m > 0 then q:= 0
            else begin q := -1; c:= c * 3 end
end prepare1;
```

Приведем описание процедуры получения половины делителя ($m = d/2$) для троичной симметричной системы. Алгоритм деления на 2 является частным случаем общего алгоритма деления. При этом учитывается то, что $d>0$ и не требуется вычислять половину делителя, так как это 1. Описание процедуры *media* (*d*, *m*, *k*) лучше привести в предположении, что операнд *d* и результат *m* являются массивами тритов.

```
procedure media (d, m, k);
value k; integer k; integer array d, m;
    comment Для d принадлежит интервалу (0.5 ; 1.5);
     begin integer i, sd, di;
    for i := 1 step 1 until k do m[i] := 0;
    di := 0;
    for i := 1 step 1 until k do
        begin di := di * 3 + d[i];
        if di < 0 then sd := -1 else sd := 1;
            if sd * di > 1 or (sd * di = 1 and sign (d, i+1) = sd) then
     comment Функция sign (d, i+1) определяет знака числа d, начиная с (i+1)-го раз-
        ряда;
                begin m[i] := sd;
                if sd > 0 then di := di - 2 else di := di + 2
                end
```

```
    end;
end media;
```

2. Алгоритм извлечения квадратного корня. Рассмотрим алгоритм извлечения квадратного корня известным методом в "столбик" [5]. Использование алгоритма деления в алгоритме извлечения квадратного корня позволяет упростить его.

Приведем алгоритм извлечения квадратного корня для системы счисления с основанием S и с неотрицательными значениями цифр от 0 до S–1. Предварительно подкоренное выражение N разбивается на пары цифр влево и вправо от запятой. Обозначим через N_i – целое число, получаемое из первых слева направо i пар цифр, A_i – целое значение корня из этого числа, D_i – i-я пара цифр, d_i – значение i-й цифры результата.

Алгоритм извлечения квадратного корня в "столбик" можно описать следующими формулами:

$$N_i = A_i^2 + B_i ,$$

$$B_1 = D_1 - d_1^2 ,$$

$$A_1 = d_1 ,$$

$$B_{i+1} = (B_i \cdot S^2 + D_{i+1}) - (2 \cdot A_i \cdot S + d_{i+1}) \cdot d_{i+1} ,$$

$$A_{i+1} = A_i \cdot S + d_{i+1} ,$$

где d_{i+1} – наибольшее значение, при котором B_{i+1} неотрицательно. Отметим, что действия производятся над целыми числами, а запятая в результате будет поставлена после k-й цифры, если в подкоренном выражении она стояла после k-й пары.

Приведенные формулы были использованы в [6] для разработки двоичного устройства извлечения квадратного корня.

Заметим, что $(B_i \cdot S^2 + D_{i+1})$ при ручном использовании алгоритма является приписыванием справа к значению B_i двух цифр пары D_{i+1}, а $2 \cdot A_i \cdot S + d_{i+1}$ и $A_i \cdot S + d_{i+1}$ – приписыванием цифры d_{i+1} к числам соответственно $2 \cdot A_i$ и A_i.

Для троичной симметричной системы (S=3) и аналогично тому, как это делается в алгоритме деления, сводя к выбору на каждом шаге между 0 и 1, формулы для вычисления квадратного корня можно записать в виде

$$C_{i+1}' = B_i \cdot 3^2 + D_{i+1} ,$$

$$C_{i+1} = |C_{i+1}'| - 2\cdot A_i\cdot 3\cdot d_{i+1} - d_{i+1}'^2 ,$$

где d_{i+1}' равно 0 или 1,

$$B_{i+1} = C_{i+1}\cdot \mathrm{sign}(C_{i+1}') ,$$

$$d_{i+1} = \mathrm{sign}(C_{i+1}')\cdot d_{i+1}' ,$$

$$A_{i+1} = A_i\cdot 3 + d_{i+1} ,$$

Формулу для C_{i+1} можно записать при d_{i+1}' ,равном 1, в виде

$$C_{i+1} = (|C_{i+1}'|-1) - 2\cdot A_i\cdot 3\cdot d_{i+1}' ,$$

Отсюда d_{i+1}' есть частное от деления $|C_{i+1}'|-1$ на $2\cdot A_i\cdot 3$, т.е. вследствие алгоритма деления в троичной системе d_{i+1}' равно единице, если

$$2\cdot(|C_{i+1}'|-1) \geq 2\cdot A_i\cdot 3,$$

или проще

$$|C_{i+1}'| > A_i\cdot 3.$$

Однако в случае $|C_{i+1}'| = A_i\cdot 3$ имеем

$$C_{i+1} = |C_{i+1}'| - 2\cdot A_i\cdot 3\cdot ½ - (½)^2$$

Поэтому, если n_{i+1} (остаток необработанной части числа N, рассматриваемой как число с запятой перед первой цифрой) больше ¼, то необходимо принять значение d_{i+1}' равным единице. Равенство невозможно, так как ¼ в троичной симметричной системе точно не представимо.

Описание процедуры вычисления квадратного корня приведено в предположении, что операнд *a* и результат *b* являются массивами тритов.

```
procedure ternsqrt (a, b, k, l);
value a, k, l; integer k, l; integer array a, b;
comment Вычисление квадратного корня a[1: k] - b[1: l], принадлежат интервалу (0.5 ; 1.5) или равны нулю и представляются посредством мантиссы m[1 : n]
в виде Σm_i·3^(1-i), m_1 = 1;
    begin integer i, j, ls, sc, g;
    ls := 0; b[1] := 1;
    for i := 1 step 1 until  l-1 do
        begin ls := ls * 9 ; j := 2 * i;
```

```
    if j ≤ k then ls := ls + a[j] * 3;
    if j+1 ≤ k then ls := ls + a[j + 1];
    if ls < 0 then sc := -1 else sc := 1;
    g := ls * sc - tern(i) * 3 ;
comment tern(i)- функция вычисления значения массива b, с 1-го по i-й разряды, как
    целого;
    if g > 0 or (g = 0 and trinary(i + 1) > 0.25) then
comment trinary(i)- функция вычисления остатка массива a, начиная с i-го разряда,
    как дробного;
        begin b[i + 1] := sc; ls := (g - tern(i) * 3) * sc -1 end
    else b[i + 1] :=0;
    end
  end ternsqrt ;
```

Реализация функций *tern*(*i*) и *trinary*(*i*) очевидна. Следует заметить, что выражение *trinary*(*i*+1)>0.25 может быть реализовано как последовательное сравнение пары *k*-го и (*k*+1)-го тритов массива *a* с 1$\underline{1}$ ($\underline{1}$ обозначает трит, равный минус единице) до первого несовпадения и знак этой разницы и будет значением выражения.

СПИСОК ЛИТЕРАТУРЫ

1. Glusker M., Hogan D., Vass P. The ternary calculating machine of Thomas Fowler // IEEE Annals of the History of Computing. 2005. **27**. N. 3. P. 4–22.
2. http://www.mortati.com/glusker/fowler/ternary.htm
3. 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.
4. Брусенцов Н.П., Маслов С.П. и др. Малая цифровая вычислительная машина "Сетунь". – М.: Изд–во МГУ, 1965.
5. Гусев В.А, Мордкович А.Г. Математика: Справочные материалы. М.: Просвещение, 1990.
6. Карцев М.А. Арифметика цифровых машин. М.: Наука, 1969.

Поступила в редакцию
10.09.07

Опубликовано в:
Вестн. Моск. ун-та. Сер. 15. Вычислительная математика и кибернетика. 2008. № 2. С. 42-45.
На английском языке:
J. Ramil Alvarez Algorithms of division and square root extraction in balanced ternary system // Moscow University Computational Mathematics and Cybernetics, Volume 32, Number 2, 2008. Pp. 103-107