ALGORITHM AND PROGRAM FOR DETERMING THE CUBE ROOT OF A LONG BINARY NUMBER

Authors: Nicolai FALICO, assoc. prof., PhD; Mihail KULEV, assoc. prof., PhD

Technical University of Moldova

Abstract*: In this paper an algorithm for finding the cube root of a long binary number has been described. The algorithm is based on the simple formula for the square root[4] of a number. The corresponding program, which uses a dynamic approach, has been presented.*

Keywords: *Algorithm, square root, long binary number, cube root, program in C language.*

1. Introduction

There is an algorithm that allows to find the cube root of a long number [1]. Theoretically there is no limit on the length of the number, it may consist of 100 or more digits. To begin with let us consider simple example of extracting the square root, This algorithm is similar to the algorithm of cubic extraction. At first we show "how the system works", and then explain why it works properly [2].

`

2. Algorithm for determing the square root

Take the number 56789 (the number is chosen randomly).

1. We divide its digits into pairs 5 67 89:
2. Extract the square root of the first group of digits from the left (it is clear that the exact root cannot be found, take the number whose square is as close to our number formed by the first group of digits, but does not exceed it). In our case it will be the number 2. Write a response – 2, that is the first digit of the root.
3. Take a number that is already in the answer. It is squared and subtracted from the first group of digits from the left. In our case remains 1.
4. To 1 we add the following group of two numbers and get 167. A number that is already in the answer 2, we multiply by 2, and get 4.
5. We need to find a number b with the following property – the product 4b by b should be as close as possible to 167, but no more than that number 167. In our case it will be 3. This is the next number in the decimal notation of our square root.
6. From the 167 we subtract 43*3 and receive 38.
7. Then repeat the same operations.

Calculations can be written as follows:

```
Square root (5 67 89 ) = 238
      4
      ------
      167
      129
       ------
        3889
        3744
        -------
          145
```

The algorithm is based on the formula:

$$(10a + b)^2 = 100a^2 + 20ab + b^2 = 100a^2 + (20a + b)b.$$

To take the square root, we need constantly subtract a number depending on intermediate results. It means the partition number into groups of two digits and subtract the square (step 2). Next, to find the next digit of result, we multiply the result by 2, assign zeroes, and we add b and b is multiplied by the amount (for example, in steps 4 and 5).

The hardest part of the algorithm is to determine the number b, we intuitively pick up of 10 digits, and the result is obtained immediately [2, 3, 4]. If the original number is long, around 50 digits or more, finding b became harder and harder. Therefore it is reasonable to go to the binary system, where there are only two choices for b, zero and one. This greatly simplifies the computation.

3. Algorithm for determing the cube root

Let us go back to the cube root . Let **x** is a string of zeros and ones. We assume that **x** is a binary number. The algorithm is based on the formula of the sum of the cube, where a is the result, and b is one or zero.
The first time we subtract 1=13, and multiplication by a thousand means a grouping of three digits. Then we should take the remainder of the formula to introduce this variable tmpx, which is the sum of the last three terms, that is 100 * 3 * rez2 * b + 10 * 3 * rez * b2 + 1. The binary code of 3 is 11.
To extract the cube root of **x**, it is necessary to end split **x** into groups of 3 digits (i.e., from the point). Consider the first group of digits (the last, counting from the decimal point). Regardless of what is in this group, the first digit of the result is always one. Zero cannot be, since the root cannot be zero (because the **x** always starts with the unit). Then one of the results, we take a cube, noting that the unit in any power remains a unit, and zero in any power is zero. Decreasing first group of numbers on one, to the difference write the next group of numbers (3 digits).
Variable tmpx must be calculated for b = 1. The initial value tmpx (when the result is equal to 1) is equal to 10011. If you can decrease the rest on tmpx, then write in result unit, the difference write to the rest, and we carry the following numbers **x**. If you cannot decrease, the result is zero, rest remains the same and carries the next group of numbers in the end of the rest.
Then, we continue these operations. The program counts cube root from binary number.

4. Program in C language for determing the cube root

```
#include<stdio.h>
#include<string.h>

const  int N=8000;
struct bi{ int l, f, a[2*N]; };

int lcmp(bi *a,bi *b)
{
        int i=a->f,j=b->f;
        if(a->l>b->l)  return 1;
        if(a->l<b->l)  return -1;
        while(i<a->l+a->f ) {
        if( a->a[i]>b->a[j]) return 1;
        else if( a->a[i]<b->a[j])  return -1;
        i++; j++;
        }
        return 0;
}

void minus(bi *c,bi *b)
{
        bi r={0};
        int p2[2][2][2]={0,1,0,0,1,1,0,1},p1[2][2]={0,0,1,0},
                r2[2][2][2]={0,1,1,0,1,0,0,1},r1[2][2]={0,1,1,0};
        int i=c->f+c->l-1,j=b->f+b->l-1,per=0,tmp;
        while(j>=b->f ){
                tmp=c->a[i];
                c->a[i]=r2[per][c->a[i]][b->a[j]];
                per=p2[per][tmp][b->a[j]];
                i--;j--;
        }
        while(i>=c->f && per ){
                tmp=c->a[i];
```

```
                c->a[i]=r1[per][c->a[i]];
                per=p1[per][tmp];
                i--;
        }
        if(i<c->f){
                i++;
                while(c->a[i]==0 && i<c->f+c->l)  i++;
                c->l=c->l-(i-c->f);
                c->f=i;
        }
}

void lcat (bi *a,bi *b)
{       int i=b->f,j=a->l+a->f;
        a->l+=b->l;
        while(i<b->l+b->f)  a->a[j++]=b->a[i++];
}

void lsum(bi *a,bi *b)  // Sum of 2 numbers
{       int p2[2][2][2]={0,0,0,1,0,1,1,1},p1[2][2]={0,0,0,1},
        r2[2][2][2]={0,1,1,0,1,0,0,1},r1[2][2]={0,1,1,0},tmp;
        int i=a->f+a->l-1,j=b->f+b->l-1,per=0,last=i;
                while(i>=a->f && j>=b->f){
                tmp=a->a[i];
                a->a[i]=r2[per][a->a[i]][b->a[j]];
                per=p2[per][tmp][b->a[j]];
                i--;j--;
        }
        while( j>=b->f){
                a->a[i]=r1[per][b->a[j]];
                per=p1[per][b->a[j]];
                i--;j--;
        }
        while(i>=a->f && per){
                tmp=a->a[i];
                a->a[i]=r1[per][a->a[i]];
                per=p1[per][tmp];
                i--;
        }
        if(i>=a->f)  i=a->f-1;
        if(per)  a->a[i--]=1;
        a->f=++i;
        a->l=last+1-a->f;
}

int main()
{
        char s[N];
        bi  x={0},rez={0},ost={0},tmpx={0},r={0},
s10011={5,N,{0}},s11={2,N,{0}},s00={2,N,{0}},s0={1,N,{0}},s000={3,N,{0}};
        int lenx,i,j,ix,lenr,ir,io,fl,fl1;
        FILE *in=fopen("zzz.txt","r");
        FILE *out=fopen("11ini.txt","w");
        s10011.a[N]=s10011.a[N+3]=s10011.a[N+4]=1;
        s11.a[N]=s11.a[N+1]=1;
        fscanf(in,"%s",s);
        x.l=lenx=strlen(s);
```

```
for(i=0;i<lenx;i++)  x.a[i+N]=s[i]-'0';
tmpx=s10011;
lenr=lenx/3;  if (lenx%3) lenr++;
rez.a[N]=rez.l=1;rez.f=N;
ix=ir=1;
io=N;ost.f=N;
if(lenx%3==0) {
        ost.a[io++]=1;ost.l++;
        if(x.a[1+N]==1)
                if(x.a[2+N]==1)  {ost.a[io++]=1; ost.a[io++]=0;ost.l+=2;}}
                else {ost.a[io++]=0; ost.a[io++]=1;ost.l+=2;}
        else if(x.a[2+N]==1) {ost.a[io++]=0; ost.a[io++]=0;ost.l+=2;}
                 else {ost.a[io++]=1;ost.l++;};
                ix=3;
}
if (lenx%3==2)
ost.a[io++]=1;ost.l++;
        if(x.a[1+N]==1){
                ost.a[io++]=0;
                ost.l++;
        }
        ix=2;
}
while(ir<lenr) {
        io=ost.l;
        if(io==0){
                if(x.a[ix+N]==0){
                        ix++;  x.l++;
                        if(x.a[ix+N]==0){
                                ix++; x.l++;
                                if(x.a[ix+N]==0){
                                        ix++;x.l++;
                                }else{
                                        ost.a[io++ +ost.f]=x.a[ix++ + N];
                                        ost.l++;x.l++;
                                }
                        }else {
                                ost.a[io++ + ost.f]=x.a[ix++ +N];
                                ost.a[io++ + ost.f]=x.a[ix++ + N];
                                ost.l+=2;x.l+=2;
                        }
                }else{
                        ost.a[io+ost.f]=x.a[ix+N];
                        ost.a[io+1+ost.f]=x.a[ix+N+1];
                        ost.a[io+2+ost.f]=x.a[ix+N+2];
                        io+=3;ix+=3;
                        ost.l+=3;x.l+=3;
                }
        }else{
                ost.a[io+ost.f]=x.a[ix+N];
                ost.a[io+1+ost.f]=x.a[ix+N+1];
                ost.a[io+2+ost.f]=x.a[ix+N+2];
                io+=3;ix+=3;
                ost.l+=3;x.l+=3;
        }
        if (ost.l==tmpx.l)
                if (lcmp(&ost,&tmpx)>=0){
```

```
                        minus(&ost,&tmpx);
                        rez.a[ir+N]=1;
                        ir++;rez.l++;
                        }
                else {rez.a[ir++ + N]=0;rez.l++;}
        else if (ost.l>tmpx.l) {
                        minus(&ost,&tmpx);
                        rez.a[ir+N]=1;
                        ++ir;rez.l++;
                        }
                else {rez.a[ir++ +N]=0;rez.l++;}

        if (rez.a[--ir+N] ==0){
                r=rez;
                r.a[ir+N]=0;r.l--;
                lcat (&tmpx,&s00);
                minus (&tmpx, &s11);
                lcat(&r,&s00);
                minus (&tmpx,  &r);
                lcat(&r,&s0);
                minus(&tmpx, &r);
                rez.a[ir+N]=0;
                }
        else {
        r=rez;
        r.l--;
        lsum(&tmpx,&r);
        lsum (&tmpx, &s11);
        lcat(&r,&s000);
        lsum(&tmpx, &r);
        lcat (&tmpx,&s11);
        rez.a[ir+N]=1;
        }
        io=ost.l-1;
        ir++;
    }
    for(i=rez.f;i<rez.l+rez.f;i++)
            fprintf(out,"%d",rez.a[i]);
    return 0;
}
```

Bibliography:

1. Окулов С. М.Программирование в алгоритмах / С. М. Окулов. —М.: БИНОМ. Лаборатория знаний, 2002. — 341 с: ил.ISBN 5-94774-010-9
2. https://ru.wikipedia.org/wiki/Квадратный_корень
3. Алгоритмы вычисления квадратного корня. http://www.azillionmonkeys.com/qed/sqroot.html
4. Соловьев Ю., Старый алгоритм. http://kvant.mccme.ru/1987/03/staryj_algoritm.htm