COL226: Assignment 1 Report

-Rajat Golechha

Table of Contents:

I. Algorithm : Contains the pseudo code for the used functions in the code.
II. Proof of Correctness : Contains the proof for the algorithm to find square root of an arbitarily large number.

Sunday,22 January 2023

Algorithm

Legend

Symbol	Meaning
@	Concatenate
hd	Head
tl	Tail

A) *fun length* { *If nil: return 0*; *Else: return 1+length(tl)*

B) *fun reverse* { *If nil : return nil*; *Else : return reverse (tl) @ [hd]*

C) *fun addzero (L,x)* { *If x<=0 : return L*; *Else : return addzero ([0]@L,x-1)*

D) *fun subtr (v1,v2,v3,n,carry)* { *If n=0: return v3*; *Else* { *If hd(v1) < hd(v2) + carry : return subtr(tl(v1),tl(v2),[10+hd(v1)-hd(v2)-carry]@v3,n-1,1)*; *Else : return subtr(tl(v1),tl(v2),[hd(v1)-hd(v2)-carry]@v3,n-1,0)*

E) *fun sublist (v1,v2)* { *Return subtr(reverse(v1),reverse(add_zero(v2,length(v1)-length(v2))),[ ],length(v1),0)*

F) *fun addr (v1,v2,v3,n,carry)* { *If n=0: return v3*; *Else* { *If hd(v1)+hd(v2)+carry > 9 : return addr(tl(v1),tl(v2),[hd(v1)+hd(v2)+carry-10]@v3,n-1,1)*; *Else : return (tl(v1),tl(v2),[hd(v1)+hd(v2)+carry]@v3,n-1,0)*

G) *fun addlist (v1,v2)* { *If length(v1)>length(v2) : addr(reverse(add_zero((v1),1)),reverse(add_zero((v2),length(v1)-length(v2)+1)),[ ],length(v1)+1,0)*; *Else : return addr(reverse(add_zero((v2),1)),reverse(add_zero((v1),length(v2)-length(v1)+1)),[ ],length(v2)+1,0)*

H) *fun multplr (v1,v2,x,n,carry)* { *If x=0 : return carry@v2*; *Else : return multplr (tl(v1),[(hd(v1)*n+carry) mod 10]@v2,x-1,n,(hd(v1)*n+carry) div 10)*

I) *fun multiplier (v1,n)* { *Return multplr(reverse(v1),[ ],length(v1),n,0)*

J) fun ldngzrorem (v1) { If length(v1) <= 1 : return v1
Elif hd(v1)=0 : return ldngzrorem(tl(v1))
Else : return v1

K) fun gte (v1,v2) { If length(v1)>length(v2) : return true
Elif length(v1) < length(v2) : return false
Else : { If hd(v1)>hd(v2) : return true
Elif hd(v1) < hd(v2) : return false
Else: { If length(v1)=1: return true
Else : return gte(tl(v1) , tl(v2))

L) fun converter { Convert string to int list

M) fun convert_back { Convert int list back to string

N) fun sqrt (v1,left,remd,otpt,n) {
If n=0 : return (reverse(otpt) ,remd)

Elif remd+(2 new terms from dividend) >= 9*(left*10+9) : return sqrt(tl(tl(v1)), 10*left+18,remd+2terms from dividend - 10*left*9+81 , [9]@otpt,n-1)

Elif remd+(2 new terms from dividend) >= 8*(left*10+8) : return sqrt(tl(tl(v1)), 10*left+16,remd+2terms from dividend - 10*left*8+64 , [8]@otpt,n-1)

Elif remd+(2 new terms from dividend) >=7*(left*10+7) : return sqrt(tl(tl(v1)), 10*left+14,remd+2terms from dividend - 10*left*7+49 , [7]@otpt,n-1)

Elif remd+(2 new terms from dividend) >= 6*(left*10+6) : return sqrt(tl(tl(v1)), 10*left+12,remd+2terms from dividend - 10*left*6+36 , [6]@otpt,n-1)

Elif remd+(2 new terms from dividend) >= 5*(left*10+5) : return sqrt(tl(tl(v1)), 10*left+10,remd+2terms from dividend - 10*left*5+25 , [5]@otpt,n-1)

Elif remd+(2 new terms from dividend) >= 4*(left*10+4) : return sqrt(tl(tl(v1)), 10*left+8,remd+2terms from dividend - 10*left*4+16 , [4]@otpt,n-1)

Elif remd+(2 new terms from dividend) >= 3*(left*10+3) : return sqrt(tl(tl(v1)), 10*left+6,remd+2terms from dividend - 10*left*3+9 , [3]@otpt,n-1)

Elif remd+(2 new terms from dividend) >= 2*(left*10+2) : return sqrt(tl(tl(v1)), 10*left+4,remd+2terms from dividend - 10*left*2+4 , [2]@otpt,n-1)

Elif remd+(2 new terms from dividend) >= 1*(left*10+1) : return sqrt(tl(tl(v1)), 10*left+2,remd+2terms from dividend - 10*left*1+1 , [1]@otpt,n-1)

Else: return sqrt(tl(tl(v1)) , left @ [0] , remd+2 terms from dividend , [0] @ otpt , n-1)

O) fun isqrtld (v1){If length is odd: convert to int list , and add zero to left { Return sqrt(int list, [0], [0], [ ] , length(int list) / 2)

Proof of Correctness

To Prove : The method of finding square roots by long division produces correct outputs for arbitarily large numbers.

Proof Method : Proof by induction

Induction Hypothesis: Let the method be valid for numbers of length less than equal to 'k'.

Base Cases :

N = 1

Consider any number of length 1, we know that using the long division method we will get integers q and r for the given number such that :

*Num = q*q + r and N < (q+1)*(q+1) and N > (q−1)*(q−1) (By definition)*

N = 2

Consider any number of length 2, we know that using the long division method we will get integers q and r for the given number such that :

*Num = q*q + r and N < (q+1)*(q+1) and N > (q−1)*(q−1) (By definition)*

Induction step:

N = k + 1

In the process of finding square root by long division,
We know that every time we add some new digit to the quotient, we add it twice in the existing divisor and multiply the whole thing by 10 before moving ahead thereby ensuring that we add 20 times the number in divisor that we add in the quotient, but we also multiply the old divisor by 10 too, and we do the same for quotient also by inserting a new digit.
Therefore we can say that the relation that current divisor = 20 times quotient remains unaffected by this.
We are going to use this fact in our proof ahead . . .

Let the above mentioned results be valid for all numbers of length <= k.

Consider the numbers of length 'k+1' of the form $a_k a_{k-1} a_{k-2} \ldots a_2 a_1 a_0$ we know that the current algorithm will give us integers q' and r' such that for N' = $a_k a_{k-1} a_{k-2} \ldots a_2$

*Num' = q'*q'+r' and N < (q'+1)*(q'+1) and N > (q'-1)*(q'-1)* (By definition)

*And N=100*N' + 10*a_1 + a_0, where we also know that the last number on divisor would have been 20*q' and therefore there would have been integers s and t such that new divisor = 20*q' + s while t = 100*r' + 10*a_1 + a_0- new divisor*s* (Euclid's division Algorithm)

*Therefore now new quotient = 10*q' + s*

*New remainder = t = 100*r' + 10*a_1 + a_0 - (20*q' + s)*s*

Now calculating quotient^2 + remainder we get,

*=> (10*q' + s)^2 + 100*r' - (20*q' + s)*s*

*=> 100*q'*q' + s*s + 20*q'*s + 100*r' + 10*a_1 + a_0 - 20*q'*s - s*s*

*=> 100*q'*q' + 100*r' + 10*a_1 + a_0*

*=> 100*N' + 10*a_1 + a_0*

=> N

Therefore P(1),P(2) . . . P(k) => P(k+1)

Q.E.D.

- For the Algorithm

Now, that we proved that the given algorithm works , we will prove that the algorithm written in the pseudocode also implements the same.

Proof by Induction

The algorithm in pseudocode first adds 0 if the number Is odd, else continues

*Once "0" is added the number becomes of length 2*k ;*

*IH : Let the algorithm be working upto numbers of length 2*k ;*

Base Case : We know that the algorithm works for all single and two digit numbers since our if else varies over all numbers from 0 through 9 and no one/two digit number has a quotient greater than them.

*Induction Step : Now we assumed that the function would evaluate the correct square root and remainder for upto 2*k length of numbers let there be a number of length 2*k+1 we know that the algorithm will add a zero to it and make it 2*k+2 now we also know that for the leftmost 2*k digits we can get the correct integer square root and a remainder, and for the remaining 2 digits we know that by earlier proof that the long division algorithm would generate unique integers such that we get the correct quotient and remainder even for the bigger number.*

Therefore P(1),P(2) . . . P(k) => P(k+1)

Q.E.D.

Now, some lemmas ;

1) *Termination of the algorithm : We used a counter in the algorithm to count the number of steps (s) and they are exactly equal to the number of two digit pairs in the number. And we initialize the counter to it and decrement the counter every time by 1 until it reaches 0 so we have exactly "s" steps and then we get the output.*
2) *All cases are covered : We used an multi-nested-if-else in the algorithm with all digits from 9 downto 0, thereby ensuring two things only the largest possible number was used and since all digits are covered it was also ensured that we get a digit in any case of Quotient and remainder.*