Indian Institute of Technology Delhi

COL226 Programming Languages: Assignment 1

Ishita Chaudhary: 2019CS10360

Contents

Problem Description

The problem requires us to suggest an algorithm to find the integer square root of a given positive integer using the long-division method. Further, we need to prove the correctness of the algorithm and implement it in SML. The code needs to be purely recursive.

The integer square root, b, of a positive integer, a, is defined as follows:

$$b^2 \leq a < (b+1)^2$$

Since the input can be arbitrarily large, it is given as a string of digits, to avoid overflow. The attached script "isqrtld.sml" takes a string of digits as input (in decimal form) and returns a tuple of strings (b, r), where r is the remainder defined as

$$r = a - b^2$$

. The remainder will be non-zero if the input is not a perfect square.

Algorithm and Pseudo Code

Given an input string A_n of arbitrary length n, as a sequence of digits $a_1a_2...a_n$. We follow the long-division method to find the integer square root 'b' of A_n such that b is the largest possible integer $b^2 \leq A_n$. We follow the steps below:

1. If n is odd, we add "0" to the left of the string. Initiate the quotient to 0.
2. Divide the string into pairs of digits and start traversing these pairs from the left hand.
3. Iterate from 0 to 9, to find the largest integer whose square is less than or equal to the pair of digits. Take that as the divisor. Append the same to the quotient. Subtract the square of the divisor from the first two pairs of digits to get the dividend.
4. Append the next pair of digits to the remainder. Now, find the largest possible digit i by traversing from 0 to 9, such that appending this digit i to the double of the current quotient becomes our divisor, and multiplying our divisor with i is just less than or equal to the dividend. Append this digit i to the quotient. Subtract the product of divisor and i from the dividend and keep repeating this step until we reach the end of the string.
5. The quotient is our integer square root 'b'.

```
Function IntegerSquareRoot(digit sequence $A_n$):
    if length $A_n$ is odd then
        $A_n$ = "0" + $A_n$
    end
    return LongDivision($A_n$, 0, 0, 0)
```

```
Function LongDivision(digit sequence $A_n$, divisor, dividend, quotient):
    if $A_n$ is empty then
        return quotient
    end
    firstElement := $A_n[1]$
    secondElement := $A_n[2]$
    $A_{n-2}$=$A_n[3:]$ /* updated seq of digits after removing the first pair  */

    dividend = dividend*100+firstElement*10+secondElement
    newDigit:=0
    forall $i \in [0, 9]$ do
        if dividend-(divisor*10+i)*i $\geq$ 0 then
            newDigit=i
            remainder=dividend-(divisor*10+i)*i
        end
    end
    quotient=quotient*10+newDigit
    divisor=quotient*2
    dividend=remainder
    return LongDivision($A_{n-2}$,divisor, dividend, quotient)
```

Complexity Analysis: The code attached in file "isqrtld.sml" is a purely recursive function, without using any imperative constructs of SML, which returns the integer square rot and remainder as a tuple of strings. First, we analyze the complexity of the *LongDivision* function.

The initialization of data, slicing of string, and arithmetic checks take O(1) time. Further, the for loop for newDigit takes 10*O(1) time as there are always ten elements to iterate. Now, the recursive calls the function itself on an input with length n-2 where n is the length of the initial input, i.e., T(n)=T(n-2)+10*O(1)+constant. The function *IntegerSquareRoot* only calls the function *LongDivision*.
Hence, the complexity of both algorithms can be determined by simplifying the expression T(n)=T(n-2)+10*O(1)+constant. This can be simplified to T(n)=(number of times the recursion is called)*10*O(1)+ constant. The recursion will be called for every pair of digits, and the number of pairs is $log_{100}n$. Hence, the time complexity is O($10log_{100}n$), where n is the number of digits in the input string.

Proof of Correctness

We will prove the correctness of the above algorithm in this section. Let the input number A_n be a sequence of digits $a_1a_2...a_{n-1}a_n$, where n denotes the number of digits in A_n.

Claim: P(n):= The algorithm described in the previous section returns the integer square root, b, of an arbitrarily large number A_n with n digits, where b is the largest possible integer such that $b^2 \leq A_n$.
Proof: Proof is by Induction
Base Case: For n $\leq 2, i.e., if$ A$_n \in \{1, 99\}$.
In our algorithm, we take every pair of numbers from the left (in case n is odd, we add a zero to the beginning of the input string) and iterate over 0 to 9 to find the largest integer b such that $b^2 \leq A_n$ is satisfied. The largest possible input in the base case is 99, which gives us the correct output of 9 in one iteration over 0 to 9. Hence, P(1) and P(2) hold true.
Induction Hypothesis: Assume the claim holds true for P(m), such that m $\geq$ 1.
Induction Step: We will prove the correctness of P(m+2).
From our induction hypothesis, we assume the algorithm gives us the largest integer b' on input A_m, where $A_m = a_1a_2...a_m$, such that

$$(b')^2 \leq A_m$$

where A_m can be written as

$$A_m = a_1 10^{m-1} + a_2 10^{m-2} + ... + a_{m-1}10 + a_m$$

substituting this in above equation we get

$$(b')^2 \leq a_1 10^{m-1} + a_2 10^{m-2} + ... + a_{m-1}10 + a_m \qquad ...(1)$$

Now, our algorithm suggests that for every pair of digits, we encounter while traversing our input from the left hand, we find the largest digit i (by iterating over 0 to 9) such that on appending this digit i to the double of the current integer square root when multiplied with this digit i gives us the largest possible integer which is less than or equal to the current remainder appended by the next pair of digits (our new dividend). In other words,

$$NewDividend - (2 * CurrentIntegerSquareRoot + append(i)) * i \qquad ...(2)$$

gives us the smallest possible remainder.
To prove P(m+2), we can write A_{m+2} as $a_1a_2...a_m a_{m+1} a_{m+2}$, and consequently

$$A_{m+2} = a_1 10^{m+1} + a_2 10^m + ... + a_{m+1}10 + a_{m+2}$$

where $a_{m+1}a_{m+2}$ are the rightmost pair of digits. We can express A_{m+2} as

$$A_{m+2} = 100A_m + 10a_{m+1} + a_{m+2} \qquad ...(3)$$

Now, for this newly appended pair of digits, the current integer square root is b' and the current remainder is $A_m - (b')^2$ concatenated with the last pair of digits. We can rewrite the equation (2) as, choose the largest possible digit i such that

$$(2b' * 10 + i) * i \leq 100 * (A_m - (b')^2) + 10a_{m+1} + a_{m+2}$$

on substituting equation (3), the expression becomes

$$(2b' * 10 + i) * i \leq A_{m+2} - 100 * (b')^2$$

rearrange the equation to get

$$(10b' + i)^2 \leq A_{m+2}$$

The above equation suggests that we have chosen the digit i such that 10b'+i is the integer square root for A_{m+2}, where b' is the integer square root of A_m. Our algorithm says for every new pair of digits, if we append the largest possible i to the current integer square root, which satisfies equation (2), we get our new integer square root. Hence, P(m) $\implies$ P(m+2).

Since P(1) and P(2) hold true, and we've proved that P(m) $\implies$ P(m+2) for any m $\geq$ 1, P(n) holds true $\forall$ n $\geq$ 1. Hence, proved.

Test Cases

The code was tested on the following test cases for correctness. We varied the length of the input string and on perfect and non-perfect squares.

Test Case:1
Input String: "3578210349689"
Output: ("1891615", "3041464")
Test Case:2
Input String: "35783397225"
Output: ("189165", "0")
Test Case:3
Input String: "357839790397225"
Output: ("18916653", "29674816")
Test Case:4
Input String: "81"
Output: ("9", "0")
Test Case:5
Input String: "999999999999999"
Output: ("31622776", "38053823")
Test Case:6
Input String: "1665770076568787"
Output: ("40813846", "51257071")

Acknowledgements

I acknowledge that I've consulted the below resources while doing this assignment. I oblige to the course policy and have not copied any part of the code from the below/ any other resources that I found on the Internet.
List of resources:
1. https://www.smlnj.org/doc/literature.html#tutorials
2.https://en.wikipedia.org/wiki/Methods_of_computing_square_roots
3. https://www.geeksforgeeks.org/long-division-method-to-find-square-root-with-examples/