COL226 Assignment 1

Pravar Kataria (2021CS10075)

January 22, 2023

§1 Description

Given a string s (value to be treated as an integer value) ,the algorithm is expected to find two values (datatype string) *quotient* and *remainder*. Which satisfy the following conditions :

1. $quotient^2 \leq s < (quotient + 1)^2$
2. $quotient^2 + remainder = s$

It can be shown that for any non-negative value in string s, there exists unique values for *quotient* and *remainder*.
This can be shown as *quotient* is the largest number whose square is less than or equal to s and from (2) condition, uniqueness of *remainder* follows.

§2 Pseudo-code

The implementation adopted is a **recursive** one.
The implementation works only for string of even length. For strings of odd length a leading 0 is added so that the 'value of input' remains unchanged but the length of the string becomes even.
The main function ,say f, takes in 4 arguments:

1. The string itself : s (whose integer square-root is to be calculated using the algorithm).
2. A counter (say i) , basically the position in the string currently into consideration. [initially set to 0]
3. A *remainder* variable holding the current value of the remainder.[initially set to 0]
4. A *quotient* variable holding the current value of the quotient.[initially set to 0]

Notes :

- The algorithm terminates when the counter i reaches the end of the string
- When the algorithm terminates a tuple is returned whose first value is the quotient variable and whose second value is the remainder variable.

The algorithm runs as follows.

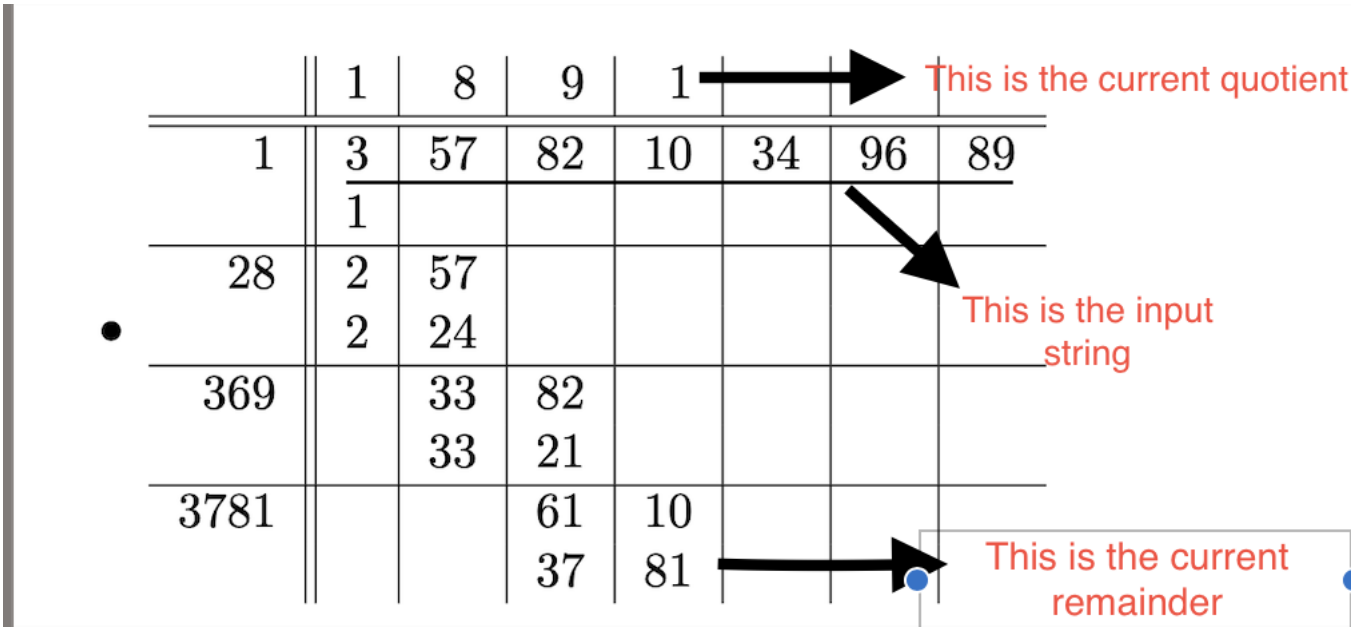

Figure 1: A visual description of variables

1. If the counter reaches the end, the algorithm terminates.
2. Otherwise , the next 2 values in the string are 'loaded' into the current remainder (or appended in front of the current remainder).
3. A single digit integer x is found by iterating through all values from 0 through 9, such that x is the largest value satisfying .
   $(20 \cdot quotient + x) \cdot x < remainder$
4. Then x is appended to the quotient and also $(20 \cdot quotient + x) \cdot x$ is subtracted from the $remainder$.
5. The counter i is then increased by 2 to get the next index. and the algorithm is repeated with new values.

Notes :

- The algorithm is guaranteed to be terminated as the value of counter i increases with every step and terminates after a finite number of steps when i reaches the end.

CHANGES to Variables :

- $q_2 \longleftarrow 10 \cdot q_1 + x$ (The new quotient)
- $r_2 \longleftarrow 100 \cdot r_1 + new - (20 \cdot q \cdot x + x^2)$ (The new remainder)
- $i \longleftarrow i + 2$

s_1	string of length n (substring of s_2)
s_2	string of length $n+2$
q_1	The 'quotient' of s_1
r_1	The 'remainder' of s_1
q_2	The 'quotient' of s_2
rem_{temp}	The remainder after appending last 2 digits
r_2	The 'remainder' of s_2
new	The last 2 digits i.e. $[s_2 - 100 \cdot s_1]$
x	A single digit number satisfying a certain condition

Table 1 This is a table indicating what certain variables mean.

§3 Proof of Correctness

As stated in the previous section, this code works recursively and for strings with even length.
So the proof follows inductively on the length of string whenever the length is even.

Proof By Induction

§3.1 Base Case

The base case is a string with 0 length. The value of a string with 0 length is 0 itself. As the counter is initially 0, which also happens to be the end of the string $\implies$ The algorithms terminates immediately returning the value 0 for both quotient and remainder.

Verifying The conditions :

1. $0^2 \leq 0 < 1^2$
2. $0^2 + 0 = 0$

§3.2 Induction Hypothesis

The proof proceeds by weak induction on the length of string.
To prove : $P(i) \implies P(i+2)$ [i+2 as the algorithm is only concerned with the correctness of strings with even length].
Where P(i) signifies that the algorithm is correct (returns quotient and remainder satisfying the description) for all strings with length i.

§3.3 Induction Step

We assume that the induction Predicate P(n) is true and try to prove predicate P(n+2) using the facts derived from P(n).

Lemma 3.1

$0 \leq remainder \leq 2 \cdot quotient$ [for the lengths of string for which the algorithm works]

Proof. According to the algorithm $quotient^2 \leq s < (quotient+1)^2$ and $quotient^2 + remainder = s$.
Now subtracting $quotient^2$ from all 3 terms of the first inequality gives $0 \leq s - quotient^2 < 2 \cdot q + 1$.
From the condition 2 , $s - quotient^2 = remainder$.
$\implies 0 \leq remainder < 2 \cdot quotient + 1 \iff 0 \leq remainder \leq 2 \cdot quotient.$ □

Let's say we are given that the string works for all strings with length n. (i.e. P(n) is True).
Now being correct means that for a string s of length n, the algorithm returns correct quotient and remainder satisfying the 2 conditions mentioned in Description.
Now to prove its correctness for a string with length n+2 , we know that it processes first the n character sub-string and then in one more step modifies the previous remainder and quotient to produce a result for string of length n+2.
i.e. The algorithm has running values after $i = n$ of variables quotient and remainder say q_1 and r_1 respectively which satisfy

$$q_1^2 \leq s_1 < (q_1+1)^2 \text{ and } q_1^2 + r_1 = s_1$$

Now the claim is that in the next step the values of remainder and quotient returned will be for
$100 \cdot s + new$.
Where new are basically the last 2 digits of the new string s_2 of length $n+2$.
According to the algorithm the r_1 is modified as follows :
The last 2 digits are appended to r_1 making its value as :

$$rem_{temp} = 100 \cdot r_1 + new$$

and then according to the algorithm a suitable x is found out which is the largest number that satisfies

$$(20 \cdot q_1 + x) \cdot x < rem_{temp}.$$

Lemma 3.2
There exists a single digit integer x which satisfies this condition.

Proof. The proof idea is by contradiction. Assuming that the maximum value of x such that the condition

$$(20 \cdot q_1 + x) \cdot x < rem_{temp}.$$

is satisfied is more than 10.
$\implies (20 \cdot q_1 + 10) \cdot 10 = 200 \cdot q_1 + 100 < rem_{temp} = 100 r_1 + new$
$200 \cdot q_1 + 100 < 100 \cdot r_1 + new$ ..(1)
Also $100 \cdot r1 + new \leq 200 \cdot q_1 + 99$ [as $r_1 \leq 2 \cdot q_1$ and $new \leq 99$] ..(2)
From (1) and (2) we arrive at a contradiction $\implies$ our assumption was wrong $\implies x < 10$ □

Thus x is a single digit number and is then appended to q_1 , hence the new quotient becomes

$$q_2 = 10 \cdot q_1 + x$$

Finally the new remainder after subtraction process becomes

$$rem_2 = rem_{temp} - (20 \cdot q_1 + x) \cdot x$$

Now we have to verify that the new quotient and remainder match the description of the algorithm.

Verifying Point 2 of the Description :
$q_2^2 + r_2 = s_2$?

$$(10 \cdot q_1 + x)^2 + (100 \cdot r_1 + new - (20 \cdot q_1 + x) \cdot x)$$
$$\implies 100 \cdot q_1^2 + 100 \cdot r_1 + new \implies 100 \cdot s_1 + new \implies s_2$$

Verifying Point 1 of the Description :

(a) Prove that $(q_2^2 \leq s_2)$

(b) Prove that $((q_2 + 1)^2 > s_2)$

Theorem 3.3

Verifying Point (a) :

> **Claim 3.4 —** That $q_1^2 \leq s_1$

> **Claim 3.5 —** That x is the largest value such that $(20 \cdot q_1 + x) \cdot x \leq 100 \cdot rem + new$

Proof. Multiplying inequality (1) by 100 and then adding the claim inequalities of the claims (2) , we get that $100 \cdot q_1^2 + 20 \cdot q_1 \cdot x + x^2 \leq 100 \cdot s_1 + new$.
Hence $(q_2^2 \leq s_2)$.

□

Theorem 3.6

Verifying Point (b) :

> **Claim 3.7 —** $100 \cdot r_1 + new < (20 \cdot q_1 + x + 1) \cdot (x + 1)$

> **Claim 3.8 —** $r_2 = 100 \cdot r_1 + new - (20 \cdot q \cdot x + x^2)$

> **Claim 3.9 —** $r_2 + q_2^2 = s_2$ [using Verification of Point 1]

Proof. Replacing $100 \cdot r_1 + new$ in claim 1 with $r_2 + (20 \cdot q \cdot x + x^2)$ using claim 2 and 3. We get $s_2 < q_2^2 + 2 \cdot (10 \cdot q_1 + s) + 1$ and because$(10 \cdot q_1 + x)$ is q_2 we get

$$s_2 < (q_2 + 1)^2$$

□

From verification of claim 1 and claim 2 we can conclude that $P(n) \implies P(n + 2)$

§3.4 Time Complexity

The time complexity of the code is $\mathbf{O}(n^2)$ where /textbfn is the length of the input string s. The helper functions used are string multiplication , string addition, string subtraction.
(all of which run in O(length of string)).
Note that the length of quotient and remainder are also $O(n)$ infact ¡= n/2 if the input is of length of n.
Some functions like remove leading 0s and others are also $O(n)$ in functioning.
The main function runs $O(n)$ times and each helper in it is used constant number of times, hence the total time complexity of the code is $O(n^2)$.
Also it can be shown that this is the best time complexity possible due to the fact that we have to atleast once process all the intermediate quotients and remainders, hence making the time complexity atleast $O(n^2)$.