Radical Algorithmics

Theodore S Norvell
Electrical and Computer Engineering
Memorial University

Written around 2005. Revised December 2010. Typeset December 26, 2010.

The natural root of a natural number x is the largest natural number not larger than the square root of x, that is $\lfloor\sqrt{x}\rfloor$. The problem is: given a natural number x, find $\lfloor\sqrt{x}\rfloor$.

1 Pencil and paper method

Here is a pencil and paper algorithm taught to me by Eric Gill who learned it from his father. I'll use x_0 to represent the original value of x. (Although you don't see it in this section, in the next section I will treat x as a program variable, so its value will change.)

1. Start with the most significant digit of x_0, if x_0 has an odd number of digits, or the two most significant digits of x_0, if x_0 has an even number of digits.

2. Find the natural root of this one or two digit number. Call it a. Here is an example. x_0 is 12345678. The first two digits are 12 and the natural root of that is 3.

$$\begin{array}{r l}
a \rightarrow & \ \ \mathbf{3} \\
x_0 \rightarrow & \big| \overline{\ 12\,34\,56\,78\ }
\end{array}$$

3. Take the square of a and subtract it from the one or two digit number

giving z

$$\begin{array}{r l}
a \rightarrow & \;\;3 \\
x_0 \rightarrow & \overline{\big|\;12\,34\,56\,78} \\
 & \;\;\underline{\mathbf{9}} \\
z \rightarrow & \;\;\mathbf{3}
\end{array}$$

4. If there are no digits of x_0 remaining to be considered, then stop. The root is a.

5. Take the next two digits from x_0 and append them to z to make a number y

$$\begin{array}{r l}
a \rightarrow & 3 \\
x_0 \rightarrow & \overline{\big|\;12\,34\,56\,78} \\
 & \;\;\underline{9}\downarrow\downarrow \\
y \rightarrow & \;\;\;3\,\mathbf{34}
\end{array}$$

6. Add a to itself to get $2a$ and then append a single digit b to that to make $20a + b$. Call this number w. But how do you know what b is? Pick b as the largest integer b such that bw (i.e. $20ab + b^2$) does not exceed y. (It happens that b can not be greater than 9). Usually you can just divide y by $20a$ to get b, since $20a$ is close to w. In the example, 334 divided by 60 (a times 20) is 5, and we then check that $5 \times 65 = 325$ is not greater than 334. It is not, so $b = 5$ and $w = 20 \times 3 + 5 = 65$. Tack the digit b onto the end of a and write bw under y.

$$\begin{array}{r l}
a \rightarrow & \;\;3\;\;\mathbf{5} \\
x_0 \rightarrow & \overline{\big|\;12\,34\,56\,78} \\
 & \;\;\underline{9} \\
y \rightarrow & \;\;3\,34 \\
bw = 5 \times (20 \times 3 + 5) \rightarrow & \;\;\mathbf{3\,25}
\end{array}$$

7. Subtract bw from y to get a new value for z.

$$\begin{array}{r l}
a \rightarrow & \;\;3\;\;5 \\
x_0 \rightarrow & \overline{\big|\;12\,34\,56\,78} \\
 & \;\;\underline{9} \\
y \rightarrow & \;\;3\,34 \\
bw \rightarrow & \;\;\underline{3\,25} \\
z \rightarrow & \;\;\;\;\;\mathbf{9}
\end{array}$$

8. Go back to step 4.

The example continues with two more iterations. The second iteration finds the next digit to be 1.

$$\begin{array}{r l}
a \rightarrow & \quad 3\ \ 5\ \ \mathbf{1} \\
x_0 \rightarrow & \overline{\big|\, 12\,34\,56\,78} \\
 & \ \underline{\ 9}\ \ \quad \downarrow\downarrow \\
 & \quad 3\,34 \\
 & \quad \underline{3\,25} \\
 & \qquad 9\,\mathbf{56} \\
1 \times (20 \times 35 + 1) \rightarrow & \qquad \underline{\mathbf{7\,01}} \\
z \rightarrow & \qquad \mathbf{2\,55}
\end{array}$$

The final iteration finds the final digit to be 3.

$$\begin{array}{r l}
a \rightarrow & \quad 3\ \ 5\ \ 1\ \ \mathbf{3} \\
x_0 \rightarrow & \overline{\big|\, 12\,34\,56\,78} \\
 & \ \underline{\ 9}\ \ \qquad \downarrow\downarrow \\
 & \quad 3\,34 \\
 & \quad \underline{3\,25} \\
 & \qquad 9\,56 \\
 & \qquad \underline{7\,01} \\
 & \qquad 2\,55\,\mathbf{78} \\
3 \times (20 \times 351 + 3) \rightarrow & \qquad \underline{\mathbf{2\,10\,69}} \\
z \rightarrow & \qquad\ \ \mathbf{45\,09}
\end{array}$$

You can continue this process to obtain more decimal places. I'm going to ignore the possibility and just stop once the integer part is obtained.

2 Deriving the algorithm

2.1 Two observations

The first observation is that you can find all the digits of the answer except the last while ignoring the last two digits of the input.

One way to see this is to consider the function $\lfloor \sqrt{t} \rfloor$ as a function of nonnegative real numbers. As t increases, the value of the function only increases when t is a perfect square and hence an integer. This means that is the same as $\left\lfloor \sqrt{\lfloor t \rfloor} \right\rfloor$, as otherwise the function would have to take a step somewhere after $\lfloor t \rfloor$ but before or at t, and none of these values is an integer.

Now consider any natural x. Using the result of the last paragraph we have

$$\lfloor \sqrt{x}/10 \rfloor = \left\lfloor \sqrt{x/100} \right\rfloor = \left\lfloor \sqrt{\lfloor x/100 \rfloor} \right\rfloor = \left\lfloor \sqrt{x \operatorname{div} 100} \right\rfloor$$

Let $\sqrt{x} = 10a + \beta$, where a is an integer and β is a real number, $0 \leq \beta < 10$. We have $\lfloor \sqrt{x}/10 \rfloor = \lfloor a + \frac{\beta}{10} \rfloor = a$. All together we have $a = \lfloor \sqrt{x}/10 \rfloor = \left\lfloor \sqrt{x \operatorname{div} 100} \right\rfloor$. Thus is, if we can find $a = \left\lfloor \sqrt{x \operatorname{div} 100} \right\rfloor$, we can rest assured that $\sqrt{x}$ will be of the form $\sqrt{x} = 10a + \beta$, where β is a real number, $0 \leq \beta < 10$, and thus that $\lfloor \sqrt{x} \rfloor = 10a + b$ where $b = \lfloor \beta \rfloor$ is an integer $0 \leq b < 10$. So, having computed $\left\lfloor \sqrt{x \operatorname{div} 100} \right\rfloor$, you have all but the last digit of $\lfloor \sqrt{x} \rfloor$.

The second observation is that once you know all but the last digit, the last digit is easy to find. Suppose we know $\lfloor \sqrt{x} \rfloor$ is of the form $10a + b$, with a and b naturals and $0 \leq b < 10$. First consider the case where x is a square; we have

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

Let $y = x - 100a^2$ then we have $y = b(20a + b)$. If we know a, it remains to find that b such that $y = b(20a + b)$.

Now if x is not known to be a perfect square, we can't be sure that a b such that $y = b(20a + b)$ will exist. If we instead find the largest integer b such that $y \geq b(20a + b)$, then we can show the following two facts: First

$$\begin{aligned} & (10a + b)^2 \\ = \; & 100a + 20ab + b^2 \\ = \; & 100a + b(20a + b) \\ \leq \; & 100a + y \\ = \; & x \end{aligned}$$

Taking the root of both sides we have $(10a + b) \leq \sqrt{x}$. Second, since b is the *largest* integer such that $y \geq b(20a + b)$, we know $(b + 1)(20a + (b + 1)) > y$, and so

$$\begin{array}{rl} & (10a+b+1)^2 \\ = & 100a + 20a(b+1) + (b+1)^2 \\ = & 100a + (b+1)(20a+(b+1)) \\ > & 100a + y \\ = & x \end{array}$$

Taking the root of both sides we have $10a+b+1 > \sqrt{x}$. Putting the two facts together gives

$$10a+b \le \sqrt{x} < 10a+b+1$$

and thus

$$10a+b = \lfloor \sqrt{x} \rfloor$$

Everything in this section works for any base greater than 2. For base 2, replace 20 with $10_2 + 10_2$, i.e., 100_2.

2.2 Functional algorithm

The above two observations lead to the following functional algorithm

```
fun nroot( $x$ : int )
    pre $x \ge 0$
    post $result = \lfloor\sqrt{x}\rfloor$
    if $x = 0$
        return 0
    else
        let $a = \text{nroot}(x \operatorname{div} 100)$
        let $y = x - 100a^2$
        let $b$ be the largest int such that $b(20a+b) \le y$
        assert $0 \le b < 10$
        return $10a + b$
```

2.3 Recursive imperative algorithms

As a first step toward an iterative algorithm we replace the recursive function above with a recursive procedure that expects its input in a parameter x and leaves its output in a variable a

```
var $a$
proc nrootImp( $x$ : int )
    pre $x \geq 0$
    post $a' = \lfloor \sqrt{x} \rfloor$
    if $x = 0$
        $a := 0$
    else
        nrootImp($x \operatorname{div} 100$)
        % Calculate the next digit of $a$
            let $y = x - 100a^2$
            let $b$ be the largest int such that $b(20a + b) \leq y$
            assert $0 \leq b < 10$
            $a := 10a + b$
nrootImp( $x$ )
```

The next step is to eliminate all local variables; in this case x is local to the procedure. To do that we put $x := x \operatorname{div} 100$ in front of the recursive call. However, after the call we need to undo this assignment and so we have to remember all the digits that have been taken off. To do this we use a variables p and i. The variable p holds all the digits that have been removed from x but that haven't yet been put back, we use i to count how many pairs of digits have been moved from x to p. For example, if the orginal value of x is 1234567890, then after 3 digit pairs have been transfered, we will have

$$i = 3 \qquad p = 567890 \qquad x = 1234$$

The condition that links x, p, and i is $x_0 = x \times 100^i + p$ where x_0 is the original value of x.

```
var $a$
var $p$
var $i$
let $x_0 = x$
proc nrootImp
    pre $x \geq 0 \wedge x_0 = x \times 100^i + p \wedge 0 \leq p < 100^i$
    post $a' = \lfloor \sqrt{x} \rfloor \wedge x_0 = x' \times 100^{i'} + p' \wedge 0 \leq p' < 100^{i'}$
    if $x = 0$
        $a := 0$
    else
        % Shift two digits from the right end of $x$ to the left end of $p$
            $p := p + (x \bmod 100) \times 100^i$
            $x := x \operatorname{div} 100$
            $i := i + 1$
        nrootImp
        % Shift two digits from the left end of $p$ to the right end of $x$
            $i := i - 1$
            $x := 100x + p \operatorname{div} 100^i$
            $p := p \bmod 100^i$
        % Calculate the next digit of $a$
            let $y = x - 100a^2$
            let $b$ be the largest int such that $b(20a + b) \leq y$
            assert $0 \leq b < 10$
            $a := 10a + b$
$i := 0$
$p := 0$
nrootImp
```

2.4 Iterative algorithm

The recursion is linear and of the form

```
proc $s$
    pre $P$
    post $Q$
    if $E$
        $C$
    else
        $B$
        $s$
        $D$
$A$
$s$
```

Such a recursion can be transformed into a pair of loops

```
$A$
while $\neg E$
    inv $P$
    $B$
$C$
while ...
    inv $\widetilde{Q}$
    $D$
```

where the second loop iterates the same number of times as the first. The invariant $\widetilde{Q}$ of the second loop is the same as Q but with all primes erased.

Applying this transformation to the most recent version gives:

```
pre $x \geq 0$
post $a' = \lfloor \sqrt{x} \rfloor$
let $x_0 = x$
var $p := 0$
var $i := 0$
while $x \neq 0$
    inv $0 \leq p < 100^i$
    inv $x_0 = x \times 100^i + p$
    % Shift two digits from the right end of $x$ to the left end of $p$
        $p := p + (x \bmod 100) \times 100^i$
        $x := x \operatorname{div} 100$
        $i := i + 1$
var $a := 0$
while $i \neq 0$
    inv $a = \lfloor \sqrt{x} \rfloor$
    inv $0 \leq p < 100^i$
    inv $x_0 = x \times 100^i + p$
    % Shift two digits from the left of $p$ to the right end of $x$
        $i := i - 1$
        $x := 100x + p \operatorname{div} 100^i$
        $p := p \bmod 100^i$
    % Calculate the next digit of $a$
        let $y = x - 100a^2$
        let $b$ be the largest int such that $b(20a + b) \leq y$
        assert $0 \leq b < 10$
        $a := 10a + b$
```

The invariants of the second loop are $a = \lfloor\sqrt{x}\rfloor$, $0 \leq p < 100^i$ and $x_0 = x \times 100^i + p$, where x_0 is the original value of x.When the loop terminates we have, $i = 0$ and so $p = 0$, and thus $x = x_0$, and so $a = \lfloor\sqrt{x_0}\rfloor$.

As an optimization, we can note that the first loop's effect on p is to set it to the initial value of x; we can remove the assignment to p in the loop and instead initialize p to x. Of course, this destroys the loop invariant of the first loop, but it still ensures that after the first loop we have

$$0 \leq p < 100^i \wedge x_0 = x \times 100^i + p$$

2.5 A strength reduction

It is inefficient to recalculate a^2 each time through the second loop. One way to avoid recalculating a^2 each time through the loop is to add a tracking variable z with an invariant of $z = x - a^2$. Using a tracking variable to track a product is similar to the classic compiler optimization of strength reduction.

This gives a formal version of the pencil and paper algorithm Eric told me about:

```
pre $x \geq 0$
post $a' = \lfloor\sqrt{x}\rfloor$
let $x_0 = x$
var $p := x$
var $i := 0$
while $x \neq 0$
    $x := x \operatorname{div} 100$
    $i := i + 1$
var $a := 0$
var $z := 0$
while $i \neq 0$
    inv $a = \lfloor\sqrt{x}\rfloor$
    inv $0 \leq p < 100^i$
    inv $x_0 = x \times 100^i + p$
    inv $z = x - a^2$
```

```
% Shift two digits $r$ from the left of $p$ to the right of $x$
    $i := i - 1$
    let $r = p \operatorname{div} 100^i$
    $x := 100x + r$
    $p := p \bmod 100^i$
% Calculate the next digit of $a$
    let $y = 100z + r$
    assert $y = x - 100a^2$
    let $b$ be the largest int such that $b(20a + b) \leq y$
    assert $0 \leq b < 10$
    $z := y - b(20a + b)$
    $a := 10a + b$
```

The new assignment to y is justified as follows. Let x_1 be the value of x at the start of the loop body. Thus we have (at the point of the declaration of y) $z = x_1 - a^2$ and hence $100a^2 = 100x_1 - 100z$. At the same point we have $x = 100x_1 + r$. Subtracting we get $x - 100a^2 = 100z + r$; the RHS is the value assigned to y and the LHS is what we require it to be.

The assignment to z in the loop is justified as follows. Let a_0 be the value of a prior to the assignment to a in the loop. Then, at the end of the loop body, we have $x = y + 100a_0^2$ and $a^2 = 100a_0^2 + b(20a_0 + b)$.So $x - a^2 = y - b(20a_0 + b)$.Thus, after the assignment to z, $z = x - a^2$, as desired.

The assignment to x in the second loop is no longer needed to calculate the result, but I've left it in because x is still used in the reasoning.

2.6 Hardware algorithm

We can apply the same algorithm to any base. An obvious choice for hardware implementation is base 2, but base 4 or 8 might lead to quicker hardware, at the price of a few more gates. The following version uses base 2.

In it, I've replaced p with a variable q which contains the same number as p but written backwards in base 4; (if $p = 1230_4$ then $q = 321_4$); in binary, the order of pairs of bits is reversed so that the most significant bit is 1, then 0, then 3, then 2, etc. Operations that previously involved the left end of p, now involve the right end of q. This representation avoids the use of powers, which, in hardware terms, means that there is no need for variable shifts: all shifts are now by one or two bits.

```
pre $x \geq 0$
post $a' = \lfloor\sqrt{x}\rfloor$
var $q := 0$
var $i := 0$
while $x \neq 0$
    % Shift two bits from $x$ to $q$.
        $q := 4q + x \bmod 4$;
        $x := x \operatorname{div} 4$
        $i := i + 1$
var $a := 0$
var $z := 0$
while $i \neq 0$
    % Shift two bits $r$ from $q$.
        $i := i - 1$
        let $r = q \bmod 4$
        $q := q \operatorname{div} 4$
    % Calculate the next bit of $a$
        let $y = 4z + r$
        let $b$ be 1 if $(4a + 1) \leq y$ and 0 otherwise
        $z := y - b(4a + b)$
        $a := 2a + b$
```

Note that all operations are easily implemented with digital hardware. The most complex components needed are adders, subtractors, and a comparator.

This algorithm is similar to one that I derived a few years ago based on an abstract binary search. However that algorithm did not take advantage of the first observation and thus uses arithmetic with more bits than needed here, especially in the early iterations. Calculating with fewer bits or digits is an advantage when you are using pencil and paper; it is also an advantage in pipelined hardware, as it leads to a saving of gates. Furthermore, as that algorithm started with binary search, it lead directly to the binary version of the algorithm and did not make the generalization to other bases quite as obvious.