An Analysis on Extracting Square and Cube Roots by Aryabhata's Methods.

Yumnam Kirani Singh, C-DAC, IIPC Building, NIT Campus, Silchar, Assam.
Email:yumnam.singh[@]cdac.in

***Abstract*—** Finding accurate root of a number is still considered as challenging in computer science community. The only popularly known method to compute root is the long division method of finding square root. Aryabhata contributed two methods similar to long division method to compute square root and cube root. In recent years, his methods have also been studied, explained and tried to implement as computer algorithms. The explained methods and proposed algorithms fail to give correct results. Some analyses have been made on these methods in order to ascertain why the algorithms fail. Improved algorithms have been provided to give correct result while computing square root or cube root using Aryabhata's methods. .

***Keywords*—cube root, square root, Bino's Model of multiplication, Large number manipulation, Long division method, Aryabhata's methods.**

(Article history: Received 12 November 2016 and accepted 30 December 2016)

I. INTRODUCTION

Aryabhatta is one of the most revered ancient Indian mathematicians who made significant contributions in various fields of mathematics and astronomy [1, 2]. His contribution in mathematics has also been studied for possible applications in cryptography [3]. He also contributed methods of finding square root and cube root of a number in his book Ganitpada. The methods are translated into English and explained with examples [1,2]. The methods are similar in approach to the long division method of extracting square root but not exactly the same. It seems to be simpler as compared to long division method especially the way how the next digit of the root is computed from the knowledge of previously computed digits of the root. However, the ways methods have been explained or algorithms have been designed have limitations in the sense that the method gives incorrect results. The explained methods fail to give the correct values of the square root or cube root of some of the numbers such as square root of 841 or cube root of 17576. The reason might be missing of some points in the translation of Aryabhata's methods. In [1], the author tried to explain the methods in binomial expansion of power 2 and 3 i.e., $(a+b)^2$ and $(a+b)^3$ in the same way some mathematicians try to explain long division method of computing square roots in terms of polynomial expansion of power of 2. But a multi-digit number is very different from a polynomial; a binomial and a 2-digit number are different entities. Elements in a polynomial do not have place value whereas digits in numbers have place values. For example, $(1+6)^2$ and $(6+1)^2$ will give 49, where as $(16)^2$ gives 256 and $(61)^2$ gives 3721. More correct ways of explaining long division of extracting square root is given in [5] and the extension of long division method for computing cube root is given in [4].

In this paper, the algorithms for extraction of square root and cube root using Aryabhata's methods are analyzed and improved so that it gives for correct results while computing the roots. The paper is divided into four sections. Section II discusses algorithm for extraction of square root and section III discusses the algorithm to compute cube root. A short conclusion is given in section IV.

II. SQUARE ROOT EXTRACTION ALGORITHM

In this section, the algorithm or method of extracting square root by using Aryabhata's method as explained in [1,2] fails to work for some numbers whose square root has 1 as the first digit. The square root algorithm given in [1] is reproduced here for discussion.

Algorithm:
Represent the given number as a series of indexed digits, i.e., $d_n d_{n-1} \cdots \cdots d_2 d_1 d_0$, where $d_0$is the digit in unit's place, $d_1$is the digit in ten's place, $d_2$is the digit in hundred's place and so on. Let R be the final root and n be the index of the leftmost digit in the number.

1. Pick d_i such that $\frac{i}{2}$ = integer,
   Where $\frac{i+k}{2} \neq integer$ and $\frac{i}{2} > \frac{i-k}{2}$, $k = 1,2,3, \cdots$
2. $p = \left\lfloor \frac{n-1}{2} \right\rfloor$
3. If d_{i+1} exists $a = 10 \times d_{i+1} + d_i$ else $a = d_i$
4. Choose A such that $A^2 \leq a$ and $a - A^2$is minimum
5. $S = a - A^2$
6. $R = A$
7. $y = 10 \times S + d_{i-1}$
8. $S = y \ mod \ (2 \times R)$
9. $B = \left\lfloor \frac{y}{2 \times R} \right\rfloor$

10. $R = 10 \times R + B$
11. $c = 10 \times S + d_{i-2}$
12. $S = c - B^2$
13. $i = i - 2$
14. $p = p - 1$
15. If $p \neq 0$ go to 7, else quit.

Step-1 is to segment the given number into segments of two digits starting from the right. Step-2 is to determine how many digits would be computed after the first square root of the first segment is computed. Step-3 is to select the leftmost segment, which may consist of a single digit or 2-digits.Step-4 is to compute the first digit of the square root, i.e., the square root of the first leftmost segment. Step -5 is to find the difference between the leftmost segment and the square of the first digit of the root. Step-6 is to store the first digit of the root in the result. Step 7 to 14 is to compute the next digits of the square roots from next segments of the numbers under control of Step-15.

Step-7 combines the difference from step-5 and first digit of the next segment. Step-8 finds the remainder when the number in step-7 is divided by twice the result obtained so far. Step-9 computes the next digit of the square root, which is the quotient when the number in step-7 is divided by twice the result of the root obtained so far. Step-10 stores the quotient obtained in 9 as the next digit of the root. Step-11 combines remainder obtained in step-8 with the second digit of the next segment being processed. Step-12 finds the difference between the number obtained in step-11 and the square of the digit of the root computed in step-9. Step-13, index for choosing the segments of two digits from the number is updated by decrementing its value by 2. Similarly, the number of digits of roots required to be computed is updated by decrementing its value by 1. Step-5 checks whether all the required number of digits for the square root are computed. If not, computation next digit of the root is repeated.

Let us analyze the reason why the above algorithm fails to work when the first digit of the root is 1 and the second digit is greater than 4.

Example: Square root of 256.

By the above algorithm, the square root is given as 17. Let us analyze, why this happens.

Step-1: The given number is divided into two segments as 2, 56. First segment is 2, and second segment is 56 and we have to pick the last digit in the first segment, so i=2.

Step-2: Number of digits in the square root after first digit.

There is minor correction in the evaluation of number digits to be computed after the square root of the first segment of the given number.

$p = \lfloor n/2 \rfloor = \lfloor 2/2 \rfloor = 1$

Step-3: Getting the first segment, a=2

Step-4: Integer part of square root of first segment, $A = \lfloor \sqrt{2} \rfloor = 1$, which is first digit of square root of the number.

Step-5: Finding difference between the first segment and square of the first digit of the root.

$S = a - A^2 = 2 - 1 = 1$

Step-6: Putting the first digit of the square root in the result.

R=1

Step-7: Combining the first digit of the next segment with the difference in step-5.

$y = 10 \times S + d_{i-1} = 10 + 5 = 15$

Step-8: Finding the remainder when the number in step-7 is divided by twice the partial root obtained so far.

$S = y \ mod \ (2 \times R) = 15 \ mod \ 2 = 1$

Step-9: Computing next digit of the square root.

$B = \left\lfloor \frac{y}{2 \times R} \right\rfloor = \left\lfloor \frac{15}{2} \right\rfloor = 7$

Step-10: Updating the result of square root

$R = 10 \times R + B = 10 \times 1 + 7 = 17$

Step-11: combining the remainder in step-8 with second digit of the second segment.

$c = 10 \times S + d_{i-2} = 10 \times 1 + 6 = 16$

Step-12: Finding the difference between the number in step-11 and square of the digit of the square root computed last.

$S = c - B^2 = 16 - 49 = -33$

Step-13: Updating i index to process next two digits of the next segment if any.

$i = i - 2 = 0$

Step-14: Updating the number of digits of the square root still left for computation

$p = p - 1 = 0$

Step-15: Check whether to proceed for computation of next digit of the square root or not. Since p=0, the algorithm exits here.

So, at the end of execution of the algorithm, 17 is given as the square root of 256. Similarly, the algorithm gives square roots of numbers 225, 289, 324, 361 as 16, 19, 111, 113 instead of 15, 17, 18 and 19. The reason why it fails to work is that when the second digit is greater than 4, some carry was added in the first segment during the multiplication process. This was not considered when determining the second digit of the root in step-9. In other words, the relation given in step-9 to compute the next digit of the root does not guarantee that it is will always be a digit and the computed digit will be the correct digit. So, in order to make the algorithm work for all numbers, we need to check whether the computed value from the step-9 could be a correct digit for the square root. This can be done by checking whether the value in step-12 is negative or not. If the value is negative, the value of the root computed in step-9 should be continuously reduced by 1, until the value in step-12 becomes greater than or equal to zero. The improved algorithm to compute the square root by Aryabhata's method is given below.

Improved Algorithm: (Square root extraction by Aryabhatta's method)

Input: A positive integer within byte size limitation

Output: A positive integer representing integer part of cube root of the given number.

1. $p = \left\lfloor \frac{n}{2} \right\rfloor$
2. Pick d_i such that $i = 2 * p$
3. If d_{i+1} exists $a = 10 \times d_{i+1} + d_i$ else $a = d_i$
4. Choose A such that $A^2 \leq a$ and $a - A^2$ is minimum
5. $S = a - A^2$

6. $R = A$
7. While $p > 0$ ---- continue step 8 to 16
8. $y = 10 \times S + d_{i-1}$
9. $S = y \ mod \ (2 \times R)$
10. $c = 10 \times S + d_{i-2}$
11. $B = \left\lfloor \frac{y}{2 \times R} \right\rfloor$
12. $S = c - B^2$
13. If $S < 0, B = B - 1$, go to Step-11.
14. $R = 10 \times R + B$
15. $i = i - 2$
16. $p = p - 1$
    End While

In the improved algorithm, the process of grouping of digits of the given number in two's starting from the right end is simplified. Also, the computation of next digit of the root is performed only when required in while loop. Suppose, we want to compute square root of 81, we do not need to proceed for computation for next digit. More importantly, the estimated value of the root in step-11 is greater than the expected value of the next digit of the square root. If the value is greater than the expected value, the estimated value is kept on reducing by 1. With these modifications, the algorithm will give correct result of integer part when finding square root using Aryabhata's method.

III. COMPUTING CUBE ROOT

The algorithm given in [1] for extraction of cube root of a positive number fails to works for numbers whose cube root has 1 as first digit and a digit greater than 3 as the second digit. The algorithm is reproduced here also for further discussion.

Algorithm:

Represent the given number as a series of indexed digits, i.e.,$d_n d_{n-1} \cdots \cdots d_2 d_1 d_0$, where d_0 is the digit in units place, $d_1$is the digit in tens place, d_2 is the digit in hundreds place and so on. Let the final root be R and$n$ is the index of the left most digit in the given number. Then the algorithm is as follows:

1. Pick the digit $d_i$such that $\frac{i}{3}$= integer, $\frac{i+k}{3} \neq$ integer and $\frac{i}{3} > \frac{i-k}{3}$, k =1,2,3, … .
2. $p = \left\lfloor \frac{n-1}{3} \right\rfloor$
3. Let $= 100 \times d_{i+2} + 10 \times d_{i+1} + d_i$. if does d_{i+2} not exist, let d_{i+2}=0; if d_{i+1}does not exist, let $d_{i+1} = 0$
4. Choose A such that $A^3 \leq k$ and $k - A^3$ is minimum.
5. $S = k - A^3$
6. $R = A$
7. $l = 10 \times S + d_{i-1}$
8. $S = l \ mod \ (3 \times R^2)$
9. $m = 10 \times S + d_{i-2}$
10. $B = \left\lfloor \frac{l}{3 \times R^2} \right\rfloor$
11. $S = m - 3 \times R \times B^2$
12. $R = 10 \times R + B$
13. $n = 10 \times S + d_{i-3}$
14. $S = n - B^3$
15. $i = i - 3$
16. $p = p - 1$
17. If $p \neq 0$ go to 7, else quit.

Step-1 is to group the given number in terms of three digits starting from the right side. Step-2 gives the number of digits in the cube root to be computed after the first digit of the cube root. Step-3 gets the leftmost segment or group of the number. Step-4 computes the first digit of the cube root. Step-5 finds the difference between the first segment and the cube of the first digit of the cube root. The first digit of the cube root is put to the result in step-6. Step-7 combines the remainder in step-5 and the first digit of the next group or segment. Step-8 finds the remainder of when the number in step-7 is divided by thrice of the square of the result obtained so far. Step-9 combines the remainder with next digit (i.e., second digit) of the segment being processed. Step-10, computes the next digit of the cube root, which is the quotient obtained when the number in step-7 is divided by thrice of square of the result obtained so far. Step-11 finds the difference between the result obtained in step-9 and multiplication of thrice the result and square the digit of the root computed in step-10. In step-12, the result is updated by incorporating the digit of the root computed in Step-10. Step-13 combines the difference obtained in step-11 with the third digit of the segment. In step-14, the difference between the number in step-13 and the cube of the digit of the root is found. Step-15 updates the index to get the next three digit of the next segment by reducing its value by 3. Step-16, updates the counter which indicates how many digits of the root are still left with for computation. Step-17 checks whether any more digit is required to be computed. If so steps the same process to compute next digit of the root is repeated from step-7 onwards.

The above algorithm fails to work to find the cube roots of some of the numbers whose first digit in cube roots is 1 and next digit is greater than 3. The reasons are the same as explained in the case of failures of extraction of square root. That is, the algorithm did not consider the cases of carries that has been transferred from right to left digits when cubing two or more digit number.

Example: Cube root of 2744.

By the above algorithm, the cube root is given as 15. Let us analyze, why this happens.

Step-1: The number is divided in two segments: 2,744. The first segment is 2 and the second segment is 744. Also, i=3.

Step-2:$p = \lfloor n/3 \rfloor$=Flooring of (3/3)=1,

That is, one more digit of the root is required to be computed after the first digit of the root is computed from the first segment.

Step-3: Getting the first segment, k=2.

Step-4: Integer part of cube root of first segment, A=1.

Step-5:$S = K - A^3 = 2 - 1^3 = 1$

Step-6: Result R=1

Step-7: combining the difference in 5 with first digit of next group, i.e., 744

$l = 10 \times S + d_{i-1} = 10 \times 1 + 7 = 17$

Step-8: $S = l \; mod \; (3 \times R^2) = 17 \; mod \; 3 = 2$

Step-9: combining the remainder in 8 with the second digit of the segment being processed, i.e., second segment i.e., 744.

$m = 10 \times S + d_{i-2} = 10 \times 2 + 4 = 24$

Step-10: computing next digit of the cube root.

$$B = \left\lfloor \frac{l}{3 \times R^2} \right\rfloor = \left\lfloor \frac{17}{3} \right\rfloor = 5$$

Step-11:

$S = m - 3 \times R \times B^2 = 24 - 3 \times 1 \times 5^2 = 24 - 75 = -51$

Step-12: Updating cube root with next digit computed in step-10

$R = 10 \times R + B = 10 \times 1 + 5 = 15$

Step-13: Combining the remainder in step-11 with next digit of the root.

$$n = 10 \times S + d_{i-3} = 10 \times -51 + 4 = -506$$

Step-14: Combining the remainder in step-13 with the third digit of the group being processed, i.e., 3rd digit in 744.

$S = n - B^3 = -506 - 5^3 = -506 - 125 = -631$

Step-15: updating i for processing next three digits of the next group if any.

$i = i - 3 = 3 - 3 = 0;$

Step-16: Updating p to determine, whether more digits of cube root need to computed.

$p = p - 1 = 1 - 1 = 0$

Step-17: Since $p = 0$, exit.

So, the result of the computation of the cube root by the above algorithm is 15 instead of 14. The reason why this happens is that the digit of the cube root by step-10 does not guarantee that the digit would be the correct digit of the cube root. Also, in the process that follows does not check bout the correctness of the computed digit of the cube root in step-10.

Improved Algorithm: (Aryabhatta's method to compute cube root)

Input: Positive integer within byte size limitation, $d_n d_{n-1} \cdots \cdots d_2 d_1 d_0$

Output: integer part of the cube root of the number $d_n d_{n-1} \cdots \cdots d_2 d_1 d_0$

Steps:

1. $p = \left\lfloor \frac{n}{3} \right\rfloor$
2. Pick d_i where $i = 3 * p$
3. Let $= 100 \times d_{i+2} + 10 \times d_{i+1} + d_i$. if does d_{i+2} not exist, let d_{i+2}=0; if d_{i+1}does not exist, let $d_{i+1} = 0$
4. Choose A such that $A^3 \leq k$ and $k - A^3$ is minimum.
5. $S = k - A^3$
6. $R = A$
7. While p>0 ----continue step 8 to 18
8. $l = 10 \times S + d_{i-1}$
9. $S = l \; mod \; (3 \times R^2)$
10. $m = 10 \times S + d_{i-2}$
11. $B = \left\lfloor \frac{l}{3 \times R^2} \right\rfloor$
12. $S = m - 3 \times R \times B^2$
13. $n = 10 \times S + d_{i-3}$
14. $S = n - B^3$
15. If $S < 0, B = B - 1$ go to step-11
16. $R = 10 \times R + B$
17. $i = i - 3$
18. $p = p - 1$

End while

In the above algorithm, the way in which digits of a given number is grouped into three's starting from the right has been simplified. The estimated value of next digit of cube root is tested for correction before adding to the result.

Also, note that the improved algorithms given in Section II and III can be used for computation of square root or cube root of a number within the maximum limit an integer variable can have in a programming language. To make the algorithms works for any arbitrarily large number, the algorithm must be modified in representa arithmetic as in [4,5, 6].

IV. CONCLUSIONS

The algorithms in [1] for Aryabhata's methods of extraction of square root and cube root have been analyzed and traced the shortcomings why the algorithms fail to give correct results for some numbers. Improved algorithms have been provided to give correct results while computing square or cube root for any positive integer that can be processed in a computer. The algorithms can be extended to find the square root or cube root of any arbitrarily large integers by implementing it in representa arithmetic.

REFERENCES

[1] Abhishek Parakh, "Aryabhata's root extraction methods", Indian Journal of History of Science, Vol. 42, No.2, pp. 149-161, 2007

[2] David H. Bailey, Jonathan M Borwein, "Ancient Indian Square roots: An Exercise in Forensic Paleo Mathematics" https://www.carma.newcastle.edu.au/jon/india-sqrt.pdf (square root and cube root, aryabahatta)

[3] T.R.N Rao and C.-H Yang, "Aryabhata remainder theorem: relevance to crypto algorithms", Circuits, Systems and Signal Processing, vol-25, pp. 1-15, 2006.

[4] Y. Kirani Singh, "Computing cube root of a positive number" ADBU journal of Engineering Technology, Special issue: pp. 85-89, Vol. 4, March 2016.

[5] Y. Kirani Singh, "Computing square root of a large positive integer" ADBU Journal of Engineering Technology, Vol. 5, June-July, 2016.

[6] Yumnam Kirani Singh , "On Some Generalized Transforms for Signal Decomposition and Reconstruction " Ph.D. dissertation, CVPR Unit, ISI, Kolktata. 2006.