Turk. J. Math. Comput. Sci.
11(2)(2019) 74–77
© MatDer
http://dergipark.gov.tr/tjmcs
http://tjmcs.matder.org.tr

MATDER

An Efficient Algorithm to Find All Primes in A Given Interval

Farhat Mechkene

Independent researcher, 7, Rue Sinaa, Menzah 8. PO Box: 2037 Ariana, Tunisia.

Received: 15-04-2019 • Accepted: 06-06-2019

Abstract. In this paper, we propose a deterministic algorithm for primality testing and primes search in a given integer interval. The algorithm use a new primality test method, which replace modulo operator with elementary arithmetic operations, hence a better efficiency than divisibility test. The algorithm is working; it generates a prime base by an expansion process and is appropriate for a fast search for small primes (a dozen of digits). We propose a filtering method to overcome memory constraints, and use the algorithm to expand much more the prime base and find medium size primes (dozens of digits).

2010 AMS Classification: 11A51, 11A07.

Keywords: Finding all frimes in an interval, deterministic algorithms for primality testing.

1. Introduction

There are two major tasks related to prime numbers: first, one is to test whether a given number is prime or not and the second one is to find all prime numbers between two given numbers [2]. In our research work, we tackle the two problems.

We can prove that if an integer n is not a prime, n must have a factor greater than 1 and $\leq \sqrt{n}$. To test the primality of n, the optimal deterministic method is to check only whether primes from 2 to $\sqrt{n}$ are possible divisors for n; i.e. check that $n \ mod \ pi \neq 0$ for each prime pi with condition $pi2 < n$ [3].

We propose an original method to obtain $n \ mod \ pi$ with only elementary arithmetic operations. The method do not use modulo operator, hence an improved efficiency in comparison with the classical divisibility test. In the following, we designate the method, by the acronym MRM (Modulo Relations Method). The proposed algorithm use the MRM method.

A Prime Base (PB) is a complete suite of successive primes, starting with the first primes: $\{2, 3, 5, \ldots, pm\}$.

The algorithm perform the following: generate efficiently a PB by a fast expansion process. In a second stage, the algorithm search for primes in a given interval $\leq pm2$, where pm is the last prime in the generated PB. The MRM method and the algorithm are new.

The paper is organized as follows. Section 2, gives the mathematical preliminaries needed, and present the Modulo Relations Method. Section 3 deals with the primes search algorithm. Finally, some concluding remarks are given.

Email address: f_mechkene@hotmail.com (F. Mechkene)

2. Preliminaries. MRM Method

Given a number n odd, we compute $(n + 2)\ mod\ p$, based on $n\ mod\ p$, with the following modulo relation:
If $(n\ mod\ p + 2) < p$, then $(n + 2)\ mod\ p = n\ mod\ p + 2$.
If $(n\ mod\ p + 2) >= p$, then $(n + 2)\ mod\ p = n\ mod\ p + 2 - p$.
With this relation, having obtained $n\ mod\ p$, we perform primality check of n, and then we reuse $n\ mod\ p$ to get $(n + 2)\ mod\ p$, by simple addition. In other words, we store modulo of last odd number, and reuse it to perform check for next odd. We continue and repeat the same process, until end of search interval.
This Modulo Relations Method (MRM) has a double advantage:

* First, a simplification of modulo calculation (hence a better speed).
* Secondly, we obtain results that are more interesting: not only allow primality check of a single number, but also reuse modulo to obtain the set of all primes included in the search interval.

To measure time efficiency, instead of running time (vary with length of search interval), we propose the following ratio: Running time / number of primes found (or msec. per prime found).
Proof of modulo relation

- $n1$ and $n2$ are two positive integers with $n1 < n2$;
- $n1\ mod\ p$: denotes $n1\ modulo\ p$.
- Let: $n1\ mod\ p = m1\ (m1 < p) \rightarrow n1 \equiv m1\ (mod\ p)$.
- Let: $n2\ mod\ p = m2\ (m2 < p) \rightarrow n2 \equiv m2\ (mod\ p)$.

The congruence relation is compatible with addition [4, 6], this means:
If $n1 \equiv m1\ (mod\ p)$ and $n2 \equiv m2\ (mod\ p)$, then: $n1 + n2 \equiv m1 + m2\ (mod\ p)$;
Which is equivalent to:
$[n1 + n2] \equiv [(n1\ mod\ p) + (n2\ mod\ p)]\ (mod\ p)\ (m1 < p)$ and $(m2 < p) \rightarrow [m1 + m2] < 2p \rightarrow [m1 + m2 - p] < p$.
Two cases:

* If $[m1 + m2] < p \rightarrow (n1 + n2)\ mod\ p = m1 + m2 \Leftrightarrow (n1 + n2)\ mod\ p = n1\ mod\ p + n2\ mod\ p$.
* If $[m1 + m2] \geq p \rightarrow$ then $0 \leq (m1 + m2) - p < p \rightarrow (n1 + n2)\ mod\ p = n1\ mod\ p + n2\ mod\ p - p$.

In the particular case where $n2 = 2$, we have: $2\ mod\ p = 2$ for any $p > 2$ (for $p = 2$, since $n1$ odd, we have: $n1\ mod\ p = (n1 + 2)\ mod\ p = 1$).
We write the preceding results for $n2 = 2$, we obtain the relation:
If $[n1\ mod\ p + 2] < p$ then $(n1 + 2)\ mod\ p = n1\ mod\ p + 2$.
If $[n1\ mod\ p + 2] \geq p$ then $(n1 + 2)\ mod\ p = n1\ mod\ p + 2 - p$.

3. Algorithm for Prime Search and Primality Check

3.1. **Method to generate a prime base by expansion.** As mentioned earlier, a prime base is a complete suite of primes $\{2, 3, \ldots, pm\}$. To generate the base, we start with an initial base that contains a reduced number of primes (minimal base: $\{2, 3\}$). If at a given stage, the base is $\{2, 3, \ldots, pi\}$, at next stage we perform the search algorithm to find all primes in interval $[pi+2, pi2]$. Found primes are append to the base: so we expand the prime base. We continue and iterate expansion process, until reach RAM limit. Beyond RAM limit, we can save the prime base to hard disk.
The expansion is very fast, the following example illustrates how the generation process works.

- Initial base contains the seven first primes $\{2, 3, 5, 7, 11, 13, 17\}$.
- After the first expansion, the base contains 61 primes,
- After second expansion, the base contains ≈ 4800 primes, and allows find primes up to $2^{31} - 1$.

3.2. **Search algorithm.** Goal of algorithm: find all primes in a given interval $[n1, n2]$, where $n1$ odd. The delivered result is an array of all primes found in interval $[n1, n2]$. We assume the knowledge of an initial PB.
Principle of the algorithm: we generate and use a PB $\{2, 3, \ldots, pm\}$. For each odd number n in search interval, the algorithm computes $n\ mod\ pi$ for all primes pi (with condition $pi2 < n2$) and store the mi results in an array to be reused for primality check of next odd number $n + 2$, we iterate for $n + 2, n + 4, \ldots$.
Algorithm:
Step 0: initialization:

- Enter the set of primes of initial prime base.

- Enter the start number of search interval ($n1$), and interval length ($n2 - n1$).
- Generate the PB to use in primes search (expansion method in Section 3.1).

Primes search iterations:

The logic of the algorithm imply two nested loops:

- Outer loop spans odd numbers in search interval, iterations start with $n1$ and move toward $n2$.
- Inner loop check perform primality check of each odd number n. The two loops perform the following:

* First iteration (outer loop): primality check for $n = n1$: inner loop compute $n1 \ mod \ pi$ for all primes pi by using modulo operator, and store the m results in a modulo-array. If $n1 \ mod \ pi \neq 0$ for all primes pi then $n1$ is prime, and $n1$ added to an array of primes in interval $[n1, n2]$.
* Next iterations: set $n = n + 2$, to move to next odd number. Inner loop compute $(n + 2) \ mod \ pi$, using relation established by MRM method:

  If $(n \ mod \ pi + 2 < pi)$ then $(n + 2) \ mod \ pi = n \ mod \ pi + 2$.

  Else If $(n \ mod \ pi + 2 \geq pi)$ then $(n + 2) \ mod \ pi = n \ mod \ pi + 2 - pi$.

  Value $(n + 2) \ mod \ pi$ replace value $n \ mod \ pi$ in modulo-array. If $(n + 2) \ mod \ pi \neq 0$ for all pi then $(n + 2)$ is prime, and added to the array of primes in interval. Iterations continue for the next odd numbers. When either loops terminate, primes-array contain all primes in interval $[n1, n2]$.

The algorithm requires the storage in RAM of the PB, however the filtering process explained below allows overcoming memory constraints. The algorithm use modulo operator only for first odd number in interval, for other numbers, an addition replace modulo operator. Time complexity of the classical divisibility algorithm is O $(n\sqrt{n}/ \log \ n)$ [3]. Since addition is much more efficient than modulo operation, time efficiency is improved notably.

3.3. **Means to overcome memory constraints: Filtering process.** Memory requirements are reasonable for small numbers. For larger numbers we use the following filtering method to overcome memory constraints. The main idea of the filtering process is to store the PB in a Database on hard disk, and divide the PB into successive subsets or segments Sk. Instead of performing primality check as an entire task, we break it into sub-tasks Tk performed independently; each sub-task requires segment Sk of PB in RAM. Each sub-task Tk perform the following:

- Process sequentially numbers of search interval (after excluding multiples of first primes).
- Check divisibility of the current number n by primes in segment Sk: the test $n \ mod \ pi \neq 0$ for all primes pi with $pi2 < n$ is replaced by $n \ mod \ pi \neq 0$ for primes pi with $pi2 < n$ AND pi in segment Sk.
- Filter-out composites and keep in results array only numbers non-divisible by primes of segment Sk.

Sub tasks are independent from each other, and may be executed sequentially on a single workstation (WS), or in parallel on a pool of networked WSs, sharing the primes Database. Each WS loads and keeps in RAM the primes in segment Sk used for its sub-task Tk. Numbers shared by all result arrays delivered by all sub-tasks are primes. With this approach, we can reach primes with several dozens of digits.

4. Conclusions

This paper propose a deterministic algorithm for primes search, and a new method to compute modulo. The proposed method use congruence relations applied to successive odd numbers tested for primality, and results in a new computing approach where divisions are replaced by additions. Unlike AKS algorithm [5], considered to have only a theoretical interest but is not practical to implement on computers [1, 7], the proposed MRM method use a simple mathematical formalism but have an obvious practical interest. The proposed algorithm offer an additional advantage; it is not limited to testing the primality of a single number, but look for all prime numbers in a given integer range.

The algorithm is working (implemented with Java NetBeans platform). It generates a prime base containing the first 5000 primes, find all primes in a given interval, within the limit of 2^{31}, and is very fast. To find larger primes (some dozens of digits), the program need simple changes: store found primes in a Database, use the Java BigInteger Class, and use the filtering process described in Section 3.3.

To look for even bigger prime numbers, we have developed a second algorithm (will be presented in a future article), that use another method for primality testing but works without a prime base, and is therefore applicable to find primes with hundreds of digits.

Further work includes the following:

* Implementation of the filtering process, save the prime base in a database on hard disk.

* Modify the Modulo Relations method to search specifically twin primes.

Conflicts of Interest

I declare that there are no conflicts of interest regarding the publication of this article. Farhat Mechkene, author of the article.

References

[1] Duta, C., Gheorghe, L., Tapus, N., Framework for Evaluation and Comparison of Primality Testing Algorithms, 20th Int. Conf. on Control Systems and Science, 2015. 4

[2] Kumar, A., Kim, T., Lee, H., An Improved Divisibility Test Algorithm for Primality Testing, Ubiquitous Information Technologies and Applications, pp. 547–554, Y.-H. Han et al. eds., 2013. 1

[3] Liang, Y.D., Efficient Algorithms for Finding Prime Numbers In Introduction to Java Programming, 10th Edition, Pearson, pp. 860-866, 2015. 1, 3.2

[4] Riesel, H., Prime Numbers and Computer Methods for Factorization, Chap. Basic Concepts in Higher Algebra, Springer Science+Business Media, LLC 2012, Modern Birkhäuser. 2

[5] Schoof, R., Four Primality Testing Algorithms, Algorithmic Number Theory, MSRI Publications, Volume 44, 2008. 4

[6] Wang, X., Mathematical Foundations of Public Key Cryptography, CRC Press, pp.27-30. 2016. 2

[7] Yan, S. Y., Computational Number Theory and Modern Cryptography, Chap. Primality Testing, Wiley, 2017. 4