American Journal of Computational Mathematics, **2013, 3, 86-89**
http://dx.doi.org/10.4236/ajcm.2013.31014 Published Online March 2013 (http://www.scirp.org/journal/ajcm)

A Sieve for Prime Based on Extension Form of Not Prime

Gabriele Martino
Via Cornelia, Rome, Italy
Email: martino.gabri@gmail.com

Received October 10, 2012; revised November 21, 2012; accepted December 3, 2012

ABSTRACT

This paper will illustrate two versions of an algorithm for finding prime number up to *N*, which give the first version complexity

$$O\left(c1\cdot N\cdot\left(\log\sqrt{N}-c2\right)\right) \tag{1}$$

where *c*1, *c*2 are constants, and *N* is the input dimension, and gives a better result for the second version. The method is based on an equation that expresses the behavior of not prime numbers. With this equation it is possible to construct a fast iteration to verify if the not prime number is generated by a prime and with which parameters. The second method differs because it does not pass other times over a number that has been previously evaluated as not prime. This is possible for a recurrence of not prime number that is (mod 3) = 0. The complexity in this case is better than the first. The comparison is made most with Mathematics than by computer calculation as the number *N* should be very big to appreciate the difference between the two versions. Anyway the second version results better. The algorithms have been developed in an object oriented language.

Keywords: Prime Number; Equation; Sieve

1. Introduction

Prime numbers are numbers that can be divided only by one or by themselves in order to get integer number. They are probably the most famous number's series. For finding the primes up to *N*, 2 main Sieves have been developed. One is the Eratosthenes Sieve [1] that is based on eliminating the not prime that is generated by multiplies of numbers over a list. For this Sieve the complexity is $O(N)$. Another Sieve is the Atkins and Bernstein Sieve [2,3], which is based on the remainder of modulo sixty. In this case the complexity is $O\left(\frac{N}{\log\log N}\right)$. This paper aims to continue on this work improving the sieving. Observe that all the odd numbers are not prime if they are of the form

$$np = p*p+2kp \tag{2}$$

where *p* is a prime number > 1 and $k=0,1,2,3,\cdots$ and *np* is for not prime.Taking all the numbers for *x* odd where the set of the odds is bigger than the set of prime, and *k* integer is possible to cover all the Not Prime set. There are some repetition in particular those that have remainder 0 divided by 3 (Of course the *x* = 3 series must be generated first and after a number np can be considered not prime already counted if divided by 3 gives remainder 0). In the first algorithm we give a solution with repetition in building the set of not prime. In the second we avoid the repetition considering that a square number generated from an odd, generate vary *k*, two possible kind of repetition: in the first is repeated for each *k* (mod 3) = 2, and in the second for each *k* (mod 3) = 3.

2. Methods

2.1. Method I

The follow is a explanation of the method for the search of prime. We want to count the prime up to *N*. We extend our iteration variable in the interval $\left[0,\sqrt{N}\right]$. The numbers (mod 2) = 0 are not primes. For the prime number we generate first $p*p$ as not prime and after generate $p*p+2\cdot k\cdot p$. With *p* that can be an also an odd number because the set of primes is in the set of odds except that number 2 and $p<\sqrt{N}$. The algorithm is presented in the appendix.

2.2. Method II

Another better result can achieved thinking that the numbers (mod 3) = 0, repeat in the count. I observed that

there are 3 kind of repetition. $np\left(\mathrm{mod}\,3\right)=0$ for every k like $9,27,\cdots$. $\left(\mathrm{odd}*\mathrm{odd}+k*\mathrm{odd}\right)\left(\mathrm{mod}\,3\right)=0$ for every k counted 1 (not counted 0) in the sequence 101-101-101 like 5. And for every k in the sequence 110-110-110 like 7.

We should make it in a way that affords us to go over a case of 0 and go to next $k = k + 1$, avoiding count repetition.

Example $7 \times 7 = 49$ (mod 3 different from 0) not counted; $7 \times 7 + 2 \times 7 = 63$ (mod 3 = 1) already counted in the 3 sequence; $7 \times 7 + 4 \times 7 = 77$ not counted. They gives the sequence 101.

3. Results

To count the primes up to 1,000,000 both methods take 3 seconds. This means that only for bigger N a comparison of methods is possible. The second should anyway take less time than the first. The test has been done with an machine Asus Intel i7 with 4 GB Ram, with an Operative System Windows 7 and the platform Java.

REFERENCES

[1] Wikipedia. http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

[2] A. O. L. Atkin and D. J. Bernstein, "Prime Sieves Using Binary Quadratic Forms," *Mathematics of Computation*, Vol. 73, 2004, pp. 1023-1030. doi:10.1090/S0025-5718-03-01501-1

[3] Wikipedia. http://en.wikipedia.org/wiki/Sieve_of_Atkin

[4] http://math.stackexchange.com/questions/141224/finite-sum-of-reciprocal-odd-integers

[5] T. M. Apostol, "Calculus I," Bollati Boringhieri, 2003, p. 481.

[6] Wikipedia. http://en.wikipedia.org/wiki/Harmonic_series

Appendix

Algorithm I

N we count the prime up to *N*
primes [*N*] = *true* for each element
$r = \sqrt{N}$
for $i = 1 \to i < N$ **do**
  primes [*i*] = *false*
  $i = i + 2$
end for
for $i = 2 \to i < r$ **do**
  $j = (i+1) * (i+1)$
  while $j <= N$ **do**
    if $j \pmod 3 != 0 \ \| \ i == 2$ **then**
      $primes[j-1] = false$
    end if
    $j = j + 2 * (i+1)$
  end while
  $i = i + 2$
end for
return primes

Computational Complexity of Algorithm I
We call *C*(*A*) the complexity of the second loop, whereas, *T*(*A*) is the complexity of all the code.

$$C(A) + O\left(\frac{N}{2}\right) = T(A)$$

We use the approximation

$$\sum_{I=1,\text{odd}}^{N} (I) = \left(\frac{N+1}{2}\right)^2 \tag{3}$$

as stated in [4]

$$\sum_{k=1}^{N} \frac{1}{a \cdot k + b} = \frac{1}{a} \cdot \left(\sum_{k=1}^{n+b/a} \frac{1}{k} - \sum_{k=1}^{\frac{b}{a}} \frac{1}{k} \right) \tag{4}$$

and

$$\sum_{k=1}^{N} \left(\frac{1}{k}\right) \square \log N \tag{5}$$

as stated in [5,6]

The complexity of the first method is given by the 2 nested loop in the algorithm.

$$C(A) = \sum_{I=3,\text{odd}}^{\sqrt{N}} \left(\frac{\left(N - I^2\right)}{2 \cdot I} \right) =$$

Separating the members of addition

$$\sum_{I=3,\text{odd}}^{\sqrt{N}} \frac{N}{2 \cdot I} - \frac{I}{2} =$$

Evaluating the second member with (3)

$$\sum_{I=3,\text{odd}}^{\sqrt{N}} \frac{N}{2 \cdot I} + \frac{-\left(\sqrt{N}+1\right)^2}{2^3} + \frac{1}{2} =$$

Taking out of the first member the constant operation

$$\frac{N}{2} \cdot \left(\sum_{I=3,\text{odd}}^{\sqrt{N}} \frac{1}{I} \right) - \frac{\left(N + 2\sqrt{N} + 1\right)}{2^3} + \frac{1}{2} =$$

Changing the form for odd number at first member

$$\frac{N}{2} \left(\sum_{k=2}^{\frac{\sqrt{N}}{2}+1} \frac{1}{2k-1} \right) - \frac{\left(N + 2\sqrt{N} + 1\right)}{2^3} + \frac{1}{2} =$$

Extending the sum adding and subtracting the same element

$$\frac{N}{2} \left(\sum_{k=1}^{\frac{\sqrt{N}}{2}+1} \frac{1}{2K-1} \right) - N/2 - \frac{\left(N + 2\sqrt{N} + 5\right)}{2^3}$$

Using the (4)

$$\frac{N}{2} \left(\sum_{k=1}^{\frac{\sqrt{N}}{2}+1+\frac{1}{2}} \frac{1}{K} - \sum_{k=1}^{\frac{1}{2}} \frac{1}{K} \right) - \frac{N}{2} - \frac{\left(N + 2\sqrt{N} + 5\right)}{2^3}$$

Using the (5)

$$\frac{N}{2} \left(\log\left(\sqrt{N} + 3\right) \right) - \frac{\left(N + 2\sqrt{N} + 5\right)}{2^3}$$

That asymptotically goes like

$$T(A) \to c1 \cdot N \left(\log \sqrt{N} - c2 \right)$$

Algorithm II

N we want to count the primes until *N*
primes [*N*] = true
$r = \sqrt{(N)}$
for $i = 1; i < N; i = i + 2$ **do**
  primes [*i*] = false
end if
for $i = 2; i < r; i = i + 2$ **do**
  square = $(i+1) * (i+1)$
  j = square
  k = 0
  condition = "";
  if $j + 2 * (i+1) \pmod 3 == 0$ And $(i+1)! = 3$ **then**
    condition = "101";
  else if $j + 4 * (i+1) \pmod 3 == 0$ And $(i+1)! = 3$ **then**
    condition = "110"
  else if $(i+1)! = 3$ **then**
    condition = "111"


```
    j = N;
end if
while   j ≤ N   And! ("111") do
   if j mod 3! = 0 or i == 2 then
      primi[ j −1] = false
   end if
   k = k +1
   if condition = ("101") And  (k) mod 3 == 1  then
      k = k +1
   end if
   if condition = ("110") And  (k) mod 3 == 2  then
      k = k +1
   end if
    j = square + 2 ∗ k ∗ (i +1)
  end while
end for
returns primes
```

