Prime Numbers
Cycle Searching Algorithm

Samuel Jansen

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
870 Dra. Maria Zélia Carneiro de Figueiredo, St
Canoas, Rio Grande do Sul, Brazil
ZIP Code 92412-240

Abstract. This paper was written as an exercise. It contains some bad writings and probably has mistakes here and there.

It first shows a set of equations to calculate the nth prime number. Then it verges into computational aspects of the ideas behind this set of equations. Then we define a Standard Searching Algorithm - SSA and present the Cycle Searching Algorithm - CSA. The main goal of this paper is making use of this CSA to generate a list of all prime numbers up to a certain limit faster than any SSA. At the end, this paper show the CSA Heuristic and point some clever ways to make use of it.

1. Mathematical Background

A positive integer p is called a *prime number* when it has only two positive integer divisors: 1 and p itself. In other hand, any number that is not a prime number is called *composite number*. It's widely know that the set P of all prime numbers is infinite and we can enumerate its elements in the following way:

$$P = \{p_1, p_2, p_3, \ldots, p_{n-1}, p_n, p_{n+1}, \ldots\}$$

That said, it emerges a reasonable question: Is there function that returns the p_n number for any given positive integer n? In other words:

Is there any equation to find the nth prime number?

This question was the first inspiration for the writing of this paper and in order to answered it, let's start going through the following theorems

Theorem 1. If the quantity of composite numbers from 1 to a positive integer $m-1$ is equal que the quantity of composite numbers from 1 to a positive integer m, then $p_n = m$, where $n = m - E(m)$

Proof of Theorem 1: The proof is by contradiction.

Let $m > 0$ and $m \in \mathbb{N}$. Let $E(m)$ be the function that returns the quantity of all composite numbers from 1 to m. Let $n = m + E(m)$. Now, lets say $m > 2$ is a prime number.

Now, let's say $E(m) > E(m-1)$. But, if $E(m) > E(m-1)$, it's a contradiction because the quantity of composite numbers from $m-1$ to m just got bigger, and therefore m is not a prime number.

So, aways $E(m) = E(m-1)$, m is a prime number. Also, if there are $E(m)$ composite numbers from 1 to m, it's because there must be n prime numbers from 1 to m as well. So, aways $E(m) = E(m-1)$, $p_n = m$

Theorem 1 is proofed

Although it's a simple definition that finds the *nth prime number*, $E(m)$ isn't a simple function to define. But it's possible. So, in order to make it easy to understand the definition of $E(m)$, let's first go through some definitions.

Let P be the set of all prime numbers, where:

$$P = \{p_1, p_2, \ldots, p_i, \ldots, p_{k-1}, p_k, \ldots\}$$

Let $p_1 < p_2 < \ldots < p_i < \ldots < p_{k-1} < p_k < \ldots$. Let $2 \le i < k$. Let's say there are no prime numbers between p_i and p_{i+1}.

Let j be any positive integer where $p_k < j < p_k^2$ and

$$\begin{gathered} j \equiv x_1 (mod\, p_1), \\ j \equiv x_2 (mod\, p_2), \\ j \equiv x_3 (mod\, p_3), \\ \ldots, \\ j \equiv x_i (mod\, p_i) \\ \ldots, \\ j \equiv x_{k-1} (mod\, p_{k-1}), \\ j \equiv x_k (mod\, p_k), \end{gathered}$$

where $x_1, x_2, x_3, \ldots, x_i, \ldots, x_{k-1}, x_k$ are the minimum possible value for the congruence [1].

Let $y_i = \lim_{y \to 0} y^{x_i}$.

Let $C(j) = \lim_{y \to 0} y^{(y_1 + y_2 + y_3 + \ldots + y_i + \ldots + y_{k-1} + y_k)}$

Theorem 2. $E(m) = \sum_{j=1}^{m} (1 - C(j))$

Proof of Theorem 2: The proof is by contradiction.

Once $C(j) = \lim_{y \to 0} y^{y_1+y_2+y_3+\ldots y_i+\ldots+y_{k-1}+y_k}$, $C(j)$ can only be one of the two values: 0 or 1.

$$C(j) = 0, \; 0 < y_1 + y_2 + y_3 + \ldots y_i + \ldots + y_{k-1} + y_k$$
$$C(j) = 1 \text{ if } 0 = y_1 + y_2 + y_3 + \ldots y_i + \ldots + y_{k-1} + y_k$$

Similarly, once $y_i = \lim_{y \to 0} y^{x_i}$, y_i can only be one of the two values: 0 or 1.

$$y_k = 0, \; x_k > 0$$
$$y_k = 1, \; x_k = 0$$

So, if , then $y_1 = 0, \; y_2 = 0, y_3 = 0, \; \ldots, y_i = 0, \; \ldots, y_{k-1} = 0, y_k = 0$, otherwise, $C(j) = 0$. It means that if any $x_i = 0$, $C(j) = 1$.

Also, if $x_k = 0$ then j divides x_k, otherwise j inst congruent to 0 module p_k

Taking everything in consideration, aways any p_k divides j, $C(j) = 0$ and j is a composite number. In other hand, aways all p_k can't divides j, $C(j) = 1$ and j is a prime number.

Now, let's say m is a prime number and $E(m) > E(m-1)$. But, if $E(m) > E(m-1)$, it's a contradiction because once m is a prime number, there is no p_k that can divides j when $j = m$.

Theorem 2 is proofed

Summarizing so far, there is actually a definition for the function E(m) such as we can use to identify the *nth prime number* through the following set of equations:

$$n = m - E(m)$$
$$E(m) = \sum_{j=1}^{m} (1 - C(j))$$
$$\text{Aways } E(m) = E(m-1), \; p_n = m$$

The computational focus of this paper will be writing an optimal way to calculate $C(j)$ (or $C(m)$ as j is the variable that iterates from 1 up to m in the sum of $E(m)$)

2. Defining and Calculating $C_n(m)$

Calculating the value of $C(m)$ is basically the same as knowing if m is a prime number or not. There is a lot of ways to evaluate if a number m is a prime number or not, and we could make us of some of them. But, for the purpose of this paper, we will define and make use of a boolean function called $C_n(m)$ to reach same outputs that $C(m)$ has. In order to achieve it, we first need to define $C_n(m)$:

$C_n(m)$ is a boolean function that assign false (0) to each m multiple of p_1, p_2, p_3, ..., p_n, including themselves. All remmaning outputs are true (1)

Theorem 3. The $C_n(m)$ image is partially equal to $C(m)$ image.

Proof of Theorem 3: We know that $C(m)$ can only have one output for each m number, and this output can only have one of the following two values: 0 or 1. We will start writing a partial image of $C(m)$ and attribute its values to $C_1(m)$. So, aways p_1 do not divides m, $C_1(m) = 1$. In other hand, aways p_1 divides m, $C_1(m) = 0$.

Once we do that, the next prime number (p_2) can only be the lowest $m > 1$, which $C_1(m) = 1$. Then, we repeat the same process thinking only about p_1 and p_2 at the same time and we attribute these new images to $C_2(m)$. So, aways p_1 and p_2 do not divides m, $C_2(m) = 1$. In other hand, aways p_1 or p_2 divides m, $C_2(m) = 0$.

After that, the next prime number (p_3) can only be the lowest $m > 1$, which $C_2(m) = 1$. Then, we repeat the same process thinking only about p_1, p_2 and p_3 at the same time and we attribute these new images to $C_3(m)$. So, aways p_1, p_2 and p_3 do not divides m, $C_3(m) = 1$. In other hand, aways p_1 or p_2 or p_3 divides m, $C_3(m) = 0$.

We repeat this process over and over until reach $C_n(m)$. Bellow, we have tables making a comparative between $C_n(m)$ and $C(m)$ image values. Note that $C(m)$ cannot return true values for any multiple of p_1, p_2, p_3, ..., p_n, excluding themselves. With $C_n(m)$, there is difference: $C_n(m)$ cannot return true values for any multiple of p_1, p_2, p_3, ..., p_n, including themselves.

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$p_1 = 2$	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$C_1(m)$	1	0	**1**	**0**	**1**	**0**	**1**	**0**	1	**0**	**1**	**0**	**1**	**0**	1	**0**	**1**	**0**	**1**
$C(m)$	0	1	**1**	**0**	**1**	**0**	**1**	**0**	0	**0**	**1**	**0**	**1**	**0**	0	**0**	**1**	**0**	**1**

Table 1: Comparative between $C_1(m)$ and $C(m)$

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$p_1 = 2$	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$p_2 = 3$	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
$C_2(m)$	1	0	0	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**
$C\,(m)$	0	1	1	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**

Table 2: Comparative between $C_2(m)$ and $C\,(m)$

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$p_1 = 2$	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$p_2 = 3$	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
$p_3 = 5$	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1
$C_3(m)$	1	0	0	**0**	0	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**
$C\,(m)$	0	1	1	**0**	1	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**

Table 3: Comparative between $C_3(m)$ and $C\,(m)$

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$p_1 = 2$	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$p_2 = 3$	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
$p_3 = 5$	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1
$p_4 = 7$	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1
$C_4(m)$	1	0	0	**0**	0	**0**	0	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**
$C\,(m)$	0	1	1	**0**	1	**0**	1	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**

Table 4: Comparative between $C_4(m)$ and $C\,(m)$

So, the $C_n(m)$ image has to be partially equal to $C(m)$ image, especially when m assume values higher than p_n.

Theorem 3 is proofed

Take a note that we can always "catch the next prime number", p_{n+1}, from $C_n(m)$. All we have to do is take the lowest $m > 1$ value where $C_n(m) = 1$. In this case, m is p_{n+1}.

Theorem 4. The number m is a prime number aways $p_n < m < p_n^2$ and $C_n(m) = 1$,

Proof of Theorem 3: Let's say we make an Eratosthenes sieve [1] that excludes all composite numbers that are multiple to at least one f the following prime numbers: p_1, p_2, p_3, ..., p_n and call it as "p_n *Eratosthenes sieve*". We know that from the range p_n to p_n^2 [2] all numbers not taken out from this sieve are prime numbers.

So, aways $C_n(m) = 1$ for any m in this same range, m has to be prime for the same reason.

Theorem 4 is proofed

3. Writing $C_n(m)$ pseudocode and comparing it to a standard algorithm

The theorems 3 and 4 of this paper were implemented in a Portugol Studio pseudocode, which was called *Cycle Searching Algorithm - CSA*. In order evaluate its performance, it was made a *Standard Searching Algorithm - SSA* for us to make comparisons.

So before jump into CSA, we start better defining what is this SSA.

Standard Searching Algorithm - SSA.

P is a vector that stores all calculated prime numbers in crescent order. n_total is the quantity of prime numbers that are stored into P. The first value of n_total is 1.

The SSA basically starts knowing that the first prime number is 2 and stores it into $P[n\_total]$.

After that, it iterates m to all integer values from $P[n\_total]+1$ up to $P[n\_total]^2-1$ and successively evaluate the rest of the following mathematical expression:

$$\frac{m}{P[n\_total]}$$

Remember that in this first loop, $n\_total = 0$. If any division returns rest equals to zero, it skips this m value and move forward to the next one, which is $m = m + 1$

Always the same m returns rest different than zero in the division, m is stored into the next free position of P and n_total is incremented by one unit.

SSA - prime verification step

In order to check if m is a prime number SSA iterates m to all integer values from $P[n\_total]+1$ up to $P[n\_total]^2-1$ and successively evaluate the rest of the following mathematical expressions:

$$S = \sqrt{P[n\_total]}$$

$P[biggest]$ is the biggest prime number lower than S

$$\frac{m}{P[0]}, \frac{m}{P[1]}, \frac{m}{P[3]}, \cdots, \frac{m}{P[biggest]}$$

If any division returns rest equals to zero, it skips this m value and move forward to the next one, which is $m = m + 1$

Always the same m returns rest different than zero in all divisions, m is stored into the next free position of P and n_total is incremented by one unit.

The process is repeated up to $n\_total = n\_max - 1$, where n_max is the maximum quantity of values a vector can store in Portugal Studio

For the purpose of this paper, this is the definition of a Standard Searching Algorithm.

Cycle Searching Algorithm - CSA.

The CSA is the straight implementation of theorems 3 and 4 of this paper with one "extra detail". Let's get int it.

First, we store p_0 value in the first position of PC vector: $PC[0] = 2$.

Next, we do a boolean vector C with all values of $C_0(m)$.

Then, we do an integer vector called PNP (Possible Next Primes) with all $m+1$ values where $C[m] = 1$.

We "catch the next prime number" from $C[m]$ and store it into the next free position of PC vector: $PC[1] = 3$

CSA - prime verification step

And now comes the "extra detail". We iterate m *from* 1 *up to* n_max *value into the following loop:*

$next\_false\_position = PNP[m] * PC[1]$

if ($next\_false\_position < n\_max$) *then* $C[m] = false$

After this loop, $C[m]$ *has stored all values of* $C_1(m)$.

Once again we assign PNP values with all $m+1$ values where $C[m] = 1$, then we "catch the next prime number" from $C[m]$ and store it into PC vector, then we iterate m from 1 up to n_max value into the following loop and, once again, $C[m]$ has stored all values of $C_2(m)$.

We do this process over and over up to $PC[\sqrt{n\_max}+1]$. After that, all $m+1$ values where $C[m] = 1$ are prime numbers.

While SSA has a "prime number checking" step based on "successive divisions" while CSA turns this step based into a "successive multiplication"

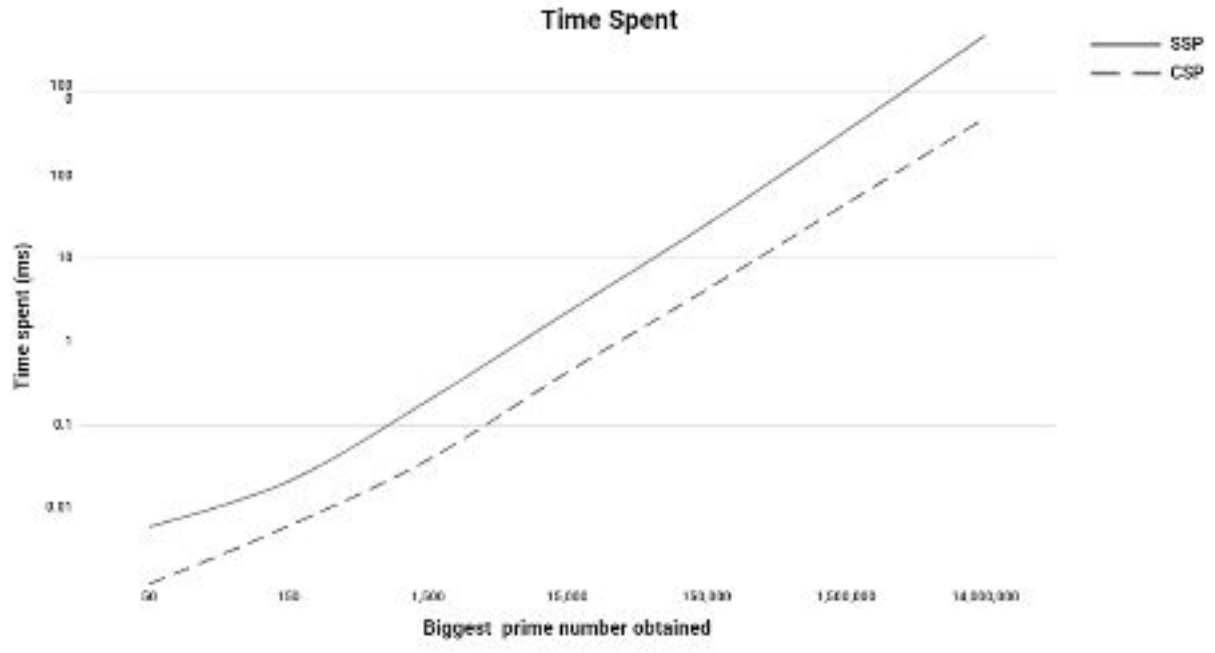
4. Performance results

It was made several tests and iterations to gather an average of time spent and quantity of iterations a "prime testing amount" in each one of the pseudocodes (SSA and CSA). The tests were made into a 3.0 GHz processor, 16GB of RAM memory with no parallel processing data. The results follow below:

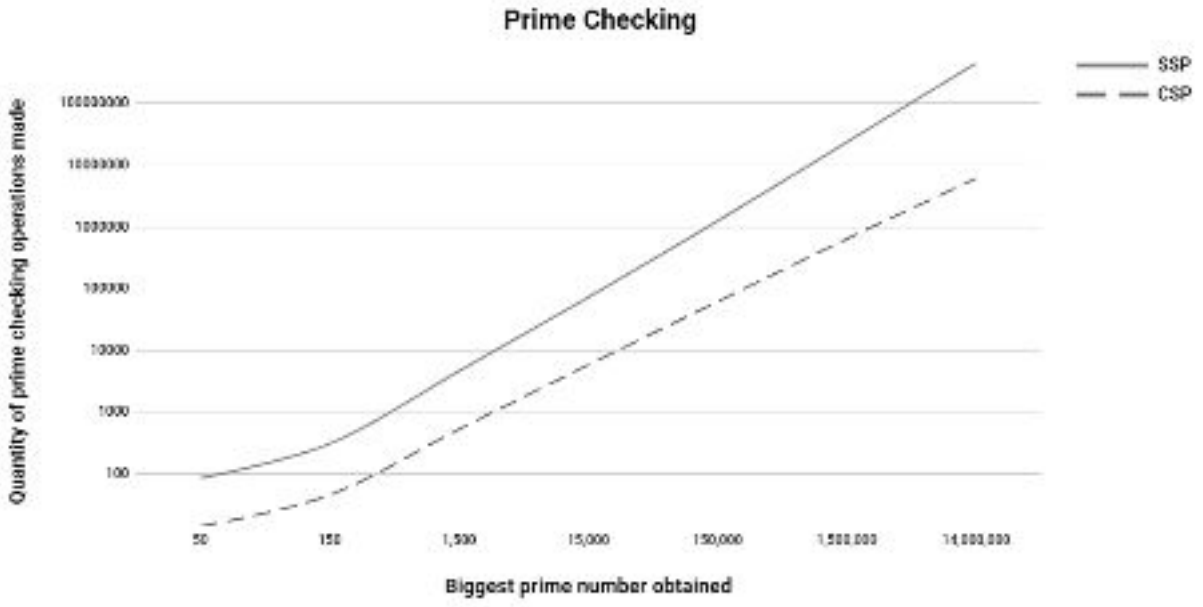
Biggest prime number	50	150	1500	15000	150000	1500000	14000000
SSA time spent (ms)	0,0059	0,0207	0,1941	2,252	25,73	343,2	4868
CSA time spent (ms)	0,0012	0,006	0,0376	0,4244	4,228	45,42	493,6
SSA prime checking operations	850000	3050000	45760000	722680E3	123814E5	2360965E5	44779367E5
CSA prime checking operations	14	45	523	5776	61228	636045	6090445
n	14	34	238	1753	13847	114154	910076

Table 5: Performance results. Observe that $2 * (n + CSA\_prime\_checking\_operations) \sim Biggest\_prime\_number$

We can see on the plots below the time spent by each algorithm to calculate all prime numbers up to a biggest prime number and how much operations each one made to do so.



Plot 1: Time spent to calculate all prime numbers from 2 up to the biggest prime number obtained.



Plot 2: Quantity of operations do calculate all prime numbers from 2 up to the biggest prime number obtained

4. CSA Heuristic

Take a look at the performance results. There is a very interesting heuristic relation between p_n, n and $CSA\_prime\_checking\_operations$ values. p_n is approximately equal to 2 times n plus 2 times $CSA\_prime\_checking\_operations$.

$$p_n \sim 2 * (n + CSA\_prime\_checking\_operations)$$

Now we go back to the beginning of this paper. Once $n = m - E(m)$ and, we now have a way to write an approximation of $E(p_n)$ value

$$\pi(p_n) \sim \frac{p_n}{ln(p_n)} \text{ , the Prime Number Theorem [2]}$$

$$E(p_n) \sim n + 2 * CSA\_prime\_checking\_operations$$

$$p_n \sim E(p_n) + \pi(p_n)$$

$$n \sim \pi(p_n)$$

$$p_n \sim \frac{p_n}{ln(p_n)} + \frac{p_n}{ln(p_n)} + 2 * CSA\_prime\_checking\_operations$$

$$CSA\_prime\_checking\_operations \sim p_n * (\frac{1}{2} - \frac{1}{ln(p_n)})$$

Assuming this $CSA\ Heuristic$, we can attribute an approximate value to p_n and predict how much time it will take to calculate all prime numbers up to p_n.

4. Conclusion

It was presented and proved a set of equations that lead us the *nth prime number*. CSA, an algorithm based on the ideas presented on this set of equations, was then presented as well. CSA performance was tested and compared to SSA performance. CSA is increasingly faster than SSA as the biggest prime number number increases. It's because CSA does less prime checking operations and therefore reaches a list of all prime number up to a biggest one faster than SSA.

At the end, it was presented the CSA Heuristic that can be used to approximate the amount of time needed to calculate the *nth prime number*, in a way that we can finish this paper giving it a spotlight

$$n = m - E(m)$$

$$E(m) = \sum_{j=1}^{m} (1 - Cn(j)) \, , \; p_n < i < p_n^2$$

$$\text{Aways } E(m) = E(m-1) \, , \; p_n = m$$

And the amount of time needed to calculate it is directly proportional to $p_n * (\frac{1}{2} - \frac{1}{ln(p_n)})$

5. References

[1] Introdução à Teoria dos Números [José Plínio de Oliveira Santos], 3 ed. Rio de Janeiro : IMPA, 2014. ISBN 978-85-244-0142-8

[2] Teoria dos Números [Fabio Brochero Martinez], [Carlos Gustavo Moreira], [Nicolau Saldanha] e [Eduardo Tengan], 1 ed. Rio de Janeiro : IMPA, 2010. ISBN 978-85-244-0312-5