Computing $\pi(x)$: the combinatorial method

Tomás Oliveira e Silva

Abstract — This article presents a self-contained description of the method proposed by Deléglise and Rivat — which has roots on earlier work by Meissel, by Lehmer, and by Lagarias, Miller and Odlyzko — to compute $\pi(x)$, which is the number of primes not larger than $x$.

To make life easier to programmers, the major parts of the computation of $\pi(x)$ are also presented in algorithmic form.

The more interesting low-level computational details are presented in the form of C code snippets.

Resumo — Este artigo descreve detalhadamente o método proposto por Deléglise e Rivat — que é baseado em trabalhos anteriores de Meissel, de Lehmer, e de Lagarias, Miller e Odlyzko — para calcular $\pi(x)$, que é o número de números primos até $x$.

I. INTRODUCTION

The problem of the computation of $\pi(x)$, which is the number of primes not larger than $x$, has been drawing the attention of countless mathematicians for a long time. Gauss, based on empirical evidence garnered from tables of primes compiled by hand, conjectured that

$$\lim_{x \to \infty} \frac{\pi(x)}{\frac{1}{2} x \log x} = \lim_{x \to \infty} \frac{\pi(x)}{\log x} = 1.$$ 

This fundamental result, known as the prime number theorem, was first proved in 1896 almost simultaneously by Hadamard and de la Vallée-Poussin [1].

The computation of $\pi(x)$ for a given value of $x$ was done at first using tables of prime numbers made using the sieve of Eratosthenes or one of its variants. Legendre was the first to show that the computation of $\pi(x)$ does not require the explicit determination of all primes up to $x$. His formula, which is based on the inclusion-exclusion principle and only requires the knowledge of the primes up to $\sqrt{x}$ [2], has a number of non-zero terms which grows asymptotically like $x$. This represents a modest improvement on the direct enumeration of all primes up to $x$ done using the sieve of Eratosthenes or one of its variants, which requires an amount of time $1$ which grows like $x \log \log x$.

In the last quarter of the XIX century, Meissel [2] found a more efficient way to compute $\pi(x)$. In 1959, Lehmer [3], [2] systematized and simplified Meissel’s method, and made what was probably the first computer program able to compute $\pi(x)$ without using a sieve. Lehmer’s method requires an amount of time which grows like $x \log^{-4} x$.

(a small improvement over Legendre’s formula), and an amount of space which grows like $x^{1/3} \log^{-1} x$ (a significant improvement over Legendre’s formula). In 1963, Mapes [2] found a way to compute $\pi(x)$ that requires amounts of time and space which grow like $x^{7/10}$ (this exorbitant space requirement precludes the utilization of Mapes’ method for large values of $x$).

In 1985, Lagarias, Miller, and Odlyzko, based on the work of Meissel and Lehmer, found a much better way to compute $\pi(x)$. Their method requires an amount of time which grows like $x^{2/3} \log^{-1} x$, and uses an amount of space which grows like $x^{1/3} \log^2 x$ [4], [5]. In 1996, Deléglise and Rivat [6] found a way to save a factor of $\log x$ in the amount of time required to compute $\pi(x)$ using the Lagarias-Miller-Odlyzko method, at the expense of an increase by a similar factor in the amount of space used in the computation.


Over the years the record of computation of $\pi(x)$ has been steadily growing. The present record appears to be the computation of $\pi(4 \cdot 10^{22})$, performed in 2001 by Gourdon.

All previously mentioned methods use combinatorial ideas to compute $\pi(x)$. In 1987, Lagarias and Odlyzko [8], [5] found an analytical way to compute $\pi(x)$ which has a better asymptotic computational complexity than the Deléglise-Rivat method. Nonetheless, so far no one was able to use it in record-breaking computations of $\pi(x)$.

The rest of this paper is organized as follows. Section II describes the improvements made by Deléglise and Rivat, by Lagarias, Miller, and Odlyzko, and by Lehmer on the Meissel method of computation of $\pi(x)$; the description of the data structure [9] used to sieve efficiently an interval is relegated to appendix A. Section III presents tables with values of $\pi(x)$ for some powers of two and some powers of ten (these last are in perfect agreement with Gourdon’s computations). Table I presents some of the notation used in the paper; the symbols there defined are not defined elsewhere in the paper.

This paper does not describe the improvements on the combinatorial method made by Gourdon, not does it describe how to parallelize the computation of $\pi(x)$. These are left to another publication.

II. COMPUTATION OF $\pi(x)$

For an integer $a \geq 0$ and a real number $x \geq 1$, let $\phi(x, a)$ be the number of positive integers, not larger than $x$, which are co-prime to each of the first $a$ primes, i.e.,

$$\phi(x, a) = \sum_{n=1}^{\lfloor x \rfloor} \left[ \left\lfloor \frac{n}{p_{\min(n)}} \right\rfloor > p_a \right].$$
Also, let \( \phi_k(x, a) \) be the number of positive integers, not larger than \( x \), which have exactly \( k \) prime factors (counting repetitions), each one of which larger than \( p_a \), i.e.,

\[
\phi_k(x, a) = \sum_{n=1}^{\lfloor \log x / \log p_a \rfloor} \left\lfloor \frac{\log x}{\log p_a} \right\rfloor \phi_k(n, p_a) \quad (1)
\]

For \( x \geq 1 \) it is obvious that

\[
\phi_0(x, a) = 1.
\]

In addition, for \( a \leq \pi(x) \) it is also obvious that

\[
\phi_1(x, a) = \pi(x) - a.
\]

Consequently, for \( x \geq 1 \) and \( a \leq \pi(x) \), it follows from (1) that

\[
\pi(x) = \phi(x, a) + a - 1 = \sum_{k=0}^{\left\lfloor \log x / \log p_a \right\rfloor} \phi_k(x, a). \quad (2)
\]

All methods of the Meissel type (combinatorial methods) are based on this equation. The computation of \( \phi(x, a) \) will be addressed in detail in subsection II-B. Depending on the value of \( a \), it may also be necessary to compute \( \phi_2(x, a) \), \( \phi_3(x, a) \), and so on. The Lagarias-Miller-Odlyzko and the Deléglise-Rivat methods use \( a = \pi(\sqrt{x}) \), \( 1 \leq a \leq \sqrt{x} \), with \( a \) growing slower that any power of \( x \), i.e., \( \kappa = O(x) \). Thus, they require the computation of \( \phi_2(x, a) \), a task that will be addressed in detail in subsection II-A. It is worthwhile to mention here that \( a = \pi(\sqrt{x}) \) is used in Legendre’s formula (in this case all \( \phi_2(x, a) \) vanish), and that \( a = \pi(\sqrt{x}) \) was used by Lehmer in [3] (in this case \( \phi_2(x, a) \) and \( \phi_3(x, a) \) must be computed).

A. Computation of \( \phi_2(x, a) \)

Let \( a = \pi(\sqrt{x}) \), with \( 1 \leq a \leq \sqrt{x} \). Thus, \( a \leq \pi(\sqrt{x}) \). When \( a = \pi(\sqrt{x}) \) it can be verified that \( \phi_2(x, a) \) vanishes. When \( \pi(\sqrt{x}) \leq a < \pi(\sqrt{x}) \) it can be verified that

\[
\phi_2(x, a) = \sum_{b=a+1}^{\left\lfloor \log x / \log p_b \right\rfloor} \left\lfloor \frac{\log x}{\log p_b} \right\rfloor \phi(\left\lfloor \frac{x}{p_b} \right\rfloor).
\]

By convention [10, exercise 2.1], \( \sum_{j=1}^{\left\lfloor \log x \right\rfloor} f(j) = 0 \); hence, (3) also gives the correct result for \( a = \pi(\sqrt{x}) \). Let

\[
z = \frac{2\sqrt{3}}{\alpha}.
\]

Since \( p_a \leq \alpha \sqrt{x} < p_{a+1} \) it follows, for \( a < b \leq \pi(\sqrt{x}) \), that

\[
\sqrt{x} \leq \frac{x}{p_b} \leq \frac{x}{p_{a+1}} < \frac{x}{\alpha \sqrt{x}} = z.
\]

Moreover, since \( z \leq \alpha^3 z < p_{a+1} \), it follows from (2) that

\[
\pi\left(\frac{x}{p_b}\right) = \phi\left(\frac{x}{p_b}, a\right) + a - 1.
\]

As will be seen in the next subsection, the computation of \( \phi(x, a) \) requires sieving the interval \([1, z]\), i.e., removing from it all multiples of the first \( a \) primes. After this is accomplished, each of the \( O(x^{1/2} \log^{-1} x) \) values of \( \phi\left(\frac{x}{p_b}, a\right) \) can be computed in \( O(\log z) \) steps using the method described in appendix A. Disregarding the cost of this sieve (accounted for latter on), it turns out that the computation of \( \phi_2(x, a) \) requires \( O(x^{1/2} \log \log x) \) steps (because it is necessary to generate all primes up to \( x^{1/2} \) using a segmented Eratosthenes sieve).

B. Computation of \( \phi(x, a) \)

Because \( \gcd(n, p_1 \cdots p_c) \) is, when \( c \) is fixed, a periodic function with period \( p_1 \cdots p_c \), the computation of \( \phi(x, c) \) can be done using the formula

\[
\phi(x, c) = \left\lfloor \frac{x}{p_1 \cdots p_c} \right\rfloor \phi(p_1 \cdots p_c, c) + \phi(\left\lfloor \frac{x}{p_1 \cdots p_c} \right\rfloor \mod p_1 \cdots p_c, c),
\]

in which

\[
\phi(p_1 \cdots p_c, c) = \varphi(p_1 \cdots p_c) = (p_1 - 1) \cdots (p_c - 1).
\]

To use (5) in an efficient way it is necessary to precompute a table of values of \( \phi(n, c) \) for \( 0 \leq n < p_1 \cdots p_c \). Obviously, this way of computing \( \phi(x, c) \) can only be used when \( c \) is very small (say, when \( c \leq 7 \)).
For $a > 0$ the computation of $\phi(x,a)$ can also be done recursively, using the formula

$$\phi(x,a) = \left\lfloor \frac{x}{a} \right\rfloor \sum_{n=1}^{\left\lfloor \frac{x}{a} \right\rfloor} \mu(n) \left\lfloor \frac{x}{n a} \right\rfloor.$$

The recursive application of this formula in the computation of $\phi(x,a)$ can be depicted as a binary tree (see figure 1), with $\phi(x,a)$ at its root. The nodes at level $k$ of this tree (with level 0 on top) have the values $\mu(n)\phi\left(\frac{x}{p_k}, a - k\right)$, with $p_{\max}(n) \leq p_k$ and $p_{\min}(n) > p_{\alpha - k}$. Since $\phi(x,0) = [x]$, growing the tree as much as possible yields

$$\phi(x,a) = \left\lfloor \frac{x}{a} \right\rfloor \sum_{n=1}^{\left\lfloor \frac{x}{a} \right\rfloor} \mu(n) \left\lfloor \frac{x}{n a} \right\rfloor.$$

It is not necessary to apply (6) when $x < p_{a+1}$, in which case $\phi(x,a) = 1$ if $x \geq 1$ and $\phi(x,a) = 0$ if $x < 1$. This last case does not occur if the recursion is terminated as soon as $x < p_{a+1}$. These observations, coupled with (2) and $a = \pi(\sqrt{x})$, are used in the incomplete C program presented in table II; it computes $\pi(x)$ in $O(\sqrt{x} \log^{-1} \log x)$ steps and uses $O(\sqrt{x} \log^{-1} \log x)$ space.

To reduce as much as possible the number of leaves in the computation of $\phi(x,a)$, in [4] Lagarias, Miller, and Odlyzko set $a = \pi(\sqrt{x})$, with $\alpha > 1$ carefully chosen, and use the following rule to continue to apply (6): split a node labeled $\pm \phi(y,b)$ if $b > c$ and $y \geq z$, with $z$ given by (4). This rule is equivalent to the following termination rule: do not split a node labeled $\pm \phi(y,b) = \mu(n)\phi\left(\frac{x}{y}, b\right)$ if either

(i) $b = c$ and $y \geq z$, i.e., $b = c$ and $n \leq \alpha \sqrt{x}$, or

(ii) $y < z$, i.e., $n > \alpha \sqrt{x}$ (if this happens then $b \geq c$).

Following [4], leaves of type (i) will be called ordinary leaves, and those of type (ii) will be called special leaves. For the sake of clarity, the contribution of these two kinds of leaves to the value of $\phi(x,a)$ will be treated separately.

**B.1 Computation of the contribution of the ordinary leaves to the value of $\phi(x,a)$**

In an ordinary leaf the conditions $n \leq \alpha \sqrt{x}$, $\mu(n) \neq 0$, and $p_{\min}(n) > p_c$ must be satisfied. The contribution of the ordinary leaves to the value of $\phi(x,a)$ is then given by

$$S_0 = \sum_{\frac{1}{n} \leq \alpha \sqrt{x}, p_{\min}(n) > p_c} \mu(n)\phi\left(\frac{x}{n}, c\right).$$

It is quite simple to identify the values of $n$ that satisfy the three conditions stated above. One possible way to do this requires the computation of the value of $\mu(n)p_{\min}(n)$ for $1 \leq n \leq \alpha \sqrt{x}$, which can be done using a simple modification of the sieve of Eratosthenes (see C code in table III), followed by the determination of the values of $n$ for which $\mu(n)p_{\min}(n) > p_c$. For each such $n$ it is possible to compute $\phi\left(\frac{x}{n}, c\right)$ quickly using (5).

It is obvious that the number of ordinary leaves cannot be larger than $\alpha \sqrt{x}$. Thus, the work required to compute $S_0$ takes no more than $O(\alpha x^{1/3})$ steps, to which must be added the $O(\alpha x^{1/3} \log \log x)$ steps required to compute $\mu(n)p_{\min}(n)$.

**B.2 Computation of the contribution of the special leaves to the value of $\phi(x,a)$**

In a special leaf the conditions $n > \alpha \sqrt{x}$, $\mu(n) \neq 0$, $p_{\min}(n) > p_c$ and $p_{\max}(n) \leq \alpha \sqrt{x}$ must be satisfied.
Moreover, its parent cannot be a special leaf, nor can it be an ordinary leaf, which implies that the special leaf was reached via the second term of (6). More precisely, if the parent node has the value \( \mu(m) \phi \left( \frac{a}{m}, b + 1 \right) \), with \( b + 1 > c \) and \( m < \alpha \sqrt{3} \), then the special leaf has the value \(-\mu(m) \phi \left( \frac{mp_{b+1}}{b}, b \right)\), with \( mp_{b+1} > \alpha \sqrt{3} \); by construction, \( p_{\text{max}}(mp_{b+1}) \leq \alpha \sqrt{3} \) is automatically ensured. The contribution of the special leaves to the value of \( \phi(x, a) \) is then given by

\[
S = - \sum_{c < b + 1 < a} \sum_{m \leq \alpha \sqrt{3} \text{ with } mp_{b+1} < \text{p}_{\text{min}}(m)} \mu(m) \phi \left( \frac{x}{mp_{b+1}}, b \right).
\]

(The case \( b + 1 = a \) cannot occur.) Since \( mp_{b+1} > \alpha \sqrt{3} \), the computation of \( S \) does not require values of \( \phi(y, b) \) for \( y \geq z \).

For a given value of \( b \) such that \( c \leq b < a - 1 \), the values of \( m \) that enter in the computation of \( S \) satisfy both

\[
\max \left( \frac{\alpha \sqrt{3}}{p_{b+1}}, p_{b+1} \right) < m \leq \alpha \sqrt{3}
\]

and \( |\mu(m)| \geq p_{b+1} \). Equipped with this way of identifying all special leaves, the computation of the special leaves can be organized in the following way. First, the contribution of the special leaves with \( b = c \) is computed using (5). Next, the multiples of the primes \( p_1, \ldots, p_e \) are removed from the interval \([1, z] \). Finally, the computation of the contribution of the rest of the special leaves is done by removing in succession from the interval \([1, z] \) the multiples of the primes \( p_{c+1}, \ldots, p_a \), extracting in between the required values of \( \phi(y, b) \). After this the primes up to \( p_a \) may be added back to the interval \([1, z] \), which simplifies slightly the computation of \( \phi_2(x, a) \).

Since the number of special leaves is large, the computation of the values of \( \phi(y, b) \) should be done as quickly as possible; the data structure used for this purpose, first used in a slightly modified form in arithmetic coders [9], [11], is described in detail in appendix A (the data structure used in [4] uses 50% more space and its update is slower, on average, by a factor of almost 2). With this data structure each value of \( \phi(y, b) \) can be evaluated in \( O(\log z) \) steps, (not counting the work required to perform the sieve operations).

### B.3 Improved computation of the contribution of the special leaves to the value of \( \phi(x, a) \)

The lower bound of (7) can take two distinct forms, according to whether \( p_{b+1}^2 \leq \alpha \sqrt{3} \) or whether \( p_{b+1}^2 > \alpha \sqrt{3} \). The first occurs when \( p_{b+1} \leq p_a \), where \( a^* = \pi(\sqrt{\alpha \sqrt{3}}) \), with \( a^* < a \), and the second occurs when \( p_{b+1} > p_a \).

The contribution of the leaves that fall in the first case is

\[
S_1 = \sum_{c < b + 1 \leq a^*} S_{1b},
\]

with

\[
S_{1b} = - \sum_{\alpha \sqrt{3} \leq m \leq \alpha \sqrt{3}} \mu(m) \phi \left( \frac{x}{mp_{b+1}}, b \right).
\]

From these formulas it follows that the computation of \( S_1 \) takes at most \( O(\alpha \sqrt{3} / x^{1/2}) \) steps (without counting the sieve work).

The contribution of the leaves that fall in the second case is

\[
S_2 = \sum_{a^* < b + 1 < a} S_{2b},
\]

with

\[
S_{2b} = \sum_{b + 1 < d \leq a} \phi \left( \frac{x}{p_{b+1}p_d}, b \right),
\]

because \( p_{\text{min}}(m) > p_{b+1} \) coupled with \( m \leq \alpha \sqrt{3} < p_{b+1}^2 \) forces \( m \) to be a prime number. The number of terms in \( S_2 \) is exactly \( \left( \frac{a - a^*}{2} \right) \). When \( a > 1 \) part of the computation of \( S_2 \) can be performed without resorting to the full machinery of appendix A [4]. Indeed, when

\[
\max \left( \frac{x}{p_{b+1}^2}, p_{b+1} \right) < p_d \leq \alpha \sqrt{3}
\]

the contribution of the special leaf is given by

\[
\phi \left( \frac{x}{p_{b+1}p_d}, b \right) = 1
\]

(this is a consequence of the fact that \( \phi(y, b) = 1 \) when \( 1 \leq y < p_{b+1} \)), and when

\[
\max \left( \frac{x}{p_{b+1}^2}, p_{b+1} \right) < p_d \leq \min \left( \frac{x}{p_{b+1}^2}, \alpha \sqrt{3} \right)
\]

the contribution of the special leaf is given by

\[
\phi \left( \frac{x}{p_{b+1}p_d}, b \right) = \pi \left( \frac{x}{p_{b+1}p_d} \right) - b + 1
\]

(this is a consequence of the fact that \( \phi(y, b) = \pi(y) - b + 1 \) when \( p_{b+1} \leq y < p_{b+1}^2 \)). In order to use (12) efficiently it is necessary to precompute a table of values of \( \pi(y) \). To avoid using an excessive amount of space, this table will only store values of \( \pi(y) \) for \( y \leq \alpha \sqrt{3} \). (For the present purpose it is convenient to exclude the case \( y = \alpha \sqrt{3} \).)

The condition \( y < \alpha \sqrt{3} \) forces \( p_d \) to be larger than \( z/p_{b+1} \). Since \( z/p_{b+1} > x/p_{b+1}^2 \) when \( p_{b+1}^2 > \alpha \sqrt{3} \), it follows that (12) can be used for the values of \( d \) that satisfy

\[
\max \left( \frac{x}{p_{b+1}^2}, p_{b+1} \right) < p_d \leq \min \left( \frac{x}{p_{b+1}^2}, \alpha \sqrt{3} \right).
\]

### Appendix A

#### Table III

<table>
<thead>
<tr>
<th>Computation of ( \mu(n)p_{\text{min}}(n) ) for ( 1 \leq n \leq N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void init_mu(int *mu, int N)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>int i,j;</td>
</tr>
<tr>
<td>for(i = 1;i &lt; N;i++)</td>
</tr>
<tr>
<td>mu[i] = 1;</td>
</tr>
<tr>
<td>for(j = 2;j &lt; N;j++)</td>
</tr>
<tr>
<td>if(mu[j] == 1)</td>
</tr>
<tr>
<td>for(i = j;i &lt; N;i += j)</td>
</tr>
<tr>
<td>mu[i] = (mu[i] == 1) ? -mu[i]: mu[i];</td>
</tr>
<tr>
<td>for(j = 2;j * j &lt; N;j++)</td>
</tr>
<tr>
<td>if(mu[j] == 1)</td>
</tr>
<tr>
<td>for(i = j * j;i &lt; N;i += j * j)</td>
</tr>
<tr>
<td>mu[i] = 0;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

#### Notes

- Use efficient data structures for storing and updating the sieve values.
- Optimize the computation of \( \phi(x, a) \) by utilizing the properties of the sieve function.
- Use precomputed tables for \( \pi(y) \) to save computation time.
- Ensure efficient space usage by dynamically adjusting the size of the data structures as needed.
It is convenient to give names to the different kinds of leaves that contribute to \( S_2 \); those that satisfy (10) will be called \textit{trivial leaves}, those that satisfy (13) will be called \textit{easy leaves}, and the rest will be called \textit{hard leaves}. (This terminology is similar to the one used in [4].)

According to (11), the contribution of the trivial leaves is their number. From (10), it follows that the number of trivial leaves, for each value of \( b \), is given by \( a + 1 - t_b \), where

\[
t_b = \begin{cases} 
  b + 2, & \text{if } \frac{x}{p_{b+1}} \leq p_{b+1}, \\
  \pi \left( \frac{x}{p_{b+1}} \right) + 1, & \text{if } p_{b+1} < \frac{x}{p_{b+1}} < \alpha \sqrt{x}, \\
  a + 1, & \text{if } \alpha \sqrt{x} \leq p_{b+1}.
\end{cases}
\]

Note that there are no trivial leaves when \( p_{b+1} \leq \sqrt{z} \), and that all leaves are trivial when \( p_{b+1} \geq \sqrt{x} \). With the help of the table of values of \( \pi(y) \) mentioned previously, the contribution of the trivial leaves corresponding to a given value of \( b \) can be computed in constant time. The contribution of all trivial leaves can then be computed in \( O(\alpha x^{1/3} \log^{-3} x) \) steps.

The contribution of the easy leaves can be split in two parts [6], according to whether \( p_d > \sqrt{x}/p_{b+1} \) or whether \( p_d < \sqrt{x}/p_{b+1} \). In the first case the value of (12) has a tendency to be the same for consecutive values of \( b \), while this does not happen in the second case. For this reason, the leaves that fall in the first case will be called \textit{clustered easy leaves}, and those that fall in the second case will be called \textit{sparse easy leaves}. Instead of computing the contribution of each clustered easy leaf individually, it is faster to determine the number of leaves for which (12) takes a given value, say \( l \), and then compute their joint contribution in a single step. Note that \( l \leq a - b + 1 \); this result follows easily from the fact that (13) can only be satisfied when \( \sqrt{x}/\alpha^2 < p_{b+1} < \sqrt{z} \), coupled with \( p_d > \sqrt{x}/p_{b+1} \). The values of \( d \) for which (12) is equal to \( l \) satisfy

\[
p_{b+1} - 1 \leq \frac{x}{p_{b+1} p_d} < p_{b+1},
\]

which is equivalent to

\[
\frac{x}{p_{b+1} p_{b+1} t} < p_d \leq \frac{x}{p_{b+1} p_{b+1} t - 1}.
\]

(Note that \( p_{a+1} \) may be needed to compute the lower bound.) Keeping in mind that (13) and \( p_d > \sqrt{x}/p_{b+1} \) must also be satisfied, it follows that the number of valid values of \( d \) can be computed in constant time, again with the help of the table of values of \( \pi(y) \) mentioned previously. (In practice the distinction between clustered and sparse easy leaves does not need to be as rigid as presented here, which simplifies somewhat the algorithm implementation; see subsection II-D for details.)

\section*{C. Choice of the value of \( \alpha \)}

In order to compute \( \pi(x) \) using the method outlined above, the first thing that must be decided is the value of \( \alpha \), which must satisfy \( 1 \leq \alpha \leq \sqrt{x} \). To deal with the non-linear effects of the processor’s data caches, this is best done by experimenting with an actual program, adjusting the value of \( \alpha \) until the execution time reaches a minimum for a given test value of \( x \). (In the author’s program, changes of \( \pm 25\% \) around the optimal value of \( \alpha \) did not increase the execution time by more than 3%). The value of \( \alpha \) found in this way will be close to optimal for values of \( x \) close to the test value. After doing this for several test values it will be possible to use some kind of curve fit to obtain a good value of \( \alpha \) for a general \( x \). The following asymptotic study shows that the best \( \alpha \) should grow like \( \log^{3} x \).

To obtain accurate estimates of the number of easy and hard leaves \( \pi(x) \) will be approximated by \( \ln(x) \). This will be achieved by replacing summations in which \( p_{b+1} \) or \( p_d \) belongs to a given interval by integrals, over the same interval, with \( p_{b+1} \) or \( p_d \) replaced respectively by \( u \) or \( v \), and using \( du = \frac{u}{\log u} \) or \( dv = \frac{v}{\log v} \) as the measure of integration. The following definitions will be useful: \( w_0 = \sqrt{\alpha} \sqrt{x}, w_1 = \sqrt{x}/\alpha \), \( w_2 = \sqrt{z} \), \( w_3 = \sqrt{\sqrt{x}} \), and \( w_4 = \alpha \sqrt{x} \). Moreover, it will be implicitly assumed that \( \alpha = O(x^{1/3}) \).

The shapes of the trivial, easy, and hard regions of the \( u \)-\( v \) plane can be determined from the following facts:

- \( w_0 < u < w_4 \), see (8);
- \( u < v < w_4 \), see (9);
- the transition between trivial and easy leaves occurs when \( x/(p_{b+1} p_d) \approx p_{b+1} \), i.e., when \( v = x/u^2 \);
- the transition between the two kinds of easy leaves occurs when \( p_d \approx \sqrt{x}/p_{b+1} \), i.e., when \( v = \sqrt{x}/u \);
- the transition between easy and hard leaves occurs when \( x/(p_{b+1} p_d) \approx \alpha \sqrt{x} \), i.e., when \( v = z/u \).

The shape of these regions is illustrated in figure 2 for the case \( x = 10^{12} \) and \( \alpha = 2 \). As mentioned above, the integral of \( 1/(\log u \log v) \) over one of these regions is a good estimate of the number of leaves it contains.

Let \( W_0 \) be the amount of work required to compute the contribution of all sparse easy leaves, \( N_s \) the number of
sparse easy leaves, and $A_e$ the area of the sparse easy leaves region. According to subsection II-B.3 the contribution of each sparse easy leaf can be computed in a constant number of steps. Thus $W_s$ is proportional to $N_s$. But

$$N_s \approx \int_{w_1}^{w_2} \frac{du}{\log u} \int_{\sqrt{x/u}}^{\sqrt{x/\log v}} \frac{dv}{\log v} + \int_{w_2}^{w_3} \frac{du}{\log u} \int_{\sqrt{x/u}}^{\sqrt{x/\log v}} \frac{dv}{\log v} \,.$$  

Since $w_1 \leq u \leq w_2$ and $w_2 \leq v \leq w_4$, it follows that

$$\frac{A_e}{\log w_2 \log w_4} \leq N_s \leq \frac{A_e}{\log w_1 \log w_2}.$$  

Because $A_e = O(x^{2/3})$ and $\alpha = O(x)$, it follows that $W_s = O(x^{2/3} \log^{-2} x)$.

According to subsection II-B.3, the work required to compute the contribution of the clustered easy leaves to the value of $S_{2b}$ is proportional to the number of values of $l$. It follows that the work required to compute the contribution of all clustered easy leaves, denoted by $W_c$, can be approximated, up to a multiplicative constant, by

$$\int_{w_1}^{w_2} \frac{du}{\log u} \left( \pi \left( \frac{x}{u \sqrt{x/u}} \right) - \pi \left( \frac{x}{uw_4} \right) \right) + \int_{w_2}^{w_3} \frac{du}{\log u} \left( \pi \left( \frac{x}{u \sqrt{x/u}} \right) - \pi \left( \frac{x}{u(x/\sqrt{x})} \right) \right) \approx N_s.$$  

Hence, $W_c$ grows at the same rate as $W_s$. (This is not a coincidence. Gourdon [7] found a way to merge the computation of the contributions of the two types of easy leaves.)

Let $W_h$ be the amount of work required to compute the contribution of all hard leaves, $N_h$ the number of hard leaves, and $A_h$ the area of the hard leaves region. According to appendix A, the work required to compute the contribution of each hard leaf takes $O(\log z)$ steps. Hence, $W_h$ will be proportional to $N_h \log z$. Since $A_h = O(z \log \alpha)$ it follows that $N_h = O(z \log^{-2} x \log \alpha)$, and that

$$W_h = O\left( \frac{\log \alpha}{\alpha} x^{2/3} \log x \right).$$  

The only other significant part of the computation is to sieve the interval $[1, z]$ using the method described in appendix A, which requires $O(\log z)$ steps (and not $O(z \log \log z)$ steps as reported in [6]). The entire amount of work required to compute $\pi(x)$ is then

$$W = O\left( \frac{\log \alpha}{\alpha} x^{2/3} \log x + \frac{x^{2/3}}{\log x} + \frac{x^{2/3}}{\alpha} \log x \right).$$  

The choice $\alpha = \beta \log^3 x$ balances the sieve work with the work required to evaluate the contribution of the easy leaves, giving a total work of $O(x^{2/3} \log^{-2} x)$ steps. The constant $\beta$ depends on the actual implementation of the algorithm; it should be determined empirically.

**D. Subdivision of the interval $[1, z]$**

Dealing with the whole interval $[1, z]$ at once is impractical for large values of $x$. It is thus usually necessary to subdivide it. This will be done at the integers $z_k$, which must satisfy the conditions $1 = z_0 < z_1 < \cdots < z_K = [z]$, giving rise to the intervals $B_k = [z_{k-1}, z_k]$, $k = 1, \ldots, K$. It is obvious that $[1, z]$ and $\bigcup_{k=1}^{K} B_k$ contain the same integers.

The intervals $B_k$ must be processed sequentially, starting with $B_1$. To compute $\phi(y, b)$ for $y \in B_k$ and for $c < b < a$, it is necessary to remove the multiples of each prime up to $p_b$ from this interval. In order to be able to count quickly the number of surviving integers, $z_k - z_{k-1}$ should be a power of $2$. The value of $\phi(y, b)$ will then be the value of the sum of the appropriate counters (see appendix A) plus the value of $\phi(z_{k-1} - 1, b)$.

When the length of each interval is $O(\alpha \sqrt{x})$ or less, the method to compute $\pi(x)$ described in this paper will use $O(x^{1/3} \log^{-3} x)$ words of storage. This goal can be achieved using $O(x^{1/3} \log^{-6} x)$ or more intervals of equal length. (In an actual program the interval length should be adapted dynamically, in order to make the program as fast as possible.) Since the amount of work spent in overheads while processing an interval is proportional to the number of primes used in the sieve (see below), to keep the total amount of work at $O(x^{2/3} \log^{-2} x)$ no more than $O(x^{1/3} \log^{1/2} x)$ intervals can be used.

There are three tasks that must be performed while the interval $B_k$ is being processed, namely, update of the value of $S_{2b}$ for $c < b + 1 \leq a^*$, update of the contribution of the hard leaves to the value of $S_{2b}$ for $a^* < b + 1 < a$, and update of the value of $\phi_B(x, a)$. To accomplish the first two it is necessary to determine the value of $\phi(y, b)$ for some $y \in B_k$ and $c < b + 1 < a$. To accomplish the third is it necessary to evaluate $\pi(y)$ for some $y \in B_k$.

For each $b$, the values of $m$ (or of $p_{b+1}$) which need to be considered when the interval $B_k$ is being processed are those for which $x/(mp_{b+1}) \in B_k$ and $\left| \phi(m) \right|_{\text{min}} > p_{b+1}$. Since in a special leaf (7) must also be enforced, it follows that

$$\max\left( \frac{\alpha \sqrt{x}}{p_{b+1}}, \frac{x}{p_{b+1}z_k} \right) < m \leq \min\left( \frac{\alpha \sqrt{x}}{p_{b+1}z_{k-1}}, \frac{x}{p_{b+1}z_k} \right).$$  

(14)

For this condition to be satisfied it is necessary that

$$\max\left( p_{b+1}, \frac{x}{z_k} \right) < p_{b+1} < \min\left( \alpha \sqrt{x}, \frac{x}{z_{k-1}} \right).$$  

It is possible to infer from this result that the range of active values of $p_{b+1}$ is larger when $z_k$ is small than when $z_k$ is large (see figure 3). Thus, the work required to process constant-length intervals is highly skewed; intervals close to $x^{1/3}$ require much more computational effort than intervals close to $x^{2/3}$.

There is no need to resort to (14) to identify the special leaves that contribute to $S_{1b}$ or $S_{2b}$ when the interval $B_k$ is being processed. The computation of $S_{1b}$, with $c < b < a^*$, can be done using the following algorithm, which uses the variable $m_{1b}$ to keep track of the largest value of $m$ not yet taken in consideration.

**Algorithm 1.** [Computation of $S_{1b}$ for a given $b$]

**Step 1.** Set $m_{1b} = \lfloor \alpha \sqrt{x} \rfloor$, $S_{1b} = 0$, and $k = 1$.

**Step 2.** Remove the multiples of $p_1, \ldots, p_b$ from $B_k$.  


Step 6. Add vertical logarithmic scale, the range of values of \( p \) done using the following algorithm, which uses the variable \( v \) if easier to understand, \( z \) (descending lines), and the prime sieve limit \( \sqrt{x} \) (ascending line), for the case \( x = 2^{48}, \alpha = 1, c = 0, \) and \( z_{k-1} = z_k = 2^{16}. \) (To make the figure vertical logarithmic scale, the range of values of \( p_{k+1} \) is larger when \( z_k \) is small than when it is large.

Step 3. If \( m_{1b}p_{b+1} \leq \alpha \sqrt{x} \) then terminate the algorithm.

Step 4. Set \( y = x / (m_{1b}p_{b+1}). \) If \( y \geq z_k \) then increment \( k \) and go to step 2.

Step 5. If \( p_{\min}(m_{1b}) \geq p_{b+1} \) then subtract \( \mu(m_{1b})\phi(y, b) \) from \( S_{1b}. \) Decrease \( m_{1b} \) and go to step 3.

(Because the computation of \( S_1 \) requires a comparatively small amount of work, it is not necessary to use the method described in section 5 of [4] to find the special leaves.)

The computation of \( S_{2b}, \) with \( a^* \leq b < a - 1, \) can be done using the following algorithm, which uses the variable \( d_{2b} \) to keep track of the largest value of \( d \) not yet taken in consideration. (When \( t = 0 \) the algorithm is evaluating clustered easy leaves, when \( t = 1 \) it is evaluating sparse easy leaves, and when \( t = 2 \) it is evaluating hard leaves.)

Algorithm 2. [Computation of \( S_{2b} \) for a given \( b \)]

Step 1. Set \( d_{2b} = t_b - 1, S_{2b} = a - d_{2b}, k = 1, \) and \( t = 0. \)

Step 2. If \( d_{2b} = b + 1 \) then terminate the algorithm. Otherwise, set \( y = x / (p_{b+1}p_{2b+1}). \) and go to step 3.

Step 3. If \( y \geq \alpha \sqrt{x} \) then set \( t = 2 \) and go to step 9. Otherwise, set \( l = \pi(y) - b + 1 \) and \( d' = \pi(x / (p_{b+1}p_{2b+1})). \) Note that \( d' + 1 \) is the smallest value of \( d \) for which (12) is equal to \( l. \)

Step 4. If \( p_{2b+1} \leq \sqrt{x / p_{2b+1}} \) or if \( d' \leq b \) then set \( t = 1 \) and go to step 6. Otherwise, add \( l(d_{2b} - d') \) to \( S_{2b}, \) set \( (\text{afterwards}) \ d_{2b} = d', \) and go to step 2.

Step 5. If \( y \geq \alpha \sqrt{x} \) then set \( t = 2 \) and go to step 9. Otherwise, set \( l = \pi(y) - b + 1. \)

Step 6. Add \( l \) to \( S_{2b}, \) decrement \( d_{2b}, \) and go to step 2.

Step 7. If \( y \geq z_k \) then increment \( k \) and go to step 9.

Step 8. Add \( \phi(y, b) \) to \( S_{2b}, \) decrement \( d_{2b}, \) and go to step 2.

Step 9. Remove the multiples of \( p_{1}, \ldots, p_{b} \) from \( B_{k}. \) Go to step 7.

The update of the value of \( \phi_2(x, a) \) can be done using the following algorithm, which uses the variable \( u \) to keep track of the largest value of \( p_u \) not yet taken in consideration, the variable \( v \) to count the number of primes up to \( \sqrt{x}, \) and the variable \( w \) to keep track of the first integer represented in an auxiliary sieve.

Algorithm 3. [Computation of \( \phi_2(x, a) \)]

Step 1. Set \( \phi_2 = a(a - 1)/2, u = \lfloor \sqrt{x} \rfloor, v = a, w = u + 1, \) and \( k = 1. \)

Step 2. Remove the multiples of \( p_{1}, \ldots, p_{a} \) from \( B_{k}. \)

Step 3. If \( u \leq \alpha \sqrt{x} \) then subtract \( v(u - 1)/2 \) from \( \phi_2 \) and terminate the algorithm.

Step 4. If \( u < w \) then set \( w = \max(2, u - \lfloor \alpha \sqrt{x} \rfloor) \) and sieve completely the interval \([w, u + 1] \).

Step 5. Using the sieve of step 3, test if \( u \) is prime. If not then decrement \( u \) and go to step 3.

Step 6. Set \( y = x / u. \) If \( y \geq z_k \) then increase \( k \) and go to step 2.

Step 7. Add \( \phi(y, a) - a + 1 \) to \( \phi_2, \) increment \( v, \) decrement \( u \) and go to step 3.

The overheads introduced in these algorithms by the subdivision of the interval \([1, z] \) are exactly the overheads introduced by a segmented sieve of Eratosthenes.

E. Outline of the algorithm used to compute \( \pi(x) \)

The following algorithm presents a high-level overview of the entire algorithm used to compute \( \pi(x). \)

Algorithm 4. [Computation of \( \pi(x) \)]

Step 1. Choose a value for \( \alpha \) using the guide-lines presented at the beginning of section II-C.

Step 2. Make a list of the primes up to \( \alpha \sqrt{x}, \) and compute the values of \( \mu(n)\min(n) \) and of \( \pi(n) \) for all odd \( n \) up to that limit. The values of \( \mu(n)\min(n) \) can be computed efficiently using a simple adaptation (to deal only with odd numbers) of the code presented in table III.

The primes can then be identified easily using the test \( \mu(n)\min(n) = -n. \) Compute \( a, a^*, \) and \( p_{b+1}. \)

Step 3. Set \( c = 7 \) (or any other reasonably small value).

Compute and store in a table the values of \( \phi(n, c) \) for \( 0 \leq n < p_{1} \cdots p_{c}. \) Next, compute \( S_{2b} \) and \( S_{1b}, \) using (5) to evaluate \( \phi(\cdot, c). \) At this point the table values of \( \phi(n, c) \) is not needed any more. However, as described in appendix A, the counter initialization can be improved when the values of \( f(n, k_c) = \phi(n, c) - \phi(n - 1, c) \) are known. To take advantage of this possibility, transform the table of values of \( \phi(n, c) \) into a table of values of \( f(n, k_c). \) (In practice, only odd values of \( n \) are used.)

Step 4. For each \( b = c + 1, \ldots, a^* - 1, \) perform step 1 of algorithm 1. For each \( b = a^*, \ldots, a - 2, \) run algorithm 2 until either it terminates or step 9 is reached. Perform step 1 of algorithm 3. For each \( b = c + 1, \ldots, a, \) set \( \phi(z_a - 1, b) = 0. \) Set \( k = 1. \)

Step 5. Initialize the sieve counters using code similar to that of table VII, and using the \( f(n, k_c) \) values (repeated periodically) computed previously. Discard the multiples of \( p_{b+1}, \ldots, p_{a} \) from \( B_{k} \) using the machinery of appendix A. In between, use the the appropriate parts of algorithms 1 and 2 to update the values of \( S_{1b} \) and \( S_{2b}, \) computing \( \phi(\cdot, b) \) as described in appendix A, and replace the value of \( \phi(z_k - 1, b) \) by that of \( \phi(z_k - 1, b). \)

Once all the primes have been processed, use the appropriate parts of algorithm 3 to update the value of \( \phi_2(x, a). \)
If one of the algorithms did not terminate, increment $k$ and repeat this step.

Step 6. Compute $\phi(x, a) = S_0 + \sum_{b=1}^{a-1} S_{1b} + \sum_{b=a}^{\infty} S_{2b}$. Compute $\pi(x) = \phi(x, a) + a - 1 - \phi_2(x, a)$ and terminate the algorithm.

In practice, there is no need to store $S_0$, the several $S_{1b}$ and $S_{2b}$, and $\phi_2(x, a)$ in separate variables (see the last step of the previous algorithm).

III. SOME VALUES OF $\pi(x)$

The algorithm described in the previous section can be adapted to compute simultaneously $\pi(x)$ for several values of $x$. Since the sieve work can be shared among the different computations, this way of doing things speeds up the preparation of extensive tables of values of $\pi(x)$. The author of this article implemented the algorithm in this way.

The first values of $\pi(x)$, and of $\log(x) - \pi(x)$, for $x$ a power of 10, or a power of 2, are presented in tables IV and V, respectively. The logarithmic-integral function was computed using the equality $\log(x) = \pi(x)$, where

$$\log(x) = \int_{2}^{x} \frac{dt}{\log(t)} = \frac{x}{\log(x)} - \sum_{k=2}^{\infty} \frac{x^{1/k}}{k}$$

($\gamma = 0.5772156649 \ldots$ is Euler’s constant). More extensive tables of values of $\pi(x)$ can be found in the web page http://www.ieeta.pt/~tos/primes.html.

It is obvious from tables IV and V that $\log(x)$ is a good approximation of $\pi(x)$. The last column of these tables suggest that $\log(x) - \pi(x)$ for $2 \leq x \leq 10^{22}$. Note, however, that Littlewood proved in the first quarter of the XX century that $\log(x) - \pi(x)$ changes sign infinitely often. It is known that the least $x$ for which $\log(x) > \pi(x)$ is smaller that $1.4 \cdot 10^{310}$ [12].

<table>
<thead>
<tr>
<th>$x$</th>
<th>$\pi(x)$</th>
<th>$\log(x) - \pi(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^1$</td>
<td>4</td>
<td>2.165 ...</td>
</tr>
<tr>
<td>$10^2$</td>
<td>25</td>
<td>5.126 ...</td>
</tr>
<tr>
<td>$10^3$</td>
<td>168</td>
<td>9.609 ...</td>
</tr>
<tr>
<td>$10^4$</td>
<td>1229</td>
<td>17.137 ...</td>
</tr>
<tr>
<td>$10^5$</td>
<td>9592</td>
<td>37.809 ...</td>
</tr>
<tr>
<td>$10^6$</td>
<td>78498</td>
<td>129.549 ...</td>
</tr>
<tr>
<td>$10^7$</td>
<td>64637</td>
<td>339.405 ...</td>
</tr>
<tr>
<td>$10^8$</td>
<td>576415</td>
<td>754.375 ...</td>
</tr>
<tr>
<td>$10^9$</td>
<td>5084753</td>
<td>1700.957 ...</td>
</tr>
<tr>
<td>$10^{10}$</td>
<td>45502511</td>
<td>3430.584 ...</td>
</tr>
<tr>
<td>$10^{11}$</td>
<td>418054813</td>
<td>11587.621 ...</td>
</tr>
<tr>
<td>$10^{12}$</td>
<td>376709120</td>
<td>38262.804 ...</td>
</tr>
<tr>
<td>$10^{13}$</td>
<td>34665536389</td>
<td>108079.050 ...</td>
</tr>
<tr>
<td>$10^{14}$</td>
<td>3206941750802</td>
<td>314889.953 ...</td>
</tr>
<tr>
<td>$10^{15}$</td>
<td>2964557042966</td>
<td>1052618.581 ...</td>
</tr>
<tr>
<td>$10^{16}$</td>
<td>27923834103925</td>
<td>3214613.792 ...</td>
</tr>
<tr>
<td>$10^{17}$</td>
<td>2623557157654233</td>
<td>7956588.778 ...</td>
</tr>
<tr>
<td>$10^{18}$</td>
<td>24739954287708600</td>
<td>21949555.022 ...</td>
</tr>
<tr>
<td>$10^{19}$</td>
<td>234037667276344600</td>
<td>99877775.223 ...</td>
</tr>
<tr>
<td>$10^{20}$</td>
<td>2220819602560918840</td>
<td>222744634.548 ...</td>
</tr>
<tr>
<td>$10^{21}$</td>
<td>2117262948601871928</td>
<td>5973942543.334 ...</td>
</tr>
<tr>
<td>$10^{22}$</td>
<td>201467286689315906290</td>
<td>1932355208.150 ...</td>
</tr>
</tbody>
</table>
I. EFFICIENT SIEVE IMPLEMENTATION

The algorithm described in section II requires frequent evaluations of the function $\phi(x, a)$. The binary tree data structure of [4] can be used to do this efficiently. The author of this paper was able to eliminate the redundancy present in this data structure, with the result that its space requirements were reduced by 33% and its update speed was increased by a factor of almost 2. Later, he found out that a similar data structure, which does not require a power of two length but treats its first data element in a different way, was proposed by Fenwick [9] to perform updates and queries of the cumulative frequency tables used in arithmetic coders [11].

Suppose that the interval $[B, B + 2^L]$ is to be sieved, and that it is necessary to evaluate $\phi(x, a)$ for “random” values of $x \in [B, B + 2^L]$ and non-decreasing values of $a$. Let $f(n, k)$, for $n = 0, \ldots, 2^L - 1$ and $k = 0, 1, \ldots$, represent the status of the integer $B + n$ after $k$ elementary sieve operations. Each elementary sieve operation amounts to mark a previously unmarked multiple of some prime: $f(n, k)$ will be equal to one if $B + n$ remains unmarked after $k$ such operations and will be equal to zero otherwise. After the multiples of the primes up to $p_n$ have been marked, a task requiring $k_n$ elementary sieve operations, it will be possible to compute the value of

$$\phi(x, a) = \phi(B - 1, a) + \sum_{n=0}^{x-B} f(n, k_n) \quad (15)$$

for any $x \in [B, B + 2^L]$. The direct use of this formula is obviously very inefficient (average and worst amount of work proportional to $2^L$), although the elementary sieve operations will be very efficient (constant amount of work).

It is possible to make both the elementary sieve operations and the evaluation of $\phi(x, a)$ very efficient using only $2^L$ counters. Each counter accumulates the values of $f(n, k)$ in a certain range, as depicted in figure 4 for $L = 3$. In practice, since it is also necessary to ascertain if a given integer has already been marked, it is possible to use one of the otherwise unused bits of its corresponding counter to store this information; the most significant bit (sign bit for signed integer data types) is a particularly good choice.

The initialization of the counters to match the situation in which no number has been marked, i.e., $f(n, 0) = 1$, is very simple: the number of consecutive least significant bits equal to one of each counter index determines the base 2 logarithm of the initial value of its corresponding counter (see C code and example in table VI). The entire initialization is done in linear time.

Since the values of $\phi(x, a)$ can be computed with (5) when $a$ is small (i.e., $a = c$), it is a waste of time to initialize the counters using the code of table VI and then mark the multiples of the primes $p \leq p_c$ which belong to the interval $[B, B + 2^L]$. It is much faster to initialize the counters directly from the values of $f(n, k_c)$, which is a function with period $p_1 \cdots p_c$. This can be done using only $2^L - 1$ additions (see C code and example in table VII).

Each elementary sieve operation requires the update of at most $L$ counters. Working in base 2, it is very easy to find the indices of the counters that need to be decremented: the first corresponds to the integer that was marked, the rest can be obtained by replacing each zero bit of the index by one, starting from the least significant bit, until all $L$ bits become equal to one (see C code and example in table VIII).

Assuming that the integers to be marked follow an uniform distribution, it can be shown that the average number of counters that need to be updated is $1 + L/2$.

After an elementary sieve operation it is possible to compute the value of $\sum_{n=0}^{x-B} f(n, k)$ by summing the values of at most $L$ counters. Working again in base 2, the indices plus one of the counters that need to be summed can be found easily: the first is equal to $1 + [x - B]$, and the others are obtained by successively changing each bit that is equal to one to zero, starting from the least significant bit, until zero is obtained (see C code and example in table IX). Assuming that the numbers $x$ follow an uniform distribution, it can be shown that the computation of $\phi(x, a)$ requires an average number of $L/2 + 2^{-L}$ summations.

![Fig. 4 - Fractal-like organization of the counters for the case $L = 3$. Each counter keeps track of the sum of values of $f(n, k)$ in its area on influence (rectangles). For example, counter 5 contains the value $f(4, k) + f(5, k).$](image-url)
When there are many values of $\phi(x, a)$ to be computed for the same value of $a$ it may be advantageous to replace the binary tree structure of the counters by a linear list structure, from which the values of $\phi(x, a) - \phi(B - 1, a)$ can be read directly. This conversion, called flattening the counters, can be done using $2^L - 1 - L$ summations (see C code and example in table X). Reversing this operation (deflattening) is equally simple.

**REFERENCES**


