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 fragments.

***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

$$\lim_{x\to\infty} \frac{\pi(x)}{\int_2^x \frac{du}{\log u}} = \lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\log x}} = 1.$$

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. In 2001, Gourdon [7] found a way to save constant factors both in the time and in the space complexity of the Deléglise-Rivat method.

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, and double checked by the author of this paper in 2006.

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.

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 $a \geqslant 0$ and a real number $x \geqslant 1$, 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.,

$$\phi(x,a) = \sum_{n=1}^{\lfloor x \rfloor} [p_{\min}(n) > p_a].$$

[1]One unit of time – one step – corresponds to one single word elementary arithmetic operation (addition, subtraction, multiplication, or division), or to one decision (branch), or to one single word data movement (load or store). One unit of storage space corresponds to one word. It is assumed that $+x$ and $-x$ can be stored without error in a single word.

Post-publication annotations (such as this one) appear in blue, either in figures or at the bottom of the page.

TABLE I
NOTATION USED

notation	short description
$[C]$	equal to 1 if condition C is true and equal to 0 otherwise
$[x,y[$	interval of the real line, closed at x and open at y
$\lfloor x \rfloor$	greatest integer $\leqslant x$
$\lceil x \rceil$	smallest integer $\geqslant x$
$\binom{a}{2}$	equal to $a(a-1)/2$
$\gcd(a,b)$	greatest common divisor of the two integers a and b
$\varphi(n)$	**(Euler's totient function)** number of integers between 0 and n that are co-prime to n
p_k	the k-th prime number, i.e., $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and so on; by convention $p_0 = 1$
$p_{\min}(n)$	smallest prime factor of n; by convention $p_{\min}(1) = +\infty$
$p_{\max}(n)$	largest prime factor of n; by convention $p_{\max}(1) = 1$
$\omega(n)$	number of distinct prime factors of n; by convention, $\omega(1) = 0$
$\Omega(n)$	number of prime factors (counting repetitions) of n; by convention, $\Omega(1) = 0$
$\mu(n)$	**(Möbius function)** equal to $(-1)^{\omega(n)}\big[\omega(n) = \Omega(n)\big]$
$\mathcal{O}(f)$	$g(x) = \mathcal{O}\big(f(x)\big)$ if there exists a positive constant C and a x_0 such that $\lvert g(x)\rvert \leqslant Cf(x)$ for all $x > x_0$
$\mathrm{li}(x)$	**(logarithmic integral)** principal value of $\int_0^x \frac{du}{\log u}$

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.,

$$\phi_k(x,a) = \sum_{n=1}^{\lfloor x \rfloor} [p_{\min}(n) > p_a]\,[\Omega(n) = k].$$

Since $\phi_k(x,a) = 0$ when $x < p_{a+1}^k$, i.e., when $a \geqslant \pi(\sqrt[k]{x})$, the fundamental theorem of arithmetic implies that

$$\phi(x,a) = \sum_{k=0}^{\left\lfloor \frac{\log x}{\log p_{a+1}} \right\rfloor} \phi_k(x,a). \tag{1}$$

For $x \geqslant 1$ it is obvious that

$$\phi_0(x,a) = 1.$$

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

$$\phi_1(x,a) = \pi(x) - a.$$

Consequently, for $x \geqslant 1$ and $a \leqslant \pi(x)$, it follows from (1) that

$$\pi(x) = \phi(x,a) + a - 1 - \sum_{k=2}^{\left\lfloor \frac{\log x}{\log p_{a+1}} \right\rfloor} \phi_k(x,a). \tag{2}$$

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 $a = \pi(\alpha\sqrt[3]{x})$, $1 \leqslant \alpha \leqslant \sqrt[6]{x}$, with α growing slower that any power of x, i.e., $\alpha = \mathcal{O}(x^\epsilon)$. 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 $\phi_k(x,a)$ vanish), and that $a = \pi(\sqrt[4]{x})$ was used by Lehmer in [3] (in this case $\phi_2(x,a)$ and $\phi_3(x,a)$ must be computed).

Note that $\phi_k(x,a) > 0$ only when $p_{a+1}^k \leqslant x$, i.e., when $k \leqslant \frac{\log x}{\log p_{a+1}}$.

$$\phi_2(x,a) = \binom{\pi(\alpha\sqrt[3]{x})}{2} - \binom{\pi(\sqrt{x})}{2} + \sum_{\alpha\sqrt[3]{x} < p_b \leqslant \sqrt{x}} \pi\Big(\frac{x}{p_b}\Big)$$

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

Let $a = \pi(\alpha\sqrt[3]{x})$, with $1 \leqslant \alpha \leqslant \sqrt[6]{x}$. Thus, $a \leqslant \pi(\sqrt{x})$. When $a = \pi(\sqrt{x})$ it can be verified that $\phi_2(x,a)$ vanishes. When $\pi(\sqrt[3]{x}) \leqslant a < \pi(\sqrt{x})$ it can be verified that

$$\phi_2(x,a) = \sum_{b=a+1}^{\pi(x)} \sum_{c=b}^{\pi(x)} [p_b p_c \leqslant x] = \sum_{b=a+1}^{\pi(\sqrt{x})} \sum_{c=b}^{\pi(x/p_b)} 1$$

does not vanish. Since the number of terms in the inner summation is $\pi(x/p_b) - (b-1)$, it follows that

$$\phi_2(x,a) = \binom{a}{2} - \binom{\pi(\sqrt{x})}{2} + \sum_{b=a+1}^{\pi(\sqrt{x})} \pi\Big(\frac{x}{p_b}\Big). \tag{3}$$

By convention [10, exercise 2.1], $\sum_{j=i}^{i-1} f(j) = 0$; hence, (3) also gives the correct result for $a = \pi(\sqrt{x})$.

Let

$$z = \frac{x^{2/3}}{\alpha}. \tag{4}$$

Since $p_a \leqslant \alpha\sqrt[3]{x} < p_{a+1}$ it follows, for $a < b \leqslant \pi(\sqrt{x})$, that

$$\sqrt{x} \leqslant \frac{x}{p_b} \leqslant \frac{x}{p_{a+1}} < \frac{x}{\alpha\sqrt[3]{x}} = z.$$

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

$$\pi\Big(\frac{x}{p_b}\Big) = \phi\Big(\frac{x}{p_b}, a\Big) + a - 1.$$

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 $\mathcal{O}(x^{1/2}\log^{-1} x)$ values of $\phi\big(\frac{x}{p_b}, a\big)$ can be computed in $\mathcal{O}(\log z)$ 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 $\mathcal{O}(x^{1/2}\log\log x)$ steps (because it is necessary to generate all primes up to $x^{1/2}$ using a segmented Eratosthenes sieve).

$\left|\frac{x}{p} - \frac{x}{q}\right| \approx |p-q|$ when p and q are near $\sqrt{x}$, and $\left|\frac{x}{p} - \frac{x}{q}\right| \approx \frac{\sqrt[3]{x}}{\alpha^2}|p-q|$ when p and q are near $\alpha\sqrt[3]{x}$. Hence, more values of $\pi(\cdot)$ are needed near $\sqrt{x}$ than near z.
In practice, it turns out to be faster to compute $\phi_2(x,a)$ and $\phi(x,a)$ separately.

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

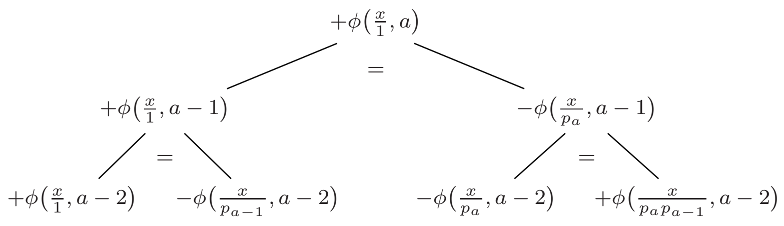
Because $\gcd(n, p_1\cdots p_c)$ is, when c is fixed, a periodic function with period $p_1\cdots p_c$, the computation of $\phi(x,c)$ can be done using the formula

$$\phi(x,c) = \left\lfloor \frac{x}{p_1\cdots p_c} \right\rfloor \phi(p_1\cdots p_c, c) + \phi\big(\lfloor x \rfloor \bmod p_1\cdots p_c, c\big), \tag{5}$$

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 $0 \leqslant n < p_1\cdots p_c$. Obviously, this way of computing $\phi(x,c)$ can only be used when c is very small (say, when $c \leqslant 7$).



$+\phi(\frac{x}{1},a)$
$=$
$+\phi(\frac{x}{1},a-1)$
$-\phi(\frac{x}{p_a},a-1)$
$=$
$=$
$+\phi(\frac{x}{1},a-2)$
$-\phi(\frac{x}{p_{a-1}},a-2)$
$-\phi(\frac{x}{p_a},a-2)$
$+\phi(\frac{x}{p_a p_{a-1}},a-2)$

Fig. 1 - The first two levels of the computational tree of $\phi(x,a)$.

For $a > 0$ the computation of $\phi(x,a)$ can also be done recursively, using the formula

$$\begin{aligned}\phi(x,a) &= \sum_{n=1}^{\lfloor x\rfloor}\big([p_{\min}(n) \geqslant p_a] - [p_{\min}(n) = p_a]\big)\\ &= \phi(x,a-1) - \sum_{n=1}^{\lfloor x/p_a\rfloor}[p_{\min}(n) \geqslant p_a]\\ &= \phi(x,a-1) - \phi(x/p_a,a-1). \qquad (6)\end{aligned}$$

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 $\mu(n)\phi(\frac{x}{n},a-k)$, with $p_{\max}(n) \leqslant p_a$ and $p_{\min}(n) > p_{a-k}$. Since $\phi(x,0) = \lfloor x\rfloor$, growing the tree as much as possible yields

$$\phi(x,a) = \sum_{\substack{1\leqslant n\leqslant x\\ p_{\max}(n)\leqslant p_a}} \mu(n)\left\lfloor\frac{x}{n}\right\rfloor.$$

It is not necessary to apply (6) when $x < p_{a+1}$, in which case $\phi(x,a) = 1$ if $x \geqslant 1$ 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 $\mathcal{O}(x\log^{-1}x)$ steps and uses $\mathcal{O}(\sqrt{x}\log^{-1}x)$ 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 $a = \pi(\alpha\sqrt[3]{x})$, with $\alpha \geqslant 1$ carefully chosen, and use the following rule to continue to apply (6): split a node labeled $\pm\phi(y,b)$ if $b > c$ and $y \geqslant z$, with z given by (4). This rule is equivalent to the following termination rule: do not split a node labeled $\pm\phi(y,b) = \mu(n)\phi(\frac{x}{n},b)$ if either

(i) $b = c$ and $y \geqslant z$, i.e., $b = c$ and $n \leqslant \alpha\sqrt[3]{x}$, or

(ii) $y < z$, i.e., $n > \alpha\sqrt[3]{x}$ (if this happens then $b \geqslant c$).

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 $n \leqslant \alpha\sqrt[3]{x}$, $\mu(n) \neq 0$, 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

$$S_0 = \sum_{\substack{1\leqslant n\leqslant \alpha\sqrt[3]{x}\\ p_{\min}(n)>p_c}} \mu(n)\phi\Big(\frac{x}{n},c\Big).$$

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 $1 \leqslant n \leqslant \alpha\sqrt[3]{x}$, 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 $\big|\mu(n)p_{\min}(n)\big| > p_c$. For each such n it is possible to compute $\phi(\frac{x}{n},c)$ quickly using (5).

It is obvious that the number of ordinary leaves cannot be larger than $\alpha\sqrt[3]{x}$. Thus, the work required to compute S_0 takes no more than $\mathcal{O}(\alpha x^{1/3})$ steps, to which must be added the $\mathcal{O}(\alpha x^{1/3}\log\log x)$ 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 $n > \alpha\sqrt[3]{x}$, $\mu(n) \neq 0$, $p_{\min}(n) > p_c$ and $p_{\max}(n) \leqslant \alpha\sqrt[3]{x}$ must be satisfied.

TABLE II
COMPUTATION OF $\pi(x)$ USING LEGENDRE'S FORMULA

```
static int sum,np,*p;

static void init_p(int x)
{
  // compute np=pi(sqrt(x)) and initialize
  // the array p[0..np] with 2,3,5,7,...,0
  ...
}

static void phi(int x,int a,int sign)
{
loop:
  if(a == 0)
    sum += sign * x;
  else if(x < p[a])
    sum += sign;
  else
  {
    --a;
    phi(x / p[a],a,-sign);
    goto loop;
  }
}

int pi(int x)
{
  init_p(x);
  sum = np - 1;
  phi(x,np,1);
  return sum;
}
```

For the nodes at level k, n has at most k prime factors.
Do not split a node if $b = c$ and $y \geqslant z$, or if $b = c$ and $y < z$, or if $b > c$ and $y < z$. Since $y = x/n$, $y < z$ implies that $n > \alpha\sqrt[3]{x}$, and $y \geqslant z$ implies that $n \leqslant \alpha\sqrt[3]{x}$.
It is possible to use a larger c without using a prohibitive amout of memory if the required values of $\phi(\cdot,c)$ are computed using (6).

In the computation of S_0, for very large values of x, use a segmented sieve to compute $\mu(n)$, $p_{\min}(n)$, and $\pi(\alpha\sqrt[3]{x})$.

TABLE III
COMPUTATION OF $\mu(n)p_{\min}(n)$ FOR $1 \leqslant n \leqslant N$

```
void init_mu(int *mu,int N)
{
  int i,j;

  for(i = 1;i <= N;i++)
    mu[i] = 1;
  for(j = 2;j <= N;j++)
    if(mu[j] == 1)
      for(i = j;i <= N;i += j)
        mu[i] = (mu[i] == 1) ? -j : -mu[i];
  for(j = 2;j * j <= N;j++)
    if(mu[j] == -j)
      for(i = j * j;i <= N;i += j * j)
        mu[i] = 0;
}
```

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 $\mu(m)\phi\big(\frac{x}{m}, b+1\big)$, with $b+1 > c$ and $m \leqslant \alpha\sqrt[3]{x}$, then the special leaf has the value $-\mu(m)\phi\big(\frac{x}{mp_{b+1}}, b\big)$, with $mp_{b+1} > \alpha\sqrt[3]{x}$; by construction, $p_{\max}(mp_{b+1}) \leqslant \alpha\sqrt[3]{x}$ is automatically ensured. The contribution of the special leaves to the value of $\phi(x,a)$ is then given by

$$S = -\sum_{c<b+1<a}\ \sum_{\substack{m \leqslant \alpha\sqrt[3]{x} < mp_{b+1}\\ p_{\min}(m) > p_{b+1}}} \mu(m)\phi\Big(\frac{x}{mp_{b+1}}, b\Big).$$

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

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

$$\max\Big(\frac{\alpha\sqrt[3]{x}}{p_{b+1}}, p_{b+1}\Big) < m \leqslant \alpha\sqrt[3]{x} \tag{7}$$

and $\big|\mu(m)p_{\min}(m)\big| > p_{b+1}$. Equipped with this way of identifying all special leaves, the computation of S 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 $p_1, \ldots, p_c$ 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 $\mathcal{O}(\log z)$ steps, (not counting the work required to perform the sieve operations).

For very large values of x, the computation of the contribution of the special leaves with $b = c$ should be done while S_0 is also being computed, using the values of $\mu(m)$ and $p_{\min}(m)$ being generated by the segmented sieve (and also using the table of values of $\phi(k,c)$ for $0 \leqslant k < p_1 \cdots p_c$).

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 $p_{b+1}^2 \leqslant \alpha\sqrt[3]{x}$ or whether $p_{b+1}^2 > \alpha\sqrt[3]{x}$. The first occurs when $p_{b+1} \leqslant p_{a^*}$, where $a^* = \pi(\sqrt{\alpha}\sqrt[6]{x})$, 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

$$S_1 = \sum_{c<b+1\leqslant a^*} S_{1b},$$

with

$$S_{1b} = -\sum_{\substack{\frac{\alpha\sqrt[3]{x}}{p_{b+1}} < m \leqslant \alpha\sqrt[3]{x}\\ p_{\min}(m) > p_{b+1}}} \mu(m)\phi\Big(\frac{x}{mp_{b+1}}, b\Big).$$

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

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

$$S_2 = \sum_{a^*<b+1<a} S_{2b}, \tag{8}$$

with

$$S_{2b} = \sum_{b+1<d\leqslant a} \phi\Big(\frac{x}{p_{b+1}p_d}, b\Big), \tag{9}$$

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

$$\max\Big(\frac{x}{p_{b+1}^2}, p_{b+1}\Big) < p_d \leqslant \alpha\sqrt[3]{x} \tag{10}$$

the contribution of the special leaf is given by

$$\phi\Big(\frac{x}{p_{b+1}p_d}, b\Big) = 1 \tag{11}$$

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

$$\max\Big(\frac{x}{p_{b+1}^3}, p_{b+1}\Big) < p_d \leqslant \min\Big(\frac{x}{p_{b+1}^2}, \alpha\sqrt[3]{x}\Big)$$

the contribution of the special leaf is given by

$$\phi\Big(\frac{x}{p_{b+1}p_d}, b\Big) = \pi\Big(\frac{x}{p_{b+1}p_d}\Big) - b + 1 \tag{12}$$

(this is a consequence of the fact that $\phi(y,b) = \pi(y) - b + 1$ when $p_{b+1} \leqslant y < p_{b+1}^2$). 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 $y \leqslant \alpha\sqrt[3]{x}$. (For the present purpose it is convenient to exclude the case $y = \alpha\sqrt[3]{x}$.) The condition $y < \alpha\sqrt[3]{x}$ forces p_d to be larger than z/p_{b+1}. Since $z/p_{b+1} > x/p_{b+1}^3$ when $p_{b+1}^2 > \alpha\sqrt[3]{x}$, it follows that (12) can be used for the values of d that satisfy

$$\max\Big(\frac{z}{p_{b+1}}, p_{b+1}\Big) < p_d \leqslant \min\Big(\frac{x}{p_{b+1}^2}, \alpha\sqrt[3]{x}\Big). \tag{13}$$

It is convenient to give names to the different kinds of leaves that contribute to S_2; those that satisfy (10) will be called **trivial leaves**, those that satisfy (13) will be called **easy leaves**, and the rest will be called **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

$$t_b = \begin{cases} b+2, & \text{if } \frac{x}{p_{b+1}^2} \leqslant p_{b+1}, \\ \pi\left(\frac{x}{p_{b+1}^2}\right)+1, & \text{if } p_{b+1} < \frac{x}{p_{b+1}^2} < \alpha\sqrt[3]{x}, \\ a+1, & \text{if } \alpha\sqrt[3]{x} \leqslant \frac{x}{p_{b+1}^2}. \end{cases}$$

Note that there are no trivial leaves when $p_{b+1} \leqslant \sqrt{z}$, and that all leaves are trivial when $p_{b+1} \geqslant \sqrt[3]{x}$. 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 $\mathcal{O}(\alpha x^{1/3} \log^{-1} x)$ 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 $p_d \leqslant \sqrt{x/p_{b+1}}$. In the first case the value of (12) has a tendency to be the same for consecutive values of d, while this does not happen in the second case. For this reason, the leaves that fall in the first case will be called **clustered easy leaves**, and those that fall in the second case will be called **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 $l \leqslant a - b + 1$; this result follows easily from the fact that (13) can only be satisfied when $\sqrt[3]{x}/\alpha^2 < p_{b+1} < \sqrt[3]{x}$, coupled with $p_d > \sqrt{x/p_{b+1}}$. The values of d for which (12) is equal to l satisfy

$$p_{b+l-1} \leqslant \frac{x}{p_{b+1}p_d} < p_{b+l},$$

which is equivalent to

$$\frac{x}{p_{b+1}p_{b+l}} < p_d \leqslant \frac{x}{p_{b+1}p_{b+l-1}}.$$

(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.)

C. *Choice of the value of* α

In order to compute $\pi(x)$ using the method outlined above, the first thing that must be decided is the value of α, which must satisfy $1 \leqslant \alpha \leqslant \sqrt[6]{x}$. 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

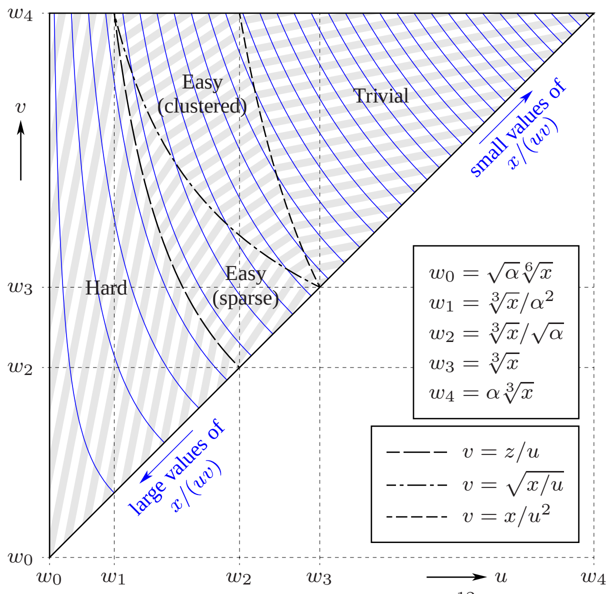


Fig. 2 - Classification of the special leaves for $x = 10^{12}$ and $\alpha = 2$. The u coordinate represents values of p_{b+1}, and the v coordinate represents values of p_d. When $w_1 < w_0$, i.e., when $\alpha > \sqrt[15]{x}$, the shape of the easy and hard regions is slightly different; since, by assumption, $\alpha = \mathcal{O}(x^\epsilon)$, there is no need to discuss this case. For practical values of x, $\alpha > \sqrt[15]{x}$. Curves of constant uv are presented in blue.

of α 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 α did not increase the execution time by more than 3%.) The value of α 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 α for a general x. The following asymptotic study shows that the best α should grow like $\log^3 x$.

To obtain accurate estimates of the number of easy and hard leaves $\pi(x)$ will be approximated by $\mathrm{li}(x)$. 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 $\frac{du}{\log u}$ or $\frac{dv}{\log v}$ as the measure of integration. The following definitions will be useful: $w_0 = \sqrt{\alpha}\sqrt[6]{x}$, $w_1 = \sqrt[3]{x}/\alpha^2$, $w_2 = \sqrt{z}$, $w_3 = \sqrt[3]{x}$, and $w_4 = \alpha\sqrt[3]{x}$. Moreover, it will be implicitly assumed that $\alpha = \mathcal{O}(x^\epsilon)$.

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);
- $u < v \leqslant w_4$, 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 $x/(p_{b+1}p_d) \approx \alpha\sqrt[3]{x}$, 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_s 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_s 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

$$N_\mathrm{s} \approx \int_{w_1}^{w_2} \frac{du}{\log u} \int_{zu^{-1}}^{\sqrt{x/u}} \frac{dv}{\log v} + \int_{w_2}^{w_3} \frac{du}{\log u} \int_{u}^{\sqrt{x/u}} \frac{dv}{\log v}.$$

Since $w_1 \leqslant u \leqslant w_3$ and $w_2 \leqslant v \leqslant w_4$, it follows that

$$\frac{A_\mathrm{s}}{\log w_3 \log w_4} \leqslant N_\mathrm{s} \leqslant \frac{A_\mathrm{s}}{\log w_1 \log w_2}.$$

Because $A_\mathrm{s} = \mathcal{O}(x^{2/3})$ and $\alpha = \mathcal{O}(x^\epsilon)$, it follows that

$$W_\mathrm{s} = \mathcal{O}(x^{2/3} \log^{-2} x).$$

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

$$\int_{w_1}^{w_2} \frac{du}{\log u} \left(\pi\Big(\frac{x}{u\sqrt{x/u}} \Big) - \pi\Big(\frac{x}{uw_4} \Big) \right) + \int_{w_2}^{w_3} \frac{du}{\log u} \left(\pi\Big(\frac{x}{u\sqrt{x/u}} \Big) - \pi\Big(\frac{x}{u(x/u^2)} \Big) \right) \approx N_\mathrm{s}.$$

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 $\mathcal{O}(\log z)$ steps. Hence, W_h will be proportional to $N_\mathrm{h} \log z$. Since $A_\mathrm{h} = \mathcal{O}(z \log \alpha)$ it follows that $N_\mathrm{h} = \mathcal{O}(z \log^{-2} x \log \alpha)$, and that

$$W_\mathrm{h} = \mathcal{O}\left(\frac{\log \alpha}{\alpha} \frac{x^{2/3}}{\log x} \right).$$

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

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

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 $\mathcal{O}(x^{2/3} \log^{-2} x)$ steps. The constant β 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 $1 = z_0 < z_1 < \cdots < z_K = \lceil z \rceil$, 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 $c \leqslant b < a$ 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 $\mathcal{O}(\alpha \sqrt[3]{x})$ or less, the method to compute $\pi(x)$ described in this paper will use $\mathcal{O}(x^{1/3} \log^3 x)$ words of storage. This goal can be achieved using $\mathcal{O}(x^{1/3} \log^{-6} x)$ 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 $\mathcal{O}(x^{2/3} \log^{-2} x)$ no more than $\mathcal{O}(x^{1/3} \log^{1/2} x)$ 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_{1b} for $c < b + 1 \leqslant a^*$, 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 $\phi_2(x, a)$. 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 is necessary to evaluate $\pi(y)$ for some $y \in B_k$.

For each b, the values of m (or of p_d) 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 $|\mu(m) p_{\min}(m)| > p_{b+1}$. Since in a special leaf (7) must also be enforced, it follows that

$$\max\left(\frac{\alpha \sqrt[3]{x}}{p_{b+1}}, p_{b+1}, \frac{x}{p_{b+1} z_k} \right) < m \leqslant \min\left(\alpha \sqrt[3]{x}, \frac{x}{p_{b+1} z_{k-1}} \right). \tag{14}$$

For this condition to be satisfiable it is necessary that

$$\max\left(p_c, \frac{z}{z_k} \right) < p_{b+1} < \min\left(\alpha \sqrt[3]{x}, \sqrt{\frac{x}{z_{k-1}}} \right).$$

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 $c \leqslant b < a^*$, 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 $m_{1b} = \lfloor \alpha \sqrt[3]{x} \rfloor$, $S_{1b} = 0$, and $k = 1$.
Step 2. Remove the multiples of $p_1, \ldots, p_b$ from $B_k$.
```

Since $z_k \leqslant z$, it is clear that $\frac{\alpha \sqrt[3]{x}}{p_{b+1}} \leqslant \frac{x}{p_{b+1} z_k}$ is always true.

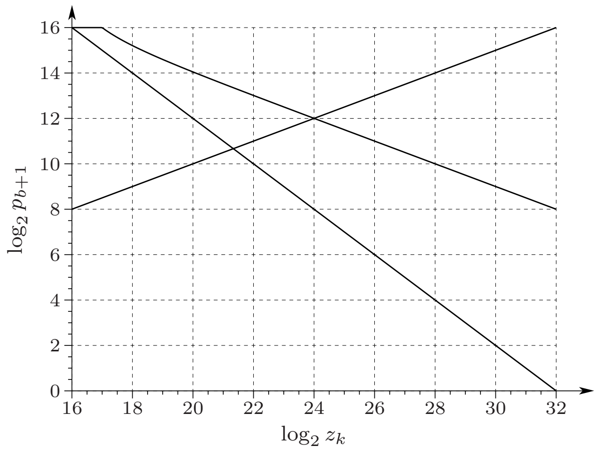


Fig. 3 - Lower and upper bounds of p_{b+1} while sieving the interval $[1, z[$ (descending lines), and the prime sieve limit $\sqrt{z_k}$ (ascending line), for the case $x = 2^{48}$, $\alpha = 1$, $c = 0$, and $z_{k-1} = z_k - 2^{16}$. (To make the figure easier to understand, z_k is treated as a continuous variable.) Due to the vertical logarithmic scale, the range of values of p_{b+1} is larger when z_k is small than when it is large.

```
Step 3. If $m_{1b}p_{b+1} \leqslant \alpha\sqrt[3]{x}$ then terminate the algorithm.
Step 4. Set $y = x/(m_{1b}p_{b+1})$. If $y \geqslant z_k$ then increment $k$ and go to step 2.
Step 5. If $p_{\min}(m_{1b}) > p_{b+1}$ then subtract $\mu(m_{1b})\phi(y, b)$ from $S_{1b}$. Decrement $m_{1b}$ 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 $a^* \leqslant b < a - 1$, 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 $y = x/(p_{b+1}p_{d_{2b}})$ and go to step $3 + 2t$.
Step 3. If $y \geqslant \alpha\sqrt[3]{x}$ then set $t = 2$ and go to step 9. Otherwise, set $l = \pi(y) - b + 1$ and $d' = \pi\bigl(x/(p_{b+1}p_{b+l})\bigr)$. Note that $d' + 1$ is the smallest value of $d$ for which (12) is equal to $l$.
Step 4. If $p_{d'+1} \leqslant \sqrt{x/p_{b+1}}$ or if $d' \leqslant b$ then set $t = 1$ and go to step 6. Otherwise, add $l(d_{2b} - d')$ to $S_{2b}$, set (afterwards) $d_{2b} = d'$, and go to step 2.
Step 5. If $y \geqslant \alpha\sqrt[3]{x}$ 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 $y \geqslant z_k$ 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_b 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 $u \leqslant \alpha\sqrt[3]{x}$ then subtract $v(v-1)/2$ from $\phi_2$ and terminate the algorithm.
Step 4. If $u < w$ then set $w = \max\bigl(2, u - \lfloor\alpha\sqrt[3]{x}\rfloor\bigr)$ 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 $y \geqslant z_k$ 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 $\alpha\sqrt[3]{x}$, and compute the values of $\mu(n)p_{\min}(n)$ and of $\pi(n)$ for all odd $n$ up to that limit. The values of $\mu(n)p_{\min}(n)$ 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 $\mu(n)p_{\min}(n) = -n$. Compute $a$, $a^*$, and $p_{a+1}$.
Step 3. Set $c = 7$ (or any other reasonably small value). Compute and store in a table the values of $\phi(n, c)$ for $0 \leqslant n < p_1 \cdots p_c$. Next, compute $S_0$ and $S_{1c}$, using (5) to evaluate $\phi(\cdot, c)$. At this point the table of 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 $\phi(z_0 - 1, b) = 0$. 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 $p_{c+1}, \ldots, p_a$ 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 $\phi(z_{k-1} - 1, b)$ 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 $\phi(x,a) = S_0 + \sum_{b=c}^{a^*-1} S_{1b} + \sum_{b=a^*}^{a-2} S_{2b}$. 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 $\mathrm{li}(x) - \pi(x)$, 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 $\mathrm{li}(x) = \mathrm{Ei}(\log x)$, where

$$\mathrm{Ei}(x) = \int_{-\infty}^{x} e^t \, \frac{dt}{t} = \gamma + \log x + \sum_{k=1}^{\infty} \frac{1}{k} \frac{x^k}{k!}$$

($\gamma = 0.5772156649\ldots$ 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 $\mathrm{li}(x)$ is a good approximation of $\pi(x)$. The last column of these tables suggest that $\mathrm{li}(x) > \pi(x)$ for $2 \leqslant x \leqslant 10^{22}$. Note, however, that Littlewood proved in the first quarter of the XX century that $\mathrm{li}(x) - \pi(x)$ changes sign infinitely often. It is known that the least x for which $\pi(x) > \mathrm{li}(x)$ is smaller that $1.4 \cdot 10^{316}$ [12].

TABLE IV
VALUES OF $\pi(x)$ AND OF $\mathrm{li}(x) - \pi(x)$ FOR POWERS OF TEN

x	$\pi(x)$	$\mathrm{li}(x) - \pi(x)$
10^1	4	2.165 ...
10^2	25	5.126 ...
10^3	168	9.609 ...
10^4	1229	17.137 ...
10^5	9592	37.809 ...
10^6	78498	129.549 ...
10^7	6 64579	339.405 ...
10^8	57 61455	754.375 ...
10^9	508 47534	1700.957 ...
10^{10}	4550 52511	3103.586 ...
10^{11}	41180 54813	11587.621 ...
10^{12}	3 76079 12018	38262.804 ...
10^{13}	34 60655 36839	1 08971.050 ...
10^{14}	320 49417 50802	3 14889.953 ...
10^{15}	2984 45704 22669	10 52618.581 ...
10^{16}	27923 83410 33925	32 14631.792 ...
10^{17}	2 62355 71576 54233	79 56588.778 ...
10^{18}	24 73995 42877 40860	219 49555.022 ...
10^{19}	234 05766 72763 44607	998 77775.223 ...
10^{20}	2220 81960 25609 18840	2227 44643.548 ...
10^{21}	21127 26948 60187 31928	5973 94254.333 ...
10^{22}	2 01467 28668 93159 06290	19323 55208.150 ...

$\pi(10^{23}) = 19\,25320\,39160\,68039\,68923$

TABLE V
VALUES OF $\pi(x)$ AND OF $\mathrm{li}(x) - \pi(x)$ FOR POWERS OF TWO

x	$\pi(x)$	$\mathrm{li}(x) - \pi(x)$
2^1	1	0.045 ...
2^2	2	0.967 ...
2^3	4	1.253 ...
2^4	6	2.519 ...
2^5	11	2.605 ...
2^6	18	3.934 ...
2^7	31	5.042 ...
2^8	54	6.513 ...
2^9	97	6.721 ...
2^{10}	172	9.078 ...
2^{11}	309	12.114 ...
2^{12}	564	12.922 ...
2^{13}	1028	19.751 ...
2^{14}	1900	19.888 ...
2^{15}	3512	32.244 ...
2^{16}	6542	41.986 ...
2^{17}	12251	45.067 ...
2^{18}	23000	69.193 ...
2^{19}	43390	63.811 ...
2^{20}	82025	112.527 ...
2^{21}	1 55611	128.964 ...
2^{22}	2 95947	166.838 ...
2^{23}	5 64163	248.512 ...
2^{24}	10 77871	350.700 ...
2^{25}	20 63689	295.678 ...
2^{26}	39 57809	540.548 ...
2^{27}	76 03553	830.150 ...
2^{28}	146 30843	934.673 ...
2^{29}	281 92750	1555.428 ...
2^{30}	544 00028	1447.618 ...
2^{31}	1050 97565	2665.676 ...
2^{32}	2032 80221	3860.999 ...
2^{33}	3936 15806	3586.424 ...
2^{34}	7629 39111	5334.930 ...
2^{35}	14802 06279	10663.828 ...
2^{36}	28743 98515	13544.223 ...
2^{37}	55865 02348	15994.979 ...
2^{38}	1 08662 66172	22830.503 ...
2^{39}	2 11519 07950	25740.119 ...
2^{40}	4 12030 88796	41644.933 ...
2^{41}	8 03165 71436	69688.200 ...
2^{42}	15 66610 34233	59035.208 ...
2^{43}	30 57617 13237	1 14792.833 ...
2^{44}	59 71163 81732	1 32860.781 ...
2^{45}	116 67467 86182	2 62854.070 ...
2^{46}	228 09987 53949	1 66928.337 ...
2^{47}	446 16329 79717	2 19714.955 ...
2^{48}	873 11888 63470	6 63697.853 ...
2^{49}	1709 44325 76778	9 17028.068 ...
2^{50}	3348 33796 03407	10 68422.765 ...
2^{51}	6561 28999 15304	18 09320.253 ...
2^{52}	12862 55036 10475	14 56904.811 ...
2^{53}	25225 27041 48404	18 67813.455 ...
2^{54}	49489 02049 04784	54 25756.994 ...
2^{55}	97126 99452 45201	58 55761.639 ...
2^{56}	1 90687 93810 28850	67 38674.599 ...
2^{57}	3 74501 11847 13964	133 83939.057 ...
2^{58}	7 35740 02678 43990	149 17783.914 ...
2^{59}	14 45879 28953 01660	172 04097.042 ...
2^{60}	28 42309 44969 53330	224 77599.971 ...
2^{61}	55 89048 40450 84135	455 08690.025 ...
2^{62}	109 93280 75854 69973	394 12395.209 ...
2^{63}	216 28961 18534 39384	875 75308.033 ...
2^{64}	425 65628 40352 17743	805 00871.808 ...
2^{65}	837 90314 54666 07212	1567 46489.268 ...
2^{66}	1649 81970 04647 85589	2035 82959.080 ...
2^{67}	3249 25438 70525 57215	1768 00774.267 ...
2^{68}	6400 77159 75449 37806	4414 08683.778 ...
2^{69}	12611 86461 87603 52880	4663 70512.942 ...
2^{70}	24855 45536 33626 85793	9353 50265.789 ...
2^{71}	48995 57160 01294 58363	8613 64388.234 ...
2^{72}	96601 07519 50751 86855	13159 80554.314 ...

$\pi(2^{73}) = 1\,90499\,82340\,13279\,05601$
$\pi(2^{74}) = 3\,75744\,16493\,76996\,09596$
$\pi(2^{75}) = 7\,41263\,52114\,07401\,13483$

APPENDIX

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_a have been marked, a task requiring k_a elementary sieve operations, it will be possible to compute the value of

$$\phi(x,a) = \phi(B-1,a) + \sum_{n=0}^{\lfloor x-B \rfloor} f(n,k_a) \qquad (15)$$

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 if small (i.e., $a = c$), it is a waste of time to initialize the

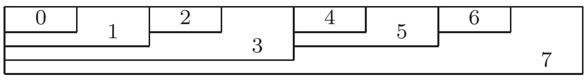


Fig. 4 - Fractal-like organization of the counters for the case $L = 3$. Each counter keeps track of the sum of values of $f(n,k)$ in its area on influence (rectangles). For example, counter 5 contains the value $f(4,k) + f(5,k)$.

TABLE VI
COUNTER INITIALIZATION

```
void cnt_init(int *cnt,int L)
{
  int i;

  for(i = 0;i < (1 << L);i++)
    cnt[i] = (i + 1) & ~i;
}
```

Indices (in decimal and in base 2) and the base 2 initial values of the counters for the case $L = 3$.

0	1	2	3	4	5	6	7
000_2	001_2	010_2	011_2	100_2	101_2	110_2	111_2
0001_2	0010_2	0001_2	0100_2	0001_2	0010_2	0001_2	1000_2

TABLE VII
COUNTER INITIALIZATION FROM $f(n,k)$

```
void cnt_finit(int *f,int *cnt,int L)
{
  int i,j,k;

  for(i = 0;i < (1 << L);i++)
  {
    cnt[j = i] = f[i];
    for(k = (i + 1) & ~i;k >>= 1;j &= j - 1)
      cnt[i] += cnt[j - 1];
  }
}
```

Indices plus one (in base 2) of the counters that must be summed to initialize the tree data structure from $f(n,k)$ for the case $L = 3$. These summations proceed from the left to the right, always using the most recent value of each counter.

0	1	2	3	4	5	6	7
0001_2	0010_2	0011_2	0100_2	0101_2	0110_2	0111_2	1000_2
	0001_2		0011_2		0101_2		0111_2
			0010_2				0110_2
							0100_2

counters using the code of table VI and then mark the multiples of the primes $p \leqslant p_c$ 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 must be decremented: the first corresponds to the integer that was marked, and 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 $\sum_{n=0}^{\lfloor x-B \rfloor} f(n,k)$ 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 $1 + \lfloor x-B \rfloor$, 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.

TABLE VIII
COUNTER UPDATE

```
void cnt_update(int pos,int *cnt,int L)
{
  do
  {
    cnt[pos]--;
    pos |= pos + 1;
  }
  while(pos < (1 << L));
}
```

Indices (in base 2) of the counters that must be decremented then an integer is marked for the case $L = 3$.

0	1	2	3	4	5	6	7
000_2	001_2	010_2	011_2	100_2	101_2	110_2	111_2
001_2	011_2	011_2	111_2	101_2	111_2	111_2	
011_2	111_2	111_2		111_2			
111_2							

TABLE IX
COUNTER QUERY

```
int cnt_query(int pos,int *cnt)
{
  int sum;

  sum = cnt[pos++];
  while(pos &= pos - 1)
    sum += cnt[pos - 1];
  return sum;
}
```

Indices plus one (in base 2) of the counters that must be summed to compute $\sum_{n=0}^{\lfloor x-b \rfloor} f(n,k)$ for the case $L = 3$.

0	1	2	3	4	5	6	7
0001_2	0010_2	0011_2	0100_2	0101_2	0110_2	0111_2	1000_2
		0010_2		0100_2	0100_2	0110_2	
						0100_2	

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.

TABLE X
COUNTER FLATTENING AND DEFLATTENING

```
void cnt_flatten(int *cnt,int L)
{
  int i,j;

  for(i = 2;i < (1 << L);i++)
  {
    j = i & (i + 1);
    if(j)
      cnt[i] += cnt[j - 1];
  }
}
```

Indices plus one (in base 2) of the counters that must be summed to flatten the tree data structure for the case $L = 3$. These summations proceed from the left to the right, always using the most recent value of the counters.

0	1	2	3	4	5	6	7
0001_2	0010_2	0011_2	0100_2	0101_2	0110_2	0111_2	1000_2
		0010_2		0100_2	0100_2	0110_2	

```
void cnt_deflatten(int *cnt,int L)
{
  int i,j;

  for(i = (1 << L) - 1;i > 1;--i)
  {
    j = i & (i + 1);
    if(j)
      cnt[i] -= cnt[j - 1];
  }
}
```

REFERENCES

[1] L. E. Dickson, *History of the Theory of Numbers*, vol. I: Divisibility and Primality, AMS Chelsea Publishing, Providence, Rhode Island, USA, 1992, Published originally by the Carnegie Institute of Washington (publication number 256) in 1919.

[2] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, vol. 126 of *Progress in Mathematics*, Birkhäuser, Boston, second edition, 1994.

[3] D. H. Lehmer, "On the exact number of primes less than a given limit", *Illinois Journal of Mathematics*, vol. 3, pp. 381–388, 1959.

[4] J. C. Lagarias, V. S. Miller, and A. M. Odlyzko, "Computing $\pi(x)$: The Meissel-Lehmer method", *Mathematics of Computation*, vol. 44, no. 170, pp. 537–560, Apr. 1985.

[5] R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*, Springer, Berlin / New York, 2002 (second printing).

[6] M. Deléglise and J. Rivat, "Computing $\pi(x)$: the Meissel, Lehmer, Lagarias, Miller, Odlyzko method", *Mathematics of Computation*, vol. 65, no. 213, pp. 235–245, Jan. 1996.

[7] X. Gourdon, "Computation of $\pi(x)$: Improvements to the Meissel, Lehmer, Lagarias, Miller, Odlyzko, Deléglise and Rivat method", Available from `http://numbers.computation.free.fr/Constants/Primes/Pix/piNalgorithm.ps`, 2001.

[8] J. C. Lagarias and A. M. Odlyzko, "Computing $\pi(x)$: an analytic method", *Journal of Algorithms*, vol. 8, pp. 173–191, 1987.

[9] P. M. Fenwick, "A new data structure for cumulative frequency tables", *Software – Practice and Experience*, vol. 24, no. 3, pp. 327–336, Mar. 1994, Corrections in Vol. 24, No. 7, p. 677, July 1994.

[10] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[11] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited", *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256–294, July 1998.

[12] C. Bays and R. H. Hudson, "A new bound for the smallest x with $\pi(x) > \mathrm{li}(x)$", *Mathematics of Computation*, vol. 69, no. 231, pp. 1285–1296, 2000.