WIKIPEDIA

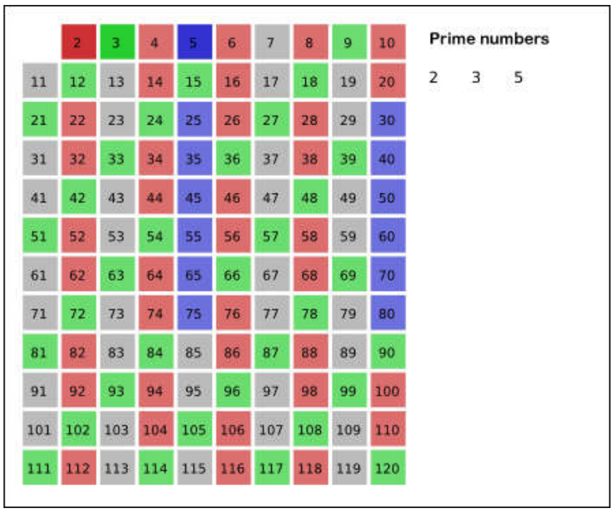
Sieve of Eratosthenes

In mathematics, the **sieve of Eratosthenes** is a simple, ancient algorithm for finding all prime numbers up to any given limit.

It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the first prime number, 2. The multiples of a given prime are generated as a sequence of numbers starting from that prime, with constant difference between them that is equal to that prime.[1] This is the sieve's key distinction from using trial division to sequentially test each candidate number for divisibility by each prime.[2]

The earliest known reference to the sieve (Ancient Greek: κόσκινον Ἐρατοσθένους, *kóskinon Eratosthénous*) is in Nicomachus of Gerasa's *Introduction to Arithmetic*,[3] which describes it and attributes it to Eratosthenes of Cyrene, a Greek mathematician.

One of a number of prime number sieves, it is one of the most efficient ways to find all of the smaller primes. It may be used to find primes in arithmetic progressions.[4]



Sieve of Eratosthenes: algorithm steps for primes below 121 (including optimization of starting from prime's square).

Contents

Overview

A prime number is a natural number that has exactly two distinct natural number divisors: 1 and itself.

> *Sift the Two's and Sift the Three's,*
> *The Sieve of Eratosthenes.*
> *When the multiples sublime,*
> *The numbers that remain are Prime.*
> Anonymous[5]

To find all the prime numbers less than or equal to a given integer n by Eratosthenes' method:

1. Create a list of consecutive integers from 2 through n: $(2, 3, 4, ..., n)$.
2. Initially, let p equal 2, the smallest prime number.
3. Enumerate the multiples of p by counting to n from $2p$ in increments of p, and mark them in the list (these will be $2p$, $3p$, $4p$, ...; the p itself should not be marked).
4. Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.
5. When the algorithm terminates, the numbers remaining not marked in the list are all the primes below n.

The main idea here is that every value given to p will be prime, because if it were composite it would be marked as a multiple of some other, smaller prime. Note that some of the numbers may be marked more than once (e.g., 15 will be marked both for 3 and 5).

As a refinement, it is sufficient to mark the numbers in step 3 starting from p^2, as all the smaller multiples of p will have already been marked at that point. This means that the algorithm is allowed to terminate in step 4 when p^2 is greater than n.[1]

Another refinement is to initially list odd numbers only, $(3, 5, ..., n)$, and count in increments of $2p$ from p^2 in step 3, thus marking only odd multiples of p. This actually appears in the original algorithm.[1] This can be generalized with wheel factorization, forming the initial list only from numbers coprime with the first few primes and not just from odds (i.e., numbers coprime with 2), and counting in the correspondingly adjusted increments so that only such multiples of p are generated that are coprime with those small primes, in the first place.[6]

Example

To find all the prime numbers less than or equal to 30, proceed as follows.

First generate a list of integers from 2 to 30:

```
2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

The first number in the list is 2; cross out every 2nd number in the list after 2 by counting up from 2 in increments of 2 (these will be all the multiples of 2 in the list):

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~

The next number in the list after 2 is 3; cross out every 3rd number in the list after 3 by counting up from 3 in increments of 3 (these will be all the multiples of 3 in the list):

```
2  3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ 25 ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~
```

The next number not yet crossed out in the list after 3 is 5; cross out every 5th number in the list after 5 by counting up from 5 in increments of 5 (i.e. all the multiples of 5):

```
2  3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~
```

The next number not yet crossed out in the list after 5 is 7; the next step would be to cross out every 7th number in the list after 7, but they are all already crossed out at this point, as these numbers (14, 21, 28) are also multiples of smaller primes because 7 × 7 is greater than 30. The numbers not crossed out at this point in the list are all the prime numbers below 30:

```
2  3     5     7           11    13          17    19          23                29
```

Algorithm and variants

Pseudocode

The sieve of Eratosthenes can be expressed in pseudocode, as follows:[7][8]

```
Input: an integer n > 1.

Let A be an array of Boolean values, indexed by integers 2 to n,
initially all set to true.

for i = 2, 3, 4, ..., not exceeding $\sqrt{n}$:
  if A[i] is true:
    for j = $i^2$, $i^2+i$, $i^2+2i$, $i^2+3i$, ..., not exceeding n:
      A[j] := false.

Output: all i such that A[i] is true.
```

This algorithm produces all primes not greater than *n*. It includes a common optimization, which is to start enumerating the multiples of each prime *i* from i^2. The time complexity of this algorithm is $O(n \log \log n)$,[8] provided the array update is an $O(1)$ operation, as is usually the case.

Segmented sieve

As Sorenson notes, the problem with the sieve of Eratosthenes is not the number of operations it performs but rather its memory requirements.[8] For large *n*, the range of primes may not fit in memory; worse, even for moderate *n*, its cache use is highly suboptimal. The algorithm walks through the entire array *A*, exhibiting almost no locality of reference.

A solution to these problems is offered by *segmented* sieves, where only portions of the range are sieved at a time.[9] These have been known since the 1970s, and work as follows:[8][10]

1. Divide the range 2 through *n* into segments of some size $\Delta \leq \sqrt{n}$.
2. Find the primes up to Δ, using the "regular" sieve.
3. For each Δ-sized segment up to *m*, with *m* from 2Δ to *n*, find the primes in it as follows:
   1. Set up a Boolean array of size Δ, and
   2. Eliminate from it the multiples of each prime $p \leq \sqrt{m}$ found so far, by calculating the lowest multiple of *p* between *m* - Δ and *m*, and enumerating its multiples in steps of *p* as usual, marking the corresponding positions in the array as non-prime.

If Δ is chosen to be $\sqrt{n}$, the space complexity of the algorithm is $O(\sqrt{n})$, while the time complexity is the same as that of the regular sieve.[8]

For ranges with upper limit *n* so large that the sieving primes below $\sqrt{n}$ as required by the page segmented sieve of Eratosthenes cannot fit in memory, a slower but much more space-efficient sieve like the sieve of Sorenson can be used instead.[11]

Incremental sieve

An incremental formulation of the sieve[2] generates primes indefinitely (i.e., without an upper bound) by interleaving the generation of primes with the generation of their multiples (so that primes can be found in gaps between the multiples), where the multiples of each prime *p* are generated directly by counting up from the square of the prime in increments of *p* (or $2p$ for odd primes). The generation must be initiated only when the prime's square is reached, to avoid adverse effects on efficiency. It can be expressed symbolically under the dataflow paradigm as

```
primes = [2, 3, ...] \ [[p², p²+p, ...] for p in primes],
```

using list comprehension notation with \ denoting set subtraction of arithmetic progressions of numbers.

Primes can also be produced by iteratively sieving out the composites through divisibility testing by sequential primes, one prime at a time. It is not the sieve of Eratosthenes but is often confused with it, even though the sieve of Eratosthenes directly generates the composites instead of testing for them. Trial division has worse theoretical complexity than that of the sieve of Eratosthenes in generating ranges of primes.[2]

When testing each prime, the *optimal* trial division algorithm uses all prime numbers not exceeding its square root, whereas the sieve of Eratosthenes produces each composite from its prime factors only, and gets the primes "for free", between the composites. The widely known 1975 functional sieve code by David Turner[12] is often presented as an example of the sieve of Eratosthenes[6] but is actually a sub-optimal trial division sieve.[2]

Computational analysis

The work performed by this algorithm is almost entirely the operations to cull the composite number representations which for the basic non-optimized version is the sum of the range divided by each of the primes up to that range or

$$n\sum_{p\le n}\frac{1}{p},$$

where n is the sieving range in this and all further analysis.

By rearranging Mertens' second theorem, this is equal to $n\ (\ \log\ \log\ n\ +\ M\ )$ as n approaches infinity, where M is the Meissel–Mertens constant of about 0.261 497 212 847 642 783 755 426 838 608 695 859 0516...

The optimization of starting at the square of each prime and only culling for primes less than the square root changes the "n" in the above expression to $\sqrt{n}$ (or $n^{\frac{1}{2}}$) and not culling until the square means that the sum of the base primes each minus two is subtracted from the operations. As the sum of the first x primes is $\frac{1}{2}x^2\ \log x$[13] and the prime number theorem says that x is approximately $\frac{x}{\log x}$, then the sum of primes to n is $\frac{n^2}{2\log n}$, and therefore the sum of base primes to $\sqrt{n}$ is $\frac{1}{\log n}$ expressed as a factor of n. The extra offset of two per base prime is $2\pi(\sqrt{n})$, where π is the prime-counting function in this case, or $\frac{4\sqrt{n}}{\log n}$; expressing this as a factor of n as are the other terms, this is $\frac{4}{\sqrt{n}\log n}$. Combining all of this, the expression for the number of optimized operations without wheel factorization is

$$\log\log n-\frac{1}{\log n}\left(1-\frac{4}{\sqrt{n}}\right)+M-\log 2.$$

For the wheel factorization cases, there is a further offset of the operations not done of

$$\sum_{p\le x}\frac{1}{p}$$

where x is the highest wheel prime and a constant factor of the whole expression is applied which is the fraction of remaining prime candidates as compared to the repeating wheel circumference. The wheel circumference is

$$\prod_{p\le x}p$$

and it can easily be determined that this wheel factor is

$$\prod_{p\le x}\frac{p-1}{p}$$

as $\frac{p-1}{p}$ is the fraction of remaining candidates for the highest wheel prime, x, and each succeeding smaller prime leaves its corresponding fraction of the previous combined fraction.

Combining all of the above analysis, the total number of operations for a sieving range up to n including wheel factorization for primes up to x is approximately

$$\prod_{p\le x}\frac{p-1}{p}\left(\log\log n-\frac{1}{\log n}\left(1-\frac{4}{\sqrt{n}}\right)+M-\log 2-\sum_{p\le x}\frac{1}{p}\right).$$

To show that the above expression is a good approximation to the number of composite number cull operations performed by the algorithm, following is a table showing the actually measured number of operations for a practical implementation of the sieve of Eratosthenes as compared to the number of operations predicted from the above expression with both expressed as a fraction of the range (rounded to four decimal places) for different sieve ranges and wheel factorizations (Note that the last column is a maximum practical wheel as to the size of the wheel gaps Look Up Table - almost 10 million values):

n	no wheel		odds		2/3/5 wheel		2/3/5/7 wheel		2/3/5/7/11/13/17/19 wheel	
10^3	1.4090	1.3745	0.4510	0.4372	0.1000	0.0909	0.0580	0.0453	0.0060	—
10^4	1.6962	1.6844	0.5972	0.5922	0.1764	0.1736	0.1176	0.1161	0.0473	0.0391
10^5	1.9299	1.9261	0.7148	0.7130	0.2388	0.2381	0.1719	0.1714	0.0799	0.0805
10^6	2.1218	2.1220	0.8109	0.8110	0.2902	0.2903	0.2161	0.2162	0.1134	0.1140
10^7	2.2850	2.2863	0.8925	0.8932	0.3337	0.3341	0.2534	0.2538	0.1419	0.1421
10^8	2.4257	2.4276	0.9628	0.9638	0.3713	0.3718	0.2856	0.2860	0.1660	0.1662

The above table shows that the above expression is a very good approximation to the total number of culling operations for sieve ranges of about a hundred thousand (10^5) and above.

Algorithmic complexity

The sieve of Eratosthenes is a popular way to benchmark computer performance.[14] As can be seen from the above by removing all constant offsets and constant factors and ignoring terms that tend to zero as n approaches infinity, the time complexity of calculating all primes below n in the random access machine model is $O(n\ \log\ \log\ n)$ operations, a direct consequence of the fact that the prime harmonic series asymptotically approaches $\log\log n$. It has an exponential time complexity with regard to input size, though, which makes it a pseudo-polynomial algorithm. The basic algorithm requires $O(n)$ of memory.

The bit complexity of the algorithm is $O(n\ (\log n)\ (\log\log n))$ bit operations with a memory requirement of $O(n)$.[15]

The normally implemented page segmented version has the same operational complexity of $O(n\ \log\ \log\ n)$ as the non-segmented version but reduces the space requirements to the very minimal size of the segment page plus the memory required to store the base primes less than the square root of the range used to cull composites from successive page segments of size $O\left(\frac{\sqrt{n}}{\log n}\right)$.