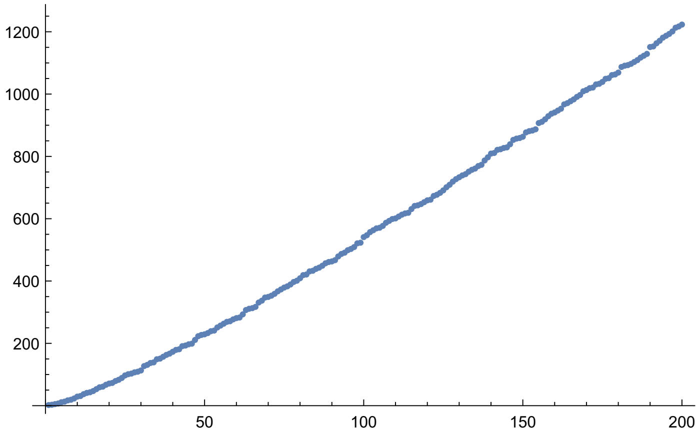
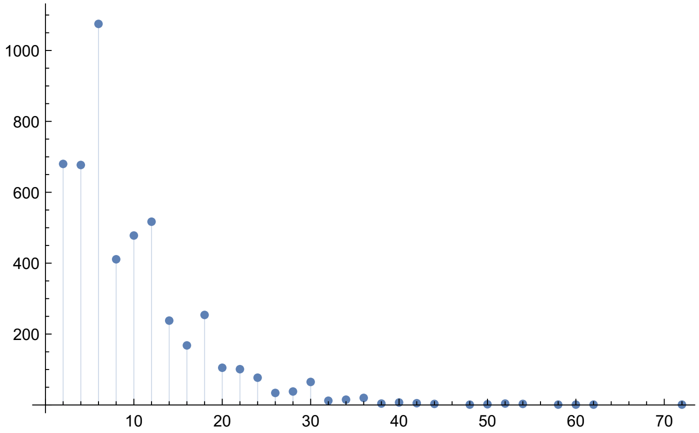
451: Hashing II

G. Miller, K. Sutner
Carnegie Mellon University
2020/10/20

Primality is one of the fundamental concepts in number theory. A lot is known about primes, yet lots of annoying questions remain, many having to do with the distribution of primes.

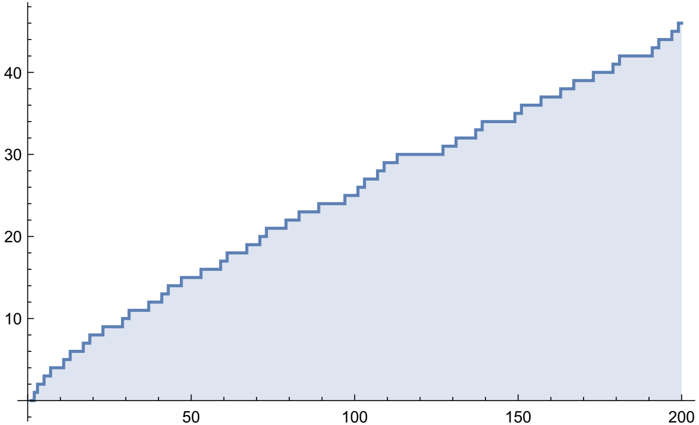


The first 200 primes.



Prime gaps between the first 5000 primes.

The prime pi function is defined by $\pi(n) = \#$ of primes in $[n]$.



As the picture suggests, calculating $\pi(n)$ precisely is going to be difficult. Fortunately, it often suffices to have reasonably good approximations.

Here is a result that was first proposed by Gauss in 1792 at the tender age of 15 (he liked to compute primes in his spare time):

$$\pi(x) \approx x/\ln x$$

In other words, the density of primes around n is about $1/\ln n$. Based on this density observation, Gauss (unpublished) and Dedekind (1838) suggested to use the logarithmic integral $\mathrm{Li}(x) = \int_2^x (\ln t)^{-1} dt$ to approximate π.

Alas, neither one was able to complete this line of reasoning.

The breakthrough came much later in the 19th century.

Theorem (Hadamard, de la Vallée Poussin, 1896)

$$\pi(x) = \mathsf{Li}(x) + O\left(\frac{x}{\ln x}\, e^{-c\sqrt{\ln x}}\right)$$

for some positive constant c.

This was an independent discovery, there are several other examples for this weird phenomenon. Sometimes, proofs are just "in the air."

If the Riemann hypothesis holds, this can be improved to

$$\pi(x) = \mathsf{Li}(x) + O\left(\sqrt{x}\ln x\right)$$

It follows from the PNT that indeed

$$\pi(n) \approx n/\ln n$$

since $\lim \mathsf{Li}(x)/(x/\ln x) = 1$ but $\lim(x/\ln x\, e^{-c\sqrt{\ln x}})/(x/\ln x) = 0$.

Corollary

Let $n \geq 2k \ln k$ for $k \geq 3$. Then $\pi(n) \geq k$.

Chebyshev (1848) produced a two-sided bound:

$$7/8 < \frac{\pi(n)}{n/\ln n} < 9/8$$

Task: Estimate the number of k-digit primes (primes in $[2^{k-1}, 2^k - 1]$).

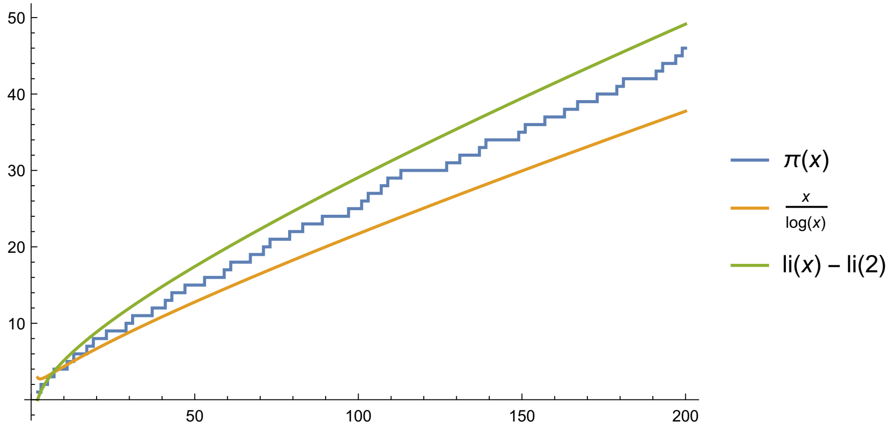
Chebyshev's two-sided bound produces the following lower bound:

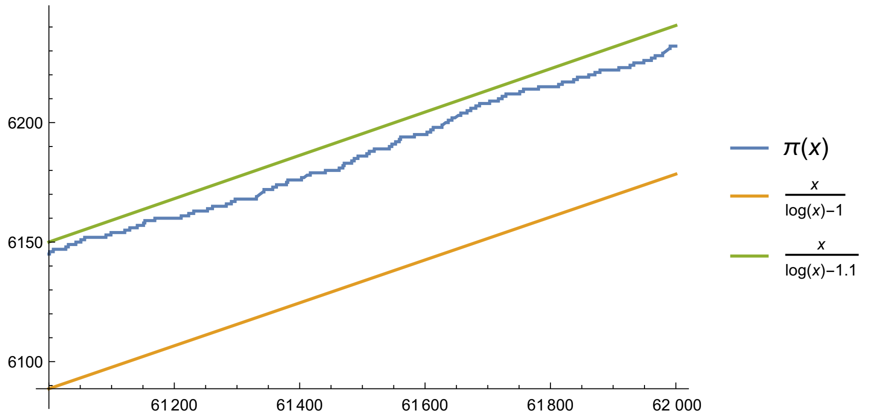
$$\frac{7\left(2^k - 1\right)}{8\ln\left(2^k - 1\right)} - \frac{9\left(2^{k-1} - 1\right)}{8\ln\left(2^{k-1} - 1\right)}$$

For example, for $k = 100$ we get 5.61×10^{27}. The length of the interval is about 6.34×10^{29}.

Exercise

Check the arithmetic. Explore better bounds obtained from slide 10.


50
40
30
20
10
50
100
150
200
$\pi(x)$
$\frac{x}{\log(x)}$
$\mathrm{li}(x) - \mathrm{li}(2)$


6100
6150
6200
61200
61400
61600
61800
62000
$\pi(x)$
$\frac{x}{\log(x)-1}$
$\frac{x}{\log(x)-1.1}$

- Sieve (Eratosthenes 276–196 BCE)
- Miller-Rabin test (1976, 1980)
- Solovay-Strassen test (1977)
- Agrawal-Kayal-Saxena test (2002)

Digression: AKS

In 2002, Agrawal, Kayal and Saxena showed that primality testing is in polynomial time, a major result in complexity theory: primality is trivially in co-$\mathbb{NP}$, by Pratt in $\mathbb{NP}$, then in BPP, and ultimately in $\mathbb{P}$.

Amazingly, the algorithm uses little more than high school arithmetic (Fermat's little theorem). This is in contrast to Miller-Rabin and Solovay-Strassen.

The original algorithm had time complexity $\widetilde{O}(n^{12})$ and has since been improved to $\widetilde{O}(n^6)$.

Alas, in practice, probabilistic algorithms are much superior. It seems AKS can't handle more that around 100 bits.

Suppose you need a random prime with k binary digits. Here is the method:

- Generate a random number n in the range $[2^{k-1}, 2^k - 1]$.
- Check if n is prime, using the algorithm of your choice.
  If not, continue with $n + 1$ (for the right value of 1).

This may sound a bit ham-fisted, but it's not bad: the probabilistic tests are fast, and the density of the primes ensures that the loop in part 2 will not execute too often.

Testing Equality

Here is the problem we already encountered in CRC stated in its own right.

Problem: Suppose $x, y \in \mathbf{2}^n$. We want to know whether $x = y$.

Of course, this is trivial as stated. So consider the following version: Alice has x, Bob has y and it is expensive for them to communicate, so we don't want to send all n bits.

Alice is going to send a single, short message to Bob to convince him that $x = y$: a fingerprint of x.

But we allow for the possibility of (one-sided) errors, with small probability:

- $x = y$ implies $\Pr[\text{Bob accepts}] = 1$,
- $x \neq y$ implies $\Pr[\text{Bob rejects}] = 1 - \varepsilon$.

Modular Arithmetic

Here is a first shot. M is a magic number whose proper value will be determined shortly.

- Alice picks a random prime p in $[M]$.
- Alice sends p and $x \bmod p$.
- Bob computes $y \bmod p$ and answers correspondingly.

Clearly there are no false negatives. How about false positives?

Lemma

Pick r and let $M = 2rn \log rn$.
Then the error probability when $x \neq y$ is less than $1/r$.

Proof

Let $0 \neq d = x - y$ and consider the prime factorization of d:

$$d = \prod_{i=1}^{k} p_i^{d_i}$$

Note that all the p_i are bounded by M. For $d = 0 \pmod{p}$ we need p to be one of these primes.

Then $2^k \leq d < p < n$: the number of prime divisors of d is bounded by n. But then the probability that one of them is equal to p is at most $n/\pi(M)$.

By our choice of M and the first section, this is at most $1/r$.

□

Matrix Multiplication

In a similar vein, suppose we have $n \times n$ matrices A, B and C, and we want to check whether $A \cdot B = C$, quickly. So we cannot compute $A \cdot B$.

- Pick $x \in \mathbf{2}^n$ uniformly at random.
- Check that $A \cdot (B \cdot x) = C \cdot x$.

Claim: Let $D = A \cdot B - C$. Then Pr$[D \neq 0 \wedge D \cdot x = 0] \leq 1/2$.

If $D \neq 0$ there is at least one row $D_k^{\rightarrow}$ with non-zero entries. But then $D_k^{\rightarrow} \circ x = 0$ with probability at most $1/2$: just flip one bit in the support of the row vector.

Let $\mathbb{F}$ be a field.

Lemma (Schwartz-Zippel 1980)

*Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be of degree d and $S \subseteq \mathbb{F}$ a set of cardinality s.
If P is not identically zero, then P has at most $d\, s^{n-1}$ roots in S.*

Proof.

The proof is by induction on n, the number of variables. The case $n = 1$ is clear.

For $n > 1$, define d_1 and $P_1(x_2, \ldots, x_n)$ to be the degree of x_1 in P and the coefficient, respectively. Hence

$$P(x_1, \ldots, x_n) = x_1^{d_1} P_1(x_2, \ldots, x_n) + \text{stuff}$$

Suppose $a_2, \ldots, a_n \in S$ is a root of P_1, by induction we know there are at most $s^{n-1}(d - d_1)$ such roots. Then $a, a_2, \ldots, a_n$ could be a root for P for all $a \in S$. Otherwise, there are at most d_1 such roots. Adding, we get the claim.

$\square$

Schwartz's Method, II

The set $S \subseteq \mathbb{F}$ here could be anything. For example, over $\mathbb{Q}$ or $\mathbb{Z}_p$ we might choose $S = \{0, 1, 2, \ldots, s-1\}$.

To check whether P is identically zero choose a point $\boldsymbol{a} \in S^n$ uniformly at random and evaluate $P(\boldsymbol{a})$. If P is not identically zero, then

$$\mathsf{Pr}[P(\boldsymbol{a}) = 0] \leq \frac{d}{s}$$

So by selecting S of cardinality $2d$ the error probability is 1/2.

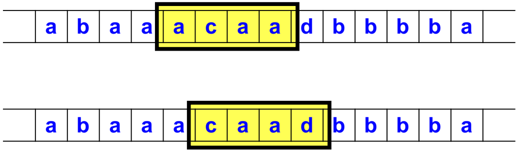
Note that the number of variables plays no role in the error bound.

Here is a modified version of the equality problem: find occurrences of a (short) word W, the pattern, inside a (long) word T, the text. Mainline algorithms:

- Brute force (stone age)
- Knuth-Morris-Pratt (1977, MP 1970)
- Boyer-Moore (1977, BMG 1979)
- Rabin-Karp (1987)
- Suffix trees (1973, 1976, 1995) — later

Design methods: combinatorics, finite state machines, data structures.

The brute force approach uses a sliding window of size $m = |W|$ across T and looks for matches with W:



Obviously, this is $O(m \cdot n)$ where $n = |T|$. Improvements are possible when one exploits a mismatch to move the window further than just one place; we won't go there.

Exercise

How could one exploit mismatches?

Write

$$T[i{:}k] = t_i\, t_{i+1}\, \ldots\, t_{i+k-2}\, t_{i+k-1}$$

for the infix (factor) of T starting at position i of length k (not from i to k).

To speed things up, we would like to have a hash function h defined on Σ^m with the following properties:

- If $T[i{:}m] \neq W$ then usually $h(T[i{:}m]) \neq h(W)$.
- $h(T[i{+}1{:}m])$ can be computed easily from $h(T[i{:}m])$ (a rolling hash).

If h is just the numerical value of the block under consideration, this is a perfect test as far as logic is concerned. Alas . . .

Rabin-Karp Fingerprinting

Suppose our text is just a sequence of bytes. We can think of the search string W as a number in base $B = 256$ (using 1-indexing). Here is a good choice for a rolling h:

$$\mathrm{val}(X) = x_1\, B^{m-1} + x_2\, B^{m-2} + \ldots + x_{m-1}\, B + x_m$$

It is indeed easy to compute $\mathrm{val}(T[i{+}1{:}m])$ from $\mathrm{val}(T[i{:}m])$.

$$\mathrm{val}(T[i{+}1{:}m]) = \big(\mathrm{val}(T[i{:}m])\, B\big) \bmod B^m + T_{i+m}.$$

Thus we only need $O(m + n)$ arithmetic steps to compute all the T values, so both conditions are satisfied. Alas, the arithmetic here is too slow, we are dealing with large numbers, not machine sized integers.

As usual, we combat large numbers by computing modulo some prime modulus.

Compute modulo p for some suitable prime p:
a machine sized integer, so that window-shifting is $O(1)$ regardless of m.

In other words, we generate the fingerprint $\mathrm{val}(T[i{:}m]) \bmod p$ and compare it to $\mathrm{val}(W) \bmod p$.

Since val is a function there are no false negatives.

To avoid false positives, we can verify a real hit at an additional cost of $O(m)$ steps using plain letter-by-letter comparisons.

Picking the Prime

Problem: How many false alarms are there?

Suppose s_0 is the number of correct hits, and s_1 the number of spurious hits. Then the running time is $O(n + (s_0 + s_1)m)$.

Let's say we pick p larger than m (which is easy to do).

Assuming that taking mods works like a random function (which is a bit of a stretch but not entirely unreasonable), we can estimate $s_1 = O(n/p)$. This yields a total running time of

$$O(n + (s_0 + n/p)m) = O(n + s_0 m).$$

Eliminating False Positives

We can use the same approach as in the last section to choose p so that the probability of getting a false positive at a particular location is $1/r$.

Using a union bound, the probability of getting at least one false positive is n/r. So by setting, say, $r = 100n$, the probability of even one error is just 1%.

This is not going to ruin running time, since the number of bits in p is still $O(\log m + \log n)$.

In the RealWorld$^{\mathrm{TM}}$, Rabin-Karp tends to be slower than KMP or Boyer-Moore.

However, Rabin-Karp has the advantage that it generalizes gracefully to multiple search words of the same length: keep their hashes in a container, and check for any match in the container.

Finite state machine based methods such as KMP also generalize to multiple search words (not necessarily of the same length), but things get a bit more complicated, see Aho-Corasick.

More Fingerprinting

The idea in CRC or Rabin-Karp is to compute a fingerprint or message digest to tackle the string equality problem. Here is another, similar idea that is closer to cryptography:

- secure hash

We want a function $h : \mathbf{2}^{\text{big}} \to \mathbf{2}^{\text{small}}$ so that for all inputs

- a minor modification x' of x results in $h(x) \neq h(x')$,
- it should be hard to compute x' such that $h(x) = h(x')$,
- it should be hard to compute $z \in h^{-1}(y)$.

Security/Speed Trade-Off

A hugely important distinction is whether the goal is to guard against unintentional errors (hard disk errors) or malicious attacks (forged documents).

In the unintentional case speed is the main goal, there is no concern about someone trying to cheat.

For cryptographically secure applications speed is (somewhat) less important, but one would like to have a clear understanding of the computational cost for an adversary during an attack (security claim).

Note that this is a moving target, as hardware gets more powerful formerly "secure" methods can become obsolete.

MD5 and SHA-1

Defining reports:

- MD5
- SHA-1
- security

Warning: Both MD5 and SHA-1 date back to the 1990s and are now considered obsolete wrto cryptographic applications. But, for plain corruption checks (say, data in a repository) they are still fine.

Basic Strategy

Let x be a byte sequence, ℓ its length. Pad with extra bits to

$$x_1 x_2 \ldots x_\ell \, 1\, 00 \ldots 00 \, \widehat{\ell}$$

where $\widehat{\ell}$ is ℓ written as a 64-bit number, and the number of 0's is chosen to insure that the new bit string has length a multiple of $\beta = 512$.

The padded message is processed in β-bit blocks in a number of rounds:

- initialize state
- mangle state and next block, get new state
- output last state

In other words, it's just a fancy finite state machine ;-)

MD5

The state consists of 4 words $\boldsymbol{h} = h_0, h_1, h_2, h_3$, all unsigned 32-bit.

Each block is written as 16 unsigned 32-bit ints $w_0, w_1, \ldots, w_{15}$.

Each block is processed in 64 rounds, using each of the w_i 4 times. The behavior in each round depends on the round number and uses auxiliary functions

$$f : (64) \times (\mathbf{2}^{32})^3 \to \mathbf{2}^{32}$$

as well as constants K_i and s_i.

Naturally,

$$K_i = \lfloor 4294967296 \, \mathrm{abs}(\sin(i+1)) \rfloor$$

The state (hash value) is initialized to

$$\begin{aligned} h_0 &= \texttt{0x01234567} \\ h_1 &= \texttt{0x89ABCDEF} \\ h_2 &= \texttt{0xFEDCBA98} \\ h_3 &= \texttt{0x76543210} \end{aligned}$$

The auxiliary functions are based on the following Boolean operations:

$$\begin{aligned} f_1(x,y,z) &= \mathsf{ift}(x,y,z) \\ f_2(x,y,z) &= (x \wedge z) \vee (y \wedge \neg z) \\ f_3(x,y,z) &= x \oplus y \oplus z \\ f_4(x,y,z) &= y \oplus (x \vee \neg z) \end{aligned}$$

See the report for details.

```
initialize

forall blocks do
    $\boldsymbol{u} = \boldsymbol{h}$                // 4 32-bit words
    forall $i = 0, \ldots, 63$
        $t = u_1 + \mathtt{rot}_{s_i}(u_0 + f(i, u_1, u_2, u_3) + K_i + w_{i \bmod 16})$
        $u_0 = u_3$
        $u_3 = u_2$
        $u_2 = u_1$
        $u_1 = t$
    $\boldsymbol{h} = \boldsymbol{h} + \boldsymbol{u}$
od

return $\boldsymbol{h}$
```

Again, each block is written as 16 unsigned 32-bit ints $w_0, w_1, \ldots, w_{15}$ and we process each block in sequence.

This time, the auxiliary function $f : (80) \times (\mathbf{2}^{32})^3 \to \mathbf{2}^{32}$ looks like so:

$$f(t,x,y,z) = \begin{cases} \mathsf{ift}(x,y,z) & \text{if } 0 \le t < 20, \\ x \oplus y \oplus z & \text{if } 20 \le t < 40 \text{ or } 60 \le t < 80, \\ (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & \text{if } 40 \le t < 60. \end{cases}$$

The state is initialized to

$$\begin{aligned} h_0 &= \texttt{0x67452301} \\ h_1 &= \texttt{0xEFCDAB89} \\ h_2 &= \texttt{0x98BADCFE} \\ h_3 &= \texttt{0x10325476} \\ h_4 &= \texttt{0xC3D2E1F0} \end{aligned}$$

The magic constants are less complicated, all 32-bit unsigned:

$$K_t = \begin{cases} \texttt{0x5A827999} & \text{if } 0 \le t < 20, \\ \texttt{0x6ED9EBA1} & \text{if } 20 \le t < 40, \\ \texttt{0x8F1BBCDC} & \text{if } 40 \le t < 60, \\ \texttt{0xCA62C1D6} & \text{if } 60 \le t < 80. \end{cases}$$

```
forall blocks do
    forall $i = 16, \ldots, 79$
        $w_i = \texttt{rot}_1(w_{i-3} \oplus w_{i-8} \oplus w_{i-14} \oplus w_{i-16})$

    $\boldsymbol{u} = \boldsymbol{h}$                // 5 32-bit words
    forall $i = 0, \ldots, 79$
        $t = \texttt{rot}_5(u_0) + f(i, u_1, u_2, u_3) + u_4 + K_i + w_i$
        $u_4 = u_3$
        $u_3 = u_2$
        $u_2 = \texttt{rot}_{30}(u_1)$
        $u_1 = u_0$
        $u_0 = t$
    $\boldsymbol{h} = \boldsymbol{h} + \boldsymbol{u}$
od

return $\texttt{rot}_{128}(h_0) \vee \texttt{rot}_{96}(h_1) \vee \texttt{rot}_{64}(h_2) \vee \texttt{rot}_{32}(h_3) \vee h_4$
```

Exercises

Exercise

Recall the requirements from the beginning of this section:

- *a minor modification x' of x has $h(x) \neq h(x')$,*
- *it should be hard to compute x' such that $h(x) = h(x')$,*
- *it should be hard to compute $z \in h^{-1}(x)$.*

Give a plausibility argument for why these should hold for MD5.

Exercise

Repeat for SHA-1.