Primality testing

Application of discrete models

In the previous documents, we have used prime numbers countless times. For example, for the RSA and the Diffie–Hellman key exchange, we needed large (thousands bits long) prime numbers. But how can we generate these primes in the first place?

This is not a trivial task. For example, the whole point of RSA was that if the secret prime numbers are large enough, then nobody can factor their product back into the prime factors – but then how do we check that they are indeed prime numbers if we can't factor such big numbers?

Also, do prime numbers of the needed size exist at all? Are there enough of them that we have any chance of finding them? Are there enough that it makes sense to choose secret prime numbers from them? (Because if there are too few, then an attacker may simply try all of them one by one.)

In this document, we attempt to find answers to these questions.

1 Number of primes

How many prime numbers are there? This can be answered in multiple ways. First:

Theorem 1.1 (Euclid's theorem)**.** *There are infinitely many prime numbers.*

(Proof: assume that there are only finitely many primes, for example $p_1, p_2, p_3, \ldots, p_n$. Then calculate the following number: $p_1 p_2 p_3 \ldots p_n + 1$. It cannot be prime, because it is greater than all assumed prime numbers, but it cannot be composite either, because none of our prime numbers divide it – so we reached a contradiction, meaning that there must be infinitely many prime numbers.)

But "infinitely many" is still not the full answer. For example, the natural numbers and the powers of two are both infinitely many, but the latters are still much less frequent. To formalize this, we can ask how many numbers of the desired type are there between 1 and n. For example:

	1 to n	1 to 1000000
Integers $(1, 2, 3, 4, 5, \ldots)$	n	1000000
Even numbers $(2, 4, 6, 8, 10, \ldots)$	$\lfloor n/2 \rfloor$	500000
Square numbers $(1, 4, 9, 16, 25, \ldots)$	$\lfloor \sqrt{n} \rfloor$	1000
Powers of two $(1, 2, 4, 8, 16, \ldots)$	$\lfloor \log_2 n \rfloor + 1$	20
Prime numbers $(2, 3, 5, 7, 11, \ldots)$	?	78498

For prime numbers, the following notation is used:

Definition 1.2 (prime-counting function)**.** Let $\boldsymbol{\pi(n)}$ be the number of primes between 1 and n.

Example. $\pi(7) = 4$, because there are four prime numbers up to 7, namely: $2, 3, 5, 7$.

There is no simple formula for the exact value of $\pi(n)$, but there is an approximation:

Theorem 1.3 (Prime number theorem)**.** $\pi(n) \approx \dfrac{n}{\ln n}$, *more precisely:* $\lim\limits_{n\to\infty} \dfrac{\pi(n)}{n/\ln n} = 1$.

In other words, the probability that a number between 1 and n is prime is ca. $1/\ln n$ (where $\ln n$ is the natural logarithm of n, i.e. $\ln n = \log_e n$, where $e \approx 2.718\ldots$).

Example. From 1 to 100, one quarter of them are primes: $\pi(100) = 25 = 100/4$, and $\ln 100 \approx 4.6$.

Example. From 1 to 1000000, ca. 1/13 of them are primes: $\pi(1000000) = 78498 \approx 1000000/12.7$, and $\ln 1000000 \approx 13.8$.

In Sage, $\pi(n)$ is called `prime_pi(n)`. The following code plots $\pi(n)$ and its approximation from 1 to 1000 (the upper, stepped line is $\pi(n)$):

```
sage: var("n"); plot([n/ln(n), prime_pi], (1, 1000))
```

2 Generating prime numbers

Assume that we want to generate a random prime number between given a and b. (For example, for three-digit prime numbers, $a = 100$ and $b = 999$, or for 1024-bit prime numbers for RSA, $a = 2^{1023}$ and $b = 2^{1024}-1$.)

The basic idea of the algorithm is simple: generate random numbers between a and b until we get a prime number. That is:

Prime generation:

Input: $a, b \in \mathbb{N}$ $(a < b)$
loop do
  $p :=$ random number between a and b
  if is_prime(p) **then**
    Output: p

But how many steps do we need with this loop until we find a prime number?

To answer this, we need to know the probability that the random number is prime. We can use the prime number theorem above to estimate the number of n-bit prime numbers ($2^{n-1} \le p \le 2^n-1$):

$$\pi(2^n) - \pi(2^{n-1}) \approx \frac{2^n}{\ln(2^n)} - \frac{2^{n-1}}{\ln(2^{n-1})} = \frac{2^n}{n\ln 2} - \frac{2^{n-1}}{(n-1)\ln 2} \approx \frac{2^n - 2^{n-1}}{n\ln 2} = \frac{2^{n-1}}{n\ln 2}.$$

This, divided by the total number of n-bit numbers, 2^{n-1}, gives that the probability of a random number being prime is $1/(n\ln 2)$. This means that the loop needs in average around $n\ln 2 \approx 0.69\,n$ steps (e.g. for $n = 1024$, it is $\sim$700 steps), which is linear in the length of the prime numbers – that is, there are enough prime numbers in the range that we can find one relatively quickly.

We can improve the algorithm by only ever generating *odd* random numbers, because prime numbers so big can only by odd anyway. (In binary, we can do this simply by changing the last bit of the random number to 1, e.g. in Sage/Python: `p |= 1`.) Then, since the base set is halved, the prime numbers will occur twice as often, so the number of steps is reduced by a half (e.g. for $n = 1024$, it now needs only $\sim$350 steps).

Note: it is worth emphasizing again that if the prime number is needed for a cryptographic purpose (such as RSA), then the random number should be generated by a cryptographically secure pseudorandom number generator (CSPRNG, see: Discrete logarithm), so that an outside listener cannot possibly predict it.

3 Types of primality tests

In the rest of this document, we will discuss how to decide whether the generated random number is prime or not, as needed by the above prime generation algorithm.

The simplest method is *trial division* (see: Divisors, prime numbers), which checks for each possible number if it is a divisor. However, this is unacceptably slow for large numbers: even the optimized version checks for divisors up to the square root of the input number, i.e. if it is 1024 bits long, the algorithm needs to check $\sim 2^{512}$ numbers – which is itself a 155-digit number! But this is not surprising, since trial division can also be used to factor the number, and the whole foundation of RSA was that it is not possible to factor such big numbers.

Regardless of this, trial division can still be useful in a limited way: it can be used to quickly filter out small prime factors before turning to other methods. For example, if our number is divisible by 3, 5 or 7 – which is true for almost half of the numbers – then we can find this out very quickly using trial division, and then we don't need to use the more complicated methods described below.

Fortunately, deciding whether a number is prime is much easier than factoring it. A method that does the former is called a **primality test**. There are two main kinds of primality tests:

1. A *deterministic primality test* can always tell correctly whether a number is prime.
2. A *probabilistic primality test* can't tell this for 100%, sometimes it makes mistakes. The commonly used tests only make mistakes in one direction: they might classify composite numbers as prime numbers, but they always give correct answer for prime numbers.

But why would we use a test that sometimes gives wrong answers? Because probabilistic primality tests are usually much faster. For example, trial divison is a deterministic primality test, and as we have seen above, it is very slow. And although there are more efficient deterministic tests too (not described here), but the commonly used probabilistic tests are even more efficient. Of course, the question is how often these tests make mistakes, and whether this frequency is still acceptable for our particular purpose.

Sage supports both types of primality tests: the built-in function `is_prime()` uses a deterministic primality test by default, but we can change it to use a probabilistic test instead, to make it faster, but allow some inaccuracy:

```
sage: n = 2^2048 + 981
sage: is_prime(n)    # deterministic: SLOW!
True
sage: proof.arithmetic(False)
sage: is_prime(n)    # probabilistic: fast, but unreliable
True
```

4 Fermat primality test

As a reminder from the earlier document called Discrete logarithm, Fermat's little theorem states the following: if p is a prime number and $a \not\equiv 0 \pmod{p}$, then

$$a^{p-1} \equiv 1 \pmod{p}.$$

This theorem is only about prime numbers, it doesn't say anything about other numbers. So we can ask the question: can it be true if p is a composite number?

To find an answer to this question, let's print the values of $a^{n-1} \bmod n$ for various n's and a's (where $1 \leq a \leq n-1$):

```
sage: for n in range(5, 16):
....:     print(n, [power_mod(a, n-1, n) for a in range(1, n)])
5 [1, 1, 1, 1]
6 [1, 2, 3, 4, 5]
7 [1, 1, 1, 1, 1, 1]
8 [1, 0, 3, 0, 5, 0, 7]
9 [1, 4, 0, 7, 7, 0, 4, 1]
10 [1, 2, 3, 4, 5, 6, 7, 8, 9]
11 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
12 [1, 8, 3, 4, 5, 0, 7, 8, 9, 4, 11]
13 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
14 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
15 [1, 4, 9, 1, 10, 6, 4, 4, 6, 10, 1, 9, 4, 1]
```

Notice the following:

1. If n is a prime number, then all values are 1, as guaranteed by Fermat's little theorem.
2. If $a = 1$ (first values), then of course its powers are always 1.
3. If $a = n-1$ (last values), then we still often get 1, because $n-1 \equiv -1 \pmod{n}$, and every second power of -1 is 1.
4. In every other case however – i.e. if n is a composite number and $2 \leq a \leq n-2$ –, we almost never get 1.
5. But sometimes we do: e.g. for $n = 15$ and $a = 4$, we get $4^{14} \equiv 1 \pmod{15}$.

We can use these observations to find out whether n is a prime number: calculate $a^{n-1} \bmod n$ for some a, and if the result is different from 1, then n is definitely not a prime number; and if it is 1, then it is *probably* prime. Therefore, we got a probabilistic primality test. It is called the **Fermat primality test**, and it works as follows:

Fermat primality test(n):

> **Input:** $n \in \mathbb{N}, n \geq 4$, the number to be tested
> Choose a random base $a \in \{2, 3, \ldots, n-2\}$.
> $x := a^{n-1} \bmod n$
> **if** $x \neq 1$ **then** $\rightarrow$ *false* (= "not prime")
> **if** $x = 1$ **then** $\rightarrow$ *true* (= "probably prime")

This is a very efficient algorithm, because the modular exponentiation can be calculated quickly using fast exponentiation (see: Discrete logarithm). Compare this with trial division, which requires $\sim \sqrt{n}$ divisions, while the Fermat primality test needs only $\leq 2 \log_2 n$ multiplications (see the fast exponentiation). E.g. if $n \approx 2^{1024}$, then the former is around 2^{512}, while the latter is $\leq 2048 = 2^{11}$.

The error probability can be reduced by repeating the test multiple times (for different random a's), and only accepting n as a prime number if it passes all tests (i.e. they all return true). For example, if $n = 15$ were tested only for $a = 4$, then n would pass the test as a "prime number", whereas if we repeat the test, e.g. for $a = 2$, then n is "caught" to be a composite number.

Definition 4.1. For an $a \in \mathbb{N}^+$, a composite number n is called a **Fermat pseudoprime** (to base a) if $a^{n-1} \equiv 1 \pmod{n}$. (It is sometimes simply called a **pseudoprime**.)

Example. $n = 15$ is a Fermat pseudoprime to base $a = 4$, because $4^{14} \equiv 1 \pmod{15}$.

That is, the Fermat pseudoprimes are the numbers which "fool" the Fermat primality test.

Note that this notion depends on the base a, i.e. the same number can be a pseudoprime for some a, but not for another a. To emphasize this, we have the following terms:

Definition 4.2. For a composite number n, the base $a \in \mathbb{N}^+$ is called a **Fermat liar** for n if n is a Fermat pseudoprime to base a, otherwise it is called a **Fermat witness** for n.

Example. $a = 4$ is a Fermat liar for $n = 15$, while $a = 2$ is a Fermat witness: $2^{14} \equiv 4 \not\equiv 1\ (15)$.

Fortunately, pseudoprimes are rare. For example, between 1 and 10000, there are only 22 Fermat pseudoprimes to base $a = 2$ (the smallest being 341), while the same range has 1229 prime numbers.

But there are certain pseudoprimes which really resist the Fermat primality test:

Definition 4.3. A composite number n is called a **Carmichael number** if for every base $a \in \mathbb{N}$ that is coprime to n, $a^{n-1} \equiv 1 \pmod{n}$.

That is, Carmichael numbers are those which are pseudoprimes to the most possible bases. (It cannot be true for more a's than this, because if a and n are not coprime, then the power cannot possibly be 1, as we have seen in the previous document.) The smallest Carmichael number is 561. Fortunately, these numbers are even rarer than regular pseudoprimes (there are only 7 Carmichael numbers between 1 and 10000).

The good news is that unless we hit a Carmichael number, it is quite easy to catch a pseudoprime:

Theorem 4.4. *If n is a composite number but not a Carmichael number, then for at least half of the a's between 1 and $n-1$, we have $a^{n-1} \not\equiv 1 \pmod{n}$.*

Which means that the Fermat primality test can encounter three possible cases:

1. If n is a prime number, then the test is always correct.
2. If n is a Carmichael number (which is rare), then the test is very unreliable.
3. Otherwise, the test is correct in at least 50% of the cases. This may seem low at first glance, but this is only a conservative lower bound, and we can increase the accuracy by repeating the test for different a's: when repeting k times, the error probability is $< 1/2^k$, e.g. for $k = 10$, the result is incorrect in less than 0.001 ($= 0.1\%$) of the cases.

5 Miller–Rabin primality test

This section introduces a more efficient and more accurate probabilistic test than the Fermat primality test. It is still based on Fermat's little theorem, and also on the following statement.

For real numbers, we know that the equation $x^2 = 1$ has only two solutions: $x = 1$ and $x = -1$. The following theorem states that in certain cases, this is also true for modular arithmetic:

Theorem 5.1. *If p is a prime number and $x^2 \equiv 1 \pmod{p}$, then $x \equiv 1 \pmod{p}$ or $x \equiv -1 \pmod{p}$.*

Example. If the modulus is $p = 5$, then the squares of $0, 1, 2, 3, 4$ are, respectively, $0, 1, 4, 4, 1$, i.e. the only solutions of $x^2 \equiv 1\ (5)$ are indeed $x \equiv 1\ (5)$ and $x \equiv 4 \equiv -1\ (5)$. But this is not necessarily true for composite numbers, e.g. $3^2 \equiv 1\ (8)$.

(Proof: the first congruence implies that $p \mid x^2 - 1 = (x-1)(x+1)$, but a prime number can only divide a product if it divides one of its factors, i.e. either $p \mid (x-1)$ or $p \mid (x+1)$.)

Going back to Fermat's little theorem, if p is a prime number and a is a nonzero base, then $a^{p-1} \equiv 1 \pmod{p}$. For example, for $p = 29$:

$$a^{28} \equiv 1 \pmod{29}.$$

Now if we apply the above theorem for $x = a^{14}$ (so that $x^2 = a^{28}$), i.e. we halve the exponent, then we get one of the following:

$$a^{14} \equiv 1 \quad \text{or} \quad a^{14} \equiv -1 \pmod{29}.$$

If $a^{14} \equiv 1$ (29), then we can halve the exponent again, and the theorem says that either

$$a^{7} \equiv 1 \quad \text{or} \quad a^{7} \equiv -1 \pmod{29}.$$

But now the exponent is odd, so we can't halve it further.

In each step of the above process, we used that p is a prime number. If it is not, than one of the steps may very easily fail (but not necessarily). This gives us another probabilistic primality test.
Example. If $n = 15$ and $a = 4$, then $a^{n-1} \equiv 1$ (n), i.e. $4^{14} \equiv 1$ (15) is true, but if we halve the exponent, then $4^7 \equiv 4$ (15), which is neither 1 nor -1, so it breaks the above "halving theorem", meaning that 15 cannot possibly be a prime number.

The **Miller–Rabin primality test** is based on the above logic, but in reverse order: instead of calculating a^{n-1} first and then halving the exponent, it first finds the last exponent that cannot be halved further (e.g. 7 for the above $p = 29$), calculates the power to that (e.g. a^7), and then doubles the exponent back until it reaches $n-1$ (e.g. $a^7 \to a^{14} \to a^{28}$). The point of this is that the powers with smaller exponents are easier to calculate, and in many cases, the test can stop sooner than $n-1$ if the result is already clear (e.g. if the above "halving theorem" is violated).

Miller–Rabin primality test(n):

Input: $n \in \mathbb{N}, n \geq 4$, the number to be tested
if n *is even* **then** $\to$ *false* (= "not prime")
Choose a random base $a \in \{2, 3, \ldots, n-2\}$.
Halve $d := n-1$ until it becomes odd, and count the halvings by k (so that $n-1 = 2^k d$).
$x := a^d \bmod n$
if $x = 1$ **then** $\to$ *true* (= "probably prime")
loop k *times* **do**
    if $x = n-1$ **then** $\to$ *true* (= "probably prime")
    $x := x^2 \bmod n$
    if $x = 1$ **then** $\to$ *false* (= "not prime")
$\to$ *false* (= "not prime")

The algorithm doubles the exponent until one of three conditions happen:

1. The power (x) becomes -1 (or $n-1$, as $n-1 \equiv -1$ (n)): then n passes the test (i.e. it is "probably prime"), because the square of -1 is 1, and so all subsequent squares will also be 1.
2. The power (x) becomes 1: then n is not a prime, because if it were, then the previous power would have been 1 or -1, and so the test would have already stopped then.
3. The exponent reaches $n-1$ (i.e. the loop terminates after doubling k times): then n is not a prime, because if it were, then the power would be 1, so we would have stopped already.

Example. $n = 29$ is a prime number, and indeed, the test accepts it for any base (because it always has a -1 before the 1's, or it has 1's from the beginning):

$$\begin{aligned} 2^7 &\equiv 12, & 2^{14} &\equiv -1, & 2^{28} &\equiv 1 \pmod{29} & \checkmark \\ 5^7 &\equiv -1, & 5^{14} &\equiv 1, & 5^{28} &\equiv 1 \pmod{29} & \checkmark \\ 7^7 &\equiv 1, & 7^{14} &\equiv 1, & 7^{28} &\equiv 1 \pmod{29} & \checkmark \end{aligned}$$

Example. $n = 45$ is a composite number, and the test indeed fails for any base we try (either because we get 1 without a -1, or because we never get 1):

$$\begin{aligned} 2^{11} &\equiv 23, & 2^{22} &\equiv 34, & 2^{44} &\equiv 31 \pmod{45} & \times \\ 8^{11} &\equiv 17, & 8^{22} &\equiv 19, & 8^{44} &\equiv 1 \pmod{45} & \times \\ 19^{11} &\equiv 19, & 19^{22} &\equiv 1, & 19^{44} &\equiv 1 \pmod{45} & \times \end{aligned}$$

We can see that the Miller–Rabin test is stronger than the Fermat test, which would accept 45 as prime for two of the above three bases (based only on the last 1).

Example. $n = 25$ is also a composite number, but the test doesn't always detect it: it does for some bases, but it passes for others:

$$\begin{aligned} 6^3 &\equiv 16, & 6^6 &\equiv 6, & 6^{12} &\equiv 11, & 6^{24} &\equiv 21 \pmod{25} & \times \\ 7^3 &\equiv 18, & 7^6 &\equiv -1, & 7^{12} &\equiv 1, & 7^{24} &\equiv 1 \pmod{25} & \checkmark\ (!) \end{aligned}$$

So the Miller–Rabin primality test also has its own pseudoprimes.

Definition 5.2. For an $a \in \mathbb{N}^+$, a composite number n is called a **strong pseudoprime** (to base a) if the Miller–Rabin primality test accepts n as a prime number for that a.

Example. $n = 25$ is a strong pseudoprime for $a = 7$ (and also a Fermat pseudoprime), see above.

Example. $n = 45$ is not a strong pseudoprime for any a (only a Fermat pseudoprime for e.g. $a = 8$).

All strong pseudoprimes are also Fermat pseudoprimes (because the Miller–Rabin test is an extension of the Fermat test), but not vice versa.

The Miller–Rabin primality test is better than the Fermat primality test for multiple reasons:

1. It doesn't have equivalents to the Carmichael numbers, i.e. numbers for which the Miller–Rabin test would be almost always wrong.
2. Furthermore, it is correct in at least 75% of the cases (compared to the Fermat test's 50%). This means that if we repeat the test k times, then the error probability is $< 1/4^k$ (compared to $1/2^k$), so the same accuracy can be achieved in half as many steps ($1/4^k = 1/2^{2k}$).
3. But even one step of the test can be faster than one step of the Fermat test, because the Miller–Rabin test doesn't always calculate the whole a^{n-1} power, as it can often stop earlier.

6 Author

This document was written by Uray M. János, partly using the following sources:

[1] https://compalg.elte.gitlab-pages.hu/dimoa-web/tematika/dimoa (Hungarian)

[2] https://en.wikipedia.org/wiki/Fermat_primality_test

[3] https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test

[4] J. Katz and Y. Lindell: Introduction to Modern Cryptography, Second Edition; Chapter 8.2: Primes, Factoring, and RSA

Errors and suggestions can be reported here: uray.janos@inf.elte.hu.