Lesson 13. Numerical Algorithms (2): Generating Prime Numbers

I111E – Algorithms and Data Structures

Ryuhei Uehara & Giovanni Viglietta
uehara@jaist.ac.jp & johnny@jaist.ac.jp

JAIST – November 28, 2019

All material is available at
www.jaist.ac.jp/~uehara/course/2019/i111e

Goals of today's lecture

- Efficiently test if a (large) number is prime
  - Use Fermat's little theorem
  - Be aware of Carmichael numbers
  - Learn about randomized algorithms
- Efficiently generate (large) prime numbers
  - Exploit the asymptotic distribution of primes
  - Correctly estimate the expected running time

Prime numbers

A prime is an integer > 1 that is only divisible by 1 and by itself.

E.g., 2, 3, 5, 7, 11, 13, 17, 19, 23, ... (there are infinitely many).

Theorem: every positive integer can be written as a product of prime numbers in a unique way. E.g., $90 = 2 \cdot 3 \cdot 3 \cdot 5$.

Prime numbers

A prime is an integer > 1 that is only divisible by 1 and by itself.

E.g., 2, 3, 5, 7, 11, 13, 17, 19, 23, ... (there are infinitely many).

Theorem: every positive integer can be written as a product of prime numbers in a unique way. E.g., $90 = 2 \cdot 3 \cdot 3 \cdot 5$.

The safety of modern cryptosystems relies on these facts:

- Testing if a (large) number is prime is easy.
- Finding a prime factor of a (large) number is hard.

  Note: if we search for the factors of a number by dividing it by all smaller numbers, we do exponentially many divisions!

How can we check if a number is prime without trying to factor it?

Fermat's little theorem

In 1640, Pierre de Fermat stated the following:

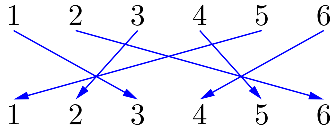
Theorem: if p is prime and $1 \leq a < p$, then $a^{p-1} \equiv 1 \pmod{p}$.

Fermat's little theorem

Theorem: if p is prime and $1 \leq a < p$, then $a^{p-1} \equiv 1 \pmod{p}$.

Proof: if we multiply the numbers $1, 2, \ldots, p-1$ by a,

we obtain a permutation of them. Example with $a = 3$ and $p = 7$:



This is because a and p are relatively prime, so:

$a \cdot i \equiv a \cdot j \pmod{p} \implies i \equiv j \pmod{p}$

(hence no two numbers are mapped into the same number)

and $a \cdot i \equiv 0 \pmod{p} \implies i \equiv 0 \pmod{p}$

(hence no number is mapped into 0).

So, $\{1, 2, \ldots, p-1\} = \{a \cdot 1 \bmod p, a \cdot 2 \bmod p, \ldots, a \cdot (p-1) \bmod p\}$.

Taking the products, $(p-1)! \equiv a^{p-1}(p-1)! \pmod{p}$.

But $(p-1)!$ is relatively prime to p, so $1 \equiv a^{p-1} \pmod{p}$.

A possible primality test

This theorem suggests a "factorless" test of primality:

- Given a positive integer N
- Randomly pick a "witness" a such that $1 \leq a < N$
- Compute $a^{N-1} \pmod{N}$ (in $O(n^3)$ time)
- If the result is <u>not 1</u>, return "N is not prime"
  (N contradicts Fermat's little theorem)
- Otherwise, return "N may be prime"

A possible primality test

This theorem suggests a "factorless" test of primality:

- Given a positive integer N
- Randomly pick a "witness" a such that $1 \leq a < N$
- Compute $a^{N-1} \pmod{N}$ (in $O(n^3)$ time)
- If the result is not 1, return "N is not prime"
  (N contradicts Fermat's little theorem)
- Otherwise, return "N may be prime"

Why "may be prime"?
Because Fermat's little theorem is not an if-and-only-if condition!

There are cases where N is not prime, but $a^{N-1} \equiv 1 \pmod{N}$:
if $N = 15 = 3 \cdot 5$ and $a = 4$, then $4^{14} \equiv (4^2)^7 \equiv 1^7 \equiv 1 \pmod{15}$.

Fortunately, if $N = 15$, all other choices of a witness $a > 1$
make the test correctly report that 15 is not a prime.

But there are much worse examples...

Carmichael numbers

There are non-prime numbers N for which every choice of a (relatively prime to N) makes the test return "N may be prime".

In 1910, Robert Carmichael found the smallest such number: 561.

Other examples are 1105, 1729, 2465, 2821, 6601, 8911, . . .

Bad news: there are infinitely many "Carmichael numbers".

Good news: they are very rare, so we may choose to ignore them!

Non-Carmichael numbers

So, our primality test is quite ineffective for Carmichael numbers. But what about all other numbers, which are the vast majority?

For a non-prime and non-Carmichael number N, there is at least a witness a relatively prime to N such that $a^{N-1} \not\equiv 1 \pmod{N}$.

We call a a "good witness", because it makes the test correctly report that N is not a prime. What about the other witnesses?

Non-Carmichael numbers

So, our primality test is quite ineffective for Carmichael numbers. But what about all other numbers, which are the vast majority?

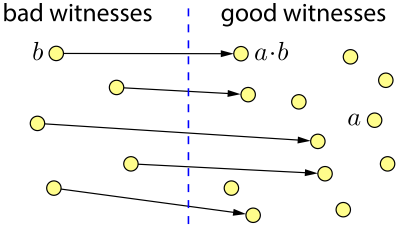
For a non-prime and non-Carmichael number N, there is at least a witness a relatively prime to N such that $a^{N-1} \not\equiv 1 \pmod{N}$.

We call a a "good witness", because it makes the test correctly report that N is not a prime. What about the other witnesses?

Theorem: if there is a good witness a relatively prime to N (i.e., if N is non-Carmichael), then at least half the witnesses are good.

Proof: every bad witness b has a good "twin" $a \cdot b$:
$(a \cdot b)^{N-1} \equiv a^{N-1} \cdot b^{N-1} \equiv a^{N-1} \cdot 1 \equiv a^{N-1} \not\equiv 1 \pmod{N}$.
And none of these twins are the same: if b and b' are bad witnesses, then $a \cdot b \equiv a \cdot b' \pmod{N} \implies b \equiv b' \pmod{N}$.
So, there are at least as many good witnesses as bad witnesses.

Fermat primality test



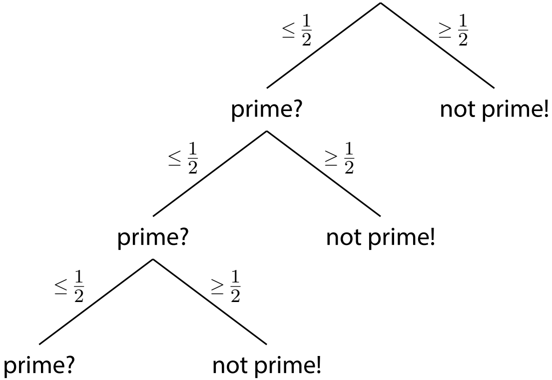
What are the consequences on our primality test?

- If N is prime, all witnesses are good (by Fermat's little theorem), so the test *always* reports that N may be prime.
- If N is not prime (and not Carmichael), then
  - $\geq 50\%$ of the witnesses are good (by the previous theorem), and correctly report that N is definitely not prime.
  - $\leq 50\%$ of the witnesses are bad, and wrongly report that N may be prime.

So the Fermat test has a probability of at most $1/2$ of being wrong!
Can we reduce this "one-sided" probability?

Fermat primality test

If we repeat the test k times (always picking a at random), the probability of getting the wrong answer is at most $1/2^k$: this can be made arbitrarily small!



Fermat primality test

An implementation using our C library from the previous lesson:

```
char* random_less(char* n) {
    int bits = num_length(n);
    char* a = malloc(bits + 1);
    for (int i = 0; i < bits; i++) a[i] = rand() % 2;
    a[bits] = -1;
    if (compare(a, n) != 1) a = sub(a, n);
    return a;
}

int test_prime(char* n, int k) {
    char* m = sub(n, one);
    for (int i = 0; i < k; i++) {
        char* a = add(random_less(m), one, 2);
        char* e = expM(a, m, n);
        if (compare(e, one) != 0) return 0;
    }
    return 1;
}
```

The running time is $O(kn^3)$, where n is the number of bits of N.

Distribution of prime numbers

We now want to generate a prime number of n bits.

How can we do it efficiently?

We need to know something about the distribution of primes:

Theorem: The number of primes $\leq x$ is asymptotic to $x/\ln x$.

If x is n bits long, then $n \approx \log_2 x$.

But $\ln x < \log_2 x \approx n$.

It follows that, among the x numbers of n bits,

at least a fraction of $1/n$ are primes.

$\rightarrow$ Prime numbers are abundant!

Prime numbers are everywhere

My lab:

67 is prime

My car's plate:

5297 is prime

Today's date in the Japanese calendar: $28/11/1 \to 28111$ is prime.

In this room, 2–3 people are likely to have a prime phone number.

Randomly generating prime numbers

This suggests a simple method for generating prime numbers:

- Pick a random number of n significant bits
- Test if it is prime: if it is, return it
- Otherwise, repeat from the first step

Randomly generating prime numbers

This suggests a simple method for generating prime numbers:

- Pick a random number of n significant bits
- Test if it is prime: if it is, return it
- Otherwise, repeat from the first step

```
char* random_bits(int bits) {
    char* a = malloc(bits + 1);
    for (int i = 0; i < bits - 1; i++) a[i] = rand() % 2;
    a[bits - 1] = 1;
    a[bits] = -1;
    return a;
}

char* generate_prime(int bits, int k) {
    char* n;
    do {
        n = random_bits(bits);
    } while (test_prime(n, k) == 0);
    return n;
}
```

How efficient is this algorithm? In the worst case, it will never find a prime! But what about the average case?

Expected running time

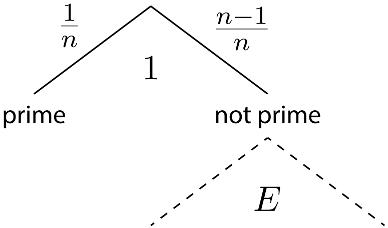
We know: a random n-bit number is prime with probability $1/n$.

We want: the expected number of times E we have to pick a random n-bit number before we find a prime.

Expected running time

We know: a random n-bit number is prime with probability $1/n$.

We want: the expected number of times E we have to pick a random n-bit number before we find a prime.



After the first extraction, we get a prime with probability $1/n$.

Otherwise, we have to perform E more extractions on average.

This yields the equation $E = \frac{1}{n} \cdot 1 + \frac{n-1}{n} \cdot (1 + E)$.

Solving for E, we get $E = n$.

On average, our algorithm runs in $O(kn^3) \cdot n = O(kn^4)$ time.

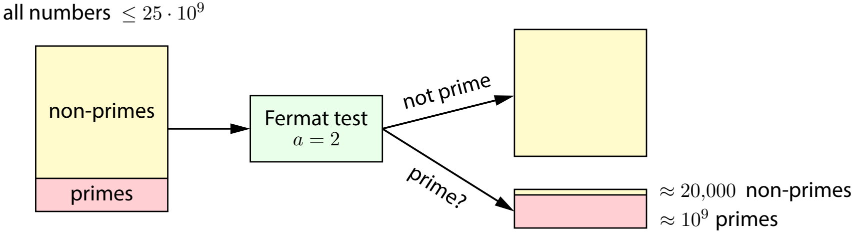
Effectiveness of the Fermat test

How good is the Fermat test at finding prime numbers?

As we know, the chance of a false positive is 50% in the worst case.

But on randomly chosen numbers, it is typically much lower!

Even $a = 2$ is a good witness for the vast majority of numbers:



The chance of erroneously outputting a non-prime 36-bit number is $\approx 20{,}000/10^9 = 0.002\%$, and it drops rapidly with higher n and k.

Announcements

Next lesson:

December 2 (Mon)—Numerical Algorithms (3): Cryptography

Questionnaire: last 10 minutes. Bring your laptop!

Final exam: December 4 (Wed), 10:50–12:30