Number Theoretic Algorithms

Dov Kruger

Department of Electrical and Computer Engineering
Rutgers University

March 26, 2024

Number theory is a branch of mathematics involving integers
Computation of integer operations (large numbers)
Primality Testing

Practical applications

- Coding and Cryptography (RSA)
- Random Number Generation
- Hash Functions
- Graphics

Warmup: GCD

The Greatest Common Divisor is a simple algorithm with a brute force and brilliant solution

Brute force: try every divisor $O(\sqrt{n})$

Euclid's algorithm is much better.

Brute Force GCD

```
brute_gcd(a, b)
  lim ← min(a,b);
  best ← 1
  for i ← 2 to lim
    if a mod i == 0 and b mod i = 0
      best ← i
    end
  end
  return best
end
```

Still Brute Force GCD

```
brute_gcd(a, b)
  lim ← min(a,b);
  for i ← lim downto 2
    if a mod i == 0 and b mod i = 0
      return i
    end
  end
end
```

Euclid, Recursively

```
gcd(a,b)
  if b == 0
    return a;
  end
  return gcd(b, a mod b)
end
```

Overview

Number theory only appears theoretical

In this session, we will answer some problems that turn out to have practical importance

- How fast can we test whether a number is prime or not?
- How fast can we factor a number into primes?

While cryptography is a huge field, we will also explore some basics

- How can messages be kept secret?
- How can keys be distributed in an environment where eavesdropping is possible?

Definition of Prime Numbers

A prime number is a positive integer that is evenly divisible only by itself and 1

1 is not considered prime

2, 3, and 5 are prime

4 is not $(2 * 2 = 4)$

In order to determine if a number is prime, divide by every number smaller than itself

```
bool isPrime(int n) {
  for (int i = 2; i < n; i++)
    if (n % i == 0)
      return false;
  return true;
}
```

No need to test numbers up to n-1
Test up to n/2

```
bool isPrime(uint64_t n) {
  for (uint64_t i = 2; i < n/2; i++) {
    if (n % i == 0)
      return false;
  }
  return true;
}
```

No need for divisors past $\sqrt{n}$

```
bool isPrime(uint64_t n) {
  for (uint64_t i = 2; i <= sqrt(n); i++) {
    if (n % i == 0)
      return false;
  }
  return true;
}
```

Eratosthenes: A Completely Different Approach

Eratosthenes' 2500 years ago

Avoid trial division by starting with the divisors

Wipe out all multiples and see what remains

[diag here showing numbers 2 to 25 with strikeouts, see notes, my diagram is not good]

Original Eratosthenes Algorithm

```
eratosthenes(n)
    isPrime ← new boolean[n+1]
    isPrime[*] ← true
    for i ← 2 to n
      if isPrime[i]
        print i
        for j ← 2*i to n step i
         isPrime[j] ← false
        end
      end
    end
end
```

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	0	1	00	1	0	0	00	1	00	1	0	00	0	1	00	1	00	0

Improved Eratosthenes Algorithm

```
improved_eratosthenes(n)
  isPrime ← new boolean[n+1]
  isPrime[*] ← true
  for i ← 2 to n
    if isPrime[i]
      print i
      for j ← i*i to n step 2i
        isPrime[j] ← false
      end
    end
  end
end
```

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0

The analysis of complexity is tough because it depends on an if statement
It turns out the density of prime numbers is $1/\log_2 n$

```
for i ← 2 to n
    if isPrime[i]           // how often is this true?
```

Q: At n = 1,000,000,000 how many numbers are prime?
At n = 1,000,000,000,000 how many numbers are prime?

Fermat is most famous for his "last theorem"

$x^n + y^n = z^n$, for $n \geq 3$ there are no solutions

https://www.npr.org/sections/thetwo-way/2016/03/17/470786922/professor-who-solved-fermat-s-last-theorem-wins-math-s-abel-prize

This one thus far has no practical uses, but his *little theorem* is a different story
*Note Mathematicians have obsessed over this problem for 300 years. Excellent movie on this (Spanish) "Fermat's Room" `https://www.imdb.com/title/tt1016301/`

For any positive integer n, pick a witness a, $1 < a < n$
if is is prime, then $a^{n-1} \mod n = 1$

On its face, this seems ridiculous
Computing a^{n-1} is far worse than the original problem. Consider...

a=2
n=10000000000000000000000017

would require computing

$2^{10000000000000000000000017}$ (a gigantic number)
3.28×10^{80} estimated number of particles in the universe
The observable universe is insufficient to store the number!

Computing function is not the same as computing the function mod n
Consider the following problem:

compute $n!,\ n = 10^{12}$

compute $n! \mod 10$

The first cannot be computed. The second $= 0$
How do we know? Because the last digit is ALWAYS zero beyond 7
$7! = 4940$
$9! = 362880$
$10! = 3628800$

Computing Exponents: The power Algorithm

The power algorithm computes x^n

```
power(x, n)
   prod ← 1
   while n > 0
      if n AND 1 ≠ 0
         prod ← prod * x
      end
      x ← x * x
      n ← n/2
   end
end
```

Power Algorithm at Work

power(2, 17)
prod starts at 1
After each row in the table n is divided by 2, x is squared

x	n	n odd	prod
2	17	yes	2
$2^2 = 4$	8	no	2
$4^2 = 16$	4	no	2
$16^2 = 256$	2	no	2
$256^2 = 65536$	1	yes	$2 * 2^{16} = 2^{17} = 131072$

The powermod algorithm computes $x^n \mod m$
Notice the two places in the code where the result is $\mod m$

```
powermod(x, n, m)
   prod ← 1
   while n > 0
      if n AND 1 ≠ 0
         prod ← prod * x   mod  m
      end
      x ← x * x   mod  m
      n ← n/2
   end
end
```

Because powermod always keeps the result modulo m, it never gets large

powermod(5, 9, 13) $= 5^9 \mod 13 = 1953125 \mod 13 = 5$

prod starts at 1

After each row in the table n is divided by 2, x is squared

x	n	n odd	$prod = prod * x \mod n$
5	9	yes	$1 * 5 \mod 13 = 5$
$5^2 = 25 \mod 13 = 12$	4	no	5
$12^2 = 144 \mod 13 = 1$	2	no	5
$1^2 = 1 \mod 13 = 1$	1	yes	$5 * 1 \mod 13 = 5$

$5^9 \mod 13 = 5$

Powermod Algorithm at Work, 2nd example

$powermod(6, 9, 17) = 6^9 \mod 17 = 10077696 \mod 17 = 11$
prod starts at 1
After each row in the table n is divided by 2, x is squared

x	n	n odd	prod
6	9	yes	$1 * 6 \mod 17 = 6$
$6^2 = 36 \mod 17 = 2$	4	no	6
$2^2 = 4 \mod 17 = 4$	2	no	6
$4^2 = 16 \mod 17 = 16$	1	yes	$(6 * 16) \mod 17 = 11$

$6^9 \mod 17 = 11$

Fermat Primality Testing

To test for primes using Fermat, reverse his theorem
if n is prime, then $a^n \mod m = 1$
So what if $a^n \mod m = 1$, does this mean that n must be prime?
Not quite. Mostly true.
$powermod(3, 911110, 911111) = 1$, 911111 is prime.
$powermod(6, 96, 97) = 1$, 97 is prime.
$powermod(5, 560, 561) = 1$, Error: 561 is not prime!

A Carmichael Number

- Is a pseudoprime that will pass the Fermat test for many witnesses but still is not prime.
- is a square-free composite number.

see: `https://en.wikipedia.org/wiki/Carmichael_number`

The first three Carmichael numbers are:

561	$3 * 11 * 17$
1105	$5 * 13 * 17$
1729	$7 * 13 * 19$

The only way a Carmichael number will return false for the Fermat test is if the witness is chosen as one of the factors of the Carmichael number.

This means, in practice, trying all factors which is the brute force algorithm (trial division).

Fermat is a Probabilistic Algorithm

Fermat is not guaranteed to work. For any single witness a
if $a^{p-1} \mod p = 1$m, p *may* be prime
The number is *probably* prime.
Perform k trials with random witnesses
If $powermod(a, p-1, p) \neq 1$ for any test, NOT PRIME

The Fermat algorithm

```
Fermat(p, k)
  for i ← 1 to k
    a ← random(2, p−1)
    if powermod(a, p − 1, p) ≠ 1
      return false // definitely not prime!
    end
  end
  return true // probably prime!
end
```

Carmichael numbers are a problem, but very rare.
There would be three mistakes for numbers below 2000, numbers which would reported prime, but are not.
But no one would ever use this method for small integers
For $p < 10^{21}$ there are only $20,138,200$ Carmichael numbers, so the probability is extremely low.
Still, there are better ways.

Miller-Rabin solves the problem of Carmichael numbers by determining whether the number is

- A Carmichael number
- A real prime

Procedure:

- Compute $p - 1$ which must be even for any prime > 2
- Split the number into the leading digits d and trailing zeros
- Count the number of trailing zeros
- Perform k trials of the Miller Rabin test
- For each test, pick a witness $2 \leq a < p - 1$
- $\leftarrow powermod(a, d - 1, d)$
- if $x = 1 or x = -1$

First split the algorithm into leading digits and trailing zeros
Example:
$p = 10001110001$ $p - 1 = 10001110000$ $d = 1000111, s = 4$

```
MillerRabin(p, k)  //O(k log n)
  s ← 0
  d ← p − 1
  while d mod 2 = 0
    d ← d / 2
    s ← s + 1
  end
```

```
witnessLoop:
  repeat k times
    a ← random(2, p − 1)
    x ← powermod(a, p − 1, p)
    if x = 1 or x = n − 1 then
      continue WitnessLoop
    repeat s − 1 times:
        x ← x^2 mod p
        if x = p − 1 then
          continue WitnessLoop
    end
    return false (definitely not prime)
  end
return true (probably prime)
```

if $x^2 \mod m = 1$ then x was ± 1

Miller-Rabin is Probabilistic

However, it has been tried and precomputed up to very large numbers

From Wikipedia:

$n < 1,373,653$	$a = 2, 3$
$n < 9,080,191$	$a = 71, 73$
$n < 25,326,001$	$a = 2, 3$
$n < 3,215,031,751$	$a = 2, 3, 5, 7$

AKS

Agrawal, Kayal, Saxena 2002

1. Far slower than Miller-Rabin
2. Revolutionary because it is deterministic
3. Know certainly whether a number is prime or not without trial division
4. If this is possible, perhaps it is possible to factor in less than $O(\sqrt{n})$?