A Brief Introduction to Computational Number Theory

Christian Engman

Georgia Tech Big O Theory Club

4/21/2023

Introduction

- Number theory is typically defined as the study of the integers.
- At the core of almost all problems in number theory is the study of prime numbers.
- The fundamental theorem: every natural number has a unique prime factorization
- Core computational questions:
  - Can we test if a number is prime?
  - Can we factor a number into primes?

The Trivial Approach

- For both factoring and prime-testing a number n, there is an obvious algorithm: for every $k < n$, determine if $k|n$
- This checks $n - 1$ numbers, but we can improve this down to $\sqrt{n}$ since factors come in pairs.
- An $O(\sqrt{n})$ algorithm seems pretty good, right?

Note on algorithmic complexity

In CS, we like to talk about complexity in terms of the number of bits in the input (b). Addition is $\Theta(b)$, and (naive) multiplication is $O(b^2)$. Trial division, however, is $O(2^{b/2})$, which is quite slow.

The Fermat Test

Let's take a look at prime testing. Our first algorithm uses the following theorem:

Fermat's Little Theorem

If p is a prime number and $a \in \mathbb{N}$ s.t. $p \nmid a$, then:

$$a^{p-1} \equiv 1 \mod p$$

- This tells us that, if $a^{n-1} \neq 1 \mod n$, n must be composite. The Fermat test, then, is simple: pick some number of a_i randomly, and if $a_i^{n-1} \neq 1 \mod n$, we return that n is composite, otherwise, return that it is prime.
- If n is composite and not a Carmichael Number, then more than half of all $a \nmid n$ will give $a^{n-1} \neq 1 \mod 1$, so, if we test enough a's, we have a high probability of being right.

A fatal flaw: Carmichael Numbers

Carmichael Number

A Carmichael Number is a composite number n s.t.

$$b^n \equiv b \mod n, \forall b \in \mathbb{N}$$

- Carmichael Numbers are relatively rare (the first one is 561, so they are much less common than primes).
- However, one can prove that there are infinitely many of them, which means that, no matter how large the number you are trying to test, there is always a chance that it is a Carmichael number.
- Because of the fundamental property of Carmichael numbers, Fermat's test will always return prime on one, no matter how many a's we test.

An Improvement: The Miller-Rabin Test

FLT Corollary

Suppose n is an odd prime and $n-1=2^s t$, where t is odd. If a is not divisible by n then one of the following is true:

$$a^t \equiv 1 \mod n$$

$$\exists i \in 0,\dots,s-1, \text{ s.t. } a^{2^i t} \equiv -1 \mod n$$

If n is composite, there exists an a s.t. neither is true.

- Algorithm: pick several a_i randomly, and, if one of the above is true for every a_i, return prime, otherwise, return composite.
- test is still probabilistic, but there are no numbers where we will always fail like in the Fermat Test.
- Miller-Rabin is often used in practice, as it requires only $\tilde{O}(kb^2)$ time, where k is the number of a_i's checked.

Other Primality Tests

- Probabilistic Tests
  - Solovay–Strassen: $\tilde{O}(kb^2)$
  - Frobenius primality test
  - Baillie–PSW primality test
- Deterministic Tests (under assumptions)
  - Miller's Test (deterministic version of Miller-Rabin): $\tilde{O}(b^4)$
  - Elliptic Curve Primality Test: $\tilde{O}(b^6)$
- Provably Deterministic tests
  - Agrawal, Kayal and Saxena: $\tilde{O}(b^6)$
- AKS actually tells us that primality testing is in $\mathcal{P}$, which is good news!

The Integer Factorization Problem

- Integer Factorization, taking a number and finding it's prime factors, is arguably the fundamental algorithmic problem in number theory.
- If we could factor numbers "fast", we would be able to break the RSA and ECC public-key cryptosystems.
- Decision variant is known to be NP and Co-NP, (since multiplication is polynomial time). However, A classical polynomial algorithm, a proof/disproof of NP-completeness, and a proof of classical hardness have evaded mathematicians and computer scientists for decades.
- We saw that trial division is $O(n^{1/2})$ time. Though it is not known if we can get to polynomial in $b = \log n$, however, we can do much better than naive.

Pollard's ρ Algorithm: A Monte-Carlo approach

- Suppose $g(x)$ is a nonlinear function on $\mathbb{F}_p$. It had been widely observed that sequences of the form $x, g(x), g(g(x)), \ldots$ behave chaotically, and have often been characterized as pseudorandom (though we do not have rigorous results characterizing this randomness).
- Note also that, since $\mathbb{F}_p$ is finite, the sequence $\mathbb{F}_p$ is guaranteed to repeat itself after some point, and, after this point, will become cyclic. This gives us the ρ shape that is often used to describe these types of sequences.
- If we pick a starting point $x_0 \in \mathbb{N}$, then, after a maximum of p iterations of the sequence $x_i = g(x_{i-1})$, we are guaranteed to have some $x_i \equiv x_j \mod p$, where $i \neq j$.

Pollard's ρ Algorithm: A Monte-Carlo approach

The Algorithm

- Pick a $g(x)$ (usually $g(x) = x^2 + 1$) and x_0 (usually 2)
- Let $x \leftarrow x_0$, $y \leftarrow x_0$, and $d \leftarrow 1$
- while $d = 1$, Let $x \leftarrow g(x) \mod n$, $y \leftarrow g(g(y)) \mod n$, $d \leftarrow \gcd(|x - y|, n)$
- If $d = n$, try again with a new x_0. $d \neq n$, we have found a nontrivial factor of n

- If we assume that $x_0, g(x_0), g(g(x_0)), \ldots,$ is roughly uniformly random in $\mathbb{F}_p$, We expect a repeated element of the sequence after about $\sqrt{p}$ iterations. We are most likely to find the smallest factor of n first.
- If p is the smallest nontrivial factor of n, we expect to terminate in about $\sqrt{p}$ iterations. Average time $O(n^{1/4})$ algorithm, which is a quadratic speedup over trial division.

Fast Algorithms For Special-Case Factorization

Similar to the Pollard ρ algorithm, there are many other algorithms that, though are generally exponential time, can give us results very quickly for special numbers:

- Pollard ρ with Brent-cycle finding (constant speedup over original)
- Fermat factorization (For 'close' factors)
- Pollard's $p-1$ algorithm ($O(\ln n)^2$ in special cases)
- William's $p+1$ algorithm (Variant of $p-1$)
- Lenstra's Elliptic Curve Method: $L_p\left[\frac{1}{2}, \sqrt{2}\right]$ (good for large numbers w/ small factors)
- Special Number Field Sieve (Generally observed to be fast for numbers of the from $r^e \pm s$, where e and s are small)

General-Purpose Factorization algorithms

Special-case algorithms perform well in many cases, but for general numbers of a large size, especially RSA numbers of the form $n = p \cdot q$ with p and q sufficiently far apart, they provide us not advantages and are far too slow. State-of-the art subexponential algorithms are commonly used in this case:

- Dixon's Algorithm: $L_n(1/2, 2\sqrt{2})$
- Quadratic Sieve (an improvement on Dixon): $L_n(1/2, 1 + o(1))$
- Rational Sieve (special case of GNFS)
- General Number Field Sieve (best known worst case): $L_n(1/3, (64/9)^{1/3})$ under GRH

The Final Boss: Shor's Algorithm

- No known polytime classical algorithms are known, however, due to Peter Shor, we know of a polynomial time algorithm for Quantum computers.
- Due to fast operations on qubits, particularly the Quantum Fourier Transform, Shor's algorithm can run in time $O(\log^2 n \log\log n)$, which is only a log factor off of naive multiplication.
- Though theoretically impressive, however, Quantum hardware is plagued by reliability issues, and, as of now, the largest number that has been factored with Shor's algorithm is 21.
- Even though quantum computing development has been relatively slow, many in cryptography are still concerned, and have looked to build public-key cryptosystems on NP-hard problems instead.

Some Other Problems

We took a look at two of the most common computational problems in number theory. Here are a couple more interesting ones:

- Solving Discrete Logarithms: given $a, b, n \in \mathbb{N}$, find e such that $a^e \equiv b \mod n$
- Solving Diophantine Equations: Given an equation in multiple integer variables, when and how quickly can we find solutions? (These problems are studied commonly in algebraic number theory)
- Solving general congruences: Can we solve congruences of the form $f(x) \equiv a \mod n$, when f is linear, quadratic, polynomial, etc?