Chapter 7
Randomization

Algorithm Theory
WS 2019/20

Philipp Bamberger

Primality Testing

Problem: Given a natural number $n \geq 2$, is n a prime number?

Simple primality test:

1. **if** n is even **then**
2.   **return** $(n = 2)$
3. **for** $i := 1$ **to** $\lfloor \sqrt{n}/2 \rfloor$ **do**
4.   **if** $2i + 1$ divides n **then**
5.   **return false**
6. **return true**

- **Running time:** $O(\sqrt{n})$

A Better Algorithm?

- How can we test primality efficiently?
- We need a little bit of basic number theory…

Square Roots of Unity: If p is a prime, the only solutions of the equation $x^2 \equiv 1 \pmod{p}$ are $x \equiv \pm 1 \pmod{p}$

- If we find an $x \not\equiv \pm 1 \pmod{n}$ such that $x^2 \equiv 1 \pmod{n}$, we can conclude that n is not a prime.

Algorithm Idea

Claim: Let $p > 2$ be a prime number such that $p - 1 = 2^s d$ for an integer $s \geq 1$ and some odd integer d. Then for all $a \in \{1, \dots, p-1\}$

$$a^d \equiv 1 \pmod{p} \;\; \textbf{or} \;\; a^{2^r d} \equiv -1 \pmod{p} \;\; \text{for some } 0 \leq r < s.$$

Proof:

- **Fermat's Little Theorem:** Given a prime number p,

$$\forall a \in \{1, \dots, p-1\}: \quad a^{p-1} \equiv 1 \pmod{p}$$

Primality Test

We have: If n is an odd prime and $n - 1 = 2^s d$ for an integer $s \geq 1$ and an odd integer d. Then for all $a \in \{1, \dots, n-1\}$,

$$a^d \equiv 1 \pmod{n} \;\; \textbf{or} \;\; a^{2^r d} \equiv -1 \pmod{n} \;\; \text{for some } 0 \leq r < s.$$

Idea: If we find an $a \in \{1, \dots, n-1\}$ such that

$$a^d \not\equiv 1 \pmod{n} \;\; \textbf{and} \;\; a^{2^r d} \not\equiv -1 \pmod{n} \;\; \text{for all } 0 \leq r < s,$$

we can conclude that n is not a prime.

- For every odd composite $n > 2$, at least $3/4$ of all possible a satisfy the above condition
- How can we find such a *witness* a efficiently?

Miller-Rabin Primality Test

- Given a natural number $n \geq 2$, is n a prime number?

Miller-Rabin Test:

1. **if** n is even **then return** $(n = 2)$
2. compute s, d such that $n - 1 = 2^s d$;
3. choose $a \in \{2, \ldots, n-2\}$ uniformly at random;
4. $x \coloneqq a^d \bmod n$;
5. **if** $x = 1$ **or** $x = n - 1$ **then return probably prime;**
6. **for** $r \coloneqq 1$ **to** $s - 1$ **do**
7. $\quad x \coloneqq x^2 \bmod n$;
8. $\quad$ **if** $x = n - 1$ **then return probably prime;**
9. **return composite;**

Analysis

Theorem:

- If n is prime, the Miller-Rabin test always returns **true**.
- If n is composite, the Miller-Rabin test returns **false** with probability at least $^3/_4$.

Corollary: If the Miller-Rabin test is repeated k times, it fails to detect a composite number n with probability at most 4^{-k}.

Running Time

Cost of Modular Arithmetic:

- Representation of a number $x \in \mathbb{Z}_n$: $O(\log n)$ bits
- Cost of adding two numbers $x + y \bmod n$:
- Cost of multiplying two numbers $x \cdot y \bmod n$:
  - It's like multiplying degree $O(\log n)$ polynomials
    → use FFT to compute $z = x \cdot y$

Running Time

Cost of exponentiation $x^d \bmod n$:

- Can be done using $O(\log d)$ multiplications
- Base-2 representation of d: $d = \sum_{i=0}^{\lfloor \log d \rfloor} d_i 2^i$
- **Fast exponentiation:**
  1. $y := 1$;
  2. **for** $i := \lfloor \log d \rfloor$ **to** 0 **do**
  3. $\quad y := y^2 \bmod n$;
  4. $\quad$ **if** $d_i = 1$ **then** $y := y \cdot x \bmod n$;
  5. **return** y;
- **Example:** $d = 22 = 10110_2$

Running Time

Theorem: One iteration of the Miller-Rabin test can be implemented with running time $O(\log^2 n \cdot \log\log n \cdot \log\log\log n)$.

if n is even **then return** $(n = 2)$
compute s, d such that $n - 1 = 2^s d$;
choose $a \in \{2, \dots, n-2\}$ uniformly at random;
$x := a^d \bmod n$;
if $x = 1$ **or** $x = n - 1$ **then return probably prime;**
for $r := 1$ **to** $s - 1$ **do**
    $x := x^2 \bmod n$;
    if $x = n - 1$ **then return probably prime;**
return composite;

Deterministic Primality Test

- If a conjecture called the generalized Riemann hypothesis (GRH) is true, the Miller-Rabin test can be turned into a polynomial-time, deterministic algorithm

  → It is then sufficient to try all $a \in \{1, \dots, O(\log^2 n)\}$

- It has long not been proven whether a deterministic, polynomial-time algorithm exists

- In 2002, Agrawal, Kayal, and Saxena gave an $\tilde{O}(\log^{12} n)$-time deterministic algorithm
  - Has been improved to $\tilde{O}(\log^6 n)$

- In practice, the randomized Miller-Rabin test is still the fastest algorithm