AAC Project Report

Eklavya Sharma (2014A7PS0130P)

26 November 2017

Abstract

This document explores the problem of primality testing. It includes some exploratory analysis of the AKS algorithm and a comparison of randomized compositeness-proving algorithms.

1 Introduction

This report explores the problem of primality testing. Given a positive integer n, the task is to determine whether n is prime or composite.

The Agarwal-Kayal-Saxena (AKS) algorithm [1] is the first general deterministic unconditional polynomial-time algorithm for primality testing. But this algorithm is very slow in practice. H. W. Lenstra Jr. and Carl Pomerance have developed a more efficient variant of AKS. But even that algorithm has time complexity $\tilde{O}(log^6 n)$ [3], which is very slow in practice.

I tried to devise a faster variant of the AKS algorithm, but I couldn't. This report includes that exploratory analysis.

I then tried to study randomized primality-proving algorithms, like the ECPP algorithm [2], and quasi-polynomial-time algorithms like the Adleman-Pomerance-Rumely (APR) primality test. But I was unable to do so due to lack of mathematical background (elliptic curves) required to study them.

2 Analysis of AKS

This is the AKS algorithm:

1. If $n = a^b$ for $a \in \mathbb{N}$ and $b > 1$, return composite.
2. Find the smallest r such that order of n in $\mathbb{Z}_r^* > \log^2 n$.
3. If $1 < \gcd(a, n) < n$ for some $a \leq r$, return composite.

4. If $n \leq r$, return prime.

5. For a from 1 to $l = \lfloor \sqrt{\phi(r)}) \log n \rfloor$,
   if $(x+a)^n \not\equiv x^n + a \pmod{x^r - 1, n}$, return composite.

6. Return prime.

Authors of the AKS test have proved that $r < \log^5 n$.

Let $M(n)$ be the time taken to multiply 2 n-bit numbers. The asymptotically best-known algorithm is the Schonage-Strassen algorithm. It takes $O(n \log(n) \log(\log(n))) \subseteq \tilde{O}(n)$ time. Division of two n-bit numbers can be done in $\tilde{O}(M(n))$ time. Time taken to multiply two degree d polynomials modulo n is $\tilde{O}(d^2 M(\log n))$ by simple multiplication and $\tilde{O}(dM(\log n))$ using Fast-Fourier Transform (FFT).

These are the running times of various steps of the AKS algorithm:

1. Write as a^b: $O(\log^2 n \log\log n M(\log n)) \subseteq \tilde{O}(\log^3 n)$

2. Find r: $O(r \log^2 n M(\log r)) \subseteq \tilde{O}(\log^7 n)$

3. Calculate gcds: $O(r \log r M(\log r)) \subseteq \tilde{O}(\log^5 n)$

4. Check if $n \leq r$: $O(\log r) \subseteq O(\log\log n)$

5. Evaluate polynomials: $O(r^{\frac{3}{2}} \log^2 n M(\log n)) \subseteq \tilde{O}(log^{\frac{21}{2}} n)$

In my variant, I have replaced $\log^2 n$ by $\log^w n$ in step 2 of the AKS algorithm and set $l = \lfloor \sqrt{\phi(r)} \log^{\frac{w}{2}} n \rfloor$ in step 5 of the algorithm. I then tried to explore what values of w can be used for the AKS algorithm. If a value of w less than 2 can be used, then the algorithm will find smaller values of r. That will reduce the running time of the AKS algorithm, as the running time depends on r.

Let p be a prime factor of n. Let $I = \{(\frac{n}{p})^i p^j | i, j \geq 0\}$ and $P = \{\prod_{a=0}^{l}(x+a)^{e_a} | e_a \geq 0\}$. Let $G = I \bmod r$ and $G^* = P \bmod (p, h(x))$, where $h(x)$ is an irreducible factor of $x^r - 1$ modulo p.

Let $|G| = t$. Since n is in G and order of n in $\mathbb{Z}_r^*$ is greater than $\log^w n$, $t \geq \lfloor \log^w n \rfloor + 1$. Since G is a subset of $\mathbb{Z}_r^*$, $t \leq \phi(r)$.

According to [1], if AKS algorithm returns 'prime' then $|G^*| \leq \binom{t+l}{t-1}$ and if n is actually prime then $|G^*| \leq n^{\sqrt{t}}$. Therefore, we need to prove that $\binom{t+l}{t-1} > n^{\sqrt{t}}$ to prove that the AKS algorithm is correct.

Let $q = \lfloor \sqrt{t} \log^{\frac{w}{2}} n \rfloor$.

$$
\begin{aligned}
&t > \log^w n \\
&\Rightarrow \sqrt{t} > \log^{\frac{w}{2}} n \\
&\Rightarrow t > \sqrt{t}\log^{\frac{w}{2}} n \\
&\Rightarrow t \geq \lfloor \sqrt{t}\log^{\frac{w}{2}} n \rfloor + 1 = q + 1
\end{aligned}
$$

$$
\begin{aligned}
&\text{Let } n \geq 2^{2^{\frac{1}{w}}} && (\because \text{ we only care about large values of } n) \\
&\Rightarrow \log n \geq 2^{\frac{1}{w}} \\
&\Rightarrow \log^w n \geq 2 \\
&\Rightarrow \lfloor \log^w n \rfloor \geq 2
\end{aligned}
$$

$$
\begin{aligned}
q &= \lfloor \sqrt{t}\log^{\frac{w}{2}} n \rfloor \\
&\geq \lfloor \log^w n \rfloor \geq 2
\end{aligned}
$$

$$
\begin{aligned}
&\binom{2q+1}{q} \\
&= \frac{1}{2}\binom{2q+2}{q+1} \\
&\geq \frac{2}{e^{\frac{1}{6}}\sqrt{\pi}}\frac{4^q}{\sqrt{q+1}} && \text{(Using improved Stirling's approximation [6])} \\
&= 2^{q+1}\frac{2^q}{e^{\frac{1}{6}}\sqrt{\pi}\sqrt{q+1}} \\
&> 2^{q+1} && (\because q \geq 2)
\end{aligned}
$$

$$
\begin{aligned}
|G^*| &\geq \binom{t+l}{t-1} \\
&= \binom{t+l}{l+1} \\
&\geq \binom{q+l+1}{l+1} \\
&= \binom{q+l+1}{q} \\
&\geq \binom{2q+1}{q} && (l = \lfloor \sqrt{\phi(r)} \log^{\frac{w}{2}} n \rfloor \geq \lfloor \sqrt{t} \log^{\frac{w}{2}} n \rfloor = q) \\
&> 2^{q+1} \\
&= 2^{\lfloor \sqrt{t} \log^{\frac{w}{2}} n \rfloor + 1} \\
&> 2^{\sqrt{t} \log^{\frac{w}{2}} n} \\
&= n^{\sqrt{t}} 2^{\sqrt{t}(\log^{\frac{w}{2}} n - \log n)}
\end{aligned}
$$

$$
\begin{aligned}
&|G^*| > n^{\sqrt{t}} \\
&\Rightarrow n^{\sqrt{t}} 2^{\sqrt{t}(\log^{\frac{w}{2}} n - \log n)} \geq n^{\sqrt{t}} \\
&\Rightarrow 2^{\sqrt{t}(\log^{\frac{w}{2}} n - \log n)} \geq 1 \\
&\Rightarrow \sqrt{t}(\log^{\frac{w}{2}} n - \log n) \geq 0 \\
&\Rightarrow \log^{\frac{w}{2}} n \geq \log n \\
&\Rightarrow \log^{\frac{w}{2}-1} n \geq 1 \\
&\Rightarrow \left(\frac{w}{2} - 1\right) \log \log n \geq 0 \\
&\Rightarrow \frac{w}{2} \geq 1 \\
&\Rightarrow w \geq 2
\end{aligned}
$$

Since $w \geq 2$, $n > 2^{2^{\frac{1}{w}}}$ is true for $n > 2$, which means that our analysis is correct for $n > 2$.

Since we want w to be as small as possible, the best value of w is 2. This makes my variant equivalent to the AKS algorithm. Therefore, we fail to improve the running time of the AKS algorithm.

Perhaps using a value of l other than $\lfloor \sqrt{\phi(r)} \log^{\frac{w}{2}} n \rfloor$ could have given better results, but I didn't find that amenable to mathematical analysis.

3 Comparison of Compositeness-Proving Algorithms

3.1 Probable primes

All such algorithms that we discuss here either declare n to be 'composite' or 'probably prime'. A composite probable prime is called a pseudoprime.

We assume here that n is odd. Let $n - 1 = 2^s d$.

With respect to a base a coprime to n:

- n is a Fermat probable prime iff $a^{n-1} \equiv 1 \pmod{n}$.
- n is an Euler-Jacobi probable prime iff $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$.
- n is a strong probable prime iff $a^d \equiv 1 \pmod{n}$ or $a^{d2^r} \equiv -1 \pmod{n}$ for some $0 \leq r < s$.

These conditions can be used as tests of compositeness by first randomly choosing a value a, checking whether it is coprime to n and then checking whether n is a probable prime with respect to base a. The Fermat primality test tests for Fermat probable primes. The Solovay-Strassen test tests for Euler-Jacobi probable primes. The Miller-Rabin test tests for strong probable primes.

3.2 Miller-Rabin vs Solovay-Strassen

Let's define the 'primality-strength' of n as

$$\frac{|\{a | gcd(a, n) = 1 \text{ and } n \text{ is a probable prime for base } a\}|}{n}$$

Euler-Jacobi-primality-strength and Strong-primality-strength are defined analogously. For a compositeness test which uses the concept of probable primes as defined above, the error probability of the algorithm for n equals the primality-strength of n when n is composite. Solovay and Strassen claimed [7] that a composite number has a Euler-Jacobi-primality-strength less than $\frac{1}{2}$. Miller and Rabbin claimed [5] that a composite number has a Strong-primality-strength less than $\frac{1}{4}$.

The Miller-Rabin algorithm requires one more multiplication than the Solovay-Strassen algorithm for calculating powers of a. But the Solovay-Strassen algorithm additionally involves calculating the Jacobi symbol $\left(\frac{a}{n}\right)$, which takes $O(\log \min(a, n) M(\log \min(a, n)))$ time. This implies that both these algorithms have comparable running times. So given their error bounds, the Miller-Rabin algorithm seems better.

However, the error bounds may not be tight for most numbers. The average-case error probability is given by the average value of the primality-strength, and the error bounds may not be a good indication of that.

Pomerance et al prove in [4] that the strong-primality-strength of n is less than or equal to the Euler-Jacobi-primality-strength of n for all composite n. This result makes the Miller-Rabin algorithm a clear winner against the Solovay-Strassen algorithm.

3.3 The Baillie-PSW Algorithm

The Baillie-PSW [4] test is a compositeness-proving heuristic algorithm which works by running 2 compositeness tests. It returns 'composite' if one of these tests returns 'composite' and returns 'probably prime' otherwise. The first test is the Miller-Rabin test with base 2 and the second test is the Lucas test with base 2. This makes the Baillie-PSW test a deterministic algorithm. In some variations of Baillie-PSW, a stronger variant of the Lucas test is used or different (either fixed or randomly-selected) bases are used.

It is not known whether the Baillie-PSW algorithm always returns the correct result. However, there are no known counterexamples (i.e. the set of strong-pseudoprimes and lucas-pseudoprimes have no known overlap), which is what makes Baillie-PSW test a good choice in practice.

4 Conclusion

The AKS algorithm is a deterministic polynomial-time algorithm for primality testing. It is however so slow that it is not used in practice. Therefore, almost all primality tests used in practice are randomized. The Solovay-Strassen algorithm was one of the first randomized algorithms for primality-testing. But it has been superseded by the Miller-Rabin algorithm and the Baillie-PSW algorithm, which are now very popular algorithms for primality testing.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[2] A Oliver L Atkin and François Morain. Elliptic curves and primality proving. *Mathematics of computation*, 61(203):29–68, 1993.

[3] Hendrik W Lenstra Jr and Carl Pomerance. Primality testing with gaussian periods. In *FSTTCS*, page 1, 2002.

[4] Carl Pomerance, John L Selfridge, and Samuel S Wagstaff. The pseudoprimes to 25×10^9. *Mathematics of Computation*, 35(151):1003–1026, 1980.

[5] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

[6] Herbert Robbins. A remark on stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.

[7] Robert Solovay and Volker Strassen. A fast monte-carlo test for primality. *SIAM journal on Computing*, 6(1):84–85, 1977.