Advanced Algorithms and Complexity
Course Project Report

Eklavya Sharma (2014A7PS0130P)

26 November 2017

Abstract

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

Contents

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 despite being a polynomial-time algorithm, it is very slow.

In this report I have attempted to to devise a faster variant of the AKS algorithm. I did this by parametrizing the AKS test, i.e. I replaced a constant in the algorithm by a parameter while preserving the correctness of the algorithm. Then I tried to find a value of the parameter which would yield the best time complexity. I proved that the constant in the original algorithm was the best value of that parameter, so I failed to improve the algorithm. This report describes my parametrized algorithm.

I then studied randomized compositeness-proving algorithms. I have compared the Miller-Rabin algorithm [3], the Solovay-Strassen algorithm [4] and the Baillie-PSW primality test [2].

2 Analysis of AKS

This is a parametrized variant of the AKS algorithm, where w and n_0 are the parameters.

0. If $n \leq n_0$, return output using a lookup table.
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 $\mathbb{Z}_r^* > \log^w n$.
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 $l = \lfloor \sqrt{\phi(r)} \log^{\frac{w}{2}} n \rfloor$,
   if $(x + a)^n \not\equiv x^n + a \pmod{x^r - 1, n}$, return composite.
6. Return prime.

The original AKS algorithm uses $w = 2$ and $n_0 = 1$.

Suppose this algorithm is correct for a value of w less than 2. Then in step 2, a smaller r can be found. The worst-case running time of the algorithm is dominated by step 5. Reducing the value of r will reduce the running time of steps 2, 3 and 5 and therefore reduce the worst-case running time of the entire algorithm.

I will now try to find constraints on the value of w imposed by the correctness of this algorithm.

2.1 Preliminary arguments

If n is prime, steps 0, 1 and 3 can never return 'composite'. Step 5 will never return composite due to the follow lemma (proof is in [1]).

Lemma 2.1. *For $n \geq 2$ and $\gcd(a,n) = 1$,*

$$n \textit{ is prime} \iff (x+a)^n \equiv x^n + a \pmod{n}$$

If steps 0, 1 and 3 return 'composite', it is easy to see that n is indeed composite. Therefore, to prove correctness of this algorithm, the only thing left is to prove that when step 5 does not return 'composite', n is indeed prime.

$\gcd(n,r) = 1$ because otherwise step 3 or 4 would decide primality of n.

Let $o_r(n)$ denote the order of n in $\mathbb{Z}_r^*$. Since $o_r(n) > 1$, there must exist a prime divisor p of n such that $o_r(p) > 1$. $p > r$, since otherwise step 3 or 4 would decide about the primality of n. $r > \log^w n$, since $o_r(n) > \log^w n$. Since $\gcd(n,r) = 1$, $p, n \in \mathbb{Z}_r^*$. The numbers p and r will be fixed throughout the discussion of the AKS algorithm.

2.2 Important lemmas from 'Primes is in P'

Let us define the following sets:

- $I = \left\{ \left(\frac{n}{p}\right)^i p^j | i, j \geq 0 \right\}$.
- $P = \left\{ \prod_{a=0}^{l} (x+a)^{e_a} | e_a \geq 0 \right\}$, where $l < p$.
- $G = I \bmod r$. G is a subgroup of $\mathbb{Z}_r^*$ generated by n and p, since $\gcd(p,r) = 1$ and $\gcd(n,r) = 1$.
- $G^* = P \bmod (h(x), p)$ where $h(x)$ is an irreducible factor of $x^r - 1$ modulo p. G^* is a subgroup of $F = \mathbb{F}_p/h(x)$ generated by $x, x+1, \ldots, x+l$. $x, x+1, \ldots, x+l$ are distinct because $l < p$.

Let $|G| = t$. The following bounds on $|G^*|$ are proven in [1].

- $|G^*| \geq \binom{t+l}{t-1}$.
- When n is not a power of p, $|G^*| \leq n^{\sqrt{t}}$.

These bounds hold under this condition:

$$\forall a \in [1, l], (x+a)^n \equiv x^n + a \pmod{x^r - 1, n}$$

2.3 Framework for correctness of my algorithm

To prove the correctness of our algorithm, we must prove that when step 5 does not output 'composite', n is indeed prime.

When step 5 does not output 'composite', we have $\forall a \in [1, l], (x+a)^n \equiv x^n + a \pmod{x^r - 1, n}$ for $l = \lfloor \sqrt{\phi(r)} \log^{\frac{w}{2}} n \rfloor$. This is the prerequisite condition for bounds on $|G^*|$.

We must first prove that this value of l is appropriate. As per the definition of P, l should be less than p. $l = \lfloor \sqrt{\phi(r)} \log^{\frac{w}{2}} n \rfloor \leq \lfloor \sqrt{r}\sqrt{r} \rfloor = r < p$, so we are good to go.

In the original paper by AKS [1], the authors used the first bound on $|G^*|$ to prove that $|G^*| > n^{\sqrt{t}}$. According to the second bound, this implies that n is a power of p. Since we have ensured in step 1 of the algorithm that n is not of the form a^b where $b \geq 2$, we can conclude that n is prime. This proves the correctness of the AKS algorithm.

I'm going to do the same thing for my variant of AKS, but my proof of $|G^*| > n^{\sqrt{t}}$ will be different because of a different value of l and different bounds on t.

2.4 Lower bound on $|G^*|$

Since G is generated by p and n and $o_r(n) > \log^w n$, $t \geq \lfloor \log^w n \rfloor + 1$. Since G is a subset of $\mathbb{Z}_r^*$, $t \leq \phi(r)$. Therefore, $\lfloor \log^w n \rfloor + 1 \leq t \leq \phi(r)$.

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

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

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

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

$\binom{2q+1}{q} > 2^{q+1}$ when $q \geq 2$. This can be proved easily using mathematical induction. $\left(\text{Hint: } \frac{\binom{2q+3}{q+1}}{2^{q+2}} = \frac{\binom{2q+1}{q}}{2^{q+1}} \left(1 + \frac{q+1}{q+2}\right)\right)$

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

For $|G^*| > n^{\sqrt{t}}$, we have to choose w so that $2^{\sqrt{t}(\log^{\frac{w}{w}} n - \log n)} \geq 1$.

$$\Rightarrow 2^{\sqrt{t}(\log^{\frac{w}{2}} n - \log n)} \geq 1$$
$$\Rightarrow \sqrt{t}(\log^{\frac{w}{2}} n - \log n) \geq 0$$
$$\Rightarrow \log^{\frac{w}{2}} n \geq \log n$$
$$\Rightarrow \log^{\frac{w}{2}-1} n \geq 1$$
$$\Rightarrow \left(\frac{w}{2} - 1\right) \log\log n \geq 0$$
$$\Rightarrow w \geq 2 \qquad (\log\log n > 0 \because n > 2)$$

Since $w \geq 2$ for the algorithm to be correct, the attempt to improve the running time of the AKS algorithm failed.

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 [4] that a composite number has a Euler-Jacobi-primality-strength less than $\frac{1}{2}$. Miller and Rabbin claimed [3] 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 [2] 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 [2] 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] Carl Pomerance, John L Selfridge, and Samuel S Wagstaff. The pseudoprimes to 25×10^9. *Mathematics of Computation*, 35(151):1003–1026, 1980.

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

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