AKS Primality Test: What It Is and Why It Is Important

Senapati Sang Diwangkara 13516107
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
diwangs@s.itb.ac.id

***Abstract*—In this paper, the author tries to explain the inner working of the AKS Primality Testing Algorithm and discusses its significance over the theoretical computer science in a laymen's term. Primality testing is an algorithm that determines whether a given number is prime or composite. The problem was known to be of class non-deterministic polynomial, but the AKS Primality Test has disproven that conjecture. This has a very big impact on the problem of computational complexity: Is P = NP?**

***Keywords*—complexity, deterministic, polynomial, prime, testing.**

I. Introduction

Prime number is one of, if not the most, challenging type of number to deal with. Despite of its simple definition – any natural number that has exactly 2 factors: 1 and itself – prime numbers are hard to tame if the given number is enormous. Testing the primality of a number is a simple task if the number is relatively small, but what if we want to test a number that has 22 338 618 digits, like $2^{74\,207\,281} - 1$?

Humankind's inability to test the primality of a gigantic number is one of the foundation of modern day cryptography, where 2 monstrous prime numbers are used to generate a private and public key pair in RSA algorithm.

A. Trial Division Algorithm

An intuitive algorithm to test the primality of a number that might come to your mind is to divide the number with every natural number between 1 and itself (exclusive) and check if the result is not a fraction, since, by definition, a prime number is a number that can't be evenly divided by every natural number in that range. This naïve algorithm is usually known by the name Trial Division.

This Trial Division algorithm has an asymptotic time complexity of $O(k^{\lfloor \log_k(p) \rfloor})$ where k is the base of the number (decimal is 10 and binary is 2) and p is the test subject. This algorithm grows exponentially, which means that we can't solve the problem in a reasonable amount of time given a sufficiently large input. We need something better.

Many people have tried to perfect the algorithm. One modification is to limit the search range to $\sqrt{p}$ where p is the number that is being tested.

Proof:

- Suppose p is composite. It has, at least, 2 factors. Let the factor be a and b. $p = a \cdot b$
- If a and b both are greater than $\sqrt{p}$, then $a \cdot b > p$, a contradiction.
- Then one of the factors (either a or b) must be less than or equal to $\sqrt{p}$ so that the multiplication equals to p.

The other modification is to only use odd number and 2 as the divisor, because every even number are evenly divided by 2.

The generalization of this concept is to only use prime divisor to test the number, since every composite number has its unique prime factor, as provided by the Fundamental Theory of Arithmetic. This method is usually called the Sieve of Eratosthenes.

But despite of the improvements, even with all of those modifications combined, the algorithm still runs with an exponential time complexity.

B. Fermat's Little Theorem

However, over the course of the 20th century, the quest to discover an efficient primality testing algorithm has primarily been focused on a theorem: Fermat's Little Theorem.

This theorem states that if p is prime and a and p are coprime then the following congruence holds.

$$a^p \equiv a \ (mod\ p)$$

Ideally, we can use this theorem to examine the congruence for various p and a and determine the primality of p. However, this theory has a fatal flaw: the converse of this theorem doesn't hold. There also exists composite number that satisfies the theorem (which will be referred to as 'Fermat's Pseudoprime', such as 341 for $a = 2$) which render the test indeterministic: if a natural number p satisfies this congruence, it is *probably* prime. So why is this theorem useful? Because it is relatively efficient: it has a polylogarithmic asymptotic time complexity.

C. Miller-Rabin Primality Testing Algorithm

Many have tried to build a primality testing algorithm with Fermat's Little Theorem as its foundation.

One example is the Miller-Rabbin Algorithm. This algorithm takes the modified version of the algorithm (where the left-hand side of the congruence is a^{p-1} and the right-hand side of the congruence is 1) and repeatedly divides the exponent of a by 2 to eventually get an odd number. Because of the fact that $x^2 \equiv 1 \ (mod\ p)$ is equivalent to $x \equiv \pm 1 \ (mod\ p)$, if the congruence doesn't hold, then p is composite. This step cuts a significant amount of processing in the Fermat's Little Theorem but unfortunately, it doesn't solve the probabilistic nature of its foundational theorem: it's still indeterministic.

D. AKS Primality Testing Algorithm

In 2002 however, 3 computer science researchers from Indian Institute of Technology Kanpur, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena have successfully created a deterministic algorithm based on the Fermat's Little Theorem while maintaining its polylogarithmic time complexity. They called it the AKS primality testing algorithm.

II. Prerequisites

Here are a few prerequisites that is not being taught in ITB's Discrete Mathematics course:

A. Multiplicative Order

Given an integer a and a natural number n, the multiplicative order of a modulo n is the smallest natural number k such that:

$$a^k \equiv 1 \ (mod\ n)$$

The order of a modulo n is usually written as $O_n(a)$

B. Euler's Totient Function

The Euler's Totient Function $\phi(n)$ counts the natural number from 1 to n (inclusive) that are relatively prime to n.

C. Soft-O Notation

The $\tilde{O}(g(n))$ notation is a variant of the Big-O notation which is a shorthand for $O(g(n) \cdot \log_2(g(n))^k)$. It ignores the logarithmic factor because it's often superseded by super-logarithmic function.

III. The Algorithm

For an integer $p > 1$

1. **If p is a perfect power, output composite**
2. **Find the smallest r such that $O_r(p) > \log_2(p)^2$**
3. **If $1 < \gcd(b, p) < p$ for some $b \leq r$, output composite**
4. **If $p \leq r$ output prime.**
5. **For $1 < a < \lfloor \sqrt{\phi(r)} \cdot \log_2(p) \rfloor$, if $(x + a)^p \not\equiv x^p + a \ (mod\ p, x^r - 1)$, output composite**
6. **Output prime.**

The core essence (and also the most laborious step) of AKS algorithm is the fifth step. It is a direct generalization of Fermat's Little Theorem, but unlike the former, the relation between the congruence and the primality of p holds conversely. So, we can use it to examine the primality of a number:

$$(x - a)^p \equiv x^p - a \ (mod\ p)$$
If and only if p is prime and a is coprime to p

Where x is an indeterminate variable.

Proof:
Suppose we have expanded the polynomial.

- Because p is prime, then the constant terms cancel out: they add to $a^p - a \ (mod\ p)$ and a is coprime to p, which is congruent to 0 by Fermat's Little Theorem. The a^p term will always be positive because there is no even prime number.
- The x^p terms also cancel out (self-explanatory)
- The rest of the polynomial term will have a coefficient of $\binom{p}{k} \cdot (-a)^{p-k}$ for the x^k term.
- Since $\frac{p!}{k! \cdot (p-k)!}$ will always have p as the one of the denominator and p is prime, no numerator can cancel it out (by definition). This part is what makes the congruence holds conversely, unlike Fermat's Little Theorem. So, the whole polynomial is divisible by p: the congruence holds.
- Suppose p is a composite (for refutation's sake). Let f be one of the factor of p. f will always be in the in the x's rank (x^f), because the factor of a number is always less of the number itself. Let m be the largest power of f that divides p. In the coefficient of the x^{f^m} term, the combination's numerator will always have enough f to cancel the denominator including p. So, the whole polynomial is not divisible by p, a contradiction.

Despite of its determinism, however, the congruence no longer has a polynomial complexity. To solve this, AKS's idea is to check the congruence on a less restrictive condition, using Ring Theory:

$$(x + a)^p \equiv x^p + a \ (mod\ x^r - 1, n)$$

This equation basically means that $(x + a)^p - (x^p - a)$ can be written as a linear combination of $x^r - 1$ and n. The point of doing this is to reduce the left-hand side of the congruence so that it can be reduced to a polynomial complexity. With this condition, the 5th step is now polynomial-time.

This step gives the algorithm a polynomial-time complexity, but it loses its determinism. To find r such that the 5th step can run as efficient as possible and give the determinism back, is described by the 2nd through the 4th step. It is essentially a Trial Division algorithm, but limited to an r and this r is going to be the one that makes the Trial Division a polylogarithmic runtime and gives the determinism back.

It is very difficult to understand and explain the 2nd step of the algorithm intuitively without a deep understanding of Ring Theory, so the author can't provide a very good rationalization as to why this step is taken.

The theorem is:

> For some r coprime to p, if the multiplicative order of p modulo r is greater than $\log_2(p)^2$ and if the congruence (not incongruence) on the 5th step is satisfied for some $1 < a < O(r \cdot \log_2(p)^{O(1)})$ then p is either a prime or the power of a prime [3]

Hence, to only let through prime numbers and not the power of a prime number, 1st step was added to the algorithm. Also, r is guaranteed to be found $< \tilde{O}(\log_2(p)^5)$

The overall complexity of the algorithm is $\tilde{O}(\log_2(p)^{10.5})$ bit operations. [4]

- The most efficient power test algorithm runs at $\tilde{O}(\log_2(p)^3)$ bit operations
- The lower bound of the second step takes $\tilde{O}(\log_2(p)^2)$, because of r's warranty, the second step takes $\tilde{O}(\log_2(p)^7)$ bit operations
- Finding the GCD with a Euclidean algorithm takes $\tilde{O}(\log_2(p)^2)$, because of r's warranty, the third step takes $\tilde{O}(\log_2(p)^7)$ bit operations
- The expansion of the polynomial is done $O(\log(p))$ times. Fast modular exponentiation algorithm takes $\tilde{O}(\log_2(p)^7)$. It loops over the square root of the totient function of r, because of r's warranty, the loop takes $\tilde{O}(\log_2(p)^{2.5})$. Hence, the fifth step takes $\tilde{O}(\log_2(p)^{10.5})$

Hence, overall the algorithm takes $\tilde{O}(\log_2(p)^{10.5})$ which is indeed polylogarithmic.

This algorithm is the original and unoptimized version of the AKS Primality Test Algorithm. There exists, however, a version of this algorithm, modified by H. W. Lenstra,Jr.and Carl Pomerance that runs at $\tilde{O}(\log_2(p)^6)$ [6]

IV. Implementation and Examples

A. $p = 31$

First step

> $for\ b = 2\ to\ \lceil \log_2(p) \rceil\ do$
> $\quad if\ p^{\frac{1}{b}}\ is\ integer, return\ composite$
>
> Because 31 is not a perfect square, the algorithm continues

Second step

> $max\_k = \lfloor \log_2(p)^2 \rfloor$
> $next\_n = true$
> $n = 1$
> $while\ next\_n\ do$
> $\quad n = n + 1$
> $\quad next\_n = false$
> $\quad while\ k \leq \max\_r\ and\ not\ next\_r\ do$
> $\quad\quad next\_r = p^k\ mod\ r == 1$ or 0
>
> r is now 29

Third step

> $for\ a = 2\ to\ r\ do$
> $\quad if\ 1 < \gcd(a, p) < p, return\ composite$
>
> Because 31 is coprime to any natural numbers less than or equal to 29, the algorithm continues

Fourth step

> $if\ p \leq r\ return\ prime$
>
> Because 29 is smaller than 31, the algorithm continues

Fifth step

> $for\ a = 1\ to \left\lfloor \sqrt{\phi(r)} \cdot \log_2(p) \right\rfloor\ do$
> $\quad re = ((x + a)^p - (x^p + a))\ mod\ (x^r - 1)$
> $\quad if\ re\ mod\ p \neq 0\ return\ compoite$
>
> For every $a \leq 26$, $a^{31} - a\ (mod\ 31)$ always equal to zero, so the algorithm continues.

Sixth step

> $return\ prime$
>
> If a given number reached this point, it must be prime. Thus, 31 must be **prime**

B. $p = 33$

First step

$for\ b = 2\ to\ \lceil \log_2(p) \rceil\ do$
$\quad if\ p^{\frac{1}{b}}\ is\ integer, return\ composite$

Because 31 is not a perfect square, the algorithm continues

Second step

$max\_k = \lfloor \log_2(p)^2 \rfloor$
$next\_n = true$
$n = 1$
$while\ next\_n\ do$
$\quad n = n + 1$
$\quad next\_n = false$
$\quad while\ k \leq \max\_r\ and\ not\ next\_r\ do$
$\quad\quad next\_r = p^k\ mod\ r == 1$ or 0
$\quad\quad k = k + 1$

r is now 6

Third step

$for\ a = 2\ to\ r\ do$
$\quad if\ 1 < \gcd(a, p) < p, return\ composite$

$a = 3$ satisfies the criterion, so 33 must be **composite**

C. $p = 13$

First step

$for\ b = 2\ to\ \lceil \log_2(p) \rceil\ do$
$\quad if\ p^{\frac{1}{b}}\ is\ integer, return\ composite$

Because 13 is not a perfect square, the algorithm continues

Second step

$max\_k = \lfloor \log_2(p)^2 \rfloor$
$next\_n = true$
$n = 1$
$while\ next\_n\ do$
$\quad n = n + 1$
$\quad next\_n = false$
$\quad while\ k \leq \max\_r\ and\ not\ next\_r\ do$
$\quad\quad next\_r = p^k\ mod\ r == 1$ or 0

r is now 19

Third step

$for\ a = 2\ to\ r\ do$
$\quad if\ 1 < \gcd(a, p) < p, return\ composite$

Because 13 is coprime to any natural numbers less than or equal to 19, the algorithm continues

Fourth step

$if\ p \leq r\ return\ prime$

Because 13 is smaller than 19, 13 must be **prime**

D. $p = 343$

First step

$for\ b = 2\ to\ \lceil \log_2(p) \rceil\ do$
$\quad if\ p^{\frac{1}{b}}\ is\ integer, return\ composite$

$b = 3$ satisfies the criterion, thus 343 must be **composite**

IV. Significance

The AKS primality test is not the only polynomial-time modern prime testing algorithm that exists. There's the Miller-Rabin primality test which has the complexity of $O(i \cdot \log_k(x)^3)$ where i is the number of iteration, there's the Solovay-Strassen primality test which also has the complexity of $O(i \cdot \log_k(x)^3)$, and there are many more polynomial-time algorithms like the Miller test (the origin of Miller-Rabin test), Lucas-Lehmer test, and Baillie-PSW primality test.

However, all of those polynomial time algorithms have their own caveat. Lucas-Lehmer test only works for Mersenne numbers, Miller-Rabin test is an indeterministic algorithm (it can deterministically tell if a number is composite, but it can't tell if a number is prime with certainty), and the Miller test is a general and deterministic test, but it relies on the Extended Riemann Hypothesis, which is unproven to this date.

The AKS algorithm however, is *the only* primality testing algorithm to date that is general, polynomial, deterministic, and doesn't rely on some unproven hypothesis. Although in practice, this algorithm is rarely used, because the number we're testing is relatively small (limited by computer's memory convention). A probabilistic or an exponential algorithm is far more superior in that range. But from a theoretical perspective, this is a huge breakthrough because it proves that determining the primality of a number is not that hard of a problem after all: it's a P class problem, not an NP one.

In computational complexity theory, there is a distinction between a P class problem and an NP class problem. P stands for polynomial: problems that can be solved in a reasonable amount of time. NP stands for non-deterministic polynomial: problems that can be verified in a reasonable amount of time. However, there are many problems that initially considered to be NP that are found to be P.

This leads to the (literally) million-dollar question: is NP equals to P? Does every problem that is easy to verify, easy to solve?

This is a very broad question to answer. That's partly the reason of why this question hasn't been correctly answered yet. And this problem has a very serious implication in the real world, such as breaking modern cryptography, which has a serious impact on many fields, including banking, economics, and politics. Majority of computer scientists, however, think that this equivalence is not true (that $P \neq NP$).

Thus, the discovery of this AKS primality testing algorithm provides one more support for this question, as prime finding is known to be an NP problem but is disproven by this algorithm. It reminds us to remain critical and open to the new facts and knowledge as they continue to develop and tested.

V. Conclusion

The AKS primality testing algorithm is the algorithm that proves that the problem of determining whether a number is prime or composite is a problem of class P, which previously known to be of class NP.

VI. Appendix

Figure 1: Photograph of Mr. Saxena, Mr. Kayal, and Mr. Agrawal. Source: http://www.ams.org/samplings/feature-column/fcarc-primes6

VII. Acknowledgment

The author would like to thank Mr. Tim Berners Lee for inventing the world wide web, so that information searching can be done with such an ease. The author would also like to thank the services that's available on the world wide web particularly Google, Quora, and LinkedIn. The author would also praise Mr. Terrence Tao who has such a helpful and read-worthy blog of mathematics.

REFERENCES

[1] M. Agrawal, N. Kayal, N. Saxena, "PRIMES is in P". https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf Accessed on December 2nd , 2017

[2] S. Bandyopadhyay, "PRIMALITY TESTING A Journey from Fermat to AKS". http://www.cmi.ac.in/~shreejit/primality.pdf Accessed on December 3rd , 2017

[3] T. Tao, "The AKS Primality Test", https://terrytao.wordpress.com/2009/08/11/the-aks-primality-test/ Accessed on December 3rd , 2017

[4] P. Bhatnagar, "Introduction to The AKS Primality Test", https://www.slideshare.net/PranshuBhatnagar/introduction-to-the-aks-primality-test Accessed on December 3rd, 2017

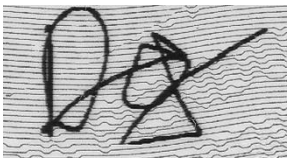
[5] https://www.quora.com/How-can-the-AKS-primality-testing-algorithm-be-explained-in-laymen%E2%80%99s-terms Accessed on December 3rd, 2017

[6] H. W. Lenstra Jr., C. Pomerance, "Primality Testing with Gaussian Period", https://www.math.dartmouth.edu/~carlp/PDF/complexity12.pdf Accessed on December 3rd, 2017

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2017



Senapati Sang Diwangkara 13516107