GE6001: Scientific Writing, Norms and Ethics May 14, 2020

Notes 1 – AKS algorithm and prime testing in engineering

Instructor: *Guoqiang Li* Scribes: *Jingyao Zeng Yuhan Li*

1 Abstraction

AKS is a deterministic primality test algorithm proposed by three Indian scientists Agrawal,Kayal and Saxena. This algorithm has a significant impact on the cryptology and is also regarded as a big step at the search area of the P and NP problems.
This notes will introduce the AKS algorithm from three aspects, including the detailed algorithm and its correctness, its time complexity analysis and the optimization of this algorithm. And in the last part of this notes, we also introduce some primality test algorithms that are widely used in the engineering area, and make some comparisons between these algorithms and the AKS algorithm.
Keywords: AKS algorithm, primality test

2 Preliminaries

AKS algorithm is based on the following identity for prime numbers which is a generalization of Fermat's Little Theorem.

Lemma 2.1. *Let $a \in Z$, $n \in N$, $n \geq 2$, and $(a, n) = 1$. Then n is prime if and only if*

$$(X + a)^n = X^n + a \ (mod \ n). \tag{1}$$

Proof. For $0 < i < n$, the coefficient of x^i in $((X + a)^n - (X^n + a^n))$ is $\binom{n}{i}a^{n-i}$.

Suppose n is prime. Then According to Fermat's Little Theorem we have $a^{n-1} = 1 \ (mod \ n)$, and $\binom{n}{i} = 0 \ (mod \ n)$, so $(X + a)^n = X^n + a \ (mod \ n)$.

Suppose n is composite. There must exist a prime q that is a factor of n and $q^k \parallel n$. Then q^k does not divide $\binom{n}{q}$ and is coprime to a^{n-q} and hence the coefficient of X^q is not zero (mod n). Thus $((X + a)^n - (X^n + a))$ is not identically zero over Z_n.(Z_n denotes the ring of numbers modulo n.) □

The above identity suggests a simple test for primality: given an input n, choose an a and test whether the congruence (1) is satisfied. However, this takes time $\Omega(n)$ because we need to evaluate n coefficients in the LHS in the worst case. A simple way to reduce the number of coefficients, which is also the way used by AKS algorithm, is to evaluate both sides of (1) modulo a polynomial of the form $X^r - 1$ for an appropriately chosen small r. In other words, test if the following quation is satisfied:

$$(X + a)^n = X^n + a \ (mod \ X^r - 1, n). \tag{2}$$

We will also need the following fact about the lcm of the first m numbers.(The proof of it is a little complicated and beyond author's acknowledgement level, so we omit the proof here and you can see, e.g.,

[5] for a proof.)

Lemma 2.2. *Let $LCM(m)$ denote the lcm of the first m numbers. For $m \geq 7$:*

$$LCM(m) \geq 2^m.$$

In the part of *time complexity analysis*, we use symbol $O^{\sim}(t(n))$ for $O(t(n) \cdot poly(\log t(n)))$, where $t(n)$ is any function of n. For example, $O^{\sim}(\log^k n) = O(\log^k n \cdot poly(\log \log n)) = O(\log^{(} k + \epsilon)n)$ for any $\epsilon > 0$. We use log for base 2 logarithms, and ln for natural logarithms.

3 AKS algorithm and its correctness

Algorithm 1 AKS(n)

Step 1:
if $n = a^b$ for $a \in N$ and $b > 1$ **then**
  output $COMPOSITE$.
end if
Step 2:Find the smallest r such that $ord_r(n) > \log^2 n$.
Step 3:
if $1 < (a, n) < n$ for some $a \leq r$ **then**
  output $COMPOSITE$.
end if
Step 4:
if $n \leq r$ **then**
  output $PRIME$.
end if
Step 5:
for $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ **do**
  if $((X + a)^n \neq X^n + a \ (mod\ X^r - 1, n))$ **then**
    output $COMPOSITE$.
  end if
end for
Step 6:output $PRIME$.

Definition 3.1. *Given $r \in N$, $a \in Z$ with $(a, r) = 1$, the order of a modulo r is the smallest number k such that $a^k = 1 \ (mod\ r)$. It is denoted as $ord_r(a)$.*

Definition 3.2. *For $r \in N$, $\phi(r)$ is Euler's totient function giving the number of numbers less than r that are relatively prime to r.*

Theorem 3.3. *The algorithm above returns $PRIME$ if and only if n is prime.*

We will establish this theorem through a sequence of lemmas.The following is trivial:

Lemma 3.4. *If n is prime, the algorithm returns $PRIME$.*

Proof. If n is prime then steps 1 and 3 can never return $COMPOSITE$. By Lemma 2.1, the for loop also cannot return $COMPOSITE$. Therefore the algorithm will identify n as $PRIME$ either in step 4 or in step. □

The converse of the above lemma requires a little work. If the algorithm returns $PRIME$ in step 4 then n must be prime since otherwise step 3 would have found a nontrivial factor of n. So the only remaining case is when the algorithm returns $PRIME$ in step 6. Here we will omit the proof of correctness of step 5, for its complexity and the author's knowledge level, and you can see [1] for a detail proof. In a word, we say that if for $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$, $((X + a)^n = X^n + a \ (mod\ X^r - 1, n))$, then n must be p^k for some $k > 0$. If $k > 1$ then the algorithm will return $COMPOSITE$ in step 1, otherwise the algorithm will return $PRIME$ and n is certainly prima, without a proof. To show the existence of such r in step 2 and for the complexity analysis, we also need bound the magnitude of the appropriate r.

Lemma 3.5. *There exists an* $r \leq max\{3, \lceil \log^5 n \rceil\}$ *such that* $ord_r(n) > \log^2 n$.

Proof. This is trivially true when $n = 2$, for $r = 3$ may satisfy all conditions. So assume that $n > 2$. Then $\lceil \log^5 n \rceil > 10$ and Lemma 2.2 can be applied. Let r_1, r_2, ..., r_t be all numbers such that either $ord_{r_i}(n) \leq \log^2 n$ or r_i divides n. Each of these numbers must divide the product

$$\begin{aligned} n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1) &< n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} n^i \\ &= n^{1 + \frac{(1 + \lfloor \log^2 n \rfloor)\lfloor \log^2 n \rfloor}{2}} \\ &< n^{\log^4 n} \\ &= (2^{\log n})^{\log^4 n} \\ &= 2^{\log^5 n} \end{aligned}$$

By Lemma 2.2, the lcm of the first $\lceil \log^5 n \rceil$ numbers is at least $2^{\lceil \log^5 n \rceil}$ and therefore there must exist a number $s \leq \lceil log^5 n \rceil$ such that $s \notin \{r_1, r_2, \cdots, r_t\}$ and s does not divide $n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor}(n^i - 1)$(Or $LCM(\lceil \log^5 n \rceil)$ will less than or equal to $n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor}(n^i - 1)$, and it conflicts with Lemma 2.2). If $(s, n) = 1$ then $ord_s(n) > \log^2 n$ and we are done. If $(s, n) > 1$, then since s does not divide n and $(s, n) \in \{r_1, r_2, \cdots, r_t\}$, let $r = \frac{s}{(s,n)} \neq 1$ and $(r, n) = 1$. If $r \in \{r_1, r_2, \cdots, r_t\}$, then $ord_r(n) \leq \log^2 n$, which means r can divide $\prod_{i=1}^{\lfloor \log^2 n \rfloor}(n^i - 1)$. However, since (s, n) can divide n, we have $s = r \cdot (s, n)$ can divide $n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor}(n^i - 1)$! It conflicts with what we have shown before and therefore $r \notin \{r_1, r_2, \cdots, r_t\}$, so $ord_r(n) > \log^2 n$. □

4 Time complexity analysis

In calculations of the time complexity of the algorithm, we use the fact that addition, multiplication and division operations between two m bits numbers can be performed in time $O\tilde{}(m)$. Similarly, these operations on two degree d polynomials with coefficients at most m bits in size can be done in time $O\tilde{}(d \cdot m)$ steps.

Lemma 4.1. *Given $n \in N$, $n > 1$, determining if n is a perfect power(i.e. whether there exists numbers $m \in N$, $k \in N$, $k > 1$ such that $n = m^k$)can be done in asymptotic time $O\tilde{}(\log^3 m)$.*

Proof. In order to test whether or not a natural number n is a perfect power, we can conduct a binary search of the integers $\{1, 2, \cdots, n\}$ for a number m such that $n = m^k$ for some $k > 1$. First we let $k = 2$. If there exists a solution for such m and k, then m must lie in some interval $[a_i, b_i]$. When $i = 0$, we take $[a_0, b_0] = [1, n]$. To define $[c_{i+1}, d_{i+1}]$, consider $\alpha = \lfloor \frac{c_i + d_i}{2} \rfloor$. If $\alpha^k = n$ then we are done. If $\alpha^k > n$, let $[a_{i+1}, b_{i+1}] = [a_i, \alpha]$; otherwise $\alpha^k < n$ and we let $[a_{i+1}, b_{i+1}] = [\alpha, b_i]$. We continue in this manner until $|a_i - b_i| \leq 1$. Then we increase k by 1 and start the loop again. Performing this loop for all $k \leq \log(n)$ completes the algorithm.

Every time we check whether $\alpha^k = n$ taking asymptotic time $O\tilde{}(k \log n)$, and each loop needs at most $O(\log n)$ times checks, and we at most need to do the loop $O(\log n)$ times, so it takes asymptotic time $O\tilde{}(\log^3 m)$ in total. □

Theorem 4.2. *The asymptotic time complexity of the algorithm is $O\tilde{}(\log^{\frac{21}{2}} n)$.*

Proof. By Lemma 4.1, we know the first step of the algorithm takes asymptotic time $O\tilde{}(\log^3 m)$.

In step 2, we find an r with $ord_r(n) > \log^2 n$. This can be done by trying out successive values of r and testing if $n^k \neq 1 \ (mod\ r)$ for every $k \leq \log^2 n$. For a particular r, this will involve at most $O(\log^2 n)$ multiplications modulo r and so will take time $O\tilde{}(\log^2 n \log r)$. By Lemma 3.5 we know that only $O(\log^5 n)$ different r's need to be tried. Thus the total time complexity of step 2 is $O\tilde{}(\log^7 n)$.

In step 3 we need compute the gcd of r numbers. Each gcd computation takes time $O(log^2 n)$(replacing modulo operation in the Euclid's algorithm by subtraction), and therefore, the time complexity of this step is $O(r \log n) = O(\log^7 n)$.

In step 4, the time complexity is just $O(\log n)$.

In step 5, we need to verify $\lfloor \sqrt{\phi(r)} \log n \rfloor$ equations. Each equation requires $O(\log n)$ multiplications of degree r polynomials with coefficients of size $O(\log n)$. So each equation can be verified in time $O\tilde{}(r \log^2 n)$ steps. Thus the time complexity of step 5 is $O\tilde{}(r\sqrt{\phi(r)} \log^3 n) = O\tilde{}(r^{\frac{3}{2}} \log^3 n) = O\tilde{}(\log^{\frac{21}{2}} n)$. This time dominates all the others and is therefore the time complexity of the algorithm. □

Furthermore, if based on some conjectures that have not been proved but probably seem to be true(like *Artin's Conjecture or Sophie−Germain Prime Density Conjecture*), the complexity of AKS algorithm can be reduced to $O\tilde{}(\log^6 n)$. Anyway, we give the following lemma without proof, which help reduce the complexity to $O\tilde{}(\log^{\frac{15}{2}} n)$:

Lemma 4.3. *Let P(m) denote the greatest prime divisor of number m. There exist constants $c > 0$ and n_0 such that, for all $x \geq n_0$:*

$$|\{q \text{ is prime, } q \leq x \text{ and } P(q-1) > q^{\frac{2}{3}}\}| \geq c\frac{x}{\ln x}$$

.

Theorem 4.4. *The asymptotic time complexity of AKS algorithm is $O\tilde{}(\log^{\frac{15}{2}} n)$.*

Proof. As argued above, a high density of prime q with $P(q-1) > q^{\frac{2}{3}}$ implies that step 2 of the algorithm will find an $r = O(\log^3 n)$ with $ord_r(n) > \log^2 n$. This brings the complexity of the algorithm down to $O\tilde{}(\log^{\frac{15}{2}} n)$. □

5 Some thoughts and improvements about AKS algorithm

As mentioned above, the asymptotic time complexity of AKS algorithm is $O^{\sim}(\log^{\frac{15}{2}} n)$ and in 2005 Lenstra[4] proposed a variant of AKS which runs in $O^{\sim}(\log^7 n)$ time. However, both of these improvements are only of theoretical interest and the algorithm suffers impracticality due to the expensive computations involved in each step and its storage requirements[2].

Since AKS algorithm was proposed, there were many works trying to implement the algorithm in different way for better performance. However, all these efforts result in slight increase in the speed of execution. Even though these works did not run the implementation on very large numbers, they provided an estimate on the number of years it would take to get the results as n grows larger and larger. In other words, no matter how much ever the performance gain, the inherent problem in the algorithm has not been solved. The storage of polynomial coefficients remains an unsolved problem and it should be noted that the coefficients belong to Z_n. Zhengjun Cao[2] describes the storage requirements for the AKS test and remarks that $1,000,000,000$ GB of storage is required for an input size of 1024 bits. Unless an alternative is devised to rule out the expensive verification of the polynomial congruence in step 5, the algorithm could never be brought into practice.

As described in the previous section, we can easily see that the value of r determines both the time and complexity of AKS. The larger the r, the larger is the space required to store all the $2r$ number of coefficients of the polynomials in step 5 as well as the computational time required to carry out the steps 3, 4 and 5. Therefore it is a question of interest to know whether it is possible to fix the lower and upper bounds for r, and determining the range of r also helps reducing the search space required in step 2, a second bottle neck for the algorithm. Lalitha Kiran Nemana and V. Ch. Venkaiah[6], through experiments on a large amount of data, proposed some interesting observations about the range and primality of r and the congruence equations needed to be verified in practical terms, which help get an improvement algorithm:

Observation 5.1. *For the numbers which enter step 5, it it quite imperative to make* $(\lceil \log n \rceil - 1)^2$ *as the lower bound, and* $1.4(\lceil \log n \rceil)^2$ *as the upper bound for* r.

Review AKS algorithm, the correctness of it does not depend on the smallest r at all, so though $r \leq (\lceil \log n \rceil - 1)^2$ for most of the composites identified at step 3(in fact, this is also an observation of the authors), we can still choose such r that is greater than $(\lceil \log n \rceil - 1)^2$.

Observation 5.2. *For the composites which enter step 5, the corresponding* r *is mostly prime.*

As mentioned above, it is not necessary to choose the smallest r for it does not affect the correctness of the algorithm at all. Therefore choosing a prime r between its bound with $ord_r(n) = r-1$ can certainly make sense. On the other hand, since the value of r happens to be much smaller when compared to n, testing whether r is prime is easy.

Observation 5.3. *Frequency of composites reported at steps 1, 3 is significantly larger than that at step 5.*

This suggests that AKS is a *better compositeness tester than a primality tester*.

Observation 5.4. *For all composite n entering step 5, the first value of a for which the loop breaks, say a', is very small in magnitude, and* $\forall a \in [a', \lfloor \sqrt{\phi(r)} \log n \rfloor]$, $(X + a)^n \neq x^n + a \ (mod\ X^r - 1, n)$.

This observation gives us an attempt to replace step 5 with just one congruence equation, and by some clever implementations(see [6] for detail) we can greatly improve the efficiency of AKS.

Based on the observations above that were made from empirical results, the authors modified AKS algorithm to give the following variant which *gave no false postives on varied bit ranges*(which can meet the needs of actual projects in a great measure):

Algorithm 2 AKS-Improvement(n)

1: **if** $n = a^b$ for $a \in N$ and $b > 1$ **then**
2: output $COMPOSITE$.
3: **end if**
4: Choose a prime $r \in ((\lceil \log n \rceil - 1)^2, 2(\lceil \log n \rceil)^2)$ and certainly $ord_r(n) \geq \lfloor \log^2 n \rfloor$.
5: **if** $1 < (a, n) < n$ for some $a < r$ **then**
6: output $COMPOSITE$.
7: **end if**
8: **if** $n \leq r$ **then**
9: output $PRIME$.
10: **end if**
11: **if** $(X + \lfloor \sqrt{\phi(r)} \log n \rfloor)^n \neq x^n + \lfloor \sqrt{\phi(r)} \log n \rfloor \ (mod\ X^r - 1, n)$ **then**
12: output $COMPOSITE$.
13: **end if**
14: output $PRIME$

Since $r = \theta(\log^2 n)$, steps 2, 3, 5 take $O^{\sim}(\log^4 n)$, $O(\log^3 n)$ and $O^{\sim}(\log^4 n)$ times respectively. Therefore, the overall running time is $O^{\sim}(\log^4 n)$. However, its correctness has to be investigated theoretically.

6 Primality Test in practice

The Primality Test is an important area in the cryptology.This part will introduce two primality-test algorithms that are useful in the engineering area.For example, the Fermat primality test algorithm is used in the encryption procedure PGP.I will introduce the Miller-Robin Primality Test and the Fermat Primality Test in order.And in the end of this part, I will make the comparisons between these algorithms and the AKS algorithm, and figure their pros and cons.

6.1 some used theorem in the Primality Test

6.1.1 Fermat Little Theorem

If p is a prime, then for every $1 \leq a \leq p-1$,we have $a^{p-1} \equiv 1(mod\ p)$

6.1.2 Quadratic detection theorem

If p is a prime, and $1 \leq x \leq p-1$,the equation $x^2 \equiv 1(mod\ p)$ has the solution of $x = 1, p-1$.
Simple Proof:
Because $x^2 \equiv 1(mod\ p)$, we can get that:

$$(x+1)(x-1) \equiv 0(mod\ p)$$

And because $1 \leq x \leq p-1$, x can only takes 1, p-1 as the value.

6.2 Miller-Robin Primality Test

6.2.1 Algorithm

Algorithm 3 Miller-Robin(n)

As n is a odd number, let $n-1 = m2^q$. Then pick a random integer $1 \leq a \leq n-1$.
if $a^{n-1} \equiv 1(mod\ n)$ is FALSE **then**
    output $COMPOSITE$.
end if
Then for the following *Miller Sequence*:
$a^m(mod\ n), a^{2m}(mod\ n), \ldots, a^{m2^{q-1}}(mod\ n)$,
if $a^m \equiv 1 mod\ n$ **then**
    output $PRIME$.
end if
if there exists some $0 \leq i \leq q-1$, such that $a^{m2^i} \equiv (n-1)(mod\ n)$ **then**
    output $PRIME$.
end if

6.2.2 Simple Proof

Assume the Input n is a Prime.And we take $n-1 = m2^q$.Pick a random number $1 \leq a \leq n$.Then according to the Fermat Little Algorithm(1.1.1),we can get $a^{n-1} \equiv 1(mod\ n)$.
As n is a Prime, so we can get the equation from the Quadratic detection theorem(1.1.2):
$a^{m2^{q-1}} \equiv 1(mod\ n)$ or $a^{m2^{q-1}} \equiv n-1(mod\ n)$.
And then we use Quadratic detection theorem to compute the above equations for some t times, and we can finally get some $0 \leq r \leq q-1$ such that:
$a^{m2^r} \equiv 1(mod\ n)$ or $a^{m2^r} \equiv n-1(mod\ n)$,where 2^r should be a little number.

6.2.3 Time Complexity

For the Input value n, we need to consider the q numbers provided in the Miller Sequence separately.And for every number, we need to do a model operation, which takes $O(log^3 n)$ time.And because $n-1 = m2^q$, we can get that $q = log(\frac{n}{m})$, So the total Time Complexity is $O(qlog^3 n) = O(log^4 n)$, where the n is the primality test target.

6.3 Fermat Primality Test

6.3.1 Algorithm

Algorithm 4 Fermat(n, k)

for each $i \in [1, k]$ **do**
    Pick a random int $1 \leq a \leq n-1$;
    if $a^{n-1} \equiv 1(mod\ n)$ is FALSE **then**
        output $COMPOSITE$.

5: **end if**
6: **end for**
7: output $PRIME$.

6.3.2 Simple Proof

We can prove it from the Fermat Little Theorem trivially.For a value n, if it doesn't satisfy the Fermat Little Theorem,it must not a prime.But the value that satisfies Fermat Little Theorem still has the probability that it is not a prime.

6.3.3 Time Complexity

There are k iterations in this algorithm.And in every iteration, we need to compute a model operation, so the total time complexity is $O(klog^3 n)$, where the n is the primality test target.

6.4 Conclusion

6.4.1 Comparison between AKS algorithm and stochastic algorithms

The algorithms introduced above are all *stochastic algorithms*, which means that their test results aren't *deterministic*.But they are used more wildly than the deterministic algorithm AKS.There are some possible reasons:
1.These algorithms have the better time complexity performance than the AKS algorithm.The AKS algorithm has the time complexity $O(log^{7.5} n)$ without optimization, but the miller-robin test algorithm only takes $O(log^4 n)$ while the Fermat test algorithm takes $O(klog^3 n)$.
2.Although these stochastic algorithms have the probability to get the wrong result, this probability is proved to be small.For example, the wrong rate of the Miller-Robin test algorithm is proved to have a upper bound $\frac{1}{4}$ in every time we run it[3].So the total wrong rate of the Miller-Robin test algorithm for running k times is $(\frac{1}{4})^k$.So we can just increase the value of k to drop the rate quickly.So we can add some restriction to make the Miller-Robin test's result more deterministic. And as the value of k won't have a big impact on the O time analysis,so we still think these stochastic algorithms are more efficient than the AKS algorithm.
3.The strict storage requirement of the AKS algorithm.As discussed in the previous section, for the original AKS algorithm, it will take $1,000,000,000$ GB to store the polynomial coefficients produced during the algorithm for the 1024 bits input.As the development of the encryption, there is a huge requirement for the primes that has the big length. This is unbearable for the engineering in practice.So this may be another reason that makes the AKS algorithm suffers the impracticality in the engineering area.

6.4.2 Some concrete examples of the application of these algorithms

1.Usage in the RSA algorithm.RSA is a famous encryption algorithm.It needs to find two primes that big enough to get the secret key.So the RSA needs to verify the two numbers found are primes.So it requires the primality test algorithm.And for a better performance, it always use the stochastic algorithms rather than the AKS algorithm.

7 Final Conclusion

In this notes, we analyse the AKS algorithm particularly.We prove the correctness of the AKS algorithm,and analyze its time complexity.But in the last part of this notes, we find that though the AKS is the first deterministic primality test algorithm, it is not widely used in the engineering area.because of its bad performance.So through this we can get a glimpse of the difference between the application and the theory, and we can release that there is still a long way to go from the number theory to the widespread use in the engineering area.

References

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

[2] Zhengjun Cao. A note on the storage requirement for aks primality testing algorithm. *Cryptology ePrint Archive*, 2013.

[3] ST Ishmukhametov, R Rubtsova, and N Savelyev. The error probability of the miller–rabin primality test. *Lobachevskii Journal of Mathematics*, 39(7):1010–1015, 2018.

[4] Hendrik W Lenstra. Primality testing with gaussian periods. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 1–1. Springer, 2002.

[5] Mohan Nair. On chebyshev-type inequalities for primes. *The American Mathematical Monthly*, 89(2):126–129, 1982.

[6] Lalitha Kiran Nemana and V Ch Venkaiah. An empirical study towards refining the aks primality testing algorithm. *Cryptology ePrint Archive*, 2016.