1 Primality Test(AKS)

- Algorithm: Primality Test(AKS) (algo. 1)
- Input: An integer n, which satisfies $n > 1$
- Complexity: $\mathcal{O}(\log^{6+\epsilon} n)$ in general cases
- Data structure compatibility: N/A
- Common applications: Generating primes for cryptographic protocols

Primality Test(AKS)

Primality test is a kind of algorithm determining whether a number is prime or not. This kind of algorithm can only determine this property rather than further gives the factors of the input. AKS primality test is the first deterministic test that can judge whether a general input is a prime or not within polynomial time. Moreover, the proof of this test does not contain any hypothesis or base on any unproven theorems, so it will not cause errors in implementation.
Multiplying large primes are easy, while factorizing a number into primes is much harder. AKS primality test can be used in some cryptosystems, such as RSA cryptosystem to generate large primes, so that if we encrypt information using these primes, the decrypting process will be rather hard, and thus information security is permitted.

Description

There are various primality tests. The most basic primality test is trial division test, in which we divide an input by numbers from 2 to the square root of that input to judge whether it is a prime. Currently, most widely used primality tests are elliptic curve primality test and the probabilistic Miller–Rabin test. They are fastest in application, but the proof of these two tests are based on some unproven conjectures and hypothesis. However, AKS primality test is a test whose proof does not contain any unproven theorems.

In the following sections, we will first prove the lemma which AKS primality test based on. Moreover, a detailed version of the algorithm will be provided and its correctness and complexity will be analyzed. We will also state why AKS primality test is a $\mathcal{P}$ problem in Complexity analysis section. At last, we will mention its applications.

Essence

AKS primality test is based on the generalization of Fermat's little theorem. We first mention Fermat's little theorem here.

Theorem 1 If p is a prime number, then for any integer a, we have

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

Noticeably, Theorem 1 does not mean that if a number p satisfies $a^p \equiv a(mod\ p)$, the number is a prime, so if we simply apply Theorem 1 in primality test, we will achieve some pseudoprimes. For instance, $8^9 \equiv 8(mod\ 9)$, but 9 is not a prime. Therefore, we need to apply a more generalized and deterministic statement.

This generalized and deterministic statement is the main idea of this paper, which is based on the generalization of the Fermat's little theorem and is stated as follows.

Lemma 1 Let $a \in \mathcal{Z}$, $n \in \mathcal{N}$ and $n \geq 2$. If a and n are relatively prime to each other, we have n is prime iff

$$(X + a)^n \equiv X^n + a(mod\ n).$$

The proof of lemma 1 is shown as follows.
If n is a prime number. We have

$$(X + a)^n - (X^n + a) = \sum_{j=1}^{n-1} \binom{n}{j} X^{n-j} a^j + a^n - a$$

Since $\binom{n}{j} = \frac{n!}{(n-j)!j!} = \frac{n(n-1)\cdots(n-j+1)}{j(j-1)\cdots 1} \in \mathcal{N}$, and n is a prime which will not be divided by any integer from 2 to j ($j \neq n$), we have $\binom{n}{j} = n{\cdot}k$, in which $k \in \mathcal{N}$. So we have $\binom{n}{j} \equiv 0(mod\ n)$ for any $j \neq 0, n$. Moreover, according to Theorem 1, $a^n - a \equiv 0(mod\ n)$. So the coefficients of all degrees of the polynomial will vanish if mod by n. We can then achieve $(X + a)^n - (X^n + a) \equiv 0(mod\ n)$. Therefore, if n is a prime, we have $(X + a)^n \equiv (X^n + a)\ (mod\ n)$.

If n is composite. Let $n = p^q{\cdot}n'$, in which p is a prime and $p \nmid n'$. In this case, $\binom{n}{p} = \frac{n(n-1)\cdots(n-p+1)}{p(p-1)\cdots 1}$. Since any integer from $n-p+1$ to $n-1$ does not have a factor p. So $\binom{n}{p} = \frac{n(n-1)\cdots(n-p+1)}{p(p-1)\cdots 1} = p^{q-1}{\cdot}k$, in which $p \nmid k$. Moreover, since a and n are relatively prime, we have $n \nmid a^p$, so we can obtain $n \nmid \binom{n}{p} a^p$. So for any composite n, we will have $(X + a)^n \not\equiv X^n + a(mod\ n)$. □

However, if we simply apply Lemma 1, it will cost $\Omega(n)$ to evaluate all $n + 1$ coefficients of the polynomial $(X + a)^n - (X^n + a)$. Therefore, in this algorithm, we will instead evaluate whether

$$(X + a)^n \equiv X^n + a(mod\ n, X^r - 1).$$

We will first divide the left and right side by $X^r - 1$, and compare whether the coefficients of the reminders are the same $mod\ n$. By this improvement, the number of the coefficients we need to evaluate will decrease to r.

Pseudocode

The pseudocode of the AKS primality test is shown in Algorithm 1.

Algorithm 1: AKS primality test

```
   Input  : an integer n > 1
   Output: PRIME or COMPOSITE
 1 Function AKS(n):
 2     if n == a^b for some integers a > 1 and b > 1 then
 3         return COMPOSITE;                          /* Test whether n is a simple power */
 4     end if
 5     r ← 0;
 6     bound ← max{3, ⌈log^5 n⌉};
 7     for num in 1 to bound do
 8         for k from 1 to log^2(n) and gcd (k, r) = 1 do
 9             if no k satisfies n^k ≡ 1(mod num) then
10                 r ← num;
11                 break;      /* Find the smallest r, which let the order of n mod r larger than log^2(n) */
12             end if
13         end for
14     end for
15     for a from 1 to r do
16         if 1 <gcd (a, n) < n then
17             return COMPOSITE;
18         end if
19     end for
20     if n ≥ r then
21         return PRIME;
22     end if
23     for a from 1 to ⌊√φ(r) log n⌋ do
24         Calculate the reminder of (X + a)^n divide X^r − 1;
25         Calculate the reminder of X^n + a divide X^r − 1;
26         if difference of the coefficients of the reminders≢ 0(mod n) then
27             return COMPOSITE;
28         end if
29     end for
30     return PRIME;
31 end
```

Correctness of the algorithm

We then simply explain the correctness of the algorithm.

The code will output COMPOSITE only in line 3, 16 and 27, which means $n = a^b$ for $a, b \geq 2$, n and some $a < n$ is not relatively prime, and $(X + a)^n \equiv (X^n + a)(mod\ n)$ is violated respectively. In all three cases, n is definitely a composite.

The code will output PRIME in line 20 and 30.

In line 20, if $n \geq r$, since we have judged whether $1 < \gcd(a, n) < n$ for all $a < r$, it mean n is relatively prime to all $a < n$. Therefore, n is a prime number.

In line 30, the situation is more complex. Although $(X + a)^n \equiv X^n + a (mod\ n)$ is the necessary and sufficient condition for n is prime, whether $(X + a)^n \equiv X^n + a (mod\ n, X^r - 1)$ has this property is not clear. In the paper, the author has proven that if we can analysis all the coefficients of the congruence equation with different values of a from 1 to $\lfloor \sqrt{\varphi(r)} \log n \rfloor$, we can determine whether n is a prime.

The proof is beyond the range of this course, so we only provide a sketch here.

Since $(X + a)^n \equiv X^n + a (mod\ n, X^r - 1)$ definitely holds for all primes, we need to analyze whether there are any composites satisfying this equation.

Assume for a composite n, we have $n = p \cdot n'$, in which p is a prime. The non-zero reminders of $(x + a)^n\ mod\ (p, h(x))$ will form a group of order $p^m - 1$. Here, m is the degree of $h(x)$, and $h(x)$ is a factor of cyclotomic polynomial $\Phi_r(x)$ which cannot be further factorized in $\mathbb{Z}$. Moreover, we denote the cardinality of the group containing all the reminders of $(\frac{n}{p}^i \cdot p^j)\ mod\ r$ as t. We denote the group formed by the non-zero reminders of $(x + a)^n\ mod\ (p, h(x))$ multiply $x + a$, in which $a \in \{1, 2, \cdots, \sqrt{\varphi(r) \log n}\rfloor\}$ by $\mathcal{G}$. Then, for all n, we will have the cardinality of the group $|\mathcal{G}| \geq \binom{t + \lfloor \sqrt{\varphi(r) \log n} \rfloor}{t - 1} \geq n^{\sqrt{t}}$. Moreover, the cardinality of the group $|\mathcal{G}|$ will be less or equal than $n^{\sqrt{t}}$ if $n \neq p^k$. Therefore, we can conclude that if a composite n satisfies all the convergence equations, it needs to be a power number. In this case, since we have excluded all the simple powers in the beginning of the algorithm, we can be sure that if all the convergence equations are satisfied, n is a prime.

Complexity analysis

We first introduce the soft-O notation, which is denoted as $\tilde{\mathcal{O}}$. It means if $f(n) \in \tilde{\mathcal{O}}(g(n))$, there $\exists k$, such that $f(n) \in \mathcal{O}(g(n) \log^k g(n))$.

In the first part of the algorithm, we calculated whether n is a simple power. We can do so by checking b from 2 to $\log n$. This process takes asymptotic time $\tilde{\mathcal{O}}(\log^3 n)$.

In the second part of the algorithm, we calculated the smallest r satisfying the order of $n\ mod\ r$ is larger than $\log^2(n)$, and we also calculated the gcd of all numbers smaller than r with n. The author has proved that the bound of r will not exceed $\log^5 n$. Moreover, for a certain r, in order to check whether the order of $n\ mod\ r$ is larger than $\log^2(n)$, we can simply check whether for all k from 1 to $\log^2 n$, we have $\gcd(k, r) = 1$. In this case, the total asymptotic time will not exceed $\mathcal{O}(\log^5(n) \cdot \log^2(n)) = \mathcal{O}(\log^7(n))$. Moreover, since calculating gcd takes asymptotic time $\mathcal{O}(\log(n))$, figuring out the gcd of all numbers smaller than r is $\mathcal{O}(\log^6 n)$.

In the last part, we need to verify $\lfloor \sqrt{\varphi(r)} \log n \rfloor$ congruence equations, which is equivalent to $\mathcal{O}(r^{\frac{1}{2}} \log n)$ equations. To minimize the complexity, we will modulus the polynomial by $X^r - 1$ and n in every step, so the degree of the polynomial we are calculating will never exceed r, and the scale of the coefficients will never exceed n.

By Fast Fourier Transform algorithm, the multiplication of two integers $a, b < n$ is $\mathcal{O}(\log n)$. Moreover, for two

polynomials with degree r, the multiplication of these two polynomials can be done in $\tilde{\mathcal{O}}(r)$ multiplications.

Since rising $(X+a)$ to $(X+a)^n$ requires at least $\log n$ multiplications. We need $\tilde{\mathcal{O}}(r \log^2 n)$ steps to evaluate the reminders for each different value of a. So the total complexity for this step is $\mathcal{O}(r^{\frac{1}{2}} \log n) \cdot \tilde{\mathcal{O}}(r \log^2 n) = \tilde{\mathcal{O}}(r^{\frac{3}{2}} \log^3 n)$. Since $r = \mathcal{O}(\log^5 n)$, we have $\tilde{\mathcal{O}}(r^{\frac{3}{2}} \log^3 n) = \tilde{\mathcal{O}}(\log^{10.5} n)$.

$\tilde{\mathcal{O}}(\log^{10.5} n)$ is only a loose upper bound for the complexity of AKS primality test. In best cases, the value of r will not exceed $\log^2 n$, so the complexity of this algorithm is $\tilde{\mathcal{O}}(\log^6 n)$ in best cases.

AKS primality test is a problem belonging to $\mathcal{P}$ because given the number n in binary, it will have $N = \log n$ digits. Therefore, for an input of length N bits, the algorithm can be finished in $\tilde{\mathcal{O}}(N^6)$, which is a polynomial time.

Application

Primality tests are widely used for cryptography. However, since there are faster tests, such as elliptic curve primality tests, AKS is rarely used in real world to generate large primes. Nevertheless, in the cases when we do not need to generate a lot of large primes, AKS primality test can still be applied. For instance, we can use AKS primality test in RSA cryptosystem.

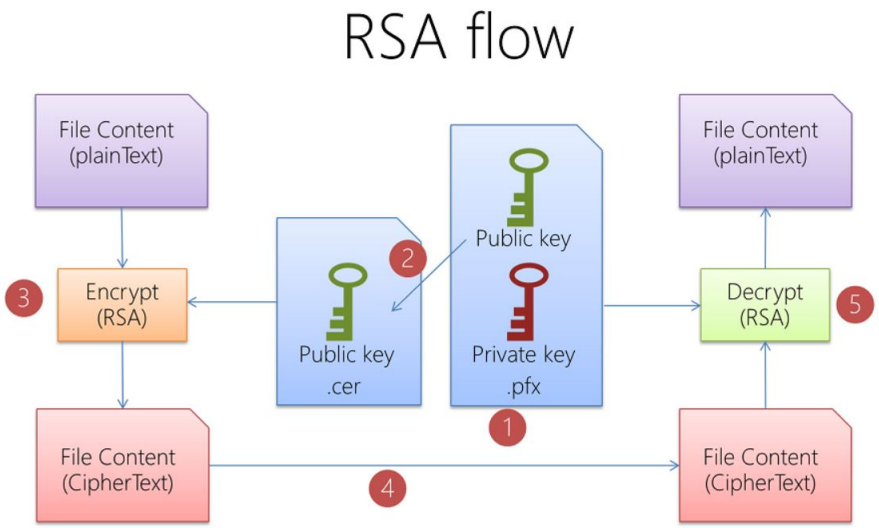

Figure 1: RSA Flow

Figure 1 shows the process of RSA. In RSA cryptosystem, a public key and a private key is involved. Public key is used to encrypt the message. The encrypted message can only be decrypted by the private key. While constructing the keys, we need to first generate two large primes p and q, then generate the corresponding public key pair (n, e), in which $n = pq$ and e satisfies $\gcd(e, \varphi(n)) = 1$. Then if user A decides to accept some secret information from user B, A can send public key (n, e) to B, and B will then encrypt the message using the public key and transmit it back to A. Finally, A can decode the message using his private key. In this whole process, AKS primality test can be applied when we need to generate the large primes used to produce pubic and private keys.

References

- Introduction to the AKS primality test. SlideShare. https://www.slideshare.net/PranshuBhatnagar/introduction-to-the-aks-primality-test
- J. Gathen and J. Gerhard. Modern Computer Algebra. Cambridge University Press, 1999.
- M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf
- "Primality test" in Wikipedia. Wikimedia foundation. https://en.wikipedia.org/wiki/Primality_test
- RSA (cryptosystem). University of North Texas. http://www.cse.unt.edu/~tarau/teaching/PP/NumberTheoretical/RSA%20(cryptosystem).pdf
- R. Singer-HeinzeRun. Time Efficiency and the AKS Primality Test. University of California,Santa Barbara. http://ccs.math.ucsb.edu/senior-thesis/Roeland-Singer.pdf
- RSA Flow. Image. https://ithelp.ithome.com.tw/articles/10188824
- "$\tilde{\mathcal{O}}$" in Wikipedia. Wikimedia foundation. https://en.wikipedia.org/wiki/%C3%95#Mathematical_use