0.1 Primality testing (AKS)

- *Algorithm:* name (algo. 1)
- *Input:* A positive integer $n > 1$
- *Complexity:* $\mathcal{O}(\log^{21/2} n)$
- *Data structure compatibility:* N/A
- *Common applications:* Primality testing is useful in the field of Cryptography and complexity theorem.

Problem. Primality testing (AKS)

Given a positive integer $n > 1$, we want to decide whether it is a prime number.

Description

A first look at the problem

In this problem, we are going to discuss an algorithm that can decide whether an integer $n > 1$ is a prime or not(which means composite). In the field of Cryptography, many protocols are designed based on large prime numbers. The question comes up: can we do primality testing in polynomial time? Note that the size of input is not the integer n but the bits that it has, which is $\mathcal{O}(\log n)$. So brute force is never recommended since it requires to test all $1 < i < \lfloor\sqrt{n}\rfloor$ and whether $i|n$. Since we need to compare the time complexity with input size, it will be an exponential-time algorithm.

Some algorithm not that good In algorithm for this problem, we need to focus on four important characters: general, deterministic, unconditional, and polynomial[7]. Here general means that all the input can be calculated, deterministic is the opposite of random algorithm, unconditional means that the proof of its correctness does not depend on any unproved theorem, lastly, polynomial is the requirement of time complexity. People are able to design algorithms that satisfy three conditions out of four, but no one can satisfy all until the idea from Agrawal Kayal Saxena.

Although there are many algorithms proposed that is not perfect, they focus on different characters and is still important. I will introduce them by some examples. Lucas–Lehmer primality test has a limitation that is only used for Mersenne numbers[3]. In real life application, we are interested in other integers as well, so we need to explore more. Miller-Rabin algorithm is introduced in VE475. It is a random algorithm, which means we can not be 100 percent sure about its result. If the algorithm says n is composite, then it must be composite. But when it says that it is a prime, it has a probability. So to increase the probability, we need to use this algorithm with different parameters[4]. Another algorithm, Miller test, can be determined, but we need to assume the correctness of an unproven lemma Riemann's Hypothesis[5].

AKS

This is an algorithm proposed by three Indian professors in 2002. Notice that it is an NP and co-NP problem. It is trivial to prove that this is in co-NP, just take an factor p of the number n as the certificate, we can easily calculate the GCD of p and n and get the decision. Pratt has proven that it belongs to NP in 1974[6]. The key idea in AKS is the equivalence of being prime and a module equation[1].

Key idea: given $n > 1$ to be an integer, a is another positive integer that is co-prime to n. Then n is prime if and only if

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

Here is a little proof of this idea. We discuss two cases. First, if n is prime, then we recall binomial expansion

and look at the coefficients carefully. Recall the definition of binomial coefficients.

$$\binom{n}{i} = \frac{n(n-1)\cdots(n-i+1)}{i!}$$

$\binom{n}{i}$ are dividable by n when i is greater than 0. For a^n, we can use Fermat little theorem and get $a^n \equiv a^{n-1} \times a \equiv a \mod n$. Another case is that when n is a composite, assume there is a prime factor of n and satisfies $p^k \| n$. Note that based on Fundamental Theorem of Arithmetic, we can always find such q and k. Then we consider the coefficient $\binom{n}{p}$, but there is an additional p in $p!$, so there is only k-1 p remaining, but we need k[2]. So this term will not satisfy the module equation. Since an integer greater than 1 can be either prime or composite, the proof is done.

Time complexity

The step that judge the perfect power cost $\mathcal{O}(\log^3 n)$[1]. So we do not need to worry about it. In the next step, we need to find the r, this step takes $\mathcal{O}(\log^7 n)$[1]. So the for loop afterwards requires to calculate $\mathcal{O}(\log^5 n)$ times gcd and thus $\mathcal{O}(\log^5 n)$. GCD takes another $\mathcal{O}(\log n)$, which leads to $\mathcal{O}(\log^6 n)$. The if afterwards does not matter, so we need to care about the next for loop. This last step requires $\mathcal{O}(r^{3/2} \times \log^3 n) = \mathcal{O}(\log^{21/2} n)$[1]. Compare the complexity before, we get the overall complexity as $\mathcal{O}(\log^{21/2} n)$.

Correctness

Too difficult.

Algorithm 1: AKS

Input : an integer $n > 1$
Output: Prime or Composite

if *n is a perfect power* **then**
  return Composite
end if
find smallest r where $o_r(n) > \log^2 n$
for $a \leq r$ **do**
  if *1 ¡ gcd(a, n) ¡ n* **then**
    return Composite
  end if
end for
if $n \leq r$ **then**
  return Prime
end if
for $a \leq \lfloor \sqrt{\phi(r)} \log n \rfloor$ **do**
  if $\left((X+a)^n \neq X^n + a \left(\mathrm{mod} X^r - 1, n\right)\right)$ **then**
    return Composite
  end if
end for
return Prime

References.

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P". In: *Annals of mathematics* (2004), pp. 781–793 (cit. on pp. 1, 2).

[2] Amnon Ta-Shma. *The AKS Algorithm*. [Online; accessed 5-October-2020]. 2018. URL: http://www.cs.tau.ac.il/~amnon/Classes/2019-Derandomization/Lectures/Lecture7-AKS-All.pdf (cit. on p. 2).

[3] JW Bruce. "A really trivial proof of the Lucas-Lehmer primality test". In: *The American mathematical monthly* 100.4 (1993), pp. 370–371 (cit. on p. 1).

[4] Manuel. *VE475 – Introduction to Cryptography (lecture slides)*. 2020 (cit. on p. 1).

[5] Gary L Miller. "Riemann's hypothesis and tests for primality". In: *Journal of computer and system sciences* 13.3 (1976), pp. 300–317 (cit. on p. 1).

[6] Vaughan R Pratt. "Every prime has a succinct certificate". In: *SIAM Journal on Computing* 4.3 (1975), pp. 214–220 (cit. on p. 1).

[7] Wikipedia contributors. *AKS primality test*. [Online; accessed 5-October-2020]. 2020. URL: https://en.wikipedia.org/wiki/AKS_primality_test (cit. on p. 1).