Naqi Jaffery
Math 168
December 9, 2005

The AKS algorithm for Polynomial-Time Primality Testing

Introduction

Given a number, can we distinguish if it is prime or composite, and if it is composite can we find its prime factorization? The former question is known as the "primality problem." Prime numbers are defined as any integer greater than one whose only positive divisors are 1 and itself. An integer that is not prime is said to be composite.

With the progression of computers, many cryptographic systems were created involving primes and factorization of a number into primes, such as RSA. For decades there have been many non-deterministic polynomial time algorithms that solve the primality question, but there has never been an unconditionally deterministic polynomial-time algorithm until 2002. What exactly makes an algorithm deterministic? Informally, it means that the algorithm behaves predictably and will always give the same output on a given input. Non-deterministic algorithms usually perform random experiments (via a random variable) or can just imply the path that the algorithm takes is not predictable or that the running time it takes is variable. For all practical (cryptographic) purposes these algorithms are still used (as they tend to be faster).

On August 6, 2002, a paper, *PRIMES is in P* written by three computer scientists: Manindra Agrawal, Neeraj Kayal, and Nitin Saxena was posted on the website of the Indian Institute of Technology in Kanpur, India (located in north India in the state of Uttar Pradesh not too far from the city of Lucknow-the capital of Uttar Pradesh, for you geography enthusiasts). This paper presented the AKS algorithm, the first deterministic polynomial-time algorithm that determines whether a number is prime or composite.

The Algorithm

The central idea of the algorithm is a generalization of Fermat's Little Theorem, which states the following:

Theorem (Fermat) *Suppose p is a prime. Then for all a in the integers* $a^p = a \pmod p$*. If p does not divide a (i.e. the greatest common divisor of p and a, is one), then*

$$a^{p-1}=1 \pmod p.$$

(See [4] for a nice proof of the theorem.)

The converse of Fermat's Little Theorem is not true, because there exists n that are not prime such that $a^{n-1}=1($mod $n)$, for all a with gcd $(a, n) = 1$. In fact these n have a special name, they are known as Carmichael numbers.

The following lemma is the generalization of Fermat's little theorem (see [1], [2], or [3] for proofs):

Lemma: *Let n be an integer greater than or equal to 2. Let a be an integer less n, furthermore a and n must be relatively prime (i.e., the greatest common divisor of a and n is 1). Then n is a prime number if and only if*

$$(X + a)^n = X^n + a \pmod{n}$$

over the ring of integers modulo n.

The above congruence is a simple test that determines whether a number n is prime. The only disadvantage is, it takes a long time to compute n coefficients in the left hand side of the identity. An easy way to reduce the number of coefficients on both sides of the identity, which leads to a more efficient algorithm, is to take the identity modulo a polynomial $X^r - 1$ for an appropriately chosen r. So now the test becomes

$$(X + a)^n = X^n + a \pmod{X^r - 1}$$

over the ring of integers modulo n, for appropriately chosen a and r. The left hand side is the same as $(X+a)^n$ mod (X^r-1). The right hand sides is the equivalent to $(X^n + a)$ mod $(X^r-1) = X^{n\ mo\ dr} + a$ (see [2] for further detail).

Here is a slightly varied algorithm from the original, as it tends to follow the algorithm from [2] than [1]. Some notation is defined for the algorithm as follows. *[X]* represents the ceiling function of *X*. Range is the list of integers between (x,y) including x, but ***not*** including y.

Input: Integer $n > 1$

```
1.    If ($n=a^b$ for a, b in the integers > 1):
2.          return "composite"
3.    r = 2
4.    while (r < n):
5.          if (r divides n):
6.                return "composite"
7.          if (r is a prime number):
8.                for i in range (1, 4*[log(n)]$^2$ ):
9.                      if ($n^i$ mod r does not equal 1):
10.                           break
11.         r = r + 1
12.   if (r = n):
13.         return "prime"
14.   for a in range (1, 2(r)$^{1/2}$*[log n]):
15.         if $(X+a)^n$ mod $(X^r-1)$ does not equal $X^{n \bmod r}$ + a in the
      ring of integers modulo n
16.               return "composite"
17.   return "prime"
```

The proof of the algorithm and running time bounds are given in [1], [2], and [3]. We implemented the AKS algorithm SAGE (Software for Algebra and Geometry Experimentation, [5]). We give a function AKS(n) below, where n is the input and the ouput is given as either “prime” or “composite.” Here are some examples:

```
sage: load "projectAKS.sage"
sage: AKS(2)
 _2 = 'prime'
sage: AKS(3)
 _3 = 'prime'
sage: AKS(5)
 _5 = 'prime'
sage: AKS(4)
 _6 = 'composite'
sage: AKS(13)
 _7 = 'prime'
sage: AKS(1001)
 _8 = 'composite'
sage: AKS(561)        561 happens to be a Carmichael number
 _9 = 'composite'
sage: AKS(1105)       1105 is also a Carmichael number
 _2 = 'composite'
sage: AKS(1729)       Oh look another Carmichael number
 _6 = 'composite'
sage: AKS(29341)       Could it be? A Carmichael number.
 _7 = 'composite'
sage: AKS(999991)
 _8 = 'composite'
sage: AKS(34567)
_10 = 'composite'
sage: AKS(3451)
_11 = 'composite'
sage: AKS(3243)
_12 = 'composite'
sage: AKS(3543)
_13 = 'composite'
sage: AKS(78563)
_14 = 'composite'
sage: AKS(9721)
_15 = 'prime'
```

SAGE Implementation of AKS

```
def AKS(n):
     for a in range(2,pari(sqrt(n)).floor()+1):
          for b in range(2,pari(sqrt(n)).floor()+1):
               if (n==a^b):
                    return 'composite'
     r=2
     m=pari(4*log(n,2))^2.floor()+1
     while(r<n):
          if(Mod(n,r)==0):
               return 'composite'
          if (is_prime(r)==True):
               for i in range(1,m):
                    if(power_mod(n,i,r)!=1):
                         break
          r=r+1
     if(r==n):
          return 'prime'
     n=pari(2*sqrt(r)*log(n,2)+1).floor()
     for a in range(1,n):
          R=PolynomialRing(Integers(n))
          x=R.gen()
          S=R.quotient((x^r)-1)
          c=S((x+a)^n)
          e=Mod(n,r)
          d=(x^e)+a
          if (c!=d):
               return 'composite'
     return 'prime'
```

References

[1] Agrawal, Kayal, Saxena. "PRIMES is in P". Annals of Mathematics, 160 (2004): 781-793.

[2] Dietzfelbinger, Martin. Primality Testing in Polynomial Time: From Randomized Algorithms to "PRIMES is in P". Berlin: Springer, 2004.

[3] Granville, Andrew. "It Is Easy To Detemine Whether A Given Integer Is Prime". Bulletin (New Series) of the American Mathematical Society, 42 (2004): 3-38.

[4] Stark, Harold M. An Introduction to Number Theory. Cambridge: The MIT Press, 1987.

[5] Stein, William. SAGE: Software for Algebra and Geometry Experimentation. http://modular.ucsd.edu. 0.9.17