Algorithms - Series Two(ii)

Mathematics is the Queen of the Sciences
Carl Friedrich Gauss

Faster Miller-Rabin

The version of the Miller-Rabin algorithm we implemented in the first series is quite slow. This is due to the expmod function we wrote. Here are a couple of functions that will speed the execution up quite a lot:

```
def do_tests(n, s, d, t):
    """Returns True only if the number passes t tests."""
    for i in range(t):
        if is_composite(n, s, d):
            return False
    return True

def as_2sd(d):
    """Returns a number in the form (2 ^ s) * d."""
    s = 0
    while even(d):
        s += 1
        d //= 2
    return [s, d]

def is_composite(n, s, d):
    """Miller-Rabin test. Returns True if n is composite."""
    a = random.randint(2, n)
    if pow(a, d, n) == 1:
        return False
    for r in range(s):
        ind = (2 ** r) * d
        if pow(a, ind, n) == n - 1:
            return False
    return True
```

You can replace this “do_tests()” for the original version and the program should work as before. Here I am using Python’s built-in power-modulus function:

```
>>> pow(4, 5, 6)
4
```

4 to the power 5 is 1024. 1024 modulus 6 is 4.

We are going to use this faster Miller-Rabin code when we look at the RSA algorithm in a later installment. For now, I have a couple of easy tasks for you.

Tasks

i) Twin Primes. Twin primes are consecutive odd numbers that are prime (for instance 101 and 103). It is conjectured (ie. not proved) that there are an infinite number of twin primes. Adapt Miller_Rabin so that you can type: “t,” followed by a number and get the first pair of twin-primes higher than that number.

ii) The Goldbach Conjecture. This states that any even number is the sum of two primes. Adapt the code so that the user can type: “g,” followed by an even number and be given a pair of primes that add up to that number. (You may like to test that the number given was even and tell the user if it was not!)

iii) Triple Primes[1]. It is also conjectured that there are an infinite number of triple primes. Adapt the program so that users can look for triple primes also.

Lucas-Lehmer Number

The Great Internet Mersenne Prime Search (GIMPS) uses a very fast algorithm to deterministically test a number for primality. In other words, numbers that pass the test are definitely prime.

What follows is a simple version of a Lucas test in Python. This is not optimised in any way and will be very slow for large numbers, however it does illustrate the Lucas-Lehmer method.

A Mersenne Number is prime *only* if it divides the corresponding Lucas number.

Here's how to find the Lucas number. L_2 is 4 and each number in the series is formed as follows:

$$\mathbf{L_n = L_{n-1}^2 - 2}$$

```
# lucas.py
# Lucas-Lehmer numbers to test Mersenne Primes.

def main():
    print("*** Lucas-Lehmer numbers ***")
    while 1:
        n = int(input("\nEnter a prime: "))
        if n < 2:
            break
        mer = mersenne(n)
        print("\nMersenne(" + str(n) + ") = " + str(mer))
        luc = lucas(n)
        print("Lucas-Lehmer(" + str(n) + ") = " + str(luc))
        print(divides(luc, mer))

def mersenne(p):
    """Returns the Mersenne Number generated from p."""
    return 2 ** p -1

def divides(x, y):
    """Returns true if y divides x."""
    return x % y == 0

def next_lucas(p):
    return p * p - 2

def lucas(n):
    """Returns the Lucas_Lehmer number of n."""
    k = 4
    if n < 2:
        return -1
    if n != 2:
        for i in range(3, n + 1):
            k = next_lucas(k)
    return k

if __name__ == "__main__":
    main()
```

[1] If you find the material on Primes interesting, I can highly recomment Marcus du Sautoy's book: *The Music of the Primes*.

So, if we run our program for the first couple of Mersenne Primes, we get this output:

```
*** Lucas-Lehmer numbers ***

Enter a prime: 3
Mersenne(3) = 7
Lucas-Lehmer(3) = 14
True

Enter a prime: 5
Mersenne(5) = 31
Lucas-Lehmer(5) = 37634
True

Enter a prime: 7
Mersenne(7) = 127
Lucas-Lehmer(7) = 2005956546822746114
True
```

As you can see, the Lucas_Lehmer numbers get very big, very quickly. The code featured here is not a practical way of testing large Mersenne Numbers. If you are interested in the optimised version that is used by GIMPS, visit www.mersenne.org.