Factoring with Python – continued
The Lucas-Lehmer test
Spring 2015

1 Mersenne primes

A *Mersenne prime* is a prime of the form $2^n - 1$. The first few are

- $M_2 = 2^2 - 1 = 3$
- $M_3 = 2^3 - 1 = 7$
- $M_5 = 2^5 - 1 = 31$
- $M_7 = 2^7 - 1 = 127$

It's easy to prove that $M_n = 2^n - 1$ can only be prime when n itself is prime. Lest you be carried away, M_{11} is *not* prime, so the converse is false.

The largest known prime at any time is pretty much guaranteed to be a Mersenne prime, because there's a (relatively) quick way to test whether M_p is prime. As of May, 2015 that's the 17,425,170 digit number $2^{57,885,161} - 1$. For more information about current work finding Mersenne primes, see GIMPS, the Great Internet Mersenne Prime Search, at `http://www.mersenne.org/`.

For general information on primes see Chris Caldwell's prime pages at `https://primes.utm.edu/`.

2 The Lucas Lehmer test

Here's the fast algorithm for determining whether M_p is prime, shamelessly stolen from `http://en.wikipedia.org/wiki/Lucas%E2%80%93Lehmer_primality_test`

The Lucas–Lehmer test works as follows. Let $M_p = 2^p - 1$ be the Mersenne number to test with p an odd prime. The primality of p can be efficiently checked with a simple algorithm like trial division since p is exponentially smaller than M_p. Define a sequence s_i for all $i \geq 0$ by

$$s_i = \begin{cases} 4 & \text{if } i = 0; \\ s_{i-1}^2 - 2 & \text{otherwise.} \end{cases}$$

The first few terms of this sequence are $4, 14, 194, 37634, \ldots$ ... (sequence A003010 in OEIS, the Online Encyclopedia of Integer Sequences, at `https://oeis.org/`).

Then M_p is prime if and only if

$$s_{p-2} \equiv 0 \pmod{M_p}.$$

The number $s_p - 2 \pmod{M)_p}$ is called the Lucas–Lehmer residue of p. (Some authors equivalently set $s_1 = 4$ and test $s_p - 1 \pmod{)Mp}$. In pseudocode, the test might be written as

```
// Determine if Mp = 2p - 1 is prime
Lucas{Lehmer(p)
    var s = 4
    var M = 2p - 1
    repeat p - 2 times:
        s = ((s * s) - 2) mod M
    if s = 0 return PRIME else return COMPOSITE
```

3 Lucas Lehmer in Python

From `http://rosettacode.org/wiki/Lucas-Lehmer_test#Python`

```
from sys import stdout
from math import sqrt, log

def is_prime ( p ):
  if p == 2: return True # Lucas—Lehmer test only works on odd primes
  elif p <= 1 or p % 2 == 0: return False
  else:
    for i in range(3, int(sqrt(p))+1, 2 ):
      if p % i == 0: return False
    return True

def is_mersenne_prime ( p ):
  if p == 2:
    return True
  else:
    m_p = ( 1 << p ) - 1
    s = 4
    for i in range(3, p+1):
      s = (s ** 2 - 2) % m_p
    return s == 0

precision = 20000   # maximum requested number of decimal places of 2 ** MP—1 #
long_bits_width = precision * log(10, 2)
upb_prime = int( long_bits_width - 1 ) / 2    # no unsigned #
# upb_count = 45      # find 45 mprimes if int was given enough bits #
upb_count = 15      # find 45 mprimes if int was given enough bits #

print (" Finding Mersenne primes in M[2..%d]:"%upb_prime)

count=0
# for p in range(2, upb_prime+1):
for p in range(2, int(upb_prime+1)):
  if is_prime(p) and is_mersenne_prime(p):
    print("M%d"%p),
    stdout.flush()
    count += 1
  if count >= upb_count: break
print
```

What's interesting here:

- **elif**
- bit manipulation
- no need to optimize `is_prime()`

4 Lucas Lehmer in Python faster

```
ef isqrt(n):
    if n < 0:
        raise ValueError
    elif n < 2:
        return n
    else:
        a = 1 << ((1 + n.bit_length()) >> 1)
        while True:
            b = (a + n // a) >> 1
            if b >= a:
```

```
11                 return a
12             a = b
13
14 def isprime(n):
15     if n < 5:
16         return n == 2 or n == 3
17     elif n%2 == 0:
18         return False
19     else:
20         r = isqrt(n)
21         k = 3
22         while k <= r:
23             if n%k == 0:
24                 return False
25             k += 2
26         return True
27
28 def lucas_lehmer_fast(n):
29     if n == 2:
30         return True
31     elif not isprime(n):
32         return False
33     else:
34         m = 2**n - 1
35         s = 4
36         for i in range(2, n):
37             sqr = s*s
38             s = (sqr & m) + (sqr >> n)
39             if r >= m:
40                 s -= m
41             s -= 2
42         return s == 0
```

What's interesting here:

- `&` operator
- read line 38: `s = (sqr & m) + (sqr >> n)`
- no division (no modular arithmetic)

Here is the LaTeX source for this document. You can cut it from the pdf and use it to start your answers. I used the `\jobname` *macro for the source file name, so you can call your file by any name you like.*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
\documentclass[10pt]{article}
\usepackage[textheight=10in]{geometry}

\usepackage{verbatim}
\usepackage{amsmath}
\usepackage{amsfonts} % to get \mathbb letters

\usepackage[utf8]{inputenc}
\DeclareFixedFont{\ttb}{T1}{txtt}{bx}{n}{9} % for bold
\DeclareFixedFont{\ttm}{T1}{txtt}{m}{n}{9}  % for normal
% Defining colors
\usepackage{color}
\definecolor{deepblue}{rgb}{0,0,0.5}
\definecolor{deepred}{rgb}{0.6,0,0}
\definecolor{deepgreen}{rgb}{0,0.5,0}

\usepackage{listings}

%Python style from
%http://tex.stackexchange.com/questions/199375/problem-with-listings-package-for-python-syntax-colo
\newcommand\pythonstyle{\lstset{
  language=Python,
  backgroundcolor=\color{white}, %%%%%%%
  basicstyle=\ttm,
  otherkeywords={self},
  keywordstyle=\ttb\color{deepblue},
  emph={MyClass,__init__},
  emphstyle=\ttb\color{deepred},
  stringstyle=\color{deepgreen},
  commentstyle=\color{red},  %%%%%%%%
  frame=tb,
  showstringspaces=false,
  numbers=left,numberstyle=\tiny,numbersep =5pt
}}
\usepackage{fancyvrb}
\usepackage{hyperref}

\begin{document}

\pythonstyle{}
\begin{center}
\Large{
Factoring with Python -- continued \\
The Lucas-Lehmer test \\
Spring 2015
}
\end{center}


\section{Mersenne primes}

A \emph{Mersenne prime} is a prime of the form $2^n -1$. The first few
are
\begin{itemize}
\item $M_2 = 2^2 -1 = 3$
```

```
\item $M_3 = 2^3 -1 = 7$
\item $M_5 = 2^5 -1 = 31$
\item $M_7 = 2^7 -1 = 127$
\end{itemize}

It's easy to prove that $M_n = 2^n -1$ can only be prime when $n$
itself is prime. Lest you be carried away, $M_{11}$ is \emph{not}
prime, so the converse is false.

The largest known prime at any time is pretty much guaranteed to be a
Mersenne prime, because there's a (relatively) quick way to test
whether $M_p$ is prime. As of May, 2015 that's the
17,425,170 digit number $2^{57,885,161}-1$. For more information about
current work finding Mersenne primes, see GIMPS, the Great Internet
Mersenne Prime Search, at \url{http://www.mersenne.org/}.

For general information on primes see Chris Caldwell's prime pages at
\url{https://primes.utm.edu/}.

\section{The Lucas Lehmer test}

Here's the fast algorithm for determining whether $M_p$ is prime,
shamelessly stolen from
\url{http://en.wikipedia.org/wiki/Lucas%E2%80%93Lehmer_primality_test}

The Lucas{Lehmer test works as follows. Let $M_p = 2^p -1$ be the
Mersenne number to test with $p$ an odd prime. The primality of $p$ can be
efficiently checked with a simple algorithm like trial division since
$p$ is exponentially smaller than $M_p$. Define a sequence $s_i$ for all
$i \ge 0$ by
%
\begin{equation*}
    s_i= \begin{cases} 4 & \text{if }i=0; \\ s_{i-1}^2-2 &
      \text{otherwise.} \end{cases}
\end{equation*}

The first few terms of this sequence are $4, 14, 194, 37634, \dots$
... (sequence A003010 in OEIS, the Online Encyclopedia of Integer
Sequences, at \url{https://oeis.org/}).

Then $M_p$ is prime if and only if
\begin{equation*}
    s_{p-2} \equiv 0 \pmod{M_p}.
\end{equation*}

The number $s_p - 2 \pmod M_p$ is called the Lucas{Lehmer residue of
$p$. (Some authors equivalently set $s_1 = 4$ and test $s_p-1 \pmod _Mp$). In
pseudocode, the test might be written as

\begin{verbatim}
// Determine if Mp = 2p - 1 is prime
Lucas{Lehmer(p)
    var s = 4
    var M = 2p - 1
    repeat p - 2 times:
        s = ((s * s) - 2) mod M
    if s = 0 return PRIME else return COMPOSITE
\end{verbatim}
%\lstinputlisting{croft.py}
```

```
\section{Lucas Lehmer in Python}

From \url{http://rosettacode.org/wiki/Lucas-Lehmer_test#Python}

\lstinputlisting{lucaslehmer1.py}

What's interesting here:

\begin{itemize}
\item \lstinline!elif!
\item bit manipulation
\item no need to optimize \lstinline!is_prime()!
\end{itemize}


\section{Lucas Lehmer in Python faster}

\lstinputlisting{lucaslehmer2.py}

What's interesting here:

\begin{itemize}
\item \lstinline!&! operator
\item read line 38: \lstinline!s = (sqr & m) + (sqr >> n)!
\item no division (no modular arithmetic)
\end{itemize}


\newpage

\emph{
Here is the \LaTeX{} source for this document. You can cut it from the
pdf and use it to start your answers. I used the} \verb!\jobname!
\emph{macro for the source file name, so you can call your file by any
  name you like.}
\verbatiminput{\jobname}

\end{document}
```