Notes on Ada 2022 Programs

These two programs illustrate the use of Big_Integers which were introduced into Ada 2022 and are described in Section A23.4a. The first program is a demonstration of the RSA algorithm for encoding. The second uses the Lucas–Lehmer test for checking whether a Mersenne number is prime or not.

RSA Algorithm

This algorithm was devised by Ronald Rivest, Adi Shamir, and Leonard Aldeman in 1977 and so is known as the RSA algorithm. A key point is that it is very easy to multiply two large prime numbers together but very hard in general to find the factors of a large number.

The essence of the algorithm is as follows. Choose two prime numbers p and q and multiply them together to give $n = p \times q$. The next stage is to choose e such that e is less than and relatively prime to $m = (p - 1) \times (q - 1)$. The numbers n and e comprise the public key and encryption is performed by converting a value v using the formula

$$c = v^e \bmod n$$

The code value c is then the coded message. Decryption is performed using the unique secret number d which is such that $e \times d \equiv 1 \bmod m$ according to the similar formula

$$v = c^d \bmod n$$

Simple text messages can be encoded by for example assigning values 1 to 26 to the upper case letters A to Z and 27 to 52 for the lower case letters with 0 representing a space and constructing the number in base 53. Thus CAT can be encoded as $3\times53^2 + 1\times53^1 + 20\times53^0 = 3\times2809 + 53 + 20 = 8500$.

As an illustration suppose we have chosen the primes p and q as 613 and 719 so that $n = 440747$ and $m = 439416$. Now $439417 = 11 \times 43 \times 929$ so we can take $\underline{e} = 929$ and $d = 473$. Using these values 8500 encrypts to 320793. We could send this as the encrypted message but it is perhaps more interesting to convert it into a string which using base 53 gives BHJk.

Note that Cat becomes ACMm and cat becomes AHcG. If we choose $e = 473$ and $d = 929$ then we find that CAT becomes AAKh, Cat becomes e_F (where _ denotes a space, remember that a space encodes as zero) and cat becomes eoZ.

The demonstration program asks for values for p and q. These can be provided as a simple integer (but not a Big Integer) such as 613 or as a Mersenne number written as M13 or m13 where the Mersenne number M_n is $2^n - 1$. So $M_{13} = 8191$. Using Mersenne numbers is a convenient way of entering a largish prime. The following Mersenne numbers are prime

$M_2 = 3$
$M_3 = 7$
$M_5 = 31$
$M_7 = 127$
$M_{13} = 8\_191$
$M_{17} = 131\_071$
$M_{19} = 524\_287$
$M_{31} = 2\_147\_483\_647$
$M_{61} = 2\_305\_843\_009\_213\_693\_951$
$M_{89} = 618\_970\_019\_642\_690\_137\_449\_562\_111$
$M_{107} = 162\_259\_276\_829\_213\_363\_391\_578\_010\_288\_127$
$M_{127} = 170\_141\_183\_460\_469\_231\_731\_687\_303\_715\_884\_105\_727$

An interesting Mersenne number is M_{67}.which Mersenne himself thought was prime but was shown by Cole in 1903 to be the product of two numbers which are themselves both prime, thus

M_{67} = 147_573_952_589_676_412_927 = 193_707_721 × 761_838_257_287

Having given the demonstration program values for p and q, it echoes them in confirmation; if they were given as Mersenne numbers they are echoed as big integers. The program then calculates and displays the public key $n = p \times q$ and $m = (p–1) \times (q–1)$.

The program then asks for a value for the encoding key e. It checks that it is relatively prime to m and if it is not, it asks for a new value for e until it is satisfied.

It then computes d such that $e \times d \equiv 1 \bmod m$ and displays the computed value of d. The calculation of d is a sort of inverse mod operation.

The program is now in a state ready to encrypt a message. The message can either be just a number or a sequence of letters. So it asks whether you want to use a numeric or alpha message. It expects a reply of N or A (upper or lower case).

If a numeric message is to be given then the program outputs the message

"Preparing for a numeric message not exceeding n"

where n is the public key $p \times q$.

This is followed by

"Message is "

and it then expects a numeric message which can be supplied using the same format as for the values of p and q, that is either an integer or a Mersenne number. If the value given is zero, then a farewell message is output and the program goes back to the beginning thus enabling other values of p and q to be tried.

If an alpha message is to be given then the program outputs the messages

"Preparing for a text message with max length M"

"Include spaces and letters only."

This is followed by

"Message is "

and it then expects an alpha message where M is such that the encoded value will not exceed the public key. If a longer message is supplied the additional characters are simply ignored. Alphabetic characters are accepted in both upper and lower case. Any unexpected character terminates the message and is crudely encoded as 99.

In both cases the program then says

"The encrypted message is"

which is followed by the appropriate encrypted form.

The program then says

"Now ready to decrypt your message"

and awaits for a couple of newlines to trigger the decryption.

It finally outputs

"The decrypted message is"

which with luck is followed by the original message.

In the case of an alpha message, if the message supplied was too long then it is simply truncated.

In the case of a numeric message, if the number exceeds the public key n, the value is taken mod n.

If the values given for p and q are not prime then decryption of the encrypted message usually does not return the original message.

Most of the program is straightforward but the computation of d such that $e \times d \equiv 1 \bmod m$ by the function Inverse_Mod is interesting. It is essentially the traditional Euclidean algorithm but the iteration is also unwound. This is explained in *Nice Numbers* by the author in the section entitled Linear Congruences.

I am particularly grateful to Jeff Cousins for his assistance in converting the program so that it does actually work using an Ada 2022 compiler from AdaCore.

Readers might like to improve the program by for example introducing subprograms Put and Get for manipulating Big Integers. Note that Put_Num does essentially do the job using a string as an intermediary but it would be nice to have underscores every third digit from the origin.

Historical note. The program has its origins in an Ada 83 program written in about 1992 using a home-brewed multilength integer system.

Perfection

This addresses two issues. How to check whether a Mersenne number is a prime number and also the relationship between Mersenne primes and perfect numbers.

Determining whether a large number is a prime is usually a tedious and generally unsolved problem. However, in the special case of Mersenne numbers there is a simple and perfect test. The theory was developed by Edouard Lucas (1842–1891) and a practical test was devised by Derrick Lehmer (1905–1991).

It goes as follows. Form the series of numbers

$$L_{i+1} = (L_i)^2 - 2, \text{ starting with } L_2 = 4$$

We get $L_2 = 4$, $L_3 = 14$, $L_4 = 194$, $L_5 = 37634$, and so on

The amazing fact is that M_p is prime if and only if L_p is exactly divisible by M_p . Thus since M_3 is 7 and L_3 is 14, we find that M_3 is prime.

The trouble with the test is that the numbers get huge very soon. In the case of M_{11} which is 2047 and not prime since $2047 = 23 \times 89$, we find that L_{11} has 293 digits. If we divide L_{11} by M_{11} we get a remainder of 1736 confirming that indeed M_{11} is not prime.

The difficulty can be overcome by doing modulo arithmetic. To check out M_{11} we do all the arithmetic mod M_{11} and then L_{11} is simply 1736 with just 4 digits.

Another interesting point regarding Mersenne primes is that if M_p is prime then p is also prime. But the opposite is not true since M_{11} is not prime as mentioned above.

The other relationship concerns perfect numbers. Remember that a perfect number is one whose factors add up to the number itself. The first perfect number is 6 and the next two are 28 and 496 thus

$6 = 1+2+3$ $28 = 1+2+4+7+14$ $496 = 1+2+4+8+16+31+62+124+248$

Strangely enough there is a close relationship between perfect numbers and Mersenne primes. Every Mersenne prime has an associated perfect number and vice versa.

It is fairly easy to show that all even perfect numbers are of the form

$$P_k = 2^{k-1} \times (2^k - 1)$$

where $(2^k - 1)$ is a Mersenne prime. So we have

$k = 2$	$M_2 = 3$	$P_2 = 2 \times M_2 = 6$
$k = 3$	$M_3 = 7$	$P_3 = 4 \times M_3 = 28$
$k = 5$	$M_5 = 31$	$P_5 = 16 \times M_5 = 496$
$k = 7$	$M_7 = 127$	$P_7 = 64 \times M_7 = 8128$
$k = 13$	$M_{13} = 8191$	$P_{13} = 4096 \times M_{13} = 33\_550\_336$
$k = 17$	$M_{17} = 131\_071$	$P_{17} = 65\_536 \times M_{17} = 8\_589\_869\_056$

The demonstration program is quite simple. It includes a table of the first 30 odd prime numbers thus

(3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
61, 67, 71, 73, 79, 83, 89, 97,101, 103, 107, 109, 113, 127)

and will analyse a range of these numbers. After a message of greeting it asks for values for Start loop and End loop. Thus given the values 3 and 4, it will process the prime numbers 7, and 11. It will grumble if the values supplied are unacceptable and ask again.

It then outputs the message

"Level of detail required, answer 1, 2, or 3"

If the answer given is not 1, 2, or 3 it just repeats the message.

Level 1 outputs a message saying whether the Mersenne number is prime or not and if it is prime it outputs the value of the corresponding perfect number. Thus we get

7 : 127 is prime
8128 is perfect

11 : 2047 is not prime

Level 2 also outputs the result of the Lucas Lehmer analysis using the version with modulo arithmetic. Thus in the case of the prime number 7, we also get

L is 12319
equals 127 times
97

and in the case of 11 we get

L is 79522
equals 2047 times
38
remainder = 1736

Level 3 give the full works. In the case of the prime number 7 we get

L is 2005956546822746114
equals 127 times
15794933439549182

and in the case of 11 we get

L is 68729 ... 203714 (293 digits)
equals 2047 times
33575 ... 341574 (290 digits)
remainder = 1736

For large primes such as 127 it is best not to use level 3. But it does work and gives the perfect number corresponding to M_{127} as

14_474_011_154_664_524_427_946_373_126_085_988_481_573_677_491_
474_835_889_066_354_349_131_199_152_128

Enjoy!!

PS This is all described in Lecture 2 of *Nice Numbers*. And again many thanks to Jeff Cousins for checking that the program does work.

May 2022