Solving xa = b (mod c) for x and Undecimating Recursions

[redacted] (b) (3) -P.L. 86-36

(U) There are many times when the ability to undo the effects of decimation on a linear recursive sequence would be of great value to the analyst. This is a thorough look at the problem of undecimating, with historical notes and several examples of important calculational techniques. While undecimating can now be done with software, this rather complete tutorial offers a worthwhile historical perspective of the process.

INTRODUCTION

(U) The ring of integers modulo c has long been a most lucrative area for mathematical recreations. On occasion, these mathematical diversions can actually be useful in cryptanalysis and signals analysis, like the solution of xa = b (mod c) described here. The approach will be to first describe the general problem of solving xa = b (mod c) for x. Then this will be extended to the special case where b = 1 and $c = 2^n - 1$ for some integer n (that is, c is the cycle length of a primitive recursion of degree n), making the process very near to undoing a decimation of a primitive linear recursive sequence.

~~(TSC)~~ [redacted]

~~(TSC)~~ [redacted]

~~(SC)~~ Once upon a time, the degrees of polynomials used in communications equipment were of such low degree that decimating and undecimating the polynomials were

(b) (1)
(b) (3) -18 USC 798
(b) (3) -P.L. 86-36

easily done with a glance at [redacted] (Peterson's) table [1,2]. However, modern equipment rarely uses degree 4 and 5 recursions, with the degree of even randomizers often in the 20s.

~~(TSC)~~ [redacted]

(b) (1)
(b) (3)-18 USC 798
(b) (3)-P.L. 86-36

SOLVING XA = B (MOD C) FOR X

(U) At the risk of presenting this algorithm less than optimally, I will build it from scratch . . . that is, from the point where one first scratches one's head and ponders it. The first question to ask is, "Does it even have a solution?" And a good question it is, too, for one is not always guaranteed an x that will solve xa = b (mod c). For example, it is easily seen that x*2 = 3 (mod 4) cannot be solved. Begging the reader's pardon, I will expand this simplistic example to show fully that there is no possible x that works. In fact, the integers modulo 4 contain only four possible replacements for x. These are 0, 1, 2, and 3. Now,

0*2 = 0 (mod 4),

1*2 = 2 (mod 4),

2*2 = 4 = 0 (mod 4),

and 3*2 = 6 = 2 (mod 4).

(U) Of course, the classical result is that a solution to the equation xa = b (mod c) is guaranteed as long as (a,c) divides (b,c), where (,) is the standard notation for the greatest common divisor function. That is, if (a,c) = 1, an x that solves xa = b (mod c) is guaranteed. If (a,c) > 1, a solution will exist only when this factor can be divided out of all three terms a, b, and c; thus the solution is the guaranteed x' that solves x'*a' = b' (mod c').

(U) The first method one comes to is what I call the hard way. That is, exhaustively test all positive integers 1, 2, . . . , c-1 until getting one that works. That is essentially what took place in the earlier example where all four integers mod 4 were checked to show there was no x that solved x*2 = 3 (mod 4). Seriously, though, zero need never be checked, so the worst case for the exhaustive method is c-1 multiplications to perform. This is fine

for c = 4, but I wouldn't care to try this method for x*11 = 19 (mod 281) or (worse) x*701 = 44 (mod 1993). So there must be a better way, and there is.

(U) The second method is a clever way of approaching this exhaustion problem. It really isn't worthwhile to check every possible integer as x. Only certain ones have a chance, and one way to write the form of those that might work is k[c/a]+n where [ ] denotes the greatest integer function, k = 1, 2, . . . , a-1 and n = [b/a]+1 initially with some modification necessary as k increases. This modification to n will become clear in the example and the next method, but the importance of this method is that it reduces an exhaustion over c-1 integers to an exhaustion over a-1 integers. I still wouldn't be happy with this for solving x*701 = 44 (mod 1993), but it is worthwhile doing an example for, say, x*11 = 19 (mod 281).

[c/a] = [281/11] = 25

[b/a]+1 = [19/11]+1 = 1+1 = 2

correct interval: (11-21)

k=1, n=2: k[c/a]+n = 25+2 = 27, 27*11 = 297 = 16 (mod 281)

k=2, n=2: k[c/a]+n = 50+2 = 52, 52*11 = 572 = 10 (mod 281)

n=3: adjust n to 3 53, 53*11 = 583 = 21 (mod 281)

k=3, n=3: k[c/a]+n = 75+3 = 78, 78*11 = 858 = 15 (mod 281)

k=4, n=3: k[c/a]+n = 100+3 = 103, 103*11 = 1133 = 9 (mod 281)

n=4: adjust n to 4 104, 104*11 = 1144 = 20 (mod 281)

k=5, n=4: k[c/a]+n = 125+4 = 129, 129*11 = 1419 = 14 (mod 281)

k=6, n=4: k[c/a]+n = 150+4 = 154, 154*11 = 1694 = 8 (mod 281)

n=5: adjust n to 5 155, 155*11 = 1705 = 19 (mod 281)

so x = 155

and 155*11 = 1705 = 19 + (6*281) = 19 (mod 281)

(U) This example shows the real reason for the form of the numbers to be tried. With the greatest integer of c/a, one picks up the number of full multiples of "a" needed to get to "c." The "+1" term of n moves past c and the [b/a] moves the appropriate number of full multiples of "a" beyond this to get to the correct interval to have a chance. The adjustments to n must be made whenever the tested number drifts out of this correct interval.

(U) The third method is related to the second method, but is an even more clever way of doing it. Instead of exhausting the possible x's of the form k[c/a]+n, only the first two need to be done and the difference will point to the correct solution through (mod a) arithmetic. Say

$([c/a]+n)*a = d_1 + c$

and $(2*[c/a]+n)*a = d_2 + 2c$,

then $d = d_2 - d_1$ is the difference (note: because of the greatest integer function, $-a < d < 0$). The set of numbers (mod a) generated by $d_1 + kd$, with n incremented and a added each time the sum goes negative, is checked until one of these numbers is $d_b = b \pmod a$. The even more clever part of the algorithm is that it has turned the multiplicative problem (mod c) into an additive problem (mod a). It also has the possibility of being applied recursively [e.g., in $d_1 + kd = d_b \pmod a$ the right k is the x' that solves the equation $x'*(-d) = (d_1-d_b) \pmod a$]. It is not worthwhile to pursue this recursive definition at present because there are still better ways, but the same example, $x*11 = 19 \pmod{281}$ is beneficial.

$[c/a] = [281/11] = 25$

$[b/a]+1 = [19/11]+1 = 1+1 = 2$

$k=1, n=2$: $k[c/a]+n = 25+2 = 27$, $27*11 = 297 = 16 + 281$

$k=2, n=2$: $k[c/a]+n = 50+2 = 52$, $52*11 = 572 = 10+(2*281)$

$[d_b = 19 = 8 \pmod{11}]$	$d = 10-16 = -6$
$d_1 = 16 = 5 \pmod{11}$	$k=1, n=2$
$d_1 + d = -1 = 10 \pmod{11}$	$k=2, n=3$
$d_1 + 2d = 4 \pmod{11}$	$k=3, n=3$
$d_1 + 3d = -2 = 9 \pmod{11}$	$k=4, n=4$
$d_1 + 4d = 3 \pmod{11}$	$k=5, n=4$
$d_1 + 5d = -3 = 8 \pmod{11}$ *	$k=6, n=5$ *

for $k=6, n=5$, $x = k[c/a]+n = (6*25)+5 = 155$

and $155*11 = 1705 = 19 + (6*281) = 19 \pmod{281}$

(U) The third method is well on the way to becoming the classical solution, the Euclidean Algorithm. Through the hinted recursive possibility, the repetitive divisions in the Euclidean Algorithm are simulated. Given two numbers (a and c in this case), the Euclidean Algorithm finds integers s and t such that $sa + tc = 1$. Thus, $bs \pmod c$ is a solution to $xa = b \pmod c$.

(U) Since the s is the only important integer to be found for this application (working mod c), the following slight modification of the Extended Euclidean Algorithm can be used here. The motivation for this particular form is that it is tailored to the case of undecimating recursions, so it is used here despite the slightly nonstandard form. This form of the Euclidean Algorithm specifically gives the multiplicative inverse for a, mod c. In the equation $xa = b \pmod c$, a will always be less than c, so let $R_0 = c$ and $R_1 = a$ (the general condition is that $R_0 > R_1 > 1$). $S_0 = 0$ and $S_1 = 1$. For $i = 1, 2, 3, \ldots$ (as needed)

$Q_{i+1} = [R_{i-1}/R_i]$, the greatest integer in this quotient,

$R_{i+1} = R_{i-1} - (Q_{i+1} * R_i)$, the remainder of the quotient,

$S_{i+1} = S_{i-1} - (Q_{i+1} * S_i)$.

If $R_{i+1} = 0$, R_0 and R_1 are not relatively prime (no inverse exists and R_i is the greatest common divisor).

If $R_{i+1} = 1$, then S_{i+1} is the multiplicative inverse of R_1.

[note: if $S_{i+1} < 0$, $S_{i+1} + R_0$ also works]

If $R_{i+1} > 1$, another step is needed.

[comment: at any step, $S_{i+1} * R_1 = R_{i+1} \pmod{R_0}$]

Examples:

$x*11 = 19 \pmod{281}$

	Q	R	S
i=0	-	281	0
i=1	-	11	1
i=2	25	6	-25
i=3	1	5	26
i=4	1	1	-51
			+281
			230

check:

$230*11 = 2530 = 1 + (9*281)$

$19*230 = 4370 = 155 + (15*281)$

so $x = 155$

and $155*11 = 1705 = 19 + (6*281)$

$x*701 = 44 \pmod{1993}$

	Q	R	S
i=0	-	1993	0
i=1	-	701	1
i=2	2	591	-2
i=3	1	110	3
i=4	5	41	-17
i=5	2	28	37
i=6	1	13	-54
i=7	2	2	145
i=8	6	1	-924
			+1993
			1069

check:

$1069*701 = 749369 = 1 + (376*1993)$

$44*1069 = 47036 = 1197 + (23*1993)$

so $x = 1197$

and $1197*701 = 839097 = 44 + (421*1993)$.

(U) With this algorithm, solving $xa = b \pmod{c}$ is fairly straightforward. It is time to move on to the special case where $b = 1$ and $c = 2^n - 1$ for some integer n.

UNDOING DECIMATIONS

(U) A decimated primitive linear recursive sequence can be undecimated by further decimating by the multiplicative inverse (mod the cycle length) of the initial decimation. That is, the effect of the two decimations is a decimation by 1 (the end decimation is a product of the multiplicative inverses). This requires that the initial decimation has a multiplicative inverse, or the decimation cannot be undone uniquely. When 2^n-1 is not a prime, the integers modulo 2^n-1 is not a field, only a ring – which means there are elements that do not have multiplicative inverses. Notably, these are the elements that are not relatively prime to 2^n-1. In terms of decimations, decimating by one of these amounts shortens the cycle length, and this can not be uniquely backed up. For example, the seventh decimation of a bit stream satisfying (0,1,6) with cycle length 63=7*9 is a 9-long cycle satisfying (0,3,6), an imprimitive irreducible. However, starting with a bit stream satisfying (0,3,6) and decimating by 7 also gives a 9-long cycle satisfying (0,3,6). So the decimation by 7 in this case cannot be uniquely undone. Additionally, there is no way to generate all 63 bits of the primitive's cycle from the 9 bits remaining in the decimated stream if the former is the case.

(U) Furthermore, if the 7th decimation of something yields (0,1,6) with cycle length 63, then the something must have a cycle length of at least 63*7 = 441 if it is recursive. In fact, (0,7,42) is a factor of a recursive something whose 7th decimation is (0,1,6) (but (0,7,42) may not be irreducible, so factoring programs may not show it this obviously). Again, there is no unique answer for undoing the the decimation. In particular, the something is not a primitive degree 6 recursion, since the 7th decimation of one of those no longer has cycle length 63. These examples illustrate the difficulties in undecimating by "a" only when "a" and 2^n-1 are not relatively prime.

(U) An answer is guaranteed only when the recursion is primitive and the width (or decimation) does not have a common factor with the cycle length (2^{degree}-1). It is still possible to undo decimations frequently for polynomials that are not primitives by treating the recursion as a product of irreducibles. This will be discussed further after the primitives case, so in the following discussion, all recursions are assumed to be primitive and (a,2^{degree}-1) = 1.

~~(SC)~~ The hard way described earlier still works, but is not worth considering, especially since cycle lengths of primitive polynomials get large very quickly as the degree increases. It doesn't take too large a degree before even the best computers are stifled by the hard way. The equivalent of a clever way and an even more clever way were the quick and easy methods offered in successive iterations of a short course on polynomials [3] that I taught a few years ago [redacted] For ease of programming and speed (additions are quicker than the divisions of the Euclidean Algorithm), this was still the method I chose for the software to undo decimations. For the case of undecimating recursions, b=1 and the starting n=1 always. The following examples are worthwhile in showing this method:

(b) (1)
(b) (3)-18 USC 798
(b) (3)-P.L. 86-36

(0,2,3,7,11) is found on width 5 (is the 5th decimation of something)

solve: x*5 = 1 (mod 2047)

[c/a]=[2047/5]=409

n=1

410*5=2050=3+2047

819*5=4095=1+2*2047

So x=819

Decimate (0,2,3,7,11) by 819 to find the base recursion whose 5th decimation is (0,2;3,7,11) this is: (0,2,11)

(0,7,10) is found down columns on width 40.

solve: x*40 = 1 (mod 1023)

[c/a]=[1023/40]=25

n=1

26*40=1040=17+1023

51*40=2040=-6+2*1023

d=-23

k=1, n=1 17

k=2, n=1 17-23=-6=34 (mod 40)

k=3, n=2 34-23=11

k=4, n=2 11-23=-12=28

k=5, n=3 28-23=5

k=6, n=3 5-23=-18=22

k=7, n=4 22-23=-1 (=39)

a "trick": since k=7, n=4 gives (7*25)+4=179

and 179*40 = -1 (mod 1023)

(-179)*40 must be 1 (mod 1023)

(-179) = 844 (mod 1023)

check: 844*40=33760=1+(33*1023)

So decimating (0,7,10) by 844 gives the base recursion whose 40th decimation is (0,7,10) this is: (0,3,7,9,10)

(U) The classical solution, the Euclidean Algorithm put forward in [4], could also be used to solve these problems. However, in practical applications the "a" is going to be small enough that perhaps the Euclidean Algorithm isn't any better. Especially when one considers using necklaces (residue classes formed by powers of 2) to get the smallest related entry, the even more clever way is often the quickest, most efficient method. For the second example above, the width 40 could have been replaced by 5 (5, 10, 20, 40, . . . are always on the same necklace), and a maximum of 5 additions is hard to beat. That example is reworked here with the even more clever way using a=5 and the Euclidean Algorithm for comparison.

solve: x*5 = 1 (mod 1023)	x*40 = 1 (mod 1023)		
[c/a]=[1023/5]=204	Q	R	S
n=1	-	1023	0
205*5=1025=2+1023	-	40	1
409*5=2045=-1+2*1023	25	23	-25
trick: (-409)*5=1 (mod 1023)	1	17	26
and (-409) =614 (mod 1023)	1	6	-51
check:	2	5	128
614*5=3070=1+(3*1023)	1	1	-179
	(-179) =844 (mod 1023)		

Decimate (0,7,10) by either 614 or 844 (both work the same) to get the recursion whose 40th decimation is (0,7,10). (b)(3)-P.L. 86-36

this is: (0,3,7,9,10)

(U) Another possibility for undecimating small degree polynomials is [redacted] (Peterson's) table. This method wasn't available for the xa=b (mod c) case because it is unique to polynomials. These tables list (as minimally as possible) all irreducible polynomials and a root of that polynomial. This is getting into some more advanced mathematics, but for the present purposes I will describe only what is needed to understand the undecimating process.

(U) Since the powers of the roots of degree n irreducible polynomials fall into the residue classes mentioned above, and these are equivalent to decimations, the first entries of the table for degree 11

1 4005E 3 4445E 5 4215E 7 4055E 9 6015G

show that 0,2,3,7,11 (4215) is the 5th decimation of 0,2,11 (4005). A diversion to describe the elements of the table and how to use them is beneficial before going on to the second example. The letter in the entry tells a number of things about the roots of the polynomial, the most important being that E, F, G, or H mean the polynomial is primitive. For further description of these letters, I refer the interested reader to Appendix C of [2]. The polynomials are entered in octal characters:

0=000 1=001 2=010 3=011 4=100 5=101 6=110 7=111.

Substituting these 3-bit values for the octal characters gives the bitmap of the polynomial taps; for example, 4005 becomes 100000000101 in bits and the 1s appear in positions 0, 2, 11 when counting from 0-up, right to left. It doesn't actually make much difference which direction one counts, since each entry in the table represents both a polynomial and its reverse (the polynomial with bitmap read the other direction). The integer in front of the polynomial tells a root of the polynomial based on w^1 being a root of the base recursion

4005. Thus w^3 is a root of 4445 or equivalently 4445 is the 3rd decimation of 4005, w^5 is a root of 4215 or 4215 is the 5th decimation of 4005, and so on. Once one root (or decimation) is known, they all are because of the necklaces described earlier. For example,

$$w^1, w^2, w^4, w^8, w^{16}, w^{32}, w^{64}, w^{128}, w^{256}, w^{512}, \text{and } w^{1024}$$

are the eleven roots of 4005. This is written in shorthand by only recording the power, and can easily be recognized as the fact that decimating a primitive lrs by a power of 2 gives the same polynomial. It is also noteworthy that w^{2048} is also a root, but $2048 = 1 \pmod{2^{11}-1}$ and it was already listed (the 2048th decimation of a 2047-long cycle is a decimation by 1). Some necklaces are

1 4005: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024

3 4445: 3, 6, 12, 24, 48, 96, 192, 384, 768, 1536, 3072 = 1025

5 4215: 5, 10, 20, 40, 80, 160, 320, 640, 1280, 2560 = 513, 1026.

The necklace of the reverse polynomial consists of complements (mod 2047, in this case) to the entries on the necklace for the polynomial. Hence

1R 5001: 2046, 2045, 2043, 2039, 2031, 2015, 1983, 1919, 1891, 1535, 1023

3R 5111: 2044, 2041, 2035, 2023, 1999, 1951, 1855, 1663, 1279, 511, 1022.

(U) This method is painless when the desired undecimation is actually the first listed polynomial in the table, and it is possible otherwise. The equation $x*a = 1 \pmod{2^{degree}-1}$ must be solved when the first polynomial in the table is not involved. The width 40 example (above) illustrates this.

Find the polynomial whose 40th decimation is (0,7,10)

(0,7,10) is the reverse of entry 1 in the degree 10

portion of Peterson's table: 1 2011E

so solve: $x*40 = 1 \pmod{1023}$

this has been done in the previous examples,

$x = -179$ (reverse has $x = 179$).

so if there is a 179 entry in Peterson's table

(and there is), it is the base recursion

179 3211G

So the 40th decimation of (0,3,7,9,10) gives (0,7,10).

(U) This was too easy, mostly because (0,7,10) is the first entry and all numbers were in Peterson's table without having to go through necklaces. A tougher example is beneficial. In fact, this example will have an imprimitive whose cycle is shortened by the proper amount from the decimation (hence there will be two possible answers):

The polynomial (0,1,2,3,5,6,7,8,11) is found

(U)

(U) One significant note on the necklace is that only the smallest entry from a necklace or its complement is listed. Unfortunately, the examples here never required using the complementary necklace, but the smallest element on the complementary necklace is easily found by (2^{degree}-1) minus the largest entry on the necklace

(U) Peterson's table seems to be much more widely available and it is exhaustive to degree 16 with selected polynomials listed up to degree 34. This limitation (based mostly on the space required to list all the irreducible polynomials as degree rises) makes this a hand calculation method only. It is not general enough to be much more than a fun exercise for mathematicians.

(U) is that it can undo the decimations that shorten cycles length (like the more difficult example above), although it gives a primitive and an imprimitive possibility to this problem that has no unique solution. In practical applications, a primitive polynomial is the most likely candidate for the base recursion. So

WHEN THE POLYNOMIAL IS NOT PRIMITIVE

(U) The previous arguments work for primitives and some imprimitive irreducibles, but the methods can be extended to reducible polynomials naturally. For reducibles, the

polynomial must be broken into its irreducible factors. Each of the factors must be undecimated, and the product of these undecimations is the base polynomial whose given decimation is the original reducible polynomial. As a final example, what is the polynomial whose 100th decimation is (0,2,19)?

First, factor (0,2,19):

$$(0,2,19) = (0,1,2) * (0,1,2,4,5,6,7) * (0,6,7,8,10)$$

Then undecimate each factor by 100 . . .

or better, by 25

[regardless of degree, 25, 50 and 100 are on the same necklace; in fact, 25 is already bigger than the cycle length of (0,1,2)]

the 25th decimation of (0,1,2) is (0,1,2)

the 25th decimation of (0,6,7) is (0,1,2,4,5,6,7)

the 25th decimation of (0,2,3,5,7,9,10) is (0,6,7,8,10)

$(0,1,13,14,15,18,19) = (0,1,2) * (0,6,7) * (0,2,3,5,7,9,10)$

So the 25th decimation of (0,1,13,14,15,18,19) is (0,2,19). This method of undecimating reducibles gives a unique solution only when all factors have a unique solution.

CONCLUSION

(U) Many polynomials, primitive, imprimitive or reducible, can be undecimated. For primitives, any of the methods discussed works, and these cannot be undecimated only when the decimation a and the cycle length $2^{degree}-1$ have a common factor. For imprimitives, if (a, $2^{degree}-1$) = 1 anything still works, if (a,cycle length) = 1 a method like the Peterson's table examples (perhaps having to extend it for degree) is required and two solutions are possible. Again there is no undecimation if (a,cycle length) > 1. For reducibles, the polynomial must be broken into its irreducible factors. Each of the factors must be undecimated, and the product of these undecimations is the base polynomial whose given decimation is the original reducible polynomial. Each time an irreducible factor has two solutions, both generate possible solutions. Any time a factor cannot be uniquely undecimated [(a,cycle length) > 1], the product cannot be uniquely backed up.

(U) There are always additional solutions of much higher degree (just interleave several different recursions), but the basic assumption here is that a simple recursion was decimated. And that very practical problem in both cryptanalysis and signals analysis is easily solved with the methods described here. Thankfully, there is software available that does most of the calculations presented here.

[redacted]

REFERENCES

[1] [redacted]

[2] Peterson, W. Wesley, *Error-Correcting Codes*, Cambridge, Massachusetts: The MIT Press, 1961. (Or 2nd edition, 1972.)

[3] [redacted] Course notes from "A Short Course on Polynomials," July 1984 and April 1985.

[4] [redacted] "A Tutorial on Converting a Distance Problem Solution to Another Polynomial of the Same Degree," H79/74/86, 28 Nov. 1986.

[5] [redacted] "Programming Note: undecim8 – Undecimating Recursions," to appear.

[6] [redacted]