Module MA2C03—Additional Notes

David R. Wilkins

April 2014

Contents

4 Computing Powers in Modular Arithmetic

4.1 Sequences of Elements of Finite Sets

Let A be a finite set, let $f\colon A \to A$ be a function from the set A to itself, and let x_0 be an element of A. Then the element x_0 of A and the function $f\colon A \to A$ determine an infinite sequence $x_0, x_1, x_2, x_3, \ldots$, where $x_n = f(x_{n-1})$ for all positive integers n.

Now the sequence $x_0, x_1, x_2, x_3, \ldots$ will become periodic at some point. The value of x_n for large values of n can then be determined once the periodic behavior of the sequence has been determined.

If the positive integer n is equal to or exceeds the number of elements of the set A then the elements $x_0, x_1, \ldots, x_n$ cannot be all distinct. It follows that there exists a positive integer q which is the smallest positive integer with the property that

$$x_0, x_1, x_2, \ldots, x_{q-1}$$

are distinct. Then $x_q = x_m$ for some integer m satisfying $0 \leq m < q$. Let $p = q - m$. Then $x_m = x_{m+p}$. But then $x_{m+1} = f(x_m) = f(x_{m+p}) = x_{m+p+1}$. Moreover if $x_{m+r} = x_{m+p+r}$ for some non-negative integer r then $x_{m+r+1} = f(x_{m+r}) = f(x_{m+p+r}) = x_{m+p+r+1}$. It follows easily by induction of k that

$x_{m+k} = x_{m+p+k}$ for all non-negative integers k. Moreover on applying this result with $k = p, 2p, 3p, \ldots$, we find that

$$x_m = x_{m+p} = x_{m+2p} = x_{m+3p} = x_{m+4p} = \cdots.$$

A straightforward proof by induction on n shows that $x_{m+kp} = x_m$ for all non-negative integers k. It follows that $x_{m+kp+r} = x_{m+r}$ for all non-negative integers k and r. Thus in order to determine x_n for large values of n, one should determine the index m where the periodicity of the sequence begins, and the smallest positive integer p for which $x_m = x_{m+p}$.

4.2 Of Example of Computing Powers in Modular Arithmetic

Let m be a positive integer, and let a be an integer satisfying $0 \leq a < m$. We wish to find the value of a^n modulo m for some fairly large value of n.

Let

$$I_m = \{n \in \mathbb{Z} : 0 \leq n < m\},$$

and let $f\colon I_m \to I_m$ be the function defined such that $f(x) \equiv ax$ modulo m for all $x \in I_m$. Let $x_0 = 1$ and let $x_n = f(x_{n-1})$ for all positive integers n. Suppose that $x_k \equiv a^k$ (mod. m) for some non-negative integer k. Then

$$x_{k+1} \equiv ax_k \equiv a^{k+1} \text{ (mod. } m).$$

It follows by induction on n that $x_n \equiv a^n$ modulo m for all non-negative integers n.

In order to determine a^n modulo m for large values of n, it suffices to determine the smallest index m such that the sequence $x_m, x_{m+1}, x_{m_2}, \ldots$ returns to x_m, and the smallest integer p for which $x_{m+p} = x_m$.

The following simple Python program computes x_n, where $0 \leq x_n < m$ and $x_n \equiv a^n$ modulo m, for values of n less than 30:

```
#!/usr/bin/env python

import sys

def print_powers(m,a):

    x = 1
    for i in range(0,30):
        print '%d raised to the power %d modulo %d is %d' % (a,i,m,x)
```

```
        x = (x * a) % m

m = int(sys.argv[1])
a = int(sys.argv[2])

print_powers(m,a)
```

Running this simple program with command line arguments 60 and 2 to compute succesive values of x_n when $m = 60$ and $a = 2$, we find that the values of x_n for $n \leq 10$ are as follows:—

```
2 raised to the power 0 modulo 60 is 1
2 raised to the power 1 modulo 60 is 2
2 raised to the power 2 modulo 60 is 4
2 raised to the power 3 modulo 60 is 8
2 raised to the power 4 modulo 60 is 16
2 raised to the power 5 modulo 60 is 32
2 raised to the power 6 modulo 60 is 4
2 raised to the power 7 modulo 60 is 8
2 raised to the power 8 modulo 60 is 16
2 raised to the power 9 modulo 60 is 32
2 raised to the power 10 modulo 60 is 4
```

Examining these values we see that $m = 2$ and $p = 4$. Thus $x_{2+4k+r} = x_{2+r}$ for all non-negative integers k and r. We conclude that

$$\begin{aligned} 2^n &\equiv 16 \quad \text{(mod. 60) when } n \geq 2 \text{ and } n \equiv 0 \text{ mod. 4;} \\ 2^n &\equiv 32 \quad \text{(mod. 60) when } n \geq 2 \text{ and } n \equiv 1 \text{ mod. 4;} \\ 2^n &\equiv 4 \quad \text{(mod. 60) when } n \geq 2 \text{ and } n \equiv 2 \text{ mod. 4;} \\ 2^n &\equiv 8 \quad \text{(mod. 60) when } n \geq 2 \text{ and } n \equiv 3 \text{ mod. 4.} \end{aligned}$$

Suppose for example that we wish to find 2^{2067} modulo 60. Now 2064 is divisible by 4. Therefore $2067 \equiv 3$ mod. 4. It follows that $2^{2067} \equiv 8$ modulo 60.

Suppose now we take $m = 360$ and $a = 54$. Thus we wish to compute 54^n modulo 360 for large values of n. We run the Python program with command line arguments 360 and 54. The values of 54^n modulo 360 for $n \leq 8$ are output as follows:—

```
54 raised to the power 0 modulo 360 is 1
54 raised to the power 1 modulo 360 is 54
54 raised to the power 2 modulo 360 is 36
```

```
54 raised to the power 3 modulo 360 is 144
54 raised to the power 4 modulo 360 is 216
54 raised to the power 5 modulo 360 is 144
54 raised to the power 6 modulo 360 is 216
54 raised to the power 7 modulo 360 is 144
54 raised to the power 8 modulo 360 is 216
```

We see that $54^n \equiv 216$ (mod. 360) when n is even and $n \geq 4$, and $54^n \equiv 144$ (mod. 360) when n is odd and $n \geq 3$.

4.3 Computing congruence classes of numbers of the form a^{2^k}.

Let m be a positive integer, and let a be an integer satisfying $0 \leq a < m$. We wish to find the value of a^{2^k} modulo m for some fairly large value of k.

Let

$$I_m = \{n \in \mathbb{Z} : 0 \leq n < m\},$$

and let $g\colon I_m \to I_m$ be the function defined such that $g(x) \equiv x^2$ modulo m for all $x \in I_m$. Let $y_0 = a$ and let $y_n = g(y_{n-1})$ for all positive integers n. Suppose that $y_j \equiv a^{2^j}$ (mod. m) for some non-negative integer j. Then

$$y_{j+1} \equiv (y_j)^2 \equiv (a^{2^j})^2 = a^{2\times 2^j} = a^{2^{j+1}} \pmod{m}.$$

It follows by induction on k that $y_k \equiv a^{2^k}$ modulo m for all non-negative integers k.

The following simple Python program can be used to compute a^{2^k} modulo m, for values of m and a specified as the first and second arguments on the command line:—

```
#!/usr/bin/env python

import sys

def print_twopowers(m,a):

    y = a
    for i in range(0,30):
        print '%d raised to the power (2^%d) modulo %d is %d' % (a,i,m,y)
        y = (y * y) % m

m = int(sys.argv[1])
```

```
a = int(sys.argv[2])

print_twopowers(m,a)
```

Running with $m = 60$ and $a = 2$ produces output that begins as follows:—

```
2 raised to the power (2^0) modulo 60 is 2
2 raised to the power (2^1) modulo 60 is 4
2 raised to the power (2^2) modulo 60 is 16
2 raised to the power (2^3) modulo 60 is 16
2 raised to the power (2^4) modulo 60 is 16
```

We see that $2^{2^k} \equiv 16 \pmod{60}$ for all $k \geq 2$. This is a consequence of the fact that $16 \times 16 = 256 \equiv 16 \pmod{60}$.

Running the Python script with $m = 360$ and $a = 2$ generates output beginning as follows:—

```
2 raised to the power (2^0) modulo 360 is 2
2 raised to the power (2^1) modulo 360 is 4
2 raised to the power (2^2) modulo 360 is 16
2 raised to the power (2^3) modulo 360 is 256
2 raised to the power (2^4) modulo 360 is 16
2 raised to the power (2^5) modulo 360 is 256
```

We conclude from this that if k is even, and if $k \geq 2$, then $2^{2^k} \equiv 16 \pmod{360}$, and if k is odd, and if $k \geq 3$, then $2^{2^k} \equiv 256 \pmod{360}$.

Running the Python program with $m = 360$ and $a = 13$ produces the following output:—

```
13 raised to the power (2^0) modulo 360 is 13
13 raised to the power (2^1) modulo 360 is 169
13 raised to the power (2^2) modulo 360 is 121
13 raised to the power (2^3) modulo 360 is 241
13 raised to the power (2^4) modulo 360 is 121
13 raised to the power (2^5) modulo 360 is 241
```

We conclude from this that if k is even, and if $k \geq 2$, then $13^{2^k} \equiv 121 \pmod{360}$, and if k is odd, and if $k \geq 3$, then $2^{2^k} \equiv 241 \pmod{360}$.

Running the Python program with $m = 227$ and $a = 13$ produces the following output:—

```
[dwilkins@mta106032 AdditionalNotes]$ ./twopowers.py 227 13
13 raised to the power (2^0) modulo 227 is 13
13 raised to the power (2^1) modulo 227 is 169
13 raised to the power (2^2) modulo 227 is 186
13 raised to the power (2^3) modulo 227 is 92
13 raised to the power (2^4) modulo 227 is 65
13 raised to the power (2^5) modulo 227 is 139
13 raised to the power (2^6) modulo 227 is 26
13 raised to the power (2^7) modulo 227 is 222
13 raised to the power (2^8) modulo 227 is 25
13 raised to the power (2^9) modulo 227 is 171
13 raised to the power (2^10) modulo 227 is 185
13 raised to the power (2^11) modulo 227 is 175
13 raised to the power (2^12) modulo 227 is 207
13 raised to the power (2^13) modulo 227 is 173
13 raised to the power (2^14) modulo 227 is 192
13 raised to the power (2^15) modulo 227 is 90
13 raised to the power (2^16) modulo 227 is 155
13 raised to the power (2^17) modulo 227 is 190
13 raised to the power (2^18) modulo 227 is 7
13 raised to the power (2^19) modulo 227 is 49
13 raised to the power (2^20) modulo 227 is 131
13 raised to the power (2^21) modulo 227 is 136
13 raised to the power (2^22) modulo 227 is 109
13 raised to the power (2^23) modulo 227 is 77
13 raised to the power (2^24) modulo 227 is 27
13 raised to the power (2^25) modulo 227 is 48
13 raised to the power (2^26) modulo 227 is 34
13 raised to the power (2^27) modulo 227 is 21
13 raised to the power (2^28) modulo 227 is 214
13 raised to the power (2^29) modulo 227 is 169
```

We see that $13^{2^{29}} \equiv 13^{2^1}$ (mod. 227). Moreover $k = 29$ is the smallest value of k for which $13^{2^k} \equiv 13^{2^j}$ (mod. 227) for some value of j satisfying $j < k$. It follows that, for values of k greater than 0, the value of 13^{2^k} (mod. 227) are determined by the congruence class of k modulo 28. Thus in order to find this value for some fairly large value of k, one should divide k by 28 in integer arithmetic, and determine the appropriate value of of a^{2^j} with $1 \leq j \leq 28$. For example, $3853 \equiv 17$ (mod. 28). Therefore $13^{2^{3853}} \equiv 13^{2^{17}} \equiv 190$ (mod. 28).