# CS 278 Lab: Sequences

One of the sequences that is often used in Computer Science is a sequence of pseudorandom numbers. For example, java.util.Random class can be used to generate random numbers. However, the numbers that are generated are not truly random. They appear to be random, but they are completely determined by a pseudorandom number generator - an algorithm that uses mathematical formulas to produce sequences of numbers.

In this assignment we will use a linear congruential random number generator. It is defined in terms of four integers: the multiplicative constant a, the additive constant b, the starting point or seed c, and the modulus M. The generator produces a sequence of integers between 0 and M-1 by starting with $x_0$ = c and iterating: $x_{n+1}$ = (a * $x_n$ + b) % M.

Here the % operator means modulus or remainder: this keeps the numbers between 0 and M-1.

For example, the following table shows the sequences that result for various choices of a, b, c, and M. Each row in the table contains the values of a, b, c, M, and the first 12 elements of the resulting sequence.

| a | b | c | M | | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | $x_{10}$ | $x_{11}$ | $x_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 0 | 10 | | 0 | 3 | 6 | 9 | 2 | 5 | 8 | 1 | 4 | 7 | 0 | 3 | 6 |
| 2 | 1 | 0 | 10 | | 0 | 1 | 3 | 7 | 5 | 1 | 3 | 7 | 5 | 1 | 3 | 7 | 5 |
| 22 | 1 | 0 | 72 | | 0 | 1 | 23 | 3 | 67 | 35 | 51 | 43 | 11 | 27 | 19 | 59 | 3 |
| 11 | 37 | 1 | 100 | | 1 | 48 | 65 | 52 | 9 | 36 | 33 | 0 | 37 | 44 | 21 | 68 | 85 |
| 8 | 20 | 10 | 100 | | 10 | 0 | 20 | 80 | 60 | 0 | 20 | 80 | 60 | 0 | 20 | 80 | 60 |

The generated sequences are not random, but (for proper choices of the parameters) they exhibit many properties of random sequences. For small values of the parameters, the non-randomness is particularly evident: whenever we generate a value that we have seen before, we enter into a cycle, where we continually generate the same subsequence over and over again. In the first two examples above, the cycles are 0-3-6-9-2-5-8-1-4-7-0 and 1-3-7-5-1, of lengths 10 and 4, respectively. Since there are only M different possibilities (for values of the elements in the sequence), we always must enter such a cycle, but we want to avoid short cycles. Notice that the cycle may start at the first element of the sequence like in the 1st sequence in the table, or it may start later in the sequence, e.g., in the 2nd sequence the cycle starts at the 2nd element $x_1$.

In practice, we often want random integers in a small range, say between 0 and 9. Typically we do so by using a large M in a linear congruential random number generator, and then use arithmetic to bring them down into a small range (e.g., use only the leading digits of the sequence elements). For example, the leading digits of the first 50 terms in the 4th sequence in the above table are:

0 4 6 5 0 3 3 0 3 4 2 6 8 7 2 5 5 2 5 6 4 8 0 9 4 7 7 4 7 8 6 0 2 1 6 9 9 6 9 0 8 2 4 3 8 1 1 8 1 2

which is a random-looking sequence of integers between 0 and 9 (although the pattern repeats itself every 50 iterations). But if we use this method for the 5th example in the table above, we get stuck in the cycle

1 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0 2 8 6 0

which is not a very random-looking sequence. This phenomenon can happen even for huge M, so we are interested in knowing that our choice of parameters does not lead to a small cycle.

We can determine the cycle length by using Floyd's algorithm for finding the cycle length or period. Floyd's algorithm consists of two phases: (i) find a point on the cycle, and (ii) compute the cycle length.

- Phase (i). To find a point on the cycle, we use the following idea: run two versions of the generator concurrently, one iterating twice as fast as the other, until both versions have the same value. At some point, both of them are on the cycle and we have a race on the cycle, with the gap between the faster one and the slower one shrinking by one on each iteration. Eventually, the faster one catches up to the slower

one, and the two generators are at the same point on the cycle. The following table contains an example of the process with a = 22, b = 1, c = 0, and M = 72. In this example, the cycle is 3-67-35-51-43-11-27-19-59-3, and its length is 9. Somewhat coincidentally, this happens to be equal to the number of steps until the fast and slow generators first meet, but this will not be true in general. The fast generator (denoted by F) happens to be at 51 when the slow one (denoted by S) first hits the cycle at 3. The fast one is six steps behind the slow one, and catches up, one step at a time.

| Sequence / Iteration | | | | Cycle | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 23 | 3 | 67 | 35 | 51 | 43 | 11 | 27 | 19 | 59 | |
| 0 | S F | | | | | | | | | | | | |
| 1 | | S | F | | | | | | | | | | |
| 2 | | | S | | F | | | | | | | | Distance |
| 3 | | | | S | | | F | | | | | | 6 |
| 4 | | | | | S | | | | F | | | | 5 |
| 5 | | | | | | S | | | | | F | | 4 |
| 6 | | | | F | | | S | | | | | | 3 |
| 7 | | | | | | F | | S | | | | | 2 |
| 8 | | | | | | | | F | S | | | | 1 |
| 9 | | | | | | | | | | S F | | | 0 |

On iteration 9, both versions of the generator (S and F) have the same value (27). 27 is a point (a value) on the cycle that was found using the above idea.

- Phase (ii). Once we know a value on the cycle, one more trip around the cycle (back to that same value) will give us the cycle length. Continuing the example above, we discovered from Phase I that 27 is in the cycle. After another 9 iterations we are back at 27, so we conclude the cycle length is 9 (as illustrated in the table below).

| Iteration | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Sequence element | 27 | 19 | 59 | 3 | 67 | 35 | 51 | 43 | 11 | 27 |

**Write a program that does the following:**

1. It prompts the user to enter values of the four integers (a, b, c, and M in this order) and prints out the first 100 elements ($x_0$, $x_1$, $x_2$, …, $x_{99}$) of the sequence produced by the linear congruential random number generator for these parameters. Use a for loop and the following update formula:

   x = (a * x + b) % M .

   Print the sequence elements in a table with 10 values per line.

2. The second part of this assignment is to implement Floyd's algorithm (described above) for finding the cycle length. First, you need to find a point on the cycle (Phase (i)). Then, you need to compute the cycle length (Phase (ii)). Output the cycle length.

**Sample runs of the program** may look like the following (user input is in **red**):

Sample run 1:

```
Please enter values of a, b, c, and M in this order: 22 1 0 72
The first 100 elements of the sequence:
0  1  23  3  67  35  51  43  11  27
19  59  3  67  35  51  43  11  27  19
59  3  67  35  51  43  11  27  19  59
3  67  35  51  43  11  27  19  59  3
67  35  51  43  11  27  19  59  3  67
35  51  43  11  27  19  59  3  67  35
51  43  11  27  19  59  3  67  35  51
43  11  27  19  59  3  67  35  51  43
11  27  19  59  3  67  35  51  43  11
27  19  59  3  67  35  51  43  11  27

Cycle length is 9
```

Sample run 2:

```
Please enter values of a, b, c, and M in this order: 123 456 789 1000000
The first 100 elements of the sequence:
789  97503  993325  179431  70469  668143  182045  391991  215349  488383
71565  802951  763429  902223  973885  788311  962709  413663  881005  364071
781189  86703  664925  786231  706869  945343  277645  150791  547749  373583
951165  993751  231829  515423  397485  891111  607109  674863  8605  58871
241589  715903  56525  953031  223269  462543  893245  869591  960149  98783
150765  544551  980229  568623  941085  753911  731509  976063  56205  913671
381989  985103  168125  679831  619669  219743  28845  548391  452549  663983
670365  455351  8629  61823  604685  376711  335909  317263  23805  928471
202389  894303  999725  966631  896069  216943  684445  187191  24949  69183
509965  726151  317029  995023  388285  759511  420309  698463  911405  103271

Cycle length is 50000
```

Sample run 3:

```
Please enter values of a, b, c, and M in this order: 124 456 789 1000000
The first 100 elements of the sequence:
789  98292  188664  394792  954664  378792  970664  362792  986664  346792
2664  330792  18664  314792  34664  298792  50664  282792  66664  266792
82664  250792  98664  234792  114664  218792  130664  202792  146664  186792
162664  170792  178664  154792  194664  138792  210664  122792  226664  106792
242664  90792  258664  74792  274664  58792  290664  42792  306664  26792
322664  10792  338664  994792  354664  978792  370664  962792  386664  946792
402664  930792  418664  914792  434664  898792  450664  882792  466664  866792
482664  850792  498664  834792  514664  818792  530664  802792  546664  786792
562664  770792  578664  754792  594664  738792  610664  722792  626664  706792
642664  690792  658664  674792  674664  658792  690664  642792  706664  626792

Cycle length is 250
```

Sample run 4:

```
Please enter values of a, b, c, and M in this order: 78 60 89 129024
The first 100 elements of the sequence:
89  7002  30120  26988  40740  81204  11796  16980  34260  91860
68820  78036  22740  96468  41172  114900  59604  4308  78036  22740
96468  41172  114900  59604  4308  78036  22740  96468  41172  114900
59604  4308  78036  22740  96468  41172  114900  59604  4308  78036
22740  96468  41172  114900  59604  4308  78036  22740  96468  41172
114900  59604  4308  78036  22740  96468  41172  114900  59604  4308
78036  22740  96468  41172  114900  59604  4308  78036  22740  96468
41172  114900  59604  4308  78036  22740  96468  41172  114900  59604
```

```
4308  78036  22740  96468  41172  114900  59604  4308  78036  22740
96468  41172  114900  59604  4308  78036  22740  96468  41172  114900

Cycle length is 7
```

**What to submit:**

- Submit the source code of your program using Canvas.
- If you write your program in a programming language other than Java, then submit instructions on how to compile and run your program on CS machines.

**Note:** This assignment is adopted from an assignment created by Robert Sedgewick and Kevin Wayne.