Random Number Generation

Raj Jain
Washington University
Saint Louis, MO 63130
Jain@cse.wustl.edu

Audio/Video recordings of this lecture are available at:

http://www.cse.wustl.edu/~jain/cse567-17/

- Desired properties of a good generator
- Linear-congruential generators
- Tausworthe generators
- Survey of random number generators
- Seed selection
- Myths about random number generation

Random-Number Generation

- Random Number = Uniform (0, 1)
- Random Variate = Other distributions
  = Function(Random number)

A Sample Generator

$$x_n = f(x_{n-1}, x_{n-2}, \ldots)$$

- For example,

$$x_n = 5x_{n-1} + 1 \mod 16$$

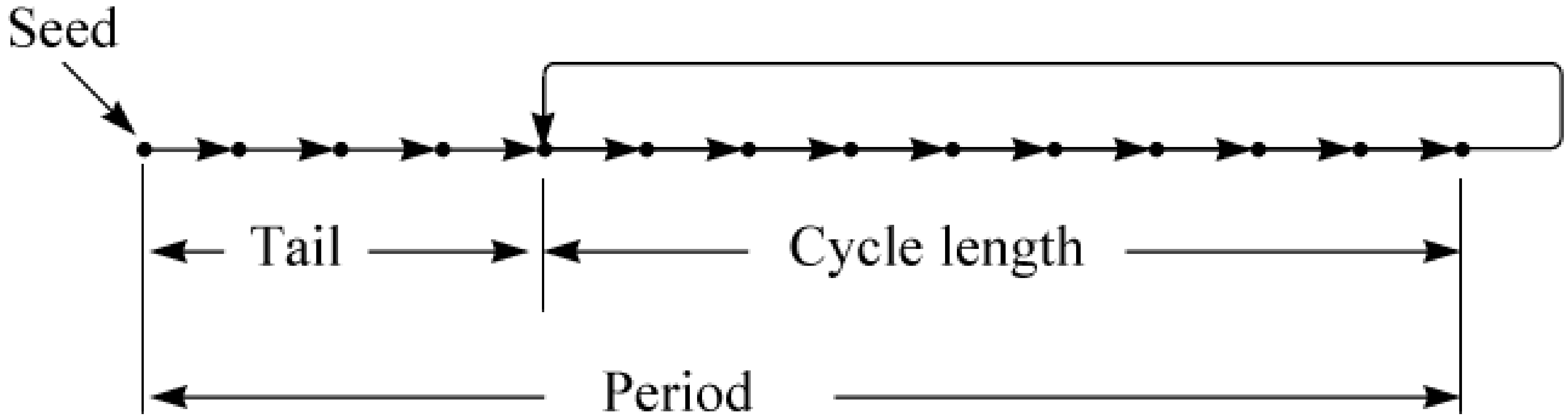
- Starting with x_0=5:

$$x_1 = 5(5) + 1 \mod 16 = 26 \mod 16 = 10$$

- The first 32 numbers obtained by the above procedure 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.
- By dividing x's by 16:
0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125, 0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125.

Terminology

- **Seed** = x_0
- **Pseudo-Random**: Deterministic yet would pass randomness tests
- Fully Random: Not repeatable
- **Cycle length**, **Tail**, **Period**



Properties of a Good Generator

- It should be efficiently computable.
- The period should be large.
- The successive values should be independent and uniformly distributed

Types of Generators

- Linear congruential generators
- Tausworthe generators
- Extended Fibonacci generators
- Combined generators

Linear-Congruential Generators

- Discovered by D. H. Lehmer in 1951
- The residues of successive powers of a number have good randomness properties.

$$x_n = a^n \bmod m$$

Equivalently,

$$x_n = a x_{n-1} \bmod m$$

a = multiplier

m = modulus

LCG (Cont)

- Lehmer's choices: a = 23 and m = 10^8+1
- Good for ENIAC, an 8-digit decimal machine.
- Generalization:

$$x_n = ax_{n-1} + b \bmod m$$

- Can be analyzed easily using the theory of congruences
  $\Rightarrow$ Mixed Linear-Congruential Generators
  or Linear-Congruential Generators (LCG)
- Mixed = both multiplication by a and addition of b

Selection of LCG Parameters

- *a*, *b*, and *m* affect the period and autocorrelation
- The modulus *m* should be large.
- The period can never be more than *m*.
- For mod *m* computation to be efficient, *m* should be a power of 2 $\Rightarrow$ Mod *m* can be obtained by truncation.

LCG Parameters (Cont)

- If *b* is nonzero, the maximum possible period *m* is obtained if and only if:
  - Integers *m* and *b* are relatively prime, that is, have no common factors other than 1.
  - Every prime number that is a factor of *m* is also a factor of *a*-1.
  - If integer *m* is a multiple of 4, *a*-1 should be a multiple of 4.
  - Notice that all of these conditions are met if $m=2^k$, a = 4c + 1, and *b* is odd. Here, *c*, *b*, and *k* are positive integers.

Period vs. Autocorrelation

- A generator that has the maximum possible period is called a full-period generator.

$$x_n = (2^{34} + 1)x_{n-1} + 1 \mod 2^{35}$$

$$x_n = (2^{18} + 1)x_{n-1} + 1 \mod 2^{35}$$

- Lower autocorrelations between successive numbers are preferable.
- Both generators have the same full period, but the first one has a correlation of 0.25 between x_{n-1} and x_n, whereas the second one has a negligible correlation of less than 2^{-18}

Multiplicative LCG

- Multiplicative LCG: b=0

$$x_n = a x_{n-1} \bmod m$$

- Two types:

$$m = 2^k$$

$$m \neq 2^k$$

Multiplicative LCG with m=2^k

- $m = 2^k \Rightarrow$ trivial division
  $\Rightarrow$ Maximum possible period 2^{k-2}
- Period achieved if multiplier a is of the form $8i \pm 3$, and the initial seed is an odd integer
- One-fourth the maximum possible may not be too small
- Low order bits of random numbers obtained using LCG's with $m=2^k$ have a cyclic pattern

Example 26.1a

$$x_n = 5x_{n-1} \bmod 2^5$$

- Using a seed of x_0=1:

  5, 25, 29, 17, 21, 9, 13, 1, 5,…

  Period = 8 = 32/4
- With x_0 = 2, the sequence is: 10, 18, 26, 2, 10,…

  Here, the period is only 4.

Example 26.1b

- Multiplier not of the form 8i ± 3:

$$x_n = 7x_{n-1} \bmod 2^5$$

- Using a seed of $x_0 = 1$, we get the sequence: 7, 17, 23, 1, 7,…
- The period is only 4

Multiplicative LCG with $m \neq 2^k$

- Modulus m = prime number

  With a proper multiplier a, period = m-1

  Maximum possible period = m
- If and only if the multiplier a is a *primitive root* of the modulus m
- a is a primitive root of m if and only if $a^n \bmod m \neq 1$ for $n = 1, 2, \ldots, m-2$.

Example 26.2

$$x_n = 3x_{n-1} \bmod 31$$

- Starting with a seed of x_0=1:

  1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, …

  The period is 30

  $\Rightarrow$ 3 is a primitive root of 31
- With a multiplier of a = 5: 1, 5, 25, 1,…

  The period is only 3 $\Rightarrow$ 5 is not a primitive root of 31

  $$5^3 \bmod 31 = 125 \bmod 31 = 1$$
- Primitive roots of 31= 3, 11, 12, 13, 17, 21, 22, and 24.

Seed Selection

- **Multi-stream simulations**: Need more than one random stream
  - Single queue $\Rightarrow$ Two streams
    = Random arrival and random service times

1. Do not use zero. Fine for mixed LCGs.
   But multiplicative LCG or a Tausworthe LCG will stick at zero.
2. Avoid even values. For multiplicative LCG with modulus $m=2^k$, the seed should be odd. Better to avoid generators that have too many conditions on seed values or whose performance (period and randomness) depends upon the seed value.
3. Do not subdivide one stream.

Seed Selection (Cont)

4. Do not generate successive seeds: u_1 to generate inter-arrival times, u_2 to generate service time ⇒ Strong correlation
5. Use non-overlapping streams.
   Overlap ⇒ Correlation, e.g., Same seed ⇒ same stream
6. Reuse seeds in successive replications.
7. Do not use random seeds: Such as the time of day.
   Can't reproduce. Can't guaranteed non-overlap.
8. Select $\{u_0, u_{100,000}, u_{200,000}, \ldots\}$

$$x_n = a^n x_0 + \frac{c(a^n - 1)}{a - 1} \bmod m$$

Table of Seeds

$$x_n = 7^5 x_{n-1} \bmod (2^{31} - 1)$$

$x_{100000i}$	$x_{100000(i+1)}$	$x_{100000(i+2)}$	$x_{100000(i+3)}$
1	46,831,694	1,841,581,359	1,193,163,244
727,633,698	933,588,178	804,159,733	1,671,059,989
1,061,288,424	1,961,692,154	1,227,283,347	1,171,034,773
276,090,261	1,066,728,069	209,208,115	554,590,007
721,958,466	1,371,272,478	675,466,456	1,095,462,486
1,808,217,256	2,095,021,727	1,769,349,045	904,914,315
373,135,028	717,419,739	881,155,353	1,489,529,863
1,521,138,112	298,370,230	1,140,279,430	1,335,826,707
706,178,559	110,356,601	884,434,366	962,338,209
1,341,315,363	709,314,158	591,449,447	431,918,286
851,767,375	606,179,079	1,500,869,201	1,434,868,289
263,032,577	753,643,799	202,794,285	715,851,524

Myths About Random-Number Generation

1. *A complex set of operations leads to random results*. It is better to use simple operations that can be analytically evaluated for randomness.
2. *A single test, such as the chi-square test, is sufficient to test the goodness of a random-number generator.* The sequence 0,1,2,...,m-1 will pass the chi-square test with a perfect score, but will fail the run test $\Rightarrow$ Use as many tests as possible.
3. *Random numbers are unpredictable*. Easy to compute the parameters, a, c, and m from a few numbers $\Rightarrow$ LCGs are unsuitable for cryptographic applications

Myths (Cont)

4. *Some seeds are better than others*. May be true for some.

$$x_n = (9806 x_{n-1} + 1) \bmod (2^{17} - 1)$$

- Works correctly for all seeds except $x_0 = 37911$
- Stuck at $x_n = 37911$ forever
- Such generators should be avoided.
- Any *nonzero* seed in the valid range should produce an equally good sequence.
- For some, the seed should be odd.
- Generators whose period or randomness depends upon the seed should not be used, since an unsuspecting user may not remember to follow all the guidelines.

Myths (Cont)

5. *Accurate implementation is not important*.
   - RNGs must be implemented without any overflow or truncation For example,
     $$x_n = 1103515245 x_{n-1} + 12345 \bmod 2^{31}$$
   - In FORTRAN:
     $$x_n = (1103515245 x_{n-1} + 12345).AND.X'7FFFFFFF'$$
   - The AND operation is used to clear the sign bit
   - Straightforward multiplication above will produce overflow.
6. *Bits of successive words generated by a random-number generator are equally randomly distributed*.
   - If an algorithm produces *l*-bit wide random numbers, the randomness is guaranteed only when all *l* bits are used to form successive random numbers.

Example 26.7

$$x_n = (25173 x_{n-1}) \bmod 2^{16}$$

Notice that:

a) Bit 1 (the least significant bit) is always 1.

b) Bit 2 is always 0.

c) Bit 3 alternates between 1 and 0, thus, it has a cycle of length 2.

d) Bit 4 follows a cycle (0110) of length 4.

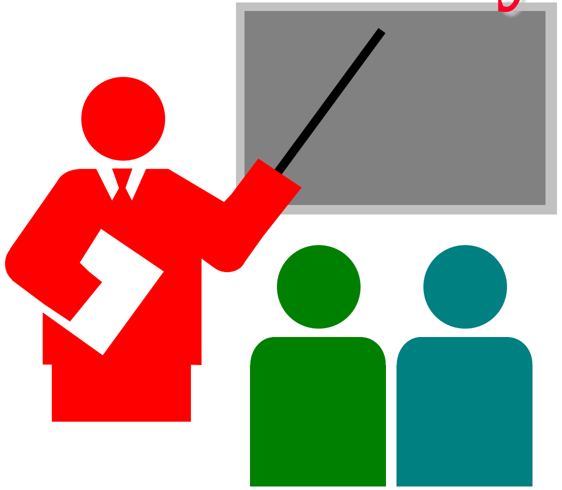
e) Bit 5 follows a cycle (11010010) of length 8.

n	x_n	
	Decimal	Binary
1	25,173	01100010 01010101
2	12,345	00110000 00111001
3	54,509	11010100 11101101
4	27,825	01101100 10110001
5	55,493	11011000 11000101
6	25,449	01100011 01101001
7	13,277	00110011 11011101
8	53,857	11010010 01100001
9	64,565	11111100 00110101
10	1945	00000111 10011001
11	6093	00010111 11001101
12	24,849	01100001 00010001
13	48,293	10111100 10100101

Example 26.7 (Cont)

- The least significant bit is either always 0 or always 1.
- The *l*th bit has a period at most 2^l. (l=1 is the least significant bit)
- For all mixed LCGs with $m=2^k$:
  - The *l*th bit has a period at most 2^l.
  - In general, the high-order bits are more randomly distributed than the low-order bits.
    $\Rightarrow$ Better to take the high-order *l* bits than the low-order *l* bits.

Summary



- Pseudo-random numbers are used in simulation for repeatability, non-overlapping sequences, long cycle
- It is important to implement PRNGs in integer arithmetic without overflow => Schrage's method
- For multi-stream simulations, it is important to select seeds that result in non-overlapping sequences
- Two or more generators can be combined for longer cycles
- Bits of random numbers may not be random

Scan This to Download These Slides

Raj Jain

http://rajjain.com

Related Modules

CSE567M: Computer Systems Analysis (Spring 2013),
https://www.youtube.com/playlist?list=PLjGG94etKypJEKjNAa1n_1X0bWWNyZcof

CSE473S: Introduction to Computer Networks (Fall 2011),
https://www.youtube.com/playlist?list=PLjGG94etKypJWOSPMh8Azcgy5e_10TiDw

Wireless and Mobile Networking (Spring 2016),
https://www.youtube.com/playlist?list=PLjGG94etKypKeb0nzyN9tSs_HCd5c4wXF

CSE571S: Network Security (Fall 2011),
https://www.youtube.com/playlist?list=PLjGG94etKypKvzfVtutHcPFJXumyyg93u

Video Podcasts of Prof. Raj Jain's Lectures,
https://www.youtube.com/channel/UCN4-5wzNP9-ruOzQMs-8NUw