Lagged Fibonacci generator

From Wikipedia, the free encyclopedia

A **Lagged Fibonacci generator** (**LFG** or sometimes **LFib**) is an example of a pseudorandom number generator. This class of random number generator is aimed at being an improvement on the 'standard' linear congruential generator. These are based on a generalisation of the Fibonacci sequence.

The Fibonacci sequence may be described by the recurrence relation:

$$S_n = S_{n-1} + S_{n-2}$$

Hence, the new term is the sum of the last two terms in the sequence. This can be generalised to the sequence:

$$S_n \equiv S_{n-j} \star S_{n-k} \pmod{m}, 0 < j < k$$

In which case, the new term is some combination of any two previous terms. m is usually a power of 2 ($m = 2^M$), often 2^{32} or 2^{64}. The $\star$ operator denotes a general binary operation. This may be either addition, subtraction, multiplication, or the bitwise arithmetic exclusive-or operator (XOR). The theory of this type of generator is rather complex, and it may not be sufficient simply to choose random values for j and k. These generators also tend to be very sensitive to initialisation.

Generators of this type employ k words of state (they 'remember' the last k values).

If the operation used is addition, then the generator is described as an *Additive Lagged Fibonacci Generator* or ALFG, if multiplication is used, it is a *Multiplicative Lagged Fibonacci Generator* or MLFG, and if the XOR operation is used, it is called a *Two-tap generalised feedback shift register* or GFSR. The Mersenne twister algorithm is a variation on a GFSR. The GFSR is also related to the linear feedback shift register, or LFSR.

Properties of lagged Fibonacci generators [edit]

Lagged Fibonacci generators have a maximum period of $(2^k - 1)*2^{M-1}$ if addition or subtraction is used, and $(2^k-1)*k$ if exclusive-or operations are used to combine the previous values. If, on the other hand, multiplication is used, the maximum period is $(2^k - 1)*2^{M-3}$, or 1/4 of period of the additive case.

For the generator to achieve this maximum period, the polynomial:

$$y = x^k + x^j + 1$$

must be primitive over the integers mod 2. Values of j and k satisfying this constraint have been published in the literature. Popular pairs are:

{j = 7, k = 10}, {j = 5, k = 17}, {j = 24, k = 55}, {j = 65, k = 71}, {j = 128, k = 159} [1], {j = 6, k = 31}, {j = 31, k = 63}, {j = 97, k = 127}, {j = 353, k = 521}, {j = 168, k = 521}, {j = 334, k = 607}, {j = 273, k = 607}, {j = 418, k = 1279} [2]

Another list of possible values for *j* and *k* is on page 29 of volume 2 of *The Art of Computer Programming*:

(24,55), (38,89), (37,100), (30,127), (83,258), (107,378), (273,607), (1029,2281), (576,3217), (4187,9689), (7083,19937), (9739,23209)

Note that the smaller number have short periods (only a few "random" numbers are generated before the first "random" number is repeated and the sequence restarts).

If addition is used, it is required that at least one of the first k values chosen to initialise the generator be odd; if multiplication is used, instead, it is required that all the first k values be odd.[1]

It has been suggested that good ratios between j and k are approximately the golden ratio.[2]

Wikipedia The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Հայերեն
Italiano
日本語
Polski
Русский
ไทย
Edit links

Problems with LFGs [edit]

In a paper on four-tap shift registers, Robert M. Ziff, referring to LFGs that use the XOR operator, states that "It is now widely known that such generators, in particular with the two-tap rules such as R(103, 250), have serious deficiencies. Marsaglia observed very poor behavior with R(24,55) and smaller generators, and advised against using generators of this type altogether. ... The basic problem of two-tap generators R(a, b) is that they have a built-in three-point correlation between x_n, x_{n-a}, and x_{n-b}, simply given by the generator itself ... While these correlations are spread over the size $p = max(a, b, c, \ldots)$ of the generator itself, they can evidently still lead to significant errors.".[3] This only refers to the standard LFG where each new number in the sequence depends on two previous numbers. A three-tap LFG has been shown to eliminate some statistical problems such as failing the Birthday Spacings and Generalized Triple tests.[4]

The initialization of LFGs is a very complex problem. The output of LFGs is very sensitive to initial conditions, and statistical defects may appear initially but also periodically in the output sequence unless extreme care is taken .[*citation needed*] Another potential problem with LFGs is that the mathematical theory behind them is incomplete, making it necessary to rely on statistical tests rather than theoretical performance.

Usage [edit]

- Freeciv uses a lagged Fibonacci generator with {j = 24, k = 55} for its random number generator.
- The Boost library includes an implementation of a lagged Fibonacci generator.
- Subtract with carry, a lagged Fibonacci generator engine, is included in the C++11 library.
- The Oracle Database implements this generator in its DBMS_RANDOM package (available in Oracle 8 and newer versions).
- The "Pocket Dungeon" on www.BoardGameGeek.com uses it as an alternative 'Stealth' dice roll generator.

See also [edit]

- Linear congruential generator
- Mersenne twister
- FISH (cipher)
- Pike
- VIC cipher

References [edit]

1. ^ http://www.cs.fsu.edu/~asriniva/papers/mlfg.ps
2. ^ "Uniform random number generators for supercomputers", Richard Brent, Proc. of Fifth Australian Supercomputer Conference, Melbourne, Dec. 1992, pp. 704-706
3. ^ "Four-tap shift-register-sequence random-number generators", Robert M. Ziff, Computers in Physics, 12(4), Jul/Aug 1998, pp. 385–392
4. ^ R. P. Brent, "Uniform Random Number Generators for Supercomputers," in "Proceedings of the Fifth Australian Supercomputer Conference", pp. 95-104, 1992.

Categories: Pseudorandom number generators | Fibonacci numbers