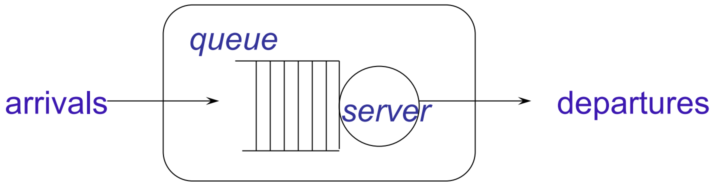
Single server queue



Consider n=10 job and given arrival and service times:

Arrival times:

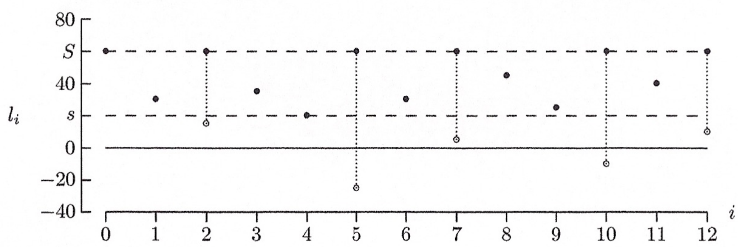
15 47 71 111 123 152 166 226 310 320

Service times:

43 36 34 30 38 40 31 29 36 30

A simple inventory system

i	:	1	2	3	4	5	6	7	8	9	10	11	12
d_i	:	30	15	25	15	45	30	25	15	20	35	20	30



- ssq1 and sis1 require input data from an outside source
- The usefulness of these programs is limited by amount of available data:
  - What if more data needed?
  - What if the model changed?
  - What if the input data set is small or unavailable?

↓

Random number generator

- It produces real values between 0.0 and 1.0
- The output can be converted to random variate via mathematical transformations

Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Random Number Generators

Università degli studi di Roma Tor Vergata
Department of Civil Engineering and Computer Science Engineering

Historically there are three types of generators
- table look-up generators (1950)
- hardware generators
- algorithmic (software) generators

Algorithmic generators are widely accepted because they meet all of the following criteria:
- *randomness* - output passes all reasonable statistical tests of randomness
- *controllability* - able to reproduce output, if desired
- *portability* - able to produce the same output on a wide variety of computer systems
- *efficiency* - fast, minimal computer resource requirements
- *documentation* - theoretically analyzed and extensively tested

Algorithmic Generators

- An *ideal* random number generator produces output such that *each* value in the interval 0.0 < u < 1.0 is *equally likely* to occur
- A *good* random number generator produces output that is (almost) statistically indistinguishable from an ideal generator

Conceptual Model

- Choose a *large* positive integer $m>0$. This defines the set
  $\chi_m = \{1,2,\ldots m-1\}$
- Fill a (conceptual) urn with the elements of χ_m
- Each time a random number u is needed, draw an integer x "at random" from the urn and let $u = x/m$

- Each draw *simulates* a sample of an independent identically distributed sequence of *Uniform*(0, 1)

- The possible values are $1/m$, $2/m$, … $(m-1)/m$

- It is important that m be large so that the possible values are densely distributed between 0.0 and 1.0

Conceptual Model

- 0.0 and 1.0 are impossible
  This is important for some random variates

- the same probability for each draw→ replacement of the drawn element

- for practical reasons, we will draw without replacement
  If m is large and the number of draws is small relative to m, then the distinction is largely irrelevant

Lehmer Generator

- is defined in terms of two fixed parameters:
  - *modulus m*, a fixed large prime integer
  - *multiplier a*, a fixed integer in χ_m
- the possible values are 1/*m*, 2/*m*, … (*m*-1)/*m*

The integer sequence x_0, x_1, … is defined by the iterative equation

$$x_{i+1} = g(x_i)$$

with

$$g(x) = ax \bmod m$$

$x_0 \in \chi_m$ is called *initial seed*

- Because of the mod operator, $0 \leq g(x) < m$
- 0 must not occur
  - since *m* is prime, $g(x) \neq 0$ if $x \in \chi_m$
  - if $x_0 \in \chi_m$, then $x_i \in \chi_m$ for all $i \geq 0$
- IF the multiplier and prime modulus are chosen properly, a Lehmer generator is statistically indistinguishable from drawing from χ_m with replacement
- NOTE, there is nothing random about a Lehmer generator

→ pseudo-random generator

Parameter Considerations

- the choice of *m* is dictated, in part, by system considerations
  - on a system with 32-bit 2's complement integer arithmetic, 2^{31}-1 is a natural choice (it is prime!)
  - with 16-bit or 64-bit integer representation, the choice is not obvious (the maxes are not prime)
  - in general, we want to choose *m* to be the largest representable prime integer
- Given *m*, the choice of *a* must be made with great care

- For a chosen (a, *m*) pair, does the function g(·) generate a **full-period** sequence?
- If a full period sequence is generated, how random does the sequence appear to be?
- Can *ax* mod *m* be evaluated efficiently and correctly?
  - Integer overflow can occur when computing *ax*

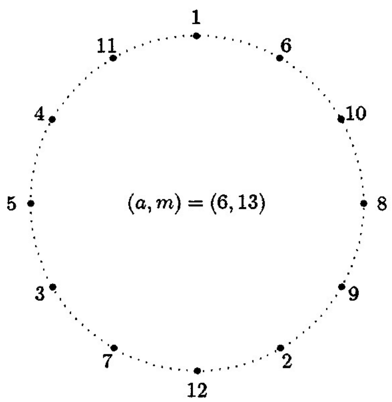
Full Period Multipliers

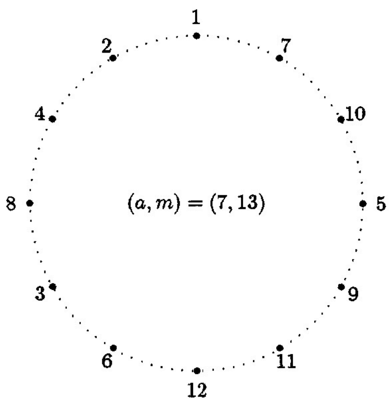
- If we pick any initial seed $x_0 \in \chi_m$ and generate the sequence x_0, x_1, x_2, … then x_0 will occur again
- Further x_0 will reappear at index p that is either $m - 1$ or a divisor of $m - 1$

We are interested in choosing full-period (FP) multipliers where p = m-1

13

Full-period multipliers generate a virtual circular list with m-1 distinct elements.

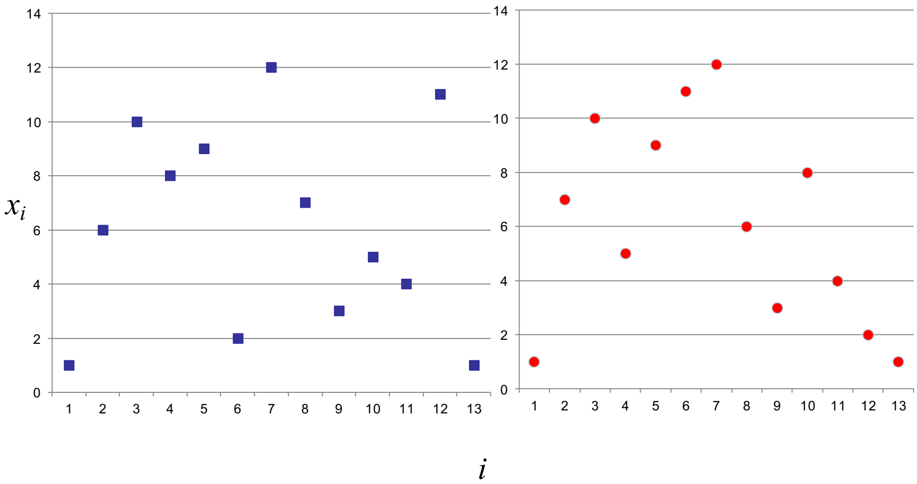




14

m=13

a=6, x_0=1 a=7, x_0=1

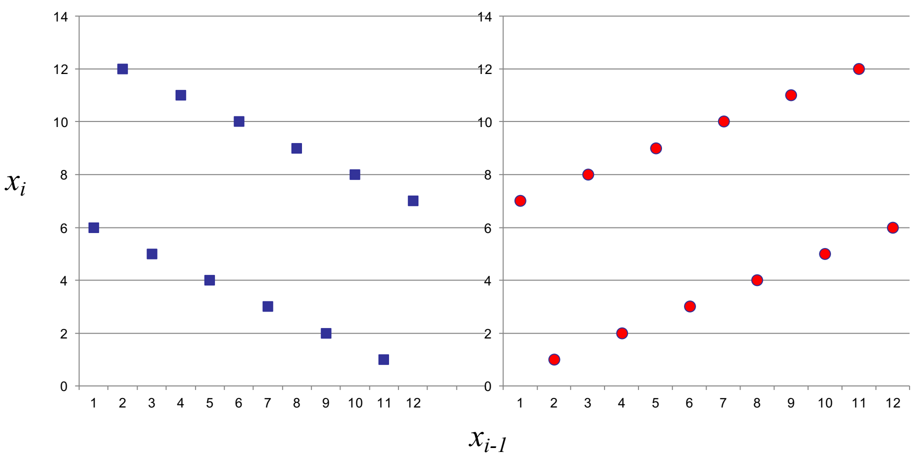


Prof. Vittoria de Nitto Personè 15

15

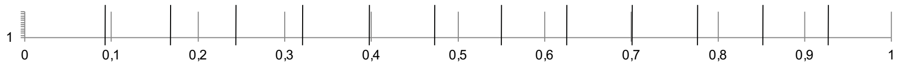
m=13

a=6, x_0=1 a=7, x_0=1



Prof. Vittoria de Nitto Personè 16

16



0,076923077	0,461538462	0,769230769	0,615384615	0,692307692	0,153846154
0,923076923	0,538461538	0,230769231	0,384615385	0,307692308	0,846153846

0,076923077	0,538461538	0,769230769	0,384615385	0,692307692	0,846153846
0,923076923	0,461538462	0,230769231	0,615384615	0,307692308	0,153846154
12	**6**	**3**	**8**	**4**	**2**