RANDOM NUMBERS

Stefan Grünberger, Markus Weninger
Martin Schobesberger, Dari Trendafilov
Institute of Pervasive Computing
Johannes Kepler University Linz
teaching@pervasive.jku.at

Algorithms and Data Structures 2

Exercise – 2021W

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

APPLICATIONS

- Simulations
- Cryptography
- Decision making
- Games
- Spot checks

RANDOM NUMBERS :: GENERATORS

Linear congruential method

- the next random number is calculated from the previous
- algorithmic -> pseudo-random numbers
- deterministic -> random sequence can be reconstructed

Calculation

$$x_{n+1} := (a * x_n + c) \bmod m$$

m *... modulus*

- *as large as possible*
- *responsible for max. period length and range of random numbers*

a *... multiplier*

- *0.01*m < a < 0.99*m*
- *without special bit pattern*

c *... increment*

- *mostly 1*

RANDOM NUMBERS :: IMPLEMENTATION EXAMPLE

`intRand()` generates random numbers between

- 0 and 2^{31}-1
- or -2147483647 to +2147483647

Pseudo code:

```
int   a = 3421
int   m = 2^31-1              // = 2.147.483.647
int   c = 1
static int x = 12345            // start Seed

static int intRand() {
  x = (a * x + c) mod m
  return x
}
```

The first 10 random numbers will always be:

```
42232246
-1552374497
-2093576380
1880653749
-144534078
-529841797
-112588624
1381373937
1216212878
-1159054185
 ...
```

RANDOM NUMBERS :: EFFICIENCY

Best practice $m = 2^n$

- If **m** is chosen to be $\mathbf{2^n}$ then: $x \bmod m = x \,\&\, (m - 1)$

- **Very** easy and fast operation
- Modulo operation is just „masking out“ the least significant n bits
- Masking operation is a **logical AND** operation

Example: n = 8 (m = 256), x = 11309

x mod 256 => x & 255

```
  00101100 00101101_bin   ... 11309_dec (=2^0+2^2+2^3+2^5+2^10+2^11+2^13)
& 00000000 11111111_bin   ... 255_dec   ...
  00000000 00101101_bin   ... 45_dec    (=2^0+2^2+2^3+2^5)
```

RANDOM NUMBERS :: EFFICIENCY

Best practice length $n = powers\ of\ 2\ (\ldots, 16, 32, 64, \ldots)$

- Modulus can be formed automatically by an **overflow**
- Does not work with all programming languages, but for example with Java (by casting a long to an int)

Example with $m = 2^{16}$ (=65536):

1. 45 mod m = 45 mod 65536 = 45 ✓

2. 65546 mod m = 65546 mod 65536 = 10

   $65546_{dec} = 01000A_{hex}$

   $= \underbrace{00000001}_{\text{dropped!}}\ \underbrace{00000000\ 00001010}_{\text{rest = }10_{dec}}{}_{bin}$

   --> first byte is dropped due to the overflow -> $000A_{hex} = 10_{dec}$ ✓

RANDOM NUMBERS :: PERIODICITY PROBLEM

Period of a random number generator is:

- length of random number sequence before previous numbers begin to repeat in a previous order

Calculation: $x_{n+1} = (a * x_n + c) \bmod m, \quad n = 0, 1, 2, 3, \ldots$

Examples:

$m = 8; a = 5; c = 3; x_0 = 0$

```
n: 1 2 3 4 5 6 7 8 9 10 11 12
x: 3 2 5 4 7 6 1 0 3 2   5 ... ⇒ period = 8 (≤ m)
```

$m = 8; a = 3; c = 5; x_0 = 0$

```
n: 1 2 3 4 5 6 7 8 9 10 11 12
x: 5 4 1 0 5 4 1 0 5 4   1 ... ⇒ period = 4
```

$m = 8; a = 3; c = 2; x_0 = 0$

```
n: 1 2 3 4 5 6 7 8 9 10 11 12
x: 2 0 2 0 2 0 2 0 2 0   2 ... ⇒ period = 2
```

RANDOM NUMBERS :: PERIOD (EXTENSION)

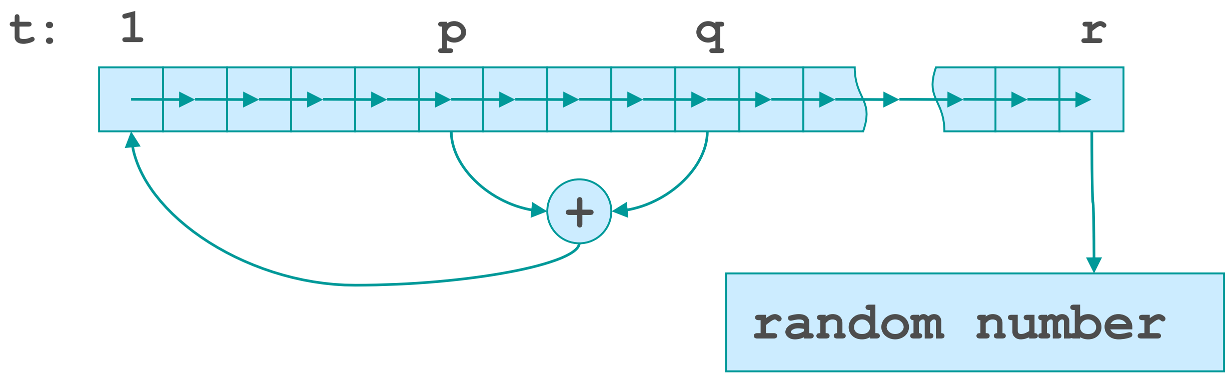
Often it is necessary to increase the *"natural"* period of a random number generator

- during a *"calculation"* the period length of a random number generator **must never** be reached
- in simulations, however, the upper limit is often reached quickly

Methods for extending the period:

- Shift register method (*Tausworthe*)
- Table method (*MacLaren* and *Marsaglia*)

RANDOM NUMBERS :: SHIFT REGISTER METHOD



according to Knuth's research good values are:

```
r = 55
p = 31
q = 55
```

- Fill the array *t* (of length *r*) with random numbers (e.g., using **`intRand()`**)
- Link the elements ***p*** and ***q*** to generate a new random number:
  - addition (**`mod m`** or overflow) --> **`random number = (t[p]+t[q]) % m`**
  - bitwise exclusive or (XOR) --> **`random number = t[p]^t[q]`**
- Shift the array by 1 position to the right (or left depending on the register structure)
- New random number is stored in **`t[0]`**

RANDOMIZED ALGORITHMS

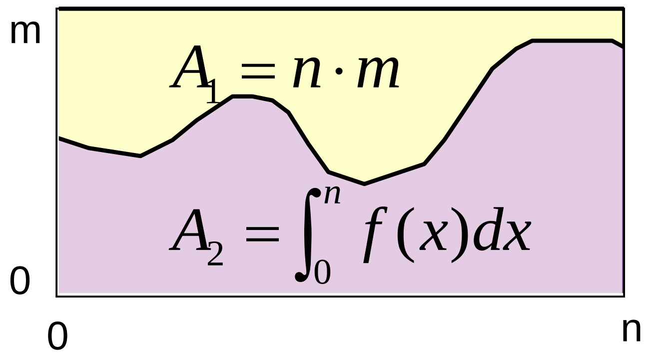
Randomized algorithms use random numbers to make **random choices** during execution → Output of the algorithm depends on a random experiment.

Monte Carlo Integration

- numerical methods for solving problems using distributions of random numbers
- shoot randomly into rectangle *((0,0)*, *(n,m))* with random number pairs *(x,y)*
- count the random number pairs that landed below the function *f(x)* (n_{in}) and the ones that landed above *f(x)* (n_{out}) $\frac{A_1}{A_2} \approx \frac{n_{in}+n_{out}}{n_{in}} \quad \Rightarrow \quad A_2 \approx A_1 \frac{n_{in}}{n_{in}+n_{out}}$
- decide if point is below or above f(x)

$n_{in} = n_{in}+1$ if $y_{rand} <= f(x_{rand})$

$n_{out} = n_{out}+1$ if $y_{rand} > f(x_{rand})$



ASSIGNMENT 01

JOHANNES KEPLER
UNIVERSITY LINZ