# Algorithms and Data Structures II

Lecture 12:

# Random Number Generators

https://elms.u-aizu.ac.jp

# Random Number

- In many applications, we need a sequence of random numbers. In this lecture we will discuss the algorithms for generating random number sequences.
- Since the numbers generated depend on the algorithm, we can not get the true random numbers. What the algorithms generate are pseudorandom numbers which are the numbers that appear to be random.

# Uniform random number

- Uniform random numbers:
  Each number is equally likely to occur.
- Now we will study how to generate uniform pseudo random numbers.

# Linear congruential method

- Linear congruential method, introduced by D. Lehmer in 1951, is the best-known method for the above purpose. In this method, numbers $x_1, x_2, \ldots$ are generated by

$$x_{i+1} = Ax_i \pmod{M}$$

where $0 < A < M$. A value $x_0$, called seed, is needed to start the sequence. Notice that $x_0$ should not be 0 otherwise we will get a sequence of 0 which is far from random.

# Linear congruential method(contd)

- For correctly chosen $A$ and $M$, any $x_i$ with $1 \leq x_0 < M$ is equally valid. If $M$ is prime then $1 \leq x_i \leq M - 1$.
  For example, if $M = 11,\ A = 6$, and $x_0 = 1$ then the sequence of numbers is

$$6, 3, 7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, \ldots$$

  If we choose $A = 5$ then the sequence is

$$5, 3, 4, 9, 1, 5, 3, 4, 9, 1, 5, \ldots$$

# Linear congruential method(contd 2)

- The first sequence has a period of $M - 1 = 10$ which is the largest, while the second one has a period of 5.
- If $M$ is prime then there are always choices of $A$ that gives a full-period of $M - 1$. If $M$ is large, e.g., a 31-bit prime, a full-period generator should satisfy most applications.

# Linear congruential method(contd 3)

- Lehmer suggested

$$M = 2^{31} - 1 = 2147483647.$$

For this prime, $A = 48271$ is one of the values that gives a full-period generator:

1. $x_0 = 1$
2. $x_{i+1} = (x_i * 48271)(\text{mod } (2^{31} - 1))$

# Linear congruential method(contd 4)

A straightforward implementation for the above generator may like this:

```
static int x = 1;
static int A = 48271;
static int M = 2147483647;
int next_rnd(){
x=(A*x) % M;
return x;}
```

However, this implementation does not work well on most computers.

# Linear congruential method(contd 5)

- The problem is that $(A * x)$ could overflow. When a overflow occurs, $(A * x)$ becomes the value of $(A * x)\%2^b$, where $b$ is chosen to match the number of bits in the computer's integer.

# Linear congruential method(contd 6)

- That is, $(A * x)\%M$ becomes $((A * x)\%2^b)\%M)$ if $(A * x)$ overflows. This changes the results of the generated sequence and thus, affects the pseudorandomness of the sequence.

# Linear congruential method(contd 7)

- Notice that if we take $M = 2^b$ then a overflow is an operation of $(A * x)\%M$.
  In fact, many libraries have the generators based on the function

$$x_{i+1} = (A * x_i + c)(\text{mod } 2^b).$$

# Linear congruential method(contd 8)

- Now we give an algorithm for generating random numbers based on the above formula in a computer with 32-bit integer.
- For a 32-bit integer, 1-bit is used for the sign and the other 31-bits are used for the absolute value of the integer, and thus the largest integer that can be expressed is

$$2^{31} - 1 = 2147483647.$$

# Linear congruential method(contd 9)

- We choose $b = 31$ and $M = 2^b = 2147483648$. Since $M$ can not be expressed by the 32-bit integer, we express $M$ by $(M - 1) + 1$ in the subroutine.

# Linear congruential method(contd 10)

```
#include <stdio.h>
static int x=53402397;

int next_rnd1(){
  x=x*65539+125654;
  if (x<0)
    {x+=2147483647;x+=1;}
  return x;
}
```

# Linear congruential method(contd 10.5)

```
#include <stdio.h>
static long x=53402397;
int next_rnd1(){
  x=x*65539+125654;
  if ((int)x<0)
    {x+=2147483647;x+=1;}
  return x;
}
```

# Linear congruential method(contd 10.56)

Above 2 affect to the sign bit... then

```
#include <stdio.h>
static long x=53402397;

unsigned int next_rnd1(){
  x=x*65539+125654;
  if ((int)x<0)
    {x+=2147483647;x+=1;}
  return x;
}
```

# Linear congruential method(contd 11)

- In the above algorithm, if $c$ is odd then the values of $x_i$ alternate between even and odd, and if $c$ is even then the parity of $x_i$ are all the same. Those may not be a nice property for a sequence of random numbers.

# Linear congruential method(contd 12)

- If we want to use

$$x_{i+1} = A * x_i (\text{mod } M)$$

to generate better random sequences then the overflow problem must be solved.

- Below is a solution for the overflow problem.

# Linear congruential method(contd 13)

- Given $M$ and $A$, let $Q = \lfloor M/A \rfloor$ and $R = M\%A$. Then it can be shown that $x_{i+1} = A * x_i (\text{mod } M)$ is equivalent to

$$x_{i+1} = A(x_i(\text{mod } Q)) - R\lfloor x_i/Q \rfloor + M(\lfloor x_i/Q \rfloor - \lfloor Ax_i/M \rfloor).$$

- For $M = 2^{31} - 1$ and $A = 48271$, it is easy to check that there will be no overflow in calculating the above formula on a computer with 32-bit integer.

# Linear congruential method(contd 14)

- Also $(\lfloor x_i/Q \rfloor - \lfloor Ax_i/M \rfloor)$ is either 0 or 1 and $(\lfloor x_i/Q \rfloor - \lfloor Ax_i/M \rfloor)$ is 1 if and only if

$$A(x_i(\text{mod } Q)) - R\lfloor x_i/Q \rfloor < 0.$$

  Therefore, we can calculate $x_{i+1}$ as follows:
  We first compute

$$y = A(x_i(\text{mod } Q)) - R\lfloor x_i/Q \rfloor.$$

  If $y \geq 0$ then $x_{i+1} = y$, otherwise $x_{i+1} = y + M$.
- Below is a sample program for the above random number generator.

```
#include<stdio.h>
static int x=1;
#define A 48271L
#define M 2147483647L
#define Q (M/A)
#define R (M%A)
int next_rnd2(){
  x=A*(x%Q)-R*(x/Q);
  if (x<0) x=x+M;
  return x;
}
```

# subtractive method

- The above algorithm has the period of $M - 1$. If we want to generate a random sequence with longer period, the subtractive method introduced below can be used.

# subtractive method(contd)

- Let M be an even integer and $x_0, x_1, \ldots, x_{54}$ be a sequence of integers such that at least one of them is odd.
  Then the numbers generated by

$$x_n = (x_{n-24} - x_{n-55}) (\text{mod } M)$$

  have a period length of at least $2^{55} - 1$.
- Taking $M = 2^{31}$, below is a sample program for the above algorithm.

# subtractive method(contd 2)

```
static int A[55];
static int next=0;

init_rnd(){
  int i;
  for (i=0;i<55;i++) A[i]=next_rnd2();
}
```

# subtractive method(contd 3)

```
int next_rnd3(){
  int i, x;
  i=(next+31)%55;
  x=(A[i]-A[next]);
  if (x<0) {
    x+=2147483647; x+=1;}
  A[next]=x;
  next=(next+1)%55;
  return x;
}
```

# 64 bits-integer linear congruential method

We mainly discussed here the linear congruential method with 32-bits integer as a simple and traditional method. Today, 64-bits integer, "long", is available and common on many computers. The 64-bits linear congruential method with a bigger prime number, M, which can be expressed in 64 bit integer, might be a solution to get a longer period although it is still simple.