Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №1

З предмету: «Теорія ймовірності»

Виконав: студент групи ІО-22

Щербина Микита

Київ 2013р.

Завдання: Створити генератор псевдовипадкових послідовностей табличного типу на основі 8-ми генераторів LFSR рознесеного типу, з розрядністю першого регістру 6. Згенерувати послідовність довжиною 10000 і протестувати її на 4 тестах.

1. Частотний тест
2. Диференційний тест
3. Ранговий тест
4. Визначення лінійної складності

Лістинг програми

import random.TableGenerator;

import random.TestRandom;

public class Main {

static final int lengthSample = 10000;

static int[] outData = new int[lengthSample];

static final int window = 5;

static TableGenerator tableGenerator;

public static void main(String[] args) {

tableGenerator = new TableGenerator(6, 8);

for (int i = 0; i < outData.length; i++) {

outData[i] = tableGenerator.next();

}

System.out.println("Результаты тестов:");

System.out.println("частотный тест = "

+ TestRandom.frequencyTest(outData));

System.out.println("дифференциальный тест = "

+ TestRandom.differentialTest(outData));

System.out.println("Сложность = "+ TestRandom.complexityTest(outData));

System.out.println("ранговый тест");

double[] result = TestRandom.rankTest(outData, window);

for (int i = 0; i < result.length; i++) {

System.out.println(Integer.toBinaryString(i)+" = "+result[i]);

}

}

}

**package** random;

/\*\*

\* регистры памяти

\*

\*/

**public** **class** Registers {

**private** **int** length;//

**private** **long** registers;//

/\*\*

\* **@param** countRegister

\* - длина регистра

\*/

**public** Registers(**int** countRegister) {

**if** (countRegister > 0)

**this**.length = countRegister;

**else**

**throw** **new** IllegalArgumentException();

registers = 0;

}

/\*\*

\* **@param** countRegister

\* -длина регистра

\* **@param** value

\* - значение

\*/

**public** Registers(**int** countRegister, **long** value) {

**this**(countRegister);

registers = value;

}

/\*\*

\* возращает длину регистра

\*

\* **@return**

\*/

**public** **int** getLength() {

**return** length;

}

/\*\*

\* задает значение регистра

\*

\* **@param** number

\* - номер

\* **@param** value

\* - значение

\*/

**public** **void** setRigester(**int** number, **int** value) {

**if** ((number < 0) || (number >= length))

**throw** **new** IndexOutOfBoundsException();

**if** ((value < 0) || (value > 1))

**throw** **new** IllegalArgumentException();

**long** v = value << number;

registers = registers | v;

}

/\*\*

\* возращает значение регистра

\*

\* **@param** number

\* - номер регистра

\* **@return**

\*/

**public** **int** getRegister(**int** number) {

**if** ((number < 0) || (number >= length))

**throw** **new** IndexOutOfBoundsException();

**long** result = registers & (1 << number);

result = result >> number;

**return** (**int**) result;

}

/\*\*

\* возращает значение всех регистов

\*

\* **@return**

\*/

**public** **long** getAllRegister() {

**return** registers;

}

/\*\*

\* сдвиг влево на один разряд

\*/

**public** **void** leftShift() {

registers = registers << 1;

}

/\*\*

\* сдвиг вправо на один разряд

\*/

**public** **void** rightShift() {

registers = registers >> 1;

}

}

**package** random;

/\*\*

\* генератор LFSR рознесеного типа

\*

\*/

**public** **class** LFSR {

// степень первого полинома

**static** **final** **int** *START\_PRIME\_POLYNOMIALS* = 4;

// коэффициенты простых полиномов [4;20]

**static** **int**[][] *coefficientsPrimePolynomials* = { { 2 }, { 1 }, { 0 }, { 2 },

{ 3, 2, 1 }, { 3 }, { 2 }, { 1 }, { 5, 3, 0 }, { 3, 2, 0 },

{ 9, 5, 0 }, { 0 }, { 11, 2, 0 }, { 2 }, { 6 }, { 4, 1, 0 }, { 2 } };

**private** **int** indexPolynomial;// индекс полинома в массиве

**private** **int** indexLast;// индекс последнего элемента

**private** Registers register;// регистры

/\*\*

\* **@param** capacity

\* - разрядность

\* **@param** startValue

\* - начальное значение регистров

\*/

**public** LFSR(**int** capacity, **long** startValue) {

register = **new** Registers(capacity, startValue);

**if** (startValue == 0)

**throw** **new** IllegalArgumentException();

indexPolynomial = capacity - *START\_PRIME\_POLYNOMIALS*;

indexLast = capacity - 1;

}

/\*\*

\* функция обратной связи

\*

\* **@return**

\*/

**private** **int** f() {

**int** result = register.getRegister(indexLast);

**for** (**int** i = 0; i < *coefficientsPrimePolynomials*[indexPolynomial].length; i++) {

result = result

^ register

.getRegister(*coefficientsPrimePolynomials*[indexPolynomial][i]);

}

**return** result;

}

/\*\*возращает следующий бит

\* **@return**

\*/

**public** **int** next() {

**int** result = register.getRegister(indexLast);

**int** valueF = f();

register.setRigester(indexLast, 0);

register.leftShift();

register.setRigester(0, valueF);

**return** result;

}

/\*\* количество регистров памяти

\* **@return**

\*/

**public** **int** getCapacity() {

**return** register.getLength();

}

}

**package** random;

**import** java.util.Random;

/\*\*

\* табличный генератор

\*/

**public** **class** TableGenerator {

**private** **int**[] table;// таблица выбора выходного сигнала

**private** LFSR[] lfsr;// генераторы случайных чисел

/\*\*

\* инициализация таблицы выбора выходного сигнала

\*/

**private** **void** initializationTable() {

**int** n = 1 << lfsr.length;

**int**[] t = **new** **int**[n];// вспомогательная таблица

**for** (**int** i = 0; i < (n / 2); i++)

t[i] = 1;

**int** countSelected = 0;

table = **new** **int**[n];

Random random = **new** Random();

**while** (countSelected < n) {

**int** nextIndex = random.nextInt(n);

**if** (t[nextIndex] > -1) {

table[countSelected] = t[nextIndex];

countSelected++;

t[nextIndex] = -1;

}

}

}

/\*\*

\* **@param** capacity

\* - разрядность первого генератора

\* **@param** countLFSR

\* - количество генераторов

\*/

**public** TableGenerator(**int** capacity, **int** countLFSR) {

lfsr = **new** LFSR[countLFSR];

Random random = **new** Random();

**for** (**int** i = 0; i < lfsr.length; i++) {

**int** startValue = random.nextInt();

**while** (startValue == 0)

startValue = random.nextInt();

lfsr[i] = **new** LFSR(capacity + i, startValue);

}

initializationTable();

}

/\*\*

\* возращает следующий сигнал

\*

\* **@return**

\*/

**public** **int** next() {

**int** tableIndex = 0;

**for** (**int** i = 0; i < lfsr.length; i++) {

tableIndex += lfsr[i].next() << i;

}

**return** table[tableIndex];

}

}

Результати роботи програми:

частотный тест = 0.5033

дифференциальный тест = 0.49884988498849886

сложность = 5000

ранговый тест

00000 = 0.02961480740370185

00001 = 0.030015007503751877

00010 = 0.03171585792896448

00011 = 0.02991495747873937

00100 = 0.032916458229114556

00101 = 0.030015007503751877

00110 = 0.03231615807903952

00111 = 0.03091545772886443

01000 = 0.031015507753876938

01001 = 0.032416208104052026

01010 = 0.02961480740370185

01011 = 0.030515257628814407

01100 = 0.03211605802901451

01101 = 0.03221610805402701

01110 = 0.02831415707853927

01111 = 0.03311655827913957

10000 = 0.030015007503751877

10001 = 0.03161580790395198

10010 = 0.03121560780390195

10011 = 0.033316658329164583

10100 = 0.030515257628814407

10101 = 0.030215107553776887

10110 = 0.032016008004002

10111 = 0.030515257628814407

11000 = 0.030615307653826914

11001 = 0.03211605802901451

11010 = 0.031115557778889445

11011 = 0.032016008004002

11100 = 0.030615307653826914

11101 = 0.030815407703851928

11110 = 0.03311655827913957

11111 = 0.0335167583791896

Висновок: з результатів тестів видно, що у вихідній послідовності кількість 1 та 0 однакова,кількість диференційних одиниць та нулів теж однакова, відносні кількість груп символів рангового тесту у послідовності лежить у допустимих межах.