Kpi-best

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. І.СІКОРСЬКОГО»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №1**

з дисципліни «Безпека програм і даних»

на тему: «Методи формування псевдовипадкових двійкових послідовностей. Методики оцінки якості псевдовипадкових послідовностей.»

Виконав:

студент 4-го курсу

факультету ІОТ

групи ІП-54

Макаренко Антон

Перевірив:

Марковський О.П.

Київ 2018

**Завдання:**

Тип: Т(р8)

Розрядність: 6+

Об’єм вибірки: 10000

Складність: Л

**Лістинг:**

**Program.cs:**

using System;

using System.Collections.Generic;

namespace Lab1

{

class Program

{

static void Main(string[] args)

{

int[] selection = new int[10000];

int[][] polynoms = {

new int[]{ 1, 0, 0, 0, 0, 1 },

new int[]{ 0, 0, 1, 0, 0, 0, 1 },

new int[]{ 0, 1, 1, 1, 0, 0, 0, 1 },

new int[]{ 0, 0, 0, 1, 0, 0, 0, 0, 1 },

new int[]{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 1 },

new int[]{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 },

new int[]{ 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1 },

new int[]{ 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 }};

List<LFSR> lfsr = new List<LFSR>();

lfsr.Add(new LFSR(polynoms[0]));

lfsr.Add(new LFSR(polynoms[1]));

lfsr.Add(new LFSR(polynoms[2]));

lfsr.Add(new LFSR(polynoms[3]));

lfsr.Add(new LFSR(polynoms[4]));

lfsr.Add(new LFSR(polynoms[5]));

lfsr.Add(new LFSR(polynoms[6]));

lfsr.Add(new LFSR(polynoms[7]));

Table table = new Table();

for (int i = 0; i < selection.Length; i++)

{

int bufer;

bufer = table.generate(lfsr);

selection[i] = bufer;

Console.Write(bufer + ",");

}

Console.WriteLine("");

Console.WriteLine("Tests: ");

Console.WriteLine("1. Frequency: " + Tester.frequencyTest(selection));

Console.WriteLine("2. XOR: " + Tester.xorFrequencyTest(selection));

int[] ranks = Tester.rankTest(selection, 3);

Console.WriteLine("3. Rank: ");

for (int i = 0; i < ranks.Length; i++)

{

Console.WriteLine(Convert.ToString(i, 2) + " " + (double)ranks[i]/\*/selection.length\*/);

}

Console.WriteLine("4. Complexity: ");

Tester.complexityTest(selection);

}

}

}

**LFSR.cs:**

using System;

namespace Lab1

{

class LFSR

{

int[] register;

int output;

int[] polynom;

int capacity;

public LFSR(int[] polynom)

{

this.register = new int[polynom.Length];

capacity = this.register.Length - 1;

Random rand = new Random();

for (int i = 0; i <= capacity; i++)

{

this.register[i] = rand.Next(2);

}

int bufer = this.register[capacity];

for (int i = capacity; i > 0; i--)

{

if (polynom[i - 1] == 1)

{

this.register[i] = this.register[i - 1] ^ bufer;

}

else

{

this.register[i] = this.register[i - 1];

}

}

this.register[0] = bufer;

this.polynom = polynom;

}

public int generateStep()

{

int bufer = this.register[capacity];

for (int i = capacity; i > 0; i--)

{

if (this.polynom[i - 1] == 1)

{

this.register[i] = this.register[i - 1] ^ bufer;

}

else

{

this.register[i] = this.register[i - 1];

}

}

this.register[0] = bufer;

output = bufer;

return output;

}

}

}

**Table.cs:**

using System;

using System.Collections.Generic;

namespace Lab1

{

class Table

{

private int[] table = new int[256];

private int position = 0;

const int NUMBER\_LFSR = 8;

public Table()

{

int number0 = 0;

int number1 = 0;

for (int i = 0; i < table.Length; i++)

{

Random rand = new Random();

table[i] = rand.Next(2);

if (table[i] == 0) { number0++; }

else { number1++; }

if (number0 == table.Length / 2)

{

for (int j = i; j < table.Length; j++)

{

table[j] = 1;

}

break;

}

if (number1 == table.Length / 2)

{

for (int j = i; j < table.Length; j++)

{

table[j] = 0;

}

break;

}

}

}

public int generate(List<LFSR> lfsr)

{

int output = 0;

position = 0;

int[] number = new int[NUMBER\_LFSR];

for (int i = 0; i < number.Length; i++)

{

number[i] = lfsr[i].generateStep();

}

for (int i = 0; i < number.Length; i++)

{

if (number[i] == 1)

{

position = position + (int)(Math.Pow(2, i));

}

}

output = table[position];

return output;

}

}

}

**Tester.cs:**

using System;

namespace Lab1

{

class Tester

{

public static double frequencyTest(int[] arr)

{

int output = 0;

for (int i = 0; i < arr.Length; i++)

{

if (arr[i] == 1)

{

output++;

}

}

return ((double)output / arr.Length);

}

public static double xorFrequencyTest(int[] arr)

{

int output = 0;

for (int i = 0; i < arr.Length - 1; i++)

{

if ((arr[i] ^ arr[i + 1]) == 1)

{

output++;

}

}

return ((double)output / (arr.Length - 1));

}

public static int[] rankTest(int[] arr, int width)

{

if (width <= 0 || width > 10)

{

throw new ArgumentException("Width must be range [2;10]");

}

int[] output = new int[(int)Math.Pow(2, width)];

for (int i = 0; i < arr.Length - width; i++)

{

int bufer = 0;

for (int j = 0; j < width; j++)

{

bufer = bufer << 1;

bufer = bufer | arr[i + j];

}

output[bufer]++;

}

return output;

}

public static void complexityTest(int[] arr)

{

int N = arr.Length;

int[] b = new int[N];

int[] c = new int[N];

int[] t = new int[N];

b[0] = 1;

c[0] = 1;

int l = 0;

int m = -1;

for (int n = 0; n < N; n++)

{

int d = 0;

for (int i = 0; i <= l; i++)

{

d ^= c[i] \* arr[n - i];

}

if (d == 1)

{

Array.Copy(c, 0, t, 0, N);

int NiM = n - m;

for (int j = 0; j < N - NiM; j++)

{

c[NiM + j] ^= b[j];

}

if (l <= n / 2)

{

l = n + 1 - l;

m = n;

Array.Copy(t, 0, b, 0, N);

}

}

}

Console.WriteLine(l);

}

}

}