Министерство образования республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Институт информационных технологий

Кафедра ПОИТ

**Отчет по лабораторной работе №3**

**по курсу САиММод**

**Исследование дискретно-стохастической СМО**

Выполнили:

студенты группы 381062

Радивил Д.В.

Черник М.Л.

Шкундалёва О.В.

Проверила: Лашкевич Е.М.

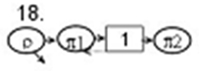
Минск 2016

**Задание**

Исследовать работу дискретно-стохастической СМО:

1) разработать алгоритм работы дискретно-стохастической СМО

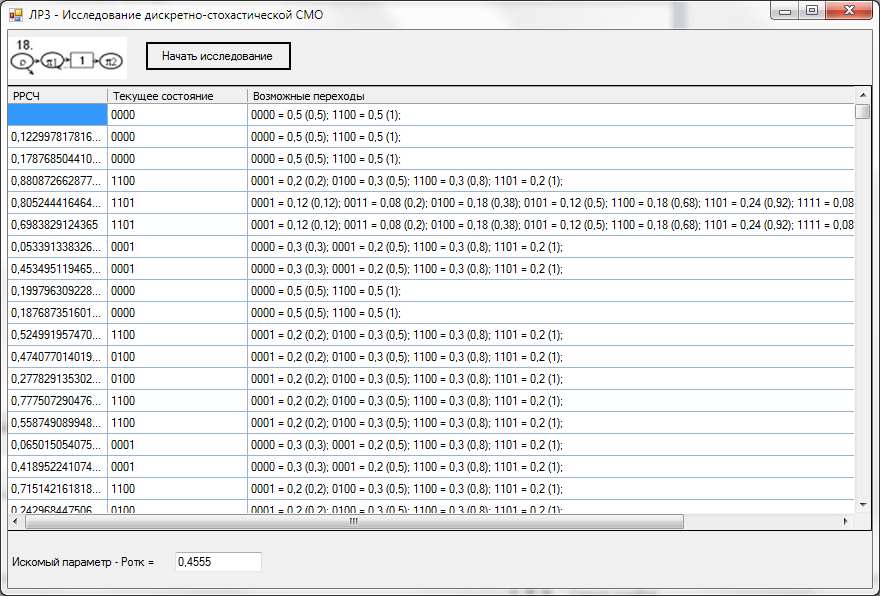
2) написать имитирующую программу, предусматривающую сбор и статистическую обработку данных для получения оценок заданных характеристик СМО).



Вариант 18:

Цель исследования - Ротк

**Результат выполнения программы:**



**Код программы:**

class Graph

{

protected State[] vertices;

protected double[,] transitions = {

{0.5, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0},

{0.3, 0.2, 0, 0, 0, 0, 0.3, 0.2, 0, 0, 0},

{0, 0.3, 0.2, 0, 0, 0, 0, 0.3, 0.2, 0, 0},

{0, 0.2, 0, 0.3, 0, 0, 0.3, 0.2, 0, 0, 0},

{0, 0.12, 0.08, 0.18, 0.12, 0, 0.18, 0.24, 0.08, 0, 0},

{0, 0, 0.12, 0, 0.18, 0.12, 0, 0.18, 0.24, 0.08, 0.08},

{0, 0.2, 0, 0.3, 0, 0, 0.3, 0.2, 0, 0, 0},

{0, 0.12, 0.08, 0.18, 0.12, 0, 0.18, 0.24, 0.08, 0, 0},

{0, 0, 0.12, 0, 0.18, 0.12, 0, 0.18, 0.24, 0.08, 0.08},

{0, 0, 0.3, 0, 0, 0, 0, 0, 0.3, 0.2, 0.2},

{0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0, 0.5}

};

public Graph()

{

string[] states = { "0000", "0001", "0011", "0100", "0101", "0111", "1100", "1101", "1111", "0211", "1211" };

vertices = new State[states.Length];

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

{

vertices[i] = new State(states[i]);

}

}

public IReadOnlyDictionary<State, double> getTransitions(State state)

{

IDictionary<State, double> states = new Dictionary<State, double>();

int j = vertices.ToList().FindIndex(x => state.Equals(x));

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

{

if (transitions[j, i] != 0)

{

states[vertices[i]] = transitions[j, i];

}

}

return (IReadOnlyDictionary<State, double>) states;

}

public IReadOnlyList<string> getVertices()

{

return (IReadOnlyList<string>)vertices.ToList();

}

class Immitation

{

protected int \_iterationsCount;

protected double \_lastRandNumber;

protected State \_startState;

protected State \_state;

protected LehmerRandom \_random;

protected Graph \_graph;

public Immitation(Graph graph, LehmerRandom random, State start)

{

\_graph = graph;

\_random = random;

\_startState = start;

reset();

}

public State CurrentState

{

get { return \_state; }

}

public int IterationsCount

{

get { return \_iterationsCount; }

}

public double LastRandom

{

get { return \_lastRandNumber; }

}

public void reset()

{

\_state = \_startState;

\_iterationsCount = 0;

\_lastRandNumber = 0;

}

public void iteration()

{

IReadOnlyDictionary<State, double> transitions = \_graph.getTransitions(\_state);

if (transitions.Count == 1)

{

\_state = transitions.Keys.ToArray()[0];

\_lastRandNumber = 0;

}

else

{

\_lastRandNumber = \_random.Next();

double L = 0;

foreach (KeyValuePair<State, double> pair in transitions)

{

if (\_lastRandNumber > L && \_lastRandNumber <= L + pair.Value)

{

\_state = pair.Key;

break;

}

L += pair.Value;

}

}

\_iterationsCount++;

}

}

class LehmerRandom

{

private readonly uint[] \_simples =

{

3, 7, 11, 17, 23, 29, 37, 47, 59, 71, 89, 107, 131, 163, 197, 239, 293, 353, 431, 521, 631, 761, 919,

1103, 1327, 1597, 1931, 2333, 2801, 3371, 4049, 4861, 5839, 7013, 8419, 10103, 12143, 14591,

17519, 21023, 25229, 30293, 36353, 43627, 52361, 62851, 75431, 90523, 108631, 130363, 156437,

187751, 225307, 270371, 324449, 389357, 467237, 560689, 672827, 807403, 968897, 1162687, 1395263,

1674319, 2009191, 2411033, 2893249, 3471899, 4166287, 4999559, 5999471, 7199369

};

private ulong a, m, Rprev, c;

public LehmerRandom(ulong a, ulong m, ulong c)

{

List<uint> simples = new List<uint>();

foreach (uint p in \_simples)

{

if (p < m - 1)

{

simples.Add(p);

}

}

this.a = a;

this.m = m;

this.Rprev = (ulong)simples[new Random().Next(simples.Count)];

this.c = c;

}

public double Next()

{

double result = Rprev;

Rprev = (a \* Rprev + c) % m;

return result / m;

}

}

class State: Object

{

public int T { get; set; }

public int N { get; set; }

public int K1 { get; set; }

public int K2 { get; set; }

public State()

{

T = 2;

N = 0;

K1 = 0;

K2 = 0;

}

public State(string state)

{

setState(state);

}

public override string ToString()

{

return T.ToString() + N.ToString() + K1.ToString() + K2.ToString();

}

public override bool Equals(object obj)

{

return obj.ToString() == this.ToString();

}

public void setState(string state)

{

T = Int32.Parse(state.Substring(0, 1));

N = Int32.Parse(state.Substring(1, 1));

K1 = Int32.Parse(state.Substring(2, 1));

K2 = Int32.Parse(state.Substring(3, 1));

}

public override int GetHashCode()

{

return Int32.Parse(ToString());

}

}