БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра программного обеспечения информационных технологий

Факультет компьютерных систем и сетей
Кафедра программного обеспечения информационных технологий

Отчет
по лабораторной работе № 3
по дисциплине «Системный анализ и машинное моделирование»
на тему «Построение и исследование аналитической модели дискретно - стохастической системы массового обслуживания»

Выполнил студент: Верещагин Н. В.
группы 851006

Проверил: Мельник Н.И.

Минск 2021

1. ЗАДАНИЕ

Варианта задания 10.

ρ	π1	π2
0,5	0,6	0,6

1.1 Построить граф состояний Р-схемы, представленной на рисунке 1.

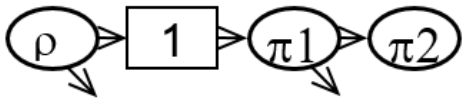

Рисунок 1 – Исходная Р-схема.

1.2 Раскрыть смысл кодировок состояний.

1.3 Построить аналитическую модель по графу, решить ее и определить вероятности состояний.

1.4 Рассчитать теоретические значения показателей эффективности.

1.5 Построить имитационную модель для данной СМО.

2. РЕШЕНИЕ

Состояние системы будет определяться трёхкомпонентным вектором: jt_1t_2.

j – количество заявок, находящихся в накопители (длина очереди), j = {0, 1}.

t_1 и t_2 – определяют состояния каналов обслуживания и могут принимать два значения: при 0 – канал свободен, 1 – канал занят обслуживанием заявки, t_1 = {0, 1} и t_2 = {0, 1}.

Граф состояний Р-схемы представлен на рисунке 2.

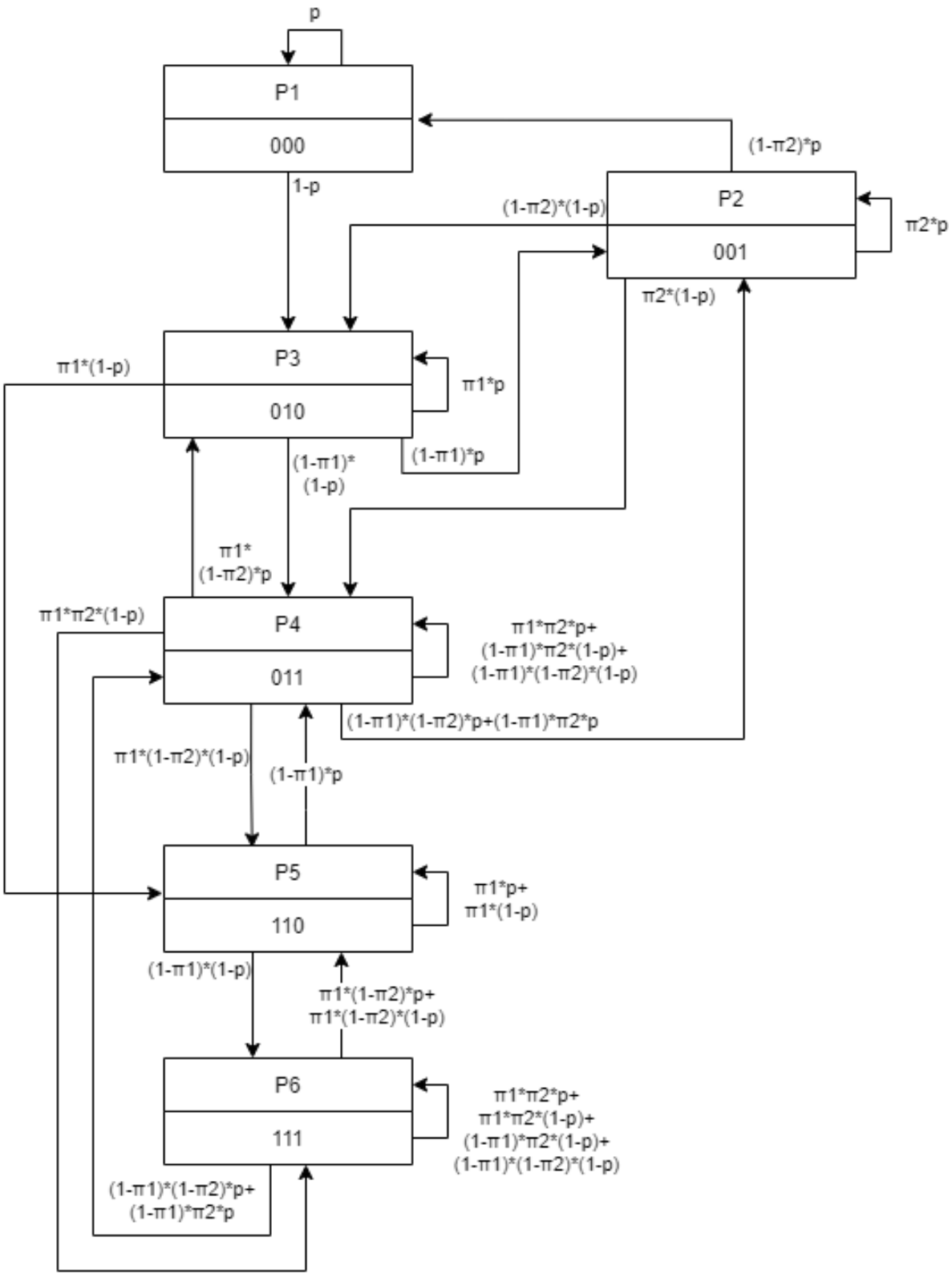

Рисунок 2 – Граф состояний Р-схемы.

С помощью графа построим аналитическую модель:

$$P1 = p * P1 + (1 - pi2) * p * P2;$$

$$P2 = pi2 * p * P2 + ((1 - pi1) * (1 - pi2) * p + (1 - pi1) * pi2 * p) * P4 + ((1 - pi1) * p) * P3;$$

$$P3 = pi1 * p * P3 + (1 - p) * P1 + \big((1 - pi2) * (1 - p)\big) * P2 + (pi1 * (1 - pi2) * p) * P4;$$

$$P4 = \big(pi1 * pi2 * p + (1 - pi1) * pi2 * (1 - p) + (1 - pi1) * (1 - pi2) * (1 - p)\big) * P4 + \big(pi2 * (1 - p)\big) * P2 + \big((1 - pi1) * (1 - p)\big) * P3 + \big((1 - pi1) * (1 - pi2) * p + (1 - pi1) * pi2 * p\big) * P6 + \big((1 - pi1) * p\big) * P5;$$

$$P5 = \big(pi1 * p + pi1 * (1 - p)\big) * P5 + \big(pi1 * (1 - pi2) * p + pi1 * (1 - pi2) * (1 - p)\big) * P6 + \big(pi1 * (1 - p)\big) * P3 + \big(pi1 * (1 - pi2) * (1 - p)\big) * P4;$$

$$P6 = (pi1 * pi2 * p + pi1 * pi2 * (1 - p) + (1 - pi1) * pi2 * (1 - p) + (1 - pi1) * (1 - pi2) * (1 - p)) * P6 + ((1 - pi1) * (1 - p)) * P5 + (pi1 * pi2 * (1 - p)) * P4.$$

Дополнив модель нормировочным уравнением и подставив данные из задания, получим значения вероятностей состояний, представленных на рисунке 3.

$$P_1 + P_2 + P_3 + P_4 + P_5 + P_6 = 1$$

```
{{P1 → 0.0394089, P2 → 0.0985222,
  P3 → 0.0985222, P4 → 0.246305, P5 → 0.28633, P6 → 0.230911}}
```

Рисунок 3 – Вероятности состояний

Из полученных значений можно сделать вывод, что состоянии, в котором система будет находиться больше всего - {1, 1, 0}, а меньше всего {0, 0, 0}.

Произведем расчет теоретических значений показателей эффективности:

Абсолютная пропускная способность

$$A = (P_2 + P_4 + P_6)(1 - \pi_2) = 0.23029528$$

Относительная пропускная способность

$$Q = \frac{A}{1 - p} = 0.46059056$$

Вероятность отказа

$$P_{\text{отк}} = 1 - Q = 0.53940944$$

Вероятность блокировки

$$P_{\text{бл}} = 0$$

Средняя длина очереди

$$L_{\text{оч}} = P_1 * 0 + P_2 * 0 + P_3 * 0 + P_4 * 0 + P_5 * 1 + P_6 * 1 = 0.517241$$

Среднее число заявок, находящихся в системе

$$L_{\text{с}} = P_1 * 0 + P_2 * 1 + P_3 * 1 + P_4 * 2 + P_5 * 2 + P_6 * 3 = 1.9550474$$

Среднее время пребывания заявки в очереди

$$W_{\text{оч}} = \frac{(P_5 * 1 + P_6 * 1)}{(P_3 + P_4 + P_5 + P_6)(1 - \pi_1)} = 1.5$$

Среднее время пребывания заявки в системе

$$W_{\text{с}} = W_{\text{оч}} + \frac{1}{1 - pi_1} + \frac{1}{1 - pi_2} = 6.5$$

Коэффициент загрузки канала 1

$$K_1 = P_3 + P_4 + P_5 + P_6 = 0.8620682$$

Коэффициент загрузки канала 2

$$K_2 = P_2 + P_4 + P_6 = 0.5757382$$

Пример работы имитационной модели:

```
Введите ρ: 0.5
Введите π1: 0.6
Введите π2: 0.6

Вероятности состояний
---------------------
P1 = 0.0394068
P2 = 0.0986623
P3 = 0.0984087
P4 = 0.2461305
P5 = 0.2862794
P6 = 0.2311123

Теоритические значения
----------------------
A = 0.2302903
Q = 0.4606485917321397
Ротк = 0.5393512082383359
Рбл = 0
Lоч = 0.5173917
Lс = 1.9552277
Wоч = 1.5006762723090679
Wс = 6.500676272309068
K1 = 0.8619309
K2 = 0.5759051
```

Рисунок 3 – Пример работы имитационной модели.

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace SAIMOD
{
    class Program
    {
        static void Main(string[] args)
        {
            Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
            var enc1251 = Encoding.GetEncoding(1251);
            System.Console.OutputEncoding = System.Text.Encoding.UTF8;
            System.Console.InputEncoding = enc1251;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.BackgroundColor = ConsoleColor.White;
            Console.Clear();

            const double tickCount = 10000000;
```

```
            Console.Write("Введите p: ");
            var p = double.Parse(Console.ReadLine());

            Console.Write("Введите p1: ");
            var p1 = double.Parse(Console.ReadLine());

            Console.Write("Введите p2: ");
            var p2 = double.Parse(Console.ReadLine());

            var generator = new TrueGenerator(p, p1, p2);
            for (var i = 0; i < tickCount; i++)
            {
                generator.GenerateNextState();
            }

            Console.WriteLine("\nВероятности состояний\n"
                              + "----------------------");
            string[] transscript = new string[7];
            transscript[1] = "000";
            transscript[2] = "001";
            transscript[3] = "010";
            transscript[4] = "011";
            transscript[5] = "110";
            transscript[6] = "111";
            for (int i = 1; i < 7; ++i)
            {
                Console.WriteLine($"P{i} =
{generator.StateCount[transscript[i]] / tickCount}");

            }

            var Woch = (generator.QueueLength / tickCount) / (
                       (generator.StateCount["010"] / tickCount +
                        generator.StateCount["011"] / tickCount +
                        generator.StateCount["110"] / tickCount +
                        generator.StateCount["111"] / tickCount) *
                       (1 - p1));

            Console.WriteLine("\nТеоритические значения\n"
                              + "----------------------");
            Console.WriteLine($"A = {generator.ProcessedCount / tickCount}");
            Console.WriteLine($"Q = {generator.ProcessedCount /
(double)generator.GeneratedCount}");
            Console.WriteLine($"Ротк = {generator.DeclineCount /
(double)generator.GeneratedCount}");
            Console.WriteLine($"Рбл = {generator.BlockCount / tickCount}");
            Console.WriteLine($"Lоч = {generator.QueueLength / tickCount}");
            Console.WriteLine($"Lс = {generator.RequestLength / tickCount}");
            Console.WriteLine($"Wоч = {Woch}");
            Console.WriteLine($"Wс = {Woch + (1 / (1 - p1)) + (1 / (1 -
p2))}");
            Console.WriteLine($"K1 = {generator.FirstChannel / tickCount}");
            Console.WriteLine($"K2 = {generator.SecondChannel / tickCount}");
        }
    }

    public class Request
```

```
{
    public int State;
    public int InQuery;
    public int InSystem;
}

public class TrueGenerator
{

    private readonly Random _generateRandom;
    private readonly Random _serviceFirstRandom;
    private readonly Random _serviceSecondRandom;

    public readonly Dictionary<string, int> StateCount;

    private readonly double _p;
    private readonly double _pi1;
    private readonly double _pi2;

    public byte J { get; set; }
    public byte T1 { get; set; }
    public byte T2 { get; set; }

    public int FirstChannel { get; private set; }
    public int SecondChannel { get; private set; }

    public int QueueLength { get; private set; }
    public int RequestLength { get; private set; }

    public int DeclineCount { get; private set; }
    public int BlockCount { get; private set; }
    public int InSystemCount { get; private set; }

    public int ProcessedCount { get; private set; }
    public int GeneratedCount { get; private set; }

    public Request FirstChannelRequest { get; set; }
    public Request SecondChannelRequest { get; set; }
    public Request FirstQueueRequest { get; set; }
    public Request SecondQueueRequest { get; set; }

    public List<Request> InSystemRequests { get; private set; }
    public TrueGenerator(double p, double pi1, double pi2)
    {
        _generateRandom = new Random();
        _serviceFirstRandom = new Random();
        _serviceSecondRandom = new Random();
        _p = p;
        _pi1 = pi1;
        _pi2 = pi2;
        InSystemRequests = new List<Request>();
        StateCount = new Dictionary<string, int>();

        FirstChannelRequest = new Request();
        SecondChannelRequest = new Request();
        FirstQueueRequest = new Request();
        SecondQueueRequest = new Request();
```

```
}

private bool IsRequestGenerated()
{
    return _generateRandom.NextDouble() <= 1 - _p;
}

private bool IsRequestServicedFirst()
{
    return _serviceFirstRandom.NextDouble() <= 1 - _pi1;
}

private bool IsRequestServicedSecond()
{
    return _serviceSecondRandom.NextDouble() <= 1 - _pi2;
}

public void GenerateNextState()
{
    var isGenerated = IsRequestGenerated();
    var isServiceFirst = IsRequestServicedFirst();
    var isServiceSecond = IsRequestServicedSecond();


    if (isServiceSecond)
    {
        if (T2 == 1)
        {
            InSystemRequests.Add(new Request()
            {
                InQuery = SecondChannelRequest.InQuery,
                InSystem = SecondChannelRequest.InSystem,
                State = 4
            });

            SecondChannelRequest.InQuery = 0;
            SecondChannelRequest.InSystem = 0;
            SecondChannelRequest.State = 0;
        }
        ProcessedCount += T2;
        T2 = 0;
    }

    if (isServiceFirst)
    {
        if (T1 == 1)
        {
            if (T2 == 1)
            {
                DeclineCount += 1;

                InSystemRequests.Add(new Request()
                {
                    InQuery = FirstChannelRequest.InQuery,
                    InSystem = FirstChannelRequest.InSystem,
                    State = 4
                });
```

```
                    }
                    else
                    {
                        T2 = 1;

                        SecondChannelRequest.State =
FirstChannelRequest.State;
                        SecondChannelRequest.InQuery =
FirstChannelRequest.InQuery;
                        SecondChannelRequest.InSystem =
FirstChannelRequest.InSystem;
                    }


                    FirstChannelRequest.State = 0;
                    FirstChannelRequest.InQuery = 0;
                    FirstChannelRequest.InSystem = 0;
                }

                T1 = 0;
            }

            if (T1 == 0)
            {
                if (J > 0)
                {
                    T1 = 1;
                    J--;

                    FirstChannelRequest.State = FirstQueueRequest.State;
                    FirstChannelRequest.InQuery = FirstQueueRequest.InQuery;
                    FirstChannelRequest.InSystem =
FirstQueueRequest.InSystem;

                    FirstQueueRequest.State = SecondQueueRequest.State;
                    FirstQueueRequest.InQuery = SecondQueueRequest.InQuery;
                    FirstQueueRequest.InSystem = SecondQueueRequest.InSystem;

                    SecondQueueRequest.State = 0;
                    SecondQueueRequest.InQuery = 0;
                    SecondQueueRequest.InSystem = 0;
                }
            }

            if (isGenerated)
            {
                GeneratedCount += 1;
                if (J < 1)
                {
                    J++;
                    InSystemCount += 1;

                    if (J == 1)
                    {
                        FirstQueueRequest.State = 0;
                        FirstQueueRequest.InQuery = 0;
                        FirstQueueRequest.InSystem = 0;
```

```
                        SecondQueueRequest.State = 0;
                        SecondQueueRequest.InQuery = 0;
                        SecondQueueRequest.InSystem = 0;
                    }
                }
                else
                {
                    DeclineCount++;
                }

                if (T1 == 0)
                {
                    J--;

                    T1 = 1;

                    FirstChannelRequest.State = FirstQueueRequest.State;
                    FirstChannelRequest.InQuery = FirstQueueRequest.InQuery;
                    FirstChannelRequest.InSystem =
FirstQueueRequest.InSystem;

                    FirstQueueRequest.State = SecondQueueRequest.State;
                    FirstQueueRequest.InQuery = SecondQueueRequest.InQuery;
                    FirstQueueRequest.InSystem = SecondQueueRequest.InSystem;

                    SecondQueueRequest.State = 0;
                    SecondQueueRequest.InQuery = 0;
                    SecondQueueRequest.InSystem = 0;
                }
            }

            QueueLength += J;
            RequestLength += J + T1 + T2;
            FirstChannel += T1;
            SecondChannel += T2;

            FirstQueueRequest.InQuery += 1;
            SecondQueueRequest.InQuery += 1;

            FirstQueueRequest.InSystem += 1;
            SecondQueueRequest.InSystem += 1;
            FirstChannelRequest.InSystem += 1;
            SecondChannelRequest.InSystem += 1;

            if (StateCount.ContainsKey($"{J}{T1}{T2}"))
            {
                StateCount[$"{J}{T1}{T2}"]++;
            }
            else
            {
                StateCount[$"{J}{T1}{T2}"] = 1;
            }
        }
    }

}
```