МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Качество и метрология программного обеспечения»

ТЕМА: ПОСТРОЕНИЕ ОПЕРАЦИОННОЙ ГРАФОВОЙ МОДЕЛИ ПРОГРАММЫ (ОГМП) И РАСЧЕТ ХАРАКТЕРИСТИК ЭФФЕКТИВНОСТИ ЕЕ ВЫПОЛНЕНИЯ МЕТОДОМ ЭКВИВАЛЕНТНЫХ ПРЕОБРАЗОВАНИЙ

Студент гр. 6304 _____ Виноградов К.А.

Преподаватель _____ Кирьянчиков В.А.

Санкт-Петербург

2020

Цель работы.

Построение операционной графовой модели программы (ОГМП) и расчет характеристик эффективности ее выполнения методом эквивалентных преобразований.

Формулировка задания.

1. Построение ОГМП.

Для рассматривавшегося в лабораторных работах 1-3 индивидуального задания разработать операционную модель управляющего графа программы на основе схемы алгоритма. При выполнении работы рекомендуется для упрощения обработки графа исключить диалог при выполнении операций ввода-вывода данных, а также привести программу к структурированному виду.

Выбрать вариант графа с нагруженными дугами, каждая из которых должна представлять фрагмент программы, соответствующий линейному участку или ветвлению. При расчете вероятностей ветвлений, зависящих от распределения данных, принять равномерное распределение обрабатываемых данных в ограниченном диапазоне (например, [0,100] - для положительных чисел или [-100,100] - для произвольных чисел). В случае ветвлений, вызванных проверкой выхода из цикла, вероятности рассчитываются исходя априорных сведений о числе повторений цикла. Сложные случаи оценки вероятностей ветвлений согласовать с преподавателем.

В качестве параметров, характеризующих потребление ресурсов, использовать времена выполнения команд соответствующих участков программы. С помощью монитора Sampler выполнить оценку времен выполнения каждого линейного участка в графе программы.

2. Расчет характеристик эффективности выполнения программы методом эквивалентных преобразований.

Полученную в части 2.1 данной работы ОГМП, представить в виде графа с нагруженными дугами, у которого в качестве параметров, характеризующих потребление ресурсов на дуге ij, использовать тройку { Pij,Mij,Dij }, где:

Pij - вероятность выполнения процесса для дуги ij,

Mij - мат.ожидание потребления ресурса процессом для дуги ij,

Dij - дисперсия потребления ресурса процессом для дуги ij.

В качестве потребляемого ресурса в данной работе рассматривается время процессора, а оценками мат. ожиданий времен для дуг исходного графа следует принять времена выполнения операторов (команд), соответствующих этим дугам участков программы. Дисперсиям исходных дуг следует присвоить нулевые значения.

Получить описание полученной ОГМП на входном языке пакета CSA III в виде поглощающей марковской цепи (ПМЦ) – (англ.) AMC (absorbing Markov chain) и/или эргодической марковской цепи (ЭМЦ) - EMC (ergodic Markov chain).

С помощью предоставляемого пакетом CSA III меню действий выполнить расчет среднего времени и дисперсии времени выполнения как для всей программы, так и для ее фрагментов, согласованных с преподавателем. Сравнить полученные результаты с результатами измерений, полученными в работе 3.

Ход работы.

Исходный тест программы:

```
#include <math.h>
#include "Sampler.h"

const double tol = 1.0e-6,
             a = 18.19,
             b = -23180.0,
             c = -.8858,
             logp = -4.60517;

void func(double* x, double* fx, double* dfx)
{
    *fx = a + b / *x + c * log(*x) - logp;
    *dfx = -b / (*x * *x) + c / *x;
}

void newton(double *x, double *fx, double *dx,
            double *dfx, double *x1)
{
    do
    {
        *x1 = *x;                                   // #3
        func(x, fx, dfx);                           // #4
        if(fabs(*dfx) < tol)                        // #5
        {
            if(*dfx >= .0)                          // #6
          {
                *dfx = tol;                         // #7
          }
            else
          {
                *dfx = -tol;                        // #8
          }                                         // #9
        }                                           // #10
        *dx = *fx / *dfx;
        *x = *x1 - *dx;
    }
    while(fabs(*dx) >= fabs(tol * *x));             // #11
}

int main()
{                                                   // #1
    double x = 5.0, fx, dfx, dx, x1;
    newton(&x, &fx, &dx, &dfx, &x1);                // #2
    return 0;                                       // #12
}                                                   // #13
```

Граф управления программы:

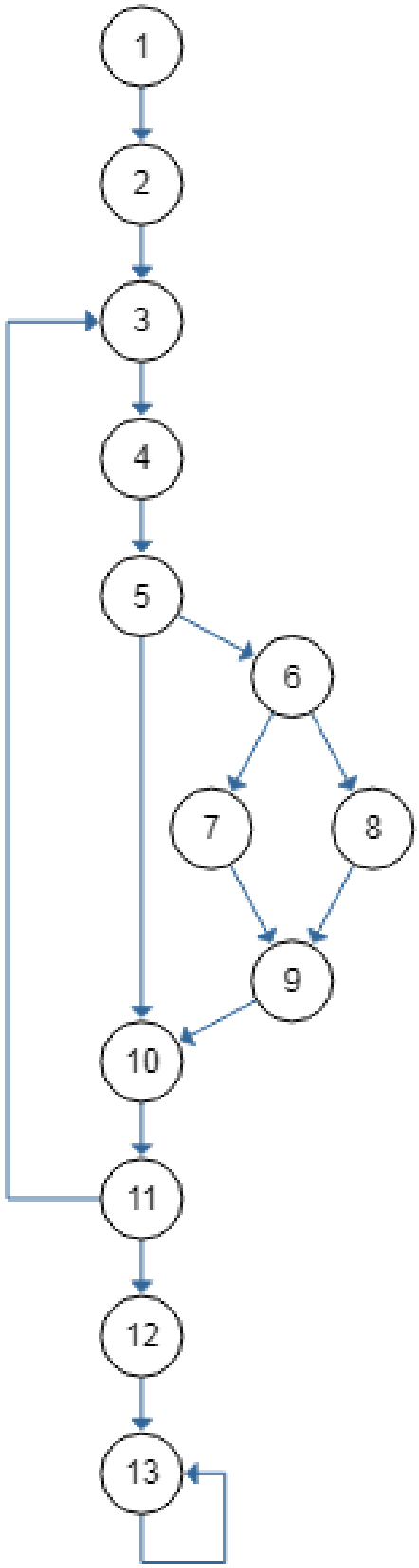

Рисунок 1 – Граф управления программы

Подготовленный для профилирования текст программы:

```
1.  #include <math.h>
2.  #include "Sampler.h"
3.
4.  const double tol = 1.0e-6,
5.                a = 18.19,
6.                b = -23180.0,
7.                c = -.8858,
8.                logp = -4.60517;
9.
10. void func(double* x, double* fx, double* dfx)
11. {
12.     SAMPLE;
13.     *fx = a + b / *x + c * log(*x) - logp;
14.     *dfx = -b / (*x * *x) + c / *x;
15.     SAMPLE;
16. }
17.
18. void newton(double *x, double *fx, double *dx, double *dfx, double *x1)
19. {
20.     SAMPLE;
21.     do
22.     {
23.         SAMPLE;
24.         *x1 = *x;                                   // #3
25.         func(x, fx, dfx);                           // #4
26.         SAMPLE;
27.         if(fabs(*dfx) < tol)                        // #5
28.         {
29.             SAMPLE;
30.             if(*dfx >= .0)                          // #6
31.             {
32.                 SAMPLE;
33.                 *dfx = tol;                         // #7
34.                 SAMPLE;
35.             }
36.             else
37.             {
38.                 SAMPLE;
39.                 *dfx = -tol;                        // #8
40.                 SAMPLE;
41.             }                                       // #9
42.             SAMPLE;
43.         }                                           // #10
44.         SAMPLE;
45.         *dx = *fx / *dfx;
46.         *x = *x1 - *dx;
47.         SAMPLE;
48.     }
49.     while(fabs(*dx) >= fabs(tol * *x));             // #11
50.     SAMPLE;
51. }
52.
53. int main()
54. {                                                   // #1
55.     SAMPLE;
56.     double x = 5.0, fx, dfx, dx, x1;
57.     newton(&x, &fx, &dx, &dfx, &x1);                // #2
58.     SAMPLE;
59.     return 0;                                       // #12
```

```
60.}                                                    // #13
```

Результаты профилирования:

```
-------------------------------------------------------------------
 NN          Имя обработанного файла
-------------------------------------------------------------------
  1.  MAIN_FU.CPP
-------------------------------------------------------------------


  Таблица с результатами измерений ( используется 10 из 416 записей )

-------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
-------------------------------------------------------------------
 1 : 12  1 : 15          846.48          13          65.11
-------------------------------------------------------------------
 1 : 15  1 : 26          32.69           13          2.51
-------------------------------------------------------------------
 1 : 20  1 : 23          0.00            1           0.00
-------------------------------------------------------------------
 1 : 23  1 : 12          132.42          13          10.19
-------------------------------------------------------------------
 1 : 26  1 : 44          258.13          13          19.86
-------------------------------------------------------------------
 1 : 44  1 : 47          232.15          13          17.86
-------------------------------------------------------------------
 1 : 47  1 : 23          421.56          12          35.13
 1 : 47  1 : 50          18.44           1           18.44
-------------------------------------------------------------------
 1 : 50  1 : 58          2.51            1           2.51
-------------------------------------------------------------------
 1 : 55  1 : 20          58.67           1           58.67
-------------------------------------------------------------------
```

Рисунок 2 – Результаты профилирования

Расчет вероятностей и затрат ресурсов дуг управляющего графа:

Дуга и время дуги	Номера строк	Количество проходов
L1 = 58.67 мкс	55:20	1
L2 = 0.00 мкс	20:23	1
L3 = 10.19 мкс	23:12	1
L4 = 67.62 мкс	12:15 ; 15:26	1 ; 1
L5 = 19.86 мкс	26:44 ; 26:29	1 ; 0
L6 = 0.0 мкс	29:32 ; 29:38	0 ; 0
L7 = 0.0 мкс	32:34 ; 34:42	0 ; 0
L8 = 0.0 мкс	38:40 ; 40:42	0 ; 0

L9 = 0.0 мкс	42:44	0
L10 = 17.86 мкс	44:47	1
L11 = 53.57 мкс	47:23 ; 47:50	0.92 ; 0.08
L12 = 2.51 мкс	50:58	1

Операционная графовая модель программы:

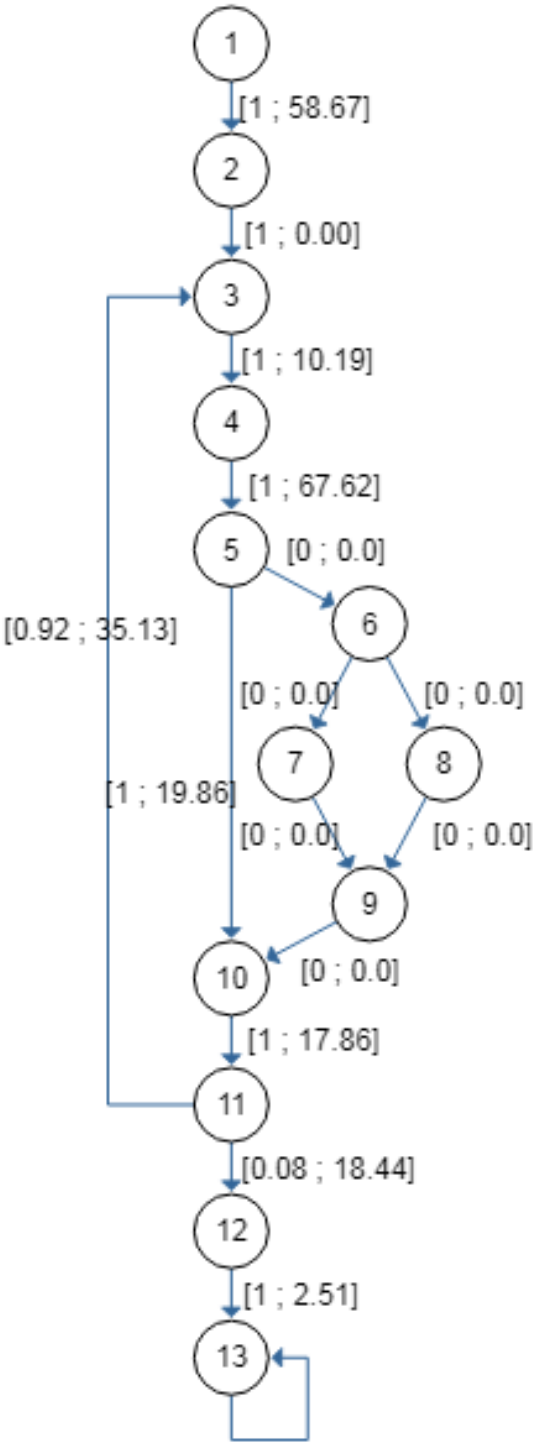

Рисунок 3 – Операционная графовая модель программы

Расчет характеристик эффективности выполнения программы с помощью пакета CSA III методом эквивалентных преобразований:

ГНД:

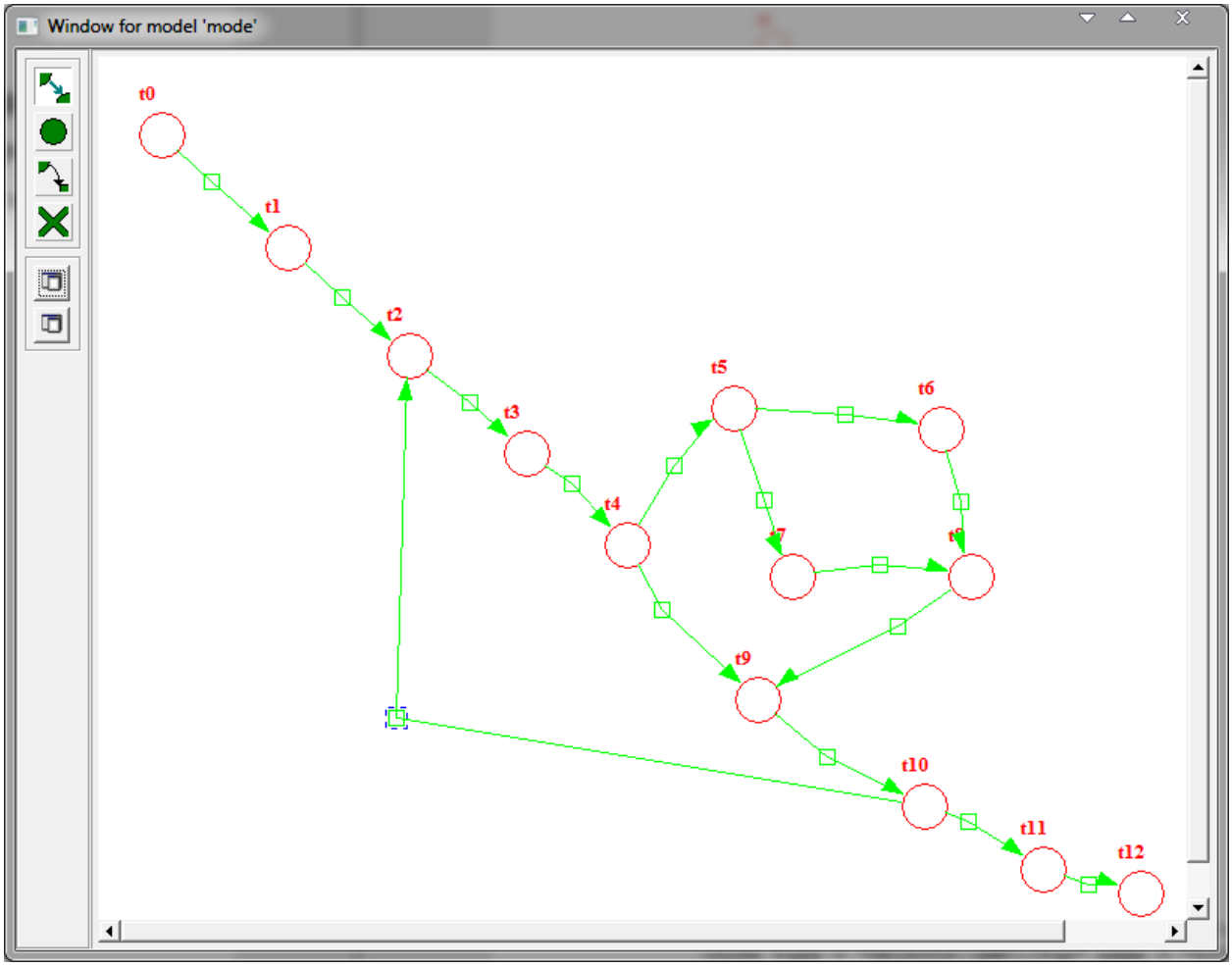

Рисунок 4 – ГНД

Описание модели:

```
<model type = "Objects::AMC::Model" name = "mode">
	<node type = "Objects::AMC::Top" name = "t0"></node>
	<node type = "Objects::AMC::Top" name = "t1"></node>
	<node type = "Objects::AMC::Top" name = "t2"></node>
	<node type = "Objects::AMC::Top" name = "t3"></node>
	<node type = "Objects::AMC::Top" name = "t4"></node>
	<node type = "Objects::AMC::Top" name = "t5"></node>
	<node type = "Objects::AMC::Top" name = "t6"></node>
	<node type = "Objects::AMC::Top" name = "t7"></node>
	<node type = "Objects::AMC::Top" name = "t8"></node>
	<node type = "Objects::AMC::Top" name = "t9"></node>
	<node type = "Objects::AMC::Top" name = "t10"></node>
	<node type = "Objects::AMC::Top" name = "t11"></node>
	<node type = "Objects::AMC::Top" name = "t12"></node>
```

```
      <link type = "Objects::AMC::Link" name = "t0-->t1" probability = "1.0"
intensity = "58.67" deviation = "0.0" source = "t0" dest = "t1"></link>
      <link type = "Objects::AMC::Link" name = "t1-->t2" probability = "1.0"
intensity = "0.0" deviation = "0.0" source = "t1" dest = "t2"></link>
      <link type = "Objects::AMC::Link" name = "t2-->t3" probability = "1.0"
intensity = "10.19" deviation = "0.0" source = "t2" dest = "t3"></link>
      <link type = "Objects::AMC::Link" name = "t3-->t4" probability = "1.0"
intensity = "67.62" deviation = "0.0" source = "t3" dest = "t4"></link>
      <link type = "Objects::AMC::Link" name = "t4-->t5" probability = "0.0"
intensity = "0.0" deviation = "0.0" source = "t4" dest = "t5"></link>
      <link type = "Objects::AMC::Link" name = "t5-->t6" probability = "0.0"
intensity = "0.0" deviation = "0.0" source = "t5" dest = "t6"></link>
      <link type = "Objects::AMC::Link" name = "t6-->t8" probability = "0.0"
intensity = "0.0" deviation = "0.0" source = "t6" dest = "t8"></link>
      <link type = "Objects::AMC::Link" name = "t5-->t7" probability = "0.0"
intensity = "0.0" deviation = "0.0" source = "t5" dest = "t7"></link>
      <link type = "Objects::AMC::Link" name = "t7-->t8" probability = "0.0"
intensity = "0.0" deviation = "0.0" source = "t7" dest = "t8"></link>
      <link type = "Objects::AMC::Link" name = "t8-->t9" probability = "0.0"
intensity = "0.0" deviation = "0.0" source = "t8" dest = "t9"></link>
      <link type = "Objects::AMC::Link" name = "t4-->t9" probability = "1.0"
intensity = "19.86" deviation = "0.0" source = "t4" dest = "t9"></link>
      <link type = "Objects::AMC::Link" name = "t9-->t10" probability = "1.0"
intensity = "17.86" deviation = "0.0" source = "t9" dest = "t10"></link>
      <link type = "Objects::AMC::Link" name = "t10-->t11" probability = "0.08"
intensity = "18.44" deviation = "0.0" source = "t10" dest = "t11"></link>
      <link type = "Objects::AMC::Link" name = "t11-->t12" probability = "1.0"
intensity = "2.51" deviation = "0.0" source = "t11" dest = "t12"></link>
      <link type = "Objects::AMC::Link" name = "t10-->t2" probability = "0.92"
intensity = "35.13" deviation = "0.0" source = "t10" dest = "t2"></link>
</model>
```

Результаты:

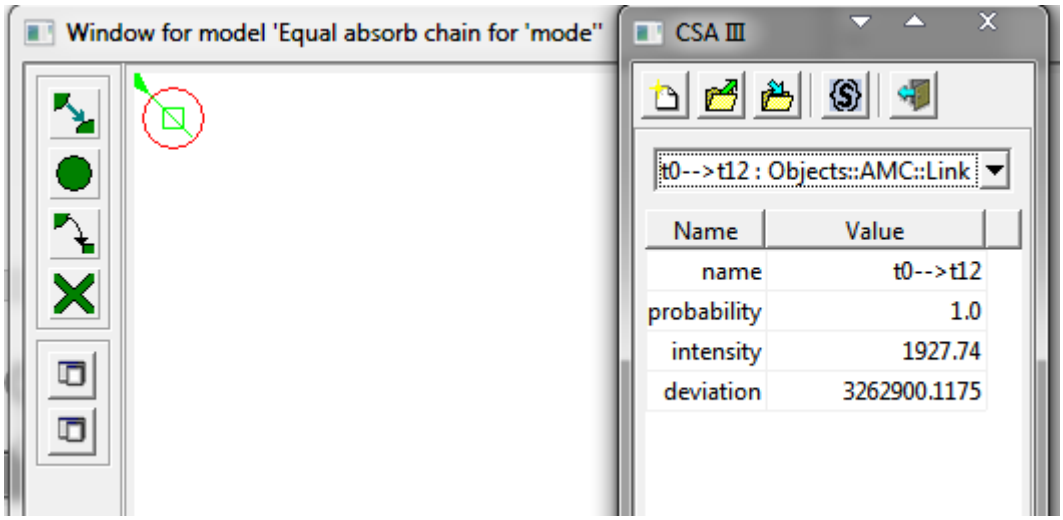

Рисунок 5 – Результаты

Выводы.

При выполнении лабораторной работы была построена операционная графовая модель заданной программы, нагрузочные параметры которой были оценены с помощью профилировщика Sampler и методом эквивалентных преобразований с помощью пакета CSA III были вычислены математическое ожидание и дисперсия времени выполнения как для всей программы, так и для заданного фрагмента. Результаты сравнения этих характеристик с полученными в работе 3 согласуются.