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 _____ Пискунов Я.А.

Преподаватель _____ Кирьянчиков В.А.

Санкт-Петербург
2020

Цель работы.

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

Формулировка задания.

2.1. 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.

2.2. 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 текст программы с пронумерованными строками и расставленными точками измерений представлен в приложении А.

Теперь создадим граф управления программой. Он представлен на рис. 1.

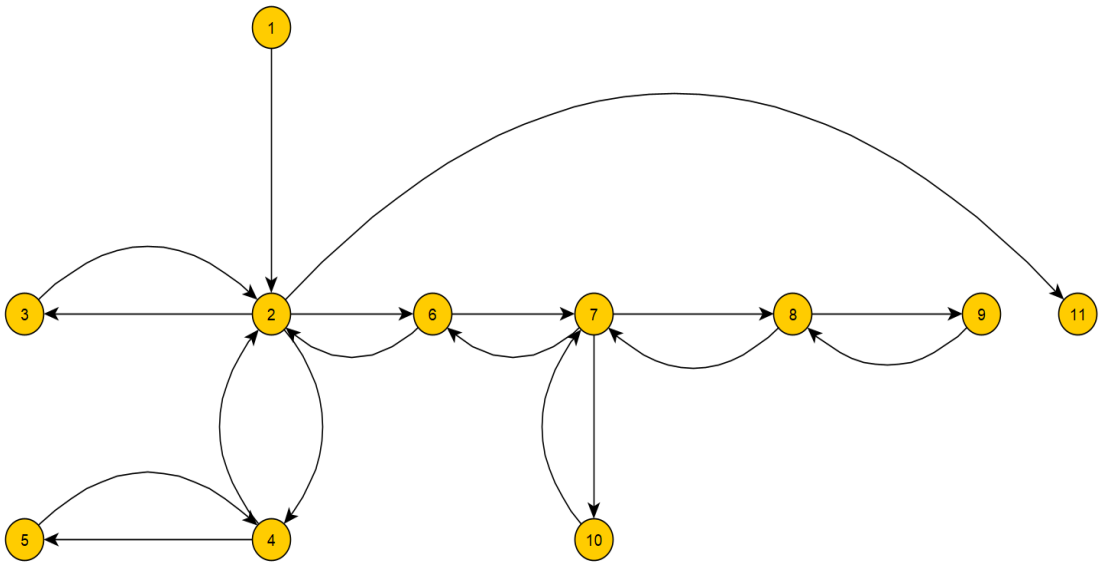

Рисунок 1 – Граф управления программой

Теперь, имея готовую для профилирования программу, произведем измерения с помощью Sampler. Результаты профилирования представлены на листинге 1.

Листниг 1. Результаты профилирования программы

```
-------------------------------------------------------------------------
  NN           €¬п ®Ўа Ў®в --®J® д ©«
-------------------------------------------------------------------------
  1.   MAIN1.CPP
-------------------------------------------------------------------------


  ' Ў«Ёж  б аЃ§г«мв в ¬Ё Ё§¬ЃаЃ-Ё© ( ЁбЇ®«м§гЃвбп 10 Ё§ 416 § ЇЁбЃ© )

-------------------------------------------------------------------------
€бе.Ц®§. ЦаЁЃ¬.Ц®§.  ЋЎйЃЃ ўаЃ¬п(¬Єб)  Љ®«-ў® Їа®е. 'аЃ¤-ЃЃ ўаЃ¬п(¬Єб)
-------------------------------------------------------------------------
 1 :  71  1 :  73               0.84            1               0.84
-------------------------------------------------------------------------
 1 :  73  1 :  75               1.68            3               0.56
-------------------------------------------------------------------------
 1 :  75  1 :  73               1.68            2               0.84
 1 :  75  1 :  77               0.84            1               0.84
-------------------------------------------------------------------------
 1 :  77  1 :  79               0.00            1               0.00
-------------------------------------------------------------------------
 1 :  79  1 :  89               0.00            1               0.00
-------------------------------------------------------------------------
 1 :  89  1 :  91               3.35            1               3.35
-------------------------------------------------------------------------
 1 :  91  1 : 104               2.51            1               2.51
-------------------------------------------------------------------------
 1 : 104  1 : 122               0.84            1               0.84
-------------------------------------------------------------------------
 1 : 120  1 :  71               0.84            1               0.84
-------------------------------------------------------------------------
```

Как можно заметить, строк в листинге значительно меньше, чем было выставлено точек измерений. Это, по всей видимости, связано с тем, что некоторые участки программы были пройдены быстрее 1 мкс и «склеились» в один. Операционная графовая модель далее строилась по фактическим результатам профилирования и не соответствует построенному ранее графу управления.

Сгруппируем полученные результаты для последующего анализа и построения операционной графовой модели. Это представлено в табл. 1.

Таблица 1. Результаты профилирования

	Номера строк	**Количество проходов**
L_1 = 0.84 мкс	120:71	1
L_2 = 1.68 мкс	71:73, 75:73	1; 2
L_3 = 0.56 мкс	73:75	3
L_4 = 7.54 мкс	75:77, 77:79, 79:80, 89:91, 91:104, 104:122	1; 1; 1; 1; 1; 1

На основании полученных в таблице результатов построим графовую модель. Она представлена на рис. 2. Как и отмечалось ранее, она не соответствует составленному ранее графу управления.

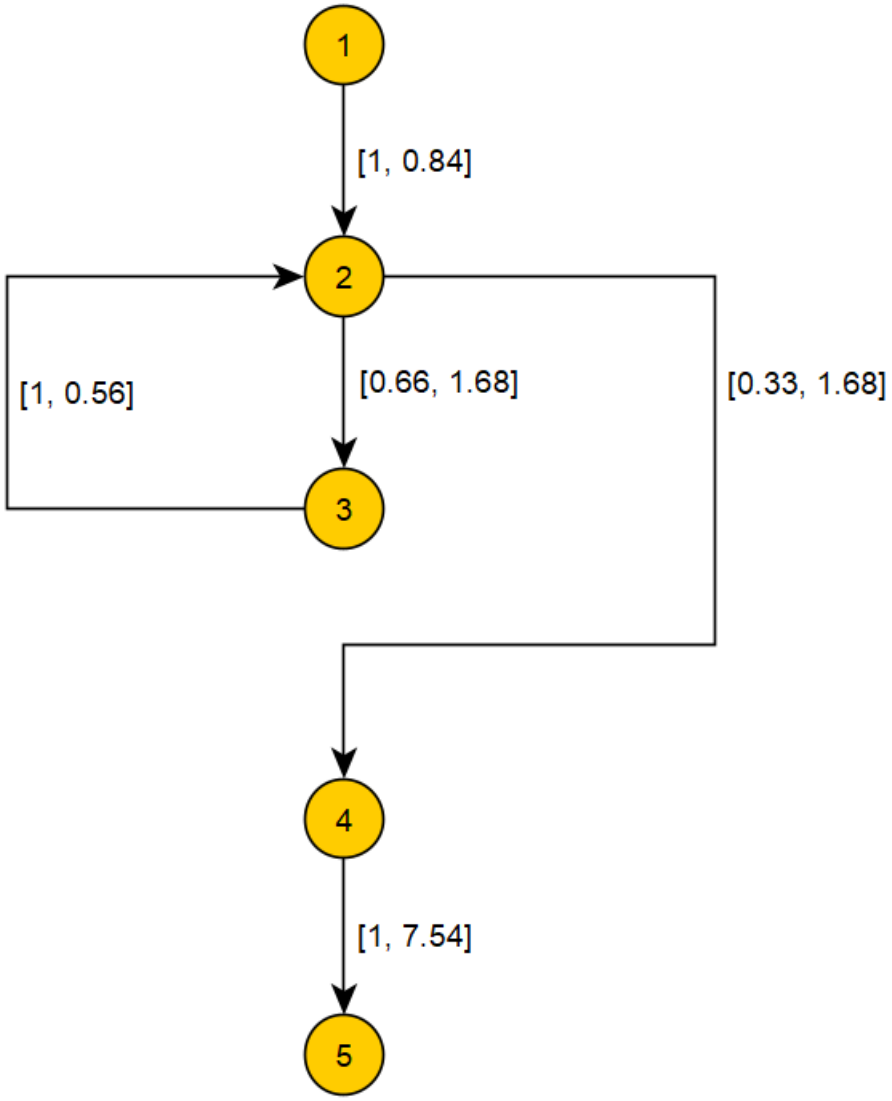

Рисунок 2 – Графовая модель

Далее произведем расчет с помощью CSA III. В приложении Б представлено XML-описание графовой модели программы. Полученное в программе изображение графа представлено на рис. 3, а результаты ее работы – на рис. 4.

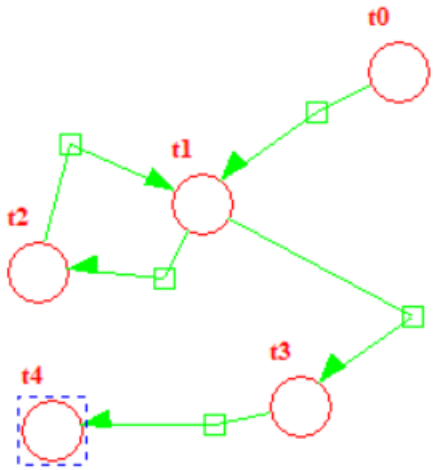

Рисунок 3 – Графовая модель в программе

t0-->t4 : Objects::AMC::Link

Name	Value
name	t0-->t4
probability	0.970588235294118
intensity	14.4082352941176
deviation	28.6471972318339

Рисунок 4 – Результаты анализа

В предыдущей работе было получено значение intensity примерно в два раза меньше. Однако, при таких маленьких значениях велико влияние скачков, вызванных использованием виртуальной машины, о котором говорилось в предыдущей работе, что также подтверждается весьма большим по сравнению с самим результатом значением дисперсии.

Выводы.

В ходе выполнения данной 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. Результаты сравнения этих характеристик с полученными в работе 3 согласуются в пределах дисперсии.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
1  #include "Sampler.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #define RMAX 3
6  #define CMAX 3
7
8  int n;
9  char yesno;
10  int oshibka;
11
12 double* y;
13 double* coef;
14 double** a;
15
16 double det;
17
18 void get_data(double** a, double* y){
19     a[0][0] = 1;
20     a[0][1] = -43;
21     a[0][2] = 19;
22     y[0] = 81;
23     a[1][0] = 145;
24     a[1][1] = -134;
25     a[1][2] = 99;
26     y[1] = 12;
27     a[2][0] = 325;
28     a[2][1] = 991;
29     a[2][2] = -199;
30     y[2] = 213;
31 }
32
33
34
35 void write_data(){
36     int i;
37     for (i=0; i<n; i++)
38         printf("%lf ", coef[i]);
```

```
39     printf("\n");
40 }
41
42 double deter(double** a){
43     return(a[0][0]*(a[1][1]*a[2][2]-a[2][1]*a[1][2])
44                 -a[0][1]*(a[1][0]*a[2][2]-a[2][0]*a[1][2])
45                 +a[0][2]*(a[1][0]*a[2][1]-a[2][0]*a[1][1]));
46
47 }
48
49 void setup(double** b, double* coef, int j){
50     int i;
51     SAMPLE;
52     for(i=0; i<n; i++){
53        SAMPLE;
54        b[i][j] = y[i];
55        SAMPLE;
56        if (j>0){
57         SAMPLE;
58           b[i][j-1] = a[i][j-1];
59         SAMPLE;
60     }
61    SAMPLE;
62  }
63  SAMPLE;
64  coef[j] = deter(b) / det;
65 }
66
67 void solve(){
68     double** b;
69     b = (double**)malloc(RMAX*sizeof(double*));
70     int i,j;
71     SAMPLE;
72     for (i=0; i<RMAX; i++){
73         SAMPLE;
74         b[i] = (double*)malloc(CMAX*sizeof(double));
75         SAMPLE;
76     }
77     SAMPLE;
78     oshibka = 1;
79     SAMPLE;
```

```
80      for(i=0; i<n; i++){
81          SAMPLE;
82          for(j=0; j<n; j++){
83              SAMPLE;
84              b[i][j] = a[i][j];
85              SAMPLE;
86          }
87      SAMPLE;
88      }
89      SAMPLE;
90      det = deter(b);
91      SAMPLE;
92      if (det==0){
93          oshibka = 0;
94          printf("ERROR: matrix is singular.");
95      } else {
96         SAMPLE;
97         setup(b, coef, 0);
98       SAMPLE;
99         setup(b, coef, 1);
100        SAMPLE;
101        setup(b, coef, 2);
102        SAMPLE;
103     }
104     SAMPLE;
105 }
106
107 int main()
108 {
109     y = (double*)malloc(CMAX*sizeof(double));
110     coef = (double*)malloc(CMAX*sizeof(double));
111
112     a = (double**)malloc(RMAX*sizeof(double*));
113     int i;
114     for (i=0; i<RMAX; i++)
115         a[i] = (double*)malloc(CMAX*sizeof(double));
116
117     printf("\n");
118     printf("Simultaneous soulution by Cramers rule");
119     get_data(a, y);
120     SAMPLE;
```

```
121    solve();
122    SAMPLE;
123    if (oshibka == 0)
124        write_data();
125    printf("\n");
126    return 0;
127 }
```

ПРИЛОЖЕНИЕ Б

XML-ОПИСАНИЕ ГРАФА

```
<model type = "Objects::AMC::Model" name = "lab4">
	<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>
	<link type = "Objects::AMC::Link" name = "t0-->t1" probability = "1.0"
intensity = "0.84" deviation = "0.0" source = "t0" dest = "t1"></link>
	<link type = "Objects::AMC::Link" name = "t1-->t2" probability = "0.66"
intensity = "1.68" deviation = "0.0" source = "t1" dest = "t2"></link>
	<link type = "Objects::AMC::Link" name = "t1-->t3" probability = "0.33"
intensity = "1.68" deviation = "0.0" source = "t1" dest = "t3"></link>
	<link type = "Objects::AMC::Link" name = "t2-->t1" probability = "1.0"
intensity = "0.56" deviation = "0.0" source = "t2" dest = "t1"></link>
	<link type = "Objects::AMC::Link" name = "t3-->t4" probability = "1.0"
intensity = "7.54" deviation = "0.0" source = "t3" dest = "t4"></link>
</model>
```