

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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Построение операционной графовой модели программы
(ОГМП) и расчет характеристик эффективности ее выполнения
методом эквивалентных преобразований»

Студентка гр. 8304

Сани З.Б.

Преподаватель

Кирияничков В. А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

Для задания из лабораторных работ 1-3 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.

Полученную ОГМП, представить в виде графа с нагруженными дугами, у которого в качестве параметров, характеризующих потребление ресурсов на дуге ij , использовать тройку $\{P_{ij}, M_{ij}, D_{ij}\}$, где:

P_{ij} - вероятность выполнения процесса для дуги ij ;

M_{ij} - мат. ожидание потребления ресурса процессом для дуги ij ;

D_{ij} - 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.

1. Текст программы (исходный).

Исходный код программы представлен в приложении А.

2. Граф управления программы.

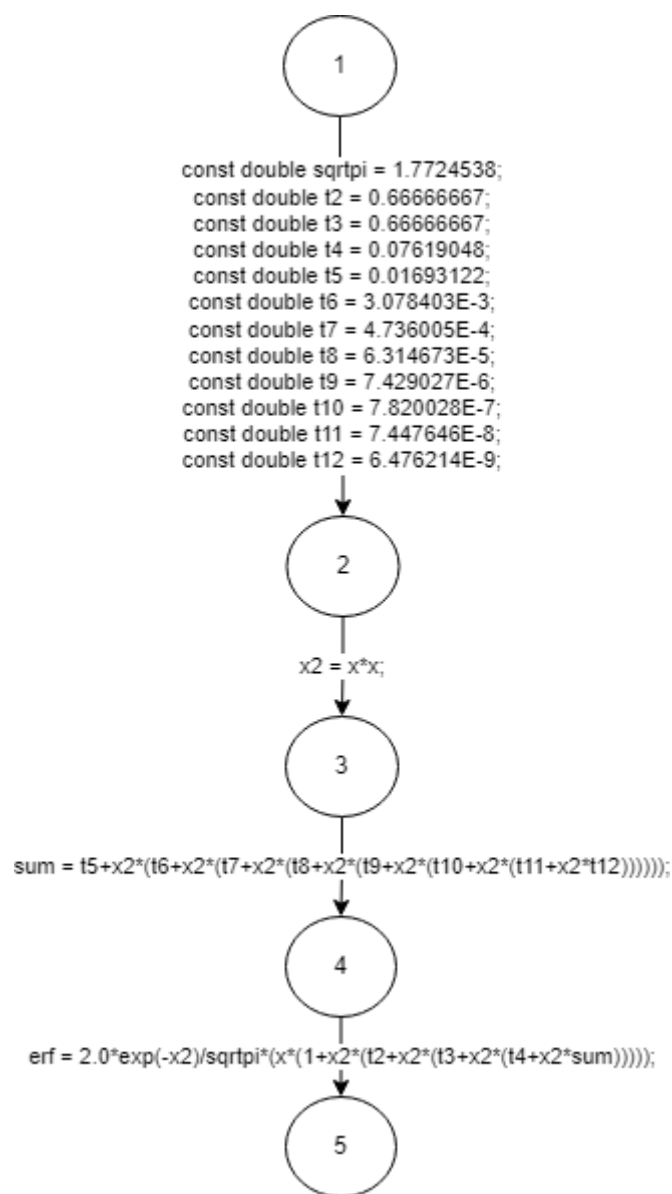


Рисунок 1 – Граф управления программы

3. Профилирование.

Код программы для профилирования, разделенной на функциональные участки, представлен в приложении Б.

4. Результаты профилирования.

Результаты профилирования из лабораторной работы №3 представлены на рисунке 2.

исх	прием	общее время	кол-во проходов	среднее время
8	11	65.609	1	65.609
11	24	32.294	1	32.294
24	26	7.294	1	7.294
26	28	9.568	1	9.568
28	30	7693.787	1	7693.787

Рисунок 1 – Результаты профилирования

5. Расчет вероятностей и затрат ресурсов для дуг управляющего графа.

Результаты расчета представлены в таблице 1.

Таблица 1 – Расчет вероятностей и затрат ресурсов для дуг.

	Номера строк	Количество проходов	Вероятность	Затраты ресурсов (среднее время), мкс
L_{1-2}	11:24	1	1	32.294
L_{2-3}	24:26	1	1	7.294
L_{3-4}	26:28	1	1	9.568
L_{4-5}	28:30	1	1	7693.787

6. Операционная графовая модель программы.

Операционная графовая модель программы представлена на рисунке 3.

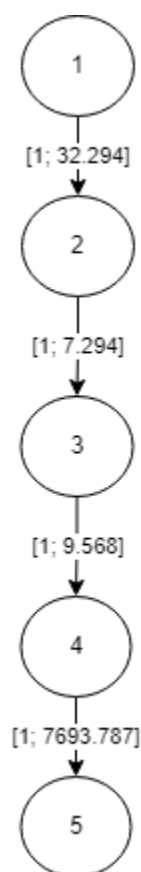


Рисунок 2 - Операционная модель

ГНД

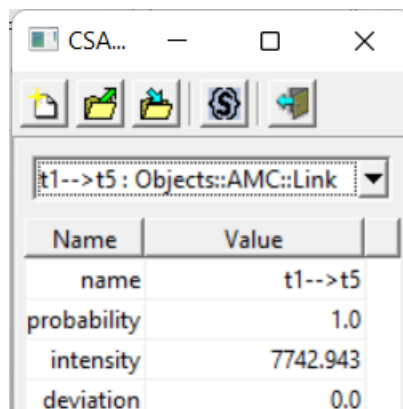


7. Описание модели model.xml.

Описание модели представлено в приложении В.

8. Результаты.

Результаты работы программы представлены на рисунке 4.



The screenshot shows a window titled 'CSA...' with a toolbar and a table. The table has two columns: 'Name' and 'Value'. The data rows are as follows:

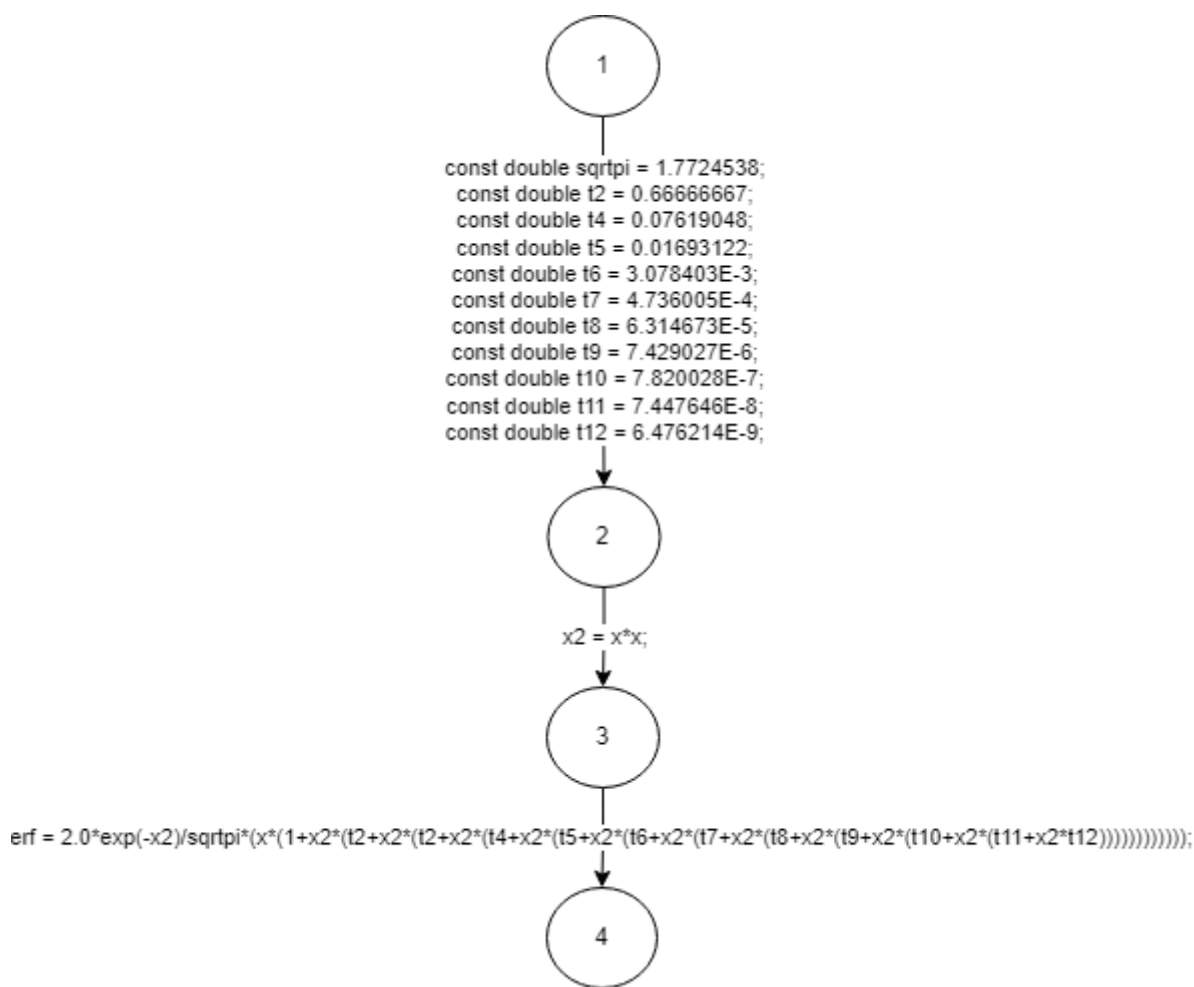
Name	Value
name	t1-->t5
probability	1.0
intensity	7742.943
deviation	0.0

Рисунок 3 - Результат работы программы

Согласно расчётам программы, среднее время выполнения составляет 7742,943 мкс. В пункте 4 данного отчёта приведен результат профилирования программы с использованием SAMPLER, где суммарное время выполнения составило 7808,552 мкс. В итоге, разница между результатами составляет менее 0,9 %.

Для модифицированной программы:

9. Граф управления программы.



10.Профилирование.

Код программы для профилирования, разделенной на функциональные участки, представлен в приложении Г.

11.Результаты профилирования.

Результаты профилирования из лабораторной работы №3 представлены на рисунке 5.

исх	прием	общее время	кол-во проходов	среднее время
8	11	6.399	1	6.399
11	23	25.566	1	25.566
23	25	7.270	1	7.270
25	27	6690.805	1	6690.805

Рисунок 5 – Результаты профилирования

12.Расчет вероятностей и затрат ресурсов для дуг управляющего графа.

Результаты расчета представлены в таблице 2.

Таблица 2 – Расчет вероятностей и затрат ресурсов для дуг.

	Номера строк	Количество проходов	Вероятность	Затраты ресурсов (среднее время), мкс
L_{1-2}	11:23	1	1	25.566
L_{2-3}	23:25	1	1	7.270
L_{3-4}	25:27	1	1	6690.805

13.Операционная графовая модель программы.

Операционная графовая модель программы представлена на рисунке 6.

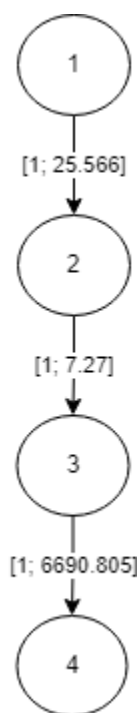
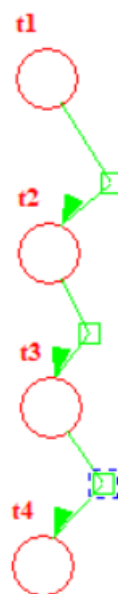


Рисунок 6 - Операционная модель

ГНД



14.Описание модели model_mod.xml.

Описание модели представлено в приложении Д.

15.Результаты.

Результаты работы программы представлены на рисунке 7.

Name	Value
name	t1-->t4
probability	1.0
intensity	6723.641
deviation	0.0

Рисунок 7 - Результат работы программы

Согласно расчётам программы, среднее время выполнения составляет 6723,641 мкс. В пункте 11 данного отчёта приведен результат профилирования программы с использованием SAMPLER, где суммарное время выполнения составило 6730,04 мкс. В итоге, разница между результатами составляет менее 0,1 %.

Выводы.

В ходе выполнения 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 фрагментов программы. Результаты сравнения этих характеристик с полученными в ходе выполнения лабораторной работы №3 согласуются.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```

1. #include <stdio.h>
2. #include <math.h>
3.
4. double x,er,ec;
5.
6. double erf(double x){
7.     const double sqrtpi = 1.7724538;
8.     const double t2      = 0.66666667;
9.     const double t3      = 0.66666667;
10.    const double  t4      = 0.07619048;
11.    const double  t5      = 0.01693122;
12.    const double  t6      = 3.078403E-3;
13.    const double  t7      = 4.736005E-4;
14.    const double  t8      = 6.314673E-5;
15.    const double  t9      = 7.429027E-6;
16.    const double  t10     = 7.820028E-7;
17.    const double  t11     = 7.447646E-8;
18.    const double  t12     = 6.476214E-9;
19.
20.    double x2,sum, erf;
21.
22.    x2 = x*x;
23.
24.    sum =
t5+x2*(t6+x2*(t7+x2*(t8+x2*(t9+x2*(t10+x2*(t11+x2*t12))))));
25.    erf = 2.0*exp(-
x2)/sqrtpi*(x*(1+x2*(t2+x2*(t3+x2*(t4+x2*sum)))));
26.    return erf;
27. }
28. double erfc(double x){
29.     const double sqrtpi = 1.7724538;
30.
31.     double x2,v,sum, erfc;
32.     x2 = x*x;
33.     v = 1.0/(2.0*x2);
34.     sum = v/(1+8*v/(1+9*v/(1+10*v/(1+11*v/(1+12*v)))));
35.     sum = v/(1+3*v/(1+4*v/(1+5*v/(1+6*v/(1+7*sum)))));
36.     erfc = 1.0/(exp(x2)*x*sqrtpi*(1+v/(1+2*sum)));
37.     return erfc;
38. }
39.
40. int main()
41. {
42.     x = 1.00;
43.
44.     if (x == 0.0){
45.         er = 0.0;
46.         ec = 1.0;
47.     }
48.     else{
49.         if (x < 1.5){

```

```
50.      er = erf(x);
51.      ec = 1.0 - er;
52.      } else {
53.      ec = erfc(x);
54.      er = 1.0 - ec;
55.      }
56.  }
57.
58.  return 0;
59. }
```

ПРИЛОЖЕНИЕ Б.

КОД ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУ

```
1. #include <stdio.h>
2. #include <math.h>
3. #include "sampler.h"
4.
5. double x,er,ec;
6.
7. double erf(double x){
8.     SAMPLE;
9.     double x2,sum, erf;
10.
11.     SAMPLE;
12.     const double sqrtpi = 1.7724538;
13.     const double t2      = 0.66666667;
14.     const double t3      = 0.66666667;
15.     const double t4      = 0.07619048;
16.     const double t5      = 0.01693122;
17.     const double t6      = 3.078403E-3;
18.     const double t7      = 4.736005E-4;
19.     const double t8      = 6.314673E-5;
20.     const double t9      = 7.429027E-6;
21.     const double t10     = 7.820028E-7;
22.     const double t11     = 7.447646E-8;
23.     const double t12     = 6.476214E-9;
24.     SAMPLE;
25.     x2 = x*x;
26.     SAMPLE;
27.
28.     sum =
t5+x2*(t6+x2*(t7+x2*(t8+x2*(t9+x2*(t10+x2*(t11+x2*t12))))));
29.     erf = 2.0*exp(-
x2)/sqrtpi*(x*(1+x2*(t2+x2*(t3+x2*(t4+x2*sum)))));
30.     SAMPLE;
31.     return erf;
32. }
33.
34. double erfc(double x){
35.     SAMPLE;
36.     double x2,v,sum, erfc;
37.     SAMPLE;
38.     const double sqrtpi = 1.7724538;
39.
40.     SAMPLE;
41.     x2 = x*x;
42.     SAMPLE;
43.     v = 1.0/(2.0*x2);
44.     SAMPLE;
45.     sum = v/(1+8*v/(1+9*v/(1+10*v/(1+11*v/(1+12*v)))));
46.     SAMPLE;
```

```

47.     sum = v/(1+3*v/(1+4*v/(1+5*v/(1+6*v/(1+7*sum)))));
48.     SAMPLE;
49.     erfc = 1.0/(exp(x2)*x*sqrtpi*(1+v/(1+2*sum)));
50.     SAMPLE;
51.     return erfc;
52. }
53.
54. int main(int argc, char **argv)
55. {
56.     sampler_init(&argc, argv);
57.
58.     x = 1.00;
59.
60.     if (x == 0.0){
61.         er = 0.0;
62.         ec = 1.0;
63.     }
64.     else{
65.         if (x < 1.5){
66.             er = erf(x);
67.             ec = 1.0 - er;
68.         } else {
69.             ec = erfc(x);
70.             er = 1.0 - ec;
71.         }
72.     }
73.
74.     return 0;
75. }

```

ПРИЛОЖЕНИЕ В.

ОПИСАНИЕ МОДЕЛИ .XML

```
<model type = "Objects::AMC::Model" name = "model">
  <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>
  <link type = "Objects::AMC::Link" name = "t1-->t2" probability
= "1.0" intensity = "32.294" deviation = "0.0" source = "t1" dest =
"t2"></link>
  <link type = "Objects::AMC::Link" name = "t2-->t3" probability
= "1.0" intensity = "7.294" deviation = "0.0" source = "t2" dest =
"t3"></link>
  <link type = "Objects::AMC::Link" name = "t3-->t4" probability
= "1.0" intensity = "9.568" deviation = "0.0" source = "t3" dest =
"t4"></link>
  <link type = "Objects::AMC::Link" name = "t4-->t5" probability
= "1.0" intensity = "7693.787" deviation = "0.0" source = "t4" dest =
"t5"></link>
</model>
```


ПРИЛОЖЕНИЕ Г.

КОД МОДИФИЦИРОВАННОЙ ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУ

```
1. #include <stdio.h>
2. #include <math.h>
3. #include "sampler.h"
4.
5. double x,er,ec;
6.
7. double erf(double x){
8.     SAMPLE;
9.     double x2, erf;
10.
11.     SAMPLE;
12.     const double sqrtpi = 1.7724538;
13.     const double t2      = 0.66666667;
14.     const double  t4      = 0.07619048;
15.     const double  t5      = 0.01693122;
16.     const double  t6      = 3.078403E-3;
17.     const double  t7      = 4.736005E-4;
18.     const double  t8      = 6.314673E-5;
19.     const double  t9      = 7.429027E-6;
20.     const double  t10     = 7.820028E-7;
21.     const double  t11     = 7.447646E-8;
22.     const double  t12     = 6.476214E-9;
23.     SAMPLE;
24.     x2 = x*x;
25.     SAMPLE;
26.     erf = 2.0*exp(-
x2)/sqrtpi*(x*(1+x2*(t2+x2*(t2+x2*(t4+x2*(t5+x2*(t6+x2*(t7+x2*(t8+x2*(
t9+x2*(t10+x2*(t11+x2*t12))))))))))));
27.     SAMPLE;
28.     return erf;
29. }
30.
31. double erfc(double x){
32.     SAMPLE;
33.     double  x2,v, erfc;
34.     SAMPLE;
35.     const double sqrtpi = 1.7724538;
36.
37.     SAMPLE;
38.     x2 = x*x;
39.     SAMPLE;
40.     v = 1.0/(2.0*x2);
41.     SAMPLE;
42.     erfc =
1.0/(exp(x2)*x*sqrtpi*(1+v/(1+2*v/(1+3*v/(1+4*v/(1+5*v/(1+6*v/(1+7*v/(
1+8*v/(1+9*v/(1+10*v/(1+11*v/(1+12*v))))))))))));
```

```

43.     SAMPLE;
44.     return erfc;
45. }
46.
47. int main(int argc, char **argv)
48. {
49.     sampler_init(&argc, argv);
50.
51.     x = 1.00;
52.
53.     if (x == 0.0){
54.         er = 0.0;
55.         ec = 1.0;
56.     }
57.     else{
58.         if (x < 1.5){
59.             er = erf(x);
60.             ec = 1.0 - er;
61.         } else {
62.             ec = erfc(x);
63.             er = 1.0 - ec;
64.         }
65.     }
66.
67.     return 0;
68. }

```

ПРИЛОЖЕНИЕ Д.

ОПИСАНИЕ МОДЕЛИ .XML

```
<model type = "Objects::AMC::Model" name = "model">
  <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 = "t1-->t2" probability
= "1.0" intensity = "25.566" deviation = "0.0" source = "t1" dest =
"t2"></link>
  <link type = "Objects::AMC::Link" name = "t2-->t3" probability
= "1.0" intensity = "7.27" deviation = "0.0" source = "t2" dest =
"t3"></link>
  <link type = "Objects::AMC::Link" name = "t3-->t4" probability
= "1.0" intensity = "6690.805" deviation = "0.0" source = "t3" dest =
"t4"></link>
</model>
```