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

ОТЧЕТ

по практической работе №3
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. 8304 _____ Птухов Д.А.

Преподаватель _____ Кирьянчиков В.А.

Санкт-Петербург
2022

Цель работы.

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

Задание.

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.

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 $\{P_{ij}, M_{ij}, D_{ij}\}$, 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.

Построение операционной графовой модели программы.

Текст программы (исходный).

Исходный код программы представлен в приложении А.

Граф управления программы.

Некоторые функциональные блоки программы были объединены в один линейный, так как без данного упрощения размер графа увеличивается примерно в 2 раза. Граф управления программы, полученный в результате объединения линейных участков программы в одну дугу для минимизации размеров графа, представлен на рисунке 1.

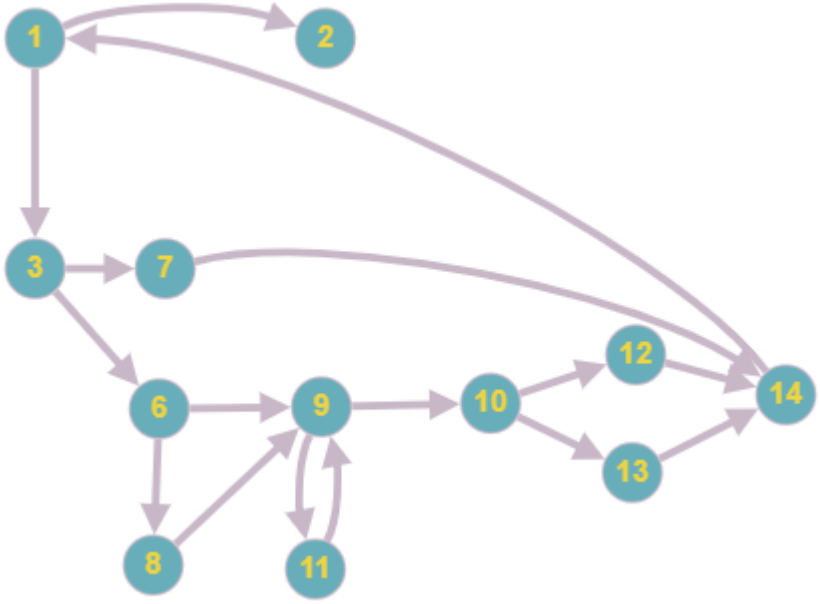

Рисунок 1 – Граф управления программы

Профилирование.

Код программы, подготовленной к профилированию, представлен в приложении Б.

Текст программы (подготовленной для профилирования)

Код программы, подготовленной для профилирования, представлен в приложении Б.

Результаты профилирования.

Результаты профилирования представлены на рисунке 2.

исх	прием	общее время	кол-во проходов	среднее время
23	26	70.000	1	70.000
26	35	2470.000	148	16.689
26	29	1090.000	149	7.315
35	41	2340.000	148	15.811
41	43	710.000	47	15.106
41	53	1950.000	101	19.307
43	51	2800.000	47	59.574
51	53	550.000	47	11.702
53	56	1050.000	148	7.095
56	58	2400.000	148	16.216
58	66	22460.000	505	44.475
66	58	2570.000	357	7.199
66	68	3420.000	148	23.108
68	72	2630.000	148	17.770
72	82	4300.000	122	35.246
72	74	810.000	26	31.154
82	86	4000.000	122	32.787
86	88	190.000	122	1.557
88	91	1320.000	148	8.919
91	26	2960.000	296	10.000
91	93	-10.000	1	-10.000
74	78	710.000	26	27.308
78	88	410.000	26	15.769
29	31	520.000	149	3.490
31	91	1550.000	149	10.403

Рисунок 2 – Результаты профилирования

Расчет вероятностей и затрат ресурсов для дуг управляющего графа.

Результаты расчета вероятностей и затрат ресурсов для дуг графа представлены в таблице 1.

Таблица 1 – Расчет вероятностей и затрат ресурсов

	Номера строк	Количество проходов	Вероятность
$L_{1-2} = 10.000$нс	23-26; 91-26; 91-93; 23-93(0)	1	0.0067
$L_{1-3} = 70.000$нс	23-26; 91-26; 91-93; 23-93(0)	297	0.9933
$L_{3-7} = 7.315$нс	26-29; 26-35;	149	0.5017
$L_{3-6} = 16.689$нс	26-29; 26-35;	148	0.4983
$L_{7-14} = 8.919$нс	88-91;	148	1
$L_{6-8} = 15.106$нс	41-43; 41-53;	47	0.3176
$L_{6-9} = 19.307$нс	41-43; 41-53;	101	0.6824
$L_{8-9} = 11.703$нс	41-43; 41-53;	47	1
$L_{9-11} = 16.216$нс	56-58; 56-68(0); 66-58; 66-68;	505	0.7734
$L_{11-9} = 44.475$нс	58-66;	505	1
$L_{9-10} = 7.199$нс	56-58; 56-68(0); 66-58; 66-68;	148	0.2276
$L_{10-12} = 31.154$нс	72-74; 72-82;	26	0.1757
$L_{10-13} = 35.246$нс	72-74; 72-82;	122	0.8243
$L_{12-14} = 27.308$нс	72-74; 74-78;	26	1
$L_{13-14} = 32.787$нс	72-82; 82-86;	122	1
$L_{14-1} = 10.000$нс	91-26;	296	1

Примечание: (0) после номеров строк обозначает отсутствие проходов между данными метками (следовательно на рисунке 2 их нет), однако при расчетах данные значения необходимо учитывать, поэтому они были включены в список.

Операционная графовая модель программы.

На основе данных из таблицы был модернизирован рисунок 1 и построена операционная графовая модель. Результат работы представлен на рисунке 3.

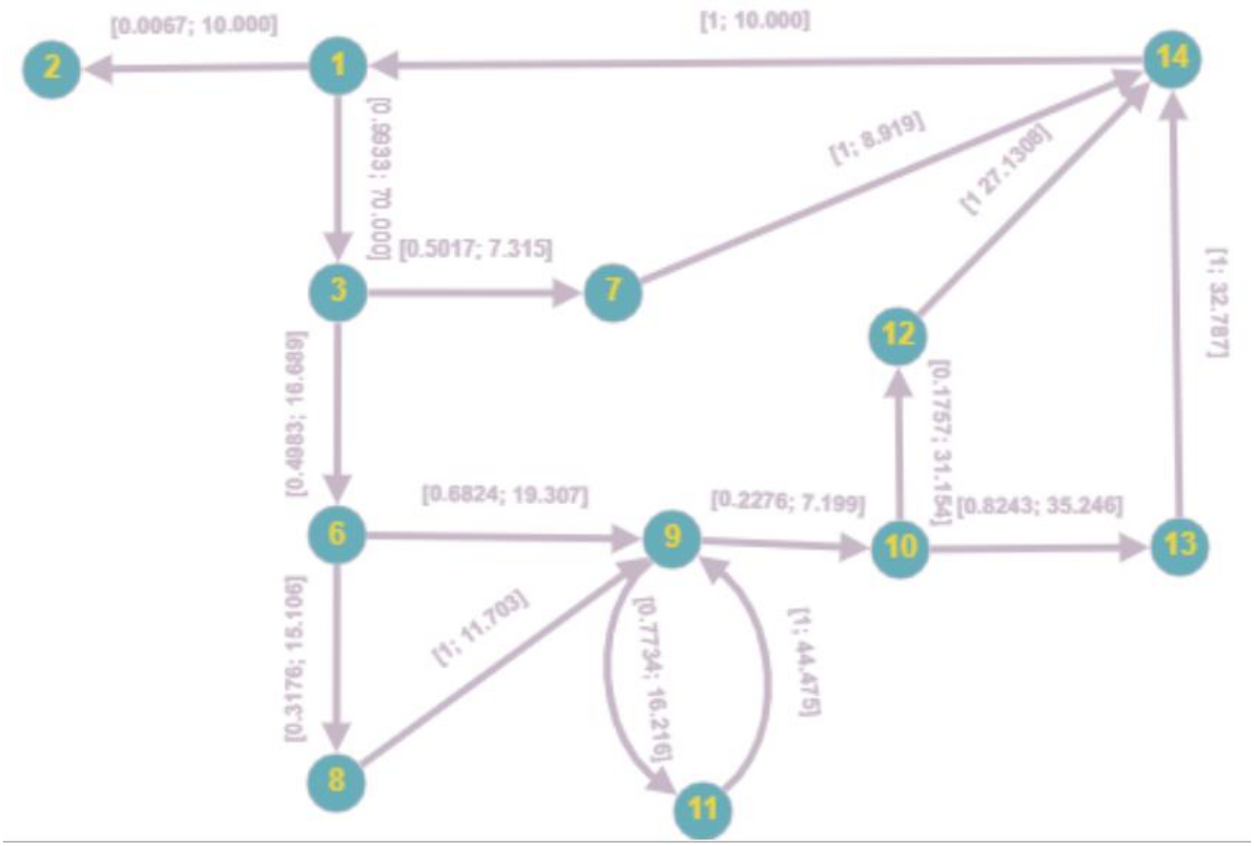

Рисунок 3 – Операционная графовая модель программы

Описание модели model.xml

Операционная графовая модель программы записана в виде xml-файла. Текст данного файла содержится в приложении В.

Результаты.

Результаты работы программы представлен на рисунке 4.

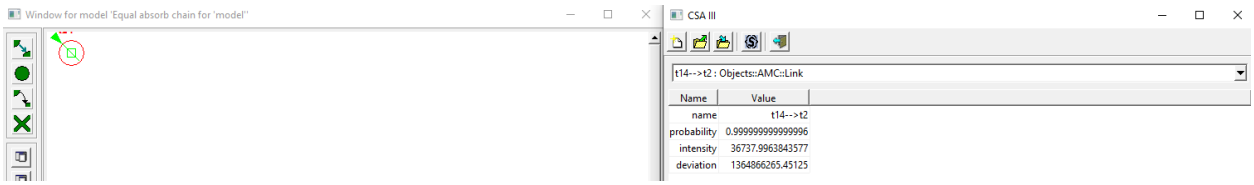

Рисунок 4 – Результат работы программы

Полученные результаты: среднее время выполнения программы – 36737.996нс.

Заключение

В ходе лабораторной работы была построена операционная графовая модель заданной программы, нагрузочные параметры которые были оценены с помощью профилировщика SAMPLER_v2 и методом эквивалентных преобразования с помощью пакета CSA III были вычислены мат ожидание и дисперсия времени выполнения программы.

ПРИЛОЖЕНИЕ А.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>


const int max = 200;

void swap(int* a, int* b)
{
    int c = *a;
    *a = *b;
    *b = c;
}

void sort(int* x, int n)
{
    int left[50], right[50], sp, i, j, mid, pivot;

    left[0] = 0;
    right[0] = n - 1;
    sp = 0;

    while (sp > -1)
    {
        if (left[sp] >= right[sp])
            --sp;
        else
        {
            i = left[sp];
            j = right[sp];
            pivot = x[j];
            mid = (i + j) / 2;

            if (j - i > 5) {
                if ((x[mid] < pivot && x[mid] > x[i]) ||
                    (x[mid] > pivot && x[mid] < x[i]))
                    swap(&(x[mid]), &(x[j]));

                else if ((x[i] < x[mid] && x[i] > pivot) ||
                         (x[i] > x[mid] && x[i] < pivot))
                    swap(&(x[i]), &(x[j]));
```

```
            }

            pivot = x[j];
            while (i < j) {
                while (x[i] < pivot)
                    ++i;
                --j;
                while (i < j && pivot < x[j])
                    --j;
                if (i < j)
                    swap(&(x[i]), &(x[j]));
            }

            j = right[sp];
            swap(&(x[i]), &(x[j]));
            if (i - left[sp] >= right[sp] - i) {
                left[sp + 1] = left[sp];
                right[sp + 1] = i - 1;
                left[sp] = i + 1;
            }
            else
            {
                left[sp + 1] = i + 1;
                right[sp + 1] = right[sp];
                right[sp] = i - 1;
            }
            ++sp;
        }
    }
}

int main() {
    int n = max;
    int* x = (int*)malloc(max*sizeof(int));
    srand(time(NULL));

    for (int i = 0; i < n; ++i)
        x[i] = rand() % 100;

    sort(x, n);

   // for (int i = 0; i < n; ++i)
   //     printf("%d ", x[i]);
```

```
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "sampler.h"

const int max = 200;

void swap(int* a, int* b)
{
    int c = *a;
    *a = *b;
    *b = c;
}

void sort(int x[max], int n)
{
    int left[50], right[50], sp, i, j, mid, pivot;

    left[0] = 0;
    right[0] = n - 1;
    sp = 0;

      SAMPLE;
    while (sp > -1)
    {
            SAMPLE;
        if (left[sp] >= right[sp])
            {
                  SAMPLE;
            --sp;
                  SAMPLE;
            }
        else
        {
                  SAMPLE;
            i = left[sp];
            j = right[sp];
            pivot = x[j];
            mid = (i + j) / 2;
```

```
      SAMPLE;
if (j - i > 5) {
            SAMPLE;
    if ((x[mid] < pivot && x[mid] > x[i]) ||
        (x[mid] > pivot && x[mid] < x[i]))
        swap(&(x[mid]), &(x[j]));

    else if ((x[i] < x[mid] && x[i] > pivot) ||
             (x[i] > x[mid] && x[i] < pivot))
        swap(&(x[i]), &(x[j]));
            SAMPLE;
}
      SAMPLE;

pivot = x[j];
      SAMPLE;
while (i < j) {
            SAMPLE;
    while (x[i] < pivot)
        ++i;
    --j;
    while (i < j && pivot < x[j])
        --j;
    if (i < j)
        swap(&(x[i]), &(x[j]));
            SAMPLE;
}
      SAMPLE;

j = right[sp];
swap(&(x[i]), &(x[j]));
      SAMPLE;
if (i - left[sp] >= right[sp] - i) {
            SAMPLE;
    left[sp + 1] = left[sp];
    right[sp + 1] = i - 1;
    left[sp] = i + 1;
            SAMPLE;
}
else
{
            SAMPLE;
    left[sp + 1] = i + 1;
```

```
                right[sp + 1] = right[sp];
                right[sp] = i - 1;
                        SAMPLE;
            }
                  SAMPLE;
            ++sp;
        }
            SAMPLE;
    }
      SAMPLE;
}

int main(int pargc, char **argv) {
      sampler_init(&pargc, argv);

    int n = max;
    int x[max];
    srand(time(NULL));

    for (int i = 0; i < n; ++i)
        x[i] = rand() % 100;

    sort(x, n);


   // for (int i = 0; i < n; ++i)
   //     printf("%d ", x[i]);

    return 0;
}
```

ПРИЛОЖЕНИЕ В.

```
<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>
      <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 = "t14"></node>
      <link type = "Objects::AMC::Link" name = "t1-->t2"    probability =
"0.0067" intensity = "10.000" deviation = "0.0" source = "t1"   dest =
"t2"></link>
      <link type = "Objects::AMC::Link" name = "t1-->t3"    probability =
"0.9933" intensity = "70.000" deviation = "0.0" source = "t1"   dest =
"t3"></link>
      <link type = "Objects::AMC::Link" name = "t3-->t4"    probability =
"0.4983" intensity = "16.689" deviation = "0.0" source = "t3"   dest =
"t4"></link>
      <link type = "Objects::AMC::Link" name = "t3-->t5"    probability =
"0.5017" intensity = "7.315"   deviation = "0.0" source = "t3"   dest =
"t5"></link>
      <link type = "Objects::AMC::Link" name = "t5-->t14"  probability = "1.0"
intensity = "8.919"  deviation = "0.0" source = "t5"  dest = "t14"></link>
      <link type = "Objects::AMC::Link" name = "t4-->t6"    probability =
"0.3176" intensity = "15.106" deviation = "0.0" source = "t4"   dest =
"t6"></link>
      <link type = "Objects::AMC::Link" name = "t4-->t7"    probability =
"0.6824" intensity = "19.307" deviation = "0.0" source = "t4"   dest =
"t7"></link>
      <link type = "Objects::AMC::Link" name = "t6-->t7"   probability = "1.0"
intensity = "11.703" deviation = "0.0" source = "t6"  dest = "t7"></link>
      <link type = "Objects::AMC::Link" name = "t7-->t9"    probability =
"0.7743" intensity = "16.216" deviation = "0.0" source = "t7"   dest =
"t9"></link>
      <link type = "Objects::AMC::Link" name = "t9-->t7"   probability = "1.0"
intensity = "44.475" deviation = "0.0" source = "t9"  dest = "t7"></link>
```

<link type = "Objects::AMC::Link" name = "t7-->t8" probability = "0.2257" intensity = "7.199" deviation = "0.0" source = "t7" dest = "t8"></link>
<link type = "Objects::AMC::Link" name = "t8-->t10" probability = "0.1757" intensity = "31.154" deviation = "0.0" source = "t8" dest = "t10"></link>
<link type = "Objects::AMC::Link" name = "t8-->t11" probability = "0.8243" intensity = "35.246" deviation = "0.0" source = "t8" dest = "t11"></link>
<link type = "Objects::AMC::Link" name = "t10-->t14" probability = "1.0" intensity = "27.308" deviation = "0.0" source = "t10" dest = "t14"></link>
<link type = "Objects::AMC::Link" name = "t11-->t14" probability = "1.0" intensity = "32.787" deviation = "0.0" source = "t11" dest = "t14"></link>
<link type = "Objects::AMC::Link" name = "t14-->t1" probability = "1.0" intensity = "10.000" deviation = "0.0" source = "t14" dest = "t1"></link>
</model>