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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Качество и метрология программного обеспечения»

Тема: Анализ структурной сложности графовых моделей программ

Студент гр. 7304 _____ Есиков О.И.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

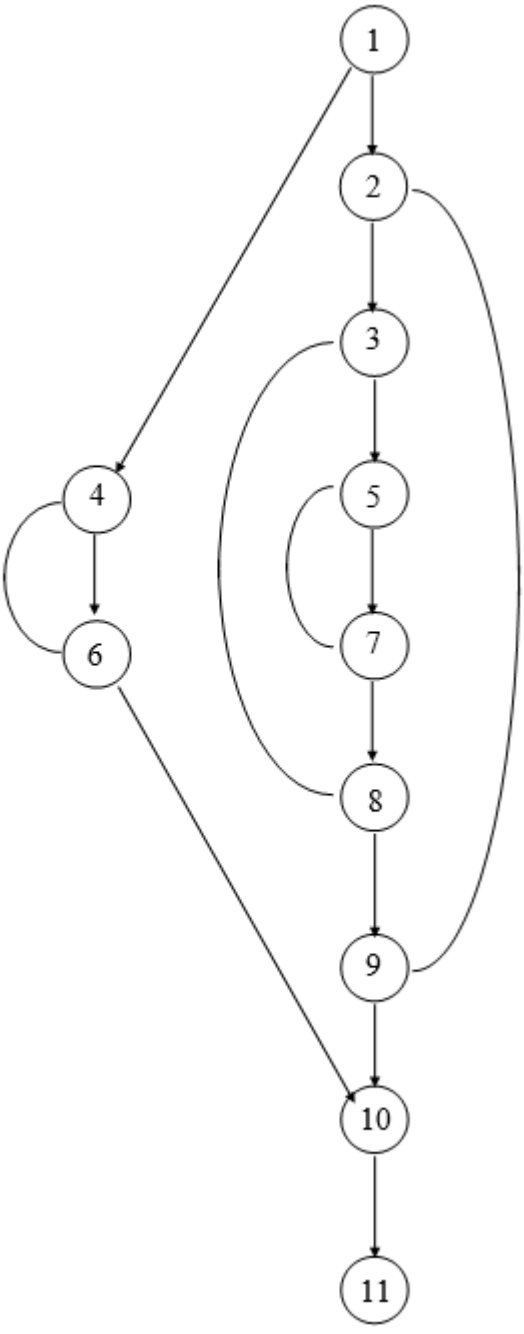
2021

Цель работы.

Изучить структурную сложность графовых моделей программ и метрики для её оценки.

Ход выполнения.

Был выбран 5 вариант задания. Структура управляющего графа для этого варианта представлена на рисунке 1.



5

Рисунок 1 – Управляющий граф для 5 варианта

Был выполнен расчёт структурной сложности этой программы по первому критерию – минимальное покрытие вершин и дуг управления.

Минимальное количество маршрутов – 2:

- **1** – 4 – **6** – 4 – **6** – 10 – 11
- **1** – 2 – 3 – 5 – **7** – **8** – **9** – 2 – 3 – 5 – **7** – **8** – 3 – 5 – **7** – 5 – **7** – **8** – **9** – 10 – 11

В итоге сложность программы $S_1 = 3 + 10 = \underline{13}$.

Был выполнен расчёт структурной сложности этой программы по второму критерию – каждый линейно-независимый цикл и ациклический участок программы.

Полное число вершин N = 11

Количество связывающих дуг Y = 15

Число связанных компонент P = 1 (максимально связанный граф получается при добавлении дуги 11 – 1)

Цикломатическое число $Z = Y – N + 2 \cdot P = 6$

Необходимые 6 линейно-независимых маршрутов:

- 4 – **6**
- 5 – **7**
- 3 – 5 – **7** – **8**
- 2 – 3 – 5 – **7** – **8** – **9**
- **1** – 4 – **6** – 10 – 11
- **1** – 2 – 3 – 5 – **7** – **8** – **9** – 10 – 11

В итоге сложность программы $S_2 = 1 + 1 + 2 + 3 + 2 + 4 = \underline{13}$.

Для выполнения автоматического расчёта с помощью программы ways.exe был подготовлен специальный файл *nodes1.txt*, задающий граф программы. Этот файл представлен в Приложении А. Результат работы программы ways.exe представлен на рисунках 2 и 3.

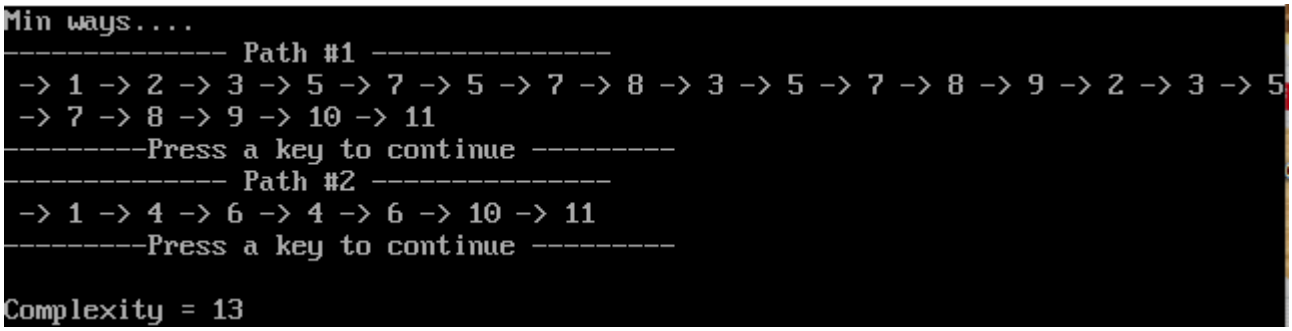


Рисунок 2 – Расчёт сложности по первому критерию программой ways.exe

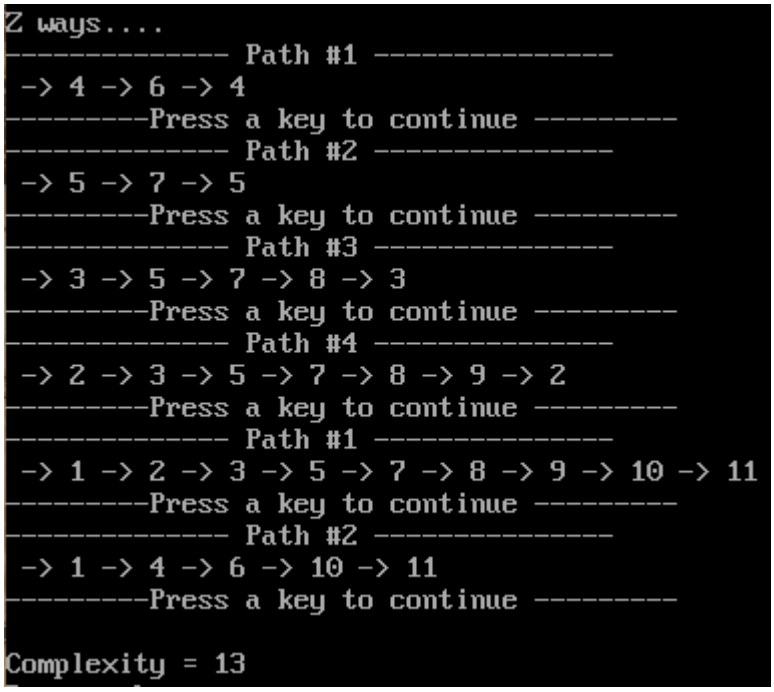


Рисунок 3 – Расчёт сложности по второму критерию программой ways.exe

С помощью команды

```
gcc -fdump-tree-cfg BubbleSort.c
```

был получен файл *BubbleSort.c.011t.cfg* с графом потока управления для каждой функции в файле. Файл *BubbleSort.c.011t.cfg* представлен в приложении Б. С помощью файла *BubbleSort.c.011t.cfg* был получен граф потока управления, представленный на рисунке 4, с указанием кода, из которого были получены вершины графа.

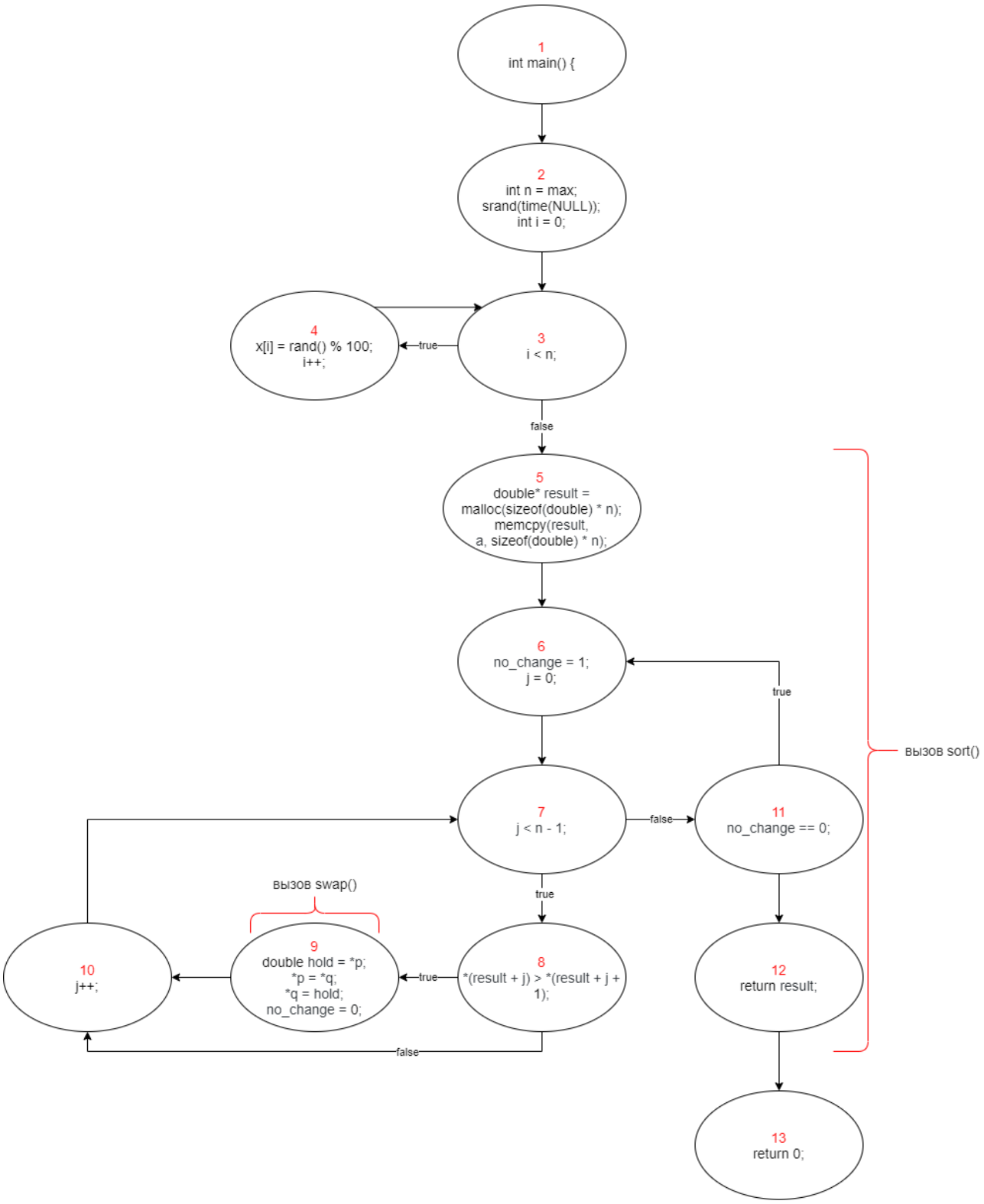


Рисунок 4 – Управляющий граф для программы из лабораторной работы №1

Но для такого графа не может быть произведён расчёт с помощью программы ways.exe, поэтому граф был преобразован. Результат представлен на рисунке 5.

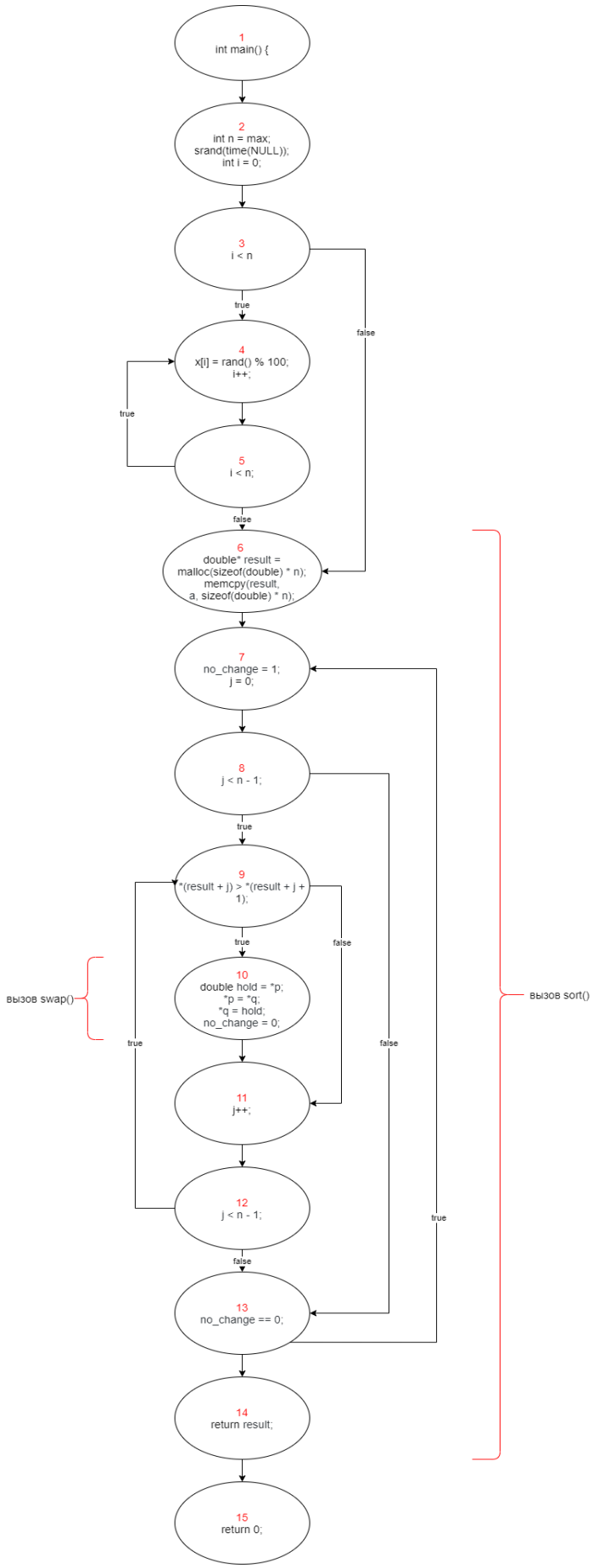


Рисунок 5 – Управляющий граф для программы из лабораторной работы №1 после преобразований

Был выполнен расчёт структурной сложности этой программы по первому критерию – минимальное покрытие вершин и дуг управления.

Минимальное количество маршрутов – 2:

- 1 – 2 – **3** – 6 – 7 – **8** – **13** – 14 – 15
- 1 – 2 – **3** – 4 – **5** – 4 – **5** – 6 – 7 – **8** – **9** – 11 – **12** – **9** – 10 – 11 – **12** – **13** – 7 – **8** – **13** – 14 – 15

В итоге сложность программы $S_1 = 3 + 11 = \underline{14}$.

Был выполнен расчёт структурной сложности этой программы по второму критерию – каждый линейно-независимый цикл и ациклический участок программы.

Полное число вершин N = 15

Количество связывающих дуг Y = 20

Число связанных компонент P = 1 (максимально связанный граф получается при добавлении дуги 11 – 1)

Цикломатическое число $Z = Y – N + 2 \cdot P = 7$

Необходимые 7 линейно-независимых маршрутов:

- 4 – **5**
- **9** – 11 – **12**
- 7 – **8** – **13**
- 1 – 2 – **3** – 6 – 7 – **8** – **13** – 14 – 15
- 1 – 2 – **3** – 4 – **5** – 6 – 7 – **8** – **13** – 14 – 15
- 1 – 2 – **3** – 4 – **5** – 6 – 7 – **8** – **9** – 11 – **12** – **13** – 14 – 15
- 1 – 2 – **3** – 4 – **5** – 6 – 7 – **8** – **9** – 10 – 11 – **12** – **13** – 14 – 15

В итоге сложность программы $S_2 = 1 + 2 + 2 + 3 + 4 + 6 + 6 = \underline{24}$.

Для выполнения автоматического расчёта с помощью программы ways.exe был подготовлен специальный файл *nodes2.txt*, задающий граф программы. Этот файл представлен в Приложении В. Результат работы программы ways.exe представлен на рисунках 6, 7 и 8.

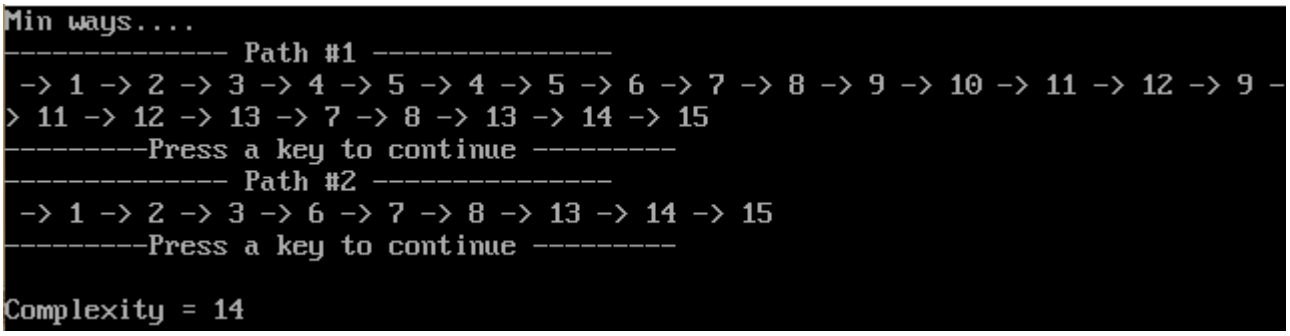


Рисунок 6 – Расчёт сложности по первому критерию программой ways.exe

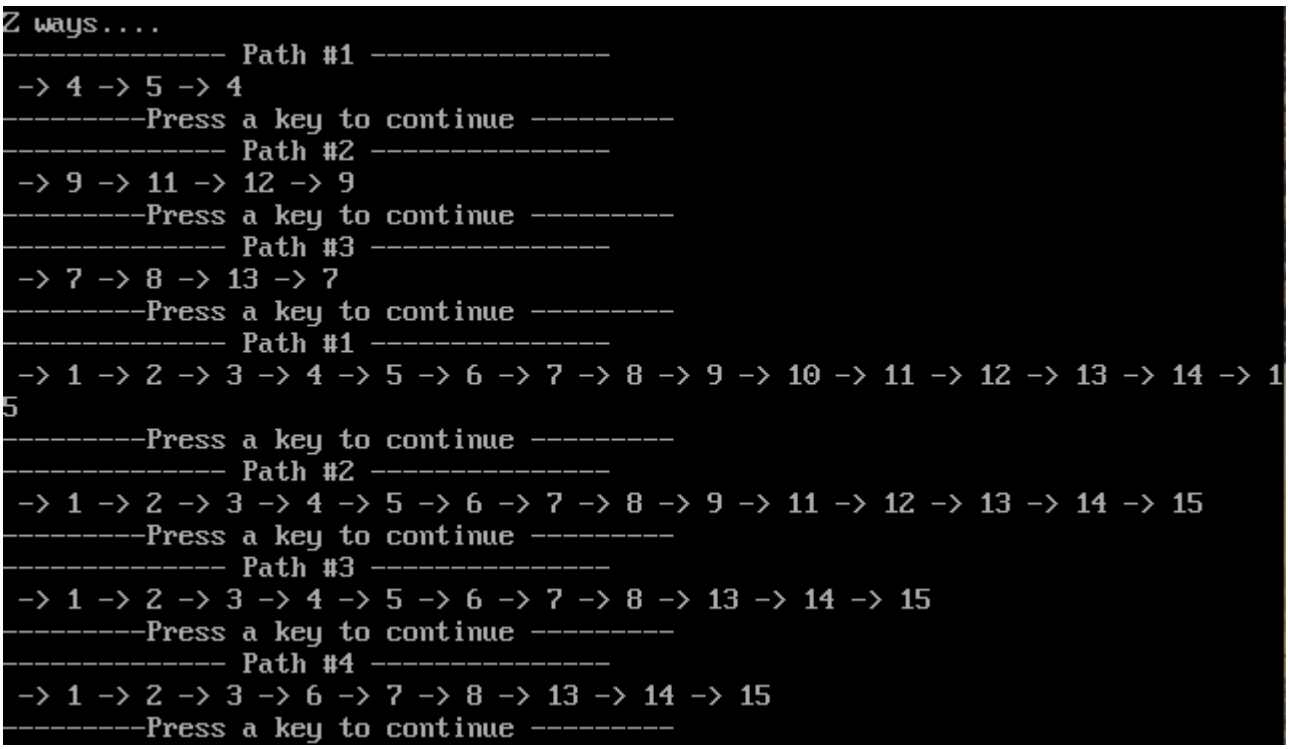


Рисунок 7 – Расчёт сложности по второму критерию программой ways.exe

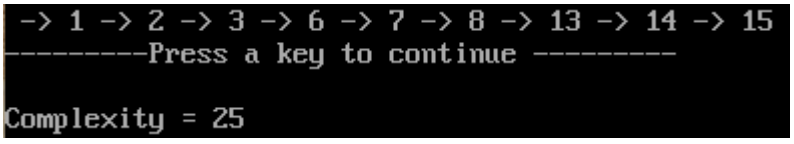


Рисунок 8 – Расчёт сложности по второму критерию программой ways.exe

Выводы.

В ходе выполнения лабораторной работы были изучены методы оценки структурной сложности программы на основе управляющего графа, была рассчитана структурная сложность двух программ по двум критериям. Для программы, взятой из первой лабораторной работы, был составлен управляющий граф на основе файла потока управления в функциях, сгенерированного с

помощью gcc. Расчёт структурной сложности программ выполнялся двумя методами: вручную и с помощью программы ways.exe.

ПРИЛОЖЕНИЯ

Приложение А. Описание первого графа на входном языке программы анализа.

```
Nodes{
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
}

Top{1}
Last{11}

Arcs{
arc(1,2);
arc(1,4);
arc(2,3);
arc(3,5);
arc(4,6);
arc(5,7);
arc(6,4);
arc(6,10);
arc(7,5);
arc(7,8);
arc(8,3);
arc(8,9);
arc(9,2);
arc(9,10);
arc(10,11);
}
```

Приложение Б. Граф потока управления, полученный с помощью gcc.

```
;; Function swap (swap, funcdef_no=5, decl_uid=3054, cgraph_uid=5, symbol_order=5)


;; 1 loops found
;;
;; Loop 0
;;  header 0, latch 1
;;  depth 0, outer -1
;;  nodes: 0 1 2
;; 2 succs { 1 }
swap (double * p, double * q)
{
  double hold;

  <bb 2> [0.00%]:
  hold = *p;
  _1 = *q;
  *p = _1;
```

```
  *q = hold;
  return;

}



;; Function sort (sort, funcdef_no=6, decl_uid=3059, cgraph_uid=6, symbol_order=6)

;; 3 loops found
;;
;; Loop 0
;;  header 0, latch 1
;;  depth 0, outer -1
;;  nodes: 0 1 2 3 4 5 6 7 8 9 10
;;
;; Loop 1
;;  header 3, latch 8
;;  depth 1, outer 0
;;  nodes: 3 8 7 6 4 5
;;
;; Loop 2
;;  header 7, latch 6
;;  depth 2, outer 1
;;  nodes: 7 6 4 5
;; 2 succs { 3 }
;; 3 succs { 7 }
;; 4 succs { 5 6 }
;; 5 succs { 6 }
;; 6 succs { 7 }
;; 7 succs { 4 8 }
;; 8 succs { 3 9 }
;; 9 succs { 10 }
;; 10 succs { 1 }
sort (double * a, int n)
{
  int j;
  char no_change;
  double * result;
  double * D.3088;

  <bb 2> [0.00%]:
```

```
_1 = (long unsigned int) n;
_2 = _1 * 8;
result = malloc (_2);
_3 = (long unsigned int) n;
_4 = _3 * 8;
memcpy (result, a, _4);

<bb 3> [0.00%]:
no_change = 1;
j = 0;
goto <bb 7>; [0.00%]

<bb 4> [0.00%]:
_5 = (long unsigned int) j;
_6 = _5 * 8;
_7 = result + _6;
_8 = *_7;
_9 = (sizetype) j;
_10 = _9 + 1;
_11 = _10 * 8;
_12 = result + _11;
_13 = *_12;
if (_8 > _13)
  goto <bb 5>; [0.00%]
else
  goto <bb 6>; [0.00%]

<bb 5> [0.00%]:
_14 = (sizetype) j;
_15 = _14 + 1;
_16 = _15 * 8;
_17 = result + _16;
_18 = (long unsigned int) j;
_19 = _18 * 8;
_20 = result + _19;
swap (_20, _17);
no_change = 0;

<bb 6> [0.00%]:
j = j + 1;

<bb 7> [0.00%]:
```

```
  _21 = n + -1;
  if (j < _21)
    goto <bb 4>; [0.00%]
  else
    goto <bb 8>; [0.00%]

  <bb 8> [0.00%]:
  if (no_change == 0)
    goto <bb 3>; [0.00%]
  else
    goto <bb 9>; [0.00%]

  <bb 9> [0.00%]:
  D.3088 = result;

<L7> [0.00%]:
  return D.3088;

}



;; Function write_arr (write_arr, funcdef_no=7, decl_uid=3071, cgraph_uid=7,
symbol_order=7)

;; 2 loops found
;;
;; Loop 0
;;  header 0, latch 1
;;  depth 0, outer -1
;;  nodes: 0 1 2 3 4 5
;;
;; Loop 1
;;  header 4, latch 3
;;  depth 1, outer 0
;;  nodes: 4 3
;; 2 succs { 4 }
;; 3 succs { 4 }
;; 4 succs { 3 5 }
;; 5 succs { 1 }
write_arr (double * a, int n)
{
```

```
  int i;

  <bb 2> [0.00%]:
  __builtin_putchar (10);
  i = 0;
  goto <bb 4>; [0.00%]

  <bb 3> [0.00%]:
  _1 = (long unsigned int) i;
  _2 = _1 * 8;
  _3 = a + _2;
  _4 = *_3;
  printf ("%7.1f ", _4);
  i = i + 1;

  <bb 4> [0.00%]:
  if (i < n)
    goto <bb 3>; [0.00%]
  else
    goto <bb 5>; [0.00%]

  <bb 5> [0.00%]:
  __builtin_putchar (10);
  return;

}



;; Function main (main, funcdef_no=8, decl_uid=3077, cgraph_uid=8, symbol_order=8)

Removing basic block 7
Merging blocks 5 and 6
;; 2 loops found
;;
;; Loop 0
;;  header 0, latch 1
;;  depth 0, outer -1
;;  nodes: 0 1 2 3 4 5 6
;;
;; Loop 1
;;  header 4, latch 3
```

```
;;  depth 1, outer 0
;;  nodes: 4 3
;; 2 succs { 4 }
;; 3 succs { 4 }
;; 4 succs { 3 5 }
;; 5 succs { 6 }
;; 6 succs { 1 }
main ()
{
  int i;
  double x[80];
  int n;
  int D.3090;

  <bb 2> [0.00%]:
  n = 80;
  _1 = time (0B);
  _2 = (unsigned int) _1;
  srand (_2);
  i = 0;
  goto <bb 4>; [0.00%]

  <bb 3> [0.00%]:
  _3 = rand ();
  _4 = _3 % 100;
  _5 = (double) _4;
  x[i] = _5;
  i = i + 1;

  <bb 4> [0.00%]:
  if (i < n)
    goto <bb 3>; [0.00%]
  else
    goto <bb 5>; [0.00%]

  <bb 5> [0.00%]:
  sort (&x, n);
  D.3090 = 0;
  x = {CLOBBER};

<L4> [0.00%]:
  return D.3090;
```

```
}
```

Приложение В. Описание второго графа на входном языке программы анализа.

```
Nodes{
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
}

Top{1}
Last{15}

Arcs{
arc(1,2);
arc(2,3);
arc(3,4);
arc(3,6);
arc(4,5);
arc(5,4);
arc(5,6);
arc(6,7);
arc(7,8);
arc(8,9);
arc(8,13);
arc(9,10);
arc(9,11);
arc(10,11);
arc(11,12);
arc(12,9);
arc(12,13);
arc(13,7);
arc(13,14);
arc(14,15);
}
```