

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)**

Факультет Информатика и вычислительная техника

(наименование факультета)

Кафедра Кибербезопасность информационных систем

(наименование кафедры)

**Практическое занятие № 4**

на тему «Оценка структурной сложности программ»

Выполнил обучающийся гр. ВКБ43

Ковалев Данил Петрович

Проверил:

Кривицкий Андрей Михайлович

Ростов-на-Дону

2025

**Цель работы:**

Изучить порядок оценки структурной сложности программного обеспечения.

**Контрольные вопросы:**

1. На какие группы можно разделить совокупность маршрутов исполнения программного модуля?

Совокупность маршрутов исполнения программного модуля условно можно разделить **на две группы**: вычислительные маршруты, маршруты принятия логических решений.

1. Как оценивается сложность вычислительных маршрутов?

Сложность вычислительных маршрутов можно оценить следующей формулой:

(1)

где: *т* - количество маршрутов исполнения программы; - число данных, обрабатываемых в *i*-м маршруте; - число значений обрабатываемых данных *j* типа (2 < < 5).

1. По каким критериям оценивается структурная сложность программ?

**Критерий 1**

Данный критерий предполагает, что граф потока управления программой должен быть проверен по минимальному набору маршрутов, проходящих через каждый оператор ветвления по каждой дуге.

**Критерий 2**

Критерий основан на анализе базовых маршрутов в программе, которые формируются и оцениваются на основе цикломатического числа графа потока управления программы.

**Критерий 3**

Более сильные критерии проверки и определения сложности структуры программы включают требования однократной проверки не только линейно независимых, но и всех линейно зависимых циклов и ациклических маршрутов. К таким критериям относится третий критерий формирования маршрутов для тестирования программ.

1. Что такое цикломатическое число исходного графа?

*Цикломатическое число Z* исходного графа потока управления программой определяется формулой

где *т -* общее число дуг в графе; *п* - общее число вершин в графе; *р -* число связных компонентов графа.

1. Что такое матрица смежности и матрица достижимости?

*Матрица смежности* - квадратная матрица, в которой единицы располагаются в позициях *(i,j)*, если в графе потока управления программой имеется дуга *(i,j).* В противном случае, при отсутствии дуги в такой позиции, ячейки матрицы просто не заполняются, обозначая нулевое значение в соответствующей позиции.

Матрицей достижимости называется квадратная матрица, в которой единицы располагаются в позиции, соответствующей дуге *(i,j)*.

1. Как оценивается структурная сложность программы по метрике Маккейба?

Метрика Маккейба характеризует цикломатическое число графа потока управления программой и определяется следующим выражением:

где m *-* количество ребер графа, n *-* число вершин графа.

Величину М*,* рассчитанную по этой формуле, называют цикломатическим числом Маккейба*.*

**Задания:**

1) Каждым студентом самостоятельно выбирается программное средство, разработанное им на других лабораторных работах по любому предмету.

2) В соответствии с разработанной программой составить блок-схему алгоритма решения задачи.

3) На основании блок-схемы составить управляющий граф программы и оценить ее алгоритмическую сложность по каждому из критериев.

4) Сделать выводы алгоритмической сложности, разработанной им программы.

Для выполнения поставленных задач мной была выбрана программа для реализации шифрования и расшифрования текста методом шифрующих таблиц.

**Реализация программы**

import java.util.Scanner;  
  
public class Exercise\_1 {  
 static int *n*; // количество столбцов  
 static int *k*; // количество строк  
 static String[] *key* = null; // ключ шифра в формате {столбец, строка}  
 static boolean *state*;  
 static boolean *space*;  
 static String *text*; // исходный текст  
  
 public static void main(String[] args) {  
 while (true)  
 try{  
 *n* = *getInt*("Введите количество столбцов: ");  
 *k* = *getInt*("Введите количество строк: ");  
 *key* = *getText*("Введите ключ шифра через пробел: ").split(" ");  
 *text* = *getText*("Введите текст: ");  
 if(*text*.toLowerCase().equals("exit") || *text*.toLowerCase().equals("выход")){  
 break;  
 }  
 *space* = *getText*("Учитывать пробелы?:\n1. Да\n2. Нет\n").equals("1") ? true : false;  
 *state* = *getText*("Выберете режим:\n1. Зашифровка\n2. Расшифровка\n").equals("1") ? true : false;  
 *text* = *space* ? *text* : *text*.replaceAll(" ", "");  
 System.*out*.println(*state* == true ? *encrypt*(*text*) : *decrypt*(*text*));  
 }catch (Exception ex){  
 System.*out*.println("Ошибка ввода. Повторите попытку");  
 }  
 System.*out*.println("Всего доброго");  
 }  
 static String getText(String message){  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.print(message);  
 String input = scan.nextLine();  
 return input;  
 }  
 static int getInt(String message){  
 Scanner scan = new Scanner(System.*in*);  
 System.*out*.print(message);  
 int number = Integer.*valueOf*(scan.nextLine());  
 return number;  
 }  
 static String encrypt(String text){  
 System.*out*.println("Шифруем текст. Ожидайте");  
 StringBuilder message = new StringBuilder(text); // за основу возьмём исходный текст  
 char ch; // символ, на который мы будет заменять конкретную похицию в тексте  
 int position; // позиция, в которой будет происходить замена  
 for(int i = 0; i < *n*; i++) // запускаем двойной цикл для вычисления индексов  
 for(int j = 0; j < *k*; j++){  
 ch = text.charAt(i \* *n* + j); // находим конкретный символ  
 position = (Character.*digit*(*key*[1].charAt(i), 10) - 1) \* *n* +  
 Character.*digit*(*key*[0].charAt(j), 10) - 1; // вычисляем его позицию в закодированной строке  
 message.setCharAt(position, ch); // вставляем символ на вычисленную позицию  
 }  
 return "Зашифрованное сообщение: " + message.toString();  
 }  
 static String decrypt(String text){  
 System.*out*.println("Расшифровываем сообщение. Ожидайте");  
 char ch; // символ, на который заменится символ из исходной закодированной строки  
 int position; // позиция, в которой будет происходить замена  
 StringBuilder message = new StringBuilder(text); // за основу берём закодированную строку  
 for(int i = 0; i < *n*; i++) // запускаем двойной цикл для вычисления позициии  
 for(int j = 0; j < *k*; j++){  
 position = (Character.*digit*(*key*[1].charAt(i), 10) - 1) \* *n* +  
 Character.*digit*(*key*[0].charAt(j), 10) - 1; // вычисляем индекс для замены  
 ch = text.charAt(position); // находим символ  
 message.setCharAt(i \* *n* + j, ch); // процедура замены происходит в порядке, обратному кодированию  
 }  
 return "Расшифрованное сообщение: " + message.toString();  
 }  
}

**Блок-схема программы**

Для представленной программы построим блок-схему управления, которая показана на рисунке 1.

Изображение выглядит как текст, снимок экрана, графический дизайн, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1 – блок-схема управления программы

Построим теперь граф управления программы, он представлен на рисунке 2.

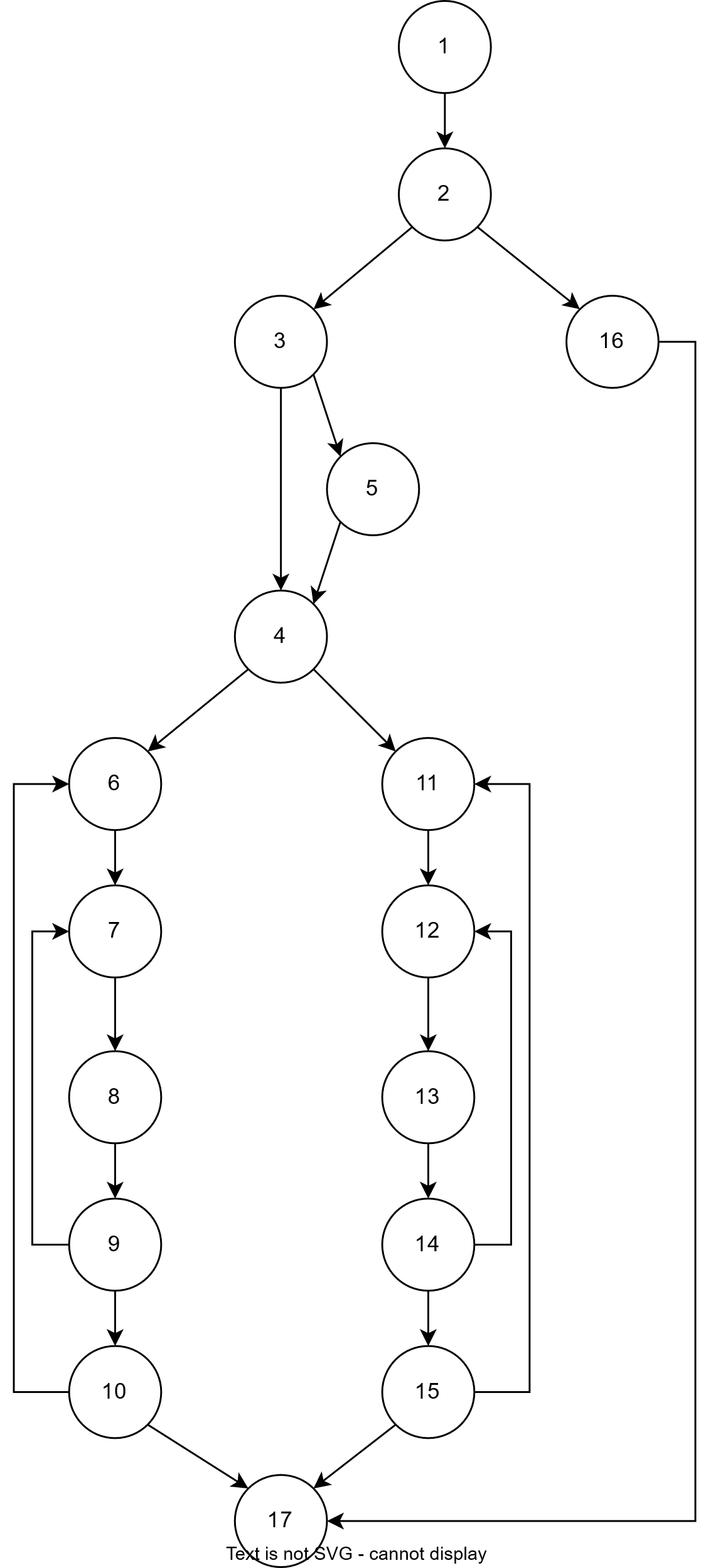


Рисунок 2 – граф управления программы

Проведем оценку алгоритмической сложности графа по первому критерию. Определим минимальный набор маршрутов, проходящих через каждый оператор ветвления и по каждой дуге.

Для составленного управляющего графа можно выделить 4 маршрута:

M1: 1-2-3-4-6-7-8-9-10-17;

M2: 1-2-3-5-4-6-7-8-9-10-17;

M3: 1-2-3-4-11-12-13-14-15-17 – 9;

M4: 1-2-3-5-4-11-12-13-14-15-17 – 10;

Данные маршруты проходят по всем вершинам ветвления, причем хотя бы один раз по каждой дуге графа, что соответствует требованиям первого критерия.

Таким образом, в соответствии **с первым критерием оценки алгоритмической сложности** необходимое количество маршрутов равно 4, количество вершин ветвления в маршрутах определяет уровень сложности:

Проведем оценку алгоритмической сложности **по второму критерию**. Необходимо определить число проверок каждого линейно независимого цикла и линейно независимого ациклического участка программы. Количество проверок определяется цикломатическим числом графа, которое определяется следующим соотношением:

,

где -число вершин ветвления.

Число вершин ветвления в составленном графе составляет 6, отсюда

Z = + 1 = 6+ 1 = 7.

Таким образом, общее число циклических и ациклических участков равно 7. Выделим маршруты на заданном графе:

- Ациклические маршруты:

M1: 1-2-3-4-6-7-8-9-10-17; 𝑝1= 10

M2: 1-2-3-5-4-6-7-8-9-10-17; 𝑝2=11

M3: 1-2-3-4-11-12-13-14-15-17 – 9; 𝑝3=10

M4: 1-2-3-5-4-11-12-13-14-15-17 – 10; 𝑝4=11

M5: 1-2-16-17; 𝑝5=4

- Циклические маршруты:

M6: 6-7-8-9-10;

M7: 11-12-13-14-15;

Тестирование программы по указанным маршрутам позволит проверить все операторы ветвления программы. Метрика структурной сложности определяется по следующему соотношению:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 1 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  | 1 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  | 1 |  |  | 1 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
| 11 |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  | 1 |  |  | 1 |  |  |  |
| 13 |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |
| 16 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  | 1 | 1 |  |

Таблица 1 – матрица смежности

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 11 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 12 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 13 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 14 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| 17 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |

Таблица 2 – матрица достижимости

Элементы, выделенные черным цветом, соответствуют маршрутам:

M1:6-7-8-9-10;

M2:11-12-13-14-15;

Все строки матрицы получились идентичными, так как все вершины графа входят в состав ацикличных маршрутов.

Проведем оценку алгоритмической сложности по третьему критерию. В соответствии с третьим критерием необходимо выделить все реально возможные маршруты управления:

M1: 1-2-3-4-6-7-8-9-10-17; 𝑝1= 10

M2: 1-2-3-5-4-6-7-8-9-10-17; 𝑝2=11

M3: 1-2-3-4-11-12-13-14-15-17 – 9; 𝑝3=10

M4: 1-2-3-5-4-11-12-13-14-15-17 – 10; 𝑝4=11

M5: 1-2-16-17; 𝑝5=4

M6: 6-7-8-9-10;

M7: 11-12-13-14-15;

Оценку структурной сложности программы рассчитываем по следующему соотношению:

Для организации автоматического анализа заданного графа по второму критерию с помощью вычислительных средств построим матрицы смежности и достижимости.

***Вывод:***

Исходя из полученных результатов расчета метрик структурной сложности по первому (*S1 = 42*), второму (S2 *= 56*) и третьему (*S3 = 56*) критериям выделения маршрутов можно сделать вывод, что программа, характеризуемая заданным графом управления, имеет невысокую алгоритмическую сложность, так как количество используемых в тексте операторов условий 7, для проверки которых необходимо проверить от 42 до 56 тестовых вариантов исходных данных.

Оценим алгоритмическую сложность программы **на основе метрики Маккейба.**

В соответствии с теорией Маккейба сложность алгоритма оценивается величиной цикломатического числа, которая определяется по следующему соотношению: Z = т - п + 2, где т - количество дуг управляющего графа, построенного на основе алгоритма программы, п - количество вершин графа. В соответствии с полученным управляющим графом т = 23, п = 17, тогда цикломатическое число равно:

***Вывод:***

В соответствии со значением цикломатического числа (*Z = 8*) в полученном графе управления программой можно выделить 8 независимых контуров, которые определяют пять управляющих маршрутов, ведущих из начальной вершины в конечную.

Значение цикломатического числа для полученного графа (*Z = 8*) не превышает значения 10, что говорит о незначительной сложности алгоритма решения задачи расчета значений функции.