ПЗ 2. Цикломатические меры сложности. Метрика Мак-Кейба

(Составили Никонова Г.В. ngvlad@mail.ru, Никонов А.В. nalva@mail.ru)

ЭЛЕКТРОННЫЙ ВАРИАНТ

https://disk.yandex.ru/d/xgBQ5OhfmR1fdg

Тематика: цикломатическая (топологическая) сложность программ. Метрика МакКейба

Задание. 1. Выбрать программу, в которой ОБЯЗАТЕЛЬНО должны быть ВЕТВЛЕНИЯ, или ЦИКЛЫ (или то и другое). Представить графически схему алгоритма программы [1]. **Установочный материал см. в приложении А.**

2. Представить поток управления в виде графа, сделав оценку свойств передачи управления в программе («дерево обязательного предшествования»). *Пример* **построения «графа потока управления»** приведён **в окончании приложении Б. Приёмы построения графов потока управления даны в приложении Д.**

3. Определить цикломатическую сложность графа потока управления. *Пример* **определения цикломатической сложности графа потока управления** приведён **в приложении В.**

4. Сделать выводы по работе, см. **приложение Г.**

1 ВВОДНЫЙ МАТЕРИАЛ

Цикломатическая сложность программ – топологическая мера сложности программ, используемая для измерения качества программного обеспечения и **основанная на методах статического анализа кода**.

Метрики сложности потока управления базируются на анализе управляющего графа программы. Представителем данной группы **является метрика МакКейба** (1976 г.):

здесь производятся непосредственные численные измерения для линейно независимых путей в исходных кодах программ.

(Граф – графическое представление объектов и связей между ними. Объекты представляются как вершины, или узлы графа, а связи – как дуги, или рёбра. Для разных областей применения виды графов могут различаться направленностью, ограничениями на количество связей и дополнительными данными о вершинах или рёбрах.)

Анализ потока управления решает задачи:

1) определение свойств передачи управления в программах – то есть **определение свойств передачи управления между операторами** программы;

2) определение зависимости операторов по управлению.

Для **анализа потока управления строят «дерево непосредственного (обязательного) предшествования» – граф, вершины которого показывают множество операторов, которые должны быть завершены до того, как начнется выполнение рассматриваемого оператора.**

Строится такое дерево по нескольким правилам:

– вершина v обязательно предшествует вершине **w**, если **v** принадлежит каждому пути в графе от `start` до **w**. В частности, любая вершина обязательно предшествует себе самой;

– вершина v строго обязательно предшествует вершине **w**, если она обязательно ей предшествует, но не совпадает с ней;

– вершина v непосредственно предшествует вершине **w**, если она является ближайшей к **w** вершиной, которая ей строго предшествует.

Пример **некоторого дерева обязательного предшествования** отображен на рисунке ниже.

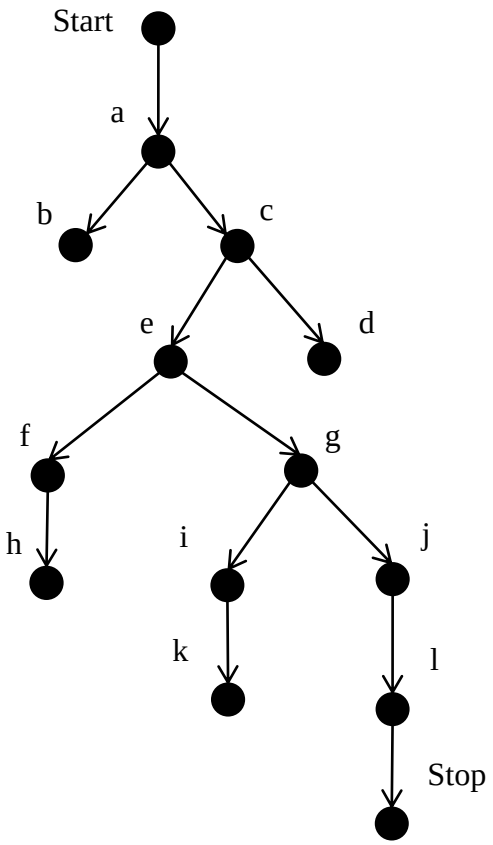


Рисунок 2.1– **Дерево непосредственного (обязательного) предшествования**

Управляющий граф программы, который используют метрики данной группы**, может быть построен на основе схем модулей в алгоритме программы**. Поэтому **метрики** этой группы **могут применяться для оценки сложности промежуточных продуктов разработки**.

Граф потока управления программы: **узлы графа соответствуют неделимым группам команд** программы **и ориентированным рёбрам**, **каждое из которых соединяет два узла и соответствует двум командам**, **вторая из которых может быть выполнена сразу после первой**.

Цикломатическая сложность может также быть применена для отдельных функций, модулей, методов или классов в пределах программы.

Измерение цикломатической сложности **позволяет** оценить качество программного кода **и выявить процедуры с повышенной сложностью. Процедуры с повышенной сложностью подвержены ошибкам и их выявление желательно для проведения их аудита**.

Показатель цикломатической сложности **по МакКейбу показывает требуемое количество проходов для покрытия всех связей в графе, то есть количество тестовых прогонов программы, необходимых для наиболее полного тестирования**.

Упрощённая формула для вычисления цикломатического числа Мак-Кейба (цикломатической сложности) $Z(G)$ следующая:

$$Z(G) = e - v + 2p, \quad (2.1)$$

где e – **число дуг** ориентированного графа G (ориентированный граф (орграф) – граф, рёбрам которого присвоено направление. Направленные рёбра называют дугами, рёбрами);

v – **число вершин**;

p – **число компонентов связности** графа.

Число компонентов связности графа **можно рассматривать как количество дуг, которые необходимо добавить для преобразования графа в сильно связный**.

Сильно связным называется граф, любые две вершины которого взаимно достижимы. Для сильносвязного графа число компонентов связности равно единице: $p = 1$.

Для графов корректных программ, т. е. графов, **не имеющих недостижимых от точки входа участков и «висячих» точек входа и выхода, сильно связный граф**, как правило, **получается путём замыкания дугой вершины**, обозначающей **конец программы, на вершину, обозначающую точку входа** в эту программу.

По сути $Z(G)$ **определяет число линейно независимых контуров в сильно связном графе,**

а цикломатическое число Мак-Кейба показывает требуемое количество проходов для покрытия всех контуров сильно связного графа или количество тестовых прогонов программы, необходимых для исчерпывающего тестирования по критерию «работает каждая ветвь».

Поток управления – это последовательность выполнения различных модулей и операторов программы.

2 МЕТРИКА МакКейба

Данная метрика **позволяет** (в отличие от метрик Холстеда) **учесть логику построения программы при оценке её сложности.**

Программное система (алгоритм, спецификация) **должно быть представлено в виде управляющего ориентированного графа** с вершинами и дугами, где **вершины соответствуют операторам**, а **дуги характеризуют переход управления от одного оператора к другому.**

Для представления программы в виде графа обычно **учитывают только исполнимые операторы и исключают группы операторов, назначением которых является описание данных**. Желательно выбирать такие синтаксические формы операторов, которые в наибольшей степени подходят для отображения в вид узла графа.

Линейные участки программы можно заменить одним узлом графа, относя к нему группы операторов, выполнение которых осуществляется в прямой последовательности без каких-либо условий (это характерно для участков программ, содержащих последовательность прямых вычислений переменных).

При любом варианте представления анализируемой программы **желательно**

преобразовать операторы цикла в эквивалентную последовательность операторов ветвления, **добавив,** при необходимости, **операторы подсчёта числа повторений цикла с «верхним» или «нижним» окончанием (счётчики циклов)**.

При оценке сложности программы с применением цикломатического числа МакКейба справедливо следующее положение: если значение цикломатического числа больше 10, это означает, что программа обладает излишней сложностью и её следует разбить на составные части (независимые модули или подпрограммы), **для которых цикломатическое число будет иметь меньшее значение**.

3 ОСОБЕННОСТИ ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ ГРАФОВ

Имеется **несколько общих замечаний, справедливых практически для любых программных блоков**.

1. При построении графа управления программой надо учитывать только исполнимые операторы, не учитывая операторы описания.

2. Если в программе существует несколько операторов, не изменяющих порядок действий в программе, и они следуют один за другим, их допускается объединять в одну вершину графа.

3. При построении графа управления операторы цикла следует заменить несколькими вершинами.

Например: для оператора типа ***for*** **должны быть вершины**, соответствующие:

– операторам начального присваивания счётчика цикла;

– **его изменению и ветвлению, определяющим продолжение или завершение выполнения цикла**.

4. Аналогично, несколькими вершинами заменяют и другие операторы цикла.

Они должны соответствовать действиям, предшествующим циклу, определяющим продолжение или прекращение выполнения цикла, а также группе операторов, составляющих тело цикла – множеству повторяемых действий.

Таким образом, в графе должны быть три группы вершин, определяющих три элемента, составляющих любой цикл:

– начало цикла;

– тело цикла;

– ветвление при окончании цикла – завершение или возвращение к исходному оператору).

Приёмы построения графов потока управления даны в приложении Д [2].

4 ГРАФ ПОТОКА УПРАВЛЕНИЯ

Для программы строится ориентированный граф (граф, рёбрам которого присвоено направление)**, содержащий лишь один вход и один выход**,

при этом **вершины графа соотносят с теми участками кода программы, в которых имеются лишь последовательные вычисления, и отсутствуют операторы ветвления и цикла**,

а дуги соотносят с переходами от блока к блоку и с ветвями выполнения программы.

Условие при построении данного графа: каждая вершина достижима из начальной, и конечная вершина достижима из любой другой вершины.

Граф потока управления – это ориентированный граф с двумя выделенными вершинами ***start*** и ***stop***, **такими, что**:

– в ***start*** не заходит ни одна дуга;

– из ***stop*** не выходит ни одна дуга;

– произвольная вершина принадлежит хотя бы одному пути из ***start*** в ***stop***.

5 ЦИКЛОМАТИЧЕСКОЕ ЧИСЛО ГРАФА КАК МЕРА СЛОЖНОСТИ ПО

По сути, цикломатическое число $Z(G)$ **определяет число линейно независимых контуров в сильно связном графе (то есть тех циклов, которые не содержат в себе других циклов)**. В корректно написанных программах $p = 1$.

Цикломатическое число МакКейба **показывает требуемое количество проходов для покрытия всех контуров** сильно связного графа, **или количество тестовых прогонов программы, необходимых для исчерпывающего тестирования** по критерию **«работает каждая ветвь»**.

Фактически, чтобы найти эту метрику **нужно определить количество ветвей запрограммированного алгоритма. Подсчёт сводится к определению числа операторов *if*, *for*, *while*, *case* в операторе *switch***. Большая цикломатическая сложность потребует больше тестов для покрытия ветвей алгоритма.

К достоинствам меры относят простоту её вычисления **и повторяемость результата**, а также **наглядность и содержательность интерпретации**.

В качестве **недостатков можно отметить**:

– **нечувствительность к размеру ПО**;

– **нечувствительность к изменению структуры ПО**;

– **отсутствие корреляции со структурированностью ПО**;

– **отсутствие различия между конструкциями «Развилка» и «Цикл»**;

– **отсутствие чувствительности к вложенности циклов**.

ПРИЛОЖЕНИЕ А

Графическое представление схемы алгоритма программы

1) представить рисунком схему алгоритма программы, выполненной на знакомом языке программирования (одном языке).

Размер листинга не более одной–двух страниц формата А4, индивидуален для каждого студента.

Пример программы алгоритма схемы приведён на рисунке А.1, [1].

Схема алгоритма программы должна отражать последовательность операций в программе, а не показывать схему работы какой-то системы.

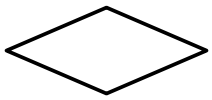
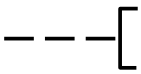
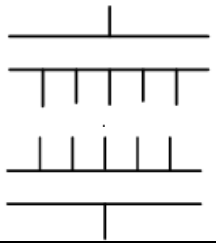
Не использовать синтаксис языка программирования, и при необходимости руководствоваться п. 4.1.4 ГОСТ [1] («Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария. Если использование символов комментария может запутать или разрушить ход схемы, текст следует помещать на отдельном листе и давать перекрестную ссылку на символ»).

Также, п. 4.1.5 ГОСТ [1]: («В схемах может использоваться идентификатор символов. Это связанный с данным символом идентификатор, который определяет символ для использования в справочных целях в других элементах документации (например, в листинге программы). Идентификатор символа должен располагаться слева над символом»).

Условные графические обозначения (УГО) и правила выполнения по ГОСТ – краткая справка в таблице А.1.

Таблица А.1 – УГО в схемах алгоритмов, программ, данных и систем

УГО	Наименование	Назначение
	**Данные**	Определяет ввод или вывод на внешнее устройство или носитель данных.
	**Процесс**	Отражает обработку данных (выполнение операции, группы операций).

	**Типовой процесс**	Отображает предопределенный процесс, состоящий из одной или нескольких операций программы, которые определены в другом месте.
	**Подготовка**	Отображает модификацию параметра для управления циклом со счетчиком.
	**Решение**	Описывает проверку условия и выполняет переключение по одному из условий. Имеет один вход и два или более альтернативных выходов, один из которых активизируется после вычисления условия.
 	**Граница цикла**	Состоит из двух частей: начала и конца цикла. Обе части имеют один и тот же идентификатор. Изменение значения идентификатора, условия для выполнения или завершения помещаются внутри символов в начале или в конце цикла.
	**Соединитель**	Используется для обрыва линии и продолжения ее в другом месте.
	**Знак завершения**	Определяет начало и конец структурной схемы алгоритма.
	**Комментарий**	Используется для добавления пояснительных записей.
	Основная линия	Отражает последовательность выполнения действий в алгоритме.
	начало **Параллельные действия** конец	Применяется в случае одновременного выполнения операций, отображаемых несколькими символами.

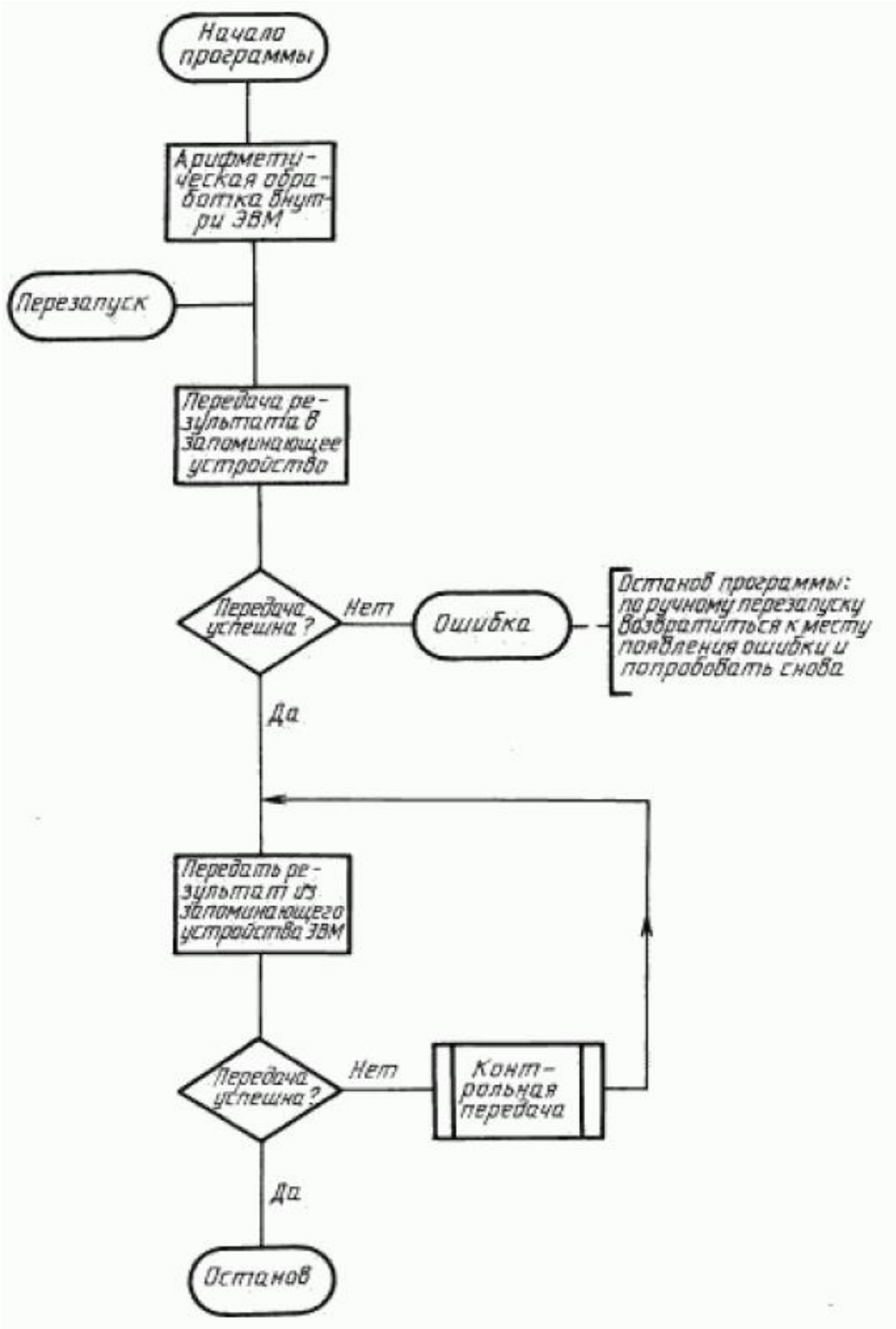


Рисунок А.1 – Пример схемы программы (алгоритма)

ПРИЛОЖЕНИЕ Б

Построение графа потока управления

Проанализировать листинг, опираясь на синтаксис языка. Листинг, рассмотренный в данном примере, показан после рисунка Б.1.

Листинг приводить в приложении к основной части отчёта по ПЗ2, а не в теле его основной части.

Для уяснения алгоритма, реализованного в программе, предварительно построить «дерево непосредственного (обязательного) предшествования».

Граф потока управления – это **множество всех возможных путей исполнения программы, представленное в виде графа.** Граф потока управления рассматриваемого программного модуля приведён на **рисунке Б.1.**

Этот граф является ориентированным с двумя выделенными вершинами `start` и `stop`, такими, что:

– в `start` не заходит ни одна дуга;

– из `stop` не выходит ни одна дуга;

– произвольная вершина принадлежит хотя бы одному пути из `start` в `stop`.

Вершинами графа являются **операции программы** или же **группа последовательных операций, которые рассматриваются как одно целое. Стрелки (дуги графа) означают передачу управления от одной вершины к другой,** тем самым обеспечивается проход графа. Так как граф ориентированный, **то передача управления обеспечивается только в направлении дуги.**

В отличие от языков программирования или схемы алгоритмов**, в графе потока управления нет отдельных операторов цикла или условия. Цикл оформ-**

ляется с помощью нескольких вершин, одна из которых определяет инициализацию цикла, остальные описывают все производимые операции в цикле, а последняя вершина обеспечивает замыкание.

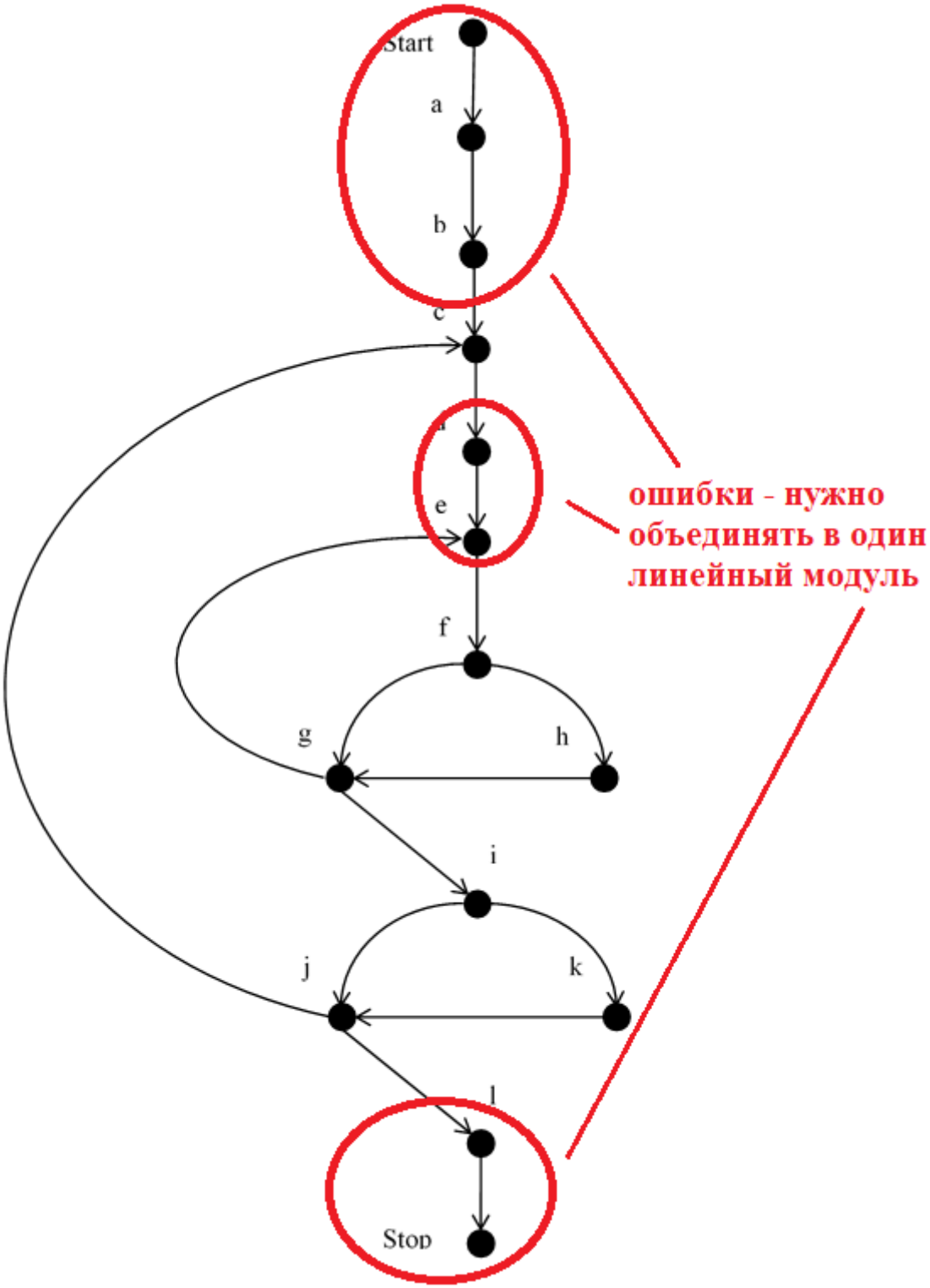


Рисунок Б.1 – **Пример графа потока управления**

Подобно отображается и «**условие**»: одна вершина **определяет само условие,** а остальные **описывают исходы этого условия. Граф,** приведённый на **рисунке Б.1, имеет два цикла, которые обеспечивают проход по символам входной строки и символам алфавита,** а также два условия: **на равенство символов, и является ли символ входной строки знаком препинания.**

Листинг программного модуля шифрования текста шифром Цезаря

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace _3_crypto
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        string alf = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU-
VWXYZ";

        private void button1_Click(object sender, EventArgs e)
        {
            int key = 3;
            string enterStr = textBox1.Text;
            string encStr = "";

            for (int i = 0; i < enterStr.Length; i++)
            {
                char p = enterStr[i];

                for (int j = 0; j < alf.Length; j++)
                {
```

```
                    char c = alf[j];

                    if (c == p)
                    {
                        encStr += alf[(j + key) % 52];
                    }
                }

                if (p == ' ' || p == '.' || p == ',')
                {
                    encStr += p;
                }
            }

            textBox1.Text = encStr;
        }
    }
}
```

ПРИЛОЖЕНИЕ В

Определение цикломатической сложности

Цикломатическая сложность программы – структурная (или топологическая) мера сложности компьютерной программы. Это количество линейно независимых маршрутов через программный код.

Для вычисления цикломатической сложности по формуле (2.1) **необходимо определить характеристики графа**. Рассмотрим граф потока управления, который изображён на рисунке Б.1. Эти характеристик приведены **в таблице В.1**.

Таблица В.1 – **Характеристики графа потока управления**

Характеристика	Значение
Количество дуг	17
Количество вершин	14
Компонента связности	1

Компонента связности в рассматриваемом примере будет равна единице, т. к., чтобы граф потока управления стал сильно связным, **необходимо добавить только одну дугу, соединяющую вершины `start` и `stop`**.

Подставив все значения в формулу (1), цикломатическая сложность графа потока управления будет равна пяти: **Z(G) = 5**:

$$Z(G) = e - v + 2p = 17 - 14 + 2 \cdot 1 = 5. \quad (2.1)$$

ПРИЛОЖЕНИЕ Г

Выводы по работе

С одной стороны, это значение $Z(G)$ является оптимальным, потому что **для метрики МакКейба принято – если цикломатическое число меньше 10**, значит программа не обладает излишней сложностью и **не нуждается в декомпозиции, включающей разделение на модули**.

Данная метрика была изучена на примере оценки сложности программного модуля шифрования текста шифром Цезаря. Один из выводов, который был сформулирован в ходе работы, основан на построенном графе потока управления и заключается в том, что **чем меньше циклов, ветвлений и непоследовательных переходов в графе**, **тем легче выполнить тестирование программы по принципу**: «**каждый оператор работает хотя бы один раз**».

Поэтому мера МакКейба (цикломатическая сложность) **указывает, сколько тестовых прогонов нужно сделать при испытаниях программы**.

Метрика Мак-Кейба, несмотря на то, что была давно введена, и что сильно изменились технологии программирования, и информационные системы достигли большого развития, не теряет своей актуальности. Сама идея **контроля увеличения сложности программы** из-за циклов и ветвлений применяется и сейчас.

ПРИЛОЖЕНИЕ Д

Приёмы построения графов потока управления

Правильно построенный управляющий граф позволяет оценить структурную сложность программы, определяемую количеством независимых контуров в полносвязном графе. Методика построения графов применима к любым языкам программирования.

Линейная последовательность операторов. На рисунке Д.1 показан управляющий граф некоторой программы на ЯВУ, реализующей поставленную

задачу с помощью **линейной последовательности операторов**.

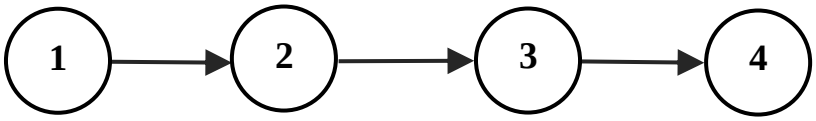


Рисунок Д.1 – **Управляющий граф линейной последовательности операторов**

Расчёты цикломатической сложности программы для этого графа управления программой по формуле (2.1) будут:

$$Z(G) = e - v + 2 = 3 - 4 + 2 = 1, \quad (Д.2)$$

где количество дуг $e = 3$ и количество вершин $v = 4$.

Эта же задача может быть решена другим кодом, для которого граф управления приведен на рисунке Д.2.

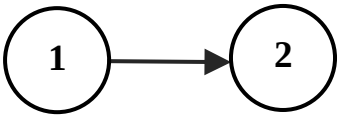


Рисунок Д.2 – **Граф управления сокращённой линейной последовательностью операторов**

Для этого графа расчёт цикломатической сложности программы будет (Д.3):

$$Z(G) = e - v + 2 = 1 - 2 + 2 = 1. \quad (Д.3)$$

Цикломатическая сложность программы осталось той же – последовательность операторов, выполняемых последовательно, может быть объединена в одну с одним входом и одним выходом, если они не изменяют последовательности выполнения программы.

Простое ветвление (оператор IF). На рисунке Д.3 показан управляющий граф некоторой программы, **реализующей задачу по вычислению** экстремальных **значений двух величин и присвоения наибольшего результата выходной переменной** ($c = max\{a, b\}$), которая может иметь граф управления программы с разветвлением по оператору **IF** (рисунок Д.3).

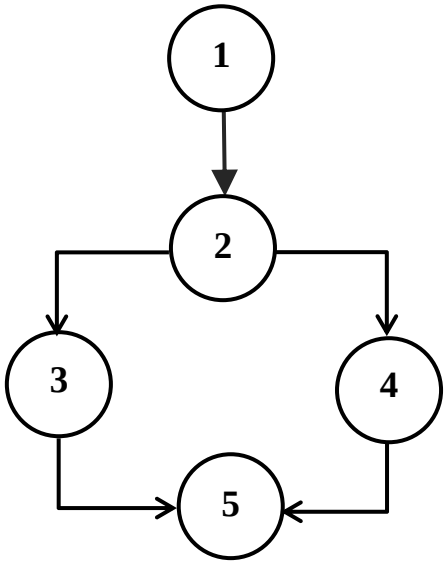


Рисунок Д.3 – **Граф управления программы с разветвлением по оператору IF**

Расчёт цикломатической сложности такой программы будет (Д.4):

$$Z(G) = e - v + 2 = 5 - 5 + 2 = 2. \quad (Д.4)$$

Переключатель с множественным выбором. Такой выбор реализуют операторы **SWITCH, SELECT, CASE** и подобные. Ниже приведён (как *пример*) граф управления программы на ЯВУ, **решающей задачу**: вводится символ, и **нужно определить, является ли** данный символ **допустимым для записи числа в римской системе счисления**. **Программа распознаёт** не все символы, а **только** три **(I, V, X).**

Граф потока управления таким фрагментом программы приведен на рисунке 3.4. Оценка цикломатической сложности этой программы будет (Д.5):

$$Z(G) = e - v + 2 = 9 - 7 + 2 = 4, \quad (Д.5)$$

где количество дуг графа 9, а количество вершин 7.

Для тестирования такого фрагмента программы **достаточно 4-х тестов** (*например*, с введёнными значениями переменной **I, V, X, v**).

В графе на рисунке 4 **введена одна фиктивная вершина**. Она **соответствует участку** программы, **куда передаётся управление после завершения оператора-SWITCH**.

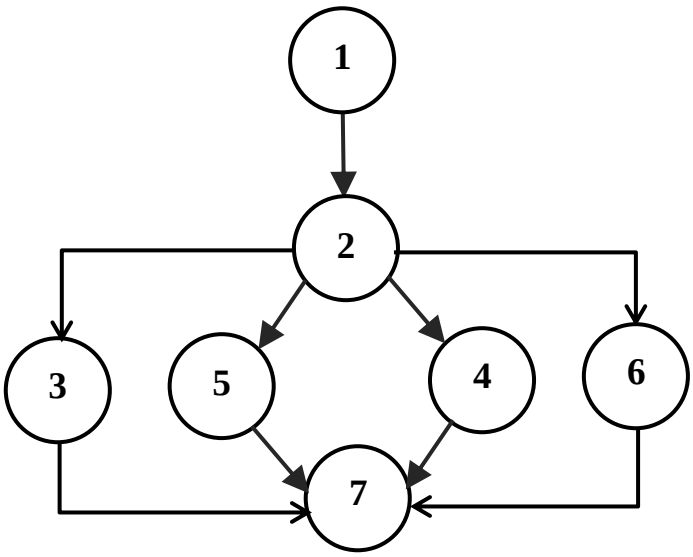


Рисунок Д.4 – **Граф управления с множественным выбором**

Программы с операторами цикла. Рассмотрим граф управления для программы на ЯВУ, вычисляющей сумму первых N членов натурального ряда:

$$S = \sum_{i=1}^{N} i, \qquad (Д.6)$$

Управляющий граф такой программы показан на рисунке Д.5. Для построения этого графа потока управления **необходимо заменить оператор цикла программы на эквивалентную последовательность операторов (с циклом DO)**. Среди них **должны быть операторы присваивания и вычисления условий**. Эти **операторы должны отразить три элемента, присущих любому циклу**:

1) начальные присваивания до выполнения цикла;

2) повторяющиеся действия (тело цикла);

3) условия окончания (или продолжения) цикла.

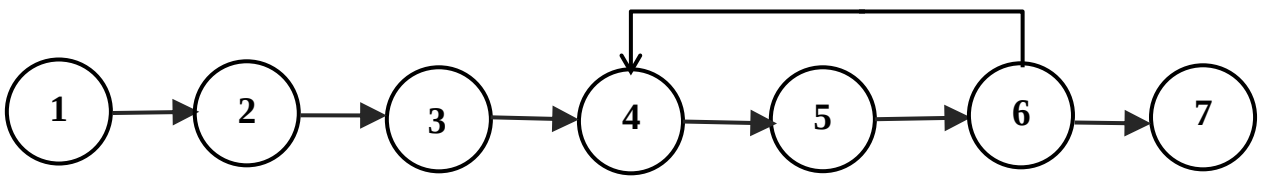


Рисунок Д.5 – **Управляющий граф программы с циклом без объединения вершин**

В графе программы **вершины с первой по третью соответствуют действиям, предшествующим циклу**, вершины **четыре и пять – группе опера-**

торов, составляющих тело цикла (множеству повторяемых действий), а **вершина шесть** – **действию, определяющему необходимость продолжения или прекращения выполнения цикла**.

Расчёт цикломатической сложности этой программы будет (Д.7):

$$Z(G) = e - v + 2 = 5 - 5 + 2 = 2, \quad (Д.7)$$

где количество дуг в графе равно 5, количество вершин 5.

Построение графа управления для программы с циклом WHILE показано на том же *примере* **вычисления суммы первых членов натурального ряда**. Так же как и в предыдущем случае (**с циклом DO**), **заменим оператор цикла эквивалентными операторами**.

На рисунке Д.6 приведен граф, в котором номера операторов указаны после их приведения к одной вершине. Объединять несколько операторов в группу и представить её одной вершиной можно в том случае, если это последовательность **линейных (следующих один за другим)** операторов без ветвления процедуры выполнения программы.

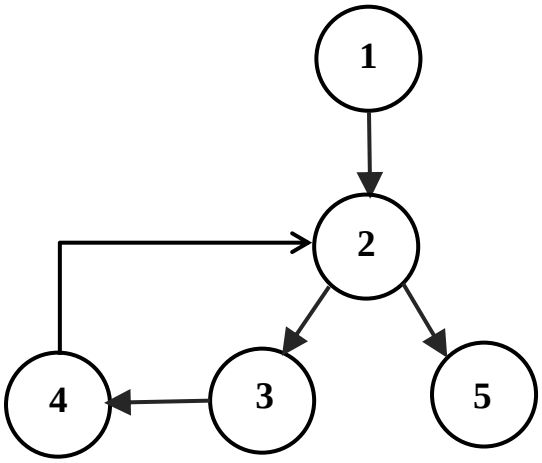


Рисунок Д.6 – **Граф управления программой с циклом WHILE**

Оценка цикломатической сложности программы будет (Д.8):

$$Z(G) = e - v + 2 = 5 - 5 + 2 = 2, \quad (Д.8)$$

где **количество дуг графа 5**, а **количество вершин 5**.

Построение графа управления для программы с циклом FOR показано на примере построения графа управления программой, вычисляющей число Фибоначчи по заданному номеру. Нужно учитывать, что **циклы могут иметь две реализации: с предусловием и постусловием.** Оператор **DO реализуется с постусловием**.

Ещё одна особенность состоит в том, что **циклическая структура обязательно содержит три элемента: действия до начала цикла, тело цикла** – повторяющиеся действия, **и условия окончания** (или продолжения) **цикла**.

Первый из элементов всегда выполняется первым.

Для циклов с предусловием вторым выполняется проверка, а **затем тело цикла.**

Циклы с постусловием реализованы иначе: **сначала выполняется тело цикла**, а **затем осуществляется проверка**. Так цикл **FOR** может быть реализован **как по схеме с предусловием, так и с постусловием**. Для средств реализации языков ***C***, ***C#*** и ***C++*** оператор **FOR** реализован по схеме с предусловием и его граф управления программой похож на аналогичную структуру для оператора цикла **WHILE**.

В *примере* графа управления программой, **вычисляющей число Фибоначчи** по заданному номеру (эти числа вычисляются для неотрицательных целых чисел), вычисление проводится по следующему соотношению:

$$f_0 = 0; \quad f_1 = 1; \quad \dots ; \quad f_i = f_{i-1} + f_{i-2}; \quad i \geq 2. \qquad (Д.9)$$

Для построения графа потока управления **надо представить программу в виде, отражающим замену цикла**. В итоге, граф потока управления для такой программы приведен на рисунке Д.7. В этом графе объединены линейные группы операторов.

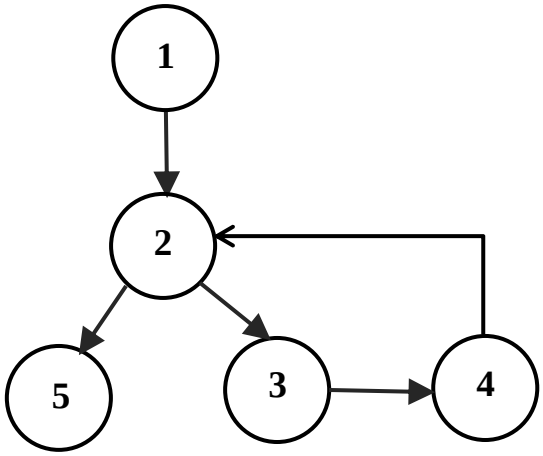


Рисунок Д.7 – **Граф управления программой с циклом FOR**

Более сложная задача – это вычисление суммы факториалов заданного

набора членов натурального ряда:

$$S = \sum_{i=1}^{n} i!. \qquad (Д.10)$$

Для реализации решения задачи **используется цикл FOR**. Фрагмент программы решения этой задачи на ЯВУ может быть таким, что **возможный управляющий граф программы будет выглядеть как показано на рисунке Д.8**. Следует обратить внимание, что **в этой программе есть вложенный цикл**.

Осуществив объединение линейных участков программы, которое позволит корректно построить граф потока управления (рисунок 8), получим **граф, где группа вершин в контуре с заливкой и штрих-пунктирными границами соответствуют вложенному циклу**.

Расчёт цикломатической сложности программы будет (Д.11):

$$Z(G) = e - v + 2 = 10 - 9 + 2 = 3, \qquad (Д.11)$$

где количество дуг в графе равно 10, количество вершин 9.

Граф управления программой с циклом **FOR** также **может быть реализован по схеме с постусловием** (для решения той же задачи). Для этого случая **граф будет похож на рассмотренный выше граф с оператором DO** (рассматривается задача вычисления суммы факториалов).

При построении графа управления нужно произвести соответствующую замену операторов цикла, и тогда граф будет выглядеть, как показано на рисунке Д.9.

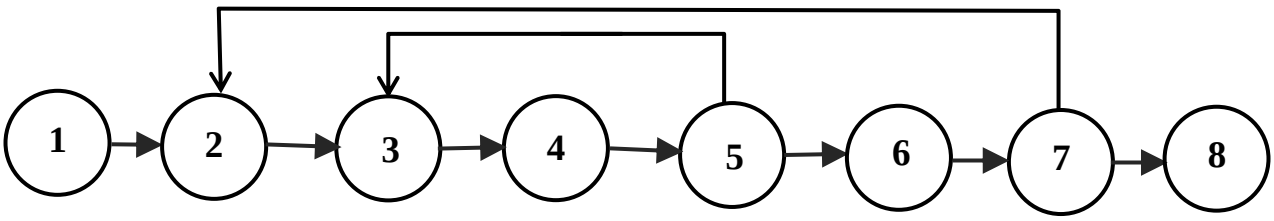


Рисунок Д.9 – **Управляющий граф для схемы с постусловием**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.701− 90. ЕСПД. Схемы алгоритмов, программ, данных и систем. – М.: Изд-во стандартов, 1990. – 8 с.

2. Черников Б.В., Поклонов Б.Е. Оценка качества программного обеспечения. Практикум. – М., ИД «Форум»–ИНФРА–М. – 2012. – 400 с.