Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Исследование алгоритмов поиска на графах

БГУИР КП I–40 01 01 323 ПЗ

Выполнил

студент гр. 051003 Павловец С.В.

Проверил: Фадеева Е.П.

Минск 2021

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2021г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Павловцу Сергею Валерьевичу*

1. Тема работы *Исследование алгоритмов поиска на графах*

2. Срок сдачи законченной работы *11.06.2021г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Среда программирования Delphi. Реализация функций создания и редактирования графов, сохранения созданных графов в удобном формате, открытия файлов с графами, сохраненными в формате программы. Возможность отображения графа на экране, импорта графа из матрицы расстояний в таблице Excel. Реализация функций демонстрации работы алгоритмов поиска на созданных графах*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*

*1 Анализ литературных источников*

*2 Постановка задачи*

*3 Разработка программного средства*

*4 Тестирование и проверка работоспособности программного средства*

*5 Руководство по установке и использованию программного средства*

*Заключение*

*Список использованных источников*

\_*Приложения*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*

6. Консультант по курсовой работе *Фадеева Е.П.*

7.Дата выдачи задания *16.02.2021г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

*Раздел 1. Введение к 03.03.2020г. – 10 % готовности работы;*

*Раздел 2 к 15.03.2020г. – 30% готовности работы*

*Раздел 3 к 15.04.2020г. – 60% готовности работы*

*Раздел 4 к 10.05.2020г. – 80% готовности работы*

*Раздел 5.Заключение. Приложения к 20.05.2020г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 01.06.2020г. – 100% готовности работы.*

*Защита курсового проекта с 02.06.2020г. по 11.06.2020г.*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фадеева Е.П.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Павловец С.В. 16.02.2021г.*

*(дата и подпись студента)*

Содержание

[Введение 7](#_Toc73268279)

[1 Анализ литературных источников 8](#_Toc73268280)

[1.1 Понятие графа и его представление в памяти компьютера 8](#_Toc73268281)

[1.2 Структура данных «однонаправленный список» 9](#_Toc73268282)

[1.3 Обходы графов в глубину и ширину 9](#_Toc73268283)

[1.4 Алгоритм Дейкстры для взвешенных графов 10](#_Toc73268284)

[1.5 Работа с текстовыми файлами 12](#_Toc73268285)

[2 Постановка задачи 14](#_Toc73268286)

[2.1 Описание предметной области 14](#_Toc73268287)

[2.2 Информационная база задачи 14](#_Toc73268288)

[2.3 Функциональное значение 14](#_Toc73268289)

[3 Разработка программного средства 15](#_Toc73268290)

[3.1 Описание алгоритмов 15](#_Toc73268291)

[3.2 Структура данных 21](#_Toc73268292)

[3.2.1 Структура типов программы 21](#_Toc73268293)

[3.2.2 Структура данных программы 23](#_Toc73268294)

[3.2.3 Структура данных алгоритма InitializeGraph 24](#_Toc73268295)

[3.2.4 Структура данных алгоритма DestroyGraph 24](#_Toc73268296)

[3.2.5 Структура данных алгоритма DestroyAdjList 24](#_Toc73268297)

[3.2.6 Структура данных алгоритма AddVertice 24](#_Toc73268298)

[3.2.7 Структура данных алгоритма AddArc 25](#_Toc73268299)

[3.2.8 Структура данных алгоритма DeleteVertice 25](#_Toc73268300)

[3.2.9 Структура данных алгоритма DeleteArc 26](#_Toc73268301)

[3.2.10 Структура данных алгоритма IsNeighbour 26](#_Toc73268302)

[3.2.11 Структура данных алгоритма GetByNumber 27](#_Toc73268303)

[3.2.12 Структура данных алгоритма GetByPoint 27](#_Toc73268304)

[3.2.13 Структура данных алгоритма OpenGraph 27](#_Toc73268305)

[3.2.14 Структура данных алгоритма SaveGraph 28](#_Toc73268306)

[3.2.15 Структура данных алгоритма ImportGraph 29](#_Toc73268307)

[3.2.16 Структура данных алгоритма ToWeightMatrix 30](#_Toc73268308)

[3.2.17 Структура данных алгоритма DFS 30](#_Toc73268309)

[3.2.18 Структура данных алгоритма BFS 31](#_Toc73268310)

[3.2.19 Структура данных алгоритма Dijkstra 32](#_Toc73268311)

[3.2.20 Структура данных алгоритма RestorePath 32](#_Toc73268312)

[3.2.21 Структура данных алгоритма InitializeStack 33](#_Toc73268313)

[3.2.22 Структура данных алгоритма InitializeQueue 33](#_Toc73268314)

[3.2.23 Структура данных алгоритма DestroyList 33](#_Toc73268315)

[3.2.24 Структура данных алгоритма Push 34](#_Toc73268316)

[3.2.25 Структура данных алгоритма Enqueue 34](#_Toc73268317)

[3.2.26 Структура данных алгоритма Pop 34](#_Toc73268318)

[3.2.27 Структура данных алгоритма Dequeue 35](#_Toc73268319)

[3.2.28 Структура данных алгоритма MakeVisited 35](#_Toc73268320)

[3.2.29 Структура данных алгоритма MakePassive 35](#_Toc73268321)

[3.2.30 Структура данных алгоритма MakeRegPolygon 36](#_Toc73268322)

[3.3 Схема алгоритмов решения задачи по ГОСТ 19.701-90 37](#_Toc73268323)

[3.3.1 Схема алгоритма AddVertice 37](#_Toc73268324)

[3.3.2 Схема алгоритма AddArc 38](#_Toc73268325)

[3.3.3 Схема алгоритма DeleteVertice 39](#_Toc73268326)

[3.3.4 Схема алгоритма DeleteArc 43](#_Toc73268327)

[3.3.5 Схема алгоритма IsNeigbour 45](#_Toc73268328)

[3.3.6 Схема алгоритма GetByNumber 46](#_Toc73268329)

[3.3.7 Схема алгоритма OpenGraph 47](#_Toc73268330)

[3.3.8 Схема алгоритма SaveGraph 50](#_Toc73268331)

[3.3.9 Схема алгоритма ToWeightMatrix 51](#_Toc73268332)

[3.3.10 Схема алгоритма DFS 52](#_Toc73268333)

[3.3.11 Схема алгоритма BFS 54](#_Toc73268334)

[3.3.12 Схема алгоритма Dijkstra 56](#_Toc73268335)

[3.3.13 Схема алгоритма RestorePath 59](#_Toc73268336)

[3.3.14 Схема алгоритма Push 60](#_Toc73268337)

[3.3.15 Схема алгоритма Enqueue 61](#_Toc73268338)

[3.3.16 Схема алгоритма Pop 62](#_Toc73268339)

[3.3.17 Схема алгоритма Pop 63](#_Toc73268340)

[3.4 Графический интерфейс 64](#_Toc73268341)

[3.4.1 Описание графических компонентов формы fmEditor 64](#_Toc73268342)

[3.4.2 Описание графических компонентов формы fmArcInput 66](#_Toc73268343)

[3.4.3 Описание графических компонентов формы fmSearchOutput 66](#_Toc73268344)

[3.4.4 Описание графических компонентов формы fmAbout 67](#_Toc73268345)

[Тестирование и проверка работоспособности программного средства 69](#_Toc73268346)

[4.1 Запуск программы 70](#_Toc73268347)

[4.1.1 Тест 1 70](#_Toc73268348)

[4.2 Добавление элементов 71](#_Toc73268349)

[4.2.1 Тест 2 71](#_Toc73268350)

[4.2.2 Тест 3 72](#_Toc73268351)

[4.2.3 Тест 4 72](#_Toc73268352)

[4.2.4 Тест 5 73](#_Toc73268353)

[4.2.5 Тест 6 73](#_Toc73268354)

[4.2.6 Тест 7 74](#_Toc73268355)

[4.3 Удаление элементов 74](#_Toc73268356)

[4.3.1 Тест 8 74](#_Toc73268357)

[4.3.2 Тест 9 75](#_Toc73268358)

[4.3.3 Тест 10 75](#_Toc73268359)

[4.3.4 Тест 11 76](#_Toc73268360)

[4.4 Дополнительные функции приложения 76](#_Toc73268361)

[4.4.1 Тест 12 76](#_Toc73268362)

[4.4.2 Тест 13 77](#_Toc73268363)

[4.4.3 Тест 14 77](#_Toc73268364)

[4.4.4 Тест 15 78](#_Toc73268365)

[4.4.5 Тест 16 78](#_Toc73268366)

[4.4.6 Тест 17 79](#_Toc73268367)

[4.4.7 Тест 18 79](#_Toc73268368)

[4.4.8 Тест 19 80](#_Toc73268369)

[4.4.9 Тест 20 80](#_Toc73268370)

[4.5 Поиск в глубину 81](#_Toc73268371)

[4.5.1 Тест 21 81](#_Toc73268372)

[4.5.2 Тест 22 81](#_Toc73268373)

[4.5.3 Тест 23 82](#_Toc73268374)

[4.5.4 Тест 24 82](#_Toc73268375)

[4.5.5 Тест 25 83](#_Toc73268376)

[4.6 Поиск в ширину 83](#_Toc73268377)

[4.6.1 Тест 26 83](#_Toc73268378)

[4.6.2 Тест 27 84](#_Toc73268379)

[4.6.3 Тест 28 84](#_Toc73268380)

[4.6.4 Тест 29 85](#_Toc73268381)

[4.6.5 Тест 30 85](#_Toc73268382)

[4.7 Алгоритм Дейкстры 86](#_Toc73268383)

[4.7.1 Тест 31 86](#_Toc73268384)

[4.7.2 Тест 32 86](#_Toc73268385)

[4.7.3 Тест 33 87](#_Toc73268386)

[4.7.4 Тест 34 87](#_Toc73268387)

[4.7.5 Тест 35 88](#_Toc73268388)

[5 Тестирование работоспособности программного средства 89](#_Toc73268389)

[5.1 Минимальные системные требования 89](#_Toc73268390)

[5.2 Установка 89](#_Toc73268391)

[5.3 Работа с приложением 93](#_Toc73268392)

[Заключение 97](#_Toc73268393)

[Список использованной литературы 98](#_Toc73268394)

[Приложение А 99](#_Toc73268395)

[Приложение Б 109](#_Toc73268396)

[Приложение В 111](#_Toc73268397)

[Приложение Г 113](#_Toc73268398)

[Приложение Д 115](#_Toc73268399)

[Приложение Е 127](#_Toc73268400)

[Приложение Ж 135](#_Toc73268401)

[Приложение З 139](#_Toc73268402)

# Введение

Данный курсовой проект посвящен исследованию алгоритмов поиска на графах.

Теория графов – раздел дискретной математики, изучающий такие математические объекты, как графы. Её родоначальником считается Леонард Эйлер, который в 1736 году формулирует и предлагает решение задачи о семи кёнигсбергских мостах. Сам термин «граф» был предложен в 1878 году Сильвестром Джеймсом Джозефом. Теория графов развивалась в процессе решения загадок и головоломок, так как графы наглядно представляются графически с помощью фигур и соединяющих их линий. И хотя долгое время она не выделялась как отдельная математическая дисциплина, но в 20-21-ые столетия графы начались использоваться во многих сферах будь то гуманитарные науки или технические. Графы также активно применяются в программировании для описания различных алгоритмов, структур данных, синтаксиса языков и т.д.

Важной частью теории графов являются задачи поиска. К примеру, согласно формулировке задачи, может быть необходимо найти некую вершину, начиная просмотр из заданной, или же некий путь, удовлетворяющий определённым условиям, или какой-либо подграф заданного графа, обладающий указанными свойствами.

Программное средство предназначено для создания и редактирования графов, а также для демонстрации некоторых алгоритмов поиска.

Целью работы является создание проекта, сопровождающегося документацией в виде пояснительной записки.

В реализации проекта попутно решаются следующие задачи:

* создание динамических структур;
* работа с файлами (текстовыми, типизированными);
* чтение/запись данных из файла;
* разработка пользовательского интерфейса для реализации перечисленных функций;
* представление содержимого файла для работы в виде динамической структуры: однонаправленного списка.

# Анализ литературных источников

## Понятие графа и его представление в памяти компьютера

Простым (n, m)-графом называется пара конечных множеств вершин и рёбер таких, что множество вершин состоит из n>0 элементов, а множество рёбер - из m элементов, каждый из которых является неупорядоченной парой из вершин. Если же помимо перечисленных условий каждому ребру ставится в соответствие некое число (вес), то такие множества называют взвешенным графом. Существует несколько способов задания графов в памяти компьютера.

Первый из них – матрица смежности. У неё номерам столбцов и строк соответствуют номера вершин графа. В каждой ячейке записывается число, определяющее смежность вершин: если вершины смежны, то элемент равен 1, иначе – 0. Для взвешенных графов вместо матрицы смежности используется матрица весов, где вместо нулей и единиц записывают вес ребра, если же ребра не существует, то в элемент записывают 0 или ∞ в зависимости от задачи. Подобный способ представления занимает O(n2) места в памяти.

Следующий способ – списками смежности. В этом случае граф представляет собой множество окрестностей всех вершин. Окрестность вершины, в свою очередь, представляется множеством смежных с ней вершин. Для взвешенного графа вместе с номером смежной вершины хранится вес соответствующего ребра. Таким образом, подобная структура занимает места в памяти.

Видно, что для графов с относительно небольшим количеством рёбер (их называют разреженными) меньше места в памяти занимает второй способ представления. Если же рёбер много (такие графы называют плотными), в частности для полного графа, у которого , списки смежности могут занимать больше памяти, чем матрица смежности. Стоит также учесть, что временная сложность проверки смежности вершин в матрице смежности составляет O(1), а в списках смежности – O(n).

Существуют и такие способы задания графов, как матрица инцидентности, строки которой соответствуют вершинам, а столбцы – рёбрам, и список рёбер, который является перечислением каждого ребра графа. Однако, они реже используются для решения задач на графах: матрица инцидентности имеет размер O(n\*m), а временная сложность проверки смежности вершин в списке рёбер – O(m), что хуже, чем у списка смежности для плотных графов. Более того, список рёбер не полностью описывает графы с изолированными вершинами, поэтому чтобы необходимо хранить дополнительный список вершин.

## Структура данных «однонаправленный список»

Списки смежности можно хранить, используя два вложенных однонаправленных списка. Однонаправленный список – структура данных, состоящая из элементов, последовательно связанных между собой с помощью указателей. Каждый элемент списка имеет указатель на следующий элемент, а последний элемент списка указывает на nil. В подобном списке можно передвигаться только в сторону его конца: узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.

## Обходы графов в глубину и ширину

Обходом графа называется процесс посещения каждой вершины графа. Результатом обхода может служить новая нумерация вершин в графе. Существуют два варианта обхода графа: в глубину и в ширину, отличающиеся порядком посещения вершин. В любом случае процесс начинается из указанной заранее вершины.

Обход в глубину использует метод поиска с возвращением, при котором после посещения очередной вершины поиск может либо продолжиться c переходом к смежной вершине, либо произойти откат к предыдущей рассмотренной вершине, если у рассматриваемой не осталось смежных с ней и не посещённых ранее вершин.

Противоположный метод обхода – обход в ширину. В этом случае у очередной вершины сначала посещаются вершины из её окрестности, после чего процесс переходит к окрестностям этих вершин.

Обходы могут описываться в терминах рекурсии, но для итеративной реализации потребуются такие структуры данных как стек и очередь. При посещении новой вершины обходом в глубину её добавляют в стек, а при необходимости отката извлекают последнюю добавленную вершину из стека. Обратный процесс происходит для поиска в ширину: для посещения очередной вершины её сначала извлекают из начала очереди, а после посещения и перед переходом к следующей вершине все вершины из её окрестности помещают в конец очередь. Обходы завершаются в том случае, когда посещены все вершины, иначе говоря, когда стек или очередь станут пусты.

Частным случаем обхода графа в глубину и ширину является обход дерева. Деревом называют граф, не содержащим циклов. Это свойство позволяет быть уверенным, что при обходе никакая из вершин не будет посещена дважды, однако в графах с циклами неизвестно заранее, была ли посещена вершина ранее или нет из-за чего обход может замедлиться или зациклиться. Чтобы избежать этого, вершины после посещения «закрашивают» в некий цвет и перед посещением следующей проверяют цвет вершины. В программе это означает, что необходимо дополнительно хранить массив флагов размером, равным количеству вершин, и изменять значения флагов при посещении очередной.

На деревьях удобно изображать процесс обхода графа. К тому же, понятно их название: при обходе в глубину процесс «спускается всё ниже», пока не дойдёт до самой отдалённой от корня, «глубокой», вершины, а при обходе в ширину процесс идёт «поярусно», посещая все вершины, одинаково удалённые от корня.

Что касается временной сложности, оба алгоритма имеют одну временную сложность, равную O(n+m).

Для работы алгоритмов необходимы стек или очередь, хранящие от 0 до n вершин графов и n флагов для хранения посещённости вершин, тогда пространственная сложность алгоритмов составляет O(n).

Оба алгоритма одинаково сложны, но для разных видов графов результаты могут отличаться: в графах, где много «тупиков» произойдёт много откатов, а в графе простой цикл откаты будут происходить один за другим после того, как все вершины уже будут помечены, и в задачах поиска могут не происходить, если вершина была найдена.

Обходы в глубину и ширину используют для решения многих задач:

* проверка связности вершин;
* проверка графа на ацикличность;
* проверка графа на двудольность;
* топологическая сортировка;
* выделение связных компонент;
* поиск точек сочленения, мостов;
* и т.д.

Главное отличие алгоритмов состоит в следующем свойстве обхода в ширину: вершина в каждом «ярусе» удалена от начальной на столько же рёбер, каков номер этого «яруса». Таким образом, алгоритм поиска в ширину можно использовать для эффективного нахождения расстояний между вершинами в простых графах. При движении «в глубину» для этого приходилось бы перебирать все простые пути между двумя вершинами.

К тому же алгоритмы используются и для задач, которые можно сформулировать в терминах графов. К примеру, область применения обхода в ширину – поиск в социальных сетях, нахождение соседних узлов в одноранговых сетях вроде BitTorrent, поисковые системы для построения индексов веб-страниц. А наиболее известное применение обхода в глубину – решения головоломок с одним решением вроде лабиринта.

В графах с ориентированными рёбрами (орграфы), петлями (псевдографы) и кратными рёбрами (мультиграфы) алгоритмы работают аналогично.

## Алгоритм Дейкстры для взвешенных графов

Алгоритм был изобретён нидерландским учёным Эдсгером Дейкстрой в 1959 году и используется для поиска кратчайшего расстояния между двумя вершинами во взвешенном графе с неотрицательными весами. При этом с помощью некоторых модификаций возможно сохранение кратчайшего пути, а также нахождение всех расстояний и кратчайших путей из заданной вершины до остальных. Алгоритм можно использовать для простых графов, орграфов, псевдографов и мультиграфов.

Важными понятиями для алгоритма являются понятия временных и постоянных меток, а также процесса релаксации.

Временная метка вершины - верхняя оценка расстояния между текущей вершиной и начальной. Она вычисляется с помощью релаксаций рёбер, инцидентных с вершиной, у которой рассматривается постоянная метка.

Постоянная метка вершины - расстояние между текущей вершиной и начальной. На каждом шаге алгоритма для одной вершины выбирается постоянная метка как минимальная из временных меток.

Изначально, метка начальной вершины считается постоянной и равна 0, а для остальных вершин временным меткам присваивается максимально допустимое значение.

Релаксация по ребру (i, j) заключается в следующем: из найденной временной метки вершины j и суммы текущей постоянной метки вершины i с весом ребра (i, j) выбирается минимальное значение и сохраняется во временную метку j. Таким образом, релаксация - процесс уменьшения временных меток.

Тогда суть алгоритма заключается в следующем: на очередном шаге находится постоянная метка и для соответствующей ей вершины происходят релаксации по инцидентным данной вершине рёбрам. Если же постоянной метке соответствовала искомая вершина, то процесс завершается.

Если необходимо найти сам кратчайший путь, то для сохранения пути необходимо на каждом шаге записывать вершину, соответствующую постоянной метке, в конец уже найденного пути и по завершении цикла записать в конец пути искомую вершину.

Метод схож с поиском в ширину в том смысле, что исследуется окрестность текущей вершины, после чего вершина помечается как посещённая (постоянная метка и закрашенные вершины). При этом вместо добавления в очередь не закрашенных вершин, происходят релаксации по рёбрам, инцидентным этой вершине, а вместо извлечения вершины из начала очереди выбирается вершина с минимальной временной меткой.

Если искомая вершина самая удалённая, то происходит n выборов постоянных меток, а также происходит m релаксаций. В зависимости от используемых структур данных для хранения меток сложность релаксации и выбора постоянной метки могут разниться. В случае массива: релаксация происходит за O(1), а поиск минимума за O(n), тогда в худшем случае временная сложность алгоритма - O(n2+m). Для алгоритма могут быть применены такие древовидные структуры как двоичная и фибоначиева кучи.

Алгоритм является жадным, то есть на каждом шаге принимает наиболее оптимальное решение для текущего шага, которое в дальнейшем не пересматривается.

Таким образом, после выбора постоянной пометки вершины предполагается, что любой другой путь приведёт к большей длине, чем рассмотренный на текущем шаге. Однако в графах с отрицательными весами это не всегда верно: даже одно ребро не рассмотренного пути может уменьшить расстояние так, что оно станет меньше, чем постоянная метка.

К тому же, в графах с отрицательными весами кратчайших путей может не существовать вовсе, так как дополнительные проходы по контуру с отрицательным суммарным весом уменьшают кратчайший путь.

Таким образом, для поиска кратчайших путей в графах с отрицательными весами используют другие алгоритмы, к примеру, алгоритмы Беллмана-Форда и Флойда-Уоршелла.

Поиск кратчайшего пути может использоваться для построения маршрута в приложениях вроде Google Maps, при создании компьютерных сетей для обеспечения минимальной задержки, в абстрактных автоматах для определения оптимальной стратегии и т.д.

## Работа с текстовыми файлами

Файловый тип – произвольная последовательность элементов, длина которой заранее не определена, а конкретизируется в процессе выполнения программы. Это определение логического файла, т.е. того, который используется в программе. Физический файл – это поименованная область памяти на внешнем носителе, в которой хранится некоторая информация.

Всего существует три типа файлов:

* типизированные файлы;
* текстовые файлы;
* нетипизированные файлы.

Типизированные файлы связываются с файловыми переменными, объявленными как file of <Тип>. Файл считается состоящим из элементов, каждый из которых имеет тип <Тип>.

Нетипизированные файлы могут быть связаны только с теми файловыми переменными, которые были объявлены как file. Файл считается состоящим из элементов, размер которых определяется при открытии файла.

Текстовый файл представляет собой последовательность символов, однако он не эквивалентен файлу типа file of Char. Текстовые файлы связываются с файловыми переменными, принадлежащими стандартному типу TextFile. Особенность текстовых файлов состоит в том, что содержащиеся в них символы разбиваются на строки. Строки имеют различную длину, а в конце каждой помещается специальный управляющий символ: возврат каретки и символ перехода на новую строку.

Для доступа к отдельным элементам в Delphi существуют специальные стандартные процедуры и функции. Их называют процедурами и функциями ввода-вывода.

Процедура AssignFile (F, Name) связывает файловую переменную с внешним файлом на диске. Здесь F – имя файловой переменной любого типа; Name – выражение строкового типа. Назначение процедуры: организует связь между конкретным физическим файлом на внешнем устройстве (конкретным набором данных) и файловой переменной (логическим файлом).

Процедура Reset (F) открывает существующий файл F. При этом открывается внешний файл с именем, присвоенным переменной F процедурой AssignFile. Если файла не существует, возникает сообщение об ошибке. Указатель файла устанавливается на первую позицию файла. Если файл был предварительно открыт, то он закрывается и повторно открывается. При выполнении процедуры содержимое файла не изменяется. Для текстовых файлов файл открывается в режиме только для чтения (read-only).

Процедура Rewrite (F) создает новый файл и открывает его. Если файл уже открыт, то закрывает и снова открывает его. Указатель файла устанавливается на начало файла (файл создается пустым). Для текстовых файлов – в режиме только для записи (write-only).

Процедура Append (F) предопределена только для текстовых файлов. Она открывает существующий файл для добавления. Если файл уже открыт, то закрывает его, а затем вновь открывает. В данном случае указатель файла устанавливается на конец файла. Файл открывается в режиме только для записи.

Для закрытия файла используется процедура CloseFile (F), где F – файловая переменная, открытая с помощью Reset, Rewrite или Append.

Функции Eof (F) и Eol(F) проверяют на конец файла или символ конца строки соответственно.

Для файлов с типом предопределены процедуры считывания компонентов файла в переменные и записи в файл компонентов из переменных Read (F, V1, V2, …, Vn) и Write (F, V1, V2, …, Vn).

Для файлов без типа есть аналоги процедур Read и Write. Это процедуры BlockRead (F, Buf, Count [, Done]) и BlockWrite (F, Buf, Count [, Done]).

Также есть функции и процедуры, перемещающие указатель файла в указанную позицию, возвращающие текущее положение указателя файла в байтах, возвращающие размер файла в компонентах и др.

# Постановка задачи

## Описание предметной области

## Информационная база задачи

В основу принципа работы редактора графов положено взаимодействие пользователя и программного модуля, обеспечиваемое GUI, реализованным в виде форм и окон.

В качестве входных данных пользователь может добавлять и удалять элементы графа: вершины и дуги (как взвешенные, так и нет), и выбрать алгоритм поиска одной одной вершины, начиная из другой.

Выходные данные: последовательность вершин, задающих путь, а также дополнительная информация: количество дуг в пути, длина пути в единицах веса и число посещений вершин при поиске.

## Функциональное значение

Функции, выполняемые программным средством:

* добавление и удаление вершины;
* добавление и удаление ребра;
* ввод веса дуги;
* возможность перемещения вершины;
* загрузка и сохранение графа из типизированного файла;
* импорт графа из файлов формата xlsx, xls, xlsb, xlsm;
* поиск пути между вершинами с помощью обхода в глубину;
* поиск пути между вершинами с помощью обхода в ширину;
* поиск пути между вершинами с помощью алгоритма Дейкстры;
* возможность удаления созданного графа;
* возможность чтения файла с информацией о программе.

Таким образом, программное средство будет представлять собой редактор графов, поддерживающая взвешенные орграфы с положительными весами без петель и кратных рёбер.

Для хранения графа будет использована такая структура данных, которая представляет собой список списков смежности вершин.

Для чтения и записи в типизированные файлы будут использованы 2 вида типизированных файлов: для вершин и рёбер.

# Разработка программного средства

## Описание алгоритмов

В таблице 1 представлены краткие описания основных алгоритмов, используемых для реализации программы.

Таблица 1 – Описание алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п.п. | Наименование алгоритма | Назначение алгоритма | Формальные параметры | Предлагаемый тип реализации |
| 1 | Инициализация главной формы | Загрузка настроек программы. Инициализация графа, установка режима работы с холстом,  инициализация активной вершины |  |  |
| 2 | InitializeGraph  (Graph) | Инициализация графа Graph | Graph – получает от фактического параметра адрес | Процедура |
| 3 | DestroyGraph  (Graph) | Освобождение динамической памяти, выделенной для графа Graph | Graph – получает от фактического параметра адрес | Процедура |
| 4 | DestroyAdjList  (Head) | Освобождение динамической памяти, выделенной для списка смежности с головой Head и её обнуление | Head – получает от фактического параметра адрес | Процедура |
| 5 | AddVertice  (Graph, C) | Создание и добавление в конец списка вершин графа Graph новой вершины с центром в точке C | Graph – получает от фактического параметра адрес;  C – получает от фактического параметра значение | Процедура |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | AddArc  (Graph, v, u, w) | Создание и добавление в начало списка смежности графа Graph вершины v соседа u с инцидентной дугой веса w | Graph – получает от фактического параметра адрес;  v, u, w – получают от фактического параметра значение | Процедура |
| 7 | DeleteVertice  (Graph, v) | Удаление и освобождение памяти, выделенной для вершины v из графа Graph, вместе со всеми инцидентными дугами, и уменьшение номеров вершин, которые больше v | Graph – получает от фактического параметра адрес;  v – получает от фактического параметра значение | Процедура |
| 8 | DeleteArc  (Graph, v, u) | Удаление и освобождение памяти, выделенной для соседа u вершины v графа Graph | Graph – получает от фактического параметра адрес;  v, u – получают от фактического параметра значение | Процедура |
| 9 | IsNeighbour  (Graph,  Vertice,  u,  Res) | Проверка принадлежности вершины u соседям вершины со звеном Vertice в графе Graph | Graph – получает от фактического параметра адрес с защитой;  Vertice – получает от фактического параметра значение;  u – получает от фактического параметра значение;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | GetByNumber  (Graph, v, Res) | Получение звена графа Graph с номером вершины v | Graph – получает от фактического параметра адрес с защитой;  v – получает от фактического параметра значение;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 11 | GetByPoint  (Graph, P, Res) | Получение звена графа Graph с помощью проверки принадлежности точки P геометрическому месту точек вершины | Graph – получает от фактического параметра адрес с защитой;  P – получает от фактического параметра значение;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 12 | OpenGraph  (Graph,  VerFileName,  ArcFileName) | Чтение графа Graph из типизированных файлов вершин и дуг с именами VerFileName и ArcFileName соответственно | Graph – получает от фактического параметра адрес;  VerFileName, ArcFileName – получают от фактического параметра значение; | Процедура |
| 13 | SaveGraph  (Graph,  VerFileName,  ArcFileName) | Запись графа Graph в типизированные файлы вершин и дуг с именами VerFileName и ArcFileName соответственно | G – получает от фактического параметра адрес с защитой;  VerFileName, ArcFileName – получают от фактического параметра значение; | Процедура |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | ImportGraph  (Graph,  ExcelFileName) | Импорт графа Graph из электронной таблицы с помощью Excel с именем FileName | Graph – получает от фактического параметра адрес;  ExcelFileName – получает от фактического параметра значение | Процедура |
| 15 | ToWeightMatrix  (Graph, Res) | Преобразование графа Graph из списков смежности в матрицу весов Res | G – получает от фактического параметра адрес с защитой;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 16 | DFS  (Graph,  Src,  Dest,  Res) | Обход графа Graph в глубину от вершины Src до вершины Dest с информацией об обходе в Res | G – получает от фактического параметра адрес с защитой;  Src, Dest – получают от фактического параметра значение;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 17 | BFS  (Graph,  Src,  Dest,  Res) | Обход графа Graph в ширину от вершины Src до вершины Dest с информацией об обходе в Res | Graph – получает от фактического параметра адрес с защитой;  Src, Dest – получают от фактического параметра значение;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 18 | Dijkstra  (Graph,  Src,  Dest,  Res) | Поиск алгоритмом Дейкстры пути в графе Graph от вершины Src до вершины Dest с информацией о поиске в Res | Graph – получает от фактического параметра адрес с защитой;  Src, Dest – получают от фактического параметра значение;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 19 | RestorePath  (Graph,  Parents,  Src,  Dest,  Res) | Восстановление пути от Src к Dest в графе Graph по массиву предков Parents | Graph, Parents – получают от фактического параметра адрес с защитой;  Src, Dest – получают от фактического параметра значение;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 20 | InitializeStack  (Stack) | Инициализация стека Stack | Stack – получает от фактического параметра адрес | Процедура |
| 21 | InitializeQueue  (Queue) | Инициализация очереди Queue | Queue – получает от фактического параметра адрес | Процедура |
| 22 | DestroyList  (Head) | Освобождение памяти, выделенной для списка с головой Head и её обнуление | Head – получает от фактического параметра адрес | Процедура |
| 23 | Push  (Stack, n) | Вставка в стек Stack числа n | Stack – получает от фактического параметра адрес;  n – получает от фактического параметра значение | Процедура |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 24 | Enqueue  (Queue, n) | Добавление в очередь Queue числа n | Queue – получает от фактического параметра адрес;  n – получает от фактического параметра значение | Процедура |
| 25 | Pop  (Stack, Res) | Извлечение числа из стека Stack в Res | Stack – получает от фактического параметра адрес;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 26 | Dequeue  (Queue, Res) | Извлечение числа из очереди Queue в Res | Queue – получает от фактического параметра адрес;  Res – получает от фактического параметра адрес | Функция  (Res – возвращаемый функцией параметр) |
| 27 | MakeVisited  (G, Path) | Изменение стиля раскраски вершин и дуг из пути Path графа Graph на посещённые | Graph – получает от фактического параметра адрес;  Path – получает от фактического параметра адрес | Процедура |
| 28 | MakePassive  (Graph) | Изменение стиля раскраски всех вершин и дуг графа Graph на изначальный | Graph – получает от фактического параметра адрес | Процедура |
| 30 | MakeRegPolygon  (Graph,  Width,  Height) | Изменение местоположение всех вершин графа Graph в вершины правильного многоугольника на холсте | Graph – получает от фактического параметра адрес;  Width, Height – получают от фактического параметра значение | Процедура |

## Структура данных

### Структура типов программы

Таблица 2 – Структура типов программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| TVerticeStyle | (stPassive = 0, stActive, stVisited) | Стиль раскраски вершины графа | |
| TPNeighbour | ^TNeighbour | Указатель на звено списка смежности | |
| TNeighbour | record  Number: Integer;  Weight: Integer;  isVisited: Boolean;  Next: TPNeighbour;  end; | Звено списка смежности;  Number – номер соседа вершины:  Weight – вес дуги, ведущей к соседу;  isVisited – флаг о посещённости дуги, ведущей к соседу, при поиске пути;  Next – ссылка на следующее звено списка | |
| TPVertice | ^TVertice | Указатель на звено списка вершин | |
| TVertice | record  Number: Integer;  Center: TPoint;  OutDeg: Integer;  Style: TVerticeStyle;  Head: TPNeighbour;  Next: TPVertice;  end; | Звено списка вершин:  Number – номер вершины;  Center – геометрический центр вершины;  OutDeg – полустепень исхода вершины;  Style – стиль раскраски вершины графа;  Head – ссылка на головной элемент списка смежности вершины;  Next – ссылка на следующее звено списка вершин | |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| TGraph | record  Head: TPVertice;  Tail: TPVertice;  Order: Integer;  isPainted: Boolean;  R: Integer;  end; | Взвешенный орграф без петель, кратных рёбер и отрицательных весов:  Head – ссылка на голову списка вершин;  Tail – ссылка на хвост списка вершин;  Order – порядок графа;  isPainted – флаг об изменении цветов у элементов графа;  R – радиус окружности, соответствующий вершине на холсте |
| TWeightMatrix | Array of Array of Integer | Матрица весов графа |
| TSearchInfo | record  Path: TStack;  PathSring: String;  ArcsCount: Integer;  Distance: Integer;  VisitsCount: Integer;  end; | Путь с информацией:  Path – список вершин в пути;  PathString – строка с вершинами пути через разделители;  ArcsCount – число дуг в пути;  Distance – длина пути в единицах веса;  Visits – число посещений при поиске |
| TPItem | ^TItem | Ссылка на звено списка натуральных чисел |
| TItem | record  Number: Integer;  Next: TPItem;  end; | Звено списка натуральных чисел:  Number – число в списке;  Next – ссылка на следующее звено списка |
| TStack | TPItem | Стек натуральных чисел |
| TQueue | record  Head: TPItem;  Tail: TPItem;  end; | Очередь натуральных чисел:  Head – голова очереди;  Tail – хвост очереди |

### Структура данных программы

Таблица 3 – Структура данных программы

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Комментарии |
| fmEditor | TfmEditor | Форма, предоставляющая графического интерфейса для редактирования созданного графа и функции чтение, сохранения, импорта, экспорта графа в файлы.  Содержит редактируемый граф, выделенную вершину и переменную состояния для работы с графом как поля | Главная форма программы |
| fmArcInput | TfmArcInput | Форма для ввода веса добавляемой дуги.  Содержит вес дуги как поле | Вспомогательная форма программы |
| fmSearchOutput | TfmSearchOutput | Форма для вывода информации о найденном пути.  Содержит информацию о пути как поле | Вспомогательная форма программы |
| fmAbout | TfmAbout | Форма для вывода информации о программе | Вспомогательная форма программы |
| INF | Integer | Большое целое число для использование в алгоритме Дейкстры и инициализации отсутствия связи в матрице расстояний |  |

### Структура данных алгоритма InitializeGraph

Таблица 4 – Структура данных алгоритма InitializeGraph (Graph)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Инициализируемый граф | Формальный |

### Структура данных алгоритма DestroyGraph

Таблица 5 – Структура данных алгоритма DestroyGraph (Graph)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Граф для освобождения памяти | Формальный |
| Vertice | TPVertice | Ссылка на удаляемую вершину из списка вершин | Локальный |

### Структура данных алгоритма DestroyAdjList

Таблица 6 – Структура данных алгоритма DestroyAdjList (Head)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Head | TPNeighbour | Ссылка на голову списка смежности | Формальный |
| Neighbour | TPNeighbour | Ссылка на удаляемого соседа | Локальный |

### Структура данных алгоритма AddVertice

Таблица 7 – Структура данных алгоритма AddVertice (Graph, C)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Граф, в который добавляется вершина | Формальный |
| C | TPoint | Геометрическое место центра вершины | Формальный |
| Vertice | TPVertice | Ссылка на добавляемую вершину графа | Локальный |

### Структура данных алгоритма AddArc

Таблица 8 – Структура данных алгоритма AddArc (Graph, v, u, w)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Граф для добавления дуги (v, u) | Формальный |
| v | Integer | Начало дуги | Формальный |
| u | Integer | Конец дуги | Формальный |
| w | Integer | Вес дуги | Формальный |
| Vertice | TPVertice | Ссылка на вершину с номером v | Локальный |
| AdjVertice | TPVertice | Ссылка на вершину с номером u | Локальный |
| Neighbour | TPNeighbour | Ссылка на добавляемого соседа | Локальный |
| isIncorrect | Boolean | Флаг об отсутствии кратных дуг и петель в случае добавления дуги | Локальный |

### Структура данных алгоритма DeleteVertice

Таблица 9 – Структура данных алгоритма DeleteVertice (Graph, v)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Граф, из которого удаляется вершина v | Формальный |
| v | Integer | Номер удаляемой вершины | Формальный |
| Vertice | TPVertice | Ссылка на текущую вершину | Локальный |
| PrVertice | TPVertice | Ссылка на вершину перед Vertice | Локальный |
| Neighbour | TPNeighbour | Ссылка на текущего соседа вершины | Локальный |
| PrNeighbour | TPNeighbour | Ссылка на соседа перед Neighbour | Локальный |
| Exists | Boolean | Флаг о наличии вершины | Локальный |

### Структура данных алгоритма DeleteArc

Таблица 10 – Структура данных алгоритма DeleteArc (Graph, v, u)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Граф, из которого удаляется дуга (v, u) | Формальный |
| v | Integer | Номер вершины-начала дуги | Формальный |
| u | Integer | Номер вершины-конца дуги | Формальный |
| Vertice | TPVertice | Ссылка на вершину-начало дуги | Локальный |
| Neighbour | TPNeighbour | Ссылка на искомого удаляемого соседа вершины | Локальный |
| PrNeighbour | TPNeighbour | Ссылка на соседа в списке смежности перед Neighbour | Локальный |
| Exists | Boolean | Флаг о наличии дуги | Локальный |

### Структура данных алгоритма IsNeighbour

Таблица 11 – Структура данных алгоритма IsNeighbour (Graph, Vertice, u, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Полученный граф | Формальный |
| Vertice | TPVertice | Ссылка на вершину-начало дуги | Формальный |
| u | Integer | Номер вершины-конца дуги | Формальный |
| Res | Boolean | Результат проверки | Формальный |
| Neighbour | TPNeighbour | Ссылка на сосед с номером u | Локальный |

### Структура данных алгоритма GetByNumber

Таблица 12 – Структура данных алгоритма GetByNumber (Graph, v, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Полученный граф | Формальный |
| v | Integer | Номер вершины для поиска | Формальный |
| Res | TPVertice | Ссылка на искомое звено списка вершин | Формальный |

### Структура данных алгоритма GetByPoint

Таблица 13 – Структура данных алгоритма GetByPoint (Graph, P, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Полученный граф | Формальный |
| P | TPoint | Переданная координата холста | Формальный |
| Res | TPVertice | Ссылка на искомое звено списка вершин | Формальный |
| Vertice | TPVertice | Ссылка на текущую вершину графа | Локальный |

### Структура данных алгоритма OpenGraph

Таблица 14 – Структура данных алгоритма OpenGraph (Graph, VerFileName, ArcFileName)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Неинициализированный граф для чтения из файлов | Формальный |
| VerFileName | String | Имя файла с информацией о вершинах графа | Формальный |
| ArcFileName | String | Имя файла с информацией о дугах графа | Формальный |

Продолжение таблицы 14

|  |  |  |  |
| --- | --- | --- | --- |
| VerFile | File of TVertice | Типизированный файл вершин | Локальный |
| ArcFile | File of TNeighbour | Типизированный файл соседей | Локальный |
| Vertice | TPVertice | Ссылка на текущую вершину графа | Локальный |
| Neighbour | TPNeighbour | Ссылка на текущего соседа вершины | Локальный |
| v | Integer | Параметр цикла по соседям вершины | Локальный |

### Структура данных алгоритма SaveGraph

Таблица 15 – Структура данных алгоритма SaveGraph (Graph, VerFileName, ArcFileName)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Граф для чтения из файлов | Формальный |
| VerFileName | String | Имя файла с информацией о вершинах графа | Формальный |
| ArcFileName | String | Имя файла с информацией о дугах графа | Формальный |
| VerFile | File of TVertice | Типизированный файл вершин | Локальный |
| ArcFile | File of TNeighbour | Типизированный файл соседей | Локальный |
| Vertice | TPVertice | Ссылка на текущую вершину графа | Локальный |
| Neighbour | TPNeighbour | Ссылка на текущего соседа вершины | Локальный |

### Структура данных алгоритма ImportGraph

Таблица 16 – Структура данных алгоритма ImportGraph (Graph, ExcelFileName)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Граф для импорта из книги Excel | Формальный |
| ExcelFileName | String | Имя книги Excel с матрицей весов | Формальный |
| MyExcel | Variant | Объект для открытия Excel | Локальный |
| Sheet | OLEVariant | Объект активного листа книги | Локальный |
| Weights | OLEVariant | Объект диапазона ячеек с матрицей весов | Локальный |
| CLSID | TCLSID | Идентификатор, определяющий тип COM объекта | Локальный |
| i | Integer | Параметр цикла по строкам Weights | Локальный |
| j | Integer | Параметр цикла по столбцам Weights | Локальный |
| Rows | Integer | Количество строк Weights | Локальный |
| Cols | Integer | Количество столбцов Weights | Локальный |
| w | Integer | Значение ячейки Weights для записи веса дуги | Локальный |
| ww | Integer | Значение ячейки Weights, симметричное w относительно главной диагонали | Локальный |
| IsIncorrect | Boolean | Флаг о существовании в Weights недопустимых дуг | Локальный |

### Структура данных алгоритма ToWeightMatrix

Таблица 17 – Структура данных алгоритма ToWeightMatrix (Graph, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Полученный для преобразования граф | Формальный |
| Res | TWeightMatrix | Матрица весов полученного графа | Формальный |
| Vertice | TPVertice | Ссылка на текущую вершину графа | Локальный |
| Neighbour | TPNeighbour | Ссылка на текущего соседа вершины | Локальный |
| v | Integer | Параметр цикла по вершинам | Локальный |
| u | Integer | Параметр цикла по соседям | Локальный |

### Структура данных алгоритма DFS

Таблица 18 – Структура данных алгоритма DFS (Graph, Src, Dest, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TWeightMatrix | Полученный для поиска граф | Формальный |
| Src | Integer | Номер вершины-начала пути | Формальный |
| Dest | Integer | Номер вершины-конца пути | Формальный |
| Res | TSearchInfo | Результат поиска в глубину | Формальный |
| v | Integer | Номер посещаемой вершины | Локальный |
| u | Integer | Номер соседа посещаемой вершины | Локальный |
| Order | Integer | Порядок графа | Локальный |

Продолжение таблицы 18

|  |  |  |  |
| --- | --- | --- | --- |
| Stack | TStack | Стек вершин для итеративного поиска в глубину | Локальный |
| isVisited | Array of Boolean | Массив флагов о посещённости вершин | Локальный |
| Parents | Array of Integer | Массив предков вершин | Локальный |

### Структура данных алгоритма BFS

Таблица 19 – Структура данных алгоритма BFS (Graph, Src, Dest, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TWeightMatrix | Полученный для поиска граф | Формальный |
| Src | Integer | Номер вершины-начала пути | Формальный |
| Dest | Integer | Номер вершины-конца пути | Формальный |
| Res | TSearchInfo | Результат поиска в ширину | Формальный |
| v | Integer | Номер посещаемой вершины | Локальный |
| u | Integer | Номер соседа посещаемой вершины | Локальный |
| Order | Integer | Порядок графа | Локальный |
| Queue | TQueue | Очередь вершин для итеративного поиска в ширину | Локальный |
| isVisited | Array of Boolean | Массив флагов о том, что вершина уже была добавлена в стек | Локальный |
| Parents | Array of Integer | Массив предков вершин | Локальный |

### Структура данных алгоритма Dijkstra

Таблица 20 – Структура данных алгоритма Dijkstra (Graph, Src, Dest, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TWeightMatrix | Полученный для поиска граф | Формальный |
| Src | Integer | Номер вершины-начала пути | Формальный |
| Dest | Integer | Номер вершины-конца пути | Формальный |
| Res | TSearchInfo | Результат поиска в ширину | Формальный |
| v | Integer | Номер посещаемой вершины | Локальный |
| u | Integer | Номер соседа посещаемой вершины | Локальный |
| Order | Integer | Порядок графа | Локальный |
| Marks | Array of Integer | Массив меток (расстояний от начальной) вершин | Локальный |
| isVisited | Array of Boolean | Массив флагов о том, что вершина уже была посещена | Локальный |
| Parents | Array of Integer | Массив предков вершин | Локальный |
| d | Integer | Расстояние до текущей вершины | Локальный |

### Структура данных алгоритма RestorePath

Таблица 21 – Структура данных алгоритма RestorePath (Graph, Parents, Src, Dest, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TWeightMatrix | Полученный для поиска граф | Формальный |
| Parents | Array of Integer | Массив предков вершин | Формальный |
| Src | Integer | Номер вершины-начала пути | Формальный |

Продолжение таблицы 21

|  |  |  |  |
| --- | --- | --- | --- |
| Dest | Integer | Номер вершины-конца пути | Формальный |
| Res | TSearchInfo | Результат поиска в ширину | Формальный |
| v | Integer | Номер вершины-начала дуги | Локальный |
| u | Integer | Номер вершины-конца дуги | Локальный |
| Splitter | String | Разделитель между вершинами в строке пути | Локальный |

### Структура данных алгоритма InitializeStack

Таблица 22 – Структура данных алгоритма InitializeStack (Stack)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Stack | TStack | Стек для инициализации | Формальный |

### Структура данных алгоритма InitializeQueue

Таблица 23 – Структура данных алгоритма InitializeQueue (Queue)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Queue | TQueue | Очередь для инициализации хвоста и головы | Формальный |

### Структура данных алгоритма DestroyList

Таблица 24 – Структура данных алгоритма DestroyList (Head)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Head | TPItem | Ссылка на голову освобождаемого списка | Формальный |
| Item | TPItem | Ссылка освобождаемое звено | Локальный |

### Структура данных алгоритма Push

Таблица 25 – Структура данных алгоритма Push (Stack, n)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Stack | TStack | Стек для вставки | Формальный |
| n | Integer | Значение вставляемого звена | Формальный |
| Item | TPItem | Ссылка на вставляемое звено | Локальный |

### Структура данных алгоритма Enqueue

Таблица 26 – Структура данных алгоритма Enqueue (Queue, n)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Queue | TQueue | Очередь для добавления | Формальный |
| n | Integer | Значение вставляемого звена | Формальный |
| Item | TPItem | Ссылка на вставляемое звено | Локальный |

### Структура данных алгоритма Pop

Таблица 27 – Структура данных алгоритма Pop (Stack, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Stack | TStack | Стек для извлечения | Формальный |
| Res | Integer | Значение извлекаемого звена | Формальный |
| Item | TStack | Ссылка на извлекаемое звено | Локальный |

### Структура данных алгоритма Dequeue

Таблица 28 – Структура данных алгоритма Dequeue (Queue, Res)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Queue | TQueue | Очередь для извлечения | Формальный |
| Res | Integer | Значение извлекаемого звена | Формальный |
| Item | TPItem | Ссылка на извлекаемое звено | Локальный |

### Структура данных алгоритма MakeVisited

Таблица 29 – Структура данных алгоритма MakeVisited (Graph, Path)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Полученный граф | Формальный |
| Path | TStack | Полученный путь | Формальный |
| Vertice | TPVertice | Ссылка на текущую вершину | Локальный |
| Neighbour | TPNeighbour | Ссылка на текущего соседа | Локальный |
| v | Integer | Номер вершины, извлечённой из пути | Локальный |

### Структура данных алгоритма MakePassive

Таблица 30 – Структура данных алгоритма MakePassive (Graph)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Полученный граф | Формальный |
| Vertice | TPVertice | Ссылка на текущую вершину | Локальный |
| Neighbour | TPNeighbour | Ссылка на текущего соседа | Локальный |

### Структура данных алгоритма MakeRegPolygon

Таблица 31 – Структура данных алгоритма MakeRegPolygon (Graph, Width, Height)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | TGraph | Полученный граф | Формальный |
| Width | Integer | Ширина квадрата на холсте, который ограничивает граф | Формальный |
| Height | Integer | Высота квадрата на холсте, который ограничивает граф | Формальный |
| Vertice | TPVertice | Ссылка на текущую вершину | Локальный |
| ImageCenter | TPoint | Координата центра квадрата, который ограничивает граф | Локальный |
| PolygonRadius | Integer | Радиус правильного многоугольника | Локальный |
| Angle | Real | Текущий угол наклона радиус-вектора вершины относительно начального положения | Локальный |

## Схема алгоритмов решения задачи по ГОСТ 19.701-90

### Схема алгоритма AddVertice

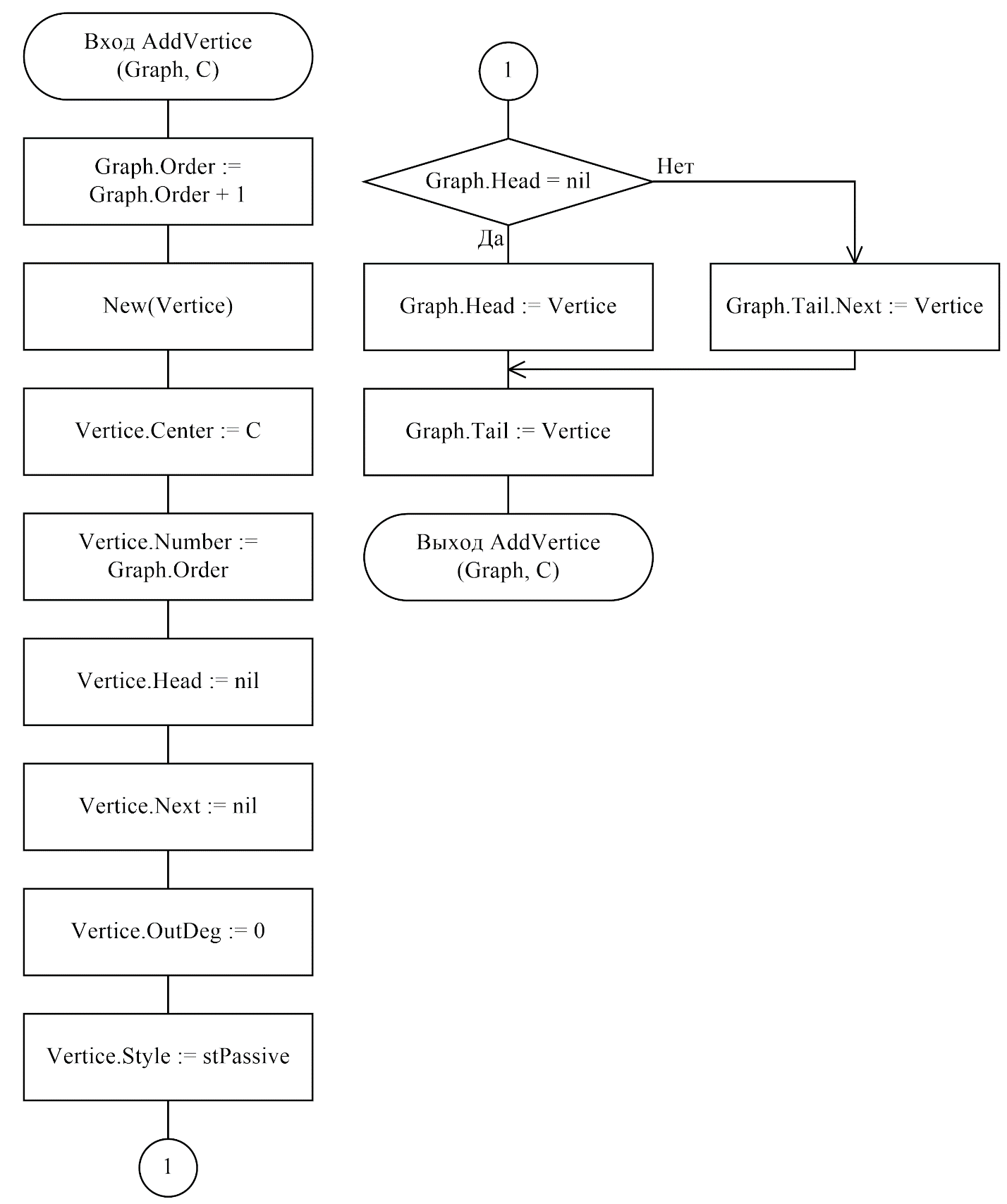


Рисунок 3.1 – Схема алгоритма AddVertice

### Схема алгоритма AddArc

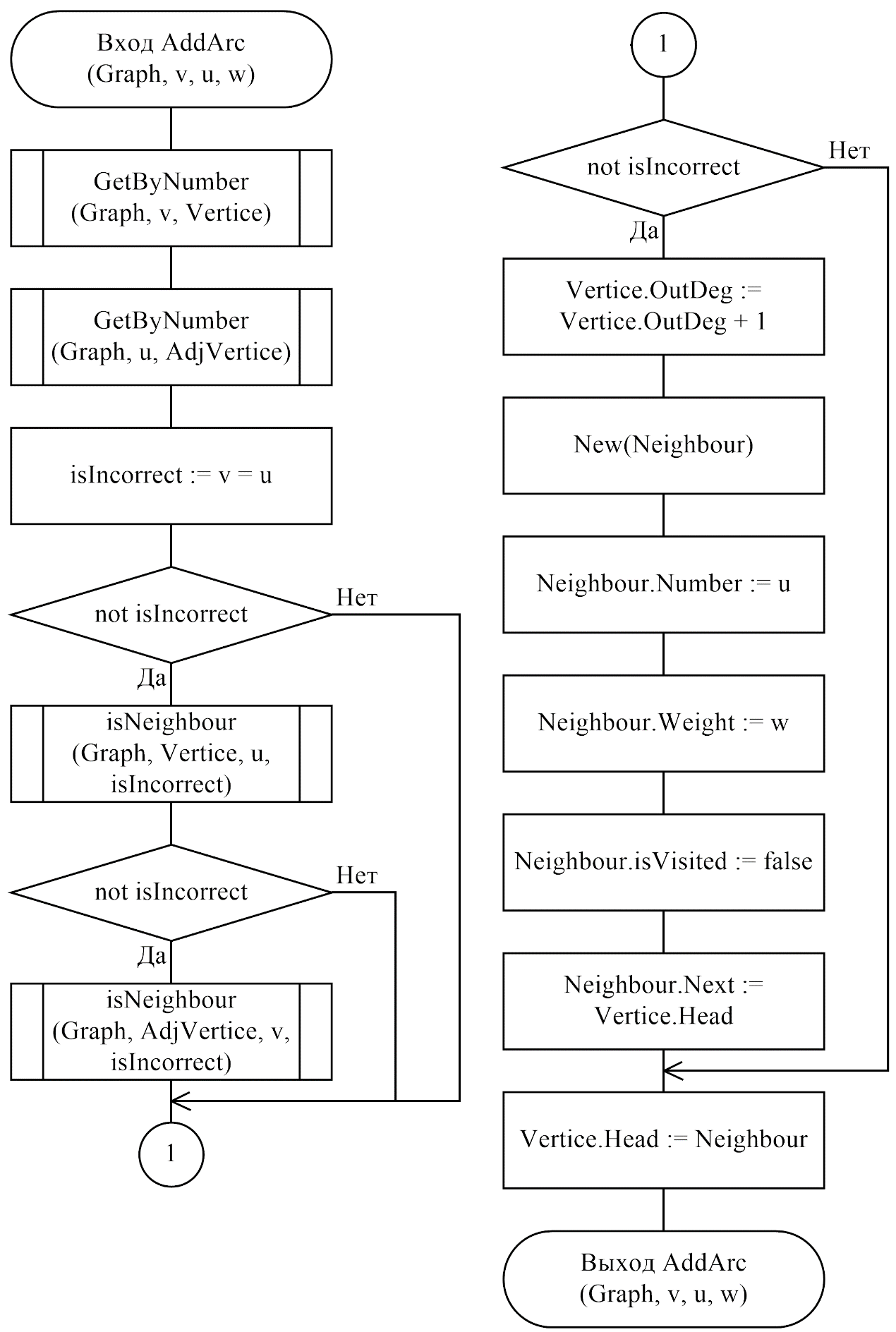


Рисунок 3.2 – Схема алгоритма AddArc

### Схема алгоритма DeleteVertice

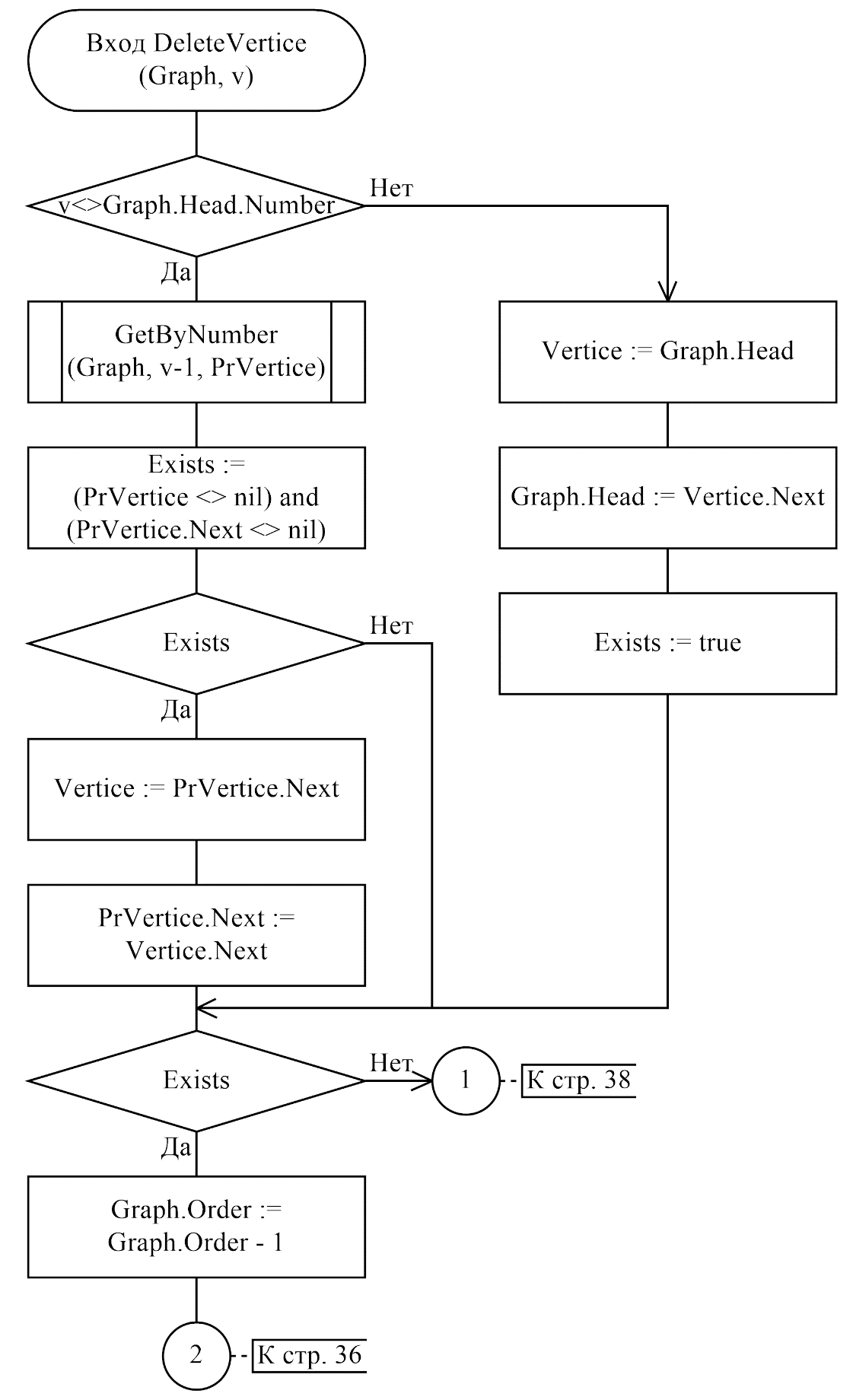


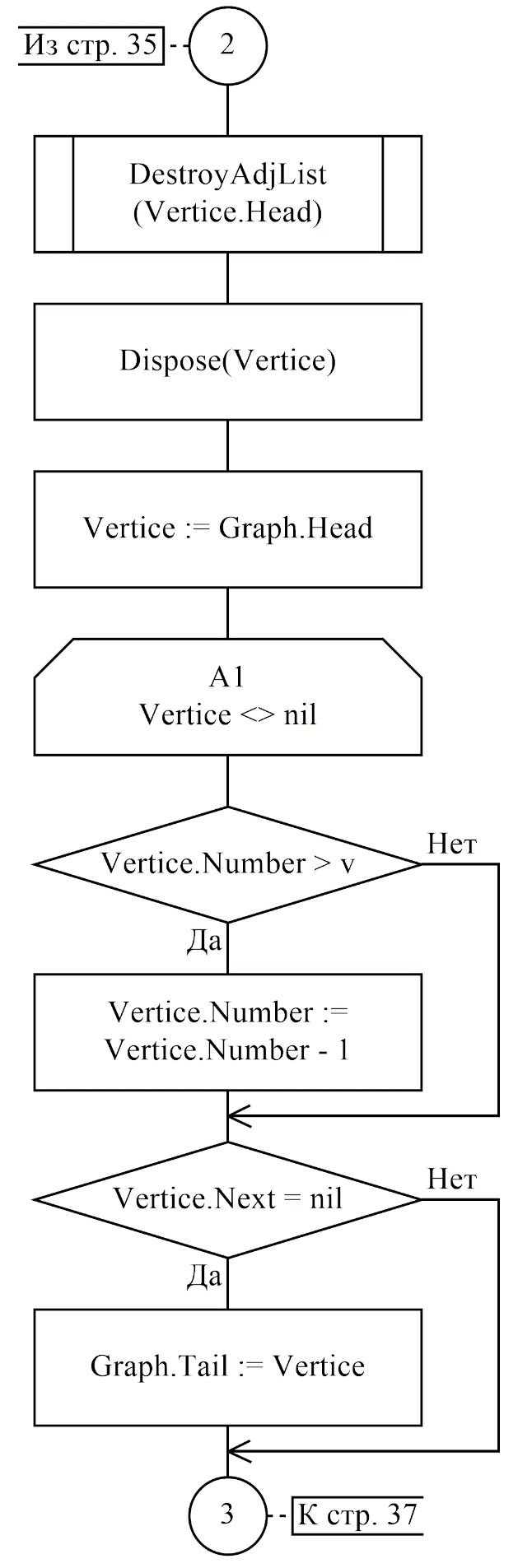
Рисунок 3.3 – Схема алгоритма DeleteVertice (часть 1)

Рисунок 3.4 – Схема алгоритма DeleteVertice (часть 2)

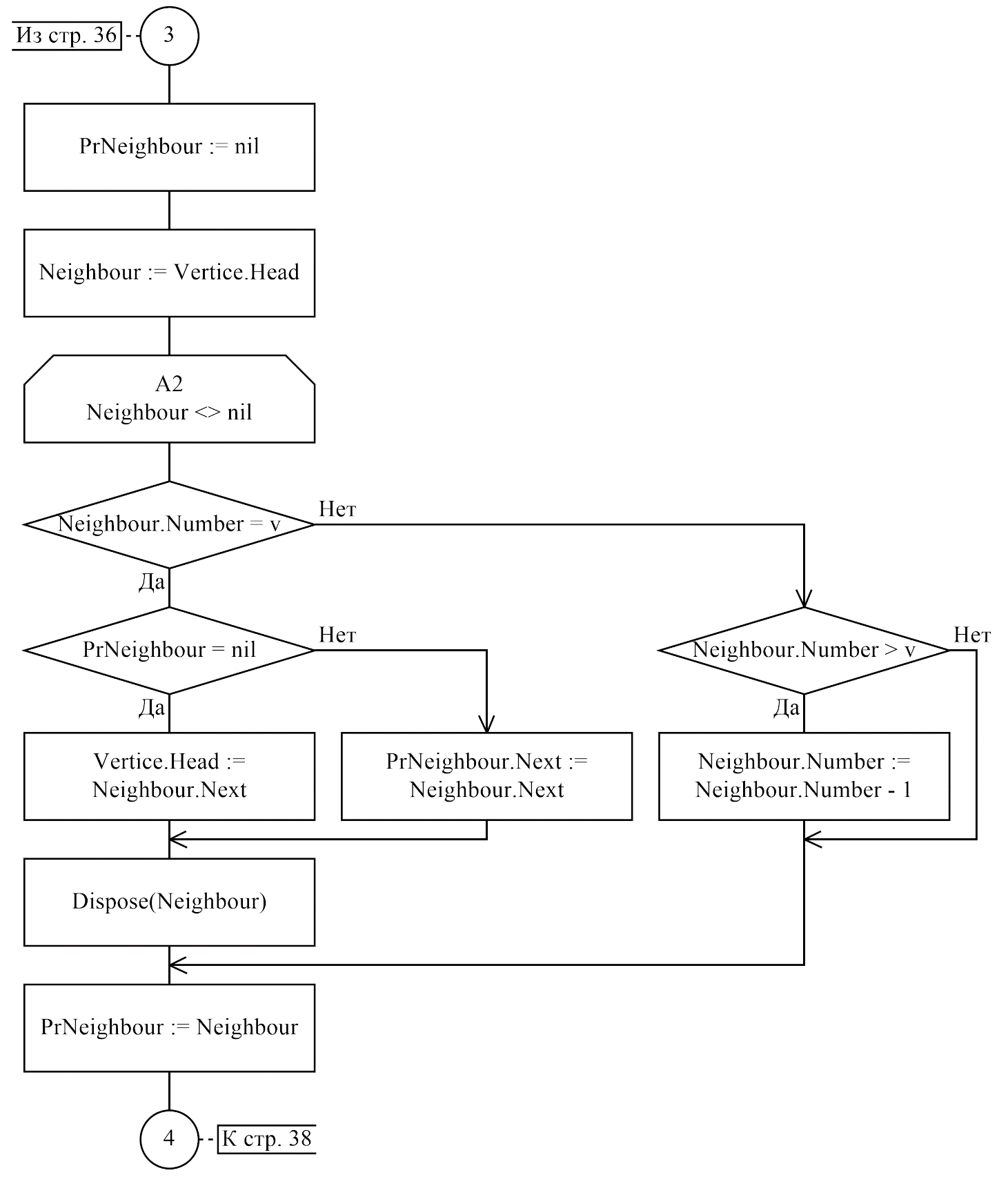


Рисунок 3.5 – Схема алгоритма DeleteVertice (часть 3)

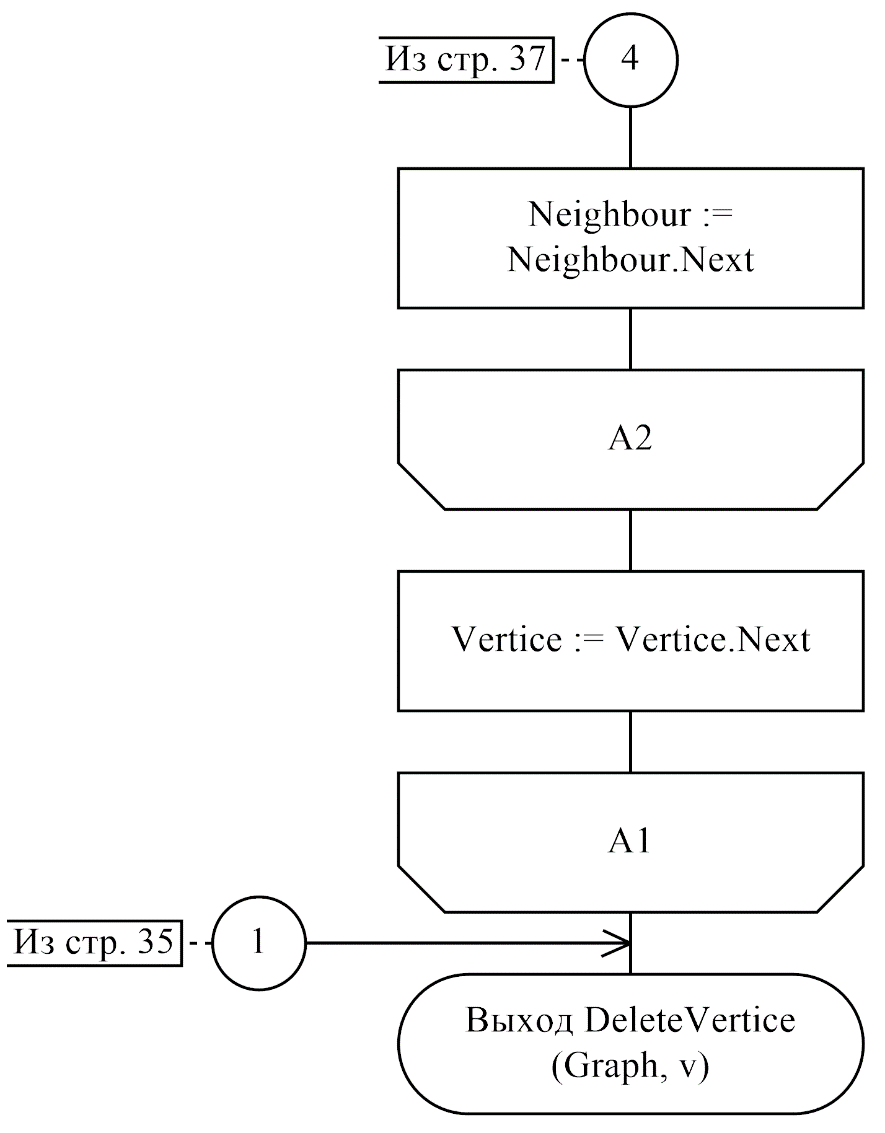


Рисунок 3.6 – Схема алгоритма DeleteVertice (часть 4)

### Схема алгоритма DeleteArc

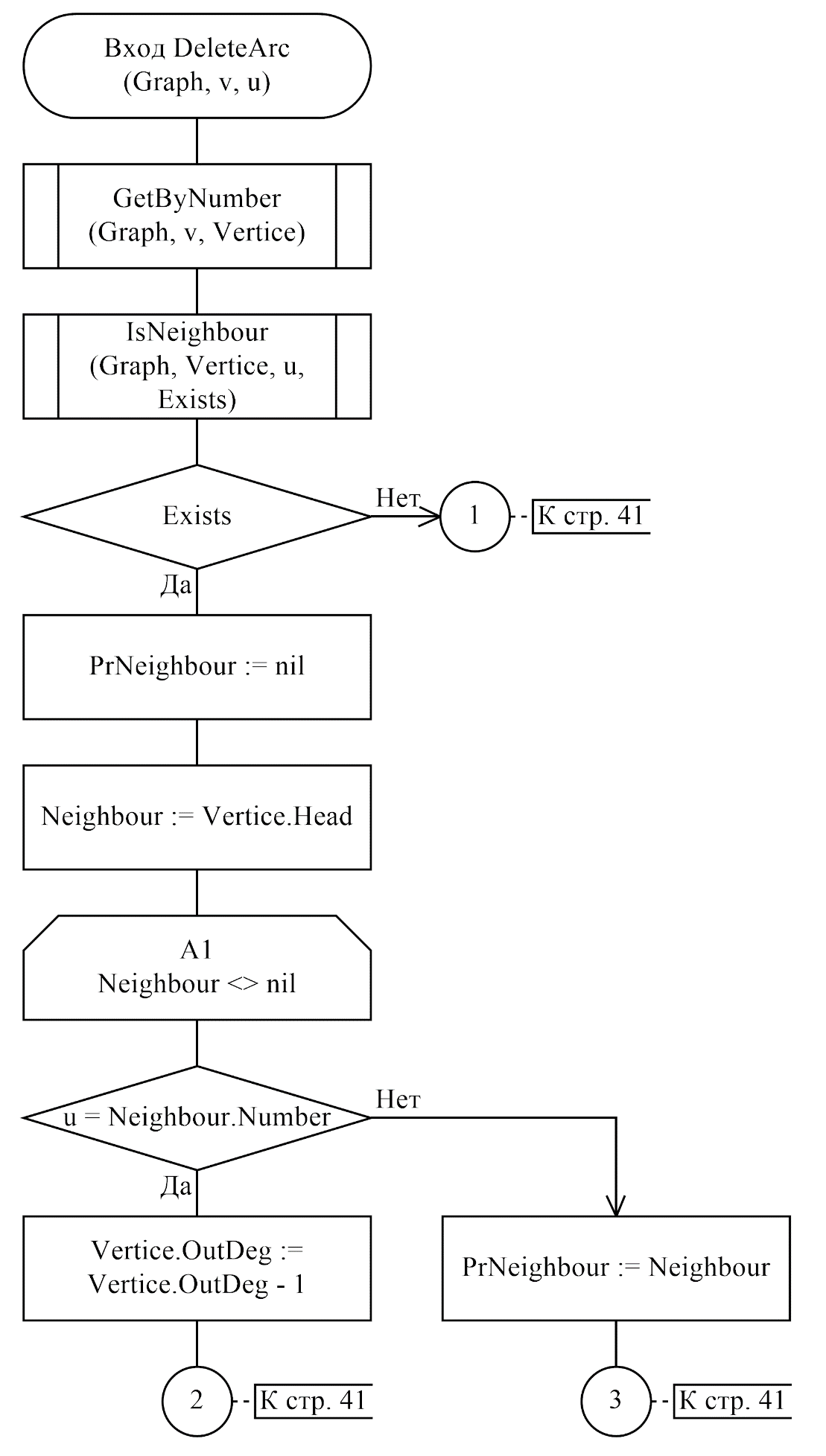


Рисунок 3.7 – Схема алгоритма DeleteArc (часть 1)

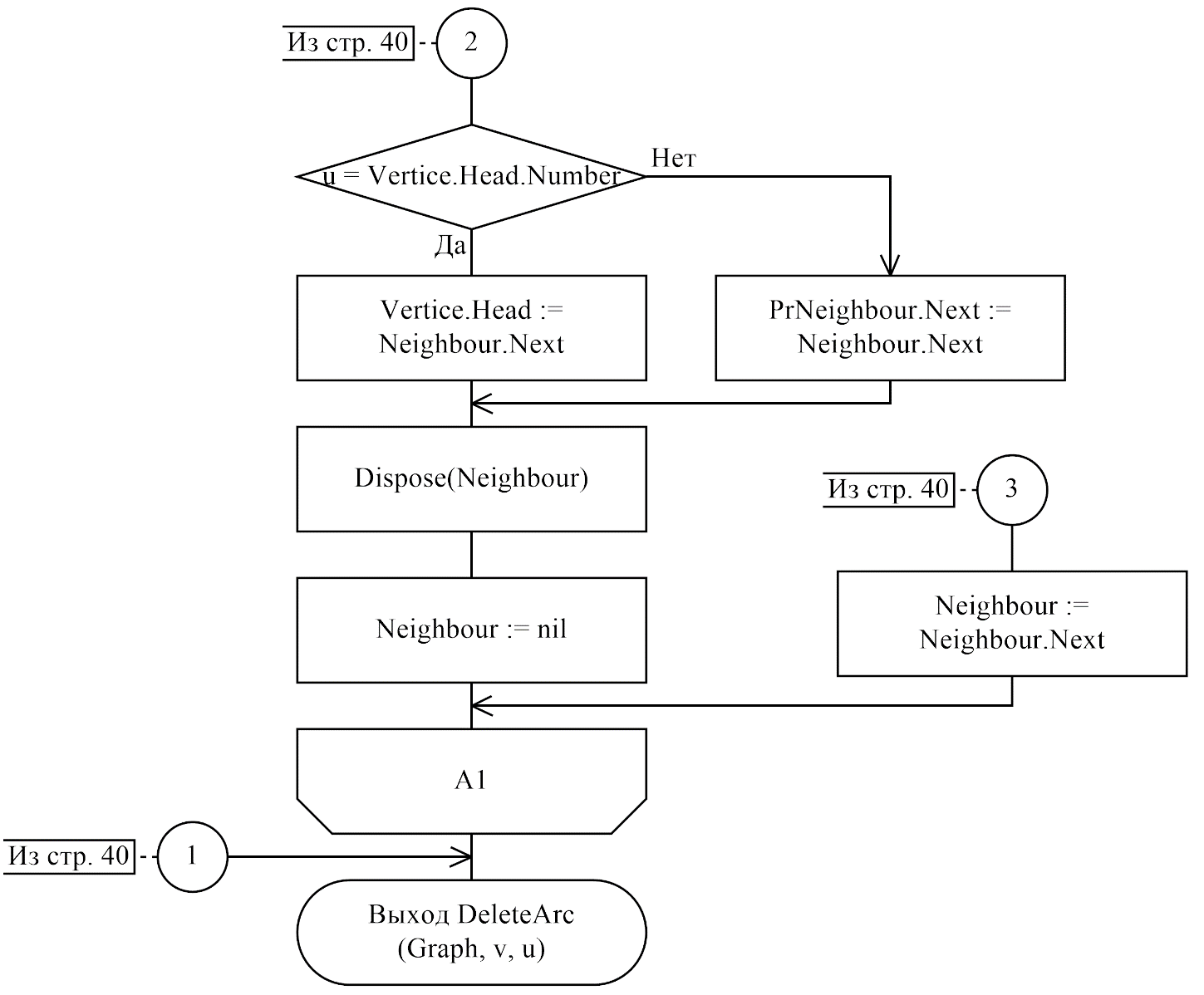


Рисунок 3.8 – Схема алгоритма DeleteArc (часть 2)

### Схема алгоритма IsNeigbour

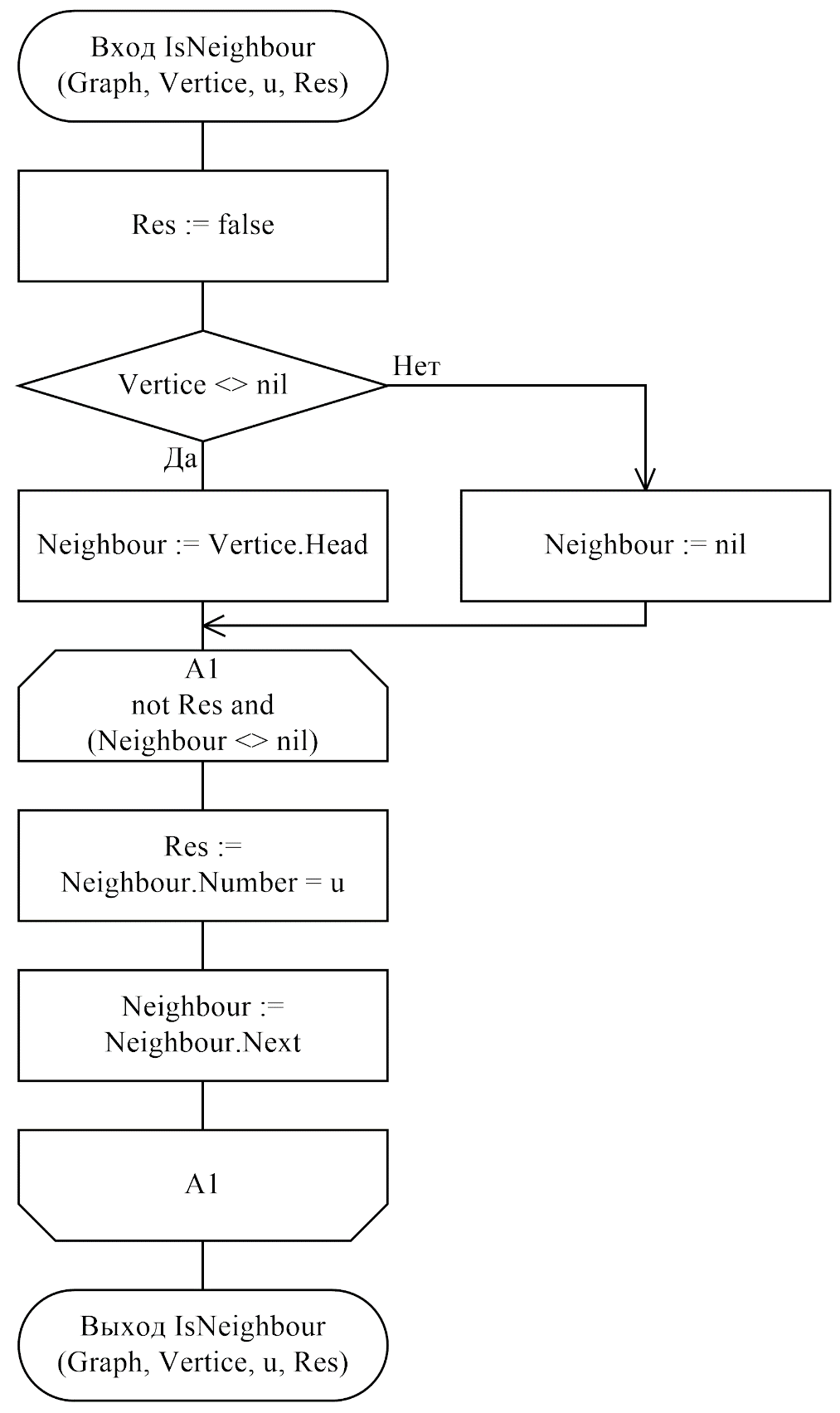


Рисунок 3.9 – Схема алгоритма IsNeighbour

### Схема алгоритма GetByNumber

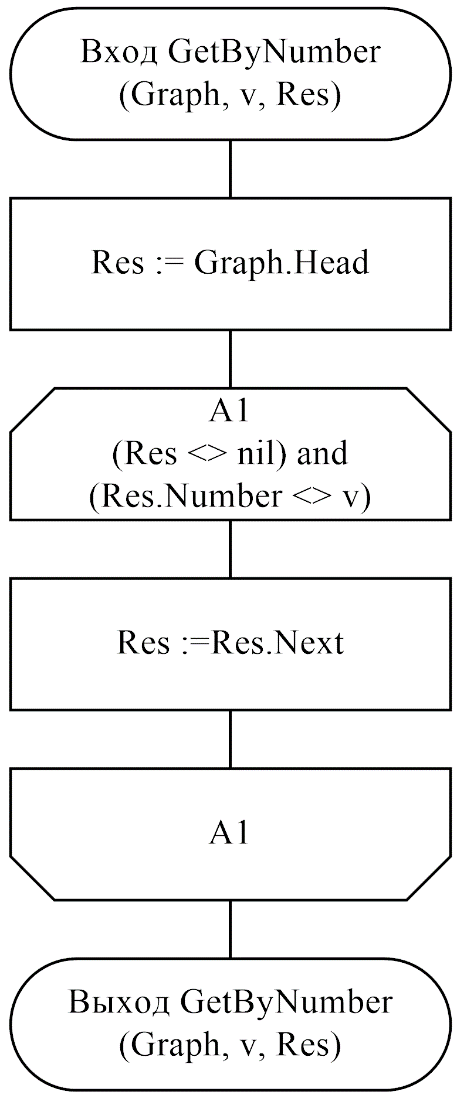


Рисунок 3.10 – Схема алгоритма GetByNumber

### Схема алгоритма OpenGraph

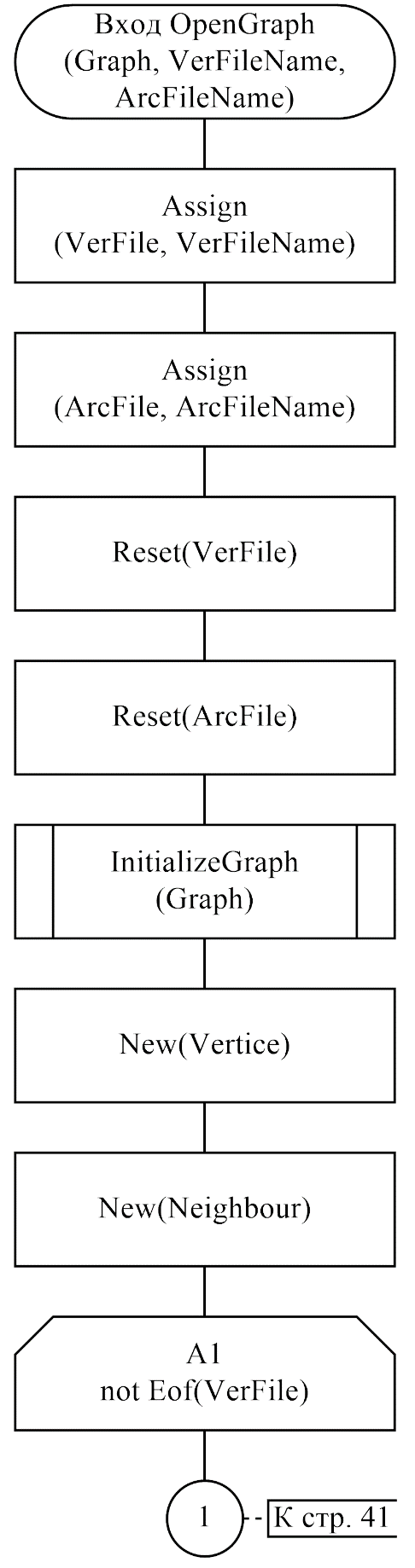


Рисунок 3.11 – Схема алгоритма OpenGraph (часть 1)

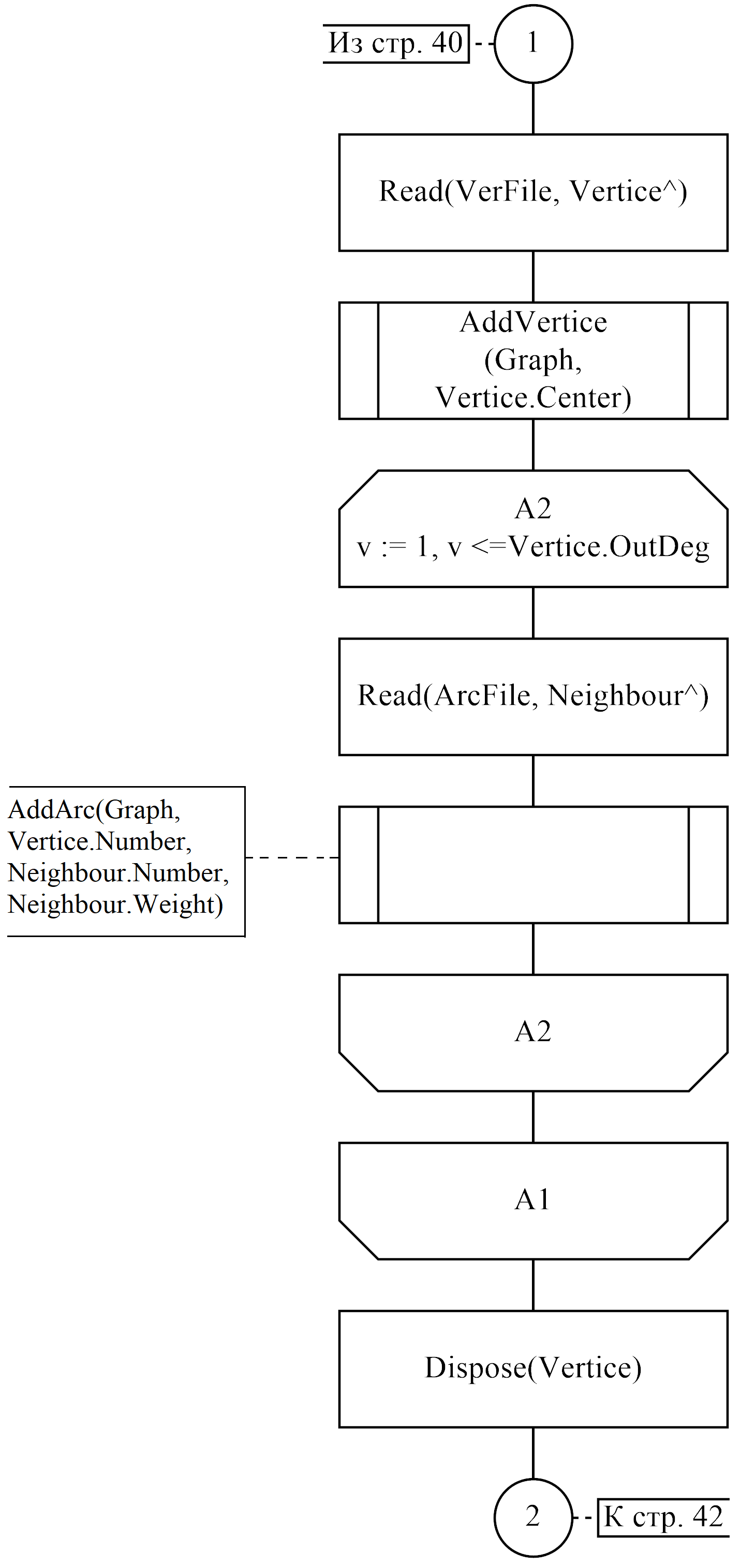


Рисунок 3.12 – Схема алгоритма OpenGraph (часть 2)

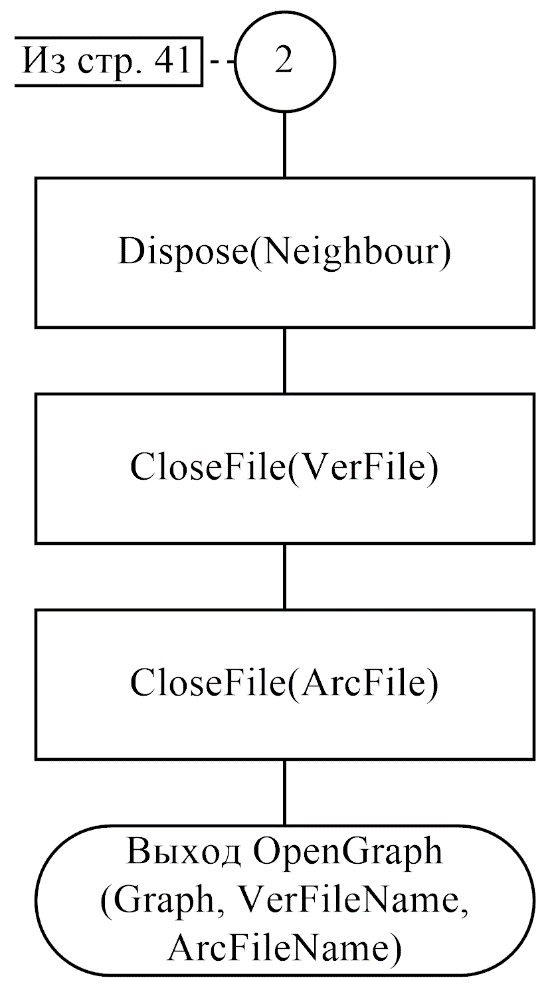


Рисунок 3.13 – Схема алгоритма OpenGraph (часть 3)

### Схема алгоритма SaveGraph

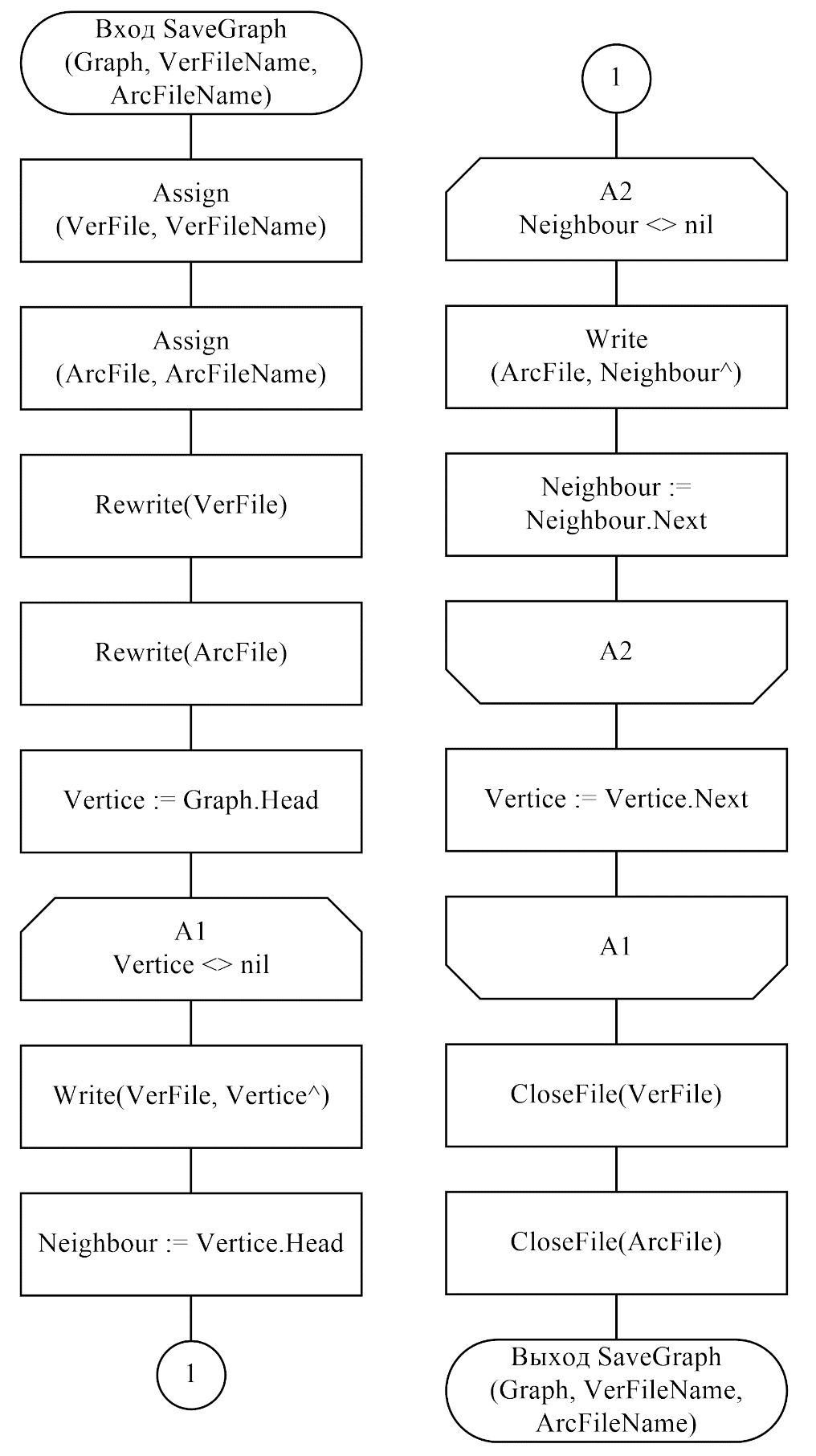


Рисунок 3.14 – Схема алгоритма SaveGraph

### Схема алгоритма ToWeightMatrix

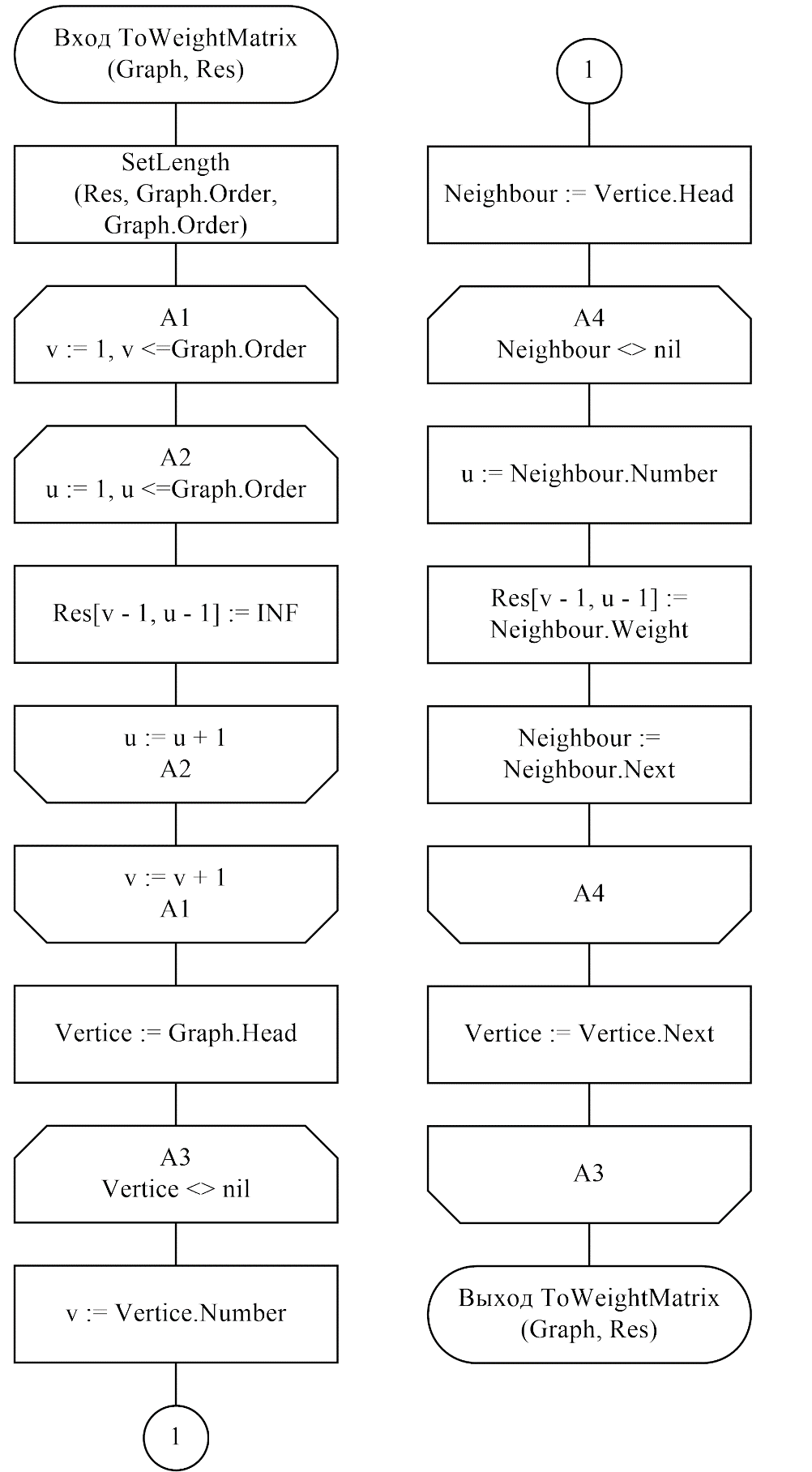


Рисунок 3.15 – Схема алгоритма ToWeightMatrix

### Схема алгоритма DFS

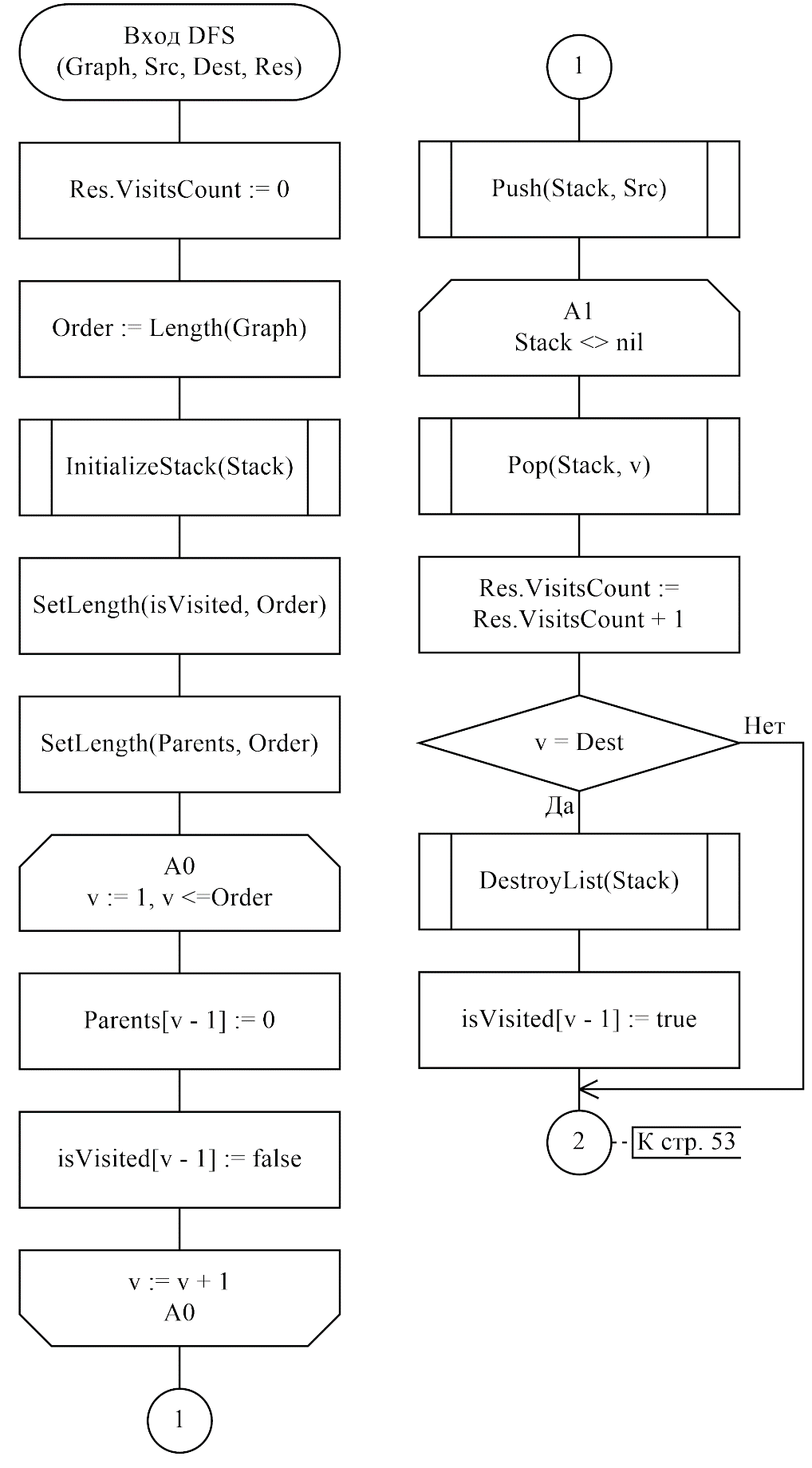


Рисунок 3.16 – Схема алгоритма DFS (часть 1)

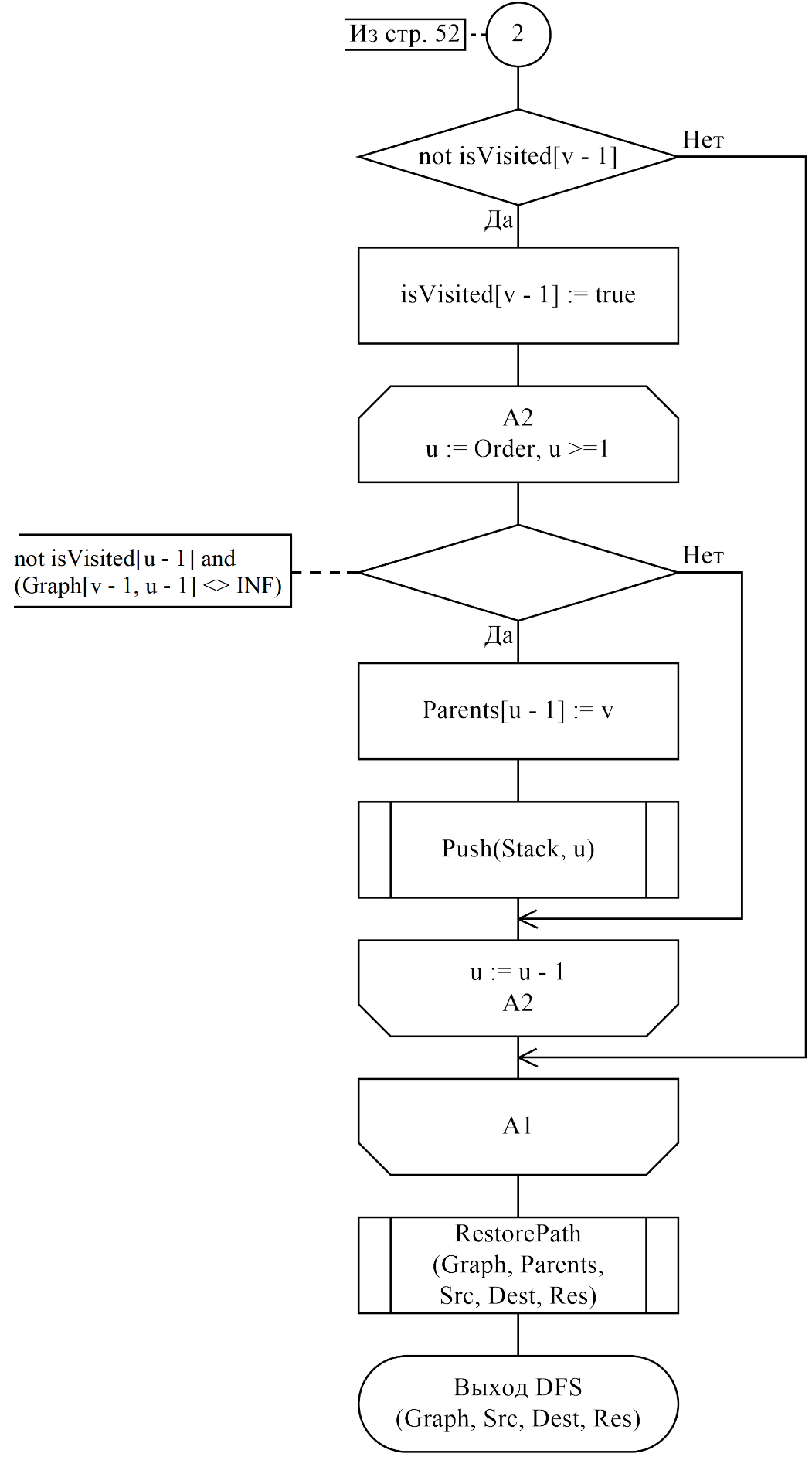


Рисунок 3.17 – Схема алгоритма DFS (часть 2)

### Схема алгоритма BFS

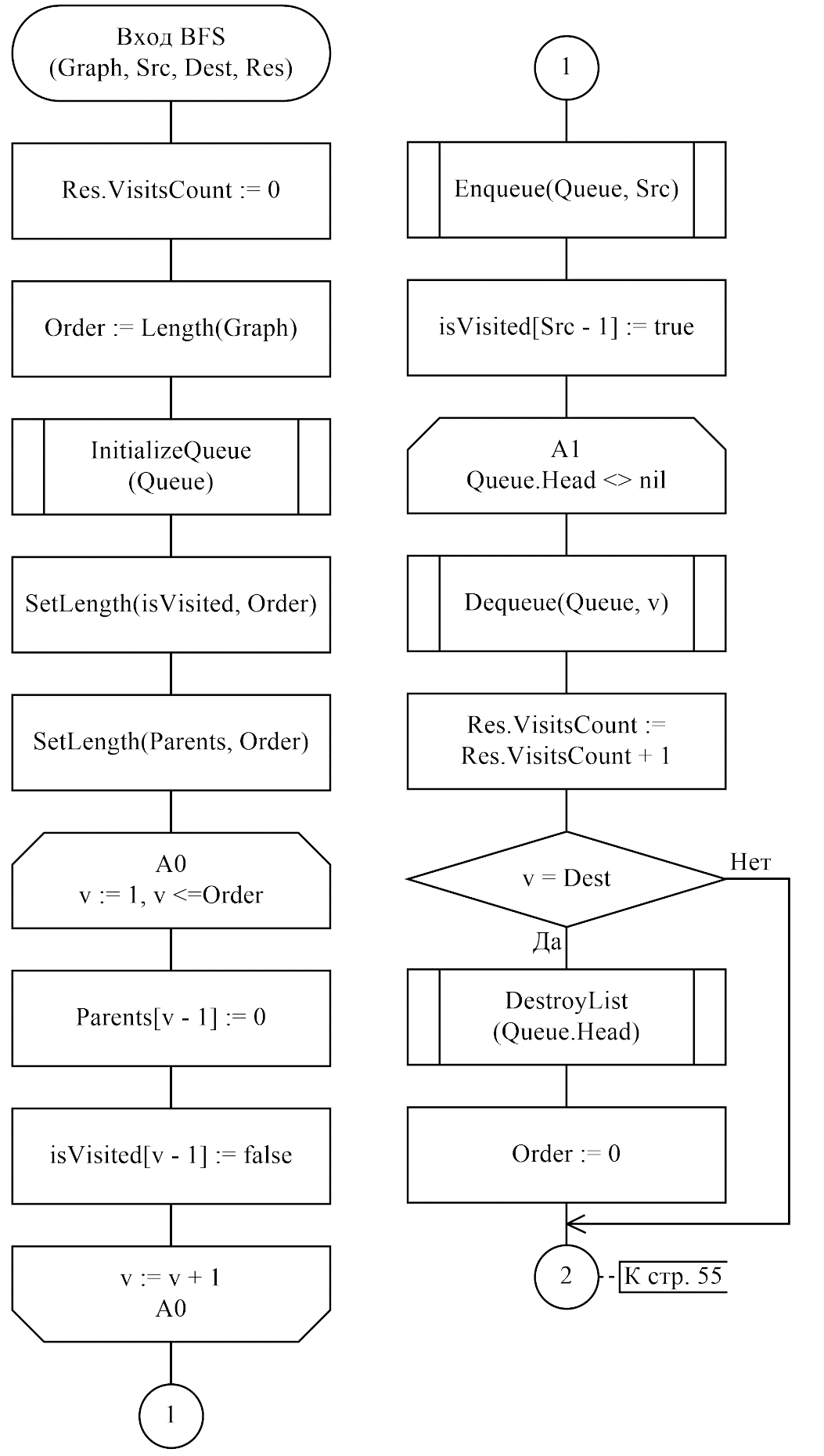


Рисунок 3.18 – Схема алгоритма BFS (часть 1)

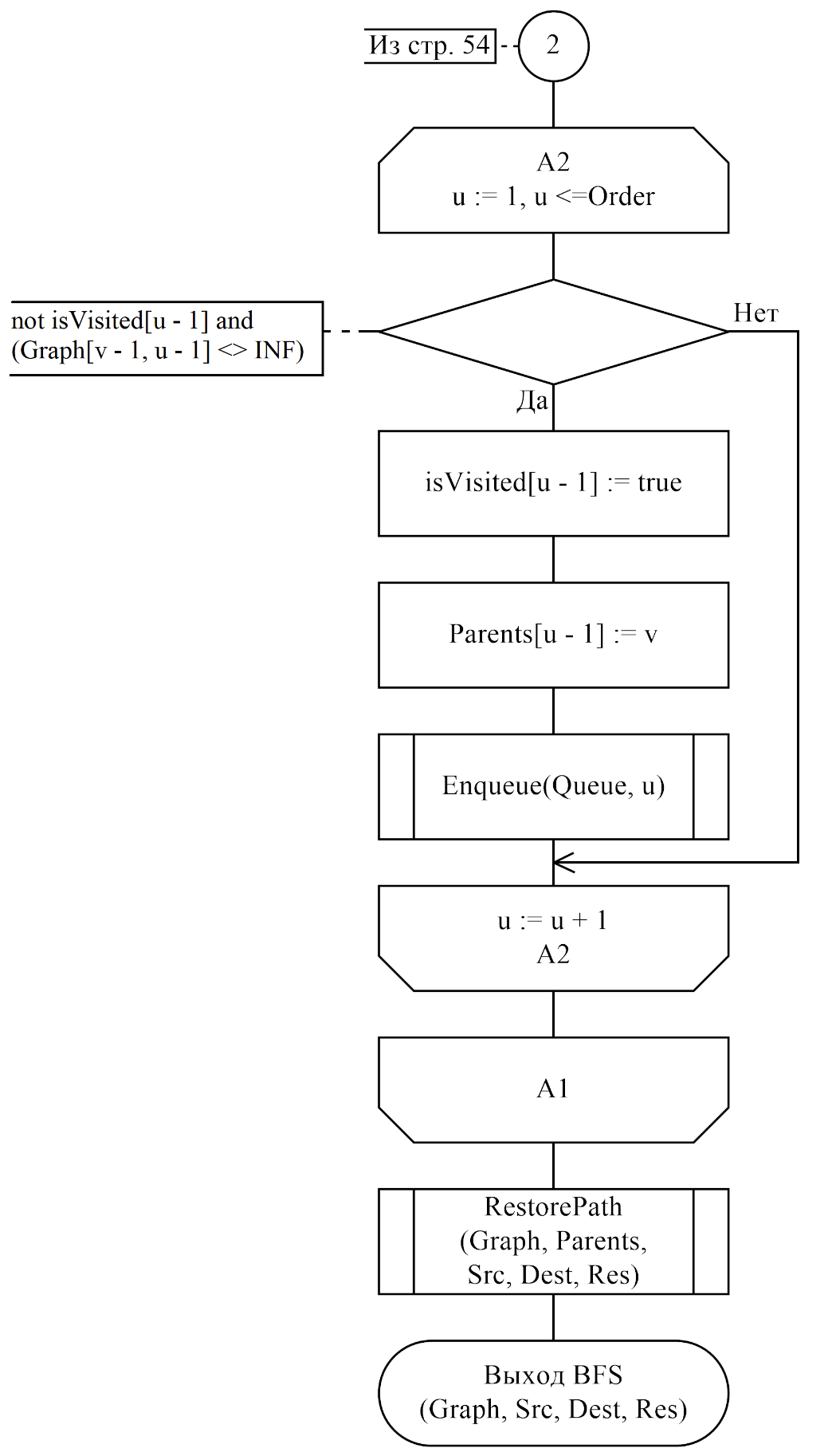


Рисунок 3.19 – Схема алгоритма BFS (часть 2)

### Схема алгоритма Dijkstra

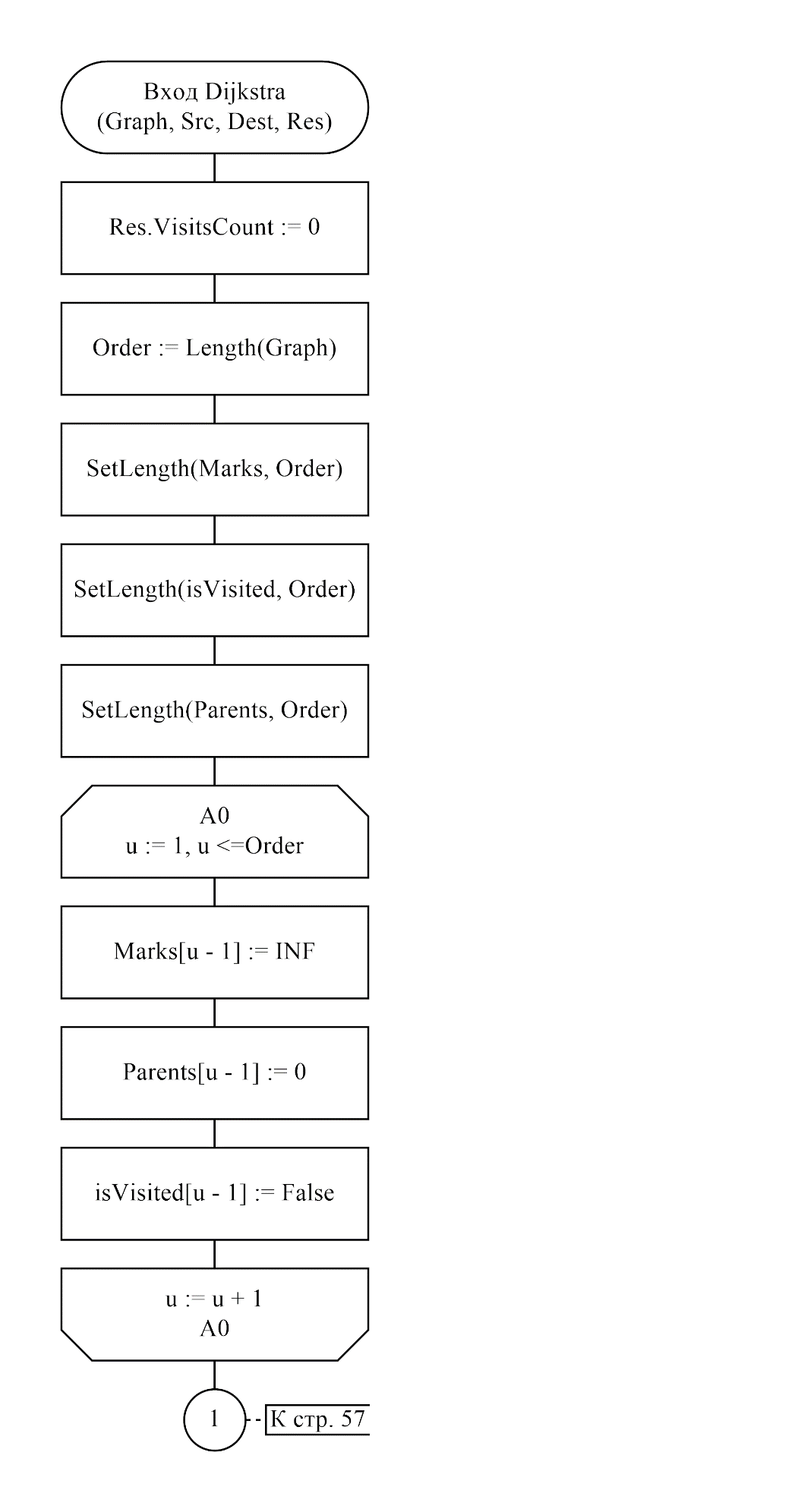


Рисунок 3.20 – Схема алгоритма Dijkstra (часть 1)

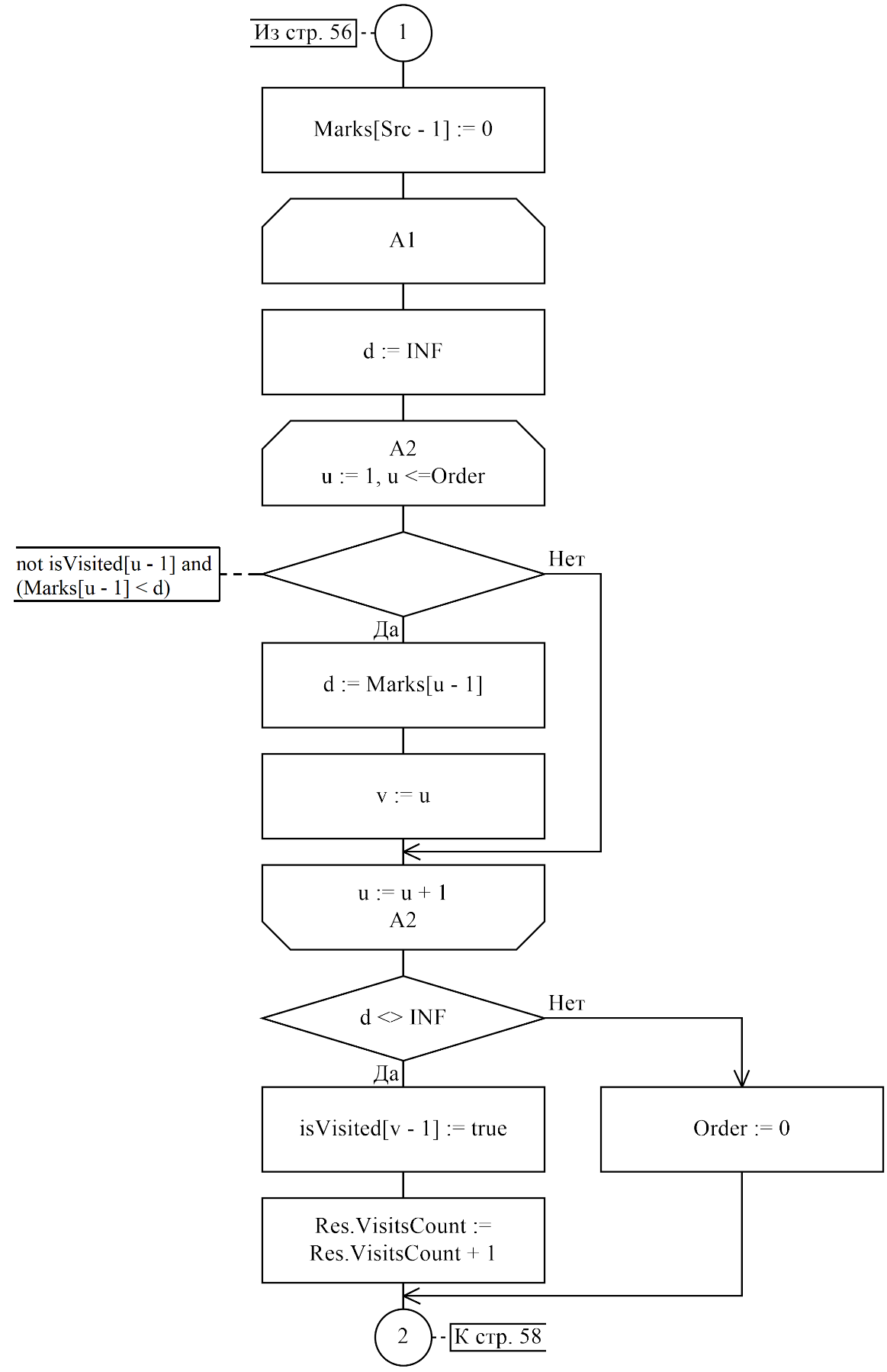


Рисунок 3.21 – Схема алгоритма Dijkstra (часть 2)

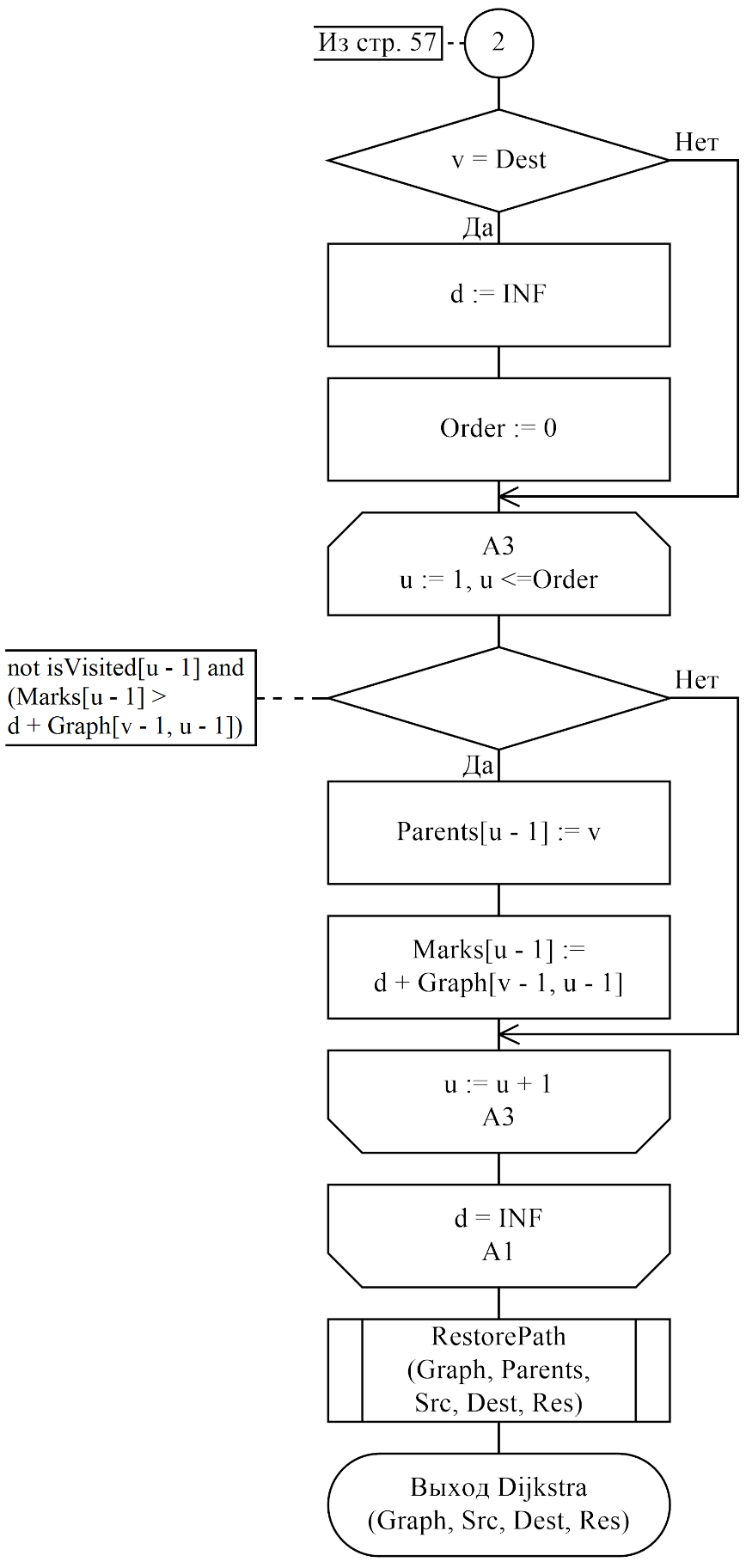


Рисунок 3.22 – Схема алгоритма Dijkstra (часть 3)

### Схема алгоритма RestorePath

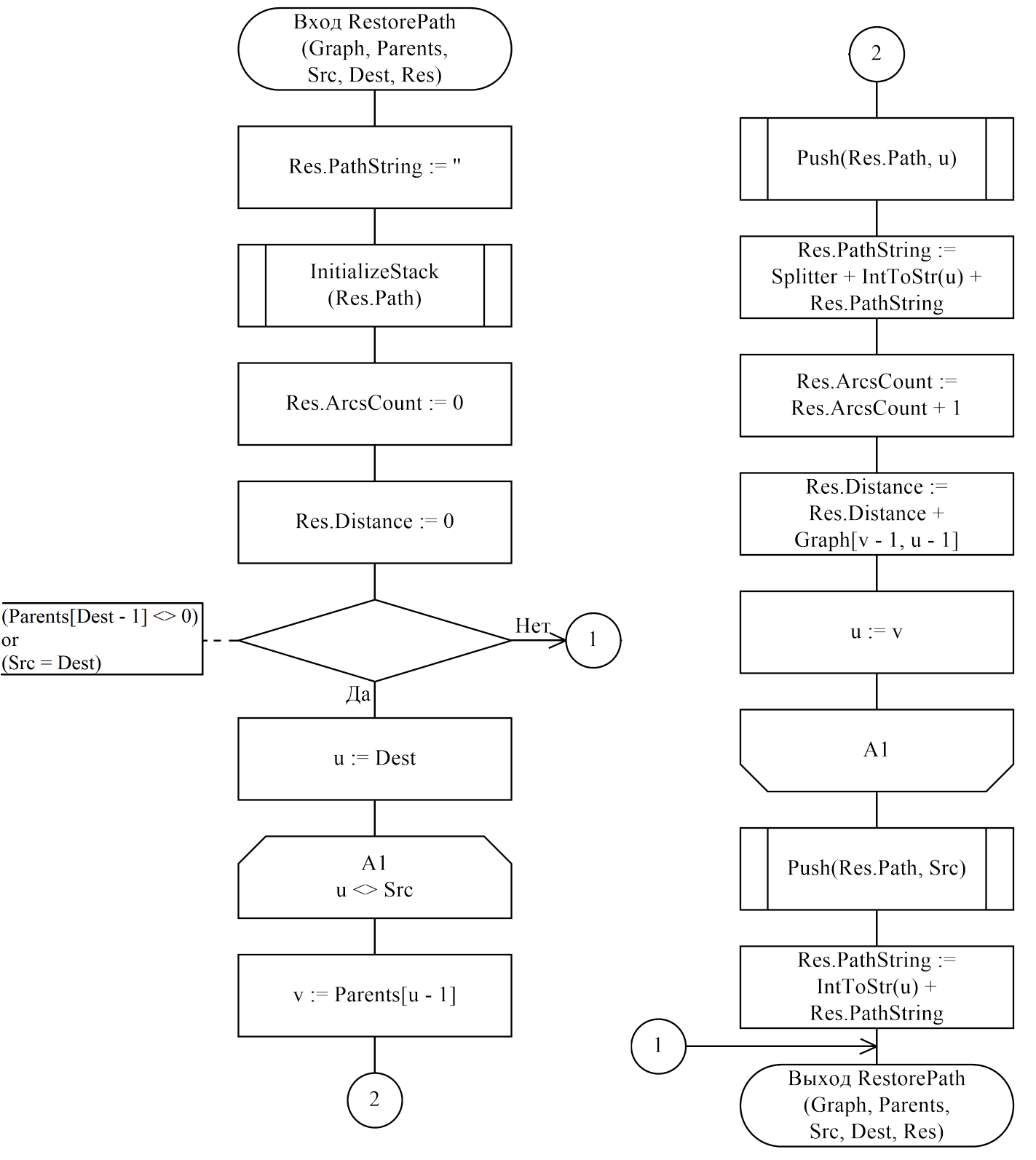


Рисунок 3.23 – Схема алгоритма RestorePath

### Схема алгоритма Push

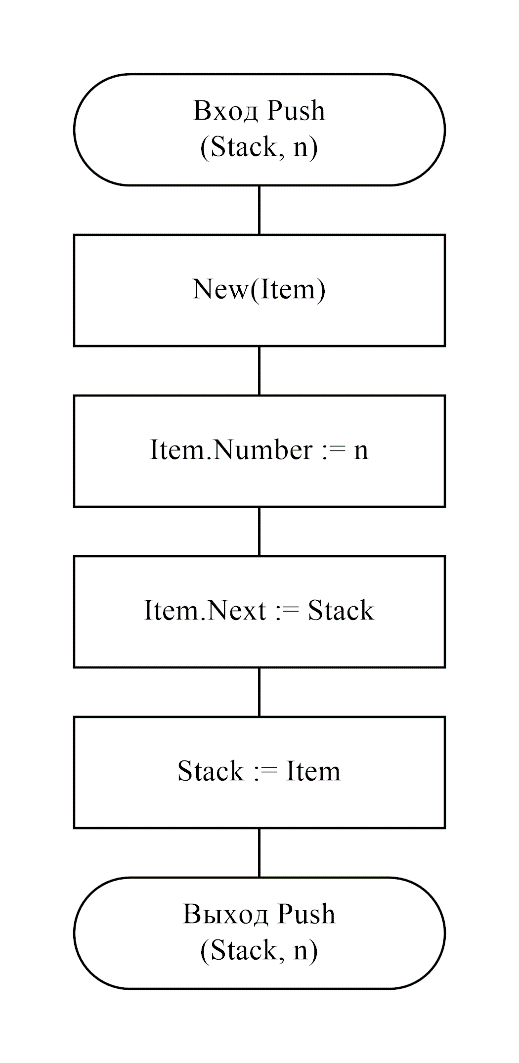


Рисунок 3.24 – Схема алгоритма Push

### Схема алгоритма Enqueue

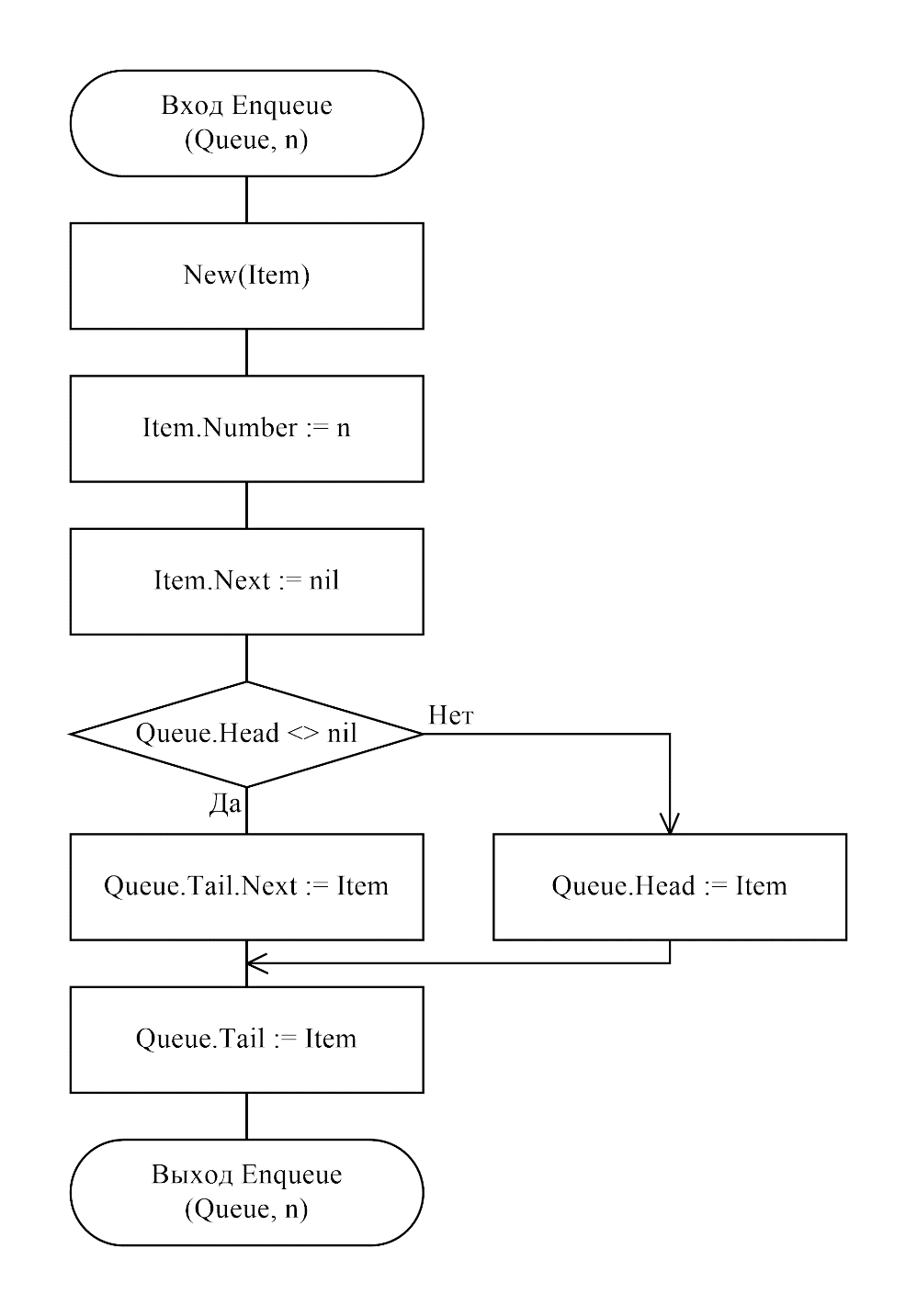


Рисунок 3.25 – Схема алгоритма Enqueue

### Схема алгоритма Pop

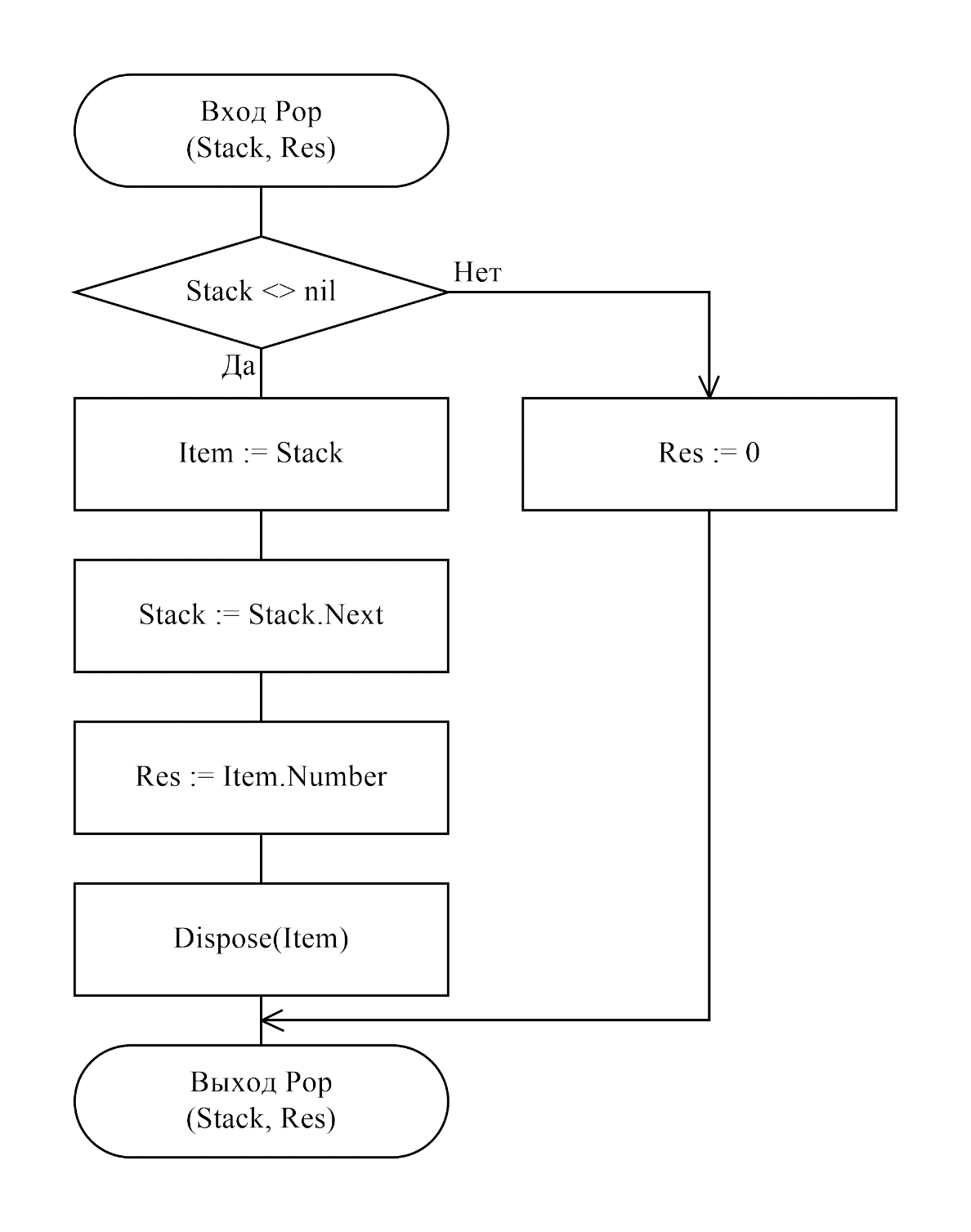


Рисунок 3.26 – Схема алгоритма Pop

### Схема алгоритма Pop

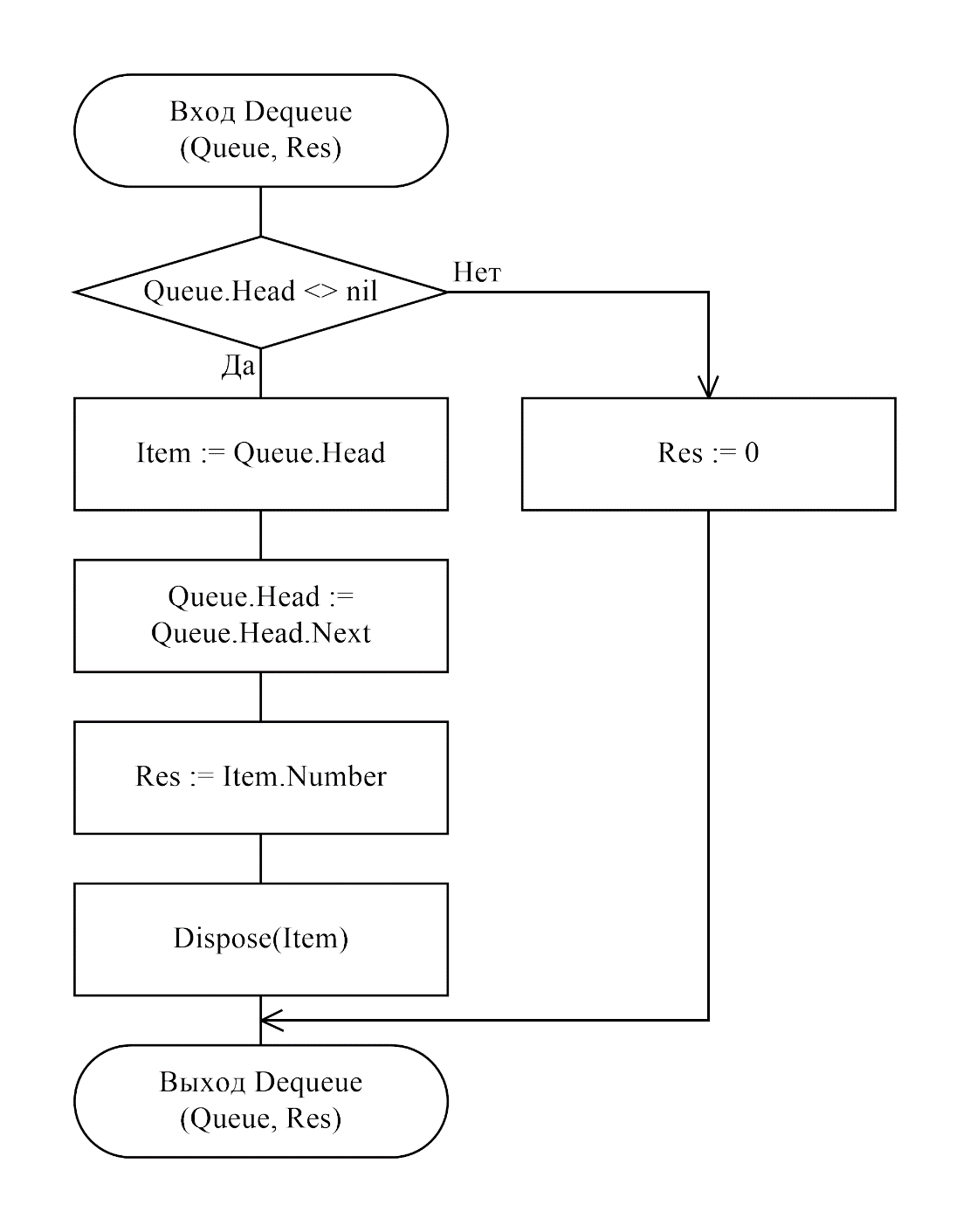


Рисунок 3.27 – Схема алгоритма Dequeue

## Графический интерфейс

Для организации графического интерфейса приложения было использовано 4 формы: fmEditor, fmArcInput, fmSearchOutput, fmAbout. Взаимосвязь между формами представлена на рисунке 28.

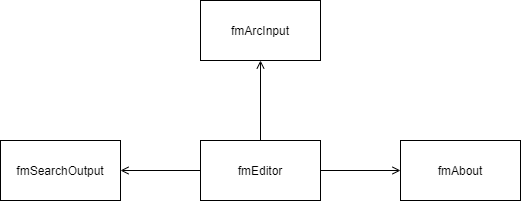


Рисунок 3.28 – Взаимосвязь графических форм

### Описание графических компонентов формы fmEditor

Форма fmEditor – это главная форма с холстом для рисования на нём новых элементов графа. На ней пользователь может выбрать как изменять созданный граф, а такжеосуществлять дополнительные действия: сохранить или открыть граф с помощью 2-х типизированных файлов, экспортировать граф в формате BMP-изображения, импортировать граф из таблицы весов из электронной книги Excel. Данная форма имеет вид, представленный на рисунке 29.



Рисунок 3.29 - Вид формы fmEditor при запуске программы

Составляющие формы fmEditor:

* область рисования «pbCanvas», на которой будет отображаться редактируемый граф;
* быстрая кнопка «Добавить вершину», которая переключает режим формы на добавление вершин;
* быстрая кнопка «Добавить дугу», которая переключает режим формы на добавление дуг;
* быстрая кнопка «Удалить вершину», которая переключает режим формы на удаление вершин;
* быстрая кнопка «Удалить дугу», которая переключает режим формы на удаление дуг;
* быстрая кнопка «Перемещать», которая переключает режим формы на изменение местоположения активной вершины;
* флажок «Невзвешенные дуги», который не позволяет открываться вспомогательной форме ArcInput и присваивает её полю веса значение 1;
* быстрая кнопка «Поиск в глубину», которая переключает режим работы формы на поиск алгоритмом обхода в глубину;
* быстрая кнопка «Поиск в ширину», которая переключает режим работы формы на поиск алгоритмом обхода в ширину;
* быстрая кнопка «Поиск Дейкстры», которая переключает режим работы формы на поиск алгоритмом Дейкстры;
* меню с вкладками «Файл», «Граф», «Помощь» и элементами «Открыть», «Сохранить», «Сохранить как», «Выход», «Экспорт в BMP», «Импорт из Excel», «Очистить», «О программе».
* панель, на которой расположены быстрые кнопки и флажок.

### Описание графических компонентов формы fmArcInput

Форма fmArcInput – это вспомогательная форма для ввода веса добавляемой дуги. Вызывается при установленном режиме главной формы в добавлении дуги, выбранных двух вершинах и отключенном флажке о невзвешенных дугах. Данная форма имеет вид, представленный на рисунке 30.

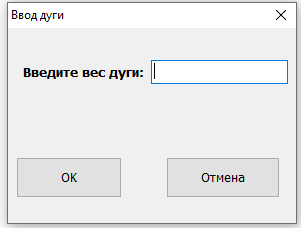


Рисунок 30 – Вид формы fmArcInput

Составляющие формы fmArcInput:

* поле ввода, куда нужно ввести вес добавляемой дуги;
* кнопка «ОК», которая добавляет записывает вес и сообщает главной форме о необходимости добавить дугу;
* кнопка «Отмена», которая закрывает форму без сообщения о добавлении дуги.

### Описание графических компонентов формы fmSearchOutput

Форма fmSearchOutput – это вспомогательная форма для вывода информации о найденном пути. Вызывается в случае, если путь был найден и отображает слещующее: последовательность вершин пути через запятые, длину пути в дугах и единицах веса, а также число посещений вершин при поиске. Данная форма имеет вид, представленный на рисунке 31.

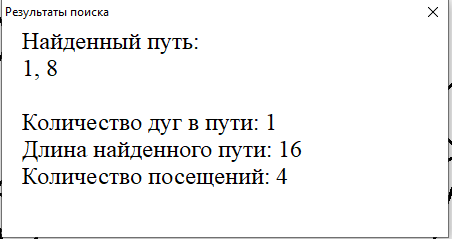


Рисунок 31 – Вид формы fmSearchOutput

Составляющие формы fmArcInput:

* поле ввода, куда нужно ввести вес добавляемой дуги;
* кнопка «ОК», которая добавляет записывает вес и сообщает главной форме о необходимости добавить дугу;
* кнопка «Отмена», которая закрывает форму без сообщения о добавлении дуги.

### Описание графических компонентов формы fmAbout

Форма fmAbout – это вспомогательная форма для вывода информации о программе. Она пытается прочеть файл README.txt, а в случае его отсутствия выводит соответствующее сообщение. Данная форма имеет вид, представленный на рисунке 32.

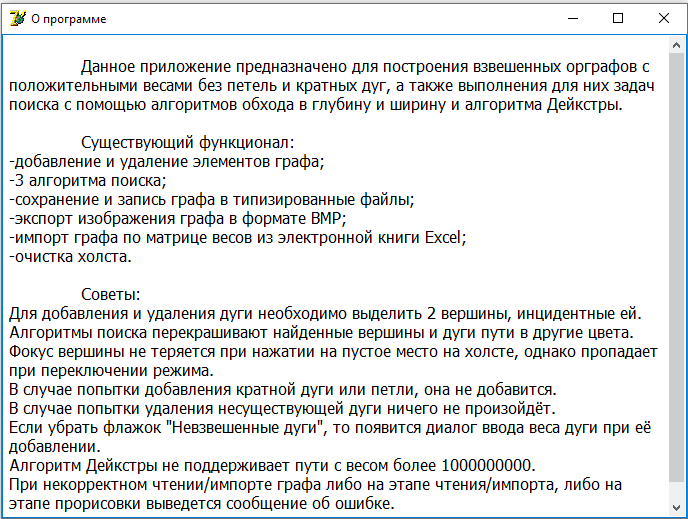


Рисунок 3.32 - Вид формы fmAbout

Составляющие формы fmAbout:

* компонент memo с прочитанным из файла текстом.

# Тестирование и проверка работоспособности программного средства

Для тестирования алгоритмов поиска был подготовлен набор из следующих графов, заданных списками смежности:

Таблица 32 – Граф G1

|  |  |  |
| --- | --- | --- |
|  | Элементы графа | Соответственные веса |
| Вершины | 1 | |
| Соседи вершины 1 | Нет | Нет |

Таблица 33 – Граф G2

|  |  |  |
| --- | --- | --- |
|  | Элементы графа | Соответственные веса |
| Вершины | 1, 2 | |
| Соседи вершины 1 | Нет | Нет |
| Соседи вершины 2 | Нет | Нет |

Таблица 34 – Граф G3

|  |  |  |
| --- | --- | --- |
|  | Элементы графа | Соответственные веса |
| Вершины | 1, 2, 3, 4, 5, 6, 7, 8, 9 | |
| Соседи вершины 1 | 2, 6, 8 | 1, 1, 1 |
| Соседи вершины 2 | 3 | 1 |
| Соседи вершины 3 | 4 | 1 |
| Соседи вершины 4 | 5 | 1 |
| Соседи вершины 5 | 9 | 1 |
| Соседи вершины 6 | 7 | 1 |
| Соседи вершины 7 | 9 | 1 |
| Соседи вершины 8 | 9 | 1 |
| Соседи вершины 9 | Нет | Нет |

Таблица 35 – Граф G4

|  |  |  |
| --- | --- | --- |
|  | Элементы графа | Соответственные веса |
| Вершины | 1, 2, 3, 4, 5, 6, 7, 8, 9 | |
| Соседи вершины 1 | 2, 6, 8 | 1, 1, 1 |
| Соседи вершины 2 | 3 | 1 |
| Соседи вершины 3 | 4 | 1 |
| Соседи вершины 4 | 5 | 1 |
| Соседи вершины 5 | Нет | Нет |
| Соседи вершины 6 | 7 | 1 |
| Соседи вершины 7 | Нет | Нет |
| Соседи вершины 8 | 9 | 1 |
| Соседи вершины 9 | Нет | Нет |

Таблица 36 – Граф G5

|  |  |  |
| --- | --- | --- |
|  | Элементы графа | Соответственные веса |
| Вершины | 1, 2, 3, 4, 5 | |
| Соседи вершины 1 | 2, 4, 5 | 1, 2, 10 |
| Соседи вершины 2 | 3 | 20 |
| Соседи вершины 3 | 4 | 4 |
| Соседи вершины 4 | 5 | 3 |
| Соседи вершины 5 | Нет | Нет |

Таблица 37 – Граф G5

|  |  |  |
| --- | --- | --- |
|  | Элементы графа | Соответственные веса |
| Вершины | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 | |
| Соседи вершины 1 | 2, 3, 4 | 10, 4, 3 |
| Соседи вершины 2 | 5, 6, 9 | 15, 34, 21 |
| Соседи вершины 3 | 2, 6, 7 | 15, 3, 20 |
| Соседи вершины 4 | 7 | 21 |
| Соседи вершины 5 | 8 | 13 |
| Соседи вершины 6 | 9, 10 | 6, 14 |
| Соседи вершины 7 | 10 | 12 |
| Соседи вершины 8 | 11 | 23 |
| Соседи вершины 9 | 8, 11 | 12, 28 |
| Соседи вершины 10 | 11 | 16 |
| Соседи вершины 11 | Нет | Нет |

## Запуск программы

### Тест 1

Таблица 38 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация: | Проверка корректности поведения программы при запуске программы |
| Исходный набор данных: | Запуск программы |
| Ожидаемый результат: | Открытие главной формы |

Продолжение таблицы 38

|  |  |
| --- | --- |
| Полученный результат: |  |

## Добавление элементов

### Тест 2

Таблица 39 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация: | Добавления нескольких вершин |
| Исходный набор данных: | Данные о координатах щелчка по 10 местам холста |
| Ожидаемый результат: | Пустой граф из 10 вершин |
| Полученный результат: |  |

### Тест 3

Таблица 40 – Тест 3

|  |  |
| --- | --- |
| Тестовая ситуация: | Добавление нескольких невзвешенных дуг |
| Исходный набор данных: | Граф-контур из 5 вершин |
| Ожидаемый результат: | Граф-контур из 5 вершин |
| Полученный результат: |  |

### Тест 4

Таблица 41 – Тест 4

|  |  |
| --- | --- |
| Тестовая ситуация: | Попытка добавить кратную дугу |
| Исходный набор данных: | Вершины 1 и 2;  Дуги: (1,2) |
| Ожидаемый результат: | Дуга (2,1) не добавится |
| Полученный результат: |  |

### Тест 5

Таблица 42 – Тест 5

|  |  |
| --- | --- |
| Тестовая ситуация: | Попытка добавить петлю |
| Исходный набор данных: | Вершина 1 |
| Ожидаемый результат: | Дуга (1,1) не добавится |
| Полученный результат: |  |

### Тест 6

Таблица 43 – Тест 6

|  |  |
| --- | --- |
| Тестовая ситуация: | Добавление положительно взвешенных дуг |
| Исходный набор данных: | 4 вершины;  Набор весов: 10, 4, 6, 143000 |
| Ожидаемый результат: | Граф-контур со взвешенными дугами |
| Полученный результат: |  |

### Тест 7

Таблица 44 – Тест 7

|  |  |
| --- | --- |
| Тестовая ситуация: | Добавление дуг с некорректных весов |
| Исходный набор данных: | 4 вершины  Набор весов: 0, -10, -143, test |
| Ожидаемый результат: | Граф-контур со невзвешенными (вес равен 1) дугами |
| Полученный результат: |  |

## Удаление элементов

### Тест 8

Таблица 45 – Тест 8

|  |  |
| --- | --- |
| Тестовая ситуация: | Удаление вершины из головы списка вершин |
| Исходный набор данных: | 4 вершины  Вершина к удалению: 1 |
| Ожидаемый результат: | Граф из 3 вершин, уменьшение номеров всех вершин на 1, удаление инцидентных вершине дуг |
| Полученный результат: |  |

### Тест 9

Таблица 46 – Тест 9

|  |  |
| --- | --- |
| Тестовая ситуация: | Удаление вершины из середины списка |
| Исходный набор данных: | 3 вершины  Вершина к удалению: 2 |
| Ожидаемый результат: | Граф из 2 вершин, уменьшение номера вершины 3, удаление инцидентных вершине 2 дуг |
| Полученный результат: |  |

### Тест 10

Таблица 47 – Тест 10

|  |  |
| --- | --- |
| Тестовая ситуация: | Удаление существующей дуги |
| Исходный набор данных: | Вершины 1 и 2;  Дуги: (1,2)  Дуга к удалению: (1,2) |
| Ожидаемый результат: | Пустой граф из 2 вершин |
| Полученный результат: |  |

### Тест 11

Таблица 48 – Тест 11

|  |  |
| --- | --- |
| Тестовая ситуация: | Удаление несуществующей дуги |
| Исходный набор данных: | Вершины 1 и 2;  Дуги: нет  Дуга к удалению: (1,2) |
| Ожидаемый результат: | Пустой граф из 2 вершин |
| Полученный результат: |  |

## Дополнительные функции приложения

### Тест 12

Таблица 49 – Тест 12

|  |  |
| --- | --- |
| Тестовая ситуация: | Очистка холста |
| Исходный набор данных: | Граф-контур из 14 вершин |
| Ожидаемый результат: | Пустой холст |
| Полученный результат: |  |

### Тест 13

Таблица 50 – Тест 13

|  |  |
| --- | --- |
| Тестовая ситуация: | Запись и чтение не созданного графа |
| Исходный набор данных: | Вершины: нет  Дуги: нет |
| Ожидаемый результат: | Пустой холст |
| Полученный результат: |  |

### Тест 14

Таблица 51 – Тест 14

|  |  |
| --- | --- |
| Тестовая ситуация: | Запись, очистка холста и чтение произвольного графа |
| Исходный набор данных: | Вершины: 1,2,3,4,5,6,7,8;  Дуги: (1, 2), (1, 4), (1, 6), (1, 8), (3, 2), (3, 4), (3, 6), (3, 8), (5, 2), (5, 4), (5, 6), (5, 8), (7, 2), (7, 4), (7, 6),(7, 8) |
| Ожидаемый результат: | Открытие того же графа |
| Полученный результат: |  |

### Тест 15

Таблица 52 – Тест 15

|  |  |
| --- | --- |
| Тестовая ситуация: | Экспорт изображения графа |
| Исходный набор данных: | Вершины: 1,2,3,4,5,6,7,8;  Дуги: (1, 2), (1, 4), (1, 6), (1, 8), (3, 2), (3, 4), (3, 6), (3, 8), (5, 2), (5, 4), (5, 6), (5, 8), (7, 2), (7, 4), (7, 6),(7, 8) |
| Ожидаемый результат: | Сохранение картинки формата BMP с графом по пути, указанном пользователем |
| Полученный результат: | D:\a.bmp |

### Тест 16

Таблица 53 – Тест 16

|  |  |
| --- | --- |
| Тестовая ситуация: | Импорт графа из Excel |
| Исходный набор данных: | Матрица весов: |
| Ожидаемый результат: | Расположение вершин в вершинах правильного треугольника;  Вершины: 1, 2, 3;  Дуги: (1, 2), (1, 3), (3, 2) |
| Полученный результат: |  |

### Тест 17

Таблица 54 – Тест 17

|  |  |
| --- | --- |
| Тестовая ситуация: | Матрица весов графа располагалась не на последней странице |
| Исходный набор данных: | Матрица весов: |
| Ожидаемый результат: | Пустой холст |
| Полученный результат: |  |

### Тест 18

Таблица 55 – Тест 18

|  |  |
| --- | --- |
| Тестовая ситуация: | Матрица весов графа содержит недопустимые дуги |
| Исходный набор данных: | Матрица весов: |
| Ожидаемый результат: | Сообщение об ошибке |
| Полученный результат: |  |

### Тест 19

Таблица 56 – Тест 19

|  |  |
| --- | --- |
| Тестовая ситуация: | Вызвана справка при существующем README.txt |
| Исходный набор данных: | Файл README.txt в папке с приложением |
| Ожидаемый результат: | Вывод содержимого README.txt |
| Полученный результат: |  |

### Тест 20

Таблица 57 – Тест 20

|  |  |
| --- | --- |
| Тестовая ситуация: | Вызвана справка при отсутствующем README.txt |
| Исходный набор данных: | Отсутствие README.txt в папке с приложением |
| Ожидаемый результат: | Вывод сообщение о том, что справка не была найдена |
| Полученный результат: |  |

## Поиск в глубину

### Тест 21

Таблица 58 – Тест 21

|  |  |
| --- | --- |
| Тестовая ситуация: | Путь состоит из 1 вершины |
| Исходный набор данных: | Граф G1 |
| Ожидаемый результат: | Путь: 1 |
| Полученный результат: |  |

### Тест 22

Таблица 59 – Тест 22

|  |  |
| --- | --- |
| Тестовая ситуация: | Пути не существует |
| Исходный набор данных: | Граф G2 |
| Ожидаемый результат: | Путь: не существует |
| Полученный результат: |  |

### Тест 23

Таблица 60 – Тест 23

|  |  |
| --- | --- |
| Тестовая ситуация: | Путь находится без возвратов |
| Исходный набор данных: | Граф G3 |
| Ожидаемый результат: | Путь: 1, 2, 3, 4, 5, 9 |
| Полученный результат: |  |

### Тест 24

Таблица 61 – Тест 24

|  |  |
| --- | --- |
| Тестовая ситуация: | Путь находится с возвратами |
| Исходный набор данных: | Граф G4 |
| Ожидаемый результат: | Путь: 1, 8, 9 |
| Полученный результат: |  |

### Тест 25

Таблица 62 – Тест 25

|  |  |
| --- | --- |
| Тестовая ситуация: | Путь находится с возвратами |
| Исходный набор данных: | Граф G5 |
| Ожидаемый результат: | Путь: 1, 2, 3, 4, 5 |
| Полученный результат: |  |

## Поиск в ширину

### Тест 26

Таблица 63 – Тест 26

|  |  |
| --- | --- |
| Тестовая ситуация: | Путь состоит из 1 вершины |
| Исходный набор данных: | Граф G1 |
| Ожидаемый результат: | Путь: 1 |
| Полученный результат: |  |

### Тест 27

Таблица 64 – Тест 27

|  |  |
| --- | --- |
| Тестовая ситуация: | Пути не существует |
| Исходный набор данных: | Граф G2 |
| Ожидаемый результат: | Путь: нет |
| Полученный результат: |  |

### Тест 28

Таблица 65 – Тест 28

|  |  |
| --- | --- |
| Тестовая ситуация: | Алгоритм выбирает кратчайший из нескольких путей |
| Исходный набор данных: | Граф G3 |
| Ожидаемый результат: | Путь: 1, 8, 9 |
| Полученный результат: |  |

### Тест 29

Таблица 66 – Тест 29

|  |  |
| --- | --- |
| Тестовая ситуация: | Алгоритм выбирает путь с меньшим числом посещений, чем DFS |
| Исходный набор данных: | Граф G4 |
| Ожидаемый результат: | Путь: 1, 8, 9 |
| Полученный результат: |  |

### Тест 30

Таблица 67 – Тест 30

|  |  |
| --- | --- |
| Тестовая ситуация: | Алгоритм выбирает путь с меньшим числом дуг, но не кратчайший |
| Исходный набор данных: | Граф G5 |
| Ожидаемый результат: | Путь: 1, 5 |
| Полученный результат: |  |

## Алгоритм Дейкстры

### Тест 31

Таблица 68 – Тест 31

|  |  |
| --- | --- |
| Тестовая ситуация: | Путь из 1 вершины |
| Исходный набор данных: | Граф G1 |
| Ожидаемый результат: | Путь: 1 |
| Полученный результат: |  |

### Тест 32

Таблица 69 – Тест 32

|  |  |
| --- | --- |
| Тестовая ситуация: | Пути не существует |
| Исходный набор данных: | Граф G2 |
| Ожидаемый результат: | Путь: 1 |
| Полученный результат: |  |

### Тест 33

Таблица 70 – Тест 33

|  |  |
| --- | --- |
| Тестовая ситуация: | Алгоритм Дейкстры вырождается в поиск в ширину |
| Исходный набор данных: | Граф G3 или G4 |
| Ожидаемый результат: | Путь: 1, 8, 9 |
| Полученный результат: |  |

### Тест 34

Таблица 71 – Тест 34

|  |  |
| --- | --- |
| Тестовая ситуация: | Алгоритм Дейкстры находит кратчайший путь не с наименьшим числом дуг |
| Исходный набор данных: | Граф G5 |
| Ожидаемый результат: | Путь: 1, 4, 5 |
| Полученный результат: |  |

### Тест 35

Таблица 72 – Тест 35

|  |  |
| --- | --- |
| Тестовая ситуация: | Алгоритм Дейкстры находит кратчайший путь в произвольной сети |
| Исходный набор данных: | Граф G6 |
| Ожидаемый результат: | Путь: 1, 3, 6, 10, 11 |
| Полученный результат: |  |

# 5 Тестирование работоспособности программного средства

## Минимальные системные требования

Для успешного запуска редактора графа и комфортной работы с ним необходимо соответствие минимальным системным требованиям:

* процессор 800 МГц или выше;
* объем оперативной памяти не менее 256 МБ;
* свободное место на диске не менее 37,1 МБ;
* операционная система Windows 2000 и выше;
* наличие Excel из пакета Microsoft Office 2007 и выше.

## Установка

На установочном диске находится установочный файл с программным средством. После открытия данного пакета на экране появляется окно, представленное на рисунке 5.1. В данном окне пользователь может продолжить установку, нажав на кнопку “Далее”, или отменить установку, нажав на кнопку “Отмена”.

Рисунок 5.1 – Установка (этап 1)

На следующем этапе установки на экране открывается окно, представленное на рисунке 5.2. Здесь пользователь выбирает место папки установки, а также пользователь может заметить, что для загрузки требуется как минимум x МБ свободного дискового пространства для продолжения установки программного средства.

Рисунок 5.2 – Установка (этап 2)

После этого этапа у пользователя есть возможность выбрать дополнительные задачи, которые необходимо выполнить. У установщика данного программного средства этой задачей является добавление значка на Рабочий стол. Данное окно имеет вид, представленный на рисунке 5.3.

Рисунок 5.3 – Установка (этап 3)

После этого этапа подготовка к установке завершается и пользователю отображается предложение установить программное средство на компьютер, показанное на рисунке 5.4.

Рисунок 5.4 – Установка (этап 4)

Об успешном завершении установки пользователя информирует окно, представленное на рисунке 5.5.

Рисунок 5.5 – Установка (этап 5)

## Работа с приложением

После установки и открытия редактора графов на экране появляется окно, представленное на 5.5.

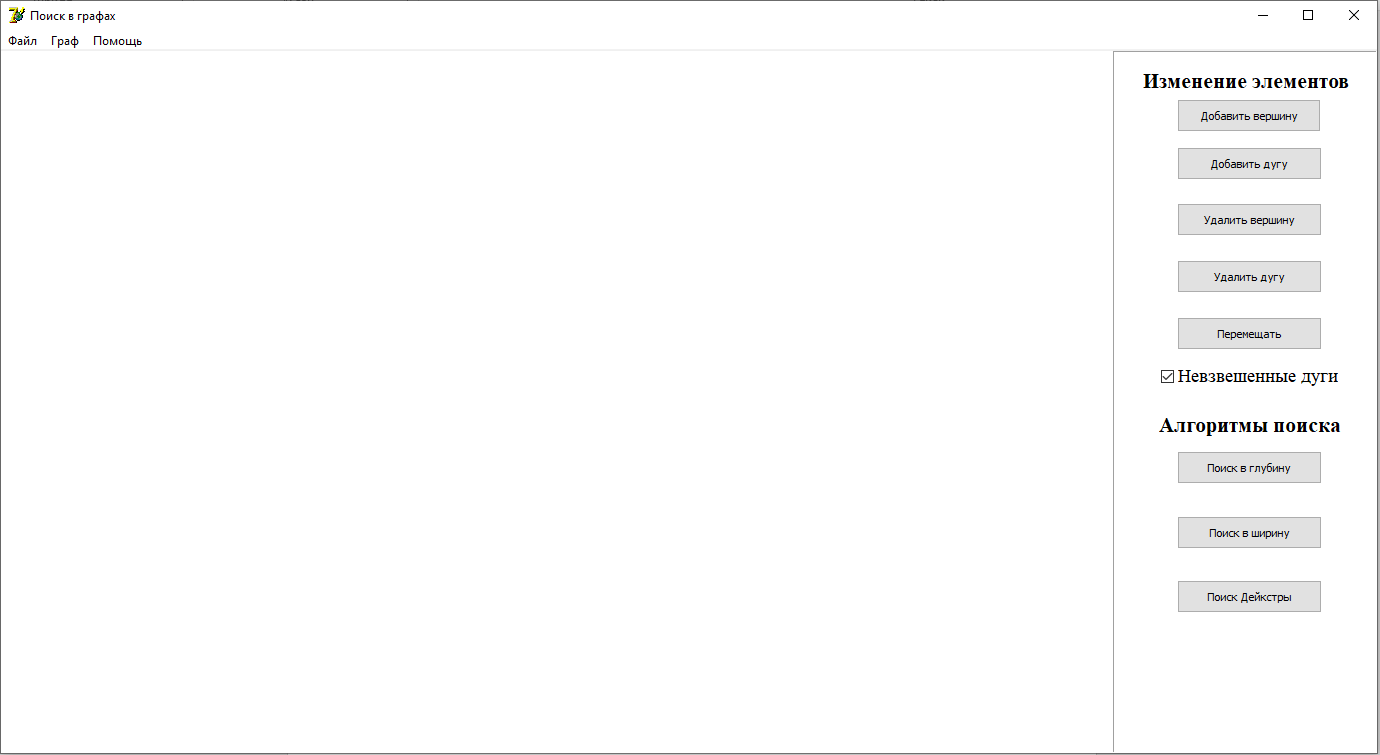


Рисунок 5.5 – Стартовое окно программы

На панели справа пользователь может выбрать одно из действий по изменению структуры графа.

Чтобы добавить новую вершину необходимо нажать на кнопку «Добавить вершину» и кликнуть мышкой по произвольному месту холста. На холсте появится изображение круга с номером в центре, как показано на рисунке 5.6.

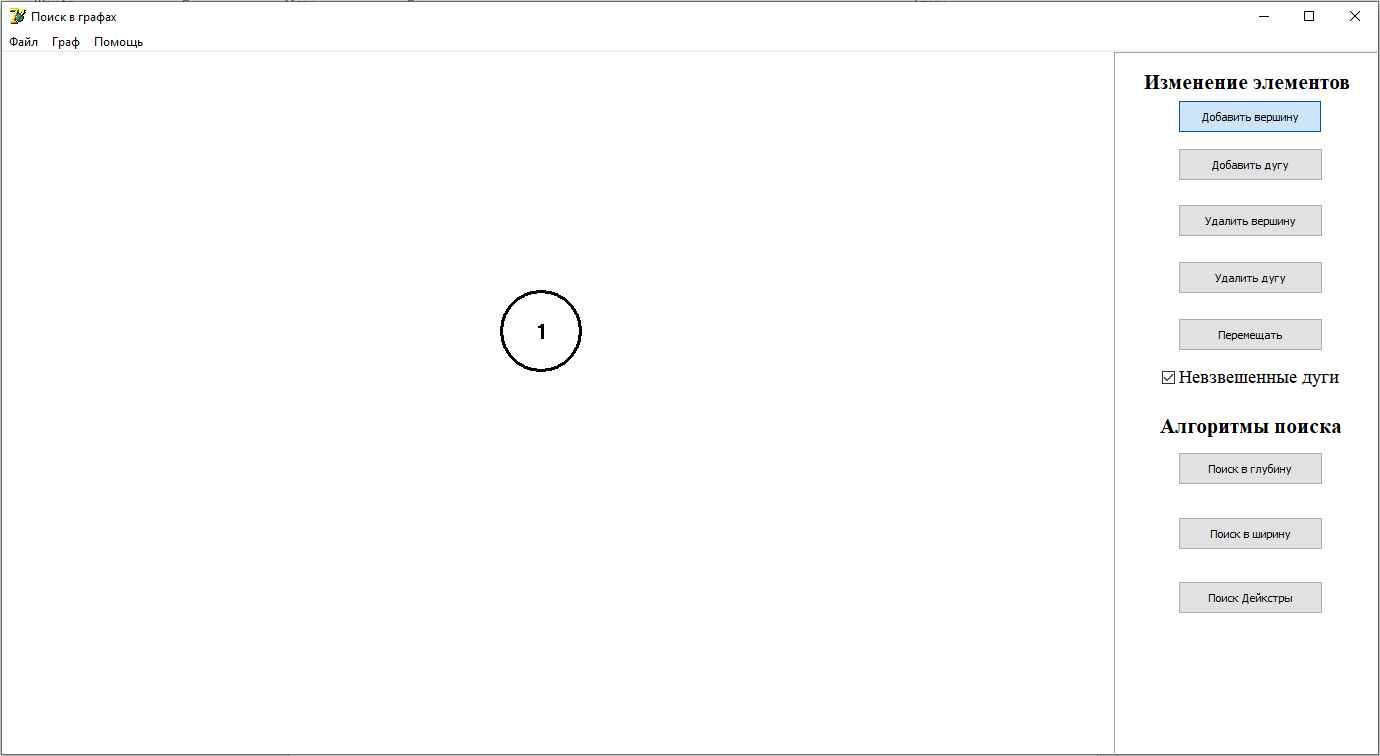


Рисунок 5.6 – Окно программы после добавления вершины

Для удаления вершины необходимо нажать на кнопку «Удалить вершину» и кликнуть мышкой по уже существующим вершинам. После чего на холсте пропадёт выбранная вершина вместе со всеми инцидентыми ей дугами.

Для добавления дуги, удаления дуги и применении алгоритмов поиска необходимо выделить начальную вершину, после чего она изменит свой цвет, как показано на рисунке 5.7.

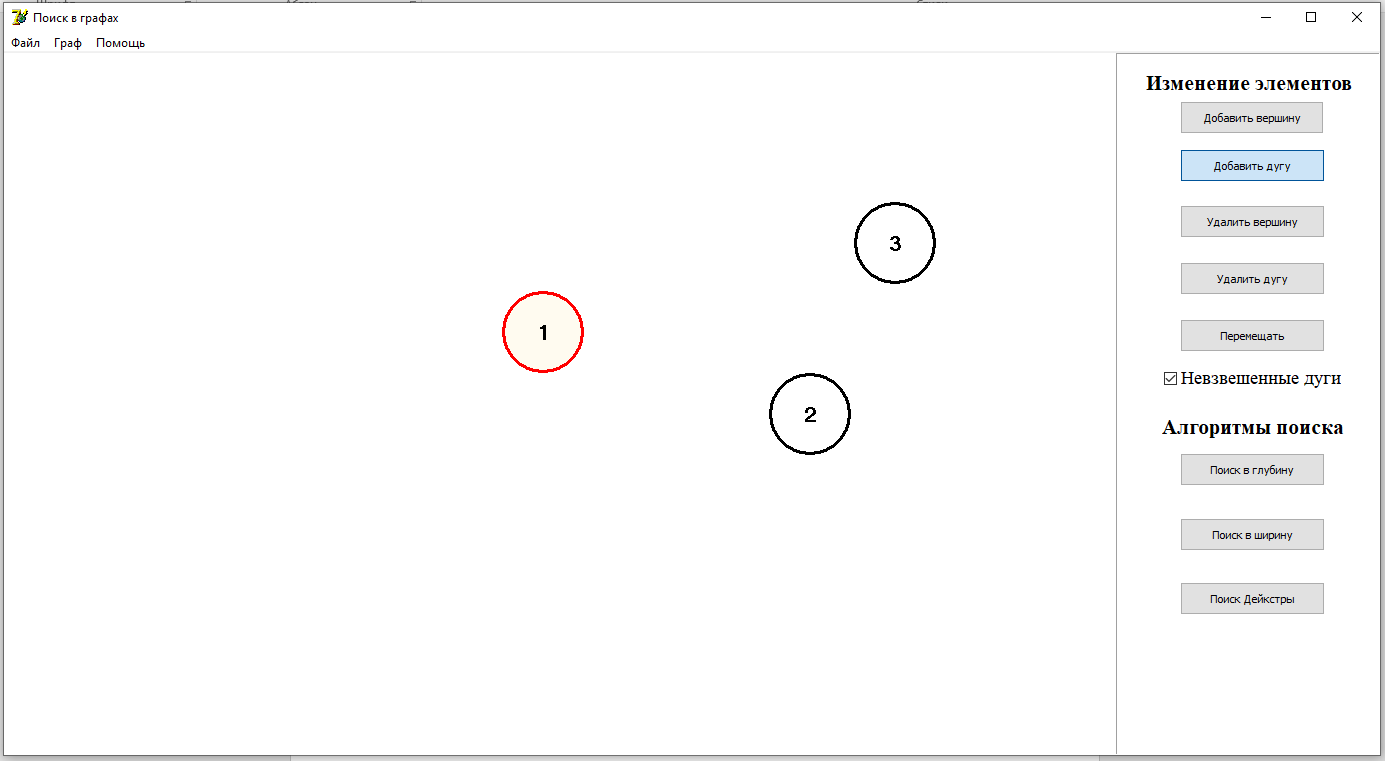


Рисунок 5.7 – Окно программы выделения вершины

После выделения второй вершины при выбранном режиме добавления дуг между вершинами появится дуга, при чём если был введён вес больше 1, то он будет отображён, что показано на рисунке 5.8. Если был выбран режим удаления дуг и между вершинами была дуга, она пропадёт.

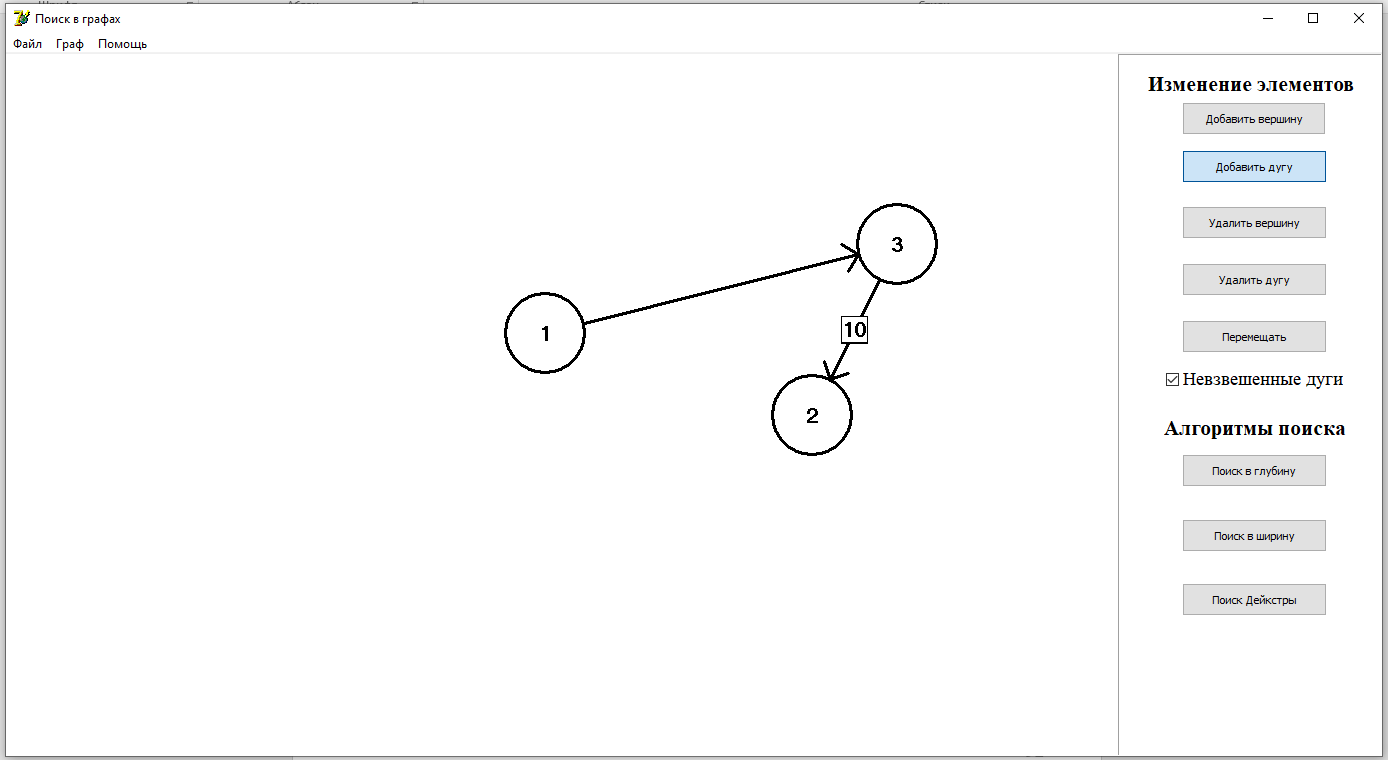


Рисунок 5.8 – Окно программы добавления дуг

При снятии флажка «Невзвешенные дуги» при добавлении дуг будет всплывать окно, предлагающее ввод веса добавляемой дуги, что показано на рисунке 5.9.

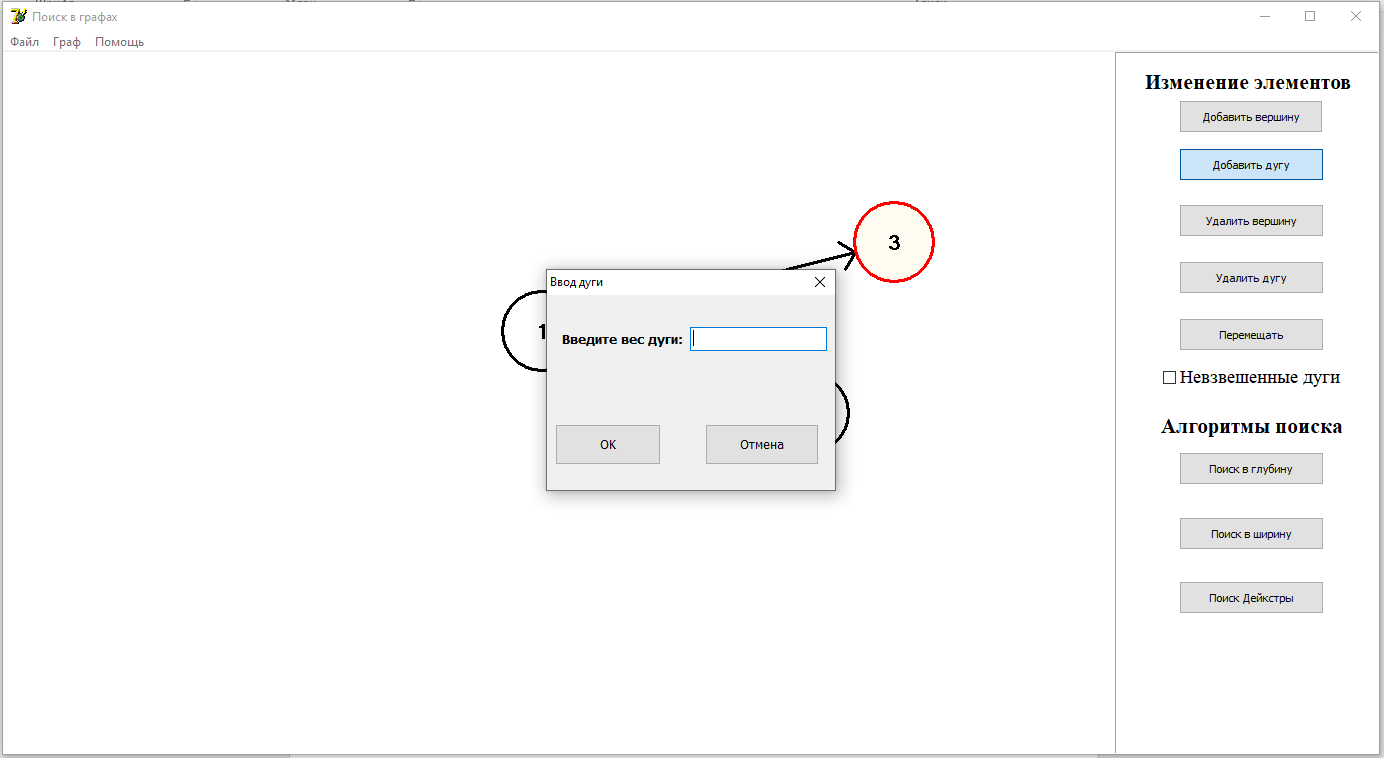


Рисунок 5.9 – Окно ввода веса дуги

Если нажать на кнопку «Перемещать», то пользователь сможет перемащть на холсте выделенную вершину, при чём она перестань быть выделенной, как только он отпустит левую кнопку мыши.

При выборе алгоритма поиска, а также начальной и конечной вершин пути, всплывёт окошко с результатами поиска и вершины с дугами, которые были в пути станут перекрашенными. На рисунке 5.10 показан результат поиска из вершины 2 в 1.

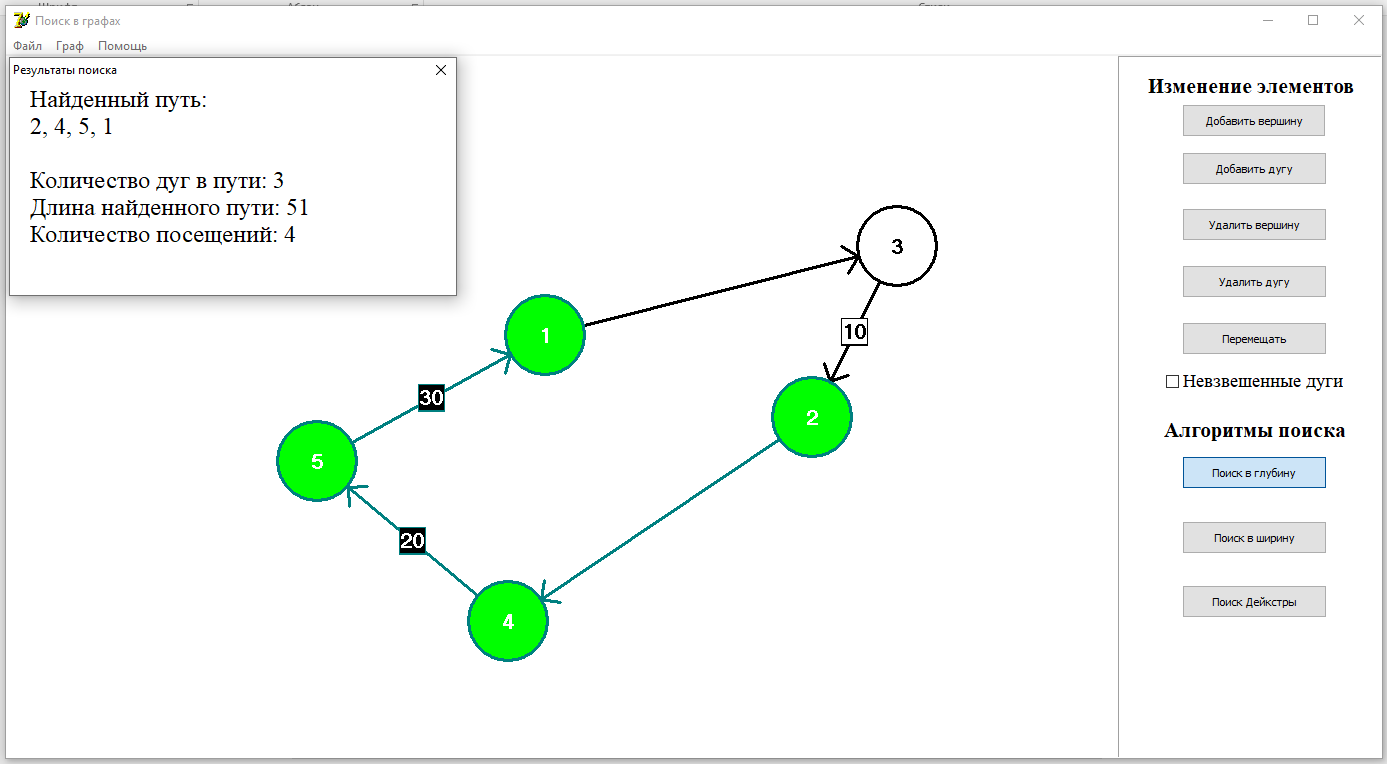


Рисунок 5.10 – Окно программы при поиске пути

Раскраска цветов графа спадёт при выборе новой вершины или переключении режима с помощью одной из кнопок справа.

Пользователь может сохранить граф в типизированных файлов, открыть граф из типизированных файлов, экспортировать его изображение в формате BMP или импортировать его из электронной книги Excel с помощью меню во вкладках «Файл» и «Граф». Во всех случаях появятся диалоговые окна с выбором пути. На рисунке 5.11 показан пример диалогового окна при импорте графа из Excel.

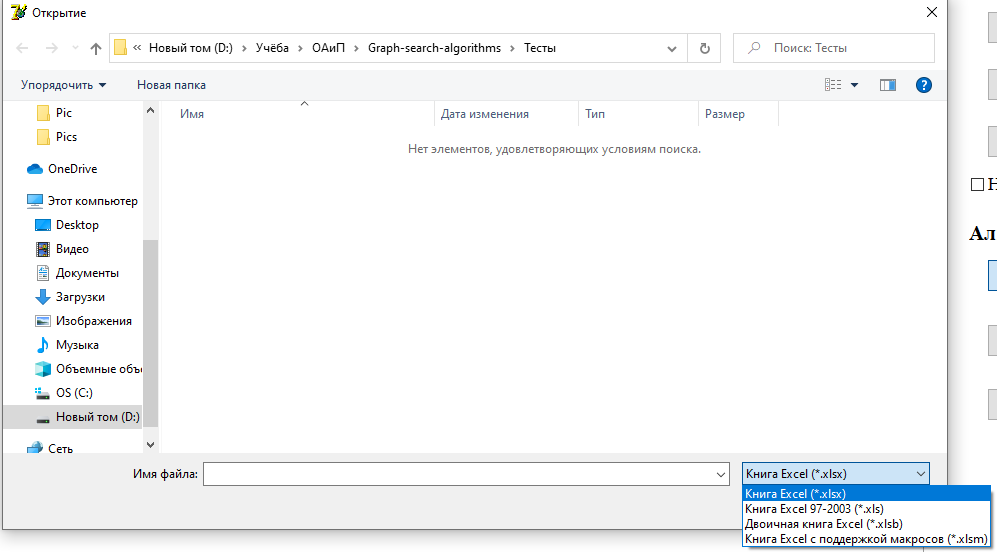


Рисунок 5.11 – Диалоговое окно поиска пути

Во вкладке меню «Помощь» пользователь может выбрать «О программе», после чего появится окно, показанное на рисунке 5.12.

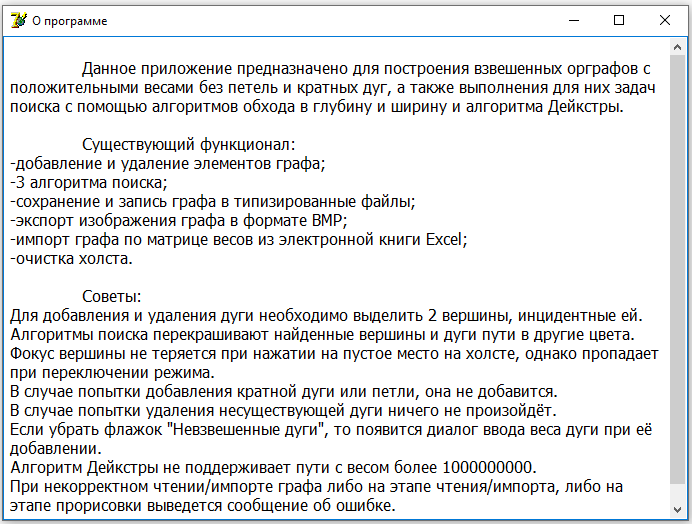


Рисунок 5.11 – Диалоговое окно поиска пути

Во вкладке меню «Граф» пользователь может выбрать «Очистить», после чего холст вновь станет пустым.

Также доступны комбинации клавиш:

* F1 – вызов справки о программе;
* Ctrl+S – сохранение открытого графа;
* Shift+Ctrl+S - сохранение графа по указанному пути;
* Ctrl+O – открытие графа по указанному пути;
* Ctrl+E – очистить холст.

Пользователь может выйти из программы с помощью элемента меню «Выход» во вкладке «Файл».

# Заключение

По итогу работы над курсовым проектом было разработан редактор графов, обладающее графическим интерфейсом для взаимодействия с пользователем, в том числе для создания графов и применении на них алгоритмов поиска.

Приложение прошло все этапы тестирования и продемонстрировало корректную быструю работу, обработав исключительные ситуации и выполнив поиск путей по данным тестовым наборам. Также был проверен вспомогательный функционал, позволяющий сохранение и чтение графа с помощью типизированных файло, импорт графа по матрице весов из электронной книги Excel, экспорт изображения графа, вызов справки.

Протестировав алгоритмы поиска, можно сделать следующие выводы:

* На примере G3 и G4 можно понять, что обход в глубину может быть менее оптимальным при поиске произвольного пути, чем поиск в ширину, если граф содержит много «тупиковых» ветвей, при которых алгоритм будет делать много откатов.
* На примере G3 и G5 можно убедиться, что алгоритм поиска в ширину действительно ищет пути с минимальным числом дуг.
* На примерах G3 и G4 можно увидеть, что алгоритм Дейкстры вырождается в поиск в ширину.
* На примере G6 можно убедиться, что алгоритм Дейкстры действительно ищет пути с минимальным расстоянием.

Данное программное средство имеет минимально необходимый набор функций для полноценного создания оргафов без кратных дуг и петель. Среди преимуществ программы можно отметить интуитивно понятный интерфейс, простоту реализации, а также большой набор справочной информации по данной теме, который помогает даже самым неопытным пользователям разобраться в том, как использовать приложение.

Бесценным результатом курсового проектирования является полученный опыт работы с динамическими структурами данных и файлами. Был получен опыт работы с графическим интерфейсом пользователя (изучение компонентов форм, свойств и методов объектов, событий), обобщены и применены все ранее полученные знания.

В перспективе данное программное средство может быть усовершенствовано: поддержка кратных дуг и петель, поддержка отрицательных весов у дуг, использование более сложного формата для сохранения графа в типизированный файл, реализация других алгоритмов поиска, более усовершенствованный экспорт и импорт графов, оптимизация кода, облегчение читаемости кода за счёт использование парадигмы ООП и т.д.

# Список использованной литературы

[1] Глухова, Л.А. Основы алгоритмизации и программирования: учебное пособие / Л.А. Глухова, В.В. Бахтизин. – Минск: БГУИР, 2003. – 72 с.

[2] Серебряная, Л.В. Структуры и алгоритмы обработки данных: учеб.-метод. пособие / Л.В. Серебряная, И. М. Марина. – Минск: БГУИР, 2013. – 51 с.

[3] Глухова, Л.А. Основы алгоритмизации и программирования. Лабораторный практикум: учеб.-метод. пособие. В 4 ч. Ч. 4/ Л. А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Минск: БГУИР, 2012. – 58 с.

[4] Фленов, М.Е. Библия Delphi. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2011. – 688 с.

[5] Фаронов, В.В. Delphi 6. Учебный курс.-М.: Издатель Молгачева С.В., 2001. – 672 с.

[6] Глухова, Л.А. Основы алгоритмизации и программирования: учебное пособие ч.2 / Л.А. Глухова. – Минск: БГУИР, 2006. – 177 с.

[7] Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы: учебное пособие – М.: Вильямс, 2000. – 384 с

[8] К. А. Сурков, Д. А. Сурков, А. Н. Вальвачев, Ю. М. Четырько Программирование на Delphi, 2009. – 494 с.: ил.

[9] Math.Semestr [Электронный ресурс]. – Режим доступа: https://math.semestr.ru/optim/zeidel.php. – Дата доступа: 01.03.2021.

[10] Mathprofi.ru [Электронный ресурс]. – Режим доступа: http://mathprofi.ru/metod\_gaussa\_dlya\_chainikov.html. – Дата доступа: 28.02.2021.

[11] Old.exponenta [Электронный ресурс]. – Режим доступа: http://old.exponenta.ru/EDUCAT/class/courses/vvm/theme\_5/theory.asp. – Дата доступа: 03.03.2021.

[12] function-x [Электронный ресурс]. – Режим доступа: https://function-x.ru/systems\_kramer.html. – Дата доступа: 25.02.2021.

Приложение А

Исходный код программы (модуль MainUnit)

unit MainUnit;

{

Модуль с описанием главной формы программы,

которая предоставляет функции редактирования

графа.

}

interface

// Подключаемые модули

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics,

Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons,

Menus, GraphSearch, Digraph, DynamicStructures,

GraphDrawing, AboutUnit, ArcInputUnit, SearchOutputUnit;

type

// Тип режима работы с холстом

TCanvasState = (stAddVertice, stAddArc, stDeleteVertice,

stDeleteArc, stMove, stDFS,

stBFS, stDijkstra, stNone);

// Тип основной формы

TfmEditor = class(TForm)

plContainer: TPanel;

btnAddNode: TSpeedButton;

btnAddLinl: TSpeedButton;

btnDeleteNode: TSpeedButton;

btnDeleteLink: TSpeedButton;

btnDFS: TSpeedButton;

btnBFS: TSpeedButton;

btnDijkstra: TSpeedButton;

mmMain: TMainMenu;

nFile: TMenuItem;

nGraph: TMenuItem;

nHelp: TMenuItem;

nSave: TMenuItem;

nExit: TMenuItem;

nClear: TMenuItem;

nAbout: TMenuItem;

nOpen: TMenuItem;

nSaveAs: TMenuItem;

sdVertices: TSaveDialog;

sdArcs: TSaveDialog;

odVertices: TOpenDialog;

odArcs: TOpenDialog;

lbEdit: TLabel;

lbSearch: TLabel;

N1: TMenuItem;

nExportBMP: TMenuItem;

sdExport: TSaveDialog;

sbMove: TSpeedButton;

cbNoWeight: TCheckBox;

nImportExcel: TMenuItem;

odImport: TOpenDialog;

N2: TMenuItem;

pbCanvas: TPaintBox;

procedure fmEditorCreate(Sender: TObject);

procedure fmEditorClose(Sender: TObject; var Action: TCloseAction);

procedure SetCanvasState(Sender: TObject);

procedure pbCanvasClick(Sender: TObject);

procedure pbCanvasMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure pbCanvasMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure pbCanvasMouseMove(Sender: TObject; Shift: TShiftState;

X, Y: Integer);

procedure nOpenClick(Sender: TObject);

procedure nSaveClick(Sender: TObject);

procedure nSaveAsClick(Sender: TObject);

procedure nExitClick(Sender: TObject);

procedure nClearClick(Sender: TObject);

procedure nExportBMPClick(Sender: TObject);

procedure nAboutClick(Sender: TObject);

procedure cbNoWeightClick(Sender: TObject);

procedure pbCanvasPaint(Sender: TObject);

procedure nImportFromExcel(Sender: TObject);

private

FState: TCanvasState; // Переменная состояния

FGraph: TGraph; // Граф для редактирования

FActiveVertice: TPVertice; // Ссылка на активную вершину

procedure StartSearch(State: TCanvasState;

v, u: Integer);

end;

//Раздел объявления переменных

var

fmEditor: TfmEditor;

// fmEditor - Основная форма программы

implementation

{$R \*.dfm}

// Обработчик создания формы

procedure TfmEditor.fmEditorCreate(Sender: TObject);

begin

FState := stNone;

InitializeGraph(FGraph);

FActiveVertice := nil;

end;

// Обработчик закрытия формы

procedure TfmEditor.fmEditorClose(Sender: TObject; var Action: TCloseAction);

begin

DestroyGraph(FGraph); // Освобождение занятой памяти

end;

// Обработчик нажатия на кнопку выбора режима

procedure TfmEditor.SetCanvasState(Sender: TObject);

begin

if FGraph.isPainted then

begin

MakePassive(FGraph);

pbCanvas.Invalidate;

end

else if FActiveVertice <> nil then

begin

FActiveVertice.Style := stPassive;

FActiveVertice := nil;

pbCanvas.Invalidate;

end;

if (Sender as TSpeedButton).Down then

FState := TCanvasState((Sender as TSpeedButton).Tag)

else

FState := stNone;

end;

// Обработчик клика ЛКМ на холст

procedure TfmEditor.pbCanvasClick(Sender: TObject);

var

Pos: TPoint;

SelectedVertice: TPVertice;

mrInput: TModalResult;

// Pos - Координата курсора мыши

// SelectedVertice - Ссылка на найденную вершину

// mrInput - Результат вызова модальной формы

begin

if FGraph.isPainted then

MakePassive(FGraph);

Pos := ScreenToClient(Mouse.CursorPos);

// Перебор переменной состояния

case FState of

// Добавление вершины

stAddVertice:

AddVertice(FGraph, Pos);

// Удаление вершины

stDeleteVertice:

begin

SelectedVertice := GetByPoint(FGraph, Pos);

if SelectedVertice <> nil then

DeleteVertice(FGraph, SelectedVertice.Number);

end;

// Действие по двум вершинам

stAddArc, stDeleteArc, stDFS, stBFS, stDijkstra:

begin

SelectedVertice := GetByPoint(FGraph, Pos);

if (SelectedVertice <> nil) and

(FActiveVertice = nil) then

begin

FActiveVertice := SelectedVertice;

FActiveVertice.Style := stActive;

SelectedVertice := nil;

pbCanvas.Invalidate;

end;

if (FActiveVertice = nil) or (SelectedVertice = nil)

then

Exit;

case FState of

// Добавление дуги

stAddArc:

begin

if not cbNoWeight.Checked then

mrInput := fmArcInput.ShowModal

else

mrInput := mrOk;

if mrInput = mrOk then

AddArc(FGraph, FActiveVertice.Number,

SelectedVertice.Number,

fmArcInput.Weight);

end;

// Удаление дуги

stDeleteArc:

DeleteArc(FGraph, FActiveVertice.Number,

SelectedVertice.Number);

// Алгоритмы поиска

stDFS, stBFS, stDijkstra:

StartSearch(FState, FActiveVertice.Number,

SelectedVertice.Number);

end; // Конец case

// Сброс начальной вершины

if FActiveVertice.Style = stActive then

FActiveVertice.Style := stPassive;

FActiveVertice := nil;

end; // Конец действий по 2 вершинам

end; // Конец case

if FState <> stNone then

pbCanvas.Invalidate;

end;

// Обработчик нажатия ЛКМ на холст

procedure TfmEditor.pbCanvasMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

var

Pos: TPoint;

// Pos - Координата курсора мыши

begin

if FState = stMove then

begin

Pos := ScreenToClient(Mouse.CursorPos);

FActiveVertice := GetByPoint(FGraph, Pos);

if FActiveVertice <> nil then

FActiveVertice.Style := stActive;

end;

end;

// Обработчик отпускания мыши после нажатия на холст

procedure TfmEditor.pbCanvasMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

if (FState = stMove) and (FActiveVertice <> nil) then

begin

FActiveVertice.Style := stPassive;

FActiveVertice := nil;

pbCanvas.Invalidate;

end;

end;

// Обработчик движения мыши по холсту

procedure TfmEditor.pbCanvasMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);

var

Pos: TPoint;

// Pos - Координата курсора мыши

begin

if (FState = stMove) and (FActiveVertice <> nil) then

begin

Pos := ScreenToClient(Mouse.CursorPos);

FActiveVertice.Center := Pos;

pbCanvas.Invalidate;

end;

end;

// Обработчик нажатия на элемент меню 'Открытие'

procedure TfmEditor.nOpenClick(Sender: TObject);

var

NewGraph: TGraph;

// NewGraph - Копия графа для открытия

begin

odVertices.FileName := '';

odArcs.FileName := '';

if not(odVertices.Execute and odArcs.Execute) then

Exit;

// Чтение графа

try

OpenGraph(NewGraph,odVertices.FileName,

odArcs.FileName);

DestroyGraph(FGraph);

FGraph := NewGraph;

pbCanvas.Invalidate;

sdVertices.FileName := odVertices.FileName;

sdArcs.FileName := odArcs.FileName;

except

DestroyGraph(NewGraph);

MessageBox(fmEditor.Handle,'Ошибка при открытии графа',

'Ошибка', MB\_OK);

end;

end;

// Обработчик нажатия на элемент меню 'Сохранение'

procedure TfmEditor.nSaveClick(Sender: TObject);

begin

// Переход к "сохранению как"

if (sdVertices.FileName = '') or (sdArcs.FileName = '') then

begin

nSaveAsClick(Sender);

Exit

end;

SaveGraph(FGraph, sdVertices.FileName, sdArcs.FileName);

end;

// Обработчик нажатия на элемент меню 'Сохранение как'

procedure TfmEditor.nSaveAsClick(Sender: TObject);

begin

sdVertices.FileName := '';

sdArcs.FileName := '';

if sdVertices.Execute and sdArcs.Execute then

nSaveClick(Sender);

end;

// Обработчик нажатия на элемент меню 'Выход'

procedure TfmEditor.nExitClick(Sender: TObject);

begin

Close;

end;

// Обработчик нажатия на элемент меню 'Очистить'

procedure TfmEditor.nClearClick(Sender: TObject);

begin

DestroyGraph(FGraph);

InitializeGraph(FGraph);

pbCanvas.Invalidate;

end;

// Обработчик нажатия на элемент меню 'Экспорт'

procedure TfmEditor.nExportBMPClick(Sender: TObject);

var

Bitmap: TBitmap;

begin

sdExport.FileName := '';

if sdExport.Execute then

begin

Bitmap := TBitmap.Create;

// Рисование на холсте сохраняемого изображения

try

Bitmap.SetSize(pbCanvas.Width, pbCanvas.Height);

RedrawGraph(Bitmap.Canvas,

Bitmap.Width, Bitmap.Height, FGraph);

Bitmap.SaveToFile(sdExport.FileName);

finally

Bitmap.Free;

end;

end;

end;

// Обработчик нажатия на элемент меню 'Импорт'

procedure TfmEditor.nImportFromExcel(Sender: TObject);

var

NewGraph: TGraph;

// NewGraph - Копия графа для открытия

begin

odImport.FileName := '';

if odImport.Execute then

begin

try

ImportGraph(NewGraph, odImport.FileName);

MakeRegPolygon(NewGraph,

pbCanvas.Width, pbCanvas.Height);

DestroyGraph(FGraph);

FGraph := NewGraph;

pbCanvas.Invalidate;

except

on E: Exception do

begin

DestroyGraph(NewGraph);

MessageBox(Handle, PWideChar(E.Message),

'Ошибка', MB\_OK);

end;

end;

end;

// Обработчик нажатия на элемент меню 'О программе'

procedure TfmEditor.nAboutClick(Sender: TObject);

begin

fmAbout.Show;

end;

// Обработчик нажатия на checkbox

procedure TfmEditor.cbNoWeightClick(Sender: TObject);

begin

fmArcInput.Weight := 1;

end;

// Метод перерисовки dirty rectangle формы

procedure TfmEditor.pbCanvasPaint(Sender: TObject);

begin

try

RedrawGraph(pbCanvas.Canvas,

pbCanvas.Width, pbCanvas.Height, FGraph);

except

DestroyGraph(FGraph);

InitializeGraph(FGraph);

end;

end;

// Метод запуска одного из алгоритмов поиска

procedure TfmEditor.StartSearch(State: TCanvasState; v, u: Integer);

var

Weights: TWeightMatrix;

Info: TSearchInfo;

// Weights - Матрица весов графа

// Info - Информация о поиске

begin

Weights := ToWeightMatrix(FGraph);

// Выбор алгоритма поиска

case State of

stDFS:

Info := DFS(Weights, v, u);

stBFS:

Info := BFS(Weights, v, u);

stDijkstra:

Info := Dijkstra(Weights, v, u);

end;

// Результат о достижимости конечной вершины

if Info.Path <> nil then

begin

MakeVisited(FGraph, Info.Path);

fmSearchOutput.Info := Info;

fmSearchOutput.Show;

end

else

MessageBox(fmEditor.Handle, 'Путь не найден.',

'Информация о пути', MB\_OK);

end;

end.

Приложение Б

Исходный код программы (модуль ArcInputUnit)

unit ArcInputUnit;

{

Модуль с описанием формы ввода веса дуги,

которая появляется при включенных взвешенных

дугах при добавлении новой дуги.

}

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls;

type

// Тип вспомогательной формы

TfmArcInput = class(TForm)

leWeight: TLabeledEdit;

btnOK: TButton;

btnCancel: TButton;

procedure btnOKClick(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure FormShow(Sender: TObject);

public

Weight: Integer; // Поле с введённым весом

end;

var

fmArcInput: TfmArcInput;

// fmArcInput - Вспомогательная форма

implementation

{$R \*.dfm}

{ TfmArcInput }

// Обработчик нажатия на кнопку

procedure TfmArcInput.btnOKClick(Sender: TObject);

var

ErrCode: Integer;

// ErrCode - Признак ошибки при преобразовании

begin

Val(leWeight.Text, Weight, ErrCode);

if (ErrCode <> 0) or (Weight <= 0) then

Weight := 1;

leWeight.Text := '';

end;

// Обработчик создания формы

procedure TfmArcInput.FormCreate(Sender: TObject);

begin

Weight := 1;

end;

// Обработчик показа формы

procedure TfmArcInput.FormShow(Sender: TObject);

begin

leWeight.SetFocus;

end;

end.

Приложение В

Исходный код программы (модуль SearchOutputUnit)

unit SearchOutputUnit;

{

Модуль с описанием формы вывода пути,

которая появляется при нахождении пути

с помощью одного из алгоритмов.

}

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, GraphSearch;

type

// Тип вспомогательной формы

TfmSearchOutput = class(TForm)

lbResults: TLabel;

procedure FormShow(Sender: TObject);

public

Info: TSearchInfo; // Информация о пути

end;

var

fmSearchOutput: TfmSearchOutput;

// fmSearchOutput - Вспомогательная форма

implementation

{$R \*.dfm}

// Обработчик показа формы

procedure TfmSearchOutput.FormShow(Sender: TObject);

const

NL = #13#10;

// NL - Переход на новую строку

var

ResString: String;

// ResString - Строка для форматирования

begin

Info.PathString := NL + Info.PathString;

ResString := 'Найденный путь: %s' + NL + NL;

ResString := ResString + 'Количество дуг в пути: %d' + NL;

ResString := ResString + 'Длина найденного пути: %d' + NL;

ResString := ResString + 'Количество посещений: %d' + NL;

with Info do

lbResults.Caption := Format(ResString, [PathString, ArcsCount, Distance,

VisitsCount]);

end;

end.

Приложение Г

Исходный код программы (модуль AboutUnit)

unit AboutUnit;

{

Модуль с описанием формы вызова окна

'О программе', при котором читается

и выводится на компонент формы

содержимое текстового файла

README.txt.

}

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils,

System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs,

Vcl.ExtCtrls, Vcl.StdCtrls;

type

// Тип вспомогательной формы

TfmAbout = class(TForm)

memText: TMemo;

procedure FormCreate(Sender: TObject);

end;

var

fmAbout: TfmAbout;

// fmAbout - Вспомогательная форма

implementation

{$R \*.dfm}

// Обработчик создания формы

procedure TfmAbout.FormCreate(Sender: TObject);

const

FileName = 'README.txt';

NL = #13#10;

// FileName - Имя файла со справкой

// NL - Переход на новую строку

var

fText: TextFile;

Line: String;

// fText - Текстовый файл

// Line - Прочитанная строка

begin

try

System.Assign(fText, FileName);

Reset(fText);

while not Eof(fText) do

begin

ReadLn(fText, Line);

memText.Lines.Add(Utf8ToAnsi(Line))

end;

CloseFile(fText);

except

memText.Lines.Add(Format

('Файл %s не был найден.', [FileName]));

end;

end;

end.

Приложение Д

Исходный код программы (модуль Digraph)

unit Digraph;

{

Модуль с типами и подпрограммами

для работой со структурой графа.

}

interface

uses System.Types, System.SysUtils, Math, ComObj, Variants,

ActiveX;

type

// Тип стиля раскраски вершины

TVerticeStyle = (stPassive, stActive, stVisited);

// Тип списка смежности графа

TPNeighbour = ^TNeighbour;

TNeighbour = record

Number: Integer;

Weight: Integer;

isVisited: Boolean;

Next: TPNeighbour;

end;

// Тип вершины графа

TPVertice = ^TVertice;

TVertice = record

Number: Integer;

Center: TPoint;

OutDeg: Integer;

Style: TVerticeStyle;

Head: TPNeighbour;

Next: TPVertice;

end;

// Тип ориентированного графа

TGraph = record

Head: TPVertice;

Tail: TPVertice;

Order: Integer;

isPainted: Boolean;

R: Integer;

end;

// Подпрограмма инициализации графа

procedure InitializeGraph(var Graph: TGraph);

// Подпрограмма удаления графа

procedure DestroyGraph(var Graph: TGraph);

// Подпрограмма добавления вершины в граф

procedure AddVertice(var Graph: TGraph; C: TPoint);

// Подпрограмма добавления дуги в граф

procedure AddArc(var Graph: TGraph; v, u, w: Integer);

// Подпрограмма удаления вершины из графа

procedure DeleteVertice(var Graph: TGraph; v: Integer);

// Подпрограмма удаления дуги из графа

procedure DeleteArc(var Graph: TGraph; v, u: Integer);

// Подпрограмма получения вершины по её номеру

function GetByNumber(const Graph: TGraph; v: Integer):

TPVertice;

// Подпрограмма получения вершины по точке на холсте

function GetByPoint(const Graph: TGraph; P: TPoint):

TPVertice;

// Подпрограмма открытия графа из типизированных файлов

procedure OpenGraph(var Graph: TGraph; VerFileName,

ArcFileName: String);

// Подпрограмма сохранения графа в типизированные файлы

procedure SaveGraph(const Graph: TGraph; VerFileName, ArcFileName: String);

// Подпрограмма экспорта графа из экселя

procedure ImportGraph(var Graph: TGraph; ExcelFileName: String);

implementation

procedure DestroyAdjList(var Head: TPNeighbour);

var

Neighbour: TPNeighbour;

// Neighbour - Ссылка на удаляемого соседа

begin

// Цикл А1. Проход по списку смежности

while Head <> nil do

begin

Neighbour := Head;

Head := Head.Next;

Dispose(Neighbour);

end; // Конец А1

end;

procedure InitializeGraph(var Graph: TGraph);

begin

with Graph do

begin

Head := nil;

Tail := nil;

Order := 0;

isPainted := false;

R := 40;

end;

end;

procedure DestroyGraph(var Graph: TGraph);

var

Vertice: TPVertice;

// Vertice - Ссылка на удаляемую вершину

begin

// Цикл А1. Проход по списку вершин

while Graph.Head <> nil do

begin

Vertice := Graph.Head;

DestroyAdjList(Vertice.Head);

Graph.Head := Graph.Head.Next;

Dispose(Vertice);

end; // Конец А1

end;

// Подпрограмма проверки смежности вершин

function IsNeighbour(const Graph: TGraph; Vertice: TPVertice;

u: Integer): Boolean;

var

Neighbour: TPNeighbour;

// Neighbour - Ссылка на сосед с номером u

begin

Result := false;

// AddArc может передать nil во время чтения из файла

if Vertice <> nil then

Neighbour := Vertice.Head

else

Neighbour := nil;

// Цикл А1. Проход по списку смежности

while not Result and (Neighbour <> nil) do

begin

Result := Neighbour.Number = u;

Neighbour := Neighbour.Next;

end; // Конец А1

end;

procedure AddVertice(var Graph: TGraph; C: TPoint);

var

Vertice: TPVertice;

// Vertice - Ссылка на добавляемую вершину

begin

Inc(Graph.Order); // Увеличение порядка

// Инициализация новой вершины

New(Vertice);

with Vertice^ do

begin

Center := C;

Number := Graph.Order;

Head := nil;

Next := nil;

OutDeg := 0;

Style := stPassive;

end;

// Запись новой вершины

if Graph.Head = nil then

Graph.Head := Vertice

else

Graph.Tail.Next := Vertice;

Graph.Tail := Vertice;

end;

procedure AddArc(var Graph: TGraph; v, u, w: Integer);

var

Vertice, AdjVertice: TPVertice;

Neighbour: TPNeighbour;

isIncorrect: Boolean;

// Vertice - Ссылка на вершину v

// AdjVertice - Ссылка на вершину u

// Neighbour - Ссылка на добавляемого соседа

// isIncorrect - Флаг об отсутствии кратных дуг и петель

begin

// Получение ссылок на вершины v и u

Vertice := GetByNumber(Graph, v);

AdjVertice := GetByNumber(Graph, u);

// Проверка корректности графа в случае добавления дуги

isIncorrect := (v = u) or IsNeighbour(Graph, Vertice, u) or

IsNeighbour(Graph, AdjVertice, v);

if not isIncorrect then

begin

Inc(Vertice.OutDeg);

// Добавление соседа в список смежности

New(Neighbour);

with Neighbour^ do

begin

Number := u;

Weight := w;

isVisited := false;

Next := Vertice.Head;

end;

Vertice.Head := Neighbour;

end;

end;

procedure DeleteVertice(var Graph: TGraph; v: Integer);

var

Vertice, PrVertice: TPVertice;

Neighbour, PrNeighbour: TPNeighbour;

Exists: Boolean;

// Vertice - Ссылка на текущего вершину

// PrVertice - Ссылка на вершину перед Vertice

// Neighbour - Ссылка на текущего соседа

// PrNeighbour - Ссылка на соседа перед Neighbour

// Exists - Флаг о наличии вершины

begin

// Удаление вершины

if v <> Graph.Head.Number then

begin

PrVertice := GetByNumber(Graph, v - 1);

Exists := (PrVertice <> nil) and (PrVertice.Next <> nil);

if Exists then

begin

Vertice := PrVertice.Next;

PrVertice.Next := Vertice.Next;

end;

end

else

begin

// Удаляется голова списка

Vertice := Graph.Head;

Graph.Head := Vertice.Next;

Exists := true;

end;

if Exists then

begin

Dec(Graph.Order);

// Освобождение памяти

DestroyAdjList(Vertice.Head);

Dispose(Vertice);

// Цикл А1. Проход по вершинам графа

Vertice := Graph.Head;

while Vertice <> nil do

begin

// Уменьшение номеров вершин

if Vertice.Number > v then

Dec(Vertice.Number);

// Изменение хвоста списка вершин

if Vertice.Next = nil then

Graph.Tail := Vertice;

// Цикл А2. Проход по соседям вершины

PrNeighbour := nil;

Neighbour := Vertice.Head;

while Neighbour <> nil do

begin

// Удаление соседа текущей вершины

if Neighbour.Number = v then

begin

if PrNeighbour = nil then

Vertice.Head := Neighbour.Next

else

PrNeighbour.Next := Neighbour.Next;

Dispose(Neighbour);

end

else if Neighbour.Number > v then

Dec(Neighbour.Number); // Уменьшение номера соседа

// Переход к следующему соседу

PrNeighbour := Neighbour;

Neighbour := Neighbour.Next;

end; // Конец А2

// Переход к следующей вершине

Vertice := Vertice.Next;

end; // Конец А1

end; // Конец if

end;

procedure DeleteArc(var Graph: TGraph; v, u: Integer);

var

Vertice: TPVertice;

Neighbour, PrNeighbour: TPNeighbour;

Exists: Boolean;

// Vertice - Ссылка на вершину-начало дуги

// Neighbour - Ссылка на искомый для удаления сосед

// PrNeighbour - Ссылка на соседа перед Neighbour

// Exists - Флаг о наличии дуги

begin

// Получение начала дуги

Vertice := GetByNumber(Graph, v);

Exists := IsNeighbour(Graph, Vertice, u);

if Exists then

begin

// Цикл А1. Проход по списку смежности

PrNeighbour := nil;

Neighbour := Vertice.Head;

while Neighbour <> nil do

begin

// Сравнение номеров текущего и удаляемого соседа

if u = Neighbour.Number then

begin

Dec(Vertice.OutDeg);

// Удаление соседа из списка смежности

if u = Vertice.Head.Number then

Vertice.Head := Neighbour.Next

else

PrNeighbour.Next := Neighbour.Next;

Dispose(Neighbour);

Neighbour := nil;

end // Конец if

else

begin

// Переход к следующему соседу

PrNeighbour := Neighbour;

Neighbour := Neighbour.Next;

end; // Конец else

end; // Конец А1

end; // Конец if

end;

function GetByNumber(const Graph: TGraph; v: Integer):

TPVertice;

begin

// Цикл А1. Проход по списку вершин

Result := Graph.Head;

while (Result <> nil) and (Result.Number <> v) do

Result := Result.Next;

end;

function GetByPoint(const Graph: TGraph; P: TPoint): TPVertice;

var

Vertice: TPVertice;

// Vertice - Ссылка на текущую вершину графа

begin

// Цикл А1. Поиск последней вершины с близкими координатами

Result := nil;

Vertice := Graph.Head;

while Vertice <> nil do

begin

// Проверка принадлежности окружности радиуса Graph.R

if P.Distance(Vertice.Center) <= Graph.R then

Result := Vertice;

Vertice := Vertice.Next;

end; // Конец А1

end;

procedure OpenGraph(var Graph: TGraph; VerFileName, ArcFileName: String);

var

VerFile: File of TVertice;

ArcFile: File of TNeighbour;

Vertice: TPVertice;

Neighbour: TPNeighbour;

v: Integer;

// VerFile - Типизированный файл вершин

// ArcFile - Типизированный файл соседей

// Vertice - Ссылка на текущую вершину

// Neighbour - Ссылка на текущего соседа

// v - Параметр цикла по соседям

begin

// Подготовка файлов

Assign(VerFile, VerFileName);

Assign(ArcFile, ArcFileName);

Reset(VerFile);

Reset(ArcFile);

// Инициализация графа, указателя на вершину и соседа

InitializeGraph(Graph);

New(Vertice);

New(Neighbour);

// Цикл А1. Проход по файлу вершин

while not Eof(VerFile) do

begin

// Чтение очередной вершины

Read(VerFile, Vertice^);

AddVertice(Graph, Vertice.Center);

// Цикл А2. Частичный проход по файлу соседей

for v := 1 to Vertice.OutDeg do

begin

// Чтение очередного соседа

Read(ArcFile, Neighbour^);

AddArc(Graph, Vertice.Number, Neighbour.Number,

Neighbour.Weight);

end; // Конец А2

end; // Конец А1

// Освобождение памяти

Dispose(Vertice);

Dispose(Neighbour);

// Закрытие файлов

CloseFile(VerFile);

CloseFile(ArcFile);

end;

procedure SaveGraph(const Graph: TGraph; VerFileName, ArcFileName: String);

var

VerFile: File of TVertice;

ArcFile: File of TNeighbour;

Vertice: TPVertice;

Neighbour: TPNeighbour;

// VerFile - Типизированный файл вершин

// ArcFile - Типизированный файл соседей

// Vertice - Ссылка на текущую вершину

// Neighbour - Ссылка на текущего соседа

begin

// Подготовка файлов

Assign(VerFile, VerFileName);

Assign(ArcFile, ArcFileName);

Rewrite(VerFile);

Rewrite(ArcFile);

// Цикл А1. Проход по вершинам

Vertice := Graph.Head;

while Vertice <> nil do

begin

Write(VerFile, Vertice^);

// Цикл А2. Проход по соседям

Neighbour := Vertice.Head;

while Neighbour <> nil do

begin

Write(ArcFile, Neighbour^);

Neighbour := Neighbour.Next;

end; // Конец А1

Vertice := Vertice.Next;

end; // Конец А2

// Закрытие файлов

CloseFile(VerFile);

CloseFile(ArcFile);

end;

procedure ImportGraph(var Graph: TGraph; ExcelFileName: String);

const

ExcelApp = 'Excel.Application';

VerticeCenter: TPoint = (x: 0; y: 0);

var

MyExcel: Variant;

Sheet, Weights: OLEVariant;

CLSID: TCLSID;

i, j, Rows, Cols: Integer;

w, ww: Integer;

isIncorrect: Boolean;

// MyExcel - Объект Excel

// Sheet - Объект активного листа книги

// Weights - Объект диапазона ячеек

// CLSID - Идентификатор, определяющий тип COM-объекта

// i, j - Параметры циклов по строкам и столбцам диапазона

// Rows, Cols - Количество строк и столбцов диапазона

// w, ww - Симметричные в диапазоне ячейки

// VerticeCenter - Центр очередной вершины на холсте

// isIncorrect - Флаг о существовании недопустимых дуг

begin

// Проверка наличия Excel на компьютере

if CLSIDFromProgID(PWideChar(WideString(ExcelApp)),

CLSID) = S\_OK then

begin

// Открытие приложения, книги и листа

CoInitialize(nil);

MyExcel := CreateOleObject(ExcelApp);

MyExcel.WorkBooks.Open(ExcelFileName);

Sheet := MyExcel.ActiveWorkBook.ActiveSheet;

// Получение используемого диапазона ячеек

Rows := Sheet.UsedRange.Rows.Count;

Cols := Sheet.UsedRange.Columns.Count;

if Rows <> Cols then

raise Exception.Create

('Матрица расстояний не была квадратной.');

Weights := Sheet.UsedRange.Value;

// Заполнение графа

try

// Цикл А1. Проход по строкам диапазона

InitializeGraph(Graph); // Инициализация графа

for i := 1 to Rows do

begin

// Добавление вершины

AddVertice(Graph, VerticeCenter);

// Цикл А2. Проход по столбцам диапазона

for j := 1 to Cols do

begin

// Получение весов в симметричных ячейках

w := StrToInt(VarToStr(Weights[i, j]));

ww := StrToInt(VarToStr(Weights[j, i]));

// Проверка на недопустимые дуги

isIncorrect := (i = j) and (w <> 0) or (w < 0) or

(w > 0) and (ww <> 0);

if isIncorrect then

raise Exception.Create

('Найдены недопустимые дуги.');

// Добавление существующей дуги

if w <> 0 then

AddArc(Graph, i, j, w);

end; // Конец A2

end; // Конец A1

finally

// Выход из Excel

MyExcel.Quit;

MyExcel := Unassigned;

CoUninitialize;

end; // Конец if

end;

end;

end.

Приложение Е

Исходный код программы (модуль GraphSearch)

unit GraphSearch;

{

Модуль с подпрограммами,

реализующими рассматриваемые

алгоритмы поиска на графах.

}

interface

uses SysUtils, Digraph, DynamicStructures;

type

// Тип матрицы весов

TWeightMatrix = array of array of Integer;

// Тип информации о поиске

TSearchInfo = record

Path: TStack;

PathString: String;

ArcsCount: Integer;

Distance: Integer;

VisitsCount: Integer;

end;

// Подпрограмма преобразования графа в матрицу весов

function ToWeightMatrix(const Graph: TGraph): TWeightMatrix;

// Подпрограмма обхода в глубину

function DFS(const Graph: TWeightMatrix; Src, Dest: Integer): TSearchInfo;

// Подпрограмма обхода в ширину

function BFS(const Graph: TWeightMatrix; Src, Dest: Integer): TSearchInfo;

// Подпрограмма поиска алгоритмом Дейкстры

function Dijkstra(const Graph: TWeightMatrix; Src, Dest:

Integer): TSearchInfo;

implementation

const

INF = 1000000000;

// Подпрограмма восстановления пути

function RestorePath(const Graph: TWeightMatrix;

const Parents: array of Integer; const Src, Dest: Integer): TSearchInfo;

const

Splitter = ', ';

// Splitter - Разделитель между вершинами в строке пути

var

v, u, w: Integer;

// v - Номер вершины-начала дуги

// u - Номер вершины-конца дуги

begin

with Result do

begin

// Инициализация начальных значений

PathString := '';

InitializeStack(Path);

ArcsCount := 0;

Distance := 0;

// Проверка существования пути или пути из 1 вершины

if (Parents[Dest - 1] <> 0) or (Src = Dest) then

begin

// Цикл А1. Проход по массиву предков

u := Dest;

while u <> Src do

begin

v := Parents[u - 1];

// Изменение информации о пути

Push(Path, u);

PathString := Splitter + IntToStr(u) + PathString;

Inc(ArcsCount);

Distance := Distance + Graph[v - 1, u - 1];

// Переход к следующей вершине

u := v;

end; // Конец А1

// Сохранение начала пути

Push(Path, Src);

PathString := IntToStr(u) + PathString;

end;

end;

end;

function DFS(const Graph: TWeightMatrix; Src, Dest: Integer): TSearchInfo;

var

v, u: Integer;

Order: Integer;

Stack: TStack;

isVisited: Array of Boolean;

Parents: Array of Integer;

// v - Номер посещаемой вершины

// u - Номер соседа вершины

// Order - Порядок графа

// Stack - Стек вершин

// isVisited - Массив флагов

// Parents - Массив предков

begin

Result.VisitsCount := 0;

// Инициализация вспомогательных данных

Order := Length(Graph);

InitializeStack(Stack);

SetLength(isVisited, Order);

SetLength(Parents, Order);

for v := 1 to Order do

begin

Parents[v - 1] := 0;

isVisited[v - 1] := False;

end;

// Цикл А1. Посещение вершин в стеке

Push(Stack, Src);

while Stack <> nil do

begin

// Получение вершины и сравнение с конечной

v := Pop(Stack);

Inc(Result.VisitsCount);

if v = Dest then

begin

DestroyList(Stack);

isVisited[v - 1] := true;

end;

// Добавление в стек соседей

if not isVisited[v - 1] then

begin

isVisited[v - 1] := true;

// Цикл А2. Добавление в стек непосещённых соседей

for u := Order downto 1 do

begin

if not isVisited[u - 1] and

(Graph[v - 1, u - 1] <> INF) then

begin

Parents[u - 1] := v; // Сохранение пути

Push(Stack, u);

end; // Конец if

end; // Конец А2

end; // Конец if

end; // Конец А1

// Восстановление пути

Result := RestorePath(Graph, Parents, Src, Dest);

end;

function BFS(const Graph: TWeightMatrix; Src, Dest: Integer): TSearchInfo;

var

v, u: Integer;

Order: Integer;

Queue: TQueue;

isVisited: Array of Boolean;

Parents: Array of Integer;

// v - Номер посещаемой вершины

// u - Номер соседа вершины

// Order - Порядок графа

// Stack - Стек вершин

// isVisited - Массив флагов

// Parents - Массив предков

begin

Result.VisitsCount := 0;

// Инициализация вспомогательных данных

Order := Length(Graph);

InitializeQueue(Queue);

SetLength(isVisited, Order);

SetLength(Parents, Order);

for v := 1 to Order do

begin

Parents[v - 1] := 0;

isVisited[v - 1] := False;

end;

// Цикл А1. Посещение вершин в очереди

Enqueue(Queue, Src);

isVisited[Src - 1] := true;

while Queue.Head <> nil do

begin

// Получение вершины и сравнение с конечной

v := Dequeue(Queue);

Inc(Result.VisitsCount);

if v = Dest then

begin

DestroyList(Queue.Head);

Order := 0; // Нарушение условия входа в А2

end;

// Цикл А2. Добавление в стек соседей вершины

for u := 1 to Order do

begin

if not isVisited[u - 1] and

(Graph[v - 1, u - 1] <> INF) then

begin

isVisited[u - 1] := true;

Parents[u - 1] := v;

Enqueue(Queue, u);

end; // Конец if

end; // Конец А2

end; // Конец А1

// Восстановление пути

Result := RestorePath(Graph, Parents, Src, Dest);

end;

function Dijkstra(const Graph: TWeightMatrix; Src, Dest: Integer): TSearchInfo;

var

v, u: Integer;

Order: Integer;

Marks: Array of Integer;

isVisited: Array of Boolean;

Parents: Array of Integer;

d: Integer;

// v - Номер посещаемой вершины

// u - Номер соседа вершины

// Order - Порядок графа

// Marks - Массив меток

// isVisited - Массив флагов

// Parents - Массив предков

// d - Метка посещаемой вершины

begin

Result.VisitsCount := 0;

// Инициализация вспомогательных данных

Order := Length(Graph);

SetLength(Marks, Order);

SetLength(isVisited, Order);

SetLength(Parents, Order);

for u := 1 to Order do

begin

Marks[u - 1] := INF;

Parents[u - 1] := 0;

isVisited[u - 1] := False;

end;

Marks[Src - 1] := 0;

// Цикл А1. Посещение вершин с минимальными метками

repeat

// Цикл А2. Поиск мин. метки у непосещённых вершин

d := INF;

for u := 1 to Order do

begin

if not isVisited[u - 1] and (Marks[u - 1] < d) then

begin

d := Marks[u - 1];

v := u;

end; // Конец if

end; // Конец А2

// Посещение найденной вершины

if d <> INF then

begin

isVisited[v - 1] := true;

Inc(Result.VisitsCount);

end

else

Order := 0; // Нарушение условия входа в А3

// Сравнение с конечной вершиной

if v = Dest then

begin

d := INF; // Выход из А1

Order := 0; // Нарушение условия входа в А3

end;

// Цикл А3. Уменьшение меток

for u := 1 to Order do

begin

if not isVisited[u - 1] and

(Marks[u - 1] > d + Graph[v - 1, u - 1]) then

begin

Parents[u - 1] := v; // Сохранение пути

Marks[u - 1] := d + Graph[v - 1, u - 1];

end; // Конец if

end; // Конец А3

until d = INF; // Конец А1

// Восстановление пути

Result := RestorePath(Graph, Parents, Src, Dest);

end;

function ToWeightMatrix(const Graph: TGraph): TWeightMatrix;

var

Vertice: TPVertice;

Neighbour: TPNeighbour;

v, u: Integer;

// Vertice - Ссылка на текущую вершину

// Neighbour - Ссылка на текущего соседа

// v - Параметр цикла по вершинам

// u - Параметр цикла по соседям

begin

SetLength(Result, Graph.Order, Graph.Order);

// Цикл А1. Проход по строкам

for v := 1 to Graph.Order do

// Цикл А2. Проход по столбцам

for u := 1 to Graph.Order do

Result[v - 1, u - 1] := INF;

// Цикл А3. Проход по вершинам

Vertice := Graph.Head;

while Vertice <> nil do

begin

v := Vertice.Number;

// Цикл A4. Проход по соседям

Neighbour := Vertice.Head;

while Neighbour <> nil do

begin

u := Neighbour.Number;

Result[v - 1, u - 1] := Neighbour.Weight;

Neighbour := Neighbour.Next;

end; // Конец A4

Vertice := Vertice.Next;

end; // Конец A3

end;

end.

Приложение Ж

Исходный код программы (модуль DynamicStructures)

unit DynamicStructures;

{

Модуль с типами и подпрограммами

для работы с динамическими структурами,

необходимыми для алгоритмов поиска.

}

interface

type

// Тип односвязного списка с натуральными числами

TPItem = ^TItem;

TItem = record

Number: Integer;

Next: TPItem;

end;

// Тип стека с натуральными числами

TStack = TPItem;

// Тип очереди с натуральными числами

TQueue = record

Head: TPItem;

Tail: TPItem;

end;

// Подпрограмма инициализации стека

procedure InitializeStack(var Stack: TStack);

// Подпрограмма инициализации очереди

procedure InitializeQueue(var Queue: TQueue);

// Подпрограмма очищения списка

procedure DestroyList(var Head: TPItem);

// Подпрограмма вставки в стек

procedure Push(var Stack: TStack; n: Integer);

// Подпрограмма добавления в очередь

procedure Enqueue(var Queue: TQueue; n: Integer);

// Подпрограмма извлечения из стека

function Pop(var Stack: TStack): Integer;

// Подпрограмма извлечения из очереди

function Dequeue(var Queue: TQueue): Integer;

implementation

procedure InitializeStack(var Stack: TStack);

begin

// Зануление указателя на голову

Stack := nil;

end;

procedure InitializeQueue(var Queue: TQueue);

begin

// Зануление указателей на голову и хвост

Queue.Head := nil;

Queue.Tail := nil;

end;

procedure DestroyList(var Head: TPItem);

var

Item: TPItem;

// Item - Ссылка на освобождаемый узел

begin

// Цикл А1. Освобождение списка

while Head <> nil do

begin

Item := Head;

Head := Head.Next;

Dispose(Item);

end; // Конец А1

end;

procedure Push(var Stack: TStack; n: Integer);

var

Item: TPItem;

// Item - Ссылка на вставляемое звено

begin

New(Item);

Item.Number := n;

Item.Next := Stack;

Stack := Item;

end;

procedure Enqueue(var Queue: TQueue; n: Integer);

var

Item: TPItem;

// Item - Ссылка на вставляемое звено

begin

// Инициализация нового элемента

New(Item);

Item.Number := n;

Item.Next := nil;

// Сохранение нового элемента

if Queue.Head <> nil then

Queue.Tail.Next := Item

else

Queue.Head := Item;

// Перемещение хвоста

Queue.Tail := Item;

end;

function Pop(var Stack: TStack): Integer;

var

Item: TPItem;

// Item - Ссылка на извлекаемое звено

begin

if Stack <> nil then

begin

// Перемещение вершины стека

Item := Stack;

Stack := Stack.Next;

// Извлечение элемента с освобождением указателя

Result := Item.Number;

Dispose(Item);

end

else

Result := 0; // Ошибка при извлечении

end;

function Dequeue(var Queue: TQueue): Integer;

var

Item: TPItem;

// Item - Ссылка на извлекаемое звено

begin

if Queue.Head <> nil then

begin

// Перемещение начала очереди

Item := Queue.Head;

Queue.Head := Queue.Head.Next;

// Извлечение элемента с очисткой указателя

Result := Item.Number;

Dispose(Item);

end

else

Result := 0; // Ошибка при извлечении

end;

end.

Приложение З

Исходный код программы (модуль GraphDrawing)

unit GraphDrawing;

{

Модуль с подпрограммами,

необходимыми для визуализации

графа.

}

interface

uses System.Types, VCL.Graphics, Math, Digraph, DynamicStructures;

// Подпрограмма изменения стиля вершин и дуг пути

procedure MakeVisited(var Graph: TGraph; var Path: TStack);

// Подпрограмма обнуления стиля вершин и дуг графа

procedure MakePassive(var Graph: TGraph);

// Подпрограмма изменения координат на вершины правильного многоугольника

procedure MakeRegPolygon(var Graph: TGraph; Width, Height: Integer);

// Подпрограмма перерисовки графа на холсте

procedure RedrawGraph(const Canvas: TCanvas; Width, Height: Integer;

const Graph: TGraph);

implementation

type

// Тип ломаной на холсте, соответствующей дуге

ArcLine = Array [1 .. 5] of TPoint;

// Подпрограмма получения ломаной для дуги

function GetArcPoints(c1, c2: TPoint; R: Integer): ArcLine;

const

ArrowAngle = pi / 4;

// ArrowAngle - Угол наклона концов стрелки

var

XStart, YStart: Integer;

XEnd, YEnd: Integer;

d: Integer;

LineAngle: Real;

Sign: Integer;

// XStart, YStart - Координаты первой вершины

// XEnd, YEnd - Координаты второй вершины

// d - Переменная для вычисления расстояния

// LineAngle - Угол наклона главной линии

// Sign - Знак разности абсцисс первой и второй вершины

begin

// Вычисление концов главной линии

d := Trunc(c1.Distance(c2));

XEnd := Round(c2.x + R \* (c1.x - c2.x) / d);

YEnd := Round(c2.y + R \* (c1.y - c2.y) / d);

XStart := Round(c1.x + R \* (c2.x - c1.x) / d);

YStart := Round(c1.y + R \* (c2.y - c1.y) / d);

// Инициализация точек главной линии

Result[1].x := XStart;

Result[1].y := YStart;

Result[2].x := XEnd;

Result[2].y := YEnd;

Result[4] := Result[2];

// Вычисление данных для концов стрелки

d := Trunc(Result[1].Distance(Result[2]));

if d <> 0 then

LineAngle := ArcSin((YStart - YEnd) / d)

else

LineAngle := 0;

Sign := 2 \* Ord(XStart >= XEnd) - 1;

d := R div 2;

// Вычисление координат стрелок

Result[3].x :=

XEnd + Sign \* Trunc(cos(ArrowAngle - Sign \* LineAngle) \* d);

Result[3].y :=

YEnd - Sign \* Trunc(sin(ArrowAngle - Sign \* LineAngle) \* d);

Result[5].x :=

XEnd + Sign \* Trunc(cos(ArrowAngle + Sign \* LineAngle) \* d);

Result[5].y :=

YEnd + Sign \* Trunc(sin(ArrowAngle + Sign \* LineAngle) \* d);

end;

// Подпрограмма рисования вершины

procedure DrawVertice(const Canvas: TCanvas; R: Integer;

const Vertice: TPVertice);

var

SNumber: String;

XMid, YMid: Integer;

// SNumber - Строка с номером вершины

// XMid, YMid - Координаты SNumber

begin

with Canvas, Vertice^, Vertice.Center do

begin

// Инициализация данных для рисования

case Style of

stPassive:

begin

Pen.Color := clBlack;

Font.Color := clBlack;

Brush.Color := clWhite;

end;

stActive:

begin

Pen.Color := clRed;

Font.Color := clBlack;

Brush.Color := clCream;

end;

stVisited:

begin

Pen.Color := clTeal;

Font.Color := clWhite;

Brush.Color := clLime;

end;

end;

// Рисование вершины с её номером

Str(Vertice.Number, SNumber);

XMid := x - TextWidth(SNumber) div 2;

YMid := y - TextHeight(SNumber) div 2;

Ellipse(x - R, y - R, x + R, y + R);

TextOut(XMid, YMid, SNumber);

end;

end;

// Подпрограмма рисования дуги

procedure DrawArc(const Canvas: TCanvas; R: Integer;

const SrcCenter, DstCenter: TPoint; const Neighbour:

TPNeighbour);

var

Points: ArcLine;

SWeight: String;

XMid, YMid: Integer;

HText, WText: Integer;

// Points - Ломаная, соотв. дуге

// Sweight - Строка с весом

// XMid, YMid - Координаты центра дуги

// HText, WText - Размеры строки с весом

begin

with Canvas do

begin

// Инициализация данных для рисования

if Neighbour.isVisited then

begin

Pen.Color := clTeal;

Font.Color := clWhite;

Brush.Color := clBlack;

end

else

begin

Pen.Color := clBlack;

Font.Color := clBlack;

Brush.Color := clWhite;

end;

Points := GetArcPoints(SrcCenter, DstCenter, R);

// Рисование дуги и веса

Polyline(Points);

if Neighbour.Weight <> 1 then

begin

Str(Neighbour.Weight, SWeight);

WText := TextWidth(SWeight);

HText := TextHeight(SWeight);

XMid := (Points[1].x + Points[2].x - WText) div 2;

YMid := (Points[1].y + Points[2].y - HText) div 2;

Rectangle(XMid, YMid,

XMid + WText + 1, YMid + HText + 1);

TextOut(XMid, YMid, SWeight);

end;

end;

end;

procedure MakeVisited(var Graph: TGraph; var Path: TStack);

var

Vertice: TPVertice;

Neighbour: TPNeighbour;

v: Integer;

// Vertice - Ссылка на текущую вершину

// Neighbour - Ссылка на текущего соседа

// v - Номер вершины, извлечённой из пути

begin

// Цикл А1. Проход по пути вершинам пути

while Path <> nil do

begin

// Изменение стиля вершины

v := Pop(Path);

Vertice := GetByNumber(Graph, v);

Vertice.Style := stVisited;

// Изменение стиля выходящей дуги

if Path <> nil then

begin

v := Pop(Path);

Neighbour := Vertice.Head;

// Цикл А2. Поиск соседа

while v <> Neighbour.Number do

Neighbour := Neighbour.Next;

Neighbour.isVisited := true;

Push(Path, v);

end; // Конец if

end; // Конец А1

// Меток о раскраске графа

Graph.isPainted := true;

end;

procedure MakePassive(var Graph: TGraph);

var

Vertice: TPVertice;

Neighbour: TPNeighbour;

// Vertice - Ссылка на текущую вершину

// Neighbour - Ссылка на текущего соседа

begin

// Цикл А1. Проход по вершинам

Vertice := Graph.Head;

while Vertice <> nil do

begin

Vertice.Style := stPassive;

// Цикл А2. Проход по соседям

Neighbour := Vertice.Head;

while Neighbour <> nil do

begin

Neighbour.isVisited := false;

Neighbour := Neighbour.Next;

end; // Конец А2

Vertice := Vertice.Next;

end;

// Изменение флага о раскраске графа

Graph.isPainted := false;

end;

procedure MakeRegPolygon(var Graph: TGraph; Width, Height: Integer);

var

Vertice: TPVertice;

ImageCenter: TPoint;

PolygonRadius: Integer;

Angle: Real;

// Vertice - Ссылка на текущую вершину

// ImageCenter - Центр изображения

// PolygonRadius - Радиус правильного многоугольника

// Angle - Текущий угол наклона радиус-вектора вершины

begin

// Инициализация угла, радиуса и центра описанной окружности

Angle := 0;

PolygonRadius := Min(Width, Height) div 2 - Graph.R;

ImageCenter.x := Width div 2;

ImageCenter.y := Height div 2;

// Цикл А1. Проход по вершинам

Vertice := Graph.Head;

while Vertice <> nil do

begin

// Присвоение координат

Vertice.Center.x := ImageCenter.x +

Trunc(PolygonRadius \* sin(Angle));

Vertice.Center.y := ImageCenter.y –

Trunc(PolygonRadius \* cos(Angle));

// Переход к следующей вершине

Vertice := Vertice.Next;

Angle := Angle + 2 \* pi / Graph.Order;

end; // Конец А1

end;

procedure RedrawGraph(const Canvas: TCanvas; Width, Height: Integer;

const Graph: TGraph);

var

Vertice, AdjVertice, Active: TPVertice;

Neighbour: TPNeighbour;

// Vertice - Текущая вершина

// AdjVertice - Смежная Vertice вершина

// Active - Вершина со стилем stActive

// Neighbour - Текущий сосед

begin

// Настройка цветов и шрифтов

with Canvas do

begin

Pen.Color := clWhite;

Rectangle(0, 0, Width, Height);

Pen.Width := 3;

Font.Size := 15;

Font.Style := [fsBold];

end;

// Цикл А1. Проход по вершинам

Vertice := Graph.Head;

while Vertice <> nil do

begin

// Цикл А2. Проход по дугам

Neighbour := Vertice.Head;

while Neighbour <> nil do

begin

AdjVertice := GetByNumber(Graph, Neighbour.Number);

// Прорисовка дуги

DrawArc(Canvas, Graph.R,

Vertice.Center, AdjVertice.Center, Neighbour);

Neighbour := Neighbour.Next;

end;

Vertice := Vertice.Next;

end;

// Цикл А3. Проход по вершинам

Vertice := Graph.Head;

while Vertice <> nil do

begin

// Прорисовка вершины

DrawVertice(Canvas, Graph.R, Vertice);

// Сохранение активной вершины

if Vertice.Style = stActive then

Active := Vertice;

Vertice := Vertice.Next;

end;

// Прорисовка активной вершины

if Active <> nil then

DrawVertice(Canvas, Graph.R, Active);

end;

end.