Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Конструирование программного обеспечения (КПО)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Игровое средство «ОсьМысление»

БГУИР КП I–40 01 01 210 ПЗ

Выполнил

студент гр.251005 Киселёв А.С.

Проверил: Фадеева Е.П.

Минск 2023

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

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

(подпись)

Лапицкая Н.В. 2023г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Киселёву Артёму Сергеевичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы Игровое средство «ОсьМысление» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

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

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

2. Срок сдачи законченной работы *03.06.2023г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Среда программирования Delphi. Возможность создания графа с последующим сохранением в типизированный файл. Возможность импорта графа из файла Excel. Возможность сохранения графа в формате .bmp. Создание графа из типизированного файла. Экспорт графа в файл Excel. Нахождение всех путей между двумя вершинами графа и их графическое отображение. Нахождение всех кратчайших путей между двумя вершинами графа. Для нахождения путей в графе алгоритмом поиска в глубину использовать структуру данных стек на основе линейного однонаправленного списка.* *Для нахождения путей в графе алгоритмом поиска в ширину использовать структуру данных очередь на основе линейного однонаправленного списка. Возможность сортировки найденных путей по длине. Возможность редактирования графа (изменения положения вершин/рёбер, их цвета, размера, веса рёбер). Возможность создания ориентированного и неориентированного графа.* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ литературных источников и формирование функциональных требований к разрабатываемому программному средству\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Проектирование и разработка программного средства* \_\_\_\_\_\_*\_\_\_\_\_\_\_\_\_\_\_*

*3 Тестирование и проверка работоспособности программного средства\_\_\_*

*4 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 *16.02.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

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

*Раздел 1, Введение к 28.02.2023г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 15.03.2023г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 15.04.2023г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4, Заключение,Приложения к 20.05.2023г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 31.05.2023г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 01.06.2023г. по 03.06.2023г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

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

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_Киселёв А.С. 16.02.2023г.*

*(дата и подпись студента)*

Содержание

[Введение 8](#_Toc135862673)

[1 Анализ литературных источников 9](#_Toc135862674)

[1.1 Анализ существующих аналогов 9](#_Toc135862675)

[1.1.1 Программное средство для нахождения кратчайшего пути в графе методом Дейкстры semestr.online 9](#_Toc135862676)

[1.1.2 Программное средство для нахождения кратчайшего пути в графе методом Дейкстры programforyou.ru 11](#_Toc135862677)

[1.1.3 Программное средство для нахождения кратчайшего пути в графе методом Дейкстры graphonline.ru 12](#_Toc135862678)

[1.2 Описание средств разработки 13](#_Toc135862679)

[1.2.1 Использование динамических структур 13](#_Toc135862680)

[1.2.2 Использование файлов 14](#_Toc135862681)

[1.2.3 Работа с Excel 16](#_Toc135862682)

[1.2.4 Алгоритм сортировки 17](#_Toc135862683)

[1.3 Описание предметной области 18](#_Toc135862684)

[1.3.1 История задачи поиска путей в графе 18](#_Toc135862685)

[1.3.2 Основные понятия теории графов 19](#_Toc135862686)

[1.3.3 Алгоритм поиска в глубину 19](#_Toc135862687)

[1.3.4 Алгоритм поиска в ширину 20](#_Toc135862688)

[1.3.5 Алгоритм Дейкстры 21](#_Toc135862689)

[1.4 Спецификация функциональных требований 22](#_Toc135862690)

[2 Проектирование и разработка программного средства 26](#_Toc135862691)

[2.1 Описание алгоритмов решения задачи 26](#_Toc135862692)

[2.2 Структура данных 34](#_Toc135862693)

[2.2.1 Структура типов программы 34](#_Toc135862694)

[2.2.2 Структура данных алгоритма ConnectedVertex 39](#_Toc135862695)

[2.2.3 Структура данных алгоритма AddVertex 40](#_Toc135862696)

[2.2.4 Структура данных алгоритма addsemivertex 40](#_Toc135862697)

[2.2.5 Структура данных алгоритма AddEdge 41](#_Toc135862698)

[2.2.6 Структура данных алгоритма deleteSemiVertex 41](#_Toc135862699)

[2.2.7 Структура данных алгоритма DeleteEdge 42](#_Toc135862700)

[2.2.8 Структура данных алгоритма DeleteVertex 42](#_Toc135862701)

[2.2.9 Структура данных алгоритма HeadCreate 43](#_Toc135862702)

[2.2.10 Структура данных алгоритма FindVertexInList 43](#_Toc135862703)

[2.2.11 Структура данных алгоритма WayFromStack 43](#_Toc135862704)

[2.2.12 Структура данных алгоритма SetVertProperties 43](#_Toc135862705)

[2.2.13 Структура данных алгоритма SetEdgeProperties 44](#_Toc135862706)

[2.2.14 Структура данных алгоритма SaveListToFile 44](#_Toc135862707)

[2.2.15 Структура данных алгоритма CreateFromFile 45](#_Toc135862708)

[2.2.16 Структура данных алгоритма StackPush 45](#_Toc135862709)

[2.2.17 Структура данных алгоритма StackPop 46](#_Toc135862710)

[2.2.18 Структура данных алгоритма DFS 46](#_Toc135862711)

[2.2.19 Структура данных алгоритма NodeInStack 47](#_Toc135862712)

[2.2.20 Структура данных алгоритма QueueInit 47](#_Toc135862713)

[2.2.21 Структура данных алгоритма QueuePush 48](#_Toc135862714)

[2.2.22 Структура данных алгоритма QueuePop 48](#_Toc135862715)

[2.2.23 Структура данных алгоритма BFS 48](#_Toc135862716)

[2.2.24 Структура данных алгоритма PrQueueInit 49](#_Toc135862717)

[2.2.25 Структура данных алгоритма QueueVisited 50](#_Toc135862718)

[2.2.26 Структура данных алгоритма UpdateDistance 50](#_Toc135862719)

[2.2.27 Структура данных алгоритма Dijcstra 51](#_Toc135862720)

[2.2.28 Структура данных алгоритма QuickSortWaysByLength 52](#_Toc135862721)

[2.2.29 Структура данных алгоритма SaveGraphToExcel 52](#_Toc135862722)

[2.3 Схемы алгоритмов решения задачи по ГОСТ 19.701-90 54](#_Toc135862723)

[2.3.1 Схема алгоритма Dijcstra 54](#_Toc135862724)

[2.3.2 Схема алгоритма PrQueueInit 58](#_Toc135862725)

[2.3.3 Схема алгоритма SaveListToFile 60](#_Toc135862726)

[2.3.4 Схема алгоритма CreateFromFile 62](#_Toc135862727)

[2.3.5 Схема алгоритма QueueVisited 64](#_Toc135862728)

[2.3.6 Схема алгоритма WayFromStack 66](#_Toc135862729)

[2.3.7 Схема алгоритма QueuePop 67](#_Toc135862730)

[2.3.8 Схема алгоритма AddVertex 68](#_Toc135862731)

[2.3.9 Схема алгоритма AddEdge 70](#_Toc135862732)

[2.3.10 Схема алгоритма DeleteEdge 71](#_Toc135862733)

[2.4 Графический интерфейс 72](#_Toc135862734)

[2.4.1 Описание графических компонентов формы FMenu 72](#_Toc135862735)

[2.4.2 Описание графических компонентов формы MainForm 73](#_Toc135862736)

[2.4.3 Описание графических компонентов формы FormWays 78](#_Toc135862737)

[3 Тестирование программного средства 80](#_Toc135862738)

[3.1 Навигация в программном средстве 80](#_Toc135862739)

[3.1.1 Тест 1 80](#_Toc135862740)

[3.1.2 Тест 2 80](#_Toc135862741)

[3.1.3 Тест 3 81](#_Toc135862742)

[3.1.4 Тест 4 82](#_Toc135862743)

[3.2 Работа формы MainForm 82](#_Toc135862744)

[3.2.1 Тест 1 82](#_Toc135862745)

[3.2.2 Тест 2 83](#_Toc135862746)

[3.2.3 Тест 3 84](#_Toc135862747)

[3.2.4 Тест 4 85](#_Toc135862748)

[3.2.5 Тест 5 86](#_Toc135862749)

[3.2.6 Тест 6 87](#_Toc135862750)

[3.2.7 Тест 7 88](#_Toc135862751)

[3.2.8 Тест 8 89](#_Toc135862752)

[3.2.9 Тест 9 90](#_Toc135862753)

[3.2.10 Тест 10 91](#_Toc135862754)

[3.2.11 Тест 11 92](#_Toc135862755)

[3.2.12 Тест 12 93](#_Toc135862756)

[3.2.13 Тест 13 94](#_Toc135862757)

[3.2.14 Тест 14 95](#_Toc135862758)

[3.2.15 Тест 15 96](#_Toc135862759)

[3.2.16 Тест 16 96](#_Toc135862760)

[3.2.17 Тест 17 97](#_Toc135862761)

[3.2.18 Тест 18 98](#_Toc135862762)

[3.3 Работа формы FormWays 99](#_Toc135862763)

[3.3.1 Тест 1 99](#_Toc135862764)

[3.3.2 Тест 2 100](#_Toc135862765)

[3.3.3 Тест 3 101](#_Toc135862766)

[3.4 Работа алгоритма Дейкстры 102](#_Toc135862767)

[3.4.1 Тест 1 102](#_Toc135862768)

[3.4.2 Тест 2 103](#_Toc135862769)

[3.5 Работа алгоритма поиска в ширину 105](#_Toc135862770)

[3.5.1 Тест 1 105](#_Toc135862771)

[3.5.2 Тест 2 106](#_Toc135862772)

[3.6 Работа алгоритма поиска в глубину 107](#_Toc135862773)

[4 Руководство по установке и использованию программного средства 110](#_Toc135862774)

[4.1 Минимальные системные требования 110](#_Toc135862775)

[4.2 Установка 110](#_Toc135862776)

[4.3 Руководство по работе с программным средством 113](#_Toc135862777)

[Заключение 118](#_Toc135862778)

[Список использованной литературы 119](#_Toc135862779)

[Приложение A 120](#_Toc135862780)

[Приложение Б 124](#_Toc135862781)

[Приложение В 130](#_Toc135862782)

[Приложение Г 132](#_Toc135862783)

[Приложение Д 138](#_Toc135862784)

[Приложение Е 141](#_Toc135862785)

[Приложение Ж 145](#_Toc135862786)

[Приложение З 149](#_Toc135862787)

[Приложение И 177](#_Toc135862788)

[Приложение К 184](#_Toc135862789)

Введение

В эпоху современных компьютерных технологий, операционные системы стали незаменимой частью нашей цифровой жизни. Они управляют ресурсами компьютера, обеспечивают взаимодействие между пользователем и железом, а также обеспечивают выполнение разнообразных программ и процессов. Операционные системы играют роль невидимых рук, которые координируют все аспекты работы компьютера.

В рамках данной курсовой работы представляется возможность погрузиться в мир операционных систем и создать собственное приложение, воплощая в жизнь забавную и увлекательную идею, предложенную Pier-Luc Brault. Эта идея предполагает не просто создание операционной системы, но и превращение вас в эту систему. Вашей задачей будет управление ядрами ЦП, процессами, управление памятью, и в общем, забота о том, чтобы ваш "пользователь" не попал в белое каление из-за того, что его компьютер тормозит.

Для реализации данной забавной игры, будет использоваться язык ассемблера FASM (Flat Assembler). Ассемблер, как наиболее близкий к аппаратному уровню язык программирования, предоставит уникальную возможность создать оптимизированное и быстрое приложение, которое будет эффективно управлять ресурсами и процессами, воплощая в жизнь забавную концепцию "операционной системы как игры".

В данной работе будут разобраны основные механизмы и принципы операционных систем, рассмотрены процессы, управление ЦП и памятью, а также интересные аспекты взаимодействия с "пользователем" в контексте этой увлекательной игры.

Таким образом, в ходе данной курсовой работы будет представлено увлекательное и образовательное путешествие в мир операционных систем и ассемблера, давая возможность погрузиться в роль самой операционной системы, чтобы избежать "игрового конца" и обеспечить надежную работу виртуального компьютера пользователя.

# Анализ литературных источников

## Анализ существующих аналогов

Идея создания игрового средства «ОсьМысление» была взята у автора Pier-Luc Brault, который придумал и разработал программное средство «you-the-os» на языке программирования Python с использованием библиотеки PyGame.

Приложение, разработанное Pier-Luc Brault, представляет собой увлекательную игру, где пользователь воплощается в роль операционной системы. Основная задача в игре - управлять ядрами центрального процессора, управлять процессами и учесть их потребности в ресурсах. Пользователь должен избегать перезагрузки "системы", что является "игровым концом". Процессы проходят через несколько уровней "голода", и если они остаются слишком долго без ресурсов, пользователь вынужден завершить их. Игра также включает в себя аспекты управления памятью, включая работу с страницами памяти и дисковым пространством. Игра предлагает разные уровни сложности и позволяет настраивать параметры для более глубокого погружения в мир операционных систем.

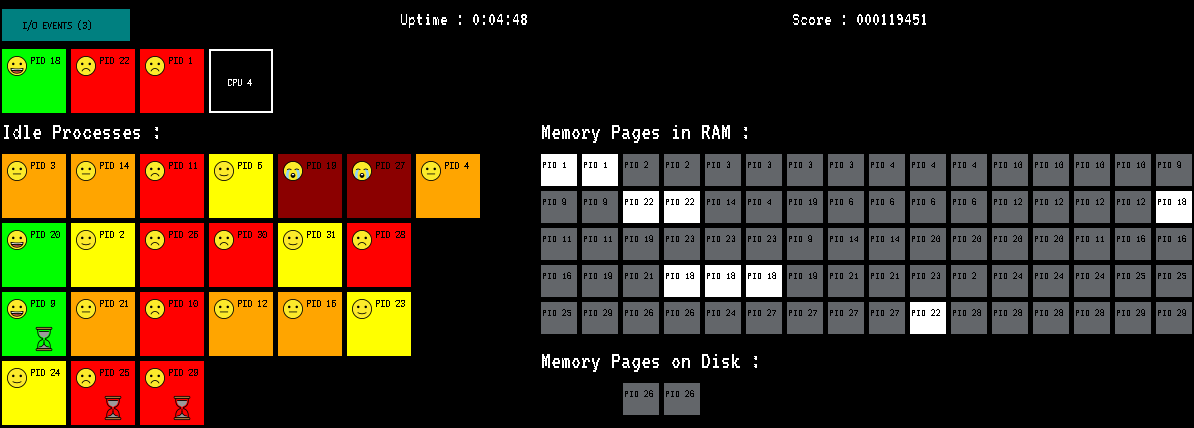


Рисунок 1.1 – Игровое средство «you-the-os»

К преимуществам «you-the-os» можно отнести:

* оригинальную идею (игровое средство предлагает уникальный и креативный подход, превращая пользователя в операционную систему, что делает его интересным и запоминающимся);
* образовательный аспект (игровое средство может служить образовательным инструментом для понимания принципов работы операционных систем, управления ресурсами и процессами).

Недостатками «you-the-os» являются:

* сложность (тема операционных систем может оказаться слишком сложной для неопытных пользователей;
* ограниченная аудитория (игровое средство скорее всего заинтересует тех, кто имеет технический бэкграунд, ограничивая свою популярность среди широкой аудитории;
* использование медленного языка программирования Python, что снижает скорость работы игрового средства.

## Описание средств разработки

### Использование динамических структур

Для реализации программного средства нахождения кратчайшего пути в графе будут использоваться динамические структуры:

* однонаправленный список;
* очередь;
* стек.

#### Использование динамической структуры данных линейный однонаправленный список

Однонаправленный список представляет собой динамическую структуру данных, число элементов которой может изменяться по мере того, как данные помещаются в список или удаляются из него. Такая реализация не требует непрерывной области памяти и позволяет не перемещать элементы при добавлении или удалении данных из списка. Т.к. пользователь может изменять структуру графа, то хранить данные о его вершинах и ребрах в массиве нецелесообразно из-за постоянного переноса данных в памяти. Однонаправленный список позволит эффективно добавлять и удалять информацию о структуре графа.

Однонаправленный список состоит из ячеек, каждая из которых содержит его элемент и указатель на следующую ячейку списка. Указатель последней ячейки списка имеет значение nil. Также в списке присутствует ячейка header, которая указывает на ячейку, содержащую первый элемент.

Основные операции над элементами списка:

* просмотр;
* вставка;
* удаление.



Рисунок 1.6 – Однонаправленный список

#### Использование динамической структуры данных очередь

Очередь – структура, из которой элементы удаляются с одного её конца (начала), а вставляются на противоположном конце (хвосте). Очереди работают по принципу FIFO (first in – first out). Тип данных очередь эффективно использовать в алгоритме поиска в ширину, так как это гарантирует, что вершины графа, которые находятся ближе к начальной вершине будут рассмотрены алгоритмом раньше, чем более удалённые от начала вершины, что и лежит в основе алгоритма поиска в ширину.

Над типом данных очередь определены две основные операции:

* вставка элементов;
* извлечение элементов.

#### Использование динамической структуры данных стек

Стек – специальный тип списка, в котором все вставки и удаления выполняются только на одном конце (вершине). Стек работает по принципу LIFO (последний вошёл – первый вышел). Для удобного представления стека можно использовать однонаправленный список, в котором элементы добавляются и извлекаются с одного конца списка.

Пример работы стека представлен на Рис. 1.7



Рисунок 1.7 – Стек

### Использование файлов

Физический файл (набор данных) — это именованная область памяти на внешнем носителе, в которой хранится некоторая информация.

Файловый тип – это 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.

Существуют следующие виды переменных файлового типа:

* типизированные файлы;
* текстовые файлы;
* не типизированные файлы.

Над переменными файлового типа не определены никакие операции, работа осуществляется с помощью процедур и функций ввода-вывода. Основные из них приведены в таблице 1.1.

Таблица 1.1 – Основные процедуры и функции для работы с файлами

| Имя | Вид | Описание | Тип результата |
| --- | --- | --- | --- |
| AssignFile  (  F,  Path  ) | Процедура | Связывает файловую переменную F с файлом по пути Path | - |
| Append  (  F  ) | Процедура | Открывает существующий файл F и ставит указатель в конец файла | - |
| CloseFile  (  F  ) | Процедура | Закрывает файл F и сохраняет изменения | - |
| Eof  (  F  ) | Функция | Проверяет, стоит ли указатель в конце файла F | Boolean |
| Eoln  (  F  ) | Функция | Проверяет, стоит ли указатель в конце строки файла F | Boolean |
| FilePos(  F  ) | Функция | Возвращает текущую позицию указателя в файле F | LongInt |
| Read  (  F,  V1,  …  Vn  ) | Процедура | Считывает из файла F значения и записывает их в переменные V1, …, Vn | - |

Продолжение Таблицы 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| Rename  (  F,  Name  ) | Процедура | Переименовывает файл F в Name | - |
| Seek  (  F,  N  ) | Процедура | Устанавливает указатель на позицию N в файле F | - |
| Truncate  (  F  ) | Процедура | Удаляет все записи в файле F после указателя | - |
| Write(  F,  V1,  …,  Vn  ) | Процедура | Записывает в файл значения V1, …, Vn | - |
| Reset  (  F  ) | Процедура | Открывает файл для чтения | - |
| Rewrite  (  F  ) | Процедура | Открывает файл для записи | - |

Для сохранения созданного графа и построения графа из файла будет использоваться типизированный файл, который будет хранить информацию, полученную из полей записи[[1]](#footnote-1)

### Работа с Excel

Граф удобно представлять в виде матрицы смежности или инцидентности, то есть, в виде таблицы. Одним из самых популярных программных средств для работы с таблицами является Excel.

В Delphi можно работать с файлами типа Excel, используя объекты из библиотеки OLE Automation. Для этого необходимо подключить библиотеку Excel, создать объекты Application и Workbook, открыть нужный файл, выбрать нужный лист и выполнять различные операции.

Пример кода для создания объектов Application и Workbook и открытия файла Excel:

var

ExcelApp, ExcelWorkbook: OleVariant;

begin

ExcelApp := CreateOleObject('Excel.Application');

ExcelWorkbook :=

ExcelApp.Workbooks.Open('C:\example.xlsx');

ExcelApp.Visible := True;

ExcelWorkbook.Worksheets[1].Activate;

// Дальнейшие операции с файлом Excel...

end;

Для работы с ячейками можно использовать свойство Range, которое позволяет обратиться к определенной ячейке или диапазону ячеек.

Например, чтобы изменить значение ячейки A1, можно использовать следующий код:

ExcelWorkbook.Worksheets[1].Range('A1').Value :=

'Новое значение';

Для добавления новых строк и столбцов можно использовать методы Insert и Delete, а для форматирования ячеек - свойство Style.

Например, чтобы задать жирный шрифт для ячейки A1, можно использовать следующий код:

ExcelWorkbook.Worksheets[1].Range('A1').Style.Font.Bold

:= True;

После завершения работы с файлом необходимо закрыть его и освободить занимаемые объектами ресурсы:

ExcelWorkbook.Save;

ExcelWorkbook.Close;

ExcelApp.Quit;

ExcelWorkbook := Unassigned;

ExcelApp := Unassigned;

### Алгоритм сортировки

При нахождении кратчайших путей в графе их количество может оказаться большим 1 (путей может быть несколько). Для того, чтобы пользователь мог увидеть пути в порядке возрастания/убывания длины необходимо использовать сортировку. В данной работе будет использоваться быстрая сортировка Хоара.

Быстрая сортировка (англ. QuickSort) — это эффективный алгоритм сортировки, который работает по принципу «разделяй и властвуй». Он основан на стратегии разделения массива на две подгруппы, затем сортирует каждую из подгрупп рекурсивно и объединяет результаты для получения окончательно отсортированного массива.

Шаги алгоритма быстрой сортировки:

1. Выбрать опорный элемент из массива. Опорный элемент может быть любым элементом массива. Обычно выбирают первый, последний или средний элемент массива.
2. Разделить массив на две подгруппы: элементы, меньшие опорного, и элементы, большие опорного. Это называется разделением. Можно использовать два индекса (назовем их low и high), которые будут двигаться друг к другу и менять местами элементы, если они находятся в неправильном порядке относительно опорного элемента.
3. Рекурсивно применить шаги 1 и 2 к обеим подгруппам, пока в подгруппе остается больше одного элемента. То есть, продолжать разделять и сортировать подгруппы до тех пор, пока в них не останется один элемент или пустой массив.
4. Когда размер подгруппы становится равным 1 или 0, завершить рекурсию.
5. Объединить отсортированные подгруппы вместе, чтобы получить окончательно отсортированный массив. Для этого можно объединить элементы меньшие опорного, сам опорный элемент и элементы большие опорного.

Алгоритм быстрой сортировки обладает хорошей производительностью в среднем случае и на практике часто является одним из самых быстрых алгоритмов сортировки.

## Описание предметной области

### История развития операционных систем (ОС)

История развития операционных систем включает в себя множество значительных изменений и технологических переходов. Вот краткий обзор ключевых этапов развития ОС:

1. Реальный режим (Real Mode): В начале истории операционных систем, особенно в DOS-эре, процессоры работали в реальном режиме. В этом режиме операционная система и приложения имели прямой доступ к физической памяти. Однако это также означало, что программы могли легко повредить друг друга и даже саму операционную систему.
2. Защищенный режим (Protected Mode): Защищенный режим, введенный в Intel 80286, предоставил механизмы защиты памяти и разделения адресного пространства между задачами (процессами). В этом режиме операционная система может настроить различные уровни защиты для сегментов памяти, предотвращая несанкционированный доступ или модификацию данных. Каждый сегмент памяти имеет свои права доступа (чтение, запись, выполнение), и нарушения прав доступа приводят к генерации исключений.
3. Страничная адресация (Paging): С развитием x86-архитектуры, в частности с введением процессора Intel 80386, стала активно использоваться страничная адресация. Эта технология позволяет разбивать физическую память и виртуальную память на страницы фиксированного размера и управлять этими страницами. Страничная адресация обеспечивает эффективное управление памятью, включая возможность использования файлов подкачки (paging file) для обмена данными между физической памятью и диском.
4. Файл подкачки (Paging File): Файл подкачки - это механизм, при котором операционная система использует часть жесткого диска в качестве виртуальной памяти. Если физическая память исчерпывается, данные могут быть выгружены на диск, и, наоборот, загружены обратно в физическую память при необходимости. Это позволяет операционной системе эффективно управлять памятью и поддерживать множество одновременно работающих процессов.
5. Виртуальная память (Virtual Memory): Виртуальная память представляет собой совокупность памяти, доступной для приложений, которая может быть больше, чем физическая память. Она достигается через страничную адресацию и механизмы файлов подкачки. Виртуальная память обеспечивает изоляцию приложений и повышает стабильность системы.
6. Многозадачность и многопоточность: Многозадачность и многопоточность достигаются через технические механизмы внутри операционных систем, включая планировщики задач и поддержку множества потоков выполнения.

### Основные понятия теории графов

В данной работе будут использоваться термины:

* граф – пара множеств (V, E), где V – множество вершин, а E – множество ребер. Ребро – это связь между двумя вершинами. Если (u, v) является ребром графа, то u и v называются смежными вершинами, а ребро (u, v) инцидентно с вершинами u и v. Графы бывают ориентированными и неориентированными. В ориентированном графе рёбра имеют направление, в неориентированном – нет. Кроме того, графы могут быть взвешенными и невзвешенными, где взвешенный граф – это граф, в котором каждое ребро имеет числовой вес или стоимость;
* дерево – частный случай графа, представляющий собой связный ациклический граф. Корень дерева является вершиной, из которой не выходит ни одного ребра, и которая служит начальной точкой для поиска и обхода дерева. Каждый узел дерева имеет ровно одного родителя, за исключением корня, и может иметь произвольное количество дочерних узлов.

### Алгоритм поиска в глубину

Алгоритм поиска в глубину используется для обхода графа или дерева. Он начинает обход графа с заданной стартовой вершины. На каждом шаге выбирается одна из необработанных смежных вершин текущей вершины и рекурсивно вызывается алгоритм поиска в глубину от этой вершины. Этот процесс повторяется до тех пор, пока не будут посещены все вершины, достижимые из начальной вершины.

Процесс обхода графа можно представить в виде дерева, в котором стартовая вершина является корнем, а все достижимые из нее вершины расположены на уровнях этого дерева. Если в процессе обхода алгоритм встречает вершину, которая уже была посещена, он пропускает ее и продолжает обход.

Когда алгоритм поиска в глубину обходит вершину, он помечает ее как посещенную и добавляет ее в стек. После того, как он обработал все смежные вершины текущей вершины, он извлекает вершину из стека и продолжает обход графа от следующей вершины в стеке. Если стек пуст, алгоритм завершается.

Пример обхода графа при помощи алгоритма поиска в глубину представлен на Рис. 1.8.



Рисунок 1.8 – Алгоритм поиска в глубину

Порядок обхода вершин: a-b-d-g-e-c-f-h-i.

### Алгоритм поиска в ширину

Алгоритм поиска в ширину – это алгоритм обхода графа, который позволяет найти все вершины, доступные из заданной начальной вершины, и определить кратчайший путь от начальной вершины до каждой достижимой вершины.

Алгоритм работает по принципу «пошагового» обхода графа, начиная с заданной начальной вершины, и постепенного расширения области поиска на все более дальние вершины от начальной. Для этого алгоритм использует очередь, в которую помещаются все вершины, достижимые из начальной вершины.

Алгоритм поиска в ширину состоит из следующих шагов:

1. Создать очередь и добавить в нее начальную вершину.
2. Отметить начальную вершину как посещенную.
3. Извлечь из очереди вершину, которая была добавлена первой.
4. Для каждой смежной с извлеченной вершиной вершины, которая еще не была посещена, отметить вершину как посещённую и добавить её в очередь.
5. Повторять шаги 3-4, пока очередь не станет пустой.

Пример обхода графа при помощи алгоритма поиска в ширину представлен на Рис. 1.9.



Рисунок 1.9 – Алгоритма поиска в ширину

Порядок обхода вершин: a-b-c-d-e-f-g-h-i.

### Алгоритм Дейкстры

Алгоритм Дейкстры был разработан нидерландским ученым Эдсгером Дейкстрой в 1956 году. Он представил свой алгоритм в статье "A Note on Two Problems in Connexion with Graphs", опубликованной в журнале Numerische Mathematik.

Алгоритм Дейкстры был разработан для нахождения кратчайшего пути между двумя вершинами во взвешенном графе с неотрицательными весами ребер.

Алгоритм Дейкстры использует жадный подход к решению проблемы поиска кратчайшего пути, то есть на каждом шаге выбирается наименьшее расстояние от начальной вершины до некоторой другой вершины, и затем этот путь добавляется в список кратчайших путей.

Алгоритм Дейкстры состоит из следующих шагов:

1. Инициализировать расстояние от начальной вершины до всех остальных вершин как бесконечность, кроме начальной вершины, у которой расстояние равно 0.
2. Добавить начальную вершину в очередь с приоритетами.
3. Пока очередь не пуста извлечь из неё вершину u с наименьшим расстоянием. Для каждой смежной вершины v с весом ребра w(u,v), если расстояние от начальной вершины до вершины v больше, чем расстояние от начальной вершины до вершины u плюс вес ребра (u,v), то обновить расстояние до вершины v. Добавить вершину v в очередь с приоритетами с новым расстоянием.
4. Вернуть найденные расстояния до всех вершин.

## Спецификация функциональных требований

Требование – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

На основании произведённого обзора существующих аналогов, выявленных преимуществ и недостатков данных приложений, были получены следующие функциональные требования, представленные в Таблице 1.2.

Таблица 1.2 – Функциональные требования

|  |  |  |
| --- | --- | --- |
| Идентификатор | Требование | Описание требования |
| ФТ-1 | Выбор уровня сложности | Предоставление пользователю выбора одного из уровней сложности (лёгкий, средний, тяжёлый) |
| ФТ-1.1 | Выбор количества ядер процессора | Предоставление пользователю выбора количества ядер процессора |
| ФТ-1.2 | Выбор количества процессов | Предоставление пользователю выбора начального количества запущенных процессов |
| ФТ-1.3 | Выбор размера оперативной памяти | Предоставление пользователю выбора размера оперативной памяти (количество страниц в памяти) |
| ФТ-2 | Перемещение процесса на ядро процессора | Предоставление пользователю возможности нажатием левой клавишей мыши переместить запущенный процесс на ядро процессора |
| ФТ-3 | Перемещение процесса с ядра процессора | Предоставление пользователю возможности нажатием левой клавишей мыши переместить запущенный процесс с ядра процессора в режим «голодания» |
| ФТ-4 | Перемещение страницы из оперативной памяти на диск | Предоставление пользователю возможности нажатием левой клавишей мыши переместить страницу процесса из оперативной памяти на диск (файл подкачки) |

Продолжение Таблицы 1.2

|  |  |  |
| --- | --- | --- |
| ФТ-5 | Перемещение страницы с диска в оперативную память | Предоставление пользователю возможности нажатием левой клавишей мыши переместить страницу процесса с диска (файла подкачки) в оперативную память |
| ФТ-6 | Выполнение операции ввода-вывода | Предоставление пользователю возможности нажатием левой клавишей мыши выполнить операцию ввода вывода для запущенных процессов |

# Проектирование и разработка программного средства

## Описание алгоритмов решения задачи

Таблица 2.1 – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование алгоритма | Назначение алгоритма | Формальные параметры | Предпола-гаемый тип реализации |
| 1 | ConnectedVertex  (  vert1,  vert2,  mainlist1,  res  ) | Проверять, есть ли ребро между вершинами vert1 и vert2 | vert1 – получает от фактического параметра адрес с защитой;  vert2 – получает от фактического параметра адрес с защитой;  mainlist1 – получает от фактического параметра адрес;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |
| 2 | AddVertex  (  pnewvertex,  vertex  ) | Добавлять вершину Vertex в список смежности PNewVertex | pnewvertex – получает от фактического параметра адрес;  vertex – получает от фактического параметра адрес с защитой | Процедура |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | AddSemivertex (  source,  destination,  semiedge,  newedge,  directed  ) | Добавлять в список смежных вершин source вершину destination с ребром semiedge | source – получает от фактического параметра адрес;  destination – получает от фактического параметра адрес с защитой;  semiedge – получает от фактического параметра адрес с защитой;  newedge – получает от фактического параметра адрес;  directed – получает от фактического параметра адрес с защитой | Процедура |
| 4 | AddEdge  (  fvertex,  svertex,  edge,  mainlist,  oriented  ) | Добавлять ребро между вершинами fvertex и svertex в список смежности mainlist  Вызывать подпрограммы AddSemivertex, FindVertexInList | fvertex – получает от фактического параметра адрес с защитой;  svertex – получает от фактического параметра адрес с защитой;  edge – получает от фактического параметра адрес с защитой;  mainlist – получает от фактического параметра адрес;  oriented – получает от фактического параметра адрес с защитой | Процедура |
| 5 | DeleteSemiVertex  (  source,  destination  ) | Удалять смежную с вершиной Source вершину Destination из списка смежных вершин | source – получает от фактического параметра адрес;  destination – получает от фактического параметра адрес с защитой | Процедура |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | DeleteEdge  (  fvertex,  svertex,  mainlist  ) | Удалять смежную с вершиной fvertex вершину svertex из списка смежности  Вызывать подпрограммы DeleteSemivertex, FindVertexInList,ConnectedVertex | fvertex – получает от фактического параметра адрес с защитой;  svertex – получает от фактического параметра адрес с защитой;  mainlist – получает от фактического параметра адрес | Процедура |
| 7 | DeleteVertex  (  vertexname,  mainlist  ) | Удалять вершину vertexname из списка смежности mainlist  Вызывать подпрограммы DeleteSemivertex, FindVertexInList | vertexname – получает от фактического параметра адрес с защитой;  mainlist – получает от фактического параметра копию значения | Процедура |
| 8 | HeadCreate  (  newvert  ) | Создавать голову списка смежности | newvert – получает от фактического параметра адрес | Процедура |
| 9 | FindVertexInList  (  SearchingName,  ListPointer  ) | Находить вершину SearchingName в списке смежности ListPointer | SearchingName – получает от фактического параметра адрес с защитой;  ListPointer – получает от фактического параметра адрес | Процедура |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | WayFromStack  (  stack,  res  ) | Извлекать из стека найденный путь между двумя вершинами | Stack– получает от фактического параметра копию значения;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |
| 11 | SetDefault  (  VertToDefualt  ) | Возвращать указатель с текущей смежной вершины на первую смежную вершину в списке | VertToDefualt  – получает от фактического параметра адрес | Процедура |
| 12 | SetVertProperties  (  name,  x,  y,  res  ) | Задавать начальные характеристики создаваемой вершине | name – получает от фактического параметра адрес с защитой;  x – получает от фактического параметра адрес с защитой;  y – получает от фактического параметра адрес с защитой;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |
| 13 | SetEdgeProperties  (  res  ) | Задавать начальные характеристики создаваемому ребру | res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | SaveListToFile  (  Graph,  EfileName,  VFileName  ) | Создавать типизированный файл для вершин и для рёбер | Graph – получает от фактического параметра адрес;  EfileName – получает от фактического параметра адрес с защитой;  VFileName – получает от фактического параметра адрес с защитой | Процедура |
| 15 | CreateFromFile  (  Graph,  EfileName,  VFileName  ) | Создать граф на основе типизированного файла для вершин и для рёбер  Вызывать подпрограммы  AddVertex, AddEdge | Graph – получает от фактического параметра адрес;  EfileName – получает от фактического параметра адрес с защитой;  VFileName – получает от фактического параметра адрес с защитой | Процедура |
| 16 | NodeInStack  (  stack,  node  res  ) | Проверять находится ли вершина node в стеке stack | Stack – получает от фактического параметра копию значения;  Node – получает от фактического параметра адрес с защитой;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |
| 17 | SaveGraphToExcel  (  Graph,  Way  ) | Создать файл Excel для вершин и рёбер | Graph – получает от фактического параметра копию значения;  Way – получает от фактического параметра адрес; | Процедура |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 18 | LoadGraphFromExcel  (  Filename,  Graph  ) | Создать граф на основе файла Excel для вершин и для рёбер  Вызывать подпрограммы  AddVertex, AddEdge | Filename – получает от фактического параметра адрес;  Graph – получает от фактического параметра адрес | Процедура |
| 19 | QueueInit  (  PQu  ) | Создавать голову очереди для алгоритма BFS | PQu – получает от фактического параметра адрес | Процедура |
| 20 | QueuePush  (  newvertex,  PQu  ) | Добавлять вершину newvertex в очередь PQu | PQu – получает от фактического параметра адрес;  Newvertex – получает от фактического параметра адрес | Процедура |
| 21 | QueuePop  (  PQu,  res  ) | Извлекать из очереди PQu для алгоритма BFS вершину | PQu – получает от фактического параметра адрес;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. Res – возвраща-емый функцией параметр |
| 22 | BFS  (  MainList,  Start,  Finish,  res  ) | Находить все пути из вершины Start в вершину Finish в списке смежности MainList  Вызывать подпрограммы QueueInit, QueuePush, QueuePop, NodeInStack,  WayFromStack | MainList – получает от фактического параметра адрес;  Start – получает от фактического параметра адрес с защитой;  Finish – получает от фактического параметра адрес с защитой;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23 | StackPush  (  NodeList,  NewElP,  res  ) | Добавлять вершину NodeList в стек NewElP | NodeList – получает от фактического параметра адрес;  NewElP – получает от фактического параметра адрес;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |
| 24 | StackPop  (  OldElP,  ChgIncVert,  res  ) | Извлекать из стека OldElP вершину  Вызывать подпрограмму FindVertexInList | OldElP – получает от фактического параметра адрес;  ChgIncVert – получает от фактического параметра адрес;  res– получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |
| 25 | DFS  (  MainList,  Start,  Finish,  res  ) | Находить все пути из вершины Start в вершину Finish в списке смежности MainList  Вызывать подпрограммы  FindVertexInList, StackPush, StackPop, WayFromStack | MainList – получает от фактического параметра адрес;  Start – получает от фактического параметра адрес с защитой;  Finish – получает от фактического параметра адрес с защитой;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. Res – возвраща-емый функцией параметр |
| 26 | PrQueueInit  (  Queue,  List  ) | Создать приоритетную очередь Queue из списка смежности List | Queue – получает от фактического параметра адрес;  List – получает от фактического параметра адрес с защитой | Процедура |

Продолжение Таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 27 | QueueVisited  (  Queue,  res  ) | Находить наименьшее расстояние до следующей непосещённой вершины в очереди Queue | Queue – получает от фактического параметра копию значения;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. Res – возвраща-емый функцией параметр |
| 28 | UpdateDistance  (  Queue,  Vertfromqueue,  dist  ) | Обновлять расстояние до вершин, смежных с вершиной Vertfromqueue | Queue – получает от фактического параметра адрес;  Vertfromqueue – получает от фактического параметра адрес;  Dist – получает от фактического параметра адрес с защитой | Процедура |
| 29 | Dijcstra  (  Graph,  Start,  Finish,  res  ) | Находить все кратчайшие пути из вершины Start в вершину Finish в списке смежности Graph  Вызывать подпрограммы  PrQueueInit, FindVertexInList,UpdateDistance,  Wayfromstack, QueueVisited | Graph – получает от фактического параметра адрес;  Start – получает от фактического параметра адрес с защитой;  Finish – получает от фактического параметра адрес с защитой;  res – получает от фактического параметра адрес, возвращаемый параметр | Функция. res – возвраща-емый функцией параметр |
| 30 | QuickSortWaysByLength  (  Ways,  Left,  Right  ) | Сортировать массив Ways в порядке возрастания длины | Ways – получает от фактического параметра адрес;  Left – получает от фактического параметра копию значения;  Right – получает от фактического параметра копию значения | Процедура |

## Структура данных

### Структура типов программы

Таблица 2.2 – Структура типов программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| PVertex | ^TVertex | Указатель на запись с информацией о вершине графа |
| TVertex | record  name:Integer;  x:integer;  y:integer;  backcolor:TColor;  bordercolor:TColor;  textcolor:TColor;  size:integer;  bordersize:Integer;  visited: Boolean;  selected:Boolean;  end | Информация о вершине графа.  Name – имя вершины  X,Y – координаты вершины  Backcolor – цвет фона вершины  BorderColor – цвет границы вершины  TextColor – цвет названия вершины  Size – радиус окружности вершины  Bordersize – толщина вершины  Visited – показывает была ли вершина посещена  Selected – показывает была ли вершина выделена |
| PSelect | ^TSelect | Указатель для построения списка с информацией о выделенной вершине |
| TSelect | record  Vertex:PVertex;  Color:TColor;  Next:PSelect;  Front:PSelect;  end | Информация о выделенной вершине  Vertex – указатель на информацию о вершине  Color – цвет вершины  Next – указатель на следующий элемент списка  Front – указатель на головной элемент списка |

Продолжение Таблицы 2.2

|  |  |  |
| --- | --- | --- |
| PEdge | ^TEdge | Указатель на запись с информацией о ребре графа |
| TEdge | record  vert1:integer;  vert2:integer;  weight:integer;  backcolor:TColor;  textcolor:TColor;  isUsed:Boolean;  Oriented:Boolean;  end | Информация о ребре графа  vert1,vert2 – инцидентные вершины  weight – вес ребра  backcolor – цвет ребра  textcolor – цвет веса ребра  isUsed – указывает на то, было ли ребро использовано при построении графа  Oriented – указывает на то, ориентированно ли ребро |
| PNode | ^TNode | Указатель для построения списка смежных вершин |
| TNode | record  vertex: pvertex;  edge: pedge;  next: PNode;  end | Информация о смежной вершине  Vertex – указатель на информацию о вершине  Edge – указатель на информацию об инцидентном ребре  Next – указатель на следующий элемент списка |
| PList | ^TList | Указатель для построения списка смежности |

Продолжение Таблицы 2.2

|  |  |  |
| --- | --- | --- |
| TList | record  startlist:PNode;  currentvertex:PNode;  vertexinf: pvertex;  front: plist;  rear: plist;  next: plist;  end | Элемент списка смежности  Startlist – указатель на начало списка смежных вершин  Currentvertex – указатель на текущую смежную вершину из списка смежных вершин  Vertexinf – указатель на информацию о вершине  Front – указатель на головной элемент списка  Rear – указатель на хвостовой элемент списка  Next – указатель на следующий элемент списка |
| PStack | ^TStack | Указатель на верхний элемент стека |
| TStack | record  vertex:PVertex;  distance:PEdge;  next:PStack;  end | Элемент стека для хранения пути  Vertex - указатель на информацию о вершине  Distance – указатель на инцидентное ребро  Next – указатель на следующий элемент стека |
| PQueue | ^TQueue | Указатель для создания очереди для алгоритма BFS |

Продолжение Таблицы 2.2

|  |  |  |
| --- | --- | --- |
| TQueue | record  vertex:PVertex;  distance:PEdge;  prevvertex:PStack;  next,front,rear:PQueue;  end | Элемент очереди  Vertex - указатель на информацию о вершине  Distance – указатель на инцидентное ребро  Prevvertex – указатель на вершину стека, в котором хранится путь к вершине Vertex  Next – указатель на следующий элемент очереди  Front – указатель на головной элемент очереди  Rear – указатель на хвостовой элемент очереди |
| TStackArr | array of pstack | Массив стеков для хранения всех кратчайших путей в алгоритме Дейкстры |
| PQueuePr | ^TQueuePr | Указатель для создания очереди для алгоритма Дейкстры |

Продолжение Таблицы 2.2

|  |  |  |
| --- | --- | --- |
| TQueuePr | record  vertex:PVertex;  priority:integer;  visited:Boolean;  previous:TStackArr;  next,front,rear:PQueuePr;  end | Элемент очереди  Vertex – указатель на инцидентное ребро  Priority – кратчайшее расстояние до вершины Vertex  Visited – указывает на то, было ли найдено кратчайшее расстояние до вершины  Previous – кратчайшие пути для вершины Vertex  Next – указатель на следующий элемент очереди  Front – указатель на головной элемент очереди  Rear – указатель на хвостовой элемент очереди |
| TWayInf | record  vertex:PVertex;  distance:PEdge;  end | Информация об узле пути  Vertex - указатель на информацию о вершине  Distance – указатель на инцидентное ребро |
| TWay | record  way:array of TWayInf;  len:Integer;  end; | Путь между двумя вершинами  Way – путь  Len – длина пути |
| TWays | array of TWay | Все пути между двумя вершинами |
| TPopQueue | record  vertex:Pvertex;  weight:PEdge;  way:PStack;  end | Информация об извлекаемой из очереди вершины для алгоритма BFS  Vertex - указатель на информацию о вершине  Weight – указатель на инцидентное ребро  Way – путь до вершины |

Продолжение Таблицы 2.2

|  |  |  |
| --- | --- | --- |
| TPopQueuePr | record  vertex:Pvertex;  distance:Integer;  end | Информация об извлекаемой из очереди вершины для алгоритма Дейкстры  Vertex - указатель на информацию о вершине  Distance – кратчайшее расстояние до вершины |
| PSelectWay | ^TSelectWay | Указатель для создания списка вершин в пути |
| TselectWay | record  vertex:PVertex;  Edge:PEdge;  Vcolor:TColor;  Ecolor:TColor;  Next:PSelectWay;  end | Информация о каждой вершине в пути  Vertex – информация о вершине  Edge – информация о ребре до вершины  Vcolor – цвет вершины  Ecolor – цвет ребра  Next – указатель на следующий элемент списка |

### Структура данных алгоритма ConnectedVertex

Таблица 2.3 – Структура данных алгоритма ConnectedVertex(vert1,vert2, mainlist1)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Vert1 | Integer | Имя первой вершины | Формальный |
| Vert2 | Integer | Имя второй вершины | Формальный |
| MainList1 | PList | Список смежности | Формальный |
| Result | Boolean | True – если вершины соединены ребром, иначе – False | Формальный |

Продолжение Таблицы 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| point1 | PList | Адрес первой вершины | Локальный |
| point2 | PList | Адрес второй вершины | Локальный |
| tmp | PNode | Текущая вершина в списке смежности | Локальный |

### Структура данных алгоритма AddVertex

Таблица 2.4 – Структура данных алгоритма AddVertex (pnewvertex, vertex)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| pnewvertex | Plist | Список смежности | Формальный |
| vertex | TVertex | Новая вершина | Формальный |
| pvertexinf | Pvertex | Указатель на информацию о новой вершине | Локальный |
| oldvertex | PList | Вспомогательная переменная для добавления вершины | Локальный |

### Структура данных алгоритма addsemivertex

Таблица 2.5 – Структура данных алгоритма addsemivertex (source, destination, SemiEdge, newedge, Directed)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| source | PList | Адрес первой вершины | Формальный |
| destination | PList | Адрес второй вершины | Формальный |
| SemiEdge | TEdge | Новое ребро | Формальный |
| newedge | PEdge | Указатель на новое ребро | Формальный |
| Directed | Boolean | Указывает на то, ориентированно ребро или нет | Формальный |
| lastvertex | PNode | Адрес последнего элемента списка смежных вершин | Локальный |

Продолжение Таблицы 2.5

|  |  |  |  |
| --- | --- | --- | --- |
| newvertex | PNode | Новый элемент списка смежных вершин | Локальный |

### Структура данных алгоритма AddEdge

Таблица 2.6 – Структура данных алгоритма AddEdge (fvertex, svertex, Edge, mainlist, oriented)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| fvertex | Integer | Имя первой вершины | Формальный |
| svertex | Integer | Имя второй вершины | Формальный |
| Edge | TEdge | Новое ребро | Формальный |
| mainlist | PList | Список смежности | Формальный |
| oriented | Boolean | Указывает на то, ориентированно ребро или нет | Формальный |
| point1 | PList | Адрес первой вершины | Локальный |
| point2 | PList | Адрес второй вершины | Локальный |
| edgepointer | PEdge | Указатель на новое ребро | Локальный |

### Структура данных алгоритма deleteSemiVertex

Таблица 2.7 – Структура данных алгоритма deleteSemiVertex (source, destination)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| source | PList | Адрес первой вершины | Формальный |
| destination | PList | Адрес второй вершины | Формальный |
| tmp | PNode | Указатель списка смежных вершин вершины source | Локальный |

Продолжение Таблицы 2.7

|  |  |  |  |
| --- | --- | --- | --- |
| flag | Boolean | Указывает на то, была ли найдена вершина destination в списке смежных вершин вершины source | Локальный |

### Структура данных алгоритма DeleteEdge

Таблица 2.8 – Структура данных алгоритма DeleteEdge (fvertex, svertex, mainlist)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| fvertex | Integer | Имя первой вершины | Формальный |
| svertex | Integer | Имя второй вершины | Формальный |
| mainlist | PList | Список смежности | Формальный |
| point1 | PList | Адрес первой вершины | Локальный |
| Point2 | PList | Адрес второй вершины | Локальный |

### Структура данных алгоритма DeleteVertex

Таблица 2.9 – Структура данных алгоритма DeleteVertex (vertexname, mainlist)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| vertexname | Integer | Имя вершины | Формальный |
| mainlist | PList | Список смежности | Формальный |
| vertexcopy | PList | Вспомогательная переменная для хранения списка смежности | Локальный |
| tmp | PList | Обновление указателя на хвостовой элемент | Локальный |

### Структура данных алгоритма HeadCreate

Таблица 2.10 – Структура данных алгоритма HeadCreate (newvert)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| newvert | PList | Указатель на начало списка смежности | Формальный |

### Структура данных алгоритма FindVertexInList

Таблица 2.11 – Структура данных алгоритма FindVertexInList (searchingName, ListPointer)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| searchingName | Integer | Имя вершины, которую нужно найти в списке смежности | Формальный |
| ListPointer | PList | Список смежности | Формальный |

### Структура данных алгоритма WayFromStack

Таблица 2.12 – Структура данных алгоритма WayFromStack (Stack)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Stack | PStack | Указатель на вершину стека путей | Формальный |
| Result | Tway | Путь между вершинами | Формальный |
| TmpRecord | TwayInf | Вспомогательная переменная для хранения информации о вершине стека | Локальный |

### Структура данных алгоритма SetVertProperties

Таблица 2.13 – Структура данных алгоритма SetVertProperties (name, x, y)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| name | Integer | Имя новой вершины | Формальный |

Продолжение Таблицы 2.13

|  |  |  |  |
| --- | --- | --- | --- |
| x | Integer | Координата новой вершины по оси абсцисс | Формальный |
| Y | Integer | Координата новой вершины по оси ординат | Формальный |
| Result | TVertex | Информация о новой вершине | Формальный |

### Структура данных алгоритма SetEdgeProperties

Таблица 2.14 – Структура данных алгоритма SetEdgeProperties ()

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Result | TEdge | Информация о новом ребре | Формальный |

### Структура данных алгоритма SaveListToFile

Таблица 2.15 – Структура данных алгоритма SaveListToFile (Graph, EfileName, VFileName)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | PList | Список смежности | Формальный |
| EfileName | String | Путь к файлу с рёбрами | Формальный |
| VFileName | String | Путь к файлу с вершинами | Формальный |
| FileEdge | File of TEdge | Типизированный файл для хранения информации о рёбрах графа | Локальный |
| FileVert | File of TVertex | Типизированный файл для хранения информации о вершинах графа | Локальный |
| CurVert | Plist | Вспомогательная переменная для хранения списка смежности | Локальный |

Продолжение Таблицы 2.15

|  |  |  |  |
| --- | --- | --- | --- |
| CurSemiVert | PNode | Вспомогательная переменная для хранения списка смежных вершин | Локальный |

### Структура данных алгоритма CreateFromFile

Таблица 2.16 – Структура данных алгоритма CreateFromFile (Graph, EfileName, VFileName)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | PList | Список смежности | Формальный |
| EfileName | String | Путь к файлу с рёбрами | Формальный |
| VFileName | String | Путь к файлу с вершинами | Формальный |
| FileEdge | File of TEdge | Типизированный файл для хранения информации о рёбрах графа | Локальный |
| FileVert | File of TVertex | Типизированный файл для хранения информации о вершинах графа | Локальный |
| Node | Plist | Вспомогательная переменная для хранения вершины из файла | Локальный |
| Arc | PNode | Вспомогательная переменная для хранения ребра из файла | Локальный |

### Структура данных алгоритма StackPush

Таблица 2.17 – Структура данных алгоритма StackPush (NodeList, NewElP)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| NodeList | PList | Список смежности | Формальный |

Продолжение Таблицы 2.17

|  |  |  |  |
| --- | --- | --- | --- |
| NewElP | PStack | Указатель на новый элемент стека | Формальный |
| Result | Boolean | Указывает на то, можно ли добавить новую переменную в стек | Формальный |
| NewNode | PStack | Указатель на новый элемент стека | Локальный |

### Структура данных алгоритма StackPop

Таблица 2.18 – Структура данных алгоритма StackPop (OldElP, ChgIncVert)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| ChgIncVert | PList | Список смежности | Формальный |
| OldElP | PStack | Указатель на удаляемый элемент стека | Формальный |
| Result | Boolean | Указывает на то, можно ли извлечь элемент из стека | Формальный |

### Структура данных алгоритма DFS

Таблица 2.19 – Структура данных алгоритма DFS (MainList, start, finish)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| MainList | PList | Список смежности | Формальный |
| start | Integer | Имя стартовой вершины | Формальный |
| finish | Integer | Имя конечной вершины | Формальный |
| Result | TWays | Массив путей от стартовой вершины до конечной | Формальный |
| startp | PStack | Создание вершины стека | Локальный |
| source | PList | Адрес первой вершины | Локальный |
| destination | PList | Адрес второй вершины | Локальный |

Продолжение Таблицы 2.19

|  |  |  |  |
| --- | --- | --- | --- |
| tmpvertex | PList | Вспомогательная переменная для хранения списка смежности | Локальный |
| tmpwayinf | TwayInf | Хранит информацию о вершине, полученной из стека | Локальный |
| tmpway | TWay | Путь между вершинами | Локальный |
| res | Boolean | Возможность/невозможность добавлять вершину в стек | Локальный |

### Структура данных алгоритма NodeInStack

Таблица 2.20 – Структура данных алгоритма NodeInStack (Stack, Node)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Stack | PStack | Указатель на стек для хранения пути | Формальный |
| Node | PVertex | Информация о вершине | Формальный |
| Result | Boolean | True – если вершина находится в стеке, иначе – False | Формальный |

### Структура данных алгоритма QueueInit

Таблица 2.21 – Структура данных алгоритма QueueInit (PQu)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| PQu | PQueue | Указатель на очередь для алгоритма BFS | Формальный |

### Структура данных алгоритма QueuePush

Таблица 2.22 – Структура данных алгоритма QueuePush (newvertex, PQu)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| PQu | PQueue | Указатель на очередь для алгоритма BFS | Формальный |
| newvertex | PNode | Указатель на вершину для добавления в очередь | Формальный |
| tmp | PQueue | Вспомогательная переменная для обновления хвостового элемента очереди | Локальный |

### Структура данных алгоритма QueuePop

Таблица 2.23 – Структура данных алгоритма QueuePop (PQu)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| PQu | PQueue | Указатель на очередь для алгоритма BFS | Формальный |
| Result | PTopQueue | Извлечённая из очереди вершина с инцидентным ребром | Формальный |

### Структура данных алгоритма BFS

Таблица 2.24 – Структура данных алгоритма BFS (MainList, start, finish)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| MainList | PList | Список смежности | Формальный |
| start | Integer | Имя стартовой вершины | Формальный |
| finish | Integer | Имя конечной вершины | Формальный |

Продолжение Таблицы 2.24

|  |  |  |  |
| --- | --- | --- | --- |
| Result | TWays | Массив путей от стартовой вершины до конечной | Формальный |
| Queue | PQueue | Указатель на очередь | Локальный |
| source | PList | Адрес первой вершины | Локальный |
| destination | PList | Адрес второй вершины | Локальный |
| tmpvertex | PList | Вспомогательная переменная для хранения списка смежности | Локальный |
| FromQ | TPopQueue | Информация об извлекаемой из очереди вершины | Локальный |
| tmpwayinf | TwayInf | Хранит информацию о вершине, полученной из стека | Локальный |
| tmpway | TWay | Путь между вершинами | Локальный |
| Prevvertex | PStack | Указатель на вершину стека для пути | Локальный |
| newVertex | PStack | Новый элемент стека для пути | Локальный |

### Структура данных алгоритма PrQueueInit

Таблица 2.25 – Структура данных алгоритма PrQueueInit (Queue, list)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Queue | PQueuePr | Указатель на очередь для Алгоритма Дейкстры | Формальный |
| list | PList | Список смежности | Формальный |
| TmpNode | PList | Вспомогательная переменная для добавления всех элементов списка смежности в очередь | Локальный |

Продолжение Таблицы 2.25

|  |  |  |  |
| --- | --- | --- | --- |
| Front | PQueuePr | Переменная для хранения адреса головного элемента очереди | Локальный |

### Структура данных алгоритма QueueVisited

Таблица 2.26 – Структура данных алгоритма QueueVisited (Queue)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Queue | PQueuePr | Указатель на очередь для Алгоритма Дейкстры | Формальный |
| Result | TPopQueuePr | Извлечённый из очереди элемент | Формальный |
| MinA | PQueuePr | Вершина в очереди с минимальный расстоянием до неё | Локальный |
| First | Boolean | Переменная для присваивания переменной MinA начально значения | Локальный |

### Структура данных алгоритма UpdateDistance

Таблица 2.27 – Структура данных алгоритма UpdateDistance (Queue, vertfromqueue, dist)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Queue | PQueuePr | Указатель на очередь для Алгоритма Дейкстры | Формальный |
| vertfromqueue | PList | Извлечённый из очереди элемент | Формальный |
| dist | integer | Расстояние до последнего извлечённого элемента | Формальный |
| i | Integer | Счётчик цикла | Локальный |

Продолжение Таблицы 2.27

|  |  |  |  |
| --- | --- | --- | --- |
| Stack | PStack | Указатель на вершину стека для пути | Локальный |
| TmpVertex | PQueuePr | Вспомогательная переменная для работы с очередью | Локальный |

### Структура данных алгоритма Dijcstra

Таблица 2.28 – Структура данных алгоритма Dijcstra (Graph, start, finish)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | PList | Список смежности | Формальный |
| start | Integer | Имя стартовой вершины | Формальный |
| finish | Integer | Имя конечной вершины | Формальный |
| Result | TWays | Массив путей от стартовой вершины до конечной | Формальный |
| QAddr | PQueue | Указатель на очередь | Локальный |
| distance | Integer | Расстояние до последнего извлечённого элемента |  |
| source | PList | Адрес первой вершины | Локальный |
| destination | PList | Адрес второй вершины | Локальный |
| i | Integer | Счётчик цикла | Локальный |
| StartVert | TWayinf | Информация о начальной вершине | Локальный |
| CurrentVertex | Integer | Имя текущей вершины | Локальный |
| AddWay | TPopQueuePr | Извлечённый из очереди элемент | Локальный |
| DelEl | PQueuePr | Указатель на извлечённый элемент | Локальный |

### Структура данных алгоритма QuickSortWaysByLength

Таблица 2.29 – Структура данных алгоритма QuickSortWaysByLength (Ways, Left, Right)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Ways | TWays | Двумерный массив путей для сортировки | Формальный |
| Left | Integer | Левая граница сортировки | Формальный |
| Right | Integer | Правая граница сортировки | Формальный |
| I | Integer | Индекс элемента левее опорного | Локальный |
| J | Integer | Индекс элемента правее опорного | Локальный |
| Pivot | Integer | Индекс опорного элемента | Локальный |
| Temp | TWays | Вспомогательная переменная для обмена местами двух элементов | Локальный |

### Структура данных алгоритма SaveGraphToExcel

Таблица 2.30 – Структура данных алгоритма SaveGraphToExcel (Graph, way)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | PList | Список смежности | Формальный |
| way | Way | Адрес сохраняемого файла | Формальный |
| ExcelApp | Variant | Приложение Excel | Локальный |
| Workbook | Variant | Книга Excel | Локальный |
| Node | PNode | Список смежности | Локальный |
| SheetVertices | Variant | Страница вершин | Локальный |
| SheetEdges | Variant | Страница рёбер | Локальный |
| VertexRow | Integer | Строка на странице вершин | Локальный |
| EdgeRow | Integer | Строка на странице рёбер | Локальный |

Таблица 2.31 – Структура данных алгоритма LoadGraphFromExcel (filename, Graph)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Graph | PList | Список смежности | Формальный |
| filename | Way | Адрес сохраняемого файла | Формальный |
| ExcelApp | Variant | Приложение Excel | Локальный |
| Workbook | Variant | Книга Excel | Локальный |
| Node | TVertex | Информация о вершине | Локальный |
| SheetVertices | Variant | Страница вершин | Локальный |
| SheetEdges | Variant | Страница рёбер | Локальный |
| Arc | TEdge | Информация о ребре | Локальный |

## Схемы алгоритмов решения задачи по ГОСТ 19.701-90

### Схема алгоритма Dijcstra



Рисунок 2.1 – Схема алгоритма Dijcstra (часть 1)



Рисунок 2.1 – Схема алгоритма Dijcstra (часть 2)



Рисунок 2.1 – Схема алгоритма Dijcstra (часть 3)



Рисунок 2.1 – Схема алгоритма Dijcstra (часть 4)

### Схема алгоритма PrQueueInit



Рисунок 2.2 – Схема алгоритма PrQueueInit (часть 1)



Рисунок 2.2 – Схема алгоритма PrQueueInit (часть 2)

### Схема алгоритма SaveListToFile



Рисунок 2.3 – Схема алгоритма SaveListToFile (часть 1)



Рисунок 2.3 – Схема алгоритма SaveListToFile (часть 2)

### Схема алгоритма CreateFromFile



Рисунок 2.4 – Схема алгоритма CreateFromFile (часть 1)



Рисунок 2.4 – Схема алгоритма CreateFromFile (часть 2)

### Схема алгоритма QueueVisited



Рисунок 2.5 – Схема алгоритма QueueVisited (часть 1)



Рисунок 2.5 – Схема алгоритма QueueVisited (часть 2)

### Схема алгоритма WayFromStack



Рисунок 2.6 – Схема алгоритма WayFromStack

### Схема алгоритма QueuePop



Рисунок 2.7 – Схема алгоритма QueuePop

### Схема алгоритма AddVertex



Рисунок 2.8 – Схема алгоритма AddVertex (часть 1)



Рисунок 2.8 – Схема алгоритма AddVertex (часть 2)

### Схема алгоритма AddEdge



Рисунок 2.9 – Схема алгоритма AddEdge

### Схема алгоритма DeleteEdge



Рисунок 2.10 – Схема алгоритма DeleteEdge

## Графический интерфейс

Для организации графического интерфейса приложения было использовано 3 формы: FMenu, MainForm, FormWays. Взаимосвязь между формами представлена на Рис. 2.11.



Рисунок 2.11 – Взаимосвязь графических форм

### Описание графических компонентов формы FMenu

Форма FMenu – это главное меню программного средства. С нее пользователь может перейти к форме создания графа.

Данная форма имеет вид, представленный на Рис. 2.12.

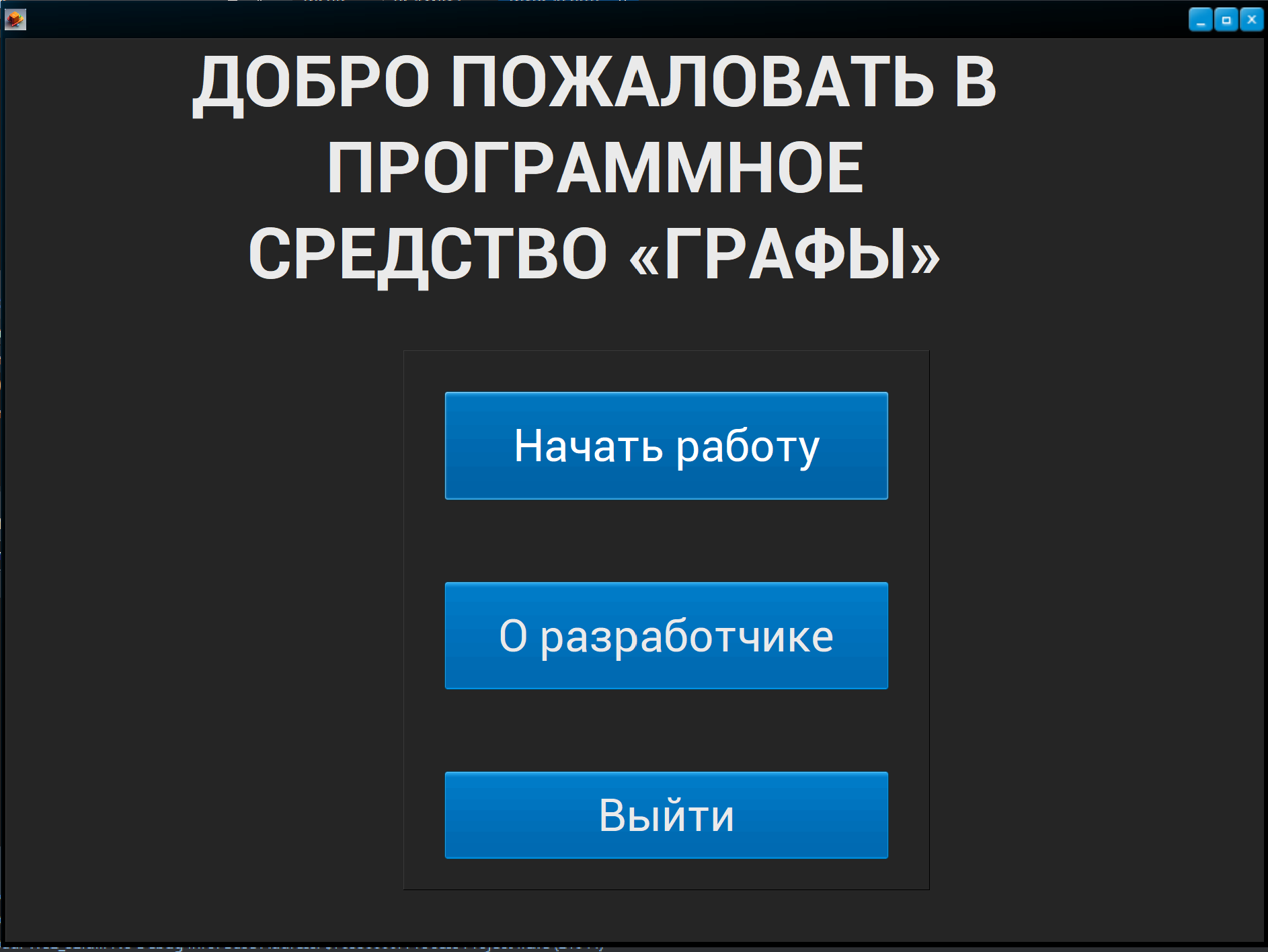


Рисунок 2.12 – Вид формы FMenu при запуске программы

Составляющие формы FMenu:

* быстрая кнопка «Начать работу», которая открывает окно работы с графами MainForm;
* быстрая кнопка «О разработчике», которая выводит информацию о разработчике программного средства;
* быстрая кнопка «Выйти», которая завершает работу программного средства.

### Описание графических компонентов формы MainForm

Форма MainForm предназначена для создания, редактирования графа, сохранения графа в файл (в том числе и файл Excel) и построения графа из файла. Выбора алгоритма нахождения путей.

Данная форма при открытии имеет вид, представленный на Рис. 2.13.

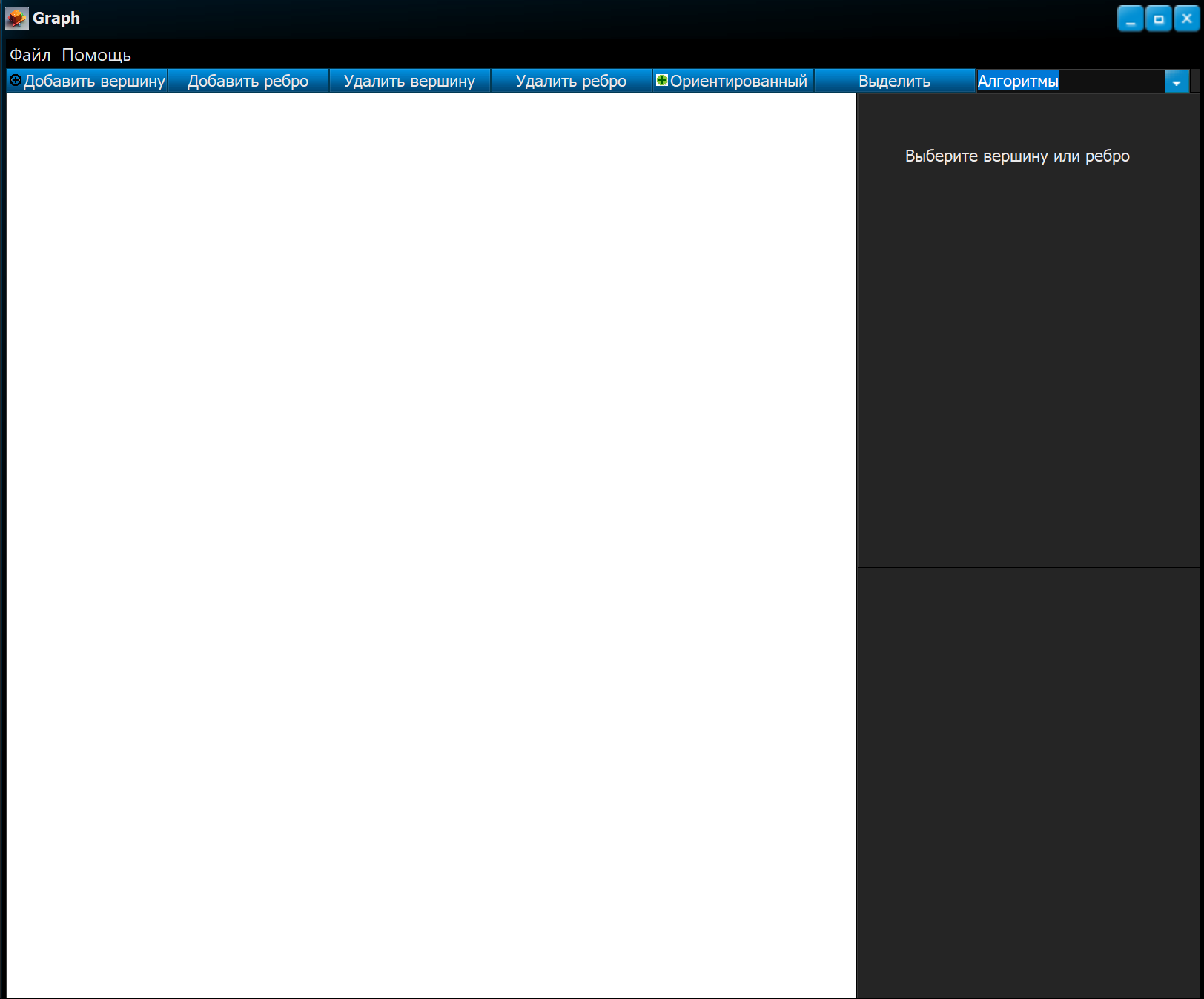


Рисунок 2.13 – Внешний вид формы MainForm при открытии

При выделении вершины форма приобретает вид, представленный на Рис. 2.14.

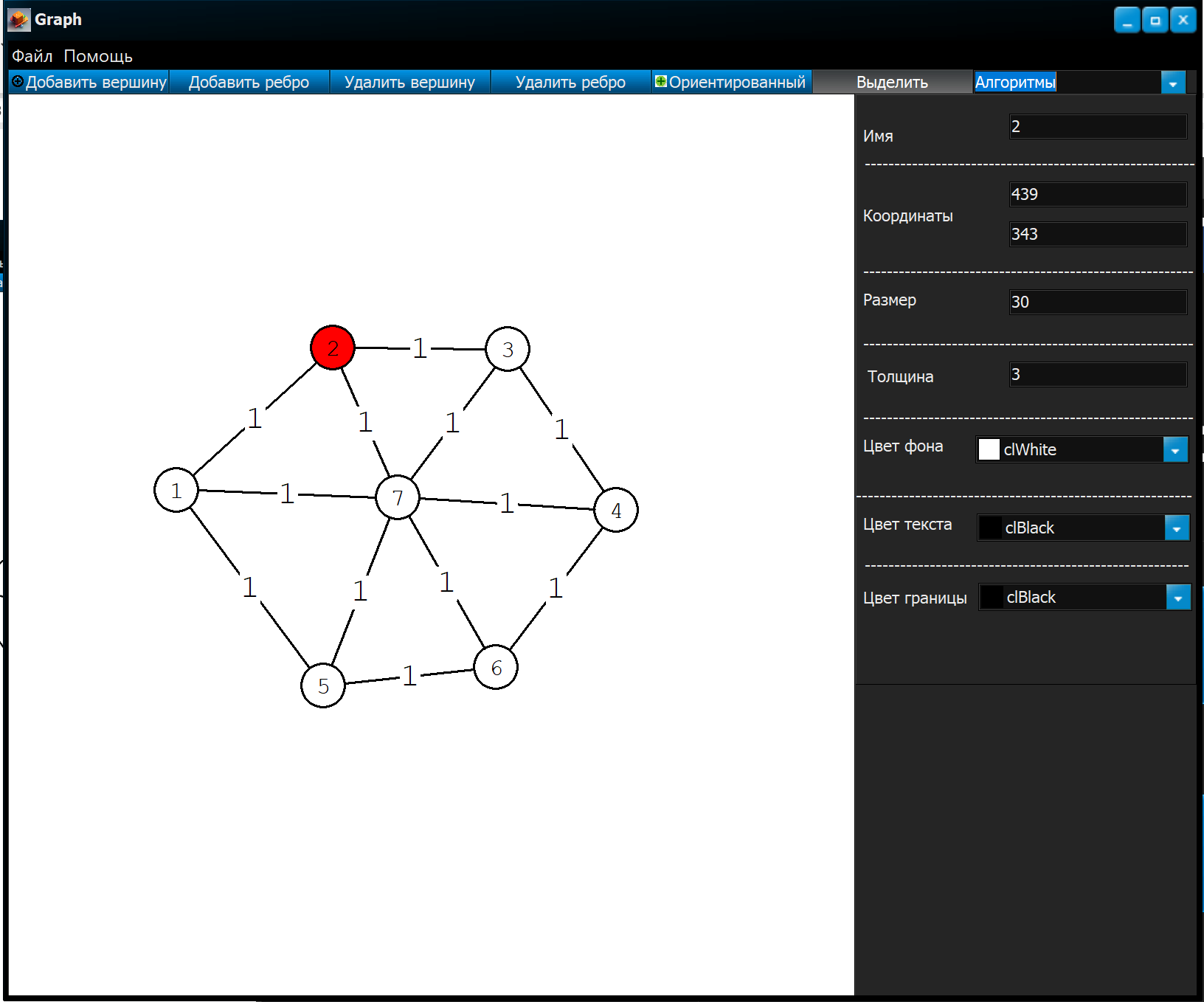


Рисунок 2.14 – Внешний вид формы MainForm при выделении вершины

При выделении ребра форма приобретает вид, представленный на Рис. 2.15.

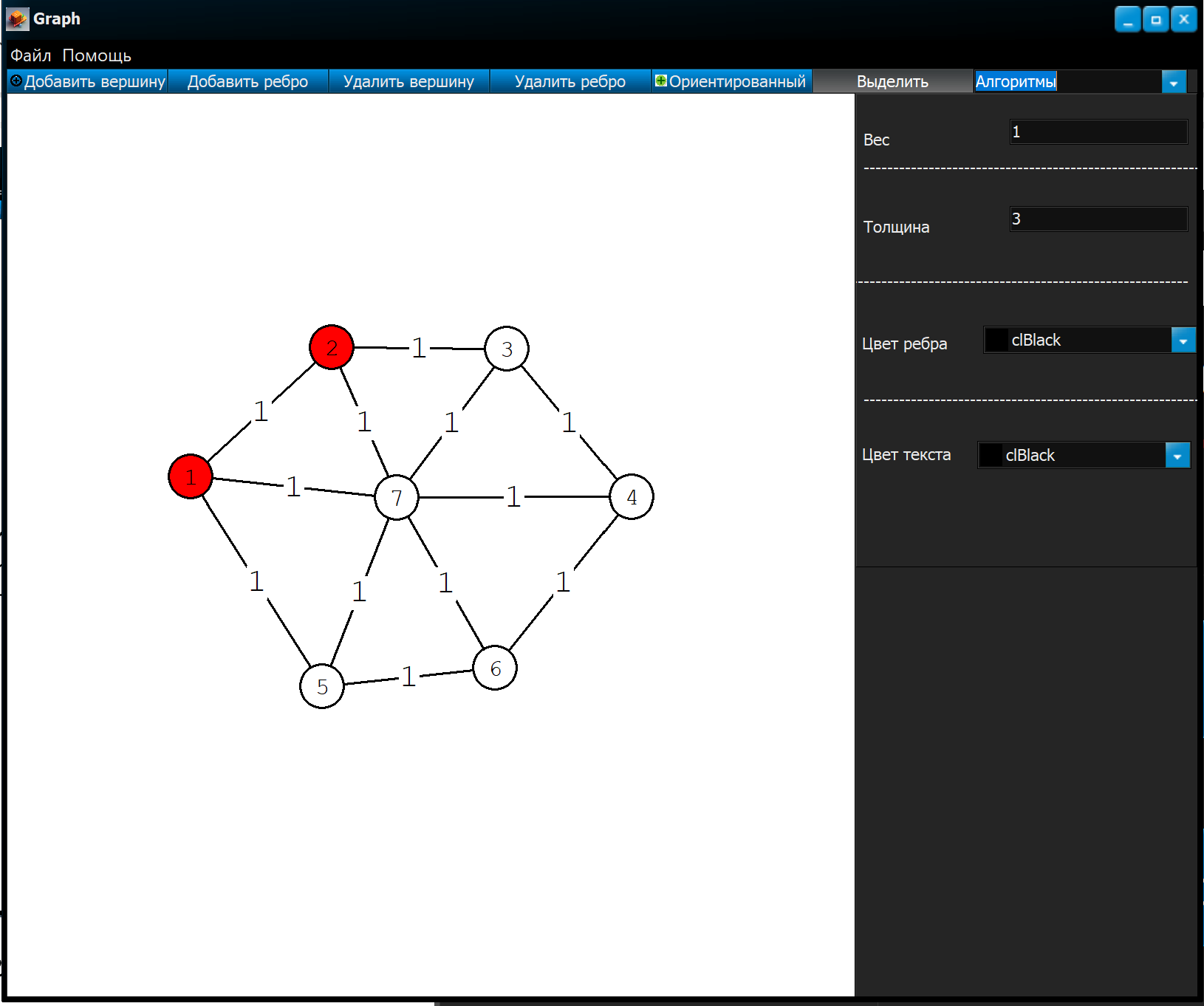


Рисунок 2.15 – Внешний вид формы MainForm при выделении ребра

При выделении пути форма приобретает вид, представленный на Рис. 2.16.

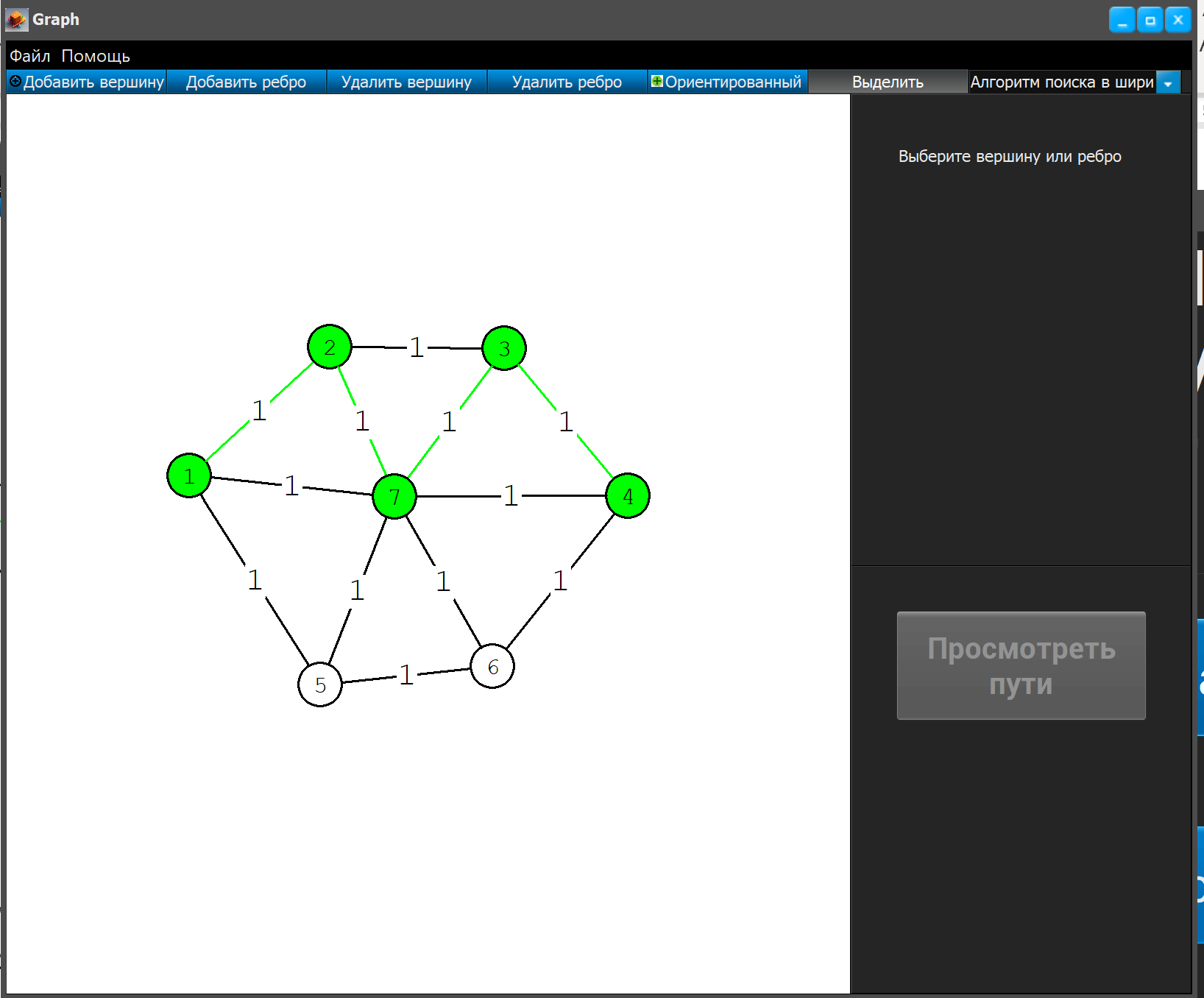


Рисунок 2.16 – Внешний вид формы MainForm при выборе пути

Составляющие формы MainForm:

* главное меню «MainMenu», для выбора текущего действия (сохранять граф в файл или открывать из него);
* панель инструментов «Toolbar», на котором расположены следующие компоненты:

1. Быстрая кнопка «Добавить вершину», которая позволяет пользователю добавлять вершину в граф через холст.
2. Быстрая кнопка «Удалить вершину», которая позволяет пользователю удалить выделенные вершины.
3. Быстрая кнопка «Добавить ребро», которая позволяет пользователю добавить ребро между двумя не смежными вершинами, предварительно выделив их.
4. Быстрая кнопка «Удалить ребро», которая позволяет пользователю удалить ребро между двумя смежными вершинами, предварительно выделив их.
5. Быстрая кнопка «Ориентированный», которая позволяет пользователю выбрать, будут ли добавляемые дуги ориентированными или нет.
6. Быстрая кнопка «Выделить», которая позволяет пользователю выделять вершины/рёбра на Paintbox, а также перемещать их.
7. Выпадающий список, который позволяет пользователю выбрать алгоритм нахождения путей, для выделенных двух вершин графа.

* холст для отрисовки графа, выделенных вершин/рёбер, путей;
* область, содержащая поля для ввода и палитры цветов, для изменения свойств вершин графа;
* область, содержащая поля для ввода и палитры цветов, для изменения свойств рёбер графа;
* быстрая кнопка «Просмотреть пути», которая открывает окно работы с путями FormWays.

### Описание графических компонентов формы FormWays

Форма FormWays, предназначена для вывода информации о всех путях между двумя вершинами, найденных выбранным пользователем алгоритмом, и их сортировки.

Данная форма имеет вид, представленный на Рис. 2.17.

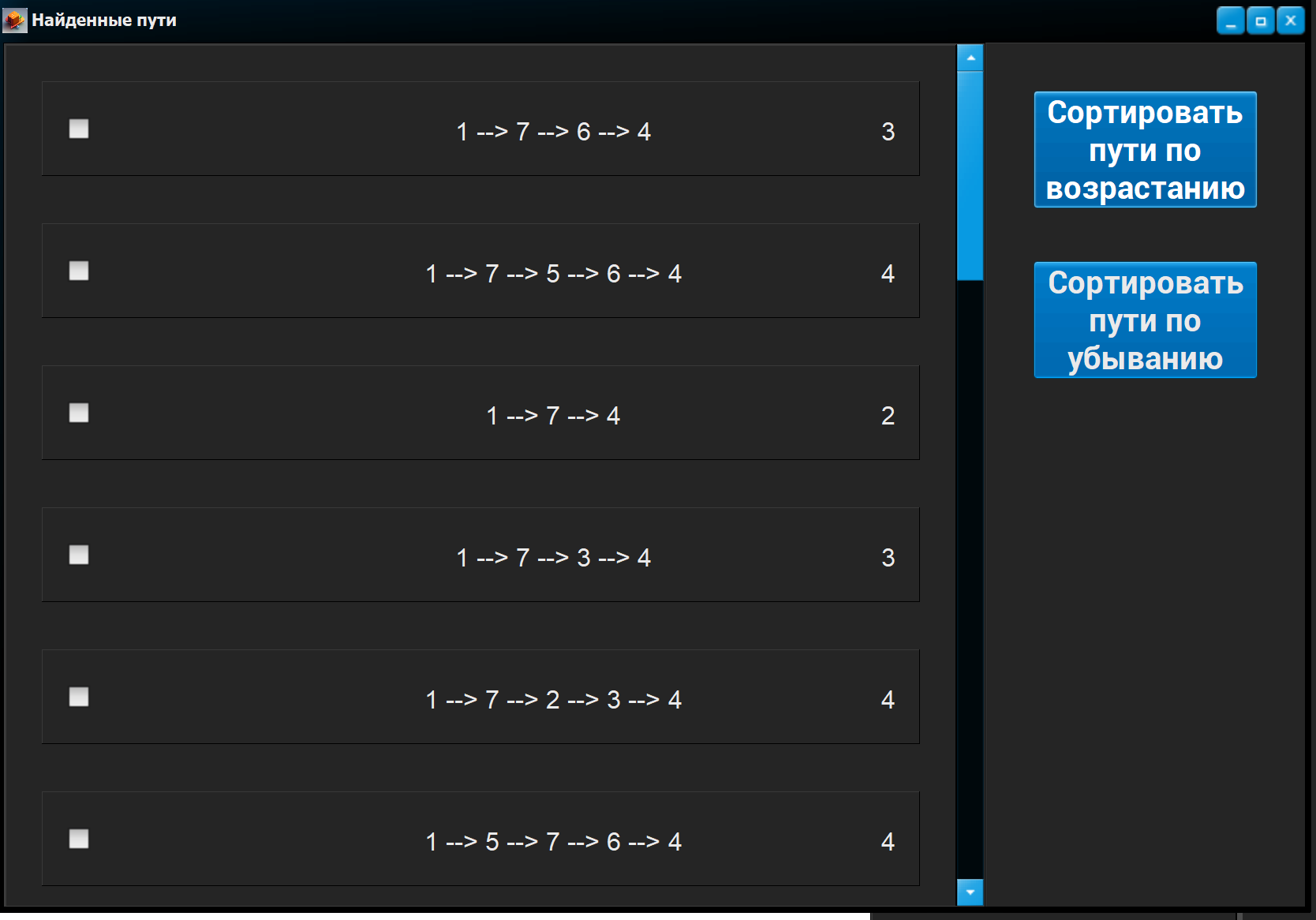


Рисунок 2.17 – Внешний вид формы FormWays при открытии

Составляющие формы FormWays:

* быстрая кнопка «Сортировать пути по возрастанию», для сортировки найденных путей по возрастанию их длины;
* быстрая кнопка «Сортировать пути по убыванию», для сортировки найденных путей по убыванию их длины;
* панель управления путями для просмотра всех найденных путей и выбора пути для отрисовки на холсте окна MainForm.

# Тестирование программного средства

## Навигация в программном средстве

### Тест 1

Таблица 3.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при запуске |
| Исходные данные | Запуск программного средства |
| Ожидаемый результат | Открытие главной формы |
| Получен-ный результат |  |

### Тест 2

Таблица 3.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Начать работу» |
| Исходные данные | Выбор быстрого действия «Начать работу» |
| Ожидаемый результат | Открытие формы работы с графом «MainForm» |

Продолжение Таблицы 3.2

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 3

Таблица 3.3 – Тест 3

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «О разработчике» |
| Исходные данные | Выбор быстрого действия «О разработчике» |
| Ожидаемый результат | Вывод сообщения о разработчике программного средства |

Продолжение Таблицы 3..3

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 4

Таблица 3.4 – Тест 4

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Выход» |
| Исходные данные | Выбор быстрого действия «Выход» |
| Ожидаемый результат | Закрытие программного средства |
| Получен-ный результат | Закрытие программного средства |

## Работа формы MainForm

### Тест 1

Таблица 3.5 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Добавить вершину» |
| Исходные данные | Выбор действия «Добавить вершину» |
| Ожидаемый результат | Активация режима добавления вершины |

Продолжение Таблицы 3.5

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 2

Таблица 3.6 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Удалить вершину» |
| Исходные данные | Выбор действия «Удалить вершину» |
| Ожидаемый результат | Активация режима удаления вершины |

Продолжение Таблицы 3.6

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 3

Таблица 3.7 – Тест 3

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Добавить ребро» |
| Исходные данные | Выбор действия «Добавить ребро» |
| Ожидаемый результат | Активация режима добавления ребра |

Продолжение Таблицы 3.7

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 4

Таблица 3.8 – Тест 4

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Удалить ребро» |
| Исходные данные | Выбор действия «Удалить ребро» |
| Ожидаемый результат | Активация режима удаления ребра |

Продолжение Таблицы 3.8

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 5

Таблица 3.9 – Тест 5

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Выделить» |
| Исходные данные | Выбор действия «Выделить» |
| Ожидаемый результат | Активация режима выделения |

Продолжение Таблицы 3.9

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 6

Таблица 3.10 – Тест 6

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме добавления вершины |
| Исходные данные | Активирован режим добавления вершины. Нажатие левой клавишей мыши на холст |
| Ожидаемый результат | Отображение вершины графа на холсте |

Продолжение Таблицы 3.10

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 7

Таблица 3.11 – Тест 7

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме выделения |
| Исходные данные | Активирован режим выделения. Нажатие левой клавишей мыши на холст в пределах отрисованной вершины |
| Ожидаемый результат | Изменение цвета отображаемой вершины, появление информации о выделенной вершине в боковой панели |

Продолжение Таблицы 3.11

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 8

Таблица 3.12 – Тест 8

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме выделения |
| Исходные данные | Активирован режим выделения. На холсте изображено две вершины. Поочерёдное нажатие на обе вершины левой клавишей мыши с зажатой клавишей Ctrl |
| Ожидаемый результат | Изменение цвета отображаемых вершин |

Продолжение Таблицы 3.12

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 9

Таблица 3.13 – Тест 9

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме добавления ребра |
| Исходные данные | Активирован режим добавления ребра. На холсте изображены две выделенные вершины |
| Ожидаемый результат | Появление ребра между выделенными вершинами, появление информации о ребре в боковой панели |

Продолжение Таблицы 3.13

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 10

Таблица 3.14 – Тест 10

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме удаления ребра |
| Исходные данные | Активирован режим удаления ребра. На холсте изображены две выделенные вершины, соединённые ребром |
| Ожидаемый результат | Удаление выделенного ребра |

Продолжение Таблицы 3.14

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 11

Таблица 3.15 – Тест 11

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме удаления вершины |
| Исходные данные | Активирован режим удаления вершины. На холсте изображены две вершины, соединённые ребром. Выделена только одна вершина |
| Ожидаемый результат | Удаление выделенной вершины и ребра |

Продолжение Таблицы 3.15

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 12

Таблица 3.16 – Тест 12

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме выделения |
| Исходные данные | Активирован режим выделения вершины. На холсте изображены две выделенные вершины, соединённые ребром. Передвижение мыши по холсту с зажатой клавишей |
| Ожидаемый результат | Изменение положения вершин и ребра на холсте |

Продолжение Таблицы 3.16

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 13

Таблица 3.17 – Тест 13

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства в режиме выбора алгоритма |
| Исходные данные | Граф. Выделены две вершины. В выпадающем списке выбран один из трёх алгоритмов |
| Ожидаемый результат | Появление быстрой кнопки «Просмотреть пути» на боковой панели |

Продолжение Таблицы 3.17

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 14

Таблица 3.18 – Тест 14

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при открытии типизированного файла |
| Исходные данные | Типизированный файл, содержащий граф |
| Ожидаемый результат | Отрисовка графа из файла на холст |

Продолжение Таблицы 3.18

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 15

Таблица 3.19 – Тест 15

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при сохранении графа в типизированный файл |
| Исходные данные | Граф |
| Ожидаемый результат | Появление типизированного файла, содержащего граф |
| Получен-ный результат |  |

### Тест 16

Таблица 3.20 – Тест 16

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при сохранении графа в файл Excel |

Продолжение Таблицы 3.20

|  |  |
| --- | --- |
| Исходные данные | Граф |
| Ожидаемый результат | Появление файла Excel, содержащего граф |
| Получен-ный результат |  |

### Тест 17

Таблица 3.21 – Тест 17

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при открытии файла Excel |
| Исходные данные | файл Excel, содержащий граф |
| Ожидаемый результат | Отрисовка графа из файла на холст |

Продолжение Таблицы 3.21

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 18

Таблица 3.22 – Тест 18

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при сохранении графа в формате .bmp |
| Исходные данные | Граф |
| Ожидаемый результат | Появление файла формата .bmp, содержащего граф |

Продолжение Таблицы 3.22

|  |  |
| --- | --- |
| Получен-ный результат |  |

## Работа формы FormWays

### Тест 1

Таблица 3.23 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе пути из списка |
| Исходные данные | Граф, список путей найденных алгоритмом |
| Ожидаемый результат | Отображение выбранного пути в окне MainForm |

Продолжение Таблицы 3.23

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 2

Таблица 3.24 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Сортировать пути по возрастанию» |
| Исходные данные | Граф, список путей найденных алгоритмом |
| Ожидаемый результат | Отображение найденных путей в порядке возрастания их длины |

Продолжение Таблицы 3.24

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 3

Таблица 3.25 – Тест 3

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности поведения программного средства при выборе быстрого действия «Сортировать пути по убыванию» |
| Исходные данные | Граф, список путей найденных алгоритмом |
| Ожидаемый результат | Отображение найденных путей в порядке убывания их длины |

Продолжение Таблицы 3.25

|  |  |
| --- | --- |
| Получен-ный результат |  |

## Работа алгоритма Дейкстры

### Тест 1

Таблица 3.26 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы алгоритма Дейкстры. Нахождение пути из одной вершины в другую, при условии, что попасть в неё невозможно |
| Исходные данные | Список смежности:  1: 3;  3: #;  4: #; |
| Ожидаемый результат | Пути не найдены |

Продолжение Таблицы 3.26

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 2

Таблица 3.27 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы алгоритма Дейкстры. Нахождение пути из одной вершины в другую, при условии, что такой путь (пути) существуют и все эти пути имеют одну длину |
| Исходные данные | Список смежности:  1: 2,3;  2: 4;  3: 4;  4: #; |
| Ожидаемый результат | Пути:  1 – 3 – 4  1 – 2 – 4 |

Продолжение Таблицы 3.27

|  |  |
| --- | --- |
| Получен-ный результат |  |

Таблица 3.28 – Тест 3

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы алгоритма Дейкстры. Нахождение пути из одной вершины в другую, при условии, что такой путь (пути) существуют. |
| Исходные данные | Список смежности:  1: 2,3,5;  2: 4;  3: 4;  4: #;  5: 3; |
| Ожидаемый результат | Пути:  1 – 3 – 4  1 – 2 – 4 |

Продолжение Таблицы 3.28

|  |  |
| --- | --- |
| Получен-ный результат |  |

## Работа алгоритма поиска в ширину

### Тест 1

Таблица 3.29 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы алгоритма поиска в ширину. Нахождение пути из одной вершины в другую, при условии, что попасть в неё невозможно |
| Исходные данные | Список смежности:  1: 2;  2: #; |
| Ожидаемый результат | Пути не найдены |

Продолжение Таблицы 3.29

|  |  |
| --- | --- |
| Получен-ный результат |  |

### Тест 2

Таблица 3.30 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы алгоритма Дейкстры. Нахождение пути из одной вершины в другую, при условии, что такой путь (пути) существуют. |
| Исходные данные | Список смежности:  1: 2,3,5;  2: 4;  3: 4;  4: #;  5: 3; |
| Ожидаемый результат | Пути:  1 – 3 – 4  1 – 2 – 4  1 – 5 – 3 – 4 |

Продолжение Таблицы 3.30

|  |  |
| --- | --- |
| Получен-ный результат |  |

## Работа алгоритма поиска в глубину

Таблица 3.31 – Тест 1

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы алгоритма поиска в глубину. Нахождение пути из одной вершины в другую, при условии, что попасть в неё невозможно |
| Исходные данные | Список смежности:  1: 2;  2: #; |
| Ожидаемый результат | Пути не найдены |

Продолжение Таблицы 3.31

|  |  |
| --- | --- |
| Получен-ный результат |  |

Таблица 3.32 – Тест 2

|  |  |
| --- | --- |
| Тестовая ситуация | Проверка корректности работы алгоритма поиска в глубину. Нахождение пути из одной вершины в другую, при условии, что такой путь (пути) существуют. |
| Исходные данные | Список смежности:  1: 2,3,5;  2: 4;  3: 4;  4: #;  5: 3; |
| Ожидаемый результат | Пути:  1 – 3 – 4  1 – 2 – 4  1 – 5 – 3 – 4 |

Продолжение Таблицы 3.32

|  |  |
| --- | --- |
| Получен-ный результат |  |

# Руководство по установке и использованию программного средства

## Минимальные системные требования

Для успешного запуска программного средства «SmartGraph» и комфортной работы с ним необходимо соответствие минимальным системным требованиям:

* процессор 1700 МГц или выше;
* объем оперативной памяти не менее 1 ГБ;
* свободное место на диске не менее 3,54 МБ;
* операционная система Windows 7 и выше.

## Установка

На установочном диске находится установочный файл с программой. После открытия данного файла на экране появляется окно, представленное на Рис. 4.1. В нем пользователь может продолжить установку, нажав на кнопку «Далее», или отменить установку, нажав на кнопку «Отмена».

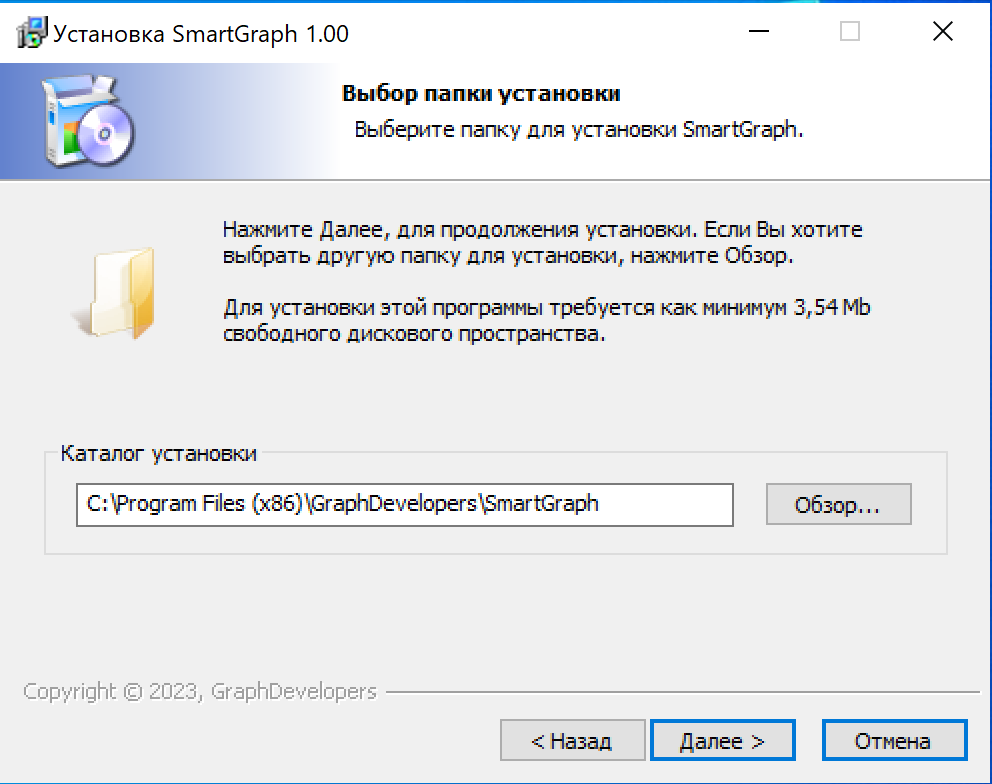


Рисунок 4.1 – Установка (этап 1)

На следующем этапе установки на экране открывается окно, представленное на Рис 4.2. Здесь пользователю предлагается выбрать расположение места папки установки. Также здесь отображается необходимое количество свободного дискового пространства.

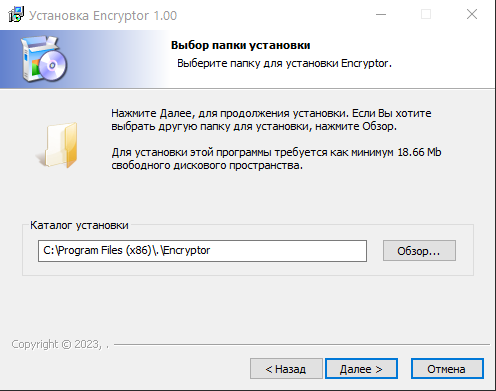


Рисунок 4.2 – Установка (этап 2)

После этого подготовка к установке завершается, и пользователю отображается предложение начать установку. Внешний вид окна представлен на Рис. 4.3.

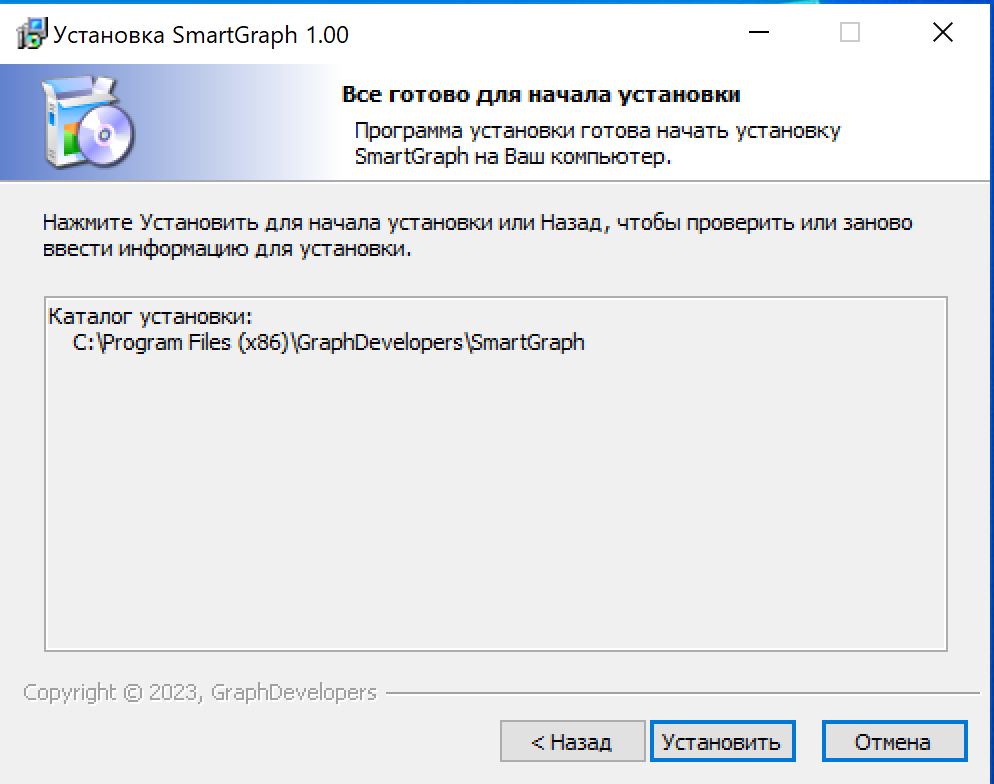


Рисунок 4.3 – Установка (этап 3)

Об успешной установке программного средства пользователя информирует окно, представленное на Рис. 4.4.

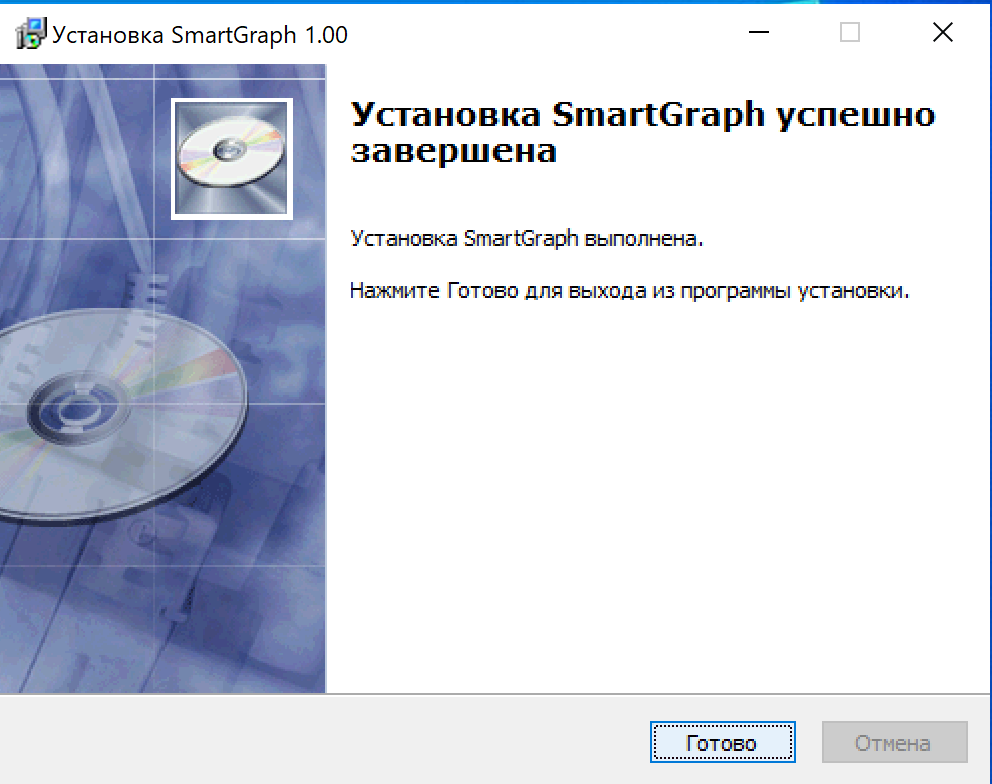


Рисунок 4.4 – Установка (этап 4)

## Руководство по работе с программным средством

После установке и запуска программного средства «SmartGraph» на экране появляется главное окно, представленное на Рис. 4.5

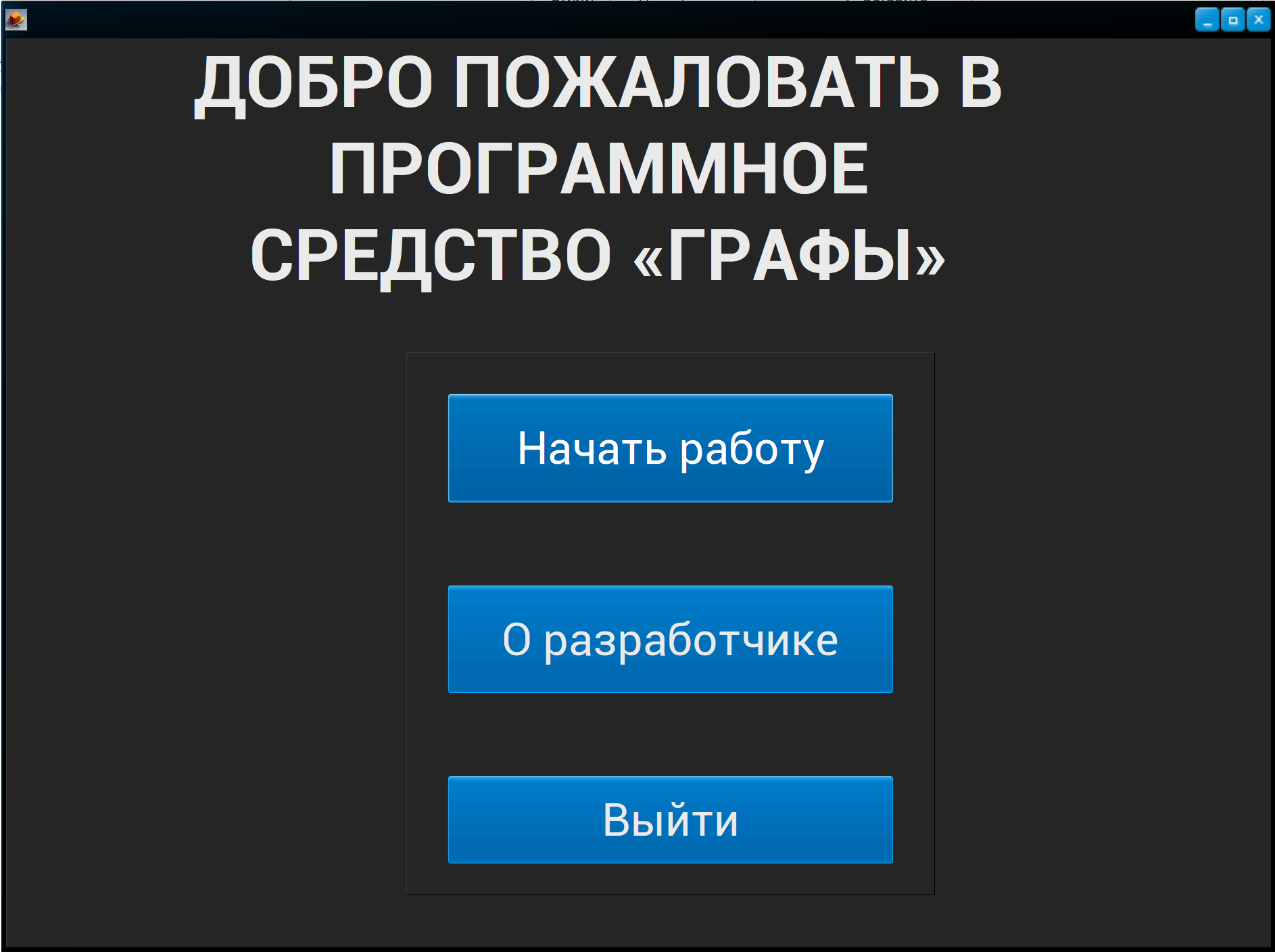


Рисунок 4.5 – Главное окно программного средства

Для того, чтобы перейти к работе с графами, необходимо нажать на кнопку «Начать работу». После чего откроется окно, представленное на Рис. 4.6.

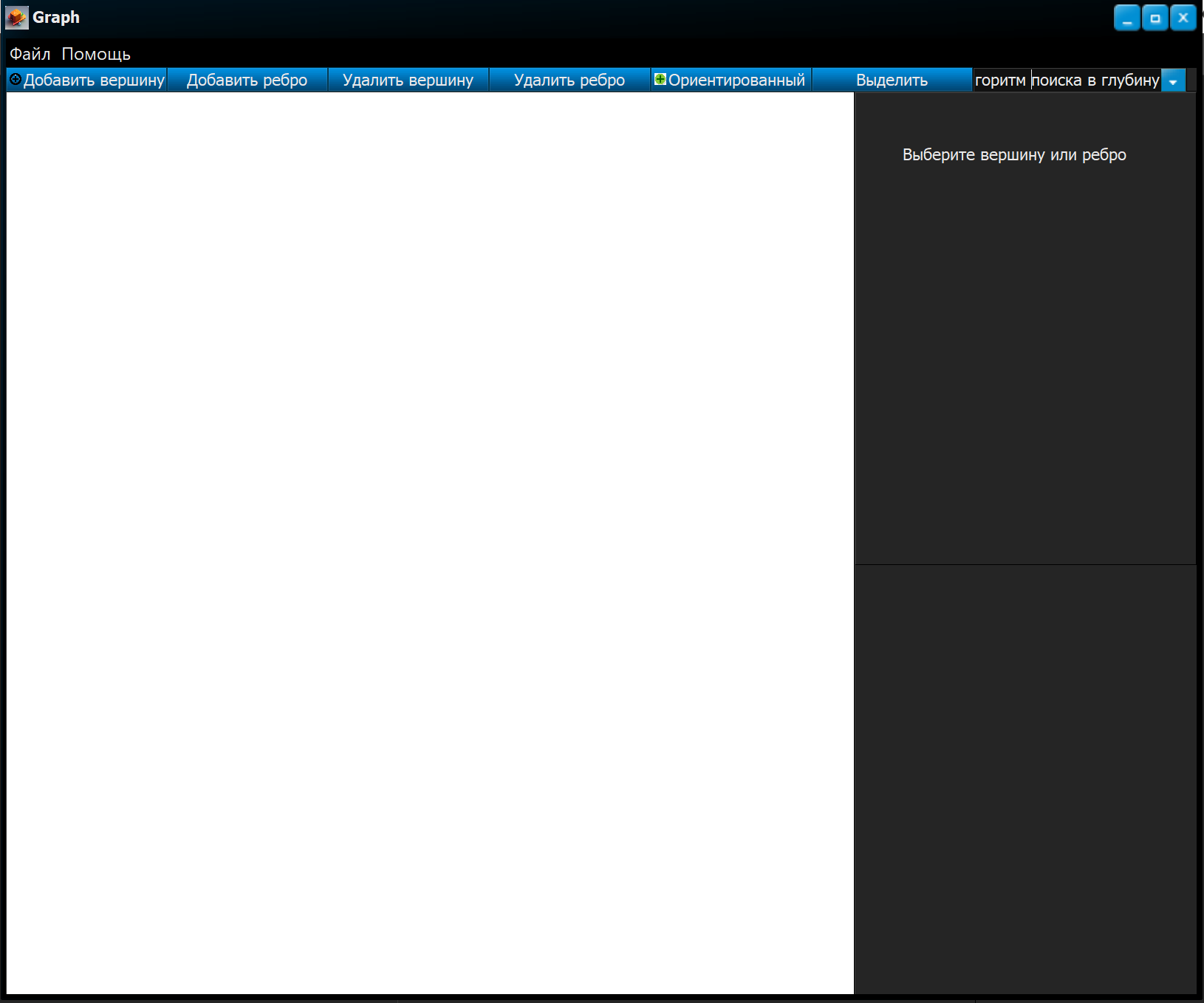


Рисунок 4.6 – Окно работы с графами

Окно работы с графами представляет собой холст, на котором можно рисовать граф, используя инструменты из верхней части окна.

Описание работы с инструментами:

* добавить вершину. При нажатии на кнопку «Добавление вершины», активируется режим добавления вершины. В режиме добавления вершины при нажатии левой клавиши мыши на холсте будут появляться вершины с центром в точке нажатия мыши;
* добавить ребро. Для добавления ребра между двумя вершинами необходимо предварительно выделить две вершины, выбрать тип ребра (ориентированное/неориентированное) и нажать на кнопку «Добавить ребро»;
* удалить вершину. Работает в двух режимах. Первый, при нажатии на кнопку «Удалить вершину» переходит в режим удаления вершины. При нажатии в этом режиме на вершину, нарисованную на холсте, произойдёт удаление этой вершины. Второй, если выделено несколько вершин или одна вершина, при нажатии на кнопку «Удалить вершину» произойдёт удаление всех выделенных вершин;
* удалить ребро. Для удаления ребра необходимо выделить две вершины, соединённые ребром и нажать клавишу «Удалить ребро»;
* ориентированный. При нажатии на кнопку «Ориентированный» произойдёт изменение иконки кнопки. Знак «-» значит, что граф неориентированный, «+» - ориентированный;
* выделить. При нажатии на кнопку «Выделить» происходит активация режима выделения. При нажатии левой клавишей на вершину происходит её выделение/снятие выделения, с зажатой клавишей Ctrl можно выделить несколько вершин;
* выпадающий список алгоритмов. Для нахождения пути между двумя вершинами необходимо выделить две вершины и выбрать алгоритм поиска пути.

При выделении одной вершины появляется информация о выделенной вершине с возможностью её редактирования. Область с информацией о вершине представлена на Рис. 4.7.

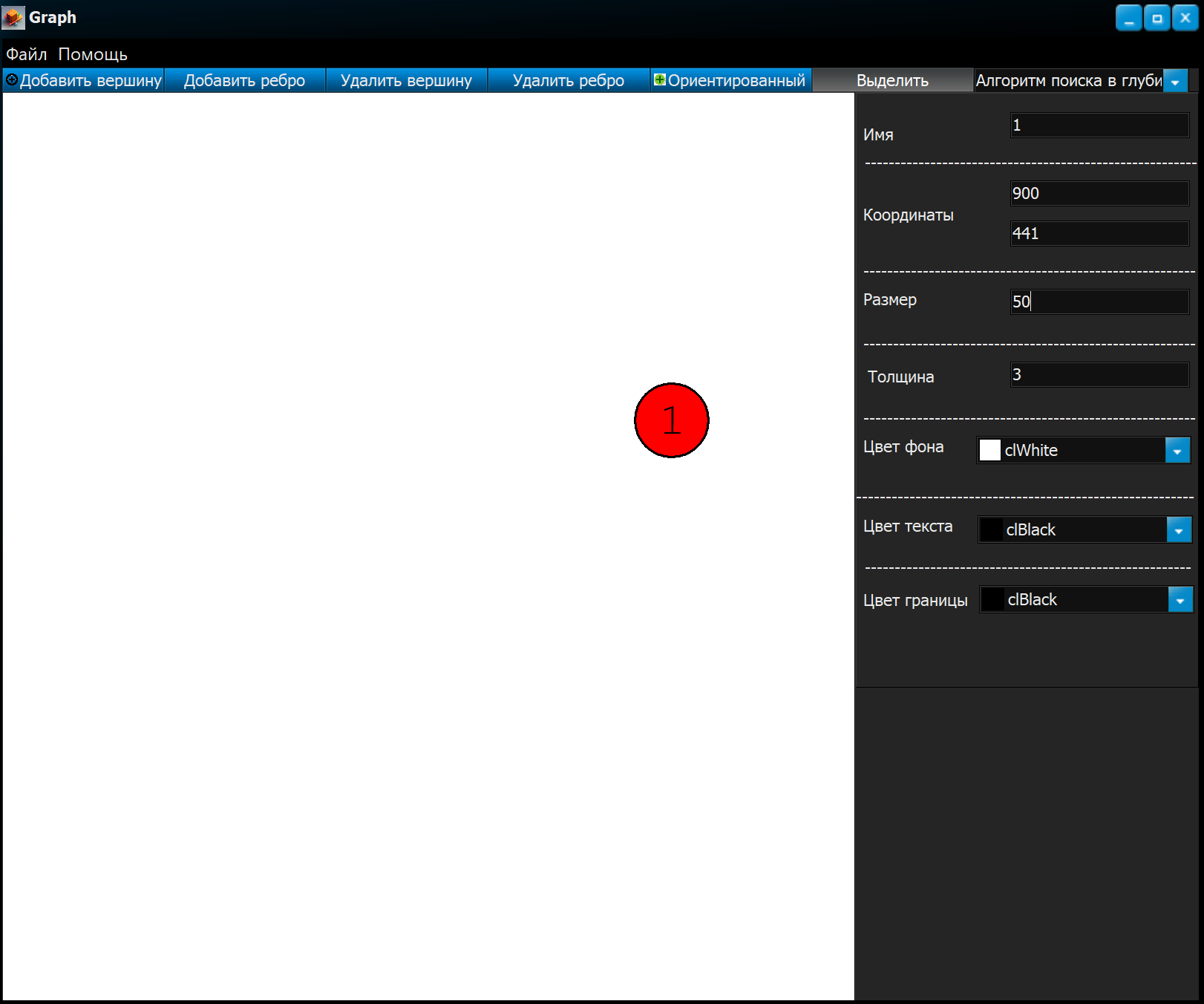


Рисунок 4.7 – Панель с информацией о вершине

При выделении двух вершин, соединённых ребром на боковой панели появляется информация о выделенном ребре с возможностью её редактирования. Область с информацией о вершине представлена на Рис. 4.8.

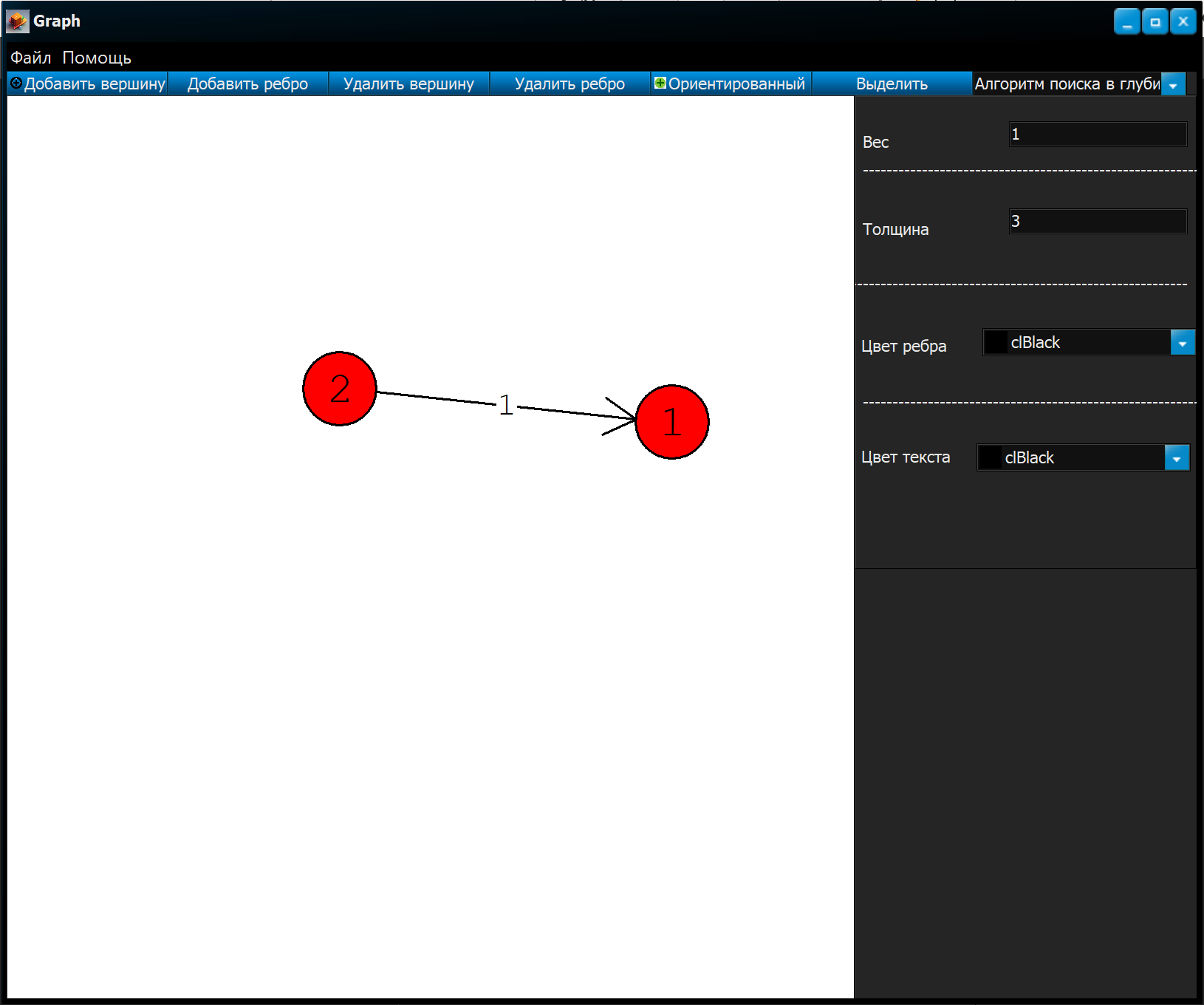


Рисунок 4.8– Панель с информацией о ребре

Для того чтобы перейти в режим работы с найденными путями необходимо выделить две вершины и выбрать предложенный алгоритм сортировки. Нажать на появившуюся кнопку «Просмотреть пути». В открывшемся окне появится область с найденными путями. Каждый путь содержит порядок обхода вершин и длину. Боковая панель содержит две кнопки для сортировки путей по возрастанию/убыванию длины пути. Для отображения пути необходимо выбрать один из найденных путей. Путь отобразится в окне работы с графом. Область с найденными путями представлена на Рис. 4.9.

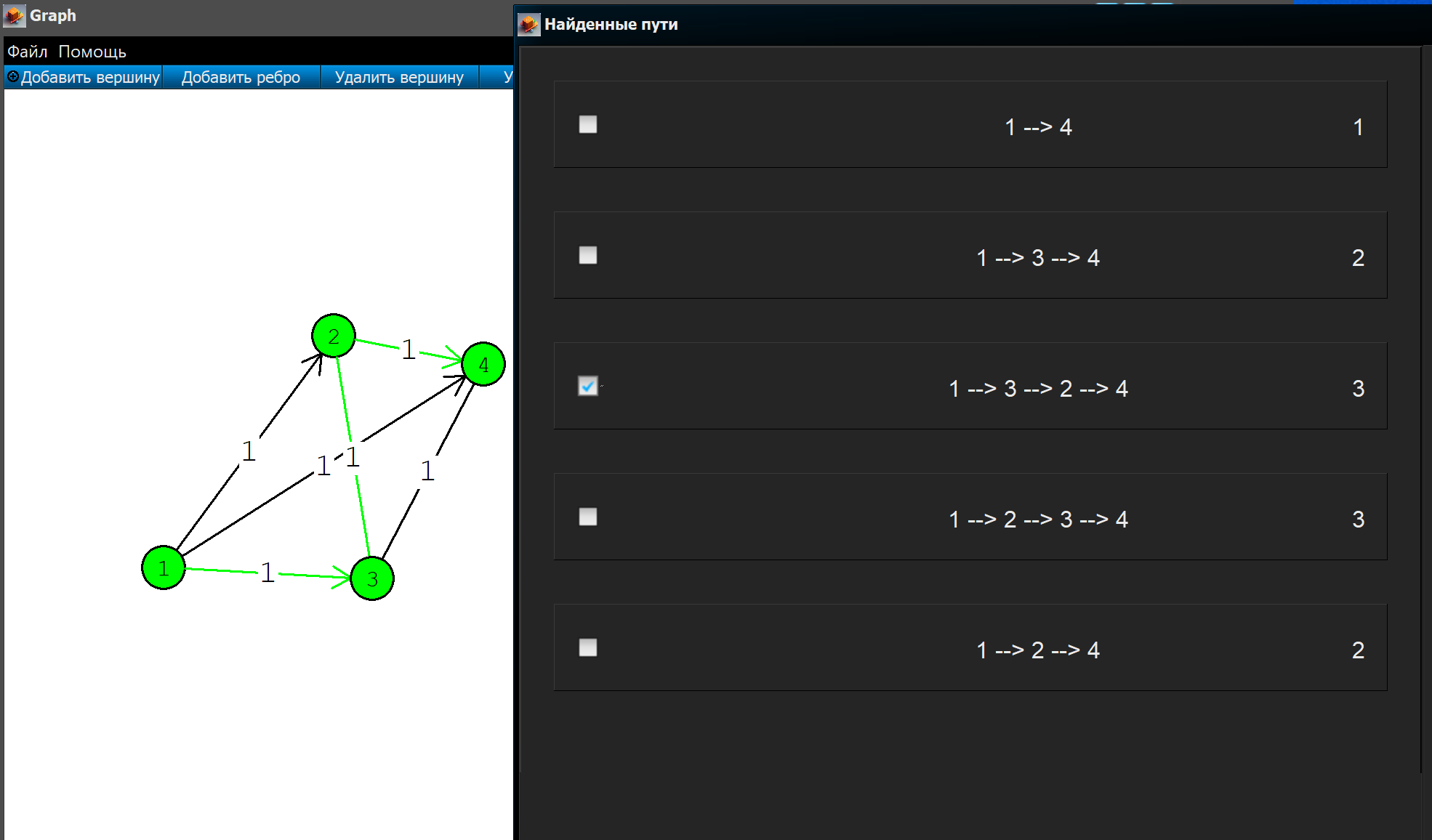


Рисунок 4.9– Отображение выбранного пути

Заключение

В данной курсовой работе было разработано программное средство для решения задач на графах с использованием алгоритма Дейкстры. Целью данного программного средства было предоставить пользователю инструмент, позволяющий эффективно решать задачи, связанные с анализом и поиском кратчайших путей в графах.

В ходе работы был реализован алгоритм Дейкстры, который является одним из наиболее эффективных алгоритмов для нахождения кратчайших путей в графах. Алгоритм позволяет определить кратчайший путь от одной вершины до всех остальных вершин во взвешенном графе. Программное средство было разработано с использованием языка программирования Delphi. Оно предоставляет пользователю удобный интерфейс, позволяющий вводить графическое представление графа, задавать начальную вершину и выполнять расчет кратчайших путей с помощью алгоритма Дейкстры, алгоритма поиска в ширину и глубину.

Основными возможностями программного средства являются:

* ввод и отображение графического представления графа;
* задание начальной вершины для поиска кратчайших путей;
* вычисление кратчайших путей с использованием алгоритма Дейкстры;
* отображение результатов расчета, включая кратчайшие пути и их веса.

В результате тестирования программного средства была подтверждена его функциональность и корректность работы. Оно успешно выполняет решение задач на графах, обеспечивая точные и быстрые результаты.

В заключение можно сказать, что разработанное программное средство представляет собой полезный инструмент для решения задач, связанных с анализом и поиском кратчайших путей в графах. Оно может быть использовано в различных областях, таких как транспортное планирование, логистика, маршрутизация сетей и другие, где требуется эффективное использование графовых структур.

Дальнейшее развитие программного средства может включать расширение функциональности, добавление поддержки других алгоритмов для работы с графами, оптимизацию производительности и улучшение пользовательского интерфейса.

В целом, данная курсовая работа позволила ознакомиться с проблематикой решения задач на графах с использованием алгоритма Дейкстры, а также разработать полезное программное средство, способствующее автоматизации и упрощению процесса решения таких задач.

Список использованной литературы

[1] Кнут Д.Э. Искусство программирования: Учеб. пособие. Т. 1. Основные алгоритмы. – М.: Вильямс, 2000. – 722 с.: ил.

[2] Вирт, Н. Алгоритмы и структуры данных. – М.: Мир, 1989. – 360 с.: ил.

[3] Серебряная, Л.В. Электронный учебно-методический комплекс по дисциплине «Структуры и алгоритмы обработки данных» [Электронный ресурс]. Режим доступа: <https://erud.bsuir.by/>. – Дата доступа: 15.03.2023.

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

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

[6] Глухова, Л. А. Электронный учебно-методический комплекс по дисциплине «Основы алгоритмизации и программирования». Часть 1. [Электронный ресурс]. Режим доступа: <https://erud.bsuir.by/>. – Дата доступа: 15.03.2023.

Приложение A

**Исходный код программы (модуль MyTypes)**

unit MyTypes;

{

Unit that include all nedeed types

and the most popular subprograms

}

interface

uses

Vcl.Graphics;

type

// information about vertex

PVertex = ^Tvertex;

TVertex = record

name:Integer;

x:integer;

y:integer;

backcolor:TColor;

bordercolor:TColor;

textcolor:TColor;

size:integer;

bordersize:Integer;

visited: Boolean;

selected:Boolean;

end;

// information about selected vertex

PSelect = ^TSelect;

TSelect = record

Vertex:PVertex;

Color:TColor;

Next:PSelect;

Front:PSelect;

end;

// information about edge

PEdge = ^TEdge;

TEdge = record

vert1:integer;

vert2:integer;

weight:integer;

pencolor:TColor;

textcolor:TColor;

penwidth:Integer;

isUsed:Boolean;

Oriented:Boolean;

end;

// all connected vetices for one edge

PNode = ^TNode;

TNode = record

vertex: pvertex;

edge: pedge;

next: PNode;

end;

// all connected vetices for each edge

PList = ^TList;

TList = record

startlist:PNode;

currentvertex:PNode;

vertexinf: pvertex;

front: plist;

rear: plist;

next: plist;

end;

// structure stack for creating way

PStack = ^TStack;

TStack = record

vertex:PVertex;

distance:PEdge;

next:PStack;

end;

// queue for BFS algorithm

PQueue=^TQueue;

TQueue = record

vertex:PVertex;

distance:PEdge;

prevvertex:PStack;

next,front,rear:PQueue;

end;

// array of stack (all ways)

TStackArr = array of pstack;

// queue for Dejcstra algorithm

PQueuePr=^TQueuePr;

TQueuePr = record

vertex:PVertex;

priority:integer; // dijkstra

visited:Boolean;

previous:TStackArr;

next,front,rear:PQueuePr;

end;

// vertex in way

TWayInf = record

vertex:PVertex;

distance:PEdge;

end;

// one way

TWay=record

way:array of TWayInf;

len:Integer;

end;

// all ways

TWays=array of TWay;

// structure to pop vertex from queue

TPopQueue = record

vertex:Pvertex;

weight:PEdge;

way:PStack;

end;

// information about selected way

PSelectWay = ^TSelectWay;

TselectWay = record

vertex:PVertex;

Edge:PEdge;

Vcolor:TColor;

Ecolor:TColor;

Next:PSelectWay;

end;

// structure to pop vertex from priority queue

TPopQueuePr = record

vertex:Pvertex;

distance:Integer;

end;

// subprogram that finds address of vertex in list

procedure FindVertexInList(const SearchingName: Integer;

var ListPointer:PList);

// subprogram that create list of ways from stack;

Function WayFromStack(Stack:PStack):TWay;

implementation

procedure FindVertexInList(const SearchingName: Integer;

var ListPointer:PList);

begin

ListPointer:=ListPointer^.front^.next;

while ListPointer^.vertexinf^.name<>SearchingName do

begin

ListPointer:=ListPointer^.next;

end;

end;

Function WayFromStack(Stack:PStack):TWay;

var

tmprecord:TWayinf;

begin

SetLength(Result.way,0);

result.len:=0;

while stack<>nil do

begin

tmprecord.vertex:=Stack^.vertex;

tmprecord.distance:=Stack^.distance;

result.way:=result.way+[tmprecord];

if tmprecord.distance<>nil then

result.len:=result.len+tmprecord.distance^.weight;

Stack:=Stack^.next;

end;

end;

end.

Приложение Б

**Исходный код программы (модуль CreateGraph)**

unit CreateGraph;

interface

uses

MyTypes;

// subprogram that checks are two vertex connected;

function ConnectedVertex(const vert1:integer;const

vert2:integer;var mainlist1:PList):Boolean;

// subprogram that add a new vertex in list;

procedure AddVertex(var pnewvertex: plist;const

vertex:TVertex);

// subprogram that add all connected vertices for the

// current vertex;

procedure addsemivertex(var source:PList; const

destination: PList; const SemiEdge:TEdge;var

newedge:pedge;const Directed:Boolean);

// subprogram that add edge in list;

procedure AddEdge(const fvertex:integer; const

svertex:integer; const Edge:TEdge;var

mainlist:Plist; const oriented:Boolean);

// subprogram that delete connected vertex for current

// vertex;

procedure deleteSemiVertex(var source:PList; const

destination: PList);

// subprogram that delete edge in list;

procedure DeleteEdge(const fvertex:integer; const

svertex:integer;var mainlist:Plist);

// subprogram that delete vertex in list;

procedure DeleteVertex(const vertexname:integer;var

mainlist:Plist);

// subprogram that creates a head of list;

procedure HeadCreate(var newvert:Plist);

implementation

function ConnectedVertex(const vert1:integer;const

vert2:integer;var mainlist1:PList):Boolean;

var

point1,point2:PList;

tmp:PNode;

begin

FindVertexInList(vert1,mainlist1);

point1:=mainlist1;

FindVertexInList(vert2,mainlist1);

point2:=mainlist1;

result:=false;

tmp:=point1^.startlist;

// we need to check for each vertex because

// they can be connected using oriented edge;

while (tmp<>nil) and not result do

begin

if tmp.vertex=point2^.vertexinf then

result:=true

else

tmp:=tmp^.next;

end;

if not result then

begin

tmp:=point2^.startlist;

while (tmp<>nil) and not result do

begin

if tmp.vertex=point1^.vertexinf then

result:=true

else

tmp:=tmp^.next;

end;

end;

end;

procedure HeadCreate(var newvert:Plist);

begin

new(newvert);

newvert^.front:=newvert;

newvert^.rear:=newvert;

newvert^.startlist:=nil;

newvert^.vertexinf:=nil;

newvert^.next:=nil

end;

procedure AddVertex(var pnewvertex: plist;const

vertex:TVertex);

var

pvertexinf:pvertex;

oldvertex:plist;

begin

pnewvertex:=pnewvertex^.front;

oldvertex:=pnewvertex;

new(pnewvertex);

pnewvertex^.front:=oldvertex;

oldvertex^.rear^.next:=pnewvertex;

pnewvertex^.startlist:=nil;

new(pvertexinf);

pnewvertex^.vertexinf:=pvertexinf;

pvertexinf^:=Vertex;

pnewvertex^.rear:=pnewvertex;

pnewvertex^.next:=nil;

// update pointer to last element;

while oldvertex<>nil do

begin

oldvertex^.rear:=pnewvertex;

oldvertex:=oldvertex^.next;

end;

end;

procedure addsemivertex(var source:PList; const

destination: PList; const SemiEdge:TEdge;var

newedge:pedge;const Directed:Boolean);

var

lastvertex,newvertex:pnode;

begin

lastvertex:=source^.startlist;

if lastvertex=nil then

begin

new(lastvertex);

source^.startlist:=lastvertex;

source^.currentvertex:=lastvertex;

end

else

begin

while lastvertex^.next<>nil do

begin

lastvertex:=lastvertex^.next;

end;

new(newvertex);

lastvertex^.next:=newvertex;

lastvertex:=newvertex;

end;

lastvertex^.vertex:=destination^.vertexinf;

lastvertex^.next:=nil;

lastvertex^.edge:=newedge;

lastvertex^.edge^:=Semiedge;

lastvertex^.edge.vert1:=source^.vertexinf^.name;

lastvertex^.edge.vert2:=destination^.vertexinf^.name;

lastvertex^.edge.oriented:=directed;

end;

procedure AddEdge(const fvertex:integer; const

svertex:integer; const Edge:TEdge;var

mainlist:Plist; const oriented:Boolean);

var

point1,point2:PList;

edgepointer:PEdge;

begin

FindVertexInList(fvertex,mainlist);

point1:=mainlist;

FindVertexInList(svertex,mainlist);

point2:=mainlist;

new(edgepointer);

// add connected vertex in list for each vertex;

addsemivertex(point1,point2,Edge,edgepointer,oriented);

if not oriented then

begin

addsemivertex(point2,point1,Edge,edgepointer,

oriented);

end;

end;

procedure deleteSemiVertex(var source:PList; const

destination: PList);

var

tmp:PNode;

flag:Boolean;

begin

tmp:=source^.startlist;

flag:=True;

if tmp<>nil then

begin

while tmp^.next<>nil do

begin

if tmp^.next^.vertex=destination^.vertexinf then

begin

source^.currentvertex:=tmp^.next^.next;

dispose(tmp^.next);

tmp.next:=source^.currentvertex;

Flag:=False;

end

else

tmp:=tmp^.next;

end;

if source^.startlist^.vertex=destination^.vertexinf

then

begin

source^.currentvertex:=source^.startlist^.next;

dispose(source^.startlist);

source^.startlist:=source^.currentvertex;

end;

end;

end;

procedure DeleteEdge(const fvertex:integer; const

svertex:integer;var mainlist:Plist);

var

point1,point2:PList;

begin

FindVertexInList(fvertex,mainlist);

point1:=mainlist;

FindVertexInList(svertex,mainlist);

point2:=mainlist;

If ConnectedVertex(fvertex ,svertex, mainlist) then

begin

deletesemivertex(point1,point2);

deletesemivertex(point2,point1);

end;

end;

procedure DeleteVertex(const vertexname:integer;var

mainlist:Plist);

var

vertexcopy,DTmp:PList;

tmp:PNode;

begin

FindVertexInList(vertexname,mainlist);

vertexcopy:=mainlist;

mainlist:=mainlist^.front;

// delete vertex from list and from all nodes;

while mainlist^.next<>nil do

begin

if mainlist^.next=vertexcopy then

begin

tmp:=mainlist^.next^.startlist;

while tmp <> nil do

begin

mainlist^.next^.currentvertex:=tmp^.next;

dispose(tmp);

tmp:=mainlist^.next^.currentvertex;

end;

if mainlist^.next^.next=nil then

begin

Dtmp:=mainlist^.front;

while dtmp<>nil do

begin

dtmp^.rear:=mainlist;

dtmp:=dtmp^.next;

end;

end;

mainlist^.next^.currentvertex:=nil;

mainlist^.next^.startlist:=nil;

DTmp:=mainlist^.next^.next;

dispose(mainlist^.next);

mainlist^.next:=DTmp;

end

else

begin

deletesemivertex(mainlist^.next,vertexcopy);

mainlist:=mainlist^.next;

end;

end;

deletesemivertex(mainlist,vertexcopy);

end;

end.

Приложение В

**Исходный код программы (модуль Properties)**

unit Properties;

interface

Uses

MyTypes,VCL.Graphics;

// subprogram that sets default vertex info;

procedure SetDefault(var VertToDefualt:PList);

// subprogram that set start vertex properties;

function SetVertProperties(const name:integer;const

x:integer;const y:Integer):TVertex;

// subprogram that set start edge properties;

Function SetEdgeProperties():TEdge;

implementation

procedure SetDefault(var VertToDefualt:PList);

begin

VertToDefualt^.currentvertex:=VertToDefualt^.startlist;

VertToDefualt^.vertexinf^.visited:=False;

end;

function SetVertProperties(const name:integer;const

x:integer;const y:Integer):TVertex;

begin

result.name:=name;

result.x:=x;

result.y:=y;

result.backcolor:=clWhite;

result.bordercolor:=clBlack;

result.textcolor:=clBlack;

result.size:=30;

result.bordersize:=3;

result.visited:=False;

result.selected:=False;

end;

Function SetEdgeProperties():TEdge;

begin

result.weight:=1;

result.pencolor:=clBlack;

result.textcolor:=clBlack;

result.penwidth:=3;

result.isused:=false;

end;

end.

Приложение Г

**Исходный код программы (UFile)**

unit UFile;

interface

Uses

MyTypes, CreateGraph,ComObj,Vcl.Graphics, Variants;

// subprogram that save grapg in typed file;

procedure SaveListToFile(var Graph: PList; const EfileName:

string;const VFileName: string);

// subprogram that create graph from typed file;

Procedure CreateFromFile(var Graph:PList; const

EFileName:string; const VFileName: string);

// subprogram that save grapg in Excel file;

procedure SaveGraphToExcel(graph: PList;var way:String);

// subprogram that create graph from Excel file;

procedure LoadGraphFromExcel(var filename: string; var

Graph:PList);

implementation

procedure SaveListToFile(var Graph: PList; const EfileName:

string;const VFileName: string);

var

FileEdge: File of TEdge;

FileVert: File of TVertex;

CurVert:PList;

CurSemiVert:PNode;

begin

try

AssignFile(FileVert, VFileName);

Rewrite(FIleVert);

AssignFile(FileEdge, EfileName);

Rewrite(FIleEdge);

CurVert:=Graph^.front^.next;

// write each vertex in typed file

while CurVert<>nil do

begin

Write(FileVert,CurVert^.vertexinf^);

CurSemiVert:=CurVert^.startlist;

// write each edge in typed file;

while cursemivert<>nil do

begin

if not cursemivert^.edge^.isused then

begin

write(FileEdge,CurSemiVert^.edge^);

if not cursemivert^.edge^.oriented then

cursemivert^.edge^.isused:=True;

end

else

cursemivert^.edge^.isused:=False;

cursemivert:=cursemivert^.next;

end;

CurVert:=CurVert^.next;

end;

finally

CloseFile(FileVert);

CloseFile(FileEdge);

end;

end;

Procedure CreateFromFile(var Graph:PList; const

EFileName:string; const VFileName: string);

var

FileEdge: File of TEdge;

FileVert: File of TVertex;

Node:TVertex;

arc:TEdge;

begin

try

AssignFile(FileVert, VFileName);

Reset(FIleVert);

// add all vertices from file;

repeat

Read(FileVert,Node);

AddVertex(Graph,Node);

until EOF(FileVert);

AssignFile(FileEdge, EfileName);

Reset(FIleEdge);

// add all edges from file;

while not EOF(FileEdge) do

begin

Read(FileEdge,Arc);

AddEdge(Arc.vert1,Arc.vert2,Arc,Graph,Arc.

Oriented);

end;

finally

CloseFile(FileVert);

CloseFile(FileEdge);

end;

end;

procedure SaveGraphToExcel(graph: PList;var way:String);

var

ExcelApp: Variant;

Workbook: Variant;

Node:PNode;

SheetVertices, SheetEdges: Variant;

vertexRow, edgeRow: Integer;

begin

// create excel book with two sheets;

ExcelApp := CreateOleObject('Excel.Application');

ExcelApp.Visible := False;

Workbook := ExcelApp.Workbooks.Add;

SheetVertices := Workbook.Sheets.Add;

SheetEdges := Workbook.Sheets.Add;

// name vertex sheet;

SheetVertices.Name := 'Vertices';

SheetVertices.Cells[1, 1].Value := 'Name';

SheetVertices.Cells[1, 2].Value := 'X';

SheetVertices.Cells[1, 3].Value := 'Y';

SheetVertices.Cells[1, 4].Value := 'BackColor';

SheetVertices.Cells[1, 5].Value := 'BorderColor';

SheetVertices.Cells[1, 6].Value := 'TextColor';

SheetVertices.Cells[1, 7].Value := 'Size';

SheetVertices.Cells[1, 8].Value := 'BorderSize';

// name edge sheet;

SheetEdges.Name := 'Edges';

SheetEdges.Cells[1, 1].Value := 'Vertex1';

SheetEdges.Cells[1, 2].Value := 'Vertex2';

SheetEdges.Cells[1, 3].Value := 'Weight';

SheetEdges.Cells[1, 4].Value := 'PenColor';

SheetEdges.Cells[1, 5].Value := 'TextColor';

SheetEdges.Cells[1, 6].Value := 'PenWidth';

SheetEdges.Cells[1, 7].Value := 'Oriented';

// fill vertex sheet;

graph := graph^.front^.next;

vertexRow := 2;

edgeRow := 2;

while graph <> nil do

begin

SheetVertices.Cells[vertexRow, 1].Value :=

graph^.vertexinf^.name;

SheetVertices.Cells[vertexRow, 2].Value :=

graph^.vertexinf^.x;

SheetVertices.Cells[vertexRow, 3].Value :=

graph^.vertexinf^.y;

SheetVertices.Cells[vertexRow, 4].Value :=

ColorToString(graph^.vertexinf^.backcolor);

SheetVertices.Cells[vertexRow, 5].Value :=

ColorToString(graph^.vertexinf^.bordercolor);

SheetVertices.Cells[vertexRow, 6].Value :=

ColorToString(graph^.vertexinf^.textcolor);

SheetVertices.Cells[vertexRow, 7].Value :=

graph^.vertexinf^.size;

SheetVertices.Cells[vertexRow, 8].Value :=

graph^.vertexinf^.bordersize;

node:=graph^.startlist;

// fill edge sheet;

while node<>nil do

begin

if not node^.edge^.isused then

begin

SheetEdges.Cells[edgeRow, 1].Value :=

node^.edge^.vert1;

SheetEdges.Cells[edgeRow, 2].Value :=

node^.edge^.vert2;

SheetEdges.Cells[edgeRow, 3].Value :=

node^.edge^.weight;

SheetEdges.Cells[edgeRow, 4].Value :=

ColorToString(node^.edge^.pencolor);

SheetEdges.Cells[edgeRow, 5].Value :=

ColorToString(node^.edge^.textcolor);

SheetEdges.Cells[edgeRow, 6].Value :=

node^.edge^.penwidth;

SheetEdges.Cells[edgeRow, 7].Value :=

node^.edge^.Oriented;

if not node^.edge^.oriented then

node^.edge^.isused:=True;

Inc(edgeRow);

end

else

node^.edge^.isused:=False;

node:=node^.next;

end;

graph := graph^.next;

Inc(vertexRow);

end;

// save Excel file and close it;

Workbook.SaveAs(Way);

Workbook.Close;

ExcelApp.Quit;

end;

procedure LoadGraphFromExcel(var filename: string; var

Graph: PList);

var

ExcelApp: Variant;

Workbook: Variant;

SheetVertices, SheetEdges: Variant;

Row: Integer;

Node: TVertex;

Arc: TEdge;

begin

// create Excel file

ExcelApp := CreateOleObject('Excel.Application');

try

ExcelApp.Visible := False;

ExcelApp.Workbooks.Open(filename);

Workbook := ExcelApp.ActiveWorkbook;

// get vertex sheet;

SheetVertices := Workbook.Worksheets['Vertices'];

for Row := 2 to SheetVertices.UsedRange.Rows.Count do

begin

Node.name := SheetVertices.Cells[Row, 1].Value;

Node.x := SheetVertices.Cells[Row, 2].Value;

Node.y := SheetVertices.Cells[Row, 3].Value;

Node.backcolor :=

StringToColor(SheetVertices.Cells[Row, 4].Value);

Node.bordercolor :=

StringToColor(SheetVertices.Cells[Row, 5].Value);

Node.textcolor :=

StringToColor(SheetVertices.Cells[Row, 6].Value);

Node.size := SheetVertices.Cells[Row, 7].Value;

Node.bordersize := SheetVertices.Cells[Row,

8].Value;

AddVertex(Graph, Node);

end;

// get edge sheet;

SheetEdges := Workbook.Worksheets['Edges'];

for Row := 2 to SheetEdges.UsedRange.Rows.Count do

begin

Arc.vert1 := SheetEdges.Cells[Row, 1].Value;

Arc.vert2 := SheetEdges.Cells[Row, 2].Value;

Arc.weight := SheetEdges.Cells[Row, 3].Value;

Arc.pencolor := StringToColor(SheetEdges.Cells

[Row, 4].Value);

Arc.textcolor := StringToColor(SheetEdges.Cells

[Row, 5].Value);

Arc.penwidth := SheetEdges.Cells[Row, 6].Value;

Arc.oriented := SheetEdges.Cells[Row, 7].Value;

AddEdge(Arc.vert1, Arc.vert2, Arc, Graph,

Arc.oriented);

end;

finally

Workbook.Close(False);

ExcelApp.Quit;

ExcelApp := Unassigned;

end;

end;

end.

Приложение Д

**Исходный код программы (модуль UDFS)**

unit UDFS;

interface

uses

MyTypes;

// Subprogram that add vertex in stack;

Function StackPush(var NodeList:PList;var

NewElP:PStack):Boolean;

// Subprogram that delete vertex from stack;

function StackPop(var OldElP:PStack; var

ChgIncVert:Plist):Boolean;

// Subprogram that finds all ways between two points using

// DFS algorithm;

Function DFS(var MainLIst:PList;const start:Integer;const

Finish:Integer):TWays;

implementation

Function StackPush(var NodeList:PList;var

NewElP:PStack):Boolean;

var

newnode:PStack;

begin

if nodelist^.currentvertex<>nil then

begin

// find vertex to push;

while (NodeList^.currentvertex^.vertex^.visited) and

(NodeList^.currentvertex^.next<>nil) do

begin

NodeList^.currentvertex:=

NodeList^.currentvertex^.next;

end;

if not NodeList^.currentvertex^.vertex^.visited then

begin

new(newnode);

newnode^.vertex:=NodeList^.currentvertex^.vertex;

newnode^.distance:=NodeList^.currentvertex^.edge;

NodeList^.currentvertex^.vertex^.visited:=True;

nodelist^.currentvertex:=

nodelist^.currentvertex^.next;

newnode.next:=NewElP;

NewElP:=newnode;

result:=True;

end

else

result:=False;

end

else

result:=False;

end;

function StackPop(var OldElP:PStack; var

ChgIncVert:Plist):Boolean;

var

DTmp:PStack;

begin

// find vertex to pop;

if OldElP<>nil then

begin

OldElP^.vertex^.visited:=False;

FindvertexInLIst(OldElP^.vertex^.name,ChgIncVert);

ChgIncVert^.currentvertex:=ChgIncVert^.startlist;

DTmp:=OldElP^.next;

Dispose(OldElp);

OldElP:=DTmp;

end;

if OldElP<>nil then

result:=True

else

Result:=False;

end;

Function DFS(var MainLIst:PList;const start:Integer;const

Finish:Integer):TWays;

var

startp:PStack;

source,destination:PList;

tmpvertex:PList;

tmpwayinf:TWayInf;

tmpway:Tway;

res:Boolean;

begin

FindVertexInList(start,mainlist);

source:=mainlist;

FindVertexInList(finish,mainlist);

destination:=mainlist;

new(startp);

startp^.vertex:=source^.vertexinf;

startp^.distance:=nil;

startp^.next:=nil;

source^.vertexinf^.visited:=True;

tmpvertex:=source;

while startp <> nil do

begin

res:=StackPush(tmpvertex,startp);

while res do

begin

// find way;

if startp^.vertex=destination^.vertexinf then

begin

res:=False;

result:=result+[wayfromstack(startp)];

setlength(tmpway.way,0);

end

else

begin

// else again push

FindVertexInList(startp^.vertex^.name,mainlist);

tmpvertex:=mainlist;

res:=StackPush(tmpvertex,startp);

end;

end;

// if cant't push try to pop;

if StackPop(startp,mainlist) then

begin

FindVertexInList(startp^.vertex^.name,mainlist);

tmpvertex:=mainlist;

end

else

startp:=nil;

end;

end;

end.

Приложение Е

**Исходный код программы (модуль UBFS)**

unit UBFS;

interface

uses

MyTypes;

// subprogram that checks vertex in stack;

function NodeInStack(Stack:PStack;const

Node:PVertex):Boolean;

// subprogram that initialise queue;

Procedure QueueInit(var PQu:PQueue);

// subprogram that adds vertex in queue;

Procedure QueuePush (var newvertex:PNode;var PQu:PQueue);

// subprogram that delete vertex from queue;

Function QueuePop (var PQu:PQueue):TPopQueue;

// subprogram that finds all ways between two points using BFS algorithm;

Function BFS (var MainList:Plist; const start:integer;const

finish:integer):TWays;

implementation

function NodeInStack(Stack:PStack;const

Node:PVertex):Boolean;

begin

result:=false;

while stack<>nil do

begin

if stack^.vertex=Node then

begin

result:=True;

stack:=nil;

end

else

stack:=stack^.next;

end;

end;

Procedure QueueInit(var PQu:PQueue);

begin

new(PQu);

PQu^.front:=PQu;

PQu^.rear:=PQu;

end;

Procedure QueuePush (var newvertex:PNode;var PQu:PQueue);

var

tmp:PQueue;

begin

new(PQu^.rear^.next);

PQu^.rear:=PQu^.rear^.next;

PQu^.rear^.vertex:=newvertex^.vertex;

PQu^.rear^.distance:=newvertex^.edge;

PQu^.rear^.front:=PQu^.front;

PQu^.rear^.next:=nil;

tmp:=PQu^.rear;

PQu:=PQu^.front;

while PQu<>nil do

begin

PQu^.rear:=tmp;

PQu:=PQu^.next;

end;

PQu:=tmp^.front;

end;

Function QueuePop (var PQu:PQueue):TPopQueue;

var

DTmp:Pqueue;

begin

result.vertex:=PQu^.front^.next.vertex;

result.weight:=PQu^.front^.next.distance;

result.way:=PQu^.front^.next.prevvertex;

PQu:=PQu^.front;

Dtmp:=PQu^.next^.next;

dispose(PQu^.next);

PQu^.next:=DTmp;

if PQu^.front^.next=nil then

PQu^.rear:=PQu^.front;

end;

Function BFS (var MainList:Plist; const start:integer;const

finish:integer):TWays;

var

Queue:PQueue;

Source,Destination:PList;

TMPVertex:Plist;

FromQ:TPopQueue;

tmpwayinf:TWayInf;

newway:Tway;

Prevvertex,newVertex:PStack;

tmpway:Tway;

begin

queueinit(Queue);

FindvertexInList(start,MainList);

Source:=Mainlist;

FindvertexInList(finish,MainList);

Destination:=Mainlist;

new(Queue^.rear^.next);

Queue^.rear:=Queue^.rear^.next;

Queue^.rear^.vertex:=Source^.vertexinf;

Queue^.rear^.distance:=nil;

Queue^.rear^.prevvertex:=nil;

Queue^.rear^.next:=nil;

Queue^.rear^.front:=Queue^.front;

Queue^.rear^.rear:=Queue^.rear;

while Queue^.next<>nil do

begin

// pop vertex;

FromQ:=queuepop(queue);

if FromQ.vertex=destination^.vertexinf then

begin

// find way;

tmpwayinf.vertex:=fromQ.vertex;

tmpwayinf.distance:=fromQ.weight;

newway:=wayfromstack(fromQ.way);

newway.way:=[tmpwayinf]+newway.way;

newway.len:=newway.len+tmpwayinf.distance^.weight;

result:=result+[newway];

end

else

begin

findvertexinlist(fromQ.vertex^.name,mainlist);

tmpvertex:=mainlist;

tmpvertex^.currentvertex:=tmpvertex^.startlist;

while tmpvertex^.currentvertex<>nil do

begin

if not NodeInStack(fromQ.way,tmpvertex^.

currentvertex^.vertex) then

begin

// push a neubhour;

QueuePush(tmpvertex^.currentvertex,queue);

prevvertex:=FromQ.way;

new(newvertex);

newvertex^.vertex:=FromQ.vertex;

newvertex^.distance:=FromQ.weight;

// update ancestors list;

if Prevvertex=nil then

begin

Prevvertex:=newvertex;

newvertex^.next:=nil;

end

else

begin

newvertex^.next:=prevvertex;

end;

queue^.rear^.prevvertex:=newvertex;

end;

tmpvertex^.currentvertex:=

tmpvertex^.currentvertex^.next;

end;

end;

end;

end;

end.

Приложение Ж

**Исходный код программы (модуль Deicstra)**

unit Deikstra;

interface

Uses

MyTypes;

// subprogram that initialise priority queue;

Procedure PrQueueInit (var Queue:PQueuePr;const

list:PList);

// Subprogram that finds next vertex to pop;

Function QueueVisited (Queue:PQueuePr):TPopQueuePr;

// Subprogram that upfates distance for each connected

// vertex;

procedure UpdateDistance(var Queue:PQueuePr;var

vertfromqueue:PList; const dist: integer);

// Subprogram that finds all shortes ways between two points

// using Dejcstra algorithm

Function Dijcstra (var Graph:Plist; const

Start:Integer;const Finish:Integer):TWays;

implementation

Procedure PrQueueInit (var Queue:PQueuePr;const

list:PList);

var

TmpNode:PList;

Front:PQueuePr;

begin

new(Queue);

Queue^.front:=Queue;

Front:=Queue;

Queue^.rear:=Queue;

tmpNode:=List^.front^.next;

// add all vertices in queue;

While tmpnode<>nil do

begin

new(Queue^.rear^.next);

Queue^.rear:=Queue^.rear^.next;

Queue^.rear^.vertex:=tmpnode.vertexinf;

Queue^.rear^.priority:=MaxInt;

Queue^.rear^.visited:=False;

Queue^.rear^.front:=front;

tmpnode:=tmpnode^.next;

end;

Queue^.rear^.next:=nil;

end;

Function QueueVisited (Queue:PQueuePr):TPopQueuePr;

var

MinA:PQueuePr;

First:Boolean;

begin

Queue:= Queue^.front^.next;

First:=True;

// find vertex with smallest distance;

while Queue<>nil do

begin

if not Queue^.visited then

begin

If First then

begin

MinA:=Queue;

first:=false;

end

else

if Queue^.priority<minA^.priority then

minA:=Queue;

end;

Queue:=Queue^.next;

end;

minA^.visited:=True;

result.vertex:=minA^.vertex;

result.distance:=minA^.priority;

end;

procedure UpdateDistance(var Queue:PQueuePr;var

vertfromqueue:PList; const dist: integer);

var

i:Integer;

Stack:PStack;

TmpVertex:PQueuePr;

begin

TmpVertex:=Queue^.front^.next;

while tmpvertex^.vertex <> vertfromqueue^.vertexinf do

begin

tmpvertex:=TmpVertex^.next;

end;

Queue:=Queue^.front^.next;

while Queue<>nil do

begin

while vertfromqueue^.currentvertex<>nil do

begin

if (Queue^.vertex=vertfromqueue^.

currentvertex^.vertex) and (Queue^.priority>=

vertfromqueue^.currentvertex^.edge^.

weight+dist) and (not Queue^.visited)then

begin

// update distance for each connected vertices;

if Queue^.priority>vertfromqueue^.currentvertex^.

edge^.weight+dist then

begin

for i := 0 to high(Queue^.previous) do

begin

dispose(Queue^.previous[i]);

end;

SetLength(Queue^.previous,0);

Queue^.priority:=vertfromqueue^.currentvertex^.

edge^.weight+dist;

end;

for i := 0 to high(tmpvertex^.previous) do

begin

new(stack);

stack^.next:=tmpVertex^.previous[i];

stack^.vertex:=queue^.vertex;

stack^.distance:=

vertfromqueue^.currentvertex^.edge;

queue^.previous:=queue^.previous+[stack];

end;

end;

vertfromqueue^.currentvertex:=

vertfromqueue^.currentvertex^.next;

end;

vertfromqueue^.currentvertex:=

vertfromqueue^.startlist;

Queue:=Queue^.next;

end;

Queue:=TmpVertex;

end;

Function Dijcstra (var Graph:Plist; const

Start:Integer;const Finish:Integer):TWays;

var

QAddr:PQueuePr;

distance:integer;

source,destination:PList;

i:Integer;

StartVert:TWayinf;

NewWay:TWay;

CurrentVertex:Integer;

AddWay:TPopQueuePr;

DelEl:PQueuePr;

begin

PrQueueInit(Qaddr,Graph);

distance:=0;

DelEl:=Qaddr^.front^.next;

FindVertexInList(start,Graph);

StartVert.vertex:=Graph^.vertexinf;

StartVert.distance:=nil;

while delEl^.vertex^.name<>start do

begin

delEl:=DelEl^.next;

end;

DelEl^.visited:=True;

DElEl^.priority:=0;

SetLength(DelEl^.previous,1);

DelEl^.previous[0]:=nil;

CurrentVertex:=start;

// Dejcstra algorithm;

While (CurrentVertex<>Finish) do

begin

FindVertexInList(currentvertex,Graph);

UpdateDistance(QAddr,Graph,distance);

AddWay:=QueueVisited(Qaddr);

distance:=Addway.distance;

CurrentVertex:=addway.vertex^.name;

end;

Qaddr:=Qaddr^.front^.next;

while qaddr^.vertex^.name<>finish do

begin

qaddr:=qaddr^.next;

end;

// Create all ways;

for i := 0 to high(qaddr^.previous) do

begin

Newway:=wayfromstack(qaddr^.previous[i]);

newway.way:=newway.way+[startvert];

result:=result+[newway];

end;

end;

end.

Приложение З

**Исходный код программы (модуль MyBestForm)**

unit MyBestForm;

interface

uses

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

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

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

Vcl.ExtCtrls, System.Actions,

Vcl.ActnList, Vcl.ToolWin, Vcl.ActnMan, Vcl.ActnCtrls,

Vcl.ComCtrls,

System.ImageList, Vcl.ImgList,Math,

MyTypes,CreateGraph,Properties,Deikstra,UDFS,UBFS,UFile,

Vcl.StdCtrls, BestWays,ComObj;

type

TMainForm = class(TForm)

MainMenu: TMainMenu;

N1: TMenuItem;

N3: TMenuItem;

N4: TMenuItem;

N5: TMenuItem;

Excel1: TMenuItem;

Excel2: TMenuItem;

ActionList1: TActionList;

ActLFileSave: TAction;

ToolBar1: TToolBar;

BAddVert: TToolButton;

PopupMenu1: TPopupMenu;

ActLDeleteVert: TAction;

N6: TMenuItem;

Change: TAction;

N7: TMenuItem;

ActLAddVertex: TAction;

ActLAddEdge: TAction;

N8: TMenuItem;

N9: TMenuItem;

BaddEdge: TToolButton;

ImageList1: TImageList;

Panel2: TPanel;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

ActLOpenFile: TAction;

PaintBox1: TPaintBox;

PaintBoxDraw: TAction;

PaintClick: TAction;

ActLOrgGraph: TAction;

BDeleteVert: TToolButton;

BDeleteEdge: TToolButton;

ACtLDeleteEdge: TAction;

BOriented: TToolButton;

ActLSelect: TAction;

BSelect: TToolButton;

ActLResize: TAction;

FVertexInfoPanel: TPanel;

ActLCombo: TAction;

EName: TEdit;

Ecoordx: TEdit;

EcoordY: TEdit;

ESize: TEdit;

CBBack: TColorBox;

CBText: TColorBox;

CBBorder: TColorBox;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Label6: TLabel;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

Label10: TLabel;

Label11: TLabel;

FEdgeInf: TPanel;

Label12: TLabel;

EWeight: TEdit;

CBTextColor: TColorBox;

CBPEn: TColorBox;

Label13: TLabel;

Label14: TLabel;

Label15: TLabel;

Label16: TLabel;

InfToUser: TPanel;

Label17: TLabel;

PanelWays: TPanel;

InformationPanel: TPanel;

Algorithms: TComboBox;

EPenWidth: TEdit;

Label19: TLabel;

Label20: TLabel;

VPenWidth: TEdit;

Label21: TLabel;

Label22: TLabel;

ButtonShowWays: TButton;

SaveIm: TAction;

N10: TMenuItem;

SaveEx: TAction;

OpenEx: TAction;

procedure ActLFileSaveExecute(Sender: TObject);

procedure PaintBox1Paint(Sender: TObject);

procedure ActLOpenFileExecute(Sender: TObject);

procedure ActLAddVertexExecute(Sender: TObject);

procedure PaintBox1MouseDown(Sender: TObject; Button:

TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure ActLOrgGraphExecute(Sender: TObject);

procedure ActLAddEdgeExecute(Sender: TObject);

procedure ActLSelectExecute(Sender: TObject);

procedure PaintBox1MouseUp(Sender: TObject; Button:

TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure ActLDeleteVertExecute(Sender: TObject);

procedure DrawArrow(Canvas: TCanvas; CoordX1, CoordY1,

CoordX2, CoordY2, arrowSize:

Integer;Angle:Double);

function Findvertex(const

coordx,coordy:Integer):PVertex;

procedure ActualCounter();

procedure SelectHead (var VertexToSelect:PSelect);

procedure AddSelect(var VertexToSelect:PSelect;var

Vert:PVertex);

procedure DeleteSelect (var VertexToUnSelect:PSelect;

var Vert:Pvertex);

procedure DeselectAll (var Deselect:Pselect);

procedure DrawVertex(Canvas:TCanvas; var

vertexinf:TVertex;var x,y,radius:Integer);

procedure ActLResizeExecute(Sender: TObject);

procedure ACtLDeleteEdgeExecute(Sender: TObject);

procedure PaintBox1MouseMove(Sender: TObject; Shift:

TShiftState; X, Y: Integer);

procedure SetDef();

procedure FVertexInfoPanelClick(Sender: TObject);

procedure ENameKeyPress(Sender: TObject; var Key: Char);

procedure CBBackChange(Sender: TObject);

procedure EcoordxKeyPress(Sender: TObject; var Key:

Char);

procedure EcoordYKeyPress(Sender: TObject; var Key:

Char);

procedure ESizeKeyPress(Sender: TObject; var Key: Char);

procedure CBTextChange(Sender: TObject);

procedure CBBorderChange(Sender: TObject);

procedure EWeightKeyPress(Sender: TObject; var Key:

Char);

procedure CBTextColorChange(Sender: TObject);

procedure CBPEnChange(Sender: TObject);

procedure AlgorithmsSelect(Sender: TObject);

procedure WayBoxSelect(Sender: TObject);

procedure updatevertinf();

procedure updateedgeinf();

procedure EPenWidthKeyPress(Sender: TObject; var Key:

Char);

procedure VPenWidthKeyPress(Sender: TObject; var Key:

Char);

procedure SelectWay(var CurrentWay:TWay;var

wayStack:PSelectWay);

procedure Deselectway(var WayStack:PSelectWay);

procedure ButtonShowWaysClick(Sender: TObject);

procedure SaveImExecute(Sender: TObject);

procedure SaveExExecute(Sender: TObject);

procedure OpenExExecute(Sender: TObject);

procedure FormClose(Sender: TObject; var Action:

TCloseAction);

private

FSelectedVertices: PSelect;

FSelectedCount: Integer;

FSelectEdge: PEdge;

FBuffer:TBitMap;

FCounter: Integer;

FDrawVertex:Boolean;

FDrawEdge:Boolean;

FDeleteVertex:Boolean;

FDeleteEdge:Boolean;

FOriented:Boolean;

FSelect:Boolean;

FMoveTo:TPoint;

FDragging:Boolean;

FWays: TWays;

procedure InitializeVariables;

procedure FinalizeVariables;

public

VertexList:PList;

SelectStack:PSelectWay;

constructor Create(AOwner: TComponent); override;

destructor Destroy; override;

end;

var

MainForm: TMainForm;

{$R \*.dfm}

implementation

uses

Menu;

// Create form;

constructor TMainForm.Create(AOwner: TComponent);

begin

inherited Create(AOwner);

InitializeVariables;

DoubleBuffered := True;

PaintBox1.ControlStyle := PaintBox1.ControlStyle +

[csOpaque];

end;

// Delete form;

destructor TMainForm.Destroy;

begin

FinalizeVariables;

inherited Destroy;

end;

procedure TMainForm.InitializeVariables;

begin

FBuffer := TBitmap.Create;

FBuffer.Width := PaintBox1.Width;

FBuffer.Height := PaintBox1.Height;

FDrawVertex := False;

FDrawEdge := False;

FOriented:= True;

FSelectedCount := 0;

FSelect:=False;

PaintBox1.Canvas.Draw(0, 0, FBuffer);

SelectHead(FSelectedVertices);

HeadCreate(VertexList);

SelectHead(FSelectedvertices);

setdef;

SelectStack:=nil;

end;

// Check resize;

procedure TMainForm.ActLResizeExecute(Sender: TObject);

begin

FBuffer.Canvas.Brush.Color:=clWhite;

FBuffer.Width := PaintBox1.Width;

FBuffer.Height := PaintBox1.Height;

PaintBox1.Invalidate;

end;

procedure TMainForm.FinalizeVariables;

begin

FBuffer.Free;

end;

// Panel subprograms

procedure TMainForm.CBBackChange(Sender: TObject);

begin

if FSelectedCount = 1 then

begin

fSelectedVertices.front^.next^.color:=

CBBack.selected;

paintbox1.invalidate;

end;

end;

procedure TMainForm.CBBorderChange(Sender: TObject);

begin

if FSelectedCount = 1 then

begin

fSelectedVertices^.front^.next^.vertex^.bordercolor:=

CBBorder.selected;

paintbox1.invalidate;

end;

end;

procedure TMainForm.CBPEnChange(Sender: TObject);

begin

if FSelectedCount = 2 then

begin

FSelectEdge^.pencolor:=CBPen.selected;

paintbox1.invalidate;

end;

end;

procedure TMainForm.CBTextChange(Sender: TObject);

begin

if FSelectedCount = 1 then

begin

fSelectedVertices^.front^.next^.vertex^.textcolor:=

CBText.selected;

paintbox1.invalidate;

end;

end;

procedure TMainForm.CBTextColorChange(Sender: TObject);

begin

if FSelectedCount = 2 then

begin

FSelectEdge^.textcolor:=CBTextcolor.selected;

paintbox1.invalidate;

end;

end;

procedure TMainForm.EcoordxKeyPress(Sender: TObject; var

Key: Char);

begin

if key =#13 then

begin

if FSelectedCount = 1 then

begin

fSelectedVertices^.front^.next^.vertex^.x:=

strtoint(ECoordx.text);

paintbox1.invalidate;

end;

key:=#0;

end;

end;

procedure TMainForm.EcoordYKeyPress(Sender: TObject; var

Key: Char);

begin

if key =#13 then

begin

if FSelectedCount = 1 then

begin

fSelectedVertices^.front^.next^.vertex^.y:=

strtoint(ECoordY.text);

paintbox1.invalidate;

end;

key:=#0;

end;

end;

procedure TMainForm.ENameKeyPress(Sender: TObject; var

Key: Char);

begin

if key =#13 then

begin

if FSelectedCount = 1 then

begin

fSelectedVertices^.front^.next^.vertex^.name:=

strtoint(EName.text);

paintbox1.invalidate;

end;

key:=#0;

end;

end;

procedure TMainForm.EPenWidthKeyPress(Sender: TObject; var

Key: Char);

begin

if key =#13 then

begin

if FSelectedCount = 2 then

begin

FSelectEdge^.penwidth:=strtoint(EPenWidth.text);

paintbox1.invalidate;

end;

key:=#0;

end;

end;

procedure TMainForm.ESizeKeyPress(Sender: TObject; var

Key: Char);

begin

if key =#13 then

begin

if FSelectedCount = 1 then

begin

fSelectedVertices^.front^.next^.vertex^.size:=

strtoint(ESize.text);

paintbox1.invalidate;

end;

key:=#0;

end;

end;

procedure TMainForm.EWeightKeyPress(Sender: TObject; var

Key: Char);

begin

if key =#13 then

begin

if FSelectedCount = 2 then

begin

FSelectEdge^.weight:=strtoint(EWeight.text);

paintbox1.invalidate;

end;

key:=#0;

end;

end;

procedure TMainForm.VPenWidthKeyPress(Sender: TObject; var

Key: Char);

begin

if key =#13 then

begin

if FSelectedCount = 1 then

begin

fSelectedVertices^.front^.next^.vertex^.

bordersize:=strtoint(VPenWidth.text);

paintbox1.invalidate;

end;

key:=#0;

end;

end;

// toolbar subprograms;

procedure TMainForm.ActLAddEdgeExecute(Sender: TObject);

begin

FDrawVertex:=False;

BAddVert.Down := False;

BDeleteVert.Down := False;

FDeleteVertex:=False;

BDeleteEdge.Down := False;

FDeleteEdge:=False;

FSelect:=False;

BSelect.Down:=False;

if FSElectedCount=2 then

begin

if not ConnectedVertex(fSelectedVertices^.front^.

next^.vertex^.name,

fSelectedVertices^.front^.next^.next^.vertex^.name,

VertexList) then

begin

AddEdge(fSelectedVertices^.front^.next^.vertex^.

name,fSelectedVertices^.front^.next^.

next^.vertex^.name,SetEdgeProperties,

VertexList,FOriented);

Paintbox1.Repaint;

setdef();

updateedgeinf;

end

else

Paintbox1.invalidate;

end

else

begin

BAddEdge.Down := True;

FDrawEdge:=True;

end;

end;

procedure TMainForm.ActLAddVertexExecute(Sender: TObject);

begin

if FSelectedCount>0 then

begin

DeselectAll(FselectedVertices);

PaintBox1.invalidate;

end;

FDrawVertex:=True;

BAddVert.Down := True;

BAddEdge.Down := False;

FDrawEdge:=False;

BDeleteVert.Down := False;

FDeleteVertex:=False;

BDeleteEdge.Down := False;

FDeleteEdge:=False;

FSelect:=False;

BSelect.Down:=False;

end;

procedure TMainForm.ACtLDeleteEdgeExecute(Sender:

TObject);

begin

if FSelectedCount=2 then

begin

if ConnectedVertex(fSelectedVertices^.front^.next^.

vertex^.name,

fSelectedVertices^.front^.next^.next^.vertex^.

name, VertexList) then

begin

DeleteEdge(fSelectedVertices^.front^.next^.vertex^.

name,fSelectedVertices^.front^.next^.

next^.vertex^.name, VertexList);

Paintbox1.invalidate;

setdef;

end;

end

else

begin

FDrawVertex:=False;

BAddVert.Down := False;

BAddEdge.Down := False;

FDrawEdge:=False;

BDeleteVert.Down := False;

FDeleteVertex:=False;

BDeleteEdge.Down := True;

FDeleteEdge:=True;

FSelect:=False;

BSelect.Down:=False;

end;

end;

procedure TMainForm.ActLDeleteVertExecute(Sender:

TObject);

var

Select:PSelect;

TmpSelect:Integer;

begin

FDrawVertex:=False;

BAddVert.Down := False;

BAddEdge.Down := False;

FDrawEdge:=False;

BDeleteEdge.Down := False;

FDeleteEdge:=False;

FSelect:=False;

BSelect.Down:=False;

if FSElectedCount<>0 then

begin

select:=FSelectedVertices^.front^.next;

while select<>nil do

begin

TmpSelect:=select^.vertex^.name;

deleteselect(FselectedVertices,select^.vertex);

DeleteVertex(tmpselect,VertexList);

select:=select^.next;

end;

setdef;

FvertexInfoPanel.visible:=False;

InfToUser.visible:=True;

Paintbox1.Invalidate;

end

else

begin

BDeleteVert.Down := True;

FDeleteVertex:=True;

end;

end;

procedure TMainForm.ActLSelectExecute(Sender: TObject);

begin

FDrawVertex:=False;

BAddVert.Down := False;

BAddEdge.Down := False;

FDrawEdge:=False;

BDeleteVert.Down := False;

FDeleteVertex:=False;

BDeleteEdge.Down := False;

FDeleteEdge:=False;

FSelect:=True;

BSelect.Down:=True;

end;

procedure TMainForm.ActLOrgGraphExecute(Sender: TObject);

begin

if FOriented then

begin

FOriented:=False;

BOriented.ImageIndex:= 5;

end

else

begin

FOriented:=True;

BOriented.ImageIndex:= 4;

end;

end;

// file subprograms;

procedure TMainForm.ActLFileSaveExecute(Sender: TObject);

var

FileName,EFNAme,VFName:String;

begin

if SaveDialog1.execute then

begin

deselectway(SelectStack);

deselectall(FSelectedVertices);

Filename:=SaveDialog1.filename;

EFName:=Filename+'.edge';

vfName:=filename+'.vert';

SaveListToFile(VertexList,EFName,VFName);

end;

end;

procedure TMainForm.ActLOpenFileExecute(Sender: TObject);

var

FileName,EFNAme,VFName:String;

MAinLIst:Plist;

begin

OpenDialog1.Options := OpenDialog1.Options +

[ofAllowMultiSelect];

if OpenDialog1.Execute then

begin

EFNAme:='';

VFName:='';

// checking files for accompience with extension

for FileName in OpenDialog1.Files do

begin

if copy(filename,length(filename)-3,4)='vert' then

VFNAme:=Filename

else

if copy(filename,length(filename)-3,4)='edge' then

EFNAme:=Filename

end;

if (EFName<>'') and (VFName<>'') then

begin

if copy(EFName,1,length(EFName)-

4)=copy(VFName,1,length(VFName)-4) then

begin

Vertexlist:=nil;

headcreate(Vertexlist);

CreateFromFile(VErtexList,EFName,VFName);

ActualCounter;

PaintBox1.Invalidate;

end

else

begin

showmessage('Choose files with the same name');

end;

end

else

begin

showmessage('Choose files with extension .vert

and .edge');

end;

end;

end;

procedure TMainForm.SaveExExecute(Sender: TObject);

var

FileName:String;

begin

if SaveDialog1.execute then

begin

deselectway(SelectStack);

deselectall(FSelectedVertices);

Filename:=SaveDialog1.filename+'.xlsx';

SaveGraphToExcel(VertexList,filename);

end;

end;

procedure TMainForm.OpenExExecute(Sender: TObject);

var

FileName:String;

begin

if OpenDialog1.execute then

begin

Filename:=OpenDialog1.FileName;

Vertexlist:=nil;

headcreate(Vertexlist);

LoadGraphFromExcel(FileName,VertexList);

ActualCounter;

PaintBox1.Invalidate;

end;

end;

procedure TMainForm.SaveImExecute(Sender: TObject);

var

FileName:String;

begin

if SaveDialog1.execute then

begin

Filename:=SaveDialog1.filename+'.bmp';

FBuffer.SaveToFile(filename);

end;

end;

// Counter for vertices;

procedure TMainForm.ActualCounter();

var

MainList:PList;

begin

MainForm.FCounter:=0;

if MainForm.VertexList<>nil then

begin

Mainlist:=MainForm.VertexList^.front^.next;

while mainlist<>nil do

begin

if mainlist^.vertexinf^.name>MainForm.FCounter then

MainForm.FCounter:=mainlist^.vertexinf^.name;

mainlist:=mainlist^.next;

end;

end;

end;

// Subprogram that checks does the point belong to any

// vertex;

function TMainForm.findvertex(const

coordx,coordy:Integer):PVertex;

var

MainList:Plist;

Flag:Boolean;

begin

MainList:=MainForm.VertexList^.front^.next;

Flag:=False;

while (not flag) and (MainList<>nil) do

begin

if (sqr(mainlist^.vertexinf^.x-

coordx)+sqr(mainlist^.vertexinf^.y-coordy)

<=sqr(mainlist^.vertexinf^.size)) then

begin

Flag:=True;

end

else

Mainlist:=Mainlist^.next;

end;

if Flag then

begin

result:=mainlist^.vertexinf;

end

else

result:=nil;

end;

procedure TMainForm.FormClose(Sender: TObject; var Action:

TCloseAction);

begin

DeselectAll(FSelectedVertices);

DeselectWay(SelectStack);

end;

// Update information about vertex;

procedure TMainForm.updatevertinf();

begin

setDef;

Inftouser.visible:=False;

FVertexInfoPanel.visible:=True;

EName.text:=inttostr(fselectedvertices^.front^.next^.

vertex^.name);

Ecoordx.text:=inttostr(fselectedvertices^.front^.next^.

vertex^.x);

EcoordY.text:=inttostr(fselectedvertices^.front^.next^.

vertex^.y);

ESize.text:=inttostr(fselectedvertices^.front^.next^.

vertex^.size);

CBBack.selected:=fselectedvertices^.front^.next^.color;

CBText.selected:=fselectedvertices^.front^.next^.

vertex^.textcolor;

CBBorder.selected:=fselectedvertices^.front^.next^.

vertex^.bordercolor;

VPenWidth.text:=inttostr(fselectedvertices^.front^.

next^.vertex^.bordersize);

end;

// update information about edge;

procedure TMainForm.updateedgeinf();

begin

setDef;

Inftouser.visible:=False;

FEdgeInf.visible:=True;

CBTextColor.selected:=fselectedge^.textcolor;

CBPen.selected:=fselectedge^.pencolor;

EWeight.text:=inttostr(fselectedge^.weight);

EPenWidth.text:=inttostr(fselectedge^.penwidth);

end;

// hide other panels;

procedure TMainForm.SetDef();

begin

InformationPanel.visible:=True;

FEdgeINf.visible:=False;

FVertexInfoPanel.visible:=False;

Inftouser.visible:=True;

PanelWays.visible:=False;

end;

// create select head;

procedure TMainForm.SelectHead (var

VertexToSelect:PSelect);

begin

new(vertextoselect);

vertextoselect^.next:=nil;

vertextoselect^.vertex:=nil;

vertextoselect^.color:=ClBlack;

vertextoselect^.front:=vertextoselect;

end;

procedure TMainForm.AddSelect(var

VertexToSelect:PSelect;var Vert:PVertex);

var

tmp:PSelect;

begin

tmp:=vertextoselect^.front;

while tmp^.next<>nil do

begin

tmp:=tmp^.next;

end;

new(vertextoselect);

vertextoselect^.next:=nil;

vertextoselect^.front:=tmp^.front;

vertextoselect^.vertex:=Vert;

vertextoselect^.color:=Vert^.backcolor;

vertextoselect^.vertex^.selected:=True;

Vert^.backcolor:=clRed;

tmp^.next:=vertextoselect;

inc(FselectedCount);

end;

procedure TMainForm.DeleteSelect (var

VertexToUnSelect:PSelect; var Vert:Pvertex);

var

tmp:PSelect;

DTmP:PSelect;

begin

tmp:=VertexToUnSelect^.front;

vertextounselect:=vertextounselect^.front;

while tmp^.next^.vertex<>vert do

begin

tmp:=tmp^.next;

end;

tmp^.next^.vertex^.backcolor:=tmp^.next^.color;

tmp^.next^.vertex^.selected:=False;

DTMp:=tmp^.next^.next;

dispose(tmp^.next);

tmp^.next:=DTmp;

dec(FSelectedCount);

end;

procedure TMainForm.DeselectAll (var Deselect:Pselect);

var

tmp:PSelect;

DTMP:PSelect;

begin

tmp:=deselect^.front^.next;

deselect:=deselect^.front;

while tmp<>nil do

begin

tmp^.vertex^.backcolor:=tmp^.color;

tmp^.vertex^.selected:=False;

DTmp:=tmp^.next;

dispose(tmp);

tmp:=DTmp;

end;

deselect^.next:=nil;

FselectedCount:=0;

end;

procedure TMainForm.AlgorithmsSelect(Sender: TObject);

var

i:Integer;

begin

if FSelectedCount=2 then

begin

case Algorithms.itemindex of

0:Fways:=Dijcstra(VertexList,Fselectedvertices^.

front^.next^.vertex^.name,

Fselectedvertices^.front^.next^.next^.vertex^.name);

1:FWays:=BFS(VertexList,Fselectedvertices^.

front^.next^.vertex^.name,

Fselectedvertices^.front^.next^.next^.vertex^.name);

2:fWays:=DFS(VertexList,Fselectedvertices^.

front^.next^.vertex^.name,

Fselectedvertices^.front^.next^.next^.vertex^.name);

end;

if length(fWays)=0 then

begin

showmessage('Пути не найдены');

end

else

begin

PanelWays.visible:=True;

end;

end;

end;

// open form with ways;

procedure TMainForm.ButtonShowWaysClick(Sender: TObject);

begin

FormWays.EnumWays := Copy(FWays);

Application.ProcessMessages;

FormWays.AlphaBlendValue := 0;

FormWays.Show;

FormWays.AlphaBlendValue := 255;

ButtonShowWays.Enabled := false;

end;

procedure TMainForm.SelectWay(var CurrentWay:TWay;var

wayStack:PSelectWay);

var

i:Integer;

newnode:PSelectWay;

begin

for i := 0 to High(CurrentWay.way) do

begin

new(newnode);

newnode.vertex:=CurrentWay.way[i].vertex;

newnode.edge:=CurrentWay.way[i].distance;

newnode.VColor:=CurrentWay.way[i].vertex^.backcolor;

CurrentWay.way[i].vertex^.backcolor:=ClLime;

if CurrentWay.way[i].distance<>nil then

begin

newnode.Ecolor:=CurrentWay.way[i].distance^.

pencolor;

CurrentWay.way[i].distance^.pencolor:=ClLime;

end;

newnode^.next:=wayStack;

wayStack:=newnode;

end;

wayStack:=newnode;

paintbox1.invalidate;

end;

procedure TMainForm.deselectway(var WayStack:PSelectWay);

var

DTmp:PSelectWay;

begin

while WayStack<>nil do

begin

waystack^.vertex^.backcolor:=waystack^.vcolor;

if waystack^.edge<>nil then

waystack^.edge^.pencolor:=waystack^.ecolor;

DTmp:=waystack^.next;

dispose(waystack);

waystack:=DTmp;

end;

paintbox1.invalidate;

end;

procedure TMainForm.WayBoxSelect(Sender: TObject);

var

i:Integer;

FWay:Tway;

begin

DeselectWay(SelectStack);

SelectWay(Fway,SelectStack);

end;

procedure TMainForm.DrawArrow(Canvas: TCanvas; CoordX1,

CoordY1, CoordX2, CoordY2, arrowSize:

Integer;angle: Double);

var

arrowX, arrowY: Integer;

begin

Canvas.MoveTo(CoordX1, CoordY1);

Canvas.LineTo(CoordX2, CoordY2);

if CoordX2-CoordX1<0 then

arrowsize:=-arrowsize;

arrowX := Round(CoordX2 - arrowSize \*

Cos(angle + Pi / 6));

arrowY := Round(CoordY2 - arrowSize \*

Sin(angle + Pi / 6));

Canvas.LineTo(arrowX, arrowY);

Canvas.MoveTo(CoordX2, CoordY2);

arrowX := Round(CoordX2 - arrowSize \*

Cos(angle - Pi / 6));

arrowY := Round(CoordY2 - arrowSize \*

Sin(angle - Pi / 6));

Canvas.LineTo(arrowX, arrowY);

end;

procedure TMainForm.DrawVertex(Canvas:TCanvas;

var vertexinf:TVertex;var x,y,radius:Integer);

const

Scale=0.6;

var

StrName:String;

CoordX,CoordY:Integer;

begin

Canvas.Brush.Color := vertexinf.backcolor;

Canvas.Pen.Color := vertexinf.bordercolor;

Canvas.Pen.Width:=vertexinf.bordersize;

Canvas.Font.Color := vertexinf.textcolor;

x:=vertexinf.x;

y:=vertexinf.y;

Canvas.Ellipse(x - radius, y - Radius,

x + Radius, y + Radius);

StrName:=IntToStr(vertexinf.name);

Canvas.Font.Size := vertexinf.size div 2-

vertexinf.bordersize;

Canvas.font.Name:='Courier New';

CoordX:=X-Trunc(abs(canvas.font.height)\*

scale)\*length(StrName) div 2;

CoordY:=Y-abs(canvas.font.height) div 2;

Canvas.TextOut(CoordX, CoordY, StrName)

end;

procedure NameEdge (Canvas:TCanvas;

Coordx1,coordx2,coordy1,coordy2: integer; var

EdgeToName:PEdge);

const

scale = 0.6;

var

center:Tpoint;

textstartx:Integer;

Textstarty:Integer;

strname:String;

begin

center.x:=abs(coordx2+coordx1) div 2;

center.y:=abs(coordy2+coordy1) div 2;

Canvas.brush.Color := clWhite;

Canvas.Pen.Color := edgetoname^.pencolor;

Canvas.Font.Color := edgetoname^.textcolor;

StrName:=IntToStr(edgetoname^.weight);

Canvas.Font.Size := edgetoname^.penwidth\*2+10;

Canvas.font.Name:='Courier New';

textstartx:=center.x-trunc(abs(canvas.font.height)

\*scale)\*length(strname) div 2;

Textstarty:=center.y-abs(canvas.font.height) div 2;

Canvas.TextOut(textstartx, Textstarty, StrName)

end;

procedure TMainForm.PaintBox1Paint(Sender: TObject);

var

x1,y1,x2,y2: Integer;

NewX1,NewX2,NewY1,NewY2:Integer;

arc: Double;

minsize:integer;

VertexRadius1,VertexRadius2:Integer;

CosX,SinY:Double;

Mainlist:Plist;

begin

FBuffer.Canvas.Brush.Color := clWhite;

FBuffer.Canvas.FillRect(PaintBox1.Canvas.ClipRect);

Mainlist:=VertexLIst^.front^.next;

while Mainlist<>nil do

begin

x1:=Mainlist^.vertexinf^.x;

y1:=Mainlist^.vertexinf^.y;

VertexRadius1:=Mainlist^.vertexinf^.size;

DrawVertex(FBuffer.Canvas,Mainlist^.vertexinf^,

x1,y2,VertexRadius1);

Mainlist^.currentvertex:=Mainlist^.startlist;

while Mainlist^.currentvertex<>nil do

begin

if not Mainlist^.currentvertex^.edge^.isused then

begin

VertexRadius2:=

Mainlist^.currentvertex^.vertex^.size;

x2:=Mainlist^.currentvertex^.vertex^.x;

y2:=Mainlist^.currentvertex^.vertex^.y;

if Mainlist^.vertexinf^.selected and

Mainlist^.currentvertex^.vertex^.selected then

begin

FSelectEdge:=Mainlist^.currentvertex^.edge;

end;

if x2-x1=0 then

begin

arc:=90;

end

else if y2-y1=0 then

begin

arc:=0;

end

else

begin

arc:=ArcTan((y2-y1)/(x2-x1));

end;

if y2-y1<0 then

begin

CosX:=cos(arc);

SinY:=sin(arc);

end

else

begin

CosX:=-cos(arc);

SinY:=-sin(arc);

end;

if arc>0 then

begin

CosX:=-cosx;

SinY:=-siny;

end;

FBuffer.Canvas.Pen.Color :=

Mainlist^.currentvertex^.edge^.pencolor;

FBuffer.Canvas.Pen.Width :=

Mainlist^.currentvertex^.edge^.penwidth;

NewX1:=x1+trunc(Cosx\*VertexRadius1);

NewX2:=x2+trunc(-CosX\*VertexRadius2);

NewY1:=y1+trunc(SinY\*VertexRadius1);

NewY2:=y2+trunc(-SinY\*VertexRadius2);

if Mainlist^.currentvertex^.edge^.oriented then

begin

Drawarrow(FBuffer.Canvas,NewX1,NewY1,NewX2,

NewY2,min(VertexRadius1,Vertexradius2),arc);

end

else

begin

Mainlist^.currentvertex^.edge^.isused:=True;

FBuffer.Canvas.MoveTo(NewX1,NewY1);

FBuffer.Canvas.LineTo(NewX2,NewY2);

end;

nameedge(FBuffer.Canvas,NewX1,NewX2,

NewY1,NewY2,Mainlist^.currentvertex^.edge);

end

else

begin

Mainlist^.currentvertex^.edge^.isused:=False;

end;

Mainlist^.currentvertex:=

Mainlist^.currentvertex^.next;

end;

Mainlist^.currentvertex:=Mainlist^.startlist;

Mainlist:=Mainlist^.next;

end;

PaintBox1.Canvas.Draw(0, 0, FBuffer);

end;

procedure TMainForm.FVertexInfoPanelClick(Sender:

TObject);

begin

end;

procedure TMainForm.PaintBox1MouseDown(Sender: TObject;

Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

var

VertexIndex: TVertex;

begin

if (Button = mbLeft) and FDrawVertex then

begin

ActualCounter();

Inc(FCounter);

AddVertex(VertexLIst,SetVertProperties

(FCounter,X,Y));

PaintBox1.Invalidate;

end

else if (FSelectedCount>0) and FSelect then

begin

FDragging:=True;

FMoveTo:=Point(FSelectedVertices^.front^.next^.

vertex^.x,FSelectedVertices^.front^.

next^.vertex^.y);

end;

end;

procedure TMainForm.PaintBox1MouseMove(Sender: TObject;

Shift: TShiftState; X,

Y: Integer);

var

DeltaX,DeltaY:Integer;

Select: Pselect;

Flag:Boolean;

begin

if FDragging then

begin

DeltaX:=X- FMoveto.X;

DeltaY:=Y- FMoveto.Y;

if sqrt(sqr(DeltaX)+sqr(DeltaY))>30 then

begin

select:=Fselectedvertices^.front^.next;

while select<>nil do

begin

select^.vertex^.x:=select^.vertex^.x+

Deltax div 10;

select^.vertex^.y:=select^.vertex^.y+

Deltay div 10;

flag:=false;

if select^.vertex^.x-select^.vertex^.size<0 then

begin

select^.vertex^.x:=select^.vertex^.size;

Flag:=True;

end

else if select^.vertex^.x+select^.vertex^.size>

FBuffer.width then

begin

select^.vertex^.x:=FBuffer.width-

select^.vertex^.size;

Flag:=True;

end;

if select^.vertex^.y-select^.vertex^.size<0 then

begin

select^.vertex^.y:=select^.vertex^.size;

Flag:=True;

end

else if select^.vertex^.y+select^.vertex^.size>

FBuffer.Height then

begin

select^.vertex^.y:=FBuffer.Height-

select^.vertex^.size;

Flag:=True;

end;

if flag then

begin

select:=nil;

end

else

begin

select:=select^.next;

end;

end;

Paintbox1.Invalidate;

end;

end;

end;

procedure TMainForm.PaintBox1MouseUp(Sender: TObject;

Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

var

V,U: PVertex;

Select:PSelect;

Flag:Boolean;

begin

if FSelect then

begin

V := FindVertex(X, Y);

if V <> nil then

begin

Select:=FSelectedVertices^.front;

Flag:=True;

while (select<>nil) and Flag do

begin

if select^.vertex=v then

begin

if ssCtrl in Shift then

begin

DeleteSelect(FSelectedVertices,v);

end

else

begin

DeselectAll(FSelectedVertices);

AddSelect(FSelectedVertices,V);

end;

Flag:=False;

end

else

select:=select^.next;

end;

if Flag then

begin

if ssCtrl in Shift then

begin

AddSelect(FSelectedVertices,V);

end

else

begin

DeselectAll(FSelectedVertices);

addSelect(FSelectedVertices,V);

end;

end;

end

else

begin

if not(ssCtrl in Shift) then

begin

deselectAll(FSelectedVertices);

end

end;

PaintBox1.Repaint;

if fselectedcount=1 then

begin

updatevertinf;

end

else if (fselectedcount=2) and

Connectedvertex(fselectedvertices^.front^.

next^.vertex^.name,

fselectedvertices^.front^.next^.next^.

vertex^.name,VertexList) then

begin

updateedgeinf;

end

else

begin

SetDef;

end;

end

else

if FDeleteVertex then

begin

V := FindVertex(X, Y);

if V<>nil then

begin

DeleteVertex(V^.name,VertexList);

end;

PaintBox1.Invalidate;

end;

FDragging:=False;

FMoveTo := Point(0, 0);

end;

end.

Приложение И

**Исходный код программы (модуль BestWays)**

unit BestWays;

interface

uses

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

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

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

MyTypes, Vcl.StdCtrls;

type

TFormWays = class(TForm)

PanelSideBar: TPanel;

ScrollBox1: TScrollBox;

PanelParentCont: TPanel;

ButtonSort: TButton;

ButtonBackSort: TButton;

Panel1: TPanel;

procedure CreatePanel(elem: TWay; index: integer);

procedure Resize();

procedure ScrollBox1MouseWheel(Sender: TObject; Shift:

TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Handled:

Boolean);

procedure FormResize(Sender: TObject);

procedure FillParentContainer(arr: TWays);

procedure ClearParentContainer();

procedure ShowHideChkBxs(state: boolean);

procedure CheckBoxClick(Sender: TObject);

procedure CheckBox1MouseUp(Sender: TObject; Button:

TMouseButton;Shift: TShiftState; X, Y: Integer);

procedure FormShow(Sender: TObject);

procedure ButtonSortClick(Sender: TObject);

procedure QuickSortWaysByLength(var Ways: TWays; Left,

Right: Integer);

procedure FormClose(Sender: TObject; var Action:

TCloseAction);

procedure ButtonBackClick(Sender: TObject);

private

public

EnumWays: TWays;

end;

var

FormWays: TFormWays;

implementation

uses MyBestForm;

{$R \*.dfm}

procedure TFormWays.ShowHideChkBxs(state: boolean);

var

i: Integer;

begin

for i := 0 to PanelParentCont.controlCount-1 do

begin

TCheckBox(TPanel(PanelParentCont.Controls[i]).

Controls[2]).Checked := state;

end;

end;

procedure TFormWays.CreatePanel(elem: TWay; index:

integer);

var

i:Integer;

panel: TPanel;

CheckBox: TCheckBox;

WayLabel: TLabel;

LenLabel: TLabel;

const

margin = 30;

panHeight = 60;

begin

panel := TPanel.Create(PanelParentCont);

panel.parent := panelParentCont;

panel.Align := altop;

panel.Margins.top := margin;

panel.Margins.right := margin;

panel.Margins.Bottom := margin;

panel.Margins.Left := margin;

panel.AlignWithMargins := true;

panel.width := PanelParentCont.width - margin;

panel.Height := panHeight + 2\*margin;

panel.ParentBackground := false;

panel.ParentColor := false;

panel.ParentFont := false;

checkBox := TCheckBox.Create(self);

checkBox.parent := panel;

checkBox.Margins.top := margin;

checkBox.Margins.right := margin;

checkBox.Margins.Bottom := margin;

checkBox.Margins.Left := margin;

checkBox.Checked := false;

checkBox.Align := alleft;

checkBox.AlignWithMargins := true;

checkBox.OnMouseUp := FormWays.CheckBox1MouseUp;

checkBox.Tag := index;

LenLabel := TLabel.Create(panel);

LenLabel.parent := panel;

LenLabel.Margins.top := margin \* 3 div 2;

LenLabel.Margins.right := margin;

LenLabel.Margins.Bottom := margin;

LenLabel.Margins.Left := margin;

LenLabel.Caption := IntToStr(Elem.len);

LenLabel.Align := alright;

LenLabel.AlignWithMargins := true;

LenLabel.Font.Name := 'Montserrat Bold';

LenLabel.Font.Size := 12;

WayLabel := TLabel.Create(panel);

WayLabel.parent := panel;

WayLabel.Margins.top := margin \* 3 div 2;

WayLabel.Margins.right := margin;

WayLabel.Margins.Bottom := margin;

WayLabel.Margins.Left := margin;

for i:=High(Elem.way) downto 1 do

begin

WayLabel.Caption:=WayLabel.Caption+'

'+inttostr(Elem.way[i].vertex^.name)+ ' -->'

end;

WayLabel.Caption:=WayLabel.Caption+'

'+inttostr(Elem.way[0].vertex^.name);

WayLabel.Align := alClient;

WayLabel.AlignWithMargins := true;

WayLabel.Alignment := taCenter;

WayLabel.Font.Name := 'Montserrat Bold';

WayLabel.Font.Size := 12;

end;

procedure TFormWays.Resize();

begin

TPanel(panel1.Controls[0]).Height := (200)\*

TPanel(panel1.Controls[0]).ControlCount;

Panel1.Height := TPanel(panel1.Controls[0]).Height;

end;

procedure TFormWays.ButtonBackClick(Sender: TObject);

var

i: integer;

temp: TWay;

begin

Panel1.Visible := false;

QuickSortWaysByLength(EnumWays, low(EnumWays),

high(EnumWays));

for i := 0 to Length(EnumWays) div 2 - 1 do

begin

temp := EnumWays[i];

EnumWays[i] := EnumWays[Length(EnumWays) - 1 - i];

EnumWays[Length(EnumWays) - 1 - i] := temp;

end;

ClearParentContainer();

FillParentContainer(EnumWays);

Resize();

Panel1.Visible := true;

end;

procedure TFormWays.ButtonSortClick(Sender: TObject);

begin

panel1.Visible := false;

QuickSortWaysByLength(EnumWays, low(EnumWays),

high(EnumWays));

ClearParentContainer();

FillParentContainer(EnumWays);

Resize();

panel1.Visible := True;

end;

procedure TFormWays.QuickSortWaysByLength(var Ways: TWays;

Left, Right: Integer);

var

I, J: Integer;

Pivot: Integer;

Temp: TWay;

begin

if Left < Right then

begin

Pivot := Ways[(Left + Right) div 2].len;

I := Left;

J := Right;

while I <= J do

begin

while Ways[I].len < Pivot do

Inc(I);

while Ways[J].len > Pivot do

Dec(J);

if I <= J then

begin

Temp := Ways[I];

Ways[I] := Ways[J];

Ways[J] := Temp;

Inc(I);

Dec(J);

end;

end;

QuickSortWaysByLength(Ways, Left, J);

QuickSortWaysByLength(Ways, I, Right);

end;

end;

procedure TFormWays.CheckBox1MouseUp(Sender: TObject;

Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

begin

CheckBoxClick(sender);

end;

procedure TFormWays.CheckBoxClick(Sender: TObject);

var

checkbox: TChecKBox;

indexWayToDraw: integer;

begin

ShowHideChkBxs(false);

MainFOrm.DeselectWay(MainForm.SelectStack);

if (Sender is TCheckBox) then

TCheckBox(Sender).Checked := not

TCheckBox(Sender).Checked;

indexWayToDraw := TCheckBox(sender).Tag;

MainForm.SelectWay(EnumWays[indexWayToDraw],

MainForm.SelectStack);

end;

procedure TFormWays.ClearParentContainer();

begin

PanelParentCont.Free;

end;

procedure TFormWays.FillParentContainer(arr: TWays);

var

i: Integer;

parentPanel: TPanel;

begin

parentPanel := TPanel.Create(FormWays);

parentPanel.Parent := Panel1;

parentPanel.Align := alTop;

parentPanel.Padding.Left := 15;

parentPanel.Padding.Top := 15;

ParentPanel.Padding.Right := 15;

parentPanel.Padding.Bottom := 15;

parentPanel.Name := 'PanelParentCont';

parentPanel.ShowCaption := False;

parentPanel.ParentBackground := false;

parentPanel.ParentColor := false;

parentPanel.ParentFont := false;

for i := Low(arr) to High(arr) do

CreatePanel(arr[i], i);

end;

procedure TFormWays.FormClose(Sender: TObject; var Action:

TCloseAction);

begin

MainFOrm.DeselectWay(MainForm.SelectStack);

MainForm.ButtonShowWays.enabled := true;

end;

procedure TFormWays.FormResize(Sender: TObject);

begin

if PanelParentCont.ControlCount > 0 then

Resize();

end;

procedure TFormWays.FormShow(Sender: TObject);

begin

Application.ProcessMessages;

ClearParentContainer();

FillParentContainer(EnumWays);

if PanelParentCont.ControlCount > 0 then

Resize;

end;

procedure TFormWays.ScrollBox1MouseWheel(Sender: TObject;

Shift: TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Handled:

Boolean);

begin

Handled := True;

if WheelDelta < 0 then

ScrollBox1.VertScrollBar.Position :=

ScrollBox1.VertScrollBar.Position +

ScrollBox1.VertScrollBar.Increment\*3

else

ScrollBox1.VertScrollBar.Position :=

ScrollBox1.VertScrollBar.Position –

ScrollBox1.VertScrollBar.Increment\*3;

end;

end.

Приложение К

**Исходный код программы (модуль Menu)**

unit Menu;

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

TFMenu = class(TForm)

ButtonWork: TButton;

ButtonDev: TButton;

ButtonExit: TButton;

Panel1: TPanel;

Label1: TLabel;

Panel2: TPanel;

procedure ButtonWorkClick(Sender: TObject);

procedure ButtonExitClick(Sender: TObject);

procedure ButtonDevClick(Sender: TObject);

private

public

end;

var

FMenu: TFMenu;

implementation

uses MyBestForm;

{$R \*.dfm}

procedure TFMenu.ButtonDevClick(Sender: TObject);

var

F:TextFile;

Str:String;

Text:String;

begin

AssignFile(F,'About.txt');

Reset(F);

while not EOF(F) do

begin

Readln(F,Str);

Text:=text+#13#10+str;

end;

CloseFile(F);

showmessage(Text);

end;

procedure TFMenu.ButtonExitClick(Sender: TObject);

begin

Close;

end;

procedure TFMenu.ButtonWorkClick(Sender: TObject);

begin

MainForm.ShowModal;

end;

end.

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | | Наименование | | | | Дополнительные сведения | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР КП 1–40 01 01 210 ПЗ | | | | | Пояснительная записка | | | | 185 с. | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.251002-01 СА | | | | | Метод Дейкстры | | | | Формат А1 | |
|  | | | | | Схема алгоритма | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 215 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Программное средство для решения задач на графах с использованием алгоритма Дейкстры  Ведомость курсового  проекта | Литера | | | Лист | Листов |
| Разраб. | | Киселёв А.С. |  |  | Т |  |  | 185 | 185 |
| Провер. | | Фадеева Е.П. |  |  | Кафедра ПОИТ  гр. 251002 | | | | |

1. Описание структуры записи на странице 24 раздела 1.4 [↑](#footnote-ref-1)