|  |  |  |
| --- | --- | --- |
|  | ***Федеральное агентство по рыболовству***  ***Федеральное государственное бюджетное образовательное учреждение***  ***высшего образования***  ***«Астраханский государственный технический университет»***  **Система менеджмента качества в области образования, воспитания, науки и инноваций сертифицирована DQS**  **по международному стандарту ISO 9001:2015** | |
| Институт информационных технологий и коммуникаций  Направление подготовки 09.03.04 Программная инженерия  Профиль «Разработка программно-информационных систем»  Кафедра «Автоматизированные системы обработки информации и управления» | | |
| **КУРСОВОЙ ПРОЕКТ**  **Учебно-демонстрационная программа**  **«Алгоритм обхода графа в ширину»**  по дисциплине «Объектно-ориентированное программирование» | | |
| Допущен к защите  «\_\_\_» \_\_\_\_\_\_\_\_202\_\_г.  Руководитель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Оценка, полученная на защите «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_» | | Проект выполнен  обучающимся группы ДИПРб-21  Лиджигоряевым В.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Руководитель  ст. преп. Алтуфьев М.Ю. |
| Члены комиссии:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |  |

Астрахань – 2022

**ФЕДЕРАЛЬНОЕ АГЕНТСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ ПО РЫБОЛОВСТВУ**

**АСТРАХАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |  |
| --- | --- |
| **УТВЕРЖДАЮ** | Кафедра «Автоматизированные системы  обработки информации и управления» |
| Заведующий кафедрой  д.т.н. профессор  Т.В. Хоменко\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202 \_\_ г. |

**ЗАДАНИЕ**

**на выполнение Курсового проекта**

Обучающийся ***Лиджигоряев Владимир Алексеевич***

Группа ***ДИПРб-21***

Дисциплина ***Объектно-ориентированное программирование***

Тема ***Учебно-демонстрационная программа «Алгоритм обхода графа в ширину»***

Дата получения задания «16» февраля 2022 г.  
Руководитель ***ст.*** ***преподаватель***\_\_\_\_\_\_\_\_\_ ***Алтуфьев М.Ю.*** «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_202\_\_г.

должность, степень, званиеподписьФИО

Срок представления обучающимся КП на кафедру ***.*** «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_202\_\_г.

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ***Лиджигоряев В.А.*** «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_202\_\_г.

подпись ФИО

**Задачи**

Разработка программного продукта, который

* предоставляет теоретический материал по теме “Алгоритм обхода графа в ширину”;
* демонстрирует работу алгоритма;
* тестирует пользователя по изученному материалу.

**Список рекомендуемой литературы**

1. Новиков Ф.А. Дискретная математика для программистов: Учебник для вузов, 3-е изд. – СПб.: Питер, 2009. – 384 с.: ил. – (Серия “Учебник для вузов”)
2. Микони С.В. Дискретная математика для бакалавра: множества, отношения, функции, графы: Учебное пособие. – СПб.: Издательство “Лань”, 2012. – 192 с.: ил.
3. Стивенс, Род. Алгоритмы. Теория и практическое применение / Род Стивенс. – Москва : Издательство “Э”, 2016. – 544 с.

|  |  |
| --- | --- |
| **УТВЕРЖДАЮ** | К заданию на курсовой проектпо дисциплине  «Объектно-ориентированное программирование» |
| Заведующий кафедрой  д.т.н., профессор  Т.В. Хоменко\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202 \_\_ г. |

**КАЛЕНДАРНЫЙ ГРАФИК**

Курсового проектирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Разделы, темы и их содержание, графический материал | Дата сдачи | Объем, % |
| 1 | Выбор темы | 16.02.2022 | 1 |
| 2 | Разработка модели, проектирование системы   * *введение,* * *технический проект,* * *программа и методика испытаний, в том числе – данные для тестирования* * *литература* | 02.03.2022 | 25 |
| 3 | Программная реализация системы   * *работающая программа,* * *рабочий проект* * *скорректированное техническое задание (при необходимости)* | 04.04.2022 | 40 |
| 4 | Тестирование и отладка системы, эксперименты   * *работающая программа с внесёнными изменениями,* * *окончательные тексты всех разделов* | 15.04.2022 | 50 |
| 5 | Компоновка текста  Подготовка презентации и доклада   * *пояснительная записка* * *презентация* * *электронный носитель с текстом пояснительной записки, исходным кодом проекта, презентацией и готовым программным продуктом* | 03.05.2022 | 59 |
| 6 | Защита курсового проекта | 06.06.2022-11.06.2022» | 60-100 |

С графиком ознакомлен «24» февраля 2022 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, обучающийся группы ДИПРб-21

(фамилия, инициалы, подпись)

**График курсового проектирования выполнен**

без отклонений / с незначительными отклонениями / со значительными отклонениями

нужное подчеркнуть

Руководитель Курсового проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. преп. Алтуфьев А.А.

подпись, ученая степень, звание, фамилия, инициалы

**СОДЕРЖАНИЕ**

[Введение](#__RefHeading___Toc531184546) 6

[1 Технический проект](#__RefHeading___Toc531184547) 7

[1.1 Анализ предметной области](#__RefHeading___Toc531184548) 7

[1.1.1 Основные понятия в объектно-ориентированном программировании](#__RefHeading___Toc531184549) 7

[1.1.2 Граф](#__RefHeading___Toc531184549) 7

[1.1.3 Смежность](#__RefHeading___Toc531184549) 7

[1.1.4 Диаграммы](#__RefHeading___Toc531184549) 8

[1.1.5 Валентность](#__RefHeading___Toc531184549) 8

[1.1.6 Операции над графами](#__RefHeading___Toc531184549) 8

[1.1.7 Способы представления графа](#__RefHeading___Toc531184549) 9

[1.1.8 Алгоритм обхода графа в ширину](#__RefHeading___Toc531184549) 10

[1.1.9 Применение алгоритма обхода графа в ширину](#__RefHeading___Toc531184549) 10

[1.1.10 Тестирование и проверка знаний](#__RefHeading___Toc531184549) 11

[1.2 Технология обработки информации](#__RefHeading___Toc531184548) 12

[1.2.1 Форматы данных](#__RefHeading___Toc531184549) 12

[1.2.2 Инфологическая модель](#__RefHeading___Toc531184549) 14

[1.2.3 Алгоритм получения содержимого всех тегов в файле](#__RefHeading___Toc531184549) 14

[1.2.4 Алгоритм получения определений из файла с теорией](#__RefHeading___Toc531184549) 15

[1.2.5 Алгоритм получения подразделов из файла с теорией](#__RefHeading___Toc531184549) 15

[1.2.6 Алгоритм перехода к следующей странице](#__RefHeading___Toc531184549) 16

[1.2.7 Алгоритм перехода к предыдущей странице](#__RefHeading___Toc531184549) 16

[1.2.8 Алгоритм добавления вершины графа](#__RefHeading___Toc531184549) 16

[1.2.9 Алгоритм добавления ребра графа](#__RefHeading___Toc531184549) 17

[1.2.10 Алгоритм удаления вершины графа](#__RefHeading___Toc531184549) 17

[1.2.11 Алгоритм обхода графа в ширину](#__RefHeading___Toc531184549) 17

[1.2.12 Алгоритм вывода результата тестирования](#__RefHeading___Toc531184549) 18

[1.3 Входные и выходные данные](#__RefHeading___Toc531184548) 18

[1.4 Системные требования](#__RefHeading___Toc531184548) 19

[2 Рабочий проект](#__RefHeading___Toc531184547) 20

[2.1 Общие сведения о работе системы](#__RefHeading___Toc531184548) 20

[2.2 Функциональное назначение программного продукта](#__RefHeading___Toc531184548) 20

[2.3 Инсталяция и выполнение программного продукта](#__RefHeading___Toc531184548) 20

[2.4 Описание программы](#__RefHeading___Toc531184548) 21

[2.5 Разработанные меню и интерфейсы](#__RefHeading___Toc531184548) 32

[2.6 Сообщения системы](#__RefHeading___Toc531184548) 41

[3 Программа и методика испытаний](#__RefHeading___Toc531184547) 42

[3.1 Проверка работоспособности показа теории](#__RefHeading___Toc531184548) 42

[3.2 Проверка работоспособности демонстрации работы алгоритма](#__RefHeading___Toc531184548) 42

[3.3 Проверка работоспособности тестирования](#__RefHeading___Toc531184548) 42

[Заключение](#__RefHeading___Toc531184570) 44

[Список использованных источников](#__RefHeading___Toc531184571) 45

[Приложение 1 Диаграммы классов](#__RefHeading___Toc531184571) 46

# ВВЕДЕНИЕ

Часто графы позволяют упростить решение математических задач. Упрощение происходит из-за того, что данные в задаче, представленные в виде графа, становятся простыми и наглядными за счет природы графа. Многие математические доказательства также упрощаются, приобретают убедительность, если пользоваться графами.

Алгоритм обхода графа в ширину имеет огромное применение и является часто используемым алгоритмом. Алгоритм может использоваться для нахождения кратчайшего пути в невзвешенном графе или построения линейных односвязных списков вершин и ребер графа. Данный алгоритм может быть использован в практических целях, например, для нахождения решения какой-либо задачи (игры) с наименьшим числом ходов. Если каждое состояние системы можно представить вершиной графа, а переходы из одного состояния в другое — рёбрами графа, то можно использовать алгоритм обхода графа в ширину для решения задачи. Одним из самых часто используемых приложений являются различные навигаторы, карты. В этим приложениях тоже используются графы, сети дорог (автомобильных и не только) представляются взвешенным графом. Населенные пункты (или перекрестки) — это вершины графа, ребра — дороги, веса ребер — расстояния по этим дорогам.

Целью создания программного продукта является автоматизация обучения алгоритму обхода графа в ширину.

Назначение программного продукта – повышение качества знаний студентов и снижение нагрузки на преподавателя.

# 1 ТЕХНИЧЕСКИЙ ПРОЕКТ

## 1.1 Анализ предметной области

### 1.1.1 Основные понятия в объектно-ориентированном программировании

Объектно-ориентированное программирование (сокр. ООП) – методология программирования, при которой взаимодействия в программе происходят с помощью объектов, которые в свою очередь содержат данные и операции (код), используемые для манипулирования этими данными.

Объекты – строительные блоки объектно-ориентированных программ, которые представляют собой экземпляры классов [7].

Данные объекта называются атрибутами, они представляют его состояние. Поведение объекта представляет то, что он может сделать, в терминологии ООП поведения объектов содержатся в методах [7].

Класс представляет собой “чертеж объекта”, он определяет атрибуты и поведения, которые будут принадлежать всем объектам, созданным с использованием данного класса [7].

### 1.1.2 Граф

Графом G(V,E) называется совокупность двух множеств — непустого множества V (множества вершин) и множества Е двухэлементных подмножеств множества V (Е — множество рёбер) [1].

Любое множество Е двухэлементных подмножеств множества V определяет симметричное бинарное отношение на множестве V.

Пусть p – число вершин графа G, а q – число рёбер.

Тогда:

### 1.1.3 Смежность

Пусть — вершины, = — соединяющее их ребро. Тогда вершина и ребро инцидентны, ребро и вершина также инцидентны. Два ребра, инцидентные одной вершине, называются смежными; две вершины, инцидентные одному ребру, также называются смежными. Множество вершин, смежных с вершиной , называется множеством смежности (или окрестностью) вершины и обозначается :

Если не оговорено противное, то символ без индекса подразумевает .

Очевидно, что . Если — множество вершин, то Г(А) — множество всех вершин, смежных с вершинами из А:

### 1.1.4 Диаграммы

Обычно граф изображают диаграммой.

Пример:

На рисунке 1 изображена диаграмма:

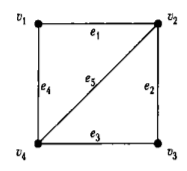


Рисунок 1 – диаграмма графа

На диаграмме вершины изображаются точками (или кружками), рёбра — линиями.

На этом рисунке граф имеет четыре вершины и пять рёбер. В этом графе вершины и , и , и , и смежны, а вершины и не смежны. Смежные рёбра: и , н , и , и , и , и , и , и . Несмежные рёбра: и , и .

### 1.1.5 Валентность

Количество рёбер, инцидентных вершине , называется степенью (или валентностью) вершины и обозначается :

Таким образом, степень вершины совпадает с количеством смежных с ней вершин. Количество вершин, не смежных с , обозначают ). Ясно, что

Если степень вершины равна нулю (то есть = 0), то вершина называется изолированной. Если степень вершины равна единице (то есть d(v) = 1), то вершина называется концевой, или висячей.

### 1.1.6 Операции над графами

Операции над графами:

1. Дополнением графа (обозначение — ) называется граф , где = = .

2. Объединением (дизъюнктным) графов и (обозначение — , при условии = ) называется граф , где = & = .

3. Соединением графов и (обозначение — , при условии = ) называется граф , где = & = .

4. Удаление вершины v из графа (обозначение — при условии ) даёт граф , где

5. Удаление ребра е из графа (обозначение — при условии

) даёт граф , где .

6. Добавление вершины v в граф (обозначение — + v при условии

) даёт граф , где .

7. Добавление ребра e в граф (обозначение — + e при условии

) даёт граф , где .

### 1.1.7 Способы представления графа

Матрица смежности – представление графа с помощью квадратной булевой матрицы размером p на p, где p – число вершин графа.

Данная матрица отражает смежность вершин. Элемент матрицы принимает следующие значения:

Матрица инцидентности – представление графа с помощью булевой матрицы размером p на q, где p – число вершин графа, q – число ребер.

Данная матрица отражает инцидентность вершин и ребер. Для неориентированного графа элемент матрицы может принимать следующие значения:

Для ориентированного графа элемент матрицы может принимать следующие значения:

Представление графа с в виде коллекции список вершин. Каждой вершине графа соответствует список, состоящий из «соседей» этой вершины.

### 1.1.8 Алгоритм обхода графа в ширину

Обход графа — это некоторое систематическое перечисление его вершин (и/или рёбер). Наибольший интерес представляют обходы, использующие локальную информацию (списки смежности). Среди всех обходов наиболее известны поиск в ширину и в глубину. Алгоритмы поиска в ширину и в глубину лежат в основе многих конкретных алгоритмов на графах. [1]

Суть алгоритма обхода графа в ширину следующая: обход начинается с посещения определённой вершины (для обхода всего графа часто выбирается произвольная вершина). Затем алгоритм посещает соседей этой вершины. За ними - соседей соседей, и так далее.

Пусть задан граф G=(V,E) и выделена исходная вершина s. Алгоритм поиска в ширину систематически обходит все ребра G для «открытия» всех вершин, достижимых из s, вычисляя при этом расстояние (минимальное количество рёбер) от s до каждой достижимой из s вершины. Алгоритм работает как для ориентированных, так и для неориентированных графов.

Поиск в ширину имеет такое название потому, что в процессе обхода мы идём вширь, то есть перед тем как приступить к поиску вершин на расстоянии k+1, выполняется обход вершин на расстоянии k.

Формальное описание алгоритма:

Дано: G = (V, E) – граф, V – множество вершин, E – множество ребер, X – множество посещенных вершин (изначально ), - начальная вершина для обхода.

Найти: список вершин V’ и список ребер E’.

Решение:

1. Зафиксировать где - начальная вершина для обхода.
2. Добавить в конец очереди Q и множество X.
3. Извлекается первый элемент Q и присваивается вершине s (s = Q\).
4. Если , выполнить пункты 5 – 6.
5. Добавить s в конец списка вершин V’.
6. , добавить в конец списка E’, добавить в конец списка V’,

конец очереди Q и множество X.

1. Пункты 3-6 повторяются для каждого первого элемента очереди Q, пока .

### 1.1.9 Применение алгоритма обхода графа в ширину

Поиск в ширину может применяться для решения задач, связанных с [теорией графов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2):

* Нахождение кратчайшего цикла в ориентированном невзвешенном графе: производим поиск в ширину из каждой вершины; как только в процессе обхода мы пытаемся пойти из текущей вершины по какому-то ребру в уже посещённую вершину, то это означает, что мы нашли кратчайший цикл, и останавливаем обход в ширину; среди всех таких найденных циклов (по одному от каждого запуска обхода) выбираем кратчайший.
* Решение какой-либо задачи (игры) за минимальное число шагов

Пример:

игра, где робот двигается по полю, при этом он может передвигать ящики, находящиеся на этом же поле, и требуется за наименьшее число ходов передвинуть ящики в требуемые позиции. Решается это обходом в ширину по графу, где состоянием (вершиной) является набор координат: координаты робота, и координаты всех коробок.

* Поиск [компонент связности](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%B0_%D1%81%D0%B2%D1%8F%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%B0) в графе;
* Поиск кратчайшего пути в невзвешенном графе.

### 1.1.10 Тестирование и проверка знаний

Вопросы, задаваемые пользователю, выбираются из базы вопросов. Выводимые вопросы зависят от выбранной темы тестирования.

Каждый вопрос в базе вопросов имеет свой уровень сложности, определяемый от 1 до 3, тему, к которой он относится и тип – открытый или закрытый. Закрытый вопрос имеет несколько вариантов ответов, открытый – не имеет вариантов ответов. При выборе конкретной темы тестирования пользователю предлагаются вопросы только данной темы.

Ответом пользователя на вопрос является только один из вариантов ответов, предложенных пользователю, в случае если это закрытый вопрос, или развернутый ответ в случае если вопрос является открытым.

Если пользователь дает правильный ответ на задание, то количество баллов, заработанных пользователем за тестирование, увеличивается на уровень сложности решенного вопроса, если же дает неправильный ответ, то количество баллов остается прежним. Пользователь может переходить к любому вопросу по ходу тестирования или временно отказаться от ответа на вопрос, перейдя к следующему. При этом ответы пользователя на отвеченные вопросы сохраняются и при повторном переходе на эти вопросы отмечается выбранный или введенный ответ.

По окончании тестирования пользователю предлагается ввести его имя, чтобы сохранить результат тестирования в бинарный файл, далее показывается итоговый результат по тестированию: имя тестируемого, количество баллов, заработанных пользователем за тестирование, и максимальное количество всех баллов, которые можно было заработать за все тестирование.

## 1.2 Технология обработки информации

Анализ предметной области показал, что программа рассчитана на одного пользователя. На рис. 1.1. показана диаграмма вариантов использования.

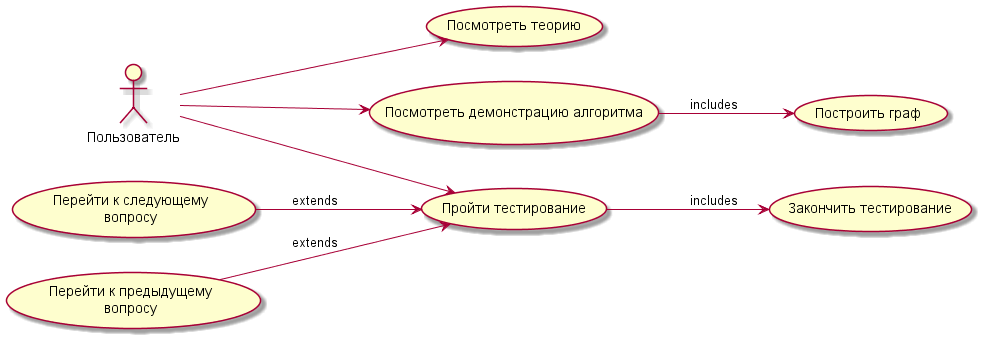


Рисунок 1.1 – Диаграмма вариантов использования

Вариант использования «Пройти тестирование» включает обязательную функцию «Закончить тестирование» и две необязательные функции «Перейти к следующему вопросу» и «Перейти к предыдущему вопросу» .

Вариант использования «Посмотреть демонстрацию алгоритма» включает обязательную функцию «Построить граф».

### 1.2.1 Форматы данных

Теоретический материал будет находиться в текстовых документа с расширением .html, написанных на языке гипертекстовой разметки HTML.

Материал разбит на подразделы и страницы. Страницы теории представляют собой сами html файлы, названия файлов теории должны быть в следующем формате: theory\_i.html, где i – индекс страницы (номер страницы – 1), любое неотрицательное число, причем номера страниц должны идти последовательно, увеличиваясь на единицу. Страницы создаются обычными html тегами, но парные теги <h1></h1> и <def></def> имеют свои дополнительные назначения:

* В тег <h1></h1> оборачиваются подразделы теоретического материала;
* В тег <def></def> оборачиваются определения.

Остальные html-теги могут использоваться по своему назначению, в любых остальных случаях.

База вопросов для тестирования будет храниться в текстовом файле формата JSON.

Файл имеет следующий формат:

{

“testTheme”: (название темы),

“questions”:

[

{

“themes”:

[

(тема 1),

(тема 2),

...

],

“type”: (тип вопроса),

“text”: (текст вопроса),

“variants”:

[

(вариант ответа 1),

(вариант ответа 2),

...

],

“correctAnswer”: (правильный вариант ответа),

“lvl”: (уровень сложности вопроса)

},

...

]

}

В данном формате любое свойство указывается в кавычках, содержимое свойство должно указываться после двоеточия, содержимое объектов указываются с помощью фигурных скобок, массивов – с помощью квадратных. Многоточие показывает, что за место них могут записываться другие элементы. В файле создается главный объект с помощью фигурных скобок ({ и }), между этими скобками указываются 2 главных свойства: testTheme – тема тестирования (строка) и questions – вопросы. В вышеописанном формате в круглых скобках должны быть пользовательские строки, окаймленные кавычками с двух сторон). Свойство questions представляет собой ассоциативный массив, состоящий из объектов (вопросов). Размерность questions определяется количеством вопросов. Каждый элемент массива questions (вопрос) состоит из свойств: themes (одномерный строковый массив, состоящий из тем к которым относится вопрос, представлено массивом, элементы – строки через запятую, размером с количеством тем, к которым относится вопрос), type (строка – тип вопроса), text (строка – текст вопроса), variants (варианты ответов, представлено массивом, элементы – строки через запятую, размерность определяется количеством вариантов ответов), correctAnswer (строка – правильный ответ на вопрос), lvl (строка – уровень сложности вопроса).

В вышеуказанном примере формата файла показан массив questions, состоящий только из одного вопроса, вопросы можно добавлять через запятую, создавая новый объект, как новый элемент массива questions.

### 1.2.2 Инфологическая модель

1. В ходе анализа предметной области были разработаны диаграммы классов подсистем и диаграмма основных классов системы, которая показывает взаимосвязь между классами трех основных подсистем (см. Приложение 1).

### 1.2.3 Алгоритм получения содержимого всех тегов в файле

1. Дано: строка filePath (путь к файлу, поле класса TheoryPage, см. Приложение 1), строка tagName – имя тега, содержимое которых собираем.
2. Требуется: сформировать одномерный массив ***result*** строкового типа, содержащий содержимое тегов с именем tagName в файле filePath.
3. Решение:
4. Создать массив строкового типа result на 0 ячеек.
5. Открыть файл на чтение.
6. Пока не конец файла, повторять
7. | Считать строку из файла в строку textLine.
8. | Записать в целочисленную переменную startIndex индекс первого
9. | вхождения подстроки “<{tagName} >” в textLine.
10. | Записать в целочисленную переменную endIndex индекс первого
11. | вхождения подстроки “</ tagName >” в textLine.
12. | Получить подстроку из textLine со startIndex до endIndex и записать ее в
13. | конец result.
14. конец цикла
15. Вернуть result.
16. Конец алгоритма.

### 1.2.4 Алгоритм получения определений из файла с теорией

1. Дано: строка filePath (путь к файлу, поле класса TheoryPage, см. Приложение 1).
2. Требуется: сформировать одномерный массив ***result*** строкового типа.
3. Решение:
4. Создать массив строкового типа result на 0 ячеек.
5. result = вызов алгоритма 1.2.3 с аргументами filePath и “dfn” массиву result.
6. Вернуть result.
7. Конец алгоритма.

### 1.2.5 Алгоритм получения подразделов из файла с теорией

1. Дано: строка filePath (путь к файлу, поле класса TheoryPage, см. Приложение 1).
2. Требуется: сформировать одномерный массив ***result*** строкового типа.
3. Решение:
4. Создать массив строкового типа result на 0 ячеек.
5. result = вызов алгоритма 1.2.3 с аргументами filePath и “h1” массиву result.
6. Вернуть result.
7. Конец алгоритма.

### 1.2.6 Алгоритм перехода к следующей странице

1. Дано: одномерный массив pages типа TheoryPage (поле класса Theory), целое число currentPageNum - номер текущей страницы теоретического материала (поле класса Теория, см. Приложение 1).
2. Требуется: сменить текущую страницу (поле currentPage класса Theory) на следующую в pages.
3. Решение:
4. Увеличить currentPageNum на 1.
5. Если currentPageNum < размер pages, то
6. | currentPage = pages[currentPageNum].
7. иначе
8. | Уменьшить currentPageNum на 1.
9. конец ветвления
10. Конец алгоритма.

### 1.2.7 Алгоритм перехода к предыдущей страницы

1. Дано: одномерный массив pages типа TheoryPage (поле класса Theory), целое число currentPageNum - номер текущей страницы теоретического материала (поле класса Теория, см. Приложение 1).
2. Требуется: сменить текущую страницу (поле currentPage класса Theory) на следующую в pages.
3. Решение:
4. Уменьшить currentPageNum на 1.
5. Если currentPageNum < размер pages, то
6. | currentPage = pages[currentPageNum].
7. иначе
8. | Увеличить currentPageNum на 1.
9. конец ветвления
10. Конец алгоритма.

### 1.2.8 Алгоритм добавления вершины графа

1. Дано: Node node – добавляемая вершина, connFlag – флаг состояния (см. Приложение 1).
2. Решение:
3. Присвоить целочисленным переменным x и y 0.
4. Присвоить целочисленной переменной nodesInRow 6.
5. Присвоить x значение (-2 \* (2\* радиус у node + 10) + (id у node) % nodesInRow) \* (2 \* радиус у node + 10).
6. Присвоить y значение (-100 + 2 \* радиус у node + 10 + (id у node) / nodeInRow) \* (2 \* радиус у node + 10).
7. Добавить в nodes node.
8. Задать координаты x и y для вершины node и вывести ее.
9. Присвоить connFlag 0.
10. Конец алгоритма.

### 1.2.9 Алгоритм добавления ребра графа

1. Дано: Edge edge – добавляемое ребро (см. Приложение 1).
2. Решение:
3. Если какая-либо вершина выделена, то
4. | Вывести сообщение о том, что требуется нажатие на еще одну вершину, чтобы
5. | провести ребро и присвоить connFlag 2.
6. иначе
7. | Присвоить connFlag 0.
8. конец ветвления
9. Конец алгоритма.

### 1.2.10 Алгоритм удаления вершины графа

1. Дано: Node node – удаляемая вершина, nodes – список вершин (поле “вершины” класса Демонстрация, см. Приложение 1).
2. Требуется: удалить вершину node.
3. Решение:
4. Удалить node из nodes.
5. Удалить все ребра из edges, которые связаны с node.
6. Вывести сообщение о том, что вершина удалена
7. Конец алгоритма.

### 1.2.11 Алгоритм обхода графа в ширину

1. Дано: Node node – начальная вершина для алгоритма, states – одномерный массив шагов алгоритма (поле “шаги” в классе Demo, см. Приложение 1)
2. Требуется: обойти все вершины графа, окрасить в конечном счете каждую в черный цвет.
3. Решение:
4. Объявить одномерный массив целочисленного типа x на N элементов.
5. Создать шаг алгоритма state типа BSFState.
6. Записать в state цвета всех вершин, как белые, и добавить state в конец states
7. Изменить цвет для node в state на серый и добавить state в конец state.
8. Объявить очередь nodesQueue типа Node.
9. Добавить в nodesQueue node.
10. Присвоить x[id у Node] 1.
11. Повторять, пока nodesQueue не пуста
12. | Исключить из nodesQueue вершину и записать ее в checkedNode
13. | Изменить цвет для checkedNode в state на черный и добавить state в конец states.
14. | Пока не пройдены все ребра инцидентные checkedNode, повторять
15. | | Присвоить очередную вершину смежную с checkedNode по очередному ребру
16. | | checkingNode.
17. | | Если x[id у checkingNode] = 0, то
18. | | | Добавить в nodesQueue checkingNode.
19. | | | Присвоить x[id у checkingNode] = 1.
20. | | | Изменить цвет для checkedNode в state на серый и добавить state в конец
21. | | | states.
22. | | конец ветвления
23. | конец цикла
24. | Добавить в конец states state.
25. конец цикла
26. Конец алгоритма.

### 1.2.12 Алгоритм вывода результата тестирования

1. Дано: test – поле типа Test в классе TestWgt, questions – массив типа Question в классе Test (см. Приложение 1).
2. Требуется: вычислить количество набранных баллов и количество всех баллов и вывести все это.
3. Решение:
4. Присвоить целочисленной переменной totalScore и userScore 0.
5. Поэлементно сравнивать поля у Test: questions[i].correctAnswer и questions[i].userAnswer при этом увеличивая totalScore на questions[i].difficultyLevel, i пробегает от 0 до размера questions.
6. Если соответствующие элементы questions[i].correctAnswer и questions[i].userAnswer равны, то увеличиваем userScore на значения поля questions[i].difficultyLevel.
7. Вывести сообщение о количество набранных баллов из всех возможных, выводя userScore и totalScore.
8. Конец алгоритма.

## 1.3 Входные и выходные данные

Входные данные:

* строка: путь к папке с теоретическим материалом;
* односвязный список типа Node: множество вершин графа;
* односвязный список типа Edge: множество ребер графа;
* строка: путь к файлу с тестом;
* односвязный список типа Question: вопросы из теста;
* вектор типа TheoryPage: страницы теории;

Выходные данные:

* одномерный массив типа BSFState: массив шагов алгоритма обхода.
* целое число: количество набранных баллов за тестирование;
* целое число: максимальное количество баллов, которое возможно набрать;
* объект типа TestResult: результат тестирования

## 1.4 Системные требования

Рекомендуемая конфигурация:

* Intel-совместимый процессор с частотой не менее 1,6 ГГц;
* не менее 2 Гб ОЗУ;
* не менее 20 MБ свободного места на диске;
* дисковод CD-ROM/DVD-ROM.

Операционная система: Windows 10.

# 2 РАБОЧИЙ ПРОЕКТ

## 2.1 Общие сведения о работе системы

Программный продукт разработан в интегрированной среде Qt Creator (версия 4.15.0) на языке C++ с использованием фреймворка Qt 5.15.2. Программа работает под управлением операционной системы Windows 10.

## 2.2 Функциональное назначение программного продукта

Разработанный программный продукт предназначен для обучения по теме “Алгоритм обхода графа в ширину”. Программа имеет следующие функциональные возможности:

* предоставление пользователю просмотра теории по теме “Алгоритм обхода графа в ширину”;
* предоставление пользователю ввода вершин и ребер графа для демонстрации работы алгоритма;
* предоставление пользователю выбора начальной вершины для начала алгоритма;
* предоставление пользователю просмотра демонстрации работы алгоритма обхода графа;
* предоставление пользователю прохождения тестирования по теме;
* проверка правильности ответа пользователем на вопрос в тесте;
* прекращение тестирования при желании пользователя;
* вывод итогового результата тестирования после окончания;

Программа имеет следующие функциональные ограничения:

* + количество вершин в графе для демонстрации должно быть не более 9.

## 2.3 Инсталляция и выполнение программного продукта

Для выполнения программы необходимо:

1. Скопировать на жесткий диск компьютера папку “Алгоритм обхода графа в ширину”, содержащую исполняемый файл bsf.exe, testing.json и папку Theory с файлами theory\_0.html, theory\_1.html, theory\_2.html, theory\_3.html.
2. Запустить на выполнение файл bsf.exe.

## 

## 2.4 Описание программы

На рисунке 2.1 показана диаграмма взаимодействия модулей программы

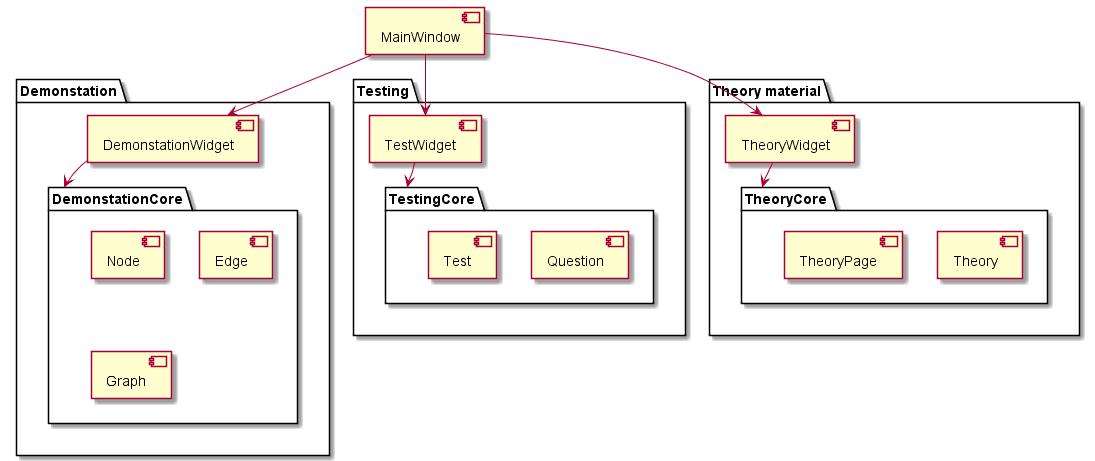


Рисунок 2.1 – диаграмма взаимодействия модулей программы

В программе модуль MainWindow является основным модулем для интерфейса с пользователем, который вызывает остальные модули взаимодействия. Модуль TheoryWidget – модуль, отвечающий за показ теории и взаимодействие с пользователем по теории. Этот модуль работает с модулями TheoryPage и Theory, которые представляют данные теории – страницу теории и теоретический материал соответственно. Модуль DemonstrationWidget отвечает за вывод графа на экран и демонстрирование работы алгоритма обхода графа в ширину, он вызывает модули из DemonstationCore, которые представляют данные для демонстрации: Node – вершина графа, Edge – ребро графа, Graph – сам граф. Модуль TestWidget отвечает за вывод теста и взаимодействие с пользователем по тестированию. TestWidget управляет работой модулей из TestingCore, представляющих данные для теста: Question – вопрос в тесте, Test – сам тест.

В таблице 2.1 приведено описание класса Node (Вершина).

Таблица 2.1 – Описание класса Node

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Квалификатор доступа** | **Назначение** |
| pen | QPen | protected | необходим для прорисовки текста вершины |
| nodeColor | QColor | protected | цвет вершины |
| edgeList | QList<Edge \*> | protected | Список инцидентных ребер |
| newPos | QPointF | protected | Координаты вершины на экране |
| idStatic | static int | protected | Количество всех вершин |

Продолжение таблицы 2.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | **Назначение** | |
| textContent | QString | | protected | Текст внутри вершины | |
| **Метод** | | **Квалификатор доступа** | | | **Назначение** |
| Node(QColor nodeClr = Qt::white, QString text = nullptr, QColor textColor = Qt::black) | | public | | | инициализация полей |
| void addEdge(Edge \*edge) | | public | | | добавляет edge в edgeList |
| void removeEdge (Edge \*edge) | | public | | | удаляет edge из edgeList |
| QList<Edge \*> edges() | | public | | | возвращает edgeList |
| QRectF boundingRect() | | public | | | Возвращает область для перерисовки |
| void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) | | public | | | Прорисовывет с помощью painter область, хранимую в option |
| int GetId() | | public | | | Возвращает id вершины |
| QColor GetNodeColor() | | public | | | Возвращает nodeColor |
| void SetNodeColor(QColor newColor, bool updating = true) | | public | | | Устанавливатет nodeColor в newColor, если updating=true, то перерисовывает вершину |
| QColor GetPenColor() | | public | | | Возвращает цвет, используемый для рисования текста внутри вершины |
| void SetPenColor(QColor penColor, bool updating = true) | | public | | | Уставливает цвет, используемый для текста внутри вершины на penColor, если updating=true, то перерисовывает вершину |
| QVariant itemChange(GraphicsItemChange change, const QVariant &value) | | protected | | | Обрабатывает изменения позиции вершин |

В таблице 2.2 приведено описание класса Edge (Ребро).

Таблица 2.2 – Описание класса Edge

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| startNode | Node\* | | protected | | указатель на первую вершину |
| endNode | Node\* | | protected | | указатель на вторую вершину |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| Edge(Node\* startNode, Node\* endNode = nullptr) | | public | | инициализация полей | |
| Node\* GetStartNode() | | public | | возвращает первую вершину | |
| Node\* GetEndNode() | | public | | возвращает вторую вершину | |

В таблице 2.3 приведено описание класса Graph.

Таблица 2.3 – Описание класса Graph

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| void addNode() | public | добавление вершины на экран |
| void shuffle() | public | случайно сместить вершину на другое место |
| void zoomIn() | public | приблизить граф |
| void zoomOut() | public | отдалить граф |
| void keyPressEvent(QKeyEvent \*event) | protected | обрабатывает нажатия клавиш |
| void mousePressEvent(QMouseEvent \*event) | protected | обрабатывает нажатия мыши |
| void wheelEvent(QWheelEvent \*event) | protected | обрабатывает движения колесика мыши |
| void scaleView(qreal scaleFactor) | protected | увеличивает граф в scaleFactor раз |

Продолжение таблицы 2.3

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| void mouseReleaseEvent(QMouseEvent \*event) | protected | обрабатывает отпускания кнопок мыши |

В таблице 2.4 приведено описание класса TheoryPage (страница теории).

Таблица 2.4 – Описание класса TheoryPage

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| filePath | QString | | private | | путь к файлу с теорией |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| explicit Theory(const QString filePath) | | public | | инициализация поля | |
| Theory() | | public | | инициализация поля | |
| void SetFilePath(const QString& filePath) | | public | | установка нового пути к файлу | |
| QString GetFilePath() | | public | | возвращает поле filePath | |
| QVector<QString> Definitions() | | public | | возвращает вектор определений на странице | |
| QVector<QString> Subsections() | | public | | возвращает вектор подзаголовков на странице | |
| bool operator==(const TheoryPage& another) | | public | | сравнивает текущую страницу с another | |

В таблице 2.5 приведено описание класса Theory (теория).

Таблица 2.5 – Описание класса Theory

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Квалификатор доступа** | **Назначение** |
| pages | QVector<TheoryPage> | private | набор страниц теории |

Продолжение таблицы 2.5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| currentPage | TheoryPage | | private | | текущая страница |
| currentPageNum | uint | | private | | номер текущей страницы |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| explicit Theory(const QString filesDirPath) | | public | | инициализация полей | |
| void LoadPages(const QString& filesDirPath) | | public | | загружает страницы теории из папки filesDirPath | |
| void RemovePage(const TheoryPage &page) | | public | | удаляет page из pages | |
| void RemoveAllPages() | | public | | удаляет все страницы из pages | |
| TheoryPage GetCurrentPage() | | public | | возвращает currentPage | |
| QVector<TheoryPage> GetPages() | | public | | возвращает pages | |
| uint GetCurrentPageNum() | | public | | возвращает currentPageNum | |
| void SaveContentOnDisc(const QString &dirPath) | | public | | сохраняет содержание теории в папке dirPath в виде файла content.html | |
| void SaveDictionaryOnDisc(const QString &dirPath) | | public | | сохраняет словарь теории в папке dirPath в виде файла dictionary.html | |

В таблице 2.6 приведено описание класса TheoryWidget.

Таблица 2.6 – Описание класса TheoryWidget

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Квалификатор доступа** | **Назначение** |
| webView | QWebEngineView\* | private | виджет для вывода теории |
| theory | Theory | private | теория |

Продолжение таблицы 2.6

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| explicit TheoryWidget(const QString filesDirPath) | public | инициализация полей |
| void InitializeUI() | private | инициализация пользовательского интерфейса |
| void Go() | private | Загружает текущую страницу теории в webView |

В таблице 2.7 приведено описание класса Demo (демонстрация).

Таблица 2.7 – Описание класса Demo

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| nodes | QList <Node\*> | | private | | путь к файлу с теорией |
| edges | QList <Edge\*> | | private | | номер страницы с началом текущего подраздела |
| states | QVector<BSFState> | | private | | определения и их страницы |
| currentState | int | | private | | подразделы и их |
| \_source | Node \* | | private | | начальная вершина при соединении |
| connFlag | int | | private | | флаг состояния соединения |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| explicit Demo(QWidget \*parent = nullptr) | | public | | инициализация полей | |
| void BSF() | | private | | производит алгоритм обхода графа | |

Продолжение таблицы 2.7

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| void loadState(int stateNum) | private | загружает states[stateNum] |
| void on\_btnAddVert\_clicked() | private | добавляет вершину |
| void on\_btnAddEdge\_clicked () | private | добавляет ребро |
| void on\_btnDeleteVert\_clicked () | private | удаляет вершину |
| void on\_btnBSF\_clicked () | private | вызывает BSF() |
| void on\_btnNextStep\_clicked () | private | переход к следующему шагу алгоритма |
| void on\_btnPrevStep\_clicked () | private | переход к предыдущему шагу алгоритма |
| void sceneSelectionChanged () | private | обрабатывает выделения вершин |
| void keyPressEvent (QKeyEvent \*event) | private | обрабатывает нажатия клавиш |

В таблице 2.8 приведено описание класса Question.

Таблица 2.8 – Описание класса Question

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| text | QString | | private | | текст вопроса |
| answerVariants | QList <QString> | | private | | список вариантов ответов |
| correntAnswer | QString | | private | | правильный ответ |
| userAnswer | QString | | private | | ответ пользователя |
| themes | QList<QString> | | private | | темы вопроса |
| type | QString | | private | | тип вопроса |
| difficultyLevel | QString | | private | | уровень сложности |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| Question(QString text, QList<QString> anVars, QString corAns, QList<QString> themes, QString type, | | public | | инициализация полей | |

Продолжение таблицы 2.8

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| QString difLvl, QString userAns) |  |  |
| Question() | public | инициализация полей |
| Question(const Question& another) | public | конструктор копирования |
| Question& operator=(const Question& another) | public | перегрузка оператора присваивания |
| QString GetText() | public | возвращает text |
| QList<QString> GetAnswerVariants() | public | возвращает answerVariants |
| QString GetCorrectAnswer() | public | возвращает correctAnswer |
| QList<QString> GetThemes() | public | возвращает themes |
| QString GetType() | public | возвращает type |
| QString GetDifficultyLevel() | public | возвращает difficultyLevel |
| QString GetUserAnswer() | public | возвращает userAnswer |
| void SetUserAnswer(const QString&) | public | устанавливает ответ пользователя |
| void FromJSON(const JSonObject& ) | public | инициализация полей из json объекта |
| void ShuffleAnswerVariants() | public | случайное перемешивание answerVariants |

В таблице 2.9 приведено описание класса Test.

Таблица 2.9 – Описание класса Test

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Квалификатор доступа** | **Назначение** |
| questions | QList<Question> | private | набор вопросов |
| currentQuestion | Question\* | private | указатель на текущий вопрос |
| currentQuestionNumInList | uint | private | номер текущего вопроса в questions |
| theme | QString | private | тема теста |
| passingUserName | QString | private | имя сдающего тест |

Продолжение таблицы 2.9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| askingQuestionsCount | uint | | private | | количество задаваемых вопросов |
| \_mode | Mode | | private | | режим тестирования |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| Test(QString testFilePath, const Mode& mode, const uint& askingQueCount = 5) | | public | | инициализация полей | |
| Test (const Test & another) | | public | | конструктор копирования | |
| Test & operator=(const Test & another) | | public | | перегрузка оператора присваивания | |
| QList<QString> Questions() | | public | | возвращает questions | |
| void SetCurrentQuestion(const uint& num) | | public | | устанавливает currentQuestion на questions[num] | |
| Question\* CurrentQuestion() | | public | | возвращает currentQuestion | |
| QString Theme() | | public | | возвращает theme | |
| void ToNextQuestion() | | public | | устанавливает currentQuestion на questions [currrentQuestionNum + 1] | |
| void ToPrevQuestion() | | public | | устанавливает currentQuestion на questions [currrentQuestionNum - 1] | |
| void SetPassingUserName(const QString &usrName) | | public | | присваивает userName usrName | |
| QString PassingUserName() | | public | | возвращает userName | |
| void Shuffle() | | public | | случайное перемешивание списка questions | |
| uint GetTotalScore() | | public | | возвращает максимальное количество очков, которое можно получить за тест | |

Продолжение таблицы 2.9

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| uint GetUserScore() | public | возвращает количество очков, набранное пользователем |
| void SaveTestResultInFile(const QString& filePath) | public | сохраняет результат теста в файле filePath |
| uint GetAskingQuestionsCount() | public | возвращает askingQuestionsCount |
| void SetAskingQuestionsCount(const uint& newCount) | public | присваивает askingQuestionsCount newCount |
| Mode mode() | public | возвращает \_mode |
| void LoadFromJSON(const QString &testFilePath) | private | читает файла testFilePath, как JSON файл и инициализирует поля из него |

В таблице 2.10 приведено описание класса TestWgt.

Таблица 2.10 – Описание класса TestWgt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| test | Test\* | | private | | указатель на тест |
| rbtns | QRadioButton\*\* | | private | | массив указателей на переключатели |
| ui | Ui::TestWgt | | private | | указатель на объект пользовательского интерфейса |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| explicit TestWgt() | | public | | инициализация полей | |
| void on\_btnToNextQ\_clicked() | | private | | обработка перехода к следующему вопросу | |
| void on\_btnToPrevQ\_clicked() | | private | | обработка перехода к предыдущему вопросу | |

Продолжение таблицы 2.10

|  |  |  |
| --- | --- | --- |
| **Метод** | **Квалификатор доступа** | **Назначение** |
| void on\_btnFinish\_clicked(); | private | обработка нажатия на кнопку завершения тестирования |
| void on\_btnOk\_clicked(); | private | обработка нажатия на кнопку сохранения результата |
| void OutCurrentQuestion(); | private | вывод текущего вопроса на экран |

В таблице 2.11 приведено описание класса MainWindow (главное окно).

Таблица 2.11 – Описание класса MainWindow

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Поле** | **Тип** | | **Квалификатор доступа** | | **Назначение** |
| ptheory | Theory\* | | private | | указатель на окно с теорией |
| pdemo | Demo\* | | private | | указатель на окно с демонстрацией |
| ptesting | Testing\* | | private | | указатель на окно с тестированием |
| **Метод** | | **Квалификатор доступа** | | **Назначение** | |
| MainWindow(QWidget \*parent = nullptr) | | public | | инициализация полей, установка окна с теорией как начальное | |

В таблице 2.12 приведены важнейшие переменные, используемые в программе.

Таблица 2.12 – Важнейшие переменные программы

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| a | QApplication | объект приложения |
| w | MainWindow | главное окно приложения |

Продолжение таблицы 2.12

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| questions | QList<Question> | вопросы теста |
| currentQuestion | Question | текущий вопрос в тесте |
| currentPage | TheoryPage | текущая страница теории |
| pages | QVector<TheoryPage> | страницы теории |
| nodes | QList<Node\*> | список указателей на вершины графа |
| edges | QList<Edge\*> | список указателей на ребра графа |
| states | QVector<BSFState> | состояния вершин при прохождении алгоритма обхода графа в ширину для каждого шага |

## 2.5 Разработанные меню и интерфейсы

После запуска программы на выполнение появится окно программы (рис.2.2).

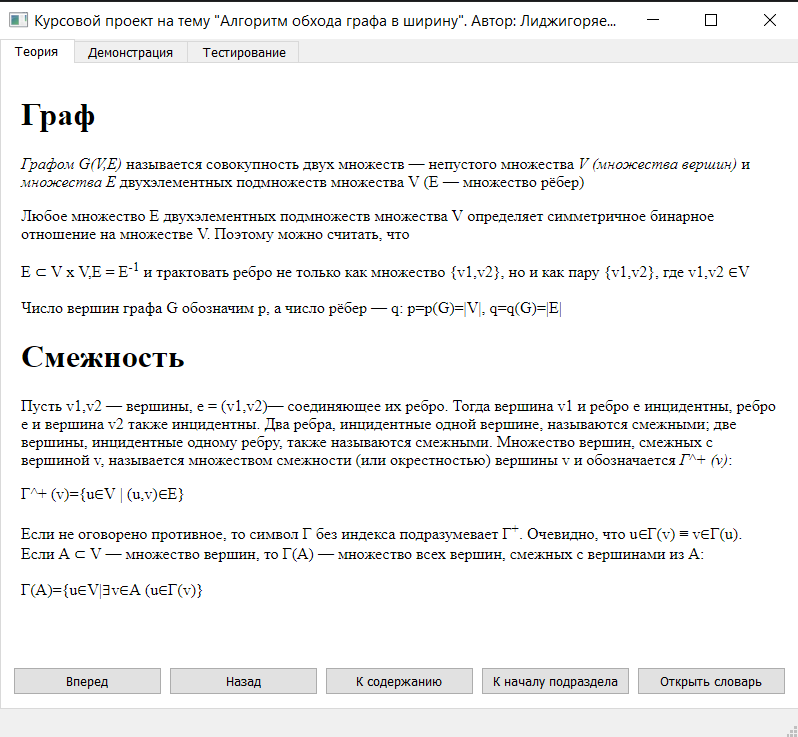


Рисунок 2.2 – Программа при запуске

При запуске стартовой вкладкой является вкладка с теорией. Между вкладками между переходить, нажимая на них.

Переходить по разным страницам теории можно переходя на следующую или предыдущую страницы. При нажатии на кнопку “вперед” показывается следующая страница (рис. 2.3).

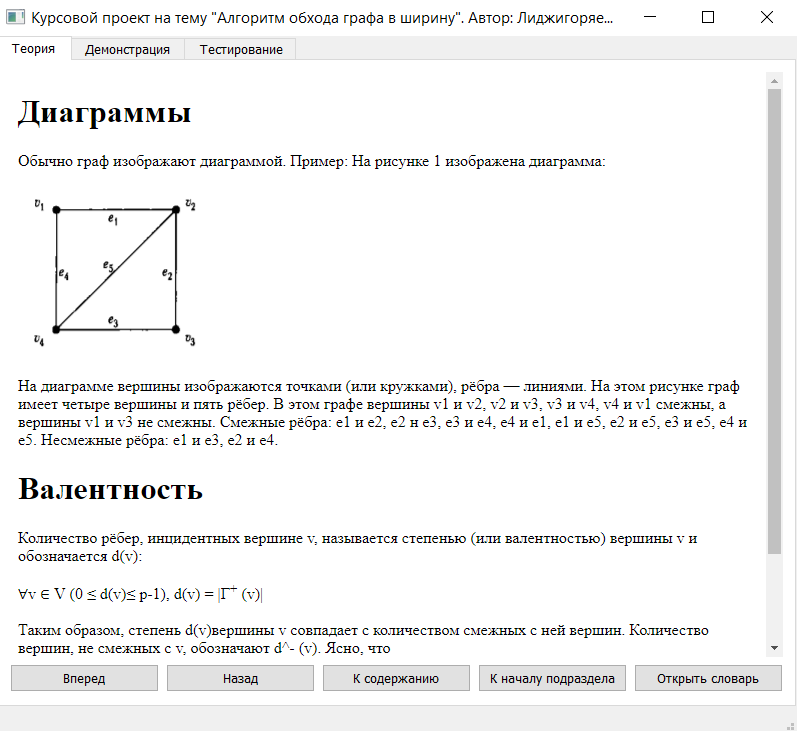


Рисунок 2.3 – Следующая страница теории

При нажатии на кнопку “назад” показывается предыдущая страница (рис. 2.2).

Посмотреть содержание теоретического материала можно, нажав на кнопку “к содержанию” (рис. 2.4).

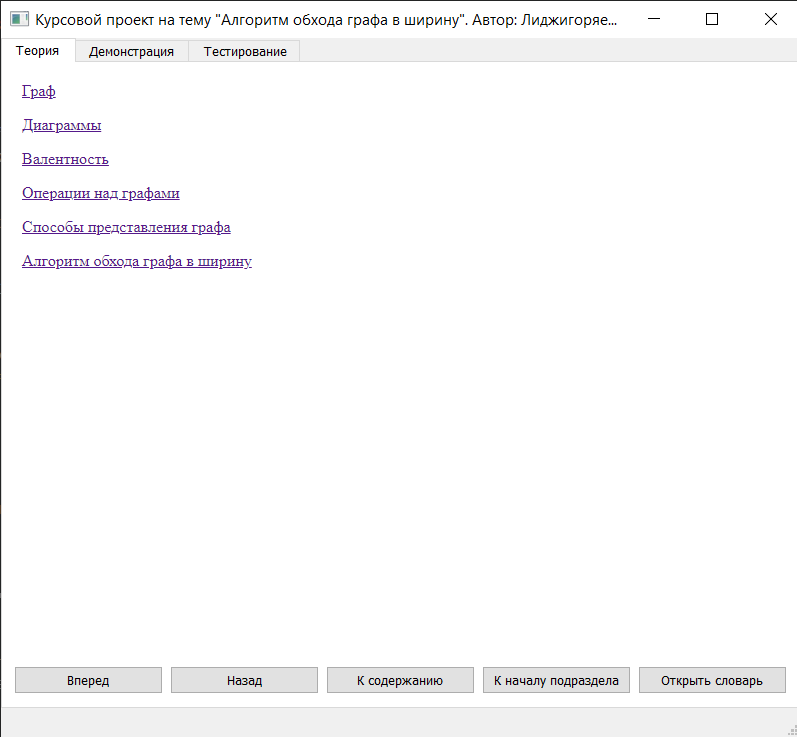


Рисунок 2.4 – Содержание теории

При нажатии на кнопку “открыть словарь” открывается словарь определений (рис 2.5).

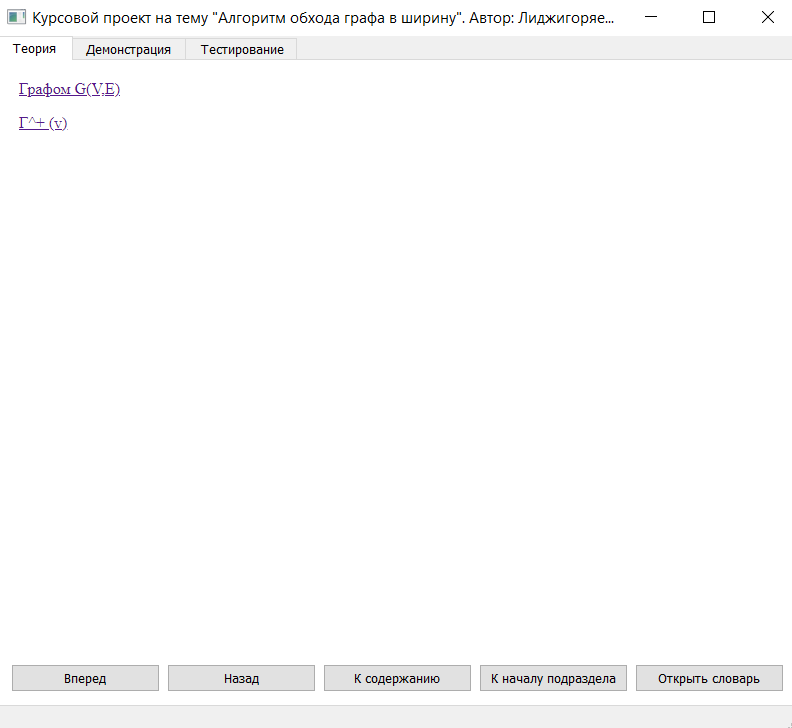


Рисунок 2.5 – Словарь определений

По нажатии на определение из словаря определений выполняется переход на страницу, где это определение было дано.

При нажатии на вкладку “Демонстрация” появляется форма для демонстрации алгоритма обхода графа (рис. 2.6).

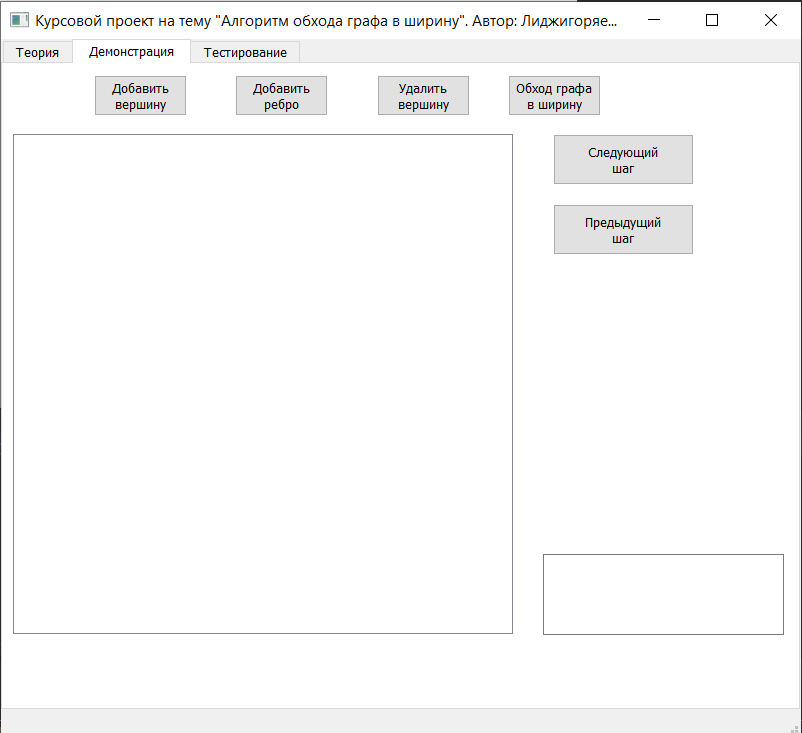


Рисунок 2.6 – форма демонстрации

При нажатии на кнопку “Добавить вершину” добавляется вершина на форму графа, с помощью нажатия на кнопку “Добавить ребро” можно соединить две вершины, кликнув по обеим вершинам. При нажатии на кнопку “Удалить вершину” происходит удаление вершины.

На рисунке 2.7 показано добавление вершины на форму.

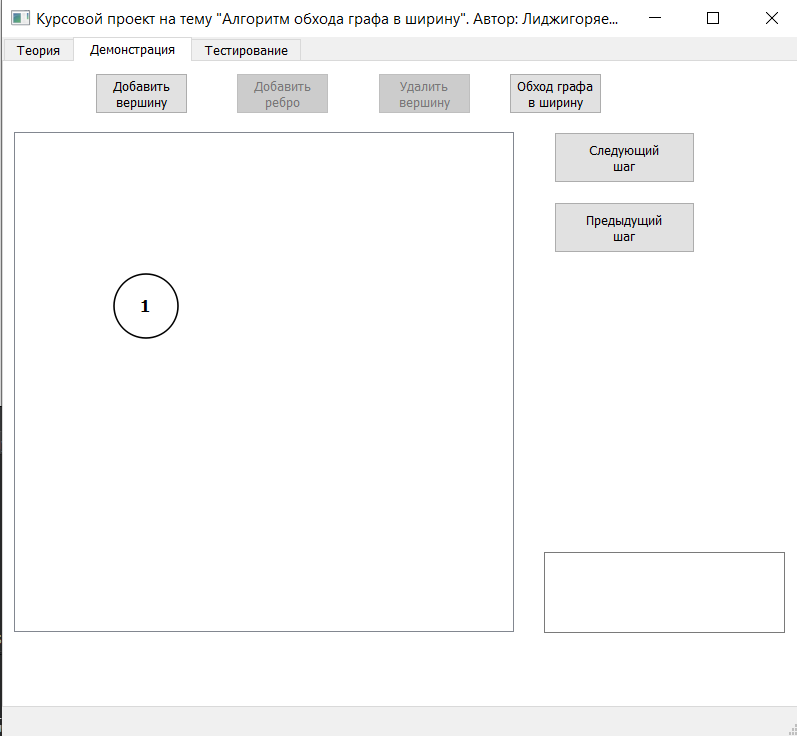


Рисунок 2.7 – Добавленная вершина

При повторном нажатии добавится еще одна вершина (рис. 2.8).

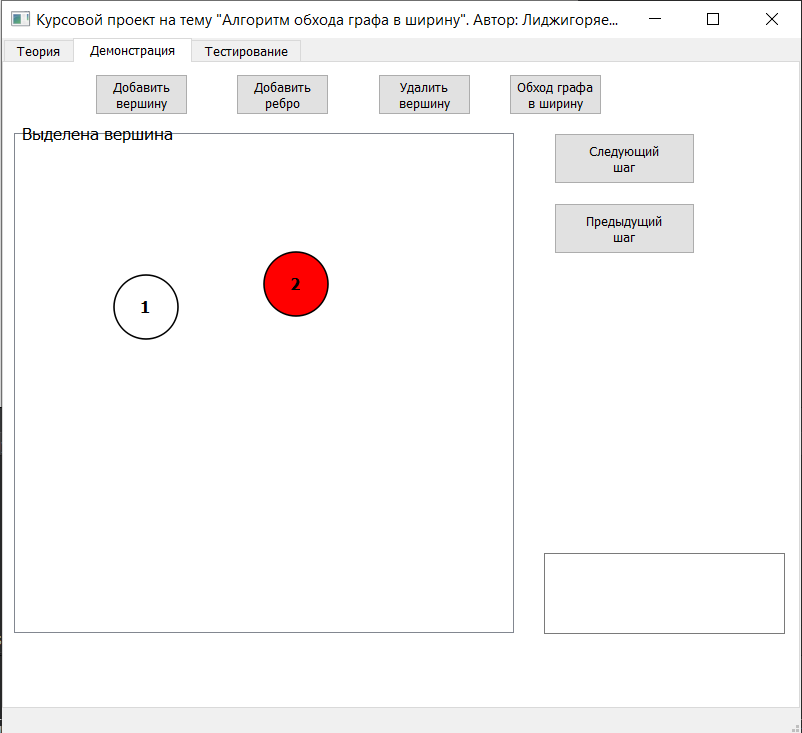


Рисунок 2.8 – Добавление второй вершины

Соединение двух вершин показано на рисунке 2.9.

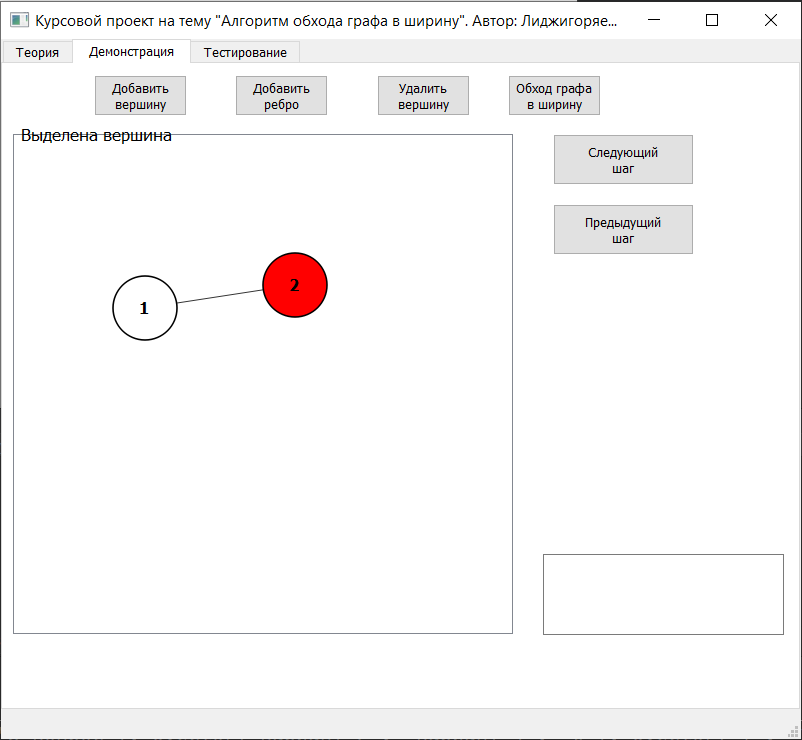


Рисунок 2.9 – Добавление ребра между 1 и 2 вершинами

При удалении вершины происходит удаление этой вершины и всех ребер, инцидентных ей с формы. На рисунке 2.10 показано удаление 2 вершины.

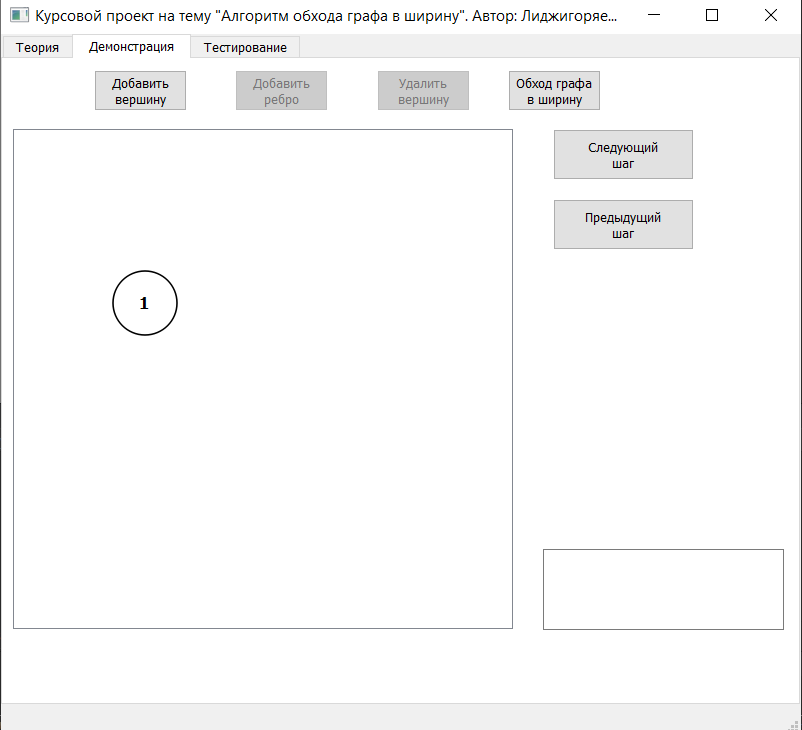


Рисунок 2.10 – Удаление вершины

На рисунке 2.11 показан граф, построенный для алгоритма обхода.

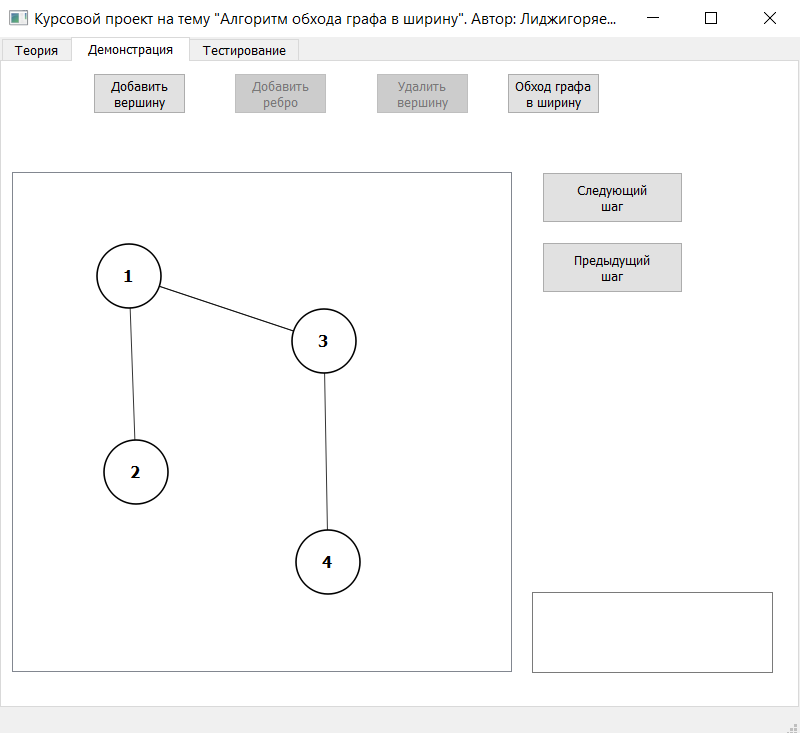


Рисунок 2.11 – Граф перед обходом

Алгоритм окрашивает вершины по мере обхода, белый цвет – вершина не пройдена, серый цвет – вершина рассматривается, черный – вершина пройдена.

На рисунке 2.12 показан первый шаг алгоритма для данного графа.

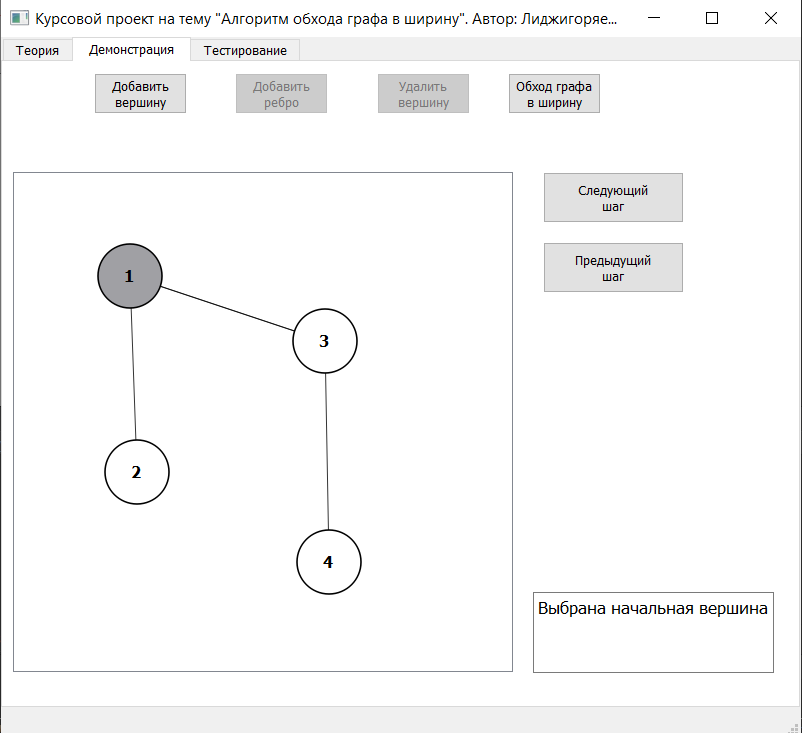


Рисунок 2.12 – 1 шаг алгоритма

На рисунке 2.13 показано состояние графа на последнем шаге алгоритма.

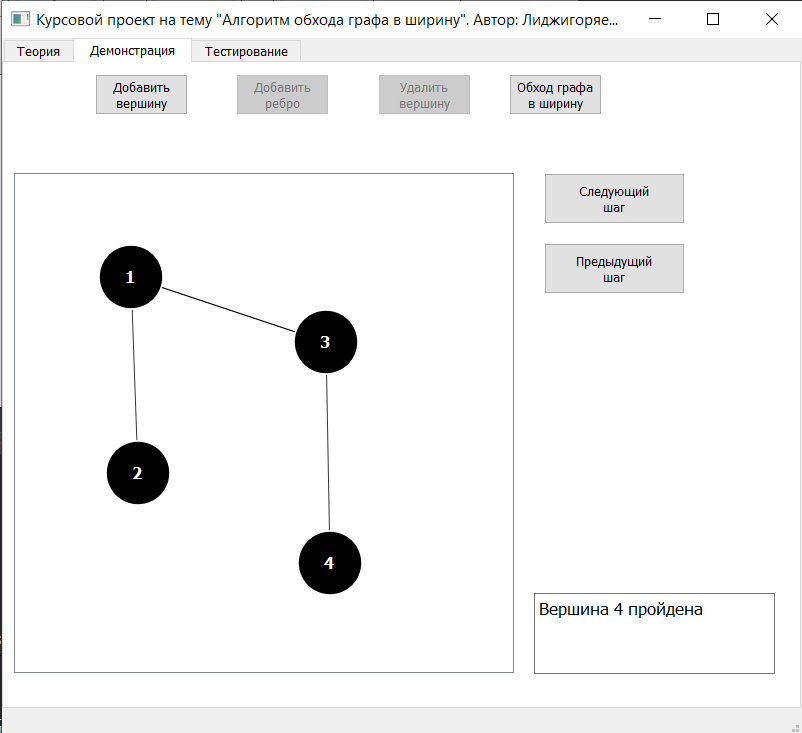


Рисунок 2.13 – Последний шаг алгоритма

При переходе на вкладку тестирования, появляется форма с вопросами, вариантами ответов или полем для ввода ответа (рис. 2.14)

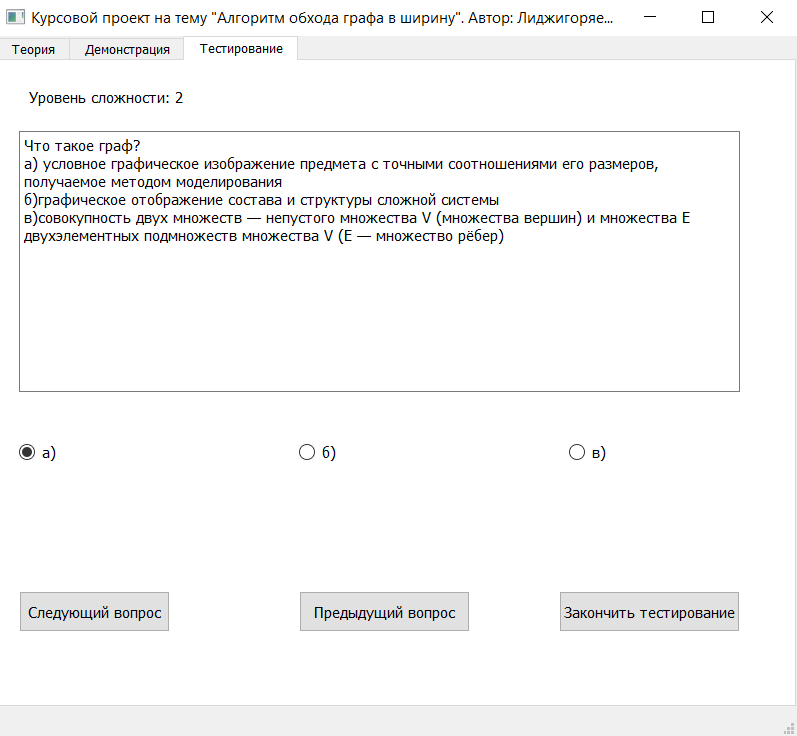


Рисунок 2.14 – Тестирование

При нажатии на “Следующий вопрос” произойдет переход на следующий вопрос теста (рис. 2.15).

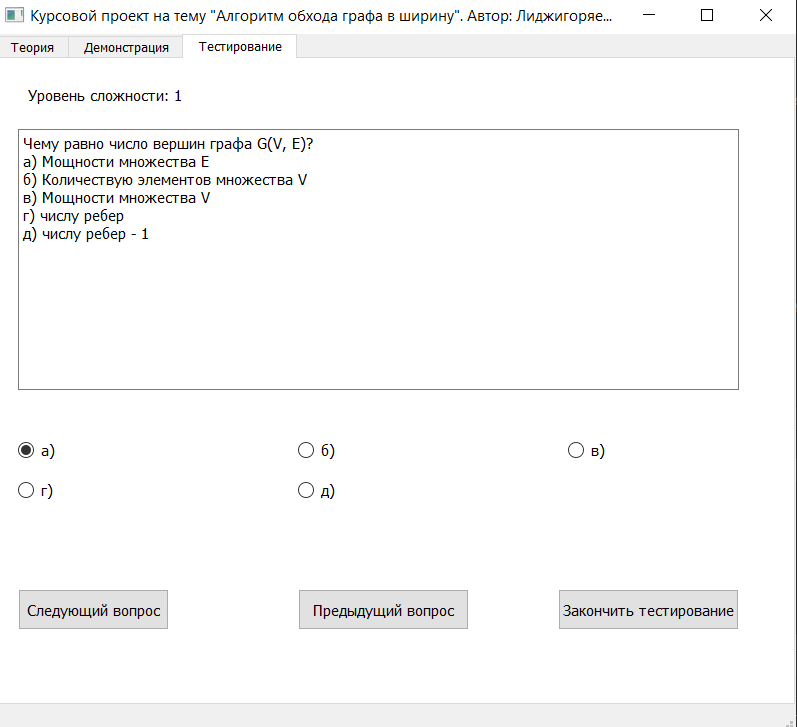


Рисунок 2.15 – Переход на следующий вопрос

При нажатии на предыдущий вопрос произойдет переход на предыдущий вопрос в тесте.

Закончить тестирование можно нажатием на кнопку “Закончить тестирование”. При нажатии на эту кнопку появится форма с приглашением для ввода своего имени (рис. 2.16).

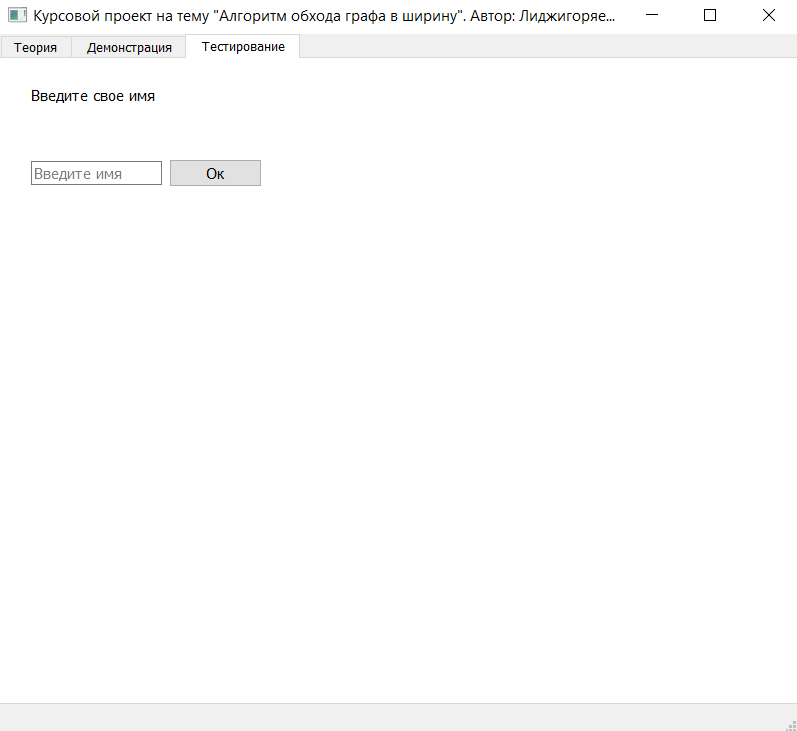


Рисунок 2.16 – Приглашение на ввод своего имени

После ввода имени и нажатии на кнопку “Ок”, происходит подсчет баллов за тестирование и вывод баллов, заработанных тестируемым и максимальное количество баллов, которое можно было заработать (рис. 2.17).

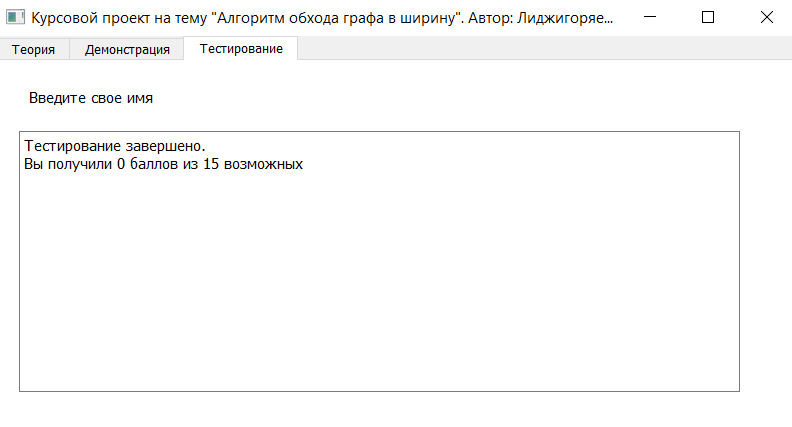


Рисунок 2.17 – Результат тестирования

## 2.6 Сообщения системы

В таблице 2.9 приведены сообщения системы.

Таблица 2.9 – Сообщения системы

|  |  |  |
| --- | --- | --- |
| **№ п\п** | **Сообщение** | **Причина возникновения,  способ устранения** |
| 1 | Вершина добавлена. | Добавлена вершина на форму |
| 2 | Вершина выделена. | Вершина выделена нажатием мыши. |
| 3 | Нажмите на вершину, к которой будет проведена дуга | Предлагается выделить вторую вершину для добавления ребра |
| 4 | Нажмите на вершину, с которой будет начинаться обход | Предлагается выделить вершину, которая будет начальной в алгоритме обхода |
| 5 | Тестирование завершено. Вы получили () баллов из () возможных | Тестирование завершено. Показываются результаты тестирования. |

В случае появления других сообщений следует обратиться к разработчику. **3 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ**

## 3.1 Проверка работоспособности показа теории

1. Запустить программу на выполнение. Появится форма с теорией (см. рис. 2.2).
2. Нажать на кнопку “назад”, убедиться, что страница теории не поменялась.
3. Нажать на кнопку “вперед”, убедиться, что выведена вторая страница теории (см. рис. 2.3).
4. Нажать на кнопку “вперед”, убедиться, что выведена четвертая страница теории.
5. Нажать на кнопку “вперед” еще раз, убедиться, что страница теории не изменилась.

## 3.2 Проверка работоспособности демонстрации работы алгоритма

1. Запустить программу на выполнение, нажать на вкладку демонстрации и убедиться, что открылась форма демонстрации (см. рис. 2.6).
2. Добавить вершину, убедиться, что появилось сообщение 1 (см. табл. 2.6).
3. Выделить эту вершину, убедиться, что выведено сообщение 2 (см. табл. 2.6) и нажать на кнопку удаления вершины, убедиться, что вершина пропала с формы.
4. Создать две вершины и соединить их ребром, убедиться, что получено сообщение 3 (см. табл. 2.6) при выделение первое вершины в соединении.
5. Создать две вершины и соединить их ребром, добавить третью вершину и соединить ее со второй вершиной, удалить первую вершину, убедиться, что первая вершина удалена, а также ребро, которое было между 1 и 2 вершинами.
6. Создать граф, как на рис. 2.11, нажать на кнопку обхода графа, выделить 4 вершину.
7. Нажать на кнопку следующего шага, убедиться, что 4 вершина стала серой.
8. Нажать на кнопку следующего шага, убедиться, что 4 вершина стала черной.
9. Нажать на кнопку следующего шага, убедиться, что 3 вершина стала серой.
10. Нажать на кнопку следующего шага, убедиться, что 3 вершина стала черной.
11. Нажать на кнопку следующего шага, убедиться, что 1 вершина стала серой.
12. Нажать на кнопку следующего шага, убедиться, что 1 вершина стала черной.
13. Нажать на кнопку следующего шага, убедиться, что 2 вершина стала серой.
14. Нажать на кнопку следующего шага, убедиться, что 3 вершина стала черной.

## 3.3 Проверка работоспособности тестирования

1. Запустить программу на выполнение, нажать на вкладку тестирования, убедиться, что открылся тест (см. рис. 2.14).
2. Убедиться, что при нажатии перехода на предыдущий вопрос, ничего не происходит.
3. Убедиться, что при нажатии перехода на следующий вопрос, происходит переход на следующий вопрос.
4. Нажимать на кнопку следующего вопроса, пока вопрос меняться не будет, убедиться что при нажатии кнопки, следующий вопрос действительно не загружается.
5. Перейти к первому вопросу и пройти тестирование, по прохождении нажать на кнопку “Закончить тестирование”, убедиться, что предлагается ввести имя (см. рис. 2.16).
6. Ввести имя и убедиться, что на форму выведены результаты тестирования, аналогично выводу на рис. 2.17.

# ЗАКЛЮЧЕНИЕ

В результате курсового проектирования разработана учебно-демонстрационная программа «Алгоритм обхода графа в ширину». Программа может выводить теоретический материал на экран, предлагает пользователю создать собственный граф и демонстрировать работу алгоритма обхода графа в ширину на нем и предлагает тестовые задания на темы, связанные с графами, алгоритмом обхода графа в ширину и проверяет правильность ответов. По окончании тестирования выводится результат тестирования: насколько успешно были выполнены задания.

Программа отвечает поставленным требованиям и может быть использована для обучения студентов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Новиков Ф.А. Дискретная математика для программистов: Учебник для вузов, 3-е изд. – СПб.: Питер, 2009. – 384 с.: ил. – (Серия “Учебник для вузов”)
2. Микони С.В. Дискретная математика для бакалавра: множества, отношения, функции, графы: Учебное пособие. – СПб.: Издательство “Лань”, 2012. – 192 с.: ил.
3. Кормен, Томас Х., Лейзерсон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд. Алгоритмы: построение и анализ, 2-е издание. : Пер. с англ. – М.: Издательский дом “Вильямс”, 2005. – 1296 с.: ил.
4. Стивенс, Род. Алгоритмы. Теория и практическое применение / Род Стивенс. – Москва : Издательство “Э”, 2016. – 544 с.
5. Шлее М. Qt 5.10. Профессиональное программирование на С++. – СПб.: -БХВ-Петербург, 2018. – 1072 с.: ил. – (В подлиннике)
6. Лаптев В.В. С++. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2008. – 464 с.: ил. – (Серия “Учебное пособие”).
7. Вайсфельд Мэтт. Объектно-ориентированный подход. 5-е межд. изд. – СПб.: Питер, 2020. – 256 с.: ил. – (Серия “Библиотека программиста”).

# ПРИЛОЖЕНИЕ 1

**Диаграммы классов**

# Диаграмма классов подсистемы теории

На рисунке П1 представлена диаграмма классов, связанных с теоретическим материалом.

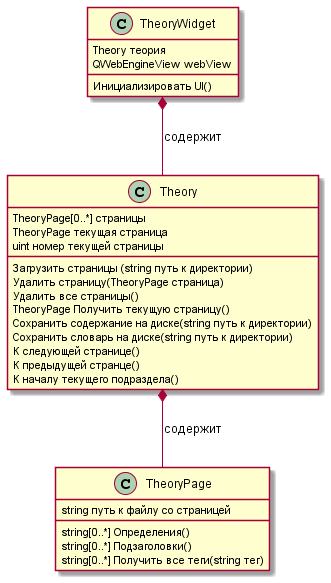


Рисунок П1 – диаграмма классов для теоретического материала

# Диаграмма классов подсистемы демонстрации

На рисунке П2 представлена диаграмма классов, связанных с демонстрацией алгоритма обхода графа в ширину.

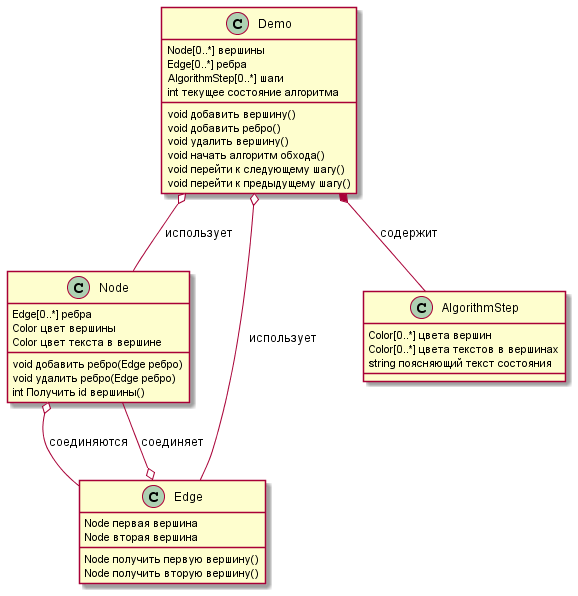


Рисунок П2 – диаграмма классов, реализующие работу демонстрационного процесса

# Диаграмма классов подсистемы тестирования

На рисунке П3 представлена диаграмма классов, связанных с тестированием пользователя.

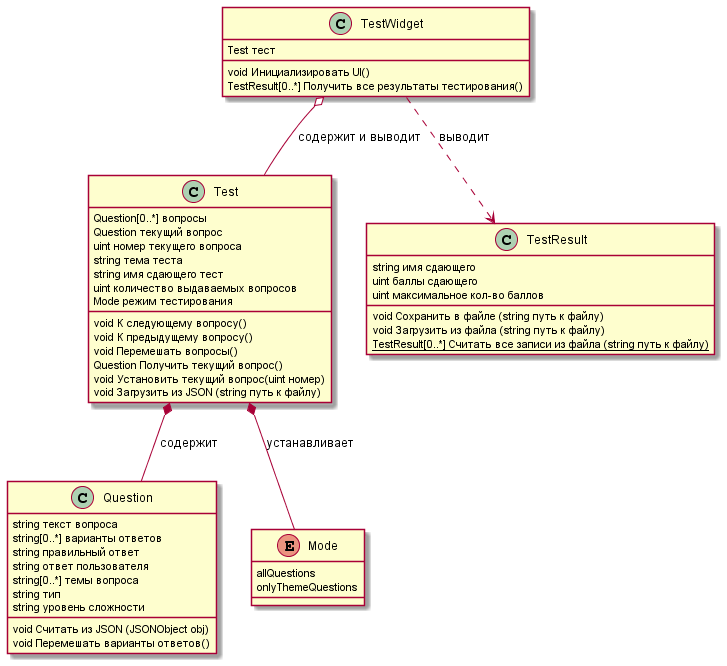


Рисунок П3 – диаграмма классов, реализующие тестирование пользователя на знание темы

# Диаграмма основных классов

На рисунке П4 представлена диаграмма классов, представляющие отображения 3 основных подсистем и главное окно.

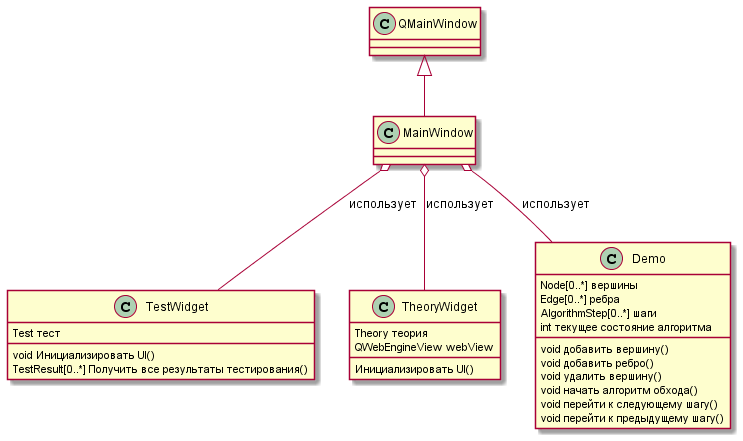


Рисунок П4 – диаграмма основных классов