Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

## ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовому проектированию

По курсу «Логика и основы алгоритмизации

в инженерных задачах»

На тему «Реализация алгоритма Дейкстры»

Выполнил:

Студент 2 курса группы 19ВВ1:

Артемов К.А.

Приняли:

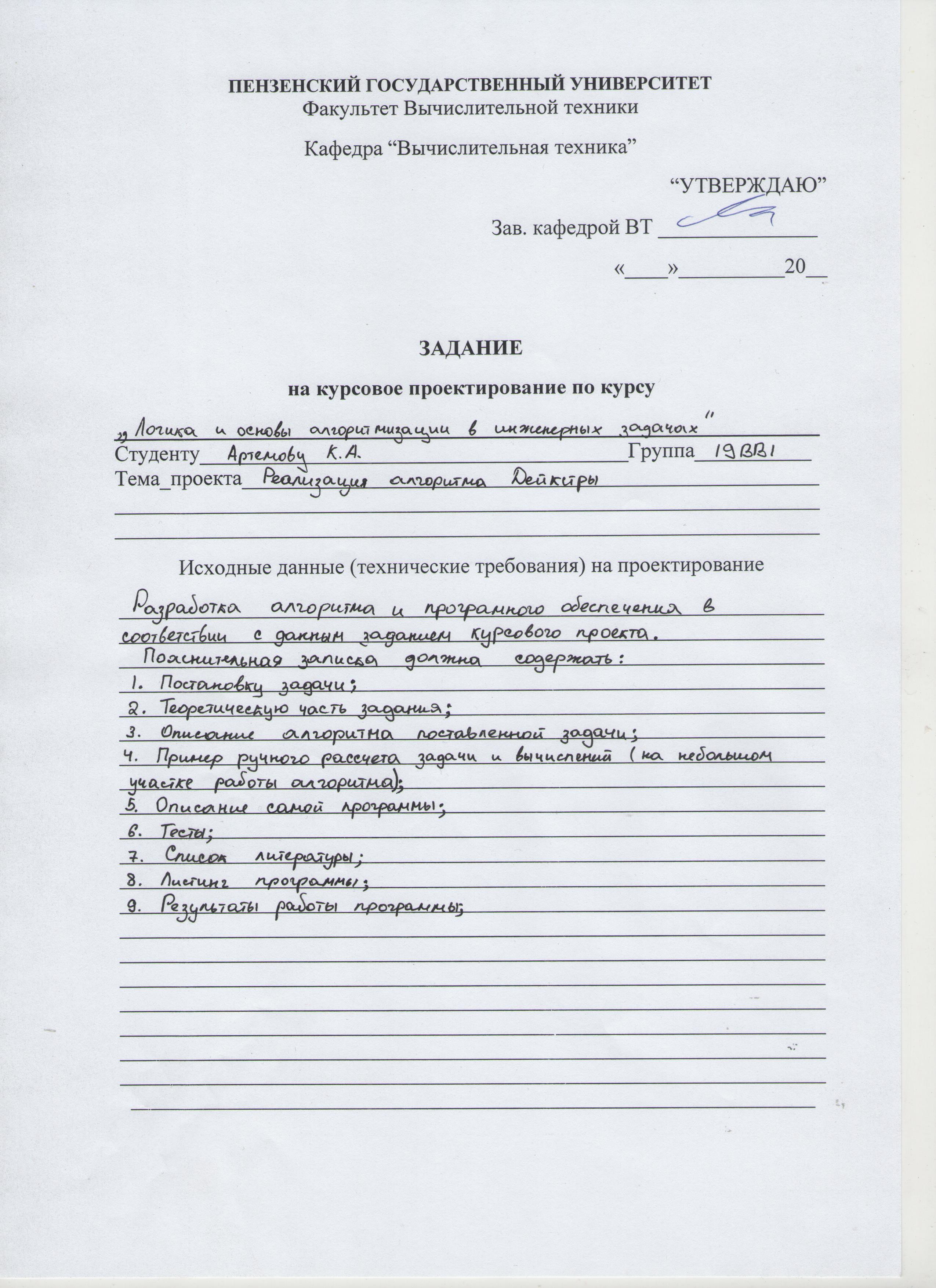
д.т.н. профессор

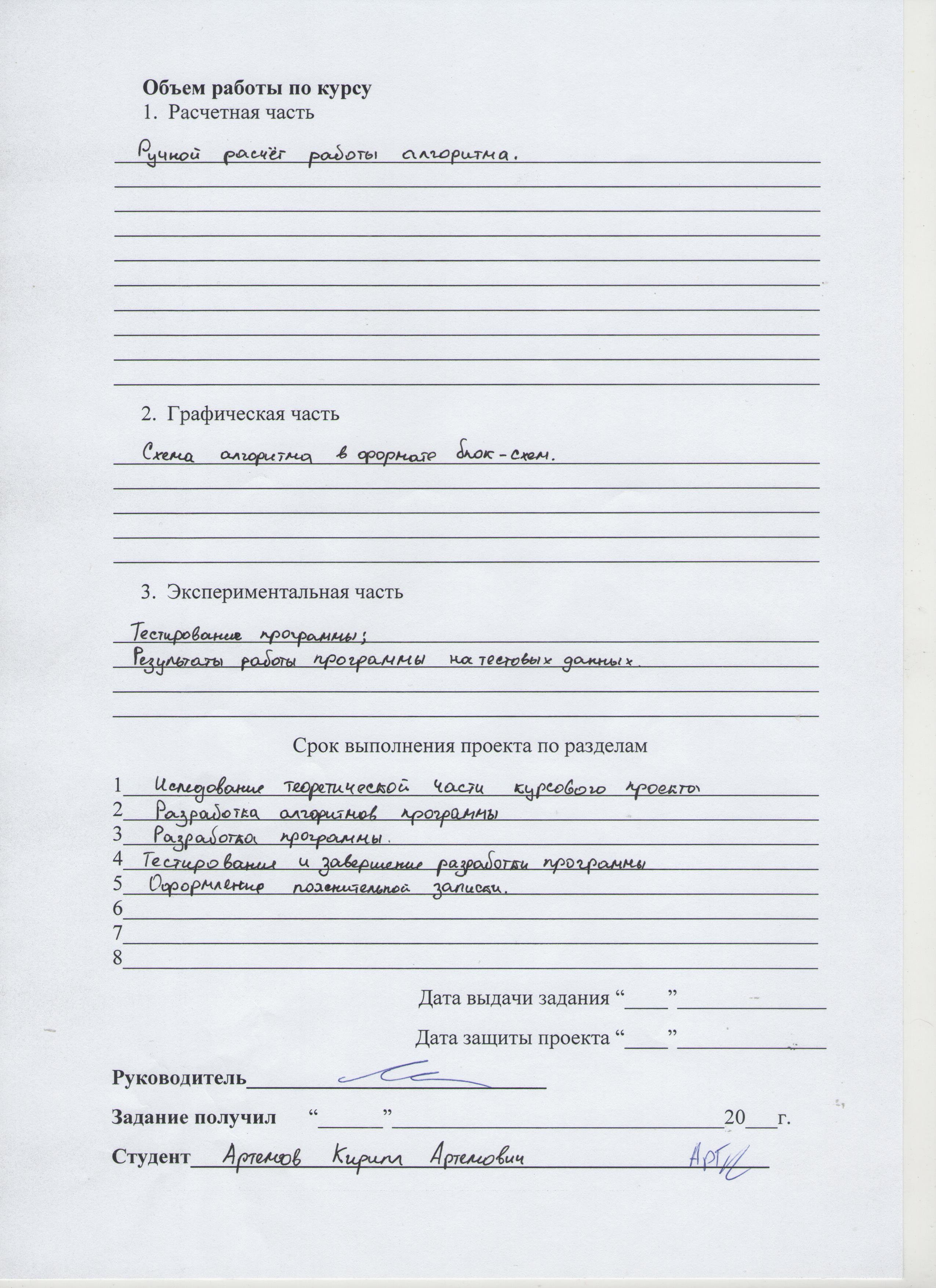
Митрохин М.А.

к.т.н. доцент

Юрова О.В.

Пенза 2020





Оглавление

[ПОЯСНИТЕЛЬНАЯ ЗАПИСКА 1](#_Toc59193007)

[Реферат 5](#_Toc59193008)

[Введение 6](#_Toc59193009)

[1. Постановка задачи 7](#_Toc59193010)

[2. Теоретическое часть программы 8](#_Toc59193011)

[2.1 Общие теоретические сведения 8](#_Toc59193012)

[3. Описание алгоритма программы 9](#_Toc59193013)

[4. Описание программы 11](#_Toc59193014)

[5. Тестирование 24](#_Toc59193015)

[6. Ручной расчёт задачи 28](#_Toc59193016)

[Заключение 34](#_Toc59193017)

[Список литературы 35](#_Toc59193018)

[Приложение А. Листинг программы. 36](#_Toc59193019)

## Реферат

Отчет 49стр, 39 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ДОСТИЖИМОСТЬ, АЛГОРИТМ ДЕЙКСТРЫ.

Цель исследования – разработка программы, способная находить кратчайшие расстояния до всех вершин графа, использую алгоритм Дейкстры.

В работе рассмотрен алгоритм Дейкстры, на основе которого находятся кратчайшие расстояния до всех вершин графа.

## Введение

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, его используют протоколы маршрутизации OSPF и IS-IS. Данный алгоритм находит очень широкое применение в практической деятельности. Он очень подходит для работы с системами, которые имеют множество связанных между собой точек – вершин.

Эдсгер Дейкстра Родился 11 мая 1930 года в [Роттердаме](https://ru.wikipedia.org/wiki/%D0%A0%D0%BE%D1%82%D1%82%D0%B5%D1%80%D0%B4%D0%B0%D0%BC), в семье учёных (отец — [химик](https://ru.wikipedia.org/wiki/%D0%A5%D0%B8%D0%BC%D0%B8%D0%BA), мать — [математик](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA)).

В 1951 году увлёкся [программированием](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), поступил на трёхнедельные [компьютерные](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) курсы в [Кембридже](https://ru.wikipedia.org/wiki/%D0%9A%D0%B5%D0%BC%D0%B1%D1%80%D0%B8%D0%B4%D0%B6). Изобрёл новое правило компиляции — «вызов по имени».

В 1960-е годы участвовал в создании [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [THE](https://ru.wikipedia.org/w/index.php?title=THE_(%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)&action=edit&redlink=1) построенной в виде множества параллельно исполняющихся взаимодействующих процессов. Именно в ходе этой работы появились понятия [синхронизации процессов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), идея [семафора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), а также была чётко осознана необходимость в структуризации процесса программирования и самих программ.

Разработал основные положения [структурного программирования](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Мною была выбрана среда программирования «Microsoft Visual Studio 2019», язык программирования – С#.

В процессе выполнения работы мною были усовершенствованы навыки работы с формами и их элементами, а также с языком программирования С#

1. Постановка задачи

Требуется разработать программу, выполняющую поиск минимального пути от одной выбираемой вершины к другим, используя алгоритм Дейкстры.

Пользователь должен иметь возможность сам выбирать размер графа. В программе должна присутствовать возможность выбора автоматического или ручного задания графа. Программа должна иметь графический интерфейс. Необходимо реализовать сохранение результата в excel.

Устройства ввода и вывода информации: мышь, клавиатура, монитор. Программа должна быть разработана для работы в ОС Microsoft Windows.

Задания выполняются в соответствии с вариантом №13 (Реализация алгоритма Дейкстры).

1. Теоретическое часть программы

2.1 Общие теоретические сведения

Пользователь должен иметь возможность сам выбирать размер графа. В программе должна присутствовать возможность выбора автоматического или ручного задания графа. Программа должна иметь графический интерфейс. Необходимо реализовать сохранение результата в excel.

Устройства ввода и вывода информации: мышь, клавиатура, монитор. Программа должна быть разработана для работы в ОС Microsoft Windows.

Задания выполняются в соответствии с вариантом №13 (Реализация алгоритма Дейкстры).

В простейшей реализации для хранения чисел d[i] можно использовать массив чисел, а для хранения принадлежности элемента множеству U — массив булевых переменных.

В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бо́льшим максимального возможного пути в [графе](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))). Массив флагов заполняется нулями. Затем запускается основной цикл.

На каждом шаге цикла мы ищем вершину v с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины  ***u***. Если в них (в ***u***) расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его. Цикл завершается, когда флаги всех вершин становятся равны 1, либо когда у всех вершин c флагом 0  d[i]=∞. Последний случай возможен тогда и только тогда, когда граф G несвязный.

1. Описание алгоритма программы

1. Метка начальной вершины полагается равной 0, метки остальных вершин – недостижимо большое число (в идеале — бесконечность). Это отражает то, что расстояния от начальной вершины до других вершин пока неизвестны. Все вершины графа помечаются как не посещенные.

2.1 Минимальную метку имеет начальная вершина. Обходим соседей вершины по удаленности.

2.2 Длина пути в соседнюю вершину через начальную равна сумме кратчайшего расстояния до начальной (значению её метки) и длины ребра, идущего из начальной в соседнюю.

2.3 Аналогично находим длины пути для всех других соседей

2.4 начальная вершина становится посещенной

3.1 Снова находим «ближайшую» из не посещенных вершин.

3.2 Пока все вершины не посещены, выполняем следующее:{

Начальная посещена, поэтому ближайшая становится начальной.

Выполняем пункт 2.2, если найденное расстояние меньше чем метка, то присваиваем его.

Выполняем пункты 2.3 и 2.4

}

Ниже представлен псевдокод алгоритма

Вход: G – матрица смежности графа, v – исходная вершина.

Выход: DIST – вектор расстояний до всех вершин от исходной.

Алгоритм ПОШ

1.1. для всех i положим D [i] = 10000 и пометим как не посещенную D[v]=0;

1.2. ВЫПОЛНЯТЬ .

2.1. minindex = 10000 min = 10000

2.2. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.3. ЕСЛИ vis[i] = = 1 И D[i] < min

2.4. ТО min = d[i] minindex = i

2.5. ЕСЛИ minindex!=10000

2.6. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.7. ЕСЛИ G [minindex , i] > 0

2.8. ТО temp = min + G [minindex, i]

2.9. ЕСЛИ temp < D[i]

2.10. ТО D[i] = temp

2.11. vis[minindex] = 0

2.12. ЕСЛИ minindex < 10000 начать с 1.2

1. Описание программы

Для написания данной программы использован язык программирования С#.

С# — это современный язык программирования, созданный компанией Microsoft. На нём можно писать программы любой сложности для любых платформ и операционных систем.

Язык C# довольно простой, он типобезопасен и объектно-ориентирован. Благодаря огромному количеству внедрений, C# обеспечивает возможность разработки приложений с минимальной затратой времени, но при этом, он сохраняет выразительность и элегантность, которая присуща языкам C.

Проект был создан в виде приложения WindowsForm.

В каждой форме кнопки  и  выполняют возврат в меню и закрыть приложение соответственно.

Заставка

Работа программы начинается с заставки (Form1.cs) (рисунок 1)



Рис. 1- Заставка приложения.

Меню

Далее перед пользователем в дальнейшем user появляется меню (vvod.cs) (рисунок 2)

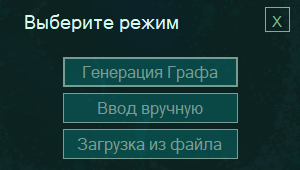


Рис. 2– Меню приложения.

Для User-а есть три варианта развития событий.

1. Использование рандомного заполнения графа;
2. Возможность ввести граф вручную;
3. И на конец загрузить данные в граф из таблицы excel;

Режим генерация графа

User-у предлагается вести размер графа (Size.cs) (рисунок3).

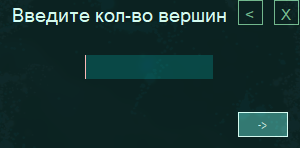


Рис. 3 – Размер графа.

User вводит размер и нажимает либо мышкой на  , либо на enter на клавиатуре.

Далее открывается основное окно для режима генерации (rand.cs)

(рисунок 4).

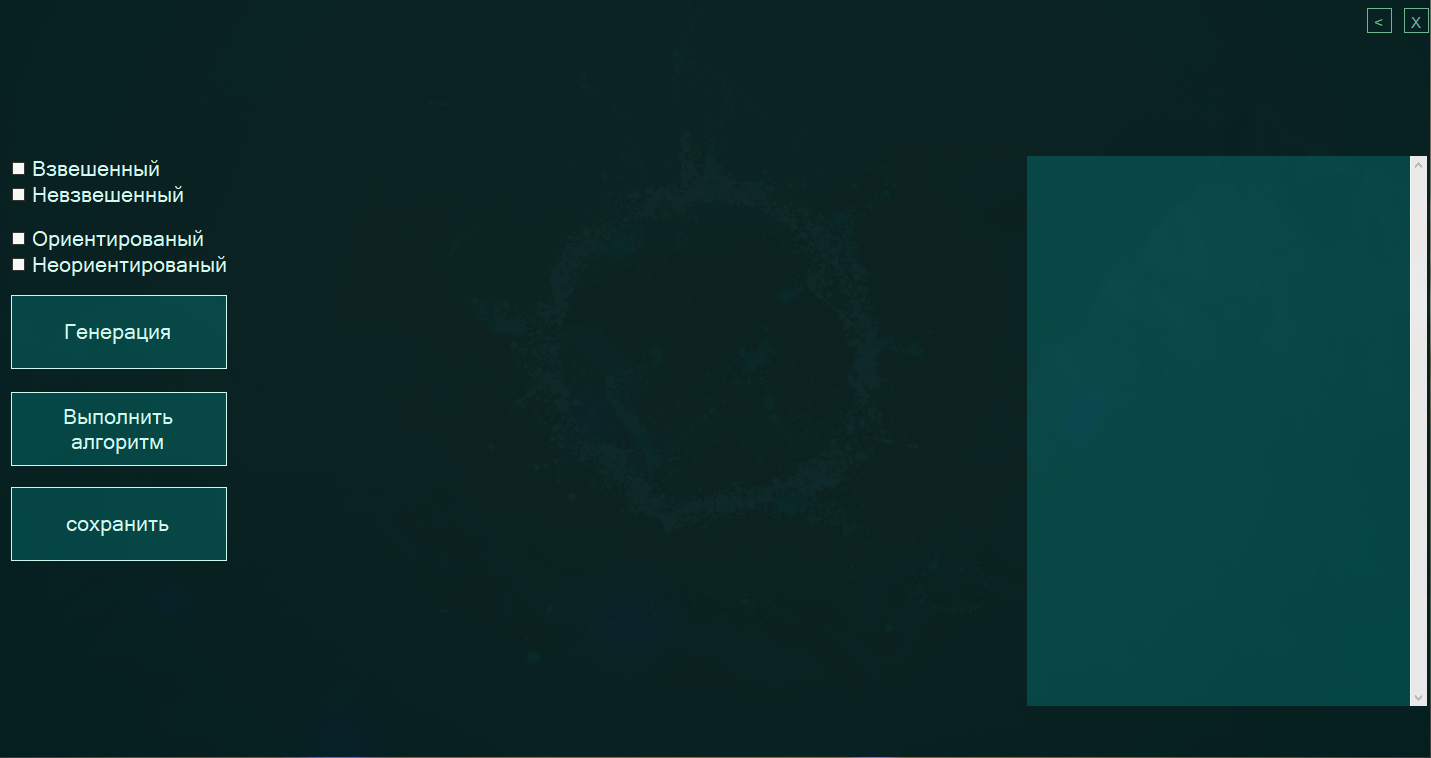


Рис. 4 - Режим генерации.

User может выбрать будут ли ребра графа иметь направление и вес. Далее он может сгенерировать граф (рисунок 5)

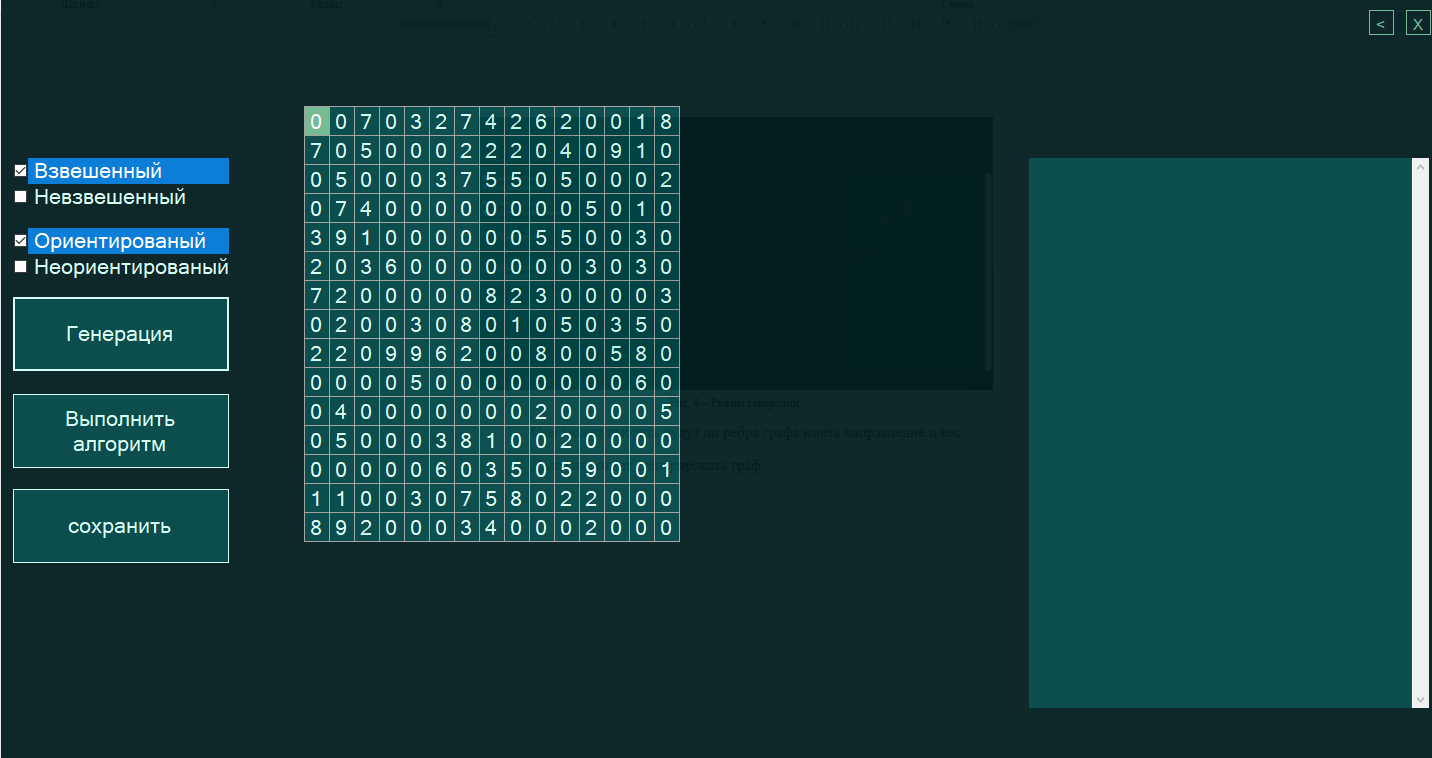


Рис. 5– Граф сгенерирован.

User может сохранить граф как без результата выполнения так и с ним. После генерации нажмите , дождитесь оповещения и граф сохранится без результата (рисунок 6).

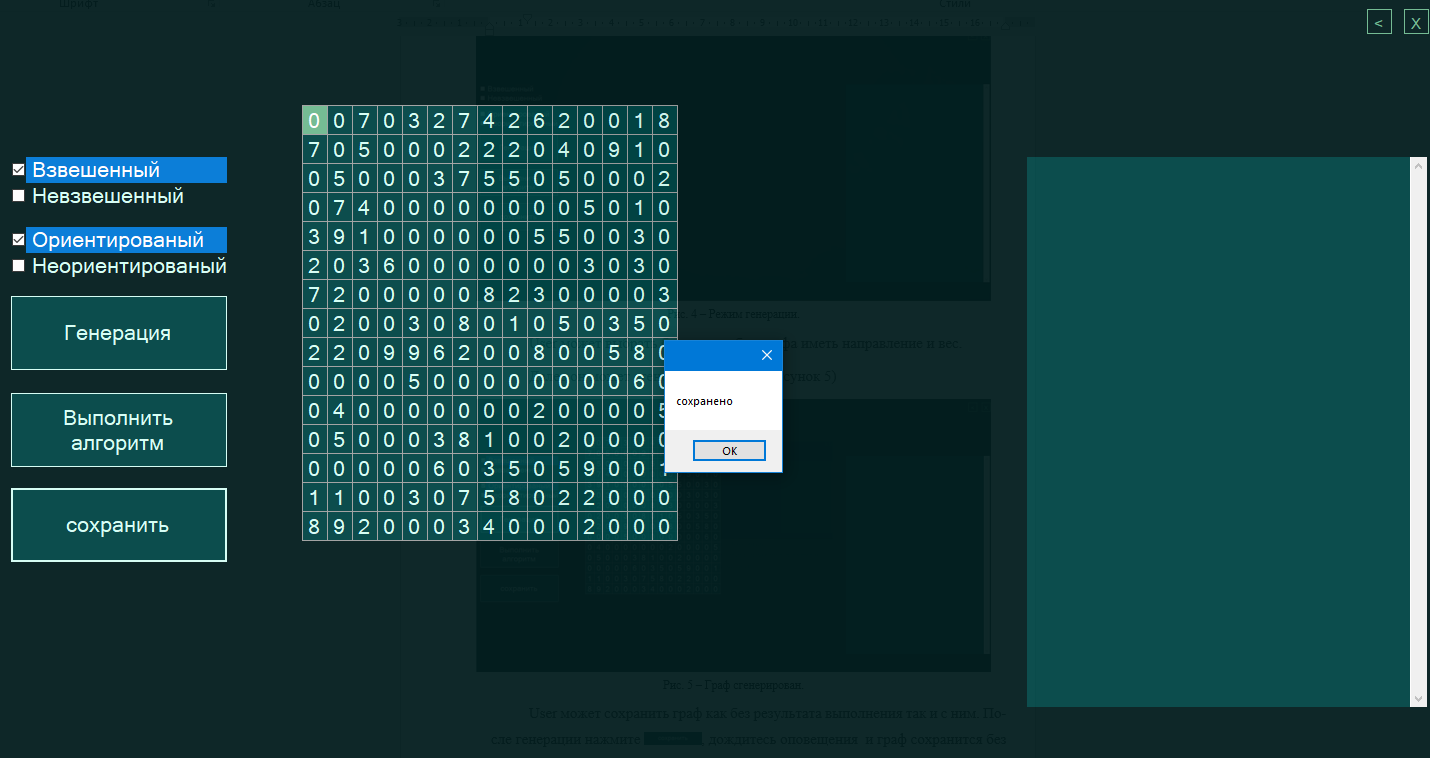


Рис. 6 - Сохранение графа.

Сохранение ведется в excel имя файла формируется как дата и время. Сохраняется в папку с приложением (рисунок7).

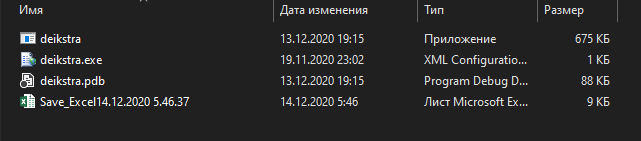


Рис. 7 – Каталог с сохранениями

В excel в ячейка А1 хранится размер графа. С ячейки B2 хранится матрица смежности графа (рисунок 8).

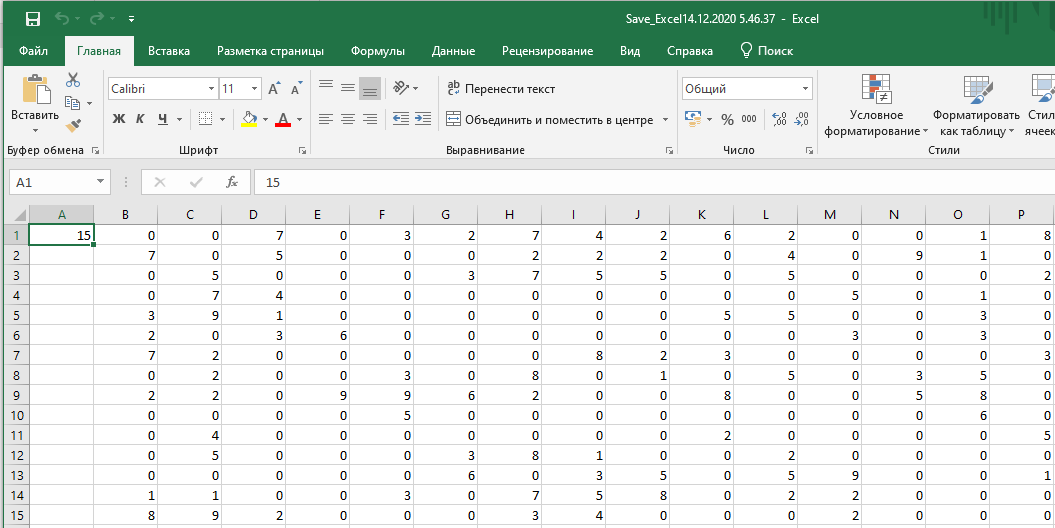


Рис. 8 – Результат сохранения графа без результата алгоритма

Для выполнения алгоритма user-у нужно нажать . Перед ним появится окно запрашивающее вершину для которой нужно найти кратчайшие расстояние до всех других вершин графа (nah.cs) (рисунок 9).

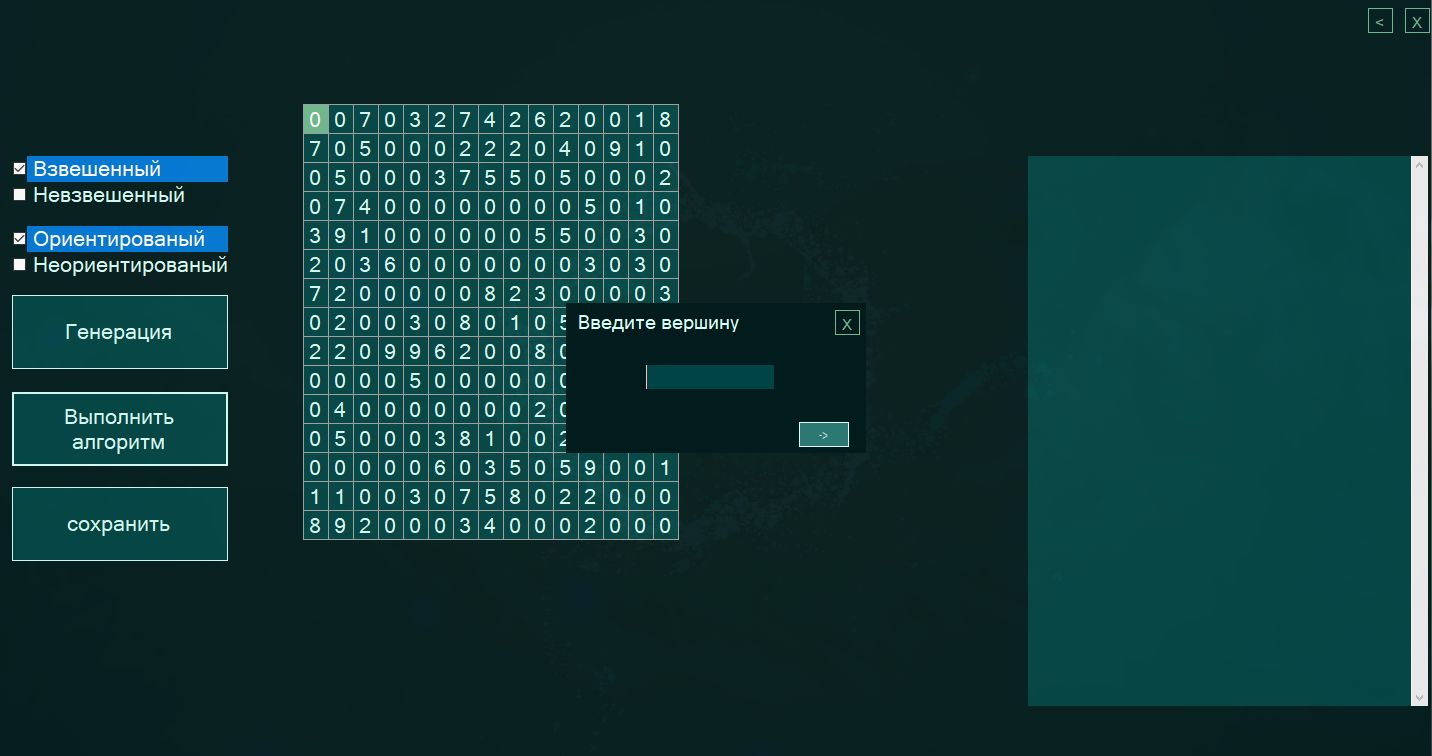


Рис. 9 - Начало обхода.

User вводит вершину и нажимает либо мышкой на  , либо на enter на клавиатуре (для удобства номера вершин начинаются с 1). Далее на основном окне появляется результат (рисунок 10).

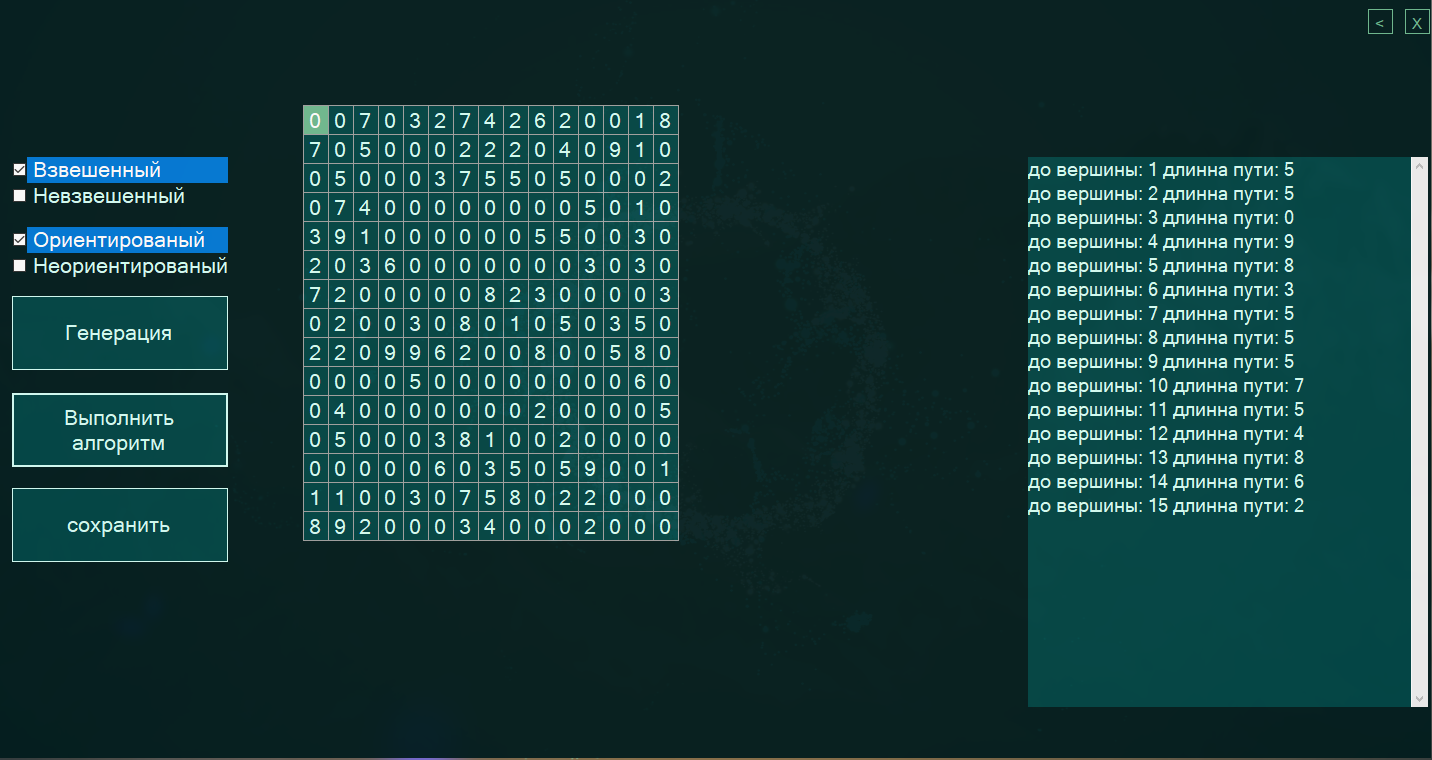


Рис. 10 – Результат выполнения алгоритма



Рис. 11 - Сохранение графа с результатом.

В excel в ячейка А1 хранится размер графа. С ячейки B2 хранится матрица смежности графа и с отступом в одну ячейку записывается результат выполнения алгоритма (рисунок 12).

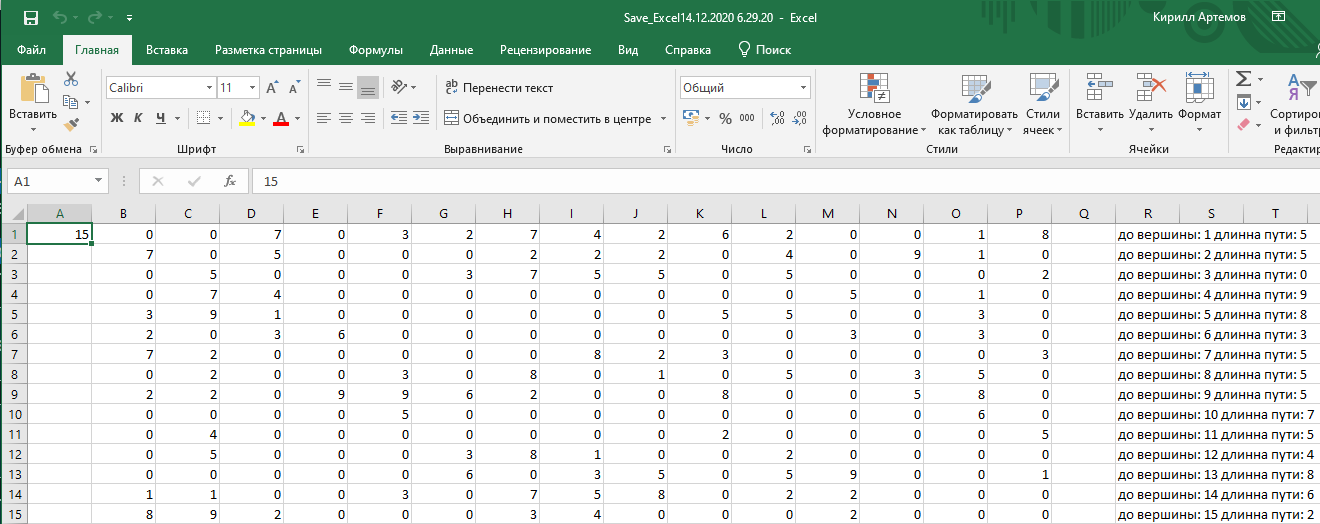


Рис. 12 - Результат сохранения графа с результатом алгоритма.

Режим ввод вручную

Так же, как и в первом режиме user-у предлагается ввести размер графа ([рисунок 3](#рисунок3))

Далее открывается основное окно для режима ввода вручную (vryh.cs)

(рисунок 13).

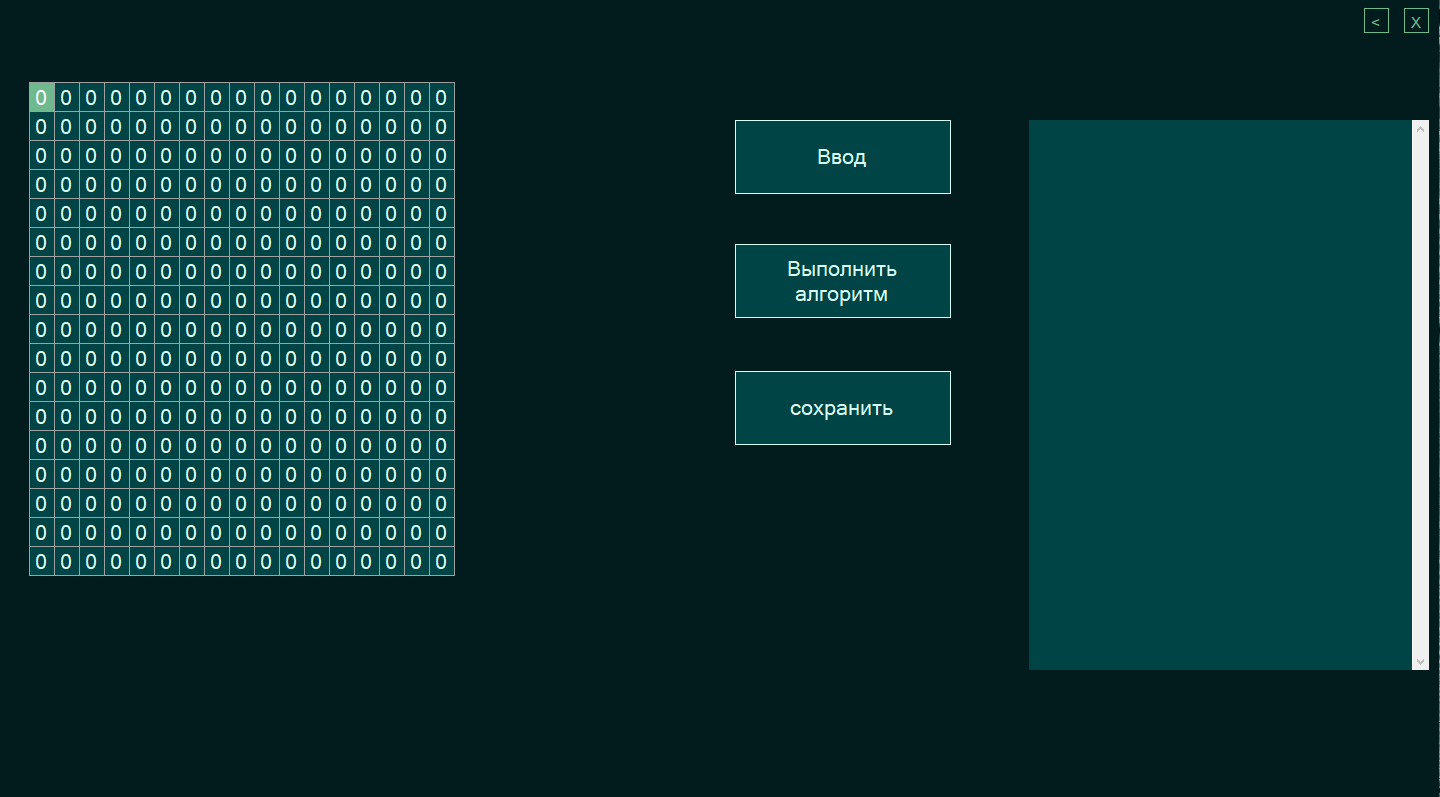


Рис. 13 – Режим ввода вручную

User заполняет матрицу смежности и нажимает (рсунок14).

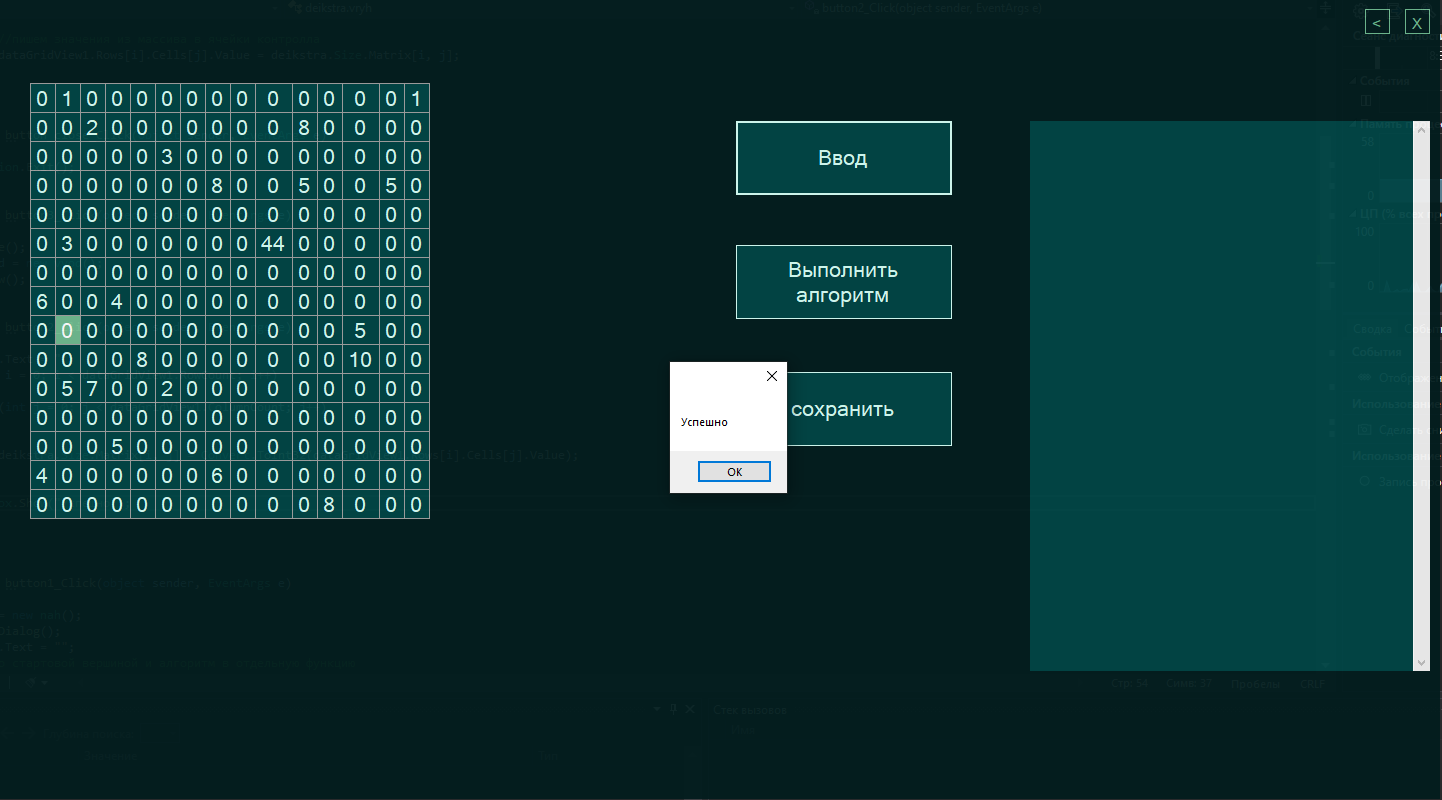


Рис. 14 – Введенная матрица

Далее все то же самое, что и в режиме генерации.

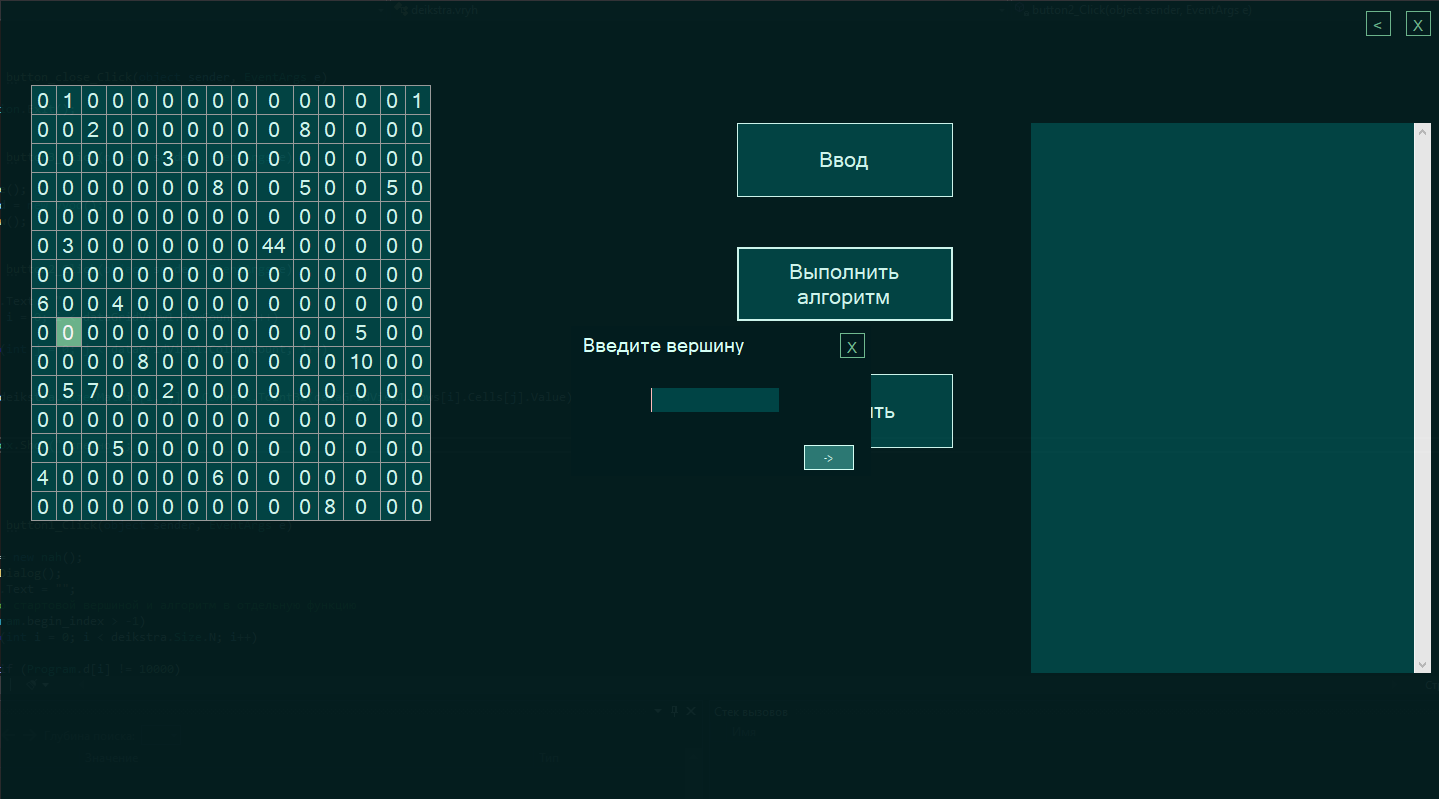


Рис. 15 - Начало обхода

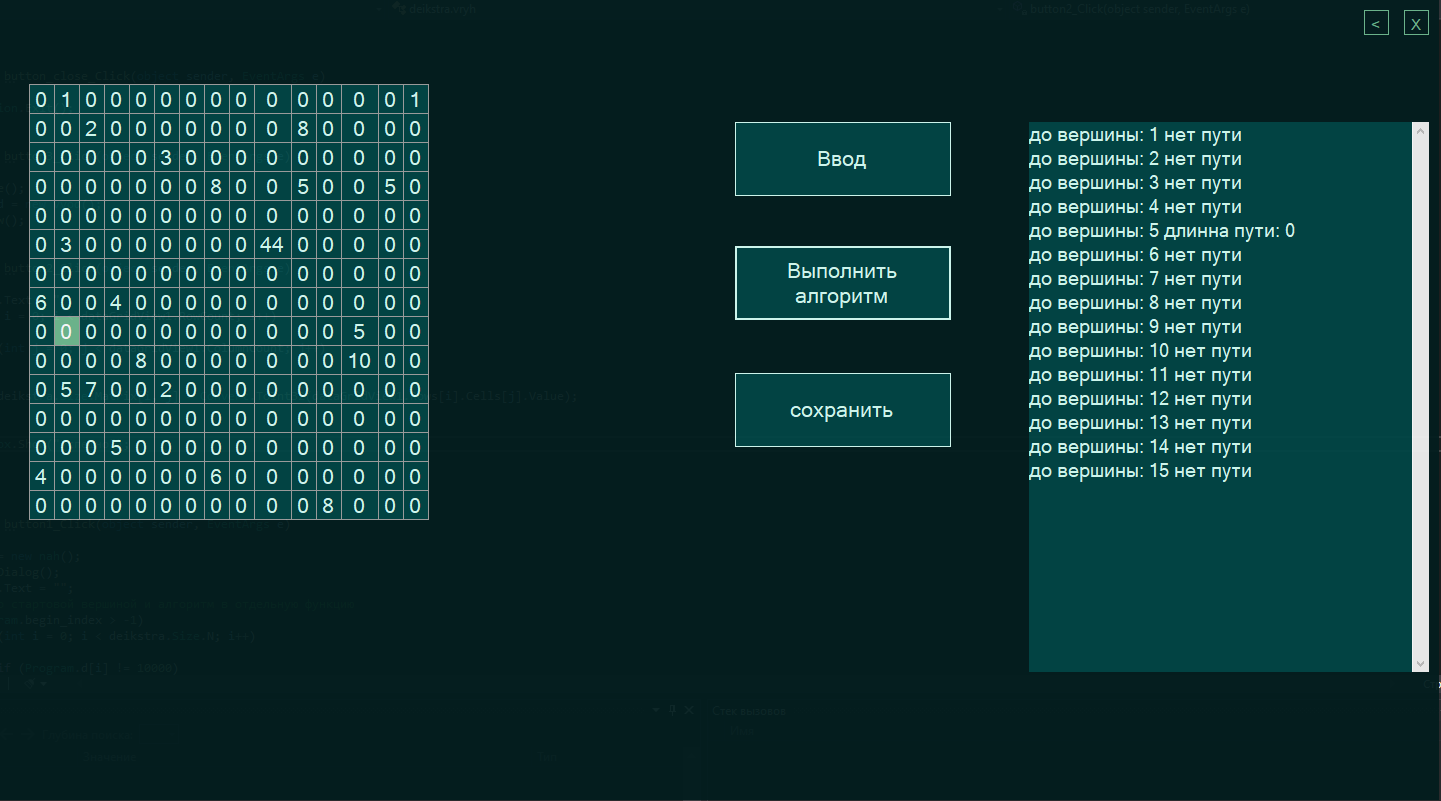


Рис. 16– Результат выполнения алгоритма

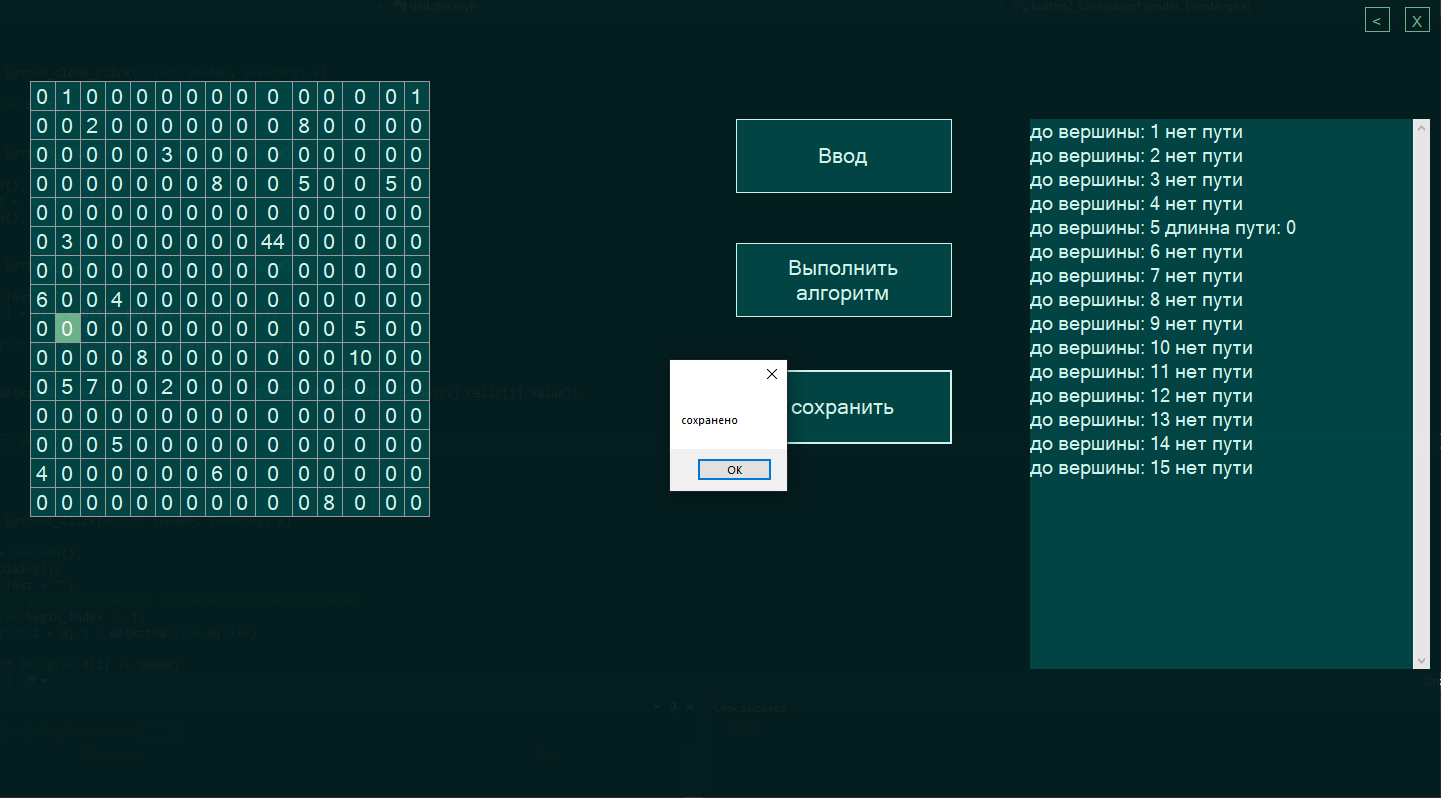


Рис. 17 - Сохранение графа с результатом



Рис. 18 - Результат сохранения графа с результатом алгоритма.

Режим загрузки из файла

Открывается основное окно для режима загрузки из файла (fail.cs)

(рисунок 19).

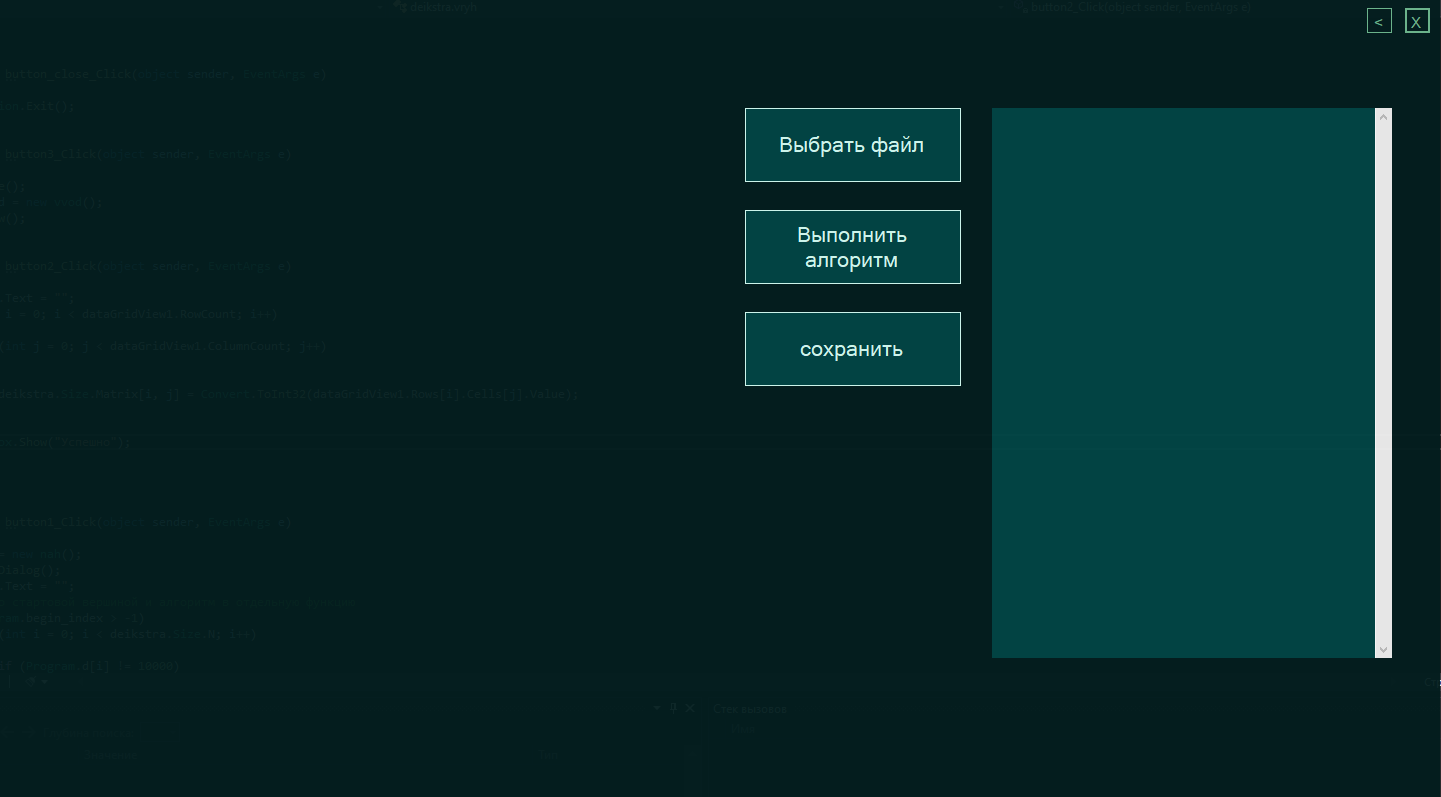


Рис. 19 – Режим загрузки из файла

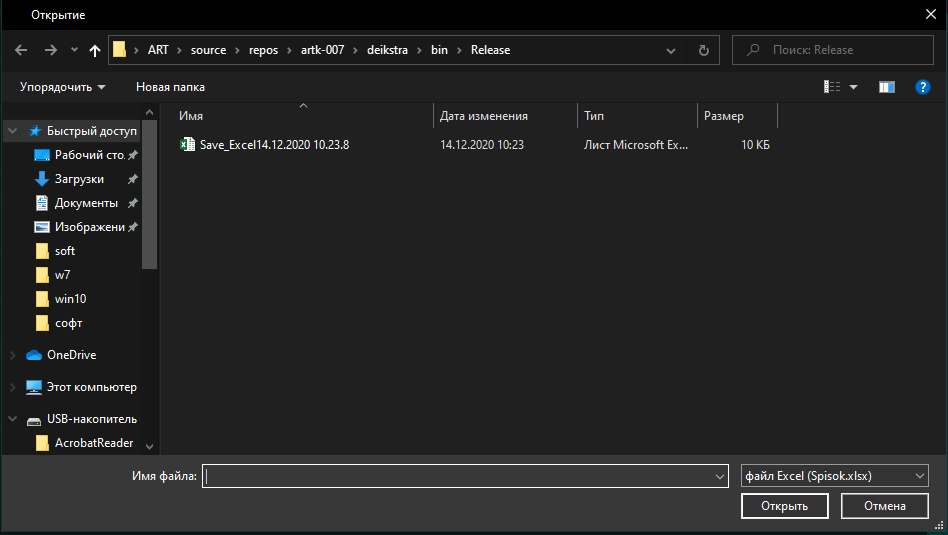
Нужно нажать на кнопку , затем откроется проводник (рисунок 20) 

Рис. 20 - Проводник

После выбора файла дождитесь загрузки и оповещения (рисунок 21)

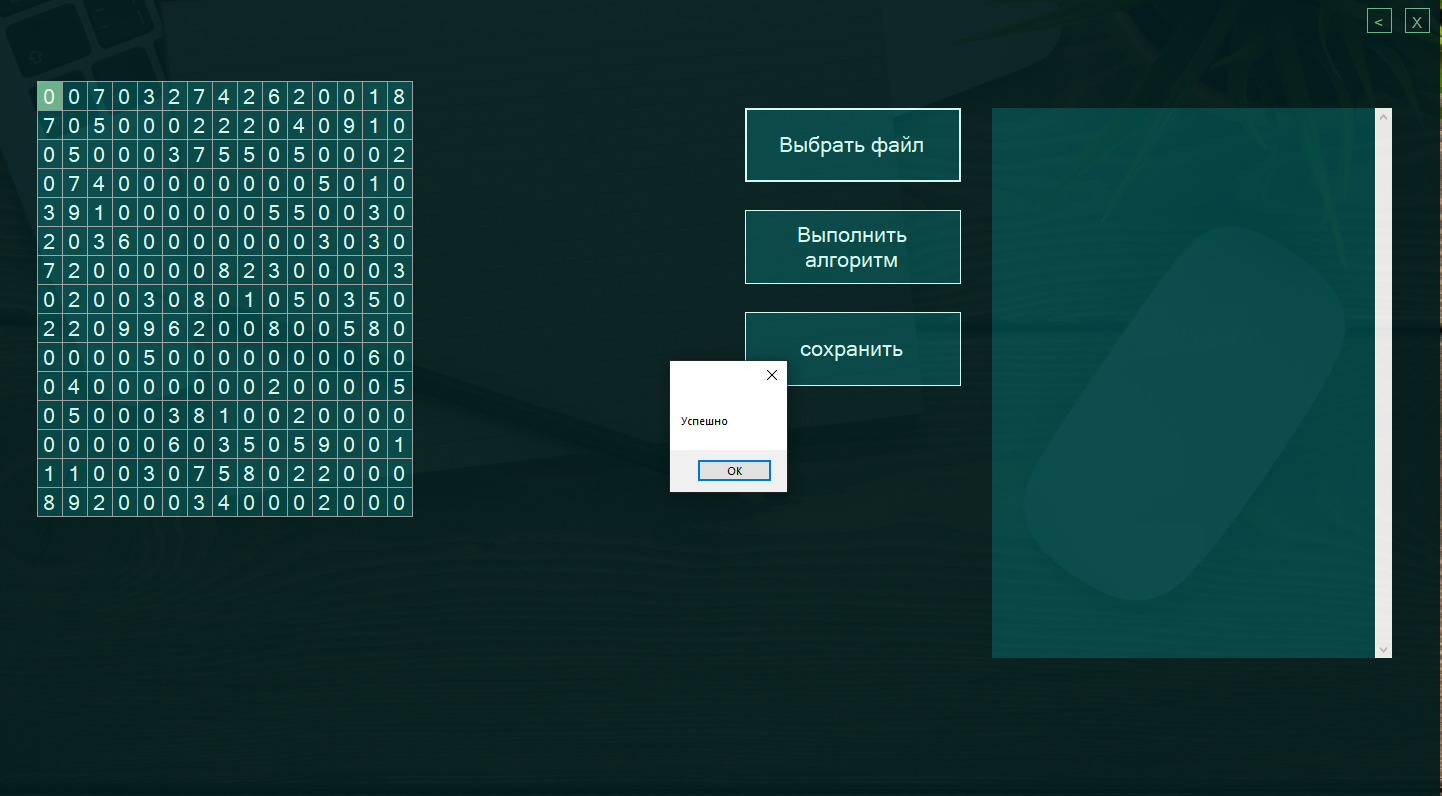


Рис. 21 – Загрузка файла

Далее всё то же самое, что и в генерации.

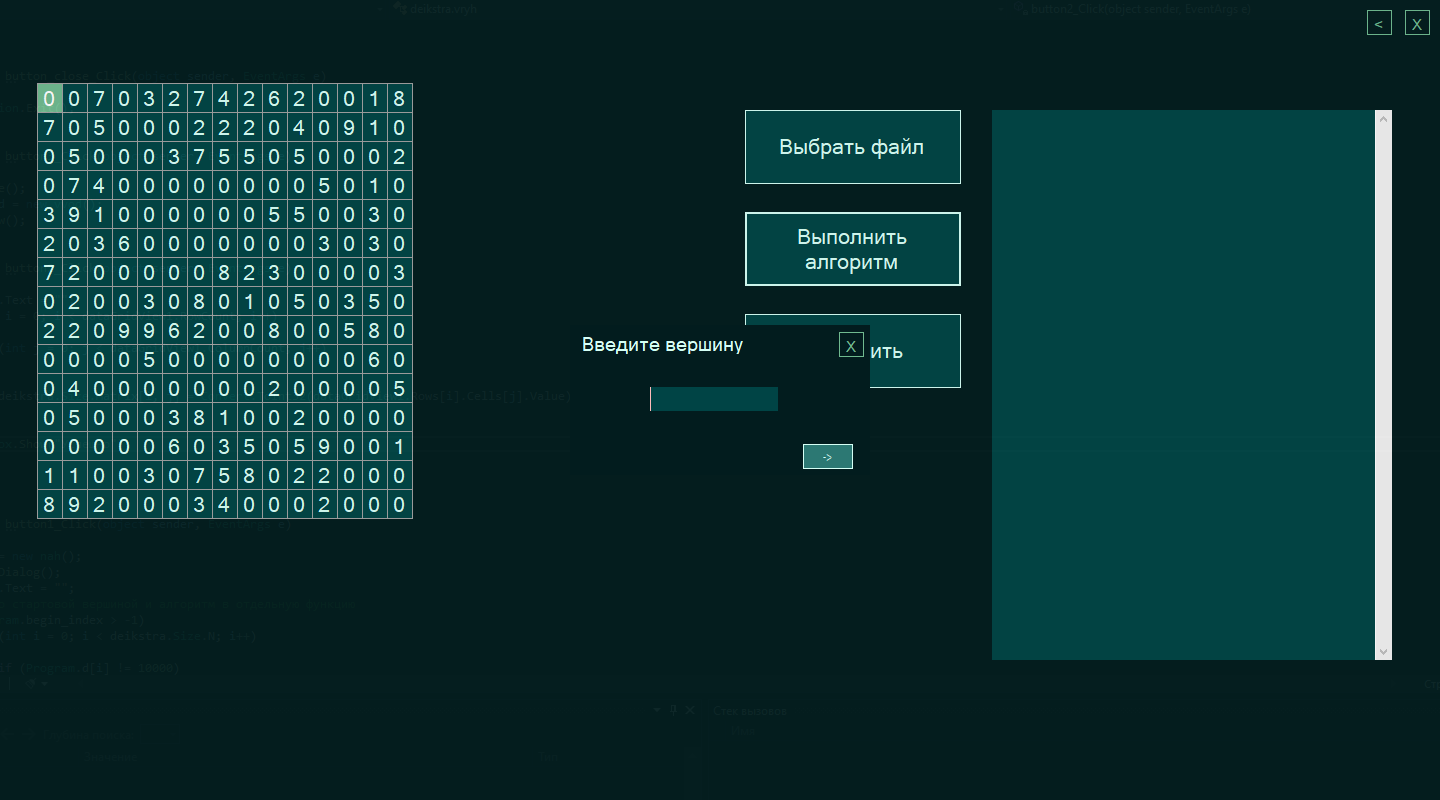


Рис. 22 - Начало обхода

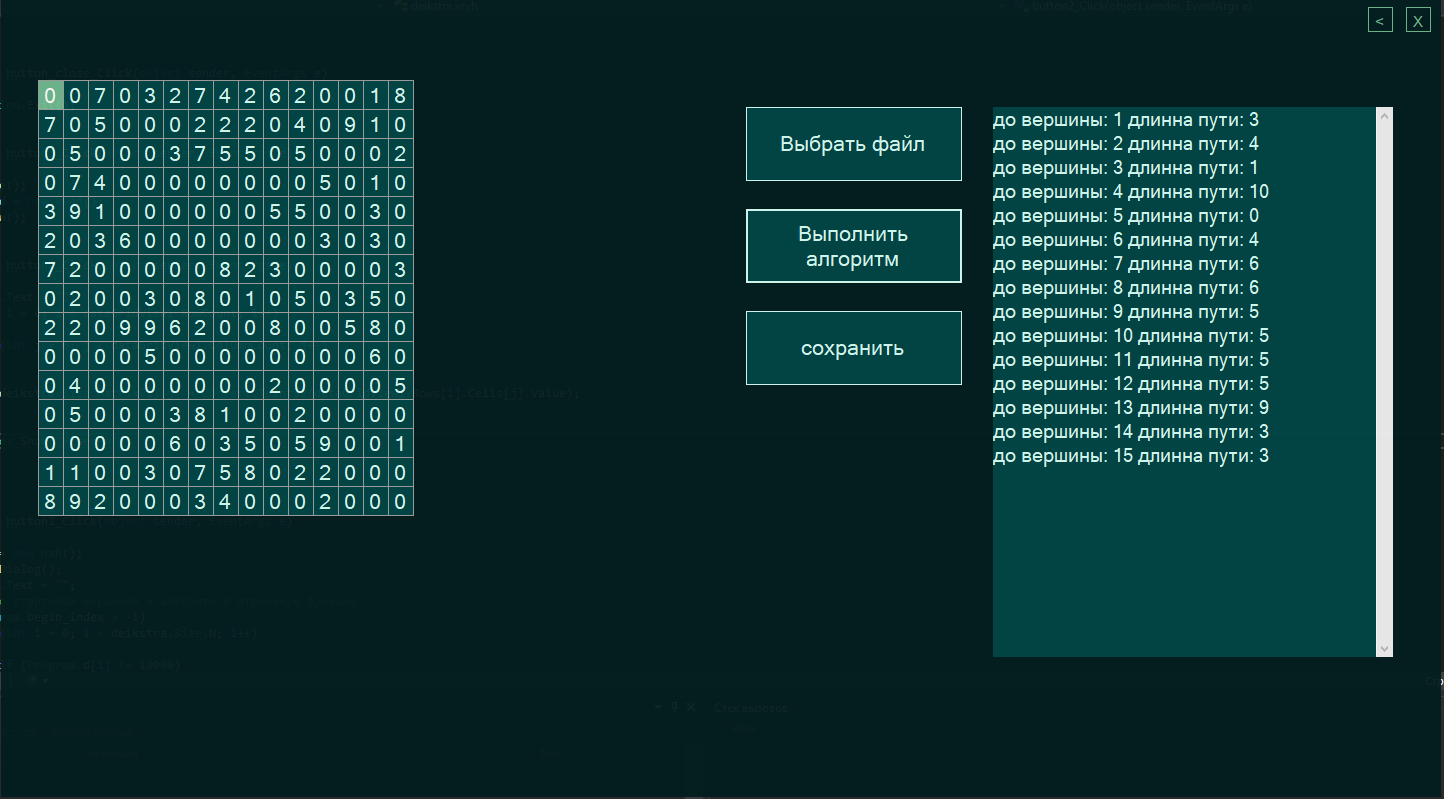


Рис. 23 - Результат выполнения алгоритма

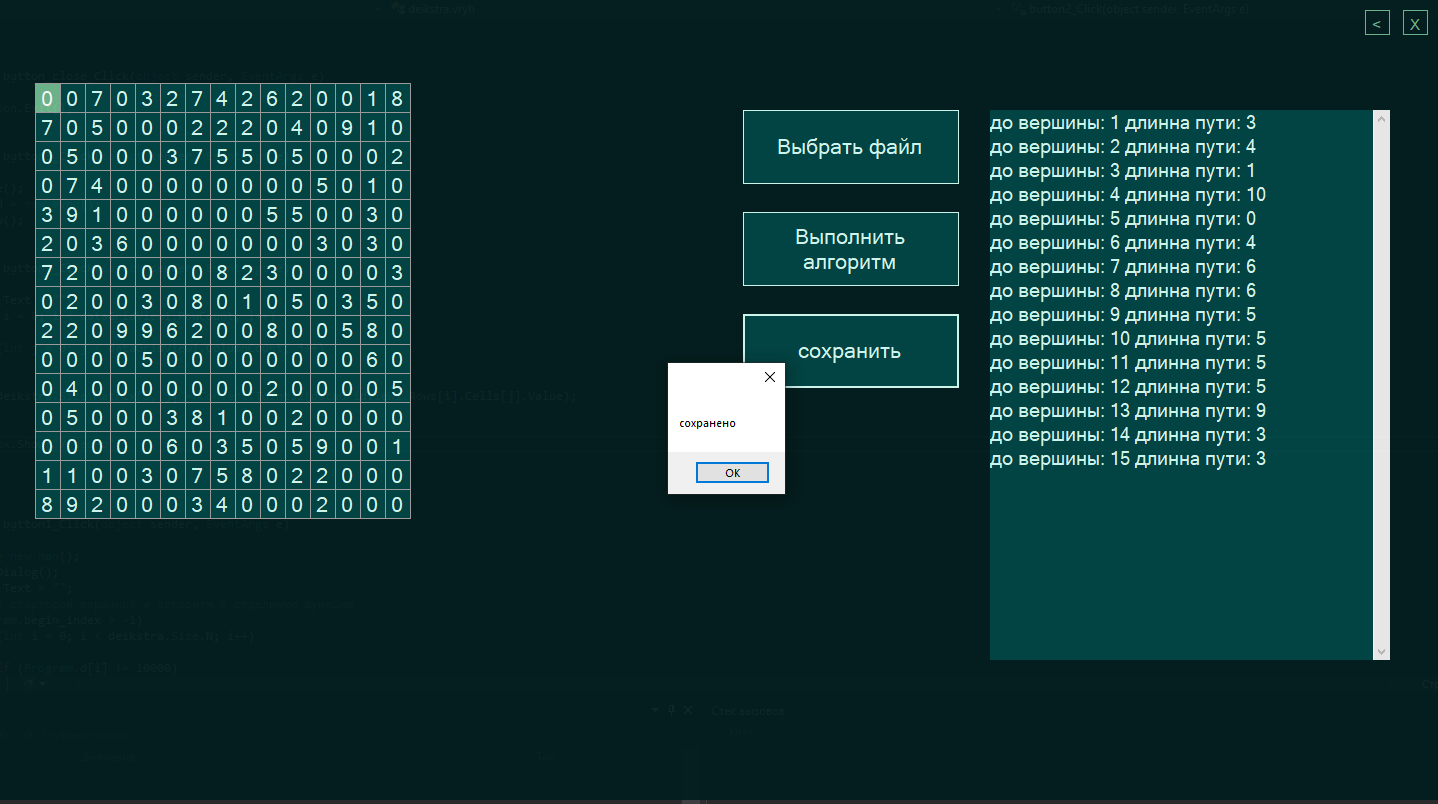


Рис. 24 - Сохранение графа с результатом

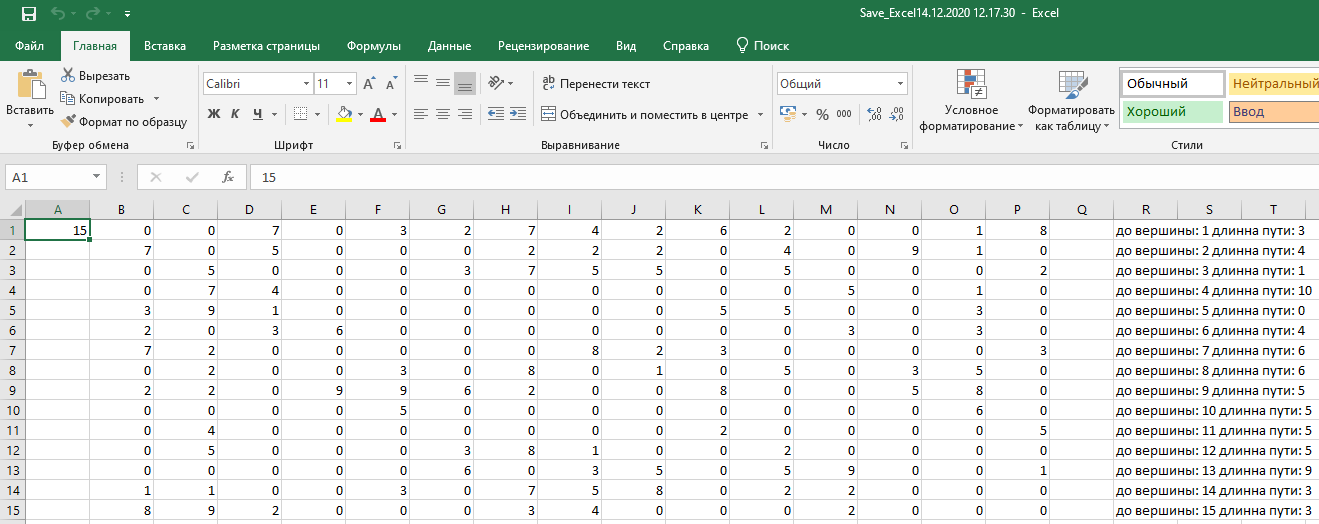


Рис. 25 - Результат сохранения графа с результатом алгоритма

1. Тестирование

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, работой алгоритма, выводом данных, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при рандомном вводе user-ом матриц различной размерностью.



Рис. 26 - Тестирование при размерности матрицы *2*

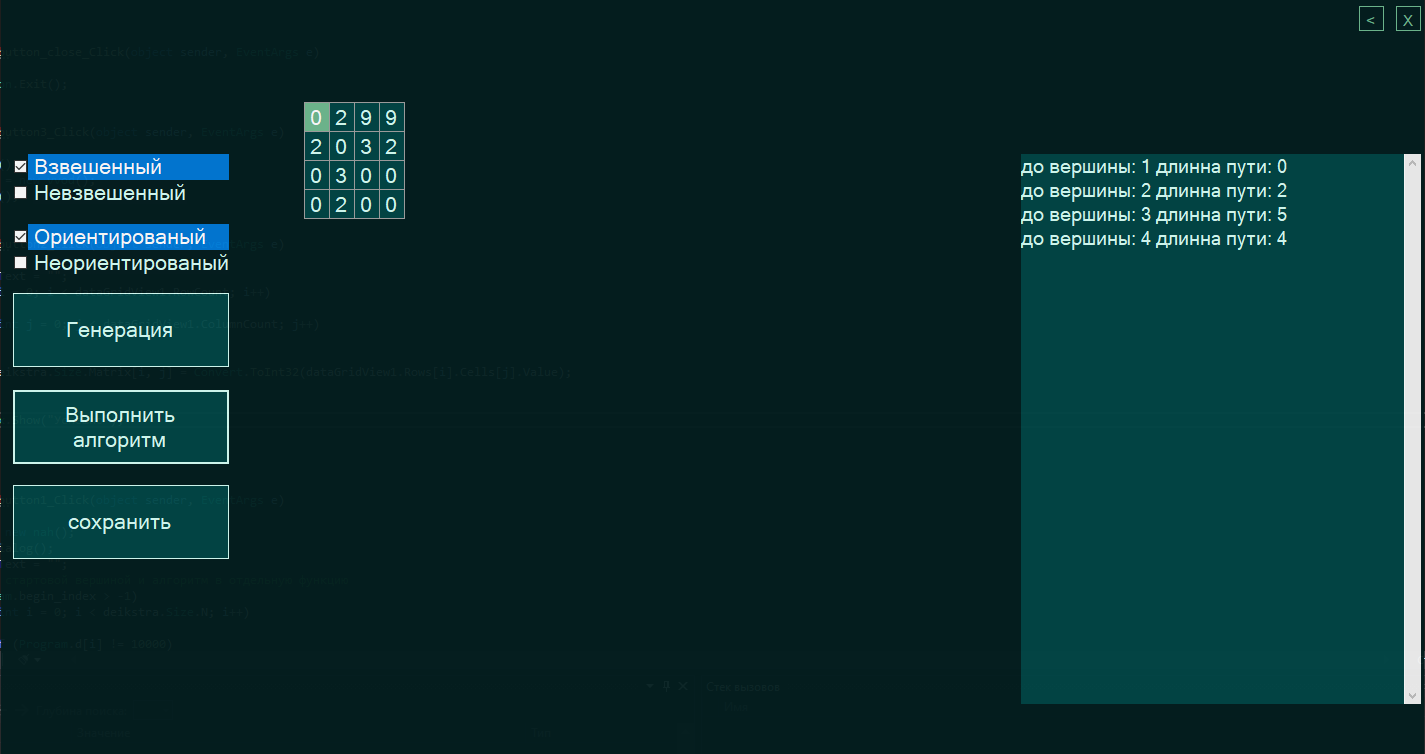


Рис. 27 - Тестирование при размерности матрицы *4*

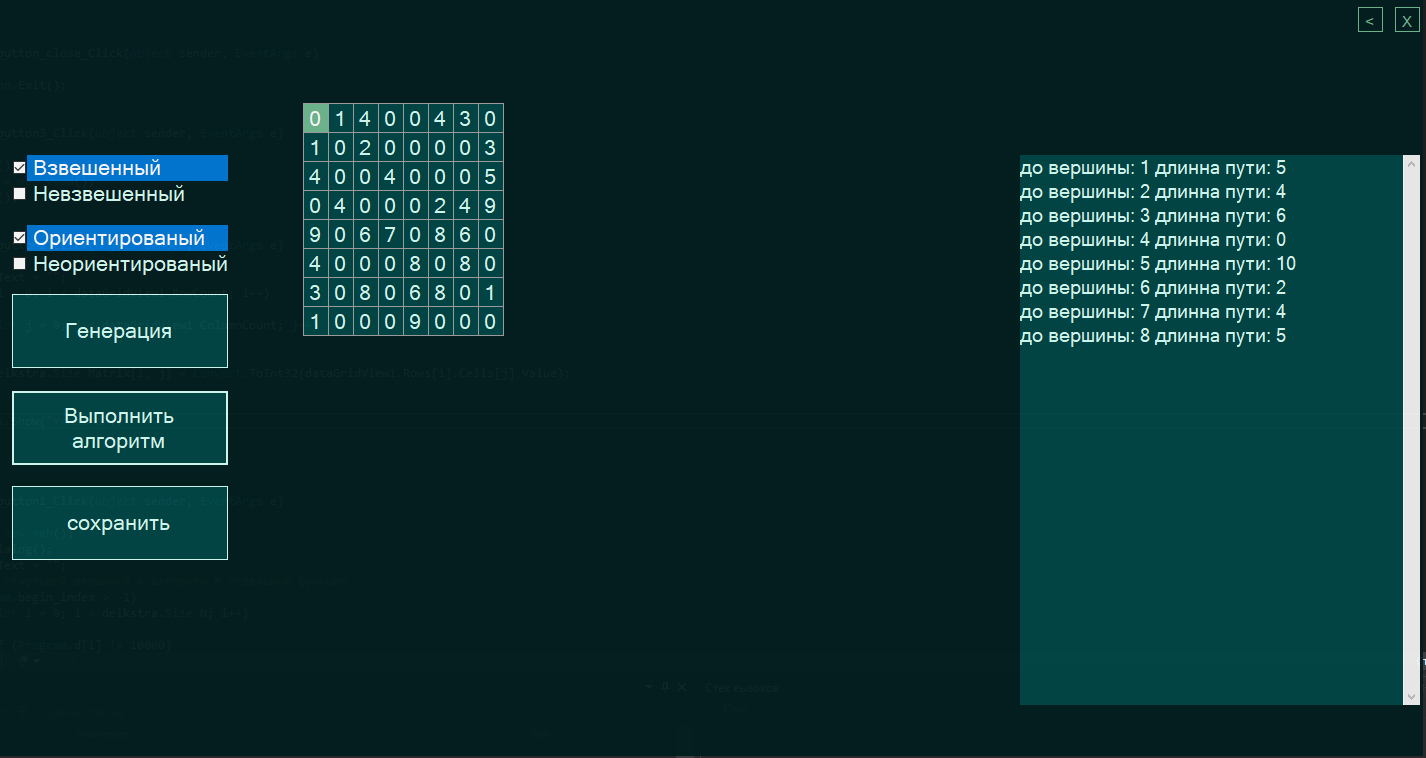


Рис. 28 - Тестирование при размерности матрицы *8*

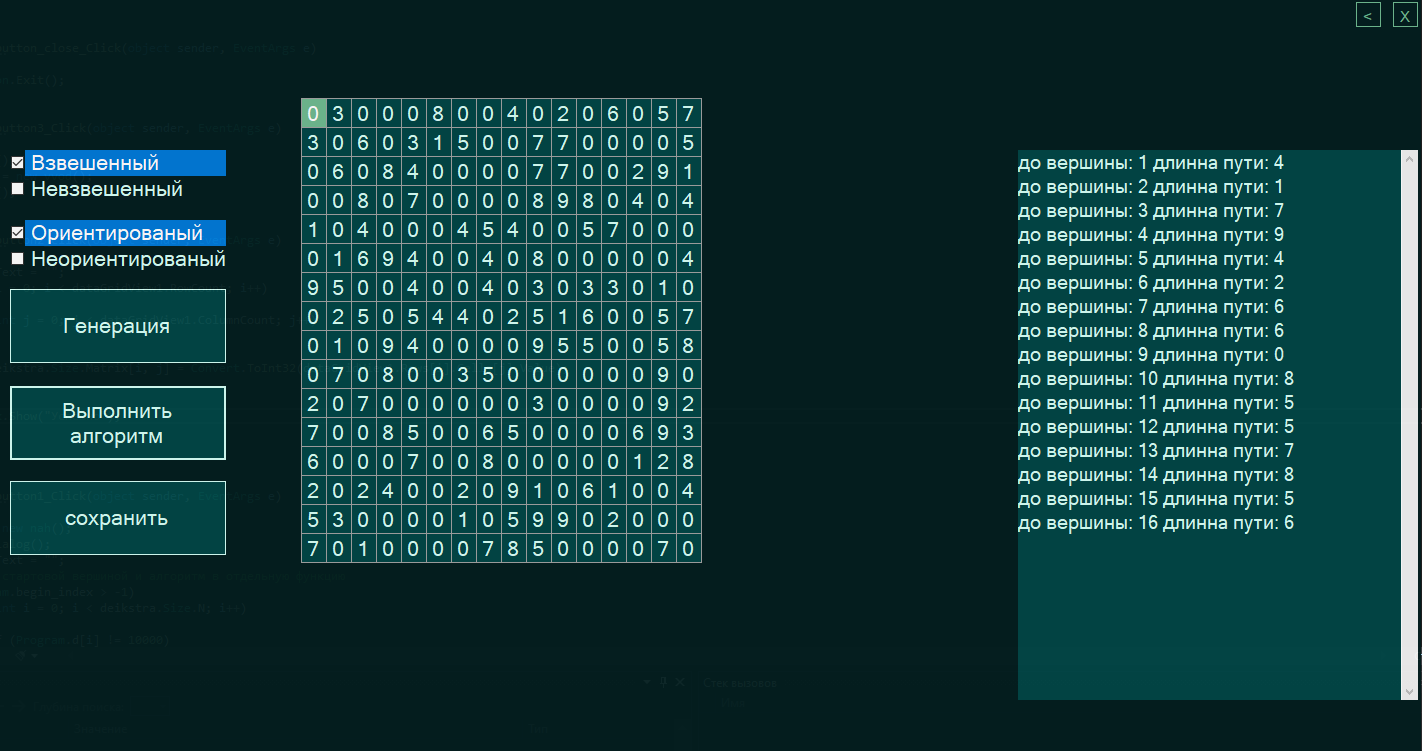


Рис. 29 - Тестирование при размерности матрицы *16*

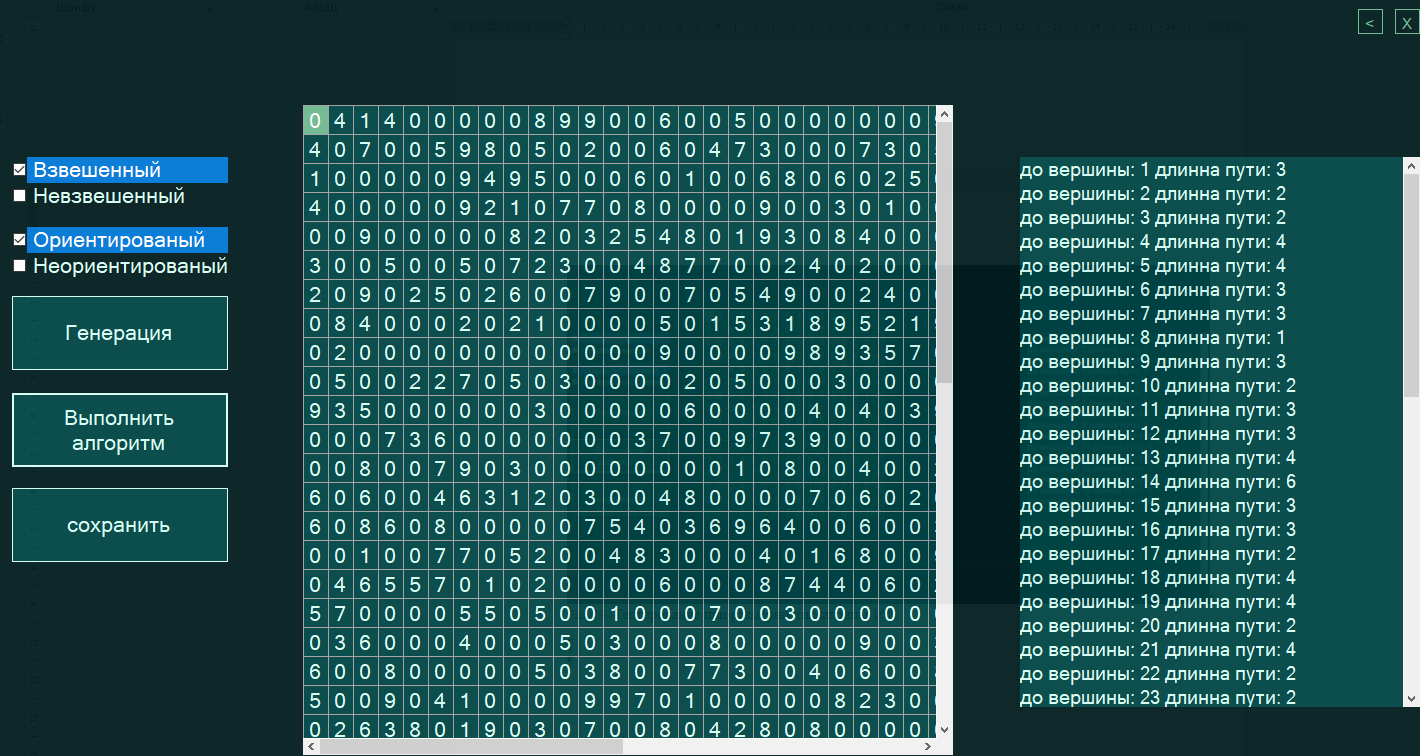


Рис. 30 - Тестирование при размерности матрицы *50*

|  |  |  |  |
| --- | --- | --- | --- |
| **Описание теста** | **Действия** | **Ожидаемый результат** | **Полученный результат** |
| Запуск программы | Нажатие на приложение | Открытие формы | Верно |
| Ручной ввод матрицы | Ввод элементов матрицы на метса. Для ввода нажатие кнопкк “Ввод” | Сообщение об успешной операции | Верно |
| Рандомный ввод матрицы | Ввод размерности матрицы и нажатие кнопки “Enter” | Вывод рандомной матрицы | Верно |
| Ввод сохраненной матрицы | Выбор сохраненного файла из проводника | Вывод сохраненной ранее матрицы | Верно |
| Результат работы программы | Нажатие кнопки “Выполнить алгоритм” | Верный вывод списка расстояний | Верно |
| Сохранение данных | Нажатие на кнопку “Сохранить”. | Сохранение данных в excel и соощение о сохранении | Верно |
| Ручной расчет | Поэтапный расчет вручную | Соответствие результату работы алгоритма | Верно |

1. Ручной расчёт задачи

Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных.

Кружками обозначены вершины, линиями – пути между ними (ребра графа). В кружках обозначены номера вершин, над ребрами обозначен их вес – длина пути. Рядом с каждой вершиной синим обозначена метка – длина кратчайшего пути в эту вершину из вершины 1.

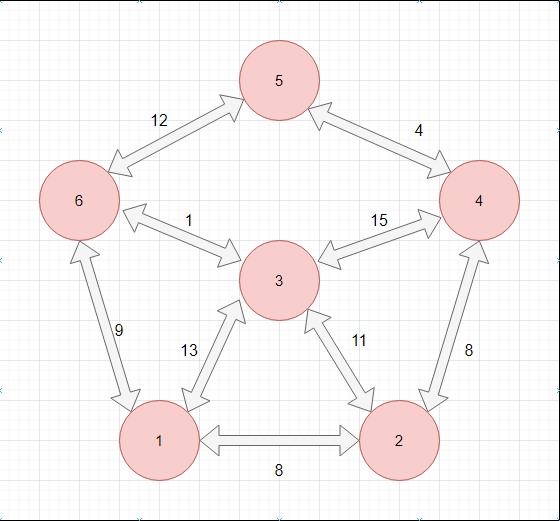


Рис. 31 граф

Инициализация

Метка самой вершины 1 полагается равной 0, метки остальных вершин – недостижимо большое число (в идеале — бесконечность). Это отражает то, что расстояния от вершины 1 до других вершин пока неизвестны. Все вершины графа помечаются как не посещенные.(цвет кружка)

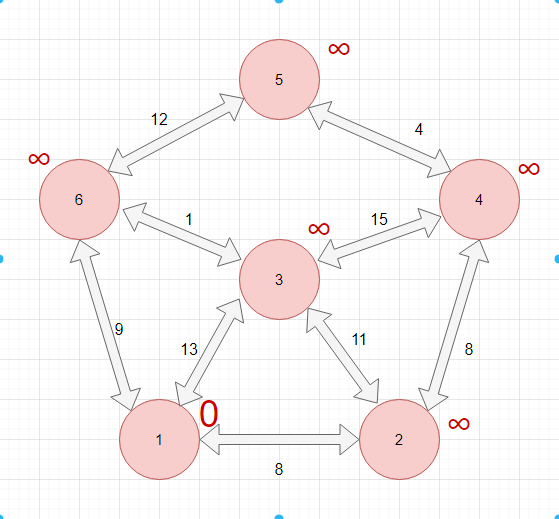


Рис. 32 инициализация

Первый шаг

Минимальную метку имеет вершина 1. Её соседями являются вершины 2, 3 и 6. Обходим соседей вершины по очереди.

Первый сосед вершины 1 – вершина 2, потому что длина пути до неё минимальна. Длина пути в неё через вершину 1 равна сумме кратчайшего расстояния до вершины 1 (значению её метки) и длины ребра, идущего из 1-й во 2-ю, то есть 0 + 8 = 8. Это меньше текущей метки вершины 2 (10000), поэтому новая метка 2-й вершины равна 8.

Аналогично находим длины пути для всех других соседей (вершины 3 и 6).

Все соседи вершины 1 проверены. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит. Вершина 1 отмечается как посещенная.

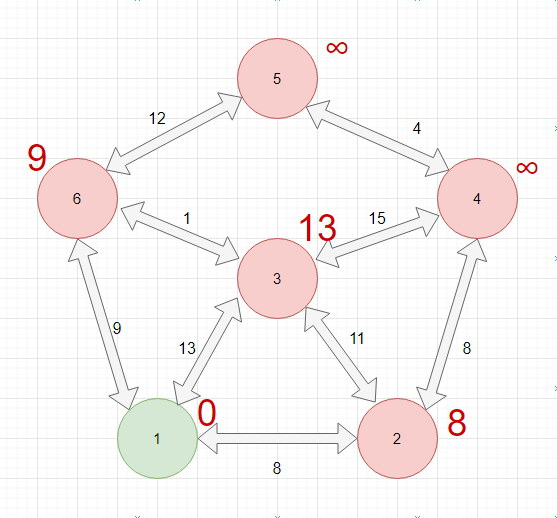


Рис. 33 шаг 1

Второй шаг

Шаг 1 алгоритма повторяется. Снова находим «ближайшую» из непосещенных вершин. Это вершина 2 с меткой 8.

Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю вершину. Соседями вершины 2 являются вершины 1, 3 и 4.

Вершина 1 уже посещена. Следующий сосед вершины 2 — вершина 3, так как имеет минимальную метку из вершин, отмеченных как не посещённые. Если идти в неё через 2, то длина такого пути будет равна 19 (8 + 11 = 17). Но текущая метка третьей вершины равна 13, а 13 < 19, поэтому метка не меняется.

Ещё один сосед вершины 2 — вершина 4. Если идти в неё через 2-ю, то длина такого пути будет равна 16 (8 + 8 = 16). Поскольку 16<10000, устанавливаем метку вершины 4 равной 16.

Все соседи вершины 2 просмотрены, помечаем её как посещенную.

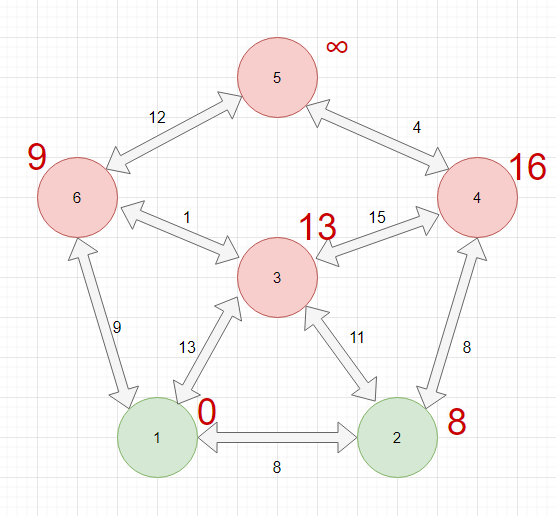


Рис. 34 шаг 2

Третий шаг

Повторяем шаг алгоритма, выбрав вершину 6. После её «обработки» получим следующие результаты.

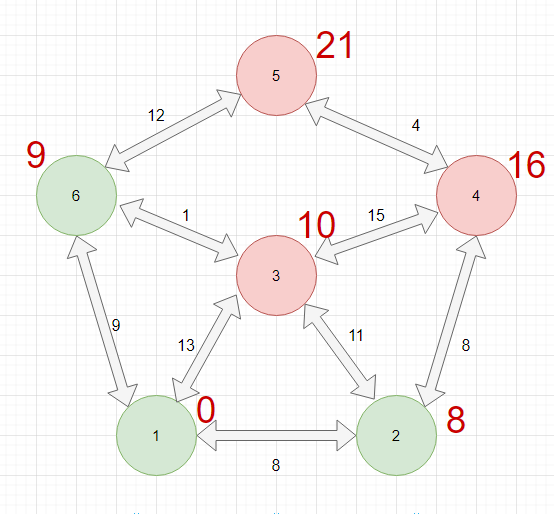


Рис. 35 шаг 3

Четвертый шаг

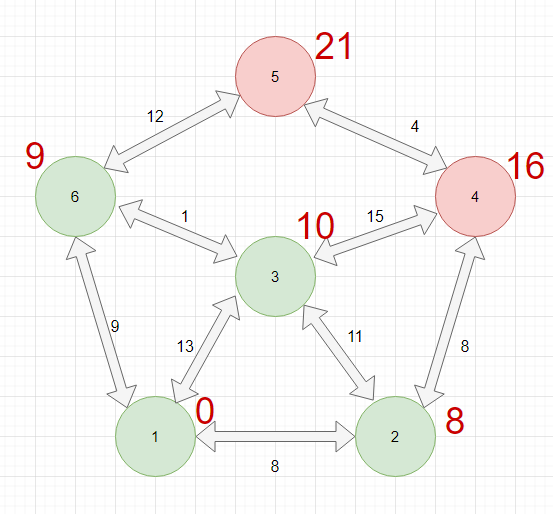


Рис. 36 шаг 4

Пятый шаг

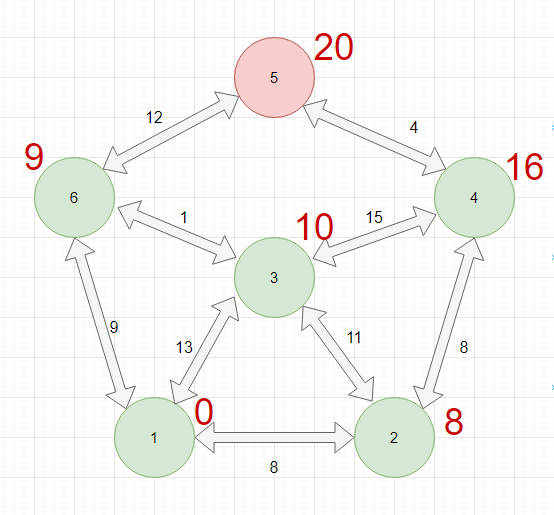


Рис. 37 шаг 5

Шестой шаг

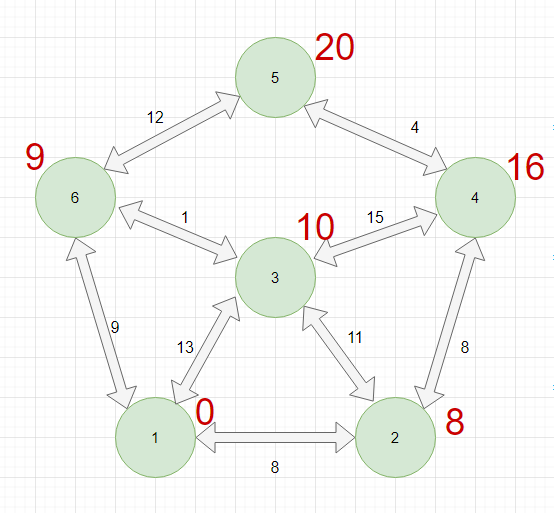


Рис. 38 шаг 6

Расстояния от первой вершины до всех остальных написаны красным.

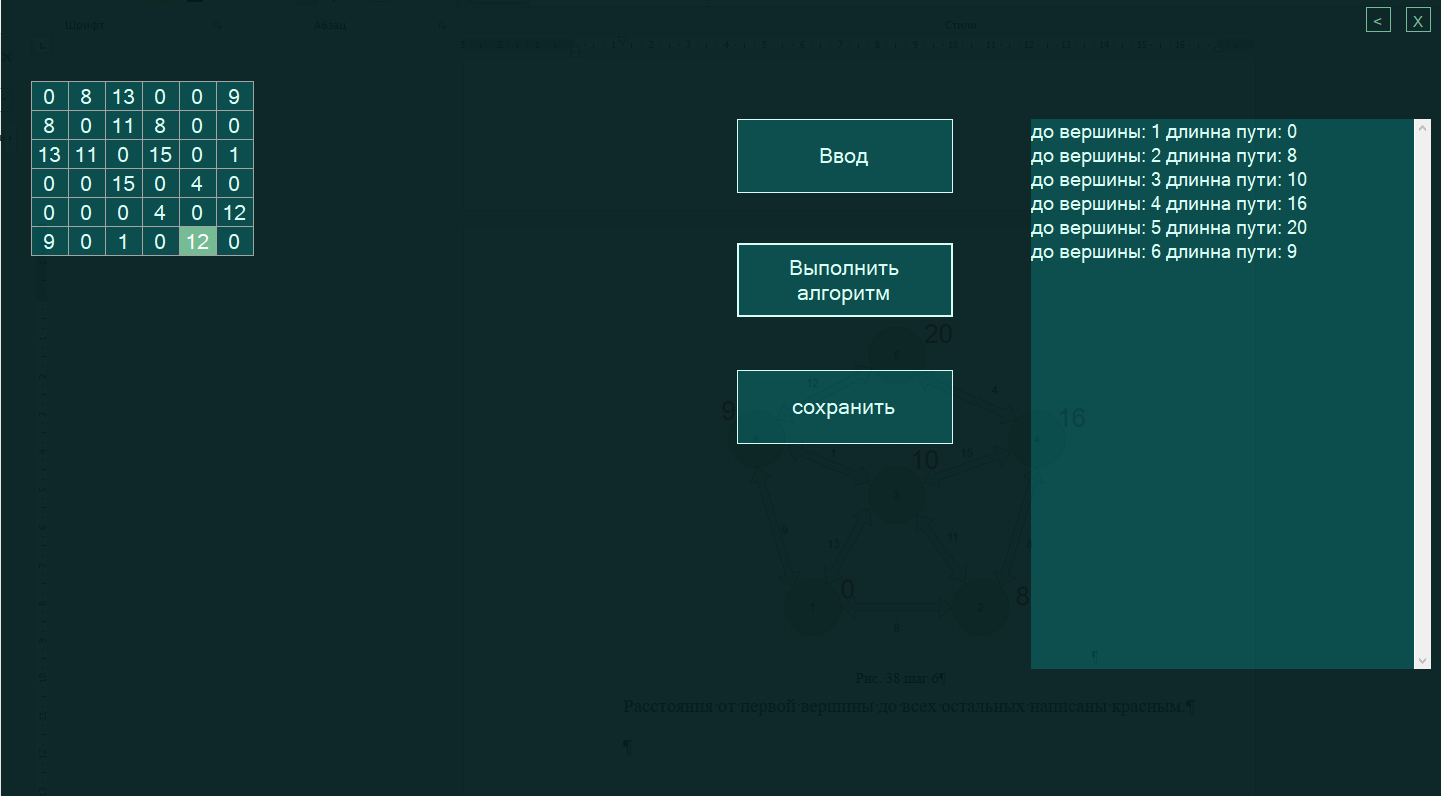


Рис. 39 – Результат проверки ручного расчета

Таким образом, можно сделать вывод, что программа работает верно.

## Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Дейкстры в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были улучшены навыки разработки программ и работы с матрицами смежностей. Приобретены навыки по осуществлению алгоритма Дейкстры. Углублены знания языка программирования C#.

Недостатком разработанной программы является отсутствие визуализации графа, понятной человеческому глазу.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

## Список литературы

* <https://prog-cpp.ru/>
* <https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры>
* <https://www.youtube.com/watch?feature=player_detailpage&v=tyQSgTytc4s>
* <https://www.youtube.com/>
* <https://www.cyberforum.ru/>
* <https://docs.microsoft.com/ru-ru/>
* <https://stackoverflow.com/>

## Приложение А. Листинг программы.

Вход программы Program.cs.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace deikstra

{

static class Program

{

public static int[] d;

public static int begin\_index = -1;

public static void alg()

{

d = new int[deikstra.Size.N]; // минимальное расстояние

int[] vis = new int[deikstra.Size.N]; // посещенные вершины

int temp, minindex, min;

//Инициализация вершин и расстояний

for (int i = 0; i < deikstra.Size.N; i++)

{

d[i] = 10000;

vis[i] = 1;

}

d[begin\_index] = 0;

// Шаг алгоритма

do

{

minindex = 10000;

min = 10000;

for (int i = 0; i < deikstra.Size.N; i++)

{ // Если вершину ещё не обошли и вес меньше min

if ((vis[i] == 1) && (d[i] < min))

{ // Переприсваиваем значения

min = d[i];

minindex = i;

}

}

// Добавляем найденный минимальный вес

// к текущему весу вершины

// и сравниваем с текущим минимальным весом вершины

if (minindex != 10000)

{

for (int i = 0; i < deikstra.Size.N; i++)

{

if (deikstra.Size.Matrix[minindex , i] > 0)

{

temp = min + deikstra.Size.Matrix[minindex, i];

if (temp < d[i])

{

d[i] = temp;

}

}

}

vis[minindex] = 0;

}

} while (minindex < 10000);

}

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace deikstra

{

public partial class Form1 : Form

{

public static int Matrix;

public Form1()

{

InitializeComponent();

timer1.Interval = 500; // 500 миллисекунд

timer1.Enabled = true;

timer1.Tick += timer1\_Tick;

}

private void timer1\_Tick(object sender, EventArgs e)

{

progressBar1.PerformStep();

if (progressBar1.Value == 100)

{

timer1.Stop();

this.Hide();

vvod vvod = new vvod();

// Показываем новое окно

vvod.Show();

}

}

private void progressBar1\_Click(object sender, EventArgs e)

{

}

}

}

vvod.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace deikstra

{

public partial class nah : Form

{

public nah()

{

InitializeComponent();

}

private void button\_close\_Click(object sender, EventArgs e)

{

this.Hide();

}

private void button1\_Click(object sender, EventArgs e)

{

if (textBox2.Text != "" && Convert.ToInt32(textBox2.Text) != 0 && Convert.ToInt32(textBox2.Text) <= deikstra.Size.N)

{

Program.begin\_index = Convert.ToInt32(textBox2.Text) - 1;

Program.alg();

this.Hide();

}

else {

MessageBox.Show("Введите вершину для начала обхода");

textBox2.Text = "";

}

}

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

char number = e.KeyChar;

if (!Char.IsDigit(number) && number != 8)

{

e.Handled = true;

}

}

private void textBox2\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

button1\_Click(this, EventArgs.Empty);

}

}

}

}

Size.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace deikstra

{

public partial class Size : Form

{

public static int N=0;

public static int[,] Matrix;

public Size()

{

InitializeComponent();

}

private void Size\_Load(object sender, EventArgs e)

{

}

private void button\_close\_Click(object sender, EventArgs e)

{

this.Hide();

vvod vvod = new vvod();

// Показываем новое окно

vvod.Show();

}

private void button1\_Click(object sender, EventArgs e)

{

if (textBox2.Text != "" && Convert.ToInt32(textBox2.Text)!= 0)

{

N = Convert.ToInt32(textBox2.Text);

this.Hide();

Matrix = new int[deikstra.Size.N, deikstra.Size.N];

switch (deikstra.vvod.p1)

{

case 1:

this.Hide();

rand rand = new rand();

rand.Show();

break;

case 2:

this.Hide();

vryh vryh = new vryh();

vryh.Show();

break;

}

}

else {

MessageBox.Show("Введите размер графа");

textBox2.Text = "";

}

}

private void textBox2\_Click(object sender, EventArgs e)

{

textBox2.Text = "";

}

private void button\_close\_Click\_1(object sender, EventArgs e)

{

Application.Exit();

}

private void button3\_Click(object sender, EventArgs e)

{

this.Hide();

vvod vvod = new vvod();

vvod.Show();

}

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

char number = e.KeyChar;

if (!Char.IsDigit(number) && number != 8)

{

e.Handled = true;

}

}

private void textBox2\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

button1\_Click(this, EventArgs.Empty);

}

}

}

}

rand.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Reflection;

using Excel = Microsoft.Office.Interop.Excel;

namespace deikstra

{

public partial class rand : Form

{

public rand()

{

InitializeComponent();

}

private void rand\_Zap\_vz\_or()

{

Random rand = new Random(DateTime.Now.Second);

for (int i = 0; i < deikstra.Size.N; i++)

for (int j = 0; j < deikstra.Size.N; j++)

{

if (i == j)

{

deikstra.Size.Matrix[i,j] = 0;

}

if (i < j)

{

if (rand.Next(0, 10) > 4)

{

deikstra.Size.Matrix[i,j] = rand.Next(0, 10);

if (rand.Next(0, 10) > 4)

deikstra.Size.Matrix[j,i] = 0;

else

deikstra.Size.Matrix[j,i] = deikstra.Size.Matrix[i,j];

}

else

if (rand.Next(0, 10) > 4)

{

deikstra.Size.Matrix[j,i] = rand.Next(0, 10);

deikstra.Size.Matrix[i,j] = 0;

}

else

{

deikstra.Size.Matrix[i,j] = 0;

deikstra.Size.Matrix[j,i] = 0;

}

}

}

}

private void rand\_Zap\_vz\_nor()

{

Random rand = new Random(DateTime.Now.Second);

for (int i = 0; i < deikstra.Size.N; i++)

for (int j = 0; j < deikstra.Size.N; j++)

{

if (i == j)

{

deikstra.Size.Matrix[i ,j] = 0;

}

if (i < j)

{

if (rand.Next(0, 10) > 4)

deikstra.Size.Matrix[i,j] = rand.Next(0, 10);

else

deikstra.Size.Matrix[i,j] = 0;

deikstra.Size.Matrix[j ,i] = deikstra.Size.Matrix[i ,j];

}

}

}

private void rand\_Zap\_nor()

{

Random rand = new Random(DateTime.Now.Second);

for (int i = 0; i < deikstra.Size.N; i++)

for (int j = 0; j < deikstra.Size.N; j++)

{

if (i == j)

{

deikstra.Size.Matrix[i,j] = 0;

}

if (i < j)

{

deikstra.Size.Matrix[i ,j] = rand.Next(0, 2);

deikstra.Size.Matrix[j,i] = deikstra.Size.Matrix[i,j];

}

}

}

private void rand\_Zap\_or()

{

Random rand = new Random(DateTime.Now.Second);

for (int i = 0; i < deikstra.Size.N; i++)

for (int j = 0; j < deikstra.Size.N; j++)

{

if (i == j)

{

deikstra.Size.Matrix[i ,j] = 0;

}

if (i < j)

{

if (rand.Next(0, 10) > 4)

{

deikstra.Size.Matrix[i ,j] = rand.Next(0, 2);

if (rand.Next(0, 10) > 4)

deikstra.Size.Matrix[j ,i] = 0;

else

deikstra.Size.Matrix[j ,i] = deikstra.Size.Matrix[i ,j];

}

else

if (rand.Next(0, 10) > 4)

{

deikstra.Size.Matrix[j,i] = rand.Next(0, 2);

deikstra.Size.Matrix[i,j] = 0;

}

else

{

deikstra.Size.Matrix[i,j] = 0;

deikstra.Size.Matrix[j,i] = 0;

}

}

}

}

private bool b1=false;

private void button1\_Click(object sender, EventArgs e)

{

b1 = true;

textBox1.Text = "";

switch (checkedListBox1.SelectedIndex)

{

case 0:

switch (checkedListBox2.SelectedIndex)

{

case 0:

rand\_Zap\_vz\_or();

break;

case 1:

rand\_Zap\_vz\_nor();

break;

}

break;

case 1:

switch (checkedListBox2.SelectedIndex)

{

case 0:

rand\_Zap\_or();

break;

case 1:

rand\_Zap\_nor();

break;

}

break;

default:

break;

}

dataGridView1.RowCount = deikstra.Size.Matrix.GetLength(0);

dataGridView1.ColumnCount = deikstra.Size.Matrix.GetLength(1);

for (int i = 0; i < deikstra.Size.Matrix.GetLength(0); i++)

{

for (int j = 0; j < deikstra.Size.Matrix.GetLength(1); j++)

{

//пишем значения из массива в ячейки контролла

dataGridView1.Rows[i].Cells[j].Value = deikstra.Size.Matrix[i, j];

}

}

}

private void checkedListBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (checkedListBox1.CheckedItems.Count > 1)

{

for (int i = 0; i < checkedListBox1.Items.Count; i++)

{

checkedListBox1.SetItemChecked(i, false);

}

checkedListBox1.SetItemChecked(checkedListBox1.SelectedIndex, true);

}

}

private void checkedListBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (checkedListBox2.CheckedItems.Count > 1)

{

for (int i = 0; i < checkedListBox2.Items.Count; i++)

{

checkedListBox2.SetItemChecked(i, false);

}

checkedListBox2.SetItemChecked(checkedListBox2.SelectedIndex, true);

}

}

private void button\_close\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void button2\_Click(object sender, EventArgs e)

{

if (b1 == true)

{

nah nah = new nah();

nah.ShowDialog();

textBox1.Text = "";

//окно со стартовой вершиной и алгоритм в отдельную функцию

if (Program.begin\_index > -1)

for (int i = 0; i < deikstra.Size.N; i++)

{

if (Program.d[i] != 10000)

textBox1.Text += String.Format("до вершины: {0} длинна пути: {1} ", i + 1, Program.d[i]) + '\r' + '\n';

else

textBox1.Text += String.Format("до вершины: {0} нет пути", i + 1) + '\r' + '\n';

}

}

else

MessageBox.Show("Сгенерируйте граф");

}

private void button3\_Click(object sender, EventArgs e)

{

this.Hide();

vvod vvod = new vvod();

vvod.Show();

}

public static void SaveTable(DataGridView What\_save, TextBox textBox)

{

string path = System.IO.Directory.GetCurrentDirectory() + @"\" + String.Format("Save\_Excel{0}.{1}.{2} {3}.{4}.{5}.xlsx", DateTime.Now.Day, DateTime.Now.Month, DateTime.Now.Year, DateTime.Now.Hour, DateTime.Now.Minute, DateTime.Now.Second) ;

Excel.Application excelapp = new Excel.Application();

Excel.Workbook workbook = excelapp.Workbooks.Add();

Excel.Worksheet worksheet = workbook.ActiveSheet;

worksheet.Rows[1].Columns[1] = What\_save.RowCount;

for (int i = 1; i < What\_save.RowCount + 1; i++)

{

for (int j = 2; j < What\_save.ColumnCount + 2; j++)

{

worksheet.Rows[i].Columns[j] = What\_save.Rows[i - 1].Cells[j - 2].Value;

}

}

if (textBox.Text != "")

{

for (int i = 1; i < What\_save.RowCount + 1; i++)

{

worksheet.Rows[i].Columns[What\_save.ColumnCount + 3] = textBox.Lines[i - 1];

}

}

excelapp.AlertBeforeOverwriting = false;

workbook.SaveAs(path);

excelapp.Quit();

GC.Collect();

}

private void button4\_Click(object sender, EventArgs e)

{

SaveTable(dataGridView1, textBox1);

MessageBox.Show("сохранено");

}

}

}

nah.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace deikstra

{

public partial class nah : Form

{

public nah()

{

InitializeComponent();

}

private void button\_close\_Click(object sender, EventArgs e)

{

this.Hide();

}

private void button1\_Click(object sender, EventArgs e)

{

if (textBox2.Text != "" && Convert.ToInt32(textBox2.Text) != 0 && Convert.ToInt32(textBox2.Text) <= deikstra.Size.N)

{

Program.begin\_index = Convert.ToInt32(textBox2.Text) - 1;

Program.alg();

this.Hide();

}

else {

MessageBox.Show("Введите вершину для начала обхода");

textBox2.Text = "";

}

}

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

char number = e.KeyChar;

if (!Char.IsDigit(number) && number != 8)

{

e.Handled = true;

}

}

private void textBox2\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

button1\_Click(this, EventArgs.Empty);

}

}

}

}

fail.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

namespace deikstra

{

public partial class fail : Form

{

public static string file=" ";

public fail()

{

InitializeComponent();

}

private void button\_close\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void button3\_Click(object sender, EventArgs e)

{

this.Hide();

vvod vvod = new vvod();

vvod.Show();

}

private bool b1=false;

private void button1\_Click(object sender, EventArgs e)

{

//Folder folder = new Folder();

//folder.Show();

OpenFileDialog OPF = new OpenFileDialog();

OPF.Filter = "файл Excel (Spisok.xlsx)|\*.xlsx";

if (OPF.ShowDialog() == DialogResult.OK)

{

fail.file = OPF.FileName;

}

if (file != " ")

{

Excel.Application ObjWorkExcel = new Excel.Application(); //открыть эксель

Excel.Workbook ObjWorkBook = ObjWorkExcel.Workbooks.Open(file); //открыть файл

Excel.Worksheet ObjWorkSheet = (Excel.Worksheet)ObjWorkBook.Sheets[1]; //получить 1 лист

int S = Convert.ToInt32(ObjWorkSheet.Rows[1].Columns[1].Text.ToString());

dataGridView1.RowCount = S;

dataGridView1.ColumnCount = S;

deikstra.Size.N = S;

for (int i = 0; i < dataGridView1.RowCount; i++) //по всем колонкам

for (int j = 0; j < dataGridView1.ColumnCount; j++) // по всем строкам

dataGridView1.Rows[i].Cells[j].Value = Convert.ToInt32(ObjWorkSheet.Rows[i+1].Columns[j+2].Text.ToString());

ObjWorkBook.Close(false, Type.Missing, Type.Missing); //закрыть не сохраняя

ObjWorkExcel.Quit(); // выйти из экселя

GC.Collect(); // убрать за собой

System.Runtime.InteropServices.Marshal.FinalReleaseComObject(ObjWorkExcel);

MessageBox.Show("Успешно");

b1 = true;

}

else

MessageBox.Show("Выберите файл");

}

private void button2\_Click(object sender, EventArgs e)

{

deikstra.Size.Matrix = new int[deikstra.Size.N, deikstra.Size.N];

if (b1 == true)

{

for (int i = 0; i < dataGridView1.RowCount; i++)

{

for (int j = 0; j < dataGridView1.ColumnCount; j++)

{

deikstra.Size.Matrix[i, j] = Convert.ToInt32(dataGridView1.Rows[i].Cells[j].Value);

}

}

nah nah = new nah();

nah.ShowDialog();

textBox1.Text = "";

//окно со стартовой вершиной и алгоритм в отдельную функцию

if (Program.begin\_index > -1)

for (int i = 0; i < deikstra.Size.N; i++)

{

if (Program.d[i] != 10000)

textBox1.Text += String.Format("до вершины: {0} длинна пути: {1} ", i + 1, Program.d[i]) + '\r' + '\n';

else

textBox1.Text += String.Format("до вершины: {0} нет пути", i + 1) + '\r' + '\n';

}

}

else

MessageBox.Show("Выберите файл");

}

private void button4\_Click(object sender, EventArgs e)

{

rand.SaveTable(dataGridView1, textBox1);

MessageBox.Show("сохранено");

}

}

}

vryh.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

namespace deikstra

{

public partial class vryh : Form

{

public vryh()

{

InitializeComponent();

dataGridView1.RowCount = deikstra.Size.Matrix.GetLength(0);

dataGridView1.ColumnCount = deikstra.Size.Matrix.GetLength(1);

for (int i = 0; i < deikstra.Size.Matrix.GetLength(0); i++)

{

for (int j = 0; j < deikstra.Size.Matrix.GetLength(1); j++)

{

//пишем значения из массива в ячейки контролла

dataGridView1.Rows[i].Cells[j].Value = deikstra.Size.Matrix[i, j];

}

}

}

private void button\_close\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void button3\_Click(object sender, EventArgs e)

{

this.Hide();

vvod vvod = new vvod();

vvod.Show();

}

private void button2\_Click(object sender, EventArgs e)

{

textBox1.Text = "";

for (int i = 0; i < dataGridView1.RowCount; i++)

{

for (int j = 0; j < dataGridView1.ColumnCount; j++)

{

deikstra.Size.Matrix[i, j] = Convert.ToInt32(dataGridView1.Rows[i].Cells[j].Value);

}

}

MessageBox.Show("Успешно");

}

private void button1\_Click(object sender, EventArgs e)

{

nah nah = new nah();

nah.ShowDialog();

textBox1.Text = "";

//окно со стартовой вершиной и алгоритм в отдельную функцию

if (Program.begin\_index > -1)

for (int i = 0; i < deikstra.Size.N; i++)

{

if (Program.d[i] != 10000)

textBox1.Text += String.Format("до вершины: {0} длинна пути: {1} ", i + 1, Program.d[i]) + '\r' + '\n';

else

textBox1.Text += String.Format("до вершины: {0} нет пути", i + 1) + '\r' + '\n';

}

}

private void button4\_Click(object sender, EventArgs e)

{

rand.SaveTable(dataGridView1, textBox1);

MessageBox.Show("сохранено");

}

private void dataGridView1\_EditingControlShowing(object sender, DataGridViewEditingControlShowingEventArgs e)

{

TextBox tb = (TextBox)e.Control;

tb.KeyPress += new KeyPressEventHandler(tb\_KeyPress);

}

void tb\_KeyPress(object sender, KeyPressEventArgs e)

{

string vlCell = ((TextBox)sender).Text;

bool temp = (vlCell.IndexOf(".") == -1);

if (!(Char.IsDigit(e.KeyChar)))

{

if (e.KeyChar != (char)Keys.Back)

{

e.Handled = true;

}

}

}

}

}