Министерство науки и высшего 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»

Выполнила:

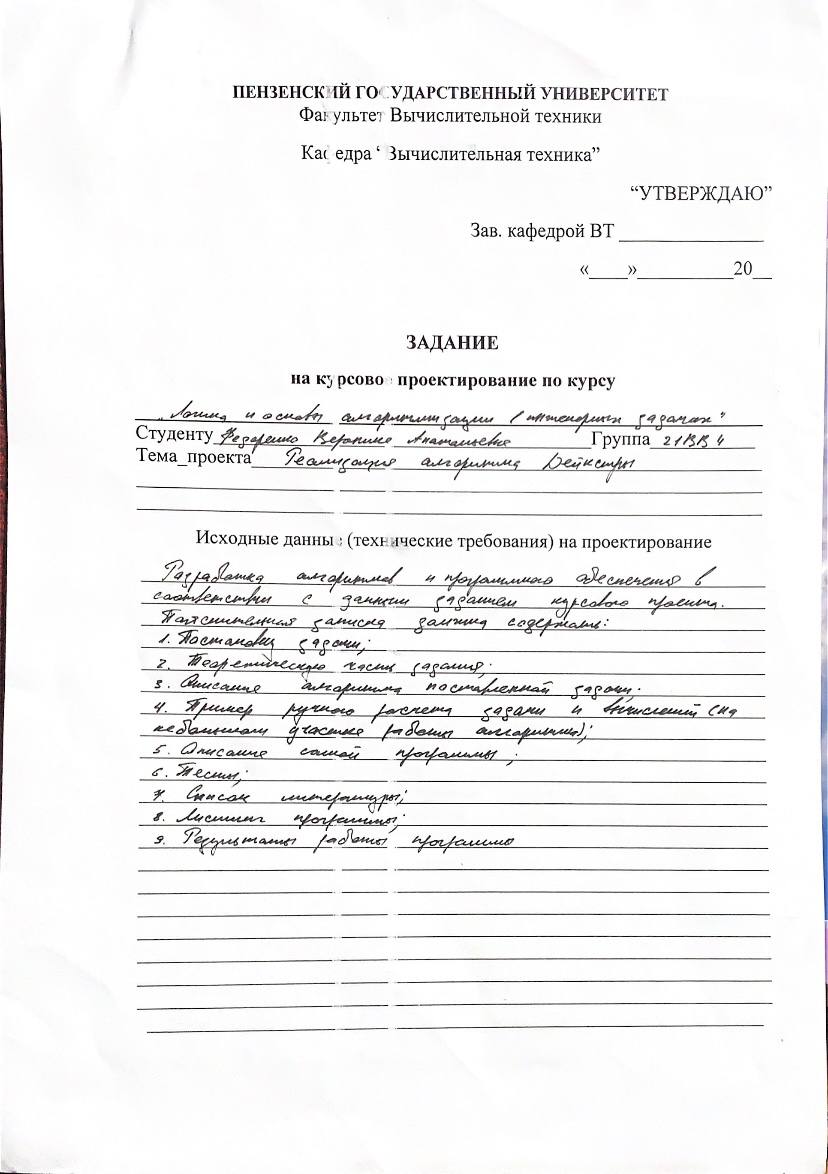
студентка группы 21ВВ4

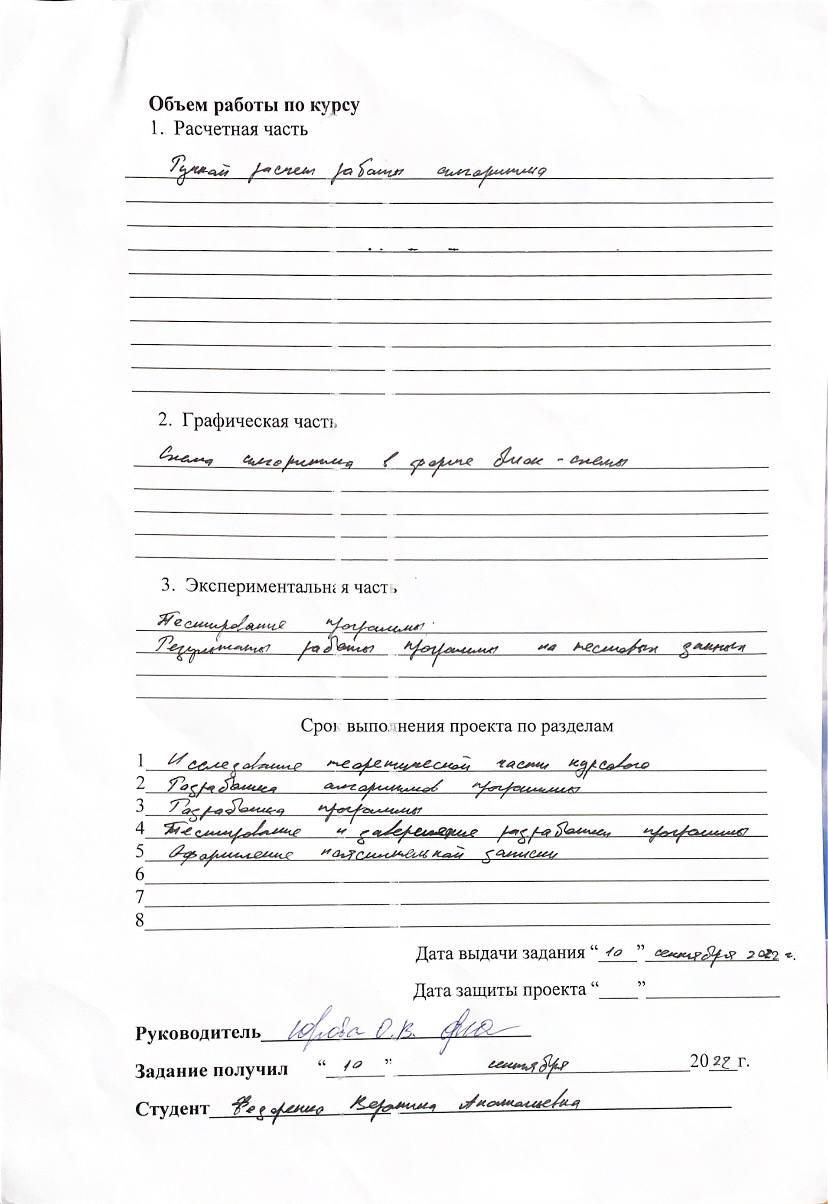
Федоренко В.А.

Приняла:

к.т.н Юрова О.В.

Пенза 2022





Оглавление

[Реферат 5](#_Toc122906582)

[Введение 6](#_Toc122906583)

[1 Постановка задачи 7](#_Toc122906584)

[2 Теоретическая часть задания 8](#_Toc122906585)

[3 Описание алгоритма программы 17](#_Toc122906586)

[4 Описание программы 21](#_Toc122906587)

[5 Тестирование 26](#_Toc122906588)

[6 Ручной расчет задачи 30](#_Toc122906589)

[Заключение 32](#_Toc122906590)

[Список литературы 33](#_Toc122906591)

[Приложение А. 34](#_Toc122906592)

# Реферат

Отчет 29 стр., 21 рисунок

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ДЕЙКСТРЫ.

Цель исследования – разработка программы, способной находить кратчайшие расстояния от одной вершины до всех остальных по алгоритму Дейкстры.

В работе рассмотрены правила реализации алгоритма Дейкстры, на основе которых находятся расстояния. Установлено, что с помощью данного алгоритма можно находить расстояния от одной вершины графа до всех остальных и кратчайший путь.

# Введение

Алгоритм Дейкстры (англ. Dijkstra’s algorithm) – алгоритм на графах, изобретенный нидерландским ученым Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной вершины до всех остальных. Алгоритм работает только для графов без ребер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, его используют протоколы маршрутизации OSPF и IS-IS.

Для поиска минимального расстояния до всех вершин графа мы не сможем использовать ни обход в глубину, ни обход в ширину, так как нужно регулярно перепроверять все имеющиеся значения с уже полученными. Для подобных задач используется алгоритм Дейкстры. Его основное преимущество заключается в том, что при минимальных вычислительных затратах машин, методом перепроверки расстояний в каждой новой вершине графа безошибочно находится кратчайшее расстояние.

В качестве среды разработки мною была выбрана IDE JetBrains Rider, язык программирования – C#.

Целью данной курсовой работы является разработка программы на языке C#. Который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Дейкстры.

# 1 Постановка задачи

Требуется разработать программу, которая будет находить кратчайшие расстояния от одной вершины графа до всех остальных, используя алгоритм Дейкстры.

Исходный граф в программе должен задаваться методом введения расстояний между всеми вершинами графа (отсутствие связи между двумя вершинами должно обозначаться «0»). Программа должна работать так, чтобы пользователь мог самостоятельно ввести вершину, для которой требуется найти все минимальные расстояния. После обработки этих данных программа должна выводить на экран матрицу графа, вершины и сами расстояния. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно. Устройства ввода – клавиатура и мышь.

# 2 Теоретическая часть задания

Рассмотрим выполнение алгоритма на примере графа, показанного на рисунке 1.

Пусть требуется найти кратчайшие расстояния от первой вершины до всех остальных.

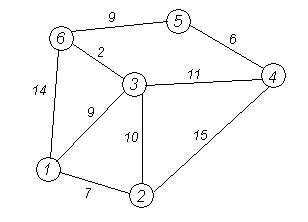


Рисунок 1 – пример графа.

Кружками обозначены вершины, линиями – пути между ними (ребра графа).

В кружках обозначены номера вершин, над ребрами обозначены их веса – длина пути.

Рядом с каждой вершиной красным обозначена метка (см. рисунок 2) – длина кратчайшего пути в эту вершину из вершины один.

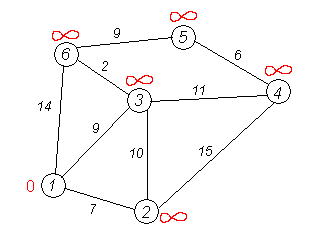


Рисунок 2 – обозначаем вершины графов.

Первый шаг.

Минимальную метку имеет вершина один. Её соседями являются вершины 2, 3 и 6 (см. рисунок 3).

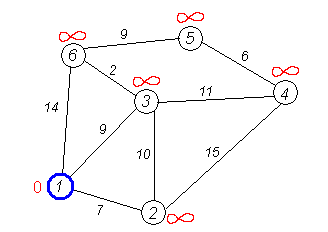


Рисунок 3 – начинаем обход с вершины 1.

Первый по очереди сосед вершины 1 – вершина два потому что длина пути до неё минимальна (см. рисунок 4).

Длина пути в неё через вершину 1 равна сумме значений метки вершины 1 и длины ребра, идущего из первой во вторую, то есть 0+7 = 7.

Это меньше текущей метки вершины два, бесконечности, поэтому новая метка второй вершины равна 7.

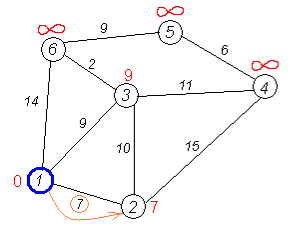


Рисунок 4 – расстояние до вершины №2.

Аналогично проделываем с двумя другими соседями первой вершины – третьей и шестой (см. рисунок 5).

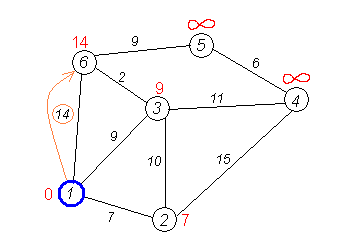


Рисунок 5 – высчитываем расстояния до вершин №3 и №6.

Все соседи первой вершины проверены.

Текущее минимальное расстояние до первой вершины считается окончательным и пересмотру не подлежит.

Вычеркнем её из графа, что отметить, что вершина посещена (см. рисунок 6).

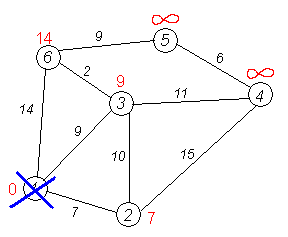


Рисунок 6 – вершина №1 считается посещенной.

Второй шаг.

Снова находим «ближайшую» из не посещенных вершин. Это вторая вершина с меткой 7 (см. рисунок 7).

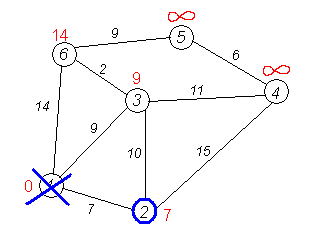


Рисунок 7- обход вершины №2.

Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через вторую вершину. Соседями второй вершины являются первая, третья и четвертая вершины.

Первый (по порядку) сосед второй вершины – первая вершина. Но она уже посещена, поэтому с первой вершиной ничего не делаем.

Следующий сосед – третья вершина, так как имеет минимальную метку.

Если идти в неё через вторую вершину, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9, а это меньше 17, поэтому метка не меняется (см. рисунок 8).

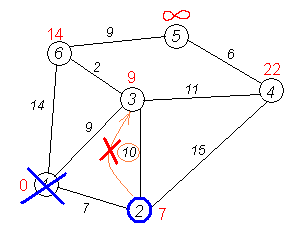


Рисунок 8 – проверка расстояния до вершины №3.

Ещё один сосед второй вершины – четвертая вершина.

Если идти в неё через вторую, то длина такого пути будет равна сумме кратчайшего расстояния до второй вершины и расстояние между второй и четвертой вершинами, то есть 22 (7 + 15 = 22).

Поскольку 22 меньше бесконечности, устанавливаем метку четвертой вершины равной 22 (см. рисунок 9).

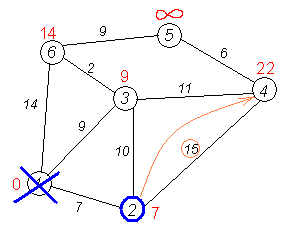


Рисунок 9 – проверка расстояний до вершины №4.

все соседи второй вершины просмотрены, замораживаем расстояние до неё и помечаем её как посещенную (см. рисунок 10).

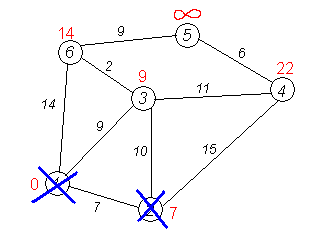


Рисунок 10 – вершина №2 считается посещенной.

Третий шаг.

Повторяем шаг алгоритма, выбрав третью вершину (см. рисунок 11). После её обработки получим такие результаты:

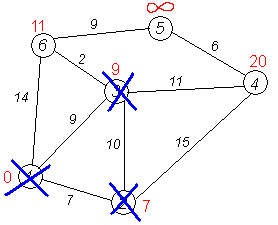


Рисунок 11 – обработка вершины №3.

Дальнейшие шаги.

Повторяем шаг алгоритма для оставшихся вершин. Это будут шестая, четвертая и пятая, соответственно по порядку (см. рисунок 12, 13, 14).

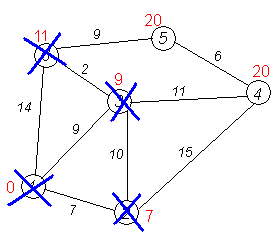


Рисунок 12 – обработка вершины №6.

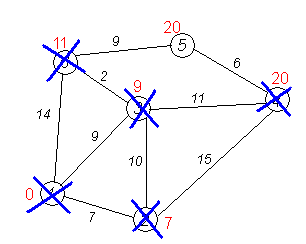


Рисунок 13 – обработка вершины №4.

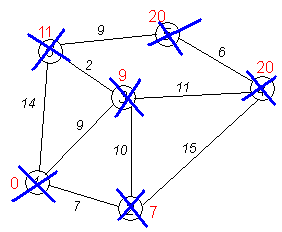


Рисунок 14 – обработка вершины №5.

Завершение выполнения программы.

Алгоритм заканчивает работу, когда все вершины посещены.

Результат работы алгоритма виден на последнем рисунке: кратчайший путь от первой вершины до второй – 7, до третьей – 9, до четвертой – 20, до пятой – 20, до шестой – 11.

Если в какой-то момент все не посещенные вершины помечены бесконечностью, то это значит, что до этих вершин нельзя добраться (то есть граф несвязный). Тогда алгоритм может быть завершен досрочно.

# 3 Описание алгоритма программы

Для начала запрашиваем у пользователя выбор действия в меню, который представляет из себя следующее:

1. Чтение с файла;
2. Ввести случайную матрицу.

Далее формируется список списков типа данных int, выделяется память под них.

Если пользователь выбрал случайную генерацию, ему предлагается ввести размер матрицы смежности, выбрать будет ли граф взвешенный или не взвешенный, ориентированный или неориентированный.

Выводим матрицу смежности для наглядности.

Далее устанавливаем расстояние до всех вершин. Для этого создаем переменную типа int и присваиваем ей максимальное значение int = 2147483647.

**Псевдокод функции Main()**

1. Создать генератор псевдослучайных чисел.
2. Конфигурировать консоль
3. Цикл ПокаТребуетсяВвод
   1. ПокаТребуетсяВвод = НапечататьМеню();
4. Конец Main

**Псевдокод функции НапечататьМеню()**

1. Напечатать сообщения В консоль
2. ПользовательскийВвод = считатьЗначениеИзКонсоли()
3. Если ПользовательскийВвод не цифра или не входит в допустимые значения

То Завершить НапечататьМеню

1. Если ПользовательскийВвод == 3

То Завершить выполнение программы

Матрица = Null

1. Если ПользовательскийВвод == 1 То

5.1 Матрица = ПрочитатьИзФайла()

6. Если ПользовательскийВвод == 2 То Матрица = ЗадатьМатрицуСлучайно()

7. Спросить пользователя С какой вершины начать алгоритм

8. вершина = СчитатьЗначениеИзКонсоли()

9. Если вершина не число или больше размерности матрицы

То Завершить НапечататьМеню()

10. НапечататьМатрицу(Матрица)

11. АлгоритмДейкстры(Матрица)

12. Конец НапечататьМеню

**Псевдокод функции ПрочитатьИзФайла()**

1. Матрица = выделитьПамятьПодМатрицу
2. Прочитать файл matrix.csv
3. Цикл пока Файл не пустой
   1. Прочитать строку в матрицу
4. ПроверитьПравильностьМатрицы(Матрица )
5. Вернуть Матрица

**Псевдокод функции СгенерироватьМатрицуПсевдослучайно()**

1. Спросить у пользователя размер матрицы

2. размерМатрицы = ПрочитатьИзКонсоли()

3. Спросить Ориентированность графа

4. графОриентированный = ПрочитатьИзКонсоли()

5. Спросить у пользователя Взвешенный ли граф

6. графВзвешен = ПрочитатьИзКонсоли()

7. матрица = СгенерироватьМатрицу(размерМатрицы , графОриентированный, графВзвешен)

8. Вернуть матрицу.

**Псевдокод функции СгенерироватьМатрицу(размер, ориентированность, взвешенность).**

1. Матрица = ВыделитьПамятьПодМатрицу()
2. Цикл i с 0 до размер
   1. Цикл j с 0 до размер
      1. Если направленный то

матрица[i][j] = ГениророватьРандомно(взвешенность)

матрица[j][i] = матрица[i][j];

Иначе

1.1.1 матрица[i][j] = ГениророватьРандомно(взвешенность)

3. Вернуть Матрица

**Псевдокод функции ГениророватьРандомно(взвешенность)**

1. Если взвешенный то
   1. Вернуть СгенерироватьЧисло() & 1;
2. Иначе
   1. Вернуть СгенерироватьЧисло() % 20;

**Псевдокод функции АлгоритмДейкстры(Матрица, вершина)**

1. количествоВершин = Матрица.Количество()
2. МассивДистанций = выделить память(количествоВершин)
3. МассивПосещенныхВершин = выделить память(количествоВершин)
4. МассивДистанций[вершина] = 0
5. Цикл i от 0 до количествоВершин
   1. Вершина = ВыбратьВершинуСМинимальнымРасстоянием
   2. МассивПосещенныхВершин[Вершина] = true

5.2.1 Цикл j от 0 до количествоВершин

5.2.2 Если Длинна неравна максимальному значению числа И

5.2.3 Есть путь И Вершина не посещена   
5.2.4 ТО МассивДистанций[j] = МассивДистанций[j]+ матрица[i][j]

1. Закончить АлгоритмДейкстры

# 4 Описание программы

Для написания данной программы использован язык программирования C#. Язык программирования C# - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря своему стилю и диапазону использования.

Проект был создан в виде консольного приложения .NetCore

Работа программы начинается с запроса – выбрать действие в меню. Затем дается два способа заполнения матрицы: ручной и автоматический. После получения необходимой информации, программа выводит на экран матрицу смежности, вершины и кратчайшие расстояния до вершин с заданной.

Листинг 1 – Точка входа в приложение.

public static void Main(string[] args)

{

\_random = new Random();

ConsoleFileOutput dualOutput = new ConsoleFileOutput("outPut.txt", Console.Out);

Console.SetOut(dualOutput);

var needQuiet = false;

while (!needQuiet)

{

needQuiet = PrintMenu();

Console.WriteLine("\n\n\n");

}

На листинге 1 показана точка входа в приложение. Метод Main() запускается первым при исполнении программы. В нем производится инициализация консоли и создание файла для записи результатов пользовательской работы. Бесконечный цикл в методе обеспечивает ввод пользователя до тех пор, пока не будет введена команда «Выйти» (Цифра 3 в интерфейсе пользователя).

Листинг 2 – Метод PrintMenu ()

public static bool PrintMenu()

{

Console.WriteLine("Выберите действие:");

Console.WriteLine("1 - Чтение с файла");

Console.WriteLine("2 - Ввести случайную матрицу");

Console.WriteLine("3 - Выйти");

var userInput = Console.ReadLine();

if (!int.TryParse(userInput, out int resultParsing))

{

Console.WriteLine("Команды не существует");

return false;

}

if (resultParsing > 3 || resultParsing <= 0)

{

Console.WriteLine("Команды не существует");

return false;

}

if (resultParsing == 3)

{

return true;

}

List<List<int>> matrix = new List<List<int>>();

if (resultParsing == 1)

{

try

{

matrix = ReadFromFile();

}

catch (Exception e)

{

Console.WriteLine($"Неправильный формат файла. Тип ошибки {e.Message}");

return false;

}

}

if (resultParsing == 2)

{

try

{

matrix = CreateMatrixByUserInput();

}

catch (Exception e)

{

Console.WriteLine($"Неправильный пользовательский ввод {e.Message}");

return false;

}

}

Console.WriteLine("Введите вершину с которой нчинается поиск");

int vertex;

try

{

vertex = int.Parse( Console.ReadLine());

if (vertex > matrix.Count || vertex < 0)

{

throw new AggregateException("Некорректный Ввод вершины");

}

}

catch (Exception e)

{

Console.WriteLine(e);

return false;

}

PrintMatrix(matrix);

DijkstraAlgo(matrix, vertex);

return false;

}

В листинге 2 показан метод PrintMenu. В нем происходит осуществление пользовательского ввода и выбор нужного поведения программы. Для предотвращения ошибки любой пользовательский ввод валидируется на предмет ошибок. При возникновении таких ошибок, они обрабатываются, пользователю выдается соответствующее сообщение об ошибке. В методе формируется матрица смежности по заданным пользователем условиям (ввод из файла, ввод случайно матрицы с дополнительными параметрами). Для сформированной матрицы смежности в конце метода вызывается алгоритм Дейкстры и печатается результат его исполнения.

Листинг 3 – Алгоритм Дейкстры

public static void DijkstraAlgo(List<List<int>> graph, int source)

{

int verticesCount = graph.Count;

int[] distance = new int[verticesCount];

bool[] shortestPathTreeSet = new bool[verticesCount];

for (int i = 0; i < verticesCount; ++i)

{

distance[i] = int.MaxValue;

shortestPathTreeSet[i] = false;

}

distance[source] = 0;

for (int count = 0; count < verticesCount - 1; ++count)

{

int u = MinimumDistance(distance, shortestPathTreeSet, verticesCount);

shortestPathTreeSet[u] = true;

for (int v = 0; v < verticesCount; ++v)

{

if (!shortestPathTreeSet[v] && graph[u][v] != 0 && distance[u] != int.MaxValue &&

distance[u] + graph[u][v] < distance[v])

{

distance[v] = distance[u] + graph[u][v];

}

}

}

Print(distance, verticesCount);

}

В Листинге 3 представлен алгоритм Дейкстры. На вход алгоритму подается матрица смежности, представленная в языке C# одним вложенным списком тип List<List<int>>. Вторым параметром передается вершина, с которой мы начинаем алгоритм. В алгоритме мы задаем два массива: для определения дистанций и уже пройденных путей, массивы определённы типом int[] и bool[] соответственно. В конце алгоритма происходит печать на экран всех вершин и дистанций от целевой вершины.

# 5 Тестирование

Среда разработки IDE JetBrains Rider предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

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, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и варианта ввода элементов.

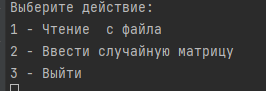


Рисунок 15 – меню приложения.

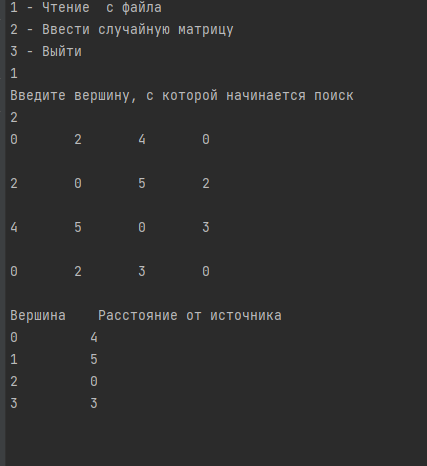


Рисунок 16 – чтение матрицы смежности из файла.

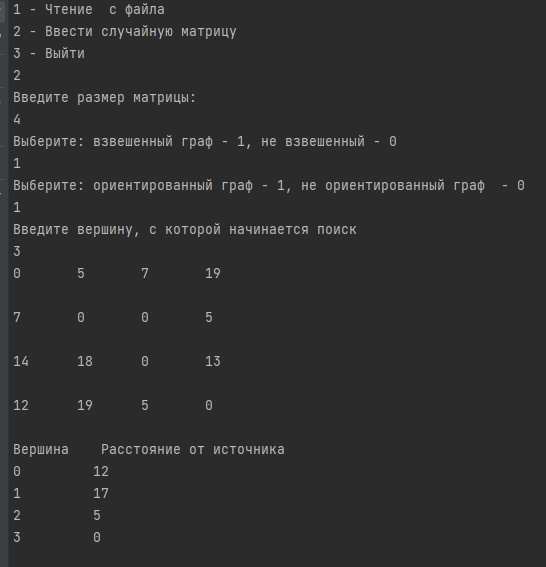


Рисунок 17 – создание матрицы с пользовательскими параметрами.

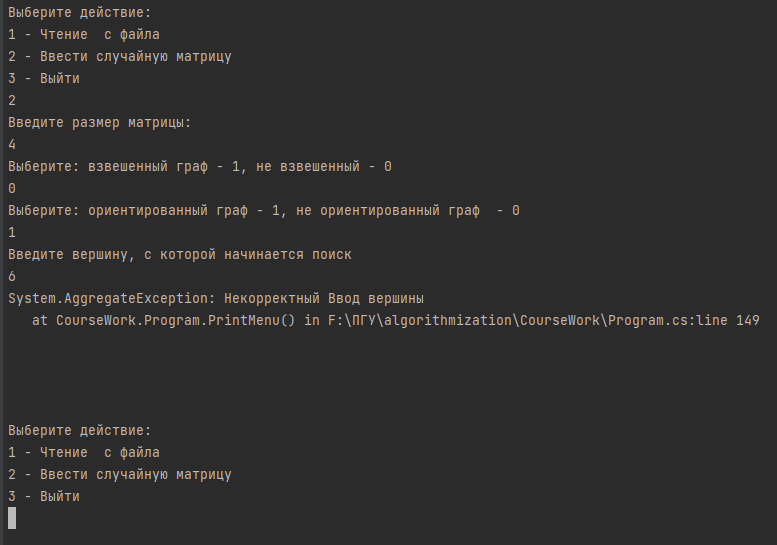


Рисунок 18 – проверка пользовательского ввода.

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщение о вводе количества вершин в графе | Верно |
| Ввод значений | Для каждого ребра в графе запрашивается значение. Также запрашивается начальная вершина. | Верно |
| Вывод матрицы смежности | Матрица выводится корректно при любых значениях | Верно |
| Ошибка при вводе | Вывод сообщение о некорректном вводе элемента матрицы и возможность повторного ввода | Верно |
| Вывод кратчайшего пути | Выводится максимально короткий путь до каждой вершины | Верно |

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

# 6 Ручной расчет задачи

Убедимся в корректности работы алгоритма по примеру (см. рисунок 19).

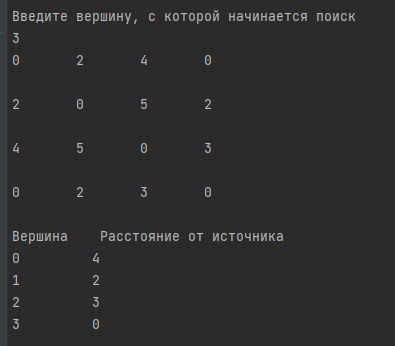


Рисунок 19 – пример работы алгоритма.

Возьмём вершину №3. Расстояние до вершины №1 = 2, №2=3.

Вершина №1: №0 = 2, №2 = 5, №3 не учитываем.

Вершина №0: 0 + 2+ 2 = 4.

Изобразим результат графически (см. рисунок 20).

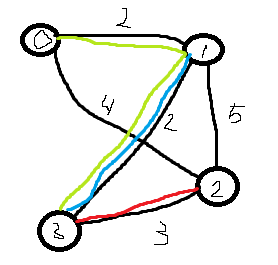


Рисунок 20 – граф, построенный по матрице смежности.

Сравним остальные вершины:  
№0=4 ; 1 2  
№1=2 ; 0 2 3  
№2=3 ; 0 1 3  
№3=0 ; 1 2

Результаты полностью совпали. Программа работает правильно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Дейкстры для поиска кратчайших расстояний и кратчайшего пути в IDE JetBrains Rider.

При выполнении данной курсовой были получены навыки разработки программ и освоены приёмы создания максимально эффективных алгоритмов. Приобретены навыки по созданию сложных алгоритмов на языке C#. Углублены знания языка программирования C#.

Недостатком данной программы является простейший пользовательский интерфейс, связанный с работой в консоли, а также ограниченные возможности пользователя в взаимодействии с данным алгоритмом.

Программа имеет небольшой, но верно работающий функционал.

# Список литературы

1. <https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры>

# Приложение А.

Листинг программы.

using System;  
using System.Collections.Generic;  
using System.IO;  
using System.Linq;  
using MultiConsoleOutput;  
  
namespace CourseWork  
**{** public class Program  
 {  
 private static Random *\_random*;  
  
 private static void PrintMatrix(List<List<int>> matrix)  
 {  
 for (int i = 0; i < matrix.Count; i++)  
 {  
 for (int j = 0; j < matrix[i].Count; j++)  
 {  
 Console.Write($"{matrix[i][j]}\t");  
 }  
  
 Console.WriteLine("\n");  
 }  
 }  
  
 private static bool ValidateMatrix(List<List<int>> matrix)  
 {  
 var countColumns = matrix.Count;  
 for (int i = 0; i < countColumns; i++)  
 {  
 if (matrix[i].Count != countColumns)  
 {  
 return false;  
 }  
 }  
  
 for (int i = 0; i < countColumns; i++)  
 {  
 if (matrix[i][i] != 0)  
 {  
 return false;  
 }  
 }  
  
 for (int i = 0; i < countColumns; i++)  
 {  
 for (int j = 0; j < countColumns; j++)  
 {  
 if (matrix[i][j] != matrix[j][i])  
 {  
 return false;  
 }  
 }  
 }  
  
 return true;  
 }  
  
 private static List<List<int>> ReadFromFile()  
 {  
 List<List<int>> matrix = new List<List<int>>();  
 using (var reader = new StreamReader("./matrix.csv"))  
 {  
 while (!reader.EndOfStream)  
 {  
 var stringLine = reader.ReadLine();  
 var values = stringLine.Split(',');  
 List<int> line = new List<int>();  
 foreach (var value in values)  
 {  
 var parsedValue = int.Parse(value);  
 line.Add(parsedValue);  
 }  
  
 matrix.Add(line);  
 }  
 }  
  
 if (!ValidateMatrix(matrix))  
 {  
 throw new ArgumentException();  
 }  
  
 return matrix;  
 }  
  
 public static bool PrintMenu()  
 {  
 Console.WriteLine("Выберите действие:");  
 Console.WriteLine("1 - Чтение с файла");  
 Console.WriteLine("2 - Ввести случайную матрицу");  
 Console.WriteLine("3 - Выйти");  
  
 var userInput = Console.ReadLine();  
  
 if (!int.TryParse(userInput, out int resultParsing))  
 {  
 Console.WriteLine("Команды не существует");  
 return false;  
 }  
  
 if (resultParsing > 3 || resultParsing <= 0)  
 {  
 Console.WriteLine("Команды не существует");  
 return false;  
 }  
  
 if (resultParsing == 3)  
 {  
 return true;  
 }  
  
 List<List<int>> matrix = new List<List<int>>();  
 if (resultParsing == 1)  
 {  
 try  
 {  
 matrix = ReadFromFile();  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine($"Неправильный формат файла. Тип ошибки {e.Message}");  
 return false;  
 }  
 }  
  
 if (resultParsing == 2)  
 {  
 try  
 {  
 matrix = CreateMatrixByUserInput();  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine($"Неправильный пользовательский ввод {e.Message}");  
 return false;  
 }  
 }  
 Console.WriteLine("Введите вершину, с которой начинается поиск");  
 int vertex;  
  
 try  
 {  
 vertex = int.Parse( Console.ReadLine());  
 if (vertex > matrix.Count || vertex < 0)  
 {  
 throw new AggregateException("Некорректный Ввод вершины");  
 }  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 return false;  
 }  
 PrintMatrix(matrix);  
  
 DijkstraAlgo(matrix, vertex);  
 return false;  
 }  
  
 private static List<List<int>> CreateMatrixByUserInput()  
 {  
 List<List<int>> matrix;  
 Console.WriteLine("Введите размер матрицы: ");  
 var userMatrixSize = int.Parse(Console.ReadLine());  
  
 Console.WriteLine("Выберите: взвешенный граф - 1, не взвешенный - 0");  
 var userNumberInput = int.Parse(Console.ReadLine());  
  
 Console.WriteLine("Выберите: ориентированный граф - 1, не ориентированный граф - 0 ");  
 var userSelectedOrintationGrapth = int.Parse(Console.ReadLine());  
  
 if (userNumberInput < 0 || userNumberInput > 1)  
 {  
 throw new ArgumentException("Пользователь ввел недопустимые цифры");  
 }  
  
 if (userSelectedOrintationGrapth < 0 || userSelectedOrintationGrapth > 1)  
 {  
 throw new ArgumentException("Пользователь ввел недопустимые цифры");  
 }  
  
 Action<int, int, List<List<int>>, int> orintationAction;  
  
 if (userSelectedOrintationGrapth == 0)  
 {  
 orintationAction = FillNotOrientedGraph;  
 }  
 else  
 {  
 orintationAction = FillOrientedGraph;  
 }  
  
  
 if (userNumberInput == 1)  
 {  
 matrix = FillMatrixRandom(userMatrixSize, GenerateAnyRandomNumber, orintationAction);  
 }  
 else  
 {  
 matrix = FillMatrixRandom(userMatrixSize, GenerateRandomZeroOreOne, orintationAction);  
 }  
  
  
 return matrix;  
 }  
   
 private static List<List<int>> FillMatrixRandom(int inputMatrixSize, Func<int> generator,  
 Action<int, int, List<List<int>>, int> fillMatrix)  
 {  
 List<List<int>> matrix = new List<List<int>>();  
  
 for (int i = 0; i < inputMatrixSize; i++)  
 {  
 List<int> line = new List<int>();  
 line.AddRange(Enumerable.Range(0, inputMatrixSize));  
 matrix.Add(line);  
 }  
  
  
 for (int i = 0; i < inputMatrixSize; i++)  
 {  
 for (int j = 0; j < inputMatrixSize; j++)  
 {  
 int randomNumber = generator.Invoke();  
 fillMatrix(i, j, matrix, randomNumber);  
 }  
 }  
  
 return matrix;  
 }  
  
 private static void FillOrientedGraph(int i, int j, List<List<int>> matrix, int randomNumber)  
 {  
 if (i == j)  
 {  
 matrix[i][j] = 0;  
 }  
 else  
 {  
 matrix[i][j] = randomNumber;  
 }  
 }  
  
 private static void FillNotOrientedGraph(int i, int j, List<List<int>> matrix, int randomNumber)  
 {  
 if (i == j)  
 {  
 matrix[i][j] = 0;  
 }  
 else  
 {  
 matrix[i][j] = randomNumber;  
 matrix[j][i] = randomNumber;  
 }  
 }  
  
 public static int GenerateRandomZeroOreOne()  
 {  
 return *\_random*.Next() & 1;  
 }  
  
 public static int GenerateAnyRandomNumber()  
 {  
 return *\_random*.Next() % 20;  
 }  
  
 private static int MinimumDistance(int[] distance, bool[] shortestPathTreeSet, int verticesCount)  
 {  
 int min = int.*MaxValue*;  
 int minIndex = 0;  
  
 for (int v = 0; v < verticesCount; ++v)  
 {  
 if (shortestPathTreeSet[v] == false && distance[v] <= min)  
 {  
 min = distance[v];  
 minIndex = v;  
 }  
 }  
  
 return minIndex;  
 }  
  
 public static void DijkstraAlgo(List<List<int>> graph, int source)  
 {  
 int verticesCount = graph.Count;  
 int[] distance = new int[verticesCount];  
 bool[] shortestPathTreeSet = new bool[verticesCount];  
  
 for (int i = 0; i < verticesCount; ++i)  
 {  
 distance[i] = int.*MaxValue*;  
 shortestPathTreeSet[i] = false;  
 }  
  
 distance[source] = 0;  
  
 for (int count = 0; count < verticesCount - 1; ++count)  
 {  
 int u = MinimumDistance(distance, shortestPathTreeSet, verticesCount);  
 shortestPathTreeSet[u] = true;  
  
 for (int v = 0; v < verticesCount; ++v)  
 {  
 if (!shortestPathTreeSet[v] && graph[u][v] != 0 && distance[u] != int.*MaxValue* &&  
 distance[u] + graph[u][v] < distance[v])  
 {  
 distance[v] = distance[u] + graph[u][v];  
 }  
 }  
 }  
  
 Print(distance, verticesCount);  
 }  
  
 private static void Print(int[] distance, int verticesCount)  
 {  
 Console.WriteLine("Вершина Расстояние от источника");  
  
 for (int i = 0; i < verticesCount; ++i)  
 Console.WriteLine("{0}\t {1}", i, distance[i]);  
 }  
  
   
 public static void Main(string[] args)  
 {  
 *\_random* = new Random();  
 ConsoleFileOutput dualOutput = new ConsoleFileOutput("outPut.txt", Console.Out);   
 Console.SetOut(dualOutput);  
 var needQuiet = false;  
 while (!needQuiet)  
 {  
 needQuiet = PrintMenu();  
 Console.WriteLine("\n\n\n");  
 }  
 }  
 }

}