Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования

«Омский государственный технический университет»

|  |  |
| --- | --- |
| Факультет (институт) | *Информационных технологий и компьютерных систем* |
|  |  |
| Кафедра | *Прикладная математика и фундаментальная информатика* |
|  |  |

**Расчетно–графическая работа**

|  |  |
| --- | --- |
| по дисциплине | ***Алгоритмизация и программирование*** |
|  |  |
| на тему | Программная реализация задач |

Пояснительная записка

|  |  |
| --- | --- |
| **Шифр проекта** | 020–РГР–02.03.02–№ 02 – ПЗ |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Студента** | | Бакланова Даниил Леонидовича | | | | | |
|  |  |  |  | | фамилия, имя, отчество полностью | | | | | |
|  |  |  | Курс | 1 |  | Группа | | МО–241 | | |
|  |  |  |  |  |  | |  |  | |  |
|  | | | **Направление (специальность)** | | | | | ***02.03.03*** | | |
|  | | | Математическое обеспечение и администрирование информационных систем | | | | | | | |
|  |  |  | код, наименование | | | | | | | |
|  |  |  | Руководитель | | ***ст. преподаватель*** | | | | | |
|  |  |  | ученая степень, звание | | | | | |
|  |  |  | ***Федотова И.В.*** | | | | | | | |
|  |  |  | фамилия, инициалы | | | | | | | |
|  |  |  | Выполнил | |  | | | | | |
|  |  |  | дата, подпись студента | | | | | |
|  |  |  |  | | | | | | | |
|  |  |  | **Работа защищена с количеством баллов** | | | | | |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | дата, подпись руководителя |  |  |  |

Омск 2025

**Содержание**

[Введение 4](#_Toc199281490)

[Постановка задачи «Города» 5](#_Toc199281491)

[Ход решения задачи «Города» 6](#_Toc199281492)

[Постановка задачи «Минное поле» 10](#_Toc199281493)

[Ход решения задачи «Минное поле» 11](#_Toc199281494)

[Постановка задачи «Расстояние между городами» 15](#_Toc199281495)

[Решение задачи «Расстояние между городами» 16](#_Toc199281496)

[Постановка задачи «из дискретной математики» 19](#_Toc199281497)

[Ход решения задачи «из дискретной математики» 20](#_Toc199281498)

[Заключение 22](#_Toc199281499)

[Список литературы 23](#_Toc199281500)

# Введение

Наша жизнь всё сильнее связывается с вычислительной техникой, которая автоматизируют различные процессы и упрощают работу, а потому важно быстро и правильно доносить свои мысли до неё. Компьютеры понимают только последовательность из нулей и единиц, а не привычный нам язык, и любая команда, поданная человеком, должна принимать именно такую форму. Для этого придумали языки программирования, которые способны наладить диалог между человеком и машиной. Человек по определенным правилам пишет команды, которые автоматически переводятся в машинный код. Но спектр задач у компьютера огромный, а потому невозможно придумать идеальный язык программирования, который подойдет под все ситуации и будет максимально эффективным. Поэтому появилось множество языков, которые эффективны в своей области. Они постепенно совершенствовались и упрощались, повышая уровень абстракции, чтобы было легче писать программы. Так мы прошли путь от Fortran к C#, JavaScript, Python и другим.

С# является высокоуровневым объектно-ориентированным языком программирования, что делает его относительно простым в изучении и использовании. Он унаследовал лучшие черты от своих предшественников: с и с++ – что делает его эффективным в работе с аппаратной частью устройства, но при этом был модифицирован: в него добавили автоматическую сборку мусора – контроль памяти и её освобождение от неиспользуемых объектов, что значительно упрощает жизнь программисту.

Целью данной расчётно-графической работы является закрепление основ C#, а также применение его для решения практических задач. В ходе работы будут изучены функции, методы работ с ними, чтение и запись файлов. А также будут задействованы основные алгоритмические структуры, и будет уделено внимание тестированию и отладке кода.

# Постановка задачи «Города»

В некоторой стране население живет в городах. Города соединены дорогами. Каждая дорога начинается в одном городе и заканчивается в другом. Длины всех дорог известны. Из любого города через систему дорог можно добраться до любого другого города.

Будем называть **расстоянием** между двумя городами длину кратчайшего пути между ними.

Требуется найти наибольшее расстояние между городами в стране.

**Входной файл**

Первая строка содержит два целых числа **N** и **M**, разделенных пробелом - количество городов и дорог (2  **N**  400, 1  **M**  100000).

Следующие **M** строк содержат по три целых числа **I**, **J, L** - номера городов, соединенных дорогой (1  **I**, **J**  **N**) и длина дороги (1  **L**  1000).

**Выходной файл**

Содержит единственное целое число - наибольшее расстояние между городами в стране.

# Ход решения задачи «Города»

Решение задачи сводится к нахождению самого длинного пути из кратчайших. Самым простым и эффективным способом будет обработка матрицы смежности с помощью алгоритма Флойда.

Сперва необходимо прочитать данные из файла. За это отвечает функция GetData, которая принимает в качестве аргумента путь к текстовому файлу (рисунок 1). На выходе формируется матрица смежности графа.

Следующим этапом будет, непосредственно, обработка матрицы с помощью алгоритма Флойда (рисунок 2). На выходе получается изменённая матрица смежности.

Окончательным этапом станет поиск наибольшего значения в матрице. Это делается с помощью функции GetMax, которая принимает матрицу и выдает число (рисунок 3).

Эти 3 этапа обработки образуют функцию ProcessTest, которая и является решением алгоритмом решения задачи (рисунок 4). На вход подается путь к файлу.

Помимо реализации решения, так же была сделана система автоматического тестирования (рисунок 5). Каждая задача представляет собой отдельную папку, а там лежит программа, файлы окружения, и папка с данными, в которой лежат еще 3 папки: test input, test output, my answer. В них лежат исходные тестовые данные, правильные ответы, и мом ответы, которая программа генерирует и записывает в файлы. Остается только последовательно считать данные, обработать их, записать в файл, и сравнить 2 файла: мой ответ и правильный. Если файлы полностью совпали, то выдается сообщение об прохождении теста, иначе сообщение об ошибке.

static int[,] GetData(string filePath)

        {

            string[] lines = File.ReadAllLines(filePath);

            string[] firstLine = lines[0].Split();

            int vertexCount = int.Parse(firstLine[0]);

            int edgeCount = int.Parse(firstLine[1]);

            int[,] data = new int[vertexCount, vertexCount];

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

            {

                for (int j = 0; j < vertexCount; j++)

                {

                    data[i, j] = (i == j) ? 0 : int.MaxValue;

                }

            }

            for (int line = 1; line <= edgeCount; line++)

            {

                string[] parts = lines[line].Split();

                int vertex1 = int.Parse(parts[0]) - 1;

                int vertex2 = int.Parse(parts[1]) - 1;

                int weight = int.Parse(parts[2]);

                if (weight < data[vertex1, vertex2])

                {

                    data[vertex1, vertex2] = weight;

                    data[vertex2, vertex1] = weight;

                }

            }

            return data;

        }

Рисунок 1– код функции считывания данных

static int[,] AlgorithmFloida(int[,] data)

        {

            int n = data.GetLength(0);

            for (int baseVertex = 0; baseVertex < n; baseVertex++)

            {

                for (int firstVertex = 0; firstVertex < n; firstVertex++)

                {

                    for (int secondVertex = 0; secondVertex < n; secondVertex++)

                    {

                        if (data[firstVertex, baseVertex] != int.MaxValue && data[baseVertex, secondVertex] != int.MaxValue)

                        {

                            int newDistance = data[firstVertex, baseVertex] + data[baseVertex, secondVertex];

                            if (data[firstVertex, secondVertex] > newDistance)

                            {

                                data[firstVertex, secondVertex] = newDistance;

                            }

                        }

                    }

                }

            }

            return data;

        }

Рисунок 2– код функции агоритма флойда

static int GetMax(int[,] data)

        {

            int mx = 0;

            for (int firstVertex=0; firstVertex < data.GetLength(0); firstVertex++)

            {

                for (int secondVertex=0; secondVertex < data.GetLength(0); secondVertex++)

                {

                    if (data[firstVertex,secondVertex]>mx) mx = data[firstVertex, secondVertex];

                }

            }

            return mx;

        }

Рисунок 3 – код функции поиска наибольшего значения в матрице

static string[] ProcessTest(string path)

        {

            int[,] data = GetData(path);

            data = AlgorithmFloida(data);

            int mx = GetMax(data);

            string[] answer = { Convert.ToString(mx) };

            return answer;

        }

Рисунок 4 – код функции решения задачи

static bool RunTests(string inputDir, string correctDir, string myDir)

        {

            bool allCorrect = true;

            foreach (var inputFile in Directory.GetFiles(inputDir).OrderBy(f => f))

            {

                string fileName = Path.GetFileName(inputFile);

                string correctFile = Path.Combine(correctDir, fileName.Replace("input", "output"));

                string myFile = Path.Combine(myDir, fileName);

                string[] myAnswer = ProcessTest(inputFile);

                File.WriteAllLines(myFile, myAnswer);

                string[] correctAnswer = File.ReadAllLines(correctFile);

                if (!myAnswer.SequenceEqual(correctAnswer))

                {

                    Console.WriteLine($"тест {fileName} не пройден");

                    allCorrect = false;

                }

                else

                {

                    Console.WriteLine($"тест {fileName} пройден");

                }

            }

            return allCorrect;

        }

static void Main(string[] args)

        {

            bool allTestsPassed = RunTests("test/test input", "test/test output", "test/my answer");

            Console.WriteLine(allTestsPassed ? "Все тесты пройдены" : "Обнаружены несоответствия");

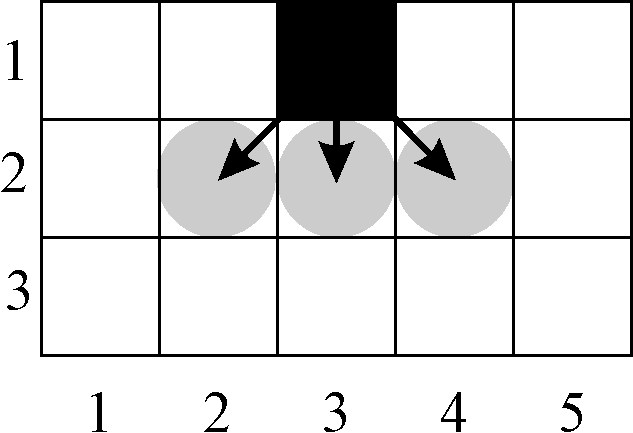
        }

Рисунок 5 – код функции автоматического тестирования

# Постановка задачи «Минное поле»

Минное поле представляет собой прямоугольник, разделенный на квадратные зоны одинакового размера. С помощью средств современной разведки было выяснено, сколько мин заложено в каждой зоне. Для разминирования каждой мины требуется одинаковое время. Для прохода по зоне, необходимо разминировать все мины на ней.

Проход между зонами возможен в следующих направлениях:

,

т.е. в следующий ряд прямо и по диагоналям.

Требуется найти один из вариантов прохода в минном поле, требующий минимального времени для разминирования.

**Примечания:**

* размер поля **N**\***M**, 0 < **N** ≤ 200, 0 < **M** ≤ 400;
* количество мин в каждой зоне от 0 до 100.

**Входной файл** содержит:

* в первой строке числа **N** и **M**, разделенные пробелом (количество рядов на минном поле и зон в ряде соответственно);
* в следующих **N** строках содержится по **M** целых чисел разделенных пробелом – количество мин в зонах **i**-го ряда (1 ≤ **i** ≤ **N**).

**Выходной файл** должен содержать **N** строк. В каждой строке указывается номер зоны в **i**-м ряду (1 ≤ **i** ≤ **N**), через которую лежит найденный проход.

# Ход решения задачи «Минное поле»

Решение этой задачи делится на 3 этапов: 1 – считывание данных, 2 – проход по полю и нахождение кратчайших путей для каждой клетки 3 – восстановление маршрута.

Для начал считаем данные с помощью функции GetData, которая принимает путь к файлу, и выдает двумерный целочисленный массив – исходные данные, где индекс в 1 измерении является номером строки, а индекс во 2 измерении – номером столбца (рисунок 6).

В первом этапе мы идем «снизу-вверх». Возьмем для примера i-ую строку в j-ом ряду. Минимальное значение клетки строится по правилу: значение в клетке + минимальное значение «соседей». Соседи ищутся по следующему правилу: в строке i-1 всегда есть сосет над клеткой (j ряд) и, если клетка не является крайней, то рассматриваются j-1 и j+1. В противном случае одного из «соседа» не будет. Таким образом происходит 1 итерация алгоритма – построение минимальных значений в каждой клетке (рисунок 7).

На втором этапе мы идем с конца массива. Мы ищем клетку с минимальным номером и заносим её индекс в наш путь. Теперь мы идем в обратную строну по тому же принципу выбора «соседей», (рисунок 8). На выходе получаем массив из индексов клеток, где индекс в массиве – номер строки, а значение по индексу – номер ряда.

Эти 3 этапа обработки образуют функцию ProcessTest, которая и является решением алгоритмом решения задачи (рисунок 9). На вход подается путь к файлу.

Помимо реализации решения, так же была сделана система автоматического тестирования (рисунок 10). Каждая задача представляет собой отдельную папку, а там лежит программа, файлы окружения, и папка с данными, в которой лежат еще 3 папки: test input, test output, my answer. В них лежат исходные тестовые данные, правильные ответы, и мом ответы, которая программа генерирует и записывает в файлы. Остается только последовательно считать данные, обработать их, записать в файл, и сравнить 2 файла: мой ответ и правильный. Если файлы полностью совпали, то выдается сообщение об прохождении теста, иначе сообщение об ошибке.

static int[,] GetData(string filePath)

        {

            string[] lines = File.ReadAllLines(filePath);

            string[] firstLine = lines[0].Split();

            int linesCount = int.Parse(firstLine[0]);

            int fieldsCount = int.Parse(firstLine[1]);

            int[,] data = new int[linesCount, fieldsCount];

            for (int line = 1; line < 1 + linesCount; line++)

            {

                string[] lineData = lines[line].Split(' ');

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

                {

                    if (i < lineData.Length && !string.IsNullOrEmpty(lineData[i]))

                    {

                        data[line - 1, i] = int.Parse(lineData[i]);

                    }

                    else

                    {

                        data[line - 1, i] = int.MaxValue;

                    }

                }

            }

            return data;

        }

Рисунок 6 – код функции считывания данных

static int[,] GetPathData(int[,] data)

        {

            int linesCount = data.GetLength(0);

            int fieldsCount = data.GetLength(1);

            int[,] pathData = new int[data.GetLength(0), data.GetLength(1)];

            for (int field = 0; field < fieldsCount; field++)

            {

                pathData[0, field] = data[0, field];

            }

            for (int line = 1; line < linesCount; line++)

            {

                for (int field = 0; field < fieldsCount; field++)

                {

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

                    fieldsBefore.Add(pathData[line - 1, field]);

                    if (field > 0)

                    {

                        fieldsBefore.Add(pathData[line - 1, field - 1]);

                    }

                    if (field < fieldsCount - 1)

                    {

                        fieldsBefore.Add(pathData[line - 1, field + 1]);

                    }

                    pathData[line, field] = data[line, field] + fieldsBefore.Min();

                }

            }

            return pathData;

        }

Рисунок 7 – код функции поиска минимального значения в каждой клетке

static int[] FindMinPath(int[,] pathData)

        {

            int linesCount = pathData.GetLength(0);

            int fieldsCount = pathData.GetLength(1);

            int[] answer = new int[pathData.GetLength(0)];

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

            for (int field = 0; field < fieldsCount; field++)

            {

                currentLine.Add(pathData[linesCount - 1, field]);

            }

            answer[linesCount - 1] = currentLine.FindIndex(x => x == currentLine.Min());

            for (int line = linesCount - 2; line >= 0; line--)

            {

                int current\_field = answer[line + 1];

                int minValue = int.MaxValue;

                int bestField = current\_field;

                for (int field = Math.Max(0, current\_field - 1); field <= Math.Min(fieldsCount - 1, current\_field + 1); field++)

                {

                    if (pathData[line, field] < minValue)

                    {

                        minValue = pathData[line, field];

                        bestField = field;

                    }

                }

                answer[line] = bestField;

            }

            return answer;

        }

Рисунок 8 – код функции восстановления пути

static string[] ProcessTest(string filePath)

        {

            int[,] data = GetData(filePath);

            int[,] pathData = GetPathData(data);

            int[] answer = FindMinPath(pathData);

            string[] stringAnswer = Array.ConvertAll(answer, x => (x+1).ToString());

            return stringAnswer;

        }

Рисунок 9 – код функции решения задачи

static bool RunTests(string inputDir, string correctDir, string myDir)

        {

            bool allCorrect = true;

            foreach (var inputFile in Directory.GetFiles(inputDir).OrderBy(f => f))

            {

                string fileName = Path.GetFileName(inputFile);

                string correctFile = Path.Combine(correctDir, fileName.Replace("input", "output"));

                string myFile = Path.Combine(myDir, fileName);

                try

                {

                    string[] myAnswer = ProcessTest(inputFile);

                    File.WriteAllLines(myFile, myAnswer);

                    string[] correctAnswer = File.ReadAllLines(correctFile);

                    if (!myAnswer.SequenceEqual(correctAnswer))

                    {

                        Console.WriteLine($"тест {fileName} не пройден");

                        allCorrect = false;

                    }

                    else

                    {

                        Console.WriteLine($"тест {fileName} пройден");

                    }

                }

                catch (Exception ex)

                {

                    Console.WriteLine(ex);

                }

            }

            return allCorrect;

        }

static void Main()

        {

            bool allTestsPassed = RunTests("test/test input", "test/test output", "test/my answer");

            Console.WriteLine(allTestsPassed ? "Все тесты пройдены" : "Обнаружены несоответствия");

        }

Рисунок 10 – код функции автоматического тестирования

# Постановка задачи «Расстояние между городами»

*Максимальное время: 0,2 с.*

*Максимальная память: 16 MB*

На поверхности некоторой планеты, представляющей собой идеальный шар радиуса **R**, заданы координаты двух городов в виде двух чисел - широты и долготы.

Требуется определить минимальное расстояние по поверхности планеты между этими городами.

**Примечание**

Пары (широта, долгота) уникальны.

**Входной файл**

Первая строка содержит два целых числа **S1** и **D1** - широту и долготу первого города в градусах (-90 **S1**  90; 0  **D1**  359).

Вторая строка содержит два целых числа **S2** и **D2** - широту и долготу второго города в градусах -90  **S2**  90; 0  **D2**  359).

Третья строка содержит целое число **R** - радиус планеты (1  **R**  30000).

**Выходной файл**

Должен содержать одно вещественное число - минимальное расстояние между городами по поверхности планеты, выведенное с тремя знаками после запятой.

# Решение задачи «Расстояние между городами»

Первым этапом происходит считывание данных из файла в массив чисел (рисунок 11).

Затем происходит сам процесс вычисления: форматирование данных (приведение в радианы) и применение формулы гаверсинусов, для вычисления расстояние между 2 точками на сфере (рисунок 12). Для этого необходимо знать:

* радиус сферы R
* ­– широты двух точек (в радианах)
* и – долготы двух точек (в радианах)

Теперь можно воспользоваться формулой гаверсинусов:

Эти 2 этапа обработки образуют функцию ProcessTest, которая и является решением алгоритмом решения задачи (рисунок 13). На вход подается путь к файлу.

Помимо реализации решения, так же была сделана система автоматического тестирования (рисунок 14). Каждая задача представляет собой отдельную папку, а там лежит программа, файлы окружения, и папка с данными, в которой лежат еще 3 папки: test input, test output, my answer. В них лежат исходные тестовые данные, правильные ответы, и мом ответы, которая программа генерирует и записывает в файлы. Остается только последовательно считать данные, обработать их, записать в файл, и сравнить 2 файла: мой ответ и правильный. Если файлы полностью совпали, то выдается сообщение об прохождении теста, иначе сообщение об ошибке.

static int[] GetData(string filePath)

        {

            string[] lines = File.ReadAllLines(filePath);

            string[] city1 = lines[0].Split();

            string[] city2 = lines[1].Split();

            int[] data = { int.Parse(city1[0]), int.Parse(city1[1]), int.Parse(city2[0]), int.Parse(city2[1]), int.Parse(lines[2]) };

            return data;

        }

Рисунок 11 – код функции считывания данных

static string[] ProcessTest(string filePath)

        {

            int[] data = GetData(filePath);

            double lat1 = data[0] \* Math.PI / 180;

            double lon1 = data[1] \* Math.PI / 180;

            double lat2 = data[2] \* Math.PI / 180;

            double lon2 = data[3] \* Math.PI / 180;

            double R = data[4];

            double dLat = lat2 - lat1;

            double dLon = lon2 - lon1;

            double underRoot = Math.Sin(dLat / 2) \* Math.Sin(dLat / 2) +

                                Math.Cos(lat1) \* Math.Cos(lat2) \*

                                Math.Sin(dLon / 2) \* Math.Sin(dLon / 2);

            double angle = 2 \* Math.Asin(Math.Sqrt(underRoot));

            double distance = R \* angle;

            string[] answer = { distance.ToString("F3").Replace(",", ".") };

            return answer;

        }

Рисунок 12 – код функции, вычисляющее расстояние

static bool RunTests(string inputDir, string correctDir, string myDir)

        {

            bool allCorrect = true;

            foreach (var inputFile in Directory.GetFiles(inputDir).OrderBy(f => f))

            {

                string fileName = Path.GetFileName(inputFile);

                string correctFile = Path.Combine(correctDir, fileName.Replace("input", "output"));

                string myFile = Path.Combine(myDir, fileName);

                try

                {

                    string[] myAnswer = ProcessTest(inputFile);

                    File.WriteAllLines(myFile, myAnswer);

                    string[] correctAnswer = File.ReadAllLines(correctFile);

                    if (!myAnswer.SequenceEqual(correctAnswer))

                    {

                        Console.WriteLine($"тест {fileName} не пройден");

                        allCorrect = false;

                    }

                    else

                    {

                        Console.WriteLine($"тест {fileName} пройден");

                    }

                }

                catch (Exception ex)

                {

                    Console.WriteLine(ex);

                }

            }

            return allCorrect;

        }

static void Main(string[] args)

        {

            bool allTestsPassed = RunTests("test/test input", "test/test output", "test/my answer");

            Console.WriteLine(allTestsPassed ? "Все тесты пройдены" : "Обнаружены несоответствия");

        }

Рисунок 13 – код функции автоматического тестирования

# Постановка задачи «из дискретной математики»

Предположим, что вы хотите проехать из Новосибирска в Волгоград, используя магистральные шоссейные дороги, соединяющие различные города.

В построенном графе определить кратчайший путь между городами.

**Формат входных данных**

Во входном файле записано сначала число N (1<=N<=100), определявшее количество рассматриваемых городов. Затем идет число M - количество дорог в стране, далее идет описание самих дорог. Каждая дорога задается тремя числами - номерами городов, которые она соединяет и расстоянием. Все дороги двухсторонние (то есть по ним можно ездить как в одну, так и в другую сторону); между двумя городами всегда существует не более одной дороги; не существует дорог, ведущих из города в себя.

**Формат выходных данных**

На экран выведите одно число - суммарную длина маршрута или -1, если добраться невозможно.

# Ход решения задачи «из дискретной математики»

Решение данной задачи является задача поиска кратчайшего расстояния между городами. Для этой задачи я выберу алгоритм Дейкстры.

Первым этапом будет считывание данных из текстового файла. Задача представляет собой сеть из дорог, а потому её удобно решать в терминологии теории графов. Входные данные – это список дорог, по которым необходимо составить матрицу смежности. Негде не сказано, что они направленные, то мы будем их считать ребрами, поэтому матрица симметричная (рисунок 1).

Следующим этапом является реализация алгоритма Крускала (рисунок 2), а потом необходимо определить, нашелся ли путь (сравнить с изначальным значением int.MaxValue). Если да, до вывести значение, иначе вывести -1.

Выбор вершин от которой ведётся поиск пути до которой осуществляется путём указания значений переменных start и end соответственно. Итогом работы получилось число 18.

static int[,] GetData(string filePath)

        {

            string[] lines = File.ReadAllLines(filePath);

            int vertexCount = int.Parse(lines[0]);

            int edge\_count = int.Parse(lines[1]);

            int[,] data = new int[vertexCount, vertexCount];

            for (int line = 2; line < 2 + edge\_count; line++)

            {

                string[] parts = lines[line].Split(' ');

                int vertex1 = int.Parse(parts[0]) - 1;

                int vertex2 = int.Parse(parts[1]) - 1;

                int weight = int.Parse(parts[2]);

                data[vertex1, vertex2] = weight;

                data[vertex2, vertex1] = weight;

            }

            return data;

        }

Рисунок 14 – код функции считывания данных

static void Main(string[] args)

        {

            int[,] data = GetData("data.txt");

            int start = 1;

            int end = 7;

            int vertexCount = data.GetLength(0);

            int[] distances = new int[vertexCount];

            int[] vertexBefore = new int[vertexCount];

            bool[] visited\_vertex = new bool[vertexCount];

            for (int vertex = 0; vertex<vertexCount; vertex++) vertexBefore[vertex] = -1;

            for (int vertex = 0; vertex<vertexCount; vertex++) distances[vertex] = (vertex == start-1) ? 0 : int.MaxValue;

            for (int step = 0; step<vertexCount-1; step++)

            {

                int closedVertex = -1;

                int min\_distance = int.MaxValue;

                for (int vertex = 0; vertex<vertexCount; vertex++)

                {

                    if (!visited\_vertex[vertex] && distances[vertex]<min\_distance)

                    {

                        min\_distance = distances[vertex];

                        closedVertex = vertex;

                    }

                }

                if (closedVertex == -1) break;

                visited\_vertex[closedVertex] = true;

                for (int neighborVertex =  0; neighborVertex<vertexCount; neighborVertex++)

                {

                    int edge\_weight = data[closedVertex, neighborVertex];

                    if (edge\_weight == 0 || visited\_vertex[neighborVertex]) continue;

                    if (distances[closedVertex] != int.MaxValue &&

                        distances[closedVertex]+edge\_weight<distances[neighborVertex])

                        {

                            distances[neighborVertex] = distances[closedVertex]+edge\_weight;

                            vertexBefore[neighborVertex] = closedVertex;

                        }

                }

            }

            if (distances[end-1] == int.MaxValue)

            {

                Console.WriteLine(-1);

            }

            else

            {

                Console.WriteLine(distances[end-1]);

            }

        }

Рисунок 15 – код основной программы

# Заключение

В ходе выполнения работы были отработаны основы языка C# и применены знания для решения практических задач. Были изучены основные алгоритмические конструкции, методы работы с функциями и написание собственных, а также библиотека System в области работы с файлами.

В процессе выполнения работы были созданы четыре программы, показывающие особенности и возможности применения языка С#. Был получен навык тестирования и отладки кода, структурирования файловой системы и написания универсальных функций, для выполнения в разных программах.

В результате выполнения работы я получил ценные знания и навыки программирования на языке C#. Но помимо этого я так же получил знания об алгоритмах, работе с данными, функциями и файлами, что так же полезно при использовании в других языках, поскольку включает общие фундаментальные принципы. Язык С# стал хорошей точкой входа в программирование. И теперь, с полученным опытом, я дальше смогу развиваться в этом направлении.

# Список литературы

1. Мюллер Д.П. C# для чайников, 3-e издание – Диалектика, Москва, 2019 – 591 с.
2. Полное руководство по языку программирования C# 13 и платформе

.NET 9: https://metanit.com/sharp/tutorial/ (дата обращения 15.12.2024).

1. Руководство по C# – управляемый язык .NET | Microsoft Learn: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения 10.12.2024)
2. C# уроки для чайников - курсы обучения C# для начинающих с нуля на itProger: https://itproger.com/course/csharp (дата обращения 16.12.2024)
3. Первая программа на C#: https://gb.ru/blog/pervaya-programma-na-c-sharp/ (дата обращения 20.12.2024).