

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-№ 3 – ПЗ

Студента Белогривцева Андрея Дмитриевича
фамилия, имя, отчество полностью

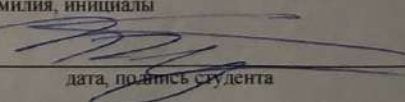
Курс 1 Группа МО-241

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 15.05.2025 
дата, подпись студента

Работа защищена с количеством баллов

15.05.2025

дата, подпись руководителя

125 баллов



Омск 2025

Содержание

Содержание	2
Введение	3
Постановка задачи «Расстояние между городами»	4
Ход решения задачи «Расстояние между городами»	5
Постановка задачи «Игра в города»	7
Ход решения задачи «Игра в города»	9
Постановка задачи «Многоугольник»	11
Ход решения задачи «Многоугольник»	12
Постановка задачи «Прямоугольник»	16
Ход решения задачи «Прямоугольник»	17
Задача из дискретной математики	19
Ход решения задачи из дискретной математики	20
Заключение	25
Список литературы	26

Введение

В наше время для решения прикладных задач активно применяются разнообразные языки программирования. Их количество очень велико (Python, C, C++, PHP и другие). Математическая сторона теоретического решения таких задач довольно сложна, и универсального алгоритма, разумеется, не существует. Требуется выявить ключевые тенденции и закономерности, изучить принципы работы механизмов, лежащих в основе каждой конкретной задачи.

Когда теоретическое описание задачи вызывает значительные трудности, применение экспериментальных и аналитических методов решения становится неэффективным и затруднительным. В таких случаях необходим инструмент, который позволит наглядно представить и решить поставленную задачу. В данной работе в качестве такого инструмента рассматривается язык программирования C#.

Постановка задачи «Расстояние между городами»

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 ($-90 \leq S_1 \leq 90$; $0 \leq D_1 \leq 359$).
- Вторая строка содержит два целых числа **S₂** и **D₂** - широту и долготу второго города в градусах ($-90 \leq S_2 \leq 90$; $0 \leq D_2 \leq 359$).
- Третья строка содержит целое число **R** - радиус планеты ($1 \leq R \leq 30000$).

Выходной файл

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

Ход решения задачи «Расстояние между городами»

```
using System;

class App
{
    public static void FindDistance(double sh1, double dl1, double sh2, double dl2,
double radius)
    {
        double sh1Rad = sh1 * Math.PI / 180;
        double dl1Rad = dl1 * Math.PI / 180;
        double sh2Rad = sh2 * Math.PI / 180;
        double dl2Rad = dl2 * Math.PI / 180;
        double deltaDl = Math.Abs(dl1Rad - dl2Rad);
        deltaDl = Math.Min(deltaDl, 2 * Math.PI - deltaDl);
        double result = Math.Sin(sh1Rad) * Math.Sin(sh2Rad) + Math.Cos(sh1Rad) *
Math.Cos(sh2Rad) * Math.Cos(deltaDl)
        result = Math.Max(-1, Math.Min(1, result));
        result = Math.Acos(result);
        result = radius * result;
        Console.WriteLine($"Расстояние между городами равно {result:F3}");
    }
    public static void Main()
    {
        Console.WriteLine("Введите широту первого города (-90 до 90):");
        double sh1 = double.Parse(Console.ReadLine());
        Console.WriteLine("Введите долготу первого города (0 до 359):");
        double dl1 = double.Parse(Console.ReadLine());
        Console.WriteLine("Введите широту второго города (-90 до 90):");
        double sh2 = double.Parse(Console.ReadLine());
        Console.WriteLine("Введите долготу второго города (0 до 359):");
        double dl2 = double.Parse(Console.ReadLine());
        Console.WriteLine("Введите радиус планеты (1 до 30000):");
        double radius = double.Parse(Console.ReadLine());

        FindDistance(sh1, dl1, sh2, dl2, radius);
    }
}
```

Ввод и вывод данных производится через консоль. В самом начале пользователю предлагается ввести в консоль широты и долготы первого и второго городов, значения которых занесутся в переменные. Также предлагается ввести радиус планеты. Далее применяется метод

«FindDistance», аргументами которого служат данные переменные. В методе производится перевод широт и долгот в радианы, а затем используется соответствующая формула (формула гаверсинуов) для подсчёта расстояния. Стоит отметить присутствие арккосинуса в формуле подсчёта => нам нужно 100% попадание в промежуток $[-1;1]$, а также учёт циклического характера долгот. Результат выводится на экран с точностью до 3 знаков после запятой.

Постановка задачи «Игра в города»

Известна игра “в города”, в которой по очереди называются уникальные названия городов, причем каждое следующее название города должно начинаться с буквы, на которую заканчивается предыдущее название города.

Будем называть цепочкой названий городов два и более названия, составленные по описанному выше принципу.

Будем называть закольцованной цепочкой названий городов такую цепочку, в которой название первого города в цепочке начинается на ту же букву, на которую заканчивается название последнего города в цепочке.

Для заданного набора уникальных названий городов требуется определить количество и длины (количество названий в цепочке) закольцованных цепочек, построенных по следующему правилу:

из исходного набора названий строится первая закольцованная цепочка максимально возможной длины;

из названий, не вошедших в первую закольцованную цепочку, строится вторая закольцованная цепочка максимально возможной длины;

из названий, не вошедших в предыдущие закольцованные цепочки, строится очередная закольцованная цепочка максимально возможной длины;

построение заканчивается, когда все названия городов использованы.

Примечание: гарантируется, что все названия городов входят в какую-нибудь цепочку.

Входной файл

Каждая строка входного файла содержит по одному названию города. Количество названий городов не превышает 30000. Название города содержит не более 15 символов. В название могут входить только маленькие латинские буквы.

Выходной файл

Первая строка должна содержать целое число **N** — количество построенных закольцованных цепочек.

Следующие N строк должны содержать длины (количество названий в цепочке) закольцованных цепочек, выведенные в порядке построения цепочек (то есть по убыванию длины).

Ход решения задачи «Игра в города»

```
using System;
using System.Collections.Generic;
using System.Linq;

class App
{
    static void Main()
    {
        Console.WriteLine("Введите количество городов");
        var n = Convert.ToInt32(Console.ReadLine());
        List<string> cities = new List<string>();
        Console.WriteLine("Вводите города");
        for (int i = 0; i < n; i++)
        {
            cities.Add(Console.ReadLine().Trim().ToLower());
        }

        List<List<string>> allChains = new List<List<string>>>();
        HashSet<int> usedIndices = new HashSet<int>();

        while (usedIndices.Count < cities.Count)
        {
            for (int i = 0; i < cities.Count; i++)
            {
                if (!usedIndices.Contains(i))
                {
                    List<string> currentChain = new List<string>();
                    currentChain.Add(cities[i]);
                    usedIndices.Add(i);

                    bool chainFlag; // флажок для расширения цепочки
                    do // построение цепочки
                    {
                        chainFlag = false;
                        char lastChar = currentChain.Last().Last();

                        for (int j = 0; j < cities.Count; j++)
                        {
                            if (!usedIndices.Contains(j) && cities[j][0] == lastChar)
                            {
                                currentChain.Add(cities[j]);
                                usedIndices.Add(j);
                                chainFlag = true;
                            }
                        }
                    } while (chainFlag);
                }
            }
        }
    }
}
```

```

        break;
    }
}
} while (chainFlag);

if (currentChain.Count >= 2 && currentChain.First()[0] ==
currentChain.Last().Last())
{
    allChains.Add(currentChain);
}
else
{
    foreach (var city in currentChain.Skip(1))
    {
        int index = cities.IndexOf(city);
        usedIndices.Remove(index);
    }
}
}
}
}
}
allChains.Sort((a, b) => b.Count.CompareTo(a.Count));
Console.WriteLine($"Количество закольцованных цепочек равно
{allChains.Count}");
Console.WriteLine("\nДлины закольцованных цепочек:\n");
foreach (var chain in allChains)
{
    Console.WriteLine(chain.Count);
}
}
}

```

Ввод и вывод данных осуществляется через консоль. В начале программы формируется список городов (пользователю предлагается ввести количество городов). Происходит считывание городов со списка: берётся первый неиспользованный город, а затем программа пытается расширить цепочку, добавляя города, начинающиеся на последнюю букву последнего добавленного города. Если расширить цепочку уже невозможно, то программа отвечает на вопрос: закольцована ли цепочка? В самом конце осуществляется сортировка цепочек по длине. Результат выводится на экран.

Постановка задачи «Многоугольник»

Дано N точек ($3 \leq N \leq 1000$) с целочисленными координатами, некоторые (или все) из которых являются вершинами выпуклого многоугольника, а остальные (или ни одной) находятся внутри него. Любые три точки не лежат на одной прямой. Определить площадь выпуклого многоугольника.

Модули координат не превосходят 30000. Считается, что кроме заданных вершин, других вершин нет.

Входной файл

- Первая строка содержит число точек N .
- Следующие N строк содержат по два числа x_i и y_i , разделенных пробелом – координаты точек.

Выходной файл

Содержит площадь многоугольника с тремя знаками после запятой.

Ход решения задачи «Многоугольник»

```
using System;
using System.Collections.Generic;
using System.Linq;
class App
{
    public static List<(float, float)> AndrewAlgorith(List<float> xCoords,
List<float> yCoords) // алгоритм Эндрю (нахождение выпуклой оболочки)
    {
        if (xCoords == null || yCoords == null)
            throw new ArgumentNullException("Списки координат не могут быть
null");
        if (xCoords.Count != yCoords.Count)
            throw new ArgumentException("Количество X и Y координат должно
совпадать.");
        var points = new List<(float x, float y)>();
        for (int i = 0; i < xCoords.Count; i++)
        {
            points.Add((xCoords[i], yCoords[i]));
        }
        points = points.Distinct().ToList(); // удаляем одинаковые точки
        if (points.Count <= 3)
            return points;
        points = points.OrderBy(p => p.x).ThenBy(p => p.y).ToList(); //
сортировка точек по X и Y

        var hull = new List<(float x, float y)>();

        for (int i = 0; i < points.Count; i++) // верхняя часть в.о
        {
            while (hull.Count >= 2 && VectorMulti(hull[hull.Count - 2],
hull[hull.Count - 1], points[i]) <= 0) // проверяем есть ли невыпуклый угол
            {
                hull.RemoveAt(hull.Count - 1);
            }
            hull.Add(points[i]);
        }

        int lowerHullStart = hull.Count;
        for (int i = points.Count - 2; i >= 0; i--) // нижняя часть в.о
        {
            while (hull.Count >= lowerHullStart && VectorMulti(hull[hull.Count - 2],
hull[hull.Count - 1], points[i]) <= 0)
            {
                hull.RemoveAt(hull.Count - 1);
            }
            hull.Add(points[i]);
        }
    }
}
```

```

        hull.RemoveAt(hull.Count - 1);
    }
    hull.Add(points[i]);
}

if (hull.Count > 1 && hull[0] == hull[hull.Count - 1]) // чистка последней
точки
    hull.RemoveAt(hull.Count - 1);
return hull;
}
private static float VectorMulti((float x, float y) O, (float x, float y) A, (float x,
float y) B)
{
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}
public static double FindSquare(List<float> list1, List<float> list2, int n)
{
    if (list1 == null || list2 == null)
        throw new ArgumentNullException("Списки координат не могут быть
null");
    if (n <= 0)
        throw new ArgumentException("Количество вершин должно быть
положительным");
    var convexHull = AndrewAlgorith(list1, list2);
    int hullCount = convexHull.Count;
    if (hullCount < n)
    {
        Console.WriteLine($"Удалено {n - hullCount} вершин, находящихся
внутри выпуклой оболочки");
    }
    if (hullCount < 3)
    {
        Console.WriteLine("Для вычисления площади нужно минимум 3
точки.");
        return 0;
    }
    float sum1 = 0;
    float sum2 = 0;
    for (int i = 0; i < hullCount; i++)
    {
        int next = (i + 1) % hullCount;
        sum1 += convexHull[i].Item1 * convexHull[next].Item2;
        sum2 += convexHull[i].Item2 * convexHull[next].Item1;
    }
    double result = 0.5 * Math.Abs(sum1 - sum2);

```

```

        Console.WriteLine($"Получившаяся площадь: {result}");
        return result;
    }
    public static void Main()
    {
        Console.WriteLine("Задача 'Многоугольник'\nВведите количество вершин
в многоугольнике");
        var n = Convert.ToInt32(Console.ReadLine());
        if (n < 3)
        {
            Console.WriteLine("Многоугольник должен иметь минимум 3
вершины");
            return;
        }
        List<float> Xs = new List<float>();
        List<float> Ys = new List<float>();
        for (int i = 0; i < n; i++)
        {
            Console.WriteLine($"Введите x координату {i + 1} вершины");
            var x = float.Parse(Console.ReadLine());
            Console.WriteLine($"Введите y координату {i + 1} вершины");
            var y = float.Parse(Console.ReadLine());
            Xs.Add(x);
            Ys.Add(y);
        }
        FindSquare(Xs, Ys, n);
    }
}

```

Ввод и вывод данных производится через консоль. Основная проблема задачи заключается в возможном появлении лишних точек внутри многоугольника. В коде программы используется алгоритм Эндрю, который строит выпуклую оболочку многоугольника (такое множество точек, что оно выпуклое и все точки фигуры также лежат в нём, то есть любой отрезок, соединяющий две точки данного множества не содержит точек, не лежащих в этом множестве). После ввода пользователем количества точек и их координат происходит подготовка: объединение координат в список точек, удаление дубликатов и возвращение точек, если их ≤ 3 . Далее происходит сортировка точек по X при возрастании, а при равенстве – по Y. Алгоритм строит верхнюю и нижнюю оболочку, основываясь на том, образует ли новая точка выпуклый

угол. В конце происходит удаление последней точки (она дублирует первую) и возврат точек выпуклой оболочки. Далее соответствующие координаты подставляются в формулу площади многоугольника.

Постановка задачи «Прямоугольник»

Имеется прямоугольная область размером $N \times M$ клеток, некоторые из которых закрашены.

Требуется найти прямоугольник максимальной площади, не содержащий закрашенные клетки. Стороны прямоугольника должны быть параллельны краям прямоугольной области.

Входной файл содержит:

- в первой строке два числа N и M – число столбцов и строк прямоугольной области соответственно ($1 \leq N, M \leq 250$);
- во второй строке число K – количество закрашенных клеток ($0 \leq K \leq (N \cdot M - 1)$);
- следующие K строк содержат по 2 числа X и Y – координаты (столбец и строка соответственно) закрашенной клетки.

Выходной файл должен содержать площадь найденного прямоугольника.

Ход решения задачи «Прямоугольник»

using System;

class App

{

static void Main()

{

Console.WriteLine("Введите размеры области (N M)");

string[] dimensions = Console.ReadLine().Split();

int N = int.Parse(dimensions[0]); // ширина

int M = int.Parse(dimensions[1]); // высота

Console.WriteLine("Введите количество закрашенных клеток (K)\n");

var n = Convert.ToInt32(Console.ReadLine());

*bool[,] isPainted = new bool[N + 1, M + 1]; // матрица для закрашенных
клеток*

Console.WriteLine("Введите координаты закрашенных клеток (X Y)\n");

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

{

string[] coords = Console.ReadLine().Split();

int X = int.Parse(coords[0]);

int Y = int.Parse(coords[1]);

isPainted[X, Y] = true;

}

int maxArea = 0;

// перебираем все возможные прямоугольники

for (int x1 = 1; x1 <= N; x1++) // левых верх

{

for (int y1 = 1; y1 <= M; y1++)

{

*if (isPainted[x1, y1]) continue; // нашли закрашенную клетку, сразу
же переходим к следующему шагу*

int maxX = N;

int maxY = M;

for (int x2 = x1; x2 <= maxX; x2++) // правый низ

{

for (int y2 = y1; y2 <= maxY; y2++)

{

bool isValid = true;

for (int x = x1; x <= x2; x++)

{

for (int y = y1; y <= y2; y++)

{

*if (isPainted[x, y]) // встретили закрашенную клетку -
уменьшили границу*

```

        {
            isValid = false;
            maxY = y - 1;
            break;
        }
    }
    if (!isValid) break;
}
if (isValid)
{
    int area = (x2 - x1 + 1) * (y2 - y1 + 1);
    if (area > maxArea)
        maxArea = area;
}
else
{
    break;
}
}
}
}
}
Console.WriteLine($"Максимальная площадь свободного прямоугольника
равна {maxArea}");
}
}

```

Ввод и вывод данных осуществляется через консоль. Пользователь вводит размер области, количество закрашенных клеток, а также их координаты. По сути алгоритм основан на переборе всех возможных вариантов прямоугольников (используется несколько циклов for). Если закрашенная клетка попадает в наш прямоугольник – сразу переходим к проверке следующего прямоугольника. Если нет – происходит сравнение с предыдущей площадью прямоугольника (используется переменная maxArea). Полученный результат выводится на экран.

Задача из дискретной математики

Фирме, осуществляющей перевозку скоропортящегося товара, дано задание на доставку товара из Ставрополя в Будённовск, при этом существует несколько путей, по которым возможно доставить товар. Расстояние между городом Ставрополь и селом К. – 26 километров, между г. Ставрополем и селом П. – 19 километров, между г. Ставрополем и селом Р. – 86 километров. Между сёлами К. и Д. – 16 километров, между сёлами К. и Л. – 66 километров. Между селом П. и городом Н. составляет 4 километра, между сёлами П. и В. – 51 километр. Между сёлами Д. и В. – 21 километр. Между городом Н. и селом М. – 21 километр. Между сёлами М. и Л. – 24 километра, между сёлами М. и В. – 34 километра. Между сёлами Л. и А. – 13 километров, между сёлами Л. и Ж. – 43 километра. Между сёлами А. и Б. – 25 километров. Между сёлами Ж. и Р. – 31 километр, между сёлами Ж. и Б. – 44 километра. Между сёлами Б. и Р. – 22 километра. Между сёлами В. Ж. – 9 километров. Необходимо найти самый короткий путь из Ставрополя в Буденновск.

Формат выходных данных

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

Ход решения задачи из дискретной математики

Для начала решим задачу при помощи программы:

```
using System;
using System.Collections.Generic;
class App
{
    static void AddDistance(int[,] graph, string[] cities, string city1, string city2, int
distance)
    {
        int i = Array.IndexOf(cities, city1);
        int j = Array.IndexOf(cities, city2);
        graph[i, j] = distance;
        graph[j, i] = distance;
    }
    static int MinDistance(int[] distances, bool[] visited)
    {
        int min = int.MaxValue;
        int minIndex = -1;
        for (int v = 0; v < distances.Length; v++)
        {
            if (!visited[v] && distances[v] <= min)
            {
                min = distances[v];
                minIndex = v;
            }
        }
        return minIndex;
    }
    static void Main()
    {
        string[] cities = { "Cm", "K", "П", "P", "Д", "Л", "H", "B", "M", "A", "Ж",
"Б" };
        int cityCount = 12;
        int[,] graph = new int[cityCount, cityCount];
        for (int i = 0; i < cityCount; i++)
            for (int j = 0; j < cityCount; j++)
                graph[i, j] = -1; // -1 означает отсутствие пути
        AddDistance(graph, cities, "Cm", "K", 26);
        AddDistance(graph, cities, "Cm", "П", 19);
        AddDistance(graph, cities, "Cm", "P", 86);
        AddDistance(graph, cities, "K", "Д", 16);
        AddDistance(graph, cities, "K", "Л", 66);
        AddDistance(graph, cities, "П", "H", 4);
```

```

AddDistance(graph, cities, "П", "Б", 51);
AddDistance(graph, cities, "Д", "Б", 21);
AddDistance(graph, cities, "Н", "М", 21);
AddDistance(graph, cities, "М", "Л", 24);
AddDistance(graph, cities, "М", "Б", 34);
AddDistance(graph, cities, "Л", "А", 13);
AddDistance(graph, cities, "Л", "Ж", 43);
AddDistance(graph, cities, "А", "Б", 25);
AddDistance(graph, cities, "Ж", "Р", 31);
AddDistance(graph, cities, "Ж", "Б", 44);
AddDistance(graph, cities, "Р", "Б", 22);
AddDistance(graph, cities, "Б", "Ж", 9);
int start = Array.IndexOf(cities, "См");
int end = Array.IndexOf(cities, "Б");
int[] distances = new int[cityCount];
int[] previous = new int[cityCount];
bool[] visited = new bool[cityCount];
for (int i = 0; i < cityCount; i++) // заполняем матрицу расстояний
{
    distances[i] = int.MaxValue;
    previous[i] = -1;
}
distances[start] = 0;

for (int count = 0; count < cityCount - 1; count++) // по сути тот же
алгоритм Дейкстры
{
    int u = MinDistance(distances, visited);
    if (u == -1) break;

    visited[u] = true;

    for (int v = 0; v < cityCount; v++)
    {
        if (!visited[v] && graph[u, v] != -1 && distances[u] != int.MaxValue
&&
            distances[u] + graph[u, v] < distances[v])
        {
            distances[v] = distances[u] + graph[u, v];
            previous[v] = u;
        }
    }
}

List<string> path = new List<string>(); // восстанавливаем путь

```

```

    int current = end;
    while (current != -1)
    {
        path.Add(cities[current]);
        current = previous[current];
    }
    path.Reverse();

    Console.WriteLine("Самый короткий путь:");
    Console.WriteLine(string.Join(" - ", path));
    Console.WriteLine($"Общее расстояние равно {distances[end]} км");
}
}

```

Ввод и вывод данных осуществляется через консоль. В начале программа формирует матрицу расстояний между городами (весовую матрицу), где -1 будет означать отсутствие пути между городами. Далее применяется классический алгоритм Дейкстры. В конце мы восстанавливаем путь по массиву previous. На экран выводится сам путь и его длина.

Теперь решим задачу вручную:

	Ст	К	П	Р	Д	Л	Н	В	М	А	Ж	Б
Ст	0	26	19	86	-	-	-	-	-	-	-	-
К	36	0	-	-	16	66	-	-	-	-	-	-
П	19	-	0	-	-	-	4	51	-	-	-	-
Р	86	-	-	0	-	-	-	-	-	-	31	22
Д	-	16	-	-	0	-	-	21	-	-	-	-
Л	-	66	-	-	-	0	-	-	24	13	43	-
Н	-	-	4	-	-	-	0	-	21	-	-	-
В	-	-	51	-	21	-	-	0	34	-	9	-
М	-	-	-	-	-	24	21	34	0	-	-	-
А	-	-	-	-	-	13	-	-	-	0	-	25
Ж	-	-	-	31	-	43	-	9	-	-	0	44
Б	-	-	-	22	-	-	-	-	-	25	44	0

	Ст	К	П	Р	Д	Л	Н	В	М	А	Ж	Б	
1	-	26	19	86	-	-	-	-	-	-	-	-	П
2	-	26	-	86	-	-	23	70	-	-	-	-	Н
3	-	26	-	86	-	-	-	70	44	-	-	-	К
4	-	-	-	86	42	92	-	70	44	-	-	-	Д
5	-	-	-	86	-	92	-	63	44	-	-	-	М
6	-	-	-	86	-	68	-	63	-	-	-	-	В
7	-	-	-	86	-	68	-	-	-	-	72	-	Л
8	-	-	-	86	-	-	-	-	-	81	72	-	Ж
9	-	-	-	86	-	-	-	-	-	81	-	116	А
10	-	-	-	86	-	-	-	-	-	-	-	106	Р
11	-	-	-	-	-	-	-	-	-	-	-	106	Б
12	0	26	19	86	42	68	23	63	44	81	72	106	-
Путь:	Ст	→	П	→	Н	→	М	→	Л	→	А	→	Б

В ручном решении используется алгоритм Дейкстры. В предпоследней строчке таблицы видны все кратчайшие пути из Ставрополя в другие сёла. Нам нужен путь в Будёновск => наш ответ 106. В последней строчке выписан восстановленный путь. Ответы в программном и ручном решениях совпадают.

Заключение

В рамках данной работы на языке C# были решены четыре практические задачи, каждая из которых требовала применения различных алгоритмов и методов программирования.

Это позволило углубить понимание языка, а также развить навыки алгоритмизации и проектирования решений. Полученный опыт способствует более уверенной работе с C# и служит основой для решения более сложных задач в будущем. Постоянное совершенствование и практика — ключевые факторы профессионального роста в сфере IT.

Список литературы

- 1) Microsoft Learn документация по языку C# <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 01.02.2025).
- 2) Stack Overflow <https://stackoverflow.com/> (дата обращения: 28.02.2025).
- 3) Википедия C# https://ru.wikipedia.org/wiki/C_Sharp (дата обращения: 11.03.2025).
- 4) Metanit C# <https://metanit.com/sharp/> (дата обращения: 06.03.2025)