Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Звіт

з лабораторної роботи № 8 з дисципліни

«Евристичні алгоритми»

Виконали ІА-31 Клим'юк В.Л, Самелюк А.С, Дук М.Д, Сакун Д.С
(шифр, прізвище, ім'я, по батькові)

Перевірив Степанов А.С
(прізвище, ім'я, по батькові)

Київ 2024

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації евристичних алгоритмів і вирішення типових задач з їх допомогою.

2 ЗАВДАННЯ

Для **задачі про найкоротший шлях**, вибрати 15 міст в країні згідно варіанту (таблиця 2.1) і записати для них найкоротшу відстань по дорозі, у випадку прямого сполучення між ними і відстань по прямій в окремі таблиці. Для визначення відстані рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Для **задачі комівояжера**, вибрати 15 міст в країні згідно варіанту (таблиця 2.1) і записати для них найкоротшу відстань по дорозі, у випадку прямого сполучення між ними. Для визначення відстані рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Для **задачі розфарбовування графа**, вибрати 15 адміністративних одиниць (областей, районів) в країні згідно варіанту (таблиця 2.1) і записати для них суміжність один з одним. Для визначення суміжностей рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Для **задачі побудови мінімального вершинного покриття**, вибрати 15 міст в країні згідно варіанту (таблиця 2.1) і записати для них суміжність один з одним, у випадку прямого сполучення між ними. Для визначення відстані рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Записати алгоритми методів відповідно до варіанту.

Виконати програмну реалізацію алгоритму використовуючи задані методи та евристики, надати відповідь згідно опису нижче.

Для **задачі про найкоротший шлях.** Розробити програму, яка буде знаходити найкоротші маршрути між кожною парою міст. У якості методів знаходження маршруту вибрати Жадібний пошук і пошук A*. У якості евристики вибрати відстань по прямій.

Відповідь вивести у вигляді (Місто1-Місто2 Відстань: 234км Маршрут: Місто1 → Місто3 → Місто4 → Місто2). **Вивести кожну пару міст, для обох алгоритмів.**

Для **задачі комівояжера.** Розробити програму, яка буде знаходити маршрут мінімальної довжини, що включає усі міста. У якості методів знаходження маршруту вибрати заданий за варіантом жадібний метод.

Відповідь вивести у вигляді (Маршрут: Місто1 → Місто3 → Місто4 → Місто2 → Місто1, Довжина: 234км).

Для **задачі розфарбовування графа.** Розробити програму, яка буде знаходити хроматичне число графа та кольори вершин. У якості методів знаходження хроматичного числа обрати пошук з поверненнями з заданою відповідно до варіанту евристикою.

Відповідь вивести у вигляді (Розфарбування: Місто 1 – Колір 1, Місто 2 – Колір 2, Місто 3 – Колір 1 …., Хроматичне число: 4).

Для **задачі побудови мінімального вершинного покриття.** Розробити програму, яка буде знаходити мінімальне вершинне покриття. У якості методів знаходження покриття вибрати жадібний метод та метод апроксимації.

Відповідь вивести у вигляді (Покриття: Місто1, Місто3, Місто2, Розмірність: 3).

Зробити узагальнений висновок з лабораторної роботи, в якому оцінити якість алгоритмів.

+1 додатковий бал можна отримати за програмне формування таблиць відстаней, суміжностей, тощо (за допомогою API інтернет сервісів) або за графічну демонстрацію роботи алгоритмів (на графі за допомогою десктопного інтерфейсу), отримати можна лише +1 бал.

Таблиця 2.1 – Варіанти алгоритмів

№	**Задача**	**Алгоритми**	**Евристика**	**Країна/Карта**
1	Задача про найкоротший шлях, пошук	Жадібний пошук, A*	Відстань по прямій	Індонезія

	шляху та його довжини			
2	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	Австралія
3	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Австрія
4	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	Азербайджан
5	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	MRV	Іспанія
6	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Албанія

7	Задача розфарбовування графа, пошук	Пошук з поверненнями	Попередня перевірка значень	Алжир
	хроматичного числа			
8	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Італія
9	Задача про найкоротший шлях, пошук шляху та його довжини	Жадібний пошук, A*	Відстань по прямій	Ангола
10	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	ОАЕ
11	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Аргентина
12	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	Вірменія

13	Задача розфарбовування графа, пошук	Пошук з поверненнями	MRV	Мексика
	хроматичного числа			
14	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Афганістан
15	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Попередня перевірка значень	Молдова
16	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Бангладеш
17	Задача про найкоротший шлях, пошук шляху та його довжини	Жадібний пошук, A*	Відстань по прямій	Барбадос
18	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	Польща

19	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Білорусь

20	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	Португалія
21	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	MRV	Бельгія
22	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Сербія
23	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Попередня перевірка значень	Болгарія

24	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Словаччина
25	Задача про найкоротший шлях, пошук	Жадібний пошук, A*	Відстань по прямій	Норвегія

	шляху та його довжини			
26	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	Нідерланди
27	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Перу
28	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	Франція
29	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	MRV	Таїланд

30	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Туреччина
31	Задача розфарбовування графа, пошук	Пошук з поверненнями	Попередня перевірка значень	Хорватія
	хроматичного числа			
32	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Чехія
33	Задача про найкоротший шлях, пошук шляху та його довжини	Жадібний пошук, A*	Відстань по прямій	Швеція
34	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	Еквадор

35	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	ЮАР
36	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	ЮАР

3 ВИКОНАННЯ

3.1 Псевдокод алгоритмів

```
    //Greedy search
    var res1 = graph.GreedySearch(from, to);
    LinkedList<City> route = new LinkedList<City>();
    City? next = res1;
    while (next != null)
    {
        route.AddFirst(next);
        next = next.Prev;
    }
    Console.WriteLine("=== Greedy search ===");
    Console.WriteLine($"{from}-{to} Distance: {res1.Total}km Route:");
    Console.WriteLine(string.Join(" -> ", route));

    //A* search
    var res2 = graph.AStarAlgorithm(from, to);
    route = new LinkedList<City>();
    next = res2;
    while(next != null)
    {
        route.AddFirst(next);
        next = next.Prev;
    }
    Console.WriteLine("=== A* search ===");
    Console.WriteLine($"{from}-{to} Distance: {res2.Total}km Route:");
    Console.WriteLine(string.Join(" -> ", route));
}
```

3.2 Вхідні дані задачі

У таблиці 3.1 наведені відстані між містами по дорозі, якщо між ними є прямий шлях.

Таблиця 3.1 – Відстань між містами по дорозі

	Balikpapan	Sambodja	Menjangau	Gitan	Buat	Loa Kulu	Sidulang	Induandjat	Djambu	Muarasiram	Sepan	Djenebora	Panajam	Sotek	Lemper
Balikpapan	0	46	81	124	127	109	174	203	245	268	70	89	19	65	317
Sambodja	46	0	54	83	101	67	147	176	223	236	92	98	55	90	285
Menjangau	81	54	0	62	44	110	91	120	162	185	49	55	80	49	234
Gitan	124	83	62	0	35	33	64	94	137	160	110	118	148	110	208
Buat	127	101	44	35	0	42	52	82	125	148	92	100	123	92	197
Loa Kulu	109	67	110	33	42	0	69	101	143	179	151	159	133	151	228
Sidulang	174	147	91	64	52	69	0	44	86	110	139	144	170	137	158
Induandjat	203	176	120	94	82	101	44	0	43	67	166	173	199	165	116
Djambu	245	223	162	137	125	143	86	43	0	31	210	215	241	210	80
Muarasiram	268	236	185	160	148	179	110	67	31	0	233	241	264	233	58
Sepan	70	92	49	110	92	151	139	166	210	233	0	29	41	10	282
Djenebora	89	98	55	118	100	159	144	173	215	241	29	0	60	29	287
Panajam	19	55	80	148	123	133	170	199	241	264	41	60	0	37	313
Sotek	65	90	49	110	92	151	137	165	210	233	10	29	37	0	282
Lemper	317	285	234	208	197	228	158	116	80	58	282	287	313	282	0

В таблиці 3.2 наведені відстані між містами по прямій.

Таблиця 3.2 – Відстані між містами по прямій.

	Balikpapan	Sambodja	Menjangau	Gitan	Buat	Loa Kulu	Sidulang	Induandjat	Djambu	Muarasiram	Sepan	Djenebora	Panajam	Sotek	Lemper
Balikpapan	0	32	39	63	68	81	96	93	85	96	29	13	12	31	83
Sambodja	32	0	31	43	53	54	80	87	88	100	49	36	43	51	98
Menjangau	39	31	0	25	28	48	56	58	57	68	35	34	44	40	68
Gitan	63	43	25	0	11	24	35	51	60	74	59	59	69	65	82
Buat	68	53	28	11	0	32	27	40	50	67	59	62	73	66	74
Loa Kulu	81	54	48	24	32	0	41	67	81	95	83	80	89	88	106
Sidulang	96	80	56	35	27	41	0	31	52	67	83	89	99	89	83
Induandjat	93	87	58	51	40	67	31	0	24	35	73	84	94	78	55
Djambu	85	88	57	60	50	81	52	24	0	14	59	73	81	63	30
Muarasiram	96	100	68	74	67	95	67	35	14	0	65	81	87	67	21
Sepan	29	49	35	59	59	83	83	73	59	65	0	16	22	6	56
Djenebora	13	36	34	59	62	80	89	84	73	81	16	0	10	16	72
Panajam	12	43	44	69	73	89	99	94	81	87	22	10	0	20	76
Sotek	31	51	40	65	66	88	89	78	63	67	6	16	20	0	57
Lemper	83	98	68	82	74	106	83	55	30	21	56	72	76	57	0

3.3 Програмна реалізація

3.3.1 Вихідний код

```
using Classes;

namespace Lab8
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Graph graph = CreateGraph();

            //string from = "Balikpapan"; string to = "Sotek";
            //string from = "Loa Kulu"; string to = "Djenebora";
            //string from = "Lemper"; string to = "Balikpapan";
            //string from = "Buat"; string to = "Muarasiram";
            string from = "Balikpapan"; string to = "Loa Kulu";

            //Greedy search
            var res1 = graph.GreedySearch(from, to);
            LinkedList<City> route = new LinkedList<City>();
            City? next = res1;
            while (next != null)
            {
                route.AddFirst(next);
                next = next.Prev;
            }
            Console.WriteLine("=== Greedy search ===");
            Console.WriteLine($"{from}-{to} Distance: {res1.Total}km Route:");
            Console.WriteLine(string.Join(" -> ", route));

            //A* search
            var res2 = graph.AStarAlgorithm(from, to);
            route = new LinkedList<City>();
            next = res2;
            while(next != null)
            {
                route.AddFirst(next);
                next = next.Prev;
            }
            Console.WriteLine("=== A* search ===");
            Console.WriteLine($"{from}-{to} Distance: {res2.Total}km Route:");
            Console.WriteLine(string.Join(" -> ", route));
        }

        static Graph CreateGraph()
        {
            int inf = int.MaxValue;
            Dictionary<string, int> cities = new Dictionary<string, int>()
            {
                { "Balikpapan", 0 },
                { "Sambodja",   1 },
                { "Menjangau" , 2 },
                { "Gitan" ,     3 },
                { "Buat" ,      4 },
                { "Loa Kulu" ,  5 },
                { "Sidulang" ,  6 },
                { "Induandjat" ,7 },
                { "Djambu" ,    8 },
                { "Muarasiram" ,9 },
```

```
            { "Sepan" ,    10 },
            { "Djenebora" ,11 },
            { "Panajam" ,  12 },
            { "Sotek" ,    13 },
            { "Lemper" ,   14 }
        };
        Dictionary<int, string> indexes = new Dictionary<int, string>()
        {
            { 0, "Balikpapan" },
            { 1, "Sambodja"   },
            { 2, "Menjangau"  },
            { 3, "Gitan"      },
            { 4, "Buat"       },
            { 5, "Loa Kulu"   },
            { 6, "Sidulang"   },
            { 7, "Induandjat" },
            { 8, "Djambu"     },
            { 9, "Muarasiram" },
            {10 , "Sepan"     },
            {11 , "Djenebora" },
            {12 , "Panajam"   },
            {13 , "Sotek"     },
            {14 , "Lemper"    }
        };

        int[][] straightMatrix = new int[][]
        {
            new int[]{  0, 32, 39, 63, 68, 81, 96, 93, 85, 96, 29, 13, 12, 31, 83}, //0
            new int[]{ 32,  0, 31, 43, 53, 54, 80, 87, 88,100, 49, 36, 43, 51, 98}, //1
            new int[]{ 39, 31,  0, 25, 28, 48, 56, 58, 57, 68, 35, 34, 44, 40, 68}, //2
            new int[]{ 63, 43, 25,  0, 11, 24, 35, 51, 60, 74, 59, 59, 69, 65, 82}, //3
            new int[]{ 68, 53, 28, 11,  0, 32, 27, 40, 50, 67, 59, 62, 73, 66, 74}, //4
            new int[]{ 81, 54, 48, 24, 32,  0, 41, 67, 81, 95, 83, 80, 89, 88,106}, //5
            new int[]{ 96, 80, 56, 35, 27, 41,  0, 31, 52, 67, 83, 89, 99, 89, 83}, //6
            new int[]{ 93, 87, 58, 51, 40, 67, 31,  0, 24, 35, 73, 84, 94, 78, 55}, //7
            new int[]{ 85, 88, 57, 60, 50, 81, 52, 24,  0, 14, 59, 73, 81, 63, 30}, //8
            new int[]{ 96,100, 68, 74, 67, 95, 67, 35, 14,  0, 65, 81, 87, 67, 21}, //9
            new int[]{ 29, 49, 35, 59, 59, 83, 83, 73, 59, 65,  0, 16, 22,  6, 56}, //10
            new int[]{ 13, 36, 34, 59, 62, 80, 89, 84, 73, 81, 16,  0, 10, 16, 72}, //11
            new int[]{ 12, 43, 44, 69, 73, 89, 99, 94, 81, 87, 22, 10,  0, 20, 76}, //12
            new int[]{ 31, 51, 40, 65, 66, 88, 89, 78, 63, 67,  6, 16, 20,  0, 57}, //13
            new int[]{ 83, 98, 68, 82, 74,106, 83, 55, 30, 21, 56, 72, 76, 57,  0}  //14
        };

        int[][] roadMatrix = new int[][]
        {
            new int[]{  0, 46,inf,inf,127,inf,inf,inf,inf,inf, 70,inf,inf,inf,inf}, //0
            new int[]{ 46,  0,inf, 83,inf, 67,inf,inf,inf,inf,inf,inf,inf,inf,285}, //1
            new int[]{inf,inf,  0, 62,inf,inf,inf,inf,inf,185,inf,inf, 80,inf,inf}, //2
            new int[]{inf, 83, 62,  0,inf,inf, 64,inf,137,inf,inf,inf,148,inf,208}, //3
            new int[]{127,inf,inf,inf,  0,inf,inf,inf,inf,148, 92,inf,inf, 92,inf}, //4
            new int[]{inf, 67,inf,inf,inf,  0,inf,inf,inf,inf,inf,inf,133,inf,228}, //5
            new int[]{inf,inf,inf, 64,inf,inf,  0, 44,inf,inf,139,inf,inf,inf,inf}, //6
            new int[]{inf,inf,inf,inf,inf,inf, 44,  0, 43, 67,inf,173,inf,inf,inf}, //7
            new int[]{inf,inf,inf,137,inf,inf,inf, 43,  0,inf,inf,inf,inf,210,inf}, //8
            new int[]{inf,inf,185,inf,148,inf,inf, 67,inf,  0,233,241,inf,inf,inf}, //9
            new int[]{ 70,inf,inf,inf, 92,inf,139,inf,inf,233,  0, 29,inf, 10,inf}, //10
            new int[]{inf,inf,inf,inf,inf,inf,inf,173,inf,241, 29,  0,inf,inf,inf}, //11
            new int[]{inf,inf, 80,148,inf,133,inf,inf,inf,inf,inf,inf,  0, 37,inf}, //12
            new int[]{inf,inf,inf,inf, 92,inf,inf,inf,210,inf, 10,inf, 37,  0,282}, //13
            new int[]{inf,285,inf,208,inf,228,inf,inf,inf,inf,inf,inf,inf,282,  0}  //14
        };

        return new Graph(straightMatrix, roadMatrix, cities, indexes);
    }
  }
}
```

```
//int[][] straightMatrix = new int[][]
//{
//    new int[]{  0, 32, 39, 63, 68, 81, 96, 93, 85, 96, 29, 13, 12, 31, 83}, //0
//    new int[]{ 32,  0, 31, 43, 53, 54, 80, 87, 88,100, 49, 36, 43, 51, 98}, //1
//    new int[]{ 39, 31,  0, 25, 28, 48, 56, 58, 57, 68, 35, 34, 44, 40, 68}, //2
//    new int[]{ 63, 43, 25,  0, 11, 24, 35, 51, 60, 74, 59, 59, 69, 65, 82}, //3
//    new int[]{ 68, 53, 28, 11,  0, 32, 27, 40, 50, 67, 59, 62, 73, 66, 74}, //4
//    new int[]{ 81, 54, 48, 24, 32,  0, 41, 67, 81, 95, 83, 80, 89, 88,106}, //5
//    new int[]{ 96, 80, 56, 35, 27, 41,  0, 31, 52, 67, 83, 89, 99, 89, 83}, //6
//    new int[]{ 93, 87, 58, 51, 40, 67, 31,  0, 24, 35, 73, 84, 94, 78, 55}, //7
//    new int[]{ 85, 88, 57, 60, 50, 81, 52, 24,  0, 14, 59, 73, 81, 63, 30}, //8
//    new int[]{ 96,100, 68, 74, 67, 95, 67, 35, 14,  0, 65, 81, 87, 67, 21}, //9
//    new int[]{ 29, 49, 35, 59, 59, 83, 83, 73, 59, 65,  0, 16, 22,  6, 56}, //10
//    new int[]{ 13, 36, 34, 59, 62, 80, 89, 84, 73, 81, 16,  0, 10, 16, 72}, //11
//    new int[]{ 12, 43, 44, 69, 73, 89, 99, 94, 81, 87, 22, 10,  0, 20, 76}, //12
//    new int[]{ 31, 51, 40, 65, 66, 88, 89, 78, 63, 67,  6, 16, 20,  0, 57}, //13
//    new int[]{ 83, 98, 68, 82, 74,106, 83, 55, 30, 21, 56, 72, 76, 57,  0}  //14
//};

//int[][] roadMatrix = new int[][]
//{
//    new int[]{  0, 46, 81,124,127,109,174,203,245,268, 70, 89, 19, 65,317}, //0
//    new int[]{ 46,  0, 54, 83,101, 67,147,176,223,236, 92, 98, 55, 90,285}, //1
//    new int[]{ 81, 54,  0, 62, 44,110, 91,120,162,185, 49, 55, 80, 49,234}, //2
//    new int[]{124, 83, 62,  0, 35, 33, 64, 94,137,160,110,118,148,110,208}, //3
//    new int[]{127,101, 44, 35,  0, 42, 52, 82,125,148, 92,100,123, 92,197}, //4
//    new int[]{109, 67,110, 33, 42,  0, 69,101,143,179,151,159,133,151,228}, //5
//    new int[]{174,147, 91, 64, 52, 69,  0, 44, 86,110,139,144,170,137,158}, //6
//    new int[]{203,176,120, 94, 82,101, 44,  0, 43, 67,166,173,199,165,116}, //7
//    new int[]{245,223,162,137,125,143, 86, 43,  0, 31,210,215,241,210, 80}, //8
//    new int[]{268,236,185,160,148,179,110, 67, 31,  0,233,241,264,233, 58}, //9
//    new int[]{ 70, 92, 49,110, 92,151,139,166,210,233,  0, 29, 41, 10,282}, //10
//    new int[]{ 89, 98, 55,118,100,159,144,173,215,241, 29,  0, 60, 29,287}, //11
//    new int[]{ 19, 55, 80,148,123,133,170,199,241,264, 41, 60,  0, 37,313}, //12
//    new int[]{ 65, 90, 49,110, 92,151,137,165,210,233, 10, 29, 37,  0,282}, //13
//    new int[]{317,285,234,208,197,228,158,116, 80, 58,282,287,313,282,  0}  //14
            //};
```

```
namespace Classes
{
    public class City
    {
        public City Prev { get; set; }
        public string Name { get; set; }
        public int H { get; set; }
        public int F { get; set; }
        public int Total { get; set; }

        public override bool Equals(object? obj)
        {
            if (obj == null)
                return false;

            if (obj is not City)
                return false;

            var c = (City)obj;
            return Name == c.Name && Total == c.Total;
        }

        public override string ToString()
        {
            return Name;
        }
    }
```

```
}
```

```
namespace Classes
{
    public class Graph
    {
        private Dictionary<string, int> _Cities = new Dictionary<string, int>();
        private Dictionary<int, string> _Indexes = new Dictionary<int, string>();

        private int[][] _straightMatrix { get; set; }
        private int[][] _roadMatrix { get; set; }

        public Graph(int[][] straightMatrix, int[][] roadMatrix, Dictionary<string, int> cities, Dictionary<int, string> indexes)
        {
            _straightMatrix = straightMatrix;
            _roadMatrix = roadMatrix;
            _Cities = cities;
            _Indexes = indexes;
        }

        public City AStarAlgorithm(string from, string to)
        {
            bool found = false;
            City? res = null;

            LinkedList<City> next = new LinkedList<City>();
            LinkedList<City> closed = new LinkedList<City>();

            //Starting city
            City start = new City()
            {
                Name = from,
                F = 0,
                H = _straightMatrix[_Cities[from]][_Cities[to]]
            };
            start.Total = start.F + start.H;

            closed.AddFirst(start);

            //Fill next list with all cities connected to start city
            int[] roadMatrix = _roadMatrix[_Cities[start.Name]];
            for (int i = 0; i < roadMatrix.Length; i++)
            {
                if (i == _Cities[start.Name] || roadMatrix[i] == int.MaxValue)
                    continue;

                int[] straightMatrix = _straightMatrix[i];
                City nextCity = new City()
                {
                    Prev = start,
                    Name = _Indexes[i],
                    F = roadMatrix[i],
                    H = straightMatrix[_Cities[to]]
                };
                nextCity.Total = nextCity.F + nextCity.H;
                next.AddFirst(nextCity);
            }

            //While route not found
            while (!found)
            {
                //Minimal cost from next list
                int min = next.Min(x => x.Total);
                //Routes with minimal cost
                IEnumerable<City> list = next.Where(x => x.Total == min);
```

```
    //Lists for deletion and addition of cities
    LinkedList<City> addNext = new LinkedList<City>();
    LinkedList<City> removeNext = new LinkedList<City>();

    //Loop through all cities with minimal route cost
    foreach (City city in list)
    {
      //If destination is reached
      if (city.Name == to)
      {
        res = city;
        found = true;
        break;
      }

      //Delete city from next list if there are any better routes through this city
      var anotherRoutes = closed.Where(x => x.Name == city.Name && x.Total < city.Total);
      if (anotherRoutes.Count() != 0)
      {
        removeNext.AddFirst(city);
        continue;
      }

      //Examine road matrix for city
      roadMatrix = _roadMatrix[_Cities[city.Name]];
      for(int i = 0; i < roadMatrix.Length; i++)
      {
        //Skip if there is no connection or route loops to itself or previous city
        if (i == _Cities[city.Prev.Name] || i == _Cities[city.Name] || roadMatrix[i] == int.MaxValue)
          continue;

        //Next city
        int[] straightMatrix = _straightMatrix[i];
        City nextCity = new City()
        {
          Prev = city,
          Name = _Indexes[i],
          F = roadMatrix[i],
          H = straightMatrix[_Cities[to]]
        };
        nextCity.Total = city.Total - city.H + nextCity.F + nextCity.H;

        //Skip this city if there is already better option in closed list
        var checkClosed = closed.FirstOrDefault(x => x.Name == nextCity.Name && x.Total < nextCity.Total);
        if (checkClosed != null)
          continue;

        addNext.AddFirst(nextCity);
      }

      removeNext.AddFirst(city);
      closed.AddFirst(city);
    }

    //Remove cities from next list
    foreach(var i in removeNext)
    {
      next.Remove(i);
    }
    //Add cities to next list
    foreach(var i in addNext)
    {
      next.AddFirst(i);
    }
  }

  if (res == null)
```

```
        throw new Exception("Path was not found");

    return res;
}

public City GreedySearch(string from, string to)
{
    HashSet<string> memo = new HashSet<string>() { from };
    City city = new City()
    {
        Name = from,
        F = 0,
        H = 0,
        Total = 0
    };

    return _GreedySearch(city, memo, to);
}

private City _GreedySearch(City current, HashSet<string> memo, string to, string? ignore = null)
{
    int[] roadMatrix = _roadMatrix[_Cities[current.Name]];

    int min = -1;
    for(int i = 0; i < roadMatrix.Length; i++)
    {
        if (ignore != null && ignore == _Indexes[i])
            continue;

        if (roadMatrix[i] == int.MaxValue || memo.Contains(_Indexes[i]))
            continue;

        if (_Indexes[i] == to)
        {
            City c = new City()
            {
                Prev = current,
                Name = _Indexes[i],
                F = roadMatrix[i],
                H = 0
            };
            c.Total = current.Total + c.F;
            return c;
        }

        if (min == -1)
            min = i;
        else if (roadMatrix[min] < roadMatrix[i])
            min = i;
    }

    //If all routes where added inside memo try to return to previous city and change the route
    if (min == -1)
    {
        memo.Remove(current.Name);
        return _GreedySearch(current.Prev, memo, to, current.Name);
    }

    City city = new City()
    {
        Prev = current,
        Name = _Indexes[min],
        F = roadMatrix[min],
        H = 0
    };
    city.Total = current.Total + city.F;
    memo.Add(_Indexes[min]);
```

```
        return _GreedySearch(city, memo, to);
    }
  }
        }
```

3.3.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для різних алгоритмів пошук.

```
=== Greedy search ===
Balikpapan-Loa Kulu Distance: 1392km Route:
Balikpapan -> Buat -> Muarasiram -> Djenebora -> Induandjat -> Sidulang -> Sepan -> Sotek -> Lemper -> Loa Kulu
```

```
=== Greedy search ===
Djambu-Sidulang Distance: 924km Route:
Djambu -> Sotek -> Lemper -> Sambodja -> Gitan -> Sidulang
```

Рисунок 3.1 – Жадібний пошук

```
=== A* search ===
Balikpapan-Loa Kulu Distance: 113km Route:
Balikpapan -> Sambodja -> Loa Kulu
```

```
=== A* search ===
Djambu-Sidulang Distance: 87km Route:
Djambu -> Induandjat -> Sidulang
```

Рисунок 3.2 – Пошук A*

ВИСНОВОК

У ході виконання лабораторної роботи були розглянуті та реалізовані два евристичні алгоритми пошуку найкоротшого шляху між містами: жадібний пошук (Greedy Search) та алгоритм A*. Лабораторна робота продемонструвала ефективність та важливість евристичних алгоритмів у вирішенні задач пошуку шляхів. Жадібний пошук показав себе як швидкий та простий алгоритм, тоді як алгоритм A* виявився більш оптимальним, але складнішим у реалізації. Обидва підходи мають свої переваги та недоліки, що дозволяє вибирати їх залежно від конкретної задачі та вимог до її вирішення.