Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки Кафедра ІПІ

Звіт

з лабораторної роботи №8
з дисципліни «Алгоритми та структури даних 2. Структури даних»

«Евристичні алгоритми»

Виконав(ла) *ІП-22, Андреєва Уляна Андріївна*

(шифр, прізвище, ім'я, по батькові)

Перевірила *Халус Олена Андріївна*

(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

Практичне завдання №8

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.**

4

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.

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

шляху та його довжини

3.1 Псевдокод алгоритму

Greedy Algorithm

```
function greedySearch(distanceMatrix, cities, startCityIndex, endCityIndex):
    path := empty list
    visited := array of boolean values with length equal to the number of cities

    startCity := cities[startCityIndex]
    endCity := cities[endCityIndex]

    path.add(startCity)
    visited[startCityIndex] := true

    while path.get(path.size() - 1) is not equal to endCity:
        minDistance := positive infinity
        nextCityIndex := -1

        for i from 0 to length of cities:
            if visited[i] is false:
                currentDistance := distanceMatrix[getIndex(path.get(path.size() - 1),
cities)][i]
                if currentDistance < minDistance:
                    minDistance := currentDistance
                    nextCityIndex := i

        if nextCityIndex is not equal to -1:
            nextCity := cities[nextCityIndex]
            path.add(nextCity)
            visited[nextCityIndex] := true
        else:
            break
```

```
    return path
```

A Algorithm*

```
function aStarSearch(distanceMatrix, cities, startCityIndex, endCityIndex):
    startCity := cities[startCityIndex]
    endCity := cities[endCityIndex]

    openList := empty list
    closedList := empty list

    startNode := Node(startCity, null, 0, calculateHeuristic(startCity, endCity))
    openList.add(startNode)

    while openList is not empty:
        insertionSort(openList)
        currentNode := openList.remove(0)

        if currentNode.getCity() is equal to endCity:
            path := empty list
            while currentNode is not null:
                path.add(0, currentNode.getCity())
                currentNode := currentNode.getParent()
            return path

        for i from 0 to length of cities:
            successorCity := cities[i]
            if successorCity is not equal to currentNode.getCity() and not
isCityInList(successorCity, closedList):
                successorCost := currentNode.getCost() +
distanceMatrix[currentNode.getCity().getIndex()][i]
                successorHeuristic := calculateHeuristic(successorCity, endCity)
                successorNode := Node(successorCity, currentNode, successorCost,
successorHeuristic)

                if isCityInList(successorCity, openList):
                    existingNode := getNodeByCity(successorCity, openList)
                    if successorNode.getTotalCost() < existingNode.getTotalCost():
                        existingNode.setCost(successorCost)
                        existingNode.setParent(currentNode)
                else:
                    openList.add(successorNode)
```

closedList.add(currentNode)

return empty list

3.2 Вихідний код

City.java

```
public class City {
    City(String name, double longitude, double latitude, int index){
        this.name = name;
        this.longitude = longitude;
        this.latitude = latitude;
        this.index = index;
    }

    private String name;
    private double longitude;
    private double latitude;
    private int index;

    public int getIndex() {
        return index;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getLongitude() {
        return longitude;
    }

    public void setLongitude(double longitude) {
        this.longitude = longitude;
    }

    public double getLatitude() {
        return latitude;
    }

    public void setLatitude(double latitude) {
        this.latitude = latitude;
    }

    public String getFullInfo(){
        return String.format("City name: %s, coordinated(latitude,
longitude): %f,%f", this.name, this.latitude, this.longitude);
    }
}
```

Functions.java

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
```

```
public class Functions {
    public static City[] createCities(){
        City[] cities = new City[15];

        cities[0] = new City("Джакарта", 106.8456, -6.2088, 0);
        cities[1] = new City("Бандунг", 107.6191, -6.9175, 1);
        cities[2] = new City("Сурабая", 112.7521, -7.2575, 2);
        cities[3] = new City("Медан", 98.6722, 3.5952, 3);
        cities[4] = new City("Макасар", 119.4327, -5.1477, 4);
        cities[5] = new City("Палембанг", 102.2763, -3.7573, 5);
        cities[6] = new City("Йогякарта", 110.3695, -7.7956, 6);
        cities[7] = new City("Семаранг", 110.4710, -7.0249, 7);
        cities[8] = new City("Балікпапан", 116.8312, -1.2650, 8);
        cities[9] = new City("Падангг", 100.3638, -0.9245, 9);
        cities[10] = new City("Бандар-Лампунг", 105.2560, -5.4254, 10);
        cities[11] = new City("Денпасар", 115.2196, -8.6526, 11);
        cities[12] = new City("Манадо", 124.8418, 1.4937, 12);
        cities[13] = new City("Палембанг",  104.744371, -2.963397, 13);
        cities[14] = new City("Депок", 106.803116, -6.394666, 14);

        return cities;
    }

    public static double[][] calculateDistanceMatrix(City[] cities){
        double[][] distanceMatrix = new
double[cities.length][cities.length];

        for (int i = 0; i < cities.length; i++) {
            for (int j = 0; j < cities.length; j++) {
                if (i == j) {
                    // Встановлюємо відстань між містом і самим собою
рівною 0
                    distanceMatrix[i][j] = 0;
                } else {
                    // Обчислюємо відстань між містами за формулою
Гаверсінуса
                    double lon1 = Math.toRadians(cities[i].getLongitude());
                    double lat1 = Math.toRadians(cities[i].getLatitude());
                    double lon2 = Math.toRadians(cities[j].getLongitude());
                    double lat2 = Math.toRadians(cities[j].getLatitude());

                    double dlon = lon2 - lon1;
                    double dlat = lat2 - lat1;

                    double a = Math.pow(Math.sin(dlat / 2), 2) +
Math.cos(lat1) * Math.cos(lat2) * Math.pow(Math.sin(dlon / 2), 2);
                    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 -
a));

                    // Радіус Землі в кілометрах
                    double radius = 6371;

                    // Обчислення відстані
                    double distance = radius * c;

                    distanceMatrix[i][j] = distance;
                }
            }
        }
```

```
        return distanceMatrix;
    }

    public static List<City> greedySearch(double[][] distanceMatrix, City[]
cities, int startCityIndex, int endCityIndex) {
        List<City> path = new ArrayList<>();
        boolean[] visited = new boolean[cities.length];

        City startCity = cities[startCityIndex];
        City endCity = cities[endCityIndex];

        path.add(startCity);
        visited[startCityIndex] = true;

        while (path.get(path.size() - 1) != endCity) {
            double minDistance = Double.MAX_VALUE;
            int nextCityIndex = -1;

            for (int i = 0; i < cities.length; i++) {
                if (!visited[i]) {
                    double currentDistance =
distanceMatrix[getIndex(path.get(path.size() - 1), cities)][i];
                    if (currentDistance < minDistance) {
                        minDistance = currentDistance;
                        nextCityIndex = i;
                    }
                }
            }

            if (nextCityIndex != -1) {
                City nextCity = cities[nextCityIndex];
                path.add(nextCity);
                visited[nextCityIndex] = true;
            } else {
                break;
            }
        }

        return path;
    }

    public static double calculatePathDistance(List<City> path, double[][]
distanceMatrix, City[] cities) {
        double distance = 0;

        for (int i = 0; i < path.size() - 1; i++) {
            int cityIndex1 = getIndex(path.get(i), cities);
            int cityIndex2 = getIndex(path.get(i + 1), cities);
            distance += distanceMatrix[cityIndex1][cityIndex2];
        }

        return distance;
    }

    public static int getIndex(City city, City[] cities) {
        for (int i = 0; i < cities.length; i++) {
            if (cities[i].equals(city)) {
                return i;
            }
        }
        return -1;
```

```
    }

    public static String formatPath(List<City> path) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < path.size(); i++) {
            sb.append(path.get(i).getName());
            if (i < path.size() - 1) {
                sb.append(" -> ");
            }
        }
        return sb.toString();
    }


    public static List<City> aStarSearch(double[][] distanceMatrix, City[]
cities, int startCityIndex, int endCityIndex) {
        City startCity = cities[startCityIndex];
        City endCity = cities[endCityIndex];

        // Створення відкритого та закритого списків для алгоритму A*
        List<Node> openList = new ArrayList<>();
        List<Node> closedList = new ArrayList<>();

        // Додавання початкового вузла в відкритий список
        Node startNode = new Node(startCity, null, 0,
calculateHeuristic(startCity, endCity));
        openList.add(startNode);

        while (!openList.isEmpty()) {
            // Вибір вузла з найменшою загальною вартістю
            InsertionSort.insertionSort(openList);
            Node currentNode = openList.remove(0);

            // Перевірка, чи досягнуто кінцевого міста
            if (currentNode.getCity() == endCity) {
                // Побудова маршруту від кінцевого до початкового міста
                List<City> path = new ArrayList<>();
                while (currentNode != null) {
                    path.add(0, currentNode.getCity());
                    currentNode = currentNode.getParent();
                }
                return path;
            }

            // Додавання потомків до відкритого списку
            for (int i = 0; i < cities.length; i++) {
                City successorCity = cities[i];
                if (successorCity != currentNode.getCity() &&
!isCityInList(successorCity, closedList)) {
                    double successorCost = currentNode.getCost() +
distanceMatrix[currentNode.getCity().getIndex()][i];
                    double successorHeuristic =
calculateHeuristic(successorCity, endCity);
                    Node successorNode = new Node(successorCity,
currentNode, successorCost, successorHeuristic);

                    if (isCityInList(successorCity, openList)) {
                        // Оновлення вартості та батьківського вузла, якщо
знайдений кращий шлях до поточного міста
                        Node existingNode = getNodeByCity(successorCity,
openList);
```

```
                        if (successorNode.getTotalCost() <
existingNode.getTotalCost()) {
                            existingNode.setCost(successorCost);
                            existingNode.setParent(currentNode);
                        }
                    } else {
                        // Додавання нового вузла до відкритого списку
                        openList.add(successorNode);
                    }
                }
            }

            // Додавання поточного вузла до закритого списку
            closedList.add(currentNode);
        }

        // Якщо не вдається знайти шлях, повертаємо порожній список
        return new ArrayList<>();
    }

    public static double calculateHeuristic(City currentCity, City endCity)
{
        // Розрахунок евристики "відстань по прямій" між двома містами
        double dx = endCity.getLongitude() - currentCity.getLongitude();
        double dy = endCity.getLatitude() - currentCity.getLatitude();
        return Math.sqrt(dx * dx + dy * dy);
    }

    public static boolean isCityInList(City city, List<Node> nodeList) {
        for (Node node : nodeList) {
            if (node.getCity() == city) {
                return true;
            }
        }
        return false;
    }

    public static Node getNodeByCity(City city, List<Node> nodeList) {
        for (Node node : nodeList) {
            if (node.getCity() == city) {
                return node;
            }
        }
        return null;
    }
}
```

InsertionSort.java

```
import java.util.List;

public class InsertionSort {
    public static void insertionSort(List<Node> nodeList) {
        int n = nodeList.size();

        for (int i = 1; i < n; i++) {
            Node key = nodeList.get(i);
            int j = i - 1;

            while (j >= 0 && nodeList.get(j).getTotalCost() >
```

```
key.getTotalCost()) {
                nodeList.set(j + 1, nodeList.get(j));
                j--;
            }

            nodeList.set(j + 1, key);
        }
    }
}
```

Main.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class Main {
    public static void main(String[] args) throws IOException {

        City[] cities = Functions.createCities ();
        double[][] distanceMatrix = new
double[cities.length][cities.length];

        try {
            for (int i = 0; i < cities.length; i++) {
                for (int j = 0; j < cities.length; j++) {
                    if (i == j) {
                        distanceMatrix[i][j] = 0;
                    } else {
                        // Create a URL object with the endpoint URL
                        URL url = new
URL("https://maps.googleapis.com/maps/api/distancematrix/json?origins=" +
cities[i].getName ()+ "&destinations="+ cities[j].getName
()+"&units=metric&key=AIzaSyDFf9YGUYEmseBaUZNnMrAotHe2lWW29D4");
                        // Open a connection to the URL
                        HttpURLConnection connection = (HttpURLConnection)
url.openConnection ();

                        // Set the request method (GET, POST, etc.)
                        connection.setRequestMethod ( "GET" );

                        // Set the request headers, if needed
                        connection.setRequestProperty ( "Content-Type",
"application/json" );

                        // Get the response code
                        int responseCode = connection.getResponseCode ();
                        System.out.println ( "Response Code: " +
responseCode );

                        // Read the response content
                        BufferedReader reader = new BufferedReader ( new
InputStreamReader ( connection.getInputStream () ) );
```

```
                        String line;
                        StringBuilder response = new StringBuilder ();

                        while ((line = reader.readLine ()) != null) {
                            response.append ( line );
                        }
                        reader.close ();

                        // Print the response content
                        //System.out.println("Response Content: " +
response.toString ());

                        // Close the connection
                        connection.disconnect ();

                        try {
                            // Parse the JSON string
                            JSONObject json = new JSONObject (
response.toString () );

                            // Get the "rows" array
                            JSONArray rows = json.getJSONArray ( "rows" );

                            // Get the first element of the "rows" array
                            JSONObject row = rows.getJSONObject ( 0 );

                            // Get the "elements" array
                            JSONArray elements = row.getJSONArray (
"elements" );

                            // Get the first element of the "elements"
array
                            JSONObject element = elements.getJSONObject ( 0
);

                            // Get the "distance" object
                            JSONObject distance = element.getJSONObject (
"distance" );

                            // Get the "text" value from the "distance"
object
                            String distanceText = distance.getString (
"text" );

                            System.out.println ( cities[i].getName () + "
to " + cities[j].getName () + " Distance: " + distanceText + " Path: " +
cities[i].getName () + " to " + cities[j].getName () );
                            var splittedString = distanceText.split ( " "
);
                            System.out.println ( splittedString[0] );
                            float distanceFloat = Float.parseFloat (
splittedString[0].replace ( ',', '.' ) );

                            distanceMatrix[i][j] = distanceFloat;

                            // Збереження відповіді API у файл
                            saveResponseToFile(cities[i], cities[j],
distanceText);
                        } catch (JSONException e) {
                            e.printStackTrace ();
                        }
```

```
                    }
                }
            }

        } catch (IOException e) {
            e.printStackTrace ();
        }
    }

    private static void saveResponseToFile(City origin, City destination,
String distance) {
        String fileName = "API.distances.txt";
        try (BufferedWriter writer = new BufferedWriter ( new FileWriter (
fileName, true ) )) {
            writer.write ( "Origin: " + origin.getName () );
            writer.newLine ();
            writer.write ( "Destination: " + destination.getName () );
            writer.newLine ();
            writer.write ( "Distance: " + distance );
            writer.newLine ();
            writer.newLine(); // Add an extra line for separating entries
        } catch (IOException e) {
            e.printStackTrace ();
        }
    }
}
```

MainGreedyAstar.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

public class MainGreedyAstar {
    public static void main(String[] args) {
        City[] cities = Functions.createCities();
        double[][] distanceMatrix =
Functions.calculateDistanceMatrix(cities);

        // Знаходження найкоротших маршрутів між кожною парою міст з
використанням жадібного пошуку
        try (BufferedWriter greedyWriter = new BufferedWriter(new
FileWriter("greedy.txt"));
             BufferedWriter astarWriter = new BufferedWriter(new
FileWriter("astar.txt"))) {
            for (int i = 0; i < cities.length; i++) {
                for (int j = 0; j < cities.length; j++) {
                    if (i != j) {
                        List<City> shortestPathGreedy =
Functions.greedySearch(distanceMatrix, cities, i, j);
                        double shortestDistanceGreedy =
Functions.calculatePathDistance(shortestPathGreedy, distanceMatrix,
cities);

                        // Запис результатів жадібного пошуку у файл
                        greedyWriter.write(cities[i].getName() + "-" +
cities[j].getName() + " (Жадібний пошук) Відстань: " +
String.format("%.2f", shortestDistanceGreedy) + " км Маршрут: " +
Functions.formatPath(shortestPathGreedy));
```

```
                        greedyWriter.newLine();
                        System.out.println(cities[i].getName() + "-" +
cities[j].getName() + " (Жадібний пошук) Відстань: " +
String.format("%.2f", shortestDistanceGreedy) + " км Маршрут: " +
Functions.formatPath(shortestPathGreedy));
List<City> shortestPathAStar = Functions.aStarSearch(distanceMatrix,
cities, i, j);
                        double shortestDistanceAStar =
Functions.calculatePathDistance(shortestPathAStar, distanceMatrix, cities);

                        // Запис результатів пошуку A* у файл
                        astarWriter.write(cities[i].getName() + "-" +
cities[j].getName() + " (Пошук A*) Відстань: " + String.format("%.2f",
shortestDistanceAStar) + " км Маршрут: " +
Functions.formatPath(shortestPathAStar));
                        astarWriter.newLine();
                        System.out.println(cities[i].getName() + "-" +
cities[j].getName() + " (Пошук A*) Відстань: " + String.format("%.2f",
shortestDistanceAStar) + " км Маршрут: " +
Functions.formatPath(shortestPathAStar));

                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Node.java

```
public class Node {
    private City city;
    private Node parent;
    private double cost;
    private double heuristic;

    public Node(City city, Node parent, double cost, double heuristic) {
        this.city = city;
        this.parent = parent;
        this.cost = cost;
        this.heuristic = heuristic;
    }

    public City getCity() {
        return city;
    }

    public Node getParent() {
        return parent;
    }

    public void setParent(Node parent) {
        this.parent = parent;
    }

    public double getCost() {
        return cost;
    }
```

```
    public void setCost(double cost) {
        this.cost = cost;
    }

    public double getHeuristic() {
        return heuristic;
    }

    public void setHeuristic(double heuristic) {
        this.heuristic = heuristic;
    }

    public double getTotalCost() {
        return cost + heuristic;
    }
}
```

3.2.1 Програмна реалізація

A* Algorithm

	0-Джакарта	1-Бандунг	2-Сурабая	3-Медан	4-Макасар	5-Палембанг	6-Йогякарта	7-Семаранг	8-Балікпапан	9-Паданг	10-Бандар-Лампунг	11-Денпасар	12-Манадо	13-Палембанг	14-Депок
0-Джакарта	**0**	116.24 км	662.57 км	1418.44 км	1397.69 км	574.86 км	427.06 км	410.59 км	1236.55 км	928.63 км	196.24 км	962.40 км	2173.71 км	429.48 км	21.19 км
1-Бандунг	**116.24 км**	**0**	**567.66 км**	**1533.81 км**	**1321.00 км**	**687.95 км**	**318.64 км**	**315.00 км**	**1199.21 км**	**1044.64 км**	**309.47 км**	**859.24 км**	**2127.76 км**	**542.86 км**	**107.25 км**
2-Сурабая	**662.57 км**	**567.66 к**	**0**	**1974.22 км**	**774.82 км**	**1222.87 км**	**269.38 км**	**253.00 км**	**805.24 км**	**1543.31 км**	**853.07 км**	**312.89 км**	**1657.09 км**	**1007.05 км**	**663.77 км**
3-Медан	**1418.44 км**	**1533.81 км**	**1974.22 км**	**0**	**2502.55 км**	**910.38 км**	**1813.33 км**	**1763.32 км**	**2089.27 км**	**536.58 км**	**1241.29 км**	**2284.88 км**	**2916.22 км**	**993.59 км**	**1431.36 км**
4-Макасар	**1397.69 км**	**1321.00 км**	**774.82 км**	**2502.55 км**	**0**	**1908.13 км**	**1043.65 км**	**1012.57 км**	**519.41 км**	**2168.34 км**	**1569.94 км**	**606.73 км**	**952.03 км**	**1647.06 км**	**1404.03 км**
5-Палембанг	**574.86 км**	**687.95 км**	**1222.87 км**	**910.38 км**	**1908.13 км**	**0**	**1001.47 км**	**977.11 км**	**1640.32 км**	**379.95 км**	**378.78 км**	**1530.39 км**	**2574.87 км**	**287.83 км**	**580.81 км**

6-Йогякарта	427.06 км	318.64 км	269.38 км	1813.33 км	1043.65 км	1001.47 км	0	86.43 км	1019.70 км	1346.44 км	623.24 км	542.19 км	1908.67 км	822.35 км	423.23 км
7-Семаранг	410.59 км	315.00 км	253.00 км	1763.32 км	1012.57 км	977.11 км	86.43 км	0	952.54 км	1309.93 км	603.25 км	553.49 км	1854.72 км	778.58 км	411.07 км
8-Балікпапан	1236.55 км	1199.21 км	805.24 км	2089.27 км	519.41 км	1640.32 км	1019.70 км	952.54 к	0	1831.14 км	1365.38 км	840.61 км	942.00 км	1356.24 км	1249.93 км
9-Паданг	928.63 км	1044.64 км	1543.31	536.58 км	2168.34 км	379.95 км	1346.44 км	1309.93 км	1831.14 км	0	738.47 км	1855.76 км	2734.84 км	537.00 км	938.18 км
10-Бандар-Лампунг	196.24 км	309.47 км	853.07 км	1241.29 км	1569.94 км	378.78 км	623.24 км	603.25 км	1365.38 км	738.47 км	0	1156.46 км	2307.27 км	279.58 км	202.23 км
11-Денпасар	962.40 км	859.24 км	312.89 км	2284.88 км	606.73 км	1530.39 км	542.19 км	553.49 км	840.61 км	1855.76 км	1156.46 км	0	1552.47 км	1319.80 км	961.11 км
12-Манадо	2173.71 км	2127.76 км	1657.09 км	2916.22 км	952.03 км	2574.87 км	1908.67 км	1854.72 км	942.00 км	2734.84 км	2307.27 км	1552.47 км	0	2288.30 км	2186.06 км
13-Палембанг	429.48 км	542.86 км	1007.05 км	993.59 км	1647.06 км	287.83 км	822.35 км	778.58 км	1356.24 км	537.00 км	279.58 км	1319.80 км	2288.30 км	0	444.54 км
14-Депок	21.19 км	107.25 км	663.77 км	1431.36 км	1404.03 км	580.81 км	423.23 км	411.07 км	1249.93 км	938.18 км	202.23 км	961.11 км	2186.06 км	444.54 км	0

ВИСНОВОК

Алгоритм A* є потужним та ефективним алгоритмом пошуку шляху, який використовується для вирішення проблем шляхового планування. Після вивчення та розуміння принципу роботи алгоритму A*, я переконаний, що він є одним з найкращих алгоритмів для пошуку найкоротшого шляху у графі зі зваженими ребрами.

Основна ідея алгоритму полягає в тому, щоб знаходити шлях, який має найменшу вартість, враховуючи як відстань від стартової вершини до поточної, так і оцінку відстані до цільової вершини. Алгоритм A* поєднує в собі алгоритм Дейкстри та евристичну оцінку, що робить його ефективним і швидким.