Name – Pranav Salunkhe

TY CS-D

Batch-1

Roll no – 09

Assignment 3

Code –

import java.util.\*;

class City {

    String name;

    int x, y;

    Map<City, Integer> neighbours = new HashMap<>();

    public City(String name, int x, int y) {

        this.name = name;

        this.x = x;

        this.y = y;

    }

    public void addNeighbour(City neighbour, int distance) {

        if (this.equals(neighbour))

            return;

        neighbours.put(neighbour, distance);

        try {

            if (neighbour.getDistanceTo(this) != this.getDistanceTo(neighbour))

                throw new NullPointerException();

        } catch (NullPointerException e) {

            neighbour.addNeighbour(this, distance);

        }

    }

    public int getDistanceTo(City neighbour) {

        return neighbours.get(neighbour);

    }

}

public class AStarRomaniaMap {

    public static List<City> aStarSearch(City start, City goal) {

        Map<City, City> parentMap = new HashMap<>();

        Map<City, Integer> gScore = new HashMap<>();

        Map<City, Integer> fScore = new HashMap<>();

        PriorityQueue<City> openList = new PriorityQueue<>(Comparator.comparingInt(fScore::get));

        Set<City> closedList = new HashSet<>();

        gScore.put(start, 0);

        fScore.put(start, heuristic(start, goal));

        openList.add(start);

        while (!openList.isEmpty()) {

            // for (City c : openList) {

            // System.out.println(c.name);

            // }

            // System.out.println("\*");

            City current = openList.poll();

            if (current.equals(goal)) {

                System.out.println("got path");

                return reconstructPath(parentMap, current, gScore, fScore);

            }

            closedList.add(current);

            for (City neighbour : current.neighbours.keySet()) {

                if (closedList.contains(neighbour))

                    continue;

                int tentativeGScore = gScore.getOrDefault(current, Integer.MAX\_VALUE)

                        + current.getDistanceTo(neighbour);

                if (tentativeGScore < gScore.getOrDefault(neighbour, Integer.MAX\_VALUE)) {

                    parentMap.put(neighbour, current);

                    gScore.put(neighbour, tentativeGScore);

                    fScore.put(neighbour, gScore.get(neighbour) + heuristic(neighbour, goal));

                    if (!openList.contains(neighbour)) {

                        openList.add(neighbour);

                    }

                }

                // for (City a : gScore.keySet()) {

                // System.out.println(current.name + " " + neighbour.name + " " + a.name + "-g"

                // + gScore.get(a));

                // }

                // for (City a : fScore.keySet()) {

                // System.out.println(current.name + " " + neighbour.name + " " + a.name + "-f"

                // + fScore.get(a));

                // }

                // System.out.println();

            }

        }

        return Collections.emptyList();

    }

    public static List<City> reconstructPath(Map<City, City> parentMap, City current, Map<City, Integer> gScore,

            Map<City, Integer> fScore) {

        List<City> path = new ArrayList<>();

        int index = 0;

        while (current != null) {

            path.add(index++, current);

            current = parentMap.get(current);

        }

        Collections.reverse(path);

        return path;

    }

    public static int heuristic(City city, City goal) {

        return Math.abs(city.x - goal.x) + Math.abs(city.y - goal.y);

    }

    // public static void main(String[] args) {

    // City A = new City("A", 0, 0);

    // City B = new City("B", 0, 1);

    // City C = new City("C", 1, 1);

    // City D = new City("D", 1, 0);

    // A.addNeighbour(B, 2);

    // A.addNeighbour(D, 1);

    // B.addNeighbour(C, 5);

    // D.addNeighbour(C, 3);

    // List<City> li = aStarSearch(A, C);

    // for (City c : li) {

    // System.out.println(c.name);

    // }

    // }

    public static void main(String[] args) {

        City Arad = new City("Arad", 91, 492);

        City Bucharest = new City("Bucharest", 400, 327);

        City Craiova = new City("Craiova", 253, 288);

        City Drobeta = new City("Drobeta", 165, 299);

        City Eforie = new City("Eforie", 562, 293);

        City Fagaras = new City("Fagaras", 305, 449);

        City Giurgiu = new City("Giurgiu", 375, 270);

        City Hirsova = new City("Hirsova", 534, 350);

        City Iasi = new City("Iasi", 473, 506);

        City Lugoj = new City("Lugoj", 165, 379);

        City Mehadia = new City("Mehadia", 168, 339);

        City Neamt = new City("Neamt", 406, 537);

        City Oradea = new City("Oradea", 131, 571);

        City Pitesti = new City("Pitesti", 320, 368);

        City RimnicuVilcea = new City("Rimnicu Vilcea", 233, 410);

        City Sibiu = new City("Sibiu", 207, 457);

        City Timisoara = new City("Timisoara", 94, 410);

        City Urziceni = new City("Urziceni", 456, 350);

        City Vaslui = new City("Vaslui", 509, 444);

        City Zerind = new City("Zerind", 108, 531);

        // Define connections and distances

        Arad.addNeighbour(Zerind, 75);

        Arad.addNeighbour(Timisoara, 118);

        Arad.addNeighbour(Sibiu, 140);

        Bucharest.addNeighbour(Giurgiu, 90);

        Bucharest.addNeighbour(Urziceni, 85);

        Bucharest.addNeighbour(Fagaras, 211);

        Bucharest.addNeighbour(Pitesti, 101);

        Craiova.addNeighbour(Drobeta, 120);

        Craiova.addNeighbour(RimnicuVilcea, 146);

        Craiova.addNeighbour(Pitesti, 138);

        Drobeta.addNeighbour(Mehadia, 75);

        Eforie.addNeighbour(Hirsova, 86);

        Fagaras.addNeighbour(Sibiu, 99);

        Giurgiu.addNeighbour(Bucharest, 90);

        Hirsova.addNeighbour(Urziceni, 98);

        Iasi.addNeighbour(Vaslui, 92);

        Iasi.addNeighbour(Neamt, 87);

        Lugoj.addNeighbour(Timisoara, 111);

        Lugoj.addNeighbour(Mehadia, 70);

        Oradea.addNeighbour(Zerind, 71);

        Oradea.addNeighbour(Sibiu, 151);

        Pitesti.addNeighbour(RimnicuVilcea, 97);

        RimnicuVilcea.addNeighbour(Sibiu, 80);

        Timisoara.addNeighbour(Arad, 118);

        Timisoara.addNeighbour(Lugoj, 111);

        Urziceni.addNeighbour(Vaslui, 142);

        Urziceni.addNeighbour(Hirsova, 98);

        Vaslui.addNeighbour(Iasi, 92);

        Zerind.addNeighbour(Oradea, 71);

        Zerind.addNeighbour(Arad, 75);

        // Perform A\* search on the Romania map

        List<City> path = aStarSearch(Arad, Bucharest);

        // Print the path

        System.out.println("Shortest Path from Arad to Bucharest:");

        for (City city : path) {

            System.out.print(city.name + " -> ");

        }

    }

}

Output

