Name: Saurav Gujar

Class : TY CS D

Roll No: 14

Topic: **Implementation of Informed strategies.**

Code :

import java.util.\*;

class Node {

    int x, y; // Position on the grid

    int cost; // Cost from the start node to this node

    int heuristic; // Heuristic estimate of cost to goal

    Node parent; // Parent node

    public Node(int x, int y, int cost, int heuristic, Node parent) {

        this.x = x;

        this.y = y;

        this.cost = cost;

        this.heuristic = heuristic;

        this.parent = parent;

    }

    public int getTotalCost() {

        return cost + heuristic;

    }

}

public class AStar {

    public static List<Node> findPath(int[][] grid, int startX, int startY, int goalX, int goalY) {

        int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // Possible movement directions

        PriorityQueue<Node> openSet = new PriorityQueue<>(Comparator.comparingInt(Node::getTotalCost)); // Open set

        Map<Node, Integer> gScores = new HashMap<>(); // Cost from start to node

        Map<Node, Node> cameFrom = new HashMap<>(); // Parent nodes

        Node startNode = new Node(startX, startY, 0, heuristic(startX, startY, goalX, goalY), null);

        openSet.add(startNode);

        gScores.put(startNode, 0);

        while (!openSet.isEmpty()) {

            Node current = openSet.poll();

            if (current.x == goalX && current.y == goalY) {

                // Reconstruct the path from goal to start

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

                while (current != null) {

                    path.add(current);

                    current = cameFrom.get(current);

                }

                Collections.reverse(path);

                return path;

            }

            for (int[] dir : directions) {

                int newX = current.x + dir[0];

                int newY = current.y + dir[1];

                if (isValid(grid, newX, newY)) {

                    int tentativeGScore = gScores.get(current) + 1;

                    Node neighbor = new Node(newX, newY, tentativeGScore, heuristic(newX, newY, goalX, goalY), current);

                    if (!gScores.containsKey(neighbor) || tentativeGScore < gScores.get(neighbor)) {

                        gScores.put(neighbor, tentativeGScore);

                        cameFrom.put(neighbor, current);

                        openSet.add(neighbor);

                    }

                }

            }

        }

        // No path found

        return null;

    }

    // Heuristic function (Euclidean distance)

    private static int heuristic(int x1, int y1, int x2, int y2) {

        return Math.abs(x1 - x2) + Math.abs(y1 - y2);

    }

    // Check if a cell is valid (within bounds and not blocked)

    private static boolean isValid(int[][] grid, int x, int y) {

        return x >= 0 && x < grid.length && y >= 0 && y < grid[0].length && grid[x][y] == 0;

    }

    public static void main(String[] args) {

        int[][] grid = {

            {0, 0, 0, 0, 0},

            {0, 1, 1, 0, 0},

            {0, 0, 0, 0, 0},

            {0, 0, 1, 1, 0},

            {0, 0, 0, 0, 0}

        };

        List<Node> path = findPath(grid, 0, 0, 4, 4);

        if (path != null) {

            for (Node node : path) {

                System.out.println("(" + node.x + ", " + node.y + ")");

            }

        } else {

            System.out.println("No path found.");

        }

    }

}

Output :

(0, 0)

(0, 1)

(0, 2)

(0, 3)

(1, 3)

(1, 4)

(2, 4)

(3, 4)

(4, 4)