NAME – SUYASH GORAKH SHINDE

ROLL N0 – 28

PRN – 12111293

ASSIGNMENT 3

BATCH – D2

import java.util.\*;

public class AStarAlgorithm {

private Map<String, Integer> heuristicValues;

private Map<String, Integer> gValues;

private Map<String, String> parents;

private Set<String> openSet;

private Set<String> closedSet;

public AStarAlgorithm() {

// Initialize data structures.

heuristicValues = new HashMap<>();

gValues = new HashMap<>();

parents = new HashMap<>();

openSet = new HashSet<>();

closedSet = new HashSet<>();

// Initialize heuristic values for nodes.

initializeHeuristicValues();

}

private void initializeHeuristicValues() {

// Define heuristic values for nodes in the graph.

heuristicValues.put("S", 14);

heuristicValues.put("B", 12);

heuristicValues.put("C", 11);

heuristicValues.put("D", 6);

heuristicValues.put("E", 4);

heuristicValues.put("F", 11);

heuristicValues.put("G", 0);

}

public List<String> findPath(String startNode, String stopNode, Map<String, List<Pair<String, Integer>>> graph) {

openSet.add(startNode);

gValues.put(startNode, 0);

parents.put(startNode, startNode);

while (!openSet.isEmpty()) {

String n = null;

for (String v : openSet) {

if (n == null || gValues.get(v) + heuristic(v) < gValues.get(n) + heuristic(n)) {

n = v;

}

}

if (n.equals(stopNode) || !graph.containsKey(n)) {

break;

}

for (Pair<String, Integer> neighbor : graph.get(n)) {

String m = neighbor.getFirst();

int weight = neighbor.getSecond();

int tentativeG = gValues.get(n) + weight;

if (!openSet.contains(m) && !closedSet.contains(m)) {

openSet.add(m);

parents.put(m, n);

gValues.put(m, tentativeG);

} else if (tentativeG < gValues.get(m)) {

gValues.put(m, tentativeG);

parents.put(m, n);

if (closedSet.contains(m)) {

closedSet.remove(m);

openSet.add(m);

}

}

}

openSet.remove(n);

closedSet.add(n);

}

if (!parents.containsKey(stopNode)) {

System.*out*.println("Path doesn't exist");

return null;

}

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

String currentNode = stopNode;

while (!currentNode.equals(startNode)) {

path.add(currentNode);

currentNode = parents.get(currentNode);

}

path.add(startNode);

Collections.*reverse*(path);

System.*out*.println("Path found: " + path);

return path;

}

private int heuristic(String node) {

return heuristicValues.getOrDefault(node, Integer.*MAX\_VALUE*);

}

public static void main(String[] args) {

AStarAlgorithm astar = new AStarAlgorithm();

// Define your graph with connections between nodes.

Map<String, List<Pair<String, Integer>>> graph = new HashMap<>();

// Define connections and distances between nodes.

graph.put("S", Arrays.*asList*(new Pair<>("B", 4), new Pair<>("C", 3)));

graph.put("B", Arrays.*asList*(new Pair<>("F", 5), new Pair<>("E", 12)));

graph.put("C", Arrays.*asList*(new Pair<>("E", 10), new Pair<>("D", 7)));

graph.put("F", Collections.*singletonList*(new Pair<>("G", 16)));

graph.put("E", Collections.*singletonList*(new Pair<>("G", 5)));

graph.put("D", Collections.*singletonList*(new Pair<>("E", 2)));

String startNode = "S";

String stopNode = "G";

List<String> path = astar.findPath(startNode, stopNode, graph);

}

}

class Pair<F, S> {

private F first;

private S second;

public Pair(F first, S second) {

this.first = first;

this.second = second;

}

public F getFirst() {

return first;

}

public S getSecond() {

return second;

}