Assignment No-3

Div-B

Seat No: B150694205

Name: Singh Ankitkumar

(i) FxComparator.java

package bfsheuristics;

import java.util.Comparator;

public class FxComparator implements Comparator<HeadNode> // Comparator for priority queue based on fx value

{

@Override

public int compare(HeadNode o1, HeadNode o2) {

if(o1.getFx()> o2.getFx())

return 1;

else if(o1.getFx() < o2.getFx())

return -1;

return 0;

}

}

(ii) Graph.java

package bfsheuristics;

import java.util.ArrayList;

import java.util.Scanner;

import javax.swing.JOptionPane;

public class Graph { // Class for graph

ArrayList<HeadNode> headNodesList;

int n;

public Graph(int size) // Initialize size and head node list

{

this.n = size;

headNodesList = new ArrayList<>();

}

public void initGraph() // Initialize graph nodes and edges

{

Scanner sc = new Scanner(System.in);

for(int i=0;i<n;i++) // Accept node names and their heuristic values

{

HeadNode hn = new HeadNode();

hn.setName(JOptionPane.showInputDialog("Enter the name of node " +(i+1)+" : "));

hn.setHx(Integer.parseInt(JOptionPane.showInputDialog("Enter the heuristic value of node " +(i+1)+" : ")));

headNodesList.add(hn);

}

for(int i=0;i<n;i++)

{

HeadNode tempHeadNode = headNodesList.get(i);

while(true) // Accept adjacent nodes and their distances

{

String name = tempHeadNode.getName();

String ans = JOptionPane.showInputDialog("\nDo you want to add any adjacent node to node "+ name + "? (y/n) : ");

if(ans.equals("n") || ans.equals("N"))

break;

// sc.skip("\n");

String tempName=JOptionPane.showInputDialog("Enter the name of adjacent node of "+ name + " : ");

//sc.skip("\n");

tempHeadNode.setNodeInfo(tempName);

headNodesList.set(i, tempHeadNode);

}

}

}

public void displayGraph() // Display graph adjacency list

{

for(int i=0;i<n;i++)

{

HeadNode tempHeadNode = headNodesList.get(i);

System.out.print("\n"+ tempHeadNode.getName() + " (hx = "+tempHeadNode.getHx()+") : ");

tempHeadNode.displayNodeList();

}

System.out.println("");

}

public int getIndex(String name) // Get index for given name

{

for(int i=0;i<n;i++)

{

HeadNode tempHeadNode = headNodesList.get(i);

if(tempHeadNode.getName().equals(name))

return i;

}

return -1;

}

public ArrayList getNeighbours(String node) // Get neighbour nodes list

{

int headIndex=getIndex(node);

return headNodesList.get(headIndex).getNodeList();

}

public HeadNode getHeadNode (String name){ // Get Head node by name

return headNodesList.get(getIndex(name));

}

(iii) HeadNode.java

package bfsheuristics;

import java.util.ArrayList;

import java.util.Iterator;

public class HeadNode // Adjacency list head node

{

private String name; // node name

private int hx; // heuristic value hx

private int fx; // fx = hx value

private ArrayList<Node> adjnodes = new ArrayList<>(); // Adjacent nodes list

public HeadNode() // Initialize hx and fx to infinity

{

hx=999;

fx = hx;

}

public int getHx() {

return hx;

}

public void setHx(int hx) { // Set hx and update fx accordingly

this.hx = hx;

setFx(this.hx);

}

public int getFx() {

return fx;

}

public void setFx(int fx) {

this.fx = fx;

}

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setNodeInfo(String name) // Set adjacent node name and distance

{

Node n = new Node(name);

adjnodes.add(n); // Add node to list

}

public ArrayList getNodeList()

{

return adjnodes;

}

public void displayNodeList() // Display adjacent nodes list (name,distance)

{

Iterator i = adjnodes.iterator();

if(i.hasNext())

{

Node temp= (Node)i.next();

System.out.print("("+temp.getName()+")");

}

while(i.hasNext())

{

Node temp= (Node)i.next();

System.out.print(", ("+temp.getName()+")");

}

}

}

(iv) Node.java

package bfsheuristics;

public class Node // Adjacent node name

{

String name;

public Node(String name)

{

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

(v)BFSHeuristics.java

package bfsheuristics;

import java.util.ArrayList;

import java.util.PriorityQueue;

import javax.swing.JOptionPane;

public class BFSHeuristics {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

// TODO code application logic here

int n;

// Scanner sc = new Scanner(System.in);

// n = sc.nextInt();

//

n=Integer.parseInt(JOptionPane.showInputDialog("Enter No of nodes")); // Enter no. of rows

PriorityQueue<HeadNode> open = new PriorityQueue<>(new FxComparator()); // Initilize priority queue openlist

ArrayList<HeadNode> closed = new ArrayList<>(n); // Initialize closed list

ArrayList<String> parent = new ArrayList<>(n); // Store parent node

for(int i=0;i<n;i++)

{

parent.add("NIL"); // Set parent of all nodes NIL

}

Graph graph = new Graph(n); // Create graph instance

graph.initGraph(); // Initialize graph

graph.displayGraph(); // Display graph as adjacency list

String start, goal; // Accept start and goal nodes

start = JOptionPane.showInputDialog("Enter the name of start node : ");

goal = JOptionPane.showInputDialog("Enter the name of goal node : ");

open.add(graph.getHeadNode(start)); // Add start node to open list

parent.set(graph.getIndex(start), "NIL"); // Set parent of start NIL

displayQueue(open);

displayClosed(closed);

while(!open.isEmpty()) // Process until open list is not empty

{

HeadNode temp = open.poll(); // Remove node with minimum fx from open list

closed.add(temp); // Add it to closed list

displayQueue(open);

displayClosed(closed);

if(temp.getName().equals(goal)) // Check if goal node is found

{

System.out.println("\nGoal node '"+temp.getName() + "' found");

break;

}

else

{

ArrayList<Node> neighbours = temp.getNodeList(); // Get the neighbours of the retrieved node

for(Node n1:neighbours) // For all adjacent nodes

{

if(inClosed(n1.getName(), closed)) // If node in closed list, process next node

continue;

if(!inOpen(n1.getName(), open)) // Check if not in open list

{

open.add(graph.getHeadNode(n1.getName())); // Add it toopen list

parent.set(graph.getIndex(n1.getName()), temp.getName()); // Set parent of neighbour node

}

}

displayQueue(open);

}

}

tracePath(parent, graph, goal);

}

private static void displayQueue(PriorityQueue<HeadNode> open) // Fuction to display queue open list

{

System.out.print("\nOpen List : ");

if(open.isEmpty())

{

System.out.println("Empty");

return;

}

for(HeadNode n: open)

{

System.out.print(n.getName()+"\t");

}

System.out.println("");

}

private static void displayClosed(ArrayList<HeadNode> closed) // Fuction to display closed list

{

System.out.print("\nClosed List : ");

if(closed.isEmpty())

{

System.out.println("Empty");

return;

}

for(HeadNode n: closed)

{

System.out.print(n.getName()+"\t");

}

System.out.println("");

}

private static boolean inClosed(String name, ArrayList<HeadNode> closed) // Check if node in closed list

{

for(HeadNode n: closed)

{

if(n.getName().equals(name))

return true;

}

return false;

}

private static boolean inOpen(String name, PriorityQueue<HeadNode> open) // Check if node in closed list

{

for(HeadNode n: open)

{

if(n.getName().equals(name))

return true;

}

return false;

}

private static void tracePath(ArrayList<String> parent, Graph graph, String goal) // Function to trace the path

{

System.out.println("\n\nPath : ");

String path = goal;

String temp = goal;

while(!parent.get(graph.getIndex(temp)).equals("NIL")) // Continue path till parent is not NIL

{

temp = parent.get(graph.getIndex(temp));

path = temp + ", " + path;

}

System.out.println(path);

}

}

/\*

OUTPUT:

10 (hx = 7) : (3)

3 (hx = 6) :

Open List : 10

Closed List : Empty

Open List : Empty

Closed List : 10

Open List : 3

Open List : Empty

Closed List : 10 3

Goal node '3' found

Path :

10, 3

\*/