**ASSIGNMENT 7**

**Aim:** Use heuristic search technique to implement best first search and A\* algorithm.

**Software and Hardware Requirements:** Netbeans IDE.

### Theory:

**Heuristic Search:**

* + Heuristic exploits additional knowledge about the problem that helps direct search to more promising path.
  + A heuristic function h(n) provides an estimate of cost of the path from given node to closest goal. Example:-Straight line distance from current location to goal location in road navigation problem.
  + A good heuristic can:
    1. Find solution for an average problem efficiently.
    2. Find reasonably good but not optimal solution efficiently. 3.

### Best first search(BFS):

At each step, BFS sorts the queue according to heuristic function.

Function BEST-FIRST-SEARCH(problem,Eval-fn) returns a solution sequence. Inputs: problem, a problem

Eval-fn,an evaluation function

Queuing-fn← a function that orders nodes by Eval-fn return GENERAL- SEARCH (problem,Queuing-fn)

### Let us understand BFS Heuristic Search through pseudocode.

1. Define list OPEN with single node „s‟- the start node.
2. If list is empty, return failure.
3. Remove node „n‟ (node with best score). Move it to closed.
4. Expand node n.
5. If any successor to n is goal node,return success and trace path from goal node to „s‟ to return the solution.
6. For each successor node:
   * Apply evaluation function f.
   * If the node isn‟t in either list, add it to the list OPEN.
7. Loop to step 2.‟
   * **A\* Algorithm:**
     + A\* combines features of uniform cost search (complete, optimal, inefficient) with best-first search.
     + Sort queue by estimated total cost of the completion of a path. f(n)=g(n)+h(n)
     + If the heuristic function always underestimates the distance to the goal, it

is said to be “admissible”.

* + - If „h‟ os admissible then f(n) never overestimates the actual cost of best solution through n.

**Diagram:**

**Example:** Find optimal path to E through S using A\*.

### Heuristic values:

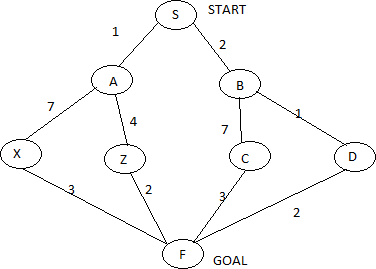
S→ 10 D→15

A→5 Z→5

B→6 Y→8

C→4

Checks the least cost path. f(n)=g(n)+h(n)





The least cost path here is S – A – Z – E with value „7‟.

* Properties:
  + A\* is complete as long as : Branching factor is always finite.

Every operation adds cost at least S>0.

* + Time and Space complexity is 0(bm) in the worst case.
  + However, with a good heuristic can find optimal solutions for many problems in reasonable time.

1. **Best First Search:**

**Code:**

Node.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package bfs;

public class Node // Class for adjacent nodes to headnode

{

String name; // node name

int distance; // distance between this node and headnode

// constructor and getters, setters for data members

public Node(String name, int dist)

{

this.name = name;

this.distance = dist;

}

public int getDistance() {

return distance;

}

public String getName() {

return name;

}

public void setDistance(int distance) {

this.distance = distance;

}

public void setName(String name) {

this.name = name;

}

}

HeadNode.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package bfs;

import java.util.ArrayList;

import java.util.Iterator;

public class HeadNode // Head node in adjacency list of graph

{

private String name; // node name

private ArrayList<Node> adjnodes = new ArrayList<>(); // List of adjacent nodes

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setNodeInfo(String name,int distance) // Add adjacent node

{

Node n = new Node(name,distance);

adjnodes.add(n);

}

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()+","+temp.getDistance()+")");

}

while(i.hasNext())

{

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

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

}

}

}

Graph.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package bfs;

import java.util.ArrayList;

import java.util.Scanner;

import javax.swing.JOptionPane;

public class Graph {

ArrayList<HeadNode> headNodesList; //arraylist to hold headNodes

int n;

public Graph(int size) //constructor to initialize the Graph

{

this.n = size;

headNodesList = new ArrayList<>();

}

public void initGraph() //method to accept graph nodes

{

Scanner sc = new Scanner(System.in);

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

{

//System.out.println("Enter the name of node" +(i+1)+" : ");

HeadNode hn = new HeadNode();

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

headNodesList.add(hn); //add the nodes to headNodeList

}

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

{

HeadNode tempHeadNode = headNodesList.get(i);

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

{

String name = tempHeadNode.getName();

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

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");

int tempDistance=Integer.parseInt(JOptionPane.showInputDialog("Enter distance between nodes " + name + " and " + tempName+ " :"));

tempHeadNode.setNodeInfo(tempName,tempDistance);

headNodesList.set(i, tempHeadNode);

}

}

}

public void displayGraph() //method to display graph in form of adjacency list

{

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

{

HeadNode tempHeadNode = headNodesList.get(i);

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

tempHeadNode.displayNodeList();

}

}

public int getIndex(String name) //method to get index by passing name of node

{

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

{

HeadNode tempHeadNode = headNodesList.get(i);

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

return i;

}

return -1; //if node not found return -1

}

public ArrayList getNeighbours(String node) //method to return neighbours of selected node

{

int headIndex=getIndex(node);

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

}

}

DistanceComparator.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package bfs;

import java.util.Comparator;

public class DistanceComparator implements Comparator<Node> // Comparator for priority queue based on Node distance

{

@Override

public int compare(Node o1, Node o2) {

if(o1.getDistance() > o2.getDistance())

return 1;

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

return -1;

return 0;

}

}

BFS.java

package bfs;

import java.util.ArrayList;

import java.util.PriorityQueue;

import java.util.Scanner;

import javax.swing.JOptionPane;

public class BFS // Class for BFS Algorithm

{

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args)

{

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<Node> pq = new PriorityQueue<>(new DistanceComparator()); // Initilize priority queue

ArrayList<Boolean> visited = new ArrayList<>(n);

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

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

{

visited.add(false); // Set visited list for all nodes false

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 : ");

pq.add(new Node(start,0)); // Add start node to priority queue with distance 0

visited.set(graph.getIndex(start), true); // Set visited true

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

System.out.println("\n\nPriority queue contents : \n");

displayQueue(pq);

while(!pq.isEmpty()) // Process untill queue is not empty

{

Node temp = pq.poll(); // Remove node with minimum distance

displayQueue(pq);

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

{

//JOptionPane.showMessageDialog(,"\nGoal node found");

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

break;

}

else

{

ArrayList<Node> neighbours = graph.getNeighbours(temp.getName()); // Get the neighbours of the retrieved node that are not visited

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

{

if(!visited.get(graph.getIndex(n1.getName())))

{

visited.set(graph.getIndex(n1.getName()), Boolean.TRUE); // Mark visited if not marked

pq.add(n1); // Add them to queue

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

}

}

displayQueue(pq); // Display the Queue

}

}

tracePath(parent,graph,goal);

}

private static void displayQueue(PriorityQueue<Node> pq) // Fuction to display queue

{

for(Node n:pq)

{

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

}

System.out.println("");

}

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 :

A : (B,3), (C,1)

B : (D,3), (E,2)

C :

D :

E :

Priority queue contents :

A

C B

B

B

E D

D

D

Goal node 'D' found

Path :

A, B, D

\*/

1. **A \*:**

Node.java :

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package astargraph;

public class Node // Adjacent node name and distance

{

String name;

int distance;

public Node(String name, int dist)

{

this.name = name;

this.distance = dist;

}

public int getDistance() {

return distance;

}

public String getName() {

return name;

}

public void setDistance(int distance) {

this.distance = distance;

}

public void setName(String name) {

this.name = name;

}

}

HeadNode.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package astargraph;

import java.util.ArrayList;

import java.util.Iterator;

public class HeadNode // Adjacency list head node

{

private String name; // node name

private int gx; // gx value

private int hx; // heuristic value hx

private int fx; // fx = gx+hx value

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

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

{

gx=hx=999;

fx = gx+hx;

}

public int getGx() {

return gx;

}

public void setGx(int gx) { // Set gx and update fx accordigly

this.gx = gx;

setFx(this.gx+hx);

System.out.println("\nFx of node "+this.name+" = "+this.fx);

}

public int getHx() {

return hx;

}

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

this.hx = hx;

setFx(this.hx+gx);

}

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,int distance) // Set adjacent node name and distance

{

Node n = new Node(name,distance);

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()+","+temp.getDistance()+")");

}

while(i.hasNext())

{

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

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

}

}

}

Graph.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package astargraph;

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");

int tempDistance=Integer.parseInt(JOptionPane.showInputDialog("Enter distance between nodes " + name + " and " + tempName+ " :"));

tempHeadNode.setNodeInfo(tempName,tempDistance);

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 void setGx(String name,int gx) // Set gx for a node and update adjacency list

{

int index = getIndex(name);

HeadNode node = headNodesList.get(index);

node.setGx(gx);

headNodesList.set(index, node);

}

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

return headNodesList.get(getIndex(name));

}

public void setFx(Node neighbour, HeadNode curr) // Set fx for neighbour via current node

{

int tempGx = curr.getGx() + neighbour.getDistance(); // Get distance from source to neighbour via current node

HeadNode adj = getHeadNode(neighbour.getName()); // Get adjacent head node

if(tempGx >= adj.getGx()) // Check if calculated distance is less than previous distance

return;

adj.setGx(tempGx); // Set gx as calculated distance

headNodesList.set(getIndex(adj.getName()), adj); // Update headnode list

}

}

FxComparator.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package astargraph;

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;

}

}

AStarGraph.java

package astargraph;

import java.util.ArrayList;

import java.util.PriorityQueue;

import javax.swing.JOptionPane;

public class AStarGraph {

/\*\*

\* @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 : ");

graph.setGx(start, 0); // Set gx=0 for start 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

{

graph.setFx(n1,temp); // Set fx for neighbour node via current

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:

A (hx = 6) : (B,1), (C,3)

B (hx = 4) : (D,2)

C (hx = 3) : (D,5)

D (hx = 1) :

Fx of node A = 6

Open List : A

Closed List : Empty

Open List : Empty

Closed List : A

Fx of node B = 5

Fx of node C = 6

Open List : B C

Open List : C

Closed List : A B

Fx of node D = 4

Open List : D C

Open List : C

Closed List : A B D

Path :

A, B, D

\*/

**CONCLUSION:**

Best first search and A\* algorithm has been implemented using heuristic search techniques.