INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prac. No. | Topic | Page No. | Date | Sign |
| 1. | Write a program to create a robot to move forward, turn, left, right without gear. |  |  |  |
| 2. | Write a program in java to create a robot with gear and move it forward, left, right. |  |  |  |
| 3. | Write a program to create a robot with two motor and move it forward, left, right. |  |  |  |
| 4. | Write a program to create a robot with light sensor to follow a line. |  |  |  |
| 5. | Write a program to create a robot that does a circle using two motor. |  |  |  |
| 6. | Write a program to create a path following robot. |  |  |  |
| 7. | Implement breadth first search for specified problem. |  |  |  |
| 8. | Implement depth first search for specified problem. |  |  |  |
| 9. | Implement “A” search for specified problem. |  |  |  |

**PRACTICAL NO :-1**

**TITLE**:- Write a program to create a robot to move forward, turn left and right without GEARS

**SOURCE CODE**:-

import ch.aplu.robotsim.\*;

class Movewithoutgears

{

Movewithoutgears ()

{

TurtleRobot robot = new TurtleRobot();

robot.forward(100);

robot.left(90);

robot.forward(50);

robot.right(90);

robot.forward(50);

robot.exit();

}

public static void main(String[] args)

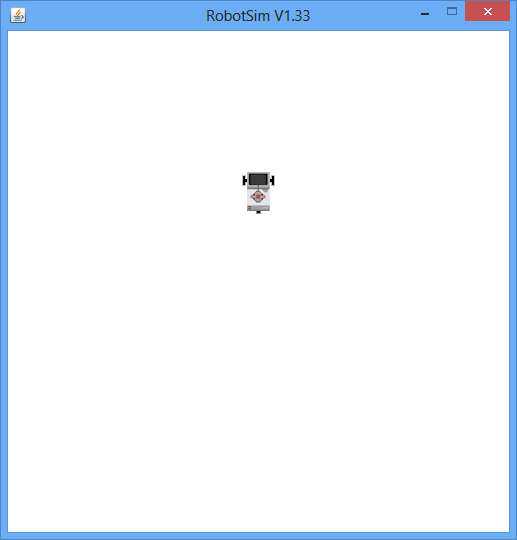
{

new Movewithoutgears ();

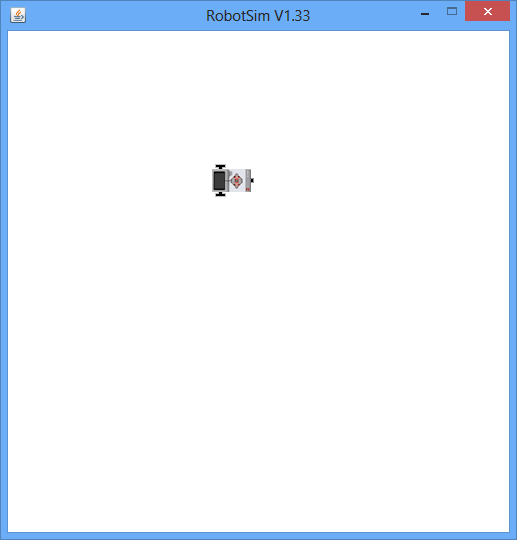
}

}

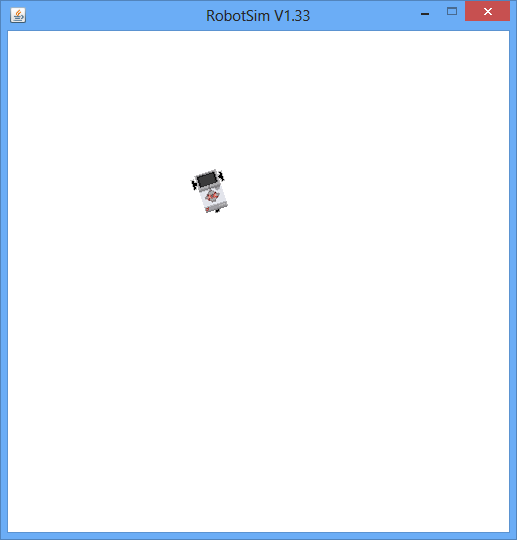
**Output: Initial State**



Turn left:-



Move forward:-



**PRACTICAL NO-2**

TITLE: Write a program in java to create a robot with gear and move it forward, left, right

Program code:

import ch.aplu.robotsim.\*;

class Movewithgear

{

Movewithgear()

{

NxtRobot robot = new NxtRobot();

Gear gear = new Gear();

robot.addPart(gear);

gear.forward(2000);

gear.setSpeed(30);

gear.left(480);

gear.forward(2000);

gear.right(480);

gear.forward();

// Tools.delay(2000);

robot.exit();

}

public static void main(String[] args)

{

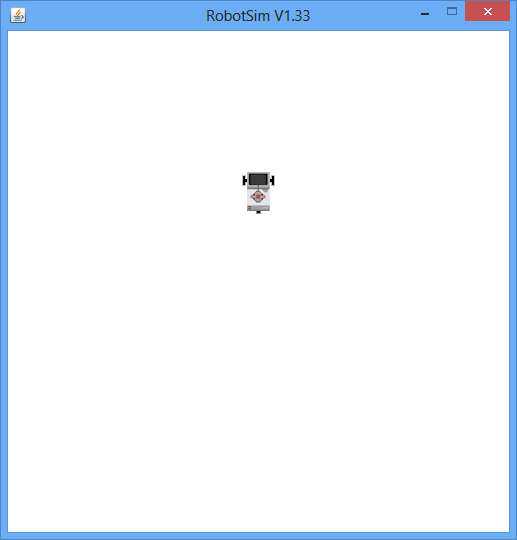
New Movewithgear();

}

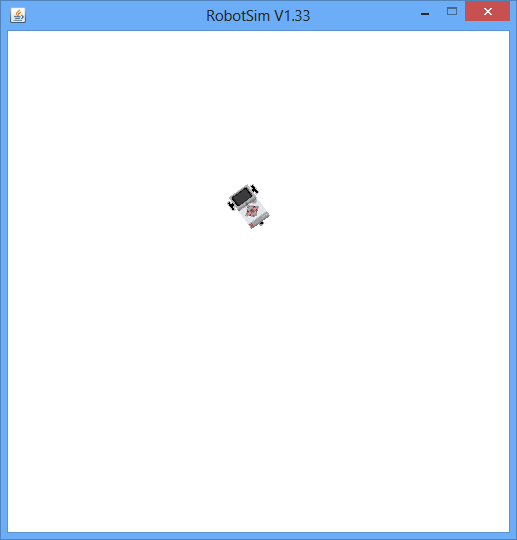
}

Output:-

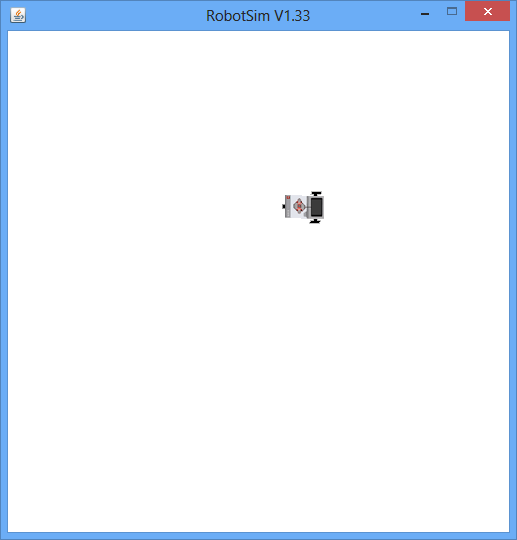
When the robot starts moving(initial State):-



Intermediate State:-



Move Right



**PRACTICAL NO-3**

TITLE : Write a program to create a robot with two motors and move it forward, left, right.

PROGRAM:-

import ch.aplu.robotsim.\*;

public class MovewithMotors

{

publicMovewithMotors ()

{

NxtRobot robot = new NxtRobot();

Motor motA = new Motor(MotorPort.A);

Motor motB = new Motor(MotorPort.B);

robot.addPart(motA);

robot.addPart(motB);

motA.forward();

motB.forward();

Tools.delay(2000);

motA.stop();

Tools.delay(1050);

motA.forward();

Tools.delay(2000);

motB.stop();

Tools.delay(1050);

motB.forward();

Tools.delay(2000);

robot.exit();

}

public static void main(String [] args)

{

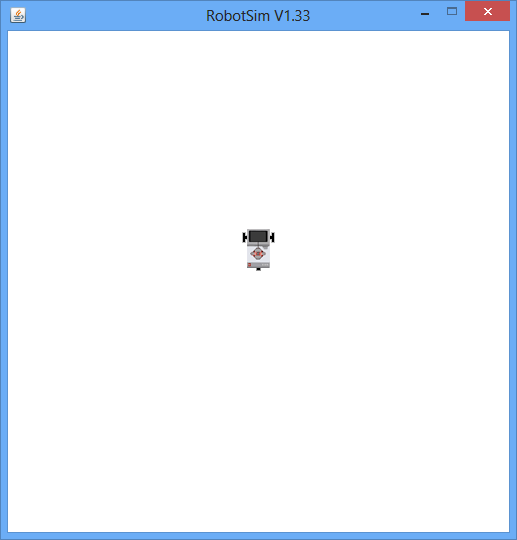
newMovewithMotors ();

}

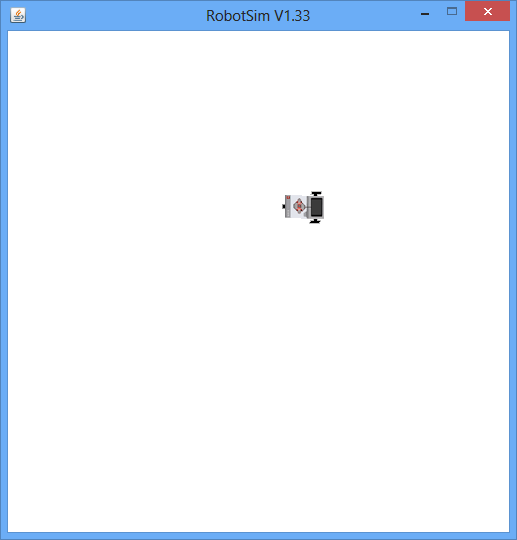
}

Output:-

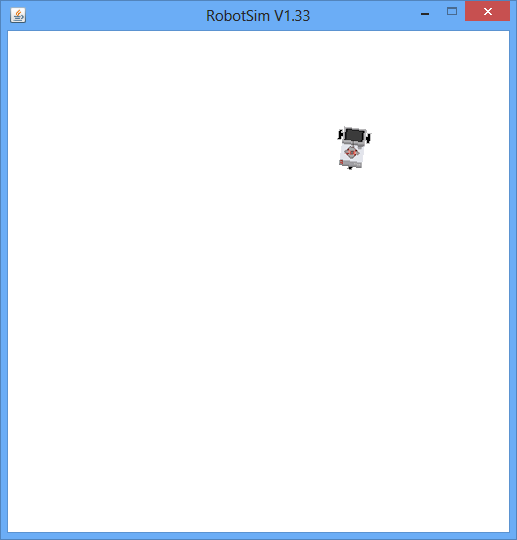
OUTPUT: MOVE FORWARD:



MOVE RIGHT



EXIT



**Practical no : 4**

Title :- Write a program to create a robot with light sensors to follow a line. In this program the robot has to follow the black line.

// LineFollower.java

import ch.aplu.robotsim.\*;

class LineFollower

{

LineFollower()

{

LegoRobot robot = new LegoRobot();

Gear gear = new Gear();

LightSensor ls = new LightSensor(SensorPort.S3);

robot.addPart(gear);

gear.setSpeed(20);

robot.addPart(ls);

while (true)

{

int v = ls.getValue();

if (v < 100) // black

gear.forward();

if (v > 300 && v < 750) // blue

gear.leftArc(0.05);

if (v > 800) // yellow

gear.rightArc(0.05);

}

}

public static void main(String[] args)

{

new LineFollower();

}

// ---------- Environment ----------------------

static

{

RobotContext.setStartPosition(50, 490);

RobotContext.setStartDirection(-90);

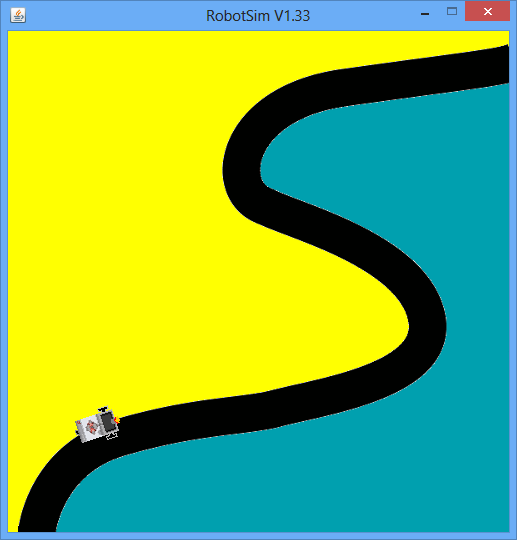
RobotContext.useBackground("sprites/road.gif");

}

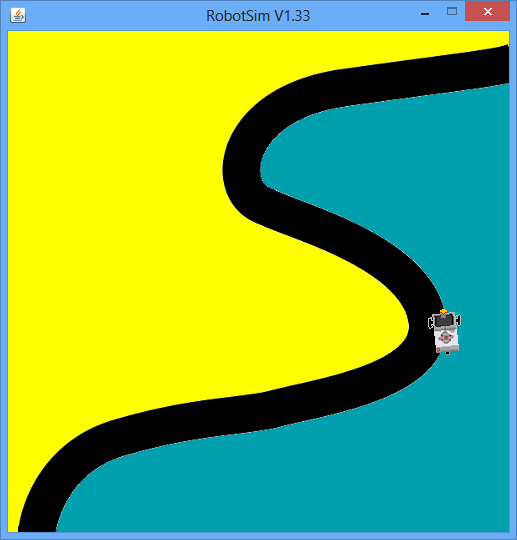
}

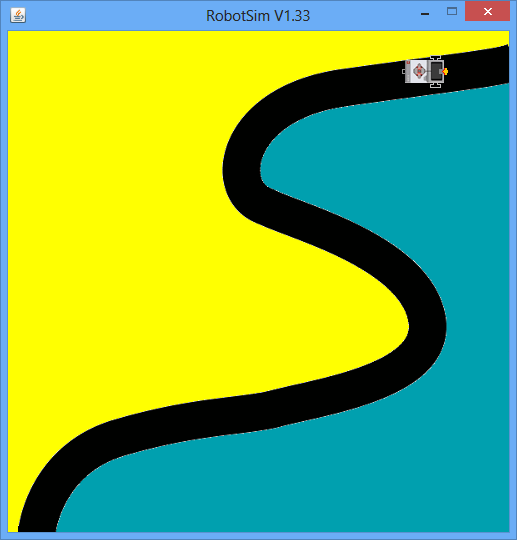
Output :-

Initial Condition:-Starting point of the Line



Intermediate condition:-



Final condition:-

**Practical No:-5**

**TITLE** :-Write a program to create a robot to that does a circle using two motors.

import ch.aplu.robotsim.\*;

class Circlem

{

Circlem()

{

NxtRobot robot = new NxtRobot();

Gear gear = new Gear();

robot.addPart(gear);

gear.setSpeed(60);

gear.leftArc(0.2,7000);

gear.rightArc(0.2);

Tools.delay(5000);

robot.exit();

}

public static void main(String[] args)

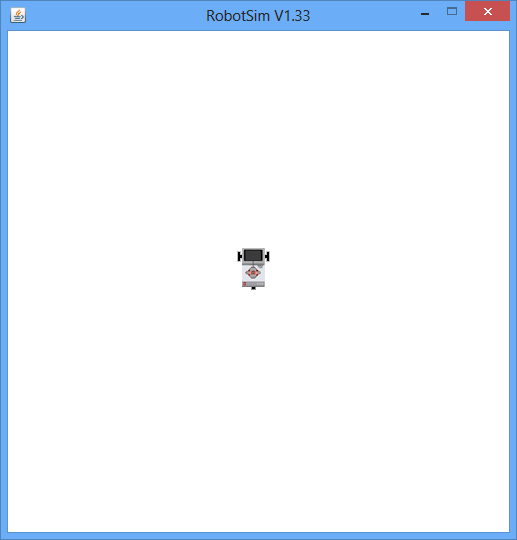
{

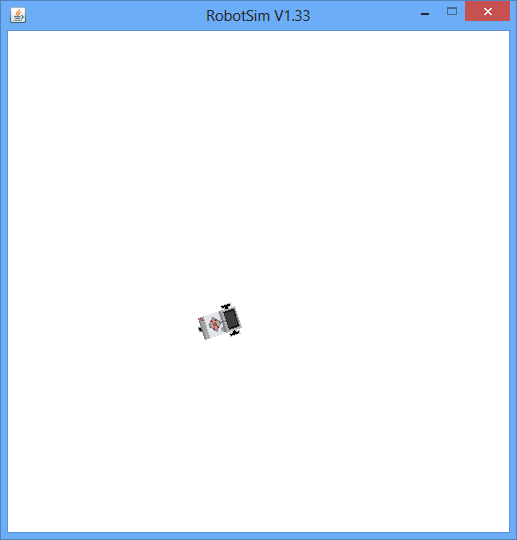
new Circlem();

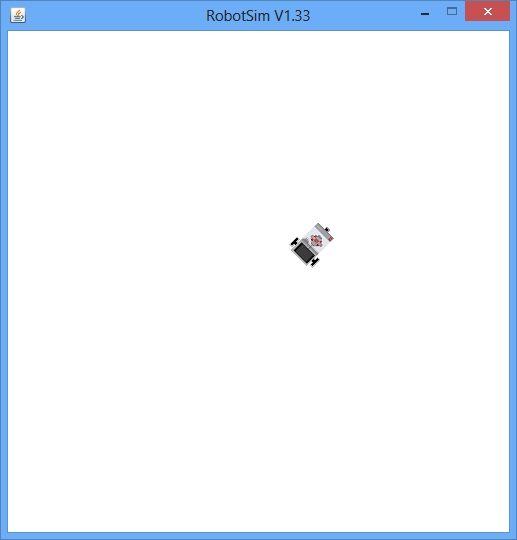
}

}

Output:-







**Practical no :-6**

**TITLE**:- Write a program to create a path following robot

// PathFinder.java

import ch.aplu.robotsim.\*;

public class PathFinder

{

publicPathFinder()

{

NxtRobot robot = new NxtRobot();

Gear gear = new Gear();

LightSensor ls1 = new LightSensor(SensorPort.S1); // right

LightSensor ls2 = new LightSensor(SensorPort.S2); // left

robot.addPart(gear);

robot.addPart(ls1);

robot.addPart(ls2);

gear.forward();

while (true)

{

intrightValue = ls1.getValue();

intleftValue = ls2.getValue();

int d = rightValue - leftValue;

if (d > 100) // left dark , turn right

gear.rightArc(0.1);

if (d < -100) // right dark, turn left

gear.leftArc(0.1);

if (d > -100 && d < 100 &&rightValue> 500)

gear.forward();

}

}

public static void main(String[] args)

{

new PathFinder();

}

// ------------------ Environment --------------------------

static

{

NxtContext.setStartPosition(250, 490);

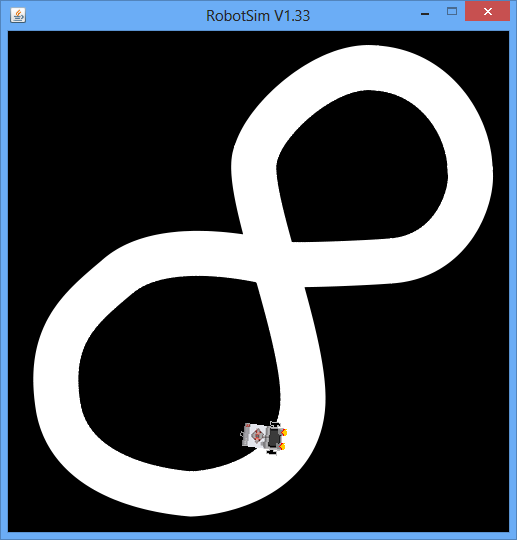
NxtContext.setStartDirection(-90);

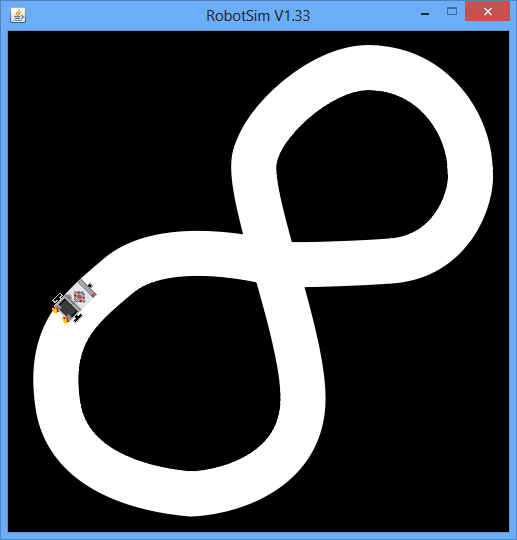
NxtContext.useBackground("sprites/path.gif");

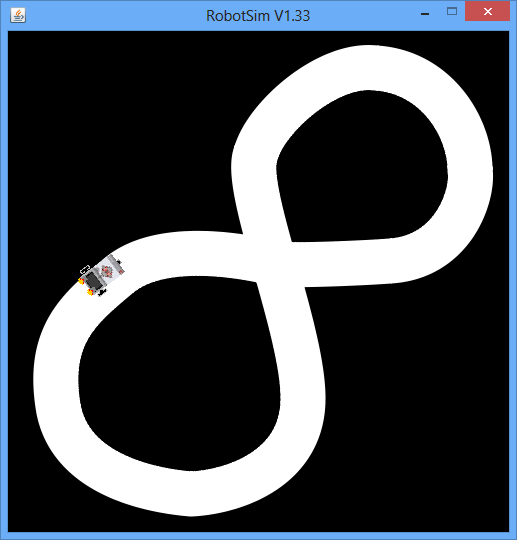
}

}

Output of the path follower:







**Practical No-7**

**Implement breadth-first search for specified problem**.

**Description:**

**breadth-first search:**

1. It is a simple strategy in which root node is expanded first, then all those are successors of root node are expanded next, then their successors and so on.
2. In general, all nodes are expanded at a given depth in the search tree before any nodes at next level are expanded.
3. It can be implemented using FIFO queue.
4. The FIFO queue puts all newly generated successors at the end of queue, which means that shallow nodes are expanded before deeper nodes.
5. This search is complete as it expands all the nodes.
6. It is not necessarily optimal. Every node that is generated must remain in memory because it is either part of fringe or is an ancestor of fringe node.
7. Thus space complexity same as time complexity.

Criteria BFS

Complete Yesa

Time O(bd+1)

Space O(bd+1)

Optimal Yesa

**Source Code:**

import java.util.\*;

import java.io.\*;

public class breadthfirst

{

ArrayList arr=new ArrayList();

String str[]=new String[2];

String path[]=new String[20];

int i,j,k=0;

public breadthfirst()

{

//Adding Name Of City To The Arraylist.

arr.add("Arad");

arr.add("Zerind");

arr.add("Sibiu");

arr.add("Timisoara");

arr.add("Rimnicu Vilcea");

arr.add("Fagaras");

arr.add("Lugoj");

arr.add("Craiova");

arr.add("Pitesti");

arr.add("Bucharest");

arr.add("Mehadia");

arr.add("Dobreta");

arr.add("Pitesti");

arr.add("Bucharest");

arr.add("Dobreta");

}

public void breadth()

{

if(arr.isEmpty())

System.out.println("Empty");

for(i=0;i<20;i++)

path[i]="";

str[0]="";

str[1]="";

//Representing Cities in the tree format.

System.out.println("\t\t\tArad\n");

System.out.println("Zerind\t\t\tSibiu\t\t\tTimisoara\n");

System.out.println("\t\tRimnicu Vilcea Fagaras\tLugoj\n");

System.out.println("\t\tCraiova Pitesti Bucharest\t Mehadia\n");

System.out.println("\tDobreta Pitesti Bucharest\t\tDobreta\n");

try

{

BufferedReader br=new BufferedReader(new

InputStreamReader(System.in));

System.out.println("Enter the initial node");

str[0]=br.readLine();

System.out.println("Enter the goal");

str[1]=br.readLine();

//To Find The Goal.

if(arr.contains(str[1]))

{

System.out.println("Goal is found");

//To Store the Entire Path from initial node to goal.

for(j=arr.indexOf(str[0]);j<=arr.indexOf(str[1]);j++)

{

path[k]=arr.get(j).toString();

k++;

}

}

}

catch(IOException e)

{}

System.out.print("Path is: ");

for(j=0;j<k;j++)

{

System.out.print(path[j]);

if(j!=k-1)

System.out.print("-->");

}

}

public static void main(String arg[])

{

breadthfirst b=new breadthfirst();

b.breadth();

}

}

**O/P:**

Arad

Zerind Sibiu Timisoara

Rimnicu Vilcea Fagaras Lugoj

Craiova Pitesti Bucharest Mehadia

Dobreta Pitesti Bucharest Dobreta

Enter the initial node:

Arad

Enter the goal node:

Bucharest

Goal is found

Path is: Arad-->Zerind-->Sibiu-->Timisoara-->Rimnicu Vilcea-->Fagaras-->Lugoj-

->Craiova-->Pitesti-->Bucharest

**Practical No:8**

**Problem Statement:**

**Implement depth-first search for specified problem.**

**Description:**

**Depth-first search:**

**1.** It always expands the deepest node in the current frindge of search tree.

**2.** The search proceeds immediately to deepest level of search tree where nodes have no successors.

**3.** As those nodes are expanded they are dropped from frindge so then search backs up to next shallowest node that still has unexplored successors.

**4.** This strategy is implemented with **LIFO** queue.

**5.** This search has very modest memory requirement.

**6.** This needs to store only a single path from root to leaf node along with unexpanded sibling nodes for each node on the path.

**7.** Once node has been expanded it can be removed from memory as soon as all its decendents have been fully explored.

**8.** It can make wrong choice & get stuck going down a very long or even infinite path when a different choice would lead to solution.

**9.** Hence DFS is not optimal.

**10.** If left subtree where of unbounded depth but contain no solution,DFS will terminate there. Hence it is not complete.

**Source Code:**

import java.util.\*;

import java.io.\*;

public class depthfirst

{

ArrayList arr=new ArrayList();

String str[]=new String[2];

String path[]=new String[20];

int i,j,k=0;

public depthfirst()

{

//Adding Name Of City To The Arraylist.

arr.add("Arad");

arr.add("Zerind");

arr.add("Sibiu");

arr.add("Rimnicu Vilcea");

arr.add("Craiova");

arr.add("Dobreta");

arr.add("Pitesti");

arr.add("Pitesti");

arr.add("Bucharest");

arr.add("Fagaras");

arr.add("Bucharest");

arr.add("Timisoara");

arr.add("Lugoj");

arr.add("Mehadia");

arr.add("Dobreta");

}

public void depthsearch()

{

if(arr.isEmpty())

System.out.println("Empty");

for(i=0;i<20;i++)

path[i]="";

str[0]="";

str[1]="";

//Representing Cities in the tree format.

System.out.println("\t\t\tArad\n");

System.out.println("Zerind\t\t\tSibiu\t\t\tTimisoara\n");

System.out.println("\t\tRimnicu Vilcea Fagaras\tLugoj\n");

System.out.println("\t\tCraiova Pitesti Bucharest\t Mehadia\n");

System.out.println("\tDobreta Pitesti Bucharest\t\tDobreta\n");

try

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the initial node");

str[0]=br.readLine();

System.out.println("Enter the goal");

str[1]=br.readLine();

if(arr.contains(str[1]))

{

System.out.println("Goal is found");

//To Store the Entire Path from initial node to goal.

for(j=arr.indexOf(str[0]);j<=arr.indexOf(str[1]);j++)

{

path[k]=arr.get(j).toString();

k++;

}

}

}

catch(IOException e)

{}

System.out.print("Path is: ");

for(j=0;j<k;j++)

{

System.out.print(path[j]);

if(j!=k-1)

System.out.print("-->");

}

}

public static void main(String arg[])

{

depthfirst b=new depthfirst();

b.depthsearch();

}

}

**O/P:**

Arad

Zerind Sibiu Timisoara

Rimnicu Vilcea Fagaras Lugoj

Craiova Pitesti Bucharest Mehadia

Dobreta Pitesti Bucharest Dobreta

Enter the initial node:

Arad

Enter the goal node:

Bucharest

Goal is found

Path is: Arad-->Zerind-->Sibiu-->Rimnicu Vilcea-->Craiova-->Dobreta-->Pitesti-->Bucharest

**Practical:-09**

**Problem-Statement:**

**Implement A\* search for specified problem.**

**Description:**

**2. A\* Search:**

1. The most widely known form of Best first search is called A\* search.
2. It evaluates node by combining g(n), the cost to get from the node to the goal.

f(n)=g(n)+h(n)

Where g (n) =Path cost from start node to node N

h (n)=Estimated cost of the cheapest path from node N to goal.

f (n)=Estimated cost of the cheapest solution through N.

1. To find cheapest solution, first try the node with lowest value of ‘f’.
2. It is complete and optimal, given a simple restriction on heuristic function h (n).
3. The restriction is to choose on ‘h’ that never overestimate the cost to reach the goal. Such ‘h’ is called admissible heuristic.
4. Along with admissibility heuristic function should posses one more property that

consistency .

**Source Code:**

import java.util.\*;

import java.io.\*;

class Astar

{

String str[]=new String[2];

String s1,s2="";

int no,i,j,min=1,temp,k=0;

String path[]=new String[50];

public Astar()throws IOException

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter the initial node:-\n");

str[0]=br.readLine();

path[k]=str[0];

System.out.print("Enter the Goal Node:-\n");

str[1]=br.readLine();

//Representing Cities in the tree format.

System.out.println("\t\t\tArad\n");

System.out.println("Zerind\t\t\tSibiu\t\t\tTimisoara\n");

System.out.println("\tOradea\tRimnicu Vilcea Fagaras\tLugoj\n");

System.out.println("\tCraiova Pitesti Bucharest\t Mehadia\n");

System.out.println(" Dobreta Pitesti Bucharest\t\t\tDobreta\n");

//Expand the appropriate node according to their minimum value.

while(!str[0].equals(str[1]))

{

System.out.print(s2);

System.out.println("Enter the no. of node of" +" "+ str[0] );

s1=br.readLine();

no=Integer.parseInt(s1);

//System.out.println(no);

String st[][]=new String[no][3];

int arr[]=new int[no];

System.out.println("Enter the successor nodes of" +" "+ str[0]+" " +"With their straight line distance to goal node h(n) and path cost from start node i.e. h(n) ");

for(i=0;i<no;i++)

{

for(j=0;j<3;j++)

{

st[i][j]=br.readLine();

}

}

//For Storing the values h(n) & f(n) of various node.

for(i=0;i<no;i++)

{

arr[i]=Integer.parseInt(st[i][1])+Integer.parseInt(st[i][2]);

}

//To find the minimum value of various node.

for(i=0;i<no;i++)

{

for(j=i+1;j<no;j++)

{

if(arr[i]<arr[j])

{

temp=arr[i];

arr[j]=arr[j];

arr[i]=temp;

}

else

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

min=arr[0];

//To find out the node name having minimum value.

for(i=0;i<no;i++)

{

if((Integer.parseInt(st[i][1])+Integer.parseInt(st[i][2]))==min)

str[0]=st[i][0];

s2="Node"+" "+str[0]+" "+"has smallest value therefore ";

}

k++;

path[k]=str[0];

}

System.out.println("The path from initial node to goal node is:");

for(i=0;i<k;i++)

System.out.print(path[i]+"--->");

System.out.print(str[1]);

}

public static void main(String arg[])throws IOException

{

Astar a=new Astar();

}

}

**O/p:**

Enter the initial node:-

Arad

Enter the Goal Node:-

Bucharest

Arad

Zerind Sibiu Timisoara

Oradea Rimnicu Vilcea Fagaras Lugoj

Craiova Pitesti Bucharest Mehadia

Dobreta Pitesti Bucharest Dobreta

Enter the no. of node of Arad

3

Enter the successor nodes of Arad With their straight line distance to goal node

h(n) and path cost from start node i.e. h(n)

Sibiu

253

140

Timisoara

329

118

Zerind

374

75

Node Sibiu has smallest value therefore Enter the no. of node of Sibiu

4

Enter the successor nodes of Sibiu With their straight line distance to goal nod

e h(n) and path cost from start node i.e. h(n)

Arad

366

280

Fagaras

176

239

Oradea

380

291

Rimnicu Vilcea

193

220

Node Rimnicu Vilcea has smallest value therefore Enter the no. of node of Rimni

cu Vilcea

3

Enter the successor nodes of Rimnicu Vilcea With their straight line distance to

goal node h(n) and path cost from start node i.e. h(n)

Craiova

366

160

Pitesti

317

100

Sibiu

253

300

Node Pitesti has smallest value therefore Enter the no. of node of Pitesti

3

Enter the successor nodes of Pitesti With their straight line distance to goal n

ode h(n) and path cost from start node i.e. h(n)

Bucharest

0

418

Craiova

160

455

Rimnicu Vilcea

193

414

The path from initial node to goal node is:

Arad--->Sibiu--->Rimnicu Vilcea--->Pitesti--->Bucharest