```java
import java.awt.Color;
import java.awt.Container;
import java.awt.Image;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Set;


import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;

public class Astar {

	//declarations
	public static Timer t1;
	public static JFrame fr;
	public static JLabel map,map1;
	public static Container c;
	public static Image backI;
	public static ImageIcon backi;
	public static JLabel
oz,za,at,as,tl,lm,md,dc,cp,gb,pb,rc,rp,sr,sf,os,fb,os1,fb1,bu,lblUh,hk,vu,lv,nl,lblHn
Gn;
	public static JLabel lblRomaniaMapA,initial;
	private JLabel Arad;
	private JLabel Sibiu;
	private JLabel RiminicuVilcea;
	private JLabel Pitesti;
	private JLabel Bucharest;
	private JLabel lblCity;
	private JLabel lblFn;
	public static JLabel aradf,sibiuf, riminiuf,pitestif,bucharestf;
	public static int loop,ifcase,call;
	public static List<Nodes> list;
	private static JTextArea textArea;

	//constructor for initialization of GUI
	public Astar() throws Exception{

		fr=new JFrame();
		fr.getContentPane().setLayout(null);
		c=fr.getContentPane();
		map=new JLabel();
		map1=new JLabel();
		map.setBounds(100, 30, 680, 470);
		map1.setBounds(100, 30, 680, 470);
```

```
             //background image
                        backI = ImageIO.read(new File("/BE/CL
pract/src/images/map.png"));
                        backI = backI.getScaledInstance(map.getWidth(),
map.getHeight(),Image.SCALE_REPLICATE);
                        backi = new ImageIcon(backI);
                        map.setIcon(backi);
                        map1.setIcon(backi);

                        initial = new JLabel("Initial State",JLabel.CENTER);
                        initial.setFont(new Font("Tahoma", Font.BOLD, 15));
                        initial.setBounds(281, 30, 357, 26);

                        fr.getContentPane().add(initial);
                        c.add(map1);

                        textArea = new JTextArea();
                        textArea.setBounds(776, 108, 216, 298);
                        textArea.setEnabled(false);
                        textArea.setBorder(null);
                        textArea.setDisabledTextColor(Color.black);
                        fr.getContentPane().add(textArea);

                        //all labels which overrides the edges
oz = new JLabel("");
oz.setBounds(169, 74, 33, 34);
oz.setOpaque(true);



za = new JLabel("");
za.setBounds(143, 129, 46, 34);
za.setOpaque(true);


at = new JLabel("");
at.setBounds(130, 186, 33, 97);
at.setOpaque(true);

as = new JLabel("");
as.setBounds(169, 180, 106, 34);
as.setOpaque(true);


tl = new JLabel("");
tl.setBounds(169, 298, 73, 53);
fr.getContentPane().add(tl);
lm = new JLabel("");
lm.setBounds(230, 347, 46, 34);
fr.getContentPane().add(lm);
md = new JLabel("");
md.setBounds(230, 404, 46, 39);
```

```
fr.getContentPane().add(md);

dc = new JLabel("");
dc.setBounds(259, 441, 92, 34);
fr.getContentPane().add(dc);
cp = new JLabel("");
cp.setBounds(376, 361, 64, 97);
fr.getContentPane().add(cp);
gb = new JLabel("");
gb.setBounds(500, 439, 64, 46);
fr.getContentPane().add(gb);
pb = new JLabel("");
pb.setBounds(450, 361, 64, 46);
fr.getContentPane().add(pb);


rc = new JLabel("");
rc.setBounds(333, 314, 40, 131);
fr.getContentPane().add(rc);

rp = new JLabel("rp");
rp.setBounds(343, 295, 83, 53);
fr.getContentPane().add(rp);

sr = new JLabel("");
sr.setBounds(291, 233, 46, 50);
fr.getContentPane().add(sr);
sf = new JLabel("");
sf.setBounds(309, 217, 100, 20);
fr.getContentPane().add(sf);

os = new JLabel("");
os.setBounds(230, 72, 65, 125);
fr.getContentPane().add(os);
fb = new JLabel("");
fb.setBounds(466, 243, 73, 118);
fr.getContentPane().add(fb);

os1 = new JLabel("");
os1.setBounds(211, 71, 46, 51);
fr.getContentPane().add(os1);

fb1 = new JLabel("");
fb1.setBounds(433, 255, 46, 40);
fr.getContentPane().add(fb1);


bu = new JLabel("");
bu.setBounds(550, 373, 55, 34);
fr.getContentPane().add(bu);
lblUh = new JLabel("");
lblUh.setBounds(624, 366, 73, 20);
fr.getContentPane().add(lblUh);

hk = new JLabel("");
```

```java
hk.setBounds(693, 385, 55, 63);
fr.getContentPane().add(hk);
vu = new JLabel("");
vu.setBounds(602, 250, 95, 115);
fr.getContentPane().add(vu);
lv = new JLabel("");
lv.setBounds(629, 166, 46, 63);
fr.getContentPane().add(lv);
nl = new JLabel("");
nl.setBounds(550, 108, 73, 55);
fr.getContentPane().add(nl);


lblRomaniaMapA = new JLabel("Romania Map- arad to bucharest");
lblRomaniaMapA.setFont(new Font("Tahoma", Font.PLAIN, 16));
lblRomaniaMapA.setBounds(357, 0, 248, 33);
fr.getContentPane().add(lblRomaniaMapA);

//timer for different states display
t1=new Timer(1500, new ActionListener() {

	@Override
	public void actionPerformed(ActionEvent arg0) {
		map1.setVisible(false);
		Astar.initial.setText("Intermediate state-Finding Optimal Path");

		init();

		t1.stop();
	}
});


tl.setOpaque(true);
cp.setOpaque(true);
dc.setOpaque(true);
tl.setOpaque(true);
gb.setOpaque(true);
lm.setOpaque(true);
md.setOpaque(true);
pb.setOpaque(true);
rc.setOpaque(true);
rp.setOpaque(true);
sr.setOpaque(true);
sf.setOpaque(true);
os.setOpaque(true);
os1.setOpaque(true);
fb.setOpaque(true);
fb1.setOpaque(true);
bu.setOpaque(true);
vu.setOpaque(true);
hk.setOpaque(true);
nl.setOpaque(true);
lblUh.setOpaque(true);
lv.setOpaque(true);
```

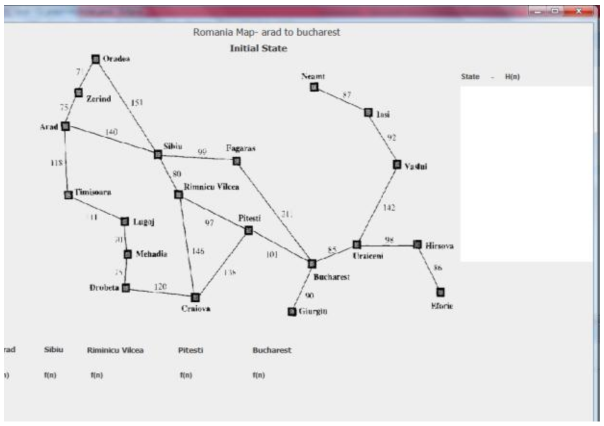
```java
map1.setVisible(true);



Arad = new JLabel("Arad");
Arad.setFont(new Font("Tahoma", Font.BOLD, 12));
Arad.setBounds(49, 545, 46, 20);
fr.getContentPane().add(Arad);
Sibiu = new JLabel("Sibiu");
Sibiu.setFont(new Font("Tahoma", Font.BOLD, 12));
Sibiu.setBounds(123, 548, 46, 14);
fr.getContentPane().add(Sibiu);
RiminicuVilcea = new JLabel("Riminicu Vilcea");
RiminicuVilcea.setFont(new Font("Tahoma", Font.BOLD, 12));
RiminicuVilcea.setBounds(190, 549, 105, 14);
fr.getContentPane().add(RiminicuVilcea);
Pitesti = new JLabel("Pitesti");
Pitesti.setFont(new Font("Tahoma", Font.BOLD, 12));
Pitesti.setBounds(333, 549, 64, 14);
fr.getContentPane().add(Pitesti);
Bucharest = new JLabel("Bucharest");
Bucharest.setFont(new Font("Tahoma", Font.BOLD, 12));
Bucharest.setBounds(450, 549, 64, 14);
fr.getContentPane().add(Bucharest);
lblCity = new JLabel("City");
lblCity.setBounds(10, 549, 46, 14);
fr.getContentPane().add(lblCity);
lblFn = new JLabel("f(n)");
lblFn.setBounds(10, 591, 46, 14);
fr.getContentPane().add(lblFn);
aradf = new JLabel("f(n)");
aradf.setBounds(49, 591, 46, 14);
fr.getContentPane().add(aradf);
sibiuf = new JLabel("f(n)");
sibiuf.setBounds(123, 591, 46, 14);
fr.getContentPane().add(sibiuf);
riminiuf = new JLabel("f(n)");
riminiuf.setBounds(196, 591, 46, 14);
fr.getContentPane().add(riminiuf);
pitestif = new JLabel("f(n)");
pitestif.setBounds(336, 591, 46, 14);
fr.getContentPane().add(pitestif);
bucharestf = new JLabel("f(n)");
bucharestf.setBounds(450, 591, 46, 14);
fr.getContentPane().add(bucharestf);


lblHnGn = new JLabel("State      -      H(n)");
lblHnGn.setBounds(776, 84, 234, 14);
fr.getContentPane().add(lblHnGn);
fr.setBounds(0, 0, 1000, 800);
fr.setVisible(true);
fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

t1.start();

c.add(as);
c.add(at);
c.add(za);
c.add(oz);
c.add(map);

       }


public void init(){
       Nodes n1 = new Nodes("Arad");
       Nodes n2 = new Nodes("Zerind");
       Nodes n3 = new Nodes("Oradea");
       Nodes n4 = new Nodes("Sibiu");
       Nodes n5 = new Nodes("Fagaras");
       Nodes n6 = new Nodes("Rimnicu Vilcea");
       Nodes n7 = new Nodes("Pitesti");
       Nodes n8 = new Nodes("Timisoara");
       Nodes n9 = new Nodes("Lugoj");
       Nodes n10 = new Nodes("Mehadia");
       Nodes n11 = new Nodes("Drobeta");
       Nodes n12 = new Nodes("Craiova");
       Nodes n13 = new Nodes("Bucharest");
       Nodes n14 = new Nodes("Giurgiu");


       // initialize the edges
       n1.adjacencies = new Edges[] { new Edges(n2, 75), new Edges(n4, 140),
               new Edges(n8, 118) };

       n2.adjacencies = new Edges[] { new Edges(n1, 75), new Edges(n3, 71) };

       n3.adjacencies = new Edges[] { new Edges(n2, 71), new Edges(n4, 151) };

       n4.adjacencies = new Edges[] { new Edges(n1, 140), new Edges(n5, 99),
               new Edges(n3, 151), new Edges(n6, 80), };

       n5.adjacencies = new Edges[] { new Edges(n4, 99), new Edges(n13, 211) };

       n6.adjacencies = new Edges[] { new Edges(n4, 80), new Edges(n7, 97),
               new Edges(n12, 146) };

       n7.adjacencies = new Edges[] { new Edges(n6, 97), new Edges(n13, 101),
               new Edges(n12, 138) };

       n8.adjacencies = new Edges[] { new Edges(n1, 118), new Edges(n9, 111) };

       n9.adjacencies = new Edges[] { new Edges(n8, 111), new Edges(n10, 70) };
```

```
        n10.adjacencies = new Edges[] { new Edges(n9, 70), new Edges(n11, 75) };

        n11.adjacencies = new Edges[] { new Edges(n10, 75), new Edges(n12, 120) };

        n12.adjacencies = new Edges[] { new Edges(n11, 120), new Edges(n6, 146),
                new Edges(n7, 138) };

        n13.adjacencies = new Edges[] { new Edges(n7, 101), new Edges(n14, 90),
                new Edges(n5, 211) };

        n14.adjacencies = new Edges[] { new Edges(n13, 90) };

        UniformCostSearch(n1, n13);

        List<Nodes> path = printPath(n13);

        System.out.println("\nPath: " + path);


}

public static void main(String[] args) throws Exception {
        Astar a=new Astar();

}

public static void UniformCostSearch(Nodes source, Nodes goal) {
call++;
list = new ArrayList<Nodes>();
source.pathCost = 0;
PriorityQueue<Nodes> queue = new PriorityQueue<Nodes>(20,
        new Comparator<Nodes>() {

            // override compare method
            public int compare(Nodes i, Nodes j) {
                if ((i.pathCost > j.pathCost)) {
                    return 1;
                }

                else if (i.pathCost < j.pathCost) {
                    return -1;
                }

                else {
                    return 0;
                }

            }

            }

);

queue.add(source);
```

```java
Set<Nodes> explored = new HashSet<Nodes>();
List<Nodes> path = new ArrayList<Nodes>();

// while frontier is not empty
do {
Loop++;
    path.clear();
    Nodes current = queue.poll();
    explored.add(current);
    for (Nodes node = current; node != null; node = node.parent) {
        path.add(node);
    }
    if (current.value.equals(goal.value)) {
       ifcase++;
        goal.parent = current.parent;
        goal.pathCost = current.pathCost;
        break;

    }

    for (Edges e : current.adjacencies) {
       Loop++;
        Nodes child = e.target;
        double cost = e.cost;
        if ((queue.contains(child) || explored.contains(child))
                && !path.contains(child)) {
             ifcase++;
            Nodes n = new Nodes(child);
            List.add(n);
            List.get(List.size() - 1).pathCost = current.pathCost
                    + cost;

            List.get(List.size() - 1).parent = current;
            queue.add(List.get(List.size() - 1));
            //System.out.println(e.target.value+"-"+e.target.pathCost);

            textArea.append("\n"+e.target.value+"    -
"+Double.toString(e.target.pathCost));



        //  textArea.append(list.get(list.size()).toString());


          // System.out.println(queue);
        } else if (!path.contains(child)) {
             ifcase++;
            child.pathCost = current.pathCost + cost;
            child.parent = current;
            queue.add(child);


           // System.out.println(queue);
        }
    }
} while (!queue.isEmpty());

System.out.println("Loop count::"+loop);
System.out.println("if case executed::"+ifcase+" times");
System.out.println("call to uniformCostSearch method::"+call);
}

public static List<Nodes> printPath(Nodes target) {
List<Nodes> path = new ArrayList<Nodes>();
for (Nodes node = target; node != null; node = node.parent) {
    path.add(node);
    System.out.println("path------\t"+node);
}
Collections.reverse(path);
return path;
}
}
class Nodes {
public final String value;
public double pathCost;
public Edges[] adjacencies;
public Nodes parent;
public static Timer t1,t2;
public static String oldvalue;
public Nodes(String val) {
value = val;
}

public Nodes(Nodes node) {
int i = 0;
adjacencies = new Edges[node.adjacencies.length];
value = node.value;
pathCost = node.pathCost;
for (Edges e : node.adjacencies) {
    adjacencies[i++] = e;
}
parent = node.parent;
}
public String toString() {
      t1=new Timer(1000, new ActionListener() {
             @Override
             public void actionPerformed(ActionEvent arg0) {
                    oldvalue=value;
                    if(value.equals("Arad")){
                           Astar.initial.setText("Final state- Optimal Path");
                           Astar.as.setVisible(false);
                           Astar.aradf.setText(Double.toString(pathCost));
                           }
                           if(value.equals("Sibiu")){
                                  Astar.sr.setVisible(false);
                                  Astar.sibiuf.setText(Double.toString(pathCost));
                           }
                           if(value.equals("Rimnicu Vilcea")){
                                  Astar.rp.setVisible(false);
```

```
                                 Astar.riminiuf.setText(Double.toString(pathCost));
                          }
                          if(value.equals("Pitesti")){
                                 Astar.pb.setVisible(false);
                                 Astar.pitestif.setText(Double.toString(pathCost));
                          }
                          if(value.equals("Bucharest")){
                                 Astar.bucharestf.setText(Double.toString(pathCost));
                          }
                          t1.stop();
                   }
       });

       t1.start();
return value+"-"+ pathCost + "    ";
}
}
class Edges {
public final double cost;
public final Nodes target;
public Edges(Nodes targetNode, double costVal) {
cost = costVal;
target = targetNode;
}
}
```

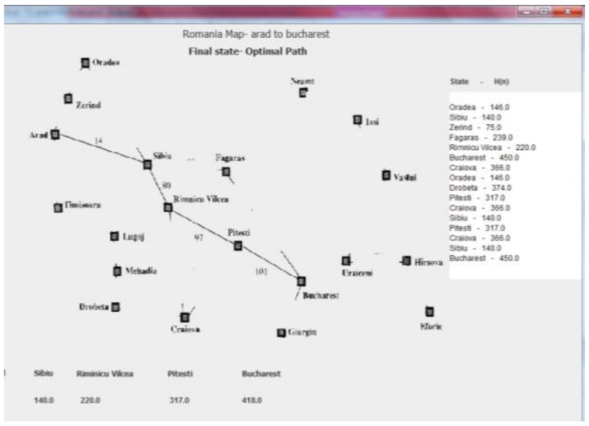
o/p:


Romania Map- arad to bucharest

Initial State

State - H(n)

```
rad     Sibiu   Riminicu Vilcea   Pitesti   Bucharest
)       f(n)    f(n)              f(n)      f(n)
```


Romania Map- arad to bucharest

Final state- Optimal Path

State - H(n)

```
Oradea  -  146.0
Sibiu  -  140.0
Zerind  -  75.0
Fagaras  -  239.0
Rimnicu Vilcea  -  220.0
Bucharest  -  450.0
Craiova  -  366.0
Oradea  -  146.0
Drobeta  -  374.0
Pitesti  -  317.0
Craiova  -  366.0
Sibiu  -  140.0
Pitesti  -  317.0
Craiova  -  366.0
Sibiu  -  140.0
Bucharest  -  450.0
```

```
Sibiu   Riminicu Vilcea   Pitesti   Bucharest
140.0   220.0             317.0     418.0
```