```java
import java.util.PriorityQueue;
import java.util.HashSet;
import java.util.Set;
import java.util.List;
import java.util.Comparator;
import java.util.ArrayList;
import java.util.Collections;

public class AstarSearchAlgo{


        //h scores is the stright-line distance from the current city to Bucharest
        public static void main(String[] args){

                //initialize the graph base on the Romania map
                Node n1 = new Node("Arad",366);
                Node n2 = new Node("Zerind",374);
                Node n3 = new Node("Oradea",380);
                Node n4 = new Node("Sibiu",253);
                Node n5 = new Node("Fagaras",178);
                Node n6 = new Node("Rimnicu Vilcea",193);
                Node n7 = new Node("Pitesti",98);
                Node n8 = new Node("Timisoara",329);
                Node n9 = new Node("Lugoj",244);
                Node n10 = new Node("Mehadia",241);
                Node n11 = new Node("Drobeta",242);
                Node n12 = new Node("Craiova",160);
                Node n13 = new Node("Bucharest",0);
                        Node n14 = new Node("Giurgiu",77);

                //initialize the edges

                //Arad
                n1.adjacencies = new Edge[]{
                        new Edge(n2,75),
                        new Edge(n4,140),
                        new Edge(n8,118)
                };

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


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

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

```

```
61
62                  //Fagaras
63                 n5.adjacencies = new Edge[]{
64                         new Edge(n4,99),
65
66                         //178
67                         new Edge(n13,211)
68                 };
69
70                  //Rimnicu Vilcea
71                 n6.adjacencies = new Edge[]{
72                         new Edge(n4,80),
73                         new Edge(n7,97),
74                         new Edge(n12,146)
75                 };
76
77                  //Pitesti
78                 n7.adjacencies = new Edge[]{
79                         new Edge(n6,97),
80                         new Edge(n13,101),
81                         new Edge(n12,138)
82                 };
83
84                  //Timisoara
85                 n8.adjacencies = new Edge[]{
86                         new Edge(n1,118),
87                         new Edge(n9,111)
88                 };
89
90                  //Lugoj
91                 n9.adjacencies = new Edge[]{
92                         new Edge(n8,111),
93                         new Edge(n10,70)
94                 };
95
96                  //Mehadia
97                 n10.adjacencies = new Edge[]{
98                         new Edge(n9,70),
99                         new Edge(n11,75)
100                };
101
102                 //Drobeta
103                n11.adjacencies = new Edge[]{
104                        new Edge(n10,75),
105                        new Edge(n12,120)
106                };
107
108                 //Craiova
109                n12.adjacencies = new Edge[]{
110                        new Edge(n11,120),
111                        new Edge(n6,146),
112                        new Edge(n7,138)
113                };
114
115                //Bucharest
116                n13.adjacencies = new Edge[]{
117                        new Edge(n7,101),
118                        new Edge(n14,90),
119                        new Edge(n5,211)
120                };
121
122                 //Giurgiu
123                n14.adjacencies = new Edge[]{
124                        new Edge(n13,90)
125                };
126
127                AstarSearch(n1,n13);
128
129                List<Node> path = printPath(n13);
130
```

```
131                         System.out.println("Path: " + path);
132
133
134         }
135
136         public static List<Node> printPath(Node target){
137                 List<Node> path = new ArrayList<Node>();
138
139         for(Node node = target; node!=null; node = node.parent){
140             path.add(node);
141         }
142
143         Collections.reverse(path);
144
145         return path;
146         }
147
148         public static void AstarSearch(Node source, Node goal){
149
150                 Set<Node> explored = new HashSet<Node>();
151
152                 PriorityQueue<Node> queue = new PriorityQueue<Node>(20,
153                         new Comparator<Node>(){
154                                 //override compare method
155                  public int compare(Node i, Node j){
156                     if(i.f_scores > j.f_scores){
157                         return 1;
158                     }
159
160                     else if (i.f_scores < j.f_scores){
161                         return -1;
162                     }
163
164                     else{
165                         return 0;
166                     }
167                  }
168
169                         }
170                         );
171
172                 //cost from start
173                 source.g_scores = 0;
174
175                 queue.add(source);
176
177                 boolean found = false;
178
179                 while((!queue.isEmpty())&&(!found)){
180
181                         //the node in having the lowest f_score value
182                         Node current = queue.poll();
183
184                         explored.add(current);
185
186                         //goal found
187                         if(current.value.equals(goal.value)){
188                                 found = true;
189                         }
190
191                         //check every child of current node
192                         for(Edge e : current.adjacencies){
193                                 Node child = e.target;
194                                 double cost = e.cost;
195                                 double temp_g_scores = current.g_scores + cost;
196                                 double temp_f_scores = temp_g_scores + child.h_score
197
198
199                                 /*if child node has been evaluated and
200                                 the newer f_score is higher, skip*/
```

```
                                        the newer f_score is higher, skip!

                                        if((explored.contains(child)) &&
                                                (temp_f_scores >= child.f_scores)){
                                                continue;
                                        }

                                        /*else if child node is not in queue or
                                        newer f_score is lower*/

                                        else if((!queue.contains(child)) ||
                                                (temp_f_scores < child.f_scores)){

                                                child.parent = current;
                                                child.g_scores = temp_g_scores;
                                                child.f_scores = temp_f_scores;

                                                if(queue.contains(child)){
                                                        queue.remove(child);
                                                }

                                                queue.add(child);

                                        }

                                }

                        }

                }

        }

}

class Node{

        public final String value;
        public double g_scores;
        public final double h_scores;
        public double f_scores = 0;
        public Edge[] adjacencies;
        public Node parent;

        public Node(String val, double hVal){
                value = val;
                h_scores = hVal;
        }

        public String toString(){
                return value;
        }

}

class Edge{
        public final double cost;
        public final Node target;

        public Edge(Node targetNode, double costVal){
                target = targetNode;
                cost = costVal;
        }
}
```

Execute Mode, Version, Inputs & Arguments

CommandLine Arguments

Stdin Inputs

Result

CPU Time: 0.09 sec(s), Memory: 33216 kilobyte(s) **compiled and executed in 0.591 sec(s)**

```
Path: [Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest]
```

Note:

1. For file operations - upload files using upload button, Files will be upload to /uploads folder. You can read those files in program from /uploads folder. To write a file from your program, write files to '/myfiles' folder. Please note the uploaded files stored in the server only for the current session.

2. For detailed documentation check - Our Documentation, or check our Youtube channel.

- Dark Theme available. Click on "•••" icon near execute button and select dark theme.
- You can embed code from JDoodle directly into your website/blog. **Click here** to know more.
- JDoodle offers an API service. You can execute programs just by calling our API. **Click here** to know more.
- If you like JDoodle, Please share us in Social Media. **Click here** to share.
- Check our **Documentation Page** for more info.

JDoodle is serving the programming community since 2013

- Looking for Multiple Files, Connecting to DB, Debugging, etc.?
- Are you building any innovative solution for your students or recruitment?
- Want to run JDoodle in-house?
- Custom Domain, White labelled pages for your institute?

Contact us - We are happy to help!