```java
 * Implementation of Graph Algorithms
import java.util.LinkedList;

public class GraphAlgos
{
    public static void bfs(Graph graph, String sourceLabel)
    {
        for (Vertex v : graph.getVertices()) {
            v.reset();
        }

        Vertex source = graph.getVertex(sourceLabel);
        source.parent = null;
        source.distance = 0.0;

        Queue<Vertex> queue = new LinkedList<>();
        queue.add(source);
        source.visited = true;

        while (!queue.isEmpty()) {
            Vertex v = queue.poll();

           System.out.print(v + " ");

            for (Edge edge : graph.getAdjacent(v)) {
                Vertex u = edge.getTarget();
                if (!u.visited) {
                    u.visited = true;
                    u.parent = v;
                    u.distance = v.distance + 1;
                    queue.add(u);
                }
            }
        }
    }

    public static void dfs(Graph graph, String sourceLabel) {
        Vertex source = graph.getVertex(sourceLabel);

```

```
45          for (Vertex v : graph.getVertices()) {
46              v.visited = false;
47              v.parent = null;
48          }
49
50          dfs(graph, source);
51      }
52
53
54      public static void dfs(Graph graph, Vertex curr)
55      {
56          curr.visited = true;
57
58          String parentLabel = curr.parent == null ? "*" :
    curr.parent.label;
59
60          double edgeWeight = 0;
61          if (curr.parent != null) {
62              for (Edge edge : graph.getAdjacent(curr.parent)) {
63                  if (edge.getTarget().equals(curr)) {
64                      edgeWeight = edge.getWeight();
65                      break;
66                  }
67              }
68          }
69
70          System.out.print(curr.label + ":" + edgeWeight + ":" +
    parentLabel + " ");
71
72          for (Edge edge : graph.getAdjacent(curr)) {
73              Vertex u = edge.getTarget();
74              if (!u.visited) {
75                  u.parent = curr;
76                  dfs(graph, u);
77              }
78          }
79      }
80
81      public static void dijkstra(Graph graph, String sourceLabel)
```

```
 82    {
 83        for (Vertex v : graph.getVertices()) {
 84            v.reset();
 85        }
 86        Vertex source = graph.getVertex(sourceLabel);
 87        source.parent = null;
 88        source.distance = 0.0;
 89
 90        PriorityQueue<Vertex> queue = new PriorityQueue<>(new
   VertexComparator());
 91        queue.add(source);
 92
 93        while (!queue.isEmpty()) {
 94            Vertex v = queue.poll();
 95
 96            System.out.print(v + " ");
 97            for (Edge edge : graph.getAdjacent(v)) {
 98                Vertex u = edge.getTarget();
 99                Double newDist = v.distance + edge.getWeight();
100
101                if ( u.distance > newDist ) {
102                    u.distance = newDist;
103                    u.parent = v;
104                    queue.add(u);
105                }
106            }
107            Vertex vertex = queue.poll();
108            printPathRec(source, vertex);
109            printPathLoop(source, vertex);
110        }
111    }
112
113    public static void printPathLoop(Vertex startVertex, Vertex
   destVertex)
114    {
115        String path = "";
116        Double totalLength = 0.0;
117
118        Vertex current = destVertex;
```

```
119
120         while (current != startVertex) {
121             path = current.label + " <--" + current.distance +
    "-- " + path;
122             totalLength += current.distance;
123             current = current.parent;
124         }
125         path = startVertex.label + " " + path;
126
127         path += "(total length " + totalLength + ")";
128
129         System.out.println(path);
130     }
131
132     public static void printPathRec(Vertex startVertex, Vertex
    destVertex)
133     {
134         if (destVertex == startVertex) {
135             System.out.print(startVertex.label);
136         }
137         else {
138             printPathRec(startVertex, destVertex.parent);
139             double weight = destVertex.distance -
    destVertex.parent.distance;
140             System.out.println( "--" + weight + "--> " +
    destVertex.label );
141         }
142     }
143
144     public static Graph prim(Graph graph, String source)
145     {
146         Graph primGraph = new Graph();
147         return primGraph.printMST();
148     }
149
150     public static Graph kruskal(Graph graph)
151     {
152         return graph;
153
```

```
154     }
155
156     public static int[][] initPredecessor(double[][] D)
157     {
158         return null;
159
160     }
161
162     public static void floydWarshall(double[][] D, int[][] P)
163     {
164
165     }
166
167     public static void floydWarshall(Graph G)
168     {
169
170     }
171
172     public static void printAllPaths(double[][] D, int[][] P,
    String[] labels)
173     {
174
175     }
176
177     public static void printPathLoop(int i, int j, double[][]
    D, int[][] P, String[] labels)
178     {
179
180     }
181
182     public static void printPath(int i, int j, double[][] D,
    int[][] P, String[] labels)
183     {
184
185     }
186 }
187
```