

UndirectedGraph.java

```

1 //*****
2 //
3 // File:    UndirectedGraph.java
4 // Package: ---
5 // Unit:    Class UndirectedGraph
6 //
7 //*****
8
9 import java.util.ArrayList;
10 import java.util.LinkedList;
11 import edu.rit.pj2.vbl.DoubleVbl;
12 import edu.rit.util.Random;
13
14 /**
15  * Class UndirectedGraph represents an undirected graph meaning that if
16  * there exists an edge connecting some vertex A to some vertex B, then
17  * that same edge connects vertex B to vertex A.
18  *
19  * @author Jimi Ford
20  * @version 2-15-2015
21  */
22 public class UndirectedGraph {
23
24     // private data members
25     private ArrayList<UndirectedEdge> edges;
26     private ArrayList<Vertex> vertices;
27     private int v;
28
29     // Prevent construction
30     private UndirectedGraph() {
31
32     }
33
34     /**
35      * Private constructor used internally by the static random graph
36      * method
37      * @param v the number of vertices in the graph
38      */
39     private UndirectedGraph(int v) {
40         this.v = v;
41         vertices = new ArrayList<Vertex>(v);
42         edges = new ArrayList<UndirectedEdge>();
43         for(int i = 0; i < v; i++) {
44             vertices.add(new Vertex(i));
45         }
46     }
47
48     /**
49      * Perform a BFS to get the distance from one vertex to another
50      *
51      * @param start the id of the start vertex
52      * @param goal the id of the goal vertex
53      * @return the minimum distance between the two vertices
54      */
55     private int BFS(int start, int goal) {
56         return BFS(vertices.get(start), vertices.get(goal));
57     }
58

```

```

59  /**
60  * Perform a BFS to get the distance from one vertex to another
61  *
62  * @param start the reference to the start vertex
63  * @param goal the reference to the goal vertex
64  * @return the minimum distance between the two vertices
65  */
66  private int BFS(Vertex start, Vertex goal) {
67      int distance = 0, verticesToProcess = 1, uniqueNeighbors = 0;
68      LinkedList<Vertex> queue = new LinkedList<Vertex>();
69      boolean[] visited = new boolean[v];
70      visited[start.n] = true;
71      Vertex current, t2;
72      queue.add(start);
73      while(!queue.isEmpty()) {
74          current = queue.removeFirst();
75          if(current.equals(goal)) {
76              return distance;
77          }
78          for(int i = 0; i < current.edgeCount(); i++) {
79              t2 = current.getEdges().get(i).other(current);
80              if(!visited[t2.n]) {
81                  visited[t2.n] = true;
82                  queue.add(t2);
83                  uniqueNeighbors++;
84              }
85          }
86          verticesToProcess--;
87          if(verticesToProcess <= 0) {
88              verticesToProcess = uniqueNeighbors;
89              uniqueNeighbors = 0;
90              distance++;
91          }
92      }
93      return 0;
94  }
95
96  /**
97  * Accumulate the distances of each pair of vertices into
98  * a "running total" to be averaged
99  *
100  *
101  * @param thrLocal the reference to the "running total"
102  * Prof. Alan Kaminsky's library handles averaging this
103  * accumulated value.
104  */
105  public void accumulateDistances(DoubleVbl.Mean thrLocal) {
106      for(int i = 0; i < v; i++) {
107          for(int j = i + 1; j < v; j++) {
108              thrLocal.accumulate(BFS(i, j));
109          }
110      }
111  }
112
113  /**
114  * Generate a random graph with a PRNG, a specified vertex count and
115  * an edge probability
116  *

```

UndirectedGraph.java

```

117  * @param prng Prof. Alan Kaminsky's Perfect Random Number Generator
118  * @param v number of vertices to use
119  * @param p edge probability between vertices
120  * @return the randomly generated graph
121  */
122  public static UndirectedGraph randomGraph(Random prng, int v, double p) {
123      UndirectedGraph g = new UndirectedGraph(v);
124      UndirectedEdge edge;
125      Vertex a, b;
126      int edgeCount = 0;
127      for (int i = 0; i < v; i++) {
128          for (int j = i + 1; j < v; j++) {
129              // connect edges
130              // always order it `i` then `j`
131              if(prng.nextDouble() <= p) {
132                  a = g.vertices.get(i);
133                  b = g.vertices.get(j);
134                  edge = new UndirectedEdge(edgeCount++, a, b);
135                  g.edges.add(edge);
136              }
137          }
138      }
139      return g;
140  }
141 }
142

```