```java
package edu.caltech.cs2.coloring;

import edu.caltech.cs2.datastructures.ChainingHashSet;
import edu.caltech.cs2.datastructures.MinFourHeap;
import edu.caltech.cs2.interfaces.IPriorityQueue;
import edu.caltech.cs2.interfaces.ISet;

public class DSatur {
    public static int color(NodeGraph g) {
        // Initialize the color mapping and priority queue
        int[] colorMapping = new int[g.numVertices()];
        IPriorityQueue<Integer> vertexQueue = new MinFourHeap<>();

        // Initialize the used colors set to keep track of already-colored neighbors
        ISet<Integer> usedColors = new ChainingHashSet<>();

        // Initialize the degree of saturation and uncolored neighbor count for each vertex
        int[] saturationDegree = new int[g.numVertices()];
        int[] uncoloredNeighbors = new int[g.numVertices()];

        // Initialize an array to keep track of which vertices are in the priority queue
        boolean[] inQueue = new boolean[g.numVertices()];

        // Add all vertices to the priority queue, ordered by decreasing degree of saturation
        for (int i = 0; i < g.numVertices(); i++) {
            int degree = g.neighbors(i).size();
            saturationDegree[i] = 0;
            uncoloredNeighbors[i] = degree;
            IPriorityQueue.PQElement<Integer> elem =
```

```
29 new IPriorityQueue.PQElement<>(i, -degree);
30             vertexQueue.enqueue(elem);
31             inQueue[i] = true;
32         }
33 
34         // Assign colors to each vertex in order of
   decreasing degree of saturation
35         // Assign colors to each vertex in order of
   decreasing degree of saturation
36         while (vertexQueue.size() > 0) {
37             IPriorityQueue.PQElement<Integer> vertex
    = vertexQueue.dequeue();
38             int v = vertex.data;
39             if (colorMapping[v] != 0) {
40                 continue;
41             }
42             System.out.println("Coloring vertex " + v
   );
43             ISet<Integer> neighborColors =
   getNeighborColors(g, v, colorMapping);
44             int color = 1;
45             boolean foundColor = false;
46             while (!foundColor) {
47                 foundColor = true;
48                 for (int c : neighborColors) {
49                     if (c == color) {
50                         color++;
51                         foundColor = false;
52                         break;
53                     }
54                 }
55             }
56             colorMapping[v] = color;
57             usedColors.add(color);
58             for (int neighbor : g.neighbors(v)) {
59                 if (colorMapping[neighbor] == 0) {
60                     neighborColors =
   getNeighborColors(g, neighbor, colorMapping);
61                     int saturation = neighborColors.
   size();
62                     saturationDegree[neighbor] =
```

```
62 saturation;
63                     uncoloredNeighbors[neighbor]--;
64                     IPriorityQueue.PQElement<Integer
   > neighborElem = new IPriorityQueue.PQElement<>(
   neighbor, -saturation);
65                     if (inQueue[neighbor]) {
66                         try {
67                             vertexQueue.decreaseKey(
   neighborElem);
68                         } catch (
   IllegalArgumentException e) {
69                             return -1;
70                         }
71                     } else if (saturation > 0) {
72                         vertexQueue.enqueue(
   neighborElem);
73                         inQueue[neighbor] = true;
74                     } else {
75                         inQueue[neighbor] = false;
76                     }
77                 }
78             }
79             inQueue[v] = false; // remove vertex
   from the queue
80         }
81
82         // Check if all vertices are colored
83         for (int i = 0; i < g.numVertices(); i++) {
84             if (colorMapping[i] == 0) {
85                 return -1; // Return a value to
   indicate an error occurred
86             }
87         }
88
89         // Check if there are any uncolored vertices
    remaining
90         for (int i = 0; i < g.numVertices(); i++) {
91             if (colorMapping[i] == 0) {
92                 return -1; // Return a value to
   indicate an error occurred
93             }
```

```
 94          }
 95          return usedColors.size();
 96      }
 97          // Helper method to count the number of
    unique colors of a vertex's neighbors
 98      private static ISet<Integer> getNeighborColors(
    NodeGraph g, int vertex, int[] colorMapping) {
 99          ISet<Integer> neighborColors = new
    ChainingHashSet<>();
100          for (int neighbor : g.neighbors(vertex)) {
101              int color = colorMapping[neighbor];
102              if (color > 0) {
103                  neighborColors.add(color);
104              }
105          }
106          return neighborColors;
107      }
108 }
109
```