```java
public class Entry implements Comparable<Entry>{
    private Vertex key;
    private double value;

    public Entry(Vertex key, double value){
        this.key = key;
        this.value = value;
    }
    public double getValue() {
        return value;
    }

    public Vertex getKey() {
        return key;
    }

    @Override
    public int compareTo(Entry o) {
        if (this.value < o.getValue()){
            return -1;
        }
        else if(this.value == o.getValue()){
            return 0;
        }
        else{
            return 1;
        }
    }
}

```

```java
public class Module implements Comparable<Module>{
    private int size;
    private int start;
    public Module(int size, int start){
        this.size = size;
        this.start = start;
    }

    public int getSize() {
        return size;
    }

    public int getStart() {
        return start;
    }

    @Override
    public int compareTo(Module o) {
        if (this.size < o.getSize()){
            return -1;
        }
        else if (this.size == o.getSize()){
            return 0;
        }
        else{
            return 1;
        }
    }
}

```

```java
import java.io.PrintWriter;

public class Vertex implements Comparable<Vertex> {
//so I can compare the vertices in a priority queue
    public String drugBankID;
    public String genericName;
    public String SMILES;
    public String url;
    public String drugGroups;
    public double score;
    public int value;
    public boolean wasVisited;
    public double dist;
    public Vertex path;
    private int module = -1;
    public Vertex(String drugBankID, String genericName, String SMILES, String url, String drugGroups, double score, boolean wasVisited, double dist){
        this.drugBankID = drugBankID;
        this.genericName = genericName;
        this.SMILES = SMILES;
        this.url = url;
        this.drugGroups = drugGroups;
        this.score = score;
        this.value = Integer.parseInt(drugBankID.split("DB")[1]);//this line will take just the number part of the drugID to compare
        this.wasVisited = wasVisited;
        this.dist = dist;
        this.path = null;
    }
    public void displayDrug(){
        System.out.print(drugBankID + ": ");
        System.out.print(wasVisited + ": ");
        System.out.print(dist + ": ");
        System.out.print(path + ": ");
        System.out.print(genericName + " ");
        System.out.print(SMILES + " ");
        System.out.print(url + " ");
        System.out.print(drugGroups + " | ");
```

```
36          System.out.println(score);
37      }
38      public void displayDrug(PrintWriter pw){
39          pw.print(drugBankID + ": ");
40          pw.print(dist + ": ");
41          pw.println();
42      }
43      public int getValue(){
44          return this.value;
45      }
46      public int getModule(){return this.module;}
47      public void setModule(int module) {this.module =
    module;}
48
49      @Override
50      public int compareTo(Vertex o) {
51          if (this.dist < o.dist){
52              return -1;
53          }
54          else if (this.dist == o.dist){
55              return 0;
56          }
57          else{
58              return 1;
59          }
60      }
61  }
```

```java
//When I first wrote this, only God and me understood how it works. Now only God does.
import java.awt.*;
import java.io.*;
import java.util.*;

public class DrugGraph {
    private static final double threshold = 0.7;
    public int moduleLen = 0;
    private int[] tempmodules;  //helper array so BFS can add a module to it without worrying about the amount of modules (initialized to capacity)
    public Module[] modules;
    private int[] tempBFTstarts; //same reason as tempmodules, needed a temporary array to have a fixed size before I find out the real size
    private int capacity;
    private Vertex[] vertices;
    private Vertex[] connectedVertices;
    private double[][] W; //weighted matrix created from (1-similarityMat[i][j])
    private int[][] A; //adjacency matrix based on results of the weighted matrix
    File f = new File("MSTPrimResult.tab");
    FileOutputStream output = new FileOutputStream(f);
    PrintWriter pw = new PrintWriter(output);
    public DrugGraph() throws FileNotFoundException {
        this(3000);
    }
    public DrugGraph(int capacity) throws FileNotFoundException {
        this.capacity = capacity;
        this.vertices = new Vertex[capacity];
        this.W = new double[capacity][capacity];
        this.A = new int[capacity][capacity];
        this.tempmodules = new int[capacity];
        this.tempBFTstarts = new int[capacity];
    }
    //Helper method to read data from a given matrix (in this case the source is hard coded into the
```

```
31 method but this can be modified)
32     //Creates an adjacency matrix from a list of
   drugs and a similarity matrix
33     //
34     //Param: Threshold - defaults to 0.7 and must be
   between 0 and 1
35     // is used to create an adjancency matrix from
   given similarity matrix where (1-sim[i][j] <=
   threshold)
36     //
37     //Result is stored in instance variables (
   vertices, W, A)
38     public void readData(double threshold) throws
   FileNotFoundException {
39         File f = new File("dockedApproved.tab");
40         String drugBankID;
41         String genericName;
42         String SMILES;
43         String url;
44         String drugGroups;
45         double score;
46         double[][] sim = new double[vertices.length][
   vertices.length];
47         Scanner in = new Scanner(f);
48         in.useDelimiter("\t|\n");
49         for (int j = 0; j<6; j++){     //ignores the
   header at the top of input
50             in.next();
51         }
52         int i = 0;
53         while (in.hasNext()) {
54             genericName = in.next();
55             SMILES = in.next();
56             drugBankID = in.next();
57             url = in.next();
58             drugGroups = in.next();
59             score = Double.parseDouble(in.next());
60             vertices[i] = new Vertex(drugBankID,
   genericName, SMILES, url, drugGroups, score, false,
   Double.POSITIVE_INFINITY);
61             i++;
```

```
62          }
63          f = new File("sim_mat.tab");
64          in = new Scanner(f);
65          in.useDelimiter("\t|\n");
66          for (int j = 0; j < vertices.length; j++) {
67              for (int k = 0; k < vertices.length; k
    ++) {
68                  sim[j][k] = Double.parseDouble(in.
    next());
69              }
70          }
71          for (int j = 0; j < sim.length; j++) {
72              for (int k = 0; k < sim.length; k++) {
73                  if (j!=k && 1-sim[j][k]<=threshold){
74                      W[j][k] = 1-sim[j][k];
75                  }
76                  else{
77                      W[j][k] = Double.
    POSITIVE_INFINITY;
78                  }
79              }
80          }
81          for (int j = 0; j < sim.length; j++) {
82              for (int k = 0; k < sim.length; k++) {
83                  if(W[j][k] != Double.
    POSITIVE_INFINITY){
84                      A[j][k] = 1;
85                  }
86                  else{
87                      A[j][k] = 0;
88                  }
89              }
90          }
91      }
92      //helper method to find adjacency array for one
    vertex
93      private int find(Vertex[] a, int id){
94          for (int i = 0; i < a.length; i++) {
95              if (a[i].value == id){
96                  return i;
97              }
```

```
 98          }
 99          return -1;
100      }
101      //breadth-first search starting at index i
102      public void BFS(int i){
103          int total = 0;
104          Queue queue = new LinkedList();
105          Vertex v = vertices[i];
106          v.wasVisited = true;
107          queue.add(v);
108          while (!queue.isEmpty()){
109              Vertex cur = (Vertex) queue.remove();
110              if (cur.getModule() == -1) {
111                  cur.setModule(i);
112              }
113              total++;
114              int curindex = find(vertices, cur.value
    );
115              for (int j = 0; j < vertices.length; j
    ++) {
116                  if (A[curindex][j] == 1 && vertices[
    j].wasVisited == false){    //for neighbours of u
117                      vertices[j].wasVisited = true;
118                      queue.add(vertices[j]);
119                  }
120              }
121          }
122          for (Vertex vertex:
123               vertices) {
124              vertex.wasVisited = false;
125          }
126          tempmodules[moduleLen] = (total);
127          tempBFTstarts[moduleLen] = i;
128          /*v.wasVisited = true;
129          Q.enqueue(v);
130          while !Q.isEmpty() // loop1
131          {
132              v = Q.dequeue();
133          while there is an unvisited vertex u
    adjacent to v // loop2
134          {
```

```
135             u.wasVisited = true;
136             Q.enqueue(u);
137         }*/
138     }
139     //sets the module id of all vertices along a
    path to the given id (starts from i with BFT)
140     private void setModuleId(int i, int id){
141         int total = 0;
142         Queue queue = new LinkedList();
143         Vertex v = vertices[i];
144         v.wasVisited = true;
145         queue.add(v);
146         while (!queue.isEmpty()){
147             Vertex cur = (Vertex) queue.remove();
148             cur.setModule(id);
149             total++;
150             int curindex = find(vertices, cur.value
    );
151             for (int j = 0; j < vertices.length; j
    ++) {
152                 if (A[curindex][j] == 1 && vertices[
    j].wasVisited == false){    //for neighbours of u
153                     vertices[j].wasVisited = true;
154                     queue.add(vertices[j]);
155                 }
156             }
157         }
158         for (Vertex vertex:
159                 vertices) {
160             vertex.wasVisited = false;
161         }
162     }
163     //returns the number of disconnected components
    in the drug graph
164     public int findModules(){
165         for (int i = 0; i < vertices.length; i++) {
166             if (vertices[i].getModule() == -1){
167                 BFS(i);
168                 moduleLen++;
169             }
170         }
```

```
171         modules = new Module[moduleLen];
172         for (int i = 0; i < moduleLen; i++) {
173             modules[i] = new Module(tempmodules[i],
    tempBFTstarts[i]);
174         }
175         int total = 0;
176         for (Vertex vertex:
177              vertices) {
178             if (vertex.getModule() == 2){
179                 total++;
180             }
181         }
182         Arrays.sort(modules);
183         Module sortedModules[] = new Module[
    moduleLen];
184         //Arrays.sort provides a sorted list in
    increasing order so we need to reverse the output
    array
185         int j = 0;
186         for (int i = moduleLen-1; i >= 0; i--) {
187             sortedModules[j] = modules[i];
188             setModuleId(modules[i].getStart(), j);
189             System.out.println(modules[i].getSize
    ());
190             j++;
191         }
192         this.modules = sortedModules;
193         return moduleLen;
194     }
195     public Vertex[] keepAModule(int moduleID){
196         //loop through all the modules and return an
     array with only the vertices with proper module ID
197         Vertex arr[] = new Vertex[modules[moduleID].
    getSize()];
198         int i = 0;
199         for (Vertex vertex:
200              vertices) {
201             if (vertex.getModule() == moduleID){
202                 arr[i] = vertex;
203                 i++;
204             }
```

```
205         }
206         return arr;
207     }
208     private void uw(int from, int to){
209         int starti = (find(connectedVertices, from
    ));
210         Queue queue = new LinkedList();
211         Vertex v = connectedVertices[starti];
212         v.dist = 0;
213         v.wasVisited = true;
214         queue.add(v);
215         while (!queue.isEmpty()){
216             Vertex cur = (Vertex) queue.remove();
217             cur.wasVisited = true;
218             int curindex = find(vertices, cur.value
    );
219             for (int j = 0; j < vertices.length; j
    ++) {
220                 if (A[curindex][j] == 1){   //for
    neighbours of u
221                     if (vertices[j].dist > cur.dist
     + 1){  //if distance can be improved by going
    through v to w
222                         vertices[j].dist = cur.dist
     + 1;
223                         vertices[j].path = cur;
224                         queue.add(vertices[j]);
225                     }
226                 }
227             }
228         }
229         //reset values of wasVisited
230         for (Vertex vertex:
231              connectedVertices) {
232             vertex.wasVisited = false;
233         }
234         int endi = (find(connectedVertices, to));
235         //now that we have the dists and paths set
    iterate backwards from the end to the start to see
    the path
236         String temparray[] = new String[(int)
```

```
236 connectedVertices[endi].dist+1];
237         Vertex curOutputVertex = connectedVertices[
    endi];
238         for (int i = (int)curOutputVertex.dist; i
     >= 0; i--) {
239             temparray[i] = curOutputVertex.
    drugBankID;
240             curOutputVertex = curOutputVertex.path;
241         }
242         //now we have an array of the path, iterate
    through the array, printing the results
243         for (int i = 0; i < temparray.length; i++) {
244             String output = (i != temparray.length-1
    )? temparray[i] + " - ":temparray[i]; //only prints
    line if there is a next drug
245             System.out.print(output);
246         }
247         System.out.println();
248     }
249     public void w(int from, int to){
250         int starti = (find(connectedVertices, from
    ));
251         Queue queue = new PriorityQueue();
252         Vertex v = connectedVertices[starti];
253         v.dist = 0;
254         queue.add(v);
255         while (!queue.isEmpty()){
256             Vertex cur = (Vertex) queue.remove();
257             if (!cur.wasVisited) {
258                 cur.wasVisited = true;
259                 int curindex = find(vertices, cur.
    value);
260                 for (int j = 0; j < vertices.length
    ; j++) {
261                     if (W[curindex][j] != Double.
    POSITIVE_INFINITY) {    //for neighbours of u
262                         if (vertices[j].dist > cur.
    dist + W[curindex][j]) {    //if distance can be
    improved by going through v to w
263                             vertices[j].dist = cur.
    dist + W[curindex][j];
```

```
264                                 vertices[j].path = cur;
265                                 queue.add(vertices[j]);
266                             }
267                         }
268                     }
269                 }
270             }
271             //reset values of wasVisited
272             for (Vertex vertex:
273                     connectedVertices) {
274                 vertex.wasVisited = false;
275             }
276             int endi = (find(connectedVertices, to));
277             int pathLength = 0;
278             //we need to figure out how many vertices it
     takes to get from S to F
279             Vertex path = connectedVertices[endi]; //
    start at the end and go back until it can't go back
    anymore
280             while (path != null){
281                 path = path.path; //this may be the
    worst line of code I've ever seen
282                 pathLength++;
283             }
284             //make an array that has the path from S to
    F
285             String temparray[] = new String[pathLength];
286             Vertex curVertex = connectedVertices[endi];
287             for (int i = pathLength-1; i >= 0; i--) {
288                 temparray[i] = curVertex.drugBankID;
289                 curVertex = curVertex.path;
290             }
291             //now we have an array of the path, iterate
    through the array, printing the results
292             for (int i = 0; i < temparray.length; i++) {
293                 String output = (i != temparray.length-1
    )? temparray[i] + " - ":temparray[i]; //only prints
    line if there is a next drug
294                 System.out.print(output);
295             }
296             System.out.println();
```

```
297     }
298     public void findShortestPath(String fromDrug,
    String toDrug, String method){
299         if (method == "unweighted"){
300             uw(Integer.parseInt(fromDrug.split("DB"
    )[1]), Integer.parseInt(toDrug.split("DB")[1]));
301         }
302         else{
303             w(Integer.parseInt(fromDrug.split("DB")[
    1]), Integer.parseInt(toDrug.split("DB")[1]));
304         }
305         for (Vertex vertex:
306              connectedVertices) {
307             vertex.path = null;
308             vertex.dist = Double.POSITIVE_INFINITY;
309             vertex.wasVisited = false;
310         }
311     }
312     public double MSTPrim(){
313         int i = 0;
314         Vertex lastRemoved = null;
315         Queue queue = new PriorityQueue();
316         Vertex v = connectedVertices[i];
317         v.dist = 0;
318         queue.add(new Entry(v, v.dist));
319         while (!queue.isEmpty()){
320             Vertex cur = ((Entry) queue.remove()).
    getKey();
321             if (!cur.wasVisited) {
322                 cur.wasVisited = true;
323                 int curindex = find(vertices, cur.
    value);
324                 for (int j = 0; j < vertices.length
    ; j++) {
325                     if (W[curindex][j] != Double.
    POSITIVE_INFINITY) {    //for neighbours of u
326                         if ((vertices[j].dist > W[
    curindex][j]) && (!vertices[j].wasVisited)) {    //
    if distance can be improved by going through v to w
327                             vertices[j].dist = W[
    curindex][j];
```

```
328                                 vertices[j].path = cur;
329                                 queue.add(new Entry(
    vertices[j], vertices[j].dist));
330                             }
331                         }
332                     }
333                 }
334             }
335             double total = 0;
336             for (Vertex ver: connectedVertices) {
337                 ver.displayDrug(pw);
338                 total += ver.dist;
339             }
340             return total;
341         }
342         public static void main(String[] args) throws
    FileNotFoundException {
343             DrugGraph d = new DrugGraph(1932);
344             d.readData(threshold);
345             System.out.println(d.findModules() + "
    modules found.");
346             d.connectedVertices = d.keepAModule(0);
347             d.findShortestPath("DB01050", "DB00316", "
    unweighted");
348             d.findShortestPath("DB01050", "DB00316", "
    weighted");
349             System.out.print(Math.round(d.MSTPrim() *
    100) / 100.0); //prints the total weight of the MST
    rounded to 2 decimal places
350         }
351 }
352
```