Problem 1. (*Graph Properties*) The *eccentricity* of a vertex v is the length of the shortest path from that vertex to the furthest vertex from v. The *diameter* of a graph is the maximum eccentricity of any vertex. The *radius* of a graph is the smallest eccentricity of any vertex. A *center* is a vertex whose eccentricity is the radius. Implement a data type `GraphProperties` that supports the following API to calculate the aforementioned graph properties:

method	description
`GraphProperties(Graph G)`	calculate graph properties for the undirected graph G
`int eccentricity(int v)`	eccentricity of vertex v
`int diameter()`	diameter of G
`int radius()`	radius of G
`Iterable<Integer> centers()`	centers of G

```
$ java GraphProperties data/tinyG.txt
Diameter = 7
Radius   = 4
Centers  = 0 4 6
```

Problem 2. (*Degrees*) The *indegree* of a vertex in a digraph is the number of directed edges that point to that vertex. The *outdegree* of a vertex in a digraph is the number of directed edges that emanate from that vertex. No vertex is reachable from a vertex of outdegree 0, which is called a *sink*; a vertex of indegree 0, which is called a *source*, is not reachable from any other vertex. A digraph where self-loops are allowed and every vertex has outdegree 1 is called a map (a function from the set of integers from 0 to $V - 1$ onto itself). Implement a data type `Degrees` that implements the following API to calculate the aforementioned properties of a digraph:

method	description
`Degrees(Digraph G)`	construct a `Degrees` object from a digraph G
`Iterable<Integer> sources()`	sources of G
`Iterable<Integer> sinks()`	sinks of G
`boolean isMap()`	is G a map?

```
$ java Degrees data/tinyDG.txt
Sources = 7
Sinks   = 1
Is Map  = false
```

Problem 3. (*Euclidean Edge*) Implement a comparable data type `EuclideanEdge` that represents an edge in an undirected graph and whose end points are points in the plane, represented as `Point2D` objects. The data type must support the following API:

method	description
`EuclideanEdge(Point2D v, Point2D w)`	construct an Euclidean edge given the end points
`Point2D either()`	either endpoint of this edge
`Point2D other(Point2D vertex)`	endpoint of this edge that is different from the given vertex
`double weight()`	weight of this edge, ie, the Euclidean distance between the endpoints
`String toString()`	a string representation of this edge
`int compareTo(EuclideanEdge that)`	compare two edges by their weights

```
$ java EuclideanEdge 0 0 1 1 1 0 0 1
(0.0, 0.0)-(1.0, 1.0) 1.41421
(1.0, 0.0)-(0.0, 1.0) 1.41421
0
```

Problem 4. (*Euclidean Edge-weighted Graph*) Implement a data type `EuclideanEdgeWeightedGraph` for representing Euclidean edge-weighted graphs whose vertices are represented as `Point2D` objects and edges are represented as `EuclideanEdge` objects. The data type must support the following API:

method	description
`EuclideanEdgeWeightedGraph(In in)`	initialize an empty Euclidean edge-weighted graph from an input stream
`int V()`	number of vertices in the graph
`int E()`	number of edges in the graph
`void addEdge(EuclideanEdge e)`	add an undirected edge to the graph
`Iterable<EuclideanEdge> adj(Point2D v)`	edges incident on vertex v
`Iterable<EuclideanEdge> edges()`	all the edges in the graph
`String toString()`	a string representation of the graph

```
$ java EuclideanEdgeWeightedGraph data/tinyEG.txt
(-2.99428874799, -4.81382481949)-(-3.50590184636, 2.70830491327) 7.53951
(1.79035963543, -4.7292107303)-(-3.50590184636, 2.70830491327) 9.13055
(-3.50590184636, 2.70830491327)-(3.88256645668, -1.23291312479) 8.37393
(-3.50590184636, 2.70830491327)-(2.95034889345, 4.14320098075) 6.61378
(-3.50590184636, 2.70830491327)-(-3.06299778289, -1.37765565012) 4.10990
(1.79035963543, -4.7292107303)-(2.95034889345, 4.14320098075) 8.94792
(3.88256645668, -1.23291312479)-(2.95034889345, 4.14320098075) 5.45634
(-0.972219535182, 0.144692907976)-(2.95034889345, 4.14320098075) 5.60130
(-3.24240294366, 3.94050921397)-(2.95034889345, 4.14320098075) 6.19607
(-2.99428874799, -4.81382481949)-(-3.24240294366, 3.94050921397) 8.75785
(-2.99428874799, -4.81382481949)-(-0.609246278064, -2.3300787821) 3.44346
(-2.99428874799, -4.81382481949)-(-0.972219535182, 0.144692907976) 5.35497
(-0.972219535182, 0.144692907976)-(-3.06299778289, -1.37765565012) 2.58629
(1.79035963543, -4.7292107303)-(-3.24240294366, 3.94050921397) 10.02461
(1.79035963543, -4.7292107303)-(-0.609246278064, -2.3300787821) 3.39322
(-2.99428874799, -4.81382481949)-(-1.34570880317, -2.97279434591) 2.47128
(-1.34570880317, -2.97279434591)-(3.88256645668, -1.23291312479) 5.51018
(-1.34570880317, -2.97279434591)-(-0.972219535182, 0.144692907976) 3.13978
(-1.34570880317, -2.97279434591)-(-3.06299778289, -1.37765565012) 2.34383
(3.88256645668, -1.23291312479)-(-3.06299778289, -1.37765565012) 6.94707
(-0.609246278064, -2.3300787821)-(-3.06299778289, -1.37765565012) 2.63211
```

Problem 5. (*Euclidean Kruskal MST*) Implement a data type `EuclideanKruskalMST` that uses Kruskal's algorithm to compute a minimum spanning tree (or forest) of an Euclidean edge-weighted graph. The data type must support the following API:

method	description
`EuclideanKruskalMST(EuclideanEdgeWeightedGraph G)`	compute a minimum spanning tree (or forest) of an Euclidean edge-weighted graph
`Iterable<EuclideanEdge> edges()`	edges in a minimum spanning tree (or forest)
`double weight()`	sum of the edge weights in a minimum spanning tree (or forest)

```
$ java EuclideanKruskalMST data/tinyEG.txt
(-1.34570880317, -2.97279434591)-(-3.06299778289, -1.37765565012) 2.34383
(-2.99428874799, -4.81382481949)-(-1.34570880317, -2.97279434591) 2.47128
(-0.972219535182, 0.144692907976)-(-3.06299778289, -1.37765565012) 2.58629
(-0.609246278064, -2.3300787821)-(-3.06299778289, -1.37765565012) 2.63211
(1.79035963543, -4.7292107303)-(-0.609246278064, -2.3300787821) 3.39322
(-3.50590184636, 2.70830491327)-(-3.06299778289, -1.37765565012) 4.10990
(3.88256645668, -1.23291312479)-(2.95034889345, 4.14320098075) 5.45634
(-1.34570880317, -2.97279434591)-(3.88256645668, -1.23291312479) 5.51018
(-3.24240294366, 3.94050921397)-(2.95034889345, 4.14320098075) 6.19607
34.69921
```

Files to Submit

1. `GraphProperties.java`
2. `Degrees.java`
3. `EuclideanEdge.java`
4. `EuclideanEdgeWeightedGraph.java`
5. `EuclideanKruskalMST.java`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

  ```
  $ python3 run_tests.py -v [<problems>]
  ```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test, separated by spaces; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

  ```
  $ check_style <program>
  ```

  where `<program>` is the `.java` file whose style you want to check.