Problem 1. (*Graph Properties*) Consider an undirected graph G with V vertices and E edges.

- The *degree distribution* of G is a function mapping each degree value in G to the number of vertices with that value.
- The *average degree* of G is $\frac{2E}{V}$.
- The *average path length* of G is the average length of all the paths in G.
- The local clustering coefficient C_i for a vertex v_i is the number of edges that actually exist between the vertices in its neighbourhood divided by the number of edges that could possibly exist between them, which is $\frac{V(V-1)}{2}$. The *global clustering coefficient* of G is $\frac{1}{V}\sum_i^V C_i$.

Implement a data type called `GraphProperties` with the following API to compute the aforementioned graph properties:

GraphProperties

`GraphProperties(Graph G)`	computes graph properties for the undirected graph `G`
`RedBlackBinarySearchTreeST<Integer, Integer> degreeDistribution()`	returns the degree distribution of the graph
`double averageDegree()`	returns the average degree of the graph
`double averagePathLength()`	returns the average path length of the graph
`double clusteringCoefficient()`	returns the global clustering coefficient of the graph

~/workspace/exercise6

```
$ java GraphProperties data/tinyG.txt
Degree distribution:
  1: 3
  2: 4
  3: 5
  4: 1
Average degree         =   2.308
Average path length    =   3.090
Clustering coefficient =   0.256
```

GraphProperties.java

```java
import dsa.BFSPaths;
import dsa.Graph;
import dsa.RedBlackBinarySearchTreeST;
import stdlib.In;
import stdlib.StdOut;

public class GraphProperties {
    private RedBlackBinarySearchTreeST<Integer, Integer> st; // degree -> frequency
    private double avgDegree;                                // average degree of the graph
    private double avgPathLength;                            // average path length of the graph
    private double clusteringCoefficient;                    // clustering coefficient of the graph

    // Computes graph properties for the undirected graph G.
    public GraphProperties(Graph G) {
        ...
    }

    // Returns the degree distribution of the graph (a symbol table mapping each degree value to
    // the number of vertices with that value).
    public RedBlackBinarySearchTreeST<Integer, Integer> degreeDistribution() {
        ...
    }

    // Returns the average degree of the graph.
    public double averageDegree() {
        ...
    }

    // Returns the average path length of the graph.
    public double averagePathLength() {
        ...
    }

    // Returns the global clustering coefficient of the graph.
    public double clusteringCoefficient() {
        ...
    }
```

```
    // Returns true if G has an edge between vertices v and w, and false otherwise.
    private static boolean hasEdge(Graph G, int v, int w) {
        for (int u : G.adj(v)) {
            if (u == w) {
                return true;
            }
        }
        return false;
    }

    // Unit tests the data type. [DO NOT EDIT]
    public static void main(String[] args) {
        In in = new In(args[0]);
        Graph G = new Graph(in);
        GraphProperties gp = new GraphProperties(G);
        RedBlackBinarySearchTreeST<Integer, Integer> st = gp.degreeDistribution();
        StdOut.println("Degree distribution:");
        for (int degree : st.keys()) {
            StdOut.println("  " + degree + ": " + st.get(degree));
        }
        StdOut.printf("Average degree         = %7.3f\n", gp.averageDegree());
        StdOut.printf("Average path length    = %7.3f\n", gp.averagePathLength());
        StdOut.printf("Clustering coefficient = %7.3f\n", gp.clusteringCoefficient());
    }
}
```

Problem 2. (*DiGraph Properties*) Consider a digraph G with V vertices.

- G is a *directed acyclic graph (DAG)* if it does not contain any directed cycles.
- G is a *map* if every vertex has an outdegree of 1.
- A vertex v is a *source* if its indegree is 0.
- A vertex v is a *sink* if its outdegree is 0.

Implement a data type called `DiGraphProperties` with the following API to compute the aforementioned digraph properties:

DiGraphProperties	
`DiGraphProperties(DiGraph G)`	computes graph properties for the digraph `G`
`boolean isDAG()`	returns `true` if the digraph is a DAG, and `false` otherwise
`boolean isMap()`	returns `true` if the digraph is a map, and `false` otherwise
`Iterable<Integer> sources()`	returns all the sources in the digraph
`Iterable<Integer> sinks()`	returns all the sinks in the digraph

~/workspace/exercise6

```
$ java DiGraphProperties data/tinyDG.txt
Sources: 7
Sinks: 1
Is DAG? false
Is Map? false
```

DiGraphProperties.java

```
import dsa.DiCycle;
import dsa.DiGraph;
import dsa.LinkedBag;
import stdlib.In;
import stdlib.StdOut;

public class DiGraphProperties {
    private boolean isDAG;               // is the digraph a DAG?
    private boolean isMap;               // is the digraph a map?
    private LinkedBag<Integer> sources; // the sources in the digraph
    private LinkedBag<Integer> sinks;   // the sinks in the digraph

    // Computes graph properties for the digraph G.
    public DiGraphProperties(DiGraph G) {
```

```java
        ...
    }

    // Returns true if the digraph is a directed acyclic graph (DAG), and false otherwise.
    public boolean isDAG() {
        ...
    }

    // Returns true if the digraph is a map, and false otherwise.
    public boolean isMap() {
        ...
    }

    // Returns all the sources (ie, vertices without any incoming edges) in the digraph.
    public Iterable<Integer> sources() {
        ...
    }

    // Returns all the sinks (ie, vertices without any outgoing edges) in the digraph.
    public Iterable<Integer> sinks() {
        ...
    }

    // Unit tests the data type. [DO NOT EDIT]
    public static void main(String[] args) {
        In in = new In(args[0]);
        DiGraph G = new DiGraph(in);
        DiGraphProperties gp = new DiGraphProperties(G);
        StdOut.print("Sources: ");
        for (int v : gp.sources()) {
            StdOut.print(v + " ");
        }
        StdOut.println();
        StdOut.print("Sinks: ");
        for (int v : gp.sinks()) {
            StdOut.print(v + " ");
        }
        StdOut.println();
        StdOut.println("Is DAG? " + gp.isDAG());
        StdOut.println("Is Map? " + gp.isMap());
    }
}
```

Files to Submit

1. `GraphProperties.java`
2. `DiGraphProperties.java`

Before you submit your files, make sure:

- You do not use concepts outside of what has been taught in class.
- Your code is adequately commented, follows good programming principles, and meets any specific requirements such as corner cases and running times.