22C:21: Computer Science II: Data Structures
Project 1: Due October 17, 2005

A *wireless network* consists of a collection of devices that are equipped with radio transmitters and receivers. We shall refer to these devices as *nodes*. Pairs of these nodes can communicate with each other through radio signals, provided they are within each other's transmission ranges. A pair of nodes that are physically far away from each other may be able use other nodes as "intermediaries" through which messages can be routed. Various models of wireless networks have been proposed, so that wireless networks and protocols on these networks can be mathematically analyzed. A simple model for wireless networks is the *disk graph*. Each vertex in the disk graph is a point in the plane and associated with each vertex u is a positive real number $t(u)$, that denotes the transmission range of u. For any pair of vertices u and v, there is a *directed* edge (u, v) in the disk graph provided the distance between u and v, denoted $|uv|$, is no greater than $t(u)$. Thus the edge (u, v) represents the fact that v is in the transmission range of u and therefore u can send messages directly to v. Note that while v may be in u's transmission range, u may not be in v's transmission range, implying that the disk graph may contain edge (u, v), but not edge (v, u). If the transmission ranges for all the vertices are equal then we call the graph a *unit disk graph* (in short, a UDG). Note that all edges in a UDG go both ways, that is, (u, v) is an edge if and only if (v, u) is an edge. Therefore, a UDG can be thought of as an undirected graph. See Figure 1 for an illustration of a UDG.

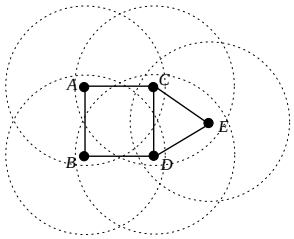

Figure 1: A unit disk graph with 5 vertices. The edges are defined by the disks shown in the figure. For example, the disk of which B is the center contains A and D as well and this implies the edges $\{A, B\}$ and $\{B, D\}$.

Disk graphs and UDGs are simple models of wireless networks and there is a large body of recent research on wireless networks that uses these models. Also, many researchers use these models to experimentally simulate various protocols designed for wireless networks. In a series of 3 projects for this course, you will have to simulate some recently proposed protocols for wireless networks. Project 1 has 3 parts: (I) UDG generation, (II) topology control, and (III) routing.

1 The `myGraph` class

At the core of your implementation is the `myGraph` class, more or less as described in my lectures. For this project, we will only be dealing with UDGs and not with arbitrary disk graphs, and therefore it is sufficient to implement undirected graphs. You should use the *adjacency matrix* representation that we discussed in class, but with two modifications, described below.

1. We noticed that an adjacency matrix that represents an undirected graph is *symmetric*, that is, the entry in row i, column j is the same as the entry in row j, column i. This means that it is enough to only store about half of the matrix. In particular, I want you to only store the portion of the matrix above the main diagonal. In other words, for row i, you will only store columns $i + 1, i + 2, \ldots,$.

2. Very often there are positive real number "weights" associated with edges. These may, for example, represent distances between endpoints of edges. To store this information, you should use the adjacency matrix, except that instead of using a `boolean` matrix, you should use a matrix of real numbers. This will allow you to store any real number weight associated with each edge. Assuming that each edge $\{u, v\}$ comes with a positive real number weight $w(u, v)$, we store the quantity $w(u, v)$ in row i and column j, where i is the index of vertex u, j is the index of vertex v, and $i < j$. Note that if u and v are not connected by an edge, then the entry in row i, column j would be 0.

At the minimum, you should implement constructors for the `myGraph` class and the public methods `addVertex`, `addEdge`, `deleteVertex`, `deleteEdge`, `getNeighbors`, `getNumberVertices`, and `getNumberEdges`. You are welcome to implement any other methods that provide additional functionality for the `myGraph` class.

2 The `wirelessNetwork` class

You are required to implement a class called `wirelessNetwork` that provides methods for generating wireless networks and for simulating protocols on these networks. The `wirelessNetwork` class would use the `myGraph` class extensively. For this project, the `wirelessNetwork` class is required to contain the following three methods.

- A constructor for generating a wireless network.
- A method for simulating a protocol for *topology control*.
- A method to perform *compass routing* on the network.

More details are provided below.

Generating wireless networks. Implement a constructor of the `wirelessNetwork` class that takes two parameters: a `float` called `size` and an `int` called `n`. The function should generate a UDG obtained by distributing `n` points, uniformly at random, on a square of dimensions `size` $\times$ `size` square units. For example, if `n` were 1000 and `size` were 10, then the constructor would distribute 1000 points uniformly at random on a 10×10 square. By "uniformly at random" I mean that the likelihood of a point falling anywhere in the square is the same. You should use the class `Random` to distribute points in this way. Each pair of points at distance at most 1 unit are connected by an edge. This means that each point represents a wireless device with a transmission range of 1 unit.

In addition to the above constructor, also provide a "default" constructor that takes no parameters, and generates the UDG obtained by distributing 500 points, uniformly at random, on a 10×10 square.

Topology control. In wired networks, more the connections between nodes, the better it is. However, in wireless networks this is not the case. If lots of nodes are physically close together and start attempting to communicate with each other, there is interference between the signals and nodes end up receiving garbled messages. To reduce interference, each node in a wireless network chooses a subset of the nodes in its transmission range and decides just to communicate with these. The protocol that each node uses to choose a subset of nodes to communicate with, is called a *topology control* protocol. At the end of topology control, we end up with a network that has all the nodes in the original network, but far fewer edges.

To be more precise let $G = (V, E)$ be the UDG representing a wireless network. Each vertex $u \in V$ has a set of neighbors, denoted $N(u)$. During topology control, each vertex u

chooses a subset $N'(u) \subseteq N(u)$ of neighbors that it will communicate with. This defines a new graph $H = (V, E')$, where $E' = \{(u, v) \mid v \in N'(u)\}$. Most topology control protocols ensure that if $v \in N'(u)$ then $u \in N'(v)$ as well. This means that $(u, v) \in E'$ if and only if $(v, u) \in E'$. Thus we can take the graph H to be an undirected graph.

I would like you to implement a simple topology control protocol called *XTC* (R. Wattenhofer and A. Zollinger, "XTC: A practical topology control algorithm for ad-hoc networks, *WMAN 2004*) given below.

The XTC protocol.
Input: A UDG $G = (V, E)$.

```
for each vertex u ∈ V do
        temp := N(u)
        for each neighbor v ∈ N(u) do
                if u and v have a common neighbor w such that
                |uw| < |uv| and |vw| < |uv| then
                        temp := temp − {v}
        N'(u) := temp
```

Output the graph $H = (V, E')$ where $E' = \{(u, v) \mid v \in N(u)\}$.

As you can see, the algorithm uses a simple rule to determine whether an edge $\{u, v\}$ should be kept or dropped. The authors of the XTC paper prove that after topology control is run on G, the output graph H has a number of properties very useful for routing. Some of these are:

- if G is connected, then H is connected.
- every vertex in H has at most 6 neighbors.
- H is *planar*, that is, no two edges in H cross

Compass routing. A wired network typically consists of nodes with considerable computational power and these devices can compute and store large *routing tables* that facilitate routing Suppose a node receives a message m intended for a destination t. The node immediately looks up its routing table to see which neighbor to forward m to, so that it will get to t. A wireless network typically consists of nodes that have limited computational power and limited amount of memory. Therefore, doing routing with the help of routing tables is not a feasible option for wireless networks. In a wireless network, when a node receives a message m intended for a destination t, it needs to decide which neighbor to forward m to without the help of a routing table. One "greedy" approach to this problem attempts to use geometric information and do *compass routing* (E. Kranakis, H. Singh and J. Urrutia, "Compass routing on geometric networks", *Proceedings of the 11th Canadian Conference on Computational Geometry, 1999.*)

Here is a description of how compass routing works. Suppose a node x receives a message m intended for destination t. Further suppose that x knows its location, the location of t, and the location of its neighbors in the plane. Then x sends m to a neighbor v that minimizes the angle $\angle txv$. The intuition is that we always want to travel in a direction that as is close to the direction $\vec{xt}$ as possible. See Figure 2 for an illustration of compass routing. In the paper on compass routing mentioned above, it is also shown that compass routing does not always work and there instances for which a message may forever cycle among a set of nodes without reaching the destination.

You should simulate compass routing by implementing a method in the `wirelessNetwork` class that takes as parameters a source vertex s and a destination vertex t. The method

Figure 2: The node x, on receiving a message m, intended for the destination t, forwards m to a neighbor v that makes the smallest angle with $\vec{xt}$.

should either return a path from s to t that was discovered by compass routing or should return an indication that no path from s to t could be found. There is a simple recursive implementation of compass routing based on the following idea: if the current vertex x equals t, then we are done; otherwise find a neighbor v of x that makes the smallest angle with the direction $\vec{xt}$ and do compass routing with source v and destination t.

To summarize, the `wirelessNetwork` class should at least contain the following methods.

```
wirelessNetwork();
wirelessNetwork(float size, int n);

wirelessNetwork topologyControl();

int[] compassRouting(int s, int t);
```

The function headers given above should be treated as rough guides; if there are other parameters you feel the need to pass or if you want to pass parameters differently, feel free to do so. You may need to implement additional "helper" functions in the `wirelessNetwork` class.

3 Experiments

Using your implementation of the `wirelessNetwork` class, you are required to perform the following experiments.

1. Generate 10 wireless networks modeled as UDGs, by distributing

$$n = 500, 550, 600, 650, 700, 750, 800, 850, 900, 950$$

points on a 10×10 grid. For each network, report the average degree and the maximum degree. Recall that the degree of a vertex is equal to the number of neighbors it has. Plot these, showing how they increase with respect to n. Run topology control on each of these networks to produce 10 new and much sparser networks. Report and plot the average and maximum degree of these 10 networks as well. Based on your data and plots, write 2-3 sentences commenting on whether performing topology control has resulted in a much sparser graph or not.

2. Generate a wireless network G with 1000 points distributed on a 10×10 square. Then repeat the following 10 times. Uniformly at random, pick a vertex and designate it the

source s and again uniformly at random, pick a vertex and designate it the destination t. Run compass routing on G, with source s and destination t. Report on whether compass routing was able to find the destination or not and if it was, report on the length of the path from s to t that was discovered. Present your results in a tabular form.

3. Start with the same wireless network G generated for Experiment (2) and run topology control on it to get a sparser network H. Run Experiment (2) again, but use the network H this time. Tabulate your results as for Experiment (2). Write 2-3 sentences commenting on how the path lengths generated in Experiment (3), compare with path lengths generated in Experiment (2). Try to explain your observations.

Place the code for your experiments in a single file called `experiments.java`. The code in the file `experiments.java` makes extensive calls to methods in the `wirelessNetwork` class and possibly to methods in the `myGraph` class as well.

4 Conclusion

You need to submit three `java` code files:

(i) `myGraph.java`, containing the `myGraph` class,

(ii) `wirelessNetwork.java` containing the `wirelessNetwork` class, and

(iii) `experiments.java`, containing the code for the experiments.

In addition, you need to submit plots, tables, and accompanying text describing the results of your experiments. Put all of this in a single file called `results`. We will be able to handle the `results` file in `pdf` format of `MSWord` format. Please submit exactly these files and no others. Instructions on how to do an electronic submission of all your files will appear on the course page 2-3 days before due date.
