PJ-04
Network Builder
Due : 11/15 by 11 : 59 pm

Goals

- Implementation of the graph data structure and understanding its use cases
- Implementing Kruskal's minimum spanning tree algorithm and understanding its application to clustering
- Implementation of graph traversals and application to find connected components

Academic Dishonesty

This is an individual project, to be completed on your own. It is considered dishonest either to read someone else's solution or to provide a classmate with a copy of your work. Do not make the mistake of thinking that superficial changes in a program (such as altering comments, changing variable names, or interchanging statements) can be used to avoid detection. If you cannot do the work yourself, it is extremely unlikely that you can succeed in disguising someone else's work. We are adamant that cheating in any form is not tolerated. Even the most trivial assignment is better not done than if you cheat to complete it.

Problem

In this project, you will be building a small-scale version of the internet using the tools and techniques you have learned in class so far. Specifically, you will be given computers, which you will group into clusters according to latency of transmissions, create a cluster-network, and implement an algorithm to handle efficient inter-cluster routing of messages. This project is divided into four milestones

1. Milestone 1: Building network as a single cluster
2. Milestone 2: k-Clustering using Kruskal's algorithm
3. Milestone 3: Building network with multiple clusters
4. Milestone 4: Inter-cluster communication

Each of theses parts is described in further detail in the sections below.

Milestone 1: Building network as single cluster

In this part, you will have to read in connections between computers, and create a graph using these connections in the method `buildNetwork()`. The input will be of the form

m
$u_1 \;\; v_1 \;\; l_1$
$u_2 \;\; v_2 \;\; l_2$
$\vdots$
$u_m \;\; v_m \;\; l_m$

Here, the first line m is the number of edges in the graph. The m lines that follow correspond to the m edges, where the ith edge u_i v_i l_i may be interpreted as an edge between nodes u_i and v_i with latency l_i. The edges are undirected and thus the graph they build is an undirected graph. as Any u_i and v_i represent IP addresses of computers being referenced with l_i as the latency of the connection between them. Note that a computer can be connected to multiple other computers and hence the IP address of that computer may repeat in the input. We need a way to keep track of which IP addresses we have already encountered and find a way to reference a certain computer within the graph we plan to build. Thus, for every edge, you must do the following:

- Read in the ith edge
- Check whether the nodes u_i and v_i have already been encountered
  - If u_i hasn't been encountered yet, assign a unique index to u_i which is how we will reference it in the graph we build.
  - Do the same procedure for node v_i
  - The unique index assigned to any node needs to be sequential. The first unencountered node is assigned a unique index of 0, with the next one being assigned 1 and so on.
  - Note that this step is crucial during testing as it gives an ordering to the nodes against which we will run test cases.
- Add the edge (u_i, v_i) to the graph with latency l_i. Note that here you must use the mapped unique indices instead of the actual computer IP addresses given as part of the input.
- Think of the data structure(s) you can use to implement this functionality.

Running the above steps will result in populating the data structure **computerConnections** with the connections between computers where each connection is represented as:
$\{$unique index of u, unique index of v, latency between u and $v\}$. Examples are given below:

Sample input:
```
6
47 53 40
40 26 90
53 13 5
40 53 25
26 47 23
47 40 48
```

Unique Index Assingment:
$47 \to 0$
$53 \to 1$
$40 \to 2$
$26 \to 3$
$13 \to 4$

`computerConnections` data structure:
`computerConnections.get(0)` $\to \{0, 1, 40\}$
`computerConnections.get(1)` $\to \{2, 3, 90\}$
`computerConnections.get(2)` $\to \{1, 4, 5\}$
`computerConnections.get(3)` $\to \{2, 1, 25\}$
`computerConnections.get(4)` $\to \{3, 0, 23\}$
`computerConnections.get(5)` $\to \{0, 2, 48\}$

Notes

- Add edges in the order given to you so that your `computerConnections` sequence matches that in the tests.

Milestone 2: k-Clustering using Kruskal's Algorithm

In this part, we will use **computerConnections** (and other data structures) constructed in part 1 and create a network of clusters (multiple smaller graphs) which is representative of how a simple internet would look like. Each cluster can be interpreted as a collection of computers we can entirely replace with a **router**. This will be implmeneted in the method `buildCluster(int k)`

This simplifies our internet form being a large and complex collection of computers with latencies to a much smaller collection of routers with inter-router latencies. When we want to send a message from one computer to another, we send it from one computer to its router, from that router to the other computer's router, and finally to the computer within that router network. Using the concept of clustering we can transform our large and complex computer newtork to a more manageable **network of networks**.

There are multiple clustering algorithms that one may use with multiple different distance metrics. For our purposes, we will use the latency as our distance metric. In particular, we want intra-cluster (within the same cluster/graph) communication to have minimum latency. We will achieve this as follows:

- Suppose we wanted k clusters. This will be provided to you as input.
- Run a variation of **Kruskal's Algorithm** on the collection of edges (**computerConnections**) from part 1 until there are k connected components. This should populate the data structure **computerGraph** which is an adjacency list containing the newly formed graph (essentially a **forest of trees**)[a] from the algorithm. You are given a **UnionFind** class to help with this implementation.
- Run a traversal algorithm on **computerGraph** produced from the previous step in order to actually find the connected components produced by it. Similar to part 1, we need to assign unique indexes to each detected component and create a mapping between the two. We do this by running a traversal on `computerGraph` and assign indexes in sequential order. The first detected connected component from the traversal has index 0, with the next one having index 1 and so on. We will call this mapping CI (Cluster to Index) which we will use in the next milestone.
  Note: Always start your traversal at node with unique index 0 in your computerGraph.
- Each of these connected components represents a cluster of computers. You need to populate the **cluster** data structure with these components. Any **cluster.get(i)** represents a list of all the nodes within that cluster.
- Every cluster needs an IP address so that we may be able to reference each cluster. For the purposes of this project, define the cluster IP as the maximum IP address value of all computers within the cluster. Interpret this IP as the router IPPrefix.
- You should end up with k clusters at the end of this step. Each router represents its own cluster. You need to create a **Router** object for each cluster with the IPPrefix from the previous step and add all the computers within that cluster to the router's **computers** data structure. You also should use some data structure to store these router objects to access them later.

At the end of this method, the data structures **computerGraph**, **cluster** and various individual **Router** objects should be correctly populated. An example with $k = 2$ is given below:

Examples

Sample input:
```
6
47 53 40
40 26 90
53 13 5
40 53 25
26 47 23
47 40 48
```

Unique Index Assingment:
$47 \to 0$
$53 \to 1$
$40 \to 2$
$26 \to 3$
$13 \to 4$

`computerConnections` data structure:
`computerConnections.get(0)` $\to \{0, 1, 40\}$
`computerConnections.get(1)` $\to \{2, 3, 90\}$
`computerConnections.get(2)` $\to \{1, 4, 5\}$
`computerConnections.get(3)` $\to \{2, 1, 25\}$
`computerConnections.get(4)` $\to \{3, 0, 23\}$
`computerConnections.get(5)` $\to \{0, 2, 48\}$

Running `buildCluster(int k)` wit $k = 2$, we get:

`computerGraph`:
`computerGraph.get(0)` $\to \{3\}$
`computerGraph.get(1)` $\to \{2, 4\}$
`computerGraph.get(2)` $\to \{1\}$
`computerGraph.get(3)` $\to \{0\}$
`computerGraph.get(4)` $\to \{1\}$

`cluster`:
`cluster.get(0)` $\to \{47, 26\}$ {`IP of index 0, IP of index 3`}
`clusterID` $\to 47$
`cluster.get(1)` $\to \{53, 40, 13\}$ {`IP of index 1, IP of index 2,IP of index 4`}
`clusterID` $\to 53$

`Routers`:
`RouterA`
`RouterA.getIPPrefix` $\to 47$
`RouterA.getComputers()` $\to 47, 26$
`RouterA` $Unique\ \ Index \to 0$
`RouterB`
`RouterB.getIPPrefix` $\to 53$
`RouterB.getComputers()` $\to 53, 40, 13$
`RouterB` $Unique\ \ Index \to 1$

[a] It is a forest of trees as each cluster/componenet is a tree and we have multiple such clusters/components hence making the graph a forest of trees

Milestone 3: Building network as multiple clusters

Once we have the k routers, each representing its own network, we would like to add capabilities for inter-network communication. The connections between routers will be given as a separate input of the same form as that of the connections between computers in part 1. We implement this functionality in `connectCluster()`

m'
$u'_1 \;\; v'_1 \;\; l'_1$
$u'_2 \;\; v'_2 \;\; l'_2$
$\vdots$
$u'_m \;\; v'_m \;\; l'_m$

Here, m' is the number of edges in this new router-network, and the m' lines following contain information about the edges. The ith line u'_i v'_i l'_i may be interpreted as an edge between router with `IPPrefix` u'_i and router with `IPPrefix` v'_i with latency l'_i. Note that u'_i, v'_i are the router ids assigned in the previous part. Just as in part 1, you must build a network (graph) by reading in this input and using the connections given in the input as connections in the network. You must do the follwoing:

- From the previous parts, we have each cluster mapped to a unique index (**CI**), each cluster mapped to a router object (let's call the mapping **CR**). We want to combine these mappings and create a mapping **RI** where each router is mapped to its respective index gotten from **CI** and **CR**
- We use this mapping to create an adjacency list `routerGraph` that represents the router graph generating using the **RI** mapping and the edges read in from the input. Any `routerGraph.get(i)` consist of a list of all nodes connected to the ith node along with their weights. For example if the ith node is connected to nodes u and v with weight w_1 and w_2 respectively, `routerGraph.get(i)` would consist of $\{\{u,w_1\},\{v,w_2\}\}$.

`cluster`:
`cluster.get(0)` $\to \{47, 26\}$ {`IP of index 0, IP of index 3`}
`clusterID` $\to 47$
`cluster.get(1)` $\to \{53, 40, 13\}$ {`IP of index 1, IP of index 2,IP of index 4`}
`clusterID` $\to 53$

`Routers`:
`RouterA`
`RouterA.getIPPrefix` $\to 47$
`RouterA.getComputers()` $\to 47, 26$
`RouterA` $Unique\ \ Index \to 0$
`RouterB`
`RouterB.getIPPrefix` $\to 53$
`RouterB.getComputers()` $\to 53, 40, 13$
`RouterB` $Unique\ \ Index \to 1$

Sample input:
```
1
47 53 10
```

`routerGraph`:
`routerGraph.get(0)` $\to \{\{1, 10\}\}$
`routerGraph.get(1)` $\to \{\{0, 10\}\}$

Milestone 4: Inter-cluster Communication

Once we have the router-graph from the previous part we would like to ensure efficient inter-cluster communication. We define efficiency in communication as minimizing the total latency required to get from the source computer to the target computer. By our network construction procedure, all computers within a cluster have a direct link to the router which represents the cluster. Thus the problem of inter-network efficient communication reduces to finding the shortest path between a source and target router. Now, our *extended IP address* is of the form

`routerID.computerID`

Where **routerID** is IP address of a router and **computerID** is IP adress of a computer withing that router's network. Note that the IP address of any computer is unique across all routers. In this part, you will be given two such extended IP addresses `source and target` . You must find the shortest path in the router-graph between the routers corresponding to these IP addresses by implementing the `traverseNetwork()` method which returns the total latency of the shortest path.

Coding

You will be given skeleton code for each of the files you need to submit. You will need to complete all the `@TODO`s in order to be able to get full points.

Notes

Submission

You are required to submit:

- `Network.java`
- `Router.java`
- `UnionFind.java`

All files will be needed to be submitted to Vocareum where they will be run against private test cases for final grading.

Grading

Project Part(s)	Points
Basic Graph	10
Intermediate Graph 1	20
Intermediate Graph 2	20
Advanced Graph 1	20
Advanced Graph 2	20
Edge Cases	10
Total	100