PROGRAMMING ASSIGNMENT 3

IMPLEMENTING THE CHORD PEER TO PEER NETWORK

Version 1.1

DUE DATE: Wednesday, April 17th, 2024 @ 8:00 pm

OBJECTIVE

The objective of this assignment is to build a simple peer to peer network where individual peers have 32-bit integer identifiers. This assignment has several sub-items associated with it: this relates to constructing the logical overlay and traversing the network efficiently to store and retrieve content. This assignment will be modified to clarify any questions that arise, but the crux of this assignment will not change.

Grading: This assignment will account for **15 points** towards your cumulative course grade. There are several components to this assignment, and the points breakdown is listed in the remainder of the text. The lowest score that you can get for this assignment is 0. The deductions will not result in a negative score for this particular assignment.

1 Peer Nodes

Peer nodes are identified using peerIDs. Each peerID is computed by calling the *hashCode()* method on the string *<IP>:<port>.* Where *IP* is the IP address of the node and *port* is the port number the node's ServerSocket is listening on. Since the pair IP and port is unique within the system, each peerID should be unique with a very small chance (1 in 4 billion) of a collision. There is 1 point for generating the peer ID, registering it with the discovery node, identifying (and rectifying) unlikely collisions.

2 The Discovery Node

There will also be a discovery node in the system that maintains information about the list of peer nodes. Every time a peer node joins the system it notifies this discovery node and performs a registration, which includes the following information:

- Its peerID
- The *IP address and port* information (please use TCP for communications)

The discovery node checks that the peerID is equal to the hash code of *<ip>:<port>.* And that the IP is the actual IP address where the registration originated.

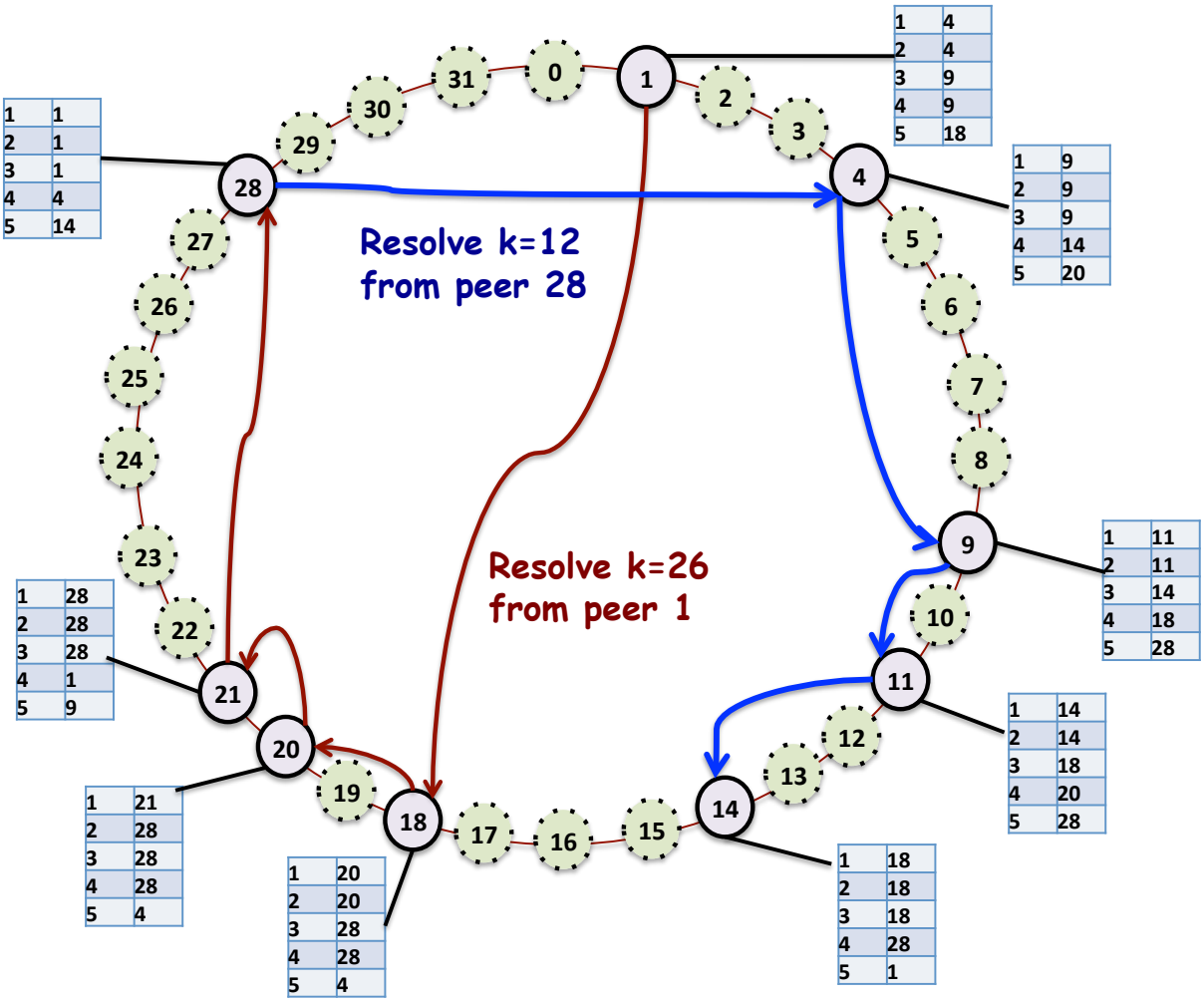
The discovery node has been introduced here to simplify the process of discovering the first peer that will be the entry point into the system. The discovery node is ONLY responsible for

1. Returning **ONE** random node from the set of registered nodes
2. Printing the list of peer nodes that are currently in the system

If the discovery node is used for anything else there will be a **13 point deduction**. An example of misusing the discovery node is to use it to give a new node information about all nodes in the system: such a misuse will defeat the purpose of this assignment.

Points:

1. Adding and removing entries from the discovery node when a peer either joins or leaves the system. (**1 point**)
2. Returning a random *live* peer's network information when a peer joins the overlay (**1 point**)



3 Finger Table:

In the Chord system with a 32-bit ID space; each peer maintains a Finger Table (FT) with 32 entries. This FT is used to traverse the overlay efficiently. The i^{th} entry in the FT of a peer corresponds to a successor peer that is 2^{i-1} hops away. The i^{th} entry in the FT of a peer with id p is $FT_p[i]$ = succ(p + 2^{i-1}). In a case where all 2^{32} peers are present; the FT allows you to reach peers that are 1, 2, 4, …, 2^{31} hops away.

A data item with a computed hash code of k is stored at a peer with the smallest identifier that is >= k. This is the successor of k, and is represented as *succ(k)*. The nodes are organized in a ring, so it is possible that a node's successor has a value that is less-than the identifier of the peer in question.

An example (from the book by Tanenbaum and van der Steen) depicting the finger table in a system with 2^5 nodes, and thus having 5 entries is depicted below.

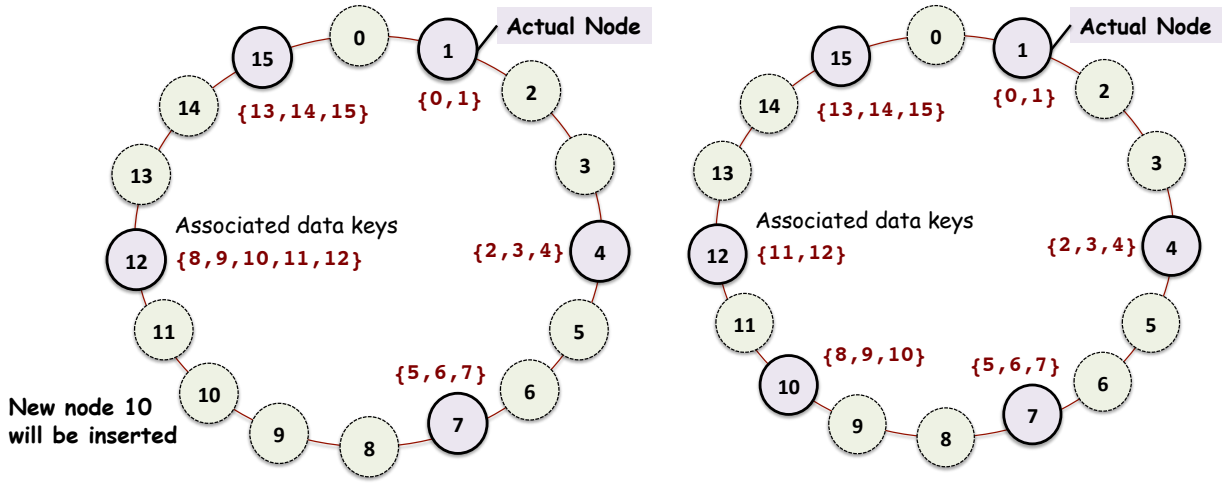
Since the system is in flux with the peers leaving and joining randomly, every time a node is added you can expect the finger tables at some of the nodes to change in response to this churn.

4 Storing data items

You must use the FT to store data items at the appropriate peer node. A data item with a key k will be stored at the peer with the *smallest* identifier, p, such that p >= k. The data items that will be given to you are files. The key is the hash code of the name of the file (including the extension and excluding the path). For example, the file */tmp/work/project.txt* will have *"project.txt".hashCode()* as key. When a peer wants to store a new file, it first computes the 32-bit hash code of the name and then it uses the FT to ***lookup*** the peer node where the file should be stored: the node that gets back to you will be the node that is most suitable to store your data. The file is then transferred to that suitable peer node, which stores the file on the machine that it is running on. Each node must store its files in the folder tmp/<peerID>/. Each node is responsible for creating its storage folder if it does not exists. For this assignment we will not test files larger than 1MB.

Points:

1. Identification of peer to store content **(1 point)**.
2. Using the FTs to take the correct route to reach the targeted peer **(3 points)**.



5 Addition of a node:

Each node also keeps track of its *predecessor*. When a node finds a successor node, it informs this node that it is now its predecessor. To maintain correctness, at regular intervals (this should be configurable so that we can test this feature during the scoring process) each node *q* uses the first entry in its FT to contact *succ(q+1)* and requests it to return its predecessor. This should be *q*: if this is not the case, we know that a new node p has entered the system *q < p <= succ(q+1)* in which case q has to reset its successor to *p*. It will then check to see if p has recorded q as its predecessor; if not, another adjustment of $FT_q[1]$ will need to be made.

The addition of a new node impacts the overlay network in two ways. First, this results in updates to the FT at one or more peers. Second, the addition of a new node should result in migration of data items from peers that were originally holding them. This is depicted in the figure above.

Points:

1. Adding a new node at the right location in the overlay using the correct route **(2 points)**
2. Creating the FT at the newly added node with the correct entries **(2 points)**
3. Updating the FT at the node that was impacted as a result of this addition. If p was the successor of a node k, and the newly added node q is now the predecessor of p; k would now be the predecessor of q, and q will be the successor of k. **(2 points)**
4. Migrating data items from peers that are now not the most suitable peers to host the data **(2 points)**

6 Commands

Both discovery and peer nodes should support an asynchronous command line interface.

6.1 Discovery Node

On the discovery node, the only command that should be implemented is:

peer-nodes

Prints the list of peer nodes currently connected. The peer nodes should be printed on separate lines sorted by peerID in ascending order. Each line should have the following format: <peerID> <ip-address>:<port>

6.2 Peer Nodes

Peer nodes should support the following commands:

neighbors

Prints information about the neighboring peer nodes in the following format:
predecessor: <peerID> <ip-address>:<port>
successor: <peerID> <ip-address>:<port>

files

Prints the list of files this peer node is responsible for. Each file should appear on a separate line with the following format:
<file-name> <hash-code>

finger-table

Prints the finger table of the peer node. Each row of the table should be on a separate line. Each line should have the format: <index> <peerID>

upload <file-path>

Stores the specified file into the chord system. The file is not directly stored on the peer node where the command is issued. Instead, the file is stored in the peer node with the smallest peerID that is greater than the hash code of the file name (with extension and without path). The peer node where the command is issued is responsible for reading the file and sending it to the correct peer node. Example usage: *upload work/projects/readme.txt*

download <file-name>

Queries the chord system and downloads the file with the given name to the current working directory. <file-name> should contain the extension, but not the original path where it was uploaded from. If the file does not exist, an error message should be printed. If the file is downloaded, the peer node should print the list of hops that was used to retrieve the file. The list should contain the starting peer and the final peer, as well as all intermediate hops. Each peer node should be on a separate line and it should be represented just by its peerID. The peer nodes should be ordered from starting node to final node. Example usage: *download readme.txt*

exit

Gracefully leaves the system and shuts down. As part of this exit the following should occur: (1) the data stored at this node must be migrated to the appropriate peer node, (2) the FT of the remaining nodes should be updated, and (3) the discovery node is notified.

7 Third-party libraries and restrictions:

You are not allowed to use any third party library (there is **15 point deduction** for this). Your build.gradle file should only contain *plugins {id 'java'}* . You can discuss the project with your peers at the architectural level, but the project implementation is an individual effort.

8 Testing Scenario

Command to start the **discovery node**:

```
java csx55.chord.Discovery portnum
```

Command to start the **peer node**:

```
java csx55.chord.Peer discovery-ip discovery-port
```

We will test your code with 1 discovery node and between 1 and 10 peer nodes. We will add and remove peer nodes throughout the execution. Your implementation should handle the migration of files. We will store and retrieve multiple files. We will check the location where the files are stored, the path taken to retrieve the files, and the correctness of the content of the files.

9 What to Submit

Use **CANVAS** to submit a single .tar file that contains:

- The src folder containing all the Java files related to the assignment (please document your code)
- the build.gradle file you use to build your assignment
- A README.txt file containing a description of each file and any information you feel the GTA needs to grade your program.

Filename Convention: All classes should reside in a package called **`csx55.chord`**. The archive file should be named as <FirstName>-<LastName>-HW<x>.tar. For example, if you are Cameron Doe then the tar file should be named Cameron-Doe-HW3.tar.

10 Version Change History

This section will reflect the change history for the assignment. It will list the version number, the date it was released, and the changes that were made to the preceding version. Changes to the first public release are made to clarify the assignment; the spirit or the crux of the assignment will not change.

Version	Date	Change
1.0	3/20/2024	First public release of the assignment
1.1	3/20/2024	Added the **exit** command in section 6.2