PROGRAMMING ASSIGNMENT 2

IMPLEMENTING THE CHORD PEER TO PEER NETWORK

Version 1.0

DUE DATE: Wednesday, October 20th, 2021 @ 5:00 pm

OBJECTIVE

The objective of this assignment is to build a simple peer to peer network where individual peers have 16-bit 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 **20 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. This is a team assignment to be done in a group of two. The scoring process will involve a one-to-one interview session of approximately 20 minutes where you will demonstrate all the required functionality based on the inputs that will be provided to you. The slots for these interview sessions will be posted a few days prior to the submission deadline.

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.

Generating identifiers **(1 point)**

The first time you create a peer you will rely on local timestamps to generate the 16-bit identifier i.e. the total number of peers in the system can be about 64,000. When a peer starts up, you should allow this ID to be specified (as a HEX string) at the command line. I have attached my Java code for converting a Hex String into a byte[] and vice-versa in the appendix portion of this assignment. Use this also for printing out the entries in the Finger Table (FT) and so forth. This feature of assigning identifiers statically to nodes will be used during the scoring process.

The Discovery Node

There will also be a discovery node in the system that maintains information about the list of peers in the system. Every time a peer joins or exits the system it notifies this discovery node. The registration information includes information about the peer such as:

- Its 16-bit identifier
- The {*host:port*} information (please use TCP for communications)
- A nickname for this node

In the unlikely case that there is a collision in the ID space, the discovery node notifies the peer about this collision at which point the peer will regenerate a new identifier and repeat the process.

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 only for

1. Returning **ONE** random node from the set of registered nodes
2. Detect collisions

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:

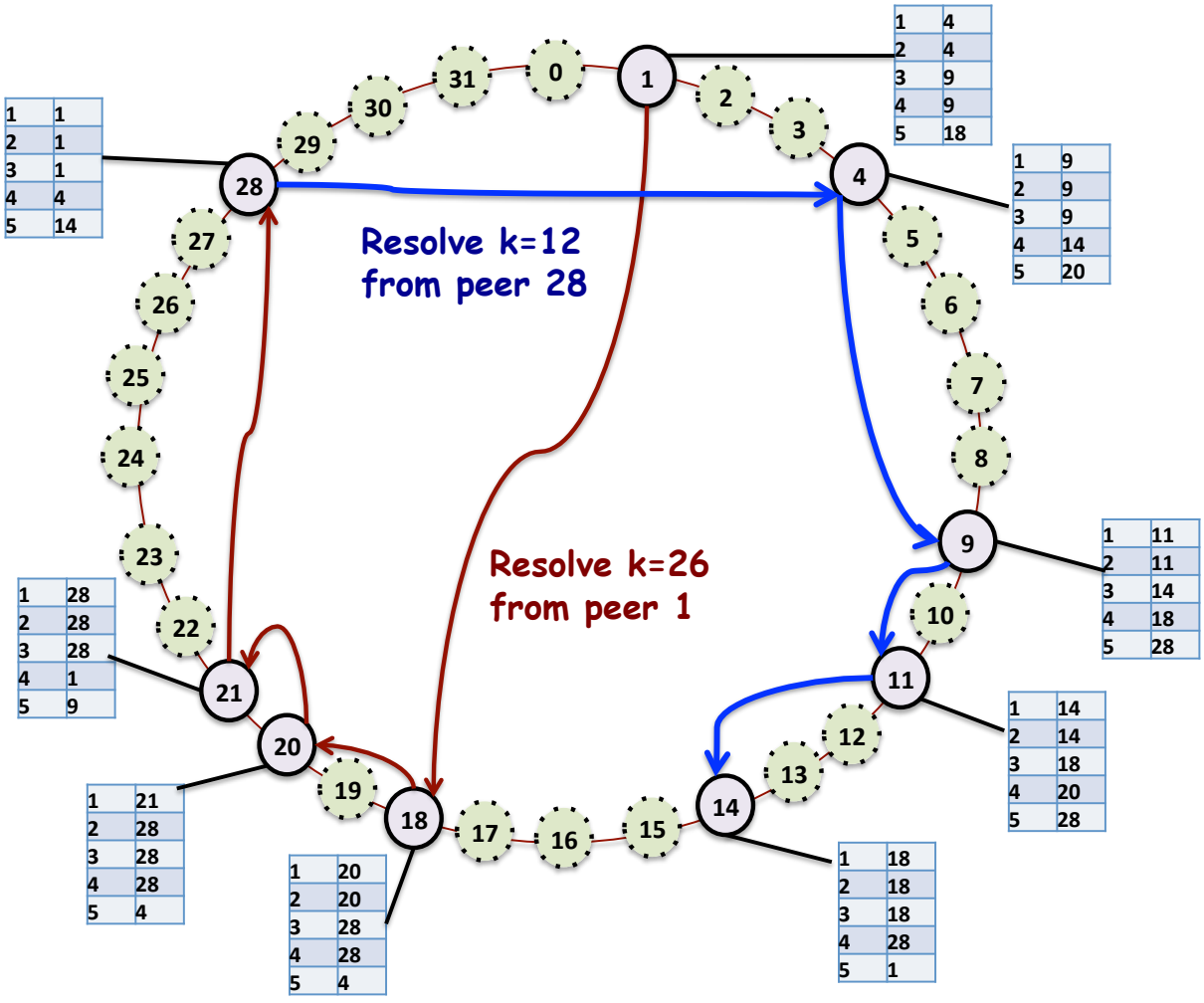
1. Adding and removing entries from the discovery node when a peer either joins or leaves the system. (**1 point**)
2. Returning a random *live* peer's network information when a peer joins the overlay (**1 point**)

Finger Table:

In the Chord system with a 16-bit ID space; each peer maintains a Finger Table (FT) with 16 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^{16} peers are present; the FT allows you to reach peers that are 1, 2, 4, …, 2^{15} hops away.

A data item with a computed digest 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.



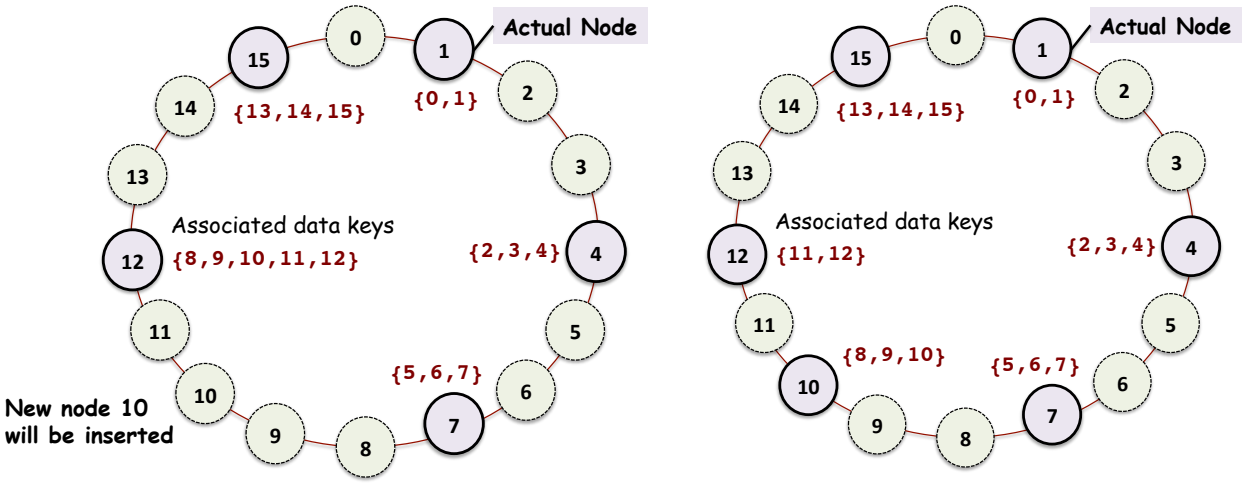
Since the system is in flux with the peers leaving and joining randomly, every time a node is added you can expect the finger tables at some of the nodes to change in response to this churn.

Storing data items

You must use the FT to store data items at the appropriate 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 item that will be given to you will be images. To support this feature you will develop a StoreData program that accepts as input the file that needs to be stored. This StoreData program contacts a random peer (you can contact the Discovery node to retrieve this information). The StoreData program will first compute the 16-bit digest for the image and then use this hash to ***lookup*** the peer where it should be stored: you will be contacting the aforementioned random peer to initiate this lookup; 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, which is responsible for storing the file in the /tmp directory of the machine that it is running on.

Points:

1. Identification of peer to store content **(2 point)**.
2. Using the FTs to take the correct route to reach the targeted peer **(4 points)**.
3. Storing the file in the /tmp directory of the target peer **(1 point)**



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 **(3 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 **(3 points)**

Diagnostics

To make sure that things are progressing correctly this assignment requires that several diagnostic information be printed on the console. These include:

1. The FT at a node
2. The successor and predecessor of a given node
3. The list of files managed by the node: This would be stored in the /tmp directory of the machine on which the peer is running
4. Print out a message every time you route a query: This message should indicate the hop number that it corresponds to in the routing path. So, if a lookup() operation has bounced off of 3 nodes, and is now received at a node … it should print a hop count of 4.
    a. We will use this information to reconstruct the path which the lookup/successor operations took.
    b. In the absence of this hop information, all that would need to be done is to sort the peer ids appropriately to simulate the correct path.

Lack of diagnostic information for checking the correctness of your routing paths will result in a **6 point deduction.**

Programs that you are responsible for developing

In summary you are responsible for developing 3 distinct programs

1. Discovery Node
2. Peer Node with a configurable port number that prints out several diagnostic information
3. StoreData which is responsible for ensuring the storage of the content on the correct peer in the system

Third-party libraries and restrictions:

You are allowed to use a 3rd party library for the hash function, but note that this will be a 16-bit hash. You can also use networking capabilities provided in the language of your choice. You are not allowed to download *any* other code from *anywhere* on the Internet. You are also not allowed to use RPC or distributed object frameworks to develop this functionality (there is **15 point deduction** for this). You can discuss the project with your peers at the architectural level, but the project implementation is an individual effort.

Testing Scenario

You will be asked to launch between 10-20 processes possibly on different machines. The port number on which your peer runs should be configurable. You may be asked to run 2 or more peers on the same machine.

Submission deadline:

Please submit the source codes for your project by 5:00 pm on the due date. Please submit a zip file containing your source codes and a Readme.txt using **CANVAS**. We will rely on the honor system: please do not make any modifications to the codebase after the submission deadline has elapsed. There will be steep deductions for making modifications to the source code after you have submitted it.

Nota Bene: Please do not e-mail the source codes to the Professor or the GTA – there will be a 2 **point** deduction for doing this.

Change History

Version	Date	Change
1.0	9/10/2021	First public release of the assignment

Appendix A

```
/**
 * This method converts a set of bytes into a Hexadecimal representation.
 *
 * @param buf
 * @return
 */
 public String convertBytesToHex(byte[] buf) {
      StringBuffer strBuf = new StringBuffer();

      for (int i = 0; i < buf.length; i++) {
            int byteValue = (int) buf[i] & 0xff;
            if (byteValue <= 15) {
                  strBuf.append("0");
            }
            strBuf.append(Integer.toString(byteValue, 16));
      }
      return strBuf.toString();
}


/**
 * This method converts a specified hexadecimal String into a set of bytes.
 *
 * @param hexString
 * @return
 */
public byte[] convertHexToBytes(String hexString) {
      int size = hexString.length();
      byte[] buf = new byte[size / 2];

      int j = 0;
      for (int i = 0; i < size; i++) {
            String a = hexString.substring(i, i + 2);
            int valA = Integer.parseInt(a, 16);

            i++;

            buf[j] = (byte) valA;
            j++;
      }

      return buf;
}
```