Mini Project
5th Semester
Chord Distributed Hash Table

Design and develop a application for implementing Chord DHT :-

1. A **distributed hash table** (**DHT**) is a class of a decentralised distributed system that provides a lookup service similar to a hash table: (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

2. In computing, **Chord** is a protocol and algorithm for a peer-to-peer distributed hash table. Chord specifies how keys are assigned to nodes, and how a node can discover the value for a given key by first locating the node responsible for that key. Chord is one of the four original distributed hash table protocols, along with CAN, Tapestry, and Pastry.

3. Read the paper to implement chord.
   https://drive.google.com/file/d/0ByssYdG6YTB4NHFnN29GdTZTcFE/view?usp=sharing

4. **Protocol :-**
   The Chord protocol and algorithms are described in the paper. You should read the paper to learn about the design of Chord, prior to starting your own implementation. The implementation in the paper, shown in Figure 5 and Figure 6, is presented as pseudocode. The pseudocode involves both local and remote function call invocations, determined by the node at which a function call is invoked.
   Note that this pseudocode is extended by Section IV.E.3 “Failure and Replication”, with changes to the ‘stabilise’, ‘closest preceding node’, and ‘find_successor’ function calls. Additionally, there is a new function call ‘get successor list’ hinted at in the description, which returns a node’s list of successors and is used in the extended function calls. We are providing the GPB protocol message types for you, which may be found in the ‘assignment5’ directory in the ‘materials’ repository. For this project, you may safely ignore the following portions of the paper:
   1. Section II “Related Work”, Section V “Evaluation”, Section VI “Future Work”, Section VII “Conclusion”.

   2. **Chord Client:-**
      The Chord client will be a command line utility which have following commands :-
      1. port \`port number\` = if port number is provided then the program will change it’s port number, otherwise will print the current port on which it is running.
      2. create = Create itself as DHT main node, every node will connect through this node.
      3. join \`ip\` \`port\` = will send a request to ip and port to join the DHT. ip and port are of main node. It will also transfer the required key-value from other nodes to itself (required means those key-value whose key’s hash is less than this but now this is the maximum node).
      4. put \`key\` \`value\` = will put the key after hashing it on a computer whose “ip:port” hash is more then key’s hash but is minimum. That is, on the computer which is key’s successor.
      5. get \`key\` = will return the value after searching it on the network. If key-value is not present it will return not found.

6. print state = will print the id of itself with ip and port, and it's complete finger table.
7. leave = will leave the network and transfer all its key value to it's successor.

The initial steps the Chord client takes when joining the network are described in detail in Section IV.E.1 "Node Joins and Stabilisation" of the Chord paper.
Periodically, the Chord client will invoke various stabilisation routines in order to handle nodes joining and leaving the network. The Chord client will invoke 'stabilise', 'fix_fingers', and 'check predecessor' every (100 mili seconds) respectively.

Commands
The Chord client will handle commands by reading from stdin and writing to stdout. There are two command that the Chord client must support: 'get', put and 'printstate'. 'get' takes as input an ASCII string (e.g., "Hello"). The Chord client takes this string, hashes it to a key in the identifier space, and performs a search for the host that is the successor to the key (i.e., the owner of the key). The Chord client then outputs the IP address and port information of the host. 'printstate' requires no input. The Chord client outputs its local state information at the current time, which consists of:

1. The Chord client's own node information.
2. The node information for all nodes in the successor list.
3. The node information for all nodes in the finger table
   where "node information" corresponds to the identifier, IP address, and port for a given node.
   An example sequence of command inputs and outputs at a Chord client is shown below. There are four participants in the ring (including the Chord client itself) and -r is set to 2. You may assume the same formatting for the input, and must match the same formatting for your output. ">" and "<" represent stdin and stdout respectively. The input commands will be terminated by newlines ('\n').

```
> put five 5
< five and 5 are stored in the table
> get five
< value found 5
> get nine
< value not found
>printstate
<Self 31836aeaab22dc49555a97edb4c753881432e01d 128.8.126.63 2001
<Successor[1] 6fa8c57336628a7d733f684dc9404fbd09020543 128.8.126.63 2002
<Successor[2] 7d157d7c000ae27db146575c08ce30df893d3a64 128.8.126.63 2003
<Finger[1] 6fa8c57336628a7d733f684dc9404fbd09020543 128.8.126.63 2002
<Finger[2] 6fa8c57336628a7d733f684dc9404fbd09020543 128.8.126.63 2002
...
<Finger[159] 7d157d7c000ae27db146575c08ce30df893d3a64 128.8.126.63 2003
<Finger[160] f4d60480373006cb24147cd17765000f14aadca3 128.8.126.63 2004
```

5. General Instructions

1. Use any object oriented language for implementation.
2. The system should be able to compile on linux and windows.
3. For creating design use Umbrello UML or Argouml.