CE252A Project A: Implementation of DHT

In this project, you are going to working on one of the most cited implementation of DHT: Chord (http://nms.lcs.mit.edu/papers/chord.pdf). This topic was covered in the lecture (https://cmpe252a-fall17-01.courses.soe.ucsc.edu/system/files/attachments/Lecture02_0.ppt).

The main task of this project is to implement different functions of a node class. The skeleton of the program will be provided for reference (it is written in c++. We are sorry for other language users). Your job is to implement some interfaces of a class. Your program will be tested against a set of member function calls.

For ease of grading, the keys and node identifiers are 8-bit. Hash functions (e.g. SHA-1 in Chord paper) are omitted in the project for the same reason.

When a node join the DHT network, the node needs to build its own finger table for routing purposes. To bootstrap this process, the node is provided with another node that is already in the Chord network by some external mechanism. The following member function should be implemented. When a node joins, some keys should be migrated to it as well.

```
void Node::join(Node* node);
```

Chord paper presents some optimization techniques to initialize the finger table, it is optional to implement one of them. We will not consider multiple simultaneous joins in this project.

When a node leave the DHT network, the keys maintained in the node should be migrated to another node. Since Chord paper does not describe the procedure to handle node leave in detail, it is optional to to complete the following function:

```
void Node::leave();
```

This function should be able to locate the node the keys should be migrated to as well as to notify other nodes to modify the finger table.

In Chord, each node maintains part of the distributed hash table. On condition that the querying key is not maintained locally, the node needs to query the key from the Chord network. Likewise, the node is also able to insert/remove keys in corresponding node. Therefore, the following three functions are crucial too.

```
uint8_t Node::find(uint8_t key);
void Node::insert(uint8_t key, uint8_t value);
void Node::remove(uint8_t key);
```

Upon join() and leave(), the most updated finger tables should be printed to the screen. For find(), the sequence of nodes that the node talks to should be printed to the screen too.

All the member functions will be tested in a set of main functions. E.g.

```
int main() {
        Node n0(5); // node_id == 5
        Node n1(63); // node_id == 63
        n0.join(null);  // the first node to join the Chord network.
        n0.insert(3, 3); // insert key = 3
        n1.join(&n0); // the second node join the Chord network.
        n1.insert(5); // insert key = 5
        n0.find(5); // key query
}
```

TIPS:
Chord leverages Remote Procedure Call (RPC) to either iteratively or recursively to lookup a key located at other nodes. It is easy to mimic RPC in simulation settings
For example,

```
int remoteLookup(uint8_t key, Node* remoteNode) {
        return remoteNode->localLookup(uint8_t key);
}
```

It can even be called recursively. For example,

```
int remoteLookup(uint8_t key, Node* remoteNode) {
        return remoteNode->remoteLookup(key, anotheRemoterNode);
}
```

Grading rubrics

1. Build the finger table correctly and print the finger table in the screen when a new node joins. (40 pts)
2. Correct keys are moved when a new node joins the DHT network. Print the keys that are migrated (20 pts)
3. Correctly lookup keys. Print the sequences of nodes get involved in this procedure. (40 pts)
4. [optional] Implement Node::leave() correctly. (20 pts)

5. [optional] Correctly Simulate Space Shuffle https://users.soe.ucsc.edu/~qian/papers/TPDS16.pdf (40 pts)

If you have any questions or find any bugs, please contact TA Xin Li (xli178@ucsc.edu).