ASSIGNMENT 1

* Chord Protocol Implementation:

1. *Run the Code:*

py .\chordProtocol

1. *Sample Input*:

m = 3

1. *Sample Output*:

Number of Bits to be allocated for search space (m): 3

Number of Nodes in the network: 0

Nodes in the network: No Nodes in the network!

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 1

Enter Node ID: 0

----------------------------------------------------------------

Number of Nodes in the network: 1

Nodes in the network: {0 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 1

Enter Node ID: 1

----------------------------------------------------------------

Number of Nodes in the network: 2

Nodes in the network: {0 , 1 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 1

Enter Node ID: 3

----------------------------------------------------------------

Number of Nodes in the network: 3

Nodes in the network: {0 , 1 , 3 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 1

Enter Node ID: 5

----------------------------------------------------------------

Number of Nodes in the network: 4

Nodes in the network: {0 , 1 , 3 , 5 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 1

Enter Node ID: 7

----------------------------------------------------------------

Number of Nodes in the network: 5

Nodes in the network: {0 , 1 , 3 , 5 , 7 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 2

Enter Node ID: 7

----------------------------------------------------------------

Number of Nodes in the network: 4

Nodes in the network: {0 , 1 , 3 , 5 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 4

i id + 2^i successor

Node 0 ->

0 1 3

1 2 3

2 4 5

Node 1 ->

0 2 3

1 3 5

2 5 0

Node 3 ->

0 4 5

1 5 0

2 7 0

Node 5 ->

0 6 0

1 7 0

2 1 3

----------------------------------------------------------------

Number of Nodes in the network: 4

Nodes in the network: {0 , 1 , 3 , 5 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 3

Enter Node ID to search for: 4

Enter Node ID to begin search from: 1

Search Path: 1 -> 3 -> 5

----------------------------------------------------------------

Number of Nodes in the network: 4

Nodes in the network: {0 , 1 , 3 , 5 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 3

Enter Node ID to search for: 6

Enter Node ID to begin search from: 0

Search Path: 0

----------------------------------------------------------------

Number of Nodes in the network: 4

Nodes in the network: {0 , 1 , 3 , 5 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 3

Enter Node ID to search for: 6

Enter Node ID to begin search from: 5

Search Path: 5 -> 0

----------------------------------------------------------------

Number of Nodes in the network: 4

Nodes in the network: {0 , 1 , 3 , 5 , }

1. Join Network

2. Leave Network

3. Search File

4. Show Finger Tables

5. Close

Enter Choice: 5

1. *Complexity Analysis*

The code defines a class called *createChordNetwork* for simulating a Chord-based distributed hash table (DHT) network. Chord is a peer-to-peer distributed hash table protocol that provides efficient key lookup in a decentralized network. Let's perform a brief complexity analysis of the code:

* **Constructor (\_\_init\_\_):**
  + Time Complexity: O(1)
  + Space Complexity: O(1)
  + Explanation: The constructor initializes the Chord network with a input number of bits (m). It calculates the network size (self.space) as 2^m and initializes an empty DHT dictionary (self.DHT).
* **update\_all\_finger\_tables:**
  + Time Complexity: O(n \* m \* log(n)), where 'n' is the number of nodes in the network, and 'm' is the number of bits.
  + Space Complexity: O(1)
  + Explanation: This method updates the finger tables for all nodes in the network. It iterates through all nodes ('n' iterations), and for each node, it updates 'm' finger table entries by calling self.find\_successor, which has a time complexity of O(log(n)) because it performs a binary search on sorted keys.
* **find\_successor:**
  + Time Complexity: O(log(n))
  + Space Complexity: O(1)
  + Explanation: This method finds the successor of a given node. It performs a binary search on the sorted keys of the DHT dictionary to find the successor.
* **find\_predecessor:**
  + Time Complexity: O(log(n))
  + Space Complexity: O(1)
  + Explanation: This method finds the predecessor of a given node. It also performs a binary search on the sorted keys of the DHT dictionary.
* **get\_dht\_size, print\_nodes, join, leave, print\_finger\_tables:**
  + These methods have a time complexity of O(1) because they involve simple operations on the DHT or printing nodes and finger tables.
* **find\_closest\_preceding\_node, find\_closest\_succeeding\_node, is\_key\_present, search:**
  + These methods are used for searching in the Chord network and have time complexity proportional to the number of nodes in the network and the number of bits (m). In the worst case, they can have a time complexity of O(n \* m), where 'n' is the number of nodes and 'm' is the number of bits.
* **input\_network (user interaction function):**
  + Time Complexity: Depends on user input, but the most time-consuming operation is the Chord network operations.
  + Space Complexity: O(1)
  + Explanation: This function allows user to interact with the Chord network, performing operations like joining, leaving, searching, and displaying finger tables. The time complexity of Chord network operations depends on the operation type and Chord method called.
* **main:**
  + Time Complexity: Depends on user input, primarily driven by input\_network.
  + Space Complexity: O(1)
  + Explanation: The main function initializes the Chord network based on user input for the number of bits (m) and then calls input\_network for user interaction.

In summary, the overall time complexity of the Chord network operations depends on the number of nodes and the number of bits (m) used in the network. The most time-consuming operations involve updating finger tables and searching for keys in the Chord network.

* Security Issues of Chord Protocol:

The Chord Protocol is a distributed hash table (DHT) protocol designed for use in peer-to-peer (P2P) networks. It is primarily used for distributed data storage and retrieval in a decentralized manner. While Chord offers several advantages, such as scalability and fault tolerance, it also comes with its share of security challenges and issues. Here are some of the key security issues associated with the Chord Protocol:

1. **Data Privacy**: Chord does not inherently provide data privacy or encryption mechanisms. As a result, data stored within the Chord network can be vulnerable to eavesdropping by malicious nodes or unauthorized access. This issue can be mitigated by using additional encryption layers or secure communication protocols on top of Chord.
2. **Sybil Attacks**: Chord is susceptible to Sybil attacks, where a malicious node can create multiple fake identities within the network to gain disproportionate control over the network. This can lead to data manipulation, censorship, or the disruption of legitimate nodes' operations.
3. **Churn and Node Failures**: Chord is designed to handle nodes joining and leaving the network gracefully, but this feature can be exploited by attackers to disrupt the network's stability. Malicious nodes can artificially generate a high rate of node churn or fail abruptly to create instability.
4. **Routing Attacks**: Chord relies on a structured routing table to find nodes in the network efficiently. An attacker may manipulate this routing table to redirect requests or disrupt the network's normal operation, causing data loss or unauthorized access.
5. **Partitioning Attacks**: Chord is vulnerable to partitioning attacks, where an attacker can isolate a subset of the network from the rest. This can be used to deny access to data or manipulate data within the isolated partition without detection.
6. **Incentive Mechanisms**: In Chord-based P2P networks, there may be issues with incentivizing nodes to participate fairly. Nodes may not always behave honestly, impacting the overall integrity and security of the network.
7. **Bootstrapping Security**: When a new node joins the Chord network, it needs to locate an existing node to obtain information about the network's structure. If an attacker controls the bootstrap node, they can manipulate the joining node's view of the network or launch attacks against it.
8. **Secure Identifier Assignment**: Chord uses identifiers to assign nodes to points in the ring-based network. Ensuring secure and unique identifier assignment is crucial to prevent attacks related to node impersonation and data tampering.

To address these security issues, Chord-based applications often implement additional security measures, such as authentication mechanisms, data encryption, and reputation systems. Additionally, research is ongoing to develop more secure DHT protocols and techniques for mitigating these vulnerabilities.

It's important to note that while Chord has security challenges, it also has many merits, such as its scalability and efficiency, making it a popular choice for building decentralized systems. However, understanding and addressing its security issues is essential when implementing Chord in practical applications.

*Sakshi Shashikant Yeole (20CS01047)*