Chord Protocol Simulation

**Main Idea:**

The Chord protocol is a scalable protocol for key-value lookup in a dynamic peer to peer system with frequent node arrivals and departures. In this assignment you will use the skeleton code provided to you and implement a simulation of the this protocol. You will be implementing all the functions on the skeleton code and testing your resulting code on the test log files given to you. The code that you implement should be robust to the arrivals and departures of nodes as well as addition of keys at any point of time.

**Part 1:**

Part 1 of the assignment will be to implement the functions for the ChordNode.

**Requirements and Restrictions:**

* You may **not** use any other imports than the ones given to you in the Skeleton Code
* Your code should compile with Python2.7 and Python3.5+
* You code should not print anything extraneous than the print statements given to you. The grading will be done using a `diff ` function between your output and the solution’s output.
* Your Chord Solution should be eventually stable meaning that all finger tables and successor, predecessor values are eventually stable.
* Your ChordNode should not access any methods or members of the ChordRing class **except** the ChordRing.m member/variable
* The solution code “assumes” that the ChordRing is stable before adding any Keys
* You may call stabilize and fix fingers anywhere in readLog()
* You may add helper functions anywhere but ChordRing. These should not trivialize the implementation of ChordNode which should be “distributed” and Nodes only have references to successor, predecessor and finger tables.
* The Log File gives you “hashed” values so assume that the id of the node is already hashed and modded by 2^m
* Assume that there will be no “hash” conflicts for keys or nodes

Step 1:

* Make sure your implementation works with two nodes
* Nodes should have predecessor and successor at this point
* Finger tables should also be populated properly

Step 2:

* Make sure your implementation works with 3+ nodes with all cases
* All nodes should have predecessor and successor pointers set properly
* Finger tables should be populated properly

Step 3:

* Make sure that your implementation allows nodes to leave
* Allow keys to be stored at nodes
* Implement the rest of the functions

**Part 2:**

After completing the implementation of the Chord protocol in part 1, you will need to answer the following questions:

1. In a future implementation of Chord, how would you take into consideration a node failure? (Remember a node voluntarily leaving is not the same as a node failing)
2. Describe briefly what happens when two nodes join in the following situations - Provide examples for both:
   1. Each node joins at different parts of the ring
   2. Each node joins to the same successor
3. When would a key lookup fail in this current implementation? Assume the key exists somewhere in the ring