README.md

Project 2

Group Members:

1. Adish Someshwar Rao
2. Sanjana Rao Guttalu Prasan

What is working:

We have implemented both the 'Chord' protocol as described in the original Chord paper Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications.

The program takes 2 inputs, mentioned shortly, and is run by calling the following command:

main:start(NoOfNodes, NoOfRequests).

1. NoOfNodes -> Any integer value.
2. NoOfRequests -> The number of requests each node must make.

At the end we will provide modifications made to the algorithm to account for various edge cases that were not accounted for in the original paper.

What is the largest network you managed to deal with?

We have managed to run the chord protocol for 200 nodes.

It should be noted that to ensure the network is stable, nodes join the network at fixed intervals of time (5 seconds). Nodes could join sooner, but if multiple nodes were to join within a very short duration of time, the network could no longer be stable, as mentioned in the original paper. It should also be noted, to ensure the network is stable and the network is not flooded with multiple join requests (which send a find successor message) along with multiple key/document requests (which also send a find successor message), the first key/document request is sent out 120 seconds after the first node is spawned (ie 120 seconds after the chord network is created).

Modifications to the Chord Protocol

Create No modifications made.

```
n.create()
    pred = nil
    successor = n
```

Join No modifications made.

```
n.join(n')
    pred = nil
    successor = n'.find_successor(n)
```

Find Successor

```
n.find_successor(id)
    if (n == successor)
        return successor
    elif (successor > n)
        if (id ∈ (n, successor))
            return successor
    elif (n > successor)
        if (id ∈ (n, 2^m] || id ∈ [0, successor])
            return successor
    else
        n' = closest_preceeding_node(id)
        return n'.find_successor(id)
```

Closest Preceeding Node

```
n.find_successor(id)
    for i=m down to 1 do
        if (id > n)
            if (finger[i] ∈ (n, id))
                return finger[i]
        elif (n > id)
            if (finger[i] ∈ (n, 2^m) || finger[i] ∈ (0, id))
                return finger[i]
    return n
```

Stabilize

```
n.stabilize()
    x = successor.predecessor
    if (x == nil)
        return
    if (sucessor == n)
        successor = x
    if (sucessor > n)
        if (x ∈ (n,successor))
            successor = x
    if (n > successor)
        if (x ∈ (n, 2^m] || x ∈ [0, successor))
            successor = x
    successor.notify(n)
```

Notify

```
n.notify(n')
    if (n' == n)
        return
    if (predecessor == nil)
        predecessor = n'
    elif (predecessor < n)
        if (n' ∈ (predecessor, n))
            predecessor = n'
    elif (predecessor > n)
        if (n' ∈ (predecessor, 2^m) || n' ∈ (0, n))
            predecessor = n'
```

Fix Fingers

```
n.fix_fingers()
    next = next + 1
```

```
        if (next > m)
            next = 1
        finger[next] = find_successor((n + 2^(next-1)) mod 2^m)
```

Check Predecessor No modifications.

```
n.check_predecessor()
    if predecessor has failed
        predecessor = nil
```