PROJECT 3
README
DISTRIBUTED OPERATING SYSTEMS PRINCIPLES
COP 5615

Group Info:

	Name	UFID
1.	Manishkumar Chopra	17967121
2.	Nimish Kochhar	61394423

What is working:
For this project, we have used the actor facility in Elixir(GenServer) to implement the Chord protocol described in the research paper *"Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications".* We have also implemented the failure model in which a specified percentage of the nodes fail. The value of m is dynamic and is calculated by taking ceiling(Log(number of nodes)).

Assumptions:
To avoid the delay in the network formation, we assume that all the nodes are in the network from the beginning and create the chord ring with the initialized nodes. For the failure model, we have assigned the nodes a status called "Active". If this value is true, it means the node is up else it means that the node has failed. We also maintain a list of 3 consecutive successors with every node that help us stabilize the network when the nodes fail, as given in the paper. To count the number of hops per request in the failure model, we have included the number of hops while stabilizing the network in the total hops as they implicitly play a role in completing the requests.

Largest Network:

<u>*Without Failure model:*</u>
No. of Nodes=30000
No. of Requests=10
Total hops = 2527534
Total requests = 300000
Average hops = 8.42538

<u>*With Failure model:*</u>
No. of Nodes=6000
No. of Requests=1
Total hops = 540471

No of failed nodes=60
Total requests = 5940
Average hops = 90.98

How to run the program:
Unzip project3.zip, cd into project3 folder
then run the project by typing on the terminal:

```
mix run lib/project3.ex <numNodes> <numRequests>
```

where, **numNodes** is the number of peers to be created in the peer to peer system
and **numRequests** is the number of requests each peer has to make. The program exits when each peer has performed that many requests.A request is sent by each peer every 1 second.

How to run the Bonus portion:
Bonus portion with failed nodes is also implemented and can be run as follows:
Unzip project3bonus.zip, cd into project3bonus folder
then run the project by typing on the terminal:

```
mix run lib/project3.ex <numNodes> <numRequests> <failPercentage>
```

where, **numNodes** is the number of peers to be created in the peer to peer system
numRequests is the number of requests each peer has to make. The program exits when each peer has performed that many requests.A request is sent by each peer every 1 second.
failPercentage is the percentage of nodes that need to be killed.

Failed nodes correctness and finding
To ensure that a failed node is not included in the system, we check every node's "Active" status before visiting it or making a request from it. This ensures that the failed nodes don't play any part in the network. An interesting find was that the percentage of failed nodes has to be very high to reach the rare scenario in which all the successors in the successor list have failed. That being said, due to the fact that we have used r=3 as the size of the successor list, we did see situations in which all the successors in the successor list of a node had failed. We have kept this scenario in mind as well and exit the program when such a case is encountered as that would mean the network is broken. As given in the paper, the value of r has to be at least Log(Number of nodes) and the higher the r the more robust would be the system, so when we take large networks (~10000 nodes) with our r=3, the network breaks when even 10% nodes fail. But when r is taken appropriately large, we see the system get more robust and not falter even when 1000 out of 10000 nodes fail. We have not given the option to dynamically choose the value of r with the program but have tested with different values of r while developing the solution.