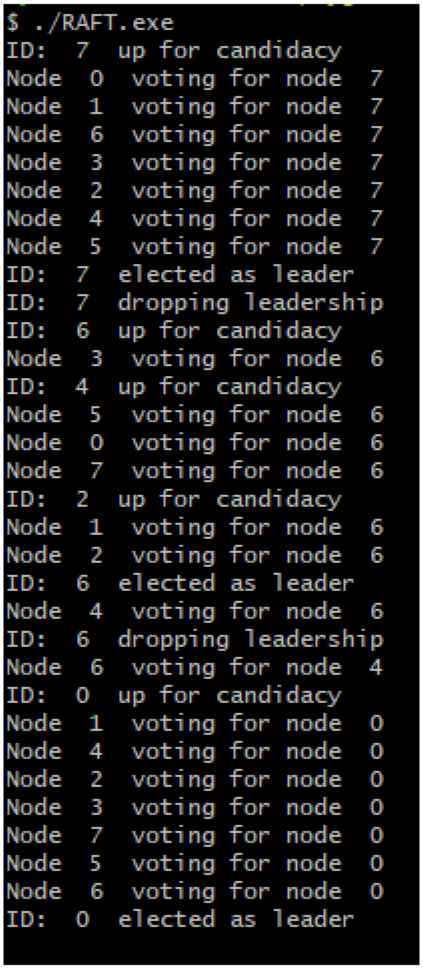
Ryan Myers
Mihir Deshmukh

Consensus Protocol

We decided to implement the RAFT protocol for this assignment. We followed the directions described and set up 8 nodes running as go routines. Each node is modelled as a state machine so that it can move between follower, candidate or leader states depending on what action it just took.

Our algorithm works pretty well and elects a reader within 0.5ms in most cases. Below we have a screenshot of nodes going up for candidacy, receiving majority votes, leaving leadership and another leader being promptly elected 3 times.



```
$ ./RAFT.exe
ID:  7  up for candidacy
Node  0  voting for node  7
Node  1  voting for node  7
Node  6  voting for node  7
Node  3  voting for node  7
Node  2  voting for node  7
Node  4  voting for node  7
Node  5  voting for node  7
ID:  7  elected as leader
ID:  7  dropping leadership
ID:  6  up for candidacy
Node  3  voting for node  6
ID:  4  up for candidacy
Node  5  voting for node  6
Node  0  voting for node  6
Node  7  voting for node  6
ID:  2  up for candidacy
Node  1  voting for node  6
Node  2  voting for node  6
ID:  6  elected as leader
Node  4  voting for node  6
ID:  6  dropping leadership
Node  6  voting for node  4
ID:  0  up for candidacy
Node  1  voting for node  0
Node  4  voting for node  0
Node  2  voting for node  0
Node  3  voting for node  0
Node  7  voting for node  0
Node  5  voting for node  0
Node  6  voting for node  0
ID:  0  elected as leader
```

Figure 1: Example of our RAFT program running with print outputs.

Occasionally, there will be a lot of rounds of elections as multiple nodes try to become leaders all at once but our program resolves itself and elects a leader at the end and will never have multiple leaders.

The code can be built using:

Go build RAFT.go

Then run the executable produced.