Raft implementation note for future review:

General:

1. Every RPC, receiver should check the incoming node term first, ignore if it has lower term. If it is higher, become follow then reset term and votedFor.
2. Every RPC, sender should check the reply term first, become follower then reset term and votedFor if other node has higher term. If not, always check the leader status and if its request.term == rf.currentTerm. Sender only handle the reply if the term doesn't change to prevent process outdated reply

Leader election:

1. After getting elected. It is important to set nextIndex[i] = lastIndex+1, and matchIndex[i]=0
2. After a node get disconnected from everyone, it will keep increasing its term. After it back online, it will make everyone follower, and this node will not be elected most of case since its log is not up to date. TODO: prevent this happen

Log replication:

1. First Log entry is a DummyEntry. In the beginning its index should be 0. After the snapshot, it has index Snapshot Index.
2. For Follwer, when it receive a AppendEntryRPC request, even though the request.PrevIndex and request.PrevTerm match this follower's log, follower shouldn't just copy the received Entries from PrevIndex. It might cause log to rollback under non-fifo-channel.For example:
   - ->Leader(L) send valide entries{1,2,3} first
   - -> L received 4 from client
   - -> the heartbeat immediately start and send valide entries{1,2,3,4}.

   There is a case that RPC1 with entries{1,2,3} and RPC2 with entries{1,2,3,4} are all in the network. If Follower(F) receives RPC2 first, then receives RPC1. This will make F has log with not updated log {1,2,3}. However, if this happens on majority of the Followers, Leader might think 4 is on majority of the servers. Thus it will commit this 4, then might go crash immediatly after commit. In the end, no servers have 4 on their log, which means system fails the correctness
3. During checking the commit for Leader, as paper says, Leader only commit Entry in current term, instead of older term. Thus, when there is a new leader get elected, if client doesn't send any new request, all the ready-to-commit entries cannot be committed, which will be inefficient for real-world system. This can be simply fixed by append a NULL command when a new leader get elected. Unfortunately, lab test doesn't allow this happen

Persistence:

1. Save the votedFor, because a node can:

   -> vote for server1
   ->die
   ->back online quickly in the same term
   ->vote for server2

Snapshot:

1. If a Follower(F) decide to install snapshot from leader, and this snapshot index is biger than this F's commitIndex, it will request KV layer to replace current state machine with this leader's state machine. For KV layer the linear order of changing its state machine should be :

   -> Apply all committed entries with index X, which for all X <= SnapshotIndex
   -> Apply Snapshot to state machine
   -> Apply entries with index Y, which all Y > SnapshotIndex

   This linear order change has to be true all the time to ensure the correctness. Otherwise, follower can be elected as leader in the future but has mismatching state machine

Key-Value layer:

1. Before send client request to Raft, we should check if the log need to be compact. If true, we need to wait until it compacts the log and takes the snapshot If we don't do this, the log might grow out of persistence memory. However, lab test doesn't allow this, since it doesn't allow '3' (in above Log replication part 3). If we put this checking condition without '3', we might stuck forever, since a new leader can't not commit the old term entries, which will trigger snapshot