Problem

From n nodes we want to choose one that will be the leader of the rest.

More formal definition of a problem

Definitions

G - Complete Graph.
$V(G)$ - Set of Vertices of graph G
$E(G)$ - Set of Edges in graph G
$P(v)$ - predicate that returns true(or returns his id) if v was chosen, else false(or returns id of the chosen node)

The leader election problem - choose exactly one node $v \in V(G)$ that $P(v) = true$ and $\forall u \in V(G)$, $u \neq v$, $P(u) = false$

Algorithm

The naive algorithm is actually quite trivial. Every node is a king at the start and he sends $n - 1$ messages to his neighbours to find the one with biggest id. The number of messages send by this algorithm is $\mathcal{O}(n^2)$.

The new algorithm will reduce the number of messages to $\mathcal{O}(n \log n)$.

The algorithm below is Afek-Gafni algorithm B.

Each node will spawn two processes at the beginning (in our code all will be done in one process but it is easier to understand when we will talk about two processes). One will be called Candidate - this one will start the conversation. Ordinary process will answer the ask from the other candidate or pass the message to his father (the node that captured him before). The node will change it's father if the new node that arrives at specific node is higher valued than himself. The candidate part of the node sent messages only to ordinary part of other nodes and ordinary part sent messages only to candidate part of the node.

Level

The main determinant which decides who won the confrontation between the nodes is **Level** - which is the number of nodes it has already captured.

Capturing and Elimination rule

To capture node v, the (level, id) of a candidate must be lexicographically larger than the (level, id) of the previous owner of v, and the previous owner must be killed.

What happens when node P arrives at node v

P arriving at node v means that node P send the message to node v which is owned by candidate Q.

- if *(Level(P), ID(P)) < (Level(v), ID(Q))* - P is killed
- if *(Level(P), ID(P)) > (Level(v), ID(Q))* - v gets P level and P is sent to Q

When P arrives at Q:

- if *(Level(P), ID(P)) < (Level(Q), ID(Q))* - P is killed
- if Q has been killed already then P returns to v and tries to capture it

State of the node

Every node will remember some things in itself.

- ID - unique int for each node.
- Untraversed - set of neighbours of this node.
- Leader - ID of leader of the entire network.
- Owner_ID - the ID of the candidate that owns this node (the candidate captured that node).
- Level - the level of the node - the number of nodes that this candidate has captured.
- Father - the node that captured this node.
- Potential_Father - the node that wants to capture this node but it waits for response from the Father.
- Status - status of node (CANDIDATE - alive, ORDINARY - dead, WAIT_FOR_ANSWER - dead but waiting for specific message, ENDING - prepared to end the program).
- Counter - only applies to leader - to end the entire program.
- Queue - the queue of messages that the node didn't need at the time.

Initialize

level := 0,
owner_id := 0,
untraversed := E,
father := nil,
potential_father := nil,
status := candidate,
queue := empty,
counter := 0,

Candidate Process

1. while $untraversed \neq \phi$
   (a) choose $l \in untraversed$
   (b) send $(level, id)$ along l
   (c) R: receive$(level', id')$ from l'
   (d) if $id' == id$ AND !$killed$ then
      - $level = level + 1$
      - $untraversed = untraversed - l$
   (e) else
      i. if $(level', id') < (level, id)$ then DISCARD message GOTO R
      ii. else
         - send $(level', id')$ along l'
         - $killed$ = true GOTO R
2. if !$killed$ then $announce(ELECTED)$

Ordinary Process

While (not terminated)

1. receive($level', id'$) from l'
2. case $level', id'$:
   (a) $level', id' < level, owner\_id$ DISCARD message
   (b) $level', id' > level, owner\_id$
      - $potential\_father = l'$
      - $level = level'$
      - $owner\_id = id'$
      - $if father == nil$ then $father = potential\_father$
      - send $(level', id')$ along $father$
   (c) $level', id' == level, owner_id$
      - $father = potential\_father$
      - send $(level', id')$ along $father$

Messages

We will define some types of messages:

- ARRIVE - the message send by candidate process to ordinary to try to capture the ordinary node. If the ordinary process of the node was never captured that means the candidate process of the node is still alive. The ARRIVE sequence will cause one of the to die.
- ACCEPT_ANSWER - the message send by the ordinary process to candidate to accept him as his father, the ordinary process is no longer candidate (if it was).
- ASK - the message send by the ordinary process to candidate because he saw node with higher (level, id) than the ordinary node - he send it to the father to eliminate one node (father or sender). He waits for one of the three messages below (he doesn't process other messages in between - not including LEADER, END).
- ANSWER_ACCEPT_CANDIDATE - the message send by candidate process in response to ASK message. The candidate accepted his fate and was killed by the ASK message, he needs to inform the sender of ASK message.
- ANSWER_DENY_CANDIDATE - the message send by candidate process in response to ASK message. The candidate managed to defeat ASK message but need to response to the asker.
- DEAD - the message send by candidate process in response to ASK message. However the node is already dead (ordinary process and candidate process runs until the end of program - dead candidate process only answers the ASK messages and don't send new ARRIVE messages).
- LEADER - the leader was found! The node send that information to every node with his id. Every other node will now send END message to the leader and prepare to end the program.
- END - if the node is not the leader then he sends this messages to the leader informing leader that he is ready to end the program (he will receive messages but it will discard them). If leader gets n - 1 END messages (from every neighbour) then he sends END message to all nodes and then the program ends. It is needed to have two kinds of messages because if there was only END message (if node receive it - ends) then there would be a possibility that some node would send on closed channel.

Implementation details

First of all algorithm above assume that channels are not blocking. However in our implementation the node can only send one message without blocking the channel. If it sends two to the same channel the node waits until the first message is read.

So the first change we implement is that every ordinary process wait for response from candidate's father. We need to change also candidate process, Now candidate on response to ASK message from ordinary process will return one of three values:

1. DEAD
2. ANSWER_ACCEPT_CANDIDATE
3. ANSWER_DENY_CANDIDATE

Proof of correctness

The pair (level, id) is always increasing so there will always be the maximum candidate.

Proof of complexity

Lemma 1. *For any given k, the number of candidates that own $\frac{n}{k}$ or more nodes is at most k.*

Proof. Let C_1 and C_2 be any two candidates which owned $\frac{n}{k}$ nodes at some point in time. C_1 and C_2 must have owned disjoint sets of at least $\frac{n}{k}$ nodes each. If they never tried to claim a node from each other, we are done. The first time that C_1 (without loss of generality) tries to claim a node, say v, from C_2, either one of them dies, or C_2 has already been killed. If C_1, without loss of generality, caused the death of C_2, then clearly it must have owned at least $\frac{n}{k}$ nodes disjoint from C_2, at the time of killing. If C_2 is already dead, C_1 must still own at least $\frac{n}{k}$ nodes in order to claim v to itself. Remember that level is the number of nodes that candidate captures. □

Corollary 1. *The largest candidate to be killed by another candidate owns at most $\frac{n}{2}$ nodes; the next largest owns at most $\frac{n}{3}$ nodes. The k-th largest candidate killed by another candidate owns at most $\frac{n}{k+1}$ nodes.*

Lemma 2. *The message complexity of Algorithm B is $4n \ln n = 2,773n \log_2 n$ messages plus $\mathcal{O}(n)$ ending sequence messages.*

Proof. Capturing the nodes take at most 4 messages (ARRIVE, ASK, response to ASK, ANSWER_ACCEPT) in other cases less. So candidate that owns k nodes, created (not send) at most $4k$ messages. By Corollary 1 the total send messages is bounded by $4n \sum_{i=1}^{n} (\frac{1}{i})$ plus the messages by ending sequence which is equal to $3(n-1)$. □

References

[1] Yehuda Afek, Eli Gafni (1985). *Time and Message Bounds for Election in Synchronous and Asynchronous Complete Networks.* SIAM Journal on Computing, Vol. 20, No. 2, pp. 376–394 (1991).