Bright Idea

Let's imagine we have a problem, we have just conquered some big area with n cities. We are unable to rule over them ourselves so we give each city to some low level governor (a King).

But we need someone who will maintain order and rule over them, like a King of Kings! Unfortunately, after we granted n kings land, we are in no position of revoking them, so one of them should be the overlord of them, they would rebel otherwise. We also don't have time to elect a king ourselves as we need to return home.

So we ordered them to find their new King. As a result each King will try to expand theirs sphere of influence as much as possible by asking (politely) other kings to join his kingdom.

The most successful will be granted the title of "King of Kings".

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 $\mathcal{O}(n^2)$.

We will try to limit the number of messages to $\mathcal{O}(n \log n)$ by using maximum spanning tree of this graph. The main idea will be using nodes that are no longer contenders as a messengers between kings.

Let's call Overlord of node the node that in the MST is the father of that node. Vassals will be the sons and the King will be the root of that tree. Each tree will act as a Kingdom. At the beginning every King has his own small Kingdom.

Every node will also remember some data:

- ID - int unique for each node
- Active/Role - true = KING the node is still a candidate for the title King of Kings, false = CITIZEN the node became the Vassal of another King and is out of question for the King of Kings title In our version of algorithm the Role will be one of 4 values:
  - KING - basically Active = true
  - CITIZEN_MAIN It won't appear in pseudocode but it is just a flag to note that this citizen is on his main loop
  - CITIZEN_UPDATE - this citizen is waiting for UPDATE from his king (after becoming citizen and changing his King)
  - CITIZEN_ASK_BEFORE - this citizen is waiting for ACCEPT or UPDATE message from his King (it is used in processAsk procedure in else if condition)
- Phase - the current phase of a King - it increases when King defeats other King with the same Phase
- Rivals - the set of unused edges from current node - for Citizen they are useless but for King it means that this are the potential King of Kings and he need to find out
- Vassals - the set of nodes that swore fealty to this node. In the words of the MST - set of sons of this node

- Overlord - the node that this node is a Vassal (father of this node)
- King - the ID of the Overlord of Overlord ... of Overlord (root of a tree). At the end of the program is will be King of Kings
- AskedBefore - only matters for Citizen - the node which sent the ASK message, it is needed if the King of this Citizen would response with ACCEPT.
- QUEUE - list of received messages - sometimes we need to wait for specific message (for example UPDATE) in that case we need to remember the received messages that will not be processed immediately.

Our algorithm will use few types of messages:

- ASK(phase, king) This message is created and sent by the King to one of his Rivals (it can also be sent by Citizen, but in that case the message is pushed to his King as a response of other King sending ASK message to the Citizen). The King is asking other Kings to join his kingdom and becoming his vassal. The outcome of ASK message is determined by phase, king values. To become King of Kings a king must receive response for every ASK message =¿ (ACCEPT, YOUR_CITIZEN)
- ACCEPT(phase) This message is sent from King to King as a response to ASK message, The sender will become Vassal of the receiver. If Citizen receives this message from his Overlord it means that his old King is now a Vassal and the sender of ASK message is now his new Overlord.
- UPDATE(phase, king) This message is sent from the King as response to ACCEPT message to his new Vassals (when Phase changes also this is sent to old Vassals). Vassals update theirs saved values to phase, king.
- YOUR_CITIZEN - This message is sent from Citizen to his King as a response to ASK message (the Vassal node is not a direct Vassal of the King, Overlord != King)
- I_AM_THE_KING - This message is sent to each node when the Leader is found.

King process

1. Repeat until is King or Rivals not empty
   (a) choose $e \in$ Rivals and send ASK message along e
   (b) LABEL: receive(m)
   (c) case m:
   YOUR_CITIZEN continue
   ACCEPT(phase) Vassals = Vassals $\cup$ e
   if phase == King.phase
   King.phase++
   send UPDATE to every Vassal
   else send UPDATE along e
   ASK(phase, king)
   – if (King.phase, King.king) > (phase, king)
   – goto LABEL
   – else King is now a Citizen
2. if King.active == false perform procedure of a citizen
   else King is the King of Kings send I_AM_THE_KING to other nodes

Citizen process

1. set as Overlord the King that has just dethroned us
2. send ACCEPT to Overlord
3. wait for UPDATE message (other messages are put in Queue)
4. perform processUpdate
5. Repeat until I_AM_THE_KING not received
   (a) receive(m) (from queue or other node)
   (b) case m:
   ASK perform processAsk
   ACCEPT perform processOldAccept
   UPDATE perform processUpdate

Procedure processAsk

m - received ASK(phase, king)

1. if (Citizen.Phase, Citizen.King) == (phase, king)
   send YOUR_CITIZEN to sender of previous message

   else if (Citizen.Phase, Citizen.King) < (phase, king)
   (a) send m to Citizen.Overlord
   (b) wait from UPDATE or ACCEPT from Citizen.Overlord
   (c) perform processUpdate
   (d) if sender == Citizen.King send YOUR_CITIZEN to sender
   (e) else if ACCEPT then perform processNewAccept

Procedure processOldAccept

1. add sender to Citizen.Vassals
2. send UPDATE to the previous sender

Procedure processNewAccept

1. add sender to Citizen.Vassals
2. set Citizen.Overlord as Citizen.AskedBefore
3. send ACCEPT to the new Citizen.Overlord
4. wait for UPDATE message from Citizen.Overlord
5. perform processUpdate

Procedure processUpdate

1. Update Citizen.phase and Citizen.king
2. send UPDATE to every Vassal

Proof of correctness

Theorem 1. *Only one king is found by the algorithm.*

Proof. Let's assume that the theorem is false. It is impossible to have zero kings, because it would mean that someone dethroned each one. But because of ASK message the king with biggest (King.Phase, King.King) will survive.

So let's assume that number of kings is $s > 1$. Let's define status(node) := (node.Phase, node.King) and the kings that remained as $king_1, king_2, ...king_s$. And assume status($king_i$) < status($king_j$) for $i < j$ □

Lemma 1. *Every node will eventually have the status equal to* $(x, king_i)$ *for some* $1 \leq i \leq s$.

Proof. Otherwise there exist some node v that has a different status forever. If v is king the lemma is true, otherwise the status must have been updated by UPDATE message from some king t. If t's status was changed so was v's. If v transferred the ACCEPT message it was followed by UPDATE message from new King. v will never become King again so the lemma is true for all nodes. □

Lemma 2. *If some King is awake he will send an ASK message to a node not in his kingdom.*

Proof. The king is sending messages to all Rivals. If he sends message to his kingdom he will receive YOUR_CITIZEN message which will cause the king to send another message. □

Lemma 3. *If* $king_s$ *send a message to another kingdom of* $king_j$, $king_j$ *will become a citizen.*

Proof. If $king_j$ directly receives the message he immediately become the citizen. Otherwise it will be transferred by Vassals to the king. The message could be blocked on the way to $king_j$ but that means that the node that blocked the message is waiting for UPDATE from his king as a response for ASK message with higher status than status($king_j$). It means that one of the messages will cause the $king_j$ to become the citizen. □

By Lemma 3 we get contradiction to the assumptions that $s > 1$ The algorithm will stop after the King of Kings announce his presence (by sending the I_AM_THE_KING message down the tree).

Proof of complexity

Lemma 4. *When the algorithm stops the biggest possible phase is bounded by* $\lfloor \log_2(n) \rfloor$

Proof. Whenever a king at phase t increases his phase, he annexes another king of phase t. Therefore, we have at most $\frac{n}{2}$ kings in phase 1, $\frac{n}{2^2}$ kings in phase 2 ... $\frac{n}{2^l}$ kings in phase l, for every $1 \leq l \leq \log_2(n)$ □

Theorem 2. *The number of messages sent is* $5n \log_2(n) + \mathcal{O}(n)$

Proof. Let's give upper bound for every type of messages

1. I_AM_THE_KING - exactly $n - 1$ messages
2. YOUR_CITIZEN - each node sends at most one such message - as a reply to an ASK message - per phase. The total number of such messages is bounded by $n \log_2(n)$
3. ASK - a king can send at most len(King.Vassals) + 1, therefore all the kings in this phase sent together at most n messages, the total number of messages sent by kings is bounded by $n \log_2(n)$, Every citizen transfers at most one ASK message per phase, the total number of such messages sent by all citizens is also bounded by $n \log_2(n)$
4. ACCEPT - Citizens send no more than ASK messages - bounded by $n \log_2(n)$, Kings send exactly $n - 1$ messages
5. UPDATE - one such message per phase - bounded by $n \log_2(n)$

So the total number of messages is bounded by $5n \log_2(n) + 2n$. This bound is pretty loose. The algorithm uses around 800 messages for n = 100 but the bound is equal to 3200. □

References

[1] Ephraim Korach, Shlomo Moran, Schmuel Zaks (1984) *Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of Processors*