Problema de la elección de líder

Cada nodo decide finalmente si es líder o no, sujeto a la restricción de que haya exactamente un líder.
Los nodos se encuentran en uno de estos tres estados: indeciso, líder, no líder. Inicialmente, cada nodo se encuentra en el estado indeciso. Al salir del estado indeciso, un nodo pasa a un estado final (líder o no líder).

OBSERVACIÓN. La posibilidad de elegir un líder en un sistema anónimo depende de si la red es simétrica (anillo, gráfica completa, gráfica bipartita completa, etc.) o asimétrica (estrella, nodo único con el grado más alto, etc.). Simplificando un poco, en este contexto una gráfica simétrica es una gráfica en el que la vecindad extendida de cada nodo tiene la misma estructura. La elección no uniforme de líderes anónimos en anillos síncronos es imposible. La idea es que en un anillo siempre se puede mantener la simetría.

LEMA 1. Después de la ronda k de cualquier algoritmo determinista en un anillo anónimo, cada nodo se encuentra en el mismo estado s_k.

Demostración. (Inducción) Todos los nodos comienzan en el mismo estado. Una ronda en un algoritmo síncrono consta de tres pasos: envío, recepción y cálculo local, por definición. Todos los nodos envían los mismos mensajes, reciben los mismos mensajes, realizan los mismos cálculos locales y, por lo tanto, terminan en el mismo estado.

□

TEOREMA 2: Elección del líder anónimo
La elección determinista del líder en un anillo anónimo es imposible.

Demostración. Si un nodo decide convertirse en líder (o no líder), entonces todos los demás nodos también lo hacen, contradiciendo la especificación del problema de la elección de líder para $n > 1$. Esto es válido para algoritmos no uniformes y, por tanto, también para algoritmos uniformes. Esto es válido para algoritmos no uniformes y, por tanto, también para algoritmos uniformes. Además, es válido para algoritmos síncronos y, por tanto, también para algoritmos asíncronos.

□

El sentido de la dirección es la capacidad de los nodos para distinguir a los nodos vecinos en un entorno anónimo. En un anillo, por ejemplo, un nodo puede distinguir al vecino que va en el sentido de las agujas del reloj y al que va en sentido contrario. El sentido de la dirección no ayuda en la elección anónima del líder.

El teorema también es válido para otras topologías de red simétricas (por ejemplo, gráficas completas, gráficas bipartitas completas, . . .).

El teorema generalmente no se cumple para algoritmos aleatorios; si se permite a los nodos lanzar una moneda, algunas simetrías pueden romperse.

Sin embargo, lo que es más sorprendente, la aleatorización no siempre ayuda. Un algoritmo anónimo uniforme aleatorio puede, por ejemplo, no elegir un líder en un anillo. La aleatorización

no ayuda a decidir si el anillo tiene $n = 3$ o $n = 6$ nodos: Uno de cada tres nodos puede generar los mismos bits aleatorios y, como resultado, los nodos no pueden distinguir los dos casos. Sin embargo, una aproximación de n que sea estrictamente mejor que un factor 2 ayudará.

OBSERVACIÓN. Los límites inferiores en computación distribuida suelen ser más fáciles que en el modelo centralizado estándar (memoria de acceso aleatorio, RAM) porque se puede discutir sobre los mensajes que hay que intercambiar. En esta sección presentamos un primer límite inferior.

Algorithm 5 Asynchronous leader election: code for processor p_i, $0 \leq i < n$.

Initially, *asleep* = true

```
1:  upon receiving no message:
2:      if asleep then
3:          asleep := false
4:          send ⟨probe,id,0,1⟩ to left and right

5:  upon receiving ⟨probe,j,k,d⟩ from left (resp., right):
6:      if j = id then terminate as the leader
7:      if j > id and d < 2^k then                          // forward the message
8:          send ⟨probe,j,k,d + 1⟩ to right (resp., left)   // increment hop counter
9:      if j > id and d ≥ 2^k then                          // reply to the message
10:         send ⟨reply,j,k⟩ to left (resp., right)
                                          // if j < id, message is swallowed

11: upon receiving ⟨reply,j,k⟩ from left (resp., right):
12:     if j ≠ id then send ⟨reply,j,k⟩ to right (resp., left)   // forward the reply
13:     else                                                     // reply is for own probe
14:         if already received ⟨reply,j,k⟩ from right (resp., left) then
15:             send ⟨probe,id,k + 1,1⟩                          // phase k winner
```

Algorithm 6 Synchronous leader election: code for processor p_i, $0 \leq i < n$.

Initially *waiting* is empty and *status* is asleep

```
1:  let R be the set of messages received in this computation event
2:  S := ∅                                          // the messages to be sent

3:  if status = asleep then
4:      if R is empty then                          // woke up spontaneously
5:          status := participating
6:          min := id
7:          add ⟨id,1⟩ to S                         // first phase message
8:      else
9:          status := relay
10:         min := ∞

9:  for each ⟨m,h⟩ in R do
10:     if m < min then
11:         become not elected
12:         min := m
13:         if (status = relay) and (h = 1) then    // m stays first phase
14:             add ⟨m,h⟩ to S
15:         else                                    // m is/becomes second phase
16:             add ⟨m,2⟩ to waiting tagged with current round number
17:     elseif m = id then become elected
                                                    // if m > min then message is swallowed

18: for each ⟨m,2⟩ in waiting do
19:     if ⟨m,2⟩ was received 2^m − 1 rounds ago then
20:         remove ⟨m⟩ from waiting and add to S

21: send S to left
```

- M. Raynal, Distributed Algorithms for Message-Passing Systems, Cap. 4
- H. Attiya, Distributed Computing, Cap. 3