Computación Distribuida

UNAM, Facultad De Ciencias.

Profesor. Fernando Michel Tavera
Ayudante. Mauricio Riva Palacio Orozco
Ayudante. Yael Antonio Calzada Martín

Tarea 2

Alumnas:
Góngora Ramírez Dania Paula (318128274)
Villafán Flores María Fernanda (318211767)

---

16 de marzo de 2023
Ciudad de México.

EJERCICIO 1. Sea $G = (\Pi, E)$ un sistema distribuido síncrono. Sea τ un árbol BFS con raíz en el proceso $p_s \in \Pi$. Demuestra que existe una ejecución del algoritmo para crear árboles generadores que tiene como resultado τ con p_s el proceso que inicia la ejecución. Hint: Recuerda que en un árbol BFS, un proceso a distancia d de la raíz G, está a distancia d en τ.

Respuesta.

Demostración por inducción fuerte sobre la distancia d de un proceso en el árbol a la raíz.

- *Caso base:* Para $d = 1$.

  Sabemos que p_s será la raíz del árbol desde el cuál se enviará el mensaje inicial a cualquier proceso p_i con $d = 1$, entonces p_i sólo recibirá ese único mensaje de p_s y en alguna ronda siguiente, volverá a enviar un mensaje de regreso por el cuál le avisará a p_s que lo tomará como padre (ya que sólo hay un proceso distinguido p_s). Lo que pasará es que para cualquier proceso a distancia $d = 1$, éste será parte del árbol BFS que estamos buscando.

  Por lo tanto, se cumple el caso base.

- *Hipótesis de Inducción:*

  Supongamos que para cualquier proceso $p_i \in \prod$ que esté a distancia d de la raíz, existe una ejecución del algoritmo en la que se haya construido el árbol τ BFS.

- *Paso inductivo:*

  Demostremos para cualquier proceso p a distancia $d + 1$.
  Tomemos p_i un proceso a distancia $d + 1$, entonces tenemos los siguientes casos:

  1) Si p_i recibe más de un mensaje de otros procesos y toma como padre a uno en donde la distancia de él a p_s es distinta de $d + 1$.

     En este caso, sabemos que esto puede ocurrir por el no-determinismo que hay al recibir más de un mensaje al mismo tiempo, entonces, ésta ejecución no se tendrá en cuenta (ya que no generará el árbol BFS que estamos buscando).

  2) Si p_i está en la misma situación del punto anterior, pero en este caso si tomamos a un proceso p_j como nuestro padre tal que se cumpla que la distancia de p_i a p_s sea $d+1$, entonces por *Hipótesis de inducción*, sabemos que se va a cumplir que para todos los procesos a distancia d ya estará construido el árbol y como p_i también estará a distancia $d + 1$, éste será el árbol BFS que estábamos buscando.

     Por lo tanto, se cumple el paso inductivo.

  Por lo tanto, se cumple que existe una ejecución del algoritmo para crear árboles generadores tal que el árbol resultante τ sea un árbol BFS.

■

EJERCICIO 2. Sea $G = (\Pi, E)$ un sistema distribuido asíncrono. Considera un algoritmo para construir un árbol BFS en el que sólo la raíz sabe que ya terminó la construcción del árbol. Diseña un algoritmo que le permita a cada proceso saber que ya terminó la construcción del árbol y que la raíz sepa que ya todos saben que se construyó el árbol. Argumentar por qué es correcto y la complejidad del número de mensajes y tiempo.

Respuesta.

Como la raíz ya sabe cuándo se acabó la construcción del árbol, entonces podemos basarnos en la misma estructura para diseminar el mensaje de que ya se acabó la construcción e igualmente para que cada proceso le mande su información a la raíz.
Utilizaremos modificaciones de los algoritmos de *Broadcast* y *Convergecast*.

BROADCAST–CON–CONVERGECAST

```
 1: Initially do
 2: begin:
 3:     if p_s = p_i then
 4:         data = finished ; // Indicamos que ya se acabo la construccion
 5:         parent_i = i ;
 6:         expected_mssg_i = |neighbors_i| ;
 7:         for each j ∈ neighbors_i do
 8:             send GO(data) to p_j ;
 9:         end for
10:     else
11:         parent_i = 0 ;
12:         data = ∅ ;
13:     end if
14:     children_i = 0 ;
15: end

16: when GO(data) is received from p_j do
17: begin:
18:     if parent_i = 0 then
19:         parent_i = j ;
20:         expected_mssg_i = |neighbors_i| − 1 ;
21:         if expected_mssg_i = 0 then
22:             v_i = "Recibido" ; // Valor que se enviara
23:             send BACK((i, v_i)) to p_j ;
24:         else
25:             for each k ∈ neighbors_i − {j} do
26:                 send GO(data) to p_k ;
27:             end for
28:         end if
29:     else
30:         send BACK(0) to p_j ;
```

```
31:     end if
32: end

33: when BACK(val_set) is received from p_j do
34: begin:
35:     expected_mssg_i = expected_mssg_i − 1 ;
36:     if val_set ≠ 0 then
37:         children_i = children_i ∪ {j} ;
38:     end if
39:     if expected_mssg_i = 0 then
40:         v_i = "Recibido" ; // Valor que se enviara
41:         val_set_i = ⋃_{x∈children_i} val_set_x ∪ {(i, v_i)} ;
42:         if parent_i ≠ i then
43:             send BACK((i, v_i)) to parent_i ;
44:         else
45:             the root p_s can compute f(val_set_i) ;
46:         end if
47:     end if
48: end
```

- ***Los algoritmos son correctos***

  Sabemos que estos algoritmos basados en *Broadcast* y *Convergecast* son correctos ya que (aunque el sistema sea asíncrono) siempre se terminará de mandar y recibir los mensajes en alguna ronda, y entonces se habrá diseminado el mensaje a todos los procesos dentro del árbol de que ya se terminó la construcción, además de que este árbol es conexo y por tanto, para la parte del *convergecast*, llegará un momento en el cuál hayamos recogido los mensajes de todos los demás procesos y finalmente la raíz podrá computar que todos ya saben que se construyó el árbol.

- ***Complejidad***

  - *Número de mensajes:*

    Debemos revisar cuántos mensajes se envían para la parte de *broadcast* y para la parte de *convergecast*.
    Como sabemos que es un árbol en donde se hará el envío de mensajes, simplemente serán $p-1$ mensajes para *broadcast* y de la misma forma serán $p-1$ mensajes para *convergecast*, por lo que la complejidad del número de mensajes será:

    $$O(2(p-1)) = O(2p-2) = O(2p), \text{ con } p \text{ el número de procesos}$$

  - *Complejidad en tiempo:*

    Podemos darnos cuenta que haremos dos vueltas a la estructura de la gráfica, una para diseminar el mensaje de la raíz a los demás procesos, y otra de regreso de las hojas hasta la raíz, además hay que tener en cuenta que podría haber un retraso debido a que es un sistema asıíncrono, por lo que la complejidad en tiempo será:

$O(2p - 2 + r)$, con r la constante de la suma de los retardos de todo el sistema

■

EJERCICIO 3. Sea $G = (\Pi, E)$ un sistema distribuido síncrono. Sea $M = \{m_1, m_2, \ldots, m_k\}$ un conjunto de mensajes que se quieren transmitir desde un proceso p_s hacia todos los demás. Debido a que el ancho de banda es muy limitado, no se puede enviar el conjunto M completo, por lo que tiene que enviarse cada m_i por separado. Cada proceso debe saber qué parte del mensaje está recibiendo, es decir que el i-ésimo mensaje es el m_i. Diseña un algoritmo de broadcast para diseminar M sin que se repita la recepción de los mensajes en los procesos, es decir, si p_i ya recibió m_j, no debe volver a recibirlo. Hint: broadcast sobre un árbol toma $O(n)$ mensajes porque no se repite la entrega de los mensajes.

Respuesta.

```
1:  Initially do
2:  begin:
3:      if p_s = p_i then
4:          data = [m_1, m_2, ..., m_k] ; // Arreglo de mensajes
5:          for each j ∈ children_i do
6:              for each m ∈ data do
7:                  send GO(m, position_m) to p_j ;
8:              end for
9:          end for
10:     else
11:         data = ∅ ;
12:     end if
13:  end

14: when GO(message, pos) is received from p_j
15: begin:
16:     if pos == 1 then
17:         data = [mensaje, ∅, ..., ∅] ;
18:     else
19:         data[pos] = mensaje ;
20:     end if

21:     for each k ∈ children_i do
22:         send GO(mensaje, pos) to p_k ;
23:     end for
24: end
```

■

Ejercicio 4. Sea $G = (\Pi, E)$ un sistema distribuido síncrono y sea $p_s \in \Pi$ un proceso que empezará a ejecutar broadcast. Demuestra que broadcast no puede ejecutarse en menos de D rondas, con D la distancia más grande entre p_s y cualquier otro proceso.

Respuesta.

Demostración por contradicción.

Supongamos que *broadcast* puede ejecutarse en menos de D rondas, con D la distancia más grande entre p_s y cualquier otro proceso.

Si ocurre esto, entonces podemos tomar cualquier proceso p_i dentro de G tal que $d = D$ y decir que p_i recibirá su mensaje en la ronda $t = D - 1$, sin embargo, sabemos que trabajamos en un sistema síncrono, por lo que se enviará el mensaje desde la raíz hacia sus hijos y estos a su vez a sus demás hijos, y como es un sistema síncrono, entonces cada ronda que avance habrá llegado un mensaje a un proceso con distancia $d = t$.

De ésta forma nuestra suposición original de que p_i haya recibido el mensaje *broadcast* la ronda anterior $t = D - 1$ es una contradicción (ya que vimos que cualquier proceso recibirá su mensaje en la ronda $t = d$).

Por lo tanto, p_i recibirá su mensaje en la ronda $t = D$, por lo que *broadcast* no puede ejecutarse en menos de D rondas en un sistema distribuido síncrono.

■

EJERCICIO 5. En la CDMX varios puntos de control (nodos) fueron agregados. Cada uno de estos puntos tiene información sobre el número de personas que viven dentro de cierto diámetro. Un punto (raíz) de la delegación Benito Juárez quiere recolectar la información de todos los puntos de control y determinar cuántas personas viven en la CDMX. Escribe un algoritmo distribuido para que cada nodo recolecte la información de sus hijos para enviársela al padre sin repetirla y que el padre reporte el total de pobladores viviendo en la CDMX.

Respuesta.

```
 1: Initially do
 2: begin:
 3:     if p_s = p_i then
 4:         parent_i = i ; expected_mssg_i = |neighbors_i| ;
 5:         v_i = informacion del punto_i sobre la cantidad de personas que viven
 6:             dentro del diametro ;
 7:         for each j ∈ neighbors_i do
 8:             send GO(data) to p_j ;
 9:         end for
10:     else
11:         parent_i = 0 ;
12:     end if
13:     children_i = 0 ;
14: end

15: when GO(data) is received from p_j do
16: begin:
17:     if parent_i = 0 then
18:         parent_i = j ; expected_mssg_i = |neighbors_i| − 1 ;
19:         v_i = informacion del punto_i sobre la cantidad de personas que viven
20:             dentro del diametro ;
21:         if expected_mssg_i = 0 then
22:             send BACK((i, v_i)) to p_j ;
23:         else
24:             for each k ∈ neighbors_i − {j} do
25:                 send GO(data) to p_k ;
26:             end for
27:         end if
28:     else
29:         send BACK(0) to p_j ;
30:     end if
31: end

32: when BACK(val_set) is received from p_j do
33: begin:
```

```
34:     expected_mssg_i = expected_mssg_i − 1 ;
35:     if val_set ≠ 0 then
36:         children_i = children_i ∪ {j} ;
37:     end if
38:     v_i = informacion del punto_i sobre la cantidad de personas que viven
39:             dentro del diametro ;
40:     if expected_mssg_i = 0 then
41:         val_set_i = ⋃_{x∈children_i} val_set_x ∪ {(i, v_i)} ;
42:         if parent_i ≠ i then
43:             send BACK((i, v_i)) to parent_i ;
44:         else
45:             the root p_s can compute SUMA(val_set_i) ;
46:         end if
47:     end if
48: end
```

■

EJERCICIO 6. Considera el algoritmo de convergecast sobre el árbol generador ya construido. Demuestra que cualquier nodo a altura h (distancia más corta desde la hoja hacia el nodo) envía un mensaje a más tardar en la ronda h.

Respuesta.

Supongamos que tenemos un árbol generador ya construido, en el cuál aplicaremos el algoritmo de *convergecast*.

Sea un nodo p a una altura h en el árbol.
En la primera ronda, p recibe los mensajes de sus hijos, y en la segunda ronda, p recibe los mensajes de los hijos de sus hijos, entonces, en la ronda i, p recibirá los mensajes de los nodos que están a una distancia i desde p.
Como p está a una altura h, todos los nodos que están a una distancia menor o igual de h desde p, son las hojas del subárbol que tiene como raíz a p. De esta forma, en la ronda h, p recibirá los mensajes de todas las hojas del subárbol en el cuál su raíz es p (ya que todas las hojas del subárbol están a una distancia menor o igual a h), entonces, p recibirá todos los mensajes que debe recibir en a lo más la ronda h.

Por lo tanto, cualquier nodo a altura h (distancia más corta desde la hoja hacia el nodo) envía un mensaje a más tardar en la ronda h.

■

Ejercicio 7. Explica por qué el algoritmo de convergecast con broadcast puede producir más de un árbol.

Respuesta.

El algoritmo de *convergecast con broadcast* puede producir más de un árbol porque la elección de un padre no es determinista y puede haber múltiples caminos con el mismo costo mínimo.

Tenemos situaciones en las que varios procesos pueden enviar su mensaje a diferentes nodos padres, por ejemplo, si tres nodos A, B y C tienen el mismo costo para llegar a un nodo D, es posible que A envíe su mensaje a B, B envíe su mensaje a C y C envíe su mensaje a D. En este caso, se producirían dos árboles diferentes:

$$\text{Uno con } A \text{ como raíz} \quad \text{y} \quad \text{otro con } B \text{ como raíz}$$

Otra problematica que puede surgir es que cuando dos o más nodos envían su mensaje a un nodo que ya ha seleccionado a su padre, lo que puede generar una nueva raíz del árbol.

■

Ejercicios EXTRAS.

EJERCICIO 3. Consideremos un sistema distribuido $G = (\Pi, E)$ representado como un árbol. Supongamos que cada proceso p_i tiene un identificador único, pero ningún proceso sabe quién es la raíz del árbol. Diseña un algoritmo distribuido que permita a cada proceso determinar quién es la raíz del árbol.

Respuesta.

```
 1: Initially do
 2: begin:
 3:     parent_i = null ; // Inicializamos el padre como nulo
 4:     id = ID(p_i) ;
 5:     data = (id, parent_i) ;
 6:     val_set = {} ; // Conjunto de valores a enviar en el convergecast

 7:     if children_i = {} then // Si soy hoja, envio mi valor al padre
 8:         val_set = {data} ;
 9:         send_val = val_set ;
10:     else
11:         for each j ∈ children_i do
12:             send GO(data) to p_j ;
13:         end for
14:     end if
15:     while |val_set| ≠ |children_i| do
16:         receive data from p_j ;
17:         val_set = val_set ∪ {data} ;
18:     end while
19:     send_val = val_set ∪ {data} ;

20:     if parent_i = id then // Si el padre es el mismo es la raiz
21:         parent_i = "root" ;
22:     end if
23:     send_val = val_set ∪ {(id, parent_i)} ;
24:     receive_val = null ;
25:     if id = root then // Si soy la raiz, difundo el mensaje
26:         send_val = "root" ; // Indicamos que somos la raiz
27:         for each j ∈ children_i do
28:             send GO(data) to p_j ;
29:         end for
30:     else
31:         while receive_val ≠ "root" do
32:           receive data from parent_i ;
33:           receive_val = data ;
34:         end while
```

```
35:     end if
36: end
```

■

EJERCICIO 5. Considere un sistema distribuido $G = (\Pi, E)$ representado como un árbol. Diseña un algoritmo distribuido que determine la altura del árbol, es decir, la longitud del camino más largo desde la raíz a cualquier hoja del árbol.

OBSERVACIÓN. Se puede suponer que cada proceso p_i conoce su proceso padre en el árbol, y el proceso raíz conoce su identificador único.

Respuesta.

```
1: Initially do
2: begin:
3:     if p_s = p_i then
4:         distance = 0 ; // Altura del nodo distinguido a el mismo es 0
5:         for each j ∈ children_i do
6:             send GO(distance + 1) to p_j ;
7:         end for
8:     else
9:         distance = ∞ ;
10:    end if
11: end

12: when GO(data) is received from p_j do
13: begin:
14:    if data + 1 < distance then
15:        distance = data + 1 ;
16:        for each k ∈ children_i do
17:            send GO(distance) to p_k ;
18:        end for
19:    end if
20: end
```

■