UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

COMPUTACIÓN DISTRIBUIDA

TAREA 2

Integrantes:

Lopez Espinoza Ashley Yael

Santiago Ivan Reyes Medina

EJERCICIO 1. Sea *G* = (Π, *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 gene radores 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*.

Para demostrar que existe una ejecución del algoritmo para construir árboles generadores que genera un árbol BFS. primero definiremos las propiedades de un árbol BFS.

- Propiedad del BFS: Si un nodo v está a distancia d de la raíz en el grafo G, entonces en el árbol τ resultante también estará a distancia d de la raíz.

Esto significa que, en un árbol BFS, los nodos están organizados en niveles de acuerdo con su distancia a la raíz ps Para que el algoritmo de construcción de árboles generadores produzca un árbol BFS, debe garantizar que cada nodo se conecte con el primer proceso desde el que recibe el mensaje de creación del árbol.

Luego el Algoritmo 5 describe cómo se construye un árbol generador en un sistema distribuido. Este se basa en el envío de mensajes `GO()` para propagar la estructura del árbol y `BACK()` para completar el proceso.

Así que para demostrar que el algoritmo genera un BFS, debemos verificar que cada nodo en el árbol resultante mantiene la misma distancia desde la raíz que tenía en el grafo G

Como el sistema es síncrono, todos los procesos ejecutan sus acciones en rondas.

-En la ronda 0, la raíz ps envía `GO()` a sus vecinos.

-En la ronda 1, los procesos vecinos de ps reciben `GO()`, marcan a ps como su padre y envían `GO()` a sus vecinos.

-En la ronda 2, los procesos que estaban a distancia 2 de ps reciben `GO()` por primera vez, eligen a su padre y continúan propagando.

Este proceso continúa de forma nivel por nivel, lo que nos asegura que cada nodo elige como padre al primer proceso que le envió GO(), como la propagación ocurre en rondas sucesivas, el primer GO() recibido por cada nodo viene del nodo más cercano a la raíz

Asi la distancia de cada nodo desde la raíz en el árbol construido es la misma que en el grafo original G lo que garantiza que el árbol es un BFS

EJERCICIO 2. Considere un sistema distribuido $G = (\Pi, E)$ representado como un árbol. Dise ña 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.

Como nosotros podemos suponer que cada proceso pi conoce su proceso padre en el árbol y el proceso ráiz conoce su identificador unico podemos crear el algoritmo modificando convergecast ya que este algoritmo cumple con la observación:

cada hoja inicia su altura con 0 y se la manda a su padre

Un nodo intermedio espera a recibir la altura de todos sus hijos y cuando las haya recibido suma +1 al máximo de eas y envia este valor a su padre La raíz recibirá la altura máxima de todas las ramas del árbol y las almacenará como su altura máxima del árbol

Este se ve de la siguiente manera:

Initially do:

Begin:

vi

if pchildreni == 0 then:

sendBack(0) to parenti ← cada hoja envia su altura cero a su padre

enf if

end

Cuando un proceso recibe Back(0) (altura)

When back(hijo_altura) is received from pj such that pj E childreni do:

Begin:

recivied_heights = recivied_heights U {hijo_altura}

if received_heights contiene alturas de todos los hijos de pi then:

altura_pi = max(received_heights) + 1

if altura_pi es la raiz then:

altura_arbol = altura_pi

else:

send_back(altura_pi) a parenti

end if

endif

end:

EJERCICIO 3. 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.

1: Construcción del árbol BFS

Se ejecuta un algoritmo para construir el árbol, como el `Algorithm 5` de las notas.

2: Confirmación de Finalización (Convergecast)

Cada nodo envía un mensaje de confirmación (`ACK`) a su padre una vez que ha recibido confirmaciones de todos sus hijos. Cuando la raíz recibe confirmaciones de todos sus hijos, envía un mensaje de finalización (`DONE`) a toda la red mediante difusión inversa.

Los nodos retransmiten el mensaje `DONE` a sus hijos.

3.Finalización

Un nodo sabe que el árbol ha sido completamente construido cuando recibe `DONE`. La raíz sabe que todos han sido informados cuando recibe confirmaciones de todos los nodos.

Algorithm : Construcción y Confirmación de Árbol BFS

```
Initially do

begin:

if ps = pi then  # Si pi es la raíz

    parenti = i

    expected_mssgi = |neighborsi|

    for each j ∈ neighborsi do

        send GO() to pj

else

    parenti = 0

end if

childreni = 0

end


Recepción de GO() para construir el árbol

when GO() is received from pj do

begin:

if parenti = 0 then
```

```
    parenti = j
    expected_mssgi = |neighborsi| - 1
    if expected_mssgi = 0 then
        send BACK(ACK) to pj
    else
        for each k ∈ neighborsi - {j} do
            send GO() to pk
end if
else
    send BACK(0) to pj
end if
end

Recepción de BACK() para confirmar construcción
when BACK(ACK) is received from pj do
begin:
expected_mssgi = expected_mssgi - 1
if expected_mssgi = 0 then
    if parenti ≠ i then
        send BACK(ACK) to parenti
    else
        send DONE() to all childreni
end if
end

Propagación de DONE() para notificación de finalización
when DONE() is received from parenti do
begin:
send DONE() to all childreni
end
```

EJERCICIO 4. Consideremos un sistema distribuido *G* = (Π, *E*) representado como un árbol. Supongamos que cada proceso p_i tiene un identificador único. Diseña un algoritmo distribuido para determinar el proceso más alejado de la raíz del árbol y la distancia de la raíz a ese proceso.

OBSERVACIÓN. Puede asumir que el proceso raíz conoce su identificador único, y cada proce so p_i conoce su proceso padre en el árbol.

Similar al ejercicio 2, usaremos y modificaremos el algoritmo converge cast ya que este algoritmo cumple que cada proceso pi conoce a su padre y a sus hijos en el árbol. Modificando el algoritmo

Las hojas inician el proceso enviando su propia identificación y una distancia de 1 a su padre.

Cada proceso pi, al recibir mensajes de sus hijos, determina cuál de ellos está más alejado y envía esa información a su propio padre.

La raíz recibe la información de todos sus hijos y determina el nodo más alejado y la distancia.

Initially do:

```
if childreni = 0 then     ← quiere decir que es una hoja
    send ((i, 0)) to  parenti
  end if


 when BACK((id_max, dist_max)) is received from pj do:
begin:
       received_messages = +=1
       if dist_max  +1 >  max_dist + 1
                farthest_node = id_max
        end if


  if received_messages = expected_messgi then:
     if parenti != i then:
       sendBack((farthest_node, max_distance)) to parenti
     else:
        print ("El proceso más alejado es",farthest_node, "con distancia",  max_distance)
   end if
end
```

EJERCICIO 5. 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 comple to, 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.

Usando el algoritmo broadcast:

Primero usamos el **Algorithm 5** para formar el árbol. Despues transmitimos cada mi individualmente por el árbol. Cada proceso lleva un registro de los mensajes ya recibidos y solo reenvía los nuevos.

```
1: Initially do

2: begin:

3: if ps = pi then        # Si soy la raíz

4:     parenti = i

5:     childreni = {nodos conectados a pi en el árbol}

6:     for each m_i ∈ M do

7:         for each j ∈ childreni do

8:             send GO(m_i, i) to pj

9:     end for

10: end if

11: received_messages = ∅  # Conjunto de mensajes recibidos

12: end

13: when GO(m_i, sender) is received from pj do

14: begin:

15: if m_i ∉ received_messages then
```

16: received_messages = received_messages ∪ {m_i}

17: for each k ∈ childreni do

18: send GO(m_i, i) to pk

19: end for

20: end if

21: end

EJERCICIO 6. Sea *G* = (Π, *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.

Usaremos inducción sobre la distancia Para demostrar que el broadcast no puede ejecutarse en menos de D rondas, con D como la distancia más grande entre ps y cualquier otro proceso en un sistema distribuido síncrono.

Demostración por Inducción sobre d(ps,v)

Caso Base:

Para el proceso ps(la raíz), se tiene que d(ps,ps)=0, y el mensaje ya está recibido en la ronda 0. Esto es inmediato.

H.I:

Supongamos que cualquier proceso v con d(ps,v)=k recibe el mensaje a más tardar en la ronda k

Paso inductivo:

Queremos demostrar que cualquier proceso u con d(ps,u)=k+1 recibe el mensaje a más tardar en la ronda k+1

Por la hipótesis de inducción, cualquier proceso v con d(ps,v)=k ha recibido el mensaje en la ronda k

Como el algoritmo de difusión nos dice que un nodo sólo reenvía el mensaje a sus vecinos una vez que lo recibe, enviará M a todos sus vecinos en la ronda k.

Así el nodo u, que es vecino de v y tiene d(ps,u)=k+1, recibirá el mensaje en la siguiente ronda, es decir, en la ronda k+1

Así aplicando la inducción hasta el nodo más alejado de ps, que se encuentra a una distancia D, este recibirá el mensaje exactamente en la ronda D

Por lo tanto, el broadcast no puede ejecutarse en menos de D rondas, ya que cada nodo en la ruta más larga necesita al menos una ronda adicional para recibir el mensaje.

EJERCICIO 7. 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 diáme tro. 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.

Para resolver este problema, utilizaremos convergecast sobre un árbol generador, ya que cada nodo debe recolectar la información de sus hijos y enviarla a su padre hasta que llegue a la raíz, que calculará el total de personas en la CDMX.

initially

begin:

vi = poblacion en el nodo i

if childreni == 0: then: ← esto quiere decir que es un hoja

BACK((i, vi)) to parent i

end if

end

when BACK(data) is received from pj such that j E childreni do:

begin:

val_seti = ∪ j ∈ childreni val_setj ∪ {(i, vi)} # Acumular datos de los hijos

if parenti != i then : # Si no es la raíz

send BACK(val_seti) to parenti # Enviar datos al padre

else:

total_poblation = sum(vi for (_, vi) in val_seti) # Calcular total

report total_poblation # Reportar el total de habitantes en la CDMX

end if

end

EJERCICIO 8. 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*.

Casos base:

Los nodos que están a altura 0 son las hojas del árbol generador.

Según el Algoritmo 3: Convergecast, cada hoja v inmediatamente envía su mensaje al nodo padre en la primera ronda, es decir, en la ronda 0.

Por lo tanto, la propiedad es válida para h=0

Paso Inductivo

Supongamos que cualquier nodo a altura h envía su mensaje a más tardar en la ronda h se cumple para todos los nodos a altura h

Ahora sea un nodo u a altura h+1. Este nodo recibe mensajes de todos sus hijos, que tienen altura h.

Según la hipótesis de inducción, cada hijo de u enviará su mensaje a más tardar en la ronda h.

En el algoritmo, cuando un nodo u recibe mensajes de todos sus hijos, entonces lo envía a su padre.

Como el nodo u está a altura h+1 recibe todos los mensajes de sus hijos a más tardar en la ronda h

En la siguiente ronda (ronda h+1), u enviará el mensaje a su padre.

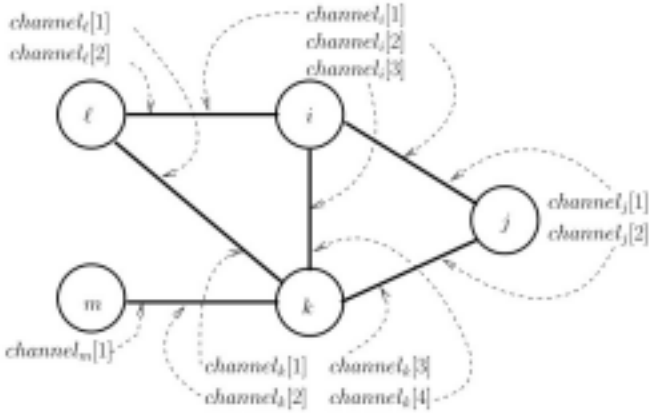
Por lo tanto, cualquier nodo a altura h+1 enviará su mensaje a más tardar en la ronda h+1.

Asi hemos demostrado que cualquier nodo a altura h envía su mensaje a más tardar en la ronda h.

EJERCICIO 9. Consideremos un gráfica de comunicación en el que los procesos no tienen iden tidad y cada proceso p_i conoce su posición en la red de comunicación con la ayuda de una matriz local $channel_i[1 \ldots c_i]$ (donde c_i es el número de vecinos de p_i). En la siguiente ima gen se muestra un ejemplo. Como podemos ver en la imagen, el canal que conecta p_i y p_j es localmente conocido por p_i como $channel_i[2]$ y localmente conocido por p_j como $channel_j[1]$. Utilizando el algoritmo DFS en el que un proceso distinguido

recibe un mensaje START(), di señe un algoritmo que asocie a cada proceso p_i una identidad id_i y una tabla de identidad $neighbor\_name_i[1 \ldots c_i]$ tal que:

- $\forall i : id_i \in \{1, 2, \ldots, n\}$
- $\forall i, j : id_i \neq id_j$
- Si p_i y p_j son vecinos y el canal que los conecta se denota $channel_i[k]$ en p_i y $channel_j[l]$ en p_j, y tenemos que $neighbor\_name_i[k] = id_j$ y $neighbor\_name_j[l] = id_i$.



1: when START() is received by process p_0 do

2: $id_0 \leftarrow 1$ # El primer proceso recibe la identidad 1

3: visited $\leftarrow \{p_0\}$ # Conjunto de procesos visitados

4: DFS(p_0, id_0, visited)

5: procedure DFS(pi, idi, visited)

6: for each neighbor channeli[k] of pi do

7: if neighbor pj not in visited then

8: idj $\leftarrow$ max(visited) + 1 # Asignar el siguiente ID disponible

9: visited $\leftarrow$ visited $\cup$ {pj}

10: neighbor_namei[k] $\leftarrow$ idj # Registrar el ID del vecino

11: neighbor_namej[l] ← idi # Registrar el ID en el otro proceso

12: DFS(pj, idj, visited)

13: end if

14: end for

15: end procedure

EJERCICIO 10. Consideremos el caso de una gráfica de comunicación dirigida donde el signi ficado de 'dirigida' es el siguiente. Un canal de p_i a p_j permite

- **Que p_i solo envíe mensajes *GO*() a p_j**
- **Que p_j solo envíe mensajes *BACK*() a p_i**

Dos procesos p_i y p_j son entonces de tal manera que o bien no hay un canal de comunicación que los conecte, o hay un canal de comunicación dirigido que conecta a uno con el otro, o hay dos canales de comunicación dirigidos (uno en cada dirección). Diseña un algoritmo distribuido que construya un árbol generador con una raíz distinguida p_a y compáralo con el algoritmo BFS. Se asume que hay un camino dirigido desde la raíz distinguida p_a a cualquier otro proceso.

Algoritmo: Construcción del Árbol Generador Dirigido

1. Inicialización en la raíz pa:

   La raíz pa se define a sí misma como su propio padre.

   Envía un mensaje GO() a todos sus procesos vecinos.

2. Recepción del mensaje GO() en un proceso pi:

   Si pi recibe un GO() por primera vez:

   Establece como padre al proceso pj del cual recibió el mensaje.

Envía GO() a todos sus vecinos, excepto al proceso que lo definió como su padre.

Si pi recibe un GO() pero ya tiene un padre, descarta el mensaje.

3. Propagación del mensaje GO():

   Cada proceso repite este procedimiento, estableciendo un único padre y enviando GO() a sus vecinos no visitados.

   Se repite hasta que todos los procesos alcanzables desde pa han sido visitados.

4. Confirmación y finalización (mensaje BACK()):

   Cuando un proceso pi ha recibido respuestas de todos los procesos a los que envió GO(), envía un mensaje BACK() a su padre.

   El mensaje BACK() se propaga hacia la raíz pa, confirmando que el árbol está construido.

5. Finalización en la raíz pa:

   La raíz pa recibe todos los mensajes BACK() de sus hijos y completa la construcción del árbol.

El BFS siempre genera el árbol de mínima profundidad, lo que es ideal para aplicaciones donde se necesita una transmisión eficiente.

Mi algoritmo sólo garantiza un árbol generador dirigido, pero sin asegurar la menor profundidad, lo que puede hacer que algunos nodos tarden más en recibir la información.

EJERCICIO 11. Considera el algoritmo BFS que no detecta terminación en un sistema síncrono (Figura 3). Sea D la distancia más grande de la raíz a cualquier otro proceso. Demuestra que para cada $1 \leq t \leq D$, después de t rondas, cada vértice p_i a distancia t ya ha recibido un mensaje con $d = t - 1$ de algún vecino p_j y por lo tanto $distance_i = t$ y $parent_i = j$ tal que $distance_j = t - 1$.

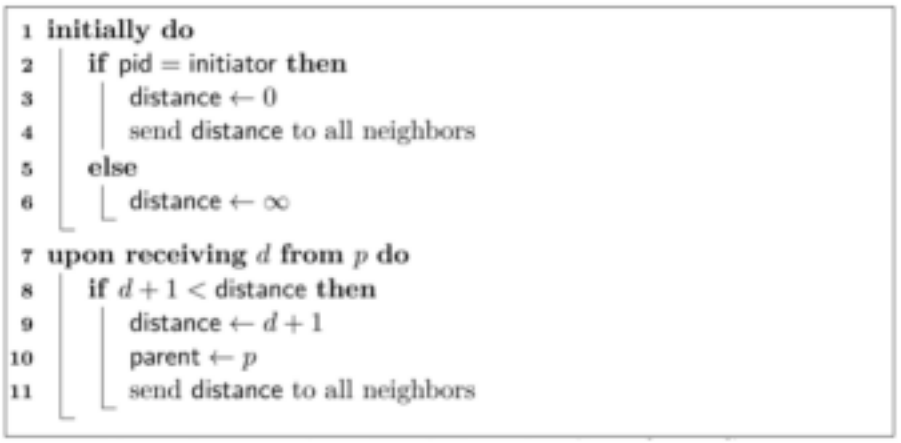


Figura 3: Algoritmo BFS que no detecta terminación

HINT. En la ronda 0 la raíz comienza su ejecución y envía su mensaje a sus vecinos y estos los reciben en la ronda 1.

Demostración del Algoritmo BFS en un Sistema Síncrono

Base de inducción (t=1)

De acuerdo con el Hint, la raíz comienza su ejecución en la ronda 0 y envía su mensaje a todos sus vecinos.

En la ronda 0, la raíz pase asigna:
distance=0 y envía distance=0 a todos sus vecinos.

En la ronda 1, los vecinos directos de la raíz pj reciben el mensaje y actualizan su distancia:
distancej=1

Además, establecen a la raíz como su padre:
parentj=pa

Hipótesis inductiva:

Supongamos que después de t rondas, todos los nodos a distancia t de la raíz ya han recibido un mensaje con d=t−1 de algún vecino pj por lo que:

distancei=t,parenti=j donde j es un nodo a distancia t−1

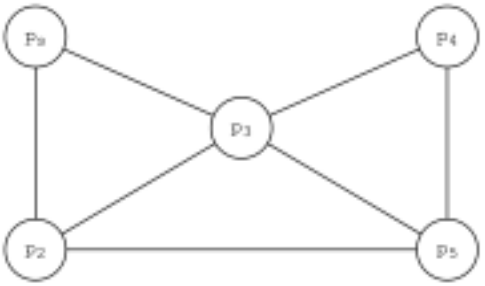
Paso inductivo:

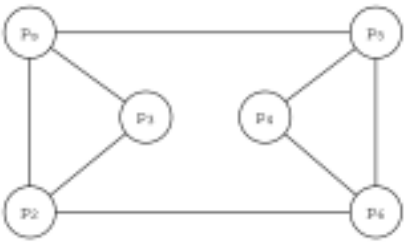
En la ronda t+1, cada nodo pi a distancia t ya ha enviado su distancia a todos sus vecinos.

Un nodo pk que está a distancia t+1 recibe por primera vez un mensaje con d=t lo que significa que: distancek=t+1 y establece como padre al nodo desde el cual recibió ese mensaje: parentk=i donde i es un nodo a distancia t

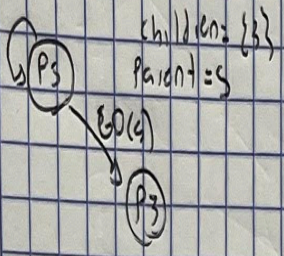
Así, se cumple que después de t rondas, cualquier nodo a distancia t ha recibido un mensaje con d=t−1, completando la demostración.

EJERCICIO 12. Ejecuta el algoritmo DFS en las siguientes gráficas.

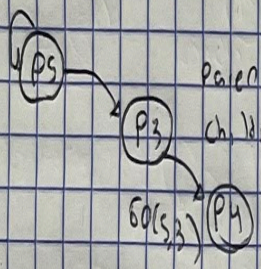




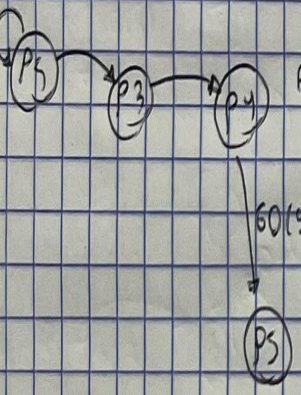
Ronda 1:



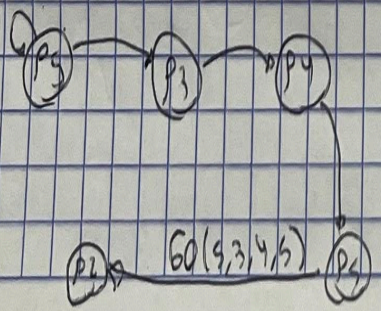
Ronda 2:



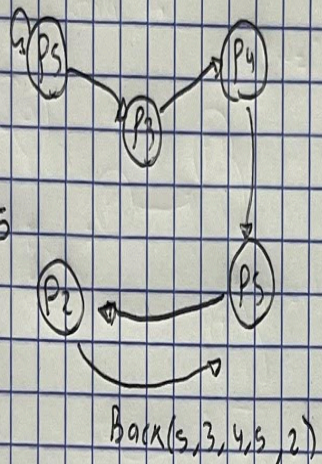
Ronda 3:



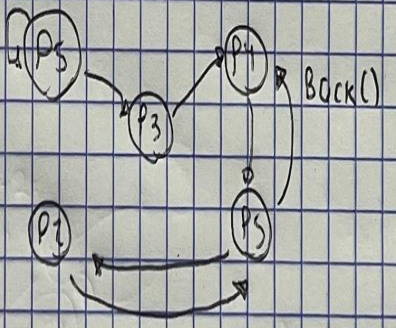
Ronda 4:



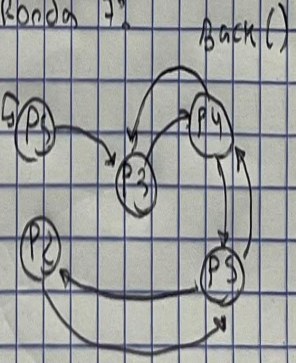
Ronda 5:



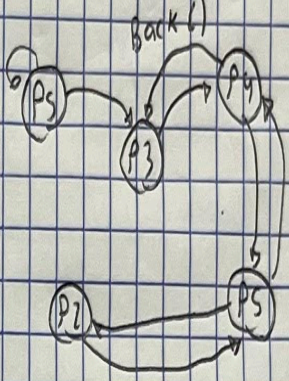
Ronda 6:



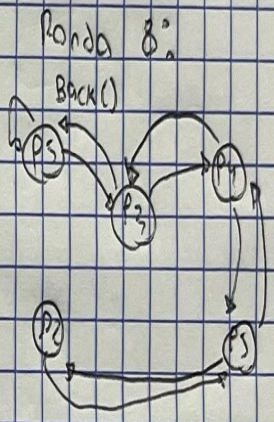
Ronda 7:



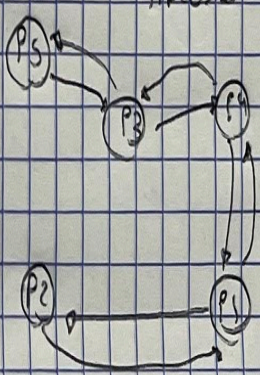
Ronda 8:



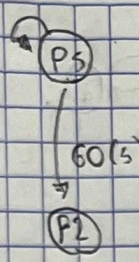
Ronda 8:



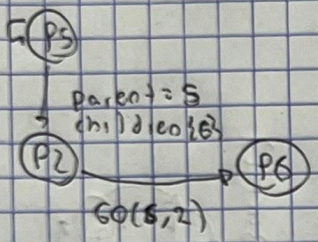
Ronda 9:



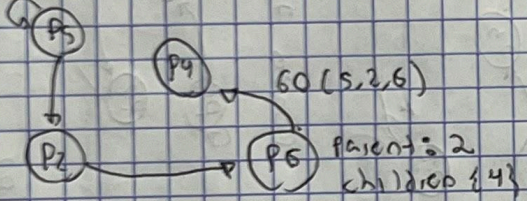
Ronda 1°:



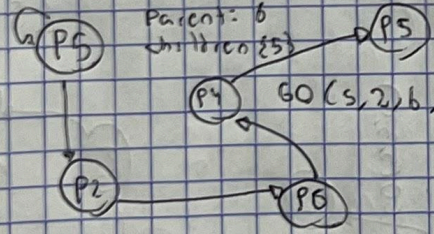
Ronda 2°:



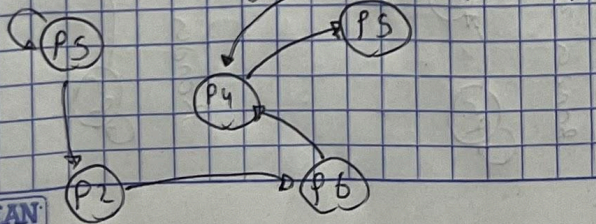
Ronda 3°:



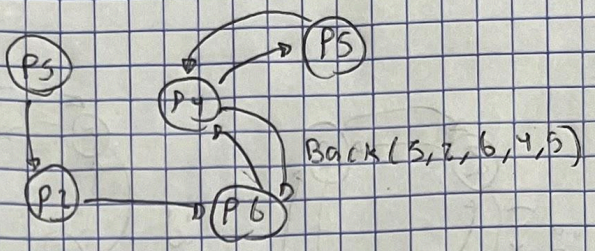
Ronda 4°:



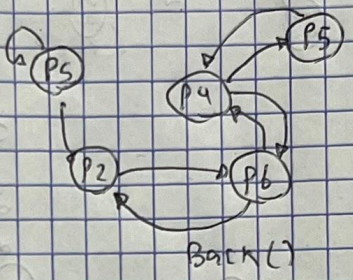
Ronda 5°:



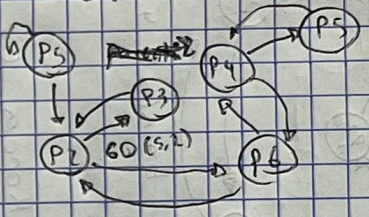
Ronda 6°



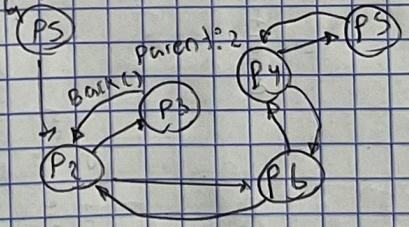
Ronda 7°:



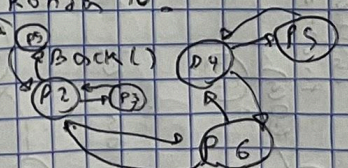
Ronda 8°:



Ronda 9°:



Ronda 10°:



Ronda 11°: Terminan el proceso

EJERCICIO 13. Consideremos una red lineal, es decir, una colección lineal de *n* procesos 1, . . . , *n*, donde cada proceso está conectado bidireccionalmente con sus vecinos. Supongamos que cada proceso *i* puede distinguir su izquierda de su derecha y sabe si es o no un punto final. Supongamos que cada proceso *i* tiene inicialmente un valor entero muy grande v_i y que pue de mantener en memoria sólo un número constante de tales valores en cualquier momento. Diseñe un algoritmo para ordenar los valores entre los procesos, es decir, para hacer que cada proceso *i* devuelva un valor de salida o_i, donde el subconjunto de salidas es igual al subcon junto de entradas y $o_1 \leq \cdot \cdot \cdot \leq o_n$. Intenta diseñar el algoritmo más eficiente que puedas tanto en número de mensajes como en número de rondas. Demuestra tus afirmaciones.

```
1: Initially do
2: begin:
3: if i = 1 then
4:     neighbor_left = None
5:     neighbor_right = i + 1
6: elif i = n then
7:     neighbor_left = i - 1
8:     neighbor_right = None
9: else
10:     neighbor_left = i - 1
11:     neighbor_right = i + 1
12: end if
13: value_i = vi
14: end


15: for round = 1 to n do
16: begin:
17:     if round mod 2 = 0 then
18:         if i mod 2 = 1 and neighbor_right ≠ None then
19:             send value_i to neighbor_right
20:             receive value_right from neighbor_right
21:             if value_i > value_right then
22:                 value_i, value_right = value_right, value_i
```

```
23:                 send value_right to neighbor_right
24:             end if
25:         else
26:             if i mod 2 = 0 and neighbor_right ≠ None then
27:                 send value_i to neighbor_right
28:                 receive value_right from neighbor_right
29:                 if value_i > value_right then
30:                     value_i, value_right = value_right, value_i
31:                     send value_right to neighbor_right
32:             end if
33:         end if
34: end for

35: return value_i
```

EJERCICIO 14. Considere un sistema distribuido *G* = (Π, *E*) representado como un árbol. Diseña un algoritmo distribuido que determine el diámetro del árbol, es decir, el camino más largo entre dos nodos cualesquiera del árbol.

OBSERVACIÓN. Puede asumir que cada proceso p_i conoce su proceso padre en el árbol, y el proceso raíz conoce su identificador único.

Algorithm. Determinación del diámetro de un árbol distribuido

```
1: Initially do
2: begin:
3:   if childreni = 0 then  # Nodo hoja
4:       send BACK((i, 0)) to parenti
5:   end if
6: end

7: when BACK(data) is received from pj such that j ∈ childreni do
8: begin:
9:   val_seti = { (j, d + 1) for (j, d) in data }  # Incrementa en 1 la distancia de cada hijo
```

```
10:   store val_seti  # Almacena las distancias recibidas de los hijos
11:   if all children have sent BACK then
12:       max1, max2 = top_two_max_distances(val_seti)  # Obtiene las dos mayores distancias
13:       if parenti ≠ pi then
14:           send BACK((i, max1 + 1)) to parenti  # Envía la distancia máxima al padre
15:       else
16:           Diámetro = max1 + max2  # El diámetro del árbol
17:       end if
18:   end if
19: end
```

EJERCICIO 15. Demuestra el siguiente lema.

LEMA 1. Si todas las aristas de una gráfica *G* tienen pesos distintos, entonces existe exactamente un árbol generador mínimo (MST) en *G*.

Demostración por contradicción

Hipótesis:
Sea G=(V,E) una gráfica conexa con pesos distintos en cada arista. Supongamos que existen dos árboles generadores mínimos diferentes T1 y T2 en G

Dado que T1 y T2 son árboles generadores, ambos tienen el mismo número de vértices $|V|$ y exactamente $|V|-1$ aristas.

Como son distintos, debe existir al menos una arista eee que pertenece a T1 pero no a T2.

Al agregar e a T2 se forma un ciclo. En este ciclo debe haber al menos otra arista e que esté en T2 pero no en T1

Como los pesos de las aristas son distintos, una de las dos aristas en este ciclo debe tener un peso mayor que la otra.

Si $\omega(e)<\omega(e')$, podemos reemplazar e' por e en T2 y obtener un árbol generador con menor peso que T2, lo que contradice la suposición de que T2 era un MST.

Si $\omega(e)>\omega(e')$, podemos hacer la misma sustitución en T1 y obtener un MST con menor peso que T1 lo que también es una contradicción.

Como en ambos casos llegamos a una contradicción, nuestra suposición inicial de que existen dos MST distintos debe ser falsa.