UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 2

INTEGRANTES

Torres Valencia Kevin Jair - 318331818
Aguilera Moreno Adrián - 421005200
Natalia Abigail Pérez Romero - 31814426

PROFESOR

Miguel Ángel Piña Avelino

AYUDANTE

Pablo Gerardo González López

ASIGNATURA

Computación Distribuida

15 de septiembre de 2022

1. Considera el algoritmo de Flooding visto en clase. Demuestra el siguiente corolario:
Todo proceso p_i que ejecuta el algoritmo de `Flooding`, recibe M en tiempo a lo más el diámetro $diam(G)$ de la gráfica del sistema distribuido.

Dem. Procedamos por inducción en el número de procesos. Supongamos que nuestros procesos viven en la gráfica G conexa.

- Observemos que pasa cuando solo hay un proceso en G, entonces este es el lider y además tiene el mensaje `M` desde antes de la primer ronda[I] (desde la ronda 0).
- Supongamos que la gráfica G, con k procesos, cumple con poder inundarse[II] en tiempo a lo más `diam(G)`.
- Veamos que pasa cuando tenemos G' conexa donde

$$V_{G'} = V_G + p_i \quad \text{y} \quad E_{G'} = E_G + \text{ al menos una arista que conecte a } p_i \text{ con } G.$$

  Existen dos casos posibles,

  1. `diam(G') = diam(G)`. Para este caso tomemos al primer vecino p_j de p_i tal que $d(p_j, v)$ sea mínima para todo $v \in V$ (en contraste con el resto de vecinos de p_i, que no necesariamente existen). Como (por el supuesto anterior) podemos inundar G en tiempo a lo más `diam(G) - 1` [Si suponemos que esto es cierto para `diam(G)` resulta que el diámetro aumenta, esto va en contra de la consideración de conservar el diámetro de G para G'].

     Así, supongamos sin pérdida de generalidad que, p_j recibe el mensaje `M` en tiempo `diam(G) - 1` (desde algún proceso distinguido como líder, según el algoritmo `flooding`). Como p_i y p_j son vecinos, distribuir `M` toma tiempo 1. Entonces la gráfica se inunda, para cada proceso, en tiempo a lo más `(diam(G) - 1) + (1) = diam(G)`.
  2. `diam(G') = diam(G) + 1`. Sabemos que p_i tiene algún vecino p_j (por conexidad de G'). Supongamos, sin pérdida de generalidad, que p_j recibe el mensaje `M` en tiempo `diam(G)`. Así, transmitir `M` de p_j a p_i toma tiempo 1, luego transmitir `M` desde un proceso distinguido como líder hacia p_i toma tiempo `diam(G) + 1`[III].

■

[I] Sabemos que el `diam(G) = 0`.
[II] Por simplicidad, llamaremos inundar a distribuir el mensaje `M` por cada proceso en G por medio del algoritmo `flooding`.
[III] El tiempo de transmitirlo hasta p_j más el tiempo de transmitirlo de p_j a p_i.

2. Considera el algoritmo de BroadcastTree visto en clase. ¿Cuál sería el peor caso en complejidad de tiempo para el algoritmo BroadcastTree? Explica detalladamente.

Algorithm 1 BroadcastTree(ID,soyRaiz,M)

1: $PADRE, HIJOS$
2: Ejecutar inicialmente
3:
4: **if** soyRaiz **then**
5: $send(< M >)$ a todos en HIJOS
6: **end if**
7: Al recibir $< M >$ de Padre
8: $send(< M >)$ a todos en HIJOS

Supongamos siempre la existencia de un árbol generador T.
Primero análicemos la complejidad del algoritmo. Proponemos una gráfica con un vértice v, para que el algoritmo pueda terminar, se necesita que este sea $raiz$. Entonces se tiene que: $u\{padre = Null,\ Hijos = \varnothing,\ soyRaiz = True\}$.

Como v es el único proceso entonces no envía mensaje a nadie y se cumple que:
$Mensajes(BroadcastTree) = |V| - 1 \rightarrow 1 - 1 = 0$
$Tiempo(BroadcastTree) = Profundidad(T) = 0$

Sea $A = \{u|\ profundidad(u) = max(profundidad(v))\ con\ v \in V\}$, para el conjunto de procesos A con mayor profundidad.
Sea $d = profundidad(T)$. Entonces, para todos los procesos en el nivel $d-1$ son hoja, por lo tanto esos procesos ya terminaron, pero podría existir un vértice v_i para cada proceso en A, tal que $d(v_i, u_i) = 1$ (i.e se repite), entonces la $profundidad(v_i) = d-1$. Por lo tanto, en el tiempo $d-1$ se recibió y envió el mensaje, por lo que en d se termina.
En este caso el mejor tiempo de complejidad, es igual a la $profundidad(T)$. Y para el peor, sería que la $profundidad(T)$ se volviese cuadrática, en caso de existir hojas que no compartan profundidad con cualquier $u_i \in A$ y no hayan recibido mensaje.

3. Considera el algoritmo de BroadConvergecast visto en clase.

1. Prueba el siguiente lema: «Todo proceso p_i a profundidad D, recibe el mensaje $< START >$ en tiempo D».

   Demostración por inducción sobre el sobre el camino formado por los *parents*

   Caso base: Sea un árbol distribuido con la raíz sin hijos, por lo tanto la profundidad es 0. Además el mensaje lo recibe en tiempo 0. Por lo tanto se cumple.

   Hipótesis inductiva: Sea un proceso u en un árbol distribuido a profundidad D, el mensaje $< START >$ llegara en tiempo D

   Paso inductivo: Sea v un proceso en el árbol distribuido tal que $u = v.parent$, es decir, u es padre de v entonces la $profundidad(v) = profundidad(u) + 1$ por hipótesis inducctiva $profundidad(v) = D + 1$, además el mensaje tomara $D + 1$ tiempo en llegar a v por que tarda D tiempo para llegar a u y v es hijo de u.

   □

2. Prueba el siguiente lema: «Todo proceso p_i a profundidad D envía su mensaje a la raíz en el tiempo $D + 2 * altura(p)$».

   Demostración por inducción sobre el sobre el camino formado por los *parents*

   Caso base: Sea un árbol distribuido con la raíz u sin hijos, por lo tanto $profundidad(u) = 0 = altura(u)$. El mensaje lo recibe en tiempo 0. Por lo tanto se cumple $D + 2 * altura(u) = 0 + 2 + (0) = 0$

   Hipótesis inductiva: Sea un proceso v en un árbol distribuido a profundidad D envía su mensaje a la raíz en el tiempo $D + 2 * altura(v)$

   Paso inductivo: Sea u un proceso en el árbol distribuido tal que $u = v.parent$, es decir, u es padre de v. Por hipótesis inductiva el proceso v, a profundidad D, envía su mensaje en $D + 2 * altura(v)$. Ya que u es padre de v su profundidad es $D - 1$ y su altura es $altura(v) - 1$ por tanto envia su mensaje en $(D - 1) + 2 * (altura(v) - 1)$

   □

4. ¿Se basan los algoritmos de BroadcastTree y ConvergeCast en el conocimiento acerca del número de nodos en el sistema? ¿Por qué?

Examinar el algoritmo BroadcastTree

El algoritmo requiere de información acerca de los nodos, necesita conocer el número de hijos de un nodo para enviar el mensaje a estos, en la linea 5 y 8 realiza esta instrucción. Además requiere conocer a su PADRE para recibir un mensaje de el en la linea 7.

Algorithm 2 BroadcastTree(ID,soyRaiz,M)

```
1: PADRE, HIJOS
2: Ejecutar inicialmente
3:
4: if soyRaiz then
5:     send(< M >) a todos en HIJOS
6: end if
7: Al recibir < M > de Padre
8: send(< M >) a todos en HIJOS
```

Examinar el algoritmo ConvergeCast

El algoritmo requiere de información acerca de los nodos, necesita conocer el número de hijos de un nodo para saber si el nodo es una hoja y debe enviar el mensaje a su padre. En la linea 6 pregunta por la cantidad de hijos del nodo, luego en la 11 requiere esta información de nuevo. Además requiere conocer a su PADRE pare enviarle el mensaje en las lineas 7 y 12.

Algorithm 3 ConvergeCast(ID,soyRaiz)

```
1: PADRE ← 0
2: HIJOS ← 0
3: noRecibidos ← 0
4: Ejecutar inicialmente
5:
6: if |HIJOS| == 0 then                                   ▷ es una hoja
7:     send(< ok >) a PADRE
8: end if
9: Al recibir < ok > de algún puerto en HIJOS
10: noRecibidos + +
11: if noRecibidos == |HIJOS| then
12:     send(< ok >) a PADRE
13: end if
```

Sin embargo, ninguno de estos algoritmos requiere conocer el total de nodos en la red ya que avanza por la red mediante relaciones de padre-hijo.

5. Generaliza el algoritmo de Broadconvergecast para:

1. Construya un árbol generador, es decir, inicialmente cada proceso tendrá sus variables $PADRE = \perp$ e $HIJOS = \emptyset$ y conforme el algoritmo vaya avanzando en el número de rondas, esas variables se vayan actualizando. El proceso raíz (distinguido) debe conocer el momento en que se terminó de construir este árbol generador.

2. Suponga que cada proceso tiene una entrada distinta para reportar algo (pueden ser información de sensores, lecturas, etc.) A partir del algoritmo anterior, indica las modificaciones que se tendrían que hacer en el algoritmo, para que se recolecte la información de estos procesos y la raíz tenga todas la entradas. Analiza la complejidad en bits, es decir, el total de bits que son enviados sobre los canales de comunicación (hint: Cada mensaje de información puede tomar k bits).

Solución: Para este ejercicio exhibamos el algoritmo que cumple con los requesitos de (1):

Algorithm 4 NewBroadconvergecast(ID,soyRaiz)

```
1: PADRE ← ⊥
2: HIJOS ← ∅
3: noVecinos ← 0
4: Ejecutar inicialmente:
5: if ID = soyRaiz then
6:     PADRE ← ⊥
7:     send(<START>) a todos los puertos;
8: end if                                   ▷ Hasta aquí termina "Ejecutar inicialmente".
9: Al recibir <START> desde algún puerto P:
10: if PADRE = ⊥ then
11:     Padre ← P
12: end if
13: if no hay puertos distintos al PADRE then
14:     send(<OK>) a PADRE
15: else
16:     send(<START>) a los puertos distintos que PADRE
17: end if                                  ▷ Hasta aquí termina "Al recibir <START> ...".
18: Al recibir <OK> desde algún puerto h:
19: noVecinos++
20: HIJOS ← h
21: if |HIJOS|, noVecinos, y el número de puertos menos uno son iguales then
22:     if soyRaiz then
23:         reportar terminación. El árbol esta construido.
24:     else
25:         send(<OK>) a PADRE
26:     end if
27: end if
```

Nota: Suponemos que cada proceso conoce a sus vecinos. Conoce desde que puerto llega la información y a que puertos enviar las información correspondiente.

Para el punto (2), tenemos que:
Cada proceso debería tener un conjunto como estructura de datos, en este conjunto guardaríamos la información de llegada (cada proceso le manda a su padre la información de llegada guardada en su propio conjunto). Si el proceso p_i tiene la información del proceso p_j, entonces p_i mete esta información a su conjunto y esta no es duplicada (por la definición de conjunto).

Para el análisis de complejidad de este algoritmo supongamos que cada mensaje de información toma k bits y que introducir nuevos elementos a los conjuntos de cada proceso toma tiempo c.

Iniciemos pensando ¿qué pansa cuando la información ha llegado a las hojas del árbol? (esto incluye arboles degenerados como una lista). Para este punto cada proceso ha enviado la información de llegada a su padre hasta que, eventualmente, llega al proceso raíz. Entonces, cada nivel del árbol manda su información y la información del resto (en profundidad), así la información enviada es

$$\frac{n \cdot (n-1)}{2} \cdot k$$

por nivel y donde n es el nivel del en el que se encuentran los procesos a enviar su información.
Este análisis no es sobre el tiempo, y como estamos trabajando con conjuntos. Entonces, podemos asegurar que las esturcturas conjunto tendrán al final exactamente $m \cdot k$ bits, donde m es el número de información enviada (si cada proceso tiene exactamente un mensaje con información única, entonces m tomará el valor de la cantidad de procesos).
En tiempo, estamos cerca de una cantidad lineal $(2 \cdot |V-1|)$ de pasos (pues este algoritmo se ejecuta en cada proceso).

6. Da un algoritmo distribuido para contar el número de procesos en cada capa de un árbol enraizado T de forma separada. Al final la raíz reportará el número de procesos por capa. Analiza la complejidad de tiempo y la complejidad de mensajes de tu algoritmo.

Necesitariamos de un algoritmo auxilar que nos permite saber que en cada proceso, su variable asignará la profundidad con su propia profundidad y regresa la profundidad del Árbol.

Se propondría que:
El algoritmo se comienza desde la raíz, indicando su propia profundidad y avisando a todos sus vecinos. Para el caso en el que, la raíz no cuente con vecinos, su profundidad será cero.

Cuando un proceso/vértice recibe un entero, se inicializa su variable de profundidad conforme al entero que recibio. En el caso de que sea HOJA, entonces termina y le notificará a su PADRE con su altura, de lo contrario, avisará a todos sus HIJOS su profundidad.

Cuando se reciba $< Acabe, n >$, el proceso indicará, si la n es mayor su variable max, de ocurrir entonces se fijará $max = n$.

Si $< Acabe, n >$ es igual al numero de HIJOS, se acabó. De ser el caso de la raíz, no sregresará su variable max, de lo contrario, le avisará a su PADRE que acabó y su profundidad máxima.

Nuestro análisis de tiempo y espacio de complejidad:
Su tiempo es la $2 \cdot profundidad(T)$, ya que recorre el árbol desde la raíz hasta las hojas y viceversa. Y se envió $2|E|$ de mensajes
Como ahora cada proceso/vértice podría conocer su profundidad, podriamos regresar la cardinalidad los HIJOS de cada capa/nivel.

Algorithm 5 nivelTree(soyRaiz)

```
1: profundidadArbol = profundidadT(soyRaiz), nivelT = 0, vecinos = 0
2: PADRE, HIJOS, profundidad, soyRaiz = soyRaiz, string =''
3: Ejecutar inicialmente
4:
5: if soyRaiz then
6:     string = '1'
7:     if |Hijos| == 0 then return '1'
8:     else
9:         string = string++ ';' ++ |HIJOS|
10: //Todos los procesos del nivel mandarán la |HIJOS|, esto hace la linea 10
11:         send(< Manda, 1 >) a todos en HIJOS
12:     end if
13: end if
14:
15: Al recibir < Manda, nivel > de Padre
16: if profundidad == capa  then
17:     send(< |HIJOS|, nivel >) a Padre
18: else
19:     send(< Manda, nivel >) a todos en HIJOS
20: end if
21:
22: Al recibir < HIJOS, nivel > de algún proceso en HIJOS
23: vecinos +=, nivelT += HIJOS
24: if vecinos == |HIJOS| then
25:     if soyRaiz then
26:         string = string ++ ';' ++HIJOS
27:         if nivel == profundidadArbol-1 then
28:             return string
29:         else
30:             send(< Manda, nivel + 1 >) a HIJOS
31:         end if
32:     else
33:         send(< nivelT, nivel >) a PADRE
34:     end if
35: end if
```

El algoritmo, recorre desde la raíz a sus hojas (niveles) y viceversa, a comparación de la mayor profundidad, entonces se tiene que el tiempo toma: $profundidad(t) * profundidad(T)$
Los mensaje solo se enviarán al nivel de interés: $n = profundidad(T) - 1$, y asi también para el recorrido de abajo hacia arriba. Finalmente nos regresará un cadena $n_0, n_1, n_2, ..., n_i$ donde n_i será el número de vértices por nivel.