



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

### **Tarea 4**

#### 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**

8 de octubre de 2022

1. Describe un algoritmo distribuido basado en *DFS* que cuente el número de procesos en un sistema distribuido cuya gráfica  $G$  es arbitraria. Al terminar de contar, debe informar a todos los procesos el resultado del conteo. Muestra que es correcto.

**Caso base.**

Sea  $G$  una gráfica tal que  $V_G = \{p_1\}$ , por la línea 8  $\text{contarProcesosDFS}() = 1$  lo cual es correcto.

**Hipótesis de inducción**

Para cualquier gráfica  $G$  con  $n$  vértices  $\text{contarProcesosDFS}()$  cuenta la cantidad procesos de  $G$

Al inicio de la ejecución si no

**Paso inductivo**

Por demostrar que dada una gráfica  $G$  con  $n + 1$  vértices  $\text{asignarEtiquetasDFS}()$  asigna etiquetas únicas en el rango  $[1, \dots, n + 1]$ .

**Algorithm 1** contarProcesosDFS(ID,soyLider)

---

```

1:  $Padre = \perp$ 
2:  $Hijos = \emptyset$ 
3:  $numProcesos = 0$ 
4:  $SinExplorar = \text{todos los vecinos}$ 
5: Si no he recibido algún mensaje
6: if soyLider and  $Padre == \perp$  then
7:    $Padre = ID$ 
8:    $numProcesos = 1$ 
9:   explore( $numProcesos$ )
10: end if
11: Al recibir  $\langle numP \rangle$  desde el vecino  $p_j$ :
12: if  $Padre == \perp$  then
13:    $Padre = j$ 
14:    $numProcesos = numP + 1$ 
15:   elimina  $p_j$  de SinExplorar
16:   explore( $numProcesos$ )
17: else
18:   if  $numProcesos < numP$  then
19:      $numProcesos = numP$ 
20:   end if
21:   send( $\langle already \rangle$ ) a  $p_j$ :
22:   elimina  $p_j$  de SinExplorar
23: end if
24: Al recibir  $\langle already \rangle$  desde  $p_j$ 
25: explore( $numProcesos$ )
26: Al recibir  $\langle parent, numP \rangle$ 
27: if  $numProcesos < numP$  then
28:    $numProcesos = numP$ 
29: end if
30:  $Hijos \cup p_j$ 
31: explore( $numProcesos$ )
32: procedure EXPLORE( $numP$ )
33:   if SinExplorar  $\neq \emptyset$  then
34:     elegir  $p_k$  en SinExplorar
35:     eliminar  $p_k$  de SinExplorar
36:     send ( $\langle numP \rangle$ ) a  $p_k$ 
37:   else
38:     if  $Padre \neq ID$  then
39:       send( $\langle parent, numP \rangle$ ) a Padre
40:     end if
41:     if  $Padre == ID$  and  $numProcesos < numP$  then
42:        $numProcesos = numP$ 
43:     end if
44:   end if
45: end procedure

```

---

2. Describe un algoritmo distribuido basado en *DFS* que, en una gráfica arbitraria  $G$  con  $n$  vértices anónimos, asigne etiquetas únicas en el rango  $[1, \dots, n]$  a los vértices de  $G$ . Muestra que es correcto.

Hint: Puedes suponer que cada proceso conoce a sus vecinos aunque estos no tengan una etiqueta explícita.

---

**Algorithm 2** asignarEtiquetasDFS()

---

```

1: Al inicio seleccionar un nodo al azar para empezar a enviar mensajes
2:  $Hijos = \emptyset$ 
3:  $ID = 1$ 
4: Padre = ID
5: SinExplorar = todos los vecinos
6: etiqueta = ID
7: EXPLORE(< etiqueta, 1 >)
8: Al recibir < etiqueta, i > desde el vecino  $p_j$ :
9: if Padre ==  $\perp$  then
10:   Padre = j
11:   ID = i
12:   elimina  $p_j$  de SinExplorar
13:   EXPLORE(etiqueta, i)
14: else
15:   send(< already >) a  $p_j$ :
16:   elimina  $p_j$  de SinExplorar
17: end if
18: Al recibir < already > desde  $p_j$ 
19: EXPLORE(etiqueta)
20: Al recibir < parent, i > desde  $p_j$ 
21: Hijos  $\cup p_j$ 
22: EXPLORE(etiqueta, i)
23: procedure EXPLORE(etiqueta, i)
24:   if SinExplorar  $\neq \emptyset$  then
25:      $k = i, j = 0$ 
26:     while  $j < |SinExplorar|$  do
27:        $k+ = 1$ 
28:       send (< etiqueta, k >) a  $p_j$ 
29:       eliminar  $p_j$  de SinExplorar
30:        $j+ = 1$ 
31:     end while
32:   else
33:     if Padre  $\neq ID$  then
34:       send(< parent, etiqueta >) a Padre
35:     end if
36:   end if
37: end procedure

```

---

Por demostrar que el algoritmo es correcto, es decir, dada una gráfica arbitraria  $G$  con  $n$  vértices anónimos, asigne etiquetas únicas en el rango  $[1, \dots, n]$  a los vértices de  $G$ . Demostración por inducción:

Demostración por inducción sobre  $n$  número de vértices anónimos en  $G$ .

**Caso base.** Sea  $G$  una gráfica tal que  $V_G = \{p_1\}$ , por la línea 3 el  $ID(p_1) = 1$  por lo tanto todos los vértices de  $G$  tiene una etiqueta única y conoce las etiquetas de sus vecinos.

**Hipótesis de inducción** Para cualquier gráfica  $G$  con  $n$  vértices `asignarEtiquetasDFS()` asigna etiquetas únicas en el rango  $[1, \dots, n]$ . Al inicio de la ejecución el nodo elegido tiene  $ID(p_1) = 1$  luego envía < etiqueta, 1 > a uno de sus vecinos  $p_j$  este asigna  $ID(p_j) = 2$  si tiene vecinos diferentes del líder, es decir  $SinExplorar \neq \emptyset$  de manera similar asigna un nuevo  $ID$  una unidad mayor, es decir  $ID(p_{j+1}) = ID(p_j) + 1$ , cuando el proceso  $p_j$  ya no tenga vecinos por explorar le envía el mensaje < parent, i > donde la  $i$  es la última etiqueta asignada, cuando los vértices reciben < parent, i > desde  $p_j$  guardan este vecino en el conjunto de sus hijos y envían la etiqueta  $i$  a su padre, cuando  $Padre == ID$  termina la asignación de  $ID$  de los vecinos de uno de los hijos del

proceso con  $ID = 1$  y procede a enviar  $\langle etiqueta, i \rangle$ , donde  $i$  es el último  $ID$  asignado, a otro de los hijos del proceso con  $ID = 1$ , de manera similar recorre los vecinos asignando un ID mayor al que recibieron. Hasta que el proceso con  $ID = 1$  no tenga vértice por explorar el algoritmo termina.

**Paso inductivo**

Por demostrar que dada una gráfica  $G$  con  $n + 1$  vértices `asignarEtiquetasDFS()` asigna etiquetas únicas en el rango  $[1, \dots, n + 1]$ .

Sea un vértice  $p_{j+1}$  que no tiene etiqueta, pero sabemos que es vecino de al menos un vértice  $p_j$ , que por hipótesis inductiva tiene una etiqueta al igual que sus vecinos excepto  $p_{j+1}$ , por lo tanto SinExplorar de  $p_j$  es  $\{p_{j+1}\}$  y por la línea 28 en el procedimiento `EXPLORE`  $P_{j+1}$  recibe el mensaje:  $\langle etiqueta, i \rangle$ , asigna a  $p_j$  como su padre y  $ID(p_{j+1}) = ID(p_j) + 1$ , por lo tanto todos los vértices  $n + 1$  de  $G$  tienen una etiqueta única en el rango  $[1, \dots, n + 1]$ .

**3.** Modifica el algoritmo DFS para que se ejecute en tiempo a lo más  $2|V|$  y no mande más de  $2|E|$  mensajes, suponiendo que las aristas son bidireccionales.

Hint: Cuando un proceso recibe el mensaje  $M$  por primer vez, este notifica a todos sus vecino pero envía el mensaje a sólo uno de ellos.

**Algoritmo:** DFS Modificado.

```

1. Padre =  $\perp$ 
2. Hijos =  $\emptyset$ 
3. sinExplorar = todos los vecinos
4.
5. Si no he recibido algún mensaje:
6.   if padre =  $\perp$  then
7.     soyLider = ID
8.     Padre =  $p_i$ 
9.     explore()
10.
11. Al recibir  $\langle leader, nuevo - id \rangle$  desde el vecino  $p_j$ :
12.   if soyLider < nuevo-id then
13.     soyLider = nuevo-id
14.     Padre =  $p_j$ 
15.     Hijos =  $\emptyset$ 
16.     sinExplorar = todos los vecinos, excepto  $p_j$ 
17.     explore()
18.   else if soyLider = nuevo-id then
19.     send  $\langle alredy, soyLider \rangle$  a  $p_j$ 
20.   //De lo contrario, soyLider > nuevo-id y el DFS para nuevo-id se detiene
21.
22. Al recibir  $\langle soyLider, nuevo - id \rangle$  desde el vecino  $p_j$ :
23.   if nuevo-id = soyLider then explore()
24.
25. Al recibir  $\langle parent, nuevo - id \rangle$  desde el vecino  $p_j$ :
26.   if nuevo-id = soyLider then
27.     Hijos  $\cup \{p_j\}$ 
28.     explore()
29.
30. procedure explore():
31.   if sinExplorar =  $\emptyset$  then
32.     elegir  $p_k$  en sinExplorar
33.     eliminar  $p_k$  de sinExplorar
34.     send  $\langle leader, soyLider \rangle$  a  $p_k$ 
35.   else:
36.     if Padre  $\neq$  ID then send  $\langle parent, leader \rangle$  a Padre
37.     terminar

```

4. Considera el algoritmo 1, que calcula una  $\Delta + 1$  coloración, donde  $\Delta$  es el grado máximo en la gráfica. Muestra una gráfica  $G$  con al menos 10 vértices y una asignación de IDs, donde el algoritmo coloree todos los procesos (el primer momento en el que todas las variables  $c$  son distintas de  $\perp$ ) en tiempo  $diam(G)$ . Muestra otra asignación de IDs para las que el algoritmo coloree en tiempo a los más  $diam(G)/2$ .

Pseudocódigo 1:  $\Delta + 1$  coloración

```

1 Algoritmo coloring(ID):
2    $c = \perp$ 
3
4 Ejecutar inicialmente:
5   send( $\langle ID, c \rangle$ )
6
7 Al tener todos los mensajes de todos mis vecinos en  $t \geq 1$ :
8   Mensajes =  $\langle ID_1, c_1 \rangle, \dots, \langle ID_j, c_j \rangle \setminus (j \setminus) = \text{grado maximo en la grafica}$ 
9    $A = \{ID_i | c_i = \perp\}$ 
10  if  $c == \perp \wedge ID = \max(A \cup \{ID\})$  then
11     $c = \min(\{1, \dots, \Delta + 1\} \setminus \{c_i \neq \perp\})$ 
12  send( $\langle ID, c \rangle$ ) a todos los vecinos

```

▷ Para este problema dividamos la solución en 3 respectivas soluciones:

1. Mostrar una ejecución en tiempo  $diam(G)$ , con  $G$  una gráfica tal que  $|V_G| = 10$ .

$p_3 \circ$

$G$

$p_2 \circ$

$p_2 \circ$

$p_0 \circ$

$p_1 \circ$

Figura 1: Gráfica  $G$ .

◁

**5.** Un toro  $n \times m$  es una versión dos dimensional de un anillo, donde un nodo en la posición  $(i, j)$  tiene un vecino hacia el norte en  $(i, j - 1)$ , al este en  $(i + 1, j)$ , al sur en  $(i, j + 1)$  y al oeste en  $(i - 1, j)$ . Esos valores se calculan módulo  $n$  para la primera coordenada y módulo  $m$  para la segunda; de este modo  $(0, 0)$  tiene vecinos  $(0, m - 1)$ ,  $(1, 0)$ ,  $(0, 1)$  y  $(n - 1, 0)$ . Supongamos que tenemos una red síncrona de paso de mensajes en forma de un toro  $n \times m$ , consistente de procesos anónimos idénticos, los cuáles no conocen  $n$ ,  $m$  o sus propias coordenadas, pero tienen sentido de la dirección (es decir, puede decir cual de sus vecinos está al norte, este, etc.). **Pruebe o refute:** Bajo estas condiciones, ¿existe un algoritmo determinista que calcule cuando  $n > m$ ?