UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 5

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

9 de noviembre de 2022

1. Considera la siguiente variante del algoritmo de consenso con terminación temprana. Contesta lo siguiente:

a) Demuestra que el algoritmo 1 soluciona el problema del consenso, tolerando $f < n$ fallas de tipo paro, donde n es el número de procesos en el sistema.

Por demostrar:

- Terminación

  El algoritmo no tiene una clara condición de salida. La linea 6 se ejecutara indefinidamente, a menos que exista una en la linea 9 cuando se ejecuta *decide max(vista).*

- Validez

  El valor de $prop = vista$ fue propuesto por algún proceso en cada ronda.

- Acuerdo. No queda claro si el algoritmo termina, sin embargo: Veamos una ejecución cuando $f = 0$: Al inicio de la ejecución $r = 0$, la *flag* = *false*, envía $< prop, false >$, luego $r = 1$, $vista = prop, prop_1$, luego $rec[1] = 1 + 1$ y no se modifica *flag* hasta que $rec[1 - 1] == rec[1]$, en esta ronda no se modifica *flag*

  Cuando $r = 2$, $send(< prop_2, false >)$ a todos $vista = \{prop, prop_1, prop_2\}$ $rec[2] = 1 + 1$

  $rec[2 - 1] == rec[2] \rightarrow rec[1] == rec[2]$, en esta ronda se modifica $flag = true$

  Cuando $r = 3$, $send(< prop_2, true >)$ a todos *decide max(vista)* $vista = \{prop, prop_1, prop_2, prop_3\}$ $dec = true$ $rec[3] = 1 + 3$ $rec[3 - 1] == rec[3] \rightarrow rec[2] == rec[3]$, en esta ronda no se modifica $flag = true$

  Cuando $r = 4$, $send(< prop_3, true >)$ a todos *decide max(vista)* $vista = \{prop, prop_1, prop_2, prop_3, prop_4\}$ $dec = true$ $rec[4] = 1 + 3$

  $rec[4 - 1] == rec[4] \rightarrow rec[3] == rec[4]$, en esta ronda no se modifica $flag = true$

  Si algún proceso tuviera una falla de tipo paro, el valor de vista sería diferente y si al menos una flag de la ronda anterior es verdadera entonces llega a un acuerdo con los valores de vista. Por lo tanto en cada dos rondas a partir de la tercera, todos los proceso (vivos) acuerdan el mismo valor.

b) ¿Es cierto que los procesos correcto terminan en a lo más $max(t + 2, f + 1)$ rondas en el algoritmo 1? Argumenta tu respuesta. Recuerda que $t \leq f$ es el número de fallas que realmente ocurren en una ejecución dada.
t es el número de fallas que realmente ocurren en una ejecución+1. f es el número de fallas de tipo paro+2.

c) Haz un análisis del número máximo de mensajes que se envían en una ejecución del algoritmo 1. Tu cota debe estar en función de n y f.

Pseudocódigo 1: Algoritmo de consenso temprano

```
1  Algoritmo consenso(prop)
2    flag = false
3    rec[0, 1, ...] = [n, n, ...]
4    r = 0
5    vista = prop
6    while True do:
7      r = r + 1
8      send(<vista, flag>) a todos
9      if flag then decide max(vista) end if
10     vista = union de todas las vistas recibidas en
11             la ronda r y la mia
12     dec = or de todas las flag's
13           recibidas en la ronda r y la mia
14     rec[r] = 1 + numero de mensajes recibidos en la
15              ronda r y la mia
16     if dec ∨ (rec[r−1] == rec[r] then
17       flag = true
18     end if
19   end while
```

2. Demuestra que el algoritmo 2 soluciona el problema del consenso en una gráfica G arbitraria, tolerando $f < k(G)$ fallas de tipo paro. $k(G)$ denota la conexidad por vértices de G, es decir, el mínimo número de vértices que se tienen que quitar de G para desconectarla. Entonces, si hay menos de $k(G)$ fallas de los procesos, la gráfica que queda sigue siendo conexa. Tip: Piensa que tanto tarda en fluir la entrada mínima más pequeña, a pesar de las fallas que puedan ocurrir.

Pseudocódigo 2: Algoritmo de consenso para gráficas árbitrarias para $f < \mathrm{k}(G)$ fallas. n es el número de procesos en el sistema

```
1  Algoritmo consenso(prop)
2    for r = 1 to n do
3      send(<prop>) a todos mis vecinos
4      vista = conjunto con todas las prop recibidas y la mia
5      prop = min(vista)
6    end for
7    decide prop
```

▷ Para mostrar que el algoritmo 2 soluciona el consenso, mostremos que

- *Terminación.* Si $f < k(G)$ entonces G siempre será conexa incluso si los f procesos fallan. Por tanto, los procesos que hayan logrado el consenso hasta antes del último fallo en los f procesos enviarán nuevamente su conjunto con $ID's$ y su propuesta de consenso, como la gráfica sigue siendo conexa, eventualmente todos los procesos tendrán la vista de todos los demás que no tuvieron fallas de tipo paro y la propuesta de cada proceso será tomada en cuenta para el consenso de los $G - f$ procesos restantes.
- *Validez.* El consenso se originó en alguno de los procesos, por tanto es válido.
- *Acuerdo.* En cuanto termine el algoritmo todos los procesos activos están conectados y tienen la misma propuesta de consenso. De no ser así, falta un proceso por enviar su consenso y el `for` no ha llegado a su

fin, esto contradice el hecho de que nuestro algoritmo haya terminado, por tanto se cumple que todos los procesos llegan al acuerdo en cuanto se termine el algoritmo (pues la función `min` es determinista).

◁

3. Considera el algoritmo de consenso con fallas de tipo paro visto en clase. Suponga que en lugar de ejecutar $f + 1$ rondas, el algoritmo sólo ejecuta f, con la misma regla de decisión. Describa una ejecución particular en la que las propiedades de validez y acuerdo sean violadas.

▷ Para esta ejecución basta observar que puede pasar en cada ronda si resultan 2 fallos de tipo paro en un sistema distribuido con 4 procesos. Esto es

- $\mathbf{R_1}$. Para esta ronda todos los procesos mandan su `ID` exceptuando al proceso p_1 que logra mandar su `ID` al proceso p_2 y después tiene un problema tipo fallo y no completa su envio a los demás procesos.
- $\mathbf{R_2}$. En la 2^{da} ronda el proceso p_2 manda su `ID` a p_4 pero no logra enviarselo al proceso p_3. Entonces p_2 es nuestro segundo proceso con fallo de tipo paro.

El algoritmo solo nos permite realizar 2 rondas, pues $f = 2$. Así, la vista de p_3 es $\{2, 3, 4\}$ y su propuesta es $prop = 2$. Mientras que p_4 contiene en su vista a $\{1, 2, 3, 4\}$ y su propuesta es $prop = 1$. Como se puede observar, los procesos restantes en el sistema no han llegado a un consenso en f rondas. ◁

4. Da una ejecución del algoritmo 3 para $n = 4$ procesos y $t = 1$ fallas bizantinas, en la que los procesos correctos no llegue a un consenso, a pesar de que los cuatro procesos, íncluido el Bizantino, empiecen con la misma propuesta y el bizantino sea el último de los coordinadores de la ejecución. Explica tu respuesta.

Pseudocódigo 3: Algoritmo de consenso bizantino

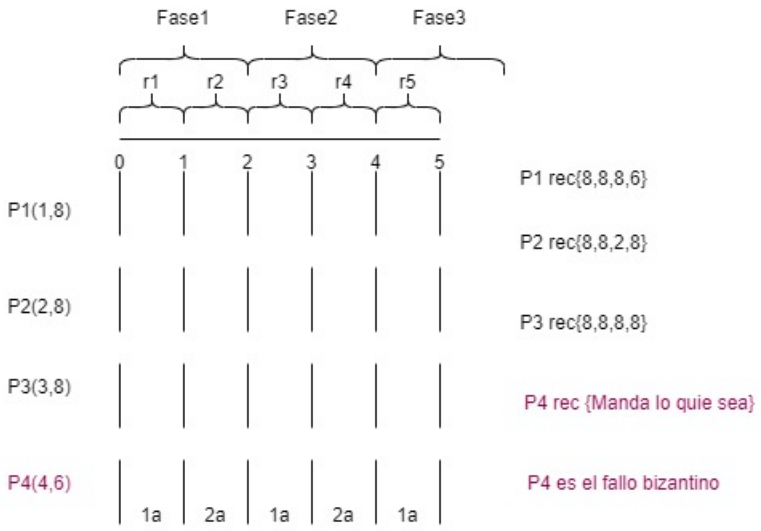
```
1  Algoritmo consensoBizantino(prop)
2    propInicial = prop
3    for fase = 0 to t do
4      // primera ronda de la fase
5      send(<prop>) a todos
6      rec = multiconjunto con todas las prop's recibidas
7            en la ronda, incluida la mia
8      frec = alguno de los valores que se repite mas en rec
9      num = numero de veces que se repite frec en rec
10     // Segunda ronda
11     if ID == fase then
12       send(<frec>)
13     end if
14     if recibi mensaje <frec'> del coordinador then
15       // se descartan los mensajes de los bizantinos
16       coord = frec'
17     else
18       coord = propInicial
19     end if
20     if num > n/2 + t then
21       prop = frec
22     else
23       prop = coord
24     end if
25   end for
26   decide prop
```

Sabemos que el algoritmo soluciona el problema para $t < \frac{n}{4}$, donde n son los proceso del sistema, como sabemos que se ejecuta por fases de 2 rondas, dado que se quiere hacer la ejecución con $n = 4$ y con $t = 1$ fallos bizantinos, no es posible hacer que estos no lleguen a un consenso. Dado que siempre estos procesos correctos:

- Siempre decidirán un valor.
- De tener una propuesta inicial v, entonces sólo decidirán v.
- Siempre las decisiones de estos, son iguales.

Pero, podemos asegurar que de tener un proceso de más, donde sea con $n = 5$ procesos y $t = 1$ fallas bizantinas es posible que no lleguen a un consenso. De misma forma podría ser que de tener $n = 4$ procesos y $t = 2$ fallas bizantinas es posible que no lleguen a un consenso. Esto debido a que no existe un algoritmo, si $t \geq \frac{n}{3}$

Se anexa ejemplo de ejecución con $n = 4$ y una falla bizantina. (Siempre se llega a un consenso).



5. El algoritmo de consenso bizantino visto en clase, resuelve el problema del consenso bizantino para $f < \frac{n}{4}$ procesos. Para valores grandes de f, el algoritmo podría fallar por violar una o más de las propiedades de terminación, validez o acuerdo. Para este algoritmo:

- ¿Qué tan grande debe ser f para evitar terminación?
  Entonces solo basta con que $f \geq t$.
- ¿Qué tan grande debe ser f para evitar validez?
  Como en la terminaciòn tenemos que $f \geq t$, es suficiente para que no se llegue tener validez en el consenso, para lograr tener validez el consenso debio haber sido enviado por un algun proceso en el sistema, entonces sin nunca hay terminaciòn por ende no habrà validez.
- ¿Qué tan grande debe ser f para evitar acuerdo?
  Solo basta con que $f \geq \frac{n}{3}$.

Asuma que los procesos conocen la nueva cota f, y cualquier umbral en el algoritmo que use f, se ajusta para corresponder con esta nueva cota. Argumente detalladamente su respuesta.