Algorithmique et programmation distribuée

Zakaria EJJED

Contents

Wednesday February 8th 2023

Exercie 2

1.

Algorithm 1 Sur reception de message Msg(v) de j)

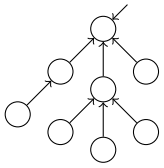
```
if vois_i == 1 then                                   ▷ (A)
    save_i ← v
    envoyer Retour(u) à j
    stop-global
end if
if vois_i == 2 then
    save_i ← v
    envoyer Msg(v) à vois_i sans {j}
    stop-global
end if
```

compléxité: O(2n) → O(n)

Algorithm 2 Sur reception de Retour(v) de j)

```
if estRacine_i == 1 then
    stop-global
end if
if vois_i == 2 then
    envoyer Msg(v) à vois_i sans {j}
    stop-global
end if
```

la racine doit recevoir une info de chacune des feuilles



Connaissance:

$père_i$ le père du $noeud_i$. Vaut null pour la racine

$fils_i$ les fils du $noeud_i$: vaut ndef pour les feuilles

$info_i$: l'information à diffuser: ndef sauf pour les noeuds

Variables:

cpt_i: compter le nombre de messages reçus

$save_i$: init à null, sert à sauvegarder l'info

Algorithm 3 INIT

```
if fils_i == 0 then
    envoyer Msg(info_i) à père_i
    stop-local
end if
```

compléxité en message: n-1 : 1 message par canal de comm

n noeuds ⇒ n-1 arêtes

en temps: hauteur de l'arbre

Algorithm 4 Sur réception de Msg(v) de j)

```
cpt_i++
if cpt_i ==|fils_i| then
    envoyer Msg(info_i) à père_i
    save_i ← v
    if père_i ≠ null then
        envoyer Msg(v) à père_i                    ▷ top-local
    else
        stop-global
    end if
end if
```

2.

Connaissance:

estRacine_i

vois_i voisins du sommet

info_i: l'information à diffuser: ndef sauf pour les noeuds

Variables:

cpt_i: compter le nombre de messages reçus

save_i: init à null, sert à sauvegarder l'info

Algorithm 5 INIT

```
if vois_i == 1 then
    envoyer Msg(v) à vois_i
    père_i ← vois_i
    fils_i ← null
end if
```

Algorithm 6 Sur réception de Msg(v) de j)

```
cpt_i++
ajouter j dans fils i
if estRacine_i == 0 then
    if |vois_i| − cpt_i == 1 then
        enoie msg(v) à vois_i sans fils_i
        père ← vois_i sans fils_i
        stop-local
    end if
else
    if père_i == null AND cpt == |vois_i| then
        stop-global
    end if
end if
```

Exercice 4

Connaissance:

vois_i

estRacine_i

info_i

Variables:

père$_i$ = $NULL$

fils$_i$ = []

save$_i$: init à null, sert à sauvegarder l'info

Algorithm 7 INIT

```
if estRacine then
    envoyer Msg(val_i) à vois_i
end if
```

Algorithm 8 Sur réception de Msg(v) de j

```
if estRacine_i == 1 then
    fils ← fils + j
    if |fils|==|vois_i| then stop-global
    end if
else if père_i == null AND cpt == |vois_i| then
    stop-global
end if
```

Exercice 5

Consigne: Soit un arbre enraciné dans lequel on a les ???, père$_i$, fils$_i$. On demande 1 algo tel qu'à la fin de l'execution, la racine connaisse le nombre de noeuds de l'arbre avec 2 fils.

Wednesday February 15th

TD2 - Horloges

Avec cette construction on aura l'équation suivante:

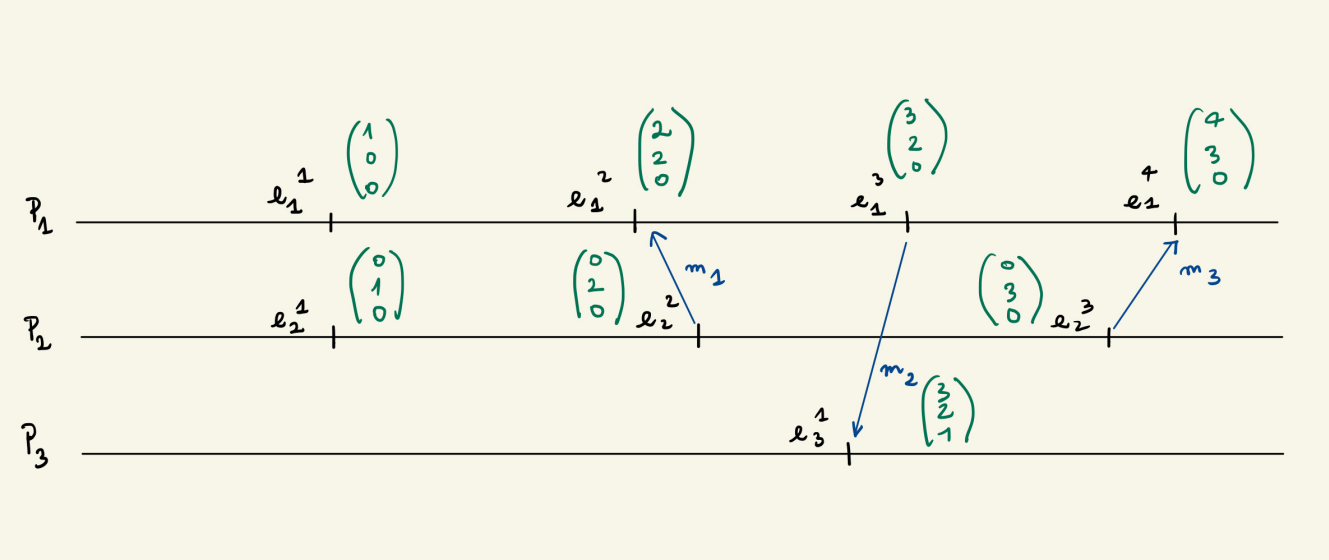
$$e \to e' \Leftrightarrow V(e) < V(e')$$

en terme d'espace mémoire $O(n)$ (il faut stocker un vecteur d'entiers de taille n)

Algorithm 9 Algorithme de construction (en supposant n sites)

init:
- un processeur p_i et un vecteur V_i de taille n dont les valeurs sont initialisés à 0.
- A chaque evenement e on associe une valeure d'horloge
- A chaque event $V_i[i] \leftarrow V_i[i] + 1$
- Lors s'une emission, le vecteur V_i est envoyé dans le message
- Lors d'une reception contenant le vecteur D:

```
for chaque case j!=i do
    V_i[j] ← max(V_i[j], D[j])
end for
```



Comment comparer 2 horloges ?

$$
\begin{array}{lll}
V \leq V' & ssi & \forall j V[j] \leq V[j'] \\
V \leq V' & ssi & V \leq V' \text{ et } \exists k \text{ tq } V[k] < V'[k] \\
V||V' & ssi & !(V \leq V') \cap !(V' \leq V)
\end{array}
$$

Exemple: Horloges incompatibles

$$
\begin{array}{lllll}
 & V(e_1^1) & = & [100] \\
\text{et} & V(e_2^1) & = & [010] \\
\text{car} & V(e_1^1)[1] & > & V(e_2^1)[1] \\
\text{et} & V(e_2^1)[2] & > & V(e_1^1)[2] \\
\text{donc} & e_1^1 & || & e_2^1
\end{array}
$$

Question 1

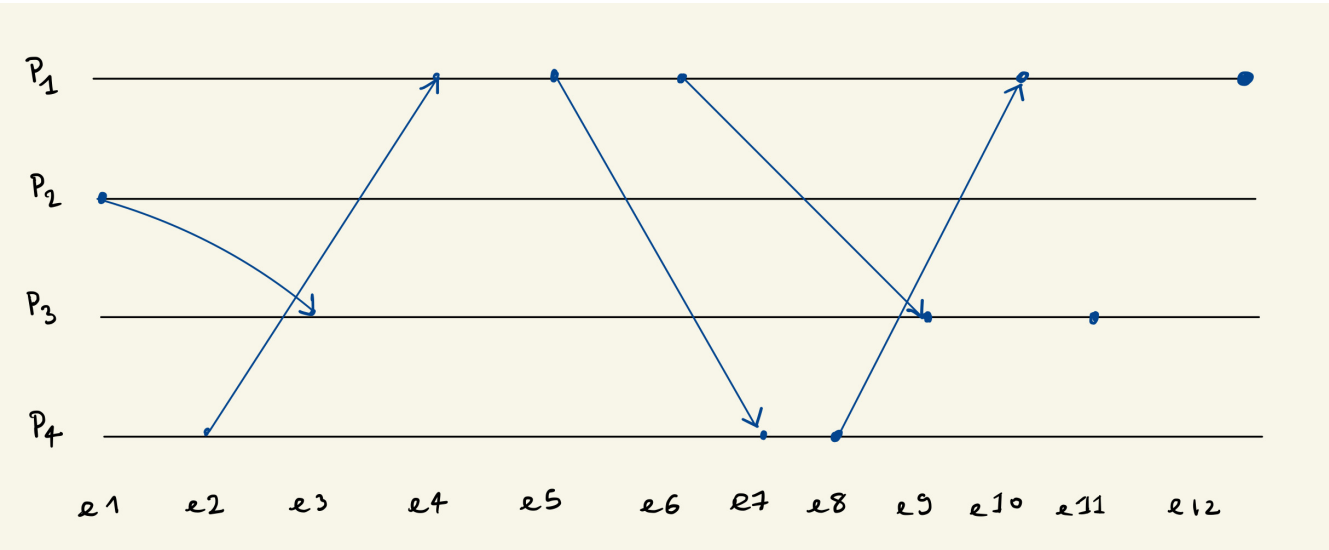


Figure 1: Diagramme Question 1

Question 2

$$e_1 < e_3 < e_2 < e_4 < e_5 < e_6 < e_7 < e_9 < e_8 < e_{11} < e_{10} < e_{12}$$

$$e_2 < e_4 < e_1 < e_3 < e_5 < e_6 < e_7 < e_9 < e_8 < e_{11} < e_{10} < e_{12}$$
$$e_2 < e_1 < e_4 < e_3 < e_5 < e_6 < e_7 < e_9 < e_8 < e_{11} < e_{10} < e_{12}$$

Question 3

- 1^{er} possible.
- 2^{e}: e_5 avant e_4 pas possible car $e_4 \rightarrow e_5$ (local).

Question 4

e_9	$\rightarrow$	e_{11}	(local)
e_5	$\rightarrow$	e_7	(message)
e_1	$\rightarrow$	e_{11}	(transitivité)
e_2	$\rightarrow$	e_{11}	(transitivité)
e_1	$\|\|$	e_5	(!concurrent)
e_7	$\|\|$	e_{11}	(!concurrent)

Question 5

Les évenements précedant e_9 sont: $e_1, e_2, e_3, e_4, e_5, e_6$

Question 6

event	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
Lanport	1,2	1,4	2,3	2,1	3,1	4,1	4,4	5,4	5,3	6,1	6,3	7,1
Vecteur	[0100]	[0001]	[0110]	[1001]	[2001]	[3001]	[2002]	[2003]	[5121]	[4003]	[3131]	[5003]

TD3 - Exclusion Mutuelle

Exercice 1 Algorithme basé sur les permissions de Ricart-Agrawala

1. P2 à fait une demande, puis P3 a fait une demande, puis P2. P1 n'a pas fait de demande.
2. P2, P2, P3, P4
3. P1 a fait 0 demande. P2 peut farie n demande, P3 peut faire m demande, P4 peut faire k demande
4. a) C'est P3 car P3 fait dem(4,3) et P1 fait dem(4,1) or (4,1) < (4,3)
   b) à la fin, P1(4,4), P2(4,1), P3(4,4), P4(4,3)
   c) après C0, il y a eu 12 messages (6 demandes et 6 permissions).
5. P1 et P4 vont faire une demande en parallele:
   P4: dem(3,4)
   P1: dem(4,1) P1 va voir sa demande refusé car 3<4

Wednesday February 22nd 2023

TD3 - Exclusion Mutuelle

Exercice 2 - Algorithme basé sur les permissions de Carvalho-Roucairol

Algorithm 10 Variables et initialisations

H est identique sauf initialisation
h $attendu_i$
if $g \in attendu_i$ **then**
  $i \notin attendu_j$
end if

Algorithm 11 Sur réception de $Dem(h', j)$ de $j \rightarrow$

Envoyer $Perm$ à $j \ldots$
$attendu_i \leftarrow attendu_i U \{j\}$
if etat = E **then**
  envoyer $Dem(last_i, i)$ à j
end if

Algorithm 12 Sur sortie de section critique $\rightarrow$

Envoyer $Perm$ à $j \ldots$
$attendu_i \leftarrow j \ldots$

TD3 - Exclusion Mutuelle

Exercice 3 - Algorithme basé sur la circulation d'un jeton de Ricart-Agrawala

a. * P_1 fait la demande de sc:
-met à jour son tableau
-envoie Dem à ses voisins
-supp qu'il reçoient la demande

- P_5 va executer le code dans sortie de sc:

P_i	nb demande	JetonPresent	valeur jeton
P_1	10000	0	00000
P_2	10000	0	
P_3	10000	0	
P_4	10000	0	
P_5	10000	1	

TD5 - Election

Exercice 1 - Algorithme de Memann et son amélioration par Chang et Roberts.

- 1. Compléxité en message: O(?)

Algorithm 13 Algo de Chang et Roberts (à faire)

- 2. **Initialement:**
Sur un ensemble non vide de sommets
$etat_i \leftarrow candidat$
envoyer $Msg(i)$ à $succ_i$
Sur réception de $Msg(v)$ **de** j**:**
if $etat_i \in \{init, battu\}$ **then**
  $etat_i \leftarrow battu$
  envoyer $Msg(v)$ à $succ_i$
  $min_i \leftarrow min(v, min_i)$
else
  if $v \neq i$ **then**
    $min_i \leftarrow min(v, min_i)$
    envoyer $Msg(v)$ à $succ_j$
  else
    if $min_i == i$ **then**
      $etat_i \leftarrow leader_i$
    else
      $etat_i \leftarrow battu_i$
    end if
  end if
end if

Wednesday March 8th 2023

- a.
  phase 0: 9, 8, 10
  phase 1: 9, 10
  phase 2: 10

- b.
  Finalement le site 10 est élu leader à la phase 4.

Wednesday March 15th 2023

Exercice 4 - Algorithme Vitesse

1. Le site élu est le plus petit.

2. On suppose que le système est synchrone.

3.

- Variables:

- idmin: id le plus petit

- sens: variable indiquant où envoyer le token

- leader: bool indiquand si on est leader

- nbticrestant: nombre de rounds restants avant de partir

le jeton:
* >0: à transmettre + tard * =0 : il faut le transmettre * -1: plus de jeton à transmettre

Algorithm 14 Algorithme Vitesse

```
Initialement:
Envoyer Token(ià à j ∈ vois
Sur chaque tic d'horloge:
if nbticrestant > 0 then
    nbticrestant–
end if
for chaque reception de token(v) do
    if v = i then
        leader ← true
    else if v < idmin then
        idmin ← v
        nbticrestant ← 2^v
        sens_i ← vois_i\{j}
    end if
end for
if nbticrestant == 0 then
    envoyer token(idmin) à sens
    nbticrestant ← -1
end if
```

4. C'est linéaire mais la preuve est compliqué.

TD 4 - Algorithmes Divers

Exercice 1 - Election dans un graphe complet

2.

Exercice 2 - Compter dans un arbre

Algorithm 15 Algorithme Vitesse

3.
Connaissance:
fils_i
pere_i
estFeuille
estRacine
Variables:
restant
counterFeuille $\leftarrow 0$
counterSpecial $\leftarrow 0$
Initialement:
restant = fils_i
if estFeuille==1 **then**
    Envoyer Msg(0,"feuille")
end if
Sur reception Msg(val,s) de j:
restant $\leftarrow$ restant $\backslash j$
if s == "feuille" **then**
    counterFeuille++
else if s == "special" **then**
    counterspecial+=val
end if
if restant == {} **then**
    counterspecial+=val
    if counterFeuille == 1 **then**
        counterspecial++
    end if
    if estRacine == 1 **then**
        StopGlobal
    else
        Envoyer Msg(counterSpecial,"special") $\rightarrow$ pere
    end if
end if