

# Algorithmique et programmation distribuée

Zakaria EJJED

## Contents

<b>Wednesday February 1st 2023</b>	<b>3</b>
<b>Wednesday February 8th 2023</b>	<b>6</b>
Complexité . . . . .	7
A - Configuration . . . . .	8
B - Execution . . . . .	8
Exercice 2 . . . . .	10
1. . . . .	10
2. . . . .	11
Exercice 4 . . . . .	11
Exercice 5 . . . . .	12
<b>Wednesday February 15th 2023</b>	<b>12</b>
Causalité et horloges distribuées . . . . .	12
Modèle proposé par Lanport . . . . .	12
Algorithme de Lanport . . . . .	13
Construction ordre global sur les événements . . . . .	14
Construction de l'ordre totale . . . . .	15
<b>Wednesday February 15th Exercises</b>	<b>15</b>
TD2 - Horloges . . . . .	15
Question 1 . . . . .	16
Question 2 . . . . .	16
Question 3 . . . . .	17
Question 4 . . . . .	17
Question 5 . . . . .	17
Question 6 . . . . .	17
<b>Wednesday February 15th Courses</b>	<b>17</b>
Exclusion Mutuelle . . . . .	17
Algo Ricart-Agrawala avec permission . . . . .	17
Hypothèse sur le réseau: . . . . .	17
Principe de l'algo . . . . .	18
<b>Wednesday February 15th 2023 Exercises</b>	<b>18</b>
TD3 - Exclusion Mutuelle . . . . .	18
Exercice 1 Algorithme basé sur les permissions de Ricart-Agrawala . . . . .	18
<b>Courses</b>	<b>18</b>
<b>Wednesday February 22nd 2023 Exercises</b>	<b>19</b>
TD3 - Exclusion Mutuelle . . . . .	19
Exercice 2 - Algorithme basé sur les permissions de Carvalho-Roucairol . . . . .	19
<b>Wednesday February 22nd 2023 Course</b>	<b>20</b>

---

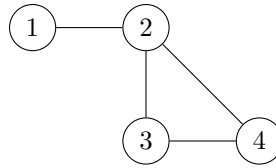
<b>Wednesday February 22nd 2023 Exercises</b>	<b>20</b>
TD3 - Exclusion Mutuelle . . . . .	20
Exercice 3 - Algorithme basé sur la circulation d'un jeton de Ricart-Agrawala . . . . .	20
<b>Wednesday February 22nd 2023 Course</b>	<b>21</b>



## Wednesday February 1st 2023

Un graphe:  $G(V,E) \rightarrow$  (ensemble sommet, ensembles arêtes)

$V=1,2,3,4 \mid E=[(1,2),(2,3),(3,4)(2,4)]$



degré d'un sommet  $x \in V$

$d(x)$ =nombre de ses voisins dans le graphe

exemple:  $d(2)=3$

soit un graphe à  $n$  sommets

$G=(V,E)$ ,  $|V|=4$  (ordre du graphe), max: $n-1$  min:0

### Exercice: Modelisation d'un problème

Montrer que dans un groupe de personnes (***n noeuds***), il y a toujours 2 personnes qui connaissent (***arêtes***) le même nombre de membres d'un groupe.

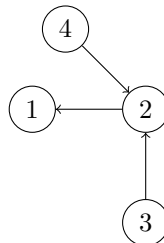
Par l'absurde, opposons que les  $n$  noeuds ont tous un degré différent

$\rightarrow$  contradiction: un noeuds doit avoir un degré  $n-1$

$\rightarrow$  Un noeud doit avoir un degré 0 ou il y a  $n$  noeuds et  $n$  degrés différents possibles or ces 2 noeuds sont pas connecté.

**Graphe orienté:**Un graphe dans lequel les arêtes (*arc*) ont une direction.

$G=(V,A) \rightarrow$  (sommets,arcs)



$V=1,2,3,4 \mid A=(2,1) (3,2) (4,2) \rightarrow$  (*origine,extrémité*)

Pour le sommet  $x$

degré entrant: nombre d'arcs dans lequel  $x$  extrémité

degré sortant: nombre d'arcs dans lequel  $x$  origine

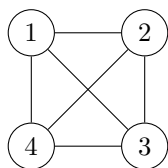
$d^+(x) \mid d^-(x)$

### Exemple de graphes:

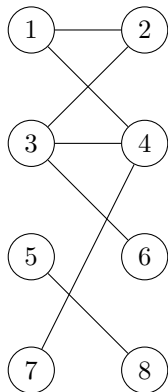
- **graphe complet:** toutes les arêtes sont presentes

$$\text{nombre d'arête} = \frac{n(n-1)}{2}$$

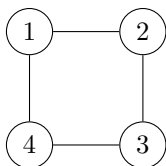




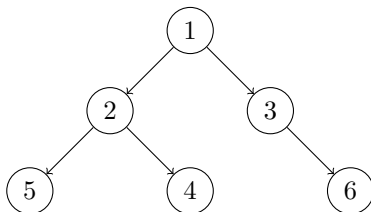
- *graphe biparti*:



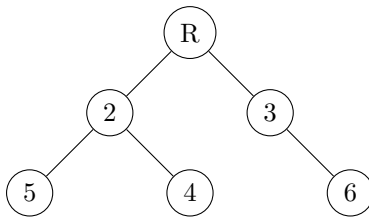
- **Arbres**: graphe qui n'a pas de cycles.  
cycle:



Nombre d'arêtes dans un arbre, sans sommet de degré 0:  $n-1$



- **arbres enracinés**: Arbres don orienté dans lequel on distingue un noeud racine.

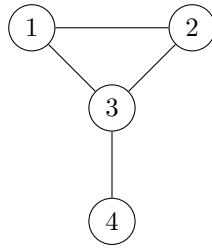


Feuille d'un arbre: sommets de degré 1 qui n'est pas la racine.

**Chemin**: ensemble de sommets  $x_1, x_2, x_3, \dots, x_{n-1}, x_n$

tel que  $(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$  sont des arêtes





**longueur chemin:** nombre de ses arêtes

**père d'un noeud  $x$ :** soit  $h(x)$  sa hauteur, Son père est son voisin dont la hauteur vaut  $h(x)-1$

**fil d'un noeuf:** comme le père mais  $h(x)+1$

**mini exo:**

Soit un arbre à  $n$  noeuds

→ hauteur max  $\Rightarrow n-1$  (arbre chemin)

→ hauteur min  $\Rightarrow 1$  ( $n$  feuilles)

→ hauteur quand un noeud a exactement 2 fils  $\Rightarrow 2^{h-1} \leq n \leq 2^h$

Soit un graphe avec  $n$  sommets,  $s_1, s_2, \dots, s_n$

Montrer que  $\sum_{i=1}^n d(s_i)$  est paire.

Dans la  $\sum$  des degrés: chaque arête est compté 2 fois, une fois pour chaque extrémité.

**Exercice:** Montrer que le nombre de sommets de degré impair est paire.

On sait que la somme des degrés est paire, donc pour chaque sommet de degré impair, il doit y avoir un second sommet de degrés impair.

Dans le cas où on aurait un nombre impair de sommet de degré impair, la somme des degrés serait elle aussi impair.

Montrer que dans un arbre avec + de 2 sommet, il y a au moins deux sommets de degré 1.

→ En supposant  $n \geq 2$  et un noeud de degré 1.

$$\sum_{i=1}^n d(s_i) \geq 2 \times (n-1) + 1$$

- **Complexité - notion :**

Notation de **Landeau**:  $O(f(x)) = g(x)$

- **informel:**

à partir d'un certain  $x$ , la valeur  $g(x)$  sera inférieur à  $f(x)$

//COURBE

- **formellement:**

on écrit

$$\begin{aligned} f(x) &= O(g(x)) \\ f(x) &\in O(g(x)) \end{aligned}$$

$$\text{Ssi: } \forall x \geq x_0, \exists k \in \mathbb{N} \text{ tel que } |f(x)| \leq k|g(x)|$$

**Exemple:**

$$\begin{aligned} f(x) &= x \\ g(x) &= x^x \end{aligned}$$



//COURBE 4

$$x_0 = 1 \quad k = 1 \quad f(x) = O(g(x))$$

Lors d'une compétition il y a 13 joueurs, est-ce qu'il est possible que chaque joueur participe à exactement 3 matchs.

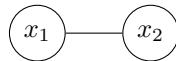
- **système distribué**: un ensemble de noeuds de calculs, autonomes interconnecté et pouvant communiquer
- **représenté par un graphe**:

1 noeud de calcul = 1 sommet

1 lien de com = 1 arête

- 1 noeud n'a accès en lecture/écriture qu'à sa propre mémoire. Tout le reste lui est envoyé sous forme de message. Un noeud a une vision locale du réseau. Il faut souvent résoudre des problèmes globaux.
- **objectif**: résoudre problèmes globaux à l'aide d'algos locaux  
 → Les noeuds vont devoir communiquer entre eux par passage de message.  
 ⇔ 1 noeud peut recevoir 1 message (on sait qui nous l'a envoyé). ⇔ 1 noeud peut envoyer 1 message à son voisin/

*mini exemple*:



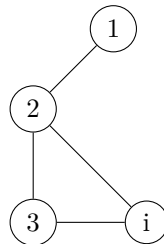
**On veut un algo**: à la fin de l'exécution chaque noeud connaît la valeur var max dans le réseau.

chaque noeud envoie son var à son voisin

**calcul**: sur réception de var faire le calcul de max (var  $x_1$ , var  $x_2$ )

## Wednesday February 8th 2023

- **2 types d'événements extérieur**:
  - a.
    - événement initial → bout de code
    - il est exécuté initialement sur un ensemble non vide, le noeud du système
    - 1 seule fois
    - il ne peut pas être exécuté sur 1 noeud qui a déjà fait du code
  - b.
    - réception



**Algo: pour le noeud i**

1. \* description des variables
  2. \* event initial: (des regles gardés)
- \* sur réception des messages: prends code classique

- **Les types de variables**:

- locals: on les indices par l'identifiant du site|noeud i



- *hookrightarrow* connaissance: elles ne sont accessibles qu'en lecture, elles servent à décider le système  
par ex: voisin  $i$ , l'ensemble des voisins du sommet
- $\leftrightarrow$  variables locales au site pour les calculs. Elles sont accessibles en lecture/écriture.
- variables de communication  
B -  
-Elle contient une info qui est transmise  
+info du site émetteur  
-elles ne sont pas indexées  
-elles sont éphémères.

### Calcul des max dans une chaîne



- **Algo pour  $i$ :**
- connaissance:  
voisin  $i$ : ensemble des voisins de  $i$  dans la chaîne  
valeur  $i$ : valeur entière du nœud  $i$
- variables:  
max  $i$ : le maximum durant calculé sur  $i$

**Initialement** (description de l'événement initial) \*sur tous les nœuds\* envoyer  $\text{Msg}(\text{val } i)$  à voisin  $i$

---

#### Algorithm 1 INIT

---

envoyer  $\text{Msg}(\text{val}_i)$  à  $\text{vois}_i$

---



---

#### Algorithm 2 Sur réception de $\text{Msg}(v)$ ( $\leftarrow$ données) de $j$ ( $\leftarrow$ nœud émetteur)

---

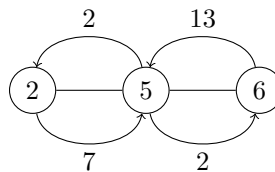
```

if  $\text{max}_i < v_i$  then
   $\text{max}_i \leftarrow v_i$ 
else if  $|\text{vois}_i| = 2$  then
  envoyer  $\text{Msg}(v)$  à  $\text{vois}_i$  sans  $\{j\}$ 
end if

```

---

**Init:**



$\text{val}_2=7; \text{val}_5=2; \text{val}_6=13$

$\text{max}_2=7; \text{max}_5=13; \text{max}_6=13$

$2 \rightarrow 5$   
 $5 \rightarrow 2 \rightarrow 6$   
 $6 \rightarrow 5$

$O(n^2)$  mémoire       $O(n+m)$  ( $m$  étant le nbres d'arêtes)

### Complexité

Configuration + Execution + complexité en temps



$A \qquad B \qquad C$

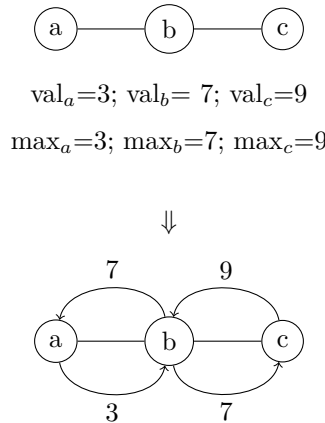
## A - Configuration

“screenshot” de votre système à 1 instant donné

- états des variables de tous les sites
- + → exo des messages en transit
- = état local de tous les sites et les messages en transit

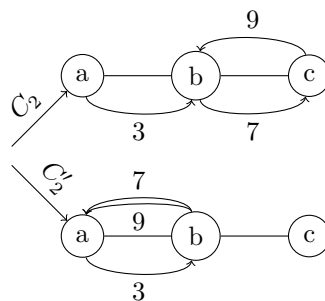
## B - Execution

- Dans 1 config donnée, 1 site est activable si il existe 1 evenement exterieur qui a été déclenché et qui est en attente de traitement
- Execution: une sequence  $C_1, C_2, \dots, C_i, C_{i+1}, \dots, C_n$  telle que entre 2 config consecutives:
- tous les sites activables ont executé les actions associées aux event exterieur (modifs des états locaux)
- les autres ne font rien



le problème qui apparait est que tous les messages ne circulent pas à la meme vitesse. Par conséquent, ils n'arrivent pas en meme temps, ce qui crée de l'indeterminisme.

La complexité en tmps est donc le tmps de la plus longue execution possible.



$C_2$  : Si msg(7) de b est reçu par a

$C'_2$  : Si msg(7) de b est reçu par c, msg(9) reçu par b

→ On observe qu'il existe plusieurs executions possibles pour 1 meme algo/reseau. Cela est du à un non determinisme provoqué par le tps de transit des msg qui est variable.

cas de figure: (execution asynchrone)





La complexité en tps va être la longueur de la plus longue exécution possible, parmi toutes les exécutions.

b - hypothèse synchrone:

- l'ensemble des msg en transit est réceptionné en 1 unité de tps
- Entre 2 config, on a 1 round
- → tous les msg sont réceptionnés + toutes les règles gardées associées ...

↓

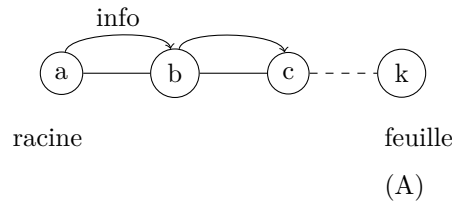
le round se termine et on est dans 1 round config.

Dans cette hypothèse: la complexité en tps = nbr de noeuds (il n'y a plus qu'une exécution possible)

### Terminaison des algorithmes

1 algo termine quand dans toutes les exécution il existe une config contenant aucun msg en transit et dans laquelle aucun site n'a de règles gardées à vrai

- terminaison explicite: Il existe au moins 1 site qui sait que l'algorithme se termine  $\leftrightarrow$  c'est indiqué stop-global dans le code.
- terminaison locale: 1 site sait qu'il a terminé sans que la terminaison soit associée par le reste des noeuds → dénoté dans le code par stop-local



### Algo pour 1 noeud i

#### Connaissance (lecture)

- $vois_i$ : les voisins de i dans la chaîne
- $estRacine_i$ : bool (=1 pour noeud racine, 0 sinon)
- $val_i$ : info à diffuser → null pour tous les noeuds sauf la racine

**Variables** \*  $Save_i$ : sert à stocker l'info à diffuser. Vaut null initialement.

---

#### Algorithm 3 INIT

---

```

if  $estRacine_i == 1$  then
    envoyer  $Msg(info_i)$  à  $vois_i$ 
     $save_i \leftarrow info_i$ 
end if

```

---



---

#### Algorithm 4 Sur réception de message $Msg(v)$ de j

---

```

if  $vois_i == 1$  then ▷ (A)
     $save_i \leftarrow v$ 
    stop-global
end if
if  $vois_i == 2$  then
     $save_i \leftarrow v$ 
    envoyer  $Msg(v)$  à  $vois_i$  sans  $\{j\}$ 
    stop-global
end if

```

---

complexité en msg: n-1

en temps ..



- synchrone:  $n-1$
- asynchrone:  $n-1$

## Exercie 2

1.

---

**Algorithm 5** Sur reception de message  $\text{Msg}(v)$  de  $j$

---

```

if  $\text{vois}_i == 1$  then ▷ (A)
     $\text{save}_i \leftarrow v$ 
    envoyer  $\text{Retour}(u)$  à  $j$ 
    stop-global
end if
if  $\text{vois}_i == 2$  then
     $\text{save}_i \leftarrow v$ 
    envoyer  $\text{Msg}(v)$  à  $\text{vois}_i$  sans  $\{j\}$ 
    stop-global
end if

```

---

complexité:  $O(2n) \rightarrow O(n)$

---

**Algorithm 6** Sur reception de  $\text{Retour}(v)$  de  $j$

---

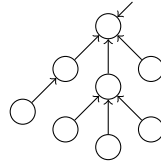
```

if  $\text{estRacine}_i == 1$  then
    stop-global
end if
if  $\text{vois}_i == 2$  then
    envoyer  $\text{Msg}(v)$  à  $\text{vois}_i$  sans  $\{j\}$ 
    stop-global
end if

```

---

la racine doit recevoir une info de chacune des feuilles



### Connaissance:

$\text{père}_i$  le père du noeud $_i$ . Vaut null pour la racine

$\text{fils}_i$  les fils du noeud $_i$ : vaut ndef pour les feuilles

$\text{info}_i$ : l'information à diffuser: ndef sauf pour les noeuds

### Variables:

$\text{cpt}_i$ : compter le nombre de messages reçus

$\text{save}_i$ : init à null, sert à sauvegarder l'info

---

**Algorithm 7** INIT

---

```

if  $\text{fils}_i == 0$  then
    envoyer  $\text{Msg}(\text{info}_i)$  à  $\text{père}_i$ 
    stop-local
end if

```

---

complexité en message:  $n-1$  : 1 message par canal de comm



**Algorithm 8** Sur réception de  $\text{Msg}(v)$  de  $j$ )

---

```

cpti++
if cpti == |filsi| then
  envoyer Msg(infoi) à pèrei
  savei ←  $v$ 
  if pèrei ≠ null then
    envoyer Msg( $v$ ) à pèrei ▷ top-local
  else
    stop-global
  end if
end if

```

---

$n$  noeuds  $\Rightarrow n-1$  arêtes

en temps: hauteur de l'arbre

2.

**Connaissance:**

estRacine<sub>i</sub>

vois<sub>i</sub> voisins du sommet

info<sub>i</sub>: l'information à diffuser: ndef sauf pour les noeuds

**Variables:**

cpt<sub>i</sub>: compter le nombre de messages reçus

save<sub>i</sub>: init à null, sert à sauvegarder l'info

**Algorithm 9** INIT

---

```

if voisi == 1 then
  envoyer Msg( $v$ ) à voisi
  pèrei ← voisi
  filsi ← null
end if

```

---

**Algorithm 10** Sur réception de  $\text{Msg}(v)$  de  $j$ )

---

```

cpti++
ajouter  $j$  dans fils  $i$ 
if estRacinei == 0 then
  if |voisi| - cpti == 1 then
    envoie msg( $v$ ) à voisi sans filsi
    père ← voisi sans filsi
    stop-local
  end if
else
  if pèrei == null AND cpt == |voisi| then
    stop-global
  end if
end if

```

---

**Exercice 4****Connaissance:**

vois<sub>i</sub>



estRacine<sub>i</sub>

info<sub>i</sub>

**Variables:**

père<sub>i</sub> = *NULL*

fil<sub>i</sub> = []

save<sub>i</sub>: init à null, sert à sauvegarder l'info

---

**Algorithm 11** INIT

---

```

if estRacine then
    envoyer Msg(vali) à voisi
end if

```

---



---

**Algorithm 12** Sur réception de Msg(v) de j)

---

```

if estRacinei == 1 then
    fili ← fili + j
    if |fili| == |voisi| then stop-global
    end if
else if pèrei == null AND cpt == |voisi| then
    stop-global
end if

```

---

## Exercice 5

**Consigne:** Soit un arbre enraciné dans lequel on a les ???. père<sub>i</sub>, fil<sub>i</sub>. On demande 1 algo tel qu'à la fin de l'exécution, la racine connaisse le nombre de noeuds de l'arbre avec 2 fils.

## Wednesday February 15th 2023

### Causalité et horloges distribuées

- Dans un système réparti, l'ordre dans lequel surviennent les événements est primordial.
- Il est nécessaire de définir des méthodes algorithmique qui permettent des relations causales entre les événements.
- Lanport qui a introduit ce concept en 1978 dans son papier. Il a cherché à créer un ordre partiel sur les événements dans un système distribué.

### Modèle proposé par Lanport

Soit  $\pi = p_1, p_2, \dots, p_n$  un système distribué avec n noeuds.

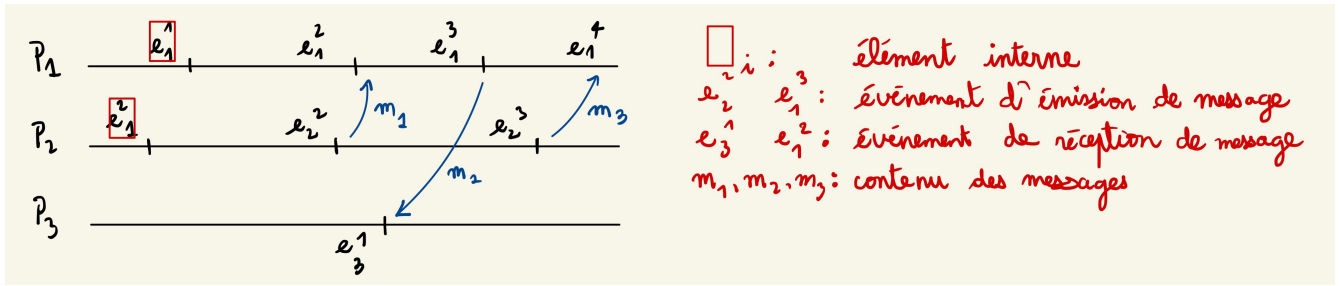
- chaque site va exécuter une séquence ordonnée d'événements  $p_1 : e_1^1, e_1^2, \dots, e_1^h$  ( $e^x$  avec x le numero de l'événement).

$e_i^1$  et  $e_i^2$  sont locaux au processeur  $i$  et  $e_i^1 < e_i^2$  } ordonnées: l'événement  $e_i^1$  apparait avant  $e_i^2$

Les événements qui apparaissent sur 1 sit sont de la forme suivante:

- réception d'un message
- envoi d'un message
- calcul interne





On va écrire qu'un événement  $e$  précède causalement un événement  $e'$  et on écrira:

$$e \rightarrow e'$$

ssi:

- $e$  et  $e'$  sont locaux au même processeur et  $e$  apparait avant  $e'$ .

**Exemple:**  $e_1^2 \rightarrow e_1^3$

- $\exists$  un message  $m$  tel que  $e = \text{émission}(m)$  et  $e' = \text{reception}(m)$ . (passage de message)

**exemple:**  $e_1^3 \rightarrow e_3^1$

- $\exists e''$  tel que  $e \rightarrow e''$  et  $e'' \rightarrow e'$  (transitivité)

**exemple:**

$$e_1^2 \rightarrow e_3^1 \text{ car } \begin{cases} e_1^2 \rightarrow e_1^3 & (\text{local}) \\ e_1^3 \rightarrow e_3^1 & (\text{message}) \end{cases}$$

Il peut aussi exister des événements qui ne sont pas liés causalement:

**exemple:**  $e_2^1$  et  $e_1^1$

ils sont dits concurrents et on écrira:

$$e_2^1 || e_1^1$$

## Algorithme de Lanport

A chaque événement  $e$  du site  $p_i$  on va associer une horloge  $H(e) = (h_i, i)$

$\rightarrow$  c'est un couple  $H(e) = (\text{compteur}, id(\leftarrow e' id \text{ du site } ))$

**init:**

- $h_i \leftarrow 0$  -event local  $e$  à  $p_i$  :  $h_i \leftarrow h_{i+1} H(e) = (h_i, i)$
- envoi de message  $m$  par  $p_i$  :  $h_i \leftarrow h_{i+1}, H(e) = (h_i, i)$ , envoi  $(m, h_i)$
- reception de message  $(m, h)$  par  $p_i$  :  $h_i = \max(h_i, h) + 1, H(e) = (h_i, i)$

---

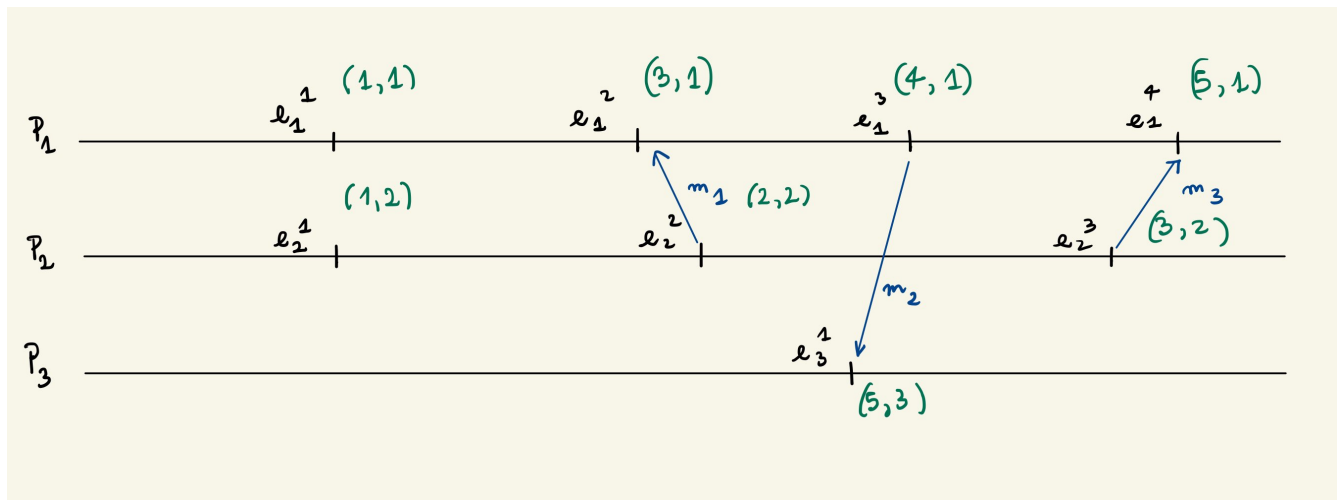
**Algorithm 13** Pour chacun des sites  $p_i$ )

---

**init:**

- $h_i \leftarrow 0$
  - event local  $e$  à  $p_i$  :  $h_i \leftarrow h_{i+1} H(e) = (h_i, i)$
  - envoi de message  $m$  par  $p_i$  :  $h_i \leftarrow h_{i+1}, H(e) = (h_i, i)$ , envoi  $(m, h_i)$
  - reception de message  $(m, h)$  par  $p_i$  :  $h_i = \max(h_i, h) + 1, H(e) = (h_i, i)$
- 





$H(e) < H(e')$  ssi:

$$\begin{cases} H(e).h < H(e').h \text{ ou} \\ H(e).h = H(e').h \text{ et} \\ H(e).id < H(e').id \end{cases}$$

**1ere remarque:** les horloges de 2 events sont tjrs differentes

**2e remarque:** L'horloge de Lamport relie la relation suivante:

$$H(e) < H(e') \implies (e \rightarrow e') \text{ ou } (e || e')$$

$$\text{et } e \rightarrow e' \implies H(e) < H(e')$$

**Exemples:**

$$H(e_1^1) < H(e_2^1)$$

$$H(e_1^2).h = 1$$

$$H(e_2^1).h = 1$$

et

$$H(e_1^2).id = 1$$

$$H(e_2^1).id = 2$$

et on a  $e_1^1 || e_2^1$  (concurrents)

$$H(e_1^1) < H(e_1^2) \text{ et } e_1^1 \rightarrow e_1^2$$

## Construction ordre global sur les événements

on va avoir un ordre total

$$e < e' < e'' \text{ sur les events}$$

$e < e'$  indique que l'ordre partiel suivant est respecté:

$$e \rightarrow e'$$

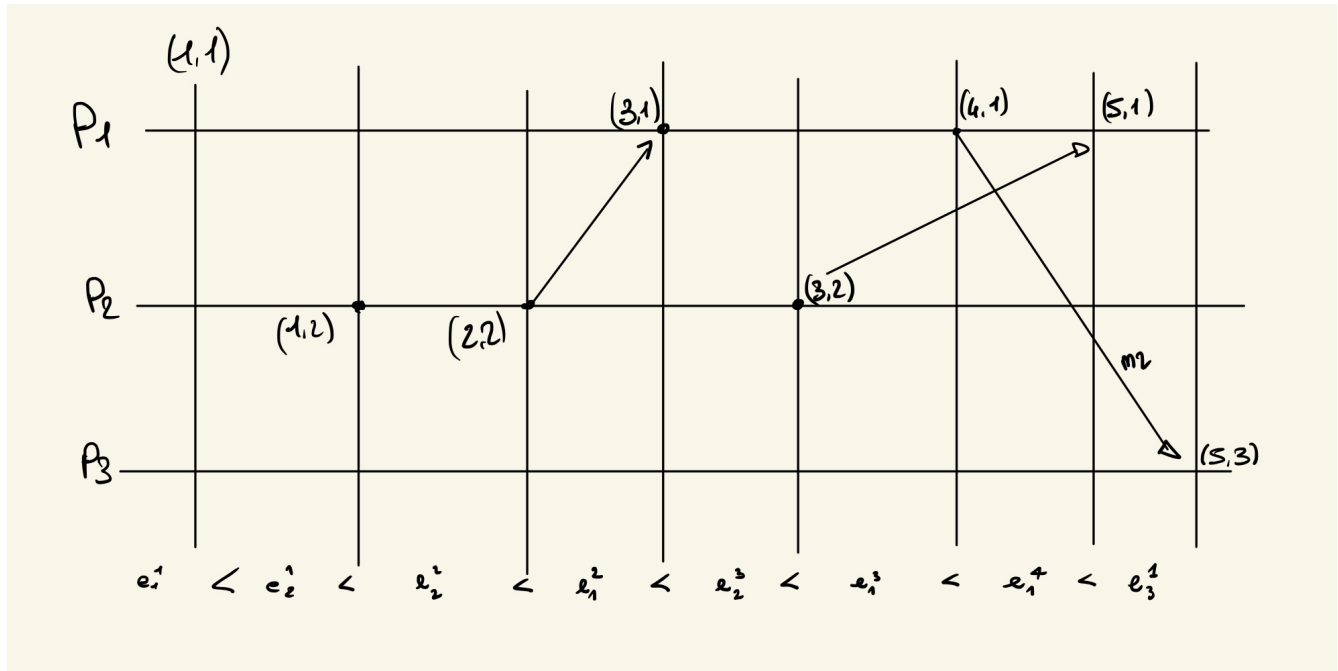
$$e || e'$$

$$e < e' \implies (e \rightarrow e') \text{ ou } (e || e').$$



## Construction de l'ordre totale

On ordonne les événements par leur valeur d'horloge.



## Wednesday February 15th Exercises

### TD2 - Horloges

Avec cette construction on aura l'équation suivante:

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

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

---

**Algorithm 14** 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 événement  $e$  on associe une valeur 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  $D$ :

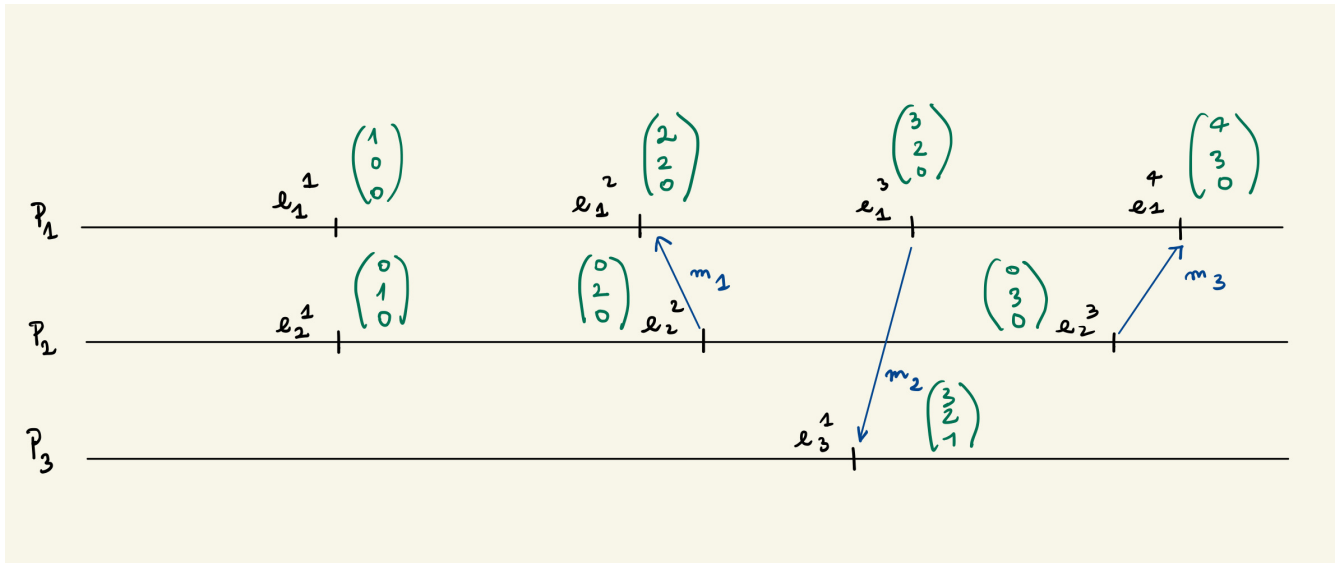
**for** chaque case  $j \neq i$  **do**

$$V_i[j] \leftarrow \max(V_i[j], D[j])$$

**end for**

---





Comment comparer 2 horloges ?

$$\begin{aligned}
 V \leq V' & \text{ssi } \forall j V[j] \leq V'[j] \\
 V \leq V' & \text{ssi } V \leq V' \text{ et } \exists k \text{ tq } V[k] < V'[k] \\
 V \parallel V' & \text{ssi } \neg(V \leq V') \cap \neg(V' \leq V)
 \end{aligned}$$

Exemple: Horloges incompatibles

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

Question 1

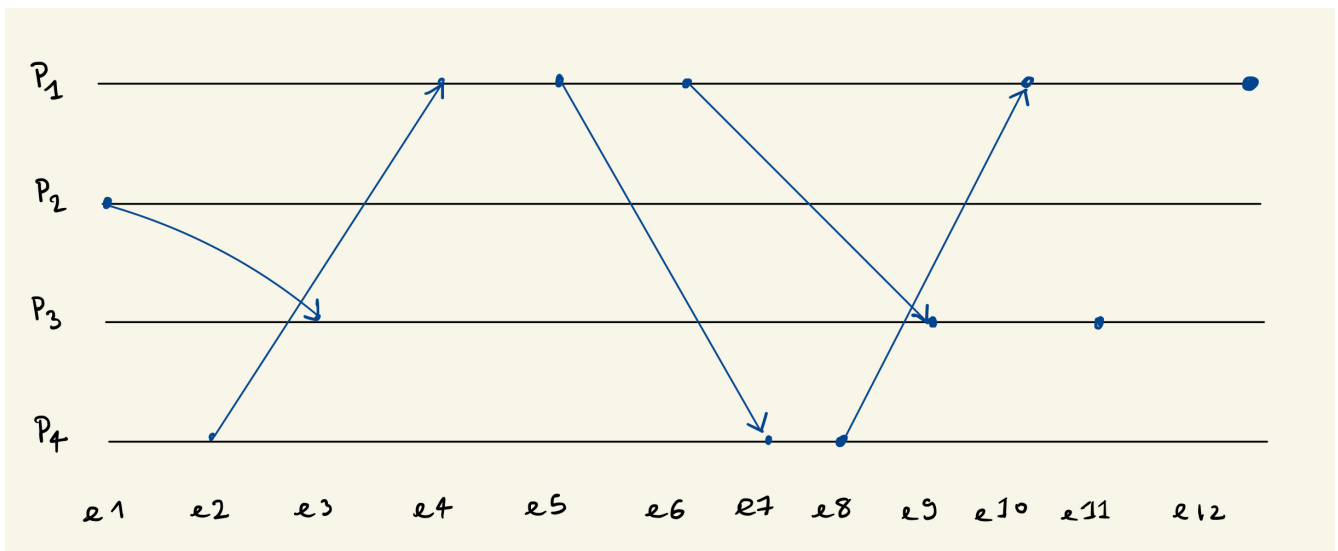


Figure 1: Diagramme Question 1

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<sup>er</sup> possible.
- 2<sup>e</sup>:  $e_5$  avant  $e_4$  pas possible car  $e_4 \rightarrow e_5$  (local).

**Question 4**

$$\begin{array}{lll} e_9 & \rightarrow & e_{11} \text{ (local)} \\ e_5 & \rightarrow & e_7 \text{ (message)} \\ e_1 & \rightarrow & e_{11} \text{ (transitivité)} \\ e_2 & \rightarrow & e_{11} \text{ (transitivité)} \\ e_1 & || & e_5 \text{ (!concurrent)} \\ e_7 & || & e_{11} \text{ (!concurrent)} \end{array}$$

**Question 5**

Les événements précédant  $e_9$  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]

**Wednesday February 15th Courses****Exclusion Mutuelle**

- a. Spécification du problème
- b. 2 algos basés sur des permissions
- c. 2 algos basés sur de la circulation de jetons

Un algo d'exclusion mutuelle doit vérifier les 2 propriétés suivantes:

- 1. Sûreté: la Section critique n'est pas accéder en parallele par des sites
- 2. Vivacité: un site qui va vouloir entrer en section critique doit le faire en temps fini.

**Algo Ricart-Agrawala avec permission****Principe:**

- Chaque site  $i$  va avoir une date de demande d'entrer sur sit ( $last_i$ )
- Un noeud va entrer en sc s'il reçoit la permission de tous ses voisins.

**Hypothèse sur le réseau:**

- le réseau forme un graphe complet, il existe un lien de com entre toutes les paires de sommets.
- identifié: tous les noeufs ont un numéro
- asynchrone: les messages arrivent en temps fini.



**Principe de l'algo**

- Un noeud qui veut rentrer en section critique va diffuser sa demande à ses voisins. Il va envoyer sa date<sub>i</sub> de dernière demande d'entrer en SC.
- Un voisin va lui accorder la permission. Si il veut aussi entrer en sc, il va accorder la permission si sa date de demande est antérieure.

**types de messages:**

- Perm: Quand un noeud accorde la permission d'entrer en sc à un autre.
- Dem(h,j): requete de demande d'entrée en sc avec h la date de j.

**Wednesday February 15th 2023 Exercises****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 faire 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  $3 < 4$

**Courses**

algo d'exclusion mutuelle avec un jeton

Hypothese: no va avoir un réseau en commun.

//TORE ORIENTE

Algo: un jeton unique circule sur l'anneau

Connaissance : successive (le voisin dans l'anneau)

variables:  $etat_i = \{S, SC, E\}$

Algo:

- sur demande d'entrée en sc
- sur sortie de sc
- sur reception du jeton de sc



## Wednesday February 22nd 2023 Exercises

### TD3 - Exclusion Mutuelle

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

---

**Algorithm 15** Variables et initialisations

---

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

---

---

**Algorithm 16** Sur réception de  $Dem(h', j)$  de  $j \rightarrow$ 

---

```
Envoyer  $Perm$  à  $j \dots$ 
 $attendu_i \leftarrow attendu_i \cup \{j\}$ 
if  $etat = E$  then
    envoyer  $Dem(last_i, i)$  à  $j$ 
end if
```

---

---

**Algorithm 17** Sur sortie de section critique  $\rightarrow$ 

---

```
Envoyer  $Perm$  à  $j \dots$ 
 $attendu_i \leftarrow j \dots$ 
```

---



## Wednesday February 22nd 2023 Course

On veut faire de l'exclusion mutuelle dans un reseau en anneau

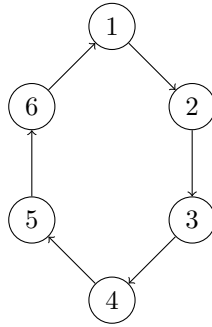


Figure 2: Orienté, les messages circulent dans une seule direction.

**L'idée:** il y a un jeton unique qui circule sur l'anneau. Le noeud qui a le jeton est le seul à pouvoir entrer en sc.

**Connaissance:**  $succ_i$ : le voisin successeur dans l'anneau avec qui on peut communiquer

**variables:**  $etat_i$ : s: sortie | sc: section critique | E: demande

---

### Algorithm 18 Algo de Lelann

---

**Sur demande d'entrée en SC:**

$etat \leftarrow E$

**Sur réception de jeton:**

**if**  $etat = E$  **then**

$etat \leftarrow sc$

**else**

    envoyer jeton à  $succ_i$

**end if**

**Sur sortie de sc:**

$etat \leftarrow s$

envoyer jeton à  $succ_i$

---

Algorithme d'exclusion mutuelle basé sur *une circulation de jeton*.

### Ricard-Agrawala

**Hypothèse:** réseau complet, identifié

**Principe:** Le jeton est un tableau avec autant de cases que de sites. Chaque case contient le nombre de demande du site associé à la case lorsque le jeton était sur le site pour la dernière fois.

## Wednesday February 22nd 2023 Exercises

### 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	

## Wednesday February 22nd 2023 Course

**Hypothèse:** anneau orienté + identifié + asynchrone

**Principe:**

- au départ, tous les sites sont dans l'état initial
- événement initial: certains noeuds se déclarent spontanément candidats.
- les candidats envoient leurs id à leur successeurs
- un candidat conserve les ids reçus
- lorsqu'un candidat reçoit son propre id, il se déclare leader si cet id est le minimum parmi ceux reçus.
  - Tous les messages font le tour de l'anneau.

**connaissance:**

- $succ_i$

**variables:**

- $max_i$  initialisé à  $id_i$
- $etat_i$ : {candidat, leader, battu, init}

**Messages:**  $Msg(id)$

---

### Algorithm 19 Algo

---

**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**

    envoyer jeton à  $succ_i$

**end if**

**Sur sortie de sc:**

$etat \leftarrow s$

envoyer jeton à  $succ_i$

---

