Algorithmique et programmation distribuée

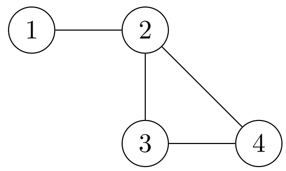
Zakaria EJJED

Contents

Wednesday February 1st 2023

Un graphe: G(V,E) → (ensemble sommet, ensembles arêtes)

V=1,2,3,4 | E=[(1,2),(2,3),(3,4)(2,4)}



degré d'un sommet x ∈ 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.

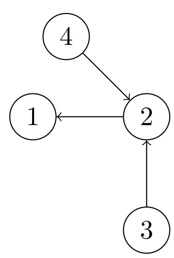
Par l'absurde, opposons que les n noeuds ont tous un degré différent

→ contradiction: un noeuds doit avoir un degré n-1

→ 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) → (sommets,arcs)



V=1,2,3,4 | A =(2,1) (3,2) (4,2) → (*origine*,*extremité*)

Pour le somemt x

degré entrant: nombre d'arcs dans lequel x extremité

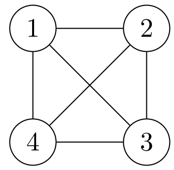
degré sortant: nombre d'arcs dans lequel x origine

$d^+(x)$ | $d^-(x)$

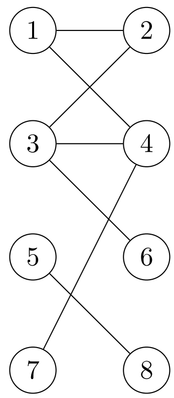
Exemple de graphes:

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

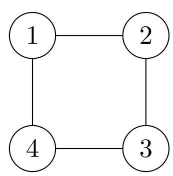
$$\text{nombres d'arête } = \frac{n(n-1}{2}$$



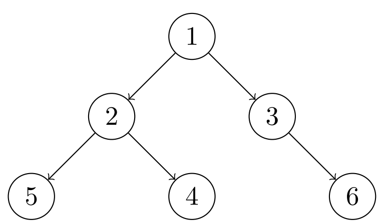
- ***graphe biparti***:



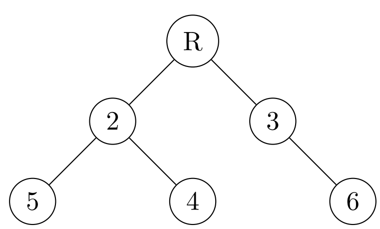
- ***Arbres***: graphe qui n'a pas de cycles.
  cycle:



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



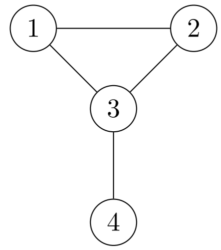
- ***arbres enracinés***: Arbres don orienté dans lequel on distingue un noeud racine.



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

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

tel que $(x_1, x_2, x_3, ..., x_{n-1}, x_n)$ 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

fils d'un noeuf: comme le père mais h(x)+1

mini exo:

Soit un arbre à n noeuds
→hauteur max ⇒ n-1 (arbre chemin)
→hauteur min ⇒ 1 (n feuilles)
→hauteur quand un noeud a exactement 2 fils ⇒ $2^{h-1} \leq n \leq 2^h$

Soit un graphe avec n sommets, $s_1, s_2, ..., s_n$

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 extremité.

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

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

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

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

→ En supposant n ≥ 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

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

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

Exemple:

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

//COURBE 4

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

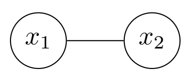
Lors d'une compétition il y a 13 joueurs, est-ce qu'il est possible que chaque joueur participe à exactement 3 matchs.

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

$$\text{1 noeud de calcul} = \text{1 sommet}$$
$$\text{1 lien de com} = \text{1 arête}$$

- 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:



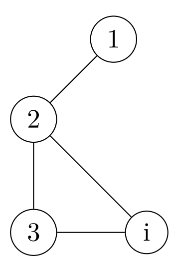
On veut un algo: à la fin de l'execution chaque noeud connait la valeur var max dans le réseau.

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'évenements exterieur:**
  a.
  -évènement initial → bout de code
  -il est executé initialement sur un ensemble non vide, le noeud du système
  -1 seule fois
  -il ne peut pas être executé 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

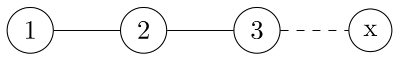
- **Les types de variables:**

- 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 voisin du sommet

- $\hookrightarrow$ variables locales au site pour les calculs. Elles sont accessible en lecture/ecriture.
- variables de communication
  B -
  -Elle contient une info qui est transmise
  +info du site emetteur
  -elles ne sont pas indices
  -elles sont ephemere.

Calcul des max dans une chaine



- **Algo pour i:**
- connaissance:
  voisin i: ensemble des voisins de i dans la chaine
  valeur i: valeur entiere du noeud i
- variables:
  max i : le maximum dourant calculé sur i

Initialement (description de l'evenement initial) *sur tous les noeuds* envoyer Msg(val i) à voisin i

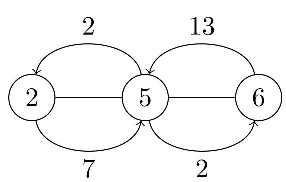
Algorithm 1 INIT

envoyer Msg(val_i) à vois_i

Algorithm 2 Sur réception de Msg(v) ($\leftarrow$ données) de j ($\leftarrow$ noeud émetteur)

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

Init:



val_2=7; val_5= 2; val_6=13

max_2=7; max_5=13; 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

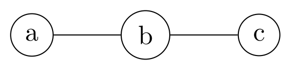
"screenshot" de votre système à 1 instant donné

$\rightarrow$ états des variables de tous les sites
$+$ $\rightarrow$ 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

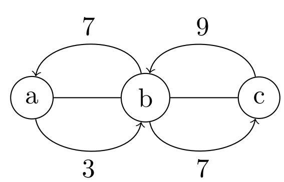
- Execution: une sequence C_1, C_2,...,C_i,C_{i+1},...,C_n
  telle que entre 2 config consecutives:
- tous les sites activables ont executé les actions associées aux event exterieur (modifs des états locaux)
- les autres ne font rien



val$_a$=3; val$_b$= 7; val$_c$=9

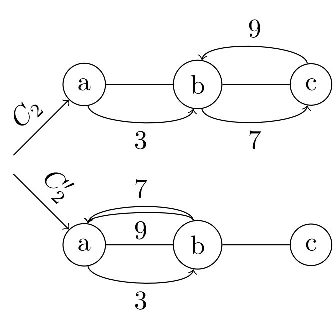
max$_a$=3; max$_b$=7; max$_c$=9

$\Downarrow$



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 exectution 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

$\rightarrow$ On observe qu'il existe plusieur execution possible 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 etre la longueure de la plus longue execution possible, parmis ttes les executions.

b - hypothèse synchrone:

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

$$\downarrow$$

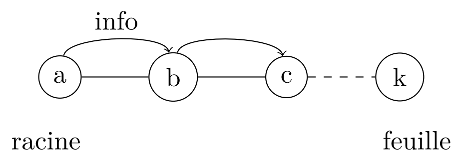
le round se termine et on est dans 1 round config.

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

Terminaison des algorithmes

1 algo termine quand dans toutes les exec il existe une config contenant aucun msd en transit et dans lequel aucun site n'a de regle gardées à vrai

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



(A)

Algo pour 1 noeud i

Connaissance (lecture)

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

Variables * Save_i: sert à stocker l'info à diffuser. Vaut ndef initialement.

Algorithm 3 INIT

if $\text{estRacine}_i == 1$ **then**
  envoyer $\text{Msg}(\text{info}_i)$ à vois_i
  $\text{save}_i \leftarrow \text{info}_i$
end if

Algorithm 4 Sur reception de message Msg(v) de j)

if $\text{vois}_i == 1$ **then** ▷ (A)
  $\text{save}_i \leftarrow v$
  stop-global
end if
if $\text{vois}_i == 2$ **then**
  $\text{save}_i \leftarrow v$
  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 Msg(v) de j)

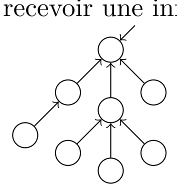
if $\text{vois}_i == 1$ **then** ▷ (A)
  $\text{save}_i \leftarrow v$
  envoyer Retour(u) à j
  stop-global
end if
if $\text{vois}_i == 2$ **then**
  $\text{save}_i \leftarrow v$
  envoyer Msg(v) à vois_i sans {j}
  stop-global
end if

compléxité: O(2n) $\rightarrow$ O(n)

Algorithm 6 Sur reception de Retour(v) de j)

if $\text{estRacine}_i == 1$ **then**
  stop-global
end if
if $\text{vois}_i == 2$ **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 7 INIT

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

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

Algorithm 8 Sur réception de Msg(v) de j)

cpt_i++
if cpt_i ==$|\text{fils}_i|$ **then**
  envoyer Msg(info_i) à père_i
  $\text{save}_i \leftarrow v$
  if $\text{père}_i \neq \textit{null}$ **then**
    envoyer Msg(v) à père_i ▷ top-local
  else
    stop-global
  end if
end if

n noeuds ⇒ n-1 arêtes

en temps: hauteur de l'arbre

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 9 INIT

if $\text{vois}_i == 1$ **then**
  envoyer Msg(v) à vois_i
  $\text{père}_i \leftarrow \text{vois}_i$
  $\text{fils}_i \leftarrow \textit{null}$
end if

Algorithm 10 Sur réception de Msg(v) de j)

cpt_i++
ajouter j dans fils i
if $\text{estRacine}_i == 0$ **then**
  if $|\text{vois}_i| - cpt_i == 1$ **then**
    enoie msg(v) à vois_i sans fils_i
    père $\leftarrow$ vois_i sans fils_i
    stop-local
  end if
else
  if $\text{père}_i == \textit{null}$ AND cpt == $|\text{vois}_i|$ **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 11 INIT

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

Algorithm 12 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 2023

Causalité et horloges distribuées

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

Modèle proposé par Lanport

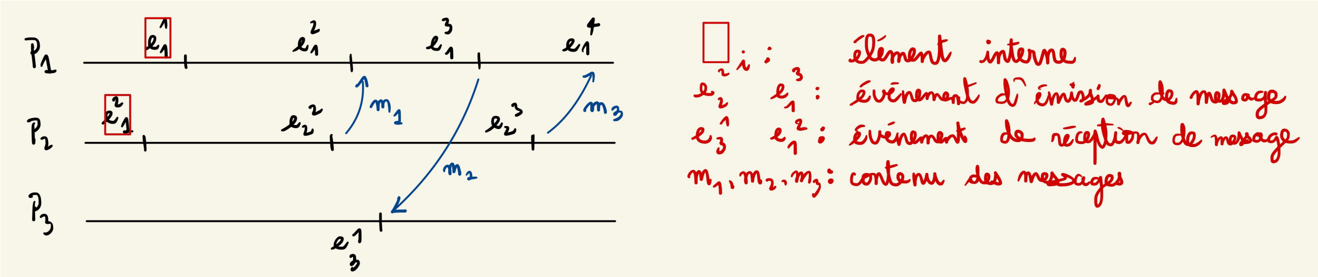
Soit $\pi = p_1, p_2, \ldots, p_n$ un système distribué avec n noeuds.

- chaque site va executer une séquence ordonnée d'évenements $p_1 : e_1^1, e_1^2, \ldots, e_1^h$ (e^x avec x le numero de l'évenement).

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

Les evenements 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 évenement e précède causalement un évenement e' et on écrira:

$$e \to e'$$

ssi:

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

Exemple: $e_1^2 \to e_1^3$

- $\exists$ un message m tel que e = émission(m) et e'= recepetion(m). (passage de message)
  exemple: $e_1^3 \to e_3^1$
- $\exists$ e" tel que $e \to e''$ et $e'' \to e'$ (transitivité)
  exemple:

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

Il peut aussi exister des evenements qui ne sont pas liées causalement:
exemple: e_2^1 et e_1^1

ils sont dits concurrents et on écrira:

$$e_2^1 || e_1^2$$

Algorithme de Lanport

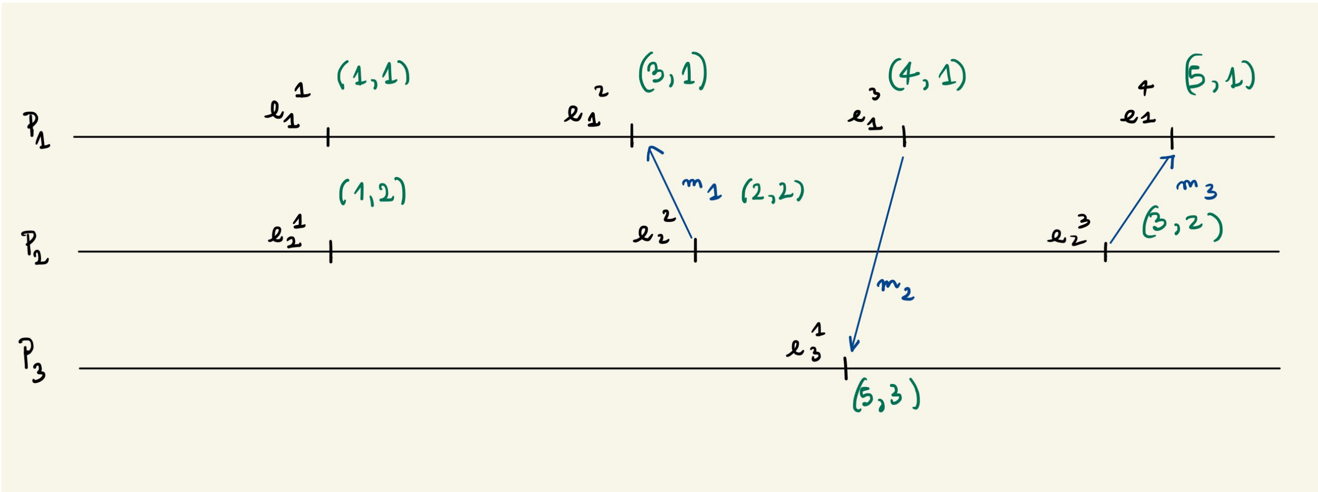
A chaque evenements e du site p_i on va associer une horloge $H(e) = (h_i, i)$

$\to$ c'est un couple $H(e) = (compteur, id(\leftarrow e'id$ 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 $p_i : hi = max(h_i, h) + 1, H(e) = (h_i, i)$

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 $p_i : hi = max(h_i, h) + 1, H(e) = (h_i, i)$



$H(e) < H(e')$ 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 Lanport rellete la relation suivante:

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

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

Exemples:

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

$$\begin{matrix} H(e_1^2).h & = & 1 \\ H(e_2^1).h & = & 1 \end{matrix}$$

et

$$\begin{matrix} H(e_1^2).id & = & 1 \\ H(e_2^1).id & = & 2 \end{matrix}$$

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

$H(e_1^1) < H(e_1^2)$ et $e_1^1 \to e_1^2$

Construction ordre global sur les évenements

on va avoir un ordre total

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

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

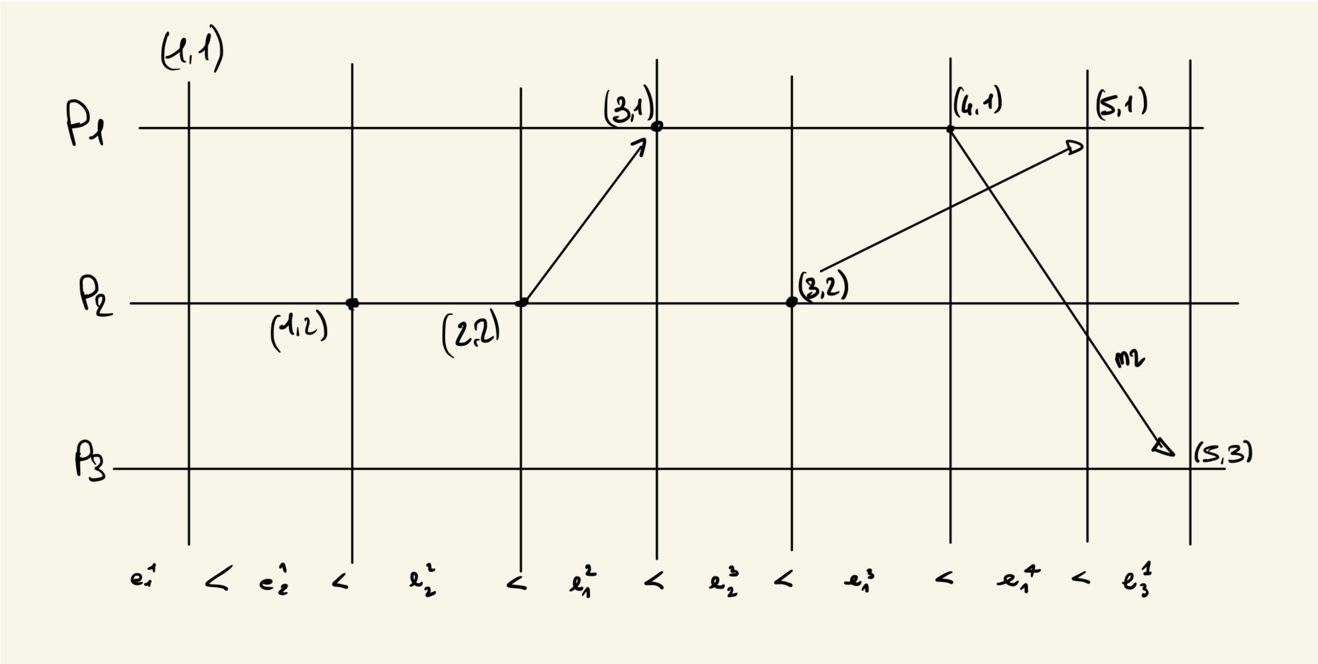
$$e \to e'$$

$$e||e'$$

$e < e' \implies (e \to e')$ ou $(e||e')$.

Construction de l'ordre totale

On ordronne les évenements par leur valeur d'horloge.



Wednesday February 15th Exercises

TD2 - Horloges

Avec cette construction on aura l'équation suivante:

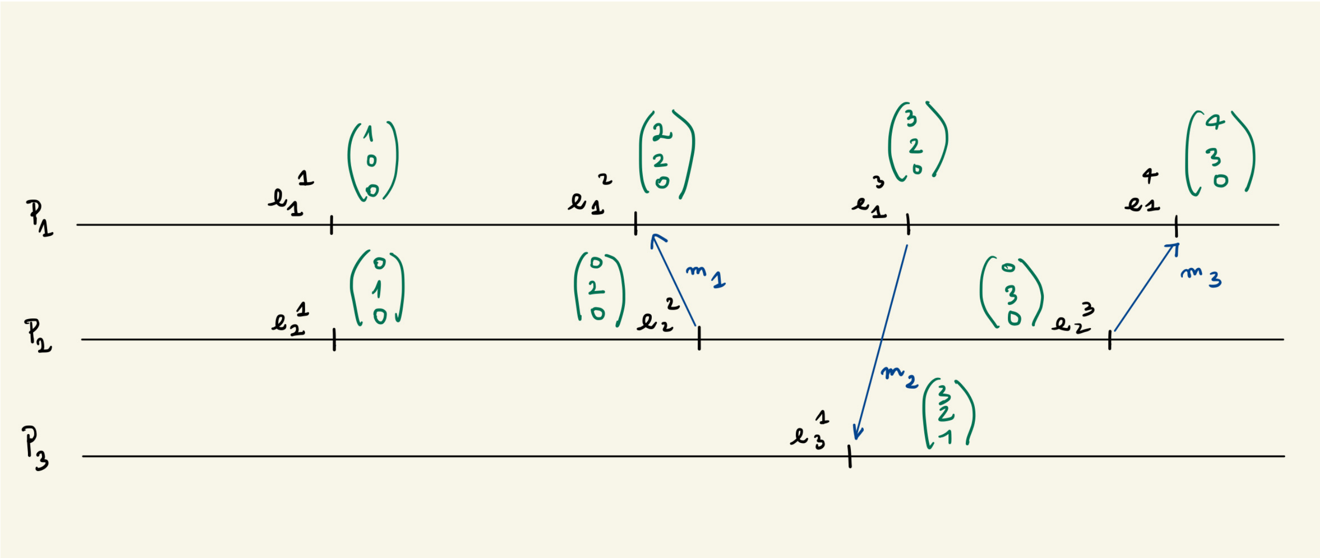
$$e \to e' \Leftrightarrow V(e) < V(e')$$

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

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

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

Question 1

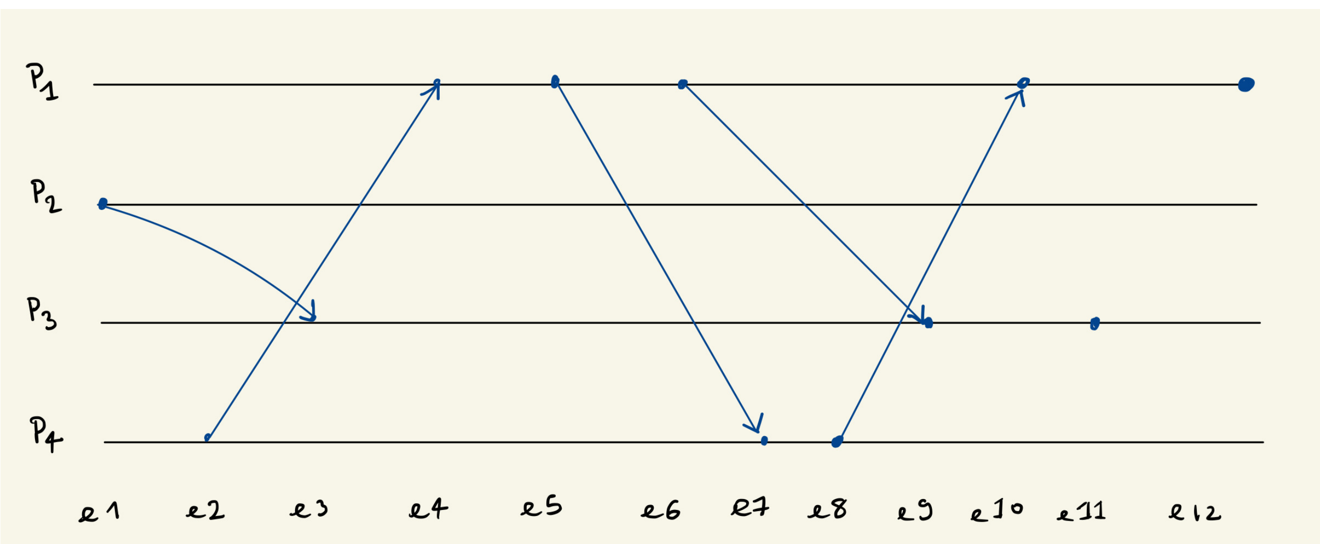


Figure 1: Diagramme Question 1

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 $e_4 \to e_5$ (local).

Question 4

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

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]

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 acceder 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 $date_i$ de derniere 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 anterieure.

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 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

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 $j \dots$
$attendu_i \leftarrow attendu_i U\{j\}$
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

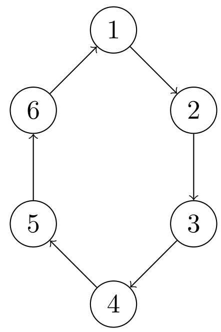


Figure 2: Orienté, les messages circulent dans une seule direction.

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 getsE
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
- évenement initial: certains noeuds se déclarent spontanement 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 de Lelann

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

Remarque 1: en supposant un anneau de taille n à k candidats.
Compléxité messages: O($n \times k$) $k = O(n)$
Compléxité round: O(n)

Remarque 2: Quand un candidat reçoit son identifiant on est sûr qu'il a reçu tous les ids de cndidats à une condition: que les canaux de com soient FIFO.

Wednesday February 22nd 2023 Exercises

TD5 - Election

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

- 1. Compléxité en message: O(?)

Algorithm 20 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 Course

Chapitre 4

Parcourir en profondeur dans le cadre général des graphes

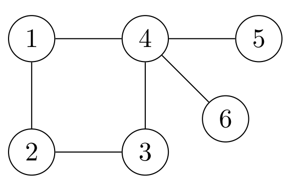


Figure 3: Exemple

On veut explorer le graphe à l'aide d'algorithme de parcours de graphe.

- Une stratiégie possible: parcours en profondeur : on explore le graphe récursivement le "plus loin" possible, avant de revenir en arrière (grâce à la récursion) pour explorer ensuite le reste.
- L'autre stratégie: exploration en largeur . C'est un algo itératif basé sur l'utilisation d'une file de priorité. Le graphe est exploré par couche.

Structure de données:
Un graphe est représenté par une matrice s'il a n sommets: $a_1, a_2, \ldots, a_n$

$$\begin{array}{c|ccc} & a_1 & \dots & a_n \\ a_1 & & & \\ \vdots & & & \\ a_n & & & \end{array} \qquad M[a_i][a_j] = 1 \text{ si } a_i - a_j \text{ est une arête et 0 sinon.}$$

Ex: écrire une fonction:
$voisin(S, G) : void$ qui affiche les voisins d'un sommet S donné dans G.

Algorithm 21 Voisin(S,G):void

for i de 1 à n **do**
  if G[S][i]==1 **then**
    Afficher i
  end if
end for

parcours en profondeur:
Il va y avoir 2 fonctions:

- La fonction principale PP(G): void
- Procédure ProcPP(G,S,marque) (G:matrice d'adjacence; S: un sommet du graphe; marque: tableau de booléen initialisé à false, taille=nbr de sommet, décrit si un sommet à déjà était exploré).

PP(G):void

- initialisé une tableau de bool marque de taille = taille(G) (le nbre de ligne de la matrice) initialisé à false (variable globale).
- Pour tout sommet S de G non marque (marque[S]==0) ProcPP(G,S,marque).

Algorithm 22 ProcPP(S,G,<marque>):void

marque[S] ← true
Afficher S
for chaque voisin t de S qui n'est pas marque (marque[t]==0) **do**
  ProcPP(G,t,marque)
end for

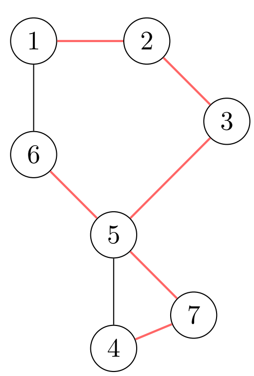


Figure 4: Parcour Profondeur en partant du sommet 1

Complexité:
Avec une matrice d'adjacence: Il faut parcourir toute la matrice pour trouver les voisins: O(n^2).

Si au lieu de la matrice d'adjacence le graphe est représenté avec des listes d'adjacences.

Wednesday March 8th 2023 Exercises

TD5 - Election

Exercice 3 - Algorithme Double Token

1.

- 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 Exercises

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

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 23 Algorithme Vitesse

Initialement:
Envoyer Token(ià à $j \in vois$
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
    $sens_i \leftarrow vois_i \backslash \{j\}$
  end if
end for
if nbticrestant == 0 **then**
  envoyer token(idmin) à sens
  nbticrestant ← -1
end if

Algorithm 24 Algorithme Vitesse

3. **Connaissance:**
fils$_i$
pere$_i$
estFeuille
estRacine
Variables:
restant
counterFeuille ← 0
counterSpecial ← 0
Initialement:
restant = fils$_i$
if estFeuille==1 **then**
  Envoyer Msg(0,"feuille")
end if
Sur reception Msg(val,s) de j:
restant ← 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") → pere
  end if
end if