TD Algorithmes distribués

Année 2022-23

Version 1.0

Université de Montpellier
Place Eugène Bataillon
34095 Montpellier Cedex 5

Rodolphe Giroudeau
161, rue Ada
34392 MONTPELLIER Cedex 5
TEL : 04-67-41-85-40
Mail : {*rgirou*}@lirmm.fr

Université de Montpellier — 7 octobre 2022
Algorithmes distribués — Durée: 15h

exclusion mutuelle, problème de l'élection, problème de la terminaison, consensus
Travaux dirigés – Séance n^o 1

---

1 Horloges

Exercice 1 – Horloges de Lamport, de Martell et matricielle

Nous considérons les horloges de Lamport définies de la manière suivante :

Une horloge est une fonction H définie à partir des événements vers un ensemble ordonné tel que $a \prec b \Rightarrow H(a) < H(b)$. L'horloge de Lamport HL affecte à chaque événement a la longueur k de la plus longue chaîne de causalité $b_1 \prec \ldots \prec b_k = a$.

Un algorithme distribué permet de calculer HL et sera basé sur les caractéristiques suivantes :

— Si a est un événement interne ou l'envoi de messages, soit k la valeur de l'horloge de l'événement précédent ($k = 0$ si il n'y a pas de événement précédent) $HL(a) = k + 1$;

— Si a est l'événement de réception, k la valeur de l'horloge de l'événement précédent ($k = 0$ si il n'y a pas de événement précédent), et $HL(b)-$ la valeur de l'horloge de l'événement b, alors $HL(a) = \max\{k, HL(b)\} + 1$.

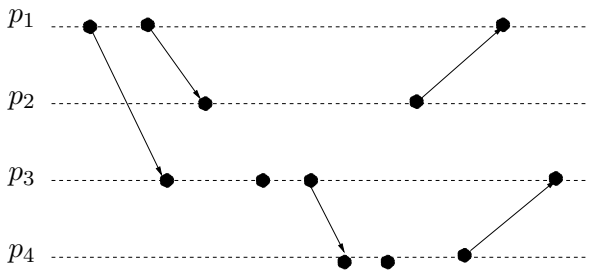


FIGURE 1 – *Sur les horloges de Lamport*

1. Donner l'algorithme qui permet de calculer les horloges logiques.
2. Donner les valeurs des horloges pour la figure 1. Avons-nous un ordre total ?
3. Que devons rajouter à la définition précédente pour créer un ordre total ?
4. Montrer que si $e \rightsquigarrow e'$ alors $H(e) \prec H(e')$. Utiliser par récurrence sur la longueur n de la suite d'évènement reliant e à e'

5. L'ordre causal est l'ordre partiel sur les événements : Si a et b sont deux événements a précède $b, (a \rightsquigarrow b)$ si et seulement si l'une des trois conditions est vraie :
   - a et b ont lieu sur le même site avec a avant b,
   - $a = ENVOYER(< M >)$ et $b = RECEVOIR(< M >)$ du même message,
   - Il existe un évènement c tel que $a \rightsquigarrow c$ et $c \rightsquigarrow b$

   (a) Soit la figure 2

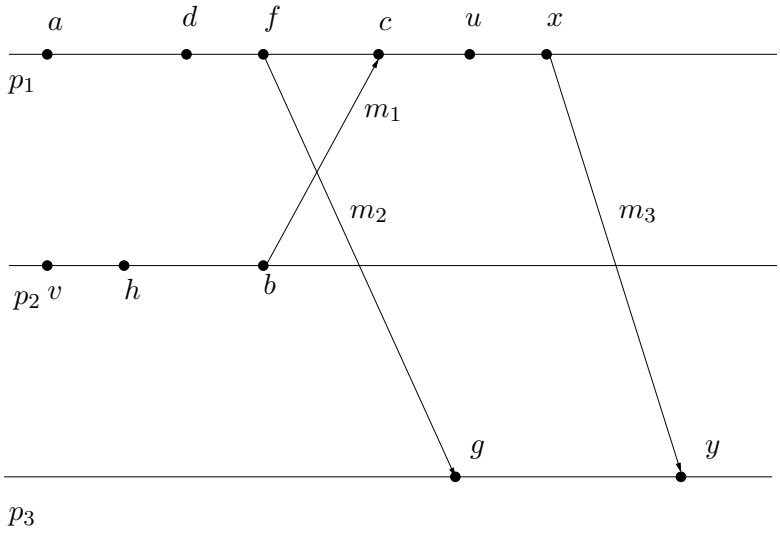


FIGURE 2 – *Détermination des relation de causalité*

   Donner les relations de causalité.

   (b) A un événement e trois ensembles d'événements sont associés :
   - $PASSE(e)$: ensemble des événements antérieurs à e dans l'ordre causal (e appartient à cet ensemble)
   - $FUTUR(e)$: ensemble des événements postérieurs à e dans l'ordre causal (e appartient à cet ensemble) ;
   - $CONCURRENT(e)$: ensemble des événements concurrents avec e.

   Notation : $e||e'$ deux événements e et e' sont concurrents
   Sur la figure 2 donner $PASSE(u), FUTUR(u), CONCURRENT(u)$.
   Délivrance causale : La délivrance causale assure que si l'envoi du message m_1 par le site S_i à destination du site S_k précède (causalement) l'envoi du message m_2 par le site S_j à destination du site S_k, le message m_1 sera délivré avant le message m_2 sur le site S_k. Formellement,

$$send_i(m_1, k) \rightsquigarrow send_j(m_2, k) \rightarrow del_k(m_1) \rightsquigarrow del_k(m_2)$$

6. Considérons maintenant la notion de vectorielle proposé par Mattern.

   — Chaque site i gère un vecteur d'entiers de n éléments (une horloge HV_i avec n le nombre de sites ;

   — Chaque message m envoyé est estampillé (daté) (EV_m) par la date de son évènement :

      — Si un événement est locale, alors $HV_i[i] \leftarrow HV_i[i] + 1$;

      — Si l'événement est l'envoi d'un message m alors

$$\begin{cases} HV_i[i] \leftarrow HV_i[i] + 1 \\ EV_m \leftarrow HV_i \end{cases}$$

      — Si l'événement est la réception du message m alors

$$\begin{cases} HV_i[i] \leftarrow HV_i[i] + 1 \\ HV_i[j] \leftarrow \max(HV_i[j], EV_m[j], j \neq i \end{cases}$$

   (a) Donner l'évolution des horloges vectorielles pour le graphe de la figure 3.

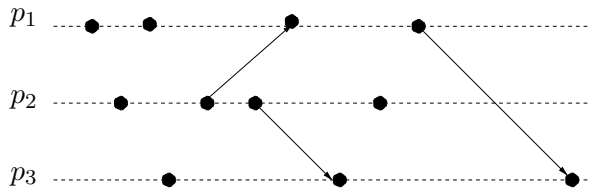


FIGURE 3 – *Sur les horloges vectorielles*

   (b) Montrer que la valeur de la $i$ème composante de EV_e correspond au nombre d'évènements du site S_i appartenant au passé de e

$$\forall i, EV_e[i] = card(\{e' : e' \in S_i \wedge e' \rightsquigarrow e\})$$

   (c) Compléter la relation d'ordre suivante :

$$EV_e \preceq EV_{e'} \text{ ssi}$$

   (d) Montrer que $e \rightsquigarrow e'$ ssi $EV_e \preceq EV_{e'}$.

   (e) Montrer que $e||e'$ ssi $EV_e||EV_{e'}$ avec $e||e'$ désigne deux événements e et e' concurrents.

   (f) Cet ordre permet-il la délivrance causale ?

7. Sur l'estampille matricielle :

   — Chaque site i gère une horloge (notée HM_i) matricielle ($n \times n$) avec n le nombre de sites

   — Chaque message m envoyé est estampillé (daté) (EM_m) par la valeur courante de HM_i

— Que signifie $HM_i[j,k]$?

  — $HM_i[j,k]$ = nombre de messages issus de p_j vers p_k dont p_i a connaissance ;

  — $HM_i[i,i]$ correspond au nombre d'évènements locaux du site i

— Comment HM_i est modifiée

  — Evènement local à p_i : $HM_i[i,i]++$

  — Emission de m vers p_j : $HM_i[i,i]++; HM_i[i,j] := HM_i[i,j]+1$ et m est estampillé : $EM_m = HM_i$

  — Réception de (m, EM_m) en provenance de p_j.

(a) Donner les modifications de HM_i lors de la réception de (m, EV_m) en provenance de p_j.
Rappel : nous voulons la délivrance causale.

(b) Considérons un système contenant trois sites. Tous les sites possèdent des horloges logiques matricielles. Supposons que l'horloge matricielle HM_3 du site 3 en B est

$$HM_3 = \begin{pmatrix} HM_3[1,1] & HM_3[1,2] & HM_3[1,3] \\ HM_3[2,1] & HM_3[2,2] & HM_3[2,3] \\ HM_3[3,1] & HM_3[3,2] & HM_3[3,3] \end{pmatrix} = \begin{pmatrix} 6 & 2 & 2 \\ 1 & 5 & 1 \\ 1 & 2 & 7 \end{pmatrix}$$

i. Comment sait-on le nombre d'événements interne d'un site i ?

ii. A quoi correspond l'élément de l'horloge matricielle $HM_3[3,1]$, pour le site 3 ?

iii. Même question pour les éléments $HM_3[1,3], HM_3[2,3]$ pour le site 3 ?

iv. Que peut déduire le site 3 par rapport aux éléments $EM_m[1,3], EM_m[2,3]$?

v. Le site 3 peut-il délivrer le message m (délivrance causale) ? Justifier votre réponse.

vi. B reçoit

$$EM_m = \begin{pmatrix} 8 & 2 & 3 \\ 2 & 9 & 2 \\ 1 & 1 & 3 \end{pmatrix}$$

. Que doit-on faire ?

vii. Donner la valeur des horloges matricielles pour la figure 4.

viii. Avez-vous détecté une erreur dans la matrice en e, p ou z ?

Correction exercice 1

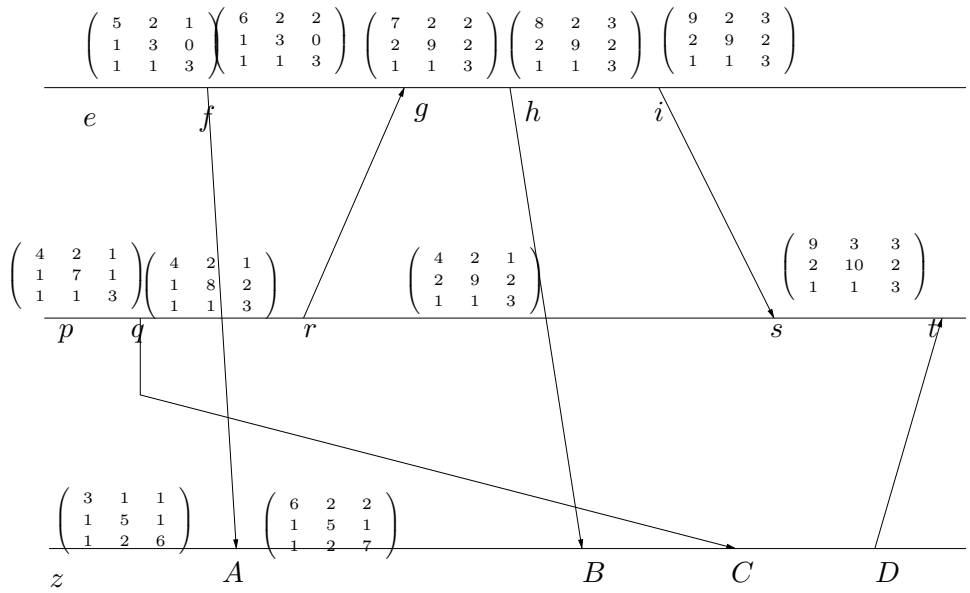


FIGURE 4 – *Début de datation avec l'horloge matricielle : attention à la délivrance des messages*

1. L'algorithme est le suivant :

   `En cas d'un événement interne`

   $\theta_p := \theta_p + 1$;

   `En cas d'un événement d'envoi de message`

   $\theta_p := \theta_p + 1$;

   `Envoyer(`$< message, \theta_P >$`);`

   `En cas d'un événement de reception de message (`$< message, \theta >$`)`

   $\theta_p := \max\{\theta_p, \theta)\} + 1$;

2. La solution est donnée par la figure 5. L'ordre des événements n'est pas un ordre strict : plusieurs événements peuvent porter la même valeur.

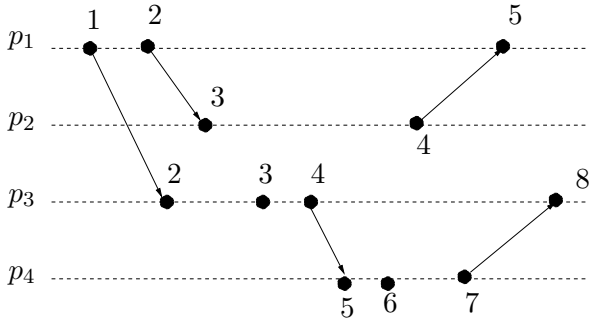


FIGURE 5 – *Solution*

3. **l'horloge logique** du site qui envoie le message. On a la relation d'ordre suivante $HL(e)$ de e sur le site i est un couple (h_i, i) :

$$(h_i, i) < (h_j, j) \Leftrightarrow (h_i < h_j \text{ ou } (h_i = h_j \text{ et } i < j))$$

4. Par récurrence :
   - si $n = 0$ alors $e = e'$ et $HL(e) \preceq HL(e')$
   - Si $n > 1$ soit $e = e_0, e_1, \ldots, e_n = e'$ une suite d'évènement tel que $\forall i \in [0, \ldots, n-1], e_i \to e_{i+1}$.
     Par récurrence on a $e \rightsquigarrow e_{n-1}$ et $HL(e) \prec HL(e_{n-1})$. Sachant que $e_{n-1} \to e_n$, on a
     - soit e_{n-1}, e_n appartiennent au même site $HL(e_{n-1}) \prec HL(e_n)$;
     - Soit il existe une message m tel que $send(m) = e_{n-1}$; par définition on a $HL(e_{n-1}) \prec HL(e_n)$
   - Par transitivité de $\preceq$ on a $HL(e) \prec HL(e_n)$

5. Sur l'ordre causal
   (a) a, d, f, c précède u ; b précède c ; $v \rightsquigarrow u$ car $b \rightsquigarrow u$ et $c \rightsquigarrow u$.
   (b) On a $PASSE(u) = \{a, b, c, f, v, h\}$; $FUTUR(u) = \{x, y\}$, $CONCURRENT(u) = \{g\}$
6. Sur l'approche vectorielle
   (a) La solution est donnée par la figure 6.

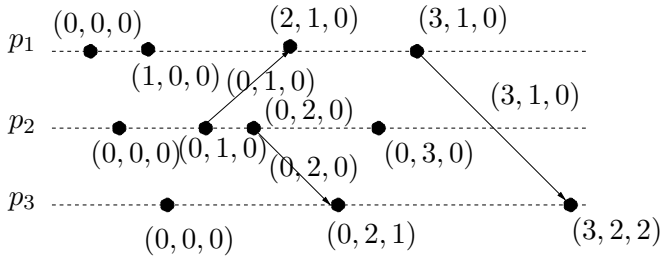


FIGURE 6 – *Solution pour les horloges vectorielles pour la figure 3*

   (b) Soit $e' \in PASSE(e)$ un évènement du site S_i : $EV_{e'}[i] \leq EV_e[i]$ où $PASSE(e)$ désigne un ensemble des événements antérieurs à e dans l'ordre causal (e appartient à cet ensemble).
   Considérons e' tel que $EV_{e'}[i] = EV_e[i]$.
   $card(PASSE(e') \cap \{e''$ appartenant à $S_i\} = EV_{e'}[i] - 1$.
   $Futur(e') \cap Passe(e) = \varnothing$ sinon $EV_{e'}[i] \neq EV_e[i]$ où $Futur(e)$: ensemble des événements postérieurs à e dans l'ordre causal (e appartient à cet ensemble).
   (c)

$$EV_e \preceq EV_{e'} \text{ ssi } \forall i, EV_e[i] \leq EV_{e'}[i]$$

(d) Si $e \rightsquigarrow e'$ alors

- $Passe(e) \subseteq Passe(e')$
- $\forall i, EV_e[i] \leq EV_{e'}[i]$
- $EV_e \preceq EV_{e'}$

(e) $e \,||\, e'$ alors : soit S_i et S_j les sites appartenant à e et e'

- si $j = i$ alors, e et e' ne peuvent pas être concurrents
- donc $j \neq i$
- Soit $x(e, j)$ le plus grand élément du $Passe_j(e)$
- on a $x(e, j) \rightarrow e'$ (car si $e' \rightsquigarrow x(e, j)$, alors $e' \in Passe(e)$)
- donc $Passe_j(e) = Passe_j(x(e, j)) \subset Passe_j(e')$
- Par conséquent : $EV_e[j] < EV_{e'}[j]$
- Utilisant le même raisonnement : on a $EV_{e'}[i] < EV_e[i]$

(f) Non il suffit de considérer le graphe donné par la figure 7

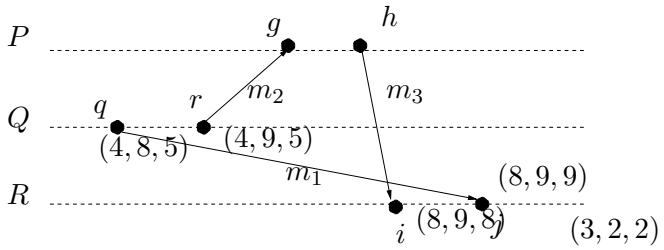


FIGURE 7 – *Contre-exemple*

7. Sur l'estampille matricielle

(a) Voici les modifications concernant la réception de (m, EM_m) en provenance de p_j.

- Quand un message peut-il être délivré ? Quand touts les messages antérieurs lui ont été délivrés i.e. :
  - Respect de l'ordre FIFO sur le canal $j \rightarrow i$; $EM_m[j, i] = HM_i[j, i]$
  - Respect de l'ordre de réception : $EM_m[k, i] = HM_i[k, i], \forall k \neq i$
- Lors de la délivrance du message (m, EM_m) en provenance de p_j mettre à jour l'horloge.
  - $HM_i[i, i] + +$
  - $HM_i[i, j] := HM_i[i, j] + 1$
  - $\forall k \neq i, \forall l \neq i; HM_i[k, l] := \max(HM_i[k, l], EM_m[k, l])$

(b) Horloges logiques matricielles

i. Il faut faire la différence entre $H[i,i] - \sum_{j \neq i} H[i,j] - \sum_{j \neq i} H[j,i]$. Le nombre de messages envoyés plus messages reçus c'est à dire la différence entre le nombre événements-total.

ii. $HM_3[3,1]$ correspond au nombre de messages issus du site 3 vers le site 1 (ici le site 3 a émis un seul message vers le site 1).

iii. Le site 3 sait que :

— $HM_3[1,3]$ correspond au nombre de messages issus du site 1 vers les site 3 (ici le site 1 a émis 2 messages vers le site 3)

— $HM_3[2,3]$ correspond au nombre de messages issus du site 2 vers les site 3 (ici le site 1 a émis 1 messages vers le site 3)

iv. Le site 3 peut en déduire en fonction de l'estampille du message m que

— le message m correspond au 3 messages envoyés par 1. Comme le site 3 a reçu 2 message venant de 1, il n'existe pas de message encore non reçu venant de 1 dans le passé de la réception de ce message.

— dans le passé de l'émission de m, il existe l'évènement suivant : le site 2 a émis son message 2 vers le site 3. Mais le site 3 ne l'a pas encore reçu

v. Non. Pour respecter la délivrance causale, le site 3 doit attendre le message m' venant de 2 dont l'élément $EM_{m'}[2,3]$ de l'horloge matricielle est égale à 2.

vi. D'après les valeurs en $H[2,3] = 2$ du message reçu, on doit attendre le message tel que $H[2,3] = 2$. C'est le cas avec l'arrivée de $\phi = \begin{pmatrix} 4 & 2 & 1 \\ 1 & 8 & 2 \\ 1 & 1 & 3 \end{pmatrix}$

vii. La solution est donnée par la figure 8

viii. Il y a effectivement une erreur en a_{11}

Fin correction exercice 1

2 Algorithmes pour l'exclusion mutuelle

Exercice 2 – Exclusion mutuelle dans les arbres

Nous utiliserons les messages $REQUEST$ pour demander à utiliser la ressource et le message $JETON$ qui représente la ressource (ou l'autorisation de l'utiliser). L'idée de ce protocole est que chaque site p a toujours un «pointeur» ($Racine_p$) qui est dirigé vers le jeton dans l'arbre.

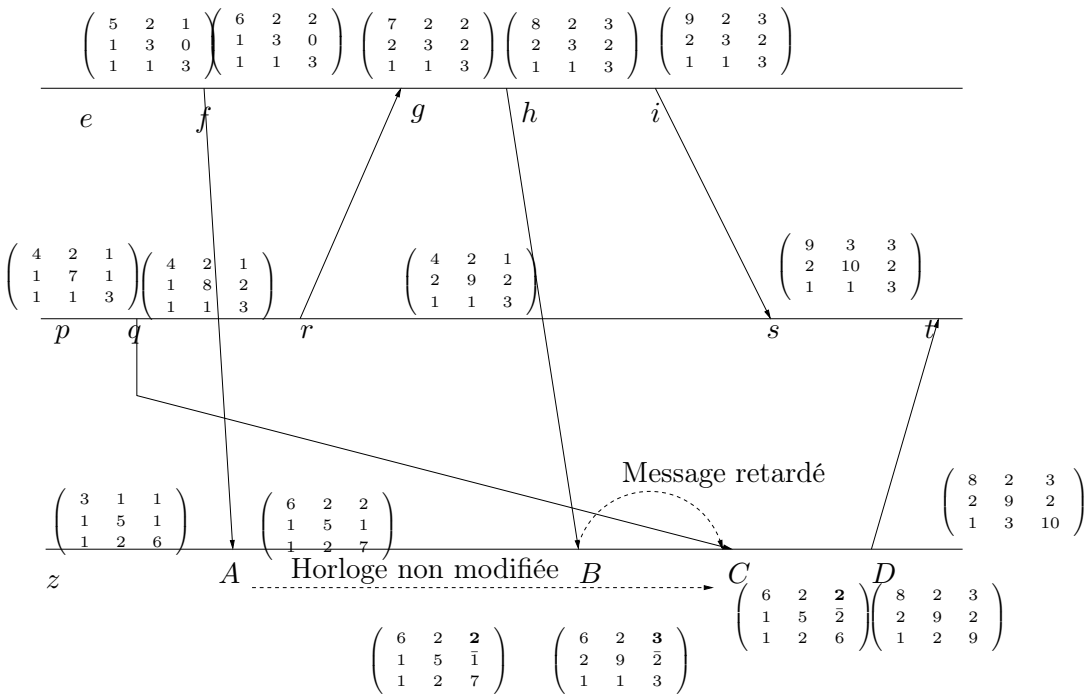


FIGURE 8 – *Solution*

`Procédure acquisition`

`Si (non` $Avoir\_jeton_p$`) alors`

`Si (File_vide(`$Request_p$`)) alors Envoyer(<` $REQUEST$ `>) à` $Racine_p$`;`

`Ajouter(`$Request_p$`,`p`);`

`Attendre(`$Avoir\_jeton_p$`);`

$En\_SC_p := vrai$`;`

`Procédure libération`

$En\_SC_p := faux$`;`

`Si (non File_vide(`$Request_p$`)) alors`

$Racine_p$ `:= Défiler(`$Request_p$`);`

`Envoyer(<` $JETON$ `>) à` $Racine_p$`;`

$Avoir\_jeton_p := faux$`;`

`Si (non File_vide(`$Request_p$`)) alors`

`Envoyer(<` $REQUEST$ `>) à` $Racine_p$`;`

`Lors de la réception de <` $REQUEST$ `> de` q

`Si (`$Avoir\_jeton_p$`) alors`

`Si (`En_SC_p`) alors Ajouter(`$Request_p$`,`q`);`

`Sinon`

$Racine_p := q$`;`

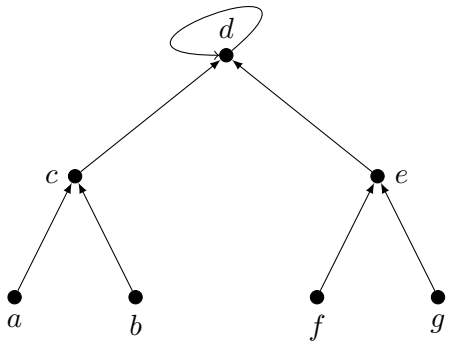


FIGURE 9 – Anti-arborescence.

```
            Envoyer(< JETON >) à Racine_p;
            Avoir_jeton_p := faux;
    Sinon
        Si (File_vide(Request_p)) alors Envoyer(< REQUEST >) à Racine_p;
        Ajouter(Request_p,q);

Lors de la réception de < JETON > de q
    Racine_p := Défiler(Request_p);
    Si (Racine_p = p) alors Avoir_jeton_p := vrai;
    Sinon
        Envoyer(< JETON >) à Racine_p;
        Si (non File_vide(Request_p)) alors Envoyer(< REQUEST >) à Racine_p;
```

La file *Request* sert à ordonner les demandes en chaque sous-arbre. Il est important de noter les points suivants dans l'algorithme.

Appliquer cet algorithme sur la figure 9, avec le scénario suivant :

— e fait une demande.

— g fait une demande

— b fait une demande et arrive avant la demande de g.

— a fait une demande qui arrive après celle de g.

Correction exercice 2

Exercice 3 – L'algorithme de Ricart-Agrawala

Dans l'algorithme suivant, chaque site désirant la ressource va demander la permission à tous les autres. On départage les conflits en étiquetant chaque demande par l'heure (logique) à laquelle on a fait cette demande. Les demandes les plus anciennes sont les plus prioritaires.

```
Procédure acquisition
    demandeur_i := vrai;
    horloge_i := horloge_i + 1;  heure_demande_i := horloge_i;
    rep_attendues_i := n − 1;
    Pour tout (x_i ∈ V − {i}) faire
        Envoyer(< REQUEST, heure_demande_i >) à x_i;
    Attendre(rep_attendues_i = 0);

Procédure libération
...
Procédure reception de request
...
Lors de la réception de < REPONSE > de j
    rep_attendues_i := rep_attendues_i − 1;
```

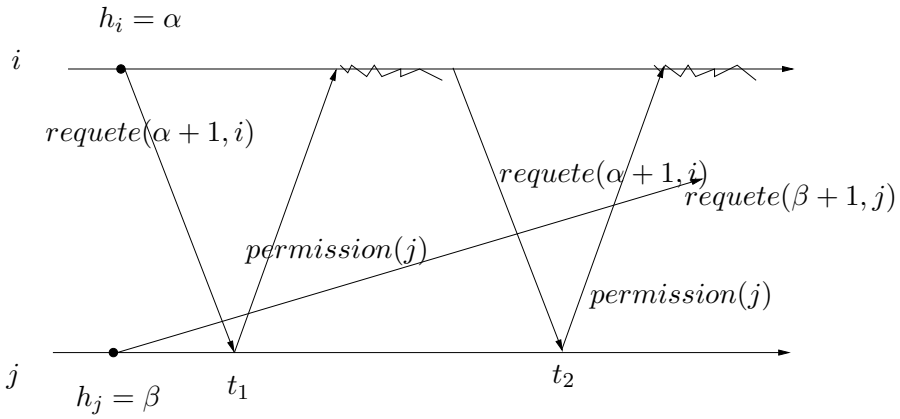


FIGURE 10 – *Effets du déséquencement*

1. Donner le rôle des variables $heure\_demande_i$, $rep\_attendues_i$, $demandeur_i$ et $differe_i$.
2. Donner la procédure de libération et de reception.
3. Donner la complexité en nombre de messages.

4. Montrer que l'algorithme de Ricart-Agrawala respecte le principe de sûreté c'est à dire que il existe au plus un site en section critique.
5. Montrer que l'algorithme de Ricart-Agrawala respecte le principe de vivacité c'est à dire que qu'un site pourra toujours rentré en section critique.
6. Donner l'intervalle de temps de non utilisation de la section critique sachant que les messages ont un temps de transit borné par T.
7. Avons-nous utilisé pour la démonstration que les canaux sont fifos ou non ? Une question se pose : Quelle incidence peut avoir le caractère non fifos des canaux sur le comportement de l'algorithme ? Pour répondre à cela nous allons nous limiter à 2 sites i et j qui obéissent au comportement suivant :

   ```
   Boucler sur Acquérir ;
       < utilisationdelasectioncritique >
       librer;
   Fin de boucle
   ```

   et nous supposons que $h_i = \alpha$ et $h_j = \beta$ avec $(\alpha, i) < (\beta, j)$. Une exécution possible est décrite par la figure 10 (la zone hachurée indique que le site correspondant est en section critique). Dans ce scénario les deux sites ont initialement fait des requêtes estampillées respectivement $(\alpha + 1, i)$ et $(\beta + 1, j)$ en t_1. A la réception de la requête, le site j renvoie sa permission qui double la requête qu'il avait envoyée précédemment. A la réception du message *permission*, le site i rentre en section critique, puis en sort et invoque à nouveau *acquerir*.
   (a) Quel est la valeur de l'horloge h_i à cet instant t_2 (que vaut x ? Quel conséquence ?
   (b) Que se passe-t-'il concernant l'horloge de i quand le site i reçoit la requête de j ?
   (c) Un site peut-il rentrer un nombre de fois arbitraire en section critique alors qu'un autre site en a fait la demande ? Ce nombre peut-il être non borné ?
   (d) Donner une exécution dans le cas où les canaux seraient fifos.
   (e) Comment obtenir, avec des canaux non fifos, un ordre de satisfaction des demandes analogues à celui que l'on obtiendrait avec des canaux fifos sans utiliser des messages supplémentaires ?
8. Quel est le défaut de cet algorithme concernant les demandes de permissions.

Correction exercice 3

1. Chaque site i maintient les variables locales suivantes. $heure\_demande_i$ représente l'heure à laquelle i a fait sa dernière demande. $rep\_attendues_i$ est un entier qui compte le nombre de réponses attendues par i suite à sa demande.

$demandeur_i$ booléen qui est vrai si et seulement si i est demandeur de la ressource ; initialisé à $faux$.
$differe_i$ tableau de n cases de booléens. $differe_i[j] = vrai$ si et seulement si i a différé sa réponse à la demande de j ; initialisé à $faux$.

2. `Procédure libération`
   $demandeur_i := faux$;
   `Pour tout` $(x_i \in V - \{i\})$ `faire`
   `Si` $(differe_i[x_i])$ `alors`
   `Envoyer(`$< REPONSE >$`)` `à` x_i;
   $differe_i[x_i] := faux$;
   et `Lors de la réception de` $< REQUEST, h >$ `de` j
   $horloge_i := \max\{horloge_i, h\}$;
   (2) `Si (non` $demandeur_i$ `ou` $(heure\_demande_i, i) > (h, j)$`) alors`
   `Envoyer(`$< REPONSE >$`)` `à` j;
   `Sinon` $differe_i[j] := vrai$;

   on fait le max pour resynchroniser les horloges, ceci garantit que toutes les requêtes futures du site i auront des dates supérieures à h. l'incrémentation pour acquérir et (2) garantit une progression correcte du temps logique.

3. La complexité est $2(n-1)$ messages par demande.
   NB : la variable $horloge_i$ est gérée implicitement ici grâce au sous-protocole présenté précédemment.

4. Il s'agit de montrer qu'il y a, à tout instant, au plus un site en section critique. Pour démontrer cela raisonnons par l'absurde en supposant qu'il y a deux sites i et j simultanément en section critique. Si tel est le cas d'une part i a envoyé à j un message $requte(h, i)$ et a reçu $permission(j)$ et d'autre part j a envoyé à i un message $requte(k, j)$ et a reçu $permission(i)$. Pour en arriver là trois cas sont a priori possibles.
   (a) le site i a envoyé sa requête à j et celui-ci l'a reçu avant de faire la sienne.
   Dans ce cas, à la reception du message de requête, le site j est dans l'état $dehors$: il renvoie sa permission à i et l'on a alors $h_j \geq h$. Lorsque le site j exécute $acqurir$ et envoie sa requête on a : $k = last_j = h_j + 1$ et donc $k > h$. A la reception du message $requte(k, j)$ le site i se trouvera donc prioritaire et ne renverra donc pas sa réponse. Cette situation n'a donc pas pu conduire à violer l'exclusion mutuelle.
   (b) le second cas a priori possible est obtenu en intervertissant les rôles de i et j.
   (c) Le troisième cas est celui où deux sites ont déjà envoyé leurs requêtes quand ils en reçoivent.Dans ce cas chacun des sites est, à la reception de la requête, dans l'état $demandeur$ et, comme l'on a soit $(h, i) < (k, j)$ soit le contraire, un seul des deux

booléens *priorit* peut prendre la valeur vrai ; Un seul des deux sites renverra donc sa permission.

Aucune n'est trois situations a priori possibles ne peut donc mettre en défaut la propriété de sûreté.

5. Les demandes de section critique sont totalement ordonnées par leurs estampilles ; considérons celle qui, à un instant donné, est dotée de la plus petite soit (h, i). Lorsque les messages de requêtes correspondant à cette demande arrivent sur les autres sites, ceux-ci ne peuvent être prioritaires et renvoient donc leurs permissions attendues et entrera alors en section critique. Comme une fois la section critique utilisée, la demande correspondante sort du système, que toutes les utilisations sont par hypothèse de durée finie et que les estampilles sont totalement ordonnées, toute demande finira par avoir la plus petite estampille et donc par être honorée.
6. En ce qui concerne le temps, le but est d'utiliser au maximum la section critique. On va donc compter le durée durant laquelle la section critique n'est pas utilisée bien qu'elle soit demandée. On suppose pour cela que les messages ont un temps de transit borné par T. Le pire des cas se produit lorsque un site désire utiliser la section critique et que celle-ci est libre. Ce site envoie donc des requêtes et reçoit les permissions par retour de courrier : soit $2T$. Si par contre le section critique est utilisée, les messages de requêtes transitent durant cette période ainsi qu'un certain nombre de permissions : au mieux un site reçoit toutes les permissions sauf celle de l'utilisateurs de la section critique ; lorsque celui-ci la libère il lui renvoie sa permission : dans ce cas la période de non utilisation est égale à T.
7. Sur le problème des canaux fifos et non fifos.

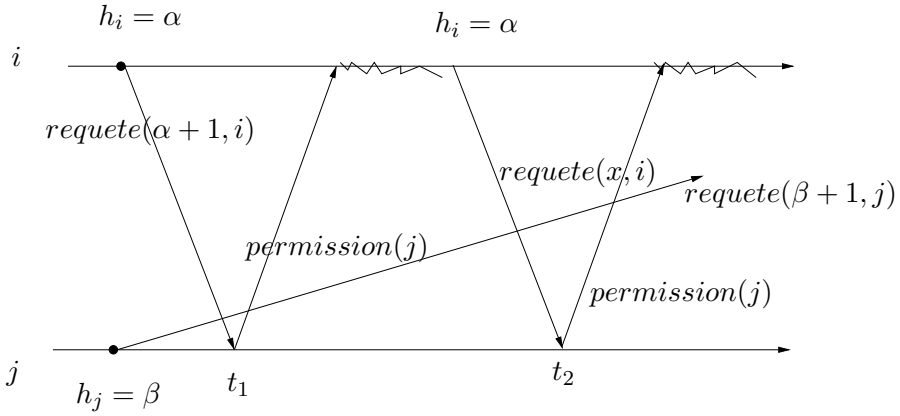


FIGURE 11 – *Effets du déséquencement*

(a) La valeur de l'horloge du site i au moment où il redemande à réentré en section critique vaut toujours α car aucune requête n'a été reçue par i. La seconde requête du site i transporte donc la même estampille que précédemment en t_2 le site j renverra sa permission, et ainsi de suite..

(b) Ceci peut se produire tant que la requête de j estampillé $(\beta+1, j)$ n'est pas parvenue au site i ; lorsqu'elle lui parviendra celui-ci mettra a jour $h_i = \max\{\alpha, \beta+1\}$ et dès lors les requêtes ultérieures de i auront des estampilles supérieures à $(\beta+1, j)$.

(c) En d'autres termes si les canaux ne sont pas fifos, il est possible qu'un site rentre en section critique un nombre arbitraire de fois alors qu'un autre site a effectué une demande. (Toutefois ce nombre est fini car les messages de requêtes finissent par arriver).

(d) Si dans l'exemple, les canaux étaient fifos les messages $permission(j)$ ne pourraient doubler le message $requete(\beta+1, j)$. Ce message serait le premier reçu par i qui, lors de cette reception, recalerait son horloge h_i à $\beta+1$; et la seconde requêtes du site i, la requête du site j serait satisfaite puisque :

$$(\alpha+1, i) < (\beta+1, j) < (\beta+2, i)$$

.

(e) Pour cela il faut, sur tout le canal, faire transporter par les messages qui peuvent doubler les requêtes, une date plus grande ou égale à celle de la dernière requête envoyée sur ce canal. Il faut estampiller les messages *permissions* envoyés de i vers j avec le couple $(\max(last_i, h_i), i)$, et recaler les horloges logiques à la reception de tout message *requete* ou *permission*.

Fin correction exercice 3

Exercice 4 – L'algorithme de Carvalho et Roucairol

Nous allons voir maintenant l'algorithme de Carvalho et Roucairol qui peut-être vu comme une amélioration du précédent. Considérons pour illustrer cela la situation suivante :deux sites, i et j tels que i veut utiliser la section critique plusieurs fois de suite alors que j n'est pas intéressé par celui-ci et reste donc dans l'état *dehors*. Avec l'algorithme précédent le site i demande la permission de j à chaque nouvelle invocation de l'opération acquérir. Le principe utilisé ici est le suivant : puisque j a donné sa permission à i, ce dernier la considère comme acquise jusqu'à ce que j demande à i sa permission ; dans la situation décrite le site i ne demande donc qu'une fois la permission à j.

1. Donner l'intervalle du nombre de messages nécessaires pour une utilisation de la section critique.
2. Donner une façon d'implémenter ce principe.
3. Montrer que la sûreté est garantie.
4. Montrer que la vivacité est garantie.
5. Donner l'algorithme.
6. Donner l'intervalle de temps de non utilisation de la section critique sachant que les messages ont un temps de transit borné par T.

7. Est-il possible de borner le domaine des valeurs d'horloges ?
8. Est-ce que le délai borné apporte quelque chose ?

Correction exercice 4

1. L'application de ce principe permet de réduire le nombre de messages nécessaires pour une utilisation de la section critique. Ce nombre est pair (à toutes requête correspond un renvoi de permission) et est fonction de l'état du système ; il varie entre 0 et $2(n-1)$.
2. Une façon d'implémenter l'idée précédente consiste à associer à tout couple de sites distincts i et j un et un seul message $permission(i,j)$ (dans ce qui suit $permission(i,j)$ et $permission(j,i)$ sont considérés comme deux synonymes du même message). Si ce message est chez i, celui-ci a la permission de j et inversement. Initialement $permission(i,j)$ est placé indifféremment sur l'un ou l'autre site. Lorsqu'un site i invoque l'opération *acqurir* il doit demander les permissions qui lui manquent ; les sites qui détiennent ces permissions constituent donc l'ensemble R_i auquel i adresse ses requêtes :

   $$R_i = \{j \text{ tel que le site } i \text{ ne possède pas } permission(i,j)\}$$

   . Il faut donc maintenant gérer les ensemble R_i qui évoluent dans le temps. Lorsque le site i reçoit $permission(i,k)$ il supprime k de R_i ; lorsqu'il envoie $permission(i,m)$ il rajoute m à R_i. Si l'on considère deux sites quelconques i et j on doit donc avoir initialement (puisque le message *permission* qui gère leurs conflits est chez un seul d'entre-eux).

   $$i \in R_j \text{ ou exclusif } j \in R_i$$

   . De plus les mises à jour ultérieures de R_i et R_j sont telles que cette relation est toujours vraie lorsque i et j invoque l'opération *acqurir*.
3. L'association à tout couple de sites d'une permission unique pour gérer leurs conflits garantit le sûreté.
4. L'association d'une estampille à toute requête va, comme pour l'algorithme précédent, garantir la vivacité. Lorsqu'un site i reçoit une requête estampillé (k,j) :
   - il renvoie $permission(i,j)$ s'il est dans l'état *dehors*
   - il mémorise le renvoi de $permission(i,j)$ s'il est dans l'état *dedans* ou s'il est prioritaire
   - enfin s'il est dans l'état *demandeur* et non prioritaire il renvoie $permission(i,j)$ au site j, suivi d'un message de requête pour indiquer au site j de lui retourner la permission après avoir utilisé la section critique.

   Il est facile de montrer que la demande dotée de la plus petite estampille ne peut être indéfiniment bloquée.

5. l'algorithme :

```
Lors d'un appel à acquérir
    etat_i = demandeur;
    last_i = h_i + 1;
    ∀j ∈ R_i : envoyer requête(last_i, i) à j;
    attendre (R_i = ∅);
    etat_i = dedans;

Lors d'un appel à liberer :
    etat_i = dehors;
    ∀j ∈ differe_i : envoyer permission(i, j) à j;
    R_i = differe_i;
    differe_i = ∅;

Lors de la reception de requete(k, j)
    h_i = max(h_i, k);
    priorité_i = (etat_i = dedans) ou ((etat_i = demandeur) et (last_i, i) < (k, j));
    Si priorité alors diffr_i = diffr_i ⋃ {j};
    Sinon;
        envoyer permission(i, j) à j;
        R_i = R_i ⋃ {j};
        Si etat_i = demandeur;
        Alors envoyer requete(last_i, i) à j;
        fsi
    fsi

Lors de la reception de permission(i, j);
    R_i = R_i − {j};
```

6. $[0, 2T]$
7. A la différence de précédemment, il n'est pas possible de borner le domaine des valeurs d'horloges. En effet un site qui a donné toutes ses permissions et qui, après une longue période, émet une requête n'a pas incrémenté son horloge logique depuis la dernière requête reçue ; son horloge peut donc avoir une différence quelconque avec une autre horloge. Ceci ne pouvait se produire avec le protocole précédent car la réception systématique de toute requête par tout site avait pour effet de bord de synchroniser les horloges locales modulo $n - 1$.
8. A priori oui

Fin correction exercice 4

Exercice 5 – **Algorithme de Maekawa et ses variantes** Dans l'algorithme de Ricart et Agrawala un site désirant la ressource doit demander la permission à tous les autres sites ce qui entraîne la génération d'un grand nombre de messages. Au lieu de demander un si grand nombre de permissions on peut imaginer que chaque site i a un ensemble de sites S_i à qui il va demander la permission d'entrer en section critique. Lorsque i obtient *toutes* les permissions de S_i il peut utiliser la ressource. Un tel ensemble S_i sera appelé *quorum*.

Nous souhaitons donc obtenir un algorithme réparti « symétrique » dans lequel chaque site joue un rôle équivalent au x autres au sens suivant :

$$
\begin{array}{lll}
(c_0) & i \in R_i & \\
(c_1) & \forall i : & Card(R_i) = K \\
(c_2) & \forall i & i \text{ est contenu dans } D \text{ ensemble } R_j
\end{array}
$$

1. Donner la signification des contraintes c_i.
2. Est-ce que l'algorithme de Ricart-Agrawala respecte les deux contraintes c_1 et c_2. Si oui, avec quelles valeurs de K et de D. Si nous utilisions l'algorithme de Ricart-Agrawala avec les valeurs données ci-avant, quel est le nombre de messages nécessaires pour accéder à la section critique ? Conclusion ?
3. Donner la règle d'intersection, pour qu'on ait la propriété de sureté.
4. Construire des quorums pour le cas où $K = 2, 3$.
5. On constate cependant qu'il peut y avoir des blocages si on ne met en place que ce mécanisme. Il faut rajouter un ordre de priorité pour s'assurer qu'en cas de multiples demandes simultanées au moins un site puisse recevoir toutes les permissions. Que va t'on utiliser ?
6. Toutes ces propriétés sont indispensables pour assurer la sûreté et la vivacité. Cependant, d'autres critères peuvent être pris en compte. Parmi ceux ci on peut distinguer la *charge induite par le mécanisme pour chaque site* ainsi que la performance en nombre de messages. Le premier point est relatif à la charge de travail que chaque site du système doit accomplir pour la communauté, même s'il n'a pas besoin de la ressource. Le deuxième point est la complexité de l'algorithme. Nous voudrions que chaque site i fasse partie de D quorums S et que pour tout j, $|S_j| = k$. Dans ce cas, la complexité de l'algorithme est en $O(k)$. Il faudrait donc minimiser k et D. Donner le nombre maximum de quorum qui peuvent être construits.
7. Construire effectivement des quorums avec ces propriétés est difficile (plans projectifs). Par contre, si $n = p^2$ il est facile de construire n quorums avec $\sqrt{n} = p$ sites dans chacun d'eux.
8. Dans ce qui suit nous décrivons les grandes lignes de l'algorithme d'allocation de ressources. Avant de mettre en place l'algorithme il faut au préalable construire les quorums et les affecter aux sites. Chaque site i connaît son quorum S_i. Chaque fois que i demande la ressource, il demande la permission, estampillée par son horloge, à tous les sites de S_i. Lorsqu'un site i reçoit une demande de permission de la part de j on peut être dans un des trois cas.

— i ne demande pas la ressource. Il envoie sa permission à j.

— i a demandé la ressource. Si sa demande a une estampille plus ancienne que celle de j il lui envoie un message pour lui dire non, sinon il lui donne la permission.

— i a déjà donné sa permission à un site k dont l'heure de la demande avait une estampille plus récente que celle de j. Dans ce cas, i va essayer de récupérer la permission qu'il avait donné à k pour la donner à j. Si k a reçu un message de refus de permission, il redonne la permission à i qui va la donner à j.

Expliquer pourquoi il n'y a pas de blocage et que la vivacité est assurée.

9. Donner l'algorithme.
10. Donner un scénario pour lequel il existe un interblocage avec $R_1 = \{1, 2\}$; $R_2 = \{2, 3\}$ et $R_3 = \{1, 3\}$, et chaque site gère une seule permission (pour entrer en SC il faut deux permissions).
11. Comment résoudre le problème de l'interblocage ?
12. Dans quel cas il y a déséquencement ?,
13. Donner le délai pour lequel la section critique bien que demandée n'est pas utilisé ?
14. Quel est le nombre de messages relatifs à une utilisation de la section critique ?

Correction exercice 5

1. La contrainte c_0 indique qu'il y a au moins n R_i ($\#R_i \geq n$. La contrainte c_1 exprime que tous les sites doivent demander et obtenir le même nombre de permissions pour pouvoir entrer en section critique (règle du même effort) ; la contrainte c_2 indique que chaque site joue un rôle d'arbitre pour le même nombre de sites (règle de la même responsabilité).
   On veut minimiser $\#R_i$ et $Card(R_i)$.
2. Les r_i utilisé dans l'algorithme de Ricart-Agrawala, respectent les deux contraintes avec $K = D = n - 1$. Le nombres de message nécessaires pour obtenir la section critique est de $3(n-1)$, c'est trop gourmand en nombre de messages. Donc cette classe d'algorithme est intéressante quand K et D sont raisonnables petits.
3. Pour être sûr que la propriété de sûreté est vérifiée il faut que les quorums vérifient la propriété suivante.

   Règle d'intersection : si $i \neq j$ alors $S_i \cap S_j \neq \varnothing$

   Cette propriété est indispensable pour s'assurer que lorsque deux sites i et j demandent les permissions pour entrer en section critique, les sites qui sont à la fois dans S_i et S_j ne peuvent pas accorder la permission aux deux. Ainsi, en cas de demandes simultanées au moins un des sites n'aura pas une permission.

4. Pour $K = 2$, on a $S_1 = \{1, 2\}$, $S_2 = \{2, 3\}$ et $S_3 = \{1, 3\}$; Pour $K = 3$, on a $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 4, 6\}$; $S_3 = \{3, 5, 6\}$; $S_4 = \{1, 4, 5\}$, $S_5 = \{2, 5, 7\}$; $S_6 = \{1, 6, 7\}$ et $S_7 = \{3, 4, 7\}$
5. Cet ordre sur les demandes sera celui des horloges logiques.
6. Avant tout i l peut montrer que $D = K$ de la manière suivante : par la contraintes c_1, en faisant la somme $\sum_i Card(R_i) = nK$. De plus, nous avons par la contraintes c_2 pour tous les sites nD. Ces deux quantités ne peuvent être égale.

   Un quorum contient K membres qui chacun font partie d'au plus $D - 1$ autres ensembles. Ainsi, le nombre maximum de quorums qui peuvent être construits est $n = K(D - 1) + 1$. De plus, $\frac{Dn}{K}$ est le nombre maximum de quorums car chaque site peut être dans au plus D quorums contenant chacun au plus k sites. S'il y a autant de sites que de quorums, on a $n = \frac{Dn}{k}$, c'est à dire $D = k$. En combinant ces égalités on obtient $n = k(k - 1) + 1$ et $k = O(\sqrt{n})$.

   On peut prendre $\#R_i = n$ car on cherche le minimum. Soit $j \in R_i$, j appartient à $D-1$ autres quorums. Le maximum de quorums disjoints 2 à 2 ($\forall i, jR_j \cap R_j \neq \varnothing$ est donc $1 + K(D - 1)$, on a donc $n = K(D - 1) + 1$

   De plus, on couvre tous les éléments $1, \ldots, n$ par les R_i car on a nK éléments et en divisant par D on a forcément n.
7. La figure 12 illustre le mécanisme pour le site x : les sites sont dessinés suivant une grille et, le quorum d'un site x est l'ensemble de tous les sites sur sa ligne et sur sa colonne dans la grille. Avec cette construction, toutes les bonnes propriétés sont vérifiées.
8. Comme l'ordre basé sur les estampilles est un ordre total il y aura toujours un site qui pourra utiliser la ressource. Ceci garantit qu'il n'y a pas de blocage et que la vivacité est assurée. La sûreté provient du fait qu'un site ne donne sa permission qu'à au plus un site à chaque instant.
9. a faire
10. Examinons le scénario suivant : trois sites 1, 2 et 3 ont pour ensembles R_i respectivement $R_1 = \{1, 2\}$, $R_2 = \{2, 3\}$ et $R_3 = \{1, 3\}$, s'ils demandent simultanément la section critique l'interblocage est possible si le site 2 (resp. (3 et 1) donne l'unique permission qui gère à 1 (resp. $2, 3$).

    Ainsi chaque site possède une seule permission.
11. La résolution de l'interblocage est basée sur un mécanisme de préemption (mise en oeuvre par les messages *rendreperm*) ou la prise ne compte des estampilles.
12. Pour résoudre ce problème il faut mettre en place un système de rendu de permissions.
13. Il y a déséquencement quand sur un site j_1 le message *rendreperm* arrive avant la permission.
14. Le délai durant lequel la section critique, bien que demandée n'est pas utilisée, est toujours égal à $2T$.

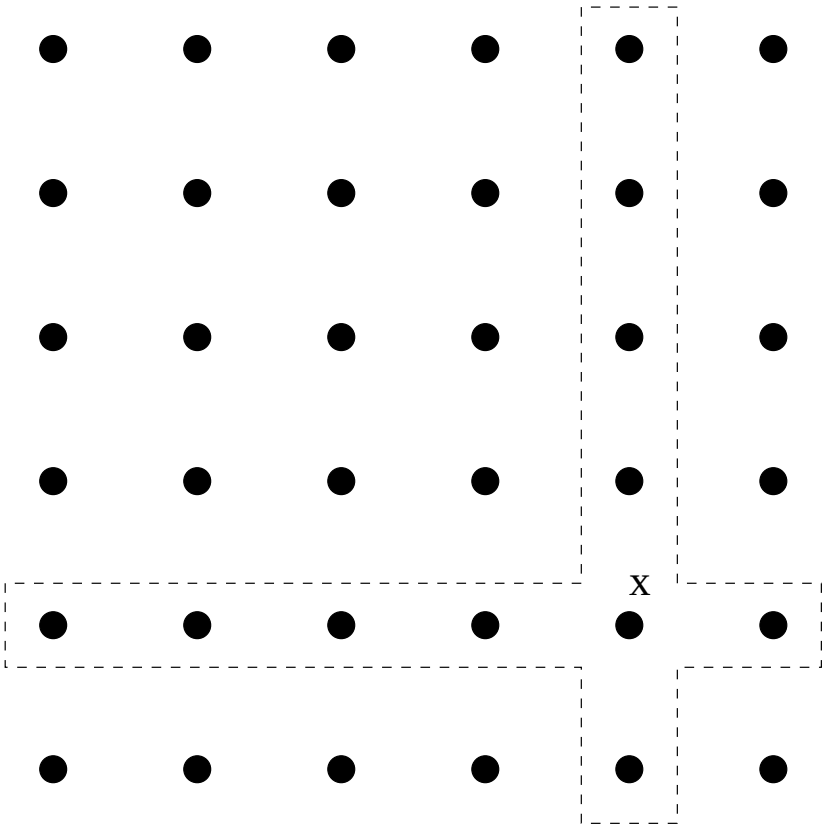


FIGURE 12 – *Illustration de la construction des quorums pour* $n = 36$

15. En conclusion, le nombre de messages relatifs à une utilisation de la section critique est compris entre $3 \times Card(R_i)$ (un message de requête + une permissions + une restitution de la permission) et $5 \times Card(R_i)$ (les 3 messages précédents plus les 2 messages éventuels nécessaires pour établir la priorité qui évitera les interblocages).

Fin correction exercice 5

Exercice 6 – L'algorithme de Naimi et Tréhel

Dans cet exercice nous nous plaçons dans le cas d'une ressource à une seule entrée (au plus un sommet peut utiliser la ressource à un instant donné). Réaliser l'exclusion mutuelle en réparti revient à proposer une mise en oeuvre distribué de l'objet informatique qu'est une file des sites. plusieurs solutions sont envisageables allant de la duplication de l'objet sur chaque site (avec un protocole qui assure la cohérence mutuelle des copies) à son partitionnement ; L'algorithme que nous présentons maintenant appartient à cette dernière catégorie. Une file va donc être implémentée en plaçant sur chaque site i un pointeur $suivant_i$, qui, s'il est différent de nil, indique le site successeur de i dans la file d'attente. La file étant définie deux problèmes se posent :

1. Comment un site sait-il qu'il est en tête de la file ?
2. Comment un site qui n'est pas dans la file peut-il venir se placer en queue de file ?

C'est la réponse à ces deux questions qui va définir l'algorithme.

1. Proposez un système qui permet de savoir si un site est en tête de file ?
2. Proposez une structure qui permet de se reconfigurer dynamiquement et qui permet de savoir qui se trouve en dernier (dernier élément de la file).
3. Proposez un protocole de gestion de la structure
4. Nous avons raisonné jusqu'à maintenant comme si la file n'est pas vide. Proposez une solution simple pour éviter que la file soit vide.
5. Donner un exemple d'utilisation de l'algorithme où s est la racine de l'arbre possèdant le jeton et p et q désirent entrer dans la file. p est traité avant q et p reçoit le jeton de s. Puis s veut rentrer en section critique.
6. Ecrire la phase de libération.
7. Donner un exemple d'exécution de cet algorithme. A qui correspondent les variables $pere_p$ et $suivant_p$?
8. Pourquoi un sommet demandant le jeton l'obtiendra t'il au bout d'un temps fini (s'il n'y a pas de panne) ?
9. Quel est le nombre de messages échangés dans le pire des cas lors d'une demande ?
10. Est-ce que pour un site i son père diffère durant l'exécution ? Qu'est-ce que cela implique pour tous les sites ?

Voici le protocole que nous proposons. Le code est décrit pour le sommet p.

```
Procédure Demander-SC;
    demandeur_p := vrai;
    Si (pere_i ≠ nil) alors
        Envoyer(< REQUEST >) à pere_p;
        pere_p := nil;
        Attendre(AvoirJeton_p);

  <SECTION CRITIQUE>

(* Phase de libération *)

Lors de la réception de < REQUEST > de q (q demandeur);
    (Demandeur_p) alors
        (Suivant_p = nil) alors suivant_p := q
```

```
Si (pere_p ≠ nil) alors
    Envoyer(< REQUEST > à pere_p
Sinon Si (AvoirJeton_p) alors
    AvoirJeton_p := faux
    Envoyer(< TOKEN >) à q
pere_p := q

Procédure recevoir < TOKEN > de q;
    AvoirJeton_p := vrai;
```

Initialisation : choisir un sommet S quelconque et

$AvoirJeton_S := vrai$; Pour tout $X \neq S$, $AvoirJeton_X := faux$;
Pour tout sommet Y, $pere_Y := S$; $suivant_Y := NIL$; $Demandeur_Y := faux$;

Correction exercice 6

1. La tête de file est unique et seul le site qui est en tête de file a besoin de la savoir ; il peut indiquer à son successeur que ce dernier devient la nouvelle tête de file lorsque lui-même en sort. Les propriétés que nous venons de décrire peuvent trivialement être mises en oeuvre par un jeton : être en section critique c'est l'avoir, et sortir de la file revient à passer le jeton à son successeur.
2. Le dernier élément de la file a, tout comme la tête, une propriété d'unicité, mais contrairement à celle-ci, le fait de ne plus être le dernier de la file ne dépend pas du dernier : tout site peut faire une requête et devenir alors le dernier. Il est donc nécessaire que tout site puisse adresser le dernier site de la file pour lui indiquer qu'il n'est plus le dernier et qu'il a désormais un successeur. En d'autres termes il faut fournir aux sites une structure d'adressage qui d'une part leur permettre de savoir qui est le dernier et qui d'autre part puisse se reconfigurer dynamiquement lorsqu'un site rentre dans la file devenant de ce fait le "nouveau dernier". Une arborescente couvrante de racine ce dernier est une telle structure (le dernier de la file est le sommet racine de l'arborescence) :
   - elle présente une propriété d'unicité (la racine est unique : il s'agit du dernier élément de la file).
   - elle offre un adressage qui permet à tout site d'adresser (indirectement) la racine.
   - elle peut facilement se reconfigurer en inversant le sens des arcs ou en détruisant certains arcs et en rajoutant d'autres. (cette technique est connue sous le nom d'inversion d'arcs "edges or path reversal").
3. Le protocole de gestion de cette structure est alors le suivant. Lorsqu'un site i veut entrer dans

la file il envoie un message $requete(i)$ à son père dans l'arborescence puis ce déclare racine, il n'a plus alors de père. Lorsqu'un site k reçoit un message $requete(i)$ il le fait progresser vers son père (pour qu'il atteigne la racine) puis déclare que son nouveau père est le site i : à la connaissance de k c'est le site i qui est maintenant le dernier de la file. Lorsqu'un site m qui est racine reçoit un message $requete(i)$ il apprend qu'il n'est plus le dernier et, outre la mise à jour $suivant_m = i$, il déclare que son père est le site i. L'arborescence couvrante est ainsi reconfigurée. Il est important de noter de remarquer que seule la branche comprise entre la nouvelle et l'ancienne racine est modifiée par ce protocole de reconfiguration. De plus, tout site doit appartenir à l'arborescence. En effet, lorsqu'un site sort de la file l'arborescence n'est pas modifiée ; or s'il veut rentrer à nouveau dans la file il doit être dans l'arborescence afin d'en connaître le dernier élément. Le jeton qui gère la tête de la file et l'arborescence qui en gère le dernier élément sont ainsi deux "structures de contrôles" utilisées de façon orthogonale (au sens où elles ne sont pas réductibles l'une à l'autre).

4. Le cas de la file vide : nous avons raisonné jusqu'à présent comme si la file n'était jamais vide. Qu'en est -il lorsque cela se produit ? Pour cela examinons le cas où il n'y a qu'un seul site dans la file, soit i ce site. Il est à la fois le premier et le dernier élément de la file et donc possède le jeton ($jetonpresent_i$ est à $vrai$) et est la racine de l'arborescence. Lorsqu'il sort de la file il ne peut transmettre le jeton puisque la file est alors vide : il reste donc possesseur du jeton et racine. Afin de distinguer les deux possibilités précédentes il suffit d'introduire un booléen local à chaque site $danslafiel_i$ qui a la valeur $vrai$ si, et seulement si, le site i est dans la file. On a alors

$$\exists i(\text{ non } danslafile_i \text{ et } (i \; est \; racine ou \; jetonpresent_i))$$

est équivalent à $lafileestvide$.

5. L'exécution est donné par la figure 13.

6. `Si (`$Suivant_p \neq NIL$`) alors`
   $AvoirJeton_p := faux$;
   `Envoyer(`$< TOKEN >$`) à` $Suivant_p$;
   $Suivant_p := NIL$;
   $Demandeur_p := faux$

7. On prend un exemple à 5 sites. Voir le déroulement à la figure**??**. Les arcs pleins pointent vers le prochain site à qui on va demander le jeton la prochaine fois (même si ce n'est pas lui qui l'a). La racine de l'anti arborescence est le sommet qui a ou aura le jeton (la variable $NouvelleRacine$ représente l'adresse de cette racine). Les arcs en pointillés représentent une **file d'attente répartie**. Lorsqu'un site demande le jeton il va être placé à la fin de cette file. Lorsque le site qui a le jeton le libère, il le passe à son successeur dans la file (variable $Suivant$).

8. Avec toutes les hypothèses, le site possédant le jeton va finir par le faire passer au suivant dans la file. Or, il est toujours possible de s'insérer dans la file. Ainsi, l'attente du jeton est finie.

9. Nombre de messages dans le pire des cas : le message $< REQUEST >$ peut être envoyé jusqu'à au plus $n-1$ fois et le jeton est envoyé une fois. Ainsi, au plus n messages échangés lors d'une demande de jeton. On montre (dur) qu'en moyenne il y a $O(\log n)$ messages échangés.
10. L'algorithme étudié est tributaire de la connaissance des identités des sites : lorsqu'un site i fait suivre $REQUEST(j)$, l'identité j a une signification dont la portée est globale.
11. Le nombre de messages nécessaires à une utilisation de la section critique est compris entre 0 et n.
12. La valeur n est obtenue lorsque l'arborescence étant dégénérée, le site i (la seule feuille) requiert la section critique alors que le jeton est possédé par le site m (le père), $n-1$ messages de requête vont être utilisés plus un message pour le jeton.
13. Remarquons simplement que l'arborescence qui fait suite à une arborescence dégénérée est optimale c'est à dire $n-1$ feuilles.
14. Le délai durant lequel la section critique, bien que demandée est inutilisé varie entre 0 et nT.

Fin correction exercice 6

3 Algorithmes pour le problème de l'élection

Exercice 7 – Anneau, Algorithme de Chang-Roberts

Nous proposons maintenant un autre algorithme pour l'élection dans un cycle unidirectionnel. Chaque site i maintient un booléen $participant_i$ qui est initialisé à $Faux$. Le code pour le site i est le suivant.

```
En cas de réveil spontané
    participant_i := Vrai;
    Envoyer(< ELECTION, val_i >);

Lors de la réception d'un message < ELECTION, V >
    Si (V > val_i) alors
        participant_i := Vrai;
        Envoyer(< ELECTION, V >);
    Si (V < val_i) et (non participant_i) alors
        participant_i := Vrai;
        Envoyer(< ELECTION, val_i >);
```

```
    Si (V = val_i) alors Envoyer(< ELU, val_i >);

Lors de la réception d'un message < ELU, V >
    Le gagnant est le site V;
    participant_i := Faux;
    si (val_i ≠ V) alors Envoyer(< ELU, V >);
```

1. Proposez une exécution dans la configuration de la figure 14. Qui détecte le gagnant et comment est-il propagé ?
2. Est-ce que cet algorithme fonctionne en mode asynchrone ?
3. Supposons que tous les sites se réveillent simultanément. Evaluez la complexité en nombre de messages échangés dans les configurations particulières suivantes en mode asynchrone et synchrone :
   - Les valeurs sont ordonnées croissantes dans le sens du cycle.
   - Les valeurs sont ordonnées décroissantes dans le sens du cycle.
4. Nous allons étudier la complexité en moyenne.
   (a) Montrer qu'un site actif possédant la $j$ième plus grande identité parcours en moyenne n/j liens.
   (b) Avec x initiateurs, donner la formule.
   (c) Conclure que la complexité en $O(n \log n)$.

Correction exercice 7

1. Ici le réveil est fait soit spontanément soit par la réception d'un message du protocole. Pendant l'algorithme, un site recevant une valeur V ne va la faire suivre que si elle est supérieure à la sienne propre. Il y a donc un mécanisme de **filtrage** des valeurs. Ce qui réduit la complexité en nombre de messages par rapport à l'exercice **??**. Qui détecte le gagnant ? C'est le gagnant lui même ! ! S'il reçoit sa propre valeur c'est que celle ci n'a jamais été arrêtée (filtrée) ; c'est donc la plus grande. Il informe donc les autres grâce au deuxième message.
2. Oui icic cet algorithme fonctionne en mode asynchrone.
3. **En mode synchrone :** Dans le premier cas, les messages seront stoppés très rapidement ; $n-1$ valeurs seront stoppées après une seule communication (car elles vont sur un site qui a une valeur plus grande). Seule la plus grosse valeur fait tout le tour du cycle. Ainsi, sans compter la proclamation du résultat on a $n-1+n = 2n-1$ messages échangés.

Dans le second cas, le message contenant la valeur i ème valeur sera renvoyée i fois. Ainsi le nombre total de messages échangés sera de $\sum_{i=1}^{i=n} i = \frac{n(n+1)}{2}$. **En mode asynchrone** : $2n$ si le plus grand est le seul à se réveiller (cas croissant) et $3n-1$ dans le cas décroissant si le site $n-1$ se réveille. Notons que nous comptons la proclamation de l'élection.

4. Etudions maintenant la complexité en moyenne

   (a) Ce résultat est obtenu simplement et en toute généralité en considérant $0 < a \leq n$ processus initiateurs déclenchant indépendamment mais pas (nécessairement) simultanément l'exécution de l'algorithme. Puisque les permutations des identités des processus sont ordonnées de manière équiprobable sur l'anneau, le jeton envoyé par un processus actif possédant la $j$ième plus grande identité parcourt en moyenne n/j liens sur l'anneau avant d'être éteint par un processus d'identité supérieure.

   (b) Par conséquent, avec a initiateurs indépendants, l'élection est réalisé en utilisant un nombre moyen de jetons exactement égale à $\sum_{j=1}^{a} n/j = nH_a$,

   (c) et si tous les processus sont initiateurs ($a = n$), on retrouve le résultat précédent : $O(nH_n) = O(n \log n)$.

Fin correction exercice 7

Exercice 8 – **Anneau bi-directionnel, Algorithme de Hirschberg et Sinclair (80)**

Nous allons maintenant travailler sur des cycles bi-directionnels (on peut envoyer des messages dans les deux sens sur le cycle). L'algorithme proposé marche par *phase* et est basé sur le filtrage. Les sites peuvent être dans deux états $actif$ ou *perdant*. Lorsqu'un site reçoit un message contenant une valeur plus grande que la sienne il devient *perdant*. Sinon il reste $actif$.

A la phase i, ($i = 0, 1, \dots$) les sites encore actifs envoient leur valeur à une distance de 2^i d'eux. Lorsqu'un site $actif$ reçoit un tel message il devient *perdant* si la valeur reçue est supérieure à la sienne sinon il passe à la phase suivante.

Nous allons supposer ici que le réseau est *synchrone par phase*. C'est-à-dire que les sites encore actifs au début de la phase i commencent tous *en même temps* à exécuter les opérations de cette phase.

1. Donner un exemple d'exécution sur la graphe de la figure 14
2. Après la phase i, quelle est la distance minimale entre deux sites $actifs$?
3. Combien y a t'il de phases ? En déduire le nombre de messages échangés pendant l'algorithme.
4. Qui connaît le gagnant ?
5. Donner l'algorithme.
6. Quel est l'inconvénient sur le nombre de messages ?

Correction exercice 8

1. Au début de la phase i, la distance entre deux sites actifs est au moins de $2^{i-1}+1$. En effet, un site reste actif après la phase $i-1$ s'il a pu éliminer tous les sites à distance 2^{i-1}. On remarquera qu'au début de la phase i il existe au plus un site $actif$ à distance 2^i à droite (et à gauche) d'un site actif (celui ci est à distance au moins $2^{i-1}+1$ et au plus 2^i). Ainsi, au début de la phase i, il reste au maximum $\frac{n}{2^{i-1}+1}$ sites actifs. Pendant la phase i il y a donc au maximum $2.2^i.\frac{n}{2^{i-1}+1}$ messages qui transitent sur le réseau.
2. Le nombre de phases est de $\log_2 n$ (au delà on "boucle"). Ainsi il y a moins de $\sum_{i=1}^{\log_2 n} 2.2^i.\frac{n}{2^{i-1}+1} \leq 4n\log_2 n$ échanges de messages pendant l'algorithme et ceci quelque soit la répartition des valeurs sur le cycle.
3. Le gagnant est celui qui reçoit sa propre valeur à la phase $\log_2 n$ et qui ne reçoit pas d'autres valeurs plus grandes. Il peut alors proclamer qu'il est le gagnant.
   NB : la complexité a été améliorée par rapport aux exercices **??** et 3 mais dans le cadre d'un cycle bi-directionnel (plus puissant donc).
4. Voici l'algorithme :

```
Phase i
    Tant que (Etat == actif)
    Envoyer(< Val_i, 2^i >);
    i = i + 1;

Reception Message(< V, TTL >)
    Si (Etat == actif) et (TTL <> 1) alors
    Faire transiter le message
    Sinon
        Si V > val_i alors etat = passif
        Sinon Si V = val_i alors Elu;
```

TTL :time to live

un autre algorithme :

```
Reveil par signal      etat=actif
    i = 0
    Tant que (etat = actif)
        Envoyer (election, val_i, 2^i)
        ATTendre (reception messages)
Réception du message (election, V, D) sur j
    si etat = actif
```

```
        si V > Val_j
            Alors etat = perdant;
        Si (V = Val_j)
            etat = gagnant
    Si etat = perdant
        Si D − 1 > 0
            Envoyer (election, V, D − 1)
```

5. L'inconvénient de cet algorithme c'est que certains sites passifs recevront, en tant que destinataire, des messages qui ne servent à rien. Il vaut mieux envoyer à un site encore actif (voir l'exercice suivant).

Fin correction exercice 8

Exercice 9 – L'algorithme de Franklin

L'algorithme proposé dans cet exercice est un raffinement du précédent.. Le code pour un site i est :

```
Lors du réveil du site i
    etat_i := actif;
    Tant que (etat_i = actif) faire
        Envoyer(val_i) à droite;
        Envoyer(val_i) à gauche;
        Attendre un message (V_d) venant de la droite;
        Attendre un message (V_g) venant de la gauche;
        V := max{V_d, V_g};
        Si (V ≥ val_i) alors etat_i := passif;
    Si (val_i = V) alors
        gagnant_i = i;
        Prévenir les autres sites
    Sinon faire suivre tous les messages.
```

1. En quoi cet algorithme est-il différent du précédent ? Peut on remplacer $\geq$ par $>$?
2. Combien de messages sont générés ?

Correction exercice 9

Cette version permet de restreindre l'envoie des messages jusqu'au prochain actif au lieu de l'envoyer

sur une distance donnée a priori.On ne peut pas supprimer $\leq$, dans le cas contraire on bouclerait. Du coup les messages ne contiennent plus le compteur de sauts et ils ne parcourent plus des distances inutiles. En effet, dans l'exercice précédent il y a envoie des messages à une distance 2^i pour un site actif. On remarque que cet algorithme marche, de manière implicite, par phase. A chaque phase au moins la moitié des sites deviennent passifs ; Il faut donc au plus $\log_2 n$ phases avant la proclamation des résultats. Pendant chaque phase chaque lien est traversé une fois dans chaque sens. Ce qui conduit à $2n$ messages par phase. D'où une complexité totale d'au plus $2n \log_2 n$ messages. La terminaison est garantie lorsque le site gagnant reçoit sa propre valeur. On peut sortir du Tant que à ce moment là. Pour décider du vainqueur, il faut sortir de la boucle Tant que.

Fin correction exercice 9

Exercice 10 – Election dans un arbre.

On suppose que l'on a un réseau d'ordinateurs connectés en *arbre* (voir exemple figure 16). Chaque canal est 'FIFO'. Les sommets (ou sites) ne communiquent directement qu'avec leurs *voisins* immédiats. Chaque sommet a un *identificateur* unique.

1. Ecrire un algorithme réparti qui :
   - "Réveille" tous les sommets à partir d'au moins un *initiateur*.
   - Calcule la plus petite étiquette de l'arbre (le gagnant étant le sommet qui possède cette étiquette). Pour cela :
     - Faire un parcours des feuilles vers l'intérieur de l'arbre en calculant la plus petite étiquette au fur et à mesure.
     - Propager le résultat de l'élection de voisin en voisin. Si un sommet reçoit son étiquette, il se déclare *vainqueur*, sinon *perdant*.

   Pour cela vous pourrez utiliser les variables suivantes :
   Un site p a les variables locales suivantes :
   $reveille_p$ booléen initialisé à $Faux$,
   $voisins\_reveilles_p$ entier initialisé à 0,
   rec_p tableau de $|voisins_p|$ booléens initialisés à $Faux$,
   val_p la valeur pour l'élection, v_p, initialisée à la valeur val_p,
   $etat_p \in \{neutre, gagnant, perdant\}$, initialisé à $neutre$.

   Vous devez commenter votre algorithme.
2. Proposez une exécution possible sur l'arbre de la figure 16.
3. Quelle est la complexité en nombre de messages échangés ?
4. Pourquoi commencer par les feuilles ?

5. Que faire si l'on perd l'hypothèse de l'identificateur unique ?

Correction exercice 10

Question 1. On supposera que l'arbre contient n sommets. Un site p a les variables locales suivantes :
$reveille_p$ booléen initialisé à $Faux$,
$voisins\_reveilles_p$ entier initialisé à 0,
rec_p tableau de $|voisins_p|$ booléens initialisés à $Faux$,
val_p la valeur pour l'élection, v_p, initialisée à la valeur val_p,
$etat_p \in \{neutre, gagnant, perdant\}$, initialisé à $neutre$.

```
Si (p est initiateur) alors
    reveille_p := Vrai;
    Pour tout (q ∈ voisins_p) faire
        Envoyer(< REVEIL >) à q;
Lors de la réception d'un message pour la première fois
 (*** PHASE DE REVEIL : ***)

    Tant que (voisins_reveilles_p < |voisins_p|) faire
        Attendre Recevoir message(< REVEIL >);
        voisins_reveilles_p := voisins_reveilles_p + 1;
        Si (non reveille_p) alors
            reveille_p := Vrai;
            Pour tout (q ∈ voisins_p) faire
                Envoyer(< REVEIL >) à q;
 (*** PHASE DE CALCUL : ***)

    v_p := val_p;
    Tant que (|{q : non rec_p[q]}| > 1) faire
        Attendre(Recevoir(< TOKEN, r >) de q);
        rec_p[q] := Vrai;
        v_p := min{v_p, r};
     Envoyer(< TOKEN, v_p >) à q_0 tel que rec_p[q_0] = Faux;

    Attendre Recevoir(< TOKEN, r >) de q_0;
    v_p := min{v_p, r};
    Si (v_p = val_p) alors etat_p := gagnant;
    Sinon etat_p := perdant;
    Pour tout (q ∈ voisins_p, q ≠ q_0) faire
        Envoyer(< TOKEN, v_p >) à q;
```

L'algorithme consiste à réveiller tous les sommets à partir d'un (ou plusieurs) initiateur. Ensuite,

à cause du premier `Tant que` (de la phase de calcul), les feuilles envoient leur valeur. Un sommet interne de l'arbre envoie sa valeur au seul sommet qui ne lui a rien envoyé (c'est la condition stricte). A chaque étape, la valeur minimale est calculée. La dernière partie consiste à propager la valeur minimale dans tous le réseau.
Question 2. Nombre de messages échangés. Pendant la phase de réveil : deux messages $< REVEIL >$ et deux messages $< TOKEN >$ sont envoyés sur chaque lien. En tout donc 4 messages par lien. D'où $4n - 4$ messages en tout. **Fin correction exercice 10**

Exercice 11 – Un algorithme asynchrone dans un réseau complet : algorithme de Humblet

Dans le but de développer des algorithmes efficaces dans le cas d'un graphe complet, nous allons utiliser la technique d'acquisition de territoire.

Dans la procédure d'acquisition chaque candidat essaye de capturer un voisin à chaque itération ; en utilisant un message du type *capture* contenant son identité et le nombre de sommets déjà capturé avec la variable $stage(x)$. Si la tentative est un succès, le voisin attaqué devient capturé, et le sommet candidat entre dans une nouvelle itération sinon le candidat devient passif. Un sommet capturé retient l'identité, et la valeur de la variable $stage(x)$ et le lien de son maître (l'identité de celui qui l'a capturé).

1. Remplissez les spécifications suivantes :
   Nous supposons qu'un candidat x envoie un message du type *capture* à y :
   (a) Si y est candidat :
      i. Si $stage(x) > stage(y)$, alors conclure sur l'attaque.
      ii. Si $stage(x) = stage(y)$, donner les états possibles pour x et y.
      iii. Si $stage(x) < stage(y)$, alors conclure sur les états de x et de y
   (b) Si y est passif, conclure sur le résultat de l'attaque de x sur y.
   (c) Si y est déjà capturé alors x doit défaire le maître de y, noté z avant de capturer y. Dans ce cadre y envoie un message du type Warning à z avec l'identité de x et $stage(x)$.
      i. Si z est candidat. Donner les différents cas de figures.
      ii. Dans le cas contraire donner tous les cas de figures.
   (d) Si l'attaque est un succès, conclure sur les états de x et y sur la valeur de $stage(x)$.
2. Donner l'algorithme maintenant.
3. Applique l'algorithme de Humblet sur le graphe donné par la figure 17.
4. Montrer que l'algorithme répond positivement au problème de l'élection.
5. Que se passe-t-'il quand un sommet a capturé au moins $n/2 + 1$ sommets ?
6. Combien de sommets peuvent être candidat à l'itération i ? Déterminer la taille maximum d'un territoire à l'itération i ? Les territoires sont-ils disjoints ?

7. Combien de messages au maximum sont nécessaires lorsque un sommet procède à une demande capture ?
8. Combien de stages sont nécessaires pour la terminaison de l'algorithme ?
9. Conclure sur la complexité en nombre de messages.
10. Donner un exemple pour illustrer le pire cas concernant la complexité.
11. Donner la complexité en temps.

Correction exercice 11

1. Remplissez les spécifications suivantes :

   Nous supposons qu'un candidat x envoie un message du type *capture* à y :

   (a) Si y est candidat :

      i. Si $stage(x) > stage(y)$, alors l'attaque est un succès.

      ii. Si $stage(x) = stage(y)$, alors l'attaque est un succès si $id(x) < id(y)$, sinon x devient passif.

      iii. Si $stage(x) < stage(y)$, alors x devient passif.

   (b) Si y est passif, alors l'attaque est un succès.

   (c) Si y est déjà capturé alors x doit défaire le maître de y, noté z avant de capturer y. Dans ce cadre y envoie un message du type Warning à z avec l'identité de x et $stage(x)$.

      i. Si z est candidat. Si $stage(z)$ est plus grand que $stage(x)$, ou $stage(x) = stage(z)$ et $id(z) < id(x)$, alors l'attaque sur y est un échec. z notifie à x son échec via y.

      ii. Dans les autres cas (z est déjà passif ou capturé, z est candidat avec un territoire plus faible que x, ou même stage mais avec une identité plus importante que x, l'attaque sur y est un succès. z notifie à x via y et si z était candidat alors il devient passif.

   (d) Si l'attaque est un succès, , y est capturé par x, x incrémente $stage(x)$ et procède à une nouvelle itération.

   Il est important de noter que chaque tentative de capture pour un sommet candidat coute exactement deux messages (un pour le message *capture* et l'autre pour sa notification) si le voisin est également candidat ou passif. Dans le cas contraire si le sommet a déjà été capturé, deux autres messages additionnels seront échangés.

   Nous pouvons optimiser l'utilisation du nombre de messages en bloquant les messages en y pour z n'ayant aucun chance de capturer z. Ainsi le sommet y transmettra un message du type warning à z et attent le retour de z, tout en stockant les messages venant d'autres sommets t ayant un stage plus grand et/ou une identité plus petite. Si l'attaque est un succès pour x alors y change de maître et envoie les messages de la file d'attente à x. Avec ce changement, il y a une exclusion mutuelle sur les territoires pour deux sommets candidats.

2. Voici l'algorithme

ETAT : endormi En cas de réveil spontané
$stage_i = 1$; value:=$id(x)$
$Voisins_i := N(x)$
$next = head(voisins_i)$;
envoyer(capture,$stage_i$,$value_i$) à next
etat:=candidate

ETAT : endormi En cas de réception de (capture,$stage_x$, $valeur_x$) de x
envoyer(acception,$stage_x$,$valeur_x$) à x
$stage_i = 1; maitre_i := x$
$stage_{dumatre} := stage_x + 1$

ETAT : candidat En cas de réception de (capture,$stage_x$, $valeur_x$) de x
Si $stage_x < stage_i$ ou ($stage_x = stage_i$ et $valeur_x > valeur_i$) alors envoyer(échec,$stage_i$) à x Sinon envoyer(acception,$stage_x$,$valeur_x$) à x
$maitre_i := x$
$stage_{dumatre} := stage_x + 1$
etat:=capturé

ETAT : candidat En cas de réception de (acceptation,$stage_x$, $valeur_x$) de x
$stage_i$++ Si $stage_i \geq 1+n/2$ alors envoyer(terminaison) à $voisin_i$ Sinon $next =$ $Head(voisin_i)$ envoyer(capture,$stage_i$,$value_i$) à next

ETAT : candidat En cas de réception de (échec,$stage_x$) de x
état=passif

ETAT : candidat En cas de réception de (alerte,$stage_x$, $valeur_x$) de y
Si $stage_x < stage_i$ ou ($stage_x = stage_i$ et $valeur_x > valeur_i$) alors envoyer(non,$stage_i$) à y Sinon envoyer(oui,$valeur_x$) à y
etat:=passif

ETAT : passif En cas de réception de (capture,$stage_x$, $valeur_x$) de x
Si $stage_x < stage_i$ ou ($stage_x = stage_i$ et $valeur_x > valeur_i$) alors envoyer(échec,$stage_i$) à x Sinon envoyer(acception,$stage_x$,$valeur_x$) à x
$maitre_i := x$
$stage_{dumatre} := stage_x + 1$
etat:=capturé

ETAT : passif En cas de réception de (alerte,$stage_x$, $valeur_x$) de y
Si $stage_x < stage_i$ ou ($stage_x = stage_i$ et $valeur_x > valeur_i$) alors envoyer(non,$stage_i$) à y Sinon envoyer(oui,$valeur_x$) à y
etat:=passif

ETAT : capturé En cas de réception de (capture,$stage_x$, $valeur_x$) de x
Si $stage_x < stage_{maitre}$
envoyer(échec,$stage_{maitre}$) à x Sinon $attaquant_i = x$
envoyer(alerte,$stage_x, valeur_x$) à $maitre_i$
enfile($voisin_i - maitre_i$)
etat:=capturé

ETAT : capturé
En cas de réception de (non,$stage_y$) de x
$Open(voisin_i)$
envoyer(échec,$stage_x$) à attaquant
Sinon
ETAT : capturé
En cas de réception de (oui,$stage_y$) de x
$stage_{maitre} = stage_x + 1$
envoyer(échec,$stage_x$) à attaquant
$Open(voisin_i)$
envoyer(acceptation,$stage_x$,$valeur_x$) à attaquant

ETAT : capturé
En cas de réception de (alerte,$stage_y$,$valeur_y$) de x
Si $stage_y < stage_{maitre}$ alors
envoyer(non,$stage_{maitre}$) à x
Sinon
envoyer(oui,$stage_x$,$valeur_x$) à x

3. L'application
4. A la fin de l'algorithme, il nous reste un seul sommet en lice (il suffit de supposer que deux sites sont élus, et ceci est impossible), donc c'est l'élu. Il est important de noter que l'élu n'est pas forcément celui qui a la plus petite identité.
5. L'algorithme se termine car aucun autres sommets ne peut avoir un territoire supérieur à celui qui a un territoire de taille $n/2 + 1$. Nous avons prouvé la terminaison.
6. Les territoires de tailles i sont disjoints, ainsi ils ne peuvent pas être plus grand que $n_i \leq n/i$ à l'itération i. A chaque itération i les territoires sont de taille i. Il ne peut y avoir plus que

$n_i \leq n/i$ candidats à l'itération i.

7. Il existe 4 messages au maximum.
8. Il y a $n/2 + 1$ itérations.
9. En rajoutant $n - 1$ messages pour avertir les voisins, nous trouvons :

$$n - 1 + \sum_{i=1}^{n/2+1} 4n_i \leq n + 1 + \sum_{i=1}^{n/2} 4n = 4nH_{n/2} + n + 1$$

10. Considérons un sommet y capturé qui reçoit une attaque après sa capture par un sommet x_1 au niveau i. Selon l'algorithme y transmet l'attaque à son chef z et attend la réponse, pour notifier à x_1 et le succès ou l'échec de l'attaque. Supposons maintenant que pendant l'attente, y reçoit des autres attaques venant des sommets $x_2, x_3, \ldots, x_k$ et toutes au niveau i. Nous avons donc $k - 1$ nouveaux messages envoyés par y à z. Remarquons que si $id'z) > id(x_1) > id(x_2) > \ldots > id(x_k)$, chaque attaque sera satisfaite y sera capturé par chacun des sommets x_i l'un à la suite de l'autre.
11. La complexité en temps est $O(n)$.

Fin correction exercice 11

4 Algorithme pour les généraux Byzantins

Exercice 12 – Généraux byzantins

1. Appliquer l'algorithme avec $m = 2$ et $n = 7$ et le commandant admet un un comportement normal.
2. Exhiber un exemple pour $m = 2$ et $n = 6$ pour lequel nous n'arrivons pas à un consensus.

Correction exercice 12

1. Using an example can make this algorithm easier to understand. Assume we have 2 traitors and 5 loyal generals. If the loyal general obtain consensus of the message from a loyal general (general 1), after mutually exchanged messages received from general 1, each identical copy of the message (one from general 1 directly and 4 other from exchanges) and 2 copies from traitors which may be di from the true message. Under majority voting, the original message can be recovered. The more complicated case would be the me a traitor (general 7) : He send out attack (1) message to generals 1 to 3 and retreat (0) message to generals 4 to 6.

$v_{71} = 1$	$v_{712} = 1$	$v_{713} = 1$	$v_{714} = 1$	$v_{715} = 1$	$v_{716} = 1$
$v_{721} = 1$	$v_{72} = 1$	$v_{723} = 1$	$v_{724} = 1$	$v_{725} = 1$	$v_{726} = 1$
$v_{731} = 1$	$v_{732} = 1$	$v_{73} = 1$	$v_{734} = 1$	$v_{735} = 1$	$v_{736} = 1$
$v_{741} = 0$	$v_{742} = 0$	$v_{743} = 0$	$v_{74} = 0$	$v_{745} = 0$	$v_{746} = 0$
$v_{751} = 0$	$v_{752} = 0$	$v_{753} = 0$	$v_{754} = 0$	$v_{75} = 0$	$v_{756} = 0$
$v_{761} = 0$	$v_{762} = 0$	$v_{763} = 0$	$v_{764} = 0$	$v_{765} = 0$	$v_{76} = 0$

TABLE 1 – Tableau synthétique des états :

$v_{761} = 0$	$v_{7612} = 0$	$v_{7613} = 0$	$v_{7614} = 0$	$v_{7615} = 0$
$v_{7621} = 1$	$v_{762} = 1$	$v_{7623} = 1$	$v_{7624} = 1$	$v_{7625} = 1$
$v_{7631} = 0$	$v_{7632} = 0$	$v_{763} = 0$	$v_{7634} = 0$	$v_{7635} = 0$
$v_{7641} = 1$	$v_{7642} = 1$	$v_{7643} = 1$	$v_{764} = 1$	$v_{7645} = 1$
$v_{7651} = 0$	$v_{7652} = 0$	$v_{765} = 0$	$v_{7654} = 0$	$v_{765} = 0$

TABLE 2 – Tableau synthétique des états (suite) :

Let us define v to be the message in concern (the message from general 7) and v_p be the message as transversed in path p. The first step $OM(2)$ produces the following :

$$v_{71} = 1, v_{72} = 1, v_{73} = 1, v_{74} = 0, v_{75} = 0, v_{76} = 0$$

Then each of the node will exchange the message they received, which the loyal lieutenants will forward the message faithfully but traitorous lieutenant may alter the message, e.g.,

Now consider node 1. It received $v_{761} = 0$ but it is not sufficient for him to confirm $v_{76} = 0$. Therefore, he perform $OM(1)$ with pretending node 6 is the general in this round sending out $v_{76} = v_{761}$. Similarly for nodes 2 to 5, and we obtained :

Therefore, nodes 1 to 5 deduced, using $OM(0)$, that :

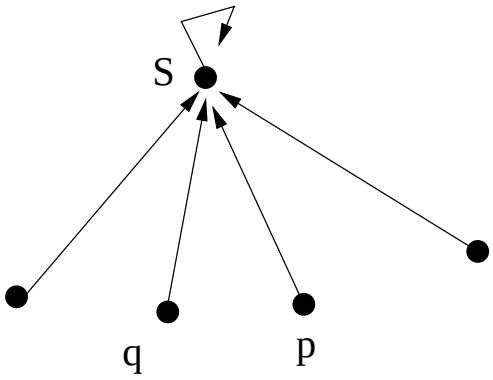
$$\begin{aligned}
v'_{761} &= majority(v_{761}, v_{7621}v_{7631}v_{7641}v_{7651}) = 0 \\
v'_{762} &= majority(v_{7612}, v_{762}v_{7632}v_{7642}v_{7652}) = 0 \\
v'_{763} &= majority(v_{7613}, v_{7623}v_{763}v_{7643}v_{7653}) = 0 \\
v'_{764} &= majority(v_{7614}, v_{7624}v_{7634}v_{764}v_{7654}) = 0 \\
v'_{765} &= majority(v_{7615}, v_{7625}v_{7635}v_{7645}v_{765}) = 0
\end{aligned}$$

Thus, the consensus reached for $v'_{76} = 0$. Similarly for the unambiguous value of $v_{71}, v_{72}, v_{73}, v_{74}, v_{75}$. Then all loyal lieutenant

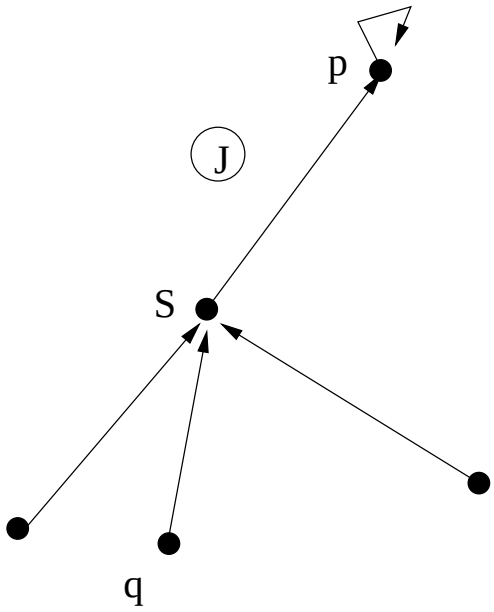
$$v'_7 = majority(v'_{71}, v'_{72}, v'_{73}, v'_{74}, v'_{75}, v'_{76})$$

which is a tied vote and default decision is applied.

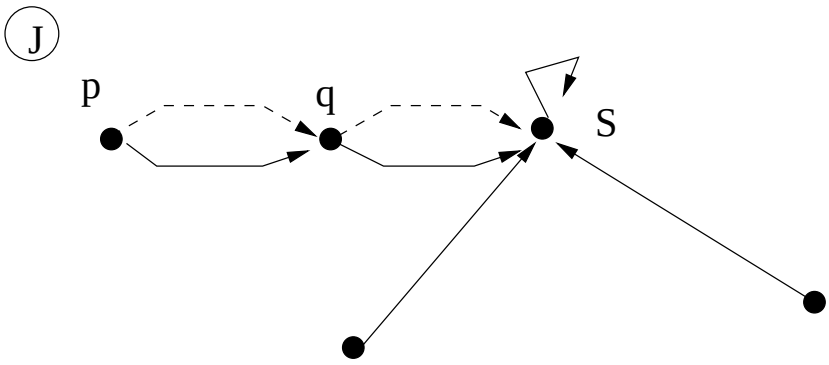
Fin correction exercice 12

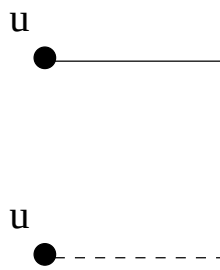


p et q veulent la ressource.



p est traité avant q.





S est traité.

FIGURE 13 – *Exemple-NaimiTrehel*

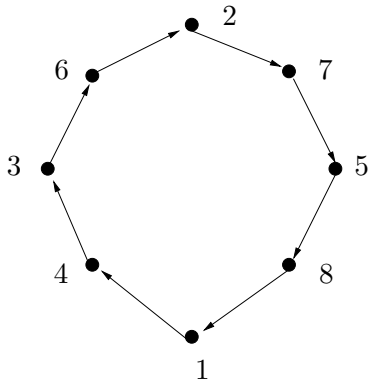
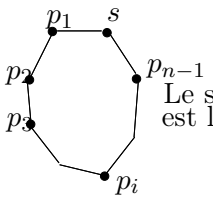


FIGURE 14 – *Une configuration dans* C_8.



Le sens de circulation des messages est le sens des aiguilles d'une montre

FIGURE 15 – *Arrangements des identités sur un anneau.*

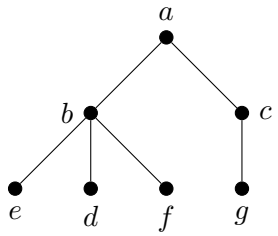


FIGURE 16 – Un réseau en arbre.

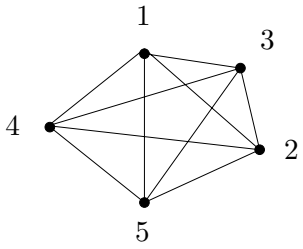


FIGURE 17 – *Graphe complet à* 5 *sommets*