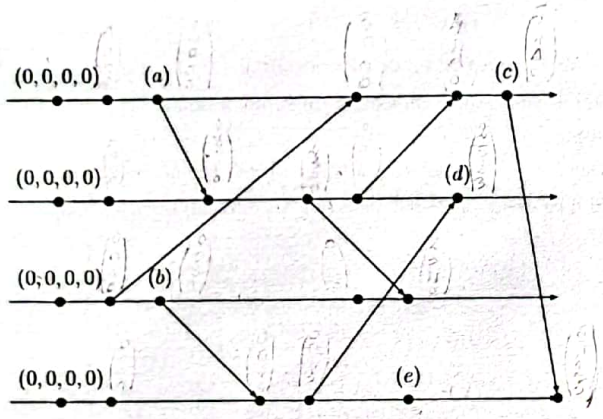

FIGURE 1 – Chronogramme avec les horloges vectorielles.

Université de Montpellier
MasterM1 d'Informatique

3 janvier 2023
Durée: 2h

Algorithmes Distribués
Examen 09 janvier 2023

Les seuls documents du type cours/TD sont autorisés. La note prendra en compte la clarté des explications, l'orthographe et la grammaire.

Exercice 1

1. On considère le chronogramme donné par la figure 1. Compléter le schéma en indiquant pour chaque événement l'horloge de Mattern associée.
   Est-ce que les vecteurs d'horloge de Mattern sont comparables ? Quelles sont les événements dans la passé de (c), de (d) et de (e) ? Trouver les événements qui sont en concurrence.
2. On considère les horloges matricielles données par la figure 2. Compléter le schéma en indiquant pour chaque événement son horloge matricielle.
3. Lors de l'envoi de messages (modèle vectoriel ou matriciel), est-ce que toutes les données sont utiles à transmettre ? Quelle solution proposerez-vous pour minimiser la quantité de données qui circule sur le réseau ?

□

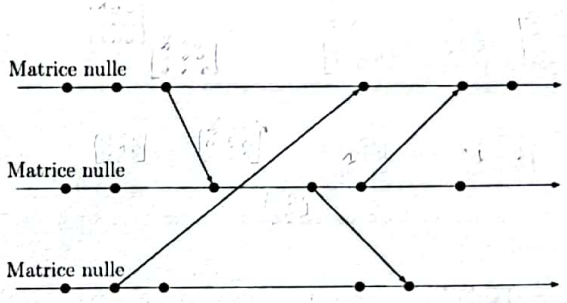

FIGURE 2 – Chronogramme avec les horloges matricielles.

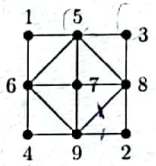

FIGURE 3 – Graphe pour le problème de l'élection.

Exercice 2

Appliquer l'algorithme YO-YO sur la Figure 3. Vous donnerez toutes les étapes. □

Exercice 3

Dans cet exercice, nous nous intéressons à un algorithme résolvant le problème de l'élection dans un anneau unidirectionnel. C'est l'algorithme de LeLann.

Chaque site p dispose d'une variable $List_p$ qui calcule la liste des identités des initiateurs, et un état $state_p$ qui définit l'état du site p avec $state_p \in \{cand, sleep, lost\}$.

Voici l'algorithme qui est proposé :

Pour un initiateur p, à exécuter qu'une seule fois
  $state_p := cand$;
  Envoyer(jeton,p) à $Next_p$

Pour un initiateur p, lors de la réception de (jeton,q)
Si $q \neq p$
  Alors $list_p := list_p \cup \{q\}$;

Envoyer(jeton,q) à $Next_p$;
Sinon Si $p = \min(list_p)$;
Alors $state_p := leader$;
Sinon $state_p := lost$;

Pour un non initiateur p, **Lors de la réception de (jeton) de** q
Envoyer (jeton,q) à $Next_p$;
Si $state_p = sleep$;
Alors $state_p := lost$;

1. Donner un exemple d'exécution sur l'anneau donné par la figure 4 tel que tous les sites sont initiateurs.

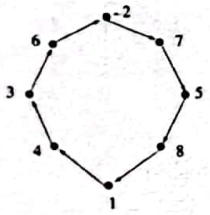

FIGURE 4 – *Anneau orienté*

2. Donner le principe de l'algorithme.
3. Donner sa complexité en nombre de messages échangés, et en temps au pire cas.
4. Pourquoi l'hypothèse des communications FIFO est primordiale ?
5. Expliquer la finitude de l'algorithme.

□

Exercice 4

Dans cet exercice on s'intéresse à calculer le diamètre d'un arbre. Le diamètre d'un graphe est la plus grande distance (en nombre d'arêtes) d'un plus court chemin entre chaque paire de sommets.

1. Calculer le diamètre de l'arbre donné par la figure 5.
2. Proposer un algorithme distribué qui détermine le diamètre d'un arbre.

3. Procéder à une exécution de votre algorithme sur la figure 5.
4. Notion de site saturé : un site est dit saturé lorsqu'il a reçu tous les messages de ces voisins.
   (a) Dans un arbre, combien y-a-t'il au plus de sommets saturés ?
   (b) Si il en existe plusieurs, quels sont les relations entre les sommets saturés ?

□

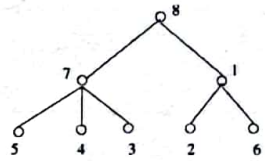

FIGURE 5 – Un réseau en arbre.

Université de Montpellier — 1er mars 2023
MasterM1 d'Informatique — Durée: 2h

Algorithmes Distribués
Examen 27 mars 2023

Les documents sont autorisés. La note prendra en compte la clarté des explications.

Exercice 1

On considère une exécution répartie sur trois sites, pour laquelle on n'a pu récupérer que l'horloge vectorielle du dernier événement sur chacun des sites :

$\begin{bmatrix} 3 & 3 & 2 \end{bmatrix}$ sur le site 1 ; $\begin{bmatrix} 1 & 3 & 2 \end{bmatrix}$ sur le site 2 ; $\begin{bmatrix} 1 & 0 & 2 \end{bmatrix}$ sur le site 3.

1. Combien y a-t-'il eu d'événements sur chacun des sites ?
2. Peut-on connaître le nombre de messages émis par le site 1 ?
3. Peut-on savoir si le site 2 a envoyé un message au site 1 ?
4. Donner un scénario pour lequel les vecteurs d'horloges sont ceux donnés précédemment.
5. Compléter le schéma donné par la figure 1 pour reconstruire l'historique des événements jusqu'aux dates indiquées. Vous donnerez la date (matricielle) de chaque événement.

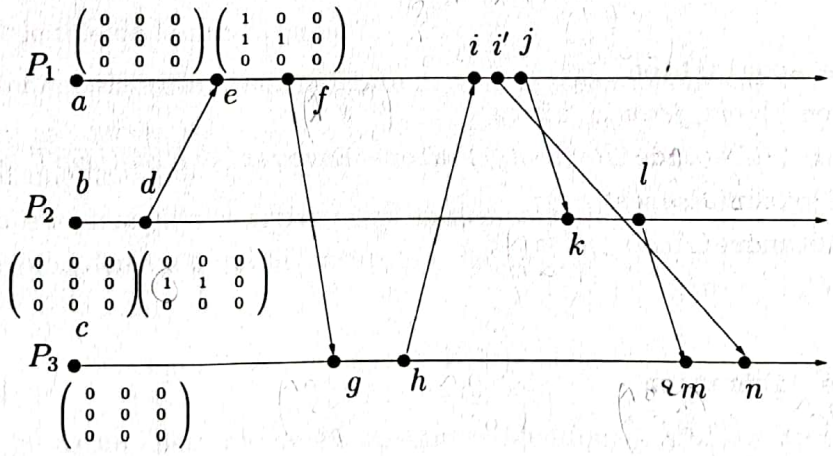

FIGURE 1 – *Schéma à compléter.*

Chacune de vos réponses devra être clairement justifier.

□

Exercice 2

Le but de cet exercice est de proposer un algorithme de parcours en profondeur d'un graphe (non-orienté) en utilisant la connaissance des voisins.

Si les sites connaissent l'identité de leurs voisins, il est possible de réduire le nombre de messages lors du passage du jeton (comme dans les autres algorithmes) en incluant la listes des sites visités dans le jeton. Le site p qui reçoit le jeton avec la liste L des sites visités ne le transfère pas le jeton aux sites appartenant à la liste L. Les variables $used_p[q]$ peuvent être omises car si le site p a transmis avant le jeton au sommet q, alors $q \in L$. Proposez un algorithme utilisant $2n-2$ messages en $2n-2$ unités de temps.

Voici les variables impliquées : var : $father_p$; init $udef$

Voici un début de code.

Pour l'initiateur; à faire une seule fois

$father_p = p$; `choisir` $q \in Voisin_p$

`Envoyer` $< tlist, \{p\} >$ `à` q

□

Exercice 3 Exclusion mutuelle dans les arbres : algorithme de Raymond

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.

```
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;
            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;
```

1. Donner le principe et le fonctionnement de l'algorithme.
2. Quel est le rôle de la fire *Request* ?
3. Appliquer cet algorithme sur la figure 2, avec le scénario suivant :
   - c 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.

□

Exercice 4

Nous considérons un ensemble de M ressources identiques, chacune devant être utilisée en section critique. ; nous nous trouvons donc dans le cas d'un seul type de ressource dont il existe M exemplaires.

Dans cet exercice, nous nous plaçons dans le cas de la demande d'un site i concernant un nombre quelconque $k_i, (1 \leq k_i \leq M)$ de ressources parmi M, deux demandes différentes du site i pouvant partir sur des nombres k_i différents.

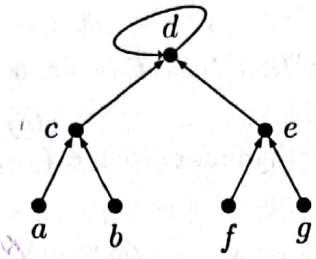

FIGURE 2 – Anti-arborescence.

1. Rappeler les deux grands principes à assurer pour une allocation correcte et expliquer leur rôle ?
2. Quel algorithme connu (vu en cours/TD) pouvons-nous utiliser ? Quelle conséquence sur le problème ?

   Pour la suite nous allons rajouter la propriété suivante :

   Propriété d'efficacité : lorsque cela est possible et ne peut remettre en question ni la sûreté ni la vivacité, plusieurs demandes doivent être satisfaites simultanément.
3. Afin de ne pas mettre en défaut la sûreté, une solution consiste à donner à un site i qui veut utiliser k_i ressources une vision cohérente du nombre de ressources utilisées par les autres sites. Appelons $utilise_i[j]$ la perception qu'a le site i du nombre de ressources utilisées par j. Donner l'inéquation reliant M, k_i et la variable $utilise_i[j]$ garantissant la sûreté et justifier-la.
4. Quel mécanisme classique va permettre de garantir la vivacité ?
5. Quel est le nombre de messages nécessaires à une utilisation des ressources ?
6. Comment l'algorithme réalise la propriété d'efficacité ?
7. Donner le code maintenant.

□