CONCOURS D'ACCES AU DOCTORAT EN INFORMATIQUE

Option : Systèmes Informatiques

Epreuve : Algorithmiques Distribués (Sujet 1)

07 Novembre 2019 / Durée : 02h

DIRECTIVES PEDAGOGIQUES:

– *Documentation, Smartphones et Calculatrices ne sont pas autorisées*
– *Il est formellement interdit de remplir la rubrique numéro d'anonymat.*
– *Il est strictement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu*

Exercice 1. (8 points)

Q1.	a représente **un événement Interne (traitement interne)** **b** représente **un événement de réception** **c** représente **un événement d'émission**	**(1pt)**
Q2.	VH (**a**)= **2** ; VH (**b**)=10; VH (c)= **2** ; VH (**d**)= 8	**(1pt)**
Q3.	l'ordre sur les estampilles est-il strict? Justifiez. **Non, n'est pas strict** parce que plusieurs événements possédant la même valeur de l'horloge (exemple VH(a)=VH(c)=2).	**(1pt)**
	Pour le rendre strict : on adjoint à la valeur de l'horloge logique d'un site l'identification du site. Ainsi, pour tous événements e_i et e_j créés par deux sites Si et Sj : $e_i << e_j$ **si est seulement si** 1) **Hi (e_i) < Hj (e_j), ou** **2) Hi (e_i) = Hj (e_j) et S_i vient avant S_j dans l'ordre des processus**	**(1pt)**
Q4.	**C1 et C2 sont-elles cohérentes ?** **C1 est cohérente** **car** tous les messages reçus dans la coupure ont été envoyés dans la coupure **C2 n'est pas cohérente** **car** le site S1 a reçu un message qui a été envoyé après la coupure.	**(2pt)**
Q5.	VHm(**a**) = **(2, 0, 0, 0)** ;VHm(**b**) = **(1, 6, 7, 7)** ; VHm(**c**) = **(0, 0, 2, 0)** ; VHm(**d**) = **(0, 1, 7, 6)**	**(2pt)**

Exercice 2. (12 points)

Pour un processus Pi,

1) ***Que représente sa variable locale Rcv[i] ? (1pt)***

Rcv[i] est le nombre de fois que le processus Pi est devenu demandeur d'accès à la section critique.

2) ***Que représente sa variable locale Rcv[j] avec j≠ i ? (1pt)***

Rcv[j] est le nombre de fois que le processus Pi a su que Pj est demandeur d'accès à la section critique (Nombre de requêtes de Pj reçues par Pi).

3) ***Quel rôle joue-t-il la variable locale Lock ? (1pt)***

Etant donné que les procédures de chaque processus peuvent être lancées concurremment, la variable Rcv doit être protégée car elle est manipulée en lecture/écriture. D'où le rôle du verrou Lock. Nous pouvons considérer le cas où le processus Pi reçoit une requête alors qu'il a invoqué la procédure d'entrée en section critique.

4) ***Rappelez les propriétés de sûreté et de vivacité que doit vérifier toute solution au problème de la section critique.***

La sûreté : Elle se résume en la propriété d'exclusion mutuelle qui stipule qu'à tout instant au plus un processus est dans sa section critique. ***(1pt)***

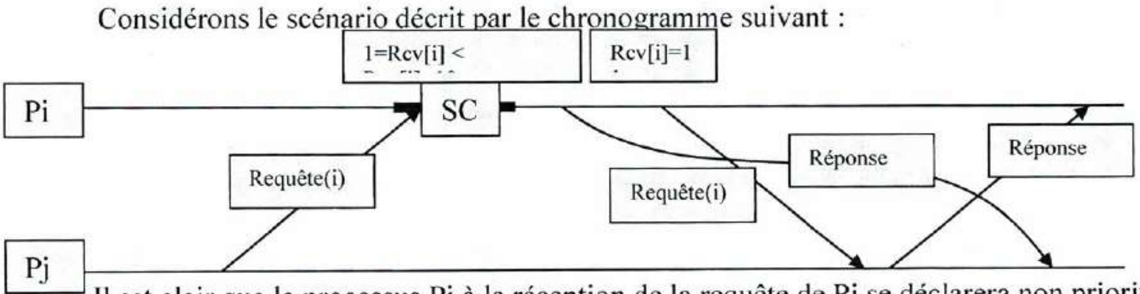
La vivacité : Tout processus demandeur d'accès à sa section critique finira par y accéder au bout d'un temps fini. ***(1pt)***

5) ***Montrer que dans le cas où dès l'état initial deux et seulement deux processus demandent l'entrée en section critique simultanément, ces derniers peuvent accéder à la section critique en même temps ? Sachant que les identités des processus sont totalement ordonnées, proposer une modification de l'instruction d'évaluation de la priorité pour éviter un tel cas. (3pts)***

- Dans ce cas le processus Pi, respectivement Pj, a incrémenté sa variable Rcv[i], respectivement Rcv[j]. De ce fait, lors de la réception de la requête de Pj par Pi, respectivement de Pi par Pj, chacun des deux processus trouvera qu'il n'est pas prioritaire et donnera sa permission à l'autre processus. D'où la violation de la propriété d'exclusion mutuelle.
- Nous pouvons utiliser l'ordre lexicographique défini comme suit :
  (K,i) < (L,j) si est seulement si
      Soit K < L
      Soit K = L et i < j
  L'évaluation de la priorité est modifiée comme suit :
  Priorité := (état = "dedans") **Or** (état = "demandeur" **and** (Rcv[i],i) < (Rcv[j],j)) ;

6) ***Montrer, à travers un scénario, que dans le cas où les liaisons ne sont pas régulières (fifo) la sûreté n'est pas vérifiée. (4pts)***

Considérons le scénario décrit par le chronogramme suivant :



Il est clair que le processus Pj à la réception de la requête de Pi se déclarera non prioritaire et donnera par conséquent sa permission à ce dernier, d'où la violation de la propriété d'exclusion mutuelle. De ce fait la régularité des liaisons est requise.

L'Algorithme :

Pour un processus Pi (i=0 .. N-1), les énoncés définissant son comportement sont comme suit :

<table>
<tr>
<td>
<u>Déclaration</u>
Priorité : Booléen ;
Etat : {"dehors", "demandeur", "dedans"}
Lock : Verrou ;
File d'attente : File de processus ;
Rcv : tableau [0..N-1] d'entiers ;
nombre de réponses : entier
<u>Initialisation</u>
état := "dehors";
File d'attente := nil ;
<u>Pour</u> chaque processus j =0 .. N-1
    Rcv[j] := 0 ;
<u>Finpour</u>
</td>
<td>
<u>Lors de l'appel à acquérir</u>
Lock := "verrouillé";
état := "demandeur";
Rcv[i]++;
nombre de réponses :=0 ;
<u>Pour</u> chaque processus j =0 .. N-1 et j≠ i
    Envoyer Requête(i) à Pj ;
<u>Finpour ;</u>
Lock := "déverrouillé" ;
Attendre (nombre de réponses = (N −1));
état := "dedans";
</td>
</tr>
<tr>
<td>
<u>Lors d'un appel à Libérer</u>
Lock := "vérouillé" ;
état:= "dehors";
<u>Repeat</u>
    Défiler (j) de la file d'attente ;
    Envoyer Réponse à Pj ;
Until file vide ;
Lock := "dévérouillé";
</td>
<td>
<u>Lors de la réception d'une réponse</u>
Lock := "vérouillé" ;
nombre de réponses++
Lock := "dévérouillé";
</td>
</tr>
<tr>
<td colspan="2">
<u>Lors de la reception de Requête(j)</u>
Lock := "vérouillé" ;
Rcv[j]++ ;
Priorité := (état = "dedans") Or (état = "demandeur" and Rcv[i] < Rcv[j]) ;
<u>If</u> Priorité <u>Then</u>
    Enfiler (j) dans la file d'attente ; /* La réponse est différée*/
<u>Else</u>   <u>Begin</u>
    Envoyer Réponse à Pj ;
    <u>End</u> ;
Lock := "dévérouillé";
</td>
</tr>
</table>

CONCOURS D'ACCES AU DOCTORAT EN INFORMATIQUE

Option : Systèmes Informatiques

Epreuve : Algorithmiques Distribués (Sujet 2)

07 Novembre 2019 / Durée : 02h

DIRECTIVES PEDAGOGIQUES:

– *Documentation, Smartphones et Calculatrices ne sont pas autorisées*

– *Il est formellement interdit de remplir la rubrique numéro d'anonymat.*

– *Il est strictement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu*

Exercice 1. (10 points)

L'utilisation des horloges logiques de **Lamport** est une solution pour régler le problème de l'exclusion mutuelle dans un contexte distribué. Ces horloges, sur un plan abstrait, prennent des valeurs qui ne peuvent que croitre. Toutefois, sur le plan pratique, ces horloges sont matérialisées par des variables entières d'intervalle de valeurs finies. Dans cet exercice nous souhaitons explorer l'alternative de considérer que même sur un plan théorique les horloges croient sur un intervalle de valeurs finies dans des situations spécifiques. Ci-après est un exemple type d'une telle situation :

Le comportement d'un processus Pi (i=0..N-1) est donné ci-dessous:

```
While (true) do
        1.Demander la ressource;
        2.Utiliser la ressource;
        3.Libérer la ressource;
Od.
```

Pour demander la ressource, un processus envoie une requête estampillée aux autres processus. À l'issue de cette requête, chaque processus répond favorablement seulement si :

(i) Il n'est pas intéressé par la ressource,

(ii) Il est intéressé mais l'estampille de sa requête est plus grande que celle de la requête reçue. Dans le cas contraire il diffère l'envoi de sa permission.

Une fois un processus a reçu un avis favorable, il utilise cette ressource pendant une période de temps finie puis il la libère.

1) Ecrire les procédures régissant la gestion de l'accès à la section critique pour chaque processus Pi (utiliser les variables locales h, lasth, état, priorité, ...pour chaque processus). Les horloges étant initialisées à 0. (4 pts)
2) Quel est l'écart maximal séparant les valeurs des deux horloges (|hi-hj|)
   a. Dans le cas de deux processus ? (1 pt)
   b. Dans le cas de N processus ? (1 pt)
3) Modifiez les procédures précédentes afin de proposer une solution dans laquelle les horloges sont bornées. (4 pts)

Exercice 2. (10 points)

L'état global d'un système distribué est constitué de l'ensemble des états locaux des processus et des canaux qui le constituent. Pour pouvoir calculer l'état global en utilisant l'algorithme de *Chandy&Lamport*, deux hypothèses sont à considérer :

Q1. Quelle hypothèse faut-il faire sur le comportement des canaux de communication ? (2,5 pt)

Q2. Quelle hypothèse faut-il faire sur la topologie du réseau de communication ? (2,5 pt)

On considère un système distribué constitué de trois processus {P_1, P_2 et P_3} reliés par des canaux de communication de manière qu'il existe entre chaque couple de processus (P_i, P_j) deux canaux différents; C_{ij} reliant P_i à P_j et C_{ji} reliant P_j à P_i. Chaque processus utilise une variable entière Xi initialisée à 2. Ensuite, après chaque unité de temps à partir de t=1 (aux instants : 1, 2, 3, ...) chaque processus met à jour sa variable en exécutant la formule suivante : Xi=Xi * i ensuite la diffuse aux autres processus.

CONCOURS D'ACCES AU DOCTORAT EN INFORMATIQUE

Option : Systèmes Informatiques

Epreuve : Algorithmiques Distribués (Sujet 1)

07 Novembre 2019 / Durée : 02h

DIRECTIVES PEDAGOGIQUES:

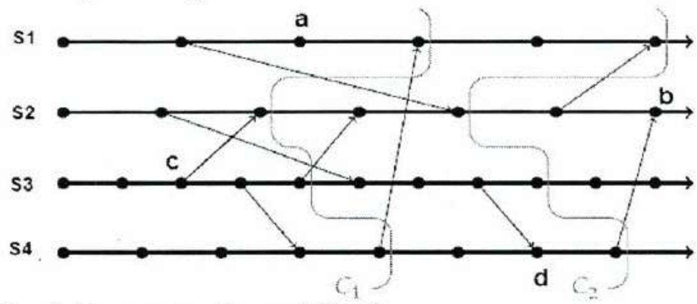
– *Documentation, Smartphones et Calculatrices ne sont pas autorisées*

– *Il est formellement interdit de remplir la rubrique numéro d'anonymat.*

– *Il est strictement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu*

Exercice 1. (8 points)

On considère un système distribué asynchrone constitué de 4 sites S1, S2, S3 et S4, s'envoyant des messages comme représenté par le diagramme de causalité suivant :



Q1. Que représentent les événements **a, b** et **c** ? (1 pt)

En considérant les horloges de *Lamport* initialisées à 0 dont la gestion se fait comme suit :

- Pour tout événement local l'horloge est incrémentée de 1
- Pour tout événement de réception de message l'horloge prend le max de sa valeur et de l'estampille du message reçu.

Q2. Donnez les valeurs des horloges des événements : **a, b, c** et **d**. (1 pt)

Q3. L'ordre sur les estampilles des événements est-il strict? Justifiez votre réponse. Sinon proposez une solution pour le rendre strict ? (2 pts)

Q4. Donnez les valeurs des horloges de *Mattern* correspondants aux événements : **a, b, c** et **d** ? (2 pts)

On considère les deux coupures C1 et C2 dans la figure précédente.

Q5. C1 et C2 sont-elles cohérentes (forment-elles des tranches) ? Justifiez votre réponse. (2 pts)

Exercice 2. (12 points)

Soit l'algorithme distribué, donné ci-après, dont l'objectif est la résolution du problème de la section critique pour N processus fonctionnant sur un réseau fiable.

Pour un processus Pi,

Q1. Que représente sa variable locale Rcv[i] ? (1 pt)

Q2. Que représente sa variable locale Rcv[j] avec $j \neq i$? (1 pt)

Q3. Quel rôle joue la variable locale Lock ? (1 pt)

Q4. Rappelez les propriétés de sûreté et de vivacité que doit vérifier toute solution au problème de la section critique. (2 pts)

Q5. Montrer que dans le cas où dès l'état initial deux et seulement deux processus demandent l'entrée en section critique simultanément, ces derniers peuvent accéder à la section critique en même temps ? Sachant que les identités des processus sont totalement ordonnées, proposer une modification de l'instruction d'évaluation de la priorité pour éviter un tel cas. (3 pts)

Q6. Montrer, à travers un scénario, que dans le cas où les liaisons ne sont pas régulières (fifo) la sûreté n'est pas vérifiée. (4 pts)

L'Algorithme :

Pour un processus Pi (i=0 .. N-1), les énoncés définissant son comportement sont comme suit :

Déclaration
Priorité : Booléen ;
Etat : {"dehors", "demandeur", "dedans"}
Lock : Verrou ;
File d'attente : File de processus ;
Rcv : tableau [0..N-1] d'entiers ;
nombre de réponses : entier
Initialisation
état := "dehors";
File d'attente := nil ;
Pour chaque processus j =0 .. N-1
        Rcv[j] := 0 ;
Finpour
	Lors de l'appel à acquérir
	Lock := "verrouillé";
	état := "demandeur";
	Rcv[i]++;
	nombre de réponses :=0 ;
	Pour chaque processus j =0 .. N-1 et j≠ i
	        Envoyer ***Requête(i)*** à Pj ;
	Finpour ;
	Lock := "déverrouillé" ;
	Attendre (nombre de réponses = (N −1));
	état := "dedans";

Lors d'un appel à Libérer
Lock := "vérouillé" ;
état:= "dehors";
Repeat
        Défiler (j) de la file d'attente ;
        Envoyer Réponse à Pj ;
Until file vide ;
Lock := "dévérouillé";
	Lors de la réception d'une réponse
	Lock := "vérouillé" ;
	nombre de réponses++
	Lock := "dévérouillé";

Lors de la reception de Requête(j)

Lock := "vérouillé" ;
Rcv[j]++ ;
Priorité := (état = "dedans") **Or** (état = "demandeur" **and** Rcv[i] < Rcv[j]) ;
If Priorité **Then**
        Enfiler (j) dans la file d'attente ; /* La réponse est differée*/
Else **Begin**
        Envoyer Réponse à Pj ;
        End ;
Lock := "dévérouillé";

CONCOURS D'ACCES AU DOCTORAT EN INFORMATIQUE

Option : Systèmes Informatiques

Corrigé Type : Algorithmiques Distribués (Sujet 2)

07 Novembre 2019 / Durée : 02h

DIRECTIVES PEDAGOGIQUES:

– Documentation, Smartphones et Calculatrices ne sont pas autorisées
– Il est formellement interdit de remplir la rubrique numéro d'anonymat.
– Il est strictement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu

---

Exercice 1. (10 points)

1) Ecrire les procédures régissant la gestion de l'accès à la section critique pour chaque processus Pi (utiliser les variables locales h, lasth, état, priorité, …pour chaque processus). Les horloges étant initialisées à 0. (4 pts)

Variables locales pour un processus Pi

R = {0,…,N-1} – {i}
H : entier initialisé à 0
Etat : {dehors,demandeur,dedans}
Lasth : entier
Priorité : Booléen
Attendus : Ensemble de processus initialisé à Ø
Différés : Ensemble de processus initialisé à Ø

Procédure exécutée lors d'un appel à acquérir

Etat := demandeur ;
H++ ;
Lasth := H ;
Attendus := R
$\forall j \in R$: Envoyer requête(lasth,i) à j ;
Attendre(Attendus = Ø)
Etat = dedans

Procédure exécutée lors d'un appel à libérer

Etat := dehors ;
$\forall j \in Diff$érés: Envoyer permission(i) à j ;
Différés := Ø ;

Procédure exécutée lors de la réception de permission(j)

Attendus := Atendus – {j} ;

Procédure exécutée lors de la réception de requête(K,j)

H := max(H,K) ;
Priorité := (Etat = dedans) ou ((Etat = demandeur) et (lasth,i)<(K,j))
Si priorité **Alors**
    Différés := Différés U {j} ;
Sinon
    Envoyer permission(i) à j ;
Finsi

2) Quel est l'écart maximal séparant les valeurs des deux horloges (|hi-hj|)

a. Dans le cas de deux processus ? (1 pt)
Remarquons que à l'état initial les horloges sont de valeurs égales, l'écart se creuse lorsque l'un des processus devient demandeur et se rétréci lorsque l'autre processus reçoit la requête. De ce fait l'écart maximal entre les horloges est de 1.

b. Dans le cas de N processus ? (1 pt)
Dans le cas de N processus, l'écart maximal est de N-1.

3) Modifiez les procédures précédentes afin de proposer une solution dans laquelle les horloges sont bornées. (4 pts)

Etant donné que l'écart est de N-1, nous pouvons travailler modulo 2*N -1 et appliquer la formule suivante dans l'évaluation de la priorité entre deux processus Pi et Pj demandeurs d'accès à la section critique :

Si | lasthi – lasthj | < N **alors** le processus de plus petite estampille est le plus ancien

Sinon le processus de plus grande estampille est le plus ancien.

Les modifications dans les procédures sont comme suit :

Procédure exécutée lors d'un appel à acquérir

```
Etat := demandeur ;
H := (H +1) modulo 2*N -1 ;
Lasth := H ;
Attendus := R
∀ j ∈ R: Envoyer requête(lasth,i) à j ;
Attendre(Attendus = Ø )
Etat = dedans
```

Procédure exécutée lors de la réception de requête(K,j)

```
H := max(H,K) ;
Si Etat = dedans Alors
        Priorité := true
Sinon
        Si Etat = demandeur alors
                Si (|lasth – K| < N)  alors
                        Si (lasth,i)<(K,j) Alors Priorité := True
                        Sinon Priorité := False Finsi
                Sinon
                        Si (lasth,i)<(K,j) Alors Priorité := False
                        Sinon Priorité := True Finsi
                Finsi
        Sinon Priorité := False Finsi
Finsi ;
Si Priorité Alors
        Différés := Différés U {j} ;
Sinon
        Envoyer permission(i) à j ;
Finsi
```

<u>Exercice 2.</u> **(10 points)**

Q1.	Quelle hypothèse faut-il faire sur le comportement des canaux de communication ? **Chaque canal de communication doit avoir un comportement FIFO**	(2,5pts)
Q2.	Quelle hypothèse faut-il faire sur la topologie du réseau de communication ? **Topologie fortement connexe** **fortement connexe : graphe orienté, où il existe un chemin entre chaque paire de sites, en respectant le sens des arcs.**	(2,5pts)
Q3.	(table below)	(5pts)

X_1	X_2	X_3	C_{12}	C_{13}	C_{21}	C_{23}	C_{31}	C_{32}
2	8	18	Φ	2	Φ	8	Φ	Φ

CONCOURS D'ACCES AU DOCTORAT EN INFORMATIQUE

Option : Systèmes Informatiques

Epreuve : Algorithmiques Distribués (Sujet 3)

07 Novembre 2019 / Durée : 02h

DIRECTIVES PEDAGOGIQUES:

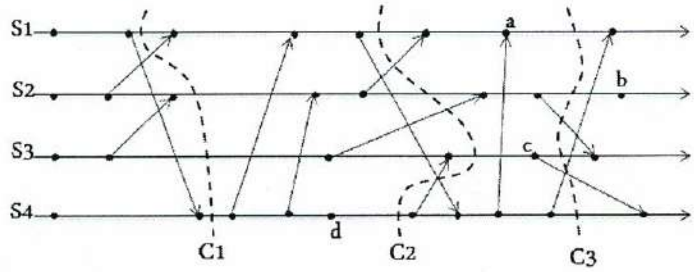
– *Documentation, Smartphones et Calculatrices ne sont pas autorisées*

– *Il est formellement interdit de remplir la rubrique numéro d'anonymat.*

– *Il est strictement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu*

Exercice 1. (8 points)

On considère un système distribué asynchrone constitué de 4 sites S1, S2, S3 et S4, s'envoyant des messages comme représenté par le diagramme de causalité suivant :



En considérant les horloges de *Lamport* initialisées à 0 dont la gestion se fait comme suit :

- Pour tout événement local l'horloge est incrémentée de 1.
- Pour tout événement de réception de message l'horloge prend le max de sa valeur et de l'estampille du message reçu.

Q1. Donnez les valeurs des horloges de *Lamport* correspondantes aux événements **a, b, c** et **d** ? (1 pt)

Q2. Donnez les valeurs des horloges de *Mattern* correspondantes aux événements : **a, b, c** et **d** ? (1 pt)

Q3. Calculer les estampilles de *Mattern* de C1, C2 et C3 (3 pts)

Q4. En se basant sur les valeurs des estampilles de *Mattern*, les coupures C1, C2 et C3 sont-elles cohérentes (Forment-elles des tranches) ? Justifiez votre réponse. (3 pts)

Exercice 2. (12 points)

Le problème de l'exclusion mutuelle constitue l'un des problèmes les plus importants aussi bien dans les systèmes centralisés que dans les systèmes distribués. Parmi les solutions proposées pour résoudre un tel problème dans un système centralisé, on distingue les sémaphores.

Q1. Peut-on utiliser les sémaphores dans un système distribué ? Justifiez. (1 pt)

Parmi les solutions proposées, pour résoudre le problème de l'exclusion mutuelle dans un système distribué, nous citons l'algorithme de *Le Lann* (1977) (à base de jeton). On Suppose que le temps des sections critiques est fini et les communications entre sites sont toutes fiables.

En utilisant l'algorithme de *Le Lann* (1977) :

Q2. Comment peut-on assurer la propriété de sûreté ? (1,5 pt)

Q3. Comment peut-on assurer la propriété de vivacité ? (1,5 pt)

Q4. En utilisant l'algorithme de *Lamport*, quel est le coût, en nombre de messages, pour l'entrée en Section Critique? Justifiez la réponse. (2 pt)

Q5. En utilisant l'algorithme de *ricart&agrawala (1981)*, Quel est le coût, en nombre de messages, pour l'entrée en Section Critique? Justifiez la réponse. (2 pt)

On considère un système distribué constitué de trois (3) sites : S_1, S_2 et S_3. On suppose que les délais de propagation des messages, entre les différents sites, sont connus. La table suivante décrit les valeurs de ces délais de propagation :

	S_1	S_2	S_3
S_1	**0**	1	1
S_2	2	**0**	2
S_3	3	3	**0**

A l'instant logique 0, les sites S1 et S3 envoient des demandes d'accès à leurs sections critiques.

Q6. En appliquant les deux algorithmes de *Lamport* et de *Ricart&Agrawala*, tracer les diagrammes décrivant les entrées aux sections critiques ainsi que les messages échangés entre les différents sites (préciser pour chaque événement sa date logique). (4 pts)

CONCOURS D'ACCES AU DOCTORAT EN INFORMATIQUE

Option : Systèmes Informatiques

Epreuve : Algorithmiques Distribués (Sujet 3)

07 Novembre 2019 / Durée : 02h

DIRECTIVES PEDAGOGIQUES:

– Documentation, Smartphones et Calculatrices ne sont pas autorisées
– Il est formellement interdit de remplir la rubrique numéro d'anonymat.
– Il est strictement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu

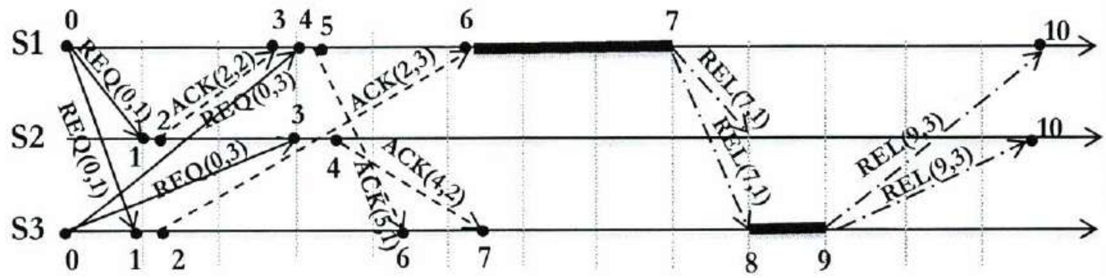
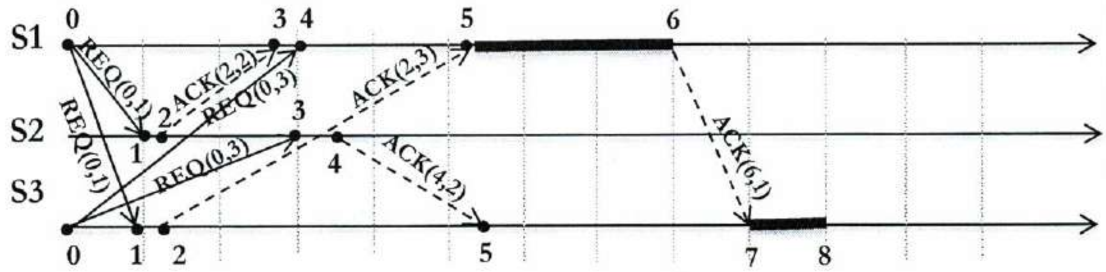
---

Exercice 1. (8 points)

Q1.	**Horloges de Lamport** VH_L (**a**) = 10 ; VH_L (**b**) = 10 ; VH_L (**c**) = 9 ; VH_L (**d**) = 6 ;	(1pt)
Q2.	**Horloges de Mattern** VH_M (**a**) = (5, 7, 2, 8) ; VH_M (**b**) = (2, 8, 3, 4) ; VH_M (**c**) = (2, 0, 5, 6) ; VH_M (**d**) = (2, 0, 0, 5) ;	(1pt)
Q3.	**Estampilles de *Mattern* des coupure C1, C2, et C3** Nous avons EV(C) = < EV(e1) [1], …, EV(ej) [j] , … EV(en) [n] > **EV(C1) = (2, 3, 2, 2) ; EV(C2) = (5, 5, 4, 5)** **EV(C3) = (7, 7, 5, 9)**	(3pts)
Q4.	***Vérification des coupures*** Une coupure C est dite cohérente **ssi** EV(C) = **sup** (EV(e_1), … EV(e_j),… EV(e_n)) **Coupure C1 :** **sup** (EV(e_1), … EV(e_j),… EV(e_n))= **sup** ((2, 0, 0, 2), (0, 3, 0, 0), (0, 2, 2, 0), (0, 0, 0, 2)) = (2, 3, 2, 2)= EV(C1) ⇒ **C1 Cohérente** **Coupure C2 :** **sup** (EV(e_1), … EV(e_j),… EV(e_n))= **sup** ((5, 2, 2, 2), (2, 5, 0, 0), (0, 2, 4, 0), (3, 4, 6, 5)) = (5, 5, 4, 6) ≠ EV(C2) ⇒ **C2 incohérente** **Coupure C3 :** **sup** (EV(e_1), … EV(e_j),… EV(e_n))= **sup** ((7, 2, 2, 5), (5, 7, 0, 2), (2, 3, 5, 0), (8, 4, 6, 9)) = (7, 7, 5, 9) = EV(C3) ⇒ **C3 Cohérente**	(3pts)

Exercice 2. (12 points)

Q1.	Peut-on utiliser les sémaphores dans un système distribué ? Justifiez **Non,** on ne peut pas utiliser les sémaphores dans un système distribué. **La raison** : on ne dispose pas de mémoire commune	(1pt)
Q2.	En utilisant *Le Lann(77)*, Comment peut-on assurer la propriété de sûreté ? La sûreté est assurée grâce au jeton unique	(1,5pt)
Q3.	En utilisant *Le Lann(77)*, Comment peut-on assurer la propriété de vivacité ? si un processus lâche le jeton en un temps fini et que tous les processus appartiennent à l'anneau reçoivent le jeton	(1,5pt)
Q4.	Coût de l'algorithme de *Lamport* = **3*(N-1) messages,** N : le nombre de site. **Justification** : **N-1** messages de type **Request**. **N-1** messages de type **ACK** **N-1** messages de type **Release**	(2pts)
Q5.	Coût de l'algorithme de *Ricart&Agrawala* = **2*(N-1) messages,** N : le nombre de site. **Justification :** **N-1** messages de type **Request**.	(2pts)

<table>
<tr><td></td><td>N-1 messages de type ACK
Les messages de type Release n'existent pas dans l'algorithme de Ricart&Agrawala.</td><td></td></tr>
<tr><td rowspan="2">Q6.</td><td>Lamport

</td><td>(2pt)</td></tr>
<tr><td>Ricart&Agrawala

</td><td>(2pt)</td></tr>
</table>

Université de Batna 2
Faculté Des Mathématiques et de l'Informatique
Département Informatique
CONCOURS D'ACCES AU DOCTORAT 3EME CYCLE EN INFORMATIQUE

Epreuve : Algorithmique et structure de données (Sujet n°02)

07 Novembre 2019 / Durée : 01h30

DIRECTIVES PEDAGOGIQUES:

– *Documentation non permise et il sera tenu compte de la clarté des copies.*
– *Calculatrice non autorisée*
– *Il est fortement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu*

Exercice 01 (5 pts)

Soit la fonction **f** suivante :

```
fonction f(X, B:Entier):Entier
var q, r : Entier
Début
        Si (X >=0 et B >= 2 et B <= 9) alors
                q ← X div 10
                r ← X mod 10
                si (q = 0) alors
                        f ← r
                sinon
                        f ← f(q, B) * B + r
                finsi
        sinon
                f ← -1
        finsi
fin
```

1) Donner la valeur de **f** pour X=120 et B=8.
2) En déduire ce que fait cette fonction.

Exercice 02 (6 pts)

1) Ecrire une fonction **Pile_inversée(p: Pile): Pile** renvoyant une copie inversée de 'p'.
2) Ecrire une fonction **Pile_copy(p: Pile): Pile** renvoyant une copie de 'p'. Attention, la pile 'p' doit être conservée.

Exercice 03 (9 pts)

Considérons l'algorithme suivant :

```
Type Matrice = Tableau [1..N, 1..N] d'entier ;
Var A, B, C : Matrice ;
    i, j, k : entier ;
Début
      Pour i ←1 à N Faire                              ┐
        Pour j ← 1 à i Faire                           │
            A[i, j] ← i – j + 1 ;                      │  Partie I
            B[i, j] ← i + j ;                          │
            C[i, j] ← 0 ;                              │
        Finpour                                        ┘
      Finpour

      Pour i ← 1 à N Faire                             ┐
        Pour k ← 1 à i Faire                           │
            Pour j ← 1 à k Faire                       │
              C [i, j] ← C[i, j]+A[i, k] * B[k, j] ;   │  Partie II
            Finpour                                    │
        Finpour                                        │
      Finpour                                          ┘
Fin
```

1) Donner le contenu de la matrice 'C' après exécution pour N = 4 ?
2) Quelle est la complexité exacte en espace mémoire de cet algorithme ?
3) Quelle est la complexité en espace mémoire de cet algorithme en utilisant la notation grand O ?
4) Quelle est la complexité exacte en nombre d'instructions arithmétiques de la **Partie I** de cet algorithme ?
5) Quelle est la complexité en nombre d'instructions de la **Partie II** de cet algorithme en utilisant la notation grand O ?
6) Quel est le nombre de cases mémoires inutilement allouées dans cet algorithme ?
7) Comment peut-on remédier à l'inconvénient cité dans la question précédente ?

Correction de l'épreuve : Algorithmique et structure de données (Sujet n°02)

Exercice 01 (5 pts)

Soit la fonction **f** suivante :

```
fonction f(X, B:Entier):Entier
var q, r : Entier
Début
        Si (X >=0 et B >= 2 et B <= 9) alors
                q ← X div 10
                r ← X mod 10
                si (q = 0) alors
                        f ← r
                sinon
                        f ← f(q, B) * B + r
                finsi
        sinon
                f ← -1
        finsi
fin
```

1) Donner la valeur de **f** pour X=120 et B=8.

Affichage après exécution : **Résultat: 80** (2 pts)

2) En déduire ce que fait cette fonction.

La fonction `f(int X, int B)` permet de convertir un nombre entier 'X' >= 0 écrit en base 'B' (B est un entier 2 ≤ B ≤ 9) en un nombre en base 10. (3 pts)

Exercice 02 (6 pts)

1) Ecrire une fonction **Pile_inversée(p: Pile): Pile** renvoyant une copie inversée de 'p'.

```
Fonction Pile_inversée(p: Pile): Pile
Début
        Tant que (p non vide) faire
                Pile_inversée.empiler(p.dépiler) ;
        FinTanque
Fin
```

2) Ecrire une fonction **Pile_copy(p: Pile): Pile** renvoyant une copie de 'p'. Attention, la pile 'p' doit être conservée.

```
Fonction Pile_copy (p: Pile): Pile
Début
        p1: Pile ;
        x: Entier ;
        p1 ← Pile_inversée(p) ;
        Tant que (p1 non vide) faire
                x ← p1.dépiler ;
                p.empiler(x) ;
                Pile_inversée.empiler(x) ;
        FinTanque
Fin
```

Exercice 03 (9 pts)

Considérons l'algorithme suivant :

```
Type Matrice = Tableau [1..N, 1..N] d'entier ;
Var A, B, C : Matrice ;
    i, j, k : entier ;
Début
      Pour i ←1 à N Faire
        Pour j ← 1 à i Faire
            A[i, j] ← i – j + 1 ;
            B[i, j] ← i + j ;                          Partie I
            C[i, j] ← 0 ;
        Finpour
      Finpour

      Pour i ← 1 à N Faire
        Pour k ← 1 à i Faire
            Pour j ← 1 à k Faire
               C [i, j] ← C[i, j]+A[i, k] * B[k, j] ;  Partie II
            Finpour
        Finpour
      Finpour
Fin
```

1) Donner le contenu de la matrice 'C' après exécution pour N = 4 ?

2
7 4
16 13 6
27 28 19 8

2) Quelle est la complexité exacte en espace mémoire de cet algorithme ?

La complexité exacte en espace mémoire : $\mathbf{3xN^2 + 3}$

3) Quelle est la complexité en espace mémoire de cet algorithme en utilisant la notation grand O ?

La complexité exacte en espace mémoire : $\mathbf{O(N^2)}$

4) Quelle est la complexité exacte en nombre d'instructions arithmétiques de la **Partie I** de cet algorithme ?

La complexité exacte en nombre d'instructions arithmétiques de la Partie I : **3xN(N+1)/2**

5) Quelle est la complexité en nombre d'instructions de la **Partie II** de cet algorithme en utilisant la notation grand O ?

La complexité en nombre d'instructions de la **Partie II** : **$O(N^3)$**

6) Quel est le nombre de cases mémoires inutilement allouées dans cet algorithme ?

Le nombre de cases mémoires inutilement allouées : **$3x(N^2 - N(N+1)/2)$**

7) Comment peut-on remédier à l'inconvénient cité dans la question précédente ?

En utilisant une structure de données qui ne stocke que les éléments du triangle inférieures des matrices.

Université de Batna 2
Faculté Des Mathématiques et de l'Informatique
Département Informatique
CONCOURS D'ACCES AU DOCTORAT 3EME CYCLE EN INFORMATIQUE

Epreuve : Algorithmique et structure de données (Sujet n°01)

07 Novembre 2019 / Durée : 01h30

DIRECTIVES PEDAGOGIQUES:

– Documentation non permise et il sera tenu compte de la clarté des copies.
– Calculatrice non autorisée
– Il est fortement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu

Exercice 01 (6 pts)

1) Ecrire une fonction récursive **fact(n)** qui retourne n!.
2) Ecrire une fonction itérative **puiss(x, n)** qui retourne x^n.
3) Ecrire une fonction **exp(x, n)** qui calcule la valeur approchée de e^x en faisant appel aux fonctions **fact** et **puiss**.

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Exercice 02 (8 pts)

Supposons qu'on a un polynôme P (x) définit comme suit :

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

tel que :

a_i sont des réels (positifs, négatifs ou nuls)
a_n est différent de zéro
n est appelé le degré du polynôme P(x).
i est le degré du $i^{ème}$ monôme.

Supposons aussi que les termes du polynôme (monômes) sont stockés dans une liste avec les puissances par ordre décroissant.

1. Donner les déclarations nécessaires de la structure de données pour représenter le polynôme P(x).
2. Donner une définition récursive puis écrire les algorithmes récursifs pour:
   a) Une fonction **Taille (L:List)** qui calcule le nombre d'éléments de la liste.
   b) Une procédure **MonomeExiste(L :List, n :Entier)** qui vérifie que le monôme de degré 'n' existe dans le polynôme représenté par la liste L.
   c) Une procédure **AjoutElemDebut(var L :List ; E :List)** qui permet d'insérer un élément 'E' au début de la liste 'L'.
   d) Une procédure **PolyAddition(var L1:List; L2:List)** qui calcule la somme de deux polynômes représentés par les listes L1 et L2. Le résultat de la somme sera stocké dans L1.

Exercice 03 (6 pts)

Soit 'A' un arbre binaire de recherche d'entiers.

1) Donner les deux propriétés principales d'un arbre binaire de recherche.
2) Ecrire une déclaration d'un type ABR de la structure A.
3) Écrire une procédure récursive **affiche_arbre(A:ABR)** qui affiche les valeurs des nœuds de 'A' par ordre croissant.
4) Donner une définition récursive de l'opération de recherche d'une valeur 'v' dans un arbre binaire de recherche, puis écrire la fonction récursive **rechercheVal(A:ABR, v:Entier): Booléen**.

Correction de l'épreuve : Algorithmique et structure de données (Sujet n°01)

Exercice 01 (6 pts)

1) Ecrire une fonction récursive **fact(n)** qui retourne n!.
2) Ecrire une fonction itérative **puiss(x, n)** qui retourne x^n.
3) Ecrire une fonction **exp(x, n)** qui calcule la valeur approchée de e^x en faisant appel aux fonctions **fact** et **puiss**.

$$e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

1) Factorielle

```
Fonction fact(n : entier) : entier;
Debut
 si n <= 1 alors
   fact ← 1
 sinon
   fact ← n * fact(n-1);
Fin
```

2) Puissance

```
Fonction puiss (x : reel; n : entier) : reel;
VAR i : entier;
       r : reel;
Debut
     r := 1.0; { init pour x^0 }
    pour i allant de 1 à n faire r ← r * x; { r = x^i }
    puiss := r;
Fin
```

3) Exponentielle

```
Fontion exp (x : reel; n : entier) : reel;
VAR i : entier;
       r : reel;
Debut
    r := 1.0; { init }
    pour i allant de 1 à n faire r ← r + puiss(x,i) / facto(i);
    exp := r;
Fin
```

Exercice 02 (8 pts)

Supposons qu'on a un polynôme P (x) définit comme suit :

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

tel que :

a_i sont des réels (positifs, négatifs ou nuls)
a_n est différent de zéro
n est appelé le degré du polynôme P(x).

i est le degré du $i^{ème}$ monôme.

Supposons aussi que les termes du polynôme (monômes) sont stockés dans une liste avec les puissances par ordre décroissant.

1. Donner les déclarations nécessaires de la structure de données pour représenter le polynôme P(x).

```
List = ↑terme
type terme =  Enregistrement
              coeff: integer;
              puiss: natural
              next: ↑ List
              Fin;
```

2. Donner une définition récursive puis écrire les algorithmes récursifs pour :

   a) Une fonction **Taille (L:List)** qui calcule le nombre d'éléments de la liste.

Définition récursive :

- Une liste vide de taille 0 ;
- La taille d'une liste non vide égale 1 + la taille de la même liste sans l'entête.

```
Fonction Taille(L:List):Entier;
Début
        si (L = nil) Alors Taille ← 0
        sinon Taille ← Size(L↑.next) + 1 ;
Fin;
```

   b) Une procédure **MonomeExiste(L :List, n :Entier)** qui vérifie que le monôme de degré 'n' existe dans le polynôme représenté par la liste L.

```
MonomeExiste (m:List, int n:Entier):booleén
Début
                Si (m = nil) alors MonomeExiste ← false;
                sinon
                        Si (m↑.puiss = n) alors
                                MonomeExiste ← true;
                        Sinon
                                MonomeExiste ← MonomeExiste(m↑.next, n);
                        FinSi
                FinSi
Fin
```

   c) Une procédure **AjoutElemDebut(var L :List ; E :List)** qui permet d'insérer un élément 'E' au début de la liste 'L'.

```
procédure AjoutElemDebut(var L :List ; E :List)
        Si (L↑.first = nil) alors
```

```
            E↑.next ← nil
            L↑.first ← E;
      sinon
            E↑.next ←L↑.first;
            L↑.first = E;
      FinSi
Fin
```

d) Une procédure **PolyAddition(var L1:List; L2:List)** qui calcule la somme de deux polynômes représentés par les listes L1 et L2. Le résultat de la somme sera stocké dans L1.

Définition récursive :

- Si la liste L1 est vide alors copier la liste L2 dans L1
- Si L2 est vide on ne fait rien et le résultat est la liste L1
- Si L1 et L2 non vides alors :
    - Si L1↑.Puiss > L2↑.Puiss alors appeler la même procédure pour la liste L1 sans l'entête et L2 tel quelle.
    - Si L1↑.Puiss = L2↑.Puiss alors le nouveau coefficient égale à la somme des deux coefficients et appeler la même procédure pour les deux listes sans les entêtes.
    - Si L1↑.Puiss < L2↑.Puiss alors créer un nouvel élément comme entête de la liste L1 et copier le contenu de l'entête de L2 dans L1 puis appeler la même procédure pour les deux listes sans les entêtes.

```
procédure PolyAddition (var L1:listptr; L2:listptr);
var temp : List
Début
      Si L1 = nil Alors CopyList(L1,L2)
      Sinon
            Si L2 <> nil alors
                  Si L1↑.Puiss > L2↑.Puiss alors PolyAddition(L1↑.next,L2)
                  Sinon
                        Si L1↑.Puiss = L2↑.Puiss alors
                              L1↑.Coeff ← L1↑.Coeff + L2↑.Coeff;
                              PolyAddition(L1↑.next,L2↑.next);
                              temp := l;
                              L1 ← L1↑.next;
                              dispose(temp)
                        Sinon
                              new (temp);
                              temp↑.Coeff ← L2↑.Coeff ;
                              temp↑.Puiss ← L2↑.Puiss ;
                              temp.next ← L1
                              L1 ← temp;
                              PolyAddition(L1↑.next,L2↑.next)
                        FinSi;
```

```
                    FinSi;
            FinSi;
    FinSi ;
Fin;

Procédure CopyList(var L1:List; L2:List);
Début
    Si L2 = nil alors L1 := nil
    Sinon
            new (L1);
            L1↑.Coeff ← L2↑.Coeff;
            L1↑.Puiss ← L2↑.Puiss;
            CopyList(L1↑.next,L2↑.next);
    FinSi;
Fin;
```

Exercice 03 (6 pts)

Soit 'A' un arbre binaire de recherche d'entiers.

1) Donner les deux propriétés principales d'un arbre binaire de recherche.

- Un nœud possède deux fils au plus.
- Fils gauche < père < fils droit

2) Ecrire une déclaration d'un type ABR de la structure A.

```
Type ABR = ↑node;
    node = record
            Gauche: treeptr;
            Val : Entier;
            Droit: treeptr
end
```

3) Écrire une procédure récursive **affiche_arbre(A:ABR)** qui affiche les valeurs des nœuds de 'A' par ordre croissant.

Il suffit d'appliquer le parcours infixé qui fournit la suite ordonnée des valeurs :

```
Procédure affiche_arbre ( A :ABR )
Début
    affiche_arbre ( A↑.Gauche) ;
    Ecrire (A↑.Val) ;
    affiche_arbre (A↑.Droit)
Fin ;
```

4) Donner une définition récursive de l'opération de recherche d'une valeur 'v' dans un arbre binaire de recherche, puis écrire la fonction récursive **rechercheVal(A:ABR, v:Entier): Booléen**.

Définition récursive

- Si la liste est vide retourner Faux ;
- Si la valeur de la racine égale à la valeur recherchée alors retourner Vari
- Sinon :
  - Si la valeur de la racine supérieure à la valeur recherchée alors chercher dans l'arbre enraciné par le fils gauche de la racine
  - Si la valeur de la racine inférieure à la valeur recherchée alors chercher dans l'arbre enraciné par le fils droit de la racine

```
Fonction rechercheVal(A:ABR, v:Entier): Booléen
Début
        si A = nil alors
                retourner Faux ;
        sinon
                si v = A↑.val(a) alors
                        retourner Vrai ;
                sinon
                        si v < A↑.val(a) alors
                                rechercheVal(A↑.Gauche, v)
                        sinon
                                rechercheVal(A↑.Droit, v)
                        finsi
                finsi
        finsi
Fin
```

Université de Batna 2
Faculté Des Mathématiques et de l'Informatique
Département Informatique

CONCOURS D'ACCES AU DOCTORAT 3EME CYCLE EN INFORMATIQUE

Epreuve : Algorithmique et structure de données (Sujet n°03)

07 Novembre 2019 / Durée : 01h30

DIRECTIVES PEDAGOGIQUES:

– Documentation non permise et il sera tenu compte de la clarté des copies.
– Calculatrice non autorisée
– Il est fortement interdit d'écrire avec des stylos en couleurs à l'exception du noir ou bleu

Exercice 01 (6 pts)

Soit le type suivant :

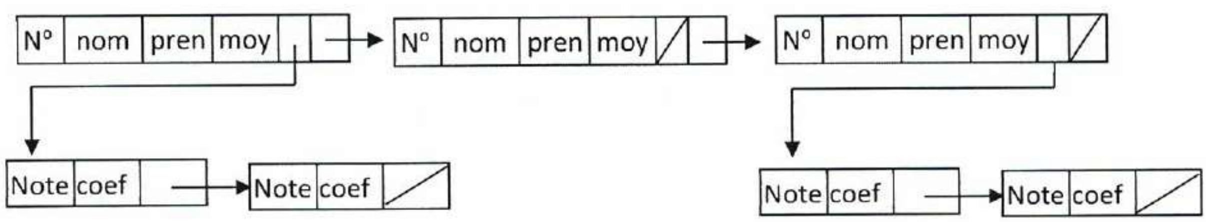
```
Type Produit = Enregistrement
                  Code : Entier ;
                  Désignation : Chaîne [ 80 ] ;
                  Prix : Réel ;
                  Fin ;
```

1) Soit 'F' un fichier de produits. Ecrire une fonction **verif** qui vérifie si les éléments de 'F' sont triés par ordre croissant de leur Code.
2) Calculer la complexité de la fonction **verif**. Justifier votre réponse.

Exercice 02 (7 pts)

Le département d'informatique souhaite gérer les notes de ses étudiants. Chaque étudiant a un numéro, un nom et un prénom. Ces informations sont stockées dans une liste chaînée dont chaque élément comporte aussi un champ 'moy' pour la moyenne de l'étudiant et un champ 'eval' qui est un pointeur sur sa liste des notes. Les notes de chaque étudiant sont aussi représentées par une liste chaînée dont la tête est le champ 'eval'. Le pointeur 'eval' est égal à Nil dans le cas où l'étudiant n'a pas encore de notes.

La figure ci-dessous représente la structure de données utilisée :



1) Donner la déclaration de cette structure de données.
2) Ecrire une procédure **moyennesEtudiants** qui calcule la moyenne de chaque étudiant et met à jour son champ 'moy'.
3) Calculer la complexité de la procédure **moyennesEtudiants**.

Exercice 03 (7 pts)

Les égyptiens de l'antiquité ne savaient pas directement multiplier deux nombres naturels, pour cela ils utilisaient les 4 opérations suivantes :

– additionner deux entiers strictement positifs,
– soustraire 1 à un entier strictement positif,
– multiplier par 1 et 2 tout entier strictement positif,
– diviser par 2 un entier strictement positif pair.

Voici un exemple qui multiplie 14 par 13 en utilisant uniquement ces opérations :

$$\begin{aligned} 14 \times 13 &= 14 + 14 \times (13 - 1) = 14 + 14 \times 12 \\ &= 14 + (14 \times 2) \times (12 / 2) = 14 + 28 \times 6 \\ &= 14 + (28 \times 2) \times (6 / 2) = 14 + 56 \times 3 \\ &= 14 + 56 + 56 \times (3 - 1) = 70 + 56 \times 2 \\ &= 70 + (56 \times 2) \times (2 / 2) = 70 + 112 \times 1 \\ &= 70 + 112 = 182 \end{aligned}$$

Nous allons écrire l'algorithme récursif qui permet la multiplication de 2 nombres naturels suivant cette méthode :

1) Donner la définition récursive (déterminer le ou les cas terminaux et l'expression récurrente).
2) Donner le corps de la fonction suivante en utilisant un algorithme récursif :
   fonction multiplicationEgyptienne (a,b : Naturel) : Naturel

Correction de l'épreuve : Algorithmique et structure de données (Sujet n°03)

Exercice 01 (6 pts)

Soit le type suivant :

Type Produit = Enregistrement
Code : Entier ;
Désignation : Chaîne [80] ;
Prix : Réel ;
Fin ;

1) Soit 'F' un fichier de produits. Ecrire une fonction **verif** qui vérifie si les éléments de 'F' sont triés par ordre croissant de leur Code. **3.5 Pts**

Fonction Verif (F : Fichier de Produit): Booléen ; 0.25 Pts
Var Eprod : Produit ; 0.25 Pts
Code : Entier ; 0.25 Pts
Début
Assigner (F,'Produit.dat') ; 0.25 Pts
Ouvrir (F) ; // ouvrir en lecture 0.25 Pts
Verif ←Vrai ; 0.25 Pts

Si (Non FDF(F)) **Alors 0.25 Pts**
Lire (F, Eprod) ; 0.25 Pts
Tantque (non FDF(F) et Verif) **Faire 0.25 Pts**
code ←Eprod.code ; 0.25 Pts
Lire(F, Eprod) ; 0.25 Pts
Si (code > Eprod.code) **Alors 0.25 Pts**
Verif ← Faux 0.25 Pts
Finsi ;
FinTanque ;
Fsi ;
Fermer (F). 0.25 Pts
Fin ;

2) Calculer la complexité de la fonction **verif.** Justifier votre réponse.
2.5 Pts (1 Pts pour le résultat et 1.5 pts pour la justification)

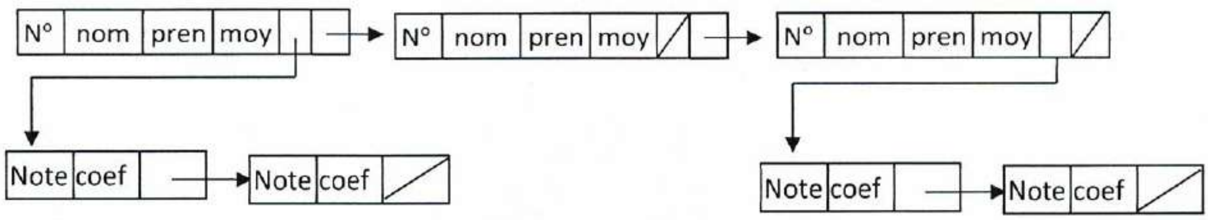
- La complexité est égale à **8+7n = O(n)**

Fonction Verif (F : Fichier de Produit): Booléen ; O(1)
Assigner (F,'Produit.dat') ; O(1)
Ouvrir (F) ; O(1)
Verif ←Vrai ; O(1) — O(7)
Si (Non FDF(F)) **Alors** O(1)
Lire (F, Eprod) ; O(2)
Tantque (non FDF(F) et Verif) **Faire** O(2)
code ←Eprod.code ; O(1)
Lire(F, Eprod) ; O(2)
Si (code > Eprod.code) **Alors** O(1) — O(7*n)
Verif ← Faux O(1)
Finsi ;
FinTanque ;
Fsi ;
Fermer (F). O(1)
Fin ;

Exercice 02 (7 pts)

Le département d'informatique souhaite gérer les notes de ses étudiants. Chaque étudiant a un numéro, un nom et un prénom. Ces informations sont stockées dans une liste chaînée dont chaque élément comporte aussi un champ 'moy' pour la moyenne de l'étudiant et un champ 'eval' qui est un pointeur sur sa liste des notes. Les notes de chaque étudiant sont aussi représentées par une liste chaînée dont la tête est le champ 'eval'. Le pointeur 'eval' est égal à Nil dans le cas où l'étudiant n'a pas encore de notes.

La figure ci-dessous représente la structure de données utilisée :



1) Donner la déclaration de cette structure de données. (1 pt)

Pe = ↑Etudiant
Pn = ↑Note
Etudiant = Structure
numero : Chaine de 10 caractères
nom : Chaine de 20 caractères
prenom : Chaine de 20 caractères
moy : réel
eval : Pn
suivant : Pe
fin Structure

Note = Structure
note : réel
coeff : Entier
suivant : Pn
fin Structure

2) Ecrire une procédure **moyennesEtudiants** qui calcule la moyenne de chaque étudiant et met à jour son champ 'moy'. (4 pts)

```
procédure moyennesEtudiants( etu : Pe)
Variables
        totCoeff : entier
        totNotes : réel
        petu : Pe /* pointeur de parcours de la liste des étudiants */
        pmat : Pn /* pointeur de parcours de la liste des notes de chaque ét.*/
Début
        petu ← etu
        tantque petu ≠ Nil        /* parcours de la liste des étudiants */ (2 pts)
                totCoeff ← 0    /* au début il n'y a ni note ni coefficient */
                totNote ← 0     /* pour l'étudiant */
                pmat ← petu↑.eval /* eval est la tête de liste des notes de l'étudiant. */
                tantque pmat ≠ Nil /* parcours de la liste des notes de l'étudiant */ (1 pt)
                        totCoeff ← totCoeff + pmat↑.coeff   /* somme des coefficients */
                        totNote ← totNote + pmat↑.note * pmat↑.coeff  /*somme
                                                                      /* pondérée */
                        pmat ← pmat↑.suivant              /* on passe à la note suivante */
                fintantque
                si petu↑.eval ≠ Nil alors /* calcul et mémorisation dans la cellule de */ (1 pt)
                                          /* l'étudiant de la moyenne de ses notes, s'il en a */
                        petu↑.moy ← totNote / totCoeff
                finsi
                petu ← petu↑.suivant              /* on passe à l'étudiant suivant */
        fintantque
```

3) Calculer la complexité de la procédure **moyennesEtudiants**. (2 pts)

Comp = O(1) + n(O(1) + O(1) + O(1) + O(1) + m(O(1) + O(2) + O(2) + O(1)) + O(1) + O(1))
= O(1) + n(O(4) + m(O(6)) + O(2))
= O(1) + n(O(4) + O(6m) + O(2))
= O(1) + n(O(6m + 6))
= O(6n*m + 6n + 1)
= O(n * m) tel que 'n' est le nombre d'étudiants et 'm' est le nombre de matières.

Calculs sur 1pt et résultat final sur 1 pt.

Exercice 03 (7 pts)

Les égyptiens de l'antiquité ne savaient pas directement multiplier deux nombres naturels, pour cela ils utilisaient les 4 opérations suivantes :

- additionner deux entiers strictement positifs,
- soustraire 1 à un entier strictement positif,
- multiplier par 1 et 2 tout entier strictement positif,
- diviser par 2 un entier strictement positif pair.

Voici un exemple qui multiplie 14 par 13 en utilisant uniquement ces opérations :

$$\begin{aligned} 14 \times 13 &= 14 + 14 \times (13 - 1) = 14 + 14 \times 12 \\ &= 14 + (14 \times 2) \times (12 / 2) = 14 + 28 \times 6 \\ &= 14 + (28 \times 2) \times (6 / 2) = 14 + 56 \times 3 \\ &= 14 + 56 + 56 \times (3 - 1) = 70 + 56 \times 2 \\ &= 70 + (56 \times 2) \times (2 / 2) = 70 + 112 \times 1 \\ &= 70 + 112 = 182 \end{aligned}$$

Nous allons écrire l'algorithme récursif qui permet la multiplication de 2 nombres naturels suivant cette méthode :

1) Donner la définition récursive (déterminer le ou les cas terminaux et l'expression récurrente).

- Cas terminal :
  - Si b = 1 alors a* b = a (1 pt)
- Cas général : si b > 1
  - Si b est pair alors a*b = 2*a*b/2 (1 pt)
  - Si b est impair alors a*b = a + a*(b-1) (1 pt)

2) Donner le corps de la fonction suivante en utilisant un algorithme récursif :

fonction multiplicationEgyptienne (a,b : Naturel) : Naturel

```
Fonction multiplicationEgyptienne (a, b : naturel) : naturel (0.5 pt)
Début
        Si b = 1 alors
                Retourner a (0.5 pt)
        Sinon
                Si b mod 2 = 0 alors
                        Retourner multiplicationEgyptienne (2*a, b div 2)  (1.5 pts)
                Sinon
                        Retourner multiplicationEgyptienne (a, b-1)        (1.5 pts)
                Finsi
        Finsi
Fin
```