

Epreuve - Matière : Composition - Informatique Session : 2024

CONSIGNES

- Remplir soigneusement, sur CHAQUE feuillet officiel, la zone d'identification en MAJUSCULES.
- Remplir soigneusement le cadre relatif au concours OU à l'examen qui vous concerne.
- Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.
- Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.
- N'effectuer aucun collage ou découpage de sujets ou de feuillet officiel.
- Numéroté chaque PAGE (cadre en bas à droite de la page) sur le nombre total de pages que comporte la copie (y compris les pages vierges).
- Placer les feuilles dans le bon sens et dans l'ordre de numérotation des pages.

Partie I. Robot de dépôt de solutions chimiques

1 Une rangée d'éprouvettes

Q1.1

Thread 1

Thread 2

① `int i = libre()`

② `int i = libre()`

③ `aller(i)`

④ `aller(i)`

⑤ `injecter(f)`

⑥ `injecter(f)`

S'il y a une éprouvette vide et que les opérations s'exécutent dans l'ordre ci-dessus, alors les deux threads vont choisir la même éprouvette.

En effet, lorsque le deuxième fait un appel à `libre()`, l'éprouvette choisie par le 1^{er} thread n'a pas encore été remplie, et est donc disponible.

Q1.2 Cette solution ne résout pas le problème montré en 1.1

En effet, lorsque le mutex est débloqué par un thread (ligne 6) l'éprouvette est toujours vide, donc un autre thread peut choisir à son tour la même éprouvette.

Q1.3 | Ici, le mutex n'est jamais débloquent car la ligne correspondante (ligne 9) est postérieure à la sortie de la fonction provoquée par le `return i;` en ligne 8.

Une correction possible est la suivante :

```

1 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
2
3 int déposer(struct formule *f) {
4     pthread_mutex_lock(&mutex);
5     int i = libre();
6     aller(i);
7     injecter(f);
8     pthread_mutex_unlock(&mutex);
9     return i;
10 }
```

Noter qu'on a juste inversé les lignes 8 et 9.

Cette fois il est impossible pour un autre thread d'appeler la fonction `libre` tant que l'éprouvette choisie n'est pas remplie.

Q1.4 | L'idée est d'encoder la formule avant de choisir l'éprouvette.

Cela nous donne le code suivant, où la seule fonction bloquante se trouve après l'encodage de la formule.


```

1 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
2
3 int déposer (struct formule * f) {
4     • struct codeage code = encoder (f, "bras modèle 1");
5     pthread_mutex_lock(&mutex);
6     int i = libre ();
7     aller (i);
8     • injecter (&code);
9     pthread_mutex_unlock(&mutex);
10    return i;
11 }

```

On a juste modifié l'injection, en encodant avant l'appel bloquant, et en modifiant l'appel à injecter.

1.1 Ajout du vldage des éprouvettes

Q1.5 Lorsque le thread connaît le temps de déplacement du bras, il faut qu'il puisse être sûr de pouvoir (ou d'avoir) prendre le mutex au moment opportun (à la seconde près), ce qui n'est pas possible.

~~Il peut réserver le mutex~~

Si l'autre thread, une minute environ avant l'analyse du premier, fait une injection ou une analyse (ce qui bloque le mutex pendant un temps de l'ordre de la minute) alors celui-ci ne pourra pas faire son analyse à temps.

Q1.6 S'il y a un seul thread, alors celui-ci peut déterminer via le calcul du temps de déplacement du bras le moment auquel il peut commencer le déplacement puis l'analyse. Par exemple, si ce temps est de 2 minutes, alors après 58 minutes (après l'injection), il peut commencer cette opération.

L'incertitude sur le temps est bien inférieure à une seconde. Donc il peut être précis à l'ordre de la seconde.

Par contre il ne peut pas être précis au temps d'un cycle processeur (ou nanoseconde) car il y a trop d'incertitude sur le coût d'une opération élémentaire même si il n'y a qu'un seul thread (par exemple un appel à une variable en mémoire).

Un tel appel a beaucoup d'incertitude en nombre de cycles nécessaires, notamment dû à l'optimisation par des caches.

Q1.7 | Cette fonction fonctionne.

lorsqu'un thread veut déposer une formule, il attend qu'une éprouvette soit libre.

À chaque fois qu'une analyse est faite, le mutex est débloquent et une éprouvette libérée. Donc un thread voulant déposer pourra obtenir une éprouvette.

Q1.8 | Un problème apparaît et que chaque thread va sans cesse faire des appels à librer lorsque toutes les éprouvettes sont pleines.

On pourrait utiliser une sémaphore pour bloquer ces threads jusqu'à ce qu'une éprouvette soit libre. Sans ça, les autres threads / processus peuvent être grandement ralentis.

Q1.9 | sem_t sem_libre;
sem_init(&sem_libre, 0, N);
sem_t sem_bras;
sem_init(&sem_bras, 0, 1);
int * libre = malloc(N * sizeof(int));
for (int i = 0; i < N; i++) {
 libre[i] = 1
}

On suppose tous ces appels déjà effectués.

libre est un tableau permettant de savoir quelles éprouvettes sont libres.

sem_bras joue le rôle du mutex précédent et
sem_libre permet de savoir si il y a une éprouvette libre.

Concours section : AGRÉGATION EXTERNE INFORMATIQUE
Epreuve matière : Composition d'informatique
N° Anonymat : N240NAT1030219 Nombre de pages : 28

14.95 / 20

Epreuve - Matière : Composition - Informatique Session : 2024

CONSIGNES

- Remplir soigneusement, sur CHAQUE feuillet officiel, la zone d'identification en MAJUSCULES.
- Remplir soigneusement le cadre relatif au concours OU à l'examen qui vous concerne.
- Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.
- Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.
- N'effectuer aucun collage ou découpage de sujets ou de feuillet officiel.
- Numéroté chaque PAGE (cadre en bas à droite de la page) sur le nombre total de pages que comporte la copie (y compris les pages vierges).
- Placer les feuilles dans le bon sens et dans l'ordre de numérotation des pages.

On a alors :

```
int déposer (&f) {
```

```
    struct formule *f {  
        sem - wait (&sem - libre);  
        i = 0  
        while (!libre[i]) { i++; }
```

5 / 28

Q1.11)

Il suffit d'un seul mutex qui permet de ne pas empêcher deux threads distincts de déplacer le robot en même temps etc...

```
void déposer (struct formule *f) {
    int x, y;
    pthread_mutex_lock (&mutex);
    libe XY (&x, &y);
    aller x(x);
    aller y(y);
    injecter (f);
    pthread_mutex_unlock (&mutex);
}
```

Q1.12)

```
void déposer (struct formule *f) {
    int x, y;
    pthread_mutex_lock (&mutex);
    libe XY (&x, &y);
    pthread_t tx;
    pthread_t ty;
    pthread_create (&tx, NULL, allerx, &x);
    pthread_create (&ty, NULL, alley, &y);
    pthread_join (tx, NULL);
    pthread_join (ty, NULL);
    injecter (f);
    pthread_mutex_unlock (&mutex);
}
```

On crée 2 threads qui déplacent les bras selon x et y en même temps.

Epreuve - Matière : Composition - Informatique Session : 2024

CONSIGNES

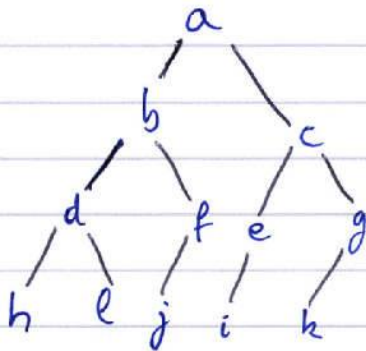
- Remplir soigneusement, sur CHAQUE feuillet officiel, la zone d'identification en MAJUSCULES.
- Remplir soigneusement le cadre relatif au concours OU à l'examen qui vous concerne.
- Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.
- Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.
- N'effectuer aucun collage ou découpage de sujets ou de feuillet officiel.
- Numéroté chaque PAGE (cadre en bas à droite de la page) sur le nombre total de pages que comporte la copie (y compris les pages vierges).
- Placer les feuilles dans le bon sens et dans l'ordre de numérotation des pages.

Partie II Tableaux flexibles

Q2.1)

a	b	c	d	e	f	g	h	i	j	k	l
---	---	---	---	---	---	---	---	---	---	---	---

Le tableau ci-dessus a comme représentation :



Q2.2)

```

let rec get (i:int) (t:tree) : elt =
  match t with
  | E -> raise Not_Found
  | N(t1,e,t2) -> if i = 0 then e else (
    if i mod 2 = 0 then get (i/2 - 1) (t2)
    else get (i/2) t1)
  
```

Noter que comme $0 \leq i < |t|$, on ne doit pas faire d'appel sur l'arbre E .

Q2.3 Montrons par induction sur la structure de tree que pour tout tableau flexible t , $h(t) \leq \lceil \log_2(|t|+1) \rceil$, si $t \neq E$.

• Si $t = E$, alors $h(t) = 0$, et $|t| = 0$,

• Si $t = E$ alors $h(t) = 0$, $|t| = 0$, et la propriété est vérifiée.

• Si $t = N(l, x, r)$ alors, on a $h(t) = 1 + h(l)$ car l'arbre gauche est plus grand que l'arbre droit. Ceci ne porte pas sur la hauteur de l mais le raisonnement suivant marche aussi sur r .

$$\begin{aligned} \text{Donc } h(t) = 1 + h(l) &\leq 1 + \lceil \log_2(1 + |l|) \rceil \\ &= \lceil 1 + \log_2(|l| + 1) \rceil \\ &= \lceil \log_2(2 + 2|l|) \rceil \end{aligned}$$

$$\text{Or } |t| = |l| + |r| + 1$$

$$\text{Donc } 2|l| + 2 \leq (|l| + |r| + 1) + 1 \text{ (cf (1))}$$

$$= |t| + 1$$

$$\text{Donc } h(t) \leq \lceil \log_2(2 + 2|l|) \rceil \leq \lceil \log_2(|t| + 1) \rceil$$

• Finalement cette propriété est vraie pour tout tableau flexible t .

Donc $h(t)$ est bien en $O(\log |t|)$

Q2.4 Hormis les appels récursifs, list ne fait que des opérations qui sont en $O(1)$

Si on, elle fait au pire cas 4 appel récursif à list avec $\frac{n}{2}$ à la place de n , et

On en déduit que le nombre d'opérations effectuées par l'ist n est, (on l'appelle $C(n)$), vérifie

$$C(n) \leq K + C\left(\frac{n}{2}\right) \quad \text{si } n \geq 2, \text{ car il n'y a pas d'appels récursifs si le tableau a un élément.}$$

où K est une constante.

On a d'ailleurs $C(1) \leq K$ (on peut choisir la même constante, quitte à ce qu'elle soit plus grande)

On montre alors par récurrence sur n que $\forall n \in \mathbb{N}^*, C(n) \leq (\lfloor \log_2(n) \rfloor + 1)K$

- Pour $n=1$, on a $C(1) \leq K$, donc la propriété est vérifiée.
- Supposons la propriété vérifiée jusqu'au rang $n-1$.
On a

$$\begin{aligned} C(n) &\leq K + C\left(\frac{n}{2}\right) \\ &\leq K + K (\lfloor \log_2\left(\frac{n}{2}\right) \rfloor + 1) \\ &= K (\lfloor \log_2\left(\frac{n}{2}\right) \rfloor + 2) \\ &= K (\lfloor \log_2(n) \rfloor + 1) \end{aligned}$$

Donc la propriété est vérifiée pour tout $n \in \mathbb{N}^*$

Finalement, $C(n) = O(\log(n))$ Donc l'ist n s'exécute en temps logarithmique.

Q2.5

① Pour la terminaison, si $n=1$, la fonction termine, car le tableau est de la forme $(E, -, E)$

Si non, l'ist n fait un appel à l'ist $\lfloor \frac{n}{2} \rfloor$. Or si $n > 1$, on a $\lfloor \frac{n}{2} \rfloor < n$. Comme $(\mathbb{N}^*, <)$ est un ordre bien fondé, on en déduit que la fonction termine forcément (i.e. la valeur de n dans les appels récursifs décroît strictement et finit par valoir 1)

On vérifie d'ailleurs que lorsque $n > 1$, $\lfloor \frac{n}{2} \rfloor > 0$

② Pour la correction partielle.

On vérifie facilement que $\text{list } n \uparrow$ renvoie le résultat attendu si l est un tableau de 1 élément.

• Sinon, soit $n \in \mathbb{N}^*$, un entier supérieur ou égal à 2. On suppose la fonction correcte pour des tableaux de 1 à $n-1$ éléments.

Cas 1 Si n est pair alors on ajoute un élément dans la case $n+1$, qui est impair. Donc cet élément se trouvera dans le fils gauche de la nouvelle représentation.

Donc le sous-arbre droit ainsi que la racine restent inchangés.

Le fils gauche sous arbre gauche est de taille $\lfloor \frac{n}{2} \rfloor = \frac{n}{2}$ (cela se montre facilement à partir de (4), selon la parité de n)

Et x doit bien

Cas 1 Si n est pair, alors on retire le n -ème élément, qui a pour indice $n-1$ dans le tableau, qui est impair.

Le premier élément et les éléments d'indices pairs ne changent pas.

Cela revient finalement à supprimer le dernier élément du tableau des éléments d'indices impairs.

Donc cela représente $N(\text{list } \frac{n}{2} \ell, x, r)$, on vérifie bien que $|\ell| = \lfloor \frac{n}{2} \rfloor$, ce qui se démontre facilement à partir de (1).

Cas 2 Le cas où n est impair est similaire.

Question 2.6 let rec snoc ($n:\text{int}$) ($t:\text{tree}$) ($x:\text{elt}$) : tree =
match t with

$| E \rightarrow N(E, x, E)$

$| N(\ell, a, r) \rightarrow \text{if } n \bmod 2 = 0 \text{ then}$

$N(\ell, a, \text{snoc } (n/2) \ r \ x)$

else $N(\text{snoc } (n/2) \ \ell \ x, a, r)$

Epreuve - Matière : Composition - Informatique Session : 2024

CONSIGNES

- Remplir soigneusement, sur CHAQUE feuillet officiel, la zone d'identification en MAJUSCULES.
- Remplir soigneusement le cadre relatif au concours OU à l'examen qui vous concerne.
- Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.
- Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.
- N'effectuer aucun collage ou découpage de sujets ou de feuillet officiel.
- Numérotter chaque PAGE (cadre en bas à droite de la page) sur le nombre total de pages que comporte la copie (y compris les pages vierges).
- Placer les feuilles dans le bon sens et dans l'ordre de numérotation des pages.

Q2-7

~~let rec tail (t: tree): tree =~~

~~match t with~~

~~| E → assert false~~

~~| N(E, -, E) → E~~

~~| N(l, x, r) → if x mod 2 = 0 then~~

let rec tail (t: tree): tree =

match t with

| E → assert false

| N(E, -, E) → E

| N(E, -, -) → assert false

| N(N(l, b, l r), a, r) → N(r, b, tail N(l, b, l r))

l'idée est la suivante, on pose le sous arbre droit à gauche, la racine du sous arbre gauche en racine, et à droite, le résultat de tail sur le sous arbre gauche.

Pour la complexité, on applique le même raisonnement qu'en Q2.4.

On a que des opérations en $O(1)$ ainsi qu'un appel récursif à un arbre de taille $\lfloor \frac{n}{2} \rfloor$. Donc le nombre d'opérations en fonction de la taille de l'arbre $T(n)$ vérifie:

- $T(1) \leq K$
- $\forall n \geq 2, T(n) \leq K + T(\frac{n}{2})$

On montre alors comme en Q2.4 que $\forall n \in \mathbb{N}^*, T(n) \leq K \lfloor \log_2(n) + 1 \rfloor$
et donc que $T(n) = O(\log_2(n))$

Pour la correction:

- On considère $\text{tail } t$ (où $\text{tail } t$ est l'arbre représentant t privé de sa racine)
- il est évident que le sous arbre gauche de $\text{tail } t$ est le sous arbre droit de t , car tous les éléments se décalent à gauche.
- De même, la racine du sous arbre gauche de t est la racine de $\text{tail } t$.
- Soit $t = t_1 \dots t_n$, on a $\text{tail } t = t_2 \dots t_n$ où t_1, \dots, t_n sont les éléments du tableau t .
- Donc le sous arbre droit de $\text{tail } t$ est $t_4 t_6 \dots t_{\lfloor \frac{n-1}{2} \rfloor}$
ce qui est l'arbre $t_2 t_4 \dots t_{\lfloor \frac{n-1}{2} \rfloor}$ privé de sa racine,
i.e. $\text{tail } t$, où t est le sous arbre gauche de t .

Q2.8

let rec cons (x:elt) (r:tree) : tree =
match r with
| E → N(E, x, E)
| N(l, a, r) → N(cons a r, x, l)

Q2-91

```
let of -array ( tab : elt array ) : tree =  
  let n = Array.length tab in  
  let rec build -parse k m =  
    if k >= n then E else  
      N ( build -parse (k+m) (2*m), tab.(k), build -parse (k+2*m) (2*m) )  
  in  
    build -parse 0 1
```

L'idée est la suivante:

Pour construire l'arbre représentant

$t[k] \ t[k+m] \dots$, i.e. $(t[k+im])_{0 \leq i < m}$

On met $t[k]$ en racine,

le sous arbre gauche représente le sous-tableau d'indice impair de $(t[k+im])$, i.e. les $t[k+im]$ pour i prenant des valeurs impaires. C'est à dire les $t[(k+m)+2im]$ pour n'importe quelle valeur de i .

On applique le même raisonnement pour le sous arbre droit.

Q2-101

```
let rec make (n: int) (x: elt) : tree =  
  if n = 0 then E else begin  
    let child = make (n/2) x in  
    if n mod 2 = 1 then N (child, x, child)  
    else N (make (n/2) child x, x, child)
```

Remarque

L'idée est que comme il n'y a qu'un seul élément, si n est impair alors le fils gauche et droit sont les mêmes. Dans ce cas, appeler deux fois la fonction récursivement serait inutile. On peut utiliser le partage du langage Ocaml pour que de plus, l'espace ne soit occupé qu'une seule fois.

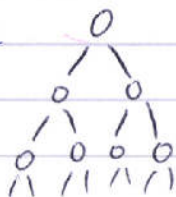
Si n est pair, on fait de même en ajoutant x au sous arbre gauche.

Pour la complexité, hormis l'appel récursif, la fonction ne fait que des opérations en $O(\log n)$ ou plus (moe)

On montre facilement qu'il y a au plus $\log n$ appels récursifs (même démonstration que Q2.4 et Q2.7)

Finalement, la complexité en temps de ^{make nx} ~~l'algo~~ est $O(\log(n)^2)$

Sous le moe, il résulte donc que la complexité en espace est logarithmique. i.e l'arbre



Je pense que c'est toujours le cas avec les appels d'moe, mais ne sache pas le prouver ici.

Epreuve - Matière : Composition - Informatique Session : 2024

CONSIGNES

- Remplir soigneusement, sur CHAQUE feuillet officiel, la zone d'identification en MAJUSCULES.
- Remplir soigneusement le cadre relatif au concours OU à l'examen qui vous concerne.
- Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.
- Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.
- N'effectuer aucun collage ou découpage de sujets ou de feuillet officiel.
- Numéroté chaque PAGE (cadre en bas à droite de la page) sur le nombre total de pages que comporte la copie (y compris les pages vierges).
- Placer les feuilles dans le bon sens et dans l'ordre de numérotation des pages.

Partie III Gestion d'un concours de recrutement

1 Base de données pour l'inscription

Q3.1 | On peut en effet supposer qu'un même candidat ne peut pas s'inscrire deux fois à l'agrégation externe d'informatique.

Q3.2 |

```
SELECT nom, COUNT(*)  
FROM AgrégExtInfo NATURAL JOIN Académie  
GROUP BY nom
```

Q3.3 |

```
SELECT C.nom, Ac1.nom, Ac2.nom  
FROM Candidat JOIN AgrégExtInfo JOIN Capes NSI  
FROM Candidat AS C JOIN AgrégExtInfo AS A JOIN Capes NSI AS N  
JOIN Académie AS Ac1 JOIN Académie AS Ac2  
ON A.cid = C.cid AND A.cid = N.cid AND  
A.cid = Ac1.cid AND N.cid = Ac2.cid  
WHERE Ac1.cid != Ac2.cid
```


En algèbre relationnelle, on obtient (je refais cette partie après la 3.6)

$$\{ (c.\text{nom}, ac1.\text{nom}, ac2.\text{nom}) \mid \exists ag, cap, \text{Candidat}(c) \wedge \text{AgrégExtInfo}(ag) \\ \wedge \text{CapesNSI}(cap) \wedge \text{Académiste}(ac1) \wedge \text{Académic}(ac2) \\ \wedge c.\text{cid} = ag.\text{cid} \wedge c.\text{cid} = cap.\text{cid} \wedge ac1.\text{aid} = ag.\text{aid} \wedge ac2.\text{aid} = cap.\text{aid} \\ \wedge (ac1.\text{aid} = ac2.\text{aid}) \}$$

Q 3.4

```
SELECT profession, COUNT(*)
FROM
  (SELECT * FROM candidat WHERE
    cid's
```

On fait d'abord une requête qui sélectionne les candidats qui posent l'agrégation

```
SELECT * FROM candidat LEFT JOIN AgrégExtInfo (Q1)
```

Ensuite pour le Capes,

```
SELECT * FROM candidat LEFT JOIN Capes NSI (Q2)
```

Finalement, la requête voulue est

```
SELECT profession, COUNT(*) FROM (Q1 EXCEPT Q2)
GROUP BY profession
```

Q3.5

Q3.6

(SELECT nom
FROM Candidat NATURAL JOIN Liste Amén NATURAL JOIN Aménagement
WHERE Amén = "Salle isolée")
EXCEPT

(SELECT nom
FROM Candidat NATURAL JOIN Liste Amén NATURAL JOIN Aménagement
WHERE Amén = "Tiers-temps")

en algèbre relationnelle, on obtient

$\Pi_{\text{nom}} (\sigma_{\text{Amén} = \text{"salle isolée"}} (\text{Candidat} \bowtie \text{Liste Amén} \bowtie \text{Aménagement}))$

$\setminus \Pi_{\text{nom}} (\sigma_{\text{Amén} = \text{"Salle Tiers-temps"}} (\text{Candidat} \bowtie \text{Liste Amén} \bowtie \text{Aménagement}))$

Q3.4 bis

En algèbre relationnelle, on obtient

$\Pi_{\text{C.nom}, \text{ac1.nom}, \text{ac2.nom}} (\sigma_{\text{ac1.aid} \neq \text{ac2.aid}} ((\rho_{\text{Ac1}} (\text{Agrég Ext Info} \bowtie \text{Académie})) \times$
 $\rho_{\text{Ac2}} (\text{Cupes NSI} \bowtie \text{Académie}))$
 $\bowtie \text{Candidat}))$

Q3-7 | On fait une requête qui détermine les académies ayant un candidat de l'agrégation ayant un tiers temps (on considère que des jointures naturelles)

SELECT Ac.nom FROM

Académie AS Ac JOIN Après EXInfo JOIN Candidat JOIN ListeAmén
JOIN Aménagement

WHERE amén = "Tiers-temps"

On note cette requête Q1, et Q2 la même requête pour le Copes.

On obtient alors la requête

(SELECT nom FROM Académie
EXCEPT Q1) EXCEPT Q2

Pour l'algèbre relationnelle, je m'autorise à renommer les tables
Am, LAm, Cd, Cp, Ag, Ac

On a

$(\pi_{\text{nom}}(\text{Ac}) \setminus \pi_{\text{Ac.nom}}(\sigma_{\text{amén}=\text{'Tiers-temps'}}(\text{Am} \bowtie \text{LAm} \bowtie \text{Cp} \bowtie \text{Ac} \bowtie \rho_{\text{nom} \rightarrow \text{nom}}(\text{Cd}))))$

$\setminus (\pi_{\text{Ac.nom}}(\sigma_{\text{amén}=\text{'Tiers-temps'}}(\text{Am} \bowtie \text{LAm} \bowtie \text{Ag} \bowtie \text{Ac} \bowtie \rho_{\text{nom} \rightarrow \text{nom}}(\text{Cd}))))$

Q3.8 |

Concours section : AGRÉGATION EXTERNE INFORMATIQUE
Epreuve matière : Composition d'informatique
N° Anonymat : N240NAT1030219 Nombre de pages : 28

14.95 / 20

Epreuve - Matière : Composition - Informatique Session : 2024

CONSIGNES

- Remplir soigneusement, sur CHAQUE feuillet officiel, la zone d'identification en MAJUSCULES.
- Remplir soigneusement le cadre relatif au concours OU à l'examen qui vous concerne.
- Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.
- Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.
- N'effectuer aucun collage ou découpage de sujets ou de feuillet officiel.
- Numéroté chaque PAGE (cadre en bas à droite de la page) sur le nombre total de pages que comporte la copie (y compris les pages vierges).
- Placer les feuilles dans le bon sens et dans l'ordre de numérotation des pages.

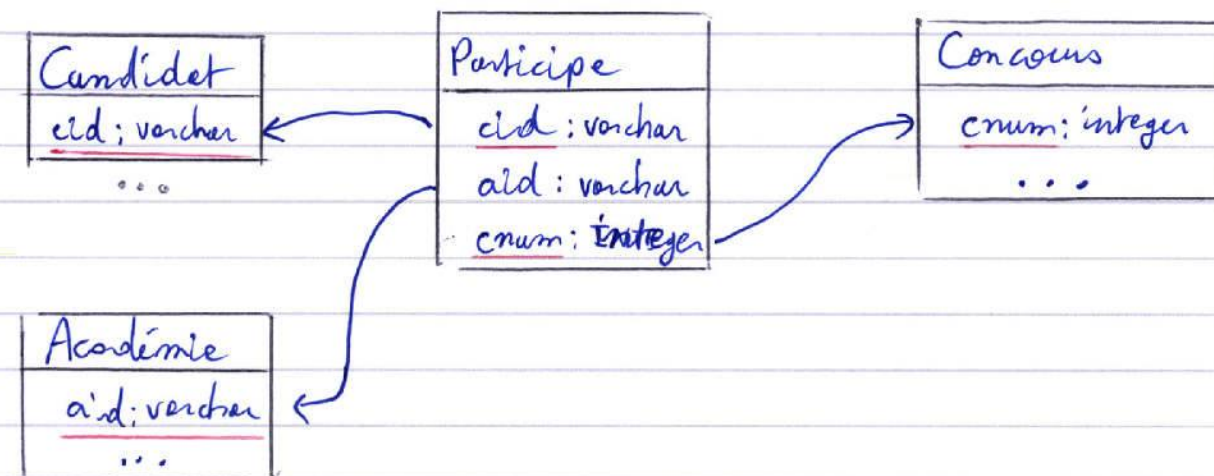
2 - Bases de données pour les notes des épreuves écrites

Q3.9 | SELECT nom
FROM Agrég Ext Infa NATURAL JOIN Candidat
ORDER BY (note-ep1 + note-ep2 + note-ep3) DESC
LIMIT 42

Q3.1d

21 / 28

Q3.11 | On crée une table participe qui indique quel candidat participe à quel concours, ainsi qu'une table dans quelle académie.
On peut alors créer une table concours, reliant l'identifiant d'un concours à son nom, et d'autres possibles attributs, comme cp par exemple.



les sources des flèches sont des clés étrangères.
les clés primaires sont soulignées en rouge.

Q3.12 |

3 - Gestion des épreuves orales

Q3.13 | def verif_nblec (N: int, t_oral):
 if len(t_oral) != 3*N:
 print("ce fichier comporte {} entrées au lieu de {}".format(len(t_oral), 3*N))
 return False
 for i in range(N):
 if len(t_oral[i]) != 10:
 print("la {}ème entrée de ce fichier comporte {} valeurs au lieu de 10".format(i, len(t_oral[i])))
 return False
 return True

Q3.14 |

```
def verif_temps(oral):  
    for i in range(len(oral)):  
        if oral[i][3] - oral[i][2] != 15:  
            print("le temps de préparation de l'entrée {} est de {} minutes  
                  au lieu de 15".format(i, oral[i][3] - oral[i][2]))  
            return False  
  
        if oral[i][5] - oral[i][4] != 60:  
            print("le temps d'interrogation de l'entrée {} est de {} minutes  
                  au lieu de 60".format(i, oral[i][5] - oral[i][4]))  
            return False  
  
        if oral[i][4] - oral[i][3] != durée_prép(i):  
            print("le temps de préparation de l'entrée {} pour l'épreuve  
                  de {} est de {} minutes au lieu de {}".format(i,  
                        oral[i][1], oral[i][4] - oral[i][3],  
                        durée_prép(i)))  
            return False  
  
    return True
```

durée-prép renvoie le temps d'épreuve de de préparation prévu pour la même entrée

```
def durée-prép(i):  
    épreuve = oral[i][1]  
    if épreuve == "TP":  
        return 300  
    elif épreuve in ["Légende", "Modélisation"]:  
        return 240  
    else:  
        print("pour l'entrée {i}, il n'existe pas d'épreuve de {}".format(i, épreuve))
```

Q3.15

```
def vérif-nbj(oral, nb-jury):  
    jury-par-h = [0] * (24 * 60)  
    n = len(oral)  
    for i in range(n):  
        for j in range(oral[i][4], oral[i][5]):  
            jury-par-h[j] += 1  
    for j in range(24 * 60):  
        if jury-par-h[j] > nb-jury:  
            print("{} jurys sont présents à {}h{} alors que il n'y a que {} jurys".format(jury-par-h[j], j//60, j%60, nb-jury))  
            return False  
    return True
```

Q3.16

Epreuve - Matière : Composition - Informatique Session : 2024

CONSIGNES

- Remplir soigneusement, sur CHAQUE feuillet officiel, la zone d'identification en MAJUSCULES.
- Remplir soigneusement le cadre relatif au concours OU à l'examen qui vous concerne.
- Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.
- Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.
- N'effectuer aucun collage ou découpage de sujets ou de feuillet officiel.
- Numéroté chaque PAGE (cadre en bas à droite de la page) sur le nombre total de pages que comporte la copie (y compris les pages vierges).
- Placer les feuilles dans le bon sens et dans l'ordre de numérotation des pages.

Q 3.16 / On a déjà vérifié en 3.14 qu'aucun autre nom d'épreuve n'était présent.

```
def verif_ep(oral):  
    N = len(oral) // 3  
    épreuves_cond = dict()  
    for i in range(3 * N):  
        cid = oral[i][7]  
        if not (cid in épreuves_cond.keys()):  
            épreuves_cond[cid] = [-1, -1, -1]  
    if len(épreuves_cond) != N:  
        print("il y a {} entrées mais {} candidats distincts".format(  
            3 * N, len(épreuves_cond)))  
    return False
```

def r_id: TP → 0 leçon → 1 modélisation → 2

```
...  
for i in range(3 * N):  
    cid = oral[i][7]  
    épreuve = oral[i][1]  
    if épreuves_cond[cid][r_id(épreuve)] != -1:  
        print("le candidat {} a 2 épreuves de {}".format(cid, épreuve))  
    return False
```

```

    | épreuves_cond[cld][to_id(épreuve)] = oral[i][0]

for k, v in épreuves_cond:
    if -1 in v:
        print("le candidat {} n'a pas 3 épreuves distinctes".format(k))
        return False
    if v[0] == v[1] or v[1] == v[2] or v[0] == v[2]:
        print("le candidat {} a 2 épreuves le même jour".format(k))
        return False

return True

```

03-17)

```

def dict_vers_nommo(dict_oral):
    nommo = dict()
    for entree in dict_oral:
        if entree['cid'] not in nommo:
            nommo['cid'] = (entree['Nom'], entree['Prénom'])
        else:
            if nommo['cid'] != (entree['Nom'], entree['Prénom']):
                print("le numéro {} correspond à deux noms, prénoms différents".format(cid))
                return False

    return True

```


Q3.18

Q3.19 • Un dictionnaire permet, dans le code, d'avoir des éléments plus lisibles
par exemple, `ord[i][0]` devient `entree[i]['Jour']`.
On a ainsi à poser par un nom de variable explicite.

• le dictionnaire a un temps d'accès moyen en $O(1)$, mais un tableau de tableau est au pire en $O(1)$, ce qui est meilleur.

Q3.20 • une base de données est très pratique pour des requêtes, mais l'est moins pour la vérification de propriété au sein des entrées.

• Un CSV va rendre plus lent les requêtes, mais on peut réintégrer les données dans une base de données une fois les données vérifiées.

