Correction

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours ICC aux sections MA et PH
Chappelier J.-C.

INFORMATIQUE, CALCUL & COMMUNICATIONS

Sections MA & PH

Correction de l'examen intermédiaire I

25 octobre 2019

SUJET 1

Instructions :

- Vous disposez d'une heure quinze minutes pour faire cet examen (15h15 - 16h30).
- L'examen est composé de 2 parties : un questionnaire à choix multiples, à 12 points, prévu sur 40–45 minutes, et une partie à questions ouvertes, à 12 points, prévue sur 30–35 minutes. Mais vous êtes libres de gérer votre temps comme bon vous semble.
- **AUCUN DOCUMENT N'EST AUTORISÉ, NI AUCUN MATÉRIEL ÉLECTRONIQUE**.
- Vous devez **écrire à l'encre noire ou bleu foncée**, pas au crayon, ni en une autre couleur.
- Pour la première partie (questions à choix multiples), chaque question n'a qu'une seule réponse correcte parmi les quatre propositions. Il n'y a pas de point négatif.
  Indiquez vos réponses en bas de **cette** page en écrivant *clairement* pour chaque question une lettre majuscule parmi A, B, C et D. (Vous êtes autorisés à dégrafer cette page)
  Aucune autre réponse ne sera considérée, et en cas de rature, ou de toute ambiguïté de réponse, nous compterons la réponse comme fausse.
- Pour la seconde partie, répondez directement sur la donnée, à la place libre prévue à cet effet. Aucune feuille supplémentaire ne sera considérée.
- Toutes les questions comptent pour la note finale.

Notations : dans cet examen, le premier élément d'une liste L est noté $L(1)$.

Réponses aux quiz :

Reportez ici *en majuscule* la lettre de la réponse choisie pour chaque question, sans aucune rature.

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	D	D	D	B	C	A	B	B

PARTIE QUIZ

Question 1) Soit L une liste de nombres entiers relatifs. Lequel des problèmes suivants est le plus simple algorithmiquement (nécessite le moins d'opérations pour être résolu) ?

✔**A]** Trier L.

B] Déterminer s'il existe deux éléments de L dont la somme est égale à 0.

C] Trouver la plus petite différence entre deux éléments de L.

D] Déterminer s'il existe un sous-ensemble d'éléments de L dont la somme est égale à 0.

Question 2) On considère la machine de Turing dont la table de transition est :

	0	1	ε
1	$(2, \varepsilon, +)$	$(4, \varepsilon, +)$	$(7, 1, -)$
2	$(2, 0, +)$	$(2, 1, +)$	$(3, \varepsilon, -)$
3	$(6, \varepsilon, -)$	$(7, 0, -)$	$(7, 1, -)$
4	$(4, 0, +)$	$(4, 1, +)$	$(5, \varepsilon, -)$
5	$(7, 0, -)$	$(6, \varepsilon, -)$	$(7, 1, -)$
6	$(6, 0, -)$	$(6, 1, -)$	$(1, \varepsilon, +)$

Quel est l'état de la bande lorsque la machine s'arrête, si elle a démarré dans l'état 1 avec sa tête de lecture positionnée comme suit :

$\cdots\varepsilon$	1	0	0	0	1	$\varepsilon\cdots$
	↑					

A] $\cdots\varepsilon\ 1\ 0\ 0\ 0\ \varepsilon\cdots$

✔**B]** $\cdots\varepsilon\ 1\ \varepsilon\cdots$

C] $\cdots\varepsilon\ 0\ 0\ 0\ \varepsilon\cdots$

D] $\cdots\varepsilon\ 0\ \varepsilon\cdots$

La bonne approche n'est pas ici de dérouler pas à pas une telle machine, mais bien de *comprendre* ce qu'elle fait : l'état 1 fait un choix (branchement conditionnel), les états (2,3) et (4,5) sont quasi identiques et vont à la fin pour tester le dernier bit, l'état 6 revient au début et recommence, ...

Question 3) Laquelle des représentations binaires suivantes sur 6 bits du nombre $e = 2.718...$ a la plus petite erreur ?

A] La représentation en virgule flottante avec 2 bits pour l'exposant et 4 bits pour la mantisse :

$$\widehat{x} = 1, m_1 m_2 m_3 m_4 \times 2^{e_1 e_2}$$

B] La représentation en virgule flottante avec 3 bits pour l'exposant et 3 bits pour la mantisse :

$$\widehat{x} = 1, m_1 m_2 m_3 \times 2^{e_1 e_2 e_3}$$

✔**C]** La représentation en virgule fixe avec 4 bits pour la partie fractionnaire et 2 bits pour la partie entière :

$$\widehat{x} = e_1 e_2, f_1 f_2 f_3 f_4$$

D] La représentation en virgule fixe avec 3 bits pour la partie fractionnaire et 3 bits pour la partie entière :

$$\widehat{x} = e_1 e_2 e_3, f_1 f_2 f_3$$

A noter tout d'abord que le nombre considéré étant fixé, la plus petite erreur absolue est aussi la plus petite erreur relative ; il n'y a donc pas lieu ici de qualifier l'erreur considérée.

Ensuite, la partie entière du nombre donné étant 2 (en décimal), elle sera de 10 en binaire, donc 2 bits sont nécessaires, ce qui élimine déjà D(1 bit inutile, moins de bits utiles).

Par ailleurs, B est clairement moins bon que A.

Finalement, sachant que la partie entière est 2 (donc e_1 de Cest non nul) et sachant qu'il n'y a qu'une seule réponse possible, c'est clairement celle qui a le plus de bit utiles (six contre cinq) qui est la bonne réponse.

(En toute rigueur, il incombe au concepteur du sujet de garantir que f_4 est non nul pour assurer n'avoir qu'une seule réponse correcte possible).

Question 4) On considère ici *uniquement* des schémas binaires sur 8 bits représentant des nombres entiers positifs. Avec

$$a = \texttt{00001011} \qquad \text{et} \qquad b = \texttt{00010101}\ ,$$

laquelle des opérations suivantes donne un résultat correct ?

A] $b \times b$ **B]** 2^a **C]** $a - b$ **✔D]** $a \times b$

Question 5) Vous organisez un tournoi d'échecs en 8 tours (chaque joueur jouera 8 parties). Vous souhaitez stocker dans un programme, pour chaque joueur, la liste de chacun de ses opposants pendant le tournoi, ainsi que savoir s'il a joué avec les blancs ou les noirs. Sachant que vous avez 100 participants en tout (et que leur liste est déjà stockée par ailleurs), combien de bits vous faut-il prévoir *au minimum par joueur* pour stocker ces informations ?

A] 808 **B]** 16 **C]** 57 **✔D]** 64

Cette question a été la moins bien réussie de tout le quiz et je ne comprends vraiment pas pourquoi ; pour moi, c'était une question facile...

On peut :

- soit considérer uniquement la situation du point de vue d'un joueur : on a besoin, par partie, de stocker sa couleur (1 bit) et son opposant (7 bits), soient 8 bits par partie ;
- soit considérer la situation globale : il y a $2^8 \times 100^8$ configurations possibles (on n'a pas dit que le tournoi était à élimination directe) et donc $\lceil \log_2(2^8 \times 100^8) \rceil = \lceil 8 + 16 \times \log_2(10) \rceil = \lceil 8 + 16 + 16 \times \log_2(5) \rceil$ qui est certainement sensisiblement plus grand que $8 + 16 + 32 = 56$, mais plus petit que $8 + 16 + 48 = 72$ (sa vraie valeur est 62).

Question 6) Considérer l'algorithme suivant, où $L(i : j)$ représente la sous-liste $(L(i), ..., L(j))$:

algo6

entrée : *L liste non vide de nombres réels*
sortie : *? ? ?*

```
t ⟵ taille(L)
Si t = 1
    Sortir : (L(1), L(1))
L' ⟵ algo6(L(2 : t))
L'' ⟵ algo6(L(1 : t − 1))
Si L(t) < L(1)
    Sortir : (L'(1), L''(2))
Sinon
    Sortir : (L''(1), L'(2))
```

Quelle est sa sortie pour l'entrée $L = (12, 24, 33, 55, -3, 43, 8, 0)$?

A] $(12, 0)$ **B]** $(0, 12)$ **C]** $(33, 33)$ **✔D]** $(-3, 55)$

Question 7) Si n est la taille de la liste L en entrée, comment se situe la complexité de l'algorithme précédent ?

A] en $\mathcal{O}(n)$, mais pas en $\mathcal{O}(\log(n))$.
B] en $\mathcal{O}(\log(n))$.
C] en $\mathcal{O}(n^2)$, mais pas en $\mathcal{O}(n)$.
✔**D]** en $\mathcal{O}(2^n)$, mais pas en $\mathcal{O}(n^2)$.

La bonne approche n'est pas ici de faire tourner l'algorithme sur une telle entrée (trop longue), mais bien de *comprendre* ce qu'il fait.

Question 8) Considérer l'algorithme suivant, où $L(i:j)$ représente la sous-liste $(L(i),...,L(j))$ si $i \leq j$, et la liste vide sinon :

```
algo8
entrée : liste L
sortie : ? ? ?
    t ⟵ taille(L)
    Si t < 2
        Sortir : 1
    Si L(1) = L(t)
        Sortir : algo8(L(2 : t − 1))
    Sinon
        Sortir : 0
```

algo8(7, 5, 12, 4, 8, 12, 5, 7) :

A] ne se termine pas. ✔**B]** sort 0. **C]** ne sort rien du tout. **D]** sort 1.

Question 9) Un algorithme a besoin de l'ordre de 2^n instructions élémentaires pour calculer une fonction $f(n)$ donnée. Sachant qu'un programme implémentant (uniquement) cet algorithme sur une machine donnée calcule $f(50)$ en 10 minutes, combien de minutes environ faudrait-il à ce même programme pour calculer $f(200)$ (en repartant du début) ?

Note : on pourra si nécessaire faire l'approximation $2^{10} \simeq 10^3$.

A] 10^{60} **B]** 10^4 ✔**C]** 10^{46} **D]** 40

10 minutes correspondent à $K \cdot 2^{50}$ instructions élémentaires, donc $K \cdot 2^{200}$ instructions élémentaires correspondent à $2^{200-50} \times 10 \simeq 10^{46}$ minutes

---

Question 10) Quelle est la sortie de l'algorithme suivant :

```
algo10
entrée : deux entiers a et b, strictement positifs
sortie : ? ? ?
    n ⟵ 1
    m ⟵ 1
    Tant que n < b
        n ⟵ n + 1
        m ⟵ m × a
    Sortir : n × m
```

✔**A]** $b \times a^{b-1}$ **B]** $(b-1) \times a^{b-1}$ **C]** $(b-1) \times a^b$ **D]** $b \times a^b$

Attention à l'inégalité stricte.

Question 11) Avec $a = b$ en entrée, comment se situe la complexité de l'algorithme précédent ?

A] en $\mathcal{O}(1)$.
✔**B]** en $\mathcal{O}(b^2)$, mais pas en $\mathcal{O}(\log(b))$.
C] en $\mathcal{O}(2^b)$, mais pas en $\mathcal{O}(b^2)$.
D] en $\mathcal{O}(\log(b))$, mais pas en $\mathcal{O}(1)$.

$\mathcal{O}(b)$ est inclus dans $\mathcal{O}(b^2)$...

Question 12) ⚠ Pour l'algorithme suivant :

algo12
entrée : *deux entiers positifs a et b*
sortie : *0 ou 1*

$b \longleftarrow a \times (a-1)/2$
$a \longleftarrow b \times (b+1)/2$
Si $a \leq 2b$
  Sortir : 1
Sinon
  Sortir : 0

laquelle des propositions suivantes est vraie ?

A] Pour l'entrée $a = 2$, $b = 2$, il sort 0.
✔**B]** Pour l'entrée $a = 1$, $b = 3$, il sort 1.
C] Pour l'entrée $a = 5$, $b = 3$, il sort 1.
D] Aucune des autres propositions.

Le signe ⚠ n'était pas là par hasard : l'algorithme est indépendant de la valeur de b en entrée (puisqu'elle est tout de suite écrasée).

---

suite au dos ☞

PARTIE EXERCICES

1 – Ecriture d'algorithmes [12 points]

On s'intéresse ici à quelques opérations sur les représentations binaires de nombres entiers positifs ou nuls. Pour cela, on représentera les nombres en question sous forme de listes de valeurs 0 ou 1 telles que $L(1)$ soit le bit de poids le plus faible du nombre représenté et $L(n)$, où n est la taille de la liste, soit toujours à 1, sauf s'il s'agit de la liste vide, laquelle représente le nombre 0. Par exemple, la liste représentant le nombre 22 (dont l'écriture binaire usuelle est `10110`) sera $(0, 1, 1, 0, 1)$.

Question 13) **[1.5 points]** Ecrivez un algorithme qui, prenant en entrée une telle liste, sort la valeur décimale correspondante. Par exemple, pour la liste $(0, 1, 1, 0, 1)$ en entrée, cet algorithme devra sortir 22. Pour la liste vide, il sortira 0.

Question 14) **[1 point]** Déterminez la complexité de l'algorithme que vous avez écrit à la question précédente (question 13). Justifiez votre réponse.

Question 15) **[3 points]** Ecrivez un algorithme *récursif, sans aucune boucle ni itération*, qui, prenant en entrée deux listes binaires, sort le nombre de bits en commun, c.-à-d. le nombre de bits ayant la même valeur à la même position dans les deux listes.

Par exemple, pour les listes $(0, 1, 1, 0, 1)$ et $(0, 0, 1, 1)$ en entrée, cet algorithme devra sortir 2 car les premier et troisième bits de ces deux listes sont égaux.

A noter que les deux listes en entrée n'ont pas nécessairement la même taille.

Question 16) **[1.5 point]** Déterminez la complexité de l'algorithme que vous avez écrit à la question précédente (question 15) appliqué à des listes de même taille (n). Justifiez votre réponse.

Question 17) **[5 points]** Ecrivez un algorithme qui, prenant en entrée deux listes binaires, sort la liste correspondant à l'addition des deux nombres représentés.

Par exemple, pour les listes $(0, 1, 1, 0, 1)$ (représentant 22) et $(0, 0, 1, 1)$ (représentant 12) en entrée, cet algorithme devra sortir la liste $(0, 1, 0, 0, 0, 1)$ (représentant 34).

A noter que les deux listes en entrée n'ont pas nécessairement la même taille et que la liste en sortie peut être plus grande que la plus grande des deux en entrée.

Réponses :

13) Version itérative

bin2dec_it

entrée : *liste L representant un nombre binaire*
sortie : *valeur décimale représentée par L*

$n \longleftarrow$ **taille**(L)
$v \longleftarrow 0$
Pour i allant de 1 à n
  $v \longleftarrow v + L(i) \times 2^{i-1}$
Sortir : v

Version récursive

bin2dec_rec

entrée : *liste L representant un nombre binaire*
sortie : *valeur décimale représentée par L*

$n \longleftarrow$ **taille**(L)
Si $n = 0$
  Sortir : 0
Sinon
  Sortir : $L(1) + 2 \times$ **bin2dec_rec**$(L(2 : n))$

où $L(2 : n)$ représente la sous-liste $(L(2), ..., L(n))$ si $n \geq 2$, et la liste vide sinon.

Quelques erreurs fréquentes à ne pas commettre :

- je ne sais pas pourquoi certains ont tenu à inverser/renverser la représentation de la liste... Cela ne change rien au calcul de la valeur décimale (de même que le sens de la boucle ne change rien non plus...)
- quelques confusions entre $L(i) \times 2^{i-1}$ et $L(i)^{i-1}$!!
- écrire « $x \longleftarrow x$ » ne sert à rien d'autre qu'à perdre du temps ; autant ne rien écrire du tout, pas même le test correspondant (cette instruction était dans un branchement conditionnel : supprimez simplement le test correspondant).

14) La complexité des algorithmes précédents est en $\mathcal{O}(n)$, où n est la taille de la liste (précisez vos notations !). On ne parcourt en effet qu'une seule fois la liste et la boucle de parcourt ne fait que des opérations élémentaires. Et il est plus que raisonnable de supposer que la complexité de **taille()** est dans $\mathcal{O}(n)$ pour la version itérative (ce qui suffit pour conclure dans ce cas là), voire dans $\mathcal{O}(1)$ pour la version récursive (il suffirait de modifier un peu l'algorithme pour la calculer une fois au départ, puis la passer comme paramètre à chaque fois, si nécessaire).

L'erreur la plus courante a ici été un manque de justification : ce n'est pas parce qu'on a une boucle que la complexité est en $\mathcal{O}(n)$: encore faut il que le contenu de la boucle soit en $\mathcal{O}(1)$, c.-à-d. ne contienne que des opérations élémentaires !

15)

intersection

entrée : *Deux listes L et L'*
sortie : *nombre de valeurs communes (aux mêmes positions)*

$n_1 \longleftarrow$ **taille**(L)
$n_2 \longleftarrow$ **taille**(L')
Si $n_1 = 0$ **ou** $n_2 = 0$
    Sortir : 0
Si $L(1) = L'(1)$
    Sortir : $1 +$ **intersection**$(L[2 : n_1], L'[2 : n_2])$
Sinon
    Sortir : intersection$(L[2 : n_1], L'[2 : n_2])$

Commentaires :

- Beaucoup trop d'erreurs sur la récursion : oubli de « **Sortir** », pas d'utilisation de la valeur retournée par la récusion, oubli de « l'accumulateur » (pas utilisé dans la solution ci-dessus, mais certaines solutions utilisaient une variable pour stocker le total), ...
- Toujours au niveau de la récursion, quelques uns sont totalement inconsistants : passent d'autres arguments (1 liste ou 1 nombre au lieu de 2 listes) ou utilisent la sortie à la fois comme un nombre et comme une liste ; restez cohérent(e)s dans vos écritures !
- Certain(e)s s'offrent la liberté de changer la donnée en ajoutant un argument (mais au moins ils/elles restent cohérent(e)s) ;
- Attention à ne pas écrire $L(i)$ pour des i non valables, p.ex. $L(1)$ pour une liste vide.
- Trop de confusion de notation : p.ex. $L(i)$ au lieu de L, $(L(2), L(n))$ (liste à deux éléments) au lieu de $(L(2), \cdots, L(n))$ ou $L(2 : n)$ (listes à $n - 1$ éléments), etc.

16) La complexité de l'algorithme ci-dessus est en $\mathcal{O}(n)$. On diminue en effet à chaque fois les listes d'un seul élément. Et, comme ci-dessus, il est plus que raisonnable de supposer que la complexité de **taille()** est dans $\mathcal{O}(1)$ (il suffirait de passer les tailles à chaque fois). De plus, tout le reste sont des opérations élémentaires.

17) Le plus simple est certainement d'utiliser l'addition en décimal (considérée comme une opération élémentaire dans ce cours) :

addition
entrée : *Deux listes L et L' représentant des nombres en binaire*
sortie : *liste représentant la valeur binaire de l'addition*
Sortir : dec2bin(**bin2dec**(L) + **bin2dec**(L'))

où, **bin2dec** est l'algorithme écrit en question 13, et **dec2bin** est l'algorithme de conversion de décimal en binaire (vu en cours du jeudi), p.ex. comme suit (à définir, donc) :

dec2bin
entrée : *v, nombre entier positif*
sortie : *liste representant v en nombre binaire*

Si $v > 0$
  Sortir : $v \mod 2 \oplus$ **dec2bin**($\lfloor v/2 \rfloor$)
Sinon
  Sortir : () // liste vide

où $\oplus$ représente l'ajout d'un élément en tête de liste : $x \oplus L = (x, L(1), ..., L(n))$ (avec n la taille de L), si L n'est pas vide, et $x \oplus () = (x)$.

Une autre idée peut être de faire l'addition de façons récursive, mais il faut alors prévoir la retenue comme argument supplémentaire. Cela nécessite donc encore deux algorithmes :

addition
entrée : *Deux listes L et L' représentant des nombres en binaire*
sortie : *liste représentant la valeur binaire de l'addition*
Sortir : addition_avec_retenue($L, L', 0$)

addition_avec_retenue

entrée : *Deux listes L et L' représentant des nombres en binaire, c un bit*
sortie : *liste représentant la valeur binaire de l'addition de L, L' et c*

```
n1 ⟵ taille(L)
Si n1 = 0
    Si c = 0
        Sortir : L'
    Sinon
        Sortir : addition_avec_retenue((1), L', 0)
n2 ⟵ taille(L')
Si n2 = 0
    Si c = 0
        Sortir : L
    Sinon
        Sortir : addition_avec_retenue((1), L, 0)
v ⟵ c + L(1) + L'(1)
Si v > 1
    c ⟵ 1
Sinon
    c ⟵ 0
Sortir : v mod 2 ⊕ addition_avec_retenue(L(2 : n1), L'(2 : n2), c)
```

Une dernière solution consiste à faire l'addition de façon itérative sur les deux listes :

addition

entrée : *Deux listes L et L' représentant des nombres en binaire*
sortie : *liste représentant la valeur binaire de l'addition*

$n_1 \longleftarrow$ **taille**(L)
$n_2 \longleftarrow$ **taille**(L')
$c \longleftarrow 0$
$R \longleftarrow ()$ // liste vide
Pour i allant de 1 à $\min(n_1, n_2)$
  $v \longleftarrow L(i) + L'(i) + c$
  $c \longleftarrow \lfloor v/2 \rfloor$
  $R \longleftarrow R \odot v \mod 2$
Si $n_1 > n_2$
  Pour i allant de $n_2 + 1$ à n_1
    $v \longleftarrow L(i) + c$
    $c \longleftarrow \lfloor v/2 \rfloor$
    $R \longleftarrow R \odot v \mod 2$
Sinon
  Pour i allant de $n_1 + 1$ à n_2
    $v \longleftarrow L'(i) + c$
    $c \longleftarrow \lfloor v/2 \rfloor$
    $R \longleftarrow R \odot v \mod 2$
Si $c = 1$
  $R \longleftarrow R \odot 1$

où $\odot$ représente l'ajout d'un élément en fin de liste : $L \odot x = (L(1), ..., L(n), x)$ (avec n la taille de L), si L n'est pas vide, et $() \odot x = (x)$.

On pourrait bien sûr modulariser en un sous-algorithme l'addition d'un bit, répétée trois fois ci-dessus :

addition_1_bit

entrée : *trois bits, a, b, c et une liste (binaire) L*
sortie : *un bit et une liste (binaire)*

$v \longleftarrow a + b + c$
Sortir : $(\lfloor v/2 \rfloor, L \odot v \mod 2)$

Quelques erreurs fréquentes à ne pas commettre :

- oubli de la retenue (en général)
- oubli de la dernière retenue (débordement d'un bit supplémentaire)
- oubli du cas $1 + 1 + 1 (=3)$
- écrire $L(i)$ pour les i non valables, p.ex. pour une liste vide ; typiquement la solution (on ne peut pas initialiser la solution, p.ex. R, à vide, puis écrire $L(i) \longleftarrow ...$)
- modifier dans la boucle les bornes d'une itération (boucle « **Pour tout** »), préférer alors une boucle conditionnelle (« **Tant que** ») ;
- (non pénalisé, mais faites attention quand même !) écrire des « **Si**» les uns derrière les autres sans les « **Sinon** » (algorithmiquement cela ne change souvent pas grand chose, surtout ici, mais pourrait parfois ; par contre en programmation cela effectue plein de tests souvent inutiles et est donc moins efficace)
- (non pénalisé, mais faites attention quand même !) des abus de notation, p.ex. $L \oplus x$ au lieu de $L \longleftarrow L \oplus x$.