Algorithmique IV (UE.41) - Licence d'Informatique

Correction du contrôle terminal (Session 1 / Mai 2026)

Rappels. Une liste L d'éléments d'un ensemble E définit une application $L : [\![1, |L|]\!] \to E$. Quand on s'y réfère, on note $L(i)$ plutôt que $L[i]$. Soit $f : X \to Y$, $A \subseteq X$ et $B \subseteq Y$. Alors $f(A) := \{f(x) \mid x \in A\}$ est l'*image directe* de A par f et $f^{-1}(B) := \{x \in X \mid f(x) \in B\}$ est l'*image réciproque* de B par f.

On note $\mathscr{L}_n$ l'ensemble des listes d'entiers naturels de longueur $n \geqslant 3$ dont toutes les valeurs sont identiques sauf une, l'*intrus* et on note $\pi : \mathscr{L}_n \to [\![1, n]\!]$ l'application qui calcule sa position.

Exemple : la liste $L = [4, 4, \mathbf{2}, 4, 4, 4] \in \mathscr{L}_6$ a pour intrus **2** et $\pi(L) = 3$.

Question 1. Vérifiez que la relation binaire $\equiv$ définie sur $\mathscr{L}_n$ par

$$(L \equiv M) \text{ si et seulement si } (\pi(L) = \pi(M)). \tag{1}$$

est une relation d'équivalence. Calculez $|\mathscr{L}_n/\equiv|$, le nombre de classes d'équivalence pour cette relation.

Réponse. Deux listes sont en relation pour $\equiv$ si leurs images par π sont égales, or l'égalité est à la fois réflexive, symétrique et transitive. Deux listes dont l'intrus est à la même positions étant équivalente, il y a exactement n classes d'éauivalences, une par position de l'intrus.

Question 2. Les deux fonctions *Python* suivantes calculent respectivement l'image directe d'une partie A de $[\![1, |L|]\!]$ et l'image réciproque d'une partie B de $\mathbb{N}$ par l'application `L` :

```
def Im(L,A):
    return {L[i] for i in A}
def ImR(L,B):
    return {i for i in range(len(L)) if L[i] in B}
```

Que renvoie l'appel `ImR(L,Im(L,{k}))` pour chaque valeur $k \in [\![0, 3]\!]$ pour la liste $L = [4, 4, 2, 4]$? Pour la liste $L = [0, 0, 1, 0]$? En déduire une fonction *Python* `EstEquiv(L,M)` qui décide si $L \equiv M$ (NB. En Python les listes sont indexées à partir de 0).

Réponse. Les ensembles renvoyés par les différents appels sont donnés dans la tableau ci-dessous :

$k =$	0	1	2	3
$L = [4, 4, 2, 4]$	{0,1,3}	{0,1,3}	{2}	{0,1,3}
$L = [0, 0, 1, 0]$	{0,1,3}	{0,1,3}	{2}	{0,1,3}

L'image d'un singleton $\{k\}$ par l'application L est le singleton $\{L[k]\}$ qui est le même pour tous les indices k associés à la même valeur, donc l'image réciproque de ce singleton est l'ensemble des indices qui donnent la même valeur dans la liste :

```
def EstEquiv(L,M):
  k = 0
  while (k < len(L)) and ImR(L,Im(L,{k})) == ImR(M,Im(M,{k})):
    k += 1
  return (k >= len(L))
```

On peut également utiliser la fonction `all` du *Python* :

```
def EstEquiv(L,M):
  return all(ImR(L,Im(L,{k})) == ImR(M,Im(M,{k})) for k in range(len(L)))
```

Question 3. Soit $L \in \mathscr{L}_n$ et $(i, j) \in [\![1, n]\!]^2$ avec $i \neq j$. Montrez que

$$L[i] = L[j] \Rightarrow \pi(L) \notin \{i, j\}.$$

Réponse. Par hypothèse, l'intrus ne peut apparaître qu'à une seule position de la liste. Ainsi, si la liste contient la même valeur à deux positions différentes, cette valeur ne peut pas être l'intrus.

Question 4. Déduire de la question précédente un algorithme `pi(L)` en pseudo-code (que vous commenterez) et qui calcule l'application π, c'est-à-dire qui renvoie la position de l'intrus dans la liste.
NB. On suppose que $L \in \mathscr{L}_n$, ne faites pas de tests de conformité inutiles.

Réponse. Notons $|L| = 2k + r$ avec $r \in \{0, 1\}$ selon que la liste est de longueur paire ou non. Le principe général consiste à balayer la liste de deux en deux tant que les deux valeurs adjacentes $L[i]$ et $L[i + 1]$ sont égales. Plus précisément, grâce à l'évaluation paresseuse de la condition de boucle $(i < |L| - 1) \wedge (L[i] = L[i + 1])$:

- La liste est de longueur paire ($r = 0$). En sortant de la boucle, l'indice i est celui du premier terme du couple qui contient l'intrus. S'il s'agit du dernier couple de la liste, la dernière comparaison réalisée entre termes de la liste dans la condition de boucle s'est faite sur l'*avant dernier* couple, en effet si ses termes sont égaux, l'intrus est nécessairement dans le dernier couple. Dans tous les cas, il faut une dernière comparaison pour décider lequel des termes du couple est l'intrus.
- La liste est de longueur impaire ($r = 1$). En sortant de la boucle, l'indice i est celui du premier terme du couple qui contient l'intrus ou celui du dernier élement de la liste qui est nécessairement l'intrus. Exceptée cette dernière situation, il faut une dernière comparaison pour décider lequel des deux termes de ce couple est l'intrus.

Dans l'algorithme ci-dessous, à l'exception du cas où l'intrus est le dernier élément de la liste ($i = |L|$ dans la condition `06`), la dernière comparaison qui décide de la position de l'intrus dans le couple se fait entre $L[i]$ et $L[i-2]$, sauf s'il s'agit du premier couple, auquel cas on compare $L[i]$ avec $L[i+2]$ (cf. instruction `05` pour en décider) :

```
    ALGORITHME Pi(L):entier
    DONNEES
       L: liste d'entiers
       i, dec: entiers
    DEBUT
01:    i ← 1
02:    TQ ((i < |L| - 1) ET (L[i] = L[i + 1])) FAIRE
03:       i ← i + 2
04:    FTQ
05:    dec ← (i > 2)? -2 : 2
06:    RENVOYER ((i = |L|) OU (L[i] != L[i + dec]))? i : i + 1
    FIN
```

Question 5. Faites une preuve d'arrêt de votre algorithme et fournissez les arguments (informels) pour la preuve de correction partielle.

Réponse. *Arrêt.* Les valeurs successives de la variable i forment la suite d'entiers naturels $(u_n)_{n\in\mathbb{N}}$ définie par la récurrence

$$u_{n+1} := u_n + 2$$

avec pour premier terme $u_0 = 1$. On sait qu'il n'existe pas de suite d'entiers naturels strictement croissante et majorée, par conséquent la condition d'arrêt $i \geqslant |L| - 1$ sera satisfaite.

Correction partielle. La condition $i < |L| - 1$ assure que l'on n'accède jamais à une position inexistante de la liste `L` dans la condition de boucle `02` puisque $i < |L| - 1 \Rightarrow i + 2 \leqslant |L|$ (évaluation paresseuse). La condition $L[i] \neq L[i + dec]$ dans l'expression ternaire `06` est toujours réalisable, car $i \in [\![1, |L| - 2]\!]$ et `dec = -2` uniquement si $i > 2$, auquel cas $i - 2 \geqslant 1$.

La condition de sortie de la boucle `02` est $(i \geqslant |L| - 1) \vee (L[i] \neq L[i+1])$. L'analyse de la sortie de boucle demande un peu de réflexion à cause de l'évaluation paresseuse. Comme nous l'avons déjà évoqué pour la description de l'algorithme, la valeur i à la sortie de la boucle est soit la position du premier terme de la paire contenant l'intrus, quelle que soit la position de cette paire (on ne compare en revanche les deux termes de cette paire que si ce n'est pas la dernière), soit la dernière position dans la liste qui contient alors l'intrus. Dans ce dernier cas l'algorithme renvoie bien la position i. Dans les autres cas, on peut décider quelle est la position de l'intrus dans le couple de positions $(i, i + 1)$ en comparant la valeur en position i à une autre valeur que celle adjacente, par exemple en position $i - 2$ sauf si l'intrus est dans la première paire auquel cas on la compare à la valeur en position $i + 2$.

Question 6. Dénombrez le nombre minimal et le nombre maximal *exacts* de comparaisons de votre algorithme (entre éléments de la liste L uniquement). En déduire sa complexité dans le meilleur et le pire des cas.

Réponse. Dans le meilleur des cas, l'intrus est dans la première paire, il faut donc deux comparaisons entre termes de la liste pour trouver sa position : la première pour déterminer cette paire, la seconde pour déterminer qui de la position 1 ou 2 est celle de l'intrus. La complexité est donc $\Theta(1)$.

Pour le pire des cas, distinguons le cas d'une liste de longueur $n = |L|$ paire d'une liste de longueur impaire. Si $n = 2k$ l'intrus est dans la dernière paire, ce qui aura occasionné $k - 1$ comparaisons pour déterminer que l'intrus est dans cette paire et une dernière pour décider sa position dans la paire, soit au total k comparaisons. Si $n = 2k+1$, il y a eu k comparaisons pour décider que l'intrus était en dernière position. Dans tous les cas, il y a

donc exactement $\lfloor \frac{n}{2} \rfloor$ comparaisons entre termes de la liste. La complexité dans le pire des cas est alors égale à $\Theta(n)$ (constante cachée $= \frac{1}{2}$).

Question 7. Dénombrez le nombre moyen *exact* de comparaisons de votre algorithme (entre éléments de la liste L uniquement). En déduire sa complexité dans le meilleur et le pire des cas. Supposez que la v.a. π définie sur $\mathscr{L}_n$ suit la loi uniforme sur $[\![1, n]\!]$, i.e. $\forall k \in [\![1, n]\!]\ \mathbb{P}[\pi = k] = \frac{1}{n}$.

Réponse. On pose $n = 2k$ ou $n = 2k + 1$ selon que la longueur de la liste L est paire ou impaire. Dans le cas pair, si l'intrus se trouve dans la p-ème paire sauf la dernière, i.e. $1 \leqslant p < k$, il y a exactement $p + 1$ comparaisons et p comparaisons s'il se trouve dans la dernière paire, i.e. si $p = k$. Dans le cas impair, si l'intrus se trouve dans l'une des k paires $1 \leqslant p \leqslant k$, il y a exactement $p + 1$ comparaisons et s'il est le dernier élément, il y en a k. Dans les deux cas, le nombre de comparaisons est compris entre 2 et k.

Comme la position $\pi(L)$ de l'intrus est uniformément distribuée dans l'intervalle $[\![1, n]\!]$, i.e. $\forall i \in [\![1, n]\!]\ \mathbb{P}[\pi = i] = \frac{1}{n}$, la probabilité qu'il soit dans n'importe quelle paire est égale à $\frac{2}{n}$ que n soit pair ou non. On peut donc calculer le nombre moyen de comparaisons entre termes de la liste selon la parité de la longueur de la liste :

$$\begin{aligned} C_{2k} &= \sum_{p=1}^{k-1} \left[\frac{2}{n}(p+1)\right] + \frac{2}{n}k \\ &= \frac{1}{k}\left(\sum_{p=1}^{k-1} [p+1] + k\right) \\ &= \frac{1}{k}\left(\sum_{p=1}^{k} [p] + (k-1)\right) \\ &= \frac{k+3}{2} - \frac{1}{k} \\ &= \frac{n+6}{4} - \frac{2}{n} \end{aligned}$$

et dans le cas impair :

$$\begin{aligned} C_{2k+1} &= \sum_{p=1}^{k} \left[\frac{2}{n}(p+1)\right] + \frac{1}{n}k \\ &= \frac{2}{n}\left(\sum_{p=1}^{k} [p+1] + \frac{k}{2}\right) \\ &= \frac{2}{n}\left(\sum_{p=1}^{k+1} [p] + \frac{k-2}{2}\right) \\ &= \frac{2}{n}\left(\frac{(k+1)(k+2)}{2} + \frac{k-2}{2}\right) \\ &= \frac{k(k+4)}{2k+1} \\ &= \frac{n+6}{4} - \frac{7}{4n} \end{aligned}$$

Asymptotiquement, la parité des longueurs des listes est uniformément répartie, on peut donc estimer le nombre moyen de comparaisons entre termes de la liste en calculant la moyenne des valeurs C_{2k} et C_{2k+1} :

$$C = \frac{n+6}{4} - \frac{15}{8n}.$$

Ainsi la complexité moyenne est en $\Theta(n)$ (constante cachée $= \frac{1}{4}$).

Question 8. Un représentant *minimal* d'une classe de $\mathscr{L}_n/\equiv$ est une liste dont la somme des termes est minimale. Montrez que chaque classe n'admet qu'un seul représentant minimal. Quel est-il ?

Réponse. Soit $L \in \mathscr{L}_n$ une liste telle que $\pi(L) = k$. Considérons la liste M définie par

$$M[i] := \begin{cases} 1 & \text{si } i = \pi(L) \\ 0 & \text{sinon.} \end{cases}$$

Il est clair que $L \equiv M$, or la somme des termes de M est égale à 1. C'est un minimum puisqu'une liste de $\mathscr{L}_n$ contient deux valeurs distinctes. Il y a donc exactement un représentant minimal par classe, caractérisé par la position de la valeur 1.