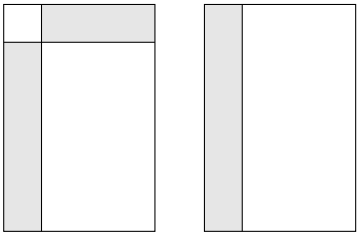
Devoir Surveillé d'Informatique

- **Durée : 4 heures**
- **L'usage de tout dispositif électronique (calculatrice, téléphone portable, ...) est interdit.**
- La clarté et la lisibilité de la copie sont des qualités essentielles. Toute copie sale, peu claire ou à la rédaction approximative sera **pénalisée**. Numérotez vos feuilles doubles et indiquez votre nom sur chacune d'entre elles.



Première feuille Autres feuilles

La première page de la copie **doit laisser un cadre de 5cm**. Votre copie doit comporter une marge d'**au moins 3cm**. Toute copie ne respectant pas ces spécifications perdra des points.

- Un problème, ou un exercice à plusieurs questions, n'est pas un grand chelem, et il est approprié en l'absence de réponse à une question de supposer les résultats et de passer à la suivante. **Ne vous arrêtez pas à la première question dont vous n'avez pas la réponse !** Aucune copie ne sera pénalisée pour avoir tenté un début de réponse.
- S'il vous semble qu'une erreur s'est glissée dans un énoncé, justifiez-vous et poursuivez l'exercice *mutatis mutandis*, c'est-à-dire *en changeant ce qui doit être changé*.

Ce sujet est composé de deux parties (Parties 1 et 2), la Partie 1 étant constituée de deux sous-parties 1.1 et 1.2. **Les trois parties 1.1, 1.2 et 2 sont indépendantes** !!

1 Similarité entre deux séquences

Cette partie vise à mesurer la "similarité" de deux séquences A et B. Ce problème a de nombreuses applications que ce soit en biologie où l'on veut déterminer la "ressemblance" entre deux séquences d'ADN, ou en "informatique" où il s'agit de distinguer les différences entre deux fichiers (par exemple, la fonction unix *diff* compare deux fichiers ligne par ligne)

Dans cette section, lorsque l'on demande de mesurer la complexité d'un algorithme, on considère le nombre d'opérations élémentaires du type "$A.(i) == B.(j)$". C'est-à-dire que l'on compte le nombre de tests d'égalité entre deux éléments de deux séquences. On demande toujours la complexité en pire cas.

Les deux sous-sections 1.1 et 1.2 sont indépendantes !!

1.1 Plus grande sous-séquence commune

Dans cette sous-section, on mesure la similarité entre deux séquences A et B par la longueur d'une plus longue sous-séquence commune.

Une séquence $(v_1, \cdots, v_p)$ est **une sous-séquence** d'une séquence $(u_1, \cdots, u_n)$ ($p \leqslant n$) si il existe $p : \{1, \cdots, p\} \to \{1, \cdots, n\}$ strictement croissante, telle que $v_i = u_{p(i)}$ pour tout $i \leqslant p$.

Précisemment, étant données deux séquences $A = (a_0, \cdots, a_{n-1})$ et $B = (b_0, \cdots, b_{m-1})$, on cherche une séquence $S = (s_0, \cdots, s_{r-1})$, la plus longue possible (on veut maximiser r), telle qu'il existe $0 \leqslant i_0 < i_2 < \cdots < i_{r-1} < n$ et $0 \leqslant j_0 < j_2 < \cdots < j_{r-1} < m$ tels que, pour tout $0 \leqslant k < r$, $s_k = a_{i_k} = b_{j_k}$.

Question 1	Donner une plus longue séquence commune de $A = (X, P, X, L, U, S, X, G, R, A, X, N, D, E, X, S, X, E, Q)$ et $B = (Y, Y, G, Y, R, Y, A, N, Y, D, E, S, E, Q, U, Y, E, N, Y, C, E)$

1.1.1 Algorithme naïf

Question 2	Écrire en CAML une fonction qui prend deux séquences en entrée et teste si elles sont égales. Donner (et prouver) la complexité de votre algorithme.

Question 3	Donner le nombre de sous-séquences d'une séquence de n éléments.

Question 4	Décrire un algorithme "naïf" pour trouver une plus longue séquence commune à deux séquences. Déduire la complexité de votre algorithme de la réponse à la question précédente.

1.1.2 Algorithme par Programmation Dynamique

Dans la suite, on va utiliser la programmation dynamique pour obtenir un bien meilleur algorithme.

Étant donnée une séquence $X = (x_0, \cdots, x_t)$, on note $X_i = (x_0, \cdots, x_i)$ pour tout $i \leqslant t$.

Question 5	Soit $S = (s_0, \cdots, s_{r-1})$ une plus longue sous-séquence commune à $A = (a_0, \cdots, a_{n-1})$ et $B = (b_0, \cdots, b_{m-1})$. Prouver que: 1. si $a_{n-1} = b_{m-1}$ alors S_{r-2} est une plus longue sous-séquence commune à A_{n-2} et B_{m-2}; 2. si $s_{r-1} \neq a_{n-1}$ alors S est une plus longue sous-séquence commune à A_{n-2} et B; 3. si $s_{r-1} \neq b_{m-1}$ alors S est une plus longue sous-séquence commune à A et B_{m-2}.

Pour tout $i < n$ et $j < m$, on note $c[i, j]$ la longueur d'une plus grande sous-séquence commune à A_i et B_j.

Question 6	Déduire de la question précédente que, pour tout $i < n$ et $j < m$: $$c[i,j] = \begin{cases} 0 & \texttt{si } (i = 0 \texttt{ ou } j = 0) \texttt{ et } a_0 \neq b_0 \\ 1 & \texttt{si } (i = 0 \texttt{ ou } j = 0) \texttt{ et } a_0 = b_0 \\ c[i-1, j-1] + 1 & \texttt{si } i, j > 0 \texttt{ et } a_i = b_j \\ \max\{c[i, j-1], c[i-1, j]\} & \texttt{sinon.} \end{cases}$$

Question 7	En utilisant la relation précédente, écrire en CAML un algorithme qui calcule la longueur d'une plus grande séquence commune à deux séquences données en entrée. Donner la complexité de votre algorithme.

1.2 Distance d'édition

Dans cette section, la similiarité va être déterminée par le nombre minimum d'opérations (définies plus loin) pour passer d'une séquence à une autre.

La distance de *Hamming* entre deux séquences $A = (a_0, \cdots, a_{n-1})$ et $B = (b_0, \cdots, b_{n-1})$ **de même longueur** est le nombre de caractères dont elles diffèrent, c'est-à-dire le nombre minimum de caractères qu'il faut substituer dans A pour obtenir B. Précisemment, $d_{Hamming}(A, B) = |\{0 \leqslant i < n \mid a_i \neq b_i\}|$. (On rappelle que $|X|$ dénote le cardinal de l'ensemble X).

Question 8	Prouver que $d_{Hamming}$ est bien une distance.

La distance d'édition permet de comparer des séquences de longueurs différentes en autorisant aussi des insertions et des suppressions. La *distance d'édition* $d_{edit}(A, B)$ entre deux séquences $A = (a_0, \cdots, a_{n-1})$ et $B = (b_0, \cdots, b_{m-1})$ est le nombre minimum d'opérations (substitutions, insertions, suppressions) qui permet de transformer A en B. Par exemple, la distance d'édition entre EDITION et DISTANCE est au plus 6:

EDITION	
DITION	supprimer E
DISTION	insérer S
DISTAON	substituer A $\leftarrow$ I
DISTAN	supprimer O
DISTANC	insérer C
DISTANCE	insérer E

1.2.1 "Oublier" les suppressions

Question 9	Soit σ une suite de ℓ opérations (substitutions, insertions, suppressions) qui permet de passer d'une séquence quelconque A à une autre B. Construire deux suites σ_1 et σ_2 d'opérations (uniquement substitutions et insertions) de longueur respectives ℓ_1 et ℓ_2 telles que • $\ell_1 + \ell_2 \leqslant \ell$, et • il existe une séquence X telle que σ_1 transforme A en X et σ_2 transforme B en X. *Indice : dans l'exemple précédent, on peut prendre X=EDISTAONCE*

La *distance d'édition sans suppression* $d'(A, B)$ entre deux séquences A et B est le nombre minimum d'opérations (substitutions, insertions) qui permet de transformer A en B.

Dans la suite, on admet que d' est bien une distance.

Question 10	Déduire de la question précédente que $d_{edit}(A, B) = \min_X d'(A, X) + d'(B, X)$ où le minimum est pris sur toutes les séquences, et d' est la distance d'édition sans suppression.

1.2.2 Lien avec la distance de Hamming

Soient A et B deux séquences, et X une séquence telle que $d_{edit}(A, B) = d'(A, X) + d'(B, X)$. Soit σ_1 une suite d'opérations (substitutions et insertions) de longueur minimum qui permet de passer de A à X. On note $A-$ la séquence obtenue à partir de A en suivant les opérations de σ_1, mais en n'effectuant pas les substitutions, et en insérant le caractère séparateur "_" à chaque insertion de σ_1. La séquence $B-$ est définie similairement à partir de σ_2 une suite d'opérations (substitutions et insertions) de longueur minimum qui permet de passer de B à X.

Dans l'exemple précédent, $A- =$ EDI_TION__ et $B- =$ _DISTA_NCE.

Question 11	Prouver que $A-$ et $B-$ ont même longueur. Montrer que $d_{edit}(A, B) = d_{Hamming}(A-, B-)$.

1.2.3 Algorithme par Programmation Dynamique

Étant donnée une séquence $X = (x_0, \cdots, x_t)$, on note $X_i = (x_0, \cdots, x_i)$ pour tout $i \leqslant t$. On pose $X_{-1} = \emptyset$.

Pour tout $i < n$ et $j < m$, on note $D[i, j]$ la distance d'édition entre A_i et B_j.

Question 12	Que vaut $D[i, -1]$? $D[-1, j]$?

Question 13	Montrer, pour tout $0 < i < n$ et $0 < j < m$: $$D[i,j] \leqslant \min \begin{cases} D[i-1,j-1] & \texttt{si } a_i = b_j \\ D[i-1,j-1]+1 & \texttt{si } a_i \neq b_j \\ D[i,j-1]+1 & \\ D[i-1,j]+1 & \end{cases}$$ *Cette question est indépendante des questions précédentes*

Question 14	Montrer que la relation de la question précédente est en fait une égalité. *Indice: on pourra utiliser la question 11*

Question 15	En utilisant la relation précédente, écrire en CAML un algorithme qui calcule la distance d'édition entre deux séquences en entrée. Donner la complexité de votre algorithme.

2 Élément majoritaire

Soit $A = (a_0, \cdots, a_{n-1})$ un séquence de n éléments. Pour tout élément x, on note $E_x = \{0 \leqslant i < n \mid a_i = x\}$. Par exemple, si x n'est pas dans la séquence, $E_x = \emptyset$. Un élément x est *majoritaire* si $|E_x| > \lfloor n/2 \rfloor$. (On rappelle que $|X|$ dénote le cardinal de l'ensemble X). On demandera toujours la complexité en pire cas.

Question 16	Prouver qu'une séquence a au plus un élément majoritaire.

Dans cette section, lorsque l'on demande de mesurer la complexité d'un algorithme, on considère le nombre d'opérations élémentaires du type "$A.(i) == B.(j)$". C'est-à-dire que l'on compte le nombre de tests d'égalité entre deux éléments de deux séquences. En CAML, les séquences seront représentées par des tableaux.

2.1 Algorithme "naïf"

Question 17	Écrire, en CAML, un algorithme qui prend en entrée une séquence S et un élément x est détermine si x est un élément majoritaire de S en faisant un nombre linéaire (en la longueur de S) de tests d'égalité.

Question 18	En utilisant l'algorithme précédent, écrire en CAML un algorithme qui prend une séquence S en entrée et détermine si S a un élément majoritaire en temps quadratique.

2.2 Diviser pour Régner

Pour faire mieux, on va utiliser la technique de Diviser pour Régner. Dans la suite, pour simplifier, on ne considère que des séquences dont la longueur est une puissance de 2.

Question 19	Soit x un élément majoritaire d'une séquence $S = (s_0, \cdots, s_{n-1})$ (n est une puissance de 2). Montrer que x est majoritaire dans $(s_0, \cdots, s_{n/2-1})$ ou dans $(s_{n/2}, \cdots, s_{n-1})$.

On va utiliser la question précédente de la façon suivante. On va écrire un algorithme qui prend une séquence $S = (s_0, \cdots, s_{n-1})$ en entrée et retourne $(0, *)$ si S n'a pas d'élément majoritaire, et retourne (k_x, x) si S contient un élément majoritaire x, où k_x est le nombre d'occurrences de x dans S. L'algorithme est récursif : il est d'abord appliqué à $S_1 = (s_0, \cdots, s_{n/2-1})$ et $S_2 = (s_{n/2}, \cdots, s_{n-1})$. Si x est un élément majoritaire de S_1 et y est un élément majoritaire de S_2, il suffit de vérifier si l'un des deux est majoritaire dans S.

Question 20	Écrire en CAML l'algorithme récursif décrit ci-dessus.

Question 21	Prouver que cet algorithme est correct.

Question 22	Calculer sa complexité dans le pire cas.

2.3 Mieux Diviser pour Régner

Pour faire encore mieux, on veut éviter de tester à chaque appel récursif si x ou y est majoritaire dans toute la séquence. Pour cela, on demande un algorithme qui prend en entrée une séquence de longueur n et tel que l'lagorithme possède la spécification suivante:

- soit l'algorithme garantit que S ne possède pas d'élément majoritaire, auquel cas il renvoie "AUCUN";
- soit l'algorithme fournit un entier $p > n/2$ et un élément x tel que x apparaît au plus p fois dans S et tout élément distinct de x apparaît au plus $n - p$ fois dans S.

Question 23	Écrire en CAML l'algorithme récursif décrit ci-dessus, et prouver que sa complexité est n.

Question 24	En déduire un algorithme de complexité linéaire pour déterminer si une séquence a un élément majoritaire. Donner la complexité précise (sans "grand O") de l'algorithme.

2.4 Encore mieux !!

On change la manière de voir les choses. On a un ensemble de n balles et on cherche le cas échéant s'il y a une couleur majoritaire parmi les balles.

Question 25	Supposons que les balles soient rangées en file sur une étagère, de manière à n'avoir jamais deux balles de la même couleur à côté. Que peut-on en déduire sur le nombre maximal de balles de la même couleur ?

On a un ensemble de n balles, une étagère vide où on peut les ranger en file et un tiroir vide. Considérons l'algorithme suivant :

Phase 1 Prendre les balles une par une pour les ranger sur l'étagère ou dans le tiroir. Si la balle n'est pas de la même couleur que la dernière balle sur l'étagère (ou si l'étagère est vide), la ranger à côté sur l'étagère, et si de plus le tiroir n'est pas vide, prendre une balle dans le tiroir et la ranger à côté sur l'étagère. SINON, c'est à dire si la balle est de la même couleur que la dernière balle sur l'étagère, la mettre dans le tiroir.

Phase 2 Soit C la couleur de la dernière balle sur l'étagère à la fin de la phase 1. On compare successivement la couleur de la dernière balle sur l'étagère avec C. SI la couleur est la même on jette les deux dernières balles de l'étagère, sauf s'il n'en reste qu'une, auquel cas on la met dans le tiroir. SINON on la jette et on jette une des balles du tiroir, sauf si le tiroir est déjà vide auquel cas on s'arrête en décrétant qu'il n'y a pas de couleur majoritaire. Quand on a épuisé toutes les balles sur l'étagère, on regarde le contenu du tiroir. Si il est vide alors il n'y a pas de couleur majoritaire, et si il contient au moins une balle alors C est la couleur majoritaire.

Question 26	Montrer qu'à tout moment de la phase 1, toutes les balles éventuellement présentes dans le tiroir ont la couleur de la dernière balle de l'étagère. En déduire que s'il y a une couleur dominante alors c'est C. Prouver la correction de l'algorithme.

Question 27	Donner la complexité dans le pire des cas en nombre de comparaisons de couleurs des balles. (sans "grand O").