



## ÉPREUVE SPÉCIFIQUE - FILIÈRE MP

---

### INFORMATIQUE

**Jeudi 2 mai : 14 h - 18 h**

---

*N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.*

---

**Les calculatrices sont interdites**

**Le sujet est composé de trois parties indépendantes.**

## Partie I - Inversions de permutations (Informatique pour tous)

Pour tout  $n \in \mathbb{N} \setminus \{0\}$ , une **permutation de taille  $n$**  est une bijection de l'ensemble  $\{0, 1, \dots, n-1\}$  dans lui-même. Dans la suite, l'ensemble des permutations de taille  $n$  est noté  $\mathfrak{S}_n$ . Étant donné une permutation  $\sigma$  de taille  $n$ , on la représente sous la forme  $\sigma_0\sigma_1 \cdots \sigma_{n-1}$  où :

$$\forall i \in \{0, \dots, n-1\}, \sigma_i = \sigma(i).$$

Par exemple,  $\sigma = 1032$  représente la permutation envoyant 0 sur 1, 1 sur 0, 2 sur 3 et 3 sur 2.

Soit  $\sigma = \sigma_0 \cdots \sigma_{n-1}$  une permutation de taille  $n$ . On dit que  $(i, j) \in \{0, 1, \dots, n-1\}^2$  est une **inversion** de  $\sigma$  si  $\sigma_i > \sigma_j$  et  $i < j$ . On note  $\text{inv}(\sigma)$  le nombre d'inversions de  $\sigma$ . Tout d'abord, on relie ce nombre avec un algorithme de tri : le tri à bulles. Puis, on s'intéresse à la table d'inversions d'une permutation, un objet qui caractérise une permutation.

Dans la suite, une permutation de taille  $n$  est représentée en Python par une liste sans répétition contenant tous les entiers compris entre 0 et  $n-1$ . Par exemple, la liste  $[1, 0, 3, 2]$  représente la permutation  $\sigma = 1032$ .

Si  $A$  est un ensemble fini,  $\text{Card}(A)$  désigne le cardinal de l'ensemble  $A$ . Pour tout entier  $n \geq 1$ , on pose  $E_n = \prod_{k=1}^n \{0, 1, \dots, n-k\}$ . Par exemple, on a  $E_3 = \{0, 1, 2\} \times \{0, 1\} \times \{0\}$ .

### I.1 - Tri et inversions

**Q1.** Déterminer l'ensemble des inversions de la permutation  $\sigma = 140253$ .

On rappelle l'algorithme de tri à bulles :

---

#### Algorithme 1 - Tri à bulles

---

**Entrées :** Une liste d'entiers  $L$

```
pour  $i$  allant de (taille de  $L$ )-1 à 1 (bornes incluses) faire
    pour  $j$  allant de (taille de  $L$ )-1 à (taille de  $L$ )- $i$  (bornes incluses) faire
        si  $L[j] < L[j-1]$  alors
            | Echanger  $L[j]$  et  $L[j-1]$ 
        fin
    fin
fin
```

---

**Q2.** Écrire une fonction Python `tri_bulle(L)` qui prend en argument une liste d'entiers  $L$  et qui trie cette liste à l'aide du tri à bulles. À l'issue de la fonction, la liste est triée.

Dans la suite, on admet que l'algorithme du tri à bulles est bien un algorithme de tri.

**Q3.** Montrer que si on effectue exactement un échange dans une liste lors du tri à bulles, la nouvelle liste a exactement une inversion en moins.

**Q4.** En déduire une fonction Python `nombre_inversions(L)` qui prend en argument une liste  $L$  correspondant à une permutation et renvoyant le nombre d'inversions de celle-ci. Cette fonction sera une légère modification du tri à bulles.

## I.2 - Table d'inversions d'une permutation

**Définition 1** (Table d'inversions). Soit  $\sigma$  une permutation de taille  $n \geq 1$ . La **table d'inversions** de  $\sigma$  est le  $n$ -uplet  $(\alpha_0, \dots, \alpha_{n-1})$  tel que :

$$\forall i \in \{0, \dots, n-1\}, \alpha_i = \text{Card}(\{j \in \{i+1, \dots, n-1\} \mid \sigma_j < \sigma_i\}).$$

Elle est notée  $\mathbf{Tab}_\sigma$ . De plus, pour tout  $i \in \{0, \dots, n-1\}$ ,  $\mathbf{Tab}_\sigma[i]$  désigne  $\alpha_i$ .

Pour tout  $n \geq 1$ , on désigne par  $\mathbf{Tab} : \mathfrak{S}_n \rightarrow E_n$  l'application qui associe à une permutation de taille  $n$  sa table d'inversions.

- Q5.** Déterminer la table d'inversions de la permutation  $\sigma = 140253$ .
- Q6.** Montrer que pour toute permutation  $\sigma$  de taille  $n \geq 1$ ,  $\mathbf{Tab}_\sigma$  est bien un élément de  $E_n$ .
- Q7.** Soit  $\sigma$  une permutation de taille  $n \geq 1$ . Montrer que  $\sigma_0 = \mathbf{Tab}_\sigma[0]$ .
- Q8.** Montrer que pour tout entier  $n \geq 1$ , l'application  $\mathbf{Tab}$  est bijective.
- Q9.** Écrire une fonction Python `permutation_vers_table(L)` qui prend en argument une permutation représentée par la liste  $L$  et qui renvoie la table d'inversions correspondante.
- Q10.** Écrire une fonction Python `table_vers_permutation(L)` qui prend en argument une liste  $L$  qui correspond à une table d'inversions et qui renvoie la permutation qui lui est associée.

## Partie II - Théorie des automates et des langages rationnels

Dans toute cette partie, la lettre  $\varepsilon$  désigne le mot vide,  $\Sigma$  désigne un alphabet et  $\Sigma^*$  l'ensemble des mots finis sur  $\Sigma$ .

### II.1 - Définitions

**Définition 2** (Automate déterministe). Un **automate déterministe**  $A$  est un quintuplet  $A = (Q, \Sigma, q_0, F, \delta)$ , avec :

- $Q$  un ensemble d'états ;
- $\Sigma$  un alphabet ;
- $q_0$  l'état initial ;
- $F \subseteq Q$  un ensemble d'états finaux ;
- $\delta : Q \times \Sigma \rightarrow Q$  une application de transition.

**Définition 3** (Langage). Un **langage** sur  $\Sigma$  est une partie de  $\Sigma^*$ .

**Définition 4** (Application de transition étendue aux mots). Soit  $A = (Q, \Sigma, q_0, F, \delta)$  un automate déterministe. On définit de manière récursive  $\delta^* : Q \times \Sigma^* \rightarrow Q$  par :

$$\begin{aligned} \forall q \in Q, \quad \delta^*(q, \varepsilon) &= q \\ \forall q \in Q, \forall a \in \Sigma, \forall w \in \Sigma^*, \quad \delta^*(q, aw) &= \delta^*(\delta(q, a), w). \end{aligned}$$

Cette application  $\delta^*$  vérifie alors la propriété admise suivante :

$$\forall q \in Q, \forall v \in \Sigma^*, \forall w \in \Sigma^*, \delta^*(q, vw) = \delta^*[\delta^*(q, v), w].$$

**Définition 5** (Langages rationnels). On rappelle que les langages rationnels sont définis de manière inductive par :

- l'ensemble  $\emptyset$  est un langage rationnel ;
- les langages  $\{a\}$  où  $a$  est une lettre, sont rationnels ;
- si  $L$  et  $L'$  sont des langages rationnels,  $L.L', L \cap L', L \cup L'$  sont des langages rationnels ;
- si  $L$  est un langage rationnel,  $L^*$  est un langage rationnel.

**Définition 6** (Carré d'un langage). Soit  $L$  un langage sur  $\Sigma$ . Le **carré du langage**  $L$  est l'ensemble  $\{uu, u \in L\}$ . Il est noté  $L \odot L$ .

**Définition 7** (Racine carrée d'un langage). Soit  $L$  un langage sur  $\Sigma$ . La **racine carrée du langage**  $L$  est l'ensemble  $\{u \in \Sigma^* \mid uu \in L\}$ . Elle est notée  $\sqrt{L}$ .

On pourra utiliser sans démonstration le théorème ci-dessous :

**Théorème 1.** Soit  $L$  un langage. Il est rationnel si et seulement s'il existe un automate déterministe et fini le reconnaissant.

Dans la suite, on décrit un langage rationnel par une expression rationnelle.

## II.2 - Racine carrée d'un langage

### Exemples

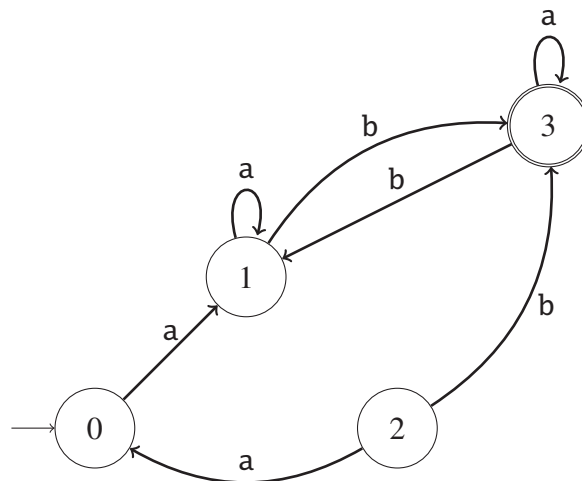
**Q11.** Décrire  $\sqrt{L}$  lorsque  $\Sigma = \{a, b\}$  et  $L$  est décrit par l'expression rationnelle  $a^*b^*$ .

**Q12.** Décrire  $\sqrt{L}$  lorsque  $\Sigma = \{a, b\}$  et  $L$  est décrit par l'expression rationnelle  $b^*a^*b^*$ .

### Construction d'automates

**Définition 8.** Soient  $A = (Q, \Sigma, q_0, F, \delta)$  un automate fini déterministe,  $q'$  un élément de  $Q$  et  $F'$  une partie de  $Q$ . L'automate  $(Q, \Sigma, q', F', \delta)$  est noté  $A_{q', F'}$ . Si on note  $L$  le langage reconnu par  $A$ ,  $L_{q', F'}$  désigne le langage reconnu par  $A_{q', F'}$ .

Ici,  $L$  désigne le langage reconnu par l'automate  $A$  suivant :



**Q13.** Construire un automate reconnaissant  $L_{3,\{1\}}$  en modifiant légèrement l'automate  $A$ .

**Q14.** On veut construire l'automate de Glushkov de  $L$  décrit par  $a(a + ba^*b)^*ba^*$ .

1. Décrire  $L'$ , le linéarisé de  $L$ .
2. Déterminer les préfixes de  $L'$  de longueur 1, les suffixes de  $L'$  de longueur 1 et les facteurs de  $L'$  de longueur 2.
3. En déduire l'automate de Glushkov  $G$  de  $L$ .

**Q15.** Déterminer l'automate  $G$ .

### Propriétés de la racine carrée d'un langage rationnel

Ici, on fixe  $L$  un langage rationnel sur un alphabet  $\Sigma$  et  $A = (Q, \Sigma, q_0, F, \delta)$  un automate fini reconnaissant celui-ci.

**Q16.** Soit  $u$  un mot de  $\Sigma^*$ . Montrer que  $u$  est un élément de  $\sqrt{L}$  si et seulement s'il existe un  $q \in Q$  tel que  $u \in L_{q_0, \{q\}}$  et  $u \in L_{q, F}$ .

**Q17.** En déduire que  $\sqrt{L}$  est un langage rationnel.

**Q18.** Montrer que l'on a  $(\sqrt{L} \odot \sqrt{L}) \subset L$ .

## Partie III - Algorithmique des mots sans facteur carré

L'objectif de cette partie est de construire différents algorithmes pour vérifier si un mot comporte des facteurs carrés ou non.

Dans toute la suite,  $\Sigma = \{a_1 < \dots < a_p\}$  désigne un alphabet totalement ordonné comportant  $p$  lettres,  $\varepsilon$  représente le mot vide et  $\Sigma^*$  est l'ensemble des mots finis obtenus à partir de  $\Sigma$ . Pour tout réel  $x$ , on note  $\lfloor x \rfloor$  la partie entière de  $x$ .

### III.1 - Définitions

**Définition 9** (Longueur d'un mot). Soit  $w = w_0 \dots w_{n-1}$  un mot de  $\Sigma^*$ . La **longueur**  $n$  de  $w$  est notée  $|w|$ , pour tout  $0 \leq i \leq j < n$ ,  $w[i, j]$  désigne le mot  $w_i \dots w_j$ . Par convention, si  $j < i$ ,  $w[i, j]$  désigne  $\varepsilon$ .

**Définition 10** (Mot carré). Soit  $w$  un mot de  $\Sigma^*$ . On dit que  $w$  est un **carré** s'il existe un mot  $x$  tel que  $w = x \cdot x$ .

**Définition 11** (Facteur d'un mot). Soient  $v$  et  $w$  deux mots de  $\Sigma^*$ . On dit que  $v$  est un **facteur** de  $w$  s'il existe  $r$  et  $s$  deux mots (éventuellement vides) tels que  $w = rvs$ .

**Définition 12** (Répétition). On dit qu'un mot  $w$  contient une **répétition** s'il contient un facteur carré différent de  $\varepsilon$ .

Dans la suite, un mot sera représenté en Caml par la liste de ses lettres. Par exemple, le mot *baba* est représenté par la liste  $['b'; 'a'; 'b'; 'a']$  et le mot vide est représenté par la liste  $[\ ]$ .

### III.2 - Fonctions utiles sur les listes

**Q19.** Écrire une fonction récursive Caml de signature `longueur : 'a list -> int` qui renvoie la longueur de la liste.

**Q20.** Écrire une fonction Caml de signature `sous_liste : 'a list -> int -> int -> 'a list` où `sous_liste L k long` renvoie une liste `S` qui est la sous-liste de `L` commençant à l'indice `k` et de longueur `long`. On suppose que l'indexation des listes commence à 0.

On pourra dans la suite de l'énoncé utiliser les fonctions `longueur` et `sous_liste`.

### III.3 - Un algorithme naïf

**Q21.** Préciser si les mots suivants contiennent ou non une répétition.

1. *aabfa*
2. *abfdanq*
3. *ababa*
4. *avba*.

**Q22.** Soit  $w$  un mot contenant au plus deux lettres différentes. Montrer que si  $|w| \geq 4$  alors  $w$  contient au moins une répétition.

**Q23.** Écrire une fonction Caml de signature `estCarre : 'a list -> bool` prenant en argument une liste  $w$  et retournant `true` si  $w$  est un carré et `false` sinon.

**Q24.** Déterminer la complexité en nombre de comparaisons de lettres de la fonction `estCarre`.

**Q25.** Écrire une fonction Caml de signature `contientRepetitionAux : 'a list -> int -> bool` prenant en argument une liste  $w$  et un entier  $m$  et retournant `true` si  $w$  contient une répétition de la forme  $xx$  avec  $x$  de longueur  $m$  et `false` sinon.

**Q26.** Montrer que toute répétition d'un mot  $w$  de longueur  $n$  est de la forme  $xx$  avec  $|x| \leq \frac{n}{2}$ .

**Q27.** En déduire une fonction Caml de signature `contientRepetition : 'a list -> bool` prenant en argument une liste  $w$  retournant `true` si  $w$  contient une répétition et `false` sinon.

**Q28.** Quelle est la complexité en nombre de comparaisons de caractères de la fonction `contientRepetition` ?

### III.4 - Algorithme de Main-Lorentz

L'algorithme de Main-Lorentz permet de détecter de manière plus efficace des répétitions d'un mot  $w$ . Il comporte essentiellement deux parties :

- la première consiste à voir si étant donné deux mots  $u$  et  $v$ , le mot  $uv$  contient un carré non nul issu de la concaténation ;
- la deuxième s'appuie sur le principe de "diviser pour régner".

Remarquons qu'un mot  $uv$  contient une répétition si et seulement si  $u$  ou  $v$  contiennent une répétition ou  $uv$  contient des répétitions provenant de la concaténation. Pour déterminer si un mot  $uv$  contient de nouvelles répétitions, on commence par effectuer des prétraitements consistant à calculer des tables de valeurs de  $u$  et de  $v$  qui sont généralement appelées tables de préfixes (ou suffixes). Avant de présenter des algorithmes permettant de générer ces tables, on commence par justifier leur application dans la détection de répétitions.

**Définition 13** (Carré centré). Soient  $u$  et  $v$  deux mots. On dit que  $uv$  contient un **carré centré** sur  $u$  (respectivement sur  $v$ ) s'il existe un mot  $w$  non vide et des mots  $u', v'', w', w''$  tels que  $u = u'ww'$ ,  $v = w''v''$ ,  $w = w'w''$  (respectivement  $u = u'w'$ ,  $v = w''wv''$ ,  $w = w'w''$ ).

**Définition 14** (Plus long préfixe commun, plus long suffixe commun). Soient  $u$  et  $v$  deux mots de  $\Sigma^*$ . Le **plus long préfixe** (respectivement **suffixe**) **commun** de  $u$  et  $v$  est le plus long mot  $w$  tel qu'il existe deux mots  $r$  et  $s$  tels que  $u = wr$  et  $v = ws$  (respectivement  $u = rw$  et  $v = sw$ ). On le note  $\text{lcp}(u, v)$  (respectivement  $\text{lcs}(u, v)$ ).

### À propos des carrés centrés

**Q29.** Dans cette question,  $\Sigma = \{a, b\}$ . Soient  $u = abababaa$  et  $v = ababaaa$ . Déterminer le plus grand préfixe commun de  $u$  et  $v$ .

**Q30.** Soient  $u$  et  $v$  deux mots de  $\Sigma^*$ . Montrer que  $uv$  contient un carré centré sur  $u$  si et seulement s'il existe  $i \in \{0, \dots, |u| - 1\}$  tel que  $|\text{lcs}(u[0, i - 1], u)| + |\text{lcp}(u[i, |u| - 1], v)| \geq |u| - i$ .

De la même manière, on peut montrer que  $uv$  contient un carré centré sur  $v$  si et seulement s'il existe  $j \in \{1, \dots, |v| - 1\}$  tel que  $|\text{lcs}(v[0, j - 1], u)| + |\text{lcp}(v, v[j, |v| - 1])| \geq |v| - j$ .

Ainsi, pour pouvoir déterminer s'il existe un carré centré sur  $u$  ou  $v$ , on peut utiliser les valeurs :

$$|\text{lcs}(u[0, i - 1], u)|, |\text{lcp}(u[i, |u| - 1], v)|, |\text{lcs}(v[0, j - 1], u)|, |\text{lcp}(v, v[j, |v| - 1])|.$$

Dans la suite, étant donné deux mots  $u$  et  $v$ , on note  $\text{pref}_u$ ,  $\text{pref}_{u,v}$ ,  $\text{suff}_u$  et  $\text{suff}_{u,v}$  les tableaux vérifiant :

$$\forall i \in \{0, \dots, |u| - 1\}, \quad \text{pref}_u[i] = |\text{lcp}(u[i, |u| - 1], u)|, \quad \text{pref}_{u,v}[i] = |\text{lcp}(u[i, |u| - 1], v)|$$

$$\text{suff}_u[i] = |\text{lcs}(u[0, i], u)|, \quad \text{suff}_{u,v}[i] = |\text{lcs}(u[0, i], v)|.$$

### Calcul de table de préfixes

On présente un algorithme permettant le calcul de la table  $\text{pref}_u$  ainsi que sa complexité en nombre de comparaisons de caractères en page 8. En adaptant cet algorithme, il est également possible de calculer la table  $\text{pref}_{u,v}$  en  $O(|u|)$  de comparaisons de caractères.

**Q31.** On pose  $u = aabbba$  et  $v = abbaab$ . Déterminer les tableaux  $\text{pref}_u$  et  $\text{pref}_{u,v}$  sans justification.

**Q32.** En déroulant l'**algorithme 2** de la page suivante appliqué au mot  $u = aaabaaabaaab$ , compléter le tableau

$i =$	$f$	$g$	$\text{pref}[i]$
0	—	0	12
1	1	3	2
2	·	·	·
⋮	⋮	⋮	⋮
11	4	12	0

de la façon suivante : pour une valeur  $i$  donnée, on indique les valeurs de  $f, g, \text{pref}[i]$  à l'issue des instructions internes de la boucle.

Par exemple, à l'initialisation,  $i = 0$ ,  $f$  n'est pas définie,  $g$  vaut 0 et  $\text{pref}[0] = 12$ . Pour  $i = 1$ , à l'issue des instructions internes à la boucle, on a  $f = 1, g = 3, \text{pref}[1] = 2$ .

**Q33.** Dédurre de l’algorithme 2 une procédure calculant  $\text{suff}_u$ .

Dans la suite, on suppose que l’algorithme  $\text{tabpref}(u, v)$  qui prend en argument deux chaînes de caractères  $u$  et  $v$  et qui renvoie la table  $\text{pref}_{u,v}$  nous est donné. On admet que la complexité de cet algorithme est de  $O(|u|)$  en nombre de comparaisons de caractères.

**Q34.** Dédurre des questions précédentes un algorithme qui, étant donnés deux mots  $u$  et  $v$ , renvoie VRAI s’il existe un carré centré sur  $u$  et FAUX sinon.

**Q35.** Quelle est la complexité de cet algorithme en nombre de comparaisons de caractères ?

### Application des tables

**Q36.** Dédurre des questions précédentes un algorithme récursif qui prend en argument une chaîne de caractères et qui renvoie VRAI si la chaîne contient une répétition et FAUX sinon.

**Q37.** Déterminer la complexité de cet algorithme en nombre de comparaisons de caractères.

---

#### Algorithme 2 - Calcul de la table $\text{pref}_u$

---

**Entrées :** une chaîne de caractères  $u$

**Sorties :** un tableau  $\text{pref}_u$

$i \leftarrow 0$ ,  $\text{pref} \leftarrow$  tableau de taille  $|u|$  initialisé à 0,  $\text{pref}[i] \leftarrow |u|$ ,  $g \leftarrow 0$

**pour**  $i$  allant de 1 à  $|u| - 1$  **faire**

**si**  $i < g$  et  $\text{pref}[i - f] < g - i$  **alors**

$\text{pref}[i] \leftarrow \text{pref}[i - f]$

**fin**

**sinon si**  $i < g$  et  $\text{pref}[i - f] > g - i$  **alors**

$\text{pref}[i] \leftarrow g - i$

**fin**

**sinon**

$(f, g) \leftarrow (i, \max(g, i))$

**tant que**  $g < |u|$  et  $u[g] == u[g - f]$  **faire**

$g \leftarrow g + 1$

**fin**

$\text{pref}[i] \leftarrow g - f$

**fin**

**fin**

---

On admet que la complexité est de  $O(|u|)$  en nombre de comparaisons de caractères.

**FIN**