On rappelle que
- `Array.make : int → 'a → 'a array` telle que `Array.make n v` renvoie un tableau de longueur n dont toutes les cases valent v.
- `Array.make_matrix : int → int → 'a → 'a array` telle que `Array.make n p v` renvoie une matrice ayant n lignes et p colonnes dont toutes les cases valent v.
- `Array.length : 'a array → int` elle que `Array.length t` renvoie la longueur de `t`.
- `String.length : String → int` telle que `String.length u` renvoie la longueur de la chaine de caractère `u`.
- Si `u` est de type `String`, `u.[i]` renvoie le i-ème caractère de `u`.

Un mot est une suite de lettres $a_0 \dots a_{n-1}$ tirées d'un alphabet fini $A = \{a, b, \dots\}$. On utilisera $u, v, u', u'', u_1, u_2, \dots$ pour dénoter les éléments de $A^\star$, c'est-à-dire les mots sur A. On note ε pour le mot vide et $|u|$ pour la longueur de u, de sorte que $|\varepsilon| = 0$.

Si un mot u se décompose sous la forme $u = u_1 v u_2$, alors v est un *facteur* de u, et même un préfixe (ou un suffixe) si $u_1 = \varepsilon$ (ou si $u_2 = \varepsilon$) dans cette décomposition. Dans le cas d'un mot $u = a_0 \dots a_{n-1}$, on écrit « $u[i, j[$ », sous la condition $0 \leqslant i \leqslant j \leqslant n$, pour désigner le facteur $a_i \dots a_{j-1}$. Cette notation s'étend à $u[i \dots [$ et $u[i]$ pour désigner, respectivement, $u[i, n[$ et $u[i, i+1[$.

Ce que l'on appelle sous-mot de u correspond à la notion classique de sous-suite, ou de suite extraite, et ne doit pas être confondu avec un facteur. Pour $u = a_0 \dots a_{n-1}$, on dira qu'un mot v de longueur m est un *sous-mot* de u, ce que l'on notera $v \preccurlyeq u$, s'il existe une suite strictement croissante $0 \leqslant p_0 < p_1 < \dots < p_{m-1} < n$ telle que $v = a_{p_0} a_{p_1} \dots a_{p_{m-1}}$. Par exemple, `caml` $\preccurlyeq$ `bechamel`. Formellement, pour tout $n \in \mathbb{N}$, nous noterons $[n]$ pour l'ensemble $\{0, 1, 2, \dots, n-1\}$, de sorte que la suite $p_0, p_1, \dots, p_{m-1}$ peut être vue comme une application strictement croissante $p : [m] \to [n]$. Pour une telle application, on note $v = u \circ p$ pour dire que v est le sous-mot extrait de u via p et on dit que p est un *plongement* de v dans u, noté $p : v \preccurlyeq u$. Notons qu'il peut exister plusieurs manières différentes de plonger v dans u.

1) a) Montrez que pour deux mots u et u' et deux lettres a et a', on a l'équivalence suivante :

$$ua \preccurlyeq u'a' \iff ua \preccurlyeq u' \text{ ou } (a = a' \text{ et } u \preccurlyeq u') \tag{1}$$

   b) Programmez une fonction Ocaml `teste_sous_mot : string -> string -> bool` décidant en temps polynomial si un mot v est sous-mot d'un mot u. Détaillez et justifiez votre analyse de complexité.

On note $\binom{u}{v}$ le nombre de plongements de v dans u, de sorte que $v \preccurlyeq u$ si et seulement si $\binom{u}{v} > 0$. Notons en particulier que $\binom{u}{\varepsilon} = 1$ pour tout mot $u \in A^\star$ car il n'existe qu'une injection de $[0]$, c'est-à-dire $\emptyset$, dans $\{0, 1, \dots |u| - 1\}$ et cette injection est bien un plongement.

2) a) Montrez que $\binom{\texttt{abab}}{\texttt{ab}} = 3$.

   b) Que vaut $\binom{a^n}{a^m}$ quand $a \in A$ est une lettre ? On rappelle que a^n est le mot constitué de n occurrences de la lettre a.

   c) Montrez que $\binom{ua}{va} = \binom{u}{va} + \binom{u}{v}$ pour tous mots $u, v \in A^\star$ et toute lettre $a \in A$.

Pour calculer $\binom{u}{v}$ on considère la fonction OCaml suivante.

```
let nb_plongements (v:string) (u:string) =
  let rec aux i j =
    if i = 0 then 1
    else if j = 0 then 0
    else if v.[i-1] = u.[j-1] then (aux (i-1) (j-1)) + (aux i (j-1))
    else aux i (j-1)
  in
  aux (String.length v) (String length u)
;;
```

3) a) Prouvez sa terminaison.

   b) Justifiez sa correction, c'est-à-dire, expliquez pourquoi elle renvoie bien la valeur $\binom{u}{v}$.

On note $T(v,u)$ le nombre de fois où la fonction `aux` est appelée lors du calcul de `nb_plongements` v u.

4) a) Montrez qu'il existe une constante C_1 telle que $T(v,u) < 2^{|u|} \cdot C_1$.

   b) Montrez que l'on ne peut pas majorer $T(v,u)$ par une fonction polynomiale de $\binom{u}{v}$.

   c) Montrez qu'il existe une constante C_2 telle que $T(v,u) \geqslant 2\binom{u}{v} + C_2$.

La question précédente a montré que la fonction `nb_plongements` proposée dans le sujet demande un temps de calcul parfois exponentiel en la taille $|u|+|v|$ de ses arguments. De meilleurs algorithmes existent…

5) Programmez en OCaml une nouvelle fonction `nb_plongements_rapide : string -> string -> int` qui calcule $\binom{u}{v}$ en temps polynomial en $|u|+|v|$. Détaillez votre analyse de complexité en temps et en espace.

   Indication : on pourra utiliser la programmation dynamique.

On cherche maintenant à dénombrer les sous-mots d'un mot u. On note $\downarrow u$ pour $\{v \mid v \preccurlyeq u\}$. Il s'agit d'un langage fini. Par exemple $\downarrow\texttt{abab} = \{\varepsilon, \texttt{a}, \texttt{b}, \texttt{ab}, \texttt{aa}, \texttt{ba}, \texttt{bb}, \texttt{aab}, \texttt{aba}, \texttt{abb}, \texttt{bab}, \texttt{abab}\}$ de sorte que `abab` a 12 sous-mots distincts, ce que l'on note $\mathrm{Card}(\downarrow\texttt{abab}) = 12$.

Les langages étant des ensembles (des parties $L, L', \ldots$ de $A^\star$), on utilisera les notations $L \cup L'$, $L \backslash L'$, etc. avec leur signification ensembliste habituelle. On utilise aussi la notation $L \cdot L'$ pour désigner le produit de concaténation de deux langages : $L \cdot L' = \{uv | u \in L, v \in L'\}$. Dans le cas d'un singleton $L = \{u\}$, on écrit souvent $u \cdot L'$ au lieu de $\{u\} \cdot L'$.

6) a) Montrez que, pour tous mots v, w et toute lettre a, on a :

$$\downarrow wava = \downarrow wav \cup (\downarrow wav \backslash \downarrow w) \cdot a \tag{2}$$

   b) Montrez que l'union $\downarrow wav \cup (\downarrow wav \backslash \downarrow w) \cdot a$ est disjointe si et seulement si le mot v ne contient pas la lettre a.

Quand l'union est disjointe dans l'équation (2), on peut obtenir $\mathrm{Card}(\downarrow u)$, pour $u = wava$, en combinant $\mathrm{Card}(\downarrow wav)$ et $\mathrm{Card}(\downarrow w)$. Cette approche se généralise au cas d'un mot u quelconque.

7) a) Donnez des équations récursives permettant de calculer $\mathrm{Card}(\downarrow u)$ en se ramenant à des préfixes de u. On pourra considérer par exemple les diverses occurrences de la dernière lettre de u quand elle existe.

   b) En se basant sur vos équations, programmez une fonction OCaml `nb_sousmots : string -> int` qui, pour un mot u donné, calcule $\mathrm{Card}(\downarrow u)$ en temps polynomial en $|u|$. Justifiez votre analyse de complexité.

Un *sur-mot* commun à u et v est un mot w tel que $u \preccurlyeq w$ et $v \preccurlyeq w$. Il existe une infinité de tels mots. Parmi tous ces sur-mots communs à u et v, on s'intéresse à celui qui est le plus court, et qui est le premier dans l'ordre lexicographique pour départager les sur-mots communs de même longueur. Ce mot est noté $\mathrm{pcsmc}(u,v)$, et par exemple $\mathrm{pcsmc}(\texttt{informatique}, \texttt{difficile}) = \texttt{difnficormatilque}$.

8) a) Soient a, b deux lettres distinctes. Montrez que si $\mathrm{pcsmc}(ua, vb) = wa$ alors $w = \mathrm{pcsmc}(u, vb)$, ceci pour tous mots u, v, w.

   b) Généralisez la propriété précédente en donnant des équations qui permettent de caractériser $\mathrm{pcsmc}(ua, vb)$ dans le cas général, y compris quand $a = b$.

   c) Programmez une fonction OCaml calculant $\mathrm{pcsmc}(u,v)$ en temps polynomial en $|u|+|v|$ pour des mots u et v arbitraires. Détaillez votre analyse de complexité.

   On pourra utiliser l'opérateur `^` qui permet de concaténer deux chaines de caractères ainsi que la fonction `String.sub` telle que si `u` est une chaine de caractère et `i` et `k` deux entiers, `String.sub u i k` renvoie la sous-chaine de caractères de `u` qui commence à la lettre d'indice `i` et de longueur `k`.