DM4

Soit Σ un alphabet contenant au moins deux lettres. Le mot vide est noté ε. Σ^* est l'ensemble des mots sur Σ. Si $u \in \Sigma^*$, on note $|u|$ sa longueur. Pour tout $k \in [\![0, |u|-1]\!]$, on note u_k la $(k+1)$-ième lettre de u. En OCaml, tout élément de Σ est représenté par le type `char`. Tout élément de Σ^* est représenté par le type `string`.

On appelle *bord* d'un mot non vide u, un mot à la fois *préfixe propre* et *suffixe propre* de u. Le *bord maximal* de u est l'unique bord de longueur maximale, noté $\beta(u)$. Par exemple, les bords du mot $u = ababa$ sont a, aba et $\beta(u) = aba$. On convient que $\beta(\varepsilon) = \varepsilon$.

Question 1.
□ **1.1.** Soit v un mot non vide. Montrer qu'il existe $k \leqslant |v|$ tel que l'ensemble $\{\beta(v), \beta^2(v), \dots, \beta^k(v)\}$ soit l'ensemble des bords de v.
□ **1.2.** Soit v un mot non vide de longueur k et a une lettre. Montrer que $\beta(va)$ est le plus long des mots de l'ensemble $\{\varepsilon, \beta(v)a, \beta^2(v)a, \dots, \beta^k(v)a\}$ qui sont préfixes de v.
□ **1.3.** Étant donné un mot non vide u de longueur n, on définit $b(0), b(1), \dots, b(n)$ en posant :

$$b(0) = -1 \qquad \text{et} \qquad \forall k \in [\![1, n]\!],\ b(k) = |\beta(u_0 u_1 \cdots u_{k-1})|$$

$b(k)$ est la longueur du bord maximal du préfixe de longueur k de u.
Soit $k < n$. On considère les entiers $j_1 = b(k)$, $j_2 = b(j_1)$, ..., $j_{k+1} = b(j_k)$. Montrer que $j_{k+1} = -1$. On peut considérer le plus petit entier α pour lequel $u_{j_\alpha} = u_k$ ou $j_\alpha = -1$. Montrer que $b(k+1) = j_\alpha + 1$.

Question 2. Déduire de ce qui précède une fonction `bord : string -> int array` qui au mot u associe le tableau des entiers $[b(0), b(1), \dots, b(n)]$. On admet que le coût de cette fonction est linéaire.

Question 3. La notion de bord est principalement utilisée dans l'algorithme KMP (*Knuth Morris Pratt*) dont l'objectif est de déterminer si un mot m (motif) est facteur d'un autre mot s (source).
□ **3.1.** En considérant une lettre $x \in \Sigma$ qui n'appartient ni à m ni à s, expliquer comment le calcul du tableau b associé au mot $u = mxs$ peut permettre de résoudre ce problème.
□ **3.2.** En déduire la fonction `kmp : string -> string -> bool` correspondante.
□ **3.3.** Quel est le coût de celle-ci ?

Question 4. Deux mots u et v sont dits *conjugués* s'il existe deux mots r et s tels que $u = rs$ et $v = sr$. À l'aide de la fonction `kmp`, écrire une fonction `conjugue : string -> string -> bool` qui détermine en temps linéaire si deux mots sont conjugués.

Question 5. Un mot u contient un facteur carré s'il existe trois mots v_1, w, v_2 tels que $u = v_1 w w v_2$. À l'aide du tableau des bords, écrire une fonction `carre : string -> bool` qui détermine en temps quadratique si un mot contient un facteur carré.

Question 6. Une *période* d'un mot u est un préfixe non vide v de u tel que u soit préfixe de v^n avec $n \geqslant 1$. Par exemple, abc est une période de $abcabcab$.
□ **6.1.** On pose $u = vw$ avec $v \neq \varepsilon$. Montrer que v est une période de u si et seulement si w est un bord de u.
□ **6.2.** En déduire une fonction `periode : string -> string` qui déterminer la plus petite période de u en temps linéaire.

Question 7. Écrire une fonction `prefixe_palindrome : string -> string list` qui détermine la liste de tous les préfixes d'un mot qui sont des palindromes.

Annexe

- `val (^): string -> string -> string`
  String concatenation. Right-associative operator.
  Raises `Invalid_argument` if the result is longer then than `Sys.max_string_length` bytes.
- `val String.length : string -> int`
  `length s` is the length (number of bytes/characters) of `s`.
- `val String.sub : string -> int -> int -> string`
  `sub s pos len` is a string of length `len`, containing the substring of `s` that starts at position `pos` and has length `len`.
  Raises `Invalid_argument` if `pos` and `len` do not designate a valid substring of `s`.
- `val Char.escaped : char -> string`
  Return a string representing the given character, with special characters escaped following the lexical conventions of OCaml. All characters outside the ASCII printable range (32..126) are escaped, as well as backslash, double-quote, and single-quote.