DM1

Exercice 1

Un mot est une suite de lettres $a_0 \dots a_{n-1}$ d'un alphabet fini $A = \{\mathtt{a}, \mathtt{b}, \dots\}$. Les éléments de $A^\star$, c'est-à-dire les mots sur A, sont dénotés par u, v, u', u'', u_1, u_2, etc. 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 $u[i, i+1[$.
Dans la suite, un *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. Formellement, pour $u = a_0 \dots a_{n-1}$, on dit 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 $0 \leqslant p_0 < p_1 < \cdots < p_{m-1} < n$ telle que $v = a_{p_0} a_{p_1} \dots a_{p_{m-1}}$. Par exemple, `caml` $\preccurlyeq$ `bechamel`.
Pour tout $n \in \mathbb{N}$, $[n]$ désigne 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. On dit que p est un *plongement* de v dans u, noté $p : v \preccurlyeq u$. Il peut exister plusieurs façons différentes de plonger v dans u.

Question 1.
□ **1.1.** Montrer que pour deux mots u et u' et deux lettres a et a', on a l'équivalence suivante.

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

□ **1.2.** Programmer une fonction `teste_sous_mot : string -> string -> bool` décidant en temps polynomial si un mot v est sous-mot d'un mot u. Détailler et justifier votre analyse de complexité.

Question 2. 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 $\varnothing$, dans $\{0, 1, \dots |u| - 1\}$ et cette injection est bien un plongement.
□ **2.1.** Montrer que $\binom{\mathtt{abab}}{\mathtt{ab}} = 3$.
□ **2.2.** 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.
□ **2.3.** Montrer que $\binom{ua}{va} = \binom{u}{va} + \binom{u}{v}$ pour tous mots $u, v \in A^\star$ et toute lettre $a \in A$.

Question 3. Pour calculer $\binom{u}{v}$ on considère la fonction 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);;
```

□ **3.1.** Prouver sa terminaison.
□ **3.2.** Justifier sa correction, c'est-à-dire, expliquer pourquoi elle renvoie bien la valeur $\binom{u}{v}$.

Question 4. Soit $T(v, u)$ le nombre d'appels à la fonction `aux` lors du calcul de `nb_plongements v u`.
□ **4.1.** Montrer qu'il existe une constante C_1 telle que $T(v, u) < 2^{|u|} \cdot C_1$.
□ **4.2.** Montrer que l'on ne peut pas majorer $T(v, u)$ par une fonction polynomiale de $\binom{u}{v}$.
□ **4.3.** Montrer qu'il existe une constante C_2 telle que $T(v, u) \geqslant 2\binom{u}{v} + C_2$.

Question 5. La fonction `nb_plongements` précédente requiert parfois un temps de calcul exponentiel en la taille $|u| + |v|$ de ses arguments. De meilleurs algorithmes existent. Proposer une nouvelle fonction `nb_plongements_rapide` qui calcule $\binom{u}{v}$ en temps polynomial en $|u| + |v|$. Détailler votre analyse de complexité en temps et en espace.

Rappels OCaml. Une chaine de caractères `s` a le type `string`, sa longueur est obtenue avec `String.length s` et son `i`-ième caractère avec `s.[i]`, les caractères étant indexés à partir de 0. Un tableau `t` a le type τ `array`, où τ est le type des éléments. Sa longueur est obtenue avec `Array.length t`. Son `i`-ième élément est obtenu avec `t.(i)` et modifié avec `t.(i) <- val`, les éléments étant indexés à partir de 0. L'expression `Array.make n val` construit un tableau de taille `n` dont les éléments sont initialisés avec la valeur `val`. Une matrice est un tableau de tableaux de même taille. L'expression `Array.make_matrix n m val` construit une matrice de `n` lignes et `m` colonnes, dont les éléments sont tous initialisés avec la valeur `val`.

Exercice 2

Question 1. Montrer que le nombre c_n de comparaisons effectuées par le tri fusion sur une liste des n éléments vérifie $c_1 = 0$ et la récurrence suivante : $\forall n \geqslant 2 \quad c_n = c_{\lfloor n/2 \rfloor} + c_{\lceil n/2 \rceil} + n$. En déduire la valeur de c_{2^p}.

Question 2. On pose $d_n = c_{n+1} - c_n$.
□ **2.1.** Quelle relation de récurrence définit la suite (d_n) ?
□ **2.2.** Montrer que $d_n = 2 + \lfloor \log_2 n \rfloor$.
□ **2.3.** En déduire une expression de c_n.

Question 3.
□ **3.1.** Combien faut-il de bits $\nu(k)$ pour coder un entier k ? Parmi les entiers $1, 2, \dots, n-1$, combien ont au moins $(p+1)$ bits dans leur représentation binaire ?
□ **3.2.** On pose $c'_n = \sum_{k=1}^{n-1} \nu(k)$. Montrer que :

$$c'_n = \sum_{p \geqslant 0} (n - 2^p)$$

la somme s'arrêtant au dernier terme positif.
□ **3.3.** Que représente finalement $c_n - (n-1)$? En déduire c_n.

Question 4.
□ **4.1.** Montrer que c_n peut s'écrire :

$$c_n = n \log_2 n + n\varphi(1 - \{\log_2 n\})$$

où $\{x\} = x - \lfloor x \rfloor$ est la partie fractionnaire de x et φ est une fonction à déterminer.
□ **4.2.** Tracer l'allure de la courbe représentative de φ sur $[0, 1]$.