Numéro à préciser

(Durée :)

On étudie dans ce problème l'ordre lexicographique pour les mots sur un alphabet fini et plusieurs constructions des cycles de De Bruijn. Les trois parties sont largement indépendantes.

Définitions

- Dans tout le problème, $\mathcal{A}$ est un alphabet fini, de cardinal k, muni d'une relation d'ordre total notée $\prec$. Pour simplifier les notations, on identifiera $\mathcal{A}$ au sous-ensemble $\{0, 1, \ldots, k-1\}$ des entiers naturels, et $\prec$ à l'ordre naturel sur les entiers. On appelle *lettres* les éléments de $\mathcal{A}$.
- $\mathcal{A}^+$ est l'ensemble des mots non vides sur l'alphabet $\mathcal{A}$. La longueur d'un mot $s \in \mathcal{A}^+$ est notée $|s|$. On note $s[i]$, $1 \leq i \leq |s|$, la i-ème lettre du mot s, appelée aussi la lettre en position i. Le *sous-mot* $s[i..j]$ de s, $1 \leq i \leq j \leq |s|$, est le mot composé des lettres de s en position allant de i à j. Les *préfixes* du mot s sont les sous-mots $s[1..j]$ pour $1 \leq j < |s|$ et ses *suffixes* sont les sous-mots $s[i..|s|]$ pour $1 < i \leq |s|$ (noter qu'on ne considère pas le mot lui-même comme son propre préfixe, ni comme son propre suffixe).

– La *concaténation* de deux mots s et t de $\mathcal{A}^+$ est notée $s \cdot t$. C'est le mot u de longueur $|s| + |t|$ tel que $u[i] = s[i]$ si $1 \leq i \leq |s|$ et $u[i] = t[i - |s|]$ si $|s| + 1 \leq i \leq |s| + |t|$. On note s^i le mot obtenu en concaténant i fois le mot s avec lui-même (formellement, $s^1 = s$ et $s^i = s \cdot s^{i-1}$ pour $i \geq 2$).

– On note $\leq_{lex}$ l'ordre *lexicographique* sur $\mathcal{A}^+$, appelé aussi ordre du dictionnaire, et défini comme suit. Soient s et t deux mots de $\mathcal{A}^+$. On note $s <_{lex} t$ si :
- $s[1] \prec t[1]$
- ou $\exists k$, $2 \leq k \leq \min(|s|, |t|)$, tel que $s[i] = t[i]$ pour $1 \leq i < k$ et $s[k] \prec t[k]$
- ou $|s| < |t|$ et $s[i] = t[i]$ pour $1 \leq i \leq |s|$

Alors $s \leq_{lex} t$ si $s = t$ ou $s <_{lex} t$.

Structures de données

– Pour représenter un mot $s \in \mathcal{A}^+$, on utilisera un tableau d'entiers de taille $|s| + 1$, indexé de 0 à $|s|$: l'élément d'indice 0 sera égal à $|s|$ et l'élément d'indice i sera égal à $s[i]$ pour $1 \leq i \leq |s|$.

– Par ailleurs, on utilisera des listes d'entiers qu'on manipulera à l'aide des primitives suivantes. On note NIL la liste vide. Si Q est non vide ($Q \neq$ NIL), la primitive `tête`(Q) renvoie le premier élément de la liste et la primitive `queue`(Q) renvoie la liste constituée des éléments suivants. La primitive `ajoute-fin`(Q, i) ajoute l'entier i à la fin de la liste Q. Si Q et Q' sont deux listes, la primitive `concat`(Q, Q') construit une liste composée des éléments de Q, suivis des éléments de Q'.

– Enfin, une liste est *triée* si ses éléments sont rangés par ordre croissant (au sens large).

Algorithmes et pseudo-programmes

– Pour les questions qui demandent la conception d'un algorithme : il s'agit de décrire en français, de façon concise mais précise, les idées essentielles de votre réponse.

– Pour les questions qui demandent l'écriture d'un pseudo-programme : il s'agit d'exprimer votre algorithme dans un langage de votre choix, avec les structures de données (tableaux ou listes) décrites ci-avant, et les structures de contrôle (boucles, conditionnelles, ...) classiques.

– Le *coût* d'un algorithme ou d'un pseudo-programme est le nombre d'opérations élémentaires qu'il effectue. Une opération élémentaire est une comparaison ou un test d'égalité entre deux lettres, un appel à l'une des primitives précédentes sur les listes d'entiers, un accès en lecture ou en écriture à une case de tableau, un incrément de compteur de boucle.

– Le coût d'un algorithme ou d'un pseudo-programme ne sera pas calculé exactement mais seulement estimé en ordre de grandeur, avec des expressions du type $O(m + n)$, $O(m^2 \log n)$, etc, où $m, n, \ldots$ sont des paramètres en entrée de l'algorithme. Bien sûr, on s'attachera à concevoir des algorithmes et des pseudo-programmes de coût le plus faible possible.

Partie 1. Tri par paquets et ordre lexicographique

Dans cette partie, on considère n mots $s_1, s_2, \ldots, s_n$ de $\mathcal{A}^+$. On pose $\ell_{\max} = \max_{1 \leq i \leq n} |s_i|$ et $M = \sum_{i=1}^{n} |s_i|$ (M est la taille des données). On veut trier ces n mots selon l'ordre lexicographique : on cherche une permutation σ de $\{1, 2, \ldots, n\}$ telle que $s_{\sigma(i)} \leq_{lex} s_{\sigma(i+1)}$ pour $1 \leq i < n$. On utilise un tableau `tab` de tableaux d'entiers : `tab`$[i]$ représente le mot s_i. La permutation σ est représentée par un tableau d'entiers `SIG` de taille n, indexé de 1 à n.

Question 1.1.

1. Écrire un pseudo-programme `compare` qui compare deux mots s et t de $\mathcal{A}^+$ pour l'ordre lexicographique $\leq_{lex}$. Quel est son coût en fonction de $|s|$ et $|t|$?

2. Proposer un algorithme de tri des n mots (calcul du tableau `SIG`) basé sur le pseudo-programme `compare`, et donner son coût dans le pire cas en fonction de n et $\ell_{\max}$.

```
Q ← NIL
pour i croissant de 1 à n faire
  ajoute-fin(Q, i)
fin pour
pour j décroissant de ℓ à 1 faire
  pour p croissant de 0 à k − 1 faire
    P[p] ← NIL
  fin pour
  tant que (Q ≠ NIL) faire
    i ← tête(Q)
    Q ← queue(Q)
    ajoute-fin(P[tab[i][j]], i)
  fin tant que
  pour p croissant de 0 à k − 1 faire
    Q ← concat(Q, P[p])
  fin pour
fin pour
pour i croissant de 1 à n faire
  SIG[i] ← tête(Q)
  Q ← queue(Q)
fin pour
```

Algorithme `TriPaquets1`

```
Q ← NIL
pour p croissant de 0 à k − 1 faire
  P[p] ← NIL
fin pour
pour j décroissant de ℓmax à 1 faire
  Q ← concat(Longueur[j], Q)
  tant que (Q ≠ NIL) faire
    i ← tête(Q)
    Q ← queue(Q)
    ajoute-fin(P[tab[i][j]], i)
  fin tant que
  tant que (Présent[j] ≠ NIL) faire
    p ← tête(Présent[j])
    Présent[j] ← queue(Présent[j])
    Q ← concat(Q, P[p])
    P[p] ← NIL
  fin tant que
fin pour
pour i croissant de 1 à n faire
  SIG[i] ← tête(Q)
  Q ← queue(Q)
fin pour
```

Algorithme `TriPaquets2`

Question 1.2.

Dans cette question, on suppose que les n mots ont la même longueur ℓ : $|s_i| = \ell$ pour $1 \leq i \leq n$ (et donc $M = n\ell$). Le principe de l'algorithme `TriPaquets1` décrit à la Figure 1 est le suivant. Il y a ℓ étapes, une pour chaque position des lettres dans les mots. On prépare k paquets $P[0], P[1], \ldots, P[k-1]$ (un paquet par lettre), réinitialisés à chaque étape. À chaque étape j, les indices i des mots s_i ayant la lettre p en position j sont rangés dans le paquet $P[p]$. Les paquets $P[p]$ et Q sont des listes d'entiers.

1. Exécuter l'algorithme `TriPaquets1` sur l'exemple suivant : $n = 5$, $\ell = 3$, $s_1 = 210$, $s_2 = 100$, $s_3 = 112$, $s_4 = 102$, et $s_5 = 110$.
2. Montrer que le coût de l'algorithme `TriPaquets1` est en $O((k+n)\ell)$.
3. Quelle propriété vérifie Q à la fin de la première itération de la boucle sur j (c'est-à-dire lorsque $j = \ell$) ?
4. Même question à la fin de la j-ème itération de cette boucle ? Conclusion ?
5. Pourquoi l'algorithme `TriPaquets1` procède-t-il à partir de la dernière lettre des mots et non pas de la première ?

Question 1.3.

On suppose maintenant que les n mots ont des longueurs arbitraires.

1. Expliquer comment se ramener au cas de n mots de longueur $\ell_{\max}$ pour utiliser l'algorithme `TriPaquets1`. Quel est le coût ?
2. On va améliorer l'algorithme `TriPaquets1` ; on procède en trois étapes :
   ÉTAPE 1 : On prépare $\ell_{\max}$ listes triées `Présent`$[j]$: pour $1 \leq j \leq \ell_{\max}$, `Présent`$[j]$ est la liste triée des lettres qui apparaissent en position j dans l'un au moins des n mots.
   ÉTAPE 2 : On prépare $\ell_{\max}$ listes `Longueur`$[j]$: pour $1 \leq j \leq \ell_{\max}$, `Longueur`$[j]$ est la liste des indices des mots de longueur j.
   ÉTAPE 3 : On utilise l'algorithme `TriPaquets2` de la Figure 1.
   (a) Exécuter les trois étapes pour l'exemple suivant : $k = 3$, $n = 4$, $s_1 = 21$, $s_2 = 0$, $s_3 = 012$ et $s_4 = 101$ (donc $\ell_{\max} = 3$).
   (b) Proposer un algorithme pour préparer les listes de l'étape 1 avec un coût $O(k + M)$. (*Indication : utiliser les idées de l'algorithme* `TriPaquets1`.)
   (c) Quel est le coût de la préparation des listes de l'étape 2 ?
   (d) Montrer que cet algorithme en trois étapes calcule `SIG` correctement.
   (e) Montrer que le coût total est en $O(k + M)$.

Partie 2. Cycles de De Bruijn

Un *cycle de De Bruijn* d'ordre n sur l'alphabet $\mathcal{A}$ est un mot $s \in \mathcal{A}^+$ de longueur $|s| = k^n$ tel que tout mot de $\mathcal{A}^+$ de longueur n est un sous-mot de $s \cdot s[1..(n-1)]$ (ce qui revient à considérer s de façon cyclique). On note $\mathcal{DB}(n)$ l'ensemble de ces cycles. Par exemple :
- si $k = 2$, $u_2 = 0011 \in \mathcal{DB}(2)$ et $u_3 = 00011101 \in \mathcal{DB}(3)$
- si $k = 3$, $v_2 = 002212011 \in \mathcal{DB}(2)$ et $v_3 = 000222122021121020120011101 \in \mathcal{DB}(3)$.
On va montrer l'existence de cycles de De Bruijn pour tout n et tout k.

Question 2.1.

1. Dans un mot de $\mathcal{DB}(n)$, combien de fois apparaît chaque lettre de l'alphabet $\mathcal{A}$?
2. Proposer un algorithme qui vérifie si un mot est un élément de $\mathcal{DB}(n)$. Quel est son coût (en fonction de k et de n) ? Peut-on diminuer le coût en augmentant l'espace mémoire utilisé ?
3. Que peut-on dire du mot infini $m = 001102122031323304142434405152535455506...$ construit par récurrence ? (*Indication : s'intéresser au cas* $n = 2$.)

Question 2.2.

Soient n et k fixés. On construit le mot s de taille maximale comme suit :
- $s[1] = s[2] = \ldots = s[n] = 0$
- pour $i \geq n$, $s[i+1]$ est la plus grande lettre de $\mathcal{A}$, si elle existe, telle que $s[i-n+2..i+1]$ (de longueur n) n'est pas un sous-mot de $s[1..i]$.

1. Écrire un pseudo-programme `suivant` qui calcule (si c'est possible) $s[i+1]$ à partir de $s[1..i]$ pour $i \geq n$. Quel serait le coût d'un pseudo-programme pour tout le calcul du mot s (en fonction de k, n et $|s|$) ?

2. On va montrer que $|s| = k^n + n - 1$ et que $s[1..k^n] \in \mathcal{DB}(n)$ (les mots u_2, u_3, v_2 et v_3 ont été construits de cette façon).
   (a) Soit z le suffixe de s de taille $n-1$. Montrer que pour toute lettre a de $\mathcal{A}$, $a \cdot z$ est un sous-mot de s. En déduire que $z = 0^{n-1}$ (c'est-à-dire que s se termine par $n-1$ zéros).
   (b) Montrer que tous les mots de $\mathcal{A}^+$ de longueur n qui finissent par $n-r$ zéros apparaissent dans s, pour tout $r \geq 1$. Conclure.

Partie 3. Colliers, primaires et cycles

On définit sur $\mathcal{A}^+$ la relation d'équivalence suivante (décalage circulaire) :

$$s \sim t \Leftrightarrow (s = t) \text{ ou } (\exists u, v \in \mathcal{A}^+, s = u \cdot v \text{ et } t = v \cdot u).$$

Un *collier* est un mot inférieur ou égal (pour $\leq_{lex}$) à chacun des mots de sa classe d'équivalence. On note $\mathcal{C}^+$ l'ensemble des colliers : $s \in \mathcal{C}^+ \Leftrightarrow s \in \mathcal{A}^+$ et $s \leq_{lex} t$ pour tout $t \in \mathcal{A}^+, s \sim t$. On dit qu'un mot $s \in \mathcal{A}^+$ est *périodique* si s peut s'écrire $s = t^p$, avec $t \in \mathcal{A}^+$ et $p \geq 2$. Un *primaire* est un collier qui n'est pas périodique. On note $\mathcal{L}^+$ l'ensemble des primaires (l'usage du $\mathcal{L}$ est en référence à Lyndon qui a étudié les propriétés de ces mots). Enfin, pour $n \geq 1$, on note C_n (resp. L_n) le nombre de colliers (resp. de primaires) de longueur n.

Question 3.1.

1. Vérifier que si $s \in \mathcal{A}^+$ est périodique et $t \sim s$, alors t est périodique.
2. Pour $n = 4$ et $k = 2$, donner tous les mots de $\mathcal{L}^+$ de longueur inférieure ou égale à n. Même question pour $n = 3$ et $k = 3$.
3. Pour les deux exemples précédents, que peut-on dire du mot obtenu en énumérant dans l'ordre lexicographique, et en les concaténant, tous les mots de $\mathcal{L}^+$ dont la longueur divise n ?

Question 3.2.

Soit $s \in \mathcal{A}^+$ s'écrivant $s = x \cdot y = y \cdot x$, où $x, y \in \mathcal{A}^+$. On va montrer que s est périodique.

1. Soit $n = |s|$ et $m = |x|$. Montrer que $s[i] = s[i+m]$ pour $1 \leq i \leq n-m$ et $s[i] = s[i+m-n]$ pour $n-m+1 \leq i \leq n$ (s est donc inchangé par décalage circulaire de m positions).
2. Soit $d = \mathtt{PGCD}(m, n)$ et $z = s[1..d]$. Montrer que $s = z^{n/d}$. (*Indication : considérer les décalages circulaires de jm positions.*)

Question 3.3.

1. Montrer que tout mot $s \in \mathcal{A}^+$ peut s'écrire de manière unique $s = t^p$ avec $t \in \mathcal{A}^+$ non périodique et $p \geq 1$.
2. Écrire un pseudo-programme `racine` qui, étant donné $s \in \mathcal{A}^+$, calcule le mot t non périodique tel que $s = t^p$. Quel est son coût en fonction de $|s|$?
3. Montrer que tout collier $s \in \mathcal{C}^+$ peut s'écrire $s = t^p$ avec $t \in \mathcal{L}^+$ et $p \geq 1$. Montrer que $C_n = \sum_{d|n} L_d$ (la somme porte sur les diviseurs positifs de n).
4. Que vaut la somme $\sum_{d|n} d\, L_d$?

Question 3.4.

1. Soit $s \in \mathcal{A}^+$. Montrer l'équivalence des trois propriétés suivantes :
   (i) $s \in \mathcal{L}^+$.
   (ii) s est inférieur à tous ses décalages cycliques : $s = u \cdot v$ avec $u, v \in \mathcal{A}^+ \Rightarrow u \cdot v <_{lex} v \cdot u$.
   (iii) s est inférieur à tous ses suffixes : $s = u \cdot v$ avec $u, v \in \mathcal{A}^+ \Rightarrow s <_{lex} v$.
2. Factorisation en mots primaires :
   (a) Soit $u, v \in \mathcal{L}^+$ avec $u <_{lex} v$. Montrer que $u \cdot v \in \mathcal{L}^+$.
   (b) Montrer que tout mot $s \in \mathcal{A}^+$ peut s'écrire sous la forme $s = p_1 \cdot p_2 \cdots p_m$, où $p_i \in \mathcal{L}^+$ $(1 \leq i \leq m)$ et $p_m \leq_{lex} \ldots \leq_{lex} p_2 \leq_{lex} p_1$.
   (c) Montrer que dans la factorisation précédente, p_m est plus petit (pour l'ordre $\leq_{lex}$) que tout suffixe de s. En déduire l'unicité de cette factorisation.
   (d) Montrer enfin que si $s \notin \mathcal{L}^+$, alors p_1 est le plus long préfixe de s appartenant à $\mathcal{L}^+$.

Question 3.5.

Un *préprimaire* est un mot de $\mathcal{A}^+$ qui est soit préfixe d'un primaire, soit un mot dont toutes les lettres sont égales à $(k-1)$. On note $\mathcal{P}^+$ l'ensemble des préprimaires. La n-extension d'un mot $s \in \mathcal{A}^+$ est le mot de taille n obtenu en répétant s suffisamment de fois et en gardant les n premières lettres (formellement, c'est le préfixe de taille n de s^i où $i \times |s| \geq n$).

1. Soit $p \in \mathcal{L}^+$. Montrer que $p^m \in \mathcal{P}^+$ pour tout $m \geq 1$.
2. *(Difficile.)* Soit $s \in \mathcal{L}^+$ et t un préfixe de s. Soit a une lettre de $\mathcal{A}$ et $u = t \cdot a$. Montrer que si $s <_{lex} u$ alors $u \in \mathcal{L}^+$.
3. Soit $s \in \mathcal{P}^+$ et p_1 le premier primaire dans la factorisation de s en mots primaires. Montrer que s est la $|s|$-extension de p_1.
4. Montrer que $s \in \mathcal{P}^+$ si et seulement si s est la $|s|$-extension d'un mot $p \in \mathcal{L}^+$ de longueur $|p| \leq |s|$. Montrer l'unicité de p.
5. Soit $s \in \mathcal{P}^+$ et p le primaire dont s est la $|s|$-extension. Montrer que $s \in \mathcal{L}^+$ si et seulement si $p = s$, et que $s \in \mathcal{C}^+$ si et seulement si $|p|$ divise $|s|$.

Question 3.6.

On note $\mathcal{P}(n)$ l'ensemble des préprimaires de longueur n.

1. Soit $s \in \mathcal{P}(n)$, $s \neq (k-1)^n$. Déterminer le successeur de s dans $\mathcal{P}(n)$, c'est-à-dire le mot $t \in \mathcal{P}(n)$ tel que $s <_{lex} t$ et $s <_{lex} u \Rightarrow t \leq_{lex} u$ pour tout $u \in \mathcal{P}(n)$. (*Indication : incrémenter une lettre de s pour obtenir t.*)
2. Écrire un pseudo-programme `successeur` qui calcule le successeur d'un mot $s \in \mathcal{P}(n)$, pour $s \neq (k-1)^n$.
3. Donner un algorithme qui énumère dans l'ordre lexicographique, tous les préprimaires de longueur égale à n. Donner un algorithme qui énumère dans l'ordre lexicographique, tous les primaires dont la longueur divise n.
4. *(Très difficile.)* Montrer que si on énumère dans l'ordre lexicographique, en les concaténant, tous les primaires dont la longueur divise n, on obtient un mot $z \in \mathcal{DB}(n)$. Montrer que $z \leq_{lex} z'$ pour tout $z' \in \mathcal{DB}(n)$ (ainsi z est le plus petit cycle de De Bruijn pour l'ordre lexicographique).