MITRO210: Automates et données structurées

Feuille d'exercices 4 – Corrigé

Antoine Amarilli

1 Plus long facteur commun

Question 0. L'algorithme le plus naïf consiste à tester tous les facteurs de u et tous les facteurs de v et à tester l'égalité : on obtient une complexité de $O(m^2n^2n)$.

Moins naïf, on peut plutôt tester toutes les positions de départ, et étendre autant que possible à droite à chaque fois. On obtient $O(mn^2)$.

Question 1. On définit un tableau à double entrée T à m lignes et n colonnes tel que $T[i, j]$ stocke la plus grande longueur d'un facteur commun commençant à la position i de u et à la position j de v.

Les cas de base sont que $T[i, m] = T[n, j] = 0$ pour chaque i et j.

L'induction est que $T[i, j]$ pour $i < m$ et $j < n$ vaut 0 si $u[i] \neq v[j]$ et sinon $T[i, j]$ vaut $1 + T[i + 1, j + 1]$.

On cherche à connaître le maximum de ce tableau.

On peut aussi voir cet algorithme sans programmation dynamique : pour chaque "décalage" de lecture entre les deux mots, on lit les mots avec ce décalage (en restant dans les bornes) et on compte la plus longue séquence consécutive de lettres égales.

Question 2. On considère le mot $w = u\#v\$$ où $\#$ et $\$$ sont des terminateurs distincts qui n'apparaissent ni dans u ni dans v.

On construit un trie suffixe compressé pour w en temps $O(|w|)$. On distingue alors deux types de feuilles : les feuilles correspondant à un suffixe de w qui contient $\#$, et correspondent donc à un suffixe de u concaténé à $\#v\$$; et les feuilles correspondant à un suffixe de w qui ne contient pas $\#$, et correspondent donc à un suffixe de v. On calcule de bas en haut dans l'arbre une information indiquant, pour chaque nœud, à quel type de feuilles il a accès.

On remarque alors qu'un nœud z du trie suffixe compressé ayant accès aux deux types de feuilles correspond à un z qui apparaît à la fois comme facteur de u (c'est-à-dire comme préfixe d'un suffixe de w de la forme $u'\#v\$$ pour u' un suffixe de u) et comme facteur de v (c'est-à-dire comme préfixe d'un suffixe de w de la forme $v'\$$ pour v' un suffixe de v). À l'inverse, pour chaque tel facteur, on peut le retrouver dans le trie comme un nœud (éventuellement un nœud implicite, c'est-à-dire éventuellement au milieu d'une transition étiquetée par un mot) ayant accès aux deux types de feuilles. Mais on remarque d'ailleurs que le plus long tel facteur n'est pas un nœud implicite : pour tout nœud implicite ayant accès aux deux types de feuilles, la destination

de l'arête au milieu de laquelle se trouve le nœud implicite a également accès aux deux types de feuilles et correspond à un facteur plus long.

On veut donc trouver le nœud z correspondant au facteur le plus long possible. Mais on peut explorer l'arbre de haut en bas, en se souvenant à chaque fois que l'on traverse une arête de la longueur du mot qui l'étiquette, ainsi on connaît la longueur des mots représentés par chaque nœud.

Cet algorithme est en temps $O(n+m)$.

Remarque : une idée analogue est de parcourir les deux tries de façon simultanée, en explorant les transitions qui existent dans les deux. Ceci dit, comme il s'agit de tries compressés, il n'est pas évident d'implémenter ce parcours en temps linéaire : si on est à des nœuds x et y des deux tries qui ont chacun une transition sortante étiquetée par des mots commençant par a, il faut voir quel est le plus long préfixe de ces deux mots dans les deux tries qui est identique. Ce problème (revenant au problème "Longest common extension") n'est pas évident a priori. La construction d'un unique trie pour w nous permet en particulier de gérer cette difficulté.

Question 3. L'algorithme présenté se généralise sans difficulté au cas d'un ensemble E de chaînes. Il s'exécute en temps $O(||E|| \times |E|)$, où $||E||$ dénote la longueur totale des chaînes et $|E|$ dénote leur nombre. En effet la concaténation et la construction du trie est en $O(||E||)$, mais ensuite on doit remonter dans le trie l'information de laquelle des $|E|$ types de feuilles est accessible depuis chaque nœud.

Un algorithme astucieux qui élimine le facteur $O(|E|)$ et s'exécute en $O(||E||)$ est donné dans [1], page 205.

On note que la complexité est très différente du problème de trouver le *sous-mot* le plus long qui soit commun à plusieurs chaînes, et qui est NP-difficile quand le nombre et la longueur des chaînes d'entrée ne sont pas bornées.

2 Énumération de facteurs

Question 0. On énumère tous les facteurs (il y en a $O(n^2)$) et on teste l'appartenance (en $O(n)$). On peut faire mieux en testant à partir de tous les points de départ (il y en a $O(n)$) et en lisant en $O(n)$ et en relevant les passages par un état final. Il faut que les facteurs soient représentés par le couple (i,j) de leurs extrémités, et non par leur contenu : ceci garantit que chaque résultat est de taille constante plutôt que linéaire, sinon la complexité dans le pire cas serait forcément $\Omega(n^3)$. Noter qu'on veut effectivement énumérer tous les occurrences de facteurs de la forme (i,j) : certains de ces facteurs peuvent en réalité représenter le même mot.

Question 1. Il y a potentiellement $O(n^2)$ résultats à énumérer, par exemple si on cherche les facteurs de la forme a^* dans $u = a^n$. Donc on ne peut pas espérer faire mieux que $O(n^2)$ dans le pire cas.

Question 2. On lit le mot u de droite à gauche. Quand on lit le i-ème caractère, a_i, on veut pouvoir savoir efficacement s'il y a des valeurs de j telles que $a_i \cdots a_{j-1}$ soit dans a^+, et si oui toutes les produire par ordre croissant.

À chaque fois qu'on découvre le caractère a_i et que ce n'est pas un a, on n'énumère rien.

À chaque fois qu'on découvre le caractère a_i et que c'est un a, on énumère le facteur a_i, puis on étend ce facteur à droite autant que possible.

La complexité de l'algorithme est $O(n)$ pour la lecture du mot, plus $O(m)$ pour m le nombre de résultats produits, soit $O(n+m)$ sur un mot de longueur n avec m résultats à produire.

Question 3. Quand on découvre a_i et que c'est un b, on s'en souvient et on stocke sa position.

Quand on découvre un c, on remet tout à zéro.

Quand on découvre a_i et que c'est un a et qu'on avait lu un b soit juste avant soit avant le début du bloc de a courant, on se souvient de la parité du nombre de a lus de manière contigus jusqu'au b. Quand ce compte est pair et non-nul, on produit le résultat $a_i \cdots a_{j-1}$ pour a_{j-1} la position du b mémorisé.

La complexité de l'algorithme est $O(n+m)$ à nouveau. Noter qu'on ne produit qu'un résultat par position gauche au plus (de fait, il ne peut pas y avoir deux résultats qui commencent à la même position gauche, contrairement au langage a^+).

Question 4. On suppose que l'automate est complet (et on rappelle qu'il est déterministe).

On va se souvenir, pour chaque $0 \leq i \leq n$, après la lecture du i-ème caractère du mot, de l'information suivante : pour chaque état q de l'automate, quel est la prochaine position où, en partant de q à partir de cette position et en lisant le reste du mot, on passera par un état final q' ; et l'identité de cet état final. (Ou bien on note qu'il n'y a pas de tel état.) Spécifiquement on fait un tableau T indexé par la position i et les états q tel que $T[i,q]$ qui contient soit une paire (j,q') avec j la première position à droite de i où le run depuis q atteint un état final et q' l'état en question, soit $\perp$.

Initialement, quand on n'a encore rien découvert, pour chaque état il n'y a pas de prochain état accessible, donc on a $T[n,q] = \perp$ pour chaque q.

Ensuite, quand on découvre le caractère a_i, connaissant les valeurs de $T[i,q]$ pour chaque q, on calcule les nouvelles informations comme suit : pour chaque état q, pour $q' = \delta(q,a_i)$ l'état obtenu depuis q' en lisant a_i, si q' est final alors $T[i-1,q] = (i,q')$, sinon $T[i-1,q] = T[i,q']$. Ensuite, on énumère les résultats : pour q_0 l'état initial, si $T[i-1,q_0] = \perp$ alors il n'y a rien à énumérer, sinon on commence par q_0, on note $(q_1,j_1) = T[i-1,q_0]$, on énumère le facteur (i,j_1), puis on recommence avec q_1 : soit $T[j_1,q_1] = \perp$ et on s'arrête, soit $(q_2,j_2) = T[j_2,q_2]$ et on recommence. On continue ainsi jusqu'à atteindre $\perp$.

La complexité est ainsi $O(nK+m)$ où n est la longueur du mot, K le nombre d'états de l'automate A, et m le nombre de résultats à produire.

Question 5. Le rapport avec la question précédente est que pour $L_1 = L_3 = \Sigma^*$ on retombe sur précisément le même problème.

Voici comment adapter la construction pour cette tâche plus complexe. Lors de la découverte d'un nouveau caractère, on fait les opérations suivantes :

— On met à jour l'information de si le suffixe actuel est accepté par A_3 ou pas. Ceci peut se faire par une construction analogue à celle de la question précédente, mais où on se souvient juste de quels états mènent à un état final à la fin du mot. On dit qu'une position est *3-bonne* si le suffixe correspondant est accepté par A_3 et *3-mauvaise* sinon.

— On fait en parallèle la construction de la question précédente pour l'automate A_2, mais on ne s'intéresse plus au premier passage par un état final mais au premier passage à un état final à une position qui est 3-bonne. La construction s'adapte vu qu'on sait à chaque position si elle est 3-bonne ou 3-mauvaise. On mémorise aussi, à chaque position, si elle est *2-mauvaise* ou *2-bonne*, selon que la valeur de q_0 pour l'état initial dans cette position est $\perp$ ou non.
— Enfin, on fait en parallèle la construction de la question précédente pour l'automate A_1, cette fois en mémorisant les passages par un état final à une position 2-bonne.

Pour l'énumération : on ne peut cette fois énumérer qu'une fois le début du mot atteint.

On regarde l'état initial de l'automate A_1 à la première position. Soit il mène à $\perp$: dans ce cas, il n'y a aucun découpage xyz du mot u tel que $x \in L_1$ et $y \in L_2$ et $z \in L_3$, sinon on verrait que la position entre y et z est 3-bonne et la position entre x et y est 2-bonne et donc la première position courante serait 1-bonne. Sinon, le tableau mène à des positions, qu'on énumère successivement comme en question précédente. Chaque position correspond à un préfixe x de u qui est accepté par 1 et qui se termine à une position 2-bonne. À chaque position, on sait que l'état initial de l'automate A_2 n'est pas envoyé à $\perp$ car la position est 2-bonne : on énumère donc les positions comme en question précédente. Pour chaque x, ceci donne des y tel que pour chaque y, on sait que y est accepté par A_2 et la position de fin est 3-bonne, ainsi on a $u = xyz$ avec z accepté par A_3.

Cet algorithme énumère donc les facteurs satisfaisant la contrainte demandée avec une complexité analogue à celle de la question précédente : $O(nK)$ dans la phase de précalcul et $O(m)$ dans la phase d'énumération, où m est le nombre de résultats, n la longueur du mot, et K le nombre total de transitions de tous les automates.

Références

[1] D. Gusfield. *Algorithms on Strings, Trees and Sequences.* Cambridge University Press, 1997.