

Les énoncés des travaux pratiques seront disponibles sur internet quelques jours avant la séance correspondante, à l'adresse indiquée ci-dessus. Les questions en rapport avec les TP, Caml Light ou l'informatique sont les bienvenues et peuvent être envoyées à [victor.nicollet@ens.fr](mailto:victor.nicollet@ens.fr)

La recherche de motifs consiste à trouver un texte court (le motif) dans un texte plus long. Il existe un algorithme naïf très simple qui consiste à essayer, pour chaque indice  $i$  dans le texte long, de voir si le motif se trouve dans le texte long à cette position.

▷ **Question 1.** Écrire la fonction utilisant l'algorithme naïf pour chercher un motif dans une chaîne. L'utiliser pour trouver "tatare" dans le texte "tardive taxonomie tatare". Quelle est la complexité de cet algorithme? ◀

## 1 KNUTH–MORRIS–PRATT

---

On sait déjà qu'on peut utiliser des automates pour reconnaître un mot dans un texte en temps constant, par exemple en recherchant le premier passage de l'automate reconnaissant ".\*tatare" par son état final.

▷ **Question 2.** Construire un automate non-déterministe reconnaissant les mots qui se finissent par "tatare", puis le déterminer. ◀

Une fois déterminisé, l'automate permet de calculer en temps  $O(n)$  l'existence d'un motif dans un texte (puisque chaque transition se fait en temps constant). Il reste néanmoins le problème de la taille de l'automate déterminisé.

▷ **Question 3.** Montrer que cette déterminisation produit un automate ayant  $n+1$  états, où  $n$  est la longueur du suffixe reconnu.

Montrer que l'on peut reconfigurer l'automate déterministe en un automate à  $\varepsilon$ -transitions où chaque état (sauf le premier et le dernier) ont exactement deux transitions sortantes, dont une qui est étiquetée par une lettre. ◀

On utilise l'approche suivante: lorsqu'on lit une lettre, si elle permet de suivre la flèche étiquetée, alors on la suit. Sinon, on suit la seule  $\varepsilon$ -transition sortante ou, si on est sur l'état initial, on reconnaît n'importe quoi en y restant. Cela permet d'avoir une complexité  $O(n)$  en mémoire et en temps.

▷ **Question 4.** On suppose que l'on se donne un automate comme une chaîne de caractères représentant le motif, et un tableau d'indices indiquant la destination de chacune des  $\varepsilon$ -transitions. Programmer une fonction `kmp_chercher m a t` qui utilise l'automate pour déterminer la présence d'un motif dans un texte. Prouver que la complexité est linéaire. ◀

Il reste une seule zone d'ombre: la construction de l'automate à partir du motif.

▷ **Question 5.** Supposons l'automate construit, et utilisons cet automate sur le motif privé de sa première lettre. Quel est le rapport entre l'état dans lequel se trouve l'automate après avoir lu  $k$  lettres, et la destination de la transition issue de l'état  $k+1$  ?

Réutiliser la fonction de reconnaissance pour écrire la fonction de création de l'automate. Quelle est sa complexité?

En déduire une fonction `kmp m t` qui cherche un motif dans un texte, en utilisant les fonctions de création et de reconnaissance. ◀

## 2 RABIN-KARP

---

On considère maintenant, lorsque  $t$  est un mot, le mot  $T_m$  défini comme:

$$T_m[i] = (t[i] \dots t[i + m - 1])$$

Autrement dit, les lettres de  $T_m$  sont des facteurs de longueur  $m$  du mot  $t$  (et qui se chevauchent les uns les autres). Dans ces conditions, déterminer si un motif  $M$  de longueur  $m$  apparaît dans  $t$ , il suffit de déterminer si  $T_m$  contient la lettre  $M$ .

▷ **Question 6.** Écrire une fonction `rk_applique a m t` qui applique la fonction `a` sur tous les facteurs de longueur `m` du mot `t`, dans l'ordre croissant. Elle renvoie `true` si `a` a renvoyé `true` sur l'un des facteurs. ◁

La méthode naïve est donc un cas particulier de cette approche. Cependant, pour gagner en performances, on va utiliser une *fonction de hachage* pour comparer les chaînes plus facilement. On associe au mot  $t[i] \dots t[j - 1]$  la clé:

$$\text{hash}(t[i] \dots t[i + m - 1]) \equiv \sum_{k=1}^m A^k t[i + m - k][q] \quad A = 256 \quad q = 524287$$

Cette fonction permet d'associer des entiers aux mots d'une manière qui minimise le risque de collision.

▷ **Question 7.** Écrire une fonction `hash s i j` qui calcule la clé associée à  $t[i] \dots t[j - 1]$ . Quelle est la complexité de cette fonction? ◁

Cette approche n'apporte néanmoins pas d'amélioration si l'on doit calculer la clé associée à chaque facteur. Néanmoins, on observe qu'il existe une relation permettant de calculer la clé d'un facteur en temps constant à partir de la clé du facteur précédent.

▷ **Question 8.** Expliciter cette relation mathématiquement, puis écrire une fonction `transition` qui effectue ce calcul.

En déduire une fonction `rk_egal m` qui construit une fonction utilisable par `rk_applique`. La fonction renvoyée doit renvoyer `true` lorsque `m` est reconnu. Toutes les exécutions de cette fonction doivent se faire en temps  $O(1)$ , sauf la première qui peut s'effectuer en temps  $O(m)$ .

En déduire une fonction `rk` qui recherche un motif dans un texte. ◁