Contrôle continu : Algorithmique du texte (2022-2023)

Utilisation de la table des suffixes

À faire en monôme ou en binôme et à rendre le vendredi 30 novembre 2022

Vous déposerez vos travaux (votre code, un rapport au format PDF comprenant notamment le résultat de vos expérimentations, et enfin un fichier README.txt qui donne la marche à suivre pour lancer vos programmes) sur ecampus sous la forme d'une archive.

Indications :

- Il ne vous est pas demandé d'implémenter le calcul de la table des suffixes et la table des hauteurs. Le code python à cette fin vous est fourni à l'URL **https://clementjo1.users.greyc.fr/algo-texte/scripts/tools_karkkainen_sanders.py** (c'est le même code que celui utilisé au TP3 sur ecampus).
- Toutes les données pour tester vos algorithmes se situent à l'URL **https://clementjo1.users.greyc.fr/algo-texte/data/**. Les sources des données sont citées pour information.
- **Attention !** Pour l'exercice 1 : le caractère '@' n'est pas inférieur aux caractères de ponctuation en ASCII (et donc la même chose en UTF8). Le plus petit caractère "affichable" est l'espace. La position du séparateur n'est donc pas `TS[0]` (la valeur de la table des suffixes à l'indice 0).
- On veillera à organiser le code « logiquement ». Les fonctions doivent être commentées à minima (description de l'entrée et de la sortie, nom de la fonction en rapport avec les opérations effectuées).

Exercice I : plus longs facteurs communs à deux textes

Dans le cadre de la détection de plagiat, on souhaite localiser les plus longs facteurs communs entre plusieurs textes. *Uun plus facteur commun est un facteur apparaissant dans chacun des textes de longueur maximale*. (N.B. : un plus long facteur commun est également parfois appelé une plus longue sous-chaîne commune).

Commençons par un exemple. Supposons que l'on souhaite déterminer les plus longs facteurs communs à deux textes $t_1 =$ `bcabbcab`, $t_2 =$ `caabba`.

L'algorithme commence par calculer les tables `TS` et `HTR` sur le texte t obtenu en concatènant les chaînes t_1 et t_2 en les séparant avec un marqueur distinct des autres caractères (ici '@') appelé séparateur. On a

$$t = \texttt{bcabbcab@caabba}.$$

Déterminer les plus longs facteurs communs à t_1 et t_2 consiste à trouver des paires d'indices consécutive $(i, i+1)$ telle que les suffixes correspondants dans t débutent avant et après le séparateur dans les deux chaînes et que le préfixe commun soit de longueur maximale.

i	$t[\texttt{TS}[i]:]$	$\texttt{TS}[i]$	$\texttt{HTR}[i]$
0	`@caabba`	8	0
1	`a`	14	0
2	`aabba`	10	1
3	`ab@caabba`	6	1
4	`abba`	11	2
5	`abbcab@caabba`	2	3
6	`b@caabba`	7	0
7	`ba`	13	1
8	`bba`	12	1
9	`bbcab@caabba`	3	2
10	`bcab@caabba`	4	1
11	`bcabbcab@caabba`	0	4
12	`caabba`	9	0
13	`cab@caabba`	5	2
14	`cabbcab@caabba`	1	3

Question 1. Quelle est l'utilité d'ajouter un séparateur '@' ?

Question 2. Comment repérer que le i-ème suffixe dans l'ordre lexicographique (i.e., en notation python $t[\texttt{TS}[i]:]$) commence dans t_1 ou t_2 ?

Question 3. En examinant les tables `TS` et `HTR` de l'exemple, justifier pourquoi le plus long facteur commun à t_1 et t_2 est `abb` relatif à la paire d'indices $(4, 5)$ et non pas le facteur plus long `bcab` relatif à la paire $(10, 11)$.

Question 4. Appliquer l'algorithme pour la paire de textes $t_1 =$ `cadad` et $t_2 =$ `cd`. Donner la table des suffixes `TS` et la table des hauteurs `HTR`. Expliquer comment trouver les deux plus facteurs communs les plus longs $\{$`c`, `d`$\}$ grâce à ces tables.

Question 5. Implémenter l'algorithme pour détecter les plus longs facteurs communs entre *deux* textes.

Question 6. Évaluer la complexité de votre algorithme.

Question 7. Tester votre algorithme sur plusieurs romans. Nous mettons à votre disposition divers textes (avec un pré-traitement : encodage en utf8 quand nécessaire, suppression des retours à la ligne) à l'URL **`https://clementj01.users.greyc.fr/algo-texte/data/`**.

Voici cinq romans (tous téléchargés sur le site du projet Gutenberg) :

a. `"dumas_le_vicomte_de_bragelonne.utf8.txt"` (1,1M) : « Le Vicomte de Bragelonne » de Alexandre Dumas (père)

b. `"hugo_notre_dame_de_paris.utf8.txt"` (1023K) : « Notre dame de Paris » de Victor Hugo

c. `"proust_du_cote_de_chez_swann.utf8.txt"` (1023K) : « Du côté de chez Swann » de Marcel Proust

d. `"zola_la_bete_humaine.utf8.txt"` (764K) : « La bête humaine » d'Émile Zola

e. `"zola_le_ventre_de_paris.utf8.txt"` (672K) : « Le ventre de Paris » d'Émile Zola

Pour chaque paire de romans (il y a 10 paires différentes), indiquer
— le temps de calcul du plus long facteur commun (conseil : utiliser `pypy3`);
— les plus longs facteurs communs ainsi que leur longueur.

Question 8. Voici trois extraits de séquences ADN (tirées du site `https://www.ncbi.nlm.nih.gov/assembly/`) :

a. `"e-coli.ascii.txt"` (4,5M) : séquence ADN de la bactérie Escherichia coli

b. `"Mus_musculus.GRCm38.dna.chromosome.1.part00.txt"` (46M) : premier quart de la séquence ADN du premier chromosome de la souris

c. `"Homo_sapiens.GRCh38.dna.chromosome.1.part00.txt"` (55M) : premier quart de la séquence ADN du premier chromosome de Homo sapiens

Pour chaque paire de séquences ADN (il y a 3 paires différentes), indiquer
— le temps de calcul du plus long facteur commun (conseil : utiliser `pypy3`);
— les plus longs facteurs communs ainsi que leur longueur.

Exercice II : Algorithme de recherche

Question 9. Soit le texte `TGCTGTTTGCTG`.

a. Construire sa table des suffixes `TS` et sa table `HTR`.

b. Déterminer également, l'inverse de sa table des suffixes. Donner le code correspondant et la complexité du calcul de cette table.

Question 10. Implémenter l'algorithme élémentaire de recherche d'un mot avec la table des suffixes.

Question 11. Tester sur des exemples simples. Déterminer le nombre d'occurrences des personnages suivants `"Anna Pavlovna"`, `"Countess Rostova"`, `"Anna Mikhaylovna"`, `"Prince Vasili"`, `"Marya Dmitrievna"`, `"Peter Nikolaevich"`, `"Vladimirovich Bezukhov"`, `"Napoleon"`, `"Veronique Terrier"` dans le texte `"Tolstoy_war_and_peace.ascii.txt"`.

Question 12. Proposer une version améliorée de l'algorithme de recherche d'un mot.

Question 13. Comparer les deux versions de l'algorithme de recherche sur des exemples représentatifs et évaluer les temps d'exécution obtenus (attention, ne pas prendre en compte le temps de construction de la table des suffixes).

Question 14. On suppose que la table des suffixes du texte est précalculée, que la complexité en moyenne de la recherche (améliorée) d'un motif x de taille m dans un texte de longueur n est $O(m \log n)$.

Évaluer la complexité de l'algorithme recherchant successivement k motifs $x_1, \ldots, x_k$ où x_i est de longueur $|x_i| = m_i$ $(1 \leq i \leq k)$ dans le même texte de longueur n ?