Lycée Louis-Le-Grand, Paris 2014/2015
MPSI 4 – Informatique pour tous
D. Cazor, A. Troesch

TP n° 7 : Recherche d'un mot dans un texte

Problème –
Le but de ce problème est d'étudier des algorithmes de recherche de mots (séquences) dans un texte donné. Dans tout l'énoncé, les variables `texte` et `mot` désigneront deux chaînes de caractères. Notre but est de trouver une ou plusieurs occurrences de `mot` dans `texte`. Nous dirons que `mot` a une occurrence dans `texte` à la place i si pour tout $k \in [\![0, \text{len}(\texttt{mot}) - 1]\!]$, on a l'égalité des caractères `texte[`$i+k$`]` = `mot[`k`]`.

Partie I – Méthode directe de recherche d'un mot dans un texte

1. Écrire une fonction `isprefixe` prenant en paramètre `texte`, `mot` et un entier i, et testant si `mot` est préfixe de `texte[i:]`, autrement dit si `mot` a une occurrence dans `texte` à la place i. La fonction renverra un booléen. On ne s'autorisera pour ce faire que des comparaisons lettre à lettre.
2. Utiliser la fonction `isprefixe` pour écrire une fonction `cherche_occurrence` donnant sous forme d'un tableau la liste des occurrences de `mot` dans `texte`. Les occurrences peuvent se chevaucher. Ainsi, dans « bonbonbon », il y a deux occurrences de « bonbon ».

Partie II – Utilisation et test de rapidité de la fonction `cherche_occurrence`

1. Écrire une fonction `texte_alea` prenant en paramètre un entier N, et retournant une chaîne de caratères aléatoire de longueur N, composée des 26 lettres de l'alphabet.
2. Estimer le nombre moyen d'occurrences du mot « llg » dans un mot aléatoire de longueur 10000
3. Estimer la probabilité qu'un mot de longueur 10000 contienne au moins une occurrence du mot « llg »
4. Écrire une fonction `teste_direct` prenant en argument trois entiers N, n et p, choisissant une fois pour toutes un texte de longueur N et répétant p fois l'opération consistant à créer un mot aléatoire de longueur n et à rechercher ses occurrences dans `texte`. La fonction doit renvoyer la durée de ces p recherches. La durée de création de la chaîne `texte` ne doit pas être prise en compte.
5. Tester votre fonction avec $N = 10000$, $n = 5$ et $p = 100$, puis $N = 20000$, $n = 5$ et $p = 100$. Quel commentaire pouvez-vous faire ?

Partie III – Utilisation d'une table des suffixes

On se propose, moyennant un traitement préalable du texte `texte`, d'accélérer la recherche d'un mot dans `texte`. Ce traitement préalable consiste en la création d'un tableau des suffixes. Les suffixes de `texte` sont toutes les chaînes terminales `texte[i:]`, pour $i \in [\![0, \texttt{len(texte)} - 1]\!]$. La table des suffixes consiste en la liste des suffixes de `texte` rangés par ordre alphabétique. Afin de ne pas utiliser trop de mémoire, ces suffixes sont représentés dans le tableau par leur indice initial dans `texte`. Par exemple, pour le texte 'abracadabra', les suffixes sont, rangés dans l'ordre croissant :

['a','abra','abracadabra','acadabra','adabra','bra','bracadabra','cadabra','dabra','ra','racadabra']

La table des suffixes sera alors la liste des indices initiaux de ces suffixes, à savoir :

$$[10, 7, 0, 3, 5, 8, 1, 4, 6, 9, 2].$$

1. À l'aide de la méthode `sort` associée à la classe `list`, et à une clé de tri bien choisie, écrire une fonction prenant en argument un texte, et renvoyant la table des suffixes associée.

2. Écrire une fonction `recherche_dichotomique` prenant en paramètres un tableau `T`, une fonction `cl` et un objet a comparable aux éléments `cl(T[i])`, et, supposant le tableau trié dans l'ordre croissant suivant la clé `cl`, retournant le plus petit indice i tel que $a \leqslant$ `cl(T[i])`. On précèdera pour cela par dichotomie.
3. Utiliser la fonction précédente pour écrire une fonction `recherche_par_suffixes` prenant en argument `texte`, sa table de suffixes (préalablement calculée) `suffixes` et `mot`, et retournant la liste des places des occurrences de `mot` dans liste.
4. Comparer les fonctions des parties 1 et 3 pour la recherche de 'ab' dans 'abracadabra'. Commentaire ?
5. Écrire une fonction `teste_recherche_par_suffixes`, prenant en argument trois entiers N, n et p, choisissant une fois pour toutes un texte de longueur N, et créant sa table des suffixes, et répétant p fois l'opération consistant à créer un mot aléatoire de longueur n et à rechercher ses occurrences dans `texte`. La fonction doit renvoyer la durée de ces p recherches. La durée de création de la chaîne `texte` et de la table des suffixes ne doit pas être prise en compte.
6. Tester votre fonction avec $N = 10000$, $n = 5$ et $p = 100$, puis $N = 20000$, $n = 5$ et $p = 100$, puis poussez un peu plus loin. Commentaires ?
7. Quelle est la complexité en temps de la recherche par cette méthode, une fois la table des suffixes crée ?

Ainsi, le traitement préalable de la table des suffixes ralentit globalement le temps de recherche, si le but est de faire une seule recherche (ce tri initial ne pouvant s'effectuer en moins de $\Theta(n\ln(n))$ en moyenne, alors que l'algorithme initial est linéaire). En revanche, une fois la table créée, la recherche est beaucoup plus rapide (logarithmique). Ainsi, la création de la table est intéressante si on a un grand nombre de recherches à effectuer dans un même texte. Cette idée est à la base des méthodes de recherche sur internet : les tables de suffixes des textes disponibles sur internet sont créées une fois pour toutes (indépendamment des demandes de recherche, et avec actualisations régulières), et prêtes à l'emploi au moment ou l'on souhaite effectuer une recherche. Ainsi, du point de vue de l'utilisateur, la recherche se fait en temps logarithmique.

Partie IV – Graphiques

Tracer sur deux graphes séparés les graphes du temps de réponse en fonction de N de chacun des deux algorithmes pour la recherche d'un mot aléatoire de longueur 5 dans un texte aléatoire de longueur N. On utilisera le module `matplotlib.pyplot`, dont certaines fonctionnalités sont décrites dans le chapitre 2 de votre cours. On fera varier N de 100 à 50000 par pas de 200.