TD2 : Rechercher un mot dans un texte et compter ses occurrences

(d'après Polytechnique PSI-PT 2010 info)

La quantité d'information disponible en ligne s'est considérablement accrue. Notons par exemple la numérisation électronique de nombreux ouvrages, le développement exponentiel du web ou la mise à disposition de données plus spécialisées comme le génome humain. Un problème clé lié à ce volume d'information est l'efficacité de la recherche d'informations. Dans un moteur de recherche internet par exemple, un facteur d'importance d'une page pour une requête donnée est l'appartenance des mots recherchés à la page ainsi que leur nombre d'apparitions. Ces informations peuvent être obtenues simplement en parcourant le texte, comme nous le verrons dans une première partie. Toutefois, cette approche simple conduit à des algorithmes lents vu la taille des textes considérés. La suite du problème propose des réalisations plus efficaces.
Dans tout le problème, nous supposerons que le texte est une chaîne de caractères de taille n, écrite en minuscule, sans accent et sans espaces.

Partie I : méthode directe

Dans cette partie, nous allons mettre en œuvre des algorithmes simples permettant d'effectuer les opérations de recherche citées précédemment.
Introduisons d'abord quelques notions. Un *mot* est, dans notre contexte, tout simplement un texte. Un mot `mot` de taille `m` apparaît dans le texte `txt` de taille `n`, si et seulement si il existe un texte extrait de `txt` qui est égal à `mot`, caractère pour caractère. Le *suffixe* numéro `k` du texte `txt` de taille `n` est le texte extrait `txt[k:n]`. On note que le mot `mot` apparaît dans le texte `txt` si et seulement si il existe un suffixe tel que mot apparaît en tête de ce suffixe (`mot` et `txt[k,k+m-1]` sont égaux, caractère pour caractère).

1. Écrire une fonction `enTeteDeSuffixe(mot:str,txt;str,k:int)->bool` qui renvoie `True` si le mot `mot` apparaît en tête du suffixe numéro `k` du texte `txt`, et `False` sinon. On pourra supposer que `k` est un indice valide du texte `txt`.
2. Écrire une fonction `rechercherMot(mot:str,txt:str)->bool` qui renvoie `True` si le mot `mot` apparaît dans le texte `txt`, et `False` sinon.

Tester l'apparition d'un mot dans un texte ne suffit pas toujours, nombre de moteurs de recherche internet prennent en compte le nombre d'occurrences des mots recherchés dans une page donnée. Nous considérons le nombre d'occurrences avec recouvrement autorisé, qui est la notion la plus simple : on compte le nombre de répétitions du mot dans le texte, sans contrainte aucune. Par exemple, dans le texte « quelbonbonbon » (quel bon bonbon) le nombre d'occurrences de `'bonbon'` est 2, même si ces occurrences se recouvrent.

				b	o	n	b	o	n			
							b	o	n	b	o	n
q	u	e	l	b	o	n	b	o	n	b	o	n

3. Écrire une fonction `compterOccurrences(mot:str,txt:str)->int` qui renvoie le nombre d'occurrences de `mot` dans le texte `txt`.

Nous allons maintenant calculer l'ensemble des mots d'une taille donnée qui apparaissent dans le texte ainsi que leur fréquence. Nous n'abordons que les tailles 2. Dans l'exemple précédent, pour la taille 2, on obtient bo(3), el(1), lb(1), nb(2), on(3), qu(1), ue(1).

4. Écrire une fonction `afficherFrequenceBigramme(txt:str)->dict` qui détermine les mots de 2 lettres présents dans le texte ainsi que leur nombre d'occurrences. La fonction doit renvoyer un dictionnaire dont les clefs sont les mots de 2 lettres présents dans le texte et dont les valeurs associées sont leur nombre d'occurrences.

Partie II : tableau des suffixes

Afin d'accélérer les fonctions précédentes, nous allons utiliser des algorithmes basés sur le tableau des suffixes, défini comme regroupant les suffixes du texte pris dans l'ordre du dictionnaire (dit ordre *lexicographique*).
Étant donné un texte `txt` de taille `n`, un indice `k` suffit à désigner un suffixe de `txt` comme le texte extrait `txt[k:n]`. Le tableau des suffixes `tabS` sera donc représenté en machine comme un tableau d'indices de `txt`. Par exemple, en prenant le texte « quelbonbonbon », on obtient les classements suivants

Suffixes classés selon leur premier indice

0	quelbonbonbon
1	uelbonbonbon
2	elbonbonbon
3	lbonbonbon
4	bonbonbon
5	onbonbon
6	nbonbon
7	bonbon
8	onbon
9	nbon
10	bon
11	on
12	n

Suffixes classés par ordre lexicographique

10	bon
7	bonbon
4	bonbonbon
2	elbonbonbon
3	lbonbonbon
12	n
9	nbon
6	nbonbon
11	on
8	onbon
5	onbonbon
0	quelbonbonbon
1	uelbonbonbon

La première colonne du classement de droite donne la liste `tabS`, c'est-à-dire : `[10,7,4,2,3,12,9,6,11,8,5,0,1]`.
Nous avons déjà remarqué (Partie I) que le mot mot apparaît dans le texte `txt`, si et seulement si mot apparaît en en tête d'un suffixe de `txt`. Or le tableau `tabS` est le tableau *trié* des suffixes du texte, ce qui nous permet d'utiliser la technique de recherche *dichotomique*.

1. Écrire une fonction `rechercherMot2(mot:str,txt:str,tabS:list)->bool` qui renvoie `True` si le mot mot apparaît dans le texte `txt`, et `False` sinon. On impose évidemment l'emploi de la technique de recherche dichotomique dans le tableau des suffixes `tabS`, que l'on suppose correct.
2. Donner un ordre de grandeur de la complexité de la fonction `rechercherMot` en fonction de la taille n de `txt` (Partie I) et par `rechercherMot2` (Partie II). Expliquer ensuite l'intérêt du tableau des suffixes, dans le cas d'un moteur de recherche internet.
3. Écrire une fonction `compterOccurrences2(mot:str,txt:str,tabS:list)->int` qui renvoie le nombre d'occurrences de mot dans le texte `txt` ; on utilisera à nouveau une recherche dichotomique et le tableau des suffixes `tabS`.
4. Écrire une fonction `triSuffixes(txt:str)->list` qui prend en argument un texte `txt` et qui renvoie la liste `tabS` définie comme précédemment ; on s'inspirera du tri par insertion.