Lycée Louis-Le-Grand, Paris
MPSI 4 – Informatique pour tous
A. Troesch, J.-P. Becirspahic

2013/2014

TP 6 : recherche d'un mot dans un texte

d'après l'épreuve de polytechnique PSI et PT 2010.

Ces dernières années, la quantité d'information numérique disponible s'est considérablement accrue : numérisation d'ouvrages, développement du web, données spécialisées (le génome humain, par exemple), etc. Un problème clé lié à cette masse considérable de données est la recherche efficace d'une information en son sein. Dans un moteur de recherche, par exemple, un facteur d'importance d'une page pour une requête donnée est l'appartenance des mots recherchés à cette page, ainsi que leur nombre d'apparitions. Ces informations peuvent être obtenues simplement en parcourant le texte, mais cette approche conduit à des algorithmes lents compte tenu de la taille des textes considérés ; nous allons voir qu'il existe des démarches plus efficaces.

1. Méthode directe

Durant tout ce TP, nous allons considérer deux variables nommées respectivement `texte` et `mot` et contenant toutes deux une chaîne de caractères. Notre objectif est d'écrire deux fonctions :
– la première déterminant si `mot` se trouve dans `texte` ;
– la seconde dénombrant le nombre d'apparitions de `mot` dans `texte`.
Par exemple, si `texte = 'quelbonbonbon'` et `mot = 'bonbon'` la première fonction doit retourner le booléen `true` et la seconde l'entier 2.

Avant de commencer, rappelons quelques notions utiles sur les chaînes de caractères :
– `len(texte)` renvoie la longueur de la chaîne de caractères `texte` ;
– `texte[k]` renvoie le $(k+1)^e$ caractère de la chaîne `texte` (les caractères des chaînes sont donc indexés entre `0` et `len(texte) - 1`) ;
– `texte[i:j]` renvoie la sous-chaîne des caractères dont les index sont compris entre i (inclus) et j (exclus). En particulier, on appellera *suffixe de rang i* la sous-chaîne `texte[i:]` (lorsque l'indice j est omis, il est implicitement pris égal à la longueur de la chaîne).

a. Écrire une fonction `enTeteDeSuffixe(mot, texte, k)` qui renvoie le booléen `True` si `mot` apparaît en tête du suffixe de rang `k` de `texte`, et `False` sinon.

b. En déduire une fonction `rechercherMot(mot, texte)` qui renvoie `True` si `mot` apparaît dans `texte`, et `False` sinon.

c. Écrire enfin une fonction `compterOccurrences(mot, texte)` qui renvoie le nombre d'occurrences de `mot` dans `texte`.

Fréquence d'apparition

Nous allons maintenant calculer l'ensemble des mots d'une taille donnée qui apparaissent dans le texte ainsi que leur fréquence. Pour le moment nous n'aborderons que les mots de taille 1 et 2.
Dans l'exemple précédent, pour la taille 1 on obtient les mots b(3), e(1), l(1), n(3), o(3), q(1), u(1) et pour la taille 2 les mots bo(3), el(1), lb(1), nb(2), on(3), qu(1), ue(1).

• Dictionnaires

Pour représenter cette information nous utiliserons un *dictionnaire* (qu'on appelle aussi une *table d'association*). Il s'agit d'une structure de données qui associe une valeur à chaque élément d'un ensemble de clés. Dans le cas qui nous intéresse, les clés seront des chaînes de caractères et les valeurs, des entiers. Voici un exemple de dictionnaire :

```
>>> d = { 'un': 1, 'deux': 2, 'trois': 3}
```

Les fonctions principales attachées à un dictionnaire `d` sont les suivantes :

- `d[k]` renvoie la valeur associée à la clé `k` (ou déclenche l'exception `KeyError` si la clé ne se trouve pas dans `d`) ;
- `d[k] = v` associe la clé `k` à la valeur `v` (et supprime une éventuelle association précédente de `k`) ;
- `del d[k]` supprime la clé `k` du dictionnaire (ou déclenche l'exception `KeyError` si la clé n'est pas présente) ;
- `k in d` renvoie un booléen traduisant la présence ou non de la clé `k` dans `d`.

Par exemple :

```
>>> d['deux']
2
>>> d['quatre'] = 4
>>> del d['trois']
>>> 'trois' in d
False
>>> d
{'quatre': 4, 'deux': 2, 'un': 1}
```

D'autres fonctions et méthodes existent mais celles-ci sont les principales (voir l'aide en ligne pour plus d'information).

d. Écrire une fonction `frequenceLettre(texte)` qui calcule et retourne le dictionnaire des fréquences des différentes lettres présentes dans le texte.

e. Écrire de même une fonction `frequenceBigramme(texte)` qui calcule et retourne le dictionnaire des fréquences des différents mots de deux lettres présents dans le texte.

2. 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 lexicographique.
Reprenons l'exemple du texte `quelbonbonbon`. Les différents suffixes sont désignés par leurs rangs :

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

Trions maintenant ces différents suffixes 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

Le *tableau des suffixes* de `quelbonbonbon` est le tableau `tabS = [10, 7, 4, 2, 3, 12, 9, 6, 11, 8, 5, 0, 1]`.

Il existe des méthodes très efficaces pour calculer le tableau des suffixes, mais nous nous contenterons d'une méthode simple utilisant les méthodes de tri existants en PYTHON.

• Trier une liste PYTHON

Lorsque les éléments d'une liste peuvent être ordonnés (nombres, chaînes de caractères, etc.), la méthode `sort()` permet de les ranger par ordre croissant (en modifiant la liste initiale) [1]. Il est par exemple très facile de trier les éléments d'une liste par ordre lexicographique :

1. il existe aussi une fonction `sorted` qui crée une nouvelle liste ordonnée.

```
>>> lst = ['Blandine', 'Alexis', 'Lucas', 'Aude']

>>> lst.sort()

>>> lst
['Alexis', 'Aude', 'Blandine', 'Lucas']
```

Le paramètre optionnel `key` permet de préciser une fonction à appliquer à chaque élément avant le tri : dans ce cas, les éléments de la liste seront triés suivant les résultats de cette fonction. Par exemple, pour trier la liste précédente suivant la longueur des différents prénoms qui la composent, il suffit d'écrire :

```
>>> lst.sort(key = len)

>>> lst
['Aude', 'Lucas', 'Alexis', 'Blandine']
```

En observant que calculer le tableau des suffixes d'une chaîne de caractères `texte` revient à trier le tableau $[0, 1, 2, \ldots, n-1]$ suivant une fonction-clé bien choisie, écrire une fonction `calculerSuffixes(texte)` qui renvoie le tableau des suffixes de la chaîne de caractères `texte`.
On désignera désormais par `tabS` ce tableau des suffixes de `texte`.

3. Exploitation du tableau des suffixes

Nous avons déjà pu observer dans la première partie que `mot` apparaît dans `texte` si et seulement si `mot` apparaît en tête d'un suffixe de `texte`. Or le tableau `tabS` est le tableau *trié* des suffixes, ce qui va nous permettre d'utiliser la technique de la recherche dichotomique.

a. Tout d'abord, nous allons avoir besoin d'une fonction de comparaison entre mot et suffixe qui élargit le cas d'égalité par rapport à une pure comparaison.
   Écrire une fonction `comparerMotSuffixe(mot, texte, k)` qui renvoie l'entier :
   - 0 si `mot` apparaît en tête du suffixe de rang k de `texte` ;
   - -1 lorsque `mot` précède le suffixe de rang k suivant l'ordre lexicographique ;
   - 1 lorsque `mot` suit le suffixe de rang k suivant l'ordre lexicographique.

b. Écrire alors une fonction `rechercherMot2(mot, texte, tabS)` qui renvoie `True` si `mot` apparaît dans `texte`, et `False` sinon. Cette fonction devra utiliser la technique de la recherche dichotomique dans le tableau des suffixes `tabS`.

c. Quel est l'ordre de grandeur du nombre de comparaisons de mots effectués par `rechercherMot` (question 1.b) et `rechercherMot2` (question3.b) ? Quel est selon vous l'intérêt du tableau des suffixes dans le cas d'un moteur de recherche internet ?

4. Nombre d'occurrences d'un mot dans un texte

Pour terminer, nous allons montrer que le tableau des suffixes permet aussi de dénombrer les apparitions d'un mot dans un texte.

a. Écrire une fonction `rechercherPremierSuffixe(mot, texte, tabS)` qui renvoie le plus petit indice i de `tabS` tel que `mot` apparaît en tête du suffixe de rang `tabS[i]` de `texte`. Si `mot` m'apparaît pas dans `texte`, la fonction devra retourner -1.
   Il faudra bien entendu adapter la technique de recherche dichotomique dans le tableau des suffixes `tabS`.

b. Écrire de même une fonction `rechercherDernierSuffixe(mot, texte, tabS)` qui renvoie le plus grand indice j tel que `mot` apparaît en tête du suffixe de rang `tabS[j]` de `texte`.

c. En déduire une fonction `compterOccurrences2(mot, texte, tabS)` qui renvoie le nombre d'occurrences de `mot` dans `texte`.

Nous revenons enfin sur le calcul des sous-mots possibles abordés à la fin de la première partie et traitons la question dans toute sa généralité.

d. Écrire une fonction `afficherFrequenceKgramme(texte, tabS, k)` qui retourne le dictionnaire des mots de k lettres présents dans le texte ainsi que leur fréquence, en exploitant judicieusement le tableau des suffixes.