Recherche d'un mot dans un texte

Devoir commun ITC 13/01/2023

MPSI - PCSI

Recommandations générales

- Dans tout le problème, seules les fonctions *usuelles* du langage python présentées en cours et en tp sont autorisées.
- Les fonctions écrites devront être commentées, avec une couleur différente de celles des instructions.

Un problème clé lié au volume d'informations numériques (sur le web par exemple) est l'efficacité de la recherche d'un mot dans une page ainsi que son 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.

Introduisons d'abord quelques notions élémentaires :

- Un mot et un texte sont des chaînes de caractères sans majuscules, accent, espace et ponctuation.
- Le 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. Par exemple le mot `'bons'` apparaît dans le texte `'desbonbons'`, mais pas dans le texte `'unbonbon'`.
- Le suffixe numéro k du texte `txt` de taille n est le texte extrait `txt`[k:n]. Par exemple le suffixe numéro 2 du texte `'unbonbon'` est le texte extrait `'bonbon'`.
- 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]` sont égaux, caractère pour caractère). Par exemple le mot `'bonb'` apparaît en tête du suffixe 2 du texte `'unbonbon'` (`mot` et `txt[2:2+4]` sont égaux sur cet exemple).

1. Quelques fonction de bases

Q1) On considère la fonction `present`(a, txt) suivante :

```
def present(a: str, txt : str) -> ... :
    n = len(txt)
    i = 0
    while i < n and txt[i] != a:
        i = i+1
    return i < n
```

a) On exécute la fonction `present('o', 'ecologie')` ; quel est le contenu de la variable `i` en sortie de boucle ? Que renvoie cette fonction ?

b) On exécute la fonction `present('f', 'ecologie')` ; quel est le contenu de la variable `i` en sortie de boucle ? Que renvoie cette fonction ?

c) À quoi correspond le résultat renvoyé par la fonction `present` ?

Q2) On souhaite écrire une fonction `occurence(a: str, txt: str)->int` *récursive* qui renvoie le nombre de fois où le caractère contenu dans la variable `a` apparaît dans la chaîne de caractères `txt` (et renverra 0 si `a` n'est pas présent dans `txt`). Pour cela, on peut, pour une chaîne `txt` non vide, comparer `a` avec le premier caractère de `txt`, et effectuer ensuite des appels récursifs en utilisant `txt` privé de son premier élément. Recopier et compléter le code suivant pour qu'il réponde au problème posé.

```python
def occurence(a: str, txt: str)->int:
    if txt == '':
        return .....
    else:
        if a == txt[0]:
            return .....
        else:
            return .....
```

Q3) Écrire une fonction `table(txt: str)->dict` qui renvoie un **dictionnaire** dont les clés sont les lettres présentes dans la chaîne de caractères `txt`, et les valeurs sont les occurrences de ce caractère.

Q4) On considère la fonction :

```python
def debut(mot: str, txt: str)->bool:
    n = len(mot)
    if n <= len(txt) and txt[:n] == mot :
        return True
    else:
        return False
```

À quelle condition cette fonction retourne `True` ?

2. Méthode directe

Dans cette partie, nous allons mettre en œuvre des algorithmes simples permettant d'effectuer les opérations de recherche citées précédemment.

Q5) On considère la fonction :

```python
def enteteDeSuffixe(mot: str, txt: str, k: int) -> .... :
    n, m = len(txt), len(mot)
    if k+m > n :
        return ....
    else:
        return debut(....,....)
```

Compléter la fonction `enTeteDeSuffixe(mot, txt, k)` pour qu'elle 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`.

Q6) Écrire une fonction `rechercherMot(mot: str, txt: str)->bool` qui renvoie `True` si le mot `mot` apparaît dans le texte `txt`, et `False` sinon (on pourra utiliser la fonction précédente).

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

Q7) Écrire une fonction `compterOccurrences(mot: str, txt: str) -> int` qui renvoie le nombre d'occurrences de `mot` dans le texte `txt`.

3. 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 l'indice du premier caractère

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 le tableau `tabS`, c'est-à-dire : [10,7,4,2,3,12,9,6,11,8,5,0,1] (pour cet exemple).

Q8) On considère la fonction suivante :

```
def inconnue(L: list)-> ... :
    n = len(L)
    R = []
    for i in range(n) :
        elt = L[i]
        j = i
        while (j > 0) and (elt < R[j-1]) :
            j = j-1
        R = R[:j] + [elt] + R[j:]
    return R
```

Donner les différents contenus de la variable R lors de l'exécution de `inconnue`(L) avec L=[10,8,12,4,5]. Quel est l'action de cette fonction ?

Q9) En s'inspirant de la fonction précédente, écrire une fonction `triSuffixes(txt: str)->list` qui retourne le tableau `tabS` des suffixes comme décrit précédemment. On appelle qu'en python, on peut comparer deux chaînes de caractères en utilisant les symboles habituels <, >, <=, >=, c'est l'ordre lexicographique qui est utilisé par python.

Q10) On désire écrire une fonction `rechercherMot2(mot: str, txt: str)->bool` qui renvoie `True` si le mot `mot` apparaît dans le texte `txt`, et `False` sinon.

a) En supposant le tableau des suffixes `tabS` connu, écrire le script d'une fonction **récursive** auxiliaire : `rec_aux(mot: str, txt: str, tabS: list, d: int, f: int)->bool` où `d` et `f` sont les indices de début et fin de la zone de recherche, c'est à dire du sous-tableau `tabS[d:f+1]` dans lequel la recherche dichotomique est en cours, cette fonction devant renvoyer `False` si `d > f` et `True` si `mot` est le début d'un des suffixes de `txt`.

b) Écrire une fonction `rechercherMot2(mot: str, txt: str)->bool` qui renvoie `True` si le mot `mot` apparaît dans le texte `txt`, et `False` sinon et qui :
   - affecte le tableau trié des suffixes de `txt` à la variable `tabS`,
   - utilise la méthode récursive de recherche dichotomique dans le tableau des suffixes `tabS`.

c) **Question bonus**. Écrire une fonction `rechercherMot3(mot: str, txt: str)->bool` qui réalise la même action que `rechercherMot2`, mais qui utilise une version itérative (c'est à dire non récursive) de l'algorithme de recherche dichotomique dans le tableau des suffixes.