Terminale - Spécialité NSI

TP - Recherche textuelle

Introduction

Le but de ce TP est de produire un algorithme performant permettant de rechercher un "motif" (mot) dans une "séquence" (du texte par exemple). Le but est d'indiquer la position du motif recherché. C'est par exemple le type de fonctionnalité utilisé lors de la recherche d'un élément sur une page web ou un document texte, avec **CTRL** + **F**.

Exercice 1 : Rechercher un caractère dans une chaîne de caractères *

Pour commencer, vous allez développer une fonction très simple permettant de trouver la position d'un caractère dans une chaîne de caractères. Pour rappel, les **chaînes de caractères** sont semblables à des **tuples**. On peut accéder à chaque caractère de la chaîne grâce à son indice (sa position dans la chaîne).

Par exemple :

```
exemple = "Hello world"
print(exemple[0]) #Affiche "H"
print(exemple[1]) #Affiche "e"
print(exemple[0:5]) #Affiche "Hello"
print(exemple[6:]) #Affiche "world!"
```

Développez et testez une fonction **rechercher_caractere(texte, caractere)** qui, à partir d'une chaîne de caractères "texte" et d'un caractère "caractère" renvoie la première position du caractère dans "texte", c'est à dire la position où il apparaît pour la première fois (dans le cas où il y aurait plusieurs fois le même caractère dans le texte...). Si le caractère n'apparaît jamais, la fonction renvoie -1.

```
def rechercher_caractere(texte, caractere):
"""
    texte : Le texte dans lequel on recherche un caractère.
    caractere : Le caractère qu'on recherche dans le texte.
    Renvoie la première position du caractère recherché, ou -1 si il
↪   n'apparaît jamais dans le texte.
    """


#Exemples de tests
print(rechercher_caractere("Hello world!", "o")) #Affiche 4
print(rechercher_caractere("Hello world!", "w")) #Affiche 6
```

```
print(rechercher_caractere("Hello world!", "d")) #Affiche 10
print(rechercher_caractere("Hello world!", "b")) #Affiche -1
```

Exercice 2 : Rechercher un motif dans une chaîne de caractères **

Maintenant, on souhaite rechercher un **motif** (mot) dans une chaîne de caractères. Le but est d'obtenir la position où débute le mot dans la chaîne (le texte).

L'algorithme ne diffère pas beaucoup du précédent, néanmoins, vous ne devez plus vérifier la présence d'un caractère à une position donnée, mais tout une chaîne de caractère.

Pour vous faciliter la tâche, pensez à utiliser le code permettant d'obtenir une sous partie de la chaîne de caractère, comme montré dans l'exemple au début de l'exercice 1.

Attention à ne pas dépasser la limite bornée par la taille la chaîne de caractère lors de vos itérations, dans votre boucle. Pour cela, vous devrez peut-être réfléchir à la borne adéquate pour votre boucle.

Développez et testez une fonction **rechercher_mot(texte, mot)** qui, à partir d'une chaîne de caractères "texte" et d'un mot "mot" renvoie la première position du mot (là où il débute) dans "texte", c'est à dire la position où il apparaît pour la première fois (dans le cas où il y aurait plusieurs fois le même mot dans le texte...). Si le mot n'apparaît jamais, la fonction renvoie -1.

```
def rechercher_mot(texte, mot):
"""
    texte : Le texte dans lequel on recherche un mot.
    mot : Le mot qu'on recherche dans le texte.
    Renvoie la première position du mot recherché, ou -1 si il n'apparaît
 ↪  jamais dans le texte.
    """


#Exemples de tests
print(rechercher_mot("Hello world!", "lo")) #Affiche 3
print(rechercher_mot("Hello world!", "world")) #Affiche 6
print(rechercher_mot("Hello world!", "or")) #Affiche 7
print(rechercher_mot("Hello world!", "banane")) #Affiche -1
print(rechercher_mot("CAAGTCGAATTGCATGCCGA", "TGCA")) #Affiche 10
print(rechercher_mot("Un algorithme glouton (greedy algorithm en anglais,
 ↪  parfois appelé aussi algorithme gourmand, ou goulu) est un algorithme qui
 ↪  suit le principe de faire, étape par étape, un choix optimum local, dans
 ↪  l'espoir d'obtenir un résultat optimum global. Par exemple, dans le
 ↪  problème du rendu de monnaie (donner une somme avec le moins possible de
 ↪  pièces), l'algorithme consistant à répéter le choix de la pièce de plus
 ↪  grande valeur qui ne dépasse pas la somme restante est un algorithme
 ↪  glouton. Dans les cas où l'algorithme ne fournit pas systématiquement la
 ↪  solution optimale, il est appelé une heuristique gloutonne. L'illustration
 ↪  ci-contre montre un cas où ce principe est mis en échec.", "monnaie"))
 ↪  #Affiche 288
```

Algorithme de Boyer-Moore-Horspool

L'algorithme de Boyer-Moore-Horspool permet d'optimiser la recherche d'un motif dans une chaîne de caractères sans passer systématiquement par chaque caractères dans le chaîne (du texte).

Exercice 3 : Table de sauts **

La table de saut est un **dictionnaire** où chaque caractère du motif (mot recherché) est associé à sa distance au **dernier caractère du motif**. On prend en compte **la dernière occurrence (position) de chaque caractère** (si il apparaît plusieurs fois...).

Attention, toutefois, **on ne prend pas en compte le dernier caractère** (décalage de 0).

Cette table sert à se décaler lors de la recherche, afin de faire coïncider les positions d'un caractère apparaissant à la fois dans le texte et dans le motif.

Pour générer cette table :

1. On calcule la taille **M** du mot.
2. On crée un dictionnaire **table**.
3. On parcours le mot jusqu'à l'avant dernier caractère, et pour chaque indice **i** (position) :
   (a) On associe, dans le dictionnaire **table**, le caractère situé à la position **i** dans le mot avec la valeur (M-i-1).
4. On retourne la table.

Développez et testez une fonction **generer_table_sauts(mot)** qui, à partir d'une chaîne de caractères "mot" effectue le pré-traitement de ce dernier et renvoie un **dictionnaire** représentant la **table de saut** associée à ce "mot", comme définie précédemment.

```
def generer_table_sauts(mot):
"""
    mot : Le mot à pré-traiter
    Renvoie un dictionnaire représentant la table de sauts associé au mot,
↪   pour ''algorithme de Boyer-Moore-Horspool.
    """


#Exemples de tests
print(generer_table_sauts("hello")) #Affiche {"h": 4, "e": 3, "l": 1}
print(generer_table_sauts("world")) #Affiche {"w": 4, "o": 3, "r": 2, "l" : 1}
print(generer_table_sauts("TGCA")) #Affiche {"T": 3, "G": 2, "C": 1}
print(generer_table_sauts("TCTACA")) #Affiche {"T": 3, "C": 1, "A": 2}
```

Exercice 4 : Algorithme de Boyer-Moore-Horspool ***

Maintenant que nous pouvons générer la table de saut, nous pouvons otpimiser la recherche textuelle en appliquant la méthode suivante (algorithme de Boyer-Moore-Horspool) :

1. On calcule la taille **N** du texte dans lequel on recherche le mot.

2. On calcule la taille **M** du mot recherché.
3. A l'aide de la fonction précédente, on calcule **T**, la table de sauts du mot recherché.
4. On initialise une variable **i**, à 0.
5. On parcours le texte tant que **i** <= **N-M** :
   (a) Si la portion du texte allant de **i** à **i+M** correspond au mot recherché, on retourne la position **i**.
   (b) Sinon, deux cas de figures :
      i. Le caractère situé à la position **i+M-1** dans le texte est présent dans la table de saut **T**, dans ce cas, on récupère la valeur associée à ce caractère et on décale en l'ajoutant à **i**.
      ii. Sinon, on décale en ajoutant **M** à **i**.
6. Si on sort de la boucle, alors le mot n'a pas été trouvé, on renvoie donc **-1**.

Développez et testez une fonction **boyer_moore_horspool(texte, mot)** qui, à partir d'une chaîne de caractères "texte" et d'un mot "mot" renvoie la première position du mot (là où il débute) dans "texte", c'est à dire la position où il apparaît pour la première fois (dans le cas où il y aurait plusieurs fois le même mot dans le texte...). Si le mot n'apparaît jamais, la fonction renvoie -1. La fonction devra appliquer **l'algorithme de Boyer-Moore-Horspool**.

```
def boyer_moore_horspool(texte, mot):
"""
    texte : Le texte dans lequel on recherche un mot.
    mot : Le mot qu'on recherche dans le texte.
    Renvoie la première position du mot recherché, ou -1 si il n'apparaît
↪   jamais dans le texte.
    Applique l'algorithme de Boyer-Moore-Horspool
    """


#Exemples de tests
print(boyer_moore_horspool("Hello world!", "lo")) #Affiche 3
print(boyer_moore_horspool("Hello world!", "world")) #Affiche 6
print(boyer_moore_horspool("Hello world!", "or")) #Affiche 7
print(boyer_moore_horspool("Hello world!", "banane")) #Affiche -1
print(boyer_moore_horspool("CAAGTCGAATTGCATGCCGA", "TGCA")) #Affiche 10
print(boyer_moore_horspool("Un algorithme glouton (greedy algorithm en
↪   anglais, parfois appelé aussi algorithme gourmand, ou goulu) est un
↪   algorithme qui suit le principe de faire, étape par étape, un choix
↪   optimum local, dans l'espoir d'obtenir un résultat optimum global. Par
↪   exemple, dans le problème du rendu de monnaie (donner une somme avec le
↪   moins possible de pièces), l'algorithme consistant à répéter le choix de
↪   la pièce de plus grande valeur qui ne dépasse pas la somme restante est un
↪   algorithme glouton. Dans les cas où l'algorithme ne fournit pas
↪   systématiquement la solution optimale, il est appelé une heuristique
↪   gloutonne. L'illustration ci-contre montre un cas où ce principe est mis
↪   en échec.", "monnaie")) #Affiche 288
```

Exercice 5 : Nombres d'opérations *

1 - Modifiez vos fonctions **rechercher_mot(texte, mot)** et **boyer_moore_horspool(texte, mot)** afin de **compter le nombre de tours de boucles effectués**. A la fin, on renvoie dans un **tuple** : La position du mot (ou -1) et ce compteur de tours de boucles.

2 - Reprenez vos tests précédents et comparez, sur un même texte et un même mot recherché, le nombre de tours de boucles effectué par **rechercher_mot** par rapport à **boyer_moore_horspool**.

Exercice 6 : Toutes les positions **

Développez et testez une fonction **boyer_moore_horspool_ameliore(texte, mot)** qui, à partir d'une chaîne de caractères "texte" et d'un mot "mot" renvoie **toutes les positions** du mot (là où il débute) dans "texte", Sous la forme d'une liste. Si le mot n'apparaît jamais, la fonction renvoie une liste vide. La fonction devra appliquer **l'algorithme de Boyer-Moore-Horspool**.

```
def boyer_moore_horspool_ameliore(texte, mot):
"""
    texte : Le texte dans lequel on recherche un mot.
    mot : Le mot qu'on recherche dans le texte.
    Renvoie une liste contenant toutes les position du mot recherché.
    Applique l'algorithme de Boyer-Moore-Horspool
    """


#Exemples de test
print(boyer_moore_horspool_ameliore("bonjour !...Qu'entendez vous par là ? Me
↪  souhaitez vous le bonjour ou constatez vous que c'est une bonne journée,
↪  que je le veuille ou non, ou encore que c'est une journée où il faut être
↪  bon ?", "bon")) #Affiche [0, 59, 99, 188]
```