Recherche textuelle

Introduction

La recherche textuelle est un domaine de l'informatique qui consiste à rechercher un mot ou une expression dans un texte.

Rappel : un texte est une suite de caractères. La chaine "chat" par exemple correspond au tableau de charactères suivant : `['c', 'h', 'a', 't']`

> Exemple : On souhaite rechercher le mot "chat" dans le texte suivant : "Le chat est un animal domestique de la famille des félidés."

Méthode naïve

La méthode naïve consiste à parcourir le texte et à comparer chaque caractère avec le premier caractère du mot recherché. Si le caractère est identique, on compare le caractère suivant avec le deuxième caractère du mot recherché. On recommence jusqu'à ce que tous les caractères du mot recherché soient identiques.

Question 1

```
Implémentez cet algorithme en Python. Testez-le avec le texte et le mot "chat",
puis avec le texte et le mot "chien".
```

```python
def rechercheNaive(texte, mot):
    """
    Recherche le mot dans le texte
    @param texte: str, le texte dans lequel on recherche le mot
    @param mot: str, le mot à rechercher
    @return: bool
    """
```

Cette méthode fonctionne mais elle est très lente. Si le mot recherché n'est pas dans le texte on doit quand meme réaliser toutes les comparaisons avec chacun des caractères du texte. On va donc chercher à améliorer cet algorithme.

Méthode de Boyer-Moore Horspool

La méthode de Boyer-Moore est un algorithme de recherche textuelle se base sur 2 principes pour améliorer la recherche :

- Comparer les caractères du mot recherché en commençant par la fin du mot.
- Utiliser une table de correspondance qui permet de sauter des caractères du texte lorsqu'ils ne correspondent pas au caractère recherché.

Déroulement de l'algorithme

> Pour l'exemple on recherchera le mot "chat" dans le texte suivant : "Le croustichat"

On commence la recherche à l'index 3 :

```
un croustichat
chat
   ^
```

Il n'y a pas de correspondance à la fin du mot : 't' != 'c', donc on avance, mais de combien de caractères avance-t-on ?

Pour le décider, on utilise le fait que le caractère 'c' apparait 3 charactères plus loin dans le mot cherché, donc on peut avancer de 3 caractères sans crainte de rater le mot.

On recherche donc à l'indice 3 + 3 = 6 :

```
un croustichat
   chat
      ^
```

'u' est différent de 'a', on va donc avancer. Mais cette fois, le caractère 'u' n'apparait pas dans le mot cherché, donc on peut avancer de la taille du mot cherché. On recherche donc à l'indice 6 + 4 = 10 :

```
un croustichat
       chat
          ^
```

Cette fois 'c' fais parti du mot recherché, on peut donc avancer selon sa position. Ici 'a' est le 4ème caractère du mot cherché, donc on avance de 3 caractères. On recherche donc à l'indice 10 + 3 = 13 :

```
un croustichat
          chat
             ^
```

Cette fois, 'c' est bien présent dans le mot cherché, on peut donc commencer à comparer les caractères du mot cherché en commençant par la fin.

On compare donc 'c' et 'c', puis 'h' et 'h', puis 'a' et 'a', puis 't' et 't'. Tous les caractères sont identiques, on a donc trouvé le mot cherché.

Table de correspondance

Pour pouvoir avancer de plusieurs caractères, on va utiliser une table de correspondance qui permet de savoir de combien on peut avancer en fonction du caractère du texte.

Pour le mot "chat", la table de correspondance est la suivante :

Caractère	c	h	a	t	(autres charactères)
Index	3	2	1	0	4

> Si une lettre se répète plusieurs fois, on veut uniquement sa dernière position dans le mot car on compare en commençant par la fin. Le mot "abcba" aura donc la même table de correspondance :

Caractère	a	c	b	(autres charactères)
Index	3	2	1	3

Question 2

```
Tracez la table de correspondance, puis appliquez l'algorithme de Boyer-Moore
Horspool pour la recherche du mot "anim" dans le texte "Un petit animal"
```

Implémentation en Python

Question 3

```
Ecrivez une fonction tableCorrespondance qui prend en paramètre un mot et qui
renvoi la table de correspondance associée. On utilisera un dictionnaire pour
stocker la table de correspondance.
```

```python
def tableCorrespondance(mot):
    """Renvoi un dictionnaire représentant la table de correspondance associée au
mot.

    @Args :
        mot : le mot dont on veut la table de correspondance
    @Return :
        dict : la table de correspondance associée au mot
    """
```

> tableCorrespondance("chat") renverra le dictionnaire suivant : {'c': 3, 'h': 2, 'a': 1, 't': 0}

Question 4

Grâce à la table de correspondance, on peut maintenant écrire une fonction avancer qui prend en paramètre un texte, un mot à chercher et un indice du dernier charactère du mot à partir duquel on va tester et qui

renvoi l'indice à partir duquel on doit rechercher le mot suivant.

```
def avancer(texte, mot, indice):
    """
    Renvoi l'indice à partir duquel on doit rechercher le mot suivant.
    @Args
        texte: str, le texte dans lequel on recherche le mot
        mot: str, le mot à rechercher
        indice: int, l'indice du dernier charactère du mot à partir duquel on va
tester
    @Return:
        int, l'indice à partir duquel on doit rechercher le mot suivant
    """
```

Question 5

Implémentez en Python une fonction memeMot telle que :

```
def memeMot(texte, mot, indice):
    """
    Verifie à partir de l'indice de dernier charactère "indice" si le mot "mot"
correspond à celui dans le texte. Renvoi l'indice du premier charactère différent
ou -1 si le mot correspond.
    """
```

> memeMot("un croustichat", "chat", 6) comparera "chat" avec "crou" et renverra 0 car le premier charactère différent est 'u' à l'indice 0.
>
> memeMot("un croustichat", "chat", 13) comparera "chat" avec "chat" et renverra -1 car les mots sont identiques.

Cette fonction sera utilisée pour tester si le mot recherché est présent à un indice donné du texte.

Question 6

Utilisez les fonctions précédentes pour compléter le code de la fonction rechercheBoyerMoore qui prend en paramètre un texte et un mot et qui renvoi si le mot est présent dans le texte.

> Le code à compléter est disponible dans le fichier rechercheBoyerMoore.py

Vous pourrez tester votre code avec le code suivant :

```
# Tests
assert recherche_mot_boyer('Le croustichat', 'chat')
assert recherche_mot_boyer('Le croustichat', 'chien') is False
assert recherche_mot_boyer('Le croustichat', 'crou')
assert recherche_mot_boyer('Le croustichat', 'alice') is False
```