	**Algorithmique**	**NSI T^{ale}**
	Recherche textuelle Algorithme de Boyer-Moore	TP

*

- Ouvrir les fichiers tp_boyer_moore.py et runtest_tpbm.py.
- Tester chaque fonction que vous codez avec la fonction test correspondante (elles sont numérotées dans l'ordre des questions).

1. Méthode find

- Taper help(str.find) dans une console Python.

L'objectif des questions suivantes est de programmer des fonctions de recherche textuelle sans bien sûr utiliser la méthode str.find déjà existante.

2. Fonction identique

La fonction **identique_0** utilise la comparaison de chaînes de caractères déjà implémentée dans Python. On va la reprogrammer afin de pouvoir la modifier.

```
def identique_0(strg, substrg, i, k):
    return strg[i:i+k]==substrg
```

a. Écrire la fonction **identique**`(strg: str, substrg: str, i: int, k: int)->bool:` qui teste si la chaîne substrg de longueur k est présente dans la chaîne strg à la position i, c'est-à-dire si `strg[i:i+k] == substrg`, en comparant les caractères de `strg[i:i+k]` et de `substrg` un par un de gauche à droite jusqu'à trouver un caractère différent.
b. Exécuter la fonction **test_identique** du module **runtest_tpbm**.

3. Algorithme naïf

a. Écrire la fonction **recherche_naive**`(strg: str, substrg: str)->int:` qui recherche la chaîne `substrg` dans la chaîne `strg` de façon naïve et renvoie un entier, égal à la position du premier caractère de `substrg` dans la chaîne `strg` si la recherche aboutit, et à -1 dans le cas contraire. La comparaison s'effectue à l'aide de la fonction identique.
b. Exécuter la fonction **test_recherche_naive**.

4. Algorithme naïf v2

a. Écrire une fonction **recherche_naive_2**`(strg: str, substrg: str)->int:` similaire à la fonction **recherche_naive** mais qui ne fait pas appel à la fonction identique (la comparaison du motif étant recodée dans la fonction **recherche_naive_2**).
b. Exécuter la fonction **test_recherche_naive_2**.

5. Un premier pas vers l'algorithme de Horspool (simplification de l'algorithme de Boyer-Moore)

a. Écrire la fonction **identique_1**`(strg: str, substrg: str, i: int, k: int)->bool:` qui vérifie les mêmes préconditions et postconditions que la fonction **identique**, mais la comparaison avec le motif s'effectue de droite à gauche, en commençant par la dernière lettre du motif `substrg`.
b. Exécuter la fonction **test_identique_1**.
c. Écrire la fonction **horspool_1**`(strg: str, substrg: str)->int:` semblable à **recherche_naive** sauf qu'elle utilise la fonction **identique_1** pour la comparaison du motif.
d. Exécuter la fonction **test_horspool_1**.

6. Vers l'algorithme de Boyer-Moore-Horspool

a. Écrire la fonction **identique_2**`(strg: str, substrg: str, i: int, k: int)->(bool, int):` similaire à **identique_1** mais qui renvoie de plus un entier égal au décalage à appliquer en cas d'échec lors de la comparaison. Si la première lettre lue dans `strg` (la plus à droite) n'apparaît pas dans le motif, ce décalage est égal à k (la longueur du motif `substrg`). Dans le cas contraire, ce décalage est de 1.
b. Exécuter la fonction **test_identique_2**.
c. Écrire la fonction **horspool_2**`(strg: str, substrg: str)->int:` qui applique le décalage renvoyé par **identique_2**.
d. Exécuter la fonction **test_horspool_2**.

7. Première table de sauts

a. Écrire la fonction **premiere_table**`(substrg: str)->dict:` qui renvoie la première table de sauts sous forme de dictionnaire.
b. Exécuter la fonction **test_premiere_table**.

8. Algorithme de Boyer-Moore : version simplifiée de Horspool

a. Écrire la fonction **identique_3**`(strg: str, substrg: str, i: int, k: int, table: dict)->(bool, int):` similaire à **identique_2** mais qui calcule le décalage à appliquer à l'aide de la table de saut et de l'indice j et renverra un décalage de 1 si le décalage calculé est négatif.
b. Exécuter la fonction **test_identique_3**.
c. Écrire la fonction **horspool_3**`(strg: str, substrg: str)->int:` qui applique le décalage renvoyé par **identique_3**.
d. Exécuter la fonction **test_horspool_3**.

9. Calcul du décalage de la seconde table de sauts

a. Écrire une fonction **recherche_suffixe_0**`(substrg: strg, m: int)->bool:` qui récupère le suffixe `substrg[k-m:]` et le caractère précédent `substrg[k-m-1]` où `k = len(substrg)` et renvoie un booléen indiquant s'il existe une précédente occurrence de ce suffixe qui n'est pas précédé du même caractère.
b. Exécuter la fonction **test_recherche_suffixe_0**.
c. Écrire une fonction **recherche_suffixe_1**`(substrg: strg, m: int)->int:` qui récupère le suffixe `substrg[k-m:]` et le caractère précédent `substrg[k-m-1]` où `k = len(substrg)`, recherche dans la chaine `substrg` la précédente occurrence de ce suffixe pas précédée du même caractère et renvoie :

- 1 si m = 0
- k si aucune autre occurrence du suffixe pas précédé du même caractère n'a été trouvée
- le décalage à appliquer pour que cette occurrence se retrouve au début de la fenêtre de recherche dans le cas contraire

On ne traite pas encore le cas où la fin du suffixe se situe au début du motif.
d. Exécuter la fonction **test_recherche_suffixe_1**.
e. ***difficile*** Écrire une fonction **recherche_suffixe_2**`(substrg: strg, m: int)->int:` semblable à la précédente mais qui traite également le cas où la fin du suffixe se situe au début du motif.
f. Exécuter la fonction **test_recherche_suffixe_2**.

10. Algorithme de Boyer-More

a. Écrire une fonction **seconde_table**`(substrg: str)->list:` qui renvoie la seconde table de sauts en utilisant les résultats renvoyés par **recherche_suffixe_2**.
b. Exécuter la fonction **test_seconde_table**.
c. Écrire une fonction

```
identique_4(strg: str, substrg: str, i: int, k: int, table_1: dict,
        table_2: list)->(bool, int):
```

qui utilise les deux tables de sauts pour renvoyer le décalage à appliquer.
d. Exécuter la fonction **test_identique_4**.
e. Écrire une fonction **boyer_moore**`(strg: str, substrg: str)->int:` qui applique l'algorithme de Boyer-Moore.
f. Exécuter la fonction **test_boyer_moore**.

Dans le cas d'un motif très long (500 caractères par exemple), il n'est probablement pas utile de construire une table de 500 ou 499 suffixes. On pourra fixer une limite de taille de suffixes à traiter

Crédit : Grégory Viateau