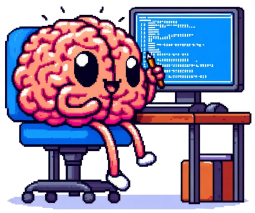
Informatique PT. 2024 - 2025

Mathématiques & Informatique - F. Gaunard
http://frederic.gaunard.com
Lycée Voltaire, Paris 11e.



Devoir maison n°1

Pour le 8 Octobre

Ce devoir est à travailler par groupe de colle. On rendra notamment une seule copie par groupe mais on aura bien pris garde à fournir un travail collectif et inclusif de tou.te.s les membres du groupes qui doivent tou.te.s être capables de commenter chacune des réponses proposées.

Les noms des fonctions doivent être ceux indiqués dans l'énoncé, et chaque fonction doit réaliser exactement la spécification demandée. Vous pouvez commenter le code dès que vous le jugerez nécessaire. Un algorithme correct, même avec des erreurs de syntaxe, peut rapporter des points.

☞ On pourra à chaque question réutiliser les programmes demandés dans les questions précédentes, même si on n'aura pas réussi à coder ceux-ci.

Préambule et objectifs

Le but de ce devoir est d'implémenter un algorithme efficace permettant de déterminer si un mot donné M apparaît dans un texte T et, le cas échéant, de préciser à quel endroit apparaît le mot.
Ce type d'algorithme possède de nombreuses applications (rechercher un mot dans une page web, rechercher une séquence codant une pathologie donnée dans un code génétique,...).

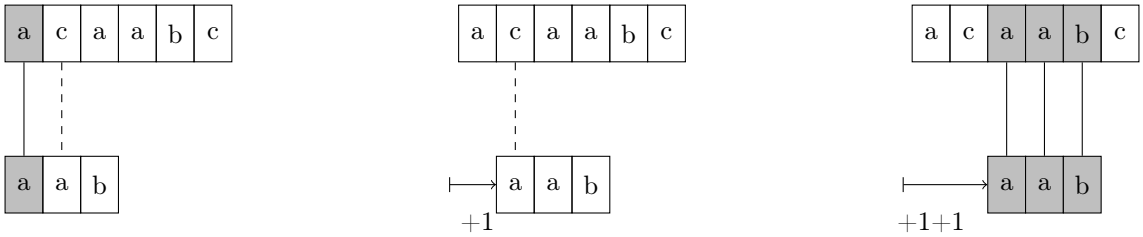
Le mot et le texte seront modélisés par des chaînes de caractères, que l'on notera M et T dans la suite: il s'agit ainsi de déterminer si la chaîne de caractères T possède une sous-chaîne égale à M. Par exemple `'jour'` est une sous-chaîne de `'bonjour'`.

On rappelle les commandes suivantes, qui s'appliquent à toute chaîne de caractères T:

- ✗ `len(T)` renvoie la longueur de T;
- ✗ si $i \leqslant n-1$ où n est la longueur de T, `T[i]` renvoie le i-ème caractère de la chaîne;
- ✗ la commande `+` permet de concaténer deux chaînes de caractères;
- ✗ si M est une seconde chaîne de caractères, alors `M==T` renvoie `True` si les deux chaînes sont égales, et `False` sinon.

1 Partie 1 : un algorithme naïf

Dans un premier temps, nous allons implémenter un algorithme naïf. Son principe est le suivant: on commence par comparer `M` aux `len(M)` premiers caractères de `T`, s'il y a égalité alors on peut renvoyer le résultat, et sinon on recommence en *décalant d'un cran* `M` pour le comparer à la sous-chaîne suivante de `T`.
Ce principe est expliqué sur la figure ci-dessous, où on a pris `T='acaabc'` et `M='aab'`.



1. Extraction d'une sous-chaîne: écrire une fonction `sous_chaine` qui prend en entrée une chaîne de caractères `T`, deux entiers $i \leqslant j \leqslant$ `len(T)`, puis renvoie la sous-chaîne de caractères de T allant du caractère d'indice i au caractère d'indice $j-1$. Par exemple, on doit avoir les résultats d'exécutions ci-dessous

```
>>>sous_chaine('bonjour',0,3)
'bon'
>>>sous_chaine('bonjour',0,1)
'b'
>>>sous_chaine('bonjour',0,0)
' '
```

2. Écrire une fonction `sous_chaine_naif` qui prend en entrée deux chaînes de caractères `M` et `T`, puis renvoie `-1` si `M` n'est pas une sous-chaîne de `T`, et le plus petit entier i tel que `M` est la sous-chaîne de `T` commençant au terme d'indice `i` sinon. *On extraira une à une les sous-chaînes de T pour les comparer à M avec la commande* `==`.

3. **Application**. Déduire de la question précédente une fonction `nb_occurrences` qui prend en entrée deux chaînes de caractères `M` et `T`, puis renvoie le nombre de fois où `M` apparaît dans `T`.

4. **Test**. Télécharger le fichier DM1_PT2425.py et tester votre algorithme en précisant le nombre d'occurrences de votre groupe de colle dans le texte `T`.

5. Justifier que le nombre total de comparaisons entre caractères effectués lors de l'appel de la fonction `sous_chaine_naif` est inférieur à mn, où m, n sont les longueurs respectives de M et T.

2 Partie 2 : préfixes, suffixes et motifs

Soit `M` une chaîne de caractères non vide. Un *préfixe* de `M` est une sous-chaîne de celle-ci, qui commence au terme d'indice 0. Par exemple, `'bon'` et `'bonj'` sont des préfixes de `'bonjour'`, mais `'bnj'` et `'onj'` n'en sont pas.

6. Quelle commande permet d'extraire de la chaîne de caractère `M` le préfixe de longueur de i ?

Un *suffixe* de `M` est une sous-chaîne de celle-ci, qui se termine au terme d'indice $n-1$, où n est la longueur de `M`. Par exemple, `'jour'` et `'our'` sont des suffixes de `'bonjour'`, mais `'jur'` et `'jou'` n'en sont pas.

Enfin, un *motif* de `M` est une sous-chaîne qui est à la fois un suffixe et un préfixe de `M`, mais qui est différente de `M`. Par exemple, `'a'` possède pour unique motif la chaîne vide `''`, et `'abcab'` possède deux motifs: `'ab'` et `''`.

7. Écrire une fonction `suffixe` qui prend en entrée deux chaînes de caractères `P` et `M`, puis renvoie `True` si `P` est un suffixe de `M`, et `False` sinon.

8. Écrire une fonction `motif_max` qui prend en entrée une chaîne de caractères `M`, puis renvoie le motif de `M` de longueur maximale.
On pourra extraire un à un les préfixes de `M` pour les étudier.

9. Écrire une fonction `motif_max_des_prefixes` qui prend en entrée une chaîne de caractères `M`, puis renvoie une liste `L` de même longueur que `M`, telle que `L[i]` est égal à la longueur du motif de longueur maximale du préfixe de longueur $i+1$ de M.
Par exemple, on doit avoir le résultat d'exécution ci-dessous

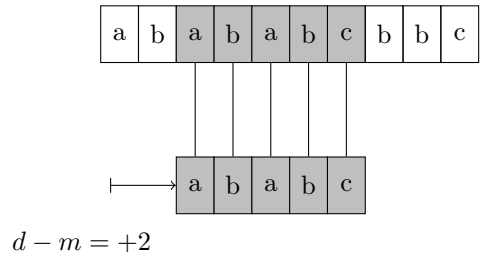
```
>>> motif_max_des_prefixes('abcab')
[0,0, 0, 1, 2]
```

3 Partie 3 : l'algorithme KMP

Le principe de l'algorithme de Knuth-Morris-Pratt (KMP) consiste toujours à énumérer les sous-chaînes de `T`, mais on optimise cette fois le décalage à effectuer lorsque la sous-chaîne étudiée ne correspond pas à `M`: dans l'algorithme naïf on se décalait à chaque fois d'un cran, mais l'algorithme KMP propose d'effectuer un décalage potentiellement plus grand, en fonction de l'analyse des motifs des préfixes de `M`.

Sur le graphe ci-dessous, l'étude du cas `M='ababc'` et `T='abababcbbc'` montre qu'il n'est pas toujours possible après un échec de se décaler du nombre d de comparaisons effectuées avec succès à l'étape précédente, car on pourrait *sauter* une occurrence de `M` dans `T`:

Sur cet exemple, le problème résulte du fait que le préfixe `abab` de `M` possède un motif maximal de longueur $m = 2$ (en l'occurrence `ab`): la sous-chaîne qui échoue au test contient donc le début de la sous-chaîne qui doit réussir le test suivant! Ici, la seconde étape de l'algorithme KMP consiste à effectuer un décalage de $d - m$, ce qui permet de renvoyer le bon résultat:



Le principe de l'algorithme KMP est donc d'effectuer un décalage dépendant du plus long motif du préfixe de `M` que l'on vient de trouver dans la chaîne `T`, et pour cela on utilise la fonction `motif_max_des_prefixes`.

10. Écrire une fonction `Egalite` qui prend en entrée deux chaînes de caractères de même longueur `M1`, `M2`, puis renvoie -1 si celles-ci sont égales, et le plus petit entier k tel que `M1`$[k] \neq$ `M2`$[k]$ sinon.

11. Écrire une fonction `KMP` qui implémente l'algorithme de Knuth-Morris-Pratt: celle-ci prend en entrée deux chaînes de caractères `M` et `T`, puis renvoie `-1` si `M` n'est pas une sous-chaîne de `T`, et le plus petit entier i tel que `M` est la sous-chaîne de `T` commençant au terme d'indice `i` sinon.

12. Pouvez-vous modifier cet algorithme de sorte que chaque caractère de `T` soit lu au plus une seule fois? Que peut-on en déduire?

Bonus : mots conjugués

Cette dernière section, facultative, propose sans indication ni détail une recherche de réponse à un problème.

On dit que deux mots `M` et `M'` sont conjugués si `M=st` et `M'=ts` pour certains `s` et `t`.
Par exemple, `'lanver'` et `'verlan'` sont conjugués ou encore `'fou'` et `'ouf'`.

13. Écrire une fonction `test_mots_conjugués` qui prend en argument deux chaînes de caractères `M` et `M'` et renvoie `True` ou `False` selon que `M` et `M'` sont conjugués ou non.