

Devoir surveillé n°1

Durée : 45 minutes

L'utilisation d'une calculatrice ou de tout document est interdite.

Toute sortie anticipée est interdite.

On rappelle les trois syntaxes d'*extraction* de sous-chaînes (encore appelée "*slicing*") sur une chaîne de caractères. Considérons une chaîne de caractères `ch` de longueur `n`, et `i < j` deux indices valides de `ch`, c'est-à-dire : $0 \leq i < j < n$.

- `ch[i:j]` est la sous-chaîne de caractères (consécutifs) de `ch`, de l'indice `i` **compris** à l'indice `j` **exclu**.
- `ch[:j]` est la sous-chaîne de caractères (consécutifs) `ch[0:j]`.
- `ch[i:]` est la sous-chaîne de caractères (consécutifs) de `ch`, à partir de l'indice `i` **compris** (et jusqu'à la fin de la chaîne).

Par exemple, si `ch = "ATGCAGT"`, alors `ch[2:5] = "GCA"`, `ch[:4] = "ATGC"` et `ch[3:] = "CAGT"`.

Ce sujet porte sur la recherche de sous-chaînes dans une chaîne de caractères : on cherchera d'abord un algorithme naïf puis on cherchera à implémenter l'algorithme KMP. Ce dernier trouve des applications en bio-informatique, dans la recherche de suites de codons dans des séquences d'ADN.

On dira dans tout le sujet qu'une chaîne de caractères `mot` de longueur `p` **est contenue et commence** à l'indice `i` dans une chaîne de caractères `texte` si les chaînes `texte[i:i+p]` et `mot` sont égales.

1. Recherche de sous-chaîne par force brute

1. Donner sans justification tous les indices auxquels la chaîne "AAT" commence et est contenue dans la chaîne de caractères "TTAATGCAATAAC".
2. Parmi les quatre fonctions ci-après, déterminer l'unique fonction `chaines_egales` qui prend en argument deux chaînes de caractères `chaine1` et `chaine2` **de même longueur** (on ne vérifiera pas cette hypothèse) et renvoie `True` si les deux chaînes sont égales, et `False` sinon.

Aucune justification n'est attendue.

```
def chaines_egales1(chaine1, chaine2):
    n = len(chaine1)
    for i in range(n):
        for j in range(n):
            if chaine1[i] != chaine2[j]:
                return False
        else:
            return True
```

```
def chaines_egales2(chaine1, chaine2):
    n = len(chaine1)
    i = 0
    while i < n:
        if chaine1[i] != chaine2[i]:
            return False
        i += 1
    return True
```

```
def chaines_egales3(chaine1, chaine2):
    n = len(chaine1)
    i = 0
    while i < n:
        if chaine1[i] == chaine2[i]:
            return True
        else:
            i += 1
    return False
```

```
def chaines_egales4(chaine1, chaine2):
    n = len(chaine1)
    for i in range(n):
        for j in range(n):
            if chaine1[i] != chaine2[j]:
                return False
    return True
```

3. À l'aide de la fonction précédente, écrire une fonction `liste_occurrences` qui prend en argument deux chaînes de caractères `texte` et `mot` et qui renvoie la liste des indices où est contenue et commence la chaîne `mot` dans la chaîne `texte`. On écrit ici un algorithme naïf qui recherche la chaîne `mot` à tous les indices valables dans la chaîne `texte`.

Par exemple, l'instruction `liste_occurrences("TTAATGCAATAAC", "AAT")` devra renvoyer `[2, 7]`, tandis que `liste_occurrences("TTAATGCAATAAC", "ATT")` devra renvoyer la liste vide `[]`.

4. En déduire une fonction qui renvoie le plus petit indice où une chaîne de caractères `mot` commence et est contenue dans une chaîne de caractères `texte`, et `None` si elle n'y apparaît pas.

2. Bord d'une chaîne de caractères

On dit qu'une chaîne de caractères `sousch` est :

- un **préfixe** d'une chaîne de caractères `ch` s'il existe un indice `j` vérifiant $0 \leq j \leq \text{len}(ch)$ et tel que les chaînes `ch[0:j]` et `sousch` soient égales ; par convention, la chaîne vide `""` est préfixe de toute chaîne de caractères.

Par exemple "GCC" est un préfixe de "GCCATC".

- un **suffixe** d'une chaîne de caractères `ch` s'il existe un indice `j` vérifiant $0 \leq j \leq \text{len}(ch)$ et tel que les chaînes `ch[j:]` et `sousch` soient égales ; par convention, la chaîne vide `""` est suffixe de toute chaîne de caractères.

Par exemple "ATC" est un préfixe de "GCCATC".

- le **bord** d'une chaîne de caractères `ch` si `sousch` est la plus longue chaîne de caractères distincte de `ch` qui soit à la fois un préfixe et un suffixe de `ch`.

Par exemple :

- "AGT" est le bord de "AGTCGAGT" ;
- la chaîne vide `""` est le bord de "AGTCGAGTC" ;
- "TTT" est le bord de "TTTGCCTTT", alors que "TT" ne l'est pas (c'est bien un préfixe et un suffixe mais il n'est pas de longueur maximale.)

1. Donner sans justification le bord de la chaîne de caractères "ATTACGTTCCATTAC".
2. Recopier sur la copie et compléter le code suivant de manière à ce que la fonction `longueur_bord` renvoie la longueur du bord d'une chaîne de caractères `ch` passée en argument.

```
def longueur_bord(ch):
    n = len(ch)
    jmax = 0
    for j in range(1,n):
        if chaines_egales(ch[:j],ch[...:]):
            jmax = ...
    return jmax
```

3. En déduire une fonction `longueurs_bords_prefixes` qui prend en argument une chaîne de caractères `ch` et qui renvoie la liste des longueurs du bord de chaque préfixe de `ch`. La liste `B` renvoyée par cette fonction vérifiera que, pour tout indice `i` de la chaîne `ch`, l'entier `B[i]` désigne la longueur du bord du préfixe `ch[:i+1]` de `ch`.
Par exemple, l'appel de la fonction `longueurs_bords_prefixes("AATGAATC")` devra renvoyer la liste `[0, 1, 0, 0, 1, 2, 3, 0]`. En effet :
 - "" est le bord de la chaîne "A" ;
 - "A" est le bord de la chaîne "AA" ;
 - "" est le bord des chaînes "AAT" et "AATG" ;
 - "A" est le bord de la chaîne "AATGA" ;
 - "AA" est le bord de la chaîne "AATGAA" ;
 - "AAT" est le bord de la chaîne "AATGAAT" ;
 - "" est le bord de la chaîne "AATGAATC".
4. Que renvoie l'instruction `longueurs_bords_prefixes("AATGCAAT")` ? On justifiera la réponse.

3. Recherche de sous-chaîne par l'algorithme de Knuth-Morris-Pratt (KMP)

On cherche à améliorer l'algorithme naïf de recherche d'une sous-chaîne dans une chaîne de caractère.

On peut remarquer que lorsque la recherche de chaîne `mot` dans `texte` échoue à l'indice `i`, il n'est pas toujours nécessaire de chercher `mot` à l'indice `i+1`. On peut parfois décaler la recherche de plusieurs caractères.

L'algorithme de Knuth-Morris-Pratt se déroule de la manière suivante :

- On commence par déterminer la liste des longueurs des bords des préfixes de `mot`. Notons `B` cette liste.
- On initialise à 0 deux variables `i` et `j` représentant respectivement des indices valides des chaînes de caractères `texte` et `mot`.
- Tant que `i` est un indice valide dans `texte` :
 - on détermine le plus petit entier `j` tel que les caractères `texte[i+j]` et `mot[j]` sont différents ;

- si j est égal à la longueur de `mot`, on a trouvé le mot à l'indice i dans `texte` ;
- si j est égal à 0, on se décale d'un seul caractère et on cherche `mot` à l'indice $i+1$ dans `texte`, à partir de l'indice $j = 0$ dans `mot` : $i = i+1$ et $j = 0$;
- sinon, on se décale de $j-B[j-1]$ caractères et on cherche `mot` à l'indice $i+j-B[j-1]$ dans `texte`, à partir de l'indice $j = B[j-1]$ dans `mot` : $i = i+j-B[j-1]$ et $j = B[j-1]$.

Illustrons l'exécution de cet algorithme sur les chaînes :

`texte = "ATCGATCGATCGATG"` et `mot = "ATCGATG"`.

Notons i les indices des caractères de la chaîne `texte` et j ceux de la chaîne `mot`.

Dans les tableaux qui suivent, les caractères dont les comparaisons réussissent sont entourés, ceux dont les comparaisons échouent sont barrés.

La liste des longueurs de bords des préfixes de `mot` est $B = [0, 0, 0, 0, 1, 2, 0]$.

On commence à l'indice $i = 0$ dans `texte`. Les six premières comparaisons réussissent mais la comparaison des caractères `texte[i+j]` et `mot[j]` échoue pour $j = 6$; la chaîne `texte` ne contient pas `mot` à l'indice 0.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
<code>texte[i]</code>	(A)	(T)	(C)	(G)	(A)	(T)	∅	G	A	T	C	G	A	T	G
<code>mot[j]</code>	(A)	(T)	(C)	(G)	(A)	(T)	∅								
j	0	1	2	3	4	5	6								

Puisque le bord du préfixe `mot[:j] = "ATCGAT"` est "AT", sa longueur est $B[j-1] = 2$. On peut alors continuer la recherche de `mot` à l'indice $i = i+j-B[j-1]$, c'est-à-dire $i = 0+6-2 = 4$, dans `texte`. Puisqu'on sait que les deux premiers caractères coïncident, on reprend à partir de l'indice $j = B[j-1]$ dans `mot`, c'est-à-dire $j = 2$.

Lorsque $i = 4$, on compare successivement les caractères suivants jusqu'à un échec à l'indice $j = 6$: les caractères `texte[i+6]` et `mot[6]` ne coïncident pas.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
<code>texte[i]</code>	A	T	C	G	A	T	(C)	(G)	(A)	(T)	∅	G	A	T	G
<code>mot[j]</code>					A	T	(C)	(G)	(A)	(T)	∅				
j					0	1	2	3	4	5	6				

Puisque le bord du préfixe `mot[:j] = "ATCGAT"` est "AT", sa longueur est $B[j-1] = 2$. On peut alors continuer la recherche de `mot` à l'indice $i = i+j-B[j-1]$, c'est-à-dire $i = 4+6-2 = 8$, dans `texte`. Puisqu'on sait que les deux premiers caractères coïncident, on reprend à partir de l'indice $j = B[j-1]$ dans `mot`, c'est-à-dire $j = 2$.

Lorsque $i = 8$, tous les caractères `texte[i+j]` et `mot[j]` coïncident pour tout indice j de `mot`: la chaîne de caractères `ch` contient `mot` à l'indice 8.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
<code>texte[i]</code>	A	T	C	G	A	T	C	G	A	T	(C)	(G)	(A)	(T)	(C)
<code>mot[j]</code>									A	T	(C)	(G)	(A)	(T)	(G)
j									0	1	2	3	4	5	6

Compléter le code de la fonction ci-dessous de manière à ce qu'elle recherche si une chaîne de caractères `mot` est contenue dans une chaîne `texte`. Cette fonction devra renvoyer `None` si `texte` ne contient pas `mot` et le plus petit indice `i` où commence la chaîne `mot` dans le cas contraire.

```
def kmp(texte, mot):
    n = len(texte)
    p = len(mot)
    B = longueurs_bords_prefixes(.....)
    i = 0
    j = 0
    while i < ..... :
        while j < ... and mot[...] == texte[i+j]:
            j += 1
        if j == ..... :
            return .....
        if j == ..... :
            i = i+1
        else:
            i = .....
            j = B[j-1]
    return .....
```

Corrigé

1. Recherche de sous-chaîne par force brute

1. La chaîne "AAT" commence et est contenue dans la chaîne "TTAATGCAATAAC" aux indices 2 et 7.
2. La fonction qui convient est `chaines_egales2(chaine1, chaine2)`. (En effet, la première et la quatrième fonctions renvoient `False` dès qu'un caractère de la `chaine1` est différent d'un caractère de la `chaine2`, quelles que soient leurs positions dans la chaîne et la troisième fonction renvoie `True` dès que les deux chaînes ont un caractère en commun au même indice.
- 3.

```
def liste_occurences(texte, mot):
    n = len(texte)
    p = len(mot)
    L=[]
    for i in range(n-p):
        if chaines_egales(texte[i:i+p], mot):
            L.append(i)
    return L
```

- 4.

```
def premiere_occurrence(texte, mot):
    L=liste_occurences(texte, mot)
    if len(L) == 0:
        return None
    return L[0]
```

2. Bord d'une chaîne de caractères

1. Le bord de "ATTACGTTCCATTAC" est "ATTAC".
- 2.

```
def longueur_bord(ch):
    n = len(ch)
    jmax = 0
    for j in range(1, n):
        if chaines_egales(ch[:j], ch[n-j:]):
            jmax = j
    return jmax
```

3.

```
def longueurs_bords_prefixes(ch):
    L=[]
    for i in range(len(ch)):
        L.append(longueur_bord(ch[:i+1]))
    return L
```

4. L'instruction `longueurs_bords_prefixes("AATGCAAT")` renvoie la liste

[0, 1, 0, 0, 0, 1, 2, 3]

car :

- "" est le bord de la chaîne "A";
- "A" est le bord de la chaîne "AA";
- "" est le bord des chaînes "AAT", "AATG" et "AATGC";
- "A" est le bord de la chaîne "AATGCA";
- "AA" est le bord de la chaîne "AATGCAA";
- "AAT" est le bord de la chaîne "AATGCAAT".

3. Recherche de sous-chaîne par l'algorithme de Knuth-Morris-Pratt (KMP)

```
def kmp(texte, mot):
    n = len(texte)
    p = len(mot)
    B = longueurs_bords_prefixes(mot)
    i = 0
    j = 0
    while i < n - j:
        while j < p and mot[j] == texte[i+j]:
            j += 1
        if j == p:
            return i
        if j == 0:
            i = i+1
        else:
            i = i + j + B[j-1]
            j = B[j-1]
    return None
```