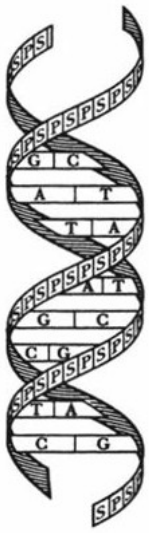
Recherche textuelle : algorithme de Boyer-Moore

Capacités attendues

- ✓ Étudier l'algorithme de Boyer- Moore pour la recherche d'un motif dans un texte.
- ✓ L'intérêt du prétraitement du motif est mis en avant.
- ✓ L'étude du coût, difficile, ne peut être exigée.



Rechercher une chaîne de caractères dans une autre (une sous-chaine dans une chaîne) est un problème récurrent en informatique et dans d'autres sciences. La question se pose particulièrement en génétique pour localiser un **motif** `M` dans une **séquence** `S` d'ADN.

On peut, par exemple, rechercher le motif `M` dans la séquence `S` suivante :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

1 Approche naïve

Nous allons appliquer une méthode itérative **brute** pour rechercher une sous-chaine dans une chaine de caractères.

Une première approche naïve est de placer le motif le plus sur la gauche possible, de le décaler vers la droite d'une lettre à la fois et de le comparer à la séquence jusqu'à trouver une correspondance.

Pour ce faire, nous allons **avancer dans le texte caractère par caractère** puis, si le caractère considéré correspond au premier caractère du mot, nous comparerons les caractères suivants à ceux du mot. La fonction renvoie un couple comprenant le motif et sa position.

```
fonction recherche_naive(sequence, motif):
    ts ← longueur de sequence
    tm ← longueur de motif
    pour i de 0 à ts - tm~:
        j ← 0
        tant que j est inférieur à tm et sequence[i+j] est égal à motif[j]:
            j ← j+1
            si j est égal à tm:
                renvoyer (motif, i)
```

Exercice 1

1. Pourquoi fait-on une boucle bornée parcourant `n-m` caractères ?

2. Que stocke la variable `j` ?

3. Comment décide-t-on quand s'arrêter ?

Estimons la complexité de cette approche. L'opération élémentaire comptabilisée est la comparaison entre deux cases des chaînes `text` et `motif`. Dans le pire des cas, les deux boucles (bornée et non-bornée) sont parcourues en entier. C'est-à-dire que la boucle bornée est itérée `ts - tm` fois et la boucle non-bornée `tm` fois. La complexité est donc de l'ordre de $(ts - tm) \times tm$, soit une **complexité quadratique**.

2 L'algorithme de Boyer-Moore : version simplifiée de Horspool

2.1 Principe

Nous allons étudier une version simplifiée du meilleur algorithme connu : l'algorithme de Boyer-Moore qui a été proposé par Nigel Horspool.

Cet algorithme repose sur deux idées :

- On compare le mot de droite à gauche à partir de sa dernière lettre.
- On n'avance pas dans le texte caractère par caractère, mais on utilise un décalage dépendant de la dernière comparaison effectuée.

Expliquons cet algorithme par un exemple, celui présenté en introduction. On commence par placer le motif `M` le plus à gauche possible de la séquence `S` :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

Le dernier caractère de `M` coïncide avec le 6ème de `S`. On compare alors les deux précédents :

S	A T A A ~~C~~ A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C ~~T~~ A

Ces deux caractères ne coïncident pas. Mais le caractère `"C"` de la séquence apparaît tout de même dans `M`. On décale alors `M` pour que le caractère `"C"` de `S` coïncide avec le même caractère dans `M` :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

On recommence alors la comparaison en partant de la droite de `M` :

S	A T A A C A ~~G~~ A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T ~~A~~

Les caractères ne coïncident pas, mais `"G"` apparaît dans `M`. On décale donc ce dernier de manière à faire coïncider le `"G"` de `S` avec le premier `"G"` de `M` rencontré en partant de la droite :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

On recommence la comparaison en partant de la droite de `M`.

S	A T A A C A G A ~~G~~ A A T A A G G C T A G G A A A T A C T G A A
M	A G G C ~~T~~ A

Les caractères les plus à droite coïncident, mais pas ceux les précédant. On décale alors `M` pour faire coïncider le `"G"` de `S` avec le premier `"G"` de `M` rencontré en partant de la droite :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

Les derniers caractères ne coïncident pas, on décale alors `M` pour faire coïncider le `"T"` de `S` avec le `"T"` de `M` :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

Les derniers et avant-derniers caractères coïncident, mais pas ceux d'avant. On décale donc pour faire coïncider le `"A"` de `S` avec le `"A"` de `M` à gauche des caractères qui coïncidaient :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

Les derniers caractères ne coïncident pas. On décale donc `M` pour faire coïncider le `"G"` de `S` avec le premier `"G"` de `M` en partant de la droite :

S	A T A A C A G A G A A T A A G G C T A G G A A A T A C T G A A
M	A G G C T A

Il y a **correspondance** entre le motif et une partie de la séquence, l'algorithme s'arrête donc.

Exercice 2

1. Quelle partie du motif commence-t-on par comparer ?

2. Quand sait-on qu'on doit décaler le motif ?

3. Comment détermine-t-on alors de combien d'éléments décaler le motif, c'est-à-dire le nombre de sauts à effectuer ? Distinguer le cas dans lequel l'élément de la séquence n'est pas dans le motif du cas dans lequel il est dans le motif.

> **Note**
> Nous avons pris l'exemple de la recherche d'un motif dans une séquence d'ADN, mais plus généralement on peut parler de la recherche de sous-chaîne dans une chaîne de caractères.

Exercice 3

Appliquer l'algorithme de Boyer-Moore pour rechercher le motif `EXEMPLE` dans le texte suivant :

`VOICI UN SIMPLE EXEMPLE`

On détaillera toutes les étapes de la même façon que dans le cours.

L'étude de la **complexité** de cet algorithme dépasse le programme de NSI, mais on observe bien qu'avec les décalages effectués, on ne parcourt pas tous les éléments. Ces décalages, ou sauts, sont à la base de l'implémentation de cet algorithme que nous allons mettre en œuvre par la suite.

2.2 Implémentation Python

Pour implémenter efficacement cet algorithme, on va passer par un prétraitement du motif pour facilement accéder au décalage à effectuer en utilisant un dictionnaire. Voyons d'abord le programme dans sa globalité.

2.2.1 Programme global

```
def recherche_bmh(sequence, motif):
    table_derniere_occurrence = calcule_table_derniere_occurrence(motif)
    idx_sequence = 0
    while idx_sequence + len(motif) <= len(sequence):
        decalage = correspondance(sequence, motif, idx_sequence,
                                  table_derniere_occurrence)
        # si on a trouvé le motif à l'indice idx_sequence de la séquence :
        if decalage == 0:
            return idx_sequence
        idx_sequence = idx_sequence + decalage
    return -1
```

Notre programme est basé sur la fonction `recherche_bmh` qui prend en paramètres deux chaînes de caractères et renvoie un entier correspondant à la position du motif dans la séquence si trouvé, -1 sinon.

Cette fonction commence par calculer la table des positions des dernières occurrences de chaque caractère du motif `calcule_table_derniere_occurrence`.

Elle fait ensuite appel à une fonction `correspondance` qui évalue à partir d'un indice dans la séquence s'il y a correspondance entre la séquence et le motif. Si non, elle renvoie le décalage à appliquer.

Cet appel à `correspondance` est fait tant qu'on n'a pas trouvé le motif et qu'on n'est pas arrivé à la fin.

Nous allons écrire les différentes fonctions intermédiaires utilisées par `recherche_bmh` avant de faire fonctionner l'ensemble du programme.

2.2.2 La table des dernières occurrences

Pour implémenter l'algorithme, nous allons commencer par établir une table des dernières occurrences, indiquant la position la plus à droite de chaque caractère du motif.

On utilisera un dictionnaire `table_derniere_occurrence` dont les clés sont les caractères du motif, et les valeurs l'indice de leur position la plus à droite par rapport au début du motif. On le remplit en effectuant une boucle bornée parcourant tous les caractères du motif.

Par exemple pour le motif « HELLO », on fera :

```
    table_derniere_occurrence['H'] = 0
    table_derniere_occurrence['E'] = 1
    table_derniere_occurrence['L'] = 2
    table_derniere_occurrence['L'] = 3
    table_derniere_occurrence['O'] = 4
```

1. (a) Après exécution de ce programme, quelles valeurs sont associées aux clés 'H', 'E', 'L' et 'O' ?

(b) On utilise une fonction `calcule_table_derniere_occurrence` pour remplir ce dictionnaire. Compléter le code suivant :

```
def calcule_table_derniere_occurrence(chaine):
    lm = len(chaine)
    table_derniere_occurrence = {}
    for j in range(lm):
        .......
    return table_derniere_occurrence
```

(c) Tester à l'aide d'une assertion que la fonction renvoie bien le résultat attendu sur le motif "HELLO".

2.2.3 Test de la correspondance

Nous allons maintenant écrire la fonction `correspondance`.

1. Compléter le pseudo-code suivant.

```
fonction correspondance(sequence, motif, i, a_droite)
    Entrée:
        - sequence: séquence (chaîne de caractères)
        - motif: motif recherché (chaîne de caractères)
        - idx_sequence: position actuelle dans le sequence, correspondant à la
          position du caractère le plus à gauche du motif (entier)
        - table_derniere_occurrence: table de sauts (dictionnaire)

    Sortie:
        - decalage (entier) pour la prochaine étape, ou 0 si la
          correspondance a été trouvée

    Pour idx_motif variant de l'indice du dernier caractère du motif au premier de
    manière décroissante:
        lettre_sequence ← le caractère à l'indice idx_sequence + idx_motif de sequence
        si lettre_sequence est différent de la lettre à l'indice idx_motif:
            si lettre_sequence n'est pas dans table_derniere_occurrence:
                renvoyer l'indice suivant le position courante
            renvoyer .........

    renvoyer .........
```

2. L'implémenter en Python.

2.2.4 Tests

Tester l'ensemble du programme, comprenant les quatre fonctions

- `calcule_table_derniere_occurrence`
- `correspondance`
- `recherche_bmh`

sur les exemples suivants :

- séquence : "AGTCCGTAATGATCGGTACCTGACTGTA", motif : "CTGAC"
- séquence : "ALGORITHME DE BOYER-MOORE-HORSPOOL", motif : "MOORE"