Recherche textuelle

Thème 5

I Introduction

Les algorithmes qui permettent de trouver une sous-chaîne de caractères dans une chaîne de caractères plus grande sont des « grands classiques » de l'algorithmique. On parle aussi de recherche d'un motif (sous-chaîne) dans un texte. Voici un exemple :

Soit le texte suivant :
« *Les sanglots longs des violons de l'automne blessent mon coeur d'une langueur monotone. Tout suffocant et blême, quand sonne l'heure, je me souviens des jours anciens et je pleure.* »

Question : le motif « vio » est-il présent dans le texte ci-dessus, si oui, en quelle(s) position(s) ? (la numérotation d'une chaîne de caractères commence à zéro et les espaces sont considérés comme des caractères)

Réponse : on trouve le motif « vio » en position 23.

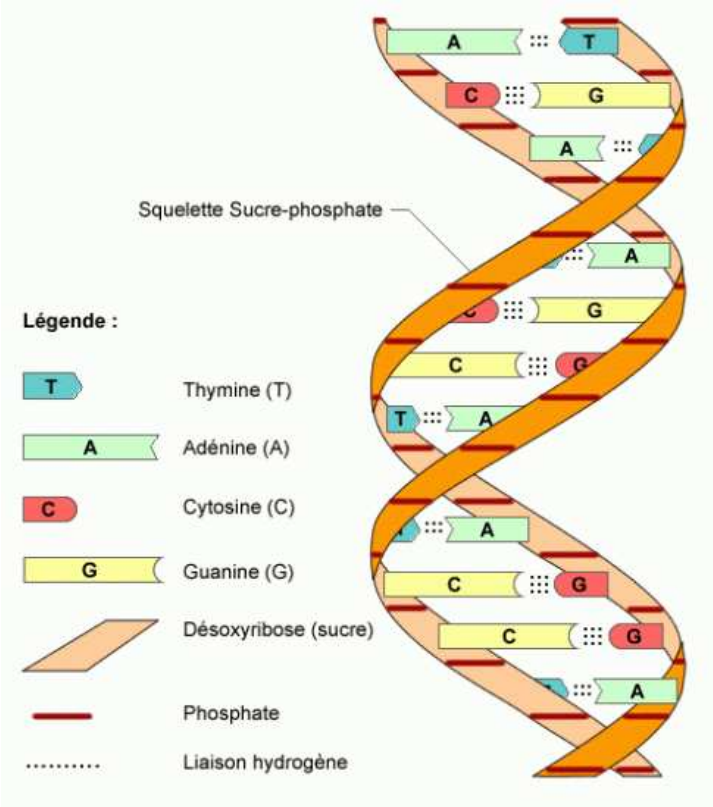
II Bioinformatique

Les algorithmes de recherche textuelle sont notamment utilisés en bioinformatique.

Comme son nom l'indique, la bioinformatique est issue de la rencontre de l'informatique et de la biologie : la récolte des données en biologie a connu une très forte augmentation ces 30 dernières années. Pour analyser cette grande quantité de données de manière efficace, les scientifiques ont de plus en plus recourt au traitement automatique de l'information, c'est-à-dire à l'informatique.

III Analyse de l'ADN

Comme vous le savez déjà, l'information génétique présente dans nos cellules est portée par les molécules d'ADN. Les molécules d'ADN sont, entre autres, composées de bases azotées ayant pour noms : Adénine (représenté par un A), Thymine (représenté par un T), Guanine (représenté par un G) et Cytosine (représenté par un C).



 Licence CC-BY-SA

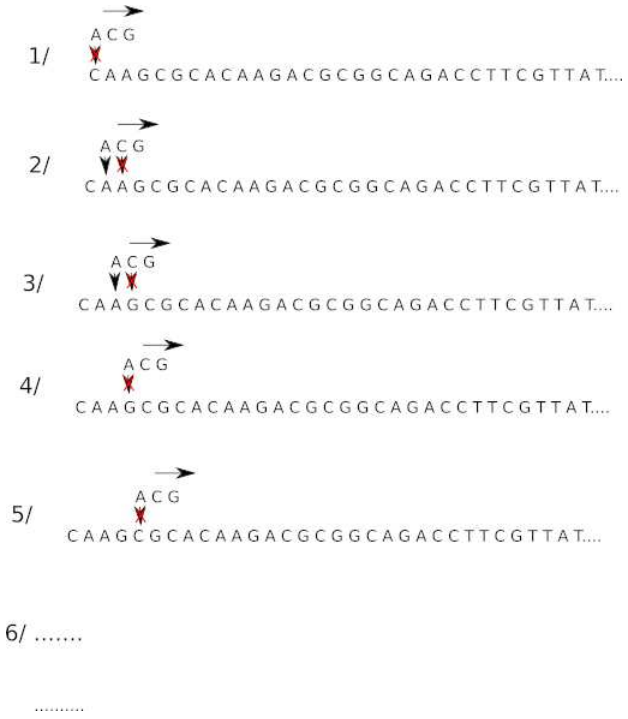
L'information génétique est donc très souvent représentée par de très longues chaînes de caractères, composées des caractères `A`, `T`, `G` et `C`. Exemple : `CTATTCAGCAGTC...`

Il est souvent nécessaire de détecter la présence de certains enchaînements de bases azotées (dans la plupart des cas un triplet de bases azotées code pour 1 acide aminé et la combinaison d'acides aminés forme une protéine). Par exemple, on peut se poser la question suivante : trouve-t-on le triplet `ACG` dans le brin d'ADN suivant (et si oui, en quelle position ?) :

```
CAAGCGCACAAGACGCGGCAGACCTTCGTTATAGGCGATGATTTCGAACCTACTAGTGGGTCTCTTAGGCCGAGCGGTTCCGAGAGATAGTGAAAGATGGCTGG
GCTGTGAAGGGAAGGAGTCGTGAAAGCGCGAACACGAGTGTGCGCAAGCGCAGCGCCTTAGTATGCTCCAGTGTAGAAGCTCCGGCGTCCCGTCTAACCGTACG
CTGTCCCCGGTACATGGAGCTAATAGGCTTTACTGCCCAATATGACCCCGCGCCGCGACAAAACAATAACAGTTTGCTGTATGTTCCATGGTGGCCAATCCGTC
TCTTTTCGACAGCACGGCCAATTCTCCTAGGAAGCCAGCTCAATTTCAACGAAGTCGGCTGTTGAACAGCGAGGTATGGCGTCGGTGGCTCTATTAGTGGTGAG
CGAATTGAAATTCGGTGGCCTTACTTGTACCACAGCGATCCCTTCCCACCATTCTTATGCGTCGTCTGTTACCTGGCTTGGCAT
```

1 Algorithme naïf de recherche

Nous allons commencer par le premier algorithme qui nous vient à l'esprit (on parle souvent d'algorithme « naïf ») :



1. on place le motif recherché au même niveau que les 3 premiers caractères de notre chaîne, le premier élément du motif ne correspond pas au premier élément de la chaîne (`A` et `C`), on décale le motif d'un cran vers la droite.
2. le premier élément du motif correspond au premier élément de la chaîne (`A` et `A`) mais pas le second (`C` et `A`), on décale d'un cran vers la droite.
3. le premier élément du motif correspond au premier élément de la chaîne (`A` et `A`) mais pas le second (`C` et `G`), on décale d'un cran vers la droite.
4. le premier élément du motif ne correspond pas au premier élément de la chaîne (`A` et `G`), on décale d'un cran vers la droite.
5. le premier élément du motif ne correspond pas au premier élément de la chaîne (`A` et `C`), on décale d'un cran vers la droite.
6. ..... on continue le processus jusqu'au moment où les 3 éléments du motif correspondent avec les 3 éléments de la chaîne situés au même niveau.

Exercice 1

1. Ecrire une fonction `recherche_naive` avec deux paramètres : le premier étant le motif que l'on cherche dans le second paramètre. La fonction renvoie un tableau d'entiers correspondant aux indices où le motif est trouvé dans le second paramètre. Ainsi on a :

```
>>> m = "jo"
>>> t1 = "Bonjour"
>>> recherche_naive(m, t1)
[3]
>>> t2 = "Bjoujojoa"
>>> recherche_naive(m, t2)
[1, 4, 6]
>>> t3 = "defghjsascj"
>>> recherche_naive(m, t3)
[]
```

2. Appliquer l'algorithme à l'exemple de recherche du triplet `ACG` dans le brin d'ADN donné.
3. Quelle est la complexité de cet algorithme dans le pire des cas ?

Cet algorithme naïf peut, selon les situations demander un très grand nombre de comparaisons, ce qui peut entraîner un très long temps de « calcul » avec des chaînes très très longues. L'algorithme de Boyer-Moore permet de faire mieux en termes de comparaisons à effectuer

 Licence CC-BY-SA

2 Algorithme de Boyer-Moore-Horspool

Un peu d'histoire

Les faibles performances de la recherche naïve lorsque la taille du texte augmente, a poussé de nombreux informaticiens à proposer des solutions pour améliorer la recherche.

L'algorithme de Boyer et Moore est un algorithme de recherche textuelle très efficace developpé en 1977. **Robert Stephen Boyer** et **J Strother Moore** travaillaient alors à l'université d'Austin au Texas en tant qu'informaticiens.

En 1980, **Nigel Horspool** a conçu une variante simplifiée de l'algorithme de Boyer-Moore.

Le principe de l'algorithme

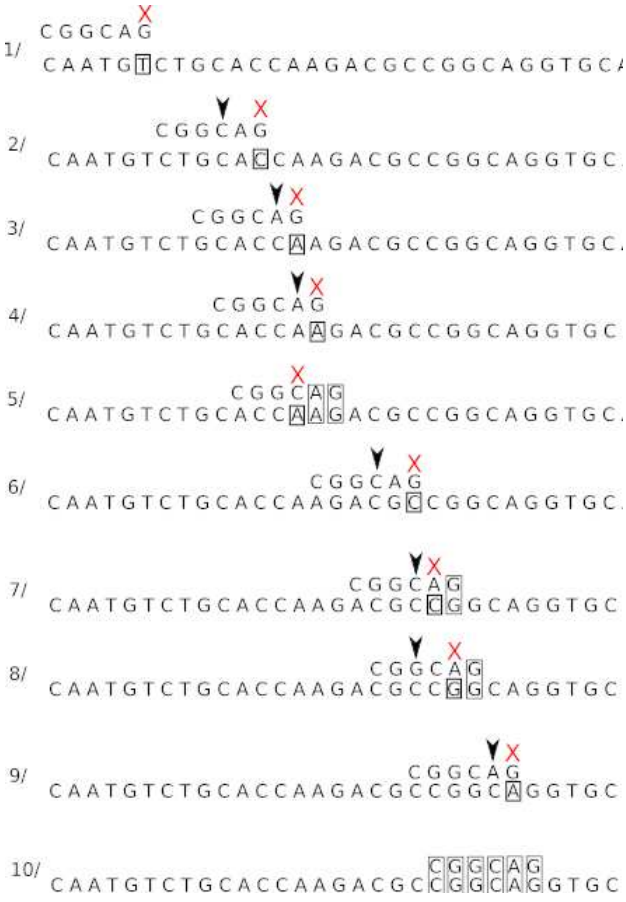
L'algorithme de Boyer-Moore-Horspool se base sur les caractéristiques suivantes :

- L'algorithme effectue un prétraitement du motif. Cela signifie que l'algorithme « connait » les caractères qui se trouvent dans le motif ;
- on commence la comparaison motif-chaîne par la droite du motif. Par exemple pour le motif `CGGCAG`, on compare d'abord le `G`, puis le `A`, puis `C`...on parcourt le motif de la droite vers la gauche ;
- Dans la méthode naïve, les décalages du motif vers la droite se faisaient toujours d'un « cran » à la fois. L'intérêt de l'algorithme de Boyer-Moore, c'est qu'il permet, dans certaines situations, d'effectuer un décalage de plusieurs crans en une seule fois.

Examinons un exemple. Soit la chaîne suivante :

```
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGGCGATGATTTCGAACCTACTAGTGGGTCTCTTAGGCCGAGCGGTTCCGAGAGATAGTGA
AAGATGGCTGGGCTGTGAAGGGAAGGAGTCGTGAAAGCGCGAACACGAGTGTGCGCAAGCGCAGCGCCTTAGTATGCTCCAGTGTAGAAGCTCCGGCGTCCCGT
CTAACCGTACGCTGTCCCCGGTACATGGAGCTAATAGGCTTTACTGCCCAATATGACCCCGCGCCGCGACAAAACAATAACAGTTT
```

et le motif : `CGGCAG`



1. on commence la comparaison par la droite, `G` et `T` ne correspondent pas. Le prétraitement du motif nous permet de savoir qu'il n'y a pas de `T` dans ce dernier, on peut décaler le motif de 6 crans vers la droite.
2. `G` et `C` ne correspondent pas, en revanche, on trouve 2 `C` dans le motif. On effectue un décalage du motif de 2 crans vers la droite afin de faire correspondre le `C` de la chaîne (encadré sur le schéma) et le `C` le plus à droite dans le motif.
3. `G` et `A` ne correspondent pas, il existe un `A` dans le motif, on effectue un décalage d'un cran.
4. `G` et `A` ne correspondent pas, il existe un `A` dans le motif, on effectue un décalage d'un cran.
5. `G` et `G` correspondent, `A` et `A` correspondent, mais `C` et `A` ne correspondent pas. À gauche du `C`, il n'y a plus de `A`, on peut donc effectuer un décalage de 4 crans.
6. `G` et `C` ne correspondent pas, on effectue un décalage de deux crans pour faire correspondre les `C`.
7. `G` et `G` correspondent, `A` et `C` ne correspondent pas, on effectue un décalage d'un cran.
8. `G` et `G` correspondent, `A` et `G` ne correspondent pas, on effectue un décalage de 2 crans (faire correspondre les `G`.
9. `G` et `A` ne correspondent pas, on effectue un décalage d'un cran
10. toutes les lettres correspondent, on a trouvé le motif dans la chaîne.

On peut remarquer que l'on a bien, en fonction des cas, effectué plusieurs décalages en un coup, ce qui, au bout du compte, permet de faire moins de comparaison que l'algorithme naïf. On peut aussi remarquer que plus le motif est grand et plus l'algorithme de Boyer-Moore sera efficace.

Version animée : https://jovilab.sinaapp.com/visualization/algorithms/strings/boyer-moore-horspool

Exercice 2

Appliquez l'algorithme de Boyer-Moore au cas suivant :

Chaîne :
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGGCGATGATTTCGAACCTACTAGTGGGTCTCTTAGGCCGAGCGGTTCCGAGAGATAGTGA
AAGATGGCTGGGCTGTGAAGGGAAGGAGTCGTGAAAGCGCGAACACGAGTGTGCGCAAGCGCAGCGCCTTAGTATGCTCCAGTGTAGAAGCTCCGGCGTCCCGT
CTAACCGTACGCTGTCCCCGGTACATGGAGCTAATAGGCTTTACTGCCCAATATGACCCCGCGCCGCGACAAAACAATAACAGTTT

Motif : ACCTTCG

Exercice 3

1. Étudiez attentivement le programme Python suivant :

```
def recherche_bmh(motif:str, txt:str)->list:
    """Renvoie un tableau contenant les index du paramètre texte où se trouve le
    paramètre mot selon l'algorithme de Boyer-Moore-Horspool.
    """
    long_motif = len(motif)
    long_texte = len(txt)
    tab_car = [-1] * 256
    for i in range(long_motif):
        tab_car[ord(motif[i])] = i
    decalage = 0
    res = []
    while(decalage <= long_texte - long_motif):
        j = long_motif - 1
        while j >= 0 and motif[j] == txt[decalage + j]:
            j = j - 1
        if j < 0:
            res.append(decalage)
            if decalage + long_motif < long_texte :
                decalage = decalage + long_motif - tab_car[ord(txt[decalage + long_motif])]
            else :
                decalage = decalage + 1
        else:
            decalage = decalage + max(1, j - tab_car[ord(txt[decalage + j])])
    return res
```

   (a) Quelles sont le groupes de lignes qui correspondent au prétraitement du motif?
   (b) Quelles sont les noms des deux variables contenant les indices de parcours respectifs du texte et du motif?
   (c) Quelle ligne permet d'indiquer que la comparaison s'effectue à partir du dernier caractère du motif?

2. Testez l'algorithme précédent avec l'exemple proposé à l'activité 2.

Source

- ROCHE D. (2021, 28 mars). *Recherche textuelle*. Informatique au lycée.
  https://pixees.fr/informatiquelycee/n_site/nsi_term_algo_boyer.html
- (2022, 20 avril). *Recherche textuelle*. Cours de terminale NSI
  https://pgdg.frama.io/tnsi/algo/textuelle/

Pour aller plus loin ...

- Découvrir d'autres algorithmes de recherche textuelle
  https://fr.wikipedia.org/wiki/Algorithme_de_recherche_de_sous-chaîne

 Licence CC-BY-SA