	**Algorithmique**	**NSI T^{ale}**
	Recherche textuelle	Cours

*

1. Le problème posé

Nous disposons d'un texte, et d'un motif. Nous nous demandons : le motif est-il présent dans le texte, et si oui, en quelle(s) position(s) ?

Exemple :
Les algorithmes du texte sont souvent utilisés en Bio-Informatique pour, par exemple, analyser de l'ADN.
Considérons par exemple la chaîne de nucléotides suivantes :

`CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG`

Le motif `CGGCAG` est-il présent ?

2. L'algorithme naïf

Naturellement, nous avons envie de procéder comme suit :

```
CGGCAG
|| x
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

 CGGCAG
 x
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

  CGGCAG
  x
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

   CGGCAG
   x
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

      [...]

                                CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG
```

Nous positionnons le motif en début de texte afin de comparer si les lettres correspondent. S'il n'y a pas correspondance, on décale le motif d'un cran vers la droite et on recommence la comparaison.

Cet algorithme est aussi appelé **algorithme de la fenêtre glissante**, puisqu'on fait glisser le motif sur le texte.
On remarque tout de suite qu'il va nous falloir faire beaucoup de comparaisons pour trouver le motif dans le texte, niveau complexité, on ne peut pas faire pire !

3. L'algorithme de Boyer-Moore

L'**algorithme de Boyer-Moore** a pour objectif d'améliorer l'algorithme précédent, en diminuant le nombre de comparaisons.
Observons d'abord l'exemple suivant avant de mettre en forme cet algorithme :

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

1ère idée de l'algorithme de Boyer-Moore : au lieu de comparer le motif au texte de gauche à droite dans le sens "naturel" de lecture, on compare les lettres de droite à gauche, en commençant donc par comparer G et T.
2ème idée : G et T ne correspondent pas. Mais on peut aller plus loin et remarquer que le motif (CGGCAG) ne contient pas la lettre T, on peut donc faire un décalage du motif plus important :

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

Puisque le motif de longueur 6 ne contient pas la lettre T, on peut faire un décalage de 6.
On recommence ensuite nos comparaisons de droite à gauche : G et C. Cela ne correspond pas. Mais le motif (CGGCAG) contient des "C" : on va se focaliser sur le dernier "C" du motif et faire un décalage pour les aligner :

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

Puisque le dernier C est la 4ème lettre d'un mot de 6 lettres, on fait un décalage de 6 - 4 = 2. Les "C" sont alignés.
On recommence ensuite nos comparaisons de droite à gauche : G et A. Cela ne correspond pas, mais le motif contient la lettre "A", on va donc décaler pour les aligner.

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

G et G correspondent, A et A aussi. Mais A et C non. Il n'y a plus de A disponible dans le motif pour faire un alignement. On peut donc faire un décalage de 4 :

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

On a trouvé une occurrence ! On poursuit en regardant la lettre suivante : G. Et on va chercher à l'aligner à nouveau avec notre motif :

CGGCAG
CAATGTCTGCACCAAGACGCCGGCAGGTGCAGACCTTCGTTATAGG

[...] et on poursuit ainsi jusqu'à la fin du texte !

Les trois grandes idées de l'**algorithme de Boyer-Moore** sont donc :

- La **comparaison de droite à gauche** des caractères du motif et du texte.
- Les **décalages du motif de plusieurs crans** vers la droite lorsque c'est possible plutôt que faire glisser la fenêtre d'un seul caractère à la fois dans l'algorithme naïf précédent.
- Pour réaliser ces "grands" décalages du motif, cela nécessite d'**avoir une bonne connaissance** de celui-ci (dans notre exemple, de savoir par exemple que le motif ne contient pas la lettre "T" ce qui permet un décalage de 6 dès les deux premières étapes) et donc de faire un pré-traitement du motif.

Cela se traduit donc par le pseudo-code suivant :

```
boyerMoore(texte, motif) :

# Initialisation des variables
longTexte = longueur(texte)
longMotif = longueur(motif)
resultat = []
decalage = 0
monDico = {}

# Pré traitement du motif
Pour i de 0 à longMotif :
     c = motif[i]
     Si c appartient monDico :
          ajouter i dans la liste monDico[c]
     Sinon
          monDico[c] = [i]

# On cherche TOUTES les occurrences du motif dans le texte
# Donc TANT qu'on n'est pas arrivé à la fin du texte :
tant que (decalage <= longTexte - longMotif) :

     # Je fais la comparaison de droite à gauche
     iMotif = longMotif - 1
     tant que (iMotif >= 0 et motif[iMotif] == texte[decalage + iMo-
     tif]) :
          iMotif = iMotif - 1

     # La comparaison aboutit
     Si iMotif < 0 :
          resultat.append(decalage)

          # Ai-je encore la possibilité de trouver le motif dans le
          texte
          # ou suis-je arrivée à la fin du texte ?
          Si decalage < longTexte - longMotif :

               # Je calcule le nouveau décalage
               # en observant la première lettre qui suit le motif
               trouvé dans le texte
               car = texte[decalage+longMotif]
```

```
                Si car appartient au motif :
                    # j'aligne car avec sa dernière occurrence dans
                    motif
                    indice = monDico[car][-1]
                    decalage = decalage + (longMotif - indice)

                Sinon :
                    # je peux décaler directement de toute la lon-
                    gueur du motif
                    decalage = decalage + longMotif

            Sinon : # j'ai terminé l'étude du texte
                decalage = decalage + 1
                #entraine la sortie de la boucle while principale
        # La comparaison n'a pas aboutit
        Sinon :
            # Le caractère qui n'a pas satisfait la comparaison
            car = texte[decalage + iMotif]

            Si car appartient au motif :
                # J'aligne car avec l'occurence pertinente de car
                dans motif
                indice = plusGrandInferieur(monDico[car], iMotif)
                # comme son nom l'indique, plusGrandInferieur(L,x)
                cherche le
                # plus grand entier appartenant à la liste L infé-
                rieur à x
                decalage = decalage + iMotif - indice

            Sinon :
                # je peux décaler directement de toute la longueur du
                motif
                decalage = decalage + iMotif + 1
```

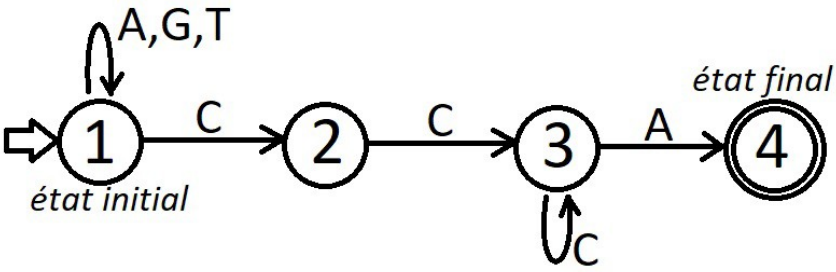
4. Travaux pratiques

Mettre en œuvre l'**algorithme naïf** et l'**algorithme de Boyer-Moore**.

5. Pour aller (beaucoup) plus loin

Il s'agit là d'une très rapide introduction à l'algorithmique du texte, destinée à répondre aux exigences du programme de Terminale NSI.
Pour celles et ceux qui souhaiteraient approfondir ce domaine de l'algorithmique, sachez que l'outil "principal" de l'algorithmique du texte est l'automate. Grossièrement, un automate est un graphe orienté dont les arêtes ont été *étiquetées.*

Exemple :
Voici un automate :



Celui-ci permet de trouver une occurrence du motif "CCA" dans une chaîne de nucléotides.
Si cela vous intéresse, nous pouvons citer l'algorithme de Knuth-Morris-Pratt, qui est un autre algorithme de recherche d'un motif dans un texte. Il existe en deux versions, une version où la présence des automates est masquée pour permettre une compréhension de l'algorithme au plus grand nombre, y compris à ceux qui n'ont jamais entendu parler d'automates, et une deuxième version dans laquelle les automates sont présents.
Tapez "Algorithme de Knuth-Morris-Pratt" ou "Automate des occurrences" dans votre moteur de recherche pour trouver ces deux versions.

Compléments :
https://www.youtube.com/watch?v=cjPv3fyarxU