Chapitre 20 - Recherche textuelle

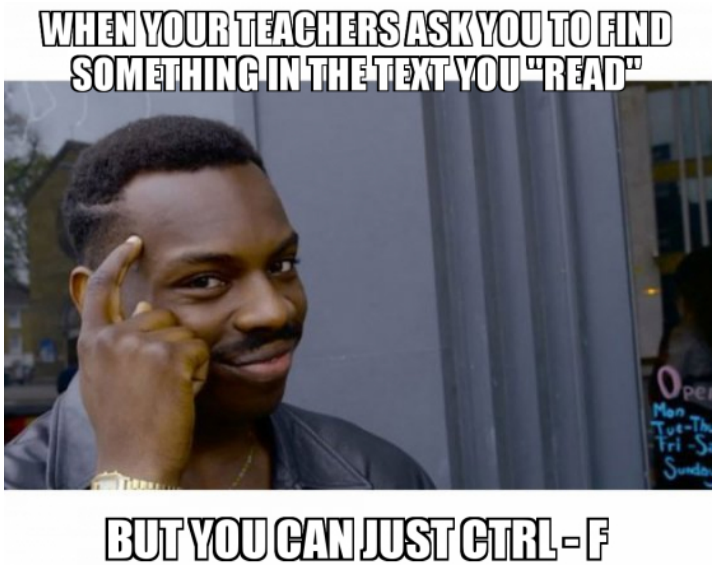
Objectifs :

- ▷ Comprendre le principe de la recherche textuelle.
- ▷ Etudier l'alogirtihme de *Boyer-Moore* pour la recherche d'un motif dans un texte.

1 Introduction

Tout logiciel de traitement de texte possède une fonction de recherche textuelle, permettant de trouver un mot ou une partie d'un mot (chaine ou sous-chaine). Il en est de même pour les navigateurs Internet.

Ainsi, le raccourci clavier `CTRL+F` permet de rechercher dans la page la chaine `"mot"`.



Les algorithmes qui permettent de trouver une **sous-chaine** de caractères dans une chaine de caractères plus grande sont des "grands classiques" de l'algorithmique. On parle aussi de recherche d'un **motif** 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 chaine de caractères commence à zéro et les espaces sont considérés comme des caractères)

Réponse : on trouve le motif "vio" en position 23

Les algorithmes de recherche textuelle sont notamment utilisés en bioinformatique.

2 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.

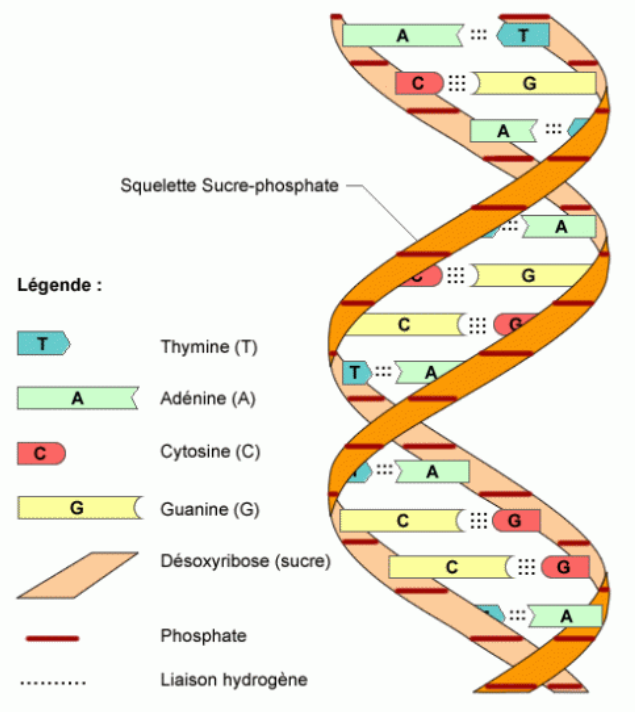
Pour analyser cette grande quantité de données de manière efficace, les scientifiques ont de plus en plus recourt au traitement automatique de l'information, c'est-à-dire à l'informatique.

2.1 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`)
— *Cytosine* (représenté par un `C`)



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

Il est souvent nécessaire de détecter la présence de certains enchainements de bases azotées.

Par exemple, on peut se poser les questions suivantes :

— Trouve-t-on le triplet `ACG` dans le brin d'ADN suivant ?
— Si c'est le cas, en quelle position ?

```
CAAGCGCACAAGACGCGGCAGACCTTCGTTATAGGCGATGATTTCGAACCTACTAGTGGGTCTCTTAGGCCGAGCGG
TTCCGAGAGATAGTGAAAGATGGCTGGGCTGTGAAGGGAAGGAGTCGTGAAAGCGCGAACACGAGTGTGCGCAAGCG
CAGCGCCTTAGTATGCTCCAGTGTAGAAGCTCCGGCGTCCCGTCTAACCGTACGCTGTCCCCGGTACATGGAGCTAA
TAGGCTTTACTGCCCAATATGACCCCGCGCCGCGACAAAACAATAACAGTTTGCTGTATGTTCCATGGTGGCCAATC
CGTCTCTTTTCGACAGCACGGCCAATTCTCCTAGGAAGCCAGCTCAATTTCAACGAAGTCGGCTGTTGAACAGCGAG
GTATGGCGTCGGTGGCTCTATTAGTGGTGAGCGAATTGAAATTCGGTGGCCTTACTTGTACCACAGCGATCCCTTCC
CACCATTCTTATGCGTCGTCTGTTACCTGGCTTGGCAT
```

Nous allons commencer par le premier algorithme qui nous vient à l'esprit. On parle souvent d'**algorithme naïf**.

3 La recherche naïve

Un premier algorithme dit **"naïf"** qui utilise une **méthode itérative brute** permet de rechercher une sous-chaine dans une chaine de caractères en parcourant le texte de gauche à droite. Il faut 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.

3.1 Algorithme

Algorithme 1 : Fonction recherche_naive(texte, motif)

Entrées : Texte et Motif
Sortie : Liste des positions

$m \leftarrow$ longueur du motif `// Longueur du motif`
$n \leftarrow$ longueur du texte $-m+1$ `// Nombre d'itérations`
positions $\leftarrow$ [] `// Liste des positions`

pour *i variant de* 0 *à* n
  si *motif* $=$ *texte*$[i...i+m]$ **alors**
  | ajouter i à la liste *positions*
fin
return *positions*

3.2 Exemple 1

L'une des citations les plus connues du chimiste et philosophe *Antoine Laurent de Lavoisier* est la suivante : `"rien ne se perd, rien ne se crée, tout se transforme"`.

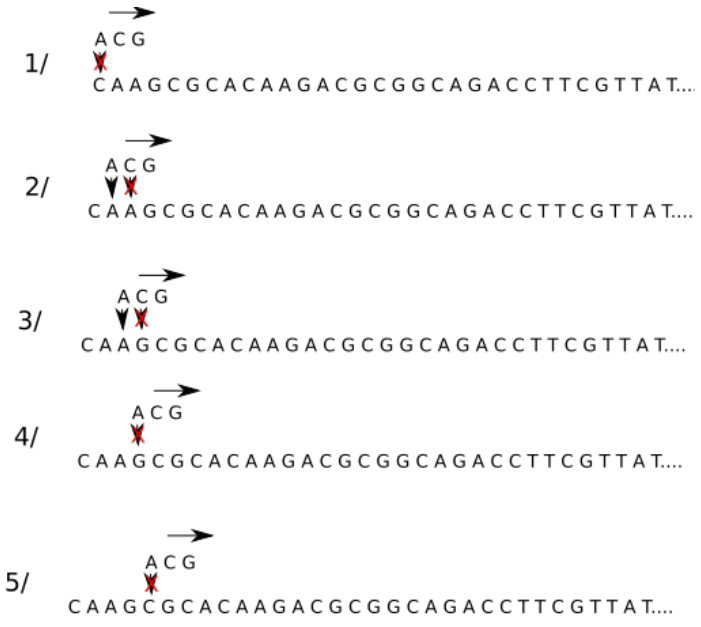
Cherchons la chaine `"rien"` dans le texte de cette citation.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
texte	r	i	e	n		n	e		s	e		p	e	r	d	,		r	i	e	n		n	e		s	e		c	r	é	e	,		t	o	u	t		s	e		t	r	a	n	s	f	o	r	m	e	.
chaine														r	i	e	n																																				

Le motif est trouvé dans le texte à deux reprises, on rajoute les index (`0` et `17`) à la liste des index puis on le décale d'un cran vers la droite.

La recherche se termine lorsqu'on atteint le rang : *longueur du texte - longueur du motif*

3.3 Exemple 2



Par exemple, on cherche la séquence `ACG` dans le brin d'ADN ci-dessus :

Déroulé (étape par étape)

1. On place le motif recherché au même niveau que les 3 premiers caractères de notre chaine, le premier élément du motif **ne correspond pas** au premier élément de la chaine (`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 chaine (`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 chaine (`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 chaine (`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 chaine (`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 chaine situés au même niveau.

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 chaines très très longues.

L'algorithme de *Boyer-Moore* permet de faire mieux en termes de comparaisons à effectuer.

4 Exercices

Exercice 1 : *Fonctions natives de Python*

1. Consulter la documentation de Python sur la méthode `find` des chaînes de caractères.
2. Quel est le rôle de cette méthode ?
3. Tester cette méthode sur les exemples suivants :
   - `'numérique et sciences informatiques'.find('que')`
   - `'numérique et sciences informatiques'.find('nsi')`

Note : Une autre méthode presque identique (`str.index`) lève une erreur lorsque le motif cherché ne se trouve pas dans la chaîne.

Exercice 2 : *Algorithme naïf*

Un outil en ligne permet de visualiser un algorithme dit de "recherche naïve".

1. Utiliser cette outil (en variant éventuellement le motif et la chaîne).
2. Expliquer en quelques mots le principe de cet algorithme de recherche.
3. Sans utiliser `find` ou `index`, et à partir de l'*algorithme 1* du cours, écrire une fonction `recherche(motif,chaine)` qui renvoie `True` ou `False` suivant que la chaîne de caractères `motif` se trouve ou non dans `chaine`.
4. Même question mais en renvoyant l'indice de la première position de `motif` dans `chaine` (si elle s'y trouve) et `-1` sinon.

Exos