18.1 Introduction à la recherche textuelle

NSI terminale - JB Duthoit

18.1.1 Introduction

On s'intéresse ici à la recherche des occurrence d'une chaîne de caractères, appelée **motif**, dans une autre chaîne de caractères, que l'on appellera **texte**.

L'objectif du chapitre est d'écrire une fonction *recherche(motif,texte)* qui affiche les positions de toutes les occurrences d'un motif dans un texte.

☛ Si on note M la longueur du motif, et N la longueur du texte, il peut être utile de noter que $M \leq N$.
☛ De même, si on considère une occurrence de *motif* dans *texte* à la position i, cela signifie que $0 \leq i \leq N - M$.
Nous sommes dans le champ de l'Intelligence Artificielle qui porte sur le traitement automatique de la (des) langue(s)
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 (sous-chaine) dans un texte.

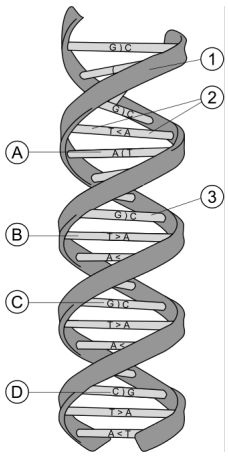
18.1.2 A quoi cela peut-il servir ?

A plein de chose :-) dont notamment :

- recherche d'information dans les grandes bases textuelles
- Fouille de l'opinion et des sentiments
- Traduction automatique
- Robots conversationnels

18.1.3 Exemples

- Recherche d'un mot dans un document (ctrl-f) chercher M = « algorithme » dans le programme NSI de Terminale
- Recherche d'une sous-séquence d'intérêt dans une séquence biologique chercher M = « TTGACA » (promoteur de gène) dans le chromosome 1
- En biologie :



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).

Il peut être utile de détecter la présence de certains enchainements de bases azotées.
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 ?) :

```
 CAAGCGCACAAGACGCGGCAGACCTTCGTTATAGGCGATGATTTCGAACCTACTAGTGGGTCTC
TTAGGCCGAGCGGTTCCGAGAGATAGTGAAAGATGGCTGGGCTGTGAAGGGAAGGAGTCGTGAA
AGCGCGAACACGAGTGTGCGCAAGCGCAGCGCCTTAGTATGCTCCAGTGTAGAAGCTCCGGCGT
CCCGTCTAACCGTACGCTGTCCCCGGTACATGGAGCTAATAGGCTTTACTGCCCAATATGACCC
CGCGCCGCGACAAAACAATAACAGTTTGCTGTATGTTCCATGGTGGCCAATCCGTCTCTTTTCG
ACAGCACGGCCAATTCTCCTAGGAAGCCAGCCAATTTCAACGAAGTCGGCTGTTGAACAGCGAG
GTATGGCGTCGGTGGCTCTATTAGTGGTGAGCGAATTGAAATTCGGTGGCCTTACTTGTACCAC
AGCGATCCCTTCCCACCATTCTTATGCGTCGTCTGTTACCTGGCTTGGCAT
```

18.1.4 Python sait le faire ! et bien !

```
def recherche(mot, texte):
  """Vérifie si une sous-chaîne est dans une chaîne."""
  return mot in texte
```

On peut admirer la concision (et l'efficacité) de cette fonction. Bien que l'on utilise ici directement un idiome PYTHON, il ne faut pas croire pour autant que cette fonction est de complexité constante. En réalité, elle est de complexité quadratique, en $O(nm)$, dans le pire cas. L'implémentation réelle est très efficace et utilise une combinaison astucieuse des algorithmes de BOYER-MOORE et HORSPOOL