Recherche textuelle

Motivation

Nombres univers

Il existe des nombres réels dont l'écriture des décimales renferme l'écriture de tous les entiers naturels, donc toutes les données informatiques imaginables : de véritables tours de Babel. Un tel nombre est appelé « nombre univers ». On peut, par exemple, en construire un artificiellement en juxtaposant simplement l'écriture de tous les entiers.

Exemple – Nombre de Champernowne

David Gawen Champernowne (1912-2000) est un statisticien anglais dont les travaux sur les « nombres normaux » ont été publiés en 1933. Il y présente en particulier ce nombre :
0,123 456 789 101 112 131 415 161 718 192 021 222 324 252 627 282 930 313 233 343 536 373 839 404 1...

Parmi les nombreuses conjectures sur le nombre π, on pense qu'il est un nombre univers.

Exemple – π serait un nombre univers

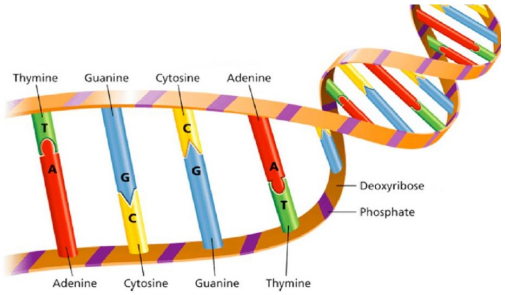
Fabien Olicard, est un mentaliste contemporain, auteur de livres, de spectacles et Youtubeur. Dans un de ses spectacles, il propose de retrouver quelles décimales de π correspondent à la date d'anniversaire d'un spectateur...
3,141 592 653 589 793 238 462 643 383 279 502 884 197 169 399 375 105 820 974 944 59**2 307** 816 4...
Quelqu'un né le 23/07 a sa date d'anniversaire entre les 63^{e} et 66^{e} décimales de π.

Bio-informatique

Le bio-informaticien / la bio-informaticienne fait le pont entre la biologie et l'informatique. Il/elle conçoit et met en œuvre des programmes informatiques pour aider les biologistes à mieux comprendre le monde du vivant.

Par exemple l'information génétique est portée par les molécules d'ADN, qui sont, entre autres, composées de bases azotées : **A**dénine, **T**hymine, **G**uanine et **C**ytosine. Ces informations sont donc souvent représentées par une longue chaine composée des caractères A, T, G et C.

C'est la succession précise et l'enchainement de groupes de trois bases qui sont susceptibles d'avoir du sens pour le « code génétique ».



Exemple – Code génétique (http://barraud.pythonanywhere.com/app/recherche-naive)

Retrouve-t-on la séquence `CAGGAG` ci-dessous ? Combien de fois ? Où ?

```
CAATGTCTGCACCATGACGCCGGCATCTGCGAAGGCAGGAGCATGACCATGATTTCGAACCTACTAGTGGGTCTCTTAGGCCGAGCGGTTCC
GAGAGATAGTGAAAGATGGCTGGGCTGTGAAGGGAAGGAGTCGTGAAAGCGCGAACACGAGTGTGCGCAAGCGCAGCGCCTTAGTATGCTCC
AGTGTAGAAGCTCCGGCGTCCCGTCTAACCGTACGCTGTCCCCGGTACATGGAGCTAATAGGCTTTACTGCCCAATATGACCCCGCGCCGCG
```

Pratiques informatiques courantes

Quand on fait une recherche d'information à travers Internet, on utilise un moteur de recherche, mais on peut parfois obtenir une page très longue ou un document de référence - une notice - de plusieurs dizaines de pages en `PDF`, ou sous un autre format. Il est alors utile de pouvoir faire une recherche de mots-clefs dans ces pages. Le raccourci clavier est généralement [Ctrl]+[F] (F pour *find*).

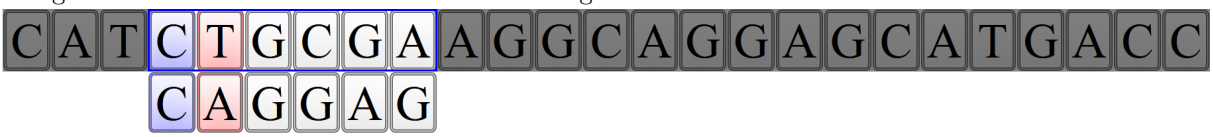
Du côté du programmeur, qui définit des variables et des fonctions dont il a besoin, certains éditeurs de textes mettent même directement en surbrillance les occurrences d'un mot sélectionné...

Algorithme « naïf »

Exo **1** Proposer un algorithme permettant de répondre au dernier exemple (de recherche génétique).

Algorithme de la fenêtre glissante

À chaque étape, on ne considère qu'une partie du texte, de la taille du motif recherché, comme si on le regardait à travers une fenêtre. On fait alors « glisser » cette fenêtre d'une lettre vers la droite.



(http ://barraud.pythonanywhere.com/app/Boyer-Moore) On en déduit l'algorithme suivant :

fonction *recherche(texte,motif)*
  position ← liste vide
  pour i ***allant de*** 0 ***à*** $taille(texte) - taille(motif)$ **faire**
    $j \leftarrow 0$
    tant que $j < taille(motif)$ *et* $texte[i+j] = motif[j]$ **faire**
      $j \leftarrow j + 1$
    si $j = taille(motif)$ **alors**
      Ajoute i à position
  renvoyer *position*

Exo 2 Programmation

1. Programmer en Python l'algorithme précédent et vérifier les résultats suivants :

```
>>> recherche('CATCTGCGAAGGCAGGAGCATGACC', 'CAGGAG')
[12]
>>> recherche('CATCTGCGAAGGCAGGAGCATGACC', 'ATGACC')
[19]
>>> recherche('CATCTGCGAAGGCAGGAGCATGACC', 'GG')
[11, 14]
```

2. Récupérer le fichier `pi1000000.txt` qui contient le premier million des décimales de π et compléter le programme précédent par l'en-tête :

```
pi = ""
with open('pi1000000.txt', encoding='utf8') as decimale:
    for line in decimale:
        pi = pi + line[:-1]
```

On code les jours de l'année sous la forme `jjmm`. Par exemple, le 1er avril est noté `0104`.

a. À quelle décimale de π trouve-t-on le 1er avril pour la première fois ?

b. À quelle décimale de π trouve-t-on votre jour d'anniversaire pour la première fois ?

c. Si on ajoute les deux derniers chiffres de l'année, le 1er avril 2021 se note `010421`. Vérifier que cette date apparait exactement une fois parmi le premier million de décimales de π. Qu'en est-il de votre jour de naissance ?

Idées d'améliorations

On considère la situation de départ suivante :

Après 3 comparaisons on obtient une non-concordance et on avance donc la fenêtre d'une lettre...
Pourrait-on, sans risque, avancer davantage la fenêtre ? (justifier)

On peut avancer de 3 car T n'est pas du tout dans le motif.

Dans la situation précédente, on a utilisé l'information du « mauvais caractère ».

L'autre idée est alors naturellement : peut-on utiliser les informations des bons caractères ?

Au premier abord, la réponse est non. Si on considère que le premier caractère est bon : dès qu'on fait glisser la fenêtre vers la droite, ce caractère n'a plus aucune importance pour les prochaines comparaisons. Deux solutions sont possibles :

⋆ Commencer la recherche par la fin du texte et faire glisser la fenêtre vers la gauche.

⋆ Commencer les comparaisons par la fin du motif.

Si on veut pouvoir modifier l'algorithme pour qu'il ne renvoie la position ou l'existence que de la première occurrence du motif dans le texte, la première solution n'est pas bonne. Pour la suite, on va donc faire les comparaisons par la droite.

Exo **3** Compléter par le nombre de décalage :

Utilisation du mauvais caractère : on avance de 5.....

Utilisation du bon suffixe : on avance de 3.....

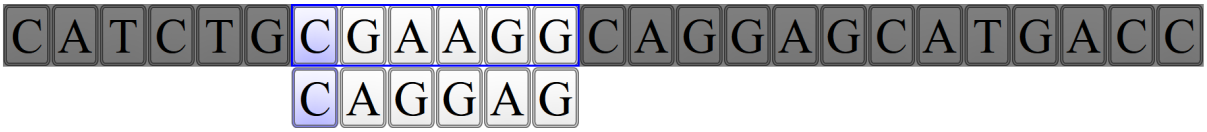
Utilisation du mauvais caractère

Pré-traitement

Si le mauvais caractère n'est pas dans le motif, on peut avancer la fenêtre jusqu'à le dépasser.

On peut également tenir compte d'un mauvais caractère présent dans le motif. Par exemple :

On sait où le mauvais caractère apparait dans le motif. On peut déplacer la fenêtre en conséquence :



« Savoir » où se situe chaque potentiel mauvais caractère, c'est avoir mémorisé leurs positions dans le motif. On peut le faire dans un pré-traitement du motif.

Définition – Tableau des mauvais caractères

Le tableau des mauvais caractères donne le nombre, strictement positif, de positions à remonter depuis la fin du motif pour trouver chaque caractère possible. Il donne la longueur du motif, si le caractère n'y est pas présent.

Exo **4** Compléter le tableau des mauvais caractères :

Motif : CAGGAG

A	C	G	T
..1...	..5...	..2...	..6...

Exo **5** **1.** Quelle structure de données est adaptée pour le traitement des mauvais caractères ?

2. Proposer un algorithme donnant le tableau des mauvais caractères, pour un motif et un alphabet donnés.

Correction pour l'exercice 1.5

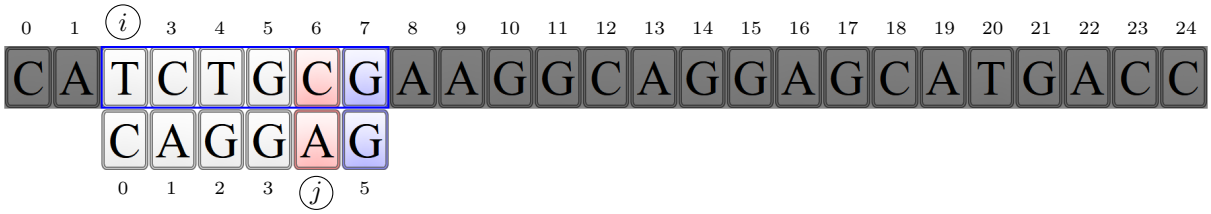
```
fonction mauvaisCaractere(motif, alphabet)
    m ← taille(motif)
    mc ← dictionnaire associant m à chaque lettre de l'alphabet
    pour i allant de 0 à m − 2 faire
        mc[motif[i]] ← m − 1 − i
    renvoyer mc
```

Exo 6 Programmer en Python la fonction proposée ci-dessus.

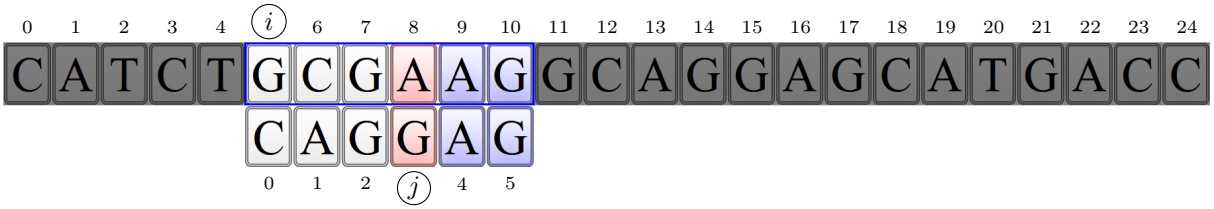
Algorithme tenant compte du mauvais caractère

Pour la suite, on notera i l'indice positionnant le début de la fenêtre de recherche sur le texte et j l'indice dans le motif du mauvais caractère s'il y en a un.

Exemple 1. Ci-dessous : $i = 2$ et $j = 4$



Exemple 2. Ci-dessous : $i = 5$ et $j = 3$



Exo 7 Compléter l'algorithme de recherche suivant, prenant en compte les mauvais caractères :

```
fonction rechercheAvecMC(texte, motif, alphabet)
    mc ← mauvaisCaractere(motif, alphabet)
    position ← liste vide
    i ← 0
    tant que i ⩽ taille(texte) − taille(motif) faire
        j ← taille(motif)−1 ....
        tant que j ⩾ 0 ..... et texte[i + j] = motif[j] faire
            j ← j − 1 .....
        si j < 0 ..... alors
            Ajoute i à position
            i ← i + 1
        sinon
            i ← i+ maximum entre 1 et j − taille(motif) + 1 + mc[texte[i + j]] .....
    renvoyer position
```

Exo 8 En utilisant l'algorithme précédent :

1. De combien décale-t-on la fenêtre de l'exemple 1 ? 4.....
2. De combien décale-t-on la fenêtre de l'exemple 2 ? 1.....
3. De combien décale-t-on la fenêtre s'il n'y a pas de mauvais caractère ? 1.....

Exo 9 Programmer en Python l'algorithme de l'exercice 7 et vérifier les résultats de l'exercice 8.

Utilisation du bon suffixe

Pré-traitement

Exo 10 Cas du bon suffixe redondant

Si le bon suffixe se retrouve dans le motif, précédé d'une lettre différente, on peut en déduire de combien on peut faire glisser la fenêtre. Compléter :

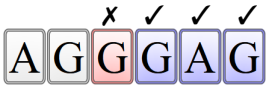
On peut décaler de 3.....

On peut décaler de 2.....

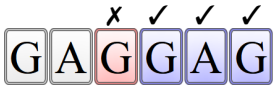
On peut décaler de 1.....

Exo 11 Cas du bon suffixe non redondant

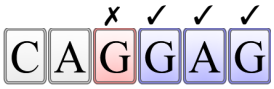
Si le bon suffixe ne se retrouve pas dans le motif, précédé d'une lettre différente, alors on cherche le plus grand suffixe dans le bon suffixe, qui soit aussi un préfixe du motif. S'il n'y en a toujours pas, on peut avancer de la taille du motif.



On peut décaler de 4.....



On peut décaler de 3.....



On peut décaler de 6.....

Définition – Tableau des bons suffixes

Pour chaque indice j entre 0 et $m-1$ donnant la position du mauvais caractère, on appelle « bon suffixe » la séquence des lettres du motif d'indices strictement supérieurs à j.
Le tableau des bons suffixes, donne, pour chaque valeur de j, le plus petit déplacement tel que :

- ⋆ un caractère prenant la place d'un caractère du bon suffixe reste « bon »,
- ⋆ un caractère prenant la place correspondante au mauvais caractère soit différent.

Exo 12 Compléter le tableau des bon suffixes :

Motif : CAGGAG

$j =$	0	1	2	3	4	5
décalage =	..6...	..6...	..6...	..3...	..2...	..1...

Exo 13 Compléter l'algorithme donnant le tableau des bons suffixes :

```
fonction bonSuffixe(motif)
    m ← taille(motif)
    bs ← tableau de m fois la valeur m
    pour j allant de 0 à m − 1 faire
        décalage ← 1
        tant que décalage < m... et bs[j] =m... faire
            # k donne les indices correspondants au bon suffixe
            k ← la plus grande valeur entre j + 1 et décalage
            tant que k < m et motif[k − décalage] = motif[k]..... faire
                k = k + 1
            si k = m et (j <décalage ou motif[j − décalage] ≠ motif[j]..... ) alors
                bd[j] ←décalage
            décalage ← décalage +1
    renvoyer bs
```

Programmer cet algorithme en Python et vérifier les résultats des exercices précédents.

Exo 14 L'algorithme de recherche d'un motif dans un texte, par fenêtre glissante, utilisant les tableaux des mauvais caractères et celui des bons suffixes s'appelle l'algorithme de Boyer–Moore.

Proposer l'écriture de cet algorithme.

Algorithme de Boyer–Moore

Résumé des algorithmes pour la recherche Boyer–Moore :

```
fonction mauvaisCaractere(motif, alphabet)
    m ← taille(motif)
    mc ← dictionnaire associant m à chaque lettre de l'alphabet
    pour i allant de 0 à m − 2 faire
        mc[motif[i]] ← m − 1 − i
    renvoyer mc
```

```
fonction bonSuffixe(motif)
    m ← taille(motif)
    bs ← tableau de m fois la valeur m
    pour j allant de 0 à m − 1 faire
        décalage ← 1
        tant que décalage < m et bs[j] =m faire
            # k donne les indices correspondants au bon suffixe
            k ← la plus grande valeur entre j + 1 et décalage
            tant que k < m et motif[k − décalage] = motif[k] faire
                k = k + 1
            si k = m et (j <décalage ou motif[j − décalage] ≠ motif[j]) alors
                bd[j] ←décalage
            décalage ← décalage +1
    renvoyer bs
```

```
fonction rechercheBoyerMoore(texte, motif, alphabet)
    mc ← mauvaisCaractere(motif, alphabet)
    bs ← bonSuffixe(motif)
    position ← liste vide
    i ← 0
    tant que i ⩽ taille(texte) − taille(motif) faire
        j ← taille(motif) − 1
        tant que j ⩾ 0 et texte[i + j] = motif[j] faire
            j ← j − 1
        si j < 0 alors
            Ajoute i à position
            i ← i + bs[0]
        sinon
            i ← i+maximum entre bs[j] et j − taille(motif) + 1 + mc[texte[i + j]]
    renvoyer position
```