Recherche_textuelle

June 30, 2020

0.1 Recherche naive par fenêtre glissante

```
In [28]: def correspondance_motif(texte, motif,i):
             """Recherche la correspondance de motif dans texte
             à partir de la position i"""
             if i + len(motif) > len(texte):
                 return False
             for j in range(0, len(motif)):
                 if motif[j] != texte[i + j]:
                     return False
             return True

         def recherche_motif_naive(texte, motif):
             """Retourne la position où le motif a été trouvé par fenetre glissante
             ou -1 si le motif ne se trouve pas dans le texte
             Si n = len(texte) et m = len(motif), la complexité est en O((n-m)*m)"""
             for i in range(len(texte) - len(motif) + 1):
                 if correspondance_motif(texte, motif,i):
                     return i
             return -1
```

0.2 Algorithme de Boyer-Moore

Sitographie :

- https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm
- http://whocouldthat.be/visualizing-string-matching/

0.2.1 Règle du mauvais caractère

```
In [29]: def mauvais_caractere(motif, alphabet):
             """Retourne un dictionnaire avec pour chaque caractère de l'alphabet, le nombre d
             à partir de la fin du motif avant de trouver ce caractère
             On ne compte pas la dernière lettre du motif et le décalage vaut m = len(motif)"
             si on ne trouve pas le caractère"""
             m = len(motif)
             #mc = [0] * len(alphabet)
             mc = {c : 0 for c in alphabet} #j préfère utiliser un dictionnaire
```

```
            for c in alphabet:
                k = 1
                while k < m and c != motif[m - 1 - k]:
                    k = k + 1
                mc[c] = k
            return mc
```

In [30]:
```
mauvais_caractere('GCAGAGAG', 'ACGT')
```

Out[30]:
```
{'A': 1, 'C': 6, 'G': 2, 'T': 8}
```

In [31]:
```
def correspondance_suffixe(motif, i, j):
    m = len(motif)
    if motif[j] != motif[i]:
        d = 1
        while  i + d < m and motif[j + d] == motif[i + d]:
            d += 1
        return i + d == m
    return False


def comparaison_prefixe_suffixe(debut_suffixe, motif):
    index_prefixe = 0
    index_suffixe = debut_suffixe
    m = len(motif)
    while index_suffixe < m and motif[index_suffixe] == motif[index_prefixe]:
        index_prefixe += 1
        index_suffixe += 1
    return index_suffixe == m

def bon_suffixe(motif):
    m = len(motif)
    bs = [0] * m
    for i in range(m - 1, -1, -1):
        j = i - 1
        while j >= 0 and not correspondance_suffixe(motif, i, j):
            j = j - 1
        if j >= 0:  #second cas du bon suffixe : recherche du début d'un suffixe/préf
            bs[i] = i - j
        else:  # premier cas du bon suffixe : recherche du
            p = i  + 1
            while p < m and not comparaison_prefixe_suffixe(p, motif):
                p = p + 1
            bs[i] = p
    return bs
```

In [32]:
```
bon_suffixe('GCAGAGAG')
```

Out[32]:
```
[7, 7, 7, 2, 7, 4, 7, 1]
```

```
In [33]: bon_suffixe('ABABA')
```

```
Out[33]: [2, 2, 4, 4, 1]
```

```
In [34]: bon_suffixe('AAA')
```

```
Out[34]: [1, 2, 3]
```

```
In [35]: def boyer_moore(texte, motif, alphabet):
             #initialisation des longueurs
             n = len(texte)
             m = len(motif)
             #pré-traitement du motif
             bs = bon_suffixe(motif)
             mc = mauvais_caractere(motif, alphabet)
             print(bs, mc)
             #recherche du motif  dans le texte
             i = 0 #indice dans le texte
             while i <= n - m:
                 j = m - 1  #on lit le motif de droite à gauche
                 while j >= 0 and motif[j] == texte[i+j]:
                     j = j - 1
                 if j < 0:
                     print(f"Motif trouvé en {i}")
                     #décalage du motif
                     i = i + bs[0]
                 else:
                     #décalage du motif
                     i = i + max(bs[j], mc[texte[i+j]] + j - m + 1)
```

```
In [36]: texte = "GCATCGCAGAGAGTATACAGTACG"
         motif = "GCAGAGAG"
         alphabet = "ACGT"
         boyer_moore(texte, motif, alphabet)
```

```
[7, 7, 7, 2, 7, 4, 7, 1] {'A': 1, 'C': 6, 'G': 2, 'T': 8}
Motif trouvé en 5
```

```
In [37]: T = "GCATCGCAGAGAGTATACAGTACG"
         M = "GCAGAGAG"
         alphabet = "ACGT"
         boyer_moore(T, M, alphabet)
```

```
[7, 7, 7, 2, 7, 4, 7, 1] {'A': 1, 'C': 6, 'G': 2, 'T': 8}
Motif trouvé en 5
```

```
In [38]: bon_suffixe(M)
```

```
Out[38]: [7, 7, 7, 2, 7, 4, 7, 1]
```

```
In [65]: T='CBABABA'
         M='ABABA'
         alphabet = "ACB"
         print("Mauvais caractère : ", mauvais_caractere(M, 'ABC'))
         print("Bon suffixe : ", bon_suffixe(M))
         print(f"Recherche de {M} dans {T} avec Boyer-Moore")
         boyer_moore(T, M, alphabet)
```

```
Mauvais caractère :  {'A': 2, 'B': 1, 'C': 5}
Bon suffixe :  [2, 2, 4, 4, 1]
Recherche de ABABA dans CBABABA avec Boyer-Moore
[2, 2, 4, 4, 1] {'A': 2, 'C': 5, 'B': 1}
Motif trouvé en 2
```

1 Version de Julien Velcin

```
In [41]: T = "GCATCGCAGAGAGTATACAGTACG"
         M = "GCAGAGAG"
         #M = "CCGGTGAA"
         #T = "AAAAAAAAAAAAAAAAAAAA"
         #M = "AAAAAA"
         #T = "AAAAAAAAAAAAAAAAAAAA"
         #M = "ACGT"
         #M = "ACGCA"

         n = len(T)
         m = len(M)
```

```
In [42]: for i in range(n-m+1):
             for j in range(m):
                 if T[i+j] != M[j]: # on s'arrête dès qu'on voit une différence (mismatch)
                     break
             if (j == (m-1)): # critère d'arrêt à (j == (m-1)) car j n'est pas incrémenté à la
                 print("motif trouvé en " + str(i))
```

```
motif trouvé en 5
```

```
In [43]: nb_comp = 0 # nombre total de comparaisons
         i = 0
         while (i <= (n-m)):
             j = 0
             while (j < m) and (T[i+j] == M[j]): # on incrémente tant que c'est identique
                 nb_comp += 1
                 j = j + 1
```

```
            if (j == m): # on remarque que le critère d'arrêt est (j == m) ici
                print("motif trouvé en " + str(i))
            else:
                nb_comp += 1 # pour ne pas oublier de compter les échecs de comparaison (mism
            i = i + 1
        print("Nombre total de comparaisons : " + str(nb_comp))
```

```
motif trouvé en 5
Nombre total de comparaisons : 30
```

1.1 Heuristique du Mauvais Caractère

```
In [46]: symboles = ["A", "C", "G", "T"] # c'est l'alphabet

         # calcul préalable de MC
         MC = {}
         for s in symboles: # on initialise à m par défaut (caractère introuvable dans le moti
             MC[s] = m
         for i in range(m-1):
             MC[M[i]] = m-i-1
```

```
In [45]: MC
```

```
Out[45]: {'A': 1, 'C': 6, 'G': 2, 'T': 8}
```

```
In [47]: import numpy as np

         nb_comp = 0 # nombre total de comparaisons
         i = 0
         while (i <= (n-m)):
             print("Position : " + str(i))
             j = m - 1 # on commence par la fin du motif
             while (j >= 0) and (T[i+j] == M[j]): # on incrémente tant que c'est identique
                 #print("comp de " + str(i+j) + " et " + str(j))
                 nb_comp += 1
                 j = j - 1
             if (j >= 0):
                 nb_comp += 1
                 i = i + np.max([1, MC[T[i+j]] + j - m + 1])
             else: # on remarque que le critère d'arrêt est à présent (j < 0)
                 print("motif trouvé en " + str(i))
                 i = i + 1

         print("Nombre total de comparaisons : " + str(nb_comp))
```

```
Position : 0
Position : 1
Position : 5
```

```
motif trouvé en 5
Position : 6
Position : 14
Position : 15
Nombre total de comparaisons : 15
```

1.2 Heuristique du Bon Suffixe (BS)

```
In [49]: # calcul préalable de BS
         # (attention, il s'agit probablement de l'implémentation la moins efficace
         # mais peut-être la plus claire)

         # calcul du plus grand préfixe qui est également suffixe (mais pas M tout entier)
         i = m-1
         while i > 0 and (M[i:m] == M[0:m-i]):
             # on vérifie que la fin (entre i et (m-1)) est identique au début (entre 0 et m-i
             i = i - 1

         BS = [i+1] * m
         BS[m-1] = 1 # cas particulier pour le dernier symbole de M
         # recherche du prochain motif le plus à droite
         i = m - 2
         while (i >= 0):
             # motif à rechercher
             MM = M[i+1:m]
             l_MM = len(MM)
             k = i
             # on cherche le motif "à rebours"
             while (k>=0):
                 if (M[k:k+l_MM] == MM) and ((k==0) or (M[k-1]!=M[i])):
                     print("à l'index " + str(i) + " : sous-motif " + MM + " trouvé en " + str
                     BS[i] = i - k + 1
                     break;
                 k = k - 1
             i = i - 1
```

```
à l'index 6 : sous-motif G trouvé en 0
à l'index 5 : sous-motif AG trouvé en 2
à l'index 3 : sous-motif AGAG trouvé en 2
```

```
In [50]: BS
```

```
Out[50]: [7, 7, 7, 2, 7, 4, 7, 1]
```

```
In [51]: import numpy as np

         nb_comp = 0 # nombre total de comparaisons
```

```
        i = 0
        while (i <= (n-m)):
            print("Position : " + str(i))
            j = m - 1 # on commence par la fin du motif
            while (j >= 0) and (T[i+j] == M[j]): # on incrémente tant que c'est identique
                nb_comp += 1
                j = j - 1
            if (j >= 0):
                nb_comp += 1
                i = i + BS[j]
            else:
                print("motif trouvé en " + str(i))
                i = i + BS[0]

        print("Nombre total de comparaisons : " + str(nb_comp))
```

```
Position : 0
Position : 1
Position : 5
motif trouvé en 5
Position : 12
Position : 16
Nombre total de comparaisons : 17
```

1.3 Boyer-Moore : mettre tout ça ensemble

```
In [52]: import numpy as np

        nb_comp = 0 # nombre total de comparaisons
        i = 0
        while (i <= (n-m)):
            print("Position : " + str(i))
            j = m - 1 # on commence par la fin du motif
            while (j >= 0) and (T[i+j] == M[j]): # on incrémente tant que c'est identique
                nb_comp += 1
                j = j - 1
            if (j >= 0):
                nb_comp += 1
                i = i + np.max([BS[j], MC[T[i+j]] + j - m + 1])
            else:
                print("motif trouvé en " + str(i))
                i = i + BS[0]
        print("Nombre total de comparaisons : " + str(nb_comp))
```

```
Position : 0
Position : 1
Position : 5
```

```
motif trouvé en 5
Position : 12
Position : 16
Nombre total de comparaisons : 17
```

1.4 Contre-Exemple illustrant la mauvais implémentation du Bon Préfixe

```
In [59]: T='CBABABA'
         M='ABABA'

         n = len(T)
         m = len(M)

         symboles = ["A", "C", "B"] # c'est l'alphabet

         # calcul préalable de MC
         MC = {}
         for s in symboles: # on initialise à m par défaut (caractère introuvable dans le moti
             MC[s] = m
         for i in range(m-1):
             MC[M[i]] = m-i-1


         # calcul préalable de BS
         # (attention, il s'agit probablement de l'implémentation la moins efficace
         # mais peut-être la plus claire)

         # calcul du plus grand préfixe qui est également suffixe (mais pas M tout entier)
         i = m-1
         while i > 0 and (M[i:m] == M[0:m-i]):
             # on vérifie que la fin (entre i et (m-1)) est identique au début (entre 0 et m-i
             i = i - 1

         BS = [i+1] * m
         BS[m-1] = 1 # cas particulier pour le dernier symbole de M
         # recherche du prochain motif le plus à droite
         i = m - 2
         while (i >= 0):
             # motif à rechercher
             MM = M[i+1:m]
             l_MM = len(MM)
             k = i
             # on cherche le motif "à rebours"
             while (k>=0):
                 if (M[k:k+l_MM] == MM) and ((k==0) or (M[k-1]!=M[i])):
                     #print("à l'index " + str(i) + " : sous-motif " + MM + " trouvé en " + st
                     BS[i] = i - k + 1
```

```
                    break;
                k = k - 1
            i = i - 1

        nb_comp = 0 # nombre total de comparaisons
        i = 0
        while (i <= (n-m)):
            print("Position : " + str(i))
            j = m - 1 # on commence par la fin du motif
            while (j >= 0) and (T[i+j] == M[j]): # on incrémente tant que c'est identique
                nb_comp += 1
                j = j - 1
            if (j >= 0):
                nb_comp += 1
                i = i + np.max([BS[j], MC[T[i+j]] + j - m + 1])
            else:
                print("motif trouvé en " + str(i))
                i = i + BS[0]
        print(MC)
        print(BS)
        print("Nombre total de comparaisons : " + str(nb_comp))
```

```
Position : 0
{'A': 2, 'C': 5, 'B': 1}
[4, 2, 4, 4, 1]
Nombre total de comparaisons : 5
```