Table des matières

On complétera au fur et à mesure le fichier python : lien

1 Recherche naïve

Cours diapo 13 à 19

Exercice n° 1

1. Combien faut-il de comparaisons pour chercher les occurrences du motif **abba** dans le texte **aabbababba**.
2. Combien faut-il de comparaisons pour chercher les occurrences du motif **bbba** dans le texte **bbbbbbbba**.
3. Soit un motif de longueur n et un texte de longueur m. Dans le pire des cas, combien faudra t-il faire de comparaisons ?
4. Dans le fichier python, implémenter la recherche naïve.
5. Modifier la fonction afin d'afficher (avec un print) le nombre de comparaison effectuées.
6. Vérifier avec les jeux de tests fournit dans le fichier python.

2 Algorithme de recherche du bon caractere

Cours diapo 20 à 36

Exercice n° 2

1. Avec la recherche naïve, combien faut-il de comparaisons pour chercher les occurrences du motif **GCAGAGAG** dans le texte **GCATCGCAGAGAGTATACAGTACG**.
2. Avec l'algorithme de de recherche du bon caractere, combien faut-il de comparaisons pour chercher les occurrences du motif **GCAGAGAG** dans le texte **GCATCGCAGAGAGTATACAGTACG**.

Exercice n° 3 Bonus

Implémenter en python l'algorithme de recherche du bon caractere

3 L'algorithme de Boyer Moore Horspool

L'algorithme de Boyer-Moore repose sur deux règles :
- celle du bon décalage, implantée dans `bons_decalages`;
- celle du bon bon suffixe, implantée dans `bons_suffixes`.

La deuxième règle peut être omise, conduisant à une version simplifiée de l'algorithme, appelé algorithme de Boyer Moore Horspool.

3.1 Implémentation des bons décalages

Voir cahier de cours

bons_decalages
Entree un motif P, un alphabet Σ.
Sortie un dictionnaire dont les clés sont les lettres de l'alphabet et les valeurs associées sont les décalages occasionnés chaque lettre.

Par exemple, pour le motif `m = GCAGAGAG`, on imagine que la comparaison entre la lettre c du texte et la dernière lettre du motif n'est pas concluante ($c \neq G$), alors :

— Si c est un `A`, alors on peut décaler de 1;
— Si c est un `C`, alors on peut décaler de 6;
— Si c est un `T`, alors on peut décaler de 8;
— Si c est un `G`, alors on peur décaler de 2. (impossible pour ce motif).

Exercice n° 4

1. Implanter cette fonction en python.
2. Vérifier votre implantation avec le jeu de données du fichier python.

3.2 Boyer-Moore-Horspool

Cours diapo 40 à 52

Dans cette version, on n'utilise que la règle du bon décalage. Pour des élèves de nsi, cela nous semble suffisant.

Boyer-moore-horspool
Entrees : Un texte T, un motif P, un alphabet Σ
Sortie : La liste des indices des occurrences de P dans T

$n \leftarrow longueur(T)$
$m \leftarrow longueur(P)$
$j \leftarrow 0$
$bd \leftarrow bons\_decalages(motif, \Sigma)$ // initialise les décalages
$res \leftarrow liste\_vide()$
Tant que $j < n - m$ **faire** // Tant qu'on peut trouver
    $i \leftarrow m - 1$ // Part de la fin du motif
    Tant que $i >= 0$ et $P[j] = T[i + j]$ **faire** // Tant que les caractères correspondent
        $i \leftarrow i - 1$
    Fin Tant que
    Si $i < 0$ **Alors** // A-t-on trouvé un motif ?
        ajoute j à res // oui
        $d \leftarrow bd[T[j + m - 1]]$ // on aligne le dernier symbole
    Sinon
        $d \leftarrow max(1, bd[T[j + i]] - (m - 1 - i))$ // on décale vers la droite
    Fin Si
    $j \leftarrow j + d$
Fin Tant que
Renvoyer (res)

Par exemple, lors de la recherche de `GCAGAGAG` dans `GCATCGCAGAGAGTATACAGTACG` : - on commence par comparer `GCAGAGAG` avec `GCATCGCA`. `G` étant différent de `A`, on décale de 1 ;
- on compare `GCAGAGAG` avec `CATCGCAG`. `G` étant différent de `C`, on décale de 4 = 6 - 2 ;
- on compare `GCAGAGAG` avec `GCAGAGAG` . La comparaison est fructueuse, on décale ensuite de 2 ;
- on compare `GCAGAGAG` avec `AGAGAGTA`. `G` étant différent de `A`, on décale de 1 ;
- on compare `GCAGAGAG` avec `GAGAGTAT`. `G` étant différent de `T`, on décale de 8.
- on compare `GCAGAGAG` avec `ACAGTACG`. `A` étant différent de `C`, on décale de 5 = 6 -1
- l'algorithme s'arrête car $j = 21 > 24 - 7$.

Exercice n° 5

1. Dans le fichier python, implémenter l'algorithme ci-dessus.
2. Modifier la fonction afin d'afficher (avec un print) le nombre de comparaison effectuées.
3. Vérifier avec les jeux de tests fournit dans le fichier python.
4. Que dire du nombre de comparaisons effectuées ?

4 Complexité en temps

Exercice n° 6

A l'aide du fichier python,

1. Réaliser un graphique permettant de comparer les deux algorithmes. (simple_search et Boyer_moore_horspool)
2. Créer un jeu de test mettant en difficulté simple_search (voir exercice 1)

5 L'algorithme de Knuth, Morris et Pratt

5.1 Le meilleur décalage

Voir cours pour les explications

Exercice n° 7

Pour chaque motif, trouver le meilleur décalage après un échec à la position i.

1. motif="GCAGAGAG"

i	0	1	2	3	4	5	6	7
decalage								

2. motif="abbabc"
3. motif="quinquina"
4. motif="agagc"

Exercice n° 8

1. Avec les meilleurs décalages, combien faut-il de comparaisons pour chercher les occurrences du motif **GCAGAGAG** dans le texte **GCATCGCAGAGAGTATACAGTACG**.
2. Comparer les résultats avec l'exercice 2.

Les bords

L'algorithme de Knuth, Morris et Pratt repose sur la fonction `decalages_bords(m)` prenant en paramètre un motif `m` et qui renvoie la liste des longueurs des bords du motif. (voir cours)

Voici le pseudo-code de cette fonction :
décalages_bords
Entrée : le motif P
Sortie : la liste des longueurs des bords des préfixes de P respectant la définition ci-dessus, on mettra -1 si un tel bord n'existe pas.
1. $m \leftarrow longueur(P)$
2. $\pi \leftarrow liste\_vide()$
3. ajouter -1 à π // le préfixe de longueur 0 n'a pas de bord respectant la définition ci-dessus
4. $k \leftarrow -1$ // on commence avant le motif
5. **Pour** q **allant de** 0 à $m-1$
6. **Tant que** $k > -1$ et $P[k] \neq P[q]$ // P[:k] n'est pas un suffixe de P[:q] $\rightarrow$ on n'a pas un bord
7. $k \leftarrow \pi[k]$ // bord précédent
8. **Fin Tant que**
9. $k \leftarrow k+1$
10. **Si** $q+1 < m$ et $P[k] = P[q+1]$ **alors** // les caractères qui suivent le bord sont-ils identiques ?
11. ajoute $\pi[k]$ à π // oui : on se contentera du bord précédent
12. **Sinon**
13. ajoute k à π // non : on enregistre la longueur du bord
14. **Fin pour**
15. **renvoyer** π

Exercice n° 9

Implanter l'algorithme ci-dessus : (voir fichier python)

Exercice n° 10

1. On donne une fonction permettant de trouver les k mots les plus intéressants pour KMP. (ceux qui ont les plus longs décalages possibles). (voir fichier python)
2. Utiliser cette fonction pour trouver dans les misérables les 10 mots les plus intéressants (pour KMP). On rappelle que pour lire toutes les lignes d'un fichier on peut utiliser la fonction `read` :

On donne maintenant le pseudo-code de l'algorithme de KMP :
Recherche_KMP
Entrées : le texte T, le motif P.
Sortie : La liste de tous les indices de P dans T.

1. $n \leftarrow longueur(T)$
2. $m \leftarrow longueur(P)$
3. $\pi \leftarrow decalages\_bords(P)$
4. $res \leftarrow liste\_vide()$
5. $q \leftarrow 0$ // nombre de caractères comparés avec succès
6. **Pour** i **allant de** 0 **à** $n-1$ // parcours le texte de gauche à droite
7. **Tant que** $q > -1$ et $P[q] \neq T[i]$
8. $q \leftarrow \pi[q]$ // le caractère suivant ne convient pas
9. **Fin Tant que**
10. $q \leftarrow q+1$
11. **Si** $q = m$ **Alors** // tous les caractères conviennent ?
12. ajouter $i+1-m$ à res // ajoute l'indice du début
13. $q \leftarrow \pi[q]$ // cherche l'occurrence suivante
14. **Fin Si**
15. **Fin Pour**
16. **Renvoyer** res.

Exercice nº 11

1. Implanter en python l'algorithme de KMP. (voir fichier python)
2. Modifier la fonction afin d'afficher (avec un print) le nombre de comparaison effectuées.
3. Vérifier avec les jeux de tests fournit dans le fichier python.

Exercice nº 12

1. Compléter le tableau suivant à l'aide des jeuxde tests données dans le fichiers python (voir ex 1)

Nb de comparaison	jeu 1	jeu 2	jeu 3	jeu 4
recherche naïve				
algorithme Boyer Moore Horspool				
algorithme de Knuth Morris et Pratt				

2. Les résultats sont-il conforme à ceux attendus ?

6 L'algorithme de Boyer Moore - le retour -

Le bon suffixe L'algorithme complet de Boyer-Moore utilise également la règle du bon suffixe. (voir cours)

On peut enfin écrire une fonction implantant l'algorithme de Boyer-Moore.
Cette fonction compare à chaque étape le décalage induit par la règle du bon caractère avec celui induit par la règle du bon suffixe (voir fichier python)

Exercice nº 13

Tester la fonction boyer_moore_search (voir fichier python)

7 Comparatif

Exercice nº 14

1. Construire un graphique permettant de comparer le temps mis par chacun des algorithme (simple_search et Boyer_Moore_Horspool_search)
2. Créer un jeu de tests qui va mettre simple_search en difficulé. (voir exercice1)
3. Faire un comparatif sur une sequence ADN (voir fichier : lien)