Math93.com

NSI - Recherche Textuelle

Objetctifs

- Thème : présentation de la méthode naïve de recherche d'un mot dans un texte et de l'algorithme de **Boyer-Moore**;
- Durée de la séquence : 3 fois 2 heures;
- Prérequis : bonne connaissance de Python, dictionnaires, listes.
- Contraintes : projecteur pour les présentations et salle informatique pour les TD.
- B.O. : « *Étudier l'algorithme de Boyer-Moore pour la recherche d'un motif dans un texte. L'intérêt du prétraitement du motif est mis en avant. L'étude du coût, difficile, ne peut être exigée.* »
- L'ensemble des documents est disponible sur : https://www.math93.com

1 La recherche textuelle : problématique

1.1 Histoire et Applications

Trouver les occurrences d'une chaîne de caractères (motif ou mot) dans un texte est un problème qui se rencontre fréquemment dans les éditeurs de texte. Les algorithmes de recherche de chaîne de caractères sont également utilisé pour trouver une séquence ADN par exemple ou pour rechercher des pages web correspondantes à une recherche Internet. Dans le domaine de la génétique, on peut par exemple rechercher :

- des 'mots' courts comme la séquence ATG(qui code une méthionine (Met)) correspondant au codon initiateur sur le brin sens(codant, non transcrit), marquant le début de la séquence transcrite d'un gène,
- des 'mots' plus longs afin de déterminer la présence d'allèles particuliers de gène.

1.2 Un premier exemple

texte :	a	b	c	a	b	a	a	b	c	a	b	a

motif :				a	b	a	a

2 La méthode naïve : implémentation

2.1 Une animation ppt

- Présentation 1 : la recherche naïve : lien Pdf

2.2 Un premier problème

Après avoir trouvé une occurrence valide, peut-on décaler de plus de 1 ?

2.3 Explication

La procédure naïve de recherche peut s'interpréter graphiquement de la façon suivante : on passe sur le texte un« modèle » contenant la chaîne recherchée et l'on note les décalages à partir desquels tous les caractères du modèle sont égaux aux caractères correspondants du texte.
L'algorithme naïf trouve tous les décalages valides en utilisant une boucle qui teste la condition

$$mot[1\cdots m] == texte[s+1\cdots s+m]$$

avec toutes les valeurs possibles de s, de 0 à $n-m$.

Algorithm 1 : recherche_naive(motif,texte)

```
1: n ⟵ longueur du texte
2: m ⟵ longueur du motif
3: compteur_occurrences ⟵ 0
4: Pour s de 0 à n − m faire
5:     Si motif == texte[s···s+m] alors
6:         compteur_occurrences ⟵ compteur_occurrences+1
7:     Fin Si
8: Fin Pour
9: Renvoyer compteur_occurrences
```

2.4 TD : Exercice implémentation sous Python

Exercice 1 Algorithme naïf

Écrire une fonction ***recherche_naive(motif,texte)*** avec en entrées des strings motif et texte et qui donne en sortie le nombre d'occurrences du motif dans le texte.
On pourra aussi proposer un affichage du décalage de chaque occurrence trouvée. *Une assertion vérifiera que le texte est de longueur supérieure ou égale au mot.*

2.5 Peut-on optimiser la recherche textuelle naïve?

Discussion :

- Dans quelles conditions de recherche textuelle pourra-t-on, après la trouvaille d'une occurrence, décaler de plus de 1 caractère? Expliquer l'avantage que ce décalage accru représenterait.
- Pourquoi le cas de la recherche d'un mot dans un texte est vraiment un cas particulier?

3 Algorithme de Boyer-Moore-Horspool

3.1 Histoire

L'algorithme de Boyer-Moore-Horspool ou Horspool est un algorithme de recherche de sous-chaîne publié en 1980 par Nigel Horspool, un professeur à l'université de Victoria au canada.
Il consiste en une simplification de l'algorithme de Boyer-Moore qui ne garde que la première table de saut . Notion que nous allons ci-après expliciter.

3.2 Des animations ppt

- Présentation 2 : Boyer-Moore-Horspool : pdf
  Construction de la table des sauts pour l'algorithme de Boyer-Moore.
- Présentation 3 (vidéo) : Boyer-Moore-Horspool : vidéo
  Construction de la table des sauts pour l'algorithme de Boyer-Moore.

3.3 Explication

- Fonctionnement.
  - L'algorithme de Boyer-Moore effectue la vérification, c'est-à-dire qu'il tente d'établir la correspondance entre le motif et le texte à une certaine position, à l'envers.
  - Par exemple, s'il commence la recherche du motif CTGCGA au début d'un texte, il vérifie d'abord la sixième position en regardant si elle contient un A. Ensuite, s'il a trouvé un A, il vérifie la cinquième position pour regarder si elle contient le dernier G du motif, et ainsi de suite jusqu'à ce qu'il ait vérifié la première position du texte pour y trouver un A.
  - Dans la méthode naïve, les décalages du motif vers la droite se faisaient **toujours d'un "cran" à la fois**. L'intérêt de l'algorithme de Boyer-Moore, c'est qu'il permet, dans certaines situations, d'effectuer **un décalage de plusieurs crans en une seule fois**. Pour comprendre la raison de cette méthode il faut étudier le cas où la vérification échoue.

— La règle du mauvais caractère.
  — Par exemple si au lieu de trouver un A en sixième position, un E est lu. Le E n'apparaît nulle part dans le motif CTGCGA (de longueur 6), ce qui signifie qu'aucune correspondance avec la sous-chaîne du texte n'existe au tout début du texte, ainsi que dans les cinq positions qui la suivent. Après la vérification d'un seul caractère, l'algorithme est capable de passer ces six caractères et donc d'effectuer un saut de 6.

indice	0	1	2	3	4	5	6	7	8	9	10	11
texte	C	T	G	G	T	E	C	T	G	C	G	A
motif étape 1	C	T	G	C	G	A						
motif étape 2							C	T	G	C	G	A

  — Par exemple si au lieu de trouver un A en sixième position, un T est lu. Le T apparaît dans le motif CTGCGA (de longueur 6), on va alors effectuer un saut de 4 pour aligner le plus proche T du motif avec celui du texte.

indice	0	1	2	3	4	5	6	7	8	9	10	11
texte	C	T	G	G	C	T	G	C	A	C	G	A
motif étape 1	C	T	G	C	G	A						
motif étape 2					C	T	G	C	G	A		

  — Des cas plus complexes peuvent se produire lorsque certaines lettres coïncident, car le saut doit être calculé par rapport à l'indice de la lettre du motif. Ils sont explicités dans la présentation powerpoint pour plus de clarté.

  Par exemple si les deux lettres G et A coïncident, on teste celle en 4e position qui ne coïncide pas puisqu'on lit un T au lieu d'un C. Le T est dans le motif, on va effectuer un saut de $4-2=2$. Si ce calcul donne une valeur négative ou nulle, on effectuera un saut de 1.

indice	0	1	2	3	4	5	6	7	8	9	10	11
texte	C	T	G	T	G	A	G	C	A	C	G	A
motif étape 1	C	T	G	C	G	A						
motif étape 2			C	T	G	C	G	A				

  Ce principe de fonctionnement explique pourquoi **plus le motif est long, et plus l'algorithme est efficace pour le trouver**.

— Le prétraitement du motif.
  — Cela signifie que l'algorithme "connaît" les caractères qui se trouvent dans le motif.
  — On va construire ce que l'on nomme une table des sauts, pour chaque caractère du motif. Celle table traduit en fait l'écart minimal entre une lettre du motif et la fin du motif. La dernière lettre du mot est traitée à part, elle renvoie un écart maximal si elle n'est pas présente ailleurs dans le mot.

Exemple

Par exemple pour le motif de longueur 6 : CTGCGA, la table des sauts sera :

Motif	**CTGCGA**			
Lettres	A et autres lettres	G	C	T
Distance à la fin du motif	6	1	2	4

Exemple

Par exemple pour le motif de longueur 6 : ATGCGA, la table des sauts sera :

Motif	**ATGCGA**				
Lettres	autres lettres	A	G	C	T
Distance à la fin du motif	6	5	1	2	4

3.4 TD : Exercice implémentation sous Python

3.4.1 Création de la table de sauts

Exercice 2

1. Écrire une fonction qui donne la table des sauts d'un motif. Utiliser un dictionnaire dont les clés seront les lettres du motif et les valeur le saut associé.
2. Effectuer des tests appropriés.
3. Faire tourner l'algorithme à la main sur quelques exemples.

```
# Par exemple :
>>>motif='bonjour'
>>>table_sauts(motif)
{'b': 6, 'o': 2, 'n': 4, 'j': 3, 'u': 1}
>>>table_sauts('ACTGACTGACTG')
{'A': 3, 'C': 2, 'T': 1, 'G': 4}
```

3.4.2 Le code

Exercice 3

On vous propose le code de la fonction **boyer_moore_horspool** qui renvoie une liste, des positions du motif dans le texte.

1. Annoter et expliquer les différentes étapes de la fonction. En faire la docstring et l'assert.
2. Effectuer des tests appropriés avec des motifs et textes de votre choix.

```
# Code Boyer-Moore-Horspool
def boyer_moore_horspool(texte, motif):
    size = len(texte)
    taille = len(motif)
    positions = []
    if(taille<=size):
        decalage = table_sauts(motif)
        i=0
        trouve=False
        while(i<=size-taille):
            for j in range (taille-1,-1,-1):
                trouve=True
                if(texte[i+j]!=motif[j]):
                    if(texte[i+j] in decalage and decalage[texte[i+j]]<=j):
                            i+=decalage[texte[i+j]]
                    else:
                        i+=j+1
                    trouve=False
                    break
            if(trouve):
                positions.append(i)
                i=i+1
                trouve=False

    return positions
```

4 Gain de temps?

4.1 Mesure du temps pour la méthode naïve

Exercice 4

Écrire une fonction renvoie le temps de recherche de toutes les occurrences d'un motif dans un texte avec l'algorithme naif. Testez-la avec un texte de votre choix .
Vous pourrez aussi évaluer avec un compteur le nombre de comparaisons effectuées avec cet algorithme.

4.2 Mesure du temps pour la méthode Boyer-Moore-Horspool

Exercice 5

Écrire une fonction renvoie le temps de recherche de toutes les occurrences d'un motif dans un texte avec l'algorithme Boyer-Moore-Horspool. Testez-la avec un texte de votre choix.
Vous pourrez aussi évaluer avec un compteur le nombre de comparaisons effectuées avec cet algorithme.

4.3 Comparaison avec l'exemple initial puis avec un fichier texte proposé « La Disparition »

Exercice 6

On va comparer les deux algorithmes sur un texte plus long en faisant varier le motif.

1. Importer le fichier texte *ladisparition.txt* qui propose un extrait du roman « La Disparition » écrit en 1968 par l'écrivain français Georges Perec (1936-1982) et publié en 1969.
2. Essayer votre fonction de recherche sur ce texte avec quelques mots comme 'envie', 'esprit', 'est', 'les'. Puis avec la lettre 'e', Que remarquez-vous?
3. Ajouter un compteur des différentes comparaisons effectuées par chacun des algorithmes.

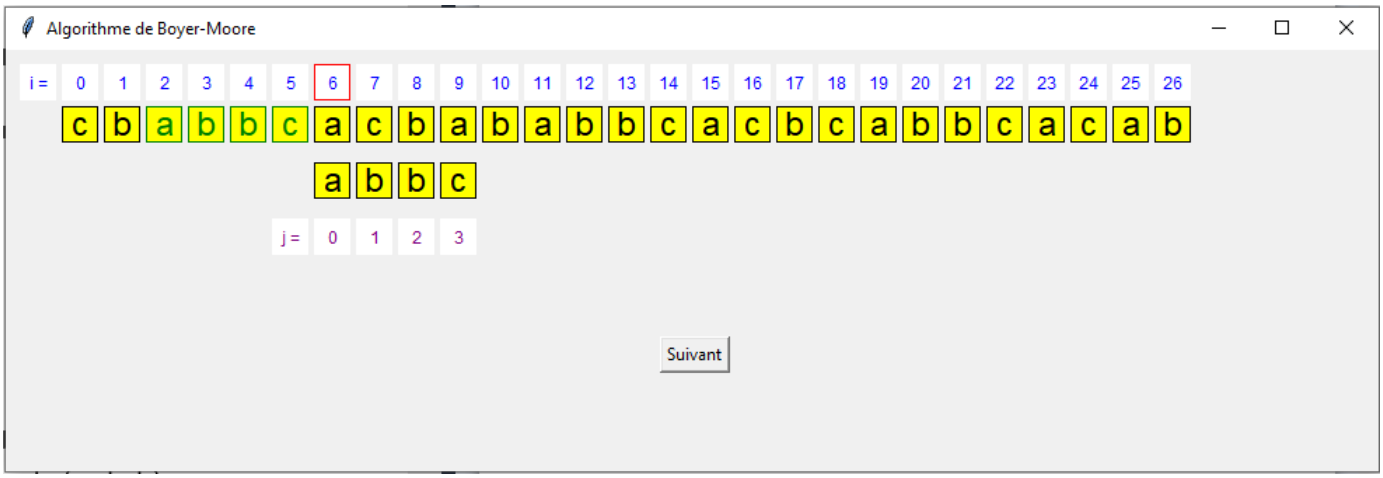
Remarque - lien vers le fichier ladisparition.txt : lien

```
# Dans l'éditeur PYTHON
FichierText = open("ladisparition.txt", "r") # ouverture en mode lecture
texte=FichierText.read()
mot='les'
```

Compléments

5 Compléments - Algorithme de Boyer-Moore-Horspool : une visualisation

— Boyer-Moore : visualisation Python 1 : lien https://repl.it/@fduffaud/Boyer-Moore-animation1
Un programme sous Python proposant une visualisation des étapes (sous tkinter) de l'algorithme.



6 Compléments - Algorithme de Boyer-Moore-Horspool : une amélioration

6.1 Principe

L'idée est d'améliorer la table des sauts afin d'optimiser les déplacements. Par exemple avec le motif **cdbca**, la table des sauts est :

c	b	d
1	2	3

Donc dans le cas suivant

texte :

a	b	b	a	c	c	a	b	a	c	b	a

motif :

c	d	b	c	a

On va directement faire un saut de 2 au lieu d'un saut de 1 avec Horspool puisque avec la table de sauts on avait $1-2=-1$.

texte :

a	b	b	a	c	c	a	b	a	c	b	a

motif :

c	d	b	c	a

Pour plus de détails, voici une présentation.

6.2 Une vidéo explicative

— Présentation 3 : lien
Boyer-Moore-Horspool avec plusieurs tables de sauts, une vidéo.

6.3 Une animation Python des algorithmes : naïf, Boyer-Moore et Horspool

— Programme Python : lien animation 2
Une visualisation sous Python des différentes comparaisons effectuées via l'algorithme naïf, puis via Boyer-Moore Horspool et Boyer-Moore avec plusieurs tables des sauts. Visuellement très pédagogique.

Fin du devoir

Corrections

Correction de l'exercice 1 : algorithme naïf

```
# Dans l'éditeur PYTHON
def recherche_naive(motif,texte):
    ''' In  : motif une chaine de caractère à chercher dans texte (le texte)
        Out : le nombre d'occurrences du mot cherché (0 si aucun)'''
    assert len(motif) <= len(texte)
    m=len(motif)
    n=len(texte)
    compteur=0
    for s in range(n-m+1)  :
        if motif==texte[s:s+m]:
            compteur=compteur+1
            print("Le motif est trouvé avec le décalage",s,
            ", nombre d'occurrences =", compteur)
    return compteur
```

Piste de discussion pour la partie 2.5 : optimiser la recherche naïve

Ça dépend de l'objectif de l'algorithme... On peut énoncer la recherche très particulière d'un motif de type 'mot' dans un texte. Le décalage après chaque comparaison, fructueuse ou non, peut aller jusqu'au prochain espace puisque chaque mot en est précédé d'un. On peut ainsi optimiser la recherche naïve en prenant un peu de recul sur les conditions dans lesquelles on se trouve.

Correction de l'exercice 2 : table des sauts

```
# Dans l'éditeur PYTHON
def table_sauts(motif):
    global T
    T={}
    i=0
    for lettre in motif[:-1]: # tout sauf le dernier
        T[lettre]=len(motif)-i-1
        i=i+1
    return T
```

Correction de l'exercice 3 : Boyer-Moore-Horspool

=> Disponible à l'adresse : https://repl.it/@fduffaud/Boyer-Moore-Correction

```
# Dans l'éditeur PYTHON
def bmh(texte, motif):
    '''In : texte et motif des chaines de caractères
       Out : une liste de la position des occurrences du motif dans le texte'''
    assert  len(texte)>= len(motif)
    # On récupère la taille de chaque chaine,
    size = len(texte)
    taille = len(motif)
    #Création d'une liste pour récupérer les positions des motifs trouvés
    positions = []
    #Si la taille du motif est inférieur ou égale à celle du texte
    if(taille<=size):
        #Remplissage de la liste: création de la table des sauts
        decalage = table_sauts(motif)
        i=0
        compteur=0
        trouve=False # variable booleenne qui vaut True si on a trouvé un mot
        #tant que l'indice i est inférieur à la taille du texte moins celle du motif
        while(i<=size-taille):
            #On teste en partant de la droite la correspondance des caractères
            du motif avec ceux du texte
            for j in range (taille-1,-1,-1):
                compteur=compteur+1
                trouve=True
                if(texte[i+j]!=motif[j]):
                    if(texte[i+j] in decalage and decalage[texte[i+j]]<=j):
                            i+=decalage[texte[i+j]]
                    else:
                        #Décalage du reste du motif
                        i+=j+1
                    trouve=False
                    break

            #Si tous les caractères correspondent
            if(trouve):
                #On ajoute la position de la portion du texte
                positions.append(i)
                #On décale de 1
                i=i+1
                trouve=False # on reinitialise la variable

            #On affiche toutes les positions trouvées

        print(size," ", compteur)
    return positions
```

Code du 6.2 : Boyer-Moore-Horspool amélioré

=> Disponible à l'adresse : https://repl.it/@fduffaud/Boyer-Moore-multitables

```
#*****************************************************************************
# Boyer-Moore-Horspool avec plusieurs tables de sauts (amélioration de Horspool)
#*****************************************************************************
def horspool(texte, motif):
    '''In : texte et le motif à chercher
       Out : nombre d'occurrences, position et nombres de tests effectués'''
    # On récupère la taille de chaque chaîne
    size = len(texte)
    taille = len(motif)
    #Création d'une liste pour récupérer les positions des motifs trouvés dans le texte
    occurences = []
    #Si la taille du motif est inférieur ou égale à celle du texte
    if(taille<=size):
        #Création d'une liste de dictionnaire
        tab = []
        #*******Tables Sauts*******
        #Remplissage de la liste: création de la table des sauts
        # tab va contenir toutes le stables de sauts sous forme de dictionnaires
        for j in range(taille):
            decalage = {}
            for i in range(taille-1-j):
                decalage[motif[i]]=taille-1-j-i
            decalage[0]=taille-j
            tab.append(decalage)
        #**************************
        #On affiche la table des sauts
        for t in range(len(tab)):
            print(tab[t])
        i=0
        compteur=0
        trouve=False
        #tant que l'indice i est inférieur à la taille du texte moins celle du motif
        while(i<=size-taille):
            #On teste en partant de la droit la correspondance des caractères du motif
            avec ceux du texte
            for j in range(taille-1,-1,-1):
                compteur=compteur+1 # on compte le nb de tests
                trouve=True
                #Si les caractères ne correspondent pas
                if(texte[i+j]!=motif[j]):
                    #Si le caractère testé dans le texte est dans le reste du motif
                    if(texte[i+j] in tab[taille-j-1]):
                        i=i+tab[taille-j-1][texte[i+j]]
                    else:
                        i=i+tab[taille-j-1][0]
                    trouve=False
                    break
            #Si tous les caractères correspondent
            if(trouve):
                #On ajoute la position de la portion du texte
                occurences.append(i)
                #On décale de 1
                i=i+1
                trouve=False
            #On affiche toutes les positions trouvées
    return len(occurences),occurences,compteur
```