Recherche de sous-chaîne

Introduction :

Nous utilisons quotidiennement des fonctionnalités de recherche d'un terme ou d'une expression, dans un texte ou dans un ensemble de documents. Cette capacité à trouver rapidement et efficacement une expression textuelle dans un ensemble parfois très vaste s'appuie sur des algorithmes de recherche textuelle ayant fait l'objet d'un certain nombre d'optimisations.

Nous présenterons dans un premier temps le fonctionnement d'une recherche naïve, afin de bien mesurer les optimisations qu'il est possible d'y apporter. Puis nous nous attèlerons à l'étude de l'algorithme de Boyer-Moore, algorithme de référence en la matière.

1 Recherche naïve

Il est assez facile de mettre au point un algorithme pour vérifier la présence d'une **sous-chaîne au sein d'une chaîne de caractères**. Cette sous-chaîne, également appelée **clé** ou **motif**, peut être absente ou présente. Si elle est présente dans le texte, elle peut l'être une ou plusieurs fois.

a. Principe de recherche

Il nous suffit de parcourir le texte séquentiellement à la recherche du motif. Schématiquement on compare les premiers caractères de la chaîne à ceux formant le motif.

Exemple

TEXTE DANS LEQUEL LE MOTIF RECHERCHE EST PEUT-ETRE PRESENT
MOTIF

S’il y a correspondance des premiers caractères, on vérifie à la suite les caractères suivants du motif. Si tous correspondent en regard du texte, on a trouvé une occurrence du motif dans le texte, dont on peut relever la position.

À retenir

Le texte est évalué sur la longueur du motif recherché. Cette portion de texte est appelée **fenêtre**.

S’il n’y a pas correspondance exacte, on se positionne sur le caractère suivant de la chaîne, et on compare à nouveau les caractères sur la longueur du motif. On procède ainsi, par décalage successifs, jusqu’à atteindre la fin du texte.

La fenêtre d’évaluation du texte se déplace donc par rapport au texte. On appelle **fenêtre glissante** ce mécanisme de décalages successifs permettant de parcourir le texte à la recherche du motif.

Exemple

Décalages successifs du motif par rapport au texte

TEXTE DANS LEQUEL LE MOTIF RECHERCHE EST PEUT-ETRE PRESENT
MOTIF

TEXTE DANS LEQUEL LE MOTIF RECHERCHE EST PEUT-ETRE PRESENT
MOTIF
TEXTE DANS LEQUEL LE MOTIF RECHERCHE EST PEUT-ETRE PRESENT
MOTIF

(décalages intermédiaires non montrés, jusqu'à la découverte d'une occurrence du motif recherché)

TEXTE DANS LEQUEL LE MOTIF RECHERCHE EST PEUT-ETRE PRESENT
MOTIF

(suite de décalages intermédiaires non montrés, jusqu'aux derniers décalages)

```
TEXTE DANS LEQUEL LE MOTIF RECHERCHE EST PEUT-ETRE PRESENT
                                                    MOTIF
TEXTE DANS LEQUEL LE MOTIF RECHERCHE EST PEUT-ETRE PRESENT
                                                          MOTIF
```

Dans une recherche naïve :

- le texte est parcouru par déplacements de la fenêtre glissante, de gauche à droite ;
- le contrôle du motif est effectué lettre à lettre, de gauche à droite.

b. Implémentation

L'implémentation de cet algorithme de recherche est assez simple. Elle consiste à imbriquer deux boucles, et à relever les positions d'éventuelles occurrences, qu'on retourne sous forme d'une liste d'indices (éventuellement vide si le motif recherché n'est pas présent).

```
def recherche(motif, texte):
    positions = []
    for i in range(len(texte) - len(motif)):
        for j in range(len(motif)):
            if not texte[i + j] == motif[j]:
                break
        if j == len(motif) - 1:
            positions.append(i)
    return positions
```

Astuce

> On peut également utiliser la construction `for` assortie de la clause optionnelle `else`, cette dernière n'étant exécutée que si la boucle s'est terminée normalement, sans avoir été interrompue par un `break`.

```
def recherche(motif, texte):
    positions = []
    for i in range(len(texte) - len(motif)):
```

```
            for j in range(len(motif)):
                if not texte[i + j] == motif[j]:
                    break
            else:
                positions.append(i)
        return positions
```

→ Comme le montre notre implémentation, il est possible d'obtenir une fonctionnalité de recherche d'un motif dans un texte en quelques lignes de code.

Pour cela nous testons toutes les positions individuelles possibles pour le motif par rapport au texte complet. C'est une approche dite par « force brute », également appelée **recherche naïve**.
Sa complexité est au mieux linéaire et dans le pire des cas $O(n \times m)$, où m est la longueur du motif et n celle du texte.

Les approches par force brute exploitent la puissance de la machine pour produire un résultat relativement rapide. Mais, quelle que soit la puissance de la machine, le résultat sera plus lent à obtenir qu'avec un algorithme optimisé, quand une optimisation est possible.

Nous allons réaliser avec l'étude de l'algorithme de Boyer-Moore que différentes optimisations sont possibles pour ce type de recherche.

2 | Algorithme de Boyer Moore

Les algorithmes de recherche de sous-chaînes se comptent en dizaines. En la matière, l'**algorithme de Boyer-Moore** fait figure de référence. Il porte le nom de ses deux co-inventeurs.

Robert Stephen Boyer et J Strother Moore sont deux informaticiens américains. Ensemble ils ont proposé en 1977 un algorithme de recherche de sous-chaîne particulièrement efficace. Robert Stephen Boyer travaillait alors au laboratoire de science informatique de Stanford, et J Strother Moore au centre de recherche Xerox de Palo Alto, tous deux situés en Californie.

Leur algorithme a ensuite fait l'objet de nombreuses variantes et adaptations, dont l'algorithme de Boyer-Moore-Horspool, parfois appelé **algorithme de Horspool**. Ce dernier, publié en 1980, constitue une version simplifiée de l'algorithme de Boyer-Moore. On le doit à l'informaticien britannique Nigel Horspool, alors enseignant à l'université canadienne de Victoria.

À retenir

> L'algorithme de Boyer-Moore sert aujourd'hui de référence pour évaluer la performance des autres algorithmes de recherche de sous-chaîne.

a. Principe général

Cet algorithme est capable d'éviter de nombreuses comparaisons inutiles, et s'avère bien plus performant que la recherche naïve.
L'approche par force brute de la recherche naïve nous faisait tester de manière successive toutes les positions possibles du motif dans le texte, de manière indépendante, sans jamais exploiter les conséquences d'éventuelles correspondances ou non-correspondances observables dans la position qui précédait la position courante de la fenêtre glissante.

L'algorithme inventé par Boyer et Moore prend en compte les occurrences des lettres composant le motif et en déduit des comparaisons inutiles, qu'il ne sera pas nécessaire d'effectuer.
Les déplacements du motif s'effectuent de gauche à droite comme dans l'algorithme naïf, mais celui de Boyer-Moore présente la particularité d'évaluer le motif de la droite vers la gauche.

À retenir

> Dans l'algorithme de Boyer-Moore :
>
> - le texte est parcouru de gauche à droite ;
> - le contrôle du motif est effectué de droite à gauche.

Le motif fait également l'objet d'un prétraitement, sur lequel nous reviendrons dans la troisième partie du cours.

b. Exemple introductif

Voici un exemple introductif illustrant la capacité de l'algorithme de Boyer-Moore à éliminer des comparaisons inutiles.

Exemple

Motif :
VERITE

Phrase :
CORRELATION N'EST PAS CAUSALITE

L'algorithme de Boyer-Moore commence par contrôler la fin du motif (matérialisé ici par un curseur au-dessus du texte).

La comparaison entre le E du motif et le L du texte permet de constater une non-correspondance.

Là où l'algorithme naïf aurait simplement décalé le motif d'un caractère avant de reprendre l'analyse à l'identique, l'algorithme de Boyer-Moore s'appuie sur l'analyse du motif : celle-ci indique l'absence totale d'occurrences du caractère « L » dans le motif « VERITE ».

→ Cette absence de la lettre « L » dans le motif garantit qu'on ne pourra trouver aucune correspondance, quelle que soit la manière dont on décalera le motif en regard de la position courante d'évaluation.

En conséquence, il est totalement inutile de vérifier les positions suivantes sur toute la longueur du motif. On décale donc le motif vers la droite jusqu'à le positionner au-delà du caractère comparé dans le texte, économisant tous les décalages intermédiaires et toutes les comparaisons caractère par caractère qui y auraient été associées. Après décalage, le motif est dans la position suivante :

L'algorithme naïf ne tirant aucune conclusion d'une non-correspondance aurait simplement décalé le motif d'une position :

L'algorithme de Boyer-Moore a permis d'éliminer cinq positions inutiles de la fenêtre glissante du motif, évitant autant de comparaisons pour chaque lettre du motif à chacune des positions évitées.

→ Cet exemple introductif permet d'entrevoir tout l'intérêt de l'algorithme inventé par Boyer-Moore.

Nous avons souligné la particularité de cet algorithme qui effectue la vérification du motif en sens inverse du déplacement de la fenêtre glissante. Nous allons maintenant décrire de manière plus formelle les deux règles complémentaires de l'algorithme de Boyer-Moore : la **règle du mauvais caractère** et la **règle du bon suffixe**.

c. Règle du mauvais caractère

La règle du mauvais caractère consiste à éliminer des positions considérées comme impossibles pour le motif, en regard du caractère évalué.

Lors de l'évaluation d'un caractère donné dans le texte où l'on constate une non-correspondance, autrement dit un « mauvais caractère », deux cas de figure sont possibles :

- soit le caractère du texte est totalement absent du motif recherché, ce qui signifie qu'aucune des positions du motif, en regard de ce caractère du texte, ne pourra aboutir à une correspondance, auquel cas on décale au-delà de la position considérée ;
- soit le caractère du texte est présent, une ou plusieurs fois, ailleurs dans le motif, auquel cas on décale le motif jusqu'à alignement de la première

occurrence du motif avec le caractère du texte, en parcourant le motif de la droite vers la gauche.

Nous avons déjà vu dans l'exemple introductif le cas de l'absence du caractère recherché dans le motif. Voyons un autre exemple où celui-ci est présent ailleurs dans le motif.

Exemple

Nous recherchons le motif IMPLEMENTATION dans le texte suivant :
LA REALISATION INFORMATIQUE PAR PROGRAMMATION CONSTITUE UNE IMPLEMENTATION LOGICIELLE.

Après quelques décalages, le motif est positionné de la manière suivante par rapport au texte (le curseur indique le point de comparaison courant) :

→ Le I ne correspond évidemment pas au N.

En appliquant la règle du mauvais caractère, nous recherchons s'il existe des occurrences de la lettre I dans le motif. Il en existe deux. Le motif peut être décalé jusqu'à aligner le caractère du texte avec la première occurrence en partant de la droite dans le motif.

Le motif a été décalé de deux positions, sans avoir à évaluer la position intermédiaire. Reprenant la comparaison à partir de la fin du motif, on constate immédiatement une non-correspondance entre le U du texte et le N du motif. Le U étant totalement absent du motif, ce dernier peut être décalé au-delà du curseur.

LA REALISATION INFORMATIQUE PAR PROGRAMMATION CONSTITUE UNE IMPLEMENTATION LOGICIELLE.

IMPLEMENTATION

La lettre N du motif ne correspond pas au M du texte mais le caractère « M » est présent dans le motif. Ce dernier est donc décalé pour mettre le M en correspondance.

⇩

LA REALISATION INFORMATIQUE PAR PROGRAMMATION CONSTITUE UNE IMPLEMENTATION LOGICIELLE.

IMPLEMENTATION

→ Sans poursuivre jusqu'au bout du texte, on peut observer que l'application de la règle du mauvais caractère permet d'effectuer d'importants décalages, éliminant un nombre significatif de traitements.

La règle du mauvais caractère aboutit à un décalage plus ou moins important du motif selon la présence ou non d'occurrences du caractère comparé dans le motif.
Une fois le décalage effectué, la comparaison reprend à partir de la fin du motif, toujours de la droite vers la gauche.

L'algorithme de Boyer-Moore comporte une autre règle liée à l'analyse du motif, la règle du bon suffixe.

d. Règle du bon suffixe

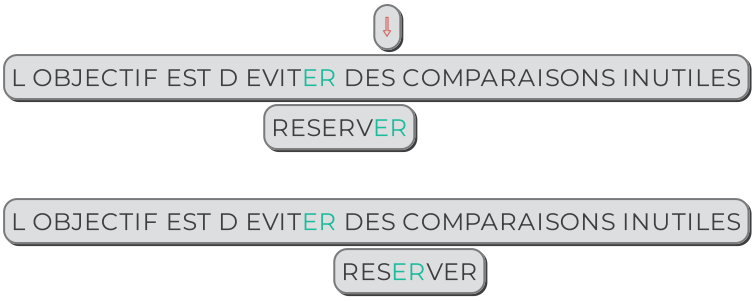
La comparaison entre le motif et le texte à une position donnée s'effectuant de la droite vers la gauche, s'il existe une correspondance au moins partielle, elle se matérialise à partir de la fin du texte et du motif, autrement dit au niveau du suffixe.

→ La règle du bon suffixe de l'algorithme de Boyer-Moore indique comment procéder quand un bon suffixe est précédé d'un caractère qui ne correspond pas entre le texte et le motif.

On recherche dans le motif, en allant de la droite vers la gauche, s'il existe une autre occurrence du suffixe qui soit précédée par un autre caractère que celui qui ne correspond pas. On s'arrête dès qu'une occurrence est

trouvée et on décale le motif pour mettre cette occurrence en correspondance avec le suffixe du texte.

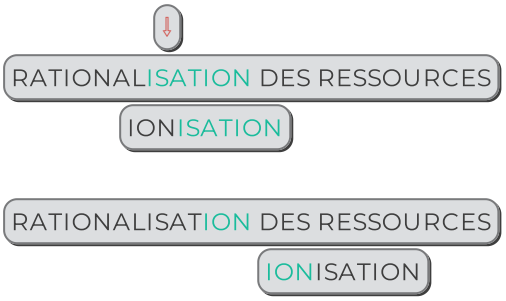
Exemple



Le suffixe -ER du texte figure ailleurs dans le motif, précédé non pas d'un V comme dans la comparaison initiale qui a échoué, mais précédé d'un S. On effectue donc le décalage correspondant.

S'il n'existe pas d'autre occurrence du suffixe dans le motif, on le décale jusqu'à la mise en correspondance du plus long suffixe possible qui corresponde à un préfixe du motif.

Exemple



S'il n'existe pas de préfixe du motif qui puisse correspondre à un suffixe du texte, on décale le motif au-delà du suffixe.

Exemple

OPTIMISATION

RATIONALISATION DES RESSOURCES
OPTIMISATION

Il existe également un cas particulier où le suffixe du texte correspond à la totalité du motif.

e. Découverte d'une occurrence

La découverte d'une **occurrence**, c'est-à-dire d'une correspondance totale entre le texte et le motif, est un cas particulier de la règle du bon suffixe, où le bon suffixe est aussi long que le motif recherché.
On applique alors les mêmes règles du bon suffixe.

Exemple

RATIONALISATION DES RESSOURCES
RATIONALISATION

RATIONALISATION DES RESSOURCES
RATIONALISATION

Attention

La découverte d'une occurrence complète n'entraîne pas toujours un décalage au-delà de la longueur du motif. En effet il est possible qu'un préfixe du motif puisse correspondre à un suffixe du bon suffixe.

Exemple

UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT
QUELCONQUE

UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT

QUELCONQUE

f. Maximisation des décalages

Chacune des deux règles (celle du mauvais caractère et celle du bon suffixe) permet d'effectuer des décalages parfois importants du motif sans risquer de manquer une occurrence de celui-ci.

L'algorithme de Boyer-Moore combine ces deux règles pour maximiser les décalages. Pour une position donnée du motif par rapport au texte, il détermine le décalage proposé par chacune des deux règles, et favorise celle qui permet le plus grand décalage.

Nous illustrons cette maximisation des décalages avec l'exemple suivant :

Exemple

Nous recherchons le motif QUELCONQUE dans le texte : UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT

```
          ⇩
UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT
QUELCONQUE
```

Le R du texte ne correspond pas au E final du motif.

- La règle du mauvais caractère propose un décalage de 10 positions.
- La règle du bon suffixe ne s'applique pas.

→ Le décalage appliqué est celui proposé par la règle du mauvais caractère.

```
UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT
          QUELCONQUE
```

- La règle du mauvais caractère propose un décalage de 1 position.
- La règle du bon suffixe ne s'applique pas.

→ Le décalage appliqué est celui proposé par la règle du mauvais caractère.

UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT
QUELCONQUE

- La règle du mauvais caractère ne s'applique pas.
- La règle du bon suffixe propose un décalage de 7 positions (du fait de la correspondance du préfixe du motif avec une partie du suffixe du motif, en l'occurrence « QUE »).

→ Le décalage appliqué est celui proposé par la règle du bon suffixe.

UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT
QUELCONQUE

- La règle du mauvais caractère propose un décalage de 6 positions.
- La règle du bon suffixe propose un décalage de 7 positions.

→ Le décalage appliqué est celui proposé par la règle du bon suffixe.

UNE VALEUR QUELCONQUE PASSEE EN ARGUMENT
QUELCONQUE

- La règle du mauvais caractère propose un décalage de 10 positions.
- La règle du bon suffixe ne s'applique pas.

→ Le décalage proposé par la règle du mauvais caractère termine l'algorithme.

Cet exemple illustre l'efficacité de l'algorithme qui évite de nombreuses comparaisons inutiles grâce à l'application des deux règles.

Nous avons pris conscience de l'efficacité de l'algorithme de Boyer-Moore. Cependant, celui-ci peut se prêter à des optimisations qui en amélioreront encore la performance.

3 | Optimisation algorithmique

Le recours à des prétraitements permet d'améliorer significativement la performance des algorithmes de recherche textuelle. On distingue deux approches :

- le prétraitement du motif ;
- le prétraitement du texte.

L'algorithme de Boyer-Moore effectue un prétraitement du motif.

a. Prétraitements du motif

Le prétraitement du motif vise à éviter d'effectuer de manière répétitive les mêmes calculs, en stockant leurs résultats afin de les réutiliser ensuite. Ce prétraitement est particulièrement adapté aux recherches d'un même motif dans plusieurs textes différents.
Les résultats de ces calculs sont stockés dans des **tables de sauts** générées en amont de la recherche proprement dite. Ces tables de saut indiquent l'ampleur du décalage à effectuer en fonction des lettres rencontrées dans le texte, en appliquant des règles de l'algorithme.

Dans sa version complète, l'algorithme de Boyer-Moore fait appel à deux tables de saut, l'une pour la règle du mauvais caractère et l'autre pour la règle du bon suffixe.
Dans sa variante de Horspool, l'algorithme utilise uniquement la table de saut correspondant à la règle du mauvais caractère, dont nous allons présenter maintenant la construction.

b. Table de sauts du mauvais caractère

La table de sauts du mauvais caractère transpose l'application de la règle du mauvais caractère à **l'alphabet des caractères possiblement présents dans le texte**. Cette table indiquera l'ampleur du décalage à effectuer en fonction du caractère rencontré dans le texte.
Il existe plusieurs méthodes équivalentes pour la génération de cette table de saut. On peut procéder :

- en traitant le motif à partir de la fin ;
- en listant successivement les lettres composant le motif ;

- et en notant le nombre de décalages par rapport à la fin du motif.

S'il existe plusieurs occurrences d'une même lettre, on ne prend en compte que la première occurrence rencontrée (dans le sens de lecture du motif de droite à gauche).
Les lettres absentes du motif, représentées collectivement par un caractère « * », entraineront un décalage de la longueur du motif complet.


Exemple

La table de sauts correspondant au motif OPTIMISATION est la suivante (les lettres ont été laissées par ordre d'ajout, dans le sens de lecture du motif de droite à gauche) :

lettre	décalage
O	1
I	2
T	3
A	4
S	5
M	7
P	10
*	12


Attention

Le caractère « N » fait partie du cas * car il n'existe pas d'autres occurrences de « N » dans le motif que celle de la dernière lettre qui, elle, ne compte pas, car elle correspond à un saut de zéro.

c. Performance

L'algorithme de Boyer-Moore s'avère très efficace. Sa complexité est en général sous-linéaire. Sa performance est d'autant plus appréciable que le motif est long, gagnant ainsi à être prétraité.

Dans certains cas très particuliers, la performance de l'algorithme de Boyer-Moore peut se dégrader au niveau de l'approche par force brute.

Exemple

Texte :
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Motif :
Z

1. « A » diffère de « Z ». C'est la règle du mauvais caractère qui s'applique : on décale la fenêtre de la longueur total du motif, c'est-dire de 1.

ABCDEFGHIJKLMNOPQRSTUVWXYZ
Z

2. « B » diffère de « Z ». C'est la règle du mauvais caractère qui s'applique : on décale la fenêtre de la longueur total du motif, c'est-dire de 1.

ABCDEFGHIJKLMNOPQRSTUVWXYZ
Z

On comprend que l'algorithme va décaler la fenêtre 1 caractère par 1 caractère, comme l'aurait également fait la recherche naïve.

Astuce

Ce site propose une animation illustrant le fonctionnement de l'algorithme de Boyer-Moore étape par étape.

Conclusion :

La recherche de sous-chaînes est omniprésente en informatique. Les algorithmes exploitant la force brute sont assez « gourmands ». L'algorithme de Boyer-Moore est emblématique des optimisations possibles pour ce type de traitement. Nous avons présenté son principe général et ses deux règles régissant le décalage du motif que sont la règle du mauvais caractère et la règle du bon suffixe. Nous avons ensuite décrit l'intérêt et les modalités des prétraitements pour ce type d'algorithme avec le recours à des tables de sauts générées en amont de la recherche proprement dite.