1 Problématique

Rechercher les occurrences d'un mot dans un texte est une fonctionnalité intégrée dans tous les logiciels de traitements de texte. Si la tâche semble aisée dans un texte d'une seule page, nous pouvons nous interroger sur l'efficacité de la recherche d'un mot dans un livre entier.

Comment effectuer une recherche textuelle efficace ?

2 Approche naïve

La première méthode à laquelle nous pouvons penser consiste à :

— observer une **fenêtre** du texte,

— dans cette fenêtre, comparer chaque lettre du **motif** recherché au texte,

— décaler la fenêtre d'un cran dès qu'il n'y a pas de correspondance.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	d	r	a	c	a	d	a	b	r	a
motif	d	a	b								
	0	1	2								

FIGURE 1 – Première comparaison : pas de correspondance

	0	1	2	3	4	5	6	7	8	9	10
texte	a	d	r	a	c	a	d	a	b	r	a
motif		d	a	b							
		0	1	2							

FIGURE 2 – Première comparaison : correspondance

	0	1	2	3	4	5	6	7	8	9	10
texte	a	d	r	a	c	a	d	a	b	r	a
motif		d	a	b							
		0	1	2							

FIGURE 3 – Deuxième comparaison : pas de correspondance

2.1 Implémentation

Activité 1 :

1. Écrire la fonction `recherche_naive(texte: str, motif: str)` → `int` qui renvoie la position du *motif* dans le *texte* ou -1 s'il n'est pas présent.
2. Estimer la complexité temporelle de cet algorithme dans le pire des cas : le motif n'est pas présent dans le texte.

3 Approche plus efficace : Boyer-Moore

3.1 Recherche à l'envers

La première idée de cet algorithme est de commencer la recherche **en partant de la fin du motif**.

Christophe Viroulaud

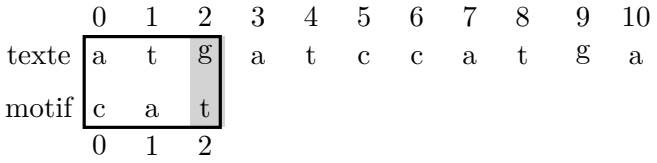

FIGURE 4 – Première comparaison : pas de correspondance

Pour l'instant cette approche ne semble par apporter d'amélioration par rapport à l'algorithme précédent.

3.2 Décalages par sauts

Si on s'intéresse au motif, on peut remarquer qu'il ne contient pas la lettre **g** (la dernière lettre de la fenêtre). Les comparaisons 5 et 6 sont donc inutiles.

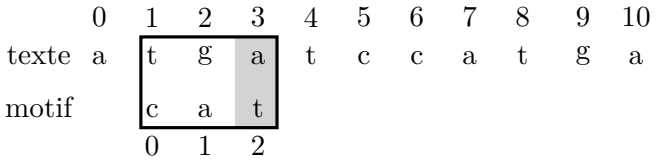

FIGURE 5 – Comparaison inutile

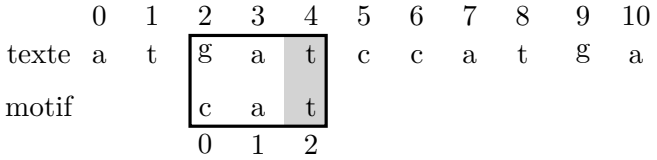

FIGURE 6 – Comparaison inutile

On peut donc directement décaler le motif à l'indice 3 du texte (figure 7).

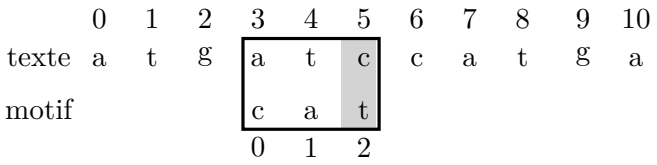

FIGURE 7 – Décalage par saut

On n'observe pas de correspondance par contre la lettre **c** existe dans le motif. On va donc le décaler pour les faire coïncider (figure 8).

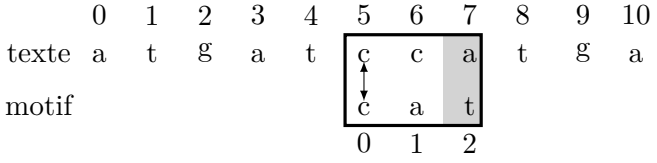

FIGURE 8 – Décalage par saut

À retenir

On décale la position de recherche dans le texte en fonction de la dernière lettre de la fenêtre.

Christophe Viroulaud

3.3 Prétraitement du motif

Pour pouvoir décaler par saut, il faut connaître la dernière position de chaque lettre dans le motif. Le prétraitement consiste à calculer le décalage à appliquer pour amener chaque caractère du motif à la place du dernier caractère.

Remarque

On ne regarde pas la dernière position de la clé (la lettre t ici). Sinon la distance associée serait nulle et on resterait sur place après l'avoir lue dans le texte.

Activité 2 : Écrire la fonction `pretraitement_decalages(motif: str)` → `dict` qui associe chaque lettre du motif (sauf la dernière) à son décalage.

3.4 Algorithme de Boyer-Moore (simplifié - version Horspool)

L'algorithme de Boyer-Moore s'écrit alors :

```
Créer le tableau des décalages
Tant qu'on n'est pas à la fin du texte
    Comparer le motif à la position du texte
    Si le motif est présent
        Renvoyer la position
    Sinon
        Décaler la fenêtre
Renvoyer -1 si le motif n'est pas présent
```

Code 1 – Algorithme de Boyer-Moore (version Horspool)

Activité 3 :

1. Écrire la fonction `compare(texte: str, position: int, motif: str)` → `bool` qui renvoie *True* si le motif est présent à la position i du texte.
2. Écrire la fonction `decalage_fenetre(decalages: dict, taille: int, lettre: str)` → `int` qui renvoie le décalage à appliquer pour faire coïncider le motif à la dernière *lettre* de la fenêtre. Si la lettre n'est pas présente, la *taille* du motif est renvoyée.
3. Écrire alors la fonction `boyer_moore(texte: str, motif: str)` → `int` qui renvoie la position du motif dans le texte et -1 sinon.

3.5 Complexité

Intuitivement l'algorithme semble plus rapide que la version naïve car il ne teste pas toutes les lettres du texte.

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

FIGURE 9 – Un cas représentatif

Christophe Viroulaud

Activité 4 : Compter le nombre d'itérations de la recherche avec l'algorithme naïf puis celui de Boyer-Moore.

Remarques

— Dans le meilleur des cas, la complexité temporelle de l'algorithme est $O(N/K)$ où N est la taille du texte et N celle du motif.

— Plus le motif est long plus l'algorithme est rapide.

Christophe Viroulaud