**République Algérienne Démocratique et Populaire Ministère de l'Enseignement Supérieur et de la Recherche Scientifique Université des Sciences et de la Technologie Houari Boumediene**


Faculté d'informatique - Département d'informatique

Rapport en Bio-ALGO
Master 1 en Bioinformatique
G1

## Thème

# Recherche exacte de motifs

**Nom et Prénom :**

BOUBRIMA Ali

MAMMERI Sara

**Chargé de Cours** : Mme BOUKHEDOUMA

# Introduction :

La correspondance ou la recherche de modèles de chaînes consiste à vérifier la présence des constituants d'un motif donné dans un texte donné où le motif et le texte sont des chaînes sur un certain alphabet.

C'est un composant important de nombreux problèmes et il est utilisé dans de nombreuses applications telles que le texte édition, récupération de données, filtrage de données (également appelé exploration de données) pour trouver des modèles sélectionnés, ADN la mise en correspondance de séquences, la détection de certains mots-clés suspects dans les applications de sécurité, et bien sûr, de nombreuses autres applications.

Le problème de recherche de chaînes ou de correspondance de chaînes consiste à trouver toutes occurrences d'un ensemble de motifs dans un texte, où le motif et le texte sont des chaînes sur certains alphabets. Les problèmes d'appariement de modèles de chaînes multiples ont fait l'objet de recherches intensives qui ont a abouti à plusieurs approches pour la solution telles que la généralisation à mots clés multiples de l'algorithme de Boyer Moore, algorithme Aho-Corasick, algorithme Commentz-Walter et algorithme Rabin-Karp.

# Partie1 :

## 1. Boyer-Moore

### 1.1. Définition :

L'algorithme de Boyer Moore est un algorithme de recherche de motif dans une chaine de caractères. Il a été développé par Robert S. Boyer et J Strother Moore en 1977.

Boyer-Moore est utilisé dans des textes de grande taille et dans nombreuse application comme les applications des « études de textes » qui cherchent des motifs, aussi il est bien utilisé dans la recherche de virus informatique.

Cet algorithme se repose sur 2 étapes principales :

- la première consiste à utiliser une heuristique de saut, appelée "règle du mauvais caractère", qui permet de sauter plusieurs caractères lorsqu'un caractère ne correspond pas dans le motif lors de la recherche dans le texte.

- la deuxième idée est l'utilisation de la "règle du bon suffixe", qui permet de déterminer si une occurrence partielle d'un motif M dans le texte T peut être étendue en une occurrence complète sans risque de manquer une occurrence complète.

### 1.2. Explication :

- La première étape consiste à calculer le tableau de saut qui permettra de sauter plusieurs caractères lorsqu'un caractère ne correspond pas dans le motif M lors de la recherche dans le texte T. Pour chaque caractère c, la valeur dans le tableau de saut est la distance à laquelle on doit sauter si ce caractère est le dernier caractère du motif M et qu'il ne correspond pas dans le texte T. Si c n'apparaît pas dans M, alors la valeur correspondante dans le tableau de saut est m.

- Ensuite, la recherche des occurrences du motif M dans le texte T se fait en commençant par comparer le dernier caractère du motif M avec le caractère à l'indice i dans le texte T. Si les caractères correspondent, on remonte dans le motif M et dans le texte T jusqu'à ce qu'on ait trouvé une occurrence du motif M ou que tous les caractères du motif M aient été vérifiés. Si tous les caractères du motif M correspondent, alors une occurrence est trouvée. On ajoute l'indice i à la liste des occurrences, on saute m caractères dans le texte T (puisque le motif M a été trouvé) et on recommence la recherche à partir de l'indice i+m-1. Si un caractère ne correspond pas, on utilise le tableau de saut pour sauter plusieurs caractères dans le texte T et on recommence la recherche à partir de l'indice i correspondant au caractère du texte T qui doit être comparé avec le dernier caractère du motif M.

- Enfin, il renvoie une liste de toutes les occurrences du motif M dans le texte T. Si le motif M ne se trouve pas dans le texte T, alors la fonction renvoie une liste vide.

## 1.3. Complexité :

Algorithme de Boyer-Moore : la complexité théorique de cet algorithme est O(n + m), où n est la longueur du texte et m est la longueur du motif. Dans le pire des cas, la complexité de l'algorithme est O(nm), mais elle est généralement bien inférieure à cela. L'algorithme de Boyer-Moore est réputé pour être l'un des algorithmes de recherche de motif les plus rapides pour un seul motif.

# 2. Aho-Corasick

## 2.1. Définition :

L'algorithme d'Aho-Corasick est un algorithme de recherche de motifs dans une chaîne de caractères. Il a été proposé en 1975 par Alfred V. Aho et Margaret J. Corasick. Cet algorithme permet de rechercher simultanément un ensemble de motifs dans une chaîne de caractères, ce qui en fait une solution efficace pour la recherche de plusieurs motifs dans de grands volumes de données.

L'algorithme d'Aho-Corasick fonctionne en construisant un automate fini déterministe (AFD) qui représente tous les motifs à rechercher. Cet automate est construit à l'aide d'un trie (arbre préfixe) et permet de rechercher tous les motifs en une seule passe dans le texte. L'algorithme d'Aho-Corasick utilise également la notion de liens de défaillance pour accélérer la recherche.

## 2.2. Explication :

La première étape consiste à construire un trie (un arbre préfixe) contenant tous les motifs à rechercher. Chaque nœud de l'arbre correspond à un préfixe de l'un des motifs et est étiqueté avec l'indice du motif correspondant dans la liste des motifs. Les nœuds qui correspondent à la fin d'un motif sont également étiquetés avec le symbole '#' suivi de l'indice du motif.

Ensuite, l'algorithme construit les liens de défaillance pour chaque nœud de l'arbre. Pour chaque nœud qui n'est pas la racine, l'algorithme cherche le lien de défaillance en remontant dans l'arbre jusqu'à ce qu'il trouve un nœud qui a un enfant avec l'étiquette correspondant au caractère manquant. Si aucun tel nœud n'est trouvé, le lien de défaillance pointe vers la racine de l'arbre. Enfin, l'algorithme calcule les sorties pour chaque nœud de l'arbre. La sortie pour un nœud est la liste des indices de tous les motifs qui peuvent être trouvés en suivant les liens de défaillance à partir de ce nœud et en ajoutant l'indice de motif correspondant au nœud courant s'il est étiqueté avec le symbole '#'.

Enfin, la recherche des motifs dans le texte T se fait en suivant :

L'algorithme parcourt le texte caractère par caractère en suivant les transitions de l'automate. Pour chaque caractère, l'algorithme suit la transition correspondante dans l'automate. Si un nœud étiqueté avec '#' est atteint, cela signifie qu'un motif a été trouvé à la position actuelle dans le texte. L'algorithme utilise ensuite les sorties pour détecter les autres occurrences du motif.

## 2.3. Complexité :

Algorithme de Boyer-Moore : la complexité théorique de cet algorithme est O(n + m),

n : est la longueur du texte

m : est la longueur du motif.

**Dans le pire des cas,** la complexité de l'algorithme est O(nm), mais elle est généralement bien inférieure à cela.

## 2.4. Exemple :

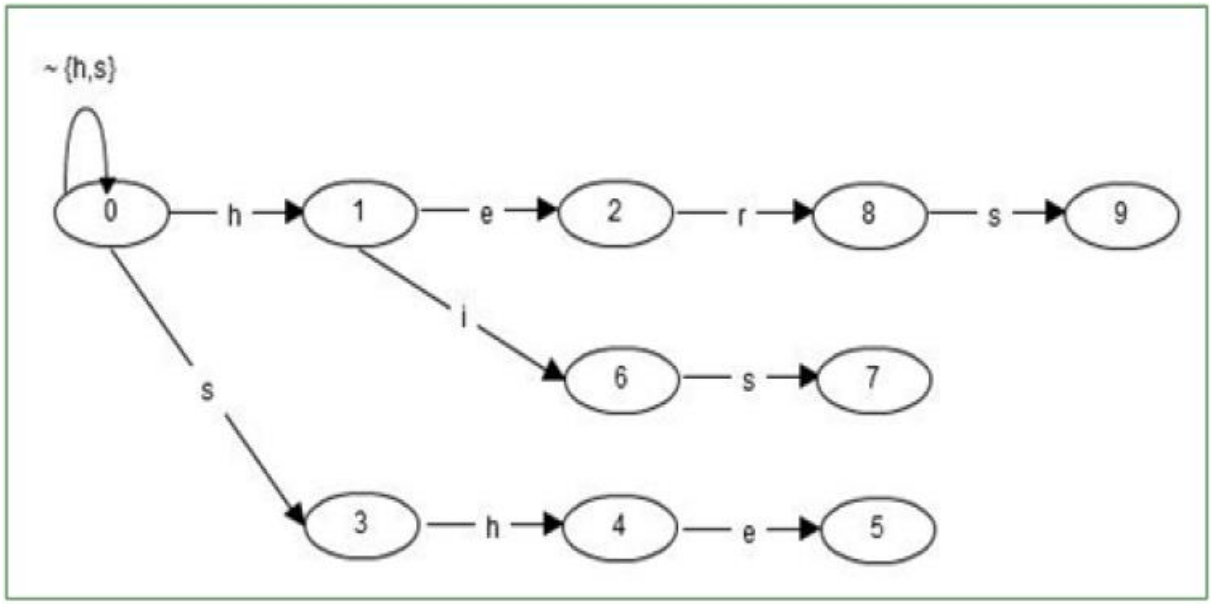


Figure1 : UN ÉCHANTILLON MACHINE D'APPARIEMENT DE MODÈLE.

## 2.5. Complexité :

L'algorithme d'Aho-Corasick a une complexité en temps linéaire O(n+m), où n est la longueur du texte et m est la longueur maximale des motifs. Cet algorithme est donc très efficace pour la recherche de plusieurs motifs dans de grands volumes de données.

# 3. Rabin-Karp [RK]:

## 3.1. définition :

L'algorithme de Rabin-Karp est un algorithme de recherche de motif dans une chaîne de caractères. Il a été proposé en 1987 par Richard M. Karp et Michael O. Rabin. Cet algorithme utilise la technique de hachage pour comparer rapidement les motifs avec des sous-chaînes de la chaîne de caractères.

L'utilisation de trois fonctions de hachage différentes (filtre de Bloom) permet d'augmenter la probabilité de trouver tous les motifs dans la chaîne de caractères et de réduire le risque de collision.

## 3.2. Explication :

Voici les étapes de l'algorithme de Rabin-Karp :

•Prétraitement : Calculer les valeurs de hachage pour chaque motif à rechercher à l'aide de trois fonctions de hachage (filtre de Bloom) différentes. Les fonctions de hachage doivent être choisies pour minimiser le risque de collision.

•Calcul des valeurs de hachage : Pour chaque sous-chaîne de longueur m de la chaîne de caractères, calculer les valeurs de hachage à l'aide des mêmes fonctions de hachage utilisées pour les motifs.

•Comparaison des valeurs de hachage : Comparer les valeurs de hachage des sous-chaînes avec les valeurs de hachage des motifs. Si les valeurs de hachage sont égales, comparer les sous-chaînes et les motifs caractère par caractère pour confirmer la correspondance.

•Gestion des collisions : Si les valeurs de hachage sont différentes, cela ne signifie pas nécessairement que la sous-chaîne ne correspond pas à un motif. Il peut y avoir une collision, c'est-à-dire que deux sous-chaînes différentes ont la même valeur de hachage. Pour gérer les collisions, il est nécessaire d'utiliser une méthode de vérification supplémentaire, comme la comparaison caractère par caractère.

•Recherche des motifs dans le texte : L'algorithme parcourt la chaîne de caractères caractère par caractère en comparant les valeurs de hachage des sous-chaînes avec les valeurs de hachage des motifs. Si une correspondance est trouvée, l'algorithme vérifie la correspondance caractère par caractère pour confirmer la présence du motif dans le texte.

## 3.3. Complexité :

L'algorithme de Rabin-Karp a une complexité en temps moyenne de O(n+m), où n est la longueur du texte et m est la longueur maximale des motifs. Cet algorithme est donc efficace

pour la recherche de plusieurs motifs dans des volumes de données de taille raisonnable. Cependant, en raison du risque de collision, cet algorithme peut ne pas trouver tous les motifs dans le texte.

# Partie2 :

## 4. Comparaison des 2 algorithmes (Aho-Corasick et Rabin-Karp) :

### 1. AHO-CARASICK

**A.** Ici on fixe la taille et on change le nombre de motifs comme suivant :

**Taille du texte : 107,675 lettres (108 KB)** *(le même fichier pour tous les tests)*

| *le nombre du motif* | *La taille du motif* | *Temps(ms)* |
|---|---|---|
| 1 | 3 | 103 |
| 2 | 3 | 110 |
| 3 | 3 | 120 |
| 4 | 3 | 124 |
| 5 | 3 | 126 |
| 6 | 3 | 136 |
| 7 | 3 | 151 |
| 8 | 3 | 166 |
| 9 | 3 | 175 |
| 10 | 3 | 183 |

TABLEAU 1 : TEMPS DE FONCTIONNEMENT D'AHO-CARASICK ALGORITHME SELON NOMBRE DE MOTIFS

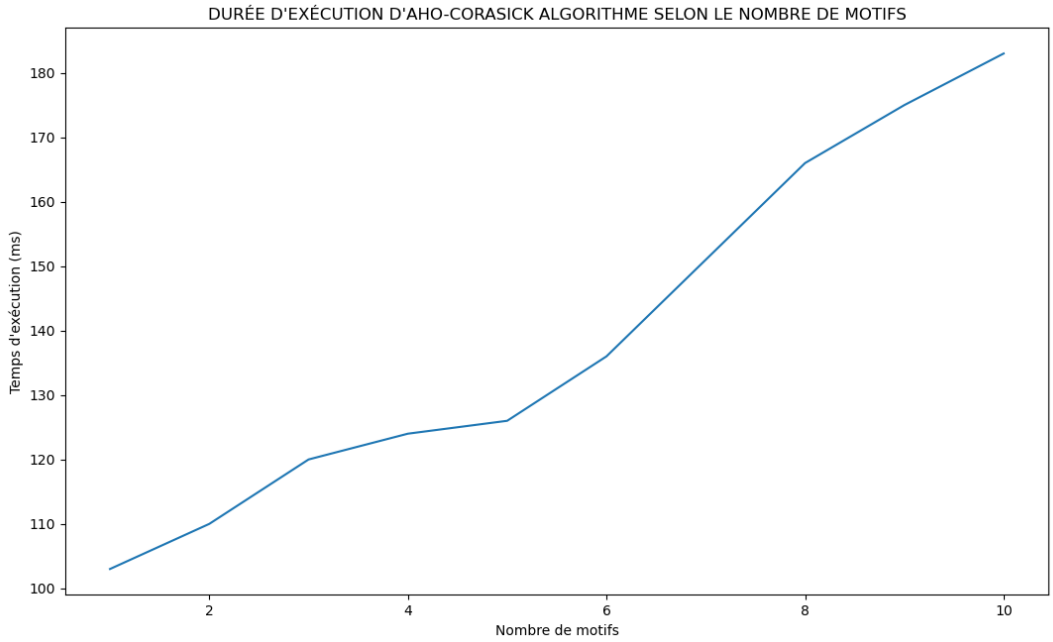


FIGURE 2 : DURÉE D'EXÉCUTION D'AHO-CORASICK ALGORITHME SELON LE NOMBRE DE MOTIFS

**B.** Dans cette étape on fait le contraire on fixe le nombre et on change la taille des motifs :

| *le nombre du motif* | *La taille du motif* | *Temps(ms)* |
|---|---|---|
| 10 | 1 | 90 |
| 10 | 2 | 105 |
| 10 | 3 | 110 |
| 10 | 4 | 124 |
| 10 | 5 | 155 |
| 10 | 6 | 156 |
| 10 | 7 | 160 |
| 10 | 8 | 161 |
| 10 | 9 | 161 |
| 10 | 10 | 167 |

TABLEAU 2 : TEMPS DE FONCTIONNEMENT D'AHOCARASICK SELON L'ALGORITHME SUR LA LONGUEUR DES MOTIFS

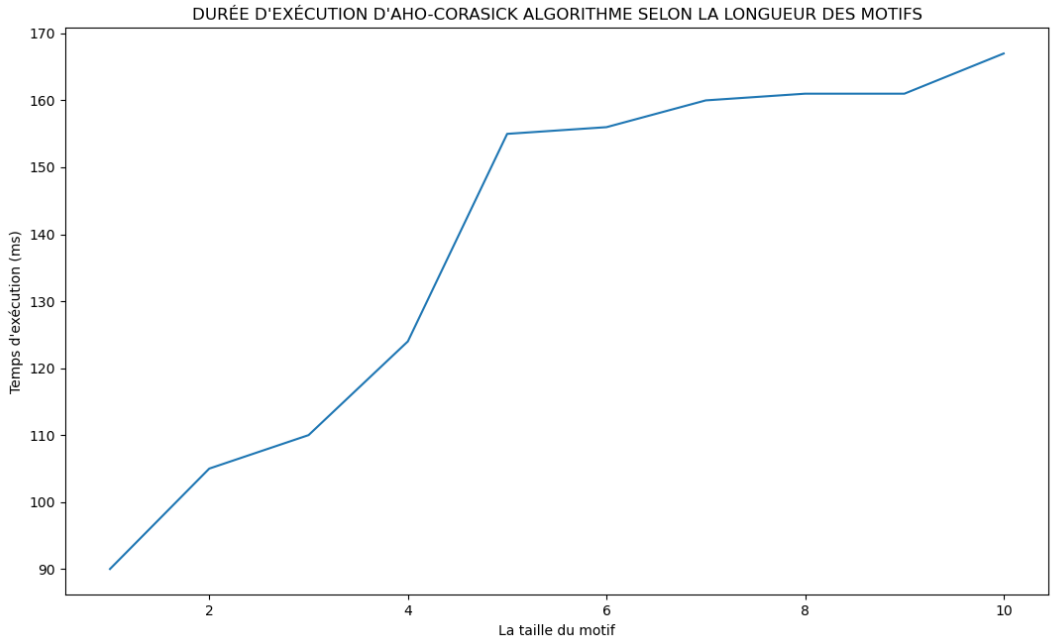


FIGURE 3 : DURÉE D'EXÉCUTION D'AHO-CORASICK ALGORITHME SELON LA LONGUEUR DES MOTIFS

## 2. Rabin-Karp :

**A.** On fixe la taille et on change le nombre de motifs comme suivant :

**Taille du texte : 107,675 lettres (108 KB)** *(le même fichier pour tous les tests)*


| *Le nombre du motif* | *La taille du motif* | *Temps(ms)* |
|---|---|---|
| 1 | 3 | 150 |
| 2 | 3 | 157 |
| 3 | 3 | 162 |
| 4 | 3 | 162 |
| 5 | 3 | 156 |
| 6 | 3 | 166 |
| 7 | 3 | 167 |
| 8 | 3 | 172 |
| 9 | 3 | 182 |
| 10 | 3 | 174 |

TABLEAU 3 : TEMPS DE FONCTIONNEMENT RABIN-KARP ALGORITHME SELON NOMBRE DE MOTIFS

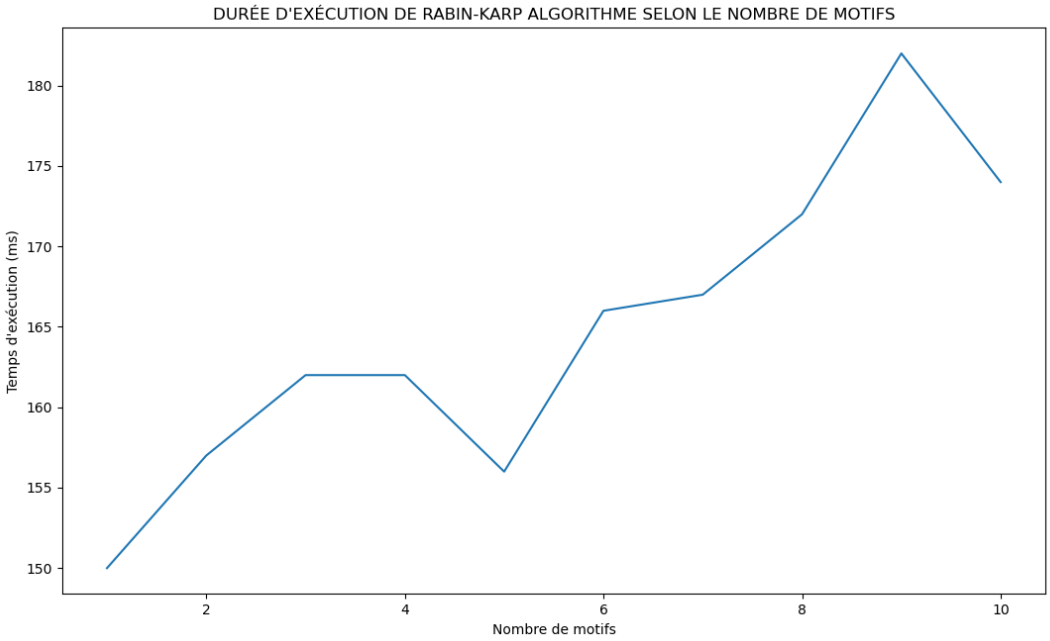


FIGURE 4 : DURÉE D'EXÉCUTION DE RABIN-KARP ALGORITHME SELON LE NOMBRE DE MOTIFS

**B.** Dans cette étape on fait le contraire on fixe le nombre et on change la taille des motifs :

| *Le nombre du motif* | *La taille du motif* | *Temps(ms)* |
|---|---|---|
| 10 | 1 | 191 |
| 10 | 2 | 187 |
| 10 | 3 | 193 |
| 10 | 4 | 197 |
| 10 | 5 | 222 |
| 10 | 6 | 240 |
| 10 | 7 | 246 |
| 10 | 8 | 256 |
| 10 | 9 | 276 |
| 10 | 10 | *290* |

TABLEAU 4 : TEMPS DE FONCTIONNEMENT DE RABIN-KARP SELON L'ALGORITHME SUR LA LONGUEUR DES MOTIFS

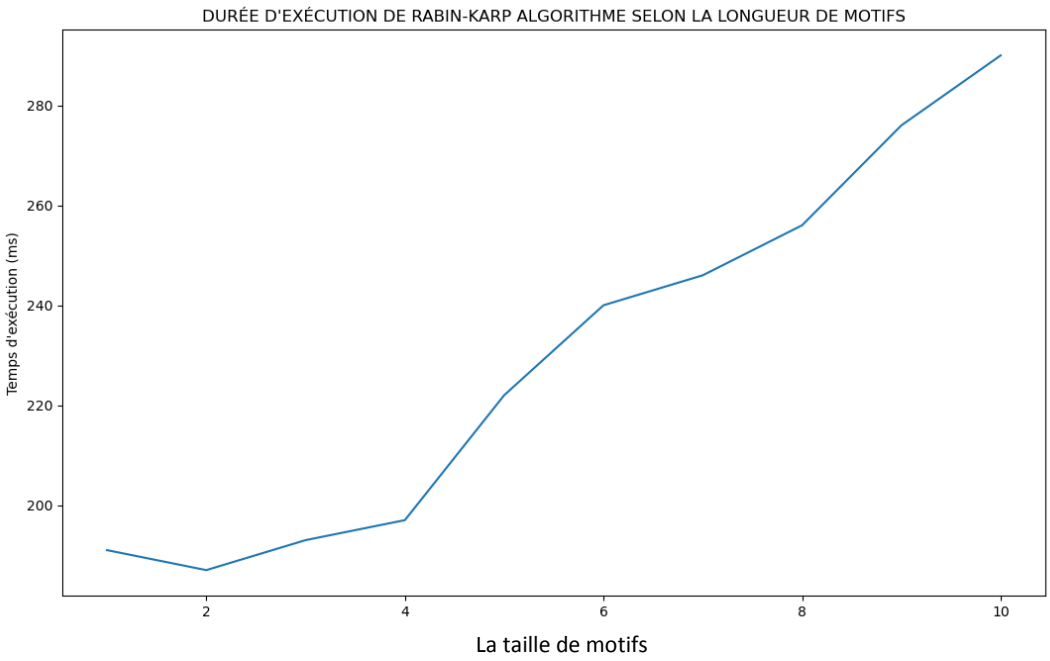


FIGURE 5 : DURÉE D'EXÉCUTION DE RABIN-KARP ALGORITHME SELON LA LONGUEUR DES MOTIFS

## 5. Analyse :

D’après les résultats expérimentaux précédents, nous pouvons clairement voir l'avantage de l'algorithme Aho-Corasick sur l'algorithme Rabin-Karp, il est presque 3 fois plus rapide sans mentionner qu'il est plus stable.

Nous pouvons clarifier cela en disant que l'algorithme Rabin-Karp utilise une méthode de hachage pour rechercher un motif dans un texte. Il calcule une valeur de hachage pour le motif et pour chaque sous-chaîne du texte, puis compare les valeurs de hachage pour trouver des correspondances potentielles, c’est pour ça la complexité temporelle de l'algorithme Rabin-Karp dépend de la longueur du texte et de la longueur du motif.

D’autre part, l'algorithme Aho-Corasick est utilisé pour rechercher un ensemble de motifs (ou sous-chaînes) dans un texte donné. Il précalcule une structure de données appelée fonction de défaillance pour stocker les transitions entre les nœuds dans un trie, qui peut être utilisée pour accélérer la recherche, donc, la complexité temporelle de l'algorithme Aho-Corasick est proportionnelle à la longueur du texte plus la somme des longueurs de tous les motifs.

En général, l'algorithme Aho-Corasick est plus efficace que l'algorithme Rabin-Karp pour rechercher plusieurs motifs dans un texte, car il peut trouver toutes les correspondances en une seule passe. Cependant, l'algorithme Rabin-Karp est plus adapté pour la recherche d'un seul motif dans un texte, car il peut être plus rapide pour des motifs de longueur relativement courte.

## Remarque :

Il est important de remarquer que le choix entre l'algorithme Aho-Corasick et l'algorithme Rabin-Karp dépend des besoins spécifiques de l'application. Si la recherche de plusieurs motifs est nécessaire, l'algorithme Aho-Corasick est généralement le choix le plus efficace. Si la recherche d'un seul motif est nécessaire, l'algorithme Rabin-Karp peut être plus approprié.

# Partie3 :

# 1. Commentz-Walter [CW]

## 1.1. Définition :

L'algorithme de Commentz-Walter est un algorithme de recherche multiple qui permet de rechercher efficacement plusieurs motifs simultanément dans un texte donné. L'algorithme repose sur deux principes clés :

1. Utilisation de l'algorithme de Aho-Corasick : L'algorithme de Aho-Corasick permet de construire un automate finie appelée l'automate AC qui permet de rechercher efficacement un ensemble de motifs dans un texte. Cette automate est construite à partir d'un arbre de trie des motifs et d'un algorithme de calcul de liens de suffixes qui permet de gérer les suffixes communs entre les motifs.

2. Utilisation de l'algorithme de Boyer-Moore : L'algorithme de Boyer-Moore est un algorithme de recherche de motifs dans un texte qui utilise une heuristique basée sur la recherche de caractères mal appariés pour accélérer la recherche. L'idée est de sauter des blocs de caractères lorsque le motif ne peut pas être présent dans cette région du texte.

## 1.2. explication :

L'algorithme de Commentz-Walter combine ces deux principes en utilisant l'automate AC pour filtrer les régions du texte qui ne contiennent aucun des motifs recherchés, puis en utilisant l'algorithme de Boyer-Moore pour effectuer une recherche exacte dans les régions filtrées.

Le principe de l'algorithme est le suivant :

1. Construire l'automate AC à partir de l'ensemble de motifs.

2. Parcourir le texte caractère par caractère, en utilisant l'automate AC pour filtrer les régions qui ne peuvent pas contenir l'un des motifs.

3. Pour chaque région filtrée, utiliser l'algorithme de Boyer-Moore pour rechercher exactement les motifs dans cette région.

4. Répéter les étapes 2 et 3 jusqu'à la fin du texte.

## 1.3. Complexité :

La complexité de l'algorithme de Commentz-Walter dépend de la taille du texte et de l'ensemble de motifs recherchés. Si la taille du texte est n et la taille de l'ensemble de motifs est m avec une longueur maximale de motif de k, alors la complexité de l'algorithme est O(n + mk).

## 1.4. exemple :

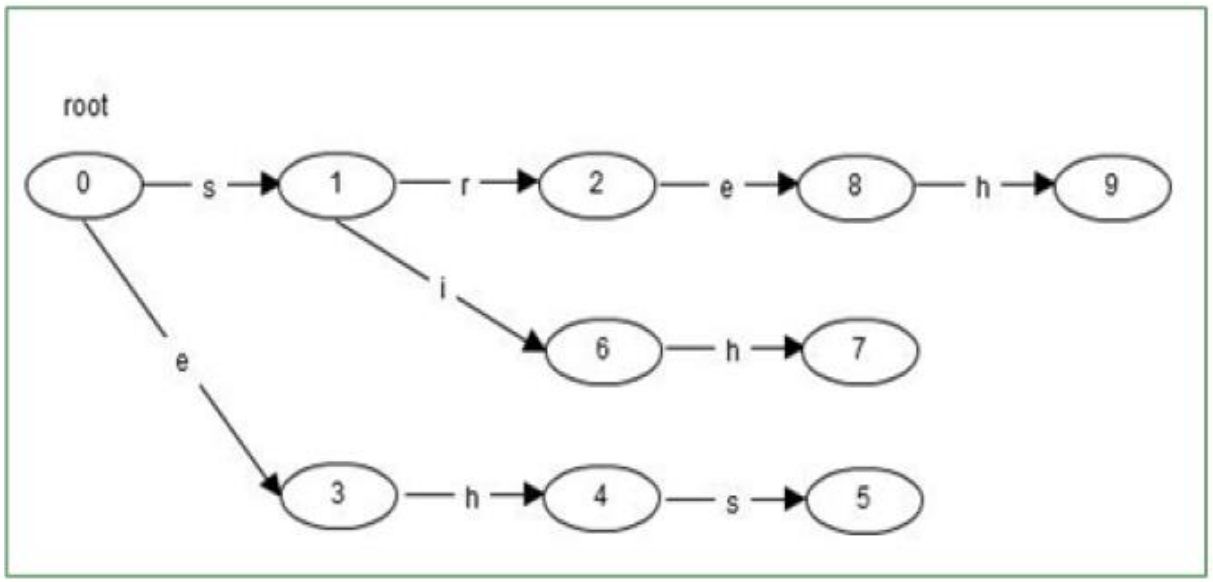


FIGURE 5 : UN EXEMPLE D'ARBRE DE MODÈLE INVERSÉ

## 2. Illustration avec un exemple :

Illustrer son déroulement sur un exemple de recherche de motifs S1, S2, S3 dans un texte T

T= '' ATCGTTACGTGCTATGCCTATGCGCTGATGCCTGCCTGCCTATGGCCT ''

S1=''ATC''

S2=''GCCT''

S3=''ATG''

**Réponse :**

**1.** Construction de l'arbre de recherche :

On construit un arbre de recherche à partir des motifs S1, S2 et S3. Cet arbre est un trie compact représentant tous les suffixes des motifs, ainsi que les correspondances entre ces suffixes. Voici l'arbre de recherche pour les motifs S1, S2 et S3:


2. Les liens de suffixes sont des pointeurs qui relient les nœuds de l'arbre aux nœuds qui représentent le plus long suffixe du nœud actuel qui est également un préfixe d'un motif différent. Voici les liens de suffixes pour chaque nœud de l'arbre Commentz-Walter de l'exemple ci-dessus :

S1 = 'ATC':

{'': 0, 'A': 1, 'AT': 2, 'ATC': 3, 'C': 0, 'TC': 0, 'T': 0}

S2 = 'GCCT':

{'': 0, 'C': 0, 'CC': 0, 'CCT': 1, 'CT': 0, 'G': 0, 'GC': 0, 'GCC': 0, 'GCCT': 4, 'T': 0}

S3 = 'ATG':

{'': 0, 'A': 1, 'AT': 2, 'ATG': 3, 'G': 0, 'T': 0}

Chaque dictionnaire représente les suffixes pour un motif donné, où la clé est le suffixe et la valeur est le nombre d'occurrences de ce suffixe dans le motif. Par exemple, pour S1, le suffixe '' (chaîne vide) apparaît 0 fois, 'A' apparaît une fois, 'AT' apparaît deux fois, 'ATC' apparaît trois fois, 'C', 'TC' et 'T' apparaissent zéro fois.

**3.** Recherche de motifs dans le texte:

On parcourt le texte T en utilisant l'arbre de recherche pour rechercher les motifs S1, S2 et S3. On commence par la racine de l'arbre et on descend dans l'arbre en suivant les liens de transition correspondants aux caractères du texte. Si on atteint une feuille de l'arbre, cela signifie qu'on a trouvé une occurrence d'un des motifs dans le texte. Si on atteint un nœud qui correspond à un suffixe d'un des motifs, on suit le lien de suffixe correspondant pour continuer la recherche à partir de cet endroit. Voici les étapes de la recherche de chaque motif:

- Recherche du motif S1:

On commence la recherche à la racine de l'arbre.

On suit le lien de transition 'A' pour aller au nœud 1.

On suit le lien de transition 'T' pour aller au nœud 2.

On suit le lien de transition 'C' pour aller au nœud 3.

On atteint une feuille de l'arbre, ce qui signifie qu'on a trouvé une occurrence du motif S1 dans le texte.

- Recherche du motif S2:

On commence la recherche à la racine de l'arbre.

On suit le lien de transition 'G' pour aller au nœud 4.

On suit le lien de transition 'C' pour aller au nœud 5.

On suit le lien de transition 'C' pour aller au nœud 6.

On suit le lien de transition 'T' pour aller au nœud 7.

On atteint une feuille de l'arbre, ce qui signifie qu'on a trouvé une occurrence du motif S2 dans le texte.

- Recherche du motif S3:

On commence la recherche à la racine de l'arbre.

On suit le lien de transition 'A' pour aller au nœd correspondant au suffixe 'A', qui correspond à la position 1 du texte.

en suit le lien de transition 'T' pour aller au nœud e correspondant au suffixe 'AT', qui correspond à la position 2 du texte.

en suit le lien de transition 'G' pour aller au nœud f correspondant au suffixe 'ATG', qui correspond à la position 16 du texte.

Le motif S3 est trouvé à la position 16 du texte.

- Ainsi de suite

Le résultat :

Le motif S1 on le trouve une fois à la position 1.

Le motif S2 on le trouve cinq fois à la position 16, 29, 33, 37, 44.

Le motif S3 on le trouve 4 fois dans les positions 14, 20,2 8,4 2.

## 3. Comparaison entre [CW] et [AC] :

Pour le tableau de [AC] on a déjà fait un tableau et son diagramme on fait maintenant pour **COMMENTZ-WALTER :**

**A.** On fixe la taille de motifs (avec différentes tailles) :

**Taille du texte : 107,675 lettres (108 KB)** *(le même fichier pour tous les tests)*

| *le nombre du motif* | *La taille du motif* | *Temps(ms)* |
|---|---|---|
| 1 | 3 | 98 |
| 2 | 3 | 105 |
| 3 | 3 | 124 |
| 4 | 3 | 124 |
| 5 | 3 | 136 |
| 6 | 3 | 145 |
| 7 | 3 | 151 |
| 8 | 3 | 170 |
| 9 | 3 | 175 |
| 10 | 3 | 175 |

TABLEAU 4 : TEMPS DE FONCTIONNEMENT COMMENTZ-WALTER ALGORITHME SELON NOMBRE DE MOTIFS

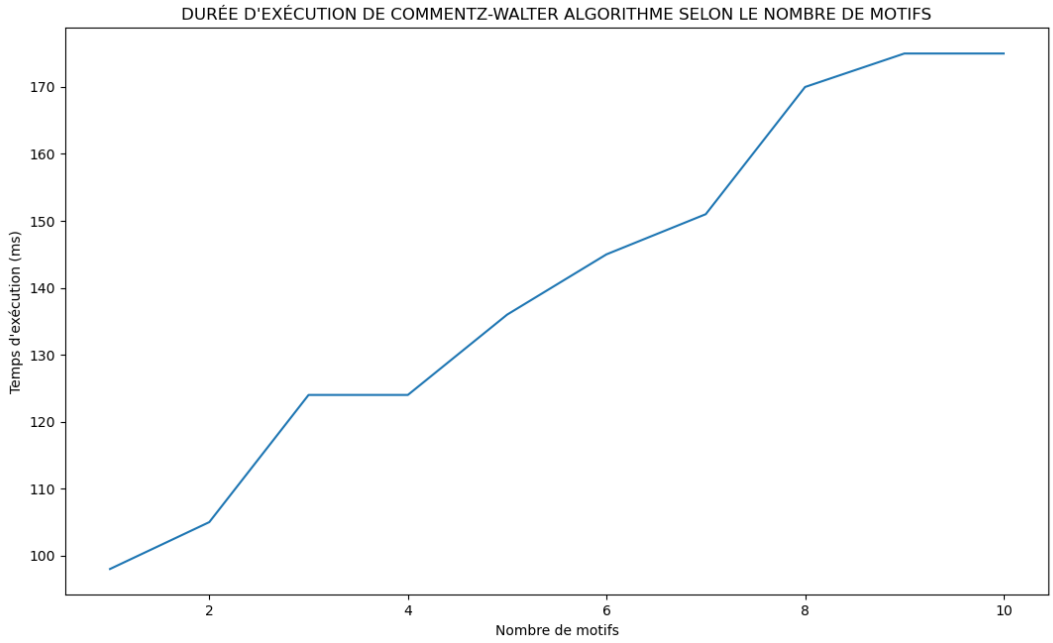


FIGURE 5 : DURÉE D'EXÉCUTION DE COMMENTZ-WALTER ALGORITHME SELON LE NOMBRE DE MOTIFS

**B.** On fixe le nombre de motifs :

| *le nombre du motif* | *La taille du motif* | *Temps(ms)* |
|---|---|---|
| 10 | 1 | 100 |
| 10 | 2 | 111 |
| 10 | 3 | 124 |
| 10 | 4 | 124 |
| 10 | 5 | 124 |
| 10 | 6 | 145 |
| 10 | 7 | 151 |
| 10 | 8 | 173 |
| 10 | 9 | 185 |
| 10 | 10 | 190 |

TABLEAU 4 : TEMPS DE FONCTIONNEMENT DE COMMENTZ-WALTER SELON L'ALGORITHME SUR LA LONGUEUR DES MOTIFS

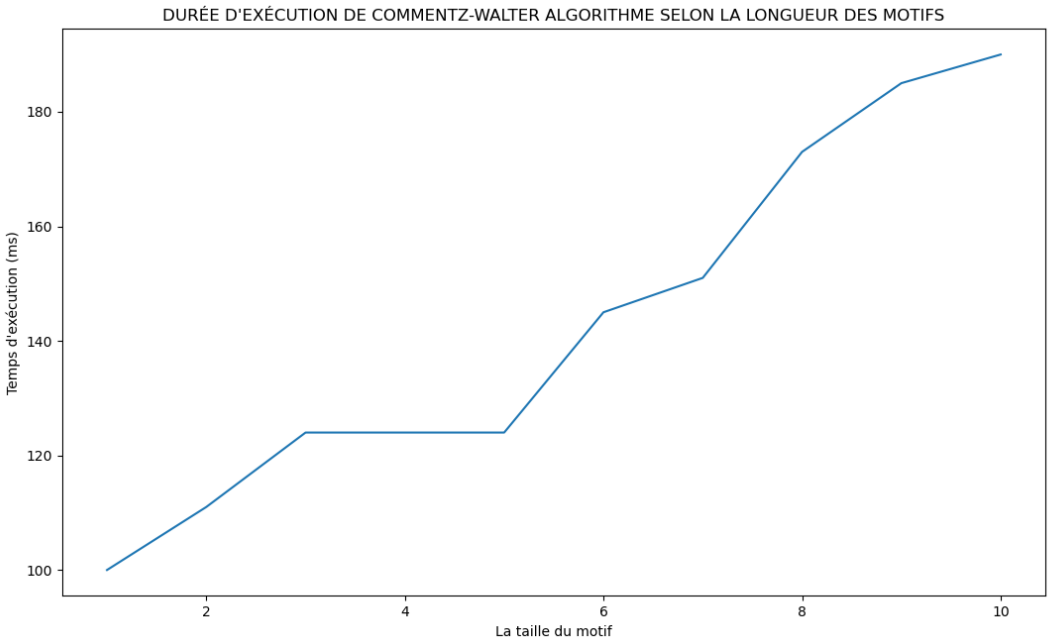


FIGURE 5 : DURÉE D'EXÉCUTION DE COMMENTZ-WALTER ALGORITHME SELON LA LONGUEUR DES MOTIFS

## 4. Analyse :

A partir des résultats expérimentaux précédents des algorithmes AC et CW, nous pouvons voir que les deux sont linéaires mais nous voyons que l'algorithme AC est meilleur lorsque nous avons des motifs de grande longueur parce que le temps d'exécution sera presque stable donc il est plus rapide que l'algorithme Commentz-Walter dans certains cas, car il précalcule la fonction d'échec et la stocke de manière plus gourmande en mémoire que l'algorithme Commentz-Walter. Cela lui permet de gérer des ensembles de motifs plus importants de manière plus efficace, car il peut rapidement passer entre les nœuds dans le trie sans avoir à calculer la fonction d'échec à la volée.

D'autre part, l'algorithme Commentz-Walter est plus efficace en termes d'utilisation de la mémoire que l'algorithme Aho-Corasick, car il utilise une structure de données compacte appelée fonction d'échec pour stocker les transitions entre les nœuds dans le trie. Cela en fait un bon choix pour les applications qui ont des ressources mémoire limitées. Cependant, l'algorithme Commentz-Walter peut être plus lent que l'algorithme Aho-Corasick dans certains cas, notamment lorsque les chaînes de motifs sont longues.

En résumé, le choix entre les algorithmes Commentz-Walter et Aho-Corasick dépend des exigences spécifiques de l'application. Si l'utilisation de la mémoire est une préoccupation et que les chaînes de motifs sont relativement courtes, l'algorithme Commentz-Walter peut être un meilleur choix. Cependant, si la vitesse est une priorité et que l'ensemble de motifs est plus important ou que les motifs sont plus longs, l'algorithme Aho-Corasick peut être une solution plus efficace.

# Conclusion :

Les algorithmes de recherche de chaînes de caractères, tels que Boyer-Moore, Aho-Corasick, Rabin-Karp et Commentz-Walter, ont chacun leurs avantages et leurs limites.

L'algorithme Boyer-Moore est efficace pour les recherches d'un seul motif de longueur moyenne à longue dans les textes de grande taille, mais il peut être moins performant pour les motifs de courte longueur.

L'algorithme Aho-Corasick est particulièrement adapté pour la recherche de plusieurs motifs dans un texte et peut être plus rapide que Commentz-Walter et Rabin-Karp comme on a vu, pour des motifs de courte longueur.

L'algorithme Rabin-Karp est plus utile pour rechercher un seul motif dans un texte, en particulier si le motif est de longueur relativement courte.

L'algorithme Commentz-Walter est également adapté pour la recherche de plusieurs motifs dans un texte, mais peut-être moins efficace que Aho-Corasick pour des motifs de longueur moyenne à longue, mais la différence n'est pas grande et ça peut être mieux qu'Aho Corasick dans certains cas

En conclusion, le choix de l'algorithme de recherche de chaînes de caractères dépend des besoins spécifiques de l'application, tels que la taille du texte et des motifs, le nombre de motifs à rechercher et le temps de réponse requis. Il est important de choisir le bon algorithme pour maximiser l'efficacité de la recherche et minimiser les temps de réponse.

Cette étude couvrait principalement l'analyse et la discussion entre Aho-Corasick, Commentz-Walter, Rabin-Karp et les algorithmes de Boyer-moore pour les problèmes de correspondance de modèles de chaînes multiples. Une solution complète étudiée sur tous les algorithmes existants de problèmes de filtrage de motifs multiples serait très exigeant dans le domaine de recherche des problèmes de correspondance de motifs multiples.