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

Mini-projet de TP

Bio-Algorithmique

Thème

Table des suffixes

Nom et Prénom :

BOUBRIMA Ali

MAMMERI Sara

Chargé de Cours : Mme BOUKHEDOUMA

Chargé de TP : Mr SOUADIA

1. Introduction :

La recherche de motifs et de répétitions dans un texte est une tâche importante en informatique, avec de nombreuses applications dans différents domaines tels que la bioinformatique, la sécurité informatique, la recherche d'informations, etc. Cependant, cette tâche peut être complexe et coûteuse en termes de temps de traitement, surtout pour les textes volumineux. C'est pourquoi l'utilisation d'une structure d'index est recommandée pour accélérer la recherche de motifs.

Dans ce travail, nous avons pour objectif d'implémenter une structure d'index permettant d'accélérer la recherche de motifs, de répétitions, etc., dans un texte. Pour ce faire, nous allons utiliser différentes structures de données telles que la table des suffixes, la table HTR, l'inverse de la table des suffixes, etc. Ces structures nous permettront de trouver efficacement les motifs et les répétitions dans le texte, en réduisant la complexité temporelle des traitements.

Nous allons commencer par construire la table des suffixes du texte et l'afficher. Ensuite, nous utiliserons cette table pour rechercher un motif donné dans le texte. Nous allons également construire la table HTR et l'afficher, ce qui nous permettra de trouver les facteurs répétés dans le texte et de déterminer les répétitions super-maximales. Nous utiliserons également l'inverse de la table des suffixes pour trouver les plus courts facteurs uniques du texte.

Enfin, nous allons utiliser les structures TS et HTR pour retrouver le plus long facteur commun entre deux textes. Pour cela, nous allons comparer les suffixes des deux textes et trouver le préfixe commun le plus long.

Le but ultime de ce travail est de proposer une solution efficace pour la recherche de motifs et de répétitions dans un texte, en utilisant différentes structures de données et en analysant la complexité spatiale et temporelle des traitements. Nous allons mettre en place un menu à choix multiples pour permettre à l'utilisateur de sélectionner les différentes tâches à exécuter et d'obtenir les résultats correspondants.

2. Description des structures de données utilisées et leur complexité spatiale :

1. **Table des suffixes (TS)** : La table des suffixes est une structure de données qui contient tous les suffixes du texte triés par ordre alphabétique. Pour construire cette table, nous commençons par extraire tous les suffixes du texte, ce qui nécessite O(n^2) comparaisons car chaque suffixe a une longueur maximale de n caractères. Ensuite, nous trions les suffixes en utilisant un algorithme de tri efficace tel que QuickSort ou MergeSort, ce qui a une complexité temporelle de O(nlogn). La complexité spatiale de la table des suffixes est O(n) si nous ne stockons que les indices, sinon, O(n^2), car il y a n suffixes et chaque suffixe a une longueur maximale de n caractères.

2. **Table de préfixes communs (HTR) :** La table de préfixes communs est une structure de données qui contient les préfixes communs entre deux suffixes consécutifs dans la table des suffixes. Pour construire cette table, nous parcourons la table des suffixes une fois, en comparant chaque suffixe avec son successeur. Nous stockons la longueur du préfixe commun entre chaque paire de suffixes dans la table HTR. La complexité spatiale de la construction de cette table est de O(n), car il y a n-1 paires de suffixes à comparer. L'espace occupé par la table de préfixes communs est également de O(n), car il y a n-1 entrées dans la table HTR.

3. **Inverse de la table des suffixes (ITS)** : L'inverse de la table des suffixes est une structure de données qui contient la position de chaque suffixe dans le texte. Pour construire cette table, nous parcourons la table des suffixes une fois, en stockant la position de chaque suffixe dans la table ITS. La complexité spatiale de la construction de cette table est de O(n), car il y a n entrées dans la table ITS. L'espace occupé par la table inverse est également de O(n), car il y a une entrée pour chaque suffixe.

4. **Table de longueurs de candidats (LgCandidat)** : La table de longueurs de candidats est une structure de données qui contient les longueurs des candidats à être les plus courts facteurs uniques du texte. Pour construire cette table, nous parcourons la table des suffixes une fois, en stockant les longueurs de chaque suffixe qui ne se répète pas dans le texte. Ensuite, nous parcourons la table LgCandidat une deuxième fois, en vérifiant que chaque longueur de candidat est la longueur maximale de tous les facteurs qui ont cette longueur dans le texte. La complexité spatiale de la construction de cette table est de O(n), car il y a n entrées dans la table LgCandidat. L'espace occupé par la table de longueurs de candidats est également de O(n), car il y a une entrée pour chaque longueur de candidat.

En résumé, la complexité spatiale totale des structures de données utilisées est de O(n^2) pour la table des suffixes et de O(n) pour les autres structures. Ces structures sont efficaces pour la recherche de motifs et de répétitions dans un texte, mais la construction de la table des suffixes peut prendre beaucoup de temps et de mémoire pour des textes très grands. Cependant, la table des suffixes peut être prétraitée et stockée pour une utilisation ultérieure, afin de réduire le temps de construction lors de requêtes ultérieures.

En outre, l'utilisation de ces structures de données pour trouver les facteurs répétés et les répétitions super-maximales peut fournir des informations utiles sur la structure du texte et

aider à identifier les sections importantes. La recherche du plus long facteur commun entre deux textes peut également être utile dans de nombreuses applications telles que la comparaison de textes et la détection de plagiat.

En somme, ce travail vise à mettre en pratique les connaissances théoriques en algorithmique et en structures de données pour développer une solution efficace pour la recherche de motifs et de répétitions dans un texte. La compréhension de la complexité spatiale et temporelle de chaque structure utilisée est essentielle pour optimiser l'algorithme et assurer une performance adéquate pour des textes de grande taille.

3. Pseudo-algorithmes

Tous les algorithmes et le codage en python ont été implémentés à partir du cours « ch4 - les structures d'Index »

1. Construire sa table des suffixes TS et l'afficher :

```
fonction table_suffix(texte):
    suffix_table = []  # Créer une liste vide pour stocker la table des suffixes

    pour i allant de 0 à longueur(texte) exclus:
        suffixe = (texte[i:], i)  # Créer un tuple avec le suffixe et son indice
        suffix_table.append(suffixe)  # Ajouter le tuple à la liste

    suffix_table.sort()  # Trier la liste des suffixes

    retourner suffix_table  # Retourner la table des suffixes
```

2. Rechercher un motif M dans le texte à l'aide de la table des suffixes TS :

```
fonction recherche_occurrences(T, M):
    exist = False  # Variable pour indiquer si le motif existe
    n = longueur(T)  # Longueur du texte
    TS = table_suffix(T)  # Table des suffixes triée
    d, f = 0, n - 1  # Indices de début et de fin de recherche

    # Recherche binaire pour trouver le début de l'occurrence du motif
    tant que d < f:
        milieu = (d + f) // 2
        si M <= T[TS[milieu][1]:]:
            f = milieu
        sinon:
            d = milieu + 1
    deb = d  # Indice de début de l'occurrence

    f = n - 1  # Réinitialiser l'indice de fin
    lg = longueur(M)  # Longueur du motif
    # Recherche binaire pour trouver la fin de l'occurrence du motif
    tant que d < f:
        milieu = (d + f) // 2
        si M == T[TS[milieu][1]:TS[milieu][1] + lg]:
            exist = True
            d = milieu + 1
        sinon:
            f = milieu - 1
    fin = f  # Indice de fin de l'occurrence

    retourner TS[deb:fin+1], exist  # Retourner les occurrences trouvées et si le motif existe
```

3. Construire la table HTR de T et l'afficher en montrant à chaque fois le préfixe commun entre deux suffixes consécutifs :

```
fonction build_HTR(TS):
    n = longueur(TS)  # Taille de la table des suffixes
    HTR = []  # Table HTR

    # Ajouter le premier suffixe à la table HTR
    HTR.append((TS[0][1], TS[0][0], '', 0))

    # Parcourir les suffixes restants dans TS
    pour i allant de 1 à n-1:
        lcp = ""  # Plus long préfixe commun
        htr = 0  # Compteur pour le préfixe commun
        suffix1 = TS[i][0]  # Suffixe courant
        suffix2 = TS[i - 1][0]  # Suffixe précédent

        # Trouver le préfixe commun entre suffix1 et suffix2
        tant que htr < longueur(suffix1) et htr < longueur(suffix2) et suffix1[htr] == suffix2[htr]:
            lcp += suffix1[htr]
            htr += 1

        si lcp == "":
            lcp = "\u03B5"  # Symbole epsilon pour représenter l'absence de préfixe commun

        # Ajouter le suffixe courant et les informations à la table HTR
        HTR.append((TS[i][1], TS[i][0], lcp, htr))

    retourner HTR  # Retourner la table HTR construite
```

4. A l'aide des tables TS et HTR, trouver et afficher
   - le(s) plus long(s) facteur(s) répété(s) dans le texte
   - les facteurs qui se répètent au moins 3 fois.

```
# Trouver le(s) plus long(s) facteur(s) répété(s) dans le texte
le_max = maximum des valeurs i[3] pour i dans HTR
afficher("Le(s) plus long(s) facteur(s) répété(s) dans le texte : ", end="")
pour i dans HTR:
    si i[3] == le_max:
        afficher(i[2], end=" ")

# Trouver les facteurs qui se répètent au moins 3 fois
arr = []
pour i dans la plage de 0 à longueur(HTR) - 2:
    pour j dans la plage de 0 à longueur(HTR[i+1][2]):
        si HTR[i+1][2][0:j+1] == HTR[i+2][2][0:j+1] et HTR[i+1][2][0:j+1] n'est pas dans arr:
            arr.append(HTR[i+1][2][0:j+1])

afficher("\nLes facteurs qui se répètent au moins 3 fois : ", arr)
```

5. Construire l'inverse ITS de la table des suffixes de T et l'afficher :

```
fonction inverse_table_suffix(ts):
    its = liste de zéros de taille len(ts)
    pour i allant de 0 à len(ts) - 1:
        its[ts[i][1]] = i
    retourner its
```

6. Déterminer les plus courts facteurs uniques du texte en construisant la table LgCandidat :

```
fonction lgCandidat(htr, its):
    lgC = liste de zéros de taille len(htr)
    pour i allant de 0 à len(htr) - 1:
        si its[i] < len(htr) - 1:
            lgC[i] = 1 + max(htr[its[i]][3], htr[its[i]+1][3])
        sinon:
            lgC[i] = 1 + htr[its[i]][3]
    retourner lgC

lgC = lgCandidat(htr, its)

htr = [(t[0], t[1], t[2], t[3], its[i], lgC[i]) pour i, t dans énumérer(htr)]

fonction lgC_facts(text, htr):
    pcf = liste vide
    pour i allant de 0 à len(htr) - 1:
        si i + htr[i][5] <= len(htr):
            pcf.append(text[i:i+htr[i][5]])
        sinon:
            pcf.append("-")
    retourner pcf

facts = lgC_facts(text, htr)

afficher "Tous les facteurs :", facts
afficher "Les plus courts facteurs uniques du texte :"
pour i allant de 0 à len(facts) - 2:
    si len(facts[i]) <= len(facts[i+1]) ou i == len(facts) - 2:
        afficher facts[i]
```

7. Déterminer les répétitions super-maximales du texte.

```
fonction rep_super_maximale(htr):
    reps = liste vide
    pour chaque tuple i dans htr:
        afficher i[2]
    pour i allant de 1 à len(htr) - 1:
        super = True
        pour j allant de 1 à len(htr) - 1:
            si htr[i][2] != htr[j][2]:
                si htr[j][2] commence par htr[i][2] ou se termine par htr[i][2]:
                    super = False
        si super:
            reps.append(htr[i][2])
    reps = list(set(reps))
    retourner reps

afficher rep_super_maximale(htr)
```

8. A l'aide des structures TS et HTR, retrouver le plus long facteur commun entre deux textes T1 et T2.

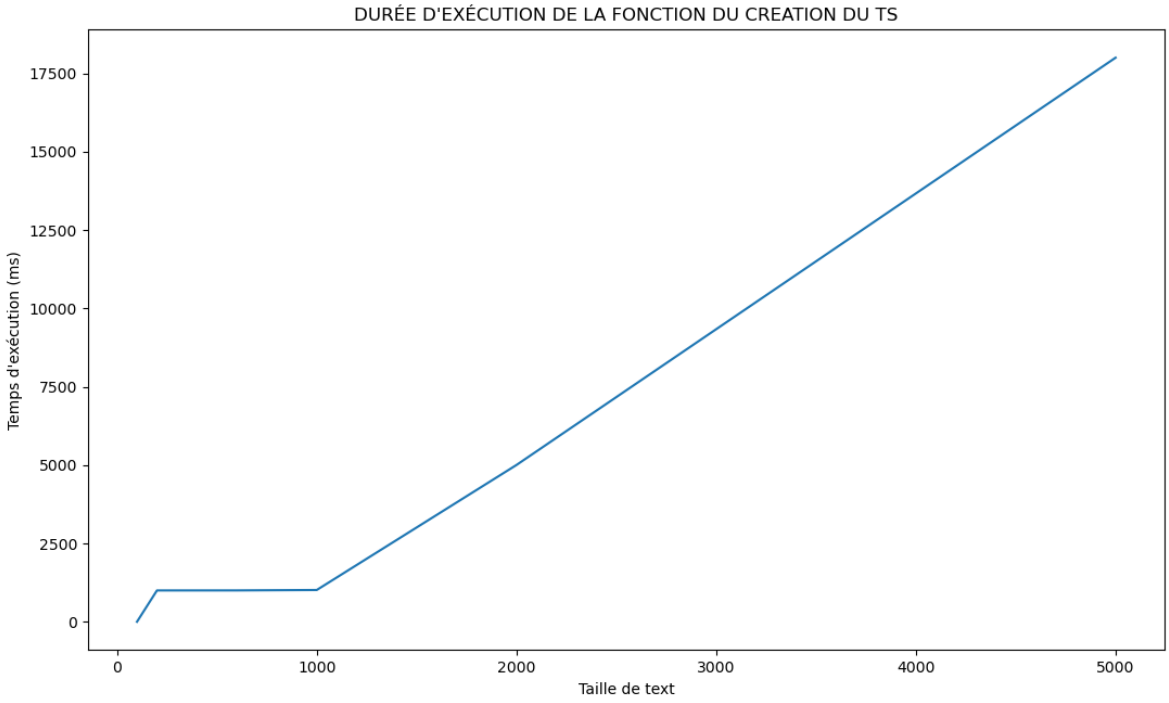
```
fonction plus_long(htr):
    paires_consecutives = liste vide
    pour i allant de 0 à len(htr) - 2:
        si htr[i][2] != htr[i+1][2]:
            paires_consecutives.append((htr[i+1][3], i+1))

    paires_consecutives = trier(paires_consecutives, reverse=True)
    plus_long_fact = [paires_consecutives[0][1]]
    pour i allant de 0 à len(paires_consecutives) - 2:
        si paires_consecutives[i+1][0] == paires_consecutives[i][0]:
            plus_long_fact.append(paires_consecutives[i+1][1])
        sinon:
            break

    facts = liste vide
    pour i dans plus_long_fact:
        facts.append((i, htr[i][0][0:htr[i][3]]))

    retourner trier(facts)

texte1 = "bcabbcab"
texte2 = "caabba"

ts1 = table_suffix(texte1)
ts2 = table_suffix(texte2)

htr2 = build_HTR_multiple(ts1, ts2)

afficher "Les plus longs facteurs communs entre '", texte1, "' et '", texte2, "' sont ", plus_long(htr2)
```

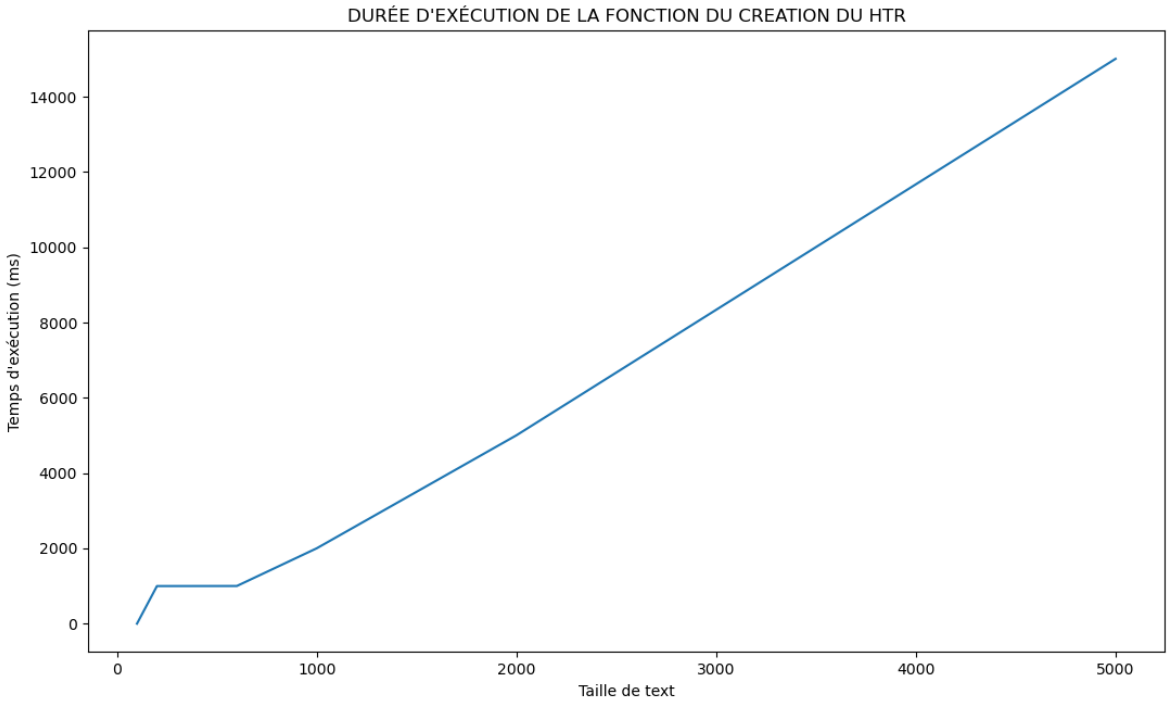
4. Les tableaux des tests

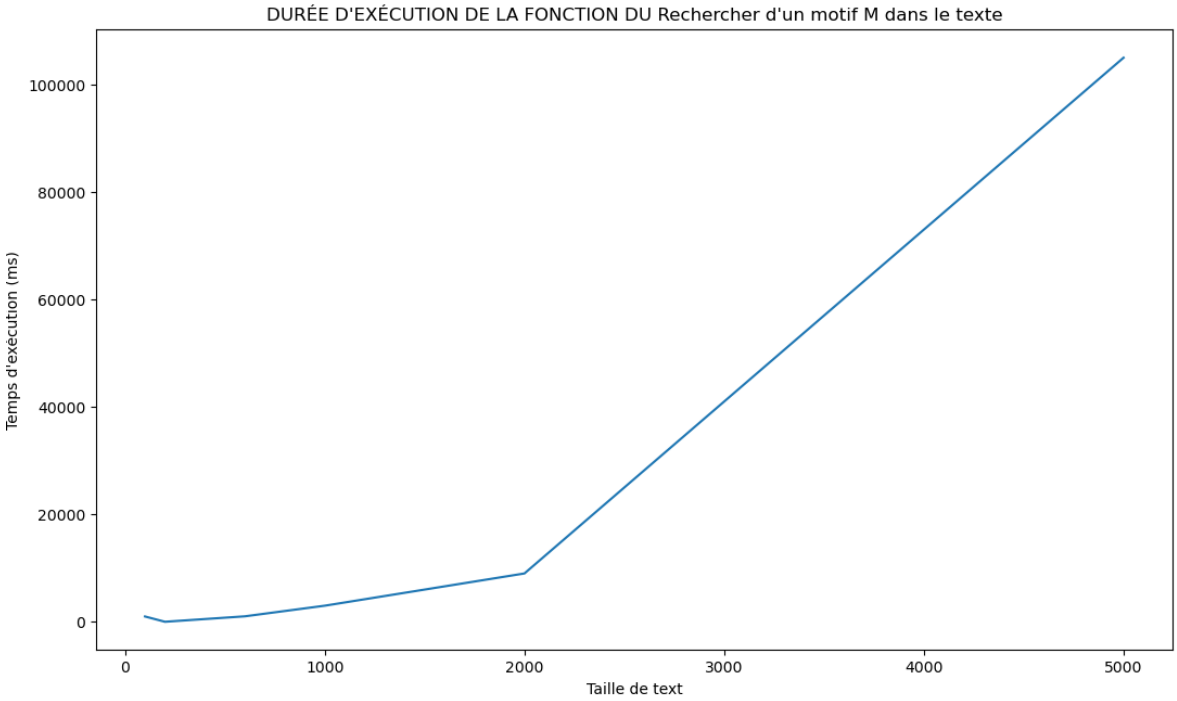
[Temps en micro-sec]

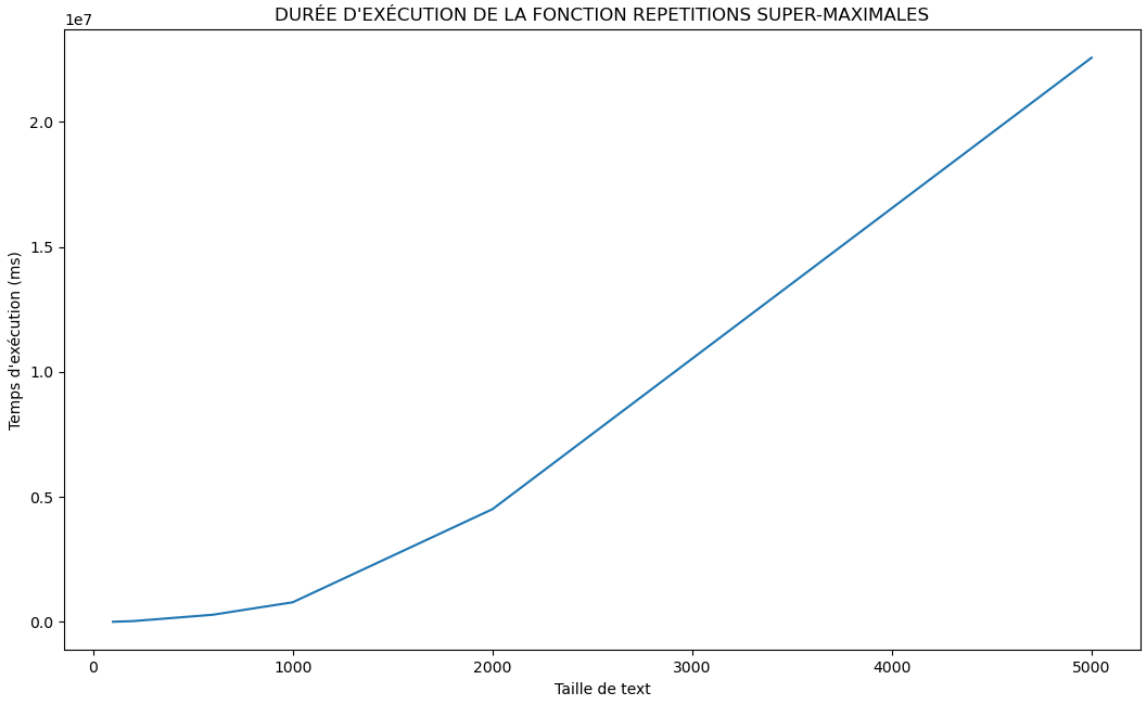
	TS	HTR	Recherche	ITS	Répétions super-maximales
100	0	0	986	1000	6005
200	1000	999	1001	1000	33046
600	1001	1001	1015	1000	287228
1000	1014	2003	2995	1000	785707
2000	5004	5006	9002	1000	4.5 sec
5000	18005	15011	105093	2002	22.5 sec

5. Les courbes (ou diagrammes) de variation temporelle






DURÉE D'EXÉCUTION DE LA FONCTION DU Rechercher d'un motif M dans le texte
Temps d'exécution (ms)
Taille de text
0
20000
40000
60000
80000
100000
0
1000
2000
3000
4000
5000


DURÉE D'EXÉCUTION DE LA FONCTION REPETITIONS SUPER-MAXIMALES
1e7
Temps d'exécution (ms)
Taille de text
0.0
0.5
1.0
1.5
2.0
0
1000
2000
3000
4000
5000

6. Analyse des résultats et conclusions

D'après les résultats obtenus, il semble que cette méthode de recherche (Tables TS et HTR) présente une complexité linéaire ou quasi-linéaire par rapport à la taille du motif. Lorsque la taille du motif augmente, on observe une augmentation proportionnelle du temps de recherche.

La construction de la table des suffixes (TS) et de la table HTR semblent être relativement efficaces, avec des temps d'exécution qui restent faibles par rapport aux autres opérations.

Cependant, la recherche d'un mot semble avoir une complexité plus élevée. Et la recherche de répétitions super-maximales prend énormément plus de temps à mesure que la taille du texte augmente.

En conclusion, cette méthode de recherche semble bien adaptée pour des motifs de taille relativement petite. Cependant, elle peut rencontrer des difficultés lors de la recherche de répétitions super-maximales sur des textes de grande taille, mais d'un point de vue général c'est tellement efficace

7. Anexe

Tous ces exemples sont les mêmes que les exemples du cours, et ils fournissent les mêmes résultats

1. Construire sa table des suffixes TS et l'afficher

```
text = "abracadabra"
```

```
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
1
   texte[TS[i]:]  TS[i]
0              a    10
1           abra     7
2    abracadabra     0
3       acadabra     3
4         adabra     5
5            bra     8
6     bracadabra     1
7        cadabra     4
8          dabra     6
9             ra     9
10     racadabra     2
```

2. Rechercher un motif M dans le texte à l'aide de la table des suffixes TS

```
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
2
Le mot 'abra' a été trouvé à l'indice [7, 0]
```

3. Construire la table HTR de T et l'afficher en montrant à chaque fois le préfixe commun entre deux suffixes consécutifs

```
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
3
    TS[i] texte[TS[i]:]   lcp  HTR
0      10            a         0
1       7         abra     a    1
2       0   abracadabra  abra    4
3       3      acadabra     a    1
4       5       adabra     a    1
5       8          bra     ε    0
6       1    bracadabra   bra    3
7       4       cadabra     ε    0
8       6        dabra     ε    0
9       9           ra     ε    0
10      2     racadabra    ra    2
```

4. A l'aide des tables TS et HTR, trouver et afficher

- le(s) plus long(s) facteur(s) répété(s) dans le texte
- les facteurs qui se répètent au moins 3 fois.

```
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
4
le(s) plus long(s) facteur(s) répété(s) dans le texte :  abra
les facteurs qui se répètent au moins 3 fois :  ['a', 'ε']
```

5. Construire l'inverse ITS de la table des suffixes de T et l'afficher

```
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
5
   ITS[i]
0       5
1       1
2       8
3       6
4       2
5       9
6       7
7       4
8       0
9       3
```

6. Déterminer les plus courts facteurs uniques du texte en construisant la table LgCandidat.

```
text = "GATGATTGAG"
```

```
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
6
   TS[i] texte[TS[i]:]  lcp  HTR  ITS  lgC
0      8           AG         0    5    4
1      1     ATGATTGAG    A    1    1    3
2      4        ATTGAG   AT    2    8    4
3      9             G    ε    0    6    4
4      7           GAG    G    1    2    3
5      0    GATGATTGAG   GA    2    9    2
6      3       GATTGAG  GAT    3    7    4
7      6          TGAG    ε    0    4    3
8      2      TGATTGAG  TGA    3    0    2
9      5         TTGAG    T    1    3    2
Tous les facteurs ['GATG', 'ATG', 'TGAT', 'GATT', 'ATT', 'TT', 'TGAG', 'GAG', 'AG', '-']
Les plus courts facteurs uniques du texte
ATG
TGAT
TT
AG
```

7. Déterminer les répétitions super-maximales du texte.

```
text = "GATAAGATTGATG"
```

```
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
7

A
A
AT
AT
ε
G
GAT
GAT
ε
T
TG
T
['GAT', 'ε', 'TG']
```

8. A l'aide des structures TS et HTR, retrouver le plus long facteur commun entre deux textes T1 et T2.

```
les plus long facteurs communs entre 'bcabbcab' et 'caabba' sont [(4, 'abb')]
les plus long facteurs communs entre 'a-b-cd-ef-gh' et 'a.b.cd.ef.gh' sont [(13, 'cd'), (17, 'ef'), (21, 'gh')]
Choisir entre 1 et 8 pour repondre au question, sinon 'q' pour quitter
q
Quitter le programme
```