

# La recherche du motif dans un texte.

Younès BENSAFIA, Oussama MAAMERI, Abderrahmane MOUSTARI, M1-MIV

---

## 1. Introduction

Ce compte rendu présente les détails de la réalisation du travail pratique portant sur la recherche de motifs dans un texte. L'objectif principal était d'explorer, implémenter et comparer plusieurs approches algorithmiques pour résoudre ce problème, tout en étudiant leurs performances empiriques en termes de temps et d'espace.

## 2. Objectifs du TP

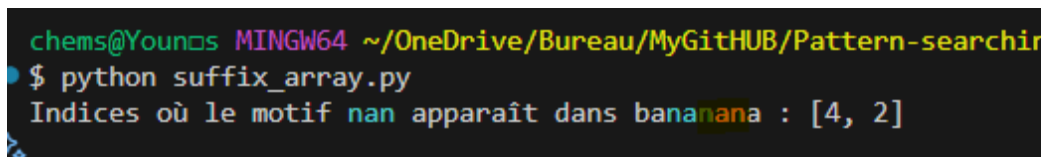
- a. Développer des solutions basées sur :
  - Le tableau de suffixes.
  - L'arbre de suffixes avec l'algorithme d'Ukkonen.
  - Le FM-index.
- b. Analyser empiriquement les complexités temporelles et spatiales de chaque méthode.
- c. Tester les algorithmes sur des données synthétiques et réelles, et identifier les facteurs influençant leurs performances.

## 3. Méthodologie

### a. Implémentation des structures et algorithmes:

#### i. Tableau de suffixes :

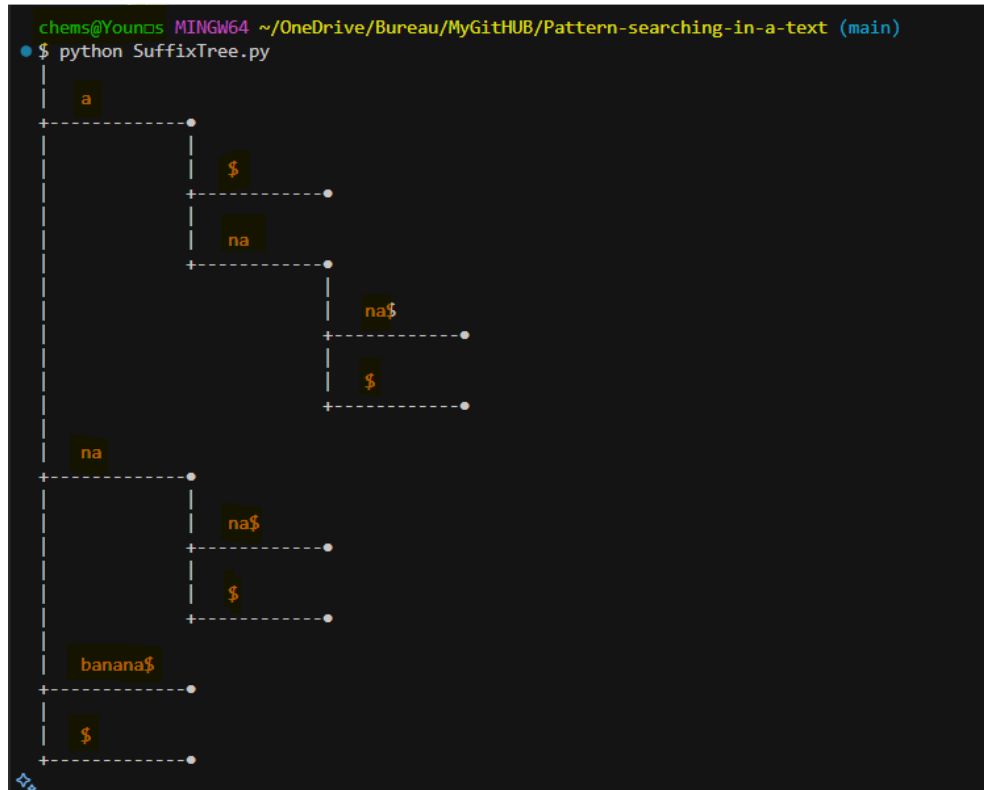
- Construction du tableau à l'aide d'un algorithme basé sur le tri.
- Recherche dans le tableau pour localiser le motif.



```
chems@Younès MINGW64 ~/OneDrive/Bureau/MyGitHub/Pattern-searchin
$ python suffix_array.py
Indices où le motif nan apparaît dans banana : [4, 2]
```

ii. **Arbre de suffixes (algorithme d'Ukkonen) :**

- Construction de l'arbre
- Recherche optimisée en parcourant l'arbre.



iii. **FM-index :**

- Prétraitement à l'aide de la transformation de Burrows-Wheeler.
- Utilisation de structures auxiliaires comme les tableaux de rangs et de fréquence.

```

chems@Younos MINGW64 ~/OneDrive/Bureau/MyGitHub/Pattern-searching-in-a-text/done (main)
$ python FmIndex.py
Le motif 'you' apparaît 2 fois dans le younessenyou.
Positions : [0, 9]

```

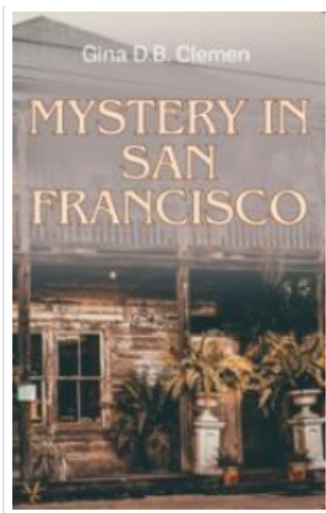
## b. Analyse empirique

### i. Cas de test :

- Chaînes de tailles variables générées aléatoirement.

```
ALHNGZDVHRZQSQRETQGFKAQJXGWRREDCYLWZYLEOTQXBMZVEFGICML,XYZ
VQKQOTVMEVUJVBVRL EHXQUMEJJCZCXKQZVQNDTSOZKCBQECDEYQIMGYAQJUFINELVWYEBQKCOLTMZF,XYZ
XECNIRETELOOPSJSTWNSVNSZLCKQVGTGKQVELEQUSROPXGZKZTHUBXZYXHAVUGTFXDATINODEMBYTIYWB,XYZ
HQSMNFSVSFZBTBNJMRXMDJGIAFTZTHNUEJLTDCLXBDCSGNIZKPHCYZHQSAVBGJOBUILLWCJGOHUUUBV,XYZ
PZCVOSDCZACBGAYKVSITEMOXVJLWZSUTPQBNIPYVZAQOGBDCSNGMTUAMDGRCEGYTHMOKCUYCRITYEIXZVZOCDFVNSWDMUJAMUREGMI
PZCQNSFTOPOLATBWMIRBNFOMHXBABRHEVZCOXEYDATMPKOJWZKLOXZJKTISBFZRMVVGAGLXGDBYBGTJFYKKTRFLYCFALGXKYAKTRIYDBKBT
BCQOYNCAHKPDMMIRJFXKJLVAMITAXYLEFERVXKTFACMPMLYRDOTPONHRXQKAPQJHYXEGVFSQZCXPNRDEPOACABBBMLFEFEQAVNITWB
GZAYUICAHMUTHERXZQVSKJDNILRKQSORPDPDRPHITJEQLVXCQYXCGLOHMTSTAMLXYZPGLZMHMBMPKMTURTPTBTHNQITUYOKGVARIT
GKZSYGKKGAZAFEDDBZKQKQKMLACHVCRUFLZLGTNETKCLCOLSCKPMNEYMTJEMIDAASKMEBCFKTFZSHMPRIJIPQBFQMSIDTQIMDAAMZKO
TFKBLLSVDEJFKOVVWKKQXVJUZTFUEJXDYJUEABDZGOVTVSRNEJXVJNFBRZHSKJTFJGJZEQCYCJLPYOGGYRJTGJFMHDI VITGOJYCC
YDCXARATQCMNUEQOJMDZHBKROAFPLVCQGHYVILLTQBTSYADHRMAESHGWFADFPGXBCOHIHLNCKQMLHEVWGESMCTKUTKDAFESRA
SQNILLZMMWQSHYEQKAYTSJLQBRRHMHJTHARVMAESGVARBNJYHJHSUXHBEQYHCYDKSZJITQSNQLCYCQFODPZSLBGUDYXZNSRYJETAU
GQRRXCCKBKHYCPRPQKQVAAQJZVXRNDLZPTNKGFRBNBYLJUBNECYCAEXZEBLZLXGLFHZVYNDLUNPVAHBNPTOCBSZUQKFEZGQCL
MGYCKDZMXAYMLWTCVMBRYECJJIYDSKSDVUXKCAXTMNGASCOHBBVRIGBONSVPFBPMZSYNEPJXOTGAWAFZCZPOVONYDLRBFVCYALABEAH
GTMAHLPGHEHQHMLWJICHLFARAVIQUIDKUGSNOVGKQZPQMLNFXKEEYYSMBFKHPYJCLGCTAGVENRRMOSLBTHAPFGYBDHLCJUVU
MJSLSGBLASHALMPTQALAFQJXBMLQTTINGIANKUJRTKEBEYNETTDXWJFMLRUBVAMIDJPHGESOBUTYJWBKAYJPLGONDFFRZGRXFER
ZNGHZAQOQHPJVBATRSKXGVDERCQANDFYDAAQSCRKOTYFBHQZTRIGNRHEORQXDNVITIXSSOKRGLJHGBKXZOMTCKKQKQDLSMFQWHT
YORKBTOIMWKKZVUQHQBQTVDUJGDXJYVKTDCZGUQFRZHVJYDQASOFJTLJLTHLXGHOHYEASPAAGJBKXBNYPQWCKSIRLWHLKLB3F
ZLQBOHIXHTFZSTAJLWKPBEIAFIHMYRGINIMNAYFQBUJVRJDSWBRSSXJACTBSHIDKZRFKFCQEGKHKKDOXTJBBWYHITDPSNPFHOFVA
NENJQQGQAKQSXGRGLSDXWNYTYDFADVORRVPISHURINTNASHRQZFKRDBGXPMQHCPOGYSSAIEENTSLXPVLUFHFNWQXCVHSBZT
ZJTGKLOENQJLHJYTSMDJFQJTMANKYNYETIAHMBZTJJAQTHUBZRAZAUCLYHBAFHXOXTYRNBQAOQJTORWFASAOQTRDKFZNEHNC
ZYNQJQJLCOEHBQWKKYKIDMHMOGECPLTFHMYFLUNDAVUDGZLHHRMBDBB3LRCLLSBTACJRYVSEPPJLUNRWXNTNFUJ3FXJYVQUSGC
FQLLEDTQZATFBKXDRJUSHSJPOJHCAUEHBTJOPPHUTVITQLEAJMSUQOZBFXEMLPEDHASNHNITSDEINVTILTNZAPCSCLPOCDOPKZPZ
YDQMSSECLPLTKDQTHLVWZYPVXKETHQXFXQNBQAUYBEACQVUJZASMBYXQJHIFESGCCOJHNRDBKHKJYAEVZDILHUPHOKGAILNSM
DVMYHJGJPELQJUDYDKAWITABRZJVOVRLPMBXURZNTKLJCCQJSHARQEAYZJODNAQFZTFGELHEHBOQANVCYEHLEFJXJDOXABGMLBT
UMQVHTYPODRFZGXPKYQYTYVIAAGRCOTLSMIFOTHHBTOFLUOQOQJLPKMTGEXTQKQAYDAYGAMICSLBVTYPYGBHRZQMDAYIGZDPQVND
TEWAGSGAIDRZFYQQUODJNZFPHRTBQZJZBFHFRANSDOZDCHTVXGAMPCHTTFXJFLLMITYADAXCQBRIGAXCTAACOMPSPHNTJUYTZRGGFVJ
EUIZQKJXCYUUEJJOVHEHNSPFGMEUHINGEHQMLVQRFYAGHBLSVTNZDOBRRVGASWIDSWBNHICMIBGKLQKVAXPGXSWBMUYLLULBF
ADVLVFLUNXESDXHPSGCMIDDRHMOUETGUPGOQMBKARVHINOQJDTMLRIDMETKTEPEJEDRWZWAENZCURVBLJDLXJCECEEGSGHIKTHCE
EYFFBORKUNREAXECSKFQQLYMHOTYXSEXRBXQAQAEBZIEELGDXFTNPKOGXOEAADYBOYRMDODREZRGJQRHMPCHLRYENMFHXEDOF
BNSNIBFDHPLULDEUEQAEABLGAZHXPHNDJYLTUAGJSJUSTJIOXIMBYCHMMPJLALBOGTLNCKBFCJLXSUDADDQDLGVZWSBDPHVBHGKIC
MOGHKFTKEJUVZHECOBXABMEPKHAEMLPZCILLUESITLXSARVTHLXYZEJVPVENKCBKMKZCHZEKZAOOLAYQCEHLMHJEICOTEDERJSEFWNR
ODHLLMORRPJKEGRJYNAEYHNTSLNRZCSGKYODMLJMKXNYHORVRSIRUVNIOXJGARGJDSBZCJCPWSDTDZVBKMPULFZMMRJRJWMPAOMBACCB
CAVPKEGHBFAEYEGELBFHYZVJDSWYTTANWJCEZSRJWJUNNRNOMMDYMLPBFWODLDXYPFTXYBIBAOOEYAWFANILVIGBLLTHEVOMEXKPZAGU
JYAJAPPTZPHUDNMJLBYOGJLVNHWJ3JFNOJHCXHXOYLQBFXDEVDJUXENTXLTJHCFYSLAHLBYDADYNUFASYNCKATYHGEAUQVGLNDQJ
```

- Données réelles extraites de séquences (une partie d'un livre "MYSTERY IN SAN FRANCISCO et le motif).



```
Length,String,Pattern
473,"Jim Reilly was fourteen years old
562,"One foggy afternoon, Jim saw the
385,"Susan took the Moreno children to
```

```
Flowers dead.",Daily routine and setting description
old and had a strange story associated with it.",Encount
.",Environmental crisis
```

```
nington Square was a historical and lively park s
she was gone. He rushed home, telling his grand
contaminated Environmental crisis with a chemica
```

- Données réelles extraites de la séquence n°2 (données ADN générées précédemment dans le projet 1)



## ii. Environnement d'expérimentation :

- Matériel : (exemple : Rayzen 5, 8 Go RAM).
- Logiciel : (VS Code, Jupyter Book, Python).

## 4. Résultats ou observations

Nous avons enregistré et analysé les temps d'exécution ainsi que l'utilisation de la mémoire pour chaque approche en fonction de la taille des entrées.

### ❖ Temps d'exécution :

➤ La structure de tableau de suffixes:  $O(\text{pow}(n,2) \cdot \log n)$  'sort',  $O(\text{pow}(n,2))$

#### "Construction"

```
chems@Younas MINGW64 ~/OneDrive/Bureau/MyGitHUB/Pattern-searching-in-a-text/done (ma
$ python suffix_array.py
• Indices où le motif nan apparaît dans banananabanananabanananabanananabananan
Suffix array construction time: 0.0000405693
Pattern search time: 0.0001013756
Total time: 0.0001419449
```

- L'arbre de suffixes (Algorithme d'Ukkonen):  $O(n+m\log n)$  ("m" taille du pattern)

```
chems@Younos MINGW64 ~/OneDrive/Bureau/MyGitHub/Pattern-searching-in-a-text/done (main)
$ python AlgoTree.py
Indices où le motif nan apparaît dans banananabanananabanananabanananabanananabananana
6, 68, 74, 76]
Suffix tree construction time: 0.0004854679
Pattern search time: 0.0003193903
Total time: 0.0008048582
```

- FM-index:  $O(n)$

```
chems@Younos MINGW64 ~/OneDrive/Bureau/MyGitHub/Pattern-searching-in-a-text/done (main)
$ python FmIndex.py
Indices where the pattern 'nan' appears in 'banananabanananabanananabanananabanananabananana'
60, 66, 68, 74, 76]
FM-index construction time: 0.0001884174 seconds
Pattern search time: 0.0000000000 seconds
Total time: 0.0001884174 seconds
```

## ❖ Utilisation de la mémoire :

- La structure de tableau de suffixes:

```
chems@Younos MINGW64 ~/OneDrive/Bureau/MyGitHub/Pattern-searching-in-a-text/done (main)
$ python suffix_array.py
Indices où le motif nan apparaît dans banananabanananabanananabanananabanananabananana
26, 18, 10, 2]
Suffix array construction memory usage: 8.4892578125 KB
Pattern search memory usage: 8.4892578125 KB
Total memory usage: 16.9785156250 KB
```

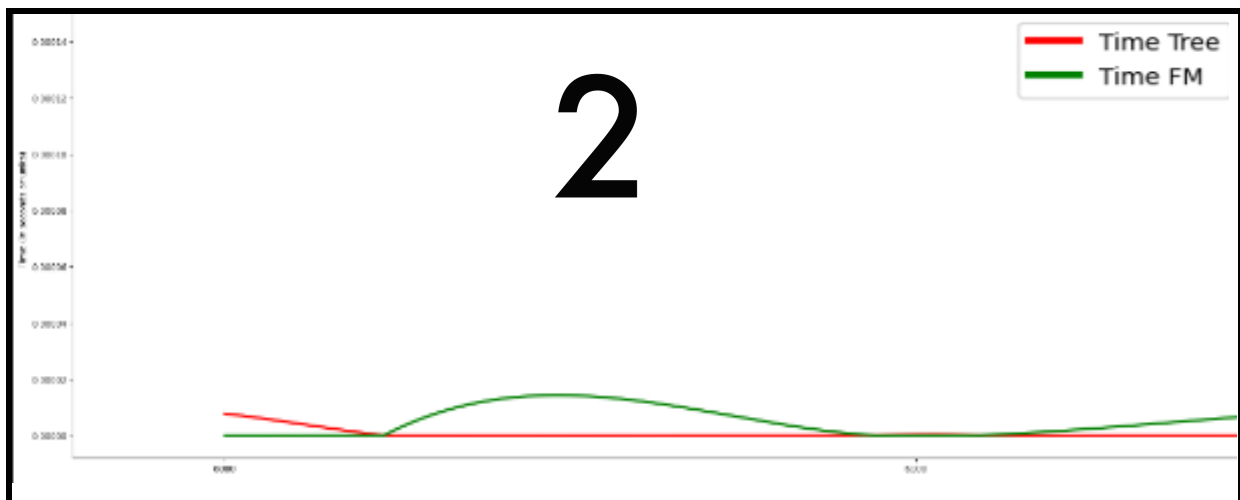
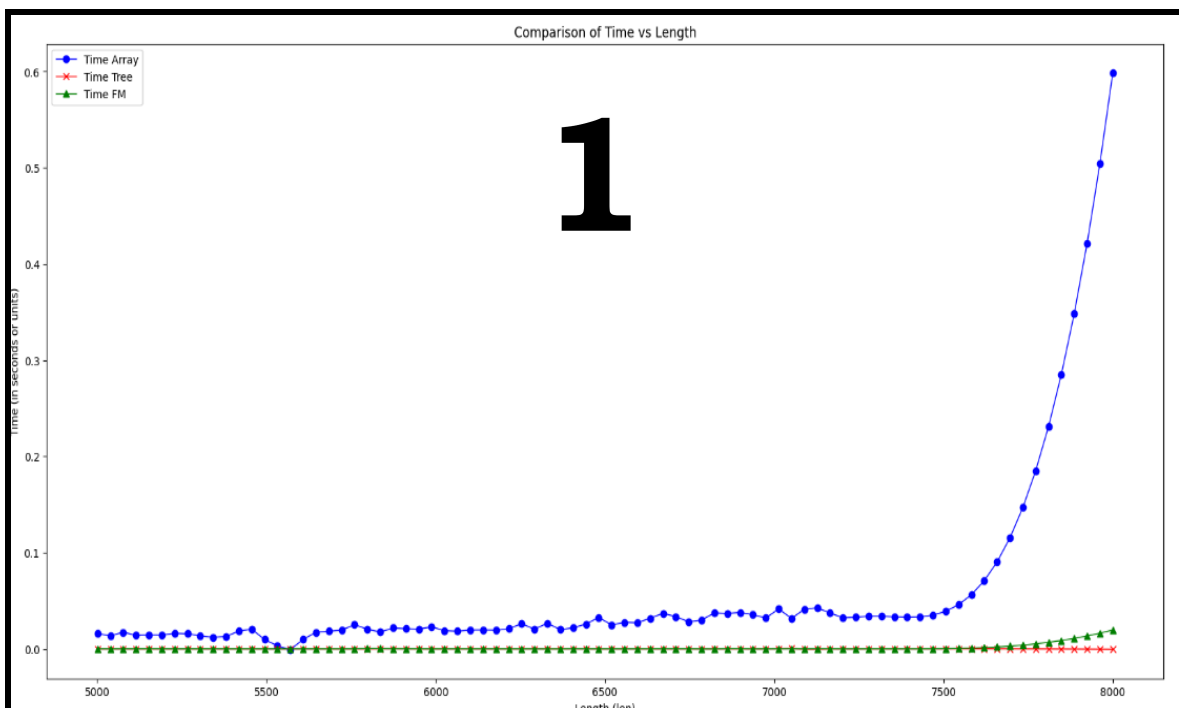
- L'arbre de suffixes (Algorithme d'Ukkonen):

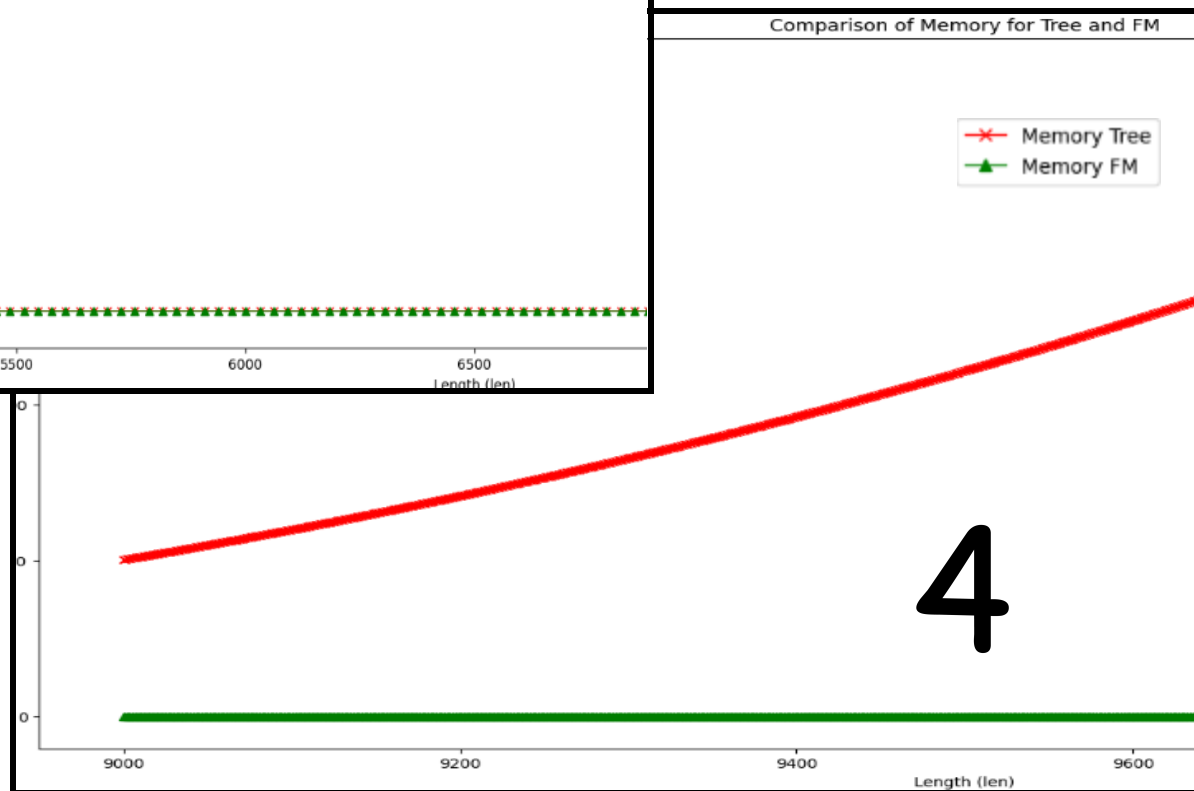
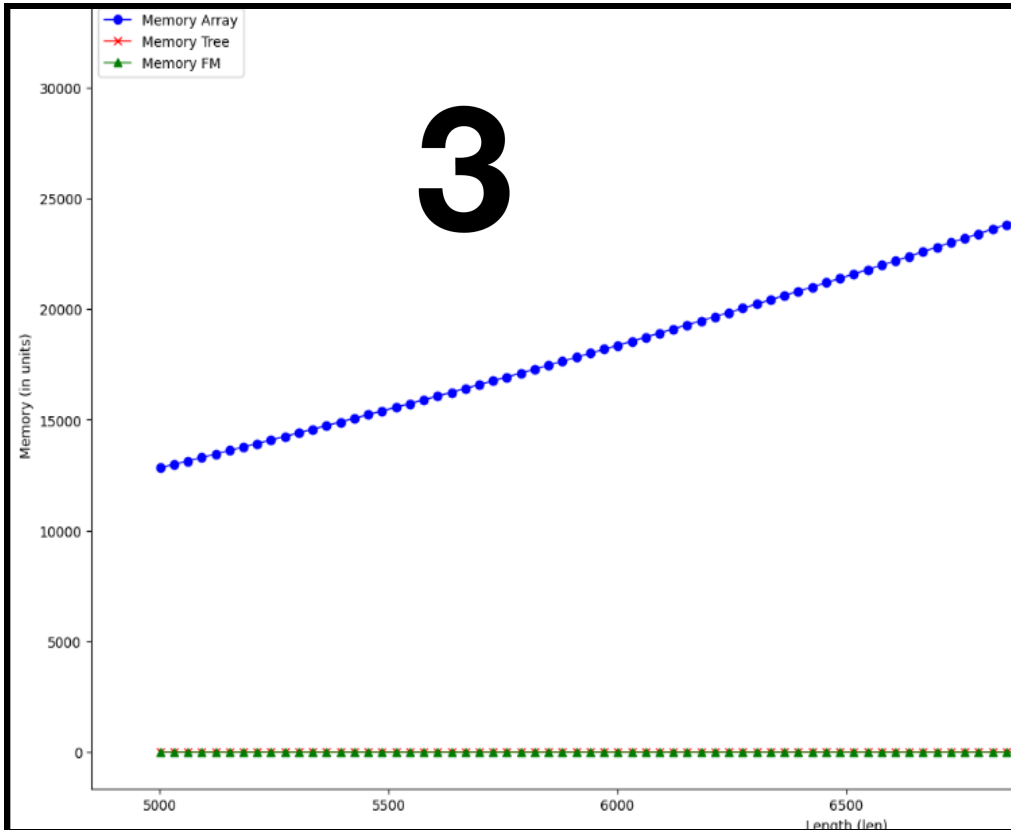
```
chems@Younos MINGW64 ~/OneDrive/Bureau/MyGitHub/Pattern-searching-in-a-text/done (main)
$ python AlgoTree.py
Indices où le motif nan apparaît dans banananabanananabanananabanananabanananabananana
6, 68, 74, 76]
Suffix tree construction memory: 34.30 KB
Pattern search memory: 1.16 KB
Total memory: 35.46 KB
```

➤ FM-index:

```
chems@YOUNES MINGW64 ~/OneDrive/Bureau/MyGITHUB/Pattern-se
$ python FmIndex.py
Indices where the pattern 'nan' appears in 'banananabananan
60, 66, 68, 74, 76]
FM-index construction memory usage: 9.8439062500 KB
Pattern search memory usage: 0.5 KB
Total memory usage: 10.2970312500 KB
```

Nous avons ensuite créé des graphes illustrant la croissance empirique des performances en temps et en espace, montrant de manière claire que





❖ **Interprétation du graphe (1):**

- Suffix\_array est significativement plus lente et moins stable (en termes de temps d'exécution) par rapport aux deux autres (FM\_Index et Suffix\_Tree), en particulier pour les grandes tailles d'entrée.
- Les structures ou algorithmes correspondants aux FM\_Index et Suffix\_Tree sont mieux adaptés à gérer des entrées de grande taille de manière efficace.

❖ **Grappe 2 (Zoom de la courbe rouge et verte):**

Ce second graphe est un **zoom sur les courbes rouge ("Time Tree") et verte ("Time FM")** présentées dans le premier graphe. Cela permet de mieux analyser

leurs détails, qui étaient aplatis dans l'échelle globale en raison de la dominance de la courbe bleue ("Time Array").

### Ce que le zoom révèle :

- **Time Tree :**
    - i. Semble presque constante, mais légèrement au-dessus de la courbe verte. Cela signifie que l'algorithme ou la structure associée à l'arbre est très efficace, mais légèrement moins rapide que FM\_index.
    - ii. Les variations sont minimales, signe de bonnes performances à des tailles croissantes des données.
  - **Time FM :**
    - iii. Elle présente une très légère fluctuation (forme d'onde), mais reste en moyenne **plus basse que la courbe rouge**.
    - iv. Cela indique que la structure ou l'algorithme FM\_index est globalement le plus efficace et stable dans ce contexte.
- ❖ **Graphe 3 et 4 (Le zoom de 3)**
- **Suffix array (Graphe 4)** : Très inefficace en termes de mémoire, surtout pour des tailles de données importantes.
  - **Tree et FM (Graphe 4)** : Hautement optimisés pour réduire la consommation mémoire, ce qui les rend plus adaptés pour des applications où les ressources sont limitées.
  - **Entre le Tree et FM (Graphe 3)** : Vous pouvez remarquer que, pour la mémoire, le Tree consomme légèrement plus que le FM

## 4. Conclusion finale

Le FM est le meilleur compromis entre **rapidité** et **faible consommation mémoire**, tandis que le **Tree** est également performant mais légèrement plus gourmand en mémoire. Le **Array**, en revanche, est moins adapté pour des applications nécessitant efficacité et optimisation des ressources.