SÉANCE
CHAÎNES DE CARACTÈRES

Julien Rodriguez & Guillaume Pérution-Kihli

Université de Montpellier
19 mars 2022

PLAN

1. **Recherche de motifs**
2. Distance de levenshtein
3. Trie
4. Arbre & tableau des suffixes

RECHERCHE DE MOTIFS

Algorithmes de recherche de motif :

- Knuth–Morris–Pratt $O(|P| + |S|)$[1]

[1] *p*41 *Programmation efficace*
[2] *p*46 *Programmation efficace*

RECHERCHE DE MOTIFS

Algorithmes de recherche de motif :

- Knuth–Morris–Pratt $O(|P| + |S|)$[1]
- Boyer–Moore–Horspool $O(|P|.|S|)$ dans le pire cas

[1] *p41 Programmation efficace*
[2] *p46 Programmation efficace*

RECHERCHE DE MOTIFS

Algorithmes de recherche de motif :

- Knuth–Morris–Pratt $O(|P| + |S|)$[1]
- Boyer–Moore–Horspool $O(|P|.|S|)$ dans le pire cas
- Rabin-Karp[2] $O(|P|.|S|)$

[1] *p41 Programmation efficace*
[2] *p46 Programmation efficace*

RECHERCHE DE MOTIFS

Algorithmes de recherche de motif :

- Knuth–Morris–Pratt $O(|P| + |S|)$[1]
- Boyer–Moore–Horspool $O(|P|.|S|)$ dans le pire cas
- Rabin-Karp[2] $O(|P|.|S|)$
- Aho-Corasick $O(|S|)$ une fois le dictionnaire construit

[1] *p41 Programmation efficace*
[2] *p46 Programmation efficace*

RECHERCHE DE MOTIFS

Algorithmes de recherche de motif :

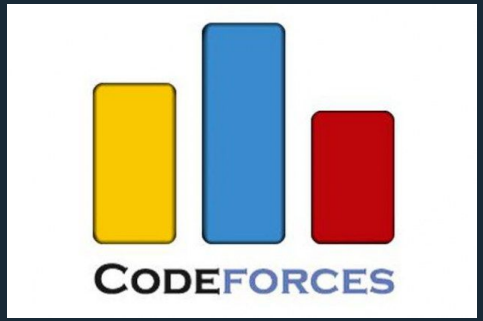
- Knuth–Morris–Pratt $O(|P| + |S|)$[1]
- Boyer–Moore–Horspool $O(|P|.|S|)$ dans le pire cas
- Rabin-Karp[2] $O(|P|.|S|)$
- Aho-Corasick $O(|S|)$ une fois le dictionnaire construit

En python : **string.find(pattern, start, end)**, **string.index(pattern, start, end)** ou l'opérateur **in**.

[1] *p41 Programmation efficace*
[2] *p46 Programmation efficace*

SOUS SÉQUENCES DE TAILLE 2

Soient deux chaînes s et t composées de caractère minuscule. Un mouvement consiste à remplacer un caractère dans s par n'importe quelle autre lettre minuscule. L'objectif consiste à maximiser le nombre d'occurrences de t dans s en effectuant au plus k mouvement.



Entrée *Ligne 1 :* n et k la taille de s et le nombre maximum de remplacement possible. *Ligne 2 :* La chaîne s de taille n *Ligne 3 :* La chaîne t de taille 2.

Limites $2 \leq n \leq 200;\ 0 \leq k \leq n$. Temps : 2 s. Mémoire : 256 MB.

Sortie Un entier correspondant au nombre maximum d'occurrences de t dans s avec au plus k changement dans s.

PLAN

1. Recherche de motifs
2. **Distance de levenshtein**
3. Trie
4. Arbre & tableau des suffixes

DISTANCE DE LEVENSHTEIN

On appelle distance de Levenshtein entre deux chaînes M et P le coût minimal pour transformer M en P en effectuant les seules opérations élémentaires (au niveau d'un caractère) suivantes :

- substitution ;
- insertion ;
- suppression.

$$\mathrm{lev}(a,b) = \begin{cases} \max(|a|,|b|) & \text{si } \min(|a|,|b|) = 0, \\ \mathrm{lev}(a-1,b-1) & \text{si } a[0] = b[0], \\ 1 + \min \begin{cases} \mathrm{lev}(a-1,b) \\ \mathrm{lev}(a,b-1) \\ \mathrm{lev}(a-1,b-1) \end{cases} & \text{sinon.} \end{cases}$$

PLAN

1. Recherche de motifs
2. Distance de levenshtein
3. **Trie**
4. Arbre & tableau des suffixes

TRIE

Définition : un trie ou arbre préfixe, est une structure de données ayant la forme d'un arbre enraciné. Il est utilisé pour stocker une table associative où les clés sont généralement des chaînes de caractères.

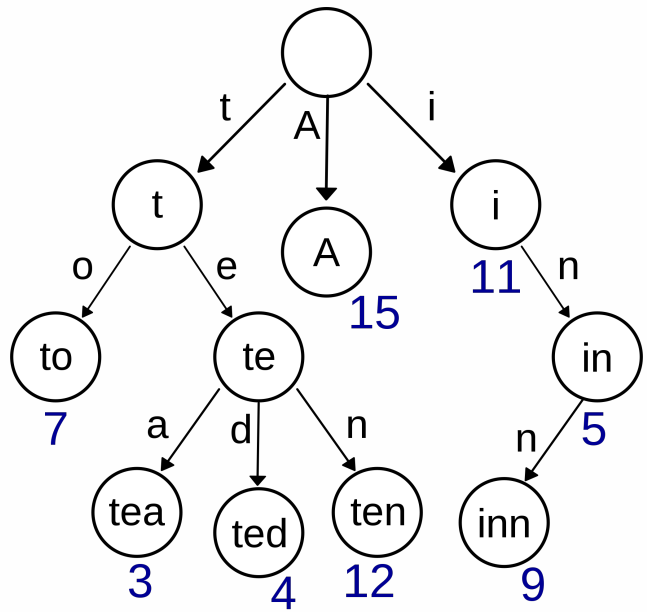


Fig. : Un trie pour les clés "A", "to", "tea", "ten", "ted", "i", "in", et "inn"

TRIE

Définition : un trie ou arbre préfixe, est une structure de données ayant la forme d'un arbre enraciné. Il est utilisé pour stocker une table associative où les clés sont généralement des chaînes de caractères.

Applications (liste non-exhaustive) :

- implémenter un tableau associatif ou un set (en pratique table de hachage souvent plus adaptée)
- trouver des redondances dans certains algorithmes de compression
- implémenter des algorithmes de correction orthographique[3]
- complétion automatique
- recherche de préfixe ou de suffixe

[3] *p38 Programmation efficace*

PLAN

1. Recherche de motifs
2. Distance de levenshtein
3. Trie
4. **Arbre & tableau des suffixes**

ARBRE DES SUFFIXES

Définition : structure de données arborescente contenant tous les suffixes d'un texte. L'arbre des suffixes est utilisé pour l'indexation de textes et la recherche de motifs.

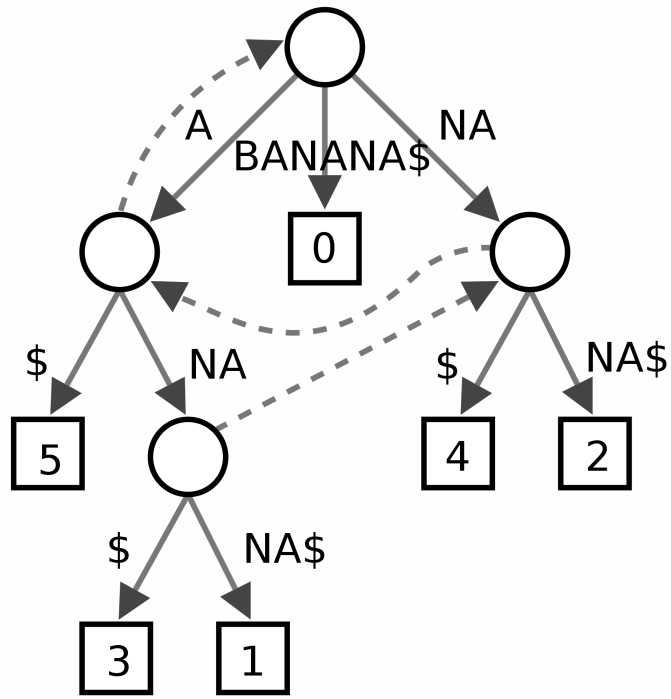


Fig. : Arbre des suffixes de BANANA

ARBRE DES SUFFIXES

Définition : structure de données arborescente contenant tous les suffixes d'un texte. L'arbre des suffixes est utilisé pour l'indexation de textes et la recherche de motifs.

Applications (liste non-exhaustive) :

- recherche de motifs
- détecter des répétitions : par exemple, si un nœud a deux fils, on sait que la chaîne qui le précède est répétée deux fois
- détecter des palindromes

ALGORITHME D'UKKONEN

- Algorithme classique : algorithme d'Ukkonen
- Construction en temps linéaire
- Mais en pratique :
  - L'arbre est gourmand en mémoire
  - Structure de données pas adaptée pour le cache processeur : peut être plus lent que des structures de données ayant une plus forte complexité mais plus adadptées au cache processeur
  - Algorithme très long, beaucoup de temps nécessaire pour le copier et beaucoup d'erreurs potentielles

TABLEAU DES SUFFIXES

Définition : pour un mot donné, le tableau contient une liste d'entiers qui correspondent aux positions de début des suffixes du mot, lorsqu'ils sont triés selon l'ordre lexicographique. L'objectif du tableau est de fournir les mêmes facilités de recherche qu'un arbre des suffixes tout en réduisant la taille mémoire utilisée.

Suffixe	Position de début
a	10
abra	7
abracadabra	0
acadabra	3
adabra	5
bra	8
bracadabra	1
cadabra	4
dabra	6
ra	9
racadabra	2

Tab. : Tableau des suffixes de abracadabra