

# Rapport

Moteur de Génération et Validation Morphologique Arabe

Eya Gharbi – Zaghouani Ayoub

Classe 1ING4

## 1. Présentation générale

Ce projet implémente un moteur de dérivation morphologique arabe capable de générer des formes dérivées à partir d'un radical trilitère et d'un schème, et de valider si un mot dérivé appartient à un radical donné.

## 2. Chargement des données

Au démarrage :

- `roots.txt` charge les racines dans un ABR.
- `patterns.txt` charge les schèmes dans une table de hachage.

## 3. Structures de données

### 3.1. Arbre Binaire de Recherche (ABR)

Les racines trilitérales (ex : ك-ت-ب) sont stockées sous forme compacte (كتب). La comparaison repose sur l'ordre Unicode.

Chaque nœud contient :

- la racine compacte.
- une liste chaînée de dérivés.
- deux pointeurs (gauche/droite).

**Normalisation** : suppression des diacritiques, unification des variantes d'alef, vérification stricte du format.

### 3.2. Liste chaînée

Chaque racine possède une liste de mots dérivés (mot + fréquence). Elle permet un stockage léger et évite les recalculs.

### 3.3. Table de hachage

Les schèmes sont stockés dans une table de hachage de taille fixe  $P = 37$  (nombre premier).

Fonction de hachage polynomiale :

$$h(k) = \left( \sum_{i=0}^{|k|-1} \text{ord}(k_i) \cdot 131^i \right) \bmod 37$$

**Type d'entrée.** Les schèmes sont validés :

- présence des lettres ف ع ل.
- lettres arabes uniquement.
- conservation de la shadda (si présente).

**Normalisation :** Chaque schème est normalisé (diacritiques supprimés sauf shadda, variantes d’alef unifiées). Les schèmes doivent contenir ف ع ل et respecter une longueur minimale.

**Règle de dérivation :** Elle est identique au schème normalisé (`rule = pattern`). La dérivation consiste à substituer ف ع ل par les lettres de la racine.

**Chaînage :** Chaque case de la table pointe vers une **liste chaînée** de schèmes. Si deux schèmes ont le même hash (collisions), ils sont ajoutés à la liste du même bucket. La recherche parcourt cette liste jusqu’à trouver la clé exacte.

## 4. Algorithmes

### 4.1. Génération

1. Normalisation de la racine donnée
2. Recherche dans l’ABR
3. Recherche du schème dans la table de hachage
4. Application de la règle de dérivation
5. Stockage dans la liste chaînée de la racine

Complexité :

$$T_{\text{gen}} = O(\log n) + O(|s|) = O(\log n)$$

Pour la génération d’une famille sur les  $p$  schèmes de longueur moyenne  $\bar{s}$  :

$$T_{\text{family}}(n, p) = O(\log n) + O\left(\sum_{i=1}^p (1 + |s_i|)\right) = O(\log n + p \cdot \bar{s})$$

### 4.2. Validation

Validation par régénération exhaustive :

1. Recherche racine dans l’ABR
2. Parcours des  $P$  schèmes
3. Génération + comparaison

$$T_{\text{val}} = O(\log n + P)$$

Cette méthode garantit la cohérence morphologique et évite les ambiguïtés d’une reconstruction inverse plus complexe.

## 5. Complexité Algorithmique

Soient :

- $n$  : nombre de racines
- $P$  : nombre total de schèmes

Opération	Cas moyen	Pire cas
Recherche ABR	$O(\log n)$	$O(n)$
Insertion ABR	$O(\log n)$	$O(n)$
Suppression ABR	$O(\log n)$	$O(n)$
Recherche hash	$O(1)$	$O(P)$
Insertion hash	$O(1)$	$O(P)$
Génération (1 mot)	$O(\log n)$	$O(n)$
Validation (1 mot)	$O(\log n + P)$	$O(n + P)$
Chargement racines	$O(n \log n)$	$O(n^2)$
Chargement schèmes	$O(P)$	$O(P^2)$

## 6. Choix algorithmiques

**Arbre Binaire de Recherche :** Une structure linéaire (tableau ou liste) impliquerait une recherche en  $O(n)$ . L'ABR permet une recherche, insertion et suppression en  $O(\log n)$  en moyenne, ce qui réduit significativement le coût des accès fréquents aux racines. Le choix d'un ABR simple a été privilégié pour limiter la complexité d'implémentation et le coût mémoire.

**Table de hachage :** Les schèmes sont accédés très fréquemment lors de la génération et surtout de la validation. Une table de hachage permet un accès en  $O(1)$  en moyenne, plus performant qu'un ABR ( $O(\log P)$ ). La taille  $P$  en nombre premier plus grand que le nombre de schèmes réduit les collisions. Le chaînage a été choisi plutôt que l'adressage ouvert afin de simplifier la gestion des collisions et éviter les problèmes liés au redimensionnement.

**Liste chaînée :** Chaque racine peut produire un ensemble variable de dérivés. Une liste chaînée permet un stockage flexible sans pré-allocation et une rapidité d'accès aux éléments. Le coût reste acceptable car le nombre de dérivés par racine est limité et l'approche évite la complexité d'une structure plus lourde.

**Validation exhaustive :** La validation se fait par régénération de tous les schèmes, puis comparaison. Une reconstruction inverse serait plus complexe et ambiguë morphologiquement, donc ce choix garantit la cohérence morphologique.

**Normalisation :** La suppression des diacritiques et l'unification des variantes Unicode assurent des comparaisons robustes et évitent les incohérences dues aux différentes formes d'écriture arabe.

## 7. Difficultés rencontrées

- Assurer une normalisation Unicode cohérente entre racines et schèmes.
- Garantir la cohérence entre génération et validation afin d'éviter toute divergence logique.
- Gérer efficacement les collisions dans la table de hachage tout en conservant une recherche rapide.

- Maintenir une complexité maîtrisée tout en conservant une implémentation claire et modulaire.

## 8. Point d'originalité

Le système est exploité comme un **dictionnaire morphologique** : chaque schème est associé à une définition, et chaque dérivé validé peut être affiché avec son sens linguistique. Cette exploitation donne une valeur appliquée aux structures de données. Cette exploitation démontre l'application concrète des structures algorithmiques développées.