Rapport Technique

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. L'objectif principal est d'optimiser les performances en choisissant des structures de données adaptées afin de minimiser la complexité algorithmique moyenne.

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 trilittérales (ex : ب-ت-ك) 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 et vérification stricte du format.

Insertion : descente récursive selon l'ordre lexicographique. Les doublons sont refusés.

Suppression : trois cas classiques (feuille, un enfant, deux enfants avec remplacement par le successeur).

Une structure linéaire aurait impliqué une recherche en $O(n)$. L'ABR permet une recherche en $O(\log n)$ en moyenne, réduisant significativement le coût des accès fréquents aux racines.

3.2. Liste chaînée

Chaque racine possède une liste de mots dérivés (mot + fréquence). Cette structure permet un stockage léger et évite les recalculs lors des validations répétées.

3.3. Table de hachage

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

Fonction de hachage polynomiale :

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

Validation des schèmes :

- présence des lettres ف ع ل,
- lettres arabes uniquement,
- conservation éventuelle de la shadda.

Normalisation : suppression des diacritiques (sauf shadda) et unification des variantes Unicode.

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

Chaînage : chaque bucket contient une liste chaînée. En cas de collision, les schèmes sont ajoutés à la liste correspondante.

Le pire cas correspond à une collision totale où tous les schèmes sont placés dans un seul bucket. En pratique, la fonction polynomiale et le choix d'un nombre premier rendent ce cas hautement improbable.

4. Algorithmes

4.1. Génération

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

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

Pour la génération d'une famille sur P schèmes de longueur moyenne $\bar{s}$:

$$T_{\text{family}}(n, P) = O(\log n + P \cdot \bar{s})$$

4.2. Validation

Validation par régénération exhaustive :

1. Recherche de la racine dans l'ABR
2. Parcours des P schèmes
3. Génération et 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.

Validation exhaustive : La validation repose sur la régénération complète des formes possibles à partir de tous les schèmes. Une reconstruction inverse serait plus complexe et ambiguë morphologiquement. La régénération garantit la cohérence avec l'algorithme de génération et assure l'unicité de la logique de dérivation.

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 moteur est exploité comme un dictionnaire morphologique : chaque schème peut être associé à une définition, et chaque dérivé validé peut être enrichi sémantiquement. Cette exploitation illustre l'application concrète des structures de données implémentées.

9. Conclusion

Le système combine ABR, table de hachage et listes chaînées de manière cohérente. Les performances sont dominées par $O(\log n)$ en moyenne pour les racines et $O(1)$ amorti pour les schèmes, ce qui rend le moteur efficace et extensible.