AI02
TP 4: Les Arbres Binaires de Recherche

NKOUNKOU Hérald
SAIDNA Amine

18 décembre 2025

Dans ce TP, nous utiliserons les arbres binaires de recherche pour implémenter un exemple d'indexation et de recherche sur un fichier contenant un texte quelconque.

1 Introduction

Ce TP implémente un système d'indexation de texte utilisant un Arbre Binaire de Recherche (ABR) pour indexer les mots d'un fichier avec leurs positions et permettre des opérations de recherche et reconstruction.

2 Structures de données

2.1 Structures de base

Les structures demandées ont été implémentées avec les champs suivants :

- **T_Position** : Position d'un mot avec `numeroLigne`, `ordre`, `numeroPhrase` et `suivant`.
- **T_Noeud** : Nœud ABR avec `mot`, `nbOccurences`, `listePositions`, `dernierePosition`, `filsGauche` et `filsDroit`.
- **T_Index** : Structure principale avec `racine`, `nbMotsDistincts`, `nbMotsTotal`, `tabPhrases` et `nbPhrasesCapacite`.

2.2 Champs supplémentaires

dernierePosition : Pointeur vers la dernière position permettant l'insertion en O(1) à la fin de liste.

tabPhrases : Tableau dynamique stockant les phrases complètes pour un accès direct en O(1).

Justification : L'indexation séquentielle ajoute les positions dans l'ordre, le pointeur de queue évite les parcours. Le cache de phrases, construit pendant l'indexation, évite les reconstructions ultérieures.

3 Complexités des fonctions

3.1 Fonctions de base

- **initIndex** : O(1)
- **ajouterPosition** : O(k) - k positions dans la liste pire cas O(n)
- **ajouterOccurence** : O(h) - Recherche O(h), insertion position O(1)
- **indexerFichier** : O($N \times h + M$) - N mots, M caractères (construction linéaire via arithmétique de pointeurs)
- **afficherIndex** : O($N \times p$) - N mots, p positions moyennes
- **rechercherMot** : O(h)

3.2 Fonctions avancées

- **verifierCapacitePhrases** : O(1) amorti
- **afficherOccurencesMot** : O($h + k$) - Recherche O(h), affichage O(1) par phrase via cache
- **construireTexte** : O(P) - P phrases, simple parcours du cache
- **libererIndex** : O($P + N$)

4 Analyse

Points forts :

- Insertion O(1) en fin de liste via pointeur de queue
- Accès O(1) aux phrases complètes via cache
- Construction linéaire sans `strcat` répété
- Gestion dynamique du cache avec réallocation automatique
- Insensibilité à la casse avec `strcasecmp`

Compromis : Le cache consomme O(M) mémoire mais offre des gains de temps significatifs sur `afficherOccurencesMot` (pas de reconstruction) et `construireTexte` (parcours direct).

Limites : L'ABR peut devenir déséquilibré (pire cas O(n)). Solution : AVL ou arbre rouge-noir. Capacité initiale du cache (100) pourrait être ajustée dynamiquement.

5 Conclusion

L'implémentation respecte les spécifications avec des structures efficaces. Le pointeur de queue et le cache de phrases assurent des complexités adaptées au traitement de fichiers texte de taille variable.

⋆ ⋆ ⋆