AI02
TP 4: Les Arbres Binaires de Recherche

NKOUNKOU Hérald
SAIDNA Amine

17 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 supplémentaires

En plus des structures de base demandées (`T_Position`, `T_Noeud`, `T_Index`), nous avons ajouté :

- **T_MotDansPhrase** : Représente un mot dans une phrase avec son ordre d'apparition. Contient `mot`, `ordre` et `suivant`.
- **T_MapPhrases** : Tableau dynamique où chaque case correspond à un numéro de phrase et contient la liste chaînée des mots de cette phrase triés par ordre.

Justification : Pour la question 6, au lieu de reconstruire toutes les phrases à chaque fois, nous reconstituons uniquement la phrase concernée à la demande via `collecterMotsPhrase`. Pour la question 7, la map permet un seul parcours de l'arbre pour organiser tous les mots par phrase, optimisant la reconstruction complète du texte.

3 Complexités des fonctions

3.1 Fonctions de base

- **ajouterPosition** : $O(n)$ - Insertion dans une liste triée de n positions.
- **ajouterOccurence** : $O(h + p)$ - Recherche dans l'ABR de hauteur h puis ajout dans une liste de p positions. Cas moyen : $O(\log n)$, pire cas : $O(n)$.
- **indexerFichier** : $O(m \times h)$ - Appel de `ajouterOccurence` pour m mots.
- **afficherIndex** : $O(n \times p)$ - Parcours infixe de l'arbre (n mots) et affichage de p positions par mot.
- **rechercherMot** : $O(h)$ - Recherche classique dans un ABR.

3.2 Fonctions auxiliaires

- **insererMotTrie** : $O(k)$ - Insertion dans une liste de k mots d'une phrase.
- **collecterMotsPhrase** : $O(n)$ - Parcours complet de l'arbre pour collecter les mots d'une phrase spécifique.
- **reconstruireEtAfficherPhrase** : $O(n+k)$ - Collection ($O(n)$) puis affichage ($O(k)$) d'une phrase de k mots.

3.3 Fonctions avancées

- **afficherOccurencesMot** : $O(h + occ \times n)$ - Recherche du mot ($O(h)$) puis reconstruction à la demande de chaque phrase contenant le mot. Pour `occ` occurrences, on effectue `occ` parcours de l'arbre. Cette approche est optimale lorsque le mot apparaît peu de fois.
- **remplirMapPhrases** : $O(n \times p)$ - Parcours infixe de l'arbre et insertion des positions dans la map.
- **construireTexte** : $O(n \times p + k \times m)$ - Remplissage de la map ($O(n \times p)$) puis écriture séquentielle de k phrases contenant en moyenne m mots.

4 Analyse

Points forts :

- Insensibilité à la casse avec `strcasecmp`
- Question 6 optimisée : reconstruction à la demande uniquement des phrases nécessaires
- Question 7 : un seul parcours de l'arbre via `T_MapPhrases`
- Gestion dynamique de la mémoire (réallocation automatique de la map)
- Libération propre de toute la mémoire allouée

Limites et optimisations :

- L'ABR peut devenir déséquilibré (pire cas $O(n)$). Solution : utiliser un AVL ou arbre rouge-noir pour garantir $O(\log n)$.
- Pour la question 6, si un mot apparaît très fréquemment, les reconstructions répétées peuvent être coûteuses. Solution : maintenir un cache des phrases déjà reconstruites.
- La capacité initiale de la map (100) est arbitraire. Solution : estimation basée sur le nombre de phrases détectées.

5 Conclusion

L'implémentation respecte toutes les spécifications avec deux approches distinctes : reconstruction à la demande pour l'affichage des occurrences (optimale pour peu d'occurrences) et reconstruction globale via map pour la génération du texte complet. Les complexités sont adaptées à chaque cas d'usage.

⋆ ⋆ ⋆