Implémentation d'un ABRnois

Auteur : Vincent PLESSY
Date : 2 Mai 2025
Groupe : L2 Informatique, Algorithmique des Arbres

1. Introduction

Ce rapport présente mon implémentation d'une structure de données appelée ABRnois, combinant les caractéristiques d'un arbre binaire de recherche (ABR) et d'un arbre tournois. L'objectif principal de ce projet était de développer un programme capable d'analyser des corpus de texte pour générer une liste des mots les plus fréquents de la langue française, similaire au travail du linguiste Étienne Brunet, mais sans la fusion des mots de même base lexicale.

2. Description des fonctions implémentées

2.1 Fonctions de base demandées

- **Noeud * alloue_noeud(char * mot)** : Cette fonction crée un nouveau nœud contenant une copie du mot passé en paramètre, avec une occurrence initialisée à 1. Elle gère correctement les erreurs d'allocation mémoire.
- **int exporte_arbre(char * nom_pdf, ABRnois A)** : Cette fonction génère une représentation graphique de l'arbre au format PDF, en utilisant Graphviz. Elle crée d'abord un fichier DOT décrivant la structure de l'arbre, puis convertit ce fichier en PDF.
- **void rotation_gauche(ABRnois * A)** et **void rotation_droite(ABRnois * A)** : Ces fonctions effectuent des rotations gauche et droite sur l'arbre, opérations essentielles pour maintenir la structure d'arbre tournois. Les rotations sont utilisées à la fois lors de l'insertion et de la suppression.
- **int insert_ABRnois(ABRnois * A, char * mot)** : Cette fonction insère un mot dans l'arbre ABRnois. Si le mot est déjà présent, son occurrence est incrémentée. Après l'insertion, la structure d'arbre tournois est maintenue en effectuant des rotations si nécessaire.
- **int extrait_priorite_max(ABRnois * A, Liste * lst)** : Cette fonction extrait tous les nœuds ayant l'occurrence maximale (celle de la racine) et les place dans une liste triée par ordre alphabétique. Les nœuds sont retirés de l'arbre et la fonction retourne le nombre de nœuds extraits.

2.2 Fonctions auxiliaires

J'ai également implémenté plusieurs fonctions auxiliaires pour faciliter le développement et améliorer la modularité du code :

- **void libere_noeud(Noeud * n)** et **void libere_arbre(ABRnois A)** : Ces fonctions permettent de libérer la mémoire allouée pour un nœud ou pour l'ensemble de l'arbre.

- **void ecrire_noeud_dot(FILE * f, ABRnois A, int id)** : Cette fonction auxiliaire est utilisée par exporte_arbre pour écrire la description d'un nœud et de ses fils dans le fichier DOT.
- **int descendre_noeud(ABRnois * A, char * mot)** : Cette fonction fait descendre un nœud dans l'arbre pour qu'il devienne une feuille, en effectuant des rotations appropriées. Elle est utilisée comme étape préliminaire à la suppression.
- **int supprime_ABRnois(ABRnois * A, char * mot)** : Cette fonction supprime un mot de l'arbre, en le faisant d'abord descendre pour qu'il devienne une feuille, puis en supprimant cette feuille.
- **int insere_dans_liste_triee(Liste * L, Noeud * n)** : Cette fonction insère un nœud dans une liste triée par ordre alphabétique, utilisée lors de l'extraction des nœuds de priorité maximale.
- **int extraire_noeuds_max_rec(ABRnois * A, Liste * lst, int priorite_max, int * nb_extraits)** : Fonction récursive appelée par extrait_priorite_max pour extraire les nœuds ayant la priorité maximale.
- **int nettoie_mot(char * mot)** : Cette fonction nettoie un mot en supprimant la ponctuation et en le convertissant en minuscules, afin de normaliser les entrées pour l'analyse de fréquence.

3. Structure du programme principal

Le programme principal (main) effectue les opérations suivantes :

1. **Traitement des arguments** : Analyse des options de la ligne de commande (-g pour générer des PDF, -n pour limiter le nombre de mots extraits).
2. **Lecture des fichiers corpus** : Parcours de chaque fichier corpus mot par mot, nettoyage des mots et insertion dans l'arbre ABRnois.
3. **Génération des représentations graphiques** : Si l'option -g est activée, création d'un fichier PDF après chaque insertion et suppression.
4. **Extraction des mots les plus fréquents** : Suppression progressive des mots de l'arbre par ordre de fréquence décroissante.
5. **Écriture des résultats** : Écriture des mots extraits dans le fichier de sortie avec leur pourcentage d'occurrence, les plus fréquents en premier et en respectant l'ordre alphabétique en cas d'égalité de fréquence.

4. Difficultés rencontrées

4.1 Maintien de la double structure

La principale difficulté a été de maintenir correctement la double structure d'ABR et d'arbre tournois, en particulier lors des opérations de suppression. Il fallait s'assurer que les rotations effectuées pour faire descendre un nœud ne perturbaient pas l'ordre des valeurs caractéristique d'un ABR.

L'implémentation de l'algorithme de descente d'un nœud pour le transformer en feuille a nécessité une attention particulière. J’ai dû prendre en compte différents cas (nœud avec un seul enfant, nœud avec deux enfants) et choisir la rotation appropriée dans chaque situation.

4.2 Extraction des nœuds de priorité maximale

L'extraction des nœuds ayant la priorité maximale a également été complexe, car il fallait maintenir à la fois l'intégrité de l'arbre et construire une liste triée par ordre alphabétique. J’ai choisi de mettre en œuvre une fonction récursive qui parcourt l'arbre, identifie les nœuds à extraire et les insère dans la liste résultat.

Le principal défi était de gérer correctement le cas où un nœud à extraire avait des enfants. J’ai implémenté une technique de remplacement par le successeur inorder pour maintenir la structure d'ABR.

4.3 Gestion de la mémoire

La gestion de la mémoire a constitué un autre défi important.J’ai dû faire attention à libérer correctement la mémoire allouée pour éviter les fuites, en particulier lors de l'extraction des nœuds de priorité maximale et de la suppression de l'arbre.

4.4 Traitement des fichiers texte

Le traitement des fichiers texte a présenté des défis spécifiques :

- La normalisation des mots (suppression de la ponctuation, conversion en minuscules)
- La gestion des mots invalides ou trop courts
- La lecture mot par mot de fichiers potentiellement volumineux

J’ai implémenté une fonction `nettoie_mot` qui s'occupe de ces aspects, en vérifiant notamment que le mot contient au moins deux caractères alphabétiques.

5. Comparaison des méthodes

5.1 Avantages de la structure ABRnois

La structure ABRnois présente plusieurs avantages par rapport à d'autres structures de données :

- **Efficacité de recherche** : Grâce à sa structure d'ABR, elle permet une recherche efficace (en O(log n) dans le cas moyen) des éléments stockés.
- **Extraction prioritaire facilitée** : La structure d'arbre tournois permet d'identifier rapidement les éléments de priorité maximale, puisqu'ils se trouvent à la racine.
- **Flexibilité** : Cette structure hybride permet de combiner deux critères différents (ordre lexicographique et priorité) dans une seule structure de données.

5.2 Inconvénients et limitations

La structure ABRnois présente également quelques limitations :

- **Complexité des opérations** : Les opérations d'insertion et de suppression sont plus complexes que dans un ABR ou un tas binaire classique, ce qui peut affecter les performances pour des ensembles de données très volumineux.
- **Variabilité de la structure** : La structure de l'arbre peut varier considérablement en fonction de l'ordre d'insertion et de suppression des éléments, ce qui peut rendre son comportement moins prévisible.
- **Équilibre non garanti** : Contrairement aux arbres AVL ou arbres rouge-noir, la structure ABRnois ne garantit pas un équilibrage de l'arbre, ce qui peut conduire à une dégradation des performances dans certains cas.

5.3 Améliorations possibles

Plusieurs améliorations pourraient être apportées à mon implémentation :

- **Optimisation de la mémoire** : Utilisation d'une structure plus compacte pour stocker les mots, par exemple avec un trie ou un tableau de hachage pour les mots les plus fréquents.
- **Traitement de texte amélioré** : Mise en œuvre d'une analyse plus sophistiquée des mots, avec reconnaissance des formes fléchies ou des variantes orthographiques.
- **Parallélisation** : Pour des corpus très volumineux, une approche parallèle pourrait être envisagée, avec fusion ultérieure des résultats.

6. Répartition du travail

En tant que monôme, j'ai dû organiser mon travail de manière méthodique pour réaliser l'ensemble du projet :

1. **Phase d'analyse et de conception** :
    - Étude approfondie du sujet et compréhension des spécificités de la structure ABRnois
    - Conception des algorithmes d'insertion et de suppression
    - Planification des différentes fonctionnalités à implémenter
2. **Phase d'implémentation** :
    - Développement des fonctions de base (alloue_noeud, rotations)
    - Implémentation de l'algorithme d'insertion
    - Développement des fonctions de descente et suppression
    - Implémentation de l'extraction de priorité maximale
    - Programmation des fonctionnalités auxiliaires et du programme principal
3. **Phase de test et de débogage** :
    - Validation des fonctions individuelles avec des cas de test simples
    - Tests d'intégration avec des jeux de données plus complexes
    - Vérification de la gestion mémoire et correction des fuites éventuelles
    - Tests de performance sur des corpus de tailles variées

Cette approche méthodique m'a permis de gérer efficacement la complexité du projet tout en maintenant une vision globale de son avancement.

7. Conclusion

L'implémentation de la structure ABRnois m’a permis d'explorer une solution hybride intéressante pour le problème de l'analyse de fréquence des mots. Cette structure combine efficacement les avantages des arbres binaires de recherche et des arbres tournois, facilitant à la fois la recherche et l'extraction des éléments les plus prioritaires.

Le programme développé permet d'analyser efficacement des corpus textuels et de générer une liste des mots les plus fréquents, avec leurs pourcentages d'occurrence. Les options supplémentaires (génération de PDF, limitation du nombre de mots) offrent une flexibilité appréciable pour différentes utilisations.

Malgré quelques limitations inhérentes à la structure ABRnois, mon implémentation constitue une solution fonctionnelle et efficace pour le problème posé.