ADLOUNI Selma **GROUPE 2**
BALLIER Thibault

PROJET B46: Dictionnaire

Télécom-Lille

Au cours de notre étude des structures de données, nous avons été amenés à implémenter une structure Dictionnaire associant un mot et sa définition de telle sorte que l'utilisateur puisse la mettre à jour ou y effectuer des recherches de définitions.

En effet, l'objectif premier de ce projet est, non seulement de réussir à utiliser un dictionnaire répondant à plusieurs requêtes (d'insertion, de suppression ou de recherche,..) mais aussi d'utiliser plusieurs types de structures de données TAD notamment, les listes chaînées, les arbres binaires de recherche, et les tables de hachage.

Comment structurer notre programme tout en dynamisant l'espace mémoire ? Et quels sont les points forts et faibles de chaque type de structure utilisé ?

Nous allons nous intéresser dans un premier temps, à cerner les démarches de réalisation de ce projet, puis à l'explication des algorithmes respectifs à chaque fonction. Puis, dans un dernier temps, nous allons relater les points forts et les limites des structures étudiées sans en oublier les risques d'utilisation.

1. PREMIERE APPROCHE : Démarches d'investigation

1.1. Objectif

L'objectif de ce projet est de réussir à implémenter un dictionnaire avec quatre fonctions demandées, et ce, suivant trois types de structures : Listes chainées, arbres binaires de recherche et tables de hachage.

Par ailleurs, l'étude séparée de ces trois structures nous permettra d'en dégager les points forts mais aussi leurs limites respectives.

1.2. Matériel

Nous disposons de trois fichiers contenant des mots ainsi que leur définition (ordonné, désordonné, et pour le test des fonctions).

1.3. Fonction demandées

Les trois fonctions demandées sont les suivantes :

- ✓ Insérer : permet de mettre à jour le dictionnaire en ajoutant un mot avec sa définition.
- ✓ Supprimer : permet de mettre à jour le dictionnaire en supprimer un mot avec sa définition.
- ✓ Rechercher : permet de retourner la définition du mot saisi.
- ✓ Rechercher approximatif : permet de retourner tous les mots débutant avec les caractères saisis.

2. STRUCTURE : Principe

2.1. Listes chaînées

Les listes chaînées sont des structures formées de maillons. Chaque maillon, dans le cas du dictionnaire, possède une case avec le couple mot-définition. Le maillon est relié à son suivant dans la liste.

2.2. Arbres binaire de recherche

Les arbres binaires de recherche sont une structure constituée de plusieurs nœuds. Chaque nœud possède un fils droit et un fils gauche, le fils droit étant le plus grand élément. Insérer ou supprimer un élément dans un arbre de recherche revient non seulement, à parcourir l'arbre mais aussi à préserver l'ordre binaire de l'arbre.

2.3. Table de hachage

Une table de hachage prend une chaîne de caractères en entrée et retourne un nombre. Ce dernier correspond à l'indice dans la table où sont stockées les données.

3. FONCTION : Algorithmes

3.1. Listes chainées :

Pour la plupart des fonctions effectuées sur les listes chainées, nous étions obligés de parcourir ces listes maillon par maillon, jusqu'à retrouver une position ou un mot.

- ✓ Pour la fonction « insérer », on crée un nouveau maillon avec le couple mot-définition, on l'insère en premier tout en le reliant au premier élément de la chaine afin de préserver la continuité de la liste.
- ✓ Pour la fonction « supprimer », on parcourt la chaine maillon par maillon. Une fois l'élément correspondant trouvé, on le supprime et on relie le maillon précèdent à son deuxième suivant.
- ✓ Pour la fonction « rechercher », on parcourt la chaine jusqu'à trouver le mot dans la chaine, on retourne la définition correspondante.
- ✓ Pour la fonction « recherche approximative », on trouve tous les mots avec les occurrences du « début » recherché, et on retourne tous ces mots, insérés au fur et à mesure dans une nouvelle liste chainée.

3.2. Arbres binaires de recherche :

Par ailleurs, les arbres binaires de recherche utilisent la même notion de parcours que les listes chainées. Cependant, un enjeu de plus rentre en jeu : Il faut sauvegarder l'ordre de l'arbre. Le fils plus grand que son parent est à droite.

- ✓ Pour la fonction « insère », on crée un nouveau, nœud avec le couple mot-définition, on parcourt le tableau d'une manière particulière : si l'élément est plus grand on se décale à droite s'il est plus petit on se décale à gauche.
- ✓ Pour la fonction « supprimer », de la même façon on parcourt l'arbre. Une fois arrivé à l'élément correspondant, on le supprime et on fait remonter à sa place le plus petit des plus grands de ses fils droits ou le plus grand des plus petit de ses fils gauches, dans le but précis de sauvegarder l'ordre de l'arbre.

- ✓ Pour la fonction « rechercher », on parcourt l'arbre jusqu'à trouver le mot dans la chaine, on retourne la définition correspondante.
- ✓ Pour la fonction « recherche approximative », on parcourt l'arbre en recherchant les occurrences du « début » et on retourne les mots portant ces occurrences, dans un nouvel arbre rempli au fur et à mesure.

3.3. Tables de hachage :

La structure « table de hachage » utilise une fonction de hachage qui prend en entrée une clef et renvoie un indice.

- ✓ Pour la fonction « insère », on trouve le hash du mot à insérer. L'indice retourné pointe vers l'emplacement dans la table où sont stockés les données sous forme de liste chainées. On ajoute l'élément dans la liste chainée (on utilise la fonction « insérer » des listes chainées).
- ✓ Pour la fonction « supprimer », on trouve le hash du mot à supprimer. L'indice retourné pointe vers l'emplacement dans la table. On supprime l'élément dans la liste chainée (on utilise la fonction « supprimer » des listes chainées.
- ✓ Pour la fonction « rechercher », le hash du mot recherché pointe vers un indice de la table. Grace à la fonction « rechercher » dans les listes chainées on retrouve la définition.
- ✓ Pour la fonction « recherche approximative », on parcourt le tableau de listes chainées et on utilise la fonction de recherche approximative des listes chainées et on retourne une nouvelle table de hachage dans laquelle on a inséré au fur et à mesure les éléments correspondants.

Note : On pourrait modifier la fonction de hachage de telle sorte à trier les entrées afin de retrouver plus facilement ce que l'on cherche.

4. MEMOIRE : Dynamisation des espaces mémoires

4.1. Problème

Les déclarations de variables en C présentent une limite à ne pas négliger. Lors de la compilation, la taille des variables doit être connue. Afin d'éviter une mauvaise gestion de la mémoire, on a procédé à une attribution d'un espace mémoire connue sous le procédé de l'allocation dynamique de la mémoire.

Techniquement, cette méthode a recours aux deux fonctions de la bibliothèque `<stdlib.h>` :

- ✓ *malloc* : pour l'allocation dynamique de mémoire ;
- ✓ *free* : pour la libération de mémoire.

Cette solution s'applique à toutes les structures utilisées. Les fonctions d'attribution et suppression de la mémoire sont établies au début de chaque structure

4.2. Comparaison de l'utilisation de l'espace mémoire

L'utilisation de l'espace mémoire de notre programme alloue pour chaque type une taille différente de celle des autres structures :

	Listes chaînées	Arbres binaires de recherche	Tables de hachage
Chargement du dictionnaire	39,97 Mo	42.15 Mo	41,57 Mo

5. RISQUES : Limites des structures utilisées

5.1. Comparaison des structures

A chaque type de structure de données, on peut à la fois trouver des facilités d'utilisation comme des difficultés.

- Les listes chaînées sont flexibles, mais il peut être long de retrouver un élément précis à l'intérieur car il faut les parcourir maillon par maillon.
- Les tables de hachage renvoient un résultat plus rapidement.
- Les arbres sont plus compliqués à gérer puisque, pour chaque fonction il faut parcourir l'arbre des deux côtés et garder l'ordre de l'arbre binaire de recherche.

5.2. Risques à gérer

5.2.1. Liste chaînée :

Si l'élément est à la fin de la chaîne, on doit la parcourir entièrement pour le trouver ce qui peut être long, comparé aux deux autres implémentations.

5.2.2. Arbre binaire de recherche :

Lors de l'utilisation des arbres binaires de recherche, si les valeurs rentrées sont triées, l'arbre ne servira plus à rien, car l'arbre ne sera plus qu'une liste chaînée avec des fonctions non adaptées pour une liste chainée, ce qui sera moins optimisé.

5.2.3. Tables hachage :

Lors de l'utilisation des tables de hachage, deux mots différents peuvent pointer vers un même indice. Il y a donc un risque de collisions. Une bonne fonction de hachage doit normalement produire le peu de collisions possible.

Les listes de hachage ne sont pas optimisées pour la recherche approximative de données, la maîtrise de l'algorithme de hachage pour trier les données est essentielle afin de pouvoir augmenter la vitesse de la recherche approximative.

En somme, la réalisation de ce projet a présenté plusieurs défis à relever, notamment l'élaboration de plusieurs structures de données mais aussi la gestion de la mémoire, sa dynamisation. Par ailleurs, il est important de noter que certaines structures posent plus de risques que d'autres. Une bonne gestion de ces risques évite à l'utilisateur un mauvais fonctionnement du programme réalisé.