Nom :TAVAKOL

Prenom : Zahra

Numero etudiant :22300070

Compte rendu TP/ Séquence 3 : Listes doublement chaînées et tri fusion

1. Description des choix d'implantation pour les structures de données

_Pour implémenter le tri fusion sur une liste doublement chaînée avec sentinelle, j'ai utilisé une structure intermédiaire SubList, qui contient uniquement deux pointeurs :

_head : pointe vers le premier élément réel de la sous-liste (sans la sentinelle),

_tail : pointe vers le dernier élément réel de la sous-liste.

Cette représentation permet de travailler localement sur une portion de la liste sans se soucier de la structure circulaire avec sentinelle. Cela facilite grandement le tri récursif et les opérations de découpage/fusion.

- **list_sort**

Avant de trier, je déconnecte la liste de sa sentinelle pour éviter des problèmes lors du traitement.

Une fois le tri terminé, je reconnecte proprement la tête et la queue à la sentinelle, en remettant sentinel->next et sentinel->previous, ainsi que les pointeurs opposés (head->previous, tail->next).

- **list_mergesort**

C'est une fonction récursive : elle divise la sous-liste en deux moitiés jusqu'à obtenir des sous-listes unitaires (ou vides).

Les pointeurs next et previous sont préservés dans chaque sous-liste, de sorte qu'elles restent utilisables directement dans la fusion

- **list_split**

J’utilise deux pointeurs slow et fast pour parcourir la liste et trouver son milieu.

Ensuite, je sépare la sous-liste en deux :

- slow->next = NULL pour couper le lien entre les deux moitiés,
- right->head->previous = NULL pour que la nouvelle tête droite ne pointe plus vers la gauche.

Je fais également un parcours complet de la partie droite pour retrouver correctement son tail, ce qui est essentiel pour terminer la fusion sans erreurs.

- **list_merge**

Fusionne deux sous-listes triées.

J'utilise un pointeur mergedTail qui suit le dernier élément ajouté à la nouvelle liste triée.

À chaque ajout :

Je mets mergedTail->next = next pour lier l’élément suivant.

Et next->previous = mergedTail pour la liaison arrière.

Si mergedHead == NULL, cela signifie que c’est le premier élément de la nouvelle liste — on l’initialise donc ici.

Les boucles finales ajoutent les éventuels éléments restants d'une des deux listes. Cela garantit qu’aucun élément n’est perdu et que tous les pointeurs sont bien mis à jour.

- **Complexité :**

Temps : O(n log n) — division en log(n) niveaux + O(n) pour chaque fusion.

Espace : O(1) supplémentaire, car aucune allocation mémoire n’est faite (hors pile d’appels récursifs).

Analyse personnelle du travail effectué

Le principal défi a été la **gestion correcte des pointeurs** dans une structure circulaire avec sentinelle.
J'ai rencontré plusieurs erreurs de segmentation au départ, causées par des next ou previous mal mis à jour (ou non initialisés). En utilisant **Valgrind**, j'ai pu identifier les accès mémoire invalides et corriger les pointeurs concernés.

J'ai aussi dû faire attention à bien **terminer chaque sous-liste par NULL pendant le tri**, puis **reconnecter à la sentinelle** une fois le tri terminé, pour restaurer la circularité de la liste.

Une fois ces problèmes résolus, le tri fonctionne parfaitement, et la structure chaînée est conservée de bout en bout.