COMPTE RENDU DU DM SKIPLIST

OBJECTIF

L'objectif de ce devoir est de programmer et de comparer, en termes de complexité et de performance les opérations de dictionnaire (insérer, rechercher, supprimer) sur une structure de données linéaire de type liste doublement chaînée. On demande donc d'implémenter une structure linéaire et probabiliste (SkipList) et respecter la spécification du document fournie et les contraintes imposer tout en cherchant l'optimisation.

BIBLIOGRAPHIE

- **TAD SkipList**
  - Operateur constructeur
    - SkipList skiplist_create(int nblevels);
    - SkipList skiplist_insert(SkipList d, int value)
  - Operateur de modifications
    - SkipList skiplist_remove(SkipList d, int value);
    - void skiplist_delete(SkipList d);
  - Operateur d'observation
    - unsigned int skiplist_size(SkipList d)
    - int skiplist_ith(SkipList d, unsigned int i);
    - bool skiplist_search(SkipList d, int value, unsigned int *nb_operations);
  - Operateur générique
    - void skiplist_map(SkipList d, ScanOperator f, void *user_data);
- **TAD SkipListIterator**
  - Operateur de l'iterateur
    - SkipListIterator skiplist_iterator_create(SkipList d, unsigned char w);
    - void skiplist_iterator_delete(SkipListIterator it);
    - SkipListIterator skiplist_iterator_begin(SkipListIterator it);
    - bool skiplist_iterator_end(SkipListIterator it);
    - SkipListIterator skiplist_iterator_next(SkipListIterator it);
    - int skiplist_iterator_value(SkipListIterator it);
- **Fonction Personnelle**
  - Fonction d'observation
    - void skiplist_dump(SkipList d);
    - void Skiplist_print_R_L (SkipList d);
    - void Skiplist_print_L_R (SkipList d);
    - bool iterate_on_skiplist_search(SkipList d, int value, unsigned int *nb_operation);
    - void itoa(int n, char ch[])
    - void skiplist_node_free(Node *x)
    - SkipList creationConst(char* chaine)

EXPLICATION DEMANDEE

3. En vous fondant sur les statistiques issues des questions 3.2.2 et 3.3.2, expliquez dans votre rapport de devoir les raisons de la différence en temps de calcul constatée, particulièrement pour le test numéro 4.

D'après les observations du comportement des deux fonctions sur les différents fichiers. Et en effet on remarque que pour les trois premier fichier la différence en temps d'exécution est moins perceptible. En revanche, le maximum, la moyenne d'opérations effectuées diffèrent de façon considérable. Pour la recherche sans littérateur est beaucoup plus rapide en temps et en opérations effectuées.

Alors Tout cella s'explique du fait est qu'avec l'littérateur on parcourt la Skip List à partir de son niveau 1 (c'est a dire à la manière d'une liste simplement chainer (on perd donc l'avantage d'utiliser les niveau). D'ailleurs dans le cas 2 on peut remarquer que les statistiques sont presque identiques car d'un côté on a un skip List de levelMax = 1 et l'littérateur parcourt nœud par nœud à l'aide d'un début, fin et suivant (on à plus cette possibilité de sauter un élément).

Alors que pour la recherche avec utilisation des niveaux est beaucoup plus rapide est moins couteux en opérations. On Explique Cela par la souplesse des recherches qui nous donne la possibilité de sauter des éléments grâce la recherche en escalier, plus on descend plus on se rapproche de l'élément recherché.

Je dirais donc pour conclure que la différence ce fait dans le parcours l'littérateur parcours (i-1) élément

DESCRIPTION ET SPECIFICATION DES FONCTION PERSONNELLE

Ces fonction essaie de respecter tant bien que mal les règles imposées dans le .h et dans le sujet

`void skiplist_dump(SkipList d);` → cette fonction affiche les différentes valeurs pointer par les nœuds de la skip List, elle m'a beaucoup aidée à débuguer, car de cette manière, je pouvais une représentation visuelle de la skip Liste (bien sur je n'affiche que les next).
Définie si et seulement si la skip List n'est pas vide

`Skiplist_print_L_R(SkipList d);` → Tout comme son homologue, elle permet d'afficher la Skip List .
Définie si et seulement si la skip List n'est pas vide

`bool iterate_on_skiplist_search(SkipList d, int value, unsigned int *nb_operation);` →cette fonction est comme la fonction skipList_search elle recherche un élément, renvoie vrai si trouvé faut si non et en même temps compte le nombre d'opérations effectuées . (bien sur j'ai mis son prototype dans le .h pour pouvoir l'utiliser coté client).

`void itoa(int n, char ch[])` → cette fonction se trouve dans « skipListTest.c », elle prends un entier et le convertie en chaine de caractère sans modification de l'entier. (C'est itoa car cette fonction n'est pas fournie dans la bibliothèque standard du c).

`void skiplist_node_free(Node *x)` → cette fonction prends pointeur de de cellule alloué et le libère correctement et met à null les adresses libérées pour bien signifier leur bonne libération .

`SkipList creationConst(char* chaine)` → cette fonction se trouve dans « skipListTest.c », elle permet de générer une skip List à partir d'un fichier texte données. Bien sur le fichier texte doit être organiser ainsi : le première entier représente le niveau de la skip List et le deuxième la taille de la skip liste et les autre sont les valeur à insérer. Je l ai créée pour factoriser car on faisait les mêmes opérations dans les 4 testes.

REMARQUES

En faisant ce devoir dans la partie Iittérateur, j'ai une autre idée qui aurais pu optimiser l'Ittérateur :
L'idée serait d'avoir une fonction changer de sens qui permet de choisir le sens de parcoure ce qui nous perpètrerait de faire le teste qu'une fois au début du parcours. Ce qui nous éviterait de tester chaque fois qu'on demande la next quel chemin on prend, car on l'a déterminé. Sur un petit nombre on ne verra pas trop la différence, mais sur une échelle de 100000 la différence serait nettement visible en temps car on aurait beaucoup de test en moins. Je ne l'ai pas implémenté, car ce n'était pas demandé et puis en le faisant, je ne respectais pas la spécification. Mais cella n'aurait pas était compliquer à faire.

RÉSUMÉ

Tout au long de ce devoir, j'ai pu découvrir beaucoup de choses, notamment la notion de skip List qui m'était tout à fait inconnue jusqu'à lors. Mais aussi cela m'a permis d'apprendre à utiliser valgrind un outil puissant et très utile non seulement pour découvrir les fuites mémoires, mais aussi permet de déboguer notre programme et le rendre propre. Ce devoir m'a permis de mesurer mon niveau en C, la faculté de suivre un algorithme et de le faire fonctionner, ou encore de cibler les documentations en fonction des besoins.

Je pense que l'objectif de ce devoir est rempli.