Examen Semestriel – Structures de Fichiers (SFSD) – 2CPI – ESI 2018/2019
Documents interdits – Durée 2h – Barème ([2+2+2] + [1+1+(2+2)] + [2+4+2])

Exercice 1 (Structures séquentielles)

a) Quels sont les avantages et inconvénients des fichiers séquentiels ordonnés, formés par des blocs contigus (dans le cas du format fixe et du format variable) ?

b) Comment peut-on séparer les enregistrements de taille variables, lorsque la longueur n'est pas limitée ?

c) Quels sont généralement les métriques (ou paramètres) permettant d'évaluer les performances d'une structure de fichier donnée ?

Exercice 2 (Hachage)

On voudrait gérer un fichier de données formé par N blocs avec une fonction de hachage 'h' statique utilisant l'algorithme de l'essai linéaire comme méthode de résolution de collisions. Les enregistrements sont de simples entiers. Avec cette méthode, deux types de suppressions sont possibles : (i) physique, entraînant éventuellement le parcours de plusieurs blocs en plus de celui contenant la donnée à supprimer et (ii) logique, entraînant une augmentation de l'encombrement mémoire (sur disque).

Afin de limiter les inconvénients des deux types de suppressions, on décide de combiner les deux approches comme suit : Lors de la suppression d'une donnée x, s'il y a risque de parcourir plusieurs blocs, on réalise alors à la place, une suppression logique.

Lors d'une insertion, on essaye de réutiliser d'abord les emplacements des données effacées logiquement et situés sur le chemin parcouru par la recherche.

a) Comment est représentée la case vide ? Comment être sur qu'il y a toujours au moins une case vide dans le fichier ?

b) Donnez les caractéristiques du fichier ainsi que la déclaration d'un buffer en spécifiant comment représenter les données effacées logiquement ?

c) Donnez l'algorithme permettant de supprimer une donnée x et un autre algorithme pour insérer une donnée x le plus proche possible de son adresse primaire.

On peut utiliser le module ci-dessous, de recherche d'une donnée x, sans l'écrire :

Rech(entrée x : entier ; sorties : trouv:bool, i,j:entiers). Les résultats de la recherche sont :

- le booléen *trouv* indiquant si *x* existe ou non dans le fichier ;
- *i* et *j* représentent l'adresse (numéro de bloc et numéro d'enregistrement dans le bloc) de la donnée *x*, si elle existe ou alors, celle du premier emplacement disponible pour l'insérer (si *x* n'existe pas dans le fichier)

Exercice 3 (Résolution d'entité)

Soit F un fichier TOF d'enregistrements d'un certain type 'Tenreg'. Ce fichier est formé de N blocs physiques.

Il est demandé de construire à partir du fichier F, un nouveau fichier S contenant toutes les correspondances pouvant exister entre les enregistrements de F.

La correspondance entre un couple d'enregistrements (e1,e2) du fichier F peut être vérifiée à l'aide d'une fonction booléenne donnée ***Match(e1,e2)***, retournant VRAI s'il existe une correspondance entre les enregistrements e1 et e2 et FAUX sinon.

Chaque enregistrement du fichier résultat S est composé de la concaténation de deux enregistrements (e1 et e2) de F.

La relation de correspondance est réflexive (c-a-d si Match(e1,e2) est vrai alors Match(e2,e1) est aussi vrai). Donc pour éviter les redondances (les répétitions) dans le fichier S et diminuer ainsi le coût en lecture, on évitera de rajouter la concaténation <e2,e1> si sa symétrique <e1,e2> est déjà présente dans S.

Nous disposons en mémoire centrale de M buffers d'entrée (sous forme d'un tableau de buffers) 'bf[1..M]' pour lire les blocs de F (avec M << N) et d'un buffer de sortie 'bs' pour écrire dans le fichier S.

a) Quelle est l'approche la plus efficace (entre 'boucles-imbriquées', 'tri-fusion' et 'hachage') pour résoudre ce problème.

b) Donnez un algorithme efficace, de type 'boucles-imbriques', pour construire le fichier S à partir du fichier F. Pour cela nous supposerons que les fonctions suivantes sont disponibles et peuvent donc être utilisées dans votre algorithme :

- ***Match(e1, e2)*** vérifie s'il existe une correspondance entre les enregistrements e1 et e2 de F
- ***Concat(e1, e2)*** retourne un enregistrement formé par la concaténation de e1 et e2

c) Donnez le coût de votre algorithme, en opérations de lecture. En déduire sa complexité.