TD2 – Structures de fichiers séquentielles

Objectifs : Ecriture d'algorithmes et programmation en langage C (à l'aide du modèle vu en cours) d'opérations sur les structures de fichiers séquentiels

1) Questions de cours

- Quels sont les avantages et inconvénients des fichiers TOF ?
- Qu'est-ce que le facteur de chargement d'un fichier ?
- Comment calculer le coût d'un algorithme sur une structure de fichiers ?
- Quels sont les avantages et inconvénients des zones de débordement pour un fichier TOF ?
- Quels sont les avantages et inconvénients des fichiers TOV{C/$\overline{C}$} ?
- Comment peut-on séparer les enregistrements de taille variables, lorsque la longueur n'est pas limitée ?
- Quels sont généralement les métriques (ou paramètres) permettant d'évaluer les performances d'une structure de fichier donnée ?

2) Les opérations d'accès de base

Considérer les douze cas de la figure présentée en cours donnant les différentes méthodes d'accès. Pour chaque cas, donnez les algorithmes et développer les programmes C pour :

- rechercher un enregistrement de clé donnée
- insérer un nouvel enregistrement
- supprimer un enregistrement de clé donnée
- rechercher tous les enregistrements dont la clé appartient à un intervalle donné

Dans le cas où le fichier est un tableau ordonné, développer en plus l'opération de chargement initial qui consiste à remplir le fichier à raison de μ% par bloc par des enregistrements qui sont lus en ordre croissant selon leur clé.

3) Compactage d'un fichier

Soit F un fichier TOF (vu comme tableau, ordonné et les enregistrements sont à taille fixe). La capacité maximale d'un bloc est fixée à b enregistrements. On désir **réduire la taille du fichier** en récupérant les espaces occupés par les enregistrements supprimés logiquement ainsi que les vides pouvant exister dans chaque blocs. Donc après cette opération, le fichier ne contiendra plus d'enregistrement supprimé logiquement et tous les blocs à part peut être le dernier seront remplis à 100% (le fichier restera ordonné).
Les caractéristiques du fichiers sont :

- Le nombre de blocs utilisés (Nblc)
- Le nombre d'enregistrements insérés (nbIns)
- Le nombre d'enregistrements supprimés logiquement (nbSup)

a) Donnez un algorithme qui **réalise cette opération en une seule passe**, sur le même fichier (sans créer un nouveau fichier ou de nouveau blocs) et en utilisant en mémoire centrale pas plus de 2 buffers.
b) Donnez le coût de cette opération en pire cas et en moyenne.

4) Tableaux volumineux en MS

Dans ce qui suit, on représente les tableaux d'entiers par des fichiers TōF, car ils ne peuvent pas être chargés entièrement en mémoire centrale.
Les éléments d'un tableau de taille n, sont stockés séquentiellement dans le fichier. La capacité des blocs est de b entiers. Tous les blocs sont donc forcément remplis à 100% sauf le dernier.
a) Quels sont les caractéristiques d'un tel fichier et donnez sa déclaration.
b) Donnez un algorithme efficace qui trouve la valeur moyenne des éléments d'un tableau.
c) Donnez un algorithme efficace qui réalise la multiplication de deux tableaux comme suit :

$(a_1, a_2, \dots a_n) \times (b_1, b_2, \dots b_n) = (a_1*b_1, a_2*b_2, \dots a_n*b_n)$

5) Matrices volumineuses en MS

Dans ce qui suit, on représente des matrices d'entiers (tableaux à deux dimensions) par des fichiers TŌF, car elles ne peuvent pas être chargées entièrement en mémoire centrale.

Les éléments d'une matrice de n lignes et m colonnes, sont stockés séquentiellement, ligne par ligne dans le fichier (c'est à dire que les éléments de la ligne i sont stockés avant ceux de la ligne i+1 et pour une même ligne, l'élément de colonne j précède celui de la colonne j+1) . La capacité des blocs est de b entiers. Tous les blocs sont donc forcément remplis à 100% sauf le dernier.

a) Quels sont les caractéristiques d'un tel fichier et donnez sa déclaration.

b) Donnez un algorithme efficace qui trouve l'adresse (numéro de bloc et numéro d'enregistrement) de l'élément m_{ij} (qui se trouve à la ligne i et colonne j) d'une matrice M.

c) Donnez un algorithme efficace qui transforme une matrice M1(n,m) représentée ligne par ligne en une nouvelle matrice M2(n,m) représentée colonne par colonne (c'est à dire que les éléments de la colonne j sont stockés avant ceux de la colonne j+1 et pour une même colonne, l'élément de la ligne i précède celui de la ligne i+1).

6) Soient F un fichier TŌF ayant pour seule caractéristique, le numéro du dernier bloc et BUF l'unique buffer disponible en mémoire centrale pour manipuler ce fichier.

- Donnez la déclaration de cette structure de fichiers (avec le modèle vu en cours) ainsi que l'algorithme permettant de supprimer physiquement l'enregistrement se trouvant à la position j du bloc numéro i en le remplaçant par le dernier enregistrement du fichier : *Supprimer(F, i, j)*
- Quel est le coût de cette opération de suppresion dans le cas favorable et dans le cas défavorable ?

7) Soient F un fichier TŌF d'entiers formé par N blocs et VP une valeur entière donnée (appelée 'pivot'). On aimerait réorganiser le contenu de F sur place (c-a-d sans rajouter de nouveaux blocs), afin que toutes les valeurs ≤ VP soient au début du fichier et toutes les valeurs > VP soient en fin de fichier. Il s'agit donc de répartir les valeurs de F entre les parties gauche et droite du fichier selon le pivot VP.

Exemple :

Le fichier avant la réorganisation:

bloc0	bloc1	bloc2	bloc3	bloc4	bloc5
[23 37 19 38 10]	[49 9 9 20 39]	[38 21 2 23 28]	[11 49 48 36 33]	[13 41 20 31 9]	[6]

Le fichier après réorganisation avec un pivot = 15:

bloc0	bloc1	bloc2	bloc3	bloc4	bloc5
[10 6 9 13 11]	[9 9 2 20 39]	[49 38 21 23 28]	[23 49 48 36 33]	[38 19 41 20 31]	[37]

|---------- partie gauche ---------||---------- partie droite ----------|

(≤ 15) (> 15)

Dans le ficher résultat, l'ordre des valeurs dans chacune des deux parties (gauche et droite) n'a pas d'importance.

a) Donnez la déclaration du fichier ainsi que les deux buffers associés.

b) Donnez un algorithme réalisant une telle opération de réorganisation utilisant seulement 2 buffers en mémoire centrale et ayant un coût (en entrées/sorties) ne dépassant pas N lectures et N écritures.

8) Donner l'algorithme de recherche dichotomique pour un fichier de type TOVC̄ dont l'unique caractéristique est : le numéro du dernier bloc.

La structure des blocs est comme suit :

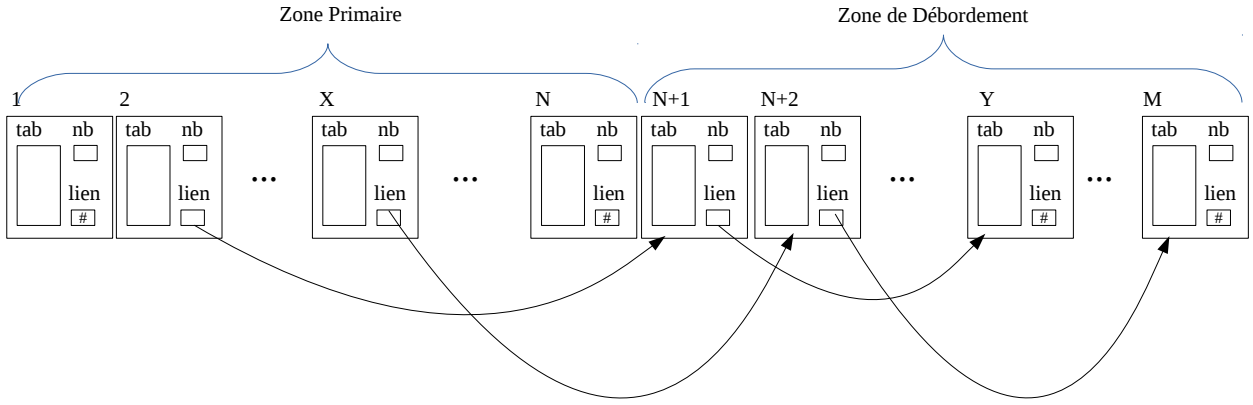
```
Tbloc = structure
    tab : tableau[b] de caractères          // contenant les enregistrements
    pos_libre : entier                      // indice de la dernière position libre dans tab
    clé1, clé2 : tableau[10] de caractères  // la plus petite et la plus grande des clés dans tab
Fin
```

On supposera que chaque enregistrement est stocké sous forme de chaîne de caractères commençant par sa taille (sur 3 caractères), un caractère d'effacement logique, puis sa clé de taille fixe (sur 10 caractères). Le reste de caractères représentent les champs restants de l'enregistrement.

9) On définit une structure de fichier ordonné avec gestion d'une zone de débordement. Les enregistrements sont à taille fixe et munis d'une clé sur un type totalement ordonné.
Le fichier, appelé F, est composé de deux parties :

- Une zone primaire ordonnée et formée de N blocs contigus. Cette zone est gérée comme une structure TOF particulière. Sa taille (N) ne peut changer que durant des opérations de réorganisation.
- Une zone de débordement, ordonnée par parties, qui commence à partir du bloc N+1 et s'étend au fur et à mesure que les enregistrements en débordement augmentent. Cette zone est gérée comme un ensemble de structures LOF. Pour chaque bloc X qui déborde dans la zone primaire, est associée une liste en zone de débordement de tête le lien du bloc X. Cette liste de blocs renferme les enregistrements qui n'ont pas pu être insérés dans le bloc X, car devenant plein.

Les blocs peuvent contenir au maximum, b enregistrements.



Lors du chargement initial, on remplit la zone primaire, avec un taux de remplissage fixé au départ, avec des enregistrements donnés en ordre croissant. La zone de débordement sera initialement vide. Le dernier enregistrement de la zone primaire représente une donnée fictive avec comme clé, la valeur infinie (la plus grande valeur du type de la clé).
La localisation d'un enregistrement de clé c, commence par une recherche dichotomique dans la zone primaire et se poursuit, éventuellement, en zone de débordement, séquentiellement sur une des listes de cette zone.
L'insertion d'un enregistrement dans le bloc i (en zone primaire) à la position j, provoque naturellement des décalages intra-bloc. Pour éviter par contre les décalages inter-blocs, dans le cas où le bloc primaire (le bloc i) est déjà plein, on déplace un des enregistrements du bloc i et on l'insère en zone de débordement dans la liste pointée par le champ lien du bloc i si elle existe déjà, sinon elle sera créée et le champ lien mis à jour en conséquence. Pour réaliser cette opération de déplacement d'un enregistrement, lors d'une insertion dans un bloc plein, on effectue les décalages intra-bloc entre les positions j et b-1 uniquement, suivie par l'expulsion de l'avant dernier enregistrement du bloc (position b-1) et son insertion en zone de débordement. De cette manière, le dernier enregistrement de chaque bloc primaire ne pourra jamais être déplacé en zone de débordement. Cette manière de procéder, nous garantit de pouvoir effectuer des recherches dichotomiques sur la zone primaire pour localiser rapidement un enregistrement de clé donnée.
L'insertion d'un enregistrement dans la zone de débordement suit le schéma général des insertions dans des structures de type LOF. Une fois le bloc de la liste localisé (séquentiellement) l'enregistrement y est inséré. Si le bloc est déjà plein, celui-ci éclate en allouant un nouveau bloc à la fin de la zone de débordement (M+1) et les enregistrements du bloc ayant subit l'éclatement, seront répartis équitablement (moitié/moitié) entre les deux blocs. Le chaînage est bien sûr mis à jour pour garder la structure de liste.
Les suppressions sont effectuées logiquement.
Les caractéristiques du fichier F sont :

N : le numéro du dernier bloc de la zone primaire
M : le numéro du dernier bloc de la zone de débordement
nbI : nombre d'enregistrements insérés

a) Donnez un algorithme efficace 'Localiser(c,i,j)' pour localiser un enregistrement de clé c donnée dans la zone primaire de ce fichier. L'algorithme devrait retourner (dans i) le numéro du bloc primaire ainsi que la position (dans j) où devrait se trouver l'enregistrement (en zone primaire seulement).
Donnez la complexité de votre algorithme.
b) Donnez un algorithme efficace 'Lister(a,b)' pour effectuer une requête à intervalle.
L'algorithme devrait afficher tous les enregistrements dont la clé ∈ [a,b].
Quelle est l'influence de la taille des listes de débordement sur les performance de cette opération ?
c) Donnez l'algorithme 'Reorganiser(taux, nouvNom)' permettant de réorganiser le fichier. Il s'agira de construire un nouveau fichier de nom 'nouvNom' à l'aide des enregistrements non supprimés logiquement de l'ancien fichier. Le paramètre 'taux' indique le chargement initial voulu pour le nouveau fichier.

10-a) Que fait l'algorithme ci-dessous et quelle est la structure du fichier utilisée ?

Remarque importante : Les blocs i et i+1 sont pleins et le bloc i+2 n'est pas plein

```
// Un fichier est défini comme suit :
F un FICHIER de TBloc BUFFER Buf, Buf2 ENTETE (entier) ;
// L'entête(F,1) désigne le nombre de blocs dans le fichier
Type TBLOC = Structure
    nb : entier ;          //Nombre d'articles dans bloc
    Tab : tableau(1..B) caractère ;          // Tableau contenant au maximum B articles de chaines de caractères
Fin
…
// Dans Buf on a déjà lu le bloc i+2
// Quotient(a,b) :entier  c'est une fonction qui calcule le quotient de a sur b et retourne un entier
// Reste(a,b) : entier  c'est une fonction qui calcule le reste de a sur b et retourne un entier
Q := QUOTIENT(2*B+Buf.nb , 3);
R := RESTE(2*B+Buf.nb , 3);
// Mise à jour bloc I+2
// Le bloc I+2 contiendra Q+R articles
// Décalage dans le bloc I+ 2
j := 0;
    Pour k := Buf.nb   jusquà   1   pas   -1
        Buf.Tab[Q+R-j] := Buf.Tab[k];
        j ++;
    FinPour
// Prendre ((B-Q)*2) articles du bloc I+1
LIREDIR (F , Buf2 , I+1);
j := 1 ;
    Pour   k :=(2*Q - B + 1)   jusquà   B
        Buf.Tab[j] := Buf2.Tab[k] ;
        j ++ ;
    FinPour
Buf.nb := Q+R;
ECRIREDIR (F , Buf , I+2);
LIREDIR (F , Buf , I);

// Mise à jour bloc I+1
// Décalage dans le bloc I+ 1
j := 0 ;
    Pour   k := 2*Q – B   jusquà   1   pas   -1
        Buf2.Tab[Q-j] := Buf2.Tab[k];
        j ++ ;
    FinPour
```

```
// Prendre (B-Q) articles du bloc I
j := 1 ;
   Pour   k := Q+1 jusquà   B
      Buf2.Tab[j] := Buf.Tab[k];
      j ++ ;
   FinPour
Buf2.nb := Q;
ECRIREDIR (F , Buf2 , I+1);
// Mise à jour bloc I
Buf.nb := Q;
ECRIREDIR (F , Buf , I);
…
```

10-b) Soit l'exemple suivant représentant 3 blocs consécutifs I, I+1 et I+2 d'un fichier donné.

En exécutant le programme ci-dessus, représentez les contenus de chaque bloc I, I+1 et I+2 tels que B=8, les blocs I et I+1 sont pleins et le bloc I+2 n'est pas plein.

Avant exécution :

	8							
Bloc I	a	b	c	d	e	f	g	h
	8							
Bloc I+1	i	j	k	k	m	n	o	p
	3							
Bloc I+2	q	r	s					

Après exécution : ???

Bloc I								
Bloc I+1								
Bloc I+2								

11) Nous souhaitons implémenter efficacement un modèle de file d'attente (FIFO) en MS en utilisant des fichiers sous forme de **listes de blocs**. Les éléments de la file sont des enregistrements d'un type 'T_enreg' donné ayant un format fixe.

Nous disposons de deux buffers en MC.
- Donnez une solution efficace pour l'implémentation du modèle, permettant la réutilisation des blocs libérés.
Rappel : Le modèle des files d'attente FIFO est constitué des opérations suivantes :
{ *CreerFile(F)*:initialise F , *FileVide(F)*:teste si F est vide , *Enfiler(F, e)*:rajoute e en queue de file , *Defiler(F, e)*:supprime et récupère dans e l'élément en tête de file }

12) Nous souhaitons implémenter efficacement un modèle de file d'attente (FIFO) en MS en utilisant des fichiers sous forme de *blocs contigus gérés de manière circulaire*. Les éléments de la file sont des enregistrements d'un type 'T_enreg' donné ayant un format fixe.

Nous disposons de deux buffers en MC.
- Donnez une solution efficace pour l'implémentation du modèle.

Le modèle des files d'attente FIFO que l'on veut implémenter est constitué des opérations suivantes :
CreerFile(F, N): initialise F avec un fichier de N blocs,
NbElement(F): retourne le nombre d'éléments présents dans la file,
EnfilerGroupe(F, n, T): rajoute les n éléments du tableau T en queue de file,
DefilerGroupe(F, n, T): supprime et récupère dans T les n premiers éléments en tête de file

13) Soit F un fichier de type T$\overline{O}$VC (fichier vu comme tableau, non ordonné, enregistrements à tailles variables avec chevauchement inter-blocs). A l'intérieur d'un bloc, les enregistrements sont préfixés par un indicateur d'effacement logique (sur un caractère) et par leur longueur (sur cinq caractères).
On aimerait compacter le fichier F en supprimant physiquement les enregistrements effacés logiquement. Donnez un algorithme pour réaliser ce traitement en minimisant le nombre de lectures physiques, sachant que l'on dispose de deux buffers (buf1 et buf2) en mémoire centrale.
Considérations à suivre :

- le type d'un bloc est : Tableau[b] caractères
- un enregistrement peut être à cheval sur plusieurs blocs
- le bloc d'en-tête contient les caractéristiques suivantes :
  - numéro du dernier bloc,
  - 1ere position libre dans le dernier bloc,
  - nombre total d'enregistrements insérés et
  - nombre total d'enregistrements effacés logiquement.

14) Fragmentation
Soient F un fichier vu comme tableau non ordonné, avec format fixe; C1 et C2 deux clés données.
On désire fragmenter F en 3 fichiers (F1, F2 et F3) de la manière suivante :
F1 doit contenir tous les enregistrements de F dont la clé est < C1,
F2 doit contenir tous les enregistrements de F dont la clé est ≥ C1 et < C2,
F3 doit contenir tous les enregistrements de F dont la clé est ≥ C2.
Donner un module qui réalise cette fragmentation en utilisant 4 buffers en MC (buf, buf1, buf2 et buf3). L'opération doit se dérouler en une seule passe (un seul parcours de F).