Corrigé Examen Semestriel – Structures de Fichiers (SFSD) – 2CPI – ESI 2017/2018

Exercice 1

a) Qu'est-ce que le facteur de chargement d'un fichier ?

Exemple de réponses :
- Le remplissage moyen des blocs d'un fichiers
- Le pourcentage de remplissage du fichiers
- Nombre_de_données_insérées / Nombre_de_places_disponibles
- Nombre_de_données_insérées / (Nombre_de_blocs_utilisés * taille_bloc)

b) Soient F un fichier TOF 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)*

```
Tbloc = structure
        tab : Tableau[ b ] de Tenreg ;
        NB : entier
Fin

F : Fichier de Tbloc Buffer BUF Entete ( entier ) ;
// Entete(F,1) : représente le numéro du dernier bloc

Sup( F, i, j )
Debut
    N ← Entete(F, 1) ;
    SI ( i == N )
        LireDir( F, i, BUF ) ;
        BUF.Tab[ j ] ← BUF.Tab[ BUF.NB ] ;
        BUF.NB-- ;
        // si le dernier bloc devient vide, ce n'est pas la peine de l'écrire ...
        SI ( BUF.NB == 0 )   Aff_Entete( F, 1, N-1 )  // juste décrémenter l'entete(F,1)
        SINON    EcrireDir( F, N, BUF ) ;
        FSI
    SINON // (donc i != N)
        LireDir( F, N, BUF ) ;
        e ← BUF.tab[ BUF.NB ] ;
        BUF.NB-- ;
        // si le dernier bloc devient vide, ce n'est pas la peine de l'écrire ...
        SI ( BUF.NB == 0 )   Aff_Entete( F, 1, N-1 )  // juste décrémenter l'entete(F,1)
        SINON    EcrireDir( F, N, BUF ) ;
        FSI ;
        LireDir( F, i, BUF ) ;
        BUF.tab[ j ] ← e ;
        EcrireDir( F, N, BUF ) ;
    FSI
Fin  // Sup
```

c) Quel est le coût de cette opération dans le cas favorable et dans le cas défavorable ?

Dans le cas le plus favorable l'opération coûte un seul accès (1 lecture de bloc)
Dans le cas le défavorable on aura 2 lectures et 2 écritures (4 accès)

Exercice 2

Soit F un fichier d'enregistrements organisé en B-arbre d'ordre 21 et de hauteur 4. Tous les nœuds sont remplis à leur strict minimum. Soient A un nœud se trouvant au niveau 2 et B son suivant sur le même niveau (pour rappel la racine est au niveau 0).
a) Donnez la structure d'un bloc pour un tel fichier.

Tbloc = structure
    Val : Tableau[20] de Tenreg ;
    Fils : Tableau[21] d'entiers ;
    Degré : entier
Fin ;

b) Si le bloc A devient indisponible (à cause d'une panne par exemple), combien d'enregistrements du fichier F seront-ils perdus ?

Il y a 5 niveaux (de 0 à 4) et chaque bloc (autre que la racine) contient 10 enregistrements (50% de 20).
Le sous-arbre A est donc formé de : 1 (au niveau 2) + 11 (au niveau 3) + 11*11 (au niveau 4) = 133 blocs
Donc le nombre d'enregistrements perdus = 133 * 10 = 1330 enregistrements.

c) Combien d'enregistrements existent dans F ayant des valeurs de clés supérieures à la plus grande clé de A et inférieures à la plus petite clé de B ?

Le nombre d'enregistrements séparant le bloc A du bloc B est calculé comme suit :
le nb d'enreg du sous-arbre le plus à droite de A + le séparateur entre A et B (un seul enreg) dans un des nœuds ascendant + le nb d'enreg du sous-arbre le plus à gauche de B.
Le sous-arbre droit de A est formé de 1 + 11 blocs (12 blocs) contenant en tout 120 enregistrements
Le sous-arbre gauche de B est formé de 1 + 11 blocs (12 blocs) contenant en tout 120 enregistrements
Le nombre d'enregistrements séparant le bloc A du bloc B = 120 + 1 + 120 = 241 enregistrements.

Exercice 3

On voudrait gérer une zone de débordement commune à tous les blocs de la zone principale dans la méthode du hachage linéaire (dynamique). Cette zone de débordement commune est un fichier de type TOF où les enregistrements provenant d'un même bloc de la zone principale sont chaînés entre eux.
a) Donnez la déclaration du fichier de la zone de débordement.

Soit Fd le fichier utilisé pour la zone de débordement :
Fd : Fichier de TblocD Buffer bufD Entete(entier , entier , entier)
// entete(F,1) : dernier bloc
// entete(F,2) et entete(F,3) représente la tête de la liste des enregistrements vide dans la zone de débordement (numBloc, Deplacement)

TblocD = structure
    tab : Tableau[b'] de <e :Tenreg, lien_numB:entier, lien_depl:entier> ;
    NB : entier
Fin

b) Donnez l'algorithme de recherche d'un enregistrement de clé x.

La zone principale est composée d'un fichier F défini comme suit :

F : Fichier de TblocP Buffer buf Entete(entier , entier , entier)
// entete(F,1) : Nombre d'enregistrements insérés (pour vérifier la condition d'éclatement)
// entete(F,2) et entete(F,3) représente les paramètres du fichier (i et p)

```
TblocP = structure
        tab : Tableau[ b ] de Tenreg ;
        NB : entier ;
        lien_numB:entier ;   // pour les débordements
        lien_depl:entier       // pour les débordements
Fin

Rech( x:clé ) : booléen
Début
    i ← Entete(F,2) ;    // le niveau du fichier
    p ←Entete(F,3) ;    // le prochain bloc à éclater
    a ← h( i, x ) ;
    SI ( a < p )  a ← h( i+1, x )  FSI ;
    // recherche dans le bloc principal ...
    LireDir( F, a, buf ) ;
    j ← 1 ;   stop ← faux ;
    TQ ( j ≤ buf.NB  && non stop )
          SI ( buf.tab[ j ].clé == x ) stop ← vrai SINON j++  FSI
    FTQ ;
    SI ( non stop && buf.lien_numB != -1 )
          // recherche dans la zone de débordement …
          i ←  buf.lien_numB ;    j ←  buf.lien_depl ;    i' ← -1 ;
          TQ ( non stop && i != -1 )
                SI ( i != i' )   LireDir( Fd, i, bufD )  FSI ;
                SI ( bufD.tab[ j ].e.clé == x )
                   stop ← vrai
                SINON
                   i' ← i ;    i ← bufD.tab[ j ].lien_numB ;   j ← bufD.tab[ j ].lien_depl ;
                FSI
           FTQ
    FSI ;
    retourner stop
Fin // Rech
```

c) Indiquez comment gérer les vides qui pourraient apparaître dans la zone de débordement lors des éclatements et fusion de blocs de la zone principale.

A chaque fois qu'on supprime un enregistrement dans la zone de débordement, on l'insère en tête de liste des enregistrements supprimés

Exercice 4

Soient F un fichier TOF 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.

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

F:fichier de Tbloc buffer buf1, buf2 entete(entier) // la seule caractéristique représente le dernier bloc

```
Tbloc = struct
        tab:tableau[b] d'entiers ;
        NB:entier
fin
```

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.

```
// *** Partitionnement sur place du buffer buf1 ***
// Toutes les valeurs ≤ vp seront placées au début du buffer et toutes les valeurs > vp à la fin du buffer
// retourne l'indice du 1er élément > vp (s'il existe) sinon buf1.NB+1 si toutes les valeurs sont ≤ vp

trait1( vp ) : entier
Début
  j ← 1;
  i ← 1;
  TQ ( i ≤ buf1.NB )
       SI ( buf1.tab[i] ≤ vp )
          tmp ← buf1.tab[j];
          buf1.tab[j] ← buf1.tab[i];
          buf1.tab[i] ← tmp;
          j++;
       FSI ;
       i++;
    FTQ ;
  return j;
Fin // trait1

// *** Partitionnement sur place du buffer buf2 ***
// Toutes les valeurs ≤ vp seront placées au début du buffer et toutes les valeurs > vp à la fin du buffer
// retourne l'indice du dernier élément ≤ vp (s'il existe) sinon 0 si toutes les valeurs sont > vp

trait2( vp ) : entier
Début
    i ← buf2.NB ;
    j ← buf2.NB ;
    TQ ( i ≥ 1 )
       SI ( buf2.tab[i] > vp )
          tmp ← buf2.tab[j];
          buf2.tab[j] ← buf2.tab[i];
          buf2.tab[i] ← tmp;
          j--;
       FSI ;
       i--;
    FTQ ;
    return j;
Fin // trait2
```

```
// *** Partitionnement sur place d'un fichier ***
// En fonction d'une valeur pivot vp

partition( f )
Début
   dern ← Entete(f, 1);
   SI ( dern == 1 )
      lireDir( f, 1, buf1 );
      j1 ← trait1( vp );
      ecrireDir( f, 1, buf1 );
   SINON
      i1 ← 1;   i2 ← dern;
      lireDir( f, i1, buf1 );
      lireDir( f, i2, buf2 );
      j1 ← trait1(vp);
      j2 ← trait2(vp);
      stop ← faux;  chg1 ← faux;   chg2 ← faux;
      TQ  ( !stop )
          SI ( j1 > buf1.NB )
             SI ( chg1 )    ecrireDir( f, i1, buf1 ) ;  chg1 ← faux  FSI ;
             i1++;
             SI ( i1 < i2 )   lireDir( f, i1, buf1 );   j1 ← trait1(vp);
             SINON          stop ← vrai;
             FSI
          SINON
             SI ( j2 < 1 )
                SI ( chg2 )  ecrireDir( f, i2, buf2 );  chg2 ← faux  FSI ;
                i2--;
                SI ( i1 < i2 ) lireDir( f, i2, buf2 );  j2 ←  trait2(vp);
                SINON         stop ← vrai;
                FSI
             SINON
                tmp ← buf1.tab[j1];
                buf1.tab[j1] ← buf2.tab[j2];
                buf2.tab[j2] ← tmp;
                j1++;
                j2--;
                chg1 ← vrai ;
                chg2 ← vrai;
             FSI
          FSI
      FTQ //  !stop
      SI ( chg1 ) ecrireDir( f, i1, buf1 );
      SI ( chg2 ) ecrireDir( f, i2, buf2 )
Fin // partition


/************************************/
```