Algo2 – Corrigé EMD3 2007/2008

Partie 1 / Exercice 1: Intersection

a- On dispose de 3 buffers en mémoire centrale (buf1, buf2 et buf3), réaliser l'opération d'intersection de 2 fichiers (F1 et F2) ordonnés à format fixe (TOF), contenant le même type d'enregistrement.
L'opération d'intersection, produit comme résultat un fichier (F3) ordonnés (TOF) contenant tous les enregistrements qui existent, simultanément, dans les deux fichiers en entrée. Les deux fichiers en entrée sont parcourus parallèlement en une seule passe.

```
// déclaration des fichiers, la seule caractéristique utilisée est le num du dernier bloc
type
        Tbloc = structure
                tab : TABLEAU[1..b]de Tenreg;
                NB : entier
        Fin;

var
        F1 : FICHIER de Tbloc BUFFER buf1 entete( entier );
        F2 : FICHIER de Tbloc BUFFER buf2 entete( entier );
        F3 : FICHIER de Tbloc BUFFER buf3 entete( entier );

        i1, i2, i3 : entier         // num de blocs
        j1, j2, j3 : entier         // num d'enregistrements dans les blocs

DEBUT  // Intersection de 2 fichiers ordonnés

        // ouverture des fichiers
        ouvrir( F1 , « fich1.dat », « A »); ouvrir( F2, « fich2.dat », « A »);
        ouvrir(F3, « result.dat », « N »);
        M1 := entete(F1,1)  // num du dernier bloc de F1;
        M2 := entete(F2,1)  // num du dernier bloc de F2;
        i1 := 1; i2 := 1; i3 := 1;
        lireBloc(F1,1 buf1 );  lireBloc(F2,1, buf2 );
        j1 := 1; j2 := 1; j3 := 1;

        TQ (i1 <= M1  ET  i2 <= M2)          // tantque non fin des 2 fichiers
                SI (buf1.tab[j1].cle = buf2.tab[j2].cle)
                        buf3.tab[j3] := buf1.tab[j1];
                        j3 := j3 + 1;
                        SI (j3 > b)  buf3.NB := b; ecrireBloc(F3,i3,buf3); i3 := i3 + 1; j3:=1 FSI;
                        // avancer dans buf1 et dans buf2
                        j1 := j1 + 1;
                        SI (j1 > b)  i1 := i1 + 1;
                                        SI (i1 <= M1) lireBloc(F1,i1,buf1); j1 := 1 FSI;
                        FSI;
                        j2 := j2 + 1;
                        SI (j2 > b)  i2 := i2 + 1;
                                        SI (i1 <= M1  ET i2 <= M1) lireBloc(F2,i2,buf2); j2 := 1 FSI;
                        FSI;
```

```
            SINON       //  buf1.tab[j1].cle <> buf2.tab[j2].cle
                    // dans ce cas on avance uniquement dans l'un des 2 fichiers
                    SI  (buf1.tab[j1].cle < buf2.tab[j2].cle)
                            j1 := j1 + 1;
                            SI (j1 > b)  i1 := i1 + 1;
                                SI (i1 <= M1) lireBloc(F1,i1,buf1); j1 := 1 FSI;
                            FSI;
                    SINON  //  donc on a : buf1.tab[j1].cle > buf2.tab[j2].cle
                            j2 := j2 + 1;
                            SI (j2 > b)  i2 := i2 + 1;
                                SI (i2 <= M1) lireBloc(F2,i2,buf2); j2 := 1 FSI;
                            FSI;
                    FSI
            FSI // (buf1.tab[j1].cle = buf2.tab[j2].cle)
    FTQ;  // (i1 <= M1  ET  i2 <= M2)

    // dernière écriture dans F3
    buf3.NB := j3 – 1;
    ecrireBloc(F3, i3, buf3);

    // MAJ de la caractéristique (num du dernier bloc)
    Aff-entete( F3, 1, i3 );

    // Ferméture des fichiers
    Fermer(F1); Fermer(F2); Fermer(F3);

FIN // Intersection
```

dans cette convention, un fichier vide est caractérisé par l'utilisation d'un seul bloc (num 1) vide (NB=0)

b- Donner le coût de cette opération dans le meilleur et dans le pire des cas.
Le coût d'une opération est mesuré en terme de nombre d'accès (lireBloc/ecrireBloc) nécessaire à sa réalisation.

Posons M1 le nombre de bloc du fichier F1 et M2 le nombre de blocs du fichier F2.
Puisque l'algorithme s'arrête dès qu'un fichier se termine, on aura alors à parcourir que les M (M=min(M1,M2)) premiers blocs de F1 et de F2.

Dans le meilleur des cas : on doit parcourir F1 et F2 et produire un fichier F3 vide (aucune clé n'existe dans les 2 fichiers en même temps).

le nombre d'accès = M + M + 1 = **2M+1 accès**
(lecture de F1) + (lecture de F2) + (écriture d'un seul bloc vide pour F3)
ou bien
si on choisit la convention de n'occuper aucun bloc pour les fichiers vides, le résultat serait alors :
le nombre d'accès = M + M = **2M accès**

Dans le pire des cas : le fichier F3 occupera la même taille que le plus petit entre F1 et F2.
Cela arrive quand le résultat de l'intersection est maximal, c-a-d toutes les clés de F1 (ou alors de F2) existent dans F2 (ou alors dans F1).

Il faudra donc parcourir F1 et F2 (2M) et construire F3 (qui occupe aussi M blocs)
le nombre d'accès = M + M + M = **3M accès.**

Partie 2 / Exercice 2 : 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.

a- 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).

```
// déclaration des fichiers, la seule caractéristique utilisée est le num du dernier bloc
type
        Tbloc = structure
                tab : TABLEAU[1..b]de Tenreg;
                NB : entier
        Fin;

var
        F   : FICHIER de Tbloc BUFFER buf   entete( entier );
        F1 : FICHIER de Tbloc BUFFER buf1 entete( entier );
        F2 : FICHIER de Tbloc BUFFER buf2 entete( entier );
        F3 : FICHIER de Tbloc BUFFER buf3 entete( entier );

        i, i1, i2, i3 : entier      // num de blocs
        j, j1, j2, j3 : entier      // num d'enregistrements dans les blocs

DEBUT // Fragmentation
  ouvrir(F, «entree.dat», «A»);
  ouvrir(F1, «sortie1.dat», «N»); ouvrir(F2, «sortie2.dat», «N»); ouvrir(F3, «sortie3.dat», «N»);
  i := 1;    i1 := 1; i2 := 1; i3 := 1;   j1 := 1; j2 := 1; j3 := 1;
  lire(C1);  lire(C2);

  TQ  ( i <= entete(F,1) )
        lireBloc( F, i, buf );
        POUR j:=1,buf.NB
                SI (buf.tab[j].cle < C1)
                        buf1.tab[j1] := buf.tab[j];
                        j1 := j1 + 1;
                        SI (j1 > b)  buf1.NB := b; ecrireBloc(F1,i1,buf1); i1 := i1 + 1; j1 := 1 FSI
                FSI;
                SI (buf.tab[j].cle >= C1 ET buf.tab[j].cle < C2)
                        buf2.tab[j2] := buf.tab[j];
                        j2 := j2 + 1;
                        SI (j2 > b)  buf2.NB := b; ecrireBloc(F2,i2,buf2); i2 := i2 + 1; j2 := 1 FSI
                FSI;
                SI (buf.tab[j].cle >= C2)
                        buf3.tab[j3] := buf.tab[j];
                        j3 := j3 + 1;
                        SI (j3 > b)  buf3.NB := b; ecrireBloc(F3,i3,buf3); i3 := i3 + 1; j3 := 1 FSI
                FSI;
        FinPOUR;
        i := i + 1
  FTQ;  // i <= entete(F,1)
```

```
  // écriture des derniers buffers de F1 , F2 et F3
  buf1.NB := j1 – 1; ecrireBloc(F1,i1,buf1);
  buf2.NB := j2 – 1; ecrireBloc(F1,i2,buf2);
  buf3.NB := j3 – 1; ecrireBloc(F1,i3,buf3);
  // affecter les caractéristiques
  Aff-entete(F1,1,i1);
  Aff-entete(F2,1,i2);
  Aff-entete(F3,1,i3);
  // fermeture des fichiers
  Fermer(F);
  Fermer(1);  Fermer(F2);  Fermer(F3)

FIN // Fragmentation
```

b- Peut-on réaliser la même opération (avec le même nombre d'accès) en diminuant le nombre de buffers en MC ?
Non, car il faudrait alors rajouter des passes supplémentaire du fichier F.

Partie 3 / Exercice 3 : Accès par hachage

On manipule des enregistrements à taille fixe avec des clés de type entier.
Soit h(c) = (c mod N) + 1, une fonction de hachage pour accéder à l'enregistrement de clé c dans un fichier vu comme tableau, formé par des blocs ayant une capacité de b enregistrements.
En cas de collision sur le bloc de numéro h(c) (c-a-d que le bloc est déjà plein), l'enregistrement de clé c sera inséré en fin de fichier (à partir du bloc N+1). Cette partie du fichier, gérée séquentiellement (et formée par les blocs N+1, N+2, ...), est appelée zone de débordement.
Quand la zone de débordement deviendra trop importante, on effectuera une réorganisation en construisant un nouveau fichier avec une valeur de N plus grande.
a- Donner les caractéristiques du fichier.
On a besoin, au moins, de connaitre la valeur de **N** et le **numéro du dernier bloc** dans la zone de débordement. La déclaration du fichier pourra être alors :

```
type
        Tbloc = structure
                tab : TABLEAU[1..b]de Tenreg;
                NB : entier
        Fin;
var
        F   : FICHIER de Tbloc BUFFER buf   entete( entier, entier );
// entete(F,1) : désigne N (change à chaque réorganisation)
// entete(F,2) : désigne le dernier bloc en zone de débordement (initalement N+1)
```

b- Donner le module de recherche d'un enregistrement de clé c.

```
Recherche( c : entier; var trouv : Booleen; var e:Tenreg )
var
        i, j, N, DernierBloc : entier;

DEBUT // Recherche
  ouvrir( F, « donnees.dat », « A » );
  N := entete(F,1);
  DernierBloc := entete(F,2);
```

```
  // on commence la recherche par un accès direct au bloc h(c)
  i := h(c); // (c mod N)+1;
  lireBloc(F,i,buf);
  j := 1;
  trouv := FAUX;
  TQ ( Non trouv ET j <= buf.NB )
        SI (buf.tab[j].cle = c)  trouv := VRAI; e := buf.tab[j]
        SINON
              j := j + 1
        FSI
  FTQ;

  // si la clé c n'existe pas dans le bloc i et celui-ci est plein, il faudra continuer la recherche
  // dans la zone de débordement (de manière séquentielle)
  SI (Non trouv et buf.NB = b)
        i := N+1;
        TQ (Non trouv ET i <= DernierBloc)   // parcours séquentiel de la zone de débordement
              lireBloc(F,i,buf);
              j := 1;
              TQ ( Non trouv ET j <= buf.NB )
                    SI (buf.tab[j].cle = c)
                          trouv := VRAI; e := buf.tab[j]
                    SINON
                          j := j + 1
                    FSI;
              FTQ;
              SI (Non trouv)  i := i + 1 FSI
        FTQ // (Non trouv ET i <= DernierBloc)
  FSI; // Non trouv et buf.NB = b

  Fermer( F );

FIN // Recherche
```

Partie 3 / Exercice 4 : Réflexion sur les B-Arbres

Dans le cas d'un fichier organisé en B-Arbre et en l'absence de suppressions, expliquer comment pourrait-on limiter le chargement minimal d'un bloc (non racine) à 66% (c-a-d 2/3) ?

Dans la méthode standard des B-Arbres, le chargement minimal des blocs (autres que la racine) est de 50%, car à chaque éclatement d'un noeud plein (lors d'une insertion), on crée un nouveau noeud pour partager le contenu de l'ancien noeud en 2 moitiés (50% chacun).
Pour avoir un chargement minimal de 66% (2/3), il suffit (lors des insertions) de retarder l'éclatement d'un noeud plein en redistribuant les valeurs avec l'un de ses frères. Lorsque les 2 noeuds sont pleins (on ne peut plus faire de redistribution entre les 2 noeuds), on les éclate en 3 noeuds. c-a-d on crée un nouveau noeud et on partage le contenu des 2 anciens noeuds en 3 tiers (un tiers pour chaque noeud), ce qui donne un taux de remplissage minimal de 2/3.