Corrigé CF - SFSD / 2022-2023

<u>Question 1</u> :

a) Quelle est l'utilité du chargement initial d'un fichier ordonné vu comme tableau (format fixe ou variable) ? **(2 pts)**

Permet de commencer les traitements avec des performances optimales
en insérant les données initiales (qui doivent être déjà prêtes) de manière
à laisser un peu d'espace dans chaque bloc.
Cela évitera les problèmes des décalages inter-blocs durant les prochaines
insertions pendant un certain temps.
……
……
……

b) Donnez la complexité et le coût de la recherche séquentielle dans un fichier T~OVC de taille N blocs physiques et où chaque enregistrement peut occuper jusqu'à M blocs. **(2 pts)**

O(N) car un nombre de lectures physiques proportionnel à N, devront être effectués
pour rechercher n'importe quel enregistrement.
Dans le pire cas on aura N lectures
Dans le cas moyen on aura N/2 lectures
Dans le cas favorable une seule lecture
……
……
……

c) Quelle est l'utilité des 'buffers' ? : **(2 pts)**

Les buffers permettent de cacher d'une certaine manière, la différence
entre le temps d'accès à la MC et celui à la MS
Il permettent donc de réaliser des opérations sur les fichiers de manière plus efficace
De plus, pour certaines opérations sur les fichiers (tri, jointures, …)
plus le nombre de buffers disponibles est grand, plus les performances s'améliorent
……
……
……

d) Quels sont les avantages et inconvénients des index bitmaps ? : **(2 pts)**

Les index bitmaps permettent de rendre les accès multiclés plus rapides
lorsque la cardinalité des champs clés est petite.
Par contre, le maintient de ces index devient coûteux en présence de fichiers dynamiques
……
……
……
……
……

Question 2 :

a) Nous disposons d'une table T en MC assez grande pour contenir tous les couples (clé,adr) récupérés à partir des K premiers niveaux d'un fichier de données F de type B-arbre. Nous souhaitons utiliser T comme une table d'index non dense vers les données se trouvant dans le niveau K+1 du B-arbre F.

Donnez une solution récursive efficace pour remplir cette table (sans avoir à faire des décalages) à partir du contenu des K premiers niveaux du B-arbre. **(6 pts)**

On réalise un parcours inordre (récursif) en se limitant aux K premiers niveaux de l'arbre

Variables globales : le fichier F et la table d'index T

Prog. Principal :

```
ouvrir( F , « fichier_b_arbre » , 'A' )
rac ← entete( F , 1 )                          // le numéro du bloc racine
nb_niv ← K
i ← 1

// la table d'index T sera principalement remplie dans la procédure 'inordre'
inordre( rac, nb_niv , dernierFils , i )
// la dernière ligne de la table d'index :
//        ∞ (la plus grande valeur possible) , dernierFils (le dernier bloc du niveau K+1)
T[ i ] ← { ∞ , dernierFils }

fermer( F )
```

```
// parcours inordre limité aux 'nbNiv' premiers niveaux de l'arbre
Procedure inordre( entrées : r , nbNiv:entier  /  entrées/sorties : dern , i : entier )
    var locales : j et buf
    SI ( r <> -1 )
        LireDir( F , r , buf )
        POUR j = 1 , buf.deg – 1
            SI ( nbNiv = 1 )
                T[ i ] ← { buf.val[ j ] , buf.Fils[ j ] }
                i++
            SINON
                inordre( buf.Fils[ j ] , nbNiv-1 , dern , i )
                T[ i ] ← { buf.val[ j ] , dern }
                i++
            FSI
        FP
        SI ( nbNiv = 1 )
            dern ← buf.Fils[ buf.deg ]
        SINON
            inordre( buf.Fils[ buf.deg ] , nbNiv-1 , dern , i )
        FSI
    FSI
```

b) Dans la fusion parallèle (sur un cluster de K nœuds) vue en cours, le premier nœud se charge, entre autre, de définir les bornes b1, b2,… bk et les diffuse aux autres nœuds N2, N3, … Nk. Donnez l'algorithme permettant aux autres nœuds (N2, N3, … Nk) de découper leurs fragments respectifs à l'aide des bornes reçus de N1. Nous supposerons que ces bornes ont déjà été reçues et stockées dans un tableau B de taille K et que toutes les valeurs (dans tous les fragments) sont inférieures ou égales à bk. **(6 pts)**

```
// F et les fragments résultats sont des fichiers TOF
// Au niveau de chaque nœud Nj (j=2 ,3,… K), on a un fichier F (trié) représentant son fragment en entrée
// Les bornes b1, b2, ...bk envoyés par N1 sont déjà placés dans un tableau B
// Le résultat de cette opération est la création de K fichiers résultats frag1, frag2, … fragK à partir de F
// Chaque fichier fragi (pour i=2..K) contiendra les données de F comprises entre bi-1 et bi
// frag1 contiendra les données de F qui sont ≤ b1
```

Première solution

On parcourt séquentiellement le fichier en entrée F bloc par bloc et enreg par enreg, tout en traitant les bornes

```
Les variables fichiers :
        F : FICHIER de Tbloc BUFFER buf ENTETE( entier )        // nombre de blocs
        G : FICHIER de Tbloc BUFFER buf2 ENTETE( entier )       // nombre de blocs

ouvrir( F , « fichier_du_noeudj » , 'A') ;  N ← Entete(F,1)     // le nombre de blocs dans F
i2 ← 1 ;  j2 ← 1 ;  m ← 1                                       // m : num du fragment à générer
ouvrir( G , « frag »+m , 'N' )
POUR i = 1 , N              // parcours séquentiel du fichier en entrée F
        LireDir( F , i , buf ) ;  j ← 1
        TQ  ( j ≤ buf.NB  )
                e ← buf.tab[ j ]
                SI ( e.clé ≤ B[m] )
                        buf2.tab[ j2 ] ← e ;  j2 ++
                        // écriture du bloc i2 lorsqu'il devient plein ...
                        SI ( j2 > b ) buf2.NB ← b ; EcrireDir( G , i2 , buf2 ) ;  i2 ++ ; j2 ← 1  FSI
                        // traitement du prochain enreg de F ...
                        j++
                SINON
                        // fin du fragment m …
                        SI ( j2 > 1 )  // écriture du dernier bloc de frag_m (s'il n'est pas vide)
                                buf2.NB ← j2 – 1 ; EcrireDir( G , i2 , buf2 )
                        SINON  i2 --
                        FSI
                        Aff_entete( G , 1 , i2 )   // nombre de blocs dans le fragment m
                        fermer( G )

                        // début du nouveau frag (m+1) ...
                        m++ ;  ouvrir( G , « frag »+m , 'N' ) ;  i2 ← 1 ;   j2 ← 1
                FSI
        FTQ
FP
fermer( F )

// Dernière écriture dans le dernier fragment K si buf non vide
SI ( j2 > 1 ) buf2.NB ← j2 - 1 ; EcrireDir( G, i2 , buf2 )   SINON i2 --    FSI
Aff_entete( G , 1 , i2 )
fermer( G )
```

Deuxième solution
On traite les bornes une par une, tout en parcourant séquentiellement le fichier en entrée F

Les variables fichiers :
F : FICHIER de Tbloc BUFFER buf ENTETE(entier) *// nombre de blocs*
G : FICHIER de Tbloc BUFFER buf2 ENTETE(entier) *// nombre de blocs*

```
ouvrir( F , « fichier_du_noeudj » , 'A') ;  i ← 1 ; j ← 1 ; N ← Entete(F,1)
ouvrir( G , « frag »+m , 'N' ) ;  i2 ← 1 ;  j2 ← 1
SI ( N > 0 )  LireDir( F , 1 , buf ) ;  FinF ← faux  SINON  FinF ← vrai  FSI

m ← 1   // numéro de la borne à traiter
TQ ( m ≤ K et non FinF )
        e ← buf.tab[ j ]
        SI ( e.clé ≤ B[m] )
                buf2.tab[ j2 ] ← e ;  j2 ++
                SI ( j2 > b )  buf2.NB ← b ;  EcrireDir( G , i2 , buf2 ) ;  i2 ++ ;  j2 ← 1  FSI

                j++
                SI ( j > buf.NB )
                        SI ( i < N )        i++ ;  j ← 1 ;  LireDir( F , i , buf )
                        SINON               FinF ← vrai
                        FSI
                FSI

        SINON
                // fin du fragment m ….
                SI ( j2 > 1 )
                        buf2.NB ← j2 – 1
                        EcrireDir( G, i2 , buf2 )
                SINON
                        i2 --
                FSI
                Aff_entete( G , 1 , i2 )   // le nombre de blocs dans G « frag_m »
                fermer( G )

                // début du prochain fragment m+1 ….
                m++
                SI ( m ≤ K ) ouvrir( G , « frag »+m , 'N' ) ;  i2 ← 1 ;  j2 ← 1 FSI
        FSI
FTQ
fermer( F )

SI ( m ≤ K )        // si le parcours de F se termine avant d'avoir traité toutes les bornes ...
        SI ( j2 > 1 )  buf2.NB ← j2 – 1
                       EcrireDir( G, i2 , buf2 )
        SINON          i2 --
        FSI
        Aff_entete( G , 1 , i2 )
        fermer( G )
        m++
        TQ  ( m ≤ K )
                ouvrir( G , « frag »+m , 'N' ) ;  Aff_entete( G , 1 , 1 ) ;  fermer( G ) ;  m++
        FTQ
FSI
```