Corrigé - Contrôle Intermédiaire - SFSD / ESI 2023-2024 – Barème : Q1(12) , Q2(8)

Q 1 : Soit F un fichier TOF
a) Qu'est-ce qu'une opération de réorganisation de F et dans quelles conditions est-elle utile ?

Réponse 1-a : (3 pts)

- Une réorganisation est une opération permettant de régénérer le fichier pour que les performances redeviennent optimales, soit en réutilisant les anciens blocs (donc le fichier existant) soit en créant un nouveau fichier que l'on charge avec les enregistrements de l'ancien fichier. Par exemple, pour un fichier TOF, on peut effectuer la réorganisation juste en créant un nouveau fichier et en laissant un peu de vide dans chaque bloc (C'est en fait un chargement initial avec les enregistrements de l'ancien fichier qui ne sont pas supprimés logiquement).
- Une opération de réorganisation d'un fichier ordonné TOF est nécessaire lorsque les performances générales se dégradent. Cela se produit lorsqu'en général le facteur de chargement moyen devient élevé (se rapproche de 100%) car l'insertion d'un nouvel enregistrement peut alors provoquer un nombre important de décalages inter-blocs (donc les insertions deviennent très coûteuses).

b) Expliquez pourquoi la complexité de l'algorithme d'insertion en opérations d'E/S est O(N) avec N le nombre de blocs de F ?

Réponse 1-b : (2 pts)

L'insertion dans un fichier TOF nécessite des décalages d'enregistrement pour maintenir l'ordre des clés. Dans le pire cas, lorsqu'un grand nombre de blocs (éventuellement tous les blocs) sont remplis à 100%) la moindre insertion provoque un grand nombre de décalages inter-blocs (éventuellement N décalages inter-blocs, avec N le nombre de blocs total du fichier). Comme chaque décalage inter-bloc nécessite une lecture et une écriture de blocs, le coût de l'opération peut atteindre 2N opérations d'E/S, d'où une complexité de O(N).

c) Si la capacité maximale des blocs est de 40 enregistrements et si le fichier F est composé de 20 million d'enregistrements avec un facteur de chargement moyen de 50 % , quelle est alors le coût d'une recherche d'un enregistrement dans F ?

Réponse 1-c : (2 pts)

La recherche dans un fichier TOF est dichotomique, cela veut dire qu'elle nécessite, dans le pire cas, Log2N opérations de lecture de blocs (avec N le nombre de blocs total du fichier).
Avec un facteur de chargement de 50 % et une capacité maximale de 40 enregistrements par blocs, en moyenne les blocs vont contenir 20 enregistrements chacun. Comme li y a 20 000 000 d'enregistrements au total, cela veut dire que le fichier est formé approximativement par :
N = 20 000 000 / 20 = 1 000 000 blocs
Donc le coût de recherche dans le pire cas est aux environs de : $Log_2\ 10^6$
Coût de recherche ≈ 20 opérations de lecture

d) Donnez un algorithme utilisant 2 buffers buf1 et buf2 permettant d'équilibrer le nombre d'enregistrements dans les 2 blocs i et i+1 de F (inclure les déclarations de F et des buffers)

Réponse 1-d : (5 pts)

```
type Tbloc = structure
       tab : tableau[ b ] d'enreg
       NB : entier
Fin

Soit F : fichier de Tbloc buffer buf1 , buf2 entete( entier )
// supposons que le fichier F est déjà ouvert, sinon on fait ouvrir( F, '…..' , 'A' )

LireDir( F , i , buf1 )
LireDir( F , i+1 , buf2 )
```

```
/*************************************************************************************************
Si les buffers buf1 et buf2 ne sont pas équilibrés, nous devons transférer un certain nombre
d'enreg de buf2 vers buf1 ou de buf1 vers buf2

Exemple1 :
|______|7            |______|7                          |______|7            |______|7
|_ffff_|6            |______|6                          |______|6            |______|6
|_eeee_|5            |______|5                          |______|5            |_iiii_|5
|_dddd_|4            |______|4                          |_dddd_|4            |_hhhh_|4
|_cccc_|3            |_iiii_|3            ===>          |_cccc_|3            |_gggg_|3
|_bbbb_|2            |_hhhh_|2                          |_bbbb_|2            |_ffff_|2
|_aaaa_|1            |_gggg_|1                          |_aaaa_|1            |_eeee_|1
  buf1                buf2                              buf1                buf2
  NB=6                NB=3       moyenne=4              NB=4                NB=5

                              nb enreg transf de buf1 à buf2 = 2


Exemple2 :
|______|7            |_____|7                           |______|7            |______|7
|_____|6             |_____|6                           |______|6            |______|6
|_____|5             |_gggg_|5                          |_____|5             |_____|5
|_____|4             |_ffff_|4                          |_____|4             |_gggg_|4
|_____|3             |_eeee_|3            ===>          |_cccc_|3            |_ffff_|3
|_bbbb_|2            |_dddd_|2                          |_bbbb_|2            |_eeee_|2
|_aaaa_|1            |_cccc_|1                          |_aaaa_|1            |_dddd_|1
  buf1                buf2                              buf1                buf2
  NB=2                NB=5       moyenne=3              NB=3                NB=4

                              nb enreg transf de buf2 à buf1 = 1

*************************************************************************************************/

moyenne ← ( buf1.NB + buf2.NB ) / 2
SI ( buf1.NB < moyenne )
        cpt ← 1
        POUR k = buf1.NB+1 , moyenne
                buf1.tab[ k ] ←  buf2.tab[ cpt ]
                cpt++
        FP
        // a ce niveau cpt-1 représente le nb d'enreg transférés de buf2 à buf1
        buf1.NB ← moyenne
        POUR k=cpt,buf2.NB
                buf2.tab[ k-cpt+1 ] ← buf2.tab[ k ]
        FP
        buf2.NB ← buf2.NB - cpt +1
        EcrireDir( F , i , buf1 ) ; EcrireDir( F , i+1 , buf2 )
SINON
        SI ( buf2.NB < moyenne )
                cpt ← moyenne - buf2.NB      // nb d'enreg à ajouter à buf2
                // décaler de cpt position vers la droite dans buf2 pour laisser la place
                // aux enreg à ramener depuis buf1 …
                POUR k = buf2.NB , 1 , -1
                        buf2.tab[ k+cpt ] ← buf2.tab[ k ]
                FP
                buf2.NB ← buf2.NB + cpt
                // transférer cpt enreg de buf1 vers buf2 …
                POUR k = 1 , cpt
                        buf2.tab[ k ] ←  buf1.tab[ buf1.NB - cpt + 1 ]
                FP
                buf1.NB ← moyenne   // ou alors buf1.NB ← buf1.NB - cpt
                EcrireDir( F , i , buf1 ) ; EcrireDir( F , i+1 , buf2 )
        FSI
FSI
```

```
/*** une 2e solution utilisant un tableau de capacité 2b ***/
Soit T:tableau[ 2b ] d'enreg

LireDir( F , i , buf1 )
LireDir( F , i+1 , buf2 )

// remplir le tableau T avec le contenu des 2 blocs ...
k ← 1
POUR j = 1 , buf1.NB  T[ k ] ← buf1.tab[ j ] ;  k++  FP
POUR j = 1 , buf2.NB  T[ k ] ← buf2.tab[ j ] ;  k++  FP

// mettre la 1ere moitié dans buf1 et la 2e moitié dans buf2 …
POUR j = 1 , k-1
        SI ( j ≤ (k-1 div 2) )
                buf1.tab[ j ] ← T[ j ]
        SINON
                buf2.tab[ j - (k-1 div 2) ] ← T[ j ]
        FSI
FP
buf1.NB ← (k-1 div 2)
buf2.NB ← k-1 – buf1.NB

EcrireDir( F , i , buf1 )
EcrireDir( F , i+1 , buf2 )
```

Q 2 :

Soit F un fichier TOVC. (blocs contigus, non ordonné, format variable sans chevauchement). L'unique caractéristique du fichier est le numéro du dernier bloc. La structure des blocs est comme suit :

```
Type Tbloc = Struct
        tab : tableau[ b ] de caractères
        NB : entier  // le nombre d'enregistrements présents dans tab
Fin
```

Les enregistrements sont formés par une clé de taille 10 caractères, et une information (représentant les autres champs) de taille variable.

a) Donnez la déclaration de F avec un seul buffer en mémoire centrale

Réponse 2-a : (1 pt)

F : fichier de Tbloc buffer buf entete(entier)

b) Dans le cas où les enregistrements d'un bloc sont stockés l'un à la suite de l'autre (et séparés par le caractère spécial '#') dans le tableau 'tab', donnez l'algorithme de recherche d'une clé C. Les résultats retournés par la recherche sont :

- un booléen (trouv) indiquant si la clé existe ou non
- l'adresse sous forme de numéro de bloc et déplacement (i,j) de l'enregistrement (s'il existe)

Réponse 2-b : (4 pts)

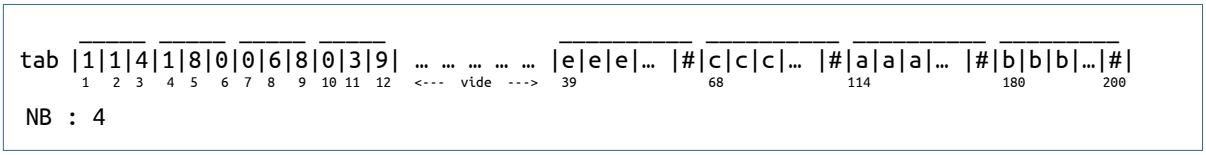
```
Rech( entrée : C:chaine , sorties : trouv:bool , i : entier, j : entier )

// supposons que le fichier F est déjà ouvert, sinon on fait ouvrir( F, '…' , 'A' )
N ← Entete( F, 1 ) // le num du dernier bloc
i ← 1 ; trouv ← faux
```

```
TQ ( i ≤ N && non trouv )   // parcours séquentiel de F en MS …
       LireDir( F , i , buf )
       cpt ← 1
       j ← 1
       TQ ( cpt ≤ buf.NB && non trouv )   // parcours séquentiel du buffer en MC …
              // récupérer la clé de l'enreg num cpt …
              POUR k = 1,10
                     cle[ k ] = buf.tab[ j+k-1 ]
              FP
              // tester si c'est l'enreg cherché …
              SI ( C == cle )
                      trouv ← vrai
              SINON
                     // avancer jusqu'à la fin de l'enreg num cpt …
                     TQ ( buf.tab[ j+10 ] <> '#' )  j++  FTQ
                     j ← j + 11
                     cpt++
              FSI
       FTQ    // fin du parcours d'un buffer en MC
       SI ( non trouv ) i++  FSI
FTQ  // fin du parcours séquentiel de F en MS
```

c) Nous souhaitons pouvoir effectuer une recherche dichotomique en mémoire centrale dans le buffer (lors de la recherche d'une clé). Est-ce que c'est possible de le faire ? (si oui, donnez une solution, si non, expliquez pourquoi ce n'est pas possible de le faire)

Réponse 2-c : (3 pts)

C'est possible de faire une recherche dichotomique dans le buffer, en gardant un petit index dans la partie gauche du tableau 'tab'. Les enregistrements seront stockés dans la partie droite du tableau 'tab'.
L'index (dans la partie gauche de 'tab') est formé par les indices de début des enregistrements se trouvant dans la partie droite. Pour pouvoir effectuer une recherche dichotomique, les entrés de l'index sont ordonnées selon les valeurs de clé des enregistrements de la partie droite.
Voir le schéma ci-dessous :



Dans cet exemple, un bloc (de taille b = 200 caractères + le champ NB) contient 4 enregistrements stockés dans la partie droite du tableau 'tab' (dans un ordre quelconque) :

'***eeeeeeeee****....#'* de longueur 28 caractères, entre les indices 39 et 67
'***ccccccccc****....#'* de longueur 45 caractères, entre les indices 68 et 113
'***aaaaaaaaa****....#'* de longueur 65 caractères, entre les indices 114 et 179
'***bbbbbbbbb****....#'* de longueur 20 caractères, entre les indices 180 et 200

Dans ce même exemple, l'index dans la partie gauche du tableau 'tab', contient 4 entrées représentant les indices de début des 4 clés appartenant aux 4 enregistrements stockés dans ce bloc (Les indices sont ordonnés selon les valeurs des clés). Chaque indice occupe 3 positions (3 caractères chiffres) :

à l'indice 1 on trouve '***114***' qui représente l'indice de début de 'aaaaa...'
à l'indice 4 on trouve '***180***' qui représente l'indice de début de 'bbbbb...'
à l'indice 7 on trouve '***068***' qui représente l'indice de début de 'ccccc...'
à l'indice 10 on trouve '***039***' qui représente l'indice de début de 'eeeee...'

La recherche dichotomique dans un buffer peut alors s'effectuer comme suit :

```
Recherche_interne( entrée  C:chaine , sorties trouv:bool, j : entier )
bi ← 1 ;  bs ← buf.NB ;  trouv ← faux
TQ ( bi ≤ bs && non trouv )
        j ← (bi + bs) div 2
       indice = buf.tab[ 3*j-2 ] * 100 + buf.tab[ 3*j-1 ] * 10 + buf.tab[ 3*j ]
       // récupérer la clé du milieu …
       POUR k = 1,10
              cle[ k ] ← buf.tab[ indice + k - 1 ]
       FP
       // tester la clé …
       SI ( C == cle ) trouv ← vrai
       SINON
              SI ( C < cle )  bs ← j - 1
              SINON
                     bi ← j + 1
              FSI
       FSI
FTQ
SI ( non trouv )  j ← bi  FSI
```

Lors de l'insertion d'un nouvel enreg de clé C, on insère l'enreg dans 'tab', dans la partie droite du vide (sans décalage et sans maintenir l'ordre entre les clés). Soit k l'indice où a été inséré le nouvel enreg. Puis on insère k par décalages dans l'index (la partie gauche de 'tab') à la position 'j' (retournée par Recherche_interne).