CORRIGE C.I. SFSD 2022/2023 (version 1)

Question 1 (version 1) :

a) Veuillez compléter l'algorithme « Insertion dans T~OF avec index dense et clés à valeurs uniques » (2 pts)

```
Type Tbloc = Struct                                  Type Tcouple = Struct
        tab : tableau[ b ] de typeEnreg ;                    clé : typeqlq ;
        NB : entier                                          numBlc , depl : entier
Fin                                                  Fin
Var     F : FICHIER de Tbloc BUFFER buf ENTETE ( entier )
        Index : tableau [ MaxIndex ] de Tcouple
        NbE : entier // nombre d'éléments dans la table index ( == nombre d'enreg dans le fichier F)

Ins( e:TypeEnreg )
        Rech( e.cle , trouv , k ) // Recherche (dichotomique) dans la table index
        SI ( Non trouv ) // insertion à la fin du fichier de données ...
                OUVRIR(F, « donnees.dat » , 'A')
                i ← Entete(F , 1) // n° du dernier bloc de F
                LireDir(F , i , buf)
                SI ( buf.NB < b ) buf.NB++ ;  j ← buf.NB ; buf.tab[ j ] ← e ;
                                  EcrireDir( F , i , buf )
                SINON  i++ ; j ← 1 ; buf.NB ← 1 ; buf.tab[ j ] ← e ;
                        Aff_entete(F, 1, i) ;
                        EcrireDir(F , i , buf)
                FSI
                FERMER(F)
                // insertion dans la table d'index ...Veuillez compléter l'algorithme.

                NbE++ ;
                m ← NbE
                TQ ( m > k ) // décalage dans la table index car table ordonnée selon la clé
                        Index[ m ] ← Index[ m-1 ] ;
                        m - -
                FTQ
                Index[ k ] ← < e.c , i , j > // insérer la clé, numBlc, depl dans l'index

        FSI
```

b) Veuillez compléter : (2 pts)

- L'index dense contient … … … **toutes les clés du fichier**
- L'utilisation de l'index non dense se limite uniquement aux fichiers … **ordonnés**

Question 2 (version 1) :

a) Quelle est l'avantage essentiel d'une structure de fichier de données vue comme tableau avec des enregistrements de taille fixe (TOF). Pourquoi ? (1 pt)

- **La recherche d'une donnée selon sa clé est dichotomique**
- **Le coût est d'ordre logarithmique selon le nombre de blocs du fichier.**

b) Quel est son inconvénient majeur dans ce cas. Pourquoi ? (1pt)

- **L'insertion d'une donnée peut provoquer des décalages inter blocs pour maintenir l'ordre dans le fichier de donnée.**
- **Son coût est le coût de la recherche + N lectures + (N+1) Ecritures, donc sa complexité est linéaire donc assez coûteuse.**

Question 3 (version 1) :

Quel est l'avantage d'utiliser la méthode des « listes inversées » par rapport à l'utilisation de plusieurs index indépendants ? (2 pts)

En cas d'opération impliquant un changement d'adresse d'un enregistrement dans le fichier de données (suppression, déplacement…), la mise-à-jour de l'index primaire suffit dans la méthode des « listes inversées », contrairement à l'utilisation de plusieurs index indépendants qui nécessitent des changement au niveau de chaque index existant.

Question 4 (version 1) :

Qu'est-ce que le facteur de chargement d'un fichier ? (2 pts)

Veuillez cocher la ou les bonnes réponses :

- √ **Le remplissage moyen des blocs d'un fichier**
- Le rapport : nombre de blocs vides / nombre de blocs pleins
- √ **Le pourcentage de remplissage du fichier**
- √ **Nombre de données insérées sur Nombre de places disponibles**
- Nombre de données dans tous le fichier
- Nombre de données dans la première moitié du fichier
- √ **Nombre de données insérées sur (Nombre de blocs utilisés * taille du bloc)**
- Le rapport : nombre d'enreg supprimés / nombre d'enregistrements insérés

Question 5 (version 1) :

a) Dessiner un B-arbre d'ordre 5, obtenu après l'insertion des 12 nombres suivants, dans cet ordre : 30 , 60 , 32 , 56 , 13 , 44 , 75 , 8 , 27 , 90 , 78 , 25 (2 pts)

racine
25 | 32 | 60 | __

f1	**f2**	**f3**	**f4**
8 | 13 | __ | __	**27 | 30 | __ | __**	**44 | 56 | __ | __**	**75 | 78 | 90 | __**

b) Dessiner le B-arbre après la suppression de 32 (2 pts)

racine
25 | 44 | 75 | __

f1	**f2**	**f3**	**f4**
8 | 13 | __ | __	**27 | 30 | __ | __**	**56 | 60 | __ | __**	**78 | 90 | __ | __**

Question 6 (version 1) :

Soit un B-arbre d'ordre 100 contenant plus de 3 niveaux. Quel est le nombre minimal de blocs dans le 3e niveau de l'arbre ? (2 pt)

Pour la racine le degré minimum est 2 (une valeur et 2 fils)
Pour les autres nœuds le degré minimum est 50 (49 valeurs et 50 fils)
Donc dans le 1er niveau il y a un seul nœud ayant 2 fils
Dans le 2e niveau il y a 2 nœuds ayant 50 fils chacun
Dans le 3e niveau il y a 2*50 nœuds = 100 nœuds

Question 7 (version 1) :
Nous disposons d'un fichier où chaque enregistrement est composé d'une dizaine de mesures générées pas des capteurs situés dans une turbine à gaz industrielle génératrice de courant électrique. La clé de recherche est une valeur horodatée (timestamp), c-a-d une concaténation de la date courant AAAA/MM/JJ et de l'heure courante HH:MM:SS sous forme d'une chaîne de caractères. Chaque seconde, un nouvel enregistrement est ajouté au fichier. La taille d'un enregistrement est de 128 octets et la taille d'un bloc physique est de 4ko (c-a-d 4096 octets). Un bloc peut donc contenir 32 enregistrements. Nous souhaitons utiliser une structure de fichier ordonné TOF avec une table d'index non dense en MC.
Tous les blocs du fichier sont pleins à 100 % sauf éventuellement le dernier.
Un bloc est alors vu comme un simple tableau de 32 enregistrements.
Donnez les déclarations nécessaires (en se limitant à une taille de l'entête minimale) ainsi que l'algorithme pour insérer le dernier enregistrement généré. (4 pt)

// déclaration du fichier de données avec son entête représentant le nb d'enreg total
F : FICHIER de tableau[32] enreg BUFFER buf ENTETE (entier)
// déclaration de la table d'index non dense (par exemple : une clé par bloc)
index : TABLEAU[MaxInd] de structure(cle:chaine , numB:entier)
nbInd = 0

// Algo d'insertion en fin de fichier avec m-a-j éventuellement de l'index …
Ins(e) //la clé du nouvelle enreg e est e.clé
OUVRIR(F , « fichier.dat », 'A') ***// optionnelle***
DernBloc = 1 + ***Entete(F,1)*** **div 32**
PosLibre = 1 + ***Entete(F,1)*** **mod 32**
SI (PosLibre > 1) // le dernier bloc n'est pas plein …
LireDir(F , DernBloc , buf)
FSI
buf[PosLibre] = e
EcrireDir(F , DernBloc , buf)
Aff_entete(F , 1 , Entete(F,1) + 1)
FERMER(F) ***// optionnelle***

// M-A-J de l'index …
SI (PosLibre > 1)
// s'il n'ya pas eu ajout d'un nouveau bloc …
// on modifie uniquement la clé de la dernière ligne de l'index
index[nbInd] . clé = e.cle
SINON // si on a ajouté un nouveau bloc (DernBloc), on ajoute une ligne à l'index
nbInd++ ;
index[nbInd] = (e.cle , DernBloc)
FSI

Une autre version peut éventuellement être acceptée, elle suppose que le dernier bloc est déjà en MC et que l'écriture ne se fait que lorsque le bloc est plein :
DernBloc = 1 + ***Entete(F,1)*** **div 32**
PosLibre = 1 + ***Entete(F,1)*** **mod 32**
SI (PosLibre == 1) ***EcrireDir(F , DernBloc , buf)*** ***FSI***
buf[PosLibre] = e
Aff_entete(F , 1 , Entete(F,1) + 1)

// M-A-J de l'index … (même chose que la 1ere solution)
SI (PosLibre > 1)
// s'il n'ya pas eu ajout d'un nouveau bloc …
// on modifie uniquement la clé de la dernière ligne de l'index
index[nbInd] . clé = e.cle
SINON // si on a ajouté un nouveau bloc (DernBloc), on ajoute une ligne à l'index
nbInd++ ;
index[nbInd] = (e.cle , DernBloc)
FSI