Corrigé de l'examen de Structures de Fichiers 2019/2020

1. *Soient E et F deux ensembles sous forme de deux fichiers d'entiers formés de N blocs chacun.*
Est-il possible de calculer la différence des deux ensembles (E - F) avec une complexité moyenne en O(N) ?
(Pour rappel, E - F représente l'ensemble des éléments de E qui ne sont pas dans F)

Oui c'est possible de calculer la différence E – F en O(N) en moyenne en adoptant un algorithme de type '**jointure par hachage**'. Le principe est comme suit :
La différence de 2 ensembles peut être effectuée comme une opération d'intersection (qui est un type d'équi-jointure). Pour chaque élément de E on vérifie s'il existe le même élément dans F, si le résultat est négatif, l'élément fait partie de l'ensemble résultat (E – F).
Donc on peut utiliser pratiquement les mêmes algorithmes que ceux de l'équi-jointure. Le plus rapide est celui par hachage (O(N) en moyenne).

```
Fragmenter E avec une fonction de hachage h en k fragments : E0, E1, … Ek-1
Fragmenter F avec la même fonction de hachage h en k fragments : F0, F1, … Fk-1
Pour i = 0, k-1
        Charger le fragment Fi en MC (et construire une table de hachage avec une autre fonction h')
        Parcourir le fragment Ei bloc par bloc et pour chaque enreg x faire :
                Vérifier avec h' si x existe dans la table de hachage de Fi
                        S'il n'existe pas, mettre x dans le résultat
        FP
FP
```

Cet algo est en O(N) en prenant comme hypothèse que l'on dispose d'assez d'espace mémoire pour charger n'importe quel fragment de Fi de F en MC.

2. *Qu'est-ce qu'une valeur fictive dans un B+ arbre et quel est son intérêt ?*
Est-ce que la fusion de deux nœuds dans un B+ arbre est différente de son équivalent dans un B arbre de base ?

Dans un B+ arbre, les valeurs se trouvant dans les nœuds internes sont normalement des copies de certaines valeurs dans le niveau feuille (ce sont les premières clés de chaque feuille, à part la première feuille). Lors de la suppression d'une valeur se trouvant en première position d'une feuille, sa copie dans les niveaux interne n'est pas forcément supprimée. Cette valeur devient donc fictive (car elle n'existe pas réellement dans le niveau feuille), elle est là juste pour guider la recherche dans une branche vers le niveau feuille.
Donc on peut dire qu'une valeur fictive dans un B+ arbre est un séparateur dans un nœud interne pour lequel il n'existe pas de clé associée dans le niveau feuille. L'intérêt de garder ces valeurs fictives est de diminuer légèrement le coût de certaines suppressions (celles concernant les premières valeurs des nœuds feuille).

La fusion dans B+ est légèrement différente avec son équivalent dans les B arbres de base lorsqu'il s'agit de nœuds feuilles. Dans ce cas et pour un B+ arbre, le séparateur se trouvant dans le nœud parent n'est pas déplacé vers le nœud feuille résultant de la fusion car soit ce séparateur était une valeur fictive (donc elle ne doit pas exister au niveau feuille), soit sa clé associée était déjà présente dans l'une des feuilles subissant la fusion et de ce fait cette clé sera déplacée (avec les autres clés) vers la feuille résultat de la fusion. Par contre la fusion de nœuds internes est similaire à celle de son équivalent dans les B arbre de base, car les nœuds internes d'un B+ forment en fait un B arbre classique.

3. *Donnez un algorithme (en étudiant son coût) qui fragmente un fichier d'entiers, formé par N blocs, en K différents fragments en utilisant une fonction de hachage h retournant des valeurs entre 0 et K-1.*
Nous disposons pour cette opération de K+1 buffers en mémoire centrale.

Soit buf un tableau de K+1 buffers. Les K premiers buf[1], … buf[K] seront utilisés comme buffers de sortie pour la construction des K fragments et le dernier buffer buf[K+1] sera utilisé comme buffer d'entrée pour lire le fichier à fragmenter.
Soit F un tableau de K+1 variables fichiers. Les K premiers représentent les fichiers de sortie (les K fragments) et le dernier élément F[K+1] représente le fichier en entrée.
Soient NumB et Ind deux tableau de K entiers représentant respectivement les numéros de blocs et les déplacements pour les K fragments en sortie.

Chaque buffer est une structure contenant un tableau de b entier (tab) et un entier (NB) représentant le nombre d'élément présents dans le tableau tab.

Tous les fichiers ont une seule caractéristique désignant le numéro du dernier bloc.

```
// ouverture des fichiers et initialisations …
Ouvrir( F[K+1] , « Fichier_a_fragmenter », 'A' ) ;
Pour i = 1, K
        Ouvrir( F[i] , « Frag » + chaine(i) , 'N' ) ;
        NumB[ i ] = 1 ;
        Ind[ i ] = 1 ;
FP ;

// Boucle principale pour parcourir le fichier en entrée séquentiellement …
Pour i = 1, Entete( F[K+1] , 1 )
        LireDir( F[K+1] , i , buf[K+1] ) ;
        Pour j = 1 , buf[K+1].NB
                e = buf[K+1].tab[j] ;
                // hacher l'enreg pour connaître son fragment de destination
                m = h(e) + 1
                // placer e dans le frag m
                buf[ m ] . tab[ Ind[m] ] = e ;
                Ind[m] ++ ;
                Si ( Ind[m] > b )
                        buf[ m ] . NB = b ;
                        EcrireDir( F[m] , NumB[m] , buf[m] ) ;
                        NumB[m]++ ;
                        Ind[m] = 1 ;
                Fsi
        FP // j
FP // i
// Vider les derniers buffers non encore écrits et fermer les fichiers …
Pour i = 1 , K
        Si ( Ind[ i ] > 1 )
                buf[ i ] . NB = Ind[ i ] - 1 ;
                EcrireDir( F[ i ] , NumB[ i ] , buf[ i ] ) ;
                Aff_entete( F[ i ] , 1 , NumB[ i ] ) ;
                Fermer( F[ i ] )
        Sinon
                Aff_entete( F[ i ] , 1 , NumB[ i ] - 1 ) ;
                Fermer( F[ i ] )
        Fsi
FP
Fermer( F[ K+1 ] )
```

Le coût en lectures est N car le fichier en entrée est parcouru une seule fois.
Le coût en écriture varie entre N et N+K
Dans le pire cas, tous les blocs du fichier d'entrée étaient pleins à 100 % (à part éventuellement le dernier). Il en sera de même pour les fichiers de sortie (les fragments), tous leurs blocs seront pleins à 100 % sauf éventuellement les dernier de chaque fragments. Ce qui fait qu'on aura, dans le pire cas, K blocs additionnels (un par fragment) rajoutés en fin de fragments.

Est-il possible d'effectuer une telle fragmentation en K fragments avec seulement 2 buffers ? Si c'est oui donnez un algorithme avec son coût, sinon dites pourquoi cela ne serait pas possible

Oui c'est possible de fragmenter le fichier en entrée, en K différents fragments avec seulement 2 buffers.
L'un des buffer buf1 sera utilisé pour parcourir les blocs du fichier en entrée F1 et le deuxième buffer buf2 pour rajouter les enregistrements aux fragments de sortie F2 sélectionné pas la fonction de hachage h.

On peut envisager pour cela deux approches :
Dans la première approche on effectue K parcours du fichier en entrée, et à chaque parcours i on construit un fragment i. **Le coût de cette approche est : K*N lectures + (N+K) écritures**
Dans la deuxième approche on effectue un seul parcours du fichier en entrée, et pour chaque enregistrement récupéré on le rajoute à l'un des fragments de sortie. Dans le pire cas on doit lire et écrire autant de blocs que d'enregistrements dans F1 (**N lectures + 2bN écritures**).

Voici une solution selon la 1ere approche :

```
Ouvrir( F1, « …. » , 'A') ;
Pour i = 1, K
        Ouvrir( F2, « frag »+chaine(i), 'N') ;
        j2 = 1 ; i2 = 1 ;
        Pour i1 = 1 Entete( F1 , 1)
                LireDir( F1 , i1 , buf1 ) ;
                Pour j1 = 1 , buf1.NB
                        e = buf1.tab[j1] ;
                        Si ( h(e) == i )
                                // placer e dans le frag i
                                buf2 . tab[ j2 ] = e ;
                                j2 ++ ;
                                Si ( j2 > b )
                                        buf2 . NB = b ;
                                        EcrireDir( F2 , i2 , buf2 ) ;
                                        i2 ++ ;
                                        j2 = 1 ;
                                Fsi
                        Fsi
                FP // j1
        FP // i1
        Si (j2 > 1 )
                buf2 . NB = j2 - 1 ;
                EcrireDir( F2 , i2 , buf2 ) ;
                Aff_entete( F2 , 1 , i2 ) ;
                Fermer( F2 )
        Sinon
                Aff_entete( F2 , 1 , i2 - 1 ) ;
                Fermer( F2 )
        Fsi
FP // i variant de 1  à K
Fermer( F1 )
```

4. *Quel est le nombre minimal de buffers à avoir pour pouvoir trier un fichier formé de N blocs.*
Quel est dans ce cas le temps nécessaire (approximativement) pour le tri d'un fichier de 3 000 000 de blocs avec une mémoire secondaire asymétrique ayant les caractéristiques suivantes :
Temps de lecture d'un bloc = 10 µs Temps d'écriture d'un bloc = 100 µs (pour rappel 1 µs = 10^{-6} s)

Le nombre minimum de buffers nécessaire pour le tri externe d'un fichier est celui nécessaire pour la fusion, c-a-d : **3 buffers**.

Exemple numérique :

Avec 3 buffers, l'étape de fragmentation va générer 1 000 000 petits fragments de taille 3 blocs chacun
Elle coûte donc 3 000 000 de lectures (30 s) et 3 000 000 d'écritures (300 s), soit 330 secondes.
L'étape de fusion nécessitera 20 ($\log_2(1000000) \approx 20$) phases de multi-fusions (des fusions deux par deux).
A chaque phase, les 3 000 000 de blocs sont lus une seule fois et écrits une seule fois.
Donc chaque phase de multi-fusion coûte aussi 3 000 000 de lectures (30 s) et 3 000 000 d'écritures (300 s).
Puisqu'il y a 20 phases, le coût de l'étape de fusion coûte donc : 20*30 + 20*300 = secondes

Donc au total le tri du fichier prendra un temps aux environs de : 330 + 6600 = **6930 seconde = 1.93 heures** (presque 2h de calcul)

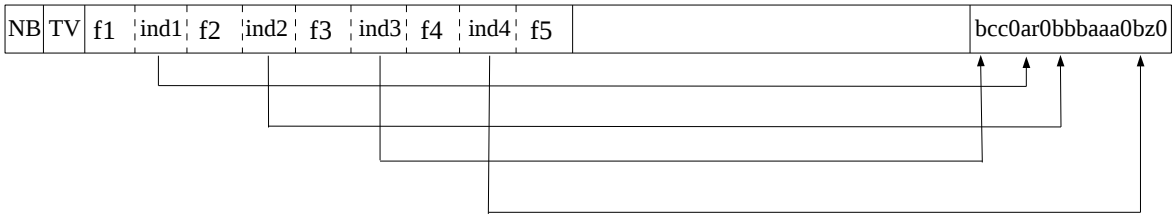
Pour comparer avec d'autres configuration, donnons rapidement (juste pour information) le temps pris avec un nombre de buffers plus grand :
Avec 10 buffers le temps sera aux environs de 38 min
Avec 100 buffers le temps sera aux environs de 22 min
Avec 1000 buffers le temps sera aux environs de 16 min
… etc

5. *Considérons la structure (slotted page) d'un bloc pour un nœud interne d'un B+ arbre préfixé :*
- Le bloc est de taille b octets (tableau de b positions)
- Les 2 premiers octets représentent le nombre de séparateurs dans le nœud.
- Les 2 prochains octets contiennent la taille de l'espace vide restant dans le bloc.
- A partir du 5e octet, on trouve la liste des fils et des indices vers les séparateurs :
$fils_1$, Ind_1 , $fils_2$, Ind_2 , $fils_3$, …. Ind_k , $fils_{k+1}$
- Les séparateurs sont des chaînes de caractères terminées par l'octet 0 ('\0') et sont placés à l'autre extrémité du bloc selon l'ordre d'insertion (le plus récent est le plus à gauche).
- L'espace vide du bloc se trouve donc entre le dernier fils et le début du dernier séparateur à avoir été inséré.
Une structure similaire a été présentée en cours. La principale différence avec celle-ci est que la séquence des séparateurs (à l'extrémité droite du bloc) n'est plus ordonnée selon les valeurs. L'ordre des valeurs est par contre maintenu, indirectement, à travers les indices placés entre les différents fils du nœud.

Tous les entiers (c-a-d le nombre de séparateur, la taille du vide, les fils et les indices) sont représentés sous forme d'une suite d'octets selon un codage particulier (Little Endian, Big Endian ...etc). Chaque indice occupe 2 octets et chaque fils occupe 4 octets.
Pour pouvoir manipuler ces différents entiers de manière transparente au format de codage utilisé, nous supposerons l'existence d'un modèle pour l'accès à ces informations :

recuperer_entier(buf, j , n) : entier
Cette fonction retourne la valeur de l'entier stocké dans le buffer buf sur n octets à partir de la position j.
ecrire_entier(buf, j, x, n)
Cette procédure stocke dans le buffer buf, la valeur de l'entier x sur n octets, à partir de la position j.

a) Dessinez de manière approximative, le contenu d'un bloc interne de taille b (b=60 octets) formé par 4 séparateurs insérés dans cet ordre : 'bz', 'bbbaaa' , 'ar' , 'bcc'
La liste des fils et indices sera représentée symboliquement par : f1 Ind1 f2 Ind2 f3 Ind3 f4 Ind4 f5
Quelles sont les valeurs des différents indices (Ind1, Ind2, Ind3 et Ind4) et celle de la taille du vide ?



Ind1 = **48** (47 si le premier indice est 0)
Ind2 = **51** (50 si le premier indice est 0)
Ind3 = **44** (43 si le premier indice est 0)
Ind4 = **58** (57 si le premier indice est 0)
TV (Taille du Vide) = **11**

b) Donnez un algorithme permettant de rechercher efficacement un séparateur donné à l'intérieur du buffer courant. Le résultat de la recherche doit être comme suit :

trouv : un booléen indiquant l'existence ou non du séparateur dans le buffer

j : le numéro de l'indice associé au séparateur (s'il existe) ou à celui qui le suit (sinon)

La recherche **doit alors être dichotomique** :

```
Rech( sep:chaine ; var trouv:bool , var j:entier )
bi = 1 ;
bs = recuperer_entier( buf, 1, 2 ) ;
trouv = faux ;
TQ ( bi ≤ bs et Non trouv )
        j = (bi+bs) div 2 ;
        ind = recuperer_entier( buf , 6*j + 1 , 2 ) ;
        // récuperer la chaine à la position ind …
        ch = '' ;
        TQ (buf[ind] != '\0' )    ch = ch + buf[ind] ;   ind++   FTQ ;
        Si ( sep == ch ) trouv = vrai
        Sinon
                Si ( sep < ch )   bs = j – 1 Sinon bi = j+1 Fsi
        Fsi
FTQ
```

c) En déduire un algorithme permettant l'insertion d'une paire (séparateur, fils-droit) dans le buffer courant.

```
Inserer( sep:chaine , n:entier , fd:entier )
// insertion du séparateur sep de taille n octets avec comme fils-droit fd dans buf
// On commence par rechercher la position logique dans le nœud ...
Rech( sep , trouv , j )
Si ( Non  trouv )
        // On récupère la taille du vide pour voir s'il y a assez de place pour l'insertion
        tv = recuperer_entier(buf, 3 , 2) ;
        Si ( tv ≤ n + 1 + 6 )       // l'espace occupé sera n+1 pour le séparateur et 6 pour le fd et l'indice
                // insertion de sep dans la séquence des séparateurs à l'extrémité droite du buffer …
                nb = recuperer_entier( buf, 1 , 2 ) ;       // le nombre de séparateurs déjà présent
                pos = 8*(nb+1) + tv – n ;                   // position à partir de laquelle on va insérer sep
                Pour k = 1 , n
                        buf[ pos + k – 1 ] = sep[ k ] ;
                FP ;
                buf[ pos + n ] = '\0' ;
                // insertion de la paire <ind , fd > dans la liste des fils et indices (par décalages vers la droite) ...
                k = 6*nb + 8  ;                             // dernière position du dernier fils
                m = nb ;
                TQ ( m  > j )
                        Pour cpt = 1, 6                     // décalage de 6 octets vers la droite :
                                buf[ k+6 ] = buf[ k ] ;     // d'une paire <fils et ind >
                                k-- ;
                        FP ;
                        m--                                 // répéter les décalages de paires (fils et ind)
                FTQ ;                                       // jusqu'à atteindre la paire numéro j
                // insertion de la nouvelle paire : l'indice du nouveau sep et son fils-droit
                ecrire_entier( buf , k , pos , 2 ) ;        // pos = l'indice du nouveau séparateur
                ecrire_entier( buf , k+2, fd , 4 ) ;        // fd = le nouveau fils-droit
                // incrémenter le champ NB
                ecrire_entier( buf , 1 , nb+1 , 2 ) ;
                // m-a-j du champ 'taille du vide'
                ecrire_entier( buf , 3 , tv - n - 7 , 2 ) ;
        Fsi // ( tv ≤ n + 1 + 6 )
Fsi // (Non trouv)
```