Corrigé du Contrôle Final - SFSD / ESI 2023-2024

Barème : (3+2+4) + (4+2+2) + (2+1)

Q 1 : Soit F un fichier de données de type B-arbre d'ordre 5. Le bloc d'entête contient les informations suivantes :
- le numéro du bloc racine
- le nombre d'enregistrements dans le fichier
- le nombre de blocs alloués au fichier
- la hauteur de l'arbre

Les enregistrements sont des entiers.

1a) Donner la déclaration des structures de données pour la manipulation de ce fichier ainsi que l'***algorithme*** implémentant l'opération d'ouverture du fichier (dans les 2 modes 'ancien' et 'nouveau')

Algorithme (pseudo algorithme en langage naturel)
ou programme C avec syntaxe très flexible

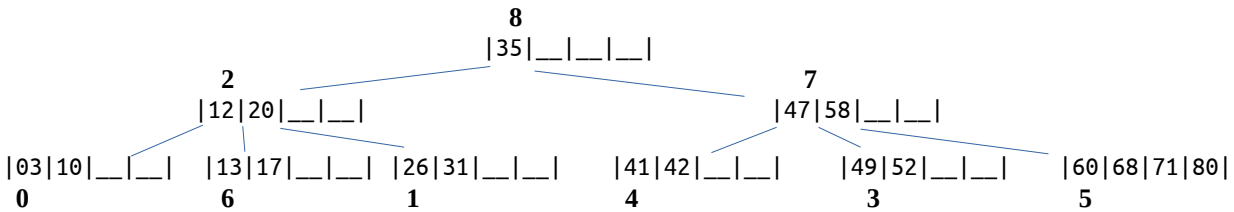
```
Tbloc = structure        // le type des buffer (et des blocs)
        val : tableau[4] d'entiers
        fils : tableau[5] d'entiers
        degré : entier   // ou alors le nombre de valeurs NB:entier
fin
F : fichier de Tbloc Buffer buf
                Entete( rac :entier , NbEnreg :entier , Nblocs :entier , hauteur :entier )

Ouvrir( F, nom , mode )
SI ( mode == 'A' )
   ouverture du fichier existant
   lire l'entete en mémoire centrale
SINON  // mode == 'N'
   créer un nouveau fichier
   Aff_entete( F , rac , -1 )
   Aff_entete( F , NbEnreg , 0 )
   Aff_entete( F , Nbloc , 0 )
   Aff_entete( F , hauteur , -1 )  // ou 0
FSI
```

1b) Choisir aléatoirement 19 valeurs de votre choix entre 0 et 99 pour former une liste non ordonnée de valeurs. Ecrire les valeurs de cette liste et dessinez l'arbre obtenu après l'insertion de toutes ces valeurs (dans l'ordre de leurs positions dans la liste). Montrer aussi les caractéristiques obtenues après l'insertion des 19 valeurs.

Soit 19 valeurs aléatoires :
17, 20, 47, 68, 10, 31, 42, 58, 13, 26, 41, 60, 12, 35, 49, 52, 3, 80, 71

L'insertion de ces valeurs dans cet ordre, donne le B-arbre suivant : (les valeurs soulignées ont provoqué des éclatements)



Suite à ces insertions, l'entête contiendra alors les valeurs suivantes :
racine : 8 , NbEnreg : 19 , Nblocs : 9 , Hauteur : 2

1c) Ecrire un ***algorithme*** récursif qui vérifie si toutes les feuilles de l'arbre se trouvent bien à une hauteur h donnée.

```
VerifHauteur( r : entier , h:entier ) : booléen
// retourne vrai si toutes les feuilles se trouvent à une profondeur h
var      // variables locales
   buf : Tbloc
   i : entier
   t : booléen

SI ( r <> -1 )
   LireDir(F , r , buf )
   SI ( buf.fils[1] == -1 )
        t ← ( h == 0 )
   SINON
        t ← vrai ; i ← 1
        TQ ( i ≤ buf.degré  &&  t == vrai )
                t ← t  &&  VerifHauteur( buf.fils[ i ] , h-1 )
                i++
        FTQ
   FSI
   retourner t
FSI
retourner faux
```

Q 2 : Soient E et F deux ensembles volumineux d'entiers. Chaque ensemble est représenté par un fichier ordonné TOF où la capacité maximale de chaque bloc est de 100 entiers. Comme E et F sont des ensembles, il n'y a donc pas de valeurs en doubles (ni dans E, ni dans F).
2a) Donnez un ***algorithme*** le plus efficace possible, permettant de calculer l'intersection de ces deux ensembles (E ∩ F) dans un fichier résultat R en utilisant 3 buffers en mémoire centrale.

L'intersection se calcule comme une équi-jointure.
Comme les 2 fichiers E et F sont déjà triés, l'algorithme le plus rapide est donc 'tri-fusion' (sans la partie qui fait le tri)

```
// le résultat de l'intersection sera dans un nouveau fichier R
Ouvrir(E , … , 'A') ; Ouvrir(F, … 'A') ; Ouvrir(R,… 'N')
i1 ← i2 ← i3 ← 1
j1 ← j2 ← j3 ← 1

N1 ← Entete(E,1)      // le nombre de blocs formant le fichier E
N2 ← Entete(F,1)      // le nombre de blocs formant le fichier F

fin1 ← ( i1 ≤ N1 ) ;   fin2 ← ( i2 ≤ N2 )
SI (non fin1 && non fin2) LireDir(E, 1, buf1) ; LireDir(F,1, buf2) FSI
```

```
TQ ( non fin1  &&  non fin2 )
   SI ( buf1.tab[ j1 ] == buf2.tab[ j2 ] )
        // mettre l'enreg de E ou de F dans le résultat R …
        buf3.tab[ j3 ] ← buf1.tab[ j1 ]          // ou buf2.tab[ j2 ]
        j3 ++
        SI ( j3 > b )     // b : capacité max des blocs
              buf3.NB ← b ; EcrireDir( R, i3, buf3 ); i3 ++ ; j3 ← 1
        FSI
        // Avancer Dans E…
        j1++
        SI (j1 > buf1.NB )
              SI (i1 < N1 ) i1 ++ ; LireDir(E,i1,buf1) ; j1← 1 SINON fin1 ← vrai FSI
        FSI
        // Avancer Dans F…
        j2++
        SI (j2 > buf1.NB )
              SI (i2 < N2 ) i2 ++ ; LireDir(F,i2,buf2) ; j2← 1 SINON fin2 ← vrai FSI
        FSI
   SINON
        SI ( buf1.tab[ j1 ] < buf2.tab[ j2 ] )
              // Avancer Dans E…
              j1++
              SI (j1 > buf1.NB )
                    SI (i1 < N1 ) i1 ++ ; LireDir(E,i1,buf1) ; j1← 1 SINON fin1 ← vrai FSI
              FSI
        SINON
              // Avancer Dans F…
              j2 ++
              SI (j2 > buf1.NB )
                    SI (i2 < N2 ) i2 ++ ; LireDir(F,i2,buf2) ; j2← 1 SINON fin2 ← vrai FSI
              FSI
        FSI
   FSI
FTQ
// écriture du dernier bloc de R (s'il n'est pas vide) …
SI ( j3 > 1 )
   buf3.NB ← j3 – 1
   EcrireDir( R, i3, buf3 )
   Aff_entete( R , 1 , i3 )
SINON
   Aff_Entete( R , 1 , i3 – 1 )
FSI
Fermer( E )
Fermer( F )
Fermer( R )
```

2b) Dans quel(s) cas le coût de cette opération d'intersection serait moins d'une minute, sachant que la lecture d'un bloc physique coûte 20 µs et l'écriture d'un bloc physique coûte 100 µs et que les fichiers E et F sont formés par 100 000 000 d'enregistrements chacun. (1µs = 10^{-6}s)

Quand les 2 fichiers en entrée sont ordonnés, l'opération de l'intersection devient semblable à celle de la fusion. Le coût en E/S est donc (en pire cas) : N1 + N2 + α
avec : N1 le nombre de blocs dans E
N2 le nombre de blocs dans F
α le nombre de blocs dans fichier résultat R
Le nombre total de lectures de blocs est : N1 + N2
Le nombre total d'écritures de blocs est : α
Le temps d'exécution total (en secondes) est donc : $\mathbf{20*10^{-6}\ (N1 + N2) + 100*10^{-6}\ \alpha}$
Dans notre cas, on a N1 = N2 = N = 100 000 000 / 100 = 10^6 blocs
Le nombre de lectures total (en pire cas) est donc : 2N (lorsque les 2 fichiers sont parcourus entièrement).

Le nombre de lectures total peut être plus petit que 2N si l'un des deux fichier n'est pas parcouru entièrement (par exemple, à partir d'une certaine position dans E, l'enregistrement courant devient plus grand que tous les enregistrements de F qui restent à parcourir, donc le fichier F atteint la fin en premier et la boucle principale s'arrête)
Dans le cas extrême, le 1er bloc d'un des 2 fichiers (par ex. E) contient des enregistrement plus grands que tous les enregistrements de l'autre fichier (par ex. F), dans ce cas le nombre total de lecture sera 1 + N

Le nombre total d'écritures de bloc dépend du nombre d'enregistrements dans le fichier résultat, donc il dépend du nombre de valeurs dans E qui existent aussi dans F (ou inversement)
Dans le pire cas, toutes les valeurs de E existent dans F (ou inversement). Dans ce cas le fichier résultat R contiendra 100 000 000 d'enregistrements (donc R utilisera 10^6 blocs)
Dans le cas le plus favorable, (le moins coûteux) le résultat sera vide (0 blocs)
Ainsi, le temps d'exécution total *t* (en secondes) de l'intersection de E et F varie entre :
$N * 20 *10^{-6} + 0$ et $2N * 20*10^{-6} + N * 100*10^{-6}$
comme $N = 10^6$ on a alors :
$t \in [\ 10^6 * 20 *10^{-6} + 0\ ,\ 2 * 10^6 * 20*10^{-6} + 10^6 * 100*10^{-6}\ ]$
$t \in [\ 20\ ,\ 140\ ]$

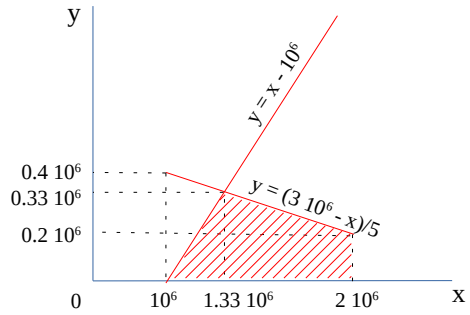
⇒ Trouvons dans quels cas on aura $t \leq 60$ secondes

Posons *x* le nombre total de lectures de blocs et *y* le nombre total d'écritures de blocs réalisées durant l'opération de l'intersection. (donc $N < x \leq 2N$ et $0 \leq y \leq N$)
On peut alors écrire : $20\ x + 100\ y \leq 60*10^6$ c-a-d : $\boldsymbol{x + 5\ y \leq 3*10^6}$
Sachant que le nombre total de lectures *x* doit vérifier : $10^6 < x \leq 2*10^6$, le nombre d'écritures total *y* ne doit donc pas dépasser $(3*10^6 - x) / 5$ pour $x \in\ ]\ 10^6\ ,\ 2*\ 10^6\ ]$

Dans le cas où le parcours de E (ou de F) n'était pas complet, on sait que le fichier résultat R ne pourra pas contenir un nombre d'enregistrements plus grand que le nombre d'enregistrements réellement parcourus dans E (ou F).

Donc si x représente le nombre total de blocs lus (dans E et dans F) alors le nombre de blocs maximal formant le résultat R sera : $y \leq (x - N)$

En résumé, pour $N = 10^6$, y doit vérifier les 2 contraintes suivantes en même temps :
$y \leq (3*10^6 - x) / 5$ et $y \leq (x - 10^6)$
c-a-d : $y \leq min \{ (3*10^6 - x) / 5 , (x - 10^6) \}$
Les valeurs de x et y vérifiant ces contraintes sont représentées dans la zone hachuré (en rouge) du tracé ci-dessous :



Par exemple, si le nombre de lectures total est 2N (c-a-d les 2 fichiers E et F ont été parcourus entièrement, donc $2*10^6$ lectures de blocs) alors pour avoir un temps d'exécution ≤ 60s il faudrait que le nombre d'écritures dans le fichier résultat $y \leq 0.2*10^6$ c-a-d R doit contenir un nombre de valeurs qui ne dépasse pas **20 000 000**, soit un maximum de **20 %** du nombre de valeur dans E (ou dans F)
Si par contre le nombre total de lectures x est compris entre $1.3333\ 10^6$ et $2\ 10^6$, alors le nombre total d'écritures y doit être au maximum = $(3\ 10^6 - x)/5$ pour que le temps d'exécution ne dépasse pas 60s
Et enfin si le nombre total de lectures est compris en 10^6 et $1.3333\ 10^6$, alors le nombre total d'écritures y doit être au maximum = $x - 10^6$

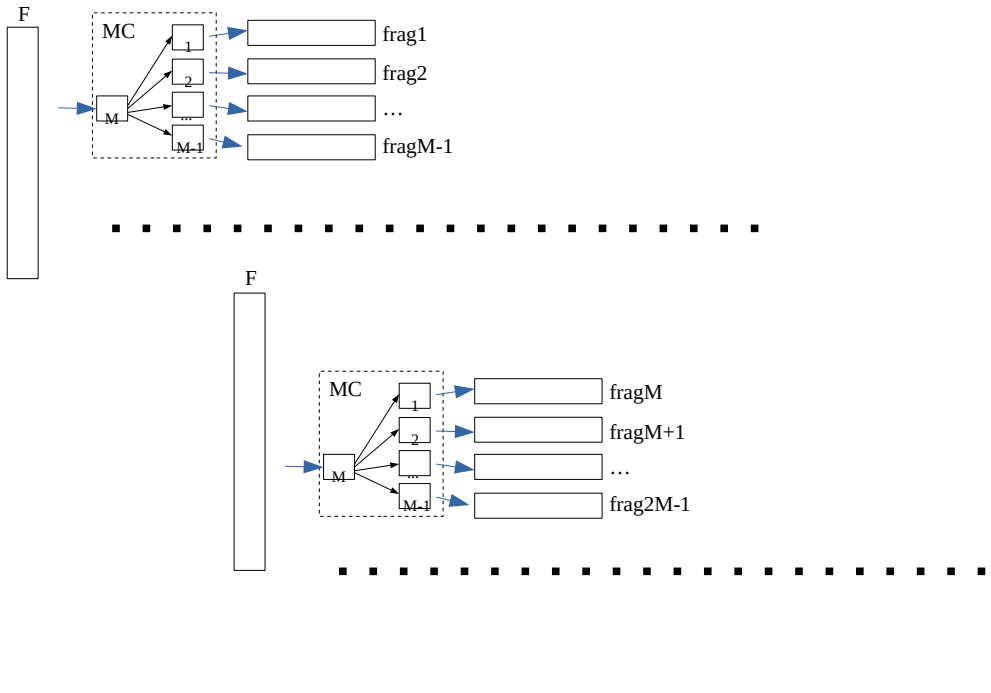
2c) Quel serait le nombre total de lectures de bloc nécessaires pour cette opération d'intersection, si on disposait de 1002 buffers en mémoire centrale ?

Comme l'algorithme utilisé est de type fusion, il ne nécessite que 3 buffers pour avoir un coût optimal. Donc si on dispose de 1002 buffers, le nombre de lectures de blocs ne diminuera pas et reste toujours dans l'intervalle $[10^6 , 2*10^6]$

Si on avait utilisé l'algorithme par boucles-imbriquées dans la question a), alors l'utilisation de 1002 buffers (à la place de 3 buffers) fera diminuer le nombre total de lectures de : N+N*N lectures (c-a-d 10^{12} donc 7,7 mois de temps total des lectures seulement) à N+N*N/1000 lectures (c-a-d 10^9 donc 5h30)

Q 3 : Soit F un fichier de type T$\overline{O}$F formé par N blocs. La capacité maximale des blocs est b enregistrements. Soit h une fonction de hachage uniforme ayant un espace adressable limité à [0 .. K-1] (c-a-d pour tout enregistrement e, on a : $0 \leq h(e) \leq$ K-1)
3a) Ecrire un ***algorithme*** le plus efficace possible, permettant de fragmenter par hachage le fichier F en K fragments. Nous disposons de M buffers en mémoire centrale (avec 2 < M < K)

Pour fragmenter par hachage un fichier F en K fragments et avec un nombre de buffers M < K, on peut procéder en plusieurs étapes, comme suit :



…

La 1ere étape va produire les M-1 premiers fragments
1 buffer est utilisé pour parcourir séquentiellement le fichier F
M-1 buffers sont utilisés pour construire les M-1 premiers fragments (ceux contenant les enregistrements e /que h(e) < M-1

La 2e étape consiste à fragmenter le fichier F en M-1 autres fragments. Ces nouveaux fragments contiendront les enregistrements e de F /que M-1 ≤ h(e) < 2(M-1)

La 3e étape consiste à fragmenter le fichier F en M-1 prochains fragments. Les nouveaux fragments contiendront les enregistrements e de F /que 2(M-1) ≤ h(e) < 3(M-1)

Ces étapes se répètent jusqu'à atteindre le dernier fragment (K).

L'algorithme suivant effectue ces étapes pour fragmenter le fichier F en K fragments en utilisant M-1 buffers à chaque étape.

Soit T un tableau[M] de buffers
Soit F le fichier en entrée (sa 1ere caractéristique est le numéro du dernier bloc)
Soient g[M-1] les fichiers de sorties (les fragments)
Tous les fichiers manipulés ont les mêmes structures et mêmes entêtes

bi ← 0
bs ← M-1
Ouvrir(F , 'fichier_a_fragmenter', 'A')

```
TQ ( bi < K )
    POUR x = bi , bs-1
        Ouvrir( g[x-bi+1] , 'frag'+x , 'N') ;  numB[x-bi+1] ← 1 ; T[x-bi+1].NB ← 0
    FP
    // fragmentation de F en M-1 fragments …
    i ← 1 ; N ← Entete(F,1) // num du dernier bloc de F
    TQ ( i ≤ N )
        LireDir( F , i , T[M] )
        POUR j = 1, buf.NB
            x' ← h( T[M].tab[j] )
            SI ( x' < bs )  x ← x' - bi + 1
                SI ( T[x].NB < b )
                    T[x].NB++ ;  T[x].tab[ T[x].NB ] ← T[M].tab[j]
                SINON
                    EcrireDir( g[x] , numB[x], T[x] )
                    T[x].tab[ 1 ] ← T[M].tab[j] ; T[x].NB ← 1 ; numB[x]++
                FSI
            FSI
        FP
        i++
    FTQ
    // dernières écritures des buffers associés aux fragments traités …
    POUR x = bi , bs-1
        EcrireDir( g[x-bi+1] , numB[x-bi+1] , T[x-bi+1] )
        Aff_entete( g[x-bi+1] , 1 , numB[x-bi+1] )     // dernier bloc du x^ieme fragment
        Fermer( g[x-bi+1] )
    FP
    bi ← bs ;  bs ← min { bs + M-1 , K }
FTQ
Fermer( F )
```

3b) Donnez une formule estimant le coût de cette opération de fragmentation.

Comme la fonction h est supposée uniforme, on peut prendre comme hypothèse que la taille de chaque fragment généré est approximativement de N/K blocs

A chaque étape on génère M-1 fragments (sur les K), donc pour arriver à générer les K fragments, il faudrait un nombre d'étapes au voisinage de : K/(M-1)
A chaque étape, on va lire N blocs du fichier F et on va écrire (M-1)*N/K blocs dans les M-1 nouveaux fragments.
Donc le coût de chaque étape est : N lectures + (M-1)*N/K écritures

Comme il y a en tout K / (M-1) étapes, le nombre total des lectures est N * K/(M-1) et le nombre total des écritures est N (car la somme des tailles des K fragments sera approximativement N)
Le coût total de l'opération de fragmentation est donc :
K*N/(M-1) + N opérations d'E/S