ESI 2010/2011

Corrigé : Examen Semestriel – SFSD – 2e année CPI

1-a) Algo : Intersection de 2 fichiers (F1 et F2) (Version avec recherche dichotomique)

Comme F1 est ordonné, il est plus efficace de parcourir séquentiellement F2 et pour chaque enregistrement, rechercher sa clé dans F1 par dichotomie:

```
Début
        OUVRIR( F1, « nomf1 », 'A' );
        OUVRIR( F2, « nomf2 », 'A' );
        OUVRIR( F3, « nomf3 », 'N' );

        i3 ← 1;  j3 ← 1;        // pour remplir le fichier résultat F3
        i2 ← 1;                 // pour lire les blocs de F2 en séquetiel
        TQ  ( i2 ≤ N2 )
                LireDir( F2, i2, buf2 );
                POUR  j2 = 1 , buf2.NB
                        Rech_dicho_F1( buf2.tab[j2].cle , trouv );
                        SI ( trouv )
                                // inserer l'enreg dans F3
                                SI  ( j3 ≤ b )
                                        // buf3 n'est pas encore plein
                                        buf3.tab[j3] ← buf2.tab[j2];
                                        j3 ← j3 + 1;
                                SINON
                                        // buf3 est plein
                                        buf3.NB ← j3 – 1;
                                        EcrireDir( F3, i3, buf3 );
                                        i3 ← i3 + 1;
                                        buf3.tab[1] ← buf2.tab[j2];
                                        j3 ← 2;
                                FSI
                        FSI
                FP;
                i2 ← i2 + 1
        FTQ
        // Ecriture du dernier buffer dans F3
        buf3.NB ← j3 – 1;
        EcrireDir( F3, i3, buf3 );
        i3 ← i3 + 1;

        // Les caractéristiques de F3
        Aff_entete( F3, 1, i3 – 1 );  // numéro du dernier bloc
        …

        Fermer( F1); Fermer( F2 ); Fermer( F3 );
Fin
```

```
Rech_dicho_F1( en entrée c : typeqlq, en sortie trouv : booléen )
Début
        bi ← 1;
        bs ← N1;
        trouv ← faux;
        stop ← faux;
        TQ  ( bi ≤ bs ET Non Trouv ET Non Stop )
                i1 ← (bi + bs) div 2;
                LireDir( F1, i1, buf1 );
                SI  ( c < buf1.tab[1].cle )
                        bs ← i1 – 1
                SINON
                        SI ( c > buf1.tab[ buf1.NB ].cle )
                                bi ← i1 + 1
                        SINON
                                // recherche interne dans buf1
                                stop ← vrai;
                                inf ← 1;
                                sup ← buf1.NB;
                                TQ ( inf ≤ sup ET Non trouv )
                                        j1 ← (inf + sup) div 2;
                                        SI ( c < buf1.tab[j1].cle )
                                                sup ← j1 – 1
                                        SINON
                                                SI ( c > buf1.tab[j1].cle )
                                                        inf ← j1 + 1
                                                SINON
                                                        trouv ← vrai
                                                FSI
                                        FSI
                                FTQ
                        FSI
                FSI
        FTQ
Fin
```

Version sans recherche dichotomique

```
Début
        OUVRIR( F1, « nomf1 », 'A' );
        OUVRIR( F2, « nomf2 », 'A' );
        OUVRIR( F3, « nomf3 », 'N' );

        i3 ← 1;  j3 ← 1;        // pour remplir le fichier résultat F3
        i2 ← 1;                 // pour lire les blocs de F2 en séquetiel
        TQ  ( i2 ≤ N2 )
                LireDir( F2, i2, buf2 );
                POUR  j2 = 1 , buf2.NB
                        Rech_Seq_F1( buf2.tab[j2].cle , trouv );
```

```
                SI ( trouv )
                        // inserer l'enreg dans F3
                        SI  ( j3 ≤ b )
                                // buf3 n'est pas encore plein
                                buf3.tab[j3] ← buf2.tab[j2];
                                j3 ← j3 + 1;
                        SINON
                                // buf3 est plein
                                buf3.NB ← j3 – 1;
                                EcrireDir( F3, i3, buf3 );
                                i3 ← i3 + 1;
                                buf3.tab[1] ← buf2.tab[j2];
                                j3 ← 2;
                        FSI
                FSI
        FP;
        i2 ← i2 + 1
    FTQ
    // Ecriture du dernier buffer dans F3
    buf3.NB ← j3 – 1;
    EcrireDir( F3, i3, buf3 );
    i3 ← i3 + 1;

    // Les caractéristiques de F3
    Aff_entete( F3, 1, i3 – 1 );  // numéro du dernier bloc
    …
    Fermer( F1); Fermer( F2 ); Fermer( F3 );
Fin

Rech_Seq_F1( en entrée c : typeqlq, en sortie trouv : booléen )
Début
    i1 ← 1;
    stop ← faux;
    TQ  ( i1 ≤ N1 ET Non trouv ET Non stop )
        LireDir( F1, i1, buf1 );
        j1 ← 1;
        TQ ( j1 ≤ buf1.NB ET Non trouv ET Non stop )
            SI ( c = buf1.tab[j1].cle )
                trouv ← vrai
            SINON
                SI ( c < buf1.tab[j1].cle )
                    j1 ← j1 + 1
                SINON
                    stop ← vrai
                FSI
            FSI
        FTQ
        SI ( Non trouv ) i1 ← i1 + 1 FSI
    FTQ
Fin
```

1-b) Coût de l'opération d'intersection

Le coût de l'opération concerne la lectures des blocs de F1 et F2 et l'écriture des blocs de F3
Le fichier F2 est lu séquentiellement une fois (soit N2 lecture)
Pour chaque enregistrement de F2, on fait une recherche dichotomique dans F1 (soit $\log_2$ N1 lectures). Comme il y a b*N2 enregistrements dans F2, le coût des lectures des fichiers F1 et F2 est dans l'ordre de N2 + b*N2*$\log_2$ N1
Le coût de l'écriture dans le fichier F3 dépend du nombre d'enregistrements qui existent dans F1 et F2 en même temps. Dans le pire des cas tous les enregistrements de F2 existent dans F1, ce qui donne un coût maximal = N2 (ou N1) écritures dans F3

Le coût total de l'opération de fusion est alors estimé à : **N2 + b*N2*$\log_2$ N1 + N2**
comme N1 est presque égal à N2 et en posant N=N1=N2
le coût = N (2 + b*$\log_2$ N) → O(N log N)

Dans la version séquentielle, le nombre de fois que l'on va parcourir le fichier F1 est estimé à : b*N2*N1,
Ce qui donne alors un coût = N2 + b*N2*N1 + N2
comme N1 est presque égal à N2 et en posant N=N1=N2
le coût = N (2 + b*N) → O(N^2)

En langage C, on peut déclarer les structures suivantes:

```
// les buffers
struct Tenreg {
    …  // les champs d'un enregistrement.
};

struct Tbuffer {
        struct Tenreg tab[b];
        int NB;
};
```

2-a) Compactage d'un fichier TOF (en une seule passe)

```
Début
        OUVRIR( F, « nomf » , 'A' );
        N ← ENTETE( F, 1 );          // numéro du dernier bloc
        i1 ← 1;  i2 ← 1; j2 ← 1; cpt ← 0;
        TQ ( i1 <= N )
                LireDir( F, i1, buf1 );
                POUR j1 = 1 , buf1.NB
                        SI ( buf1.tab[j1].effacé = faux )
                                // si l'enregistrement n'est pas supprimé logiquement...
                                buf2.tab[j2] ← buf1.tab[j1]; // on le copie dans buf2
                                j2 ← j2 + 1;
                                cpt ← cpt + 1;// pour compter le nb d'enreg insérés
                                SI ( j2 > b )
                                        // si buf2 devient plein, on l'écrit sur disque
                                        buf2.NB ← b;
                                        EcrireDir( F, i2, buf2 );
                                        i2 ← i2 + 1;
                                        j2 ← 1;
                                FSI
                        FSI
                FP;
                i1 ← i1 + 1
        FTQ;
        SI ( j2 > 1 )
                buf2.NB ← j2 - 1;
                EcrireDir( F, i2, buf2 );
                i2 ← i2 + 1
        FSI;
        Aff_Entete( F, 1, i2 – 1 );     // le numéro du dernier bloc utilisé
        Aff_Entete( F, 2, cpt );        // le nombre d'enregistrements insérés
        Aff_Entete( F, 3, 0 );          // le nombre d'enregistrements supprimé logiquement

        Fermer( F );
Fin
```

2-b) Les coûts de l'opération de compactage

Le coût en pire cas

Dans le pire des cas, il faut lire Nblc blocs et écrire Nblc blocs (soit **2 Nblc** opérations d'E/S)
Cela arrive lorsque le fichier est déjà rempli à 100% (donc pratiquement pas de vide) et il y a un nombre très petit d'enregistrements supprimés logiquement (par ex 1, au début du fichier)

Le coût en moyenne

Cela dépendra du nombre d'enregistrements supprimés logiquement
Il faut lire Nblc blocs et écrire X blocs (avec X < N)
Comme les X blocs que l'on va écrire seront remplis à 100%, on aura X = 1 + (nbIns – nbSup) div b
Le coût de l'opération est donc au voisinage de **Nblc + 1 + (nbIns – nbSup) div b** opérations d'E/S.