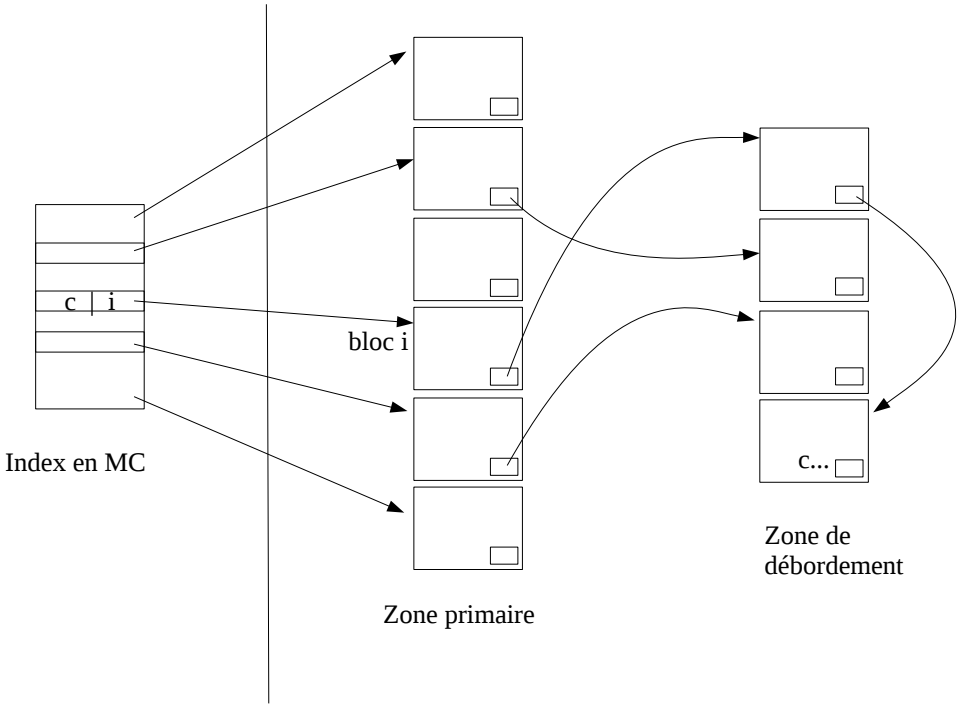
Fichier TOF et index non-dense, avec gestion d'une zone de débordement.



La zone primaire est formée par un fichier ordonné TOF.
Pour gérer les insertions et afin d'éviter les décalages inter-blocs, une zone de débordement est maintenue contenant les enregistrements expulsés des blocs de la zone primaire éventuellement lors des décalages intra-blocs.
Pour pouvoir retrouver les enregistrements déplacés dans la zone de débordement, un lien entre les blocs de la zone primaire est utilisé comme tête de liste de blocs en débordement. Cette liste doit être parcourue séquentiellement pour rechercher ces enregistrements.
L'index en MC contient la plus grande clé associée à chaque bloc de la zone primaire.
L'enregistrement portant cette clé ne se trouve donc pas forcément dans le bloc de la zone primaire référencé par l'entrée de la table d'index. Celui-ci peut en effet, avoir été déplacé en zone de débordement, suite à une insertion dans un bloc primaire déjà plein.

Nous avons donc trois différents fichiers :
F1 : un fichier ordonné contenant les enregistrements en zone primaire.
F2 : un fichier contenant les enregistrements en zone de débordement (dans un ordre quelconque).
F3 : un fichier ordonné de sauvegarde de la table d'index.

Les déclarations des 3 types de blocs :

```
// Pour le fichier F1
Tbloc1 = structure
        tab : tableau[b1] de Tenreg;   // tableau d'enregistrements (format fixe).
        nb : entier ;                  // nombre d'enreg dans tab (entre 0 et b1).
        lien : entier                  // numéro du bloc en débordement associé,
                                       // (-1 si le bloc n'a pas encore débordé).
fin ;
```

```
// Pour le fichier F2 (le même type de bloc que F1)
Tbloc2 = structure
        tab : tableau[b2] de Tenreg;   // tableau d'enregistrements (format fixe).
        nb : entier ;                  // nombre d'enreg dans tab (entre 0 et b2).
        lien : entier                  // numéro du bloc suivant en débordement,
                                       // (-1 si c'est le dernier bloc de la liste).
fin ;

// Pour le fichier F3
Tbloc3 = structure
        tab : tableau[b3] de Tcouple; // tableau d'enregistrements (format fixe).
        nb : entier ;                 // nombre d'enreg dans tab (entre 0 et b3).

Fin ;

Tenreg = structure
        cle : typeqlq ;          // champs clé
        eff : booléen ;          // indicateur d'effacement logique
        ch : typeqlq ;           // autres champs …
fin ;

Tcouple = structure
        c : typeqlq ;            // clé
        adr : entier ;           // numéro du bloc en zone primaire
fin ;
```

Déclaration des fichiers et variables globales :

```
F1 : fichier de Tbloc1 buffer buf1 entete (entier) ;   // caract1 : numéro du dernier bloc
F2 : fichier de Tbloc2 buffer buf2 entete (entier) ;   // caract1 : numéro du dernier bloc
F3 : fichier de Tbloc3 buffer buf1 entete (entier) ;   // caract1 : numéro du dernier bloc

Index : tableau[N] de Tcouple ;       // table d'index
nbI:entier ;                          // nombre d'éléments présents dans la table d'index
```

Les opérations d'accès

1. Chargement_initial :
Cette opération consiste à construire un nouveau fichier à partir d'un certain nombre d'enregistrements donnés. La zone de débordement (le fichier F2) sera créée vide. L'index en MC sera construit en même temps.

2. Réorganisation :
Cette opération consiste à construire un nouveau fichier (F1) à partir des données de l'ancien fichier. La zone de débordement (le nouveau fichier F2) sera créée vide. L'index en MC sera construit en même temps.

3. Recherche :
La recherche d'une clé se fait par dichotomie dans la table d'index et se poursuit séquentiellement dans les fichiers F1 et éventuellement F2 (les enregistrements dans les blocs de F2 ne sont pas forcément ordonnés).
Cette opération retourne comme résultat la position (i,j) de l'enregistrement cherché et un indicateur booléen de débordement (est-ce que i représente un bloc de F1 ou bien de F2).

4. Requête à intervalle :
On trouve le premier enregistrement associé à la borne inférieure, en utilisant le module de recherche et on continue séquentiellement pour ramener les autres.

5. Insertion :
Si la clé de l'enregistrement n'existe pas, le module de recherche retourne l'emplacement (i, j, indicateur de débordement) où il doit être inséré.
Si l'insertion doit se faire dans un bloc de F1 (débordement=FAUX) des décalages intra-bloc sont peut être nécessaires pour maintenir l'ordre à l'intérieur du bloc i. Si le bloc était déjà plein, le dernier enregistrement sera éjecté pour être placé en zone de débordement (gestion de la liste de blocs de débordement de tête buf1.lien).
Si l'insertion doit se faire dans F2 (débordement=VRAI), L'enregistrement sera inséré en fin de liste (i,j). Si le bloc i est déjà plein, un nouveau bloc est alors alloué pour contenir l'enregistrement et un chaînage est mis en place avec le bloc i.
Eventuellement, l'entrée dans la table d'index est mise à jour.

6. Suppression : (logique)
L'enregistrement est localisé avec le module de recherche, puis l'indicateur d'effacement logique (eff) est mis à vrai.

7. Chargement_Index :
La table d'index est chargée à partir d'un fichier (F3).

8. Sauvegarde_Index :
La table d'index est sauvegardée dans un nouveau fichier (F3). On écrase l'ancien fichier F3.

Les algorithmes

Chargement_initial(nomf:chaîne, n:entier, u:reel)
// Construit un nouveau fichier (F1) avec n enregistrements au total et en remplissant les blocs à u %
// La table d'index est aussi construite avec les plus grandes clés de chaque bloc.

```
Debut
        i, j, k :entier ;
        e:Tenreg ;
        Ouvrir(F1, nomf, 'N' );
        i ← 1 ;
        j ← 1 ;
        nbI ← 0 ;                   // pour l'index en MC
```

```
        Pour k = 1,n                    // pour chaque enregistrement e lu à partir de la console
                Lire(e.cle) ;
                Lire(e.ch) ;
                e.eff ← FAUX ;
                SI ( j <= u*b1 )        // insérer e dans le buffer de F1
                        buf1.tab[j] ← e ;
                        j++ ;
                SINON                   // si le buffer est plein à u %
                        // m-a-j de la table d'index ...
                        nbI++ ;
                        Index[nbI] ← ( buf1.tab[j-1].cle , i ) ;
                        // écriture du buffer dans le fichier ...
                        buf1.nb ← j-1 ;
                        buf1.lien ← -1 ;
                        EcrireDir( F1, i, buf1 ) ;
                        // initialisation du nouveau buffer ...
                        i++ ;
                        buf.tab[1] ← e ;
                        j ← 2 ;
                FSI
        FP ;    // (k = 1, n)

        // écriture du dernier buffer …
        nbI++ ;
        Index[ nbI ] ← ( buf1.tab[ j-1 ].cle , i ) ;
        buf1.nb ← j-1 ;
        buf1.lien ← -1 ;
        EcrireDir( F1, i, buf1 ) ;
        Aff_entete( F1, 1, i ) ;
        Fermer( F1 )
Fin
```

Reoragnisation(nomf1:chaîne, nomf2:chaîne, nomf:chaîne, u:reel)

```
// Construit un nouveau fichier F (nomf) avec les enregistrements des anciens fichiers F1 (nomf1)
// et F2 (nomf2). Les blocs du nouveau fichier F (nomf) seront remplis à u % en moyenne.
// La nouvelle table d'index est reconstruite à partir des dernières clés de chaque bloc de F.
Debut
        i, j, i1, j1, i2, j2, k :entier ;
        e:Tenreg ;
        F : fichier de Tbloc1 buffer buf entete (entier) ;      // caract1 : numéro du dernier bloc
        T : Tableau[M] de Tenreg ;
        // T est un tableau utilisé pour trier les enregistrements d'une liste de blocs, avant de les
        // transférer vers le nouveau fichier F. La liste de blocs est formée par un bloc de F1 et
        // quelques blocs de F2 liés au bloc de F1.
        Ouvrir(F, nomf, 'N' );
        Ouvrir(F1, nomf1, 'A' );
        Ouvrir(F2, nomf2, 'A' );
```

```
i ← 1 ; j ← 1 ;                 // pour remplir les blocs du nouveau fichier F
nbI ← 0 ;                       // pour l'index en MC
Pour i1 = 1, Entete(F1, 1)      // Parcours séquentiel de F1
        LireDir(F1, i1, buf1 ) ;
        Pour j1 = 1, buf1.nb
                SI ( buf1.tab[j1].eff = FAUX )
                        SI ( j <= u*b1 )
                                buf.tab[j] ← buf1.tab[j1] ; j++ ;
                        SINON
                                nbI++ ;                         // insérer une nouvelle entrée
                                Index[nbI] ← ( buf.tab[j-1].cle , i ) ; // dans la table d'index
                                buf.nb ← j-1 ;   buf.lien ← -1 ;
                                EcrireDir( F, i, buf ) ;        // écrire le buffer en MS (F)
                                i++ ;
                                buf.tab[1] ← buf1.tab[j1] ;     // initialiser le nouveau buffer
                                j ← 2 ;
                        FSI
                FSI
        FP ;    // (j1 = 1, buf1.nb)

        SI ( buf1.lien ≠ -1 )   // s'il y a des enreg en débordement :
                k ← 0 ;                 // a) remplir T avec ces enreg
                i2 ← buf1.lien ;
                TQ ( i2 ≠ -1 )
                        LireDir( F2, i2, buf2 ) ;
                        Pour j2 = 1, buf2.nb
                                SI ( buf2.tab[j2].eff = FAUX )
                                        k++ ;  T[k] ← buf2.tab[j2]
                                FSI
                        FP ;
                        i2 ← buf2.lien
                FTQ ;  // ( i2 ≠ -1 )

                Trier( T, k ) ;         // b) Trier les k enreg dans T (en MC)

                Pour j2 = 1, k          // c) Insérer les k enreg ainsi triés, dans F
                        SI ( j <= u*b1 )
                                buf.tab[j] ← T[ j2 ] ;   j++ ;
                        SINON
                                nbI++ ;                         // insérer une nouvelle entrée
                                Index[nbI] ← ( buf.tab[j-1].cle , i ) ; // dans la table d'index
                                buf.nb ← j-1 ;    buf.lien ← -1 ;
                                EcrireDir( F, i, buf ) ;   i++ ;
                                buf.tab[1] ← T[ j2 ] ;   j ← 2 ;
                        FSI
                FP      // (j2 = 1, k)
        FSI     // fin de traitement des enreg en débordement (buf1.lien ≠ -1 )
FP      // passer au prochain bloc de F1 ( i1 = 1, Entete(F1, 1) )
```

```
        // écriture du dernier buffer...
        nbI++ ;     Index[ nbI ] ← ( buf.tab[ j-1 ].cle , i ) ;
        buf.nb ← j-1 ;     buf.lien ← -1 ;
        EcrireDir( F, i, buf ) ;
        Aff_Entete(F, 1, i ) ;
        Fermer( F ) ;
        Fermer( F1 ) ;
        Fermer( F2 ) ;
Fin
```

Recherche(c:typeqlq,
var trouv:booléen, var i:entier, var j:entier, var debord:booléen, var k:entier)

```
// Rechercher la cle c dans les fichiers F1 et éventuellement F2 en utilisant l'index en MC.
// On supposera que les 2 fichiers sont déjà ouverts en mode 'A'.
// Les résultats sont un booléen (trouv) qui indique si l'enregistrement de clé c existe, une adresse
// (i,j) représentant un numéro de bloc et une position dans le bloc où se trouve (ou bien, devrait se
// trouvait) l'enregistrement, ainsi qu'un indicateur de débordement, spécifiant si le bloc i appartient
// au fichier F1 (debord=FAUX) ou alors au fichier F2 (debord=VRAI).
// Ce module effectue une recherche dichotomique en MC dans la table d'index. Il retourne dans k,
// l'indice de l'entrée dans la table Index du couple (clé,adr) associé à la clé c.

bi, bs, k : entier ;
continu : booléen ;
Debut
        // On commence par une recherche dichotomique dans la table Index...
        trouv ← FAUX ;       bi ← 1 ;    bs ← nbI ;   continu ← VRAI

        TQ ( Non trouv & bi <= bs )
                k ← (bi+bs) div 2 ;
                SI ( c < Index[k].cle )
                        bs ← k-1
                SINON
                        SI ( c > Index[k].cle )    bi ← k+1
                        SINON                      trouv ← VRAI
                        FSI
                FSI
        FTQ ;
        SI ( Non trouv )
                SI (bi  <= nbI )   k ← bi ;
                SINON    k ← nbI ; continu ← FAUX ;    // cas où c > à la plus grande clé du fichier
                FSI
        FSI ;

        // On continue la recherche dans un bloc de F1 et éventuellement qlq blocs de F2...
        trouv ← FAUX ;       debord ← FAUX ;
        i ← Index[k].adr ;
        LireDir( F1, i, buf1 ) ;
```

```
SI ( c <= buf1.tab[ buf1.nb ].cle )
        // recherche dichotomique dans buf1 …
        bi ← 1 ;  bs ← buf1.nb ;
        TQ ( Non trouv & bi <= bs )
                j ← (bi+bs) div 2 ;
                SI ( c < buf1.tab[j].cle )
                        bs ← j-1
                SINON
                        SI ( c > buf1.tab[j].cle )
                                bi ← j+1
                        SINON
                                trouv ← VRAI
                        FSI
                FSI
        FTQ ;
        SI (Non trouv)
                j ← bi
        SINON // vérifier si l'enregistrement a été supprimé logiquement ….
                SI ( buf1.tab[j].eff ) trouv ← FAUX  FSI
        FSI

SINON  // donc ( c > buf1.tab[ buf1.nb ].cle )
        j ← buf1.nb + 1 ;
        SI ( buf1.lien ≠ -1 )
                // recherche séquentielle en zone de débordement (F2) …
                debord ← VRAI ;  i ← buf1.lien ;  i' ← -1 ;
                SI ( continu )  // si la clé cherchée n'est pas la plus grande clé du fichier …
                   TQ ( Non trouv & i ≠ -1 )
                        LireDir( F2, i, buf2 ) ;
                        j ← 1 ;
                        TQ ( j < buf2.nb  & Non trouv )
                                SI ( c = buf2.tab[j].cle )   trouv ← VRAI
                                SINON  j++
                                FSI
                        FTQ ;
                        SI ( Non trouv )   i' ← i ;  i ← buf2.lien   FSI
                   FTQ ;
                   SI ( trouv )
                        // vérifier si l'enregistrement a été supprimé logiquement ….
                        SI ( buf2.tab[j].eff ) trouv ← FAUX  FSI
                   SINON  // donc (i = -1)
                        i ← i'              // pour garder la trace du dernier bloc visité
                   FSI
                FSI   //  ( continu )
        FSI  // ( buf1.lien ≠ -1 )
FSI   // ( c <= buf1.tab[ buf1.nb ].cle )

Fin     // (Recherche)
```

Requete_intervalle(a:typeqlq, b:typeqlq)
// Affiche en ordre croissant les enregistrements ayant des clés appartenant à [a,b].

Insertion(e:Tenreg, i:entier, j:entier, k:entier, debord:booléen)

```
// Insère l'enregistrement e à sa position (i,j) dans F1 (si debord=FAUX) ou dans F2 (sinon).
// On supposera que les fichiers F1 et F2 sont déjà ouverts.
// L'insertion de e dans F1 se fera par décalages dans le bloc i. Dans ce cas le dernier enreg de i
// risque d'être expulsé et transféré en zone de débordement (fichier F2).
// L'insertion de e dans F2, se fera en fin de liste des blocs de débordement associés au bloc de F1.
// Eventuellement la dernière entrée de l'index est m-à-j (Si e.cle > la dernière clé de Index).
// k représente l'indice de l'entrée dans la table Index du couple (clé,adr) associé à la clé c.

e' : Tenreg ;
m, n : entier ;
Debut
        SI ( debord = FAUX )          // insertion dans F1
                LireDir( F1, i, buf1 ) ;
                // les décalages ...
                e' ← buf1.tab[ buf1.nb ] ;      // le dernier enreg du bloc
                m ← buf1.nb ;
                TQ ( m > j )
                        buf1.tab[ m ] ← buf1.tab[ m-1 ] ;
                        m-- ;
                FTQ ;
                buf1.tab[j] ← e ;
                SI (buf1.nb < b1 )
                        buf1.nb++ ;     buf1.tab[ buf1.nb ] ← e' ;
                        EcrireDir( F1, i,  buf1 )
                SINON // Le bloc i était déjà plein, donc on insère e' en débordement (F2). Pour
                        // éviter le parcours de toute la liste associée à i, on insère en tête de liste.
                        SI ( buf1.lien ≠ -1 )
                                LireDire( F2, buf1.lien, buf2 ) ;
                                SI ( buf2.nb < b2 )
                                        buf2.nb++ ;      buf2.tab[ buf2.nb ] ← e' ;
                                        EcrireDir( F2, buf1.lien, buf2 )
                                SINON // Allouer un nouveau bloc en tête de liste pour mettre e'
                                        n ← AllocBloc( F2 ) ;
                                        buf2.lien ← buf1.lien ;   buf2.tab[1] ← e' ;   buf2.nb = 1 ;
                                        EcrireDir( F2, n, buf2 ) ;
                                        buf1.lien ← n ;                  // la nouvelle tête de liste
                                        EcrireDir(F1, i, buf1 )
                                FSI  // ( buf2.nb < b2 )
                        SINON // donc ( buf1.lien = -1 )
                                n ← AllocBloc( F2 ) ;
                                buf2.lien ← -1 ;   buf2.tab[1] ← e' ;   buf2.nb = 1 ;
                                EcrireDir( F2, n, buf2 ) ;
                                buf1.lien ← n ;                  // la nouvelle tête de liste
                                EcrireDir(F1, i, buf1 )
                        FSI // ( buf1.lien ≠ -1 )
                FSI // (buf1.nb < b1 )
```

```
        SINON // donc insertion dans F2 (debord = VRAI).
                // Comme les insertions en débordement ont été faits en LIFO (comme une pile)
                // On va alors insérer l'enreg e en tête de liste
                LireDir( F1, Index[k].adr, buf1 ) ;     // pour récupérer la tête de liste
                m ← buf1.lien ;
                LireDire( F2, m, buf2 ) ;
                SI ( buf2.nb < b2 )   // Si le bloc de tête n'est pas plein ...
                        buf2.nb++ ;
                        buf2.tab[ buf2.nb ] ← e ;
                        EcrireDir( F2, m, buf2 )
                SINON                   // S'il est plein, Allouer un nouveau bloc en tête de liste
                        n ← AllocBloc( F2 ) ;
                        buf2.lien ← m ;   buf2.tab[1] ← e ;   buf2.nb = 1 ;
                        EcrireDir( F2, n, buf2 ) ;
                        buf1.lien ← n ;                 // la nouvelle tête de liste
                        EcrireDir(F1, Index[k].adr, buf1 )
                FSI  // ( buf2.nb < b2 )

        FSI // (debord = FAUX )

        // Si la clé e.cle de l'enregistrement inséré e est plus grande que la dernière clé de l'index c,
        // il faudrait alors modifier l'index pour contenir e.cle à la place de c.
        SI ( e.cle > Index[k].cle )
                Index[k].cle ← e.cle
        FSI

Fin // (Insertion)
```

Suppression(i:entier, j:entier, debord:booléen)

```
// Supprime logiquement l'enregistrement qui se trouve à la position j du bloc numéro i.
// Si debord=FAUX, le bloc i fait partie du fichier F1 (zone primaire)
// Sinon il en débordement, dans le fichier F2.
// Cette opération consiste juste à mettre le booléen 'eff' à VRAI dans l'enregistrement (i,j)
// On suppose qu'on a déjà réalisé la recherche de la clé et que les 2 fichiers soient ouverts.
…
…
```

Chargement_Index(nomf:chaîne)

```
// Charge la table Index depuis un fichier de sauvegarde (nomf) préalablement construit.
…
…
```

Sauvegarde_Index(nomf:chaîne)

```
// Sauvegarde le contenu de la table Index dans un nouveau fichier de sauvegarde (nomf).
...
...
```