Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

Ecole supérieure en sciences et technologies de l'informatique et du numérique

Sequential structures : Part 2

Presented by : Dr. Daoudi Meroua

Academic year: 2024/2025

Sequential structures

The TOF Method:

It represents the organization of a file viewed as a table (T), ordered (O), with fixed-size records (F).

The search for a record is performed using binary search (fast).

Insertion may cause intra- and inter-block shifts (costly).

Deletion can be done through reverse shifts (physically costly) or simply by using a boolean indicator (logical deletion, much faster).

The initial load operation consists of constructing an ordered file with n initial records, leaving some empty space in each block. This helps minimize the shifts that future insertions might cause.

Periodic reorganization is recommended

TOF : Search Module

File Declaration:

```
Const b = 30; // Maximum Block Capacity (in number of records)

type
Tenreg = structure /
    efface : booleen; // Boolean for Logical Deletion
    cle : typeqlq; // Field Used as Search Key
    champ2 : typeqlq;
    champ3 : typeqlq;
    ...
Fin;
Tbloc = structure
        tab : tableau[1..b] de Tenreg;
        NB : entier; // nombre d'enreg dans tab ( <= b)
Fin;
```

Var F : Fichier de Tbloc Buffer buf Entete (entier, entier);

TOF : Search Module

Input: The key (c) to search for and the external file name (nomfich).

Output: The boolean Trouv, the block number (i) containing the key, and the displacement (j).

```
Rech( c:typeqlq; nomfich:chaine; var Trouv:bool; var i,j:entier )
var
bi, bs, inf, sup : entier;
trouv, stop : booleen;
DEBUT
Ouvrir( F, nomfich, 'A' );
bs ← entete( F,1 ); // la borne sup (le num du dernier bloc de F)
bi ← 1; // la borne inf (le num du premier bloc de F)
Trouv ← faux; stop ← faux; j ← 1;
```

TOF : Search Module

```
TQ ( bi ≤ bs et Non Trouv et Non stop ) // External Search
    i ← (bi + bs) div 2; // le bloc du milieu entre bi et bs
    LireDir( F, i, buf );
    SI ( c ≥ buf.tab[1].cle et c ≤ buf.tab[buf.NB].cle )
    // Binary Search Within the Block (in the variable buf)
    inf ← 1; sup ← buf.NB;
    TQ (inf ≤ sup et Non Trouv) // Internal Search
        j ← (inf + sup) div 2;
        SI (c = buf.tab[j].cle)
            Trouv ← vrai
        SINON
            SI (c < buf.tab[j].cle)
                sup ← j-1
            SINON
                inf ← j+1
            FSI
        FSI
```

TOF : Search Module

```
SI ( inf > sup )
    j ← inf FSI // fin de la recherche interne. // j : The Position Where c Should Be
Located in buf.tab
  stop ← vrai
SINON // non ( c ≥ buf.tab[1].cle et c ≤ buf.tab[buf.NB].cle )
  SI ( c < buf.tab[1].cle )
      bs ← i-1
 SINON // donc c > buf.tab[buf.NB].cle
      bi ← i+1
FSI
FSI
FTQ
SI ( bi > bs )
i ← bi ; j ← 1
FSI //
End of External Search
// i : Block Number Where c Should Be Located
fermer( F )
FIN
```

TOF : insertion Module

```
Inserer( e:Tenreg; nomfich:chaine )
var
trouv : booleen;
i,j,k : entier;
e,x : Tenreg;
DEBUT

// We start by searching for the key e.cle using the previous module to locate the
position (i,j) where e should be inserted in the file.

Rech( e.cle, nomfich, trouv, i, j );

SI ( Non trouv ) // e must be inserted in block i at position j.

    Ouvrir( F,nomfich, 'A'); // en décalant les enreg j, j+1, j+2, ... vers le bas
    continu ← vrai;
```

TOF : insertion Module

```
TQ ( continu et i ≤ entete(F,1) )
  LireDir( F, i, buf );

// Before making the shifts, save the last record in a variable x...

 x ← buf.tab[buf.NB];

// Shift within buf...

 k ← buf.NB;
 TQ k > j
    buf.tab[k] ← buf.tab[k-1];
    k ← k-1
 FTQ

// Insert e at position j in buf...

buf.tab[j] ← e;
```

TOF : insertion Module

If buf is not full, place x at position NB+1 and stop...

```
SI ( buf.NB < b )
    buf.NB ← buf.NB+1;
    buf.tab[buf.NB] ← x;
    EcrireDir( F, i, buf );
    continu ← faux;
SINON // If buf is full, x must be inserted in block i+1 at position 1...
    EcrireDir( F, i, buf );
    i ← i+1;
    j ← 1;
    e ← x; //  This will be done in the next iteration. (the insertion of e)
 FSI // not ( buf.NB < b )
FTQ
```

TOF : insertion Module

```
// If we exceed the end of the file, we add a new block containing a single record
e.

SI i > entete( F, 1 )
   buf.tab[1] ← e;
   buf.NB ← 1;

   EcrireDir( F, i, buf ); // It is enough to write a new block at this location.

   Aff-entete( F, 1, i ); // We save the number of the last block in header 1.
FSI

Aff-entete( F, 2 , entete(F,2)+1 ); // We increment the insertion counter.

Fermer( F );
FSI
FIN
```

TOF : deletion Module

```
Suppression( c:typeqlq; nomfich:chaine )
var
trouv : booleen;
i,j : entier;
DEBUT
// We start by searching for the key c to locate the position (i,j) of the record to be deleted.
Rech( c, nomfich, trouv, i, j );
// Then we logically delete the record
SI ( trouv )
    Ouvrir( F,nomfich, 'A’);
    LireDir( F, i, buf );
    buf.tab[j].efface ← VRAI;
    EcrireDir( F, i, buf );
    Fermer( F )
FSI
FIN
```

Merge of Two TOF

```
Fusion (nom1,nom2, nom3: chaine)
var
F1 : Fichier de Tbloc Buffer buf1 Entete( entier, entier);
F2 : Fichier de Tbloc Buffer buf2 Entete( entier, entier);
F3 : Fichier de Tbloc Buffer buf3 Entete( entier, entier);
i1, i2, i3 : entier;
j1, j2, j3 : entier;
continu : booleen;
e, e1, e2 : Tenreg;
buf : Tbloc;
i, j, indic : entier;
```

Merge of Two TOF

```
Debut
ouvrir(F1, nom1, 'A' );
ouvrir(F2, nom2, 'A' );
ouvrir(F3, nom3, 'N' );
i1←1; i2←1; i3 ←1; // The block numbers of F1, F2, and F3.
j1←1; j2←1; j3 ←1; // The record numbers in buf1, buf2, and buf3.
LireDir(F1, 1, buf1) ;
LireDir(F2, 1, buf2) ;
continu ← vrai ;
```

Merge of Two TOF

```
TQ ( continu ) // While not end of file in F1 and F2 do...
SI ( j1 ≤ buf1.NB et j2 ≤ buf2.NB ) // Choose the smallest record from buf1 and buf2.
    e1←buf1.tab[j1];
    e2 ← buf2.tab[j2] ;
SI ( e1.cle ≤ e2.cle )
     e ← e1; j1← j1 + 1;
SINON
    e ← e2; j2← j2 + 1;
FSI // and place it in buf3.
SI ( j3 ≤ b )
   buf3.tab[j3] ← e; j3 ← j3 + 1
SINON
  buf3.NB ← j3 – 1;
  EcrireDir(F3, i3, buf3 );
  i3 ← i3 + 1;
  buf3.tab[1] ← e;
  J3 ← 2;
FSI
```

Merge of Two TOF

```
SINON // not ( j1 ≤ buf1.NB et j2 ≤ buf2.NB )
 // If all the records of one of the blocks (buf1 or buf2) have been processed, move
on to the next one.
SI ( j1 > buf1.NB )
SI ( i1 < entete(F1, 1) )
    i1 ← i1 + 1;
    LireDir( F1, i1, buf1 ) ;
    j1 ← 1
SINON // ( so i1 ≥ entete(F1, 1) )
   continu ← faux ;
   i ← i2; //For the continuation of the TQ.
  j ← j2;
   N ← entete(F2,1) ;
   buf ← buf2 ;
   Indic ← 2
FSI // ( i1 < entete(F1, 1) )
```

Merge of Two TOF

```
SINON //  ( j2 > buf2.NB )
SI ( i2 < entete(F2, 1) )
    i2 ← i2 + 1;
    LireDir( F2, i2, buf2 );
    j2 ← 1
SINON // ( i2 ≥ entete(F2, 1) )
  continu ← faux;
   i ← i1; // For the continuation of the TQ.
   j ← j1;
   N ← entete(F1,1);
   buf ← buf1;
  Indic ← 1;
FSI // ( i2 < entete(F2, 1) )
FSI // ( j1 > buf1.NB )
FSI // ( j1 ≤ buf1.NB et j2 ≤ buf2.NB )
FTQ
```

Merge of Two TOF

```
// continue to copy the records of a single file(i,j,buf) in F3
continu ← vrai;
TQ ( continu ) // while not end of file F1 or F2 do
    SI ( j ≤ buf.NB )
      SI ( j3 ≤ b )
          buf3.tab[j3] ← buf.tab[j]; j3 ← j3 + 1
      SINON
          buf3.NB ← j3 – 1;
          EcrireDir(F3, i3, buf3 );
          i3 ← i3 + 1;
          buf3.tab[1] ← buf.tab[j];
          J3 ← 2;
       FSI ; // ( j3 ≤ b )
       j ← j + 1
```

Merge of Two TOF

```
SINON // not ( j ≤ buf.NB )
SI ( i ≤ N )
    i ← i + 1;
  SI ( Indic = 1 )
    LireDir( F1, i, buf )
  SINON
    LireDir( F2, i, buf )
  FSI ;
  j ← 1
SINON
    continu ← faux
FSI
FSI // ( j ≤ buf.NB )
FTQ ;
```

Merge of Two TOF

```
// The last buffer (buf3) has not yet been written to disk...
buf3.NB ← j3 – 1 ;
EcrireDir( F3 , i3, buf3 ) ;
Aff-entete( F3, 1, i3) ; // blocks number of F3
Aff-entete( F3, 2, entete(F1,1) + entete(F2,1) ) ; // records number of
F3
Fermer( F1 ) ;
Fermer( F2 ) ;
Fermer( F3 ) ;
Fin
```