**Corrigé type SFSD**

**Exercice 1 :**

Tbloc= structure

tab : Tableau[1..b] d’enrg

nb :entier

Fin

F fichier de Tbloc buffer buf, buf 2 entete(entier)

buf contient dejà le bloc i+2 \*/

       q:= ( 2 \* b + buf.nb ) / 3 ;

       r := mod ( 2 \* b + buf.nb, 3 ) ;

       Mise à jour bloc I+2

      /\* Le bloc I+2 contiendra q+r articles\*/

      {décalage dans le bloc  I+ 2}

       j := 0 ;

       pour k := buf.nb , 1 , - 1

buf.tab[q+r-j]=buf.tab[k]

            j := j + 1 ;

       Fp;

      /\* prendre du bloc I+1    (b-q)\*2  articles \*/

       lireDir ( F , buf2 , I + 1 ) ;

       j:= 1 ;

       pour k:=  2 \* q - b + 1 , b

buf2.tab[j]=buf.tab[k]

            j := j+ 1

       Fp;

buf.nb=q+r

       ecrireDir ( F , buf , I + 2 ) ;

       lireDir ( F , buf , I ) ;

      {Mise à jour bloc I+1}

      {décalage dans le bloc  I+ 1}

       j := 0 ;

       pour k := 2 \* q - b + 1 , - 1

buf2.tab[q-j]=buf2.tab[k]

j:= j+ 1

      Fp ;

      /\* prendre du bloc I    (b-q) articles \*/

       j := 1 ;

       pour k := b , q + 1 , - 1

buf2.tab[j]=buf.tab[k]

            j:= j + 1

       Fp;

buf2.nb=q

       ecrireDir ( F , buf2 , I + 1 ) ;

      {Mise à jour bloc I}

       buf2.Nb=q

       ecrireDir( F , buf , I ) ;

**Remarque: b-(b-q)\*2= 2\*q-b**

**Exercice 2**

**Insérer-élément (Var racine :arbre, clé :entier,i,j :entier)**

Si racine==Nul

N=allouerNoeud()

N.val.clé=clé

N.fg=Null

N.fd=Null

Sinon

Si (clé<N.val)

Insérer-élément(fg, clé)

Sinon

Insérer-élément(fD, clé)

Finsi

Finsi

**Fin**

**créer\_Arbre(Var racine :Arbre)**

ouvrir(F,’nom.txt’,’A’)

N🡨entete(F,1)

Tanque (i<=N) faire

Liredir(F,i,buf)

Pour j allant de 1 à buf.Nb

Insérer-élément (racine,buf.tab[j].clé,i,j )

FinP

i🡨i+1

finTq

**Fin**

**RechercheArbre(racine: Arbre,clé :entier, var trouve :bool)**

Tanque (racine <> nill ) et (non trouve)

Si racine.val.clé= clé

trouve=vrai

Sinon

Si (clé<racine.val.clé)

RechercheArbre(Fg, clé, trouve)

Sinon

RechercheArbre(Fd, clé, trouve)

Finsi

Finsi

FinTq

**Fin**

**insererElm(var racine :arbre, elm :enregistrement)**

trouve=faux

RechercheArbre(racine,elm.clé, trouve)

Si (non trouve) alors

N🡨entete(F,1)

Liredir(F,N,buf)

Si (buf.Nb<b) alors

buf.Nb++

buf.tab[buf.Nb]=elm

Sinon

N=N+1

buf.Nb=1

buf.tab[1]=elm

Affecter-entete(F,1,N)

FinSI

FinSI

Ecriredir(F,N,buf)

Fermer(f)

Insérer-élément (racine, elm.clé,N,buf.nb)

Fin

**Avantages :**

* Structure dynamique
* Les méthodes d’accès (recherche, insertion et suppression) sont plus rapides et faciles

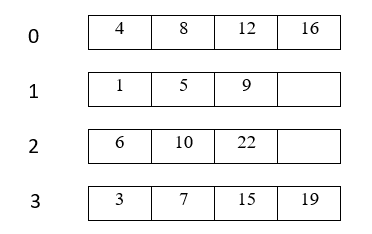
**Inconvénients :**

* Encombrement mémoire causé par les pointeurs
* Risque d’avoir un arbre déséquilibré

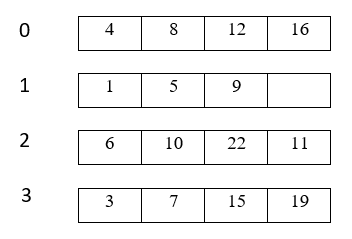
**Exercice 3**

1. **Méthode essai linéaire :**
2. **La fonction d’hachage utilisée est h(x)= x mod N dans ce cas = x mod 4**

* **Insertion de 9 : h(9)=1**



* **Insertion de 11 : h(11)=3**

****

* **Insertion de 33,81,et 29 impossible d’insérer, le fichier est plein**

1. **Les caractéristiques :**

* N : le nombre de blocs formant le fichier
* nbIns : le nombre de données insérées

1. **Module de recherche essai linéaire**

**Rech( entrée : x sorties : trouv, i, j )**

*// on suppose que le fichier F est déjà ouvert*

i ← *h(x)* ; trouv ← faux ; stop ← faux ; N ← Entete( F, 1 )

**TQ** ( Non trouv && Non stop )

**LireDir( F, i, buf )**

j ← 1 // Recherche interne dans le bloc i

**TQ** ( j ≤ buf.NB && Non trouv )

**SI** ( x = buf.tab[ j ].cle ) trouv ← vrai **SINON** j ← j+1 **FSI**

**FTQ**

**SI** ( buf.NB < b ) // Si présence d'une case vide (bloc non plein)

stop ← vrai // Alors fin de la séquence de tests

**SINON** i ← i - 1 ; **SI** ( i < 0 ) i ← N-1 **FSI** // Sinon on continue les tests

**FSI**

**FTQ**

1. **Hachage linéaire :**
2. **Les caractéristiques du fichier :**

* Tour i représente le nombre de fois que le fichier a doublé de taille
* Le pointeur n représente le numéro du bloc qui subira l’opération d’éclatement

1. **Insertion des données :**

**Le tour i**

Les fonctions d’hachage à utiliser :

* (x)= x mod (dans ce cas i=0)
* (x)= x mod

n pointeur vers la case à éclater, initialisé à 0

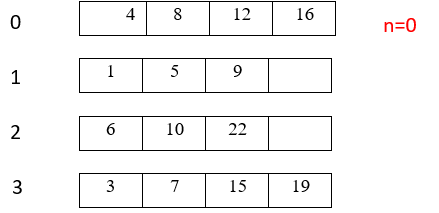
**Dans ce cas tour i=0, donc :**

(x)= x mod = x mod 4

(x)= x mod = x mod 8

n=0

* **Insertion de 9 : h0(9)=1**

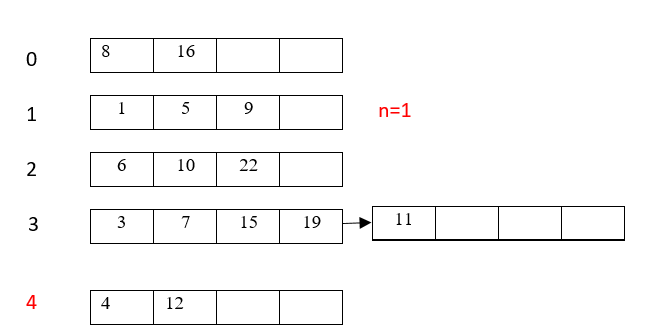
****

* **Insertion de 11 : h0(11)=9**

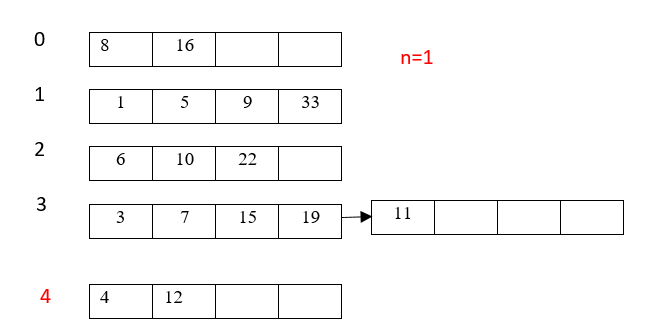
**n=0**

**h0(11)=3 mais le bloc 3 est plein, donc ajouter une zone de débordement pour la case 3 et éclatement de la case 0 (pointé par n) et redistribution des éléments de la case 0, entre la case 0 et la case 4 en utilisant h1 : h1(4)=4 h1(8)=0 h1(12)=4 h1(16)=0**

**Incrémenter n**

****

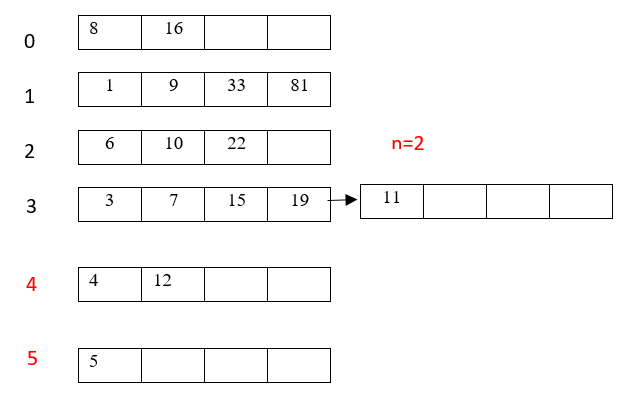
* **Insertion de 33 : h0(33)= 1**

****

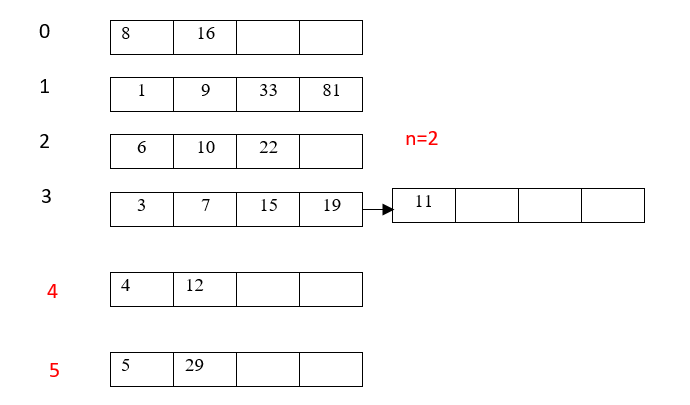
* **Insertion de 81 : h0(81)=1 mais le bloc 1 est plein, aucune zone de débordement est nécessaire, éclater la case 1 et redistribuer ses éléments entre la case 1 et 5 avec la fonction h1**

**h1(1)=1, h1(5)=5, h1(9)=1, h1(33)=1, h1(81)=1**

**Incrémenter n**

****

* **Insertion de 29 : h0(29)=1 mais le bloc 1 est plein et 1<n (n=2) donc h1(29)= 5**

****

1. L’incrémentation du tour i signifie que le fichier a doublé du volume, il est incrémenté lorsque n atteint 2i\*N
2. Dans la méthode statique, lorsque le fichier devient plein on ne peut pas ajouter d’autres éléments (nombre de cases statique), Cependant que, le hachage dynamique linéaire, permet de maintenir des structures de hachage efficaces lorsque il y’a des mises à jour fréquentes au niveau du fichier (insertions ou suppressions fréquentes). Donc on peut ajouter ou supprimer des cases selon les besoins.