Variantes des B-Arbres

Il existe plusieurs variantes de la méthode de base.

Elles peuvent être plus efficaces que la méthode de base dans certains cas.

1) B^+-arbre

→ Séparation du niveau feuille des niveaux internes

2) B^+-arbre préfixé

→ B^+-arbre avec taille minimale des séparateurs (format variable)

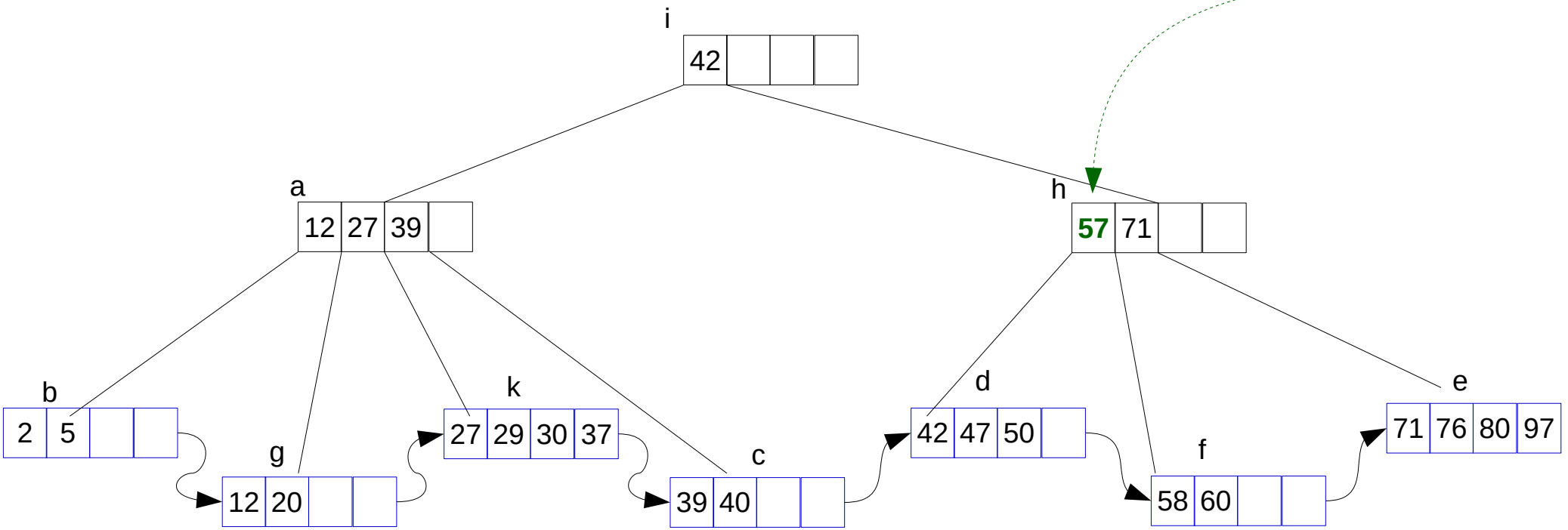
3) B^*-arbre

→ Eclatement amélioré (min 2/3)

1) B$^+$-arbre

- Les feuilles sont chaînées entre elles et contiennent *toutes les valeurs* de l'arbre

- Les nœuds **internes** forment un *index non dense* sur le niveau feuille.
  Ils forment un B-arbre classique.
  Les ***clés** dans les **nœuds internes** peuvent **éventuellement** être **fictives***.
  Le rôle des nœuds internes est uniquement de **guider** la recherche vers le bon nœud feuille.
  Les valeurs (ou clés) dans les nœuds internes sont appelées '**séparateurs**'

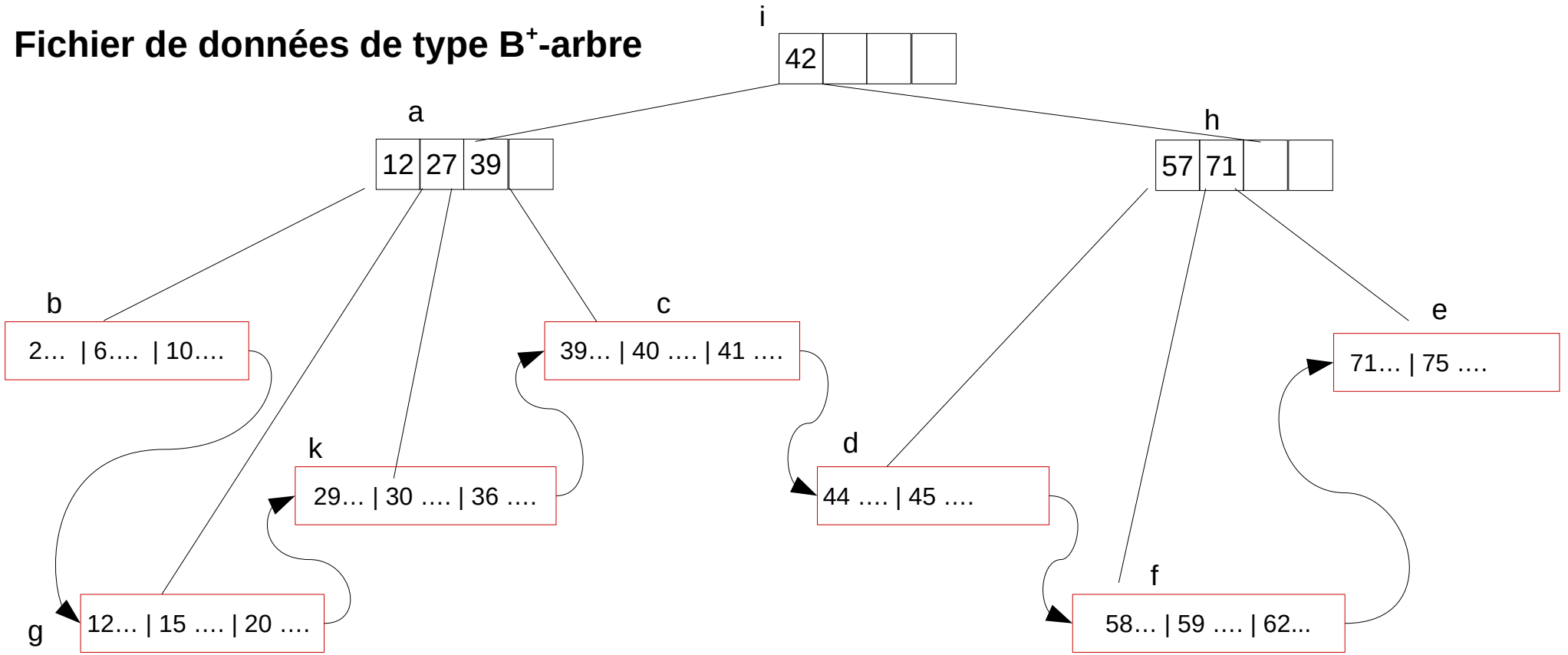


Utilisation des B$^+$-arbres

Si l'arbre représente un **fichier de données**, le niveau feuille contiendra tous les enregistrements (ordonnés). Les nœuds internes contiendront certaines clés (fictives ou non) pour séparer les différents blocs feuilles. (*utilisation courante*)

Si l'arbre est utilisé comme index d'un fichier de données (voir figure), le niveau feuille contiendra les différents couples <clé,adr> formant l'index. Les nœuds internes contiendront certaines clés (fictives ou non) pour séparer les différents blocs feuilles.

Fichier de données de type B$^+$-arbre

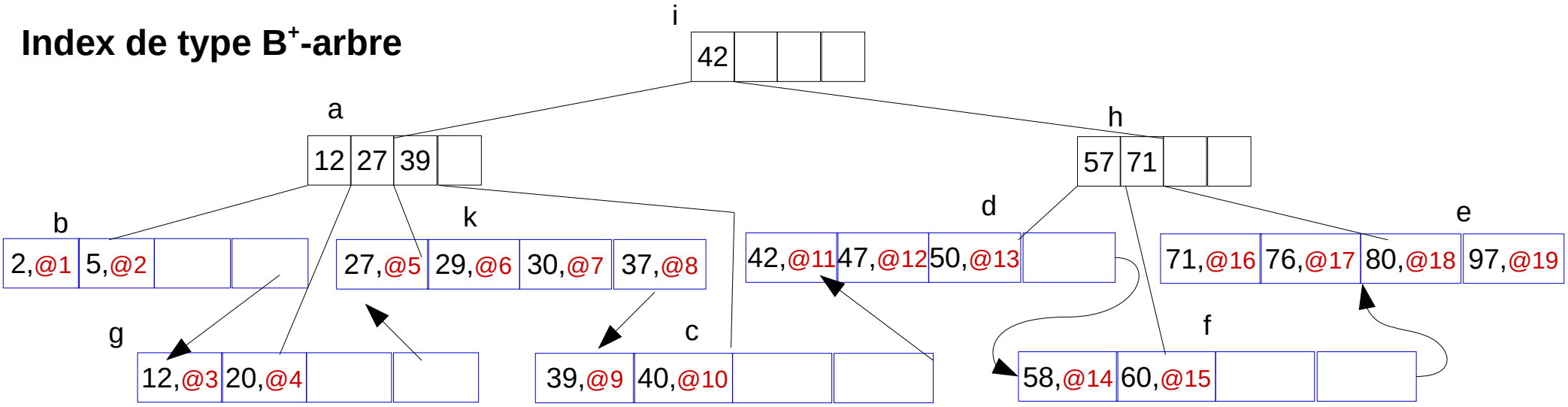


Utilisation des B$^+$-arbres

Si l'arbre représente un fichier de données, le niveau feuille contiendra tous les enregistrements (ordonnés). Les nœuds internes contiendront certaines clés (fictives ou non) pour séparer les différents blocs feuilles. (*utilisation courante*)

Si l'arbre est utilisé comme **index** d'un fichier de données (voir figure), le niveau feuille contiendra les différents couples <clé,adr> formant l'index. Les nœuds internes contiendront certaines clés (fictives ou non) pour séparer les différents blocs feuilles.

Index de type B$^+$-arbre



Fichier de données TOF

@5	@15	@9	@2	@12	@1	@18	@11	@7	@14	@6	@16	@13	@3	@4	@10	@8	@19	@17
27…	60 ….	39 ….	5 ….	47….	2…	80 ….	42 ….	30 ….	58….	29…	71 ….	50 ….	12 ….	….	20…	40 ….	37 ….	97 ….	76 ….

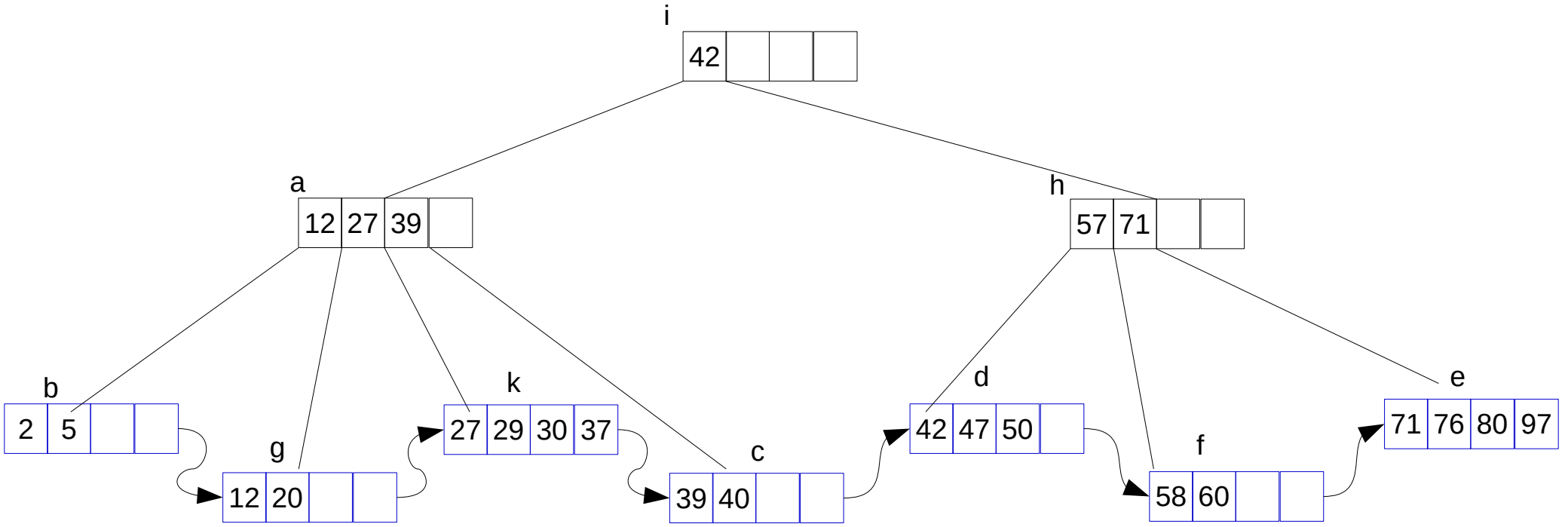
Recherche d'une clé v dans un B$^+$-arbre

Descendre jusqu'aux feuilles en sélectionnant à chaque nœud interne, un des fils du Nœud comme suit :

$Fils_1$ si $\mathbf{v} < Val_1$

$Fils_j$ si $Val_{j-1} \leq \mathbf{v} < Val_j$ (pour : $2 \leq j \leq$ degré-1)

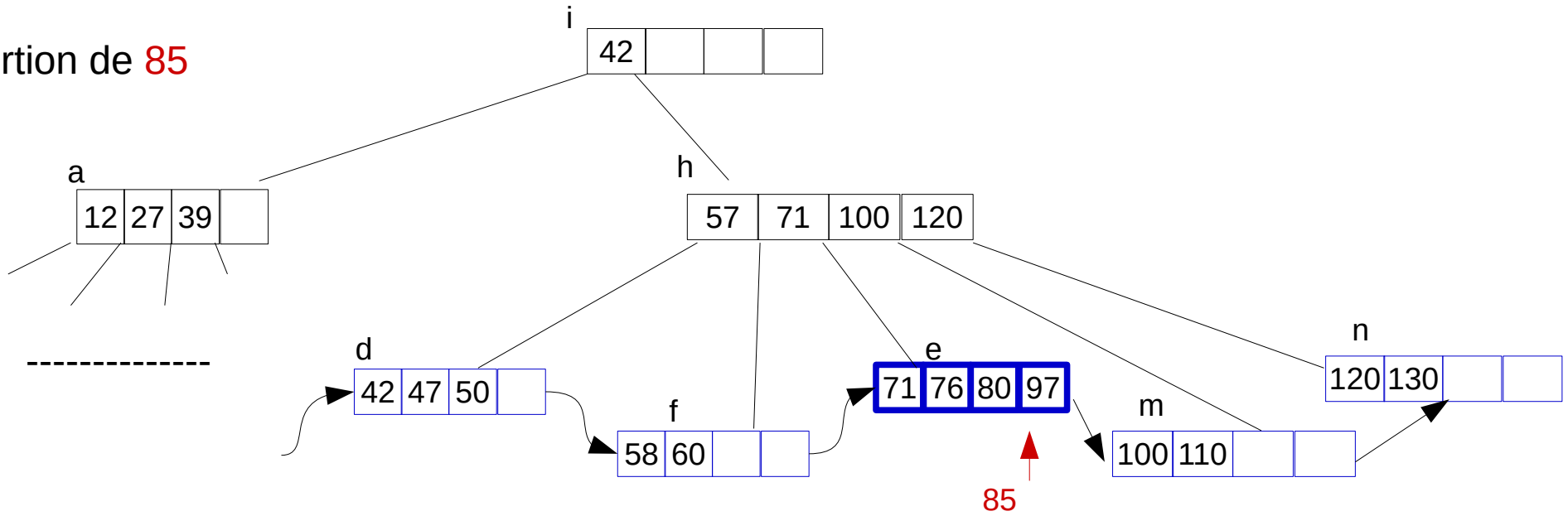
$Fils_{degré}$ si $\mathbf{v} \geq Val_{degré-1}$



Insertion d'une nouvelle clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille ***p*** devant contenir ***v***
2- Si ***p*** n'est pas plein, insertion par décalages internes à ***p*** ; aller à 4 (Fin)
3- *// **p** est donc un bloc plein ...*
Si ***p*** est une feuille, alors *Eclatement_Feuille* :
1ere moitié dans ***p***
2e moitié (incluant la val du milieu) dans un nouveau bloc feuille ***q***
v ← val_milieu (1ere valeur de ***q***) ; ***fd*** ← ***q*** ; ***p*** ← pere(***p***) ; aller à 2
Sinon, Eclatement non feuille (classique) :
1ere moitié dans ***p***
dernière moitié (sans la val du milieu) dans un nouveau bloc interne ***q***
v ← val_milieu ; ***fd*** ← ***q*** ; ***p*** ← pere(***p***) ; aller à 2
4- stop

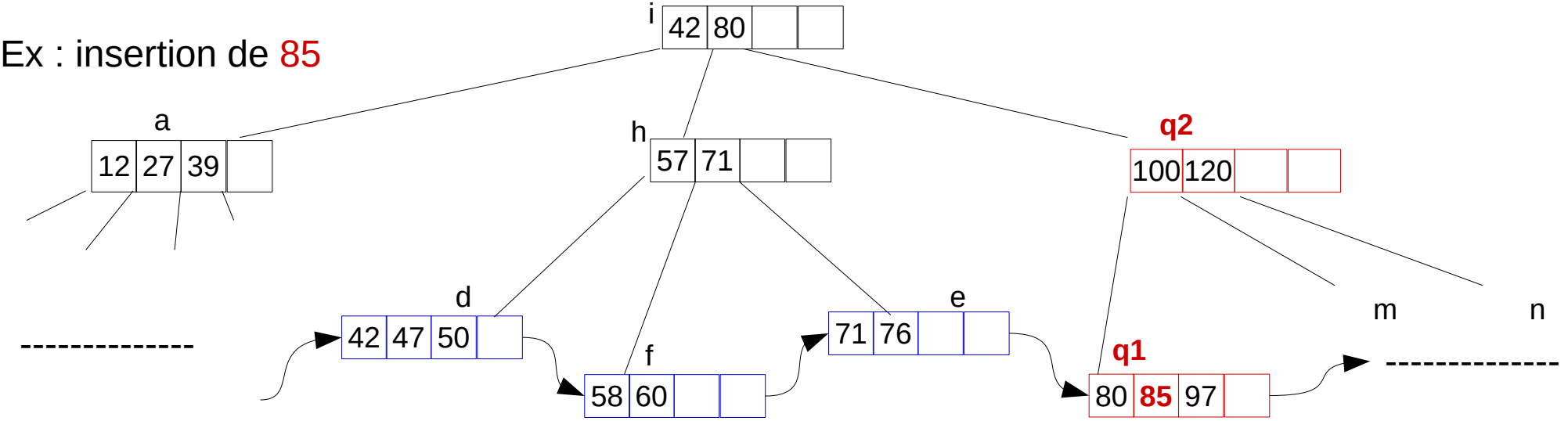
Ex : insertion de 85



Insertion d'une nouvelle clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille ***p*** devant contenir ***v***
2- Si ***p*** n'est pas plein, insertion par décalages internes à ***p*** ; aller à 4 (Fin)
3- *// **p** est donc un bloc plein ...*
Si ***p*** est une feuille, alors *Eclatement_Feuille* :
1ere moitié dans ***p***
2e moitié (incluant la val du milieu) dans un nouveau bloc feuille ***q***
v ← val_milieu (1ere valeur de ***q***) ; ***fd*** ← ***q*** ; ***p*** ← pere(***p***) ; aller à 2
Sinon, Eclatement non feuille (classique) :
1ere moitié dans ***p***
dernière moitié (sans la val du milieu) dans un nouveau bloc interne ***q***
v ← val_milieu ; ***fd*** ← ***q*** ; ***p*** ← pere(***p***) ; aller à 2
4- stop

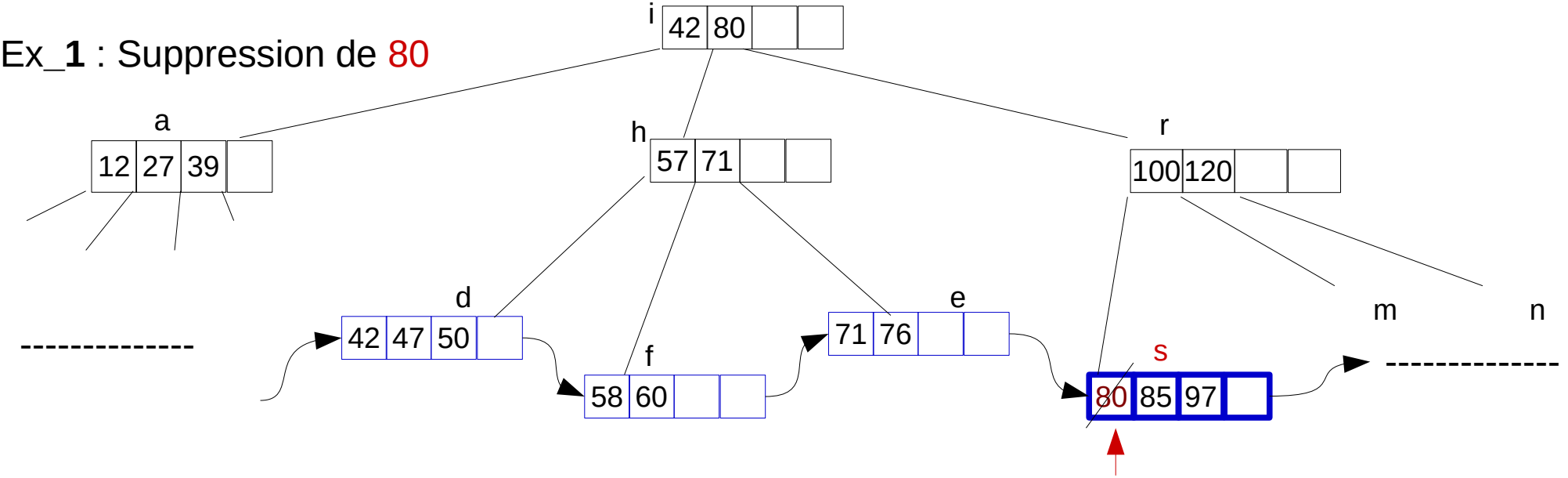
Ex : insertion de 85



Suppression d'une clé v dans un B+-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors *Redist_Feuille(p,q)* Sinon *Redist_interne(p,q)* Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors *Fusion_Feuille(p,q)* Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

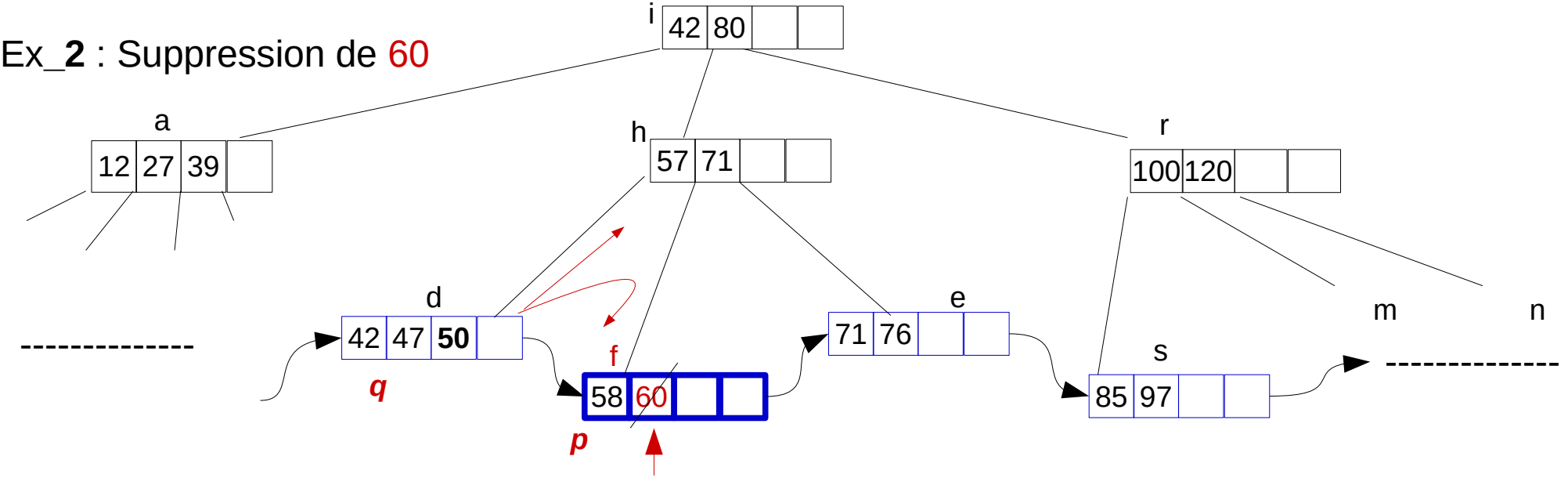
Ex_**1** : Suppression de 80



Suppression d'une clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors ***Redist_Feuille(p,q)*** Sinon *Redist_interne(p,q)* Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors *Fusion_Feuille(p,q)* Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

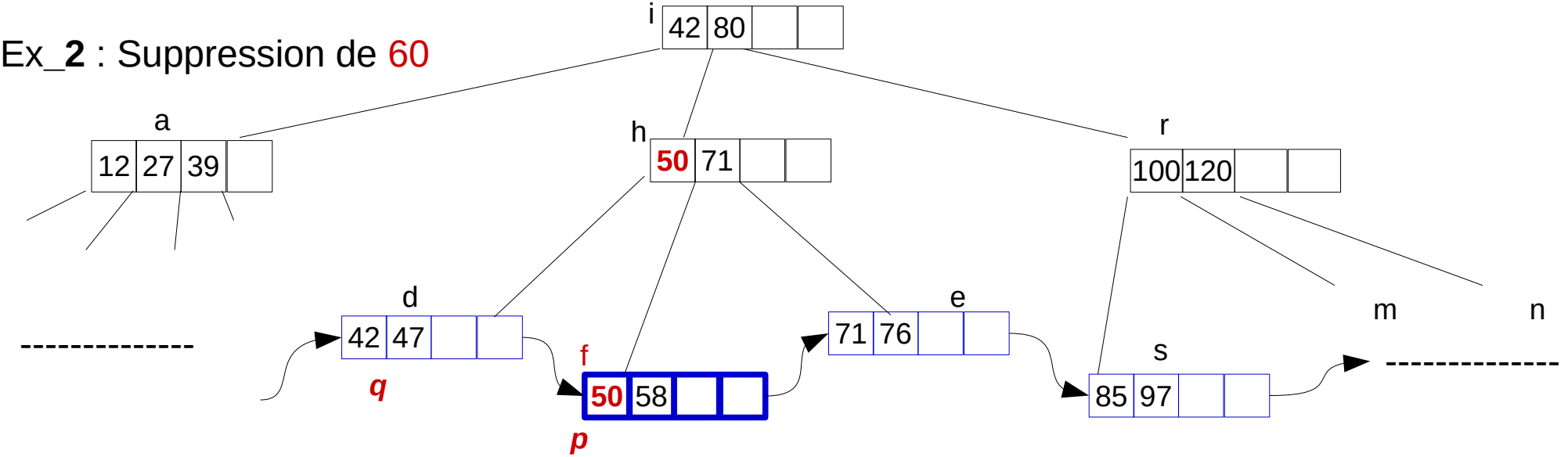
Ex_**2** : Suppression de 60



Suppression d'une clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors ***Redist_Feuille(p,q)*** Sinon *Redist_interne(p,q)* Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors *Fusion_Feuille(p,q)* Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

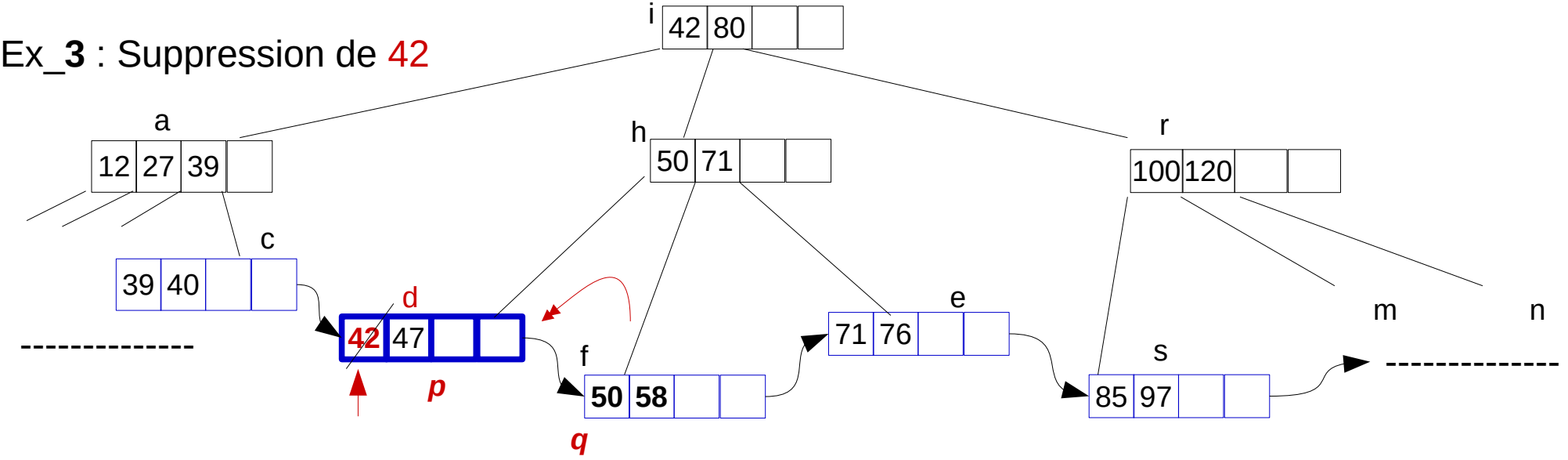
Ex_**2** : Suppression de 60



Suppression d'une clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors *Redist_Feuille(p,q)* Sinon *Redist_interne(p,q)* Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors ***Fusion_Feuille(p,q)*** Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

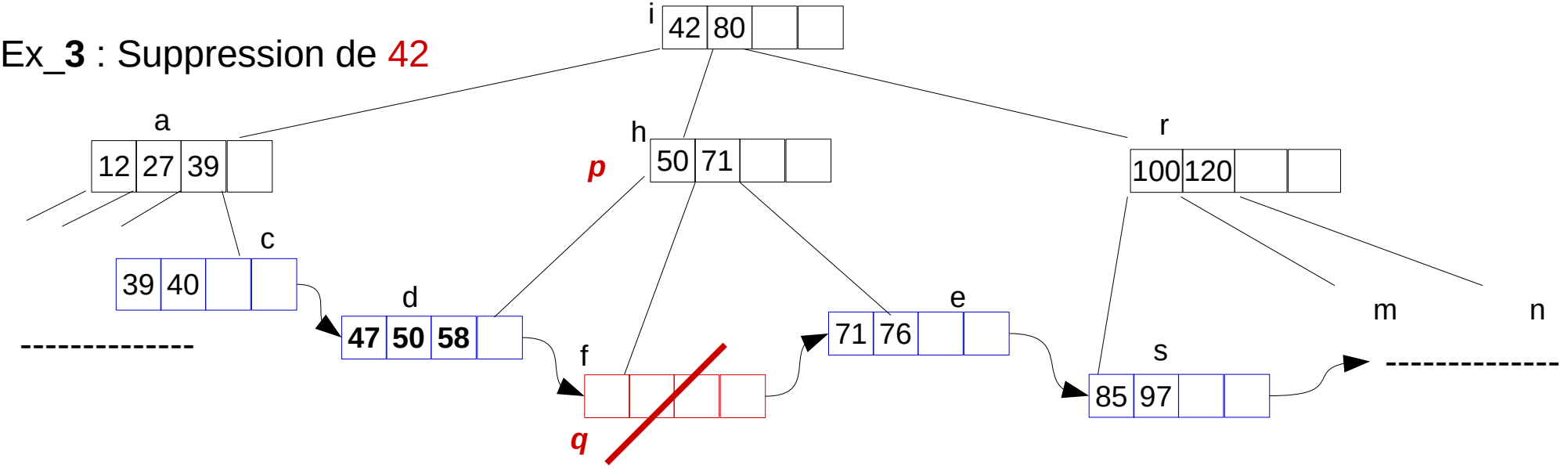
Ex_**3** : Suppression de 42



Suppression d'une clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors *Redist_Feuille(p,q)* Sinon *Redist_interne(p,q)* Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors *Fusion_Feuille(p,q)* Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

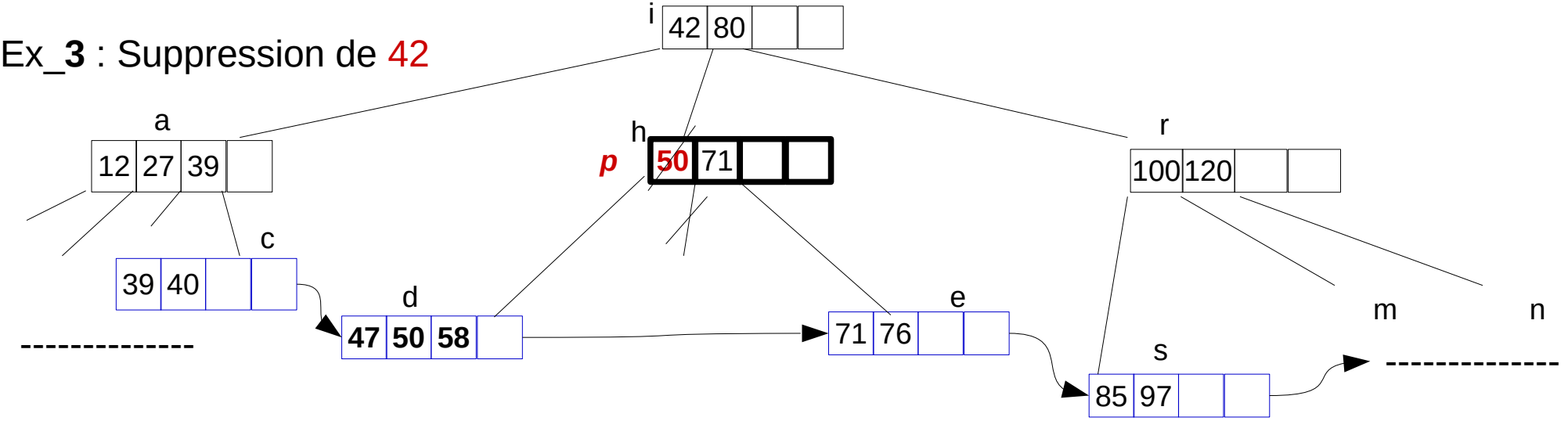
Ex_**3** : Suppression de 42



Suppression d'une clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors *Redist_Feuille(p,q)* Sinon *Redist_interne(p,q)* Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors *Fusion_Feuille(p,q)* Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

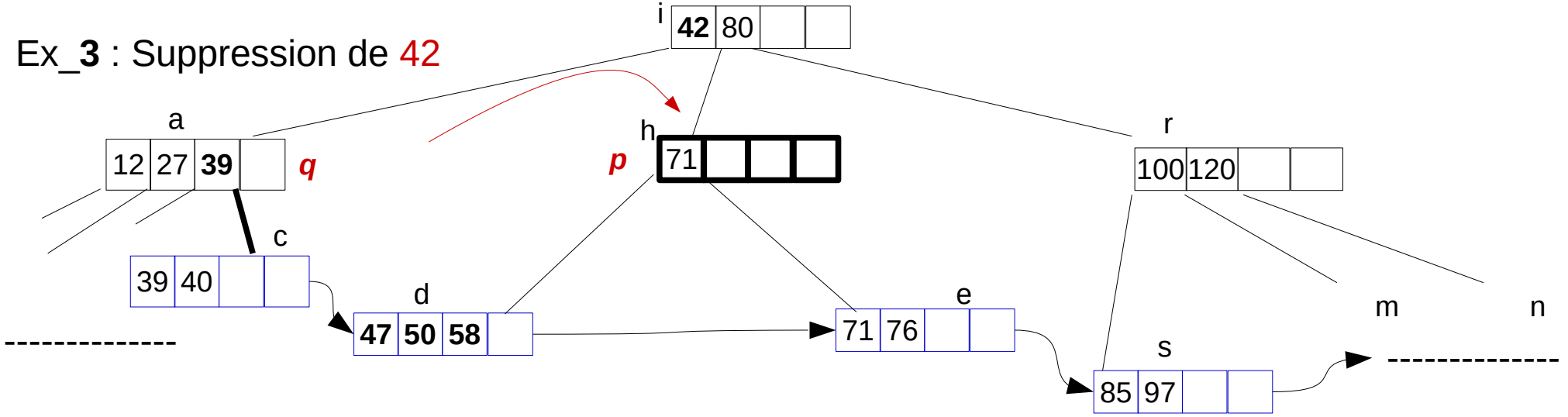
Ex_**3** : Suppression de 42



Suppression d'une clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors *Redist_Feuille(p,q)* Sinon ***Redist_interne(p,q)*** Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors *Fusion_Feuille(p,q)* Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

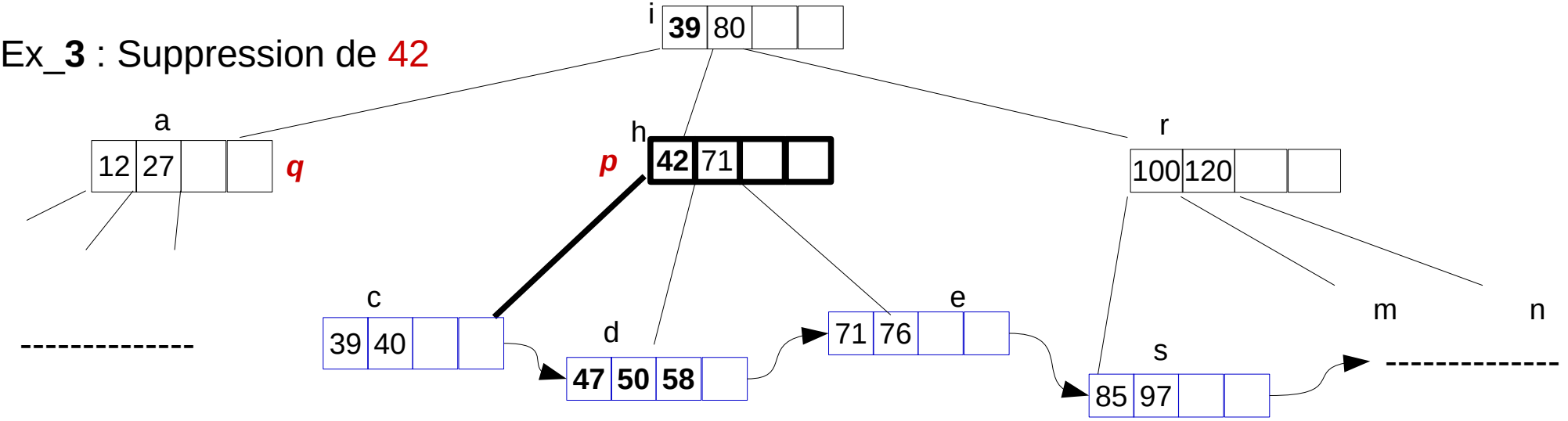
Ex_**3** : Suppression de 42



Suppression d'une clé v dans un B$^+$-arbre

1- Rechercher le bloc feuille p contenant v
2- Suppression de v par décalages internes dans p
3- *TQ (p est sous-chargé)*
SI un des frères q de p peut donner une valeur,
Si p est une feuille alors *Redist_Feuille(p,q)* Sinon ***Redist_interne(p,q)*** Fsi
Sinon, // donc aucun frère de p ne peut donner une valeur ...
Si p est une feuille alors *Fusion_Feuille(p,q)* Sinon *Fusion_interne(p,q)* Fsi
Libérer(q) ; p ← pere(p) ;
v ← val_milieu qui séparait les 2 nœuds fusionnés
Suppression de v (et son fils-droit) par décalages internes dans p
Fsi
FTQ

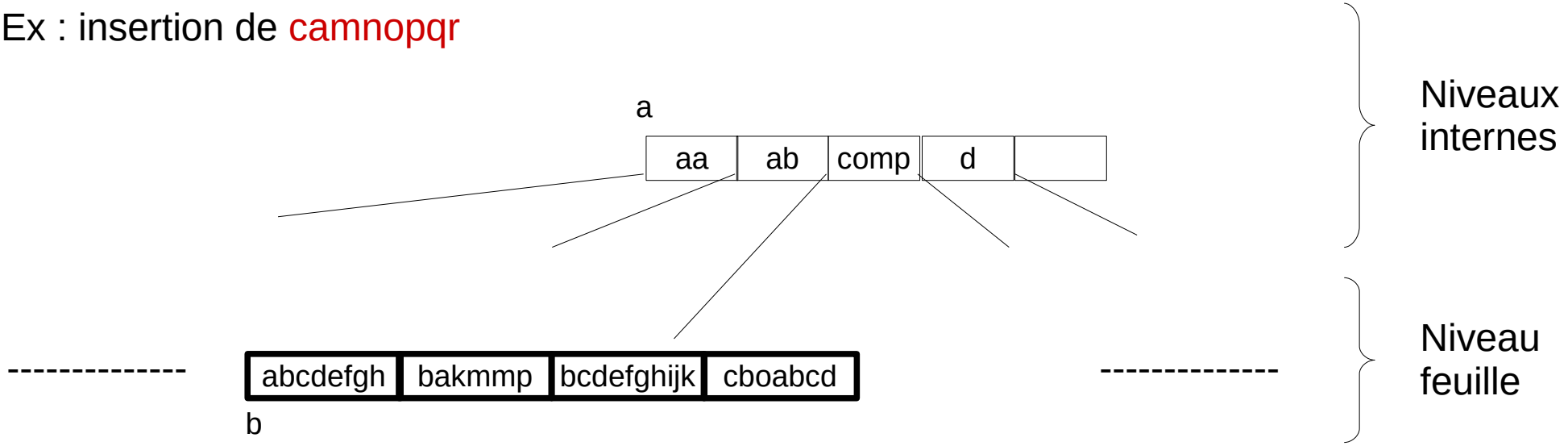
Ex_**3** : Suppression de 42



2) B+-arbres préfixés

Idée principale : Lors d'un éclatement de feuille, le séparateur à promouvoir vers le niveau supérieur est *le plus petit préfixe* de la clé du milieu permettant de séparer le contenu des deux feuilles issues de l'éclatement.

Ex : insertion de camnopqr



La séquence ordonnée = abcdefg bakmnp **bcdefghijk** camnopqr cboabcd

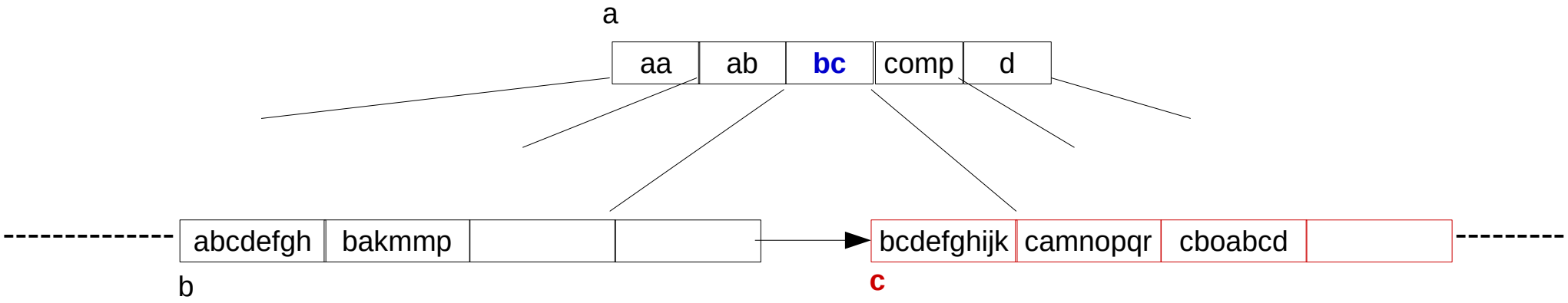
1ere moitié *2e moitié*

La clé du milieu = **bcdefghijk**

Le plus petit préfixe de la clé du milieu pouvant séparer le 2 moitiés est : **bc**

Idée principale : Lors d'un éclatement de feuille, le séparateur à promouvoir vers le niveau supérieur est *le plus petit préfixe* de la clé du milieu permettant de séparer le contenu des deux feuilles issues de l'éclatement.

Insertion de camnopqr
- Eclatement de b ⇒ Allocation de **c** et partage de la séquence ordonnée entre les 2 blocs
- Promotion du séparateur **bc** vers le bloc parent (a)



La séquence ordonnée = abcdefg bakmnp **bcdefghijk** camnopqr cboabcd

1ere moitié *2e moitié*

La clé du milieu = **bcdefghijk** (le plus petit préfixe de la clé du milieu est : **bc)**

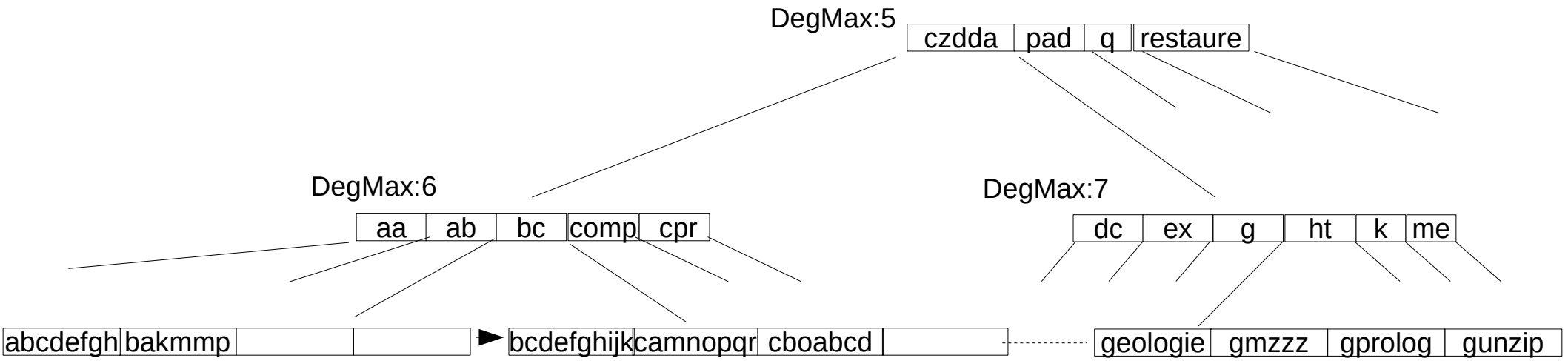
Exemple de calcul du plus petit séparateur

```
// Précondition : cle1 < cle2
// Sorties :        sep : le plus petit préfixe de cle2 > cle1          et taille : sa longueur
void separateur_min( char *cle1, char *cle2, char *sep, int *taille )
{
    *taille = 0 ;

    while ( 1 ) {

            *sep = *cle2;

            sep++;
            (*taille)++ ;

            if ( *cle1  !=  *cle2 ) break;
            if ( *cle2  ==  0 ) break;

            cle1++;
            cle2++;

    }

    *sep = 0;
}
```

Idée principale : Lors d'un éclatement de feuille, le séparateur à promouvoir vers le niveau supérieur est *le plus petit préfixe* de la clé du milieu permettant de séparer le contenu des deux feuilles issues de l'éclatement.

⇒ Maximiser le nombre de séparateurs dans les nœuds internes :
donc cela augmente le **degré moyen** et donc la **hauteur de l'arbre reste faible**

⇒ L'ordre de l'arbre devient variable ! (il n'y a plus de degré maximum fixe) :
Les séparateurs (dans les nœuds internes) sont de **longueur variable**

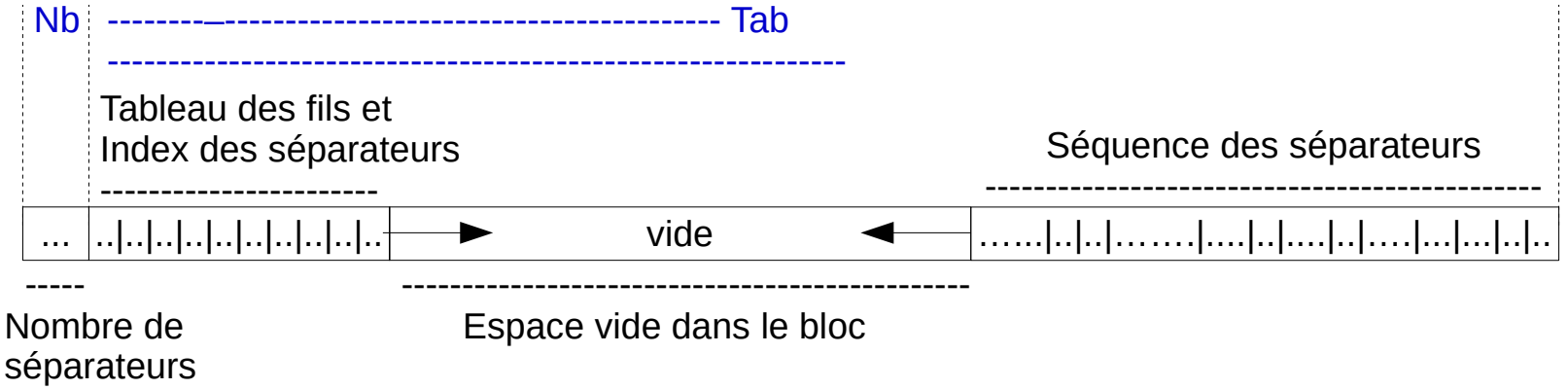


Structure d'un bloc interne :

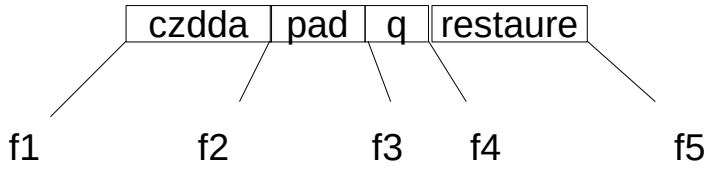
Le bloc doit contenir :

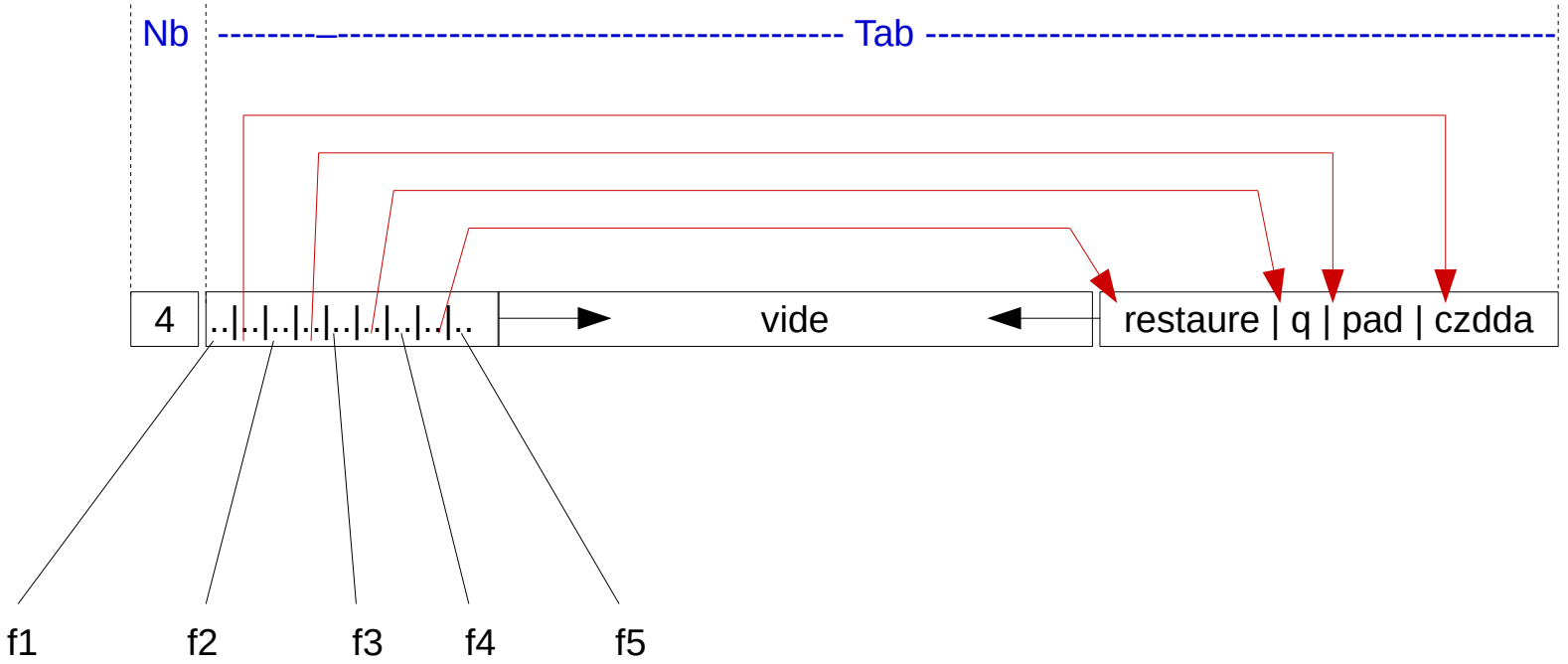
- les séparateurs (avec format variable et en nombre variable)
- les n° des blocs fils (avec format fixe mais en nombre variable = nb de sep + 1)

⇒ Une solution possible :




```
Type Tbloc = Structure {
                Nb : entier ;
                Tab : tableau[b] de caractères ;        // ou octets
             }
```





Tab : |f1|96|f2|93|f3|92|f4|84|f5|……..|RestaureQPadCzdda|

84 92 93 96 100

Implémentation de la fonction d'accès : *Fils(i)*

Conventions (exemple numérique) :

Nb sur **2 octets**

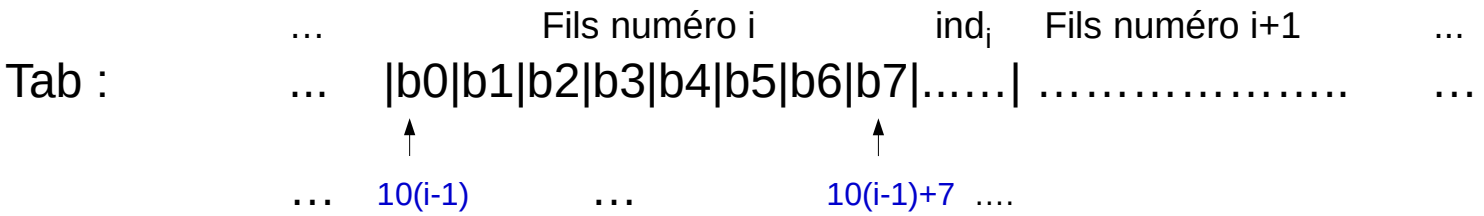
N° de fils sur **8 octets**

Indices des séparateurs sur **2 octets**

Les entiers seront représentés avec le format *little endian* (l'octet de poids faible en premier)

Tab : [f_1 ind_1 f_2 ind_2 f_3 ind_3 … ind_n f_{n+1} sep_n sep_{n-1} …. sep_1]

Pour récupérer le $Fils_i$ il faut récupérer les octets entre les positions 10(i-1) et 10(i-1)+7



Le $Fils_i$ est reconstitué à l'aide de la formule suivante :

$Fils_i = b0 + b1 * 256 + b2 * 256^2 + b3 * 256^3 + … + b7 * 256^7$

Fils(i:entier) : entier

```
P ← 1 ;  S ← 0 ;
Pour k = 0 , 7
        S ← S + buf.Tab[ 10*(i-1)+k ] * P ;
        P ← P * 256
FP ;
Retourner S ;
```

Implémentation de la fonction d'accès : *Val(i, long)*

Conventions :

Nb sur **2 octets**

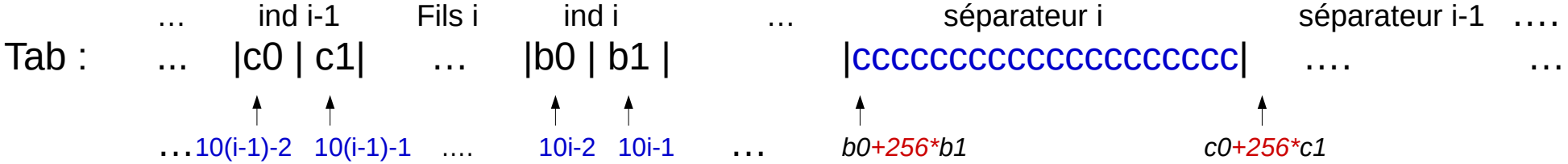
N° de fils sur **8 octets**

Indices des séparateurs sur **2 octets**

Les entiers seront représentés avec le format *little endian* (l'octet de poids faible en premier)

Tab : [f_1 ind_1 f_2 ind_2 f_3 ind_3 … f_n ind_n f_{n+1} < - - - vide - - - > sep_n sep_{n-1} …. sep_1]

Pour récupérer la valeur du séparateur **N° i**, il faut récupérer d'abord son indice représenté par les octets entre les positions **10 i - 2** et **10 i - 1**, ensuite récupérer sa valeur à partir de l'indice récupéré. Sa longueur est déterminée par l'indice du séparateur précédent (le **N° i-1**).



L'indice est reconstitué à l'aide de la formule suivante :

ind_i = *b0 + b1 * 256* avec b0 = Tab[10 i - 2] et b1 = Tab[10 i - 1]

L'indice du séparateur précédent (N° i-1) est reconstitué de la même manière:

Ind_{i-1} = *c0 + c1 * 256* avec c0 = Tab[10(i-1) - 2] et c1 = Tab[10(i-1) - 1]

La valeur du séparateur N° i se trouve alors entre les indices : **ind_i** et **ind_{i-1} - 1** pour i > 1

La valeur du premier séparateur (N° 1) se trouve entre les indices : **ind_1** et **b** pour i = 1

Implémentation de la fonction d'accès : *Val(i, long)*

Val(i:entier ; sortie:Long:entier) : chaine

```
// Ind : indice du début de la chaine ...
Ind ← buf.Tab [ i*10 - 2 ] + buf.Tab[ i*10 - 1 ] * 256 ;

// IndPrec : indice du début de la prochaine chaine (ou b+1 si c'est le 1er séparateur)...
Si ( i > 1 )
   IndPrec ← buf.Tab [ (i-1)*10 - 2 ] + buf.Tab[ (i-1)*10 - 1 ] * 256 ;
Sinon
   IndPrec ← b+1 ;
FSI ;

// récupération de la chaine représentant la valeur du séparateur i ...
Ch ← AllouerChaine( IndPrec - Ind ) ;        // malloc
Pour k = Ind , IndPrec - 1
   Ch[ k - Ind ] ← buf.Tab[ k ]
FP ;
Ch[ IndPrec - Ind ] ← '\0' ;

//  La longueur de la chaine récupérée ...
Long ← IndPrec - Ind ;

Retourner Ch ;
```

Implémentation des fonctions d'accès : *Généralisation*

Conventions : *N° de fils* sur **X octets**
Indices des séparateurs sur **Y octets**

Tab : [f_1 ind_1 f_2 ind_2 f_3 … ind_n f_{n+1} | < - - - vide - - - > | sep_n sep_{n-1} …. sep_1]

------- ------ ------ ------- ------ ------ ------

<--X--> <--Y--> <--X--> <--Y--> <--X--> … <--y--> <--X-->

debutVide **finVide**

Le $Fils_i$ se trouve entre les positions : $(X + Y)(i - 1)$ **et** $(X + Y)(i - 1) + X - 1$

L'indice du séparateur N°i (Ind_i) se trouve entre les positions : $(X + Y)\, i - Y$ **et** $(X + Y)\, i - 1$

L'espace vide dans le bloc se trouve entre les positions : **debutVide** et **finVide** :

debutVide = la position du dernier octet de $Fils_{Nb+1} + 1 = (X+Y)\, Nb + X$

finVide = la position qui précède le premier octet de la valeur de $séparateur_{Nb}$

= $Ind_{Nb} - 1$

= $Tab[(X+Y)Nb - Y] + Tab[(X+Y)Nb - Y - 1] * 256 + \ldots Tab[(X+Y)Nb - 1] * 256^{Y-1} - 1$

La taille de l'espace vide dans le bloc est :

Taille_Vide = finVide - debutVide +1

Pour **Insérer une nouvelle paire < Val , fd >** dans le bloc à une certaine position j,

on vérifie d'abord : $(Longueur(Val) + X + Y \leq Taille\_Vide)$

Sinon on réalise l'éclatement du bloc.

3) B*-arbre

C’est un B-arbre classique avec un facteur de chargement minimal plus élevé
→ au voisinage de 66 % (au lieu des 50 % de la méthode de base)

L’idée principale est de **retarder les éclatements** le plus possible lors des insertions :

Lorsqu’un bloc est déjà plein, lors d’une insertion d’une nouvelle valeur, on utilise des redistributions avec le frère gauche ou droit pour dégager un emplacement libre pour accueillir la nouvelle valeur.

Lorsque la redistribution n’est plus possible (c’est le cas lorsque les frères sont pleins eux aussi), on réalise un éclatement de 2 blocs en 3, chacun rempli à ***2/3*** de sa capacité maximale (≈ 66 %)

L’avantage est d’augmenter le degré moyen des blocs, diminuant ainsi la hauteur de l’arbre.

Pour aller plus loin

Voir l’article de synthèse :

Goetz Graefe, ***Modern B-Tree Techniques***
Foundations and Trends in Databases, Vol. 3, No. 4 (2010) 203–402

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.219.7269&rep=rep1&type=pdf