REPUBLIQUE DU CAMEROUN
REPUBLIC OF CAMEROON
Peace-Work-Fatherland
****
UNIVERSITE DE DSCHANG
UNIVERSITY OF DSCHANG
Scholar-Thesaurus DschangensisIbiCordum
BP 96, Dschang(Cameroun)- Tel/Fax:(237)232 45 1381
Website: http://www.univ.dschang.org/iutfv
Email: iutfv-bandjoun@univ-dschang.org

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
FOTSO VICTOR DE BANDJOUN
FOTSO VICTOR UNIVERSITY INSTITUTE OF YECHNOLOGY
Département de Mathématiques- et Informatique
Departement of Computer Engineering
BP 134, Bandjoun- Tel. / Fax (237) 699 31 61 30 /670 64 23 92
Website: http://www.univ.dschang.org/iutfv
Email: iutfv-bandjoun@univ-dschang.org

EXPOSÉ DE COMPLEXITÉS ET ALGORITHMIQUES AVANCÉES

THEME : LES B-ARBRES

Effectué par le **GROUPE 3**

Membres du groupe

NOMS ET PRENOMS	Matricule
DEUDJUI CHAMEDEU MARCEL PARFAIT	
NGAMEGNE KOPJIP JAURES PARFAIT	
SIBAFO KENGNE ANNE PATRICIA	CM-UDS 17SCI1494
TCHUENKO DJOKO CELINE CABRELLE	CM-UDS 18SCI 1479

Sous la coordination de :

Dr. KENGNE Vianney

Année académique 2021 / 2022

PLAN DE L'EXPOSE

LISTES DES FIGURES

INTRODUCTION

Le monde de l'informatique est en constante évolution et surtout à la recherche permanente de moyens pour optimiser la représentation des données en mémoire, la gestion de base de données et de système de fichiers. C'est pour cette raison que de nombreuses structures de données ont été créées et optimisées parmi lesquelles on retrouve les arbres binaires, les arbres binaires de recherche, les arbres rouges et noirs, les arbres B équilibrés. Dans le cadre de ce cours, nous nous intéresserons aux arbres B.

Les B-arbres sont une manière efficace, élégante et assez simple de conserver les avantages des arbres équilibrés tout en les adaptant aux caractéristiques particulières des fichiers. Bien sûr, l'idée de départ est de gonfler les nœuds et de leur faire porter plus d'une clé.

Ainsi nous commencerons par définir un arbre B tout en donnant ses différentes propriétés. Ensuite, nous allons d'une part présenter les opérations élémentaires applicables dans un arbre B suivies des différents algorithmes et leurs complexités respectives. Enfin, nous donnerons l'intérêt et quelques applications des arbres B dans le domaine de l'informatique et de la technologie.

1) Rappel sur les arbres

A. Définition: Etiquette ou clés

Un arbre dont tous les nœuds sont nommés est dit étiqueté. L'étiquette (ou nom du sommet) représente la "Clé" du nœud ou bien l'information associée au nœud. Ci-dessous un arbre étiqueté dans les entiers entre 1 et 10 :

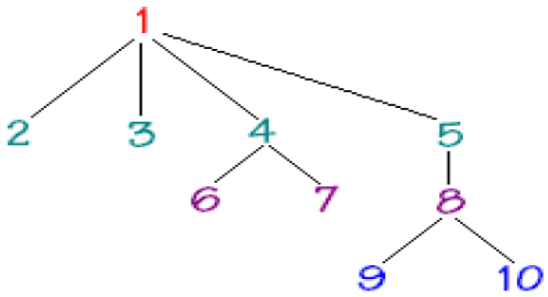


Figure 1: Etiquettes ou cles d'un arbre

B. Racine, feuille, branche

La racine d'un arbre représente le premier nœud de l'arbre.

Une feuille est un nœud qui ne possède pas de fils.

Nous rappelons la terminologie de base sur les arbres sur le schéma ci-dessous :

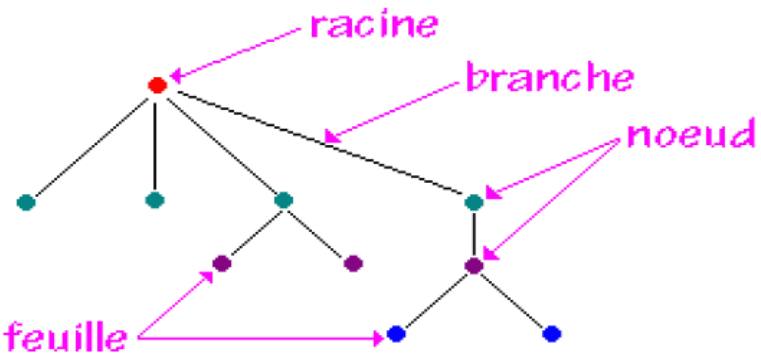


Figure 2: Racine, branche et noeud d'un arbre

C. Hauteur, profondeur ou niveau d'un nœud

Nous conviendrons de définir la hauteur (ou profondeur ou niveau) d'un nœud X comme égale au nombre de nœud à partir de la racine pour aller jusqu'au nœud X. Notant h la fonction hauteur d'un nœud et considérons l'arbre suivant :

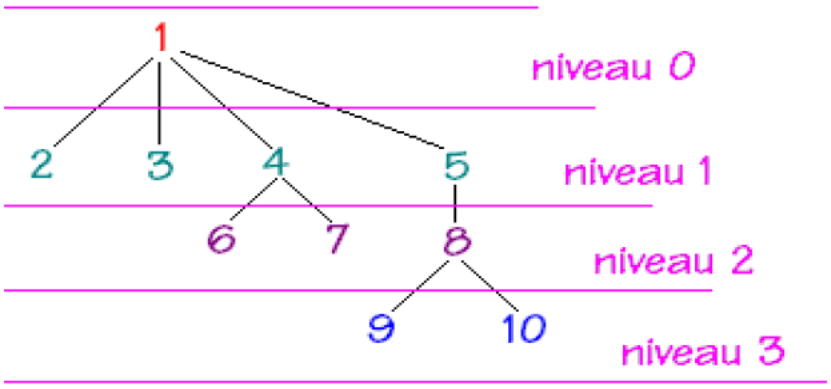


Figure 3: Hauteur, profondeur ou niveau d'un noeud

Pour atteindre le nœud étiqueté 9, il faut parcourir le lien 1—>5, puis 5—>8, puis enfin 8—>9 soient 4 nœuds donc 9 est de profondeur ou de hauteur égale à 4, soit h(9) = 4. Pour atteindre le nœud étiqueté 7, il faut parcourir le lien 1—>4, et enfin 4—>7, donc 7 est de profondeur ou de hauteur égale à 3, soit h(7)=3. Par définition la hauteur de la racine est égale à 1 mais Certains auteurs adoptent une autre convention pour calculer la hauteur d'un nœud : la racine a pour hauteur 0 et donc n'est pas comptée dans le nombre de nœuds, ce qui donne une hauteur inférieure d'une unité à notre définition.

D. Chemin d'un nœud

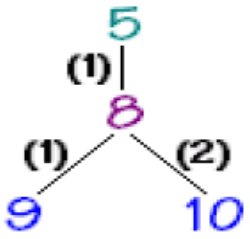
On appelle chemin du nœud X la suite des nœuds par lesquels il faut passer pour aller de la racine vers le nœud X. en considérant le précèdent arbre, on a :

Chemin du nœud 10 = (1, 5, 8,10)
Chemin du nœud 9 = (1, 5, 8,9)
Chemin du nœud 7 = (1, 4,7)
Chemin du nœud 5 = (1,5)

Remarque **:** la hauteur h d'un nœud X est égale au nombre de nœud dans le chemin :
h(X) = NbrNoeud(Chemin(X)).

E. Dégré d'un nœud

Par définition le degré d'un nœud est égal au nombre de ses descendants (enfants). Soit l'exemple ci-dessous :



Le nœud 5 n'ayant qu'un enfant son degré est 1.

Le nœud 8 est de degré 2 car il a 2 enfants.

Figure 4: Dégré d'un noeud

F. Hauteur ou profondeur d'un arbre

Par définition c'est le nombre de nœuds du chemin le plus long dans l'arbre. La hauteur h d'un arbre correspond donc au nombre de niveau maximum. La hauteur de l'arbre ci-dessous est 4.

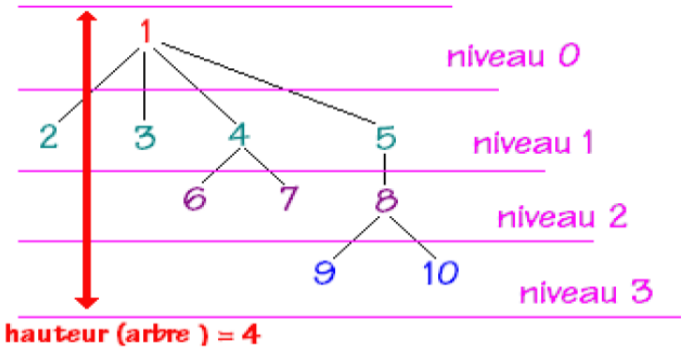
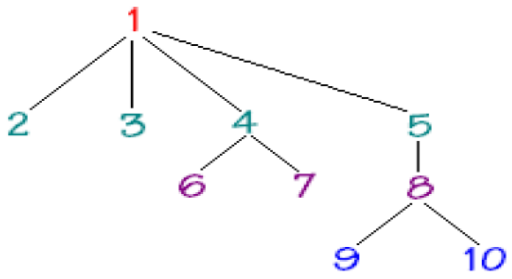


Figure 5: Hauteur ou profondeur d'un arbre

G. Dégré et taille d'un arbre

Le degré d'un arbre est égal au plus grand des degrés de ses nœuds. La taille d'un arbre quant à elle représente le nombre total de nœuds de cet arbre.

Soit à répertorier dans l'arbre ci-dessous le degré de chacun des nœuds :



d°(1) = 4 d°(2) = 0
d°(3) = 0 d°(4) = 2
d°(5) = 1 d°(6) = 0
d°(7) = 0 d°(8) = 2
d°(9) = 0 d°(10) = 0

Figure 6: Dégré et taille d'un arbre

La valeur maximale est 4, donc cet arbre est de degré **4.**

Par ailleurs, sa taille est de **10** car il possède 10 nœuds.

2) La notion de B-arbre

A. Contexte

Les B-arbres (ou B-Tree) sont une structure de définition de données utilisée dans les domaines tels que :

- Systèmes de gestion de fichiers : ReiserFS (version modifiée des B-arbres) ou Btrfs (B-Tree file system) ;
- Bases de données : gestion des index

Les B-arbres reprennent le concept d'arbre binaire de recherche (ABR) équilibré mais en stockant dans un nœud k les valeurs nommées clés et en référençant k + 1 sous arbres, minimisant ainsi la taille de l'arbre et réduisant le nombre d'opération d'équilibrage ; Les B-arbre sont utilisés pour un stockage sur disque (Unité de masse).

B. Définition

Un B-arbre d'ordre m est un arbre tel que :

- Chaque nœud contient k clés triées avec : $\mathbf{m \leq k \leq 2m}$ pour un nœud non racine et $\mathbf{1 \leq k \leq 2m}$ pour un nœud racine.
- Chaque chemin de la racine à une feuille est de même longueur à 1 près
- Un nœud est :
- Soit terminal : C'est une feuille
- Soit possède *(k + 1)* fils tels que les clés du *i* ème fils ont des valeurs comprises entre les valeurs du *i-1* ème et *i* ème clés du père.

C. Structure d'un nœud

Un nœud est composé de :

- k clés triées ;
- k+1 pointeur tels que :
- Tous sont différents de NIL si le nœud n'est pas une feuille,
- Tous sont égale à NIL si le nœud est une feuille.

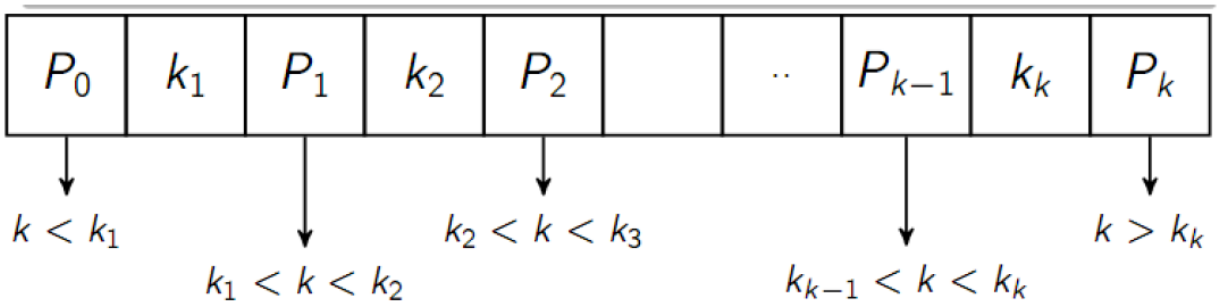


Figure 7: Structure d'un noeud

Exemple de B arbre d'ordre 2

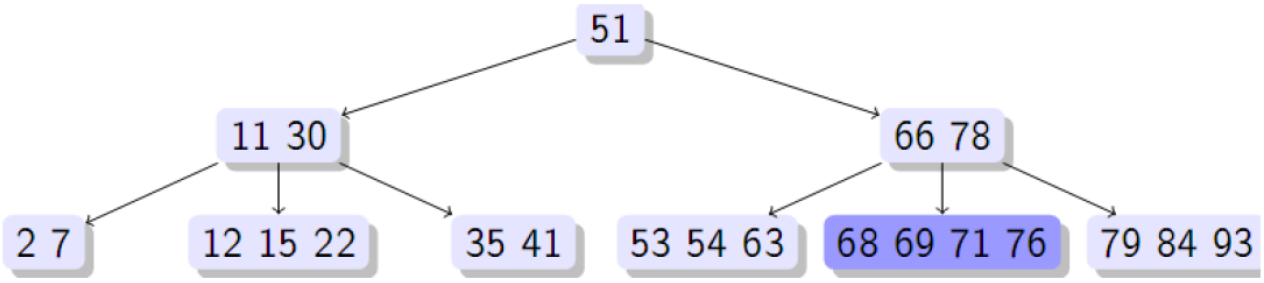


Figure 8: Exemple de B-arbre d'ordre 2

Algorithme et opération sur les B arbres

D. Création d'un B arbre

- Algorithme

***ALGORITHME** : B-arbre*

Début

Type *Barbre = Noeud*

Type *Noeud =* ***Structure***

nbClés : ***Naturel_Non_Nul***

clés : ***Tableau*** *[1..MAX] de* ***Valeur***

sous_Arbres : Tableau [0..MAX] de B arbre

Finstructure

#Valeur = Possède des valeurs ordonnées

Fin

- **Complexité**

Nombre d'opérations oe =0

Nombre d'affectations oa =1

Nombre de comparaisons oc =0

Nombre de boucles eb=0

La complexité de création est donc **O(1)** car ici il n'y a que identification des différentes variables constituant le B-arbre.

E. Recherche dans un arbre

- **Principe**

Considérons une clé C à rechercher alors à partir de la racine pour chaque nœud examiné :

- Si la clé C est présente alors retourner la clé C ;
- Si C < K1 alors rechercher dans le sous-arbre le plus à gauche via le pointeur P0
- Si C > Kk alors rechercher dans le sous-arbre le plus à droite via le pointeur Pk

- Si Ki < C < Ki+1 alors rechercher la clé C dans le sous arbre via le pointeur Pi
- Si l'arbre est vide le pointeur vaut NIL et il y a échec.

Remarque : la recherche peut se faire à travers les processus de dichotomie.

- **Algorithme**

```
ALGO : RechercheCleBarbre
Fonction estPresent (a : B-Arbre, c : Valeur) : Booleen Début
    Si a==NIL alors
        Retourner
        FAUX ;
    Sinon
        Si c<a.cles[1] alors
          Retourner estPresent(a.sousArbres[0],c) ;
        Sinon
          Si c>a.cles[a.nbCles] alors
                Retourner estPresent(a.sousArbre[a.nbCles] , c) ;
            Sinon
                RechercherDansNoeud(a,c,res,ssArbre) ;
              Si res alors
                  Retourner VRAI ;
              Sinon
                  Retourner estPresent(ssArbre) ;
               Finsi
            Finsi
        Finsi
    Finsi
Fin
```

```
ALGO : RechercheCleBarbre
Procedure rechercherDansNoeud(n : Nœud, c : Valeur,  estPresent :Booleen, sousArbre :B-Arbre) Declaration g, d, m : Naturel & Non Nul
Début
```

g ← 1;

d ← n.nbCles;

Tant que g ≠ d faire

m ← (g+d) div2 ;

si n.cles[m] ≥ alors

d ← m;

sinon

g ← m+1;

finsi

fintanque

si n.cles[g]==c alors

estPresent ← VRAI;

sousArbre ← NIL;

sinon

estPresent ← FAUX;

sousArbre ← n.sousArbres[g-1];

finsi

Fin

- **Complexité**

  Calcul de la complexité de la fonction **rechercherDansNoeud** (n : Nœud, c : Valeur, estPresent : Booleen, sousArbre : B-Arbre) **:**

  Soit **oe** le nombre d'opérations, **ae** le nombre d'affectations, **ce** le nombre de comparaisons.

  T(n)= ae +n/2(oe + ae +ce) +ce

  T(n)=O (n/2)

  Complexité de la procédure de recherche =O ($\log_2(n)$) ;

  Calcul de la complexité de la procédure **estPresent (a : B-Arbre, c : Valeur)** :

  Complexité = O ($\log_2(n)$);

Complexité total ALGO : RechercheCleBarbre :

Complexité = O ($\log_2(n)$);

F. Insertion dans un B-arbre

➢ **Principe**

- L'insertion se fait récursivement au niveau des feuilles
- Si un nœud a alors plus $2m + 1$ clés, il y a éclatement du nœud et remontée (grâce à la récursivité) de la clé médiane au niveau du père
- Il y a augmentation de la hauteur de l'arbre lorsque la racine se retrouve avec $2m + 1$ clés (l'augmentation de la hauteur de l'arbre se fait donc au niveau de la racine)

➢ **Algorithme**

On suppose posséder les fonctions/procédures suivantes :

- **fonction** creerFeuille **(**c : **Tableau [**1..MAX**] de** Valeur, nb : **NaturelNonNul) :** ArbreB
- **fonction** estUneFeuille **(**a : ArbreB**) : Booleen**
- **fonction** eclatement **(**a : ArbreB, ordre : **NaturelNonNul) :** ArbreB (**précondition(s)** a.nbCles>2*ordre)
- **fonction** positionInsertion **(**a : ArbreB, c : Valeur**) : NaturelNonNul**
- **procédure** insererUneCleDansNoeud **(E/S** n : Noeud, **E** c : Valeur, pos :**NaturelNonNul)**
- **procédure** insererUnArbreDansNoeud **(E/S** n : Noeud, **E** a : ArbreB, pos : **NaturelNonNul)**
  - **procédure** inserer **(E/S** a : ArbreB, **E** c : Valeur, ordre : **NaturelNonNul) debut** a←insertion(a,c,ordre)

Fonction *insertion (a : ArbreB, c : Valeur, ordre : NaturelNonNul) : ArbreB*
Déclaration tab : Tableau[1..MAX] de Valeur
debut
si a = NIL alors
tab[1] ←c ;
retourner creerFeuille(tab,1) ;
sinon

```
            pos← positionInsertion(a,c) ;

        si estUneFeuille(a) alors
        insererUneCleDansNoeud(a^,c,pos) ;

            si a^.nbCles≤2*ordre alors
                    retourner a ;
             sinon
                    retourner eclatement(a, ordre) ;
            finsi

        sinon

            ssArbre= a^.sousArbres[pos] ;
                    temp =
            insertion(ssArbre,c,ordre)

            si ssArbre = temp alors
                    retourner a ;
            sinon
                    insererUnArbreDansNoeud(a^,temp,pos) ;

                si a^.nbCles≤2*ordre alors
                        retourner a ;
                sinon
              retourner eclatement(a, ordre) ;
                finsi

            finsi

        finsi

    finsi
```

Fin

- **Exemples**

<u>Exemple 1 : Insertion dans un nœud plein</u> :

<u>Principe</u> :

Principe : Eclatement du nœud en deux :

- Les (deux) plus petites clés restent dans le nœud

- Les (deux) plus grandes clés sont insérées dans un nouveau nœud
- Remontée de la clé médiane dans le nœud père

Exemple 2 : Insertion d'une clé dans un B-arbre d'ordre 2 (Insertion de 75)

On veut insérer la clé 75, on procède ainsi : On effectue tout d'abord la recherche et le repérage du nœud concerné, on vérifie à partir de l'ordre de l'arbre si le nombre de clé est atteint dans le nœud :

- Si le nœud n'est pas saturé on insère la clé 75 et on réordonne les valeurs des clés du nœud,
- Si le nœud est saturé, on effectue l'éclatement du nœud et la remonté de la clé médiane vers le nœud père.

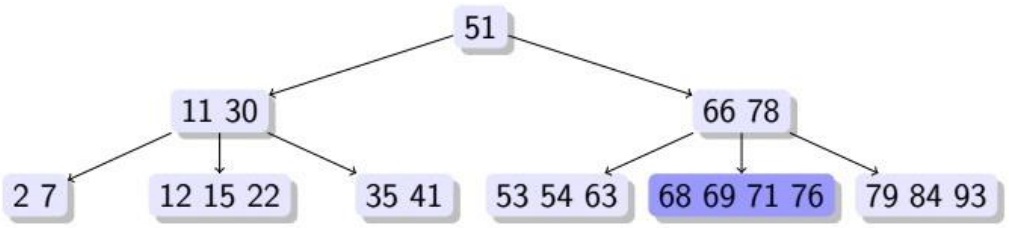


Figure 9: Arbre avant insertion de la clé

Rappel : ici nombre de clés par nœud ≤ 4

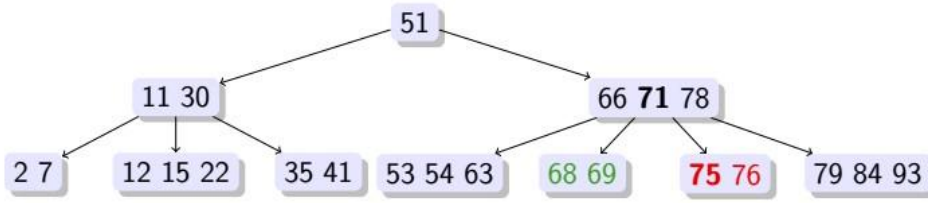


Figure 10: Arbre après insertion de la clé

Exemple 3 : Insertion dans un nœud plein avec augmentation de la hauteur de l'arbre

Dans le cas de l'insertion dans un nœud plein qui produit une augmentation de la hauteur de l'arbre la méthode peut se décrire ainsi :

- Insertion dans un nœud plein telle que présentée précédemment (insertion de la nouvelle clé suivie de l'éclatement du nœud et de la remontée de la clé médiane au nœud père),
- Ensuite il y a éclatement du nœud père et création du nouveau nœud père ou d'une nouvelle **racine** (cas où le nœud père était la racine et l'ordre était atteint).

- Enfin il y a eu Augmentation d'une ou plusieurs unité(s) de la hauteur du B-arbre Arbre-B d'ordre 2

Exemple : Insertion de 9

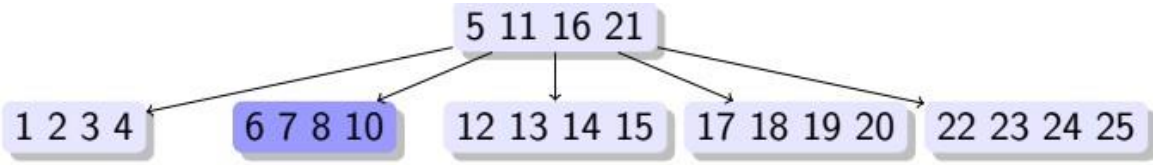


Figure 11: Arbre initial avant éclatement

- Insertion clé **9** → Eclatement + remontée de la clé **8** au nœud père
- Remontée de la clé **8** au nœud père →Eclatement + création nouvelle racine

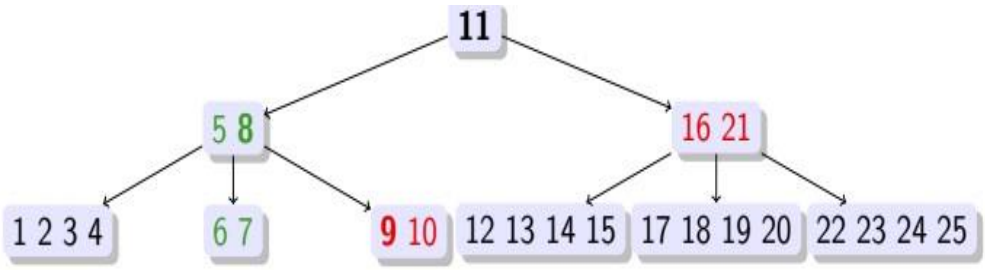


Figure 12: Arbre après éclatement et insertion

➢ **Complexité**

Complexite = O(log(n)) ;

G. Suppression dans un B-arbre

➢ **Principe**

Dans un B-arbre, la suppression peut se faire au niveau des feuilles ; si la clé à supprimer n'est pas dans une feuille, alors on la remplace par la plus grande valeur des plus petites clés (ou plus petite valeur des plus grandes clés) et on supprime cette dernière. Si la suppression de la clé d'une feuille (récursivement d'un nœud) amène à avoir moins de **m** clés dans un nœud : on

fait une combinaison (fusion) de ce nœud avec un nœud voisin (avant ou après) et on descend la clé associant ces deux nœuds (avec éclatement du nœud si nécessaire), la récursivité de ce principe pouvant amener à diminuer la hauteur de l'arbre par le haut.

- **Exemple**

Exemple 1 de suppression : combinaison avec un nœud voisin

Le B-arbre est d'ordre 2, le nombre de clés par nœud non racine > 1 ; Pour la suppression de la clé 25, on procède comme suit :

- ✓ Combinaison avec un nœud voisin ([12 ,14] et 20) ;
- ✓ Descente de la clé médiane (ici 15) ;
- ✓ Suppression du nœud.

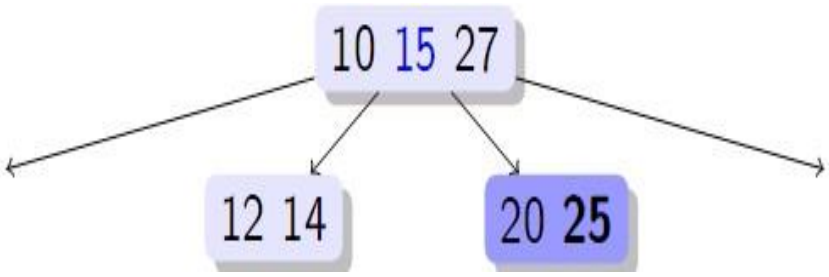


Figure 13: Exemple : B-arbre d'ordre 2- Suppression de la clé 25

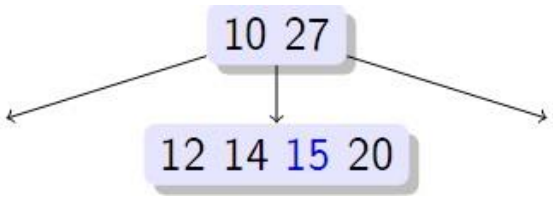


Figure 14: Exemple: Combinaison avec le Nœud voisin et descente de la clé médiane

Exemple 2 de suppression : Avec éclatement d'un nœud

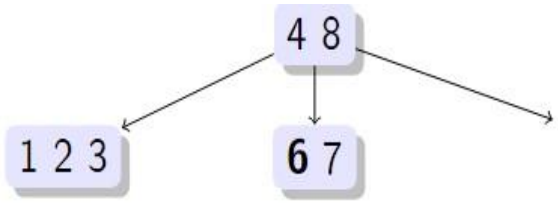


Figure 15: Suppression avec éclatement du nœud

Dans cet exemple, le nombre de clés par nœud non racine est < 5 et > 1 ; on veut supprimer la clé **6**, on procède comme suit:

Suppression clé **6**, Combinaison des clés **[1 2 3]** et **7** puis descente de la clé **4** au nœud fils ; ensuite redistribution des clés et remontée de clé médiane.

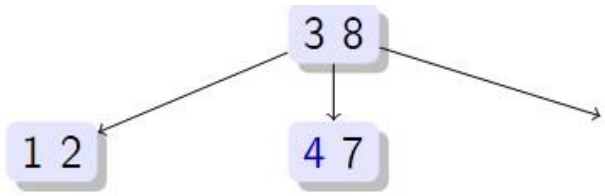


Figure 16: Suppression + éclatement du nœud : redistribution des clés et remontée clé médiane

Exemple 3 de suppression : Avec un nombre de clé inférieur à m

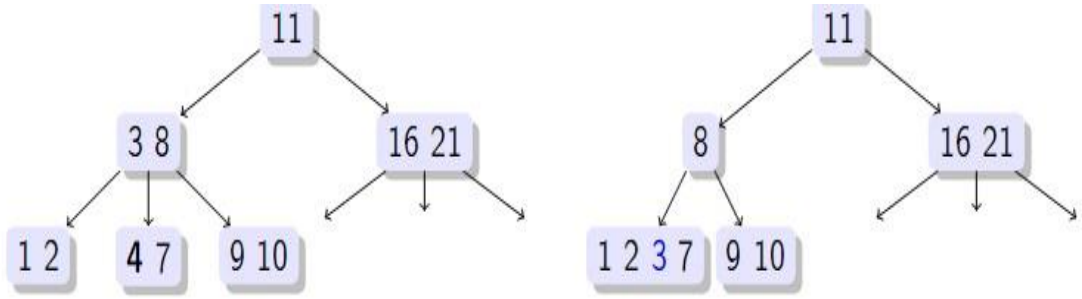


Figure 17: Suppression avec un nombre de clé inferieur à m

On veut supprimer la clé 4 :

On effectue tout d'abord une Combinaison des clés ([1 2] et 7) et une descente de la clé 3 ; Combinaison des clés (8 et [16 21]) et descente de la clé 11. Ceci entraine une diminution d'une unité de la hauteur de notre B-arbre.

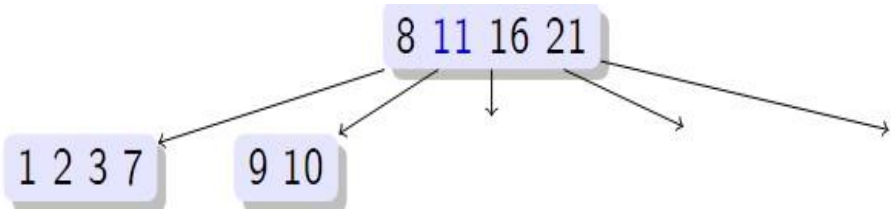


Figure 18: Suppression + nombre de clé inferieur à m : diminution hauteur B-arbre

Exemple 4 de suppression : dans un nœud non feuille

Principe :

Pour effectuer une telle suppression, on procède comme suit :

- ✓ Tout d'abord on effectue la recherche d'une clé adjacente **Ca** à la clé à supprimer puis on choisit la plus grande du sous arbre gauche ;
- ✓ On effectue le remplacement de la clé à supprimer par **Ca ;**
- ✓ On supprime la clé remplacée du sous arbre gauche.

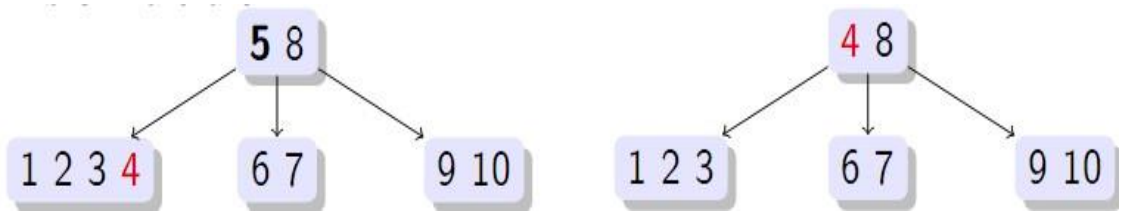


Figure 19: Suppression nœud non feuille

On veut supprimer la clé 5 du nœud racine de notre figure précédente, on procède ainsi :

Recherche de la plus grande clé adjacente à 5 dans le sous arbre gauche ici 4 ; on effectue ensuite une permutation entre la clé 5 de la racine et 4 du nœud feuille, enfin on supprime la clé 5.

➢ **Algorithmes**

- ❖ **Fonction** plusGrandeValeur (a : B-Arbre) : Valeur
- ❖ précondition(s) ≠ NIL ;
- ❖ **Fonction** positionCleDansNoeudRacine (a : B-Arbre, c : Valeur) : Entier
  - o précondition(s) a ≠ NIL ;
- ❖ **Fonction** positionSsArbrePouvantContenirValeur (a : Arbre, c : Valeur) : Naturel
- ❖ **Procedure** freres (a : B-Arbre, posSSArbre : Naturel,S frereG, frereD : B-Arbre) précondition(s) a ≠ NIL et a.sousArbres[posSsArbre] ≠ NIL ;

- **Procedure** supprimerCleDansNoeudFeuille (n : Nœud, E c : Valeur)
- **Procedure** decalerVersGaucheClesEtSsArbres (E/S n : Noeud, E aPartirDe : NaturelNonNul, nbCran : NaturelNonNul)

ALGO : Suppression B-arbre

Procedure supprimer (a : B-Arbre, E c : Valeur, ordre : NaturelNonNul)

Debut

a ← suppression(a,c,ordre,NIL,NIL,uneCle) ;

Fin

Fonction *suppression(a : B-Arbre, c : Valeur, ordre : NaturelNonNul, frereG, frereD : BArbre, clePere : Valeur) : B-Arbre*

Début

```
si a = NIL alors
    retourner a ;
sinon
    pos ← positionCleDansNoeudRacine(a,c) ;

        si estUneFeuille(a) alors
            si pos = 1 alors

                retourner a ;
            sinon

                si frereG = NIL &&  frereD = NIL &&  a.nbCles=1 alors
                            desallouer(a) ;
                sinon
                                # supprimer c dans une feuille
                        supprimerCleDansNoeudFeuille(a,c) ;
                finsi
                finsi
        sinon
                si pos = -1 alors
                        posSsArbre ← positionSsArbrePouvantContenirValeur(a,c) ;
                sinon
                        cleRemplacement ← plusGrandeValeur(a.sousArbres[pos-1]) ;
                          a.valeurs[pos] ← cleRemplacement ;
                          c ← cleRemplacement posSsArbre ← pos-1 ;
                finsi
```

```
                freres(a,posSsArbre,frereG,frereD) ;
        finsi
    finsi
Fin

Fonction supprimerCleDansNoeudFeuille(a,c)
Début
    si a.nbCles ≥ ordre ou (frereG = NIL et frereD = NIL)
        alors  retourner a ;
    sinon
        si frereG 6= NIL alors
                copierValeurs(tab,frereG.valeurs,1,1,frereG.nbCles)
        ;
                tab[frereG.nbCles+1] ← clePere ;
                copierValeurs(tab,a.valeurs,frereG.nbCles+2,1,a.nbCles) ;
                nb ← 1+a.nbCles+frereG.nbCles ;
                desallouer(frereG) ;
         sinon
              copierValeurs(tab,a^.valeurs,1,1,a.nbCles) ;
                tab[a.nbCles+1] ← clePere ;
                copierValeurs(tab,frereD.valeurs,a.nbCles+2,1,frereD.nbCles) ;
                nb ← 1+a.nbCles+frereD.nbCles ;
                desallouer(frereD) ;
         finsi
                res ← creerFeuille(tab,nb) ;
                desallouer(a) ;

                    si nb>2*ordre alors
                        retourner
                        eclatement(res, ordre) ;
                    finsi
                        retourner res ;
        finsi
Fin

Fonction freres(a,posSsArbre,frereG,frereD)
Début

        res←suppression(a.sousArbres[posSsArbre],c,ordre,frereG,frereD,a
        .valeurs[ posSsArbre]) ;
    si res.nbCles=1 alors
```

```
                a.valeurs[posSsArbre] ← res.valeurs[1] ;
                a.sousArbres[posSsArbre-1] ← res.sousArbres[0] ;
                a.sousArbres[posSsArbre] ← res.sousArbres[1] ;
  sinon
     decalerVersGaucheClesEtSsArbres(aˆ,posSsArbre,1) ;
          a.sousArbres[posSsArbre] ← res ;
 finsi
        retourner a ;
Fin
```

- **Calcul de la complexité de l'algorithme de suppression :**

Calcul de la complexité de la procédure **ALGO** : Suppression B-arbre

Nombre d'opérations eo =n/2

Nombre d'affectations ea = n/2

Nombre de comparaisons ec =n/2

Nombre de boucles eb=1

Complexité=O ($\log_2(n)$) ;

H. Interets des B-arbre

Les b-arbres apportent de solides avantages en terme de rapidité et d'efficacité par rapport à d'autres mises en œuvre lorsque la plupart des nœuds sont dans le stockage secondaire, comme le disque dur. En maximisant le nombre de nœuds enfants pour chaque nœud, la hauteur de l'arbre est réduite, l'opération d'équilibrage est nécessaire moins souvent et donc l'augmentation de l'efficacité. En général, ce nombre est réglé de telle sorte que chaque nœud occupe tout un groupe de secteurs : ainsi étant donné que les opérations de bas niveau pour accéder au disque de cluster, il réduit au minimum le nombre d'accès à elle. Ils offrent d'excellences performances par rapport aux deux opérations de recherche et ceux de rénovation, car les deux peuvent être faites avec la complexité logarithmique et par utilisation des procédures très simples. Sur eux, il est également possible d'effectuer un traitement séquentiel d'archivage primaire sans qu'il soit nécessaire de le soumettre à une réorganisation.

CONCLUSION

En définitive, nous avons détaillé la structure des B-arbres et analysé ses performances à travers les algorithmes de recherche, d'insertion et de suppression. Ces algorithmes ont une complexité de O (log n) au pire des cas ce qui fait des B-arbre une structure de données efficace. Même si les B-arbres possèdent des propriétés intéressantes, ils n'en demeurent pas moins que des désavantages subsistent pour quelques applications. Par exemple les B-arbres peuvent nécessiter plusieurs opérations de regroupements ou d'éclatement après une opération insertion ou de suppression.

Bibliographie

- Introduction-à-l'algorithmique-Cours-et-exercices-corrigés Dunod, Paris, 2004, pour la présente édition ISBN 2 10 003922 9

Webographie

https://fr.wikipedia.org/wiki/Arbre_B

https://rmdiscala.developpez.com/cours/LesChapitres.html/Cours4/Chap4.7.htm

http://igm.univ-mlv.fr/~mac/ENS/DOC/barbr_17.pdf

https://www.labri.fr/perso/maylis/ASDF/supports/notes/FILM/B-arbreExemples.pdf

Fiche de TD

Exercice 1

Soient les valeurs suivantes :

3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 8, 19, 4, 31, 35, 56

1) Construire un B-arbre d'ordre 5 avec ces valeurs.
2) Donner la forme de l'arbre ainsi obtenu après les opérations suivantes.

- Insertion des clés : 2, 6, 12
- Suppression les clés : 4, 5, 7, 3, 14

Exercise 2

Soient les valeurs suivantes

2, 3, 5, 7, 11, 17, 19, 23, 29, 31

En supposant que l'arbre est initialement vide et que les clés s'insèrent en partant de la dernière,

1) Construire un B-arbre d'après les valeurs de **d** (Ordre de l'arbre = nombre minimal de clés par nœud) suivantes :

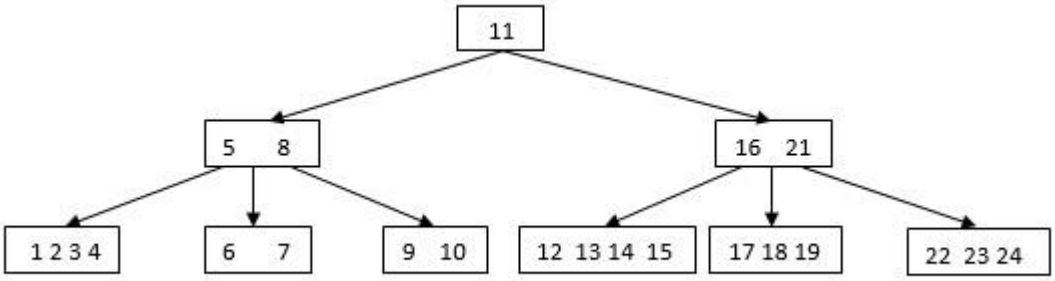
• 4 • 7

2) Donner la forme de l'arbre après la séquence d'opérations suivante : (considérer le cas d=4)

• **Insertion** 9, 10, 8, 6, 1, 4 • **Suppression** de 10, 19, 7

Exercice 3

On considère un arbre B d'ordre 2 suivant.



1. Représentez l'arbre après suppression de la clé 5.
2. Représentez l'arbre après la suppression de la clé 6.
3. Représentez l'arbre après la suppression de la clé 4.

Exercise 4

1) Expliquer comment trouver la plus petite clé contenue dans un B-arbre. Proposer un algorithme correspondant à votre description.

2) Dire comment trouver le prédécesseur et le successeur d'une clé donnée dans un arbre-B.

Exercise 5

Un B arbre T contient $10^5 + 1$ clés. Le nombre maximal de clés par nœud est de **18**. Combien d'accès disque sont nécessaires pour récupérer une valeur *a* contenue dans l'arbre?

Exercise 6

Mr Valentin est employé par un le entrepreneur qui a besoin de ses services pour développer une application de gestion des locations d'automobiles. Lors de la conception de l'application, il décide d'utiliser la structure de B-arbres pour des

questions d'optimisation d'opérations. Suite à une analyse faites du projet, Mr Valentin parviens aux résultats suivants :

- Les ordinateurs de la structure de son employeur ont des disques dont les blocs peuvent contenir 8192 octets,
- Une clé (valeur identifiant un automobile) occupe un espace de 8 octets,
- Un pointeur occupe 8 octets,
- Le pointeur sur le nœud père (Mr Valentin souhaite toujours avoir une référence aux pères des nœuds) a **8 octets**,
- La référence vers l'espace de stockage en mémoire d'une donnée (constituée du numéro de bloc et de l'offset de déplacement) occupe 16 octets.

Il se rend compte qu'au final il aura à manipuler environ 300 000 000 données sur les automobiles.

1) Quelle valeur Mr Valentin doit-il utiliser comme **ordre** de son B-arbre afin de gérer l'espace de manière optimale? Expliquez votre raisonnement. (Un nœud correspond à un bloc sur le disque)

2) Combien de blocs seront utilisés au total au pire et au meilleur des cas?

3) Si nous supposons que les données des nœuds situés à la même hauteur sont stockées dans une même page du disque, de combien de pages aura-t-il besoin pour stocker toutes ses données?

4) Combien d'accès disque seront nécessaires au maximum pour récupérer une informations?

5) Même question si Mr Valentin décide de sauvegarder ses données plutôt dans une structure d'arbre binaire.

Exercice 7

soit une base de données utilisée pour stocker quelques informations des étudiants de IN4 de l'université de Dschang. Cette base de donnée contient une table.

Etudiant (matricule, nom, sexe, date de naissance).

Cette table contient 100 enregistrements. Si on suppose que les champs de cette table sont tels que :

Matricule (20 Octets), Nom (50 Octets), Sexe (8 Octet), Date de naissance(12 Octets), Lieu de naissance (27 Octets) Contact (11 Octets).

1- Quel est le nombre total de blocks que nous pouvons utiliser pour stocker les 100 enregistrements sur le disque ?
2- Si on suppose qu'un pointeur sur un enregistrement utilise 6 Octets sur le disque, utilisez les index pour déterminer le nombre de blocks que nous pouvons accéder lorsqu'on recherche une information dans la table Etudiant.

Exercice 8

Un B arbre T contient 105 + 1 clés. Le nombre maximal de clés par nœud est de **18**. Combien d'accès disque sont nécessaires pour récupérer une valeur *a* contenue dans l'arbre ?

Exercice 9

Mr ASHU est employé par un entrepreneur qui a besoin de ses services pour développer une application de gestion des locations d'automobiles. Lors de la conception de l'application, il décide d'utiliser la structure de B-arbres pour des questions d'optimisation d'opérations. Suite à une analyse faites du projet, Mr Valentin parviens aux résultats suivants :

- Les ordinateurs de la structure de son employeur ont des disques dont les blocs peuvent contenir 8192 octets.
- Une clé (valeur identifiant un automobile) occupe un espace de 8 octets,
- Un pointeur occupe 8 octets,
- Le pointeur sur le nœud père (Mr Valentin souhaite toujours avoir une réfé- rence aux pères des nœuds) a **8 octets**,

• La référence vers l'espace de stockage en mémoire d'une donnée (constituée du numéro de bloc et de l'offset de déplacement) occupe 16 octets. Il se rend compte qu'au final il aura à manipuler environ 300 000 000 données sur les automobiles.

1) Quelle valeur Mr Valentin doit-il utiliser comme **ordre** de son B-arbre afin de gérer l'espace de manière optimale ? Expliquez votre raisonnement. (Un nœud correspond à un bloc sur le disque) ***2)*** Combien de blocs seront utilisés au total au pire et au meilleur des cas ?

3) Si nous supposons que les données des nœuds situés à la même hauteur sont stockées dans une même page du disque, de combien de pages aura-t-il besoin pour stocker toutes ses données ?

4) Combien d'accès disque seront nécessaires au maximum pour récupérer une informations ?

5) Même question si Mr Valentin décide de sauvegarder ses données plutôt dans une structure d'arbre binaire.

Exercice 10

Un arbre B indexe un fichier de 300 enregistrements.

Dans un premier temps, on suppose que l'ordre de l'arbre est de 5. Chaque nœud stocke donc au plus

10 entrées. Quelle est la hauteur minimale de l'arbre et sa hauteur maximale ? (Un arbre constitué uniquement de la racine a pour hauteur 0).

Inversement, on ignore l'ordre de l'arbre mais on constate qu'il a pour hauteur 1. Quel est l'ordre maximal compatible avec cette constatation ? Et l'ordre minimal ?

Exercice 8

Soit un fichier de 1 000 000 enregistrements répartis en blocs de 4 096 octets. Chaque enregistrement fait 45 octets et il n'y a pas de chevauchement de blocs. Répondez aux questions suivantes en justifiant vos réponses (on suppose que les blocs sont pleins). ***1)*** Combien faut-il de blocs ? Quelle est la taille du fichier ?

2) Quelle est la taille d'un index de type arbre-B si la clé fait 32 octets et une adresse 8 octets ? Détaillez le calcul niveau par niveau.

3) Même question si la clé fait 4 octets

4) Si on suppose qu'une lecture coûte 10 ms, quel est le coût moyen d'une recherche d'un enregistrement par clé unique, avec index et sans index dans le pire des cas ?