ARBRE B

HAVART FLORENTIN
BUGNON FLORENTIN

Dépôt Github

Description

Un arbre-B est un arbre (au sens informatique du terme) tel qu'à chaque nœud on associe plusieurs clés prises dans un ensemble donné.

Soient L et U deux entiers naturels non nuls tels que L ≤ U. En toute généralité, on définit alors un L-U arbre B3 de la manière suivante : chaque nœud, sauf la racine, possède un minimum de L−1 clé (appelées aussi éléments), un maximum de U−1 clé et au plus “U” fils. Pour chaque nœud interne - nœud qui n’est pas une feuille -, le nombre de fils est toujours égal au nombre de clés augmenté d’une unité. Si “n” est le nombre de fils, alors on parle de n-nœud. Un L-U arbre ne contient que des n-nœuds avec L ≤ n ≤ U. Souvent on choisit la configuration L = t et U = 2 × t : t est appelé le degré minimal de l’arbre B.

De plus, la construction des arbres B garantit qu’un arbre B est toujours équilibré. Chaque clé d’un nœud interne est en fait une borne qui distingue les sous-arbres de ce nœud. Par exemple, si un nœud a 3 fils — lesquels constituent les racines respectives de trois sous-arbres : sous-arbre gauche, sous-arbre du milieu et sous-arbre droit —, alors il a 2 clés notées c1 et c2 qui délimitent les clés de chaque sous-arbre : les clés du sous-arbre gauche seront inférieures à c1 ; les clés du sous-arbre du milieu seront comprises entre c1 et c2 ; les clés du sous-arbre droit seront supérieures à c2.

Nous avons décidé de coder ce projet en Java.

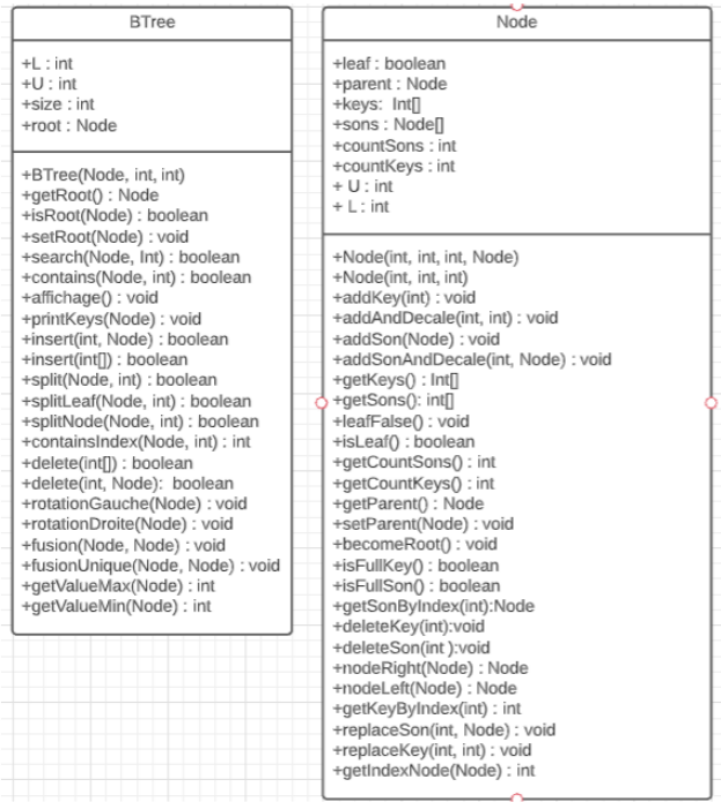
Concernant la structure de données, nous avons une classe représentant l'arbre B, ainsi qu'une classe représentant un nœud contenu dans l'arbre B.

Nous avons rencontré un problème lors de la définition de l'ensemble des fils d'un nœud. En premier lieu nous avions choisi une ArrayList qui permettait une allocation dynamique et l'utilisation de toutes les méthodes qui y sont liées, mais cela nous posait des problèmes d'indexation lors du parcours de la liste dans une boucle. Nous avons alors décidé d'utiliser un tableau simple de Java, de taille fixée au début. Ainsi, les cases ne contenant aucune valeur aurait simplement la valeur "null"

Avancement du projet

12/01/2022	initial commit
26/01/2022	Recherche et structure
13/01/2022	Structure de Btree et de Node
09/02/2022	Début de l'insertion
05/03/2022	Fin de l'insertion
23/03/2022	Fin de Delete / Fin du projet

Le répertoire git contient un git ignore, un makefile (make cls pour compiler , make run pour lancer le projet, make doc pour créer la documentation et make clean pour enlever la documentation et les classes du répertoire), un dossier image avec l'image de l'uml, le pdf du rapport, un readme, et le dossier Arbre-B avec le code du projet.



Contenu

Node.java

Cette classe représente un nœud d'un arbre-B. On l'a défini avec une valeur, un nombre pour le nombre de valeur minimum, et un nombre pour le nombre de valeur maximum qui sera contenu dans chaque nœud de l'arbre.

On a un tableau de clés et de fils, ainsi que deux attributs représentant le nombre de clés et de fils contenus dans le nœud. On initialise le tableau de clés avec la valeur 0 (à changer)

On initialise le tableau de sons avec des valeurs null.

Il y a aussi un attribut booléen qui est True si le node est une feuille. Il y a un attribut node parent qui représente le parent du nœud. Ce parent est “null” si le nœud est la racine de l'arbre.

Btree.java

Cette classe représente un arbre N-Aire. On l'a défini avec un nœud racine, un nombre pour le nombre de valeur minimum, et un nombre pour le nombre de valeur maximum qui sera contenu dans chaque nœud de l'arbre.

On a, en plus comme attribut, la taille de l'arbre.

Algorithmes

Recherche

On commence à la racine

1. Si le noeud est une feuille
    a. On regarde si la valeur qu'on recherche est dans les clés du noeud
2. Sinon
    a. on regarde si la valeur qu'on recherche est dans les clés du noeud
    b. Si elle n'y est pas
        i. On cherche le fils qui peut contenir la valeur qu'on cherche
        ii. On applique l'algo de manière récursive sur ce nœud.
3. Pour trouver le fils correspondant, on incrémente une variable i de 0 à n tant que:
    a. n est plus petit que le nombre de clé maximum dans un noeud
    b. la valeur qu' on cherche est plus petite que la valeur à l' index i dans le tableau de clé du noeud
    c. l'index de i dans le tableau des clés soit différent de 0 (comme on initialise à 0)
4. On appelle la fonction de **recherche** sur le fils correspondant (celui qui correspond à l' index i dans le tableau des fils)

Insertion

On commence à la racine

1. Si le noeud est une feuille
    a. Si le noeud est plein
        i. On split le noeud en question
    b. Sinon
        i. On ajoute tout simplement la clé au tableau de clés du noeud
2. Sinon
    a. On cherche dans quel enfant il faut continuer l'insertion récursivement

Split

On commence à la racine

1. Si le noeud est une feuille
    a. Si le noeud a un parent
        i. Si ce parent n'est pas plein (rempli de clés) :
            1. On ajoute valeur médiane du noeud au parent
            2. On crée un second noeud, qui aura le même parent, avec les valeurs supérieures à la médiane
        ii. Sinon
            1. on split le parent
            2. On ajoute la valeur médiane du noeud au parent
            3. On crée un second noeud, qui aura le même parent, avec les valeurs supérieures à la médiane
    b. Sinon
        i. On crée un nouveau noeud qui sera le parent du noeud actuel
        ii. On ajoute valeur médiane du noeud au parent
        iii. On crée un second noeud, qui aura le même parent, avec les valeurs supérieures à la médiane
2. Sinon
    a. Si le noeud a un parent
        i. Si ce parent n'est pas plein (rempli de clés)
            1. On ajoute valeur médiane du noeud au parent
            2. On crée un second noeud, qui aura le même parent, avec les clés et les enfants supérieurs à la médiane
        ii. Sinon
            1. On split le parent
            2. On ajoute valeur médiane du noeud au parent
            3. On crée un second noeud, qui aura le même parent, avec les clés et les enfants supérieurs à la médiane
    b. Sinon
        i. On crée un nouveau noeud qui sera le parent du noeud actuel
        ii. On ajoute valeur médiane du noeud au parent
        iii. On crée un second noeud, qui aura le même parent, avec les clés et les enfants supérieures à la médiane

Suppression

paramètre 1 : la valeur à supprimer

paramètre 2 : le noeud où on commence la suppression

1. Si le noeud contient la valeur à supprimer
    a. Si c'est une feuille
        i. s'il reste + de L-1 valeurs après suppression
            1. On supprime simplement la valeur du noeud
        ii. Sinon
            1. si l'un des frères a + de L-1 Valeurs
                a. On effectue une ***rotation*** avec ce frère
            2. Sinon
                a. on effectue une ***fusion***
    b. Sinon
        i. On remplace la valeur par la plus petite valeur du sous-arbre droit de cette valeur
        ii. on rappelle **delete** sur cette plus petit valeur (on se ramène alors au cas du delete dans une feuille)
2. Sinon
    a. On cherche dans quel enfant on va continuer la suppression

Fusion

paramètre 1 : le noeud qui va accueillir toutes les clés et les enfants après la fusion

paramètre 2 : le noeud qui va fusionner avec le paramètre 1 (son frère gauche ou droit)

1. On supprime la valeur à supprimer du noeud (paramètre 1)
2. On ajoute au noeud(paramètre 1) la clé du parent qui se trouve juste après celui-ci dans le parent
3. On ajoute au noeud(paramètre 1) toutes les clés et tous les enfants du noeud voisin (paramètre 2)
4. On supprime du parent le pointeur vers le noeud voisin (paramètre 2)
5. On supprime du parent la clé qu'on a passé au noeud(paramètre 1)
6. Si le parent n'a plus assez de valeur et qu'il n'est pas la racine
    a. On rappelle fusion sur le parent et son voisin de droite ou gauche
7. Si le parent n'a plus assez de valeur et qu'il est la racine
    a. on fusionne ses enfants une fois et on arrête la récursivité de fusion

Rotation gauche

paramètre 1 : le noeud où la clé est supprimée

1. On supprime du noeud actuel la valeur qu'on souhaite supprimer
2. on ajoute au noeud actuel, la valeur de la clé du parent, qui est juste après le pointeur du noeud actuel
3. On remplace cette valeur par la plus petite valeur du voisin droit du noeud actuel
4. On appelle **suppression** sur la plus petite valeur du voisin droit

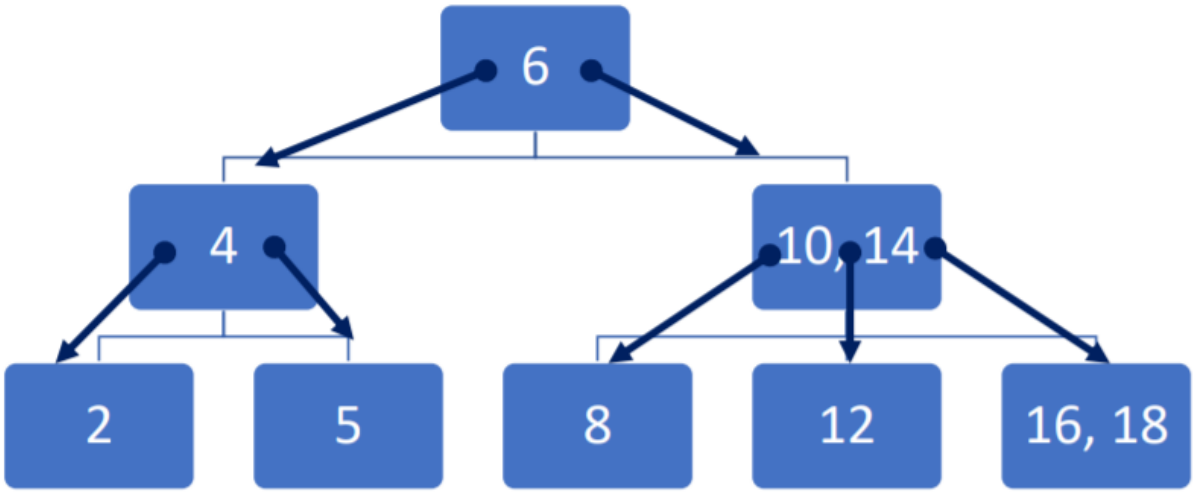
Rotation droite

paramètre 1 : le noeud où la clé est supprimée

1. On supprime du noeud actuel la valeur qu'on souhaite supprimer
2. on ajoute au noeud actuel, la valeur de la clé du parent, qui est juste avant le pointeur du noeud actuel
3. On remplace cette valeur par la plus grande valeur du voisin droit du noeud actuel
4. On appelle delete sur la plus grande valeur du voisin gauche

Exemples

Recherche



exemple de la recherche avec le nombre 6:

le noeud n'est pas une feuille donc on rendre dans le sinon

la valeur est dans le noeud donc on retourne true

exemple de la recherche avec le nombre 10:

le noeud n'est pas une feuille donc on rendre dans le sinon

la valeur n’est pas dans le noeud donc on itère sur i

i=0 , i < 3 , 10 > 6 , 6 != 0 -> i=1

i =1 , i<3 , 10 > 0 , 0 n est pas différent de 0 , donc on arrête le tant que

on recommence la recherche sur le noeud 10,14

le noeud n'est pas une feuille donc on rendre dans le sinon

la valeur est dans le noeud donc on retourne true

exemple de la recherche avec le nombre 12:

le noeud n'est pas une feuille donc on rendre dans le sinon

la valeur n’est pas dans le noeud donc on itère sur i

i=0 , i < 3 , 12 > 6 , 6 != 0 -> i=1

i =1 , i<3 , 12 > 0 , 0 n est pas différent de 0 , donc on arrête le tant que

on recommence la recherche sur le noeud 10,14

le noeud n'est pas une feuille donc on rendre dans le sinon

la valeur n' est pas dans le noeud donc on itère sur i

i=0 , i< 3 , 12 > 10 , 10 != 0 -> i=1

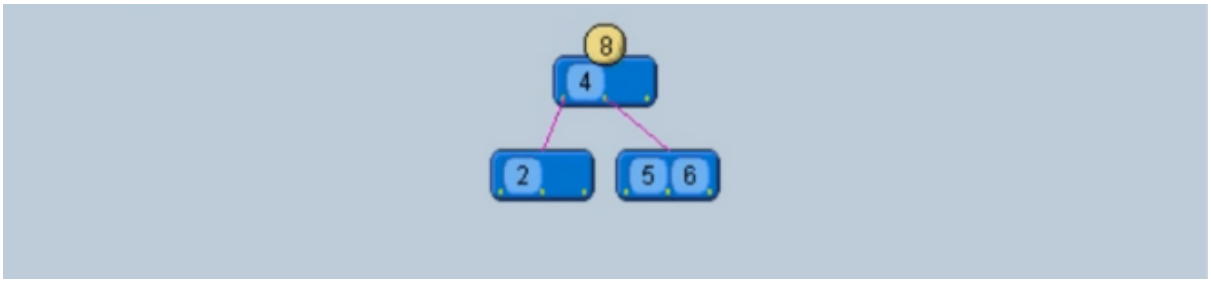
i=1 , i<3 , 12 < 14 -> donc on arrête i=1

on fait la recherche sur le noeud index 0 dans le tableau de fils

le noeud est une feuille , et il contient 12 donc on retourne true

(si la valeur n'est pas dans l'arbre , on retrouvera le même résultat sauf qu'on trouvera pas la valeur dans le noeud et on retournera false)

Insertion



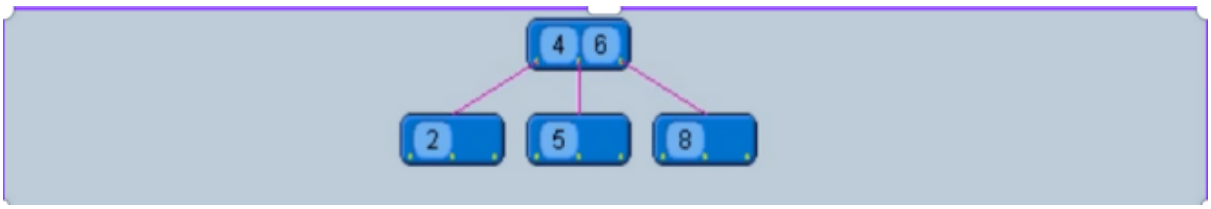
Insérer 8 :

- Le noeud racine n'est pas une feuille, donc on regarde dans quel enfant on va chercher à insérer 8

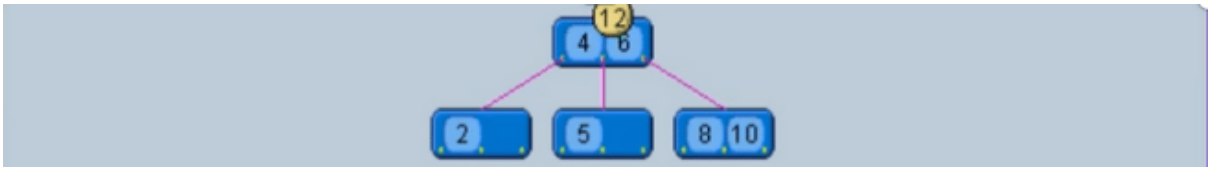
- On veut insérer 8 dans le noeud à droite mais il est plein alors on le split

- Cela fait remonter la valeur médiane du nœud au parent (ici c'est 6).

- Voici l'état de l'arbre après insertion de 8



Insertion 2



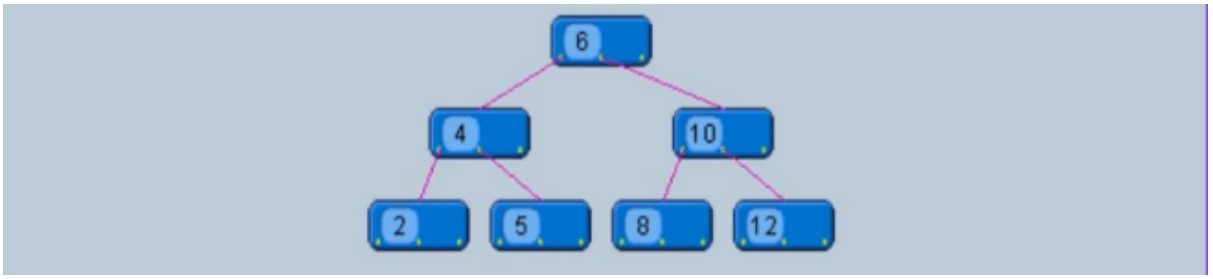
insérer 12 :

-La racine n'est pas une feuille donc on cherche l'enfant dans lequel on va continuer l'insertion

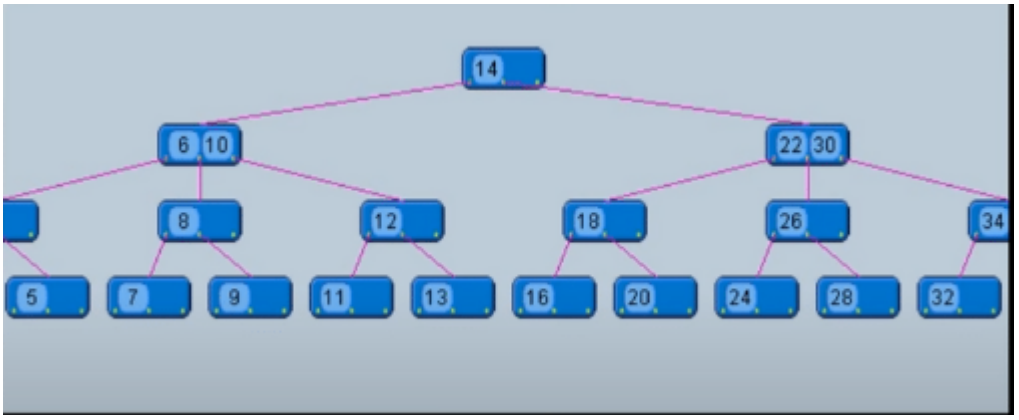
- On veut insérer dans la feuille de droite mais celle-ci est pleine alors on split

-Ce split va entraîner le split du noeud parent

-Voici l'état de l'arbre à la fin de l'insertion



Suppression



-> 14 se trouve dans le noeud racine

-> on remplace 14 par la valeur la plus petite du sous-arbre droit qui est 16

-> on rappelle delete sur la clé 16, en partant du sous-arbre droit

-> 16 se trouve dans une feuille, aucune rotation n'est possible, dans cette situation une fusion sera effectuée

-> le parent n'aura plus assez de clés, il y aura à nouveau une fusion

-> on obtient finalement le résultat final ci-dessous