Projet-s6 groupe 2

Membres du groupe :

Hicham HARRA et Alexandre PY

git : https://gitlab-etu.fil.univ-lille1.fr/harra/projet-s6

Arbre B (B-tree) :

L'arbre B sont principalement mis en œuvre dans les mécanismes de gestion de bases de données et de système de fichiers.

*** Avantages :**

grâce à la taille de l'arbre est minimisée, arbre équilibré et aussi les données stockées de maniéré triée ; cela nous permet d'exécuter les les instruction sur l'arbre (recherche, insertion …) avec une complexité en temps logarithmique.

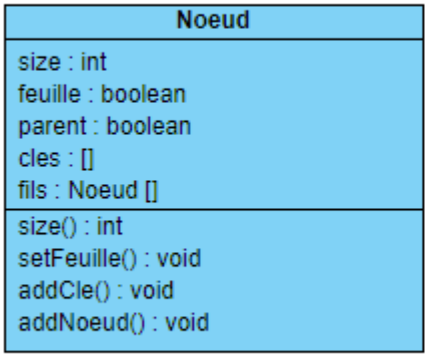
Objectif du projet :

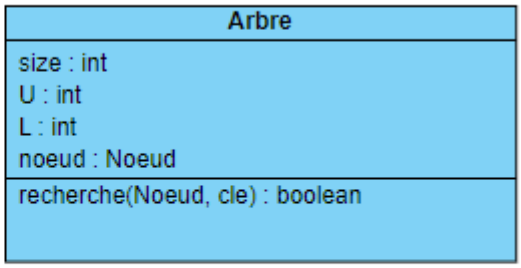
Modélisation de la structure Arbre_B et l'implémentation de l'algorithme de recherche, d'insertion et de suppression.

Livrable 1 :

Modélisation des Classes Nœud et Arbre : Nous avons modélisé l'arbre avec deux classes :

- Classe Nœud : qui définit les nœud de l'arbre, qui caractérisé par
- La taille : nombre de clés dans l'arbre -feuille : bool True si le nœud est une feuille -parent : le nœud parent -clés : liste des clés (les valeurs) -fils : liste des nœuds fils à ce nœud (de taille maximale : taille de la liste des clés +1 (notions des pointeurs)).
- Classe Arbre qui représente l'arbre_B, caractérisé par : -size : taille de l'arbre -U : nombre de nœuds fils max -L : nombre de nœuds fils min -racine d : nœud racine





Conditions de l'arbre B (B-tree) :

-U : nombre de nœud fils max.

-L: nombre de nœud fils min.

- le nombre de clés d'un nœud (sauf racine) est compris entre L-1 et U-1

- Arbre équilibré : toutes les feuilles ont la même hauteur

Algorithmes

Pour ce livrable nous avons réussi à finir l'algorithme de recherche

Algorithme de recherche :

Entrées : nœud (racine de l'arbre) e élément à chercher
Sortie : un boolean True s'il exite dans l'arbre, False sinon.

```
si le noeud est une feuille:
        resultat <- e in liste des cles du noeud
sinon si e est dans la liste des cles :
        resultats  <- True
sinon:
        i=0
        ind=-1;
        tant que n < nombre des cles et ind ==-1:
                si e liste des cles[i]:
                    ind =i
                i++
        si ind ==-1:
             ind = nombre de cles
        resultat <- recherche(e,fils[ind])
renvoyer resultat
```

- Dans l'appel récursif à chaque fois on cherche l'indice de sous arbre où on cherche, donc la récursivité s'applique seulement sur un sous nœud à chaque fois. Cela nous donne une complexité égale à la hauteur de l'arbre dans le pire des cas donc Θ(logn(n)).
- La recherche dans la liste des clés on utilise la recherche dichotomique afin de garantir la bonne complexité.
- Nous avons défini une classe tests qui vérifie le bon fonctionnement de l'algorithme de recherche.

Livrable 2 et 3 (3: mise à jour insertion)

Algorithme d’insertion :

L’objectif est de réaliser un algorithme qui nous permet d’insérer un élément à l’arbre, en le gardant équilibré et avec une bonne complexité logarithmique.

Mécanisme :

Pour garder l’arbre équilibré on commence par insérer l’élément dans la feuille dont l’intervalle correspond à l’élément.

Ensuite si la feuille est saturée, on l’explosé en deux nœuds puis on insère la médiane dans le nœud parent, les clés de nœud explosé la moitié devienne les clés du premier nouveau nœud, l’autre moitié les clés de deuxième nœud créée,

et si ce dernier est saturé on répète le même processus sur le parent de manière récursif, et si on est dans la racine (donc pas de parent), on crée une nouvelle racine .

On remarque qu’au pire des cas (quand à chaque fois le parent est saturé) , on refait les mêmes instructions(Explosion..) dans le parent, ce qui donne la complexité en temps égale à la hauteur de l’arbre donc logarithmique Θ(logn(n).

Pseudo code :

Entrées : nœud (racine de l'arbre) e élément à insérer
feuille ← feuille ou on insère
si la feuille n'est pas saturer:
    ajouter e au clés de feuille
sinon si la feuille est la racine:
    on insère e dans la feuille
    newRacine = nouveau noeud
    m=valeur médiane
    n1 ← nœud avec les clés feuille (de la première jusqu'à la médiane)
    n2 ← nœud avec les clés feuille (de la médiane jusqu'à la dernière)
    ajouter la médiane au clés de nouvelle racine
    les fils (si n'est pas une feulle) de feuille partager dans n1 et n2
                                moitié moitié
    les clés de feuille partager dans n1 et n2 moitié moitié
    dans les fils de nouvelle racine on ajoute n1 et n2

sinon :
    on insère e dans la feuille
    m=valeur médiane
    n1 ← nœud avec les clés feuille (de la première jusqu'à la médiane)
    n2 ← nœud avec les clés feuille (de la médiane jusqu'à la dernière)
    ajouter la médiane au clés de nouvelle racine
    les fils (si n'est pas une feulle) de feuille partager dans n1 et n2
                                moitié moitié
    les clés de feuille partager dans n1 et n2 moitié moitié
    dans les es fils de parant on remplace le nœuds feuille par n1 et n2
    insertion(parent,elt) ## insertion par appel récursif dans parent

Pour trouver la feuille ou on insère nous avons utilisé un algorithme comme l'algorithme de recherche, ce dernier nous renvoie le nœud où l'élément se trouve s'il existe sinon il renvoie le nœud où la recherche s'est arrêté qui correspond au nœud où on insère.

Avancement du projet:

Dans livrable 3 Nous avons réussi à finir l'algorithme de recherche avec les tests.
Nous avons corriger l'algorithme d'insertion et on a codé ses test.
Touts les tests sont validés.

À cause du problème de l’insertion de livrable2,on est obligé de recommencer l’insertion, nous n’avons pas réussi à faire l’algorithme de suppression.

Exemple qu’on a testé pour l’algorithme d’insertion :

Ça donner le résultat suivant :