Rapport sur les arbre B-tree

SOMMAIRE:

Présenté Par :

- LEKHAL Massinissa
- MOULOUEL Tarik

C’est quoi un arbre B-tree ?

• Un arbre B-tree (ou arbre B) est une structure de données utilisée dans les bases de données, les systèmes de fichiers et d’autres applications où il est important d’effectuer des recherches, des insertions et des suppressions efficaces dans de grands ensembles de données.

Historique de l’arbre B-tree

• L’histoire de l’arbre B-tree remonte aux années 1960, lorsque le chercheur en informatique Rudolf Bayer a publié un article intitulé “Organisation et recherche de grands fichiers aléatoires en 1968. Dans cet article, Bayer a présenté une nouvelle structure d’arbre de recherche qui a été appelée plus tard “B-tree” (pour “Bayer tree”).Au fil des années, l’arbre B-tree a été largement adopté dans les systèmes de gestion de base de données, où il a été utilisé pour stocker des index sur les colonnes de la table pour permettre des recherches et des mises à jour rapides. Des variantes de l’arbre B-tree, telles que les arbres B-tree et les arbres B-tree, ont également été développées pour répondre à des besoins spécifiques de performance et d’utilisation des ressources.
• Aujourd’hui, l’arbre B-tree continue d’être largement utilisé dans les systèmes de gestion de base de données, les systèmes de fichiers et d’autres applications où il est important de stocker et de rechercher efficacement de grandes quantités de données.

Les caractéristiques d’un arbre B-tree

1. Structure équilibrée : l’arbre B-tree est un arbre équilibré dans lequel tous les chemins depuis la racine jusqu’aux feuilles ont la même longueur. Cela signifie que les opérations de recherche, d’insertion et de suppression ont une complexité temporelle garantie et prévisible.
2. Nœuds avec plusieurs clés : chaque nœud de l’arbre B-tree contient plusieurs clés triées. Le nombre de clés par nœud est généralement compris entre m et 2m, où m est un paramètre fixé lors de la création de l’arbre.
3. Pointeurs vers des sous-arbres : chaque nœud de l’arbre B-tree contient également des pointeurs vers des sous-arbres. Les clés stockées dans un nœud déterminent les plages de clés stockées dans les sous-arbres pointés.

4. Hauteur minimale : la hauteur minimale de l'arbre B-tree est garantie par la structure équilibrée de l'arbre et par une stratégie de division de nœud qui garantit que chaque nœud est rempli à au moins la moitié de sa capacité.
5. Optimisé pour le stockage sur disque : l'arbre B-tree est particulièrement adapté pour stocker des données sur des disques, car il minimise le nombre d'accès aux disques nécessaires pour accéder à une donnée ou un ensemble de données.
6. Performances prévisibles : grâce à la structure équilibrée et à la hauteur minimale garantie, les performances des opérations de recherche, d'insertion et de suppression sont prévisibles et efficaces, même pour des ensembles de données très volumineux.
•En raison de ces caractéristiques, l'arbre B-tree est largement utilisé dans les systèmes de gestion de base de données, les systèmes de fichiers et d'autres applications où il est important de stocker et de rechercher efficacement de grandes quantités de données.

Structure d'arbre B-tree en Python

• Durant ce projet nous avons implémenté deux classes, la première classe s'agit de la classe Node (nœud de l'arbre) qui est caractérisé par les fils(childs), les différentes valeurs du nœud (keys) ainsi que le parent du nœud (parent).
• Nous avons également implémenté deux fonctions importantes dans les arbres B-tree :
1. Recherche : qui permet de rechercher la présence ou non d'une valeur dans B-tree :

```
   fonction recherche(valeur):
   si valeur est dans self.root.keys :
       retourner True
   sinon si self.root est une feuille :
       retourner False
   sinon :
       i = 0
       tant que (i < longueur(self.root.keys) et valeur >
self.root.keys[i]) :
           i += 1
       sous_arbre = Btree(self.root.childs[i], self.k)
       retourner sous_arbre.recherche(valeur)
```

• Ce pseudo-code effectue une recherche récursive dans l'arbre B-tree. Si la valeur recherchée est trouvée dans la racine de l'arbre, la fonction retourne True. Si la racine de l'arbre est une feuille et la valeur n'est pas trouvée, La fonction retourne False. Sinon, la fonction recherche la clé dans le sous-arbre approprié et appelle récursivement la fonction sur ce sous-arbre. Le paramètre self.k est la taille maximale des nœuds de l'arbre et self.root est la racine de l'arbre.

2. Insertion : qui permet d'insérer une valeur dans un B-tree :

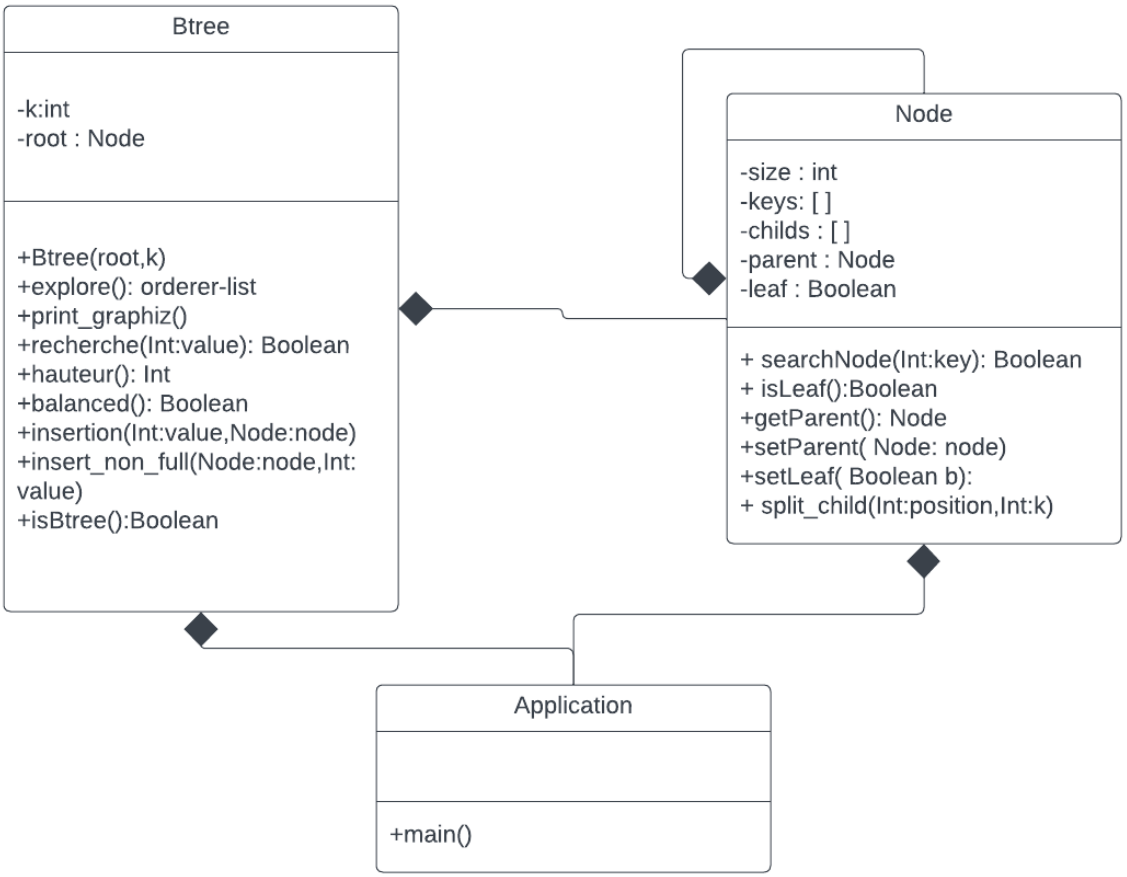
```
```
   fonction insertion(valeur):
   si self.root est vide :
       créer un nouveau nœud et ajouter la valeur
       self.root = ce nouveau nœud
   sinon :
       si self.root est plein :
           créer un nouveau nœud et le définir comme la nouvelle racine
           ajouter self.root comme sous-arbre de ce nouveau nœud
           diviser le nœud plein (self.root) en deux nœuds fils
           choisir le sous-arbre correct pour insérer la valeur
           appeler la fonction insertion_non_plein sur le sous-arbre
choisi
       sinon :
           appeler la fonction insertion_non_plein sur la racine de
l'arbre (self.root)
```
```

• Ce pseudo-code utilise une stratégie de division de nœud pour insérer de nouvelles valeurs dans l'arbre B-tree. Si la racine de l'arbre est pleine, le code divise la racine en deux nouveaux nœuds et crée une nouvelle racine pour l'arbre. Ensuite, il choisit le sous-arbre correct pour insérer la nouvelle valeur et appelle la fonction insert_non_full sur ce sous-arbre.
• Si la racine n'est pas pleine, la fonction insert_non_full est simplement appelée sur la racine de l'arbre pour insérer la nouvelle valeur dans l'arbre.
• Nous avons aussi implémenté plusieurs functions dans B-tree comme la function balanced qui permet de vérifier si un arbre B-tree est équilibré ou non, nous avons aussi implémenté une fonction hauteur qui calcule la hauteur d'un arbre B-tree .
• Nous avons aussi utilisé Graphiz pour la visualisation graphique qui nous permet de visualiser l'insertion et comment ça fonctionne .

Problèmes rencontrés

• On a pas réussi à faire la suppression dans un arbre B-tree.

Diagramme UML



Tests

- Pour les tests on a crée une classe Application qui contient les différents tests des différentes méthodes comme les tests de la méthode recherche ou encore les tests de la méthode explore.