Санкт-Петербургский политехнический университет Петра Великого

Кафедра компьютерных систем и программных технологий

**Отчёт по курсовой работе**

**Дисциплина**: Алгоритмы и структуры данных

**Тема**: «B-Tree»

Выполнил студент гр. 3530901/90004 И.А. Сергеев

(подпись)

Преподаватель А.О. Алексюк

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Оглавление**

[Определение 3](#_Toc65158888)

[Преимущества 3](#_Toc65158889)

[Атрибуты B-дерева 4](#_Toc65158890)

[Основные операции 5](#_Toc65158891)

[Структура проекта. 6](#_Toc65158892)

[Заключение 7](#_Toc65158893)

[Источники 8](#_Toc65158894)

# **Определение**

B-деревья представляют собой сбалансированные деревья поиска, созданные специально для эффективной работы с дисковой памятью (и другими типами вторичной памяти с непосредственным доступом). B-деревья похожи на красно-черные деревья, но отличаются более высокой оптимизацией дисковых операций ввода-вывода. Многие СУБД используют для хранения именно B-деревья или их разновидности.

B-деревья отличаются от красно-черных деревьев тем, что узлы B-дерева могут иметь много дочерних узлов – от нескольких штук до тысяч, так что степень ветвления B-дерева может быть очень большой.

B-деревья представляют собой естественное обобщение бинарных деревьев поиска. Если внутренний узел x B-дерева содержит x.n ключей, то у него x.n + 1 дочерних узлов. Ключи в узле x используются как разделители диапазона ключей, с которыми имеет дело данный узел, на x.n + 1 поддиапазонов, каждый из которых относится к одному дочерних узлов x.

# **Преимущества**

Другие структуры данных, такие как двоичное дерево поиска, дерево AVL, красно-черное дерево и т.д., могут хранить только один ключ в одном узле. Если вам нужно хранить большое количество ключей, то высота таких деревьев становится очень большой и время доступа увеличивается.

# **Атрибуты B-дерева**

Каждый узел x содержит следующие атрибуты:

1. x.n, количество ключей, хранящихся в настоящий момент в узле x;
2. Собственно, x.n ключей, хранящихся в неубывающем порядке.
3. Булево значение x.isLeaf, равное true, если x представляет собой лист, и false, если x является внутренним узлом.
4. Кроме того, каждый внутренний узел x содержит x.n + 1 указателей на дочерние узлы. У листьев дочерних узлов нет, так что значения их атрибутов не определены.
5. Ключи x.keys[i] разделяют поддиапазоны ключей, хранящихся в поддеревьях.

# **Основные операции**

1. Search.

Поиск в B-дереве очень похож на поиск в бинарном дереве поиска, но с тем отличием, что если в бинарном дереве мы выбирали один из двух путей, то здесь предстоит сделать выбор из большего количества альтернатив, зависящего от того, сколько дочерних узлов имеется у текущего узла.

1. Insert.

Как и в случае бинарных деревьев поиска, мы ищем позицию листа, в который будет вставлен новый ключ. Однако при работе с B-деревом мы не можем просто создать лист и вставить в него ключ, поскольку такое дерево не будет являться корректным B-деревом. Вместо этого мы вставляем новый ключ в существующий лист. Поскольку вставить новый ключ в заполненный лист невозможно, мы вводим новую операцию – разбиение (split) заполненного узла y на два, каждый из которого содержит минимум по T / 2 ключей для нечетного T и минимум по (T / 2) – 1 ключей для дерева с заданным четным T. Медиана, или средний ключ при этом перемещается в родительский по отношению к y узел, где становится разделительной точкой для двух вновь образовавшихся поддеревьев. Однако, если родительский становится узел заполнен, то для него также запускается операция разделения, и такой процесс разбиения может выполняться по восходящей до самого корня.

1. Remove.

Удаление ключа из B-дерева возможно не только из листа, но и из любого узла дерева. Удаление из внутреннего узла требует определенной перестройки дочерних узлов. Как и в случае вставки, мы должны обеспечить сохранение свойств B-дерева при выполнении операции удаления. Аналогично тому, как мы обеспечивали отсутствие переполнения при вставке нового ключа, нам предстоит обеспечивать условие, чтобы узел не становился слишком мало заполненным в процессе удаления ключа. Так же, как алгоритму вставки может потребоваться возврат, если узел на пути вставки ключа заполнен, так и алгоритму удаления может потребоваться возврат, если узел (не корневой) на пути вставки ключа имеет минимально допустимое количество ключей.

# **Структура проекта.**

Главный класс проекта – Tree содержит следующие поля:

* T – максимальное количество ключей, которые могут хранится в узле.
* Root – корень дерева.
* minimumT – переменная, необходимая для определения минимального количества ключей, которые могут хранится в дочерних узлах.

Также проект содержит два внутренних класса:

* Node – класс определяющий строение узла дерева.
* BTreeIterator – класс реализующий методы для итерации.

Внутренний класс Node содержит следующие поля:

* isLeaf – переменная типа Boolean, которая дает информацию является ли данный узел листом или нет.
* keys – массив, хранящий ключи типа int, размером равным T.
* children – массив, который хранит в себе дочерние узлы, если таковые имеются, размерность которого равна T + 1.

Пример того, как визуально выглядит B-дерево, протестированное в отдельном для тестов классе, изображено на рисунке 1.

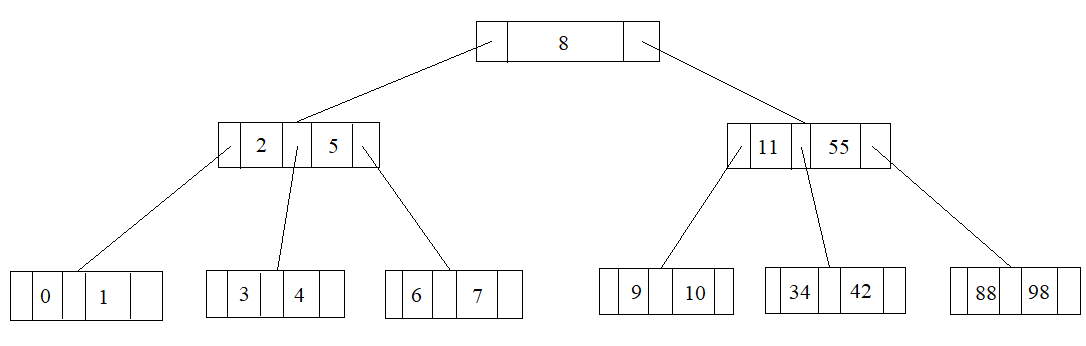


Рисунок 1. B-Tree.

# **Заключение**

В ходе выполнения курсовой работы мной были получены знания такой структуры данных, как B-Tree, которое используется преимущественно для эффективной работы с дисковой памятью. Были реализованы все основные методы, такие как удаление, вставка, поиск, разбиение. Правильность выполнения работы была проверена с помощью библиотеки JUnit

# **Источники**

1. https://www.cs.usfca.edu/~galles/visualization/BTree.html
2. https://habr.com/ru/post/114154/
3. https://en.wikipedia.org/wiki/B-tree
4. Т. Кормен «Алгоритмы: построение и анализ» второе издание