Министерство образования и науки Российской Федерации

Федеральное государственное образовательное бюджетное учреждение

Тверской государственный технический университет

Кафедра “Программное обеспечение”

Курсовая работа

по теме: «B+-tree, B\*-tree»

Выполнил:

Студент группы «Б.ПИН.РИС.22.06»

Невзоров Павел Евгеньевич

Проверил профессор:

Мальков Александр Анатольевич

г. Тверь, 2024.

**Оглавление**

[Введение 3](#_Toc161518504)

[Аналитическая часть 4](#_Toc161518505)

[Определение Дерева, виды деревьев 6](#_Toc161518506)

[Виды деревьев 6](#_Toc161518507)

[Разница между B+-tree, B\*-tree 8](#_Toc161518508)

[Где используется 10](#_Toc161518509)

[Средства разработки 11](#_Toc161518510)

[Проектирование 12](#_Toc161518511)

[Логика приложения 13](#_Toc161518512)

[Требования к приложению 14](#_Toc161518513)

[Код алгоритма и этапы его работы 15](#_Toc161518514)

[Тестирование 16](#_Toc161518515)

[Документация 17](#_Toc161518516)

[Характеристики используемой техники 18](#_Toc161518517)

[Краткое руководство пользователя 19](#_Toc161518518)

[Заключение 20](#_Toc161518519)

[Список литературы 21](#_Toc161518520)

# Введение

**Цели:**

* Изучение и понимание основных принципов работы B+-tree и B\*-tree.
* Разработка эффективной реализации этих структур данных на практике.
* Оценка производительности и сравнение эффективности B-tree, B+-tree и B\*-tree.

**Задачи:**

* Изучение теоретических основ B-деревьев и их вариаций.
* Реализация базовых операций (вставка, удаление, поиск) для B+-tree и B\*-tree.
* Анализ и сравнение производительности реализованных структур данных на различных наборах данных.

**Актуальность:**

B-tree, B+-tree и B\*-tree являются важными структурами данных, широко применяемыми в базах данных и системах индексации. Понимание их работы и возможностей позволяет эффективно организовывать данные, ускорять операции поиска, вставки и удаления, а также оптимизировать использование ресурсов хранения.

# Аналитическая часть

**История создания алгоритма**

B-tree было впервые предложено Рудольфом Бэйером и Эдвардом МакКрэй в 1972 году. Они разработали эту структуру данных для использования в базах данных, где требовалась эффективная организация и быстрый доступ к отсортированным данным.

B+-tree был разработан для улучшения производительности при работе с дисковым хранилищем. Его основная идея заключается в том, чтобы хранить все данные только в листовых узлах, что упрощает операции поиска и позволяет быстрее обрабатывать диапазонные запросы.

B\*-tree представляет собой дальнейшее развитие B+-tree с более жесткими правилами заполнения узлов, что улучшает производительность в некоторых сценариях использования.

**Модификации**

Существует несколько модификаций B-tree, B+-tree и B\*-tree, направленных на оптимизацию производительности, уменьшение потребления памяти или упрощение реализации. Некоторые из них включают кэширование, сжатие ключей и значения, адаптивное управление структурой дерева и др.

**Места применения с примерами**

B-tree: широко используется в базах данных, файловых системах, поисковых движках и других системах, где требуется эффективное управление большими объемами отсортированных данных. Примеры включают PostgreSQL, Oracle Database.

B+-tree: применяется в системах индексации баз данных, где основной упор делается на эффективный доступ к данным на диске. Примеры включают MySQL, SQLite.

B\*-tree: используется в высоконагруженных системах, где требуется уменьшить число операций ввода-вывода. Примеры включают файловые системы NTFS, ReiserFS.

# Определение Дерева, виды деревьев

Дерево — это иерархическая структура данных, состоящая из узлов, где каждый узел содержит некоторую информацию (ключ) и ссылки на его дочерние узлы. Корень дерева является верхним узлом, а листья - самыми нижними узлами, которые не имеют дочерних элементов.

Основные характеристики дерева:

* **Корень:** Верхний узел дерева, от которого начинается структура.
* **Узлы:** Элементы дерева, каждый из которых может иметь ноль или более дочерних узлов.
* **Дочерние узлы:** Узлы, которые непосредственно связаны с данным узлом.
* **Листья:** Узлы, которые не имеют дочерних узлов.
* **Ключи:** Информация, хранящаяся в каждом узле, используемая для навигации и поиска.

Обычные деревья часто используются для организации данных, таких как файловые системы, иерархии каталогов, структуры документов и т. д.

## Виды деревьев

1. **АВЛ-дерево:** Сбалансированное бинарное дерево, где для каждого узла разность высот его двух поддеревьев (балансный фактор) не превышает единицы. Используется для эффективного поиска, вставки и удаления элементов.
2. **Матричное дерево:** Двоичное дерево, в котором каждый узел имеет до двух потомков, но в отличие от обычного бинарного дерева, потомки узла хранятся в виде матрицы, что позволяет эффективно представлять разреженные структуры данных.
3. **Идеально сбалансированное дерево:** Двоичное дерево, в котором все уровни заполнены наиболее полно, за исключением, возможно, последнего, которое заполняется слева направо. Такое дерево обеспечивает минимальную высоту для заданного числа узлов.
4. **Расширяющееся дерево:** Структура данных, которая автоматически расширяется при необходимости добавления новых элементов. Обычно используется в динамических структурах данных, таких как хеш-таблицы.
5. **B+-деревья:** Вариация B-дерева, где все ключи хранятся только в листовых узлах, что упрощает операции поиска и диапазонные запросы.
6. **2-3-дерево:** Сбалансированное дерево, в котором каждый узел может содержать либо два, либо три дочерних узла. Это обеспечивает более равномерное распределение данных и более эффективные операции поиска.
7. **R-дерево:** Специализированное дерево для организации пространственных данных, таких как точки, линии или многоугольники. Обеспечивает эффективную индексацию и поиск географических объектов.
8. **B\*-дерево**: Расширение B-дерева, где уровень заполненности узлов должен быть не менее 2/3, что улучшает производительность и оптимизирует использование памяти.
9. **Красно-чёрное дерево:** Самобалансирующееся двоичное дерево поиска, где каждый узел имеет дополнительный бит цвета, который используется для обеспечения баланса. Обеспечивает эффективное выполнение операций вставки, удаления и поиска.

## Определение B-tree

B-tree (Б-дерево) - это сбалансированная структура данных, которая обеспечивает эффективное хранение и операции с отсортированными данными. Она используется для поиска, вставки, удаления и обхода данных в упорядоченном виде. B-tree характеризуется следующими особенностями:

* Сбалансированность: Все пути от корня до листьев одинаковой длины, что обеспечивает равномерное распределение данных и быстрый доступ к элементам.
* Коэффициент заполнения: В каждом узле B-tree содержится не менее t-1 и не более 2t-1 ключей, где t - параметр, называемый степенью дерева. Обычно он выбирается так, чтобы узлы занимали один блок памяти.
* Упорядоченность ключей: Ключи в каждом узле упорядочены, что обеспечивает возможность быстрого бинарного поиска.
* Структура узлов: B-tree состоит из корневого узла, внутренних узлов и листовых узлов. Внутренние узлы содержат ключи и указатели на дочерние узлы, а листовые узлы содержат сами данные.

## Разница между B+-tree, B\*-tree

Чтобы подробнее разобраться в этом вопросе, приведем таблицу:

| **Основа сравнения** | **В-дерево** | **B+ дерево** | **B\* дерево** |
| --- | --- | --- | --- |
| **Указатели** | Все внутренние и конечные узлы имеют указатели данных. | Только конечные узлы имеют указатели данных. | То же, что и в B+ дереве. |
| **Поиск** | Поскольку на листе доступны не все ключи, поиск часто занимает больше времени. | Все ключи находятся в конечных узлах, поэтому поиск происходит быстрее и точнее. |
| **Резервные ключи** | В дереве не сохраняются дубликаты ключей. | Сохраняются дубликаты ключей, и все узлы присутствуют на листе. |
| **Вставка** | Вставка занимает больше времени и иногда непредсказуема. | Вставка проще, а результаты всегда одинаковы. |
| **Удаление** | Удаление внутреннего узла очень сложное, и дерево должно претерпеть множество преобразований. | Удалить любой узел легко, поскольку все узлы находятся на листе. |
| **Листовые узлы** | Листовые узлы не хранятся в виде структурного связанного списка. | Листовые узлы хранятся в виде структурного связанного списка. |
| **Доступ** | Последовательный доступ к узлам невозможен. | Последовательный доступ возможен так же, как связанный список. |
| **Высота** | Для определенного количества узлов высота больше | Высота меньше, чем у дерева B для того же количества узлов. |
| **Приложение** | B-деревья, используемые в базах данных и поисковых системах. | Деревья B+, используемые в многоуровневом индексировании, индексировании баз данных |
| **Количество узлов** | Количество узлов на любом промежуточном уровне 'l' равно 2 l. | Каждый промежуточный узел может иметь от n/2 до n дочерних узлов. |

Как мы можем заметить, по основным критериям отличаются только B-дерево и B+-дерево. Разницы же между B+-деревом и B\*-деревом по приведенным критериям нету.

Тем не менее, разница между ними все же есть.

B\*-дерево — разновидность B-дерева, в которой каждый узел дерева заполнен не менее чем на 2/3 (в отличие от B-дерева, где этот показатель составляет 1/2).

В этом определении и кроется главное отличие в их реализации.

B\*-дерево относительно компактнее, так как каждый узел используется полнее. В остальном же этот вид деревьев не отличается от простого B-дерева.

Для выполнения требования «заполненность узла не менее 2/3», приходится отказываться от простой процедуры разделения переполненного узла. Вместо этого происходит «переливание» в соседний узел. Если же и соседний узел заполнен, то ключи приблизительно поровну разделяются на 3 новых узла.

B+-дерево, удовлетворяющее таким требованиям, называется B\*-деревом.

# Средство разработки – Rider

## Плюсы среды разработки

* **Кроссплатформенность:** Rider поддерживает разработку на различных операционных системах, включая Windows, macOS и Linux, что обеспечивает гибкость и удобство для разработчиков.
* **Интеграция с ReSharper:** Rider включает в себя многие функции ReSharper, что обеспечивает мощные инструменты для рефакторинга, анализа кода и автоматической поддержки.
* **Удобная отладка:** Rider предоставляет интуитивно понятные средства отладки, включая поддержку точек останова, просмотр значений переменных и трассировку стека, что упрощает процесс поиска и исправления ошибок.
* **Поддержка языков:** Rider поддерживает широкий спектр языков программирования, включая C#, F#, VB.NET, JavaScript, TypeScript, HTML, CSS и другие, что делает его универсальным инструментом разработки для различных проектов.
* **Интеграция с системами контроля версий:** Rider обеспечивает интеграцию с популярными системами контроля версий, такими как Git, SVN и Mercurial, что облегчает отслеживание изменений в коде и совместную работу над проектом.

## Особенности

* **Расширяемость:** Rider поддерживает расширение функциональности с помощью плагинов, что позволяет настраивать и дополнять среду разработки под индивидуальные потребности.
* **Удобный интерфейс:** Интерфейс Rider интуитивно понятен и удобен в использовании, что облегчает работу с проектами любого размера.
* **Быстрая скорость работы:** Rider обладает высокой производительностью и отзывчивостью, что позволяет разработчикам эффективно работать с большими проектами.

## Возможности

* **Автоматическое дополнение кода:** Rider предлагает автодополнение кода на основе контекста, что ускоряет процесс написания кода и уменьшает вероятность ошибок.
* **Интегрированный терминал:** Встроенный терминал Rider позволяет выполнять команды и скрипты прямо из среды разработки, что облегчает управление проектом.
* **Анализ кода:** Rider предоставляет мощные инструменты для анализа кода, включая поиск ошибок, предупреждений и непотребного кода, что помогает поддерживать высокое качество кода проекта.
* **Поддержка технологий:** Rider поддерживает различные технологии разработки, включая ASP.NET, Xamarin, Unity и другие, что делает его универсальным инструментом для разработки различных типов приложений.

Среда разработки Rider предоставляет широкий спектр возможностей и инструментов, делая процесс разработки более удобным, эффективным и продуктивным для разработчиков.

# Проектирование

## Логика приложения

* Приложение будет реализовывать структуры данных B-tree, B+-tree и B\*-tree.
* Для каждой структуры данных будет реализован класс, который позволит выполнять основные операции, такие как вставка, удаление и поиск элементов.
* Каждая реализация будет учитывать особенности соответствующей структуры данных, такие как специфика разделения узлов при переполнении, уровень заполненности узлов и т. д.

## Требования к приложению

* Понятный и интуитивно понятный интерфейс для взаимодействия с пользователями.
* Возможность выбора типа структуры данных (B-tree, B+-tree, B\*-tree).
* Операции вставки, удаления и поиска элементов должны быть реализованы для каждой структуры данных.
* Возможность отображения структуры дерева для наглядного представления данных.

## Схематичное описание функций приложения

## Логика приложения

## Требования к приложению

## Код алгоритма и этапы его работы

# Тестирование

# Документация

## Характеристики используемой техники

## Краткое руководство пользователя

# Заключение

# Список литературы

1. Bayer, R., & McCreight, E. M. (1972). Organization and maintenance of large ordered indexes. Acta Informatica, 1(3), 173-189.
2. Comer, D. (1979). The Ubiquitous B-Tree. ACM Computing Surveys (CSUR), 11(2), 121-137.
3. Garcia-Molina, H., & Ullman, J. D. (1992). Database systems: the complete book. Pearson Education India.
4. O'Neil, P., O'Neil, E., & Weikum, G. (1996). The LRU-K page replacement algorithm for database disk buffering. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data (pp. 297-306).
5. Litwin, W., Neimat, M., & Schneider, D. (1994). LH\* tree: A general structure for indexing multidimensional information. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94) (pp. 212-223).
6. Leutenegger, S. T., & Lopez, M. A. (1997). B\*-trees and fragments: New and improved database methods for XML-like documents. ACM Transactions on Database Systems (TODS), 22(3), 256-296.
7. Stonebraker, M., Rowe, L., & Lindsay, B. (1986). The design of POSTGRES. In Proceedings of the 1986 ACM SIGMOD international conference on Management of data (pp. 340-355).
8. Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD international conference on Management of data (pp. 47-57).
9. Mehta, S., & Sartaj, S. (2004). Data Structures Using C and C++. PHI Learning Pvt. Ltd.