ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

КУРСОВОЙ ПРОЕКТ

по дисциплине «Объектно-ориентированное программирование»

«B$^+$-дерево на C++»

Пояснительная записка

Выполнил
студент гр. 3331506/20401 _____ (подпись) Макаров И. М.

Работу принял _____ (подпись) Ананьевский М. С.

Санкт-Петербург
2025 г.

Содержание

1 Индивидуальное задание **6**

2 Ход работы **7**
2.1 Изучение структуры данных «B$^+$-дерево» 7
2.2 Изучение алгоритмов поиска, вставки, удаления 8
2.2.1 Поиск в B$^+$-дереве 8
2.2.2 Вставка в B$^+$-дерево 8
2.2.3 Удаление в B$^+$-дереве 8
2.3 Программная реализация B$^+$-дерева 9

3 Реализация **10**
3.1 Архитектура проекта 10
3.2 Архитектура методов класса **BplusTree** 11
3.3 Архитектура класса **BPlusTreeNode** 12

4 Список использованных источников **13**

1 Индивидуальное задание

Цель: Реализация библиотеки для работы со структурой данных B^+-дерево на языке C++ с поддержкой основных операций: поиска, вставки и удаления элементов.

2 Ход работы

2.1 Изучение структуры данных «B$^+$-дерево»

B$^+$-дерево — структура данных на основе B-дерева, сбалансированное n-арное дерево поиска с переменным, но зачастую большим количеством потомков в узле. B$^+$-дерево состоит из корня, внутренних узлов и листьев; корень может быть либо листом, либо узлом с двумя и более потомками.

Изначально структура предназначалась для хранения данных в целях эффективного поиска в блочно-ориентированной среде хранения — в частности, для файловых систем. Применение связано с тем, что в отличие от бинарных деревьев поиска, B$^+$-деревья имеют очень высокий коэффициент ветвления (число указателей из родительского узла на дочерние — обычно порядка 100 или более), что снижает количество операций ввода-вывода, требующих поиска элемента в дереве.

Вариант B$^+$-дерева, в котором все значения сохранялись в листовых узлах, систематически рассмотрен в 1979 году, при этом отмечено, что такие структуры использовались IBM в технологии файлового доступа для мейнфреймов VSAM по крайней мере с 1973 года.

Структура широко применяется в файловых системах:

- NTFS, ReiserFS, NSS, XFS, JFS, ReFS и BFS используют этот тип дерева для индексирования метаданных
- BeFS использует B$^+$-деревья для хранения каталогов

Реляционные СУБД, поддерживающие B$^+$-деревья:

- DB2, Informix, Microsoft SQL Server
- Oracle Database (начиная с версии 8)
- Adaptive Server Enterprise и SQLite

Среди NoSQL-СУБД, работающих с моделью «ключ-значение», структура реализована в:

- CouchDB
- MongoDB (при использовании подсистемы хранения WiredTiger)
- Tokyo Cabinet

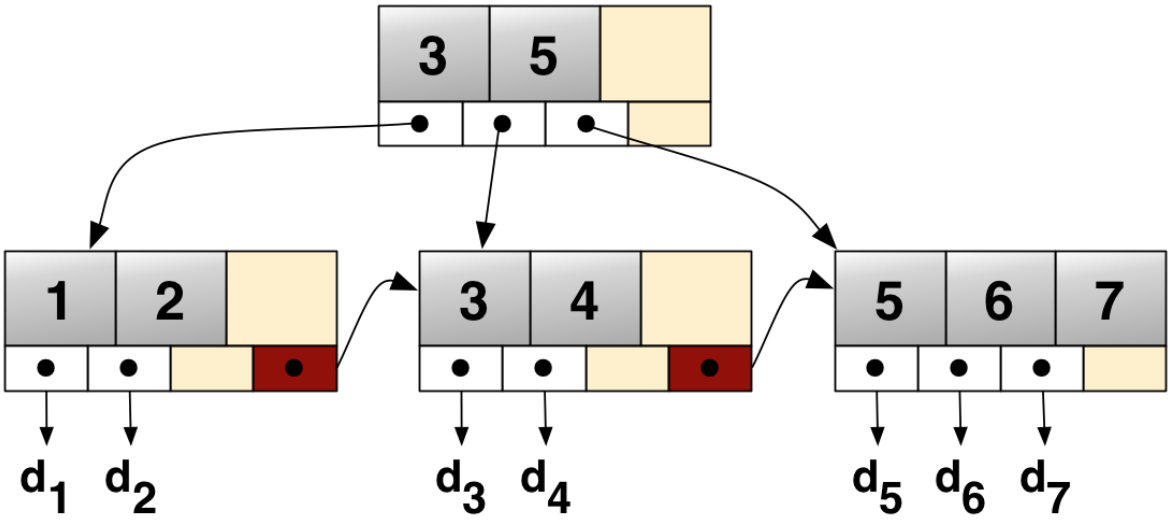

Рис. 1: B$^+$-дерево

2.2 Изучение алгоритмов поиска, вставки, удаления

Все операции поддерживают строгую сбалансированность дерева, обеспечивая предсказуемую производительность. Листовые узлы связаны в односвязный список для эффективного диапазонного поиска.

2.2.1 Поиск в B$^+$-дереве

Поиск начинается с корня. На каждом внутреннем узле выбирается поддерево по диапазону ключей, пока не будет достигнут лист, где происходит проверка наличия искомого ключа. Если ключ найден, возвращается соответствующее значение, иначе — nullptr. Поиск гарантированно работает за O(log n) благодаря сбалансированности дерева.

2.2.2 Вставка в B$^+$-дерево

Вставка сначала находит целевой лист через обычный поиск. Если в листе есть свободное место, ключ и значение добавляются в отсортированную позицию. При переполнении лист разделяется на два: левый сохраняет первые n элементов, правый получает остальные, а средний ключ поднимается в родительский узел. Если переполнение доходит до корня, создаётся новый корень, увеличивая высоту дерева. Вставка также выполняется за O(log n).

2.2.3 Удаление в B$^+$-дереве

Удаление сначала локализует ключ в листе и удаляет его. Если после удаления в узле остаётся меньше n элементов, система пытается заимствовать элемент у соседнего узла того же уровня. При невозможности заимствования происходит слияние узлов с корректировкой ключей в родительских узлах. Если слияние затрагивает корень и приводит к его опустошению, высота дерева уменьшается. Удаление сохраняет баланс и работает за O(log n).

2.3 Программная реализация B^+-дерева

3 Реализация

3.1 Архитектура проекта

Структура проекта организована следующим образом:

```
/
└── B_PLUS_TREE/                    (корневая директория проекта)
    ├── vscode/                     (конфигурация для VS Code)
    ├── build/                      (директория сборки)
    ├── include/                    (заголовочные файлы)
    ├── src/                        (исходные файлы реализации)
    ├── CMakeLists.txt              (конфигурация сборки CMake)
    ├── main.cpp                    (главный исполняемый файл)
    ├── README.md                   (документация проекта)
    └── test_data.txt               (тестовые данные)
```

- `B_PLUS_TREE` - корневая директория проекта
- `build/` - директория для файлов сборки (генерируется при компиляции)
- `include/` - содержит заголовочные файлы (.h, .hpp) с объявлениями классов и функций
- `src/` - содержит файлы реализации (.cpp) с исходным кодом
- `CMakeLists.txt` - основной конфигурационный файл системы сборки CMake
- `main.cpp` - точка входа в программу
- `README.md` - файл с основной документацией проекта
- `test_data.txt` - файл с тестовыми данными для проверки работы B^+-дерева

3.2 Архитектура методов класса BplusTree

Приватные методы

- **delete_node** - Рекурсивное удаление поддерева с очисткой памяти. Основа деструктора.
- **find_leaf** - Поиск целевого листового узла для заданного ключа с обходом от корня.
- **split_leaf** - Обработка переполнения листа: разделение с созданием нового узла и перераспределением ключей.
- **split_internal** - Разделение переполненного внутреннего узла с сохранением структуры дерева.
- **fix_internal_underflow** - Восстановление свойств дерева после удаления (перераспределение/слияние узлов).
- **print_tree** - Вспомогательный метод визуализации иерархии узлов с отступами.

Публичный интерфейс

- **Конструктор/Деструктор** - Инициализация пустого дерева и корректное освобождение ресурсов.
- **search** - Проверка существования ключа в структуре.
- **insert** - Добавление ключа с автоматической балансировкой (вызов split-методов при необходимости).
- **remove** - Удаление ключа с поддержанием свойств дерева через fix-методы.
- **print_tree** - Публичный метод отображения текущего состояния структуры.

3.3 Архитектура класса BPlusTreeNode

Публичные поля

- **keys** - Вектор отсортированных ключей узла:
  - Хранит значения ключей в узле
  - Всегда поддерживается в отсортированном состоянии
- **children** - Вектор указателей на дочерние узлы:
  - Для внутренних узлов: содержит M потомков
  - Для листовых узлов: пустой вектор
- **next** - Указатель на следующий лист:
  - Актуален только для листовых узлов
  - Образует односвязный список листьев
  - Упрощает последовательный обход значений

Конструкторы и деструкторы

- **BPlusTreeNode()** - Конструктор по умолчанию:
  - Инициализирует указатель `next` значением `nullptr`
  - Векторы `keys` и `children` инициализируются автоматически
- **$\tilde{\text{B}}$PlusTreeNode()** - Деструктор:
  - Стандартное поведение (не удаляет дочерние узлы рекурсивно)
  - Управление памятью осуществляется классом `BPlusTree`

Методы узла

- **is_leaf()** - Проверка типа узла:
  - Возвращает `true` если узел листовой (нет дочерних узлов)
  - Критерий: пустой вектор `children`
- **get_parent()** - Поиск родительского узла:
  - Принимает корень дерева как входной параметр
  - Рекурсивно обходит дерево для поиска родителя
  - Возвращает `nullptr` для корневого узла

4 Список использованных источников

[1] NITC Base. *B+ Trees*. 2021.
https://nitcbase.github.io/docs/Misc/B+%20Trees/.

[2] GeeksForGeeks Team. *C++ Program to Implement B+ Tree*. GeeksForGeeks.
https://www.geeksforgeeks.org/cpp-program-to-implement-b-plus-tree/.

[3] Wikipedia. *B+ tree*. Англ. 2023.
https://en.wikipedia.org/wiki/B%2B_tree.

[4] Википедия. *B+-дерево*. 2023.
https://ru.wikipedia.org/wiki/B%E2%81%BA-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE.