МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ДОКЛАД
о практическом задании по дисциплине АиСД
«Сравнение деревьев: B-tree, B+-tree и B*-tree»

направление подготовки 09.03.03 «Прикладная информатика»
профиль «Прикладная информатика в компьютерном дизайне»

Студент группы
Б9121-09.03.03пикд
Соченко Алёна Сергеевна
_____
(подпись)
«_____» _____ 2023г.

Доклад защищен
с оценкой _____

Руководитель практики
Доцент ИМКТ А.С. Кленин
_____
(подпись)
«_____» _____2023г.

Регистрационный номер _____
«____» _____2023г.

г. Владивосток
2023

Содержание

Аннотация

В данном докладе рассматриваются формы деревьев поиска – B-tree, B+-tree, B*-tree, описывается принцип работы и реализация этих структур данных. Исследованию и сравнению подлежит их производительность.

1. Введение

B-tree — структура данных, сбалансированное, сильно ветвистое дерево поиска, предложенное Р. Бэйером и Э. МакКрейтом в 1970 году для хранения данных во внешней памяти [1].

B+-tree — структура данных на основе B-дерева, сбалансированное 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 [22].

B*-tree — разновидность B-дерева, в которой каждый узел дерева заполнен не менее чем на ⅔ (в отличие от B-дерева, где этот показатель составляет 1/2). B*-деревья предложили Рудольф Байер и Эдвард МакКрейт, изучавшие проблему компактности B-деревьев.

1.1. Глоссарий

B-tree — структура данных, сбалансированное, сильно ветвистое дерево поиска, предложенное Р. Бэйером и Э. МакКрейтом в 1970 году для хранения данных во внешней памяти [1].

B+-tree — структура данных на основе B-дерева, сбалансированное 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 [2].

B*-tree — разновидность B-дерева, в которой каждый узел дерева заполнен не менее чем на ⅔ (в отличие от B-дерева, где этот показатель составляет 1/2). B*-деревья предложили Рудольф Байер и Эдвард МакКрейт, изучавшие проблему компактности B-деревьев [3].

Бинарное дерево поиска — это дерево, в котором элементы размещаются согласно определенному правилу (упорядоченно): Каждый узел должен быть "больше" любого элемента в своем левом поддереве.

Дерево поиска — это древовидная структура данных, используемая для поиска определенных ключей из набора. Для того, чтобы дерево функционировало как дерево поиска, ключ для каждого узла должен быть больше любых ключей в поддеревьях слева и меньше любых ключей в поддеревьях справа.

Сбалансированное дерево поиска — такое двоичное дерево поиска, в котором высота каждого из поддеревьев, имеющих общий корень, отличается не более чем на некоторую константу k, и при этом выполняются условия характерные для двоичного дерева поиска.

Уровень вершины — вертикальная высота соответствующей вершины.

Эффективность алгоритма — свойство алгоритма, связанное с вычислительными ресурсами, используемыми алгоритмом.

1.2. Постановка цели и задач

Цель: разработать и описать реализацию алгоритмов B-tree, B+-tree и B*-tree. Протестировать данные алгоритмы с помощью автоматических тестов и сравнить результаты.

Задачи:

1. Изучить теоретический материал по B-tree, B+-tree и B*-tree
2. Описать B-tree, B+-tree и B*-tree
3. Реализовать B-tree
4. Продумать тесты, подходящие для каждого вида дерева
5. Реализовать тесты к B-tree
6. Реализовать B+-tree
7. Реализовать B*-tree

8. Описать результаты тестирования B-tree, B+-tree и B*-tree

1.3. Описание предметной области

Направление исследований: сравнение B-tree, B+-tree и B*-tree, нахождение ситуаций, когда эффективнее тот или иной алгоритм.

Эффективность алгоритма будет оцениваться по времени, затраченному на тот или иной тест.

В бинарном дереве поиска каждый узел содержит лишь одно значение (ключ) и не более 2-х потомков. Но существует особый вид древа поиска, называемый B-дерево (Би-дерево). Здесь узел содержит больше одного значения и больше 2-х потомков. Также его называют сбалансированным по высоте деревом поиска порядка m (Height Balanced m-way Search Tree). B-дерево представляет собой сбалансированное дерево поиска, где каждый узел содержит много ключей и имеет больше двух потомков.

В B+-деревьях данные ключей хранятся только в листовых узлах, а в промежуточных хранятся копии значений ключей. Это помогает эффективно организовывать поиск в блочных средах хранения. Например, в файловых системах, где B+-деревья применяют, чтобы хранить каталоги.

B*-дерево ориентировано на более плотное хранение ключей во внутренних узлах. Этот вариант гарантирует, что некорневые узлы заполнены как минимум на ⅔m вместо m/2.

1.4. Обзор существующих методов решения

Существуют готовые реализации рассматриваемых структур данных.
B-дерево — браузерная реализация
Возможности:

- Визуализация данных;
- Добавление, удаление, поиск элемента;
- Выбор скорости анимации.

B+-дерево — браузерная реализация
Возможности:

- Визуализация данных;
- Добавление, удаление, поиск элемента;
- Выбор скорости анимации.

2. Спецификация данных

B-дерево порядка m — это дерево, которое удовлетворяет следующим свойствам [13]:

1. Каждый узел имеет не более m дочерних элементов;
2. Каждый внутренний узел имеет не менее ⌈m/2⌉ дочерних элементов;
3. У каждого нелистового узла есть как минимум два дочерних узла;
4. Все листья появляются на одном уровне;
5. Нелистовой узел с k дочерними элементами содержит k−1 ключей.

Ключи каждого внутреннего узла действуют как разделительные значения, которые разделяют его поддеревья. Например, если внутренний узел имеет 3 дочерних узла (или поддерева), то он должен иметь 2 ключа: a_1 и a_2. Все значения в крайнем левом поддереве будут меньше a_1, все значения в среднем поддереве будут между a_1 и a_2, а все значения в крайнем правом поддереве будут больше a_2 [21].

3. Функциональные требования

Разработанные структуры данных должны:

- представлять собой библиотеку-класс с определёнными методами:
  - вставки
  - удаления
  - поиска
- хранить данные

Тесты должны:

- полностью покрывать функционал программы;
- быть автоматически генерируемыми (для тестов производительности)

По завершении теста должна выводиться информация о затраченном времени.

4. Формальное описание алгоритмов

B, B+ и B* деревья являются модификациями бинарного дерева поиска, а также принадлежат к классу сбалансированных структур данных.

Обращаясь к дереву, мы подразумеваем обращение к его корневому узлу. Дерево, корневой узел которого пустой, является пустым.

Основные операции:

- insert(x) – вставить элемент x, если его ещё не существует в дереве;
- delete(x) – удалить элемент x, если он есть в дереве;
- search(x) – найти элемент x, если он существует в дереве.

4.1. B-tree

B-дерево — сильноветвящееся сбалансированное дерево поиска, позволяющее проводить поиск, добавление и удаление элементов за $O(log\, n)$ [10]. B-дерево с n узлами имеет высоту $O(log\, n)$. Количество детей узлов может быть от нескольких до тысяч (обычно степень ветвления B-дерева определяется характеристиками устройства, на котором производится работа с деревом). B-деревья также могут использоваться для реализации многих операций над динамическими множествами за время $O(log\, n)$. B-дерево является идеально сбалансированным, то есть глубина всех его листьев одинакова. B-дерево имеет следующие свойства (t — параметр дерева, называемый минимальной степенью B-дерева, не меньший 2) [6].

Назначение

B-деревья разработаны для использования на дисках (в файловых системах) или иных энергонезависимых носителях информации с прямым доступом, а также в базах данных [11]. B-деревья похожи на красно-чёрные деревья (например, в том, что все B-деревья с n узлами имеют высоту $O(log\, n)$, но они лучше минимизируют количество операций чтения-записи с диском.

Поиск ключа

Если ключ содержится в текущем узле, возвращаем его. Иначе определяем интервал и переходим к соответствующему сыну. Повторяем пока ключ не найден или не дошли до листа.

Добавление ключа

Ищем лист, в который можно добавить ключ, совершая проход от корня к листьям. Если найденный узел не заполнен, добавляем в него ключ. Иначе разбиваем узел на два узла, в первый добавляем первые t−1 ключей, во второй — последние t−1 ключей. После добавляем ключ в один из этих узлов. Оставшийся средний элемент добавляется в родительский узел, где становится разделительной точкой для двух новых поддеревьев.

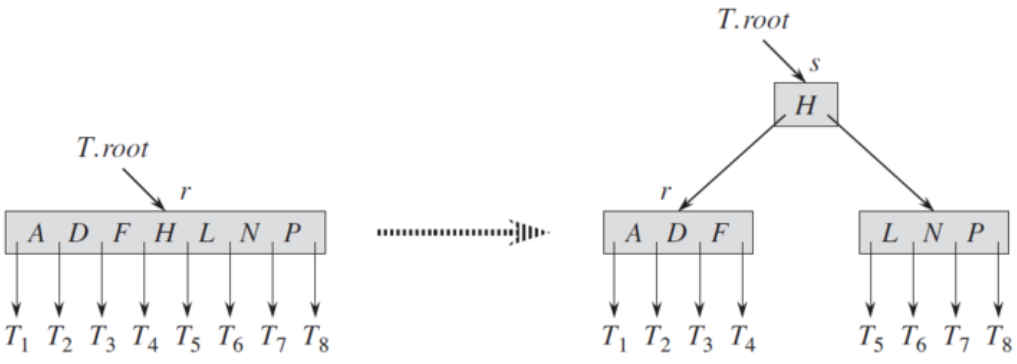

Рисунок 1. Добавление ключа в дерево

Если и родительский узел заполнен — повторяем пока не встретим незаполненный узел или не дойдем до корня. В последнем случае корень разбивается на два узла, и высота дерева увеличивается. Добавление ключа в B-дереве может быть осуществлена за один нисходящий проход от корня к листу. Для этого не нужно выяснять, требуется ли разбить узел, в который должен вставляться новый ключ. При проходе от корня к листьям в поисках места для нового ключа будут разбиваться все заполненные узлы, которые будут пройдены (включая и сам лист). Таким образом, если надо разбить какой-то полный узел, гарантируется, что его родительский узел не будет заполнен.

Вставка ключа в B-дерево T высоты h за один нисходящий проход по дереву потребует O(h) обращений к диску и $O(th)=O(t \log_t n)$ процессорного времени.

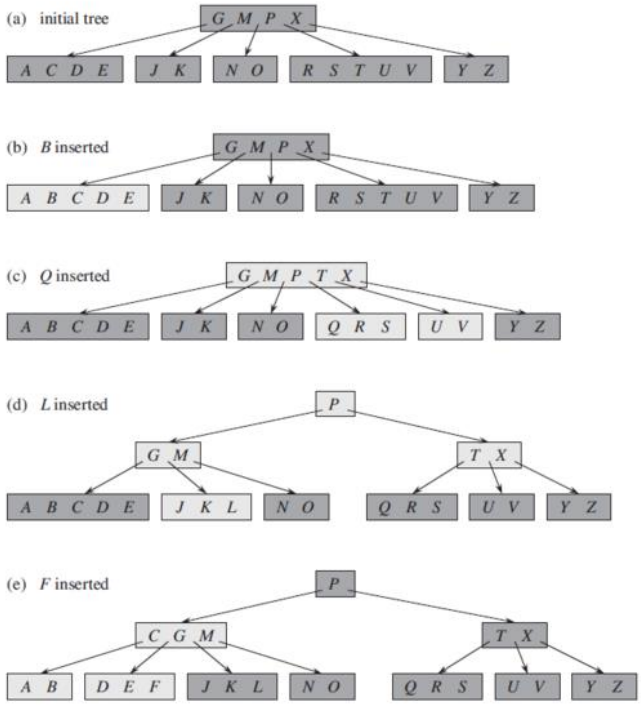

Рисунок 2. Вставка ключей B, Q, L и F в дерево, где узлы не могут содержать 5 ключей

Разбиение узла

Функция разбиения получает в качестве входного параметра незаполненный внутренний узел x (находящийся в оперативной памяти), индекс t и узел y (также находящийся в оперативной памяти), такой что $y=c_i[x]$ является заполненным дочерним узлом x. Процедура разбивает дочерний узел на два и соответствующим образом обновляет поля x, внося в него информацию о новом дочернем узле. Для разбиения заполненного корневого узла мы сначала делаем корень дочерним узлом нового пустого корневого узла, после чего можно вызвать функцию. При этом высота дерева увеличивается на 1.

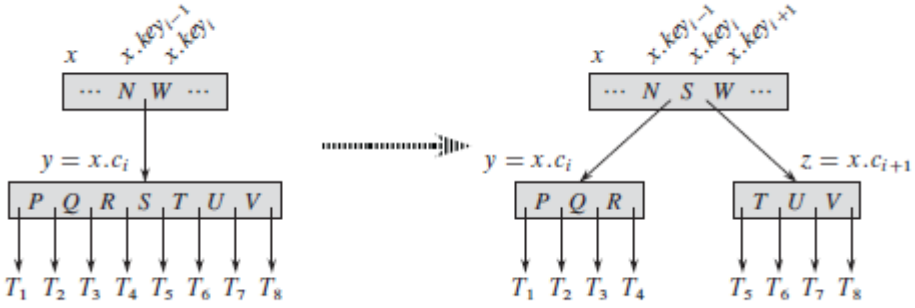

Рисунок 3. Разбиение ключа дерева

Удаление ключа

Операция удаления ключа несколько сложнее, нежели добавление оного, так как необходимо убедиться, что удаляемый ключ находится во внутреннем узле. Процесс похож на поиск подходящего места для вставки ключа, с той разницей, что перед спуском в поддерево проверяется, достаточность количества ключей в нем, а также возможность провести удаление, не нарушив структуры B-дерева. Таким образом, удаление аналогично вставке, и его проведение не потребует последующего восстановления структуры B-дерева. Если поддерево, выбранное поиском для спуска, содержит минимальное количество ключей $t-1$, производится либо перемещение, либо слияние. Для удаления требуется время $O(t \log_t n)$ и $O(h)$ дисковых операций.

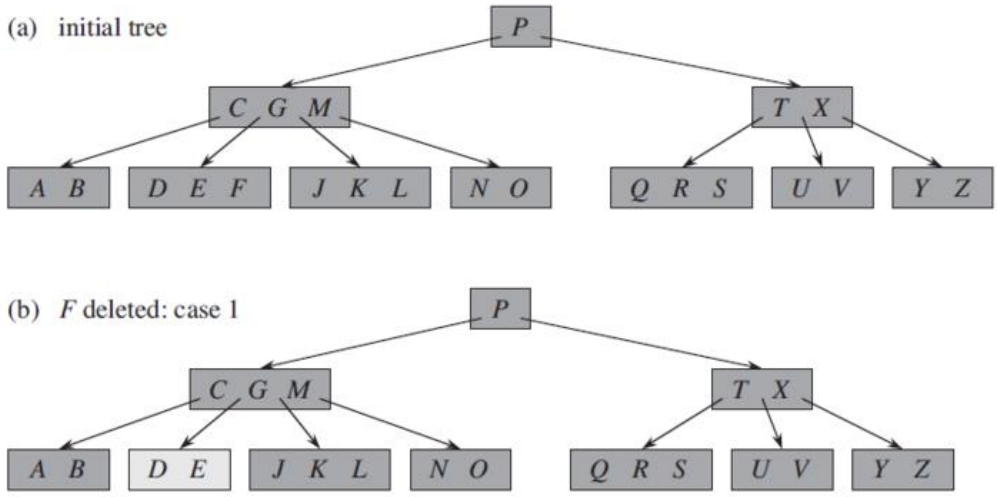

Рисунок 5.Удаление ключа из листа дерева

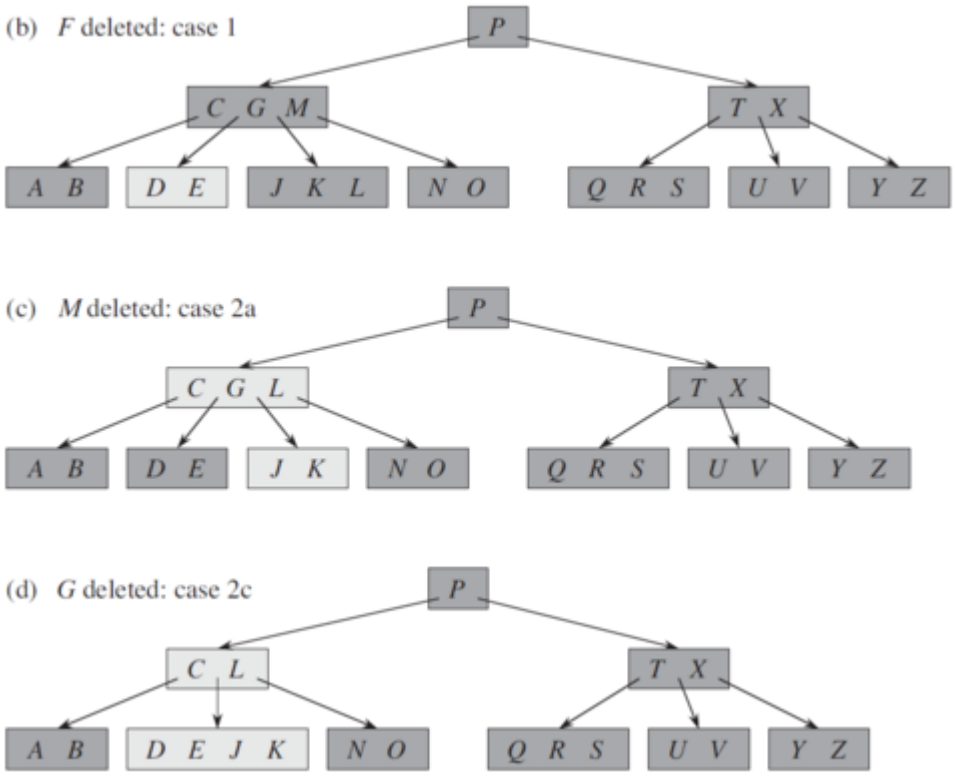

Рисунок 4. Удаление ключа из внутреннего узла дерева

4.2. B+-tree

B+-дерево — структура данных на основе B-дерева, сбалансированное n-арное дерево поиска с переменным, но зачастую большим количеством потомков в узле. 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. Асимптотика операций вставки, поиска и удаления элемента – $O(log\, n)$.

Отличия от B-дерева

В B-дереве во всех вершинах хранятся ключи вместе с сопутствующей информацией. В B+-деревьях вся информация хранится в листьях, а во внутренних узлах хранятся только копии ключей. Таким образом удается получить максимально возможную степень ветвления во внутренних узлах. Кроме того, листовой узел может включать в себя указатель на следующий листовой узел для ускорения последовательного доступа, что решает одну из главных проблем B-деревьев.

Структура узла

```
struct Node
   bool leaf // является ли узел листом
   int key_num // количество ключей узла
   int key [] // ключи узла
   Node parent // указатель на отца
   Node child [] // указатели на детей узла
   Info pointers [] // если лист — указатели на данные
   Node left // указатель на левого брата
   Node right // указатель на правого брата
```

Структура дерева

```
struct BPlusTree
   int t // минимальная степень дерева
   Node root // указатель на корень дерева
```

Поиск

Находим нужный лист и ищем нужный ключ в нем.

Добавление ключа

Ищем лист, в который можно добавить ключ и добавляем его в список ключей. Если узел не заполнен, то добавление завершено. Иначе разбиваем узел на два узла. Будем считать, что в дереве не может находиться 2 одинаковых ключа, поэтому insert будет возвращать информацию о том, был ли добавлен ключ [17].

Разбиение узла

Разбиение на два узла происходит следующим образом: в первый добавляем первые t ключей, во второй последние t−1. Если узел — лист, то оставшийся ключ также добавляется в правое поддерево, а его копия отправляется в родительский узел, где становится разделительной точкой для двух новых поддеревьев.

Если и родительский узел заполнен — поступаем аналогично, но не копируем, а просто перемещаем оставшийся перемещаем ключ в родительский узел, так как это просто копия.

Повторяем пока не встретим незаполненный узел или не дойдем до корня. В последнем случае корень разбивается на два узла, и высота дерева увеличивается.

Поскольку в родителя всегда отправляется минимальный ключ из второй половины, то каждый ключ, который хранится во внутренней вершине — это минимум правого поддерева для этого ключа.

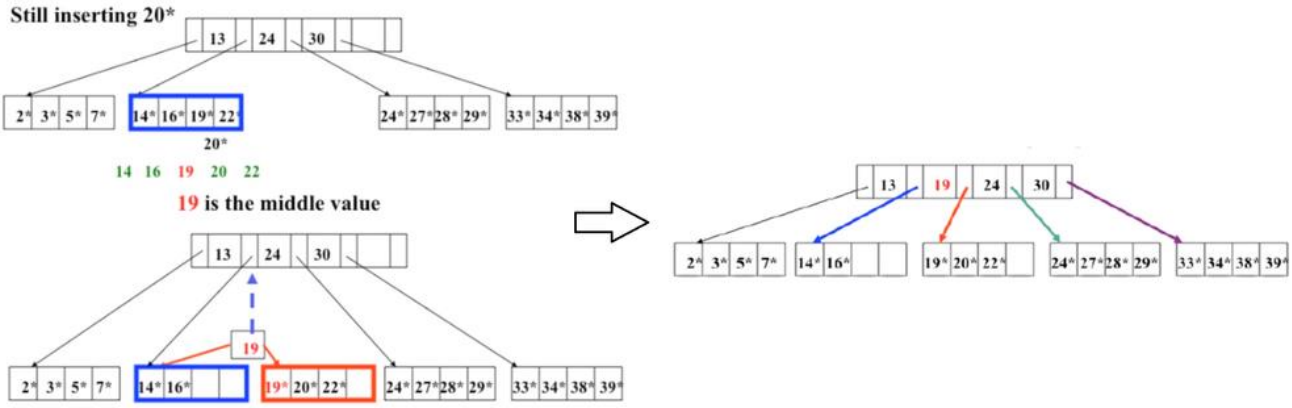

Рисунок 6. Разбиение узла B+-дерева

Удаление ключа

Поскольку все ключи находятся в листах, для удаления в первую очередь необходимо найти листовой узел, в котором он находится. Если узел содержит не менее $t-1$ ключей, где t — это степень дерева, то удаление завершено. Иначе необходимо выполнить попытку перераспределения элементов, то есть добавить в узел элемент из левого или правого брата (не забыв обновить информацию в родителе). Если это невозможно, необходимо выполнить слияние с братом и удалить ключ, который указывает на удалённый узел. Объединение может распространяться на корень, тогда происходит уменьшение высоты дерева. Так как мы считаем, что в дереве не может находиться 2 одинаковых ключа, то delete будет возвращать информацию о том, был ли удален ключ.

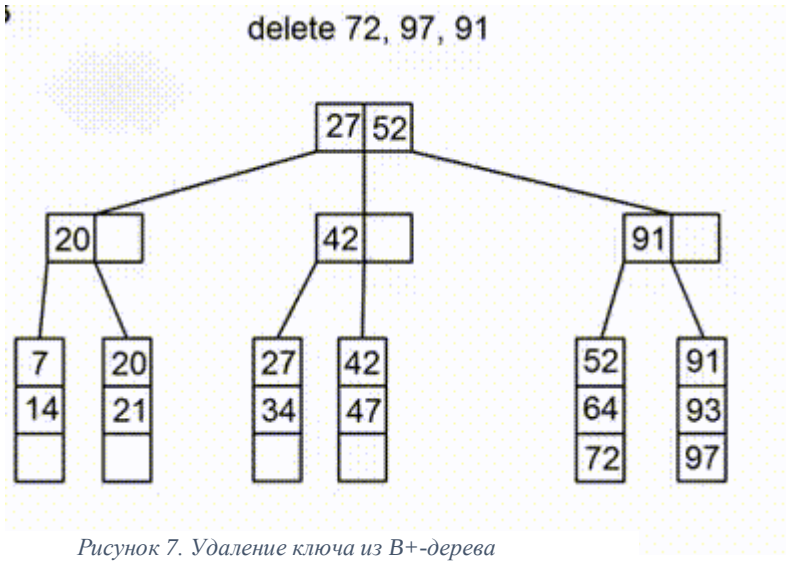

Рисунок 7. Удаление ключа из B+-дерева

4.3. B*-tree

B*-дерево — разновидность B-дерева, в которой каждый узел дерева заполнен не менее чем на ⅔ (в отличие от B-дерева, где этот показатель составляет 1/2).

B*-деревья предложили Рудольф Байер и Эдвард МакКрейт, изучавшие проблему компактности B-деревьев. 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 [9].

5. Реализация

5.1. Средства реализации

В качестве языка программирования был выбран C++. В качестве среды разработки выбран Visual Studio Code, так как он является одним из самых популярных средств написания кода.

5.2. Структуры данных

Минимальной структурной единицей во всех структурах данных является узел Node [23].

Структура узла

```
struct BTreeNode {
   int *data;
   BTreeNode **child_ptr;
   bool leaf;
   int n;
}
```

6. Реализация и тестирование

Физические характеристики текущей системы:

- Язык программирования алгоритма — C++;
- Объём написанного кода для B-дерева — 10 Кб;
- Объём написанного кода для B-дерева в строках – 365 строк;
- Объём папки с проектом — 91,2 Мб.

Реализация программной системы успешна лишь наполовину из-за возникших сложностей с написанием юнит-тестов для алгоритмов.

6.1. Вычислительный эксперимент

Так как различия между B, B+ и B* деревьями в написании кода минимальны [5], было решено написать полноценный алгоритм лишь для B-дерева. Выдвигаемая гипотеза: структура данных B-tree быстрее выводит строение дерева на экран, но удобнее в использовании всё же структура данных B+-tree.

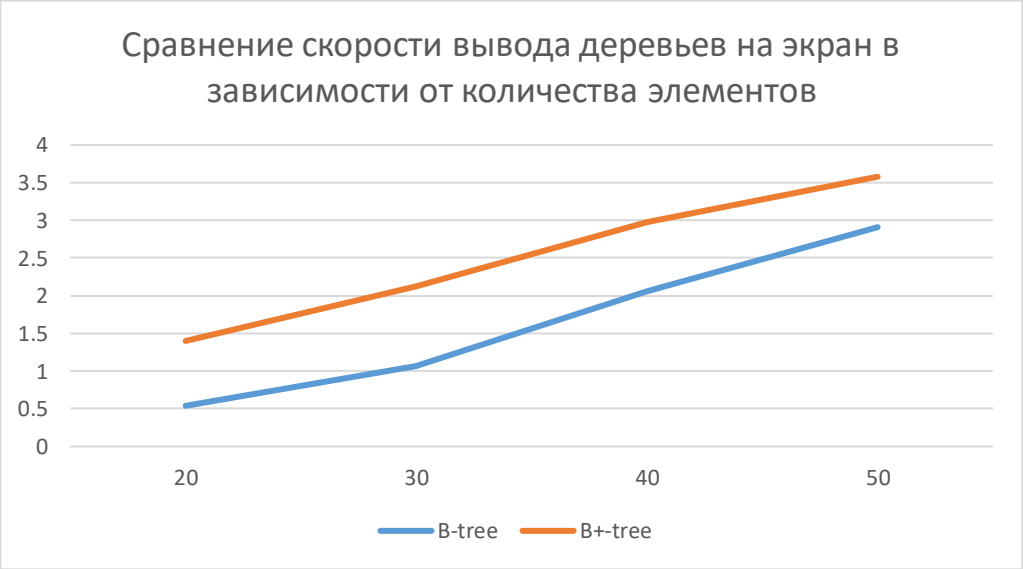

Рисунок 8. Сравнение вывода деревьев на экран в зависимости от количества элементов

Гипотеза подтверждается: в процессе выполнения практической работы можно выявить множество не особо существенных различий, от которых все же зависит производительность алгоритмов.

B-дерево	**B+-дерево**
В дереве B все ключи и записи хранятся как во внутренних, так и в конечных узлах.	В дереве B+ ключами являются индексы, хранящиеся во внутренних узлах, а записи хранятся в конечных узлах.
В дереве B ключи не могут храниться повторно, что означает отсутствие дублирования ключей или записей.	В дереве B+ может быть избыточность в появлении ключей. В этом случае записи хранятся в конечных узлах, тогда как ключи хранятся во внутренних узлах, поэтому во внутренних узлах могут присутствовать избыточные ключи.

В B-tree конечные узлы не связаны друг с другом.	В дереве B+ конечные узлы связаны друг с другом для обеспечения последовательного доступа.
В B-tree поиск не очень эффективен, потому что записи хранятся либо в конечных, либо во внутренних узлах.	В дереве B+ поиск выполняется очень эффективно или быстрее, потому что все записи хранятся в конечных узлах.
Удаление внутренних узлов - очень медленный и трудоемкий процесс, поскольку нам также необходимо учитывать дочерний элемент удаленного ключа.	Удаление в дереве B+ происходит очень быстро, потому что все записи хранятся в конечных узлах, поэтому нам не нужно рассматривать дочерний элемент узла.
В B-tree последовательный доступ невозможен.	В дереве B+ все конечные узлы соединены друг с другом через указатель, поэтому возможен последовательный доступ.
В B-tree выполняется большее количество операций разделения, из-за чего высота увеличивается по сравнению с шириной,	Дерево B+ имеет большую ширину по сравнению с высотой.
В B-tree каждый узел имеет по крайней мере две ветви, и каждый узел содержит несколько записей, поэтому нам не нужно проходить до конечных узлов, чтобы получить данные.	В дереве B+ внутренние узлы содержат только указатели, а конечные узлы содержат записи. Все конечные узлы находятся на одном уровне, поэтому нам нужно пройти до конечных узлов, чтобы получить данные.
Корневой узел содержит от 2 до m дочерних элементов, где m - порядок дерева.	Корневой узел содержит от 2 до m дочерних элементов, где m - порядок дерева.

Заключение

Таким образом, в процессе выполнения практического задания мною были изучены три алгоритма: B-дерево, B+-дерево и B*-дерево. Изучено строение алгоритмов, их отличие друг от друга и производительность алгоритмов на примерах. Углублены знания в области сбалансированных деревьев поиска, повышены навыки в области скорости написания кода.

Был разработан алгоритм B-дерева, который выполняет несколько функций: добавление, удаление, поиск элемента в структуре данных, а также вывод дерева в консоль и выход из программы.

Список литературы

1. Википедия. B-дерево. https://ru.wikipedia.org/wiki/B-дерево
2. Википедия. B+-дерево. https://ru.wikipedia.org/wiki/B+-дерево
3. Википедия. B*-дерево. https://ru.wikipedia.org/wiki/B*-дерево
4. B-деревья. Алгоритмы на деревьях. https://ru.hexlet.io/courses/algorithms-trees/lessons/btrees/theory_unit
5. Различия между деревьями B и B+. https://kzen.dev/ru/50940310
6. Файловые структуры поиска. http://gubsky.ru/study/5/soad/ext/
7. Разница между B Tree и B+ Tree. https://ru.natapa.org/difference-between-b-tree-and-b-plus-tree-2650
8. Бинарное дерево – что это? B-деревья. https://otus.ru/nest/post/1801/
9. B*-Trees implementation in C++. https://www.geeksforgeeks.org/b-trees-implementation-in-c/
10. B-tree. https://habr.com/ru/post/114154/
11. Структура данных B-дерево. https://habr.com/ru/company/otus/blog/459216/
12. Анализ и реализация требований алгоритма B-дерева. https://russianblogs.com/article/7345459005/
13. B-дерево. https://neerc.ifmo.ru/wiki/index.php?title=B-дерево#.D0.A0.D0.B0.D0.B7.D0.B1.D0.B8.D0.B5.D0.BD.D0.B8.D0.B5_.D1.83.D0.B7.D0.BB.D0.B0
14. B-дерево. https://codechick.io/tutorials/dsa/dsa-b-tree
15. Б, Б+ и Б++ деревья. https://algolist.manual.ru/ds/s_btr.php
16. Лекция 4 - B-дерево. https://studfile.net/preview/16403690/
17. Insertion in a B+ tree. https://www.geeksforgeeks.org/insertion-in-a-b-tree/
18. B-tree. https://www.programiz.com/dsa/b-tree
19. B and B+ Trees. https://www.geeksforgeeks.org/data-structure-gq/b-and-b-trees-gq/
20. Различия между деревьями B и B+. https://stackru.com/questions/38765748/razlichiya-mezhdu-derevyami-b-i-b
21. B-дерево - B-tree. https://ru.wikibrief.org/wiki/B-tree
22. Algorithm Implementation/Trees/B+ tree. https://en.wikibooks.org/wiki/Algorithm_Implementation/Trees/B%2B_tree
23. Примечания к изучению бинарного дерева B-дерева, B+ tree, B* tree. https://russianblogs.com/article/50041274143/
24. Using Precompiled Headers. http://gcc.gnu.org/onlinedocs/gcc/Precompiled-Headers.html
25. B+ Trees Data Structure. https://www.studytonight.com/advanced-data-structures/b-plus-trees-data-structure
26. Сбалансированное дерево поиска B-tree (t=2).https://habr.com/ru/post/337594/
27. Юнит-тесты. Быстрый старт – эффективный результат (с примерами на C++). https://habr.com/ru/company/tensor/blog/347358/