**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра дискретной математики и алгоритмики

**Применение B-деревьев для эффективных операций в базах данных**

Курсовой проект

Мелеха Алексея Александровича

студента 3 курса,

специальности “Информатика”

Научный руководитель:

Лукьянов Иван Денисович

Ассистент кафедры

ДМА

Минск, 2020

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc58943657)

[ГЛАВА 1 B-ДЕРЕВО 4](#_Toc58943658)

[1.1 Определение 4](#_Toc58943659)

[1.2 Структура 4](#_Toc58943660)

[1.2.1 Свойства: 5](#_Toc58943661)

[1.2.2 Структура узла: 5](#_Toc58943662)

[1.2.3 Структура класса B-дерева: 5](#_Toc58943663)

[1.3 Применение 5](#_Toc58943664)

[1.4 Структуры данных во внешней памяти 6](#_Toc58943665)

[1.5 Высота дерева 7](#_Toc58943666)

[1.6 Основные операции 8](#_Toc58943667)

[1.6.1 Добавление ключа 8](#_Toc58943668)

[1.6.2 Поиск ключа 9](#_Toc58943669)

[1.6.3 Удаление ключа 9](#_Toc58943670)

[1.6.4 Разбиение узла 10](#_Toc58943671)

[1.6.5 Перемещение ключа 10](#_Toc58943672)

[1.6.6 Слияние узлов 11](#_Toc58943673)

[1.6.7 Сложность 11](#_Toc58943674)

[ГЛАВА 2 ХРАНЕНИЕ В ПАМЯТИ 12](#_Toc58943675)

[ГЛАВА 3 ВЫБОР ПАРАМЕТРА ДЕРЕВА 14](#_Toc58943676)

[ГЛАВА 4 АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ 15](#_Toc58943677)

[ГЛАВА 5 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 17](#_Toc58943678)

[ЗАКЛЮЧЕНИЕ 18](#_Toc58943679)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc58943680)

# ВВЕДЕНИЕ

Деревья представляют собой структуры данных, в которых реализованы операции над динамическими множествами. Из таких операций хотелось бы выделить — поиск элемента, поиск минимального (максимального) элемента, вставка, удаление, переход к родителю, переход к ребенку. Таким образом, дерево может использоваться и как обыкновенный словарь, и как очередь с приоритетами.

Основная проблема большинства сбалансированных деревьев заключается в хранении всей информации в оперативной памяти. Данный способ не подходит в случае хранения огромной базы данных, которая может поместиться в оперативной памяти только частично. Такие деревья требуют большое количество обращений к стороннему носителю, на которые уходят значительные потери производительности. Эту проблему успешно решают B-деревья.

B-деревья были первоначально изобретены для хранения структур данных на диске, где локальность даже более важна, чем с памятью. Доступ к местоположению диска занимает около 5 мс = 5 000 000 НС. Поэтому, если хранить дерево на диске, нужно убедиться, что данное чтение диска является максимально эффективным. B-деревья имеют высокий коэффициент ветвления, намного больший, чем 2, что гарантирует, что для перехода к месту хранения данных потребуется мало операций чтения с диска. B-деревья также могут быть полезны для структур данных в памяти, потому что основная память почти так же медленна по отношению к процессору, как дисководы были к основной памяти, когда B-деревья были впервые введены.

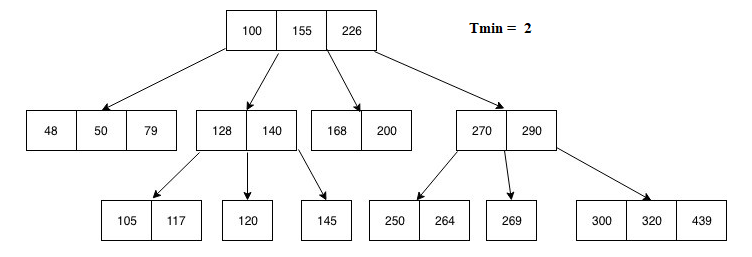
# ГЛАВА 1 B-ДЕРЕВО

## Определение

B-дерево — сильноветвящееся сбалансированное дерево поиска, позволяющее проводить поиск, добавление и удаление элементов за . B-дерево с *n* узлами имеет высоту . Количество детей узлов может быть от нескольких до тысяч (обычно степень ветвления B-дерева определяется характеристиками устройства (дисков), на котором производится работа с деревом). B-деревья также могут использоваться для реализации многих операций над динамическими множествами за время .

*Сбалансированность дерева* — свойство дерева, при котором длина любых двух путей от корня до листьев различается не более, чем на единицу.

*Сильная ветвистость дерева* — свойство дерева, при котором любой узел может ссылаться на большое число узлов-потомков.

B-дерево было изобретено Р. Бэйером и Е. МакКрейтом в 1970 году [3].

**Рис 1.1** Пример B-дерева

## Структура

B-дерево является идеально сбалансированным, то есть глубина всех его листьев одинакова. B-дерево имеет параметр, называемый минимальной степенью B-дерева, который должен быть не меньше 2. В дальнейшем будем обозначать параметр минимальной степени как ***Tmin***.

### Свойства:

* Каждый узел, кроме корня, содержит не менее ***Tmin – 1*** ключей
* Каждый узел, не являющийся листом, имеет по меньшей мере ***Tmin*** узлов-потомков
* Если дерево непустое, то корень содержит как минимум один ключ
* Каждый узел, кроме корня, содержит не более ***2Tmin - 1*** ключей
* Каждый узел, кроме корня и листьев, содержит не более ***2Tmin*** узлов-потомков
* Корень содержит от ***1*** до ***2Tmin - 1*** ключей при высоте большей 0
* Корень содержит от ***2*** до ***2Tmin - 1*** узлов-потомков при высоте большей 0
* Ключи в узлах упорядочены по неубыванию
* Каждый узел, кроме листьев, содержащий ключей, содержит ***n + 1*** узлов-потомков. При этом i-ый потомок содержит ключи из отрезка
* Все листья находятся на одинаковой глубине

### Структура узла:

**template <typename T>**

**struct** BNode

**bool** isLeaf **//** **является ли узел листом**

**size\_t** size **// количество ключей узла**

**T[]** keys **// массив узлов**

**unique\_ptr[]** childrens **// массив указателей на узлов-потомков**

### Структура класса B-дерева:

**template <typename T>**

**struct** BNode

**size\_t** tMin **//** **параметр минимальной степени**

**unique\_ptr<BNode<T>>** root **//** **указатель на корень**

## Применение

B-деревья используются для работы с дисками (в файловых системах) или иных энергонезависимых носителях информации с прямым доступом, а также в базах данных. Данное дерево оптимизирует количество операций чтения-записи с диском, при этом сохраняя асимптотику операций, в сравнении с другими деревьями.

## Структуры данных во внешней памяти

Кроме оперативной памяти, в компьютере используется внешний носитель, как правило, представляющий собой магнитные диски (или твердотельный накопитель). Хотя диски существенно дешевле оперативной памяти и имеют высокую емкость, они гораздо медленнее оперативной памяти из-за механического построения считывания.

Для того чтобы снизить время ожидания, связанное с механическим перемещением, при обращении к диску выполняется обращение одновременно сразу к нескольким элементам, хранящимся на диске. Информация разделяется на несколько страниц одинакового размера, которые хранятся последовательно друг за другом в пределах одного цилиндра (набора дорожек на дисках на одном расстоянии от центра), и каждая операция чтения или записи работает сразу с несколькими страницами. Для типичного диска размер страницы варьируется от ***2*** до ***16*** КБайт. После того, как головка установлена на нужную дорожку, а диск поворачивается так, что головка становится на начало интересующей нас страницы, чтение и запись становятся полностью электронными процессами, не зависящими от поворота диска, и диск может быстро читать или писать крупные объёмы данных.

В типичном приложении с B-деревом, объём хранимой информации так велик, что вся она просто не может храниться в основной памяти единовременно. Алгоритмы B-дерева копируют выбранные страницы с диска в основную память по мере надобности и записывает обратно на диск страницы, которые были изменены. Алгоритмы B-дерева хранят лишь определённое количество страниц в основной памяти в любой момент времени; таким образом, объём основной памяти не ограничивает размер B-деревьев, которые можно создавать.

Система в состоянии поддерживать в процессе работы в оперативной памяти только ограниченное количество страниц. Будем считать, что страницы, которые более не используются, удаляются из оперативной памяти системой; наши алгоритмы работы с В-деревьями не будут заниматься этим самостоятельно. Поскольку в большинстве систем время выполнения алгоритма, работающего с В-деревьями, зависит в первую очередь от количества выполняемых операций чтения/записи с диском, желательно минимизировать их количество и за один раз считывать и записывать как можно больше информации. Таким образом, размер узла В-дерева обычно соответствует дисковой странице. Количество потомков узла В-дерева, таким образом, ограничивается размером дисковой страницы. Для больших В-деревьев, хранящихся на диске, степень ветвления обычно находится между ***50*** и ***2000***, в зависимости от размера ключа относительно размера страницы. Большая степень ветвления резко снижает как высоту дерева, так и количество обращений к диску для поиска ключа. Например, если есть миллиард ключей, и ***Tmin=1001***, то поиск ключа займёт две дисковые операции.

## Высота дерева

Количество обращений к диску напрямую зависит от высоты дерева. Оценим высоту дерева в наихудшем и наилучшем случае:

**Теорема**

Пусть,

— высота дерева  
 — количество узлов в дереве   
 — параметр минимальной степени (каждый узел может иметь не более ключей)

Тогда

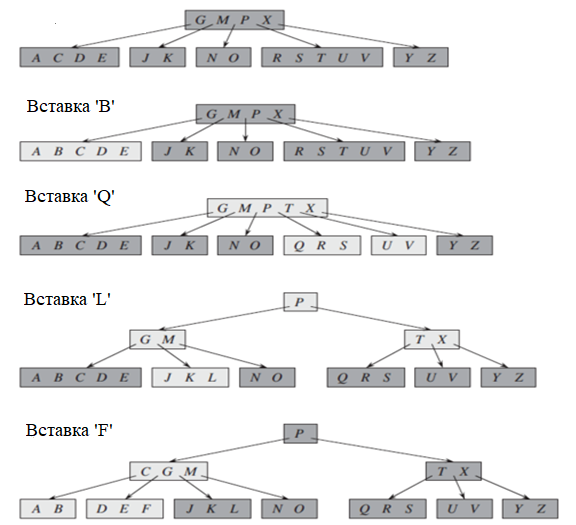
Корень B-дерева содержит как минимум 1 ключ, остальные узлы — Tmin – 1 ключей. На каждом уровне дерева содержится . Таким образом количество ключей удовлетворяет неравенству:

Из этого следует

Аналогично для .

## Основные операции

### Добавление ключа

Совершая проход от корня к листьям, пытаемся найти лист, в который добавим новый ключ. Если нашли какой-то незаполненный лист — добавляем ключ в него. Иначе разбиваем узел на два новых (первые Tmin – 1 ключей отойдут к первому узлу, последние Tmin – 1 отойдут ко второму узлу, средний элемент отойдет к родительскому узлу, который является разделительным узлом двух новых). Подробнее операцию разбиения рассмотрим дальше. Если родительский узел заполнен — повторим алгоритм, пока не дойдем до корня. Корень разбивается на два узла, и высота дерева увеличивается на 1.

**Рис 1.2** Пример добавления

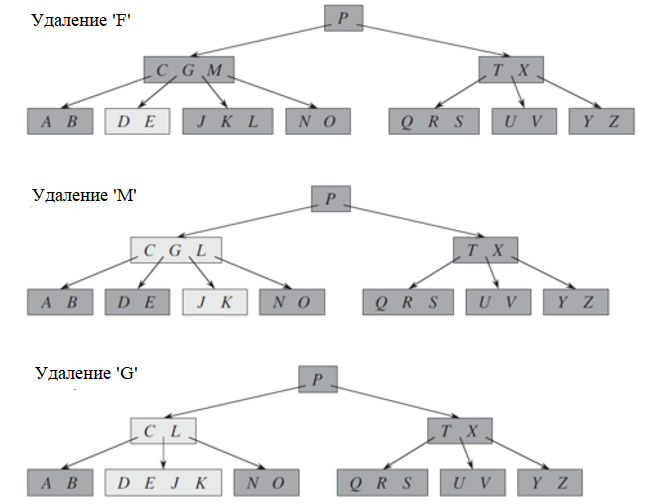
### Поиск ключа

Если ключ находится в текущем узле, возвращаем его, иначе определяем интервал поиска (бинарным поиском) и переходим к соответствующему потомку. Интервал поиска определяется промежутком двух последовательных ключей, между которыми находится желаемый ключ. Алгоритм продолжается пока не найдется ключ или не дошли до листа.

### Удаление ключа

Процесс удаления ключа является самым трудоемким в данной структуре. Для начала проделаем алгоритм поиска ключа, и, в случае нахождения, рассмотрим несколько вариантов:

* Текущий узел является листом — если количество ключей в узле не меньше чем Tmin — просто удалим ключ из узла. Легко заметить, что структура дерева, которая была обговорена вначале, не нарушится. Если количество ключей не более чем Tmin – 1, то рассмотрим соседние узлы с данным через его предка. Если оба этих узла содержат Tmin – 1 ключей, проведем слияние листьев (эта операция будет описана дальше), иначе произведем перемещение ключа из одного из соседних узлов (эта операция будет описана дальше). Структура дерева не нарушится, так как суммарное количество ключей будет равно 2(Tmin – 1).
* Текущий узел не является корнем и листом. Имеется внутренний узел xx и ключ, который нужно удалить, k. Если дочерний узел, предшествующий ключу k, содержит больше Tmin−1 ключа, то находим k1 – предшественника k в поддереве этого узла. Удаляем его. Заменяем k в исходном узле на k1. Проделываем аналогичную работу, если дочерний узел, следующий за ключом k, имеет больше Tmin−1 ключа. Если оба (следующий и предшествующий дочерние узлы) имеют по Tmin−1 ключу, то [объединяем](https://neerc.ifmo.ru/wiki/index.php?title=B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#.D0.A1.D0.BB.D0.B8.D1.8F.D0.BD.D0.B8.D0.B5) этих детей, [переносим](https://neerc.ifmo.ru/wiki/index.php?title=B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#.D0.9F.D0.B5.D1.80.D0.B5.D0.BC.D0.B5.D1.89.D0.B5.D0.BD.D0.B8.D0.B5_.D0.BA.D0.BB.D1.8E.D1.87.D0.B0) в них k, а далее удаляем k из нового узла.
* Если [сливаются](https://neerc.ifmo.ru/wiki/index.php?title=B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#.D0.A1.D0.BB.D0.B8.D1.8F.D0.BD.D0.B8.D0.B5) 2 последних потомка корня – то они становятся корнем, а предыдущий корень освобождается.

**Рис 1.3** Пример удаления

### Разбиение узла

Данная операция выполняется для разбиения заполненных узлов. Процесс заключается в разбиении узла на два новых. Первому узлу отходят первые Tmin−1 ключей, а второму последние Tmin−1, средняя вершина становится разделяющей вершиной. Стоит отметить, что высота дерева может быть увеличена только после операции разбиения в корне

### Перемещение ключа

Если выбранное для нисходящего прохода поддерево содержит минимальное количество ключей Tmin−1, и предшествующие и следующие узлы-братья имеют по меньшей мере Tmin ключей, то ключ перемещается в выбранный узел. При выборе перемещающегося ключа, у левого брата будем брать больший элемент, у правого меньший элемент.

### Слияние узлов

Для произвольных же слияний потребуется приведение сливаемых деревьев к одной степени и высоте. Для этого откладывается ключ из родительского узла x, который разделяет ключи на два сливаемых узла, в то время средний ключ перемещается в слитый узел. Ссылки на слитые дочерние узлы заменяются ссылкой на новый узел. Так как алгоритм гарантирует, что узел, в который будет совершаться спуск, содержит по меньшей мере Tmin ключей вместо требуемых условиями B-дерева Tmin−1 ключей, родительский узел x содержит достаточное количество ключей, чтобы выделить ключ для слияния. Это условие может быть нарушено, только в том случае, если два ребенка корня сливаются, так как поиск начинается с этого узла. По условиям B-дерева у корня должен быть как минимум один ключ, если дерево не пусто. При слиянии двух последних детей корня последний ключ перемещается во вновь возникшего единственного ребёнка, что приводит к пустому корневому узлу в не пустом дереве. В этом случае пустой узел корня удаляется и заменяется на единственного ребенка.

### Сложность

|  |  |  |
| --- | --- | --- |
| **Алгоритм** | **Обращений к диску** | **Процессорное время** |
| Добавление ключа |  |  |
| Удаление ключа |  |  |
| Поиск ключа |  |  |
| Разбиение узла  Слияние узлов  Перемещение ключа |  |  |

**Таблица 1.1** Сложность операций

# ГЛАВА 2 ХРАНЕНИЕ В ПАМЯТИ

Вторичное хранилище обычно относится к фиксированным дискам, найденным в современных компьютерах. Эти устройства содержат несколько пластин из быстро вращающегося магнитно-чувствительного материала. Данные хранятся в виде изменений магнитных свойств на различных частях пластин. Данные разделены на дорожки, концентрические круги на пластинах. Каждая дорожка далее делится на сектора, которые образуют единицу транзакции между диском и процессором. Типичный размер сектора составляет 512 байт. Данные считываются и записываются головкой, которая идет по пластинам, получая доступ к различным секторам по мере их запроса. Диск вращается с постоянной скоростью (7200 об/мин характерно для систем среднего класса 1998 года).

Время, необходимое для доступа к данным во вторичном хранилище, зависит от трех переменных:

* Время, необходимое головке для перемещения на дорожку, где находится требуемый сектор. Обычно около 10 миллисекунд.
* Время, необходимое правому сектору, чтобы крутануться под головкой. Для привода с частотой вращения 7200 об/мин это составляет 4,1 миллисекунды.
* Время, необходимое для чтения или записи данных. В зависимости от плотности данных это время ничтожно мало по сравнению с двумя другими.

Таким образом, произвольный 512-байтовый сектор может быть доступен (прочитан или записан) примерно за 15 миллисекунд. Последующее считывание на соседнюю область диска будет происходить намного быстрее, так как головка уже находится точно в нужном месте. Данные могут быть организованы в "блоки", которые являются этими смежными многосекторными агрегатами.

В сравнении со временем последовательного доступа к оперативной памяти, время доступа к диску очень велико (доступ к оперативной памяти примерно 5 микросекунд). Это в 3000 раз быстрее; мы можем сделать по крайней мере 3000 обращений к памяти за время, необходимое для выполнения одного доступа к диску, и, вероятно, больше, поскольку алгоритм, выполняющий доступ к памяти, обычно следует принципу локальности.

Поэтому нам лучше сделать так, чтобы каждый доступ к диску считался как можно больше. Это то, что делают B-деревья.

Размер блока и объем данных можно настроить с помощью экспериментов или анализа, выходящих за рамки данной курсовой работы. На практике иногда во внутренних узлах B-дерева хранятся только "указатели" на другие дисковые блоки, причем листовые узлы содержат реальные данные; это позволяет хранить гораздо больше ключей и/или иметь меньшие (а значит, и более быстрые) блоки.

Обычно алгоритмы сортировки и поиска характеризуются количеством операций сравнения, которые должны быть выполнены с использованием нотации порядка. Двоичный поиск отсортированной таблицы с N записями, например, может быть выполнен примерно в log N сравнениях. Если таблица содержит 1 000 000 записей, то конкретная запись может быть найдена не более чем с 20 сравнениями.

Время не будет таким уж плохим, потому что отдельные записи сгруппированы вместе в дисковом блоке. Дисковый блок может составлять 16 килобайт. Если каждая запись составляет 160 байт, то в каждом блоке может храниться 100 записей. Время чтения диска выше было фактически для целого блока. Как только дисковая головка находится в нужном положении, один или несколько дисковых блоков могут быть считаны с небольшой задержкой. При 100 записях на блок последние 6 или около того сравнений не нуждаются в чтении диска—все сравнения находятся в пределах последнего прочитанного блока диска.

Чтобы ускорить поиск, необходимо ускорить первые 13-14 сравнений (каждое из которых требовало доступа к диску).

# ГЛАВА 3 ВЫБОР ПАРАМЕТРА ДЕРЕВА

Производительность B-дерева напрямую зависит от параметра ***Tmin***. Таким образом следует позаботиться о выборе параметра, чтобы обеспечить производительность как можно более хорошей.

Чтобы получить очень грубое приближение к производительности схемы мы делаем следующие предположения:

* Время, затраченное на каждую страницу, которая записана или извлечена, может быть выражено в виде:

фиксированное время, затраченное на страницу, например, среднее время

поиска диска, а также фиксированные затраты процессора и т. д.

время передачи на одну страницу записи.

константа для логарифмической части времени, например,

для двоичного поиска.

коэффициент средней заполняемости страницы,

Мы предполагаем, что изменение страницы не требует перемещения ключей внутри страницы, но что необходимые подкоманды канала генерируются для записи страницы путем объединения нескольких фрагментов информации в главном хранилище. Это является причиной нашего предположения, что извлечение и написание страницы занимает одно и то же время.

Среднее число страниц, извлеченных и записанных за одну транзакцию в среде смешанных извлечений, вставок и удалений, примерно пропорционально h. Затем общее время t, затраченное на одну транзакцию, может быть аппроксимировано:

*I – размер индекса*

Отсюда получаем такую оценку:

Теперь легко получить минимум Ta, если ***Tmin*** выбрано таким образом, что

# ГЛАВА 4 АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ

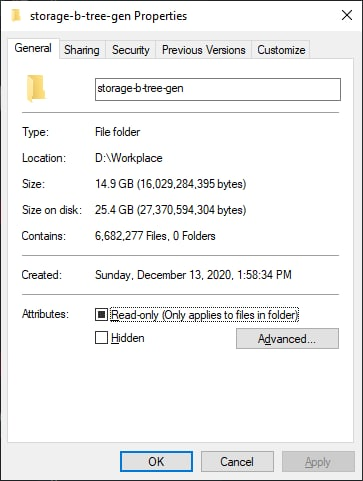
Для анализа производительности я выбрал способ хранения узлов дерева в отдельных файлах.

Вся информация дерева хранится в какой-то папке, а узел имеет вид:

* Storage/node-k{index}.bin (в этом файле хранятся захешированные значения узла и информация о потомках узла)
* Storage/node-v{index}.bin (в этом файле хранятся сами значения)

Сгенерируем таблицу NxN, N = 35000

В качестве информации будем хранить ячейки таблицы, к которым мы будем иметь доступ на изменение/удаление/добавление информации.

В качестве хеша ячейки (i,j) будем использовать size \* i + j для большей локальности.

**Рис 4.1** Папка хранилища

Все тестирование проводилось на ноутбуке c процессором Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 ядра, 8 логических процессоров и 8 ГБ оперативной памяти. В качестве внешнего хранилища использовался встроенный Seagate Hybrid Drive ST2000DX001 2TB MLC/8GB 64MB Cache SATA 6.0Gb/s NCQ 3.5" Desktop SSHD.

Тестирование B-дерева проводилось с параметром 160, так как при этом параметре вершина помещается на одну буферную страницу в памяти. В качестве ключей и значений использовались целые числа типа int, файлы вершин хранились в бинарном формате. Результаты приведены в таблице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1000  запросов вставки | 1000  запросов удаления | 1000 запросов поиска | 1000  запросов обновления |
| 108 записей | 8.83 c | 8.25 c | 4.01 c | 5.40 c |
| 109 записей | 10.56 c | 10.53 c | 5.76 c | 6.79 c |

**Таблица 4.1** Производительность B-дерева

Из результатов видно, что операции поиска выполняются немного быстрее всех остальных. Это связано с тем, что при остальных операциях необходимо совершить перезапись вершин. Также следует отметить, что при многократном обращении к одним и тем же вершинам, то есть при использовании близких ключей, производительность сильно возрастает, т. к. файлы кэшируются операционной системой. При увеличении количества записей в 10 раз, затраченное время увеличивается несущественно, что и является главным преимуществом B-дерева.

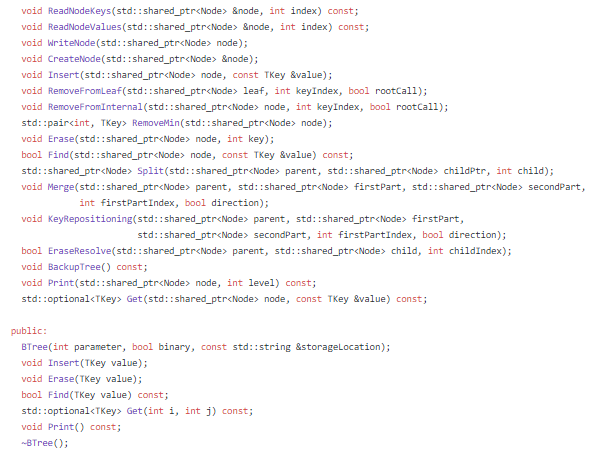
Большое время модификации структуры данных обусловлено тем, что был выбран способ хранения "каждый узел в отдельном файле", а взаимодействие с ними осуществляется через файловые системы операционной системы.

# ГЛАВА 5 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Исходный код: [**https://github.com/MelehAlexei/b-trees**](https://github.com/MelehAlexei/b-trees)

Реализованное дерево имеет операции:

* Вставка ключа
* Удаление ключа
* Поиск ключа
* Обращение к значению ключа

В качестве формата хранения данных был выбран формат бинарного файла, поскольку им удобнее и быстрее записывать/читать структуры любого содержания. В качестве полезной информации ячеек таблицы была выбрана структура из 3 типов int. Класс дерева содержит шаблонный параметр hash, который помогает поддерживать локальность ячеек в дереве.

**Рис 5.1** Список методов класса

# ЗАКЛЮЧЕНИЕ

В ходе курсового проекта был изучен и реализован алгоритм B-дерева, который использует взаимодействие с диском через файловые системы операционной системы. Были изучены свойства, которые помогают выбрать оптимальный параметр ветвления для B-дерева. Для анализа было сгенерировано два B-дерева, одно из которых содержало по меньшей мере полезной информации, а другое . Полезная информация — структура данных, хранящаяся в ячейках таблицы.

На данный момент, обработка больших данных является задачей первостепенной важности. Не всегда имеется возможность разместить эти данные полностью в оперативной памяти. В таких случаях, применение алгоритмов во внешней памяти является необходимым. Например, в современных системах управления базами данных применяется улучшенная реализация B-дерева, называемая B+-дерево. Несмотря на то, что алгоритмы во внешней памяти позволяют обрабатывать большие объемы данных, следует помнить, что они часто медленнее своих классических аналогов и, если данные возможно разместить в оперативной памяти, то не следует использовать алгоритмы во внешней памяти.

В курсовой работе планируется реализация своей файловой системы для ускорения B-дерева, а также реализации других деревьев и своей базы данных.

# СПИСОК ЛИТЕРАТУРЫ

1. Д. Кнут «Искусство программирования. Сортировка и поиск», часть 6.2.4
2. Т. Кормен «Алгоритмы: построение и анализ» второе издание, глава 18
3. Е. МакКрейт [Электронный ресурс]

https://infolab.usc.edu/csci585/Spring2010/den\_ar/indexing.pdf

1. Алгоритм B-дерева [Электронный ресурс] https://habr.com/ru/post/114154/
2. B-деревья и TRIE-деревья [Электронный ресурс]

https://www.ibm.com/developerworks/ru/library/l-data\_structures\_10/