**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра дискретной математики и алгоритмики

**Анализ эффективности B-деревьев при реализации операций в базах данных**

Курсовая работа

Мелеха Алексея Александровича

студента 3 курса,

специальности “Информатика”

Научный руководитель:

Лукьянов Иван Денисович

Ассистент кафедры

ДМА

Минск, 2021

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc72322151)

[ГЛАВА 1 B-ДЕРЕВО 4](#_Toc72322152)

[1.1 Определение 4](#_Toc72322153)

[1.2 Структура 4](#_Toc72322154)

[1.2.1 Свойства: 5](#_Toc72322155)

[1.2.2 Структура узла: 5](#_Toc72322156)

[1.2.3 Структура класса B-дерева: 5](#_Toc72322157)

[1.3 Применение 5](#_Toc72322158)

[1.4 Структуры данных во внешней памяти 6](#_Toc72322159)

[1.5 Высота дерева 7](#_Toc72322160)

[1.6 Основные операции 8](#_Toc72322161)

[1.6.1 Добавление ключа 8](#_Toc72322162)

[1.6.2 Поиск ключа 9](#_Toc72322163)

[1.6.3 Удаление ключа 9](#_Toc72322164)

[1.6.4 Разбиение узла 10](#_Toc72322165)

[1.6.5 Перемещение ключа 10](#_Toc72322166)

[1.6.6 Слияние узлов 11](#_Toc72322167)

[1.6.7 Сложность 11](#_Toc72322168)

[ГЛАВА 2 B+-ДЕРЕВО 12](#_Toc72322169)

[ГЛАВА 3 ХРАНЕНИЕ В ПАМЯТИ 16](#_Toc72322170)

[ГЛАВА 4 ВЫБОР ПАРАМЕТРА ДЕРЕВА 18](#_Toc72322171)

[ГЛАВА 5 АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ 19](#_Toc72322172)

[ГЛАВА 6 СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА B И B+-ДЕРЕВЬЕВ 21](#_Toc72322173)

[ГЛАВА 7 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 28](#_Toc72322174)

[ЗАКЛЮЧЕНИЕ 34](#_Toc72322175)

[СПИСОК ЛИТЕРАТУРЫ 35](#_Toc72322176)

# ВВЕДЕНИЕ

Деревья представляют собой структуры данных, в которых реализованы операции над динамическими множествами. Основными операциями являются добавление, удаление, поиск. 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.

В данной курсовой работе будет произведена сравнительная характеристика алгоритмов B-дерева и B+-дерева. Выявлены недостатки и преимущества каждого из этих алгоритмов при работе с БД.

# 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.

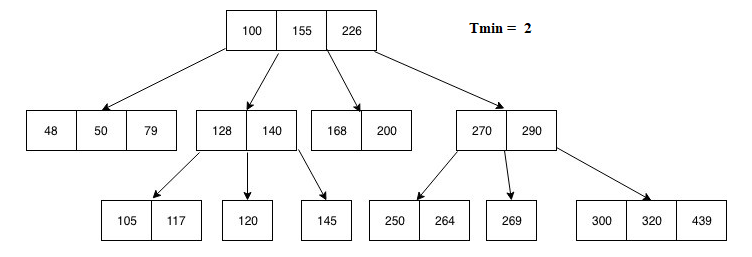
B-дерево было изобретено Р. Бэйером и Е. МакКрейтом в 1970 году [3].

Рисунок 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 ***T***.

### Свойства:

* Каждый узел, кроме корня, содержит не менее ***T – 1*** ключей
* Каждый узел, не являющийся листом, имеет по меньшей мере ***T*** узлов-потомков
* Если дерево непустое, то корень содержит как минимум один ключ
* Каждый узел, кроме корня, содержит не более ***2T - 1*** ключей
* Каждый узел, кроме корня и листьев, содержит не более ***2T*** узлов-потомков
* Корень содержит от ***1*** до ***2T - 1*** ключей при высоте большей 0
* Корень содержит от ***2*** до ***2T - 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** t **//** **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 ***T=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)

**Доказательство:**

Тогда

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

Из этого следует

Аналогично для .

## Основные операции

### Добавление ключа

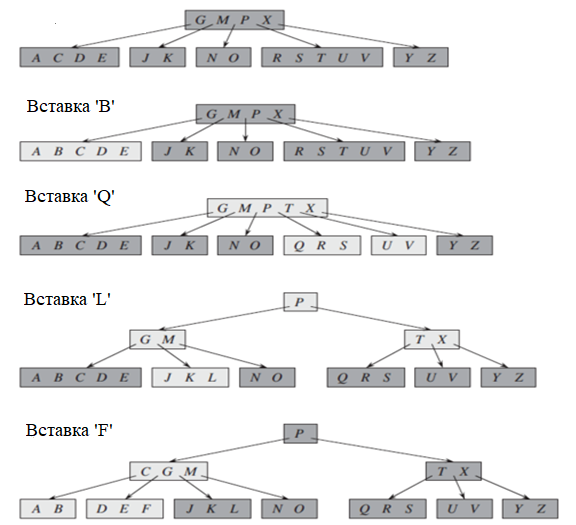
Совершая проход от корня к листьям, пытаемся найти лист, в который добавим новый ключ. Если нашли какой-то незаполненный лист — добавляем ключ в него. Иначе разбиваем узел на два новых (первые T – 1 ключей отойдут к первому узлу, последние T – 1 отойдут ко второму узлу, средний элемент отойдет к родительскому 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.

Рисунок 1.2. Пример добавления узла из B-дерева

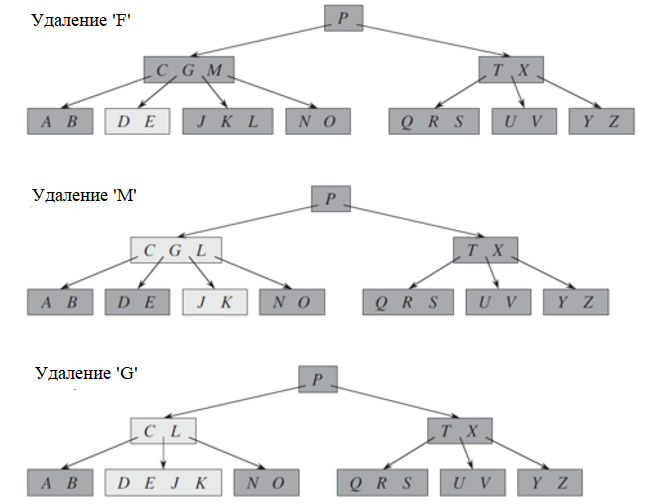
### Поиск ключа

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 T — просто удалим ключ из узла. Легко заметить, что структура дерева, которая была обговорена вначале, не нарушится. Если количество ключей не более чем T – 1, то рассмотрим соседние узлы с данным через его предка. Если оба этих узла содержат T – 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(T – 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 T−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 T−1 ключа. Если оба (следующий и предшествующий дочерние узлы) имеют по T−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.

Рисунок 1.3. Пример удаления узла из B-дерева

### Разбиение узла

Данная операция выполняется для разбиения заполненных узлов. Процесс заключается в разбиении узла на два новых. Первому узлу отходят первые T−1 ключей, а второму последние T−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 T−1, и предшествующие и следующие узлы-братья имеют по меньшей мере T 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 T ключей вместо требуемых условиями B-дерева T−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 B+-ДЕРЕВО

* 1. **Опреление**

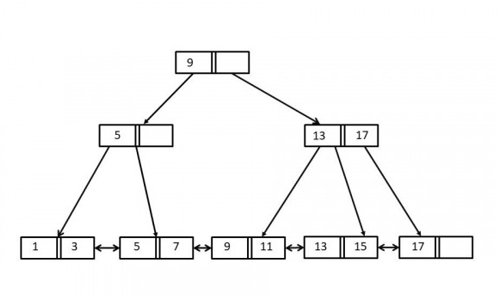
B+-дерево — структура данных на основе B-дерева, сбалансированное ***n***-арноедерево поиска с переменным, но зачастую большим количеством потомков в узле. B+-деревья имеют очень высокий коэффициент ветвления (число указателей из родительского узла на дочерние, обычно порядка 100 или более), что снижает количество операций ввода-вывода, требующих поиска элемента в дереве.

Рисунок 2.1. Пример B+-дерева

* 1. **Отличия от B-дерева**

B 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 степерь 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.

* 1. **Структура**

Свойства B+-дерева аналогичны свойствам B-дерева (за исключением отличий описанных в прошлом пункте).

1. 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

1. Структура дерева

**struct** BPlusTree

**int** t // минимальная степень дерева

**Node** root // указатель на корень дерева

* 1. **Оценка высоты дерева**

**Теорема:** Если , то для B+-дерева с узлами и минимальной степенью высота .

***Доказательство:***

Так как , то корень B+-дерева содержит хотя бы один ключ, а все остальные узлы — хотя бы ключей. имеет хотя бы 2 узла на высоте 1, не менее узлов на глубине 2, и так далее. То есть на глубине , оно имеет хотя бы узлов. Так как сами ключи хранятся только в листах, а во внутренних вершинах лишь их копии, то для ключей .

Как можно заметить, высота B+-дерева не более чем на 1 отличается от высоты B-дерева, то есть хранение информации только в листах почти не ухудшает эффективность дерева.

* 1. **Основные операции**

B+-деревья являются сбалансированными, поэтому время выполнения стандартных операций в них пропорционально высоте, то есть . Однако стоит заметить, что так как степень дерева зачастую выбирается большой, константа при выполнении операций тоже большая. Это связано с большим количеством ключей в узлах, которые необходимо сравнить. Но из-за небольшой высоты дерева это не сильно сказывается на скорости работы.

1. Разбиение узла

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

Если и родительский узел заполнен — поступаем аналогично, но не копируем, а просто перемещаем оставшийся 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.

1. 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 добавление будет возрващать был ли добавлен ключ.

1. 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  ключей, где  — 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 22 одинаковых ключей, то удаление будет возвращать был ли удален ключ.

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

Таблица 2.1.Сложность операций

# ГЛАВА 3 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 4 ВЫБОР ПАРАМЕТРА ДЕРЕВА

Производительность B-дерева напрямую зависит от параметра ***T***. 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 ***T*** выбрано таким образом, что

# ГЛАВА 5 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.

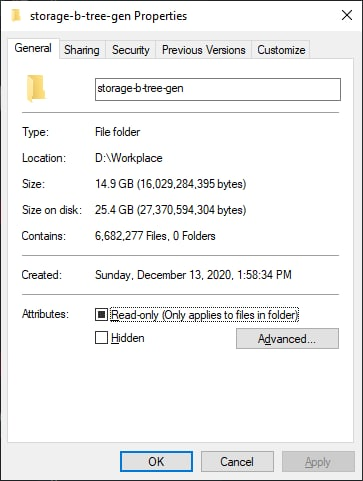
В качестве хеша ячейки (i,j) будем использовать size \* i + j для большей локальности.

Рисунок 5.1. 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 5.1. 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 6 СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА B И B+-ДЕРЕВЬЕВ

Поскольку дерево B+ является продолжением дерева B, основные операции, которые мы обсуждали в разделе дерево B, все еще выполняются.

При вставке, а также удалении мы должны сохранять основные свойства деревьев B+ нетронутыми. Однако операция удаления в дереве B+ сравнительно проще, так как данные хранятся только в конечных узлах, и они всегда будут удалены из конечных узлов.

1. **Преимущества Деревьев B+**

* Мы можем извлекать записи в равном количестве обращений к диску.
* По сравнению с деревом B высота дерева B+ меньше и остается сбалансированной.
* Мы используем ключи для индексации.
* К данным в дереве B+ можно обращаться последовательно или напрямую, поскольку конечные узлы расположены в связанном списке.
* Поиск выполняется быстрее, так как данные хранятся только в конечных узлах и в виде связанного списка.

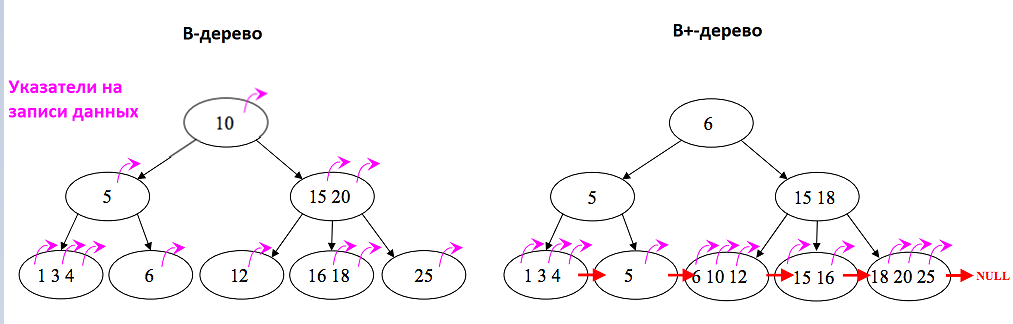


Рисунок 6.1.Графическое представление деревьев

|  |  |
| --- | --- |
| **B-дерево** | **B+-дерево** |
| Данные хранятся как в конечных узлах, так и во внутренних узлах. | Данные хранятся только в конечных узлах. |
| Поиск немного медленнее, так как данные хранятся как во внутренних, так и в конечных узлах. | Поиск выполняется быстрее, так как данные хранятся только в конечных узлах.  Могут присутствовать избыточные ключи поиска. |
| Операция удаления сложна. | Операция удаления проста, так как данные могут быть непосредственно удалены с конечных узлов. |
| Конечные узлы не могут быть связаны друг с другом. | Конечные узлы связаны друг с другом, образуя связанный список. |

Таблица 6.1.Различия деревьев

1. **B+-индексирование деревьев**

Все записи таблицы в СУБД хранятся на диске, чтобы гарантировать их постоянство в случае сбоев программного и/или аппаратного обеспечения. Записи в таблице организованы в файлы, управляемые СУБД, а не ОС. Всякий раз, когда запрос отправляется в СУБД, запрос обрабатывается, и записи в таблицах реляционной базы данных, удовлетворяющие запросу, возвращаются СУБД. Способ обработки запроса зависит от конкретной структуры хранения записей в файле. Если записи организованы не в определенном порядке, то структура хранения называется кучей. В то время как когда записи упорядочены в соответствии со значениями некоторых его атрибутов, структура хранения файла считается отсортированной.

Чтобы получить записи в файле, организованном в кучу, все, что может сделать СУБД, - это последовательное сканирование страниц диска, на которых хранятся записи. Как правило, страницы, составляющие файл, являются смежными на диске.

С другой стороны, если записи находятся в отсортированном файле, то двоичный поиск на страницах диска файла может использоваться для извлечения записей, если критерии запроса используют те же атрибуты, что и те, которые используются для поддержания порядка сортировки записей в файле. Например, если файл, в котором хранятся записи для таблицы "Студенты", отсортирован по фамилии и имени (в таком порядке), то двоичный поиск можно использовать для обработки запросов, которые ищут студентов только по фамилии, или запросов, которые ищут студентов по их полному имени. Запрос, который ищет студентов по дате их рождения, должен быть обработан с помощью последовательного сканирования отсортированного файла; аналогично, если запрос ищет студентов по их имени.

Индекс таблицы-это структура данных и связанные с ней алгоритмы, которые обеспечивают механизм, с помощью которого записи таблицы можно искать более эффективно, чем последовательное сканирование или двоичный поиск. Наиболее распространенными индексными структурами, используемыми СУБД, являются B+-деревья-другая индексная структура-хэш-таблица. Остальная часть этого документа описывает структуру индекса B+-дерева.

Время извлечения из внешней памяти (например, диска) в тысячи раз больше, чем в высокоскоростной памяти. Цель внешнего поиска состоит в том, чтобы свести к минимуму количество обращений к диску, поскольку доступ к диску занимает больше времени, чем вычисления. Это отличается от цели минимизации количества вычислений (т.е. сравнений) при поиске данных, которые находятся в более быстрой основной памяти. По этой причине типичный доступ к записи из внешней памяти считывает сразу всю страницу или блок данных. Извлекая страницу диска, содержащую много записей, мы используем локальность ссылки-если извлекается запись и мы организовали записи в зависимости от того, как к ним обращаются, то, скорее всего, записи, находящиеся в извлеченном блоке, будут извлечены следующими.

B+-дерево-это одно из семейств деревьев многоходового поиска (другие: B-дерево, 2-3-4 дерева, B\*-деревья). Эти деревья были впервые предложены Байером и Маккрайтом в 1972 году и всего за несколько лет заменили почти все методы доступа к большим файлам, кроме хэширования. Эти многоходовые сбалансированные деревья поиска теперь являются стандартной организацией файлов для приложений, требующих вставки, удаления и поиска по диапазону ключей. Они имеют следующие преимущества:

B-деревья всегда сбалансированы по высоте, все листовые узлы находятся на одном уровне

Операции обновления и поиска затрагивают только несколько страниц диска, поэтому производительность хорошая.

B-деревья хранят связанные записи на одной и той же странице диска, что позволяет использовать локальность ссылок.

B-деревья гарантируют, что каждый узел в дереве будет заполнен, по крайней мере, до определенного минимального процента. Это повышает эффективность использования пространства при одновременном уменьшении типичного количества извлечений диска, необходимых во время операции поиска или обновления по многим тысячам записей.

Индекс таблицы-это структура данных и связанные с ней алгоритмы, которые обеспечивают механизм, с помощью которого записи таблицы можно искать более эффективно, чем последовательное сканирование или двоичный поиск. Наиболее распространенными индексными структурами, используемыми СУБД, являются B+-деревья-другая индексная структура-хэш-таблица. Остальная часть этого документа описывает структуру индекса B+-дерева.

Простейшее B+-дерево возникает, когда m = 3: каждый внутренний узел имеет либо 2, либо 3 дочерних элемента. На практике размер узла в дереве B+выбирается для заполнения страницы диска. Реализация узла B+-дерева обычно допускает от 100 до 200 дочерних элементов, фактическое число зависит от требований к памяти ключей и адресов памяти. Было сообщено (Рамакришнаном и Герке), что типичный коэффициент заполнения узлов составляет 67%, что дает средний коэффициент ветвления или разветвления 133 (т.е. Каждый внутренний узел указывает на 133 дочерних узла). В B+-дереве высотой 4, которое имеет средний разветвитель 133, можно использовать для индексации таблицы с более чем 300 миллионами записей (если быть точным, это 1334 = 312 900 700). B+-дерево высотой 3 может индексировать таблицу из 1333 = 2 352 637 записей. Получив доступ всего к пяти страницам диска (количество пройденных уровней на один больше, чем высота дерева), СУБД может найти одну из более чем 300 миллионов записей. Кроме того, обычно верхние уровни дерева B+кэшируются в основной памяти. Требования к памяти не так высоки. При типичном размере страницы 8 Кб первые два уровня B+-дерева (до 134 страниц на диске) могут быть кэшированы примерно за 1 Мбайт. Теперь СУБД необходимо получить доступ только к трем страницам диска, соответствующим трем нижним уровням индекса B+-дерева, чтобы найти одну из более чем 300 миллионов записей! Это ясно иллюстрирует, почему B+-деревья так успешно применяются для индексирования данных в базах данных.

1. **Определение размера B+-дерева**

Рассмотрим отношение из 2 000 000 кортежей (это 2 миллиона кортежей), хранящихся в файловой структуре кучи. Насколько большим (по количеству страниц на диске) будет индекс B+-дерева в этом отношении? Будем считать, что мы строим B+-дерево, которое заполнено как можно больше. Чтобы определить это, нам нужно знать следующее:

Размер одной страницы диска. В то время как страницы обычно составляют 4 Кб или 8 Кб, чтобы упростить наши вычисления, мы будем использовать размер страницы 1000 байт

Размер значения ключа поиска. Предположим, что мы индексируем отношение с помощью атрибута, представляющего собой 40-байтовую строку и являющегося ключом-кандидатом отношения.

Размер физических адресов страниц. Это необходимо для наших вычислений, так как каждый узел в нашем B+-дереве будет хранить значения ключей поиска, а также ссылки на страницы диска, т. е. Адрес страницы диска, на которую ссылаются. Опять же, для удобства вычислений мы используем 10-байтовый адрес страницы.

Теперь, когда у нас есть информация выше, мы можем приступить к определению размера нашего индекса B+-дерева. Мы определим это, работая снизу вверх по дереву:

Поскольку размер страницы составляет 1000 байт, она может хранить только ограниченное количество записей индекса. Кроме того, мы строим дерево B+, которое заполняется как можно больше, поэтому мы будем хранить столько записей индекса, сколько может поместиться на странице диска. Запись индекса представляет собой пару значений: [значение ключа поиска, ссылка на кортеж], и в нашем случае размер(значение ключа поиска)=40 и размер(ссылка на кортеж)=10. Таким образом, чтобы определить количество записей индекса, которые могут поместиться на листовой странице, мы можем использовать этот расчет:

Количество записей индекса на листовую страницу =

Количество записей индекса на листовой странице = = 20

Листовые страницы B+-дерева связаны друг с другом, эти ссылки хранятся в виде адресов соседних дисковых страниц. Опять же, для простоты, дисковое пространство, используемое для хранения этих ссылок, не учитывается в расчетах, показанных здесь.

Теперь мы должны определить необходимое количество листовых страниц. Поскольку таблица, которую мы индексируем, хранится в виде кучи, кортежи не упорядочены в файле данных. Таким образом, наш индекс B+-дерева должен содержать один ввод индекса на кортеж в таблице; это называется плотным индексом. Таким образом, на листовых страницах будет храниться 2 миллиона записей индекса в совокупности. Необходимое количество листовых страниц задается:

Количество листовых страниц =

Количество листовых страниц = = 100,000

Затем мы должны определить коэффициент ветвления, m, нашего B+-дерева. Это можно сделать, повторив шаг 1 выше с некоторыми небольшими изменениями. В B+-дереве порядка m каждый внутренний узел должен иметь возможность хранить до m ветвей (т.е. указателей на узлы на уровне ниже) и, таким образом, до m-1 значений ключей поиска (обычно называемых значениями разделителей, когда они появляются во внутренних узлах).

Page\_size >= (m-1) \* размер(значение ключа поиска) + m \* размер(указатель)

В нашем случае мы имеем: 1000 >= (m-1) \* 40 + m \* 10

Решая для m, которое должно быть целым числом, мы получаем: m = = 20

Таким образом, каждый внутренний узел будет хранить 20 указателей на узлы на уровне ниже и соответствующие 19 значений ключа поиска. Обратите внимание на небольшое, но важное различие между внутренними узлами и конечными узлами.

Теперь, когда у нас есть m, мы можем определить количество страниц на каждом уровне в дереве B+. Поскольку на каждый лист в нашем дереве нужно ссылаться с уровня выше, в совокупности нам понадобится 100 000 указателей (т.е. ветвей), по одному на листовую страницу. Поэтому количество внутренних страниц на уровне над листами равно:

.

Количество страниц на уровне выше листьев =

Количество страниц на уровне выше листьев = = 5000

Нам нужно продолжать выполнять эти вычисления до тех пор, пока мы не достигнем точки, когда у нас будет только один внутренний узел, который будет корнем нашего дерева

Наконец-то мы можем ответить на вопрос о размере нашего B+ - дерева:

Общее количество страниц = 100,000 + 5,000 + 250 + 13 + 1 = 105,264 страницы.

У нас есть 5-уровневое B+-дерево, индексирующее 2 миллиона кортежей, поэтому, обратившись к 5 дисковым страницам B+-дерева, за которыми следует 1 страница файла, мы можем найти любой из этих кортежей.

Обратите внимание, что системы баз данных используют кэширование, и обычно два верхних уровня дерева B+находятся в кэше, что позволяет сохранить два доступа к диску в этом примере.

Наконец, стоит отметить, что в этом примере для упрощения вычислений использовался небольшой размер страницы в 1000 байт. Общие размеры страниц составляют 4 Кб, 8 Кб или даже 16 Кб. Большие размеры страниц приводят к большим факторам ветвления и, следовательно, к более мелким деревьям. Конечно, существует компромисс между размером страницы и использованием страницы. Занимать только половину страницы размером 4 Кб не так важно, как занимать только половину страницы размером 16 Кб. B\*-деревья могут решить эту проблему, см. Следующий раздел.

Наконец, если бы нас попросили найти размер дерева B+, страницы которого заполнены до минимальной емкости, мы могли бы следовать описанному выше процессу с одним дополнительным соображением. Страницы B+-дерева должны иметь минимум потолочных(m/2) дочерних узлов (т.е. половина страницы должна быть заполнена). Таким образом, приведенные выше вычисления должны были бы принять это во внимание, чтобы найти размер такого дерева.

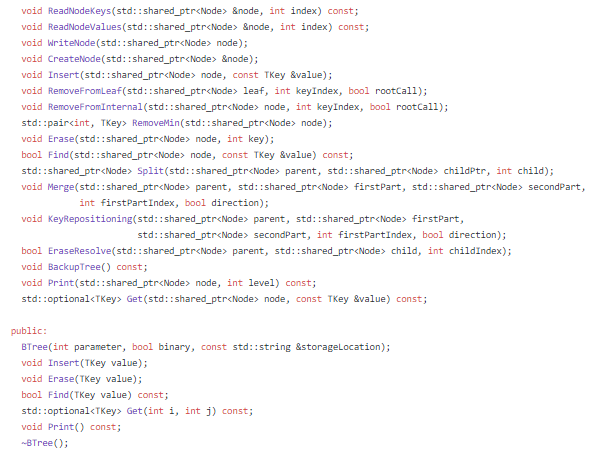
# ГЛАВА 7 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

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

Реализованные деревья имеют 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 была выбрана база данных IMDB, которая содержит в кортеже информацию о теге фильма, его среднюю оценку и количество голосовавших людей. В качестве индексации была выбрана колонка со средней оценкой фильма. Для тестирования был выбран размер блока в 100B, где каждая запись фильма примерно 20B. В каждом блоке может храниться несколько записей. Память дерева B+ динамически выделяется при создании. Каждый узел дерева B+ имеет размер одного блока. Конечные узлы связаны в двусвязном списке. Конечные узлы поддерживают указатели на фактический адрес данных в пуле памяти.

Рисунок 7.1. Список методов класса

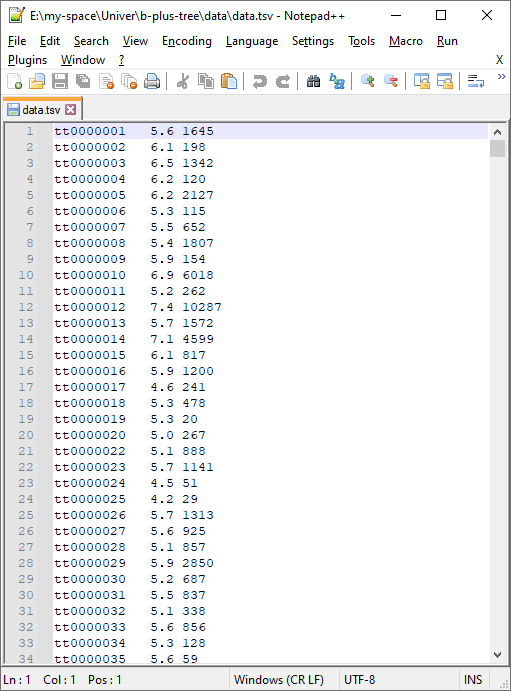


Рисунок 7.2. Список кортежей

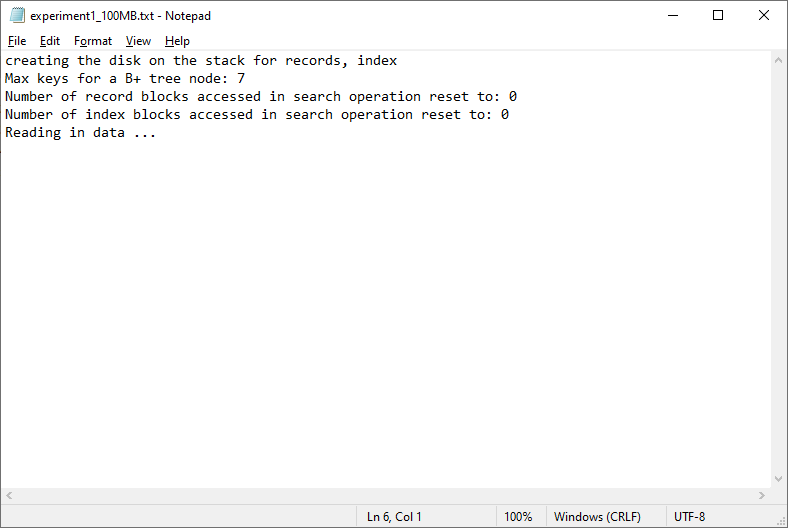


Рисунок 7.3. Результат эксперимента №1



Рисунок 7.4. Результат эксперимента №2

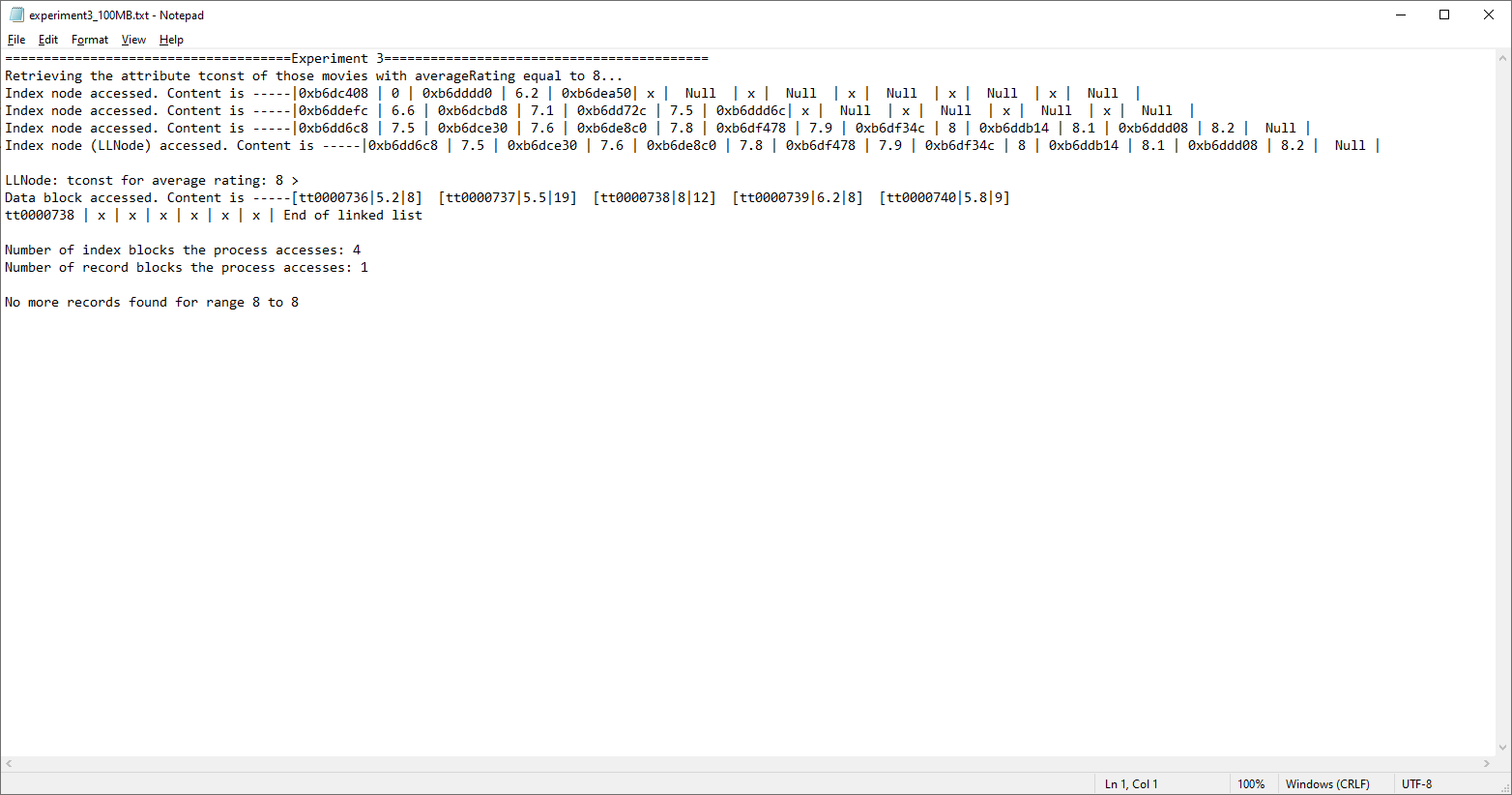


Рисунок 7.5. Результат эксперимента №3

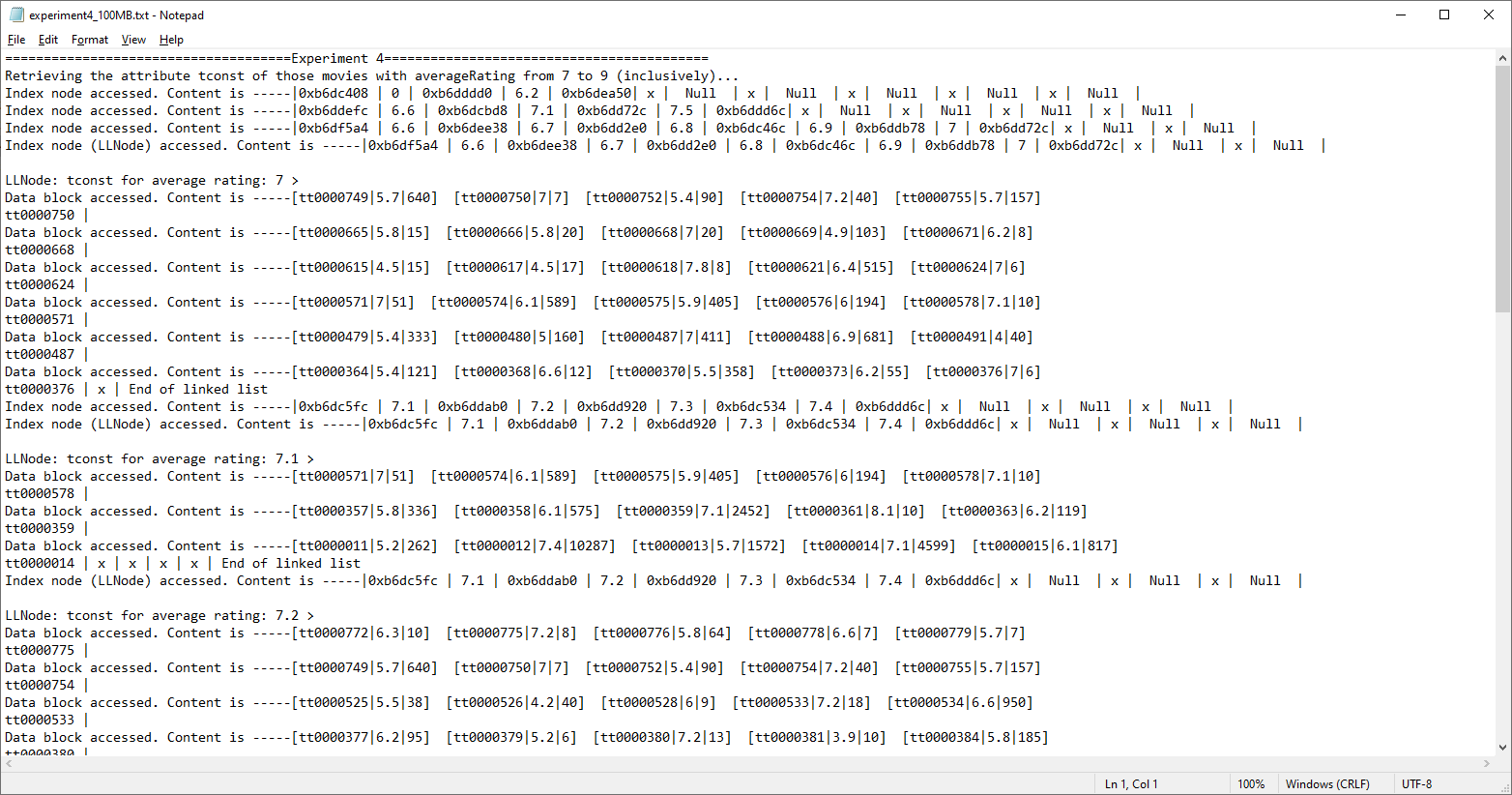


Рисунок 7.6. Результат эксперимента №4

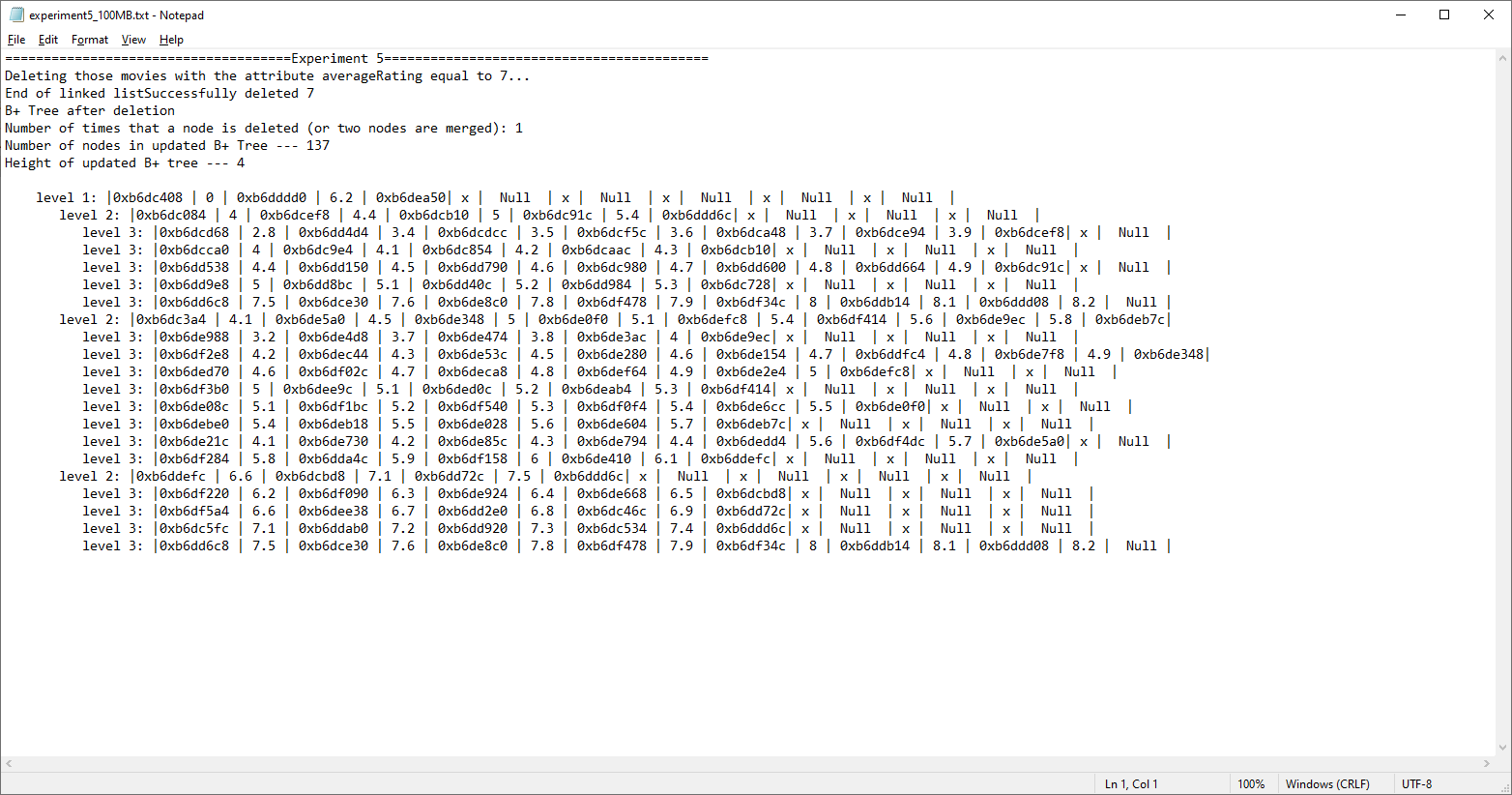


Рисунок 7.7. Результат эксперимента №5

# ЗАКЛЮЧЕНИЕ

В ходе курсового работы был изучен и реализован алгоритм B-дерева и B+-дерева, которые используют взаимодействие с диском через файловые системы операционной системы. Были изучены свойства, которые помогают выбрать оптимальный параметр ветвления для B-деревьев. Для анализа была взята база данных фильмов IMDB, численностью более 10 миллионов фильмов. Была проведена сравнительная характеристика B-дерева и B+-дерева, выявлены преимущества и недостатки каждого из алгоритмов.

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. 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.

# 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/