УДК 004.62

**Исследование эффективности сильно ветвящихся деревьев в задаче индексирования структурированных данных**

*на Конкурс научно-исследовательских работ студентов  
НИУ ВШЭ (Конкурс НИРС),  
направление: «Компьютерные науки»*

**Москва 2018**

# Реферат

40 c., 27 рис., 9 источн.

***Ключевые слова:*** *сильно ветвящиеся деревья, B-деревья, B+-деревья B\*-деревья, B\*+-деревья, индексирование структурированных данных, эффективность, сложность, СУБД.*

**Объект исследования** — сильно ветвящиеся деревья.

**Предмет исследования** — эффективность основных операций над сильно ветвящимися деревьями.

**Цель исследования** — исследование эффективности основных операций над сильно ветвящимися деревьями, структура которых задаётся определённым набором параметров, в задаче индексирования структурированных данных, в том числе, получение эмпирической оценки временной сложности основных операций, а также используемой памяти и числа дисковых операций, с такими деревьями, при различных параметрах построения дерева.

**Задачи исследования:**

1. выработка рабочих гипотез по оценке временной сложности основных операций, а также используемой памяти и числа дисковых операций;
2. реализация структур данных – B-дерева, B+-дерева, B\*-дерева на языке C++, а также инструмента для проведения экспериментов и вывода их результатов;
3. разработка новой модификации B-дерева – B\*+-дерева, обладающего свойствами B+-дерева и B\*-дерева, и предположительно дающего лучшие, чем B-дерево, результаты относительно вставки и удаления ключей;
4. построение схемы экспериментов, в том числе, создание набора изменяемых параметров построения B-дерева, B+-дерева, B\*-дерева и B\*+-дерева для проведения различных экспериментов;
5. проведение экспериментов (измерение сложности основных операций над деревьями – поиска, вставки и удаления элементов – при различных параметрах построения дерева, с экспериментальной оценкой занимаемой памяти и замером числа дисковых операций) согласно построенной схеме экспериментов;
6. интерпретация и анализ результатов экспериментов.

**Методы и алгоритмы исследования**

Методами решения являются стандартные методы дискретной математики и теории алгоритмов для исследования сложности операций и экспериментальное подтверждение теоретических утверждений.

Алгоритмы основных операций над сильно ветвящимися деревьями – поиска, вставки и удаления элементов – сами по себе являются предметом исследования данной работы.

Алгоритм проведения экспериментов в соответствии со схемой проведения экспериментов представлен в подразделе 3.2.

**Научная новизна** работы состоит в следующем:

1. выполнена экспериментальная оценка параметров эффективности операций над сильно ветвящимися деревьями, и приведено сравнение оценок для разных наборов параметров построения дерева и разных типов сильно ветвящихся деревьев;
2. разработана новая модификация B-дерева – B\*+-дерево, обладающее свойствами B+-дерева и B\*-дерева и дающее лучшие, чем B-дерево, результаты относительно вставки и удаления ключей.

**Достоверность научных результатов** подтверждена экспериментами с сильно ветвящимися деревьями, предусматривающими, в том числе, оценку времени выполнения каждой операции с деревом (вставка, поиск, удаление ключей), объёма используемой оперативной памяти во время операции, количества дисковых операций (чтения и записи).

**Практическая значимость:** результаты данной работы могут быть использованы учащимися и исследователями в области алгоритмов и структур данных, а также систем управления базами данных (СУБД), для проведения новых исследований в области, и для улучшения производительности СУБД.

**Результаты работы:**

1. разработаны гипотезы о различных параметрах эффективности (время выполнения, объём используемой оперативной памяти, число дисковых операций – чтения и записи) операций (вставка, поиск, удаление ключей) над четырьмя типами сильно ветвящихся деревьев (B-дерево, B+-дерево, B\*-дерево B\*+-дерево);
2. разработана новая модификация B-дерева – B\*+-дерево, дающая хорошие результаты (относительно времени) на операции удаления ключей из дерева (и лучшие, чем B-дерево, результаты относительно времени вставки ключей в дерево);
3. реализованы в виде библиотеки все четыре рассматриваемых типа сильно ветвящихся деревьев с операциями вставки, поиска и удаления на языке C++11;
4. разработан программный инструмент на языке C++11 для проведения экспериментов по индексации в дереве, поиску по индексу и экспериментов со стандартными операциями над деревом (вставка, поиск и удаление ключей), программный код представлен в репозитории: <https://github.com/Glost/btrees>;
5. построена схема экспериментов и проведены эксперименты, получены их результаты, из которых при помощи инструментария Python 2 построены графики зависимости параметров эффективности от порядка и размера дерева, которые приведены и проанализированы в настоящей работе;
6. часть гипотез подтверждена, обобщение дано в подразделе 4.2.

Содержание

[Реферат 2](#_Toc527207339)

[Содержание 5](#_Toc527207340)

[Обозначения и сокращения 6](#_Toc527207341)

[Введение 7](#_Toc527207342)

[1. Теоретические основы. Определения и понятия 8](#_Toc527207343)

[2. Основные операции с сильно ветвящимися деревьями 10](#_Toc527207344)

[2.1. B-дерево 10](#_Toc527207345)

[2.1.1. Поиск в B-дереве 11](#_Toc527207346)

[2.1.2. Вставка в B-дерево 11](#_Toc527207347)

[2.1.3. Удаление из B-дерева 13](#_Toc527207348)

[2.2. Модификации B-дерева 14](#_Toc527207349)

[2.2.1. B+-дерево 14](#_Toc527207350)

[2.2.2. B\*-дерево 15](#_Toc527207351)

[2.2.3. B\*+-дерево 16](#_Toc527207352)

[3. Схема исследования 17](#_Toc527207353)

[3.1. Схема экспериментов 17](#_Toc527207354)

[3.2. Алгоритм проведения экспериментов 18](#_Toc527207355)

[4. Разработанный инструмент для проведения экспериментов 19](#_Toc527207356)

[5. Результаты экспериментов и их анализ 20](#_Toc527207357)

[5.1. Результаты экспериментов 20](#_Toc527207358)

[5.1.1. Индексация сгенерированных с псевдослучайными строками CSV-файлов 20](#_Toc527207359)

[5.1.2. Индексация CSV-файла с реальными данными 29](#_Toc527207360)

[5.1.3. Операции с целочисленными ключами в дереве 32](#_Toc527207361)

[5.2. Анализ результатов экспериментов и выводы 38](#_Toc527207362)

[Заключение 39](#_Toc527207363)

[Список использованных источников 40](#_Toc527207364)

# Обозначения и сокращения

* **СУБД** — система управления базами данных.

# Введение

Сильно ветвящиеся деревья активно используются в различных задачах, связанных с индексированием структурированных данных для снижения времени, затрачиваемого на выполнение различных поисковых операций, в особенности, в СУБД. Так, например, B-деревья используются в индексе, создаваемом по умолчанию (без явного указания структуры данных) в СУБД PostgreSQL [6], они и их модификации используются в качества индекса в MySQL [5] и других популярных СУБД.

В настоящее время объём данных, обрабатываемых различными интернет-сервисами и другими системами, достаточно быстро растёт, растёт и объём баз данных. Возникает проблема работы с большими данными (big data) [4]. Для поддержания требуемого времени обращения к данным в рабочем состоянии, приходится разрабатывать новые и модифицировать существующие алгоритмы индексации баз данных.

Сильно ветвящиеся деревья отличаются от остальных типов деревьев тем, что обычно не хранятся целиком в оперативной памяти, сильно ветвящиеся деревья хранятся на постоянном запоминающем устройстве (например, жёстком диске), в то время как в оперативной памяти, как правило, хранится лишь необходимая для быстрого доступа информация (например, корневой узел). Это определено их назначением — сильно ветвящиеся деревья используются для хранения большого количества данных (например, индекса в СУБД), которые в оперативной памяти разместить представляется затруднительным. Хранение других типов деревьев (например, красно-чёрного дерева) на жёстком диске решило бы проблему, но создало бы другую: большинство операций над деревом стали бы весьма медленными, поскольку приходилось бы считывать с жёсткого диска по одной вершине, а операции с жёстким диском значительно более «дорогие» по времени, нежели операции с оперативной памятью. Сильно ветвящиеся деревья решают эту проблему хранением в одном узле более одного ключа, при этом с жёсткого диска все ключи узла считываются за один проход.

Данная работа посвящена исследованию эффективности индексирующих структур данных на основе B-деревьев измерением затрачиваемого на операцию со структурой (вставку ключа, поиск ключа, удаление ключа) времени, объёма выделяемой в процессе выполнения операций оперативной памяти и количества дисковых операций (чтения и записи). Исследуется зависимость эффективности операций над данными в дереве от типа дерева (B-дерево, B+-дерево, B\*-дерево или B\*+-дерево), порядка дерева и вида индексируемых данных. Будем считать, что эффективность выше, если значение соответствующей величины (времени, объёма памяти, количества дисковых операций) ниже.

Оставшаяся часть работы организована следующим образом. В разделе 1 вводятся необходимые определения и понятия. Раздел 2 описывает основные операции с сильно ветвящимися деревьями. Там же приводятся теоретические оценки их сложности. В разделе 3 приведена схема проведённого исследования. Раздел 4 описывает разработанный программный инструмент для проведения экспериментов (testbed). В разделе 5 представлены результаты экспериментов и их анализ. В заключении приведены основные выводы работы. После заключения приведён список использованных источников.

# 1. Теоретические основы. Определения и понятия

Введём понятие сильно ветвящегося дерева.

**Определение 1.** *Сильно ветвящееся дерево* представляет собой множество  
,

где:

* – множество узлов (вершин) дерева;
* – множество дуг дерева, подмножество декартова произведения , каждая дуга соединяет два узла дерева, где первый узел считается родительским узлом, а второй – дочерним узлом;
* – множество ключей, хранящихся в дереве;
* – порядок дерева (натуральное число), параметр построения дерева, определение которого будет введено позже;
* – функция, являющаяся отображением из множества узлов дерева во множество (ключей) – для узла функция возвращает множество – множество ключей, содержащихся в узле ;
* – функция, являющаяся отображением из результата применения функции к множеству узлов дерева и множества натуральных чисел во множество (ключей) либо пустое множество – для узла и натурального числа функция возвращает -й ключ в данном узле, либо пустое множество, если больше числа ключей в данном узле;
* – функция, являющаяся отображением из множества узлов дерева во множество (дуг) либо пустое множество – для узла функция возвращает множество – множество дуг, исходящих из узла , либо пустое множество, если узел является листовым (то есть не содержит исходящих дуг);
* – функция, являющаяся отображением из результата применения функции к множеству узлов дерева и множества натуральных чисел во множество (дуг) либо пустое множество – для узла и натурального числа функция возвращает -ю исходящую дугу в данном узле, либо пустое множество, если больше числа дуг в данном узле либо узел является листовым (то есть не содержит исходящих дуг);
* – корневой узел дерева, обязательно хранящийся в оперативной памяти.

Таким образом, сильно ветвящееся дерево – это структура данных (дерево), содержащее в одном узле (вершине) более одного элемента (ключа) и более одного указателя на дочерний узел.

Сильно ветвящееся дерево является деревом поиска. Из этого следует, что верно утверждение ,

где:

* – множество ключей, которых можно достичь при переходе по ребру, получаемому функцией ;
* – множество ключей, которых можно достичь при переходе по ребру, получаемому функцией ;
* – ключ, получаемый функцией .

То есть, для любого ключа верно, что он больше или равен любого ключа в его «левом» поддереве и меньше или равен любого ключа в его «правом» поддереве, если эти деревья существуют (то есть ключ не находится в листовом узле).

Для любого нелистового узла верно, что .

Теперь определим ранее упомянутый порядок дерева и введём понятие B-дерева – одного из самых распространённых типов сильно ветвящихся деревьев.

**Определение 2.** *Порядком (степенью) сильно ветвящегося дерева* называется такое целочисленное число, что значения минимального и максимального количества ключей в узле сильно ветвящегося дерева могут быть вычислены как линейная функция от .

**Определение 3.** *B-дерево* – сильно ветвящееся дерево, построенное таким образом, что если порядок дерева равен , то для любого некорневого узла дерева верно неравенство: , где – число ключей в узле. Корневой узел для непустого B-дерева содержит ключей, для пустого B-дерева – 0 ключей [9].

B-дерево является сбалансированным деревом, поэтому его высота будет равна , где – число ключей в дереве [9].

Также определим ряд ранее известных модификаций B-дерева.

**Определение 4.** *B+-дерево* – модификация B-дерева. В B+ -дереве настоящие ключи хранятся лишь в листьях дерева, а во внутренних узлах хранятся лишь ключи-маршрутизаторы, необходимые для поиска по дереву. Листья в B+ -дереве содержат ключей, где – порядок дерева, ограничения для внутренних узлов такие же, как и в B-дереве [3].

**Определение 5.** *B\*-дерево* – модификация B-дерева. Каждый узел заполняется не менее, чем на 2/3, а не 1/2 [1]. Для удобства вставки ключей примем, что в каждом некорневом узле содержится ключей, где – порядок дерева. Корневой узел для непустого B\*-дерева содержит ключей, для пустого B\*-дерева – 0 ключей.

Введём определение разработанной в рамках настоящей работы модификации B-дерева (вернее, B\*-дерева) – B\*+-дерева, сочетающего в себе свойства ранее известных B+-дерева и B\*-дерева.

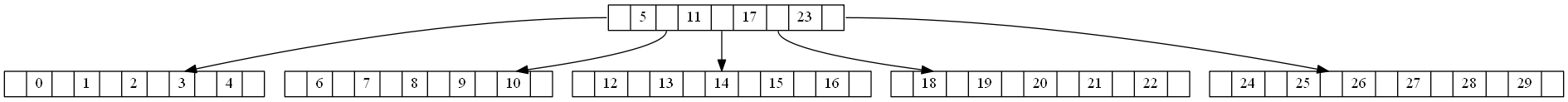
**Определение 6.** *B\*+-дерево* – модификация B\*-дерева. В B\*+-дереве настоящие ключи хранятся лишь в листьях дерева, а во внутренних узлах хранятся лишь ключи-маршрутизаторы, необходимые для поиска по дереву. Ограничения по количеству ключей в узлах совпадают с таковыми в B\*-дереве.

# 2. Основные операции с сильно ветвящимися деревьями

## 2.1. B-дерево

Наиболее распространённым и известным типом сильно ветвящихся деревьев является B-дерево. Рассмотрим выполнение операций над B-деревом. Под понятием эффективности данных операций мы объединим три параметра: сложность (по времени), объём необходимой памяти, число дисковых операций. Под дисковыми операциями мы будем иметь в виду чтение узла из долгосрочной памяти и запись узла в долгосрочную память.

Пример B-дерева приведён на рисунке 1.



*Рисунок 1. B-дерево порядка 6*

### 2.1.1. Поиск в B-дереве

Поиск в B-дереве представляет собой процедуру, во многом аналогичную поиску в бинарном дереве.

Поиск начинается с корня. Последовательно перебираются ключи в узле, до тех пор, пока поиск не перешагнёт последний ключ данного узла, либо не найдётся ключ, больший или равный искомому. Если найден ключ, равный искомому, то такой ключ возвращается в качестве результата. Если в данном узле не нашлось ключа, и узел является листом, то искомый ключ отсутствует в дереве. В противном случае процедура поиска рекурсивно выполняется в поддереве, предшествующем ключу, на котором остановилась процедура перебора ключей данного узла [9].

#### 2.1.1.1. Эффективность поиска

Во время поиска будет задействовано число узлов, не превышающее высоту дерева, кроме того, будет выполняться линейный перебор в каждом из задействованных узлов, длина каждого из них зависит от порядка дерева t. Таким образом, операция поиска в B-дереве имеет вычислительную сложность , где — порядок дерева, а — число ключей в дереве [9].

При реализации функции поиска в виде рекурсии потребуется памяти, где — порядок дерева, а — число ключей в дереве. Это связано с тем, что в памяти будут храниться ключи каждого пройденного узла, и количество пройденных узлов будет не больше высоты. Если реализовать данную функцию в виде цикла, то потребуется в один момент хранить лишь один узел. Тогда потребуется памяти, где t — порядок дерева.

Кроме того, поскольку, как уже сказано, число пройденных вершин не будет превышать высоту дерева, число дисковых операций будет равно .

### 2.1.2. Вставка в B-дерево

Вставка нового ключа в B-дерево осуществляется в уже существующий узел, а точнее — в один из листов. Перед вставкой рекурсивно выполняется поиск необходимого листа, аналогичный поиску ключа в B-дереве, однако при проходе через каждый заполненный узел для него вызывается процедура разбиения. Это гарантирует, что если будет необходимо разбить какой-то заполненный узел, то его родитель не будет заполнен [9].

#### 2.1.2.1. Разбиение заполненного узла

В B-дереве ни один узел не может иметь вершин больше, чем , где — порядок дерева [9]. Данное свойство поддерживает сбалансированность B-дерева.

В случае, если при процедуре вставки нового ключа в B-дерево будет обнаружен заполненный узел, он будет разбит на два незаполненных.

При данной процедуре средний ключ ( является нечётным числом, а значит, в любом заполненном узле можно будет чётко выделить средний ключ) перемещается в родительский узел — в место, соответствующее указателю на данное поддерево, а ключи, находящиеся слева и справа от среднего ключа образуют соответственно два новых узла.

Указатель на левый из этих узлов находится там же, где был указатель на «единый» узел, а указатель на правый из этих узлов — справа от перемещённого в родительский узел среднего ключа узла, который подвергся разбиению [9].

#### 2.1.2.2. Эффективность разбиения и вставки

Сложность операции разбиения не зависит от высоты дерева, поскольку при операции затрагиваются лишь два узла (родительский и разбиваемый), однако зависит от количества ключей в узле, поскольку в родительском узле выполняются линейные проходы для сдвига ключей и указателей на поддеревья влево, перед вставкой среднего ключа разбиваемого узла. Количество ключей в узле, в свою очередь, линейно зависит от порядка дерева . Таким образом, вычислительная сложность разбиения равна , где — порядок дерева, а n — число ключей в дереве.

Из-за того, что при операции разбиения затрагиваются лишь два узла (родительский и разбиваемый), объём используемой памяти при этой операции будет равен , где — порядок дерева, а число дисковых операций — .

Сложность операции вставки зависит от высоты дерева, количества ключей в узле и сложности операции разбиения, которая тоже зависит от количества ключей в узле. Количество ключей в узле зависит от порядка дерева . Соответственно, вычислительная сложность операции вставки равна , где — порядок дерева, а — число ключей в дереве.

При реализации функции вставки в виде рекурсии потребуется памяти, где — порядок дерева, а — число ключей в дереве. Это связано с тем, что в памяти будут храниться ключи каждого пройденного узла, и количество пройденных узлов будет не больше высоты. Если реализовать данную функцию в виде цикла, то потребуется в один момент хранить лишь один узел. Тогда потребуется памяти, где — порядок дерева.

Кроме того, поскольку, как уже сказано, число пройденных вершин не будет превышать высоту дерева, число дисковых операций будет равно .

### 2.1.3. Удаление из B-дерева

Алгоритм удаления ключа из B-дерева во многом представляет собой алгоритм, обратный вставке. Алгоритм выполняется рекурсивно, начиная от корня.

Удаление из листа тривиально — необходимо просто удалить из него соответствующий ключ и сдвинуть те ключи, которые находятся справа от него, влево [9].

Если же ключ находится во внутреннем узле дерева, то возможно несколько вариантов действий. Если дочерний узел, предшествующий удаляемому, содержит не менее ключей (где — порядок дерева), то в нём находится предшественник удаляемого ключа (наибольший ключ в поддереве, корнем которого является дочерний узел, предшествующий удаляемому ключу), который рекурсивно удаляется из данного поддерева, после чего удаляемый ключ заменяется данным предшественником.

Если же данный дочерний узел содержит менее t ключей, но дочерний узел, следующий за удаляемым, содержит не менее t ключей, то аналогичная процедура проводится в поддереве, следующем за удаляемым ключом, с ключом, следующим за удаляемым (наименьшим ключом в поддереве, корнем которого является дочерний узел, следующий за удаляемым ключом). Если же оба этих дочерних узла содержат по ключей, то выполняется процедура их объединения, обратная разбиению узла на два: средним ключом объединённого узла становится удаляемый ключ, для этого он переносится в конец дочернего узла, предшествующего ему. Все ключи из дочернего узла, следующего за удаляемым, также переносятся в дочерний узел, ранее предшествовавший ему, после удаляемого ключа. После выполнения процедуры объединения удаляемый ключ рекурсивно удаляется из вновь полученного объединённого узла [9].

Если ключ отсутствует в текущем внутреннем узле дерева, то определяется, в каком из поддеревьев он должен содержаться (если он вообще содержится в дереве), после чего он рекурсивно удаляется из соответствующего поддерева. При этом, если соответствующий дочерний узел содержит лишь ключей, но один из его непосредственных соседей (дочерних узлов, отделённых от него одним ключом), содержит не менее t ключей, то ключ-разделитель между дочерним узлом, являющимся корнем поддерева, где должен находиться удаляемый ключ, и соответствующим его непосредственным соседом, помещается в дочерний узел, являющийся корнем поддерева, где должен находиться удаляемый ключ, а на его место помещается крайний ключ из соответствующего непосредственного соседа. Соответствующий перенесённому крайнему ключу указатель на потомка переносится из соответствующего непосредственного соседа в дочерний узел, являющийся корнем поддерева, где должен находиться удаляемый ключ. Если и корень такого поддерева, и оба его непосредственных соседа содержат по ключей, то выполняется операция объединения с любым из его непосредственных соседей. При этом бывший ключ-разделитель становится медианой нового объединённого узла [9].

Если ключ отсутствует в текущем листе, то он отсутствует в дереве.

#### 2.1.3.1. Эффективность удаления

Вычислительная сложность операции удаления зависит от количества пройденных вершин, не превышающего высоты дерева, и порядка дерева, и равняется , где — порядок дерева, а — число ключей в дереве [9].

В связи с потенциальной необходимостью рекурсивно удалять предшествующие или следующие за удаляемым ключом ключи из соответствующих поддеревьев, операцию удаления невозможно реализовать как цикл, её можно реализовать лишь рекурсивно, поэтому объём используемой оперативной памяти будет зависеть от высоты дерева и его порядка, и, соответственно, будет равен , где — порядок дерева, а — число ключей в дереве.

Количество дисковых операций также зависит от высоты дерева, и, соответственно, равняется , где — порядок дерева, а — число ключей в дереве [9].

## 2.2. Модификации B-дерева

### 2.2.1. B+-дерево

B+-дерево является модификацией B-дерева, в которой настоящие ключи хранятся лишь в листьях дерева, а во внутренних узлах хранятся лишь ключи-маршрутизаторы. Ключи-маршрутизаторы хранятся в B+-дереве для возможности рекурсивного поиска настоящих ключей [3].

В B+-дереве максимальным числом узлов в листовых вершинах является , а не  
, где — порядок дерева. Это связано с тем, что при разбиении листа на две вершины, в родительскую вершину можно будет вставить лишь ключ-маршрутизатор (поскольку родительская вершина не является листом), соответственно, лист необходимо разбить ровно на две части. Отсюда же следует, что минимальным числом узлов в листовых вершинах будет являться , а не , где — порядок дерева [3].

Новые ключи-маршрутизаторы генерируются при вставке нового ключа в дерево, а если быть точнее, то при разбиении заполненного листа на две части, когда необходимо выбрать новый ключ-маршрутизатор, помещаемый в родительский узел. Одним из возможных вариантов для генерации нового ключа-маршрутизатора в этом случае является выбор в качестве такового последнего ключа из левой части разбиваемого листа [3].

В случае выполнения объединения узлов при операции удаления ключа из B+-дерева, ключ-маршрутизатор, созданный при разбиении соответствующих узлов для помещения в родительский узел, удаляется из дерева.

В остальном операции поиска, вставки и удаления в B+-дереве, в целом, аналогичны таковым в B-дереве. Асимптотика у этих операций по этой причине также совпадает с асимптотикой соответствующих операций в B-дереве, если используется алгоритм генерации новых ключей-маршрутизаторов, работающий за по вычислительной сложности, объёму занимаемой памяти и числу дисковых операций (например, описанный выше выбор в качестве нового ключа-маршрутизатора последнего ключа из левой части разбиваемого листа дерева).

Однако фактическая эффективность на деле должна отличаться от таковой в B-дереве. Поиск ключа всегда будет заканчиваться в листе, так как во внутренних вершинах хранятся лишь ключи-маршрутизаторы, что должно увеличить среднюю вычислительную сложность поиска и число необходимых дисковых операций для него, и приблизить их к максимальным. С другой стороны, удаление ключа всегда будет выполняться из листа, что, в среднем, должно его упростить. Операция вставки должна быть приблизительно равна по эффективности.

### 2.2.2. B\*-дерево

B\*-дерево отличается от обычного B-дерева тем, что каждый узел заполняется не менее, чем на 2/3 от размера узла, а не на 1/2, как в B-дереве. Как говорилось в определении, для удобства вставки ключей примем, что в каждом некорневом узле содержится ключей, где – порядок дерева. Корневой узел для непустого B\*-дерева содержит ключей, для пустого B\*-дерева – 0 ключей. По этой причине, вместо традиционного разбиения узла, происходит перераспределение ключей между соседними узлами-потомками, либо, если нет незаполненных соседей, то узел разбивается на три (а не на две, как в B-дереве) части [8] [1]. Данное решение имеет смысл по той причине, что разбиение вершины — достаточно дорогостоящая (по времени и числу дисковых операций) операция из-за того, что в неё входит создание новой вершины (на которую необходимо добавлять указатель в вершине родителе, со сдвигом остальных указтаелей, которую необходимо учитывать в метаданных дерева, и т. д.).

Вставка в B\*-дерево аналогична вставке в B-дерево, но имеет несколько отличий. Прежде всего, при вставке как можно дольше избегаются операции разбиения вершин. Для этого применяются операции переливания. В случае, если наша вершина заполнена, но у неё есть хотя бы один сосед с числом ключей меньшим, чем максимальное, то мы можем «отдать» ей соответствующий крайний ключ, переместив его на место ключа в узле-родителе, разделяющего данный узел и соответствующего его соседа, а ключ, находившийся ранее на этом месте, сделать крайним в соответствующим узле-соседе. В случае, если таких соседей нет, то разделим две вершины (текущую вершину и одного из её соседей) на три вершины. При этом один из ключей «правой» из этих двух вершин переместится в узел-родителя. В случае, если происходит разбиение корневой вершины, то она разбивается на две вершины, как это реализовано в B-дереве [2].

Алгоритм поиска в B\*-дереве полностью соответствует таковому в B-дереве.

Удаление в B\*-дереве также аналогично удалению в B-дереве, за тем лишь исключением, что вместо объединения двух вершин в одну применяется объединение трёх вершин в две. При этом ключ в узле-родителе, разделяющий «средний» и «правый» узел из трёх объединяемых, перемещается в «правый» из двух созданных при объединении узлов. В случае, если объединяются узлы-потомки корневого узла (которых, если они есть, может быть только ровно две), то такое объединение происходит так же, как происходит объединение вершин в B-дереве — две вершины объединяются в одну, единственный ключ в корневом узле перемещается в объединённую вершину, которая становится новым корнем дерева [2].

Операция поиска по эффективности должна быть аналогичной таковой в B-дереве, операции вставки и удаления — аналогичные таковым в B-дереве, но с учётом иной процедуры разбиения заполненного узла. Однако, изменения процедуры разбиения не делают её зависимой от высоты дерева, а зависимость от порядка дерева остаётся линейной. Соответственно, асимптотика у операций поиска, вставки и удаления в B\*-дереве совпадает с таковой у соответствующих операций в B-дереве. Однако фактическая эффективность на деле должна отличаться от эффективности этих операций в B-дереве, поскольку из-за того, что каждый узел более заполнен, высота дерева будет меньше, но будет увеличена линейная составляющая. Кроме того, в B\*-дереве должны реже происходить операции разбиения вершин, в связи с этим можно ожидать более быстрой вставки в дерево.

### 2.2.3. B\*+-дерево

B\*+-дерево является разработанной в рамках настоящей работы модификацией B-дерева (также оно является модификацией B\*-дерева). Его можно считать смесью B\*-дерева и B+-дерева — оно аналогично B\*-дереву, за тем лишь исключением, что, как и в B+-дереве, реальные ключи хранятся лишь в листьях, а в остальных узлах — лишь ключи-маршрутизаторы, создающиеся при разбиении листовых узлов. Ограничения по количеству ключей в узлах совпадают с таковыми в B\*-дереве.

Асимптотика основных операций такая же, как и у B\*-дерева, однако, фактическая сложность может отличаться. Поиск, аналогично B+-дереву всегда будет завершаться в листовом узле, а значит, в среднем должен стать дольше, чем поиск в B\*-дереве. Операция удаления в среднем должна быть проще, чем в B\*-дереве, т. к. удаляемый ключ теперь гарантированно находится в листе. Операция вставки по сложности в среднем не должна измениться по сравнению с B\*-деревом.

# 3. Схема исследования

## 3.1. Схема экспериментов

Схема экспериментов включает набор данных, необходимых для осуществления расчётов. Отдельный эксперимент включает запись о каждом из экспериментов, которая состоит из следующих параметров:

1. тип дерева (B-дерево, B+-дерево, B\*-дерево либо B\*+-дерево);
2. порядок дерева;
3. количество ключей для замера параметров эффективности вставки, поиска и удаления целочисленных ключей в дереве;
4. путь к файлу для индексации;
5. ключ (имя) строки, для замера времени поиска по индексу.

Для измерения параметров эффективности при индексации сгенерированных CSV-файлов со случайными строками используются комбинации следующих параметров построения дерева:

* тип дерева: B-дерево, B+-дерево, B\*-дерево, B\*+-дерево;
* порядок дерева: 50, 100, 500, 1000, 1500, 2000;
* распределение «имён» (ключей) строк (см. подраздел 4.1) в CSV-файле для индексации: все ключи одинаковые, ключи равномерно распределены (каждый в среднем встречается в файле 1000 раз), все ключи разные;
* количество строк в CSV-файле для индексации: 25000, 50000, 75000, 100000.

Для измерения параметров эффективности при индексации реального CSV-файла с логами выполнения различных медицинских процедур в одном из госпиталей (см. описание в п. 5.1.2) используются комбинации следующих параметров построения дерева:

* тип дерева: B-дерево, B+-дерево, B\*-дерево, B\*+-дерево;
* порядок дерева: 50, 100, 500, 1000, 1500, 2000.

Для измерения параметров эффективности при операциях с целочисленными ключами (int — 32-битное целое число) в дереве используются комбинации следующих параметров построения дерева:

* тип дерева: B-дерево, B+-дерево, B\*-дерево, B\*+-дерево;
* порядок дерева: 50, 100, 500, 1000, 1500, 2000;
* количество ключей в дереве: 50, 100, 500, 1000, 5000, 10000, 50000, 100000.

## 3.2. Алгоритм проведения экспериментов

Алгоритм проведения экспериментов для каждого определённого набора параметров построения дерева (из ранее построенной схемы экспериментов) включает следующие шаги:

1. применение данного набора параметров для построения дерева;
2. измерение необходимых для исследования показателей (см. ниже в данном подразделе);
3. данные экспериментов агрегируются, строятся графики;
4. результаты набора экспериментов интерпретируются и анализируются.

В рамках каждого эксперимента выполняются следующие операции:

1. вставка, поиск и удаление целочисленных ключей (int — 32-битное целое число) в дереве;
2. индексация CSV-файла по ключам (где ключ («имя») каждой строки – её первая ячейка);
3. поиск случайно выбранного (из имеющихся в файле) ключа по индексу, извлечение из индекса сдвига данного ключа от начала файла в байтах и проверка наличия данного ключа на соответствующем месте в CSV-файле.

В качестве показателей эффективности по итогам экспериментов рассчитываются:

1. время выполнения (в мс);
2. объём используемой структурами данных оперативной памяти (в байтах) – максимальный за время операции;
3. количество дисковых операций (чтения и записи).

# 4. Разработанный инструмент для проведения экспериментов

Для проведения экспериментов разработан инструмент (testbed) на языке C++11. Использовалась стандартная библиотека C++11 Standard Library и библиотека Google Test (gtest) для разработки модульных тестов (unit tests).

В первую очередь, были реализованы структуры данных – B-дерево, B+-дерево, B\*-дерево и B\*+-дерево – в виде библиотеки *btrees\_lib*. Для структур данных также написаны модульные тесты (unit tests) с использованием инструментария библиотеки Google Test (gtest). После этого был реализован сам инструмент для проведения экспериментов – *btrees\_exp*.

Инструмент принимает на вход в качестве аргумента командной строки адрес (в файловой системе) CSV-файла со схемой экспериментов. Данный файл имеет следующий формат – в каждой строке (начиная со второй) указываются параметры эксперимента, в порядке, указанном в подразделе 3.2.

На выход инструмент подаёт CSV-файл с результатами экспериментов. Каждая строка (начиная со второй) соответствует строке с таким же порядковым номером в CSV-файле со схемой экспериментов. В каждой строке (начиная со второй) находятся следующие данные (в указанном порядке):

* 1. номер эксперимента;
  2. время вставки всех целочисленных ключей в дерево;
  3. время поиска всех целочисленных ключей в дереве;
  4. время удаления всех целочисленных ключей из дерева;
  5. время индексации файла;
  6. время поиска по индексу;
  7. объём (максимальный за время операции) используемой оперативной памяти для вставки всех целочисленных ключей в дерево;
  8. объём (максимальный за время операции) используемой оперативной памяти для поиска всех целочисленных ключей в дереве;
  9. объём (максимальный за время операции) используемой оперативной памяти для удаления всех целочисленных ключей из дерева;
  10. объём (максимальный за время операции) используемой оперативной памяти для индексации файла;
  11. объём (максимальный за время операции) используемой оперативной памяти для поиска по индексу;
  12. количество дисковых операций (чтения и записи) для вставки всех целочисленных ключей в дерево;
  13. количество дисковых операций (чтения и записи) для поиска всех целочисленных ключей в дереве;
  14. количество дисковых операций (чтения и записи) для удаления всех целочисленных ключей из дерева;
  15. количество дисковых операций (чтения и записи) для индексации файла;
  16. количество дисковых операций (чтения и записи) для поиска по индексу;
  17. максимальная достигнутая глубина дерева (высота дерева) при поиске всех целочисленных ключей в дереве;
  18. максимальная достигнутая глубина дерева (высота дерева) при поиске по индексу.

Исходный разработанного программного инструмента опубликован и доступен в открытом репозитории на портале GitHub по ссылке: <https://github.com/Glost/btrees>

# 5. Результаты экспериментов и их анализ

## 5.1. Результаты экспериментов

При помощи программного инструмента получены результаты 504 экспериментов с операциями над сильно ветвящимися деревьями, согласно построенной схеме экспериментов. По результатам данных экспериментов при помощи пакетов анализа данных matplotlib и pandas языка Python 2 и инструмента Jupyter Notebook построены различные графики зависимости каждого из параметров эффективности (время выполнения, используемая память, число дисковых операций) операции с деревом от порядка дерева либо от количества ключей в нём. Построено 213 графиков, часть из них (наиболее важные и значимые) приведена в настоящем отчёте. Остальные можно найти в репозитории по ссылке: <https://github.com/Glost/btrees/tree/master/root/prj/0.1/sol/samples/btrees/data_csv_files>

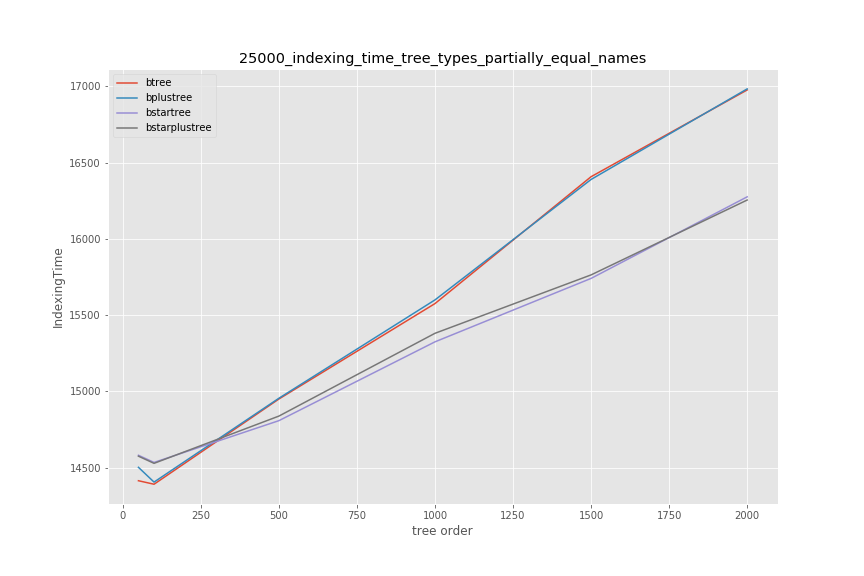
Все графики для времени выполнения операции в качестве единицы измерения используют миллисекунды, для памяти – байты, везде под словом «память», если не указано иное, подразумевается оперативная память.

### 5.1.1. Индексация сгенерированных с псевдослучайными строками CSV-файлов

Согласно схеме экспериментов, в работе были сгенерированы CSV-файлы с 25000, 50000, 75000 и 100000 строками. Значение первой ячейки каждой из строк является «именем» (ключом) данной строки и сохраняется в дереве вместе с отступом (в байтах) данной строки от начала индексируемого файла. Для каждого из этих размеров файла были сгенерированы:

* 1. файл, где все «имена» (ключи) строк равны;
  2. файл, где каждое «имя» (ключ) строки встречается в среднем 1000 раз в файле (равномерное распределение);
  3. файл, где все «имена» (ключи) строк разные.

Приведём график зависимости времени индексации от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 25000 строк (рисунок 2).



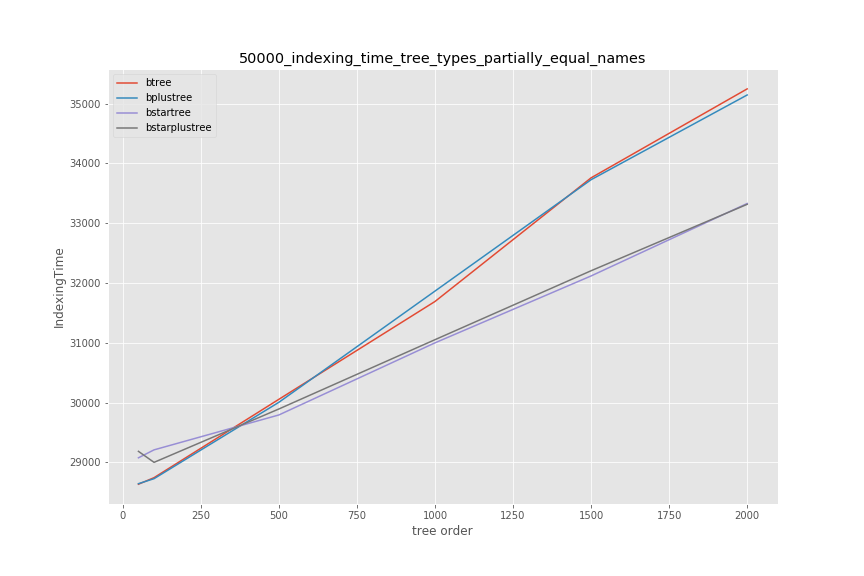
*Рисунок 2. График зависимости времени индексации от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 25000 строк*

Каждая из линий на данном графике соответствует одному из типов сильно ветвящихся деревьев. Как мы можем заметить, графики B-дерева и B+-дерева приблизительно совпадают, в то время как B\*-дерево и B\*+-дерево дают значительно лучший результат (меньшее время индексации) на больших значениях порядка дерева. При этом графики B\*-дерева и B\*+-дерева приблизительно совпадают.

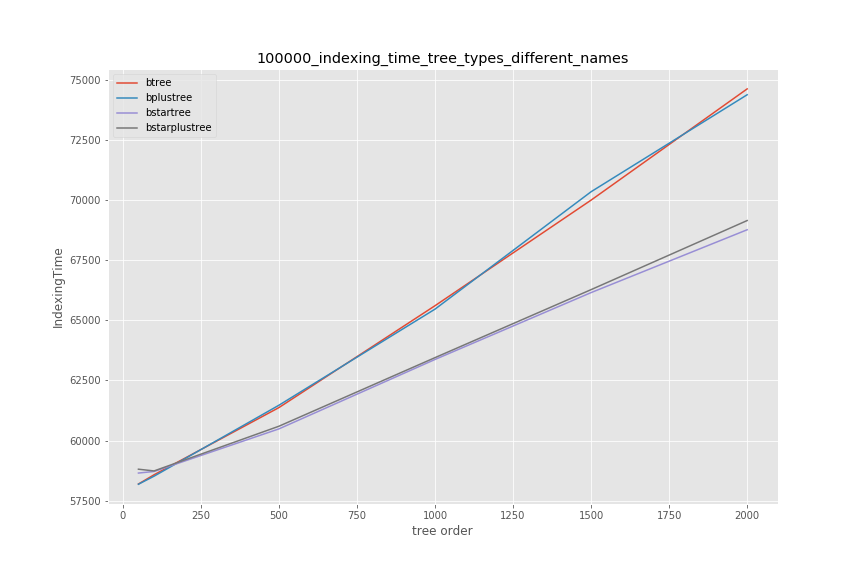
Таким образом, снижение числа разбиений в B\*-дереве (и производном от него B\*+-дереве) действительно ускоряет процесс вставки, как и предполагалось. В то же время, видно, что операция вставки в B+-дереве приблизительно совпадает по времени с таковой в B-дереве, а в B\*+ она приблизительно совпадает по времени с таковой в B\*-дереве, что соответствует изначальным предположениям.

Кроме того, видно, что если в самом начале графика (на меньших порядках) время уменьшается с ростом порядка дерева, что связано с уменьшением числа необходимых дисковых операций, то в дальнейшем время растёт, что связано с увеличением линейной составляющей сложности операции вставки при росте порядка дерева. Также видно, что график действительно похож на логарифмический.

Эти выводы подтверждаются аналогичными данными экспериментов с другими комбинациями параметров. Например, эксперимента с файлом, состоящим из 50000 строк, в котором ключи строк встречаются в среднем по 1000 раз в файле (рисунок 3), и эксперимента с файлом, состоящим из 100000 строк, в котором все ключи строк разные (рисунок 4).

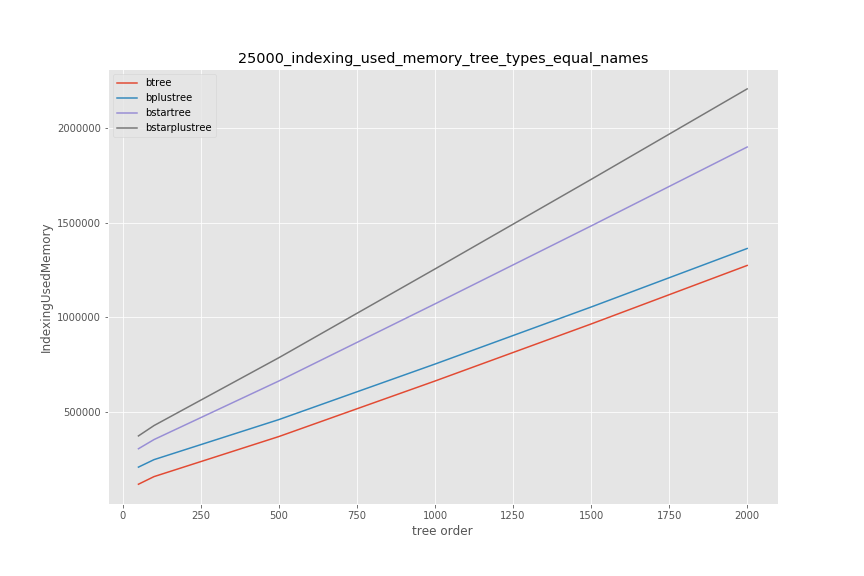


*Рисунок 3. График зависимости времени индексации от порядка дерева для файла, где каждое «имя» (ключ) строки встречается в среднем 1000 раз в файле, с размером 50000 строк*



*Рисунок 4. График зависимости времени индексации от порядка дерева для файла, где все «имена» (ключи) строк разные, с размером 100000 строк*

Рассмотрим график зависимости объёма занимаемой памяти (максимальной за время операции) во время индексации от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 25000 строк (рисунок 5).

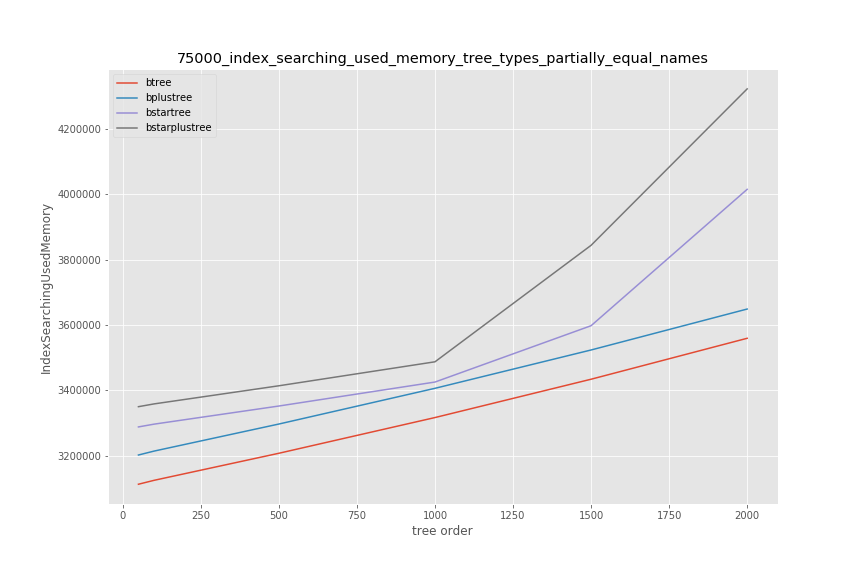


*Рисунок 5. График зависимости объёма занимаемой памяти (максимальной за время операции) во время индексации от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 25000 строк*

Из данного графика видно, что уменьшение времени индексации достигнуто ценой большего объёма занимаемой памяти для B\*-дерева и B\*+-дерева.

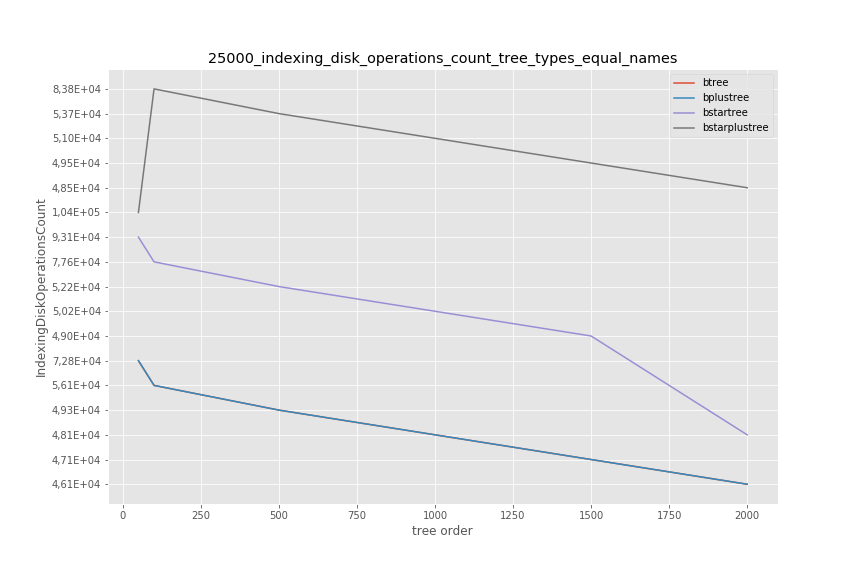
Кроме того, можно заметить, что с ростом порядка дерева размер используемой оперативной памяти также растёт, т. к. большие по длине узлы дерева занимают больший объём оперативной памяти.

В случае рассмотрения аналогичного графика для файла, где ключи строк равномерно распределены, и который содержит 75000 строк (рисунок 6), результаты приводят к тем же выводам, хотя графики памяти, занимаемой при индексации B+-деревом и B\*-деревом, значительно ближе.



*Рисунок 6. График зависимости объёма занимаемой памяти (максимальной за время операции) во время индексации от порядка дерева для файла, где «имена» (ключи) строк равномерно распределены, с размером 75000 строк*

Рассмотрим также график зависимости числа дисковых операций (чтения и записи) от порядка дерева, для файла, где все «имена» (ключи) строк равны, с размером 25000 строк (рисунок 7).

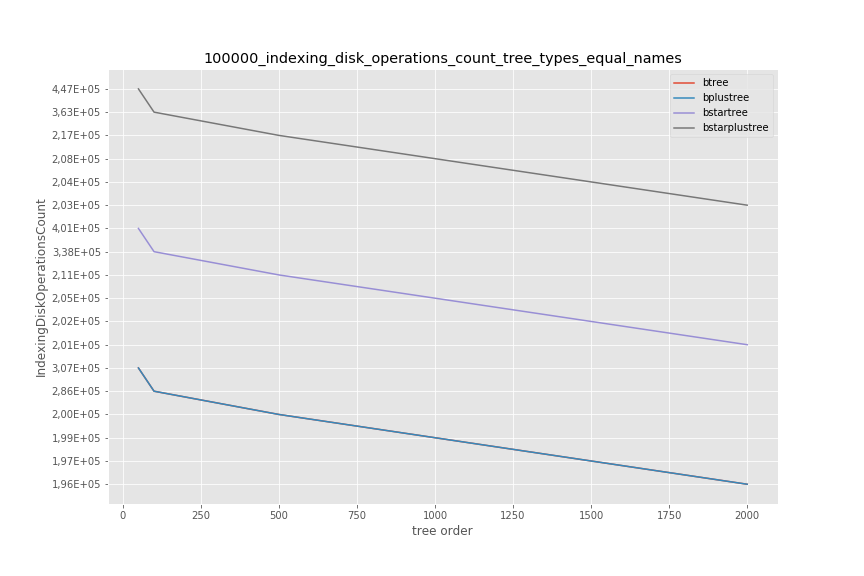


*Рисунок 7. График зависимости числа дисковых операций (чтения и записи) во время индексации от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 25000 строк*

В данном графике видно, что число дисковых операций в B\*-дереве и в B\*+-дереве значительно превышает соответствующие показатели в B-дереве и B+-дереве, что недостижимо интуитивным путём, т. к. обычно с ростом числа дисковых операций растёт и время выполнения всей операции со структурой данных, что, однако, в данном случае не выполняется. При этом число дисковых операций в B\*+-дереве превышает его в B\*-дереве.

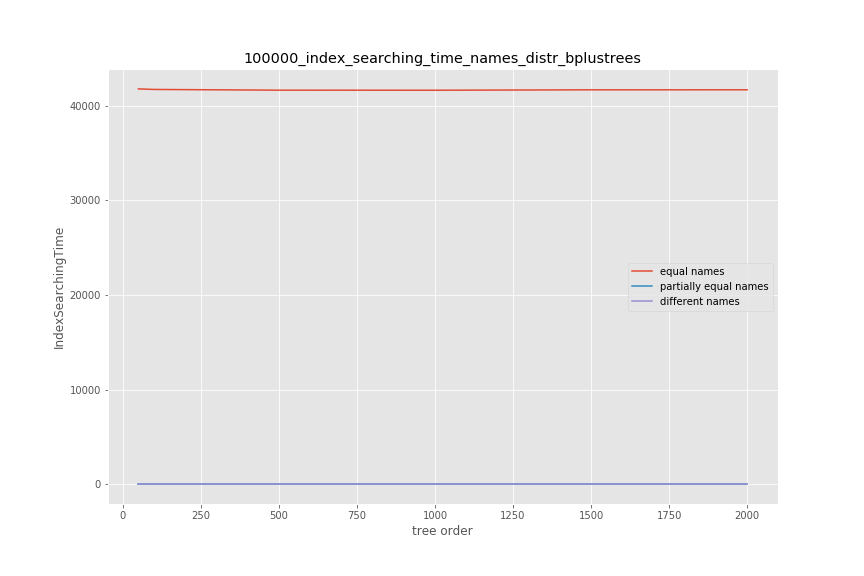
Также мы можем заметить, что при увеличении порядка дерева число дисковых операций в среднем снижается, так как требуется считывать с диска меньше узлов. В то же время, как было показано выше, время выполнения операций с ростом порядка дерева увеличивается, и это означает, что уменьшение количества дисковых операций не нивелирует увеличение линейной составляющей сложности.

Аналогичные выводы можно сделать и из графика зависимости числа дисковых операций во время индексации от порядка дерева для файла, в котором все ключи строк одинаковые, и который содержит 100000 строк (рисунок 8).



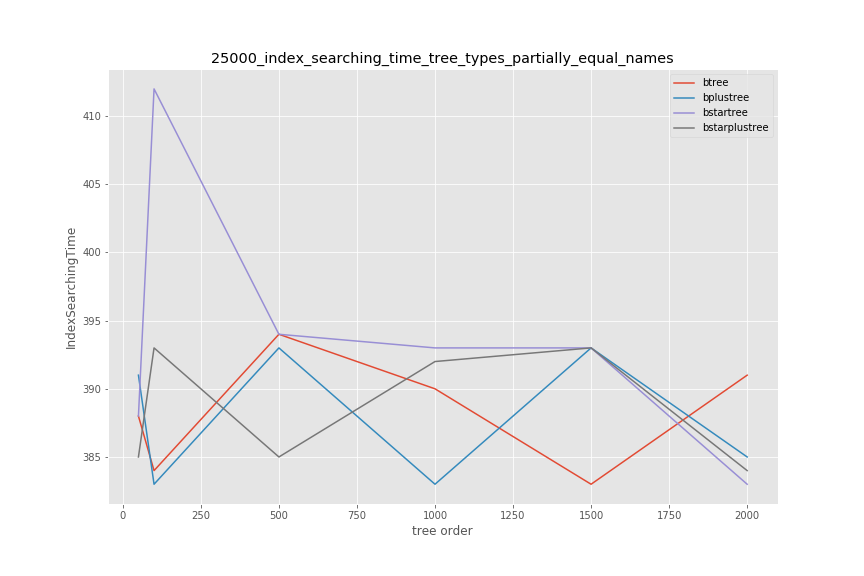
*Рисунок 8. График зависимости числа дисковых операций (чтения и записи) во время индексации от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 100000 строк*

Если рассматривать поиск по построенному индексу (сильно ветвящемуся дереву) всех вхождений случайно выбранного ключа в файле, то время его выполнения стремится к нулю при разных и равномерно распределённых ключах строк в CSV-файле, и к максимуму при равных ключах, что подтверждается графиком на рисунке 9. Это верно для всех типов деревьев, и объясняется тем, что при первых двух случаях, как правило, достаточно одного прохода от корня дерева в направлении к его листьям, в то время, как в последнем случае требуется обходить всё дерево.



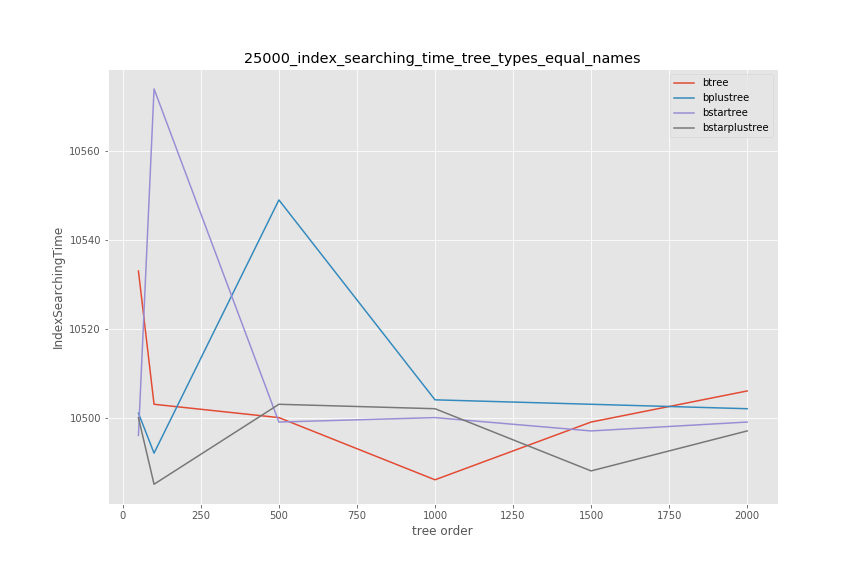
*Рисунок 9. График зависимости времени поиска по индексу, для трёх типов файлов, от порядка дерева*

Кроме того, рассмотрим график зависимости времени выполнения поиска по индексу от порядка дерева, для файла с равномерно распределёнными ключами (рисунок 10). Мы видим, что невозможно определить какую-либо монотонную зависимость времени поиска от порядка дерева, равно как и от типа дерева, хотя B\*-дерево здесь даёт худший результат (при порядке дерева меньшем, чем 1500).



*Рисунок 10. График зависимости времени поиска по индексу от порядка дерева для файла, где «имена» (ключи) строк равномерно распределены, с размером 25000 строк*

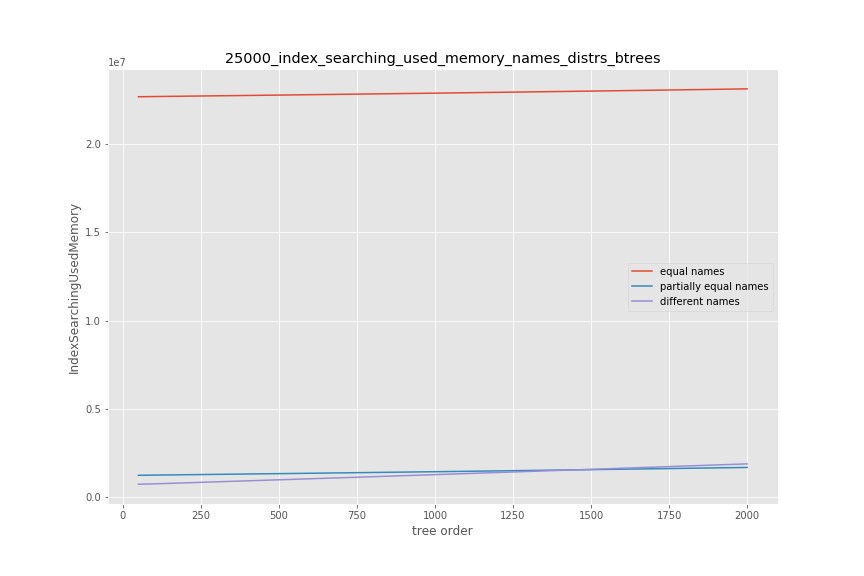
К аналогичным выводам ведёт график поиска по индексу файла, где все ключи строк равны, на рисунке 11.



*Рисунок 11. График зависимости времени поиска по индексу от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 25000 строк*

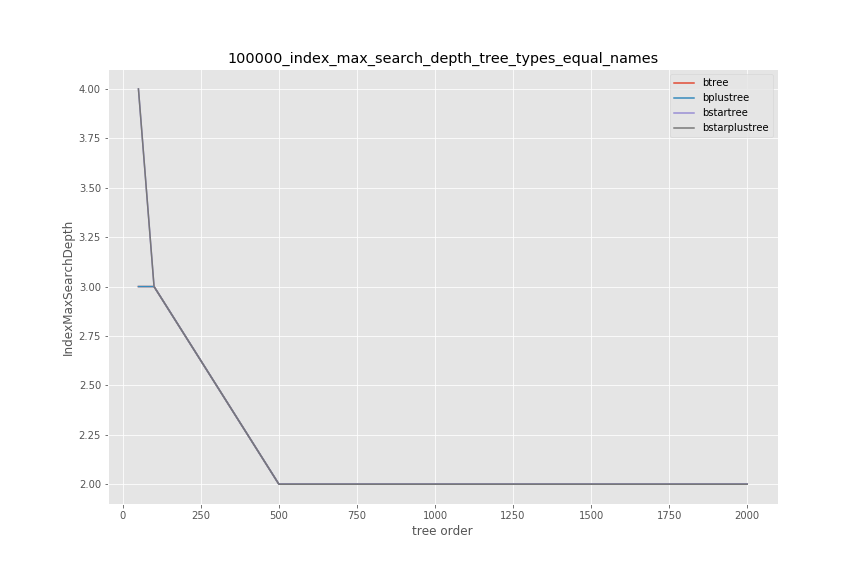
Здесь «худший» результат уже даёт B+-дерево, а не B\*-дерево, как на прошлом графике.

Необходимо отметить, что для файлов с равными, равномерно распределёнными и разными ключами строк, объём занимаемой памяти при поиске по индексу ведёт себя аналогично времени поиска по индексу. Это подтверждается графиком на рисунке 12.



*Рисунок 12. График зависимости объёма используемой памяти при поиске по индексу, для трёх типов файлов, от порядка дерева*

Также представим пример графика зависимости высоты дерева от его порядка (рисунок 13).



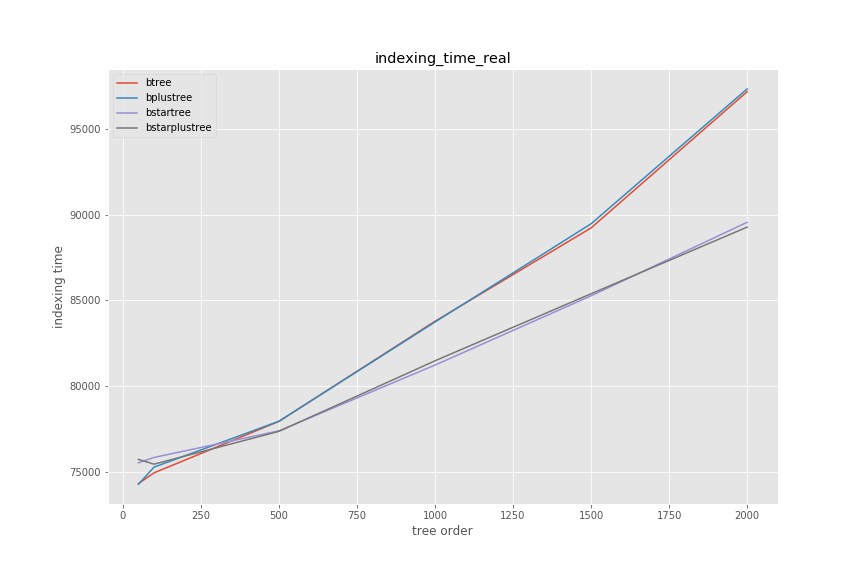
*Рисунок 13. График зависимости высоты дерева от порядка дерева для файла, где все «имена» (ключи) строк равны, с размером 100000 строк*

По графику видно, что при больших значениях порядка дерева его высота становится равна 2 – дерево состоит только из корня и листьев.

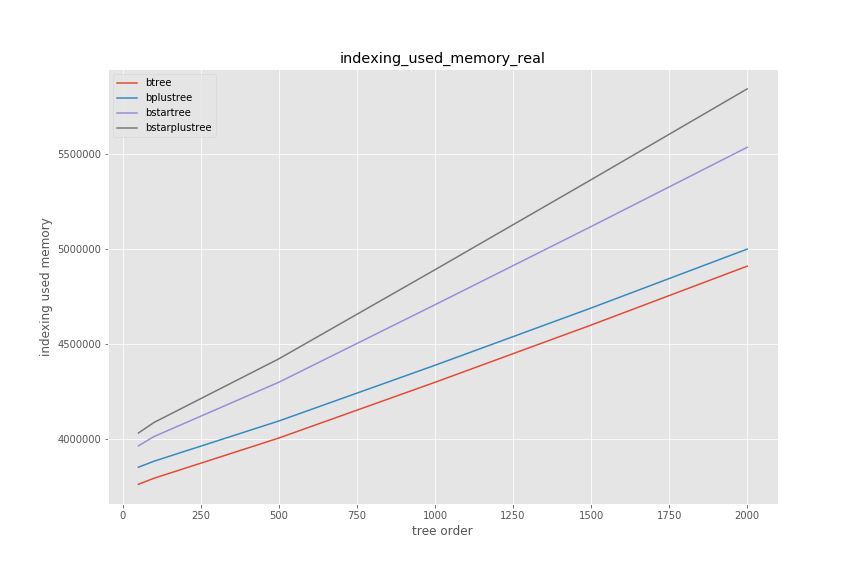
### 5.1.2. Индексация CSV-файла с реальными данными

Для индексации был выбран CSV-файл из открытого доступа (с сайта Process Mining[[1]](#footnote-1) Технологического университета Эйндховена TU/e, Нидерланды [7]) с логами выполнения различных медицинских процедур в одном из госпиталей, записанных в информационную систему госпиталя. Файл загружен в XES-формате и конвертирован в CSV-формат. Размер данного файла составляет 150291 строк.

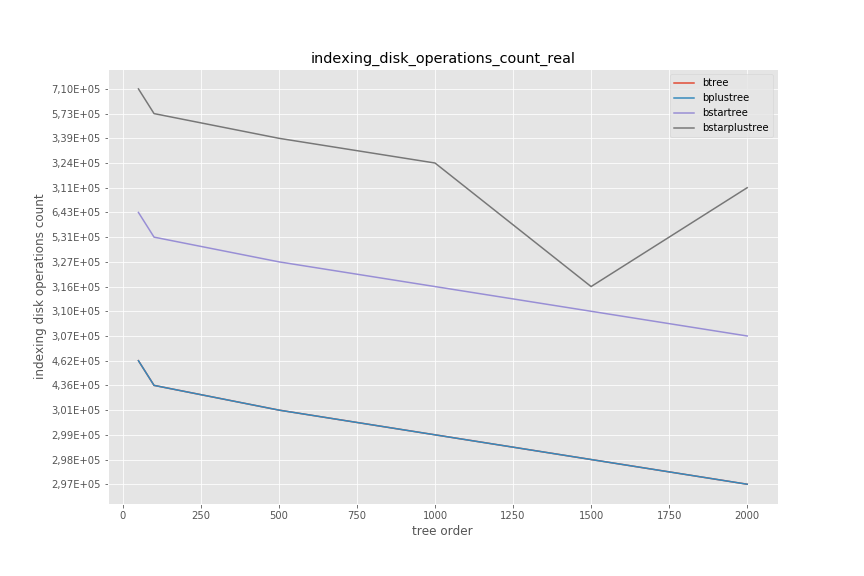
Результаты экспериментов с данным файлом не имеют принципиальных отличий от таковых со сгенерированными файлами. В качестве примера для подтверждения приведём графики на рисунках 14 – 18.



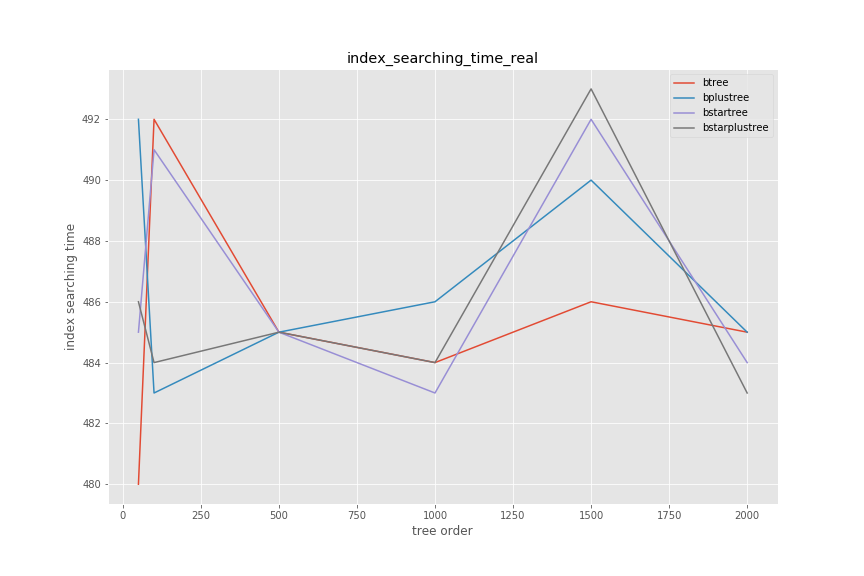
*Рисунок 14. График зависимости времени индексации от порядка дерева для файла с реальными данными*



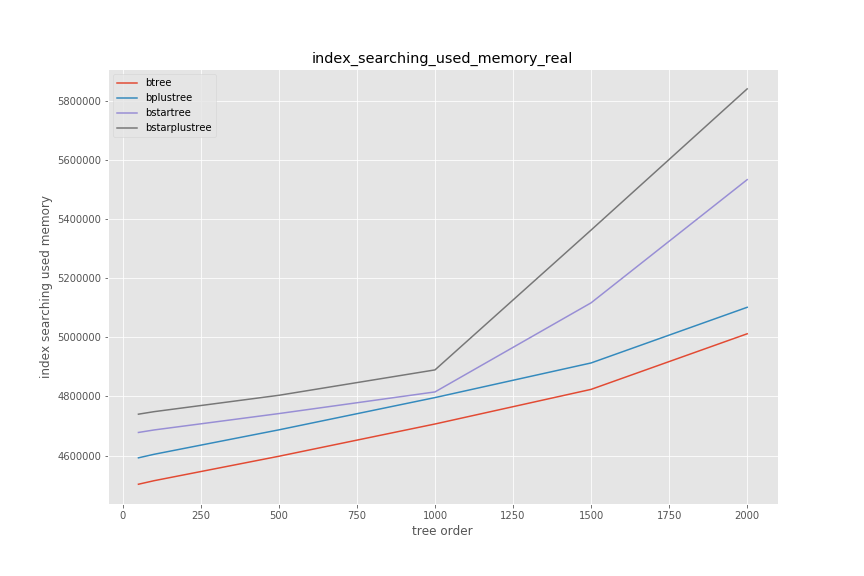
*Рисунок 15. График зависимости объёма используемой памяти при индексации от порядка дерева для файла с реальными данными*



*Рисунок 16. График зависимости количества дисковых операций при индексации от порядка дерева для файла с реальными данными*



*Рисунок 17. График зависимости времени поиска по индексу от порядка дерева для файла с реальными данными*



*Рисунок 18. График зависимости объёма используемой памяти при поиске по индексу от порядка дерева для файла с реальными данными*

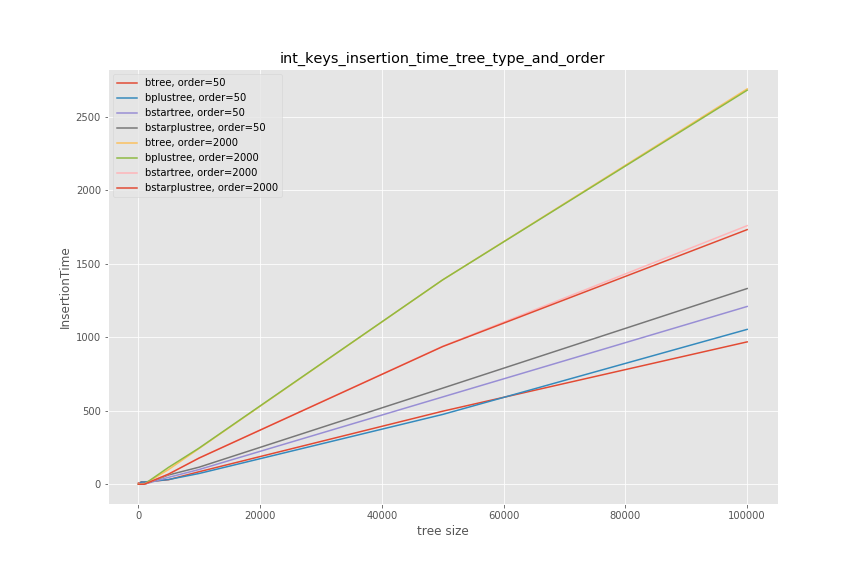
### 5.1.3. Операции с целочисленными ключами в дереве

В рамках данного блока экспериментов в дерево вставляется определённое число целочисленных (int — 32-битное целое число) ключей (положим его равным N), от 0 до N – 1, после этого проводится поиск по дереву каждого из ключей, и, затем, удаление.

Построены графики зависимости значений параметров эффективности различных операций от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000.

#### 5.1.3.1. Вставка в дерево

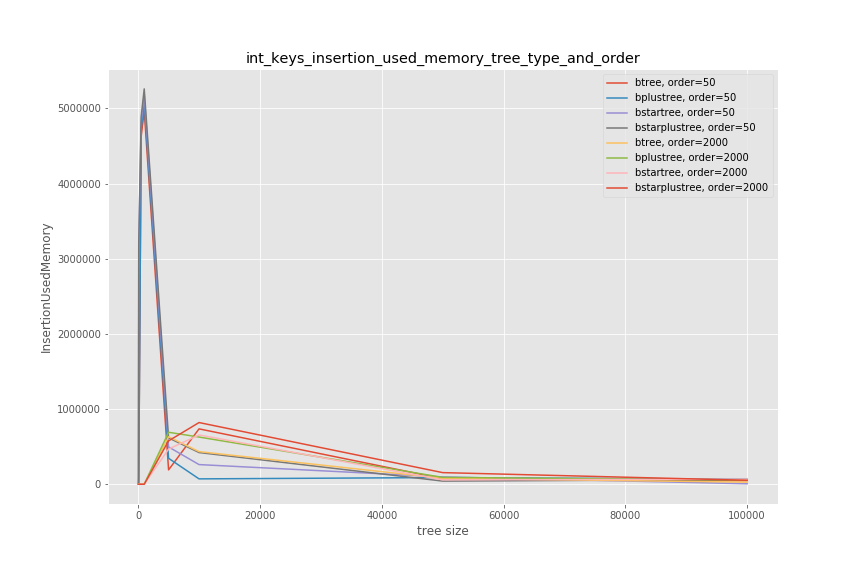
График зависимости времени выполнения вставки всех ключей в дерево от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000, приведён на рисунке 19.



*Рисунок 19. График зависимости времени выполнения вставки всех ключей в дерево от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*

Из данного графика видно, что B\*-дерево и B\*+-дерево при порядке дерева, равном 2000, дают, как и при индексации, лучший результат, чем B-дерево и B+-дерево. В то же время при порядке дерева, равном 50, всё ровно наоборот – лучший результат дают B-дерево и B+-дерево. Также видно, что график действительно похож на логарифмический.

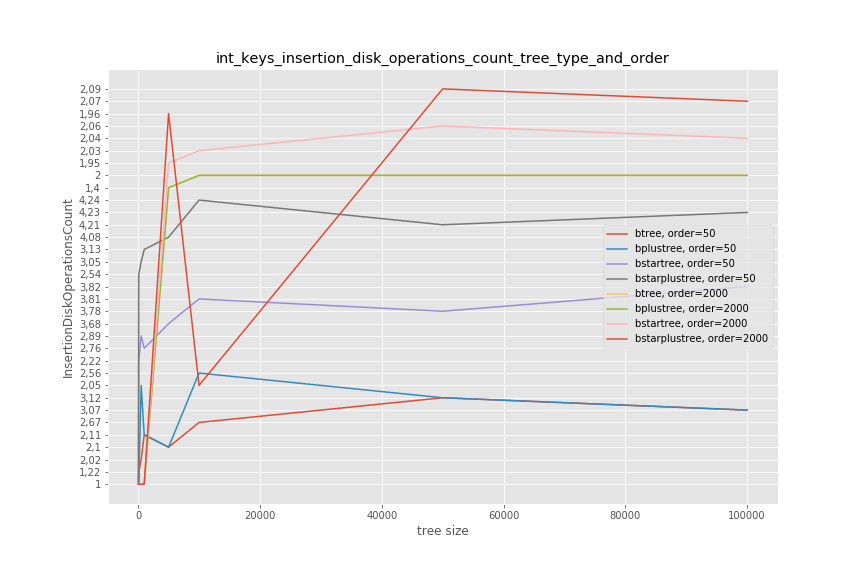
Теперь посмотрим на аналогичный график для объёма используемой памяти во время вставки ключа в дерево (рисунок 20).



*Рисунок 20. График зависимости среднего объёма используемой памяти при вставке ключа в дерево от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*

Как мы видим на данном графике, объём используемой памяти уменьшается с ростом количества ключей в дереве, что на данный момент не поддаётся аналитическому объяснению и противоречит теоретической оценке сложности.

Кроме того, посмотрим на график зависимости среднего числа дисковых операций (чтения и записи) от размера дерева (рисунок 21).

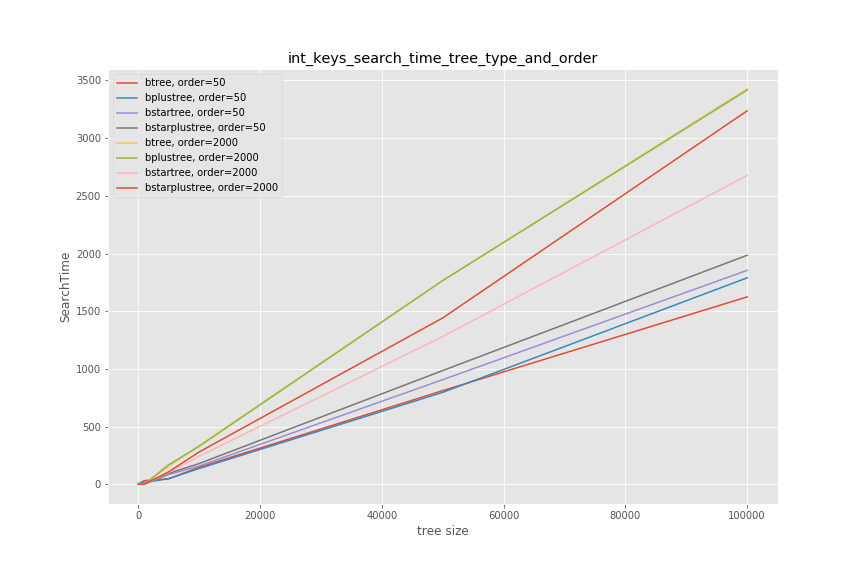


*Рисунок 21. График зависимости среднего количества дисковых операций (чтения и записи) при вставке ключа в дерево от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*

Из данного графика видно, что чёткая монотонная зависимость среднего количества дисковых операций от размера дерева не выявляется.

#### 5.1.3.2. Поиск в дереве

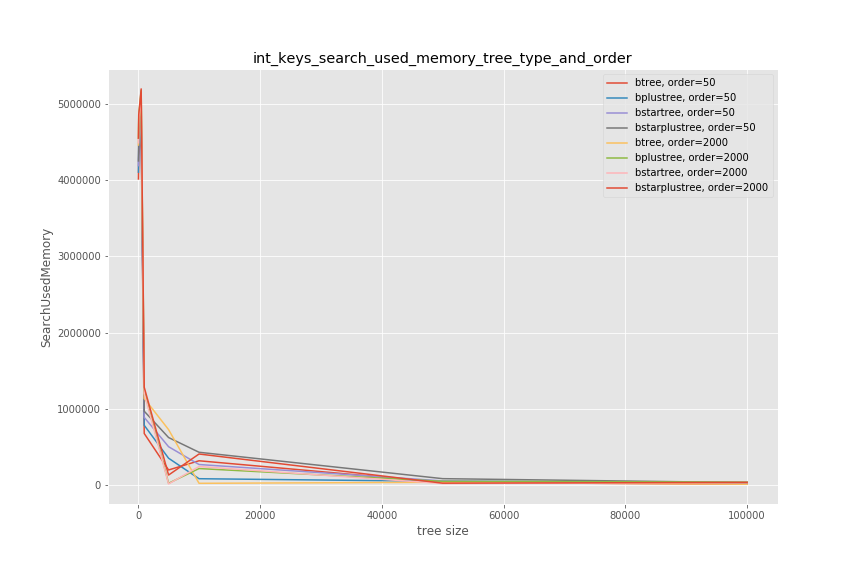
Изучим график зависимости времени выполнения поиска всех ключей в дереве от размера дерева (рисунок 22).



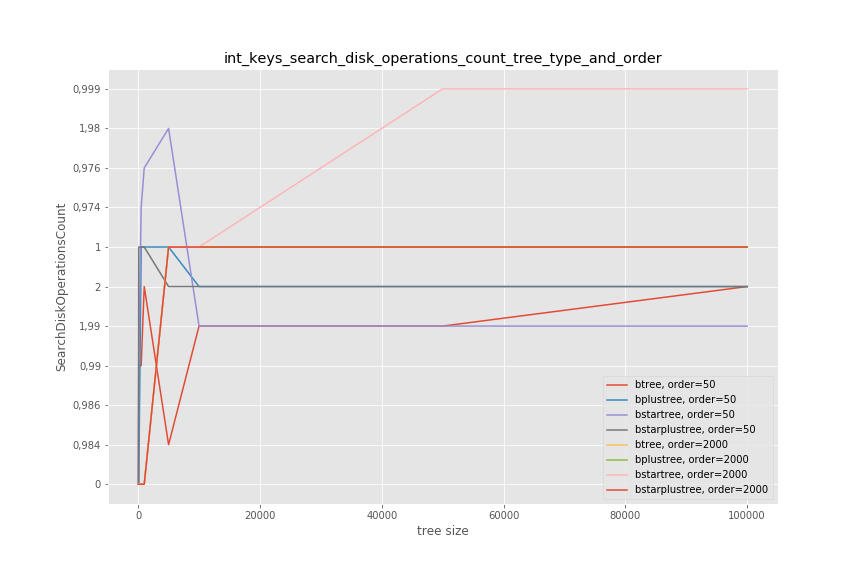
*Рисунок 22. График зависимости времени выполнения поиска всех ключей в дереве от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*

Из данного графика можно сделать выводы, аналогичные таковым для графика зависимости времени вставки от размера дерева (см. п. 4.1.3.1).

Аналогичные выводы делаются и из графиков зависимости объёма используемой памяти и числа дисковых операций от размера дерева при поиске (рисунки 23 – 24).



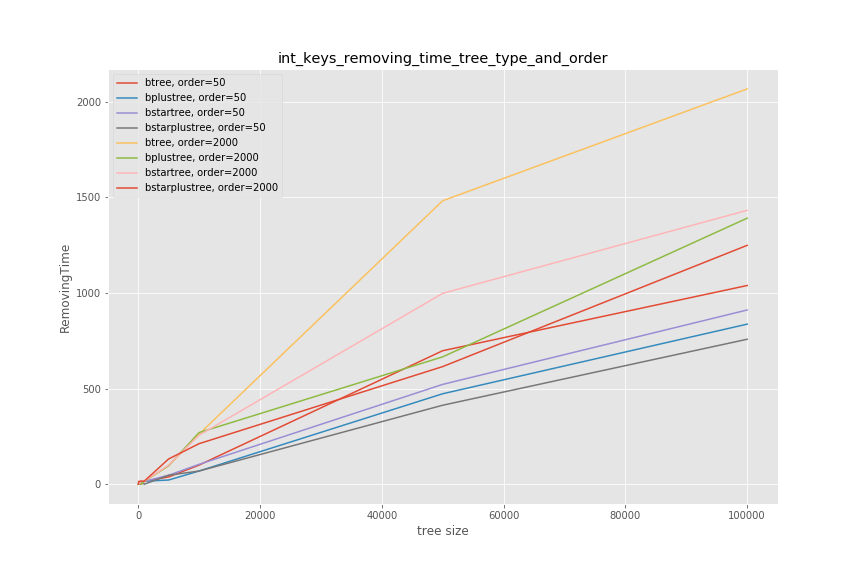
*Рисунок 23. График зависимости среднего объёма исп. памяти при поиске ключа в дереве от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*



*Рисунок 24. График зависимости среднего числа дисковых операций при поиске ключа в дереве от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*

#### 5.1.3.3. Удаление из дерева

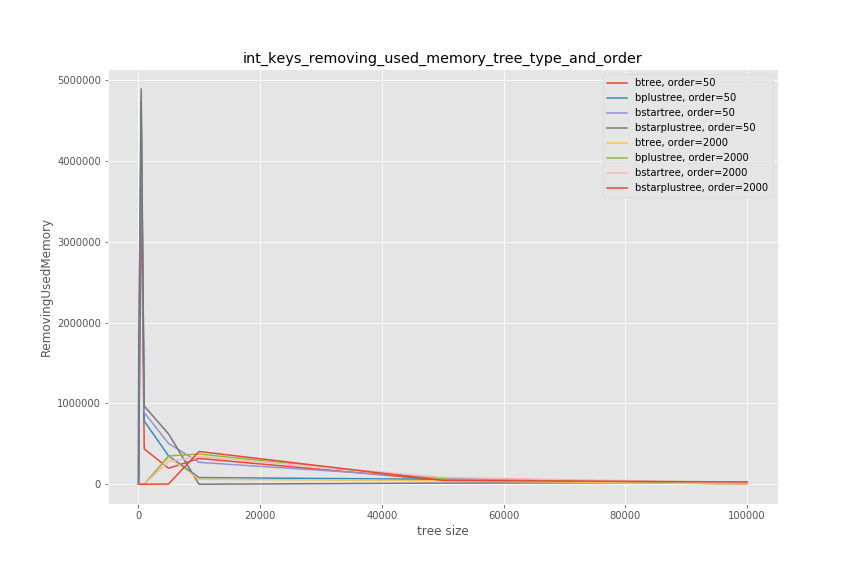
Рассмотрим график зависимости времени выполнения удаления всех ключей из дерева от размера дерева (рисунок 25).



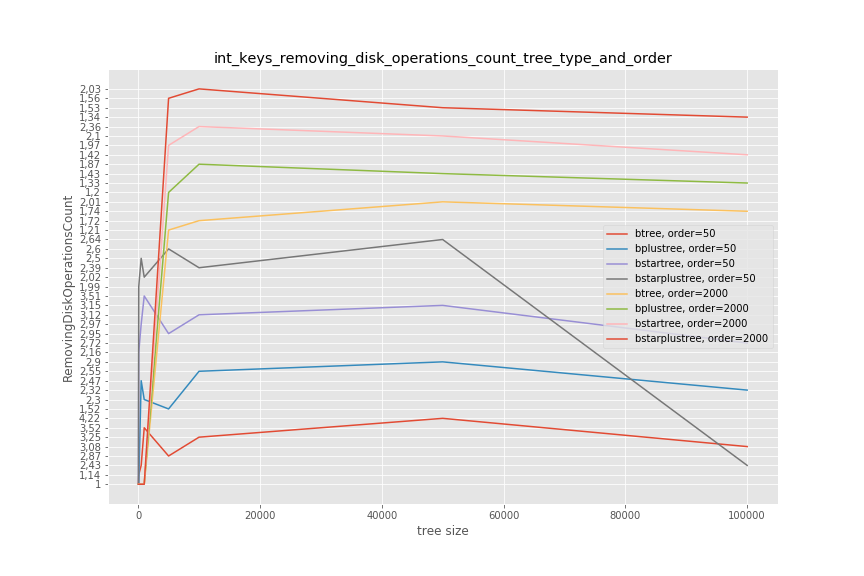
*Рисунок 25. График зависимости времени выполнения удаления всех ключей в дереве от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*

Как видно из данного графика, удаление из B+-дерева выполняется быстрее, чем удаление из B-дерева, что согласуется с выдвинутыми гипотезами. Аналогично, удаление из B\*+-дерева также выполняется быстрее, чем удаление из B\*-дерева, что, опять же, соответствует изначальным предположениям. Кроме того, B\*+-дерево порядка 50 даёт лучший результат, чем все остальные деревья на данном графике. Также отлично видно, что график действительно похож на логарифмический.

Из графиков зависимости объёма используемой памяти и числа дисковых операций от размера дерева при поиске (рисунки 26 – 27), делаются выводы, аналогичные таковым для операций вставки (п. 4.1.3.1) и поиска (п. 4.1.3.2) в дереве.



*Рисунок 26. График зависимости среднего объёма исп. памяти при удалении ключа в дереве от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*



*Рисунок 27. График зависимости среднего числа дисковых операций при удалении ключа в дереве от количества ключей (размера дерева) для значений порядка дерева, равных 50 и 2000*

## 5.2. Анализ результатов экспериментов и выводы

Согласно приведённым в подразделе 4.1 результатам экспериментов, могут быть сделаны следующие выводы.

Вставка в B\*-дерево и B\*+-дерево действительно происходит быстрее, чем вставка в B-дерево и B+-дерево соответственно, за счёт того, что процедура разбиения вершин выполняется реже. При этом B-дерево и B+-дерево между собой имеют приблизительно равное время вставки, как и B\*-дерево и B\*+-дерево между собой. При малых значениях (до 100) порядка дерева вместе с ростом порядка дерева уменьшается время вставки, при больших – увеличивается. Чёткой монотонной зависимости времени поиска от типа дерева либо его порядка не обнаружено. Зависимость времени выполнения любой операции от количества ключей в дереве, как и предполагалось, близка к логарифмической.

Использование памяти при индексации и при поиске по индексу увеличивается с ростом порядка дерева, так как приходится хранить в оперативной памяти большие по размеру узлы дерева. Число дисковых операций при индексации снижается вместе с увеличением порядка дерева.

Поведение параметров эффективности при индексации CSV-файла с реальными данными и при поиске по его индексу (а также, при вставке, поиске и удалении, за исключением времени выполнения удаления, целочисленных ключей), в основном, аналогично таковому при индексации сгенерированных CSV-файлов.

Удаление в B+-дереве, как и ожидалось, выполняется быстрее, чем в B-дереве, а удаление в B\*+-дереве – быстрее, чем в B\*-дереве, в связи с тем, что удаление происходит только из листовых узлов.

# Заключение

Выдвинуты рабочие гипотезы о различных параметрах эффективности (время выполнения, объём затрачиваемой оперативной памяти, число дисковых операций – чтения и записи) операций над четырьмя типами сильно ветвящихся деревьев (B-дерево, B+-дерево, B\*-дерево B\*+-дерево). Разработана новая модификация B-дерева – B\*+-дерево, дающая хорошие результаты относительно времени удаления ключей из дерева (и лучшие, чем в B-дереве, результаты относительно времени вставки ключей в дерево).

Реализованы в виде библиотеки все четыре рассматриваемых типа сильно ветвящихся деревьев с операциями вставки, поиска и удаления на языке C++. Разработан программный инструмент на языке C++ для проведения экспериментов с индексацией в дереве, поиском по индексу и стандартными операциями (вставка, поиск и удаление).

Построена схема экспериментов и проведены эксперименты, получены их результаты, из которых при помощи инструментария Python и Jupyter Notebook построены графики, которые приведены, проанализированы и интерпретированы в настоящей работе. Часть выдвинутых гипотез подтверждена, обобщение дано в подразделе 4.2.

Направлениями дальнейшего развития работы могут быть добавление новых параметров для измерения их влияния на параметры эффективности операций над деревом в разных экспериментах, проведение экспериментов с другими комбинациями параметров, а также интерпретация тех случаев, когда результаты не совпали с выдвинутыми гипотезами.

Результаты данной работы могут быть использованы учащимися и исследователями в области алгоритмов и структур данных, а также СУБД, для проведения новых исследований в области, и для улучшения производительности СУБД.

# Список использованных источников

1. B\*-tree [Электронный ресурс] // NIST Dictionary of Algorithms and Data Structures. Режим доступа: <https://xlinux.nist.gov/dads/HTML/bstartree.html>, свободный. (дата обращения: 23.12.2017)
2. B-Star Trees [Электронный ресурс] // College of Engineering, Computer Science & Construction Management, Chico, California State University. Режим доступа: <http://www.ecst.csuchico.edu/~mjstapleton/courses/Fall2007CSCI311/ProgTwo_Bstar.htm>, свободный. (дата обращения: 15.01.2018)
3. Kerttu Pollari-Malmi. B+-trees // [Электронный ресурс]: Computer Science | University of Helsinki. Режим доступа: <https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf>, свободный. (дата обращения: 07.12.2017)
4. Manuika, J. Big data: The next frontier for innovation, competition, and productivity / Manuika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Hung Byers, A. // [Электронный ресурс]: McKinsey Global Institute, McKinsey & Company, May 2011. Режим доступа: <https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/Big%20data%20The%20next%20frontier%20for%20innovation/MGI_big_data_exec_summary.ashx>, свободный. (дата обращения: 20.03.2018)
5. MySQL :: MySQL 8.0 Reference Manual :: 8.3.9 Comparison of B-Tree and Hash Indexes [Электронный ресурс] // MySQL :: Developer Zone. Режим доступа: <https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html>, свободный. (дата обращения: 20.03.2018)
6. PostgreSQL: Documentation: 9.3: CREATE INDEX [Электронный ресурс] // PostgreSQL. Режим доступа: <https://www.postgresql.org/docs/9.3/static/sql-createindex.html>, свободный. (дата обращения: 20.03.2018)
7. Process Mining [Электронный ресурс]. Режим доступа: <http://www.processmining.org/>, свободный. (дата обращения: 10.06.2017)
8. Д. Кнут. Искусство программирования. Том 3. 2-е изд. – М.: ИД «Вильямс». – 2017. – 824 с.
9. Т. Кормен. Алгоритмы: построение и анализ. 3-е изд. / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. — М.: ИД «Вильямс». — 2013. — 1324 с.

1. <http://www.processmining.org/> [↑](#footnote-ref-1)