План проведения предварительного исследования для курсовой работы на тему "Исследование эффективности сильно ветвящихся деревьев в задаче индексирования структурированных данных"

Антон Ригин, ПИ, 3 курс, группа БПИ153

1 Обзор источников

В качестве источников рассмотрены русскоязычные и англоязычные источники, представляющие собой как учебные пособия, так и статьи в научных изданиях.

Из учебных пособий можно отметить «Алгоритмы: построение и анализ» (3-е изд.) Т. Кормена, в главе 18 которого описаны B-деревья, их структура, основные операции с ними (поиск, вставка и удаление), а также их сложность, и «Искусство программирования. Том 3. Сортировка и поиск» (3-е изд.) Д.Э. Кнута, в разделе 6.2.4 которого также описаны B-деревья, и, кроме того, некоторые их модификации, включая B^+-деревья.

Из научных статей можно выделить “Organization and Maintenance of Large Ordered Indexes”, опубликованную Р. Бэйером и Э. МакКрейтом в научном журнале “Acta Informatica”в 1972 году и давшую начало использованию B-деревьев в информатике, “An Astounding Example of Efficiency with B-Trees”, опубликованную Дж. Уошэмом на интернет-портале “Startup Nextdoor”и описывающую пример эффективного применения B-деревьев для индексации файла, а также статью “B^+-trees”, опубликованную К. Поларри-Малми на сайте Департамента компьютерных наук Хельсинского университета.

Кроме того, отдельно стоит упомянуть формальное определение B^*-дерева, представленное в NIST Dictionary of Algorithms and Data Structures.

2 B-деревья

B-дерево — сильно ветвящееся дерево, то есть дерево, узлы которого представляют собой списки ключей и списки указателей на узлы-потомки [1]. Для типа, к которому относятся ключи дерева, должны быть определены операции сравнения.

B-дерево построено так, что если какой-то узел содержит n ключей, то у данного узла $n+1$ потомков, и для любого i, такого, что $1 \leq i \leq n+1$, верно, что все ключи в i-м потомке данного узла не меньше, чем i-й ключ данного узла, и не больше, чем $i+1$-й ключ данного узла. Для 1-го потомка данного узла верно, что все ключи в нём не больше 1-го узла данного потомка, а для $n+1$-го потомка данного узла верно, что все ключи в нём не меньше n-го ключа данного узла [1].

B-дерево задаётся параметром, называемом степенью. Данный параметр не может быть меньше 2. Обозначим его t. Если B-дерево задано со степенью t, то каждый узел, кроме корневого, будет содержать не менее $t-1$ ключей, а корневой узел — не менее 1 ключа (кроме случая пустого дерева, в таком случае, корневой узел будет содержать 0 ключей), и каждый узел будет содержать не более $2t-1$ ключей (и, соответственно, иметь не более $2t$ потомков) [1].

B-дерево является сбалансированным деревом, поэтому его высота равняется $O(\lg n)$, где n — число ключей в дереве. Очевидно, что основанием логарифма в данном случае будет степень дерева t, поэтому правильнее сказать, что высота B-дерева равна $O(\log_t n)$, где t — степень дерева, а n — число ключей в дереве [1].

Для B-дерева определены основные операции: поиск, вставка и удаление. Под понятием эффективности данных операций мы объединим три параметра: сложность (по вычислениям), объём необходимой памяти, число дисковых операций.

Под дисковыми операциями мы будем иметь в виду чтение узла из постоянной памяти и запись узла в постоянную память. Здесь отдельно стоит отметить, что B-дерево, как правило, не хранится целиком в оперативной памяти, а располагается в постоянной памяти. Это связано с тем, что B-дерево обычно используется для хранения данных больших размеров, например, индексов в базах данных. Структура B-дерева специально предназначена для такого хранения. В оперативную память подгружаются при необходимости лишь отдельные узлы.

2.1 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 [1].

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 степени дерева t. Таким образом, операция поиска в B-дереве имеет вычислительную сложность $O(t \log_t n)$, где t — степень дерева, а n — число ключей в дереве [1].

При реализации функции поиска в виде рекурсии потребуется $O(t \log_t n)$ памяти, где t — степень дерева, а n — 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 $O(t)$ памяти, где t — степень дерева.

Кроме того, поскольку, как уже сказано, число пройденных вершин не будет превышать высоту дерева, число дисковых операций будет равно $O(\log_t n)$.

2.2 Вставка ключа 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 [1].

2.2.1 Разбиение заполненного узла

В B-дереве ни один узел не может иметь вершин больше, чем $2t - 1$, где t — степень дерева [1]. 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 ($2t - 1$ 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 [1].

2.2.2 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 степени дерева t. Таким образом, вычислительная сложность разбиения равна $O(t)$, где t — степень 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 $O(t)$, где t — степень дерева, а число дисковых операций — $O(1)$.

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 степени дерева t. Соответственно, вычислительная сложность операции вставки равна $O(t \log_t n)$, где t — степень дерева, а n — число ключей в дереве.

При реализации функции вставки в виде рекурсии потребуется $O(t \log_t n)$ памяти, где t — степень дерева, а n — 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 ре-

ализовать данную функцию в виде цикла, то потребуется в один момент хранить лишь один узел. Тогда потребуется $O(t)$ памяти, где t — степень дерева.

Кроме того, поскольку, как уже сказано, число пройденных вершин не будет превышать высоту дерева, число дисковых операций будет равно $O(\log_t n)$.

2.3 Удаление ключа 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 [1].

Если же ключ находится во внутреннем узле дерева, то возможно несколько вариантов действий. Если дочерний узел, предшествующий удаляемому, содержит не менее t ключей (где t — степень 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 (наименьшьим ключом в поддереве, корнем которого является дочерний узел, следующий за удаляемым ключом). Если же оба этих дочерних узла содержат по $t - 1$ 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 [1].

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 $t - 1$ 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 $t-1$ ключей, то выполняется операция объединения с любым из его непосредственных соседей. При этом бывший ключ-разделитель становится медианой нового объединённого узла [1].

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

2.3.1 Эффективность удаления

Вычислительная сложность операции удаления зависит от количества пройденных вершин, не превышающего высоты дерева, и степени дерева, и равняется $O(t \log_t n)$, где t — степень дерева, а n — число ключей в дереве [1].

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 степени, и, соответственно, будет равен $O(t \log_t n)$, где t — степень дерева, а n — число ключей в дереве.

Количество дисковых операций также зависит от высоты дерева, и, соответственно, равняется $O(\log_t n)$, где t — степень дерева, а n — число ключей в дереве [1].

2.4 Модификации B-дерева

2.4.1 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 [5].

В B^+-дереве максимальным числом узлов в каждой вершине является $2t$, а не $2t-1$, где t — степень 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 каждой вершине будет являться t, а не $t-1$, где t — степень дерева [5].

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-машрутизатора в этом случае является выбор в качестве такового последнего ключа из левой части разбиваемого листа [5].

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 $O(1)$ 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 B-дерева. 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.

2.4.2 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. 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 [6].

Операция поиска 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 степени дерева t 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.

3 Проведение экспериментов и используемые средства

В ходе выполнения данной работы будут проведены эксперименты, оценивающие функциональную зависимость эффективности операций поиска, вставки и удаления в B-дереве и его модификациях (B^+-дереве и B^*-дереве) от основного параметра построения дерева — его степени t, используемого типа дерева (самого B-дерева, B^+-дерева или B^*-дерева) и количества хранимых в дереве ключей n.

Для этого будет разработан специальный программный инструмент на языке C++, с использованием библиотеки Qt (для отображения графического интерфейса пользователя и иного использования её графических возможностей, в том числе, отрисовки дерева), который будет позволять автоматизированно проводить эксперименты по заданной схеме, и выводить результаты в удобном для исследователя виде.

Для проведения самих экспериментов предварительно будут проведены формальное построение теоретической модели и выработка теоретических гипотез с доказательствами, и разработана схема экспериментов, содержащая набор параметров (степень дерева, тип дерева и количество хранимых в дереве ключей) для каждого эксперимента.

4 Список используемых источников

[1] Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы: построение и анализ. 3-е изд. — М.: ИД "Вильямс". — 2013. — 1324 с.

[2] Кнут Д.Э. Искусство программирования. Том 3. Сортировка и поиск. 2-е изд. — М.: ИД "Вильямс". — 2002. — 800 с.

[3] Bayer R., McCreight E. Organization and Maintenance of Large Ordered Indexes // Acta Informatica. — 1972, V. 1. — P. 173-189

[4] Washam J. An Astounding Example of Efficiency with B-Trees [Electronic Source] // Startup Nextdoor. — URL: https://startupnextdoor.com/an-astounding-example-of-efficiency-with-b-trees/ [Date: 2017/12/07]

[5] Kerttu Pollari-Malmi. B^+-trees [Electronic Source] — URL: https://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf [Date: 2017/12/07]

[6] B^*-tree [Electronic Source] // NIST Dictionary of Algorithms and Data Structures — URL: https://xlinux.nist.gov/dads/HTML/bstartree.html [Date: 2017/12/23]