Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

ИСПОЛЬЗОВАНИЕ ДЕРЕВЬЕВ ПОИСКА ДЛЯ РАЗЛИЧНЫХ ТИПОВ ДОКУМЕНТОВ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Тарасовой Алисы Андреевны

Научный руководитель		
доцент, к. ф.-м. н.	____________	А. С. Иванова
Заведующий кафедрой		
к. ф.-м. н.	____________	С. В. Миронов

Саратов 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С развитием компьютерной техники и внедрением ее во все сферы повседневной жизни стало удобно хранить данные в электронной форме. На сегодняшний день накоплено огромное количество информации, поэтому вполне естественно, что возникает необходимость управления ими, например, создание, хранение, удаление и поиск данных.

В условиях современного ритма жизни логично задумываться над скоростью выполнения поставленных задач выбранными методами. В связи с этим, актуальной является проблема поиска с использованием вычислительных машин. Данные размещаются в различных структурах данных, таких как бинарные деревья, для повышения эффективности поиска.

Работа содержит введение, 3 главы, заключение, 24 рисунка, 10 использованных источников и 1 приложение с программным кодом.

Работа включает в себя три главы: исследование самобалансирующихся деревьев поиска, сравнение красно-черного дерева со сбалансированным AVL-деревом и экспериментальное сравнение эффективности деревьев.

— В первой главе приведены основные теоретические понятия о самобалансирующихся деревьях поиска, алгоритмы вставки и удаления элементов для двух видов деревьев поиска.
— Во второй главе представлены теоретические границы, проведено сравнение показателей эффективности на основе оценок максимальной высоты деревьев, операции вставки, удаления и поиска, потребляемой памяти в зависимости от количества его элементов.
— В третьей главе представлено экспериментальное доказательство теоретических границ, проведено сравнение показателей эффективности на основе операции поиска и исследована зависимость высоты дерева от количества его элементов.

Целью данной работы является исследование материала о самобалансирующихся деревьях поиска и сравнение скорости выполнения поставленных задач на двух видах деревьев поиска.

Для достижения поставленной цели необходимо решить следующие задачи:

— реализовать AVL-деревья и красно-черные деревья на языке C++;
— протестировать реализации на различных типах данных;

– протестировать программу на различных форматах документов.

1 Основное содержание работы

1.1 AVL-деревья

AVL-дерево — это двоичное дерево поиска, ключи которого удовлетворяют стандартному свойству: ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла. [1]

Особенностью AVL-дерева является то, что оно сбалансированное по высоте дерево поиска, то есть для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу. [2]

1.1.1 Балансировка

Балансировка вершины относительно AVL-деревьев — это операция, которая в случае, если разница высот левого и правого поддеревьев $= 2$, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится $\leqslant 1$, иначе ничего не меняет. [3]

Данный результат реализуется вращениями поддерева данной вершины.

1.1.2 Алгоритм добавления вершины

Показатель сбалансированности в дальнейшем будет интерпретироваться, как разность между высотой левого и правого поддерева. Непосредственно при вставке элемента листу присваивается нулевой баланс. [4]

Процесс включения вершины состоит из трех частей:

1. Обход пути поиска, пока не убедимся, что ключа в дереве нет.
2. Включение новой вершины в дерево и определения результирующих показателей балансировки.
3. Возвращение по пути поиска и проверки в каждой вершине показателя сбалансированности. Если необходимо — балансировка.

Будет возвращаться в качестве результата функции ответ - уменьшилась высота дерева или нет. [5] Предположим, что процесс из левой ветви возвращается к родителю (рекурсия идет назад), тогда включение вершины в левое поддерево приведет к: $\{h_l$ — высота левого поддерева, h_r — высота правого поддерева$\}$

1. $h_l < h_r$: выравниваются $h_l = h_r$. Ничего делать не нужно.
2. $h_l = h_r$: теперь левое поддерево будет больше на единицу, но балансировка пока не требуется.

3. $h_l > h_r$: теперь $h_l{}^{-}h_r = 2$, следовательно, требуется балансировка.

В третьей ситуации требуется определить балансировку левого поддерева. Если левое поддерево этой вершины выше правого, то требуется большое правое вращение, иначе хватит малого правого. Симметричные рассуждения можно привести и для включения в правое поддерево.

1.1.3 Алгоритм удаления вершины

Был реализован рекурсивный алгоритм удаления. [6]

1. Если вершина — лист, то удалим её и вызовем балансировку всех её предков в порядке от родителя к корню.
2. Иначе найдём самую близкую по значению вершину в поддереве наибольшей высоты и переместим её на место удаляемой вершины, при этом вызвав процедуру её удаления.

1.2 Красно-черные деревья

Красно-чёрное дерево — это одно из самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и быстро выполняющее основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный». [7]

Красно-черные деревья представляют собой двоичные деревья, обладающие следующей четверкой свойств:

1. Каждый узел окрашен либо в черный, либо в красный цвет;
2. Листьями объявляются виртуальные NIL-узлы (обычно листьями называют их предков). Листья окрашены в черный цвет;
3. Если узел красный, то оба его потомка черные;
4. На всех ветвях дерева, ведущих от его корня к листьям, число черных узлов одинаково.

Красно-черные деревья не являются идеально сбалансированными, однако соблюдение четырех свойств гарантирует, что количество узлов между корнем и его самым ближним и дальним листьями отличается не более чем в два раза. [7]

Количество черных узлов в пути между узлом V и ближайшим листом называется черной высотой дерева — $bh(V)$. Перечисленные свойства гарантируют, что самая длинная ветвь от корня к листу не более чем вдвое длиннее любой другой ветви от корня к листу. [8]

Очевидно, для соблюдения свойства 4 при вставке, узлы дерева периодически необходимо перекрашивать в красный цвет. Из свойства 4 и определения черной высоты дерева очевидно, но может быть доказано по индукции, что дерево с корнем V содержит не менее $2^{bh(V)} - 1$ внутренних узлов, т.е. $n \geqslant 2^{bh(V)} - 1 \geqslant \log_2 n \geqslant bh(V)$.

Согласно правилу 3, если $h(V)$ — высота дерева, то $bh(V)$ не может быть меньше $h(V)/2$, значит $\log_2 n \geqslant bh(V) \geqslant h(V)/2$. Значит, можно утверждать, что $h = O(\log n)$, то есть высота дерева (а значит, и время поиска) имеет логарифмическую асимптотическую оценку в зависимости от количества узлов дерева. [9]

1.2.1 Балансировка

При любых операциях с деревом должны соблюдаться описанные выше правила, для этого после каждого изменения (вставки или удаления узла) выполняется балансировка. При балансировке красно-черных деревьев применяют малые левые и правые повороты дерева, изменяющие высоту правого и левого поддерева узла P.

При балансировке будет выполнено $O(h(V))$ операций, т. е. трудоемкость добавления и удаления элемента оценивается сверху логарифмической функцией. [10]

1.2.2 Алгоритм добавления вершины

Каждый элемент вставляется вместо листа, поэтому для выбора места вставки идем от корня до тех пор, пока указатель на следующего сына не станет NULL. [2]

Вставляем вместо него новый элемент с двумя листами потомками и красным деревом, а далее выполняем балансировку:

1. Если отец нового элемента черный, то завершаем алгоритм.
2. Если отец нового элемента красный, то достаточно рассмотреть два случая:
   а) «Дядя» этого узла тоже красный. Перекрашиваем «отца» и «дядю» в черный цвет, а «деда» – в красный. Далее рекурсивно пытаемся восстановить свойства дерева, продвигаясь к предкам.

б) «Дядя» – черный. Если добавляемый узел X был правым потомком «отца» A, то необходимо сначала выполнить левое вращение, которое сделает «отца» A левым потомком X. Выполняем правый поворот и перекрашиваем A и B. Если «дядя» левый, то порядок действий симметричен описанному.

Каждая корректировка, производимая при вставке узла, заставляет нас подняться в дереве на один шаг. В этом случае до остановки алгоритма будет сделано 1 вращение (2, если узел был правым потомком). Аналогичен и метод удаления. [3]

1.2.3 Алгоритм удаления вершины

В зависимости от количества «детей» вершины, при ее удалении могут возникнуть три случая:

1. Если у вершины нет детей, то изменяем указатель на нее у родителя на фиктивный лист.
2. Если у нее только один ребенок, то делаем у родителя ссылку на него вместо вершины.
3. Если же имеются два ребенка, то находим вершину со следующим значением ключа. У такой вершины нет левого ребенка. Удаляем уже эту вершину, описанным во втором пункте, способом, скопировав ее ключ в изначальную вершину.

При удалении красной вершины свойства дерева не нарушается. [7]

Восстановление свойств красно-черного дерева потребуется только при удалении черной вершины:

Удаление черной вершины с потомком. Единственным потомком черной вершины может быть только красная вершина. Иначе нарушилось бы свойство постоянства черной глубины для потомка и его соседней фиктивной вершины. Для балансировки выполним следующие действия: в черную вершину заносим данные красной, затем удаляем красную.

Удаление черной вершины без потомков. Для этого удалим черную вершину. Лист на месте удаленной вершины обозначим X. Путь в X имеет меньшее количество черных вершин, чем в других вершинах. Теперь X будем называть дважды черным.

1.3 Сравнение красно-черного дерева с сбалансированным AVL-деревом

У AVL и цветных деревьев есть как общие черты, так и отличия. Однако, и у тех, и у других все базовые операции над бинарными деревьями имеют сложность $O(\log N)$.

1.3.1 Высота дерева

Пусть высота дерева h, минимальное количество вершин N. Тогда:

1. для AVL-дерева $N(h) = N(h-1) + N(h-2) + 1$. Поскольку $N(0) = 0$, $N(1) = 1$, $N(h)$ растёт как последовательность Фибоначчи, следовательно $N(h) = \Theta(\lambda^h)$, где $\lambda = (\sqrt{5}+1)/2 \approx 1,62$.
2. для красно-чёрного дерева $N(h) \geqslant 2^{(h-1)/2} = \Theta(\sqrt{2^h})$. Следовательно, при том же количестве листьев красно-чёрное дерево может быть выше AVL-дерева, но не более чем в $\log_{\sqrt{2}} \lambda \approx 1,388$ раз, поэтому для выполнения поиска по красно-черному дереву, возможно, требуется больше времени.

1.3.2 Поиск

Красно-чёрное дерево в худшем случае выше AVL-дерева, но не более чем в $\log_{\sqrt{2}} \lambda \approx 1,388$ раз, поэтому для выполнения поиска по красно-черному дереву требуется больше времени, т.е. поиск в нём медленнее, но проигрыш по времени не превышает $\approx 39\%$.

1.3.3 Вставка

При вставке красно-черные деревья выполняют балансировку за $O(1)$, AVL же за $O(\log N)$. Однако, из-за большей высоты красно-чёрного дерева, вставка может занимать больше времени. Тем не менее, тесты показывают, что красно-чёрные деревья производительнее при больших объёмах памяти.

1.3.4 Удаление

Касательно удаления, выигрывают красно-черные, так как им потребуется $O(1)$ (максимум 3 поворота), тогда как для AVL может потребоваться $O(\log N)$ — числа поворотов до глубины дерева (до корня). Поэтому удаление из красно-чёрного дерева быстрее, чем из AVL-дерева.

1.3.5 Память

AVL-дерево должно хранить высоту (целое число от -1 до +1, т. е. для кодирования нужно 2 бита) в каждом узле, в то время как в узле красно-черного дерева хранится только 1 бит, определяющий цвет узла.

Теоретически считается, что красно-чёрное дерево требует меньшего объёма памяти для хранения отдельного узла, чем AVL-дерево, т.к. для представления цвета достаточно всего одного бита. Но на практике это преимущество реализовать без потерь на дополнительные операции доступа к отдельным битам весьма сложно. Даже один из вариантов реализации красно-чёрного дерева – когда "красные"узлы обозначаются нечётными ключами, а "чёрные"узлы – чётными (или наоборот), не всегда пригоден на практике, т.к. решаемая задача может не допускать такого разделения узлов.

Поэтому, если учитывать, что в современных вычислительных системах память выделяется кратно байтам, то деревья абсолютно одинаковы.

1.4 Экспериментальное сравнение эффективности деревьев

Экспериментально сравнили эффективность AVL и красно-черного деревьев. Реализация проводилась на `Microsoft Visual C++ 2010 Express` под 32х разрядной операционной системой `Windows 7 PC`. Входные данные вводились с помощью генератора в файл `test.txt` и `test.xml`, а выходные данные (координаты) выводились в папку проекта с названиями `dataRBHeight.csv`, `dataRBSearch.csv`, `dataAVLHeight.csv`, `dataAVLSearch.csv` в зависимости от поставленного аргумента в `main.cpp`.

Рассмотрен рост максимальной высоты дерева в зависимости от количества элементов. Для этого был проведен несложный вычислительный эксперимент: генерировался массив из случайно расположенных чисел, эти числа последовательно вставлялись в изначально пустое AVL-дерево и красно-черное дерево, далее измерялась высота деревьев после каждой вставки. Полученные результаты были усреднены по 100 расчетам. В данной работе были приложены графики, на которых показана зависимость от n максимальной высоты деревьев и видно, что максимальная высота красно-черного дерева растет быстрее, чем у AVL-дерева.

Для экспериментальной проверки скорости операции поиска рассматривали два тестовых файла с различными расширениями: `test.txt` и `test.xml`.

Был проведен следующий эксперимент: генерировались два тестовых файла с различными расширениями, далее данные расформировывались в AVL-дерево и красно-черное дерево, после чего измерялось время поиска в двух видах деревьев после каждой вставки из документов этих форматов. Полученные результаты были усреднены по 100 расчетам.

В качестве результата в работе приложены графики, на которых видно, что красно-черные деревья справляются с этой задачей значительно быстрее AVL-деревьев, и на больших значениях числа элементов цветные деревья сильно выигравают у AVL по скорости.

Также в работе приложены рисунки, на которых видно, что поиск данных в xml-документах производится гораздо дольше, чем в .txt. Однако, красно-черные деревья справляются с этой задачей значительно быстрее AVL-деревьев, и на больших значениях числа элементов цветные деревья сильно выигравают у AVL по скорости так же, как и в txt-формате.

Полный код программы приведен в приложении данной работы.

ЗАКЛЮЧЕНИЕ

В общем случае сравнение двух рассмотренных типов деревьев провести затруднительно, поскольку для разных задач и наборов данных лучший тип дерева может быть разным, для каждого определенного случая нужно подбирать оптимальный вид сбалансированного дерева.

В работе были исследованы общие свойства красно-черных и AVL-деревьев, различные виды балансировки деревьев, а так же алгоритмы удаления и вставки элементов. Проведено сравнение показателей эффективности на основе операции поиска и исследована зависимость высоты дерева от количества его элементов.

Самое главное преимущество красно-черных деревьев в том, что при вставке выполняется не более $O(1)$ вращений. Процедуру балансировки практически всегда можно выполнять параллельно с процедурами поиска, так как алгоритм поиска не зависит от атрибута цвета узлов.

Сбалансированность этих деревьев хуже, чем у AVL, но работа по поддержанию сбалансированности в красно-чёрных деревьях эффективнее. Для балансировки красно-чёрного дерева производится минимальная работа по сравнению с AVL-деревьями.

Несмотря на преимущества, предоставляемые AVL-деревьями при выполнении поиска по дереву, их использование затрудняется необходимостью балансировки после выполнения операций добавления и удаления узлов. Алгоритм балансировки представляет основную проблему, так как его неудачная реализация может негативно влиять на производительность программы, использующей данный тип дерева.

Цветные деревья используют всего 1 бит дополнительной памяти для хранения цвета вершины. Но в современных вычислительных системах память выделяется кратно байтам, поэтому это не является преимуществом относительно AVL-дерева, которое хранит 2 бита. Красно-чёрные деревья являются наиболее активно используемыми на практике самобалансирующимися деревьями поиска. В частности, ассоциативные контейнеры библиотеки `STL(map, set, multiset, multimap)`, `TreeMap` в `Java` реализованы на основе красно-чёрных деревьев.

По сложности реализации самые простые – AVL-деревья, сложнее – красно-черные деревья, поскольку приходится рассматривать много нетривиальных случаев при вставке и удалении узла. Восстановление свойств как AVL-дерева, так и красно-черного дерева после вставки требует не более двух поворотов. Однако, после удаления узла из красно-черного дерева потребуется не более трех поворотов, а в AVL-дереве после удаления узла может потребоваться количество поворотов равное высоте дерева (от листа до корня), поэтому операция удаления в красно-черном дереве эффективнее, в связи с этим они более распространены.

Результатами исследования являются следующие выводы:

1. Максимальная высота красно-черного дерева растет быстрее, чем у AVL дерева, что приводит к увеличению времени работы алгоритма поиска в красно-черном дереве, однако на практике AVL-дерево работает дольше, чем красно-черное.
2. AVL-деревья лучше использовать, когда необходим быстрый поиск элемента в фиксированных данных. Если же данные динамические, т.е. много операций вставки и удаления, то лучше использовать красно-черные деревья.
3. Поиск данных в xml-документах производится гораздо дольше, чем в .txt, однако красно-черные справляются с этой задачей быстрее, чем AVL-деревья.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Ершов, Н.* Разработка. АВЛ-деревья. [Электронный ресурс] / Н. Ершов. — URL: https://habrahabr.ru/post/150732/ (Дата обращения 06.05.2016). Загл. с экр. Яз. рус.

2 *Матьяш, В. А.* // Структуры данных и алгоритмы обработки. Учебное пособие. — Апатиты: 2000. — P. 80.

3 *Вирт, Н.* // Алгоритмы и структуры данных. — М.: Мир, 1989. — P. 272—286.

4 *Адельсон-Вельский, Г. М.* // Один алгоритм организации информации. — Vol. Т. 146, № 2. — 1962. — P. С. 263—266.

5 *Скиена, С.* // Алгоритмы. Руководство по разработке. — 2-е изд.: Пер. с англ. - СПб: БХВ-Петербург, 2011. — P. 720.

6 *Ахо, А.* // Структуры данных и алгоритмы. — М.: Вильямс, 2-е издание, 2000. — P. 384.

7 Структуры данных: Красно-черные деревья [Электронный ресурс]. — URL: http://algolist.manual.ru/ds/rbtree.php (Дата обращения 06.05.2016). Загл. с экр. Яз. рус.

8 *Кормен, Т.* // Алгоритмы: построение и анализ. — Вильямс, 2-е издание, 2005. — P. 1296.

9 *Ниман, Т.* Сортировка и поиск: Рецептурный справочник. [Электронный ресурс] / Т. Ниман. — URL: http://cs.mipt.ru/docs/comp/rus/programming/algorithms/nimansortpoisk/main.pdf (Дата обращения 06.05.2016). Загл. с экр. Яз. рус.

10 *Кнут, Д.* // Искусство программирования. — М.: Мир, 1989. — P. раздел 6.2.3.