МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

ДЕРЕВЬЯ: ОЦЕНКА ЭФФЕКТИВНОСТИ АЛГОРИТМОВ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Барбашина Алексея Андреевича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

ВВЕДЕНИЕ

Для эффективного поиска данных достаточно часто используется такая структура, как дерево поиска . Под деревом поиска понимается специальным образом построенное дерево — слева от корня расположены элементы, меньшие по значению, справа — большие. В случае правильно построенного дерева его высота пропорциональна $\log N$, где N — это количество элементов в дереве. Но можно подобрать данные так, что высота дерева увеличится до N и поиск становится неэффективным.

Поэтому используются самобалансирующиеся деревья поиска, когда к построению дерева поиска добавляются дополнительные условия, что позволяет создать дерево минимально возможной высоты. Наиболее известные самобалансирующиеся бинарные деревья поиска (каждый узел содержит не более двух детей) — это красно-чёрное дерево и AVL–дерево.

Самобалансирующееся дерево может быть и не бинарным. Примером такого дерева является B–дерево. Каждый узел дерева содержит не менее $t - 1$ и не более $2t - 1$ элементов и не более $2t$ детей. t — это *степень узла*, определяется пользователем. Обычно B–дерево используется для поиска во внешней памяти, так как позволяет хранить не значения, а указатели на данные. Обычно B–дерево применяется для организации индексов во многих СУБД. Но в данной работе будет рассмотрена реализация B–дерева для внутренней памяти. Красно-черное дерево можно представить как B–дерево со степенью 4.

Часто встречается ситуация, когда дерево уже построено и надо часто обращаться к нему с какими-либо запросами. Обычно часть ключей являются часто используемыми, часть же используются достаточно редко. Тогда для m запросов необходимо $m \log n$ времени, что не всегда удобно.

Для решения таких задач используются сплай деревья, впервые описанные в 1985 году Робертом Тарьяном и Даниэлем Слейтором.

Эти деревья не являются постоянно сбалансированными, более того их высота

может быть пропорциональна количеству элементов, но перебаланси- ровка после каждого действия, такого как поиск, вставка или удаление элемента, таким образом, чтобы искомый элемент (или ближайший к нему по значению в случае удаления) становился корнем, приводит к тому, что поиск часто встречающихся элементов может быть пропорционален единице, так как часто встречающиеся элементы будут находиться вблизи корня и их перебалансировка и поиск потребует всего пары поворотов

Целью данной работы является реализация таких структур данных, как: красно-черное дерево, сплай–дерево и B–дерево, их анализ и оценка эффективности при работе с внутренней памятью.

1 Динамические структуры данных деревья

1.1 Красно–черное дерево

Красно–черное дерево — дерево бинарного поиска, то есть, слева от корня находятся элементы, меньшие корня, справа — элементы, большие корня и обладающее следующими дополнительными свойствами:

1. Все узлы окрашены двумя цветами — либо красным, либо черным.
2. Корень — черный.
3. Все листья — черные.
4. Оба потомка красного узла — черные.
5. Длина простого пути от данного узла до любого листа содержит одинаковое количество черных узлов. Эта высота называется *черной высотой* и обозначать h_b.

Для упрощения алгоритма под листьями понимают пустые узлы (то есть, все узлы дерева являются внутренними). Таким образом, все пустые листья являются черными, а узлы могут иметь любой цвет. В данном случае используются *фиктивные листья*.

Иногда снимается требование черного корня, так как корень является потомком всех узлов и перекраска с красного на черный цвет просто увеличивает черную высоту на единицу, что сохраняет свойство 5 красно-черного дерева.

1.2 Сплай–дерево

Сплай–дерево — это самобалансирующиеся дерево поиска. Самобалансировка происходит на каждом шаге вставки, поиска и удаления. В случае встав- ки новый элемент становится корнем. В случае поиска — искомый элемент становится корнем. В случае удаления — искомый элемент становится корнем, после чего дерево разбивается на два, потом сливается обратно.

Таким образом, наиболее часто используемые элементы находятся ближе к корню, наиболее редкие — дальше корня.

Основными операциями самобалансировки являются повороты, одинарные и двойные. Одинарные повороты ставят элемент x на место его родителя, двойные повороты — элемент x на место «деда».

Каждый узел дерева содержит 4 поля: информационное, левого ребенка, правого ребенка и родителя.

1.3 B–дерево

B–дерево — это самобалансирующееся сильно ветвистое (имеет больше двух детей у каждого узла) дерево поиска. В отличие от красно-черного дерева, каждый узел содержит больше одного элемента. Для B–дерева определяется минимальная степень узла t, которая выбирается в зависимости от условий поиска. В итоге каждый узел содержит от $t - 1$ до $2t - 1$ узла (корень может содержать один узел). Количество детей у каждого узла — не более $2t$. Элементы в узле расположены в порядке возрастания. Если узел содержит X элементов, то он имеет $X + 1$ детей.

B–дерево обладает следующими свойствами:

1. Каждый узел содержит следующие поля:

— количество элементов $n[x]$ в узле в настоящий момент;

— список элементов key_i, расположенных в порядке возрастания;

— переменную, определяющую является ли узел листом.

2. Каждый внутренний узел содержит еще и список из $n[x] + 1$ указателей $c_i(x)$ на дочерние узлы. У листьев эти указатели не определены.

3. Ключи key_i разделяют поддиапазоны ключей, хранящиеся в поддеревьях: если k_i — произвольный узел, хранящийся в поддереве с корнем $c_i(x)$,

то $k_1 \le key_1[x] \le k_2 \le key_2[x] \le \ldots \le key_{n[x]}[x] \le k_{n[x]+1}$.

4. Все листья расположены на одной и той же глубине, которая равна высоте дерева.

5. Существует минимальная степень узла $t > 2$, которая ограничивает количество элементов в узле и количество детей:

— Каждый узел, кроме корневого, содержит минимум $t - 1$ элемент и, соответственно, каждый внутренний узел содержит как минимум t детей.

— Каждый узел содержит не более $2t-1$ элементов, и, соответственно, не более $2t$ детей.

2 Сравнение эффективности алгоритмов

Так как красно-черное дерево можно представить как B–дерево со степенью 4, то основные исследования для B–дерева будут проведены для степени 4.

2.1 Теоретическая оценка высоты красно-черного дерева

При высоте h_b количество внутренних узлов не менее $2^{h_b-1} - 1$.

Докажем по индукции, используя обычную высоту узла x — $h(x)$:

1. При $h(x) = 1$ — получаем, что x — это узел, имеющий в качестве детей только фиктивные листья. Следовательно, черная высота $h_b = 1$ и количество внутренних узлов в поддереве x равно $2^{h_b-1} - 1 = 2^0 - 1 = 0$.

2. Так как любая внутренняя вершина содержит двух потомков (как узлы, так и фиктивные листы), их высоты на единицу меньше высоты узла x.

Черные высоты этих вершин могут быть h_b или h_b-1, если потомок красный или черный, соответственно. Тогда в каждом поддереве $2^{(h_b-1)-1} - 1$ вершин. Всего в поддереве $2 * (2^{h_b-2} - 1) + 1 = 2^{h_b-1} - 1$ вершин($+1$ — это сама вершина x).

Теперь представим, что x — это корень, тогда в дереве содержится не менее $2^{h_b-1} - 1$ внутренних узлов.

Красно-черное дерево с N узлами имеет высоту $h = O(\log N)$. Сложность: для вставки, удаления, поиска элемента — $O(\log N)$, расход памяти — $O(N)$.

2.2 Теоретическая оценка высоты для B–дерева

Пусть B–дерево имеет высоту h. Корень дерева содержит как минимум

один элемент, а остальные узлы — как минимум $t - 1$ элемент. Таким образом, на глубине 1 содержится как минимум два узла, на глубине 2 — минимум $2t$ узлов, на глубине 3 — минимум $2t^2$ узлов, на глубине h — $2t^{h-1}$.

Следовательно, общее число элементов:

$$n \geqslant 1 + (t-1)\sum_{i=1}^{n} 2t^{i-1} = 1 + 2(t-1)\frac{t^h - 1}{t-1} = 2t^h - 1.$$

Отсюда высота B–дерева не превышает превышает $\log_t \frac{n+1}{2}$.

Сложность: для вставки, удаления, поиска элемента — $O(\log N)$, расход памяти — $O(N)$.

2.3 Экспериментальная оценка высоты и времени создания дерева

Исследования проводились для двух наборов данных: случайные данные (диапазон от 0 до n) и упорядоченные данные (диапазон также от 0 до n.

Поскольку доказано, что все листья находятся на одинаковой высоте, то и для красно-черного и для B–дерева высота определяется просто как длина пути от корня до листа (минимального элемента) по левой ветке.

Так как поиск и удаление элемента из дерева соответствует проходу по одной из веток, а теоретически рассчитанная высота даже для $n = 5 \times 10^6$ не превосходит 30, то время работы функций поиска и удаления занимает 0 миллисекунд, поэтому учитывалось только время полного построения дерева и определялась высота полученного дерева.

В случае упорядоченных данных время работы больше, чем для случайных данных, что объясняется тем, что упорядоченные данные всегда добавляются в крайнюю правую ветку, что на каждом шаге при- водит к нарушению условий красно-черного дерева и приводит к постоянной перебалансировке.

Как и в теоретических вычислениях, так и в экспериментальных вычислениях высота для B–дерева меньше, чем высота красно-черного дерева, что говорит о том, что поиск в B–дереве происходит быстрее за счет меньшей высоты.

Был проведен эксперимент для построения B–дерева для различного количества элементов (от 500000 до 5000000). Время вычислено в миллисекундах. В случае упорядоченных данных время работы меньше, что для случайных данных, что объяс- няется тем, что в данном случае упорядоченные данные всегда добавляются в крайнюю правую ветку, но это приводит к тому, что разбиение на два узла происходит не на каждом шаге, а через 7 шагов, когда заполнится крайний правый лист. А случайные данные могут добавляться в разные листы и заполняются они тоже по-разному.

Также было проведено исследование как влияет степень B–дерева на высоту и время его построения. Эксперименты проводились для n = 500000. Высота в случае и случайных и упорядоченных данных остается одинаковой и уменьшается с ростом степени, что не вызывает сомнений. Время вычислялось в миллисекундах. Время работы для случайных данных существенно меньше, чем для упорядоченных, так как число узлов на каждом уровне равно $2t$, а упорядоченные данные добавляются всегда в самый правый узел.

Был проведен эксперимент по построению сплай дерева для различного количества элементов (от 500000 до 5000000). Время вычислено в миллисекундах. Видно, что в случае упорядоченных данных время работы меньше, чем для случайных данных, что объясняется тем, что последний добавленный элемент в любом случае является корнем дерева, а, поскольку последовательность возрастающая, то в любой момент дерево содержит только одну левую ветку, а новый элемент добавляется в правую ветку, что приводит к меньшему количеству поворотов, чем в случае случайных данных.

Высота дерева может быть больше, чем $O(\log n)$, так как новый элемент будет обязательно в корне. Но для случая неупорядоченных данных высота дерева сравнима с теоретической.

Для случая упорядоченных данных рекурсивная реализация поиска элемента приводит к переполнению стека, так как бинарное дерево поиска в данном случае приводит к дереву, имеющему только одну ветку высотой n.

Поскольку поиск элемента приводит к перестройке дерева, то было исследована глубина узла (расстояние от корня до данного узла).

В качестве искомых элементов были выбраны первые десять элементов входного файла:

[1489174, 730934, 796859, 1253125, 4931978, 3724078, 4375385, 3155324, 3821989, 844757].

Для каждого элемента сначала определялась текущая глубина, потом происходил поиск и перестройка дерева, так что искомый элемент становился корнем. Данный процесс повторялся семь раз. Данные исследования

проводились для $n = 500000$, $n = 2500000$, $n = 5000000$. Полученные результаты позволяют определить, что при каждом следующем вызове функции поиска элемент находится достаточно близко к корню, что позволяет делать меньшее количество поворотов.

На основании полученных результатов можно сделать вывод о том, что время построения красно-черного дерева для одних и тех же данных меньше, чем для сплай–дерева и B–дерева, что и позволяет использовать красно-черное дерево для встроенных структур многих языков программирования.

При этом, высота B–дерева существенно меньше, чем для красно–черного дерева и сплай–дерева. Но, если нет большой необходимости постоянно добавлять элементы, а необходим поиск данных, то лучше использовать сплай–дерево, так как часто использующиеся узлы будут находится на более высоком уровне дерева.

ЗАКЛЮЧЕНИЕ

В ходе выпускной квалификационной работы было проведено исследование распределения данных во внутренней памяти с помощью красно-черного, сплай–дерева и B–дерева.

Были разработаны соответствующие программы, проведен анализ эффективности данных структур для разного количества элементов.

Показано, что поиск и удаление элемента происходит быстро для $n =$ 5000000 составляет меньше миллисекунды. Построение дерева происходит медленнее для B–дерева, но высота для B–дерева существенно меньше, что позволяет предположить, что для больших данных поиск и удаление в B–дереве будет происходить эффективнее, чем в случае красно-черного дерева. Построение сплай–дерева происходит медленнее для случайных данных, чем для упорядоченных. Высота дерева в общем случае больше, чем для других сбалансированных деревьев. Но перебалансировка дерева, вызываемая после каждого действия, такого как поиск, вставка и удаление, приводит к тому, что наиболее часто вызываемые элементы находятся ближе к корню, а редко вызываемые элементы — дальше.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Кормен, Т. Х.* Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лазерсон, Р. Р. Риверст, К. Штайн. — М.: Издательский дом «Вильямс», 2019.

2 *Седжвик, Р.* Алгоритмы на C++ / Р. Седжвик. — М.: НОУ «ИНТУИТ», 2016.

3 *Sedgewick, R.* A dichromatic framework for balanced trees // 19th Annual Symposium on Foundations of Computer Science (sfcs 1978). — 1978. — Pp. 8– 21.

4 *Bayer, R.* Symmetric binary b–trees / R. Bayer // *Acta Informatica*. — 1972. — Vol. 1. — Pp. 290–306.

5 *Адельсон-Вельский, Г. М.* Один алгоритм организации информации / Г. М. Адельсон-Вельский, Е. М. Ландис // *Доклады АН СССР*. — 1962. — Т. 146,
№ 2. — С. 263–266.