



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)


ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ДОКЛАД

**о практическом задании по дисциплине АиСД
«Сбалансированные деревья: scaregoat дерево»**

направление подготовки 09.03.03 «Прикладная информатика»
профиль «Прикладная информатика в компьютерном дизайне»

Выполнил студент
гр. Б9121-09.03.03 пикд
Козлова  Светлана Евгеньевна

(подпись)

Руководитель практики
Доцент ИМКТ А.С Кленин
(должность, уч. звание)

(подпись)

«_____» _____ 2022г.

Доклад защищен:

С оценкой _____

Рег. № _____

«_____» _____ 2022 г.

г. Владивосток
2022

Оглавление

Введение	3
Сбалансированные деревья	5
Дерево козла отпущения	5
Степень сбалансированности	8
Структура вершины n :	9
Структура дерева T	9
Операция вставки элемента	9
Сравнение производительности различных структур данных	10
Список литературы	12

Введение

Бинарные деревья поиска обычно применяются для реализации множеств и ассоциативных массивов. Многообразие существующих бинарных деревьев и алгоритмов работы с данными большей частью обусловлены стремлением к максимально эффективному способу использования имеющихся ограниченных физических ресурсов. Кроме этого, различия в логике работы с данными отражают также и культурное влияние на умы человечества.

Одним из показательных примеров влияния мировой культуры на логические процессы человека можно считать “scapegoat tree” (англ. “дерево козла отпущения”). Название структуры данных отражает основной принцип ее работы: после критического изменения структуры перебалансировка происходит не от нового элемента (виновника ситуации), а от элемента (козла отпущения), поддеревья которого получились несбалансированными.

Одно из первых упоминаний данного феномена - возложения вины одного субъекта на другого - находит в религиозных трактатах иудаизма и христианства.

В праздник Йом-Киппур в Иерусалимский храм приводили жертвенный скот: овна во всесожжение, тельца и двух козлов. Первосвященник бросал жребий, и по его жребию одного из козлов закалывали в жертву вместе с тельцом (за грехи священника и народа), их кровью освящали скинию, а туши позже выносили и сжигали вне стана; а на другого козла первосвященник символически возлагал грехи всего еврейского народа, и его уводили в пустыню на мучительную смерть. Отсылание козла в пустыню символизирует очищение от всех грехов и уничтожение последствий всех дурных дел сынов Израиля в результате полного раскаяния и стремления приблизиться ко Всевышнему.

Талмуд рассказывает, что в период Первого Храма, когда Божественное присутствие было ощутимо и выражалось в постоянных чудесах, происходивших в Храме, одно из них было связано с козлом, посылаемым в пустыню. К рогам козла привязывали кусок красной шерсти, и, когда животное выводили из ворот храмового двора, один из кознов разрывал этот кусок шерсти пополам: одну половину вешал над воротами, а другую — снова привязывал к рогам козла. Если раскаяние народа было искренним и чистосердечным, то в тот момент, когда козла сбрасывали со скалы, кусок

красной шерсти, повешенный над воротами, становился белым в соответствии с тем, что сказано в книге пророка Исая: «Если будут грехи ваши как пурпур, побелеют они как снег, а если будут ярко-красными, станут белыми, подобно шерсти».

Таким образом, “Scapegoat Tree” — структура данных, представляющая собой частично сбалансированное дерево поиска (степень сбалансированности может быть настроена), такое что операции поиска, вставки и удаления работают за $O(\log n)$, при этом скорость одной операции может быть улучшена в ущерб другой.

Цель Scapegoat Tree - баланс дерева поиска без поворота дерева. Это большая выгода в тех случаях, когда после перебалансировки нужно пересчитывать узлы дерева.

Дерево было впервые разработано Arne Andersson в 1989 г. и, независимо от первой публикации, повторно описано Igal Galperin в соавторстве с Ronald L. Rivest в 1993 г.

Сбалансированные деревья

Деревья поиска — это структуры данных для упорядоченного хранения и простого поиска элементов. Широко применяются двоичные деревья поиска, в которых у каждого узла есть только два потомка. В этой статье рассмотрим два метода организации двоичных деревьев поиска: алгоритм Адельсон-Вельского и Ландиса (АВЛ-деревья) и ослабленные АВЛ-деревья (WAVL-деревья).

Двоичное дерево состоит из узлов, каждый узел хранит запись в виде пар ключ-значение (либо, в простом случае, только значений) и имеет не более двух потомков. Узлы-потомки различаются на правый и левый, причём выполняется условие на упорядоченность ключей: ключ левого потомка не больше, а правого — не меньше, чем ключ родительского узла. Дополнительно в узлах может храниться (и обычно хранится) служебная информация — например, ссылка на родительский узел или другие данные. Специальными случаями являются корневой узел, с которого происходит вход в дерево, и пустой узел, который не хранит никакой информации. Узлы, у которых оба потомка пустые, называются листьями дерева. Узел со всеми потомками образует поддереву. Таким образом, каждый узел либо является корнем какого-то поддерева, либо листом.

В связи с тем, что есть алгоритмы, в которых дереву позволяют "разбалтываться", но после этого довольно долго операции можно проводить за логарифмическое время, прежде чем придется долго приводить дерево обратно в сбалансированное состояние, различают гарантированное и амортизированное время вставки/удаления элемента. Для некоторых реализаций операций с двоичными деревьями поиска сложность вставки и удаления $O(\log 2N)$ является гарантированной, для некоторых — амортизированной, с ухудшением до $O(N)$.

Дерево козла отпущения

Дерево козла отпущения — это самобалансирующееся бинарное дерево поиска, основанное на концепции козла отпущения. Как правило, козел отпущения — это тот, кого обвиняют, когда что-то идет не так. Для бинарных деревьев поиска проблема, которая замедляет поиск, заключается в том, что дерево становится несбалансированным. Двоичные деревья поиска могут стать несбалансированными после вставок и удалений.

Деревья козла отпущения ищут узел козла отпущения, чтобы «обвинить» дерево в несбалансированности, чтобы дерево могло начаться с этого узла и решить проблему. Определив козла отпущения или узел, вызвавший несбалансированное дерево, можно выполнить операцию частичного восстановления. Эта операция берет все поддереву, где находится козел отпущения, деконструирует и перестраивает его в идеально сбалансированное поддерево.

К достоинствам Scapegoat-дерева можно отнести:

- Отсутствие необходимости хранить какие-либо дополнительные данные в вершинах (а значит мы выигрываем по памяти у красно-черных, АВЛ и декартовых деревьев)
- Отсутствие необходимости перебалансировать дерево при операции поиска (а значит мы можем гарантировать максимальное время поиска $O(\log N)$, в отличие от Splay-деревьев, где гарантируется только амортизированное $O(\log N)$)
- Амортизированная сложность операций вставки и удаления $O(\log N)$ — это в общем-то аналогично остальным типам деревьев
- При построении дерева мы выбираем некоторый коэффициент «строгости» α , который позволяет модифицировать дерево, делая операции поиска более быстрыми за счет замедления операций модификации или наоборот.

К недостаткам можно отнести:

- В худшем случае операции модификации дерева могут занять $O(n)$ времени (амортизированная сложность у них по-прежнему $O(\log N)$, но защиты от «плохих» случаев нет).

- Можно неправильно оценить частоту разных операций с деревом и ошибиться с выбором коэффициента α — в результате часто используемые операции будут работать долго, а редко используемые — быстро.

Пусть n = количество элементов в дереве, а q = верхняя граница количества элементов в дереве, где q иногда корректируется, чтобы гарантировать, что n всегда находится между $q/2$ и q .

Дерево козла отпущения имеет следующие ограничения:

- $q/2 \leq n \leq q$
 - Значение n всегда отслеживает количество узлов в дереве. Значения n и q являются счетчиками, которые увеличиваются при добавлении элементов в дерево. Значение n уменьшается на 1, когда элемент удаляется из дерева. Если n когда-либо становится меньше, чем $q/2$, дерево перестраивается, чтобы оставаться в пределах ограничения (q сбрасывается до $q = n$) и поддерживать баланс дерева.
- высота: Максимально допустимая высота составляет $\log_{1/\alpha} q \leq \log_{1/\alpha} 2n < \log_{1/\alpha} n + 2$, где α — значение от $1/2$ до 1 . Для этих уроков α было выбрано равным $2/3$, чтобы пояснения были более понятными.
 - С заменой α на $2/3$ высота теперь составляет $\log_{3/2} q \leq \log_{3/2} 2n < \log_{3/2} n + 2$. Если дерево является действительным деревом козлов отпущения, максимальная высота составляет $\log_{3/2} n$. кв.
 - Логарифмическая функция описывает, сколько раз сбалансированное дерево расщепляется и разветвляется в зависимости от высоты. Если дерево имеет высоту, которая в настоящее время слишком высока для его размера, это означает, что дерево несбалансировано.
 - Высота вычисляется как максимальное количество ребер, чтобы добраться от корня до листа. Это число равно максимальной глубине, которая представляет собой количество ребер от самого глубокого узла до корня. При таком определении корень находится на глубине 0.
 - С альфой, установленной на $2/3$, деревья, которые кажутся мне несбалансированными, на самом деле сбалансированы. Ниже

приведен пример из книги « Открытые структуры данных » Пэта Морины , где показано сбалансированное дерево козла отпущения $2/3$, которое может показаться несбалансированным. Оно сбалансировано, потому что $height = 5$, $n = q = 10$ и $height = 5 < \log_{3/2} q = 5,68$.

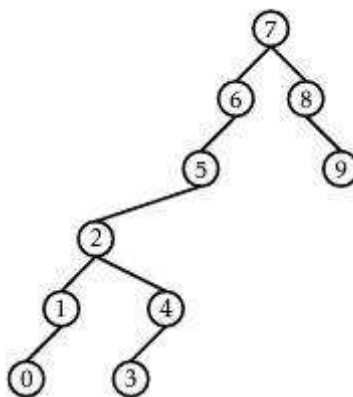


Рисунок 1 - Пример дерева с 10 элементами

Степень сбалансированности

Будем считать, что дерево является сбалансированным, если выполняются следующие: Введем коэффициент α , который показывает, насколько дерево может быть несбалансированным. Математически это выглядит следующим образом: $1/2 \leq \alpha \leq 1$; $size(left[x]) \leq \alpha \cdot size(x)$; $size(right[x]) \leq \alpha \cdot size(x)$, где $size(left[x])$ и $size(right[x])$ — размер левого и правого поддерева вершины x .

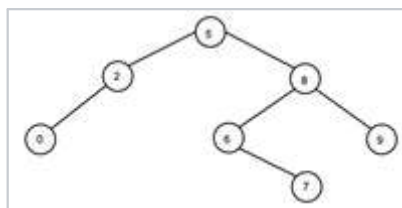


Рисунок 2 - Пример дерева с $\alpha = 0.6$

Чем больше α , тем глубже получится дерево, следовательно, запросы к дереву будут более эффективные, но балансировка дерева будет менее эффективная.

Структура вершины n :

$n.key$ — значение в вершине

$n.left$ — левый ребенок вершины

$n.right$ — правый ребенок вершины

n.height — высота поддерева данной вершины

n.depth — глубина вершины

n.parent — ссылка на родителя

n.sibling — ссылки на "братьев" данной вершины

Структура дерева T

T.root — ссылка на корень дерева

T.size — число вершин в дереве

T.maxSize — максимальное число вершин в дереве после последней перебалансировки

T.hα — вспомогательное значение, вычисляется как:

$T.h\alpha = \lfloor \log 1/\alpha(T.size) \rfloor$

$T.h\alpha = \lfloor \log 1/\alpha(T.size) \rfloor$

T.height — высота дерева

Операция вставки элемента

Пока дерево остается α-сбалансированным, выполняем модифицированную вставку элемента в дерево, которая аналогична обычной вставке в двоичное дерево, но операция ввода будет возвращать глубину данной вершины. В тот момент, когда дерево стало несбалансированным, надо начать поиск вершины, которая нарушает условие сбалансированности. Для этого надо пройти по дереву вверх. Только что вставленная вершина ей быть не может, так как является листом - идеально сбалансированным элементом. После нахождения этой вершины надо запустить операцию перебалансировки.

Нам нужна специальная функция, которая позволяет найти тот элемент дерева, который испортил баланс. Её смысл заключается в рекурсивной проверке родительских элементов на сбалансированность поддеревьев. После нахождения нужного элемента все его дерево перестраиваем, положив значения в вспомогательный массив и воссоздав из него сбалансированное дерево.

Сравнение производительности различных структур данных

Как и любое другое сбалансированное дерево, у дерева козла отпущения имеется преимущество перед стандартными структурами хранения данных.

Таблица 1. Сравнение асимптоматики операций

Операция	Scapegoat tree	хэш-таблица	Список	Сортированный массив
Поиск	$O(\log N)$	$O(1)$	$O(N)$	$O(\log N)$
Вставка	$O(1)$	$O(1)$	$O(1)$	$O(N)$
Удаление	$O(\log N)$	$O(1)$	$O(N)$	$O(N)$
Поиск ближайшего ключа	$O(\log N)$	$O(N)$	$O(N)$	$O(\log N)$

Для улучшения работы имеющейся структуры можно применять следующее: хранить все данные в списке, разбитом на $\log n$ подмножеств, среднее значение которых хранить в отдельном небольшом scapegoat tree. Тогда получится, что при помощи дополнительного дерева ускорятся операции вставки и удаления, так как операции над небольшими объемами данных заведомо быстрее операций над большими структурами.

Также можно назначить бинарные представления элементам деревьев для более быстрого сравнения, причем можно сэкономить место, занося данные в бинарном виде. Например, левое поддерев будет обозначаться 00, правое - 11, а сам элемент - 01. Бинарные номера для удобства можно будет хранить и сравнивать в целочисленном значении. Приблизительное ограничение длины бинарных номеров $4 \cdot \log n \geq L \geq \log n$. В случае переполнения длины номеров можно будет переназначить номера большего размера за $O(\log n)$, так как мы просто сделаем обход имеющегося дерева.

Список литературы

1. Структура данных - Википедия [Электронный ресурс] - Режим доступа: https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#:~:text=%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%20
2. Сбалансированное бинарное дерево из отсортированного массива | by Viktor Karpov | Medium [Электронный ресурс] - Режим доступа: <https://medium.com/@vitkarpov/cracking-the-coding-interview-4-2-9567d6986853>
3. Сбалансированное дерево поиска B-tree (t=2) [Электронный ресурс] - Режим доступа: <https://habr.com/ru/post/337594/>
4. Структура данных, Список структур данных - Структуры и типы данных языка программирования. Трансляция, компиляция, интерпретация [Электронный ресурс] - Режим доступа: https://studbooks.net/2048107/informatika/ctrukтура_dannyh
5. АВЛ-дерево (AVL-Tree) - что это: построение, балансировка, удаление. [Электронный ресурс] - Режим доступа: <https://blog.skillfactory.ru/glossary/avl-derevo/#:~:text=%D0%B5%D1%89%D0%B5%20%D0%BC%D0%BE%D0%B6%D0%B5%D1%82%20%D0%B2%D1%8B%D1%80%D0%BE%D0%B4%D0%B8%D1%82%D1%8C%D1%81%D1%8F.-%D0%A7%D1%82%D0%BE%20%D1%82%D0%B0%D0%BA%D0%BE%D0%B5%20%D0%B1%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0,%D0%B1%D0%BE%D0%BB%D1%8C%D1%88%D0%B5%20%D1%87%D0%B5%D0%BC%20%D0%BD%D0%B0%20%D0%BE%D0%B4%D0%B8%D0%BD%20%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C>
6. ScapeGoat Tree | Set 1 (Introduction and Insertion) - GeeksforGeeks [Электронный ресурс] - Режим доступа: <https://www.geeksforgeeks.org/scapegoat-tree-set-1-introduction-insertion/>
7. Scapegoat Tree | Brilliant Math & Science Wiki [Электронный ресурс] - Режим доступа: <https://brilliant.org/wiki/scapegoat-tree/>

8. Lesson 1: How Do Scapegoat Trees Work? [Электронный ресурс] - URL: <http://www.domiciaherring.com/lesson-1-how-do-scapegoat-trees-work>
9. Краткий анализ Balance Tree (2) -Treap - Русские Блоги [Электронный ресурс] - Режим доступа: <https://russianblogs.com/article/8514309979/>
10. Segment tree with insert operator without using tree rotations - Codeforces [Электронный ресурс] - Режим доступа: <https://codeforces.com/blog/entry/13554?locale=ru>
11. Структуры данных: бинарные деревья. Часть 2: обзор сбалансированных деревьев / Хабр [Электронный ресурс] - Режим доступа: <https://habr.com/ru/post/66926/>
12. Scapegoat-деревья / Хабр [Электронный ресурс] - Режим доступа: <https://habr.com/ru/company/infopulse/blog/246759/>
13. Scapegoat trees | Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms [Электронный ресурс] - Режим доступа: <https://dl.acm.org/doi/10.5555/313559.313676>
14. Scapegoat Tree — Викиконспекты [Электронный ресурс] - Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Scapegoat_Tree
15. A. Andersson. Improving Partial Rebuilding by Using Simple Balance Criteria, In Proc. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 382, Springer Verlag, pages 393--402, 1989. [Электронный ресурс] - Режим доступа: <https://user.it.uu.se/~arnea/abs/partb.html>
16. ScapeGoat Tree | Set 1 (Introduction and Insertion) - GeeksforGeeks [Электронный ресурс] - Режим доступа: <https://www.geeksforgeeks.org/scapegoat-tree-set-1-introduction-insertion/>
17. Scapegoat Tree | Brilliant Math & Science Wiki [Электронный ресурс] - Режим доступа: <https://brilliant.org/wiki/scapegoat-tree/>
18. Lesson 1: How Do Scapegoat Trees Work? - Domicia Herring [Электронный ресурс] - Режим доступа: <http://www.domiciaherring.com/lesson-1-how-do-scapegoat-trees-work>
- 19.8 Scapegoat Trees [Электронный ресурс] - Режим доступа: <https://opendatastructures.org/newhtml/ods/latex/scapegoat.html>
20. Scapegoat tree [Электронный ресурс] - Режим доступа: <https://iq.opengenus.org/scapegoat-tree/>
21. About Scapegoat — Domicia Herring [Электронный ресурс] - Режим доступа: <http://www.domiciaherring.com/history-of-the-scapegoat-tree>
22. A data structure called Scapegoat Tree | by Sani Ahamed | Medium [Электронный ресурс] - Режим доступа:

<https://medium.com/@thesua711/a-data-structure-called-scapegoat-tree-6a33528e21ed>

23. GitHub - thesua7/Scapegoat-Tree-Data-Structure: Scapegoat tree implementation [Электронный ресурс] - Режим доступа:
<https://github.com/thesua7/Scapegoat-Tree-Data-Structure>
24. Дополнительные главы алгоритмов, часть 1, осень 2021 [Электронный ресурс] - Режим доступа:
<https://compscicenter.ru/courses/advanced-algo/2021-autumn/classes/>
25. Левит — Викитека [Электронный ресурс] - Режим доступа:
<https://ru.wikisource.org/wiki/%D0%9B%D0%B5%D0%B2%D0%B8%D1%82#16>