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 задание по дисциплине АиСД

«Сбалансированные деревья: scapegoat дерево»

направление подготовки 09.03.03 «Прикладная информатика»

профиль «Прикладная информатика в компьютерном дизайне»

|  |  |  |
| --- | --- | --- |
|  |  | Выполнил студент  гр. Б9121-09.03.03пикд  Филончикова Анна Сергеевна \_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Доклад защищен:  С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | (подпись)  Руководитель практики  Доцент ИМКТ А.С Кленин  (должность, уч. звание)  \_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2022г. |
| Рег. № \_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |  |  |

г. Владивосток

2022

# Оглавление

[1. Аннотация 3](#_Toc125317422)

[2. Введение 4](#_Toc125317423)

[2.1. Неформальная постановка задачи 4](#_Toc125317424)

[2.2. Формальная постановка задачи 4](#_Toc125317425)

[2.3. Авторы и история 4](#_Toc125317426)

[2.4. Суть алгоритма 4](#_Toc125317427)

[2.5. Перспективы использования 4](#_Toc125317428)

[3. Метод 6](#_Toc125317429)

[3.1. Описание алгоритма 6](#_Toc125317430)

[3.1.1. Поиск 6](#_Toc125317431)

[3.1.2. Вставка элемента 6](#_Toc125317432)

[3.1.3. Удаление элемента 7](#_Toc125317433)

[3.1.4. Перебалансировка 7](#_Toc125317434)

[3.2. Псевдокод 8](#_Toc125317435)

[3.2.1. Поиск 8](#_Toc125317436)

[3.2.2. Вставка элемента 8](#_Toc125317437)

[3.2.3. Удаление элемента элемента 8](#_Toc125317438)

[3.2.4. Получение списка 9](#_Toc125317439)

[3.2.5. Построение дерева 9](#_Toc125317440)

[3.2.6. Перестроение дерева 9](#_Toc125317441)

[3.3. Пример 10](#_Toc125317442)

[4. Результаты исследования 11](#_Toc125317443)

[5. Заключение 12](#_Toc125317444)

[6. Список литературы 13](#_Toc125317445)

# Аннотация

Бинарные деревья поиска обычно применяются для реализации множеств и ассоциативных массивов (например, set и map в с++).

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.

При работе необходимо поддерживать состояние сбалансированного дерева, иначе время работы операции поиска может превысить O(log⁡n).

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

Этот тип дерева был изобретен в 1989 году Арне Андерссоном, затем заново изобретен в 1993 году Игалом Гальперином и Рональдом Л. Ривестом .

На данное время на просторах интернета достаточно мало информации об этой структуре данных. Чтобы в полной мере узнать эту структуру данных, нужно прочитать достаточно много источников и выбрать из них только важную информацию.

# Введение

## Неформальная постановка задачи

Алгоритм дерева должен принимать на ввод числа и выдавать построенное дерево в прямом и симметричном обходе. Они должны соответствовать описанию данных деревьев и выполнять функции поиска, вставки и удаления элемента, функция вывода дерева. Дерево должно самостоятельно балансироваться при нарушении баланса.

Основной набор тестов должен определять время выполнения набора команд и затрачиваемую память. Также следует написать тесты, проверяющие его производительность и корректность работы.

## Формальная постановка задачи

1. Найти и проанализировать различные русскоязычные и англоязычные источники. Реализовать алгоритм ScapeGoat Tree и описать его в форме научного доклада
2. Реализовать алгоритм
3. Написать тесты для анализа производительности алгоритма
4. Загрузить результаты работ на GitHub

## Авторы и история

Этот тип дерева был изобретен в 1989 году Арне Андерссоном, затем заново изобретен в 1993 году Игалом Гальперином и Рональдом Л. Ривестом .

## Суть алгоритма

В отличие от большинства других самобалансирующихся деревьев, дерево козла отпущения реструктурируется реже. Таким образом, структура дерева мало-помалу становится неуравновешенной, пока алгоритм не определит узел- козел отпущения, ответственный за этот дисбаланс. Затем выполняется перебалансировка, чтобы дерево находило удовлетворительную структуру.

## Перспективы использования

Главный интерес к дереву козла отпущения связан с его пространственной сложностью. Это первое двоичное дерево поиска, чьи операции в среднем составляют O (log (n)), где n - количество узлов, и которое не занимает больше памяти, чем двоичное дерево поиска. Действительно, в отличие от двухцветных деревьев и AVL, которые хранят дополнительную информацию в узлах (например, его цвет или высоту), козел отпущения хранит в памяти только метку узла и два указателя на его дочерние элементы. Таким образом, это дерево более экономно по памяти.

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 поддерево, где находится козел отпущения, реконструирует и перестраивает его в идеально сбалансированное поддерево.

# Метод

## Описание алгоритма

Будем считать, что дерево является сбалансированным, если выполняются следующее: Введем коэффициент α, который показывает, насколько дерево может быть несбалансированным. Математически это выглядит следующим образом:

1/2⩽α⩽1 size(left[x])⩽α⋅size(x) ; size(right[x])⩽α⋅size(x), где size(left[x]) и size(right[x]) — размер левого и правого поддерева вершины x.

### Поиск

Поскольку это двоичное дерево поиска, то и поиск будет стандартным: идём от корня, сравниваем вершину с искомым значением, если нашли — возвращаем, если значение в вершине меньше — рекурсивно ищем в левом поддереве, если больше — в правом. Дерево по ходу поиска не модифицируется.

### Вставка элемента

Пока дерево остается α-сбалансированным, выполняем модифицированную вставку элемента в дерево, которая аналогична обычной вставке в двоичное дерево, но операция InsertKey(k) 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. После нахождения этой вершины надо запустить операцию перебалансировки.

Нам нужна специальная функция FindScapegoat(n), которая позволяет найти тот элемент дерева, который испортил баланс (именно из-за этой процедуры дерево так называется. Scapegoat - "козел отпущения", который испортил баланс).

n — узел дерева. Обычно, процедура вызывается от только что добавленной вершины.

### Удаление элемента

Сначала надо удалить вершину, как в обычном двоичном дереве. Потом надо проверить дерево на сбалансированность. Если дерево осталось сбалансированным, ничего делать не надо. В противном случае надо начать перебалансировку дерева.

Функция DeleteKey(k) удаляет элемент, аналогично удалению в бинарном дереве, и возвращает глубину удаленного элемента.

k — ключ, который будет удален.

### Перебалансировка

Общая идея процедуры выглядит следующим образом: сначала из исходного дерева мы получаем список вершин в неубывающем порядке, при этом вершина стоит в списке перед детьми. После этого мы создаем новое дерево из списка.

1. Обходим всё поддерево Scapegoat-вершины (включая её саму) с помощью in-order обхода — на выходе получаем отсортированный список (свойство In-order обхода бинарного дерева поиска).
2. Находим медиану на этом отрезке, подвешиваем её в качестве корня поддерева.
3. Для «левого» и «правого» поддерева рекурсивно повторяем ту же операцию.

## Псевдокод

### Поиск

root — корень дерева или поддерева, в котором происходит поиск.

k — искомый ключ в дереве.

**Search**(root, k):

**if** root = null or root.key = k:

**return** root

**else if** k ≤ root.left.key:

**return** Search(root.left, k)

**else**:

**return** Search(root.right, k)

**FindScapegoat**(n):

size = 1

height = 0

**while** (n.parent <> null):

height = height + 1

totalSize = 1 + size + n.sibling.size()

**if** height > ⌊log1/α(totalSize)⌋:

**return** n.parent

n = n.parent

size = totalSize

### Вставка элемента

k — ключ, который будет добавлен в дерево.

**Insert**(k):

height = InsertKey(k)

**if** height = −1:

**return** false;

**else if** height > T.hα:

scapegoat = **FindScapegoat**(Search(T.root, k))

**RebuildTree**(n.size(), scapegoat)

**return** true

### Удаление элемента элемента

k — ключ, который будет удален.

**Delete**(k):

deleted = **DeleteKey**(k)

**if** deleted:

**if** T.size < (T.α · T.maxSize):

**RebuildTree**(T.size, T.root)

### Получение списка

root — корень дерева, которое будет преобразовано в список.

**FlattenTree**(root, head):

**if** root = null:

**return** head

root.right = FlattenTree(root.right, head)

**return** FlattenTree(root.left, root)

### Построение дерева

size — число вершин в списке.

head — первая вершина в списке.

BuildHeightBalancedTree(size, head):

**if** size = 1 then:

return head

**else if** size = 2 then:

(head.right).left = head

**return** head.right

root = (BuildHeightBalancedTree(⌊(size − 1)/2⌋, head)).right

last = BuildHeightBalancedTree(⌊(size − 1)/2⌋, root.right)

root.left = head

**return** last

### Перестроение дерева

size — число вершин в поддереве.

scapegoat — вершина, которая испортила баланс.

**RebuildTree**(size, scapegoat):

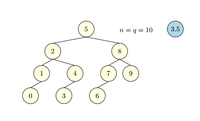
head = **FlattenTree**(scapegoat, null)

**BuildHeightBalancedTree**(size, head)

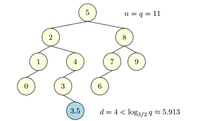
**while** head.parent!=null do

head = head.parent

## Пример



Вставка без нарушения баланса 1



Вставка без нарушения баланса 2



# Результаты исследования

# Заключение

# Список литературы

1. [Структура данных — Википедия](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. [АВЛ-дерево — Википедия](https://ru.wikipedia.org/wiki/%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE)
3. [Сбалансированное бинарное дерево из отсортированного массива | by Viktor Karpov | Medium](https://medium.com/@vitkarpov/cracking-the-coding-interview-4-2-9567d6986853)
4. [Сбалансированное дерево поиска B-tree (t=2)](https://habr.com/ru/post/337594/)
5. [3. Операции](https://studfile.net/preview/386395/page:2/)
6. [Cтруктура данных, Список структур данных - Структуры и типы данных языка программирования. Трансляция, компиляция, интерпретация](https://studbooks.net/2048107/informatika/ctruktura_dannyh)
7. [АВЛ-дерево (AVL-Tree) - что это: построение, балансировка, удаление](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).
8. [Scapegoat Tree — Викиконспекты](https://neerc.ifmo.ru/wiki/index.php?title=Scapegoat_Tree)
9. [Scapegoat tree - Wikipedia](https://en.wikipedia.org/wiki/Scapegoat_tree)
10. [ScapeGoat Tree | Set 1 (Introduction and Insertion) - GeeksforGeeks](https://www.geeksforgeeks.org/scapegoat-tree-set-1-introduction-insertion/)
11. [Scapegoat Tree | Brilliant Math & Science Wiki](https://brilliant.org/wiki/scapegoat-tree/)
12. [Lesson 1: How Do Scapegoat Trees Work?](http://www.domiciaherring.com/lesson-1-how-do-scapegoat-trees-work)
13. [Scapegoat Trees](http://webhotel4.ruc.dk/~keld/teaching/algoritmedesign_f03/Artikler/03/Galperin93.pdf)
14. [Scapegoat Trees](https://people.csail.mit.edu/rivest/pubs/GR93.pdf)
15. [Scapegoat tree](https://iq.opengenus.org/scapegoat-tree/)
16. [Scapegoat-деревья / Хабр](https://habr.com/ru/company/infopulse/blog/246759/)
17. [Дерево со штрафами (scapegoat tree)](https://barmrus.livejournal.com/2823.html)
18. !! [Структура данных Scapegoat Tree презентация, доклад](https://thepresentation.ru/uncategorized/struktura-dannyh-scapegoat-tree)
19. [Структура данных Scapegoat Tree - online presentation](https://en.ppt-online.org/205988)
20. [Дерево козлов отпущения - Scapegoat tree - Википедия](https://wiki5.ru/wiki/Scapegoat_tree)
21. [Статьи по теме тега: "Scapegoat дерево"](https://russianblogs.com/tag/Scapegoat+%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE/)
22. [ключ-значение» для использования в самоадаптирующихся контейнерах данных' - журнал 'Кибернетика и программирование' - NotaBene.ru](https://e-notabene.ru/kp/article_23557.html)
23. [Козел отпущения - Scapegoat tree](https://wikicsu.ru/wiki/Scapegoat_tree)
24. [Сборка Знаний. Д.Потапов, С. Переслегин](https://novids.com/video/-166353059_456247154)
25. [Алгоритмы на сайте Игоря Гаршина](https://www.garshin.ru/it/algorythms.html)
26. [Краткий анализ Balance Tree (2) -Treap - Русские Блоги](https://russianblogs.com/article/8514309979/)
27. [Segment tree with insert operator without using tree rotations - Codeforces](https://codeforces.com/blog/entry/13554?locale=ru)
28. [height-balanced tree — с английского на все языки](https://translate.academic.ru/height-balanced%20tree/en/xx/)