

Scapegoat Tree

Выполнил Димов А.А. ИМ15-06Б

Руководитель Олейников Б.В.

Разработана Arne Andersson, Igal Galperin, Ronald L. Rivest в 1962г.



Ronald L. Rivest



Arne Andersson

Определение

- Scapegoat tree – структура данных, представляющее из себя самобалансирующееся бинарное дерево поиска.
- Операции поиска, вставки и удаления работают за $O(\log N)$, при этом скорость одной операции может быть улучшена за счет другой

Понятия, необходимые для работы с данным деревом:

- T -дерево
- $root[T]$ – корень дерева T
- $left[x], right[x]$ – левые и правый "сын" вершины
- $brother(x)$ – брат вершины (имеет общего родителя)
- $depth(x)$ – глубина вершины (расстояние от вершины до корня)
- $height(x)$ – глубина дерева T
- $size(x)$ – вес вершины x (кол-во всех ее дочерних вершины+1)
- $size[T]$ – размер дерева T (вес корня)
- $max_size[T]$ – максимальный размер дерева T .

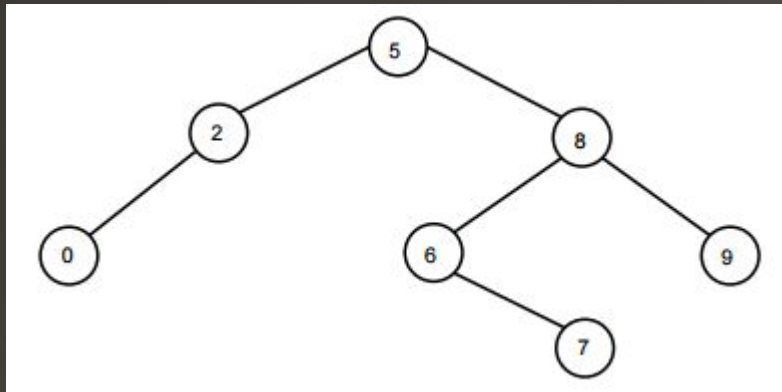
- При работе необходимо поддерживать состояние сбалансированного дерева, иначе время работы операции поиска может превысить $O(\log n)$.

Степень сбалансированности

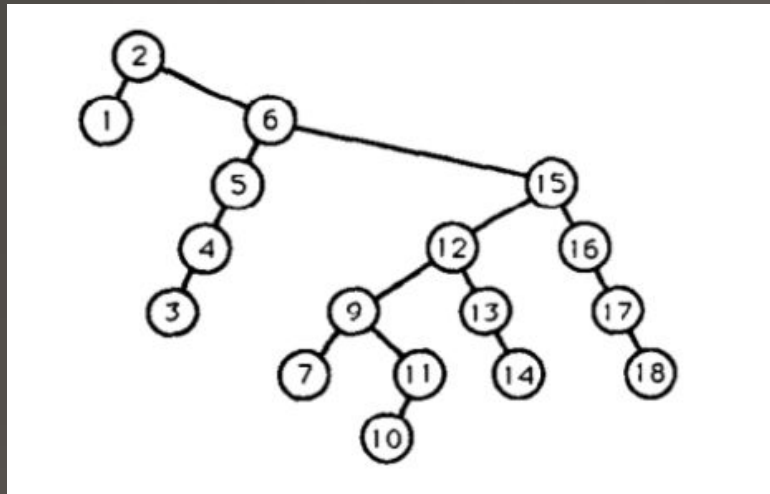
- Коэффициент α — это число в диапазоне от $[0.5; 1)$, определяющее требуемую степень качества балансировки дерева. Некоторая вершина x называется " α -сбалансированной по весу", если вес её левого сына меньше либо равен $\alpha * \text{size}(x)$ и вес её правого сына меньше либо равен $\alpha * \text{size}(x)$.

$$\begin{aligned} \text{size}(\text{left}[x]) &\leq \alpha * \text{size}(x) \\ \text{size}(\text{right}[x]) &\leq \alpha * \text{size}(x) \end{aligned}$$

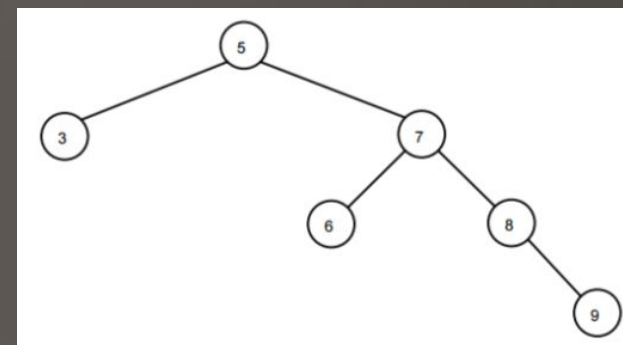
Примеры



$a = 0.6$



$a = 0.57$



$a = 0.55$

Плюсы и минусы Scaregoat дерева

Плюсы

- Скорость одних операций возможно улучшить за счет других операций.
- Scaregoat tree работает быстрее, чем красно-черное дерево и декартово.
- Требуется меньше памяти (не надо хранить информацию для балансировки).
- Настройки скорости меняются в процессе выполнения.
- Не требуется перебалансировать дерево при поиске.

Минусы

- В худшем случае операции модификации дерева могут занять $O(N)$ времени.
- В случае неправильного выбора параметра *alpha* часто используемые операции будут работать долго, а редко используемые – быстро. При этом дерево будет уступать остальным по скорости

Поиск

- Данная операция стандартна для двоичного дерева поиска. Необходимо пройти от корня, сравнивая каждую вершину с искомым значением, если найдено – возврат значения, иначе, если значение в вершине меньше, то рекурсивно ищем в левом поддереве, если больше – в правом.
- Сложность операции зависит от *alpha*:
 - $O(\log_{1/a} N)$

Вставка

- Начинается вставка нового элемента в Scaragoat-дерево классически: поиском ищем место, куда бы повесить новую вершину и подвешиваем.
- Легко понять, что это действие могло нарушить α -балансировку по весу для одной или более вершин дерева. И вот теперь начинается то, что и дало название структуре данных: мы ищем «козла отпущения» (Scaragoat-вершину) — вершину, для которой потерял α -баланс и её поддереву должно быть перестроено.
- Сама только что вставленная вершина, хотя и виновата в потере баланса, «козлом отпущения» стать не может — у неё ещё нет «детей», а значит её баланс идеален.
- Соответственно, нужно пройти по дереву от этой вершины к корню, пересчитывая веса для каждой вершины по пути. Если на этом пути встретится вершина, для которой критерий α -сбалансированности по весу нарушился — мы полностью перестраиваем соответствующее ей поддерево так, чтобы восстановить α -сбалансированность по весу.

Перебалансировка

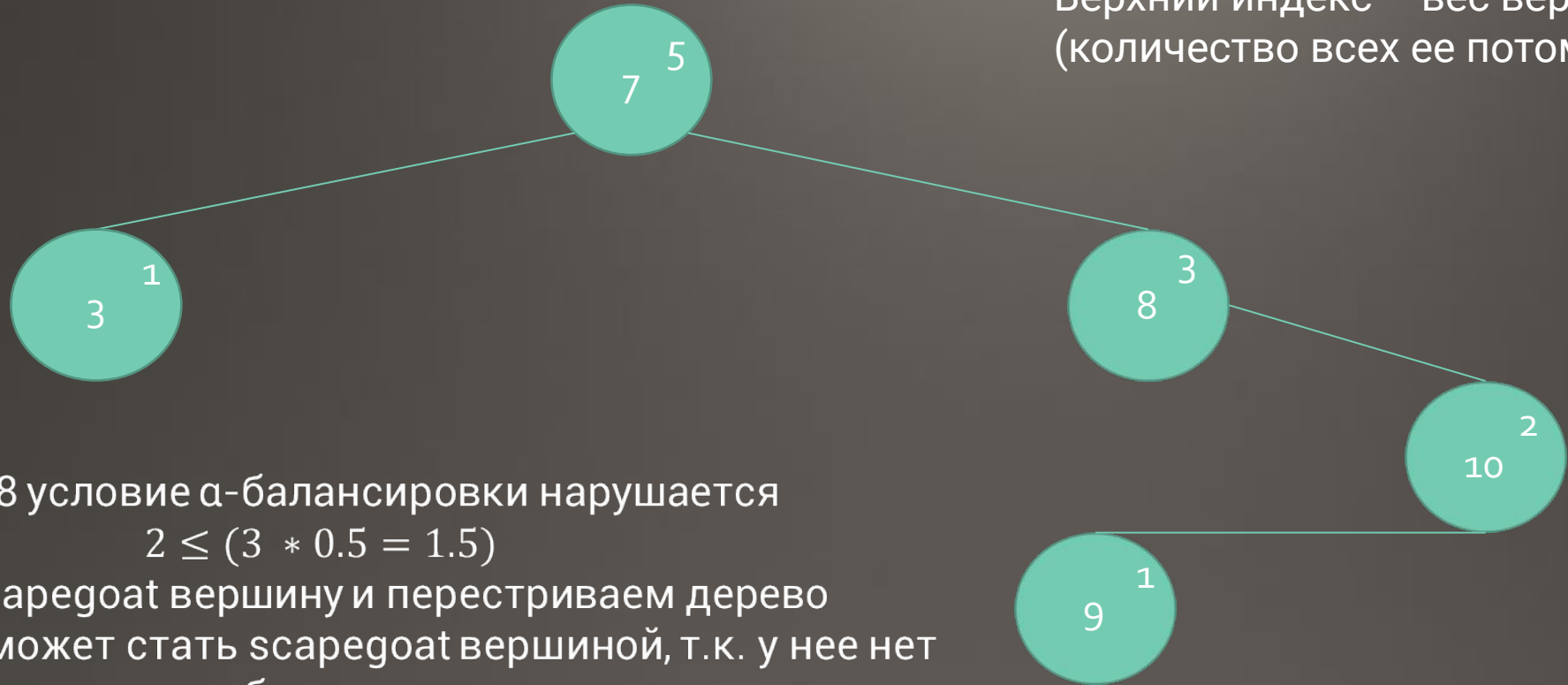
Обходим всё поддерево Scaregoat-вершины (включая её саму) с помощью in-order обхода — на выходе получаем отсортированный список (свойство In-order обхода бинарного дерева поиска).

- Находим медиану на этом отрезке, подвешиваем её в качестве корня поддерева.
- Для «левого» и «правого» поддерева рекурсивно повторяем ту же операцию.

```
if start > ends then
  begin
    result := Nil;
    exit;
  end;
mid := ceil((start + ends) / 2.0);
d := nodesList[mid];
p := Node.Create(d.key);
leftNode := self.buildTreeFromSortedList(nodesList, start, mid-1);
p.left := leftNode;
rightNode := self.buildTreeFromSortedList(nodesList, mid+1, ends);
p.right := rightNode;
result := p;
```

Пример:
 $\alpha = 0.5$

Верхний индекс – вес вершины
(количество всех ее потомков + 1)



- Для вершины 8 условие α -балансировки нарушается
 $2 \leq (3 * 0.5 = 1.5)$
- Берем 8 как sарегоат вершину и перестраиваем дерево
- Вершина 9 не может стать sарегоат вершиной, т.к. у нее нет детей, а следовательно ее баланс идеален

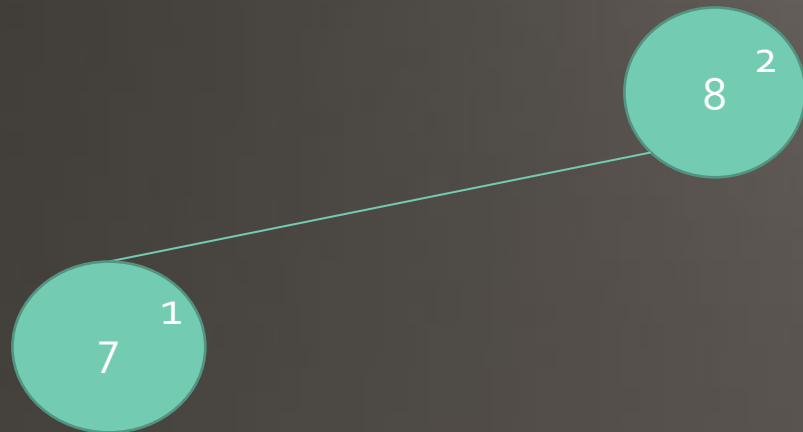
Пример:
 $\alpha = 0.5$



Верхний индекс – вес вершины
(количество всех ее потомков + 1)

- Вычислив медиану (2) ($start = 0$, $ends = 4$), подвешиваем 8 в качестве корня поддерева (в данном случае корень самого дерева)

Пример:
 $\alpha = 0.5$

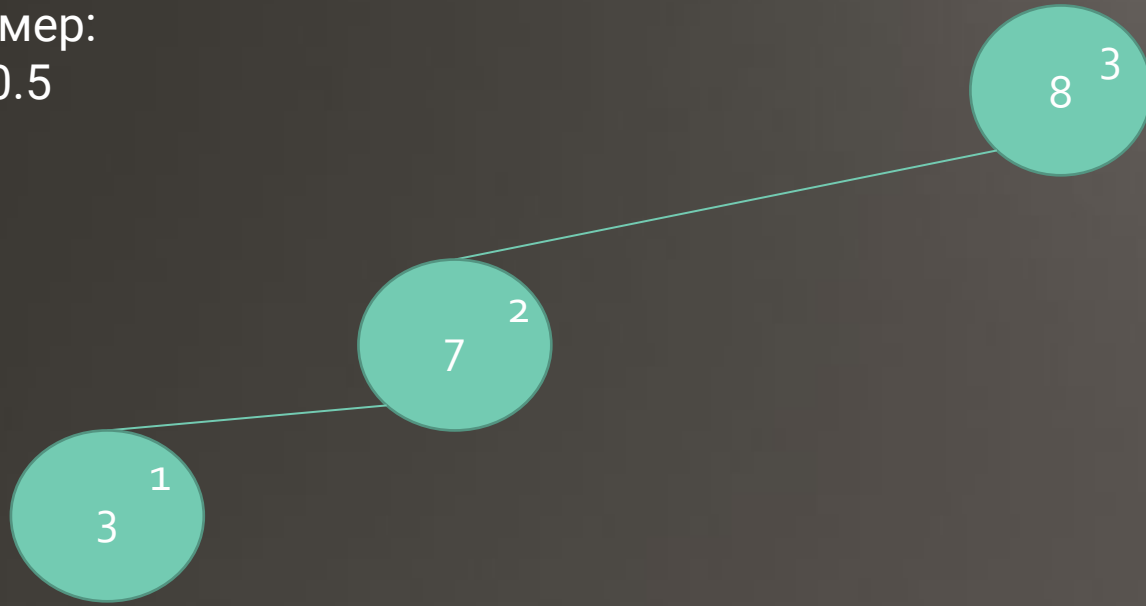


Верхний индекс – вес вершины
(количество всех ее потомков + 1)

- Далее, рекурсивно проходим влево, выполняем ту же функцию, уменьшая параметр ends.
- В левое поддерево подвешиваем 7

Пример:
 $\alpha = 0.5$

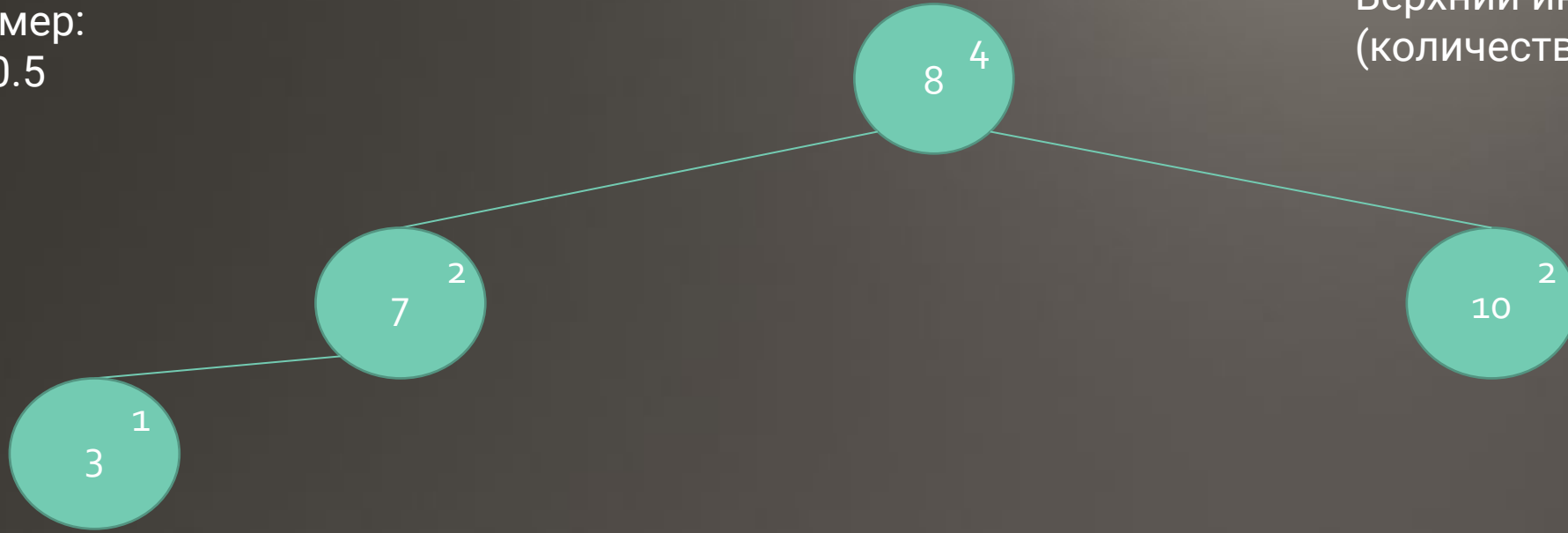
Верхний индекс – вес вершины
(количество всех ее потомков + 1)



- Далее, рекурсивно проходим влево, выполняем ту же функцию, уменьшая параметр ends.
- В левое поддерево подвешиваем 3

Пример:
 $\alpha = 0.5$

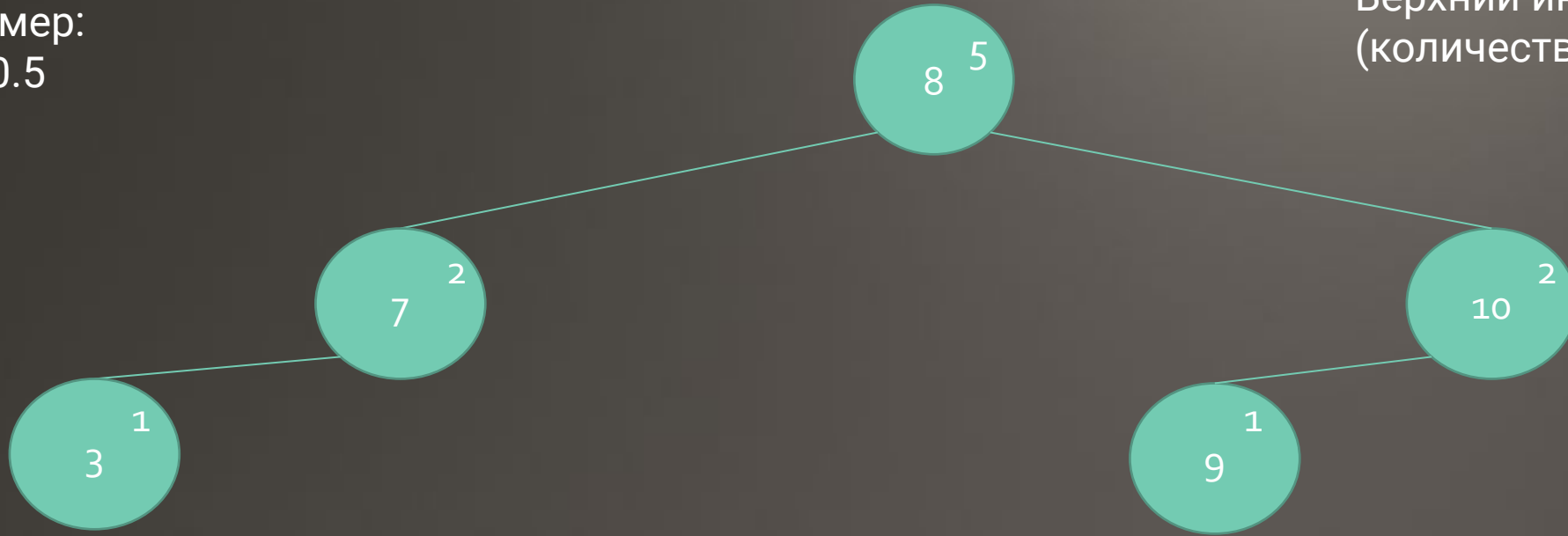
Верхний индекс – вес вершины
(количество всех ее потомков + 1)



- Далее, рекурсивно проходим вправо, выполняем ту же функцию, увеличивая параметр ends.
- В правое поддерево подвешиваем 10

Пример:
 $\alpha = 0.5$

Верхний индекс – вес вершины
(количество всех ее потомков + 1)



- Далее, рекурсивно проходим вправо, выполняем ту же функцию, увеличивая параметр ends.
- В левое поддерево вершины 10 подвешиваем 9
- Выходим из рекурсии
- Дерево имеет α -балансировку

Удаление

Удаляем вершину обычным удалением вершины бинарного дерева поиска (поиск, удаление, возможное переподвешивание детей).

Далее проверяем выполнение условия

$$\text{size}[T] < \alpha * \text{max_size}[T]$$

Если оно выполняется — дерево могло потерять α -балансировку по весу, а значит нужно выполнить полную перебалансировку дерева (начиная с корня) и присвоить $\text{max_size}[T] = \text{size}[T]$