Федеральное государственное автономное
образовательное учреждение
высшего профессионального образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики

ПРАКТИЧЕСКАЯ РАБОТА

Дерево «Козел отпущения» (Scapegoat tree)

Руководитель Б.В. Олейников

Студент ИМ15-06Б, 171509380 А.А. Димов

Красноярск 2016

Содержание

1. Определение структуры

Scapegoat tree (Дерево козёл отпущения) - структура данных, представляющая собой частично сбалансированное дерево поиска (степень балансировки может быть различной), такое что операции поиска, вставки и удаления работают за $O(log\ N)$, при этом скорость одной операции может быть улучшена за счет другой. [1]

2. Авторы

Разработана Arne Andersson, Igal Galperin, Ronald L. Rivest в 1962 году. [3]

3. Описание структуры. Основные свойства.

Существует множество деревьев поиска и главная их идея: при поиске проходить не все данные, а только их часть, в лучшем случае порядка $\log(N)$.

Каждая из вершин хранит ссылки на своих детей и имеет определенный критерий по которому осуществляется обход дерева. Чтобы добиться логарифмического времени работы необходимо, чтобы дерево было сбалансированным (высота каждого из поддеревьев всех примерно одинакова).

3.1 *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.*

Круглые скобки – значение вычисляется в процессе.

Квардратные скобки – значение хранится явно.

- Параметр $a\ (alpha)$ показывает степень несбалансированности дерева.

$$0.5 \leq alpha < 1$$

(1.1)

Рассмотрим некоторые примеры

То есть, например, при $a = 0.5$ в левом и правом поддеревьях будет одинаковое количество вершин ± 1.

При $a \to 1$ дерево будет считаться сбалансированным даже в виде связного списка.

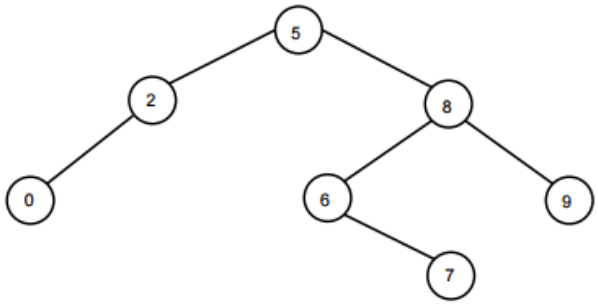

Рисунок 1 - $a = 0.6$

Вершина x называется "a − сбалансированной по весу", если

$$size(left[x]) \leq a * size(x)$$

$$size(right[x]) \leq a * size(x)$$

(1.2)

3.2 *Плюсы и минусы Scapegoat дерева:*

Плюсы:

- Скорость одних операций возможно улучшить за счет других операций. Scapegoat tree работает быстрее, чем красно-черное дерево.
- 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 парметра $alpha$ часто используемые операции будут работать долго, а редко используемые – быстро. При этом дерево будет уступать остальным по скорости.

4. Области назаначения (использования) структуры

- Базы данных
- Случаи, когда необходима компактная отсортированная структура.

5. Реализуемые операции

Дерево, которое a − сбалансированное по весу должно быть так же a − сбалансировано по высоту, т. е.

$$height(T) \leq \log_{\frac{1}{a}}(NodeCount) + 1$$

(1.3)

5.1 *Поиск:*

Данная операция стандратна для бинарного дерева поиска. Необходимо пройти от корня, сравнивая каждую вершину с искомым значением, если найдено – возврат значения, иначе, если значение в вершине меньше, то начинаем рекурсивный поиск в левом поддереве, если больше – в правом.

Сложность операции зависит от параметра $alpha$:

$$O(\log_{\frac{1}{a}} N)$$

(1.4)

При $alpha$ равным/близком к 0.5 мы получим двоичный логарифм, что подрузумевает идеальную или практически идеальную скорость поиска.

5.2 *Вставка:*

На вход подается новый элемент, мы ищем место, где будет находится новая вершина. После вставки новой вершины могла нарушится $a -$ балансировка по весу, тогда мы будем искать scapegoat-вершину и перестраивать дерево. Сама вершина не может стать scapegoat, необходимо пройти по дереву, пересчитывая веса для каждой веришны, если нашли какой элемент нарушил баланс, то перебалансируем ее поддерево.

Вес вершины вычисляется по формуле:

$$sizes[i] = sizes[i-1] + sizeOf(brotherOf(parents[i-1], parents[i])) + 1$$

(1.5)

Если существует несколько вершин, для которых нарушился баланс, то мы можем выбрать любую.

5.3 *Перебалансировка:*

Существует два варианта перебалансировки:

5.3.1 *Первый способ:*

1) Обходим дерево, посредством in-order обхода и заполняем list вершинами, в нашем случае это процедура flatten.
2) Теперь начинается сама перебалансировка
   a) Вычисляем медиану:

$$ceil((start + ends) / 2.0)$$

(1.6)

(start изначально равен 0, ends равен весу вершины)

   b) В нашем list находим эту вершину и подвешиваем в качестве корня поддерева.
3) Рекурсивно выполняем операцию 2

5.3.2 *Второй способ:*

С помощью этого варианта можно сэкономить память.

Мы вычисляем медиану, подвешиваем корень, и дерево однозначно делится на два поддерева. На каждом шаге все будет выбираться однозначно, для списка вершин, отсортированных в возрастающем порядке, у нас будет созданное алгоритмом дерево. Для каждой вершины из списка будет существовать одно место в дереве. Однако мы удаляем еще не просмотренную вершину, значит ее необходимо хранить, мы выделим для нее память, но пямяти нужно будет не $O(size(N))$, а $O(log\ N)$, что естественно лучше.

5.4 *Удаление:*

Удаление происходит как и в обычном бинарном дереве поиска. Необходимо проверить дерево на сбалансированность:

$$size\ <\ (a\ *max\_size) \tag{1.7}$$

если данное условие выполяется, то мы выполняем перебалансировку и присваем

$$max\_size\ =\ size \tag{1.8}$$

6. Тестовые примеры

```
Enter alpha: 0.5
1: Add element. 2: Print tree. 3
Enter: 1
7
Enter: 1
3
Enter: 1
8
Enter: 1
10
Enter: 1
9
Enter: 2
      3
   7
8
      9
  10

Enter: 3
Some of the node to delete?: 3
Deleted
Enter: 2
   7
8
      9
  10
```

До введения значения 9 данное дерево является сбалансированным по весу, однако, после баланс нарушается, потому что

$$2 \leq 0.5 * 3$$

Неверно, поэтому мы перестраиваем дерево. Поэтому, 8 становится корнем данного дерева.

```
Enter alpha: 0.6
1: Add element. 2: Print tree. 3
Enter: 1
6
Enter: 1
2
Enter: 1
1
Enter: 1
0
Enter: 2
      0
   1
2
   6

Enter: 3
Some of the node to delete?: 0
Deleted
Enter: 2
   1
2
   6

Enter: 4
What the node want to find?: 2
Found: 2
```

Баланс по весу нарушается при $2 \leq 3 * 0.6$

Баланс нарушился при

$$2 \leq 3 * 0.5$$

```
Enter alpha: 0.5
1: Add element. 2: Print tree.
Enter: 1
5
Enter: 1
6
Enter: 1
4
Enter: 1
7
Enter:
1
2
Enter: 2
      2
   4
5
   6
      7

Enter: 1
8
Enter: 2
      2
   4
5
      6
   7
      8
```

7. Список использованных источников

[1] Электронные конспекты ИТМО (Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики) [Электронный ресурс] : Scapegoat Tree – Режим Доступа: http://neerc.ifmo.ru/wiki

[2] Коллективный блог Habrahabr [Электронный ресурс] : Scapegoat-деревья – Режим Доступа: http://habrahabr.ru

[3] Wikipedia, the free encyclopedia[Электронный ресурс]: Scapegoat tree – Режим Доступа: https://en.wikipedia.org/wiki/Scapegoat_tree

[4] Open Data Structures. An open content textbook [Электронный ресурс]: Scapegoat tree – Режим Доступа: http://opendatastructures.org/ods-java/8_Scapegoat_Trees.html

[5] Phuong Dao A STUDY AND AN IMPLEMENTATION OF SCAPEGOAT TREES – Оттава, 2006