1 Бинарное дерево поиска

В каждом узле бинарного дерева поиска хранятся *ключ* a и два поддерева, правое и левое. Все ключи в левом поддереве не превосходят a, а в правом — не меньше a. Алгоритм поиска — начиная с корня, сравниваем искомый ключ с ключом в узле, в зависимости от сравнения спускаемся в правое или в левое поддерево.

Вставка в бинарное дерево — поиск + вставляем туда, куда пришёл поиск. Чтобы удалить элемент — ставим на его место самый левый элемент в его правом поддереве.

Проблема такой наивной структуры — может вместо дерева получиться палка (если, например, ключи приходят в порядке по убыванию), и поиск будет занимать $\mathcal{O}(n)$. Красно-чёрные деревья, например, следят за тем, чтобы дерево всегда имело высоту $\mathcal{O}(\log n)$.

Хотим научиться поддерживать ±баланс, не храня много дополнительной информации (такой, как атрибуты red/black) — в идеале, $\mathcal{O}(1)$ дополнительных данных, какие-нибудь несколько чисел про дерево в целом. Это умеют две структуры.

2 Scapegoat tree

Источники: [GR93; And89]. Мы в основном опираемся на [GR93].

Зафиксируем константу $\frac{1}{2} \leq \alpha < 1$. Будем рассматривать структуру данных, в которой храниться дерево tree. Также будем хранить текущее количество узлов в дереве — size. У каждого узла node есть родитель parent, дети left, right и ключ key.

Желаемая максимальная высота дерева (n — количество узлов с ключами) —

$$\left\lfloor \log_{\frac{1}{\alpha}} n \right\rfloor + 1 = H + 1.$$

Если $\alpha = \frac{1}{2}$, то результатом будет идеально сбалансированное дерево, то есть α — это, грубо говоря, разрешённое отклонение размера поддеревьев от состояния баланса.

Узел называется *глубоким,* если его глубина не меньше H. Свойство дерева, которое нам хотелось бы поддерживать —

$$\mathrm{size}(x.\mathrm{left}) \leq \alpha \cdot \mathrm{size}(x.\mathrm{right}).$$

Дерево с таким свойством называется α-weight balanced. Если weight balanced, то и α-height balanced (то есть высота не превосходит H). Обратного следствия нет, потому что может быть «один сын справа, а слева сбалансированное поддерево».

Поиск в α-сбалансированном дереве занимает $\log n$ времени — это очевидно, если инвариант выдержан. Вставка и удаление требуют частичной перестройки дерева.

Чтобы реализовать вставку и удаление, нам также потребуется хранить величину maxSize для всего дерева tree. maxSize — штука, которая не больше, чем максимальный размер дерева с данной идеальной высотой. (То есть, кроме собственно дерева с ключами, мы храним дополнительно только size и maxSize — два числа.)

Инвариант: $\alpha \cdot \mathrm{maxSize} \leq \mathrm{size}(T) \leq \mathrm{maxSize}$ — это про размер всего единого дерева.

Удаление: просто удаляем. Проверяем, не нарушился ли инвариант. Если нарушился — просто перестроим всё дерево с нуля, сделав массив с ключами за линию и соорудив из него идеально сбалансированное дерево. size при этом уменьшается на 1, а maxSize = size.

Вставка: сначала стандартная вставка, добавляем ключ в лист. При этом size увеличивается на 1,

$$\text{maxSize} \coloneqq \max(\text{maxSize}, \text{size}).$$

Может, однако, оказаться так, что новый узел x оказался глубоким (на максимально разрешённой глубине $H+1$). Тогда рассмотрим путь от x до корня $a_1 \dots a_{H+1}$ и найдём среди этих узлов (просто за линию, посчитав количество) самый нижний, не сбалансированный по весу (такой найдётся, докажем) и перестраиваем (глупо, за линию) дерево под ним.

Theorem 1. *Среди $a_1 \dots a_{H+1}$ всегда найдётся узел, не сбалансированный по весу (козёл отпущения).*

Доказательство. Пусть нет, тогда $\text{size}(a_i) \le \alpha \cdot \text{size}(a_{i+1})$. Тогда $\text{size}(x) \le \alpha^H \cdot \text{size}(T)$. Прологарифмируем это неравенство по основанию $\frac{1}{n}$:

$$0 \le -H + \log_{\frac{1}{\alpha}} n$$

□

Theorem 2. *При удалении элемента сохраняется сбалансированность по высоте.*

3 Splay tree

Оригинальная статья: [ST85]

Список литературы

[And89] Arne Andersson. «Improving partial rebuilding by using simple balance criteria». В: *Workshop on Algorithms and Data Structures*. Springer. 1989, с. 393—402.

[GR93] Igal Galperin и Ronald L Rivest. «Scapegoat Trees.» В: *SODA*. Т. 93. 1993, с. 165—174.

[ST85] Daniel Dominic Sleator и Robert Endre Tarjan. «Self-Adjusting Binary Search Trees». В: *J. ACM* 32.3 (июль 1985), с. 652—686. ISSN: 0004-5411. DOI: `10.1145/3828.3835`. URL: `https://doi.org/10.1145/3828.3835`.