winger 12 августа 2009 в 21:01

# Структуры данных: бинарные деревья. Часть 2: обзор сбалансированных деревьев

Алгоритмы

Первая статья цикла

## Интро

Во второй статье я приведу обзор характеристик различных сбалансированных деревьев. Под характеристикой я подразумеваю основной принцип работы (без описания реализации операций), скорость работы и дополнительный расход памяти по сравнению с несбаланчированным деревом, различные интересные факты, а так же ссылки на дополнительные материалы.

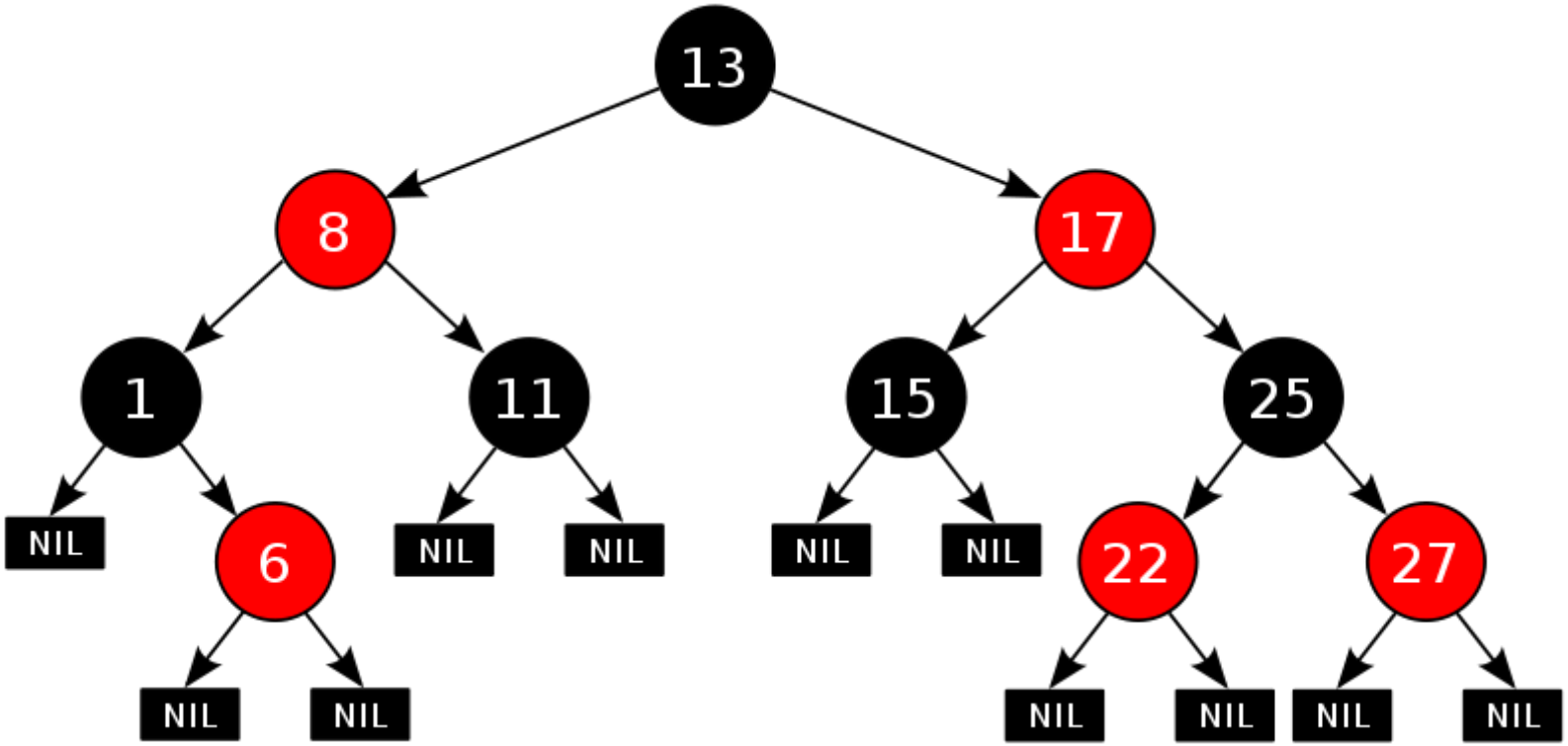
## Красно-черное дерево

Другие названия: Red-black tree, RB tree.
В этой структуре баланс достигается за счет поддержания раскраски вершин в два цвета (красный и черный, как видно из названия :), подчиняющейся следующим правилам:

1. Красная вершина не может быть сыном красной вершины
2. Черная глубина любого листа одинакова (черной глубиной называют количество черных вершин на пути из корня)
3. Корень дерева черный

Здесь мы несколько меняем определение листа, и называем так специальные null-вершины, которые замещают отсутствующих сыновей. Будем считать такие вершины черными.

Пример:




Давайте посмотрим, какой может быть максимальная глубина корректного красно-черного дерева с n вершинами.
Возьмем самый глубокий лист. Пусть он находится на глубине h. Из-за правила 1, как минимум половина вершин на пути из корня будет черными, то есть черная высота дерева будет не меньше h/2. Можно показать, что в таком дереве будет не менее 2^(h/2)-1 черных вершин (так как у каждой черной вершины с черной глубиной k, если она не лист, должно быть как минимум два потомка с черной глубиной k+1). Тогда 2^(h/2)-1 <= n или h <= 2*log2(n+1).

Все основные операции с красно-черным деревом можно реализовать за O(h), то есть O(log n) по доказанному выше.

Классическая реализация основана на разборе большого количества случаев и довольно трудна для восприятия. Существуют более простые и понятные варианты, например в статье Криса Окасаки. К сожалению, в ней описана только операция вставки в дерево. Простота по сравнению с классической реализацией получается за счет ориентации на понятность, а не на оптимизацию количества элементарных модификаций дерева (вращений).

Для реализации этого вида сбаласированных деревьев, нужно в каждой вершине хранить дополнительно 1 бит информации (цвет). Иногда это вызывает большой overhead из-за выравнивания. В таких случаях предпочтительно использовать структуры без дополнительных требований к памяти.

Красно-черные деревья широко используются — реализация set/map в стандартных библиотеках, различные применения в ядре Linux (для организации очередей запросов, в ext3 etc.), вероятно во многих других системах для аналогичных нужд.

Красно-черные деревья тесно связаны с B-деревьями. Можно сказать, что они идентичны B-деревьям порядка 4 (или 2-3-4 деревьям). Более подробно об этом можно прочитать в статье на википедии или в книге «Алгоритмы: построение и анализ», упомянутой в прошлой статье.

Статья в википедии
Статья в английской википедии (с описанием операций)
визуализатор красно-черных деревьев

## AA-дерево

Модификация красно-черного дерева, в которой накладывается дополнительное ограничение: красная вершина может быть только правым сыном. Если красно-черное дерово изоморфно 2-3-4 дереву, то AA-дерево изоморфно 2-3 дереву.

Из-за дополнительного ограничения операции реализуются проще чем у красно-черного дерева (за счет уменьшения количества разбираемых случаев). Оценка на высоту деревьев остается прежней, 2*log2(n). Эффективность по времени у них примерно одинаковая, но так как в реализации вместо цвета обычно хранят другую характеристику («уровень» вершины), overhead по памяти достигает байта.

Статья в английской википедии

## АВЛ-дерево

Названо так по фамилиям придумавших его советских математиков: Г.М. Адельсон-Вельского и Е.М. Ландиса.

Накладывает на дерево следующее ограничение: у любой вершины высоты левого и правого поддеревьев должны отличаться не более чем на 1. Легко доказать по индукции, что дерево с высотой h должно содержать как минимум F_h вершин, где F_i — i-ое число Фибоначчи. Так как F_i ~ phi^n (phi=(sqrt(5)+1)/2 — золотое сечение), высота дерева с n вершинами не может превысить log2(n)/log2(phi) ~ 1.44*log2(n)

Реализация, как и у красно-черного дерева, основана на разборе случаев и достаточно сложна для понимания (хотя имхо проще красно-черного) и имеет сложность O(log(n)) на все основные операции. Для работы необходимо хранить в каждой вершине разницу между высотами левого и правого поддеревьев. Так как она не превосходит 1, достаточно использовать 2 бита на вершину.

Подробное описание можно найти в книге Н. Вирта «Алгоритмы + структуры данных = программы» или в книге А. Шеня «Программирование: теоремы и задачи»

Статья в википедии

## Декартово дерево

Другие названия: Cartesian tree, treap (tree+heap), дуча (дерево+куча).

Если рисовать дерево на плоскости, ключ будет соответствовать x-координате вершины (за счет упорядоченности). Тогда можно ввести и y-координату (назавем ее высотой), которая будет обладать следующим свойством: высота вершины больше высоты детей (такое же свойство имеют значения в другой структуре данных на основе двоичных деревьев — куче (heap). Отсюда второй

вариант названия той структуры)

Оказывается, если высоты выбирать случайным образом, высота дерева, удовлетворяющего свойству кучи наиболее вероятно будет O(log(n)). Численные эксперименты показывают, что высота получается примерно 3*log(n).

Реализация операций проста и логична, за счет этого структура очень любима в спортивном программировании). По результатам тестирования, признана наиболее эффективной по времени (среди красно-черных, AA и АВЛ — деревьев, а так же skip-list'ов (структура, не являющаяся двоичным деревом, но с аналогичной областью применения) и radix-деревьев). К сожалению, обладает достаточно большим overheadom по памяти (2-4 байта на вершину, на хранение высоты) и неприминима там, где требуется гарантированная производительность (например в ядре ОС).

## Splay-дерево

Эта структура данных сильно отличается от всех перечисленных до этого. Дело в том, что оно не накладывает никаких ограничений на структуру дерева. Более того, в процессе работы дерево может оказаться полностью разбалансированным!

Основа splay-дерева — операция splay. Она находит нужную вершину (или ближайшую к ней при отсутствии) и «вытягивает» ее в корень особой последовательностью элементарных вращений (локальная операция над деревом, сохраняющая свойство порядка, но меняющая структуру). Через нее можно легко выразить все оснавные операции с деревом. Последовательность операций в splay подобрана так, чтобы дерево «магически» работало быстро.

Зная магию операции splay, эти деревья реализуются не легко, а очень легко, поэтому они тоже очень популярны в ACM ICPC, Topcoder etc.

Ясно, что в таком дереве нельзя гарантировать сложность операций O(log(n)) (вдруг нас попросят найти глубоко залегшую вершину в несбалансированном на данный момент дереве?). Вместо этого, гарантируется амортизированная сложность операции O(log(n)), то есть любая последовательность из m операций с деревом размера n работает за O((n+m)*log(n)). Более того, splay-дерево обладает некоторыми магическими свойствами, за счет которого оно на практике может оказаться намного эффективнее остальных вариантов. Например, вершины, к которым обращались недавно, оказываются ближе к корню и доступ к

ним ускоряется. Более того, доказано что если вероятности обращения к элементам фиксированы, то splay-дерево будет работать асимптотически не медленней любой другой реализации бинарных деревьев. Еще одно преимущество в том, что отсутствует overhead по памяти, так как не нужно хранить никакой дополнительной информации.

В отличие от других вариантов, операция поиска в дереве модифицирует само дерево, поэтому в случае равномерного обращения к элементам splay-дерево будет работать медленней. Однако на практике оно часто дает ощутимый прирост производительности. Тесты это подтверждают — в тестах, полученных на основе Firefox'a, VMWare и Squid'a, splay-дерево показывает прирост производительности в 1.5-2 раза по сравнению с красно-черными и АВЛ- деревьями. В тоже время, на синтетических тестах splay-деревья работают в 1.5 раза медленней. К сожалению, из-за отсутствия гарантий на производительность отдельных операций, splay-деревья неприминимы в realtime-системах (например в ядре ОС, garbage-collector'ax), а так же в библиотеках общего назначения.

Статья в английской википедии
Оригинальная статья Р. Тарьяна и Д. Слейтора

## Scapegoat-дерево

Это дерево похоже на предыдущее тем, что у него отсутствует overhead по памяти. Однако это дерево является в полной мере сбалансированным. Более того, коэффициент 0 < alpha < 0.5 «жесткости» дерева можно задавать произвольно и высота дерева будет ограничена сверху значением k*log(n)+1, где k=log2(1/alpha). К сожалению, операции модификации будут амортизированными как и у прошлого дерева.

Коэффициент жесткости сильно влияет на баланс производительности: чем «жестче» дерево, тем меньше у него будет высота и тем быстрее будет работать поиск, но тем сложнее будет поддерживать порядок в операциях модификации. Например, так как АВЛ-дерево «жестче» красно-черного, поиск в нем работает быстрее, а модификация медленней. Если же пользоваться scapegoat-деревом, баланс между этими операциями можно выбирать в зависимости от специфики применения дерева.

Статья в английской википедии

## Еще пара слов

Два последних дерева сильно отличаются от своих конкурентов. Например, только они могут использоваться в эффективной реализации структуры данных link/cut tree, использующейся в основе наиболее быстрого известного алгоритма поиска потока в графе. С другой стороны из-за их амортизационной сути они не могут использоваться во многих алгоритмах, в частности для построения ropes. Свойства этих деревьев, особенно splay-дерева, в настоящее время активно изучаются теоретиками.

Кроме сбалансированных деревьев, можно использовать следующий трюк: реализовать обычное бинарное дерево и в процессе работы периодически делать ребалансировку. Для этого существует несколько алгоритмов, например DSW algorithm, работающий за O(n)

## В следующей серии

Я расскажу более подробно про декартовы деревья и их реализацию

## Общие ссылки

визуализатор деревьев (умеет визуализировать все деревья из обзора)

**Теги:** структуры данных, алгоритмы, двоичные деревья