АВЛ-дерево

АВЛ-дерево (англ. *AVL-Tree*) — сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ-деревья названы по первым буквам фамилий их изобретателей, Г. М. Адельсона-Вельского и Е. М. Ландиса, которые впервые предложили использовать АВЛ-деревья в 1962 году.

Содержание

Высота дерева

Теорема:

АВЛ-дерево с n ключами имеет высоту $h = O(\log N)$.

Доказательство:
▷

Высоту поддерева с корнем x будем обозначать как $h(x)$, высоту поддерева T — как $h(T)$.

Лемма:
Пусть m_h — минимальное число вершин в AVL-дереве высоты h, тогда $m_h = F_{h+2} - 1$, где $F_h - h$-ое число Фибоначчи.
Доказательство:
▷

Если m_h — минимальное число вершин в AVL-дереве высоты h. Тогда, как легко видеть, $m_{h+2} = m_{h+1} + m_h + 1$. Равенство $m_h = F_{h+2} - 1$ докажем по индукции.

База индукции $m_1 = F_3 - 1$ — верно, $m_1 = 1, F_3 = 2$.

Допустим $m_h = F_{h+2} - 1$ — верно.

Тогда $m_{h+1} = m_h + m_{h-1} + 1 = F_{h+2} - 1 + F_{h+1} - 1 + 1 = F_{h+3} - 1$.

Таким образом, равенство $m_h = F_{h+2} - 1$ — доказано.
◁

$F_h = \Omega(\varphi^h), \varphi = \frac{\sqrt{5}+1}{2}$. То есть

$n \geqslant \varphi^h$

Логарифмируя по основанию φ, получаем

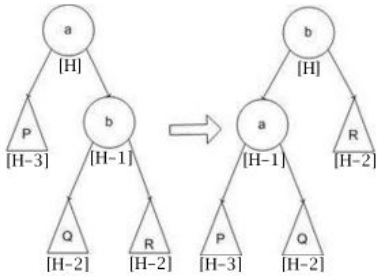
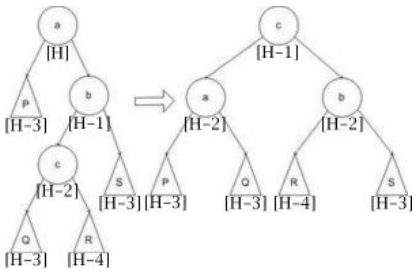
$\log_\varphi n \geqslant h$

Таким образом, получаем, что высота AVL-дерева из n вершин — $O(\log n)$.
◁

Балансировка

Балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев $|h(L) - h(R)| = 2$, изменяет связи предок-потомок в поддереве данной вершины так, чтобы восстановилось свойство дерева $|h(L) - h(R)| \leqslant 1$, иначе ничего не меняет. Для балансировки будем хранить для каждой вершины разницу между высотой её левого и правого поддерева $diff[i] = h(L) - h(R)$

Для балансировки вершины используются один из 4 типов вращений:

Тип вращения	Иллюстрация	Когда используется	Расстановка балансов
Малое левое вращение		$diff[a] = -2$ и $diff[b] = -1$ или $diff[a] = -2$ и $diff[b] = 0$.	$diff[a] = 0$ и $diff[b] = 0$ $diff[a] = -1$ и $diff[b] = 1$
Большое левое вращение		$diff[a] = -2$, $diff[b] = 1$ и $diff[c] = 1$ или $diff[a] = -2, diff[b] = 1$ и $diff[c] = -1$ или $diff[a] = -2, diff[b] = 1$ и $diff[c] = 0$.	$diff[a] = 0, diff[b] = -1$ и $diff[c] = 0$ $diff[a] = 1, diff[b] = 0$ и $diff[c] = 0$ $diff[a] = 0, diff[b] = 0$ и $diff[c] = 0$

Малый левый поворот:

```
function rotateLeft(Node a):
  Node b = a.right
  a.right = b.left
  b.left = a
  корректировка высоты a
  корректировка высоты b
```

Большой левый поворот пишется проще:

```
function bigRotateLeft(Node a):
  rotateRight(a.right)
  rotateLeft(a)
```

Малое правое и большое правое вращение определяются симметрично малому левому и большому левому вращению. В каждом случае операция приводит к нужному результату, а полная высота уменьшается не более чем на 1 и не может увеличиться.

Все вращения, очевидно, требуют $O(1)$ операций.

Операции

Добавление вершины

Пусть нам надо добавить ключ t. Будем спускаться по дереву, как при поиске ключа t. Если мы стоим в вершине a и нам надо идти в поддерево, которого нет, то делаем ключ t листом, а вершину a его корнем. Дальше поднимаемся вверх по пути поиска и пересчитываем баланс у вершин. Если мы поднялись в вершину i из левого поддерева, то $diff[i]$ увеличивается на единицу, если из правого, то уменьшается на единицу. Если пришли в вершину и её баланс стал равным нулю, то это значит высота поддерева не изменилась и подъём останавливается. Если пришли в вершину и её баланс стал равным 1 или -1, то это значит высота поддерева изменилась и подъём продолжается. Если пришли в вершину и её баланс стал равным 2 или -2, то делаем одно из четырёх вращений и, если после вращения баланс стал равным нулю, то останавливаемся, иначе продолжаем подъём.

Так как в процессе добавления вершины мы рассматриваем не более, чем $O(h)$ вершин дерева, и для каждой запускаем балансировку не более одного раза, то суммарное количество операций при включении новой вершины в дерево составляет $O(\log n)$ операций.

Удаление вершины

Для простоты опишем рекурсивный алгоритм удаления. Если вершина — лист, то удалим её, иначе найдём самую близкую по значению вершину a, переместим её на место удаляемой вершины и удалим вершину a. От удалённой вершины будем подниматься вверх к корню и пересчитывать баланс у вершин. Если мы поднялись в вершину i из левого поддерева, то $diff[i]$ уменьшается на единицу, если из правого, то увеличивается на единицу. Если пришли в вершину и её баланс стал равным 1 или -1, то это значит, что высота этого поддерева не изменилась и подъём можно остановить. Если баланс вершины стал равным нулю, то высота поддерева уменьшилась и подъём нужно продолжить. Если баланс стал равным 2 или -2, следует выполнить одно из четырёх вращений и, если после вращений баланс вершины стал равным нулю, то подъём продолжается, иначе останавливается.

В результате указанных действий на удаление вершины и балансировку суммарно тратится, как и ранее, $O(h)$ операций. Таким образом, требуемое количество действий — $O(\log n)$.

Поиск вершины, минимум/максимум в дереве, etc.

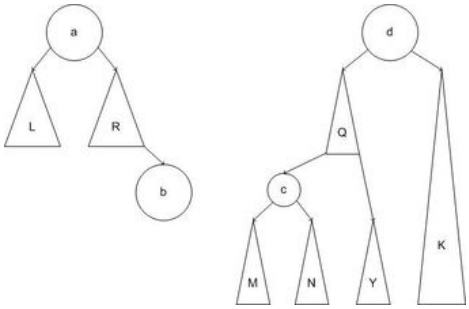
Остальные операции не меняют структуры дерева, поэтому выполняются так же, как и в наивной реализации дерева поиска.

Слияние двух AVL-деревьев

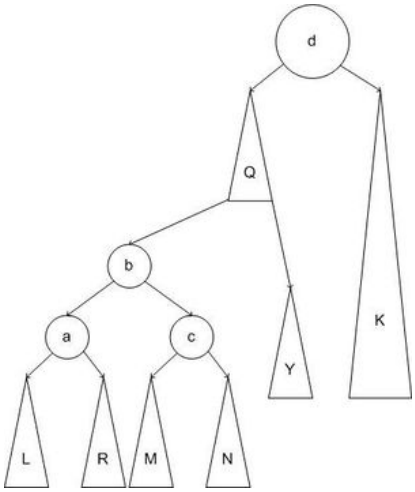
Дано два дерева T_1 и T_2, все ключи в T_1 меньше ключей в T_2, $h(T_1) \leqslant h(T_2)$.

В дереве T_1 удаляем самую правую вершину, назовём её b. Высота дерева T_1 может уменьшиться на единицу. В дереве T_2 идём от корня всегда в левое поддерево и, когда высота этого поддерева P будет равна высоте дерева T_1, делаем новое дерево S, корнем S будет вершина b, левым поддеревом будет дерево T_1, а правым дерево P. Теперь в дереве T_2 у вершины, в которой мы остановились при спуске, левым поддеревом делаем дерево S и запускаем балансировку. Таким образом, дерево T_2 будет результатом слияния двух АВЛ-деревьев.

Дерево T_1 и T_2 до слияния



Дерево T_2 после слияния



Алгоритм разделения AVL-дерева на два

Алгоритм первый

Пусть у нас есть дерево T. Мы должны разбить его на два дерева T_1 и T_2 такие, что $T_1 \leqslant x$ и $x < T_2$.

Предположим, что корень нашего дерева $\leqslant x$, в таком случае все левое поддерево вместе с корнем после разделения отойдет в дерево T_1. Тогда рекурсивно спускаемся в правое поддерево и там проверяем это условие (так как часть правого поддерева тоже может содержать ключи $\leqslant x$). Если же корень оказался $> x$, то мы спускаемся той же рекурсией, но только в левое поддерево и ищем там.

Пусть мы пришли в поддерево S, корень которого $\leqslant x$. В таком случае этот корень со своим левым поддеревом должен отойти в дерево T_1. Поэтому мы делаем следующее: запоминаем ссылку на правое поддерево S, удаляем корень, запоминая его значение (не меняя конфигурацию дерева, то есть просто делаем ссылки на него NULL'ами). Таким образом, мы отделяем сбалансированное АВЛ-дерево (бывшее левое поддерево S). Делаем новую вершину со значением

бывшего корня правым листом самой правой вершины S и запускаем балансировку. Обозначим полученное дерево за T'. Теперь нам нужно объединить его с уже построенным ранее T_1 (оно может быть пустым, если мы первый раз нашли такое дерево S). Для этого мы ищем в дереве T_1 самое правое поддерево P высоты, равной высоте T' (спускаясь от корня всегда в правые поддеревья). Делаем новое дерево K, сливая P и T' (очевидно, все ключи в T_1 меньше ключей в T', поэтому мы можем это сделать). Теперь в дереве T_1 у отца вершины, в которой мы остановились при поиске дерева P, правым поддеревом делаем дерево K и запускаем балансировку. После нужно спуститься в правое поддерево бывшего дерева S (по ссылке, которую мы ранее запомнили) и обработать его.

Если мы пришли в поддерево Q, корень которого $> x$, совершаем аналогичные действия: делаем NULL'ами ссылки на корень Q, запоминая ссылку на его левое поддерево. Делаем новую вершину со значением бывшего корня левым листом самой левой вершины Q и запускаем балансировку. Объединяем полученное АВЛ-дерево с уже построенным ранее T_2 аналогичным первому случаю способом, только теперь мы ищем самое левое поддерево T_2.

Рассмотрим пример (рис. 1). Цветом выделены поддеревья, которые после разделения должны отойти в дерево T_1. $x = 76$.

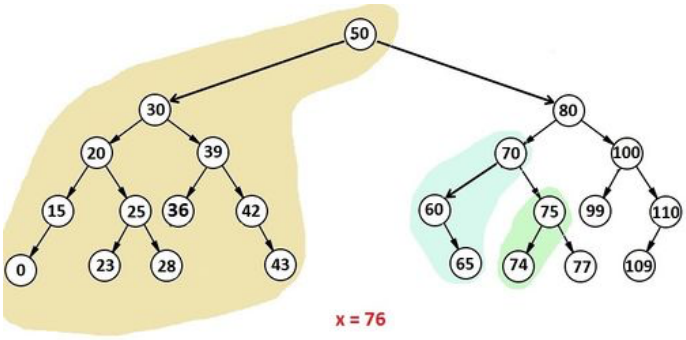


Рис. 1. Разделение АВЛ-дерева на два.

Корень дерева $\leqslant x$, поэтому он со всем выделенным поддеревом должен отойти в дерево T_1. По описанному выше алгоритму отделяем это поддерево с корнем и делаем из них сбалансированное АВЛ-дерево T' (рис. 2). Так как это первая ситуация, в которой корень рассматриваемого поддерева был $\leqslant x$, T' становится T_1. Далее по сохраненной ссылке спускаемся в правое поддерево. Его корень $> x$. Следовательно, строим из него и его правого поддерева T_2 и спускаемся в левое поддерево. Снова корень $\leqslant x$. Строим новое T' и объединяем его с уже существующим T_1 (рис. 3).

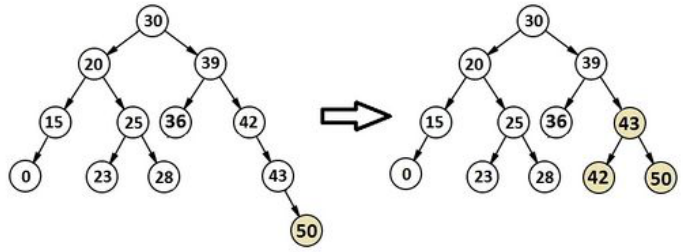


Рис. 2. Создание T'.

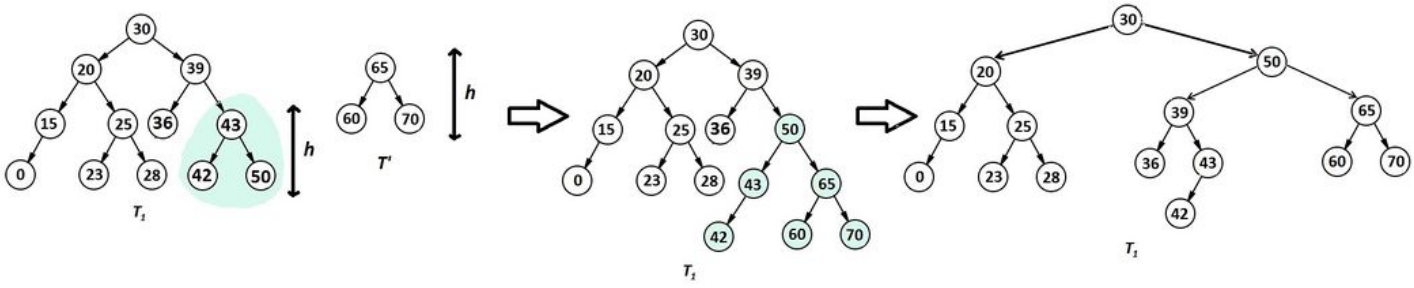


Рис. 3. Объединение T' и T1.

Далее действуем по алгоритму и в итоге получаем (рис. 4):

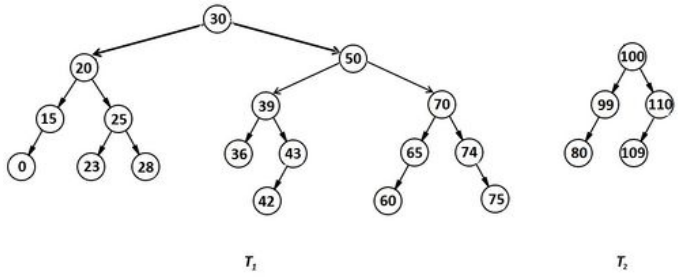


Рис. 4. АВЛ-деревья после разделения.

Данный алгоритм имеет сложность $O(\log^2 n)$.

Алгоритм второй

Рассмотрим решение, которое имеет сложность $O(\log n)$.

Вернемся к примеру (рис. 1). Теперь рекурсивно спустимся вниз и оттуда будем строить деревья T_1 и T_2, передавая наверх корректные АВЛ-деревья. То есть для рис. 1 первым в дерево T_1 придет вершина 75 с левым поддеревом (выделено светло-зеленым цветом), так как это корректное АВЛ-дерево, оно же и вернется из рекурсии. Далее мы попадем в вершину со значением 70 и должны слить ее и ее левое поддерево (выделено светло-синим) с тем, что нам пришло. И сделать это нужно так, чтобы передать наверх корректное АВЛ-дерево. Будем действовать по такому алгоритму, пока не дойдем до вершины.

Пусть мы пришли в поддерево S с корнем $\leqslant x$. Тогда сольем его с уже построенным на тот момент T_1 (T_1 пришло снизу, а значит по условию рекурсии это корректное АВЛ-дерево, $S \leqslant T_1$ и $h(T_1) \leqslant h(S)$). Но так как обычная процедура слияния сливает два АВЛ-дерева, а S не является корректным АВЛ-деревом, мы немного ее изменим. Пусть мы в дереве S нашли самое правое поддерево K, высота которого равна высоте T_1. Тогда сделаем новое дерево T', корнем которого будет вершина S (без нее это дерево является сбалансированным), правым поддеревом — T_1, левым — K. И подвесим T' на то место, где мы остановились при поиске K. Запустим балансировку. В случае, когда корень поддерева, в которое мы пришли, $> x$, все аналогично.

Разберем пример на рис. 1. Пусть мы рекурсивно спустились до узла 77. Ключ больше x, поэтому эта вершина станет деревом T_2 и передастся наверх. Теперь мы поднялись в узел 75. Он со своим левым поддеревом станет деревом T_1 и мы снова поднимемся наверх в узел 70. Он со своим левым поддеревом снова должен отойти в дерево T_1, и так как теперь дерево T_1 уже не пустое, то их надо слить. После слияния по описанному выше алгоритму получим (рис. 5)

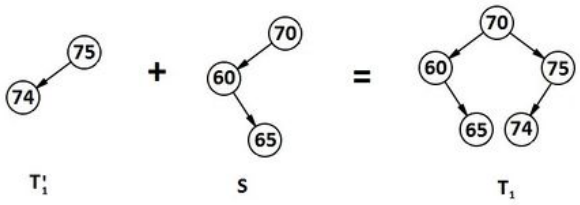


Рис. 5.

После мы поднимемся в вершину с ключом 80. Она с правым поддеревом отойдет в дерево T_2 (рис. 6).

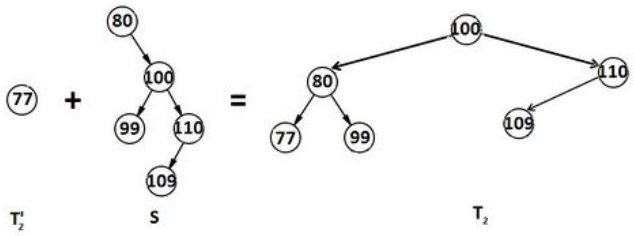


Рис. 6.

И на последней итерации мы поднимемся в корень дерева с ключом 50, он с левым поддеревом отойдет в дерево T_1, после чего алгоритм завершится.

Пусть поддеревьев с ключами $\leqslant x$ оказалось больше, чем поддеревьев с ключами $> x$. Докажем для них логарифмическую асимптотику. Дерево на последнем уровне имеет высоту H_k (она может быть не равна 1, если мы придём в x). Его мы передаем наверх и вставляем в поддерево высотой H_{k-1}. $H_k \leqslant H_{k-1}$, так как разница высот поддеревьев у любой вершины не больше 1, и мы при переходе от H_k к H_{k-1} поднимаемся как минимум на одну вершину вверх. Слияние этих поддеревьев мы выполним за $H_{k-1} - H_k$, получим в итоге дерево высоты не большей, чем H_{k-1}. Его мы передадим наверх, поэтому в следующий раз слияние будет выполнено за $H_{k-2} - H_{k-1}$ и так далее. Таким образом мы получим
$(H - H_1) + (H_1 - H_2) + (H_2 - H_3) + \cdots + (H_{k-1} - H_k) = H - H_k = O(\log n)$.

Итоговая асимптотика алгоритма — $O(\log n)$.

АВЛ-дерево с O(1) бит в каждом узле

Идея

В обычной реализации АВЛ-дерева в каждом узле мы хранили высоту этого узла. Так как высоты левых и правых поддеревьев в АВЛ-дереве отличаются максимум на 1, то мы будем хранить не всю высоту дерева, а некоторое число, которое будет показывать, какое поддерево больше, или равны ли они, назовём *фактор баланса*. Таким образом в каждом узле будет хранится 1 — если высота правого поддерева выше левого, 0 — если высоты равны, и -1 — если правое поддерево выше левого.

Операции

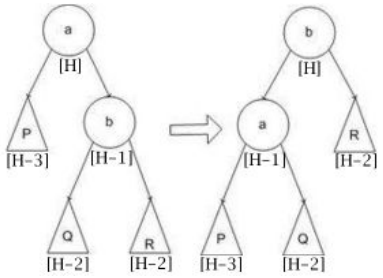
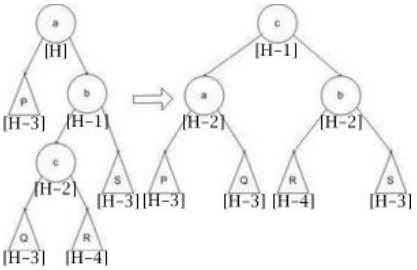
Операция добавления
Пусть нам надо добавить ключ t. Будем спускаться по дереву, как при поиске ключа t. Если мы стоим в вершине a и нам надо идти в поддерево, которого нет, то делаем ключ t листом, а вершину a его корнем. Пересчитываем баланс данного узла a. Если он оказался 0, то высота поддерева с корнем в этой вершине не изменилась и пересчет балансов останавливается. Дальше начинаем подниматься вверх по дереву, исправляя балансы попутных узлов. Если мы поднялись в вершину i из левого поддерева, то баланс увеличивается на единицу, если из правого, то уменьшается на единицу. Если мы пришли в вершину и её баланс стал равным 1 или -1, то это значит, что высота поддерева изменилась и подъём продолжается. Если баланс вершины a, в которую мы собираемся идти из ее левого поддерева, равен 1, то делается поворот для этой вершины a. Аналогично делаем поворот, если баланс вершины a, в которую мы идем из ее правого поддерева, равен -1. Если в результате изменения узла, фактор баланса стал равен нулю, то останавливаемся, иначе продолжаем подъём.

Операция удаления
Если вершина — лист, то просто удалим её, иначе найдём ближайшую по значению вершину a, поменяем ее местами с удаляемой вершиной и удалим. От удалённой вершины будем подниматься вверх к корню и пересчитывать фактор баланса вершин. Если мы поднялись в вершину i из левого поддерева, то фактор баланса уменьшается на единицу, если из правого, то увеличивается на единицу. Если мы пришли в вершину и её баланс стал равным 1 или -1, то это значит, что высота поддерева не изменилась и подъём можно остановить. Если баланс вершины стал равным нулю, то высота поддерева уменьшилась и подъём нужно продолжить. Если баланс вершины a, в которую мы собираемся идти из ее левого поддерева, равен -1, то делается поворот для этой вершины a. Аналогично делаем поворот, если баланс вершины a, в которую мы идем из ее правого поддерева, равен 1. Если в результате изменения узла, фактор баланса стал равен нулю, то подъём продолжается, иначе останавливается.

Балансировка

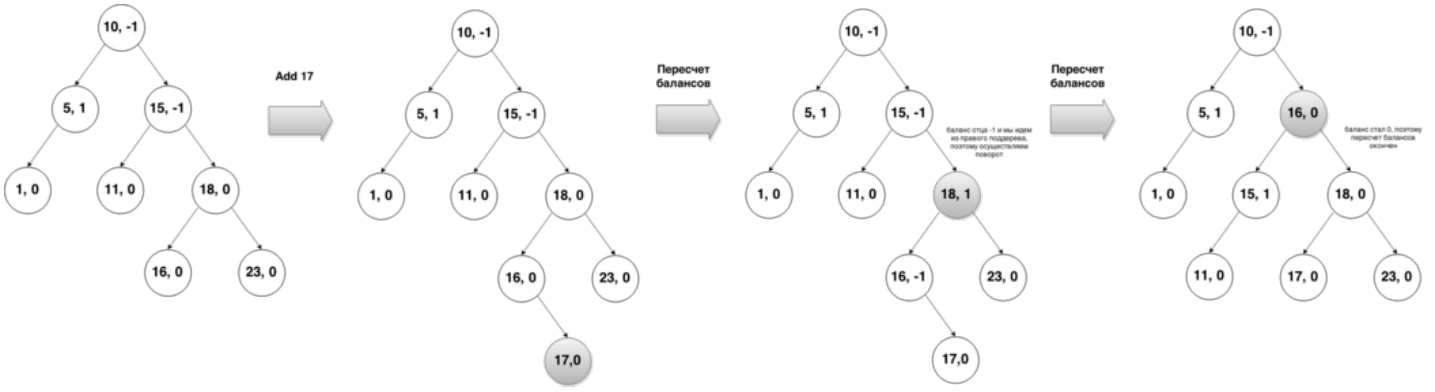
Опишем операции балансировки, а именно малый левый поворот, большой левый поворот и случаи их возникновения. Балансировка нам нужна для операций добавления и удаления узла. Для исправления факторов баланса, достаточно знать факторы баланса двух(в случае большого поворота — трех) вершин перед поворотом, и исправить значения этих же вершин после поворота. Обозначим фактор баланса вершины i как $balance[i]$. Операции поворота делаются на том шаге, когда мы находимся в правом сыне вершины a, если мы производим операцию добавления, и в левом сыне, если мы производим операцию удаления. Вычисления производим заранее, чтобы не допустить значения 2 или -2 в вершине a. На каждой иллюстрации изображен один случай высот поддеревьев. Нетрудно убедиться, что в остальных случаях всё тоже будет корректно.

Тип вращения	Иллюстрация	Факторы балансов до вращения	Факторы балансов после вращения

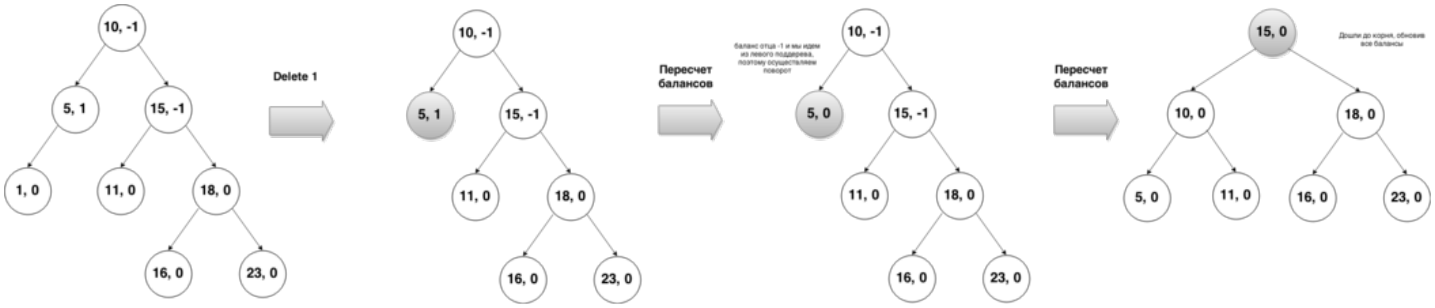
Малое левое вращение		**1 вариант:** $balance[a] = -1$ и $balance[b] = -1$ **2 вариант:** $balance[a] = -1$ и $balance[b] = 0$	**1 вариант:** $balance[a] = 0$ и $balance[b] = 0$ **2 вариант:** $balance[a] = -1$ и $balance[b] = 1$
Большое левое вращение		**1 вариант:** $balance[a] = -1$, $balance[b] = 1$ и $balance[c] = 1$ **2 вариант:** $balance[a] = -1$, $balance[b] = 1$ и $balance[c] = -1$ **3 вариант:** $balance[a] = -1$, $balance[b] = 1$ и $balance[c] = 0$	**1 вариант:** $balance[a] = 0$, $balance[b] = -1$ и $balance[c] = 0$ **2 вариант:** $balance[a] = 1$, $balance[b] = 0$ и $balance[c] = 0$ **3 вариант:** $balance[a] = 0$, $balance[b] = 0$ и $balance[c] = 0$

Примеры

Ниже приведены примеры добавления и удаления вершины с подписанными изменениями факторов баланса каждой вершины.



Добавление



Удаление

См. также

- Splay-дерево
- Красно-черное дерево
- 2-3 дерево

Источники информации

- Habrahabr — АВЛ-деревья (http://habrahabr.ru/post/150732/)
- Википедия — АВЛ-дерево (http://ru.wikipedia.org/wiki/АВЛ-дерево)

- Эта страница последний раз была отредактирована 15 февраля 2018 в 23:12.