Алгоритмы. Домашнее задание №12

Горбунов Егор Алексеевич

10 декабря 2015 г.

Задача №1 (степень одиночества)

$LR(T) = \frac{\textit{Количество единственных детей в } T}{\textit{Количество вершин в } T}$. *Ребёнок единственный, если он один такой у своего родителя. Корень не является единственным ребёнком.*

(a) Покажите, что в любом ненулевом AVL-дереве T $LR(T) \le \frac{1}{2}$

(b) Правда ли, что если $LR(T) \le \frac{1}{2}$, то $h(T) \le \log size(T)$?

(c) Пусть в дереве T есть всего $\Theta(size(T))$ единственных детей и все они — листья. Правда ли, что для любого такого дерева $h(T) \le \log size(T)$?

Решение: Обозначим $LC(T)$ — множество единственных детей в T.

(a) В AVL-дереве на 1 вершине $LR(T) = 0 \le \frac{1}{2}$. На 2 вершинах $LR(T) = \frac{1}{2}$. Пускай утверждение верно для всех AVL-деревьев на $1, 2, \ldots, n-1$ вершине. Рассмотрим AVL-дерево T на $n > 2$ вершинах пусть T_l и T_r — это левое и правое поддерево, подвешенные к корню T. Т.к. высоты T_l и T_r отличаются максимум на 1, а $n > 2$, то оба T_l и T_r — непустые AVL-деревья. Пусть размер T_l — $n_l < n$, а размер T_r — $n_r < n$. Заметим, что индукционное предположение для T_l и T_r верно и $n = n_l + n_r + 1$.

$$LR(T) = \frac{|LC(T_l)| + |LC(T_r)|}{n} = \frac{n_l n_r}{n}\Big(\frac{|LC(T_l)|}{n_l n_r} + \frac{|LC(T_r)|}{n_l n_r}\Big) \le \frac{n_l n_r}{n}\Big(\frac{1}{2n_l} + \frac{1}{2n_r}\Big) =$$
$$= \frac{1}{2}\frac{n_l + n_r}{n} = \frac{1}{2}\frac{n-1}{n} \le \frac{1}{2}$$

Итого по индукции показали, что утверждение (a) верно. ■

(b) Нет, не правда. Рассмотрим дерево T на n вершинах (в данном случае n чётное) изображённое на рисунке 1. Высота этого дерева равна $\frac{n}{2} + 1$, но при этом единственный ребёнок лишь один — это лист n. Таким образом $LR(T) = \frac{1}{n} \le 12$, но при этом высота $h(T) = \frac{n}{2} + 1 > \log n$ ■

(c) Обозначим $n = size(T)$. Рассмотрим дерево изображённое на рисунке 2. В этом дереве $n = 3m + 1$ вершина. Высота этого дерева равна $m + 2$ (m «веточек» влево и ещё 2 уровня под корень и его сына 2). Ясно, что в таком графе одиноких детей ровно m,

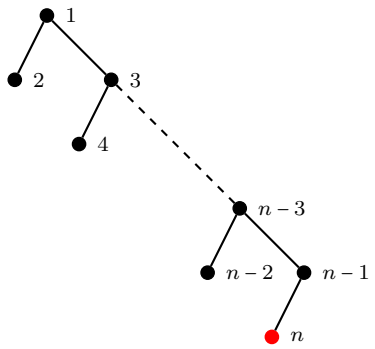

Рис. 1: К задаче 1 (b)

они выделены красным на изображении и каждый из них является листом, т.е. дерево T удовлетворяет условию задачи ($m = \lfloor \frac{n}{3} \rfloor$). Но, как было сказано выше, высота этого дерева равна $m + 2 = \lfloor \frac{n}{3} \rfloor + 2 \geq \log n$ начиная с некоторого n. Таким образом утверждение пункта задачи неверно! ■

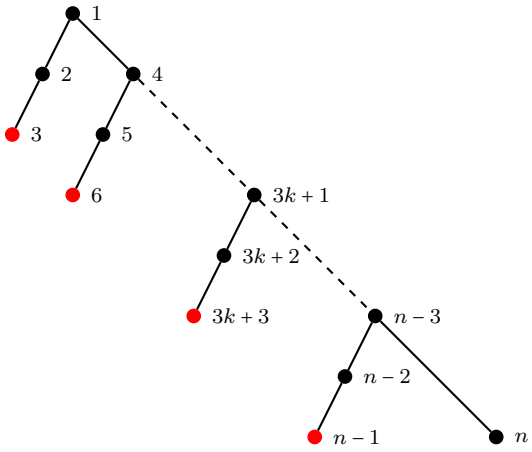

Рис. 2: К задаче 1 (c)

Задача №2 (Слияния AVL-деревьев)

Пусть даны 2 AVL-дерева T_1 и T_2. Придумать, как построить AVL-дерево T, являющееся объединением деревьев T_1 и T_2 за время:

(a) $\mathcal{O}\left(h(T_1) size(T_2)\right)$

(b) $\mathcal{O}\left(\max\{size(T_1), size(T_2)\}\right)$

Решение:

(a) Не умаляя общности $size(T_1) \geq size(T_2)$. Обойдём дерево T_2 за $\mathcal{O}\left(size(T_2)\right)$ и в порядке обхода будем добавлять вершины из T_2 в T_1. Заметим, что размер дерева T_2 увеличится, во время выполнения такой процедуры, не более чем в 2 раза, т.к. $size(T_1) \geq size(T_2)$. Но значит высота дерева может измениться лишь на константу, т.к. в AVL-дереве на n вершинах равна $\Theta(\log n)$, тогда высота дерева на $2n$ вершинах $\Theta(\log 2 \log n) = \Theta(\log n)$. Таким образом на каждую операцию добавления в T_1 уйдёт $\mathcal{O}\left(\log size(T_1)\right) = \mathcal{O}\left(h(T_1)\right)$,

а на всё объединение $\mathcal{O}(h(T_1)size(T_2))$ ■

(b)

Задача №3 (Порядковые статистики)

(a) Придумайте, как в AVL-дереве T реализовать операцию получения k-ой порядковой статистики за $\mathcal{O}(h(T))$

(b) Придумайте, как в AVL-дереве T найти $index(k)$ — позицию ключа k в отсортированном массиве ключей за $\mathcal{O}(h(T))$

(c) Придумайте, как в AVL-дереве T найти количество ключей между k_1 и k_2 за $\mathcal{O}(h(T))$

Решение: Будем в каждой вершине v дерева T дополнительно хранить размер поддерева $v.size$ с корнем в v, включая саму вершину v. Покажем, что размер поддерева легко поддерживать сохраняя асимптотику операций на AVL-дереве. Действительно, в стандартной реализации AVL-дерева мы вынуждены поддерживать в каждой вершине высоту поддерева, аналогично ей будем поддерживать и размер поддерева. При добавлении вершины мы соответственно увеличиваем в каждой вершине, которая была затронута при добавлении (путь до места, куда добавляем), размер поддерева. Т.е. после этой части операции размеры корректны. Далее корректность может нарушаться при балансировке, но она легко чинится из $\mathcal{O}(1)$, т.к. при поворотах, которые используются в балансировке, корректность параметра размера может нарушиться лишь в паре вершин. Причём корректность легко восстановить посчитав размеры заново, исходя из размеров левых и правых поддеревьев вершин (нужно, собственно, их сложить и ещё 1 добавить), в которых она нарушилась. Таким образом мы умеем поддерживать в каждой вершине размер поддерева корнем в этой вершине.

(a) Найдём k-ую порядковую статистику. Т.е. найдём в дереве T вершину v с ключом x таким, что в дереве ровно $k-1$ элемент меньше x. Следующая процедура это делает:

```
procedure kStatistic(T, k)
    cur ← T.root
    numLess ← cur.l.size
    while cur ≠ nil AND numLess ≠ k − 1 do
        if numLess < k − 1 then
            cur ← cur.r
            numLess ← numLess + 1 + cur.l.size
        if numless > k − 1 then
            cur ← cur.l
            numLess ← numLess − 1 − cur.r.size
    return cur.key
```

На каждом шаге приведённой процедуры поддерживается инвариант $numLess$ — число ключей в дереве, которые меньше, чем $cur.key$. Вначале мы стоим в корне $T.root$ и число ключей, меньших $T.root.key$ равно $size(T.root.l)$. Далее $numLess$ легко поддерживать. Пускай мы из cur перешли в $cur.r$, тогда $numLess$ стало больше на число вершин, непосредственно подвешенных слева от $cur.r$, аналогично для перехода в $cur.l$. Ясно, что если в дереве T есть $\geq k$ вершин, которые меньше $T.root.key$, что k-ая порядковая статистика где-то в левом поддереве от корня, иначе она находится в правом поддереве, таким образом оправданы переходы по условиям $numLess < k - 1$ и $numLess > k - 1$. Как только мы находим ключ такой, что в дереве строго $k - 1$ ключ, меньше его, то значит, что мы нашли k-ую порядковую статистику.

На каждом шаге цикла мы делаем один спуск на уровень ниже по дереву. Т.е. цикл отработает максимум за $h(T)$ операций, а значит время работы алгоритмы на AVL-дереве равно $\mathcal{O}(\log size(T))$ ■

(b) Нужно найти $index(k)$. Будем искать переданный ключ k в нашем дереве стандартной процедурой, но только будем поддерживать, аналогично предыдущему пункту, инвариант $numLess$ — число ключей в дереве, строго меньших $cur.key$, где cur — вершина, рассматриваемая в качестве кандидата при поиске в данный момент. Вначале стоим в корне и $numLess$ равен числу ключей в дереве, меньших корня, т.е. $numLess$ есть индекс (если нумерация с 0) $T.root.key$ в отсортированном массиве. Пересчёт $numLess$ при переходах по дереву такой же, как и в предыдущем пункте. Таким образом, как только стандартный алгоритм поиска натыкается на вершину с нужным ключом, так сразу возвращаем $numLess$. Алгоритм отработает за столько же, за сколько работает обычный поиск, т.е. за $\mathcal{O}(h(T)) = \mathcal{O}(\log size(T))$ ■

(c) Ясно, что число элементов между ключом k_1 и k_2 равно $index(k_2) - index(k_1) - 1$, но $index(k)$ мы умеем находить за $\mathcal{O}(\log size(T))$ по предыдущему пункту. ■