ОРГАНИЗАЦИЯ ПОИСКА

СБАЛАНСИРОВАННЫЕ ПОИСКОВЫЕ ДЕРЕВЬЯ

Одним из *инвариантов* сбалансированных поисковых деревьев является свойство сбалансированности по высоте.

Определение. Корневое дерево *k-сбалансировано по высоте*, если для каждой ее вершины высоты поддеревьев отличаются не более, чем на k. Если $k = 1$, то говорят, что дерево *сбалансировано*.

Определение. Корневое дерево *k-идеально сбалансировано по количеству вершин*, если для каждой ее вершины количество вершин в ее поддеревьях отличаются не более, чем на k. Если $k = 1$, то говорят, что дерево *идеально сбалансировано*.

Любое идеально сбалансированное дерево является сбалансированным, а вот обратное верно не всегда. Примером сбалансированного дерева является полное бинарное дерево, которое мы рассматривали при работе с бинарной кучей.

АВЛ-дерево

АВЛ-дерево – это бинарное поисковое дерево, которое является сблансированным . Т.е. это такое бинарное поисковое дерево, у которого для каждой вершины *v* выполняется следующее свойство: высота поддерева, корень которого – левый сын вершины *v*, отличается не более чем на единицу от высоты поддерева, корень которого – правый сын вершины v.

АВЛ — аббревиатура, образованная первыми буквами фамилий создателей Г. М. **А**дельсон-**В**ельского и Е.М.**Л**андиса (советских учёных), которые в 1962 предложили такую структуру данных.

https://ru.wikipedia.org/wiki/Адельсон-Вельский,_Георгий_Максимович
https://ru.wikipedia.org/wiki/Ландис,_Евгений_Михайлович

ТЕОРЕМА. Пусть n – число внутренних вершин АВЛ-дерева, а h – его высота. Тогда справедливы следующие неравенства:

$$\log(n+1) \le h < 1{,}4404 \cdot \log(n+2) - 0{,}328. \qquad (1)$$

Доказательство. Поскольку АВЛ-дерево является бинарным деревом, то оно не может содержать более чем $(2^h - 1)$ внутренних вершин.

Поэтому

$$n \le 2^h - 1;\ h \ge \log(n+1),$$

и мы получаем левую часть неравенства (1).

Определим теперь минимальное количество внутренних вершин для АВЛ-дерева высотой h.

Пусть T_h – АВЛ-дерево высотой h с минимальным числом внутренних вершин. Поскольку принцип построения деревьев напоминает построение чисел Фибоначчи, то такие деревья обычно называют *деревьями Фибоначчи*.

На рис. 1 представлены деревья T_0, T_1, T_2, T_3.

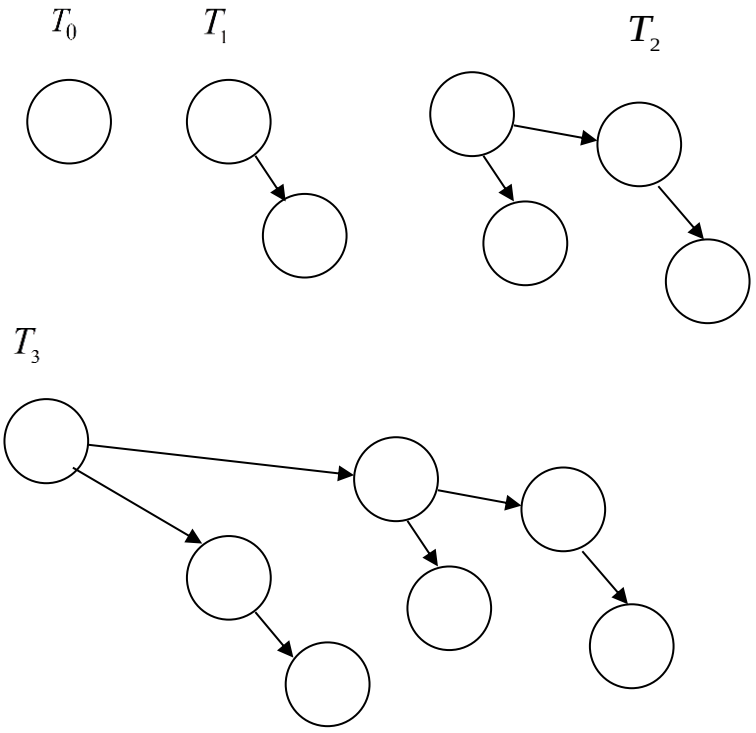


Рис. 1

Пусть N_h – количество внутренних вершин в дереве T_h. Тогда

$$N_{h+1} = N_{h-1} + N_h + 1.$$

Сделаем замену переменной $F'_i = N_i + 1$.

Тогда выражение $F'_{h+1} = = F'_h + F'_{h-1}$ соответствует числам Фибоначчи.

Напомним, что если F_h – h-е число Фибоначчи, то

$$F_h = \frac{\Phi^h - \hat{\Phi}^h}{\sqrt{5}},$$

где

$$\Phi = \frac{1+\sqrt{5}}{2}, \hat{\Phi} = \frac{1-\sqrt{5}}{2}.$$

Найдем соответствие между числами F_i и F'_i. Поскольку $F_i = F'_j$ при $i = 2 + j$, то $F_{2+h} = F'_h$, и

$$F'_h = \frac{\Phi^{h+2}}{\sqrt{5}} - \frac{\hat{\Phi}^{h+2}}{\sqrt{5}}.$$

Поэтому в силу того, что $N_h = F'_h - 1$ и $\left|\frac{1-\sqrt{5}}{2}\right| < 1$, имеем

$$n \geq N_h = F'_h - 1 \geq \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+2}}{\sqrt{5}} - \frac{\left(\frac{1-\sqrt{5}}{2}\right)^{h+2}}{\sqrt{5}} - 1 \geq \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+2}}{\sqrt{5}} - 2.$$

Логарифмируя выражение, получаем правую часть неравенства (1). Доказательство теоремы завершено.

Из теоремы следует, что для АВЛ-дерева с n вершинами высота $\mathrm{O}(\log n)$. Поэтому все базовые операции можно выполнить за время $\mathrm{O}(\log n)$.

Поскольку в результате выполнения базовых операций для АВЛ-дерева инвариант (сбалансированность по высотам) может нарушиться, то необходимо иметь процедуры, обеспечивающие поддержание инварианта. Причем трудоемкость этих процедур не должна превосходить трудоемкости базовых операций.

Основные операции с АВЛ-деревьями

1. *Добавление вершины с заданным ключом*

- Проходим по пути поиска (аналогично поиску элемента в бинарном поисковом дереве), пока не определим место добавления вершины *v* с заданным ключом.
- Добавляем новую вершину *v* и определяем, произошла ли вдоль пути поиска разбалансировка.

Пусть k_1 – вершина максимальной глубины, для которой нарушился инвариант. Легко проверить, что все вершины, для которых нарушился инвариант, лежат на пути от корня дерева до вершины *v* (это связано с тем, что поиск местоположения для добавляемой вершины *v* проходил именно по этой цепочке, и только по этой цепочке высоты поддеревьев могли увеличиться на 1).

• Если произошла разбалансировка, то выполнить процедуру, которая восстановит нарушенный инвариант (процедура восстановления инварианта будет рассмотрена позже). Процедура добавления элемента завершена.

В дальнейшем будет показано, что достаточно выполнить процедуру, восстанавливающую нарушенный инвариант, только для вершины k_1, так как это приведет к тому, что для всех вершин дерева будет выполнено свойство сбалансированности по высотам.

2. *Удаление вершины с заданным ключом*

• Осуществляем поиск и удаление вершины с заданным ключом. Данная процедура аналогична процедуре удаления для бинарного поискового дерева (рассматриваются варианты: удаляемый элемент имеет не более одного сына; удаляемый элемент имеет двух сыновей).

• Определяем, произошла ли вдоль пути поиска разбалансировка. Если после выполнения процедуры удаления высота некоторой вершины уменьшилась на 1, то для нее выполняется свойство сбалансированности по высотам. Нарушение инварианта могло произойти только для одной вершины на пути поиска, так как высоты только тех вершин, которые лежат на пути поиска могли уменьшиться на 1.

Пусть k_1 – вершина, для которой нарушился инвариант (высота вершины k_1 после выполнения процедуры удаления не изменилась, так как нарушение инварианта для нее произошло при удалении вершины из ее поддерева меньшей высоты).

• Если произошла разбалансировка, то выполнить процедуру (процедура восстановления инварианта будет рассмотрена позже), которая восстановит нарушенный инвариант, для вершины k_1 (z – отец вершины k_1). После выполнения балансировки для вершины k_1 может произойти разбалансировка еще одной вершины, лежащей на пути от корня дерева до вершины z. Находим опять вершину, для которой произошло нарушение инварианта, и выполняем балансировку для нее и т. д. Количество повторных балансировок ограничено высотой дерева.

Восстановление инварианта после выполнения операции добавления нового элемента

1-й случай. Пусть k_1 – вершина на максимальной глубине, для которой произошла разбалансировка после выполнения операции добавления нового элемента (z – отец вершины k_1) и для которой:

• высота ее левого поддерева больше высоты ее правого поддерева на 2 (балансировка происходит после каждой операции добавления и удаления, и максимально возможная разность высот поддеревьев для каждой

вершины не превышает двух, так как перед выполнением этой операции она не превышала 1);

- у левого сына k_2 вершины k_1 высота левого поддерева больше высоты правого поддерева (это соответствует добавлению элемента в поддерево, корень которого – левый сын вершины k_2).

До выполнения операции добавления нового элемента в поддерево, корень которого – левый сын вершины k_2 (на рис. 2, *а* добавление новой вершины выполнялось в дерево *A*, с высотой до выполнения процедуры добавления $h-3$), высота вершины k_1 была равна $h-1$ (поддеревья на рисунке обозначены прямоугольниками, вершины дерева – кругами). На рис. 2, *б* показано нарушение инварианта для вершины k_1.

На рис. 2, *в* показана процедура *LL*-поворота, которую необходимо выполнить, чтобы для вершины k_1 выполнился нарушенный инвариант. После выполнения процедуры *LL*-поворота высоты всех вершин, для которых было возможно нарушение инварианта (это вершины, лежащие на пути от корня дерева - вершине *z*), остались такими же, как и до процедуры добавления (рис. 2, *а* и рис. 2, *в*), т. е. для них выполняется свойство сбалансированности по высотам. Следовательно, *LL*-поворот приводит к сбалансированности по высотам всех вершин дерева.

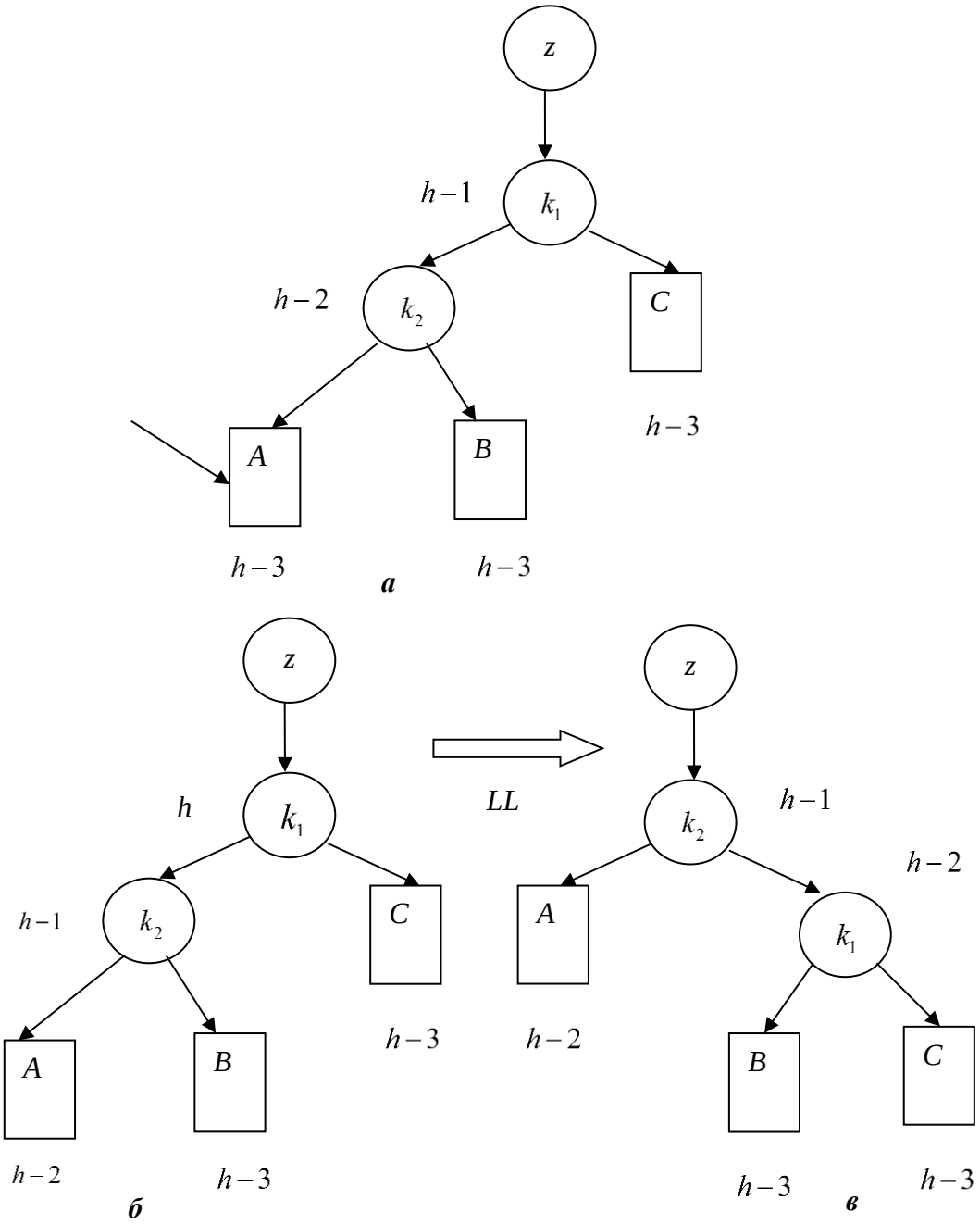


Рис. 2

2-й случай. Пусть k_1 – вершина на максимальной глубине, для которой произошла разбалансировка в результате выполнения операции добавления нового элемента и для которой:

- высота правого поддерева вершины k_1 больше высоты ее левого поддерева на 2;
- у правого сына вершины k_1 высота его правого поддерева больше высоты его левого поддерева (происходило добавление вершины в дерево, корнем которого является правый сын вершины k_2).

На рис. 3, *а* показано нарушение инварианта для вершины k_1. На рис. 3, *б* показана процедура *RR*-поворота, которую необходимо произвести, чтобы выполнился нарушенный инвариант для вершины k_1.

До выполнения операции добавления нового элемента в поддерево, корнем которого является правый сын вершины k_2 (на рис. 3 добавление новой вершины происходило в дерево *C*, с высотой до выполнения процедуры добавления $h-3$), высота вершины k_1 была равна $h-1$.

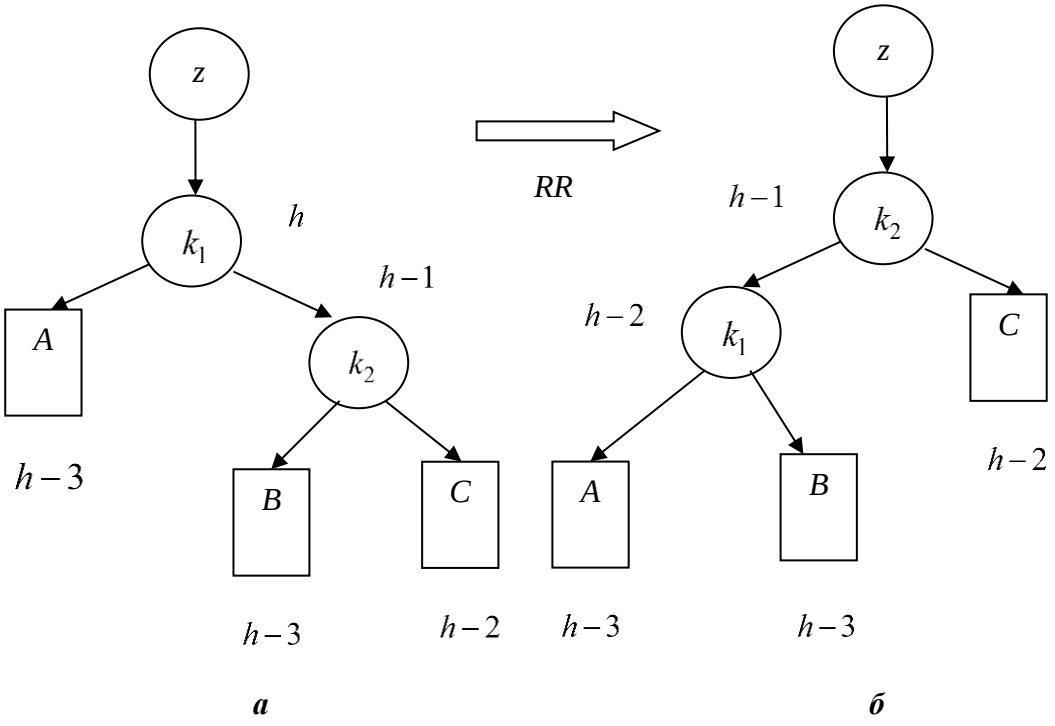


Рис. 3

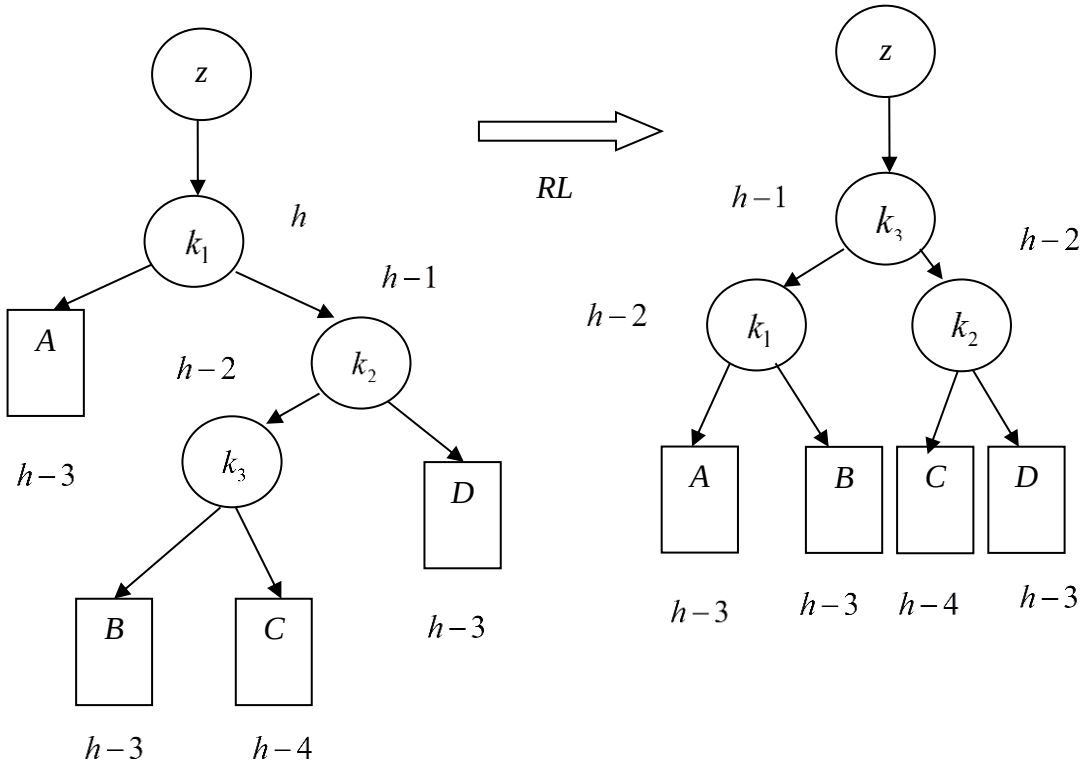
Аналогично, как и для процедуры *LL*-поворота, процедура *RR*-поворота (примененная к вершине, которая расположена на максимальной глубине и для которой нарушается инвариант), приводит к выполнению свойства сбалансированности по высотам для всех вершин дерева.

3-й случай. Пусть k_1 – вершина на максимальной глубине, для которой произошло нарушение инварианта в результате выполнения процедуры добавления нового элемента (*z* – отец вершины k_1) и для которой:

- высота правого поддерева вершины k_1 больше высоты ее левого поддерева на 2;
- у правого сына k_2 вершины k_1 высота левого поддерева больше высоты его правого поддерева.

На рис. 4, *а* показано нарушение инварианта для вершины k_1. На рис. 4, *б* изображена процедура *RL*-поворота, которую необходимо выполнить, чтобы выполнился нарушенный инвариант для вершины k_1.

Заметим, что нарушение инварианта произошло после добавления новой вершины в дерево *B* (рис. 4). До процедуры добавления это дерево имело высоту $h-4$, а высота вершины k_1 была равна $h-1$. После выполнения процедуры *RL*-поворота высоты вершин, лежащих на пути от корня до вершины *z*, остались теми же, что и до выполнения операции добавления элемента. Следовательно, выполнение процедуры *RL*-поворота привело к выполнению инварианта для всех вершин дерева.



а ***б***

Рис. 4

4-й случай. Пусть k_1 – вершина на максимальной глубине, для которой произошло нарушение инварианта в результате выполнения процедуры добавления нового элемента (*z* – отец вершины k_1) и для которой:

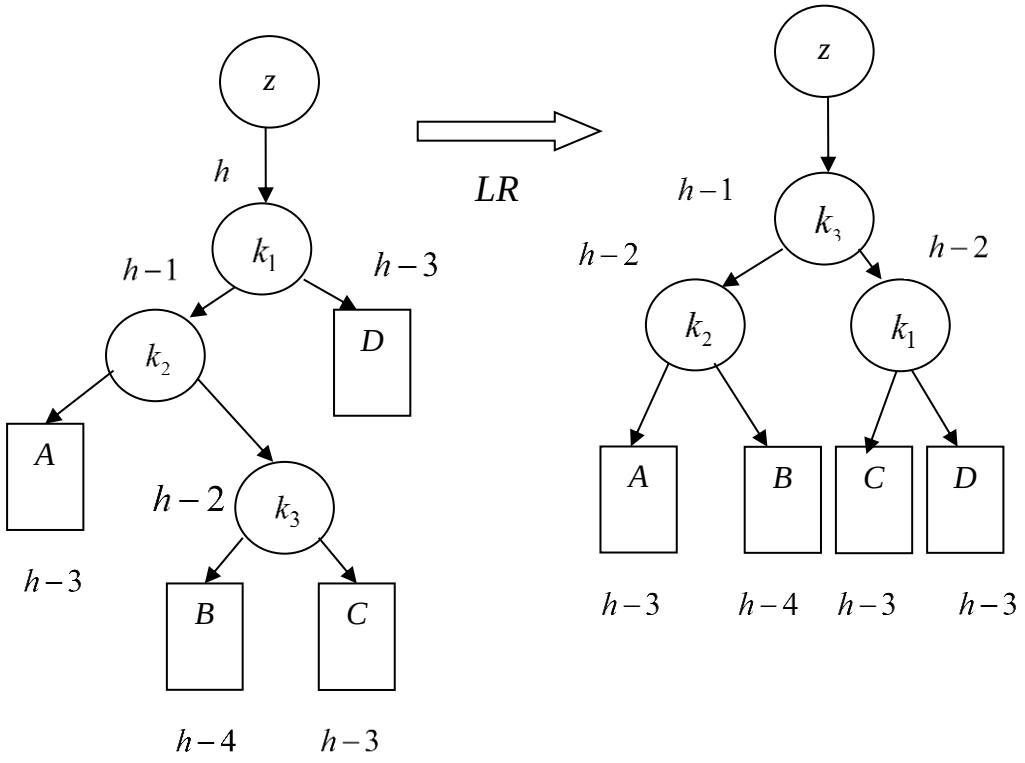
- высота левого поддерева вершины k_1 больше высоты правого поддерева на 2;
- у левого сына k_2 вершины k_1 высота правого поддерева больше высоты его левого поддерева.

На рис. 5, *а* показано нарушение инварианта для вершины k_1. На рис. 5, *б* изображена процедура *LR*-поворота, необходимая для того, чтобы выполнился нарушенный инвариант для вершины k_1.

Заметим, что нарушение инварианта произошло после добавления новой вершины в дерево *C* (рис. 5). До процедуры добавления это дерево имело высоту $(h-4)$, а высота вершины k_1 была равна $h-1$.

После выполнения процедуры *LR*-поворота высоты вершин, лежащих на пути от корня дерева до вершины *z*, остались теми же, что и до выполнения операции добавления нового элемента.

Следовательно, выполнение процедуры *LR*-поворота привело к выполнению инварианта для всех вершин дерева.



а ***б***

Рис. 5

Восстановление инварианта

после выполнения операции удаления элемента

1-й случай. Пусть k_1 – вершина, для которой произошла разбалансировка в результате выполнения процедуры удаления элемента из АВЛ-дерева (z – отец вершины k_1) и для которой:

- высота ее левого поддерева больше высоты ее правого поддерева *C* на 2 (происходило удаление вершины из дерева *C*);
- у левого сына k_2 вершины k_1 высота левого поддерева *A* больше или равна высоты ее правого поддерева *B*.

Процедура удаления вершины выполнялась для дерева *C* (рис. 6). До удаления вершины высота дерева *C* была равна $h-2$, а высота вершины k_1 равнялась *h* (рис. 6, *а*).

Рассмотрим сначала ситуацию, когда высоты поддеревьев *A* и *B* равны.

На рис. 6, *б* показано нарушение инварианта для вершины k_1.

На рис. 6, *в* показана процедура *LL*-поворота, которую необходимо выполнить, чтобы для вершины k_1 выполнился нарушенный инвариант.

После выполнения процедуры *LL*-поворота высоты всех вершин, лежащих на пути от корня дерева до вершины *z*, остались теми же, что и до выполнения операции удаления. Следовательно, для всех вершин дерева выполняется инвариант.

Рассмотрим теперь ситуацию, когда высота дерева *A* больше высоты дерева *B* (высота дерева *B* равна $h-3$). Выполнение процедуры *LL*-поворота (рис. 6, *в*) приведет к тому, что высота вершины k_1 станет равной $h-2$, а у вершины k_2 высота будет равна $h-1$.

Следовательно, после выполнения процедуры *LL*-поворота высоты вершин, лежащих на пути от корня дерева до вершины *z*, могли уменьшиться на 1, а это может привести к тому, что для одной из них нарушится инвариант.

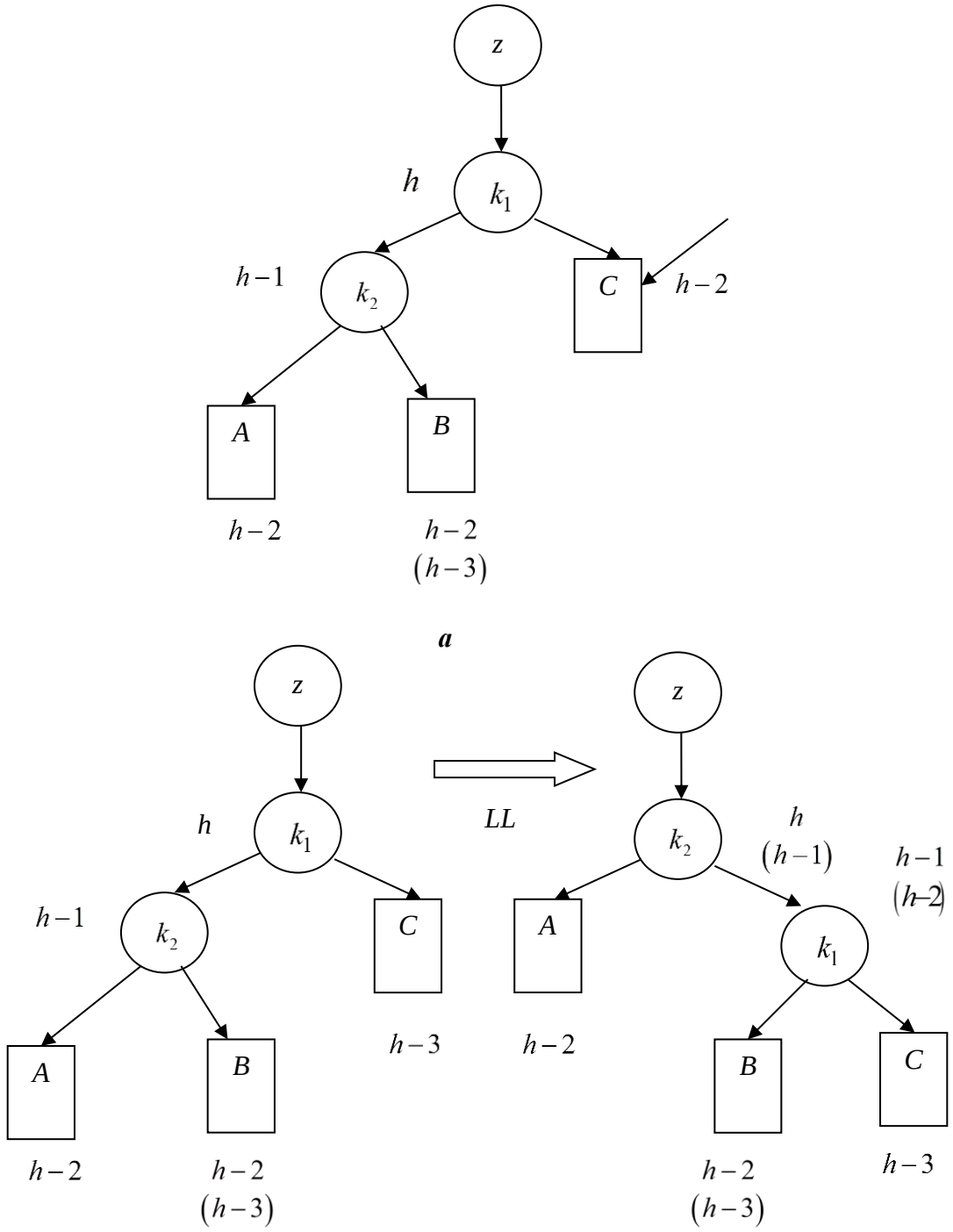

z
h
k_1
$h-1$
k_2
C
$h-2$
A
B
$h-2$
$h-2$
$(h-3)$
а
z
z
LL
h
k_1
$h-1$
k_2
C
A
$h-3$
$h-2$
B
$h-2$
$h-2$
$(h-3)$
h
$(h-1)$
$h-1$
$(h-2)$
k_1
B
C
$h-2$
$(h-3)$
$h-3$
б
в

Рис. 6

2-й случай. Пусть k_1 – вершина, для которой произошла разбалансировка в результате выполнения процедуры удаления элемента (*z* – отец вершины k_1) и для которой:

- высота ее правого поддерева больше высоты ее левого поддерева *A* на 2 (происходило удаление вершины из дерева *A*);
- у правого сына k_2 вершины k_1 высота правого поддерева *C* больше или равна высоте левого поддерева *B*.

Сначала рассмотрим первую ситуацию, когда высота дерева *C* больше высоты дерева *B*.

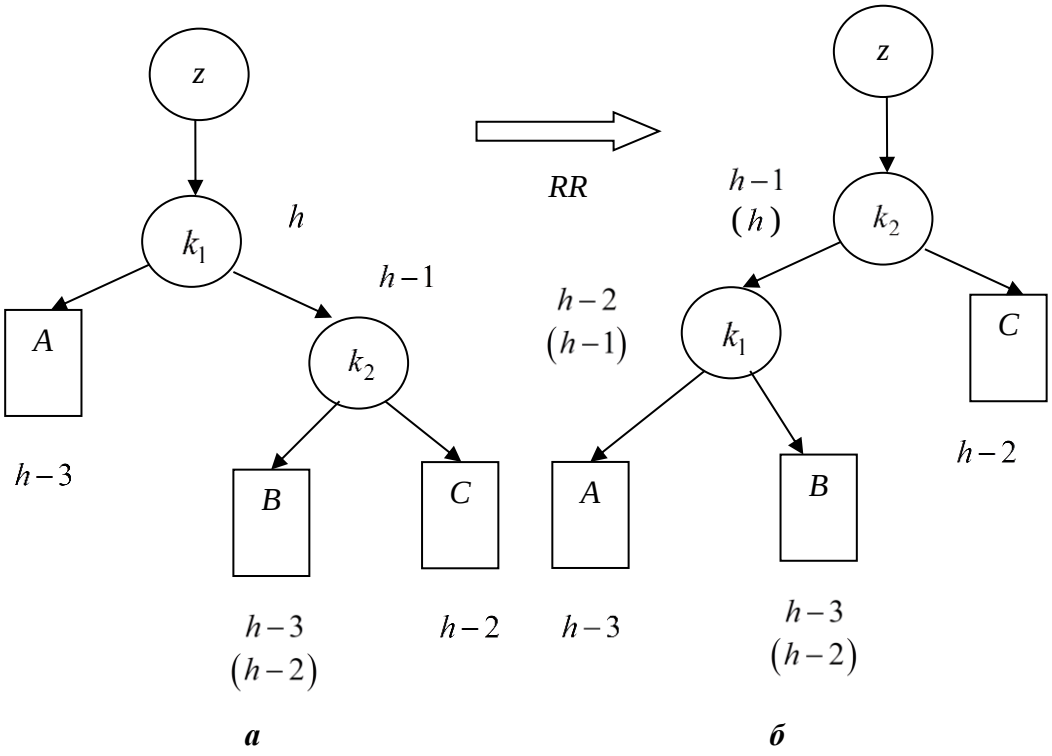


Рис. 7

На рис. 7, *а* показано нарушение инварианта для вершины k_1, на рис. 7, *б* – процедура *RR*-поворота, необходимая для того, чтобы выполнился нарушенный инвариант. До выполнения операции удаления элемента из дерева *A* его высота равнялась $h-2$, а высота вершины k_1 равнялась *h*. После выполнения процедуры *RR*-поворота высота вершины k_2 стала равной $h-1$, что может привести к появлению вершины, лежащей на пути от корня дерева до вершины *z*, у которой будет нарушен инвариант.

Теперь перейдем к рассмотрению второй ситуации, когда высоты деревьев *B* и *C* равны (на рис. 7, *а* высоты поддеревьев равны $h-2$). После выполнения процедуры *RR*-поворота высота вершины k_1 будет равна $h-1$, а высота вершины k_2 – *h*. Следовательно, выполнение процедуры *RR*-поворота привело к выполнению инварианта для всех вершин дерева.

3-й случай. Пусть k_1 – вершина, для которой произошло нарушение инварианта в результате выполнения процедуры удаления элемента (z – отец вершины k_1) и для которой:

- высота правого поддерева вершины k_1 больше высоты ее левого поддерева A на 2 (происходило удаление вершины из дерева A);
- у правого сына k_2 вершины k_1 высота левого поддерева больше высоты его правого поддерева.

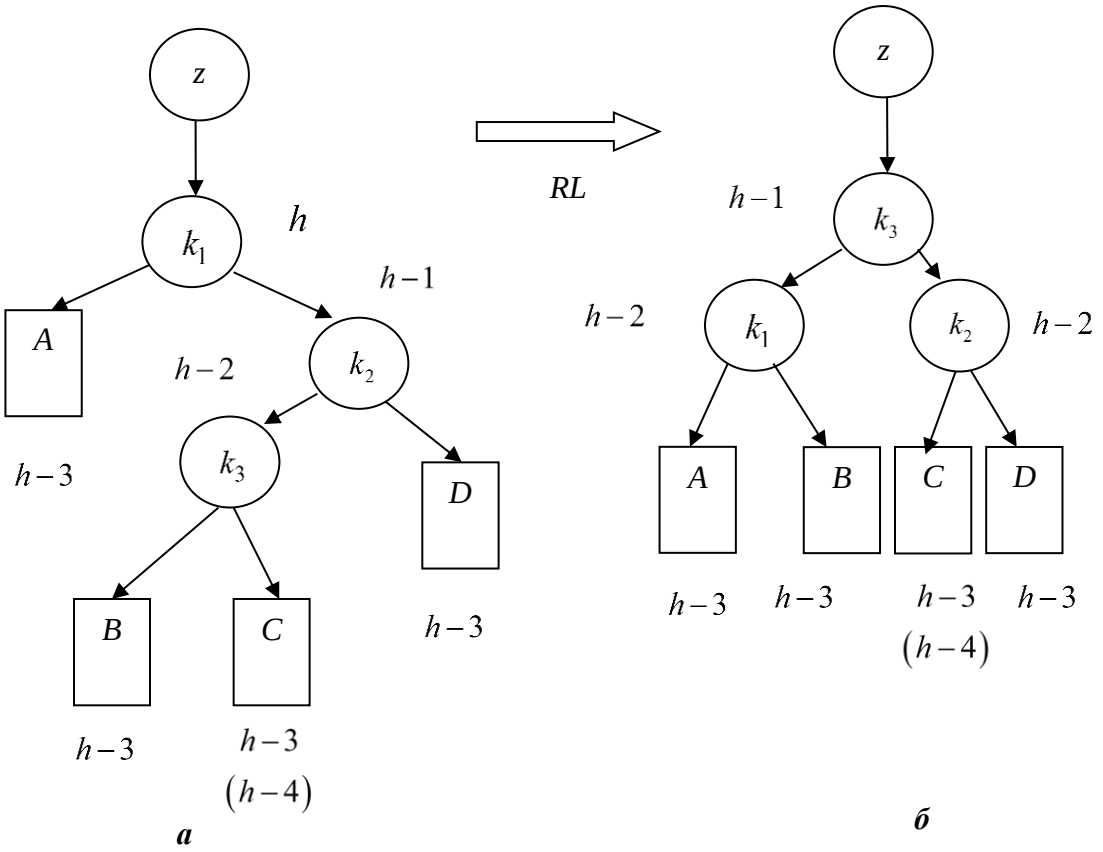


Рис. 8

На рис. 8, *а* показано нарушение инварианта для вершины k_2, на рис. 8, *б* изображена процедура *RL*-поворота, которую необходимо выполнить для восстановления инварианта.

До выполнения процедуры удаления вершины из дерева A его высота была равна $h-2$, а высота вершины k_1 равнялась h. После выполнения процедур *RL*-поворота высота вершины k_3 стала равной $h-1$ (рис. 8, *б*).

Следовательно, возможно появление вершины, лежащей на пути от корня дерева до вершины z, у которой будет нарушен инвариант.

4-й случай. Пусть k_1 – вершина, для которой произошло нарушение инварианта в результате выполнения процедуры удаления элемента (z – отец вершины k_1) и для которой:

- высота левого поддерева вершины k_1 больше высоты правого поддерева D на 2 (происходило удаление вершины из дерева D);
- у левого сына k_2 вершины k_1 высота правого поддерева больше высоты его левого поддерева.

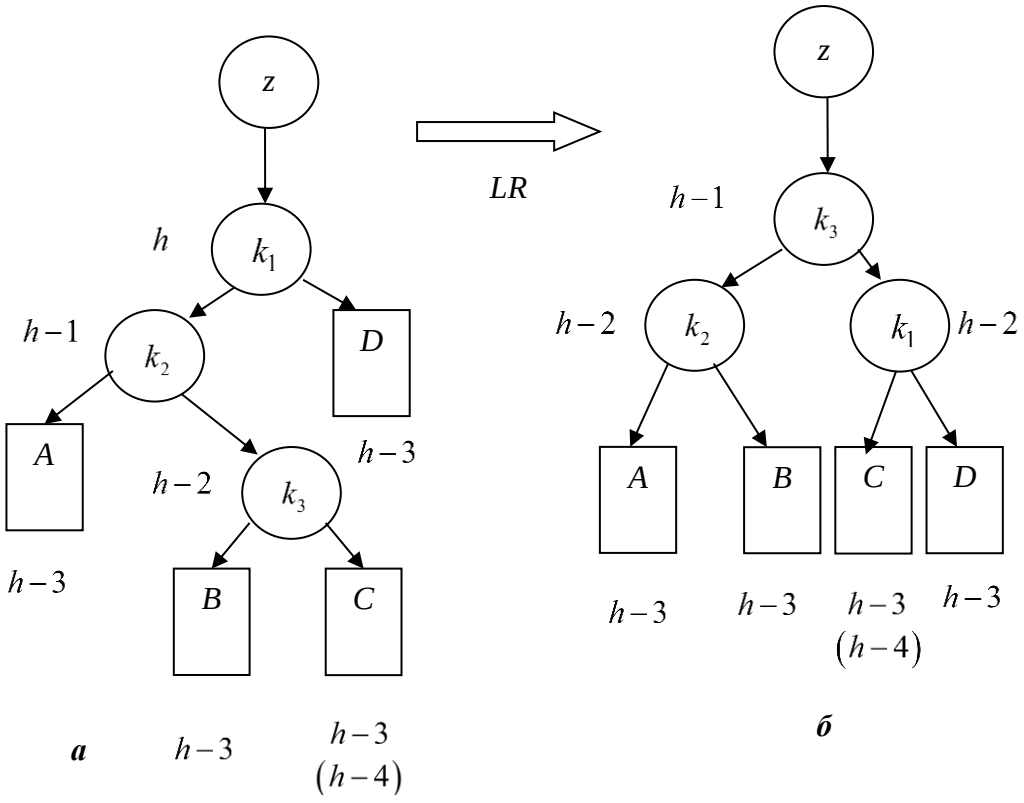


Рис. 9

На рис. 9, *а* показано нарушение инварианта для вершины k_1, на рис. 9, *б* изображен *LR*-поворот, который необходимо выполнить для восстановления инварианта. До выполнения процедуры удаления вершины из дерева D его высота была равна $h-2$, а высота вершины k_1 равнялась h. После выполнения *LR*-поворота высота вершины k_3 стала равной $h-1$ (рис. 9, *б*). Следовательно, возможно появление вершины, лежащей на пути от корня дерева до вершины z, у которой будет нарушен инвариант.

Трудоемкость выполнения всех базовых операции для АВЛ-дерева есть $O(\log n)$. Экспериментальные исследования показывают, что в среднем приблизительно на две процедуры добавления нового элемента приходится одна балансировка, а при выполнении процедуры удаления элемента балансировка происходит даже в одном из пяти случаев.

2-3-дерево

Определение. Поисковое дерево называется 2-3-деревом, если оно обладает следующими свойствами (*инвариантами*):

- каждая вершина x, не являющаяся листом, содержит два или три сына; при этом его сыновья классифицируются как левый $ls(x)$, средний $ms(x)$ и (возможно) правый сын $rs(x)$;
- все висячие вершины находятся на одной глубине.

Отметим, что дерево, состоящее из одной единственной вершины, также считается 2-3-деревом.

Поскольку 2-3-дерево является поисковым, то для каждой вершины v выполняются следующие свойства:

1) значение ключей в поддереве, корень которого – левый сын вершины v, меньше значений ключей в поддереве, корень которого – средний сын вершины v;

2) значение ключей в поддереве, корень которого – средний сын вершины v, меньше значений ключей в поддереве, корень которого – правый сын вершины v.

Структура 2-3-дерева

1. Информация (ключи) хранится только в висячих вершинах, а все внутренние вершины – справочные.
2. Каждая внутренняя вершина i имеет две метки:
   - $l(i)$ – максимальное значение ключа в поддереве, корень которого – левый сын вершины i;
   - $m(i)$ – максимальное значение ключа в поддереве, корень которого – средний сын вершины i.

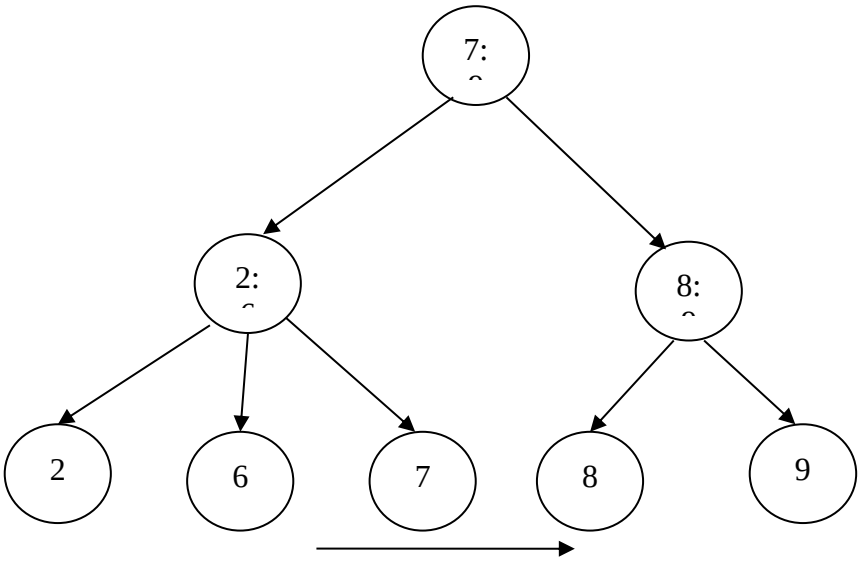


Рис. 10

Любое упорядоченное множество A можно представить в виде 2-3-дерева (рис. 10), если присваивать значения элементов этого множества висячим вершинам (элементы можно представить в порядке слева направо).

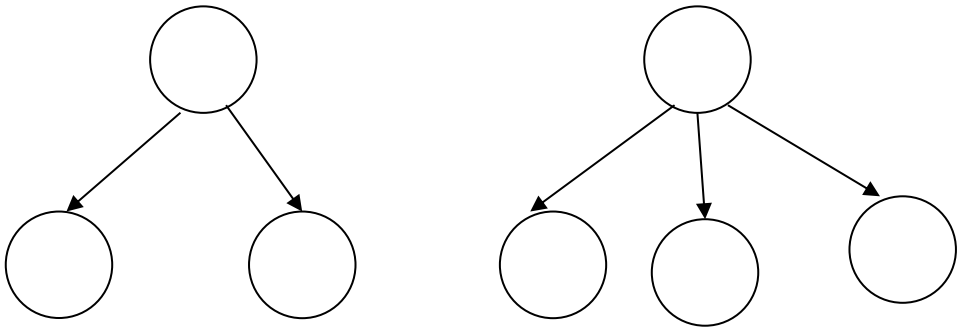
Докажем сейчас теорему, которая устанавливает связь между количеством узлов, количеством листьев и высотой 2-3-дерева.

ТЕОРЕМА. Пусть n – общее количество вершин в 2-3-дереве (включая корень и листья); l – количество листьев; h – высота дерева. Тогда справедливы следующие неравенства:

$$2^h \le l \le 3^h,$$

$$2^{h+1} - 1 \le n \le \frac{3^{h+1} - 1}{2}.$$

Доказательство. Пусть высота 2-3-дерева $h = 1$, тогда существуют только два дерева, которые приведены на рис. 11.



$l = 2,$ $n = 3$ $\qquad$ $l = 3,$ $n = 4$

Рис. 11

Для высоты дерева $h = 1$ утверждение верно, так как

$$2^1 \le l \le 3^1,\ 4 - 1 \le n \le \frac{9-1}{2}.$$

Предположим, что неравенства выполняются для дерева высотой h, докажем, что они верны и для $h = h + 1$.

Пусть имеется дерево T_h высотой h. Сначала рассмотрим соотношения для листьев. Обозначим через l_h количество листьев в дереве T_h, а через $l_h^{\min}$ ($l_h^{\max}$) – минимально (максимально) возможное количество листьев в дереве T_h. Увеличение высоты дерева T_h на единицу приводит к тому, что

на глубине $h+1$ максимальное количество листьев $l_{h+1}^{\max}$ не превосходит $3l_h^{\max}$ (когда к каждому листу T_h добавляется три новых сына) и не меньше величины $2l_h^{\min}$ (когда к каждому листу T_h добавляется два новых сына).

Следовательно, выполнены следующие неравенства:

$$l_{h+1}^{\min} \le l_{h+1} \le l_{h+1}^{\max},$$

$$2l_h^{\min} \le l_{h+1} \le 3l_h^{\max}.$$

В силу сделанного ранее нами индукционного предположения имеем

$$2^{h+1} \le l_{h+1} \le 3^{h+1}.$$

Теперь докажем соотношение для общего количества вершин. Обозначим через n_h общее количество вершин в дереве T_h, а через $n_h^{\min}$ ($n_h^{\max}$) – минимально (максимально) возможное количество вершин в дереве T_h.

Тогда верны следующие неравенства:

$$n_h^{\min} + l_{h+1}^{\min} \le n_{h+1} \le n_h^{\max} + l_{h+1}^{\max}.$$

В силу индукционного предположения и с учетом того, что

$$2^{h+1} - 1 + 2^{h+1} = 2^{h+2} - 1$$

и

$$\frac{3^{h+1}-1}{2} + 3^{h+1} = \frac{3^{h+1} + 2\cdot 3^{h+1} - 1}{2} = \frac{3\cdot 3^{h+1} - 1}{2} = \frac{3^{h+2}-1}{2},$$

получаем требуемое неравенство:

$$2^{h+2} - 1 \le n_{h+1} \le \frac{3^{h+2}-1}{2}.$$

Доказательство теоремы завершено.

Основные операции с 2-3-деревом

1. ***Поиск элемента с заданным ключом a***

- Пусть *root* – корень 2-3-дерева. Полагаем $t = root$.
- Пока *t* не лист, повторяем следующую последовательность шагов:

1) если $a \le l(t)$, то $t := ls(t)$;

2) иначе если $a \le m(t)$ или $rs(t) = 0$ (у вершины только два сына), то $t := ms(t)$;

3) иначе $t := rs(t)$.

- Сравниваем ключевое значение висячей вершины *t* с ключом *a*; если они совпадают, то вершина найдена, в противном случае вершины с заданным ключом *a* в дереве не существует.

Процедура поиска элемента с заданным ключом *a* завершена.

2. ***Добавление элемента с ключом a*** (предположим, что нам необходимо добавить вершину с ключом *a* в дерево, у которого имеется хотя бы одна вершина).

Если дерево состоит из единственной вершины *w* ($key(w)$ – ключ вершины *w*), то:

1) образуем новую вершину *v* (лист) с ключом $key(v) := a$;

2) образуем новый корень *r*;

3) если $key(w) < key(v)$, то полагаем *w* и *v* левым и правым сыном корня *r* соответственно; в противном случае *w* и *v* – правый и левый сын *r*;

4) формируем метки вершины *r*.

Данная ситуация представлена на рис. 12. К дереву, состоящему из единственной вершины с ключом 2, добавляется элемент с ключом 3.

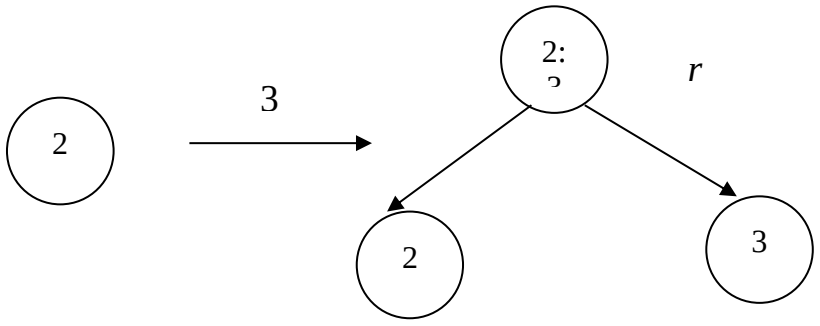


Рис. 12

Рассмотрим сейчас ситуацию, когда в дереве более одного узла. В этом случае для добавления элемента с заданным ключом *a* необходимо выполнить следующие действия.

1) Осуществляем поиск отца *f* для вновь добавляемой вершины *v* с ключевым значением *a* (это будет отец вершины *t* в описанной процедуре поиска элемента с заданным ключом *a*).

2) Формируем новый лист *v* и полагаем $key(v) := a$.

3) Определяем позицию вершины *v* по отношению к отцу.

Если у отца *f* было только два сына – v_1 и v_2.

а) если $key(v) < key(v_1)$, то v, v_1, v_2;

б) если $key(v) > key(v_2)$, то v_1, v_2, v;

в) если $key(v_1) < key(v) < key(v_2)$, то v_1, v, v_2.

Если у отца f уже было три сына – v_1, v_2, v_3, то аналогичным образом определяем позицию v по отношению к отцу.

Возможна одна из следующих конфигураций:

$$(v, v_1, v_2, v_3);$$
$$(v_1, v, v_2, v_3);$$
$$(v_1, v_2, v, v_3);$$
$$(v_1, v_2, v_3, v).$$

4) Если у отца f было только два сына – v_1 и v_2, то в соответствии с найденной конфигурацией устанавливаем для вершины f ссылки на левого, среднего и правого сына и корректируем метки вершин вдоль пути от корня к добавленной вершине v; процедура добавления элемента завершена (рис. 13).

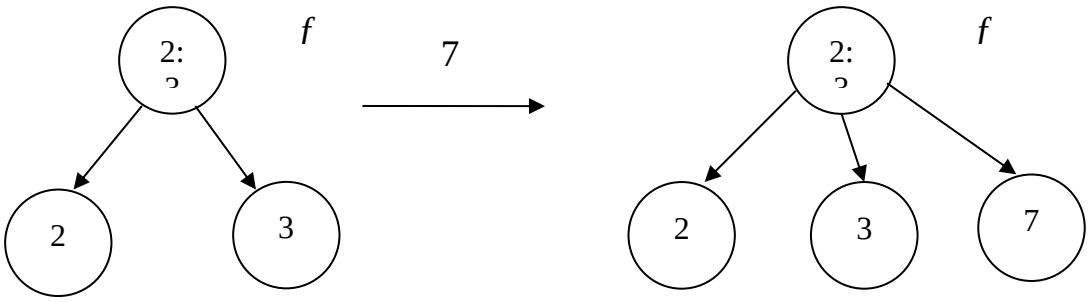


Рис. 13

5) Если у отца f уже было три сына – v_1, v_2, v_3, то будем выполнять следующие действия:

а) в соответствии с найденной конфигурацией, предположим для определенности, что она равна v_1, v, v_2, v_3, т. е.

$$key(v_1) < key(v) < key(v_2) < key(v_3),$$

образуем новую вершину v'.

Сыновьями вершины v' полагаем две максимальные вершины из найденной конфигурации, а сыновьями вершины f – две минимальные вершины (рис. 14);

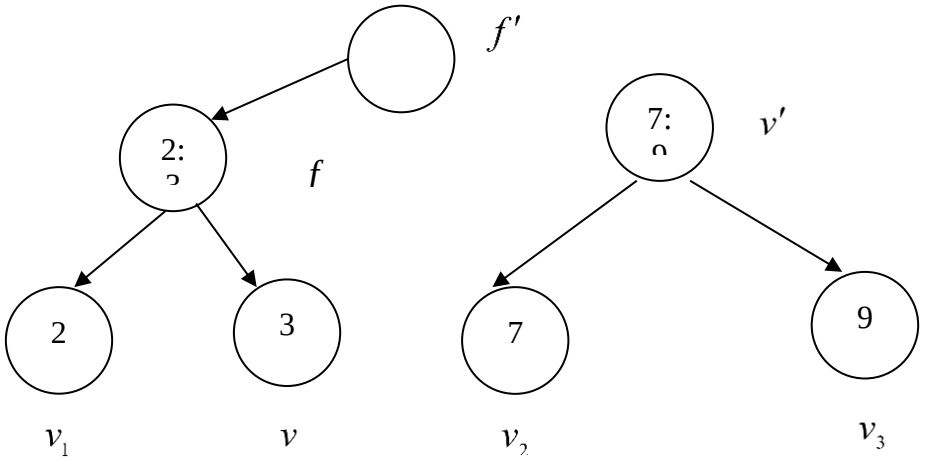


Рис. 14

б) пусть f' – отец вершины f (рис. 14).

Если у вершины f нет отца (f – корень дерева), то образуем новый корень r' и назначаем f и v' левым и средним сыновьями корня r' (а высота 2-3-дерева увеличивается на единицу). Корректируем метки и завершаем процедуру добавления элемента (рис. 15.)

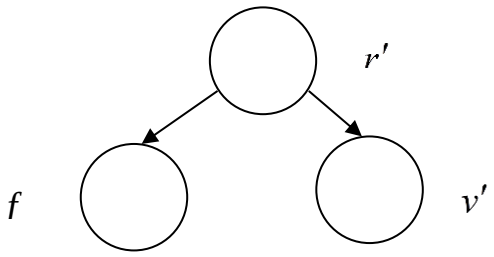


Рис. 15

Если у вершины f есть отец f', то определяем конфигурацию сыновей f и v' вершины f' следующим образом: вершина v' следует непосредственно за вершиной f (рис. 16).

Полагаем $f = f', v = v'$ и возвращаемся к шагу 4) алгоритма.

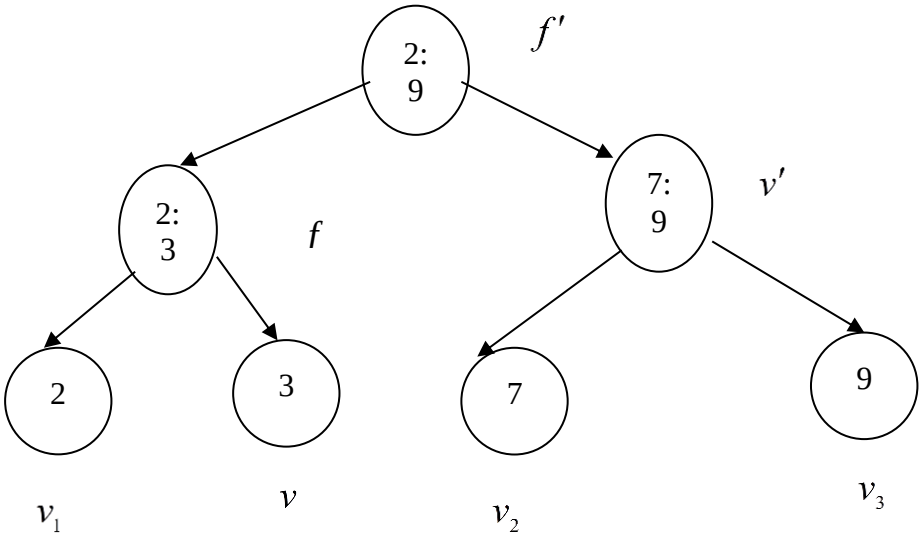


Рис. 16

Трудоемкость алгоритма добавления элемента с заданным ключом зависит от общего количества добавлений вершин (начальной вершины v и вершин v', добавление которых потребовалось для поддержания инвариантов 2-3-дерева).

Очевидно, что количество добавляемых вершин не превосходит высоты дерева h, так как на каждом следующем шаге добавляется вершина на большей высоте.

Как следует из теоремы 2, высота 2-3-дерева с n листьями не превосходит $\log n$, и так как для добавления одной вершины требуется конечное число операций, то сложность алгоритма добавления нового элемента в 2-3-дерево есть $O(\log n)$.

3. Удаление элемента с ключом a

4.

1) Найдем вершину v с ключом $key(v) = a$.

2) Если вершина v была корнем дерева, то удаляем вершину v, получаем пустое дерево и завершаем процедуру удаления.

3) Пусть f – отец вершины v. Удаляем вершину v.

4) Если у вершины f осталось два сына (это допустимо для 2-3-дерева), то определяем конфигурацию сыновей вершины f, формируем метки вершин и завершаем процедуру удаления.

5) Если у вершины f остался только один сын – v_1, то:

а) если вершина f – корень дерева, то вершину v_1 делаем корнем дерева, формируем метки вершин и завершаем процедуру удаления;

б) если вершина f – не корень дерева, то она должен иметь, по крайней мере, одного брата g.

Предположим, что брат g находится справа от вершины f.

- Если у брата g только два сына – w_1 и w_2, то делаем v_1 левым сыном вершины g, а вершины w_1 и w_2 – средним и правым сыновьями. Полагаем $v = f$ и возвращаемся к шагу 3) алгоритма.

- Если у брата g три сына – w_1, w_2 и w_3, то определяем v_1 и w_1 как левого и среднего сына вершины f, а w_2 и w_3 – как левого и среднего сына вершины g. Формируем метки и завершаем процедуру удаления элемента.

Предположим, что брат g находится слева от вершины f.

- Если у брата g только два сына – w_1 и w_2, тогда делаем v_1 правым сыном вершины g, полагаем $v = f$ и возвращаемся к шагу 3) алгоритма (рис. 17).

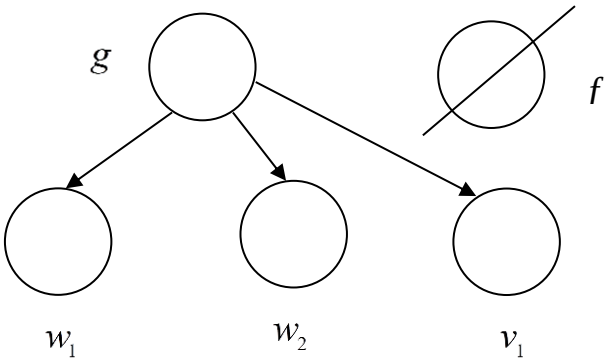


Рис. 17

- Если у брата g три сына – w_1, w_2 и w_3, то самого правого сына – w_3 – делаем левым сыном вершины f, а v_1 делаем средним сыном вершины f, формируем метки и завершаем процедуру удаления элемента (рис. 18).

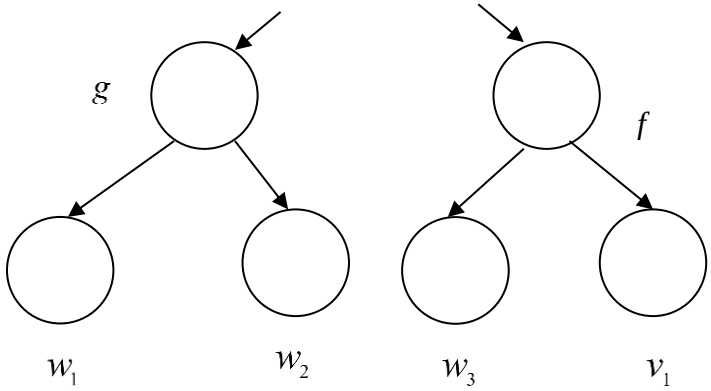


Рис. 18

Трудоемкость алгоритма удаления вершины *v* зависит от общего количества удаляемых вершин (начальной вершины *v* и тех, удаление которых потребовалось для поддержания инвариантов 2-3-дерева).

Общее количество удаляемых вершин не превосходит высоты дерева, так как на каждом очередном шаге алгоритма происходит удаление вершины на большей высоте.

Как следует из теоремы 2, высота 2-3-дерева с n листьями не превосходит $\log n$, и так как удаление одной вершины выполняется за конечное число шагов, то сложность алгоритма удаления элемента из 2-3-дерева есть $O(\log n)$.

Важной операцией со данной структурой данных является процедура слияния (*merge*) двух 2-3-деревьев, при это

м все листья одного из деревьев меньше листьев другого дерева. Время, котороется требуется для слияния деревьев пропорционально разности их высот, т.е. есть $O(\log n)$.

Еще одной операцией, которая выполняется за время $O(\log n)$, является операция «разрезания» (*split*) по ключу *x*. В результате выполнения данной операции формируются два 2-3-деревьева, причем у первого дерева значения листьев не превосходят величины *x*.

Упражнение 1. Смоделируйте операцию *split*, которая должна работать за время $O(\log n)$. В каком порядке нужно выполнять слияние 2-3-деревьев, полученных на промежуточных этапах алгоритма?

Упражнение 2. Разработайте алгоритм, который удаляет из 2-3-дерева висячие вершины ключи которых лежат на отрезке [a, b]. Алгоритм должен работать за время $O(\log n)$, т.е. время его работы не должно зависеть от числа висячих вершин, которые лежат на отрезке [a, b].