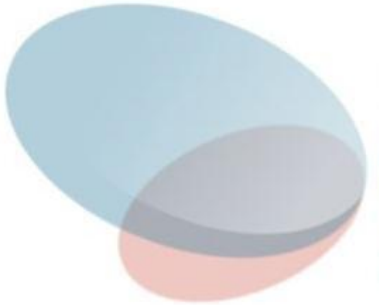


Алгоритмы и структуры данных
Лекция 7
Сбалансированные деревья. АВЛ-деревья

Идеально сбалансированные деревья

Определение. Дерево называется **идеально сбалансированным**, если **все его уровни, за исключением, может быть, последнего, полностью заполнены**. В бинарном дереве полностью заполненный уровень n содержит 2^n узлов.

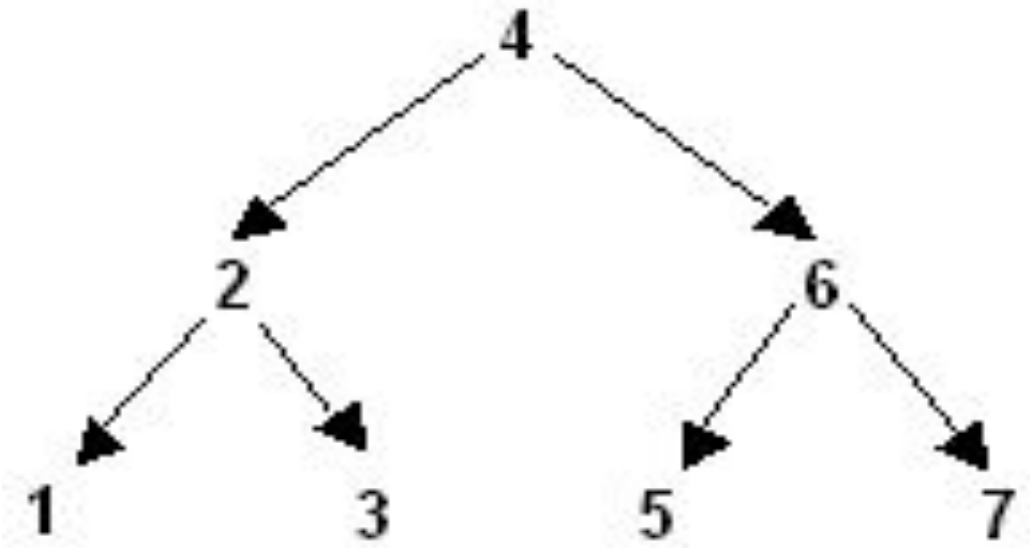
Если **дерево поиска** близко к сбалансированному, то даже **в худшем случае** за время порядка O(log_2 n) в нем можно:

- **Найти** вершину (узел) с заданным значением или выяснить, что такой вершины нет.
- **Включить (добавить)** новую вершину.
- **Исключить (удалить)** вершину.

Идеально сбалансированные деревья

Среднее время выполнения операций поиска, добавления, удаления будет также близким к O(log_2 n), так как в идеально сбалансированном дереве при полностью заполненных уровнях **на последнем уровне находится больше половины узлов дерева**.

Например, последовательность значений 4, 6, 2, 1, 5, 3, 7 дает следующее идеально сбалансированное дерево:



АВЛ-деревья (AVL-деревья)

Приведение уже существующего дерева к идеально сбалансированному - процесс сложный. Проще **балансировать дерево в процессе его роста (после включения каждого узла).**

Однако требование **идеальной сбалансированности** делает и этот процесс достаточно сложным, способным затрагивать все узлы дерева.

В **1962 году** советские математики **Адельсон-Вельский Г.М.** и **Ландис Е.А.** предложили **метод балансировки,** требующий после включения или исключения узла лишь **локальные изменения вдоль пути от корня к данному узлу, то есть времени не более O(log_2 n).**

При этом деревьям разрешается отклоняться от идеальной сбалансированности, но в небольшой степени, чтобы среднее время доступа к узлам было лишь немногим больше, чем в идеально сбалансированном дереве. Такие деревья получили название АВЛ-деревьев (AVL).

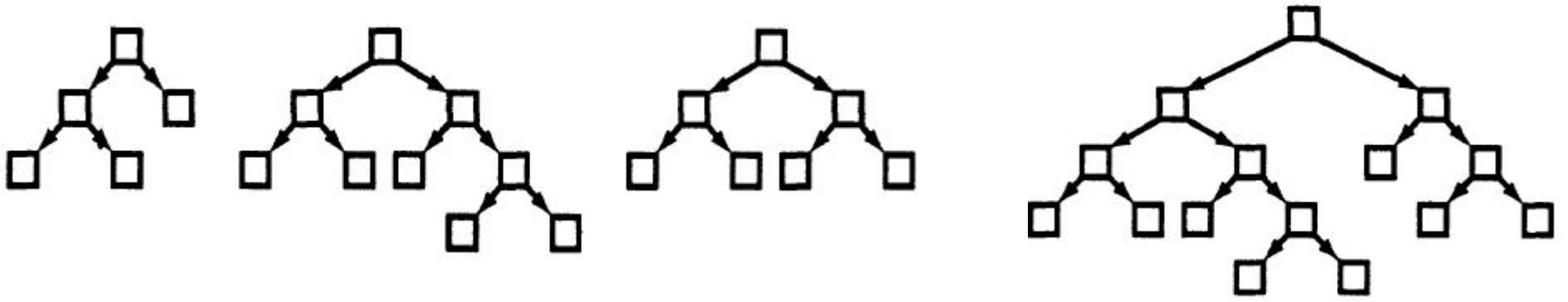
АВЛ-деревья (AVL-деревья)

Определение. АВЛ-деревом называется сбалансированное по высоте двоичное дерево поиска, для каждой вершины которого высота её двух поддеревьев различается не более чем на 1.

Высота узла x равна длине самого длинного пути от узла x вниз до внешнего узла (листа) поддерева x. **Высота дерева** определяется как высота его корневого узла.

На рисунке показано несколько АВЛ-деревьев.

АВЛ-дерево имеет **высоту порядка O(log(N)).** Это означает, что **поиск узла** в АВЛ-дереве **занимает время порядка O(log(N))**, что достаточно быстро. Не столь очевидно, что **вставить или удалить элемент из АВЛ-дерева можно за время порядка O(log(N)),** сохраняя при этом баланс дерева.



АВЛ-деревья (AVL-деревья)

Под **балансом узла T двоичного дерева** понимают разность высот его правого TR и левого TL поддеревьев:

balance(T)=h(TR) - h(TL)

(или разность высот его левого TL и правого TR поддеревьев:

balance(T)=h(TL) - h(TR)).

Определение. Двоичное дерево поиска называется **сбалансированным по высоте или AVL-деревом**, если для любого его узла выполняется условие |balance(T)|<=1.

Добавление узла к AVL-дереву

Процедура, которая **вставляет в дерево новый узел**, **рекурсивно спускается вниз по дереву** (начиная путь с корня), **чтобы найти местоположение узла** (нисходящая рекурсия).

После вставки вершины, **она исследует узлы в обратном порядке – к корню** (восходящая рекурсия)**, проверяя, чтобы высота поддеревьев отличалась не более чем на единицу.**

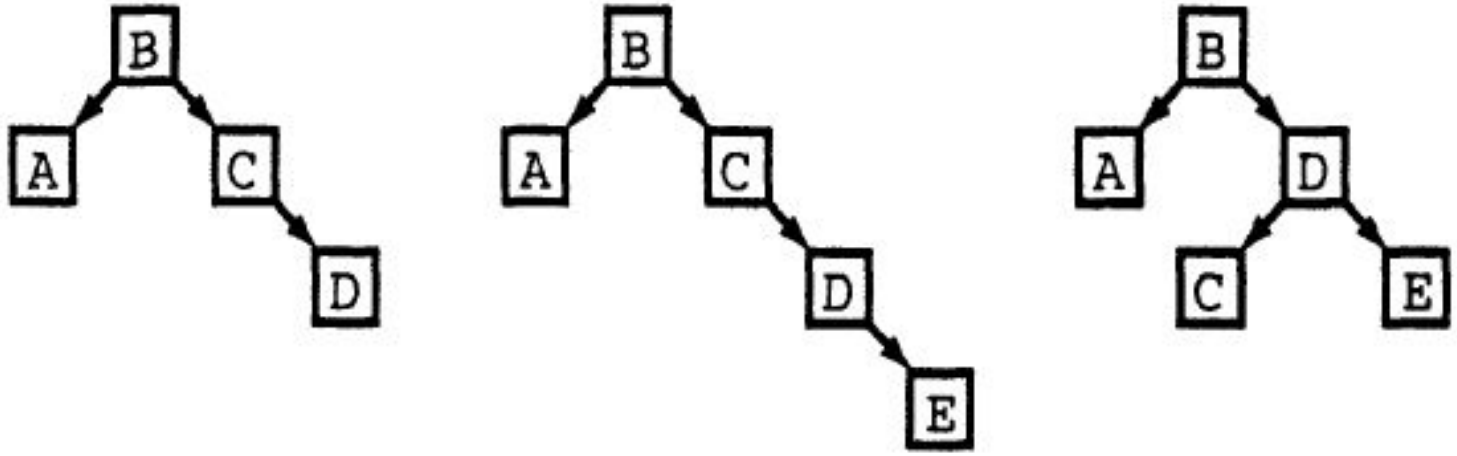
Если найдена вершина, где это **условие не выполняется**, то **элементы сдвигаются по кругу** (как - будет показано далее), чтобы сохранить выполняемость условия AVL-дерева.

Добавление узла к AVL-дереву

Пример. Дерево на рисунке слева является сбалансированным АВЛ-деревом. Если **добавить к дереву новый узел E**, то получится среднее дерево на рисунке, не являющееся сбалансированным. После добавления выполняется **проход вверх по дереву от нового узла E**.

В самом **узле E дерево сбалансировано**, так как оба его поддерева пустые и имеют одинаковую высоту 0.

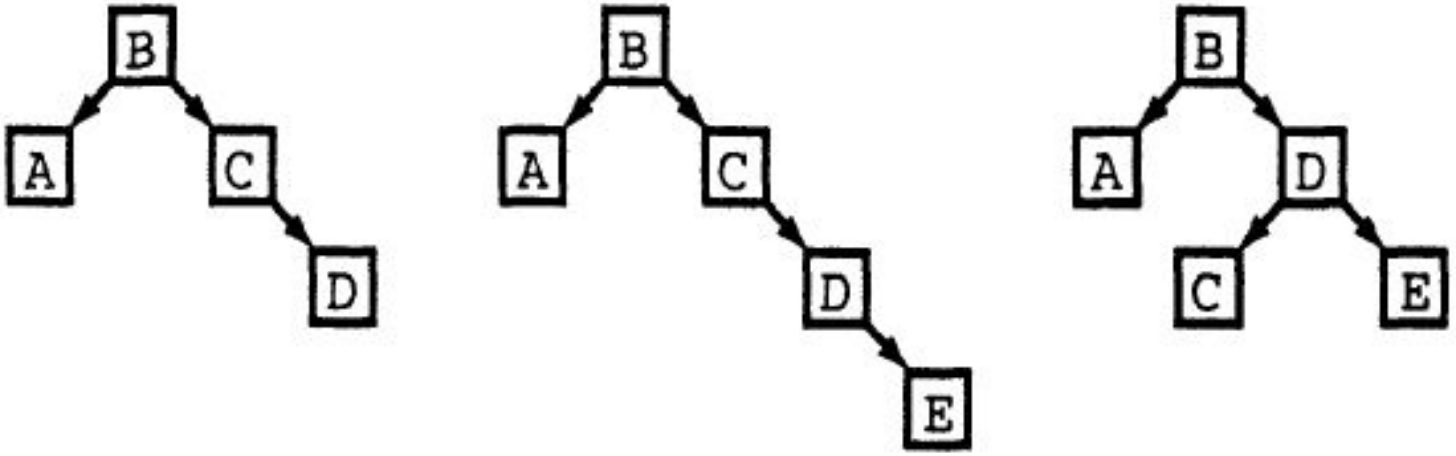
В узле D дерево также сбалансировано, так как его левое поддерево пустое, и имеет поэтому высоту 0. Правое поддерево содержит единственный узел E, и поэтому его высота равна 1.



Добавление узла к AVL-дереву

В узле C дерево уже не сбалансировано. Левое поддерево узла C имеет высоту 0, а правое — высоту 2. Эти поддеревья можно сбалансировать, как показано на рисунке справа, при этом узел **C заменяется узлом D. Теперь поддерево с корнем в узле D имеет высоту 2**. Заметим, что **высота поддерева с корнем в узле C до вставки нового узла, которое ранее находилось в этом месте, также была равна 2**.

Так как высота поддерева не изменилась, то дерево также окажется сбалансированным во всех узлах выше D. (Балансировка: левое вращение.)



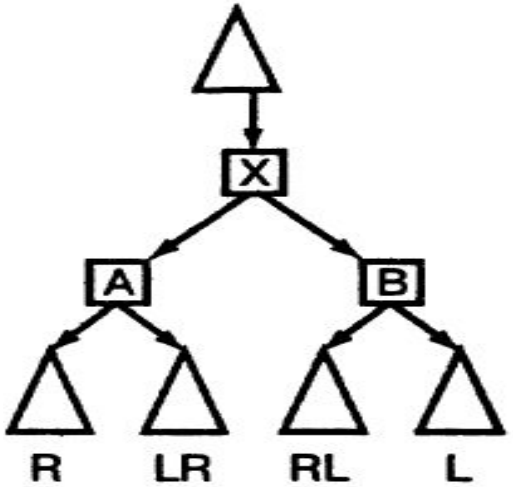
Вращения АВЛ-дерева

При вставке узла в АВЛ-дерево, в зависимости от того, в какую часть дерева добавляется узел, **существует четыре варианта балансировки**. Эти способы называются **правым** (малое правое) и **левым** (малое левое) **вращением**, и **вращением влево-вправо** (большое правое вращение) и **вправо-влево** (большое левое вращение), и обозначаются **R, L, LR и RL** соответственно.

Предположим, что в АВЛ-дерево вставляется новый узел, и теперь дерево становится несбалансированным в узле X, как показано на рисунке. Остальные части дерева обозначены треугольниками, так как их не требуется рассматривать подробно.

Новый узел может быть вставлен в любое из четырех поддеревьев узла X, изображенных в виде треугольников. Если вставляется узел в одно из поддеревьев, то для балансировки дерева потребуется выполнить соответствующее вращение.

Балансировка не нужна, если вставка нового узла не нарушает свойство AVL-дерева.

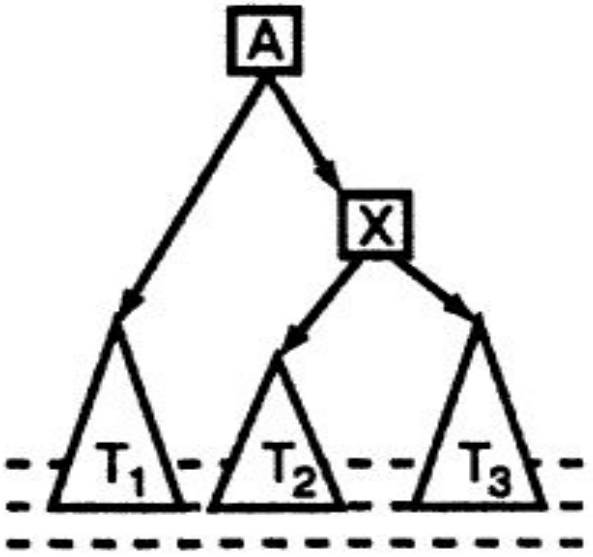


Правое вращение R (малое правое вращение)

Новый **узел вставляется в поддерево R (на рисунке - T1)**. В этом случае **не нужно изменять два поддерева правого дочернего узла X**, их можно изобразить одним треугольником.

При этом поддерево TA с корнем в узле A становится по крайней мере на два уровня выше, чем поддерево T3, то есть balance(X)= h(TR) - h(TL) =-2.

(Поскольку до вставки нового узла дерево было АВЛ-деревом, то TA должно было быть выше поддерева T3 не больше, чем на один уровень. После вставки одного узла TA должно быть выше поддерева T3 ровно на два уровня.)



Правое вращение R (малое правое вращение)

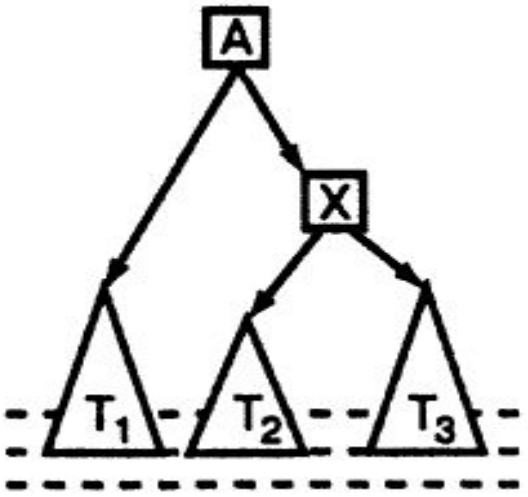
Также известно, что **поддерево T1 выше поддерева T2 не больше, чем на один уровень**. Иначе бы узел A, а не узел X был самым нижним узлом с несбалансированными поддеревьями. (Если бы T1 было на два уровня выше, чем T2, то дерево было бы несбалансированным уже в узле A.)

В этом случае, **можно переупорядочить узлы при помощи правого вращения R** (**right rotation**), как показано на рисунке справа. Это вращение называется **правым**, так как **узлы A и X** как бы вращаются вправо (относительно узла A).

Малое правое вращение используется тогда, когда

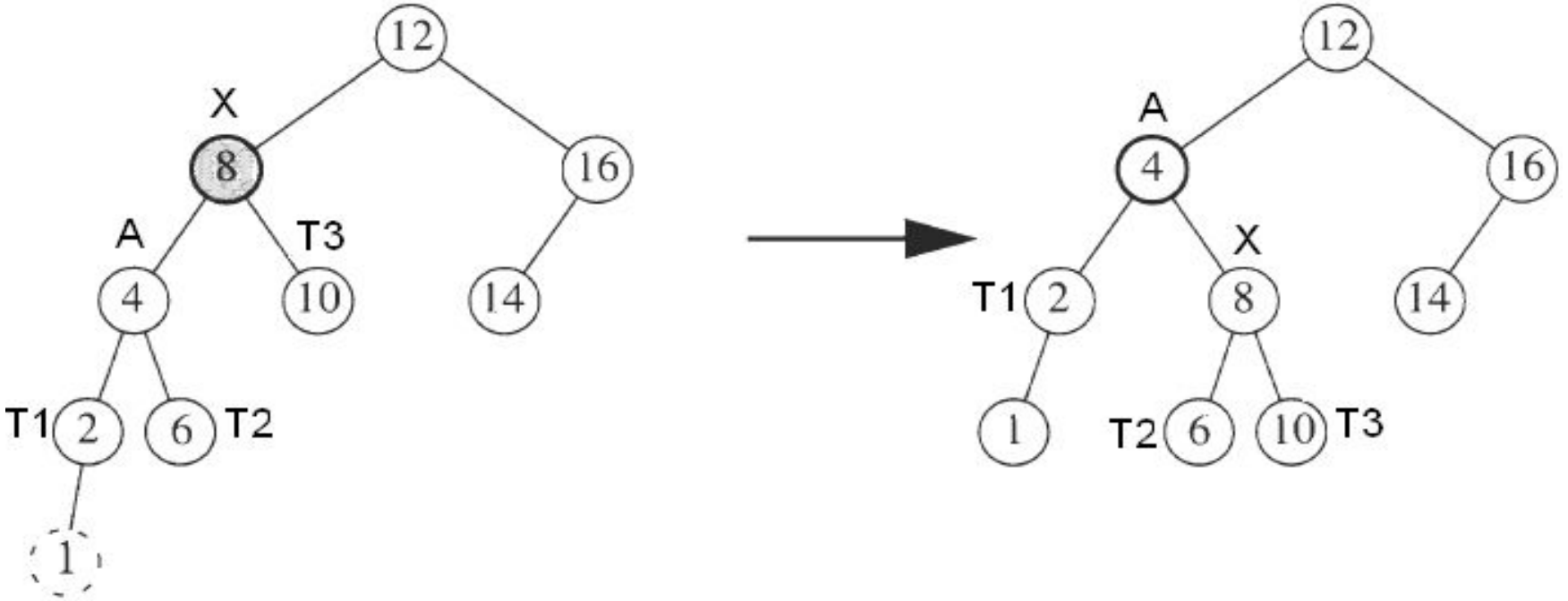
h(XR)-h(XL)=-2 и h(AR)< h(AL)**, то есть**

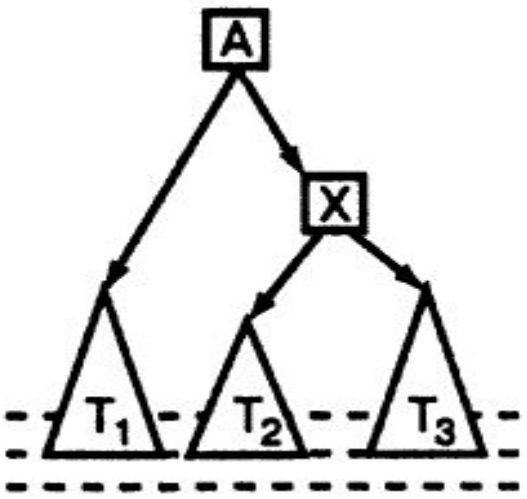
balance(X)=-2 и balance(XL)=-1.



Правое вращение R (малое правое вращение)

Пример 1. Правое вращение R (малое правое вращение). Выполняется, когда “перевес” идет по пути L-L от узла с нарушенной балансировкой.





Балансировка вершины

Определение. Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, **изменяет связи предок-потомок в поддереве данной вершины** так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

Утверждение. Это **вращение** (в данном случае, правое) **сохраняет порядок «меньше» расположения узлов дерева**.

При симметричном обходе любого из таких деревьев **обращение ко всем** поддеревьям и **узлам дерева происходит в одном и том же порядке** (для правого вращения - T1, A, T2, X, T3). Следовательно и порядок расположения элементов в них будет одинаковым.

Утверждение. Высота поддерева, с которым мы работаем, остается неизменной. Все части дерева, лежащие выше узла X при этом также остаются сбалансированными, поэтому не требуется продолжать балансировку дерева дальше.

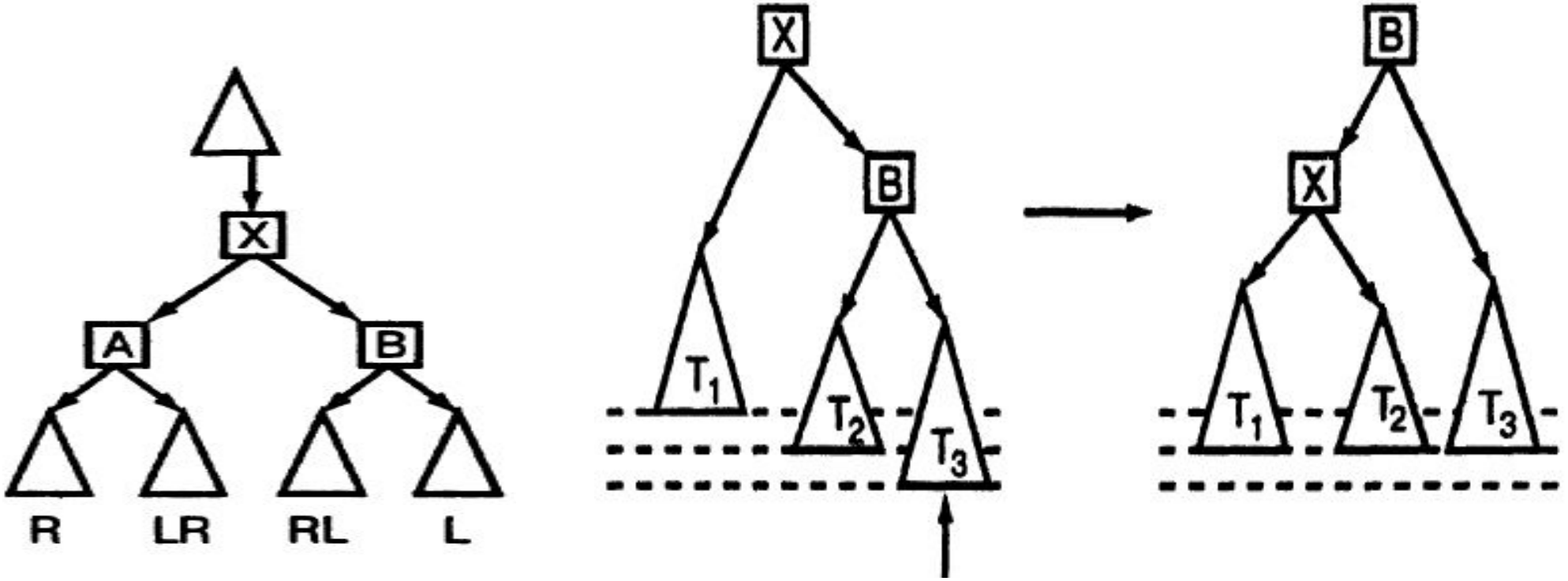
Левое вращение L (малое левое вращение)

Левое вращение *L* (**left rotation**) выполняется аналогично правому. Оно используется, если новый узел вставляется в поддерево L (T3), показанное на рисунке.

Малое левое вращение используется тогда, когда

h(XR)-h(XL)=2 и h(BL)< h(BR)**, то есть**

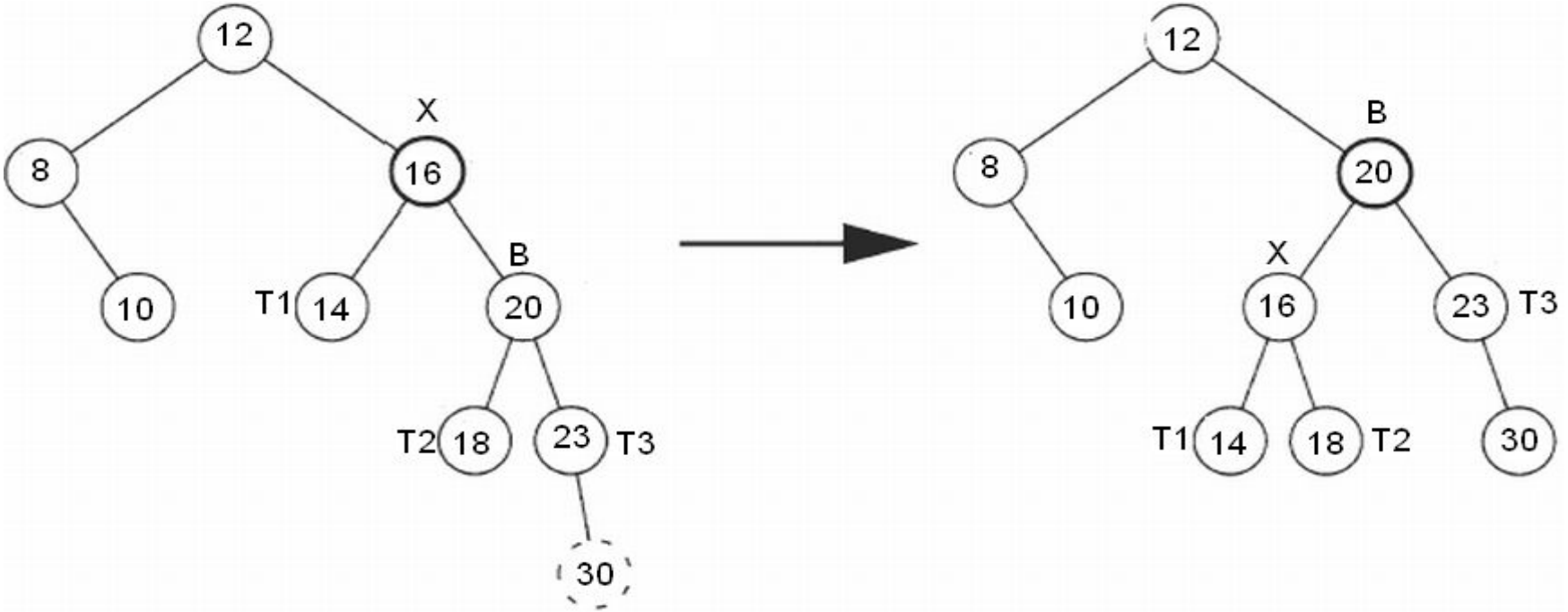
balance(X)=2 и balance(XR)=1.

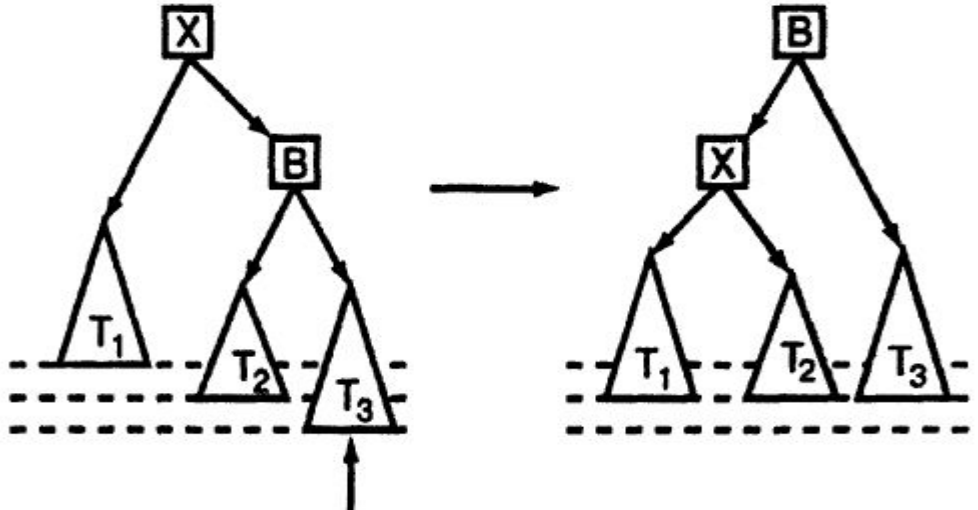


Левое вращение L (малое левое вращение)

Пример 2. Левое вращение L (малое левое вращение).

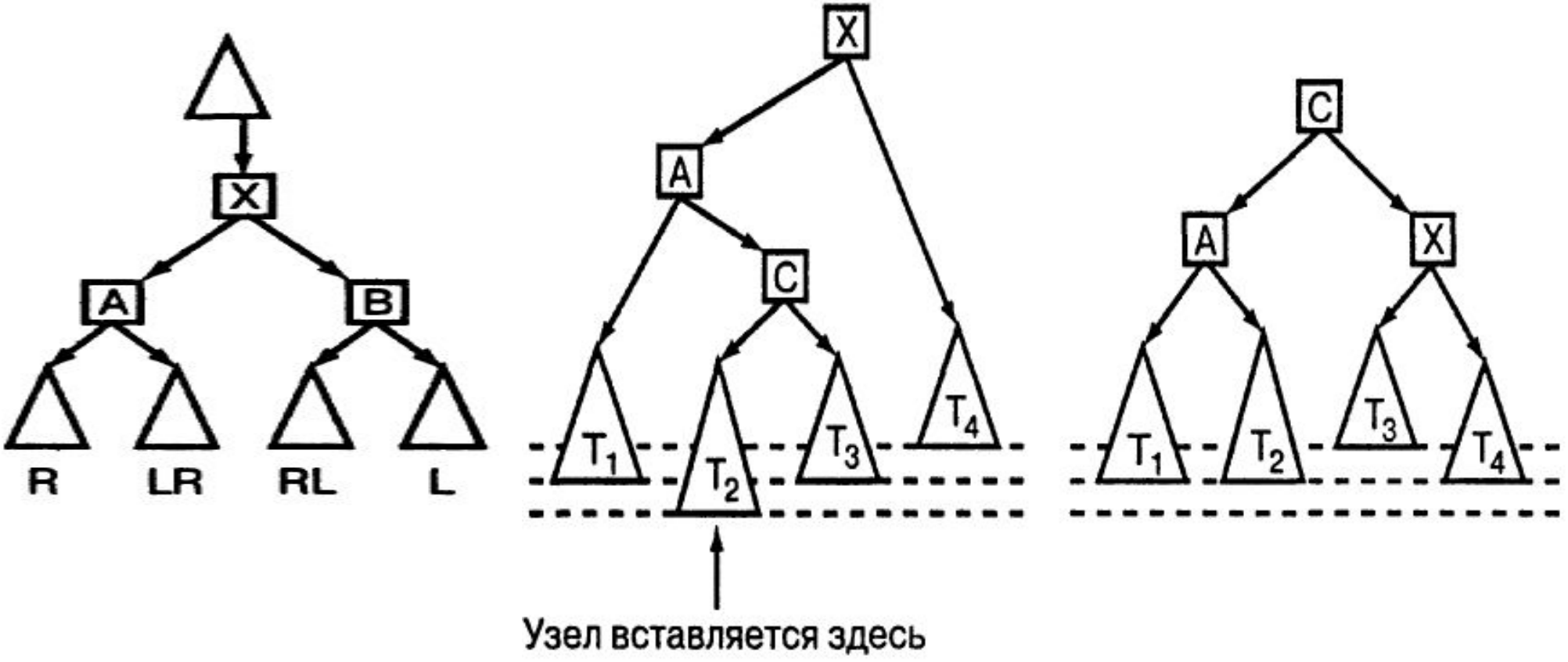
Выполняется, когда “перевес” идет по пути R-R от узла с нарушенной балансировкой.





Вращение влево-вправо LR (большое правое)

На рисунке показано дерево, в котором новый **узел вставляется в левую часть T2 поддерева LR**. Так же легко можно вставить узел в правое поддерево T3. В обоих случаях, поддеревья TA и TC останутся АВЛ-поддеревьями, но поддерево TX уже не будет таковым.



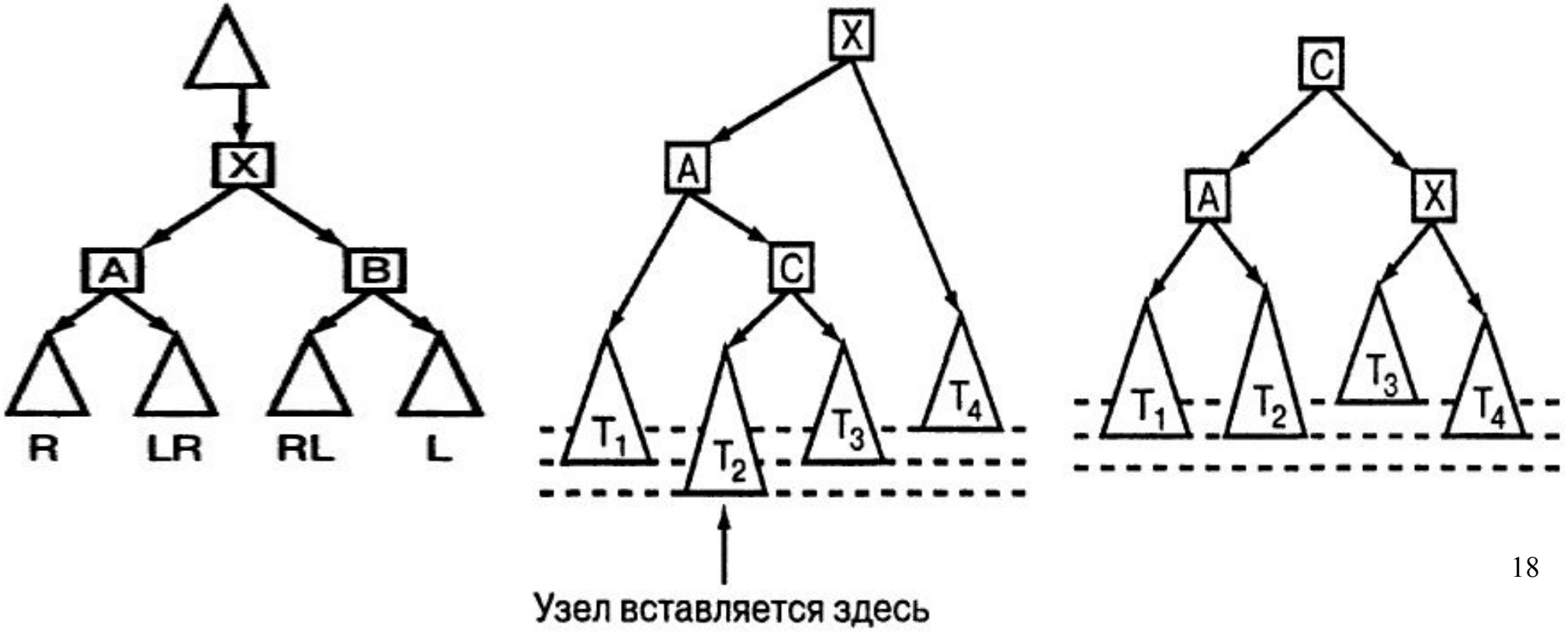
Вращение влево-вправо LR (большое правое)

Применяемое вращение называется **вращением влево-вправо** (**left-right** rotation), так как **при этом вначале узлы A и C как бы вращаются влево** (относительно узла A), **а затем узлы C и X вращаются вправо** (относительно узла C, занявшего место узла A).

Большое правое вращение используется тогда, когда

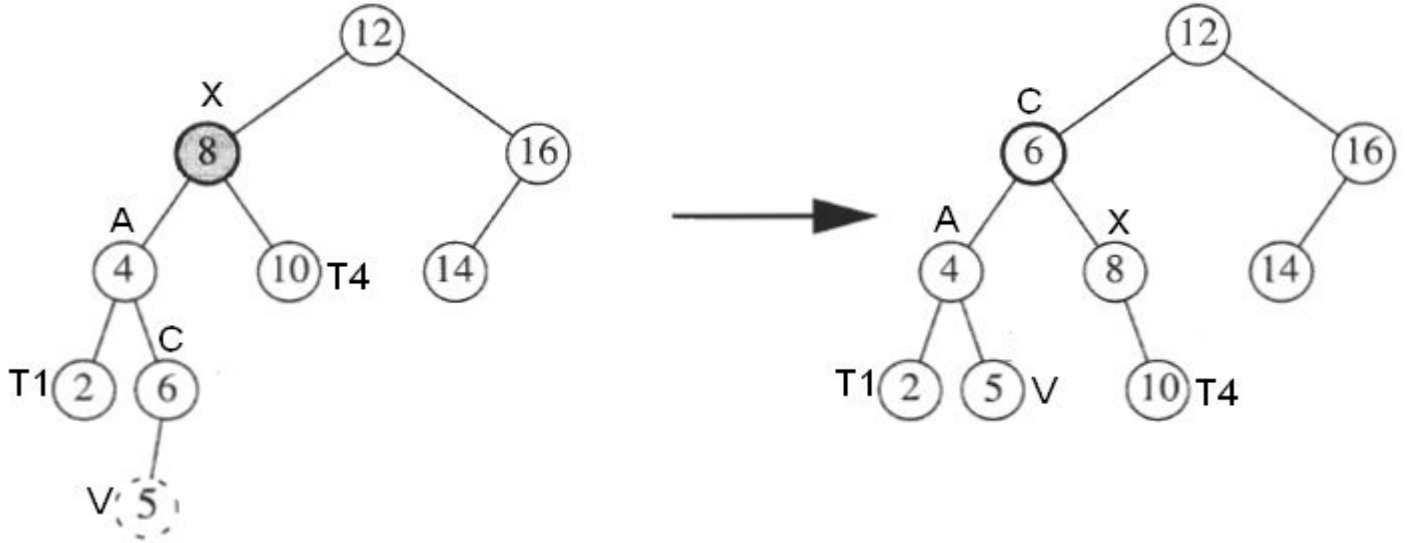
h(XR)-h(XL)=-2 и h(AR)>h(AL), то есть

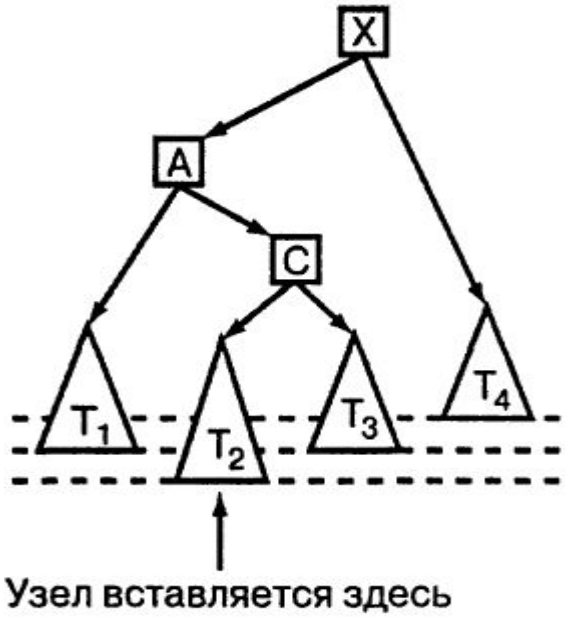
balance(X)=-2 и balance(XL)=1.

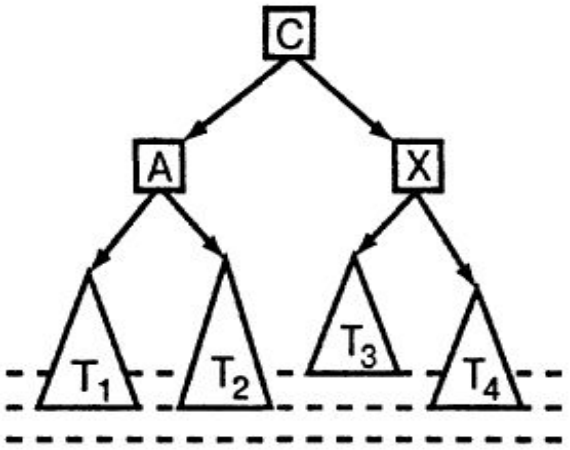


Вращение влево-вправо LR (большое правое)

Пример 3. Вращение влево-вправо LR (большое правое вращение). Выполняется, когда “перевес” идет по пути L-R от узла с нарушенной балансировкой.







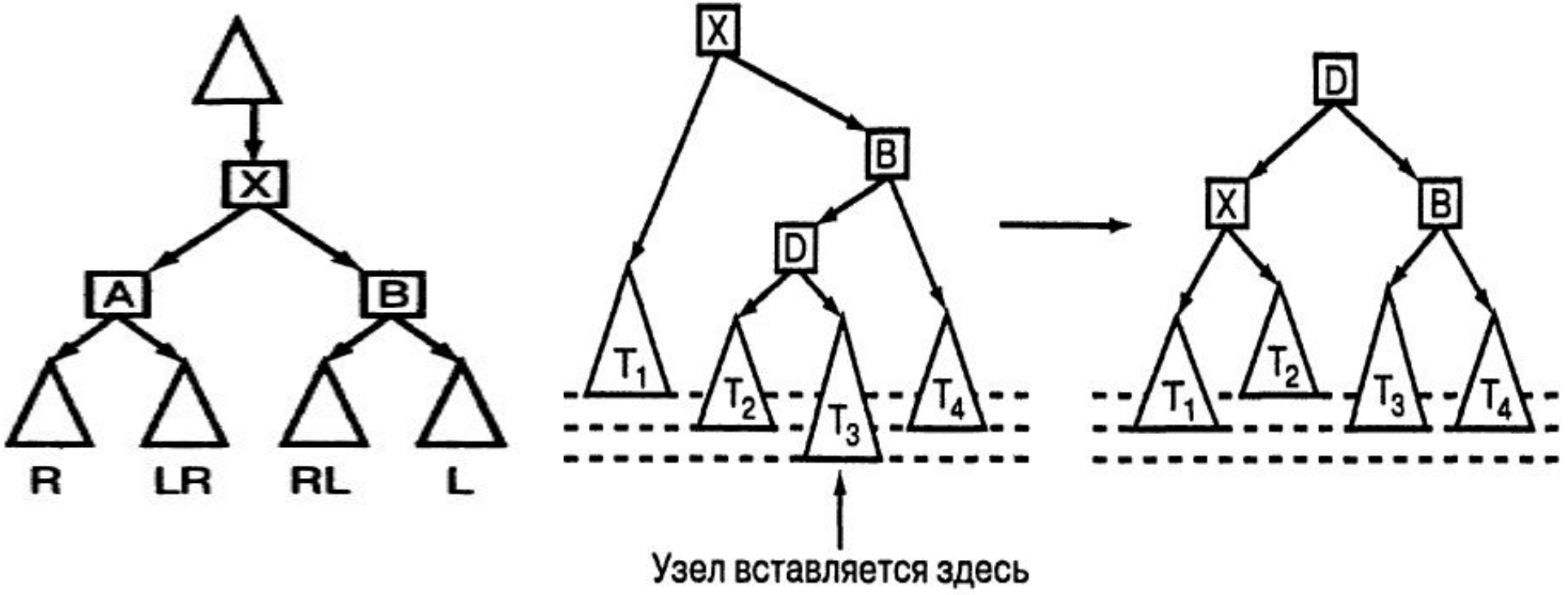
Вращение вправо-влево RL (большое левое)

Вращение **вправо-влево** (**right-left rotation**) аналогично вращению влево-вправо. Оно используется для балансировки дерева после **вставки узла в поддерево RL**. На рисунке показано АВЛ-дерево до и после вращения вправо-влево.

Большое левое вращение используется тогда, когда

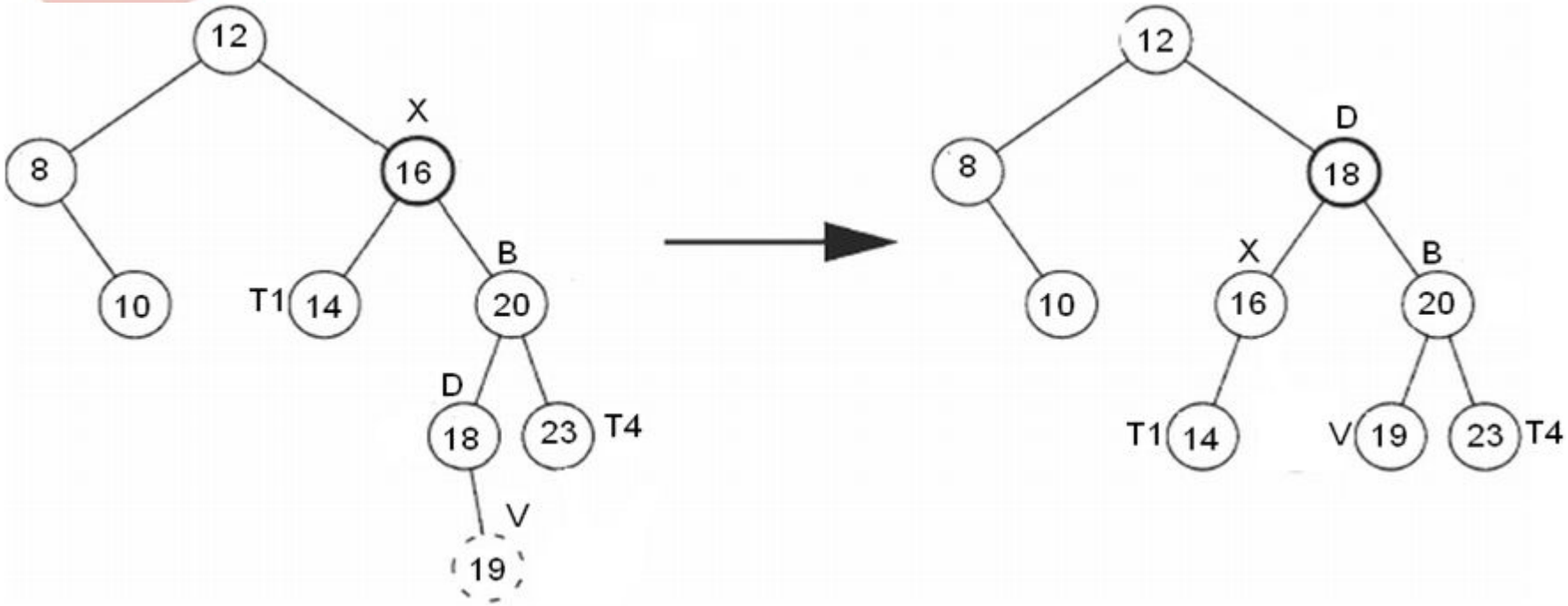
h(XR)- h(XL)=2 и h(BR)<h(BL), то есть

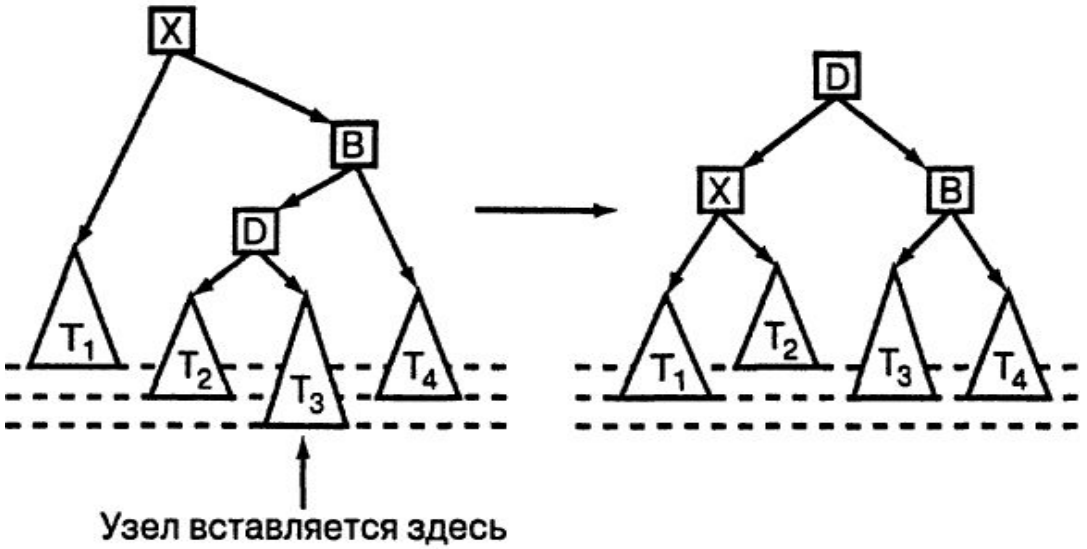
balance(X)=2 и balance(XR)=-1.



Вращение вправо-влево RL (большое левое)

Пример 4. Вращение вправо-влево RL (большое левое вращение). Выполняется, когда “перевес” идет по пути R-L от узла с нарушенной балансировкой.





Вставка узлов в АВЛ-дерево

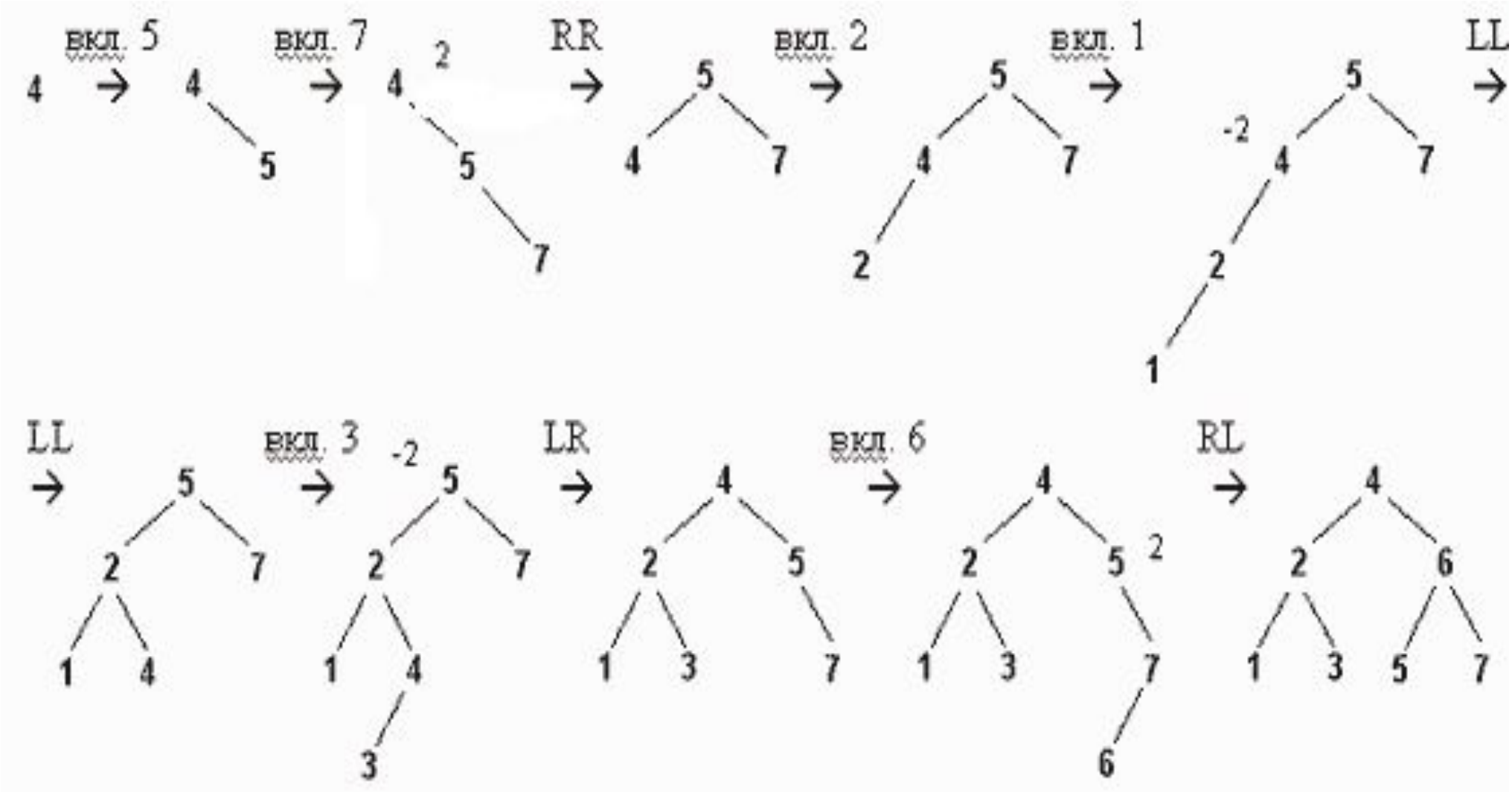
Пример 5. Постройте диаграмму роста АВЛ-дерева, получающегося из последовательности значений 4, 5, 7, 2, 1, 3, 6 (укажите тип применяемых поворотов и показатель сбалансированности у узлов с нарушенным балансом).

Пример 6. Постройте диаграмму роста АВЛ-дерева, получающегося из последовательности значений 2, 4, 5, 6, 7, 8, 9.

Пример 7. Постройте диаграмму роста АВЛ-дерева, получающегося из последовательности значений 1, 6, 5, 2, 3, 4.

Вставка узлов в АВЛ-дерево

Пример 5. Рассмотрим диаграмму роста АВЛ-дерева, получающегося из последовательности значений 4, 5, 7, 2, 1, 3, 6 (будем указывать тип применяемых поворотов и показатель сбалансированности у узлов с нарушенным балансом).



Виды вращений

Пусть в АВЛ-дерево вставляется новый узел, и дерево становится несбалансированным в узле X; A=XL, B=XR.

Малое правое вращение R используется тогда, когда h(XR)-h(XL)=-2 и h(AR)< h(AL), то есть **balance(X)=-2 и balance(XL)=-1.** “Перевес” идет по пути L-L от узла с нарушенной балансировкой.

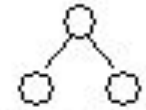
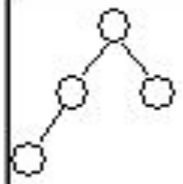
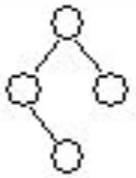
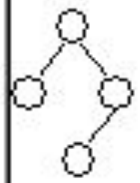
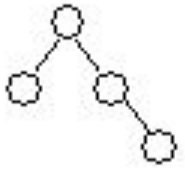
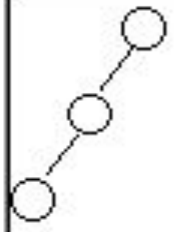
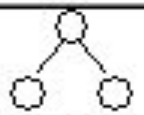
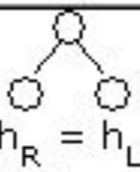
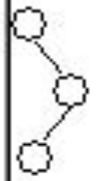
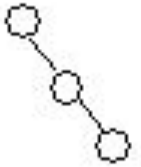
Малое левое вращение L используется тогда, когда h(XR)-h(XL)=2 и h(BL)< h(BR), то есть **balance(X)=2 и balance(XR)=1.** “Перевес” идет по пути R-R от узла с нарушенной балансировкой.

Большое правое вращение LR используется тогда, когда h(XR)-h(XL)=-2 и h(AR)>h(AL), то есть **balance(X)=-2 и balance(XL)=1.** “Перевес” идет по пути L-R от узла с нарушенной балансировкой.

Большое левое вращение RL используется тогда, когда h(XR)-h(XL)=2 и h(BR)<h(BL), то есть **balance(X)=2 и balance(XR)=-1.** “Перевес” идет по пути R-L от узла с нарушенной балансировкой.

Виды балансировок

Показатель сбалансированности (для узла X) при включении в АВЛ-дерево

До включения (дерево АВЛ-сбалансировано)	После включения	
	В левое поддерево	В правое поддерево
 $h_R = h_L$ $h_R - h_L = 0$	 или $h_R < h_L$ $h_R - h_L = -1$  критерий сбалансированности не нарушен	 или $h_R > h_L$ $h_R - h_L = 1$  критерий сбалансированности не нарушен
 $h_R < h_L$ $h_R - h_L = -1$	 или $h_R < h_L$ $h_R - h_L = -2$  критерий сбалансированности нарушен, нужна балансировка	 $h_R = h_L$ $h_R - h_L = 0$ критерий сбалансированности не нарушен
 $h_R > h_L$ $h_R - h_L = 1$	 $h_R = h_L$ $h_R - h_L = 0$ критерий сбалансированности не нарушен	 или $h_R > h_L$ $h_R - h_L = 2$  критерий сбалансированности нарушен, нужна балансировка

Виды балансировок

Таким образом, есть 4 варианта нарушения балансировки:

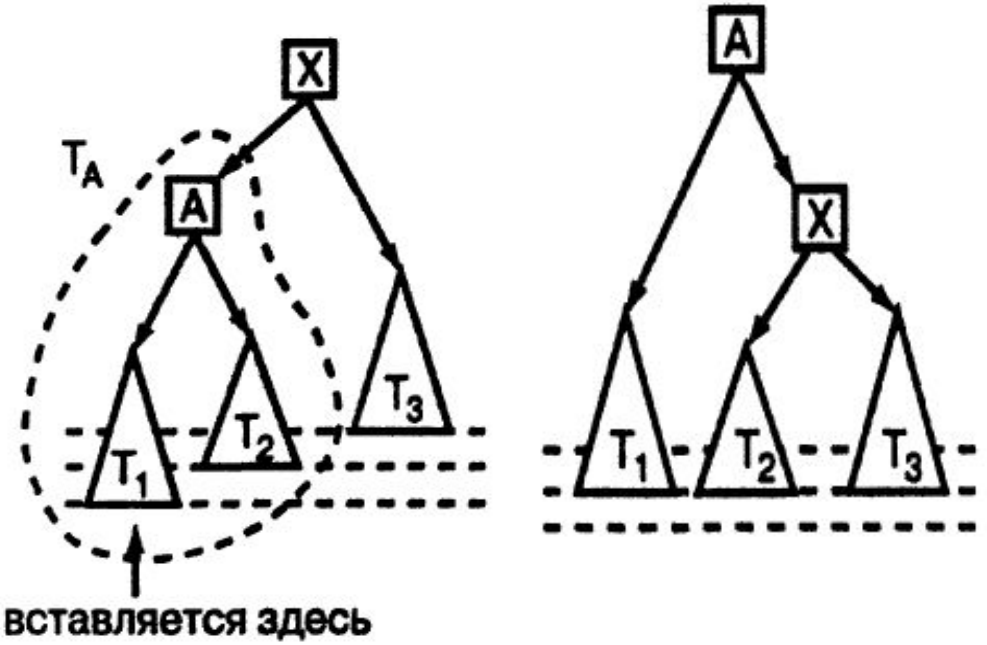
Балансировка выполняется с помощью действий, называемых поворотами узлов (вращениями). Рассмотрим алгоритмы поворотов, используя указатели p, p1, p2 и считая, что **p указывает на узел с нарушенной балансировкой.**

При включении узлов повороты выполняются для ближайшего узла к включенному с нарушенной балансировкой. То есть если после включения узла в дереве образуется несколько узлов с нарушенной балансировкой, поворот выполняется для того, который находится ниже (то есть ближе к включенному). После балансировки этого узла восстанавливается баланс и выше расположенных узлов.

Виды балансировок

Одинарный LL-поворот (вращение вправо R). Выполняется, когда ‘перевес’ идет по пути L-L от узла с нарушенной балансировкой.

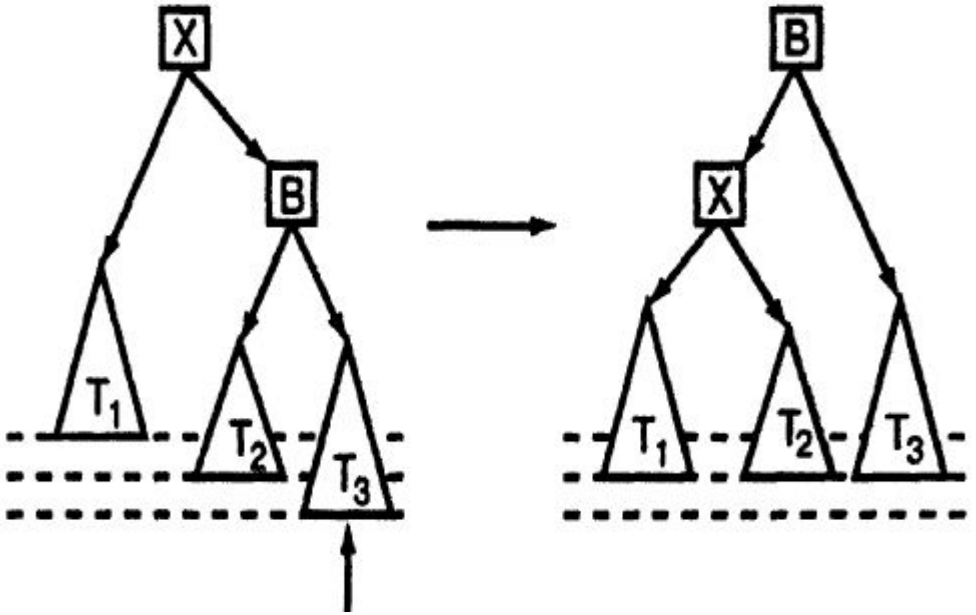

```
p1 = p -> llink;//
p->llink = p1->rlink; // T2
p1->rlink = p;
p = p1;
```



Виды балансировок

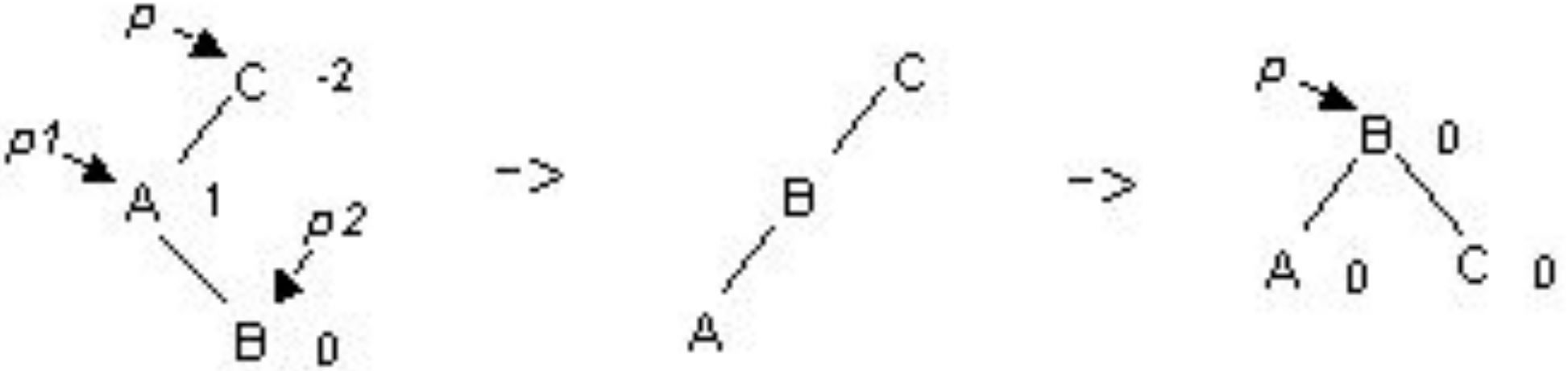
Одинарный RR-поворот. Выполняется, когда ‘перевес’ идет по пути R-R от узла с нарушенной балансировкой.


```
p1 = p -> rlink;
p -> rlink = p1 -> llink; // T2
p1 -> llink = p;
p = p1;
```

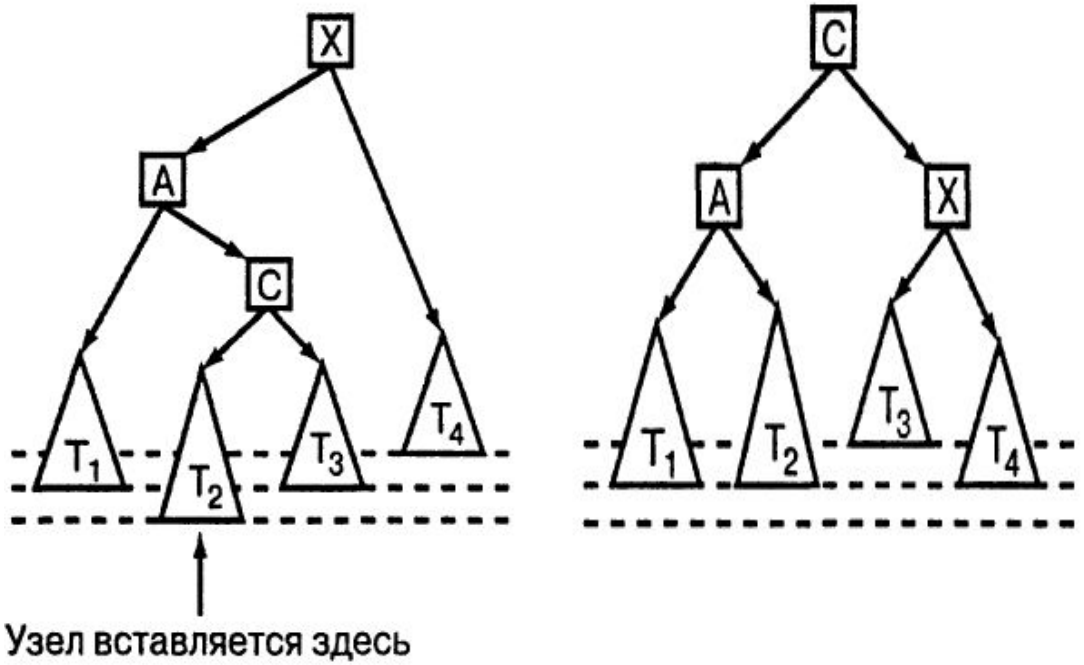


Виды балансировок

Двойной LR-поворот. Выполняется, когда ‘перевес’ идет по пути L-R от узла с нарушенной балансировкой.

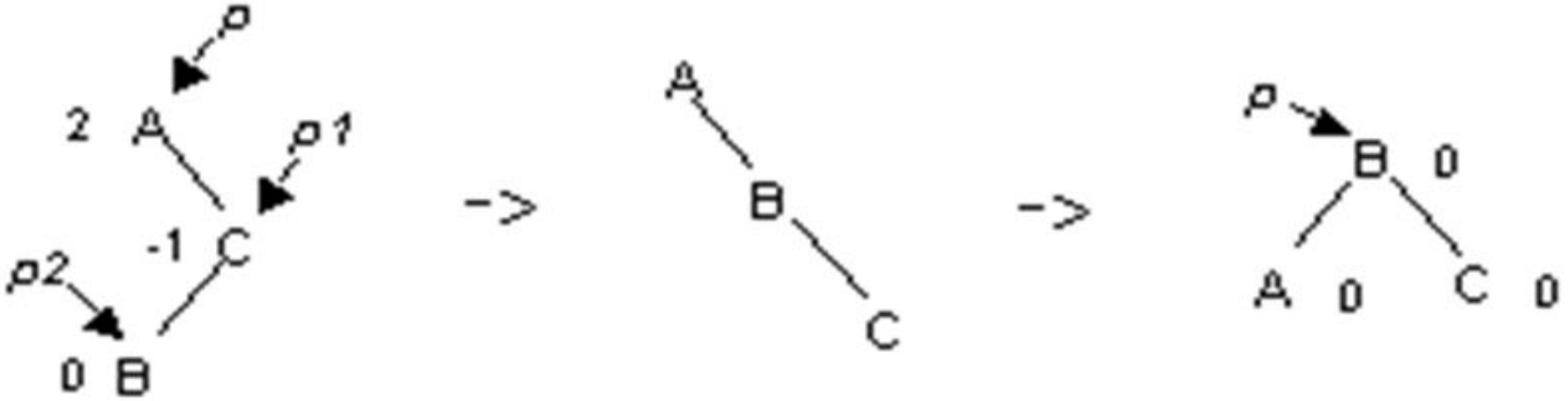



```
p1 = p->llink;//A
p2 = p1->rlink;//
p1->rlink = p2->llink;// T2
p2->llink = p1;
p->llink = p2->rlink;//T3
p2->rlink = p;
p = p2;
```

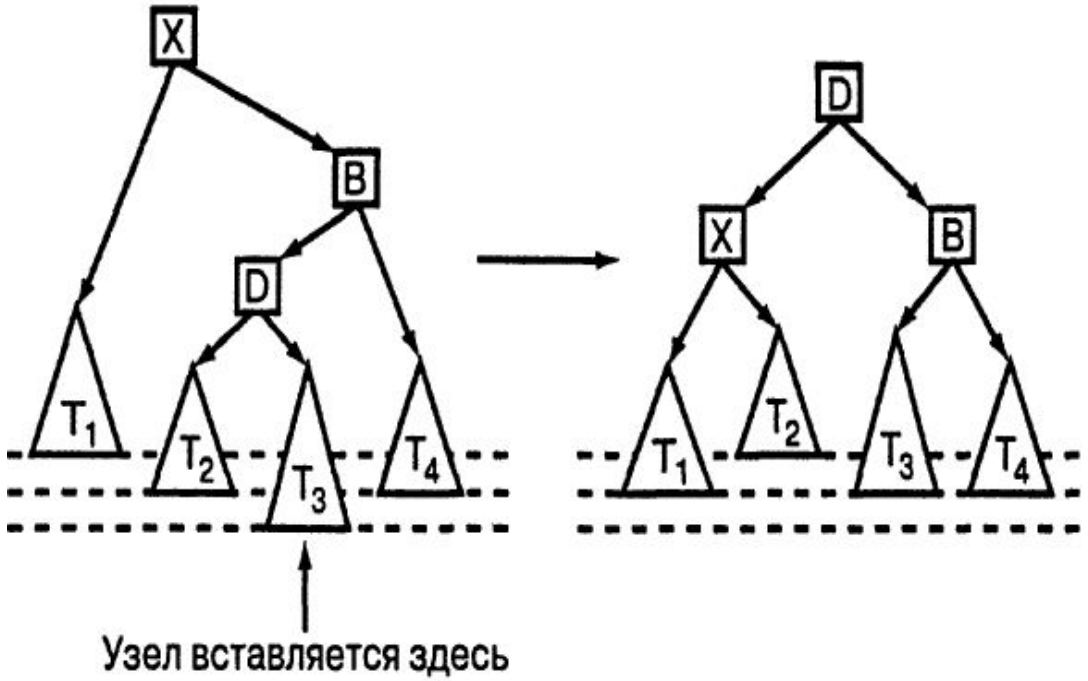


Виды балансировок

Двойной RL-поворот. Выполняется, когда ‘перевес’ идет по пути R-L от узла с нарушенной балансировкой.




```
p1 = p->rlink;
p2 = p1->llink;
p1->llink = p2->rlink;//T3
p2->rlink = p1;
p->rlink = p2->llink;//T2
p2->llink = p;
p = p2;
```



Вставка узлов в АВЛ-дерево

Процедура вставки предложенная Н.Виртом

Показатель сбалансированности (баланс) в ней **интерпретируется как разность между высотой левого и правого поддерева (**hl - hr). В алгоритме используется тип TAVLNode (описан ниже). **Непосредственно при вставке листу присваивается нулевой баланс**.

Процесс добавления вершины состоит из трех частей:

1. Проход по пути поиска, пока не убедимся, что ключа (Id) в дереве нет.

2. Включение новой вершины в дерево и определение результирующих показателей балансировки.

3. "Отступление" назад **по пути поиска** и проверка в каждой вершине показателя сбалансированности. Если необходимо - балансировка.

Вставка узлов в АВЛ-дерево

В обычную процедуру вставки включен еще один параметр-переменная Grew (boolean), означающий, что высота дерева увеличилась.

Включение вершины в левое поддерево

Пусть процесс из левой ветви возвращается к родителю (рекурсия идет назад), тогда возможны три случая.

1. hl < hr: выравняется hl = hr. Ничего делать не нужно.

2. hl = hr: теперь левое поддерево будет больше на единицу, но балансировка пока не требуется.

3. hl > hr: теперь hl - hr = 2 - требуется балансировка.

В третьей ситуации требуется определить балансировку левого поддерева. Если левое поддерево этой вершины (TAVLNode^.left^.left) выше правого (TAVLNode^.left^.right), то хватит и малого правого вращения, иначе требуется большое правое вращение.

Включение в правое поддерево

Проводятся аналогичные (симметричные) рассуждения.

Вставка узлов в АВЛ-дерево

Оценка эффективности алгоритма вставки

Г.М.Адельсон-Вельский и Е.М.Ландис доказали теорему, согласно которой **высота АВЛ-дерева с N внутренними вершинами заключена между log_2(N+1) и 1.4404*log_2(N+2)-0.328**, то есть **высота АВЛ-дерева никогда не превысит высоту идеально сбалансированного дерева более, чем на 45%**.

Для больших N имеет место **оценка 1.04*log_2(N)**. Таким образом, выполнение основных операций 1 – 3 требует порядка log_2(N) сравнений.

Затраты на балансировку AVL - дерева значительны. Экспериментально выяснено, что **одна балансировка приходится на каждые два включения** и **на каждые пять исключений**, **однократный и двукратный повороты равновероятны**.

Из-за сложности операций балансировки AVL - деревья рекомендуется использовать в случае, если **число операций поиска значительно превышает число операций включения и удаления**.

Вставка узлов в АВЛ-дерево

Программное добавление узлов

Кроме обычных полей LeftChild и RightChild, класс AVLNode содержит также поле Balance, которое указывает, которое из поддеревьев узла длиннее. Его значение равно:

LeftHeavy, если левое поддерево длиннее,
RightHeavy, если длиннее правое, и
Balanced, если оба поддерева имеют одинаковую глубину.

```
type
 TBalance = (LeftHeavy, Balanced, RightHeavy) ;
 TAVLNode = class (TObject)
 private
 public
    Id : Integer;
    LeftChild, RightChild : TAVLNode;
    Balance : TBalance;
    Position : TPoint;
 // Код опущен...
 end;
```

Вставка узлов в АВЛ-дерево

Программное добавление узлов

Сначала процедура AddNode, рекурсивно спускается вниз по дереву в поисках места для нового элемента. Когда она доходит до нижнего уровня дерева, то создает новый узел и добавляет его в дерево.

Затем процедура AddNode использует восходящую рекурсию для балансировки дерева. При каждом из рекурсивных вызовов процедуры, она движется назад по дереву. **При каждом возврате она устанавливает параметр grew в значение True, если увеличилась высота поддерева, которое она покидает**. Процедура использует этот параметр для определения того, является ли проверяемое поддерево **несбалансированным**. Если это так, то процедура применяет для балансировки дерева соответствующее вращение.

Вставка узлов в АВЛ-дерево

Предположим, что процедура в настоящий момент исследует узел X.

A. **Допустим, что она перед этим обращалась к его правому поддереву и параметр grew (для корня правого поддерева) равен true** (правое поддерево стало выше).

1. **Если поддеревья узла X до этого имели одинаковую высоту**, то правое поддерево станет теперь выше левого. Поддерево в точке X сбалансировано, но оно выросло, так как выросло его правое поддерево – **grew для X равен true**.

2. **Если левое поддерево узла X вначале было выше, чем правое, то левое и правое поддеревья теперь будут иметь одинаковую высоту.** Высота поддерева с корнем **в узле X** не изменилась — она по-прежнему равна высоте левого поддерева плюс 1. **В этом случае процедура AddNode установит значение переменной grew равным false,** указывая, что его высота не изменилась (дерево сбалансировано).

3. Наконец, **если правое поддерево узла X было первоначально выше левого**, то вставка нового узла делает дерево **несбалансированным** в узле X. **Процедура AddNode вызывает процедуру RebalanceRigthGrew** для перебалансировки дерева. Процедура RebalanceRigthGrew выполняет левое вращение или вращение вправо-влево, в зависимости от ситуации.

Вставка узлов в АВЛ-дерево

B. Если **новый элемент вставляется в левое поддерево**, то процедура AddNode работает по похожему алгоритму и вызывает аналогичную процедуру RebalanceLeftGrew.

Вставка узлов в АВЛ-дерево

```
//Д.1
procedure TAVLTree.AddNode (var parent : TAVLNode; new_id : Integer;
        var grew : Boolean);
 begin
 // Если это основание дерева, то создаем новый узел и заставляем
 // родительский узел указывать на новый.
 if  parent = nil  then
 begin
    parent := TAVLNode.Create;
    parent.Id := new_id;
    parent.Balance := Balanced;
    grew := True;
    exit;
end;
```

Вставка узлов в АВЛ-дерево

```
//Д.2
// Продолжаем двигаться вниз по соответствующему поддереву.
if  new_id<=parent.Id  then
 begin
 // Вставка дочернего узла в левое поддерево.
    AddNode(parent.LeftChild,new_id,grew);
 // Нужна ли перебалансировка?
    if  not grew  then exit;
    if  parent.Balance = RightHeavy  then
    begin
 // Правое поддерево было длиннее. Левое поддерево выросло,
 // поэтому дерево сбалансировано.
       parent.Balance := Balanced;
       grew := False;
    end
```

Вставка узлов в АВЛ-дерево

```
//Д.3
else if parent.Balance = Balanced  then
 begin
 // Был баланс. Левое поддерево выросло,
 // поэтому левое поддерево длиннее. Дерево все еще
 // сбалансировано, но оно выросло, поэтому необходимо
 // продолжить проверку баланса еще выше.
     parent.Balance := LeftHeavy;
 end else begin // parent.Balance =LeftHeavy
 // Левое поддерево длиннее. Оно выросло, поэтому имеем
 // разбалансированное дерево слева. Необходимо выполнить
 // соответствующее вращение для перебалансирования.
     RebalanceLeftGrew(parent);
     grew := False;
 end;
end // Конец проверки баланса родительского узла.
```

Вставка узлов в АВЛ-дерево

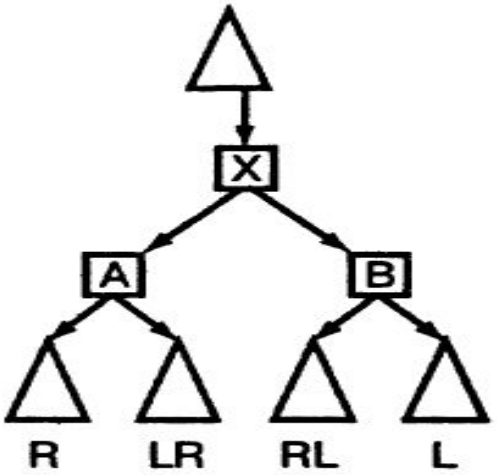
```
//Д.4
else begin  // new_id>parent.Id
 // Вставка дочернего узла в правое поддерево.
    AddNode(parent.RightChild,new_id,grew);
 // Нужна ли перебалансировка?
    if (not grew) then exit;
    if (parent.Balance = LeftHeavy) then
 begin
 // Левое поддерево было длиннее. Правое поддерево выросло,
 // поэтому дерево сбалансировано.
    parent.Balance := Balanced;
    grew := False;
 end
```

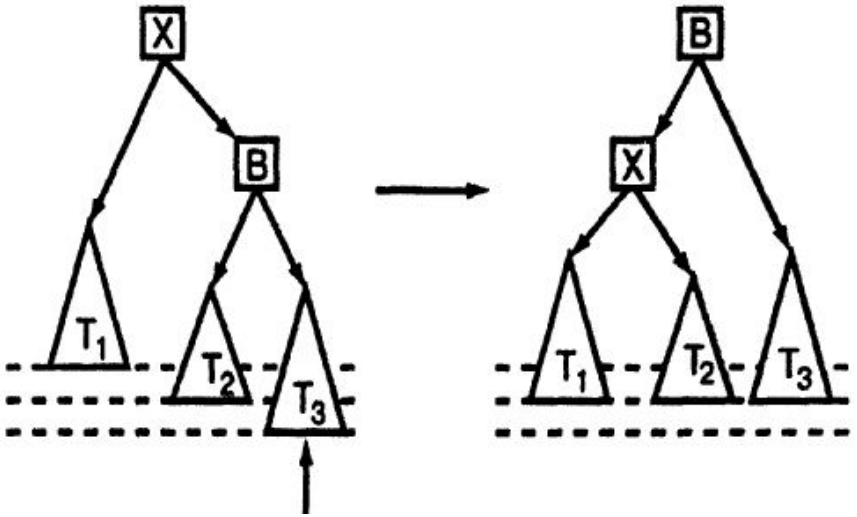
Вставка узлов в АВЛ-дерево

```
//Д.5
else if (parent.Balance = Balanced) then
 begin
 // Был баланс. Правое поддерево выросло,
 // поэтому оно длиннее. Дерево все еще сбалансировано,
 // но оно выросло, поэтому необходимо продолжить проверку
 // баланса еще выше.
    parent.Balance := RightHeavy;
 end else begin
 // Правое поддерево длиннее. Оно выросло, поэтому имеем
 // разбалансированное дерево справа. Необходимо выполнить
 // соответствующий сдвиг для перебалансирования.
    RebalanceRightGrew(parent);
    grew := false;
 end; // Конец проверки баланса родительского узла.
 end; // Конец if (левое поддерево) ... else (правое поддерево) ..
 end; // Конец procedure
```

Вставка узлов в АВЛ-дерево

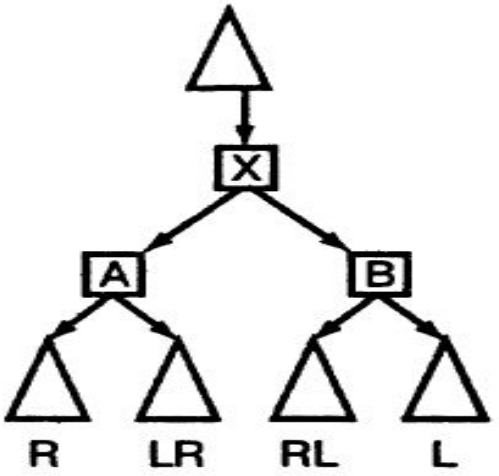
```
//Д.6
// Выполнение левого вращения или вращения вправо-влево
 // для перебалансирования дерева в данном узле.
 procedure TAVLTree.RebalanceRightGrew (var parent : TAVLNode);// parent - X
 var
    child, grandchild : TAVLNode;
 begin
    child := parent.RightChild; // B
    if (child.Balance= RightHeavy) then
    begin
// Вращение L - влево.
       parent.RightChild := child.LeftChild; // T2
       child.LeftChild := parent; // X
       parent.Balance := Balanced;
       parent := child; // B
    end
```

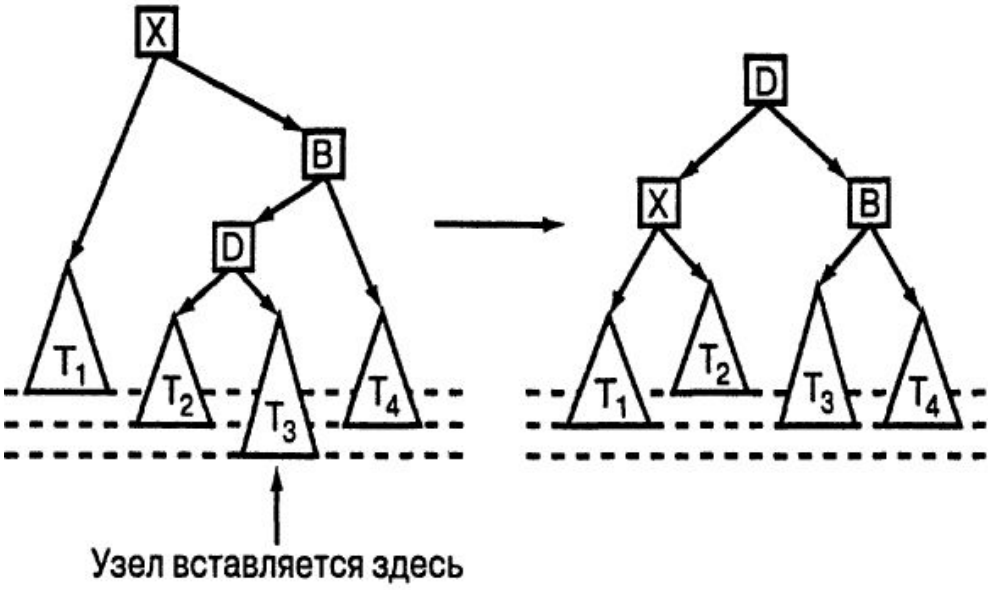




Вставка узлов в АВЛ-дерево

```
{Д.7} else begin
 // Вращение RL - вправо-влево.
    Grandchild := child.LeftChild; // D
    child.LeftChild := grandchild.RightChild; // T3
    grandchild.RightChild := child; // B
    parent.RightChild := grandchild.LeftChild; // T2
    grandchild.LeftChild := parent; // X
    if (grandchild.Balance=RightHeavy)
      then  parent.Balance := LeftHeavy
      else  parent.Balance := Balanced;
    if (grandchild.Balance=LeftHeavy)
      then  child.Balance := RightHeavy
      else  child.Balance := Balanced;
    parent := grandchild;
end; // Конец if ... else ...
parent.Balance := Balanced;
end; //Конец procedure
```





Удаление узла из АВЛ-дерева

При удалении узлов из AVL - дерева операция балансировки в основном остается такой же, что и при включении и заключается в однократном или двукратном повороте.

Балансировка выполняется с помощью тех же L - , R - , RL - и LR - поворотов.

Удаление элементов также имеет сложность O(log_2 n). Но если включение может вызвать один поворот, **удаление может потребовать повороты в каждом узле пути поиска при обратном ходе рекурсии**.

Эмпирические проверки показали, что если **при включении выполняется один поворот на каждые два включения**, то **при удалении один поворот приходится на 5 удалений.**

Удаление узла из АВЛ-дерева

Удаление узла из АВЛ-дерева производится так же, **как из обычного дерева поиска**, но **после этого** может возникнуть необходимость проведения **балансировки**.

Если удаляемый узел имеет один дочерний, то его можно заменить этим дочерним узлом, сохранив порядок расположения элементов дерева.

Если узел имеет два дочерних элемента, его нужно заменить крайним правым узлом в левом поддереве. Если у заменяющего узла существует левый потомок, то левый потомок занимает место заменяющего узла.

Поскольку **AVL-деревья - это один из видов упорядоченных деревьев**, потребуется выполнить те же самые шаги. Но после их завершения необходимо проверить баланс дерева.

Если найдется узел, где не выполняется свойство AVL, необходимо осуществить соответствующее вращение, чтобы перебалансировать дерево.

Хотя это те же самые вращения, использовавшиеся ранее для вставки узла в дерево, рассматриваемые случаи немного отличаются.

Удаление узла из АВЛ-дерева

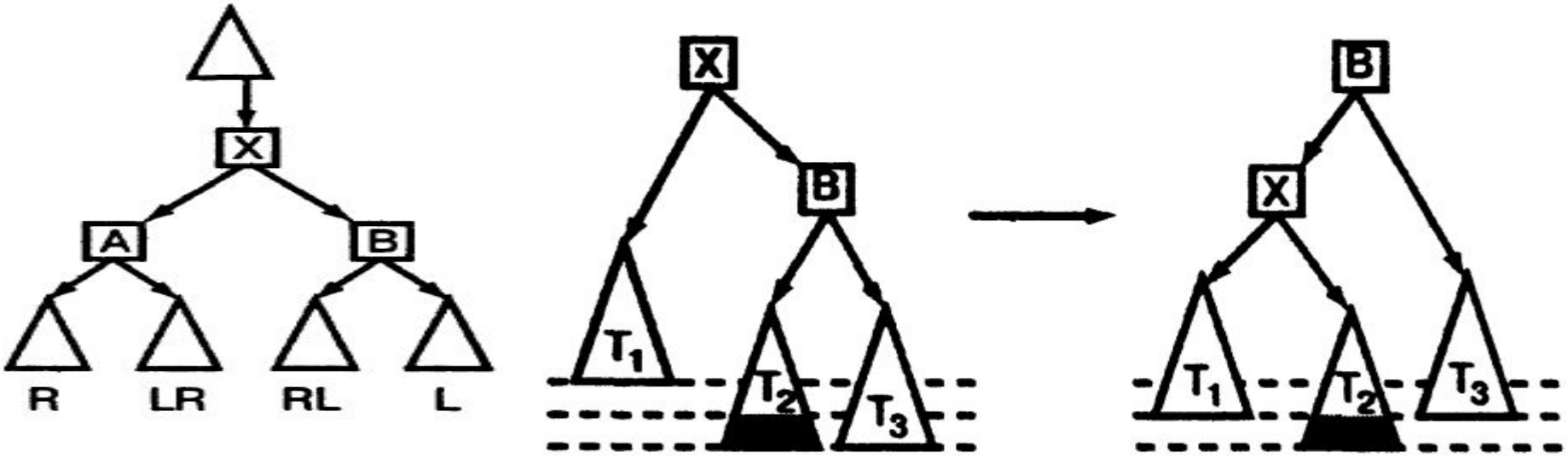
Левое вращение L

Удаляется узел **из левого поддерева T1 под узлом X**.

Допустим, что **правое поддерево либо точно сбалансировано, либо его правая половина имеет глубину на единицу больше, чем левая.**

Тогда **левое вращение** (**L**), показанное на рисунке, перебалансирует дерево в узле X.

Нижний закрашенный уровень поддерева T2 может существовать (поддерево Тв точно сбалансировано - T2 и T3 имеют одинаковую глубину), или отсутствовать (T3 длиннее T2 – перебалансировка уменьшит глубину дерева на 1, и **необходимо продолжить проверку выполнения свойства AVL для всех предков узла X**).



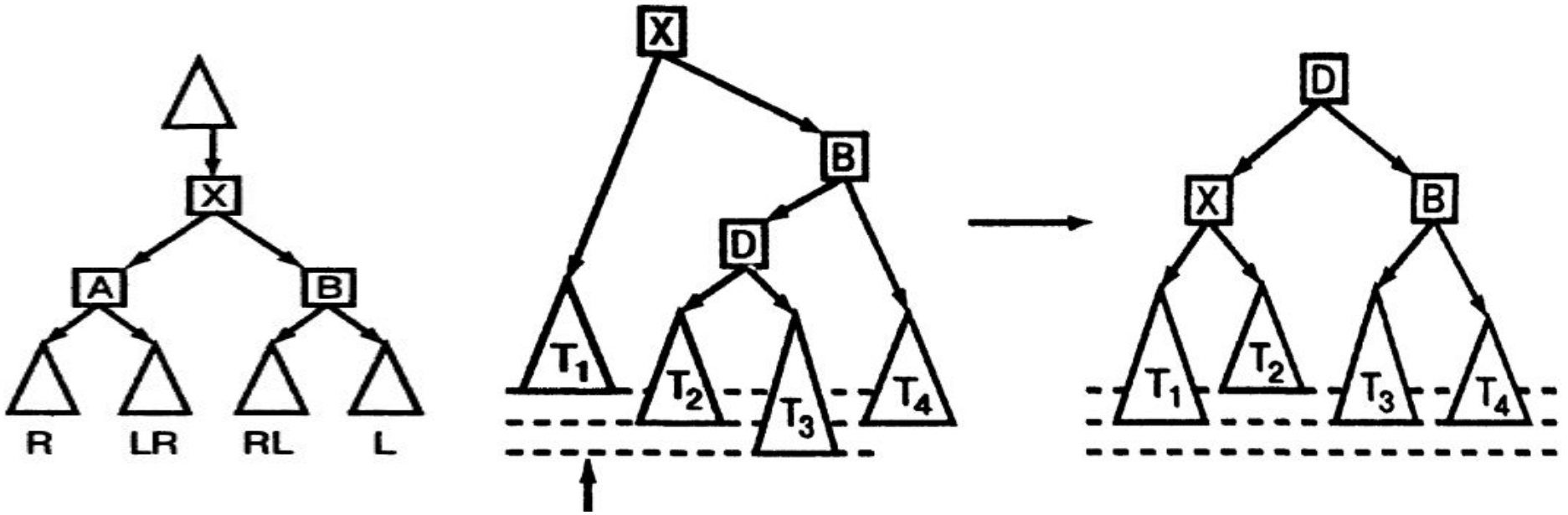
Удаление узла из АВЛ-дерева

Вращение вправо-влево R-L (большое левое вращение)

Узел удаляется **из левого поддерева T1 под узлом X**, но **левая половина правого поддерева длиннее правой половины**.

В этом случае для перебалансирования дерева необходимо использовать вращение **вправо-влево (R-L)**.

Если левое или правое поддеревья T2 длиннее T3 или наоборот, вращение вправо-влево перебалансирует поддерево Tx и сократит при этом глубину Tx на 1. Это означает, что **дерево выше узла X может быть разбалансировано**, поэтому **необходимо продолжить проверку выполнения свойства AVL для всех предков узла X**.



Удаление узла из АВЛ-дерева

Другие типы вращений

Допустим, **удаляемый узел находится в правом поддереве ниже узла X.** Другие типы вращения подобны описанным выше для добавления узла. **Опишите их и нарисуйте схему.**

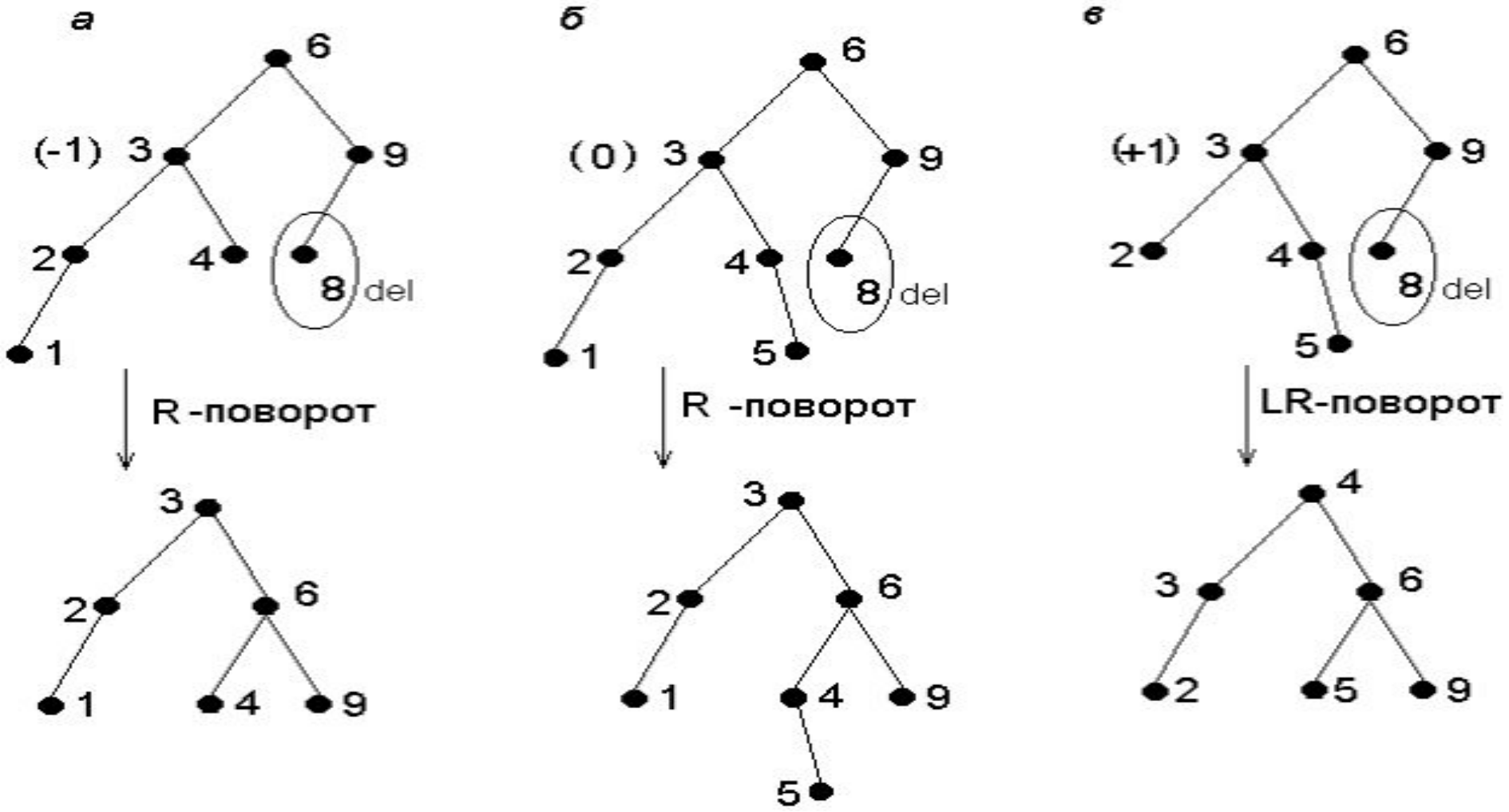
Все четыре типа вращения те же самые, что использовались для балансировки дерева при вставке узла, за одним исключением.

Когда добавляется новый узел к дереву, первое вращение перебалансирует поддерево Tx без изменения его высоты. Это означает, что дерево выше Tx должно оставаться сбалансированным.

Когда вращение используется после удаления узла, оно может уменьшить высоту поддерева Tx на 1. В этом случае нельзя быть уверенным, что дерево выше узла X все еще сбалансировано. Нужно продолжить проверку выполнения свойства AVL.

Удаление узлов из АВЛ-дерева

Пример. Пусть в результате удаления элемента недопустимо **уменьшилась высота правого поддерева** вершины X. Эта ситуация целиком **совпадает со случаем**, когда недопустимо **увеличилась высота левого поддерева** этой вершины. Какой вид балансировки применить по-прежнему зависит от показателя сбалансированности левого потомка вершины X. На рисунке показаны все три принципиально различные ситуации.

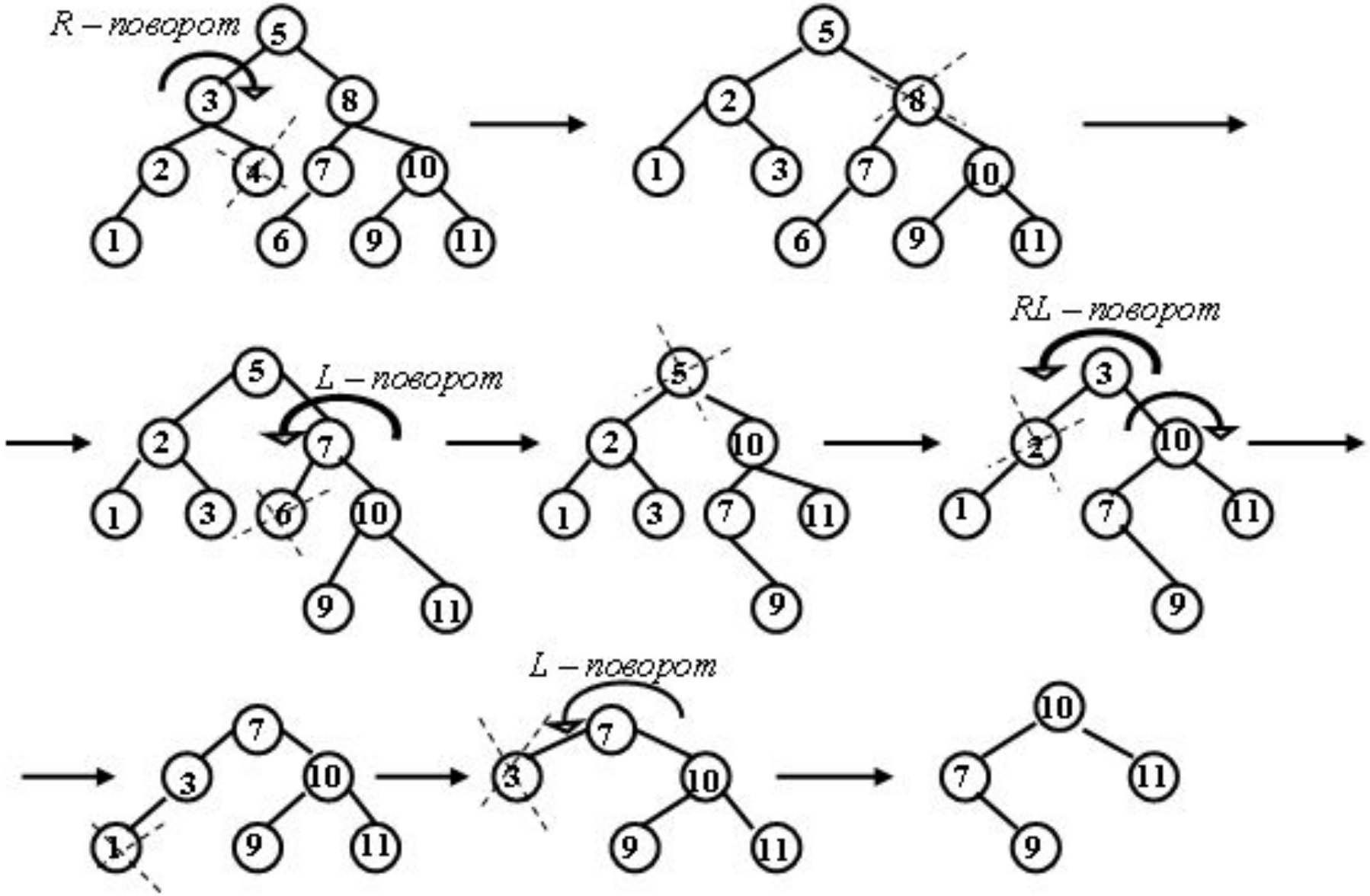


Удаление узлов из АВЛ-дерева

Пример 6. Постройте AVL-дерево из последовательности 5, 3, 8, 10, 7, 2, 4, 1, 6, 9, 11. Последовательно исключите из него узлы 4, 8, 6, 5, 2, 1, 3.

Удаление узлов из АВЛ-дерева

Пример 6. Удаление узлов из АВЛ-дерева. Последовательное исключение узлов 4, 8, 6, 5, 2, 1, 3.

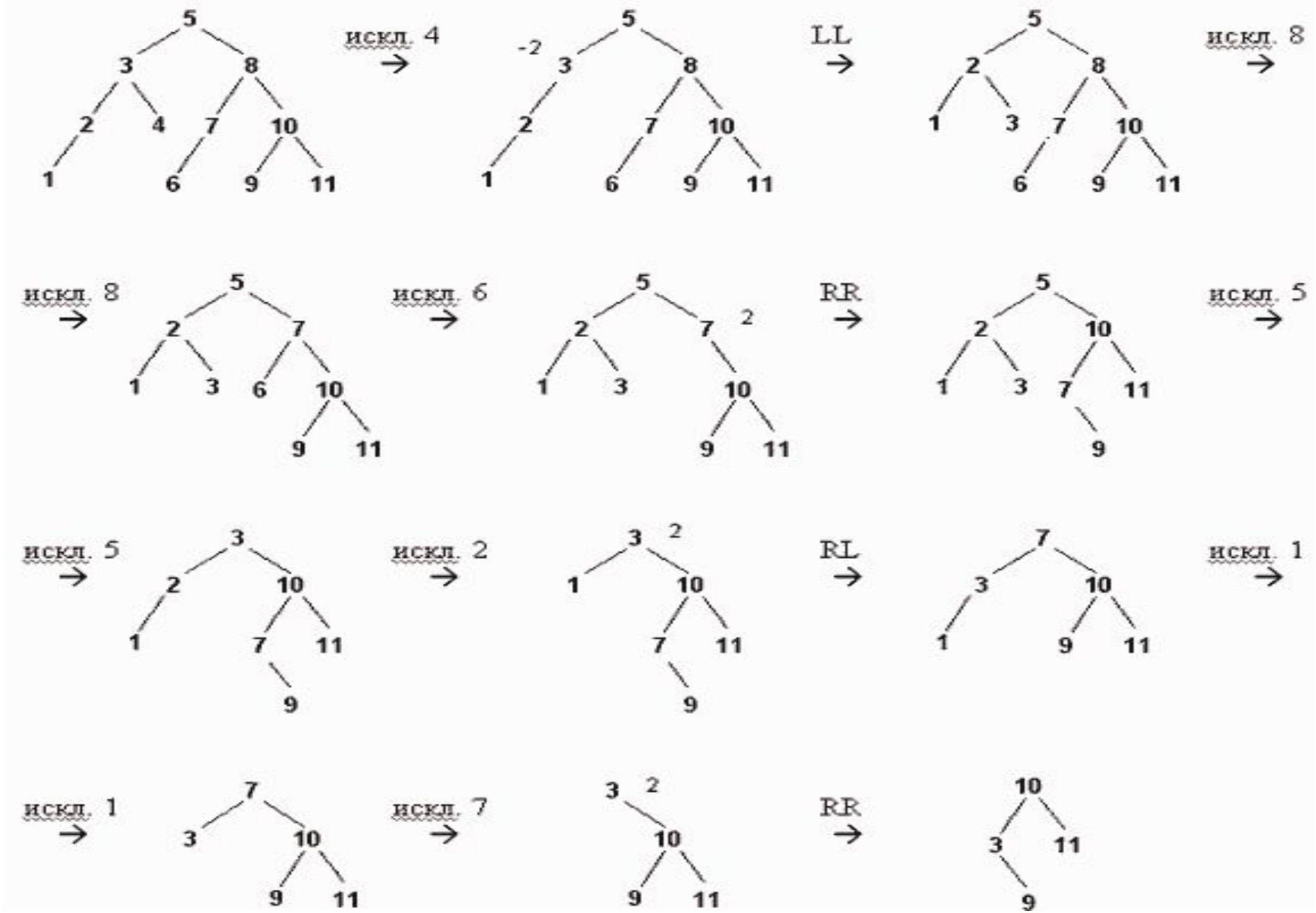


Удаление узлов из АВЛ-дерева

Пример 7. Постройте AVL-дерево из последовательности 5, 3, 8, 10, 7, 2, 4, 1, 6, 9, 11. Последовательно исключите узлы из него узлы 4, 8, 6, 5, 2, 1, 7.

Удаление узлов из АВЛ-дерева

Пример 7. Рассмотрим последовательное исключение узлов 4, 8, 6, 5, 2, 1, 7 из следующего дерева:



Удаление узлов из АВЛ-дерева

Пример 8. Постройте AVL-дерево из последовательности 12, 6, 22, 3, 9, 18, 28, 2, 4, 8, 15, 20, 24, 30, 1, 17, 21, 23, 26, 35, 27. Последовательно исключите узлы из него узлы 8, 18, 20.

Удаление узлов из АВЛ-дерева

Программное удаление узла из АВЛ-дерева

Процедура RemoveFromNode рекурсивно обращается к дереву и когда **находит искомый узел, удаляет его**.

Если у него нет дочерних узлов, то процедура заканчивается.

Если имеется один дочерний узел, то удаляемый узел заменяется его потомком.

Если узел имеет двух потомков, то процедура **RemoveFromNode** вызывает процедуру **ReplaceRightMost**, чтобы заменить удаляемый узел крайним **правым узлом в его левой ветви** – удаление как из обычного (несбалансированного) сортированного дерева.

Основное отличие возникает при возврате из процедуры и рекурсивном проходе вверх по дереву. При этом процедура **ReplaceRightMost** использует восходящую рекурсию, чтобы проверить баланс в каждом узле дерева.

Удаление узлов из АВЛ-дерева

Программное удаление узла из АВЛ-дерева

Когда вызов процедуры (ReplaceRightMost) завершается, вызывающая **ReplaceRightMost** процедура (RemoveFromNode) использует **RebalanceRightShrunk для проверки баланса во всех точках дерева.** Так как ReplaceRightMost исследует **правые ветви дерева**, она всегда использует процедуру **RebalanceRightShrunk**.

Процедура RemoveFromNode первый раз вызывает ReplaceRightMost, заставляя ее двигаться влево вниз от удаляемого узла. Когда первый вызов процедуры ReplaceRightMost завершается, RemoveFromNode вызывает процедуру **RebalanceLeftShrunk**, чтобы проверить баланс во всех точках дерева.

После этого рекурсивные обращения к RemoveFromNode завершаются и процедура выполняется обычным способом снизу вверх. Как и ReplaceRightMost, RemoveFromNode использует восходящую рекурсию для проверки баланса дерева.

После каждого вызова этой процедуры следует вызов процедуры RebalanceRightShrunk или RebalanceLefShrunk в зависимости от того, по какому пути происходит спуск по дереву.

Процедура RebalanceLeftShrunk аналогична RebalanceRightShrunk.

Удаление узлов из АВЛ-дерева

```
//Д.1 Удаление значения ниже выделенного узла.
procedure TAVLTree.RemoveFromNode (var node : TAVLNode;
    target_id : Integer; var shrunk : Boolean);
var   target : TAVLNode;
begin
// Если мы у основания дерева, то искомый узел находится не здесь.
    if node = nil then  begin
        shrunk := False;
        exit;
    end;
    if  target_id < node.id  then begin
// Поиск левого поддерева.
      RemoveFromNode(node.LeftChild, target_id, shrunk);
      if  shrunk  then RebalanceLeftShrunk(node, shrunk) ;// если узел удален
   end                                          //дерево сократилось
```

Удаление узлов из АВЛ-дерева

```
//Д.2
else if target_id>node.Id then
 begin
 // Поиск правого поддерева.
    RemoveFromNode(node.RightChild, target_id, shrunk);
    if (shrunk) then RebalanceRightShrunk(node, shrunk);
 end else begin // target_id=node.Id
 // Это искомый узел.
    target : = node ;
    if (node.RightChild = nil) then
    begin
 // Узел или не имеет дочерних, или имеет только левый.
       node := node.LeftChild;
       shrunk := True;
    end
```

Удаление узлов из АВЛ-дерева

```
//Д.3
else if  node.Lef tChild=nil then  begin
 // Узел имеет только правый дочерний.
         node := node.RightChild;
         shrunk := True;
       end else begin
 // Узел имеет два дочерних.
              ReplaceRightmost(node, node.LeftChild, shrunk);
              if  shrunk  then RebalanceLeftShrunk(node, shrunk);
            end; // Завершение удаления искомого узла.
 // Удаление искомого узла.
 target.LeftChild := nil;
 target.Rightchild := nil;
 target.Free;
 end;
 end; //Конец procedure
```

Удаление узлов из АВЛ-дерева

```
//Д.4 Замена искомого узла крайним правым потомком слева.
procedure TAVLTree.ReplaceRightmost (var target, repl : TAVLNode;
      var shrunk : Boolean);
var  old_repl : TAVLNode;
begin
if (repl.RightChild = nil) then  begin
// repl - это узел, которым заменят искомый.
// Запоминание положения узла.
   old_repl := repl;
// Замена repl его левым дочерним узлом.
   repl := repl.LeftChild;
// Заменить искомый узел переменной old_repl.
   old_repl.LeftChild := target.LeftChild;
   old_repl.RightChild := target.RightChild;
   old_repl.Balance := target.Balance;
   target := old_repl;
   shrunk := True;
end
```

Удаление узлов из АВЛ-дерева

```
//C.5
else begin
 // Рассмотрение правых ветвей.
    ReplaceRightmost(target,repl.RightChild,shrunk) ;
    if (shrunk) then RebalanceRightShrunk(repl,shrunk);
 end;
 end;
```

Удаление узлов из АВЛ-дерева

```
//C.6  Выполнение вращения вправо или влево-вправо после сокращения
 // правой ветви.
procedure TAVLTree.RebalanceRightShrunk (var node : TAVLNode;
var shrunk : Boolean);
var   child, grandchild :T AVLNode;
        child_bal,  grandchild_bal : TBalance;
begin
   if (node.Balance = RightHeavy) then  begin
// Правое поддерево было длиннее. Теперь сбалансировано.
     node.Balance := Balanced;
  end else if  node.Balance=Balanced  then  begin
// Был баланс. Теперь левое поддерево длиннее.
            node.Balance := LeftHeavy;
            shrunk := False;
         end
```

Удаление узлов из АВЛ-дерева

```
//C.7
else begin
 // Левое поддерево было длиннее. Теперь разбалансировано.
    Child := node.LeftChild;
    child_bal := child.Balance;
    if  child_bal<>RightHeavy  then begin
 // Вращение вправо.
      node.LeftChild := child.RightChild;
      child.RightChild := node;
      if  child_bal = Balanced  then  begin
        node.Balance := LeftHeavy;
        child.Balance := RightHeavy;
        shrunk := False;
      end
```

Удаление узлов из АВЛ-дерева

```
//C.8
else begin
  node.Balance := Balanced;
  child.Balance := Balanced;
 end;
   node := child;
 end else begin
 // Вращение влево-вправо.
        grandchild := child. RightChild;
        grandchild_bal := grandchild.Balance;
        child.RightChild := grandchild.LeftChild;
        grandchild.LeftChild := child;
        node.LeftChild := grandchild.RightChild;
        grandchild.RightChild := node;
```

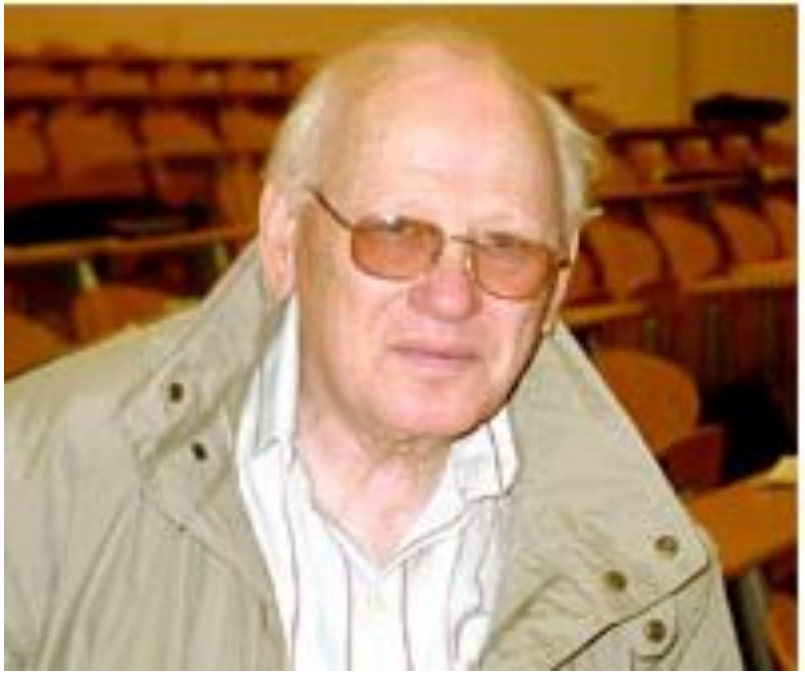
Удаление узлов из АВЛ-дерева

```
//C.9
if (grandchild_bal = LeftHeavy) then
   node.Balance := RightHeavy
else
   node.Balance := Balanced;
if (grandchild_bal = RightHeavy) then
   child.Balance := LeftHeavy
else
   child.Balance := Balanced;
 node := grandchild;
 grandchild.Balance := Balanced;
end; // Конец if ... else .. .
end; // Конец if balanced/left heavy/left unbalanced
end;
```

Сбалансированные деревья. АВЛ-деревья

Гео́ргий Макси́мович Адельсон-Вельский (родился 8 января 1922) — советский математик. Ученик А.Н. Колмогорова. Вместе с Е. М. Ландисом в 1962 изобрёл структуру данных, получившую название АВЛ-дерево.

С 1957 занимался проблемами искусственного интеллекта, в 1965 руководил разработкой компьютерной шахматной программы, которая победила американскую программу на первом шахматном матче между компьютерными программами; впоследствии на её основе была создана программа «Каисса», в 1974 ставшая первым компьютерным чемпионом мира по шахматам на чемпионате в Стокгольме.



Сбалансированные деревья. АВЛ-деревья

Ландис Евгений Михайлович (06 октября 1921 - 12 декабря 1997)

выдающийся советский математик, профессор, доктор физико-математических наук.

Хотя Е.М. Ландис наиболее известен своими работами в области дифференциальных уравнений, при этом он является автором известного алгоритма построения сбалансированного AVL-дерева. АВЛ-деревья названы по первым буквам фамилий их изобретателей, Г. М. Адельсона-Вельского и Е. М. Ландиса.