БАЛАНСИРОВКА ПРИ ВКЛЮЧЕНИИ В AVL ДЕРЕВО.

Методическая разработка. Матвеева Т.К.

Литература.

[1] **Н.Вирт, Алгоритмы + Структуры данных = Программы**, Москва , Мир, 1985г. шифр 5ВГ66 – В526

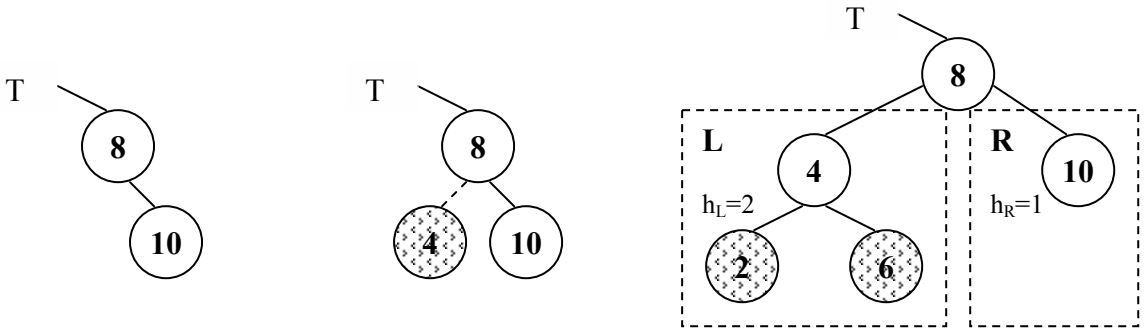
([1] стр.248) Обозначим дерево с высотой h через T_h.
Тогда T_0 – пустое дерево, T_1 – дерево с одним узлом и высотой h = 1.

Определение ***сбалансированного дерева***. (Критерий сбалансированности.)
Дерево является сбалансированным тогда и только тогда, когда *для каждого* узла высота его двух поддеревьев различается не более, чем на 1.

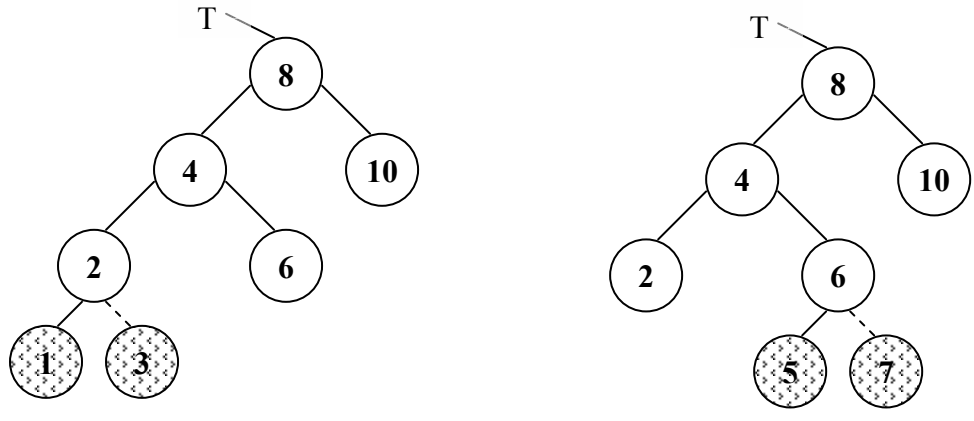
([1] стр.250) 4.4.7. Включение в сбалансированное дерево.
Посмотрим, что может произойти, когда в сбалансированное дерево включается новый узел. Пусть непустое дерево T имеет корень с L левым и R правым поддеревьями. Обозначим как h_L, h_R высоты левого и правого поддеревьев. ***Предположим, что новый узел включается в L – левое поддерево, вызывая увеличение его высоты на 1***.

Возможны три варианта (слева соотношение высот до включения **:** после).
Вариант 1. $h_L < h_R$ **:** L и R приобретают равную высоту, т.е. сбалансированность даже улучшается.
Вариант 2. $h_L = h_R$ **:** L и R становятся неравной высоты, но критерий сбалансированности не нарушается.
Вариант 3. $h_L > h_R$ **:** для L и R *критерий сбалансированности нарушается и дерево нужно перестраивать*.

Рассмотрим деревья на рис.1.



(T исходное) (a) (b)



(c) (d)

Рис.1.

Включение узла с ключом **4**, вызывает увеличение h_L на 1, при этом сбалансированность улучшается: $h_L = h_R$; вариант 1, рис 1(a).

Включение узла с ключом **2** (или включение узла с ключом **6,** или последовательное включение обоих) вызывает увеличение h_L на 1, при этом сбалансированность не нарушается: h_L и h_R различаются на 1; вариант 2, рис 1(b).

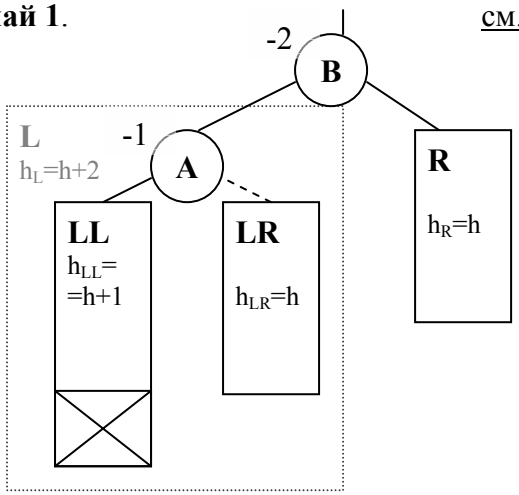
Включение узла с ключом **1** (или 3), вызывает увеличение h_L на 1, при этом сбалансированность нарушается: требуется перестройка дерева; вариант 3, рис 1(c).
Включение узла с ключом **5** (или **7**), также приводят к варианту 3, рис 1(d).

При внимательном изучении варианта 3 можно обнаружить,
что имеются лишь **две существенно различные** ситуации:
Случай 1 соответствует рис.1(c) и **Случай 2** соответствует рис.1(d).

Эти два случая, в общем виде, показаны ниже на рис.2, где поддеревья обозначены прямоугольниками, увеличение высоты поддерева при включении указано перечеркнутыми квадратами, слева от узла – значение показателя сбалансированности.

Определим **показатель сбалансированности узла**, например, так**:**
высота правого поддерева узла минус высота его левого поддерева.

Случай 1. см. рис.1 (c) включение ключей 1 или 3



Случай 2. см. рис1 (d) включение ключей 5 или 7

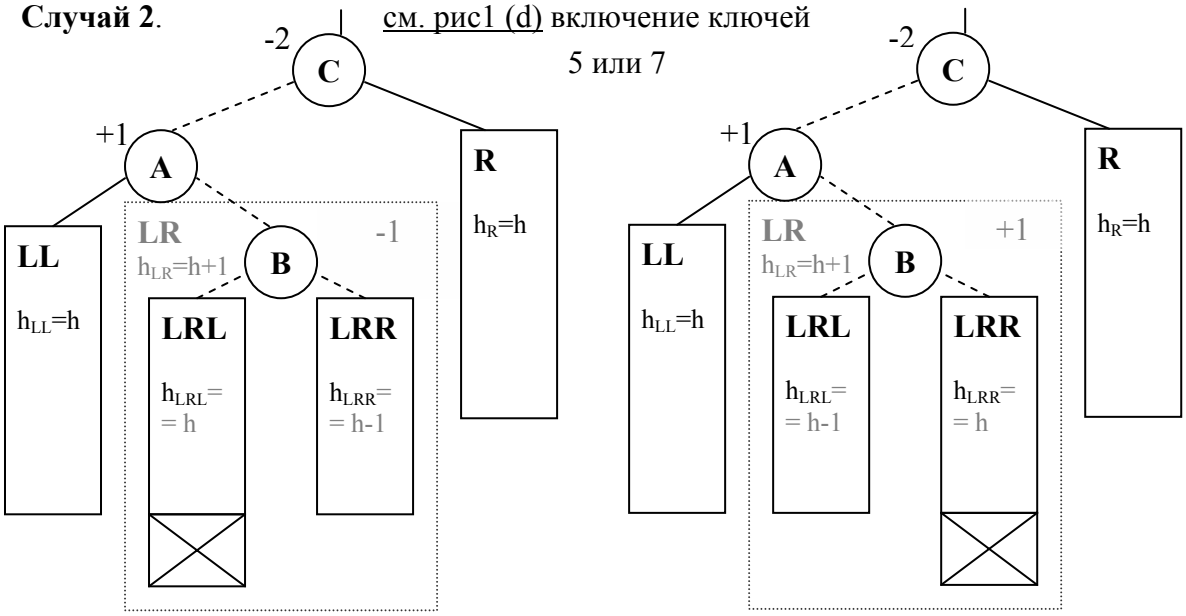


Рис.2.

Напомним, что пока ананиз ведется в ***предположении, что новый узел включается в L (левое поддерево), вызывая увеличение его высоты на 1***.

Заметим, что если на рис.1 исходное дерево будет с ключами **8** и **4**, то включение узла с ключом **2** дает вырожденный **случай 1**, а включение узла с ключом **6** дает вырожденный **случай 2.**

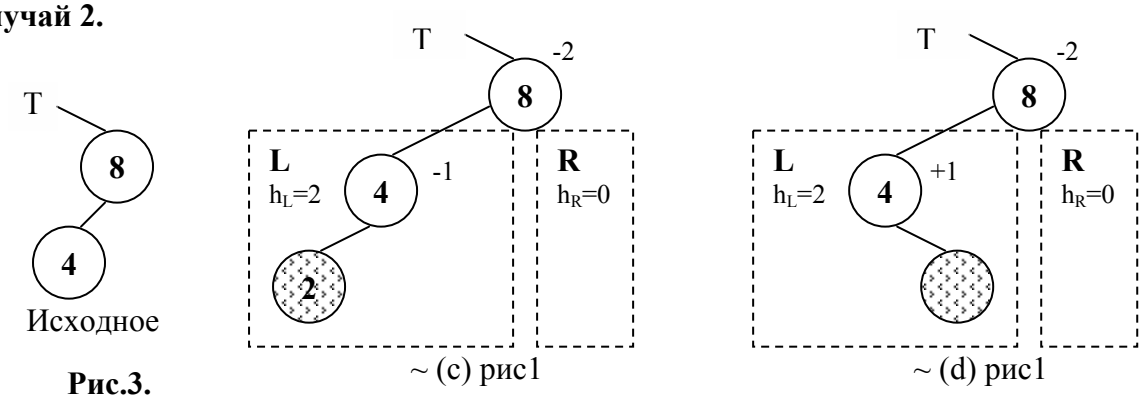


Рис.3.

Для дерева с ключами **8, 10, 4, 2**, включение узла с ключом **1** приводит к нарушению сбалансированности в узле с ключом **4** (не в корне), имеем **случай 1 для узла с ключом 4.**

Для дерева с ключами **8, 10, 4, 2**, включение узла с ключом **3** приводит к нарушению сбалансированности в узле с ключом **4,** имеем **случай 2 для узла с ключом 4.**

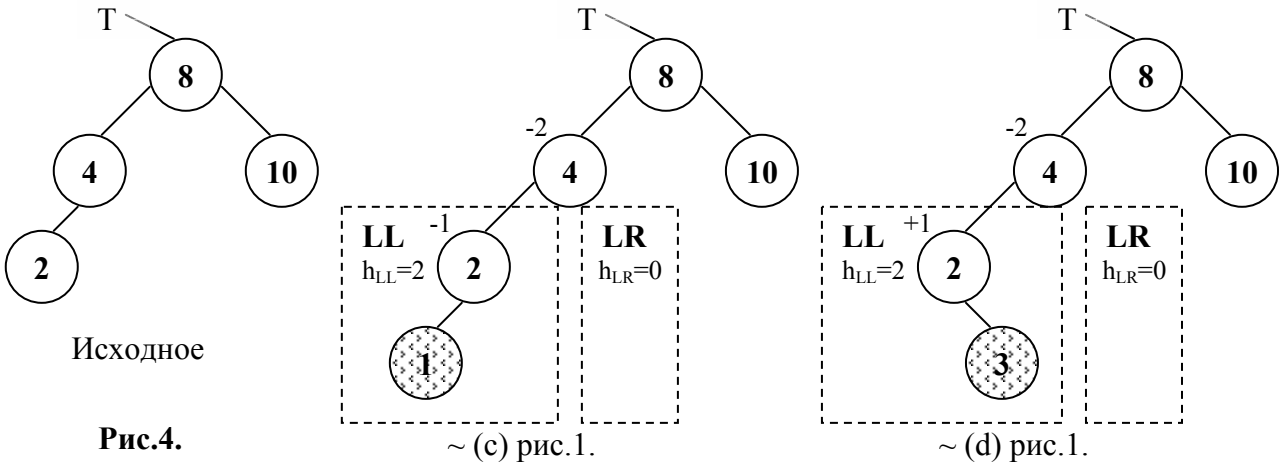


Рис.4.

С другой стороны, для дерева с ключами **8, 10, 4, 6**, включение узла с ключом **5** приводит к нарушению сбалансированности в узле с ключом **4** или включение узла с ключом **7** приводит к нарушению сбалансированности в узле с ключом **4. В последних двух случаях увеличилась высота R правого поддерева узла с ключом 4.** Имеем, соответственно, **случай 3** – симметричный случаю 1 и **случай 4** – симметричный случаю 2.

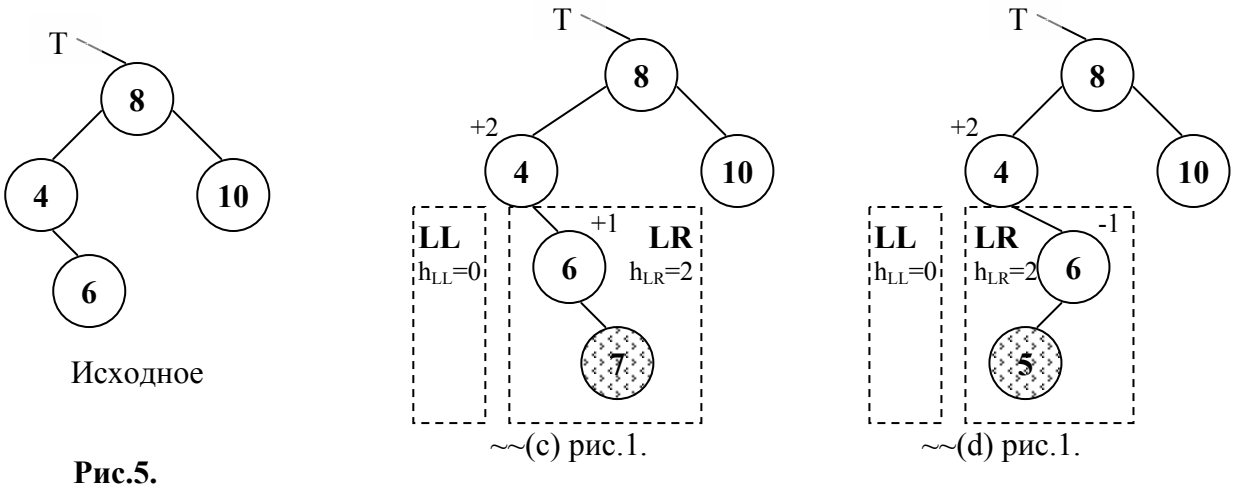
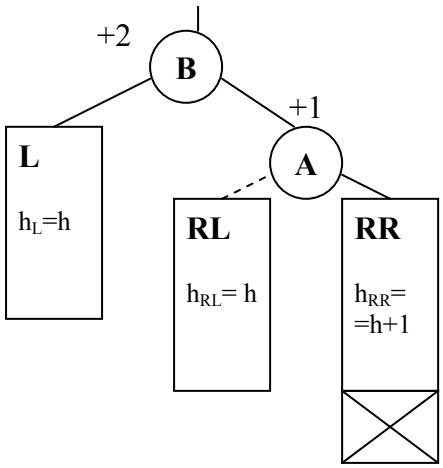


Рис.5.

Эти два симметричных случая возникают, ***если новый узел включается в R – правое поддерево с увеличением его высоты на 1, и это приводит к перестройке.***

Случай 3 (симметричный случаю 1).



Случай 4 (симметричный случаю 2).

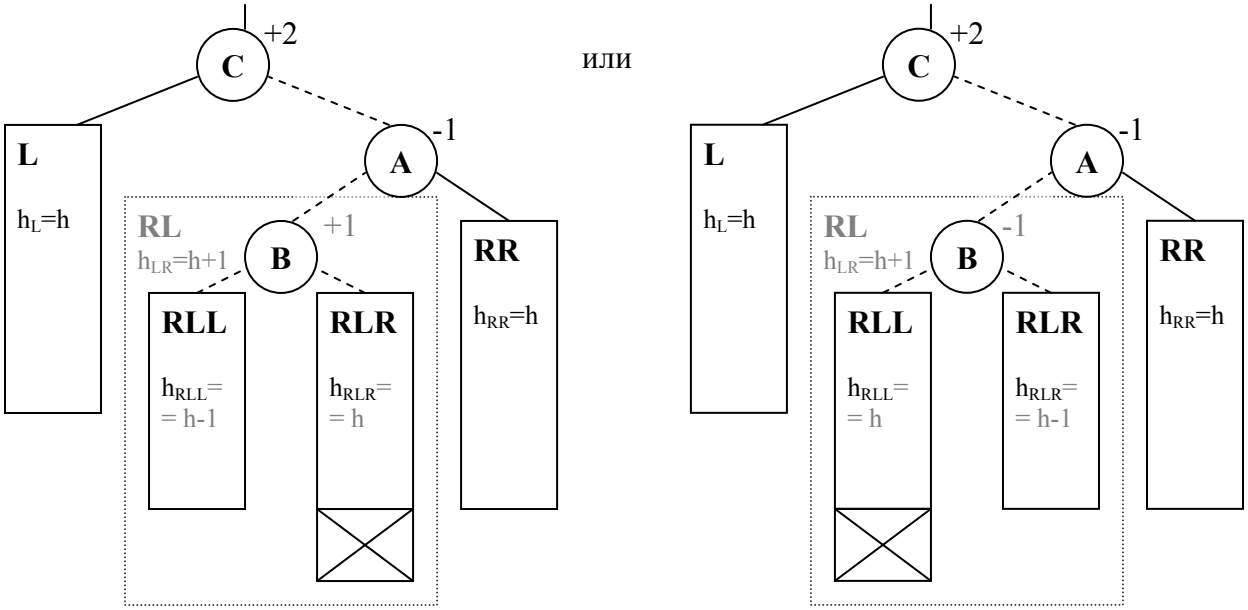
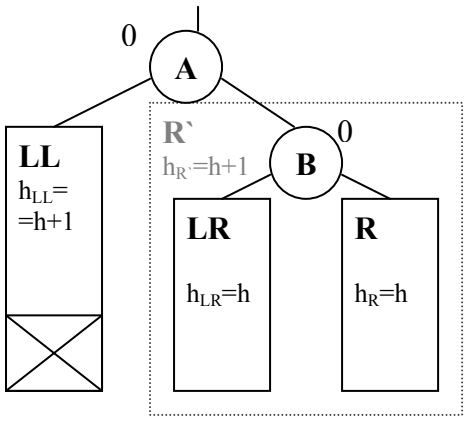


Рис.6.

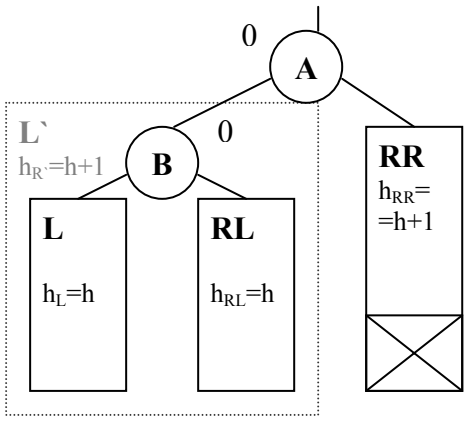
Простые преобразования в этих четырех случаях восстанавливают нужную сбалансированность. Результаты преобразований приведены на рисунках ниже. Отметим, что допускаются перемещения лишь в вертикальном направлении, в то время как относительное горизонтальное расположение обозначенных узлов и поддеревьев должно оставаться без изменений.

Результаты балансировки.

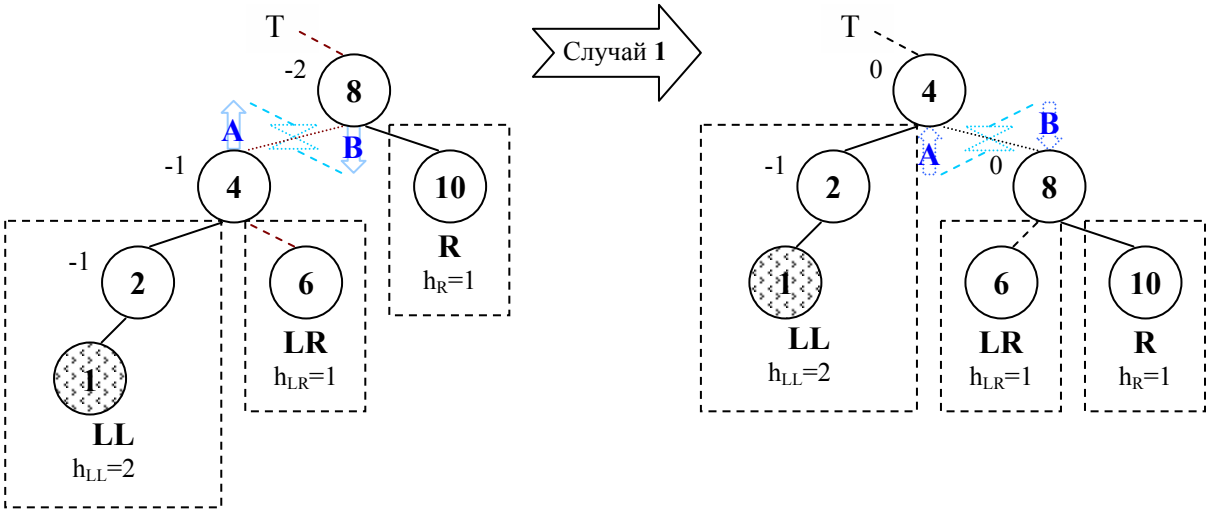
В случае **1 результат**



В случае **3 результат**

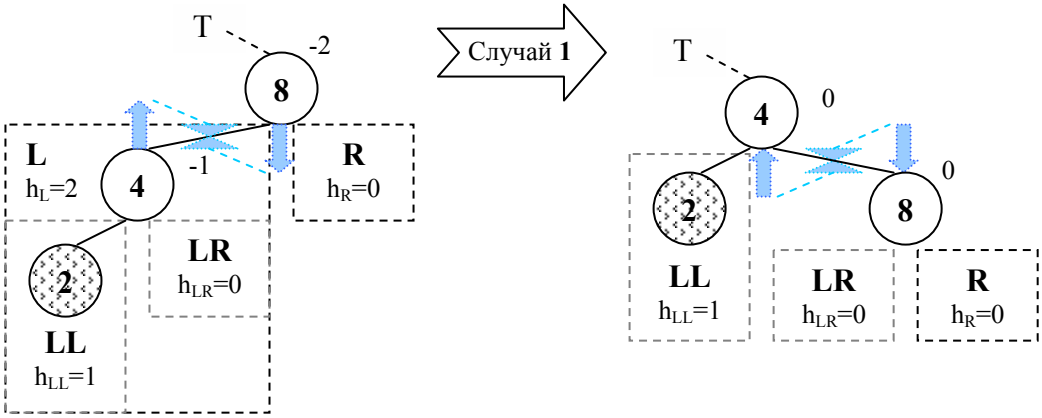


Примеры.



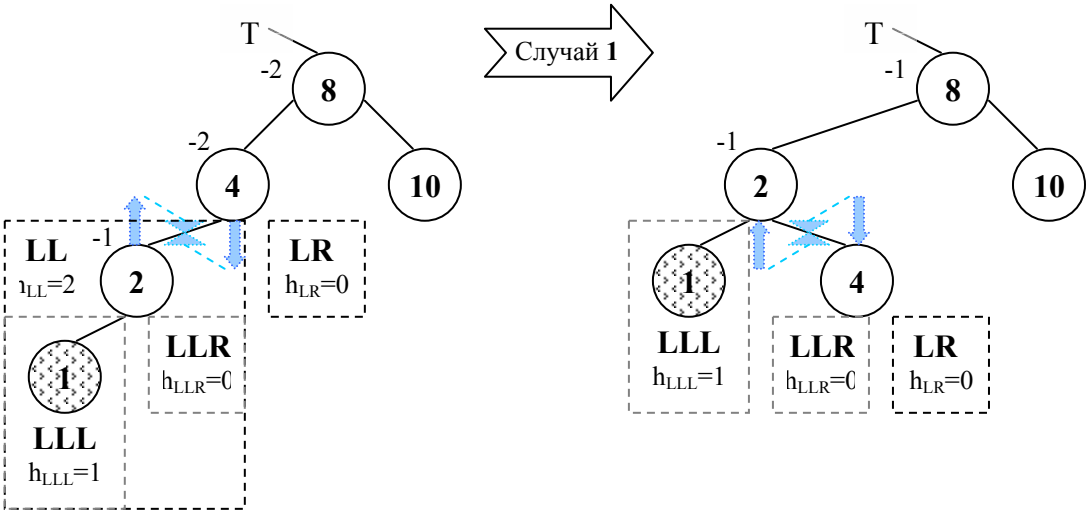
Исходное Т см рис.1. (с).

После балансировки



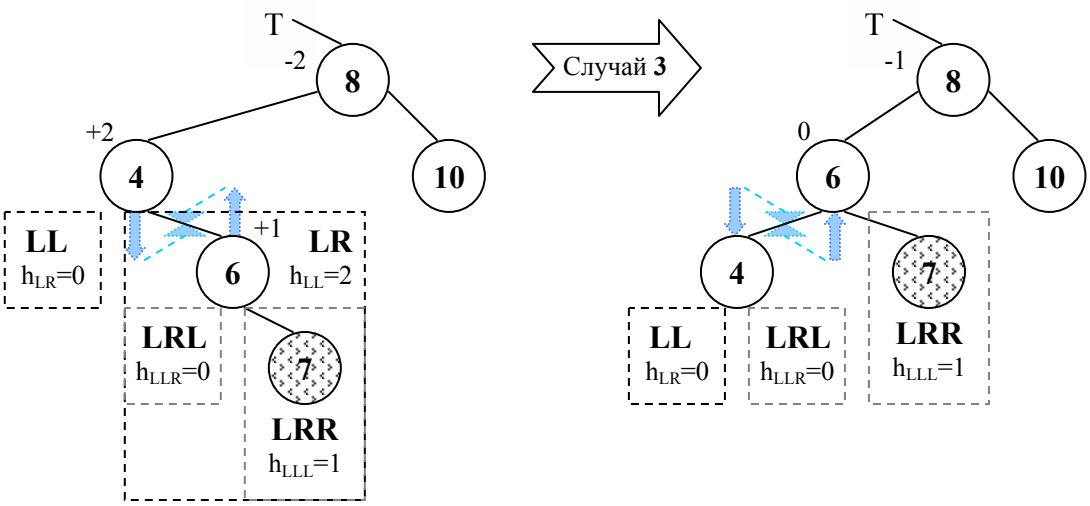
Исходное T см. рис.3. ~ (c)

После балансировки



Исходное T см. рис.4. ~ (c)

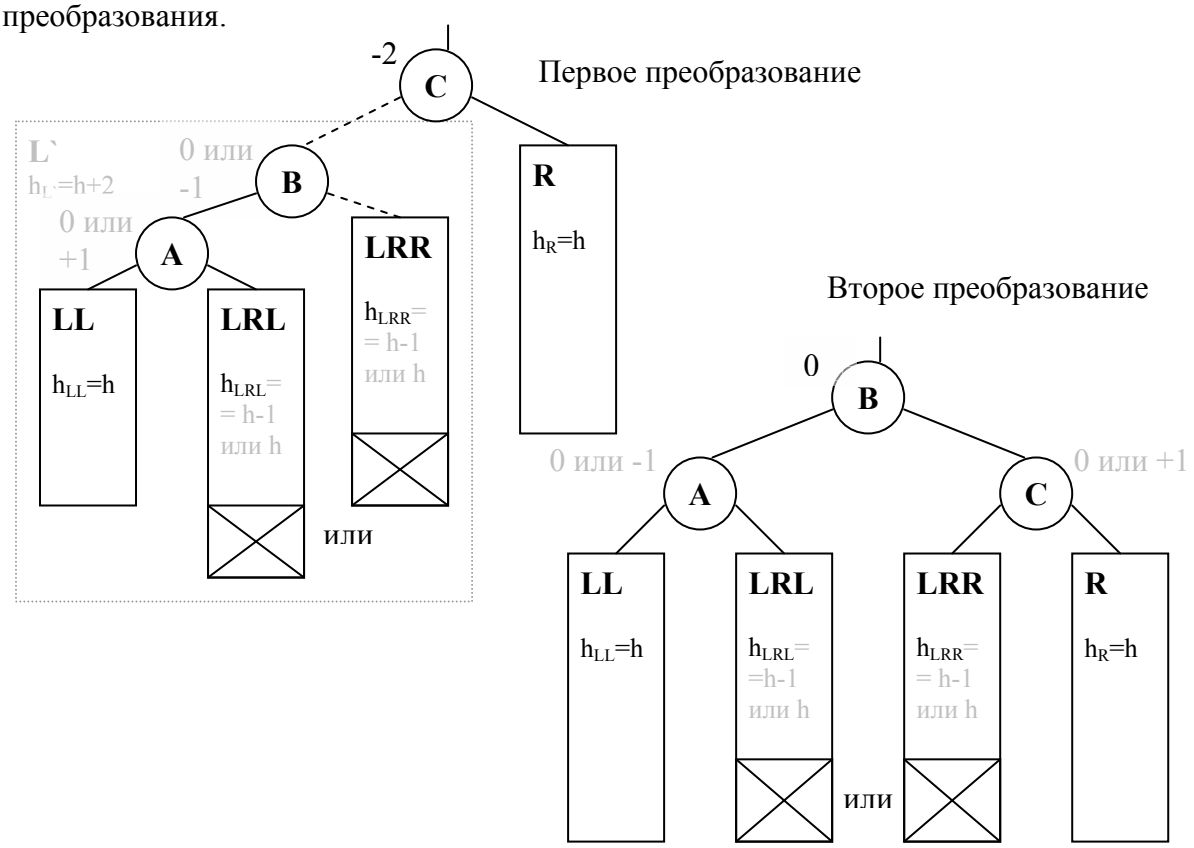
После балансировки



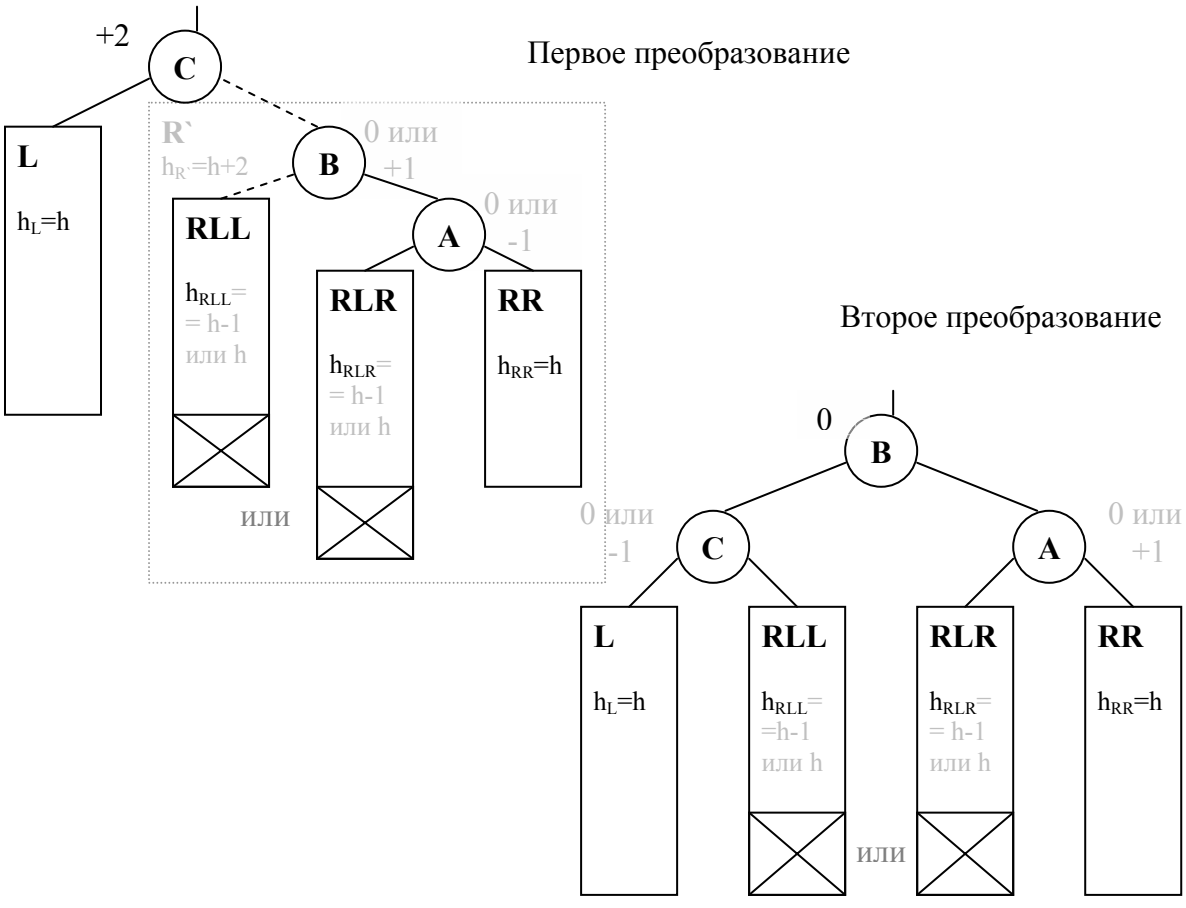
Исходное T см. рис.5. ~ (c)

После балансировки

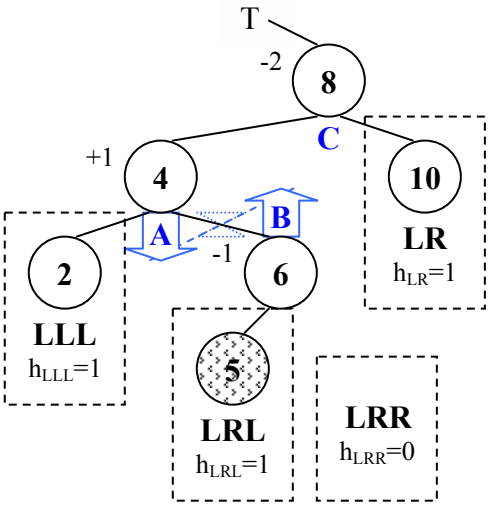
Случай 2. Независимо от высот LRL и LRR выполняются последовательно два преобразования.



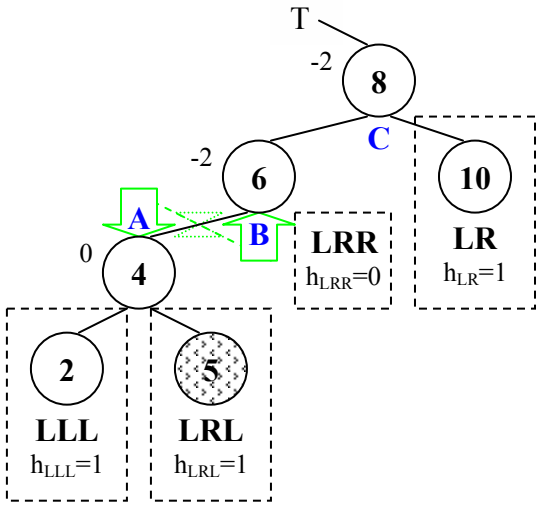
Случай 4. Аналогично, независимо от высот RLL и RLR последовательно выполняются следующие два преобразования



Примеры.

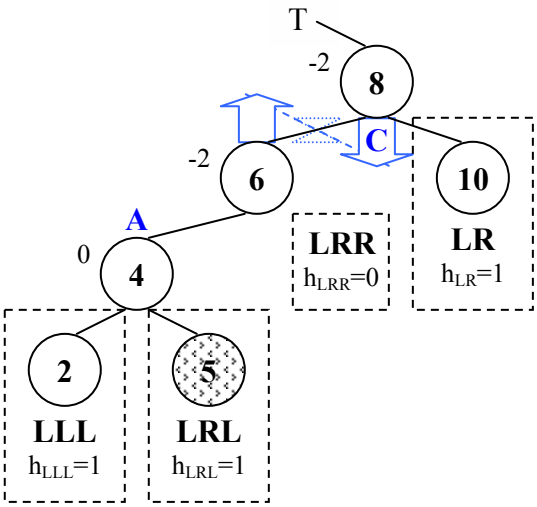


Исходное T см рис.1. (d).

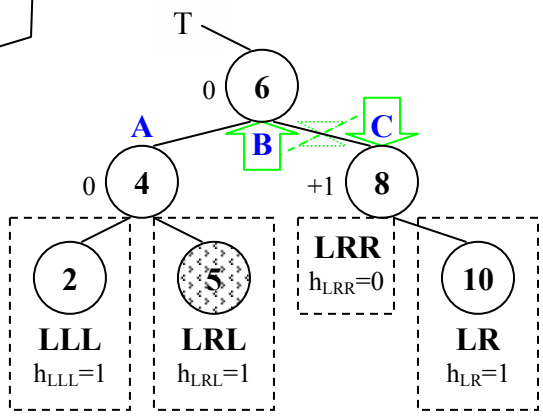


Первое преобразование

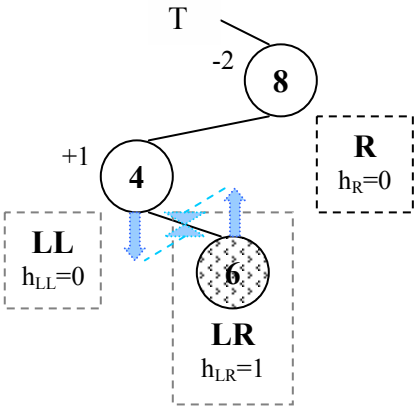
Случай **2**



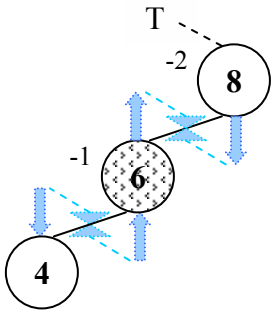
Исходное для следующего



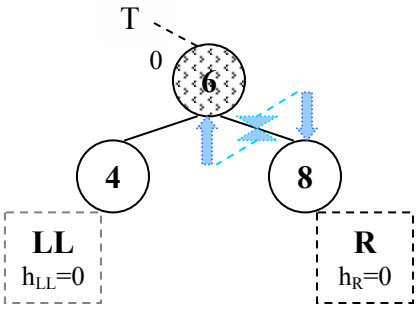
Второе преобразование



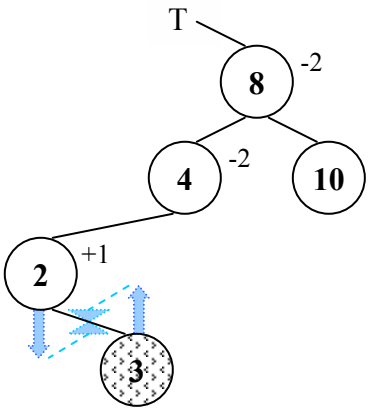
Исходное T см. рис.3. ~ (d)



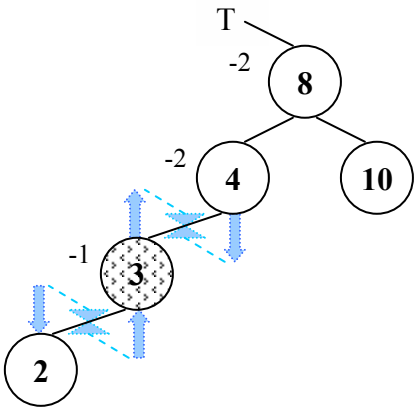
Первое преобразование



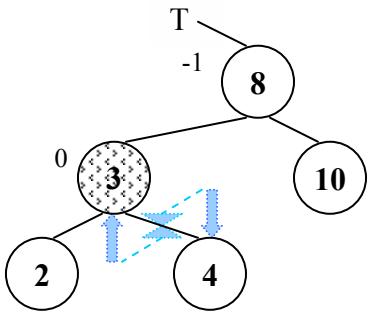
Второе преобразование



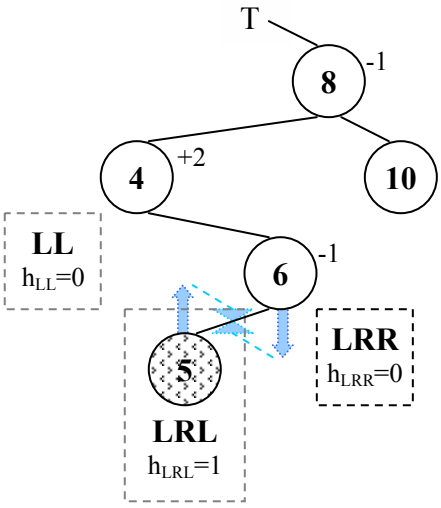
Исходное T см. рис.4. ~ (d)



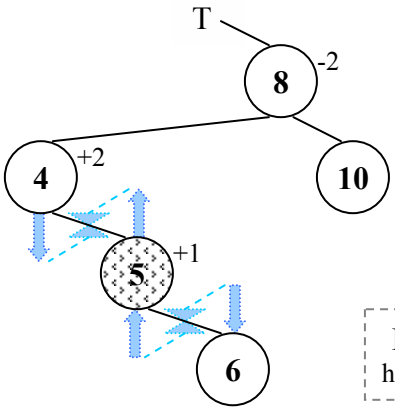
Первое преобразование



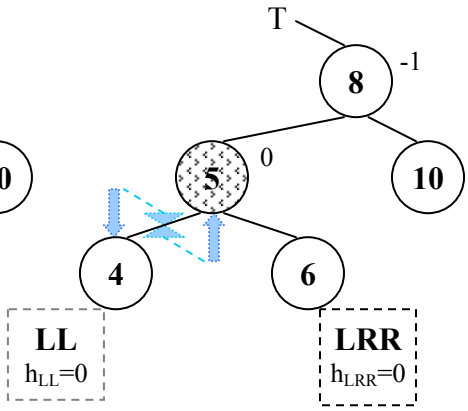
Второе преобразование



Исходное T см. рис.5. ~ (d)



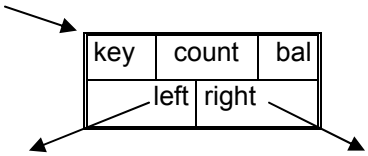
Первое преобразование



Второе преобразование

Алгоритм включения и балансировки полностью определяется способом хранения информации о сбалансированности дерева. Будем явно хранить показатель сбалансированности в информации, связанной с каждым узлом. Тогда определение типа для узла:

```
type node = record   key : integer ;
                     count : integer ;
                     left, right : ref ;
                     bal: –1 .. +1
            end
```



В дальнейшем мы будем интерпретировать *bal* – показатель сбалансированности узла, как высоту его правого поддерева минус высота его левого поддерева и будем строить алгоритм, исходя из узлов описанного выше типа.

В общих чертах процесс включения узла состоит из последовательности таких трех этапов:
1. Следовать по пути поиска, пока не окажется, что ключа нет в дереве.
2. Включить новый узел и определить новый показатель сбалансированности.
3. Пройти обратно по пути поиска и проверить показатель сбалансированности у каждого узла.

Хотя этот метод требует некоторой избыточной проверки (если сбалансированность установлена, то для предков соответствующего узла ее проверять уже не надо), мы будем вначале придерживаться этой, очевидно, корректной схемы, так как ее можно реализовать с помощью простого расширения процедуры поиска с включением (📖 [1] стр.236). Эта процедура описывает операцию поиска места включения нового узла с ключом *x* в дерево для каждого отдельного узла. Ссылка на текущий узел передаётся при помощи параметра-переменной *p*, что обеспечивает изменение значения *nil* на новое при включении нового узла в дерево. Благодаря рекурсивной формулировке ее можно дополнить операцией балансировки, выполняемой «по дороге назад вдоль пути поиска». На каждом шаге должна передаваться информация о том, увеличилась ли высота поддерева (в которое произведено включение). Поэтому мы добавим к списку параметров процедуры булевскую переменную *h*, означающую «*высота поддерева увеличилась*». Очевидно, что *h* должна быть параметром-переменной, поскольку используется для передачи результата.

Теперь предположим, что алгоритм возвращается к узлу с левой ветви (см. рис.2) с указанием, что ее высота. увеличилась. Мы должны различать три возможные ситуации в зависимости от высоты поддеревьев перед включением:

1. $h_L < h_R$, $p\uparrow.bal = +1$, предыдущая несбалансированность в p уравновешивается.
2. $h_L = h_R$, $p\uparrow.bal = 0$, вес склонился влево.
3. $h_L > h_R$, $p\uparrow.bal = -1$, необходима балансировка.

В третьем случае показатель сбалансированности корня левого поддерева определяет, который из случаев (1 или 2 на рис.2) имеет место. Если левое поддерево этого узла тоже выше правого, то мы имеем дело со случаем 1, иначе – со случаем 2.
(Убедитесь, что в этом случае левое поддерево корня не может иметь показатель сбалансированности, равный 0). Необходимые операции балансировки полностью заключаются в обмене значениями ссылок. Фактически ссылки обмениваются по кругу, что приводит к однократному или двукратному «повороту» двух или трех узлов. Кроме «вращения» ссылок следует также изменить соответствующие показатели сбалансированности узлов. Подробно это показано в процедуре поиска, включения и балансировки.

Принцип работы алгоритма показан на рис.7.

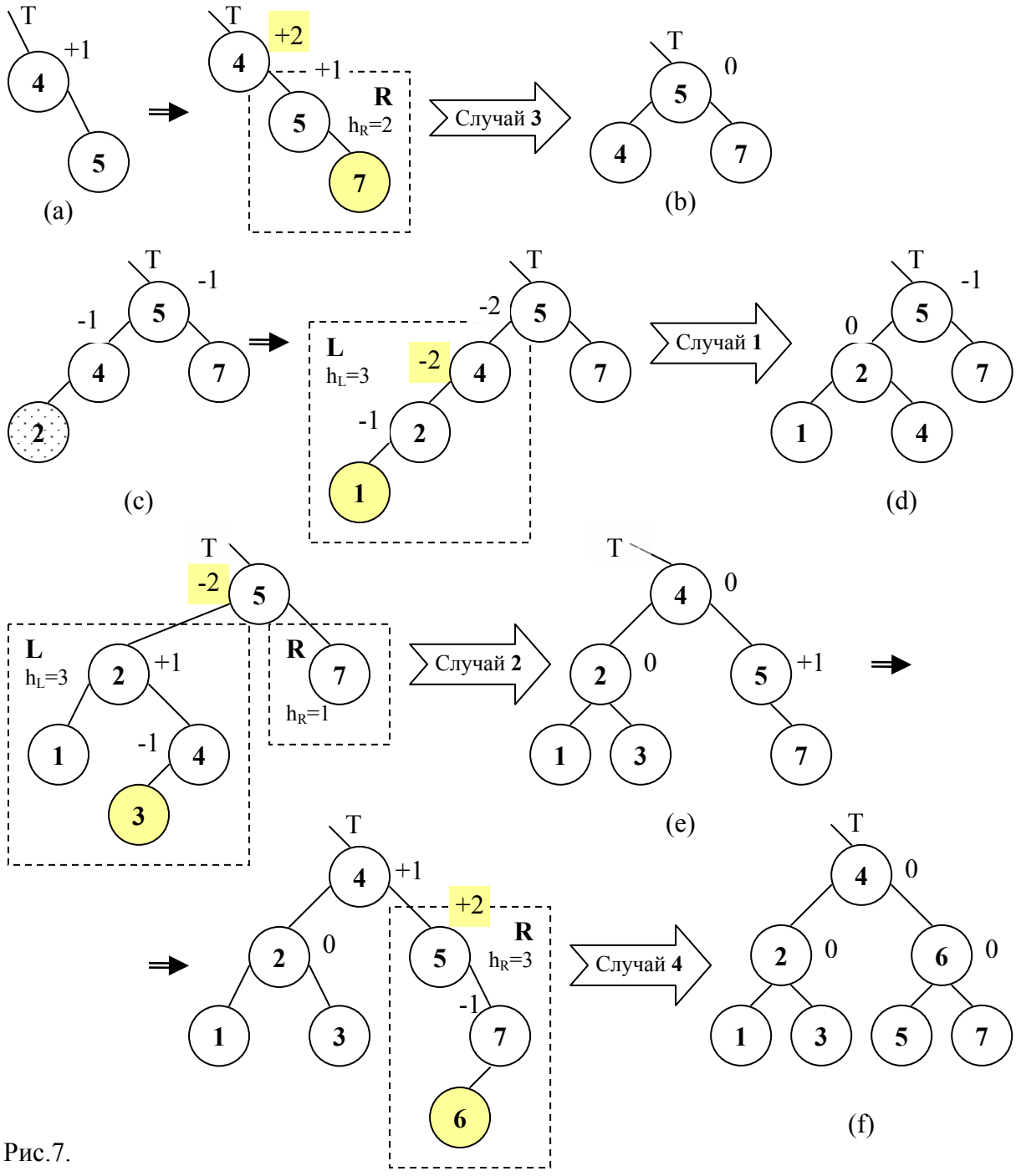
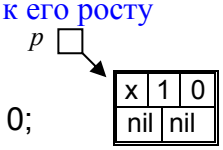


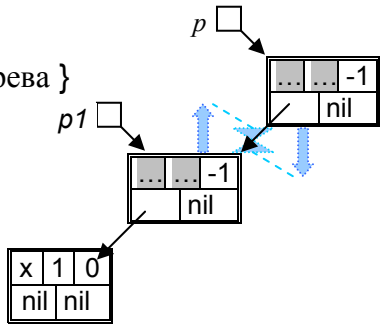
Рис.7.

Рассмотрим бинарное дерево (a), которое состоит только из двух узлов. Включение ключа 7 вначале дает несбалансированное дерево (т.е. линейный список). Его балансировка требует однократного правого (RR) поворота, давая в результате идеально сбалансированное дерево (b). Последующее включение узлов 2 и 1 дает несбалансированное поддерево с корнем 4. Это поддерево балансируется однократным левым (LL) поворотом (d). Далее включение ключа 3 сразу нарушает критерий сбалансированности в корневом узле 5. Сбалансированность теперь восстанавливается с помощью более сложного двукратного поворота налево и направо (LR); результатом является дерево (e). Теперь при следующем включении потерять сбалансированность может лишь узел 5. Действительно, включение узла 6 должно привести к четвертому виду балансировки: двукратному повороту направо и налево (RL). Окончательное дерево показано на рис.7 (f).

{ Процедура поиска, включения и балансировки. } (📖 [1] стр.254 – 255)

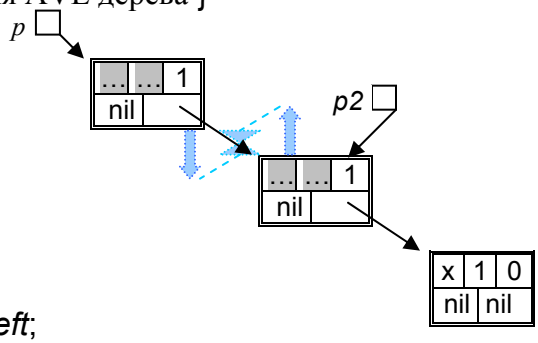
```
procedure search ( x: integer; var p: ref; var h: boolean );
var  p1, p2: ref;  { x - включаемый ключь, p - ссылка на текущий узел дерева ,                }
                   { на прямом ходу флаг h=false, т.к дерево пока не подвергось дополнению.}
begin
  if p=nil then { Найдено место для узла с ключём x }
   begin          { Включение нового узла в дерево     }
     new(p); h: = true;  {  приводит к его росту          }
     with p↑ do
     begin key := x; count := 1;
        left := nil; right := nil; bal := 0;
     end
   end else
   if x < p↑.key then
   begin search( x, p↑.left, h );
     if h then                { левая ветвь выросла              }
     case p↑.bal of
   1:  begin p↑.bal :=0; h := false
       end;                   { поддерево в целом не выросло }
   0:  p↑.bal := – 1;     { поддерево выросло допустимо для AVL дерева }
  –1:  begin              { требуется балансировка }
         p1 :=p↑.left;
         if p1↑.bal = –1 then
         begin            { однократный LL - поворот }
           p↑.left := p1↑.right; p1↑.right := p;
           p↑.bal := 0; p := p1
         end else
         begin            { двукратный LR – поворот }
           p2 := p1↑.right;
           p1↑.right := p2↑.left; p2↑.left := p1;
           p↑.left := p2↑.right; p2↑.right :=p;
           if p2↑.bal = –1 then p↑.bal := +1 else p↑.bal :=0;
           if p2↑.bal = +1 then p1↑.bal := –1 else p1↑.bal :=0;
           p :=p2
         end;
          p↑.bal := 0; h := false
      end
     end
   end else
```





```
  if x > p↑.key then
  begin search( x, p↑.right, h );
    if h then              { правая ветвь выросла          }
    case p↑.bal of
 –1:  begin p↑.bal := 0; h := false
      end;                 { поддерево в целом не выросло }
  0:  p↑.bal := +1;        { поддерево выросло допустимо для AVL дерева }
  1:  begin                { требуется балансировка }
       p1 := p↑.right;
       if p1↑.bal = +1 then
       begin { однократный RR – поворот }
         p↑.right := p1↑.left; p1↑.left := p
         p↑.bal := 0; p := p1
       end else

       begin { двукратный RL - поворот } p2 := p1↑.left;
         p1↑.left := p2↑.right; p2↑.right := p1;
         p↑.right := p2↑.left ; p2↑left := p;
         if p2↑.bal = +1 then p↑.bal := –1 else p↑.bal := 0;
         if p2↑.bal = –1 then p1↑.bal := +1 else p1↑.bal := 0;
         p := p2
       end ;
       p↑.bal := 0; h := false
      end
    end
    else
    begin p↑.count := p↑.count + 1; h := false
    end
end { search } ;
```



Эмпирические проверки показывают, что ожидаемая высота сбалансированного дерева, равна h = log(n)+c, где c – малая константа (c ≈ 0.25). Это значит, что на практике АВЛ - деревья ведут себя так же, как идеально сбалансированные деревья, хотя с ними намного легче работать. Эмпирически можно также предположить, что в среднем балансировка необходима приблизительно один раз на каждые два включения. При этом однократный и двукратный повороты одинаково вероятны. Пример на рис.4 явно был тщательно подобран, чтобы показать как можно больше поворотов при минимальном числе включений.

Из-за сложности операций балансировки считается, что сбалансированные деревья следует использовать лишь в том случае, когда поиск информации происходит значительно чаще, чем включение. «Скорость» изменения показателей сбалансированности, занимающих только два разряда каждый, и обращения к ним часто является решающим фактором, определяющим эффективность операции перебалансировки. Эмпирические оценки говорят, что сбалансированные деревья теряют большую часть своей привлекательности, если нужна плотная упаковка записи.