Соловьев С.Ю.

soloviev@glossary.ru

Алгоритмы и Алгоритмические языки

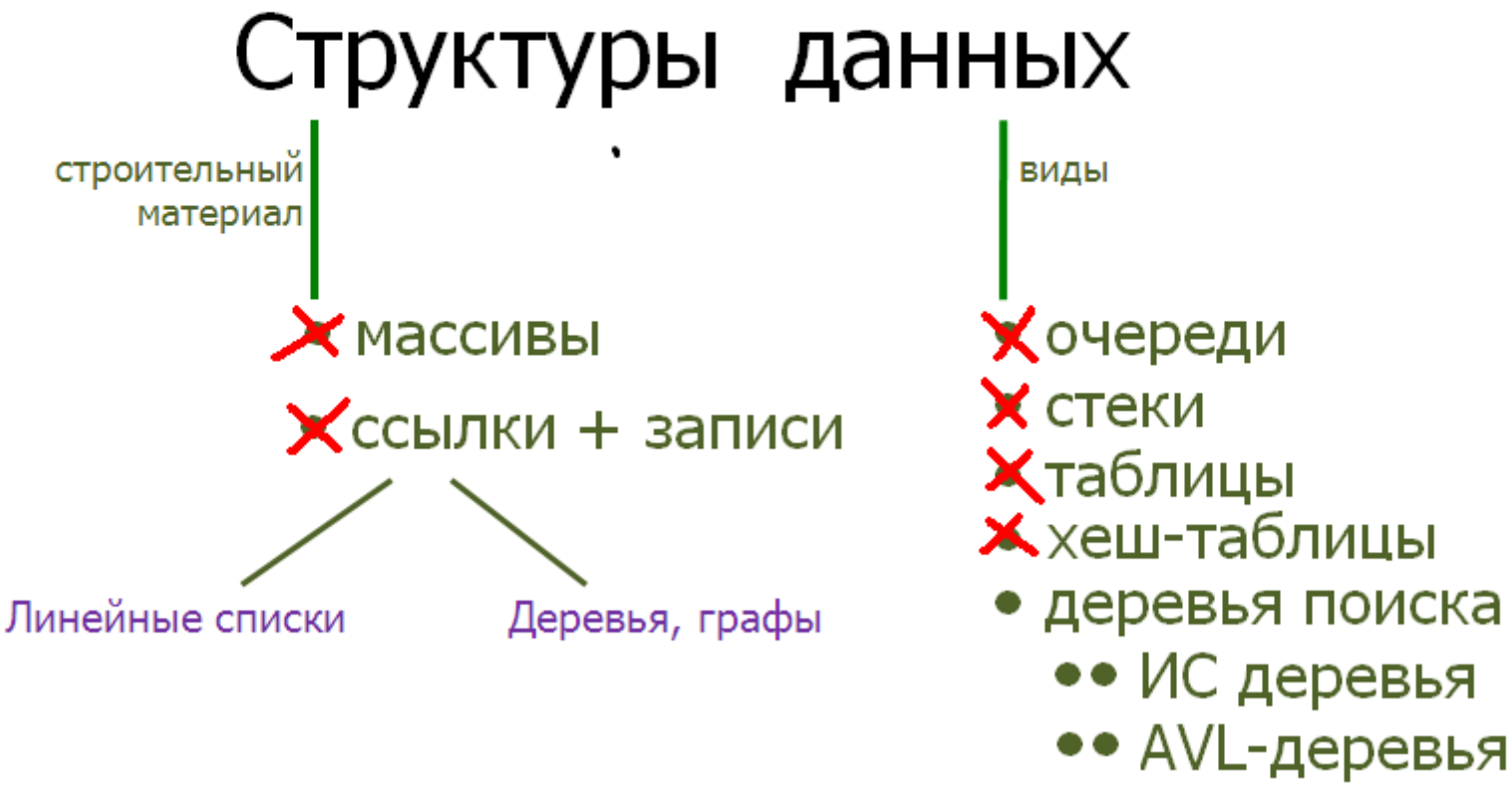
www.park.glossary.ru/pascal/

Лекция No. 15

2022

Напоминание

Структура данных – способ организации (однотипных) элементов данных, удобный для решения конкретной задачи.



FindSTr ✦ FormSTr ✦ KillSTr

Напоминание

```
function FindSTr(T : pDoTree; K : integer) : pDoTree;
begin    if     T = nil
         then FindSTr:=nil
         else   with T^ do
                      if Key = K then  FindSTr:=T
                else  if Key < K then  FindSTr:=FindSTr(Right,K)
                else                   FindSTr:=FindSTr(Left  ,K)   end;
```

```
function FormSTr(var T : pDoTree; K : integer) : pDoTree;
begin    if  T = nil  then Born(T,K);
         with T^ do
                      if Key = K then  FormSTr:=T
                else  if Key < K then  FormSTr:=FormSTr(Right,K)
                else                   FormSTr:=FormSTr(Left  ,K)   end;
```

```
procedure KillSTr(var T : pDoTree; K : integer);
begin    if  T = nil  then Exit;
         with T^ do
                      if Key = K then  KPECT(T)
                else  if Key < K then  KillSTr(Right,K)
                else                   KillSTr(Left  ,K)   end;
```

Сложность ✦ Высота дерева

Напоминание


```
with T^ do
      if Key = K then        обр
else  if Key < K then        рек Right
else                         рек Left
```

Количество сравнений ⩽ *Высота дерева* — есть — ***max*** количество вершин на пути от корня к листу

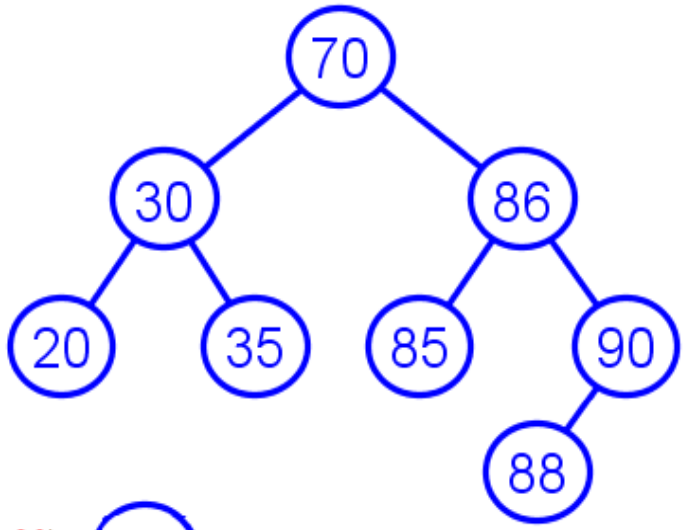
К-во операций **FindSTr, FillSTr, KillSTr** ⩽ **C·H**, где H – высота дерева T.

Если **H ⩽ Φ(n)**, где n – к-во вершин в T, то **сложность** есть **O(Φ(n))**.

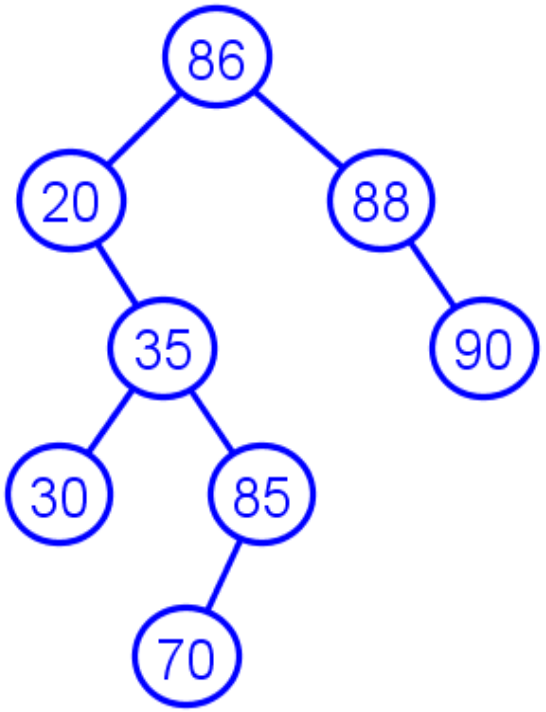
Примеры

Напоминание

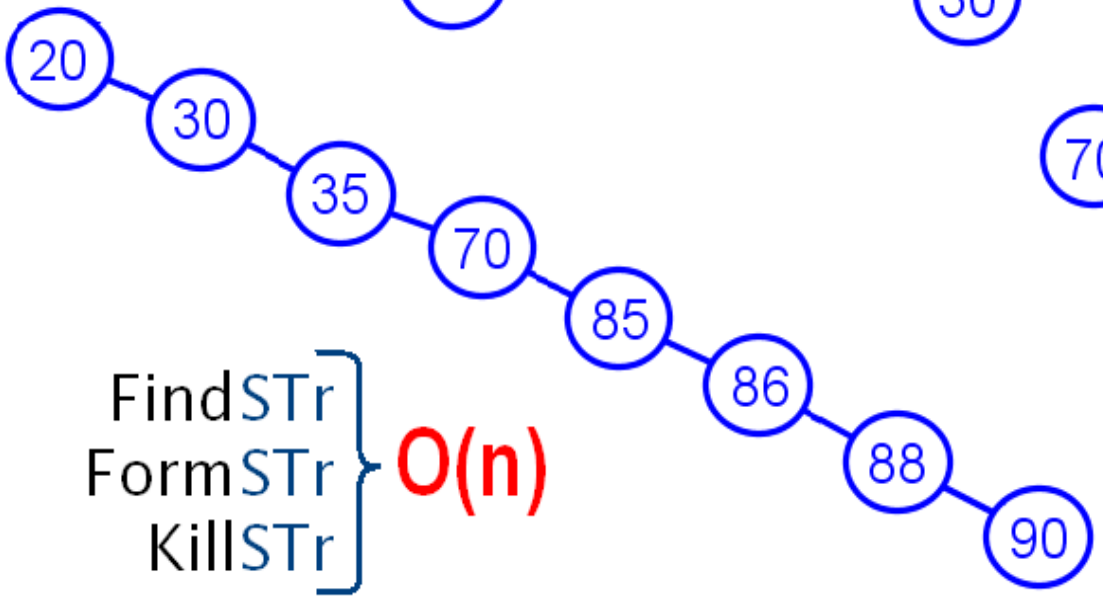
FormSTr(T,70);
FormSTr(T,86);
FormSTr(T,30);
FormSTr(T,35);
FormSTr(T,20);
FormSTr(T,90);
FormSTr(T,85);
FormSTr(T,88);



FormSTr(T,86);
FormSTr(T,88);
FormSTr(T,20);
FormSTr(T,35);
FormSTr(T,85);
FormSTr(T,30);
FormSTr(T,70);
FormSTr(T,90);

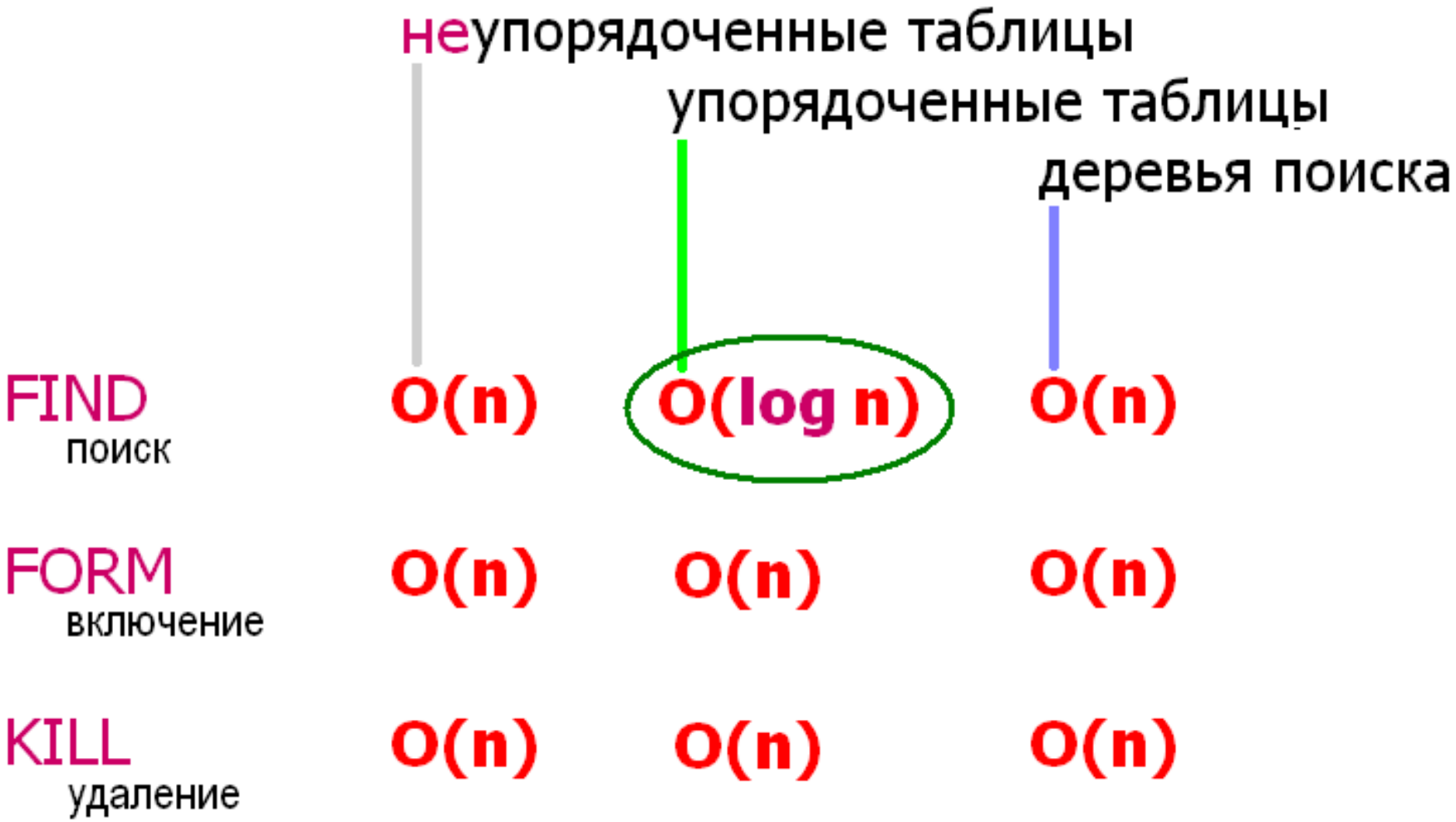


FormSTr(T,20);
FormSTr(T,30);
FormSTr(T,35);
FormSTr(T,70);
FormSTr(T,85);
FormSTr(T,86);
FormSTr(T,88);
FormSTr(T,90);



FindSTr
FormSTr
KillSTr
} O(n)

Оценки сложности

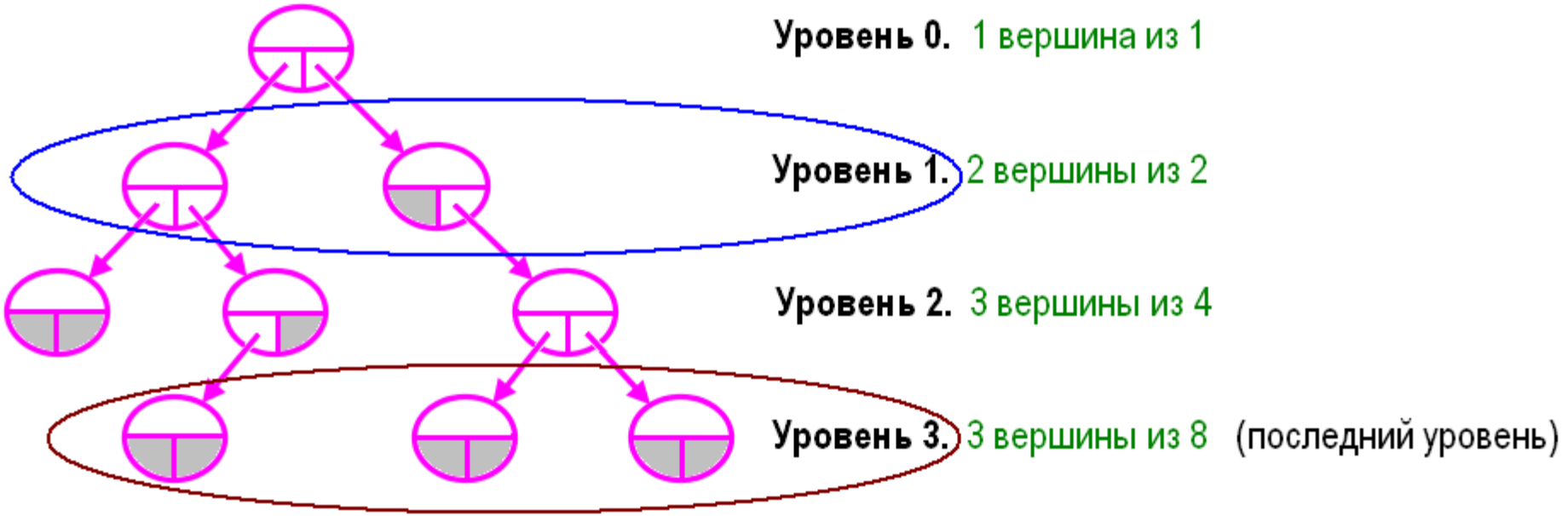


Идеально/полностью сбалансированное дерево

Идеально сбалансированное дерево – дерево поиска, все уровни которого (возможно, за исключением последнего уровня) полностью заполнены.

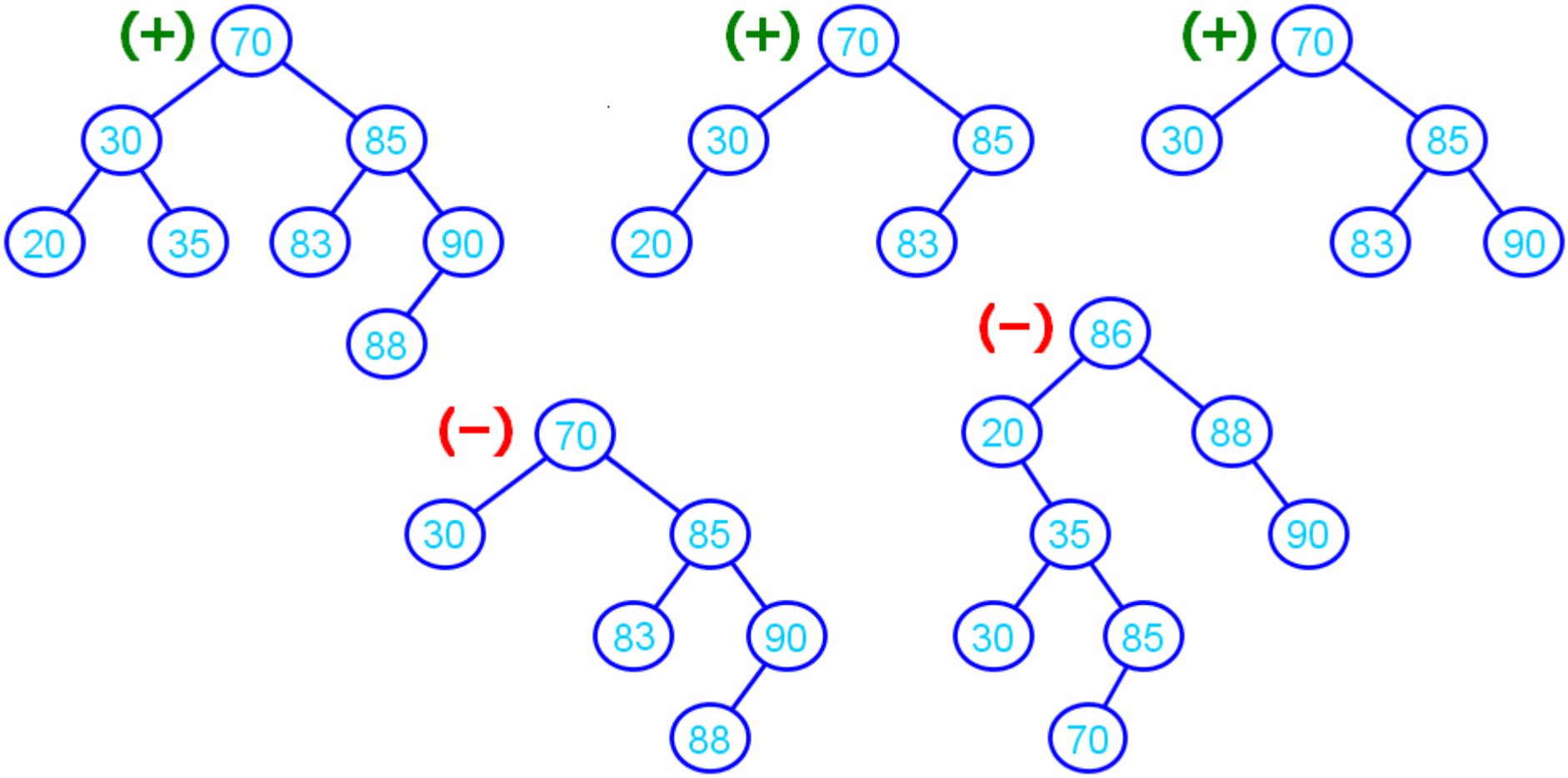
В бинарном дереве с корнем

- вершины уровня 0 = { корень дерева };
- вершины уровня n+1 – непосредственные потомки вершин уровня n.



Примеры

Идеально сбалансированное дерево – дерево поиска, все уровни которого (возможно, за исключением последнего уровня) полностью заполнены.



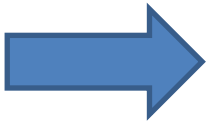
Примеры

Идеально сбалансированное дерево – дерево поиска, все уровни которого (возможно, за исключением последнего уровня) полностью заполнены.

Пусть **n** – количество вершин в идеально сбалансированном дереве. Тогда:

$[\log_2 n]$ – последний непустой уровень дерева,

$[\log_2 n]+1$ – мах сравнений при поиске ключа.

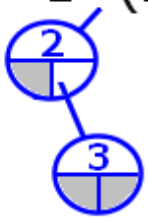


В ИС дереве

Поиск	**O(log n)**
Включение	**O(n)**
Удаление	**O(n)**

Включить в ИС дерево новую вершину

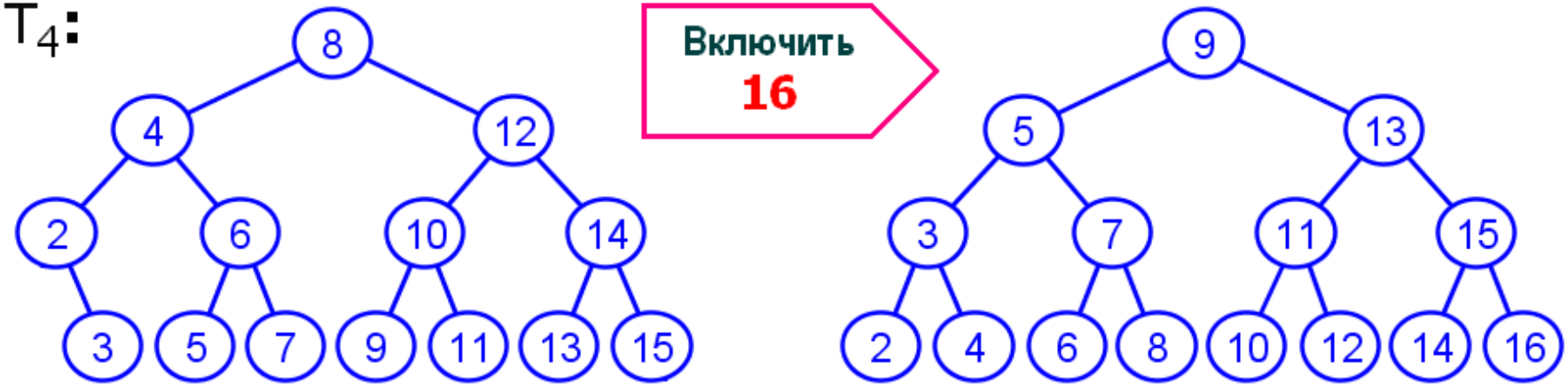
Пусть T_k – ИС дерево, в котором представлены вершины с ключами 2, 3, . . . , 2^k-1 (n = 2^k-2), и вершина с ключом 2 имеет вид



Задача. Включить в T_k вершину с ключом 2^k.

Задача имеет единственное решение.

T_4:

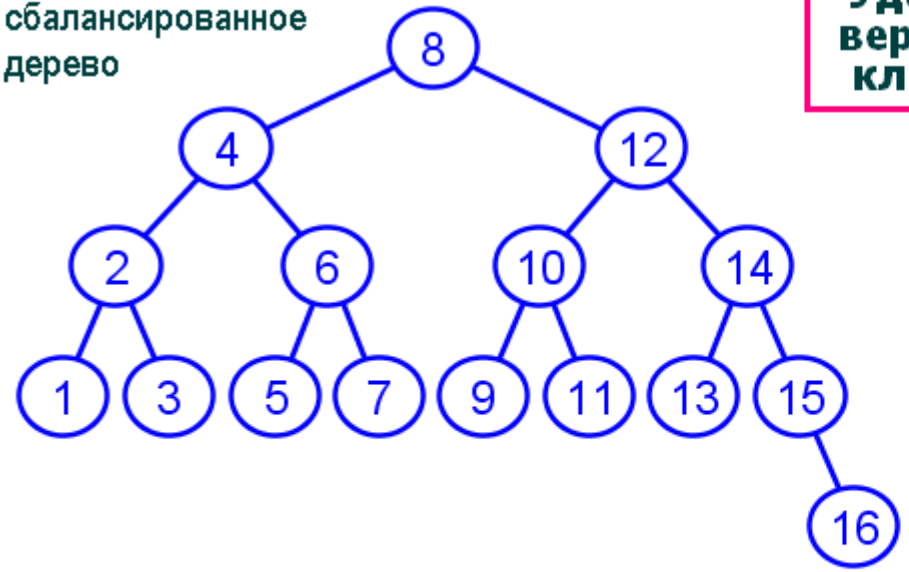


На своих местах остались **0** вершин

Сложность **O(n)**

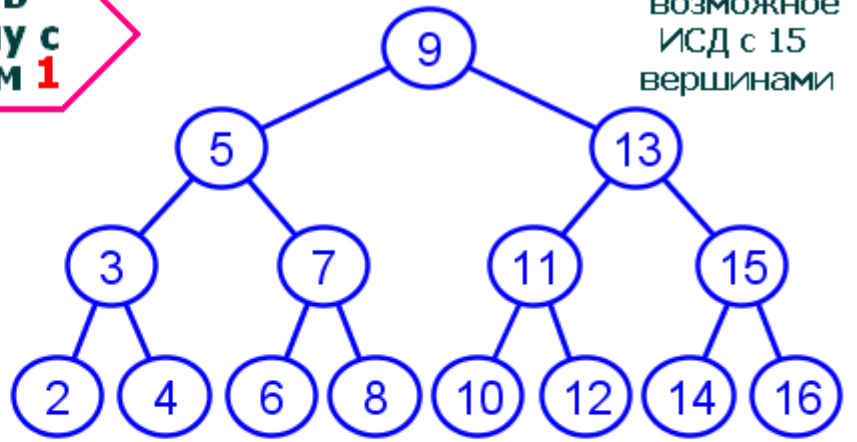
Удалить в ИС дереве заданную вершину

Идеально сбалансированное дерево



Удалить вершину с ключом 1

Единственно возможное ИСД с 15 вершинами



На своих местах остались **0** вершин

Сложность **O(n)**

Оценки сложности

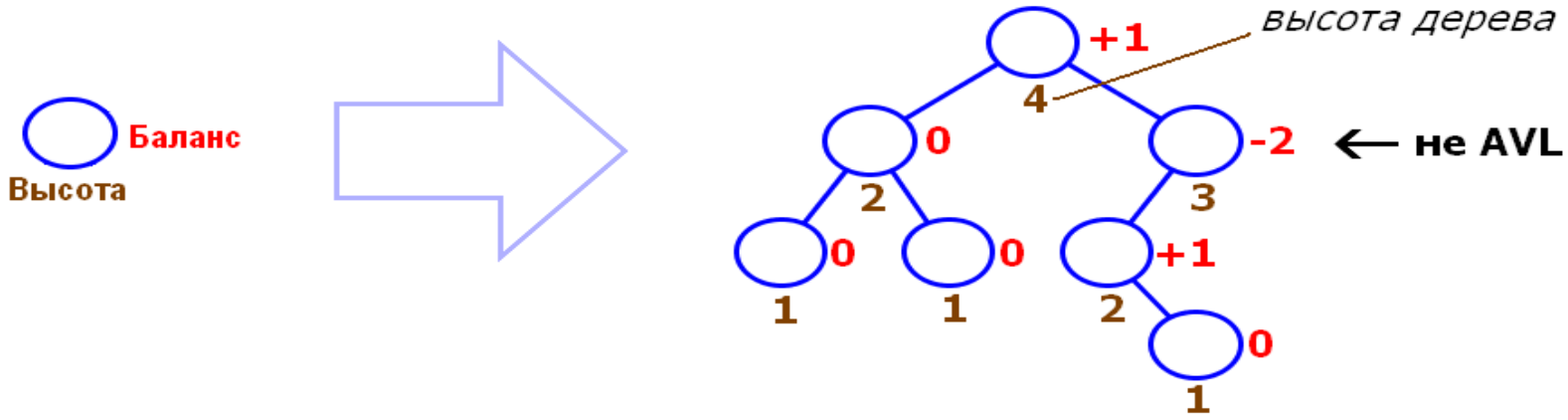
AVL-деревья

AVL-дерево – дерево поиска, в котором
баланс каждой вершины есть –1, 0 или +1.

Пусть var T : pDoTree; { – ссылка на ∀ вершину дерева }

$$\text{Высота}(T) = \begin{cases} 1+\max(\text{Высота}(T^{\wedge}.\text{Right}),\text{Высота}(T^{\wedge}.\text{Left})) \\ 0, \text{ если } T = \text{nil} \end{cases}$$

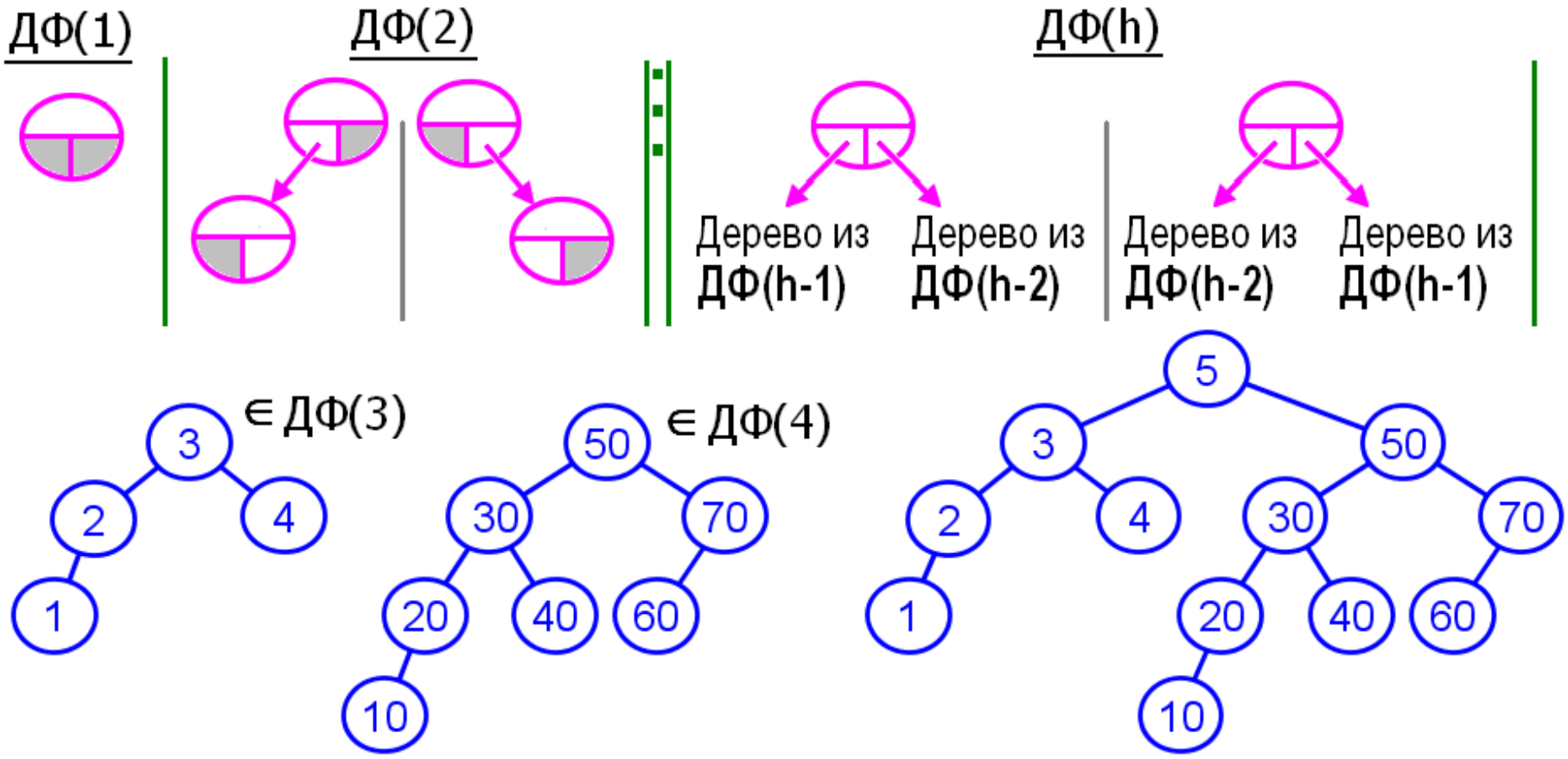
Баланс(T) = Высота(T^.Right) – Высота(T^.Left)



Рассуждение о высоте AVL-деревьев 1/4

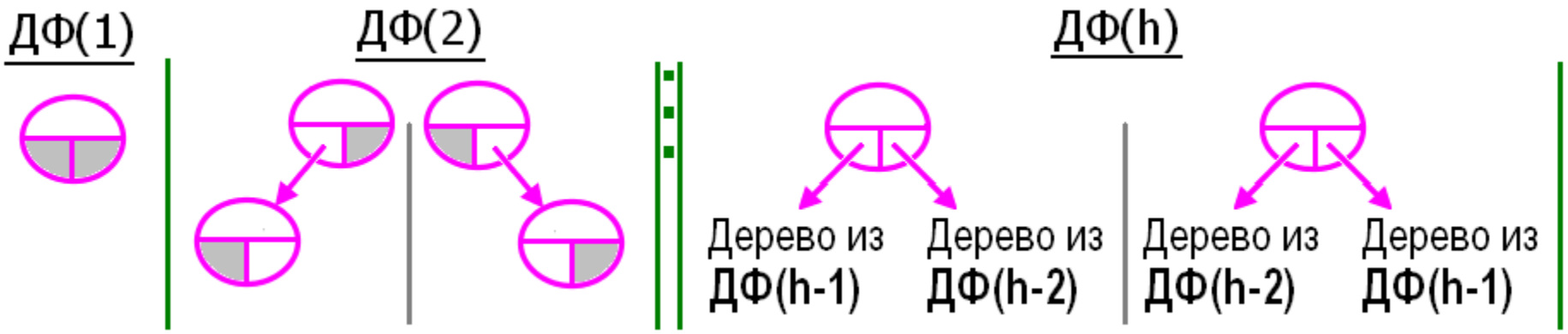
Дерево Фибоначчи – бинарное дерево с корнем, в котором баланс всех вершин, отличных от листьев, есть –1 или +1.

ДФ(h) – деревья Фибоначчи высоты h:



Рассуждение о высоте AVL-деревьев 2/4

ДФ(h) – деревья Фибоначчи высоты h:



Все деревья из **ДФ(h)** содержат одинаковое к-во вершин Φ_h.

Φ_h : 1, 2, 4, 7, 12, 20, 33, … $\Phi_h = \Phi_{h-2}+\Phi_{h-1}+1$

F_n : 1, 1, 2, 3, 5, 8, 13, 21, 34, … *числа Фибоначчи*

$\Phi_h = F_{h+2}-1$

Числа Фибоначчи: $F_n = F_{n-2} + F_{n-1}$, $F_1 = 1$, $F_2 = 1$,

Формула Бине: $F_n = \Omega^n/q + \Delta_n/q$, $q = \mathrm{sqrt}(5)$, $\Omega = (1+q)/2$, $\Delta_n \to 0$.

Рассуждение о высоте AVL-деревьев 3/4

Пусть AVL(h) – AVL-деревья высоты h.

⇒ В классе AVL(h) наименьшее количество вершин имеют деревья Фибоначчи. (Индукция по h + От противного)

Пусть **T** – AVL-дерево, содержащее **n** вершин.

Обозначим **H** высоту **T**.

⇒ $\Phi_H \le n \Rightarrow F_{H+2} \le n + 1$

⇒ $\Omega^{H+2} + \Delta_{H+2} \le q(n+1)$

⇒ $H \le \log_\Omega(q(n+1) - \Delta_{H+2}) - 2$

⇒ $\exists\, C_2 > 0 :$ $\boxed{H \le C_2 \log_2 n}$

ИС → $\exists\, C_1 > 0 :$ $C_1 \log_2 n < H$ ← интересно

Рассуждение о высоте AVL-деревьев 4/4

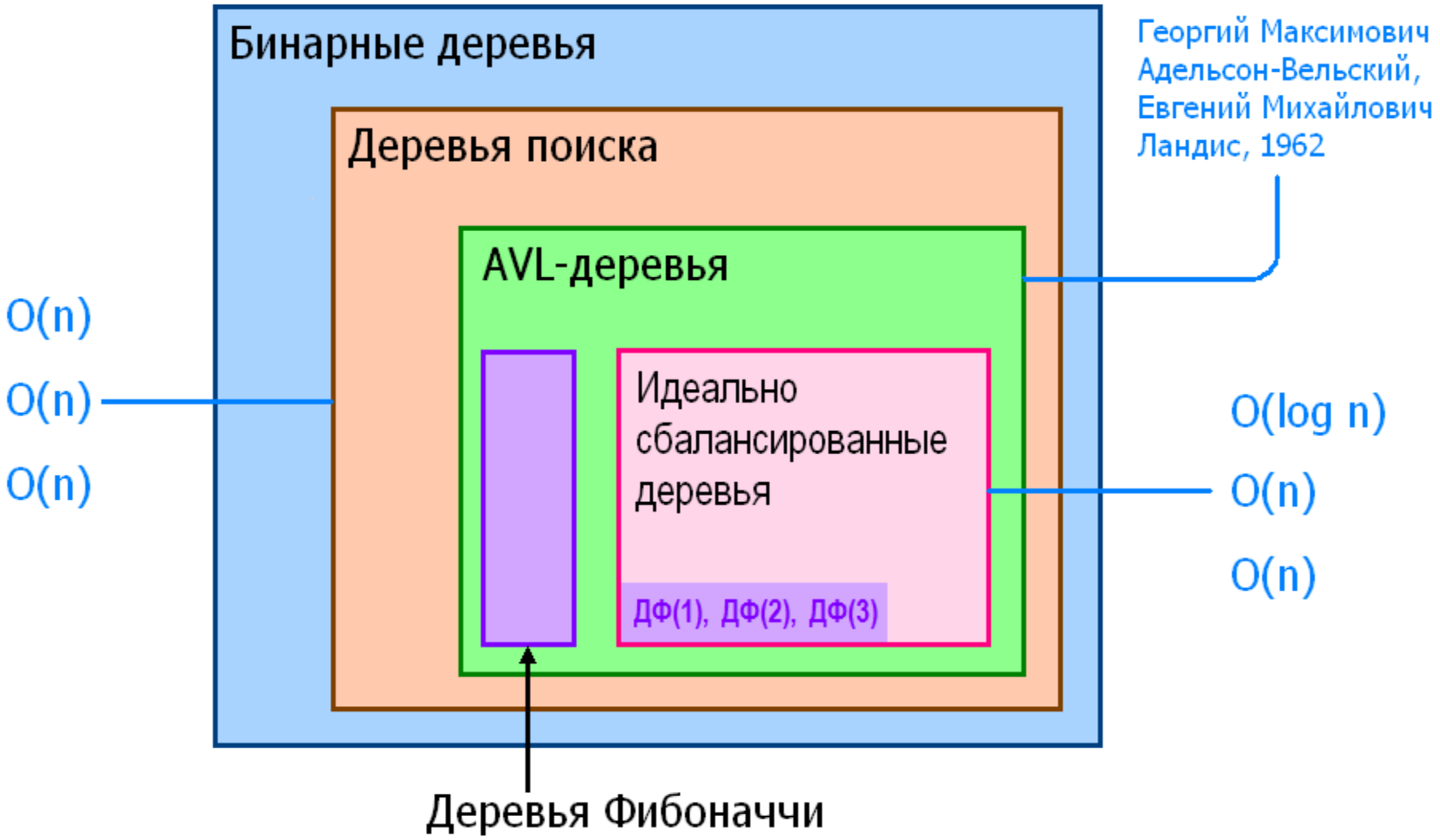
Существуют константы C_1 и C_2 такие, что для любого AVL-дерева **T** имеет место неравенство

$$C_1 \log_2(n) \leq \text{высота}(\mathbf{T}) \leq C_2 \log_2(n),$$

где n – количество вершин в T.

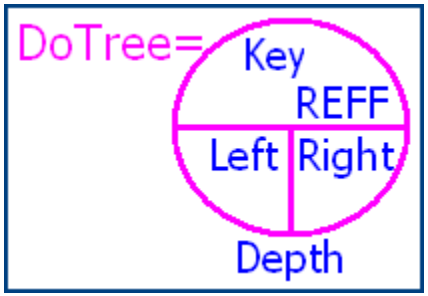
$C_1 \approx 1.00$ $C_2 \approx 1.44$

Классы деревьев



Пусть

```
program … ;
type          Info = record . . . end;
          pRecord = ^Info;
            DoTree = record      Key : integer;
                                 Left,Right : pDoTree;
                                      REFF : pRecord;
            { new }                  Depth : integer;
                          end;
          pDoTree = ^DoTree;
                          ...
var        T : pDoTree;   ...
```



```
            (* ------------ сервис ---------------- *)

            (* ------  procedure | function  ------ *)
            function FindSTr(T : pDoTree; K : integer) : pDoTree;
            begin . . . end;
```

```
begin   T:=nil;  { породить пустое дерево }
          ...
end.
```

Сервис / 1

Дано: **P** – вершина. Получить L – высота левого п/дерева | **0**
и **R** – высота правого п/дерева | **0**.

```
procedure dSons(var L,R : integer; P : pDoTree);
begin   L:=0;
        R:=0;
        if P <> nil then with P^ do begin
           if Left  <> nil then L:= Left^.Depth;
           if Right <> nil then R:=Right^.Depth
        end                                   end;
```

Вычислить баланс вершины **P**.

```
function Ba(P : pDoTree): integer;
   var L,R : integer;
begin   dSons(L,R,P);
        Ba:=R-L                    end;
```

Сервис / 2

Корректировать поле **P^.Depth**

```
procedure xDepth(P : pDoTree);
   var L,R : integer;
begin  dSons(L,R,P);
       if L <  R   then L:=R;                  { L:=max(L,R) }
       if P <> nil then P^.Depth:=L+1   end;
```

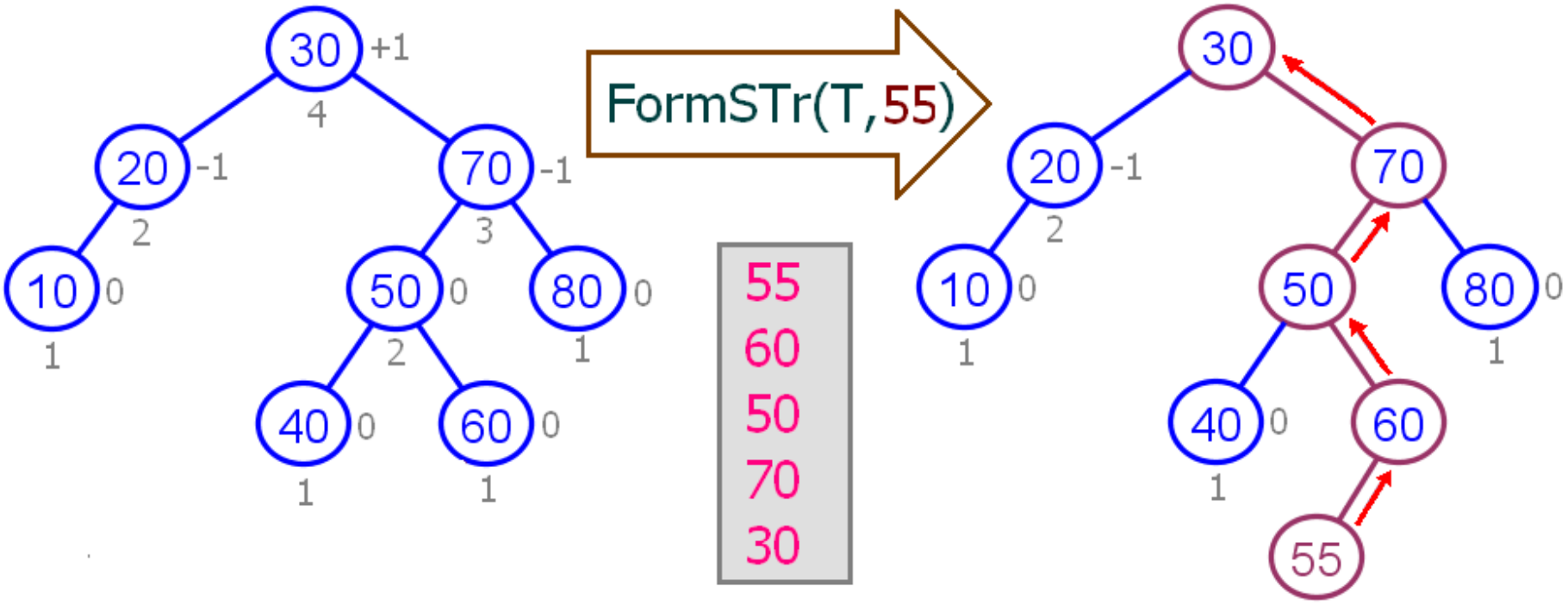
Операции сборки/разборки дерева

```
procedure Link(var A,B,C : pDoTree);
begin   A^.Left :=B;
        A^.Right:=C                    end;

procedure Take(var A,B,C : pDoTree);
Begin   B:=A^.Left;
        C:=A^.Right                    end;
```

от FormSTr к FormAVL / 1

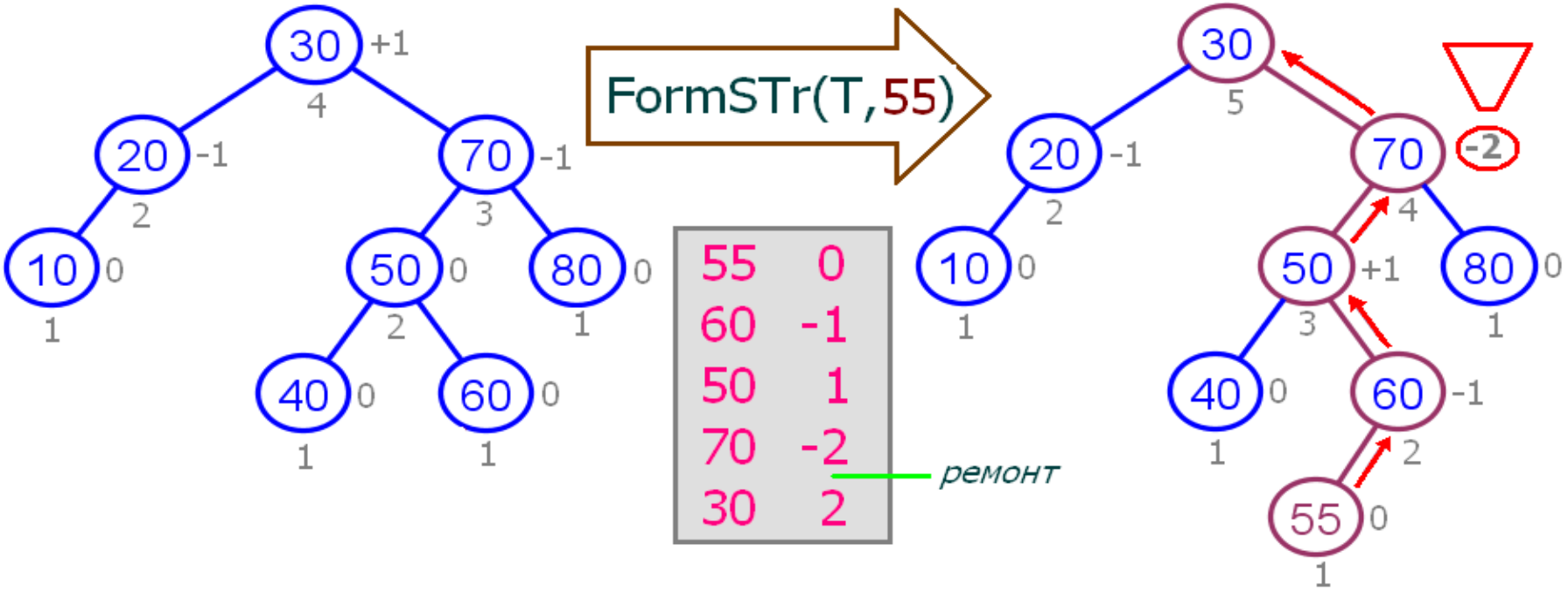
Напоминание

```
function FormSTr(var T : pDoTree; K : integer) : pDoTree;
begin    if  T = nil  then Born(T,K);
         with T^ do
               if Key = K then FormSTr:=T
         else if Key < K  then FormSTr:=FormSTr(Right,K)
         else                  FormSTr:=FormSTr(Left  ,K);
         writeln(T^.Key)   { отладочная печать }     end;
```

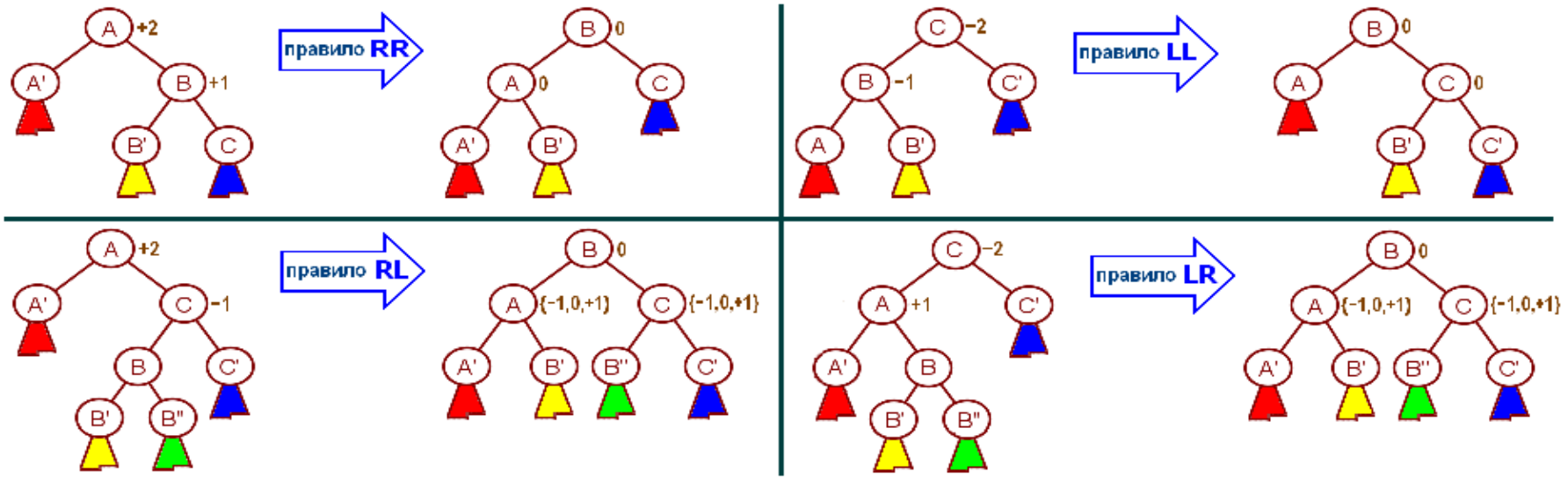


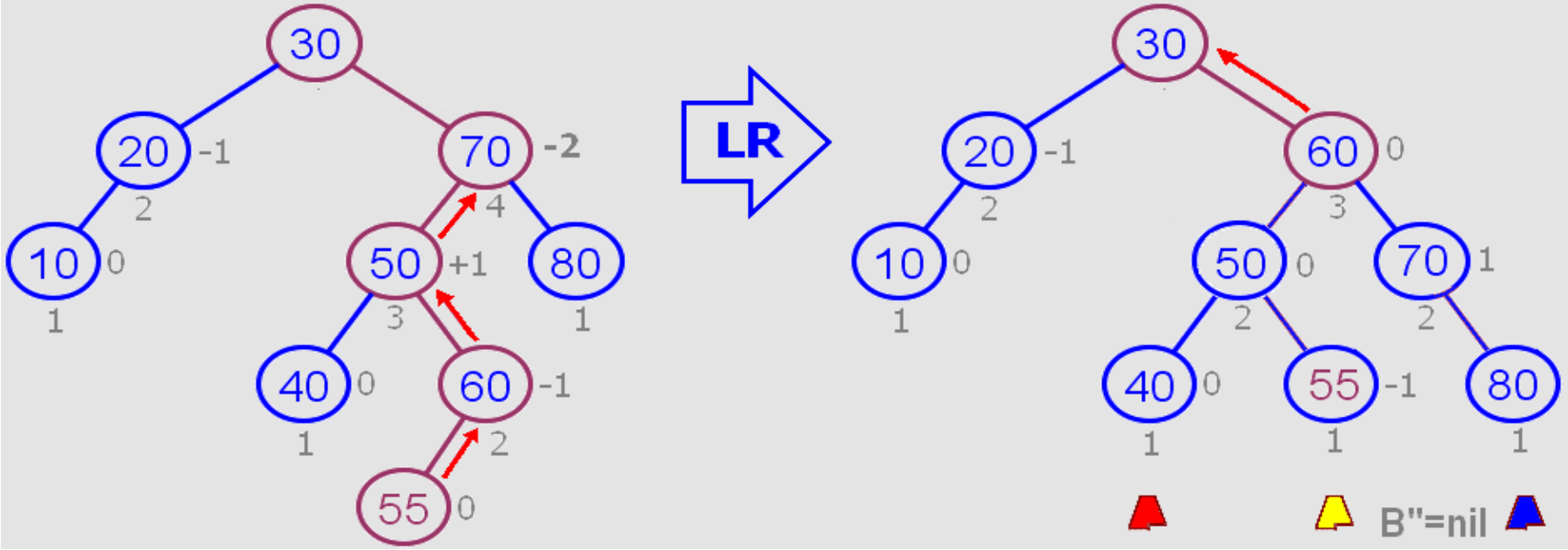
от FormSTr к FormAVL / 2

```
function FormSTr(var T : pDoTree; K : integer) : pDoTree;
begin    if  T = nil  then Born(T,K);
         with T^ do
               if Key = K then FormSTr:=T
         else if Key < K  then FormSTr:=FormSTr(Right,K)
         else                  FormSTr:=FormSTr(Left  ,K);
         xDepth(T);
         writeln(T^.Key:3,Ba(T):3)                    end;
```

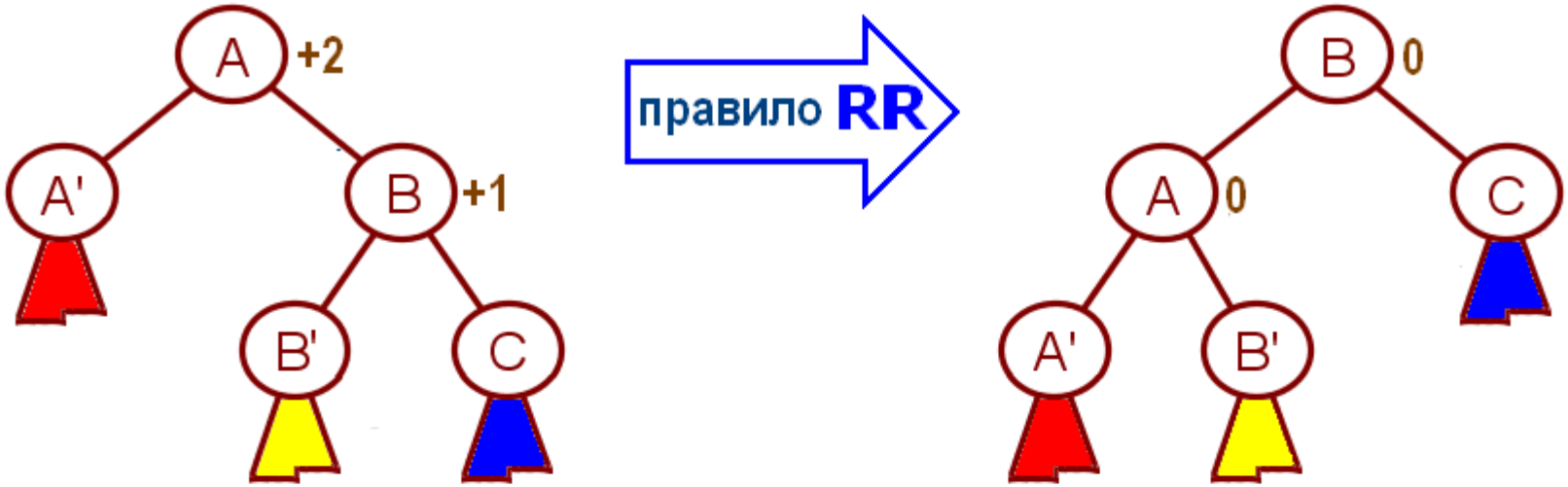


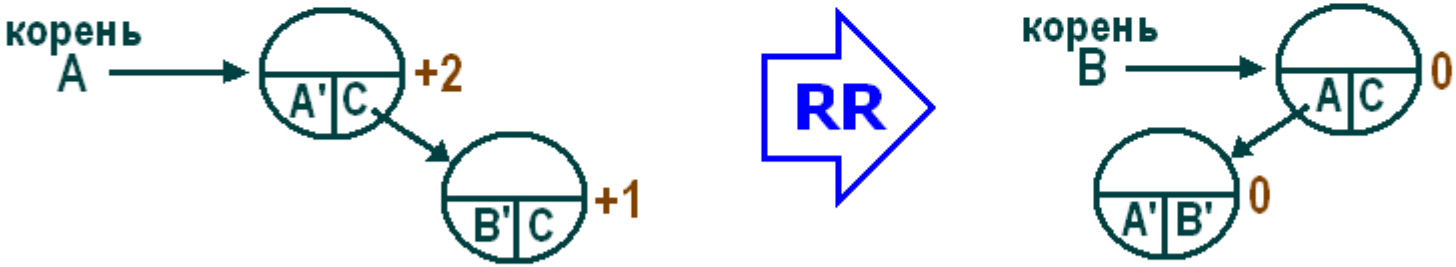
от FormSTr к FormAVL / правила корректировки





Правило **RR**



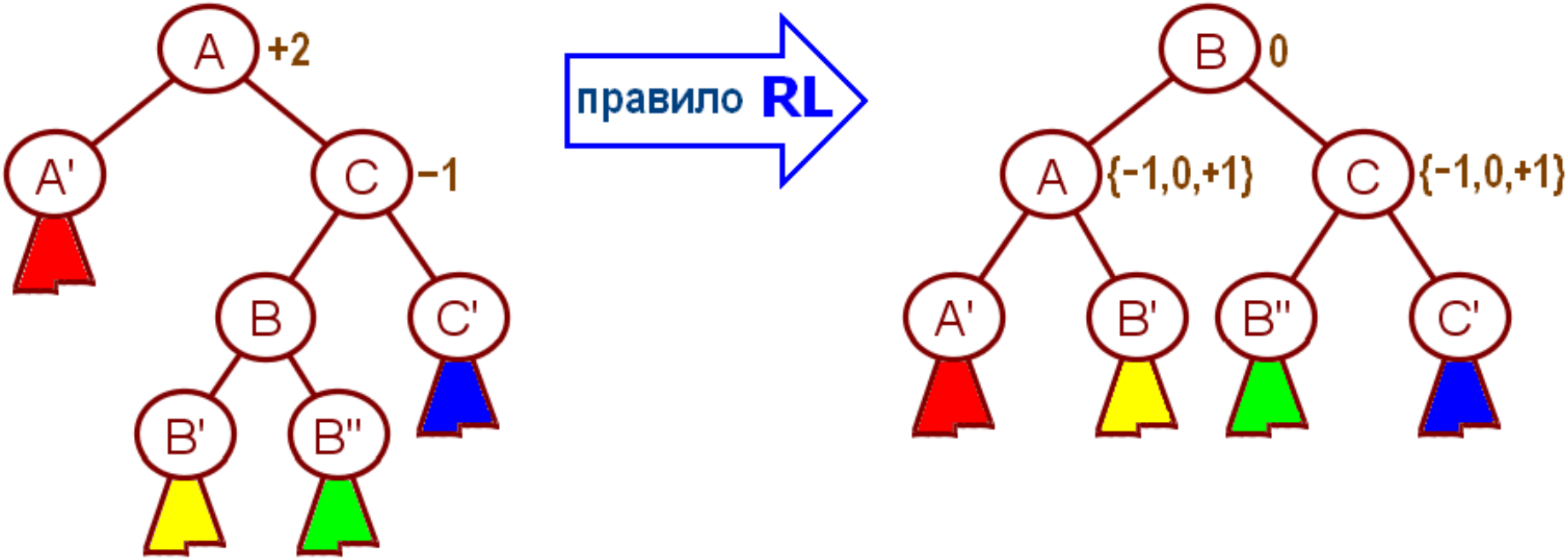


***Левая часть правила*:**

высота(B) – высота(A') = 2
высота(C) – высота(B') = 1
высота(B) = 1 + max(высота(B'),высота(C))

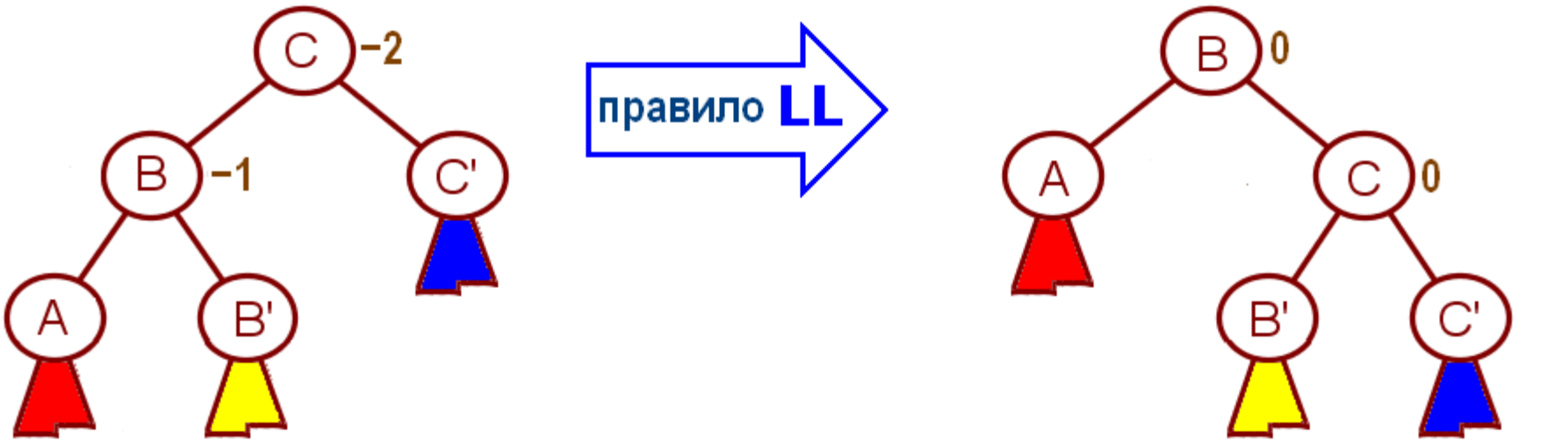
высота(B') = высота(A')
высота(C) = высота(A')+1

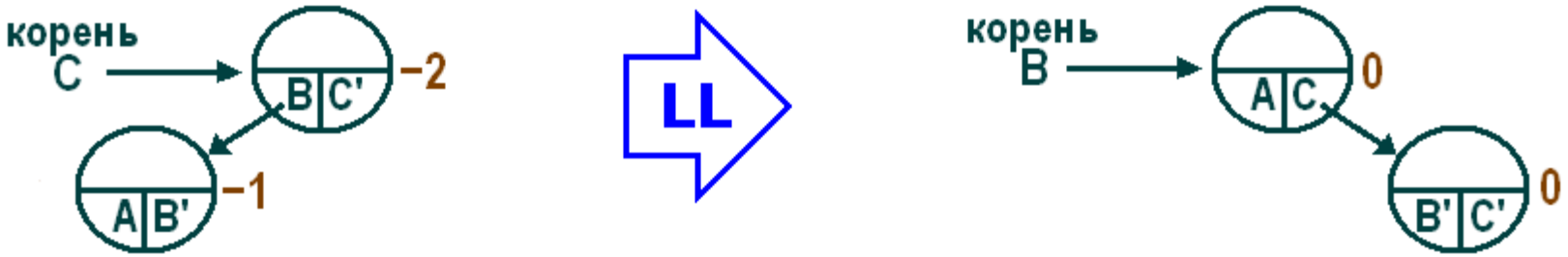
Правило **RL**


A
+2
A'
C
−1
B
C'
B'
B''
правило RL
B
0
A
{−1,0,+1}
C
{−1,0,+1}
A'
B'
B''
C'


корень
A
A'
C
+2
B
C'
−1
B'
B''
RL
корень
B
A
C
0
A'
B'
{−1,0,+1}
B''
C'
{−1,0,+1}

Правило **LL**

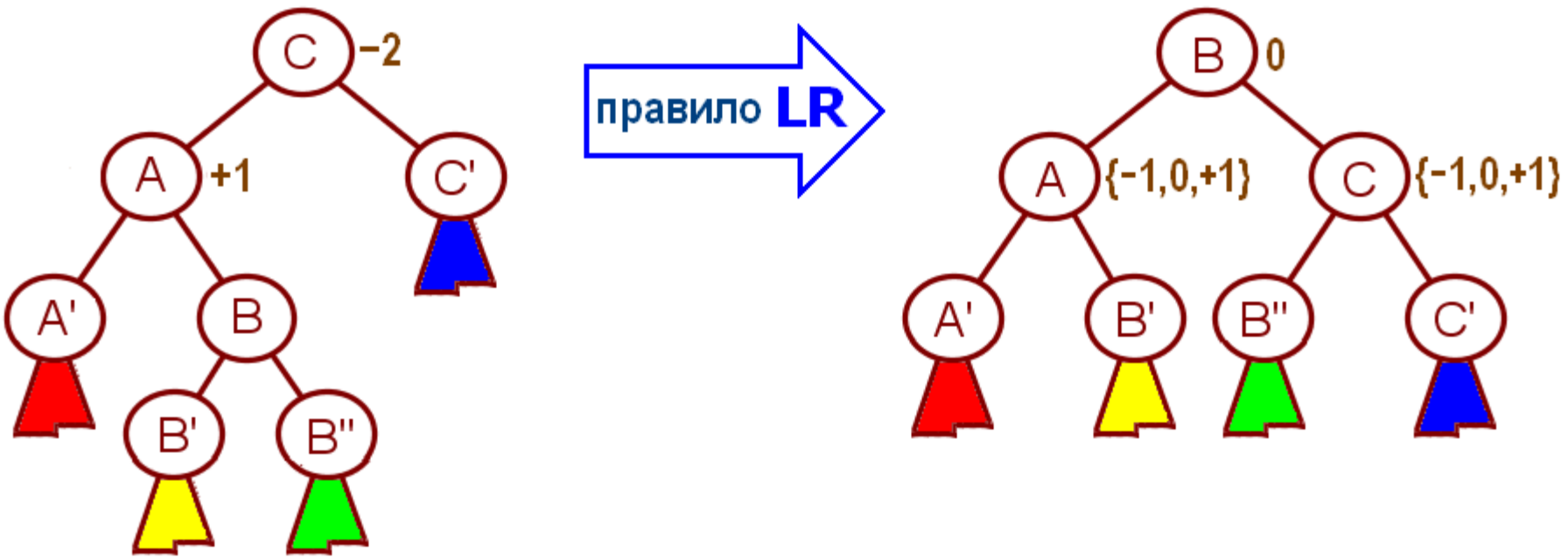


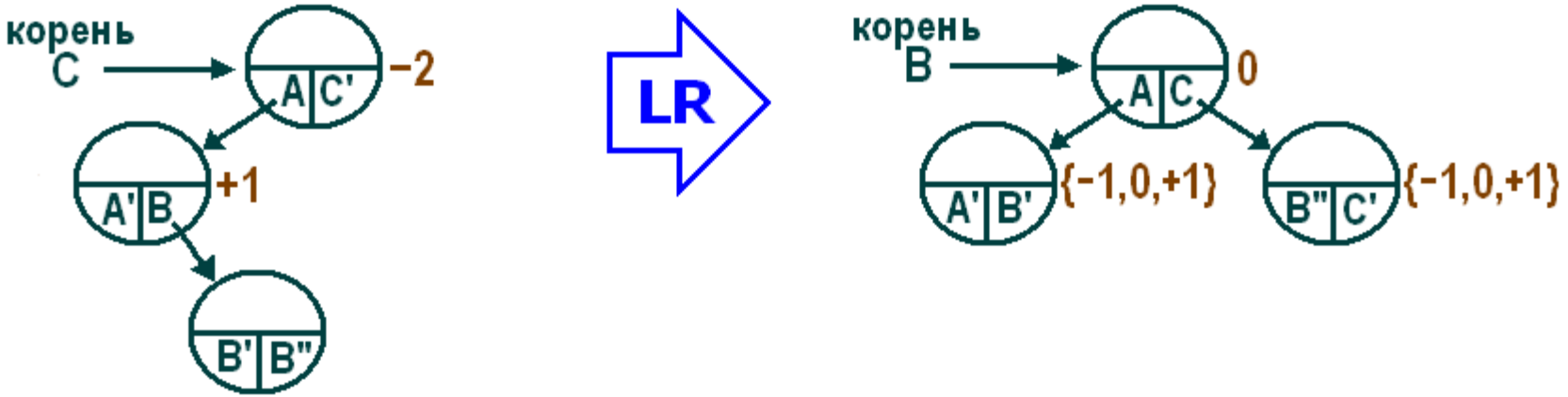



```
if  Ba(C)          = -2 then
if  Ba(C^.Left) = -1 then begin
    { действие LL }
end;
```

{ проверка применимости }

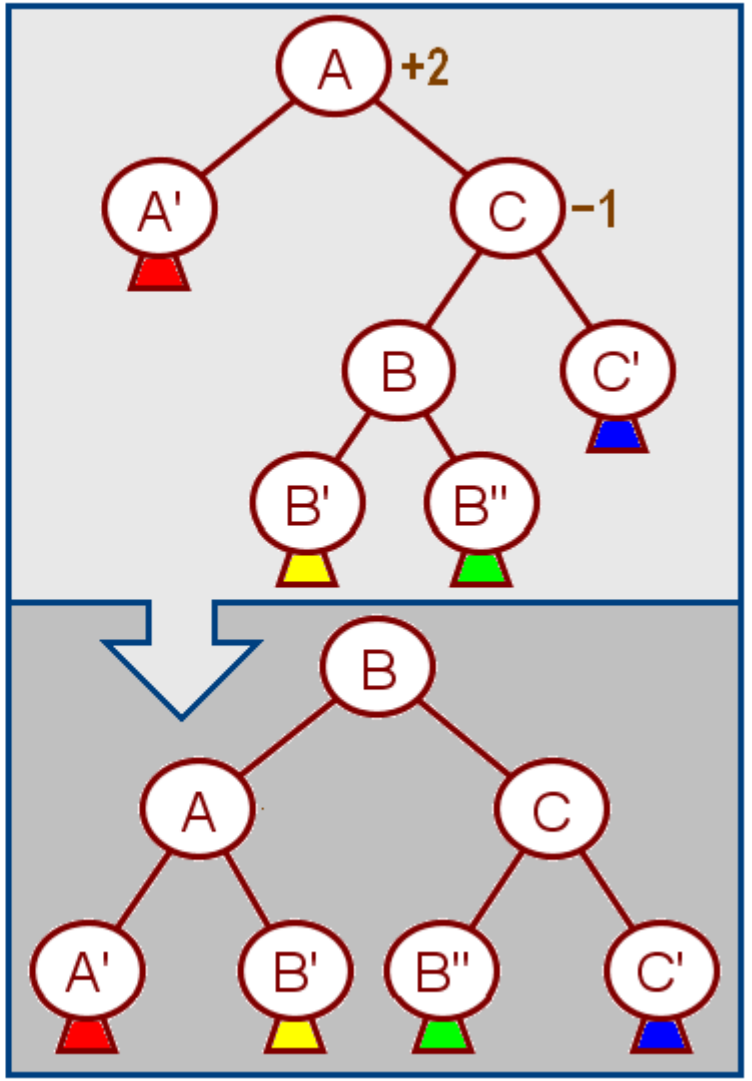
Правило **LR**


C
−2
A
+1
C'
A'
B
B'
B''
правило LR
B
0
A
{−1,0,+1}
C
{−1,0,+1}
A'
B'
B''
C'


корень
C
A C'
−2
A' B
+1
B' B''
LR
корень
B
A C
0
A' B'
{−1,0,+1}
B'' C'
{−1,0,+1}

Правило **RL** ➡ function **RL**

```
function RL(var ROOT : pDoTree) : boolean;
  var  A, A1, B, B1, B2, C, C1 : pDoTree;
begin   A:=ROOT;
        RL:=false;
        if  Ba(A)          = +2 then
        if  Ba(A^.Right) = −1 then begin
            Take(A,A1,C  );
            Take(C,B  ,C1);
            Take(B,B1,B2);
            {1}   Link(A,A1,B1);  xDepth(A);
            {2}   Link(C,B2,C1);  xDepth(C);
            {3}   Link(B,A  ,C  );  xDepth(B);
            ROOT:=B;
            RL:=true
        end
end;
```



Включить в AVL-дерево **T** вершину с ключом **K**

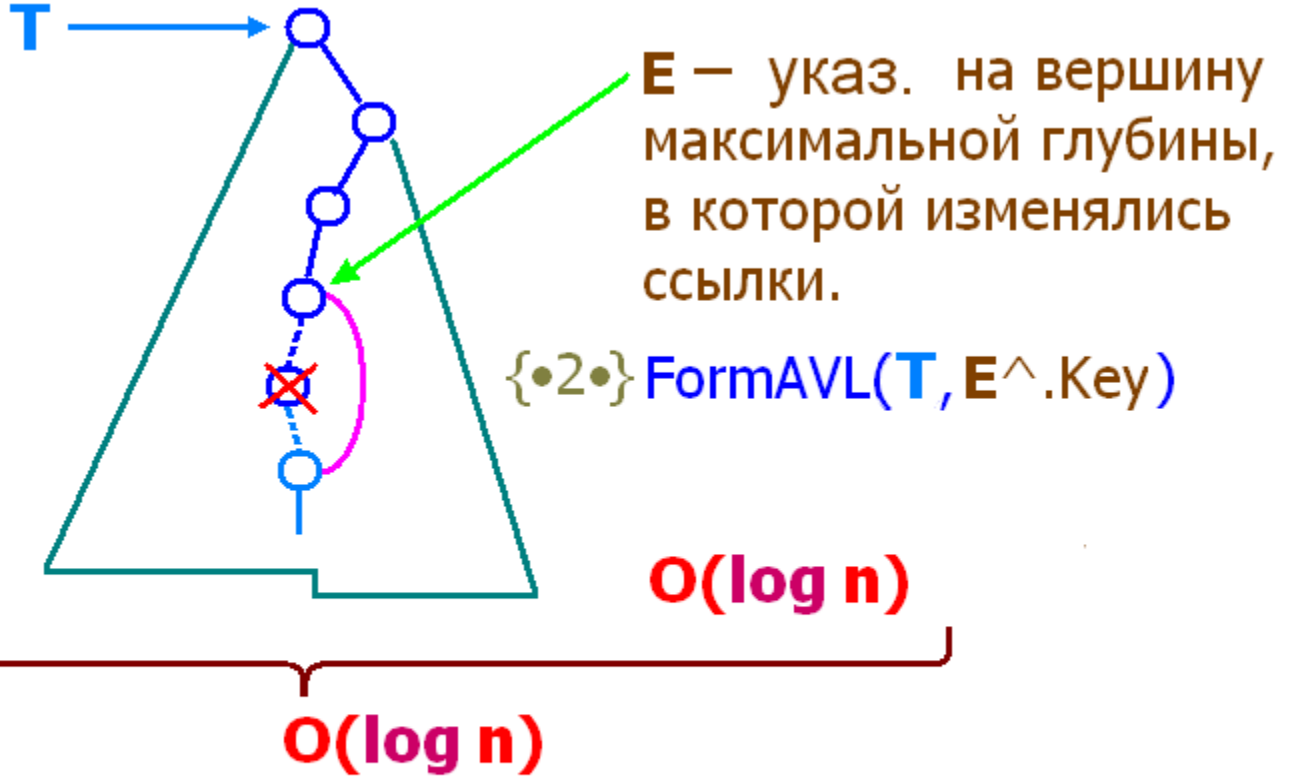
```
function FormAVL(var T : pDoTree; K : integer) : pDoTree;
begin   if  T = nil  then begin
            Born(T,K);
                  T^.Depth:=1
        end;

        with T^ do
              if Key = K  then FormAVL:=T
        else if Key < K  then FormAVL:=FormAVL(Right,K)
        else                  FormAVL:=FormAVL(Left  ,K);

        if not LL(T) then
        if not LR(T) then
        if not RL(T) then
        if not RR(T) then xDepth(T)
end;
```

O(log n)

Удалить вершину в AVL-дереве

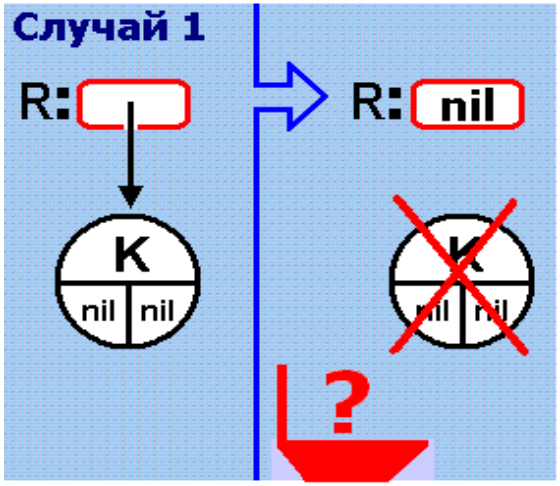
Дано: T – указатель на AVL-дерево; K – ключ

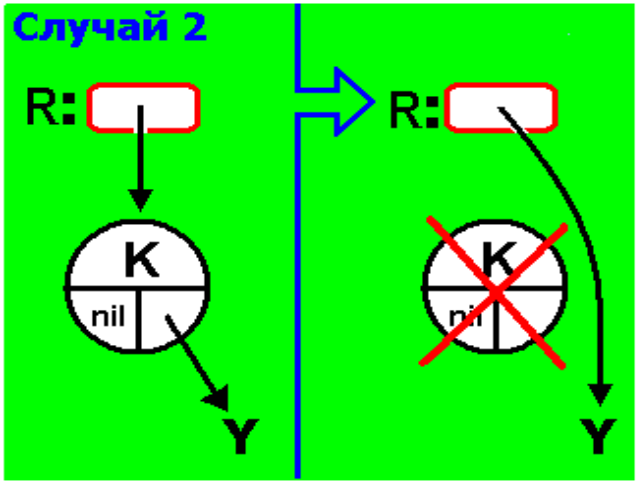
Подход

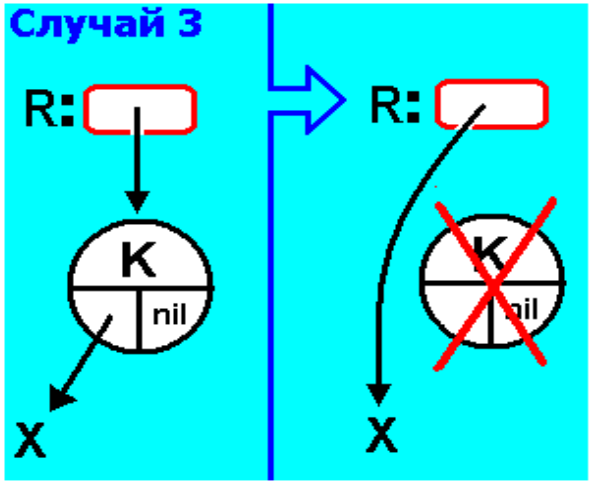


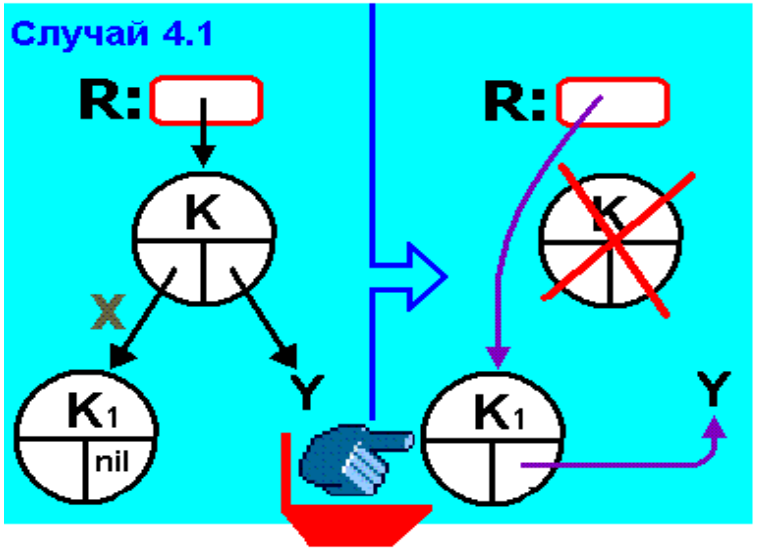
KillSTr(…) ⟶ E:=mKillSTr(…)

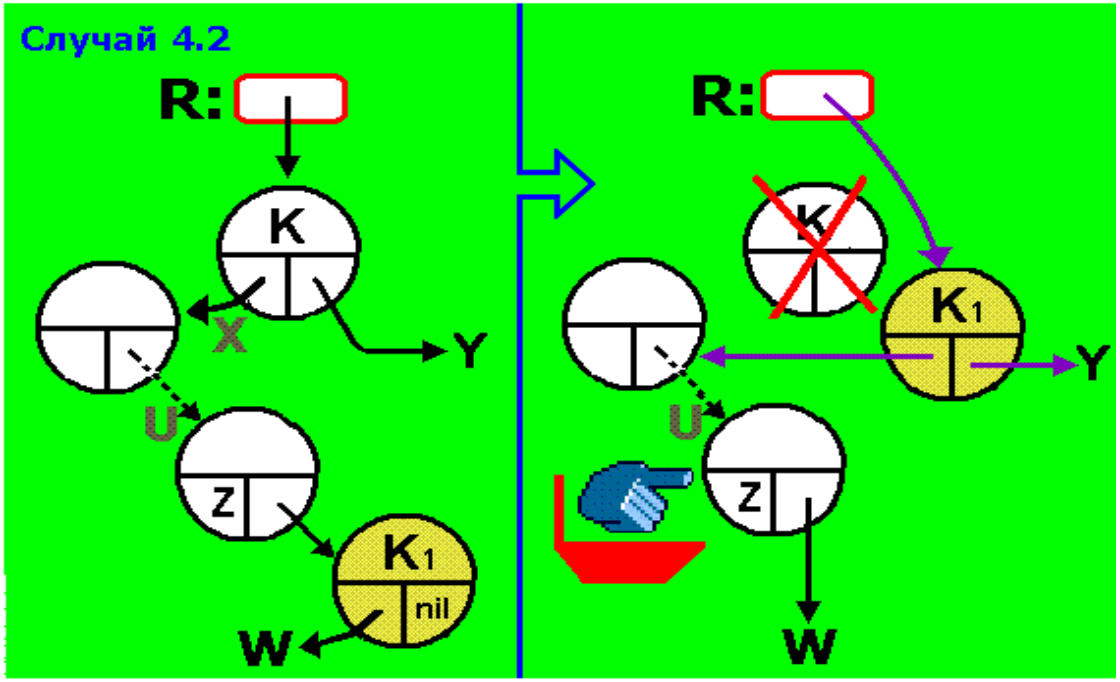
Вопросы реализации



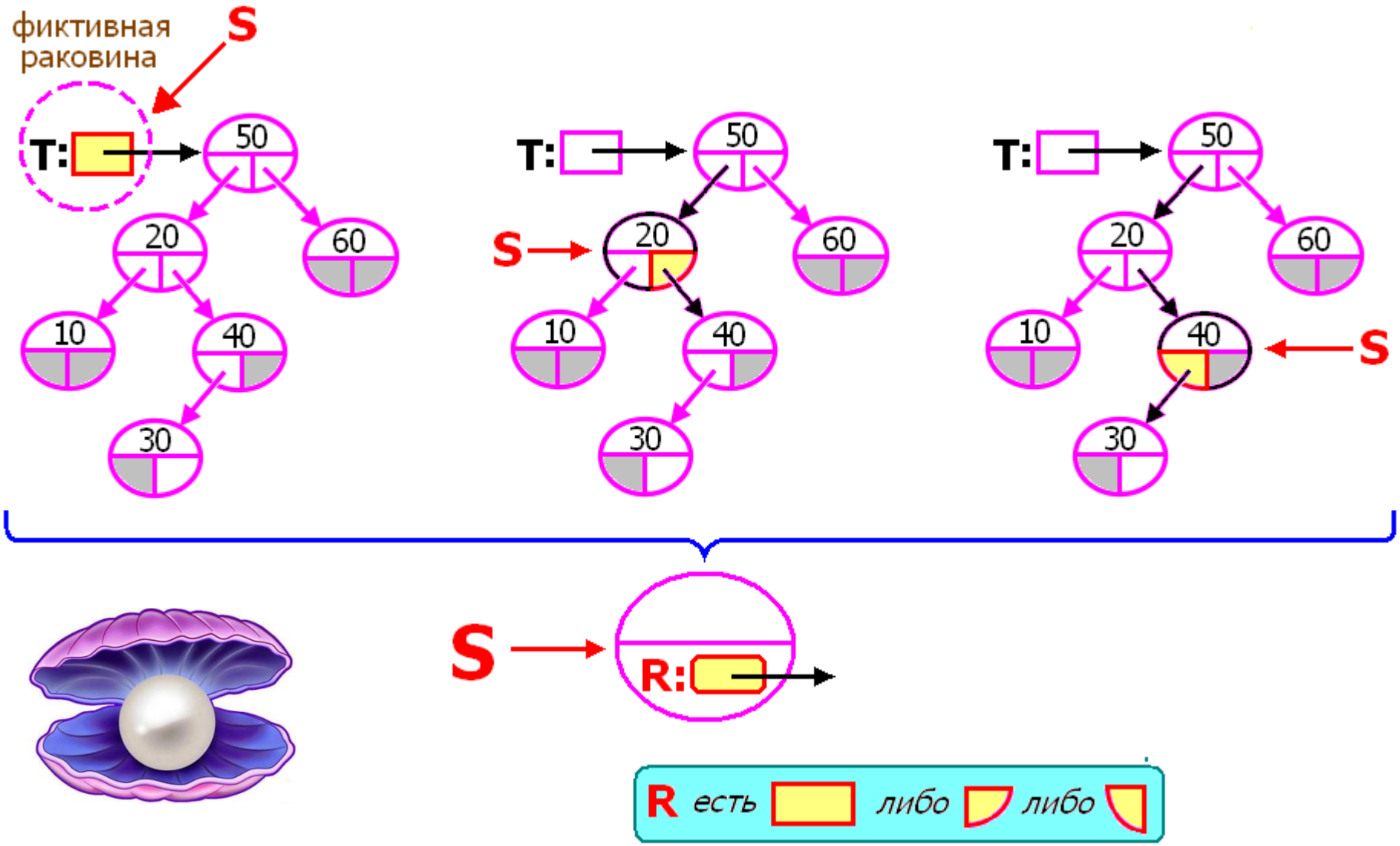








Пары ссылок


фиктивная раковина
S
T:
50
20
60
10
40
30
T:
50
S
20
60
10
40
30
T:
50
20
60
10
40
S
30
S
R:
R есть
либо
либо

procedure **KillSTr** → function **mKillSTr**

mKillSTr(S,T,K) = { pDoTree — *указ. на вершину max глубины, в которой изменились ссылки*; nil — *удаление не потребовалось* }

```
function mKPECT(      S : pDoTree;
                  var T : pDoTree) : pDoTree;   begin     end;
```

```
function mKillSTr(      S : pDoTree;
                    var T : pDoTree;
                        K : integer) : pDoTree;
begin   if     T = nil
        then mKillSTr:=nil
        else  with T^ do
              if Key = K then mKillSTr:=mKPECT(S, T)
        else  if Key < K then mKillSTr:=mKillSTr(T, Right, K)
        else                  mKillSTr:=mKillSTr(T, Left   , K)  end;
```

Удалить вершину в AVL-дереве

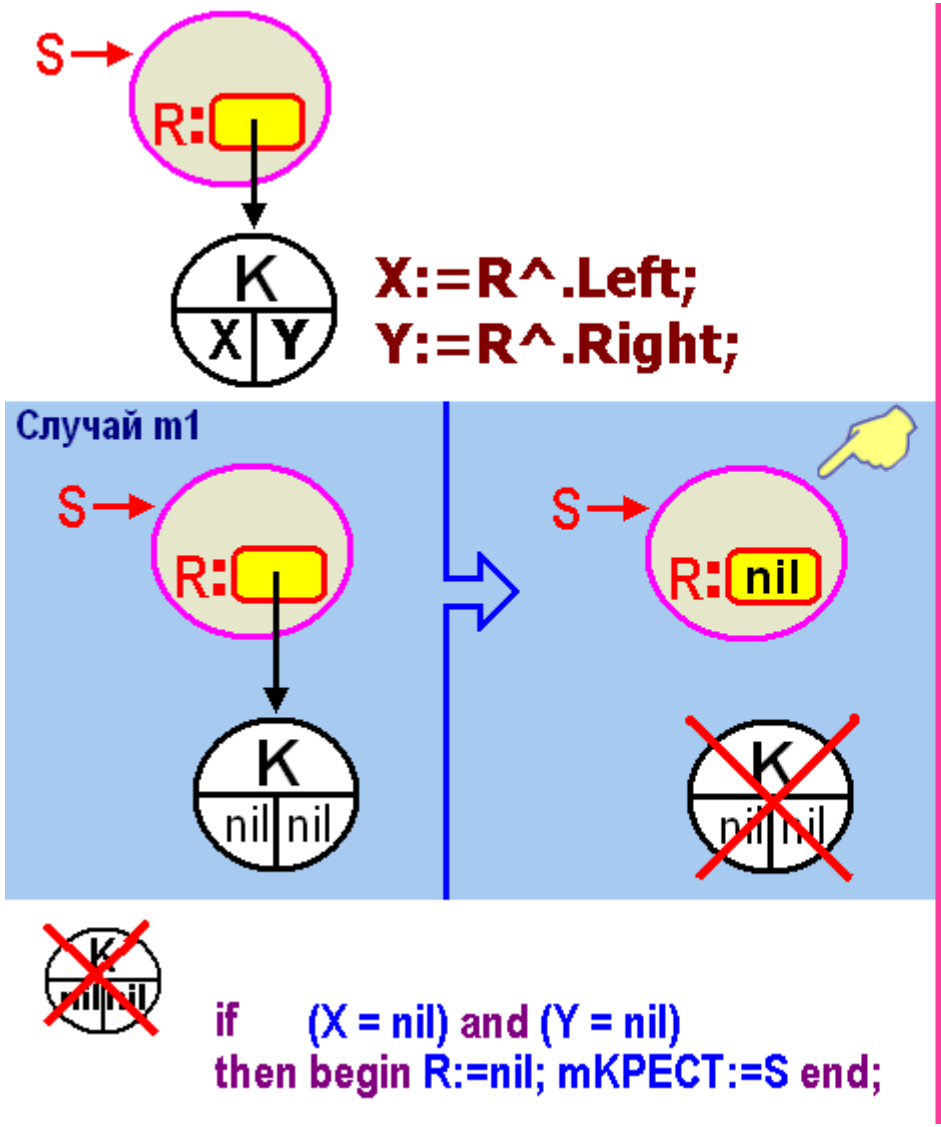
Дано: T – указатель на AVL-дерево; K – ключ.
Удалить (если она существует) вершину с ключом K из T.

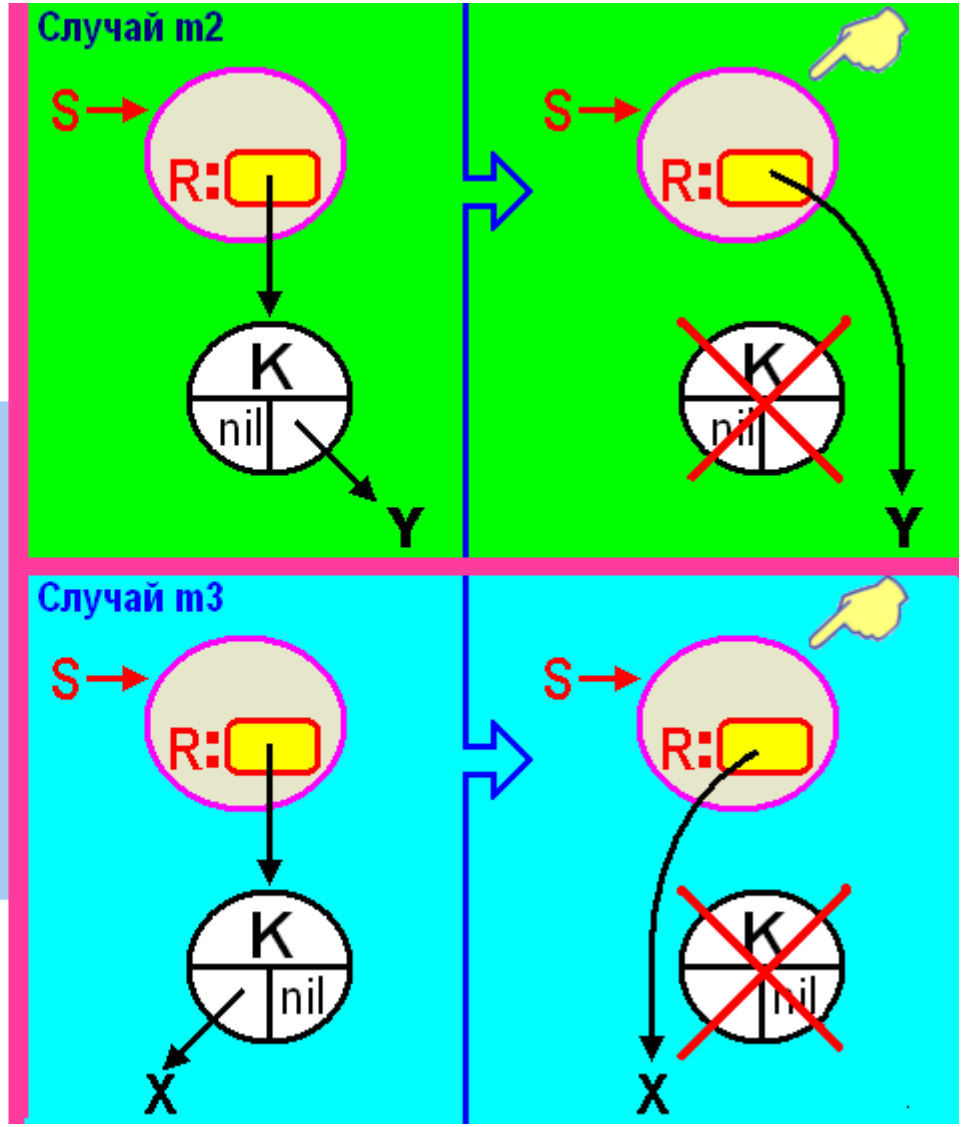
```
procedure KillAVL(var T : pDoTree; K : integer);
    var  E, W, fict : pDoTree;
begin    if T = nil then Exit;
         new(fict);                    { фиктивная раковина для T }
{•1•}    E:=mKillSTr(fict, T, K);
         if  T <> nil  then
         if  E <> nil  then
         if  E = fict    then W:=FormAVL(T, T^.Key)
{•2•}                    else  W:=FormAVL(T, E^.Key);
         dispose(fict)                                  end;
```

O(log n)

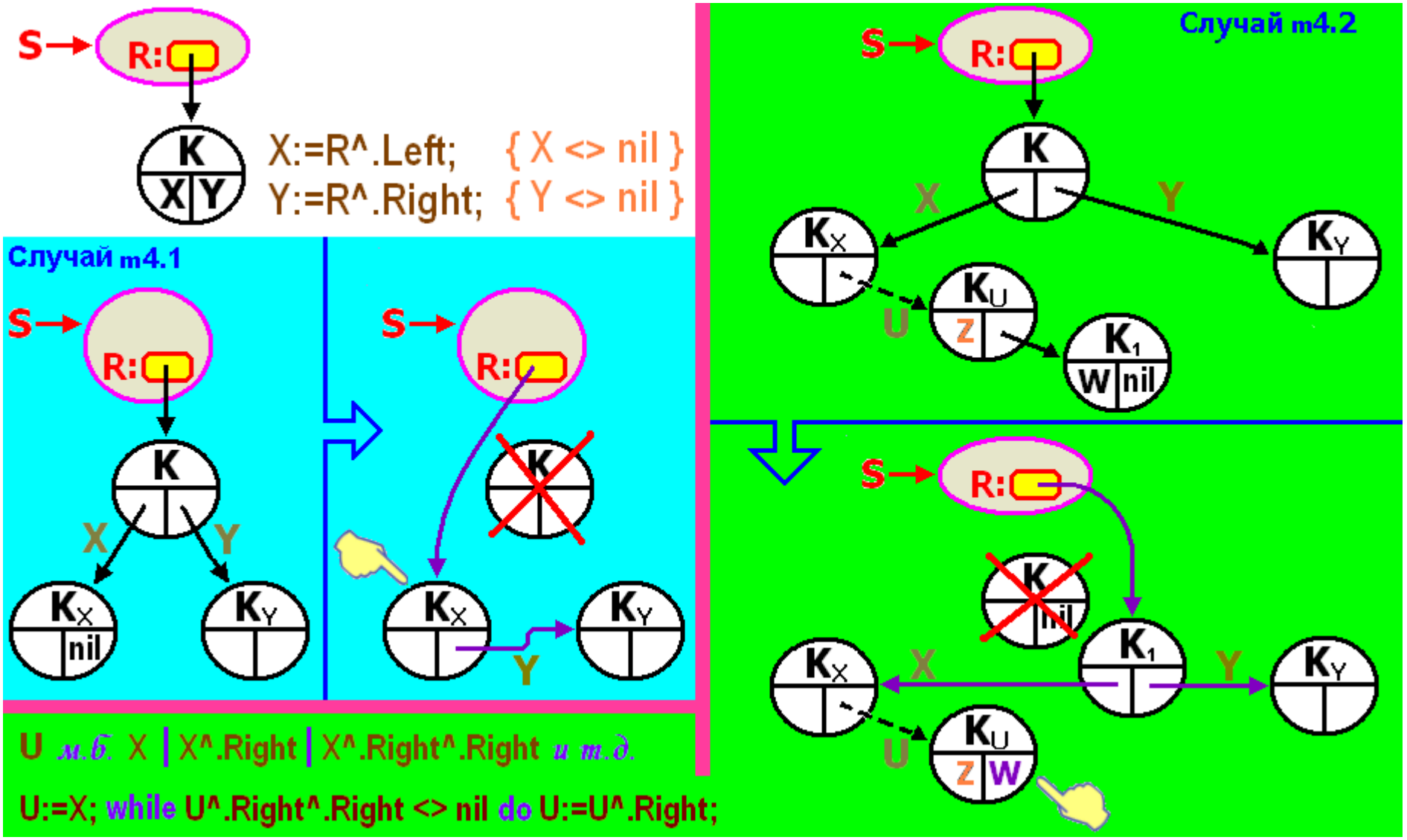
```
procedure KillSTr(T : pDoTree; K : integer);
begin   if mKillStr(nil, T,K) = nil  then;  end;
```

procedure **KPECT** → function **mKPECT** / 1



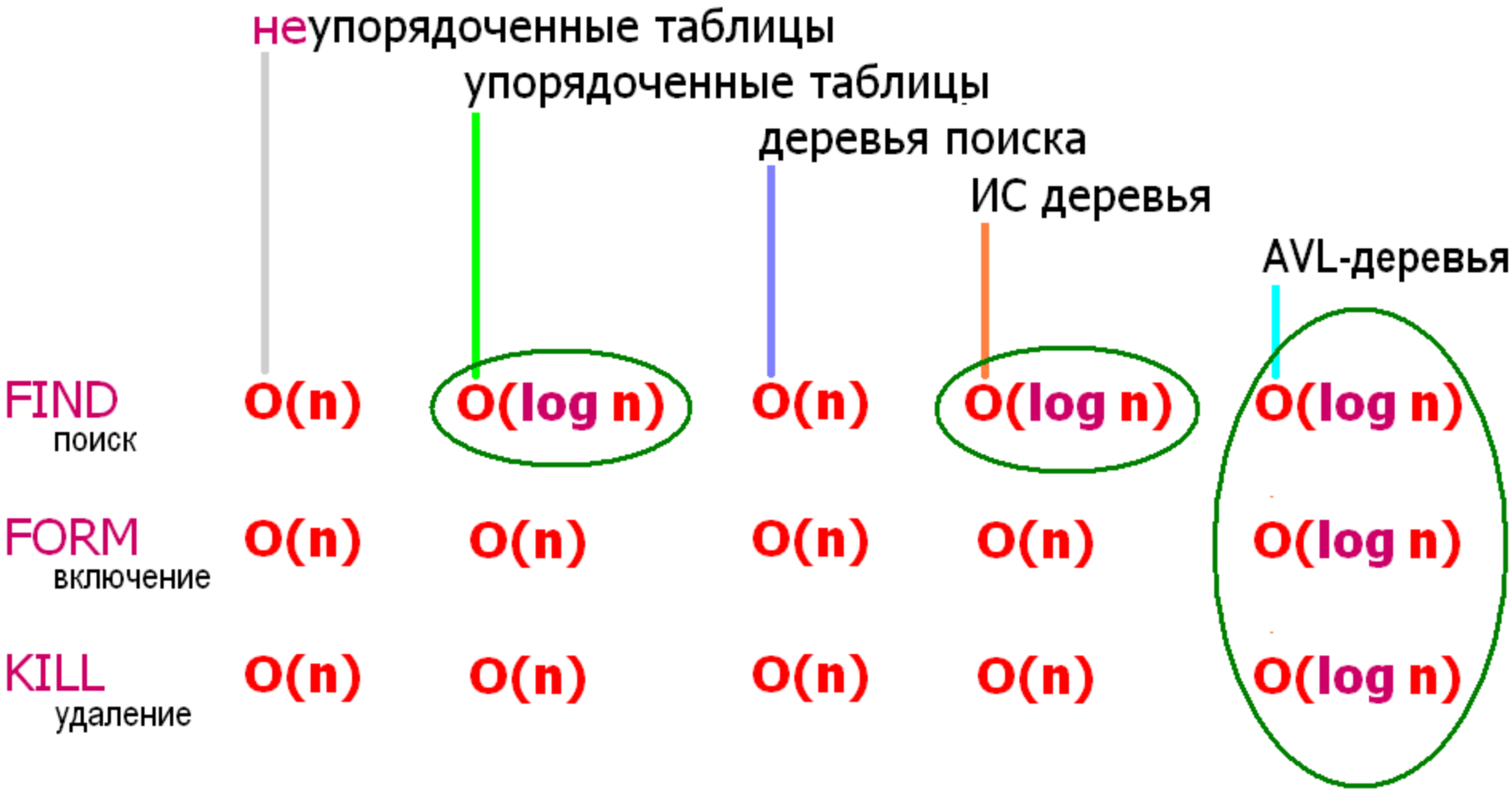


procedure **KPECT** → function **mKPECT** / 2



procedure **KPECT** → function **mKPECT** / 3

```
function mKPECT(S : pDoTree; var R : pDoTree) : pDoTree;
    var X,Y,U,W : pDoTree;
begin   X:=R^.Left;
        Y:=R^.Right;
        if R^.REFF <> nil then  dispose(R^.REFF);
                                dispose(R);
{ m1 }        if (X = nil) and (Y = nil) then begin R:=nil; mKPECT:=S end
{ m2 } else   if  X = nil                then begin R:=Y;   mKPECT:=S end
{ m3 } else   if                Y = nil  then begin R:=X;   mKPECT:=S end
{m4.1} else   if  X^.Right = nil         then begin R:=X;   mKPECT:=R;
                                                            R^.Right:=Y end
{m4.2} else   begin
            U:=X;
            while U^.Right^.Right <> nil do U:=U^.Right;
              R:=U^.Right;
              W:=R^.Left;          U^.Right:=W;        { ! }
                   R^.Left:=X;     R^.Right:=Y;
              mKPECT:=U
end         end;
```


неупорядоченные таблицы
упорядоченные таблицы
деревья поиска
ИС деревья
AVL-деревья
FIND
поиск
O(n)
O(log n)
O(n)
O(log n)
O(log n)
FORM
включение
O(n)
O(n)
O(n)
O(n)
O(log n)
KILL
удаление
O(n)
O(n)
O(n)
O(n)
O(log n)

ФИНИШ.