Соловьев С.Ю.

soloviev@glossary.ru

# Алгоритмы и Алгоритмические языки

www.park.glossary.ru/pascal/

*Лекция No. 14*

2022

# Напоминание

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

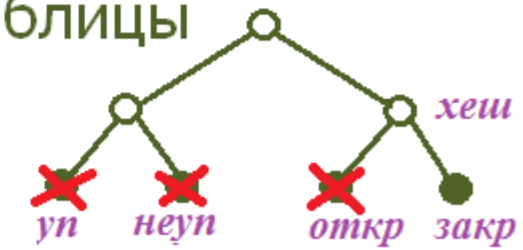
## Структуры данных

строительный материал

- массивы
- ссылки + записи
  - Линейные списки
  - Деревья, графы

виды

- очереди
- стеки
- таблицы




- деревья поиска

# *Напоминание* Хеш–таблицы

Хеш-таблица – таблица, в которой доступ к записям определяется хеш-функцией.

Хеш-функция (Функция расстановки) – функция, которая вычисляет **п**о заданному ключу

**н**омер позиции в таблице,

начиная с которого следует искать/размещать запись.

$$\mathbf{hash : \{\ Ключи\ \} \rightarrow \{\ 0,1,\ \ldots,\ So\text{-}1\ \}}$$

Коллизия – наличие в хеш-таблице ключей $K_1 \neq K_2$: $hash(K_1) = hash(K_2)$.

хеширование

открытое

закрытое

# Закрытое хеширование

Хеш-таблица – таблица, в которой доступ к записям определяется хеш-функцией.

*закрытое хеширование*

Хеш-функция с параметром = номером попытки разрешения коллизии.

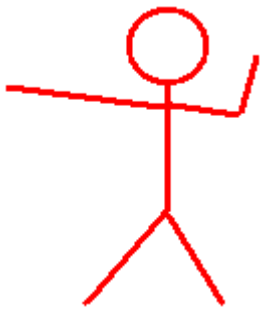
$$hast(K,V) = \begin{cases} hash(K), & \text{если } V = 0 \\ hash2(K,V), & \text{если } V > 0 \end{cases}$$

Пример 1.
```
function hash2(K,V : integer) : integer;
begin    hash2:=(hash(K)+V) mod So    end;
```

Пример 2.
```
function hash2(K,V : integer) : integer;
begin    hash2:=(hash(K)+sqr(V)) mod So    end;
```

# Пусть

```
program … ;
const  So = 1024;
       Sx = 1023;            { Sx = So-1 }
       …
type     Elem = record Key : integer; …  end;
        pElem = ^Elem;
       …
var     T : array [0..Sx] of pElem;
       function hash  (K    : integer) : integer;  begin . . . end;
       function hash2(K,V : integer) : integer;  begin . . . end;
       function HAST (K,V : integer) : integer;
       begin   if V = 0 then HAST:=hash(K) else HAST:=hash2(K,V) end;
       (*   FindHTL      FormHTL      KillHTL   *)
begin   { породить пустую таблицу }
       for I:=0 to Sx do T[I]:=nil;
       …
end.
```

# Соглашение

Запись с ключом K находится в позициях

HAST(K,0), HAST(K,1), HAST(K,2), . . . , HAST(K,L)

L? ➢ либо T[HAST(K,L+1)] = nil, если L < Sx,

либо L = Sx

---

Пусть var V,N : integer;

Перечислить все позиции:

```
for V:=0 to Sx do begin
    N:=HAST(K,V);
    if T[N] = nil then Exit;
    writeln(N)
end;
```

# Закрытое хеширование :: Поиск в хеш-таблице записи с заданным ключом

$$FindHTL(K) = \begin{cases} pElem & \textit{с ключом } K \\ nil & \textit{иначе} \end{cases}$$

```
function FindHTL(K : integer) : pElem;
    var  V,N : integer;
begin  FindHTL:=nil;
       for V:=0 to Sx do begin
           N:=HAST(K,V);
           if T[N]          = nil then Exit;
           if T[N]^.Key = K   then begin
                FindHTL:=T[N];                { Успех }
                Exit
           end
end    end;
```

# Закрытое хеширование:: Включить в хеш-таблицу запись с заданным ключом

FormHTL(K) = { позиция в T, если T[позиция] = nil, то K – новый ключ
если T[позиция] ≠ nil, то K есть в T
-1, если в T нет места

```
function FormHTL(K : integer) : integer;
   var V,N : integer;
begin for V:=0 to Sx do begin
         N:=HAST(K,V);
         if  T[N]       = nil then begin FormHTL:=N; Exit end;
         if  T[N]^.Key = K then begin FormHTL:=N; Exit end
      end;
      FormHTL:=-1
end;
```

# Закрытое хеширование:: Удалить в хеш-таблице запись с заданным ключом

```
procedure KillHTL(K : integer);
   var F,L : integer;

begin  FINDxLAST;
       if  F < 0 then Exit;
       if  L < 0 then Exit;
       dispose(T[F]);
       T[F]:=T[L];
       T[L]:=nil
end;
```

```
procedure FINDxLAST;
   var  V,N : integer;
begin   F:=-1;                  { позиция K }
        L:=-1;    { позиция последнего }
        for V:=0 to Sx do begin
            N:=HAST(K,V);
            if  T[N]        = nil  then Exit;
            if  T[N]^.Key = K   then F:=N;
                                     L:=N
        end
end;
```

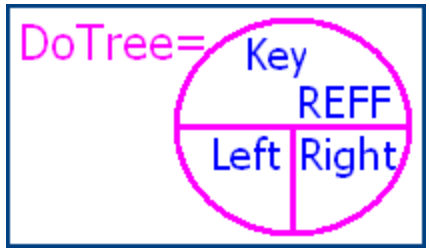
# Напоминание Таблицы

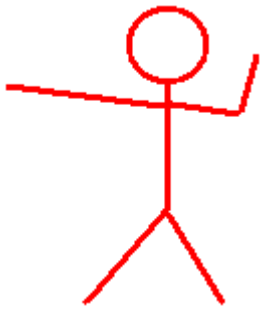
| | | неупорядоченные | упорядоченные |
|---|---|---|---|
| FIND | поиск | **O(n)** | **O(log n)** |
| FORM | включение | **O(n)** | **O(n)** |
| KILL | удаление | **O(n)** | **O(n)** |

# Бинарные деревья

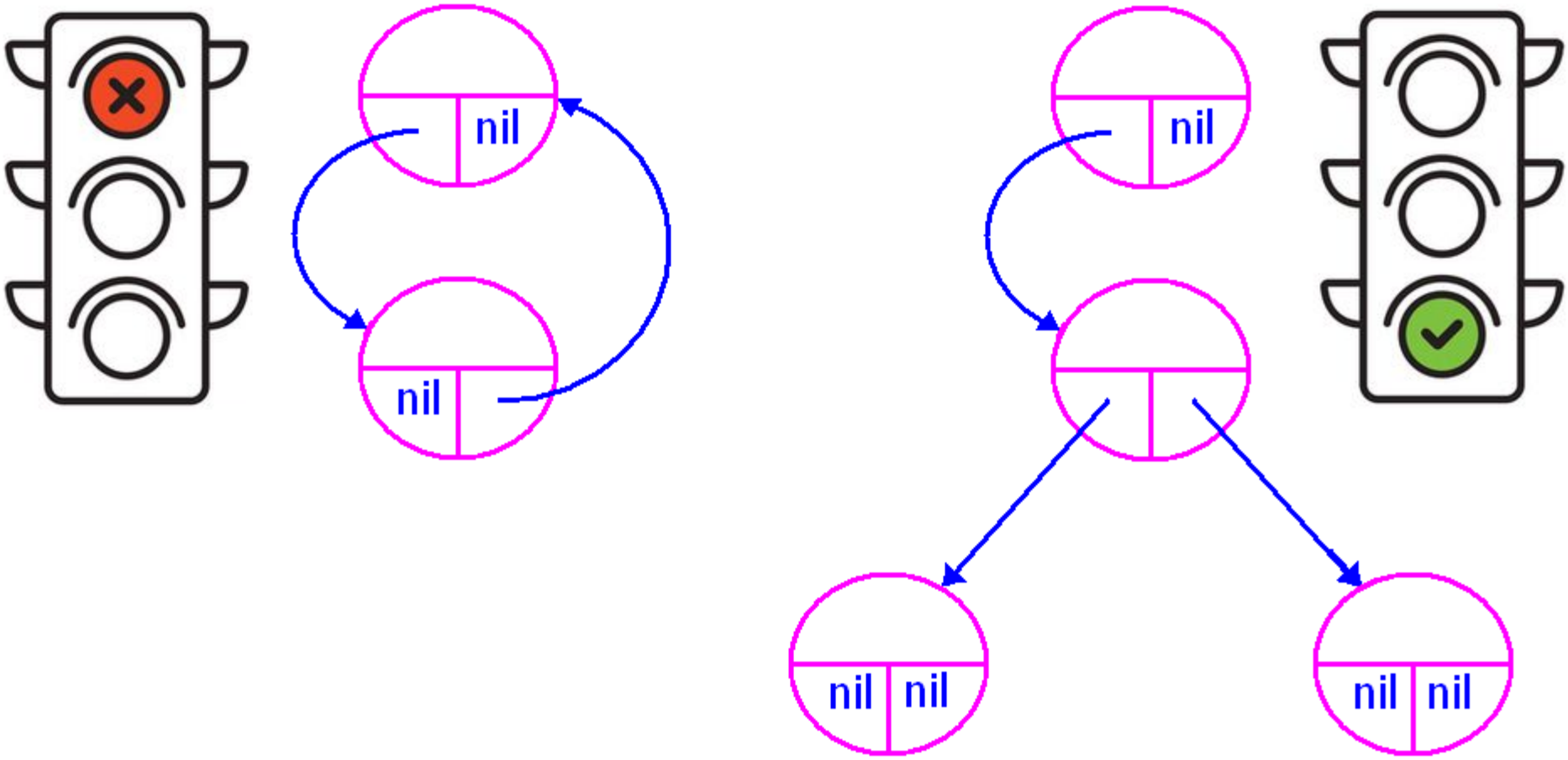
```
program … ;
type Info = record . . . end;
        pRecord = ^Info;
          DoTree = record      Key : integer;
                          Left,Right : pDoTree;
                               REFF : pRecord
                     end;
        pDoTree = ^DoTree;
          …
var      T : pDoTree;
        …
        (* ------  procedure | function  ------ *)

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



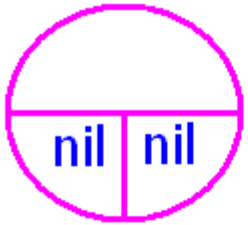



# Построения

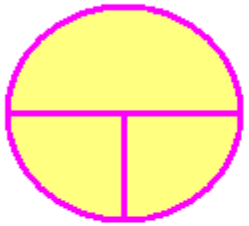


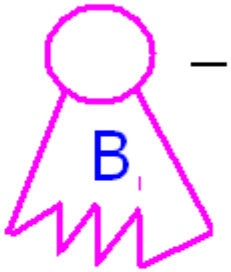

*Бинарное дерево с корнем*

# *Определение*
# Бинарное дерево с корнем

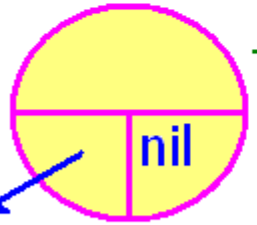
1. Вершина вида 

есть бинарное дерево с корнем.

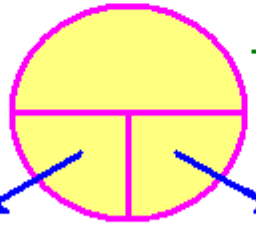
2. Пусть  – вершина, 

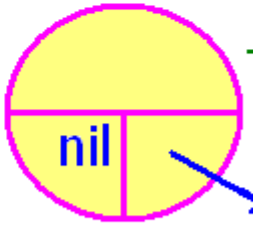
– бинарное дерево с корнем, 

– другое бинарное дерево с корнем.

Тогда 

– бинарное дерево с корнем, 



– бинарное дерево с корнем, 

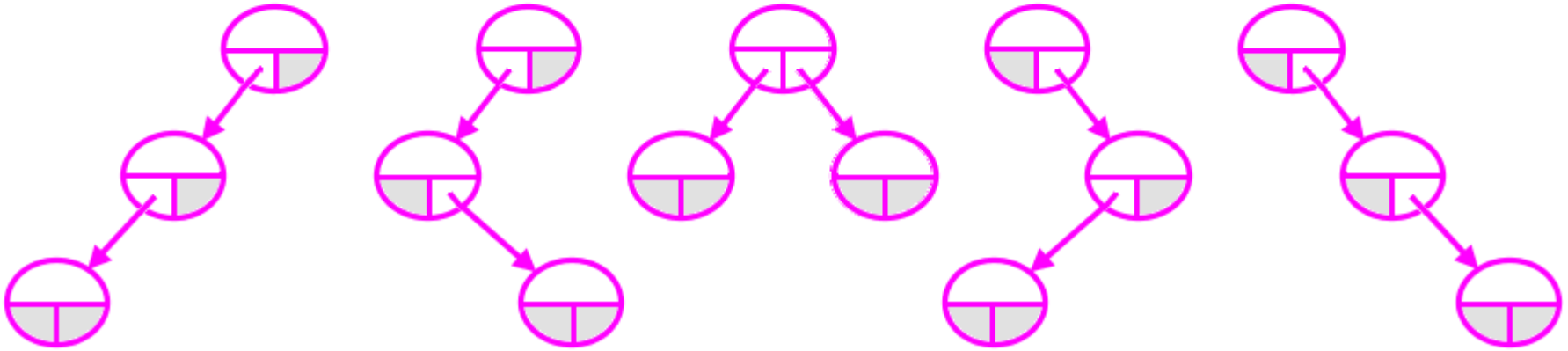



– бинарное дерево с корнем. 

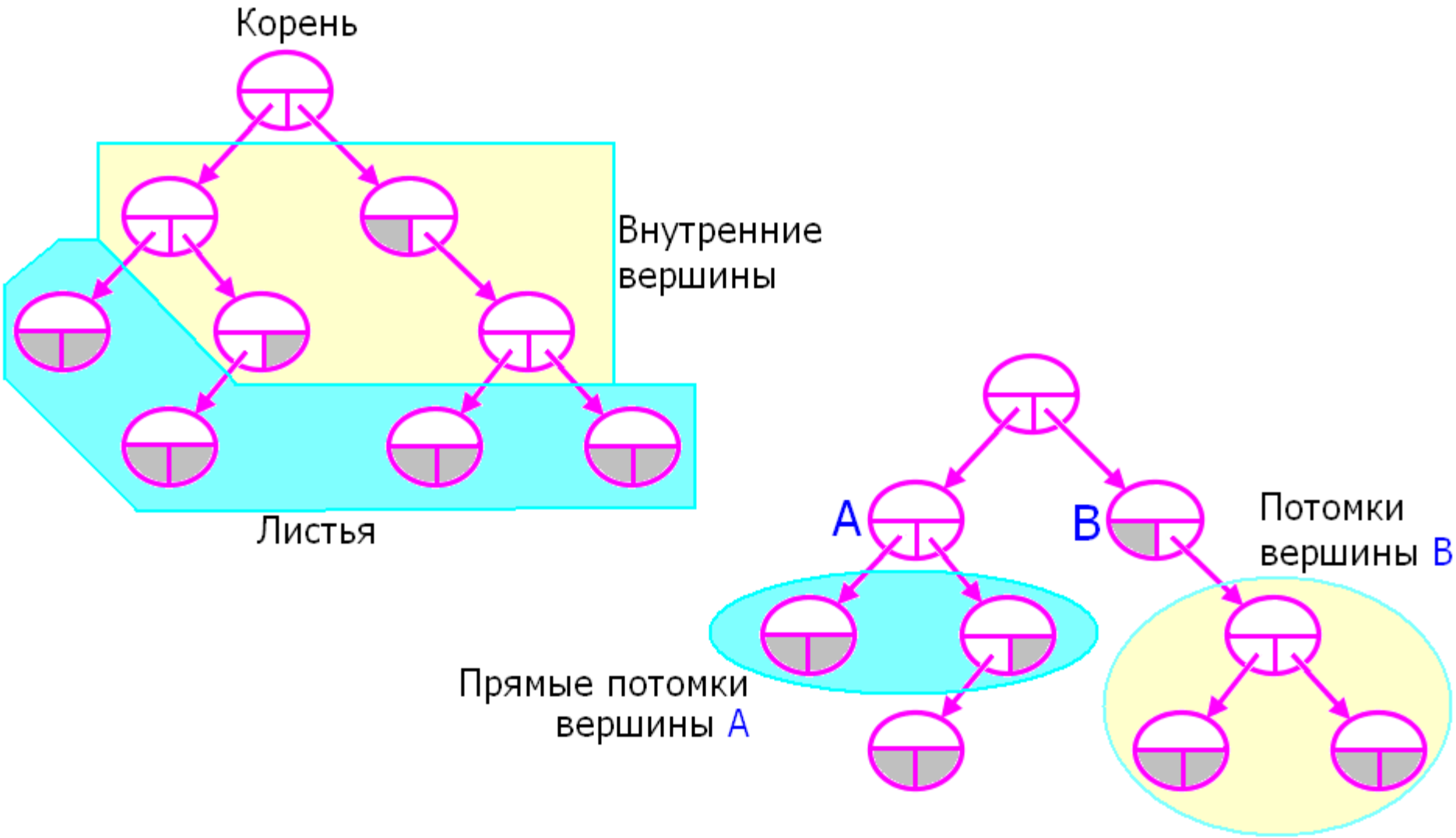
# Примеры бинарных деревьев

Бинарные деревья, содержащие 2 вершины. 2 шт.

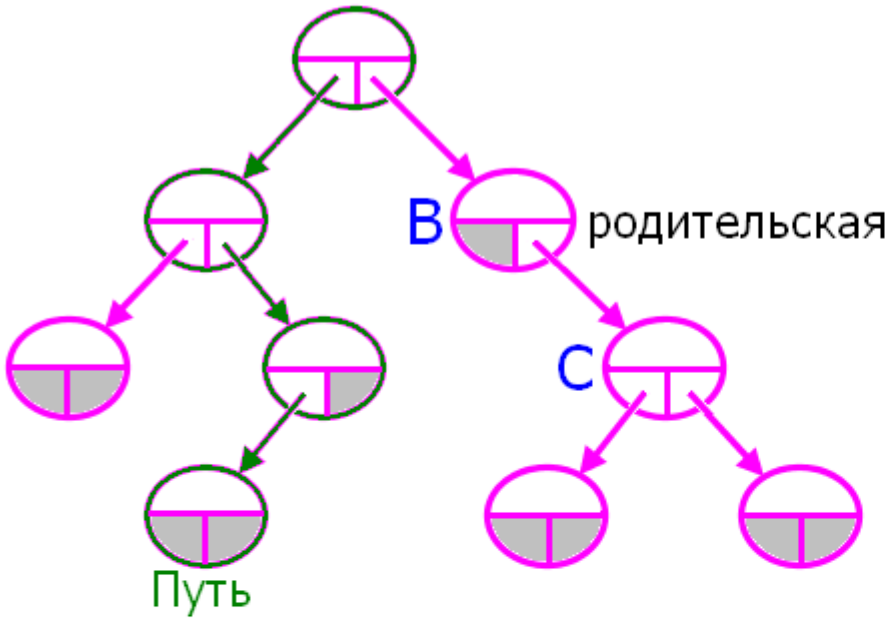
Бинарные деревья, содержащие 3 вершины. 5 шт.

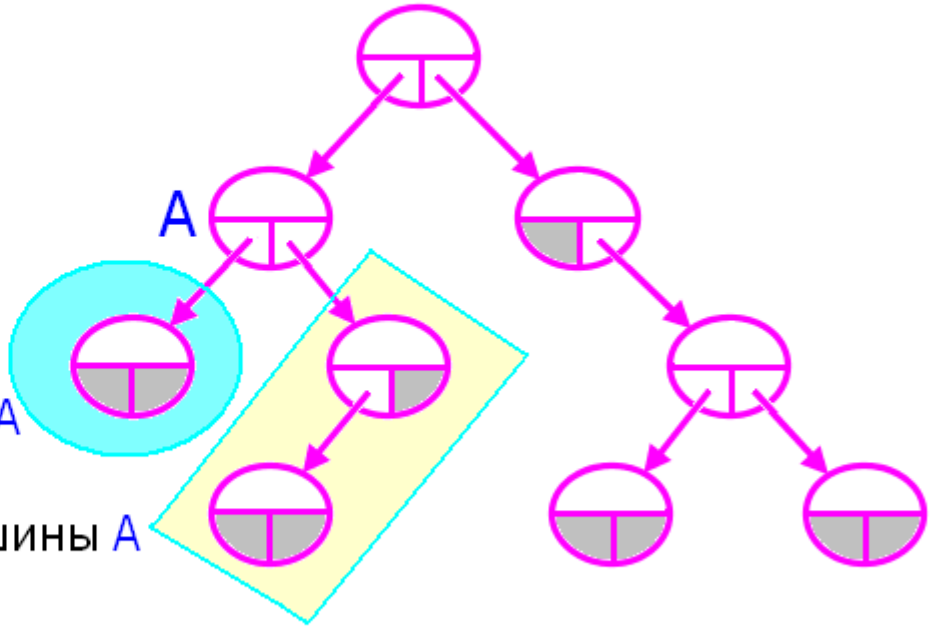


# Терминология (бинарных) деревьев/1



# Терминология (бинарных) деревьев/2




Левое поддерево вершины A

Правое поддерево вершины A

*Важно! Поддерево есть либо бинарное дерево с корнем либо nil .*

LK(A) – мн-во ключей из левого поддерева

RK(A) – мн-во ключей из правого поддерева

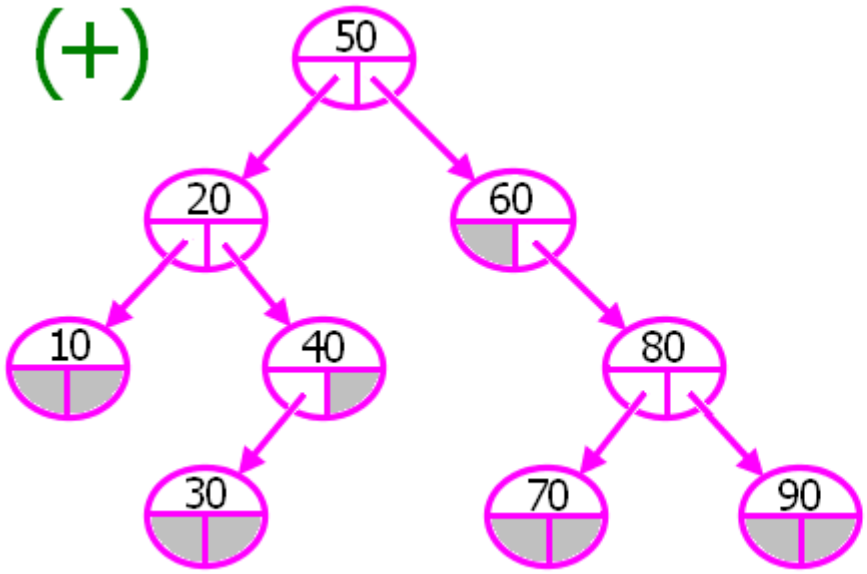
$K_A$ – ключ вершины A

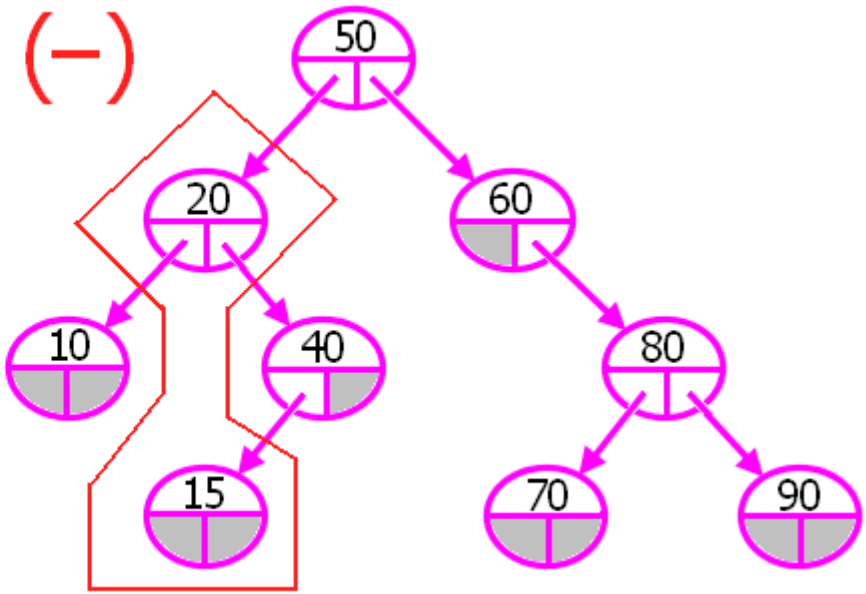
# Деревья поиска

Бинарное дерево с корнем называется деревом поиска, если для любой вершины A выполняются неравенства

$K_A < \mathit{min}\ RK(A)$ при $RK(A) \neq \varnothing$,

$\mathit{max}\ LK(A) < K_A$ при $LK(A) \neq \varnothing$.



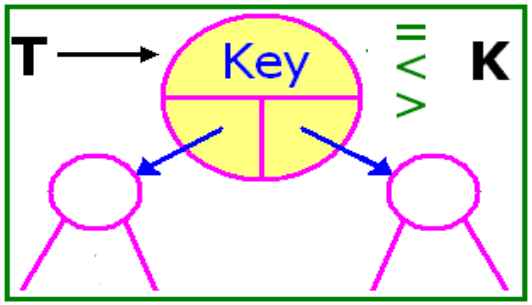



# Поиск по ключу в дереве поиска

Дано: T – указатель на ДП;  K – ключ.

$$FindSTr(T,K) = \begin{cases} pDoTree & \textit{с ключом K} \\ nil & \textit{иначе} \end{cases}$$

```
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;
```



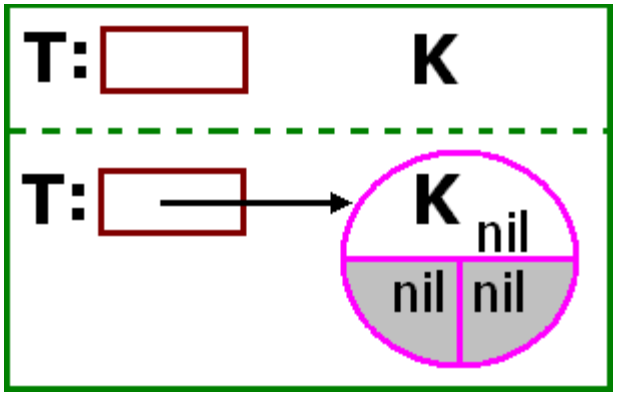
# Включить в дерево поиска новую вершину

Дано: T – указатель на ДП;  K – ключ

FormSTr(T,K) = pDoTree *с ключом K,*

*если при этом* REFF= nil, *то вершина новая.*

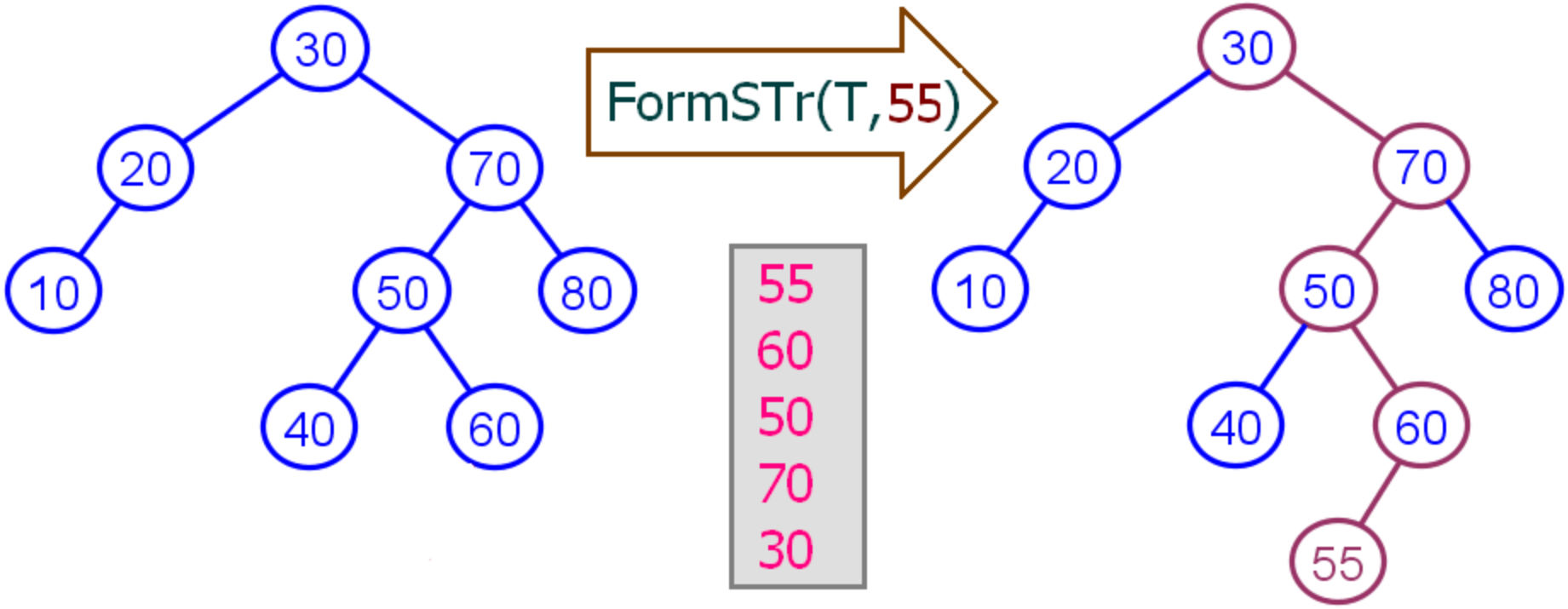
```
procedure Born(var T : pDoTree; K : integer);
begin    new(T);    T^.Key   :=K;
                    T^.Left  :=nil;
                    T^.Right:=nil;
                    T^.REFF:=nil
end;
```




```
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;
```

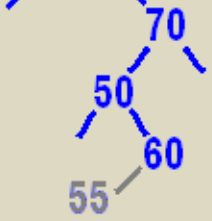
# Замечание

```
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;
```



# Объяснение

FormSTr(→30,55);




if false . . .
FormSTr(→70,55);

writeln(30);

if false . . .
FormSTr(→50,55);

writeln(70);
writeln(30);

if false . . .
FormSTr(→60,55);

writeln(50);
writeln(70);
writeln(30);

if false . . .
FormSTr(→nil,55);

writeln(60);
writeln(50);
writeln(70);
writeln(30);

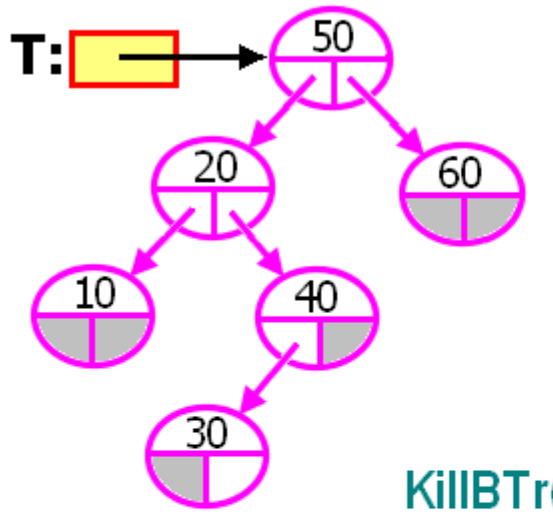
Born( … ,55);
FormSTr:= . . .

writeln(55);
writeln(60);
writeln(50);
writeln(70);
writeln(30);

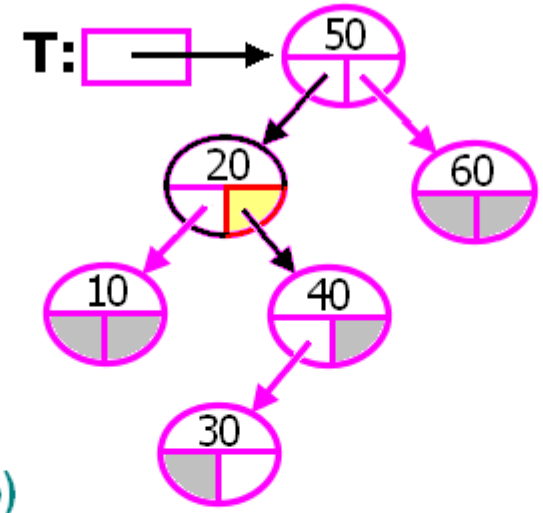
# Удалить вершину в дереве поиска

Дано: T – указатель на ДП;  K – ключ

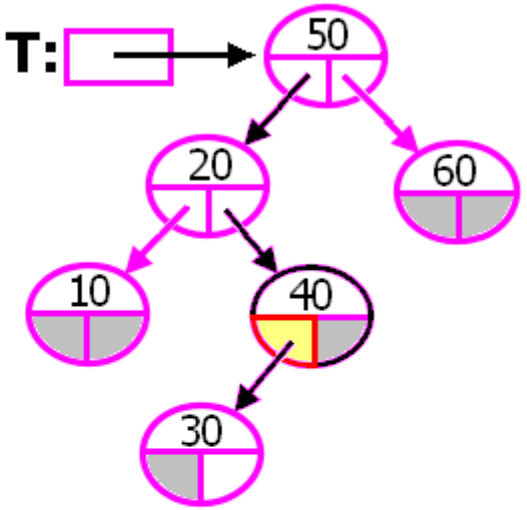
KillBTr(T,50)




KillBTr(T,40)




KillBTr(T,30)




KillBTr(T,35)

```
procedure KPECT(var R : pDoTree); begin   end;
```

R *есть* *либо* *либо*

```
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;
```

# Устранение существующей вершины в ДП / 1

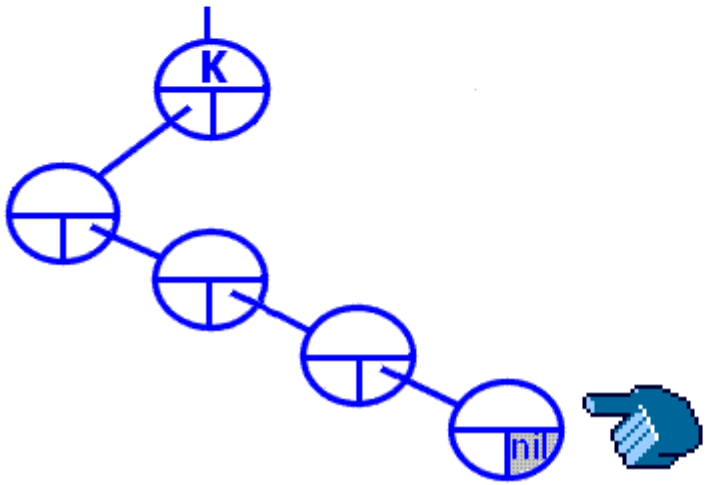
procedure **КРЕСТ**(**var** R : pDoTree)

(1) Ликвидировать - - - - - - - - - - - - - **dispose**

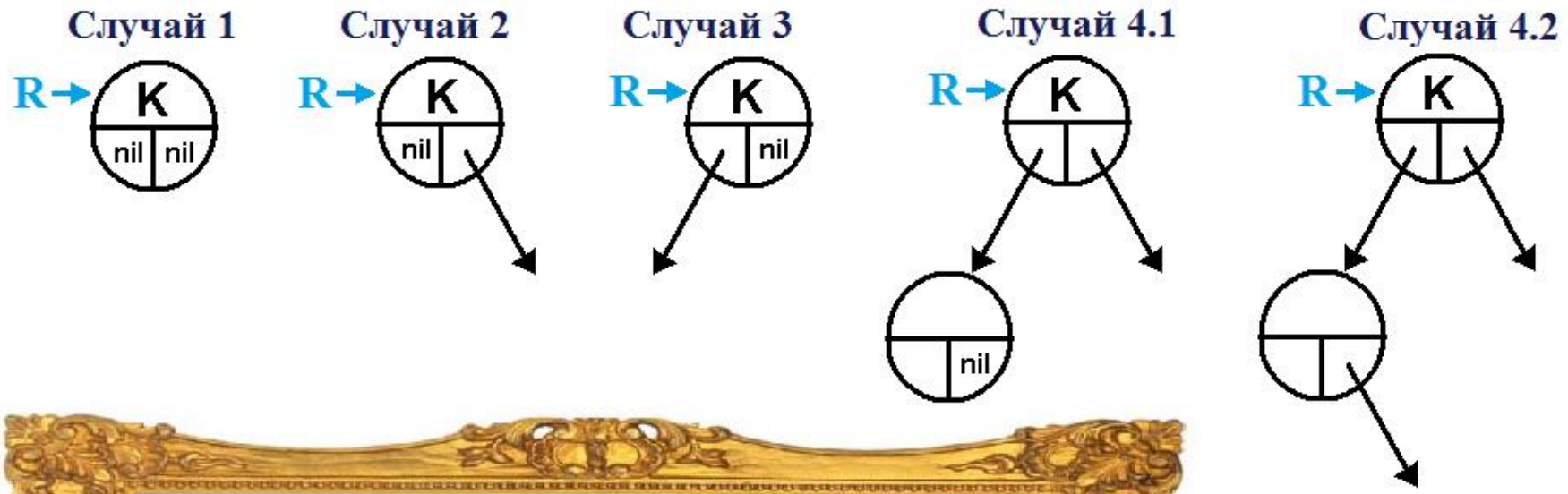
(2) Заменить на

наибольший в левом поддереве

наименьший в правом поддереве



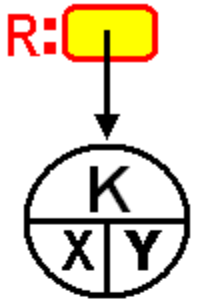
# Устранение существующей вершины в ДП / 2



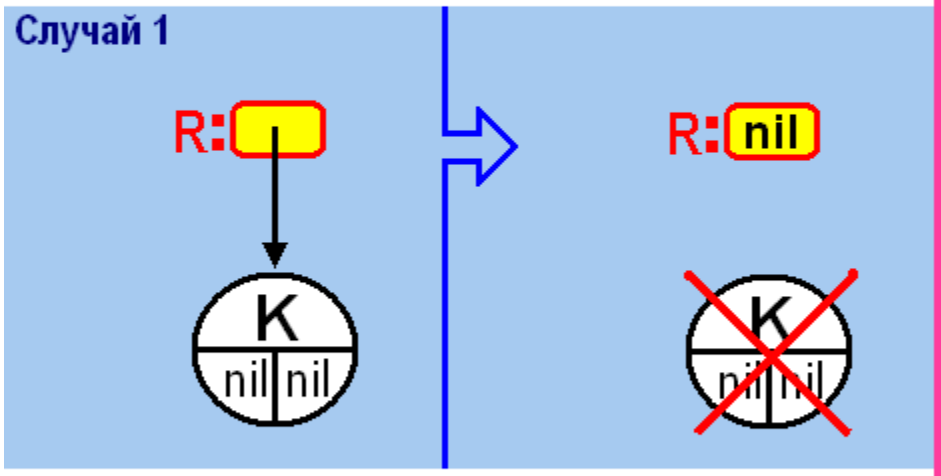

```
with R do
      if (Left = nil) and (Right = nil) then { 1   }
else if  Left = nil                     then { 2   }
else if                  Right = nil    then { 3   }
else if  Left^.Right = nil              then { 4.1 }
else                                         { 4.2 }
```

# Устранение существующей вершины в ДП / 3

procedure **KPECT**(**var** R : pDoTree)

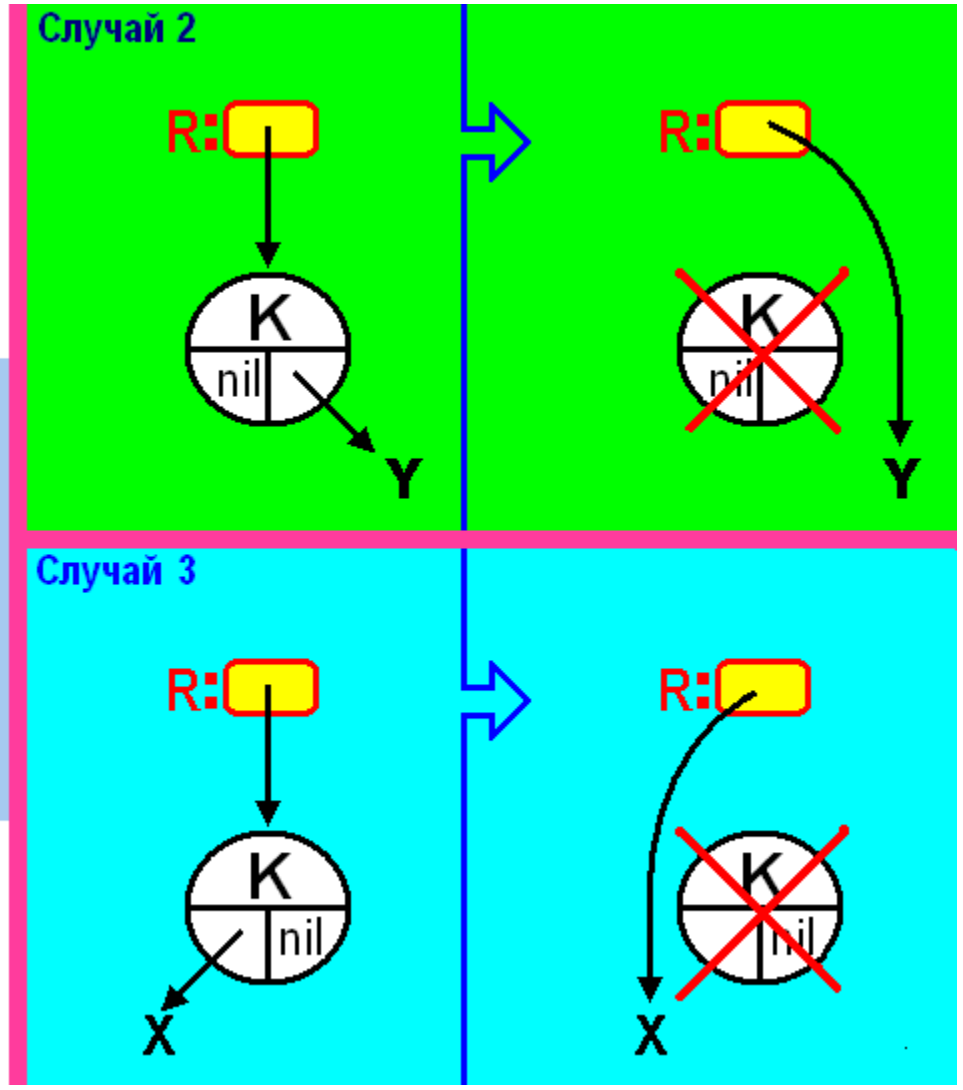



X:=R^.Left;
Y:=R^.Right;

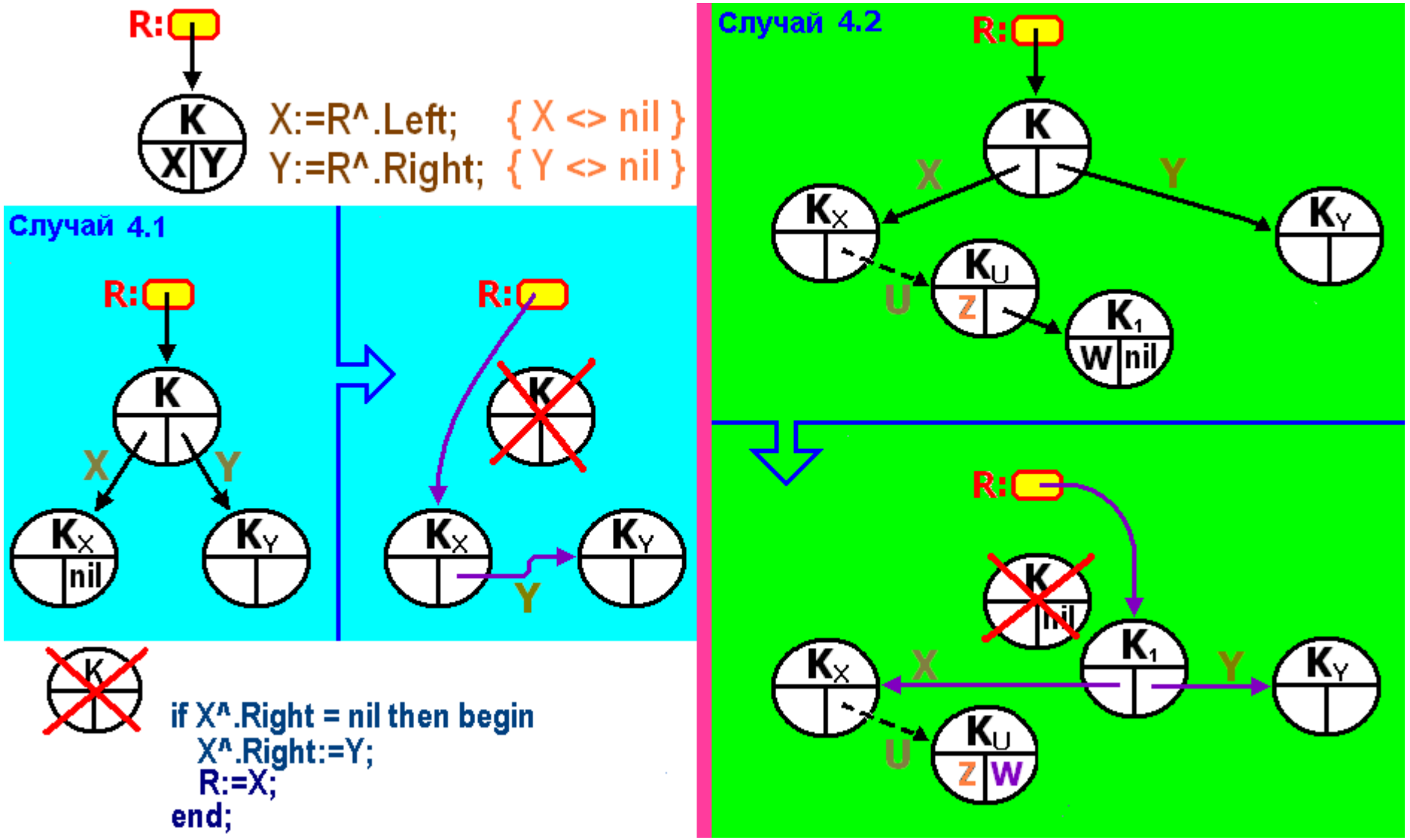




if (X = nil) and (Y = nil) then R:=nil;



# Устранение существующей вершины в ДП / 4



# Устранение существующей вершины в ДП / 5

```
procedure КРЕСТ(var R : pDoTree);
    var X,Y,U,W : pDoTree;
begin   X:=R^.Left;
        Y:=R^.Right;
        if R^.REFF <> nil then  dispose(R^.REFF);
                                dispose(R);

{  1 }          if (X = nil) and (Y = nil) then         R:=nil
{  2 }  else    if  X = nil                then         R:=Y
{  3 }  else    if                 Y = nil  then         R:=X
{4.1}   else    if  X^.Right = nil          then begin R:=X; R^.Right:=Y end
{4.2}   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
end     end;
```

# FindSTr ✦ FormSTr ✦ KillSTr

```
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


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

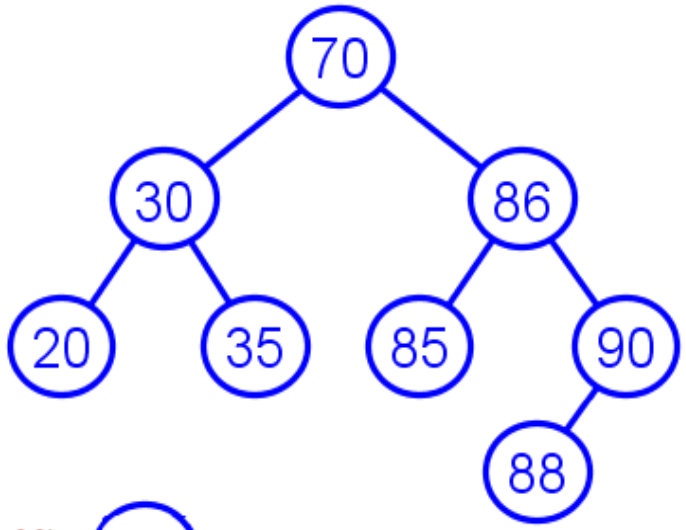
*Количество сравнений* ⩽ *Высота дерева* — есть — ***max*** количество вершин на пути от корня к листу

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

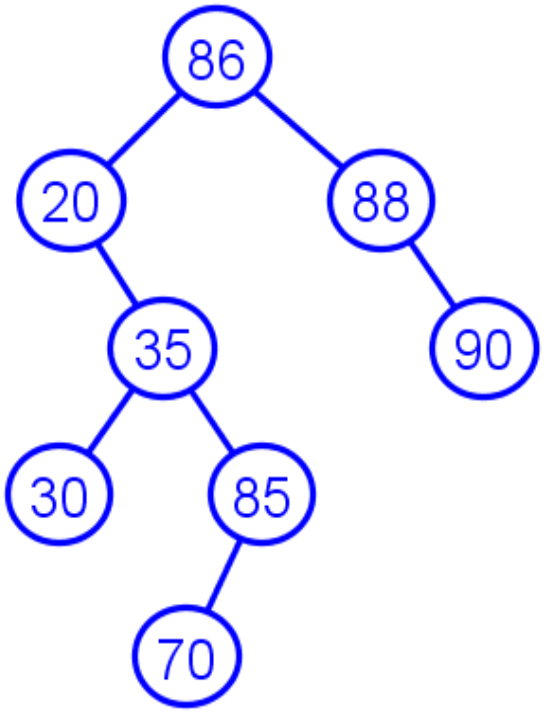
**Если** $H \leqslant \varphi(n)$, где n – к-во вершин в T, **то сложность есть** $O(\varphi(n))$.

# Примеры

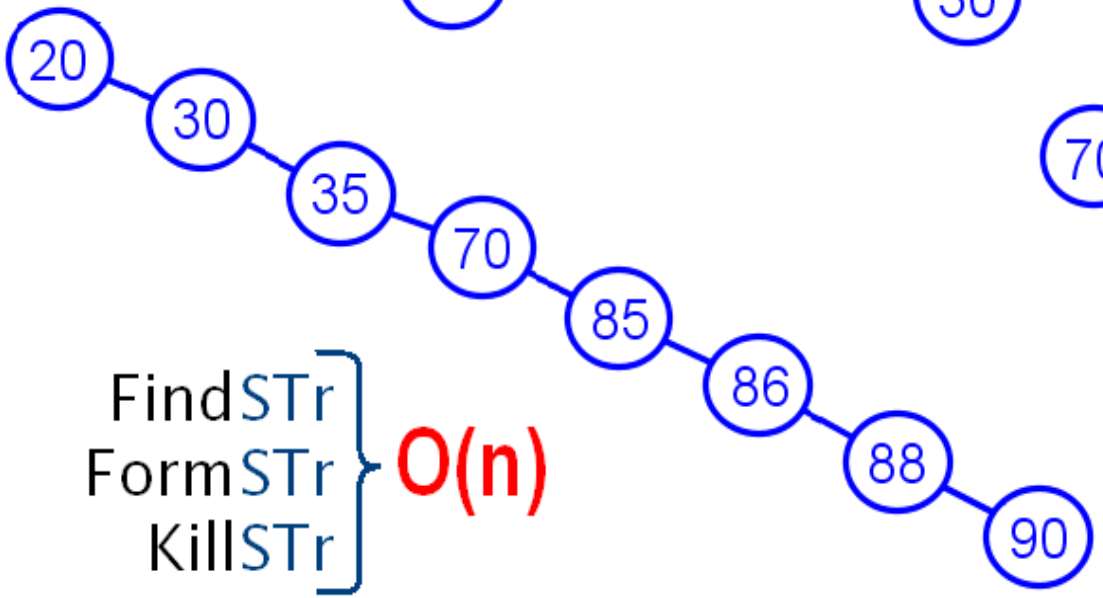
FormSTr(T,70);
FormSTr(T,86);
FormSTr(T,30);
FormSTr(T,35);
FormSTr(T,20);
FormSTr(T,90);
FormSTr(T,85);
FormSTr(T,88);




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)

## Обзор лекции No.14

Закрытое хеширование

Операции поиска, вставки и удаления элементов при закрытом хешировании

Бинарные деревья с корнем

Терминология бинарных деревьев

Деревья поиска

Операции поиска, вставки вершин в деревьях поиска

Операция удаления вершины в дереве поиска

Высота дерева

Оценки  сложности операций поиска, вставки и удаления вершин в дереве поиска



***--- Конец лекции No. 14 ---***