Přednáška IAL - 8

Přehled témat přednášky

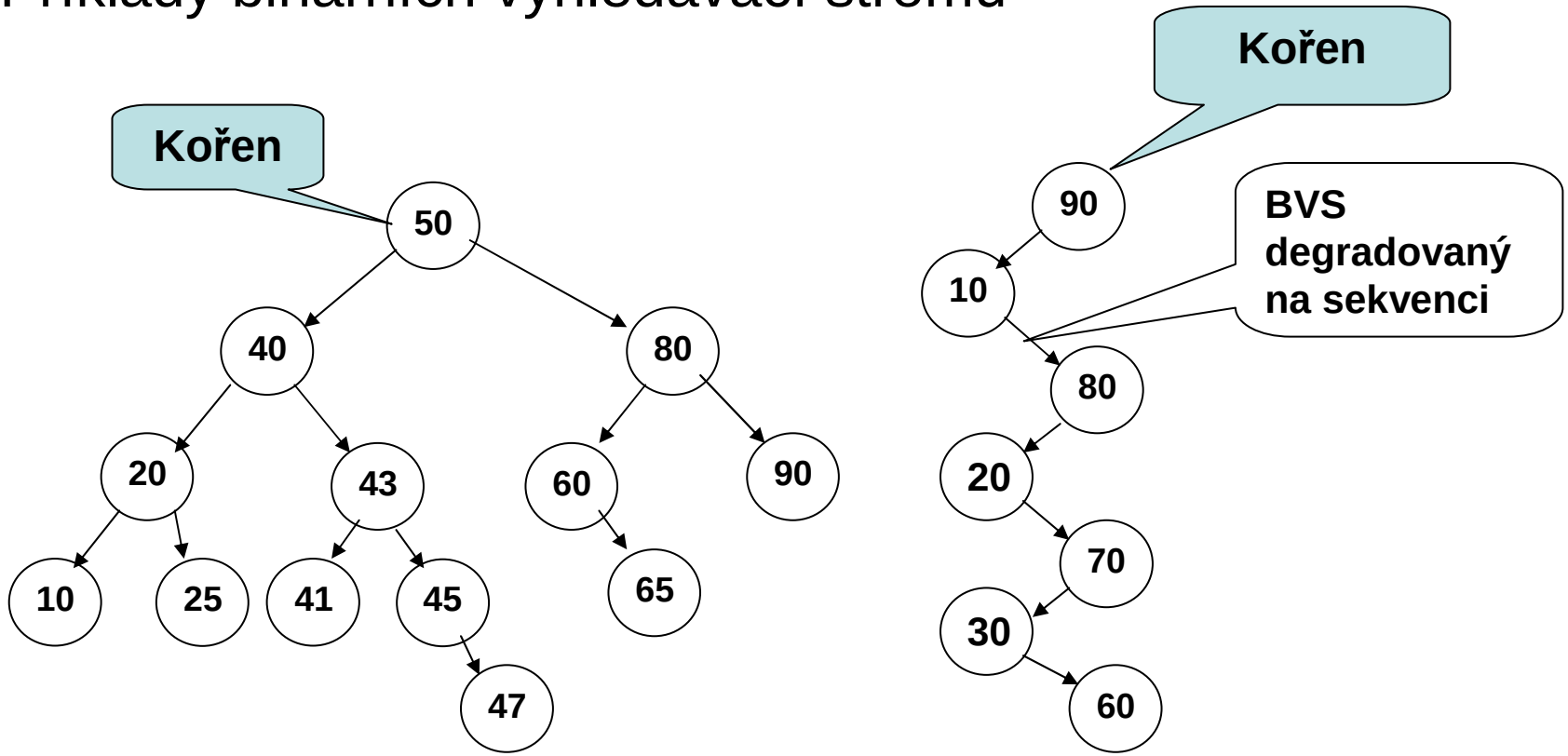
- Binární vyhledávací stromy (BVS)
- Rekurzivní i nerekurzívní verze operací nad BVS
- BVS se zarážkou
- BVS se zpětnými ukazateli
- AVL stromy

Definice BVS

Uspořádaný strom je kořenový strom, pro jehož každý uzel platí, že n-tice kořenů podstromů uzlu je uspořádaná.

Binární vyhledávací strom je takový binární uspořádaný strom, pro jehož každý uzel platí, že levý podstrom tohoto uzlu je buď prázdný, nebo obsahuje uzly, jejichž hodnoty jsou menší, než hodnota tohoto uzlu a stejně tak pravý podstrom tohoto uzlu je buď prázdný nebo obsahuje uzly, jejichž hodnota je větší, než hodnota tohoto uzlu.

Příklady binárních vyhledávací stromů



Průchod InOrder BVS stromem dává seřazenou posloupnost:
1 strom: 10,20,25,40,41,43,45,47,50,60,65,80,90
2 strom: 10,20,30,60,70,80,90

Vyhledávání v binárním stromu

- Vyhledávání v BVS je podobné binárnímu vyhledávání v seřazeném poli.
- Je-li vyhledávaný klíč roven kořeni, vyhledávání končí úspěšným vyhledáním. Je-li klíč menší, pokračuje vyhledávání v levém podstromu, je-li větší, pokračuje v pravém podstromu. Vyhledávání končí neúspěšně, pokud je prohledávaný (pod)strom prázdný.

Datové typy používané pro BVS jsou stejné jako pro dvojsměrný seznam resp. binární strom:

```
type
  TUk=^TUzel;

  TUzel=record
    Klic:TKlic;
    Data:TData;
    LUk, PUk:TUk
  end;
```

Rekurzívní verze vyhledávání v BVS

```
function Search(UkKor:TUk; K:TKlic):Boolean;
(* UkKor je ukazatel kořene BVS *)
begin
  if UkKor<>nil
  then
    if UkKor^.Klic=K
    then
      Search:=true
    else
      if UkKor^.Klic>K
      then (* hledání pokračuje v levém podstromu *)
        Search:=Search(UkKor^.LUk,K)
      else (* hledání pokračuje v pravém podstromu *)
        Search:=Search(UkKor^.PUk,K)
   else (* cesta končí na term. uzlu – neúspěšné hledání *)
      Search:=false
end;
```

Varianta vyhledávání v BVS jako procedura

```
procedure SearchTree(var UkKor:TUk; K:TKlic; var
Kde:TUk);
begin
  if UkKor = nil
  then
     Kde:=nil
  else
    if UkKor^.Klic <> K
    then
         if UkKor^.Klic > K
         then
             SearchTree(UkKor^.LUk, K, Kde)
         else
             SearchTree(UkKor^.PUk, K, Kde)
     else Kde:=UkKor   (* našel a nastavuje výstupní parametr Kde *)
end; (* procedure *)
```

Nerekurzívní zápis vyhledávání v BVS

```
function Search(UkKor:TUk; K:TKLic):Boolean;
var  Fin:Boolean (* Řídicí proměnná cyklu *)
begin
   Search:=false;
   Fin:=UkKor=nil;
   while not Fin do begin
     if UkKor^.Klic=K
     then begin
       Fin:=true;
       Search:=true;
     end else
       if UkKor^.Klic > K
       then UkKor:=UkKor^.LUk   (* Pokračuj v levém podstromu *)
       else
         UkKor:=UkKor^.PUk; (* Pokračuj v pravém podstromu *)
     if UkKor=nil
     then Fin:=true
  end (* while *)
end; (* procedure *)
```

K domácímu procvičení

Vytvořte následující nerekurzívní varianty zápisu procedury pro vyhledávání v BVS:

a) procedura vrátí Booleovský parametr Search a ukazatel UkKde pro nalezený uzel (v případě Search=false je UkKde nedefinováno)

```
procedure InsertSearch1(UkKor:TUk; K:TKlic;
          var Search:Boolean;  var UkKde:TUk);
```

b) procedura vrátí ukazatel UkKde na nalezený uzek a nil v případě neúspěšného vyhledávání.

```
procedure InsertSearch2(UkKor:TUk; K:TKlic;
     var UkKde:TUk);
```

Operace Insert

Operace Insert aplikuje „aktualizační sémantiku“, t.zn., že v případě, že uzel s daným klíčem existuje, přepíše stará data aktuálními. V případě, že uzel s daným klíčem neexistuje, vloží nový uzel jako terminální uzel tak, aby se zachovala pravidla BVS.

pomocná procedura pro vytvoření uzlu zkrátí zápis na stránce:

```
procedure VytvorUzel(var UkUzel:TUk; K:TKlic;
D:TData);
begin
   new(UkUzel);
   UkUzel^.Klic:=K;
   UkUzel^.Data:=D;
   UkUzel^.Luk:=nil;
   UkUzel^.Puk:=nil
end
```

Rekurzívní zápis operace insert

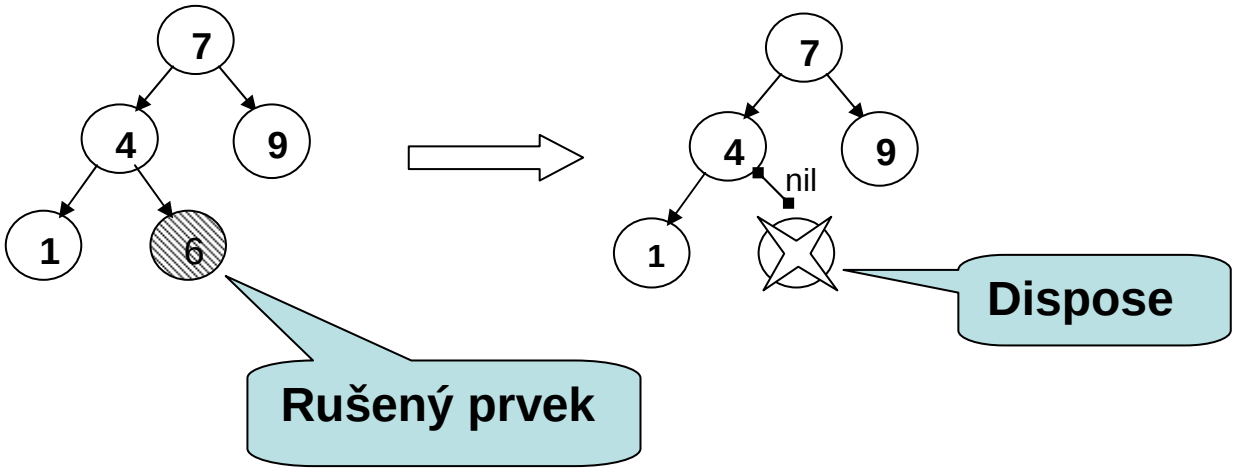
```
procedure Insert (var UkKor:TUk; K:TKlic; D:TData);
begin
  if UkKor=nil
  then
    VytvorUzel(UkKor, K, D) (* vytvoření kořene resp. term. uzlu *)
  else
    if K<UkKor^.Klic
    then
      Insert(UkKor^.LUk,K,D)(*pokračuj v levém podstromu  *)
    else
      if K>UkKor^.Klic
      then
        Insert(Ukkor^.PUk,K,D) (* pokračuj v pravo *)
      else UkKor^.Data:=D  (* přepiš stará data novými *)
end; (* procedure *)
```

```
procedure SearchIns(UkKor:TUk;K:TKlic;var Found:Boolean;
                    var Kde:TUk);
(* Vyhledání za účelem nerekurzívního vkládání *)
  var PomUk:TUk;
  begin
    if UkKor=nil
    then begin
      found:=false; Kde:=nil; (* prázdný BVS *)
    end else
      repeat
        Kde:=UkKor;    (* uchování výstupní hodnoty Kde *)
        if UkKor^.Klic > K
        then
          UkKor:=UkKor^.Luk (* posun doleva *)
        else
          if UkKor^.Klic < K
          then
            UkKor:=UkKor^.Puk   (* posun doprava *)
          else
            found:=true (* našel, vrací hodnotu Kde *)
      until found or (UkKor=nil)
  end; (* procedure *)
```

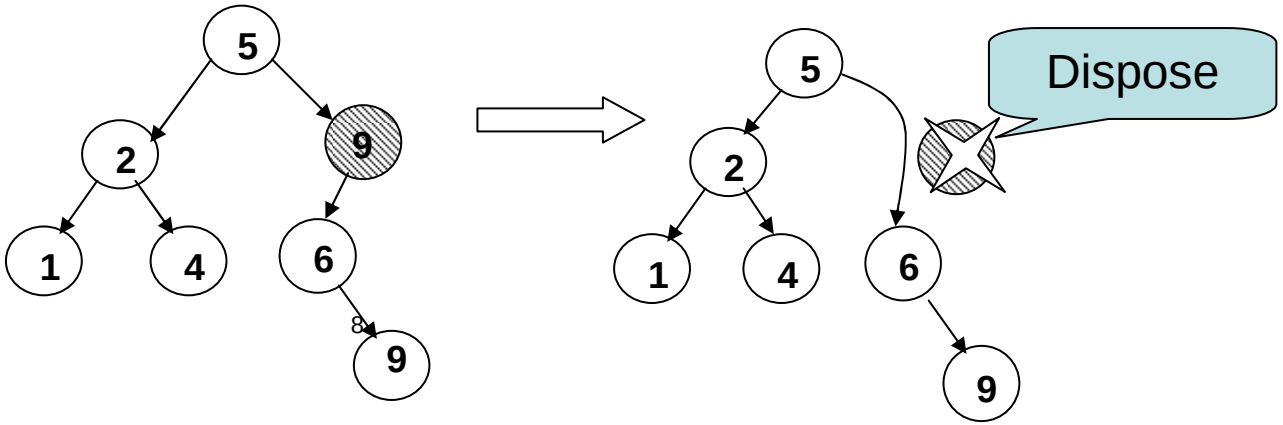
```
procedure Insert (var UkKor:TUk; K:TKlic; D:TData);
var
  PomUk, Kde:TUk;
  Found:Boolean;
begin
  SearchIns(UkKor,K,Found,Kde);
  if Found
  then
    Kde^.Data:=D; (* přepsání starých dat novými *)
  else
    VytvorUzel(PomUk,K,D);
    if Kde =nil
    then UkKor:=PomUk; (* prázdný strom, nový uzel bude kořen *)
    else (* neprázdný strom, nový uzel se připojí jako terminál *)
      if Kde^.Klic>K
      then (* nový uzel se připojí vlevo *)
        Kde^.LUk:=PomUk
      else (* uzel se připojí vpravo *)
           Kde^.PUk:=PomUk
end; (* procedure *)
```

Rušení uzlu – operace Delete

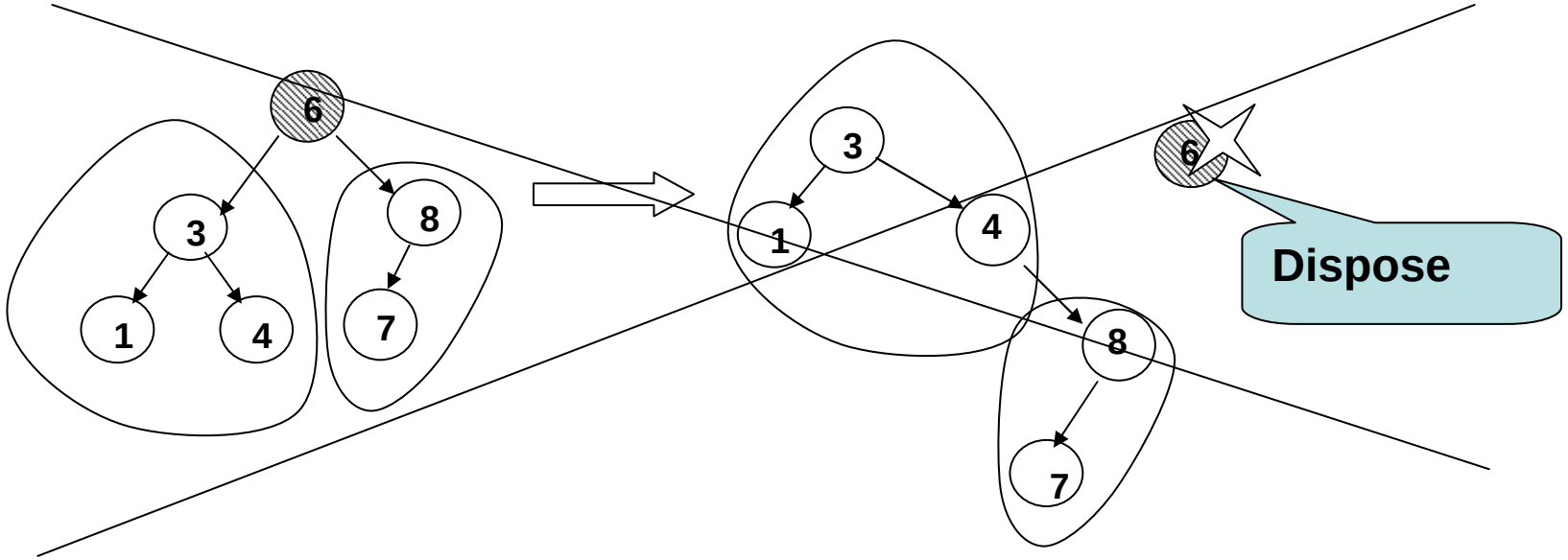
Rušení terminálního uzlu je snadné



Zrušení uzlu, který je jediným synovským uzlem je také snadné



Zrušit uzel, který má dva podstromy lze také tak, že levý podstrom připojíme na nejlevější uzel pravého podstromu nebo tak, že pravý podstrom rušeného uzlu připojíme na nepravější uzel levého podstromu, jak je to uvedeno na následujícím obrázku.

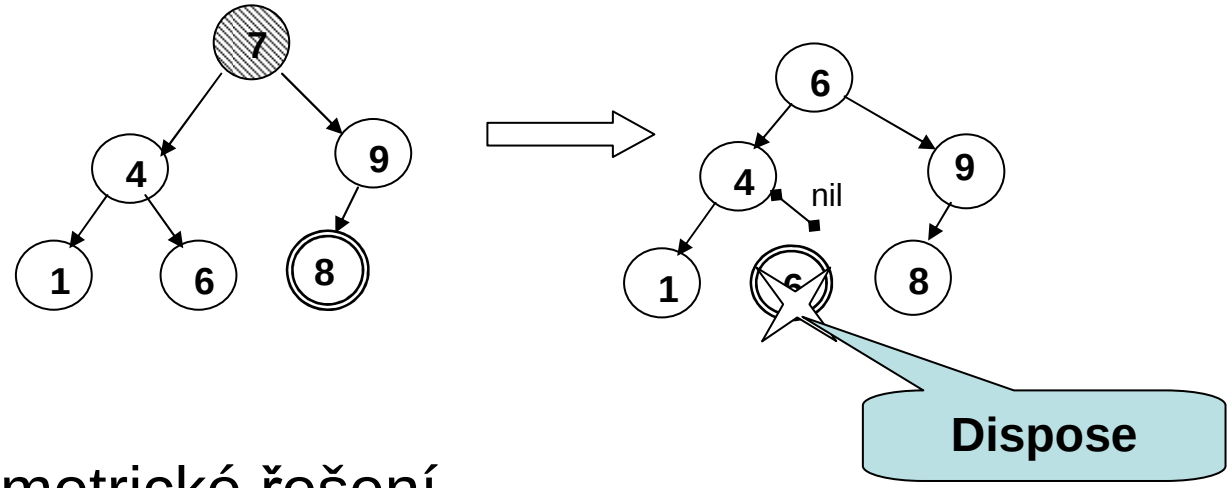


Toto řešení je však pro vyhledávací stromy nepřijatelné, protože zbytečně zvyšuje výšku stromu a tím i maximální dobu vyhledávání.

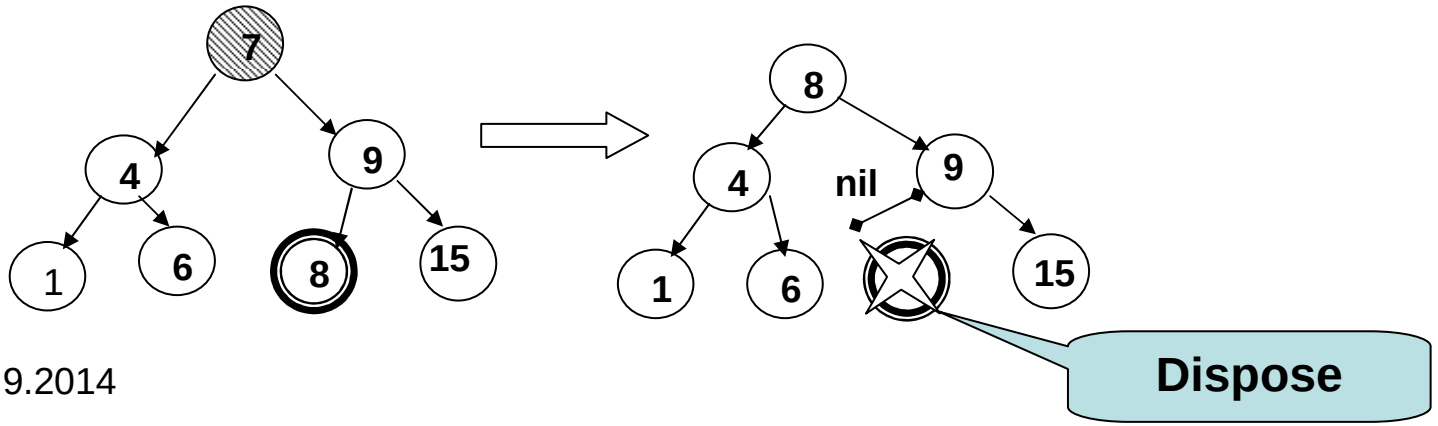
Rušení uzlu se dvěma syny v binárním stromu

Rušený uzel se dvěma syny se nezruší „fyzicky“, ale jeho hodnota se přepíše hodnotou takového uzlu, který lze zrušit snadno a přitom při přepisu nesmí dojít k porušení uspořádání (pravidel) BVS. Takovým uzlem je nejpravější uzel levého podstromu rušeného uzlu nebo symetricky nejlevější uzel pravého podstromu rušeného uzlu.

Příklad:



Symetrické řešení

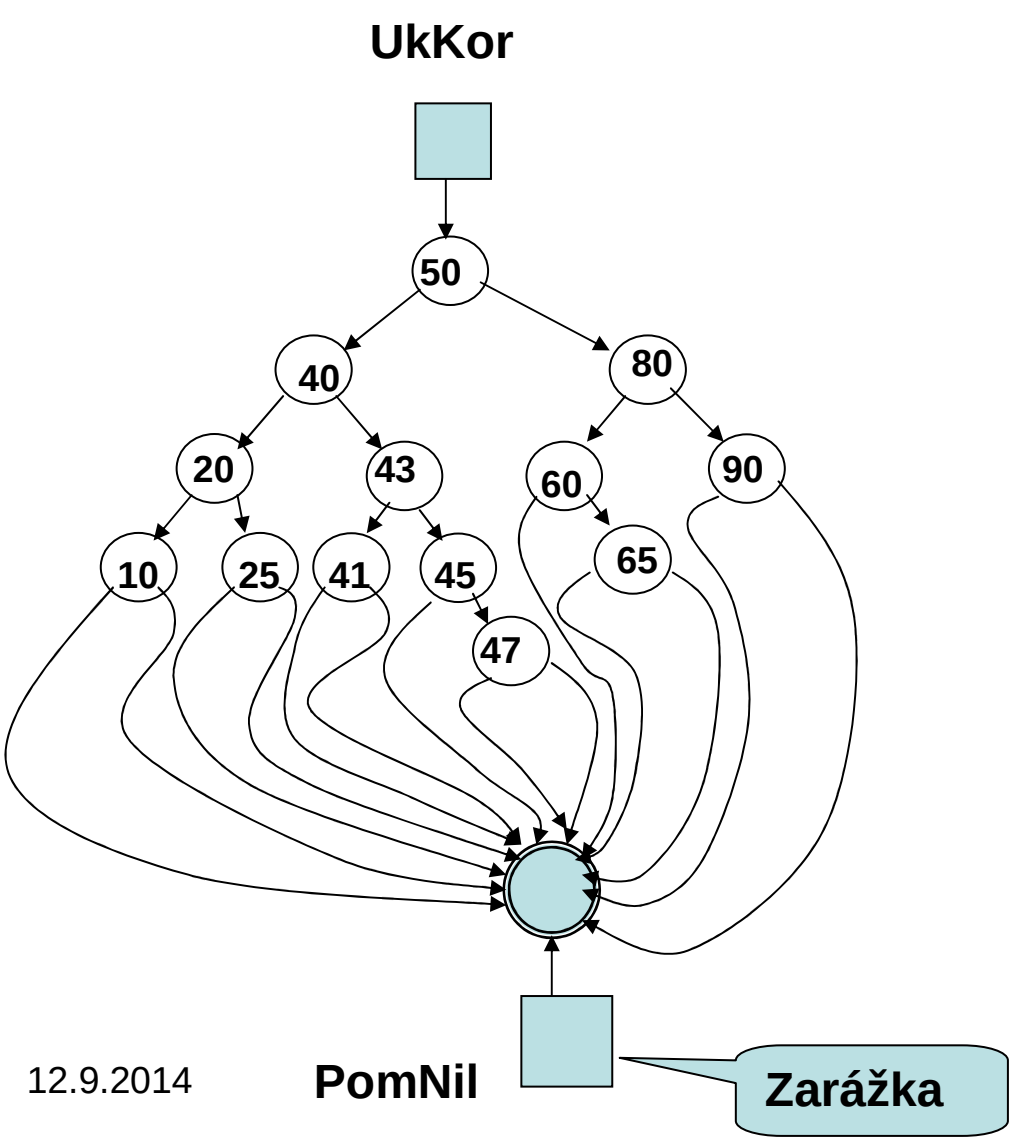


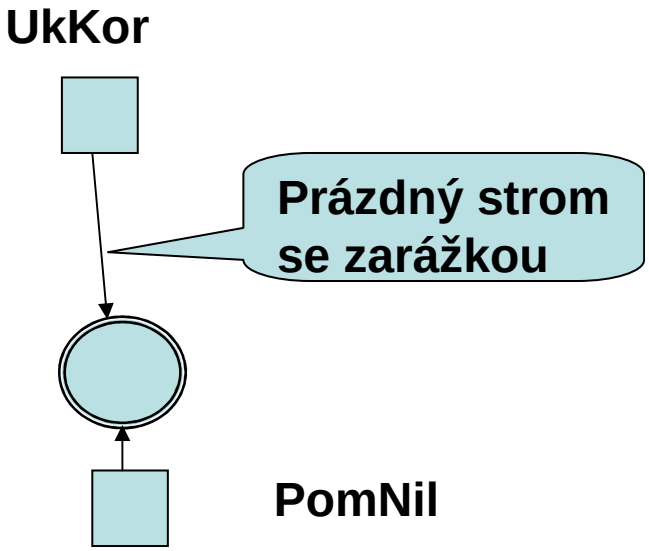
Rekurzívní verze procedury Delete je popsána v obou textech uvedených jako zdroje. Součástí procedury je rekurzívní procedura Del, která prochází po pravé diagonále levého podstromu a hledá prvek, jehož hodnotou se přepíše rušený uzel a který je následně rušen.

Algoritmus rekurzívní procedury je netriviální a u písemných zkoušek se neočekává jeho „rekonstrukce“. Student, který si myslí, že má předpoklady stát se inženýrem, mu však musí porozumět (bakalář nemusi...☺). U souhrnné zkoušky lze žádat vysvětlení předloženého algoritmu.

Binární vyhledávací strom se zarážkou *

Před vyhledáváním se do zarážky vloží klíč a nemusí se kontrolovat konec.





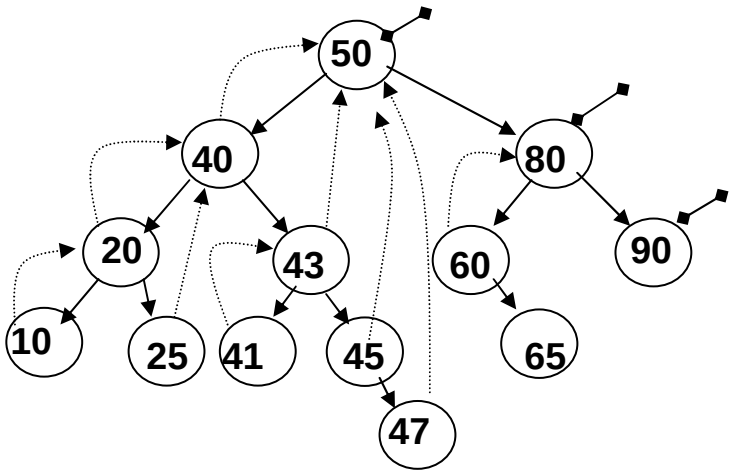
```
procedure TInit(var UkKor:TUk);
(* proměnná PomNil je globální *)
begin
  new (PomNil);
  UkKor:=PomNil
end;
function SearchTree(Uk:TUk; K:TKlic):Boolean;
(* proměnná PomNil je globální *)
begin
  PomNil^.Klic:=K;
  while Uk^.Klic <> K do begin
    if  Uk^.Klic>K
    then Uk:=Uk^.Luk
    else  Uk:= Uk^.Puk
  end; (* while *)
  SearchTree:= Uk<>PomNil
end; (* procedure *)
```

Binární vyhledávací strom se zpětnými ukazateli

Tento strom má význam pouze tehdy, chceme-li při průchodu InOrder vyhnout rekurzi nebo použití zásobníku.

Platí pravidla:

1. Zpětný ukazatel kořene ukazuje na nil (všechny uzly vedlejší diagonály ukazují na nil...)
2. Zpětný ukazatel levého syna ukazuje na svého otce
3. Zpětný ukazatel pravého syna dědí ukazatele od otce (ukazuje tam kam otec).



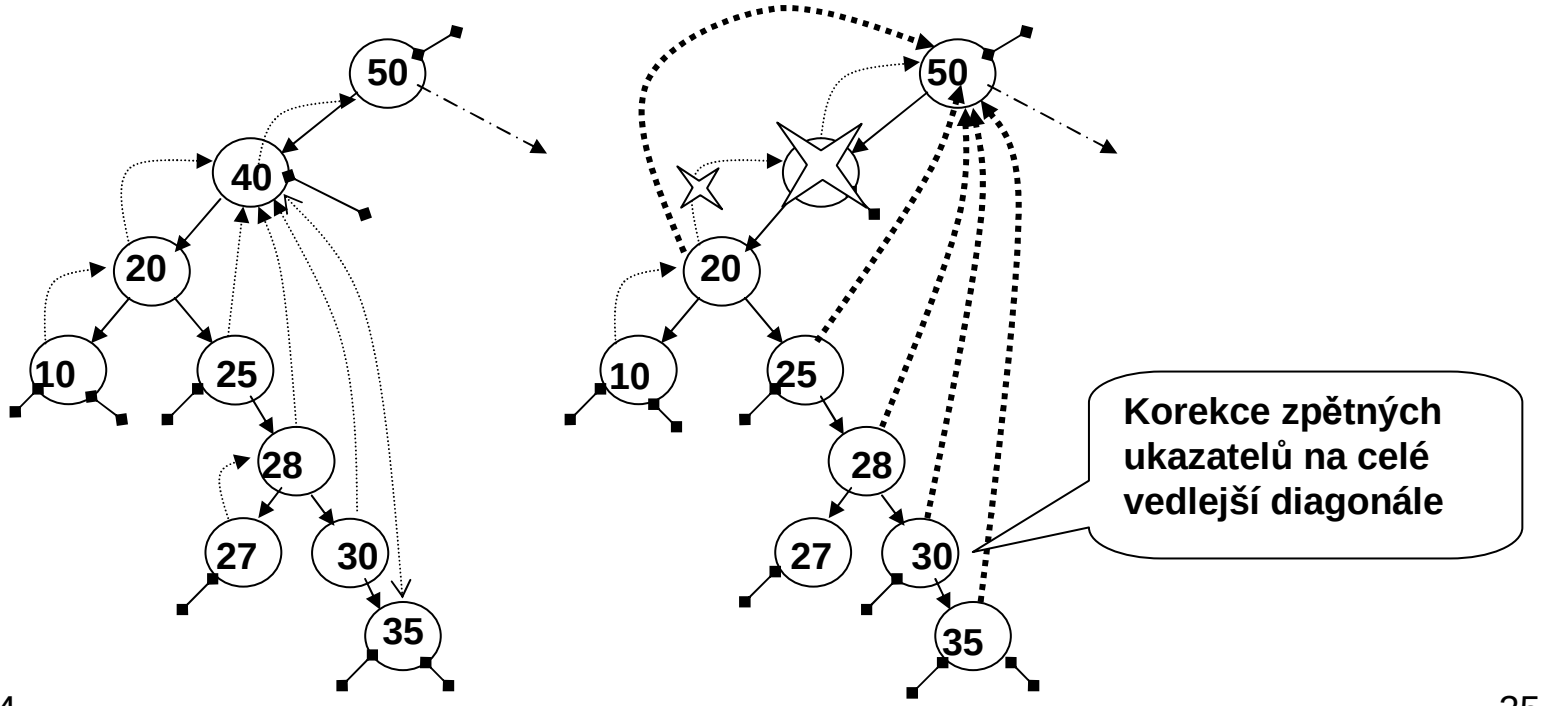
Definujme typ:

```
 TUk=^TPoI;
  TPol=record
    Klic:TKlic;
    Data:TData;
    LUk,PUk,ZpetUk:TUk;
  end;
procedure Nejlev(UkKor:TUk;var UkNaNejlev:TUk);
(* procedura vrátí ukazatel na nejlevější uzel stromu *)
begin
  UkNaNejlev :=UkKor;
  while UkKorr <> nil do begin
    UkNaNejlev :=UkKor;
    UkKor:=UkKor^.LUk
  end;
```

```
procedure Inorder(UkKor:TUk; var DL:TDlist);
(* Průchod Inorder vkládá data do dvojsměrného seznamu *)
var  Fin:Boolean;
       UkNaNejlev:TPtr;
begin
  DListInit (DL); (* Inicializace seznamu *)
  Nejlev(Ukkro,UkNaNejlev);
  Fin:=UkNaNejlev=nil;  (* řídicí proměnná cyklu *)
  while not Fin do begin
    DInsertLast(DL, UkNaNejlev^.Data); (*vkládání do sezn. *)
    if UkNaNejlev^.PUk<>nil
    then Nejlev(UkNaNejlev^.PUk, UkNaNejlev)
    else
    if UkNaNejlev^.ZpetUk=nil
      then Fin:=true
      else UkNaNejlev:=UkNaNejlev^.ZpetUk   (* pohyb zpet *)
  end;
end; (* procedure *)
```

*Korekce BVS se zpětnými ukazateli při operaci Delete

Operace Delete je stejná, jako u normálního BVS. Ke korekci ukazatelů dochází pouze v případě, že se ruší uzel jen s jedním a to levým podstromem. Pak se korigují všechny ukazatele na pravé diagonále jeho levého podstromu.



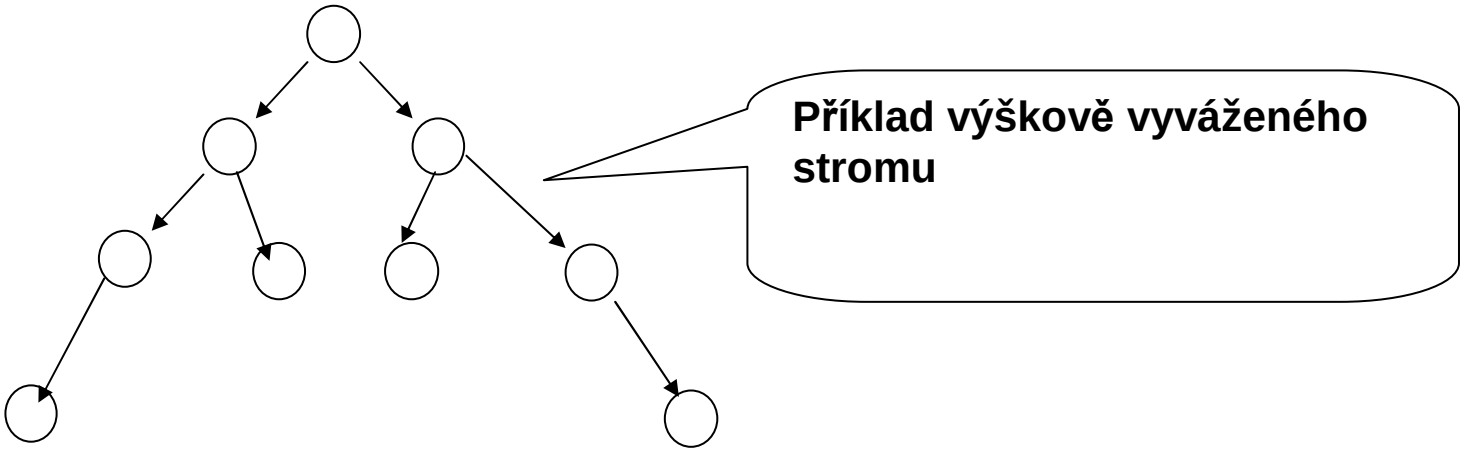
Domácí úloha

- Je dán BVS (nevyvážený). Je zadán (maximální) počet jeho uzlů. Vytvořte jeho váhově vyváženou verzi. Zapište řešení ve formě rekurzívní i nerekurzívní procedury. Zvolte vhodnou datovou strukturu uzlů a definujte potřebné typy.

Pozn. Řešte s použitím pomocného pole, do kterého vložíte všechny hodnoty nevyváženého stromu. Z pole pak vytvořte nový, váhově vyvážený strom.

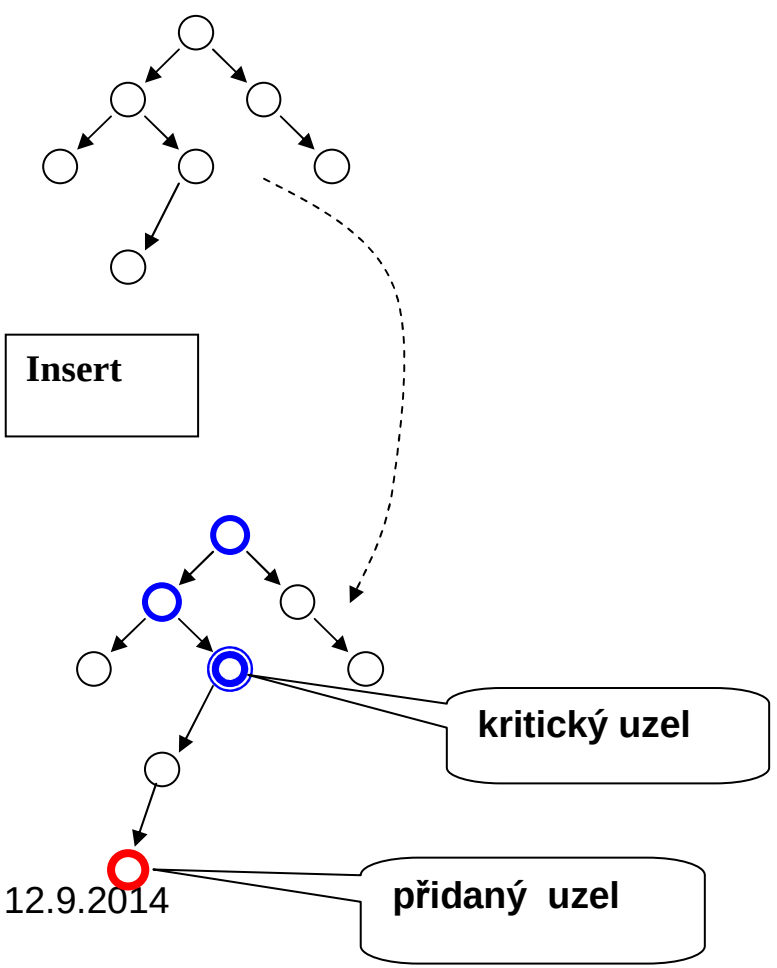
AVL stromy

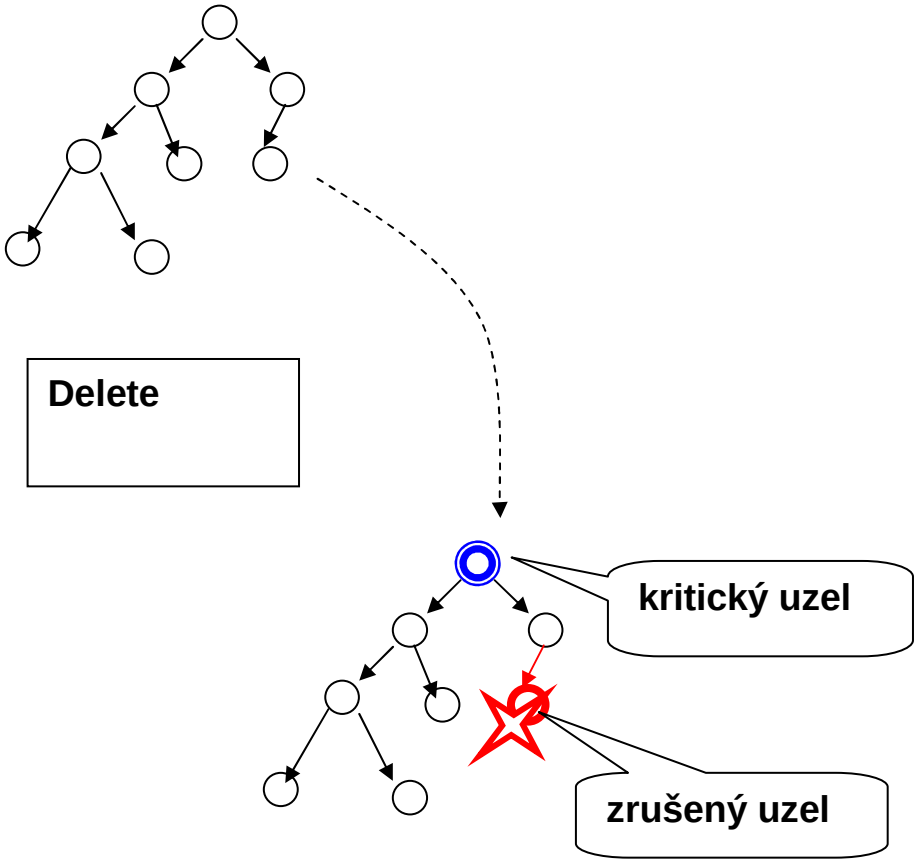
- Výškově vyvážený strom – AVL podle ruských matematiků Adělson- Velski a Landis.
- Je maximálně o 45% vyšší než váhově vyvážený strom.
- Výškově vyvážený binární vyhledávací strom je strom, pro jehož každý uzel platí, že výška jeho dvou podstromů je stejná nebo se liší o 1.
- Na rozdíl od váhově vyvážených stromů, jejichž vyváženost se při porušení v důsledku vložení nebo zrušení uzlu znovuustavuje obtížně, znovuustavení výškové vyváženosti AVL stromu lze provést omezenou rekonfigurací uzlů v okolí tzv. kritického uzlu.


Příklad výškově vyváženého stromu

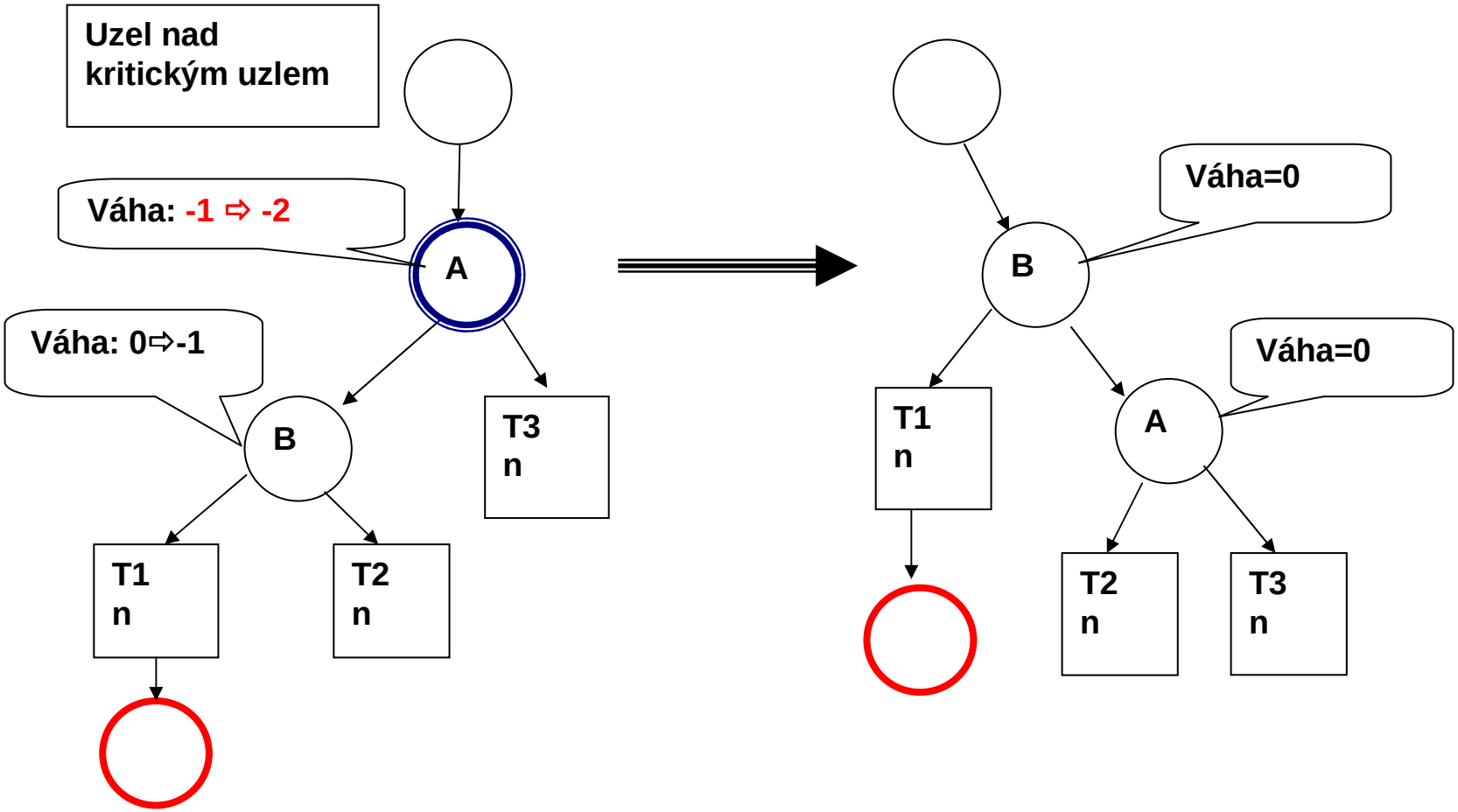
Kritický uzel nejvzdálenější uzel od kořene, v němž je v důsledku vkládání nebo rušení porušená rovnováha.

Příklady porušení rovnováhy operací Insert a Delete:



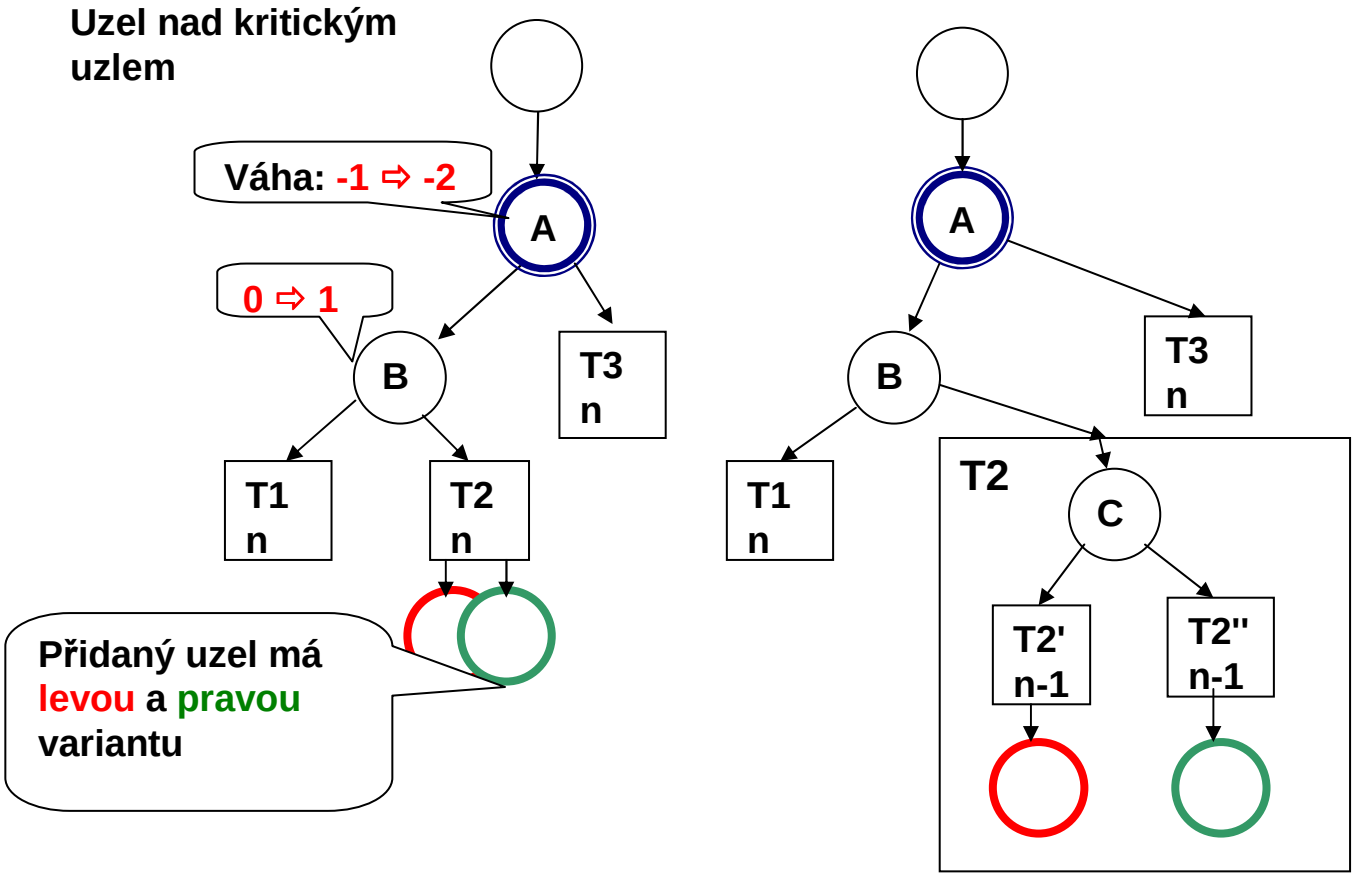


Rotace LL

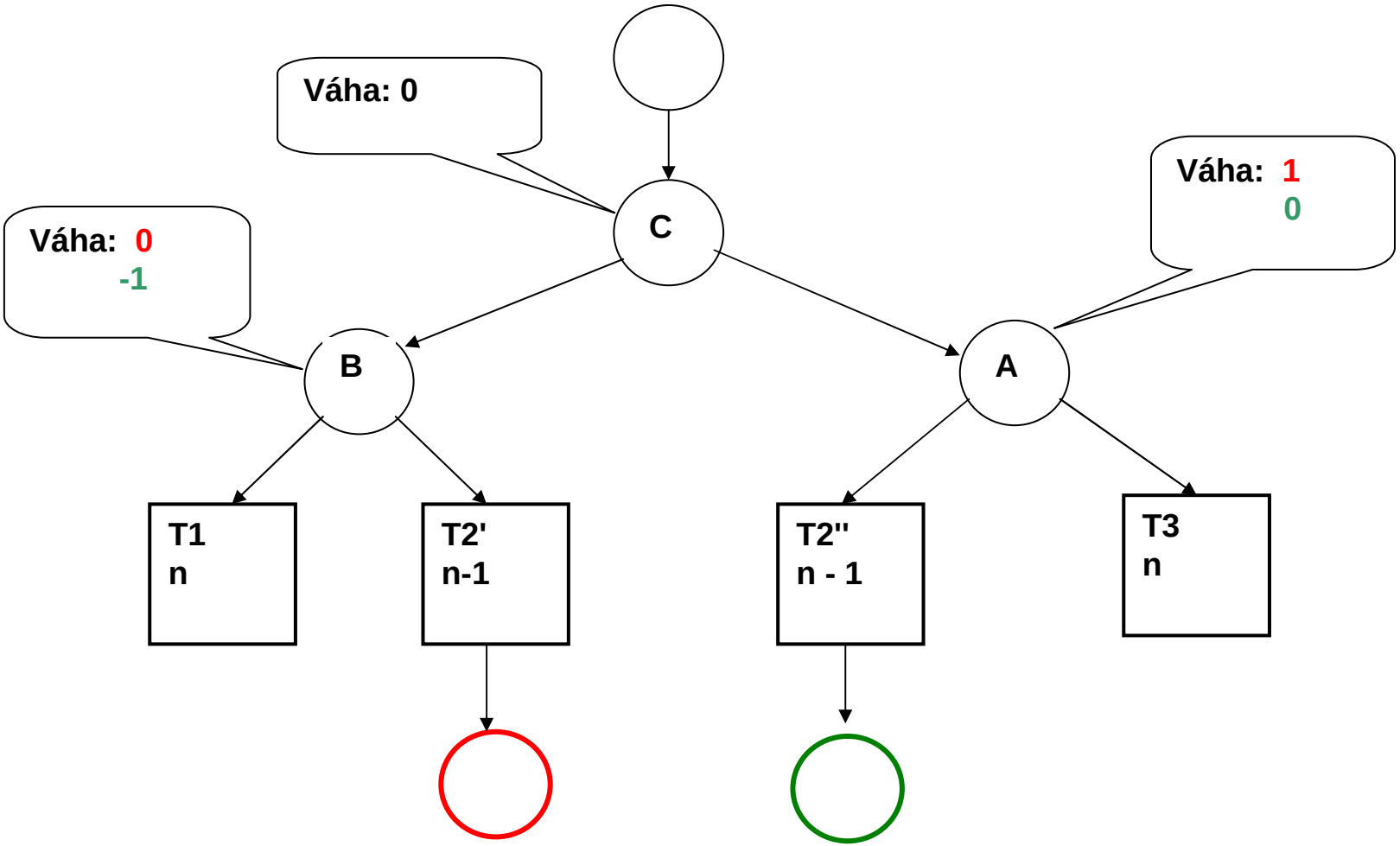

Uzel nad kritickým uzlem
Váha: -1 ⇨ -2
A
Váha: 0⇨-1
B
T3
n
T1
n
T2
n
B
Váha=0
T1
n
A
Váha=0
T2
n
T3
n

Rotace DLR

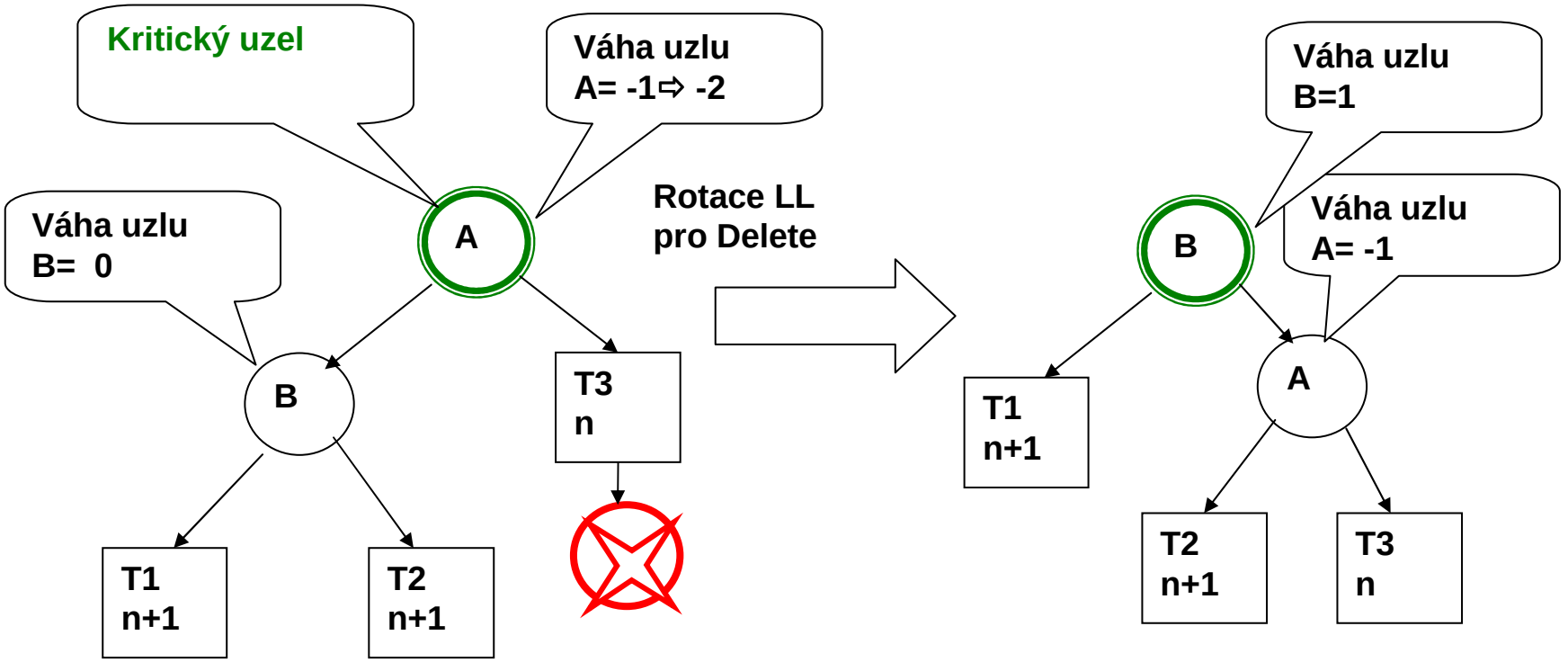
Konfiguraci lze prekreslit do tvaru uvedeného vpravo



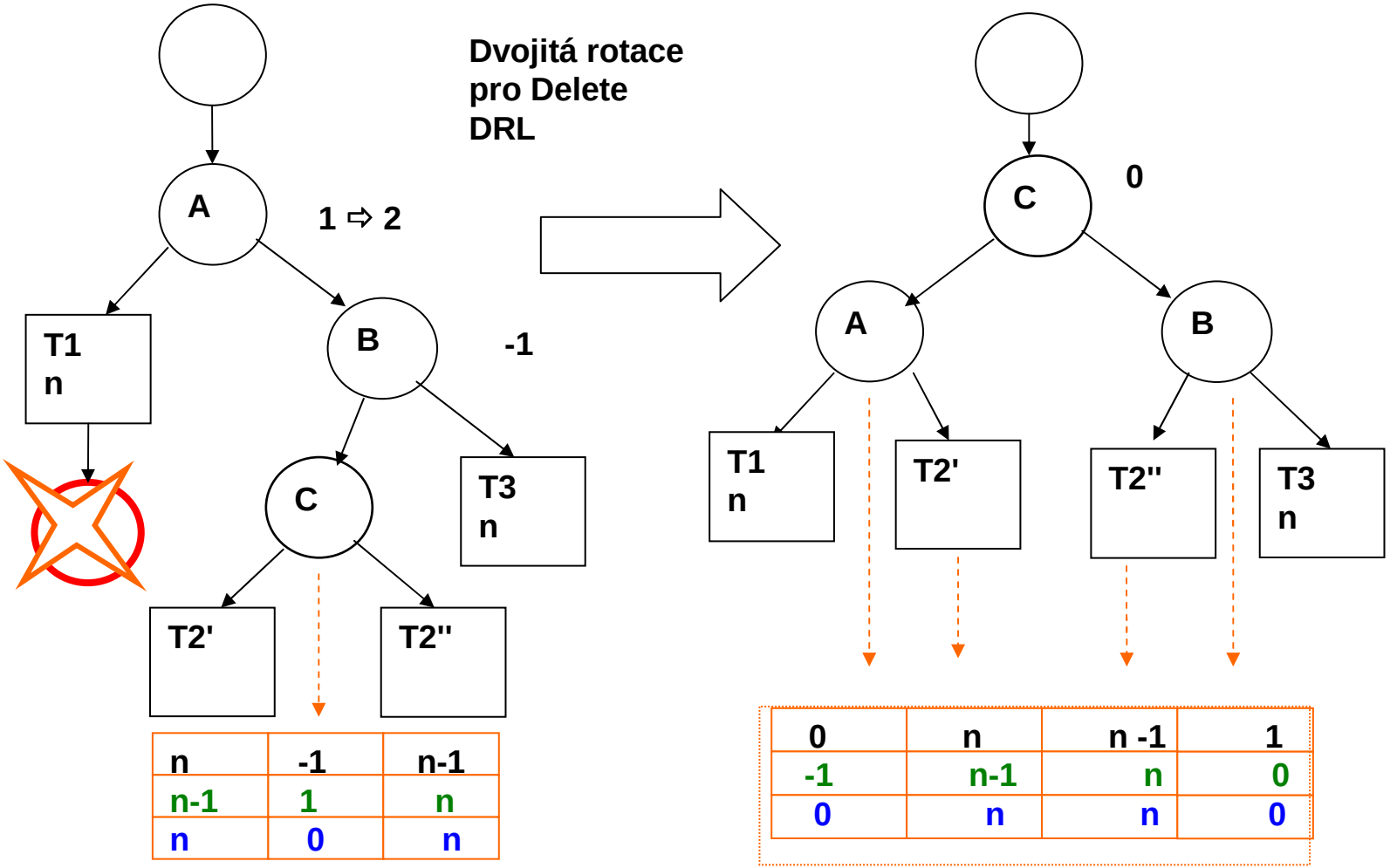
Výsledná rotace DLR



Jednoduchá rotace pro Delete LL



Dvojitá rotace pro Delete DRL



Kontrolní otázky

- Co je to Fibonacciho posloupnost?
- Co je to Fibonacciho strom.
- Definujte binární vyhledávací strom.
- Jaká je výhoda vyhledávání ve Fibonacciho stromu?
- Co je to binární vyhledávací strom se zpětnými ukazateli?
- Vysvětlete rozdíl mezi průchody preorder, inorder a postorder.
- Můžeme některým průchodem získat z binárního vyhledávacího stromu seřazenou posloupnost. Pokud ano, kterým?
- Jak se ruší uzel v binárním vyhledávacím stromu
- Nakreslete Fibonacciho strom
- Nakreslete jednoduchou rotaci pro vyvážení porušené rovnováhy po operaci Insert v AVL stromu