# IAL - Algorithmy
# 6. přednáška

Stromové etudy - implementace některých operací nad BS :

- ekvivalence struktur dvou BS
- ekvivalence dvou BS
- kopie BS
- destrukce BS
- počet listů BS,
- výška BS
- nalezení nejdelší cesty od kořene k listu
- váhová/výšková vyváženost stromu).

# Nerekurzívní zápis PreOrder

```
procedure NejlevPre(Uk:TUk; var L:TList);
(* globální ADT zásobník ukazatelů *)
begin
   while Uk<>nil do begin
        Push(S,Uk);
        InsertLast(L,Uk^.Data)
         Uk:= Uk^.LUk
   end
end;
```

```
procedure PreOrder(Uk:TUk; var L:TList);
begin
   ListInit(L); SInit(S);
   NejlevPre(Uk,L);
   while not SEmpty(S) do begin
      Top(S,Uk); Pop(S);
       NejlevPre(Uk^.PUk,L)
   end
end;
```

# Nerekurzívní procedura InOrder

```
procedure NejlevIn(Uk:TUk);
(* globální ADT zásobník ukazatelů *)
begin
   while Uk<>nil do begin
       Push(S,Uk);
       Uk:= Uk^.LUk;
    (* InsertLast(L, Uk^.Data) *)
   end
end;
```

```
procedure InOrder(Uk:TUk; var L:TList);
begin
   ListInit(L); SInit(S);
   NejlevIn(Uk,L);
   while not SEmpty(S) do begin
      Top(S,Uk); Pop(S);
       InsertLast(L, Uk^.Data);
       NejlevIn(Uk^.Puk,L)
   end
end;
```

# Nerekurzívní PostOrder – dvouzásobníkový průchod

Postorder se vrací k „otci“ dvakrát. Poprvé zleva – aby šel doprava a podruhé zprava, aby „zpracoval” otcovský uzel. Zásobník booleovských hodnot provede rozlišení

```
procedure NejlevPost(Uk:TUk);
(* Globální ADT zásobník ukazatelů S a zásobník
Booleovských hodnot SB *)
begin
   while Uk<>nil do begin
      Push(S,Uk);
      PushB(SB, true);
      Uk:=Uk^.LUk
end;
```

```
procedure PostOrder(Uk:TUk; L:TList);
var   Zleva:Boolean;
begin
  ListInit(L); SInit(S); SInitB(SB);
  NejlevPost(Uk);
  while not SEmpty(S) do begin
    Top(S,Uk); TopB(SB, Zleva); PopB(SB);
    if Zleva
    then begin (* přichází zleva, půjde doprava *)
      PushB(SB, false);
      NejlevPost(Uk^.Puk)
    end else begin(* zprava, odstraní a zprac. otc. uzel *)
      Pop(S);
      InsertLast(L,Uk^.Data)
    end (* if *)
  end (* while *)
end;
```

x+y

# Úloha k procvičení

- Vytvořte nerekurzívní proceduru, která vhodným parametrem určí, zda se zadaný průchod binárním stromem do zadaného seznamu uloží v podobě PreOrder, InOrder neb PostOrder.
- Napište nerekurzívní proceduru PostOrder pomocí inverzního PreOrderu s „jedním“ zásobníkem.

# Stromové etudy, rekurzívní i nerekurzívní verze

1. Vytvořte rekurzívní/nerekurzívní proceduru, která zruší všechny uzly zadaného BS.
2. Vytvořte rekurzívní/nerekurzívní proceduru, která vytvoří kopii (duplikát) zadaného zdrojového BS.
3. Vytvořte rekurzívní/nerekurzívní proceduru pro ekvivalenci struktur dvou stromů. (Procedura neporovnává hodnoty uzlů, jen to, zda oba mají stejnou konfiguraci synů).

4. Vytvořte rekurzívní/nerekurzívní proceduru pro ekvivalenci dvou binárních stromů.(Procedura na rozdíl od předchozí procedury porovnává i hodnoty uzlů)
5. Vytvořte rekurzívní/nerekurzívní proceduru/funkci, která zjistí výšku binárního stromu.
6. Vytvořte proceduru/funkci, která spočítá průměrnou vzdálenost a rozptyl vzdáleností listů od kořene BS.

7. Vytvořte proceduru, která nalezne a do výstupního seznamu uloží nejdelší cestu od kořene k listu (od listu ke kořeni). Pozn. Můžete využít procedury z příkladu 2.
8. Je dáno seřazené pole prvků typu integer. Vytvořte Binární vyhledávací strom, který je váhově vyvážený

## Zrušení BS

```
procedure ZrusStrom(var kor:TUkUz);
(* předpokládá se rušení neprázdného stromu, tedy kor<>nil *)
begin
  InitStack(S); (* inicializace zásobníku S pro ukazatele na uzel *)
    repeat  (* cyklus rušení *)
    if kor=nil (* je-li ukazatel nilový, vezmu další ze zásobníku *)
    then begin
      if not SEmpty(S)
      then begin
        kor:=TOP(S);
        POP(S);
      end  (* if not SEmpty *)
    end else begin (* jde doleva, likviduje a pravé strká do zásobníku *)
      if kor^PUK <>nil
      then PUSH(S,Kor^.PUK);
      PomPtr:=Kor; (* uchování dočasného kořene pro pozdější zrušení *)
      Kor:=Kor^.LUK; (* posud doleva *)
      dispose(PomPtr); (* zrušení starého uchovaného kořene *)
     end; (* if*)
 until (Kor=nil) and SEMPTY(S) (* končím: Kor je nil, zás. je prázdný *)
end;
```

## Obdobný algoritmus zrušení stromu s využitím „Nejlev“

```
procedure ZrusUzel (var kor:TUkUz);
(* tato verse s cyklem repeat předpokádá rušení neprázdného BS *)
var
  uz:TUkUz;
procedure nejlev (kor:TUkUz);
begin
  while kor<>nil do begin
   Push(S,kor);
   kor:=kor^.LUk;
end;  (* procedure *)
begin; (* tělo hlavní procedury *)
  SInit(S);
  uz:=kor;
  repeat
    nejlev(uz);
    uz:=Top(S);Pop(S); (* uz je nejlevější na diag. *)
    if uz^.PUK <> nil
    then  Push(S,uz^.PUk);
    dispose(uz);
  until SEMPTY(S);
end;
```

# Vytvoření váhově vyváženého binárního stromu ze seřazeného pole (rekurzívně)

```
pocedure StromZPole(var Kor:TUk; LevyInd,
                    PravyInd:  integer; Pole:TPole);
var Stred:integer;
begin
  if LevyInd <= PravyInd
  then begin
      Stred:=(LevyInd+PravyInd) div 2;
      Vytvoř(Koren, Stred);
      StromZPole(Koren^.Luk, Levy, Stred-1,Pole);
      StromZPole(Koren^.Puk, Stred+1, Pravy,Pole);
   end else begin
     Koren:= nil
   end (* if *)
end
```

# Výška stromu rekurzívně

```
procedure VyskaBS(Kor:TUk; var Max:integer);
var Pom1, Pom2:integer;
begin
   if Kor <> nil
   then begin
      VyskaBS(Kor^.LUk, Pom1);
      VyskaBS(Kor^.PUk, Pom2);
       if Pom1> Pom2
       then Max := Pom1 + 1
       else Max := Pom2 + 1
   else Max:= 0
end; (* procedure *)
```

# Varianta rekurzívní výšky

```
function Vyska (Uk:TUk):integer;
function Max(N1,N2):integer;
```

(* funkce vrátí hodnotu většího ze dvou vstupních parametrů *)

```
begin
   if N1 > N2
   then Max:= N1
   else Max:= N2
end;
begin
  if Uk=nil
  then Vyska:= 0
  else Vyska:= Max(Vyska(Uk^.Luk),
                   Vyska(Uk^.Puk)) + 1
end; (* function*)
```

# Ekvivalence struktur dvou stromů – rekurzívní zápis

```
function EQTS(Kor1, Kor2:Tuk):Boolean;
begin
  if (Kor1=nil) or (Kor2=nil)
  then EQTS:= (Kor1=Kor2)
  else EQTS := EQTS(Kor1^.LUk, Kor2^.LUk)
               and EQTS(Kor1^.PUk, Kor2^.PUk)
       (* and (Kor1^.Data = Kor2^.Data)  *)
       (* pro ekvivalenci  BS*)
end;
```

# Kopie BS – rekurzívní zápis

```
procedure CopyTree (KorOrig:TUk; var KorCopy:TUk);
begin
  if KorOrig <> nil
  then begin
     new(KorCopy);
     KorCopy^.Data := KorOrig^.Data;

     CopyTree(KorOrig^LUk, KorCopy^.LUk);
     CopyTree(KorOrig^PUk, KorCopy^.PUk);
  end else
     KorCopy:=nil
end;
```

# Nerekurzívní kopie BS

```
procedure NRCT(RootPtrI:TPtr; var RootPtrO:TPtr);
procedure LeftMost(Ptr1:TPtr; var Ptr2:TPtr);
var
    TmpPtr:TPtr; (* náhrada za výst. par. Ptr2, který je
   třeba uchovat, protože je to kořen výsledného stromu *)
begin
  if PTr1<>nil
  then begin
    new(Ptr2);
    Ptr2^.Data:=Ptr1^.Data; (* kopírování dat *)
    PushPtr(S1,Ptr1); (* originál do zásobníku *)
    Ptr1:=Ptr1^.LPtr; (* posun po diagonále
   originálu *)
    TmpPtr:=Ptr2; (* Náhrada výst. par. který musí
   být uchován *)
```

```
    while Ptr1<>nil do begin
      PushPtr(S2,TmpPtr);
      new(TmpPtr^.LPtr);
      TmpPtr:=TmpPtr^.LPtr; (*po diagon. kopie *)
      TmpPtr^.Data:=Ptr1^.Data; (* kopie dat *)
      Ptr1:=Ptr1^.LPtr; (* po diagonále originálu *)
      PushPtr(S1,Ptr1); (* vlož uk na orig. do
  zásobníku *)
    end;
  end else begin
    Ptr2:=nil (* vyvoření nilového kořenu,  nebo
  nilového pravého ukazatele z nejlevějšího uzlu  *)
  end; (* if *)
end; (* procedure LeftMost *)
```

```
var
   S1,S2:TStackPtr;
   AuxPtrI,AuxPtrO:TPtr; (* pomocný ukazatel *)
begin (* tělo hlavní procedury *)
  SInitPtr(S1); (* Inicializace zásobníků *)
  SInitPtr(S2);
  LeftMost(RootPtrI,RootPtrO);
  while not SEmptyPtr(S1) do begin
    TopPtr(S1,AuxPtrI);
    PopPtr(S1);
    TopPtr(S2,AuxPtrO);
    PopPtr(S2);
    LeftMost(AuxtPtrI^.RPtr,AuxPtrO^.RPtr);
  (* LeftMost pro pravé syny *)
  end; (* while *)
end; (* procedure *)
```

Test váhové vyváženosti BS

```
procedure TESTBVS (var Kor:TUk; var
Balanced:Boolean;var Pocet:integer);
var  VYVL, VYVP:Boolean;
      PocL, PocP:integer;
begin
   if Kor <> nil
   then begin
     TESTBVS(Kor^.Luk, VYVL, PocL);
     TESTBVS(Kor^.Puk, VYVP, PocP);
      Pocet:=PocL + PocP +1;
      Balanced:= VYVL and VYVP and
                          (abs(PocL-PocP) <= 1);
    end else begin
       Pocet:=0;
       Balanced:= true
   end (* if*)
end (* procedure *)
```