Stromy

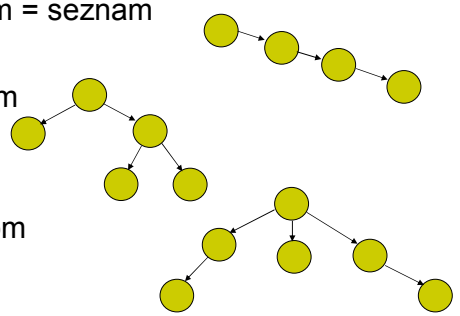
doc. Ing. Miroslav Beneš, Ph.D.
katedra informatiky FEI VŠB-TUO
A-1007 / 597 324 213
http://www.cs.vsb.cz/benes
Miroslav.Benes@vsb.cz

Základní pojmy

- Graf
  - uzly
  - hrany – orientované / neorientované
- Souvislý graf
- Acyklický graf
- Strom
  - souvislý, acyklický orientovaný graf – DAG
  - každý uzel má *nejvýše jednoho* předchůdce
  - kořen, list

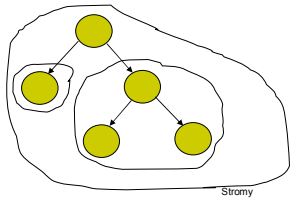
Příklady

- Unární strom = seznam
- Binární strom
- Obecný strom

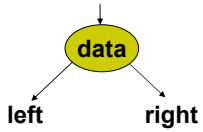


Rekurzivní datové struktury

- Seznam = Nic
  | Uzel(data: Object, next: Seznam)
- Strom = Nic
  | Uzel(data: Object, left: Strom, right: Strom)



Binární strom



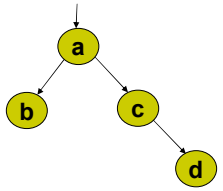

```
public interface Node
{
  Object getData();            // data
  void setData(Object data);

  Node   getLeft();     // levý podstrom
  Node   getRight();    // pravý podstrom
}`
```


Implementace binárního stromu

```
public class TreeNode implements Node
{
   public TreeNode(Object d, Node l, Node r)
   {
      data = d; left = l; right = r;
   }
   public Object getData() { return data; }
   public void setData(Object data) {
      this.data = data;
   }
   public Node getLeft() { return left; }
   public Node getRight() { return right; }
   private Object data;
   private Node left, right;
}
```


Příklad




```
Node root =
   new TreeNode("a",
      new TreeNode("b", null, null),
      new TreeNode("c",
         null,
         new TreeNode("d", null, null)));
```


Rekurzivní funkce

Problém: výpočet délky seznamu

Rozdělíme na dvě části:

a) Triviální případ – řešení známe přímo

b) Obecný případ

- předpokládáme, že už máme výsledek vypočtený pro problém o krok „menší“
- z této hodnoty a dalších údajů vypočteme výsledek

length [ ] = 0

length (x : zbytek) = 1 + length zbytek

Počet uzlů ve stromu

```
static int pocetUzlu(Node root) {
   if( root == null )
      return 0;   // triviální případ
   // řešení „menšího" problému
   int pocet_vlevo =
      numNodes(root.getLeft());
   int pocet_vpravo =
      numNodes(root.getRight());
   // výsledek
   return pocet_vlevo + pocet_vpravo + 1;
}
```


Nerekurzivní řešení

- Často bývá náročnější
- Strom = nelineární struktura
  - v cyklu můžeme projít přes celou levou větev
  - co s pravou větví?
    - poznamenáme si ji do zásobníku
    - dojdeme-li na levý konec, vybereme prvek ze zásobníku a pokračujeme opět směrem doleva
    - je-li zásobník prázdný, výpočet končí
- **Závěr: snažte se najít rekurzivní řešení**

Převod stromu na text

```
public String toString()
{
  return " ("
    + data
    + (left==null ?
         " null" : left.toString())
    + (right==null ?
         " null" : right.toString())
    + ")";
}

// operace + se vnitřně realizuje pomocí
// objektu StringBuffer
```


Převod stromu na text

```
// ekvivalentní řešení
public String toString()
{
  StringBuffer sb = new StringBuffer(" (");
  sb.append(data)
    .append(left==null ?
            " null" : left.toString())
    .append(right==null ?
            " null" : right.toString())
    .append(")");
  return sb.toString();
}
```

Převod stromu na text

```
// efektivnější řešení
// StringBuffer se vytváří pouze jednou

public String toString()
{
   StringBuffer sb = new StringBuffer();
   this.toString(sb);
   return sb.toString();
}
```


Převod stromu na text

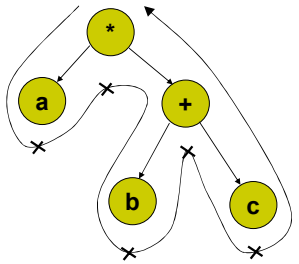
```
// pomocná metoda

private void toString(StringBuffer sb)
{
  sb.append(" (");
  sb.append(data);
  if( left==null ) sb.append(" null");
    else left.toString(sb);
  if( right == null ) sb.append(" null");
    else right.toString(sb);
  sb.append(")");
}
```


Průchody binárním stromem

- Preorder
  - data – levý podstrom – pravý podstrom
- Inorder
  - levý podstrom – data – pravý podstrom
- Postorder
  - levý podstrom – pravý podstrom - data

Průchody binárním stromem



inorder

preorder * a + b c

inorder a * b + c

postorder a b c + *

Převod stromu na seznam

```
public static final int PREORDER = 1;
public static final int INORDER = 2;
public static final int POSTORDER = 3;

public List traverse(int order)
{
  List list = new ArrayList();
  traverse(list, order);
  return list;
}
```

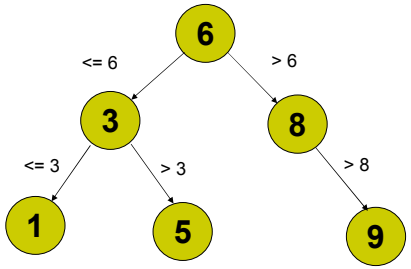

Převod stromu na seznam

```
private void traverse(List list, int order)
{
  if( order == PREORDER )
    list.add(data);
  if( left != null )
    left.traverse(list, order);
  if( order == INORDER )
    list.add(data);
  if( right != null )
    right.traverse(list, order);
  if( order == POSTORDER ) list.add(data);
}
```


Příklad – výpis uzlů stromu

```
public void print(int order)
{
  List list = this.traverse(order);
              //  System.out.println(list);
  Iterator it = list.iterator();
  System.out.print("[ ");
  while( it.hasNext() ) {
     System.out.print(it.next());
     System.out.print(" ");
  }
  System.out.println("]");
}
```


Binární vyhledávací strom



Vyhledání uzlu ve stromu - rekurzivní varianta

```
boolean contains(Comparable obj)
{
 int c = obj.compareTo(this.data);
 if( c == 0 ) return true;
 if( c < 0 )
   return left==null ?
      false : left.contains(obj);
 else
   return right==null ?
      false : right.contains(obj);
}
```


Vyhledání uzlu ve stromu - nerekurzivní varianta

```
boolean contains(Comparable obj)
{
  Node node = this;
  while( node != null ) {
    int c = obj.compareTo(node.getData());
    if( c == 0 ) return true;
    node = c < 0 ? node.getLeft();
                 : node.getRight();
  }
  return false;
}
```


Reprezentace výrazu stromem

- Vnitřní uzly:
  - operátory: + - * / ...
- Listy:
  - operandy: čísla, proměnné, ...
- Vyhodnocení výrazu
  - průchod typu postorder
  - nejprve se vyhodnotí operandy, pak operátor

Reprezentace výrazu stromem

```
interface Expr
{
   int eval();  // vyhodnocení výrazu
}

abstract class BinaryExpr implements Expr
{
   Expr left;
   Expr right;
}
```

Reprezentace výrazu stromem

```
class PlusExpr extends BinaryExpr
{
   public int eval() {
      return left.eval() + right.eval();
   }
}
class ConstExpr implements Expr
{
   public ConstExpr(int value) {
      this.value = value;
   }
   public int eval() { return value; }
   private int value;
}
```


Reprezentace výrazu stromem

```
// (3 + 2) + 5

Expr expr =
   new PlusExpr(
      new PlusExpr( new ConstExpr(3),
                    new ConstExpr(2) ),
      new ConstExpr(5)
   );

System.out.println(expr.eval());
```


Úkol pro cvičení

- Rozšiřte reprezentaci výrazu o třídy pro operátory násobení, změny znaménka
- Doplňte reprezentaci výrazu o metodu toString() umožňující převod výrazu na řetězec
  - pozor na závorky, je třeba je doplnit!
  - “quick&dirty” řešení – závorky doplnit vždy