# Aplikovaná informatika

Podklady předmětu
Aplikovaná informatika
pro akademický rok 2013/2014
Radim Farana

8



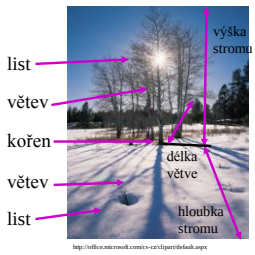
## Obsah

- Dynamické datové struktury.
- Strom.
- Binární stromy.
- Vyhledávací stromy.
- Vyvážené stromy.
- AVL stromy.



## Strom

- Název z analogie se stromy.
- Základní prvek – uzel, vrchol.
- Odkazy na konečný počet podstromů.
- Stupeň stromu – maximální počet podstromů.



## Strom

- Kořen – základní prvek stromu, nemá předchůdce, jen následníky.
- Prázdný strom neobsahuje žádný prvek.
- Prvky se stejnou vzdáleností od kořene leží na stejné úrovni.
- Maximální úroveň = hloubka (výška) stromu.
- Délka cesty k prvku – počet spojnic na cestě od kořene k prvku.
- Součet délek cesty všech prvků – délka cesty stromu.



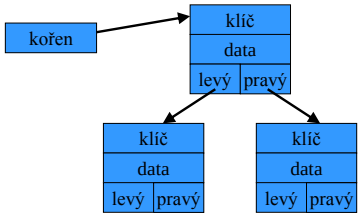
## Strom

- Minimální délku má strom jestliže na každé úrovni kromě poslední má maximální možný počet prvků.
- Pak má současně minimální hloubku.



## Binární strom

- Každý prvek má nejvýše dva následníky.



## Vyhledávací strom

- Vyhledávací stromy jsou stromy, u kterých zařazujeme prvky podle klíčové položky tak, že pro všechny prvky platí – levý následník má vždy nižší hodnotu klíče než prvek a pravý následník hodnotu vyšší.
- Algoritmy práce se stromy jsou velmi často rekurzívní.



## Binární vyhledávací strom

```
hledej (klíč, ByRef prvek)
  w = prvek
  if w = nil then
    new(w)
    w↑.klíč = klíč
    w↑.data = data
    w↑.levý = nil
    w↑.pravý = nil
  else
    if klíč < w↑.klíč then
      hledej (klíč, w↑.levý)
    else
      if klíč > w↑.klíč then
        hledej (klíč, w↑.pravý)
      else
        REM zpracování dat w.data
      end if
    end if
  end if
end hledej
```

Vyhledávání a vkládání prvků do stromu

Nebezpečí:
Degenerace stromu na lineární seznam při vkládání prvků v pořadí podle velikosti.
Přetečení zásobníku při rekurzivním volání.



## Binární vyhledávácí strom

```
h = true
w = kořen
while w <> nil and h
  if w↑.klíč = klíč then
    h = false
  else
    if w↑.klíč > klíč then
      w = w↑.levý
    else
      w = w↑.pravý
    end if
  end if
end while
```

Vyhledávání prvků iterační algoritmus

# Binární vyhledávací strom

```
projdi (prvek)
  if prvek <> nil then
    projdi (prvek↑.levý)
    zpracování dat prvek↑.data
    projdi (prvek↑.pravý)
  end if
end projdi
```

Procházení a zpracování prvků stromu v pořadí jejich klíčů

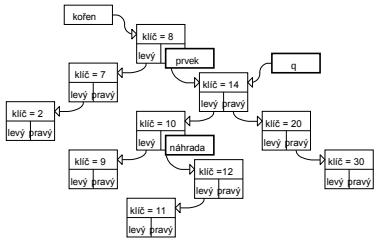
```
zruš (ByRef prvek)
  if prvek <> nil then
    zruš (prvek↑.levý)
    zruš (prvek↑.pravý)
    dispose(p)
  end if
end zruš
```

Zrušení celého stromu



# Binární vyhledávací strom

Vyhledávání a rušení prvku klíč = 14






# Binární vyhledávací strom

```
vezmi (klíč, ByRef prvek)
  if prvek = nil
    REM hledaný prvek ve stromu není
  else
    if klíč<prvek↑.klíč then
      vezmi (klíč, prvek↑.levý)
    else
      if klíč>prvek↑.klíč then
        vezmi (klíč, prvek↑.pravý)
      else
        REM hledaný prvek byl nalezen
        q = prvek
        if q↑.pravý = nil then
          prvek = q↑.levý
        else
          if q↑.levý = nil then
            prvek = q↑.pravý
          else
            najdi (q↑.levý)
          end if
        end if
        dispose (q)
      end if
    end if
  end if
end vezmi
```

Vyhledávání a rušení prvků stromu

```
najdi (ByRef náhrada)
  if náhrada↑.pravý<>nil
    najdi (náhrada↑.pravý)
  else
    q↑.klíč = náhrada↑.klíč
    q↑.data = náhrada↑.data
    q = náhrada
    náhrada = náhrada↑.levý
  end if
end najdi
```

## Vyvážený strom

- Odstranění nebezpečí degenerace stromu.
- Konstrukce stromu s minimální hloubkou.
- Strom je vyvážený, jestliže pro každý prvek platí, že počet prvků v jeho levém a pravém podstromu se liší nejvýše o jeden prvek.



## Vyvážený strom

- Nejjednodušší je umisťování prvků střídavě na levou a pravou stranu stromu – strukturování stromu:
  - zvolíme jeden z prvků jako kořen stromu,
  - vytvoříme levý podstrom s počtem $n_l = n$ div 2 prvků.
  - vytvoříme pravý podstrom s počtem $n_r = n - n_l - 1$ prvků
- Výsledný strom je dokonale vyvážený.
- Pro realizaci vyhledávacího stromu při náhodném vkládání prvků je po každém vložení nutná restrukturalizace stromu.



## AVL-strom

- Strom je vyvážený právě tehdy, když se výšky obou podstromů každého prvku liší nejvýše o jeden prvek.
- Formulovali: Adelson-Velskij a Landis.
- V nejhorším případě bude AVL-strom o 45 % vyšší než dokonale vyvážený strom.
- Se složitostí $O(\log n)$ lze realizovat všechny základní operace – vložení, vyhledání i zrušení prvku s daným klíčem

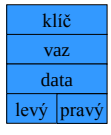
## AVL-strom

- Po vložení nového prvku do levého podstromu se z pohledu prvku:
  - vyváženost zlepší, pokud výška levého byla menší než u pravého podstromu,
  - vyváženost se zhorší, ale neporuší, pokud byly obě výšky stejné,
  - vyváženost se poruší, pokud výška levého podstromu byla větší než u pravého, vyváženost je třeba upravit.
- Oprava vyvážení se realizuje rotacemi.



## AVL-strom

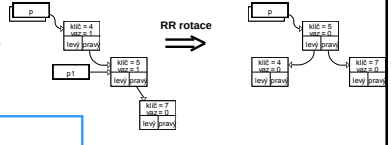
- Prvek AVL-stromu,
- proměnná *vaz* má hodnoty:
  - -1 : levý podstrom je delší,
  - 0 : oba podstromy mají stejnou délku,
  - 1 : pravý podstrom je delší.
- Proměnné:
  - p – ukazatel na prvek,
  - h – informace o zvětšení výšky.






## AVL-strom, rotace RR

- Porušení způsobil u pravého následníka pravý podstrom
- Jednoduchá rotace RR



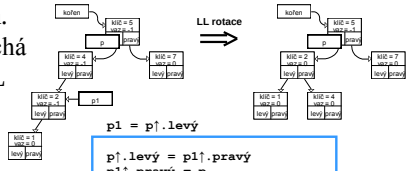

```
p1 = p↑.pravý

p↑.pravý = p1↑.levý
p1↑.levý = p
p↑.vaz = 0
p = p1

p↑.vaz = 0
h = false
```

# AVL-strom, rotace LL

- Porušení způsobil u levého následníka levý podstrom.
- Jednoduchá rotace LL




```
p1 = p↑.levý

p↑.levý = p1↑.pravý
p1↑.pravý = p
p↑.vaz = 0
p = p1

p↑.vaz = 0
h = false
```



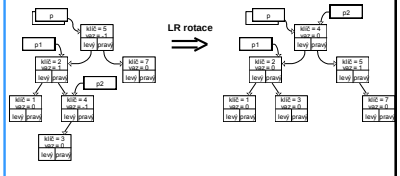
# AVL-strom, LR-rotace

```
p1 = p↑.levý

p2 = p1↑.pravý
p1↑.pravý = p2↑.levý
p2↑.levý = p1
p↑.levý = p2↑.pravý
p2↑.pravý = p
if p2↑.vaz = -1 then
        p↑.vaz = 1
else
        p↑.vaz = 0
end if
if p2↑.vaz = 1 then
        p1↑.vaz = -1
else
        p1↑.vaz = 0
end if
p = p2

p↑.vaz = 0
h = false
```

- Dvojitá LR rotace





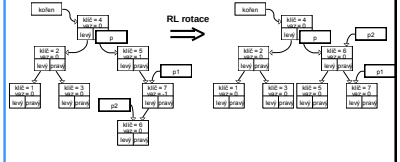

# AVL-strom, RL-rotace

```
p1 = p↑.pravý

p2 = p1↑.levý
p1↑.levý = p2↑.pravý
p2↑.pravý = p1
p↑.pravý = p2↑.levý
p2↑.levý = p
if p2↑.vaz = 1 then
        p↑.vaz = -1
else
        p↑.vaz = 0
end if
if p2↑.vaz = -1 then
        p1↑.vaz = 1
else
        p1↑.vaz = 0
end if
p = p2

p↑.vaz = 0
h = false
```

- Dvojitá RL rotace



# AVL-strom, vkládání prvků

```
hledej1 (ByRef p, ByRef h)
if p = nil
  REM vložení nového prvku do proměnné p,
      h = true
else
  if klíč < p↑.klíč then
    hledej1 (p↑.levý,h)
    if h then
      case p↑.vaz = 1
        p↑.vaz = 0
        h = false
      case p↑.vaz = 0
        p↑.vaz = -1
      case p↑.vaz = -1
        p1 = p↑.levý
        if p1↑.vaz = -1 then
          REM rotace LL
        else
          REM rotace LR
        end if
        p↑.vaz = 0
        h = false
      end case
    end if
  else
    if klíč > p↑.klíč then
      hledej1 (p↑.pravý,h)
      if h then
        case p1↑.vaz = -1
          p1↑.vaz = 0
          h = false
        case p1↑.vaz = 0
          p↑.vaz = 1
        case p1↑.vaz = 1
          p1↑.vaz = pravý
          if p1↑.vaz = 1 then
            REM rotace RR
          else
            REM rotace RL
          end if
          p1↑.vaz = 0
          h = false
        end case
      end if
    else
      REM zpracování dat p↑.data, h = false
    end if
  end if
end if
end hledej1
```