Datové struktury 1

3. přednáška

Pavel Veselý

(s díky Petrovi Chmelovi za slajdy z minula)

Plán

- Splay stromy – dokončení
- (a, b)-stromy a jejich amortizace

Zdroj: https://flog.pravda.sk/loxodonta.flog?foto=679353

Zdroj: Lípa poustevník, Tomáš Kalous – Strom roku

Amortizace

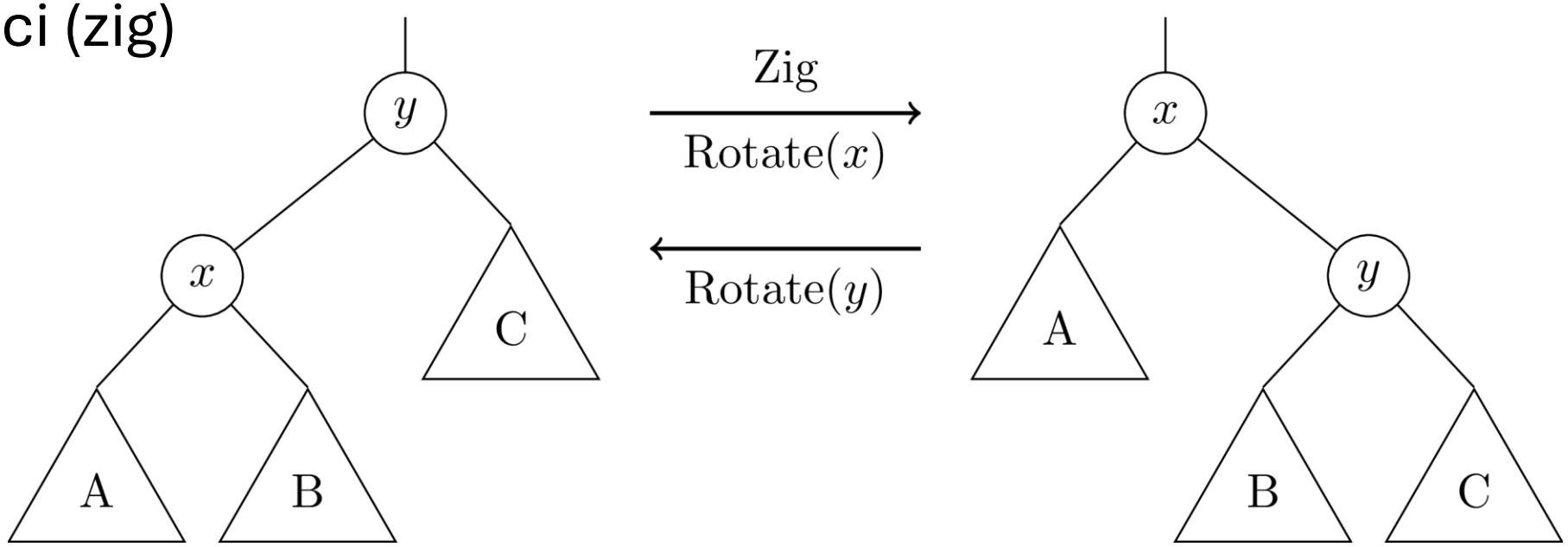
- Potenciál Φ je funkce, která na základě stavu datové struktury (a případně i historie operací) přiřadí nějakou hodnotu
  - Vysoký potenciál ~ jsme připraveni na drahou operaci
  - Nízký potenciál ~ teď budou operace levné
- Definice: operace má amortizovanou složitost *A*, pokud existuje potenciál Φ takový, že skutečná cena *C* splňuje

$$A_i \geq C_i + \Phi_i - \Phi_{i-1}$$

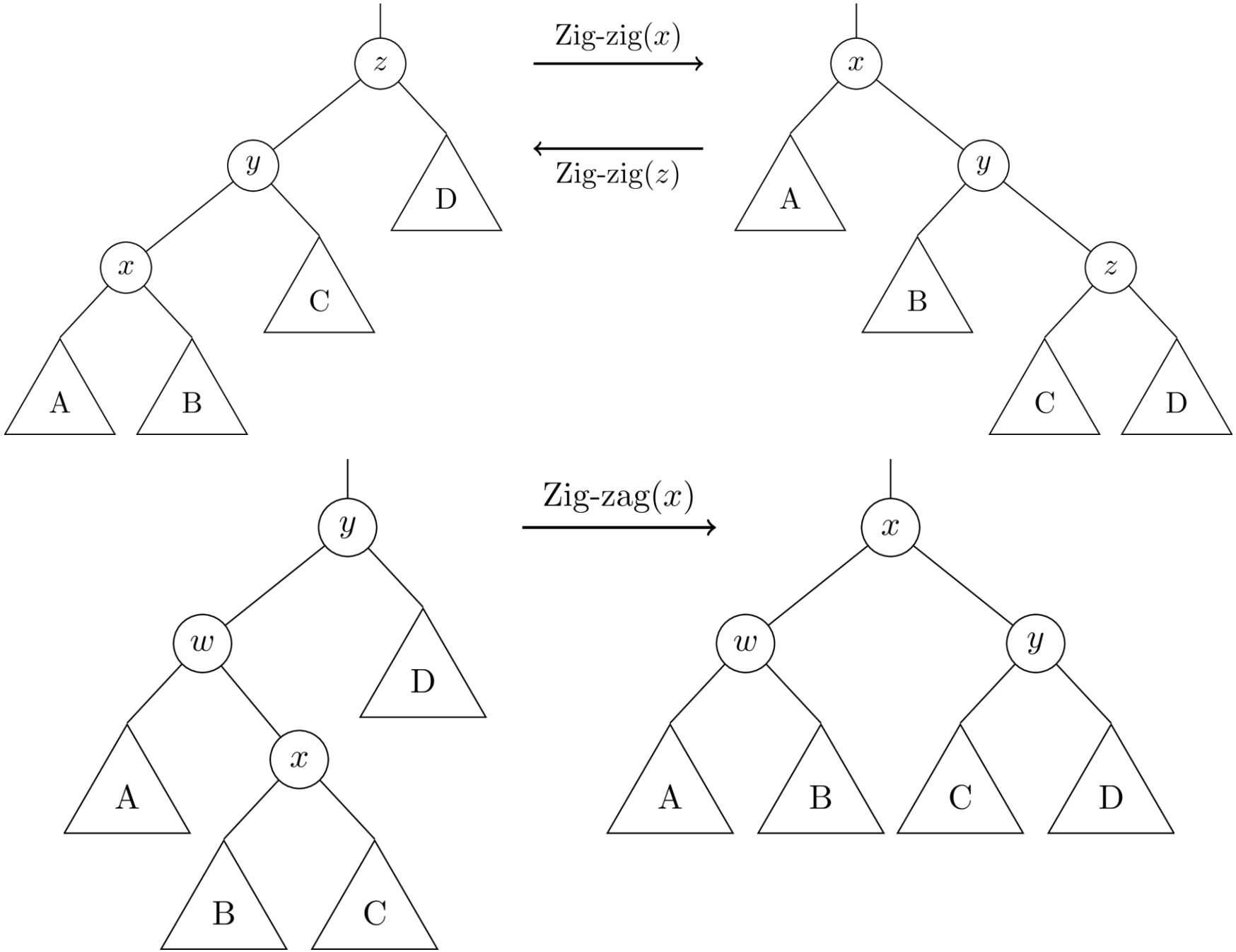
- Potom máme $\sum_i A_i \geq \Phi_m - \Phi_0 + \sum_i C_i$

Splay operace (Sleator & Tarjan 1985)

- Splay(x): přesune x do kořene
- Vezmeme vrchol x, a dokud můžeme, provádíme dvourotace (zig-zig, nebo zig-zag), na konci možná provedeme jednu jednoduchou rotaci (zig)



Dvojrotace



Cena operace Splay

- Cíl: operace Splay má amortizovanou cenu $\mathcal{O}(\log n)$
- Pro vrchol *v* zadefinujeme jeho rank $r(v) \coloneqq \log_2 s(v)$
- Pak definujeme potenciál stromu $\Phi(T) = \sum_{v \in T} r(v)$
- Věta

  Amortizovaná cena operace Splay(*x*) je $3(r'(x) - r(x)) + 1$, kde $r'(x)$ je rank po provedení splaye a $r(x)$ je rank před provedením splaye

- Zpozorujeme $r(x) \leq \log_2 n$, tedy cena Splay(x) je $\mathcal{O}(\log n)$

Find, Insert, Delete na splay stromech

- Neformálně: operaci provedeme stejně jako na standardních BVS, a pak vysplayujeme nejhlubší prvek, kterého jsme se dotkli
- `Find`
  - Úspěšný: vysplayujeme nalezený vrchol
  - Neúspěšný: vysplayujeme poslední navštívený vrchol
- `Insert`
  - Úspěšný: vysplayujeme vložený vrchol
  - Neúspěšný: vysplayujeme už existující vrchol
- `Delete`
  - Úspěšný: vysplayujeme rodiče smazaného vrcholu
  - Neúspěšný: vysplayujeme poslední navštívený vrchol

Amortizovaná složitost Find/Insert/Delete

- Tvrzení

  Všechny tyto operace mají amortizovanou časovou složitost $\mathcal{O}(\log n)$

- Lemma (pro Insert)

  Přidání listu zvýší potenciál o $\mathcal{O}(\log n)$

Vlastnosti splay stromů

- Věta (sequential access theorem, Tarjan 1985)

  Vyhledávání prvků $1, \ldots, n$ v tomto pořadí má celkovou cenu $\mathcal{O}(n)$.

- Věta (working set theorem, Sleator & Tarjan 1985)

  Buď $x_1, \ldots, x_m$ posloupnost vyhledávání prvků ve splay stromě. Označme jako z_i počet různých prvků vyhledaných od posledního hledání prvku x_i. Pak celková složitost hledání prvků je $\mathcal{O}(n \log n + m + \sum_i \log(1 + z_i))$.

- Důsledek (o vyhledávání podmnožiny prvků)

  Pokud provedeme m vyhledání prvků z podmnožiny $X \subseteq [n]: |X| = s$, pak celková cena vyhledávání bude $\mathcal{O}(n \log n + m + m \log s)$.

  (A tím pádem se dobře chová k CPU cache.)

Vlastnosti splay stromů II

- Věta (statická optimalita, Sleator & Tarjan 1985)

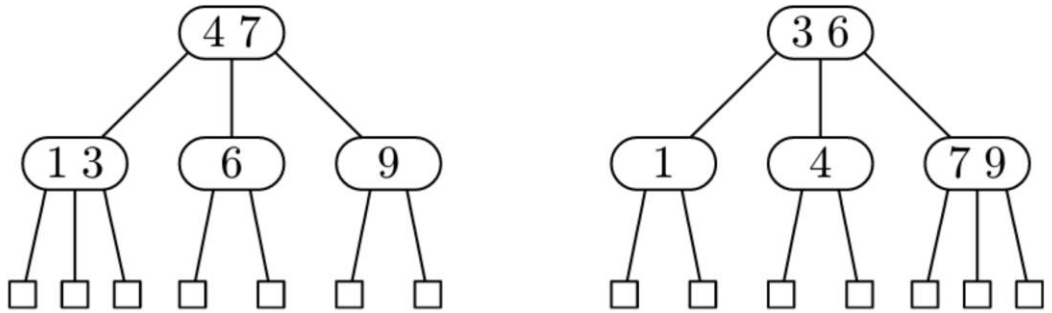
  Buď $x_1, \ldots, x_m$ posloupnost vyhledávání prvků z množiny X, kde každý prvek je vyhledán aspoň jednou. Buď T statický BVS na množině X s celkovou cenou vyhledání posloupnosti f. Pak celková cena téže posloupnosti ve splay stromě na X je $\mathcal{O}(f)$.

- Domněnka (dynamická optimalita, Sleator & Tarjan 1985)

  Buď $x_1, \ldots, x_m$ posloupnost vyhledávání prvků z množiny X, kde každý prvek je vyhledán aspoň jednou. Buď T dynamický BVS na množině X s celkovou cenou vyhledání posloupnosti f. Pak celková cena téže posloupnosti ve splay stromě na X je $\mathcal{O}(f)$.

Vícecestný vyhledávací stromy: (a, b)-stromy

- Ve vnitřním vrcholu $\ell \geq 1$ klíčů (seřazených), $\ell + 1$ synů
  - Listy prázdné (`None` / `nullptr`)
- (a, b)-stromy pro $a \geq 2$ a $b \geq 2a - 1$
  - Vnitřní vrchol má a až b synů, kořen má 2 až b synů
  - Listy na stejné hladině
  - Hloubka = # hran od kořene do listů



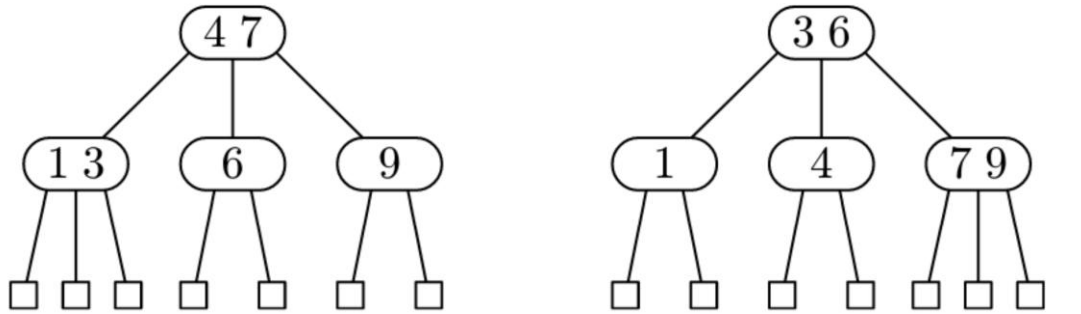
Obrázek 8.7: Dva (2, 3)-stromy pro tutéž množinu klíčů

Lemma: Hloubka (a, b)-stromu je mezi $\log_b(n+1)$ a $1 + \log_a \frac{(n+1)}{2}$

B-stromy [Bayer, McCreight '70]

- $(a, 2a-1)$- nebo $(a, 2a)$-stromy
- Různé varianty, např.:
  - data někdy jen nejnižší hladině nad listy, pak vyšší hladiny obsahují pomocné klíče

(a, b)-stromy: operace



Obrázek 8.7: Dva (2, 3)-stromy pro tutéž množinu klíčů

- `Find` – podobně jako v BVS
- `Insert`
  - Najdeme list, přidáme do vrcholu nad ním
  - Při přetečení provedeme rozštěpení
- `Delete`
  - Najdeme klíč
    - pokud není ve vrstvě nad listy, provedeme nahrazení s minimem v pravém podstromu
  - Smažeme klíč
  - Při podtečení
    - Převedeme klíč od bratra – pokud jich má dost
    - Nebo se s ním sloučíme

(a, b)-stromy: amortizovaná složitost

… proč, když každá operace trvá $\Omega(\log n)$?

- Změny stromu jsou dražší
- Vyhledávání někdy není potřeba
  - Např.: máme odkaz na mazaný prvek nebo list pro Insert

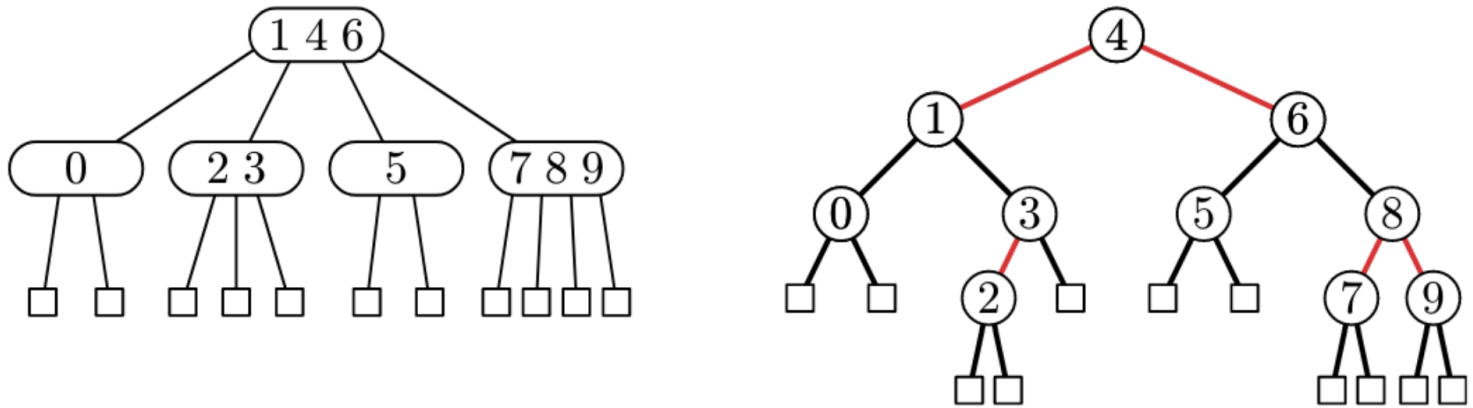
Pozorování: m operací `Insert` na prázdném (a,b)-stromě provede $O(m)$ změn uzlů.

Věta: m operací `Insert` a `Delete` na $(a, 2a)$-stromě provede $O(n + m)$ změn uzlů.

Červeno-černé stromy [Bayer '72]

varianta left-leaning (**LLRB**) [Sedgewick '08]

- překlad $(2,4)$-stromů na binární stromy



Obrázek 8.11: Překlad $(2,4)$-stromu na LLRB strom

Jiná varianta **červeno-černých** stromů:

- Každý vrchol je **černý** nebo **červený**
- Kořen a (virtuální) listy jsou **černé**
- Pokud je vrchol **červený**, oba jeho synové jsou **černí**
- Pro lib. vrchol v platí, že každá cesta z v do listu má stejný # **černých** vrcholů.