

Datové struktury

Líné vyvažování stromů

Uvažujme binární vyhledávací strom, který je na počátku vyvážený a na kterém chceme vykonávat jen operace `SEARCH` a `DELETE`.

- ▶ `DELETE` je zbytečně komplikované: Co takhle jeho složitost odložit?

Líné vyvažování stromů 2

Uvažujme nyní binární vyhledávací strom, který je na počátku prázdný a na kterém chceme vykonávat jen operace SEARCH a INSERT.

- ▶ Kdy o stromu řekneme, že je vyvážený?
- ▶ Jak těžké je vyvážit nevyvážený strom?
- ▶ Jak vyvažovat po INSERTu tak, aby to nebylo moc často?

Scapegoat tree (scapegoat – obětní beránek)

- ▶ kombinace dvou předchozích nápadů
- ▶ zjednodušení: maximálně jeden podstrom se musí vyvažovat po INSERTu (to je *scapegoat*)
- ▶ další zjednodušení: nemusíme si nic pamatovat v uzlech, stačí tři čísla pro celý strom, tj. extra paměť je pouze $\mathcal{O}(1)$

Datové struktury

Soubor řetězců

Chceme datovou strukturu, která udržuje soubor řetězců a podporuje následující operace:

- ▶ $\text{NEWSTRING}(a)$ – vytvoří nový řetězec o jednom znaku
- ▶ $\text{CONCAT}(S, T)$ – odstraní řetězce S a T ze souboru a vloží řetězec $S \cdot T$
- ▶ $\text{REVERSE}(S)$ – odstraní řetězec S ze souboru a vloží jeho zrcadlový obraz
- ▶ $\text{LOOKUP}(S, k)$ – vrátí k tý znak z řetězce S

Chceme, aby CONCAT měla amortizovanou složitost $\mathcal{O}(\log n)$ a ostatní operace měly v nejhorším případě složitost $\mathcal{O}(1)$.

Datové struktury

Splay tree

- ▶ vyvažování pomocí rotací
- ▶ *splay* (rozevření) – posunutí vrcholu až do kořene pomocí rotací (cena: hloubka vrcholu)

Operace SEARCH, INSERT, DELETE používají *splay*.
Jaká je jejich amortizovaná složitost?