

Datové struktury I

1. přednáška: Úvod do amortizované složitosti

Jirka Fink

<https://ktiml.mff.cuni.cz/~fink/>

Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

Zimní semestr 2026/27

Licence: Creative Commons BY-NC-SA 4.0

Kontakt

E-mail fink@ktiml.mff.cuni.cz

Homepage <https://ktiml.mff.cuni.cz/~fink/>

Konzultace Individuální domluva

Kontakt

E-mail fink@ktiml.mff.cuni.cz

Homepage <https://ktiml.mff.cuni.cz/~fink/>

Konzultace Individuální domluva

Cíle předmětu

- Naučit se navrhovat a analyzovat netriviální datové struktury
- Porozumět jejich chování — jak asymptoticky, tak na reálném počítači
- Zajímá nás nejen chování v nejhorším případě, ale i průměrně/amortizovaně
- Nebudujeme obecnou teorii všech DS ani neprobíráme všechny varianty DS, ale ukazujeme na příkladech různé postupy a principy

- M. Mareš: Lecture notes on data structures, 2019.
- M. Mareš, T. Valla: Průvodce labyrintem algoritmů, CZ.NIC, 2017.
- A. Koubková, V. Koubek: Datové struktury I. MATFYZPRESS, Praha 2011.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms. MIT Press, 2009
- K. Mehlhorn: Data Structures and Algorithms I: Sorting and Searching. Springer-Verlag, Berlin, 1984
- D. P. Mehta, S. Sahni eds.: Handbook of Data Structures and Applications. Chapman & Hall/CRC, Computer and Information Series, 2005
- E. Demaine: Cache-Oblivious Algorithms and Data Structures. 2002.
- R. Pagh: Cuckoo Hashing for Undergraduates. Lecture note, 2006.
- M. Thorup: High Speed Hashing for Integers and Strings. Lecture notes, 2014.
- M. Thorup: String hashing for linear probing (Sections 5.1-5.4). In Proc. 20th SODA, 655-664, 2009.
- Záznamy z přednášek Martina Mareše, Pavla Veselého a Petra Gregora (na SISu)

- 1 Amortizovaná složitost: binární čítač, dynamické pole
- 2 Stromy: $BB[\alpha]$ -stromy, splay stromy, (a,b) -stromy
- 3 Paměťová hierarchie: transpozice matic, merge sort
- 4 Hešování: volba hešovací funkce a řešení kolizí
- 5 Bloom filtry
- 6 Suffixové pole
- 7 Geometrické datové struktury: intervalové stromy
- 8 Paralelní datové struktury nepotřebující zámky

Jaké datové struktury znáte?

Jaké datové struktury znáte?

- Spojový seznam
- Pole
- Fronta, zásobník
- Vyhledávací stromy (např. AVL-stromy)
- Hešovací tabulky
- Binární halda

Popis

- Paměť je rozdělena do bloků, které jsou indexovány čísla 1, 2, ...
- V každém bloku je uloženo přirozené číslo velikosti nejvýše $poly(n)$
- Význam hodnoty v bloku: klíč nebo hodnota prvku, adresa a další pomocné proměnné
- Instrukční sada: +, -, *, celočíselné dělení a modulo
- Argumenty instrukcí: konstanta, číslo bloku, blok obsahující číslo bloku

Popis

- Paměť je rozdělena do bloků, které jsou indexovány čísly 1, 2, ...
- V každém bloku je uloženo přirozené číslo velikosti nejvýše $poly(n)$
- Význam hodnoty v bloku: klíč nebo hodnota prvku, adresa a další pomocné proměnné
- Instrukční sada: +, -, *, celočíselné dělení a modulo
- Argumenty instrukcí: konstanta, číslo bloku, blok obsahující číslo bloku

Složitost

- Paměťová: největší index použitého bloku
- Časová: počet vykonaných instrukcí

Motivace

- Uvažujme datovou strukturu, která zvládá nějakou operaci většinou velmi rychle
- Ale občas potřebuje reorganizovat svoji vnitřní strukturu, což operaci v těchto výjimečných případech značně zpomaluje
- Tudíž je časová složitost v nejhorším případě velmi špatná
- Představme si, že naše datová struktura je použita v nějakém algoritmu, který operaci zavolá mnohokrát
- V této situaci složitost algoritmu ovlivňuje celkový čas mnoha operací, nikoliv složitost operace v nejhorším případě
- Cíl: Chceme zjistit „průměrnou“ dobu výpočtu operace v rámci posloupnosti, případně celkovou složitost posloupnosti operací

Motivace

- Uvažujme datovou strukturu, která zvládá nějakou operaci většinou velmi rychle
- Ale občas potřebuje reorganizovat svoji vnitřní strukturu, což operaci v těchto výjimečných případech značně zpomaluje
- Tudíž je časová složitost v nejhorším případě velmi špatná
- Představme si, že naše datová struktura je použita v nějakém algoritmu, který operaci zavolá mnohokrát
- V této situaci složitost algoritmu ovlivňuje celkový čas mnoha operací, nikoliv složitost operace v nejhorším případě
- Cíl: Chceme zjistit „průměrnou“ dobu výpočtu operace v rámci posloupnosti, případně celkovou složitost posloupnosti operací

Metody výpočtu amortizované složitosti

- Agregovaná metoda
- Účetní metoda
- Potenciální metoda

Binární čítač

- Máme n -bitový čítač inicializovaný libovolnou hodnotou
- Při operaci INCREMENT se poslední nulový bit změní na 1 a všechny následující jedničkové bity se změní na 0
- Počet změněných bitů v nejhorším případě je n
- Kolik bitů se změní při k operacích INCREMENT?

Binární čítač

- Máme n -bitový čítač inicializovaný libovolnou hodnotou
- Při operaci INCREMENT se poslední nulový bit změní na 1 a všechny následující jedničkové bity se změní na 0
- Počet změněných bitů v nejhorším případě je n
- Kolik bitů se změní při k operacích INCREMENT?

Agregovaná metoda

- Poslední bit se změní při každé operaci — tedy k -krát
- Předposlední bit se změní při každé druhé operaci — nejvýše $\lceil k/2 \rceil$ -krát
- i -tý bit od konce se změní každých 2^i operací — nejvýše $\lceil k/2^i \rceil$ -krát

Binární čítač

- Máme n -bitový čítač inicializovaný libovolnou hodnotou
- Při operaci INCREMENT se poslední nulový bit změní na 1 a všechny následující jedničkové bity se změní na 0
- Počet změněných bitů v nejhorším případě je n
- Kolik bitů se změní při k operacích INCREMENT?

Agregovaná metoda

- Poslední bit se změní při každé operaci — tedy k -krát
- Předposlední bit se změní při každé druhé operaci — nejvýše $\lceil k/2 \rceil$ -krát
- i -tý bit od konce se změní každých 2^i operací — nejvýše $\lceil k/2^i \rceil$ -krát
- Celkový počet změn bitů je nejvýše $n + 2k$

Binární čítač

- Máme n -bitový čítač inicializovaný libovolnou hodnotou
- Při operaci INCREMENT se poslední nulový bit změní na 1 a všechny následující jedničkové bity se změní na 0
- Počet změněných bitů v nejhorším případě je n
- Kolik bitů se změní při k operacích INCREMENT?

Agregovaná metoda

- Poslední bit se změní při každé operaci — tedy k -krát
- Předposlední bit se změní při každé druhé operaci — nejvýše $\lceil k/2 \rceil$ -krát
- i -tý bit od konce se změní každých 2^i operací — nejvýše $\lceil k/2^i \rceil$ -krát
- Celkový počet změn bitů je nejvýše $n + 2k$
- Časová složitost k operací INCREMENT nad n -bitovým čítačem je $\mathcal{O}(n + k)$
- Amortizovaná složitost na jednu operaci je $\mathcal{O}(1)$

Účetní metoda

- Změna jednoho bitu stojí jeden žeton a na každou operaci dostaneme dva žetony
- Invariant: U každého jedničkového bitu si uschováme jeden žeton
- Při inkrementu máme vynulování jedničkových bitů předplaceno
- Oba žetony poskytnuté k vykonání operace využijeme na jedinou změnu nulového bitu na jedničku a předplacení jeho vynulování

Účetní metoda

- Změna jednoho bitu stojí jeden žeton a na každou operaci dostaneme dva žetony
- Invariant: U každého jedničkového bitu si uschováme jeden žeton
- Při inkrementu máme vynulování jedničkových bitů předplaceno
- Oba žetony poskytnuté k vykonání operace využijeme na jedinou změnu nulového bitu na jedničku a předplacení jeho vynulování
- Na začátku potřebujeme dostat nejvýše n žetonů
- Celkově dostaneme na k operací $n + 2k$ žetonů
- Amortizovaný počet změněných bitů při jedné operaci je $\mathcal{O}(1)$

Definice

Potenciál Φ je funkce, která každý stav datové struktury ohodnotí nezáporným reálným číslem. Operace nad datovou strukturou má amortizovanou složitost A , jestliže vykonání libovolné operace splňuje

$$T \leq A + (\Phi(S) - \Phi(S')) ,$$

kde T je skutečný čas nutný k vykonání operace, S je stav před jejím vykonáním a S' je stav po vykonání operace.

Definice

Potenciál Φ je funkce, která každý stav datové struktury ohodnotí nezáporným reálným číslem. Operace nad datovou strukturou má amortizovanou složitost A , jestliže vykonání libovolné operace splňuje

$$T \leq A + (\Phi(S) - \Phi(S')),$$

kde T je skutečný čas nutný k vykonání operace, S je stav před jejím vykonáním a S' je stav po vykonání operace.

Intuice

- Potenciál je banka, do které ukládáme čas (peníze) pro budoucí operace
- Na každou operaci dostaneme A peněz
- Je-li operace rychlá ($T < A$), tak do banky můžeme přidat $A - T$ peněz
- Pomalé operace naopak musíme doplatit z banky
- Je-li počet operací k dostatečně velký, tak celková časová složitost je $\mathcal{O}(kA)$

Definice

Potenciál Φ je funkce, která každý stav datové struktury ohodnotí nezáporným reálným číslem. Operace nad datovou strukturou má amortizovanou složitost A , jestliže vykonání libovolné operace splňuje

$$T \leq A + (\Phi(S) - \Phi(S')),$$

kde T je skutečný čas nutný k vykonání operace, S je stav před jejím vykonáním a S' je stav po vykonání operace.

Intuice

- Potenciál je banka, do které ukládáme čas (peníze) pro budoucí operace
- Na každou operaci dostaneme A peněz
- Je-li operace rychlá ($T < A$), tak do banky můžeme přidat $A - T$ peněz
- Pomalé operace naopak musíme doplatit z banky
- Je-li počet operací k dostatečně velký, tak celková časová složitost je $\mathcal{O}(kA)$

Zjednodušený zápis

$$\Phi := \Phi(S) \text{ a } \Phi' := \Phi(S')$$

Potenciální metoda

- Potenciál nulového bitu je 0 a potenciál jedničkového bitu je 1
- Potenciál čítače je součet potenciálů všech bitů
- Potenciál po provedení j -té operace označme Φ_j
- Skutečný počet změněných bitů při j -té operaci označme T_j
- Chceme spočítat amortizovaný počet změněných bitů, který označíme A

Potenciální metoda

- Potenciál nulového bitu je 0 a potenciál jedničkového bitu je 1
- Potenciál čítače je součet potenciálů všech bitů
- Potenciál po provedení j -té operace označme Φ_j
- Skutečný počet změněných bitů při j -té operaci označme T_j
- Chceme spočítat amortizovaný počet změněných bitů, který označíme A
- Pro každou operaci j musí platit $T_j \leq A + (\Phi_{j-1} - \Phi_j)$ pro libovolnou operaci j

Potenciální metoda

- Potenciál nulového bitu je 0 a potenciál jedničkového bitu je 1
- Potenciál čítače je součet potenciálů všech bitů
- Potenciál po provedení j -té operace označme Φ_j
- Skutečný počet změněných bitů při j -té operaci označme T_j
- Chceme spočítat amortizovaný počet změněných bitů, který označíme A
- Pro každou operaci j musí platit $T_j \leq A + (\Phi_{j-1} - \Phi_j)$ pro libovolnou operaci j
- Zvolíme $A = 2$

Potenciální metoda

- Potenciál nulového bitu je 0 a potenciál jedničkového bitu je 1
- Potenciál čítače je součet potenciálů všech bitů
- Potenciál po provedení j -té operace označme Φ_j
- Skutečný počet změněných bitů při j -té operaci označme T_j
- Chceme spočítat amortizovaný počet změněných bitů, který označíme A
- Pro každou operaci j musí platit $T_j \leq A + (\Phi_{j-1} - \Phi_j)$ pro libovolnou operaci j
- Zvolíme $A = 2$
- Celkový počet změněných bitů při k operacích je

$$\sum_{j=1}^k T_j \leq \sum_{j=1}^k (2 + \Phi_{j-1} - \Phi_j) \leq 2k + \Phi_0 - \Phi_k \leq 2k + n,$$

protože $0 \leq \Phi_j \leq n$

Dynamické pole

- Máme zásobník (prvky přidáváme na konec a z konce i mažeme)
- Počet prvků označíme n a velikost pole p
- Jestliže $p = n$ a máme přidat další prvek, tak velikost pole zdvojnásobíme
- Jestliže $p = 4n$ a máme smazat prvek, tak velikost pole zmenšíme na polovinu

Dynamické pole

- Máme zásobník (prvky přidáváme na konec a z konce i mažeme)
- Počet prvků označíme n a velikost pole p
- Jestliže $p = n$ a máme přidat další prvek, tak velikost pole zdvojnásobíme
- Jestliže $p = 4n$ a máme smazat prvek, tak velikost pole zmenšíme na polovinu

Intuitivní přístup

- Zkopírování celého pole trvá $\mathcal{O}(n)$
- Jestliže po realokaci pole máme n prvků, pak další realokace nastane nejdříve po $n/2$ operacích INSERT nebo DELETE
- Amortizovaná složitost je $\mathcal{O}(1)$

Dynamické pole

- Máme zásobník (prvky přidáváme na konec a z konce i mažeme)
- Počet prvků označíme n a velikost pole p
- Jestliže $p = n$ a máme přidat další prvek, tak velikost pole zdvojnásobíme
- Jestliže $p = 4n$ a máme smazat prvek, tak velikost pole zmenšíme na polovinu

Intuitivní přístup

- Zkopírování celého pole trvá $\mathcal{O}(n)$
- Jestliže po realokaci pole máme n prvků, pak další realokace nastane nejdříve po $n/2$ operacích INSERT nebo DELETE
- Amortizovaná složitost je $\mathcal{O}(1)$

Agregovaná metoda: Celkový čas

- Nechť k_i je počet operací mezi $(i - 1)$ a i -tou realokací $\Rightarrow \sum_i k_i = k$
- Při první realokaci se kopíruje nejvýše $n_0 + k_1$ prvků, kde n_0 je počáteční počet
- Při i -té realokaci se kopíruje nejvýše $2k_i$ prvků, kde $i \geq 2$
- Celkový počet zkopírovaných prvků je nejvýše $n_0 + k_1 + \sum_{i \geq 2} 2k_i \leq n_0 + 2k$

Potenciální metoda

- Uvažujme potenciál

$$\Phi = \begin{cases} 0 & \text{pokud } p = 2n \\ n & \text{pokud } p = n \\ n & \text{pokud } p = 4n \end{cases}$$

a tyto tři body rozšíříme po částech lineární funkcí

Potenciální metoda

- Uvažujme potenciál

$$\Phi = \begin{cases} 0 & \text{pokud } p = 2n \\ n & \text{pokud } p = n \\ n & \text{pokud } p = 4n \end{cases}$$

a tyto tři body rozšíříme po částech lineární funkcí

- Explicitně

$$\Phi = \begin{cases} 2n - p & \text{pokud } p \leq 2n \\ p/2 - n & \text{pokud } p \geq 2n \end{cases}$$

Potenciální metoda

- Uvažujme potenciál

$$\Phi = \begin{cases} 0 & \text{pokud } p = 2n \\ n & \text{pokud } p = n \\ n & \text{pokud } p = 4n \end{cases}$$

a tyto tři body rozšíříme po částech lineární funkcí

- Explicitně

$$\Phi = \begin{cases} 2n - p & \text{pokud } p \leq 2n \\ p/2 - n & \text{pokud } p \geq 2n \end{cases}$$

- Změna potenciálu při jedné operaci bez realokace je $\Phi' - \Phi \leq 2$

Potenciální metoda

- Uvažujme potenciál

$$\Phi = \begin{cases} 0 & \text{pokud } p = 2n \\ n & \text{pokud } p = n \\ n & \text{pokud } p = 4n \end{cases}$$

a tyto tři body rozšíříme po částech lineární funkcí

- Explicitně

$$\Phi = \begin{cases} 2n - p & \text{pokud } p \leq 2n \\ p/2 - n & \text{pokud } p \geq 2n \end{cases}$$

- Změna potenciálu při jedné operaci bez realokace je $\Phi' - \Phi \leq 2$
- Skutečný počet zkopírovaných prvků T vždy splňuje $T + (\Phi' - \Phi) \leq 2$

Potenciální metoda

- Uvažujme potenciál

$$\Phi = \begin{cases} 0 & \text{pokud } p = 2n \\ n & \text{pokud } p = n \\ n & \text{pokud } p = 4n \end{cases}$$

a tyto tři body rozšíříme po částech lineární funkcí

- Explicitně

$$\Phi = \begin{cases} 2n - p & \text{pokud } p \leq 2n \\ p/2 - n & \text{pokud } p \geq 2n \end{cases}$$

- Změna potenciálu při jedné operaci bez realokace je $\Phi' - \Phi \leq 2$
- Skutečný počet zkopírovaných prvků T vždy splňuje $T + (\Phi' - \Phi) \leq 2$
- Celkový počet zkopírovaných prvků při k operacích je nejvýše $2k + \Phi_0 - \Phi_k \leq 2k + n_0$

Potenciální metoda

- Uvažujme potenciál

$$\Phi = \begin{cases} 0 & \text{pokud } p = 2n \\ n & \text{pokud } p = n \\ n & \text{pokud } p = 4n \end{cases}$$

a tyto tři body rozšíříme po částech lineární funkcí

- Explicitně

$$\Phi = \begin{cases} 2n - p & \text{pokud } p \leq 2n \\ p/2 - n & \text{pokud } p \geq 2n \end{cases}$$

- Změna potenciálu při jedné operaci bez realokace je $\Phi' - \Phi \leq 2$
- Skutečný počet zkopírovaných prvků T vždy splňuje $T + (\Phi' - \Phi) \leq 2$
- Celkový počet zkopírovaných prvků při k operacích je nejvýše $2k + \Phi_0 - \Phi_k \leq 2k + n_0$
- Celkový čas k operací je $\mathcal{O}(n_0 + k)$
- Amortizovaný čas jedné operace je $\mathcal{O}(1)$

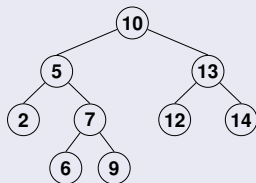
Definice

- Binární strom (každý vrchol obsahuje nejvýše dva syny)
- Klíč v každém vnitřním vrcholu je větší než všechny klíče v levém podstromu a menší než všechny klíče v pravém podstromu
- Prvky mohou být uloženy pouze v listech nebo též ve vnitřních vrcholech (u každého klíče je uložena i hodnota)

Definice

- Binární strom (každý vrchol obsahuje nejvýše dva syny)
- Klíč v každém vnitřním vrcholu je větší než všechny klíče v levém podstromu a menší než všechny klíče v pravém podstromu
- Prvky mohou být uloženy pouze v listech nebo též ve vnitřních vrcholech (u každého klíče je uložena i hodnota)

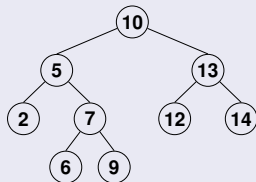
Příklad



Definice

- Binární strom (každý vrchol obsahuje nejvýše dva syny)
- Klíč v každém vnitřním vrcholu je větší než všechny klíče v levém podstromu a menší než všechny klíče v pravém podstromu
- Prvky mohou být uloženy pouze v listech nebo též ve vnitřních vrcholech (u každého klíče je uložena i hodnota)

Příklad



Složitost

- Paměť: $\mathcal{O}(n)$
- Časová složitost operace Find je lineární ve výšce stromu
- Výška stromu může být až $n - 1$, a proto musíme stromy vyvažovat

BB[α]-strom (Nievergelt, Reingold, 1973)

- Binární vyhledávací strom
- Počet vrcholů v podstromu vrcholu u označme s_u
- Pro každý vrchol u platí, že podstromy obou synů u musí mít nejvýše αs_u vrcholů

BB[α]-strom (Nievergelt, Reingold, 1973)

- Binární vyhledávací strom
- Počet vrcholů v podstromu vrcholu u označme s_u
- Pro každý vrchol u platí, že podstromy obou synů u musí mít nejvýše αs_u vrcholů
- Zřejmě musí platit $\frac{1}{2} < \alpha < 1$

BB[α]-strom (Nievergelt, Reingold, 1973)

- Binární vyhledávací strom
- Počet vrcholů v podstromu vrcholu u označme s_u
- Pro každý vrchol u platí, že podstromy obou synů u musí mít nejvýše αs_u vrcholů
- Zřejmě musí platit $\frac{1}{2} < \alpha < 1$

Výška BB[α]-stromu

- Podstromy všech vnuků kořene mají nejvýše $\alpha^2 n$ vrcholů
- Podstromy všech vrcholů v i -té vrstvě mají nejvýše $\alpha^i n$ vrcholů

BB[α]-strom (Nievergelt, Reingold, 1973)

- Binární vyhledávací strom
- Počet vrcholů v podstromu vrcholu u označme s_u
- Pro každý vrchol u platí, že podstromy obou synů u musí mít nejvýše αs_u vrcholů
- Zřejmě musí platit $\frac{1}{2} < \alpha < 1$

Výška BB[α]-stromu

- Podstromy všech vnuků kořene mají nejvýše $\alpha^2 n$ vrcholů
- Podstromy všech vrcholů v i -té vrstvě mají nejvýše $\alpha^i n$ vrcholů
- $\alpha^i n \geq 1$ jen pro $i \leq \log_{\frac{1}{\alpha}}(n)$
- Výška BB[α]-stromu je $\Theta(\log n)$

BB[α]-strom (Nievergelt, Reingold, 1973)

- Binární vyhledávací strom
- Počet vrcholů v podstromu vrcholu u označme s_u
- Pro každý vrchol u platí, že podstromy obou synů u musí mít nejvýše αs_u vrcholů
- Zřejmě musí platit $\frac{1}{2} < \alpha < 1$

Výška BB[α]-stromu

- Podstromy všech vnuků kořene mají nejvýše $\alpha^2 n$ vrcholů
- Podstromy všech vrcholů v i -té vrstvě mají nejvýše $\alpha^i n$ vrcholů
- $\alpha^i n \geq 1$ jen pro $i \leq \log_{\frac{1}{\alpha}}(n)$
- Výška BB[α]-stromu je $\Theta(\log n)$

Operace BUILD: Vytvoření BB[α]-stromu ze setříděného pole

- Prostřední prvek dáme do kořene
- Rekurzivně vytvoříme oba podstromy
- Časová složitost je $\mathcal{O}(n)$

Operace INSERT (DELETE je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)

Operace INSERT (DELETE je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací BUILD (složitost: amortizovaná analýza)

Operace INSERT (DELETE je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací BUILD (složitost: amortizovaná analýza)

Amortizovaná složitost operací INSERT a DELETE: Agregovaná metoda

- Jestliže podstrom vrcholu u po provedení operace BUILD má s_u vrcholů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u (cvičení)

Operace INSERT (DELETE je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací BUILD (složitost: amortizovaná analýza)

Amortizovaná složitost operací INSERT a DELETE: Agregovaná metoda

- Jestliže podstrom vrcholu u po provedení operace BUILD má s_u vrcholů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u (cvičení)
- Rebuild podstromu vrcholu u trvá $\mathcal{O}(s_u)$
- Amortizovaný čas vyvažování jednoho vrcholu je $\mathcal{O}(1)$

Operace INSERT (DELETE je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací BUILD (složitost: amortizovaná analýza)

Amortizovaná složitost operací INSERT a DELETE: Agregovaná metoda

- Jestliže podstrom vrcholu u po provedení operace BUILD má s_u vrcholů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u (cvičení)
- Rebuild podstromu vrcholu u trvá $\mathcal{O}(s_u)$
- Amortizovaný čas vyvažování jednoho vrcholu je $\mathcal{O}(1)$
- Při jedné operaci INSERT/DELETE se prvek přidá/smaže v $\Theta(\log n)$ podstromech

Operace INSERT (DELETE je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací BUILD (složitost: amortizovaná analýza)

Amortizovaná složitost operací INSERT a DELETE: Agregovaná metoda

- Jestliže podstrom vrcholu u po provedení operace BUILD má s_u vrcholů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u (cvičení)
- Rebuild podstromu vrcholu u trvá $\mathcal{O}(s_u)$
- Amortizovaný čas vyvažování jednoho vrcholu je $\mathcal{O}(1)$
- Při jedné operaci INSERT/DELETE se prvek přidá/smaže v $\Theta(\log n)$ podstromech
- Amortizovaný čas vyvažování při jedné operaci INSERT nebo DELETE je $\mathcal{O}(\log n)$

Operace INSERT (DELETE je analogický)

- Najít list pro nový prvek a uložit do něho nový prvek (složitost: $\mathcal{O}(\log n)$)
- Jestliže některý vrchol porušuje vyvažovací podmínku, tak celý jeho podstrom znovu vytvoříme operací BUILD (složitost: amortizovaná analýza)

Amortizovaná složitost operací INSERT a DELETE: Agregovaná metoda

- Jestliže podstrom vrcholu u po provedení operace BUILD má s_u vrcholů, pak další porušení vyvažovací podmínky pro vrchol u nastane nejdříve po $\Omega(s_u)$ přidání/smazání prvků v podstromu vrcholu u (cvičení)
- Rebuild podstromu vrcholu u trvá $\mathcal{O}(s_u)$
- Amortizovaný čas vyvažování jednoho vrcholu je $\mathcal{O}(1)$
- Při jedné operaci INSERT/DELETE se prvek přidá/smaže v $\Theta(\log n)$ podstromech
- Amortizovaný čas vyvažování při jedné operaci INSERT nebo DELETE je $\mathcal{O}(\log n)$
- Celkový čas k operací je $\mathcal{O}((k + n) \log n)$

Amortizovaná složitost operací INSERT a DELETE: Potenciální metoda

- V této analýze uvažujeme jen čas na postavení podstromu, zbytek trvá $\mathcal{O}(\log n)$

Amortizovaná složitost operací INSERT a DELETE: Potenciální metoda

- V této analýze uvažujeme jen čas na postavení podstromu, zbytek trvá $\mathcal{O}(\log n)$
- Potenciál vrcholu u definován

$$\Phi(u) = \begin{cases} 0 & \text{pokud } |s_{l(u)} - s_{r(u)}| \leq 1 \\ |s_{l(u)} - s_{r(u)}| & \text{jinak,} \end{cases}$$

kde $l(u)$ a $r(u)$ jsou levý a pravý synové u .

- Potenciál BB[α]-stromu Φ je součet potenciálů vrcholů

Amortizovaná složitost operací INSERT a DELETE: Potenciální metoda

- V této analýze uvažujeme jen čas na postavení podstromu, zbytek trvá $\mathcal{O}(\log n)$
- Potenciál vrcholu u definován

$$\Phi(u) = \begin{cases} 0 & \text{pokud } |s_{l(u)} - s_{r(u)}| \leq 1 \\ |s_{l(u)} - s_{r(u)}| & \text{jinak,} \end{cases}$$

kde $l(u)$ a $r(u)$ jsou levý a pravý synové u .

- Potenciál BB[α]-stromu Φ je součet potenciálů vrcholů
- Při vložení/smazání prvku se potenciál $\Phi(u)$ jednoho vrcholu zvýší nejvýše o 2
- Pokud nenastane Rebuild, pak se potenciál stromu zvýší nejvýše o $\mathcal{O}(\log(n))$

Amortizovaná složitost operací INSERT a DELETE: Potenciální metoda

- V této analýze uvažujeme jen čas na postavení podstromu, zbytek trvá $\mathcal{O}(\log n)$
- Potenciál vrcholu u definován

$$\Phi(u) = \begin{cases} 0 & \text{pokud } |s_{l(u)} - s_{r(u)}| \leq 1 \\ |s_{l(u)} - s_{r(u)}| & \text{jinak,} \end{cases}$$

kde $l(u)$ a $r(u)$ jsou levý a pravý synové u .

- Potenciál BB[α]-stromu Φ je součet potenciálů vrcholů
- Při vložení/smazání prvku se potenciál $\Phi(u)$ jednoho vrcholu zvýší nejvýše o 2
- Pokud nenastane Rebuild, pak se potenciál stromu zvýší nejvýše o $\mathcal{O}(\log(n))$
- Pokud nastane Rebuild vrcholu u , pak $\Phi(u) \geq \alpha s_u - (1 - \alpha)s_u \geq (2\alpha - 1)s_u$
- Po rekonstrukci mají všechny vrcholy v podstromu u nulový potenciál
- Při rekonstrukci poklesne potenciál Φ alespoň o $\Omega(s_u)$, což zaplatí čas na rekonstrukci

Amortizovaná složitost operací INSERT a DELETE: Potenciální metoda

- V této analýze uvažujeme jen čas na postavení podstromu, zbytek trvá $\mathcal{O}(\log n)$
- Potenciál vrcholu u definován

$$\Phi(u) = \begin{cases} 0 & \text{pokud } |s_{l(u)} - s_{r(u)}| \leq 1 \\ |s_{l(u)} - s_{r(u)}| & \text{jinak,} \end{cases}$$

kde $l(u)$ a $r(u)$ jsou levý a pravý synové u .

- Potenciál BB[α]-stromu Φ je součet potenciálů vrcholů
- Při vložení/smazání prvku se potenciál $\Phi(u)$ jednoho vrcholu zvýší nejvýše o 2
- Pokud nenastane Rebuild, pak se potenciál stromu zvýší nejvýše o $\mathcal{O}(\log(n))$
- Pokud nastane Rebuild vrcholu u , pak $\Phi(u) \geq \alpha s_u - (1 - \alpha)s_u \geq (2\alpha - 1)s_u$
- Po rekonstrukci mají všechny vrcholy v podstromu u nulový potenciál
- Při rekonstrukci poklesne potenciál Φ alespoň o $\Omega(s_u)$, což zaplatí čas na rekonstrukci
- Dále platí $0 \leq \Phi \leq hn = \mathcal{O}(n \log n)$, kde h je výška stromu
- Celkový čas na k operací INSERT nebo DELETE je $\mathcal{O}((k + n) \log n)$

- Samovyvažovací stromy (Splay stromy)