

# Datové struktury I

## 3. přednáška: (a,b)-stromy

Jirka Fink

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

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

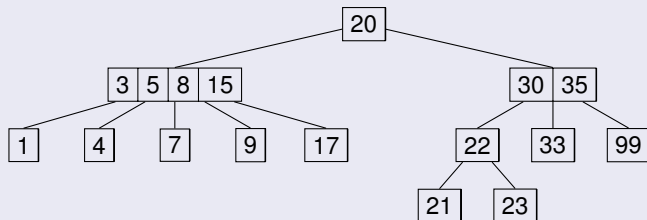
Zimní semestr 2024/25

Licence: Creative Commons BY-NC-SA 4.0

## Popis

- Vnitřní vrcholy mají libovolný počet synů (typicky alespoň dva)
- Vnitřní vrchol s  $k$  syny má  $k - 1$  setříděných klíčů
- V každém vnitřním vrcholu je  $i$ -tý klíč větší než všechny klíče v  $i$ -tém podstromu a menší než všechny klíče v  $(i + 1)$  podstromu pro všechny klíče  $i$
- Prvky mohou být uloženy pouze v listech nebo též ve vnitřních vrcholech (u každého klíče je uložena i hodnota)
- Pokud máme hodnoty jen v listech, pak  $i$ -tý klíč vrcholu může být roven největšímu klíči v  $i$ -tém podstromu

## Příklad: Hodnoty jsou ve všech vrcholech



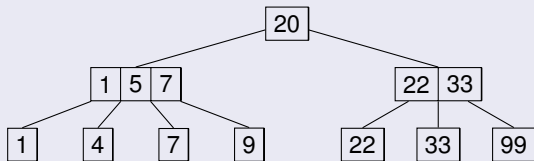
## Definice

$a, b$  jsou celá čísla splňující  $a \geq 2$  a  $b \geq 2a - 1$

- (a,b)-strom je vyhledávací strom
- Všechny vnitřní vrcholy kromě kořene mají alespoň  $a$  synů a nejvýše  $b$  synů
- Kořen má nejvýše  $b$  synů
- Všechny listy jsou ve stejné výšce

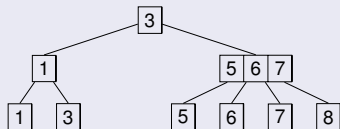
Pro zjednodušení uvažujeme, že prvky jsou jen v listech ①

## Příklad: (2,4)-strom

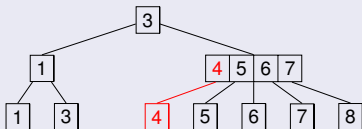


- 1 V domácím úkolu jsou prvky i ve vnitřních vrcholech, tak si rozmyslete, jak se změní operace Insert a Delete.

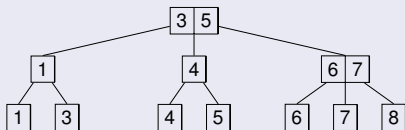
Vložte prvek s klíčem 4 do následujícího (2,4)-stromu



Nejprve najdeme správného otce, jemuž přidáme nový list



Opakovaně rozdělujeme vrchol na dva

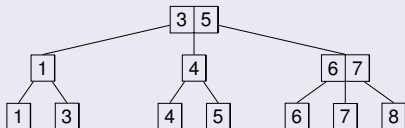


## Algoritmus

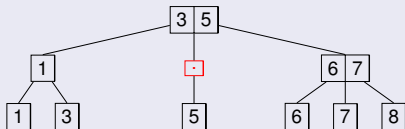
```
1 Najít otce  $v$ , kterému nový prvek patří
2 Přidat nový list do  $v$ 
3 while  $\text{deg}(v) > b$  do
    # Najdeme otce  $u$  vrcholu  $v$ 
4   if  $v$  je kořen then
5     | Vytvořit nový kořen  $u$  s jediným synem  $v$ 
6   else
7     |  $u \leftarrow$  otec  $v$ 
    # Rozdělíme vrchol  $v$  na  $v$  a  $v'$ 
8   Vytvořit nového syna  $v'$  otci  $u$  a umístit jej vpravo vedle  $v$ 
9   Přesunout nejpravějších  $\lfloor (b+1)/2 \rfloor$  synů vrcholu  $v$  do  $v'$ 
10  Přesunout nejpravějších  $\lfloor (b+1)/2 \rfloor - 1$  klíčů vrcholu  $v$  do  $v'$ 
11  Přesunout poslední klíč vrcholu  $v$  do  $u$ 
12   $v \leftarrow u$ 
```

Musíme ještě dokázat, že po provedení všech operací doopravdy dostaneme  $(a,b)$ -strom. Ověříme, že rozdělené vrcholy mají alespoň  $a$  synů (ostatní požadavky jsou triviální). Rozdělovaný vrchol má na počátku právě  $b + 1$  synů a počet synů po rozdělení je  $\lfloor \frac{b+1}{2} \rfloor$  a  $\lceil \frac{b+1}{2} \rceil$ . Protože  $b \geq 2a - 1$ , počet synů po rozdělení je alespoň  $\lfloor \frac{b+1}{2} \rfloor \geq \lfloor \frac{2a-1+1}{2} \rfloor = \lfloor a \rfloor = a$ .

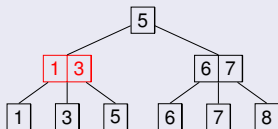
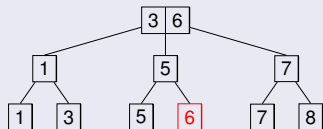
Smažte prvek s klíčem 4 z následujícího (2,4)-stromu



Nalezneme a smažeme list



Přesuneme jedno syna od bratra nebo spojíme vrchol s bratrem





## Algoritmus

```
1 Najít list  $l$  obsahující prvek s daným klíčem
2  $v \leftarrow$  otec  $l$ 
3 Smazat  $l$ 
4 while  $\text{deg}(v) < a$  &  $v$  není kořen do
5    $u \leftarrow$  sousední bratr  $v$ 
6   if  $\text{deg}(u) > a$  then
7     | Přesunout správného syna  $u$  pod  $v$  ①
8   else
9     | Přesunout všechny syny  $u$  pod  $v$  ②
10    | Smazat  $u$ 
11    | if  $v$  nemá žádného bratra then
12      | Smazat kořen (otec  $v$ ) a nastavit  $v$  jako kořen
13    | else
14      |  $v \leftarrow$  otec  $v$ 
```

- 1 Při přesunu je nutné upravit klíče ve vrcholech  $u$ ,  $v$  a jejich otci.
- 2 Vrchol  $u$  měl  $a$ , vrchol  $v$  měl  $a - 1$  synů. Po jejich sjednocení máme vrchol s  $2a - 1 \leq b$  syny.

## Cíl

- Chceme najít nejlepší hodnoty  $a, b$  minimalizující složitost
- Složitost budeme počítat v závislosti na  $a, b$  a počtu prvků  $n$

## Výška

- (a,b)-strom výšky  $d$  má alespoň  $a^{d-1}$  a nejvýše  $b^d$  listů.
- Výška (a,b)-stromu splňuje  $\log_b n \leq d \leq 1 + \log_a n$ .

## Operace FIND

- Nalezení správného syna ve vrcholu:  $\mathcal{O}(\log b)$  ①
- Celá operace:  $\mathcal{O}(\log b \cdot \log_a n) = \mathcal{O}\left(\log n \frac{\log b}{\log a}\right) = \mathcal{O}(\log n)$  pokud  $b = \text{poly}(a)$

## Operace INSERT a DELETE

- Rozdělení nebo sloučení vrcholů:  $\mathcal{O}(b)$
- Celá operace:  $\mathcal{O}(b \cdot \log_a n) = \mathcal{O}\left(\log n \frac{b}{\log a}\right)$
- Optimální volba  $a = 2, b = 3$

- 1 Teoreticky k dosažení nejlepší složitosti použijeme binární půlení, ale pokud hodnota  $b$  je malá, tak v praxi je lepší lineárně projít všechny klíče.

## Podobné datové struktury

- B-tree, B+ tree, B\* tree
- 2-4-tree, 2-3-4-tree, atd.

## Aplikace

- File systems např. Ext4, NTFS, HFS+
- Databáze

## Volba parametrů $a, b$

Hodnotu  $b$  volíme tak, aby se jeden vrchol vešel do bloku a  $a = \lceil b/2 \rceil$

- 4KB stránka, 32-bitový klíč a ukazatel  $\Rightarrow$  (256,511)-strom
- 64B řádka keše  $\Rightarrow$  (4,7)-strom

### Věta (počet modifikovaných vrcholů při vytvoření stromu operací INSERT)

Amortizovaný počet štěpení v libovolné posloupnosti operací INSERT začínající na prázdném stromu je  $\mathcal{O}(1)$ . ①

### Důkaz

- Při každém štěpení vrcholu vytvoříme nový vnitřní vrchol
- Po vytvoření má strom nejvýše  $n$  vnitřních vrcholů
- Celkový počet štěpení je nejvýše  $n$  a počet modifikací vrcholů je  $\mathcal{O}(n)$
- Amortizovaný počet modifikovaných vrcholů na jednu operaci INSERT je  $\mathcal{O}(1)$

- 1 Při jedné vyvažovací operaci (štěpení vrcholu) je počet modifikovaných vrcholů omezený konstantou (štěpený vrchol, otec a synové). Asymptoticky jsou počty modifikovaných vrcholů a vyvažovacích operací stejné.

## Věta (Huddleston, Mehlhorn, 1982)

Amortizovaný počet štěpení a slučování v libovolné posloupnosti  $k$  operací INSERT a DELETE v  $(a, 2a)$ -stromu je  $\mathcal{O}(1)$  a celkový počet je  $\mathcal{O}(n + k)$ .

## Důkaz

- Potenciál vrcholu  $u$  závisí na počtu jeho synů takto: ①

synů	$a-1$	$a$	$a+1$	...	$2a-1$	$2a$	$2a+1$
$\Phi(u)$	2	1	0	...	0	2	4

- Potenciál stromu je součet potenciálů vrcholů
- Změny potenciálu při štěpení vrcholu  $u$  s otcem  $p$  jsou:
  - $u$  s potenciálem 4 rozdělíme na dva vrcholy s potenciály 0 a 1
  - Potenciál vrcholu  $p$  se zvýší nejvýše o 2
  - Potenciál se sníží alespoň o 1, což zaplatí štěpení
- Změny potenciálu při slučování vrcholů  $u$  a  $u'$  s otcem  $p$  jsou:
  - $\Phi(u) = 2$ ,  $\Phi(u') = 1$  a sloučený vrchol má potenciál 0
  - Potenciál vrcholu  $p$  se zvýší nejvýše o 1
  - Potenciál se sníží alespoň o 2, což zaplatí slučování
- Přidání, smazání a přesun vrcholu zvýší potenciál nejvýše o 2, ale tyto operace provádíme nejvýše jednou
- Jelikož  $0 \leq \Phi \leq 4n$ , celkové snížení potenciálu při všech operacích je  $\mathcal{O}(n)$



- 1 Kořen může mít méně než  $a - 1$  synů a nemáme pro něj definovaný potenciál. Rozmyslete si, jak do důkazu doplnit práci s kořenem. Důležité je, že štěpení či slučování mimo kořen může být logaritmicky mnoho, takže je nutné je platit z potenciálu, ale u kořene nastanou jen jednou.

## Cíl

Umožnit efektní paralelizaci operací Find, Insert a Delete (předpoklad:  $b \geq 2a$ ).

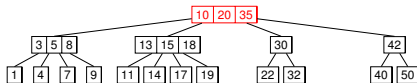
## Operace Insert

Preventivně rozdělit každý vrchol na cestě od kořene k hledanému listu s  $b$  syny na dva vrcholu.

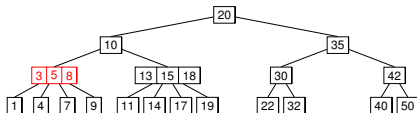
## Operace Delete

Preventivně sloučit každý vrchol na cestě od kořene k hledanému listu s  $a$  syny s bratrem nebo přesunout synovce.

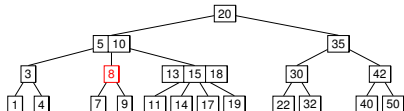
- Vložte prvek s klíčem 6 do následujícího (2,4)-stromu



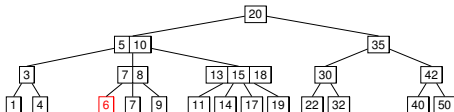
- Nejprve rozdělíme kořen



- Pak pokračujeme do levého syna, který taky rozdělíme



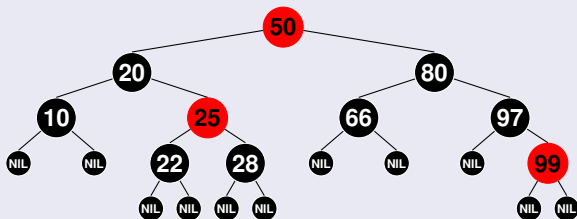
- Vrchol s klíčem 8 není třeba rozdělovat a nový klíč můžeme vložit



## Definice

- 1 Binární vyhledávací strom s prvky uloženými ve všech vrcholech
- 2 Každý vrchol je černý nebo červený
- 3 Všechny cesty od kořene do listů obsahují stejný počet černých vrcholů
- 4 Otec červeného vrcholu musí být černý
- 5 Listy jsou černé ①

## Příklad

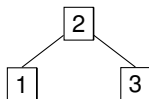
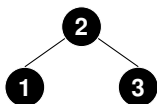


- 1 Nepovinná podmínka, která jen zjednodušuje operace. V příkladu uvažujeme, že listy jsou reprezentovány NIL/NULL ukazateli, a tedy imaginární vrcholy bez prvků. Někdy se též vyžaduje, aby kořen byl černý, ale tato podmínka není nutná, protože kořen můžeme vždy přebarvit na černo bez porušení ostatních podmínek.

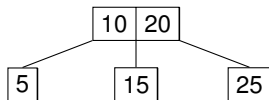
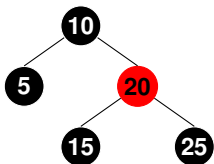
## Červeno-černé stromy: Ekvivalence s (2,4)-stromy

Každému černému vrcholu odpovídá jeden vrchol (2,4)-stromu

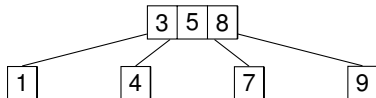
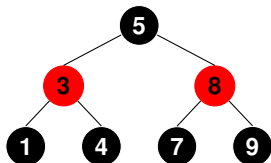
- Vrchol bez červených synů



- Vrchol s jedním červeným synem ①



- Vrchol s dvěma červenými syny



- 1 Převod mezi červeno-černými stromy a (2,4)-stromy není jednoznačný, protože vrchol (2,4)-stromu se třemi syny a prvky  $x < y$  lze převést na černý vrchol červeno-černého stromu s prvkem  $x$  a pravým červeným synem  $y$  nebo s prvkem  $y$  a levým červeným synem  $x$ .

## Vlastnosti

- Výška červeno-černého stromu je nejvýše  $2 + 2 \log_2 n$
- Časová složitost operací Find, Insert a Delete je  $\mathcal{O}(\log n)$
- Pro vkládání a mazání vrcholů existuje varianta shora dolů i zdola nahoru
- Amortizovaný počet změn stromu při balancování zdola nahoru je  $\mathcal{O}(1)$

## Aplikace

- Asociativní pole např. `std::map` and `std::set` v C++, `TreeMap` v Java
- The Completely Fair Scheduler in the Linux kernel
- Computational Geometry Data structures
- Persistentní datové struktury



## Cíl

Setřídít „skoro“ setříděné pole

## Modifikace (a,b)-stromu

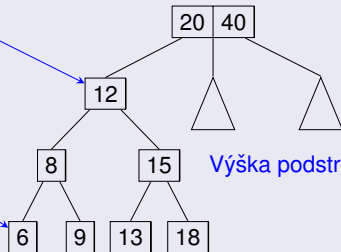
Máme uložený ukazatel na vrchol s nejmenším klíčem

## Příklad: Vložte klíč s hodnotou $x_i = 16$

- Začneme od vrcholu s nejmenším klíčem a postupujeme ke kořeni, dokud  $x_i$  nepatří podstromu aktuálního vrcholu
- V rámci tohoto podstromu spustíme operaci Insert
- Výška podstromu je  $\Theta(\log f_i)$ , kde  $f_i$  je počet klíčů menších než  $x_i$

Prvek  $x_i = 16$   
patří do tohoto  
podstromu

Nejmenší klíč



**Input:** Posloupnost  $x_1, x_2, \dots, x_n$

```
1  $T \leftarrow$  prázdný (a,b)-strom
2 for  $i \leftarrow n$  to 1 # Prvky procházíme od konce
3 do
4     # Najdeme podstrom, do kterého vložíme  $x_i$ 
5      $v \leftarrow$  list s nejmenším klíčem
6     while  $v$  není kořen a  $x_i$  je větší než nejmenší klíč v otci vrcholu  $v$  do
7         |  $v \leftarrow$  otec  $v$ 
8         | Vložíme  $x_i$  do podstromu vrcholu  $v$ 
Output: Projdeme celý strom a vypíšeme všechny klíče (in-order traversal)
```

## Nerovnost mezi aritmetickým a geometrickým průměrem

Jestliže  $a_1, \dots, a_n$  nezáporná reálná čísla, pak platí

$$\frac{\sum_{i=1}^n a_i}{n} \geq \sqrt[n]{\prod_{i=1}^n a_i}.$$

## Časová složitost

- 1 Necht  $f_i = |\{j > i; x_j < x_i\}|$  je počet klíčů menších než  $x_i$ , které již jsou ve stromu při vkládání  $x_i$
- 2 Necht  $F = |\{(i, j); i > j, x_i < x_j\}| = \sum_{i=1}^n f_i$  je počet inverzí
- 3 Složitost nalezení podstromu, do kterého  $x_i$  patří:  $\mathcal{O}(\log f_i)$
- 4 Nalezení těchto podstromů pro všechny podstromy  
 $\sum_i \log f_i = \log \prod_i f_i = n \log \sqrt[n]{\prod_i f_i} \leq n \log \frac{\sum_i f_i}{n} = n \log \frac{F}{n}$ . ①
- 5 Rozdělování vrcholů v průběhu všech operací Insert:  $\mathcal{O}(n)$
- 6 Celková složitost:  $\mathcal{O}(n + n \log(F/n))$
- 7 Složitost v nejhorším případě:  $\mathcal{O}(n \log n)$  protože  $F \leq \binom{n}{2}$
- 8 Jestliže  $F \leq n \log n$ , pak složitost je  $\mathcal{O}(n \log \log n)$  ②

- 1 Místo AG nerovnosti můžeme použít Jensenovu nerovnost, ze které přímo plyne  $\frac{\sum_i \log f_i}{n} \leq \log \frac{\sum_i f_i}{n}$ .
- 2 Tento algoritmus je bohužel efektivní jen pro „hodně skoro“ setříděné posloupnosti. Jestliže počet inverzí je  $n^{1+\epsilon}$ , pak dostáváme složitost třídění  $\mathcal{O}(n \log n)$ , kde  $\epsilon$  je libovolně malé kladné číslo.

Jak navrhovat a analyzovat algoritmy a datové struktury, aby efektivně využívali keše procesorů?