

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

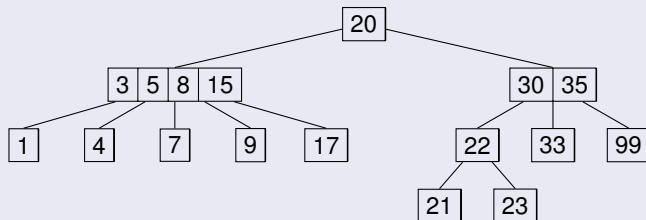
Zimní semestr 2026/27

Licence: Creative Commons BY-NC-SA 4.0

Popis

- Vnitřní vrcholy mají libovolný počet synů (typicky alespoň dva)
- Vnitřní vrchol s k syny má $k - 1$ seřazených klíčů
- V každém vnitřním vrcholu je i -tý klíč větší než všechny klíče v i -tém podstromu a menší než všechny klíče v $(i + 1)$ -ním podstromu pro všechna 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



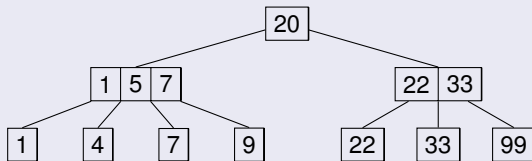
Definice

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

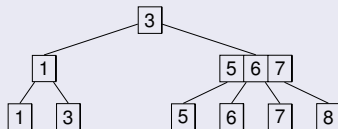
Pro zjednodušení uvažujeme, že prvky jsou jen v listech^①

Příklad: (2,4)-strom

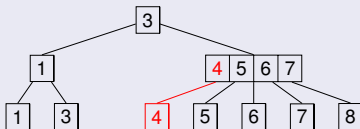


- 1 V domácím úkolu jsou prvky i ve vnitřních vrcholech, tak si rozmyslete, jak se změní operace Insert a Delete.

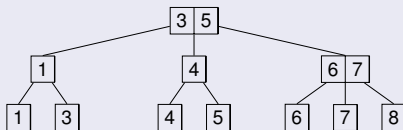
Vložte prvek s klíčem 4 do následujícího (2,4)-stromu



Nejprve najdeme správného otce, jemuž přidáme nový list



Opakovaně rozdělujeme vrchol na dva

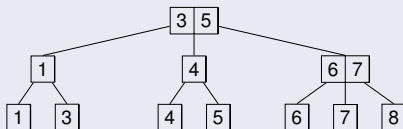


Algoritmus

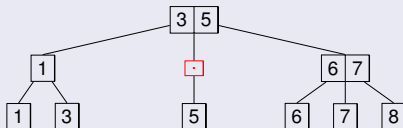
```
1 Najít otce  $v$ , kterému nový prvek patří
2 Přidat nový list do  $v$ 
3 while  $\deg(v) > b$  do
    # Najdeme otce  $u$  vrcholu  $v$ 
4    if  $v$  je kořen then
5        | Vytvořit nový kořen  $u$  s jediným synem  $v$ 
6    else
7        |  $u \leftarrow$  otec  $v$ 
    # Rozdělíme vrchol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

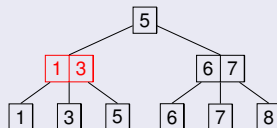
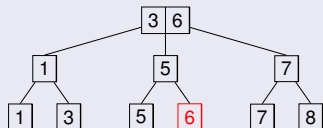
Smažte prvek s klíčem 4 z následujícího (2,4)-stromu



Nalezneme a smažeme list



Přesuneme jednoho syna od bratra nebo spojíme vrchol s bratrem



Algoritmus

```
1 Najít list / obsahující prvek s daným klíčem
2  $v \leftarrow \text{otec } l$ 
3 Smazat  $l$ 
4 while  $\text{deg}(v) < a$  &  $v$  není kořen do
5    $u \leftarrow \text{sousední bratr } v$ 
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       $v \leftarrow \text{otec } v$ 
```

- 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: $\mathcal{O}(b)$ ②
- 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 je hodnota b malá, je v praxi lepší lineárně projít všechny klíče.
- 2 Při rozdělení nebo sloučení vrcholů je nutné přesunout všechny syny a klíče, což je $\mathcal{O}(b)$.

Podobné datové struktury

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

Aplikace

- Souborové systémy 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 $\Rightarrow$ (256,512)-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	$a+1$	$\dots$	$2a-1$	$2a$	$2a+1$
$\Phi(u)$	2	1	0	$\dots$	0	2	4

- Potenciál stromu je součet potenciálů vnitřních 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 ②
- 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.
- 2 Při přesunu vrcholu se změní potenciál vrcholu, jeho otce a bratra. Po přesunu operace DELETE končí, takže se děje jen jednou.

Cíl

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

Operace Insert

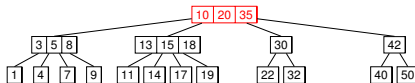
Preventivně rozdělit každý vrchol na cestě od kořene k hledanému listu s b syny na dva vrcholy.

Operace Delete

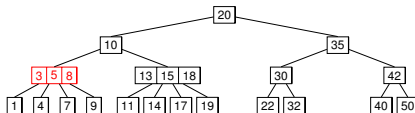
Preventivně sloučit každý vrchol na cestě od kořene k hledanému listu s a syny s bratrem nebo přesunout synovce.

(a,b)-strom: Balancování shora dolů: Příklad

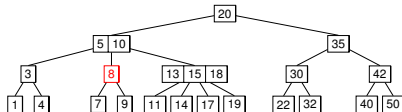
- Vložte prvek s klíčem 6 do následujícího (2,4)-stromu



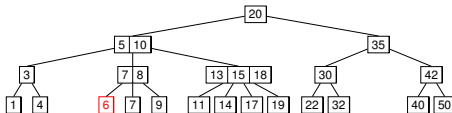
- Nejprve rozdělíme kořen



- Pak pokračujeme do levého syna, který také rozdělíme



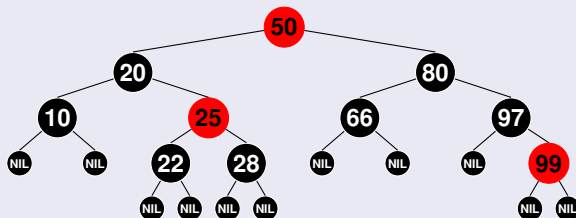
- Vrchol s klíčem 8 není třeba rozdělovat a nový klíč můžeme vložit



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

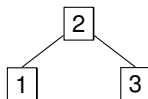
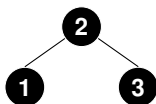


- 1 Nepovinná podmínka, která jen zjednodušuje operace. V příkladu uvažujeme, že listy jsou reprezentovány NIL/NULL ukazateli, a tedy imaginární vrcholy bez prvků. Někdy se též vyžaduje, aby kořen byl černý, ale tato podmínka není nutná, protože kořen můžeme vždy přebarvit na černo bez porušení ostatních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

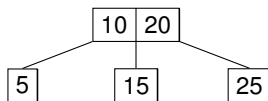
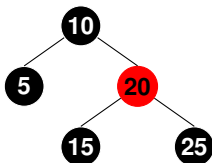
Červeno-černé stromy: Ekvivalence s (2,4)-stromy

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

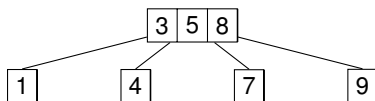
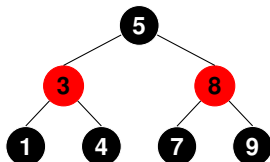
- Vrchol bez červených synů



- 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

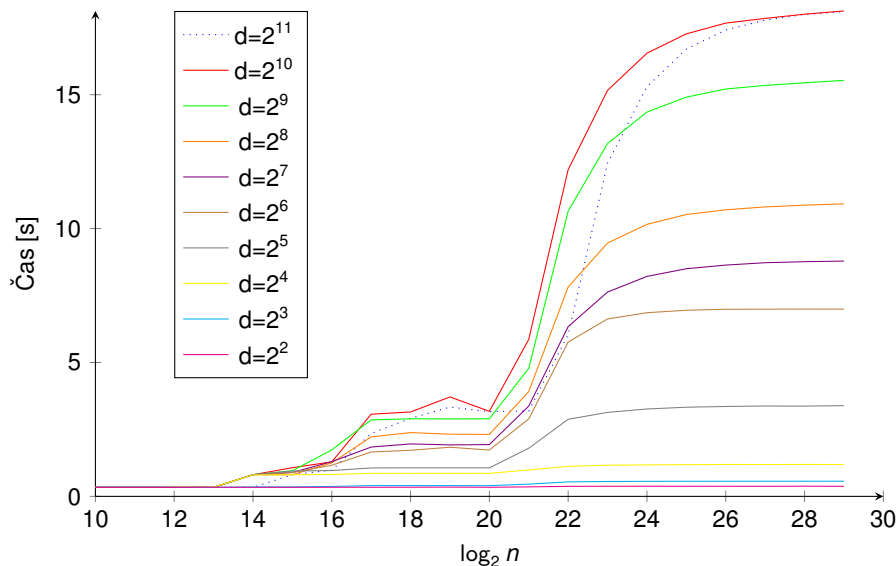
- Asociativní pole např. `std::map` a `std::set` v C++, `TreeMap` v Javě
- The Completely Fair Scheduler v jádru Linuxu
- Datové struktury ve výpočetní geometrii
- Persistentní datové struktury

Příklad velikostí a rychlostí různých typů pamětí

	velikost	rychlost
L1 cache	32 KB	223 GB/s
L2 cache	256 KB	96 GB/s
L3 cache	8 MB	62 GB/s
RAM	32 GB	23 GB/s
SSD	112 GB	448 MB/s
HDD	2 TB	112 MB/s

Triviální program

```
# Inicializace pole 32-bitových čísel velikosti  $n$ 
1 for ( $i=0; i+d<n; i+=d$ ) do
2    $A[i] = i+d$  # Vezmeme každou  $d$ -tou pozici a vytvoříme cyklus
3  $A[i]=0, i=0$ 
# Měříme dobu průběhu cyklu v závislosti na parametrech  $n$  a  $d$ 
# Počet operací je nezávislý na  $n$  a  $d$ 
4 for ( $j=0; j<2^{28}; j++$ ) do
5    $i = A[i]$  # Dokola procházíme cyklus  $d$ -tých pozic
```



Zjednodušený model paměti

- Uvažujeme pouze dvě úrovně paměti: pomalý disk a rychlá keš (cache)
- Paměť je rozdělená na bloky (stránky) velikosti B ①
- Velikost keše je M , takže keš má $P = \frac{M}{B}$ bloků
- Procesor může přistupovat pouze k datům uloženým v keši
- Paměť je plně asociativní ②
- Data se mezi diskem a keší přesouvají po celých blocích a cílem je určit počet bloků načtených do keše

Cache-aware algoritmus

Algoritmus zná hodnoty M a B a podle nich nastavuje parametry (např. velikost vrcholu B-stromu při ukládání dat na disk).

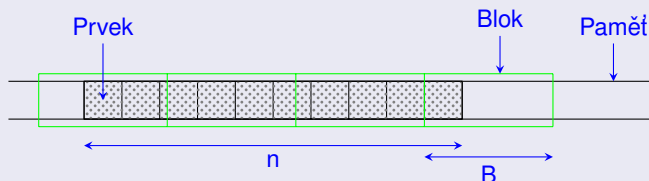
Cache-oblivious algoritmus

Algoritmus musí efektivně fungovat bez znalostí hodnot M a B . Důsledky:

- Není třeba nastavovat parametry programu, který je tak přenositelnější
- Algoritmus dobře funguje mezi libovolnými úrovněmi paměti (L1 – L2 – L3 – RAM)

- 1 Pro zjednodušení předpokládáme, že jeden prvek zabírá jednotkový prostor, takže do jednoho bloku se vejde B prvků.
- 2 Předpokládáme, že každý blok z disku může být uložený na libovolné pozici v keši. Tento předpoklad výrazně zjednodušuje analýzu, i když na reálných počítačích moc neplatí, viz https://en.wikipedia.org/wiki/CPU_cache#Associativity.

Přečtení souvislého pole (výpočet maxima, součtu a podobně)



- Minimální možný počet přenesených bloků je $\lceil n/B \rceil$
- Skutečný počet přenesených bloků je nejvýše $\lceil n/B \rceil + 1$
- Předpokládáme, že máme k dispozici $\mathcal{O}(1)$ registrů k uložení iterátoru a maxima

Obrácení pole

- Potřebujeme si v keši držet dva bloky, jinak je analýza stejná jako u přečtení souvislého pole
- Skutečný počet přenesených bloků je nejvýše $\lceil n/B \rceil + 1$ za předpokladu, že $P \geq 2$

Binární vyvážený strom

- Jeden vrchol je uložen v nejvýše 2 blocích ①
- Výška stromu je $\mathcal{O}(\log n)$
- Na cestě z kořene do listu načteme $\mathcal{O}(\log n)$ bloků

(a,b)-strom

- Předpokládejme, že můžeme zvolit $b = \Theta(B)$
- Jeden vrchol je uložen v nejvýše 2 blocích a má $\Theta(B)$ synů
- Výška stromu je $\Theta(\log_B n)$
- Na cestě z kořene do listu načteme $\Theta(1 + \log_B n)$ bloků
- Toto je cache-aware přístup

Reprezentace binárního stromu pomocí van Emde Boas

- Cache-oblivious reprezentace
- Na cestě z kořene do listu načteme $\Theta(1 + \log_B n)$ bloků
- Podrobnosti na Datových strukturách II

- 1 Předpokládáme, že prvky jsou dostatečně malé, aby se vrchol vešel do jednoho bloku a druhý blok máme pro případ, že nemůžeme alokovat paměť tak, aby paměť pro vrcholy byla zarovnána se začátky bloků.

Binární halda v poli: Průchod od kořene k listu



- Cesta má $\Theta(\log n)$ vrcholů
- Posledních $\Theta(\log B)$ vrcholů leží v nejvýše dvou blocích
- Ostatní vrcholy jsou uloženy v po dvou různých blocích
- $\Theta(1 + \log n - \log B) = \Theta(1 + \log \frac{n}{B})$ přenesených bloků ①

B-regulární halda v poli: Průchod od kořene k listu

- Opět předpokládáme, že do jednoho bloku se vejde B prvků
- Cesta má $\Theta(1 + \log_B n)$ vrcholů
- Na cestě z kořene do listu načteme $\Theta(1 + \log_B n)$ bloků

- 1 Odečtením členu $\log B$ asymptotickou složitost moc nezlepšíme, a proto se v literatuře obvykle neuvažuje. Cílem bylo nahlédnout, že náš odhad je doopravdy asymptoticky těsný.

Binární vyhledávání

- Uvažujeme neúspěšné vyhledávání, protože úspěšné může skončit dříve
- Porovnáváme $\Theta(\log n)$ prvků s hledaným prvkem
- Posledních $\Theta(\log B)$ prvků je uloženo v nejvýše dvou blocích
- Ostatní prvky jsou uloženy v po dvou různých blocích
- $\Theta(1 + \log \frac{n}{B})$ přenesených bloků

Pokračování v cache-oblivious algoritmech a datových strukturách.