Úvod do programování

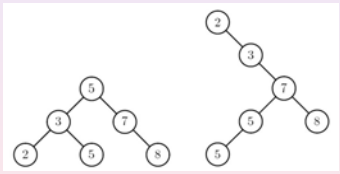
Michal Krátký[1], Jiří Dvorský[1]

[1]Katedra informatiky
VŠB–Technická univerzita Ostrava

Úvod do programování, 2004/2005

Dokonale vyvážené stromy

Časové složitosti operací nad binárním stromem závisí na jeho výšce ($\log n \leq h \leq n-1$). Snahou tedy je tuto výšku minimalizovat pro daný počet uzlů. Budeme chtít vytvořit strom, který je nějakým způsobem **vyvážený**.



Vyvážený strom lze intuitivně chápat jako strom jehož *pravý podstrom* je přibližně stejně velký jako *levý podstrom*, čili mají zhruba stejnou výšku.

Dokonale vyvážené stromy

Jestliže chceme vytvořit z n uzlů strom, který má minimální *výšku*, bude nutno abychom na každou úroveň umístili co největší počet uzlů. Tzn. jednotlivé uzly umístíme rovnoměrně na pravou a levou stranu právě vytvářeného uzlu.

Pravidlo rovnoměrné distribuce známého počtu n uzlů se dá nejlépe formulovat rekurzivně takto:

1. Zvolíme jeden uzel za *kořen stromu*.
2. Vytvoříme levý podstrom s počtem uzlů $n_l = n \operatorname{div} 2$.
3. Vytvoříme pravý podstrom s počtem uzlů $n_r = n - n_l - 1$.

Aplikací tohoto pravidla na posloupnost prvků dostaneme tzv. dokonale vyvážený binární strom.

Dokonale vyvážené stromy

Definice (Dokonale vyvážené stromy)
Strom se nazývá **dokonale vyvážený**, jestliže pro každý uzel stromu platí, že počet uzlů v jeho levém a pravém podstromu se liší nejvýše o jeden.

Tato definice nám zaručuje, že *délka všech cest* z *kořene* do *listů* se bude lišit nejvíce o jeden uzel. Dokonalá vyváženost stromu má však obrovskou nevýhodu v tom, že každé přidání uzlu nebo jeho smazání naruší vyváženost binárního stromu, který je nutno zkonstruovat znovu.

AVL stromy (AVL-Tree)

Obnova dokonalého vyvážení po náhodném přidání je složitá operace. Možné zlepšení spočívají ve formulování méně přísných definic vyváženosti. Tato nedokonalá kritéria vyváženosti by měla vést k jednodušším procedurám stromové reorganizace za cenu pouze minimálního zhoršení průměrné výkonnosti vyhledávání.

Adelson-Velskii a **Landis** zformulovali méně restriktivní definici vyváženosti.

Definice (Vyvážené stromy)
Strom je vyvážený tehdy a jen tehdy, je-li rozdíl *výšek* každého *uzlu* nejvýše 1.

AVL stromy

Stromy, které splňují toto kritérium, se často nazývají **AVL–stromy**. Definice je nejen jednoduchá, ale vede i k proceduře znovu vyvážení a k průměrné *délce cesty* vyhledávání, která je prakticky identická s délkou cesty *dokonale vyváženého stromu*.

Na AVL-stromech je možno v čase $O(\log n)$ vykonávat následující operace:

1. Vyhledání uzlu s daným *klíčem*.
2. Vložení uzlu s daným klíčem.
3. Zrušení uzlu s daným klíčem.

Výška stromu je určena Fibonacciho čísly, často tedy mluvíme o Fibonacciho stromu.

AVL stromy

Adelson-Velskii a Landis dokázali, že *AVL-strom* bude maximálně o 45% (nikdy ne víc) vyšší než jeho dokonale vyvážený *dvojník* bez ohledu na počet uzlů. Jestliže symbolem $h_b(n)$ označíme *výšku* AVL-stromu s n uzly, potom

$$\log(n+1) \leq h_b(n) \leq 1,4404 \log(n+2) - 0,328$$

Optimální hodnotu získáme v případě *dokonale vyváženého stromu*.

Vkládání do AVL-stromů

Při vkládání uzlu do *AVL-stromu* s *kořenem r* a dvěma podstromy *levým* T_L a *pravým* T_R mohou nastat tři případy. Předpokládejme, že se nový *uzel* přidá do levého podstromu T_L a tím způsobí zvýšení jeho *výšky* o 1. Výška podstromů před vložením uzlu:

1. $h_L = h_R$: T_L a T_R budou mít rozdílné výšky, ale kritérium vyváženosti zůstává neporušené.
2. $h_L < h_R$: T_L a T_R budou mít stejnou výšku tj. vyváženost se dokonce ještě zlepší.
3. $h_L > h_R$: kritérium vyváženosti se poruší, strom bude potřeba znovu vyvážit.

Vyvažovací faktor

Vyvažovací faktor $B_i = (h_{L_i} - h_{R_i})$, kde h_{L_i} je výška levého podstromu T_{L_i} uzlu i, h_{R_i} je výška pravého podstromu T_{R_i} uzlu i.

Tedy:

- Jestliže strom s kořenem i je vyvážený pak $|B_i| \leq 1$.
- Jestliže T_{R_i} je vyšší, $B_i < -1$.
- Jestliže T_{L_i} je vyšší, $B_i > 1$.

Vkládání do AVL stromu

1. Prohledání stromu abychom zjistili, zda se ve stromu daný uzel již nenachází.
2. Přidání nového uzlu a určení výsledného vyvažovacího faktoru.
3. Zkontrolování vyvažovacího faktoru, každého uzlu na cestě opačným směrem tj. na cestě od uzlu ke *kořeni*.

Metoda umožňuje implementovat jednoduché rozšíření vkládání do *binárního stromu*. Algoritmus je rekurzivní.
V každém kroku je potřeba vrátit informaci o tom, zda se *výška* podstromu (ve kterém se vkládání uskutečnilo) zvětšila nebo ne.

Vyvažování AVL stromu

Vyvažování stromu provádíme pomocí cyklické záměny ukazatelů - rotacemi.

Rotace:

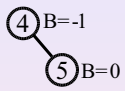
- Jednoduché (RR, LL).
- Dvojité (LR, RL).

Při vkládání a rušení uzlu do stromu je nutné uchovávat informace o vyváženosti struktury (*vyvažovací faktor*). Jednou z možností je uchovávat informaci o vyváženosti v každém uzlu.

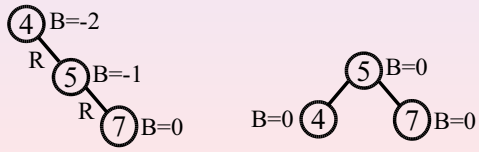
Uzel AVL stromu

```
public class AVLTreeNode
{
  public AVLTreeNode mLeftChild = null;
  public AVLTreeNode mRightChild = null;
  public int mKey = -1;
  public int mBalancedFactor = 0;
}
```

RR rotace, vkládání klíče 7

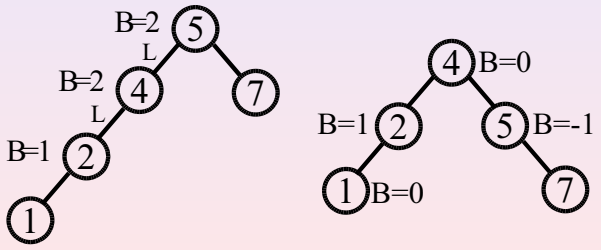


- **R**: levá rotace: 4 se stane levým dítětem uzlu 5.
- **R**: levá rotace: levé dítě uzlu 5 se stane pravým dítětem uzlu 4.

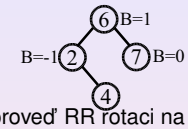


LL rotace, vkládání klíče 2 a 1

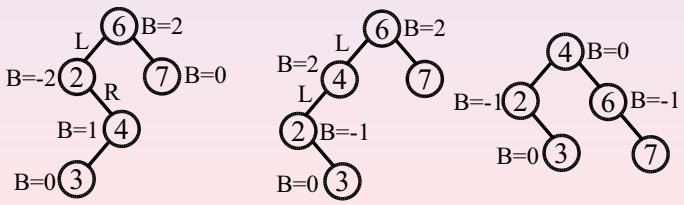
- **L**: pravá rotace: 5 se stane pravým dítětem uzlu 4.
- **L**: pravá rotace: pravé dítě uzlu 4 se stane levým dítětem uzlu 5.



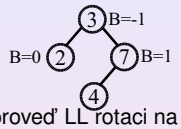
LR rotace, vkládání klíče 3



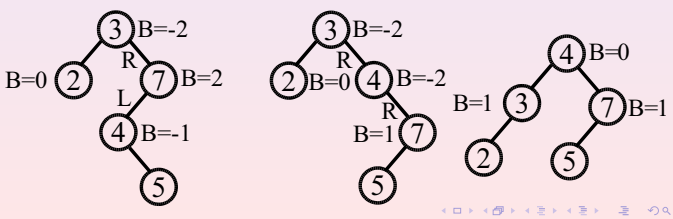
- **R**: levá rotace: proveď RR rotaci na uzlu 2.
- **L**: pravá rotace: proveď LL rotaci na uzlu 6.



RL rotace, vkládání klíče 5



- **L**: pravá rotace: proveď LL rotaci na uzlu 7.
- **R**: levá rotace: proveď RR rotaci na uzlu 4.



Analýza vkládání do AVL stromu

1. Jaká bude očekávaná *výška* AVL-stromu, jestliže se všech $n!$ *permutací* n klíčů vyskytuje se stejnou pravděpodobností?
2. Jaká je pravděpodobnost, že přidání uzlu způsobí znovu vyvažování?

Matematická analýza tohoto problému patří mezi otevřené otázky. Empirické testy potvrzují domněnku, že očekávaná výška AVL-stromu je $h = \log n + c$, kde c je konstanta s malou hodnotou ($c \cong 0,25$). AVL-stromy tedy vykazují podobné chování jako dokonale vyvážené stromy. Empirické testy ukazují, že v průměru je potřeba jedno vyvažování na přibližně každé dvě přidání nových uzlů.

Rušení uzlu v AVL stromu

Základním schématem realizace rušení uzlu v *AVL-stromu* je procedura rušení uzlu v *binárním vyhledávacím stromu*. Jednoduché jsou opět případy *listů* a *uzlů*, které mají jediného *potomka*. Pokud má uzel, který chceme zrušit dva *podstromy*, nahradíme jej opět nejpravějším z levého podstromu.

Při porušení podmínky vyváženosti musím opět provést příslušnou rotaci.

Analýza rušení uzlů z AVL stromu

Rušení uzlu v *AVL-stromu* je možné vykonat – v nejhorším případě – prostřednictvím $O(\log n)$ operací. Nepřehlédněme však podstatný rozdíl mezi chováním algoritmu přidávání a rušení uzlů. Zatímco přidání uzlu může způsobit nejvýše jednu rotaci (dvou nebo tří uzlů), rušení může vyžadovat rotaci všech uzlů absolvované cesty.

Výsledky empirických testů ukazují, že zatímco pro přibližně každé druhé přidání uzlu je potřeba jedna rotace, až každé páté zrušení vyvolá rotaci. Proto lze považovat rušení uzlů v *AVL-stromech* za stejně složité jako přidávání.

2-3-4 stromy

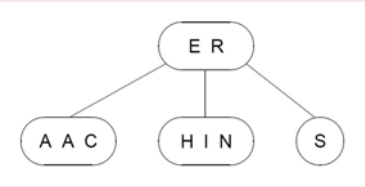
Abychom eliminovali nejhorší stavy při prohledávání *binárních stromů* potřebujeme vytvořit u námi používaných datových struktur jakousi flexibilitu. Zatím jsme se zabývali, uvolňováním striktnosti kritéria *dokonalé vyváženosti*. Nyní se zaměříme na uzly binárního stromu.

Uzly **2-3-4 strom** stromu mohou obsahovat více než jeden *klíč*. Tento strom obsahuje tři typy uzlů:

- **2-uzel**
- **3-uzel**
- **4-uzel**

2-3-4 stromy

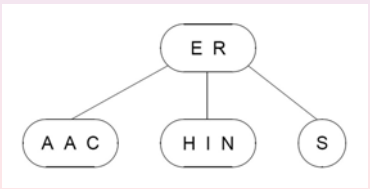
Tyto uzly obsahují 2, 3, resp. 4 ukazatele na *potomky*. 3-uzel resp. 4-uzel obsahuje dva resp. tři klíče. První ukazatel v 3-uzlu ukazuje na uzel s klíči menšími než oba klíče aktuálního uzlu, druhý ukazuje na uzel s hodnotami klíčů mezi oběma klíči aktuálního uzlu a třetí na uzel s klíči vyššími. Obdobná situace nastává u 4-uzlu.



Vložení klíče do 2-3-4 stromu

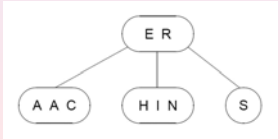
Při vložení klíče do stromu provedeme (neúspěšné) hledání a poté navážeme vkládaný klíč na posledně vyhledaný uzel. Mohou nastat tyto situace:

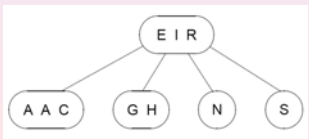
- *2-uzel* se změní na *3-uzel*. Např. klíč *X* by byl vložen do uzlu obsahující *S*.
- *3-uzel* se změní na *4-uzel*.
- *4-uzel* je rozštěpen. Např. vložení písmene G.



Vložení klíče do 2-3-4 stromu

Jedna možnost je navázat nový uzel na již existující uzel obsahující H, I a N, ale takové řešení není optimální. Lepší řešení je rozštěpit uzel na dva 2-uzly a přesunout jeden z klíčů do rodičovského uzlu. Nejdříve tedy rozdělíme H I a J 4-uzel na dva 2-uzly (jeden bude obsahovat H a druhý N) a *prostřední klíč* I přesuneme do 3-uzlu obsahujícího E a R, čímž z něho vytvoříme 4-uzel. Tím se pro klíč G vytvoří místo ve 2-uzlu obsahujícím H.





Vložení klíče do 2-3-4 stromu

Co když ale potřebujeme rozdělit 4-uzel, jehož rodičem je také *4-uzel*? Jednou z možností by bylo rozdělit i *rodiče*, ale i prarodič může být 4-uzel a i jeho rodič atd. Štěpení může pokračovat až ke kořeni.

Jednodušší je zajistit, aby žádný rodič jakéhokoliv uzlu nebyl 4-uzel tím, že cestou *dolů* rozdělíme všechny 4-uzly, na které narazíme.

Vložení klíče do 2-3-4 stromu

Pokud narazíme na 2-uzel na nějž je napojený 4-uzel transformujeme jej na 3-uzel na nějž jsou napojeny dva 2-uzly. Stejně tak, když narazíme na 3-uzel k němuž je připojený 4-uzel změníme jej na 4-uzel uzel na nějž jsou napojeny dva 2-uzly. Procházíme-li strom shora dolů máme jistotu, že uzel, který opouštíme není 4-uzel.

Kdykoliv se stane, že by kořen byl 4-uzlem, rozdělíme ho do tří 2-uzlů stejně tak, jak jsme to udělali v předchozích případech. Rozdělení kořene je jedinou operací která způsobí nárůst výšky stromu o jednu.

Vložení klíče do 2-3-4 stromu

Dělení uzlů je založeno na přesouvání klíčů a ukazatelů. Modifikace jsou lokální, případně se modifikují uzly aktuální cesty.

Takto navržený algoritmus ukazuje jak ve *2-3-4 stromu* vyhledávat a jak do něj vkládat. Jednoduše je třeba dělit 4-uzly na menší při cestě shora dolů. Proto se anglicky těmto stromům také říká *top-down 2-3-4 stromy*. Ačkoliv jsme se vůbec nestarali o vyvažování stromu, je vždy *dokonale vyvážený*!

Analýza vyhledávání v 2-3-4 stromu

Věta
Předpokládejme, že je dán *2-3-4 strom* s n uzly. Vyhledávací algoritmus navštíví nejvýše $\log n + 1$ uzlů.

Důkaz
Vzdálenost od *kořene* ke všem *listům* je stejná: transformace, jak jsme si ukázali, nemají na vzdálenost uzlů od kořene žádný vliv. Pouze pokud dělíme kořen, mění se výška stromu a v tom případě se hloubka všech uzlů zvýší o jedna. Pokud jsou všechny uzly 2-uzly je výška stromu stejná jako u *úplného binárního stromu*, pokud jsou přítomny i *3-uzly* a *4-uzly* může být výška vždy jenom menší.

Analýza vkládání do 2-3-4 stromu

Věta
Nechť je dán 2-3-4 strom s n uzly. Vkládací algoritmus potřebuje méně než $\log n + 1$ rozdělení uzlů a předpokládá se, že využívá průměrně méně než jedno rozdělení.

Důkaz
Nejhorším případem je dělení všech uzlů které po cestě dolů navštívíme. U stromu postaveného z náhodné permutace n prvků je tato situace krajně nepravděpodobná, čímž lze ušetřit mnoho dělení, protože ve stromu není mnoho 4-uzlů a drtivá většina z nich jsou *listy*. Výsledky analýzy průměrného chování 2-3-4 stromu mohou odrazovat, ale empirickým měřením bylo zjištěno, že se dělí jen velmi málo uzlů.

Implementace 2-3-4 stromu

Strom, který jsme definovali je vhodný pro definici vyhledávacího algoritmu se zaručenou nejhorší variantou. Implementace stromu umožňující za chodu přechod mezi jednotlivými uzly není příliš efektivní. Vyhledávací algoritmus bude pomalejší než standardní vyhledávací algoritmus na binárním stromu.

Je nutné jednoduše reprezentovat jednotlivé typy uzlů.

Implementace 2-3-4 stromu

2-3-4 strom je možno reprezentovat jako standardní binární strom (pouze 2-uzly) použitím speciálního bitu v každém uzlu. Myšlenka spočívá v tom, že 3-uzly a 4-uzly převedeme do malých binárních stromů spojených *červenými* ukazateli. Tyto pak kontrastují s *černými* ukazateli spojujícími dohromady vlastní 2-3-4 strom.