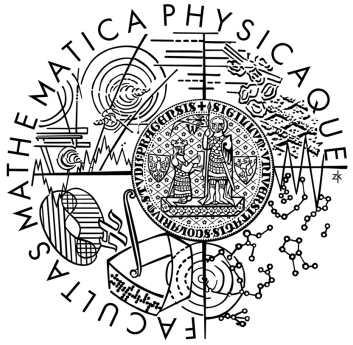
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Martin Kříž

Relaxované vyvažování binárních vyhledávacích stromů

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Alena Koubková, CSc.
Studijní program: Informatika
Studijní obor: ISS softwarové systémy

Na tomto místě bych rád poděkoval svým rodičům za podporu během studia, RNDr. Aleně Koubkové CSc. za čas a energii, kterou věnovala vedení této práce, a za cenné rady při konzultacích. V neposlední řadě bych rád poděkoval Ing. Janě Klozové za podporu a trpělivost během psaní diplomové práce a za pomoc při závěrečných úpravách.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 13. prosince 2006

Martin Kříž

Obsah

Abstrakt

Název práce: Relaxované vyvažování binárních vyhledávacích stromů
Autor: Martin Kříž
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Alena Koubková, CSc.
E-mail vedoucí: Alena.Koubkova@mff.cuni.cz

Abstrakt: Na rozdíl od klasických vyvážených binárních vyhledávacích stromů, kdy proces vyvažování následuje bezprostředně po každém vložení nebo ubrání prvku, relaxované vyvažování umožňuje oddělit tyto fáze a provést vyvažování odděleně. Má význam například při paralelním přístupu k datům, kdy je možné vyvažování odložit na dobu, kdy je systém málo zatížen požadavky uživatelů.

Další výraznou výhodou popsaných typů relaxovaného vyvažování je to, že při paralelním přístupu k datům potřebují držet pouze konstatní počet zámků při modifikujících operacích a umožňují tak více modifikujících operací současně ve stromu.

Cílem této práce je experimentálně porovnat klasickou a relaxovanou variantu AVL stromu v několika různých scénářích v paralelním prostředí podle počtu porovnání, počtu a typu rotací a podle spotřebovaného času.

Klíčová slova: datové struktury, binární vyhledávací stromy, AVL stromy, relaxované vyvažování, rebalancer, paralelní přístup

Abstract

Title: Binary search trees with relaxed balance
Author: Martin Kříž
Department: Department of Software Engineering
Supervisor: RNDr. Alena Koubková, CSc.
Supervisor's e-mail address: Alena.Koubkova@mff.cuni.cz

Abstract: In standard binary trees the rebalancing is carried out in connection with and immediately folowing the updates. Relaxed balancing allows to separate updates and rebalancing. First advantage of this approach is to keep the tree partially unbalanced and leave rebalancing to the different time when system is idle.

Other great benefit of presented relaxed balancing algorithms in concurrent enviroment is necessity of keeping only small constant number of locks for modifying operations and thus allowing more modifying operations in the tree at the same time.

The aim of this thesis is to empirically compare standard and relaxed variant of AVL tree in several different scenarios in concurrent enviroment according to number of data compares, number and type of rotations and according to the time requirements.

Keywords: Data structures, Search trees, AVL trees, Relaxed balance, Rebalancer, Parallel data access

1 Úvod

Existuje mnoho datových struktur, které umožňují efektivně implementovat slovníkové operace *search*, *insert* a *delete*. Mezi nejpoužívanější patří různé varianty binárních vyhledávacích stromů, kde časová složitost operací je úměrná výšce stromu, a proto se používají vyvážené stromy zajišťující logaritmickou výšku stromu. Při implementacích různých databází se používají především *AVL stromy*, *červeno-černé stromy* a *(a,b) stromy*.

Standardní implementace těchto algoritmů vyžadují po každé modifikující operaci (insert, delete) sekvenci vyvažovacích kroků, které obnoví podmínku vyváženosti těchto stromů. Dodržováním podmínky vyváženosti docílíme toho, že strom nezdegeneruje do lineárního seznamu, výška stromu bude logaritmická a operace na něm budou efektivní.

K databázím však obvykle přistupuje paralelně několik klientů najednou, a proto by také mělo být možné přistupovat paralelně do vyhledávací datové struktury. Je třeba předejít tomu, aby se procesy ve vyhledávacím stromu navzájem ovlivňovaly. To můžeme řešit například zamykáním určitých uzlů ve stromu pro individuální potřeby procesu, čímž zamezíme ostatním procesům v přístupu k nim. Každý proces by měl uvolnit zámek hned poté, co ho již nebude potřebovat.

Aby byl paralelní přístup do stromu co nejefektivnější, neměl by žádný proces nikdy držet více než malý a konstantní počet zámků. Toho lze velmi dobře docílit, pokud uvažujeme pouze operace search, insert a delete bez vyvažování stromu [7]. Avšak aby byly operace ve stromu efektivní, potřebujeme mít strom vyvážený. Pokud bychom implementovali standardní vyvažování zezdola-nahoru [1], nemůžeme implementovat modifikující operace tak, aby držely pouze konstantní počet zámků. Toho můžeme docílit dvěma způsoby.

Můžeme použít vyvažování seshora-dolů [5, 16], kdy modifikující operace při postupu dolů modifikuje strom tak, že bude vyvážen po dokončení updatu. Nevýhodou této metody u AVL stromů, které jsou používány především pro svou jednoduchost, je obtížné programování tohoto typu vyvažování (viz [16]).

Druhým způsobem je oddělení vyvažování stromu od modifikujících operací. Modifikující operace poznamená určitým způsobem u uzlů, jestli jsou nevyvážené. Vyvažování pak probíhá jako zvláštní proces nazývaný *rebalancer*, který lokálními transformacemi modifikuje strom a odstraňuje z něj nevyváženosti. Tomuto

způsobu vyvažování se říká *relaxované vyvažování.*

Myšlenka oddělit vyvažování od updatů byla poprvé zmíněna v souvisloti s červeno-černými stromy [5]. Částečné řešení vytvořil pro AVL stromy Kessels [6]. Toto řešení počítá pouze s operacemi search a insert. První úplné řešení relaxovaného vyvažování AVL stromů, ovšem bez důkazů složitosti, lze najít v [14] a je popsáno v kapitole 4 této práce. Důkazy složitosti vyvažování na velmi podobném stromu jsou pak uvedeny v [9]. V tomto řešení, stejně jako v jiných navrhovaných řešení pro červeno-černé stromy [3, 15], může být strom plný konfliktů, které musí rebalancer řešit, a přitom může být strom vyvážený. Možnost zapomenout nějaké problémy nevyváženosti, způsobené například přidáním a odebráním téhož prvku, může být přínosná. Velmi elegantní řešení, které toto umožňuje, bylo prezentováno v [17] a následně rozšířeno o důkazy složitosti v [9], je popsáno v kapitole 5.

Relaxované vyvažování umožňuje efektivní implementaci paralelního přístupu do vyhledávacího stromu. To je ale pouze jedna z jeho výhod. Tou druhou je samotné oddělení vyvažování od modifikujících operací, které sice může znamenat dočasnou ztrátu vyváženosti stromu, ale na druhou stranu dojde k výraznému zrychlení modifikujících operací.

To lze v praxi využít například v situaci, kdy požadavky přicházejí po dávkách a vyhledávací strom není schopen vyřizovat požadavky tak rychle, jak by bylo třeba. Může se jednat například o časovou špičku, kdy je systém zatížen mnoha různými uživateli, nebo o situaci, kdy administrátor nějakou dávkou operací upravuje data ve vyhledávacím stromu. V takové situaci lze díky relaxovanému vyvažování vypnout vyvažování na kratší čas, dokud dávka (špička) neskončí. Vyhledávací strom si bude pamatovat jaké uzly je třeba vyvážit a po ukončení dávky, zatímco do vyhledávacího stromu dále mohou přistupovat uživatelé a provádět operace search, insert i delete dle libosti, lze vyhledávací strom postupně vyvažovat zpět.

Čím déle bude vyvažování vypnuto, tím více hrozí degenerace vyhledávacího stromu a tím pomalejší jednotlivé operace mohou být. Z teoretického i praktického hlediska je jistě zajímavé, zda strom s vypnutým vyvažováním půjde efektivně vyvážit zpět.

1.1 Cíle práce

Tato práce se zaměřila na zkoumání AVL stromů. AVL stromy jsou vhodné především pro implementace databází s menším počtem prvků v operační paměti počítače, kde uplatní svoji jednoduchost a rychlost jednotlivých operací. Například jsou částí stránkovacího mechanismu jádra operačního systému Linux. Tomu bylo přizpůsobeno i prostředí pro testování, kdy se celý vyhledávací strom nachází v operační paměti počítače.

Cílem této diplomové práce je nejprve popsat některé známé algoritmy relaxovaného vyvažování AVL stromů (kapitoly 4 a 5). Poté implementovat paralelní prostředí pro testy a naprogramovat v tomto prostředí standardní algoritmus pro AVL strom a algoritmus pro relaxované vyvažování AVL stromu (dle kapitoly 6).

Tyto algoritmy pak postavit před různé scénáře lišící se stupněm paralelity či pravděpodobnostmi jednotlivých operací, porovnat je mezi sebou na základě počtu porovnání prvků, počtu a typu použitých rotací, času potřebného pro jednotlivé operace a celkového času. Výsledky měření těchto testů pak srozumitelnou formou prezentovat (kapitola 7).

Tato práce si klade za cíl ukázat, že i přes vypnutí vyvažování na relativně dlouhou dobu, kdy počet operací od vypnutí vyvažování bude srovnatelný s celkovým počtem prvků ve stromu, se strom nevymkne kontrole, bude stále velmi podobný tradičnímu AVL stromu, a výhody relaxovaného vyvažování, které spočívají v urychlení modifikujících operací a ve vysoké míře paralelity předčí nevýhodu částečně nevyváženého stromu a pomalejších operací na něm.

Dalším cílem této práce je ukázat, že úplné vyvážení relaxovaného stromu zpět do AVL stromu je i přes dočasnou ztrátu kontroly nad stromem stále efektivní, a že celkový počet provedených rotací se nebude příliš lišit od standardní varianty AVL stromu, spíše bude díky zapomínání změn i nižší.

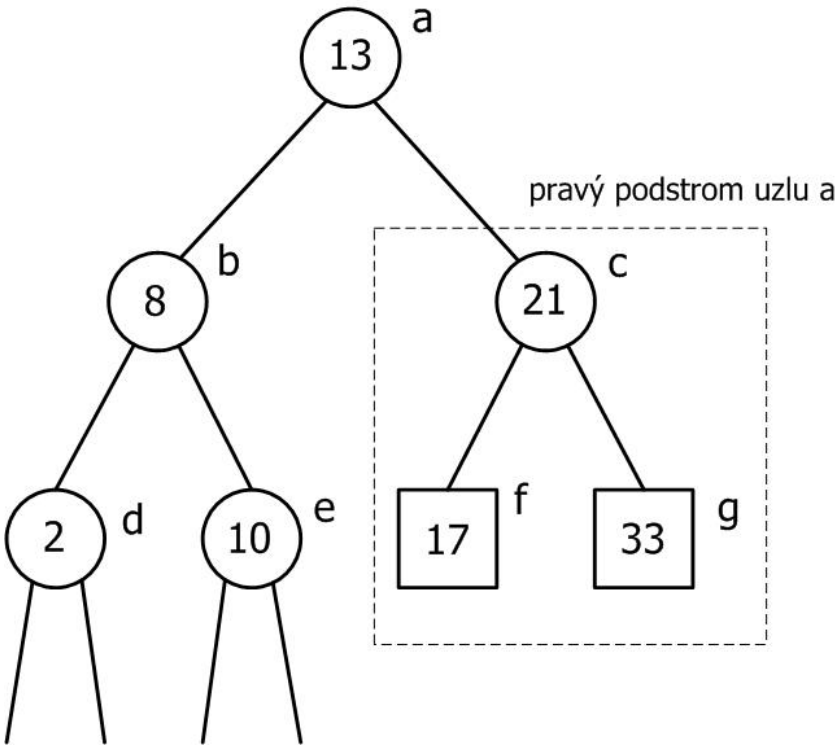
2 Základní pojmy

V této kapitole uvedeme základní pojmy a označení, s kterými budeme pracovat v celém dalším textu.

Stromem nazveme souvislý neorientovaný graf, který neobsahuje cykly a který má jeden speciální uzel nazývaný *kořen stromu*.

Uvažujme uzel x ve stromu, různý od kořene. Pak libovolný uzel u na cestě z kořene stromu do uzlu x nazýváme *předchůdcem* uzlu x. Pokud u je předchůdce x pak říkáme, že x je *následníkem* (potomkem) u. Pokud existuje hrana mezi u a x a u je blíže ke kořeni, říkáme, že u je *otcem* (rodičem) uzlu x a x je *synem* uzlu u. Dva uzly mající stejného rodiče označujeme ja *sourozence* (bratry).

Uzel, který nemá žádné syny navýváme *listem*. Uzly, které nejsou listy nazýváme *vnitřní uzly*.



Obrázek 1: Uzly b,c,d,e,f,g jsou potomci uzlu a. Uzly b,c jsou synové uzlu a. Uzly b,c jsou sourozenci (bratři). Uzel b je otcem uzlů d,e. Uzly a,b,c,d,e jsou vnitřní uzly. Uzly f,g jsou listy. Listy budeme značit čtverečkem. Uzly značené kolečkem mohou být vnitřní uzly, ale i listy.

Výška uzlu u je délka nejdelší cesty z u do listu. Značíme ji $height(u)$. Tedy pro libovolný uzel u a jeho syny c a s platí:

$$height(u) = \begin{cases} 0, & \text{pokud } u \text{ je listem,} \\ max(height(c), height(s)) + 1, & \text{jinak.} \end{cases}$$

Výška stromu T je rovna výšce jeho kořene.

Velikost stromu T je počet listů ve stromu. Značíme $|T|$.

Levým (resp. pravým) podstromem uzlu u nazveme strom s kořenem v levém (resp. pravém) synovi uzlu u.

2.1 Binární vyhledávací stromy

V celém světě relaxovaného vyvažování se používají úplné, na listy orientované binární vyhledávací stromy, a proto výjimkou nebude ani tato práce. Někde se místo orientace na listy používá pojem *externí*.

Definice: *Úplný, listově orientovaný binární vyhledávací strom je strom, pro který platí:*

1. *každý uzel má buď dva nebo žádného následníka. (úplnost)*
2. *klíče jsou uloženy pouze v listech. Vnitřní uzly obsahují pouze tzv. směrovače, které slouží po určení správné cesty stromem k listu. (orientace na listy)*
3. *pro každý uzel u platí, že všechny hodnoty (klíče i směrovače) v levém podstromu uzlu u jsou menší nebo rovny hodnotě v uzlu u a všechny hodnoty z pravého podstromu u jsou větší než hodnota v u.*

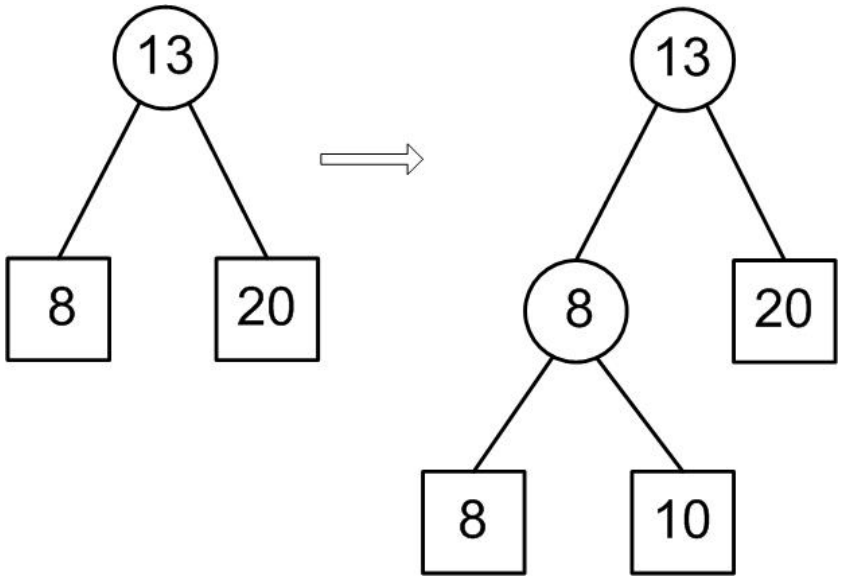
Binární vyhledávací strom umožňuje následující operace:

2.1.1 Operace SEARCH

Operace *search* (někdy označována jako *find* nebo *member*) slouží k vyhledání prvku ve stromu. Operace prochází strom od kořene směrem dolů k listům a na základě porovnání hledaného klíče a hodnoty směrovače v uzlu se rozhodne, zda pokračovat do levého nebo pravého podstromu. Pokud algoritmus dojde do listu a hledaný klíč se shoduje s hodnotou v listu, je požadovaný klíč nalezen. V opačném případě není vyhledávaný klíč ve stromu.

2.1.2 Operace INSERT

Operace *insert* slouží ke vložení klíče do binárního vyhledávacího stromu. Operace nejprve projde stromem směrem k listu a hledá vkládaný klíč. Pokud jej nalezne, operace končí. Pokud jej nenalezne, zastaví se v listu. Označme jej l. Pro nově vkládaný klíč je vytvořen nový list $l2$. List l je nahrazen vnitřním uzlem u se dvěma syny - listy l a $l2$. Levým synem uzlu u se stane ten s menší hodnotou klíče. Směrovač v uzlu u je kopií klíče v levém synovi.

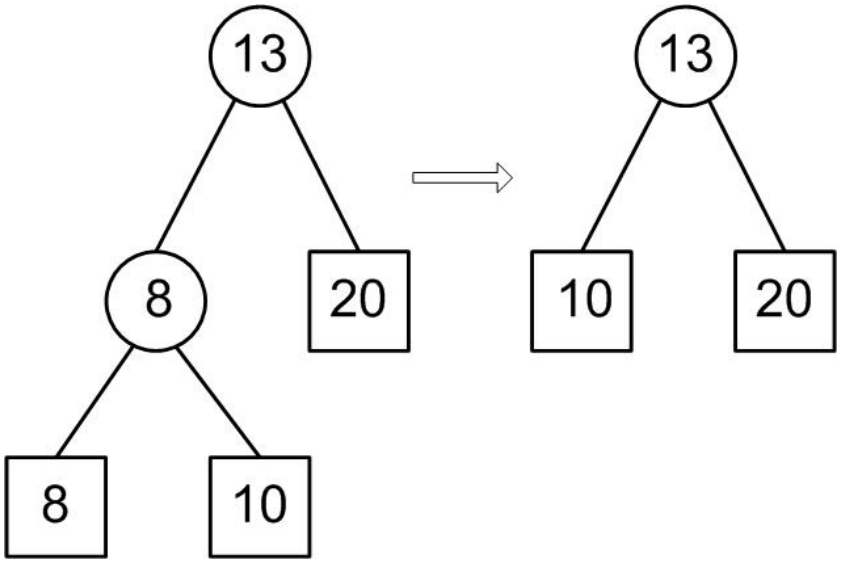


Obrázek 2: Vložení klíče 10 do binárního vyhledávacího stromu.

2.1.3 Operace DELETE

Operace *delete* slouží ke smazání klíče z binárního vyhledávacího stromu. Samotnému mazání opět předchází hledání uzlu s mazaným klíčem. Pokud ve stromu takový list není, operace skončí neúspěchem. Pokud tam takový list je, je smazán spolu se svým rodičem a jeho místo zabere bratr mazaného listu.

Časová složitost všech tří výše popsaných operací na binárním vyhledávacím stromu je úměrná výšce stromu, která je v ideálním případě rovna $log(|T|)$. Strom však může posloupností operací insert a delete zdegenerovat na obyčejný spojový seznam a jeho výška se změní na $|T|$. Stromy zajišťující optimální výšku $O(log|T|)$



Obrázek 3: Smazání klíče 8 z binárního vyhledávacího stromu.

a tím i časovou složitost v nejhorším případě $O(log|T|)$ se nazývají *vyvážené stromy*. Mezi ně patří například AVL stromy, červeno-černé stromy nebo B-stromy.

Použití na listy orientovaných stromů přináší oproti interní variantě hned několik výhod. Klíče se nacházejí pouze v listech, a proto veškeré operace jsou čistě lokální (v interní variantě například mazání klíče z kořene znamená získat novou hodnotu až z listu), což je velkou výhodou především v paralelním prostředí. Při relaxovaném přístupu, kdy se strom nevyvažuje, stačí při sestupu stromem držet pouze malý počet zámků.

2.2 Další pojmy

Dále se v textu můžeme setkat s následujícími pojmy:

modifikující operace/update: INSERT nebo DELETE

reader: Proces (klient), který nad stromem provádí operaci SEARCH.

writer/updater: Proces, který nad stromem provádí modifikující operaci.

rebalancer: Proces provádějící vyvažování relaxovaného stromu.

3 AVL stromy

AVL stromy jsou poměrně známou a hojně popsanou strukturou, proto v této práci budou popsány pouze zběžně a více prostoru bude věnováno relaxovaným variantám AVL stromů.

Definice: *Binární vyhledávací strom nazveme AVL stromem právě tehdy, když se u každého vnitřního uzlu výšky jeho dvou podstromů liší maximálně o jedna.*

Jak vypadá nejhorší možný AVL strom? Tedy AVL strom, který při dané výšce h obsahuje nejmenší počet uzlů? Označme takový strom T_h. T_0 je list. T_1 je strom obsahující vnitřní uzel a list. Abychom sestrojili strom T_h pro $h > 1$ připojíme ke kořeni dva podstromy s minimálním počtem uzlů. Levý o výšce $h - 1$ a pravý o výšce $h - 2$. Výsledný strom bude AVL stromem s minimálním počtem uzlů a s výškou h. Protože konstrukce těchto stromů je velmi podobná definici Fibonacciho čísel, označují se tyto stromy jako *Fibonacciho stromy.*

Přidáním nového listu do AVL stromu může dojít ke zlepšení podmínky vyváženosti (podstrom, do kterého se přidávalo, byl nižším podstromem), zhoršení podmínky vyváženosti (výšky obou podstromů byly stejné) nebo k porušení podmínky vyváženosti (přidávalo se do vyššího podstromu). Pokud nastane třetí případ, je potřeba strom vyvážit tak, aby opět odpovídal definici AVL stromu.

Důkladným rozborem situace zjistíme, že se jedná o 4 případy, které mohou nastat, z nichž 2 a 2 jsou symetrické. Uvažujme proto uzel p takový, že došlo k přidání listu do levého podstromu uzlu p, který byl vyšší než pravý podstrom uzlu p. Výška uzlu c je po přidání o 2 větší než výška uzlu s. Označme levého syna p jako c a pravého jako s, levý podstrom uzlu c jako A a pravý podstrom c jako B. Nyní mohou nastat dvě možnosti:

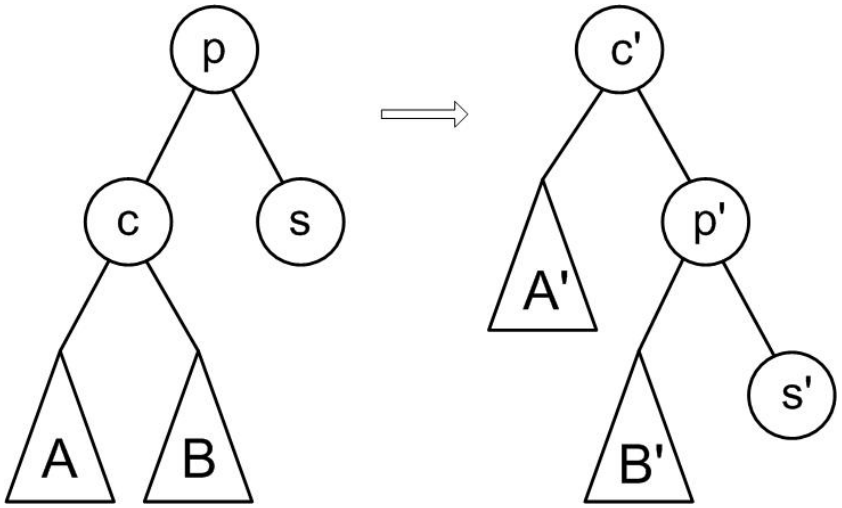
- výška A $\geq$ výška B

  V takovém případě provedeme jednoduchou rotaci (viz Obrázek 4).

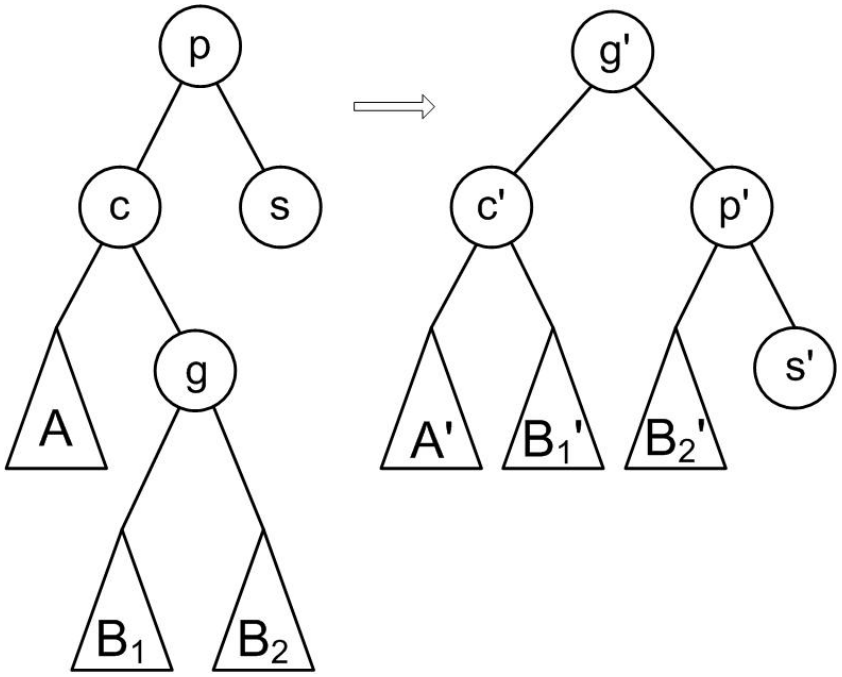
- výška A $<$ výška B

  V tomto případě provedeme dvojitou rotaci (viz obrázek 5).

Rotace zachovávají správné uspořádání klíčů ve stromu a zachovávají nebo snižují výšku ovlivněného podstromu o 1. Vzhledem k tomu, že je výška uzlu počítána



Obrázek 4: Jednoduchá rotace



Obrázek 5: Dvojitá rotace

rekurzivně na základě výšek jeho předchůdců, lze snadno nahlédnout, že pokud zůstane zachována výška nějakého uzlu u, do jehož podstromu byl proveden update, zůstane zachována i výška všech předchůdců uzlu u.

Lze ukázat, že po vložení nového klíče stačí provést nejvýše jednu rotaci. V případě mazání klíče ze stromu už tomu tak být nemusí. Uvažme mazání nejpravějšího listu z Fibonacciho stromu.

3.0.1 Operace INSERT

Nejprve dojde k hledání klíče, jestli se již ve stromu nevyskytuje. Poté následuje přidání nového vrcholu a nakonec algoritmus postupuje proti směru vyhledávání do kořene a kontroluje podmínku vyváženosti. Pokud je porušena, použije jednu z výše uvedených rotací. Po provedení jedné rotace algoritmus může skončit.

3.0.2 Operace DELETE

Nejprve dojde k vyhledání klíče, pokud se nachází ve stromu, je spolu se svým otcem smazán a nahrazen bratrem mazaného listu. Poté algoritmus postupuje nahoru proti směru hledání ke kořenu a v každém uzlu, pokud je porušena podmínka vyváženosti, použije jednu z výše uvedených rotací. Algoritmus končí až poté, co dosáhne kořene stromu.

Pokud označíme $h(n)$ výšku AVL stromu s n vrcholy, Adelson-Velskii a Landis dokázali v [1] následující větu:

Věta 1: $log(n+1) \leq h(n) \leq 1,4404.log(n+2) - 0,328$

Přímým důsledkem této věty je to, že v AVL stromech lze v čase $O(log(|T|)$ vykonávat operace search, insert a delete.

Pro analýzu složitosti relaxovaného stromu, který je uveden v kapitole 5 této práce se nám budou hodit následující tvrzení:

Tvrzení 2: *Počet uzlů ve Fibonacciho stromu* $|T_h| = F_{h+3} - 1$*, kde Fibonacciho čísla* F_i *jsou definována takto:* $F_1 = F_2 = 1, F_i = F_{i-1} + F_{i-2}, i > 2$

Důkaz: jednoduchý, indukcí.

Tvrzení 3: *Hodnota Fibonacciho čísla* $F_i = \frac{(\frac{1+\sqrt{5}}{2})^i - (\frac{1-\sqrt{5}}{2})^i}{\sqrt{5}}$

Důkaz: jednoduchý, indukcí.

Tvrzení 4: *Hodnota Fibonacciho čísla* $F_i \geq \frac{\varphi^i}{\sqrt{5}} - 1$ *kde* $\varphi = \frac{1+\sqrt{5}}{2}$ *a nazývá se zlatý poměr.*

Důkaz: jednoduchý, z výrazu z Tvrzení 3.

4 AVL stromy s relaxovaným vyvažováním

Relaxované vyvažování znamená oddělení vyvažovacích kroků od modifikujících operací. Vyvažování by mělo být možné provádět při paralelním přístupu ke stromu "na pozadí", tzn. spolu s probíhajícími updaty. Aby rebalancer poznal místo, kde je strom nevyvážený, nechává updater po cestě dolů v uzlech značky nazývané *tagy*. Podle těchto tagů a podle faktorů vyváženosti se rebalancer rozhodne, je-li třeba provést vyvažování a pokud ano, modifikuje strom tak, že se více podobá AVL stromu (podle určité míry podobnosti, která bude specifikována později).

Updater způsobí nevyváženost stromu buď prodloužením nebo zkrácením cesty z nějakého vnitřního uzlu do listu. Toto prodloužení (resp. zkrácení) cesty kompenzujeme změnou číselné hodnoty tagu u nějakého uzlu na cestě tak, aby strom byl vyvážený pokud použijeme délky cest a přičteme k nim hodnoty tagů jednotlivých uzlů. Kapitola dle [14].

4.1 Definice

Každému uzlu u binárního vyhledávacího stromu přiřadíme celočíselnou hodnotu $tag(u)$, což je integer větší než nebo roven -1. Relaxovaná výška uzlu u je definována podobně jako u standardních AVL stromů.

Definice: *Relaxovaná výška uzlu u, označená* $rh(u)$ *je rovna:*

$$rh(u) = \begin{cases} 1 + tag(u), & \textit{pokud } u \textit{ je listem,} \\ max(rh(left(u)), rh(right(u))) + 1 + tag(u), & \textit{jinak.} \end{cases}$$

Poznámka: Výška listu bývá definována jako 0. Jednička v definici relaxované výšky listu je proto, že hodnota tagu může být -1. Relaxovaná výška žádného uzlu tak nebude negativní.

Relaxovaná výška stromu T (značme ji $rh(T)$) je rovna relaxované výšce jeho kořenu. Kritérium vyváženosti je zde definováno analogickým způsobem jako u AVL stromů.

Nechť u je nějaký vnitřní uzel. Pak *relaxovaný faktor vyváženosti* uzlu u (značený $rbf(u)$) je roven rozdílu relaxovaných výšek jeho podstromů. Tj: $rbf(u) = rh(left(u)) - rh(right(u))$.

Strom nazveme *relaxovaným AVL stromem*, pokud pro všechny jeho vnitřní uzly platí podmínka: $-1 \leq rbf(u) \leq 1$.

Zjevně platí, že relaxovaný AVL strom je AVL stromem, pokud všechny tagy mají nulovou hodnotu. Opačná implikace neplatí, neboť strom může být AVL stromem i přesto, že některé tagy mají nenulové hodnoty.

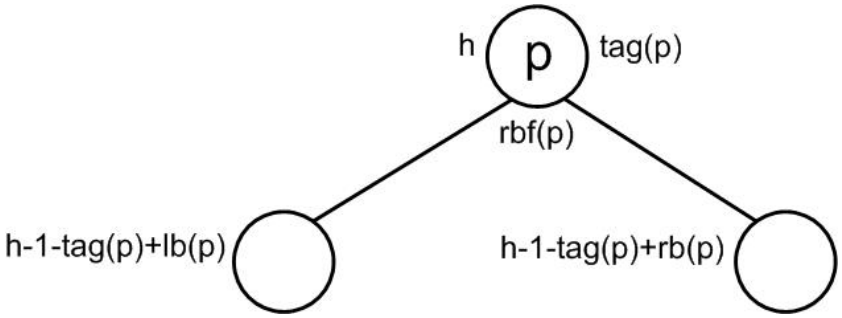
Pro zjednodušení následujících částí textu definujeme dvě pomocné funkce $lb(u)$ a $rb(u)$ takto:

$$lb(u) = \begin{cases} -1, & \text{pokud } rbf(u) = -1, \\ 0, & \text{jinak.} \end{cases}$$

$$rb(u) = \begin{cases} -1, & \text{pokud } rbf(u) = 1, \\ 0, & \text{jinak.} \end{cases}$$

Všimněme si, že $lb(u) - rb(u) = rbf(u)$ pro každý uzel u v relaxovaném AVL stromu.

Tyto pomocné funkce jsou užitečné kvůli tomu, že pokud nějaký vnitřní uzel p relaxovaného AVL stromu má relaxovanou výšku h, pak relaxovaná výška levého syna uzlu p může být vyjádřena jako $h - 1 - tag(p) + lb(p)$ a relaxovaná výška pravého syna uzlu p jako $h - 1 - tag(p) + rb(p)$ (viz Obrázek 6).



Obrázek 6: Vztah mezi relaxovanými faktory vyváženosti a relaxovanými výškami podstromů. Tagy jsou zobrazeny napravo od příslušných uzlů, relaxované faktory vyváženosti pod nimi a relaxované výšky jsou zobrazeny vlevo od nich.

Je dobře viditelné, že relaxované kritérium vyváženosti je jednodušší dodržet po změně struktury stromu, než je tomu u klasických AVL stromů. U nich je kritérium vyváženosti závislé pouze na výškách podstromů a jediná možnost jak dodržet toto

kritérium je provést další strukturální změny (rotace), které změní odpovídajícím způsobem výšky podstromů. Zatímco u relaxovaných AVL stromů lze kritérium vyváženosti dodržet zvýšením či snížením hodnot tagů některých uzlů. Jak ukážeme dále, stačí modifikovat hodnoty tagů u malého pevně určeného počtu uzlů.

Začínáme s vyváženým AVL stromem. Hodnoty tagů u všech uzlů mají hodnotu 0. Updater může vytvářet po své cestě nenulové hodnoty tagů. Úkolem rebalanceru je pak pomocí lokálních změn struktury stromu hodnoty těchto tagů vrátit zpět na nulu. V momentě, kdy všechny tagy budou nulové, bude strom opět AVL stromem.

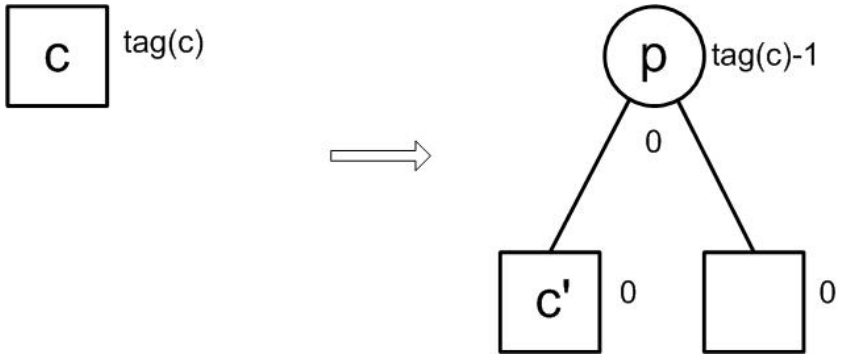
4.2 Operace

4.2.1 Operace INSERT

Procházíme stromem a hledáme list l obsahující klíč k. Pokud jej najdeme, operace je ukončena. Pokud jej nenajdeme, pak vyhledávání bylo neúspěšné a zastavilo se v listu. Označme jej c. Nahradíme c novým vnitřním uzlem p a list c spolu s nově přidávaným listem se stanou syny uzlu p. Tito synové jsou setříděni tak, že ten s menší hodnotou klíče je levým synem uzlu p. Klíč uzlu p je kopií klíče jeho levého syna. Hodnoty tagů nově přidávaného listu a listu c' nastavíme na 0. Protože výška uzlu p je o 1 větší než byla výška uzlu c, musíme tento rozdíl kompenzovat tím, že nastavíme hodnotu tagu uzlu p na $tag(c) - 1$. Je zřejmé, že teď je relaxovaná výška uzlu p stejná, jako byla relaxovaná výška uzlu c, a není tedy potřeba propagovat změny výše do stromu. Relaxovaný faktor vyváženosti $rbf(p)$ je nastaven na 0, protože výšky obou jeho podstromů jsou rovny 1 (viz Obrázek 7).

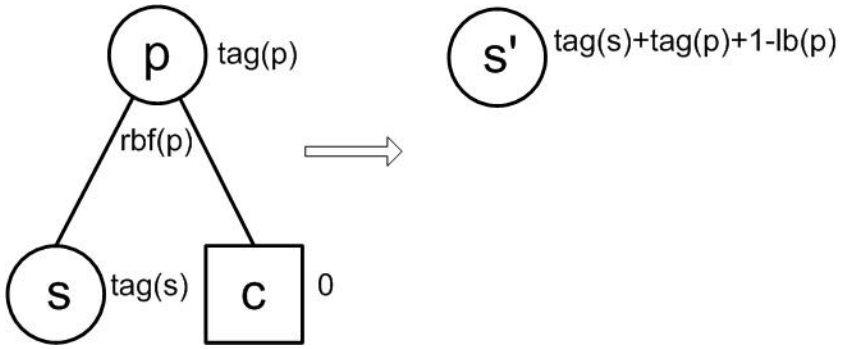
4.2.2 Operace DELETE

Stejně jako v případě operace insert procházíme stromem a hledáme list obsahující klíč k. Pokud takový list není nalezen, proces je ukončen s chybou. V opačném případě odstraníme list c obsahující daný klíč k a nahradíme otce listu c (označme jej p) sourozencem uzlu c (označme jej s). Čili dojde k odebrání dvou uzlů (listu i vnitřního uzel). Nyní je relaxovaná výška uzlu s o jedna menší, než byla relaxovaná výška uzlu p. Tuto změnu můžeme kompenzovat zvýšením hodnoty tagu uzlu s tak, aby relaxovaná výška zůstala zachována. Toto zvýšení hodnoty tagu je opět lokální operace a změna není propagována výše do stromu.



Obrázek 7: Insertem vzniká pravý (na obrázku) nebo levý sourozenec v závislosti na hodnotě klíče vkládaného uzlu. Tagy jsou zobrazeny napravo a relaxované faktory vyváženosti pod příslušnými uzly. Po manipulaci je uzel t označen t'.

Předpokládejme, že $rh(p) = h$ a s je levým synem uzlu p. Potom relaxovaná výška uzlu s může být vyjádřena jako $h - 1 - tag(p) + lb(p)$. Označme požadované zvýšení hodnoty tagu uzlu s jako x. Potřebujeme tedy, aby $rh(s) + x = h$. Čili $h - 1 - tag(p) + lb(p) + x = h$ nebo $x = tag(p) + 1 - lb(p)$. Pokud hodnota tagu uzlu s byla $tag(s)$, pak hodnota tagu uzlu s' je rovna $tag(s) + tag(p) + 1 - lb(p)$. Relaxovaný faktor vyváženosti uzlu s zůstane zachován (viz Obrázek 8).



Obrázek 8: Pravý sourozenec je smazán. Levým sourozencem může být list nebo vnitřní uzel. Hodnota tagu uzlu s' je spočítána výše.

Zatím jsme neuvažovali triviální případy, kdy strom je buď prázdný nebo obsahuje jeden list. Pokud vkládáme klíč do prázdného stromu, vytvoříme list obsahující daný klíč a nastavíme hodnotu tagu tohoto listu na 0. Pokud mažeme klíč ze stromu, který obsahuje pouze jeden list, je tento list zrušen (samozřejmě pouze za

předpokladu, že obsahuje správný klíč) a výsledkem je prázdný strom.

Všimněme si, že hodnotu tagu nějakého uzlu snižujeme pouze při vkládání nového uzlu. V tomto případě hodnotu tagu nového vnitřního uzlu spočítáme tak, že snížíme o 1 hodnotu tagu nějakého listu. Listy ale mají vždy nezápornou hodnotu tagu, protože počáteční hodnota tagu je 0, a v případě mazání není nově spočítaná hodnota tagu jakéhokoliv uzlu nikdy menší než hodnota předchozí. Z toho tedy plyne následující fakt.

Fakt 1: *Pokud začínáme s prázdným stromem, pak při jakékoliv sekvenci operací INSERT a DELETE (jak jsou popsány výše) nikdy nevznikne hodnota tagu nějakého uzlu menší než -1.*

Ukázali jsme (v popisu operací), že kdykoliv operace INSERT nebo DELETE manipuluje nějakým podstromem daného stromu, pak relaxovaná výška kořene onoho podstromu zůstane zachována. To znamená, že zůstane zachována i relaxovaná výška otce modifikovaného podstromu a tedy i relaxované kritérium vyváženosti celého stromu. Proto můžeme Fakt 1 rozšířit následovně:

Fakt 2: *Pro daný relaxovaný AVL strom T s hodnoty tagů většími než nebo rovnými nule u vnitřních uzlů a většími než nebo rovnými -1 u listů a pro danou libovolnou sekvenci operací INSERT a DELETE platí, že výsledný strom je relaxovaným AVL stromem, který nemá hodotu tagu žádného uzlu menší než -1.*

4.3 Vyvažování

Nyní definujeme lokální vyvažovací operace, které po aplikování na relaxovaný AVL strom modifikují tento strom na jiný relaxovaný AVL strom, který je "více podobný" AVL stromu. Navrhneme rebalancer tak, že každý jeho krok sníží míru nevyváženosti daného relaxovaného AVL stromu.

Formálně definujeme pro strom T a uzel u z T hodnotu $outside(u)$ jako počet uzlů stromu T, které se nenacházejí v podstromu stromu T s kořenem u. Potom pro uzel u z T definujeme:

$$unbalance(u) = |tag(u)| * outside(u).$$

Pro relaxovaný AVL strom T je jeho celková nevyváženost rovna součtu nevyvážeností jednotlivých uzlů.

$$unbalance(T) = \sum_{u \in T} unbalance(u).$$

Rebalancer sníží nevyváženost stromu v každém kroku. Pokud bude $unbalance(T)$ rovno 0, pak $unbalance(u) = 0$ pro všechny uzly u ve stromu T. Tedy po konečném počtu vyvažovacích kroků je jakýkoliv relaxovaný AVL strom transformován na regulérní AVL strom. Hodnota tagu kořene stromu neovlivňuje nevyváženost celého stromu, protože je násobena nulou. Proto můžeme hodnotu tagu kořene stromu vždy nastavit na nulu a předpokládat tedy, že je vždy nula.

Předpokládejme, že rebalancer našel uzel p s hodnotou $tag(p) = 0$, který má syna c s hodnotou $tag(c) \neq 0$. Pokud strom obsahuje nějaké uzly s nenulovými hodnotami tagů, pak takový uzel vždy existuje, protože hodnota tagu kořene je vždy nula. Každé spuštění rebalanceru nalezne takový uzel a provede nějaké lokální vyvažovací změny, které sníží nevyváženost stromu a současně také zachovají relaxovanou výšku stromu. Toho docílí jednak změnou hodnot některých tagů a také příslušnými rotacemi.

Mějme uzel p s $tag(p) = 0$, který má alespoň jednoho syna s nenulovou hodnotou tagu. Vyvažovací krok rozdělíme do dvou fází. V první fázi je absolutní hodnota tagu onoho syna snížena o jedna a hodnota tagu uzlu p je nastavena tak, aby byla zachována relaxovaná výška uzlu p. Toto snížení hodnoty tagu může vést k tomu, že u uzlu p vznikne faktor vyváženosti 2 či -2. V druhé fázi je tento faktor vyváženosti snížen rotací nebo změnou hodnot některých tagů. Hodnoty tagů jsou nastaveny tak, aby operace nepropagovala změny výše do stromu a zůstala tak lokální. Libovolný uzel u budeme značit u' po transformaci na konci fáze.

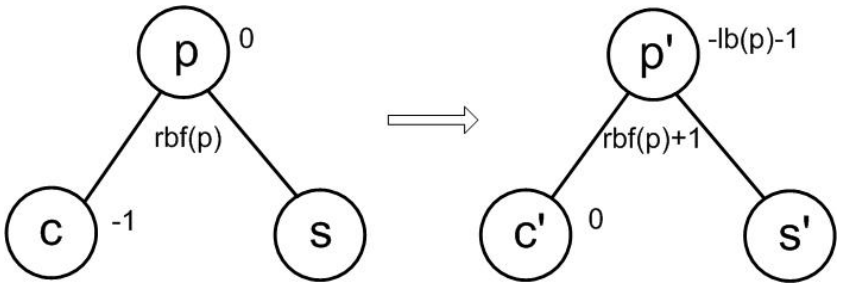
Fáze 1: Snížení absolutní hodnoty tagu.

Máme uzel p se dvěma syny c a s, pro které platí $tag(p) = 0$ a $tag(c) \neq 0$. Popíšeme zde pouze případ, kdy c je levým synem uzlu p, protože v opačném případě je řešení symetrické. Můžou nastat následující dva případy.

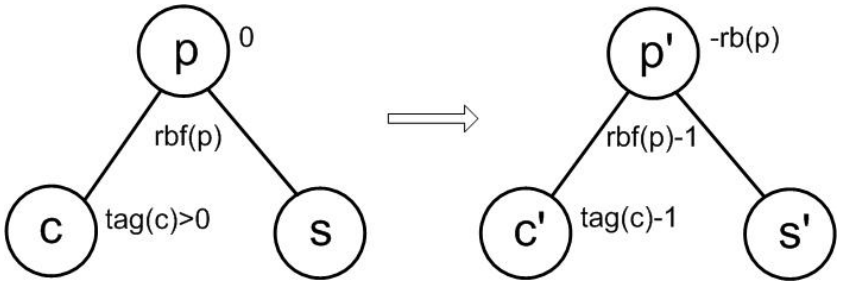
Případ 1: $tag(c) = -1$.

Nastavíme $tag(c')$ na 0. Tím se nám zvýšila relaxovaná výška uzlu c', a proto nastavíme $rbf(p')$ na $rbf(p) + 1$ (viz Obrázek 9). Pokud byl $rbf(p) = 1$, pak $rbf(p') = 2$, a je potřeba vyvážit strom ve fázi 2. (Pokud by byl uzel c pravým synem uzlu p, pak $rbf(p')$ by byl o 1 menší než $rbf(p)$, což by mohlo vést ke vzniku relaxovaného faktoru vyváženosti s hodnotou -2.)

Pokud byla relaxovaná výška uzlu c větší nebo rovna relaxované výšce uzlu s, pak po jejím zvýšení o 1 bude i relaxovaná výška uzlu p' o 1 větší, než byla relaxovaná výška uzlu p. To kompenzujeme nastavením $tag(p')$ na $= -lb(p)-1$ (jde o hodnotu 1 nebo 0 podle toho, který ze synů uzlu p měl větší relaxovanou výšku), aby se nevyváženost nepropagovala výše do stromu. Proč vypadá zvýžení tagu právě takto bude dokázáno později v důkazu Věty 4.



Obrázek 9: Fáze 1, případ 1: Odstranění záporné hodnoty tagu.



Obrázek 10: Fáze 1, případ 2: Snížení kladné hodnoty tagu.

Případ 2: $tag(c) > 0$.

Nastavíme $tag(c')$ na $tag(c) - 1$, čímž jsme snížili relaxovanou výšku uzlu c' o jedna, a proto musíme nastavit $rbf(p')$ na $rbf(p) - 1$. Pokud $rbf(p) = -1$, pak $rbf(p') = -2$, a je potřeba ve fázi 2 vyvážit strom tak, aby byla zachována relaxovaná podmínka vyváženosti. (V symetrickém případě může vzniknout uzel p' s $rbf(p') = 2$).

Opět, protože se snížila hodnota tagu uzlu c, snížila se i relaxovaná výška uzlu c. Abychom předešli propagaci výše do stromu a zachovali

operaci lokální, kompenzujeme tuto změnu relaxovaných výšek nastavením $tag(p') = -rb(p)$. Opět jde o hodnoty 1 nebo 0 v závislosti na relaxovaných výškách uzlů c a s (viz Obrázek 10).

Fáze 2: Obnovení relaxované podmínky vyváženosti.

Po skončení první fáze může mít relaxovaný faktor vyváženosti uzlu p hodnotu 2 nebo -2. Pokud tomu tak je, musíme znovuvyvážit tento podstrom. Pokud tomu tak není, podstrom je již relaxovaným AVL stromem a tento krok vyvažovací operace je u konce. Opět budeme uvažovat pouze případ, kdy $rbf(p) = 2$. Druhá možnost je symetrická.

Pro přehlednost, stejně jako ve fázi 1, bude označen levý syn uzlu p jako c a pravý syn jako s. Pokud je $tag(c) > 0$, pak můžeme vyvážit strom v uzlu p pouhou změnou hodnoty tagu úplně stejně, jako je uvedeno v případě 2 fáze 1. Dále tedy můžeme předpokládat, že $tag(c) = 0$. V tom případě můžou nastat dvě možnosti v závislosti na relaxovaném faktoru vyváženosti uzlu c.

Případ 1: $rbf(c) \geq 0$.

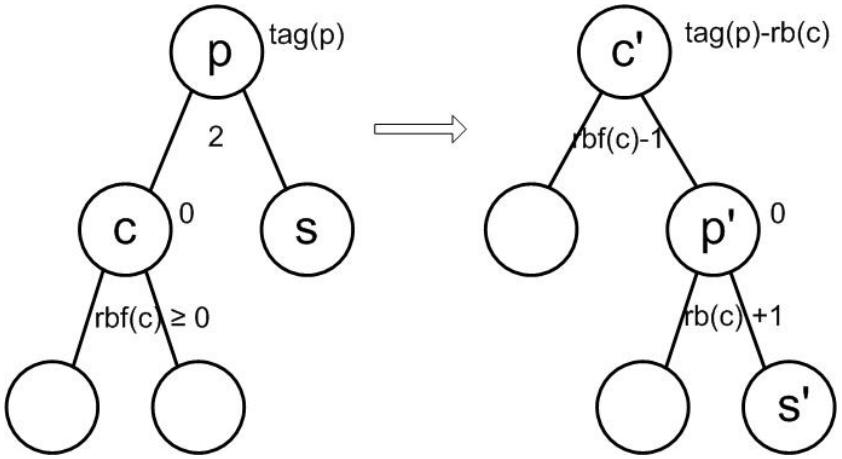
Strom můžeme vyvážit jednoduchou rotací. Po rotaci nastavíme $tag(p')$ na 0. Nové relaxované faktory vyváženosti uzlů p' a c' jsou nastaveny na $rb(c) + 1$ a $rbf(c) - 1$. Abychom zamezili propagaci nevyváženosti výše do stromu, nastavíme hodnotu $tag(c')$ na $tag(p) - rc(c)$ (viz Obrázek 11).

Případ 2: $rbf(c) = -1$.

Strom může být vyvážen pouhou změnou hodnot tagů některých uzlů nebo dvojitou rotací, po které bude následovat odpovídající úprava hodnot tagů u některých uzlů. Všimněme si, že u klasických AVL stromů je vždy zapotřebí dvojité rotace v odpovídajících případech. Dále v textu budeme označovat pravého syna uzlu c jako g. V závislosti na hodnotě tagu uzlu g máme tři podpřípady.

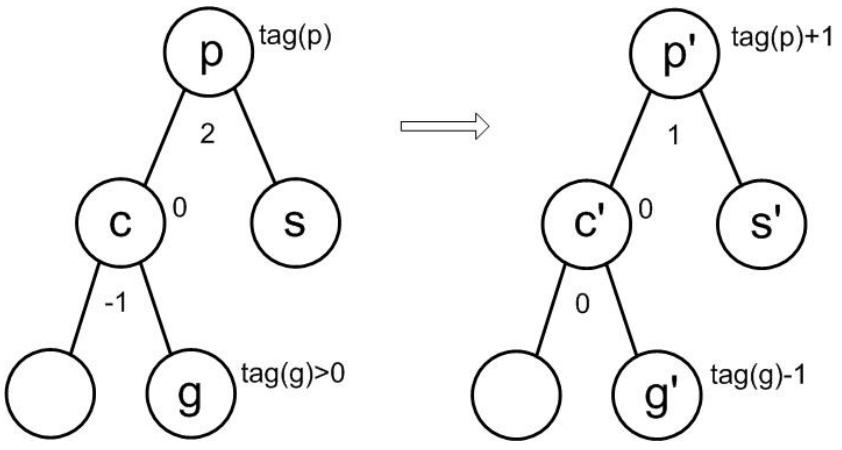
Případ 2a: $tag(g) > 0$.

Strom vyvážíme nastavením hodnoty $tag(g')$ na $tag(g) - 1$. Tím dojde ke snížení relaxované výšky uzlu g o 1, a proto nastavíme $rbf(c')$ na 0. Relaxovaná výška uzlu c se také snížila o 1, a proto nastavíme



Obrázek 11: Fáze 2, případ 1: Jednoduchá rotace v případě že $rbf(c) \geq 0$

$rbf(p')$ na 1. Díky tomu je splněna relaxovaná podmínka vyváženosti a strom je opět relaxovaným AVL stromem. Na závěr, protože se také snížila relaxovaná výška uzlu p o 1, nastavíme $tag(p')$ na $tag(p) + 1$, abychom předešli propagaci nevyváženosti výše do stromu (viz Obrázek 12). Jak je vidět, v tomto případě není třeba použít dvojitou rotaci.

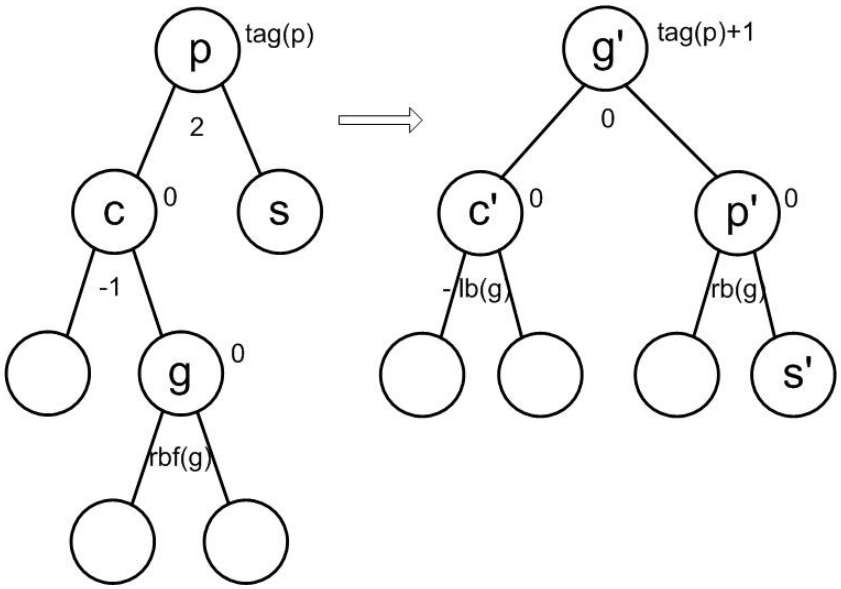


Obrázek 12: Fáze 2, případ 2a: Vyvažování je realizováno pouze změnou hodnot tagů

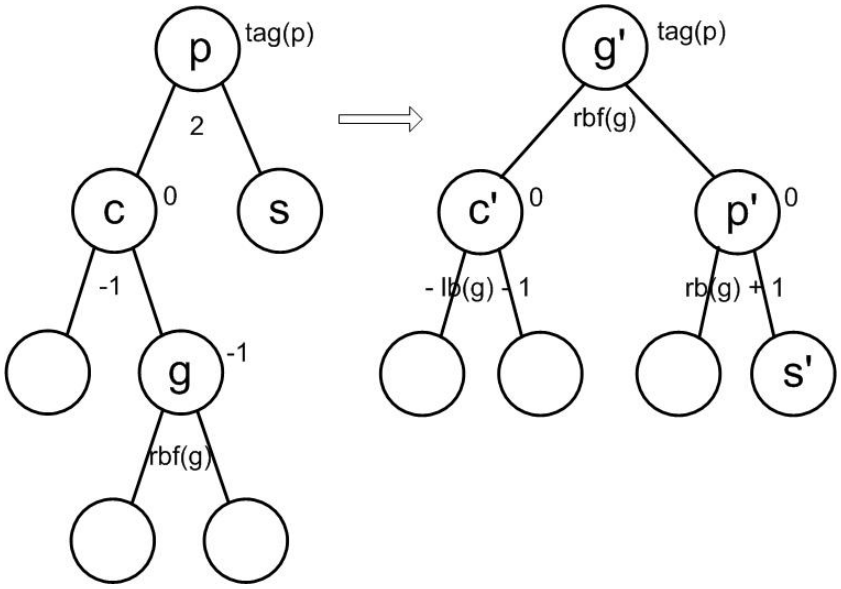
Případ 2b: $tag(g) = 0$.

Strom lze vyvážit použitím dvojité rotace v uzlu p (viz Obrázek

13). Nové relaxované faktory vyváženosti uzlů c' a p' jsou $-lb(g)$ a $rb(g)$. Hodnoty tagů uzlů c' a p' nastavíme na 0. Jak lze vidět v důkazu Věty 4, relaxované výšky dětí uzlu g' jsou stejné, a proto relaxovaný faktor vyváženosti uzlu g' je 0. Abychom opět zamezili propagaci neyváženosti výše do stromu, nastavíme hodnotu $tag(g')$ na $tag(p) + 1$.



Obrázek 13: Fáze 2, případ 2b: Dvojitá rotace



Obrázek 14: Fáze 2, případ 2c: Dvojitá rotace

Případ 2c: $tag(g) = -1$.

Strom lze v tomto případě vyvážit za použití dvojité rotace v p (viz Obrázek 14). Nové relaxované faktory vyváženosti uzlů c' a p' jsou $-lb(g) - 1$ a $rb(g) + 1$. Opět nastavíme $tag(c')$ a $tag(p')$ na 0, a proto $rbf(g') = rbf(g)$. Hodnota tagu uzlu g' je nastavena na $tag(p)$ abychom předešli propagaci nevyváženosti.

V [14] neřeší variantu, kdy by po skončení fáze 1 byl $tag(c) = -1$, ale jak si ukážeme, tuto variantu lze za určitých podmínek eliminovat. Zamysleme se nad tím, jak mohla ve fázi 1 vzniknout varianta $rbf(p) = 2$ a $tag(c) = -1$. V případě 1 se změní $tag(c)$ na 0. V symetrické variantě vznikne $rbf(p) = -2$. V případě 2 je snížena hodnota $tag(c)$ o 1, minimálně však na 0, a navíc může vzniknout pouze $rbf(p) = -2$. Zajímavá je však symetrická varianta případu 2, pokud $tag(c) = -1$ a $tag(s) > 0$, a vyvažování by se rozhodlo pro snížení tagu u uzlu s. Pak by mohla nastat výše zmíněná situace. K tomu však nemusí dojít, pokud rebalancer dá vždy přednost odstranění záporných tagů před snižováním kladných tagů.

Předtím, než ukážeme, že rebalancer z relaxovaného AVL stromu vytvoří opět relaxovaný AVL strom, ukážeme, že jakákoliv modifikace ve fázi 1 následovaná v případě potřeby odpovídající modifikaci ve fázi 2, sníží celkovou nevyváženost

stromu.

Věta 3: *Nechť T je relaxovaným AVL stromem, na kterém je vykonán jeden vyvažovací krok, a T' je stromem po ukončení vyvažovacího kroku. Pak platí:*

$$unbalance(T') < unbalance(T).$$

Důkaz: Ve fázi 1 je hodnota $unbalance(T') - unbalance(T)$ redukována pouze na $unbalance(p')+unbalance(c')-unbalance(p)-unbalance(c)$. Protože $unbalance(p)$ je 0, $|tag(p')| \leq 1$ a $outside(p) < outside(c)$, lze již jednoduše dopočítat, že $unbalance(T') < unbalance(T)$. Pokud na konci fáze 1 je $rbf(p')$ 2 nebo -2, může dojít ve fázi 2 ke zvýšení absolutní hodnoty tagu u kořene modifikovaného podstromu, ale protože ve fázi 1 byla snížena absolutní hodnota tagu u uzlu c o 1 a $outside(c) > outside(p) = outside(c')$ (obdobně pro uzel s), lze snadno ověřit, že $unbalance(T') < unbalance(T)$.

Věta 4: *Nechť T je relaxovaným AVL stromem, na kterém je vykonán jeden vyvažovací krok. Pak po tomto kroku platí:*

1. *Relaxovaná výška modifikovaného podstromu zůstává zachována.*
2. *Přiřazené relaxované faktory vyváženosti jsou správné.*

Důkaz: Označme relaxovanou výšku původního podstromu jako h. V každém případu každé fáze musíme ukázat, že relaxovaná výška modifikovaného stromu je také rovna h, a že kdykoliv rebalancer přiřadí k nějakému uzlu relaxovaný faktor vyváženosti k, tak platí, že $k = rh(left(u)) - rh(right(u))$. Abychom to ukázali, spočteme u každého případu relaxované výšky potřebných uzlů.

Fáze 1: Ve fázi 1 došlo ke zvýšení relaxované výšky uzlu c (v případě 1) resp. ke snížení (v případě 2) o 1. Relaxovaný faktor vyváženosti byl tedy o jedna zvětšen resp. zmenšen a je správný.

Zřejmě platí: $rh(c) = h - 1 + lb(p)$ a $rh(s) = h - 1 + rb(p)$. V modifikovaném stromu platí:

$rh(p') = max(rh(c'), rh(s')) + 1 + tag(p')$.

Případ 1: $tag(c) = -1$.

Protože $tag(c') = tag(c) + 1, rh(c') = rh(c) + 1 = h + lb(p)$. Protože jsme nastavili $tag(p')$ na $-lb(p) - 1$, tak:

$$\begin{aligned} rh(p') &= max(rh(c'), rh(s')) + 1 + tag(p') \\ &= max(h + lb(p), h - 1 + rb(p)) + 1 - lb(p) - 1 \\ &= h \end{aligned}$$

Případ 2: $tag(c) > 0$.

Protože $tag(c') = tag(c) - 1$, platí $rh(c') = rh(c) - 1 = h - 2 + lb(p)$. A dále jsme nastavili $tag(p') = -rb(p)$.

$$\begin{aligned} rh(p') &= max(rh(c'), rh(s')) + 1 + tag(p') \\ &= max(h - 2 + lb(p), h - 1 + rb(p)) + 1 - rb(p) \\ &= h \end{aligned}$$

Fáze 2: Podobně jako ve fázi 1 zde platí: $rh(c) = h - 1 - tag(p)$ a jelikož $rbf(p)$ je dočasně roven 2, $height(s) = h - 3 - tag(p)$.

Případ 1: $rbf(c) \geq 0$.

Vzhledem k tomu, že $tag(c) = 0$, relaxovaná výška uzlu $left(c)$ je $h - 2 - tag(p) + lb(c)$ a relaxovaná výška $right(c)$ je $h - 2 - tag(p) + rb(c)$. Tyto dvě relaxované výšky spolu s relaxovanou výškou uzlu s se během rotace nemění. Po rotace je $left(c') = left(c), right(c') = p', left(p') = right(c), right(p') = s$.

$$\begin{aligned} rbf(p') &= rh(left(p')) - rh(right(p')) \\ &= h - 2 - tag(p) + rb(c) - (h - 3 - tag(p)) \\ &= rb(c) + 1 \end{aligned}$$

Protože $tag(p)$ je nastaven na 0, pak platí:

$$\begin{aligned} rh(p') &= max(h - 2 - tag(p) + rb(c), h - 3 - tag(p)) + 1 \\ &= h - 1 - tag(p) + rb(c) \end{aligned}$$

A proto je hodnota $tag(c)$ nastavena na $tag(p) - rb(c)$. Výška modifikovaného podstromu tedy je

$$\begin{aligned} rh(c') &= max(h-2-tag(p)+lb(c), h-1-tag(p)+rb(c))+ \\ &\quad 1+tag(p)-rb(c) \\ &= max(lb(c)-1, rb(c))+h-rb(c) \\ &= h \end{aligned}$$

Zbývá nám $rbf(c')$.

$$\begin{aligned} rbf(c') &= h-2-tag(p)+lb(c)-(h-1-tag(p)+rb(c)) \\ &= rbf(c)-1 \end{aligned}$$

Případ 2: $rbf(c) = -1$. Protože důkaz případu 2a je jednoduchý a důkaz případu 2c je podobný důkazu případu 2b a navíc je uveden v [14], provedeme zde důkaz pouze pro případ 2b.

Případ 2b: $tag(g) = 0$.

Protože $rbf(c) = -1$ a $tag(c) = 0$, tak můžeme vyjádřit $rh(left(c)) = h-3-tag(p)$ a $rh(g) = h-2-tag(p)$. Dále, protože $tag(g) = 0$, pak $rh(left(g)) = h-3-tag(p)+lb(g)$ a $rh(right(g)) = h-3-tag(p)+lb(g)$. Po dokončení rotace je g' kořen podstromu, $left(g') = c', right(g') = p', left(c') = left(c), right(c') = left(g), left(p') = right(g)$ a $right(p') = s$ (viz Obrázek 13).

$$\begin{aligned} rbf(c') &= h-3-tag(p)-(h-3-tag(p)+lb(g)) \\ &= -lb(g) \end{aligned}$$

$$\begin{aligned} rbf(p') &= h-3-tag(p)+rb(g)-(h-3-tag(p)) \\ &= rb(g) \end{aligned}$$

Protože jsme nastavili $tag(c')$ na 0, získáváme

$$\begin{aligned} rh(c') &= max(h-3-tag(p), h-3-tag(p)+lb(g))+1 \\ &= h-2-tag(p) \end{aligned}$$

Obdobně pro p'

$$\begin{aligned} rh(p') &= max(h-3-tag(p)+rb(g), h-3-tag(p))+1 \\ &= h-2-tag(p) \end{aligned}$$

Z toho je přímo vidět, že $rbf(g') = 0$. Hodnotu tagu g' nastavíme na $tag(p)+1$ a relaxovaná výška g' je tedy

$$\begin{aligned} rh(g') &= max(h-2-tag(p), h-2-tag(p))+1+tag(p)+1 \\ &= h \end{aligned}$$

5 Výškou ohodnocené binární vyhledávací stromy

Na rozdíl od dříve popsaných AVL stromů s relaxovaným vyvažováním v kapitole 4, kdy si strom "pamatuje" v každém uzlu, jak moc je tento uzel nevyvážen, a v případě vyšší hodnoty tagu nějakého uzlu je tento uzel vyvážen na několik kroků rebalanceru, jsou výškou ohodnocené binární vyhledávací stromy výrazně jednodušším a elegantnějším řešením. Neukládají si nikde informace o míře nevyváženosti uzlu, pouze si zaznamenávají, že ten či onen vrchol je nevyvážený. Rebalancer se orientuje pouze podle výšek jednotlivých uzlů. Velkou výhodou tohoto přístupu je skutečnost, že některá vyvažování, která by nastala u klasických AVL stromů, mohou být díky odložení vyvažování úplně vynechány. Nevyvážený strom může být operacemi insert nebo delete uveden zpátky do rovnováhy bez použití rebalanceru.

Aby rebalancer poznal místo, kde je strom nevyvážený, updater zneplatňuje hodnoty výšek uzlů. Nastavuje je na -1. Rebalancer pak najde takový uzel, který lze vyvážit. Pokud je potřeba, tak provede jednoduchou či dvojitou rotaci známou z klasických AVL stromů, a nahradí hodnoty výšky -1 skutečnou výškou vyvažovaného uzlu. Vyvažovací operace zůstane pouze lokální záležitostí a ovlivní pouze určité malé a konstantní množství uzlů.

Kapitola dle [10].

5.1 Definice

Ke každému uzlu u binárního vyhledávacího stromu přiřadíme číslo nazvané *hodnota výšky*, označíme ho $hv(u)$. Toto číslo se rovná buď -1 nebo $height(u)$ (výška uzlu u.)

Definice: *Pokud* $hv(u) \neq -1$ *(čili, když* $hv(u) = height(u)$*), pak pro syny uzlu* u *(označme je* v_1 *a* v_2*) musí platit:*

1. $hv(v_1) = height(v_1)$, $hv(v_2) = height(v_2)$ *a současně*
2. *rozdíl výšek uzlů* v_1 *a* v_2 *je nejvýše jedna, tj.:* $|height(v_1) - height(v_2)| \leq 1$.

Binární vyhledávací strom splňující tyto podmínky nazveme *výškou ohodnocený binární vyhledávací strom (height-valued tree).*

Říkáme, že uzel u způsobuje *konflikt vyváženosti* právě tehdy, když $hv(u) = -1$. V opačném případě říkáme, že uzel u je *vyvážený*.

Výškou ohodnocený binární vyhledávací strom T je *úplně vyvážený*, pokud jsou všechny jeho uzly vyvážené. V tomto případě je zřejmé, že T je AVL stromem.

Myšlenka použití výškou ohodnocených binárních vyhledávacích stromů spočívá v tom, že operace insert a delete nastavují po cestě k požadovanému listu *hodnotu výšky* na -1 ve všech uzlech, kde byla tato hodnota různá od -1. Hodnoty -1 jsou pak postupně vyvažovacím procesem nahrazovány pravými výškami za použití standardních vyvažovacích operací známých z AVL stromů.

5.2 Operace

5.2.1 Operace INSERT

Procházíme stromem a hledáme list l obsahující klíč k. V každém vnitřním uzlu u, kterým projdeme, kde je hodnota $hv(u) \neq -1$, nastavíme hodnotu $hv(u) = -1$ (zneplatníme výšku - a to i v případě kdy vkládáme uzel, který již ve stromu je). Pokud najdeme klíč k, ukončíme proces.

Neúspěšné vyhledávání skončí v listu. Nazvěme jej c. Vytvoříme nový vnitřní uzel p na místě c. Jeho syny budou listy c a nově vytvořený list, který obsahuje klíč k. Pořadí synů je takové, že list obsahující menší klíč se stane levým synem uzlu p. Hodnota uložená v uzlu p je kopií klíče levého syna p. Hodnota výšky nově vytvořeného listu je nastavena na 0 a hodnota výšky uzlu p je nastavena na 1.

5.2.2 Operace DELETE

Stejně jako v případě operace insert procházíme stromem a hledáme list l obsahující klíč k. V každém vnitřním uzlu u, kterým projdeme, kde je hodnota $hv(u) \neq -1$, nastavíme hodnotu $hv(u) = -1$. Pokud nenajdeme klíč k, ukončíme proces. V opačném případě list (označme jej opět c) obsahující klíč k odstraníme ze stromu a jeho otce nahradíme sourozencem uzlu c.

Pozorování 1: *Pokud aplikujeme na výškou ohodnocený binární vyhledávací strom sekvenci operací insert a delete, získáme opět výškou ohodnocený binární vyhledávací strom.*

5.3 Vyvažování

Úkolem vyvažování výškou ohodnoceného binárního vyhledávacího stromu je odstranění všech konfliktů vyváženosti ze stromu. Toto by mělo být navíc možné pouze za použití malých lokálních transformací, které by umožňovaly kromě souběžného vyhledávání ve stromu také souběžné vkládání a mazání.

U tohoto typu stromu probíhá řešení konfliktů vyváženosti zezdola-nahoru. Čili konflikt v uzlu u bude řešen pouze tehdy, když oba dva podstromy uzlu u nebudou obsahovat žádné konflikty. Toto se dá zjistit pouhým podíváním se na oba syny uzlu u, protože oba podstromy neobsahují žádný konflikt právě tehdy, když hodnoty výšky obou dvou synů se rovnají jejich pravým výškám.

Nechť p je uzel, který způsobuje konflikt vyváženosti, $hv(p) = -1$. Nechť c a s jsou syny uzlu p takoví, že $hv(c) \neq -1$ a $hv(s) \neq -1$. Nechť c je levým synem. p nazveme *kořenem operace před vyvažováním*. Nyní máme dvě možnosti.

Případ 1: Ukončující operace.

$|height(c) - height(s)| \leq 1$.

V tomto případě platí vyvažovací podmínka AVL stromu a pouze nastavíme

$$hv(p) = max(hv(c), hv(s)) + 1.$$

Případ 2: Rotace.

$|height(c) - height(s)| > 1$.

V tomto případě použijeme jednoduchou nebo dvojitou rotaci přesně stejným způsobem jako bychom použili u klasických AVL stromů. Předpokládejme, že $height(c) > height(s)$. Opačný případ je symetrický. Dále označme levý podstrom uzlu c jako A a pravý podstrom jako B. Podle výšek stromů A a B mohou nastat dva dílčí podpřípady.

Případ 2a: Jednoduchá rotace.

$height(A) \geq height(B)$.

V tomto případě provedeme jednoduchou rotaci (viz Obrázek 4). Po rotaci jsou uzly p, c, s a podstromy A, B označeny p',c',s',A',B'. Hodnoty výšky uzlů c' a p' nastavíme na -1. Uzel c' nazveme *kořenem operace po vyvažování*.

Případ 2b: Dvojitá rotace.

$height(A) < height(B)$.

Kořen podstromu B označíme g a jeho podstromy jako B_1 a B_2. V tomto případě provedeme dvojitou rotaci (viz Obrázek 5). Hodnoty výšky uzlů g' a p' nastavíme na -1. Uzel g' nazvene *kořenem operace po vyvažování.*

Pro výškou ohodnocené binární vyvažovací stromy dokážeme následující invariant:

Invariant 2: *Pokud má uzel hodnotu výšky různou od* -1*, pak pod tímto uzlem nemůže existovat žádný konflikt. Navíc, pokud strom obsahuje nějaký konflikt, pak vždy existuje uzel způsobující konflikt, který má oba dva syny bez konfliktu.*

Důkaz: Nejprve dokážeme první část invariantu indukcí přes počet provedených operací. V triviálním případě, kdy na zadaném výškou ohodnoceném stromu nebyla vykonána žádná operace, jsou hodnoty výšky všech uzlů různé od -1 a tvrzení tedy platí. V indukčním kroku předpokládejme, že tvrzení až do nějakého okamžiku platí a zvažme všechny možné operace, které mohou nastat. Operace *search* nemění hodnoty $hv(u)$. Pokud je následující operací update stromu (*insert* či *delete*), pak libovolný uzel u s hodnotou $hv(u) \neq -1$ po provedené operaci musí ležet mimo cestu hledání, a žádný uzel v jeho podstromu nebyl operací ovlivněn. Tedy dle indukčního předpokladu tvrzení platí. Pokud je následující operací vyvažovací operace v nějakém vrcholu p, pak, protože $hv(c) \neq -1$ a $hv(s) \neq -1$ (jinak by se nevyvažoval vrchol p), z indukčního předpokladu plyne, že libovolný uzel u z podstromů c a s má $hv(u) \neq -1$. Hodnotu výšky uzlů s' v případě 2a a c' a s' v případě 2b můžeme bezpečně nastavit různou od -1 aby invariant stále platil. Druhý bod tvrzení plyne z faktu, že hodnota výšky listů je vždy různá od -1.

5.4 Složitost

Předpokládejme, že v určitém bodě máme výškou ohodnocený strom, který je úplně vyvážený. V této sekci dokážeme, že vyvažování tohoto stromu je logaritmické. Konkrétně, pokud M operací typu update je provedeno na stromu, který byl úplně vyvážený, pak k jeho opětovnému vyvážení potřebujeme $O(Mlog(N+M))$ kroků rebalanceru, kde N je velikost stromu, když byl naposledy úplně vyvážený. Dokážeme to pomocí amortizované složitosti bez ohledu na to, jak za sebou následují jednotlivé operace.

Pro účely analýzy rozdělíme uzly do tří kategorií: *pasivní*, *aktivní* a *hyperaktivní*. Na začátku jsou všechny uzly pasivní. Poté, co se uzel účastní nějaké operace, mění stav následujícím způsobem. List je vždy pasivní. Při operaci update se stav všech uzlů ležících na vyhledávací cestě změní na aktivní, nezávisle na tom, jaký byl předtím. Ukončující vyvažovací operace změní stav kořene operace na pasivní. Ostatní vyvažovací operace ponechávají kořen operace v původním stavu a uzel u' se stává hyperaktivní. Protože vyvažování postupuje zezdola-nahoru, dostáváme následující pozorování.

Pozorování 3:

1. *Pasivní uzel má pouze pasivní syny.*
2. *Synové hyperaktivního uzlu jsou vždy pasivní.*

Tvrzení 4: *Na libovolné cestě z kořene stromu do uzlu se nachází nejvýše jeden hyperaktivní uzel.*

Důkaz: Hyperaktivní uzel může vzniknout pouze rotací. Představme si cestu z kořene přes hyperaktivní uzel, který je nově vytvořen rotací. Jediný uzel na této cestě, který mohl být hyperaktivní před provedením rotace, je kořen operace před vyvažováním, protože dle Pozorování 3 uzly výše ve stromu nemají pasivní syny.

Maximální výškový rozdíl mezi dvěma podstromy uzlu u, které jsou oba AVL stromy, označíme jako $mhd(n)$, kde n je počet uzlů ležících ve stromu s kořenem u. Někdy místo $mhd(n)$ budeme používat $mhd(u)$. Protože nejmenší možný podstrom je list, $mhd(n)$ se musí rovnat výšce nejvyššího AVL stromu s $n-2$ uzly. Jak jsme ukázali v kapitole 3 je minimální počet uzlů v AVL stromu o výšce h roven $F_{h+3}-1$. Z nerovnosti $F_{h+3}-1 \leq n-2$ lze pomocí Tvrzení 3 z kapiloty 3 dokázat, že $h \leq \lfloor log_{\varphi}(\sqrt{5n})\rfloor - 3 = mhd(n)$, kde φ je zlatý poměr $(1+\sqrt{5})/2$.

Tvrzení 5: *Vyvažovací operace může zvýšit rozdíl výšek maximálně u jednoho uzlu, který není zahrnut do operace, a to nejvýše o 1.*

Důkaz: Je zřejmé, že toto se týká pouze uzlů ležících mezi místem vyvažování a kořenem. Nechť $u_1, u_2, ..., u_k$ jsou tyto uzly, kde u_k je kořen stromu a u_1 je kořen operace před vyvažováním. Všimněme si, že výška podstromu uzlu u_1 se nemůže zvýšit. Zůstane buď stejná nebo se sníží o 1, a to jak u jednoduché, tak u dvojité rotace. Vzhledem k tomu, že výška je definována rekurzivně, tak jakmile hodnota nějakého u_i zůstane stejná, zůstanou zachovány i výšky všech uzlů ležících nad u_i.

Předpokládejme tedy dále, že výška uzlu u_1 se snížila. Označme t nejmenší index takový, že výška uzlu u_t se nezměnila. Protože se výška všech uzlů u_j, kde $j < t$, snížila, podstrom obsahující u_1 byl tím větším před operací. Čili rozdíl výšek se v u_j sníží. Jediný uzel, kde se může rozdíl výšek zvýšit je tedy u_t a zvýší se o 1.

Definice: *Vyvažovací operaci a jí příslušný uzel u_t z Tvrzení 5 a jeho důkazu nazveme operací zvyšující rozdíl výšek v uzlu u_t.*

Nyní definujeme potenciálovou funkci, která bude vyjadřovat nevyváženost stromu vůči standardnímu AVL stromu. Uzel u se postupem času stává aktivní, poté nějakou dobu zůstává aktivní a zůstává i kořenem svého podstromu. Během této doby mohou uzlem u procházet různé updaty a mohou se provádět rotace zvyšující rozdíl výšek v uzlu u. Obojí přispívá k nevyváženosti uzlu u. Na druhou stranu nevyváženost uzlu u nemůže být vyšší než $mhd(u)$ (největší možný rozdíl výšek). Pro aktivní uzel u označme počet updatů procházející uzlem u, od doby kdy u byl naposledy neaktivní, jako $upd(u)$, počet vyvažovacích operací zvyšujících rozdíl výšek v uzlu u, od doby kdy u byl naposledy neaktivní, jako $hio(u)$ a rozdíl výšek dvou podstromů uzlu u od doby, kdy u byl naposledy neaktivní, jako $lna(u)$.

Definice: *Potenciál výškou ohodnoceného binárního vyhledávacího stromu T se rovná součtu potenciálů všech jeho uzlů. Potenciál uzlu u se syny v a w je definován takto:*

Stav	*Potenciál*
Pasivní	*0*
Aktivní	$2[min(upd(u) + hio(u) + lna(u), mhd(u)) - 1] + 1$
Hyperaktivní	$2[\|height(v) - height(w)\| - 1] + 1$

kde $[x] = x$, *pokud* $x \geq 0$ *a* $[x] = 0$ *jinak.*

Pro uzel u budeme výrazu $2[mhd(u) - 1] + 1$ říkat *maximální potenciál uzlu u*. Proč jsme zvolili potenciálovou funkci právě takto? -1 zohledňuje vlastnost AVL stromu, který povoluje, aby se výšky podstromů lišily o 1. $+1$ u aktivního a hyperaktivního uzlu je proto, že i přesto, že uzel je vyvážený, potřebujeme snížit potenciál na ukončující vyvažovací operaci. Násobení dvěma zajišťuje to, aby vyvažovací operace při snížení rozdílu výšek o 1 uvolnila 2 jednotky. Jednu na potřebné snížení potenciálu a druhá jde na "účet" uzlu, který se stane při této vyvažující operaci aktivní.

Nyní potřebujeme dokázat, že libovolný update zvýší potenciál stromu nejvýše

o $O(log(n))$, kde n je aktuální velikost stromu, zatímco vyvažovací operace sníží potenciál nejméně o 1.

Lemma 8: *Pro libovolnou cestu z listu do kořene nechť $u_1, \ldots, u_k$ jsou uzly na této cestě, které mají nenulový potenciál, ale menší než maximální. Označme velikost podstromu u_i jako n_i. Uzly jsou uvedeny v pořadí, v jakém se nacházejí na cestě. Pak $k \leq \lfloor \log_\varphi \left(\sqrt{5}(n_k)\right) \rfloor - 3 + \log_\varphi (2(n_k + 1))$.*

Důkaz: Vezměme si uzel u_k. Protože jeho potenciál není maximální, méně než $mhd(n_k) - 1$ updatů prošlo uzlem u_k od doby, kdy byl naposledy pasivní nebo hyperaktivní. Protože, dle Pozorování 3, uzel, který je pasivní nebo hyperaktivní, má vyvážené podstromy, u_k měl tedy vyvážený podstrom s výškou nejméně $h = k - 1 - upd(u_k) \geq k - mhd(n_k)$. Takový strom obsahoval určitě nejméně $F_{h+3} - 1$ uzlů. Tedy $n_k \geq F_{h+3} - 1 - upd(u_k) \geq F_{h+3} - 1 - mhd(n_k) - 1$.

Protože $F_i \geq \varphi^{i-2}$, pak

$$\varphi^{k-mhd(n_k)} - 2 \leq F_{k-1-mhd(n_k)+3} - 2 \leq n_k + mhd(n_k).$$

Tedy

$$\varphi^k \leq \varphi^{mhd(n_k)}(n_k + mhd(n_k) + 2).$$

Z čehož po dosazení za $mhd(n_k)$ a triviálního faktu, že $mhd(n_k) \leq n_k$ plyne výsledek.

Lemma 9: *Pro libovolnou cestu z listu do kořene nechť $u_1, \ldots, u_k$ jsou uzly na této cestě, které mají maximální potenciál. Uzly jsou uvedeny v pořadí, v jakém se nacházejí na cestě. Pak celkové zvýšení potenciálu těchto uzlů způsobené vložením pod u_1 je omezeno $2(mhd(n_k + 1) - mhd(n_1))$.*

Důkaz: Nejprve si všimněme, že u_i leží v podstromu uzlu $u_{i+1}, i \in \{1, \ldots, k-1\}$. Proto $n_i < n_{i+1}$ a tedy $n_i + 1 \leq n_{i+1}$ a $mhd(n_i + 1) \leq mhd(n_{i+1})$, protože se jedná o neklesající funkci. Nyní spočítáme celkový přírůstek k potenciálu.

$$\begin{aligned} I &= \textstyle\sum_{i=1}^{k}(2(mhd(n_i + 1) - 1) + 1) - \sum_{i=0}^{k}(2(mhd(n_i) - 1) + 1) \\ &= 2\textstyle\sum_{i=0}^{k}(mhd(n_i + 1) - mhd(n_i)) \\ &\leq 2(mhd(n_k + 1) - mhd(n_1)) \end{aligned}$$

podle výše uvedeného pozorování.

Lemma 10: *Pro libovolnou cestu z listu do kořene nechť $u_1, \ldots, u_k$ jsou uzly na této cestě, které mají nulový potenciál. Uzly jsou uvedeny v pořadí, v jakém se nacházejí na cestě. Pak $k \leq \lfloor \log_\varphi \left(\sqrt{5}(n_k + 1) + 1\right) \rfloor - 2$.*

Důkaz: Protože uzly mají nulový potenciál, jsou všechny pasivní. u_k je tedy kořenem podstromu, kde jsou všechny uzly pasivní a tudíž se jedná o AVL strom. Proto dle Tvrzení 2 kapitoly 3 platí: $F_{k-1+3} - 1 \leq n_k$ z čehož, za použití Tvrzení 4 kapitoly 3, vyplývá výsledek.

Lemma 11: *Update zvýší potenciál nejvíce o* $O(log(N + M))$.

Důkaz: Podle Tvrzení 4 je na cestě updatu pouze jeden hyperaktivní uzel. Tento uzel se stane aktivním. Zvýšení potenciálu u tohoto uzlu je určitě omezeno $O(mhd(N))$.

Zbytek uzlů na cestě jsou buď aktivní nebo pasivní. Aktivní uzly zůstanou aktivními, ale protože se změnil počet uzlů v jejich podstromech, může se zvýšit jejich potenciál. Aktivní uzly mohou mít maximální potenciál nebo ne.

Podle Lemma 8 je počet uzlů na cestě, které nemají maximální potenciál, logaritmicky omezen, a tudíž celkový přírůstek potenciálu u těchto uzlů je logaritmicky omezen.

Dle Lemma 9 je logaritmicky omezen i přírůstek potenciálu u uzlů, které již měly maximální hodnotu potenciálu. Nakonec pasivní uzly se stanou aktivními a jejich potenciál se zvedne z 0 na konstantní hodnotu (nejvýše 3).

Dle Lemma 10 je počet těchto uzlů logaritmicky omezen, tedy i celkový nárůst potenciálu za tyto uzly je logaritmicky omezený. Z toho vyplývá, že update zvýší potenciál stromu nejvýše o $O(|T|)$, kde $|T|$ je aktuální velikost stromu. To je určitě menší než $O(log(N + M))$.

Lemma 12: *Vyvažovací operace sníží potenciál stromu nejméně o jedna.*

Důkaz: Existují pouze tři vyvažovací operace (ukončující, jednoduchá a dvojitá rotace). Probereme zvlášť tyto tři případy.

Ukončující. Pouze nejvyšší uzel operace změní svůj stav z aktivního či hyperaktivního na pasivní. Podle definice mají aktivní a hyperaktivní uzly nenulový potenciál a pasivní uzly mají potenciál 0.

Rotace. U obou typů rotace mění kořen podstromu před provedením rotace (označ. p) svůj stav z aktivního na hyperaktivní (nebyl-li již hyperaktivní). Když byl uzel p naposledy hyperaktivní nebo pasivní, měl potenciál $2[x-1]+1$, kde x byl rozdíl výšek jeho podstromů. Od té se rozdíl výšek podstromů uzlu p mohl pouze zvýšit. V nejhorším případě o 1 za každý procházející

update a o 1 za každou operaci zvyšující rozdíl výšek v uzlu p. Čili maximální rozdíl výšek podstromů je $upd(p) + hio(p) + lna(p)$. Na druhou stranu je uzel p otcem dvou vyvážených podstromů, takže rozdíl výšek těchto podstromů nemůže překročit $mhd(n)$, kde n je velikost podstromu s kořenem p. Takže změna stavu uzlu p z aktivního na hyperaktivní nezvýší potenciál stromu. Ve zbytku důkazu můžeme předpokládat, že p je hyperaktivní.

Jednoduchá rotace. Předpokládali jsme, že $height(A) \geq height(B)$ (viz Obrázek 4). Protože podstrom s kořenem c je vyvážený $height(A) = height(B)$ nebo $height(A) = height(B) + 1$. Rozeberme tyto případy zvlášť.

$height(A) = height(B) + 1$. Rozdíl výšek podstromů p' je o dva menší než byl rozdíl výšek u p. To nám dává snížení potenciálu o čtyři jednotky. Uzel c' se stal aktivním, ale rozdíl výšek jeho podstromů je 0. Stačí k němu tedy přesunout jednu jednotku. Pokud vlivem updatu dojde ke zvýšení rozdílu výšek podstromů u nějakého uzlu, který není zahrnut do rotace, tento uzel je dle Tvrzení 5 pouze jeden a zvýšení je také pouze o jedna. K takovému uzlu tedy stačí přesunout dvě jednotky. Celkový potenciál je snížen.

$height(A) = height(B)$. Výška podstromu před rotací a po rotaci je stejná. Proto operace nezvýší rozdíl výšek u žádného jiného uzlu ve stromu. Rozdíl výšek podstromů u uzlu p' je o jedna menší, čímž se uvolní dvě jednotky. Jedna z nich je přesunuta kvůli změně stavu z pasivního na aktivní na uzel c' a druhá tvoří požadovaný úbytek potenciálu.

Dvojitá rotace Všimněme si, že uzel s' zůstává pasivní (viz Obrázek 5). Rozdíl výšek u p' je o dva nižší než byl u p a výška podstromu se sníží o jedna, takže důvod je zde stejný jako u větve "$height(A) = height(B) + 1$".

Věta 13: *Vyvažování je amortizovaně logaritmické.*

Důkaz: Předpokládejme, že máme vyvážený AVL strom o velikosti $|N|$. Potenciál takového stromu je 0. Podle Lemma 11 update zvýší potenciál stromu nejvýše o $O(log(M + N))$ a podle Lemma 12 vyvažovací operace sníží potenciál minimálně o jedna.

6 Implementace

Pro vzájemné srovnání byl implementován v paralelním prostředí klasický AVL strom a relaxovaný strom popsaný v kapitole 5. Tento algoritmus byl vybrán z důvodu dokázané logaritmické časové složitosti vyvažování a kvůli své schopnosti zapomínat rotace. Program nicméně může být snadno rozšířen i o další algoritmy relaxovaného vyvažování. V této kapitole uvedeme některé podstatné aspekty implementace.

6.1 Správa vyvažovacích požadavků

Modifikující operace v relaxovaných verzích binárních vyhledávacích stromů narušují vyváženost některých uzlů stromu. Současně však v těchto uzlech ponechávají informaci, která pomůže vyvažovacímu procesu (rebalanceru) vybrat ten správný typ transformace a strom opět vyvážit. V této části textu se zaměříme na to, jak rebalancer nachází právě ty nevyvážené uzly, na kterých poté provádí své kroky.

V literatuře se objevují dva zásadně odlišné přístupy detekce takovýchto uzlů (nebo též vyvažovacích požadavků). Jednou z možností je náhodně procházet stromem od kořene směrem k listům a hledat uzel, u kterého je možné provést nějaký vyvažovací krok [13]. V každém uzlu se náhodně rozhodneme, jestli budeme pokračovat do pravého nebo levého podstromu. Když se dostaneme do listu, začínáme znovu v kořeni stromu. Výhodou tohoto přístupu je, že správa vyvažovacích požadavků nepotřebuje žádnou paměť navíc. Na druhou stranu u téměř vyváženého stromu může být problém nalézt posledních několik požadavků. Proto někteří autoři navrhují během vkládání a mazání označit uzly na cestách mezi kořenem a listem, po kterých tyto operace šly, a následný průchod stromem pak omezit pouze na tyto cesty.

Druhým návrhem je použití tzv. problémové fronty (problem queue) [8, 3]. Kdykoliv nějaký uzel způsobí nevyváženost stromu, je vložen do fronty, ze které potom vyvažovací proces tyto uzly odebírá. Nemusí se jednat pouze o jeden spojový seznam vrcholů, které je třeba vyvážit. Front může být více, třeba i s možností prioritního přidání na začátek fronty.

V naší implementaci jsme zvolili problémovou frontu a to hned z několika důvodů. Vyvažovací proces snadno pozná, kdy je strom vyvážený (fronta je prázdná).

U implementovaného algoritmu relaxovaného vyvažování postupuje vyvažovací proces zezdola-nahoru, a proto by zde první varianta detekce vyvažovacích požadavků postupující z kořene směrem dolů byla neefektivní.

Každý uzel (požadavek na vyvážení) by se měl v problémové frontě vyskytovat pouze jednou. Implementované modifikující operace zneplatňují hodnoty výšek a nastavují je na -1. Aby byl každý uzel způsobující konflikt vyváženosti v problémové frontě pouze jednou, stačí ho vložit do fronty pouze v případě, kdy se mění výška uzlu ze skutečné výšky na -1.

6.1.1 Implementace problémové fronty

V naší implementaci máme 32 problémových front, kde každá fronta odpovídá vzdálenosti uzlu od kořene stromu. Rebalancer vybírá přednostně uzly, které jsou od kořene nejdále, a tím minimalizuje možnost, že by z fronty získal uzel, na který by nemohl aplikovat vyvažovací operaci (uzel s nevyváženými syny). Pokud ale taková možnost nastane, rebalancer vrátí uzel na konec odpovídající fronty a vezme jiný uzel, který se pokusí vyvážit. Aby rebalancer věděl ze které fronty má vzít uzel, přidali jsme v implementaci ukazatel na nejnižší obsazenou frontu. Do fronty požadavků přistupuje současně více procesů, proto je potřeba frontu zamykat. Fronta je zamykána exkluzivně vždy pro jeden proces, který vloží požadavek na konec příslušné fronty a případně updatuje ukazatel na nejnižší obsazenou frontu, nebo vyjme požadavek ze začátku nejnižší obsazené fronty - a pokud tuto frontu vyprázdní, updatuje ukazatel na nejnižší obsazenou frontu. Aby doba zamčení problémové fronty byla co nejkratší, je potřeba, aby odebírání požadavku ze začátku a přidávání požadavku na konec bylo co nejrychlejší. Každá fronta je reprezentována spojovým seznamem požadavků a obsahuje 2 speciální ukazatele na začátek a konec spojového seznamu.

6.2 Zamykání stromu

V paralelním prostředí přistupuje do stromu současně více procesů a každý z nich může vykonávat jednu z operací search, insert, delete a krok rebalanceru. Nemělo by docházet k situacím, kdy se vyhledávání rozhodne pokračovat třeba do levého podstromu a mezitím vyvažovací proces změní strukturu stromu. Vyhledávání by se pak ocitlo na špatné cestě a dalo by špatnou odpověď. Dále by nemělo docházet

k situacím, kdy se procesy navzájem zablokují. Je třeba vyloučit vzájemné interference procesů ve stromu. Toho lze docílit například zamykáním určitých částí stromu a vyloučením ostatních procesů z operací nad těmito částmi.

Dále v textu předpokládejme, že máme množinu operací insert, delete, search, rebalance. Nazvěme ji S. Jednotlivé operace trvají konečně dlouhou dobu a mohou být různě rychlé (i dvě stejné operace se stejnou hodnotou klíče). Potom potřebujeme, aby paralelní vykonání těchto operací trvalo pouze konečný čas, a tedy žádná operace se nezablokovala, a aby výsledky vrácené jednotlivými operacemi byly správné.

Vzhledem k tomu, že nemůžeme předem přesně určit pořadí, v jakém budou jednotlivé operace vykonány, budeme požadovat, aby vrácené výsledky odpovídaly výsledkům nějaké sekvenčně vykonávané setříděné posloupnosti operací z S.

Naše řešení je podobné jako v [4], kdy používáme zámky pro uzamčení přístupu k jednotlivým uzlům stromu. V případě AVL stromů i v případě relaxované varianty používáme tři typy zámků. Tzv: R-lock, což je zámek pro čtení, W-lock (zámek pro zápis) a X-lock (exkluzivní zámek).

1. Několik různých procesů může zamknout uzel pro čtení (R-lock), i když je již zamčený pro zápis (W-lock).

2. Pouze jediný proces může zamknout uzel pro zápis, i když je uzel uzamčen pro čtení.

3. Pouze jediný proces může zamknout uzel exkluzivně, pokud uzel není žádným jiným způsobem uzamčen.

4. Zámky pro zápis mohou být konvertovány na exkluzivní zámky, pokud uzel není uzamčen pro čtení.

Abychom předešli deadlockům, všechny uzly jsou zamykány ve směru seshora-dolů a zleva doprava. Jak lze vidět, celé řešení je navrženo ve snaze co nejvíce upřednostnit operace typu search, které se navzájem neblokují.

V sériovém prostředí, kdy do stromu přistupuje jeden proces, není třeba uzly zamykat, protože nemůže k žádnému nechtěnému ovlivňování dojít.

6.2.1 AVL strom

Operace SEARCH:

Při cestě dolů stromem používá tzv. *zdvojování zámků* pro čtení. Nejprve zamkne kořen stromu pro čtení (R-lock). Porovná hodnotu klíče s hodnotou v kořeni a zvolí syna kořene, kam bude pokračovat. Poté zamkne pro čtení zvoleného syna kořene. Až pak uvolní zámek kořene a pokračuje ve vyhledávání. Má zamčené maximálně dva uzly pro čtení. Takto postupuje až k listu a podle hodnoty klíče v listu vrátí odpověď. Těsně předtím odemkne list.

Operace INSERT:

Cestou ke kořeni zamyká všechny uzly po cestě pro zápis. Tím vyloučí ostatní modifikující operace ze stromu. Operace search probíhat může. V listu provede vložení nebo zahlásí chybu. Pokud zahlásí chybu, strom nebyl modifikován a operace pouze uvolní všechny zámky. V opačném případě zkonvertuje W-lock v listu na X-lock, provede vložení a uvolní X-lock. Nyní má zamknutého otce uzlu, pod který se vkládalo a všechny jeho předky pro zápis.

Operace postupuje směrem zpět ke kořeni a uvolňuje zámky. Pokud dojde u nějakého uzlu k porušení podmínky vyváženosti, je třeba znovu vyvážit strom. Dochází k modifikaci struktury stromu, a je proto třeba vyloučit operace search z modifikovaných uzlů. Uzly, které se budou účastnit potřebné rotace jsou exkluzivně uzamčeny (u těch, co byly zamčeny pro zápis je zámek zkonvertován na exkluzivní), rotace je provedena a exkluzivní zámky uvolněny. Poté je odemknuta celá cesta až ke kořeni.

Operace DELETE:

Z hlediska zamykání postupuje úplně stejně jako operace INSERT poze s tou výjimkou, že po ukončené rotaci neodemyká celou cestu ke kořeni, ale postupuje po té cestě vzhůru a případně provádí další rotace.

6.2.2 Relaxovaný strom

Na rozdíl od klasického AVL stromu, kde může současně operovat ve stromu pouze jedna modifikující operace, u relaxovaného stromu stačí zamykat pouze malé konstantní množství zámků, a proto může ve stromu probíhat současně více modifikujících operací. Navíc jsou implementačně modifikující operace jednodušší.

Operace SEARCH:

Je implementována stejně jako u klasické verze. Zdvojování čtecích zámků, uvolnění zámku listu, report výsledku.

Operace INSERT:

Používá zdvojování zámků pro zápis. Pokud nalezne klíč ve stromu, uvolní zámek listu a skončí. V opačném případě konvertuje tento zámek na exkluzivní, vloží klíč do stromu, a nakonec uvolní exkluzivní zámek. Vložení je provedeno tak, že je uzamčený list l zkonvertován na vnitřní uzel a jsou k němu přidány 2 nové listy. Díky tomu není třeba upravovat ukazatele v otci listu l, a proto otce l není třeba zamykat.

Operace DELETE:

Při postupu dolů stromem používá ztrojování zámků pro zápis (na rozdíl od operace INSERT potřebuje mít zamčené poslední dva uzly). Předpokládejme, že klíč se shoduje s klíčem nalezeného listu (l). Nyní je zamčen pro zápis tento list a jeho otec. Tyto zámky jsou zkonvertovány na exkluzivní, poté je exkluzivně uzamčen sourozenec listu l, aby došlo k vyloučení všech ostatních operací z modifikovaných uzlů.

Mazání je opět provedeno tak, že smazány jsou uzly l a jeho sourozenec, předtím jsou ovšem potřebné hodnoty překopírovány do otce uzlu l. Není proto třeba zamykat žádný jiný uzel. Pokud na mazané uzly existuje odkaz v problémové frontě, samotné mazání provádí až rebalancer.

Operace REBALANCE:

Rebalancer nejprve získá vyvažovací požadavek na uzel u. Poté zamkne uzel u pro zápis. Pokud je uzel u označen jako smazaný, smaže ho a vezme další vyvažovací požadavek. Pokud má uzel u kladnou výšku, je již vyvážen a rebalancer vezme další požadavek.

Nyní má rebalancer uzel u, který není vyvážený, zamčený pro zápis. Zamkne pro zápis jeho dva syny. Poté zkontroluje výšky těchto synů. Pokud nemají oba dva synové nezáporné výšky, uzel u nelze vyvážit, požadavek je vrácen do fronty a rebalancer vezme další požadavek.

Synové uzlu u mají nezáporné výšky, tudíž jsou kořeny AVL stromů. Může se začít vyvažovat. Pokud je v uzlu u splněna podmínka vyváženosti, je pouze správně nastavena výška uzlu a rebalancer vezme další požadavek. Pokud není splněna pod-

mínka vyváženosti, je provedena příslušná rotace. Předtím jsou ovšem zámky pro zápis konvertovány na exkluzivní zámky a jsou exkluzivně zamčeny další potřebné uzly zapojené do rotace.

Kořen operace po vyvažování a hyperaktivní syn (viz kapitola 5) jsou vloženy do problémové fronty a rebalancer vezme další požadavek.

Pokud rebalancer nemá žádný požadavek ve frontě, je uspán, dokud požadavek nepřijde. Poté, co je na stromu provedena celá sada operací, je signalizován rebalanceru konec. Rebalancer poté dále vyvažuje frontu požadavků a až ji vyčerpá, také ukončí svou činnost.

6.2.3 Měření času

Pro implementaci jednotlivých klientů, kteří paralelně přistupují do vyhledávacího stromu, jsme zvolili vlákna místo samostatných procesů. Výhod vláken je hned několik. Sdílejí společný adresní prostor a tedy i strom a seznam operací k provedení. Dále odpadne vzájemná komunikace mezi klientem a serverem se stromem a přepínání vláken na procesoru je výrazně rychlejší, protože není třeba měnit adresový prostor. Strom je potřeba při paralelním přístupu zamykat. Tato práce si nekladla za cíl změřit náročnost zamykání stromové struktury, a proto se neměří zvlášť systémový čas strávený obsluhou zamykacích požadavků. Měříme pouze celkový strávený čas na procesoru za určité množství operací. Ve výchozím nastavení měříme celkový čas na procesoru pro každých 1000 operací, nezávisle na tom, kolik operací provedl který klient. K měření používáme funkci *clock_gettime()*.

Nevýhodou tohoto měření je fakt, že je do výsledných časů započítána i režie spojená s měřením času. Při měření po 1000 operacích to znamená, při zvoleném počtu operací, 6000 krát změřit čas. Vzhledem k tomu, že jsme zjistili, že systém zvládne provést 6 miliónů měření času za vteřinu a vzhledem k délce našich experimentů, které trvají přes 30 vteřin, lze režii potřebnou na měření času zanedbat.

V případě testu jednotlivých operací je čas měřen 6 miliónkrát, což se již v průměrném čase na jednu operaci projeví. V případě operace SEARCH, která je nejrychlejší a tudíž nejvíce ovlivněna, tvoří samotné měření času cca 4% celkového času. To je ještě statisticky zanedbatelná hodnota. Navíc vzhledem k tomu, že nás nezajímají absolutní čísla o rychlosti jednotlivých operací, která vypovídají spíše o výkonnosti hardware, ale vzájemné porovnání jednotlivých operací mezi

AVL stromem a relaxovaným stromem, výsledky jsme nijak neupravovali a zde prezentované časy jsou uvedené včetně režie na měření času.

6.3 Generování dat pro testy

Pro generování vstupních dat testu slouží jednoduchý textový konfigurační soubor, který obsahuje následující parametry testu:

SEED	Náhodné číslo sloužící k počáteční inicializaci pseoudonáhodného generátoru čísel, aby bylo možné generovat stejná data na různých počítačích.
MAX_KEY	Maximální možná hodnota vygenerovaného klíče.
LEAVES	Počet listů v počátečním stromu.
OPERATIONS	Číslo udávající celkový počet provedených operací.
P_SEARCH	Číslo udávající pravděpodobnost operace SEARCH v %.
P_INSERT	Číslo udávající pravděpodobnost operace INSERT v %.
P_DELETE	Celé číslo udávající pravděpodobnost operace DELETE v %.

Součet pravděpodobností jednotlivých operací musí být roven 100. U všech zde prezentovaných testů byla počáteční velikost stromu nastavena na 1 milión klíčů a hodnota MAX_KEY (velikost univerza) na 100 miliónů.

Náhodný klíč je generován z uzavřeného intervalu 1-MAX_KEY. Generování vstupních dat pro testy pak probíhá tak, že nejprve je vygenerován vstupní strom s přesně LEAVES klíči. Pokud je generovaný klíč již obsažen ve stromu, je místo něj generován jiný. Poté jsou generovány jednotlivé operace. Nejprve se dle pravděpodobností operací rozhodneme, o kterou operaci půjde, a podle typu operace pak generujeme klíč. V případě operace SEARCH se jedná o náhodný klíč. Z hlediska testování je totiž jedno, jestli vyhledávací operace skončí úspěchem nebo neúspěchem. V případě modifikujících operací již potřebujeme možný neúspěch co nejvíce eliminovat. V případě operace INSERT generujeme náhodný klíč tak dlouho, dokud nezískáme klíč, který se ve stromu nenachází. V případě operace DELETE klíč generujeme náhodně tak dlouho, dokud nezískáme klíč, který se ve stromu nachází.

Data pro testy jsou generována před samotným testem. Je tomu tak proto, aby byly dosaženy pravděpodobnosti operací, které jsou definovány v konfiguračním souboru a aby samotné testování nebylo ovlivněno přípravou dat. Před spuštěním

testu je vytvořen počáteční strom a seznam prováděných operací je načten do paměti.

Jako náhodný generátor byl původně zvolen generátor *rand()* jazyka C, který vracel pseudonáhodné číslo v rozsahu $0-2^{31}$. Tento generátor se ukázal jako nevhodný, protože jeho rozsah není garantován na každém počítači a program by nemusel jít spustit.

Proto byl jako generátor zvolen *lrand48()* jazyka C, který také vrací čísla v rozsahu $0-2^{31}$, a tato čísla jsou rovnoměrně rozložena po celém intervalu.

Každý test byl proveden na dvaceti různých sadách vstupních dat lišících se od sebe pouze jiným počátečním nastavením náhodného generátoru.

6.4 Hardwarová konfigurace

Testy byly měřeny na následující hardwarové konfiguraci:

CPU	AMD Athlon XP 2500+ 1822MHz
RAM	640MB DDR 400 MHz
OS	Gentoo Linux
Kernel	2.6.15-r1
Kompilátor	gcc 3.4.6

7 Experimenty

Pro testování jsme zvolili čtyři základní scénáře, které se liší v pravděpodobnostech jednotlivých operací a které budou dále podrobněji popsány. Pro každý scénář jsme na základě definovaných pravděpodobností generovali 20 sad vstupních dat a nad každou vstupní sadou dat jsme prováděli testování při různém počtu paralelně pracujících procesů: 1, 2, 4, 8, 16, 32, 64. V případě relaxovaného stromu se jeden z těchto procesů stará o vyvažování stromu. S výjimkou jednoho procesu, kdy je vyvažování úplně vypnuto a veškeré vyvažování operace jsou provedeny až po skončení testu. Všechny testy mají společné to, že byly prováděny na počátečním stromu o velikosti 1000000 klíčů, do kterého bylo provedeno 3000000 operací. Strom je, jak uvidíme, mírně rostoucí, což je zajištěno vyšší pravděpodobností operace insert oproti operaci delete. U testu operací jsme měřili počet porovnání a čas každé operace zvlášť. U ostatních testů jsou veškerá měření prováděna po 1000 operacích.

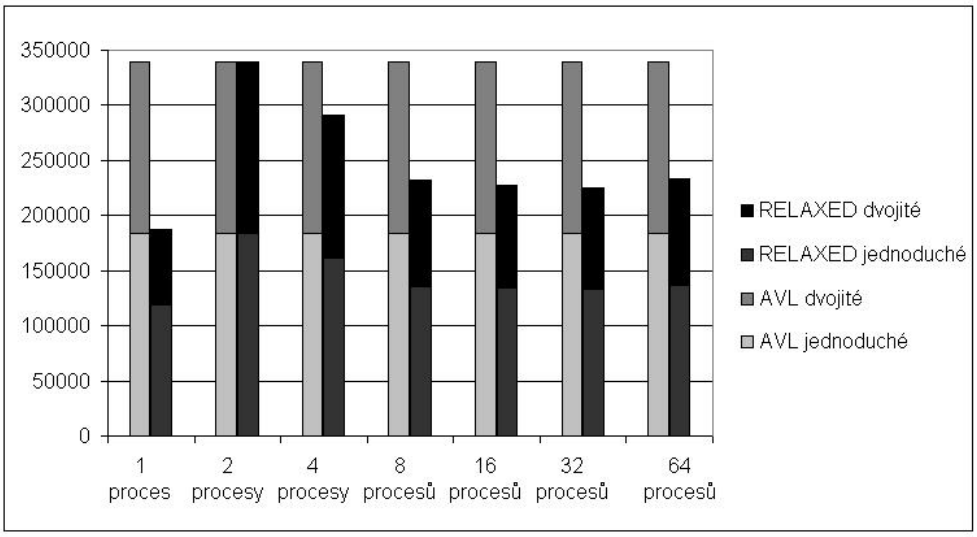
Pro každý scénář, stupeň paralelity a číslo testu jsou k dispozici na přiloženém CD výstupní naměřené hodnoty, grafy profilů naměřených hodnot a zpracované tabulky se statistickými údaji za jednotlivá měření. Tyto údaje jsou pro účely této práce příliš detailní. Zde budeme pracovat s celkovými časy za jednotlivé testy a s celkovými počty provedených rotací.

7.1 Test 1: časté vyhledávání

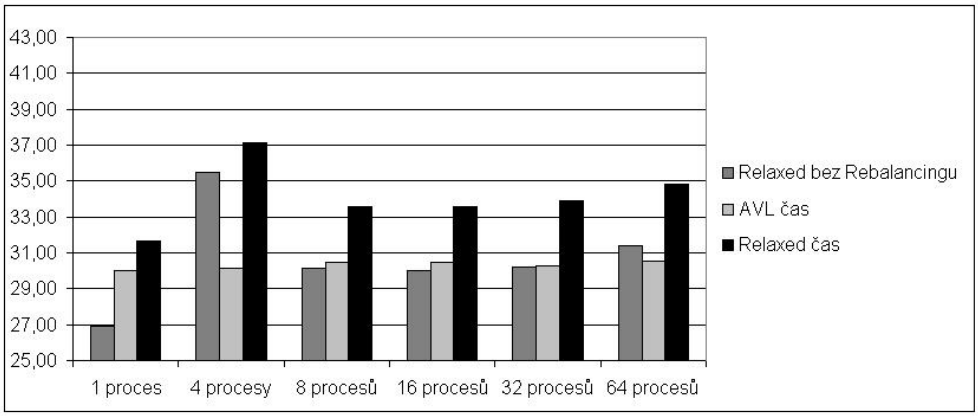
Tento test byl zaměřen na situaci, kdy datová struktura slouží spíše k vyhledávání než k modifikaci. Pravděpodobnosti jednotlivých operací byly zvoleny 70:20:10 pro operace search, insert a delete. Před spuštěním testu jsme očekávali lepší výsledky u AVL stromu, který garantuje výšku stromu a vyhledávací operace proto budou rychlejší.

V následující tabulce jsou uvedeny průměrné výsledky testů:

Průměrné údaje	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
AVL čas	29,98	30,04	30,17	30,45	30,45	30,27	30,52
Relaxed čas	26,91	125,02	35,45	30,11	29,99	30,23	31,38
Rebalancing	4,73	0,00	1,68	3,47	3,57	3,65	3,44



Graf 1: Počty a zastoupení rotací na testu časté vyhledávání



Graf 2: Celkové časy na testu časté vyhledávání

Je vidět, že AVL strom provádí stále stejné množství rotací. Oproti tomu, kromě 2 a 4 procesů, u relaxovaného stromu dochází k zapomínání velkého počtu rotací. Počet rotací provedených na relaxované struktuře klesá při vyšší míře paralelity kvůli tomu, že rebalancer má méně prostoru. Nejméně rotací je provedeno na relaxovaném stromu, do kterého přistupuje jeden proces. To není překvapivý výsledek, protože vyvažování neprobíhá během testu a nakonec je celý strom vyvážen jen potřebným počtem rotací.

V případě relaxovaného stromu se dvěma procesy je počet rotací stejný jako u klasického AVL stromu a dosažený čas je přímo alarmující. Důvod je ten, že jeden proces je rebalancer, který vyvažuje strom, a druhý proces provádí operace. Vždy když dojde k vložení uzlu do fronty rebalanceru updaterem, updater s ním zrovna pracuje nebo se pohybuje ve stromu někde velmi blízko. Rebalancer pak tento uzel vybere a snaží se ho vyvážit, zamyká jeho syny. Vzájemně si tak strukturu zamykají a překážejí si. V této konfiguraci by bylo jistě lepší vyvažování omezit, nebo úplně vypnout. S touto situací se setkáme ve všech scénářích, a proto ze všech grafů s časy v této práci byly vynechány případy, kdy do stromu přistupují 2 procesy, protože docházelo ke zhoršení přehlednosti grafů. V případě relaxovaného stromu se čtyřmi procesy je vidět, že rebalancer má pořád značnou váhu a provede dost rotací.

Co se týče celkového času, AVL strom dosahuje v tomto testu stabilních výsledků, které jsou ve většině případů srovnatelné, nebo i lepší než výsledky relaxovaného stromu bez závěrečného rebalancingu. Test dopadl dle očekávání spíše lépe pro AVL strom.

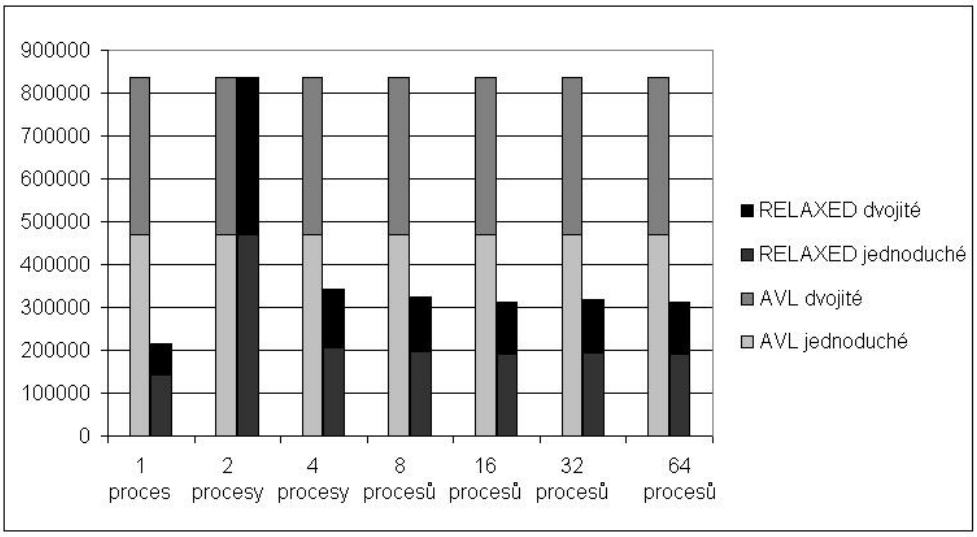
7.2 Test 2: časté modifikace

Tento test byl zaměřen na situaci, kdy dochází k častým změnám vyhledávací struktury, v případě AVL stromu dochází k mnoha rotacím, které by u relaxovaného stromu mohly být zapomenuty. Dále i zrychlení modifikujících operací by mělo být spíše na straně relaxovaného stromu. Pravděpodobnosti jednotlivých operací byly zvoleny 20:45:35 pro search, insert, delete a očekávali jsme zde lepší výsledky relaxovaného stromu než tomu bylo u prvního testu.

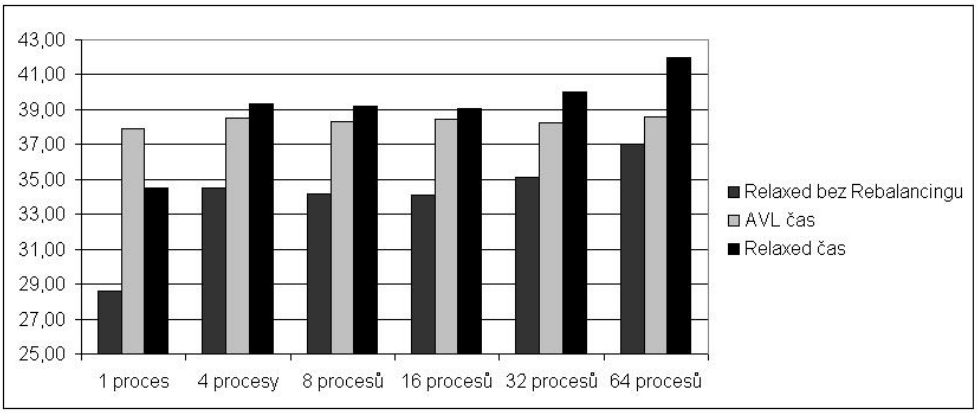
AVL strom opět provádí přibližně stejný počet rotací. Profil provedených rotací u relaxovaného stromu je velmi podobný jako u prvního testu. Velký rozdíl je zde v tom, jaká část vyvažovacích operací je zapomenuta. Relaxovaná struktura v tomto scénáři vykonává méně než polovinu rotací oproti AVL stromu.

Celkový čas AVL stromu se pohybuje na hranici času relaxované struktury včetně závěrečného rebalancingu a v porovnání s předchozím testem se zde více projeví výhody relaxovaného stromu.

Test dopadl dle očekávání. Relaxovaná struktura zde díky zrychlení modifikujících operací dosahuje velmi dobrých výsledků.



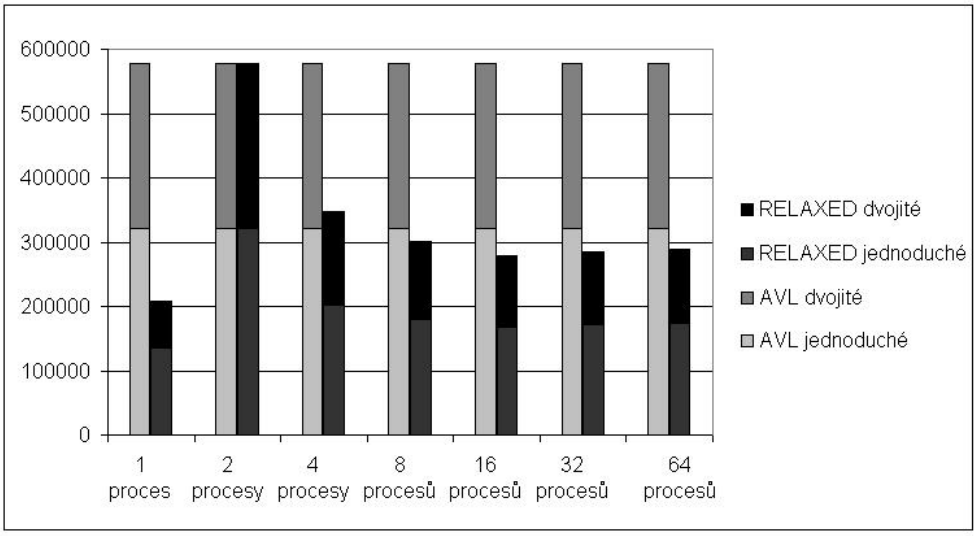
Graf 3: Počty a zastoupení rotací na testu časté modifikace



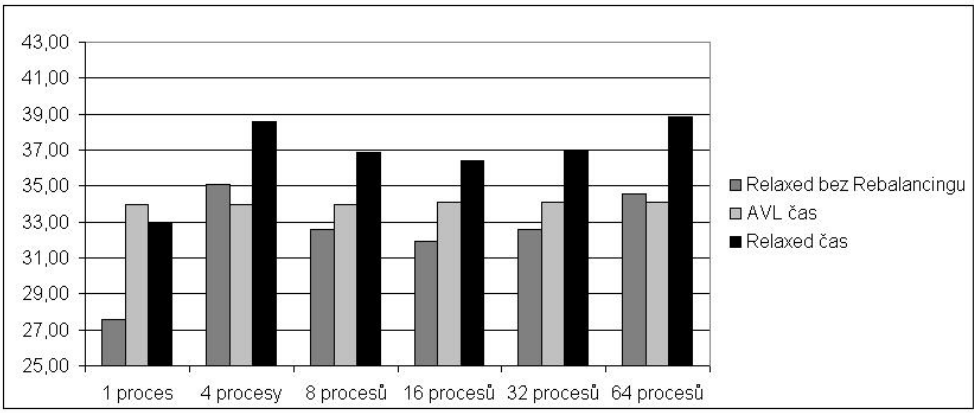
Graf 4: Celkové časy na testu časté modifikace

Překvapivý je zde ale výsledek relaxované struktury v sériovém prostředí s jedním procesem. Provedli jsme proto detailní analýzu jednotlivých měření v testu operací při jednom procesu a zjistili jsme, díky měření po jedné operaci, že maximální (resp. minimální) počet porovnání u relaxovaného stromu byl 30 (resp 15), zatímco u AVL stromu se počet porovnání pohyboval mezi 17-27. Při testu operací nebyl strom lehce rostoucí a pravděpodobnosti operací byly také trochu jiné, ale můžeme říci, že relaxovaná struktura byla pořád blízko AVL stromu a díky mnoha

zapomenutým rotacím tak efektivní. Toto chování si vysvětlujeme rovnoměrným rozložením vstupních dat, díky němuž je i vyhledávací struktura modifikována rovnoměrně.



Graf 5: Počty a zastoupení rotací na testů průměrné modifikace



Graf 6: Celkové časy na testu průměrné modifikace

7.3 Test 3: průměrné modifikace

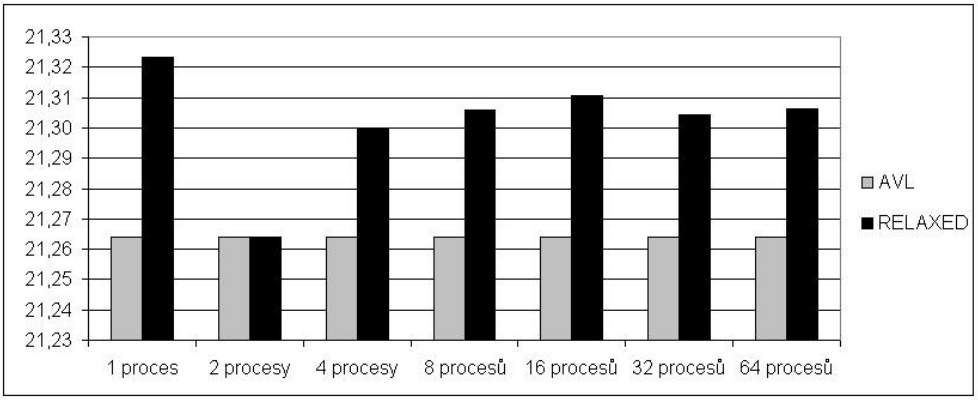
Tento test sloužil jako mezikrok mezi předchozími dvěma testy. Pravděpodobnosti jednotlivých operací byly zvoleny jako 46:32:22 pro search, insert a delete.

Celkové počty rotací a dosažené časy odpovídají tomu, že se jedná o mezikrok mezi předchozími dvěma testy. Počet i profil provedených rotací je odpovídající a podobný předchozímu testu. Vzájemné porovnání celkových časů zde vychází také podobně jako u předchozího testu. AVL strom je na tom v celkovém čase lépe než relaxovaná struktura. Ta je zase bez závěrečného vyvážení rychlejší než AVL strom.

7.4 Test operací

Tento test byl zaměřen na měření rychlosti a počtu porovnání jednotlivých operací. Měření počtu porovnání a celkového času jednotlivých operací bylo prováděno po jedné operaci. Pravděpodobnosti operací byly zvoleny 30:35:35 pro search, insert a delete. Očekávali jsme zde rychlejší modifikující operace u relaxovaného stromu a menší počet porovnání na operaci u AVL stromu.

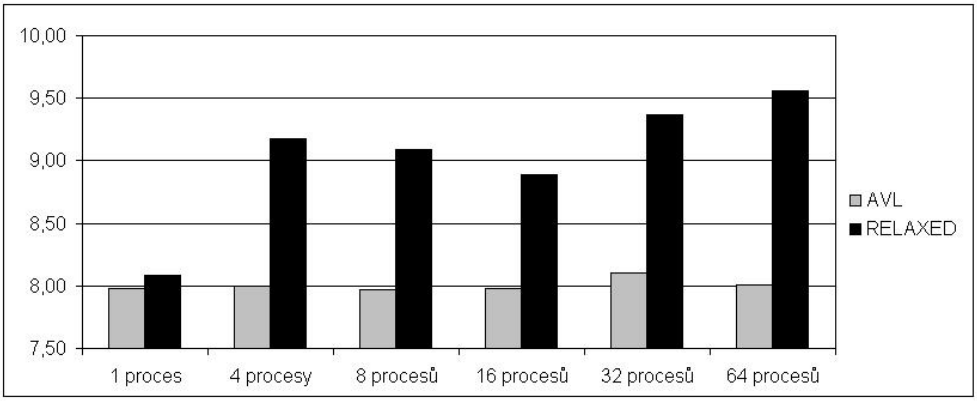
7.4.1 search



Graf 7: Průměrný počet porovnání při operaci SEARCH.

Relaxovaný strom potřebuje k nalezení klíče více porovnání než klasický AVL strom. Nejvíce při jednom procesu, kdy strom není vyvažován. U 2 procesů vidíme,

že relaxovaná struktura se velmi podobá AVL stromu, protože se často vyvažuje. Průměrný počet porovnání prvků u obou variant se příliš neliší.

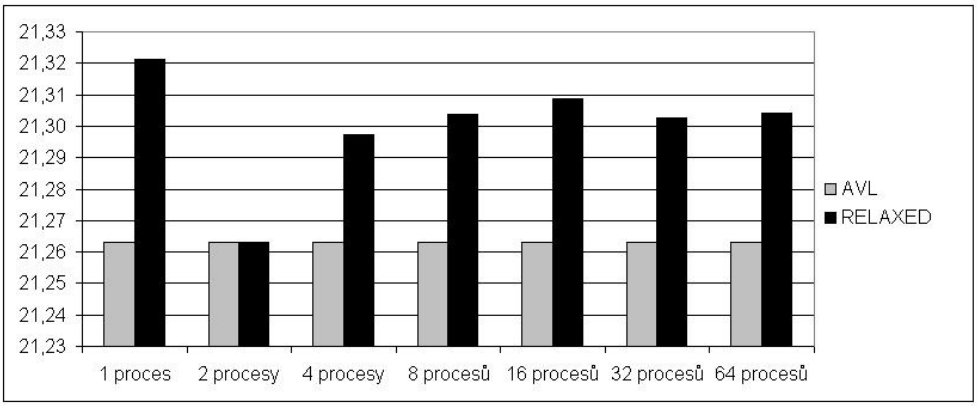


Graf 8: Průměrný čas na operaci SEARCH.

Průměrný čas operace search je lepší u AVL stromu. To plně odpovídá naměřenému počtu porovnání, kterých AVL strom udělá méně, a proto je operace search na AVL stromu rychlejší.

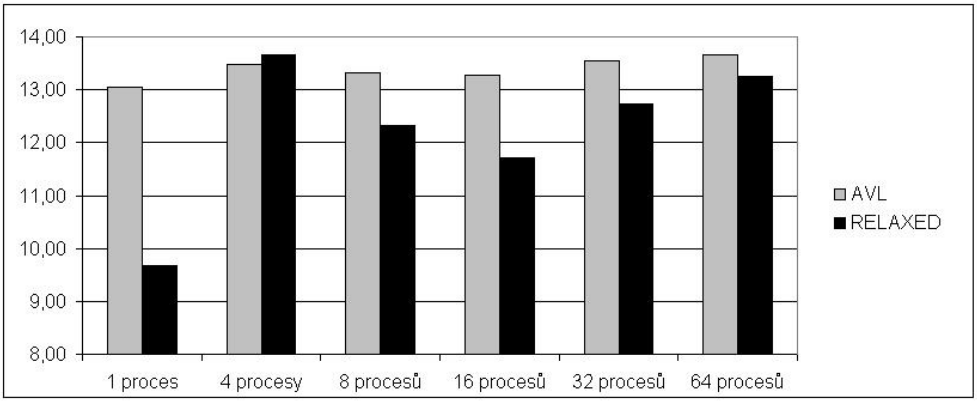
7.4.2 insert a delete

Grafy pro operaci delete jsou velmi podobné grafům pro operaci insert, proto nejsou v textu uvedeny. Jsou ale k dispozici na přiloženém CD.



Graf 9: Průměrný počet porovnání při operaci INSERT.

Průměrný počet porovnání pro modifikující operace je téměř stejný jako u operace search a tedy nižší u AVL stromu. I důvod je stejný.



Graf 10: Průměrný čas na operaci INSERT.

Zajímavé zde jsou průměrné časy modifikujících operací. Nejvýraznější rozdíl mezi rychlostí operací je vidět při jednom procesu, kdy u relaxované struktury neprobíhá vyvažování a rychlost modifikujících operací je tedy velmi blízká rychlosti operace search. U modifikujících operací je navíc oproti operaci search pouze režie na vytvoření (resp. smazání) uzlů.

Rozdíly mezi rychlostmi modifikujících operací jsou při více procesech menší, neboť během nich dochází na pozadí k vyvažování stromu. Test dopadl dle očekávání.

Následující tabulka je společná pro všechny operace. Najdeme v ní maxima a minima počtu porovnání klíčů. Tabulka nám vypovídá o nejkratší (resp. nejdelší) cestě ve zkoumaných stromech.

Počet porovnání	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
AVL MIN	17	17	17	17	17	17	17
AVL MAX	25	25	25	25	25	25	25
RELAXED MIN	15	17	16	16	16	16	16
RELAXED MAX	30	25	27	28	28	28	28

Zajímavé jsou zde výsledky relaxované struktury. Při dvou procesech je nejkratší i nejdelší naměřená cesta stejná jako u AVL stromu. Při čtyřech procesech jsou naměřené hodnoty stále blízko hodnot AVL stromu kvůli častému vyvažování. Při

větším počtu paralelních procesů lze pozorovat snižující se význam rebalanceru a lehké zhoršování struktury relaxovaného stromu. Nejvýraznější rozdíl je při jednom procesu, kdy vyvažování neprobíhá vůbec.

Vzhledem k tomu, že vyvažování probíhá zezdola-nahoru, a že rebalancer prioritně bere nejhlubší uzly dochází k udržení rozumné výšky stromu. Při naší implementaci nehrozí vznik dlouhé a degenerované cesty.

8 Závěr

Cílem této práce bylo porovnat klasický AVL strom s relaxovanou variantou, kde je vyvažování stromu odděleno od updatů. Vyvažování je pak prováděno buď na pozadí probíhajících operací, nebo je úplně vypnuto. Testy začínaly s vyváženým AVL stromem a skončily také s vyváženým AVL stromem.

Chtěli jsme demonstrovat výhody relaxovaného přístupu ke struktuře v praxi. Především možnost vypnout vyvažování struktury, pokud je systém přetížen požadavky a zrychlit tak modifikující operace a odezvu klientům. Chtěli jsme ukázat, že i v případě vypnutí vyvažování na dlouhou dobu nedojde ke ztrátě kontroly nad strukturou a operace budou stále efektivní.

Další možné využití vypnutí vyvažování je v situaci, kdy je strom dávkou operací insert a delete updatován, zatímco do něj přistupují klienti. Ti by pak nebyli zpomalováni intenzivním vyvažováním stromu. Dále jsme chtěli prakticky ukázat, že závěrečné vyvážení relaxované struktury zpět do AVL stromu je rychlé a efektivní.

Ukázali jsme, že počet porovnání, které musí vykonat nějaká operace, je menší u AVL stromu, který garantuje maximální výšku. Proto u AVL stromu byla rychlejší operace search a AVL strom si vedl lépe v testu s větší pravděpodobností této operace.

U modifikujících operací jsme demonstrovali větší rychlost relaxované struktury, i přes větší výšku stromu, která je způsobena odložením vyvažování na pozdější dobu.

Téměř ve všech testech se ukázalo, že relaxovaná struktura bez závěrečného vyvážení dosahuje lepších časů než AVL strom, a to i v případě častého vyhledávání. Rozdíly mezi oběma typy stromů se zvětšovaly s větším počtem modifikujících operací.

V praktických testech na relaxovaném stromu dopadly nejhůře situace, kdy do stromu současně přistupovaly dva (resp. čtyři) procesy. A to ve všech zvolených testech. To je způsobeno vzájemným zamykáním stromu v prvním případě. V druhém případě je to spíše způsobeno příliš velkou vahou rebalanceru. Dochází zde k mnoha rotacím, čímž je částečně eliminována výhoda námi implementovaného stromu tyto rotace zapomínat. Navíc zde také ještě bude docházet k vzájemné interferenci rebalanceru s updatery. V těchto případech doporučujeme vyvažování na pozadí úplně vypnout, protože je spíše na škodu.

Od osmi paralelně přistupujících procesů dosahuje relaxovaný strom lepší výsledky. Rebalancer si s ostatními procesy nepřekáží a vyvažováním nejhlubších uzlů udržuje rozumnou výšku stromu. Tím zajišťuje dostatečně efektivní operace. Nejlépe v našich testech dopadlo 16 procesů (z nichž je jeden rebalancer).

V sériovém prostředí, kdy do stromu přistupuje pouze jeden proces, jsou ve všech případech výsledky lepší než u paralelních variant. Je tomu tak proto, že zde odpadá režie na zamykání stromu. Výsledky v tomto případě nás i tak trochu překvapily. Rebalancer zde nevyvažuje spodní patra stromu a tak není žádným způsobem "garantována" výška stromu. Přesto, jak můžeme vidět v testu operací, nedochází k výraznější degeneraci vyhledávací struktury. Tuto skutečnost přičítáme rovnoměrnému rozložení vstupních dat.

Velkou výhodu relaxovaného vyvažování, která spočívá v zamykání pouze malého množství zámků, jsme v našich testech nemohli pořádně demonstrovat. Testy probíhaly na jednom procesoru, který prováděl činnost všech paralelně přistupujících klientů. Proto vždy, když došlo k uspání nějakého procesu na zámku, procesor začal vykonávat jiný proces, který čekat nemusel, a tak byl pořád vytížen na 100%. V případě, kdybychom výkon procesoru rovnoměrně rozdělili mezi testovaný počet procesů, byly by výsledky AVL stromu s rostoucí mírou paralelity výrazně horší. Docházelo by více k blokování přístupu do stromu při modifikujících operacích a výkon procesoru by nebyl plně využit.

8.1 Shrnutí práce

Ukázali jsme, že téměř ve všech testech relaxovaná struktura bez závěrečného vyvažování dosahovala lepší výsledky než AVL strom. Při vyvažování na pozadí rebalancer udržoval rozumnou hloubku námi implementovaného stromu a operace v něm byly stále efektivní. Celkový čas včetně závěrečného vyvažování byl u relaxované struktury horší, než u AVL stromu. Operace search byla rychlejší u AVL stromu, modifikující operace zase u relaxovaného stromu.

Použití relaxovaného vyvažování v situacích, kdy je systém přetížen, nebo kdy přijde dávka modifikujících operací, lze doporučit. Celkový čas je sice horší, ale díky relaxovaná struktuře dojde k rozložení zátěže do delšího časového období.

8.2 Budoucí práce

Ukázali jsme, že i přes vypnutí vyvažování na poměrně dlouhou dobu, nedocházelo k příliš velkým degeneracím datové struktury a ta byla stále velmi efektivní. To je způsobenou volbou náhodného generátoru, který generuje čísla rovnoměrně z celého intervalu a tím dochází i k rovnoměrné modifikaci datové struktury. Pro další výzkum relaxované struktury bychom mohli rozšířit program o testování na datech, která mají jiné rozložení a výsledky mezi sebou porovnat.

Operace prováděné do testovaných stromů byly náhodně generovány. Pro další výzkum relaxované struktury bychom mohli implementovat jiné testovací scénáře, kdy by updaty chodily po dávkách, nebo by jednotlivé procesy byly striktně rozděleny na readery a updatery, ...

V situaci, kdy je zřejmé, že updaty budou chodit po dávkách (posílaných třeba administrátorem), by mohly být efektivnější datové struktury umožňující skupinové updaty (bulk updates). Ty jsou popsány například v [11, 18].

9 Přílohy

A Výsledky a tabulky

A.1 Test časté vyhledávání:

RELAXED	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
Průměr	26,90	125,65	35,44	30,09	29,97	30,21	31,36
Směrodat. odch.	0,75	5,00	2,82	1,61	1,16	1,34	0,93
Medián	26,53	125,50	35,94	30,62	30,00	30,49	31,29
Min	26,42	120,52	27,62	27,23	27,60	27,99	29,47
Max	29,31	140,39	39,47	32,76	31,56	32,95	33,33
0,1-useknutý průměr	26,79	125,11	35,65	30,11	30,01	30,18	31,35

AVL	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
Průměr	29,98	30,04	30,17	30,45	30,45	30,27	30,52
Směrod. odch.	0,43	0,45	0,58	0,71	0,68	0,52	0,46
Medián	29,77	29,91	29,90	30,00	30,02	30,07	30,29
Min	29,67	29,68	29,70	29,80	29,88	29,89	29,98
Max	31,05	31,49	31,41	31,76	31,68	31,70	31,29
0,1-useknutý průměr	29,94	29,97	30,13	30,42	30,41	30,21	30,51

Průměrné údaje	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
AVL čas	29,98	30,04	30,17	30,45	30,45	30,27	30,52
Relaxed čas	31,64	125,02	37,13	33,58	33,56	33,88	34,83
Relaxed rebalancing	4,73	0,00	1,68	3,47	3,57	3,65	3,44
bez rebalancingu	26,91	125,02	35,45	30,11	29,99	30,23	31,38

A.2 Test průměrné modifikace:

RELAXED	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
Průměr	28,60	293,76	34,48	34,13	34,06	35,14	37,02
Směrodat. odch.	0,50	11,14	3,82	2,13	2,72	2,37	1,50
Medián	28,43	289,83	36,11	34,32	35,03	36,00	37,42
Min	28,27	282,85	29,51	30,63	30,38	30,88	34,21
Max	30,24	329,21	40,60	38,08	37,95	38,53	39,76
0,1-useknutý průměr	28,53	292,40	34,42	34,10	34,05	35,19	37,02

AVL	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
Průměr	37,90	38,07	38,53	38,31	38,43	38,24	38,59
Směrodat. odch.	0,47	0,56	0,64	0,48	0,49	0,41	0,52
Medián	37,68	37,81	38,32	38,12	38,42	38,08	38,30
Min	37,59	37,60	37,81	37,83	37,75	37,87	38,10
Max	39,09	39,51	39,71	39,41	39,47	39,67	39,89
0,1-useknutý průměr	37,85	38,02	38,50	38,28	38,41	38,18	38,55

Průměrné údaje	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
AVL čas	37,90	38,07	38,53	38,31	38,43	38,24	38,59
Relaxed čas	34,50	293,84	39,34	39,18	39,04	39,99	41,95
Relaxed rebalancing	5,89	0,00	4,84	5,03	4,96	4,83	4,91
bez rebalancingu	28,61	293,84	34,50	34,15	34,08	35,15	37,04

A.3 Test časté modifikace:

RELAXED	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
Průměr	28,60	293,76	34,48	34,13	34,06	35,14	37,02
Směrodat. odch.	0,50	11,14	3,82	2,13	2,72	2,37	1,50
Medián	28,43	289,83	36,11	34,32	35,03	36,00	37,42
Min	28,27	282,85	29,51	30,63	30,38	30,88	34,21
Max	30,24	329,21	40,60	38,08	37,95	38,53	39,76
0,1-useknutý průměr	28,53	292,40	34,42	34,10	34,05	35,19	37,02

AVL	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
Průměr	37,90	38,07	38,53	38,31	38,43	38,24	38,59
Směrodat. odch.	0,47	0,56	0,64	0,48	0,49	0,41	0,52
Medián	37,68	37,81	38,32	38,12	38,42	38,08	38,30
Min	37,59	37,60	37,81	37,83	37,75	37,87	38,10
Max	39,09	39,51	39,71	39,41	39,47	39,67	39,89
0,1-useknutý průměr	37,85	38,02	38,50	38,28	38,41	38,18	38,55

Průměrné údaje	1 pr.	2 pr.	4 pr.	8 pr.	16 pr.	32 pr.	64 pr.
AVL čas	37,90	38,07	38,53	38,31	38,43	38,24	38,59
Relaxed čas	34,50	293,84	39,34	39,18	39,04	39,99	41,95
Relaxed rebalancing	5,89	0,00	4,84	5,03	4,96	4,83	4,91
bez rebalancingu	28,61	293,84	34,50	34,15	34,08	35,15	37,04

B Struktura přiloženého CD

Na přiloženém CD se nachází následující adresáře:

bin	obsahuje a generuje se sem přeložený program
doc	programátorská dokumentace a text práce ve formátu PDF
results	adresářová struktura, do které se zapisují výstupy z měření
src	zdrojové kódy programu
tables	grafy a tabulky výsledků zpracované v aplikaci Microsoft Excel
tests	konfigurační soubory s definicí jednotlivých testů, adresářová struktura do které se generují data pro jednotlivé testy.
text	zdrojové soubory textu této práce

Adresáře obsahují soubor .Readme.txt, který popisuje jejich obsah a případnou strukturu adresářů.

C Kompilace programu a jeho spuštění

V této příloze popíšeme způsob kompilace programu a uvedeme význam a možné hodnoty všech parametrů jeho spouštění. Zdrojové kódy jsou k dispozici na přiloženém CD v adresáři **src**.

Zdrojové soubory jsou překládány kompilátorem g++. Pro snažší práci s tímto kompilátorem pomocí utility **make** jsme vytvořili soubor **Makefile**, který je také k dispozici na CD v adresáři společně se zdrojovými soubory. Podrobný popis použití je k dispozici na CD.

Samotný program je po zkompilování možné spouštět z příkazové řádky a pomocí parametrů lze ovlivňovat jeho chování. Jedno spuštění programu provede jeden test. Zde jsou uvedeny nejdůležitější parametry programu.

$-h$		Nápověda
$-a$	algoritmus	Typ stromu na který bude testován. Možné hodnoty jsou: RELAXED - relaxovaný AVL strom AVL - standardní AVL strom
$-t$	typ_testu	Název konfiguračního souboru testu. Vyžaduje existující soubor s konfigurací *tests/test_type.ini*
$-n$	číslo	Číslo opakování testu.
$-p$	počet_procesů	Počet vláken paralelně přistupujících do stromu.
$-g$		Pouze vygeneruj test pro danou konfiguraci a konec.
$-r$		Negeneruj test, použij vygenerovaný.
$-d$		Po ukončení testování smaž vygenerovaný test.

Kompilace vytvoří v adresáři *bin* spustitelný soubor se jménem *avlserver*.

Příklady spouštění programu:

Pouze vygenerování testu číslo 7 dle konfiguračního souboru average.ini

avlserver -t average -n 7 -g

Spustit testování AVL stromu na testu číslo 10 scénáře dle konfiguračního souboru heavy_search.ini. Data testu negenerovat, testovat pro 16 procesů. Po skončení testu vygenerovaný test smazat.

avlserver -a AVL -t heavy_search -p 16 -r -d -n 10

Případné logování pomocných výpisů lze zapnout v hlavičkovém souboru *globals.h*. Poté je potřeba program znovu přeložit.

Reference

[1] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the Organization of Information. *Doklady Akademi Nauk SSSR*, 146:263-266, 1962. In Russian. English translation in *Soviet Math. Doklady*, 3:1259-1263, 1962.

[2] A.V.Aho, J.E.Hopcroft and J.D.Ullman. Data Structures and Algorithms. *Addison-Wesley, Reading*, MA, 1983.

[3] J. Boyar, K. S. Larsen. Efficient rebalancing of chromatic search trees. *Journal of Computer and System Sciences*, 49(3):667-682, 1994.

[4] C. S. Ellis. Concurrent search in AVL-trees. *IEEE Transaction on Computers*, C-29:811-817, 1980.

[5] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. *Proceedings of the 19 Annual Symposium on Foundations of Computer Science*, 8-21, 1978.

[6] J. L. W. Kessels. On-the-fly optimization of data structures. *Communications of ACM*, 26:895-901, 1983.

[7] H. T. Kung and P. L. Lehman. A concurrent database manipulation problem: Binary search trees. *ACM Transactions on Database Systems*, 3:339-353, 1980.

[8] K. S. Larsen. Amortized constant relaxed rebalancing using standard rotations. *Acta Informatica*, 35(10):859-874, 1998.

[9] K. S. Larsen. AVL Trees With Relaxed Balance. *Proceeding of the 8th International Parallel Processing Symposium*,pp. 196-202. IEEE Computer Society Press 1995.

[10] K. S. Larsen, E. Soisalon-Soininn and P. Widmayer. Relaxed Balance Using Standard Rotations. *Algoritmica*, 31(4):501-512, 2001.

[11] L. Malmi, E. Soisalon-Soininen. Group updates for relaxed height-balanced trees. *Proceedings of the 18th ACM Symposium on Principles of Database Systems*, pp. 358-367, 1999.

[12] O. Nurmi, E. Soisalon-Soininen and D. Wood. Relaxed AVL Trees, Main-Memory Databases, and Concurrency. *Technical Report HKUST*, CS96-22, 1996.

[13] O. Nurmi and E. Soisalon-Soininen. Uncoupling updating and rebalancing in chromatic binary search trees. *Proceedings of the 10th ACM Symposium on Principles of Database Systems*, pp. 192-198, 1991

[14] O. Nurmi, E. Soisalon-Soininen and D. Wood. Concurency Control in Database Structures with Relaxed Balance. *Proceeding of the 6th ACM Symposium on Principles of Database Systems*, pp. 170-176, ACM Press 1987.

[15] T. Ottman and E. Soisalon-Soininen. Relaxed balancing made simple. *Technical Report 71*, Institut für Informatik, Universität Freiburg, 1995.

[16] T. Ottman, M. Schrapp, and D. Wood. Purely top-down updating algorithms for stratisfied search trees. *Acta Informatica*, 22:85-100, 1985.

[17] E. Soisalon-Soininen and P. Widmayer. Relaxed balancing in search trees. *Advances in Algorithms, Languages, and Complexity* (D.-Z. Du and K.-I. Ko, eds.), Kluwer, Dordrecht, pp. 267-283, 1997.

[18] E. Soisalon-Soininen, P. Widmayer. Amortized Complexity of Bulk Updates in AVL-trees. *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, pp. 439-448, 2002.