VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

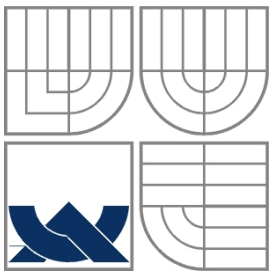
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PŘÍPRAVA DOMÁCÍCH ÚLOH PRO PŘEDMĚT ALGORITMY

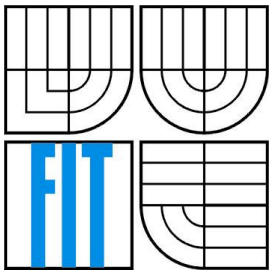
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE MARTIN FELIX
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PŘÍPRAVA DOMÁCÍCH ÚLOH PRO PŘEDMĚT ALGORITMY

PREPARATION OF HOMEWORKS IN THE COURSE ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — MARTIN FELIX
AUTHOR

VEDOUCÍ PRÁCE — ING. ROMAN LUKÁŠ, PH.D.
SUPERVISOR

BRNO 2008

Zadání bakalářské práce/6662/2007/xfelix00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Felix Martin**
Obor: Informační technologie
Téma: **Příprava domácích úloh pro předmět Algoritmy**
Kategorie: Alg. a datové struktury

Pokyny:

1. Seznamte se se systémem pro automatizované zadávání a hodnocení domácích úloh v předmětu Algoritmy.
2. Po dohodě s vedoucím práce vyberte pokročilé tři příklady z původních příkladů v jazyce Pascal nebo navrhněte příklady nové.
3. Vybrané příklady vzorově implementujte v jazyce C. Součástí příkladů budou i testy pro ověření správnosti implementace.
4. V případě potřeby rozšiřte stávající systém pro zadávání a hodnocení domácích úloh podle pokynů vedoucího.
5. Zhodnoťte dosažené výsledky a navrhněte možné směry dalšího vývoje.

Literatura:

- Honzík, J. M., Hruška, T., Máčel, M.: Vybrané kapitoly z programovacích technik, Vysoké učení technické v Brně, Brno, 1991. ISBN 80-214-0345-4.
- Herout, P.: Učebnice jazyka C, 3. upravené vydání, Kopp, České Budějovice, 1998. ISBN 80-85828-21-9.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Lukáš Roman, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Martin Felix**
Id studenta: 79040
Bytem: Lesní 1486, 696 42 Vracov
Narozen: 21. 11. 1984, Kyjov
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Příprava domácích úloh pro předmět Algoritmy
Vedoucí/školitel VŠKP: Lukáš Roman, Ing., Ph.D.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Tato bakalářská práce se zabývá AVL stromy, které spadají do kategorie binárně vyhledávacích stromů. Práce se skládá ze tří částí, jedna část má za cíl naučit studenta vkládat uzly do AVL stromu rekurzivním způsobem, druhá část rušit uzly z AVL stromu nerekurzivním způsobem a poslední část má doplnit studentovy znalosti o stromech. Studentovy výsledky se porovnávají s výsledky vzorového řešení a podle toho jsou mu následně přiděleny body.

Klíčová slova

AVL strom, samovyvažující se binární vyhledávací strom, vyhledání uzlu, vložení uzlu, zrušení uzlu, vyvážení, jednoduchá rotace, dvojitá rotace, LL rotace, RR rotace, DLR rotace, DRL rotace.

Abstract

This Bachelor's thesis is about AVL trees, which belongs to the binary search trees. Thesis contains three parts, objective of the first one is to learn student how insert nodes into AVL trees in recursive way, objective of the second part is to learn student how delete nodes from AVL trees in non-recursive way and objective of the last one is to complete student's knowledge about AVL trees . Student's results are mached with results of model solution and this evaluation establishes student's points.

Keywords

AVL tree, self-balancing binary search tree, lookup, insertion, deletion, balancing, simple rotation, double rotation, LL rotation, RR rotation, DLR rotation, DRL rotation.

Citace

Felix Martin: Příprava domácích úloh pro předmět Algoritmy. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Příprava domácích úloh pro předmět Algoritmy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Romana Lukáše, Ph.D. Další informace mi nikdo neposkytl.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Felix
12.5.2008

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu práce Ing. Romanu Lukášovi, Ph.D. za vedení při tvorbě této práce a za cenné připomínky a rady.

Obsah

Úvod

Tato bakalářská práce se zabývá AVL stromy, které spadají do kategorie binárně vyhledávacích stromů. Práce se skládá ze tří částí, jedna část má za cíl naučit studenta vkládat uzly do AVL stromu rekurzivním způsobem, druhá část rušit uzly z AVL stromu nerekurzivním způsobem a poslední část má doplnit studentovy znalosti o stromech. Studentovy výsledky se porovnávají s výsledky vzorového řešení a podle toho jsou mu následně přiděleny body.

První kapitola práce má název Úvod. Slouží k zasazení řešené problematiky do širšího kontextu a v podobě stručného obsahu jednotlivých kapitol definuje strukturu písemné práce.

Druhá kapitola práce má název AVL strom. Zabývá se teorií těchto stromů, jednotlivými operacemi nad nimi, jako například vyhledávání, vkládání a rušení uzlů ve stromu. Dále podrobnou analýzou problematiky jednotlivých operací a možnými metodami průchodu stromem.

Třetí kapitola práce má název Implementace práce. Rozebírám v ní návrh vlastního řešení jednotlivých funkcí, tedy sestavováním jejich algoritmu, volbou datových typů a návrhem testů, kontrolujících správnost řešení jednotlivých funkcí.

Čtvrtá kapitola práce má název Specifikace testů. Zabývá se testováním programu, tedy kontrolou správnosti implementace. K testování byly použity mezní hodnoty, které byly odvozeny při návrhu algoritmu.

V závěrečné kapitole je vyhodnoceno řešení programu a provedené testy.

1 AVL strom

Zkratka **AVL** pochází z iniciál jeho objevitelů, G. M. **A**delson-**V**elsky a E. M. **L**andis. kteří tuto strukturu poprvé popsali v roce 1962, v článku „*An algorithm for the organization of information*“. Tito matematici dokázali, že jeho výška je maximálně o 45% větší než u váhově vyváženého stromu.

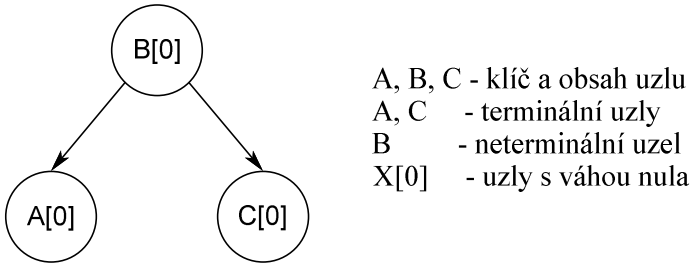
Složitost „O“ vyhledávání, vkládání a rušení v AVL stromu je logaritmická: $O(\log n)$, kde *n* je počet uzlů stromu. Při maximálním počtu porovnání v co nejvyšším stromě je složitost ve srovnání s váhově vyváženým BVS nanejvýš $1{,}45$*krát* větší. Tedy $1{,}45 * \log n$, kde *n* je počet uzlů stromu.

AVL strom je samovyvažující se binární vyhledávací strom. Je to tedy datová struktura vycházející z binárního stromu, v němž jsou jednotlivé uzly uspořádány tak, aby bylo možné co nejrychleji vyhledat požadovanou hodnotu. To je docíleno následujícími vlastnostmi.

AVL strom je buď prázdný, nebo sestává z jednoho uzlu zvaného kořen a dvou podstromů – levého a pravého, oba tyto podstromy mají vlastnosti stromu. AVL strom tvoří kořen, neterminální uzly, které mají ukazatel na jeden nebo dva uzly synovské (potomky) a terminálních uzlů (listů), které nemají žádné potomky. Každý uzel je kořenem svého podstromu a obsahuje klíč (hodnota podle které se vyhledává) a koeficient vyváženosti (o té později). Podstromu (*subtree*) se říká také větev (*branch*). Uzly vycházející na cestě doleva z kořene vytvářejí hlavní (levou) diagonálu, doprava vedlejší (pravou) diagonálu. Pro AVL stromy je typické, že výška jeho dvou podstromů se liší nejvýše o jedna, proto se mu také říká výškově vyvážený BVS. Výšková vyváženost může být narušena při vkládání či rušení prvku, z tohoto důvodu u AVL stromů mluvíme o tzv. znovuvyvážení stromu. Znovuvyvážení probíhá za pomoci několika druhů rotací.

Ukázka stromu je na obrázku *Obr. 1.1*.

Obr. 1.1
Příklad uzlu s dvěma postromy



Každý uzel stromu má určitý koeficient vyváženosti, zjednodušeně „váhu“, hodnota váhy reprezentuje rozdíl výšek pravého a levého podstromu tohoto uzlu. Váha může být buď uložena přímo jako hodnota v každém uzlu anebo může být dopočítávána z výšek jeho podstromů. Z principu AVL stromu může váha nabývat těchto hodnot:

Váha = -2 - narušena vyváženost uzlu, výška levé větve uzlu je o 2 větší

Váha = -1 - uzel je vyvážený, ale levá větev je „těžší“, výška levé větve uzlu je o 1 větší

Váha = 0 - uzel je dokonale vyvážený

Váha = 1 - uzel je vyvážený, ale pravá větev je „těžší“, výška pravé větve uzlu je o 1 větší

Váha = 2 - narušena vyváženost uzlu, výška pravé větve uzlu je o 2 větší

Pokud tedy bude váha uzlu +/-2 je potřeba provést rotaci ke znovuustavení výškové vyváženosti stromu. Oproti váhově vyváženým stromům je znovuustavení výškové vyváženosti poměrně jednoduché a to rekonfigurací několika uzlů v okolí tzv. kritického uzlu a současné úpravě koeficientů vyváženosti. Kritický uzel je nejvzdálenější uzel od kořene, v němž díky vložení či zrušení uzlu došlo k porušení rovnováhy, tedy ke změně váhy uzlu.

Nad AVL stromy lze provádět stejné operace jako nad binárně vyhledávacími stromy a to: vyhledávání, vkládání a rušení uzlu. Vyhledání uzlu je totožné jako u BVS. Zbylé operace vycházejí z BVS s tím rozdílem, že jsou doplněny o úpravu vah a případnou rotaci zajišťující znovuustavení výškové vyváženosti. Podrobněji o těchto operacích v následujících kapitolách.

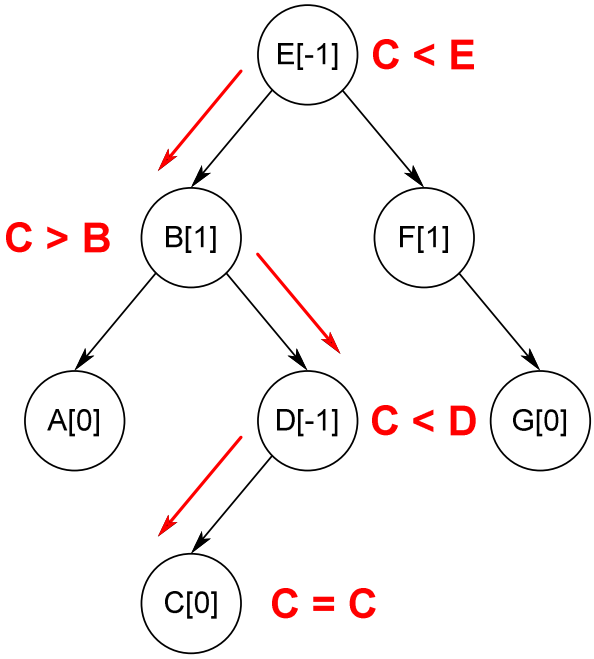
1.1 Vyhledání uzlu

Vyhledání uzlu v AVL stromu se provádí stejně jako u nevyvážených binárních vyhledávacích stromů. Vyhledání konkrétní hodnoty v AVL stromu probíhá typicky rekurzivně, přestože paměťová náročnost nerekurzivního způsobu je menší. Díky uspořádání stromu je cesta k hledané hodnotě jednoznačně určena.

Vyhledávání funguje následujícím způsobem. Začíná se zpravidla na kořeni. V každém kroku se porovnává hledaný klíč s klíčem zkoumaného uzlu. Jsou-li si rovny, vyhledávání úspěšně končí. Je-li hledaný klíč menší, než klíč kořene, pokračuje se v levém podstromu kořene. Naopak pokud je klíč větší, pokračuje se v pravém podstromu. Tímto způsobem se postupně prochází další uzly, dokud není uzel s požadovaným klíčem nalezen. Pokud se během průchodu stromem narazí na neexistující uzel (prohledávaný uzel je prázdný), vyhledávání skončí neúspěšně. Ukázka vyhledávání ve stromu je zobrazena na obrázku *Obr. 1.2*.

Během vyhledávání nedochází k žádným zásahům do stromu, na rozdíl od některých jiných metod. Díky výškové vyváženosti stromu je časová náročnost vyhledání uzlu je $O(\log n)$.

Obr. 1.2
Diagram vyhledání uzlu s klíčem 'C'



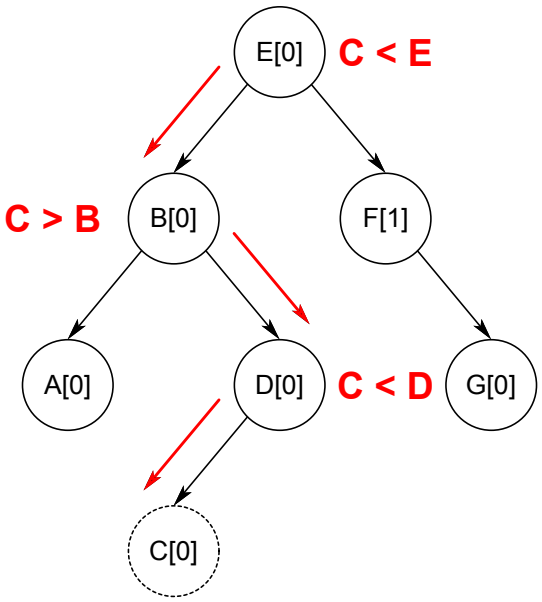
1.2 Vkládání uzlu

Vložení uzlu do AVL stromu opět vychází z binárně vyhledávacích stromů. Časová náročnost vložení uzlu je $O(\log n)$. Jako hledaná hodnota se použije klíč vkládaného uzlu. Lze použít jak rekurzivní, tak nerekurzivní zápis. Tato fáze může vést k několika různým výsledkům.

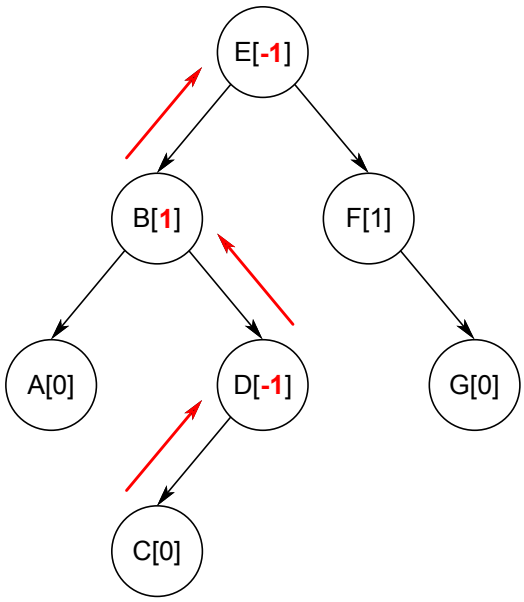
Pokud je (pod)strom prázdný, vložíme nový uzel a přiřadíme mu váhu 0. Pro neprázdný strom platí, že pokud je klíč vkládaného uzlu menší nežli klíč aktuálního uzlu, pokračujeme v levém podstromu tohoto uzlu a naopak. Pokud narazíme na uzel se stejným klíčem, jako má vkládaný klíč, přepíšeme obsah tohoto uzlu vkládanou hodnotou. Váha uzlu se přitom nijak nemění. Některé varianty stromů připouštějí vícenásobný výskyt uzlu se stejným klíčem. Příklad vložení uzlu do stromu je zobrazen na diagramu *Obr. 1.3*.

Po vložení uzlu následuje zkontrolování a případné upravení váhy pro každý uzel na cestě směrem ke kořeni stromu, jelikož vložením nového prvku došlo ke změně výšky některého z podstromů. Celá procedura viz. *Obr 1.4*. Není-li splněna podmínka vyváženosti stromu tzn. některý z uzlů má váhu +/-2 je potřeba provést cyklickou záměnu uzlů neboli rotaci. Rotace můžeme rozdělit na jednoduché a dvojité, z nichž každá má svoji stranovou variantu, více o těchto rotacích v podkapitolách 1.2.1 - 1.2.4.

Obr. 1.3
Diagram vložení uzlu s klíčem 'C'



Obr. 1.4
Diagram úpravy vah uzlů
po vložení uzlu s klíčem 'C'

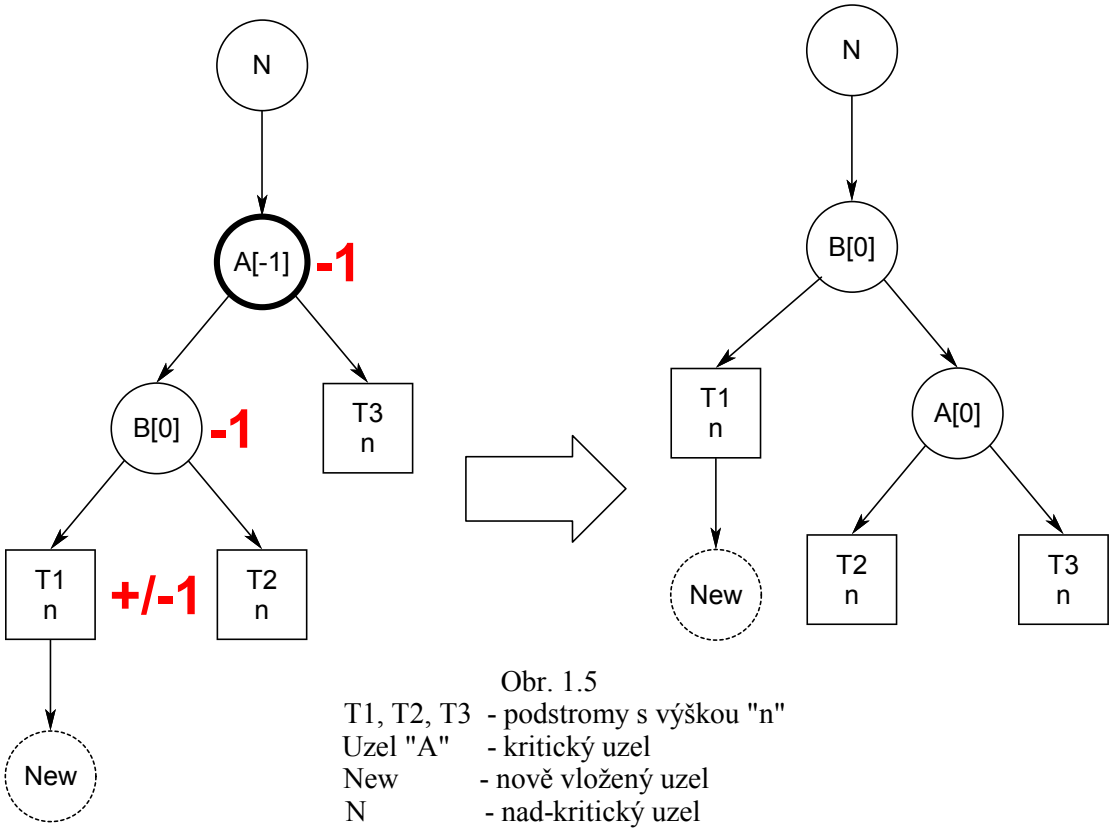


1.2.1 Jednoduchá rotace LL pro vkládání

Jednoduchá rotace Left-Left (dále jen LL) koriguje porušení vyváženosti po vložení nového uzlu (*New*). Vznikne kritický uzel (*A*). Kritický uzel i jemu podřízené uzly se posunou zleva doprava. Časová náročnost celé prováděné rotace je konstantní.

Nechť *A* je kritický uzel a *B* jeho levý potomek. Tato varianta rotace nastane v případě, že váha uzlu *A* je rovna -2, váha uzlu *B* je -1 a podstromy napojené na tyto uzly mají stejnou výšku. Není potřeba speciálně ošetřovat zda je vložený prvek pravým nebo levým potomkem. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *B*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů *A* a *B* na hodnotu 0.

Celá rotace je znázorněna na obrázku O*br. 1.5*:



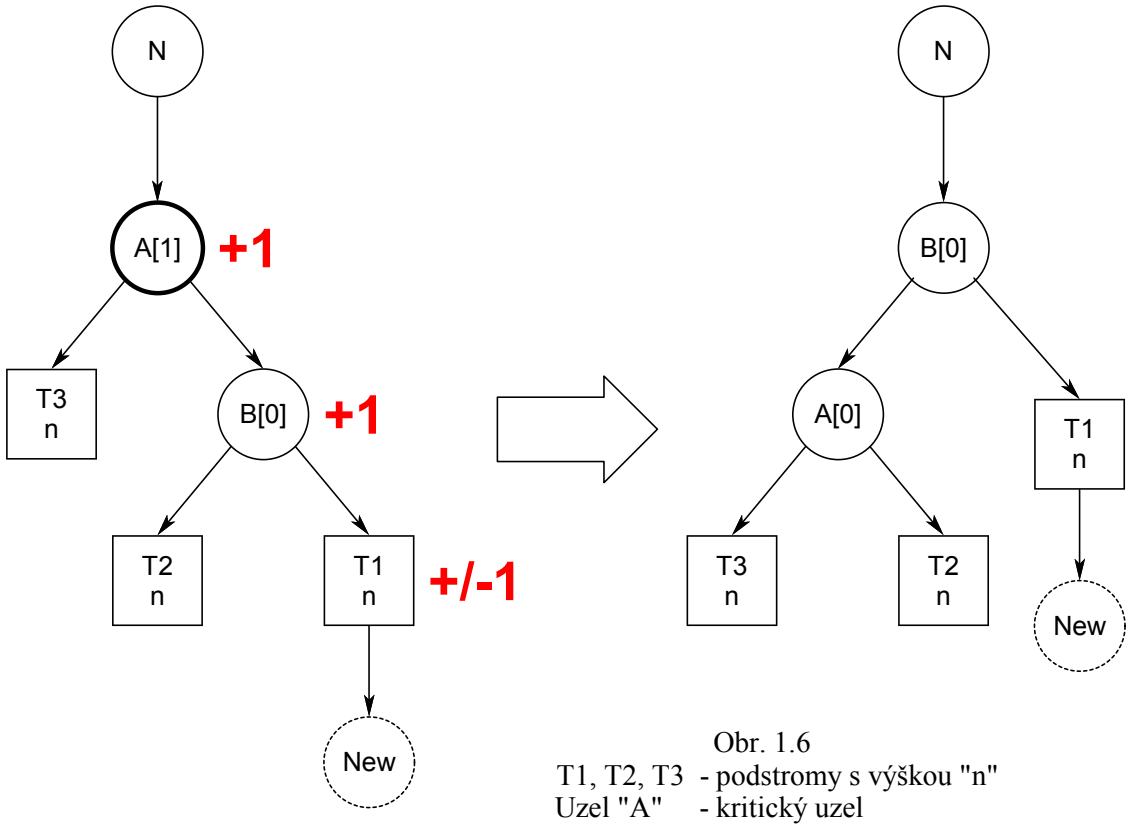
Obr. 1.5
T1, T2, T3 - podstromy s výškou "n"
Uzel "A" - kritický uzel
New - nově vložený uzel
N - nad-kritický uzel

1.2.2 Jednoduchá rotace RR pro vkládání

Jednoduchá rotace Right-Right (dále jen RR) koriguje porušení vyváženosti po vložení nového uzlu (*New*). Vznikne kritický uzel (*A*). Kritický uzel i jemu podřízené uzly se posunou zprava doleva. Časová náročnost celé prováděné rotace je konstantní.

Nechť *A* je kritický uzel a *B* jeho levý potomek. Tato varianta rotace nastane v případě, že váha uzlu *A* je rovna 2, váha uzlu *B* je 1 a podstromy napojené na tyto uzly mají stejnou výšku. Není potřeba speciálně ošetřovat zda je vložený prvek pravým nebo levým potomkem. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *B*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů *A* a *B* na hodnotu 0.

Celá rotace je znázorněna na obrázku O*br. 1.6*:

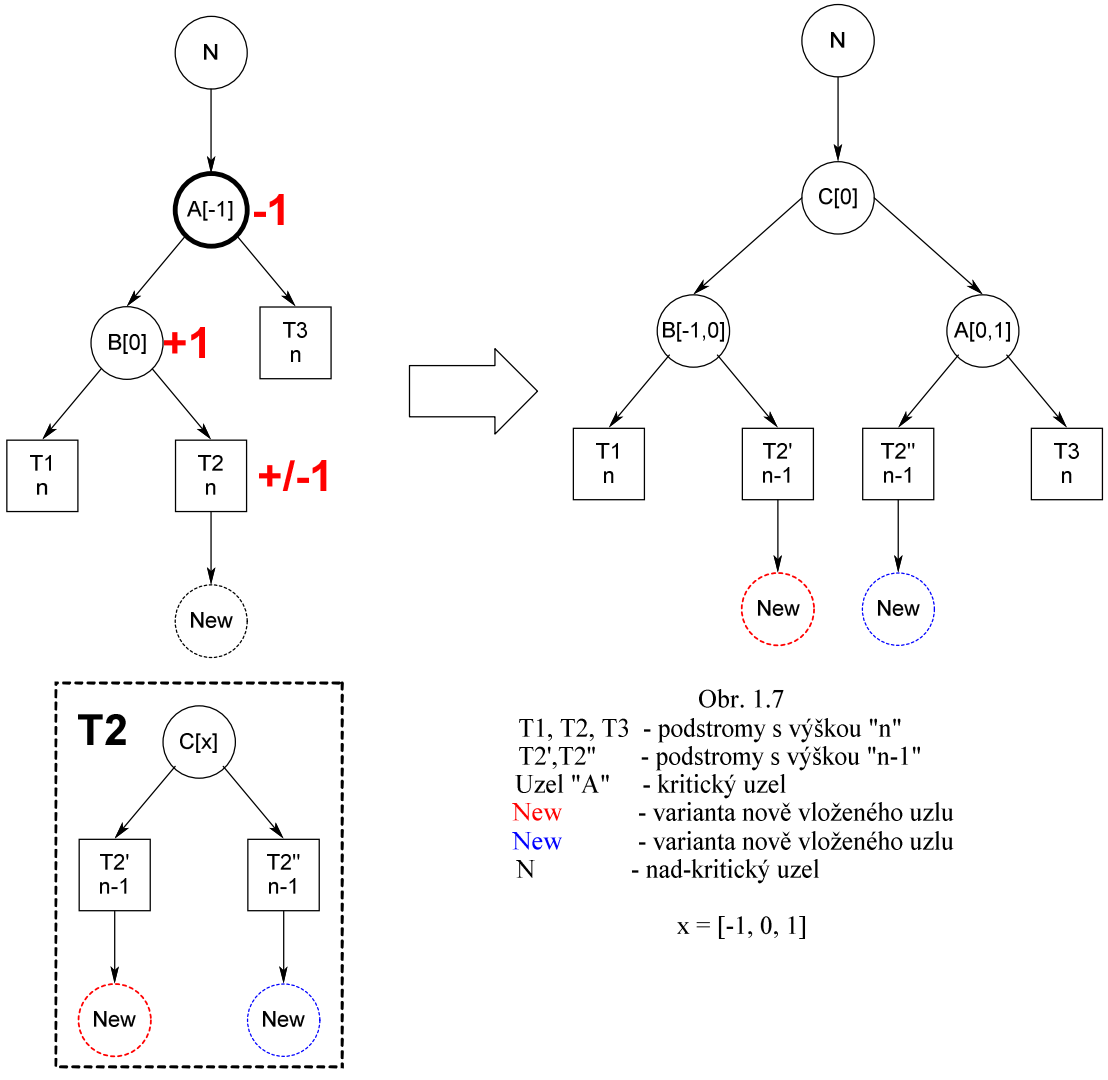


Obr. 1.6
T1, T2, T3 - podstromy s výškou "n"
Uzel "A" - kritický uzel
New - nově vložený uzel
N - nad-kritický uzel

1.2.3 Dvojitá rotace DLR pro vkládání

Dvojitá rotace rotace Left-Right (dále jen DLR) koriguje porušení vyváženosti po vložení nového uzlu (*New*). Vznikne kritický uzel(*A*). Nechť *A* je kritický uzel, *B* je levý potomek uzlu *A* a *C* je pravý potomek uzlu *B*. Nejdříve se provede rotace vlevo uzlů *B* a *C*, poté rotace vpravo uzlů *A* a *C*. Časová náročnost celé prováděné rotace je konstantní.

Tato varianta nastane v případě, že váha uzlu *A* je rovna -2, uzel *B* má váhu 1 a uzel *C* váhu -1, 0 nebo 1. Pravý podstrom napojený na uzel *A* má stejnou výšku jako podstromy napojené na uzel *B*. Vkládaný uzel je buď pravým nebo levým potomkem pravého podstromu uzlu *B*. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *C*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů - *C* na hodnotu 0 a váhy uzlů *A* a *B* podle toho, na který podstrom uzlu *C* byl vložen nový uzel.
Celá rotace je znázorněna na obrázku *Obr. 1.7*:



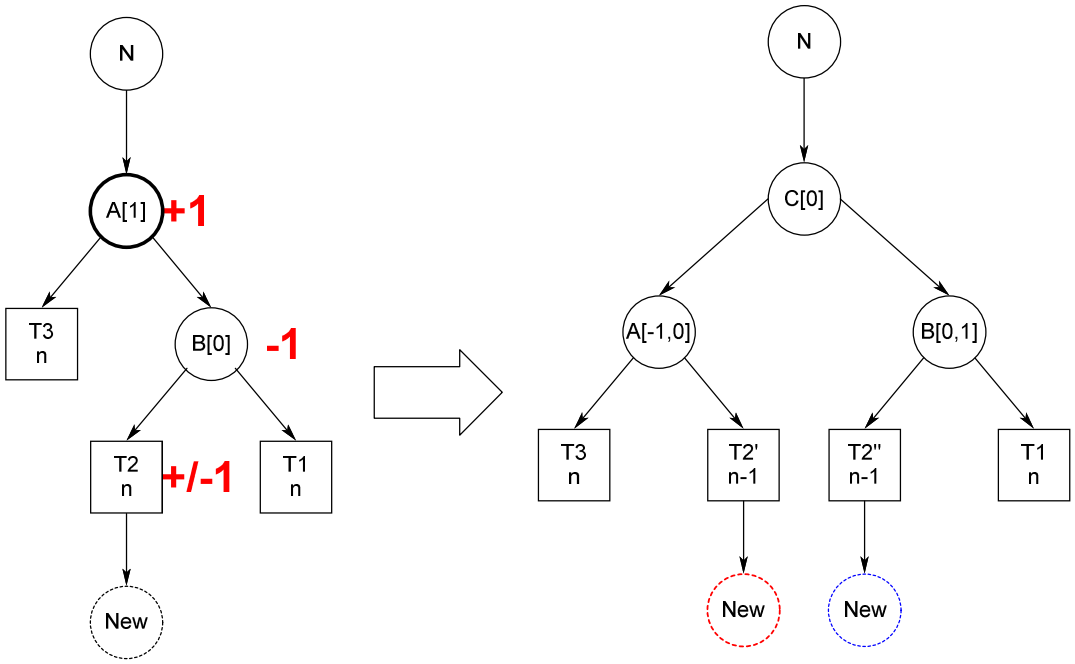
Obr. 1.7
T1, T2, T3 - podstromy s výškou "n"
T2',T2" - podstromy s výškou "n-1"
Uzel "A" - kritický uzel
New - varianta nově vloženého uzlu
New - varianta nově vloženého uzlu
N - nad-kritický uzel

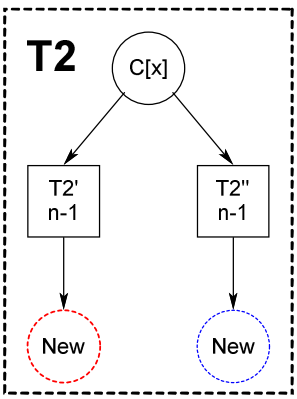
x = [-1, 0, 1]

1.2.4 Dvojitá rotace DRL pro vkládání

Dvojitá rotace rotace Right - Left (dále jen DRL) koriguje porušení vyváženosti po vložení nového uzlu (*New*). Vznikne kritický uzel (*A*). Nechť *A* je kritický uzel, *B* je pravý potomek uzlu *A* a *C* je levý potomek uzlu *B*. Nejdříve se provede rotace vpravo uzlů *B* a *C*, poté rotace vlevo uzlů *A* a *C*. Časová náročnost celé prováděné rotace je konstantní.

Tato varianta nastane v případě, že váha uzlu *A* je rovna 2, uzel *B* má váhu -1 a uzel *C* váhu -1, 0 nebo 1. Levý podstrom napojený na uzel *A* má stejnou výšku jako podstromy napojené na uzel *B*. Vkládaný uzel je buď pravým nebo levým potomkem pravého podstromu uzlu *B*. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *C*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů - *C* na hodnotu 0 a váhy uzlů *A* a *B* podle toho, na který podstrom uzlu *C* byl vložen nový uzel.
Celá rotace je znázorněna na obrázku *Obr. 1.8*:





Obr. 1.8
T1, T2, T3 - podstromy s výškou "n"
T2',T2" - podstromy s výškou "n-1"
Uzel "A" - kritický uzel
New - varianta nově vloženého uzlu
New - varianta nově vloženého uzlu
N - nad-kritický uzel

x = [-1, 0, 1]

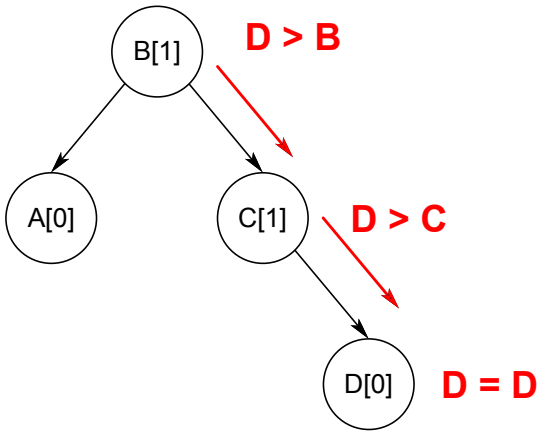
1.3 Rušení uzlu

Rušení uzlu AVL stromu vychází z binárně vyhledávacích stromů. Časová náročnost zrušení uzlu je $O(\log n)$. Nejprve vyhledáme rušený uzel podle zadaného klíče, v případě že uzel neexistuje, strom zůstane nezměněn. Lze řešit, jak rekurzivně, tak nerekurzivně. V případě, že byl uzel nalezen, může nastat několik případů:

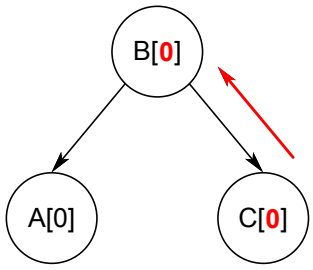
- Uzel je listem, lze ho bez problému odstranit. (*Obr. 2.1*, *Obr. 2.2*)
- Uzel má jednoho potomka, pak je rušený uzel nahrazen tímto potomkem. (*Obr. 2.3*, *Obr. 2.4*)
- Uzel má dva potomky, hodnota obsahu a klíče rušeného uzlu je nahrazena nejbližší nižší (nejpravější uzel levého podstromu - *Obr. 2.5*, *Obr. 2.6*) nebo vyšší hodnotou (nejlevější uzel pravého podstromu - *Obr. 2.7*, *Obr. 2.8*). Takový uzel má nanejvýš jednoho potomka, lze jej tedy ze stromu vyjmout podle jednoho z předchozích pravidel.

Následně je potřeba upravit váhy uzlů směrem od rušeného uzlu ke kořeni stromu. Zpětná úprava vah může skončit pokud je váha procházeného uzlu 1 nebo -1, což indikuje, že se výška jeho podstromu nezměnila. Pokud je váha 0, je potřeba pokračovat ve zpětné úpravě vah. Není-li splněna podmínka vyváženosti stromu tzn. některý z uzlů má váhu +/-2 je potřeba provést cyklickou záměnu uzlů neboli rotaci. V důsledku rotace může dojít ke změně výšky podstromu tzn. váha kořenového uzlu podstromu po rotaci je rovna 0, v tomto případě je nutné pokračovat ve zpětné úpravě vah. Toto je v rozporu s rotacemi při vkládání uzlu, kde nulová váha uzlu říká, že nedošlo ke změně výšky podstromu. Rotace můžeme rozdělit na jednoduché a dvojité z nichž každá má svoji stranovou variantu, více o těchto rotacích v podkapitolách 1.3.1 – 1.3.4.

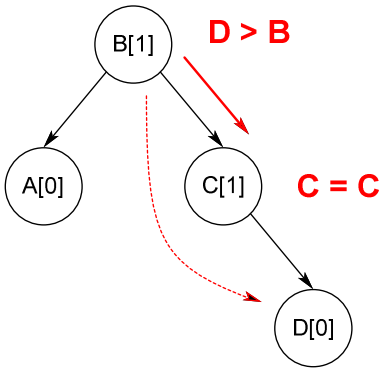
Obr. 2.1
Diagram zrušení listového (terminálního) uzlu s klíčem 'D'



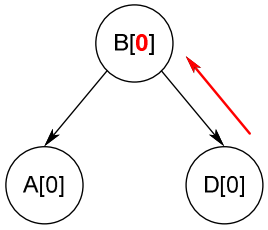
Obr. 2.2
Diagram úpravy vah uzlů po zrušení uzlu s klíčem 'D'



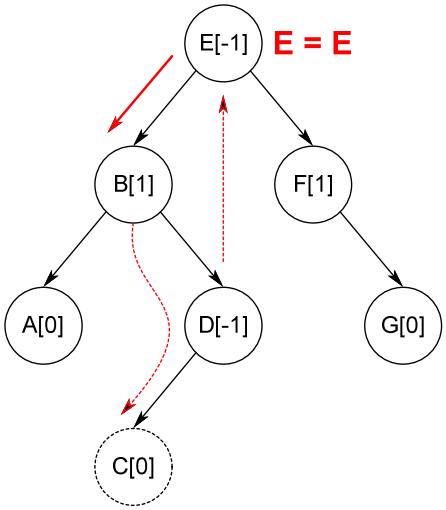
Obr. 2.3
Diagram zrušení
neterminálního uzlu s klíčem 'C'



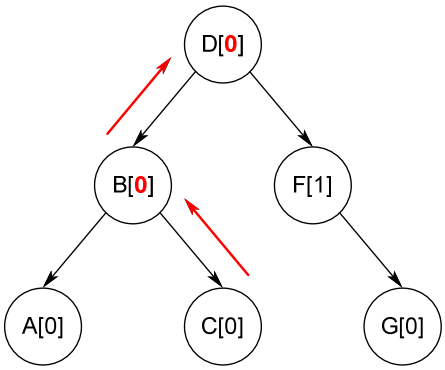
Obr. 2.4
Diagram úpravy vah uzlů
po zrušení uzlu s klíčem 'C'



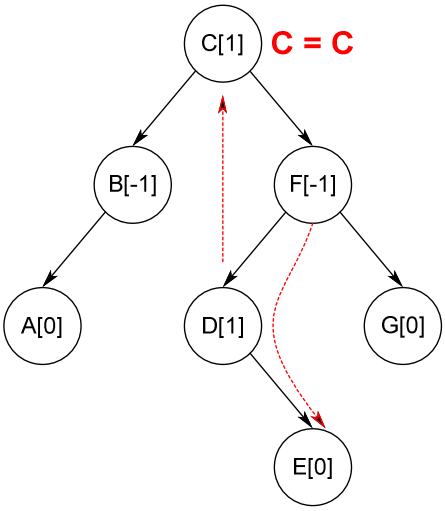
Obr. 2.5
Diagram rušení uzlu s klíčem 'E'
Nahrazení uzlu nejpravějším uzlem jeho levého potomka



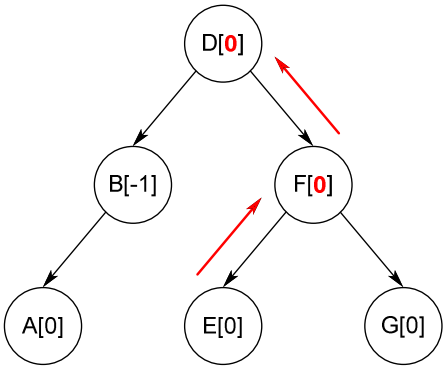
Obr. 2.6
Diagram úpravy vah uzlů
po zrušení uzlu s klíčem 'E'



Obr. 2.7
Diagram rušení uzlu s klíčem 'C'
Nahrazení uzlu nejlevějším uzlem jeho pravého potomka



Obr. 2.8
Diagram úpravy vah uzlů
po zrušení uzlu s klíčem 'C'

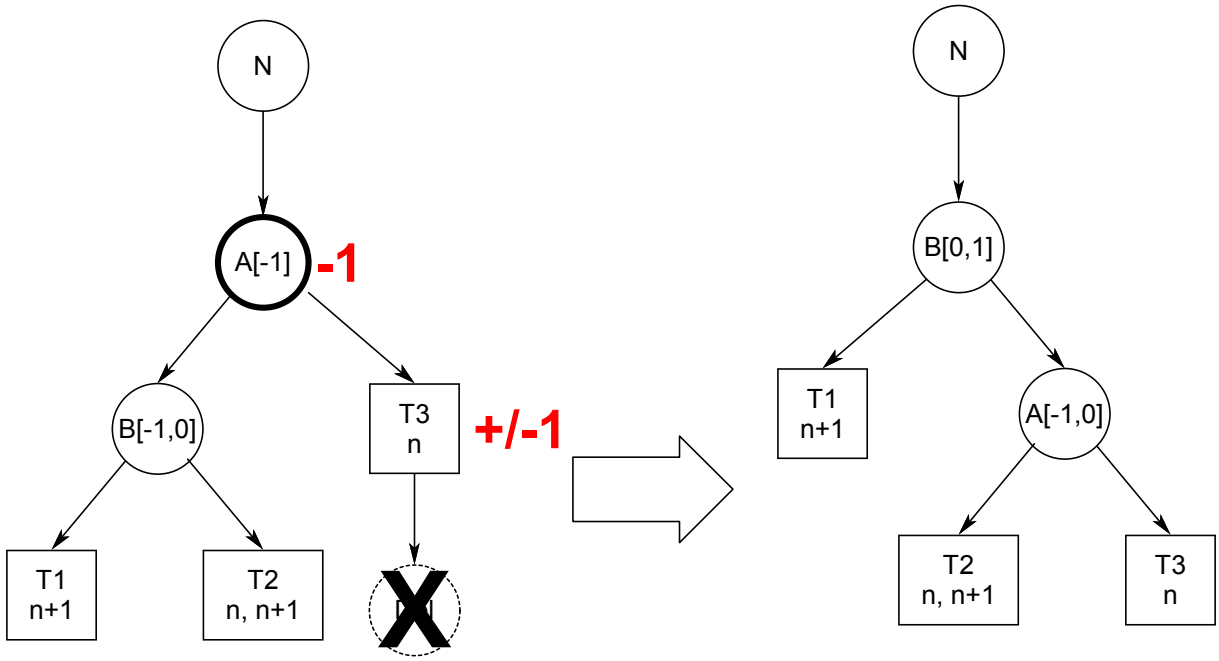


1.3.1 Jednoduchá rotace LL pro rušení

Jednoduchá rotace Left-Left (dále jen LL) koriguje porušení vyváženosti po zrušení uzlu stromu (*Del*). Vznikne kritický uzel (*A*). Kritický uzel i jemu podřízené uzly se posunou zleva doprava. Časová náročnost celé prováděné rotace je konstantní.

Nechť *A* je kritický uzel a *B* jeho levý potomek. Tato varianta rotace nastane v případě, že váha uzlu *A* je rovna -2 a výška vyššího podstromu napojeného na uzel *B* je o jedna větší než výška pravého podstromu uzlu *A*, na kterém se nachází rušený uzel. Není potřeba speciálně ošetřovat zda je rušený uzel pravým nebo levým potomkem. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *B*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů *A* a *B* podle toho, jaké byly výšky stromů *T1* a *T2*.

Celá rotace je znázorněna na obrázku O*br. 3.1*:



Obr. 3.1
T1, T2, T3 - podstromy s výškou "n" nebo "n+1"
Uzel 'A' - kritický uzel
Del **X** - rušený uzel
N - nad-kritický uzel

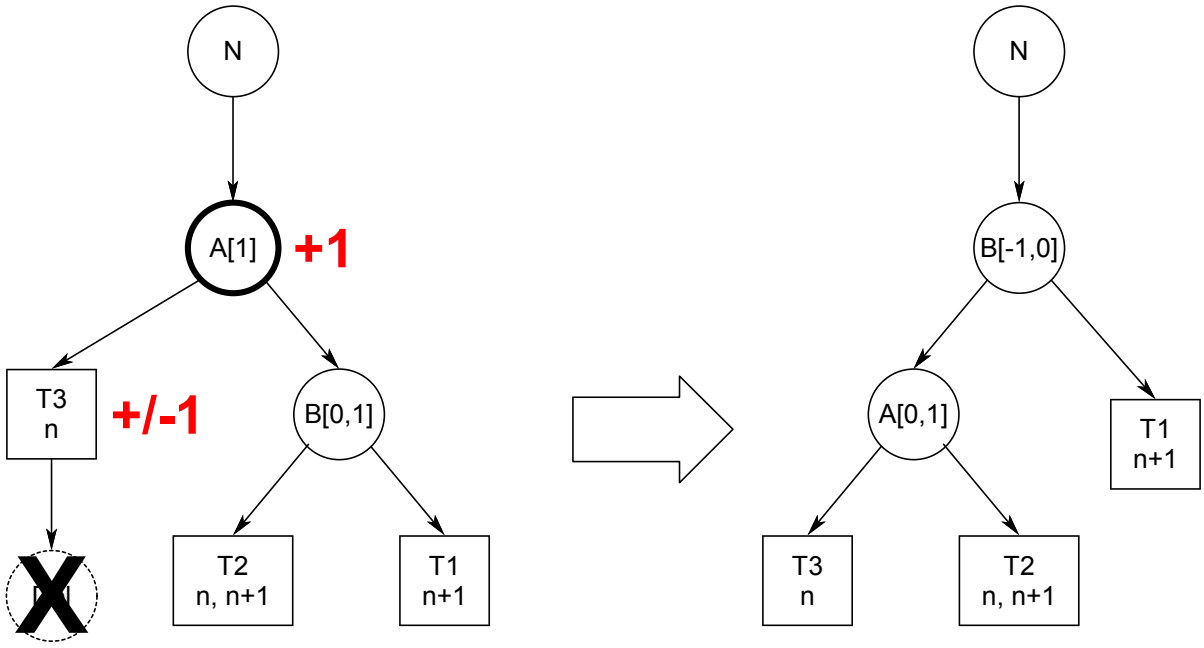
Podm.:
T1[n] >= T2[n] && T1[n] == T3[n+1]

1.3.2 Jednoduchá rotace RR pro rušení

Jednoduchá rotace Right-Right (dále jen RR) koriguje porušení vyváženosti po zrušení uzlu stromu (*Del*). Vznikne kritický uzel (*A*). Kritický uzel i jemu podřízené uzly se posunou zprava doleva. Časová náročnost celé prováděné rotace je konstantní.

Nechť *A* je kritický uzel a *B* jeho pravý potomek. Tato varianta rotace nastane v případě, že váha uzlu *A* je rovna 2 a výška vyššího podstromu napojeného na uzel *B* je o jedna větší než výška levého podstromu uzlu *A*, na kterém se nachází rušený uzel. Není potřeba speciálně ošetřovat zda je rušený uzel pravým nebo levým potomkem. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *B*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů *A* a *B* podle toho, jaké byly výšky stromů *T1* a *T2*.

Celá rotace je znázorněna na obrázku O*br. 3.2*:



Obr. 3.2
T1, T2, T3 - podstromy s výškou "n" nebo "n+1"
Uzel 'A' - kritický uzel
Del **X** - rušený uzel
N - nad-kritický uzel

Podm.:
T1[n] >= T2[n] && T1[n] == T3[n+1]

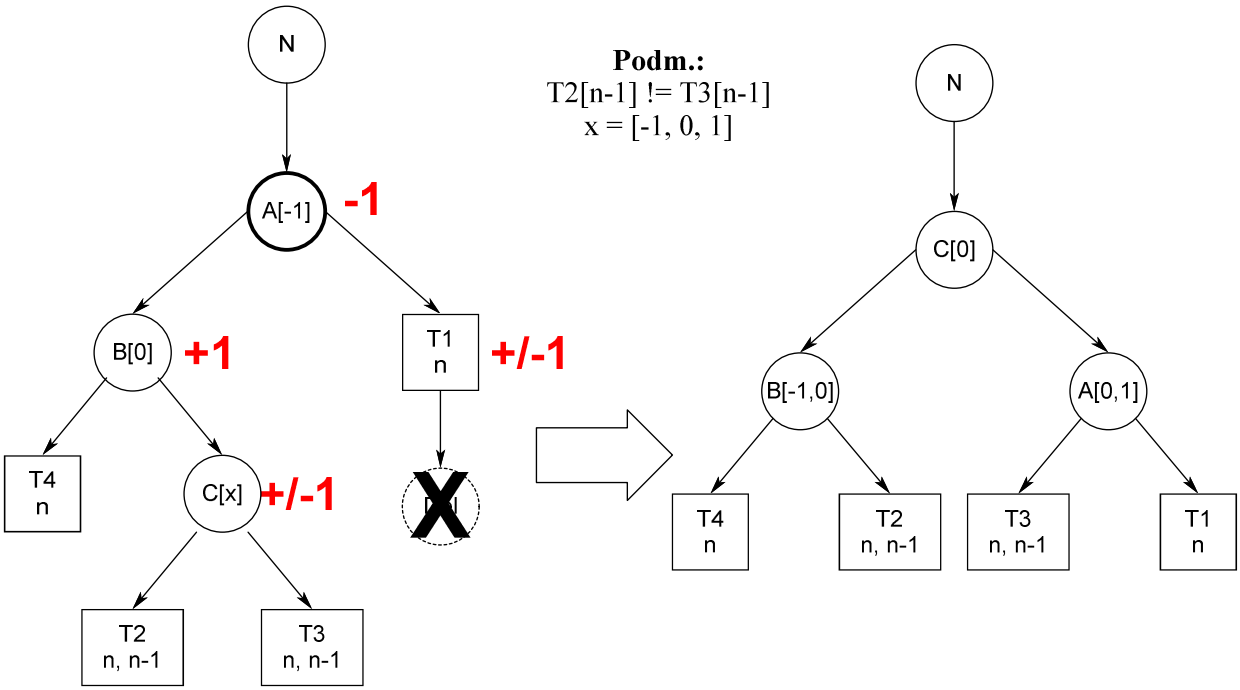
1.3.3 Dvojitá rotace DLR pro rušení

Dvojitá rotace rotace Left-Right (dále jen DLR) koriguje porušení vyváženosti po zrušení uzlu stromu (*Del*). Vznikne kritický uzel(*A*). Nechť *A* je kritický uzel, *B* je levý potomek uzlu *A* a *C* je pravý potomek uzlu *B*. Nejdříve se provede rotace vlevo uzlů *B* a *C*, poté rotace vpravo uzlů *A* a *C*. Časová náročnost celé prováděné rotace je konstantní.

Tato varianta nastane v případě, že váha uzlu *A* je rovna -2, uzel *B* má váhu 1 a uzel *C* váhu -1, 0 nebo 1. Výška vyššího podstromu uzlu *C* je stejná jako výška pravého podstromu uzlu *B* a levého podstromu uzlu *A*, na který je napojen rušený uzel. Výška podstromů *T2* a *T3* nesmí být současně o jedna menší než výška podstromu *T1* a *T4*. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *C*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů - *C* na hodnotu 0 a váhy uzlů *A* a *B* podle toho, jaké byly výšky podstromů *T2* a *T3*.

Celá rotace je znázorněna na obrázku *Obr. 3.3*:

Obr. 3.3
Uzel "A" - kritický uzel
Del **X** - rušený uzel
N - nad-kritický uzel
T1, T2, T3, T4 - podstromy s výškou "n" nebo "n-1"



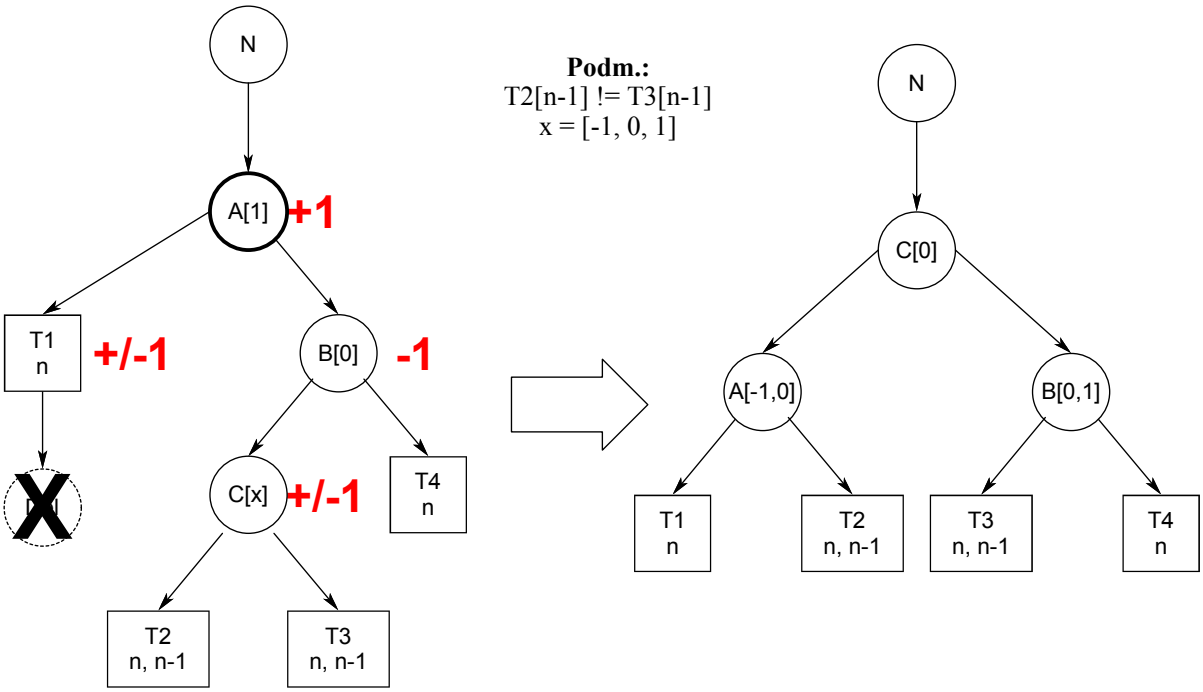
1.3.4 Dvojitá rotace DRL pro rušení

Dvojitá rotace rotace Right - Left (dále jen DRL) koriguje porušení vyváženosti po zrušení uzlu stromu (*Del*). Vznikne kritický uzel(*A*). Nechť *A* je kritický uzel, *B* je pravý potomek uzlu *A* a *C* je levý potomek uzlu *B*. Nejdříve se provede rotace vpravo uzlů *B* a *C*, poté rotace vlevo uzlů *A* a *C*. Časová náročnost celé prováděné rotace je konstantní.

Tato varianta nastane v případě, že váha uzlu *A* je rovna 2, uzel *B* má váhu -1 a uzel *C* váhu -1, 0 nebo 1. Výška vyššího podstromu uzlu *C* je stejná jako výška pravého podstromu uzlu *B* a levého podstromu uzlu *A*, na který je napojen rušený uzel. Výška podstromů *T2* a *T3* nesmí být současně o jedna menší než výška podstromu *T1* a *T4*. Po provedení rotace je potřeba opravit (pokud je to nutné) napojení nad-kritického uzlu z uzlu *A* na uzel *C*. Samozřejmě také nesmíme zapomenout aktualizovat koeficienty vyváženosti uzlů - *C* na hodnotu 0 a váhy uzlů *A* a *B* podle toho, jaké byly výšky podstromů *T2* a *T3*.

Celá rotace je znázorněna na obrázku *Obr. 2.4*:

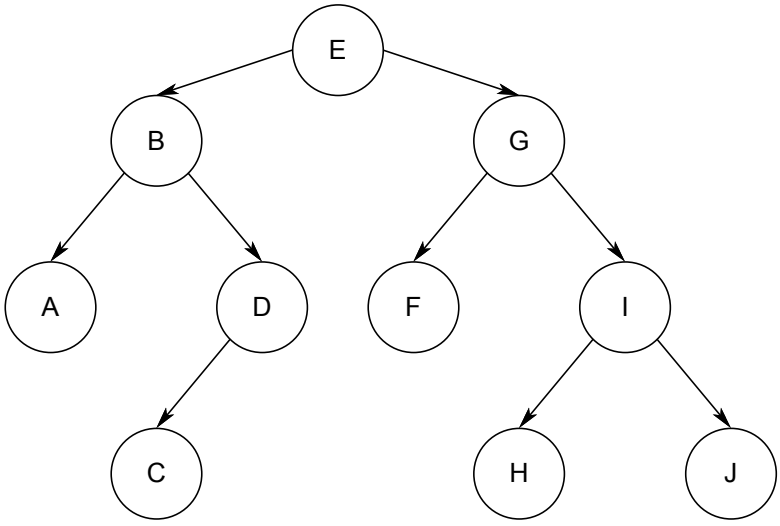
Obr. 2.4
Uzel "A" - kritický uzel
Del **X** - rušený uzel
N - nad-kritický uzel
T1, T2, T3, T4 - podstromy s výškou "n" nebo "n-1"



1.4 Metody průchodu stromem

Průchod stromem je posloupnost všech uzlů stromu, v níž se žádný uzel nevyskytuje dvakrát. Průchod transformuje nelineární strukturu stromu na lineární. Metody průchodu můžeme rozdělit na: procházení stromu do hloubky a do šířky. Budeme se jimi zabývat v následujících kapitolách. Na obrázku *Obr. 3.1* je zobrazen náhodně vytvořený AVL strom, na kterém si ukážeme jednotlivé typy průchodů.

Obr. 3.1
Příklad AVL stromu



1.4.1 Prohledávání do šířky

Prohledávání do šířky (anglicky „level-order“) je algoritmus či metoda, která systematickým prohledáváním stromu postupuje přes všechny uzly. Strom se prochází postupně po vrstvách nebo-li hladinách. Jinak řečeno, nejprve se procházejí uzly zleva s výškou nula, poté s výškou jedna atd. Algoritmus tohoto průchodu je poměrně složitý a pro mé řešení nepotřebný, proto se jím nebudu zabývat. Při použití této metody bude mít průchod stromem na obrázku *Obr.3.1* následující tvar:
E, B, G, A, D, F, I, C, H, J

1.4.2 Prohledávání do hloubky

Procházení začíná na kořeni stromu a pokaždé se postupuje na potomky daného vrcholu. Procházení končí, když v žádné větvi již není další potomek. V nadcházejících třech podkapitolách se budeme zabývat různými metodami průchodu do hloubky, které se od sebe liší v pořadí, ve kterém se procházejí jednotlivé uzly. Průchodů stromem se typicky využívá pro destrukci stromu, výpis stromu nebo vytváření kopií stromu.

1.4.3 PreOrder

Tato metoda se nejčastěji využívá při vytváření kopie stromu. Zjednodušený rekurzivní algoritmus tohoto průchodu je následující:

a) Proveď akci.

b) Projdi levou větev.

c) Projdi pravou větev.

Při použití této metody bude mít průchod stromem na obrázku *Obr.3.1* následující tvar:

E, B, A, D, C, G, F, I, H, J

1.4.4 InOrder

Často se tato metoda používá na binárně vyhledávacích stromech, jelikož při použití této metody se prochází uzly podle jejich přirozeného pořadí. Zjednodušený rekurzivní algoritmus tohoto průchodu je následující:

a) Projdi levou větev.

b) Proveď akci.

c) Projdi pravou větev.

Při použití této metody bude mít průchod stromem na obrázku *Obr.3.1* následující tvar:

A, B, C, D, E, F, G, H, I, J

1.4.5 PostOrder

Tato metoda se zdá být vhodná pro destrukci stromu. Zjednodušený rekurzivní algoritmus tohoto průchodu je následující:

a) Projdi levou větev.

b) Projdi pravou větev.

c) Proveď akci.

Při použití této metody bude mít průchod stromem na obrázku *Obr.3.1* následující tvar:

A, C, D, B, F, H, J, I, G, E

2 Implementace BP

Tato kapitola obsahuje stručnou charakteristiku implementovaných funkcí a datových typů. Příklady byly implementovány s ohledem na provedenou analýzu v předcházející kapitole. Tato kapitola obsahuje tři podkapitoly, každá podkapitola se zabývá jednou sadou úloh.

Důležitým faktorem při vypracovávání těchto příkladů bylo jejich dostatečné komentování tak, aby student měl při jejich řešení dostatečné informace a byl schopen dané funkce bez problémů řešit. Především bylo dbáno na to, aby před definicí každé funkce byl jasně popsán cíl této funkce, požadavky na vstupní a výstupní hodnoty, postup při řešení těchto funkcí a na které záležitosti by si student měl dávat zvláštní pozor – tím je myšleno například použití ukazatelů na ukazatele. Studentovy znalosti ještě nemusí být na takové úrovni, aby s nimi dokázal bezproblémově pracovat a proto se v komentářích objevují rady, jak s těmito ukazateli pracovat a jak je předávat dalším funkcím.

Dále bylo nutné se držet pravidel stávajícího systému pro automatické zadávání a hodnocení domácích úloh. Začátek a konec vzorově vyřešeného kódu bylo potřeba ohraničit značkami /*v*/, kdy se po zadání těchto úloh do systému automaticky odstraní kód vymezený těmito značkami. Vzniknou tedy funkce neobsahující kód a student je podle přiložených informací doplní.

Ve všech studentem řešených funkcích je využita nadeklarovaná proměnná SOLVED, určující řešenost příslušné funkce. Pokud student danou funkci řeší, obsahuje tedy nějaký smysluplný kód, smaže definici této proměnné. Tohoto se využívá z důvodu výrazného zrychlení vyhodnocování a zabránění možným pádům programu.

Cílem těchto příkladů není jenom studenta naučit jeden postup při programování algoritmů pro AVL stromy, ale také aby měl široký přehled o tom, jakými cestami se lze při vytváření těchto algoritmů pohybovat. Rozhodl jsem se, že první sada úloh bude řešena čistě rekurzivně, druhá naopak čistě nerekurzivně a třetí bude směsicí obou. Samozřejmostí také je, aby studentem doplněný kód byl řádně komentován. Student je na to upozorněn a pokud nesplní tento požadavek lze očekávat postihy, jelikož nekomentovaný kód je špatný kód.

Pro zpracování druhé a třetí sady úloh jsou nutné operace nad zásobníky. Jelikož cílem těchto úloh není studenta naučit práci se zásobníky, jsou v těchto sadách pro studenty předpřipraveny základní funkce pracující nad těmito zásobníky.

2.1 Volba datových typů

Abstraktní datový typ AVL strom reprezentuji kořenovým ukazatelem stromu typu tAVLNodePtr. Tato struktura obsahuje následující položky:

- ukazatel LPtr na levý a ukazatel RPtr na pravý podstrom, obě položky typu tAVLNode.
- Key typu char obsahující vyhledávací klíč.
- Cont typu int obsahující hodnotu uzlu.
- Bal typu reprezentující váhu daného uzlu.

Dále používám zásobníkové struktury typu StackP a StackB. Struktura StackP je zásobník na hodnoty typu tAVLNodePtr. Obsahuje dvě položky, z nichž jedna je jednorozměrné pole obsahující hodnoty typu tAVLNodePtr a druhá je typu int udávající index poslední přidané hodnoty. Struktura StackB je zásobník na hodnoty typu bool. Obsahuje dvě položky, z nichž jedna je jednorozměrné pole obsahující hodnoty typu bool a druhá je typu int udávající index poslední přidané hodnoty.

2.2 Vkládání prvků do AVL stromu

Soubor těchto funkcí je řešen v úloze c501. Při řešení tohoto problému lze postupovat několika směry. Hlavní specifikací této úlohy je, že veškeré funkce zmíněné v této kapitole budou řešeny, pokud to půjde, rekurzivně. Ne vždy je tato metoda výhodná, ale jde především o procvičení studentových znalostí o rekurzi. Dále lze několika způsoby vyhledat tzv. kritický uzel, studenti mají za úkol možné kritické uzly zapisovat již při cestě za místem, kam bude vložen nový uzel.

V každé podkapitole bude vysvětlena činnost jedné či více funkcí. Bude vysvětleno, jaký má daná funkce smysl, požadavky na vstupní a výstupní parametry, jak jsem postupoval při její implementaci a na jaké případné problémy jsem narazil. Vycházel jsem z kapitoly 1.2.

2.2.1 AVLInit

Počáteční inicializace stromu před prvním použitím struktury AVL stromu, strom je dán jediným parametrem funkce. Inicializace je nutná z důvodu počátečního testování stromu, jelikož strom obsahuje nedefinovanou (libovolnou) hodnotu.

2.2.2 AVLSearch

Vyhledávání uzlu v AVL stromu podle zadaného klíče K. Pokud je uzel nalezen, vrátí funkce hodnotu TRUE a v proměnné Content vrátí obsah vyhledaného uzlu. V případě, že nalezen není, vrátí funkce hodnotu FALSE a obsah proměnné Content zůstane nedefinován.

Vycházím z teorie vyhledávání nad BVS viz. kapitola 1.1. Funkce je řešena rekurzivně, jak je požadováno. Kód této funkce sestává ze tří částí. Nejprve proběhne kontrola prázdnosti uzlu, pokud je výsledek kladný, vyhledávání skončí. V opačném případě se porovná vyhledávaný klíč s klíčem uzlu a při shodě se hodnotu uzlu vrátí v proměnné Content. Jinak se rekurzivně zavolá tato funkce na levý nebo pravý podstrom, podle toho zda byl vyhledávaný klíč menší či větší.

2.2.3 RotLL, RotRR, RotDLR, RotDRL

Vycházím z kapitol 1.2.1 až 1.2.4. Všechny procedury zmíněné v této podkapitole jsou v zásadě totožné. Procedury RotLL a RotRR jsou jednoduché rotace, od sebe se lišící pouze stranovou verzí. Procedury RotDLR a RotDRL jsou dvojité rotace, od sebe se lišící pouze stranovou verzí.

Pro všechny procedury platí, že nejprve provedu rekonfiguraci uzlů, tzn. vhodně zaměním jednotlivé pravé a levé ukazatele uzlů viz. Obr. 1.5 ž 1.8. Poté opravím napojení rekonfigurovaného uzlu na jeho rodiče. Pokud se rekonfigurace účastnil kořenový uzel, musím opravit ukazatel na tento kořenový uzel. Na samý závěr upravím podle typu funkce, její varianty a případnému směru napojení, jednotlivé váhy uzlů a proceduru ukončím.

2.2.4 nodalloc

Jednoduchá funkce pro alokaci a inicializaci nového uzlu stromu. Využívám dynamické alokace paměti s kontrolou nedostatku paměti. Po alokaci místa inicializuji uzel a nastavím na požadované hodnoty položky klíč a obsah. Tyto položky jsou předány jako parametry této funkce. Váha uzlu je při vytváření nového uzlu vždy nulová. Na závěr nastavím ukazatel na ukazatel předaný funkci na právě vytvořený uzel.

2.2.5 SearchCrit

Funkce vyhledá kritický a nadkritický uzel ve stromu a vrátí je jako parametry funkce. Kritický uzel je nejvzdálenější uzel od kořene, v němž je v důsledku vkládání porušena rovnováha. Řeším opět rekurzivně, přestože rekurzivní způsob není u této funkce příliš vhodný. Jak již ale bylo řečeno, jedná se pouze o zažití programování pomocí rekurzí.

Vycházím z algoritmu pro vyhledávání uzlu ve stromu. Pokud je strom prázdný, skončím a vrátím hodnotu FALSE. Jestliže narazím na uzel se shodným klíčem, vrátí funkce hodnotu TRUE a přepíši obsah uzlu proměnnou Content. V opačném případě si zaznamenám potencionálně kritický a nadkritický uzel a pokračuji rekurzivně v levé nebo pravé větvi uzlu, podle hodnoty klíče.

2.2.6 AVLAddNode

Tato funkce upraví váhu kritického uzlu (předán parametrem) a váhy uzlů ve větvi, na kterou po dosažení konce stromu vložím nový uzel. Řešeno opět rekurzivně.

Pokud je klíč kritického uzlu větší než klíč vkládaného uzlu, upravím váhu kritického uzlu zmenšením hodnoty váhy o jedna. Jestliže je levý syn uzlu prázdný (konec stromu) vložím na toto místo nová uzel zavoláním procedury nodalloc. V opačném případě volám rekurzivně funkci na levý podstrom. Pokud je klíč kritického uzlu menší než klíč vkládaného uzlu, upravím váhu kritického uzlu zvětšením hodnoty váhy o jedna. Jestliže je pravý syn uzlu prázdný (konec stromu) vložím na toto místo nová uzel zavoláním procedury nodalloc. V opačném případě volám rekurzivně funkci na pravý podstrom.

2.2.7 AVLInsert

Tato funkce kompletně zajišťuje vložení nového uzlu. Za pomoci dalších funkcí zajišťuje alokaci nového uzlu, vyhledání kritického a nadkritického uzlu, úpravu vah jednotlivých uzlů a v případě, že je to potřeba, zavolání příslušné rotace.

Pokud je strom prázdný, volám proceduru nodalloc pro vložení nového uzlu a následně proceduru ukončím. Poté volám funkce na vyhledání (nad)kritického uzlu a na samotné vložení nového uzlu se současným upravením vah uzlů. Po provedení těchto funkcí, zkontroluji zda nedošlo k porušení výškové rovnováhy stromu, tzn. že zkontroluji, zda hodnota váhy kritického uzlu není +/- dvě. V tomto případě zavolám příslušnou rotaci, podle váhy potomka kritického uzlu určím zda se jedná a jednoduchou či dvojitou rotaci a následně proceduru ukončím.

2.2.8 RecAVLDispose

Kompletně uvolní celý binární vyhledávací AVL strom. Vzhledem k tomu, že je jedno, v jakém pořadí budeme prvky odstraňovat, zvolil jsem metodu průchodu stromem PostOrder.

Nejprve zkontroluji prázdnost stromu, poté rekurzivně zavolám funkci nejdříve na levý, poté na pravý podstrom a poté uvolním uzel.

2.3 Rušení prvku AVL stromu

Soubor těchto funkcí je řešen v úloze c502. Při řešení tohoto problému lze postupovat několika směry. Hlavní specifikací této úlohy je, že veškeré funkce zmíněné v této kapitole budou řešeny nerekurzivně. Ne vždy je tato metoda výhodná, ale jde především o procvičení studentových znalostí.

V každé podkapitole bude vysvětlena činnost jedné či více funkcí. Bude vysvětleno, jaký má daná funkce smysl, požadavky na vstupní a výstupní parametry, jak jsem postupoval při její implementaci a na jaké případné problémy jsem narazil. Vycházel jsem z kapitoly 1.3.

2.3.1 AVLDelInit

Počáteční inicializace stromu před prvním použitím struktury AVL stromu, strom je dán jediným parametrem funkce. Inicializace je nutná z důvodu počátečního testování stromu, jelikož strom obsahuje nedefinovanou (libovolnou) hodnotu.

2.3.2 AVLSearchPush

Vyhledávání rušeného uzlu v AVL stromu podle zadaného klíče K. Pokud je uzel nalezen, vrátí funkce tento uzel. V případě, že nalezen není, vrátí funkce hodnotu NULL. Při vyhledávací cestě za rušeným uzlem (počínaje kořenem) ukládám do zásobníku ukazatele uzlů a směr, jímž se z uzlu postupuje dále (true pro levou stranu, false pro pravou).

Vycházím z teorie vyhledávání nad BVS viz. kapitola 1.1. Funkce je řešena nerekurzivně, jak je vyžadováno. Kód této funkce sestává z podmínky na test prázdnosti stromu a z cyklu, zajišťující samotný průchod stromem společně s testem na nalezení rušeného uzlu. Tělem cyklu je podmínka určující, zda budeme pokračovat v pravém či levém podstromu. Poté, pokud není potomek aktuálního uzlu prázdný, uložím jeho ukazatel a směr na zásobník a pokračuji v cyklu na tomto potomku. Jinak funkci ukončím a vrátím hodnotu NULL. V případě úspěšného nalezení rušeného uzlu vrátím tento uzel a funkci ukončím.

2.3.3 DelRotLL, DelRotRR, DelRotDLR, DelRotDRL

Vycházíme z kapitol 1.3.1 až 1.3.4. Všechny procedury zmíněné v této podkapitole jsou v zásadě totožné. Procedury DelRotLL a DelRotRR jsou jednoduché rotace pro rušení, od sebe se lišící pouze stranovou verzí. Procedury DelRotDLR a DelRotDRL jsou dvojité rotace pro rušení, od sebe se lišící pouze stranovou verzí.

Pro všechny procedury platí, že nejprve provedu rekonfigurace uzlů, tzn. vhodně zaměním jednotlivé pravé a levé ukazatele uzlů viz. *Obr. 2.1* až *2.4*. Poté opravím napojení rekonfigurovaného uzlu na jeho rodiče. Pokud se rekonfigurace účastnil kořenový uzel, musím opravit ukazatel na tento kořenový uzel. Na samý závěr upravím podle typu funkce, její varianty a případnému směru napojení, jednotlivé váhy uzlů a proceduru ukončím.

2.3.4 AVLDelNode

Vycházím z kapitoly 1.3. Tato procedura uvolní uzel stromu, předaný jí jako parametr. Procedura sestává ze tří hlavních podmínek, které odlišují různé případy počtu potomků rušeného uzlu. Pokud je splněna podmínka, že rušený uzel nemá pravého potomka, tak nastane případ, že rušený uzel má pouze levého nebo žádného potomka. V tomto případě načtu ze zásobníku ukazatelů rodičovský uzel, ze zásobníku booleanů směr napojení z rodičovského uzlu a přepíši levý(pravý) ukazatel rodičovského uzlu levým potomkem rušeného uzlu, ať už je prázdný či nikoliv. Na závěr vrátím rodičovský uzel a směr zpět na zásobník a uvolním samotný uzel. V případě, že je splněna druhá podmínka, má rušený uzel pouze pravého potomka a postupujeme stejně jako v minulé podmínce pouze s opačnými uzly. Poslední podmínka bude splněna, pokud má rušený uzel oba podstromy, v tom případě zavolám funkci ReplaceByRightmost viz. následující kapitola.

2.3.5 ReplaceByRightmost

Pomocná procedura, která vyhledá, přestěhuje a uvolní nejpravější uzel levého podstromu daného uzlu. Opět vycházím z kapitoly 1.3. a řeším nerekurzivně.

Postupně procházím pravou větev levého podstromu rušeného uzlu, procházené uzly, a směr, ze kterého se pohybuji ukládám na zásobníky. Toho je dosaženo jednoduchým nekonečným cyklem. Po nalezení nejpravějšího uzlu, přepíši obsah a klíč rušeného uzlu hodnotami z tohoto uzlu a následně nejpravější uzel uvolním viz. obrázek *Obr. 2.5*.

2.3.6 AVLDelete

Tato procedura kompletně obstará zrušení uzlu. Za pomocí dalších funkcí jednak zajišťuje vyhledání rušeného uzlu, jeho uvolnění, úpravu vah uzlů postižených uvolněním uzlu a v případě, že je to potřeba, zavolání příslušné rotace.

Pokud je strom prázdný, není co rušit a ukončím proceduru. Jestliže je strom neprázdný, zavolám funkci AVLSearch pro vyhledání rušeného uzlu a následně ho pomocí funkce AVLDelNode uvolním. Následuje závěrečný cyklus, který se skládá ze dvou téměř identických částí, rozdílné jsou pouze ve směru napojení uzlu a ve stranové verzi. Budu se zabývat pouze jednou částí, druhá lze jednoduše odvodit. Nejprve vyjmu ukazatel na kritický uzel ze zásobníku, následně vyjmu ze zásobníku směr napojení uzlu a podle toho se rozhodnu, v které stranové části kódu budu pokračovat.

Posléze upravím váhu uzlu, pakliže je váha uzlu +/- jedna nebo je uzel uzlem kořenovým, můžeme skončit. V opačném případě je potřeba provést příslušnou rotaci nad kritickým uzlem. Pokud se během rotace změní výška podstromu tzn. kořenový uzel tohoto zrotovaného podstromu má výšku nula, je nutné pokračovat dále v úpravě vah, jinak ukončím cyklus a proceduru.

2.3.7 AVLDispose

Kompletně uvolní celý binární vyhledávací AVL strom. Vzhledem k tomu, že je jedno, v jakém pořadí budeme prvky odstraňovat, zvolil jsem metodu průchodu stromem PostOrder.

Nejprve zkontroluji prázdnost stromu a případně proceduru ukončím, poté se pomocí cyklu posunu na nejlevější uzel se současným uložením procházených uzlů na zásobník. Následně odebírám uzly ze zásobníku a posunuji se opět na nejlevější uzel jejich pravého podstromu (opět ukládám uzly na zásobník), poté uvolním uzel, odeberu uzel ze zásobníku a pokračuji dál. Po uvolnění celého stromu je ještě nutno inicializovat kořenový uzel, aby bylo možné provádět testování.

2.4 Stromové etudy nad AVL stromy

Soubor těchto funkcí je řešen v úloze c503. Vzhledem k jisté složitosti předchozích dvou úloh byla zvolena pro tuto úlohu složitost výrazně menší a slouží jen jako doplnění znalostí o AVL stromech. Příklady v této úloze jsou jednou řešeny jak rekurzivně tak nerekurzivně.

V každé podkapitole bude vysvětlena činnost jedné funkce. Bude vysvětleno, jaký má daná funkce smysl, požadavky na vstupní a výstupní parametry, jak jsem postupoval při její implementaci a na jaké případné problémy jsem narazil. Vycházel jsem z kapitoly 1.4.

2.4.1 AVLTree_Height,

Funkce zjistí a vrátí výšku AVL stromu. Řešeno rekurzivně.

Pokud je uzel prázdný, vrátí funkce hodnotu nula. Pokud je váha uzlu menší nebo rovna nule, rekurzivně volám tuto funkci na levý podstrom se současným navýšením výšky stromu o jedna. V opačném případě rekurzivně volám tuto funkci na pravý podstrom se současným navýšením výšky stromu o jedna.

2.4.2 Non_AVLTree_Height

Funkce zjistí a vrátí výšku AVL stromu. Řešeno nerekurzivně.

Celá funkce sestává z jednoho cyklu probíhajícího, dokud je uzel neprázdný. Tělo cyklu obsahuje navýšení proměnné určující výšku stromu a podmínku, zda se má pokračovat v levém či pravém podstromu uzlu - to je určenou vahou uzlu.

2.4.3 AVLTree_Copy

Procedura vytvoří kopii stromu. Používám metodu PreOrder k průchodu stromem. Řeším rekurzivně.

Pokud je originální uzel prázdný, pak je prázdná i kopie uzlu. Následně vytvořím kopii uzlu a naplním ji hodnotami z uzlu originálního. Poté rekurzivně zavolám funkci na levý a pravý podstrom.

2.4.4 Leftmost

Pomocná procedura pro proceduru Non_AVLTree_Copy. Pohybujete se po levé větvi podstromu, až narazí na jeho nejlevější uzel. Ukazatele na všechny navštívené uzly ukládá do zásobníku a současně vytváří jejich kopii. Stromy jsou dány jako ukazatele v parametrech procedury. Řeším rekurzivně.

Pokud je uzel originální uzel prázdný, je i kopie uzlu prázdná a ukončím proceduru. V opačném případě vytvořím kopii uzlu pomocí procedury nodalloc, naplním ji hodnotami uzlu originálního a uložím jak originální uzel, tak jeho kopii, na zásobník. Poté se posunu na další uzel v levé větvi originálu i kopie a celou akci opakuji dokud se nedostanu na nejlevější uzel. Na závěr nastavím levý podstrom nejlevějšího uzlu na NULL.

2.4.5 Non_AVLTree_Copy

Procedura vytvoří kopii stromu. Používám metodu PreOrder k průchodu stromem a využívám procedury Leftmost pro průchod po levé diagonále. Stromy jsou dány jako ukazatele v parametrech procedury. Řeším rekurzivně.

Nejprve inicializuji zásobníky, následně zavolám proceduru Leftmost, pro průchod hlavní levou diagonálou. Poté pomocí cyklu vyjímám uzly vložené na zásobník a volám funkci Leftmost na pravý podstrom tohoto vyjmutého uzlu. Pokud je zásobník prázdný procedura končí.

2.5 Základní a rozšířené testy

Součástí úloh pro rušení a vkládání uzlu do AVL stromu jsou i testy kontrolující jejich funkčnost. Dá se říct, že tyto testy jsou nejdůležitější částí celého programu, jelikož vyhodnocují správnost řešení jednotlivých funkcí. Jednotlivé typy testů se mezi sebou liší rozsahem a důkladností jednotlivých testů. Budu mluvit o všech testech společně, jelikož funkce zajišťující jejich práci jsou ve všech příkladech víceméně totožné.

Každá funkce kontroluje určitou sadu úloh z příslušného studentem vyplněného souboru. Nejprve se zkontroluje, zda student vůbec danou úlohu řešil, pokud ne, vypíši upozornění na standardní výstup. Poté se zavolá samotný úkol a po jeho dokončení se zhodnotí průběh tohoto úkolu např. vypsáním obsahu stromu. Vypsání stromu je zajištěno speciální funkcí, která do jisté míry vypíše strom na standardní výstup graficky.

Samotný proces testování funguje následovně. Každý test volá jednu či více testovacích funkcí, které mají za úkol navodit nějaký jednoduchý či složitý problém týkající se AVL stromu - např.: vyhledání nějakého prvku, vložení jednoho či více prvků, uvolnění stromu atd. Odtud rozdělujeme testy na základní a rozšířené. V základních testech se nachází testy kontrolující pouze některé části AVL stromu a často se vyskytující problémy, zatímco v rozšířených jsou pokud možno vystiženy všechny možné případy rotací v AVL stromu, jejich stranové varianty, mazání z prázdného stromu atd.

3 Specifikace testů

Během návrhu a implementace se objevuje několik kritických oblastí, které je třeba otestovat. Proto zde vypíši několik testů. Tyto testy jsou výsledkem mého programu.

Test 1: Pokus o zrušení prázdného stromu → Nedojde k pádu aplikace

Výstup: Struktura AVL stromu:

- Strom je prazdny

Test 2: Pokus o vyhledání neexistující položky s klíčem 'E' → Nenalezne se

Výstup: Polozka 'E' NEnalezena!

Test 3: Vložení prvku s klíčem 'D' a hodnotou 1 → Potvrzení výpisem stromu

Výstup: Struktura AVL stromu:

```
+--+=[D,1,0]
```

Test 4: Pokus o vyhledání existující položky s klíčem 'D' → Nalezne se

Výstup: Polozka 'D' nalezena!

Obsahovala hodnotu: [1]

Test 5: Vložíme prvky 'E', 'F', 'C', 'B', 'A', 'G', 'H' → Rotace proběhnou v pořádku

Výstup: Struktura AVL stromu:

```
          +--+=[H,8,0]
          |
       +--+=[G,7,0]
       |  |
       |  +--+=[F,6,0]
       |
    +--+=[E,5,1]
    |  |
    |  +--+=[D,4,0]
    |
+--+=[C,3,1]
    |
    +--+=[B,2,-1]
       |
       +--+=[A,1,0]
```

Test 6: Vložíme prvky 'C', 'A' a 'B' do prázdného stromu → Dvojitá rotace proběhne v pořádku.

Výstup: Struktura AVL stromu:

```
    +--+=[C,3,0]
    |
+--+=[B,2,0]
    |
    +--+=[A,1,0]
```

Test 7: Pokusíme se zrušit uzel nad prázdným stromem → Nedojde k pádu aplikace

Výstup: Struktura AVL stromu:

- Strom je prazdny

Test 8: Vložíme několik uzlů do prázdného stromu a uvolníme kořenový uzel s klíčem 'H' → Správné uvolnění a nahrazení uzlu.

Vstup:

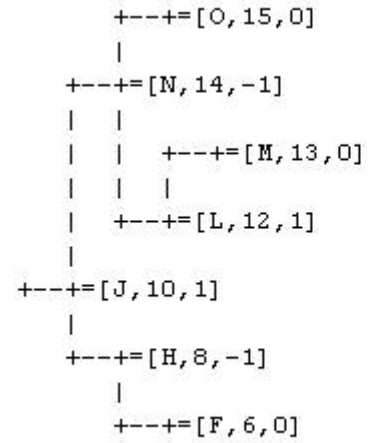
```
        +--+=[O,11,0]
        |
    +--+=[L,1,-1]
    |   |
    |   +--+=[J,0,-1]
    |       |
    |       +--+=[I,76,0]
    |
+--+=[H,18,0]
    |
    |       +--+=[G,23,0]
    |       |
    |   +--+=[F,9,0]
    |   |   |
    |   |   +--+=[E,88,0]
    |   |
    +--+=[D,7,1]
        |
        +--+=[A,6,0]
```

Výstup:

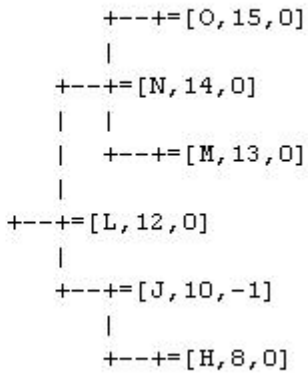
```
        +--+=[O,11,0]
        |
    +--+=[L,1,-1]
    |   |
    |   +--+=[J,0,-1]
    |       |
    |       +--+=[I,76,0]
    |
+--+=[G,23,0]
    |
    |   +--+=[F,9,-1]
    |   |   |
    |   |   +--+=[E,88,0]
    |   |
    +--+=[D,7,1]
        |
        +--+=[A,6,0]
```

Test 9: Vložíme několik uzlů do prázdného stromu a uvolníme uzel s klíčem 'F' → Správné uvolnění a provedení dvojité rotace DRL pro rušení.

Vstup:

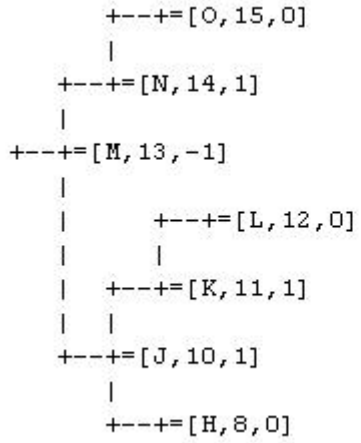


Výstup:

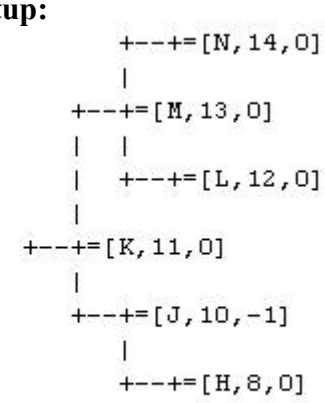


Test 10: Vložíme několik uzlů do prázdného stromu a uvolníme uzel s klíčem 'O' → Správné uvolnění a provedení dvojité rotace DLR pro rušení.

Vstup:



Výstup:



Závěr

Program splňuje všechny požadavky zadání a provádí korektně všechny operace nad AVL stromy. Vzhledem k danému způsobu řešení operací nemusí však být program zcela efektivní a rychlý. Program byl otestován se všemi navrženými testovacími hodnotami a všechny testy proběhly správně podle předpokladů. Aplikace byla úspěšně otestována v prostředí operačních systémů Linux a FreeBSD.

Příklady lze bez problémů použít jako domácí úkoly pro studenty předmětu Algoritmy, bylo dbáno na důkladné komentování kódu a popis definic jednotlivých funkcí tak, aby bylo studentům jasné, co mají v jednotlivých funkcích za úkol. Dále jsem se snažil, aby výstup jednotlivých testů byl smysluplný a studenti z nich dokázali pochopit, co měli dobře případně špatně. Vzhledem k poměrně velké složitosti prvních dvou sad úloh, byla náročnost třetí úlohy značně zjednodušena.

Částečnou nevýhodou tohoto souboru úloh je, že pro základní testování druhé a třetí sady úloh jsou využity funkce z první sady. Pokud bude mít student funkce v první úloze špatně, pak mu tyto testy nebudou fungovat. Bohužel nebylo možné zvolit jiné řešení.

Během řešení celého projektu jsem narazil na několik problémů týkajících se správnosti a úplnosti kódu ve Studijní opoře předmětu Algoritmy. Rozhodně bych tyto nedostatky doporučil odstranit, aby nedocházelo k dezinformaci studentů, kteří z této opory čerpají.

Celý projekt by se dále mohl vyvíjet z hlediska efektivity kódu, již by ale zřejmě nebyl zcela vhodný jako soubor příkladů pro domácí úlohy. V tom případě by příklady mohly například sloužit studentům jako vzorové příklady pro práci s AVL stromy v jiných předmětech (Formální jazyky a překladače). Operace nad AVL stromy lze řešit různými směry, proto by se také projekt mohl dále vyvíjet z hlediska různých variací kódu, kdy by funkce řešené rekurzivně mohly být řešeny nerekurzivně a naopak.

Literatura

[1] Honzík, J.M.: *Studijní opora*, Algoritmy - IAL. Brno, 2007.

[2] Herout, P.*: Učebnice jazyka C*, Kopp, České Budějovice, 1998. ISBN 80-85828-21-9.

[3] Wikipedie: *Otevřená encyklopedie*

Dostupné z WWW: http://www.wikipedia.org

Seznam příloh