VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

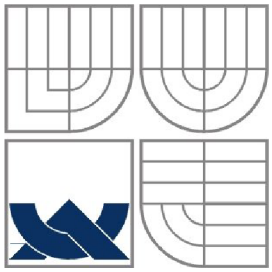
VYHLEDÁVÁNÍ V AVL STROMECH V JAZYCE C

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

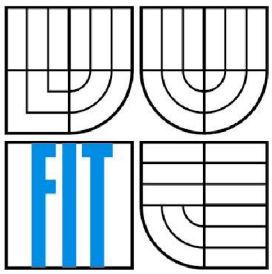
AUTOR PRÁCE
AUTHOR

TOMÁŠ MINTĚL

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYHLEDÁVÁNÍ V AVL STROMECH V JAZYCE C

SEARCHING IN AVL TREES IN C LANGUAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ MINTĚL

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. JAN M. HONZÍK, CSc.

BRNO 2007

Zadání bakalářské práce/4998/2006/xminte00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Mintěl Tomáš**
Obor: Informační technologie
Téma: **Vyhledávání v AVL stromech v jazyce C**
Kategorie: Alg. a datové struktury

Pokyny:

1. Seznamte se detailně s textem kapitoly 4 o rekurzívním i nerekurzívním vyhledávání v AVL stromu ve studijní opoře pro předmět IAL a s doporučenými literárními zdroji vyhldávání AVL
2. Modifikujte text kapitoly zapsané pro pascalovský jazyk tak, aby zachoval co nejvíce (i v detailech) původní obsah, ale aby se vztahoval k zápisu příkladů a algoritmů v jazyce C
3. Převeďte všechny příklady a algoritmy opory do jazyka C.
4. Převeďte algoritmus rekurzívního zápisu operací a vytvořte algoritmy nerekurzívního zápisu operací v jazyce C a odladťe je v prostředí vytrvořeném pro tento účel.
5. Vytvořte text ukázky komentovaných algoritmů pro přílohu studijní opory.
6. Vytvořte program pro animovanou demonstraci rotací v operacích insert a delete s využitím grafických nástrojů pro zobrazení
7. Navrhněte další vhodné kontrolní otázky a příklady.
8. Navrhněte příklady vhodné pro formulářově orientovanou písemnou zkoušku ze znalostí tohoto okruhu.

Literatura:

- Honzík J.M.: Algoritmy. Studijní opora pro předmět Algoritmy, Elektronický text, FIT VUT v Brně
- Honzík J.M. a kolektiv: Vybrané kapitoly z programovacích technik. Skriptum VUT.

Při obhajobě semestrální části projektu je požadováno:

1. Odladěné algoritmy v jazyce C, funkční animovaná demonstrace principu funkce tabulky implementované AVL stromem

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 paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Honzík Jan M., prof. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:	**Tomáš Mintěl**
Id studenta:	84238
Bytem:	17.listopadu 1105/24, 736 01 Havířov
Narozen:	06. 09. 1984, Ostrava

(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“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP:	Vyhledávaní v AVL stromech v jazyce C
Vedoucí/ školitel VŠKP:	Honzík Jan M., prof. Ing., CSc.
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.

---

* hodící se zaškrtněte

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 pořizovaní 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 práce se zabývá vyhledáváním v AVL stromu. Jsou zde vysvětleny principy vyhledávacích metod. Následně je podrobně vysvětlen princip funkčnosti AVL stromu. Cílem je přepsání nerekurzivních algoritmů zapsaných v jazyce Pascal do jazyku C, vytvoření rekurzivních obdob těchto algoritmů a vytvoření aplikace pro animovanou demonstraci těchto operací.

Klíčová slova

Algoritmy, AVL strom, vyhledávaní, jazyk Pascal, jazyk C, C#, demonstrační animace, rotace

Abstract

This thesis engages in search in AVL tree. There are explained principles of researching methods. Consequently is in detail explained principle of utility AVL tree. The main aims of this thesis are transcribing non-recursive algorithms from Pascal to C language, create recursive analogies of this algorithms and create application for animated show of these operations.

Keywords

AVL tree, searching, Pascal language, C language, C#, demonstrational animation, rotation

Citace

Tomáš Mintěl: Vyhledávaní v AVL stromech v jazyce C, bakalářská práce, Brno, FIT VUT v Brně, 2007

Vyhledávaní v AVL stromech v jazyce C

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Ing. Jana Maxmiliána Honzíka, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Mintěl
2.5.2007

Poděkování

Tímto bych rád poděkoval vedoucímu mé bakalářské práce, panu Prof. Ing. Janu Maxmiliánu Honzíkovi, CSc. Děkuji za rady, snahu a ochotu pomoci i za čas věnovaný při konzultacích.

Obsah

1 Úvod

Tato bakalářská práce byla zadána jako přepis algoritmu v programovacím jazyce Pascal do jazyka C. Jedná se o algoritmy operací vkládání, mazání a příslušných rotací nad AVL stromem. Podnětem byla neustále se zvyšující obliba jazyku C mezi programátory. Výuka jazyka Pascal na fakultě informatiky přešla do ústraní a ve většině případů se mladí programátoři učí pouze v jazyku C. V předmětu Algoritmy se sice setkáme při výuce s Algoritmickým jazykem, což je podmnožina Pascalovských instrukcí, ovšem domácí úkoly a projekty se vypracovávají v jazyce C. Součástí je také kontrola studijní opory a její přepis tak, aby vyhovovala zápisu přepsaných algoritmů.

Druhou částí této bakalářské práce je vytvoření rekurzivní obdoby již dříve zmíněných algoritmů. Obě tyto verze by měly sloužit jako příklady používané při výuce.

Posledním bodem je pak vytvoření aplikace pro animační demonstraci jednotlivých funkcí prováděných nad AVL stromem. Aplikace nemá stanovený jazyk, ve kterém má být napsána. Zvolil jsem C#, který je příbuzný jazyku C.

Dokumentace obsahuje 6 kapitol. První je úvod, poté následuje kapitola, v níž je objasněno několik pojmů úzce souvisejících s touto prací. Jedná se o vysvětlení pojmu Algoritmus, programování a popis programovacích jazyků Pascal a C.

V kapitole tři jsou objasněny principy vyhledávání, uvedena jednotlivá kritéria pro hodnocení různých vyhledávacích algoritmů a popsány nejpoužívanější vyhledávací tabulky. Mezi ně patří sekvenční vyhledávaní, nesekvenční vyhledávaní v poli, vyhledávaní v binárním vyhledávacím stromu a vyhledávaní v tabulce s rozptýlenými položkami.

Čtvrtá kapitola je věnována AVL stromu, popisu vlastností celého AVL stromu i jednotlivého uzlu. Následuje vysvětlení principu rotací. Podrobně jsou popsány jak zápisy nerekurzivních, tak rekurzivních operací pro vkládání a mazání.

Pátá kapitola se zabývá samotnou aplikací pro animační demonstraci funkcí nad AVL stromem. Postupně popisuje jak výběr jazyka, tak samotnou implementaci až po uživatelské rozhraní.

2 Algoritmy a programování

2.1 Algoritmus

Algoritmus je konečná uspořádaná množina úplně definovaných pravidel pro vyřešení nějakého problému. Algoritmus musí splňovat několik pravidel a je těmito pravidly definován:

- **Konečnost** – algoritmus musí po určitém počtu kroků skončit, ať už je tento počet jakkoliv velký, vždy musí být konečný. Pokud se program dostane do smyčky, ze které se nikdy nedostane, říkáme tomu „zacyklení“.
- **Hromadnost** – vlastnost, která zaručuje, že algoritmus proběhne správně a nalezne správný výsledek pro jakékoliv vstupní hodnoty. Je nesmysl psát algoritmus pro výpočet 1 + 1, ale není nesmysl napsat algoritmus pro výpočet a + b.
- **Determinovanost** – algoritmus musí v každém kroku vědět, jak pokračovat. Musí být jednoznačně definován. Program nesmí obsahovat žádné „slepé ulice“. Tato vlastnost také zaručuje, že při zadání stejných hodnot na vstupu, bude při každém spuštění výsledek totožný.
- **Efektivnost** – tato vlastnost se nevztahuje ke správnosti výsledku, ale zajišťuje, aby program proběhl co možná nejrychleji. Pokud je v programu několik výpočtů prováděno zbytečně, může to program zpomalit, což je často nepřípustné.

2.2 Programování a programovací jazyky

Programování je zápis algoritmu v programovacím jazyce. Volba programovacího jazyka hraje při programování většinou zásadní roli. Programovacích jazyků je několik desítek. Dělí se podle několika kriterií. Podle míry abstrakce: nižší (assembler) a vyšší (většina jazyků) programovací jazyky. Vyšší programovací jazyky se pak dělí na:

- Procedurální
    - Strukturované (např. Pascal, C, BASIC)
    - Objektově orientované (např. Smalltalk, Java)
- Neprocedurální
    - Funkcionální (např. Lisp)
    - Logické (např. Prolog)

Jazyky jako je například C++ tyto přístupy kombinují.

2.2.1 Pascal

Programovací jazyk Pascal vznikl v 70. letech především pro potřeby výuky programování. Za jeho vznikem stojí profesor Niklaus Wirth z vysoké školy technické v Curychu. Hlavní cíle byli jednoduchost a srozumitelnost. Pascal se rychle rozšířil do světa a až dodnes je hojně používán při výuce. Jeho obliba mezi programátory už určitou dobu klesá a mnohem častěji je nahrazován jazykem C nebo jiným jazykem.

2.2.2 Jazyk C

Vznik je datován na přelom 70. let minulého století. Autory jsou Ken Thompson a Dennis Ritchie z Bellových laboratoří AT&T. Jméno bylo odvozeno z jazyka „B“, ze kterého autoři většinu přebírali. Jazyk byl vytvořen pro potřeby operačního systému Unix. Poté, co byla dokončena práce na stabilní verzi tohoto jazyka, byla většina jádra operačního systému Unix přepsána do tohoto jazyka.

V dnešní době je jazyk C nejpoužívanějším programovacím jazykem na světě, a to jak pro tvorbu malých programů až po rozsáhlé aplikace.

3 Vyhledávání

Při ukládání a skladování velkého množství položek je dobré určitým způsobem je ukládat a mít možnost k nim co možná nejrychleji přistoupit. K tomuto nám slouží abstraktní datový typ „vyhledávací tabulka“.

Jako nejvýznamnější metody toho typu můžeme jmenovat:

- sekvenční vyhledávaní
- nesekvenční vyhledávaní v poli
- vyhledávaní v binárním vyhledávacím stromu
- vyhledávaní v tabulce s rozptýlenými položkami

3.1 Přístupová doba

Tato vlastnost je nejpodstatnější vlastností algoritmů vyhledávaní. Jedná se o dobu potřebnou k vyhledání položky se zadaným klíčem. Pro toto hodnocení se používá hned několik časových přístupů. Navíc se zvlášť musí vzít doba, za kterou byl hledaný prvek nalezen, a doba, za kterou nám vyhledávací algoritmus oznámí, že prvek se zadaným klíčem není obsažen mezi prohledávanými prvky.

Doby, které jsou důležité z hlediska hodnocení algoritmu:

- Minimální doba úspěšného vyhledání
- Minimální doba neúspěšného vyhledání
- Maximální doba úspěšného vyhledání
- Maximální doba neúspěšného vyhledání
- Průměrná doba úspěšného vyhledání
- Průměrná doba neúspěšného vyhledání

Průměrná doba se získá vyhledáním všech prvků ve vyhledávané oblasti a podělí se počtem těchto prvků.

3.2 Vyhledávací klíč

Může být jednoduchý nebo složený. Při jednoduchém se při rozhodování porovnají oba klíče a určí se, prvek s menší a větší hodnotou.

Složený prvek obsahuje několik složek typu jednoduchý klíč. Každá složka má pak určitou váhu neboli prioritu. Porovnání dvou složených klíčů se provádí tak, že se nejdříve porovná složka s nejvyšší prioritou. Při rovnosti se porovná složka s nižší prioritou a tímto způsobem to při rovnosti

pokračuje, než se porovná složka s nejnižší prioritou. Pouze prvky, které mají shodné všechny složky klíče, jsou totožné.

Příkladem může být složený klíč datum. Ten se skládá ze složek rok, měsíc a den. Při vyhledávaní podle věku budou mít prvky tuto prioritu: 1. rok, 2. měsíc, 3. den. Číslo jedna značí nejvyšší prioritu. Pokud budeme ovšem chtít vyhledat podle pořadí narozenin v roce, budou priority změněny: 1. měsíc, 2. den, 3. rok.

3.3 Sekvenční vyhledávaní v poli

Principy sekvenčního vyhledávání lze ilustrovat na sekvenčním vyhledávání v tabulce implementované seznamem s použitím základních operací ATD seznam. Vyhledání je nerovnoměrné podle pozice prvku v poli. Vyhledávaní se provádí postupným prohledáním od prvního prvku k nalezenému, popřípadě poslednímu v případě, že prvek nebyl nalezen. Při N prvcích jsou jednotlivé doby vyhledání tyto:

- Minimální doba vyhledání = 1
- Maximální doba vyhledání = N
- Průměrná doba vyhledání = N/2
- Doba neúspěšného vyhledání = N

Realizace tohoto vyhledávacího algoritmu je jednoduchá, ovšem časová náročnost například u operací delete je vysoká. Proto se tento algoritmus nehodí pro větší objemy dat.

3.4 Nesekvenční vyhledávaní v poli

Bude-li vyhledávací tabulka implementována seřazeným polem, je sekvenční vyhledávaní nevhodné. Mnohem vhodnější je postupovat metodou „rozděl a panuj“ nebo metodou „půlení intervalu“.

3.4.1 Binární vyhledávaní

Při vyhledávaní se hledaný klíč porovná s klíčem prvku, který je v polovině vyhledávaného intervalu. V prvním případě se jedná o celou oblast dat. Klíče se porovnají. Pokud se klíče rovnají, prvek je nalezen. Pokud je hledaný klíč menší, opakuje se stejný proces, ovšem pouze pro interval první prvek až prvek, se kterým jsme teď pracovali. Pokud je větší, prohledá se oblast od daného prvku po poslední prvek. Takto se postupně dělí interval dvěma.

Tímto způsobem nám vznikne logaritmická složitost vyhledání. Základ logaritmu je dvě.

3.4.2 Fibonacciho vyhledávaní

Existuje vzájemný vztah mezi binárním vyhledáváním v seřazeném poli a binárním stromem. Algoritmus Fibonacciho vyhledávaní pracuje podobně jako binární vyhledávaní. Daný interval v poli však nedělí na dvě poloviny, ale dělící bod odvozuje z Fibonacciho posloupnosti a k jeho získaní stačí aditivní operace, což zvýší rychlost vyhledání tam, kde aditivní operace jsou výrazně rychlejší než celočíselné dělení číslem 2.

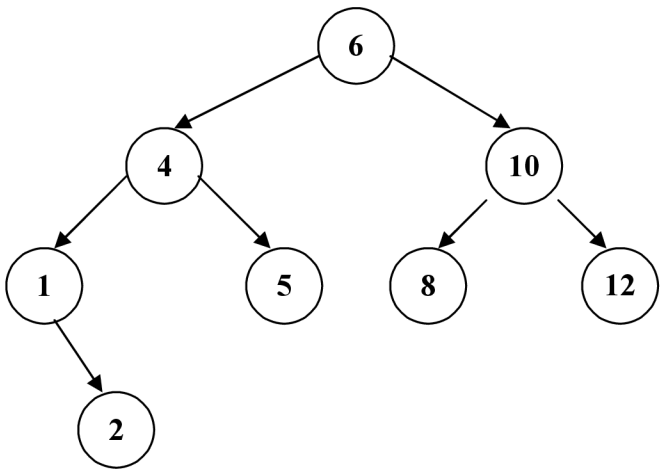
3.4.3 Indexsekvenční vyhledávaní

Stejně jako vyhledáváme ve slovníku, který je opatřen indexy písmen, tak i tato metoda nejprve nalezne podle určitého indexu oblast, v níž by se mohl nalézat hledaný prvek. Následné vyhledávaní pokračuje sekvenčně. Tato metoda je vhodná především pro externí vyhledávaní.

3.5 Vyhledávaní v binárním vyhledávacím stromu

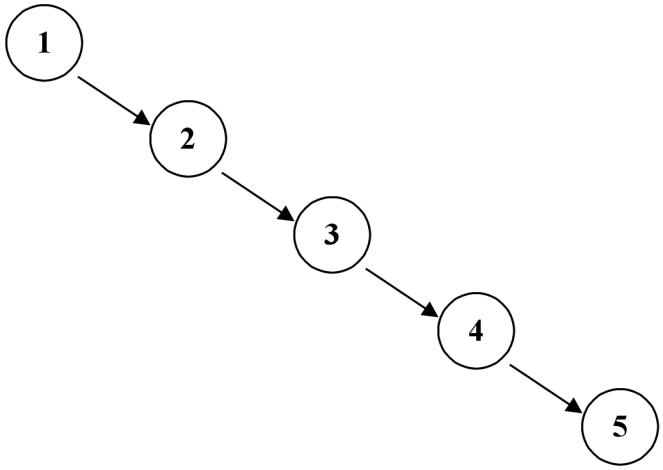
Jedná se o jednu z nejpoužívanějších metod pro plně dynamické vyhledávací tabulky.

Binární vyhledávací strom je takový uspořádaný strom, pro jehož každý uzel platí, že jeho levý podstrom je buď prázdný, nebo sestává z uzlů, jejichž hodnoty klíčů jsou menší než hodnota klíče daného uzlu, a podobně jeho pravý podstrom je buď prázdný, nebo sestává z uzlů, jejichž hodnoty klíčů jsou větší než hodnota klíče daného uzlu.



Obr. 2.5.1 Binární vyhledávací strom

Při nevhodné kombinaci vkládaní může vzniknout strom, který ovšem bude mít vlastnosti sekvence. Například při postupném vložení prvku 1-5 budou časové náročnosti operací tohoto vyhledávacího algoritmu totožné s časovými náročnostmi operací při sekvenčním vyhledávaní.



Obr. 2.5.2 Degradovaný binární vyhledávací strom

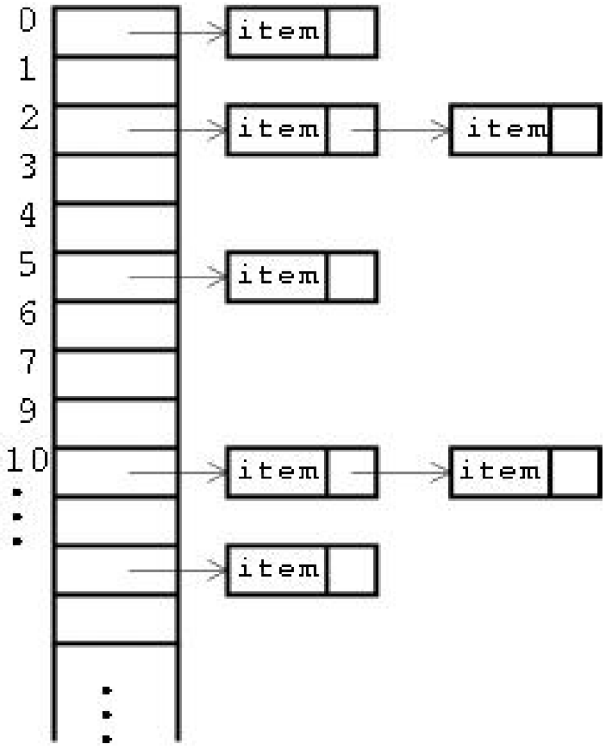
Vkládaní probíhá tak, že se vyhledává uzel s daným klíčem, pokud nalezneme uzel se stejným klíčem, jako je klíč nového uzlu, jsou data aktualizována, to znamená, že se původní data přepíší novými daty. Pokud se dostaneme do uzlu, jehož klíč bude větší než klíč daného uzlu, a zároveň bude jeho levý podstrom prázdný, pak vložíme nový uzel místo tohoto prázdného podstromu. Obdobně tomu je u pravé strany.

Rušení uzlů je mnohem náročnější než vkládaní. Rušíme-li neterminální uzel, který má oba synovské uzly, pak levý synovský uzel připojíme k nadřazenému uzlu místo rušeného uzlu a pravý synovský uzel připojíme k nejpravějšímu listu podstromu levého synovského uzlu (tzv. „levá“ varianta) nebo provedeme stranově sdruženou variantu („pravá“ varianta). Je-li levý nebo pravý synovský prázdný, pak se situace zjednoduší. Na místo rušeného otcovského uzlu se připojí neprázdný synovský podstrom. Ruší-li se kořen, stane se kořenem levý (resp. pravý) synovský uzel a pravý (resp. levý) podstrom se připojí k nejpravějšímu (resp. nejlevějšímu) listu levého (resp. pravého) podstromu.

3.6 Vyhledávaní v tabulce s rozptýlenými položkami

Princip vyhledávání v tabulkách s rozptýlenými položkami (dále jen TRP) je velmi podobný principu vyhledávání v indexsekvenčním souboru. Pomocí rozptylovací funkce se získá index pole, na nějž se

uloží (od něj se vyhledává) položka s daným klíčem. Obsahuje-li tabulka synonyma vzhledem k danému indexu (více různých klíčů mělo shodnou hodnotu rozptylovací), pak na daném indexu začíná lineární seznam synonym, v němž se vyhledává položka s daným klíčem. Vyhledávání v TRP bude účinné tehdy, jestliže počet seznamů synonym bude co největší a jejich délka bude co možná nejkratší.



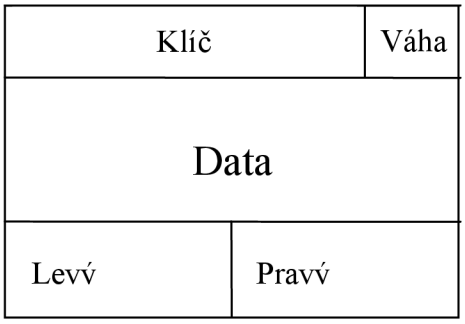
Obr. 2.6 Tabulka s rozptýlenými položkami

Hlavním problémem této metody je nalezení vhodné mapovací funkce. Jak pro 31 různých prvků, které se mají zobrazit do 41prvkové množiny, existuje 41^{31} (tj. cca 10^{50}) možných mapovacích funkcí. Přitom pouze (41!/31!) z nich dává odlišné hodnoty pro různé klíče (tzn. že funkce je jedno-jednoznačná). Tzn. že poměr vhodných funkcí ke všem možným funkcím je asi 1 : 10 miliónům. Jednoznačné funkce jsou tedy velmi řídkým jevem.

4 AVL strom

Za vznikem a zároveň za názvem stojí ruští matematici. Jmenovitě G. M. Adelson-Velsky a E. M. Landis. Ti v roce 1962 popsali tuto datovou strukturu v článku „An algorithm for the organization of information“. V tomto článku popsali jak jednotlivé principy vyvažování při vkládání a mazání prvků, tak i složitost s jakou algoritmus přistupuje k jednotlivým položkám.

4.1 Vlastnosti uzlu



Obr. 3.2 Struktura uzlu

AVL strom je výškově vyvážený binární strom. Je složen z jednotlivých uzlů. Každý uzel obsahuje ukazatel na levý podstrom, ukazatel na pravý podstrom, klíč, data, váhový koeficient (při rekurzivním zápisu není nezbytný).

Ukazatelé na levý a pravý podstrom jsou buďto prázdné, nebo ukazují na další uzel. Hodnota klíče uzlu, na který ukazuje levý ukazatel, je menší než hodnota daného uzlu. V pravém podstromu je pak hodnota uzlu větší. Pokud je jeden z ukazatelů prázdný, druhý může ukazovat pouze na list. Není možné, aby ukazoval na podstrom obsahující více než jeden prvek.

Hlavní a jediný uzel, který je nutno uchovat, abychom měli možnost přistoupit k celému stromu, se obvykle nazývá kořen. Pomocí kořene se můžeme dostat k jakémukoliv uzlu stromu.

Klíč je prvek, podle kterého se prvek ukládá, vyhledává nebo maže. Většinou se jedná o celé číslo nebo řetězec.

Data mohou být jakákoliv. Ať už se jedná jen o jméno a příjmení v nějakém seznamu nebo o objemná data.

Váhový koeficient je číslo v rozmezí -2 až 2. Toto číslo udává vyváženost daného uzlu. Pokud se jedná o rozdíl výšky levého a pravého podstromu, při hodnotě 0 jsou oba podstromy stejně vysoké. Při hodnotě -1 nebo –2 je levý podstrom vyšší než pravý podstrom. Při kladných hodnotách

je vyšší pravý podstrom. Při hodnotě -2 a 2 je uzel nevyvážený a je nazýván kritickým uzlem. Pro tento uzel je nutné zavolat příslušnou rotaci, která upraví okolí tohoto uzlu.

4.2 Rotace

Při operacích vložení a mazání může dojít k porušení vyváženosti uzlu a vzniku kritického uzlu. Proto se po provedení těchto operací provádí kontrola kritického uzlu. Pokud je jeho hodnota rovna 2 nebo -2, je provedena rotace. Jelikož celý AVL strom je postaven na ukazatelích jednoho prvku na další, je třeba provádět rotaci neboli změnu ukazatelů v určitém pořadí.

4.2.1 Rotace při rekurzivním zápisu

Při rekurzivním zápisu operací nad AVL stromem není nutno uchovávat váhu jednotlivých uzlů, proto se provede pouze výměna ukazatelů a to tak, že se vytvoří nové pomocné uzly a pomocí nich se vymění ukazatele podle příslušné rotace. Funkci, která reprezentuje rotaci předáváme kritický uzel. Funkce vrací ukazatel na uzel, který zaujal místo kritického uzlu.

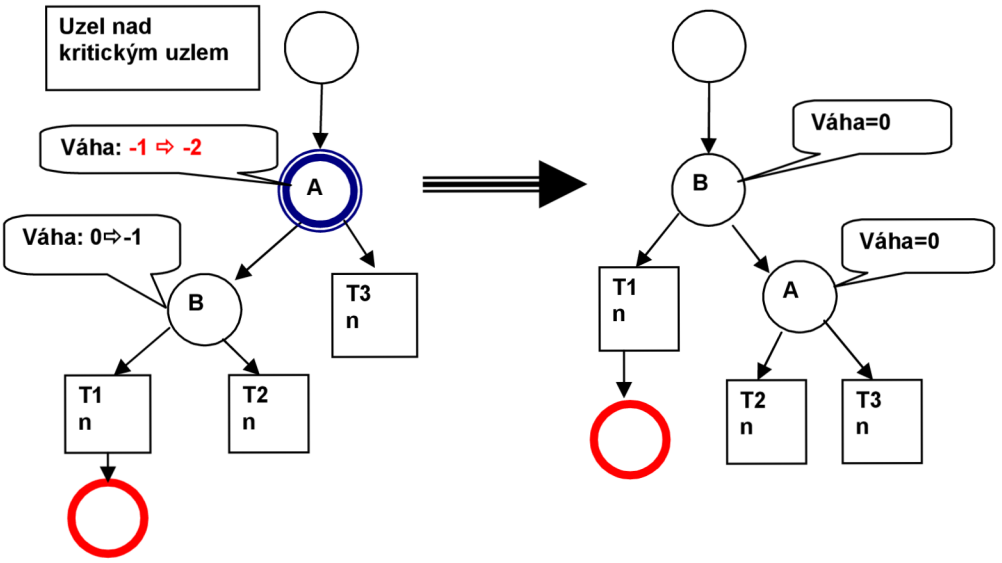
4.2.2 Rotace při nerekurzivním zápisu

Pro rotace při nerekurzivním zápisu operací neplatí totéž co pro operace zapsané rekurzivně. Musíme si uchovávat váhu jednotlivých uzlů, proto provádíme rekonfiguraci zúčastněných uzlů. Je třeba také provést napojení na nadkritický uzel, případně změnit ukazatel na kořenový uzel.

Rekonfigurace uzlů se liší u operace vložení a mazání. Jinak jsou tyto rotace pro obě operace totožné.

Funkcí, která reprezentuje rotaci, předáváme odkaz na kritický uzel, odkaz na nadkritický uzel a odkaz na kořenový uzel. Funkce nevrací žádnou hodnotu.

4.2.3 Jednoduchá pravá rotace



Obr. 3.3.1 Jednoduchá pravá rotace

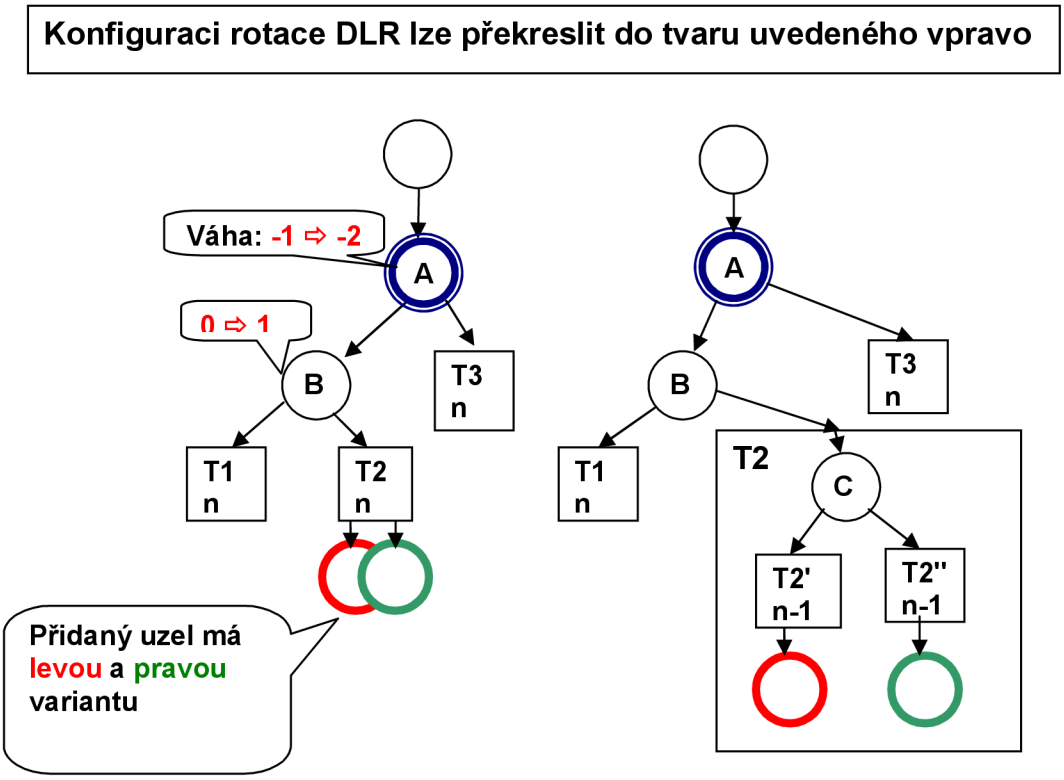
Tato rotace bude zavolána pokud vložíme například uzly s hodnotou 3, 2, 1.
Jednoduchá pravá rotace zapsaná v jazyce C. Jedná se o rotaci při rekurzivním zápisu operací.

```
tAVLTNodePtr SingleRightRotation (tAVLTNodePtr *Koren)
{
  tAVLTNodePtr K1,K2;

  K2 = (*Koren);
  K1 = K2->LPtr;
  K2->LPtr = K1->RPtr;
  K1->RPtr = K2;

  return K1;
}
```

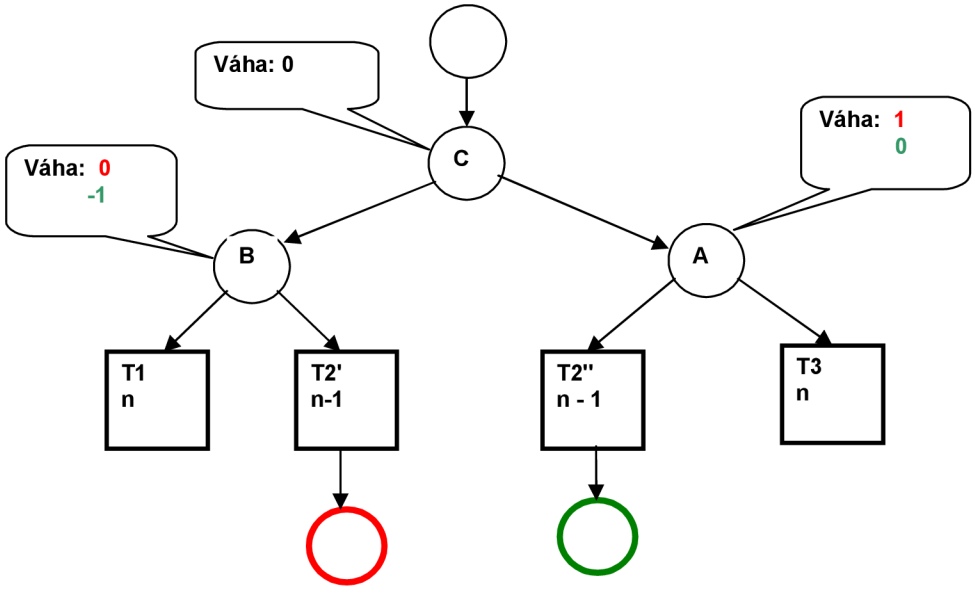
4.2.4 Jednoduchá levá rotace

Jedná se o zrcadlovou verzi jednoduché pravé rotace. Tato rotace bude zavolána, pokud vložíme například uzly s hodnotou 1, 2, 3.

4.2.5 Dvojitá pravá rotace



Obr. 3.3.3 AVL strom po vložení uzlu a před dvojitou pravou rotací.



Obr. 3.3.3 AVL strom po provedení rotace.

Tato rotace bude zavolána, pokud vložíme například uzly s hodnotou 3, 1, 2.

Jednoduchá pravá rotace zapsaná v jazyce C. Jedná se o rotaci při rekurzivním zápisu operací.

```
tAVLTNodePtr DoubleRightRotation (tAVLTNodePtr *Koren)
{
  tAVLTNodePtr K1,K2,K3;

  K3 = (*Koren);
  K1 = K3->LPtr;
  K2 = K1->RPtr;

  K1->RPtr = K2->LPtr;
  K3->LPtr = K2->RPtr;
  K2->LPtr = K1;
  K2->RPtr = K3;

  return K2;
}
```

4.2.6 Dvojitá levá rotace

Jedná se o zrcadlově otočenou verzi dvojité pravé rotace. Tato rotace bude zavolána, pokud vložíme například uzly s hodnotou 1, 3, 2 .

4.3 Vložení uzlu

4.3.1 Rekurzivní zápis vkládání

Funkci se předávají tři parametry. Kořenový uzel, klíč a data. Pokud je kořenový uzel prázdný, pak na toto místo vložíme vkládaný uzel. Pokud není tento uzel prázdný a klíč vkládaného uzlu je větší než klíč kořenového uzlu, zavolá se rekurzivně funkce vkládání, přičemž kořenovým uzlem se stává uzel, na který ukazuje levý ukazatel kořenového uzlu. V opačném případě je klíč vkládaného větší a zavolá se obdobně funkce pro pravý podstrom. Po vložení uzlu postupně vystupujeme z rekurze a kontrolujeme, zda je kritický uzel nevyvážený, a tím pádem je nutné zavolat jednu z rotací. Zde jsou případy, které mohou nastat, a název rotace, která bude zavolána. Váha udává rozdíl výšky pravého a levého podstromu.

- Váha kořenového uzlu je rovna -2 a váha uzlu, na který ukazuje levý ukazatel, není rovna 1, zavolá se jednoduchá pravá rotace
- Váha kořenového uzlu je rovna -2 a váha uzlu na, který ukazuje levý ukazatel, je rovna 1, zavolá se dvojitá pravá rotace
- Váha kořenového uzlu je rovna 2 a váha uzlu, na který ukazuje levý ukazatel, je rovna -1, zavolá se jednoduchá levá rotace
- Váha kořenového uzlu je rovna 2 a váha uzlu, na který ukazuje levý ukazatel, není rovna -1, zavolá se dvojitá levá rotace

4.3.2 Nerekurzivní zápis vkládání

Nerekurzivní zápis je mnohem složitější než zápis rekurzivní, je však mnohem rychlejší, a proto je i přes svou složitost více využíván. Nerekurzivní verze musí používat mnoho proměnných, ať už pro uzel, který je kritický, pro otce kritického uzlu, pro aktivní uzel či pro uzel, který je nad aktivním. Funkci zde předávají stejné parametry jako při rekurzivním zápisu.

Pokud je kořenový uzel prázdný, znamená to, že i celý strom je prázdný, a proto se pouze vytvoří první uzel s daným klíčem. Pokud není kořenový uzel prázdný, pak se dostaneme do cyklu, který slouží k vyhledání vhodného místa pro vložení nového uzlu. Tento cyklus musí především zajistit nastavení všech potřebných proměnných. Zanořování probíhá tak, že se aktivní uzel přepíše uzlem, na který sám ukazuje, tedy levým nebo pravým ukazatelem. Cyklus je řízen proměnnou typu boolean. Dokud je tato proměnná rovna hodnotě false, pak je tento cyklus opakován. Cyklus je ukončen tak, že se do této proměnné zapíše hodnota true. Toto se stane v případě, že se nalezlo místo, kam se uzel vloží, nebo v případě nalezení uzlu se stejným klíčem, který má mít i vkládaný uzel.

Následuje cyklus, který projde uzly od kritického až po uzel, na který se napojí nový uzel. Jelikož při nerekurzivním zápisu musíme ukládat váhový koeficient, je třeba od kritického uzlu všem uzlům přičíst či odečíst jedničku. Až dojdeme do místa, kde vložíme uzel, ukončíme cyklus. Následuje ověření kritického uzlu, na nutnost provedení rotace. Pro výběr rotace platí totéž co u rekurzivního zápisu.

4.4 Mazání uzlu

4.4.1 Rekurzivní zápis mazání

Funkci se předávají dva parametry. Kořenový uzel a klíč. Pokud je kořenový uzel prázdný, pak uzel se zadaným klíčem neexistuje. Pokud není tento uzel prázdný a klíč mazaného uzlu je větší než klíč kořenového uzlu, zavolá se rekurzivně funkce mazání, přičemž kořenovým uzlem se stává uzel, na který ukazuje levý ukazatel kořenového uzlu. V opačném případě je klíč vkládaného větší a zavolá se

obdobně funkce pro pravý podstrom. Pokud není uzel prázdný, větší ani menší, pak jsme mazaný uzel nalezli.

Dále se dělí algoritmus na dvě varianty. Pokud má uzel oba podstromy neprázdné, pak je třeba vyhledat nejpravější uzel z levého podstromu. To znamená nalézt uzel s největším klíčem menším než je mazaný uzel. Poté se klíč a data mazaného uzlu přepíší hodnotami nejpravějšího uzlu a zavolá se funkce mazání pro levý podstrom mazaného uzlu pro klíč nejpravějšího uzlu.

Pokud má uzel alespoň jeden podstrom prázdný, pak se pouze neprázdný podstrom napojí místo mazaného uzlu.

Vždy při výstupu z rekurze se provádí kontrola daného uzlu na vyváženost a případně se zavolá rotace.

4.4.2 Nerekurzivní zápis mazání

Nerekurzivní zápis mazání je nejsložitější operací nad AVL stromem. Je třeba využít zásobníku pro ukládání uzlů, kterými jsme při hledání mazaného uzlu prošli. Do zásobníku je třeba uložit kromě samotného uzlu i to, jestli jsme se dál vydali do levého či pravého podstromu.

Samotný algoritmus obdobně jako u rekurzivního zápisu porovná klíč mazaného uzlu s klíčem aktivního uzlu. Při nerovnosti klíčů se zanořujeme stromem a postupně nasouváme uzly do zásobníku. Při rovnosti, a tím pádem nalezení uzlu, se opět rozdělí podle toho, zda má uzel alespoň jeden prázdný podstrom. Následující postup je totožný s rekurzivním zápisem.

Po nalezení a smazaní uzlu se dostáváme do cyklu, který slouží ke kontrole vyváženosti. Vytahujeme postupně uzly ze zásobníku a kontrolujeme, zda je potřeba provést rotaci.

5 Animovaná demonstrace AVL stromu

Animace jako taková by se měla dělit na dvě trochu odlišné části. První by byla spíše výuková, druhá demonstrační. Animace by měla být jednoduchá jak po stránce ovládaní, tak po stránce možností.

Výuková část pouze ukáže studentovi, jak se přesunou uzly, nastane-li jedna z rotací. Uživatel si pouze vybere, kterou rotaci by rád viděl a s jakou rychlostí.

Demonstrační část má ukázat uživateli, jak vypadá AVL strom po vložení či mazání jednotlivých uzlů. Uživatel může uzly přidávat a odebírat buďto se zadanou hodnotou klíče, popřípadě s náhodně vygenerovanou hodnotou. Může také samozřejmě smazat všechny uzly najednou. Opět může určit, s jakou rychlostí se bude animace provádět. Uživatel také může zobrazit průchod stromem třemi metodam, Pre-Order, In-Order a Post-Order.

5.1 Volba programovacího jazyku

Volba programovacího jazyka byla prvním rozhodováním během psaní bakalářské práce. Vybíral jsem z actionscriptu zastoupeného Flashem, Javou a mezi C#. Flash většinou zaujme především svou jednoduchostí a možností rychle vytvořit pěknou grafickou animaci. Ovšem práce s tímto jazykem mi přišla dosti odlišná než to, co jsem se během minulých semestrů naučil. Proto jsem tento jazyk vyloučil poměrně brzy.

Java je jeden z nejpoužívanějších a nejpopulárnějších programovacích jazyků na světě. Za vznikem stojí firma Sun Microsystems. Díky své přenositelnosti se používá na velkém množství zařízení. Mezi ně patří mobilní telefony, deskové počítače nebo čipové karty. Nevýhody Javy jsou především v absenci preprocesoru, což znamená, že se aplikace spouští pomaleji, jelikož je třeba aplikaci vždy přeložit. Další nevýhodou je paměťová náročnost při jednodušších aplikacích.

Jazyk C# vznikl jako reakce společnosti Microsoft na jazyk Java. Je třeba říct, že se na internetu objevují články, které tvrdí, že Microsoft použil spoustu myšlenek právě z Javy. Použití pro formulářové aplikace ve Windows, databázové či webové aplikace. Jeho syntax je většinou přebraná z jazyka C. Jako vývojové prostředí se nejčastěji používá Microsoft Visual Studio.

Pro tuto bakalářskou práci jsem zvolil jazyk C#. Hlavním důvodem byla znalost jazyka C jako takového. Navíc jsem byl seznámen se základy tohoto programovacího jazyku v předmětu Tvorba uživatelského rozhraní v pátém semestru studia. Jako programovací prostředí jsem si vybral Microsoft Visual Studio. Především pak pro jednoduchost a intuitivní jednání při ladění či samotném psaní kódu.

5.2 Implementace

Implementace vycházela z návrhu popsaného na začátku této kapitoly. Nejprve byly rozmístěny ovládací prvky. Mezi ně patří tlačítko pro přepínání výukové a demonstrační části, prvky ovládající AVL strom, průchod stromem a posuvník pro změnu rychlosti animace.

Uzly a metody nad nimi jsou definovány ve třídě Uzel. Tato třída zajišťuje funkce samotného stromu. Obsahuje metody pro vložení a mazání uzlu, pomocné funkce pro tyto operace, dále pak jednotlivé rotace, metodu pro vykreslení uzlu, přepočtení pozice jednotlivých uzlů, přičítání bodu při vodorovném pohybu uzlu, zjišťování, zda byl vybrán uzel, a metody pro průchod stromem.

Metoda vykreslení nejprve vykresluje na plátno úsečky, které demonstrují propojení jednotlivých uzlů, a následně se zanořuje, dokud nenarazí na prázdný ukazatel. Při výstupu z rekurze se vykreslují jednotlivé uzly. Uzel je tvořen podkladem, který zastupuje obrázek, a textem, který zobrazuje hodnotu klíče daného uzlu. Text je centrován vodorovně i svisle do středu obrázku.

AVL strom je binární strom. Toto je třeba brát v potaz při vytváření algoritmu pro výpočet vodorovné vzdálenosti mezi dvěma uzly. Nejjednodušeji se to dá realizovat vzorcem, kde se vzdálenost zvětšuje exponenciálně podle výšky podstromu. Tvar vzorce pro výpočet uzlu levého podstromu:

$$X = X1 - 20 * 2^{\text{výška podstromu}}$$

X je X-ová souřadnice daného uzlu a X1 otce daného uzlu. Tento vzorec používá konstantu 20 vzhledem k šířce uzlu. Pro pravý podstrom se pouze změní znaménko ze záporného na kladné.

Druhou třídou je třída form. V této třídě jsou spravovány a ošetřovány především zásahy uživatelů. Je třeba definovat funkce při stisku tlačítek, kliknutí do oblasti vykreslování a podobně. Prvním tlačítkem je tlačítko pro změnu výukové a demonstrační části aplikace. Toto tlačítko pouze skryje, respektive zobrazí ovládací prvky, které jsou potřebné v jedné či druhé části.

Textové pole je pouze dvouciferné, lze zadávat pouze hodnoty 0-99. Nelze zadat jiné znaky než číslice 0-9.

Při stisku tlačítka Vložit nebo Smazat se ověří zda není Textové pole prázdné. Pokud ano, vloží se nebo smaže uzel s náhodně vygenerovanou hodnotou z intervalu 0-99. Nastaví se také výstupní text, který se zobrazí na stavovém řádku.

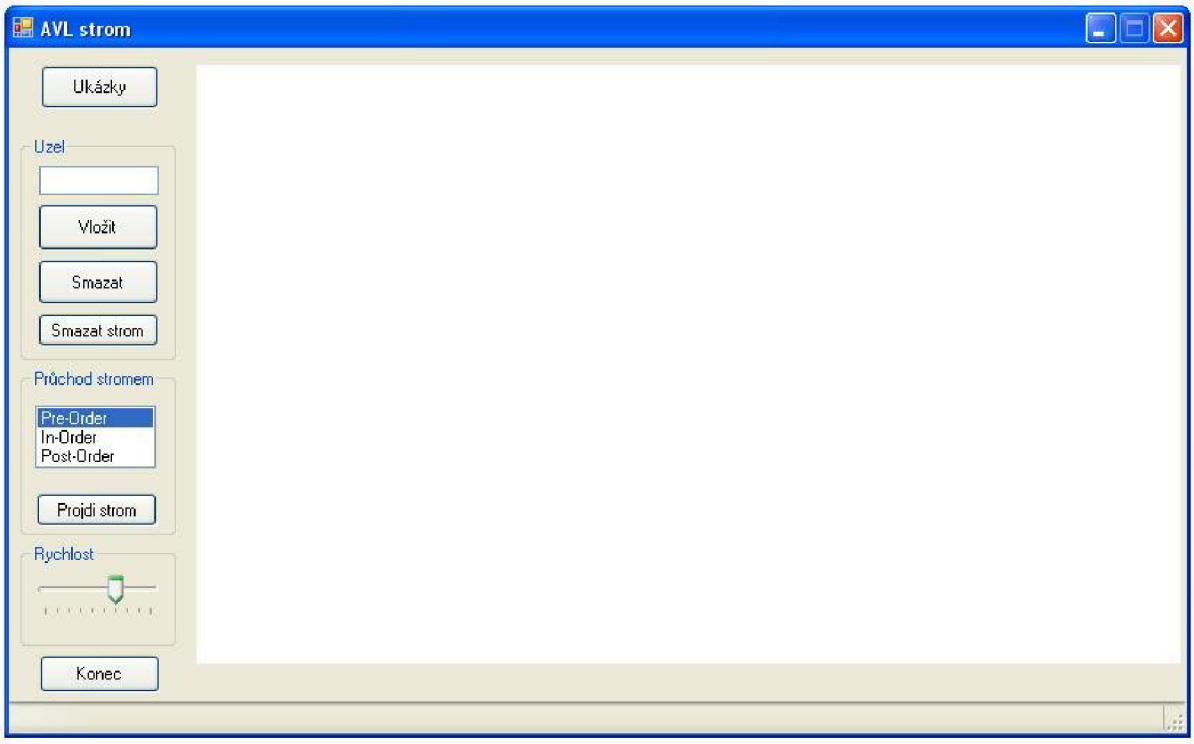
Při kliknutí myší do vykreslované oblasti dochází k zavolání metody třídy Uzel, která zjistí, zda jsme vybrali některý z uzlů. Pokud ano, tento uzel se vykreslí s jiným barevným podkladem, a tím pádem se jeví jako vybraný. Při kliknutí kdekoliv mimo stromovou strukturu se všechny uzly odznačí.

Animace je vytvářena přesunem animovaného objektu o jeden pixel. Toto je využito u rotací, kde je třeba rotovat zúčastněné uzly a jejich podstromy. Pokud dojde k situaci, kdy je třeba rotovat část stromu, je aktivován čítač, který je aktivní, dokud všechny uzly nejsou na pozici, na které mají

být. Během čítání je pro každý uzel, který uchovává jeho současnou pozici, volána funkce „presun“, ve které se uzel posune o jeden pixel směrem k cílové pozici. Ta je uchovávána v proměnné určené pouze pro tento účel.

Pokud dosáhne strom výšky 5, zobrazí se vodorovný posuvník, který umožní zobrazení stromu s větším počtem prvků.

5.3 Uživatelské rozhraní



Obr. 5.3 Úvodní obrazovka aplikace

Rozhraní je navrženo tak¨, aby byla co možná největší plocha pro vykreslovaní AVL stromu. V levé části nalezneme všechna potřebná tlačítka pro manipulaci s AVL stromem. Zobrazení, které vidíme na obrázku 5.3, zobrazuje demonstrační část aplikace. Výuková část je obdobná, proto není nutné ji zde také ukazovat.

Uživatel má možnost zadávat hodnotu uzlu pomocí textového pole. Pokud v tomto poli není hodnota zadána, aplikace přidělí novému či mazanému uzlu náhodnou hodnotu klíče.

Strom je vykreslován na střed plátna. X-ová souřadnice bodu synovského uzlu je určena jeho podstromem, jak je uvedeno výše. Y-ová souřadnice je tvořena pouze součtem otcovské souřadnice a konstanty.

Stavový řádek zobrazuje právě provedenou operaci. Při vložení nebo mazání se nám zobrazí hodnota prvku daného uzlu. Pokud vkládaný uzel již existuje nebo mazaný uzel neexistuje, pak se

nám vypíše tato informace také. Při průchodu stromem se nám postupně označují uzly a na stavovém řádku se vypisují jejich hodnoty tak, jak je postupně procházíme. Při změně rychlosti se na stavovém řádku zobrazuje aktuální nastavená rychlost.

6 Závěr

Úkolem této bakalářské práce bylo přiblížení studijních materiálů studentům předmětu Algoritmy více zaměřených na jazyk C. Oporu jsem přepsal podle potřeby a v ní i všechny příklady kódu. Převedl jsem všechny operace v nerekurzivním zápise nad AVL stromu. Vytvořil jsem rekurzivní obdobu těchto operací. Všechny funkce jsou plně funkční a odladěny na školním serveru Merlin.

Vytvořil jsem aplikaci pro demonstrační animaci operací nad AVL stromem. Aplikace je psána v jazyce C# a poskytuje ucelený náhled na strom jako takový i na jednotlivé rotace. Pomocí této animace by měl být každý student schopen pochopit problematiku AVL stromu během několika minut. Pro spuštění aplikace je třeba mít nainstalovaný .NET framework 2.0, který je součástí novějších aktualizací operačního systému Microsoft Windows.

Práce na této bakalářské práci probíhala již od čtvrtého semestru mého studia. Zpočátku se jednalo pouze o informativní schůzky s profesorem Honzíkem, následoval přepis opory a nerekurzivního zápisu operací v následujícím semestru. Práce v tomto semestru zahrnovala tvorbu rekurzivního zápisu operací a tvorbu aplikace. V průběhu celé práce probíhaly konzultace s profesorem Honzíkem.

Rozšíření této bakalářské práce bych viděl ve vytvoření uceleného výukového programu pro demonstraci všech metod a algoritmů vyučovaných v předmětu Algoritmy. Tyto demonstrace by také mohly zobrazovat právě prováděný kód, a to jak v jazyce Pascal, tak v jazyce C. Popřípadě by tato aplikace mohla být multi-jazyková.

Zadání obsahuje také tvorbu kontrolních otázek a otázek ke zkoušce. Tyto body zůstaly nesplněny po konzultaci s vedoucím bakalářské práce. AVL strom není probírán v předmětu Algoritmy tak důkladně, aby byl předmětem zkoušení.

Literatura

[1] Honzík J.M.: Algoritmy. Studijní opora pro předmět Algoritmy, Elektronický text, FIT VUT v Brně

[2] Honzík J.M. a kolektiv: Vybrané kapitoly z programovacích technik. Skriptum VUT

[3] Wikipedie, free encyclopedia [online]. AVL-TREE 2007. [cit. 2.5.2007]. Dostupný z WWW: <http://en.wikipedia.org>.

[4] Wikipedie, Otevřená encyklopedie [online] Dostupný z WWW: <http://cs.wikipedia.org>

Seznam příloh

Příloha 1. CD