VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

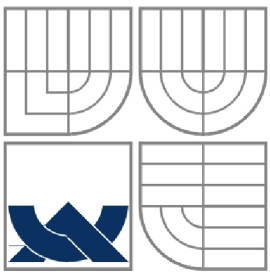
VÝUKOVÝ PROGRAM PRO DEMOSTRACI PRINCIPŮ INDEXOVÁNÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

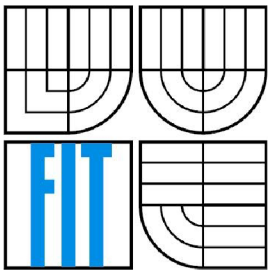
AUTOR PRÁCE
AUTHOR

JÁN ŠKURLA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VÝUKOVÝ PROGRAM PRO DEMOSTRACI PRINCIPŮ INDEXOVÁNÍ

EDUCATIONAL PROGRAM FOR DEMONSTRATION OF INDEXING PRINCIPLES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JÁN ŠKURLA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BÁRTÍK, Ph.D.

BRNO 2009

Zadání bakalářské práce/8271/2008/xskurl00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2008/2009

Zadání bakalářské práce

Řešitel: **Škurla Ján**
Obor: Informační technologie
Téma: **Výukový program pro demostraci principů indexování**
Kategorie: Databáze

Pokyny:

1. Prostudujte principy indexování v relačních databázích.
2. Seznamte se s principy tvorby appletů v jazyce Java.
3. Navrhněte výukovou aplikaci, která bude demonstrovat princip indexovacích metod, zaměřte se především na B+ strom.
4. Navrženou aplikaci implementujte.
5. Zhodnoťte dosažené výsledky a další možnosti pokračování v tomto projektu.

Literatura:

- Silberschatz, A., Korth H.F, Sudarshan, S.: Database System Concepts. Fourth Edition. McGRAW-HILL. 2001, 1088 p.
- Herout, P.: Učebnice jazyka Java. Kopp, 2001, 381 p.

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků.

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: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2008
Datum odevzdání: 20. května 2009

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tématem této bakalářské práce je demonstrace jedné z metod indexování, a to B+ stromu pro předmět databázové systémy. Cílem bylo vytvoření výukového appletu, kterí má sloužit jak názorná ukázka vnitřní struktury B+ stromu s popisem jednotlivých operací: vkládání, vyhledávávání a mazání.

.

Abstract

The topic of this bachelor thesis is to demonstrate B+ Tree indexing method for the course Database Systems. The goal was to create education applet, which will demonstrate internal structure of B+ Tree with description of operations inserting, searching and deleting.

Klíčová slova

B+ strom, výuková aplikace, Java, applet, swing

Keywords

B+ tree, educational application , Java, applet, swing

Citace

Ján Škurla: Výukový program pro demostraci principů indexování, bakalářská práce, Brno, FIT VUT v Brně, 2009

Výukový program pro demostraci principů indexování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval sám pod vedením pána Ing. Vladimíra Bártíka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ján Škurla
18.5.2009

Poděkování

Chtěl bych poděkovat vedoucímu své bakalářské práce, panu Ing. Vladimíru Bártíkovi, Ph.D., za odbornou pomoc, připomínky a trpělivost při tvorbě tohoto projektu.

Obsah

1 Úvod

Všetky informácie nachádzajúce sa na internete sú uložené vo forme dát, ktorých počet a veľkosť sa neustále zvyšuje. Pokiaľ v dátach chceme efektívne vyhľadávať, je nevyhnutné mať tieto dáta uložené v databázach. V nich sú informácie indexované, to znamená, že im je priradený kľúč a nakoniec sú uložené do tabuliek. V súčastnosti sú tieto kľúče zväčša usporiadané vo forme n-nárneho binárneho stromu, konkrétne B+ stromu. Aby sme mohli presne pochopiť ako pracuje tento strom a pritom sa nemuseli až tak podrobne zaoberať teóriou, bolo by najvhodnejšie použiť výukový program.

Cieľom mojej práce je vytvoriť aplikáciu, ktorá bude jednoducho reprezentovať vnútornú štruktúru B+ stromu. Počas každej operácie bude zobrazovať a vysvetľovať jej zmenu. Program by mal slúžiť ako učebná pomôcka, na ktorej študenti názorne uvidia fungovanie B+ stromov.

Jadro tejto práce je rozvrhnuté do šiestich kapitol. Začínam vysvetlením dôležitých pojmov a logickým prepojením medzi nimi. Tieto pojmy sú základom, z ktorého sme vychádzali pri tvorbe tejto práce. Nasleduje stručný popis programovacieho jazyka Java s vyzdvihnutím jeho výhod, čo sa týka tvorby appletov v ňom. Ďalej v kapitole Analýza a návrh aplikácie sa snažím zhrnúť požiadavky na aplikáciu, jej obmedzenia a štruktúru. Navyše obsahuje predpokladaný diagram užitia výsledného programu. V kapitole päť opisujem implementáciu. Hlavne zjednodušený postup vytvárania tried a implementačné prostredie. Ďalej spomínam vzniknuté problémy pri tvorbe tohto programu a postup ich riešenia. Konečne sa dostávame ku konkrétnym príkladom vkladania, vyhľadávania a mazania v B+ strome, kde každé je doložené obrázkom pred a po operácii.

V závere tejto práce hodnotím poznatky z tvorby a testovania tejto výukovej aplikácie a taktiež ponúkam námety na budúce rozšírenia a optimalizáciu práce.

2 Teoretická časť

2.1 Databáza a databázové modely

Databáza[1] je usporiadaná zbierka *dát* alebo *informácií* s opisom jednej, alebo viacerých organizácií. Skladá sa z objektov a vzťahov medzi nimi. Napr. za organizáciu môžeme považovať vysokú školu, pričom objektami sú v tomto prípade študenti, profesori, učebne, fakulta atď. a vzťahmi sú: obsadenosť učebne študentmi, zápis študentov do kurzov atď. .

V každej databáze naviac existuje popis štruktúry dát a typu dát, ktoré sa v nej nachádzajú. Spolu s objektmi vytvárajú *logické schéma*. Toto schéma presne popisuje objekty, ktoré sa v databáze nachádzajú a ich vzájomné prepojenie. Existuje viacero možností tvorby logických schém, ktoré sa nazývajú *databázovými modelmi*. Podľa spôsobu ukladania dát a väzieb medzi nimi môžu byť databázy rozdelené na tieto základné databázové modely :

- Objektová relačná databáza
- Objektová databáza
- Relačná databáza
- Hierarchická databáza
- Sieťová databáza

2.1.1 Relačná databáza

Relačná databáza je typ databázového modelu, ktorý je založený na relačnom modele dát a relačnej algebre. Technológia relačných databáz bola pôvodne navrhnutá Edgarom F. Coddenom v roku 1970[6]. Pre jednoduchšie pochopenie modelu môžeme povedať, že dáta sú v relačnej databáze štruktúrované do tabuliek s usporiadanými stĺpcami. Pojem tabuľka však neodpovedá exaktnému matematickému vyjadrovaniu. Preto sa ďalej bude používať pojem relácia.

Definícia relácie [1]:

Nech $D_1, D_2, \ldots, D_n$ sú množiny atomických hodnôt označované ako *domény*. *Relácia (databázová)* na doménach $D_1, D_2, \ldots, D_n$ je dvojica $\mathbf{R} = (R, R^*)$, kde $R = R(A_1{:}D_1, A_2{:}D_2, \ldots A_n{:}D_n)$ je *schéma relácie, kde* A_i ($A_i \neq A_j$ pro $i \neq j$) značí meno atribútu definovaného na doméne D_i, a $R^* \subseteq D_1 \times D_2 \times \ldots \times Dn$ je telo *relácie*. Počet atribútov n relácie sa označuje *stupeň (rad) relácie*, kardinalita tela relácie $m = |R^*|$ sa označuje *kardinalita relácie*.

Stĺpce relačnej databázy sa nazývajú *atribúty* alebo pole, riadky sú potom *záznamy*. Atribúty môžu mať hodnoty z niektorej množiny prípustných hodnôt. Tieto množiny sa nazývajú *domény*. Nie všetky tieto hodnoty sa ale musia v tabuľke vyskytovať. Množina hodnôt atribútu, ktoré sa v danom okamžiku v tabuľke vyskytujú sa označuje ako *aktívna doména.* Riadok je rezom cez stĺpce tabuľky a slúži k vlastnému uloženiu dát. Počet riadkov tabuľky i ich obsah sa v čase mení.

---

1 Pri spracovaní tejto témy bolo čerpané z týchto materiálov: [1], [2] a [4]

Hlavné typy kardinality vzťahu medzi tabuľkami:

- V tabuľkách nie sú súvisiace údaje, a preto medzi nimi nie je žiadny vzťah.
- Relácia typu 1:1 znamená,že práve jednému záznamu v prvej tabuľke odpovedá práve jeden záznam v druhej tabuľke. Jednotlivé záznamy v obidvoch tabuľkách sú priamo prepojené. Tento typ relácie sa používa veľmi sporadicky, pretože takto spojené údaje idú umiestniť iba do jednej tabuľky. Relácia 1:1 má význam hlavne u rozľahlých tabuliek kde slúžia pre sprehľadnenie.
- Relácia typu 1:N je najpoužívanejším typom. Umožňuje,aby jednému záznamu v prvej tabuľke odpovedalo viacero záznamov v druhej tabuľke. Znamená to, že v prvej tabuľke sa nachádza jeden záznam a v druhej sa k nemu nachádza 0 až N záznamov.
- Relácia typu N:N umožňuje, aby niekoľkým záznamom v prvej tabuľke odpovedalo niekoľko záznamov v druhej tabuľke. U tohoto typu relácie, je pre jej vytvorenie potrebná dekompozícia vzťahov, alebo ich rozdelenie vytvorením spojovacej tabuľky. Vytvoríme pomocnú tabuľku spojením dvoch primárnych kľúčov, kde potom definujeme reláciu typu 1:N.

Primárny kľúč

Primárny kľuč je jednoznačný identifikátor každého záznamu v tabuľke. Primárnym kľúčom môže byť jediný stĺpec, či kombinácia viacerých stĺpcov tak, aby bola zaručená jeho jednoznačnosť. Kľúč musí obsahovať hodnotu, tzn. nesmie sa tu vyskytovať nedefinovaná prázdná hodnota NULL. Typickým príkladom primárneho kľúča je rodné číslo v tabuľke so zoznamom osôb.

Pokiaľ neexistuje žiadny prirodzený kľúč, používajú sa umelé kľúče, čo sú číselné alebo písomné identifikátory. Každý nový záznam dostáva identifikátor odlišný od identifikátora všetkých predchádzajúcich záznamov (požiadavok na unikátny kľúč), obyčajne sa jedná o celo číselné rady a každý nový záznam dostáva číslo vždy o jednotku vyššie ako je číslo u posledného vloženého záznamu.

2.2 Index

Štruktúra[2] pre zrýchlenie prístupov do riadkov tabuľky, založenej na hodnotách jedného alebo viacerých stĺpcov. Prítomnosť indexu výrazne zlepšuje efektivitu dotazov nad tabuľkami. Indexy sú zväčša vytvorené, aby spĺňali špecifické vyhľadávacie kritéria postupne sa zväčšujúcich databáz. Umožňujú efektívne získavať všetky záznamy v tabuľke, ktoré spĺňajú podmienky vyhľadávania. Index je tvorený záznamami, ktoré nazývame *položky indexu.* Na uchovávanie indexov sa najčastejšie používa *sekvenčný súbor* a *stromy.*

Pre lepšie pochopenie sa index v databázach prirovnáva k registru v knihye. Register takisto slúži na rýchlejšie nájdenie miesta, kde sa vyskytuje nejaký pojem. Ak chceme tento pojem nájsť musíme sa pozrieť do registra, kde zistíme, či sa daný pojem nachádza v knihe a pokiaľ áno, tak na akom mieste ho môžeme nájsť. Pre jednoduché hľadanie býva abecedne usporiadaný. Podobným spôsobom pracuje aj usporiadaný index.

Súbor zo záznamami tabuľky môžme označovať ako primárny *súbor* a stĺpec, podľa ktorého záznamy sprístupňujeme budeme nazývať vyhľadávajúci *kľúč*. Každý index obsahuje hodnotu

2 Pri spracovaní tejto témy bolo čerpané z týchto materiálov: [1], [2] , [4] a [5]

vyhľadávajúceho kľúča. Tento kľúč slúži pre rýchle pristupovanie do stĺpcov tabuľky vybraného súboru.

Primárny index
Primárny index tvoríme nad usporiadaným súborom unikátnych (unique) záznamov pevnej dĺžky. Súbor môže mať len jeden primárny index (označovaný ako hlavný index). Taktiež sa nazývajú zhlukovacie *(clustering) indexy*, pretože odkazujú na zhlukované súbory.

Sekundárny index
Sekundárne indexy poskytujú podstatne väčšie zrýchlenie prístupu priamo do súboru než primárne indexy. Poskytujú logické usporiadanie záznamov. Sekundárnych kľúčov môže byť niekoľko (i pre všetky polia záznamu). Sekundárny index môže byť vytvorený nad kľúčovým poľom (sekundárny kľúč), čo umožní binárne vyhľadávanie.

Hustý index
Obsahuje položku pre každú hodnotu vyhľadávacieho kľúča, ktorá sa v primárnom súbore vyskytuje. V prípade hustého primárneho indexu obsahuje položka indexu hodnotu vyhľadávacieho kľúča a ukazovateľ na prvý záznam s danou hodnotou vyhľadávajúceho kľúča. V prípade sekundárneho indexu musí položka indexu obsahovať ukazovatele na všetky záznamy s rovnakou hodnotou, keďže sa môžu nachádzať kdekoľvek v primárnom súbore.

Riedky index
Obsahuje iba niektoré hodnoty vyhľadávacieho kľúča. Tento typ ide použiť iba pri primárnom indexe, keďže ide využiť usporiadanie záznamov v primárnom súbore. Hodnoty vyhľadávacieho kľúča, ktoré sa nachádzajú v položkách indexu, sú zväčša hodnoty prvých záznamov v diskových blokoch, takže ukazovateľ potom ukazuje na blok, ktorý sa má prehľadať. U riedkeho indexu sa kombinuje priamy prístup so sekvenčným prehľadávaním časti súboru. Preto je tento typ indexu použiteľný iba u primárneho indexu. Sekundárny index musí byť vždy hustý, keďže primárny súbor môže byť usporiadaný iba jedným spôsobom a to podľa hodnôt primárneho vyhľadávacieho kľúča.

Viacúrovňové indexy
Využíva sa rovnaký prístup ako v prípade primárneho súboru a môže sa vytvoriť index indexu, čiže viacúrovňový index. Pod týmto pojmom budeme rozumieť index, ktorý má stromovú štruktúru, pričom každá k-tá úroveň ($1 \leq k \leq N$ - 1, kde N je počet úrovní indexu) je *jednoúrovňovým indexom úrovne $k + 1$*. Príkladom štruktúry ktorá využíva viacúrovňový index je *ISAM* alebo *B+ strom*.

2.3 B+ strom

2.3.1 Základné fakty o B+ strome

B+ strom[3] je dynamická stromová štruktúra umožňujúca rýchle vkladanie, mazanie a vyhľadávanie dát, s čo možno najmenšou réžiou. Považujeme ho za dynamickú, pretože v uzloch je ponechaný

3 Téma bola spracovaná z týchto materiálov: [1],[7] ,[8] a [9]

priestor pre ďalší rast tým, že každý uzol s výnimkou vrcholu je zaplnený minimálne z 50%. Patrí spolu s ISAM[4] indexom k najpoužívanejším databázovým indexovým štruktúram.

B+ strom je viacúrovňový index v tvare n-nárneho vyváženého stromu, ktorý možno charakterizovať nasledujúcimi dôležitými vlastnosťami[1]:

- Každý uzol s výnimkou koreňa a listu má $\lceil (n/2) \rceil$ [5] až n následníkov.
- Koreň, pokiaľ nie je súčasne listom, má nejemnej 2 následníkov.
- List obsahuje $\lceil (n-1)/2 \rceil$ až n-1 hodnôt vyhľadávacieho kľúča.
- Listové uzly tvoria jednosmerný usporiadaný zoznam.
- Pre K hodnôt vyhľadávacieho kľúča nie je cesta od koreňa k listu dlhšia než $\lceil \log_{\lceil n/2 \rceil} (K) \rceil$

Ďalšie vlastnosti B+ stromu:

- Všetky listy sú na rovnakej úrovní (v rovnakej hĺbke).
- Operácie (vloženie, mazanie) udržujú strom v rovnováhe.
- Hodnoty vyhľadávacieho kľúča sú v indexe zoradené vzostupne alebo zostupne.

Tvar uzla B+ stromu

Predpokladajme vzostupné usporiadanie hodnôt vyhľadávacieho kľúča v uzle. Pri úplnom zaplnení je v uzle n ukazovateľov na následníkov (značených symbolom P) a n-1 hodnôt vyhľadávacieho kľúča.

P_1	V_1	P_2	V_2	...	P_{n-1}	V_{n-1}	P_n

Obrázok 2.1 Typický tvar uzla B+ stromu.

Nelistový uzol

Pokiaľ nie je uzol listom stromu, ukazuje ukazovateľ P_1 na *vrchol podstromu*, v ktorom sú uložené hodnoty vyhľadávacieho kľúča $h < V_1$. Keď je v uzle k použitých ukazovateľov ($k \leq n$), potom ukazovateľ P_k ukazuje na vrchol podstromu, v ktorom sú uložené hodnoty vyhľadávacieho kľúča $h \geq V_1$. Pre ostatné ukazovatele P_x ($2 \leq x \leq k-1$) v uzle platí, že ukazujú na vrchol podstromu, v ktorom sú hodnoty vyhľadávacieho kľúča h také, že $V_{x-1} \leq h < V_x$.

Listový uzol

V uzle je použitých k ukazovateľov ($k \leq n$). V prípade listového uzla ukazujú ukazovatelia P_x ($1 \leq x < k-1$) na záznam primárneho súboru alebo na tzv. sektor ukazovateľov v prípade, že index nie je unikátny. Naviac je ukazovateľ P_k použitý k vytvoreneniu *jednosmerne viazaného zoznamu*.

B+ Strom sa skladá z dvoch častí – sekvenčnej a indexovej.

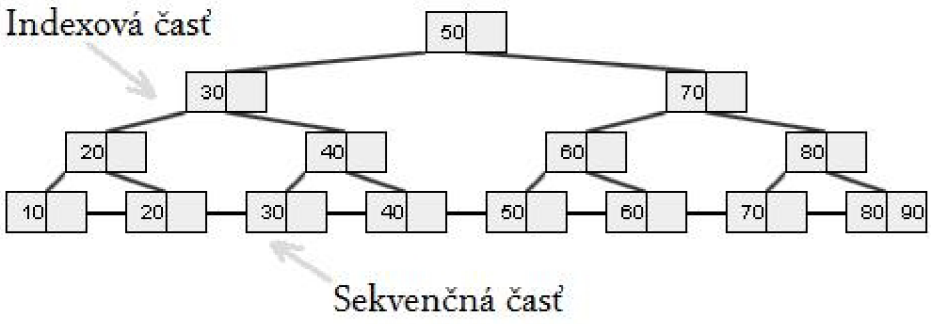
Sekvenčná časť stromu je tvorená listovými uzlami. Keď má mať B+ strom hustý index, je sekvenčná časť hustým indexom pre daný vyhľadávací kľúč. Uzly sekvenčnej časti tvoria jednosmerne viazaný

4 Metóda ukladania dát s možnosťou rýchleho vyhľadávania dát pomocou indexu.

5 Výraz $\lceil x \rceil$ značí najmenšie celé číslo väčšie alebo rovné x, tj. zaokrúhľovanie nahor na celé číslo

zoznam, ktorý sa využíva pri sekvenčnom spracovávaní záznamov. Toto sa využíva v intervalových dotazoch. Slúžia na to posledné ukazovatele blokov. Každý ďalší použitý ukazovateľ P_x ($1 \leq x \leq k-1$), kde $k \leq n-1$ je počet hodnôt vyhľadávacieho kľúča v uzle, ukazuje na záznam v primárnom súbore v prípade, že index je unikátny. Minimálne zaplnenie listových blokov je $\lceil (n-1)/2 \rceil$ hodnôt vyhľadávacieho kľúča a teda i ukazovateľov na záznamy tabuľky, resp. sektory ukazovateľov. Maximálne zaplnenie je (n-1) hodnôt vyhľadávacieho kľúča.

Indexová časť B+ stromu tvorí riedky viacúrovňový index sekvenčnej časti. Minimálne zaplnenie uzlov okrem koreňa je $\lceil n/2 \rceil$ ukazovateľov, a teda $\lceil n/2 \rceil - 1$ hodnôt vyhľadávacieho kľúča. Koreň môže mať minimálne 2 ukazovateľov a jednu hodnotu vyhľadávacieho kľúča. Maximálne zaplnenie je n ukazovateľov a n - 1 hodnôt vyhľadávacieho kľúča.



Obrázok 2.2 Sekvenčná a indexová časť stromu.

2.3.2 Základné operácie nad B+ stromom

Pre jednoduchosť budeme používať vzostupné usporiadanie hodnôt vyhľadávacieho kľúča, hustý a unikátny index[6]. Pre zjednodušenie budeme hodnotu vyhľadávacieho kľúča nazývať skrátene „kľuč“.

Vkladanie

Pri vkladaní prvého kľúča je strom prázdny a bude obsahovať jediný uzol, ktorý bude zároveň koreňom a aj listom. Splýva teda sekvenčná a indexová časť stromu. Predpokladajme, že budú vkladané odpovedajúce unikátne kľúče. Pri vkladaní sa kontroluje, či sa rovnaká hodnota kľúča v strome už nenachádza, pokiaľ áno dochádza k chybe. Vkladanie je zatiaľ jednoduché a výpočtovo nenáročné.

Teraz predpokladajme, že je potreba vložiť do už zaplneného bloku koreňa stromu v sekvenčnej časti stromu ďalšiu položku. V tomto okamžiku je potreba vykonať už zložitejšiu operáciu nazývanú *štepenie uzlov* (*splitting*). Vytvorí sa nový uzol a presunú sa doňho položky tak, že $\lceil (n-1)/2 \rceil$ položiek bude v pôvodnom uzle a zostatok v novom uzle. Vzniknú teda dva uzly sekvenčnej časti indexu s minimálne 50% zaplnením, ktoré sa prepoja do jednosmerne viazaného zoznamu. Keďže prišlo k štiepeniu je potrebné pridať ďalšiu úroveň, nový koreň. Bude tvoriť indexovú časť B+ stromu. Taktiež bude obsahovať ukazovatele na obidva uzly sekvenčnej časti a najnižšiu hodnotu vyhľadávacieho kľúča druhého z blokov.

---

6 Operácie pre zostupné usporiadanie a u indexu pre vyhľadávací kľúč, ktorého hodnoty v tabuľke nemusia byť unikatné, by vyžadovali iba mierne úpravy.

Nová položka indexu je vždy vkladná do niektorého uzla sekvenčnej časti. Operácia je preto prevedená v dvoch krokoch. V prvom analogicky podobnom postupe s operáciou vyhľadávaní sa nájde uzol sekvenčnej časti, do ktorej je potreba vložiť novú položku indexu. V druhom kroku prebieha samotné vloženie položky indexu do nájdeného bloku. Keď je v bloku miesto pre uloženie položky, je tam vložená a operácia končí. V opačnom prípade musí nastať štiepenie uzlu. Vytvorí sa nový uzol, a presunú sa doňho položky tak, že $\lceil (n\text{-}1)/2 \rceil$ položiek bude v pôvodnom uzle a zostatok v novom uzle. Nový uzol sa zapojí do zoznamu uzlov sekvenčnej časti. Tým však vkladanie nekončí. Informácie o vzniku nového uzla v sekvenčnej časti sa musia vložiť do príslušného bloku najnižšej úrovne indexovej časti stromu. Je opäť treba vložiť indexovú položku tvorenú najmenšou hodnotou vyhľadávacieho kľúča v novom uzle a ukazovateľom na uzol.

Po štiepení uzlov pri vkladaní do sekvenčnej časti môže nastať štiepenie aj v indexovej časti stromu. Vytvorí sa nový uzol, presunie sa doňho polovica položiek a ďalšia ostane v pôvodnom uzle ($\lceil (n\text{-}1)/2 \rceil$). Znova musíme vložiť do rodičovského uzla najmenšiu hodnotu kľúča a ukazovateľ na novo vytvorený uzol. Pri vkladaní môže nastať opäť potreba štiepenia uzla. Takým spôsobom sa môže operácia štiepenia uzla šíriť od listov ku koreňu stromu a môže viesť až k zvýšeniu počtu úrovní stromu o jednu, pokiaľ nastane štepenie koreňa.

B+ strom ostáva vždy vyvážený a všetky uzly, okrem koreňa, sú zaplnené minimálne z 50%. Pokiaľ nenastane štiepenie uzlov, je náročnosť operácie vloženia prakticky rovnaká ako operácia vyhľadávania. Iba pri štiepení uzla je náročnosť väčšia. To je dôvod, prečo je v uzloch ponechaný priestor pre ďalší rast.

```
Nájdi listový uzol U, ktorý by mal obsahovať novú položku indexu;
IF je v uzlu U miesto pre vloženie položky THEN
   Vlož položku indexu do uzlu U
ELSE BEGIN /* Rozštiepenie uzlu */
   Alokuj nový uzol U';
   Rozdeľ hodnoty štiepaného uzlu U spolu s vkladanou hodnotou
   hodnoty na dve rovnako veľké skupiny. Jedna zostane
   v uzlu U a druhá sa presune do uzlu U';
   REPEAT
      Rekurzívne pokračuj s vkladaním novej položky
      s prvou hodnotou vyhľadávacieho kľúča uzlu U' a
      ukazovateľom na tento uzol do uzlu predchodca v strome
   UNTIL nedošlo k ďalšiemu štiepeniu alebo bol rozštiepený koreň
END;
IF koreň bol rozštiepený THEN
   Vytvor nový koreň s ukazovateľmi na dva uzly vzniklé
   rozštiepením pôvodného koreňa;
```

Algoritmus 2.1 Vloženia položky do unikátneho hustého indexu v B+ strome

Vyhľadávanie

Vyhľadávanie začína v koreni a rekurzívne pokračuje dole stromom až k sekvenčnej časti stromu, v ktorom by mala byť uložená hodnota vyhľadávacieho kľúča. Prehľadávanie uzla indexovej časti vždy

spočíva v nájdení najmenšej hodnoty vyhľadávacieho kľúča, ktorého hodnota je väčšia ako zadaná, alebo dosiahnutie konca zaplnenej časti uzla. V prvom prípade je následníkom uzol, na ktorý ukazuje ukazovateľ bezprostredne predchádzajúci nájdenej hodnote, v druhom prípade posledný ukazovateľ na uzol. Keďže záznamy uzla sú usporiadané, ide použiť rýchle binárne vyhľadávanie. Týmto spôsobom sa pokračuje pokiaľ sa nedosiahne bloku sekvenčnej časti stromu. Tam sa analogicky prehľadá uzol, ale už iba na rovnosť s hľadanou hodnotou. Keď je nájdená, bezprostredne predchádzajúci ukazovateľ ukazuje na hľadaný záznam primárneho súboru.

```
Nastav koreň ako aktuálny uzol;
WHILE aktuálny uzol nieje listom DO
BEGIN
   Nájdi v uzle najmenší hodnotu vyhľadávacieho kľúča
   Vi, ktorá je väčšia ako hľadaná hodnota;
   IF existuje THEN
      Nastav ako aktuálny uzol, na ktorý ukazuje Pi
   ELSE
      Nastav ako aktuálny uzol, na ktorý ukazuje
      posledný ukazovateľ v aktuálnom uzlu
END;
IF existuje v aktuálnom uzlu hodnota Vi rovná hľadanej
hodnote THEN
   Ukazovateľ Pi ukazuje na hľadaný záznam
ELSE
   Záznam s hľadanou hodnotou neexistuje;
```

Algoritmus 2.2 Vyhľadanie hodnoty v unikátnom hustom indexu v podobe B+ stromu

Rozsahové vyhľadávanie

Jednou z výhod použitia B+ stromu je, že umožňuje efektívny prístup k záznamom tabuľky pomocou *rozsahových dotazov* (*range query*), ktoré pre operácie vyberajú záznamy z určitého intervalu hodnôt. Využívajú toho, že sú uzly v sekvenčnej časti stromu usporiadané do jednosmerného zoznamu. Potom iba stačí, pomocou operácie vyhľadanie nájsť v sekvenčnej časti prvý záznam vyhovujúci dolnému intervalu. Postupne prechádza sekvenčnou časťou stromu, pokiaľ sa nenarazí na prvú hodnotu vyhľadávacieho kľúča, ktorá sa už nenachádza v zadanom intervale, alebo sa nenarazí na koniec zoznamu.

Mazanie

Operácia taktiež ako vkladanie prebieha v dvoch krokoch. Prvým je vyhľadanie položky indexu v sekvenčnej časti stromu. Keď sa nenachádza v strome, nemožno ju vymazať a nastáva chyba. Pokiaľ bola položka indexu nájdená, tak sa zruší. Taktiež aj tu sa musia dodržiavať pravidlá obsadenosti uzla, minimálneho zaplnenia.

Pokiaľ neklesne zaplnenie pod minimálnu prístupnú hranicu, operácia končí. V opačnom prípade sa operácia mazania pokúsi o *zlievanie uzlov* (*coalescence*) so susedmi. Zisťuje, či nemôže položky z málo zaplneného uzla presunúť do susedného. Ak je to možné tak sa presunú, odstráni sa

uzol a zruší sa položka predchodcu v B+ strome, ktorá ukazovala na zrušený uzol. Pokiaľ klesne zaplnenie uzla, kde sa nachádzal predchodca, opäť môže nastať zlievanie alebo redistribúcia. Pokiaľ sa zlievanie šíri celým stromom až ku koreňu, a ten má iba jedného nasledovníka, nastáva zníženie výšky stromu. Novým koreňom sa stáva uzol, do ktorého boli naposledy presunuté položky z mazaného uzla.

Keď sa nenájde sused, ktorý ma dostatok voľného miesta pre zlievanie, musí nastať druhá možnosť a tou je redistribúcia. Jedná sa o presunutie položiek zo susedného uzla do mazaného, aby obidvaja spĺňali podmienku minimálneho zaplnenia. Nezaniká žiadny uzol, len sa mení najmenšia hodnota vyhľadávajúceho kľúča v susednom uzle alebo mazanom. Zmena hodnoty sa musí premietnuť do uzla predchodcu v B+ strome a môže sa šíriť až ku koreňu.

```
Nájdi listový uzol U, ktorý obsahuje rušenú položku indexu;
Zruš položku indexu v uzlu U;
IF kleslo zaplnenie uzlu U pod ⌈(n-1)/2⌉ THEN
   IF ide presunúť obsah do ľavého alebo pravého suseda
      uzlu U THEN
      /* zlievanie uzlov*/
   BEGIN
      Presuň obsah uzlu U a zruš prázdny uzol U;
      REPEAT
         Rekurzívne pokračuj s rušením položky indexu
         s ukazovateľom na uzol U v uzlu predchodcu v strome
      UNTIL nedošlo k ďalšiemu zlievaniu alebo bolo dosiahnuté
         koreňa;
      IF koreň má iba jedného nasledovníka THEN
         Zruš koreň, novým koreňom sa stane následník
   END
   ELSE
   BEGIN /* redistribúcia hodnôt*/
      Redistribuj hodnoty a ukazovatele susedených uzlov
      tak, aby oba splňovali podmienku minimálneho
      zaplnenia;
      Rekurzívne aktualizuj hodnotu vyhľadávacieho kľúča v uzlu
      predchodcu v strome
END;
```

Algoritmus 2.3 Mazanie hodnoty v unikátnom hustom indexu v podobe B+ stromu

3 Java Applet

3.1 Java

Java[7] je programovací jazyk vyvinutý vo firme Sun Microsystems pôvodne pod menom Oak v roku 1991. Pôvodným zámerom Javy malo byť poskytnutie programovacieho jazyka pre spotrebnú elektroniku, aby program bolo možné napísať, skompilovať a spustiť na rôznych platformách bez opätovnej kompilácie. Java je objektovo orientovaná, ktorej syntax je odvodená (trošku upravená a zjednodušená) od jazyka C++. Bola odstránená nielen väčšina konštrukcií, ktoré spôsobovali programátorom problémy, ale aj pribudli niektoré užitočné rozšírenia. Java sa taktiež delí o podobné znaky s jazykmi ako sú Lisp a SmallTalk a tiež so skriptovacími jazykmi ako Perl a Tcl.

Základné vlastnosti JAVY:

- **jednoduchá** – hlavná výhoda oproti ostatným programovacím jazykom je v jednoduchosti. Umožňuje programovať bez predchádzajúceho profesionálneho tréningu, za pomoci bežných programovacích praktík pri tvorbe softvéru. Programátori majú dostupný veľký počet knižníc na tvorbu objektov. Od základný typov dát cez I/O objekty, sieťové rozhranie až po nástroje pre tvorbu grafického používateľského rozhrania. Tieto knižnice môžu byť rozširované, aby poskytli nové funkčnosti.
- **objektovo orientovaná** – je navrhnutá tak, aby bola od základu objektovo orientovaná. Ide o techniku programovania, pri ktorej sa zameriava na *objekty* (*triedy*) a *interface* k nim. Všetky dátové typy sú objektové, okrem ôsmych základých dátových typov.
- **interpretovaná** – na rozdiel od C++ sa nevytvára strojový kód na špecifickú platformu, ale na nezávislý *medzikód* (*byte code*). Vytvorený medzikód je nezávislý na architektúre zariadenia. Java interpreter môže tento medzikód interpretovať priamo na zariadení pre ktoré je určený. V súčasnosti sa na urýchlenie používajú technológie, ktoré byte code najprv interpretujú a na základe analýzy štatistik získaných z tejto interpretácie vykonajú preklad často používaných častí do strojového kódu.
- **distribuovaná** – je navrhnutá pre podporu aplikácií v sieti (obsahuje obrovskú knižnicu rutín zameraných na TCP/IP protokoly). Pracuje so vzdialenými súbormi pomocou URL, taktiež má prístup k objektom na lokálnom súborovom systéme.
- **robustná** – je určená pre tvorbu vysoko spoľahlivého softvéru. Poskytuje skoré kontrolovanie problémových situácií, po ktorom nasleduje dynamické (*run time*) kontrolovanie. Neumožňuje používať niektoré programátorské konštrukcie, ktoré sú častou príčinou chýb (napr. `GOTO`, odkazy ...). Obsahuje *Garbage collector*, ktorý slúži na automatickú správu pamäti. Automaticky vyhľadáva už nepoužívané časti pamäti a uvoľňuje ich pre ďalšie použitie. Tento jednoduchý model správy pamäte odstraňuje všetky druhy programátorských chýb.

7 Pri spracovaní tejto témy bolo čerpané z týchto materiálov:[10] a [11]

- **bezpečná** – je vytvorená pre sieťové prostredie, kde je bezpečnosť nesmierne dôležitá. Java umožňuje konštruovať aplikácie, ktoré nemôžu byť infikované zvonku. Má aj vlastnosti, na celkovú ochranu počítača, kde je program vykonávaný, pred nebezpečnými operáciami nepriateľského kódu.
- **platformovo nezávislá** – na každom počítači s *Java Virtual Machine* (JVM) môže bežať akýkoľvek Java program, bol kdekoľvek napísaný. Nezávislosť jazyka funguje, preto lebo zdrojové programy sa nekompilujú do špecifického strojového kódu, ale na nezávislý byte code. Tento byte code je interpretovaný pomocou JVM na zvolenej platforme. Podľa konkrétnej platformy sa môže prispôsobiť vzhľad a chovanie aplikácie.
- **viacvláknová** – podporuje spracovanie viacvláknových aplikácií na úrovni programovacieho jazyku s pridanými sofistikovanými synchronizačnými metódami,čo umožňuje vykonávať viac vecí naraz. Knižnice poskytujú *Thread*[8] triedy, a run-time systém umožňuje sledovať stav a upravovať zamykacie primitívy. Systémové knižnice boli napísané, tak aby vlákno bolo zabezpečené a používalo funkcie, ktoré nespôsobujú konflikty viacerých paralelných vlákien.

3.2 Applet

Java *applety* sú jednoduché programy, ktoré sú užívateľom distribuované v medzikóde. Býva väčšinou orientovaný na vykonávanie konkrétnej funkcie a nepredpokladá sa, že bude používaný ako samostatná aplikácia. Sú zväčša vytvorené v Jave, ale nemusí to byť vždy pravda. Môžu byť naprogramované aj v iných jazykoch, ktoré tak isto vytvárajú medzikód. Nie sú samostatne spustiteľné, ale musia byť interpretované v aplikácii pomocou Java Virtual Machine. Takýto nástroj v podobe pluginu používa väčšina webových prehliadačov. Ďalším obmedzením je, že je *bezstavový* a kvôli bezpečnosti disponuje len obmedzenými prístupovými právami. Pri načítaní internetovej stránky, kde je zakomponovaný applet sa najprv zobrazí šedá plocha. Potom sa pomaly do prehliadača nahráva celý applet. Po načítaní sa nakoniec objaví na mieste šedej plochy applet.

Podľa použitej technológie môže byť applety vytvorené pomocou rozličných technológií. V *AWT* sa používa triedu `Java.awt.Applet` definujúca applet, v *Swingu* je touto triedou `Javax.swing.JApplet`. Tieto triedy poskytujú všetky základné vlastnosti a metódy, ktoré vytvárajú objekt applet.

Životné cykly appletu

Sú spravované prehliadačom pri načítaní appletu z webovej stránky. V zásade existujú štyri metódy v triede Applet, na ktorých je applet postavený a predstavujú jeho životný cyklus.

1)**Init:** Táto metóda je zavolaná po nahratí celého appletu do prehliadača,čiže v tomto okamžiku je už applet aktívny. Preto by ste mali v metóde `init()` mať inicializačný kód.

2)**Start**: Po inicializačnej metóde je automaticky volaný tento spôsob. Od tohoto okamihu, môže užívateľ začať používať applet. Je tiež volaná keď sa kedykoľvek užívateľ vráti na stránky obsahujúce applet po návšteve inej stránky.

---

8 Thread - vlákno vykonávané v programe.

3)**Stop**: Je volaná automaticky, keď užívateľ zmení internetovú stránku. Táto metóda sa môže použiť na pozastavenie animácie. Po návrate sa späť k tomuto appletu browser znova zavolá metódu `start()` a applet je hneď pripravený na používanie.

4)**Destroy**: Volá sa pri normálnom vypnutí prehliadača. Ak sa užívateľ rozhodne zatvoriť browser, tak ten ešte pred ukončením zavolá metódu `destroy()`, aby sa uvoľnili systémové prostriedky, ktoré používal daný applet.

V podstate je možné obísť hociktorú metódu, alebo ju upraviť a vytvoriť si tak vlastnú logiku v cykle appletu. Napríklad v inicializácii apletu nastaviť každému objektu inú veľkosť, zaviesť vlastnú logiku do metódy stop(), ktorá bude vykonávať zálohovacie operácie. Vytvoriť vlastné pravidlá do metódy destroy(). Príkladom môže byť vyčistenie vlákien, ktoré boli spustené v `init()` metóde.

3.3 Applet a HTML

Pre použitie appletu na internetových stránkach musíme dodržať niekoľko základných postupov. Zo zdrojového kódu kompilovanim vytvoríme súbory z príponou `.class`, slúžiacich na spustenie programu. Musíme si vytvoriť novú *HTML* stránku, ktorá zobrazí applet za použitia tagov `<APPLET> </APPLET>`.

Keď internetový prehliadač interpretuje stránku zaslanú web serverom, tak ju začne hneď dekódovať. Ak narazí na `<APPLET>` tag nastávajú dve možnosti. Prvá, browser nepodporuje Javu, tak ho ignoruje. Druhá, požiada server o applet, ktorý sa nachádza na mieste, ktoré je zadefinované v `<APPLET>` tagu.

```
<!-- Hello.html -->
<html>
  <head>
    <title>Hello World Applet</title>
  </head>
  <body>
    <applet code="Hello" width="200" height="200">
    </applet>
  </body>
</html>
```

Príklad 3.1 Vloženie appletu do HTML stránky.

4 Analýza a návrh aplikácie

Pred samotnou implementáciou aplikácie je vhodné si vytvoriť návrh ako celý program bude pracovať a z čoho sa bude skladať. Tiež by sa mali vymedziť niektoré ciele, ktoré chceme splniť, čiže určiť funkčnosť, použitie a v neposlednom rade ovládanie. Všetky tieto časti sú z programátorského hľadiska dôležité pre vytvorenie samostatných tried, ktoré budú medzi sebou v programe komunikovať.

4.1 Analýza

Funkčnosť

Program by mal vedieť vykonávať všetky základné operácie nad B+ stromom. Mal by byť schopný vytvárať uzle a do nich vkladať kľúče, mazať kľúče a pri splnení podmienok rušiť uzle. Jednotlivé uzly budú spojené čiarami reprezentujúcimi jednotlivé hrany. Všetky operácie s uzlami budú animované, taktiež aj vyhľadávanie jednotlivých kľúčov bude animované. Priebeh operácií s uzlami bude komentovaný, aby užívateľ vedel aké pravidlá používa B+ strom pre jednotlivé úkony.

Použitie

Aplikácia bude určená predovšetkým na výučbu B+ stromu profesorom počas prednášky. Predpokladá sa že aplikácia bude umiestnená aj na internete, kde si ju môžu spustiť študenti k upevneniu svojich vedomosti o tvorbe B+ stromov, alebo na prípravu pred skúškou z predmetu Databázové systémy.

Ovládanie

Jeden z cieľov tejto aplikácie je čo najjednoduchšie a intuitívne ovládanie. Preto v návrhu bolo vybrané ako najvhodnejšie ovládanie myšou a vkladanie kľúčov pomocou klávesnice. Pre výukový charakter aplikácie sa zvolené ovládanie javí ako dostatočné.

Návrh aplikácie

Keďže aplikácia je vytvorená v programovacom jazyku Java, bude objektovo orientovaná. Bude obsahovať rôzne typy objektov, ktoré budú medzi sebou komunikovať a zaisťovať potrebnú funkčnosť, nazveme ich triedy. V prehľade vyberám iba tie najdôležitejšie triedy pre chod celej aplikácie.

Triedy (abecedne):

- Hlavny_program - trieda spájajúca uzly, ich 2D polohu a základné operácie nad nimi.
- JPanel_Ciare - trieda slúžiaca na vykreslenie čiary.
- JPanel_Hlavna_plocha - trieda predstavujúca kresliacu plochu, do ktorej vkladáme ďalšie JPanel objekty.
- JPanel_Prvky - trieda slúžiaca na vykreslenie kľúča.
- JPanel_Uzly - trieda slúžiaca na vykreslenie uzla.
- B_plus_applet - hlavná trieda na reprezentáciu aplikácie.
- Prvky - trieda obsahujúca informácie o kľúči.

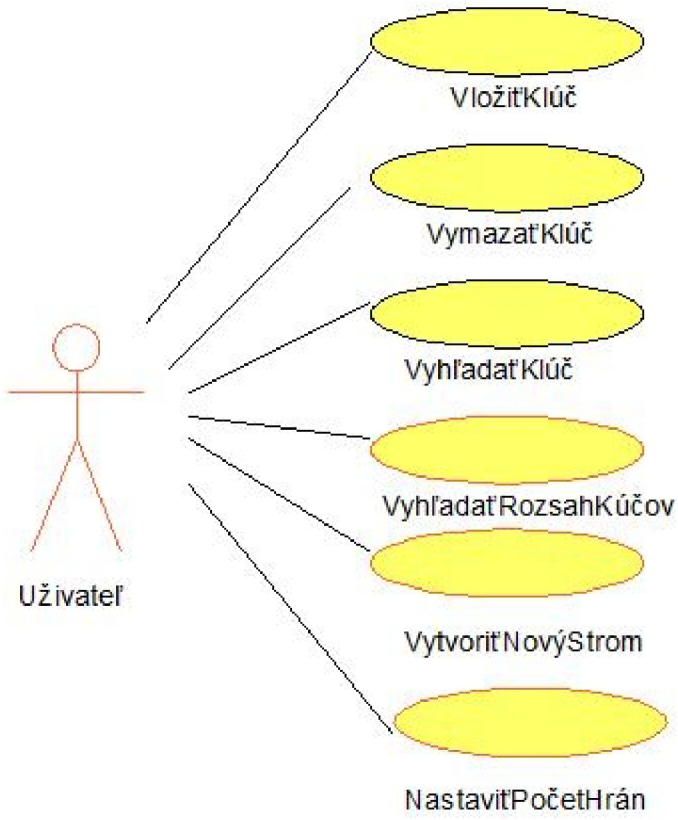
- Root - trieda obsahujúca informáciu o hodnote roota.

4.2 Obmedzenia aplikácie

Pretože vykreslovacia plocha výukového appletu je obmedzená. Kvôli tomu, že dôraz sa kladie hlavne na prehľadnosť, musí dojsť k niektorým obmedzeniam.

- Kľúče môžu byť iba celé čísla od -99 do 99.
- Obmedzený maximálny počet dátových uzlov.
- Počet hrán stromu a jazyk aplikácie možno meniť iba pri vytvorení nového stromu.
- Maximálny počet hrán je nastavený na 7
- Súčasne sa môže vykonávať iba jedna operácia

4.3 Diagram užitia



Obrázok 4.1 Diagram užitia

4.4 Zhrnutie požiadavkov

Zhrnutie základných požiadaviek na vytvorenú aplikáciu.

- Rýchle a plynulé vykresľovanie bez blikania.
- Jednoduché ovládanie.
- Možnosť spustenia na bežnom osobnom počítači.
- Možnosť nastavenia počtu hrán v strome.
- Možnosť vkladania, mazania a vyhľadávania v strome.
- Názornosť prebiehajúcej operácie s výpismi na postrannej ploche.
- Viacjazyčnosť : minimálne slovenčina a angličtina.

5 Implementácia aplikácie

5.1 Implementačné prostredie a použité nástroje

Aplikácia je podľa zadania implementovaná v programovacom jazyku Java. Poskytuje hlavnú výhodu oproti ostatným jazykom, je multiplatformová a pri naprogramovaní vo Windows funguje rovnako ako v Linuxoch, ale aj v MAC OS. Java podporuje dva typy programov – aplikácie a applety. Na rozdiel od programov, applet nemôže fungovať samostatne. Musí byť spustený v kontajneri poskytovanom hostiteľským programom, obyčajne pomocou pluginu. Applet je bezstavový a z bezpečnostných dôvodov disponuje iba obmedzenými možnosťami prístupu na disk.

V tomto výukovom programe kvôli jednoduchosti a možnosti rýchleho spustenia na internete je použitý typ applet. Kvôli nasadeniu na internete nemôže byť applet príliš rozsiahly, aby ho bolo možné pustiť, v čo možno najkratšej dobe. Z bezpečnostných dôvodov nemôžu applety načítavať súbory z disku, a ani ich zapisovať na disk. Preto applet neumožňuje vytvorené B+ stromy ukladať.

Aplikácia bude vytváraná vo vývojovom prostredí NetBeans, kde bude zároveň aj testovaná. Prostredio bolo vybrané pre svoju jednoduchosť a prehľadnosť pri tvorbe zdrojového kódu. Vývojové prostredie sa nachádza na školskom serveri Merlin. Pre testovanie funkčnosti appletu sa používa server Eva, na ktorom sú umiestnené študentské internetové stránky.

5.2 Implementácia

Hlavným cieľom bude vytvoriť aplikáciu podľa požiadavkov uvedených v predošlom návrhu. Implementácia bude prebiehať postupne, vo viacerých na sebe nezávislých krokoch. Medzi hlavné etapy implementácie, ktoré budú vysvetlené podrobne, patrí :

- Vytvorenie základných objektov ako uzol, prvok, koreň a ich vzájomné prepojenie.
- Implementácia vyhľadávania v popredu zadaných uzloch.
- Vkladanie do B+ stromu s dodržaním charakteristických pravidiel.
- Mazanie zo stromu.
- Vytvorenie 2D objektov, ktoré budú reprezentovať B+ strom užívateľovi a ich vykreslenie.
- Animácia vykreslených objektov.

5.3 Implementované triedy

Usporiadané podľa jednotlivých vývojových etáp, postupne ako jednotlivé triedy vznikali. Triedy aplikácie sú pomenované výstižne, aby už z názvu bolo jasné na čo slúžia. Každá vysvetľovaná trieda je zjednodušená a obsahuje iba niektoré najdôležitejšie funkcie a premenné, ktoré budú ďalej objasnené.

5.3.1 Základné objekty aplikácie

V prvej etape implementácie boli vytvorené triedy `Prvky` a `Uzly`, ktoré uchovávajú základné informácie o jednotlivých uzloch a prvkov sa v nich nachádzajúcich. Prvky obsahujú hodnotu kľúča, odkaz na pravého a ľavého suseda. Každý uzol má vlastné ID a typ. Podľa typu uzla sa rozdeľujú na 2

druhy dátové a odkazy. Uzol ďalej obsahuje vektor prvkov, do tohoto vektoru môžme vkladať, mazať alebo v ňom vyhľadávať. Na tieto operácie slúžia funkcie napr. `vloz_prvok`,`odstran_prvok`. Niekedy je vhodnejšie hodnotu prvku iba zameniť, tým sa nám nezmenia odkazy na uzle, ale iba ich hodnota. Pri tejto operácii sa používa funkcia `zmen_hodnotu_prvku`. Objekty boli implementované hneď na začiatku podľa návrhu a ich funkčnosť sa počas vývoju aplikácie menila iba minimálne.

5.3.2 Hlavný program

Trieda Hlavny_program tvorí najpodstatnejšiu časť celej aplikácie, keďže sa v nej nachádzajú všetky základné operácie nad B+ strom a zároveň spája dátovú časť stromu s grafickou. V tejto triede sú implementované vyhľadávanie, vkladanie a mazanie. Taktiež obsahuje vektor prvkov triedy `Uzly` a `Uzly_2D` ktoré reprezentujú pole všetkých uzlov z ktorými aplikácia pracuje a ich polohu.

Prvou operáciou je vyhľadávanie, má dva tvary: jednoduché hľadanie a hľadanie rozsahov. Patrí k dôležitej časti programu, ktorá je volaná samostatne, ale aj z vkladania a mazania, keď chceme zistiť, či sa zadaný kľúč nachádza v strome. Vyhľadávanie je implementované podľa predošlej teórie 2.3.2 . Pri vyhľadávaní rozsahu musia byť zadané dva kľúče, tj. začiatok a koniec rozsahu. Ďalšou funkciou v programe je vkladanie, ktoré je riešené triedou `vloz`. Najprv zavolá funkciu na vyhľadávanie a zistí či sa kľúč v strome nachádza, ak áno vkladanie sa končí neúspešne. Inak sa pokračuje ďalej vo vkladaní na základe pravidiel o vkladaní, ktoré boli spomenuté v 2.3.2 str.8 .

Poslednou funkciou v stromovej štruktúre je mazanie. Patrí k najzložitejšej časti celej aplikácie. Mazanie rozdeľujeme na 3 druhy podľa maximálneho počtu hrán, ktoré môže mazaný uzol obsahovať. Prvé je jednoduché mazanie, ktoré sa používa pri počte hrán 5 a vyššom. Druhé nastáva pri štyroch hranách a trieda, ktorá ho reprezentuje sa volá `vymaz_N_je_4`. Tretie a najzložitejšie pri nastavení troch hrán je `vymaz_N_je_3`. K rozdeleniu došlo kvôli zjednodušeniu riešenia pri presune uzlov s jednou hranou. Všetky druhy mazania používajú navyše triedy hlavného programu, ktoré sú pomenované ako `najdi_suseda` a `zamen_prvok_v_postupnosti_uzlov`. Trieda `najdi_suseda` slúži na nájdení ľavého, alebo pravého suseda. Je volaná pri hľadaní suseda pred splývaním prvkov, alebo pred ich redistribúciou. Funkcia zistí akého suseda má hľadať podľa zadaného čísla, pre pravého suseda to je 1, pre ľavého -1. Trieda vracia vektor s postupnosťou uzlov k danému susednému uzlu. Keď daný uzol nemá suseda na určenej strane tak sa vracia prázdny vektor.

5.3.3 Užívateľské rozhranie

Rozhranie je tvorené objektom triedy `B_plus_applet`, ktorý je odvodený od triedy `Applet`. Táto trieda pri načítaní celého appletu do internetového prehliadača inicializuje zobrazenie všetkých častí rozhrania. Zároveň obsluhuje vstupy z klávesnice a myši. Pri prvej operácii, alebo vymazaní celého stromu vytvorí novú triedu `Hlavny_program` a podľa zvoleného `N` nastaví počet hrán. Rozhranie podľa stlačených tlačidiel posiela kľúč a funkciu, ktorú má Hlavny_program vykonať. Všetky ovládacie prvky sú vytvorené odvodením z triedy `swing`, presnejšie `Javax.swing.JButton` pre tlačítka, `Javax.swing.JTextField` pre formulár a `Javax.swing.JComboBox` pre výber počtu hrán. Rozhranie ďalej obsahuje hlavnú a vedľajšiu plochu, pomocou ktorých sa vykresluje celá animácia appletu a celý postup operácií nad B+ stromom pri jednotlivých funkciách triedy `Hlavny_program`. Hlavná plocha je odvodená od triedy `Javax.swing.JPanel` a navyše obsahuje premenú `n`, do ktorej sa ukladá zvolený počet hrán. `JPanel` je vlastne kontajner, do

ktorého môžme vkladať 2D objekty appletu. Mazanie je riešene nastavením vybraného objektu na neviditeľný. Vedľajšiu plochu tvorí upravená trieda `Javax.swing.JTextArea`, ktorej funkcie z hlavného programu posielajú informácie o tom, aké operácie boli vykonané. Podľa nastaveného jazyka ich v príslušnej jazykovej mutácii vypíše.

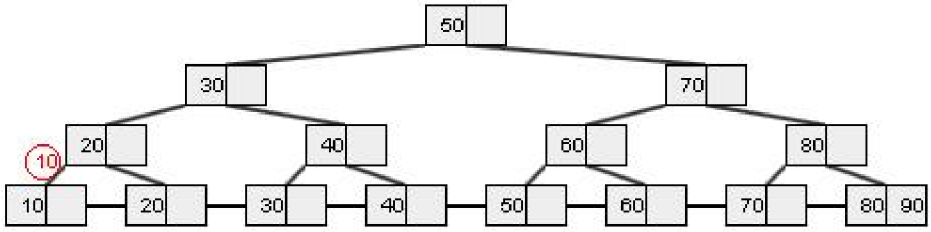
Všetky objekty v užívateľskom rozhraní sú vložené bez správcu rozmiestnenia. S absolútnym pozicovaním, určeným presnými súradnicami. Jednotlivé komponenty a ich umiestnenie je vidieť z obrázku.

1 - formulár na vkladanie kľúča
2, 4, 5, 6, 8 – tlačidlá
3 - výber vyhľadávania rozsahu
7 - výber počtu hrán
9 - hlavná plocha
10 - vedľajšia plocha

Obrázok 5.1 Výrez užívateľského rozhrania s krátkym popisom

5.3.4 2D objekty appletu

Do tejto časti spadajú všetky objekty, ktoré sa vkladajú pri jednotlivých operáciach do kontajneru `hlavna_plocha`. Všetky sú odvodené od triedy `JPanel`, tým že je im zmenená štandardná funkcia `paintComponent` pre vykresľovanie a je nahradená vlastnými grafickými primitívmi. Pre objekt `JPanel_Uzly` to sú jednotlivé štvorčeky, ktoré závisia od počtu hrán B+ stromu a čísla jednotlivých kľúčov sa v nich nachádzajúcich. Naopak pre `JPanel_Ciare` je to čiara, ktorá má zadanú polohu v podobe `X1, Y1, X2, Y2` a podľa toho sa počíta veľkosť objektu, poloha a orientácia vykreslenej čiare. Každá čiara má navyše nastavené vyhladzovanie hrán pre krajší vzhľad. `JPanel_Prvky` obsahujú kružnicu a v nej je číslo kľúča. Všetky objekty sa animujú zmenou polohy pomocou `JPanel` triedy `setLocation`, ktorá vyvolá pohyb objektu v hlavnej ploche a následné prekreslenie vybranej časti plochy.

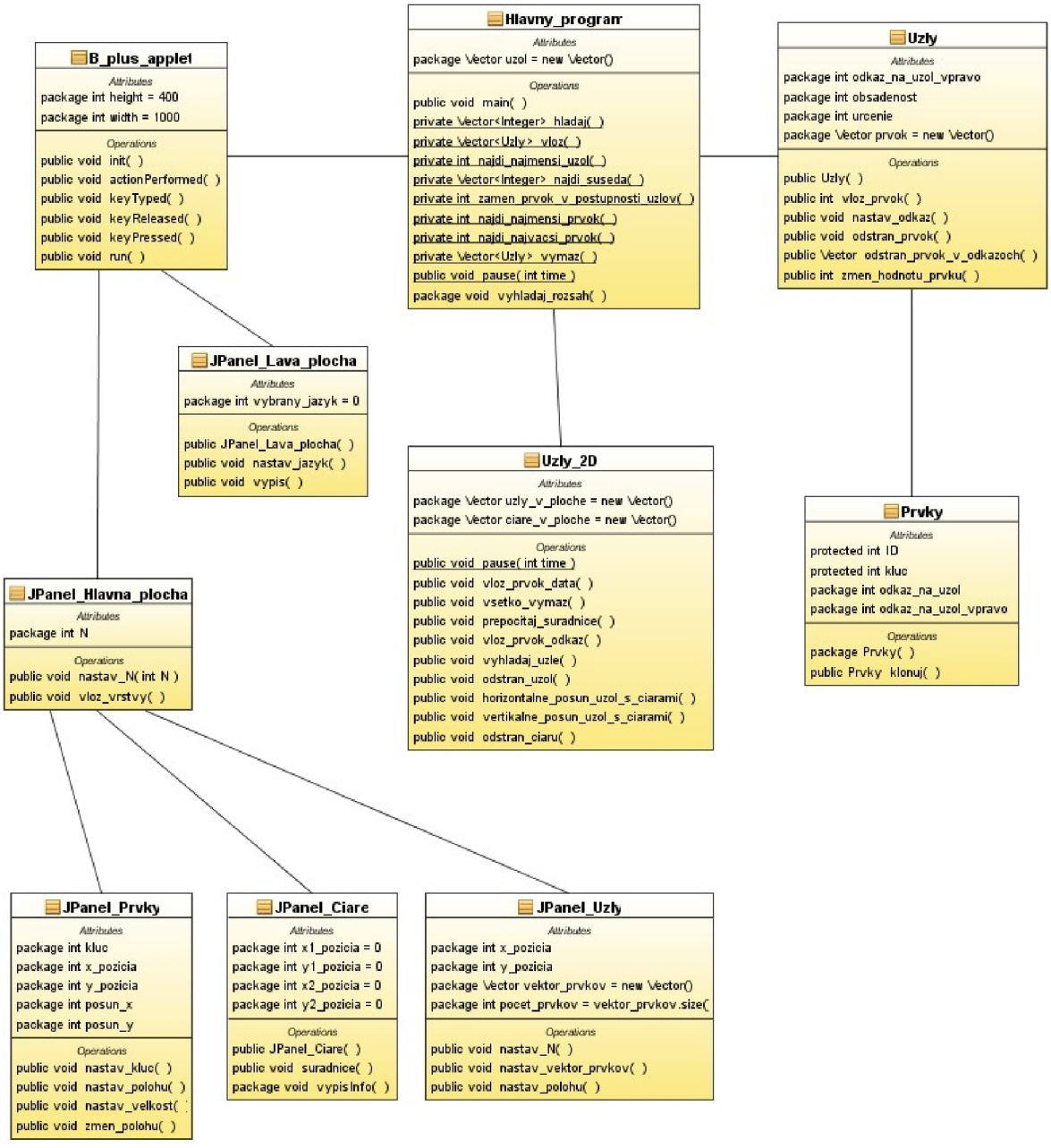


Obrázok 5.2 Všetky 2D objekty v hlavnej ploche

5.3.5 Animácia 2D objektov

Na zmenu polohy všetkých objektov odvodených od triedy `JPanel` v applete sa využíva trieda `2D_uzly`. Je volaná z jednotlivých častí hlavného programu, podľa toho akú operáciu chceme zo stromom vykonať. Pre animáciu vyhľadávania v B+ strome slúži trieda `vyhladaj_uzle`, ktorej stačí iba vedieť `ID` vyhľadávaného uzla a postará sa o animáciu prvku v ploche od rodičovského prvku k zadanému. Zistí si polohu uzla a pomocou zmeny polohy prvku demonštruje vyhľadávanie. Vkladanie podľa toho, čo chceme vložiť používa 3 druhy tried. Prvou je `vloz_prvok_root`, slúži na jednoduché vloženie koreňa stromu, všetkým jeho synovským uzlom musia byť zmenené úrovne, musí prísť k prepočtu súradnice Y. Pri každej zmene je volaná funkcia `vertikalne_posun_uzol_s_ciarami`, ktorá mení polohu uzla a všetkých jeho hrán. Druhou triedou je `vloz_prvok_data`, na vloženie kľúča do sekvenčnej časti stromu. Všetkým susedným uzlom sa pred vložením nového uzla prepočíta súradnica X, tým sa zmení ich poloha a až nakoniec sa vkladá zadaný uzol. Posledným druhom je `vloz_prvok_odkaz`. Uzol sa vkladá do indexovej časti stromu a podľa úrovne sa vkladá do vektoru vektorov. Prvý reprezentuje úroveň a v každej úrovni je vložené pole uzlov nachádzajúcich sa v nej. Tento princíp je použitý kvôli zmene polohy susedných vektorov. Postup vkladania je totožný ako v predchádzajúcej triede. Do stromu môžeme nielen vkladať, ale aj z neho mazať. Princíp je podobný ako pri vkladaní, lenže sa najskorej vymaže zadaný uzol a až potom sa posunú jednotlivé uzly. Pri mazaní koreňa sa uzlom mení súradnica Y a je volaná funkcia `vertikalne_posun_uzol_s_ciarami`, inak sa mení súradnica X a volá sa `horizontalne_posun_uzol_s_ciarami`. Pri odstránení uzla musí navyše zavolať `odstran_ciaru`, ktorá odstráni všetky čiare spojené s daným uzlom.

5.4 Diagram implementovaných tried



Obrázok 5.3 Zjednodušený diagram implementovaných tried

5.5 Vzniklé problémy počas implementácie a ich finálne riešenie

V tejto časti, by som chcel spomenúť aspoň niektoré problémy, ktoré ma postihli a moje riešenie.

V rannej verzii vývoja appletu, bola aplikácia vytváraná v NetBeans ako program, čo malo neskoršie za následok, že v internetovom prehliadači nechcela korektne pracovať, jednalo sa o chybu v distribúcii novej verzii Java. Problém bol vyriešený prechodom na najnovšiu beta verziu vývojového prostredia 6.7M2.

Najväčšie zdržanie vzniklo pri implementácii triedy mazania, kde som si musel uvedomiť ako mazanie v B+ strome vlastne funguje a ako vytvoriť triedu na hľadanie susedov mazaného uzla.

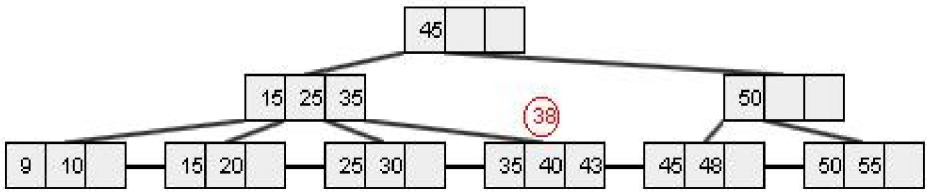
Pri tvorbe animácii dochádzalo k preblikávaniu jednotlivých objektov, čo som sa najprv snažil odstrániť vykreslením celej plochy appletu naraz, čo však neprinieslo očakávaný výsledok. Preto jednotlivé objekty, či sa jedná u uzly, alebo hrany, sú vytvorené pomocou pretypovania swing objektu JPanel.

Pri vkladaní jednotlivých uzlov a čiar prichádzalo k nežiaducim prekryvom, tento problém bol som čiastkovo vyriešil, že som uzle nastavil do vyššej vrstvy ako čiare. A čiaram som ešte zmenil pozadie na prehliadne, tým nedochádza k spomenutým prekryvom.

6 Príklady použitia aplikácie

Nie je možné do technickej spravy vložiť animácie, tak všetky obrázky aplikácie budú zodpovedať stavu pred operáciou prevedenou appletom a stavu po. Ďalej bude nasledovať popis prečo sa tak stalo, aké pravidlá B+ stromu pri jednotlivých operáciach boli použité. Pri každej operácii pre názornosť bude požitý iný počet hrán a iné hodnoty kľúčov.

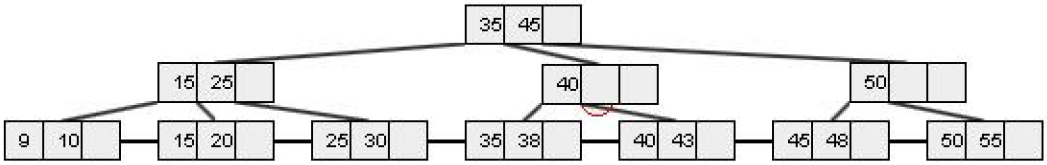
6.1 Vkladanie



Obrázok 6.1 Strom pred vkladaním

Popis vkladania

Do stromu vkladáme kľúč s číslom 38. Najprv sa zistí, či sa vkladaná položka v strome nenachádza a vyberie sa uzol sekvenčnej časti indexu pre vloženie novej položky. Uzol obsahuje kľúče s hodnotami 35,40 a 43. Keďže obsahuje maximálny počet položiek a zaplnenie by bolo väčšie ako n-1, tj. 3, musí sa rozštiepiť. Alokuje sa diskový blok pre uloženie nového uzlu a následne sa prevedie rozdelenie položiek do týchto dvoch uzlov. Musia sa rozdeliť štyri kľúče s hodnotami 35, 38, 40 a 43, tak aby $\lceil (n-1)/2 \rceil$ položiek bolo v pôvodnom uzle a zostatok v novo vytvorenom. V pôvodnom uzle ostane položka s hodnotou 35 a vloží sa nová položka 38. Do nového uzla sa presunie položka 40 a 43. Keďže prišlo k vytvoreniu nového uzla v sekvenčnej časti, musia sa informácie o novom uzle a jeho najmenšej hodnote položky vložiť do rodičovského uzlu v B+ strome, kde je ukazovateľ na uzol, ktorý sa rozštiepil. Položka s hodnotou 40 a ukazovateľom na novo alokovaný uzol sa musí vložiť do uzlu s hodnotami 15,25 a 35. Pretože tento uzol pred vložením znova obsahuje n-1 položiek, tak sa musí takisto analogicky rozštiepiť ako predchádzajúci uzol. V pôvodnom uzle ostanú položky s hodnotou 15 a 25 a do novo vytvoreného uzlu sa presunie ukazovateľ z položky s hodnotou 35 a vloží sa novo vkladaná s hodnotou 40. Opäť je v dôsledku rozštiepenia uzlu potreba vložiť informácie do uzlu rodičovského uzlu. Tím je v našom prípade už koreň. Je v ňom miesto na vloženie novej položky, takže vložením položky celá operácia úspešne skončí.

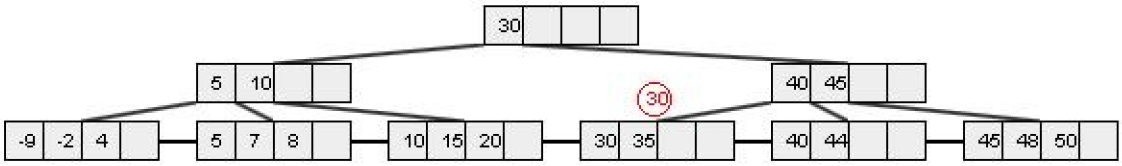


Obrázok 6.2 Strom po vkladaní

6.2 Vyhľadávanie

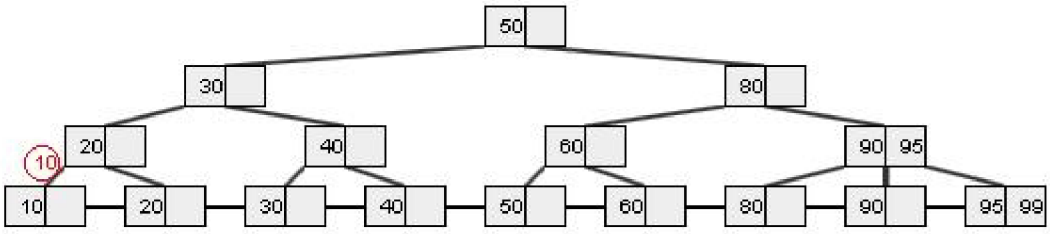
Popis vyhľadávania

Vyhľadáva sa kľúč s číslom 30.Najprv sa načíta uzol koreňa. Jeho prehľadaním sa nájde najbližšiu väčšiu hodnotu 30. Nasleduje načítanie ďalšieho uzla s hodnotami 40 a 45. Jeho prehľadaním sa znovu nájde najbližšia väčšia hodnota 40. Znova nasleduje načítanie uzla, ale to už je uzol sekvenčnej časti. Jeho prehľadaním sa nájde hodnota 30 a vyhľadávanie končí úspešne.



Obrázok 6.3 Obrázok po vyhľadaní

6.3 Mazanie

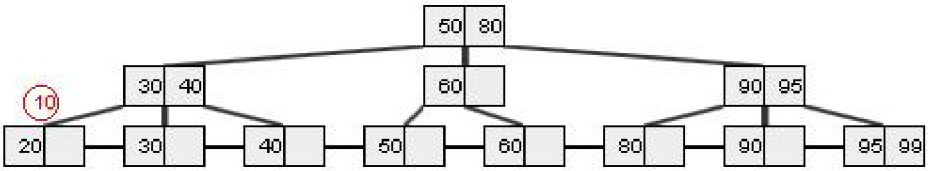


Obrázok 6.4 Obrázok pred mazaním

Popis mazania

Zo stromu mažeme kľúč s číslom 10. Musí sa zistiť, či sa kľúč nachádza v sekvenčnej časti stromu, pokiaľ áno, tak sa vymaže. Keďže to bol jediný kľúč v uzle, klesne zaplnenie uzlu pod prípustné minimum. Nasleduje zlievanie uzlov v sekvenčnej časti. To je v našom prípade jednoduché, zrušenie

prázdneho uzlu. Následok zrušenia uzlu je nutné premietnuť do rodičovského uzlu. Zrušením odpovedajúcej položky tam zostane iba ukazovateľ na najpravejší uzol sekvenčnej časti, ktorý obsahuje položku s hodnotou 20. Zaplnenie uzlu teda kleslo pod prípustné minimum. Ľavého suseda uzol nemá, ale ide previesť zlievanie s pravým susedom. Z mazaného uzla sa presunie kľúč 30 a následne je vymazaný. Pretože i na tejto úrovni stromu zanikol uzol, pokračuje mazanie položky indexu analogicky ďalej. V ďalšej úrovni je uzol s kľúčom 30, ktorému zostal iba jeden ukazovateľ na najpravejší uzol, čiže znovu sa musí vyhľadať sused. Našiel sa pravý sused s hodnotou kľúča 80, znovu nastáva zlievanie a vymazanie mazaného uzla. Keďže rodičovský uzol je koreň a má iba jeden ukazovateľ, tak na koreň sa zmení uzol na ktorý ukazuje, klesne úroveň stromu o jeden a uzol sa vymaže.

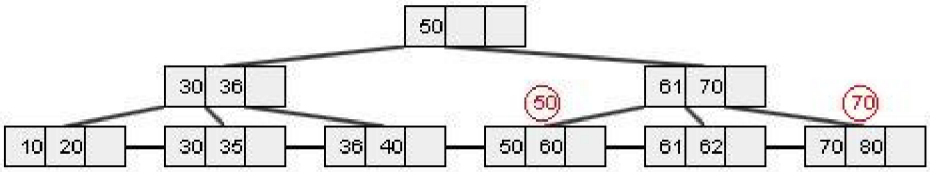


Obrázok 6.5 Obrázok po mazaní

6.4 Vyhľadávanie rozsahov

Popis vyhľadávania rozsahov

Postup vyhľadávania je v indexovej časti stromu rovnaký ako v prípade jednoduchého vyhľadávania. Tentoraz sa vyhľadáva rozsah od 50 do 70. Najprv sa načíta uzol koreňa. Jeho prehľadaním sa dosiahlo konca zaplnenej časti uzlu. Nasleduje načítanie ďalšieho uzla s hodnotami 61 a 70. Jeho prehľadaním sa znovu nájde najbližšia väčšia hodnota 50. Znova nasleduje načítanie uzla, ale to už je uzol sekvenčnej časti. Jeho prehľadaním sa nájde hodnota 50, ale vyhľadávanie ešte nekončí. V sekvenčnej časti vyhľadáva najprv vo svojom uzle, či nemá hodnotu väčšiu alebo rovnú 70. Pomocou odkazu v sekvenčnej časti sa dostáva do uzla s hodnotami 61a 62. Znovu sa nenašla hľadaná hodnota, pokračuje do uzla s hodnotami 70 a 80. Hneď prvý kľúč zodpovedá podmienke, tak vyhľadávanie končí. Našli sa kľúče 50,60,61,62,70 , ktoré spĺňajú zadanú podmienku.



Obrázok 6.6 Obrázok po vyhľadaní rozsahov

7 Záver

Úlohou tejto práce bolo vytvorenie výukovej aplikácie, ktorá má slúžiť na zoznámenie s metódami indexovania v relačných databázach, konkrétne demonštrovanými na B+ strome, ktorý bol vytvorený na základe návrhu popísanom v teoretickej časti. Aplikáciu sa podarilo naprogramovať tak, aby sa dala využívať na vzdelávacie účely. Je možné na nej ukázať ako funguje vkladanie, vyhľadávanie a mazanie v B+ strome. Všetky jej operácie sú totiž plynulo a prehľadne animované s popismi jednotlivých činností.

Pri programovaní som narazil na viacero problémov, ktoré mi pomohli sa zdokonaliť v programovacom jazyku Java.

Čo sa týka možnosti rozvíjať túto prácu, tak som si pri tvorbe programovej časti vytvoril triedy, ktoré idú ďalej použiť na rozšírenie tohto projektu až na úroveň diplomovej práce. V budúcnosti by sa do práce mohli pridať ďalšie stromy (AVL strom, červeno-čierny strom, 2-3-4 strom), alebo hashovanie. Pri vložení viacerých kľúčov by mohlo prísť k zmenšeniu jednotlivých uzlov, a tým by bolo umožnené vytvárať stromy až so stovkami kľúčov.

Za hlavný prínos práce považujem skrátenie času na pochopenie B+ stromu. Študenti si môžu vyskúšať všetky operácie nad B+ stromom a zistiť aká je jeho štruktúra. Pritom im bude pomáhať jednoduché ovládanie a vlastné nastavenie parametra pre počet hrán, čo ovplyvňuje veľkosť a chovanie stromu.

Literatura

[1] Zendulka J., Rudolfová I.: *Databázové systémy*, Brno, 2006, Vysoké učení technické v Brně, Fakulta informačních technologií, [online], [cit. 2009-05-12]
URL https://www.fit.vutbr.cz/study/courses/IDS/private/IDS_predn.pdf

[2] Ramakrishnan R., Gehrke J.:*Database Managment Systems. Third Edition.* McGRAW-HILL. 2003, 1098 p.

[3] Wikipedia: *Database*. [online], posledná aktualizácia 12. 05. 2009, [cit. 2009-05-13].
URL http://en.wikipedia.org/wiki/Database

[4] Kováč M., Kotys M.: *Indexácia B-stromy*, Košice, Technická univerzita Košice, Fakulta elektrotechniky a informatiky, [online], [cit. 2009-05-13]
URL http://hornad.fei.tuke.sk/~genci/Vyucba/SRBDp/2003-2004/04Indexovanie/KovacKotys/srbd.doc

[5] Wikipedia: *Index (database)*. [online], posledná aktualizácia 10. 05. 2009, [cit. 2009-05-14].
URL http://en.wikipedia.org/wiki/Index_(database)

[6] Codd, E.F.: *A relational model of data for large shared data banks*, San Jose, 1970, IBM Research Lab, San Jose, CA.

[7] Elmasri R., Shamkant B. N.: *Fundamentals of database systems Fourth Edition*, Addison-Wesley, 1029 p.

[8] Wikipedia: *B+ srom*. [online], posledná aktualizácia 21. 07. 2008, [cit. 2009-05-15].
URL http://cs.wikipedia.org/wiki/B%2B_strom

[9] Silberschatz A., Korth H., Sudarshan S.: *Database System Concepts Fourth Edition* ,2001 , McGRAW-HILL, 1088 p.

[10] Wikipedia: *Java (programovací jazyk).* [online], posledná aktualizácia 23. 04. 2009, [cit. 2009-05-16].
URL http://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)

[11] Gosling J.,McGilton H.: *The Java Language Environment: A White Paper* , 1995, Sun Microsystems [online], [cit. 2009-05-16]
URL http://java.sun.com/docs/white/langenv/

Zoznam príloh

Príloha 1. CD obsahujúce zdrojové kódy, aplikáciu, manuál a inštalačný súbor vývojového prostredia