UNIVERZITA PARDUBICE
DOPRAVNÍ FAKULTA JANA PERNERA
KATEDRA INFORMATIKY V DOPRAVĚ

IMPLEMENTAČNÍ STRATEGIE SOUBORŮ S ÚPLNÝM INDEXEM

DIPLOMOVÁ PRÁCE

AUTOR PRÁCE: Bc. Jiří Pinkas
VEDOUCÍ PRÁCE: doc. Ing. Antonín Kavička, Ph.D.

2006

UNIVERSITY OF PARDUBICE
THE JAN PERNER TRANSPORT FACULTY
DEPARTMENT OF INFORMATICS IN TRANSPORT

IMPLEMENTATION STRATEGIES OF DENSE INDEX FILES

THESIS

AUTHOR: Bc. Jiří Pinkas
SUPERVISOR: doc. Ing. Antonín Kavička, Ph.D.

2006

Zde bych chtěl poděkovat svému vedoucímu doc. Ing. Antonínu Kavičkovi, Ph.D. za pomoc a odborné vedení při tvorbě diplomové práce.

Fakulta: Dopravní fakulta Jana Pernera, Univerzita Pardubice

Katedra: Katedra informatiky v dopravě

Akademický rok: 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

Pro: **Bc. Jiří Pinkas**

Studijní program: Dopravní inženýrství a spoje

Studijní obor: Aplikovaná informatika v dopravě

Název tématu: **Implementační strategie souborů s úplným indexem**

Zásady pro zpracování:

- Diplomová práce slouží k demonstraci koncepce souborů s úplným indexem, přičemž má následující základní cíle:
  a) vzorovou implementaci souboru s úplným indexem postavenou na abstraktních datových strukturách,
  b) vizualizaci činností základních operací *Vlož* a *Odeber* s akcentem na demonstraci samoorganizujících mechanismů v rámci (hierarchického) indexu,
  c) vytvoření jednoduchého interaktivního softwarového produktu pro podporu výuky problematiky souborů s úplným indexem v rámci předmětu *Datové struktury a algoritmy*.
- Zmíněné implementace jsou realizovány ve vyšším programovacím jazyku Java.

Seznam odborné literatury:

- CENEK, P., KLIMA,V., JANÁČEK, J.: *Optimalizace dopravních a spojových procesů*, Vysoká škola dopravy a spojů, Žilina, 1994
- POKORNÝ, J.: *Databázové systémy a jejich použití v informačních systémech*, Academia, Praha, 1992
- THARP,A.,L.: *File Organization and Processing*, Wiley & Sons, New York, 1988
- FOLK, M.,J.: *File Structures: An Object-Oriented Approach with C++*, Addison Wesley, Boston, 1997

Rozsah: cca 50 stran

Vedoucí práce: **doc. Ing. Antonín Kavička, Ph.D.**

Vedoucí katedry : **doc. Ing. Josef Volek, CSc.**

Datum zadání práce: 30.11. 2005

Termín odevzdání práce: 30.5. 2006

prof. Ing…….. děkan

doc. Ing…… vedoucí ústavu (katedry)

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 29.05.2006

Jiří Pinkas

Abstrakt

V diplomové práci je popsána problematika správy dat na externích paměťových médiích se zaměřením na datovou strukturu Soubor s úplným indexem a popsána vlastní implementace této datové struktury. Součástí implementace je i animace datové struktury Soubor s úplným indexem. Druh animace, který byl navrhnut, je vhodný pro animaci všech druhů k-cestných stromů.

Abstract

In thesis are described techniques for effective managing of data on secondary data storage with accent on data structure called dense index file. There is also described my own implementation of Dense index file data structure. Part of my own implementation is animation of dense index file data structure. Designed animation can be used in future implementation of all n-ary trees.

Obsah

Přílohy

1. Úvod

V současné době se setkáváme se širokým využitím databází. Databáze používají pro uložení dat také systémy v dopravě. Při velkém množství dat je vhodné pro rychlý přístup vytvořit přístupový index. Pro tvorbu indexů se v databázových systémech (např. Oracle, MySQL, PostgreSQL) používají B-stromy nebo jejich variace. Významnou datovou strukturu tvoří Soubor s úplným indexem, který umožňuje vybudovat více přístupových indexů nad jedním bázovým souborem. Tyto indexy jsou tvořeny B^+-stromy.

Tato diplomová práce je pokračováním bakalářské práce, která byla na téma Aplikace B-stromů v organizacích indexovaných souborů. V bakalářské práci byla implementována datová struktura B^+-strom ve vyšším programovacím jazyce Pascal. V diplomové práci je implementována datová struktura Soubor s úplným indexem, ve které je index realizován datovou strukturou B^+-strom. Dále byla navrhnuta a implementována animace datové struktury Soubor s úplným indexem. Praktická část diplomové práce byla naprogramována ve vyšším programovacím jazyce Java a je možné ji spustit přímo z internetového prohlížeče, nebo jako samostatnou aplikaci.

Ve třetí kapitole bude prezentována problematika ukládání dat na vnější paměťová média. Nejprve se zadefinují důležité pojmy z této problematiky, poté bude vysvětlen teoretický základ datové struktury typu B-strom se zaměřením na variaci B-stromu, jíž je datová struktura B^+-strom. Zde se uvedou algoritmy základních operací nad B^+-stromem, jimiž jsou nalezení záznamu, vložení záznamu a odebrání záznamu. Poté se uvede datová struktura Neutříděný soubor s přímým přístupem (halda-soubor) a nakonec bude prezentována datová struktura Soubor s úplným indexem, kde index je tvořen B^+-stromem.

Ve čtvrté kapitole bude prezentována vlastní implementace datové struktury Soubor s úplným indexem. Tato kapitola je rozdělena na dvě části. První částí je implementace Souboru s úplným indexem a ve druhé části je návrh a implementace grafické reprezentace datové struktury Soubor s úplným indexem.

V další kapitole bude uvedeno nejznámější použití datové struktury Soubor s úplným indexem v praxi, jímž je použití v databázových systémech pro ukládání dat.

Příloha A obsahuje uživatelskou příručku programové aplikace. V této příloze je popsáno ovládání aplikace, demo režim, a také omezení aplikace pro výukové účely.

Součástí diplomové práce je programová aplikace, která se nachází na přiloženém CD médiu. V této aplikaci je znázorněna grafická podoba datové struktury Soubor s úplným indexem a její animace. Aplikaci je možné umístit na web a pak lze spustit přímo z Internetového prohlížeče pomocí technologie Java WebStart. Aplikaci lze také spustit klasickým způsobem. Ke správné funkčnosti je zapotřebí prostředí Java Runtime Edition od společnosti Sun verze 1.4 a vyšší.

Dále jsou součástí diplomové práce kompletní zdrojové kódy, které obsahují implementaci Souboru s úplným indexem bez grafického výstupu, implementaci Souboru s úplným indexem s animací a nakonec moduly pro rozšíření animace o jiné druhy stromů.

2. Cíl práce

Tato práce slouží k demonstraci koncepce souborů s úplným indexem. Praktická část práce má za cíl vytvořit vzorovou implementaci souboru s úplným indexem postavenou na abstraktních datových strukturách, vizualizaci činností základních operací *Vlož* a *Odeber* s akcentem na demonstraci samoorganizujících mechanismů v rámci hierarchického indexu a vytvoření jednoduchého interaktivního softwarového produktu pro podporu výuky problematiky souborů s úplným indexem v rámci předmětu *Datové struktury a algoritmy*. Vše je realizováno ve vyšším programovacím jazyku Java.

V textu diplomové práce je cílem přiblížit problematiku ukládání dat na externí paměťová média se zaměřením na datovou strukturu Soubor s úplným indexem, kde index je tvořen datovou strukturou typu B^+-strom.

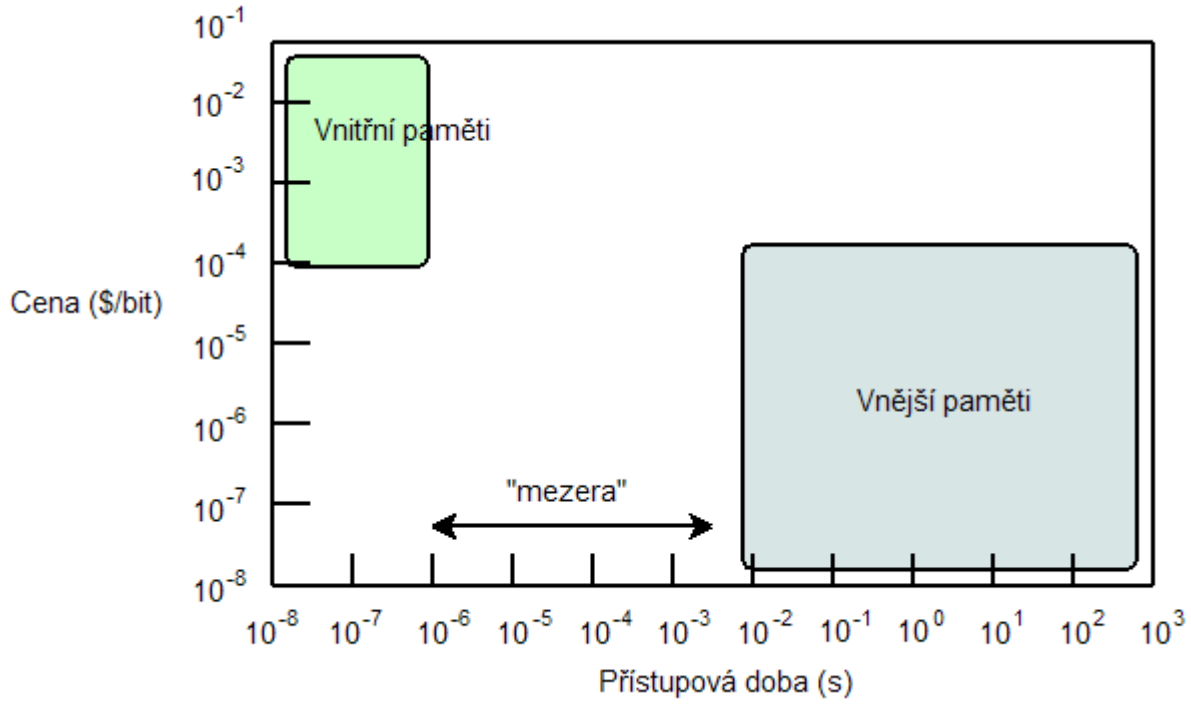
Cílem práce je vytvořit vhodnou učební pomůcku při studiu datové struktury Soubor s úplným indexem s ohledem na budoucí rozšíření o animaci dalších typů stromů.

3 Stav problematiky v literatuře

3.1 Fyzické vlastnosti externích paměťových médií

V počítači existuje mnoho hardwarových komponentů, které zabezpečují paměť pro uložení dat. Paměti se dělí na vnitřní a vnější. Vnitřní paměť (nebo též operační paměť) počítačového systému se v současné době skládá z křemíkových čipů. Tato technologie je obvykle 2x dražší než externí paměťové média, která jsou v současné době nejčastěji zastoupena pevnými disky a magnetickými páskami a tvoří tzv. vnější paměť. Naprostá většina počítačových systémů se v současné době skládá z vnitřní a vnější paměti, viz. [8], [27].

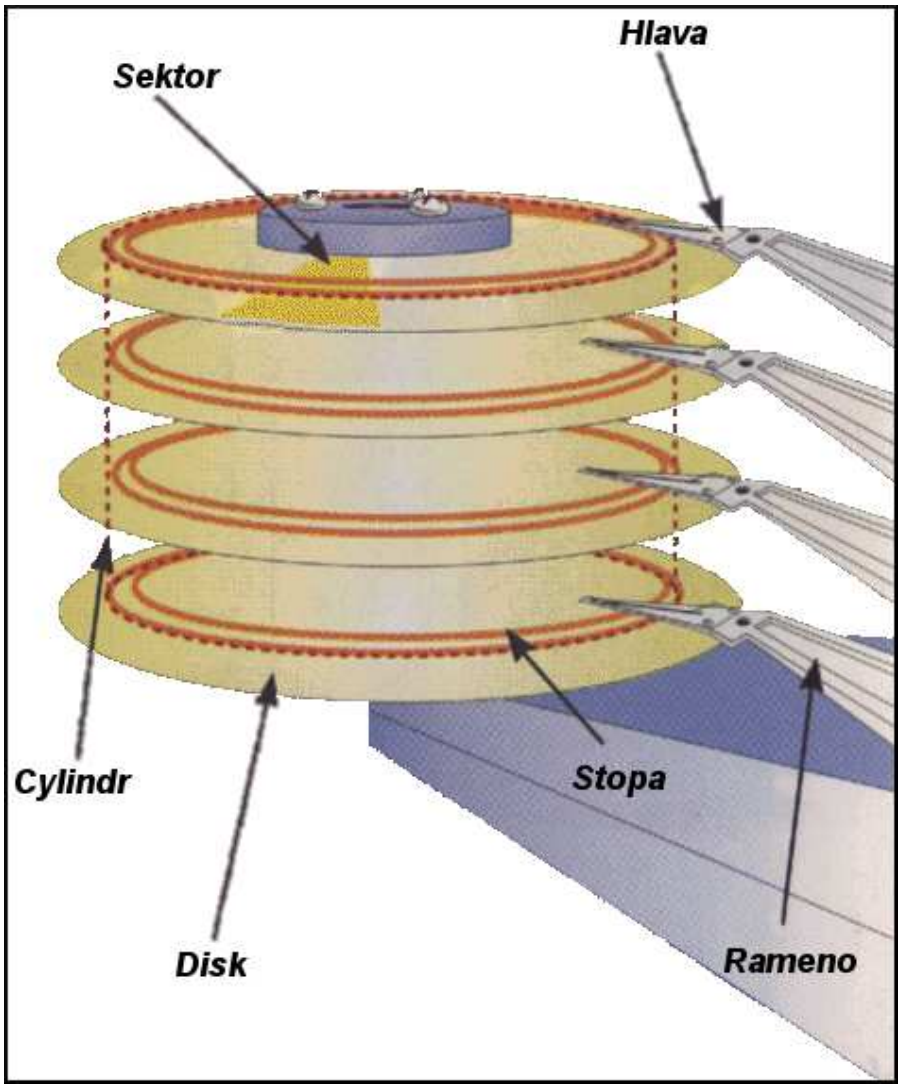
Na obrázku 1 je znázorněna cena za bit vs. přístupová doba vnitřní a vnější paměti. Z obrázku je zřetelná určitá „mezera“, která ukazuje, že přístupová doba u vnější paměti je několikrát větší než u vnitřní paměti. Příčiny a důsledky této „mezery“ jsou vysvětleny dále v této kapitole.



Obr. 1: Cena za bit vs. přístupová doba. Zdroj: [8], [25], [26].

Na obrázku 2 je zobrazeno schéma typického pevného disku (harddisku), který se skládá z:

- magnetických hlav pro zápis a čtení,
- médií na kterých jsou uložena data (cylindry),
- mechaniky, jejímž úkolem je pohybovat hlavami,
- řadiče pro řízení práce disku.



Obr. 2: Diskový svazek. Zdroj: [5].

Přestože jsou disky lacinější a mají větší kapacitu než vnitřní paměť, jsou pomalejší, protože mají pohyblivé části. Harddisk obsahuje dvě pohyblivé části: rotace cylindru a pohyb přístupového ramene [8]. V současné době rotují disky rychlostí 5400-15000 RPM (otáček za minutu), přičemž nejčastěji jsou zastoupeny disky s rychlostí 7200 RPM. Přestože 7200 RPM se může zdát rychlé, jedna rotace zabere 8.33 milisekundy, zatímco přístupová doba křemíkové paměti je v současné době 3 – 10ns (nanosekund). V současné době je průměrná přístupová doba disku v rozmezí 3 až 9 milisekund, viz. [24], [25] a [26].

Často tudíž trvá mnohem déle zpřístupnění informace z externího paměťového média, než její zpracování. Z toho důvodu je vhodné načíst větší množství dat najednou z externího paměťového média a poté v operační paměti tyto data zpracovat.

3.2 Přístup k záznamům v souboru

Zpřístupnění záznamů v souboru je možné provést dvěma způsoby. Sekvenčním přístupem, nebo přímým přístupem. V sekvenčním přístupu se přistupuje k datům podle následníka. Při přímém přístupu se přistupuje k záznamům pomocí klíče nebo podle indexu (jako v poli).

Přenos údajů mezi vnitřní a vnější pamětí se provádí po blocích. Přenos bloku je časově náročnější než prohledávání záznamů přeneseného bloku v operační paměti. Proto je hlavním ukazatelem časové efektivnosti počet přenosů bloku mezi operační a vnější pamětí [1].

3.2.1 Sekvenční přístup

Sekvenční přístup se využívá v sekvenčním souboru. V sekvenčním souboru jsou záznamy uloženy za sebou v libovolném (neutříděném) pořadí, posledním záznamem je speciální záznam, označovaný jako **eof** (end of file). Tento druh přístupu je aplikovatelný na pásce, nebo na disku [1].

Sekvenční přístup je vhodné použít v případě, že je potřebné zpracovat všechny záznamy v souboru. Lze jej tedy rovněž používat i u souborů, u nichž je implementovaný přímý přístup, pro případy skupinového zpracování celého souboru.

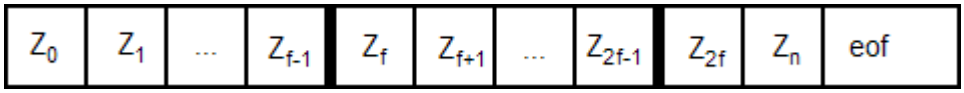
3.2.2 Přímý přístup

V přímém přístupu se k jednotlivým záznamům přistupuje pomocí klíče (tj. jedná se o implementaci ADT tabulka na disku), nebo podle indexu (implementace ADT pole na disku). Tento druh přístupu je aplikovatelný na disku [1].

3.3 Způsoby organizace souboru

3.3.1 Sekvenční organizace

Na obrázku 3 je znázorněn sekvenční soubor. Tento soubor je rozdělen na do bloků a obsahuje celkem *n* záznamů. Počet záznamů v každém bloku je *f*. Posledním záznamem v sekvenčním souboru je speciální záznam *eof* (end of file), který označuje konec souboru. Data mohou být v souboru utříděná nebo neutříděná.

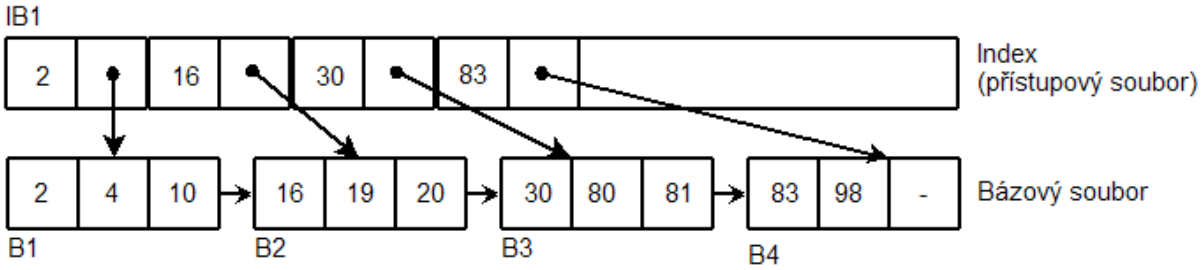


Obr. 3: Sekvenční soubor. Zdroj: [28].

3.3.2 Indexovaná organizace

Indexovaná organizace se používá v datových strukturách, které obsahují přístupovou strukturu, tzv. index. Datové struktury na externích paměťových médiích se skládají z indexu a bázového souboru. Index je obvykle tvořen utříděným souborem, jehož záznamy obsahují reference na záznamy (data) v bázovém souboru.

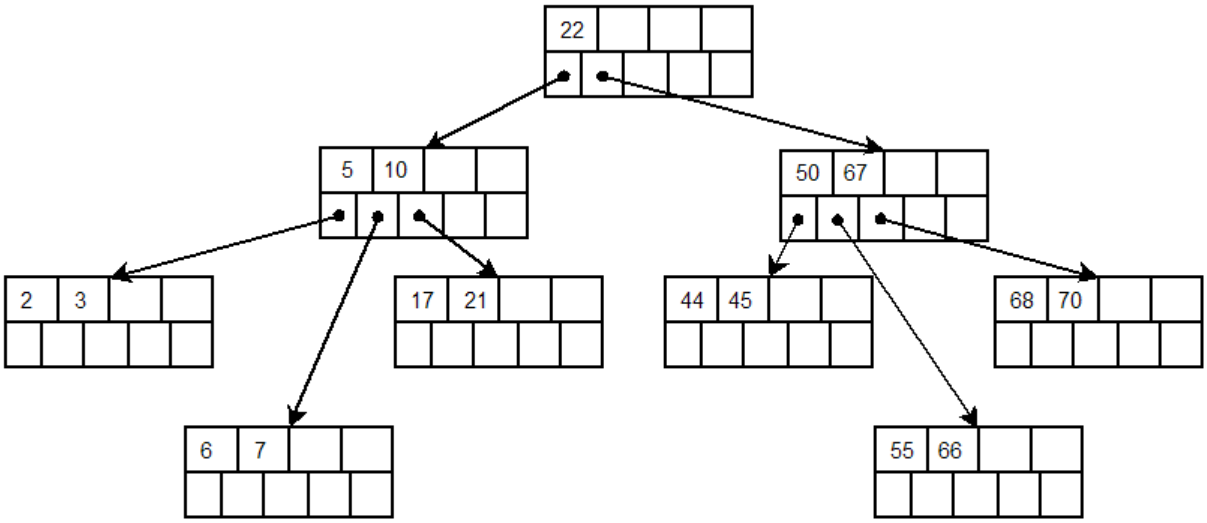
Na dalším obrázku je typický zástupce indexované organizace, jímž je indexsekvenční soubor.



Obr. 4: Datová struktura indexsekvenční soubor. Zdroj: [28].

3.3.3 Hierarchická organizace

Hierarchická organizace je podmnožina indexové organizace, přičemž index je tvořen stromem. Na dalším obrázku je jedna z datových struktur, která používá hierarchickou organizaci. Touto datovou strukturou je datová struktura B-strom.



Obr. 5: Datová struktura B-strom. Zdroj: [29].

3.3.4 Hashovací organizace

Hashovací organizace je taková organizace, která je tvořena tzv. hashovacím souborem. Jedná se o soubor, který pro zpřístupnění záznamu podle klíče využívá techniku transformace klíče pomocí hashovací funkce. Hashovací funkce vrací adresu bloku, v němž je obsažen záznam s hledaným klíčem.

3.4 Datová struktura typu B-strom

3.4.1 Soubory s přímým přístupem

Soubory s přímým přístupem zajišťují přímý přístup k záznamům v souboru. K dispozici jsou následující operace:

- `procedure CtiBlok (CisloBloku: integer);`
- `procedure ZapisBlok (CisloBloku: integer);`

Procedura *CtiBlok()* přečte z vnějšího paměťového média blok *CisloBloku* a uloží ho do určené části operační paměti (nazývané buffer). Procedura *ZapisBlok()* zapíše buffer z operační paměti na vnější paměťové médium na místo *CisloBloku*.

Záznamy se vyhledávají a identifikují na základě klíče. Klíčem je zvolený atribut (resp. kolekce atributů) záznamu, který tento záznam jednoznačně identifikuje a odlišuje ho od ostatních záznamů. V případě, že je klíčem kolekce atributů, jedná se o tzv. složený klíč [1]. Z toho vyplývá, že se na soubor s přímým přístupem můžeme dívat jako na implementaci abstraktního datového typu Tabulka ve vnější paměti.

Záznam tabulky lze stylizovaně vyjádřit:

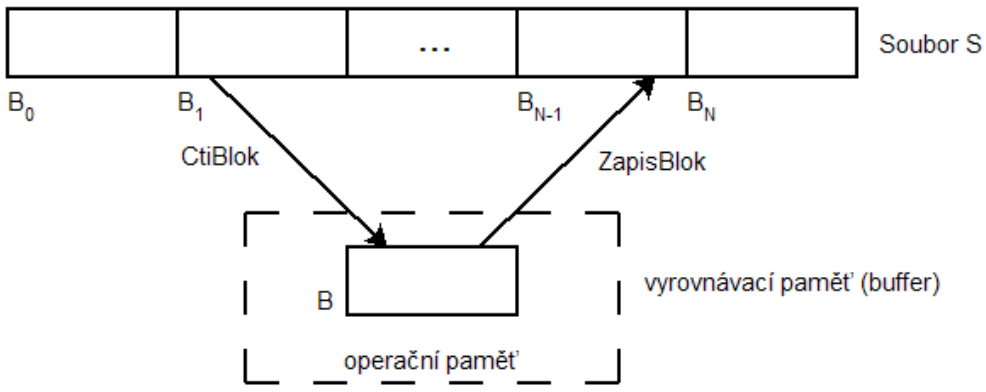
```
TypZaznamu = class
    Klic : TypKlice;
    Data : TypDat;
end;
```

Abstraktní datový typ Tabulka má definovány operace pro práci podle klíče. Těmito operacemi jsou následující metody (stylizovaně vyjádřeno):

- `function NajdiZaznam (Klic : TypKlice) : TypZaznamu;`
- `procedure VlozZaznam (Zaznam: TypZaznamu);`
- `function OdeberZaznam (Klic : TypKlice) : TypZaznamu;`

Přenos údajů mezi vnitřní a vnější pamětí se provádí po blocích. Soubor považujeme za posloupnost bloků B_0, B_1, … B_N, kde N je aktuální počet bloků v souboru. K příslušnému bloku lze přistoupit podle jeho pořadového čísla.

Na obr. 6 je uveden vztah mezi souborem s přímým přístupem a bufferem a princip operací *CtiBlok* a *ZapisBlok*:



Obr. 6: Princip operací CtiBlok a ZapisBlok. Zdroj: [1]

Uveďme příklad zdrojového kódu bufferu:

```
TBuffer = class
    Buffer : array of Byte;         //Fyzický Buffer
    VelBuf : integer;               //Velikost bufferu
    VelZaznamu : integer;           //Velikost záznamu
    Data : file;                    //Soubor s přímým přístupem
    jmsouboru : string;             //Jméno souboru
    procedure CtiBlok (CisloBloku: integer);
    procedure ZapisBlok (CisloBloku: integer);
end;
```

Třída *TBuffer* se skládá z vyrovnávací paměti (*Buffer*), souboru s přímým přístupem (*Data*) a dalších pomocných atributů. Dále obsahuje metody *CtiBlok()* a *ZapisBlok()*.

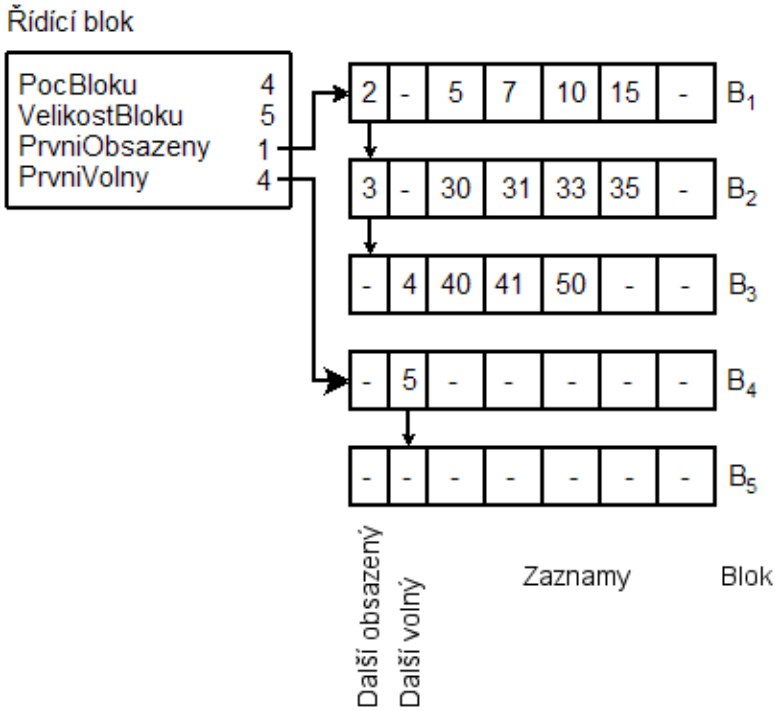
Nyní následuje příklad zdrojového kódu souboru s přímým přístupem:

```
TSouborSPrimymPristupem = class
    Buffer : TBuffer;               //Buffer
    RidiciBlok : TRidiciBlok;       //Řídící blok
    procedure CtiRidiciBlok;
    procedure ZapisRidiciBlok;
    function Najdi (Zaznam : TZaznam) : TZaznam;
    procedure Vloz (Zaznam : TZaznam);
    procedure Odeber (Zaznam : TZaznam) : TZaznam;
    property VolnyBlok : integer read GetVolnyBlok write SetVolnyBlok;
end;
```

Soubor s přímým přístupem se skládá z bufferu a řídícího bloku. V souboru s přímým přístupem se nachází metody pro práci s řídícím blokem *CtiRidiciBlok()* a *ZapisRidiciBlok()*, dále jsou zde metody pro operace se záznamy v bufferu a property *VolnyBlok*, která je určena pro správu volných bloků v souboru.

Součástí souboru s přímým přístupem je *řídící blok*. V něm jsou obsaženy informace, které charakterizují celý soubor (např. velikost jednoho bloku, počet bloků, …). *Řídící blok* je vhodné uložit do nultého bloku souboru.

Na následujícím obrázku je znázorněn vztah mezi řídícím blokem a jednotlivými bloky v souboru s přímým přístupem.



Obr. 7: Schéma řídícího bloku. Zdroj: [1].

Příklad zdrojového kódu řídícího bloku (stylizovaně):

```
TRidiciBlok = class
   PocBloku : integer;         //Počet bloků
   VelikostBloku : integer;    //Velikost jednoho bloku
   PrvniObsazeny : integer;    //První blok v logické posloupnosti bloků
   PrvniVolny : integer;       //Číslo vrcholu seznamu volných bloků
end;
```

V řídícím bloku jsou dvě posloupnosti bloků. Posloupnost neprázdných bloků začíná v bloku, jehož index se nachází v řídícím bloku v atributu *PrvniObsazeny*. Každý blok obsahuje atribut *Další obsazený*, který obsahuje referenci na další obsazený blok, případně referenci neobsahuje, pokud je posledním blokem zřetězení neprázdných bloků. Zřetězeny jsou i bloky, které neobsahují žádné záznamy. První blok zřetězení volných bloků je obsažen v atributu *PrvniVolny* v řídícím bloku a bloky jsou zřetězeny pomocí atributu *Další volný*, který obsahují všechny bloky souboru. Zřetězení volných bloků je tvořeno obdobným způsobem jako zřetězení neprázdných bloků.

3.4.2. Indexové soubory

Nejvýznamnější vlastností indexovaných souborů je, že soubor záznamů (hlavní soubor) je rozdělen do bloků a utříděn podle klíče. To znamená, že záznamy jsou uvnitř každého bloku utříděny a pro $i < j$ mají všechny záznamy bloku B_i hodnoty klíčů menší než libovolný záznam z bloku B_j [1]. Díky této vlastnosti lze efektivně realizovat algoritmy přístupu.

Strukturu indexovaného souboru lze vyjádřit:

```
TIndexovanySoubor = class
    Index : TSouborSPrimymPristupem;           //Indexový soubor
    HlavniSoubor : TSouborSPrimymPristupem;    //Hlavní (bázový) soubor
end;
```

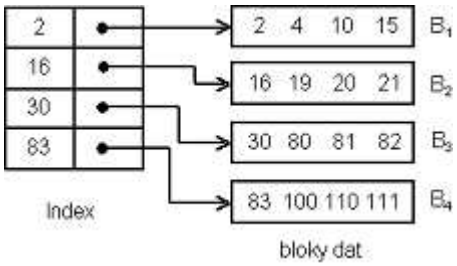
V hlavním souboru se nachází záznamy, které mají stejné složení jako záznamy v souborech s přímým přístupem:

```
TZaznam = class
    Klic : TKlic;            //Klíč (např. číslo)
    Data : TData;            //Data (může být cokoliv)
end;
```

V indexovém souboru jsou záznamy, které mají následující strukturu:

```
TIndexZaznam = class
    Klic : TKlic;            //Klíč (např. číslo)
    B : integer;             //Reference na blok
end;
```

Takových dvojic je právě tolik, kolik je bloků v hlavním souboru, přičemž tato dvojice *(Klic, B)* znamená, že první záznam bloku *B* hlavního souboru má hodnotu klíčového záznamu *Klic*.



Obr. 8: Struktura indexovaného souboru. Zdroj: [1].

Vyhledávání v indexovém souboru se liší od klasického vyhledávání – hledá se *pokrytí* [1], které je definováno: je-li dán klíč *K* indexovaného souboru (argument hledání), je třeba v indexu najít takový záznam ($Klic_1$, B_1), aby platilo $Klic_1 \leq K$, přičemž záznam ($Klic_1$, B_1) je buď poslední v indexovém souboru, anebo pro následující záznam ($Klic_2$, B_2) platí $K < Klic_2$. Říkáme, že $Klic_i$ pokrývá *K*.

Vyhledávání v záznamu s klíčem *K* je dvojstupňové. Nejprve se indexu najde záznam *P*, ve které *P.Klic* pokrývá *K* a potom se zpřístupní a následně v operační paměti prohledá blok *P.B* hlavního souboru.

3.4.3. Datová struktura B-strom

V roce 1972 dva počítačoví odborníci pracující pro Boeing Company v Seattlu vytvořili novou stromovou strukturu, kterou nazvali *B-strom (balanced tree)* [7]. B-stromy jsou *k*-cestné stromy, kde *k* je řád B-stromu, které jsou optimalizovány pro diskové operace. V B-stromu každý uzel obsahuje seznam klíčů (jejich počet je maximálně 2*k* - 1). Výška B-stromu není předem dána, může automaticky růst (v důsledku přidávání záznamů), anebo klesat (při odebírání záznamů) tak, že se B-strom udržuje vždy dokonale vyvážený. Při

vyhledávání záznamů v B-stromu se začíná u kořene a hledáním pokrytí v uzlu algoritmus rozhoduje, kterou cestou pokračovat. Díky tomu se při vyhledávání záznamu v B-stromu musí načíst do paměti pouze malé množství uzlů a složitost je tak v nejhorším případě řádu $log_k(n)$, kde k je řád B-stromu a n je počet záznamů v B-stromu [6].

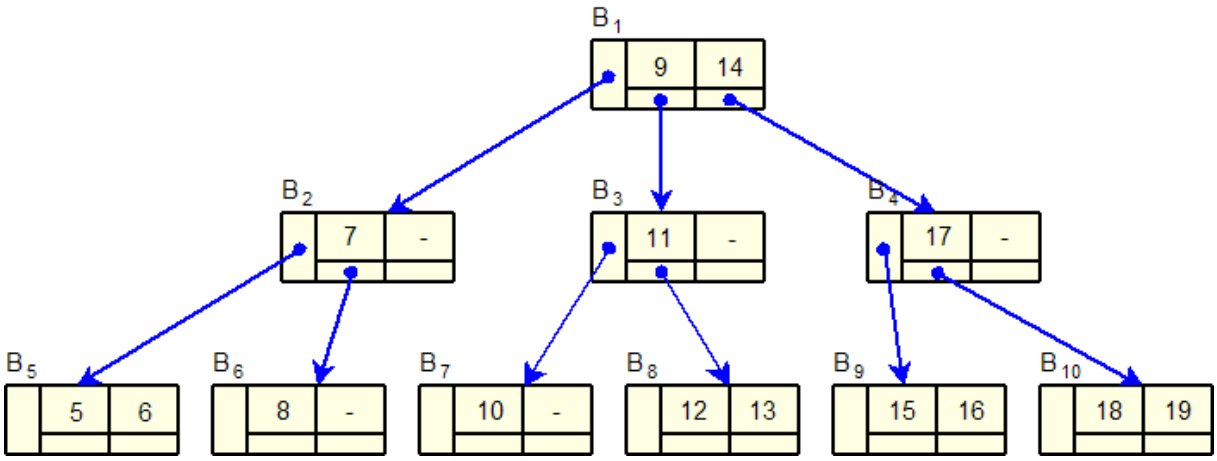
Strukturu B-stromu lze vyjádřit ve tvaru:

```
TBStrom = class (TSouborSPrimymPristupem)
    function Najdi (Zaznam : TZaznam) : TZaznam;
    procedure Vloz (Zaznam : TZaznam);
    function Odeber (Zaznam : TZaznam) : TZaznam;
end;
```

B-strom je odvozen od ADT Soubor s přímým přístupem. Jsou v něm dále uvedeny metody *Najdi()*, *Vloz()* a *Odeber()* pro základní operace se záznamy.

V následujícím textu budou používány pojmy uzel a blok jako synonyma. B-strom je stromová struktura, která je složena z uzlů. Fyzicky je ale B-strom složen ze dvou souborů s přímým přístupem, které se skládají z bloků. Pojem uzel bude použit v souladu s tím v kontextu logického uspořádání B-stromu, zatímco pojem blok bude používán v kontextu fyzické reprezentace B-stromu.

Na obr. 9 je uveden příklad B-stromu řádu 2:



Obr. 11: Datová struktura B-strom řádu 2. Zdroj: vlastní.

B-strom má následující vlastnosti [1]:

1. Každý blok obsahuje maximálně $2N$-1 záznamů, přičemž $N \geq 2$ se nazývá *řád B-stromu*. Minimální počet záznamů v bloku je N, což jinými slovy znamená, že využití bloku v B-stromu musí být vždy minimálně 50%. Složení záznamů je následující (stylizovaně):

```
TBStromZaznam = class
    Klic : TKlic;       //Klíč
    B : integer;        //Reference na blok
    Data : TData;       //Data
end;
```

   Atribut *B* je adresa bloku v B-stromu. První záznam každého bloku indexu má atribut *Klic* nevýznamný (často se nezobrazuje, v implementaci nemusí existovat vůbec).

Způsob hledání pokrytí v B-stromu bude nyní demonstrován na několika situacích z obr. 9:

- První záznam bloku B_1 pokrývá klíče z intervalu <-∞, 9).
- Třetí záznam bloku B_1 pokrývá klíče z intervalu <14, ∞).
- První záznam bloku B_2 pokrývá klíče z intervalu (-∞, 7).
- Druhý záznam bloku B_3 pokrývá klíče z intervalu <9, 11).

2. Kořen B-stromu obsahuje 1 až 2*N*-1 záznamů.
3. B-strom je vždy dokonale vyvážený, tzn. všechny cesty od kořene po list mají stejnou délku. Vyváženosti B-stromu se dosahuje proměnnou výškou. Při vkládání záznamů se může výška B-stromu zvyšovat, při odebírání se naopak může snižovat. B-strom se tudíž řadí do skupiny *samoorganizujících struktur*.

3.4.4 Datová struktura B$^+$-strom

Klasická definice B-stromu říká, že B-strom je *k*-cestný strom, kde *k* je řád B-stromu, který splňuje následující omezení [1]:

- Kořen má nejméně dva přímé potomky, pokud není listem.
- Každý uzel kromě kořene a listů má nejméně *k* přímých potomků.
- Všechny cesty stromu od kořene k listům jsou stejně dlouhé.

V současné době neexistuje jednotná definice B$^+$-stromů. Byla vybrána následující definice B$^+$-stromu:

B$^+$-strom je variace B-stromu, která má následující odlišnosti [5]:

1. Všechna data jsou soustředěna na úrovni listů, ve vrchních uzlech (nelistech) jsou pouze klíče.
2. Každý list má navíc referenci, která slouží k získání logicky následujícího listu (pro potřeby uspořádaného průchodu všech záznamů v bázovém souboru podle klíče vzestupně).

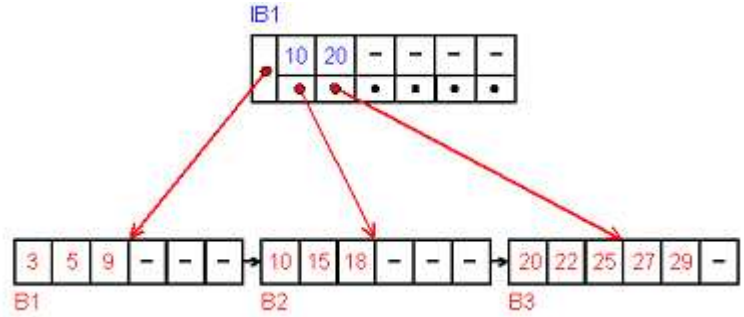
Složení záznamů indexového souboru je následující (stylizovaně):

```
TIndexZaznam = class
    Klic : TKlic;            //Klíč (např. číslo)
    B : integer;             //Reference na blok
end;
```

Strukturu záznamu bázového souboru lze vyjádřit následovně (stylizovaně):

```
TBaseZaznam = class
    Klic : TKlic;            //Klíč
    Data : TData;            //Data
end;
```

Na obr. 10 je znázorněn příklad B^+-stromu řádu 6:



Obr. 10: Datová struktura B^+-strom řádu 6. Zdroj: vlastní.

Původní B-strom je *homogenní struktura* [6]. B^+-strom je naopak *heterogenní strukturou* [6]. Záznamy indexu obsahují pouze klíče, záznamy bázového souboru obsahují klíče a data. Díky referencím na následující logické bloky v bázovém souboru je k dispozici možnost jednoduše vypsat všechny záznamy bázového souboru vzestupně.

3.4.5. Operace hledání záznamu v B^+-stromu

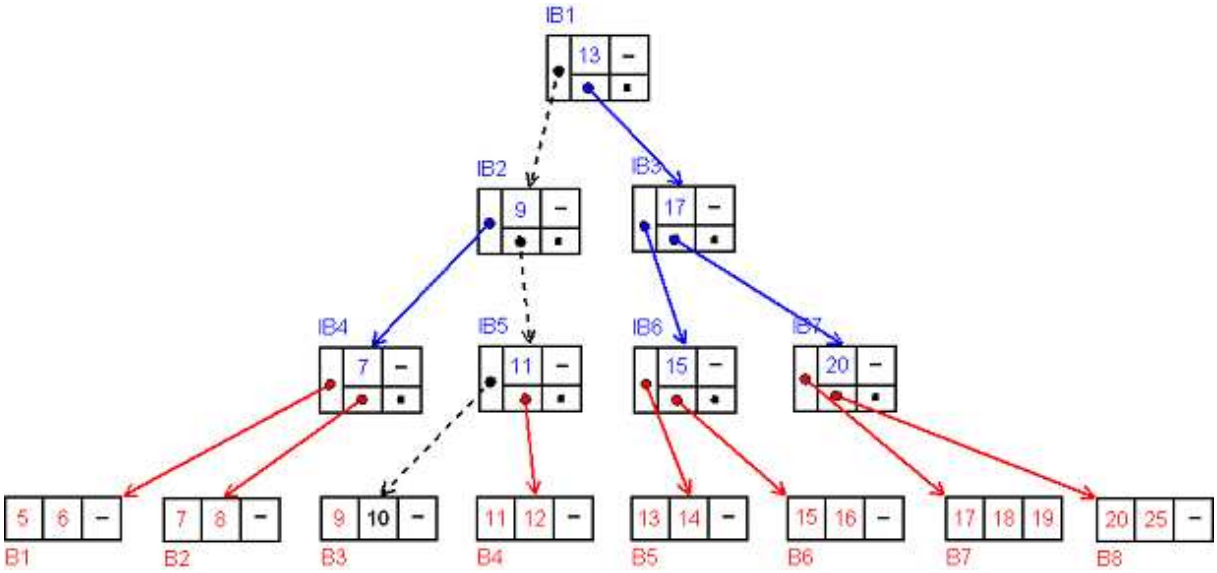
Hledání záznamu v B^+-stromu je založeno na stejném principu jako hledání záznamu v binárním vyhledávacím stromu. Rozdíl mezi B^+-stromem a binárním vyhledávacím stromem je v tom, že místo rozhodování mezi dvěma potencionálními cestami u každého uzlu se rozhoduje podle aktuálního počtu záznamů v uzlu. Začíná se vždy v kořeni, dále se hledá pokrytí v uzlu a podle výsledku se pokračuje dál.

Hledání záznamu s hodnotou vyhledávaného klíče *k*:

1. Start hledání v kořeni.
    a. Hledá se záznam s nejmenší hodnotou vyhledávacího klíče $\geq k$.
    b. Jestliže taková hodnota existuje, předpokládejme, že je to K_i. Pak se přejde na uzel, který zpřístupňuje reference B_i.
    c. Pokud neexistuje ($k < K_1$), pak se přejde na uzel, na který ukazuje reference B_0.
2. Pokud aktuální uzel není listem, pak se opakuje bod 1.
3. Dojde se do listu (v bázovém souboru). Jestliže v listu existuje $K_i = k$, výsledkem je tento záznam. V opačném případě záznam s hodnotou hledaného klíče *k* neexistuje.

Příklad hledání záznamu s klíčem 10 (obr. 11):

- Porovná se hledaný klíč 10 s klíči v kořeni – zde existuje jen jeden záznam s klíčem 13. Klíč 10 je menší než 13, tudíž vyhledávání bude pokračovat v bloku *IB2*. Zde se opět provede stejné porovnání klíčů (10 je větší než 9), tudíž se přejde do indexového bloku *IB5*, kde je realizována stejná koncepce vyhledávání a pokračuje se do bázového bloku *B3*, který je listem. Zde se hledaný záznam nachází.



Obr. 11: Příklad vyhledání záznamu s klíčem 10. Zdroj: vlastní.

Stejný koncept algoritmu vyhledávání záznamu je použit jako součást algoritmů vkládání a odebírání záznamů v B^+-stromu, protože se vždy musíme dostat do požadovaného bloku v bázovém souboru. Tam se následně vykonají další činnosti.

3.4.6. Operace vložení záznamu do B^+-stromu

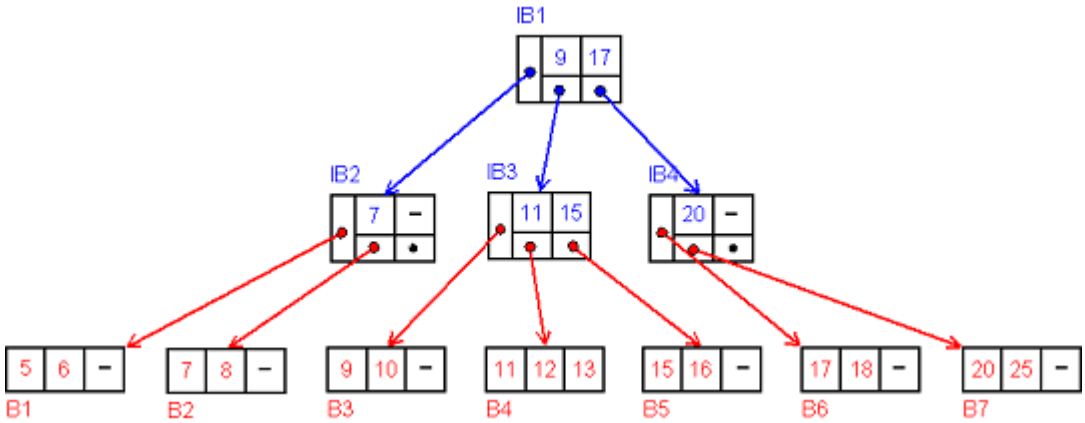
Při vkládání záznamu do B^+-stromu je nejprve nutné zpřístupnit blok v bázovém souboru pro potenciální vkládání. To lze realizovat algoritmem, který je shodný s algoritmem vyhledávání záznamu v B^+-stromu.

V bázovém souboru poté mohou nastat dvě možnosti:

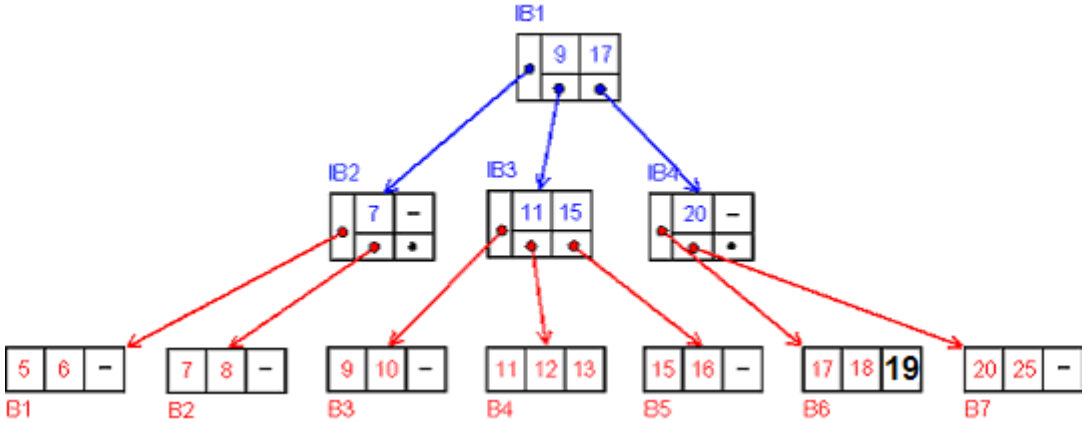
1. Blok v bázovém souboru není plný.
2. Blok v bázovém souboru je plný.

Pokud blok není plný, záznam se do něj jednoduše vloží. Kam přesně se záznam do bloku vloží záleží na dané implementaci. Záznamy mohou být v bloku uloženy v utříděném nebo neutříděném pořadí.

Na následujících dvou obrázcích je prezentován stav B^+-stromu před a po vložení záznamu s klíčem 19.



Obr. 12: Stav B^+-stromu před vložením záznamu s klíčem 19. Zdroj: vlastní.

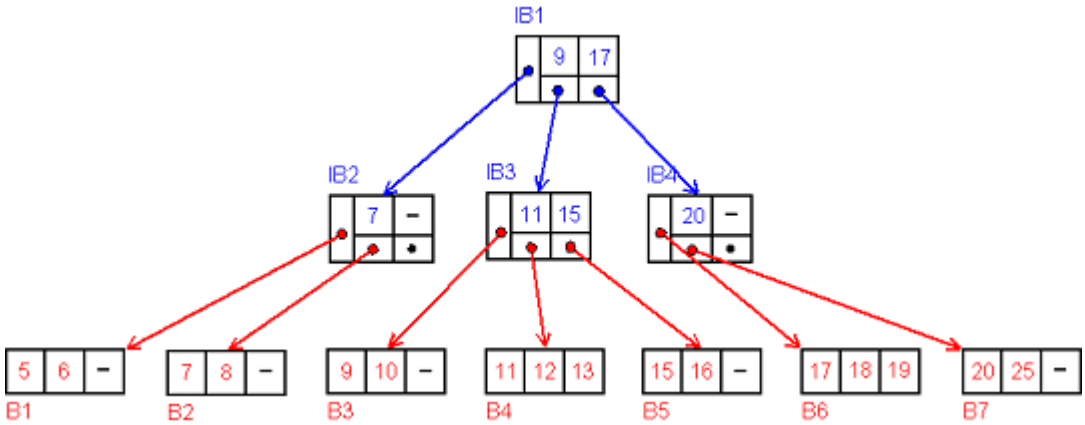


Obr. 13: Stav B^+-stromu po vložení záznamu s klíčem 19. Zdroj: vlastní.

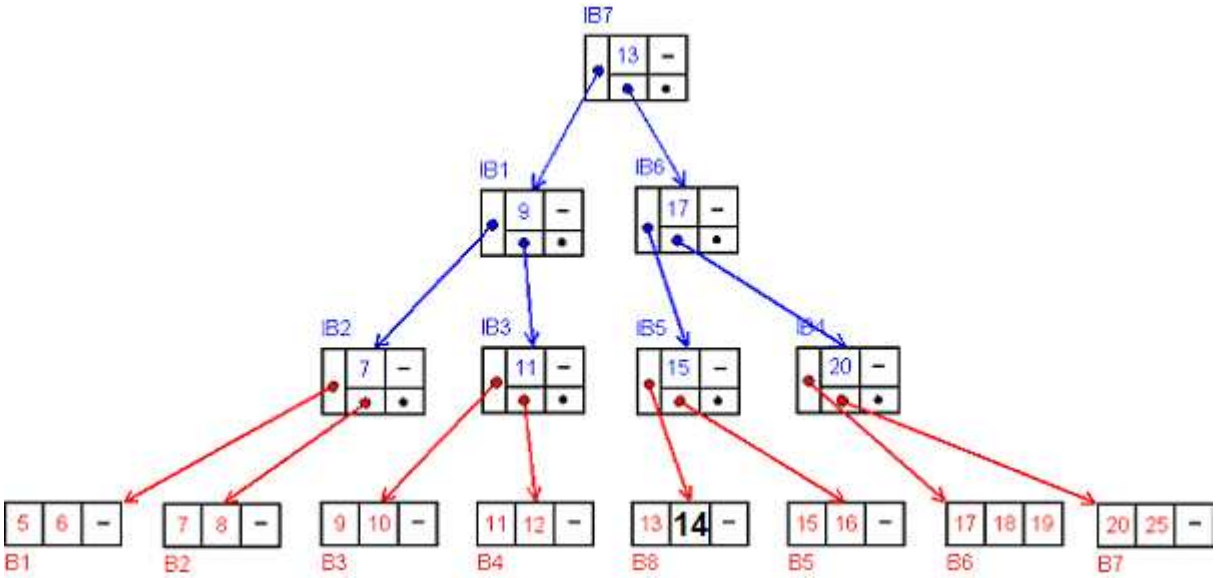
V případě, že je blok do kterého se má záznam vložit plný, nastává jiná situace – kdybychom záznam do bloku vložili, pak by došlo k *přetečení*. To znamená, že by v bloku bylo více než povolený počet záznamů. V tom případě musí dojít ke *štěpení* [1]. Jinými slovy dojde k vytvoření nového bloku. Dále vzhledem k podmínce, že v bloku B^+-stromu nemůže být méně než *k* záznamů, kde *k* je řád B^+-stromu, musí se polovina záznamů z původního bloku, který přetekl, přemístit do nově vyalokovaného bloku.

Z prostředního záznamu (mediánu) z obou bloků se dále musí vzít klíč a zkopírovat do bloku nad těmito dvěma bloky. Zde mohou opět nastat dva výše uvedené případy – buď blok je, nebo není plný – pokud je blok plný, musí se výše uvedené *štěpení* opakovat – a může se opakovat až do kořene, kde způsobí zvýšení výšky B^+-stromu.

Na následujících dvou obrázcích je prezentován stav B^+-stromu před a po vložení záznamu s klíčem 14.



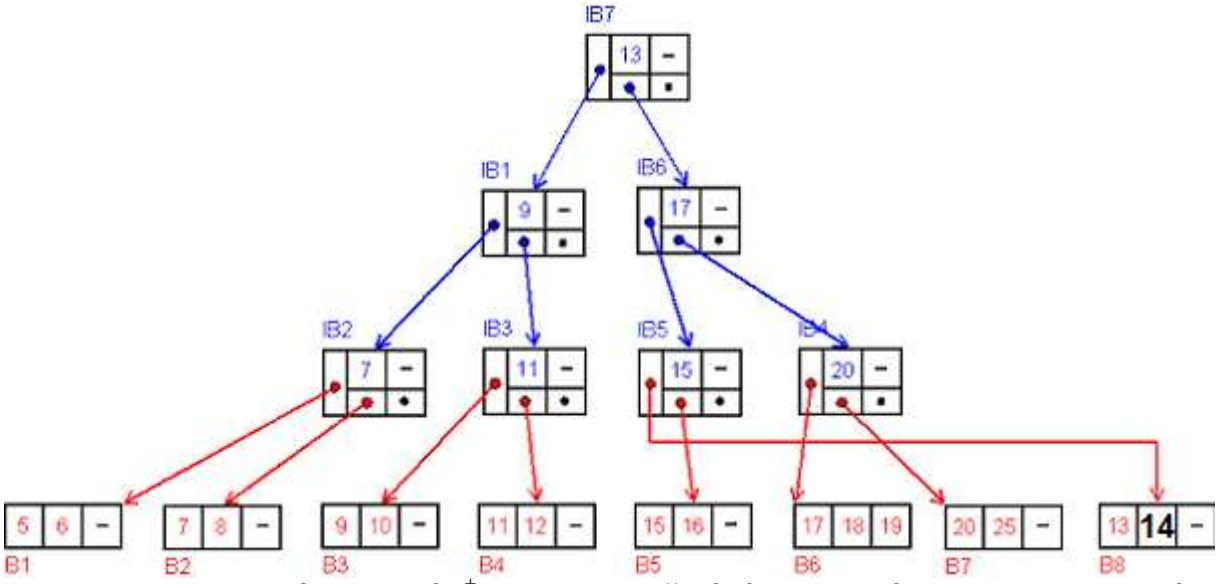
Obr. 14: Stav B^+-stromu před vložením záznamu s klíčem 14. Zdroj: vlastní.



Obr. 15: Stav B^+-stromu po vložení záznamu s klíčem 14. Zdroj: vlastní.

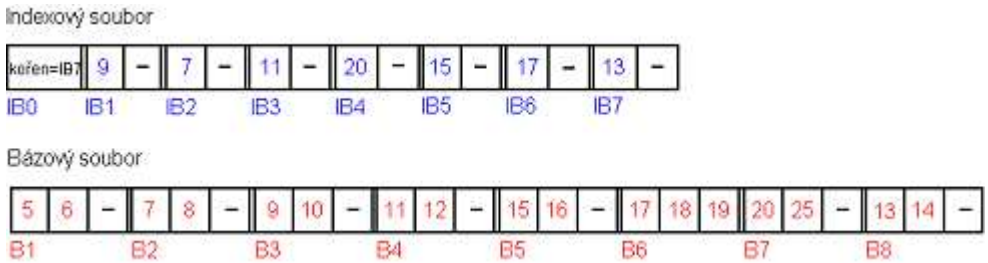
Výše uvedený obr. 14 zobrazuje „logický stav“ po vložení záznamu s klíčem 14 a následné alokaci nového bloku (*B8*). Na disku to je ale ve skutečnosti řešeno jiným způsobem. Vzhledem k tomu, že fyzicky jsou bloky v indexovém a bázovém souboru uloženy sekvenčně za sebou, připojí se nový blok na konec souboru.

Na obr. 16 je prezentováno alternativní zobrazení B^+-stromu po vložení záznamu s klíčem 14.



Obr. 16: Alternativní zobrazení B^+-stromu po vložení záznamu s klíčem 14. Zdroj: vlastní.

Na obr. 17 je znázorněn fyzický stav indexového a bázového souboru na externím paměťovém médiu.



Obr. 17: Symbolické znázornění fyzického stavu souborů na externím paměťovém médiu. Zdroj: vlastní.

Z výše uvedeného příkladu je zřejmé, že operaci vkládání do B^+-stromu je vhodné provádět pomocí rekurze. Štěpení bloků se totiž může opakovat až do kořene B^+-stromu a tím způsobit nárůst výšky stromu.

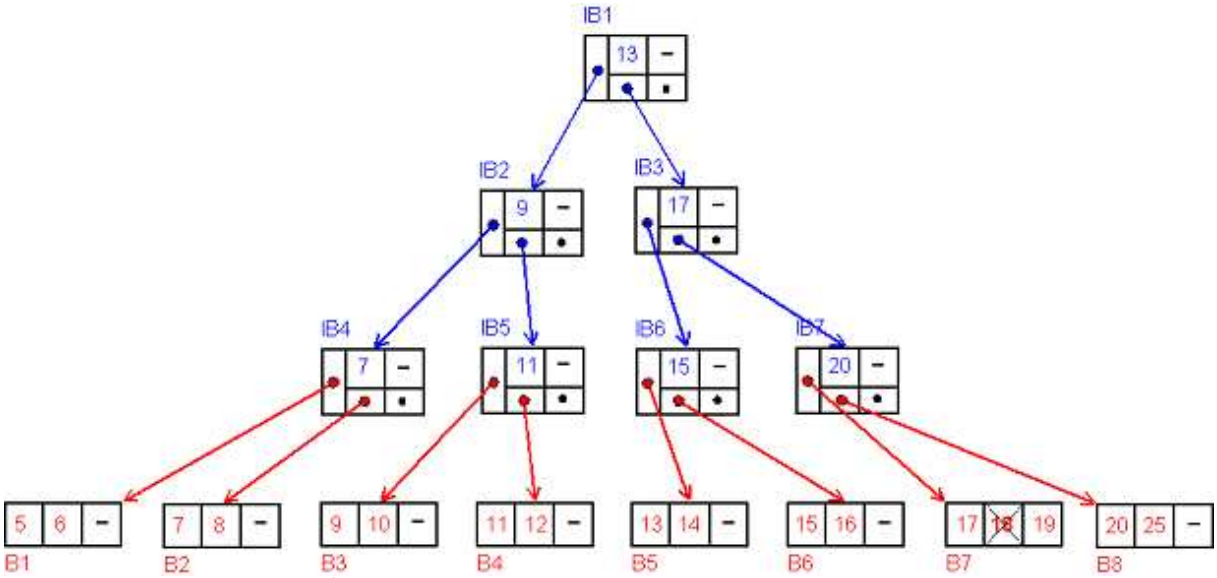
3.4.7. Operace odebrání záznamu z B^+-stromu

Analogicky jako u vkládání záznamu do B^+-stromu mohou při odebírání záznamu nastat dvě základní situace:

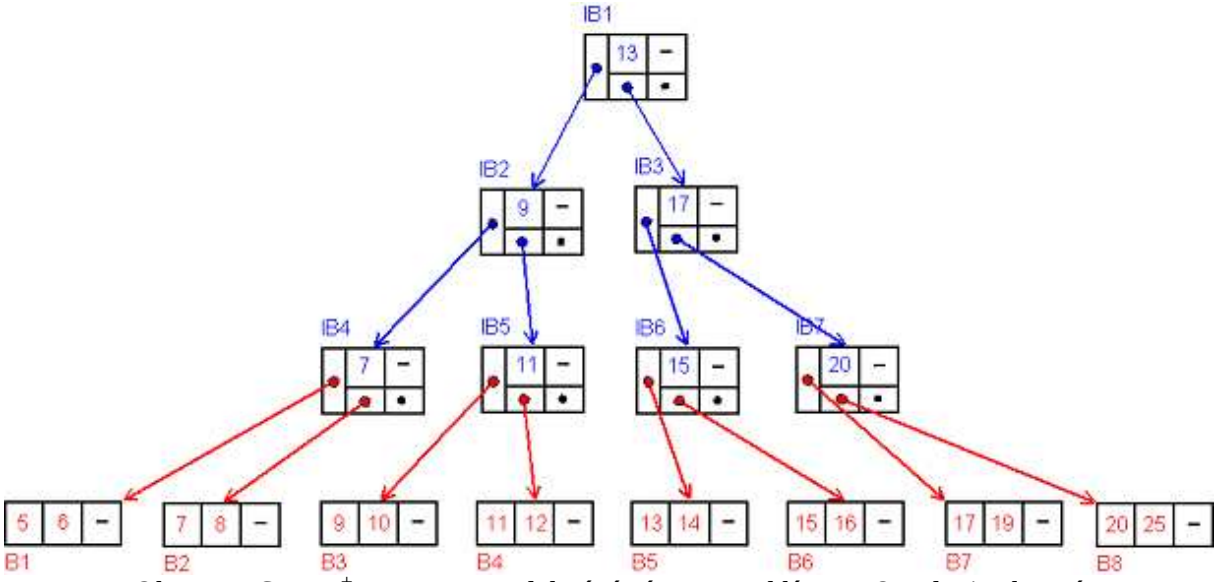
1. Blok v bázovém souboru má více než *k* záznamů, kde *k* je řád B^+-stromu.
2. Blok v bázovém souboru má právě *k* záznamů.

Tyto dvě situace jsou ilustrovány na následujících příkladech.

Na následujících dvou obrázcích je prezentován stav B^+-stromu před a po odebrání záznamu s klíčem 18.



Obr. 18: Stav B^+-stromu před odebrání záznamu s klíčem 18. Zdroj: vlastní.

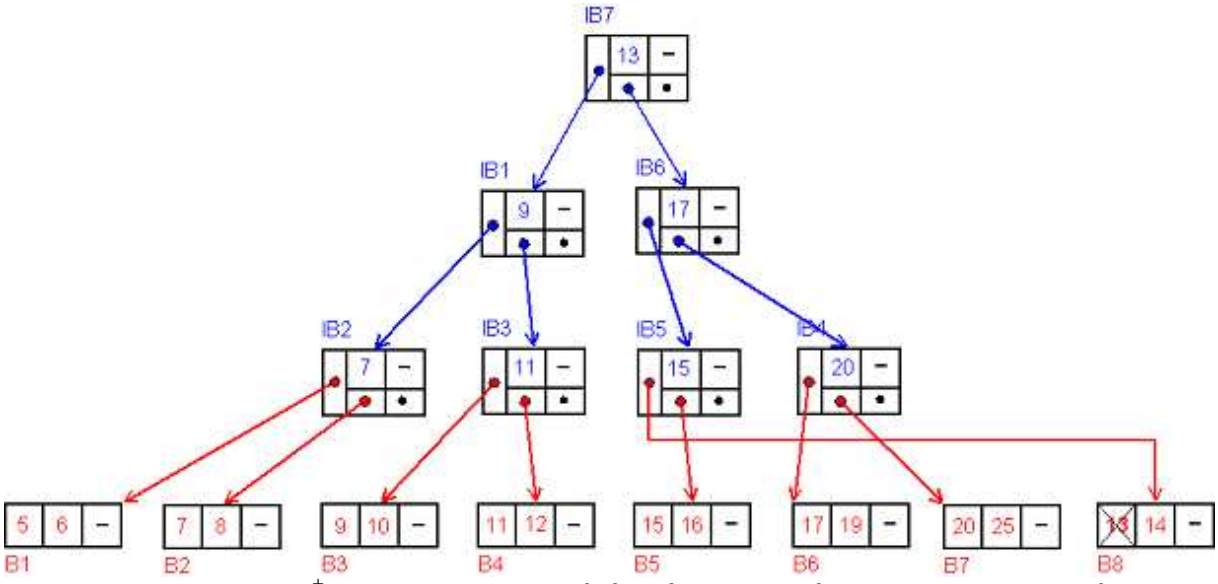


Obr. 19: Stav B^+-stromu po odebrání záznamu s klíčem 18. Zdroj: vlastní.

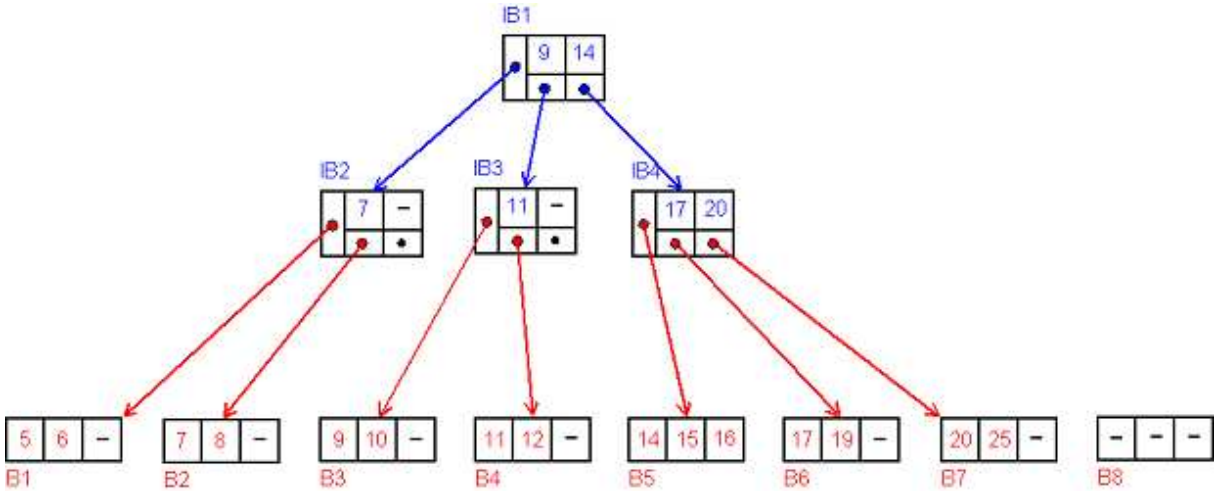
Druhý případ už není tak jednoduchý, protože kdybychom záznam odebrali, pak bychom porušili základní podmínku existence B^+-stromu – každý blok obsahuje minimálně *k* záznamů, kde *k* je řád B-stromu. Tím by v uzlu vzniklo *podtečení* [9]. Tento případ může vést až ke *slévání uzlů* [9].

Nejprve se však musí záznam, který chceme odebrat najít. Použije se zde algoritmus hledání záznamu v B^+-stromu. Dále se zjistí, jestli logicky sousedící bloky mají více než *k* záznamů. Pokud ano, pak je může blok, který by podtekl převzít. Tím by se tato situace vyřešila. Sousedící bloky ale nemusí mít záznamy navíc, v tom případě dojde ke sloučení dvou logicky sousedních bloků. Poté se musí odstranit jejich medián z nadřazeného bloku. Zde opět může dojít k podtečení a výše uvedený algoritmus se může opakovat až ke kořeni a může vést až ke snížení výšky B^+-stromu.

Na následujících dvou obrázcích je prezentován stav B^+-stromu před a po odebrání záznamu s klíčem 13.



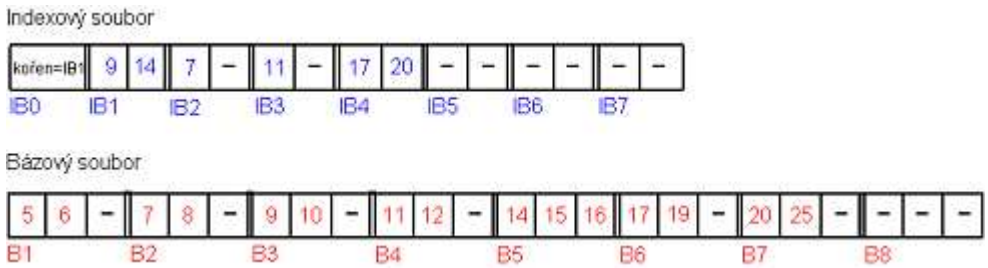
Obr. 20: Stav B^+-stromu před odebráním záznamu s klíčem 13. Zdroj: vlastní.



Obr. 21: Stav B^+-stromu po odebrání záznamu s klíčem 13. Zdroj: vlastní.

Opět tuto operaci, podobně jako operaci vložení, je vhodné implementovat pomocí rekurze.

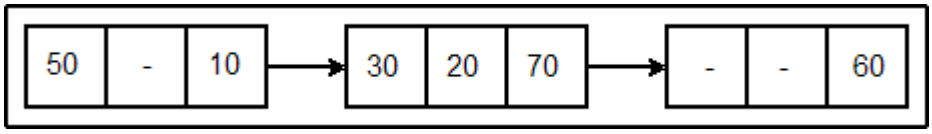
Na obr. 22 je znázorněn fyzický stav indexového a bázového souboru na externím paměťovém médiu.



Obr. 22: Symbolické znázornění fyzického stavu souborů na externím paměťovém médiu. Zdroj: vlastní.

3.5 Datová struktura Neutříděný soubor s přímým přístupem

Datovou strukturu Neutříděný soubor s přímým přístupem (halda-soubor) lze interpretovat jako implicitní neutříděnou tabulku na disku. Záznamy jsou uloženy v libovolném pořadí bez ohledu na hodnoty jejich klíčů.



Obr. 23: Příklad datové struktury Neutříděný soubor s přímým přístupem (halda-soubor). Zdroj: vlastní.

3.6 Datová struktura Soubor s úplným indexem

3.6.1 Soubor s úplným indexem

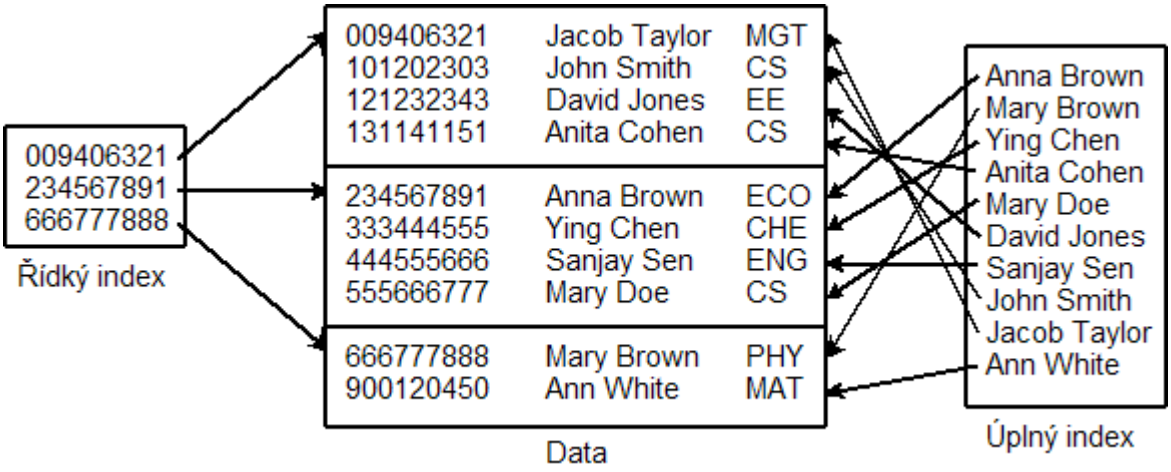
Soubory s rozptýlenými záznamy i soubory s utříděnými záznamy (indexované soubory, B-stromy) dovolují dostatečně efektivní přístup k záznamům. Přesto mají i vážné nevýhody [1]:

- problémy s implementací v případě fixovaných záznamů, které vedou ke snížení efektivnosti operací,
- nutnost periodické reorganizace, protože struktura není samoorganizující (kromě B-stromů),
- nezaplněnost bloků, která v případě B-stromu způsobuje zvětšení počtu úrovní a vede ke snížení efektivnosti operací.

Pokud se vrátíme k neutříděné organizaci (datová struktura Neutříděný soubor s přímým přístupem), zjistíme, že tato organizace nemá žádnou z uvedených nevýhod. Protože záznamy nejsou utříděné, nemusí se při operacích vkládání a rušení záznamy přemísťovat, a proto odpadají problémy s fixovanými záznamy. Záznamy se přidávají na konec souboru, proto mohou být bloky zaplněné a není důvod na reorganizaci souboru. Neutříděná organizace má samozřejmě dvě podstatné nevýhody. Kromě nemožnosti přístupu podle pořadí hodnot klíčů je to hlavně úplně neefektivní přistup podle klíče (realizuje se sekvenčním vyhledáváním) [1].

Existuje organizace, která dovoluje poměrně dobře eliminovat tyto nevýhody. Jejím principem je vybudování tzv. *úplného indexu (hustého indexu)* [1], který slouží jako přístupový mechanismus do neutříděného souboru. Úplný index můžeme implementovat libovolným známým způsobem, za nejefektivnější se považuje implementace B^+-stromem. Úplný index má záznamy tzv. fixované. Nefixované záznamy jsou např. v datové struktuře B-strom. Taková struktura obsahuje tzv. *řídký index* [1]. Princip úplného indexu dovoluje rychlý přístup do souboru i podle více klíčů [1].

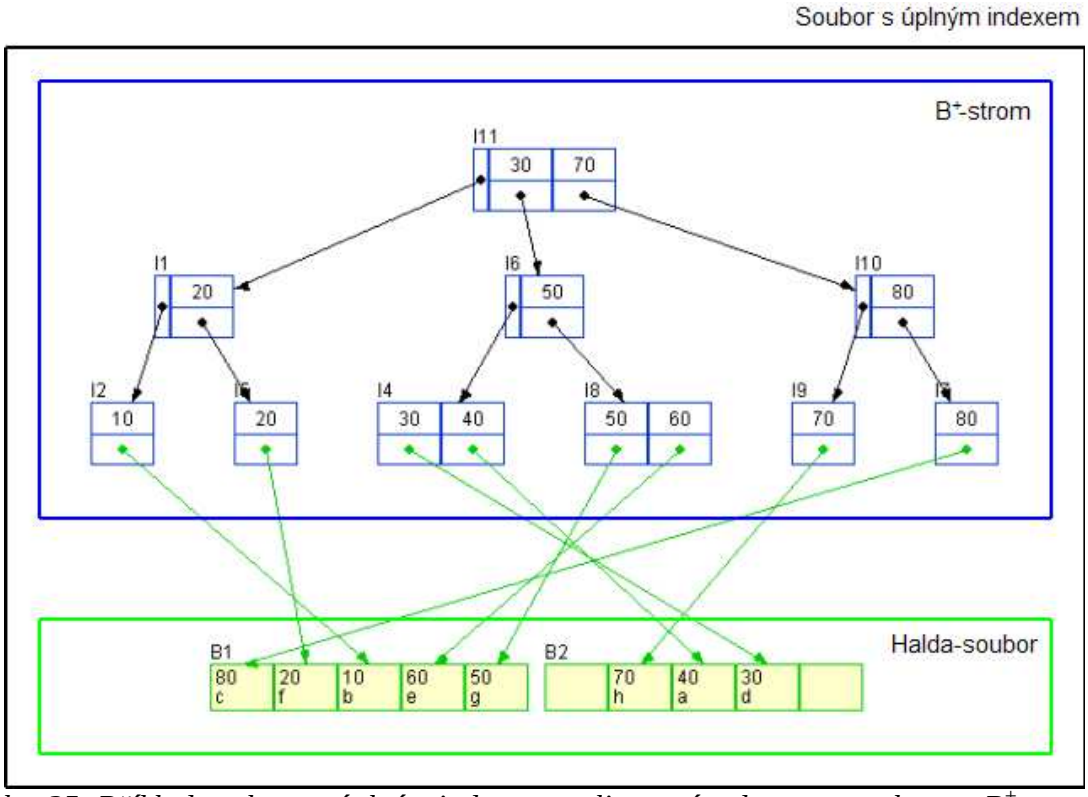
Na následujícím obrázku je ilustrován princip fixace záznamů. Vpravo na obrázku je znázorněn přístup pomocí *řídkého (sparse) indexu* a vlevo je ilustrován přístup k datům pomocí *úplného (dense) indexu*.



Obr. 24: Ilustrace přístupu k datům pomocí řídkého a úplného indexu. Zdroj: [30]

3.6.2 Struktura Souboru s úplným indexem

Soubor s úplným indexem se skládá ze dvou souborů. Indexový soubor obsahuje přístupový index, zatímco bázový soubor obsahuje data. V další části textu se budeme zabývat souborem s úplným indexem realizovaným datovou strukturou B^+-strom. V listech B^+-stromu jsou klíče a adresy na kompletní záznamy v bázovém souboru. Bázový soubor je tvořen datovou strukturou halda-soubor, ve které jsou uloženy kompletní záznamy tabulky. Na obrázku 26 je zobrazena datová struktura Soubor s úplným indexem a dvě datové struktury ze kterých se skládá.



Obr. 25: Příklad souboru s úplným indexem realizovaným datovou strukturou B^+-strom. Zdroj: vlastní.

3.6.3 Operace hledání záznamu v Souboru s úplným indexem

Vzhledem k tomu, že se Soubor s úplným indexem fyzicky skládá ze dvou datových struktur, obsahuje i operace hledání záznamu 2 základní kroky. Tyto kroky jsou následující:

1. Najde se hledaný klíč v listu B^+-stromu (viz. kapitola 3.4.5).
2. Zpřístupní se příslušný záznam z bázového souboru.

V případě, že se záznam s hledaným klíčem ve stromu nevyskytuje, pak se samozřejmě bod 2 nevykoná.

3.6.4 Operace vložení záznamu do Souboru s úplným indexem

Operace vložení záznamu do Souboru s úplným indexem obsahuje 3 kroky. Tyto kroky jsou následující:

1. Získání libovolné volné adresy v bázovém souboru pro vložení záznamu.
2. Vložení klíče do B^+-stromu (s případnou reorganizací) - viz. kapitola 3.4.6.
3. Vložení záznamu do bázového souboru na adresu zjištěnou v kroku 1.

V případě, že se záznam s klíčem ve stromu již vyskytuje, bod 3 se nevykoná.

3.6.5 Operace odebrání záznamu ze Souboru s úplným indexem

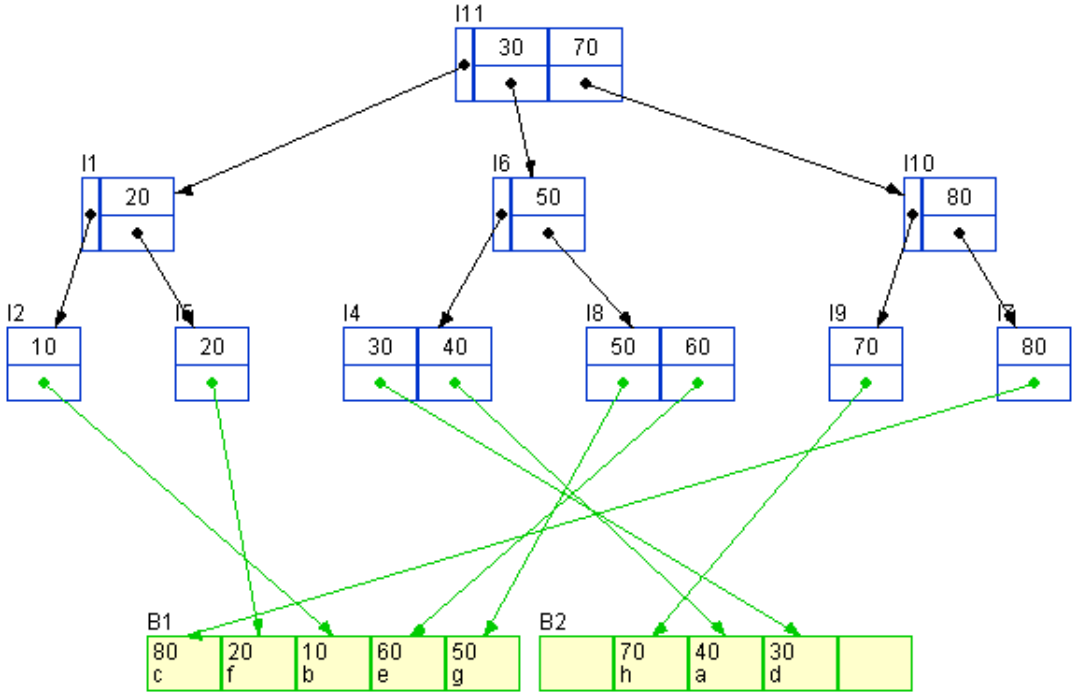
Operace odebrání záznamu ze Souboru s úplným indexem obsahuje 3 kroky.

1. Nalezení klíče v listu B^+-stromu (viz. kapitola 3.4.5).
2. Odebrání záznamu z bázového souboru.
3. Odebrání klíče v B^+-stromu (s případnou reorganizací) - viz. kapitola 3.4.7.

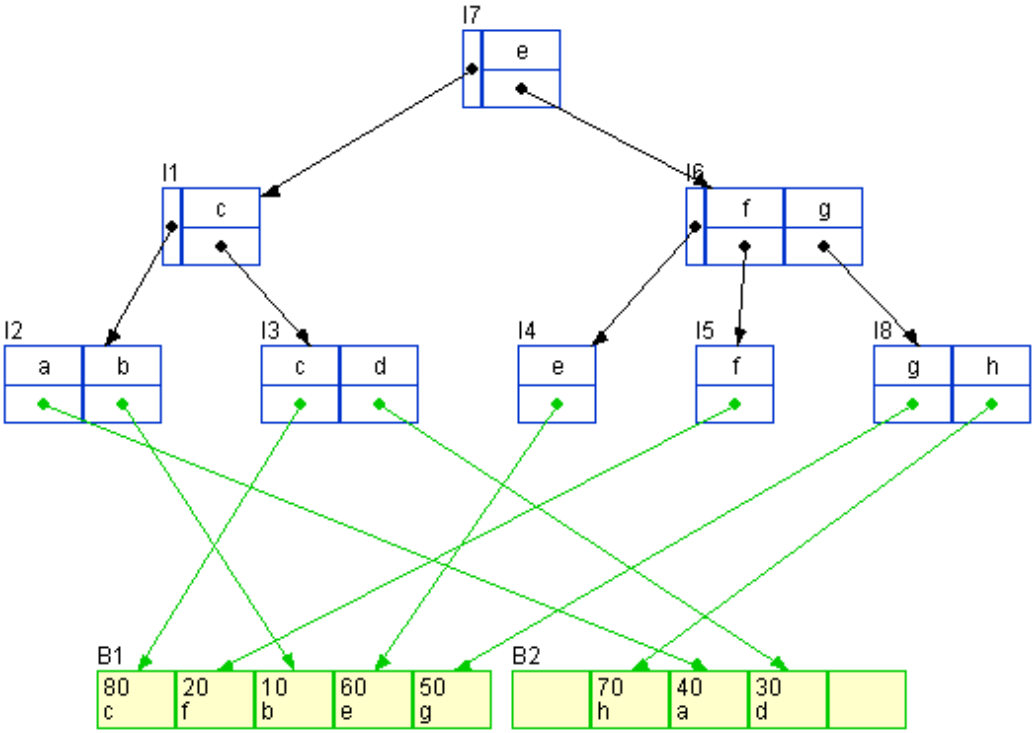
V případě, že se záznam s klíčem ve stromu nevyskytuje, pak se body 2 a 3 nevykonají.

3.6.6 Vytvoření dalšího přístupového indexu

Z důvodu neutříděnosti bázového souboru je možné vytvořit nad jedním bázovým souborem více přístupových indexů podle různých klíčů. Na obrázku 26 je zobrazen Soubor s úplným indexem, který obsahuje B^+-strom s klíčem typu *Integer*. Na obrázku 27 je zachycen Soubor s úplným indexem, který obsahuje stejný halda-soubor, jako je na obrázku 26, ale s jiným přístupovým indexem. B^+-strom v druhém případě obsahuje klíče typu *String*.

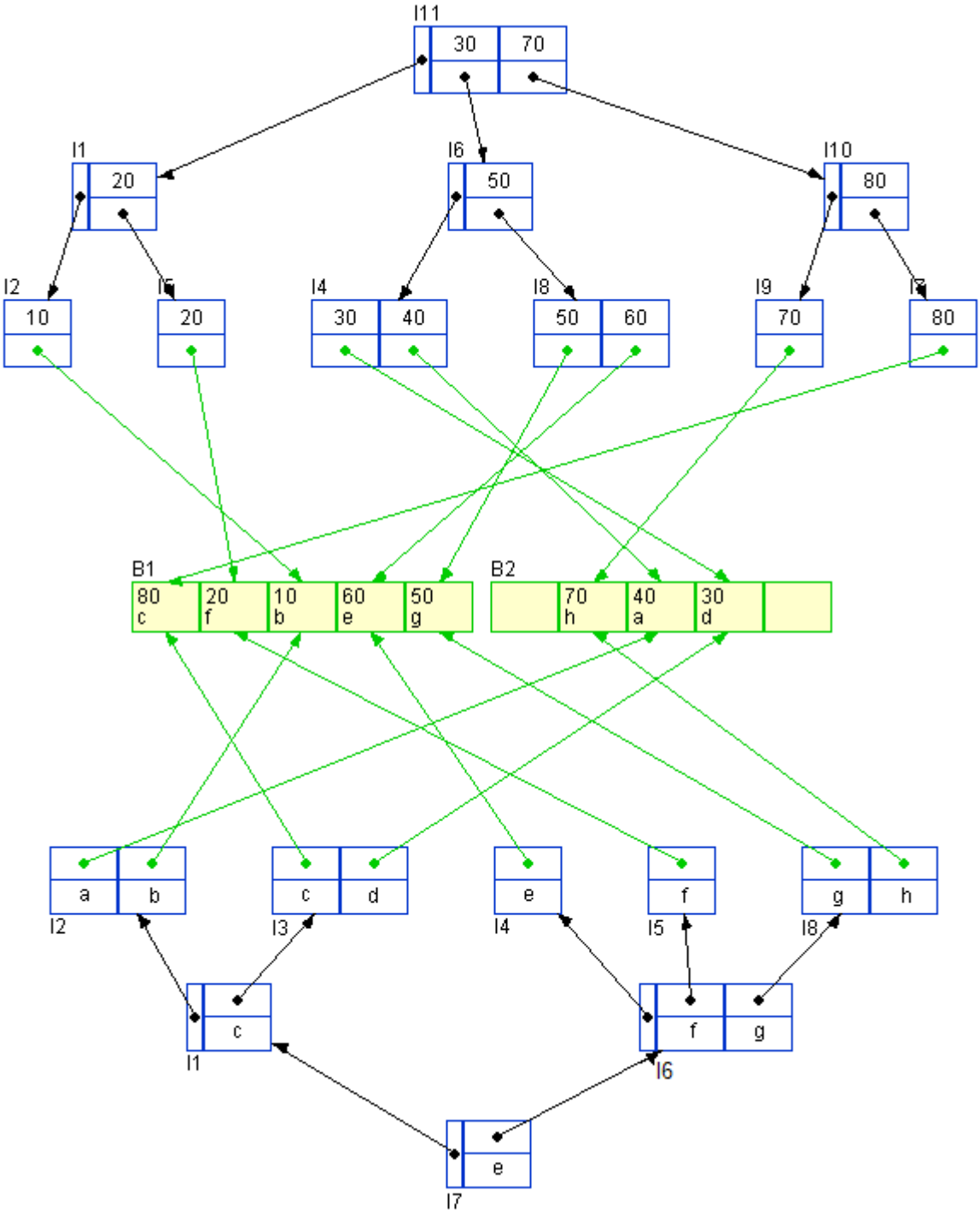


Obr. 26: B^+-strom s klíčem typu Integer. Zdroj: vlastní.



Obr. 27: B^+-strom s klíčem typu String. Zdroj: vlastní.

Na obrázku 28 je ilustrace dvou přístupových indexů nad jedním bázovým souborem. Tyto indexy již byly prezentovány v oddělené podobě v předcházejících dvou obrázcích.



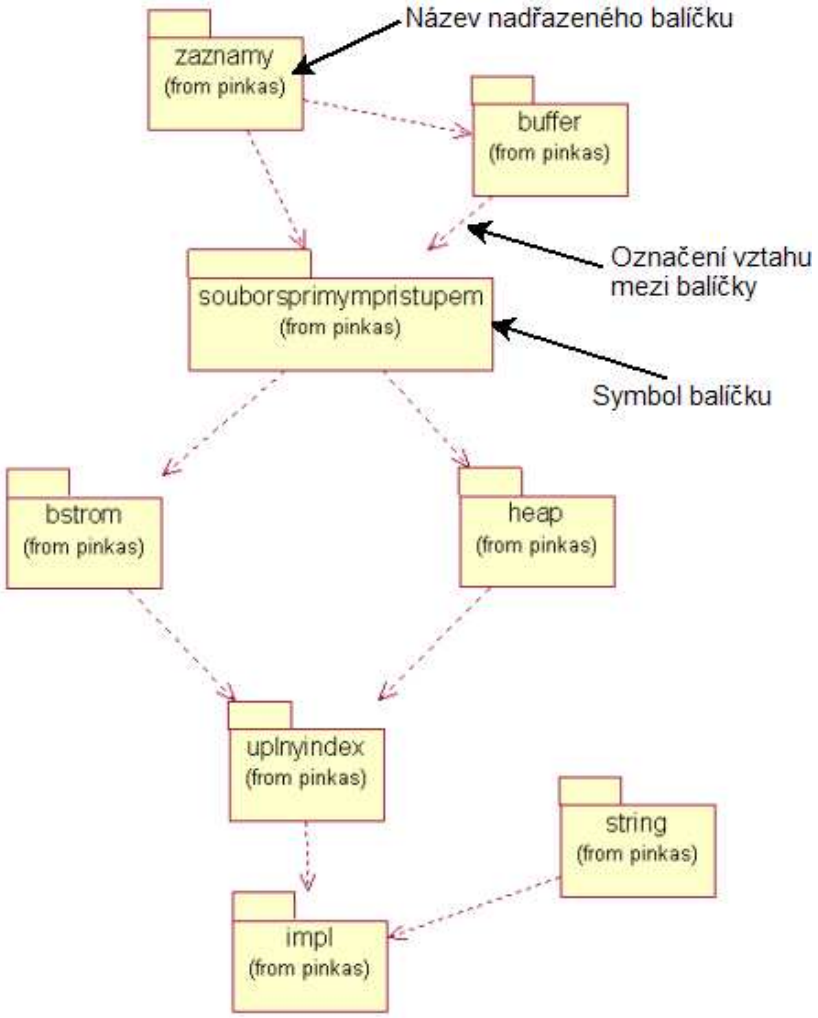
Obr. 28: Ilustrace dvou indexů nad jedním bázovým souborem. Zdroj: vlastní.

4. Vlastní implementace

4.1 Implementace Souboru s úplným indexem

V této kapitole je přehled implementace datové struktury Soubor s úplným indexem. Tato struktura byla implementována ve vyšším programovacím jazyku Java. V dalším textu bude pomocí jazyka UML tato implementace vysvětlena. Pro tvorbu UML diagramů byla použita aplikace Rational Rose.

Na následujícím obrázku je přehled nadefinovaných balíčků, ze kterých se skládá implementace Souboru s úplným indexem. Balíčky v jazyce Java slouží pro definování jmenných prostorů (namespace). Mezi balíčky jsou také zobrazeny jejich vzájemné závislosti. Všechny balíčky jsou v balíčku *info.pinkas* podle konvence tvorby balíčků v jazyku Java. Skutečnost že třída (příp. balíček) je součástí nějakého balíčku se v následujícím zobrazí tak, že se v hlavičce třídy (příp. balíčku) napíše do závorek *from* a název balíčku, např. *(from pinkas)*

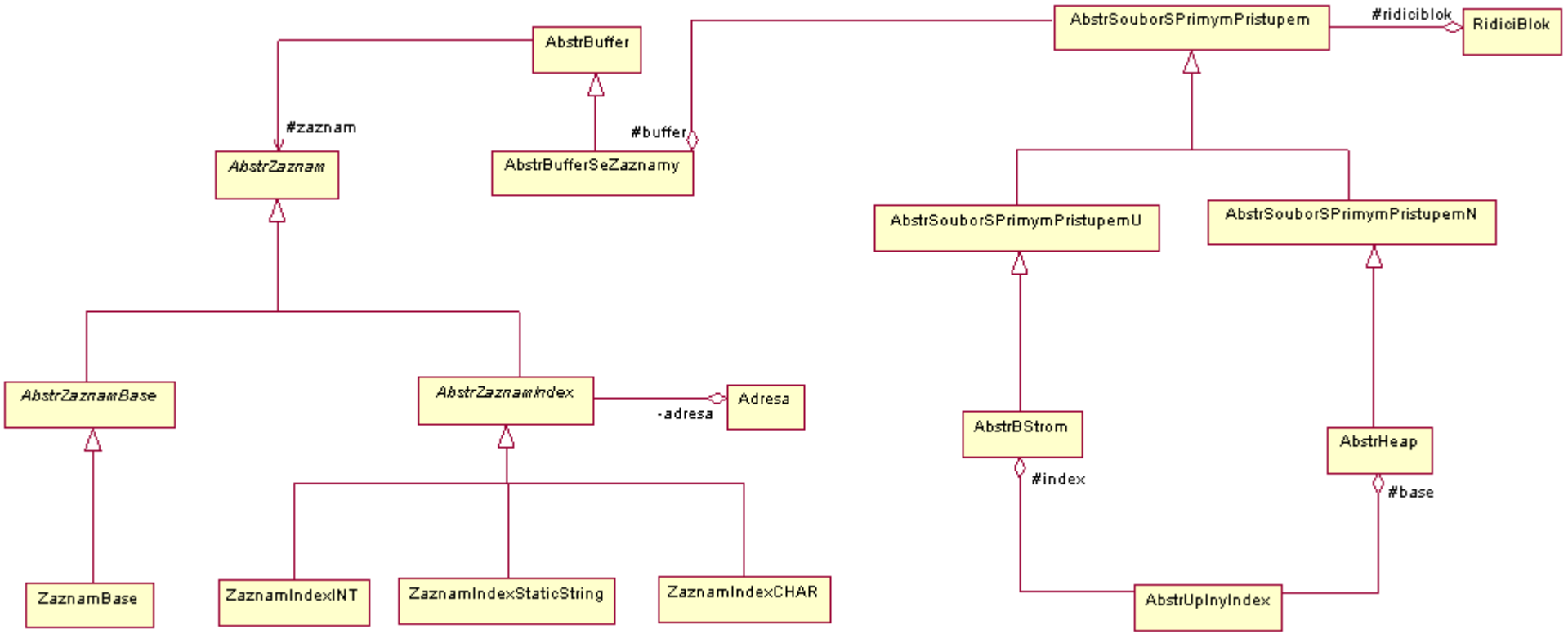


Obr. 29: Přehled nadefinovaných balíčků. Zdroj: vlastní.

Na obrázku 30 je znázorněn celý diagram tříd Souboru s úplným indexem. V dalším textu se bude používat pro Soubor s úplným indexem též označení *Úplný index*. Jednotlivé části diagramu jsou podrobně rozebrány v následujících podkapitolách. Všechny abstraktní třídy mají předponu *Abstr*. Třídy na implementační úrovni tuto předponu nemají.

V diagramu je použita následující konvence značení viditelnosti atributů:

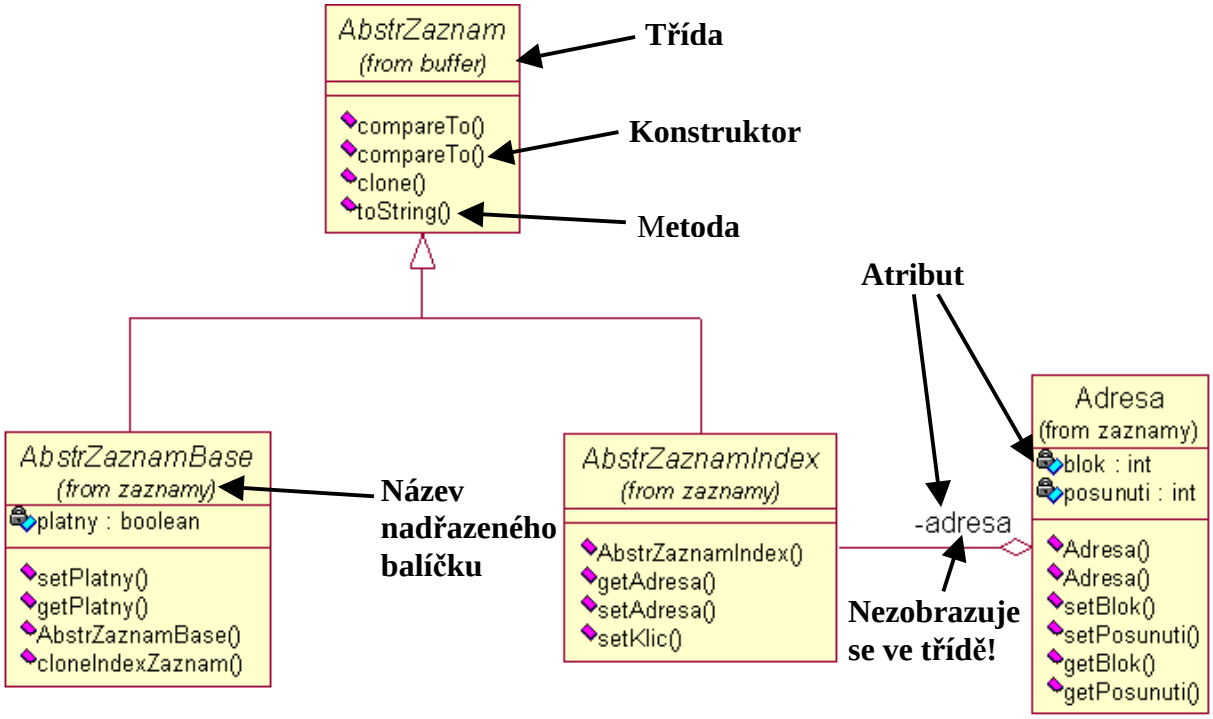
Znak	**Viditelnost**
-	Private
#	Protected
+	Public



Obr. 30: Kompletní diagram tříd Souboru s úplným indexem. Zdroj: vlastní.

4.1.1 Záznamy

Na obrázku 31 jsou podrobně zobrazeny na abstraktní úrovni třídy záznamů. Všechny záznamy mají společného předka, jímž je třída *AbstrZaznam*.



Obr. 31: Záznamy na abstraktní úrovni. Zdroj: vlastní.

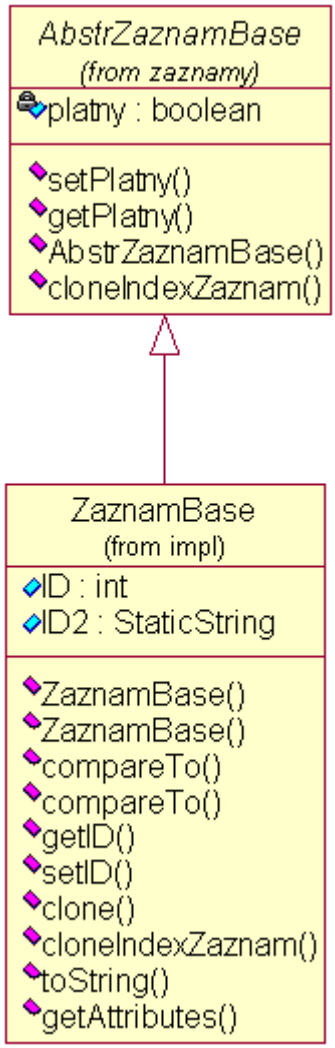
Třída *AbstrZaznamBase* (záznam z bázového souboru) obsahuje atribut *platny* typu *Boolean*. Tento atribut určuje, jestli je příslušný záznam platný, či nikoliv. Při vkládání nového záznamu je hodnota tohoto atributu nastavena na *true*, při odebrání záznamu (a tím jeho zrušení) je nastavena na *false*.

Třída *AbstrZaznamIndex* (záznam z indexového souboru) obsahuje atribut *adresa*, který je typu *Adresa* a tím může „ukazovat“ na blok do bázového souboru. Tento atribut plní tuto funkci v případě, že příslušný záznam je v listu B^+-stromu. V tom případě určuje blok v bázovém souboru a posunutí v tomto bloku. Tímto způsobem se zpřístupní příslušný záznam v bázovém souboru. V případě, že se záznam nenachází v bloku který je listem, se atribut *adresa* použije jako ukazatel na uzel ve stromu.

Na obrázku 32 je podrobněji zobrazena třída *ZaznamBase* (záznam z bázového souboru). Objekty této třídy se nacházejí v bázovém souboru a obsahují plná data. Pro demonstrační účely v tomto případě obsahuje uživatelská třída *ZaznamBase* pouze dva atributy:

- Atribut **ID** typu *Integer*.
- Atribut **ID2** typu *StaticString*.

V typické reálné aplikaci by ale byl počet těchto atributů pravděpodobně větší. Při přidávání dalších atributů je ale důležité si uvědomit, že atribut musí mít stanovenu maximální velikost. Například typ *String* v programovacím jazyku Java může mít libovolnou délku. Proto byla vytvořena třída *StaticString*, jejíž řetězec může mít maximálně 255 znaků. Toto omezení lze samozřejmě zvýšit nebo snížit podle potřeb aplikace. Toto omezení maximální délky atributu je kvůli tomu, že každý záznam B^+-stromu musí mít stejnou velikost.



Obr. 32: Třídy záznamů z bázového souboru. Zdroj: vlastní.

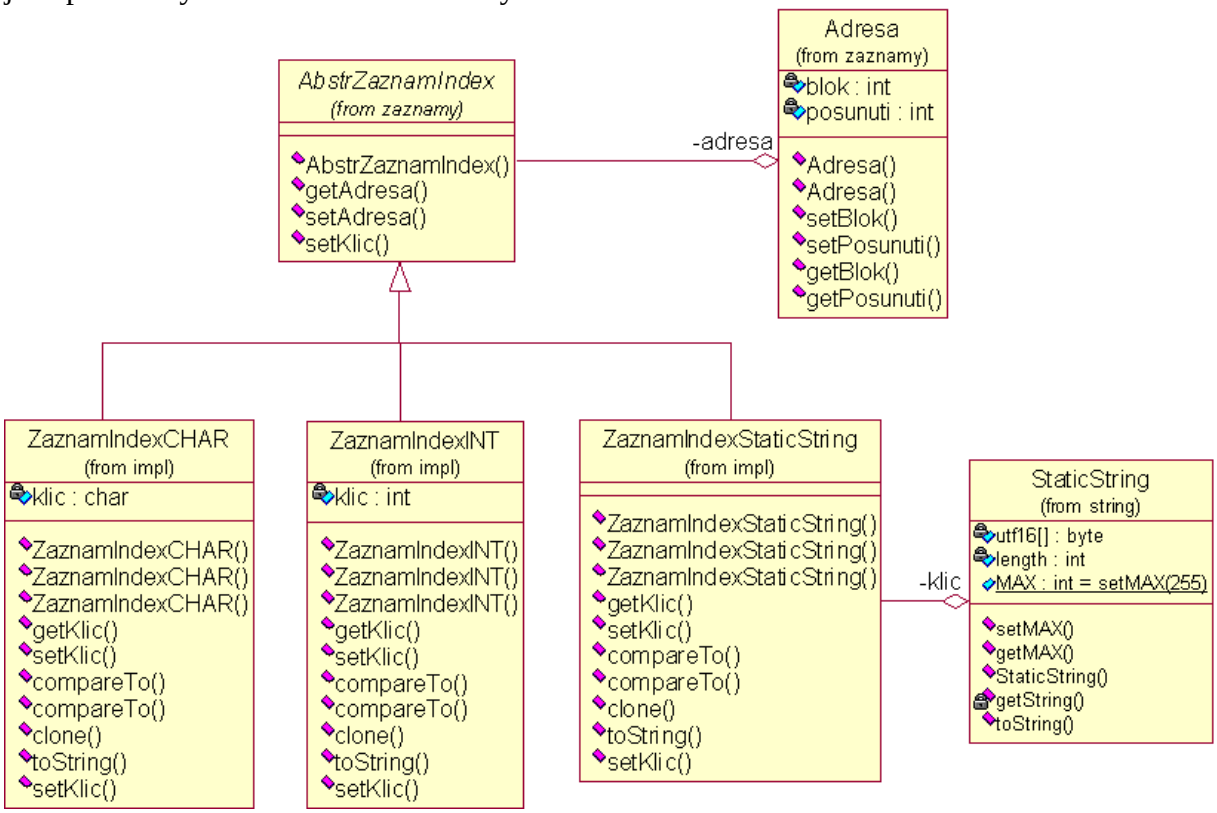
Na obrázku 33 je zobrazena třída *AbstrZaznamIndex* (záznam z indexového souboru) a třídy od ní odvozené.

Každý datový typ, který může být klíčem v záznamu bázového souboru, musí být odvozen od třídy *AbstrZaznamIndex*. Je vhodné, aby názvy takových tříd splňovaly konvenci, jaká je zřejmá z obrázku 33. Každou novou třídu je vhodné pojmenovat *ZaznamIndexXXX*, kde *XXX* je název příslušného datového typu. V případě implementace složeného klíče by bylo vhodné třídu, která by toto umožňovala nazvat např. *ZaznamIndexSlozenyKlic.*

Dále je nutné, aby každá třída měla stejné metody jako zde uvedené třídy. Jedná se o metody:

- **Konstruktor** – vytvoří se nový objekt.
- Metody **getKlic/setKlic** – získá/nastaví se klíč.
- Metody **compareTo** – slouží pro porovnání s dalším objektem podle klíče.
- Metoda **clone** – slouží pro vytvoření kopie objektu.
- Metoda **toString** – slouží pro textový výpis klíče (např. pro výpis do *JTree*).

Některé z metod se navíc mohou volat s jinými vstupními parametry, proto je 3x přetížený *konstruktor* a 2x jsou přetíženy metody *compareTo()* a *setKlic().* Pro zachování přehlednosti jsou parametry metod na obrázcích skryté.



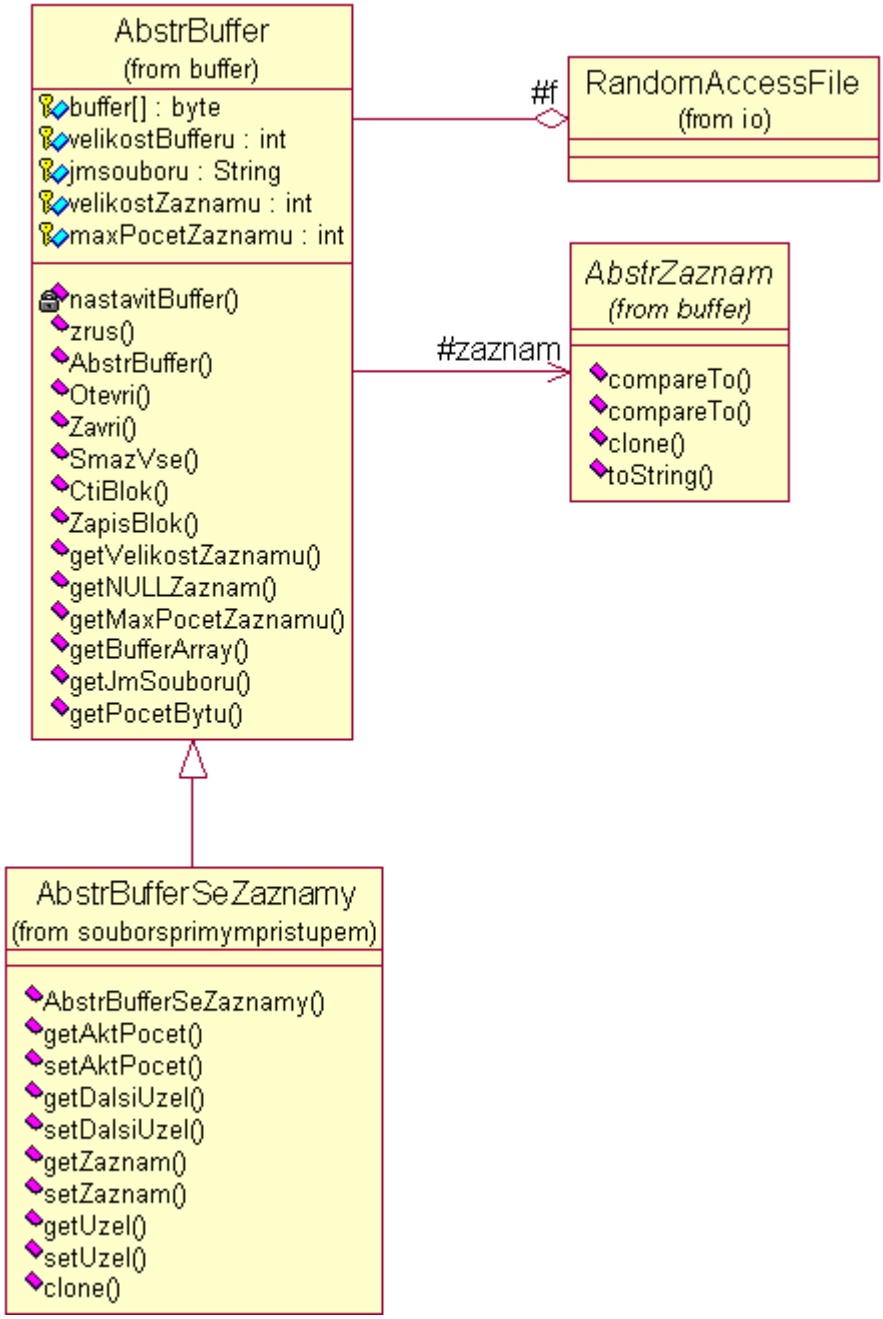
Obr. 33: Třídy záznamů z indexového souboru. Zdroj: vlastní.

4.1.2 Buffer

Na obrázku 34 je zobrazen buffer. Buffer je rozdělen do tříd *AbstrBuffer* a *AbstrBufferSeZaznamy*. Rzdělení bufferu do dvou tříd je z důvodu větší přehlednosti. Zatímco třída *AbstrBuffer* obsahuje metody pro práci s bloky jako např. metodu *CtiBlok()* pro čtení bloku ze souboru a metodu *ZapisBlok()* pro zapsání bloku do souboru, třída *AbstrBufferSeZaznamy* obsahuje metody pro práci s jednotlivými záznamy v bufferu.

Buffer se skládá z následujících atributů:

- **buffer[]** – v tomto poli jsou fyzicky uložena data bufferu,
- **polozka** – objekt stejného typu jako záznamy v bufferu,
- **velikostBufferu** – velikost bufferu v bytech,
- **f** –soubor s přímým přístupem,
- **jmsouboru** – jméno souboru, který je k příslušnému bufferu přidružen,
- **velikostZaznamu** – velikost jednoho záznamu v bufferu,
- **pocetZaznamu** – počet záznamů v bufferu.



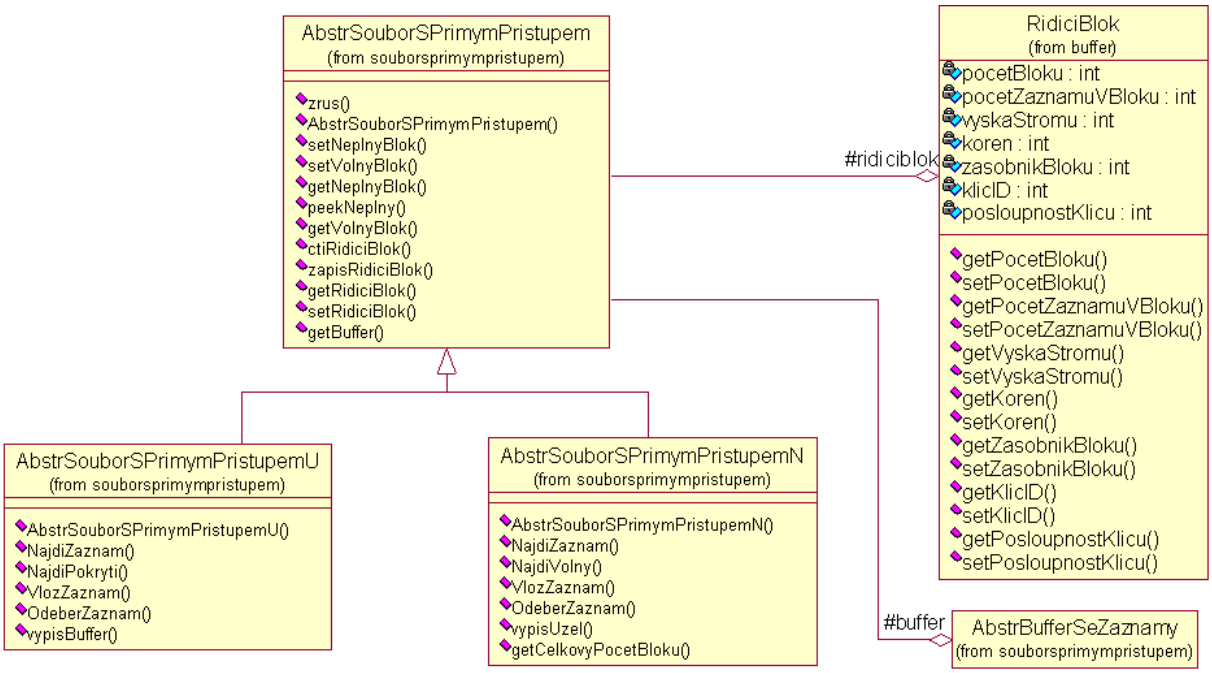
Obr. 34: Třídy, které tvoří buffer. Zdroj: vlastní.

4.1.3 Soubor s přímým přístupem

Na obrázku 35 je zobrazena třída *AbstrSouborSPrimymPristupem* a dvě třídy, které jsou od ní odvozené: *AbstrSouborSPrimymPristupemU* a *AbstrSouborSPrimymPristupemN*.

Třída *AbstrSouborSPrimymPristupem* se skládá ze dvou atributů:

- **ridiciblok** – Řídící blok souboru s přímým přístupem,
- **buffer** – Buffer typu *AbstrBufferSeZaznamy*.



Obr. 35: Třídy, které tvoří Soubor s přímým přístupem. Zdroj: vlastní.

Řídící blok je uložen v prvním bloku souboru a obsahuje následující atributy:

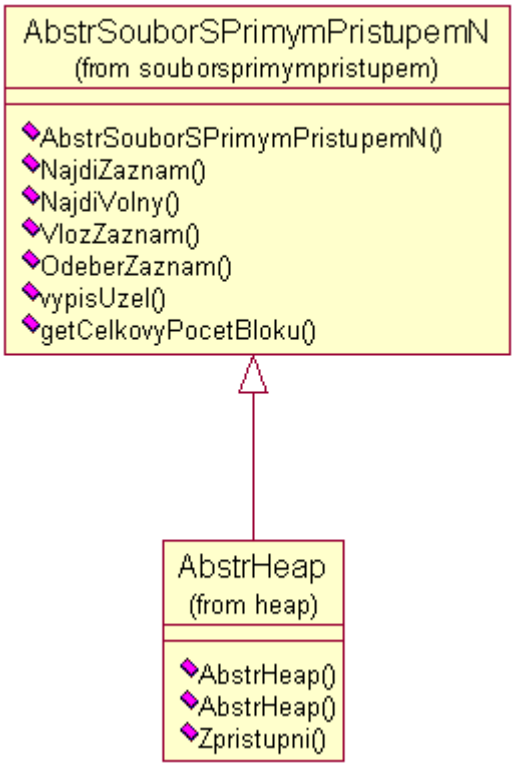
- **pocetBloku** – Aktuální počet bloků v souboru.
- **pocetPrvkuVBloku** – Maximální počet záznamů v jednom bloku.
- **vyskaStromu** – V případě, že soubor je index, pak je zde uvedena aktuální výška stromu.
- **koren** – V případě, že soubor je index, pak je zde uvedeno číslo kořenového bloku.
- **zasobnikBloku** – Začátek zásobníku volných bloků. Zásobník je implementačně tvořen logickým seznamem.
- **klicID** – V případě, že se soubor je index, pak je zde uveden index atributu v bázovém souboru, který je klíčem.
- **posloupnostKlicu** – Začátek posloupnosti zřetězení bloků. V případě že se jedná o index, pak je tato posloupnost bloků uspořádána vzestupně podle hodnot klíčů obsažených v blocích.

Třída *AbstrSouborSPrimymPristupemU* je takový soubor s přímým přístupem, v jehož blocích jsou záznamy uspořádány podle klíče vzestupně. Pokud buffer obsahuje aktuálně *k* záznamů, potom jsou v bufferu obsazeny záznamy 0 až *k* – 1. Ostatní záznamy mají nedefinovanou hodnotu. Tato třída je předchůdcem třídy *AbstrBStrom*.

AbstrSouborSPrimymPristupemN je takový soubor s přímým přístupem, v jehož blocích jsou záznamy neuspořádány. Tato třída je předchůdcem třídy *AbstrHeap*. Jestli je záznam platný nebo není, se určuje podle atributu *platny* ve třídě *AbstrZaznamBase*. Tato třída je abstraktním neuspořádaným souborem s přímým přístupem (halda-soubor).

4.1.4 Halda-soubor

Na obrázku 36 je zobrazena třída *AbstrHeap*. Tato třída dědí naprostou většinu operací pro práci se záznamy z třídy *AbstrSouborSPrimymPristupemN*. Přestože již třída *AbstrSouborSPrimymPristupemN* je datovým typem halda-soubor, bylo pro zvýšení abstraktnosti vhodné rozdělit třídy *AbstrSouborSPrimymPristupemN* a *AbstrHeap*, obdobně jako byl rozdělen buffer na dvě třídy. Třída *AbstrHeap* obsahuje předefinovaný konstruktor a metodu *Zpristupni()*. Tyto metody byly implementovány z důvodu efektivnější práce Úplného indexu s haldou a nejsou v ADT Halda-soubor nezbytné, tudíž bylo vhodné vytvořit třídu *AbstrHeap*, která tyto metody obsahuje.



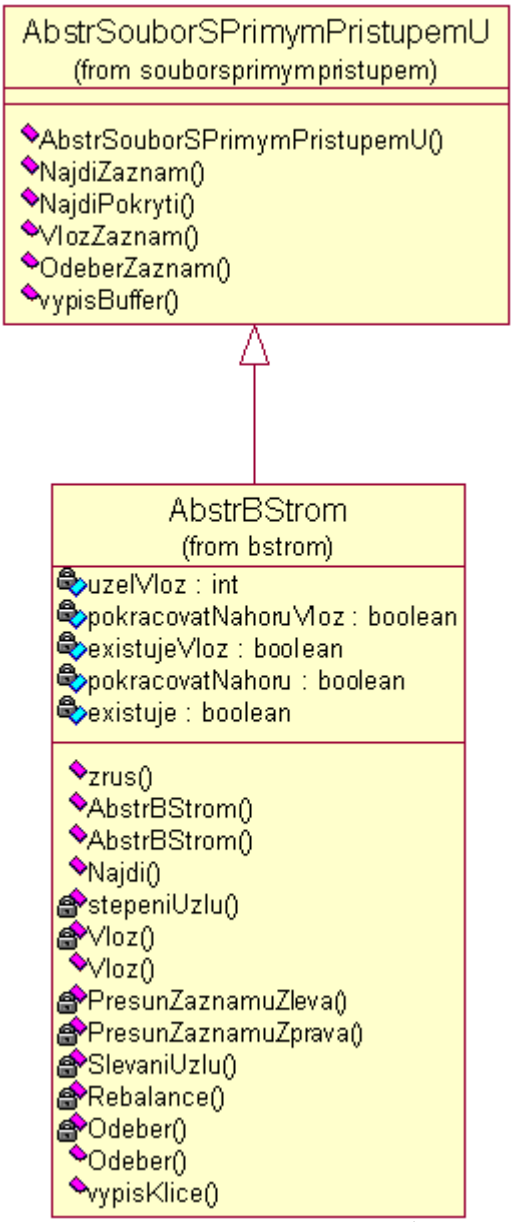
Obr. 36: Třídy, které tvoří Halda-soubor. Zdroj: vlastní.

4.1.5 B$^+$-strom

Na obrázku 37 je třída *AbstrBStrom*. . V této třídě jsou standardní metody

- **Vloz()** – Vložení záznamu do B$^+$-stromu.
- **Odeber()** – Odebrání záznamu z B$^+$-stromu.
- **Najdi()** – Nalezení záznamu v B$^+$-stromu a jeho zpřístupnění.

Algoritmy těchto metod již byly popsány v předcházejícím textu.



Obr. 37: Třídy, které tvoří datovou strukturu B$^+$-strom. Zdroj: vlastní.

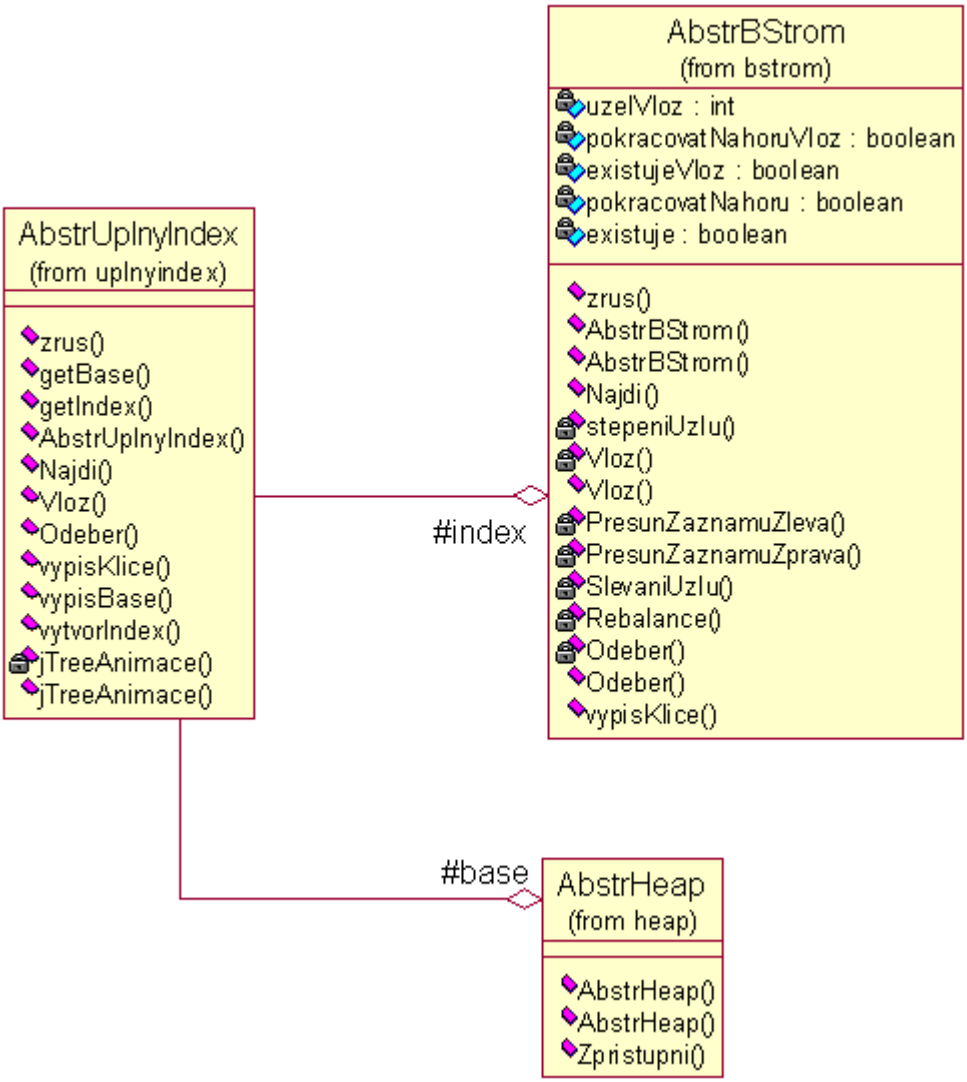
4.1.6 Úplný index

Na obrázku 38 je znázorněna třída *AbstrUplnyIndex* (datová struktura Soubor s úplným indexem). Tato třída je složena ze dvou dílčích struktur:

- **index** – Indexový soubor, který je tvořen datovou strukturou B$^+$-strom.
- **base** – Bázový soubor, který je tvořen datovou strukturou Halda.

Třída *AbstrUplnyIndex* obsahuje následující operace pro práci s Úplným indexem:

- **Najdi()** – Vyhledá záznam v Úplném indexu.
- **Vloz()** – Vloží záznam do Úplného indexu.
- **Odeber()** – Odebere záznam z Úplného indexu.
- **VypisKlice()** – Vypíše všechny klíče vzestupně.
- **VypisBase()** – Vypíše všechny záznamy z bázového souboru.
- **VytvorIndex()** – Vytvoří druhý přístupový index nad bázovým souborem.
- **JTreeAnimace()** – Exportuje B$^+$-strom do vizuální komponenty *JTree*.

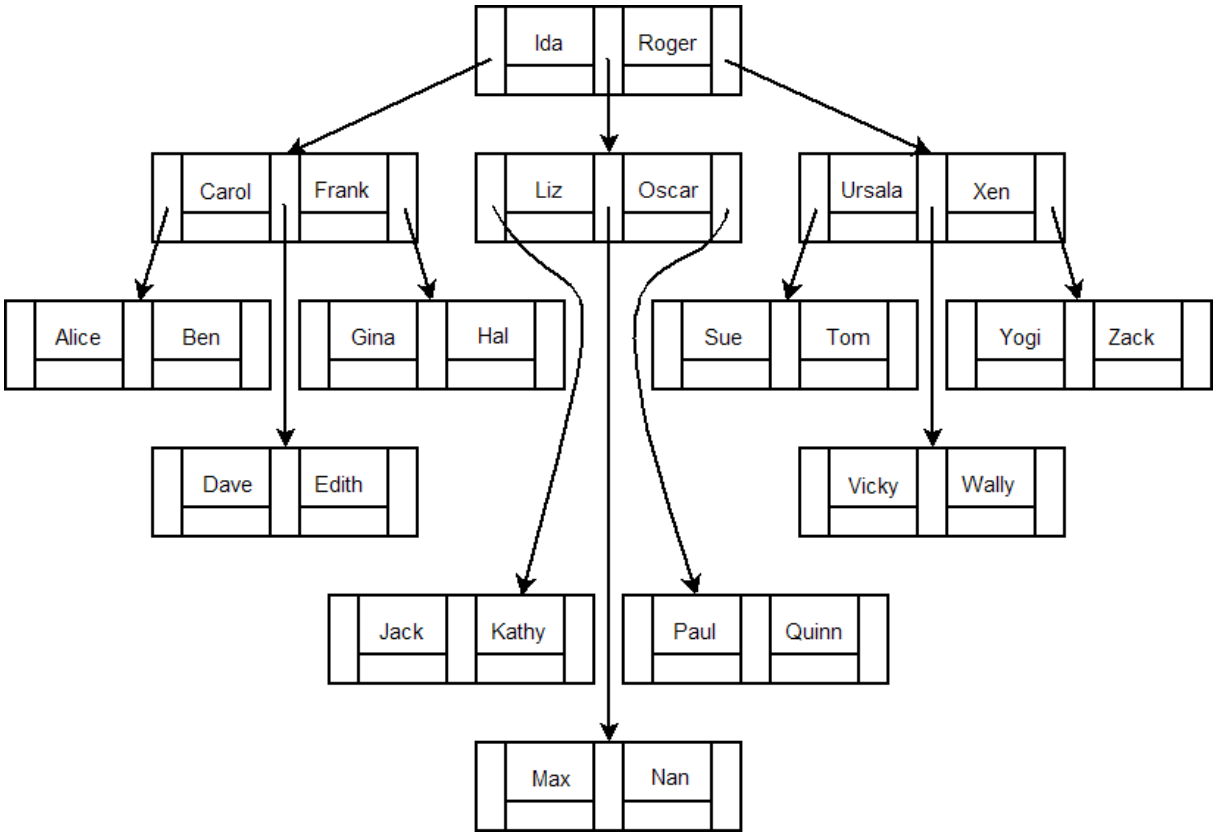


Obr. 38: Třídy, které tvoří datovou strukturu Soubor s úplným indexem. Zdroj: vlastní.

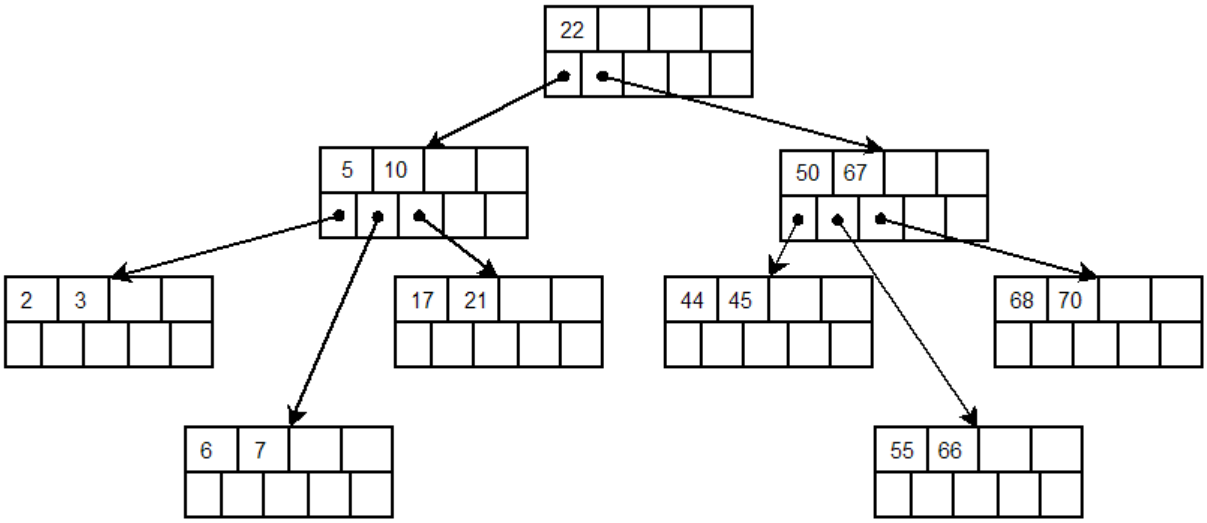
4.2 Implementace grafické reprezentace k-cestných stromů

4.2.1 Analýza stávajícího stavu v literatuře

V literatuře je několik typů zobrazení stromů. Vzhledem k omezenému místě na papíře jsou některé stromy zobrazovány „úsporným způsobem“, čili tak, aby strom měl co nejmenší šířku a vešel se tak na stránku. Autoři těchto zobrazení stromů nezobrazují jednotlivé úrovně stromů na stejné výšce, ale vhodně seskupí bloky tak, aby měl strom co nejmenší šířku za cenu zvýšení výšky obrázku grafické reprezentace stromu. Tímto způsobem se ale sníží čitelnost takových stromů. Příklady těchto druhů stromů jsou na obrázcích 39 a 40.

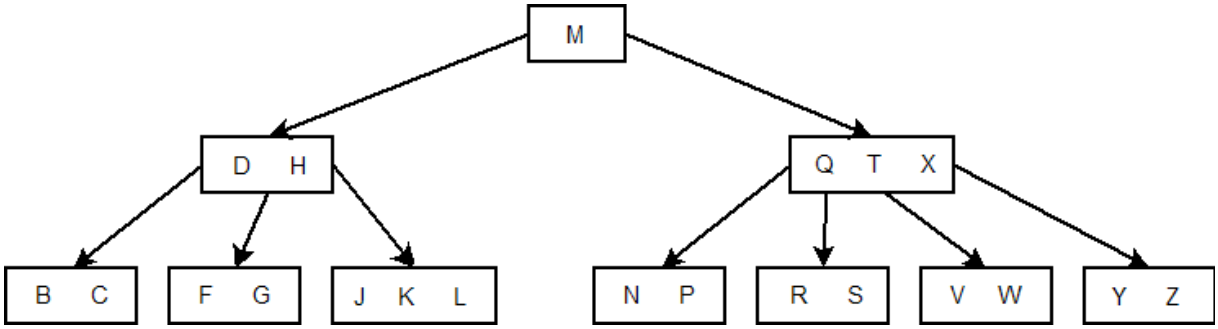


Obr. 39: Příklad stromu, ve kterém je snížena šířka obrázku na úkor jeho výšky. Zdroj: [6]

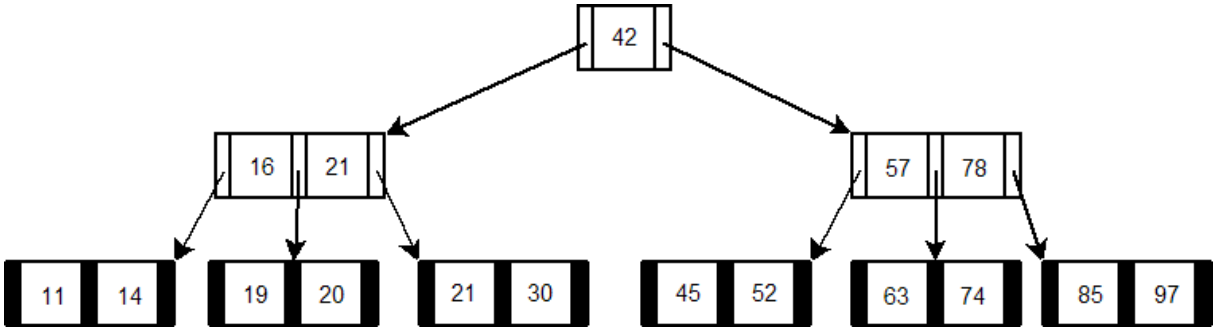


Obr. 40: Příklad stromu, ve kterém je snížena šířka obrázku na úkor jeho výšky. Zdroj: [29]

Jiní autoři volí přístup, při kterém se vynechají prázdné záznamy v blocích. Tímto způsobem se sníží šířka obrázku na kterém se strom nachází, ale čitelnost takových stromů se výrazně nesníží. Příklady takových stromů jsou uvedeny na obrázcích 41 a 42.



Obr. 41: Příklad stromu, ve kterém jsou vynechány prázdné záznamy v blocích. Zdroj: [4]



Obr. 42: Příklad stromu, ve kterém jsou vynechány prázdné záznamy v blocích. Zdroj: [7]

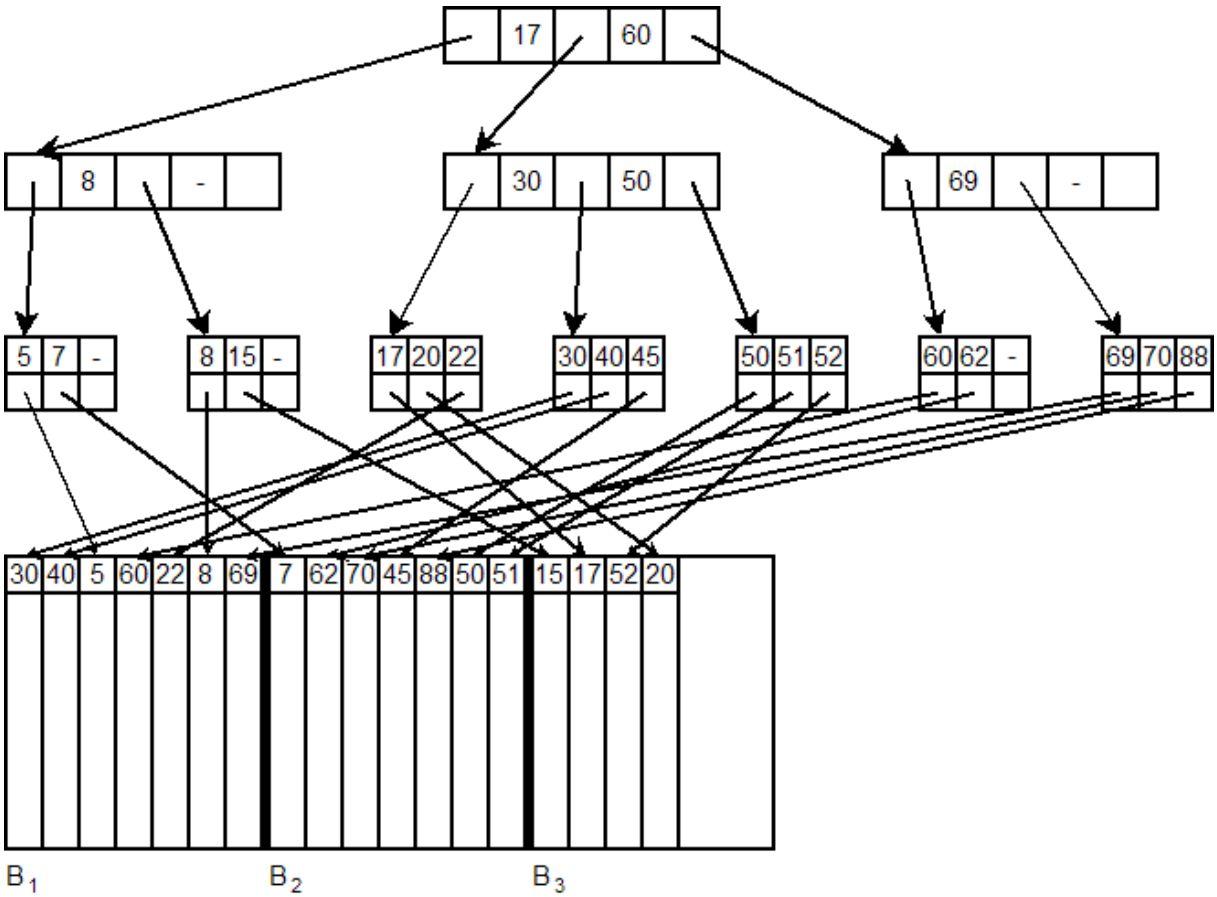
Autoři obrázků stromů se neliší pouze v prostorovém uspořádání bloků, ale i v samotné struktuře bloku. Vzhled bloků je uniformní – všechny bloky jsou tvořeny obdélníkem, příp. obdélníkem s oblými rohy.

Záznamy mají většinou podobu čtverce, příp. obdélníku. Výjimečně se lze setkat se stromy, ke nejsou záznamy nijak ohraničeny (např. obr. 41). Takový způsob zobrazení je ale vhodný pouze pro záznamy, které se skládají z pouze jednoho písmena.

Nejvýrazněji se autoři liší v zobrazení místa, ze kterého vychází reference, která spojuje záznam s příslušným blokem. Tato místa se většinou zobrazují vedle záznamu, nebo pod záznamem. Ve výjimečných případech se nezobrazují vůbec.

Druhů zobrazení stromů je tudíž nepřeberné množství a lze je libovolně kombinovat. Jediné co je důležité mít na paměti je přehlednost stromu pro čtenáře.

Na obrázku 43 je uveden strom, ve kterém autor nezvolil ani jeden z výše uvedených kompromisů pro zmenšení šířky stromu. Při tvorbě stromů s vyšší výškou nebo vyššího řádu stromu by se ale narazilo na výše uvedená omezení a obrázek by musel být podle toho vhodně upraven.



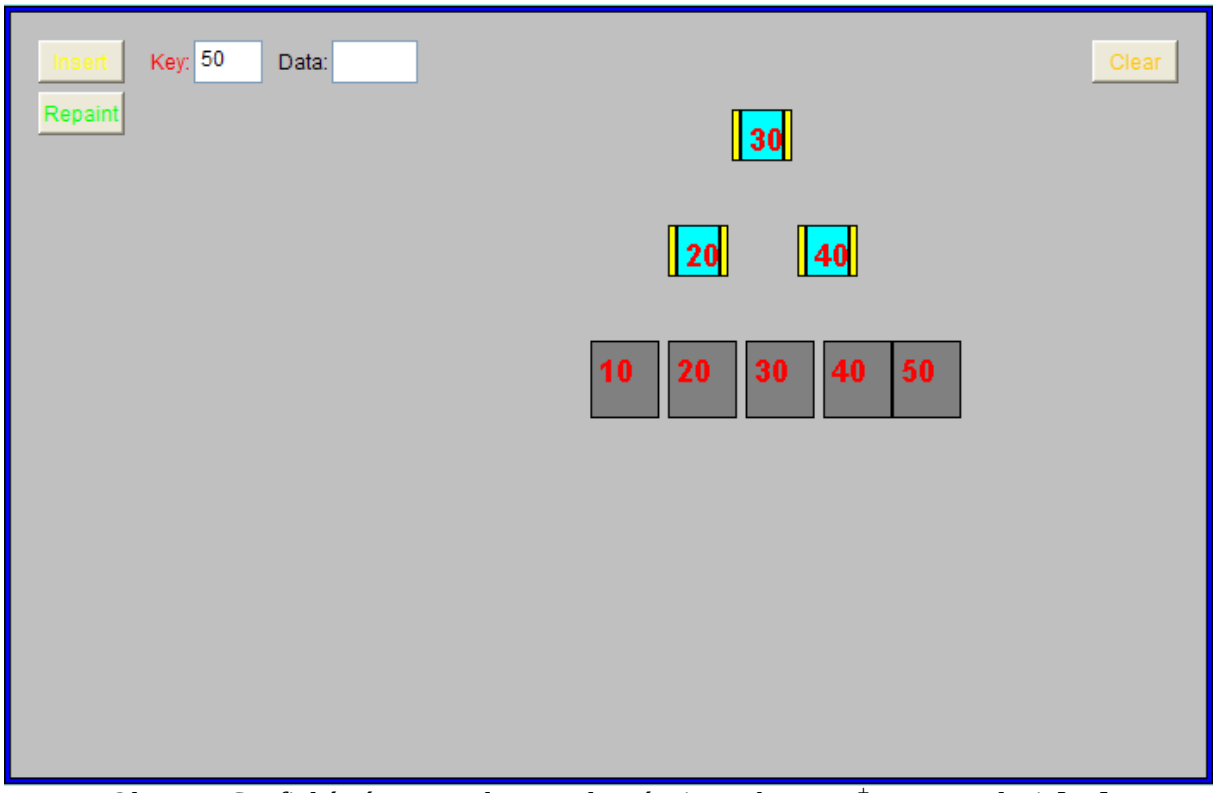
Obr. 43: Kompletní obrázek Souboru s úplným indexem. Zdroj: [1]

4.2.2 Analýza stávajícího stavu na Internetu

Na Internetu je k dispozici zdarma množství appletů (Java aplikace, která „běží“ v Internetovém prohlížeči), které zobrazují k-cestné stromy. Na rozdíl od stromů, které jsou zobrazeny v knihách, jsou takové stromy většinou navíc animované. Na následujících obrázcích jsou uvedeny 2 vybrané applety, které zobrazují k-cestné stromy

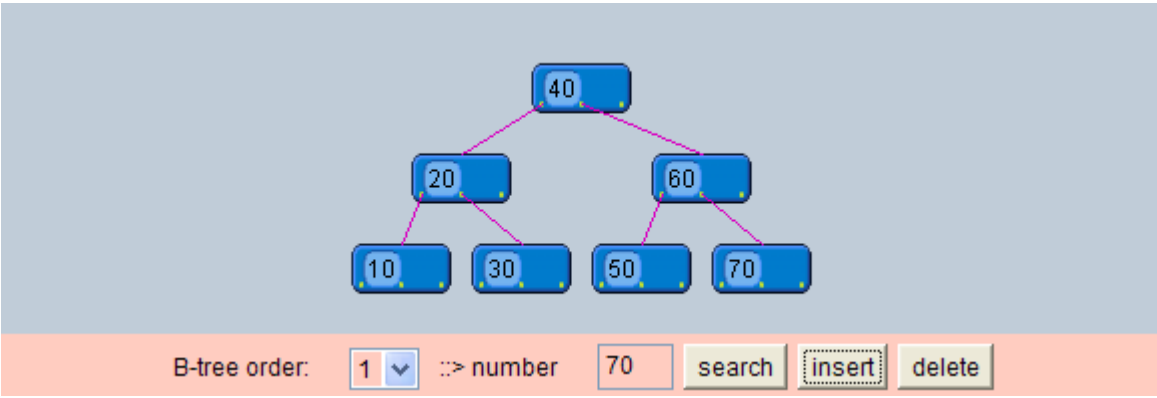
Úroveň appletů, které jsou volně k dispozici na Internetu, je velice rozdílná. Počet operací, které je možno se stromem provádět, je často velice omezený. Nezřídka se lze setkat s případem, kdy je k dispozici pouze operace Vlož záznam, příp. i operace Najdi záznam. Úroveň animace se také velice liší. Animace by měla sloužit pro lepší pochopení problematiky příslušných operací prováděných ve stromu, občas je ale na škodu. Navíc většině aplikací úplně chybí možnost vypnutí animace. Na přiloženém CD médiu jsou video soubory, ve kterých je zobrazena animace uvedených appletů pro porovnání s animací aplikace, která je součástí diplomové práce.

Na následujícím obrázku je zachycen applet, který zobrazuje B^+-strom. V souboru *analyza_1.avi*, který je přiložen k diplomové práci na CD médiu je zachycena animace vkládání záznamu s klíčem 50.



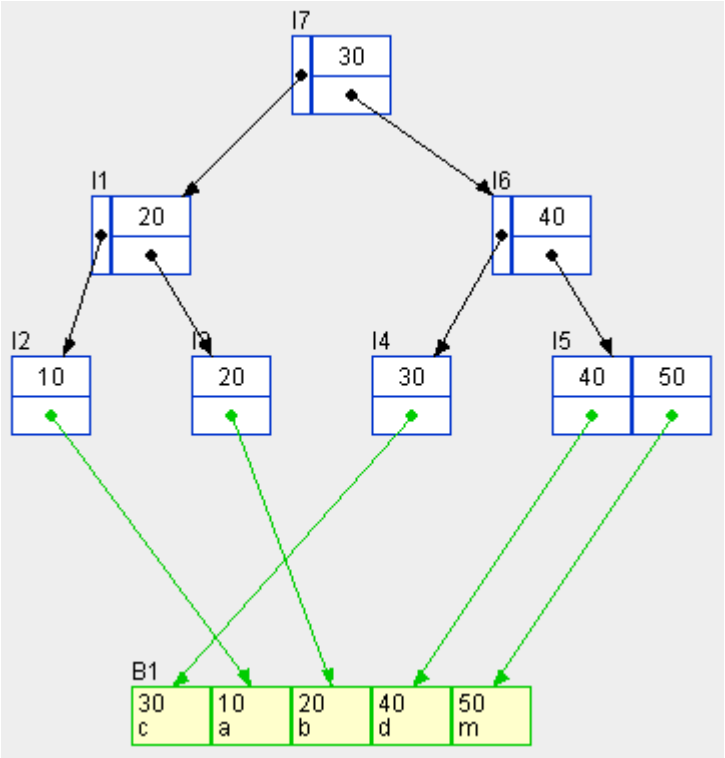
Obr. 44: Grafický výstup appletu, na kterém je zachycen B^+-strom. Zdroj:[22]

Na obrázku 45 je screenshot appletu, který zobrazuje B-strom. V souboru *analyza_2.avi*, přiloženém na CD médiu je zachycena animace vkládání záznamu s klíčem 70.



Obr. 45: Grafický výstup appletu, na kterém je zachycen B-strom. Zdroj: [23]

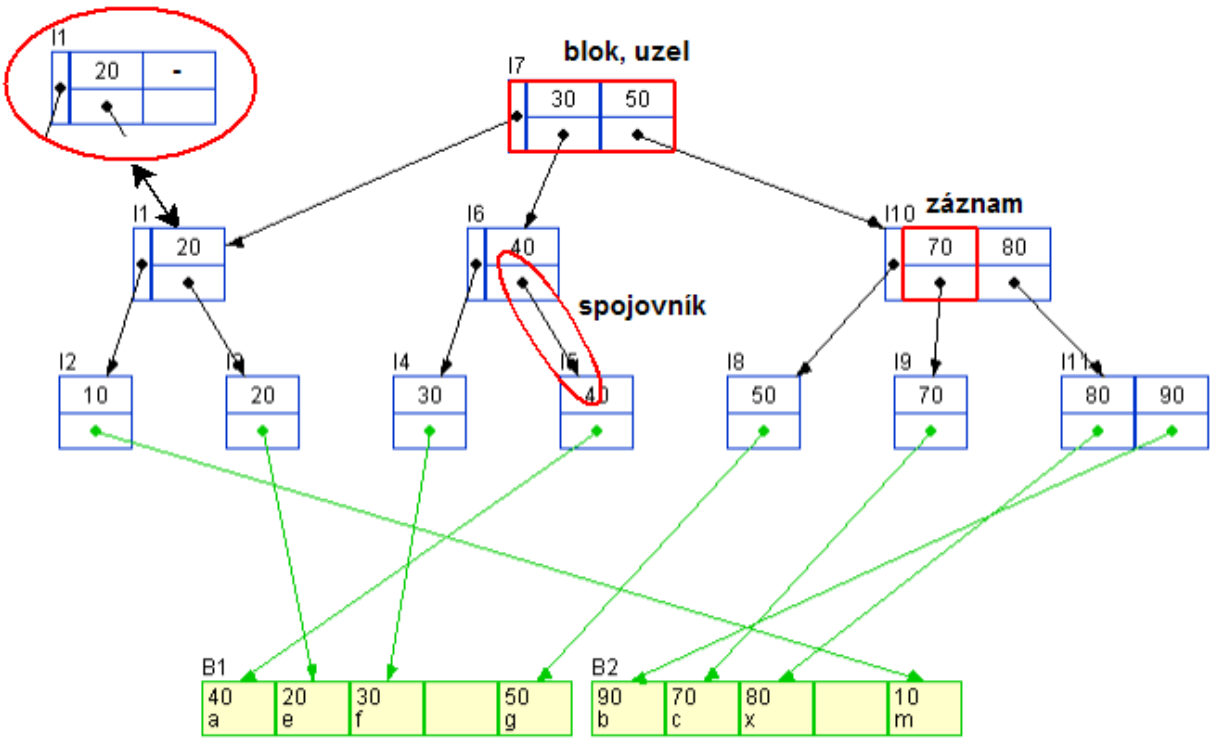
Pro srovnání je zde nyní prezentován obrázek Souboru s úplným indexem z aplikace, která je součástí diplomové práce. Pro srovnání animace tří zmíněných aplikací je na přiloženém CD médiu k dispozici soubor *analyza_3.avi*, na němž je prezentováno vložení záznamu s klíčem 50 do Souboru s úplným indexem.



Obr. 46: Grafický výstup aplikace, která je součástí diplomové práce. Zdroj: vlastní.

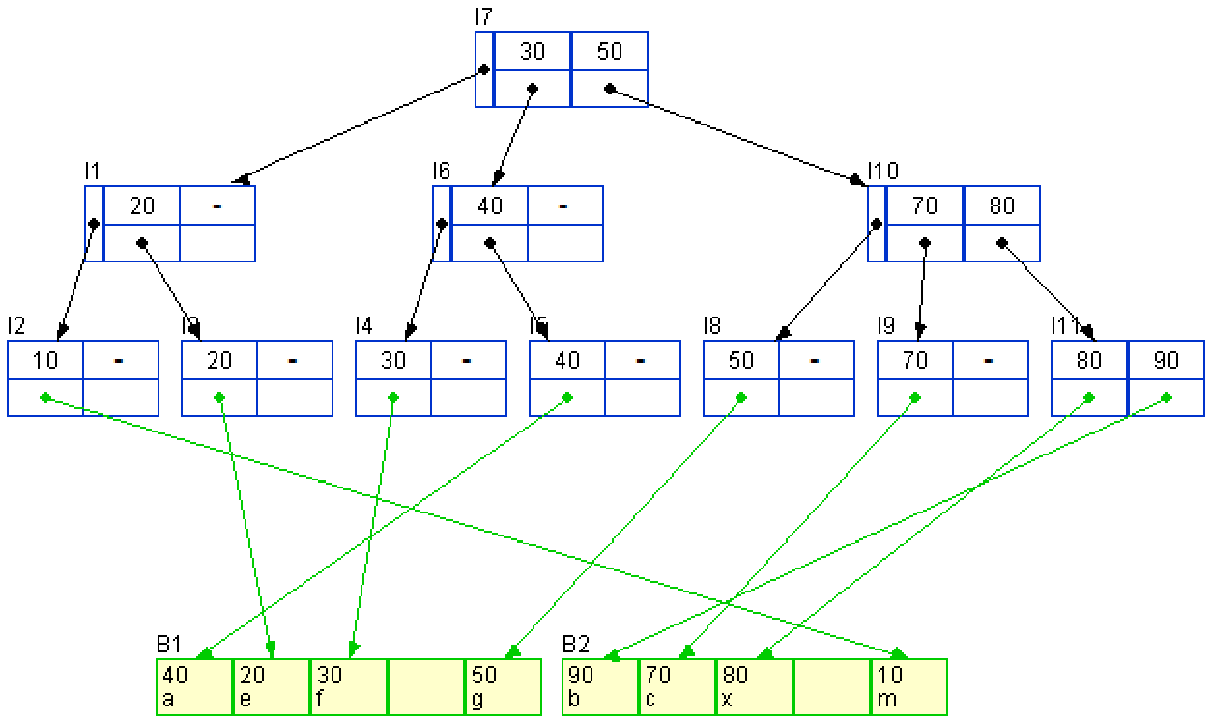
4.2.3 Vnitřní uspořádání stromu

V teoretické části byla zavedena konvence, že blok a uzel stromu jsou synonyma. V dalším textu se bude postupovat podle této konvence, tudíž pojem *blok* se bude používat ve smyslu fyzické reprezentace na disku, zatímco pojem *uzel* se bude používat ve smyslu logického uspořádání ve stromu. Blok dále obsahuje *záznamy*. Každý ze záznamů také obsahuje *spojovník*, což je reference na další blok. Tyto pojmy jsou také zobrazeny na obrázku 47. První úrovní stromu bude myšlen kořen, kdežto poslední úrovní stromu budou myšleny listy stromu.



Obr. 47: Přehled částí grafického k-cestného stromu. Zdroj: vlastní.

Na následujícím obrázku je znázorněno, jak by vypadal strom, kdyby byly zobrazeny všechny záznamy. Z důvodu šetření s místem byl ale zvolen „úsporný“ způsob zobrazování stromu. Záznamy, které jsou prázdné se ve stromu nezobrazují.

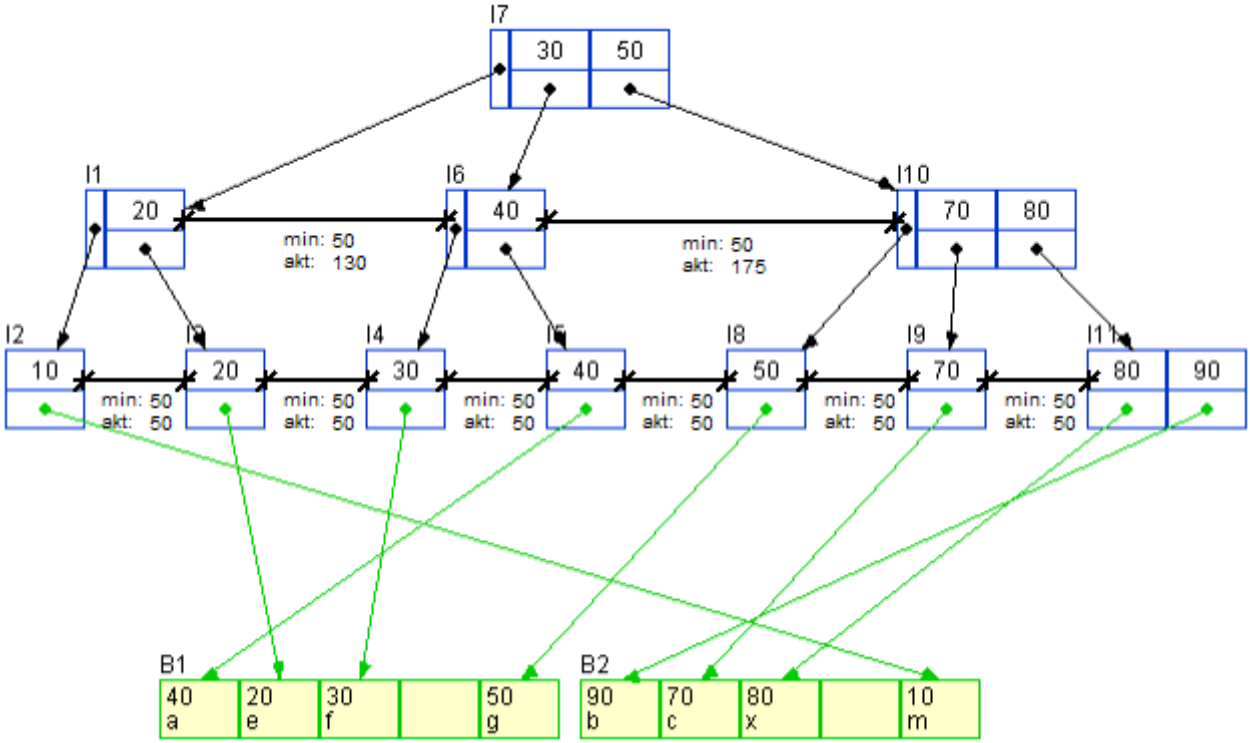


Obr. 48: Ilustrace Souboru s úplným indexem se zobrazením všech záznamů v blocích. Zdroj: vlastní.

Grafická reprezentace stromu má své vnitřní uspořádání a pravidla pro tvorbu tohoto stromu. Tyto pravidla, která jsou pro uživatele skrytá, budou v této kapitole vysvětlena.

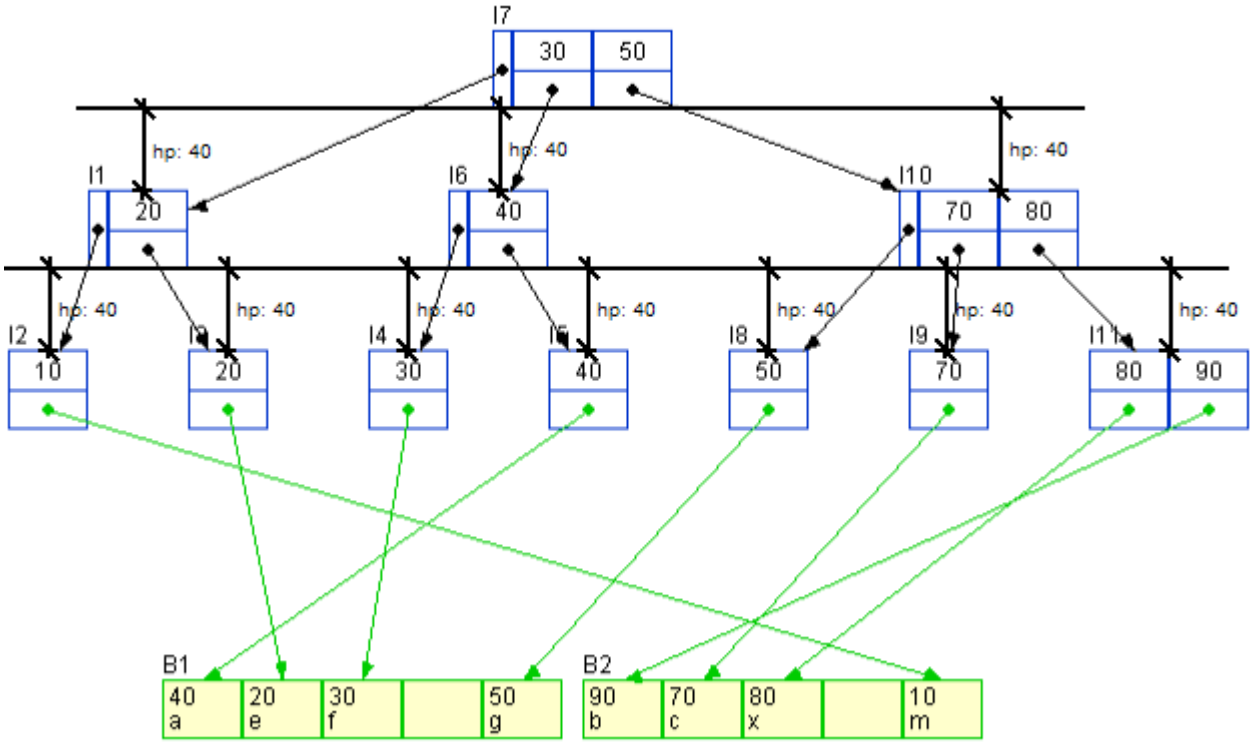
Na obrázcích 49 a 50 je znázorněn mechanismus, který zabezpečuje, aby mezi uzly stromu grafické reprezentace byly přiměřené vzdálenosti a zajišťuje zamýšlenou strukturu stromu.

Na obrázku 49 je znázorněna aplikace atributů *aktPadding* a *minPadding*. „Padding“ je anglické slovo, které v překladu znamená „výplň“. Každý blok stromu obsahuje tyto dva atributy, které představují neviditelnou výplň mezi bloky. Oba atributy jsou definovány ve třídě *AbstrBlok*. Atribut m*inPadding* představuje minimální vzdálenost mezi dvěma uzly na horizontální úrovni. Atribut a*ktPadding* obsahuje aktuální vzdálenost mezi dvěma horizontálními uzly. Hodnota atributu *minPadding* je stanovena na začátku jako konstanta a tudíž je pro všechny uzly stejná. Tato hodnota byla stanovena na 50 pixelů. Minimální hodnota atributu *aktPadding* je hodnota atributu *minPadding*. Aktuální hodnota se nastaví v metodě *update()*, která bude podrobně vysvětlena později. Mezi listy stromu je hodnota *aktPadding* vždy rovna hodnotě *minPadding*.



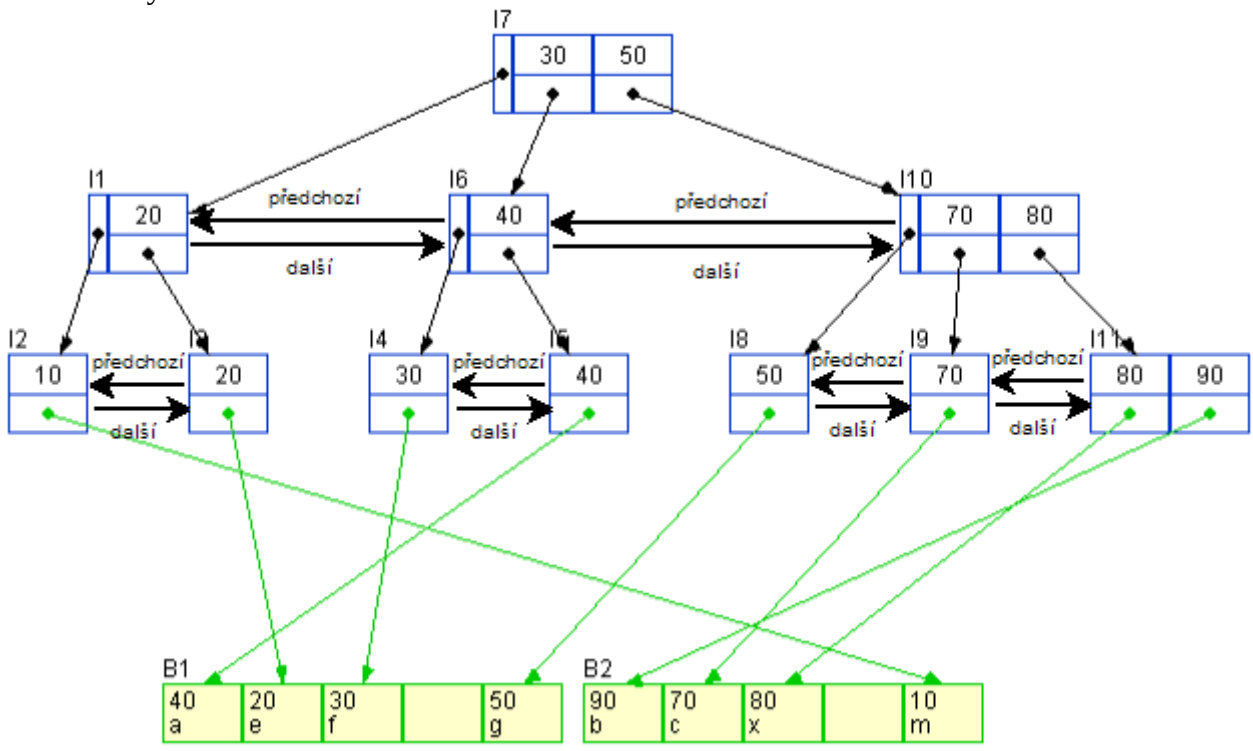
Obr. 49: Ilustrace mechanismu zachování vzdáleností mezi bloky na horizontální úrovni. Zdroj: vlastní.

Na obrázku 50 je znázorněna aplikace atributu *horizontalniPadding*. Tento atribut určuje vertikální rozložení uzlů ve stromu a tudíž vzdálenosti mezi dvěma úrovněmi stromu. Hodnota atributu *horizontalniPadding (hp)* je nastavena konstantou na 40 pixelů.



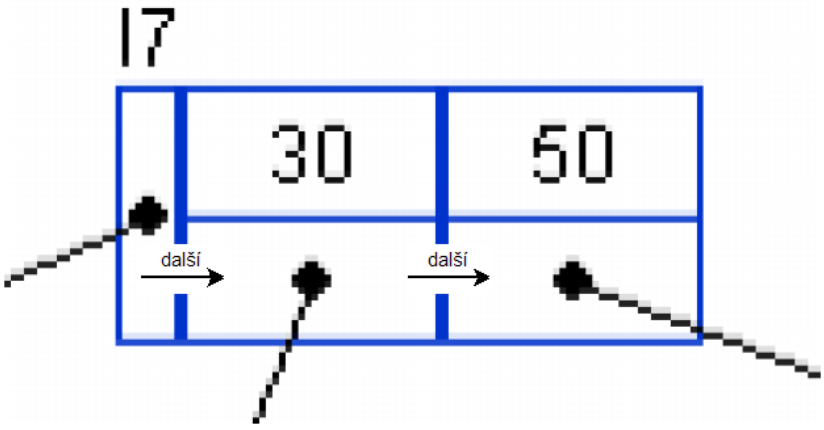
Obr. 50: Ilustrace mechanismu zachování vzdáleností mezi bloky na vertikální úrovni. Zdroj: vlastní.

Na dalších dvou obrázcích jsou znázorněny vnitřní vazby mezi grafickou reprezentací bloků a záznamů. Na obrázku 51 je znázorněno, že uzly stromu, které mají stejného otce jsou spojeny obousměrným seznamem.



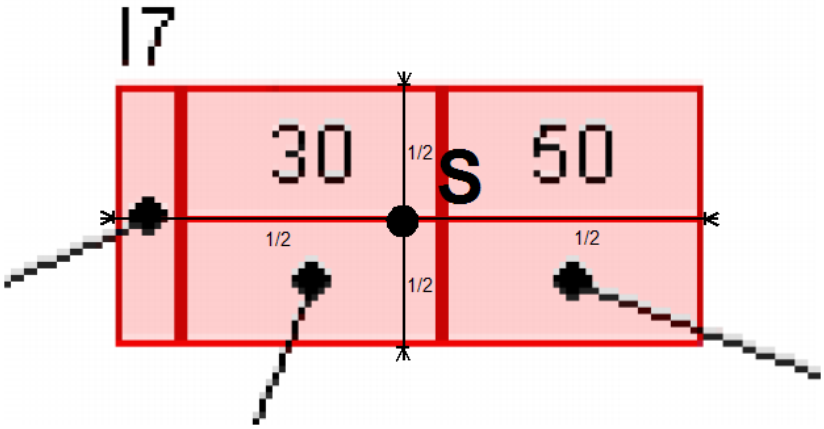
Obr. 51: Vnitřní vazby mezi bloky. Zdroj: vlastní.

Na obrázku 52 jsou znázorněny vazby mezi záznamy v bloku. Tyto záznamy jsou spojeny v jednosměrném seznamu. Je důležité si uvědomit, že směrník na nultý uzel bloku je ve skutečnosti v prvním záznamu v bloku. Na obrázku jsou tudíž fyzicky tři záznamy. První má rozměry 10x40 a další dva mají rozměry 40x40. Blok má celkové rozměry 90x40, přičemž jednotky jsou v pixelech.



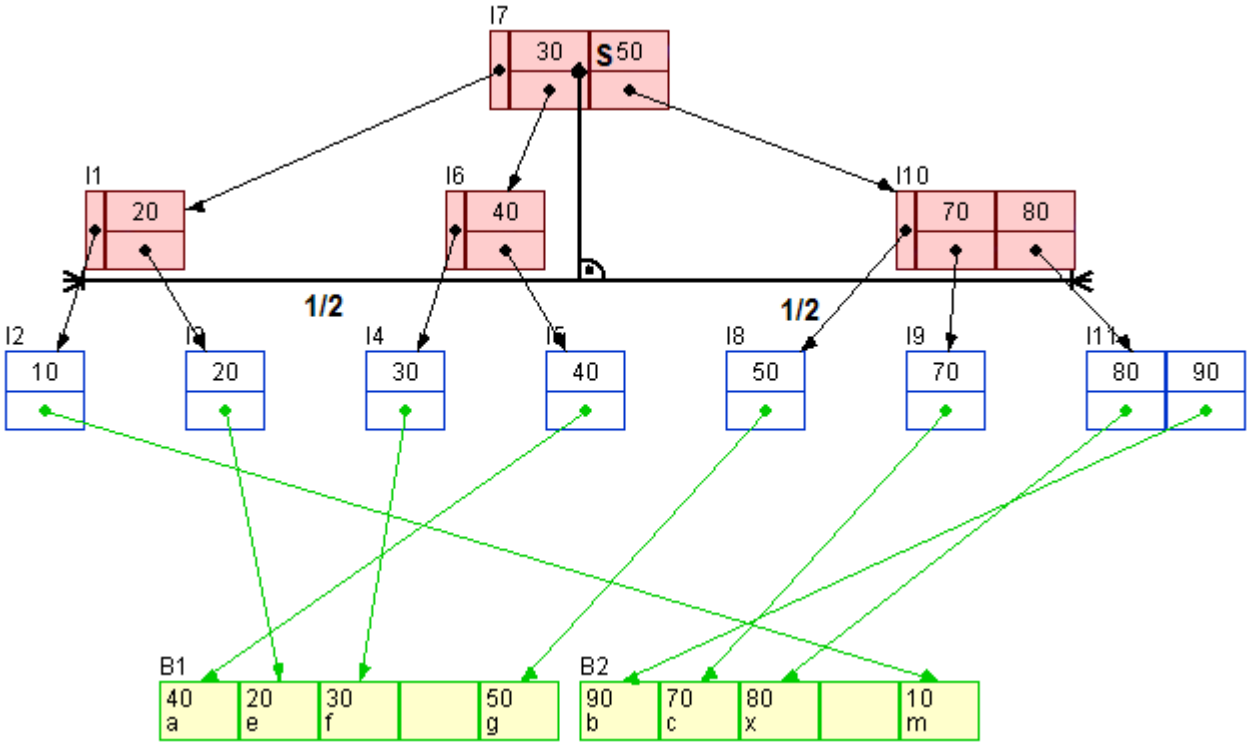
Obr. 52: Vnitřní vazby mezi záznamy. Zdroj: vlastní.

Na dalších dvou obrázcích je ilustrováno fyzické rozložení uzlů vůči svému otci. Nejprve je na obrázku 53 znázorněno co se míní pod pojmem *střed bloku*. Střed bloku (S) je bod, který je přesně uprostřed bloku. Blok na obrázku 53 má velikost 90x40 pixelů. Střed bloku je tudíž bod o souřadnicích [45,20].



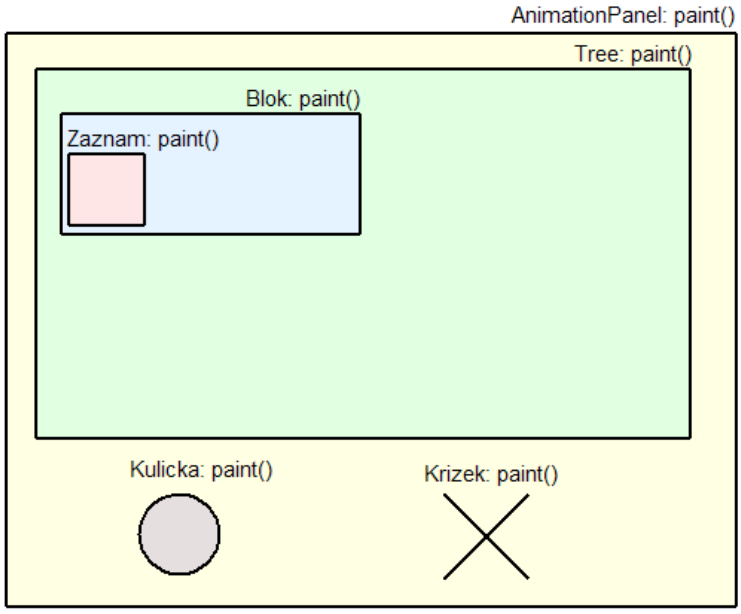
Obr. 53: Znázornění středu bloku (S). Zdroj: vlastní.

Na obrázku 54 je znázorněno fyzické rozložení uzlů vůči svému otci. Je vidět, že x-ová souřadnice středu jejich otce je umístěna v polovině vzdálenosti mezi jeho prvním a posledním synem. Umístění uzlů má na starosti metoda *update()*, které bude věnována později celá kapitola.



Obr. 54: Proporcionální rozložení bloků vůči svému otci. Zdroj: vlastní.

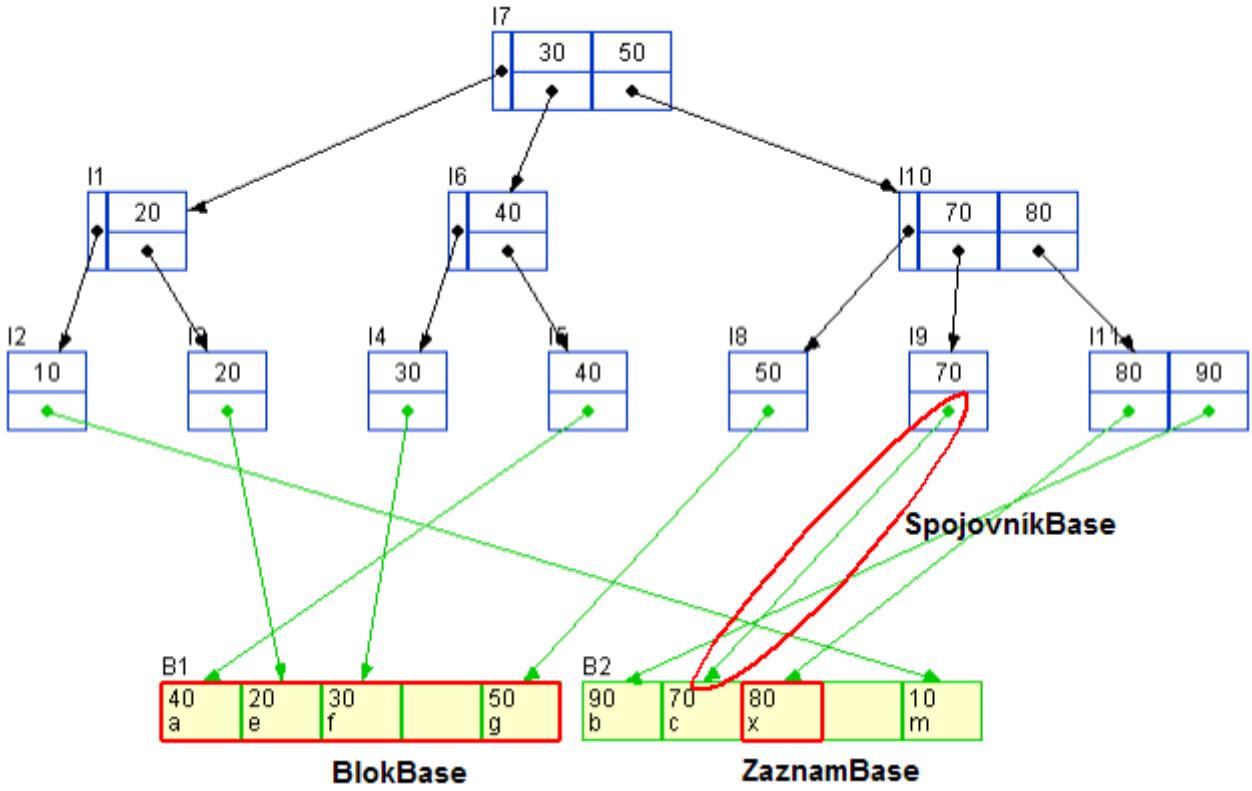
Na následujícím obrázku jsou zobrazeny metody, které slouží pro vykreslení objektů na obrazovku. Každá třída, která se zobrazí na obrazovce obsahuje metodu *paint()*. Tato metoda vykreslí příslušný objekt na obrazovku. Některé objekty mohou navíc obsahovat další objekty (např. objekt typu *Blok* obsahuje objekty typu *Zaznam*). V takovém případě zavolá metoda *paint()* typu *Blok* metody *paint()* objektů typu *Zaznam*, které obsahuje.



Obr. 55: Princip vykreslování stromu a dalších objektů na obrazovku. Zdroj: vlastní.

4.2.4 Diagram tříd grafické reprezentace Úplného indexu

Pojmy *blok*, *spojovník* a *záznam* již byly vysvětleny v předcházející kapitole. Nyní se ještě pro úplnost zadefinují zbylé grafické struktury. Jejich přehled je na obrázku 56. Tyto objekty plní stejnou funkci jako stejně nazvané objekty ve stromu. Jediný rozdíl je ten, že obsluhují grafickou reprezentaci a animaci haldy.



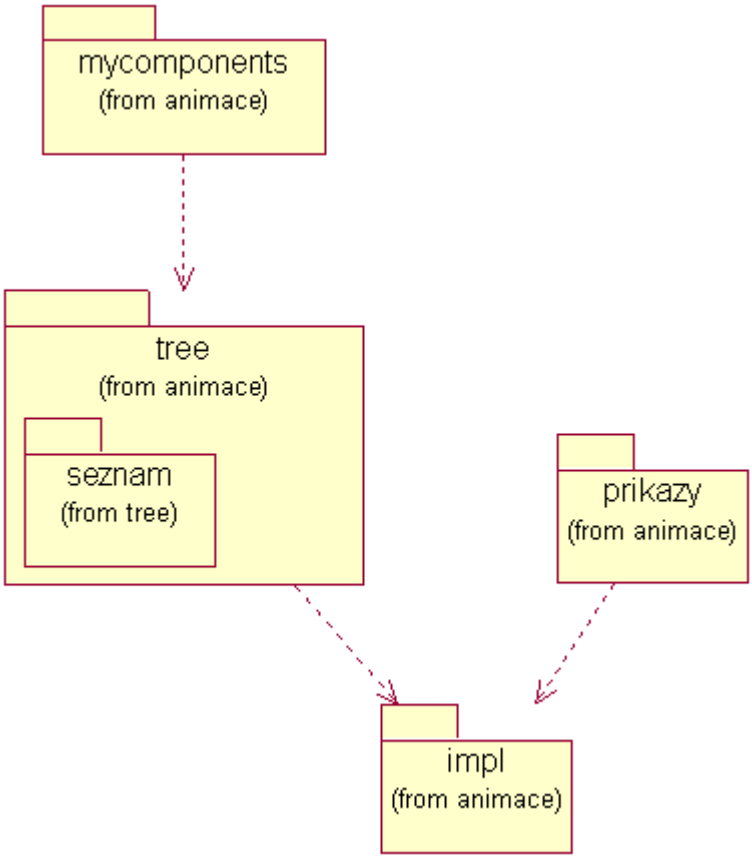
Obr. 56: Přehled zbylých částí grafického k-cestného stromu. Zdroj: vlastní.

Všechny třídy jsou potomky tříd *MyJComponent* a *MyContainer*. Třída *MyJComponent* je odvozena od třídy *JComponent* [33]. Třída *MyJContainer* je odvozena od třídy *Container* [34]. Třída *MyJComponent* je grafická komponenta. Třídu *Container* lze stylizovaně nadefinovat následovně:

```
class Container extends Component {
      Component[] pole;
}
```

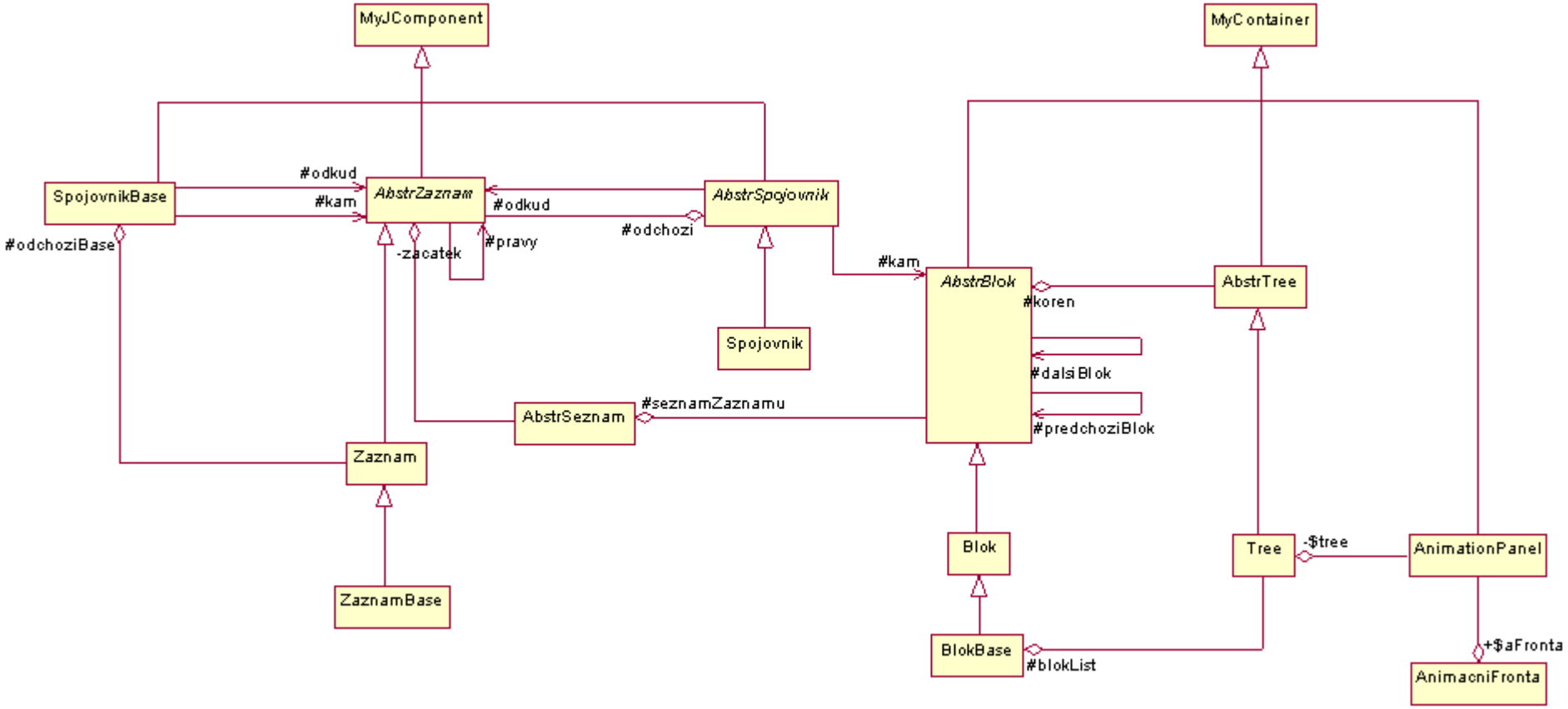
Objekt typu *Container* tudíž obsahuje další komponenty typu *Component* [35]. Třída *Component* je ekvivalentem třídy *JComponent*.

Na následujícím obrázku je přehled nadefinovaných balíčků, ze kterých se skládá implementace grafického reprezentace k-cestného stromu. Všechny balíčky jsou v balíčku *info.pinkas.animace* podle konvence tvorby balíčků v jazyku Java. Balíček *tree* obsahuje balíček *seznam*.



Obr. 57: Přehled nadefinovaných balíčků. Zdroj: vlastní.

Na obrázku 58 je znázorněn celý diagram tříd grafické reprezentace Souboru s úplným indexem. Vzhledem k tomu, že se Soubor s úplným indexem skládá ze dvou datových struktur, jsou zde tyto dvě struktury také zastoupeny prostřednictvím tříd. Třídy, které mají na konci příponu *Base*, zastupují datovou strukturu typu Neutříděný soubor s přímým přístupem (halda-soubor) a její návaznost na B^+-strom. Ostatní třídy obsluhují činnost stromu.



Obr. 58 – Kompletní diagram tříd grafické reprezentace Souboru s úplným indexem. Zdroj: vlastní.

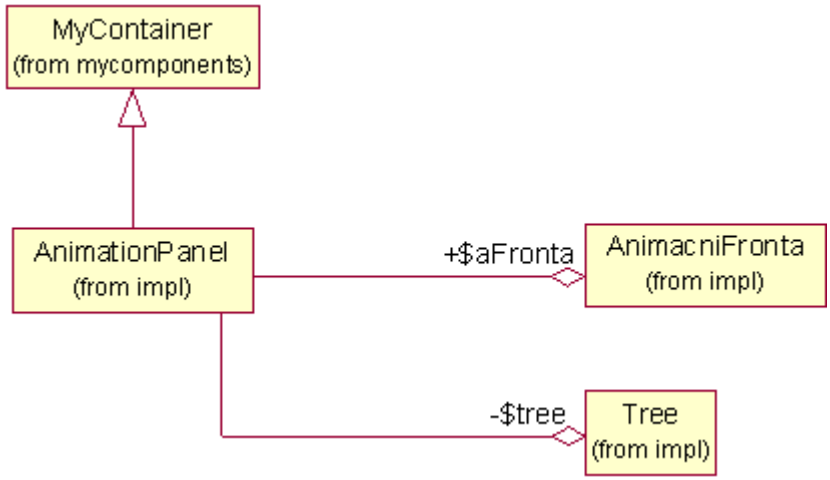
V následujících kapitolách se proberou jednotlivé části podrobněji. Na rozdíl od předcházejícího popisu implementace datové struktury Souboru s úplným indexem zde kvůli přehlednosti nebudou zobrazeny plné hlavičky tříd.

4.2.5 Animační panel a animační fronta

Na obrázku 59 je znázorněna třída *AnimationPanel*. Třída *AnimationPanel* je kontejner, který obsahuje všechny objekty na plátně (plocha aplikace, na kterou se dá kreslit). Mezi tyto objekty patří strom a další objekty, které zabezpečují animaci stromu (objekty typu *Kulicka* nebo *Krizek* a další).

Hlavní metodou, která čte příkazy pro animaci je metoda *importBTree()*, která je obsažena ve třídě *AnimationPanel*. Tato metoda načte vnější příkazy pro animaci, zkonvertuje je na vnitřní příkazy a uloží do animační fronty. Podrobně budou příkazům věnovány dvě samostatné kapitoly.

Instance třídy *AnimationPanel* běží jako vlákno nezávisle na aplikaci. Toho je docíleno v její metodě *run()*, ve které je nekonečný cyklus, který vykonává příkazy pro animaci a periodicky vykresluje strom na obrazovku [14].



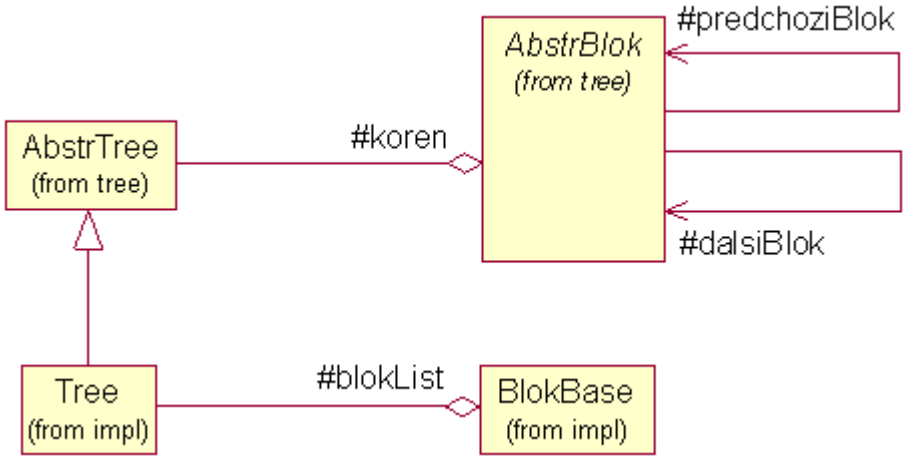
Obr. 59: Třída AnimationPanel a její vazba na animační frontu a strom. Zdroj: vlastní.

Pozn. Znak $ před názvem atributu znamená, že atribut je **static**.

Trída *AnimationPanel* obsahuje animační frontu, která je instancí třídy *AnimacniFronta*. Ve třídě *AnimacniFronta* jsou uloženy vnitřní příkazy, které se z vrcholu fronty odebírají pro zobrazení na obrazovce a na konec fronty se další příkazy vkládají. Fronta je implementována obousměrným seznamem.

4.2.6 Strom

Třída *AbstrTree* obsahuje kořen stromu. Pomocí něj má přístup k celému B^+-stromu. V této třídě se nachází metoda *update()*, která udržuje grafickou strukturu stromu. O této metodě bude pojednáno v samostatné kapitole. Na obrázku 60 je zobrazen strom a jeho dva hlavní atributy.

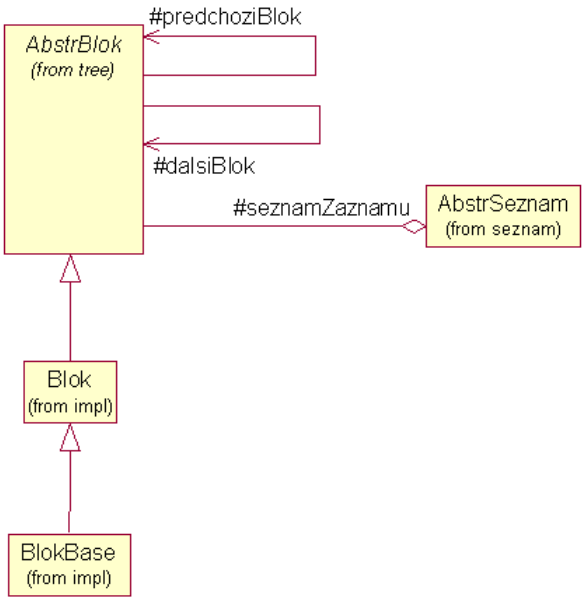


Obr. 60: Třídy, které tvoří grafický Soubor s úplným indexem. Zdroj: vlastní.

Od třídy *AbstrTree* je odvozena třída *Tree*, která přidává ke stromu Halda-soubor. Tato struktura je tvořena atributem *blokList*, což je ukazatel na první záznam obousměrného seznamu, který tvoří grafickou reprezentaci haldy. Přidána je také metoda *updateBase()*, která udržuje grafickou strukturu haldy.

4.2.7 Bloky

Na dalším obrázku jsou ilustrovány třídy pro grafickou reprezentaci bloků použitých ve stromu. Třída *AbstrBlok* je předchůdcem všech bloků ve stromu. Obsahuje ukazatele na další a předchozí blok, protože bloky, které mají stejného otce, jsou zřetězeny v obousměrném seznamu. Dále obsahuje atribut *seznamZaznamu*, ve kterém je uložen jednosměrný seznam záznamů v bloku. Třída *AbstrBlok* obsahuje také atributy *minPadding*, *aktPadding* a *horizontalniPadding*, o kterých již bylo pojednáno.

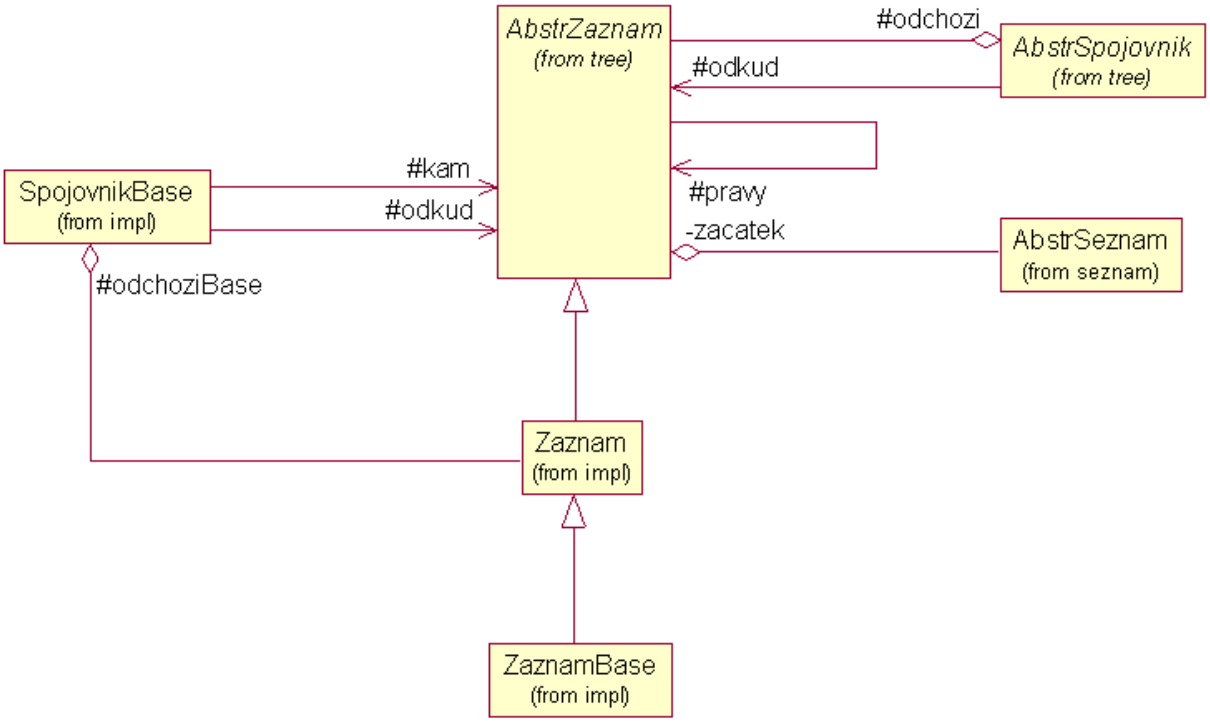


Obr. 61: Třídy, které tvoří grafickou reprezentaci bloků ve stromu. Zdroj: vlastní.

4.2.8 Záznamy

Na následujícím obrázku jsou ilustrovány záznamy. Všechny záznamy jsou potomky třídy *AbstrZaznam*. Tato třída obsahuje atribut *pravy*, který obsahuje ukazatel na další záznam v jednosměrném seznamu záznamů v bloku.

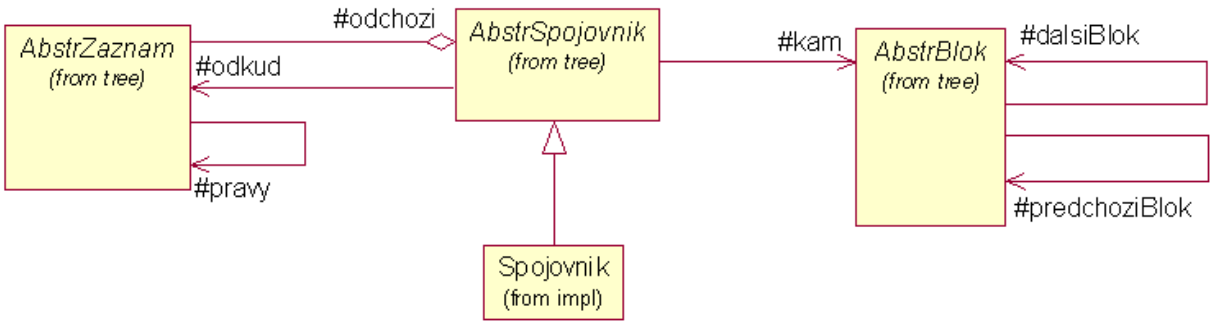
Ve třídě *AbstrZaznam* je definován atribut *odchozi*, který jej spojuje se spojovníkem, který graficky spojuje záznam s příslušným blokem. Instance třídy *AbstrZaznam* jsou použity jako atributy ve třídách *AbstrSpojovnik* a *SpojovnikBase*, ale o tom bude pojednáno v následující kapitole.



Obr. 62: Třídy, které tvoří grafickou reprezentaci záznamů ve stromu. Zdroj: vlastní.

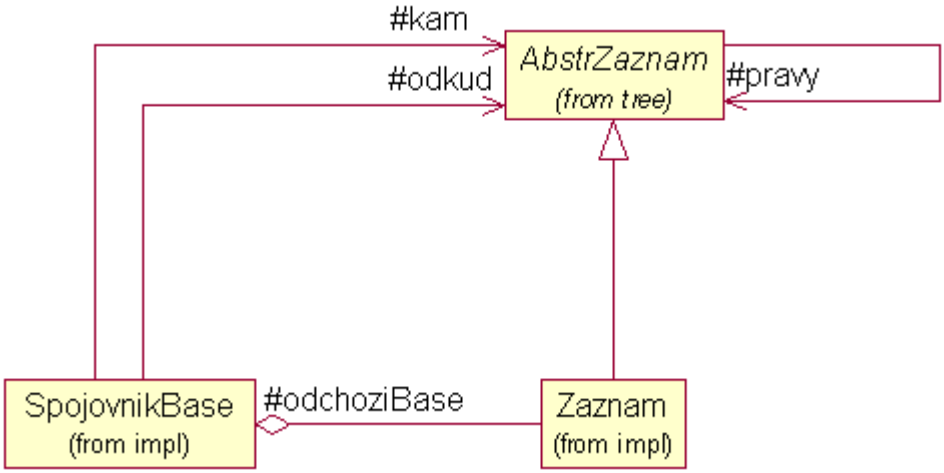
4.2.9 Spojovníky

Spojovníky plní důležitou roli ve stromu. Spojují v grafické reprezentaci záznamy s bloky, nebo záznamy se záznamy. Plní funkci reference z datové struktury Soubor s úplným indexem. Třída *AbstrSpojovník* spojuje záznam s blokem. Toto spojení se používá ve stromu. Instance třídy *Spojovnik* je ve stromu vykreslena ve formě šipky. Důležité vazby třídy *AbstrSpojovnik* na ostatní objekty jsou zobrazeny na obrázku 63.



Obr. 63: Třídy, které tvoří grafickou reprezentaci referencí ve stromu. Zdroj: vlastní.

Na obrázku 64 je ilustrace třídy *SpojovnikBase*. Tato třída není odvozena od třídy *AbstrSpojovnik*, přestože plní stejnou funkci. Důvodem je to, že třída *AbstrSpojovnik* spojuje záznam a blok stromu, zatímco třída *SpojovnikBase* spojuje dva záznamy.

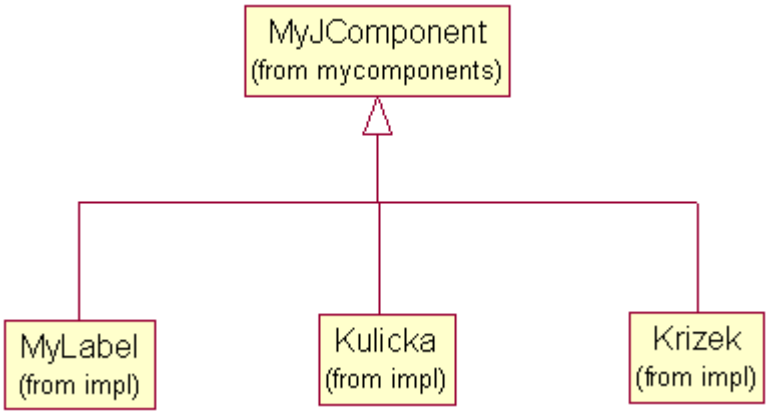


Obr. 64: Třídy, které tvoří grafickou reprezentaci referencí, které spojují záznam z indexu a záznam bázového souboru. Zdroj: vlastní.

4.2.10 Ostatní použité grafické objekty

Ostatní použité grafické objekty jsou pro podporu animace. Všechny musí být odvozeny od tříd *MyJComponent*, nebo *MyContainer*. V současné době existují tři grafické objekty, které jsou všechny odvozeny od třídy *MyJComponent*.

- **MyLabel** – Zobrazuje na obrazovce text „B^+-strom řádu …“
- **Kulicka** – Zobrazuje kuličku na obrazovce. Kulička slouží pro animaci průchodu stromem.
- **Krizek** – Zobrazuje křížek na obrazovce. Křížek se zobrazí před fyzickým odebráním záznamu ze stromu.

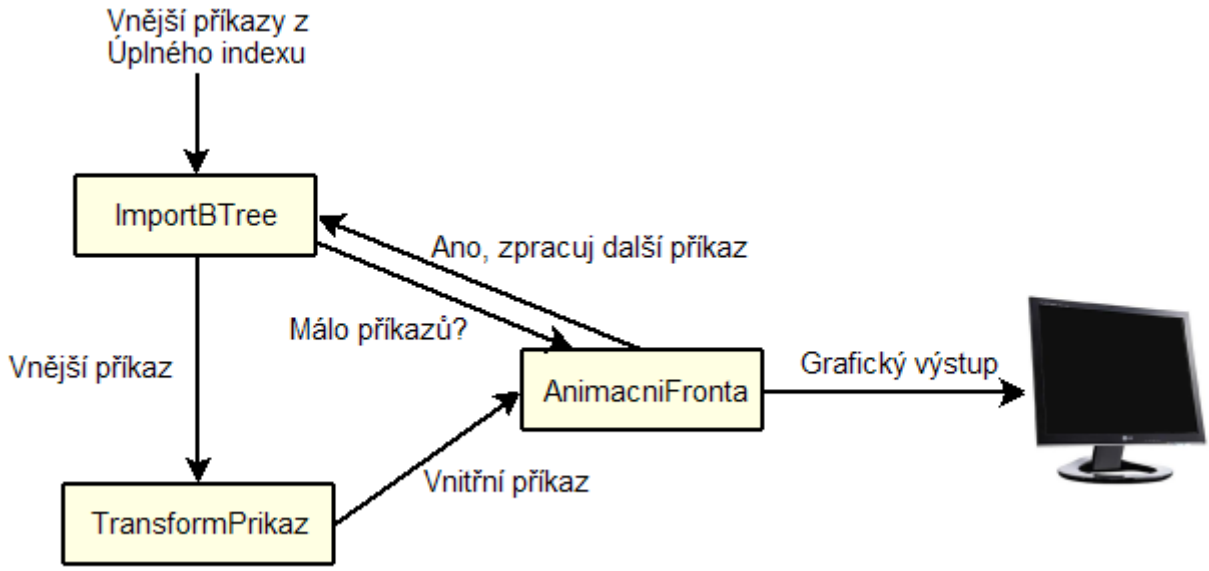


Obr. 65: Ostatní grafické objekty. Zdroj: vlastní.

4.2.11 Vnější příkazy pro animaci

Strom a jeho animace je ovládána vnějšími příkazy ze Souboru s úplným indexem (Úplného indexu). V Úplném indexu se provede určitá operace (např. operace Vlož záznam) a v průběhu této operace se získávají data o změnách v Úplném indexu. Tato data jsou nazvána *vnější příkazy*. Tyto vnější příkazy načte animační jádro grafického stromu a překonvertuje je na vnitřní příkazy, se kterými poté pracuje.

Na obrázku 66 je zobrazen výše popsaný princip funkce animačního jádra.

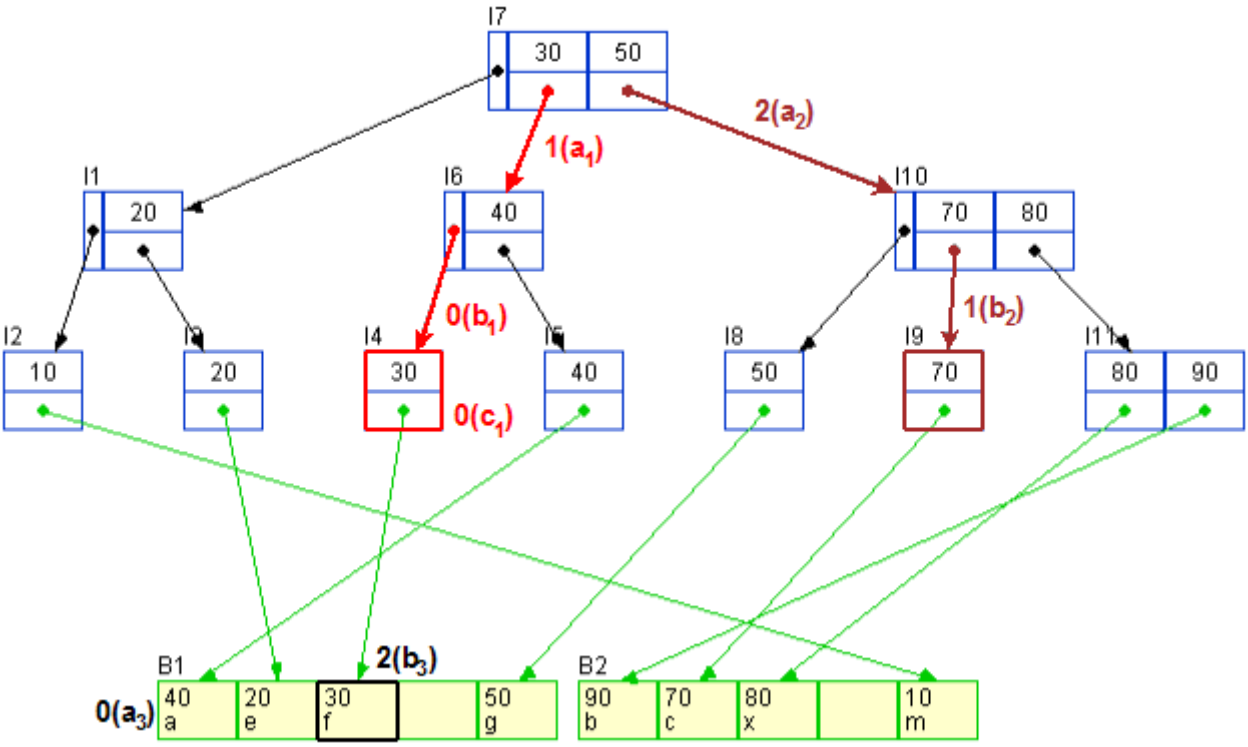


Obr. 66: Princip funkce animačního jádra. Zdroj: vlastní.

Ve vnějších příkazech, které budou v následujícím textu popsány, jsou použity následující termíny:

- **CESTA, CESTA1, CESTA2** – Cesta stromem od kořene k bloku, který předchází příslušnému záznamu, resp. bloku
- **UMISTENI, UMISTENI1, UMISTENI2** – Umístění záznamu v bloku
- **CESTAB** – Cesta zleva doprava v haldě (písmeno B je kvůli tomu, že halda reprezentuje bázový soubor)
- **UMISTENIB** – Umístění záznamu v bloku haldy
- **STRING** – Textová hodnota
- **INTEGER** – Hodnota typu Integer
- **VELIKOST** – Velikost záznamu (ve formátu délka,výška)

Nyní bude na obrázku 67 demonstrováno na třech příkladech použití výše zmíněných *cest* a *umístění* ve stromu. Pozn. indexy záznamů v bloku jsou 0 … *n* - 1, kde *n* je počet záznamů v bloku.



Obr. 67: Demonstrace principu cest a umístění ve stromu. Zdroj: vlastní.

Př. 1 – Zpřístupnění záznamu 30 ve stromu
CESTA = a_1, b_1 (zpřístupní se blok I4)
UMISTENI = c_1 (zpřístupní se záznam 30)
Pozn. Cestou se zpřístupňuje blok, který obsahuje hledaný záznam.

Př. 2 – Zpřístupnění bloku I9 ve stromu
CESTA = a_2 (zpřístupní se blok I10)
UMISTENI = b_2 (zpřístupní se blok I9)
Pozn. Cestou se zpřístupňuje blok, který předchází hledanému bloku.

Př. 3 – Zpřístupnění záznamu [30;d] v Halda-souboru
CESTAB = a_3 (zpřístupní se blok B1)
UMISTENIB = b_3 (zpřístupní se záznam [30;f])
Pozn. Bloky se v Halda-souboru zpřístupňují zleva doprava. První blok má index 0.

Následuje přehled vnějších příkazů, které jsou textovým výstupem Souboru s úplným indexem. Příkazy se importují do grafického stromu prostřednictvím metody *importBTree()* ve třídě *AnimationPanel*. Příkazy se předávají do metody prostřednictvím datové struktury *ArrayList* nebo souboru, kde je každý příkaz na jednom řádku. V případě že příkaz začíná znakem #, je ignorován. V případě, že příkaz nebyl rozpoznán, je o tom uživatel informován v konzoli. V případě nesprávného pořadí příkazů (a tudíž nesprávné tvorbě stromu) je vytvořena výjimka. Detaily výjimky jsou vypsány v konzoli a uživatel je o výskytu výjimky informován. Pozn. konzola je standardně skrytá. Ještě je důležité upozornit, že „ \t “ je značka pro tabulátor. V příkazu je tedy místo značky „ \t “ ve skutečnosti tabulátor.

```
VLOZ \t BLOK \t CESTA \t UMISTENI \t STRING
```
Vloží se nový blok do stromu ke stávajícímu záznamu, který lze zpřístupnit pomocí CESTA, UMISTENI. Blok má popisek STRING (např. I7)

```
VLOZ \t BLOKB \t \t STRING
```
Vloží se další blok do haldy. Blok má popisek STRING (např. B3)

```
VLOZ \t IBLOK \t CESTA \t UMISTENI \t STRING
```
Vloží se nový blok stejným způsobem jako u příkazu VLOZ \t BLOK. Blok je ale nastaven jako neviditelný.

```
VLOZ \t ZAZNAM \t CESTA \t UMISTENI \t STRING \t VELIKOST
```
Vloží se nový záznam do stromu na místo CESTA, UMISTENI. UMISTENI vyjadřuje index v bloku, na který se záznam vloží. Velikost záznamu je daná atributem VELIKOST (např. 40,40). První záznam v bloku má index nula.

```
VLOZ \t IZAZNAM \t CESTA \t UMISTENI \t STRING \t VELIKOST
```
Vloží se nový záznam do stromu stejným způsobem jako u příkazu VLOZ \t ZAZNAM. Záznam je ale nastaven jako neviditelný.

```
VLOZ \t ZAZNAMB \t CESTAB \t UMISTENIB \t STRING \t VELIKOST
```
Vloží se nový záznam do haldy do bloku CESTAB na místo UMISTENIB. Záznam má popisek STRING (např. 10;a). Středník je ekvivalentem \n (nový řádek).

```
VLOZ \t SPOJB \t CESTA \t UMISTENI \t CESTAB \t UMISTENIB
```
Vloží se nový spojovník, který ukazuje ze stromu do haldy. Výchozí záznam je určen CESTA, UMISTENI a koncový záznam je určen CESTAB, UMISTENIB.

```
VLOZ \t KOREN \t STRING \t VELIKOST
```
Vloží se nový kořen do stromu. Kořen obsahuje popisek STRING a jeden počáteční záznam, který obsahuje spojovník na původní kořen. Atribut VELIKOST obsahuje velikost tohoto záznamu.

```
ODEBER \t SPOJB \t CESTA \t UMISTENI
```
Odebere se spojovník, který ukazuje ze stromu do haldy.

```
ODEBER \t ZAZNAM \t CESTA \t UMISTENI
```
Odebere se záznam ve stromu.

```
ODEBER \t SUBBLOK \t CESTA \t UMISTENI
```
Odebere se blok a spojovník, na který ukazuje CESTA, UMISTENI. Tento příkaz se příliš nevyužívá, protože většinou se odebírají neplatné bloky a spojovníky automaticky. Může ale nastat situace, ve které je použití tohoto příkazu žádoucí.

```
ODEBER \t KOREN
```
Odebere se kořen stromu a nahradí se blokem, na který ukazuje první záznam ve stávajícím kořenu.

```
NAJDI \t ZAZNAM \t CESTA \t UMISTENI
```
Provede se animace vyhledání záznamu ve stromu.

```
UPDATE \t ZAZNAM \t CESTA \t UMISTENI \t STRING
```
Zpřístupní se záznam ve stromu pomocí CESTA, UMISTENI a poté se změní jeho popisek na hodnotu STRING.

```
UPDATE \t ZAZNAMB \t CESTAB \t UMISTENIB \t STRING \t VELIKOST
```
Zpřístupní se záznam v haldě a změní se jeho popisek na hodnotu STRING. Lze také nastavit novou velikost záznamu.

```
UPDATE \t BLOKS \t CESTA \t STRING
```
Zpřístupní se blok v souboru a změní se jeho popisek na hodnotu STRING.

```
SET \t VISIBLEB \t CESTA \t UMISTENI
```
Blok, který se zpřístupní pomocí CESTA, UMISTENI se nastaví jako viditelný.

```
SET \t INVISIBLEB \t CESTA \t UMISTENI
```
Blok, který se zpřístupní pomocí CESTA, UMISTENI se nastaví jako neviditelný.

```
SET \t VISIBLEZ \t CESTA \t UMISTENI
```
Záznam, který se zpřístupní pomocí CESTA, UMISTENI se nastaví jako viditelný.

```
SET \t INVISIBLEZ \t CESTA \t UMISTENI
```
Záznam, který se zpřístupní pomocí CESTA, UMISTENI se nastaví jako neviditelný.

```
ZVYRAZNI \t ZAZNAM \t CESTA \t UMISTENI
```
Záznam ve stromu, který se zpřístupní pomocí CESTA, UMISTENI se na několik vteřin zvýrazní (vykreslí se červenou barvou).

```
ZVYRAZNI \t ZAZNAMB \t CESTAB \t UMISTENIB
```
Záznam v haldě, který se zpřístupní pomocí CESTAB, UMISTENIB se na několik vteřin zvýrazní (vykreslí se červenou barvou).

```
KRIZEK \t ZAZNAM \t CESTA \t UMISTENI
```
Přes záznam ve stromu, který se zpřístupní pomocí CESTA, UMISTENI se na pár vteřin vykreslí křížek. Tento křížek značí následující odebrání záznamu.

```
KRIZEK \t ZAZNAMB \t CESTAB \t UMISTENIB
```
Přes záznam v haldě, který se zpřístupní pomocí CESTAB, UMISTENIB se na pár vteřin vykreslí křížek. Tento křížek značí následující odebrání záznamu.

```
KULICKA \t B \t CESTA \t UMISTENI
```
Zobrazí se animace kuličky mezi záznamem, na který ukazuje CESTA, UMISTENI a záznamem v bázovém souboru, na nějž tento záznam ukazuje pomocí *SpojovnikBase*.

```
VYTVOR \t STROM \t STRING
```
Vytvoří se nový strom. Kořen stromu má popisek STRING.

```
ZRUS \t STROM
```
Zruší se aktuální strom.

```
SET \t VYSKA \t INTEGER
```
Nastaví se aktuální výška stromu. Výška stromu se automaticky zvyšuje při přidávání kořenu stromu a snižuje při odebírání kořenu stromu. Tento příkaz tu je z důvodu nastavení větší vzdálenosti mezi stromem a bázovým souborem, tudíž je specifický pro strukturu Úplný index.

```
FORCE \t REFACTOR
```
Vynutí se vykonání metody *update()*. Tato metoda se sice zavolá po každém vložení (odebrání) záznamu, ale při dávkovém vkládání (odebírání) záznamů je vhodné tuto metodu zavolat, protože v tom případě by automatické volání metody nefungovalo.

```
ADD \t INFO \t STRING
```
Přidá se popisek s informacemi na obrazovku. Tyto informace jsou vlevo nahoře na obrazovce. Středník v tomto popisku je ekvivalentem \n (nový řádek).

```
ADD \t LABEL \t STRING \t VELIKOST
```
Přidá se popisek do horního středu obrazovky s informacemi o řádu B^+-stromu.

Dále jsou vypsány příkazy, které jsou použité pro animaci přesunů záznamů v B^+-stromu. Tato sekvence příkazů začíná příkazem PRESUN \t ZAZNAM \t START a končí příkazem PRESUN \t ZAZNAM \t KONEC. Mezi těmito dvěma příkazy může být řada příkazů, které ovládají vlastní animaci přesunu záznamu. Tyto příkazy jsou popsány níže.

```
PRESUN \t ZAZNAM \t START
```
Tento příkaz označuje aktivaci animace přesunu záznamu.

```
PRESUN \t ZAZNAM \t KONEC
```
Tento příkaz označuje konec animace přesunu záznamu.

```
VLOZ \t ZAZNAM \t CESTA \t UMISTENI \t STRING \t VELIKOST
```
Vloží se záznam do souboru na místo CESTA, UMISTENI. Záznam má popisek STRING a jeho velikost je VELIKOST.

```
POSUN \t ZAZNAM \t CESTA1 \t UMISTENI1 \t CESTA2 \t UMISTENI2
```
Přesune se záznam pomocí vnitřního příkazu *Posun*. Tento typ posunu je vhodný při přesouvání záznamů na stejné úrovni stromu.

```
POSUN \t ZAZNAM2 \t CESTA1 \t UMISTENI1 \t CESTA2 \t UMISTENI2
```
Přesune se záznam pomocí vnitřního příkazu *APosun*. Tento typ posunu je vhodný při přesouvání záznamů, které jsou na jiných úrovních stromu.

```
PRESUN \t SPOJ \t CESTA1 \t UMISTENI1 \t CESTA2 \t UMISTENI2
```
Výchozí bod spojovníku, který původně byl záznam CESTA1, UMISTENI1 se změní na záznam CESTA2, UMISTENI2.

```
ODEBER \t ZAZNAM \t CESTA \t UMISTENI
```
Odebere se záznam, který lze zpřístupnit pomocí CESTA, UMISTENI.

```
UPDATE \t ZAZNAM \t CESTA \t UMISTENI \t STRING
```
Změní se popisek záznamu, který lze zpřístupnit pomocí CESTA, UMISTENI na novou hodnotu STRING.

4.2.12 Vnitřní příkazy pro animaci

Vnější příkazy pro animaci nedokáží plně ovládat všechny objekty na obrazovce a jejich animaci. Ani to není vhodné, protože pak by vnější příkazy pro animaci zacházely do implementačních detailů animace. Proto pracuje animační jádro s vnitřními příkazy animace. Díky tomuto rozdělení a zjednodušení vnějších příkazů je také možné jednoduše implementovat různé druhy stromů s použitím stejných knihoven animace.

Vnitřní příkazy pro animaci jsou fyzicky objekty tříd, které jsou v balíčku *prikazy*. Tyto objekty obsahují informaci o určité činnosti ve stromu. V metodě *importBTree()* ve třídě *AnimationPanel* se vnější příkazy převádějí na vnitřní příkazy a vkládají se na konec

animační fronty. Příkazy z vrcholu animační fronty jsou periodicky odebírány a provádějí animaci na obrazovce.

Vnitřní příkazy jsou následující třídy v balíčku *prikazy*:

- **Posun** – Posune se záznam nebo blok po plátně. Tento příkaz dokáže posouvat pouze části stromu jako jsou bloky nebo záznamy.
- **APosun** – Posune se objekt po plátně. Tento příkaz posouvá objekty, které nejsou součástí stromu
- **PosunBase** – Posune se objekt typu *BaseBlok* po plátně.
- **PresunSpojovnik** – Přesune výchozí bod spojovníku z jednoho záznamu na druhý
- **PresunSpojovnikBase** – Tento příkaz plní stejnou funkci jako příkaz PresunSpojovnik, ale protože třídy *Spojovnik* a *SpojovnikBase* jsou fyzicky 2 rozdílné třídy, tak jsou k dispozici i 2 rozdílné příkazy plnící stejnou funkci
- **Stop** – Tento příkaz ohraničuje konec série příkazů, které se vykonávají paralelně
- **Strom** – Uloží se aktuální pozice všech objektů ve stromu pro pozdější obnovení takového stromu v průběhu animace. Tento příkaz nahrazuje sadu jiných příkazů, které by byly bez tohoto mechanismu nutné
- **Velikost** – Změní se velikost záznamu nebo bloku
- **Viditelnost** – Záznam nebo blok se nastaví jako viditelný, nebo neviditelný
- **ZmenitKlic** – Změní se hodnota klíče záznamu
- **Zobrazeni** – Zobrazí se cizí objekt na plátně. Takovým objektem je objekt typu *Krizek* nebo *Kulicka*
- **Zvyrazneni** – Zvýrazní se záznam nebo blok. Zvýrazněním je myšleno to, že se příslušný objekt vykreslí červenou barvou
- **ZvyrazneniBase** – Zvýrazní se záznam nebo blok haldy. Pro zvýraznění je opět použita červená barva

Všechny příkazy se odebírají z vrcholu fronty po jednom. Výjimkou však může být příkaz typu *Posun* nebo *PosunBase*. Tyto třídy obsahují atribut *soubezne*, jehož hodnota je implicitně nastavena na *false*. Pokud se v animační frontě narazí na příkaz, který má tento atribut nastaven na *true*, potom se pseudoparalelně v čase vykonává série těchto příkazů, které jsou ve frontě za sebou. Ostatní příkazy jsou odděleny od této sekvence příkazem Stop. Toto pseudoparalelní vykonávání příkazů se využívá při animaci výsledků metody *update()*.

4.2.13 Udržování struktury stromu

Udržování struktury stromu má na starosti metoda *update()*. Na uživatelské úrovni je tato metoda ve třídě *AnimationPanel*. Tato metoda volá dvě podřízené metody:

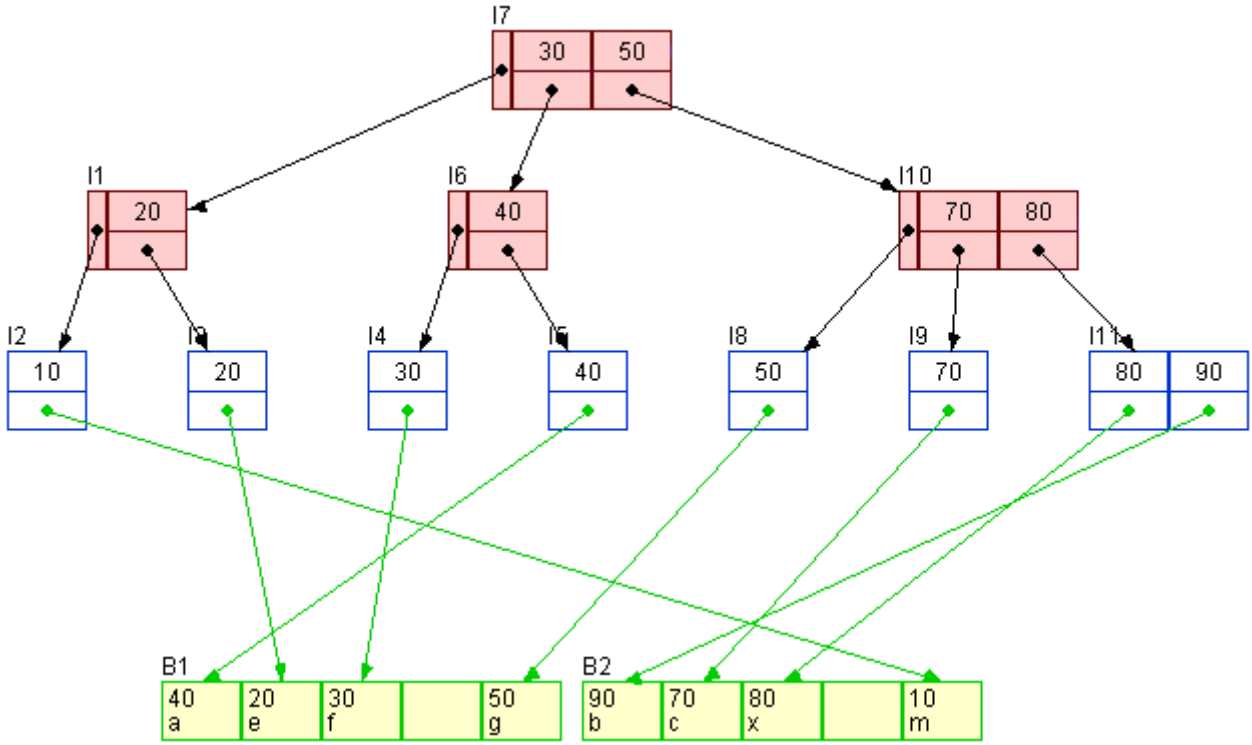
- tree.update()
- tree.updateBase()

Nejprve bude popsána metoda *tree.update()*, která se nachází ve třídě *AbstrTree*. Pseudokód metody *tree.update()* je následující:

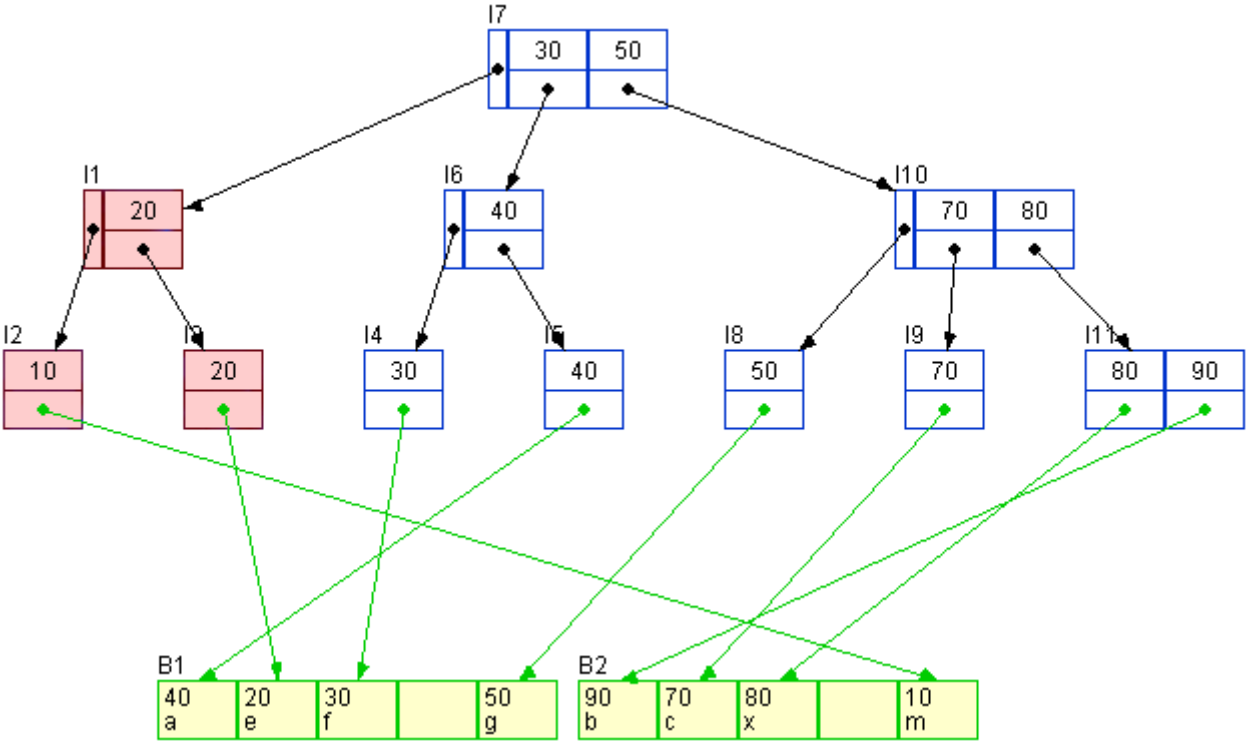
```
1. ArrayList listUmisteni1 = getListUmisteni()
2. UpdateRozlozeni()
3. UpdatePadding()
4. UpdateMinPadding()
5. ArrayList listUmisteni2 = getListUmisteni()
6. VytvorPosuny(listUmisteni1, listUmisteni2)
```

Tento kód ve zkratce znamená, že se na začátku uloží umístění všech bloků na obrazovce, poté se zavolají tři metody update, jejichž výsledkem je správná grafická reprezentace Souboru s úplným indexem, poté se opět získá seznam umístění všech bloků na obrazovce a nakonec se z těchto dvou seznamů vytvoří příkazy typu *Posun* a vloží se do animační fronty.

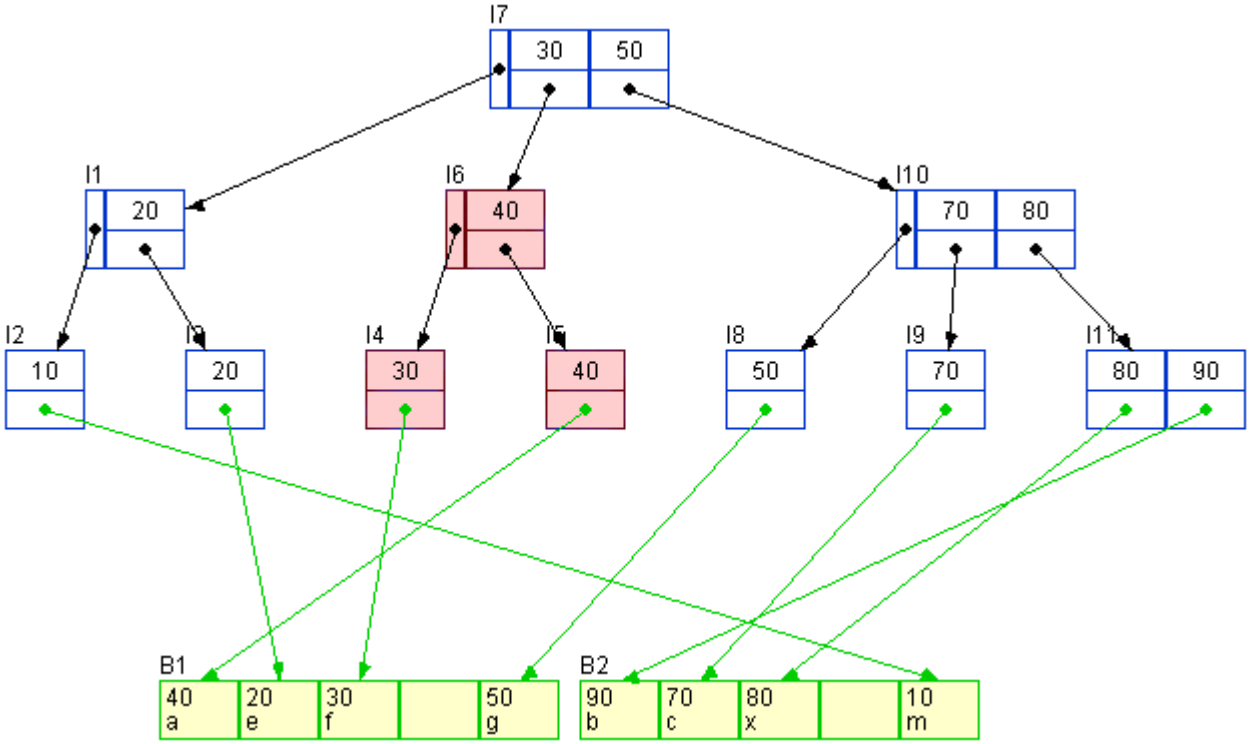
Na následujících obrázcích je názorně uveden princip metody *updateRozlozeni()*. V této metodě se projde strom rekurzivně a všechny uzly jsou centrovány vůči svému otci. Princip proporcionálního rozložení vůči svému otci byl již podrobněji znázorněn na obrázcích 53 a 54. Na obrázku 68 je zobrazen první průchod metodou, kdy jsou uzly I1, I6 a I10 vycentrovány vůči svému otci, jímž je uzel I7. Na obrázku 69 je zobrazen druhý průchod metodou, kdy se uzly I2 a I3 vycentrují vůči uzlu I1. Na obrázku 70 je zobrazen třetí průchod metodou, kdy se uzly I4 a I5 vycentrují vůči uzlu I6 a nakonec na obrázku 71 je ukázáno, jak jsou uzly I8, I9 a I11 vycentrovány vůči uzlu I10.



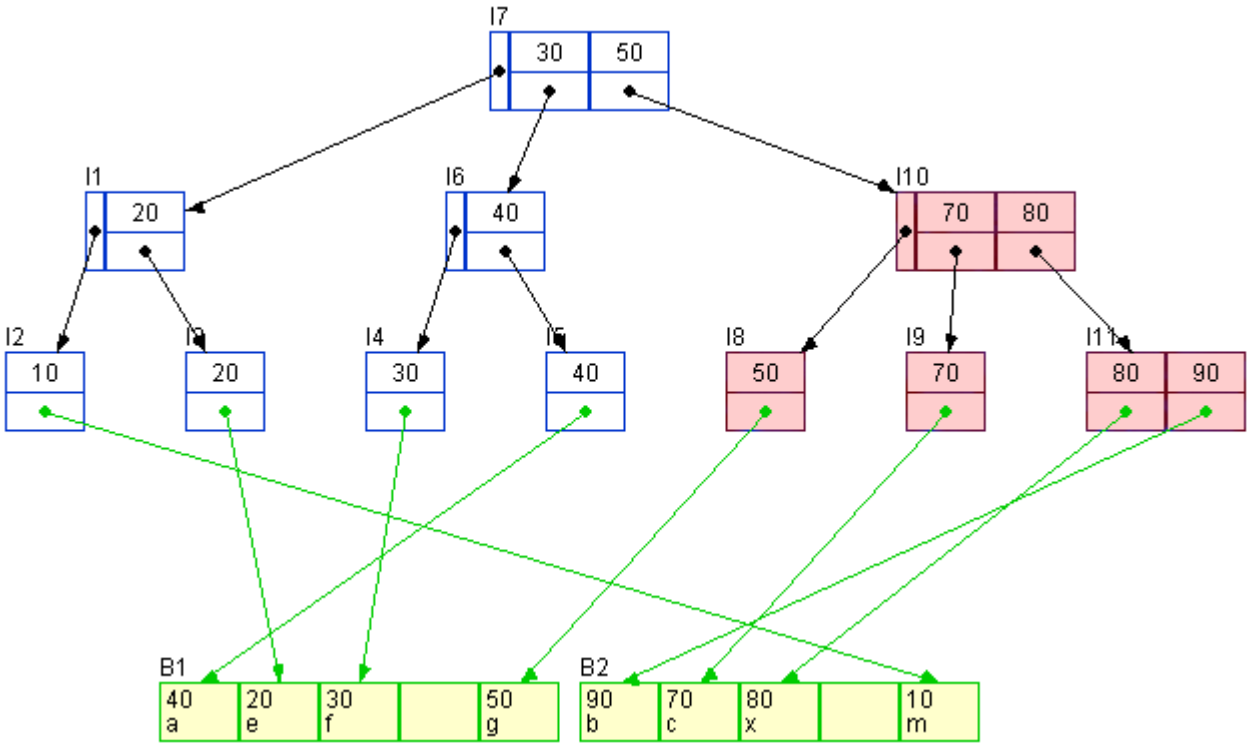
Obr. 68: První průchod metodou updateRozlozeni(). Zdroj: vlastní.



Obr. 69: Druhý průchod metodou updateRozlozeni(). Zdroj: vlastní.



Obr. 70: Třetí průchod metodou updateRozlozeni(). Zdroj: vlastní.

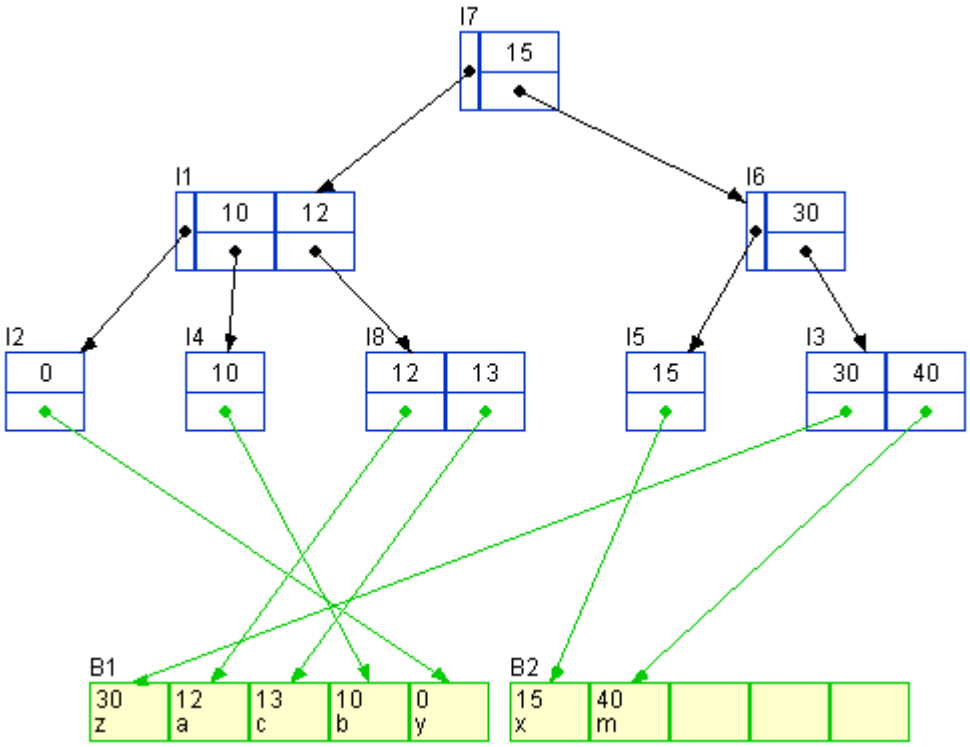


Obr. 71: Čtvrtý průchod metodou updateRozlozeni(). Zdroj: vlastní.

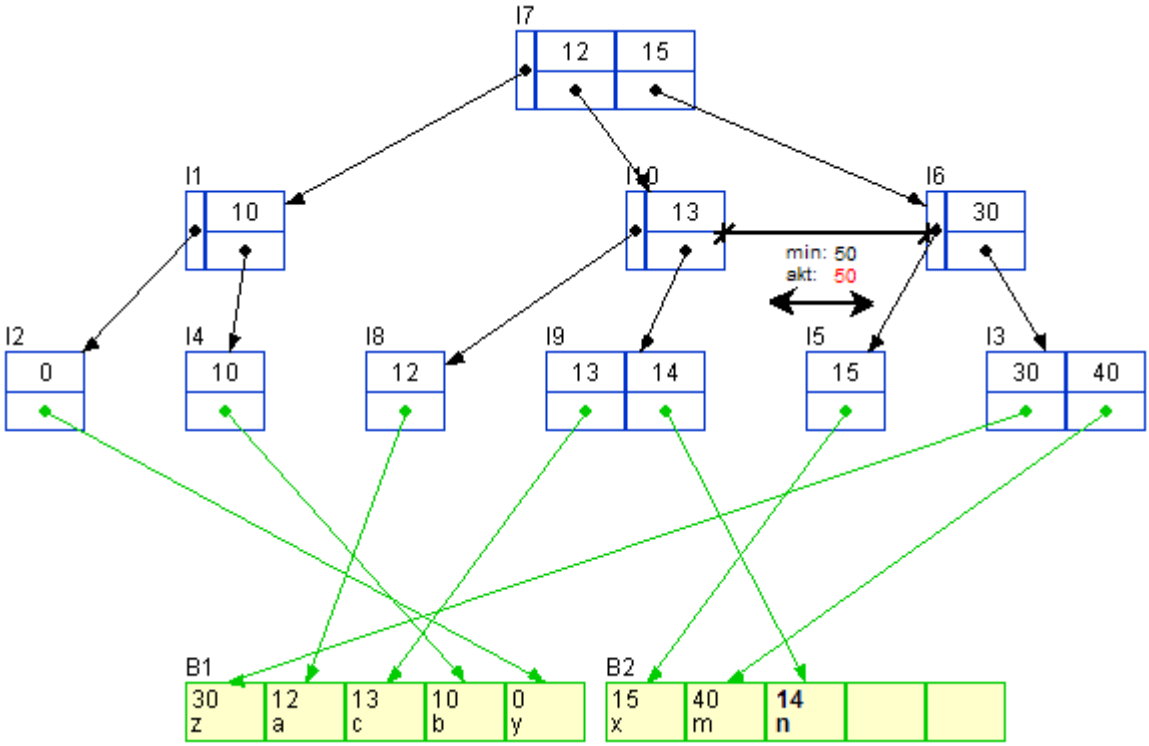
Samotná metoda *updateRozlozeni()* nedokáže udržet strukturu stromu za všech okolností. Rozložení stromu je velkou mírou ovlivněno aktuálním paddingem jednotlivých bloků. Aktuální padding se udržuje takový, aby se nejvíce blížil minimálnímu paddingu. Existují ovšem situace, kdy je nutné aktuální padding zvýšit. To má na starosti metoda *updatePadding()*. Tato metoda prochází strom po úrovních od kořenu po listy a při každém průchodu hledá možnost zvýšení aktuálního paddingu. Pokud najde takovou situaci, ve které je vhodné zvýšit aktuální padding, poté jej zvýší. Tato situace nastává např. při vkládání do Souboru s úplným indexem.

Na obrázku 72 je stav před vložením záznamu s klíčem 14. Tento záznam se vloží do bloku I8, který přeteče a vytvoří se blok I9, do nějž se přesunou záznamy 13 a 14. Do bloku I1 se vloží medián nového bloku, jímž je záznam s klíčem 13. Tento blok opět přeteče a vyalokuje se blok I10, do nějž se přesune záznam s klíčem 13. Nově vyalokovaný blok I10 má hodnotu *aktPadding* rovnu hodnotě *minPadding*, což je evidentně nekorektní. Tato situace je zachycena na obrázku 73. Pro napravení této situace se vypočítá nová hodnota *aktPadding* a chybový stav se tak vyřeší. Výsledek je zobrazen na obrázku 74.

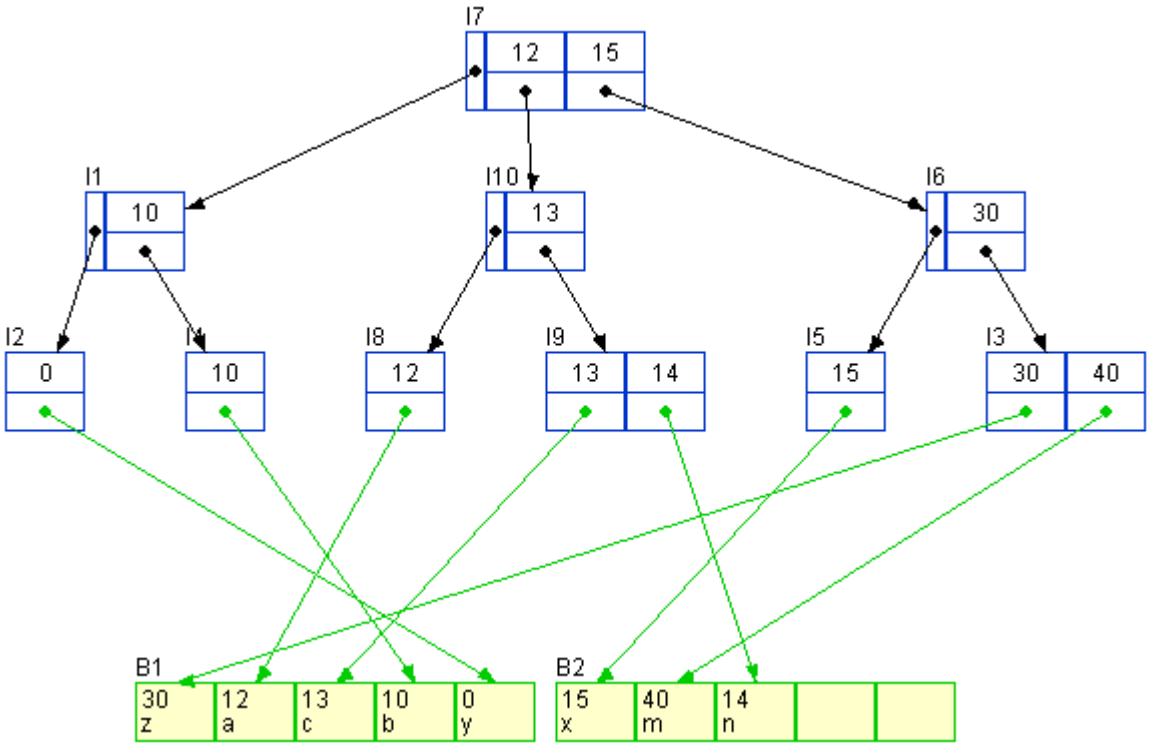
Pro upřesnění je nutno zmínit, že nová hodnota *aktPadding* se vypočítá nejen pro blok I10, ale i pro blok I1. Pro oba bloky se vypočítají takové hodnoty *aktPadding*, aby byla celá úroveň obsahující tyto 2 bloky centrovaná vůči kořenu.



Obr. 72: Výchozí situace před vložením záznamu s klíčem 14. Zdroj: vlastní.



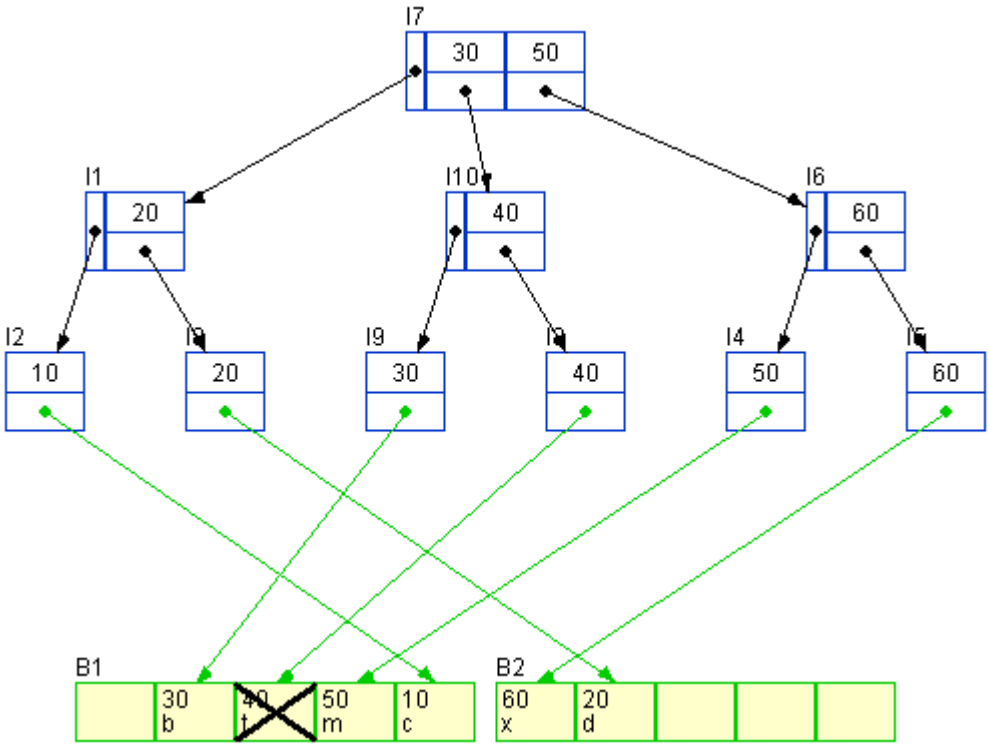
Obr. 73: Situace, která nastane po vložení záznamu s klíčem 14 a následné alokaci bloků I9 a I10. Zdroj: vlastní.



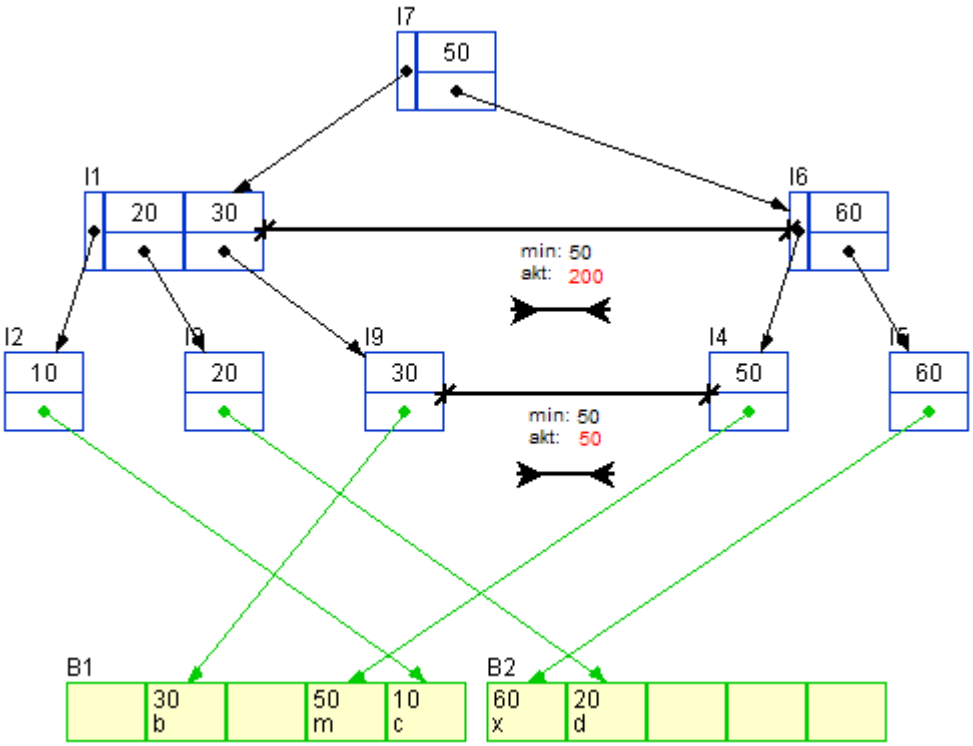
Obr. 74: Výsledný strom po reorganizaci. Zdroj: vlastní.

Zvyšování paddingu nestačí pro udržování struktury stromu. Padding je nutné také v určitých případech, jako například při odebírání záznamu ze stromu, snižovat. Snižování aktuálního paddingu má na starosti metoda *updateMinPadding()*. Tato metoda prochází strom po úrovních obdobně jako metoda *updatePadding()* s tím rozdílem, že strom neprochází od kořene k listům, ale naopak. Na úrovni listů je hodnota *aktPadding* vždy rovna hodnotě *minPadding* a celá poslední úroveň musí být centována vůči kořenu. Díky tomu lze vypočítat hodnoty aktuálního paddingu pro všechny bloky až ke kořenu.

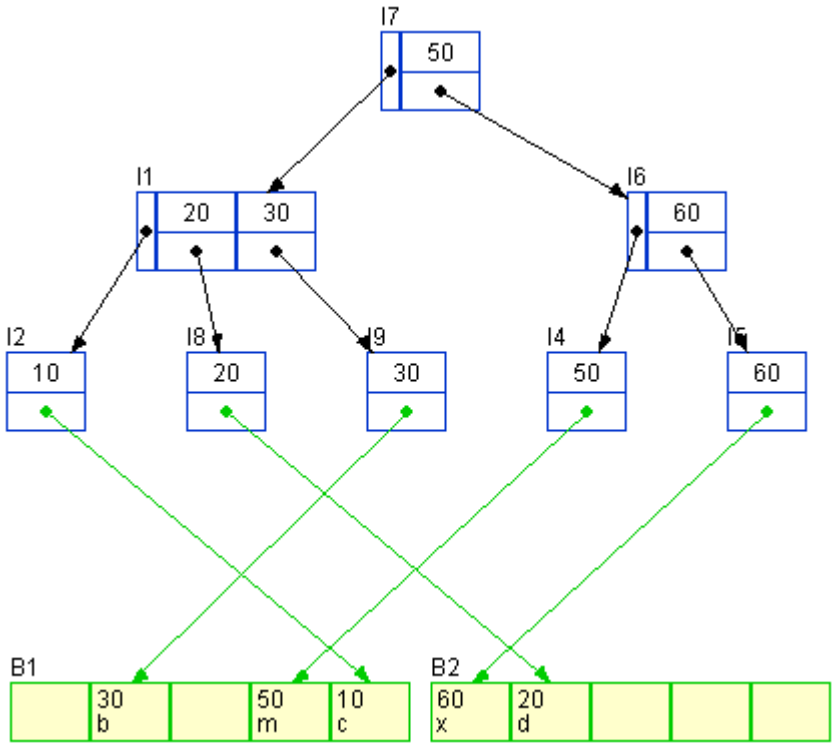
Na obrázku 75 je stav před odebráním záznamu s klíčem 40. Jeho odebrání způsobí „mezeru“ mezi bloky I9 a I4. Následně dojde k podtečení v bloku I10 a tento blok se sloučí s blokem I1. Mezi bloky I1 a I6 ovšem vznikne „mezera“. Tato situace je zobrazena na obrázku 76. Následně se toto chybné uspořádání napraví v metodě *updateMinPadding()*. Výsledek je zobrazen na obrázku 77.



Obr. 75: Výchozí situace před odebráním záznamu s klíčem 40.

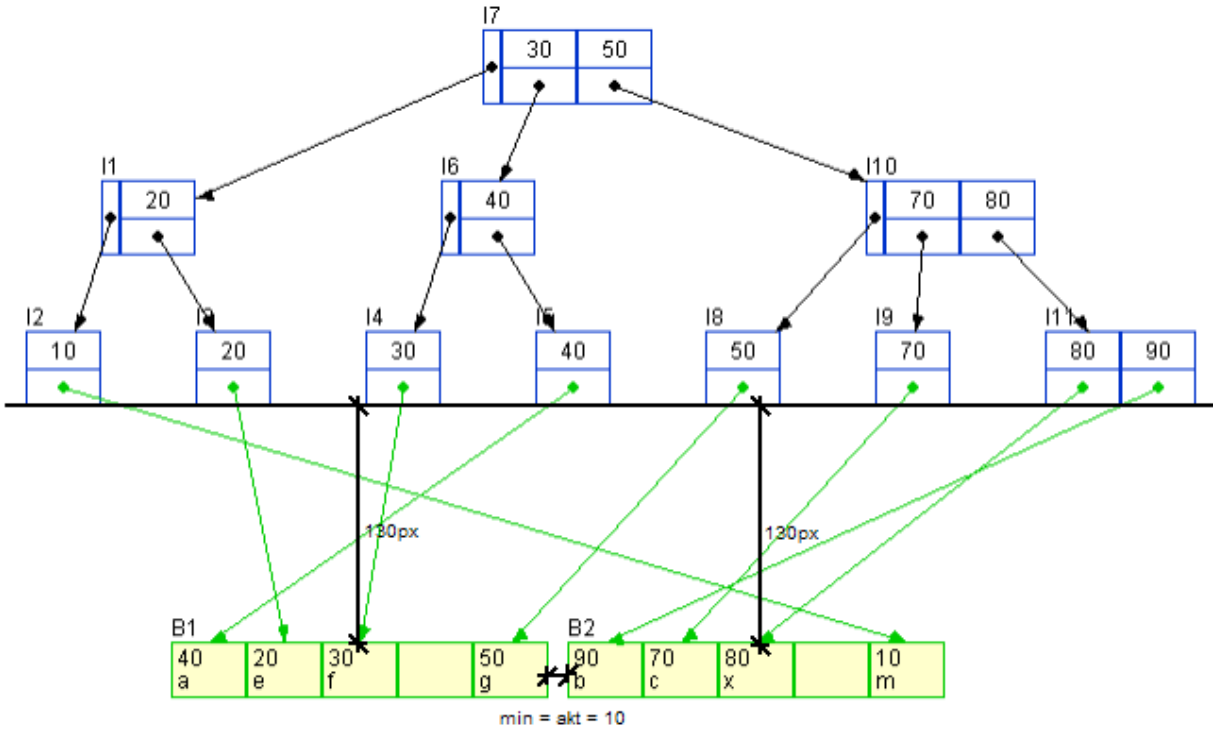


Obr. 76: Situace, která nastane po odebrání záznamu s klíčem 40 a následné dealokaci bloků I3 a I10. Zdroj: vlastní.



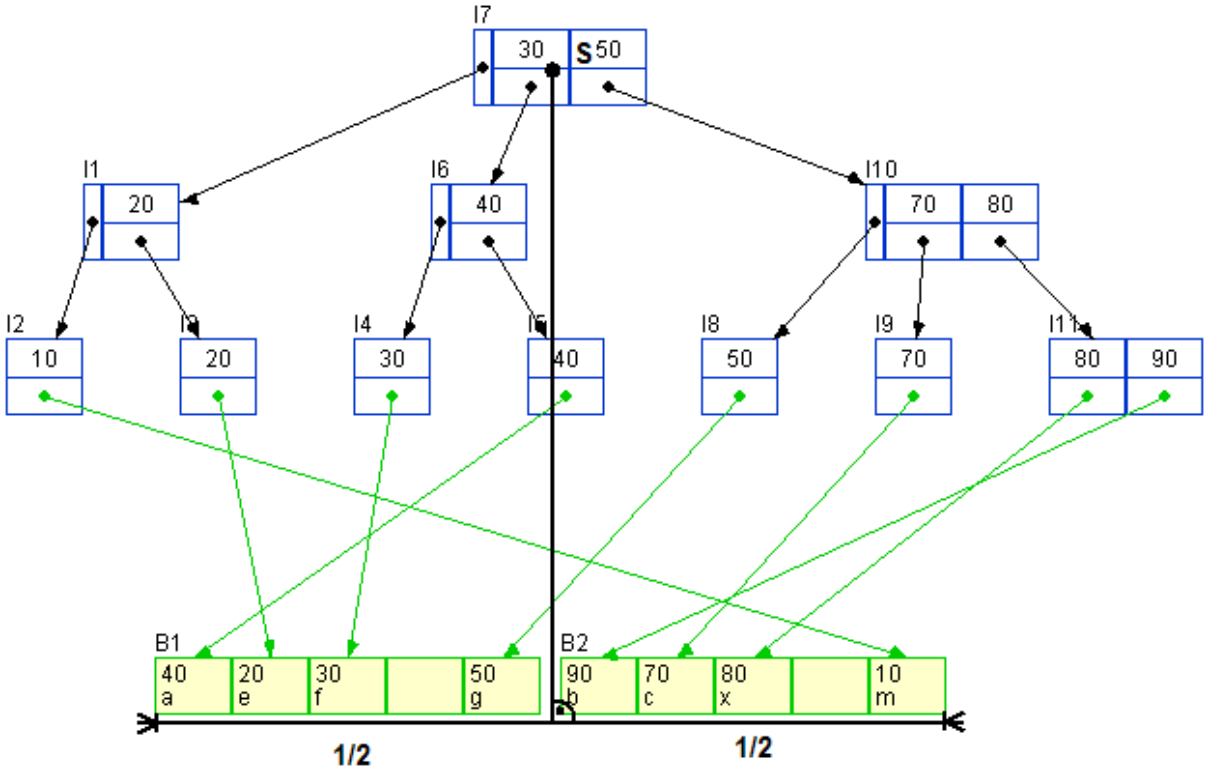
Obr. 77: Výsledný strom po reorganizaci. Zdroj: vlastní.

V předcházejícím textu byla popsána metoda *tree.update()*, která udržuje grafickou strukturu stromu. Následuje popis metody *tree.updateBase()*, která udržuje grafickou reprezentaci datové struktury Halda-soubor. Ta se skládá z bloků, které leží v řadě vedle sebe. Vzdálenost mezi těmito bloky je stanovena jako konstanta a byla zvolena hodnota 10 pixelů. Tato hodnota je uložena jak v atributu *minPadding*, tak i v atributu *aktPadding*. Mezi listy stromu a bloky halda-souboru se udržuje konstantní vzdálenost 130 pixelů. Tato vzdálenost slouží k většímu oddělení těchto dvou struktur. Na obrázku 78 je ilustrace vzdáleností bloků v haldě i vzdálenost od listů stromu.



Obr. 78: Ilustrace vzdáleností bloků v halda-souboru. Zdroj: vlastní.

Podobně jako u stromu jsou listy vycentrovány vůči kořenu a jednotlivé uzly jsou centrovány vůči svým otcům, bloky v haldě jsou vycentrovány vůči středu kořenu. Ilustrace tohoto principu je zobrazena na obrázku 79.

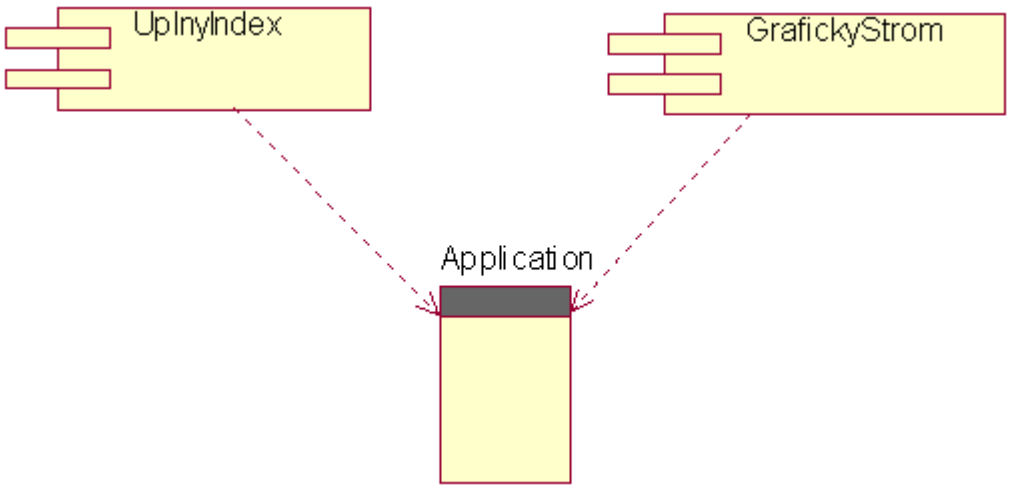


Obr. 79: Ilustrace rozvržení halda-souboru vůči kořenu. Zdroj: vlastní.

4.2.14 Vztah mezi Úplným indexem a jeho grafickou reprezentací

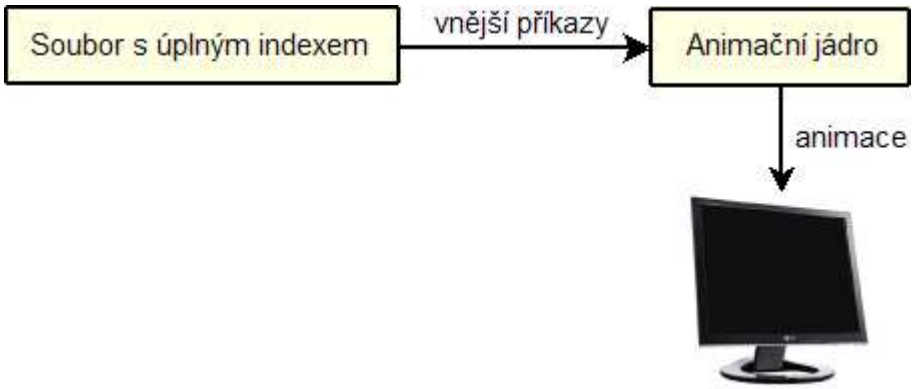
Datová struktura reprezentující Soubor s úplným indexem a její grafická reprezentace jsou dvě rozdílné části, které jsou na sobě nezávislé. Soubor s úplným indexem lze použít bez jeho grafického výstupu a grafický strom lze také aplikovat i na jiné stromy než pouze Soubor s úplným indexem.

Tyto dvě komponenty jsou spojeny ve výsledné grafické aplikaci, jak je zobrazeno na obrázku 69. Při vykonání nějaké operace, se nejprve tato operace provede v Úplném indexu, kde se zároveň shromažďují data o průběhu operace. Poté se tyto data vloží do grafického stromu, který z nich vytvoří animaci.



Obr. 80: Vztah Souboru s úplným indexem a jeho grafické reprezentace. Zdroj: vlastní.

Na následujícím obrázku je zobrazen vztah mezi datovou strukturou Soubor s úplným indexem a jádrem animace. Vnější příkazy, které jsou výsledkem operací Vlož, Odeber nebo Najdi v Souboru s úplným indexem, se předají do animačního jádra, jsou následně zpracovány a výsledek se zobrazí na monitoru.



Obr. 81: Jiné zobrazení vztahu mezi komponentami. Zdroj: vlastní.

4.2.15 Rozšíření grafické aplikace

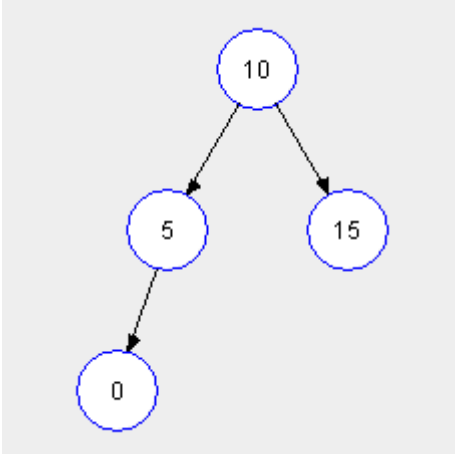
Vzhledem k důslednému oddělení implementace Souboru s úplným indexem a jeho grafické reprezentace je možné vytvořit po vzoru této aplikace grafické znázornění dalších druhů stromů.

Stromy se dělí na dva druhy:

1. Strom, který má v každém uzlu pouze jeden záznam.
2. Strom, který má v každém uzlu více záznamů.

Grafická reprezentace Souboru s úplným indexem, která byla v této práci řešena, je strom druhého typu. Při odebrání tříd, které zastupují Halda-soubor z třídy *Tree* je možné grafickou aplikaci použít pro animaci jakéhokoliv typu stromu, který obsahuje v každém uzlu záznamy. Typickou aplikací jsou všechny druhy B-stromů.

Pro podporu stromů, které jsou prvního typu, je k dispozici rozšíření grafického stromu, jehož grafický výstup je na obrázku 82. S tímto druhem zobrazení je možné vytvořit jakoukoliv animaci stromu, který obsahuje v každém uzlu pouze jeden záznam. Zdrojové kódy tohoto typu zobrazení stromu jsou na CD médiu přiloženém k diplomové práci.



Obr. 70 – Jiný druh zobrazení stromu

Pomocí grafické reprezentace k-cestných stromů, která byla jako součást diplomové práce navrhnuta a implementována je možné vytvořit všechny druhy stromů, mezi něž patří:

- binární strom,
- k-cestný strom,
- B-strom a jeho variace,
- splay tree,
- Red-Black tree,
- AVL-strom,
- treap a další.

5. Použití v praxi

Databáze je souhrn dat organizovaných tak, aby umožňovaly vkládání, odebírání, upravování a zpřístupňování dat. V dopravě mají databáze, jako prostředek pro uchovávání velkého množství dat, výraznou roli. Velké množství systémů v dopravě uchovává svá vnitřní data v databázi, nebo s jinými databázovými systémy spolupracují [32].

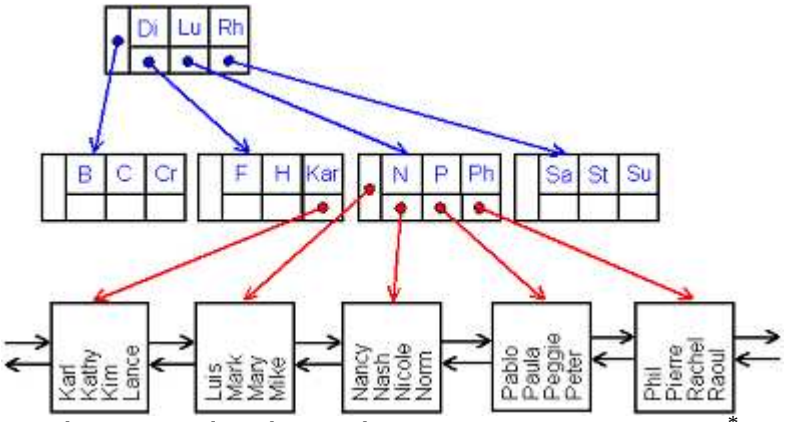
Nejpoužívanějšími databázovými systémy jsou v současnosti Oracle, MS-SQL, MySQL a PostgreSQL. Data získaná z databáze MS-SQL jsou např. použita pro tvorbu binárních souborů systému IDOS [36]. Jádro systému CEVIS (Centrální vozový informační systém) [32] je postaveno na databázi INFORMIX a přímo spolupracuje s dalšími systémy, které jsou také postaveny na databázovém serveru INFORMIX, nebo Oracle. Systém CDS (Centrální dispečerský systém) [32] je postaven na databázovém systému Oracle.

Databáze může obsahovat obrovské množství dat. Aby byla databáze použitelná v praxi, musí zvládat požadované operace v krátkém čase. Protože data nemohou být uložena celé v operační paměti, musí být uložena na nějakém externím médiu, jako je např. harddisk. Pro rychlé operace s daty na externím paměťovém médiu se používají B-stromy, tudíž je logické použití B-stromů v databázích.

B-stromy se v databázových systémech používají v indexech. Index je struktura přidružená k tabulce dat, která slouží pro rychlý přístup k datům. Nad jednou tabulkou může být několik indexů. Indexy jsou logicky a fyzicky nezávislé od dat v asociované tabulce. Je možné kdykoli vytvořit nebo zrušit index bez ovlivnění bázových dat nebo ostatních indexů. Index tvoří pouze rychlý přístup k hledaným položkám.

B-stromy nejsou jediným způsobem indexování záznamů. Pro indexování se používají také hašovací tabulky a bitmapové indexy. B-strom je ale oproti jiným způsobům univerzální a používá se implicitně.

Následující obrázek zobrazuje část databáze s indexem typu B-strom v databázovém systému Oracle:



Obr. 71 –Část databáze v databázovém systému Oracle s indexem typu B-strom. Zdroj: [11].*

Oracle používá v indexech B*-stromy [7]. Bázový soubor obsahuje tabulku, ve které jsou data uspořádána dle klíče vzestupně. Klíče jsou zde celé. Navíc mezi jednotlivými bloky jsou obousměrné směrníky. V indexovém souboru jsou na pozici klíčů minimální části klíče potřebné pro jednoznačné rozhodnutí mezi dvěma klíči [11].

6. Závěr

Cíl diplomové práce, jímž bylo navrhnout a implementovat datovou strukturu Soubor s úplným indexem a vytvořit její grafickou reprezentaci s animací, byl splněn. Animace Souboru s úplným indexem byla navíc navrhnuta pro budoucí rozšíření o animaci dalších druhů k-cestných stromů.

K diplomové práci je přiloženo několik aplikací a jejich zdrojových kódů. První aplikací je implementace Souboru s úplným indexem bez podpory její grafické reprezentace. Tato aplikace je vhodná pro praktickou implementaci datové struktury Soubor s úplným indexem pro rychlé zpřístupnění dat uložených na vnějším paměťovém médiu.

Další aplikací je implementace Souboru s úplným indexem s animací. Implementace této aplikace byla náplní diplomové práce. Aplikaci lze spustit přímo z CD, nebo může být umístěna na Internetu, kde lze spustit pomocí technologie Java Web Start. Tato aplikace je vhodná pro výukové účely při prezentaci datové struktury Soubor s úplným indexem.

Poslední aplikací je rozšíření grafické reprezentace Souboru s úplným indexem pro podporu animace jiných druhů stromů, nejen datové struktury Soubor s úplným indexem.

7. Literatura

[1] CENEK, Petr, KLIMA, Valent, JANÁČEK, Jaroslav. *Optimalizace dopravních a spojových procesů*, Vysoká škola dopravy a spojov v Žiline, 1994, ISBN 80–7100–197–X

[2] POKORNÝ, Jaroslav. *Databázové systémy a jejich použití v informačních systémech*, Academia, 1992, ISBN 80–200–0177–8

[3] SNÁŠEL, Václav, DVORSKÝ, Jiří. *Algoritmická matematika I.*, Univerzita Palackého v Olomouci, Přírodovědecká fakulta, 1999, ISBN 80–244–0013–8

[4] CORMEN, Thomas, LEISERSON, Charles, RIVEST, Ronald, STEIN, Clifford. *Introduction to Algorithms*, MIT Press, Cambridge MA, 2001, ISBN 0–262–03293–7

[5] MILLER, Nancy. *File structures using Pascal*, The Benjamin/Cummings Publishing Company, Inc., 1987, ISBN 0–8053–7082–X

[6] HARBON, Thomas. *File Systems: Structures and Algorithms*, Anderson University, Prentice-Hall, Inc., 1988, ISBN 0–13–314709–6

[7] GILBERG, Richard, FOROUZAN, Behrouz. *Data Structures: A Pseudocode Approach with C++*, Brooks/Cole, 2001, ISBN 0–534–95216–X

[8] THARP, Alan. *File organization and processing*, John Wiley & Sons, Inc., 1988, ISBN 0-471-60521-2

[9] FOLK, M., J. *File structures: An Object Oriented Approach with C++*, Addison Wesley, Boston, 1997, ISBN 0-201-87401-6

[10] ECKEL, Bruce. *Myslíme v jazyku Java, knihovna programátora*, Grada, 2001, 432s, ISBN 80-247-9010-6

[11] ECKEL, Bruce. *Myslíme v Jazyku Java, knihovna zkušeného programátora*, Grada, 2001, 472s, ISBN 80-247-0027-1

[12] SPELL, Brett. *Java: Programujeme profesionálně*, Computer Press, 2002, 1022s, ISBN 80-7226-667-5

[13] KISZKA, Bogdan. *1001 tipů a triků pro programování v jazyce Java*, Computer Press, 2003, 520s, ISBN 80-7226-989-5

[14] FAN, Joel, RIES, Eric, TENITCHI, Calin. *Black Art of Java Game Programming*, Waite Group Press, 1996, 900s, ISBN 1-571-69043-3

[15] DAVISON, Andrew. *Killer Game Programming in Java*, O'Reilly, 2005, 1008s, ISBN: 0-596-00730-2

[16] ARLOW, Jim, NEUSTADT, Ila, *UML a unifikovaný proces vývoje aplikací*, Computer Press, 2005, 376s, ISBN 80-7226-947-X

[17] SCHMULLER, Joseph. *Myslíme v jazyku UML*, Grada, 2001, 360s, ISBN 80-247-0029-8

[18] SHIRAZI, Jack. *Java Performance Tuning*, O'Reilly, 2003, 600s, ISBN 0-596-00377-3

[19] JANNINK, Jan. *Implementing Deletion in B^+-Trees* [online]. 2002 [cit. 2006-05-15]. Dostupný z www: <http://www-db.stanford.edu/~jan/btree/btree.ps>

[20] *Windows Server 2000 Resource Kit: Chapter 3 - File Systems* [online]. 2000 [cit. 2006-05-15]. Dostupný z www: <http://www.microsoft.com/resources/documentation/windows/2000/server/reskit/en-us/serverop/part1/sopch03.mspx>

[21] *Oracle9i Database Concepts* [online]. 2004 [cit. 2006-05-15]. Dostupný z www: <http://download-west.oracle.com/docs/cd/A91202_01/901_doc/server.901/a88856/c11schem.htm>

[22] *Animace B^+-stromu* [online]. 2006 [cit. 2006-05-15]. Dostupný z www: <http://www.seanster.com/BplusTree/BplusTree.html>

[23] *Animace B-stromu* [online]. 2006 [cit. 2006-05-15]. Dostupný z www: <http://slady.net/java/bt/>

[24] *Rychlosti otáček pevných disků* [online]. 2006 [cit. 2006-05-22]. Dostupný z www: <http://www.svethardware.cz/art_doc-54F5BC8BF60F569BC1257111007360A4.html>

[25] *Přístupové doby pevných disků* [online]. 2006 [cit. 2006-05-22]. Dostupný z www: <http://www.svethardware.cz/art_doc-908DD7A56D13E69DC12571110073F5A4.html>

[26] *Přístupové doby operační paměti (paměti RAM)* [online] 2005 [cit. 2006-05-22]. Dostupný z www: <http://www.svethardware.cz/art_doc-6964EE3A72AF8B27C12570AA005E1277.html>

[27] *Paměti a jejich organizace* [online]. 2006 [cit. 2006-05-23]. Dostupný z www: <http://www.student.cvut.cz/cwut/index.php/Pam%C4%9Bti_a_jejich_organizace>

[28] KAVIČKA, A. *Elektronické slajdy k přednáškám,* 2005

[29] KRÁTKÝ, Michal. DVORSKÝ, Jiří. *Nelineární datové struktury IV*. [online]. 2005 [cit. 2006-05-23]. Dostupný z www: <http://www.cs.vsb.cz/kratky/courses/2004-05/udp/presentation/udp-10_6.pdf>

[30] RAFIEI, Davood. *Data organization on files.* [online]. 2006 [cit. 2006-05-24]. Dostupný z www: <http://www.cs.ualberta.ca/~drafiei/291/notes/Files-4.pdf>

[31] *IBM Informix Dynamic Server Administrator's Reference: Disk Structures and Storage* [online]. 2005 [cit. 2006-05-27]. Dostupný z www: <http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.adref.doc/adref235.htm>

[32] ŠOTEK, Karel. *Výpočetní technika a informatika v dopravě*, Univerzita Pardubice, 1999, 245s., ISBN 80-7194-230-8

[33] *API třídy JComponent* [online]. 2003 [cit. 2006-05-24]. Dostupný z www: http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JComponent.html

[34] *API třídy Container* [online]. 2003 [cit. 2006-05-24]. Dostupný z www: http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Container.html

[35] *API třídy Component* [online]. 2003 [cit. 2006-05-24]. Dostupný z www: http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Component.html

[36] *Základní informace o systému IDOS* [online]. 2003 [cit. 2006-05-28]. Dostupný z www: http://www.chaps.cz/idos.asp

[37] *MS-SQL: Extensible Storage Engine Architecture* [online]. 2006 [cit. 2006-05-28]. Dostupný z www: http://www.microsoft.com/technet/prodtechnol/exchange/guides/E2k3TechRef/764d8347-a99b-408a-a774-f1263797c3b0.mspx?mfr=true

Příloha A: Uživatelská příručka

Aplikaci lze spustit pomocí souboru *spustit.bat* na disku CD, nebo prostřednictvím Internetu kliknutím na odkaz, který ukazuje na spustitelný soubor s příponou *jnlp*. V prohlížeči Internet Explorer se aplikace spustí automaticky. V jiných prohlížečích může být uživatel informován o nutnosti potvrzení spuštění tohoto souboru.

V případě spouštění aplikace pomocí webového prohlížeče se automaticky spustí aplikace Java Web Start, která je na následujícím obrázku. Tato aplikace stáhne na disk a spustí aplikaci Úplný index. Aplikace má velikost 1,07 MB. Při prvním spuštění aplikace se data stáhnou a uloží na disk. Při dalším spouštění se už znovu z Internetu nestahují.

Obr. A1: Stahování aplikace z Internetu

Při spuštění aplikace bude uživateli prezentován dialog, který je na následujícím obrázku, který musí uživatel potvrdit. Je vhodné zaškrtnout *„Always trust content from this publisher“*. Po zaškrtnutí tohoto políčka se už tento dialog při příštím spuštění aplikace nezobrazí. Tento dialog se zobrazí z toho důvodu, že aplikace bude moci při svém běhu přistupovat na disk počítače při práci s indexovým a bázovým souborem v Souboru s úplným indexem.

Obr. A2: Potvrzení spuštění aplikace

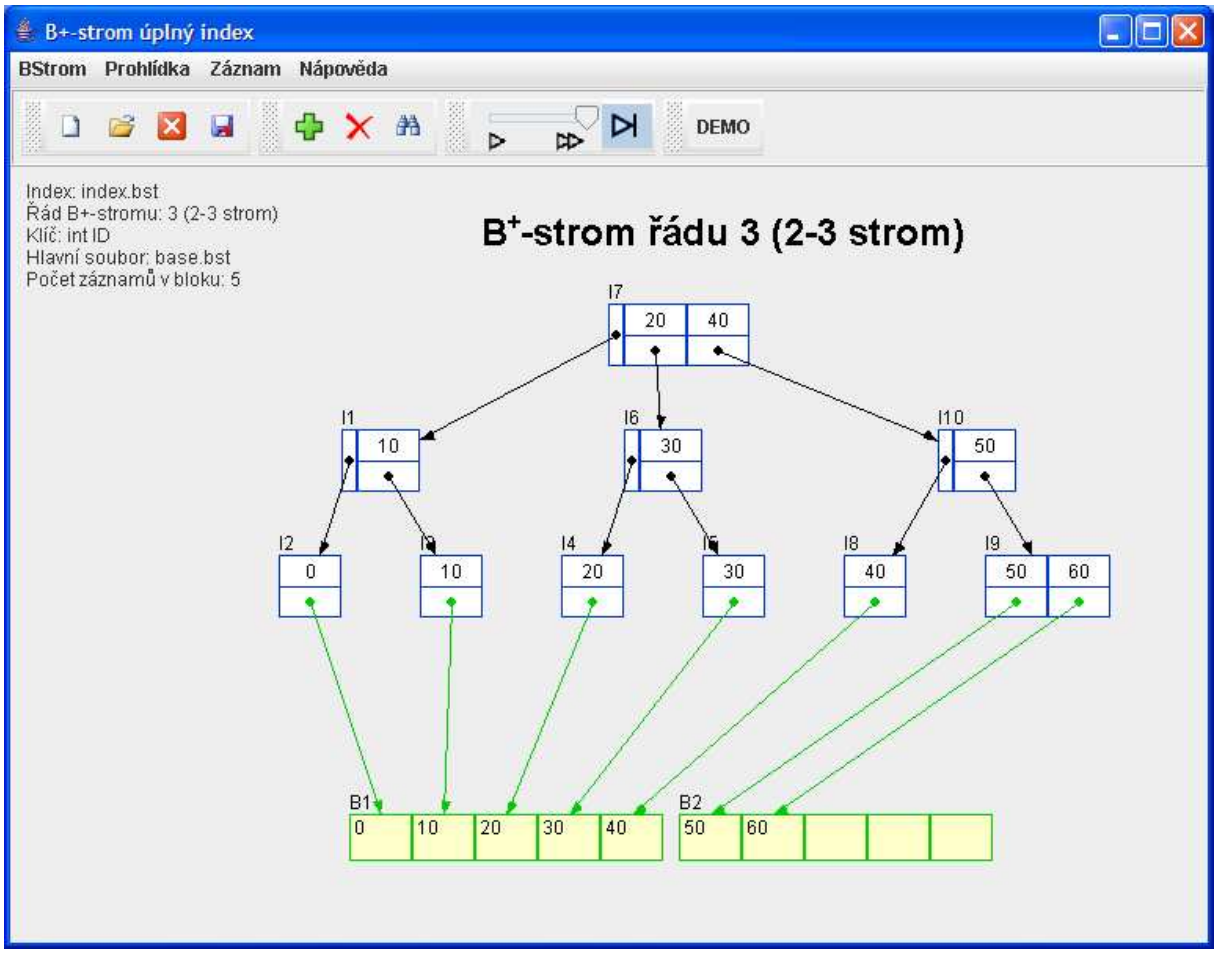
Po spouštění aplikace pomocí Internetového prohlížeče je možné okno tohoto prohlížeče zavřít, protože prohlížeč funguje pouze jako prostředník pro spuštění Java aplikace a při běhu této aplikace již není využíván.

Při práci s aplikací je nutné ukládat soubory, které používá Soubor s úplným indexem, na místo, kde má uživatel práva měnit soubory. V případě, že aplikace nezíská tyto práva, bude o tom uživatel informován chybovou hláškou.

Na obrázku A3 je screenshot běžící aplikace. Na následujících stránkách je podrobně popsáno ovládání aplikace. Pro běh aplikace je důležité mít nainstalováno prostředí *Java Runtime Edition*. Toto prostředí je možné stáhnout na stránkách firmy Sun. V době psaní diplomové práce je k dispozici pro stáhnutí tohoto prostředí následující adresa:

```
http://java.com/en/download/index.jsp
```

Aplikace byla naprogramována a odzkoušena ve verzi Javy 1.5.0. Aplikace by měla být funkční ve verzích Javy 1.4.2 a vyšší, včetně budoucích verzí.



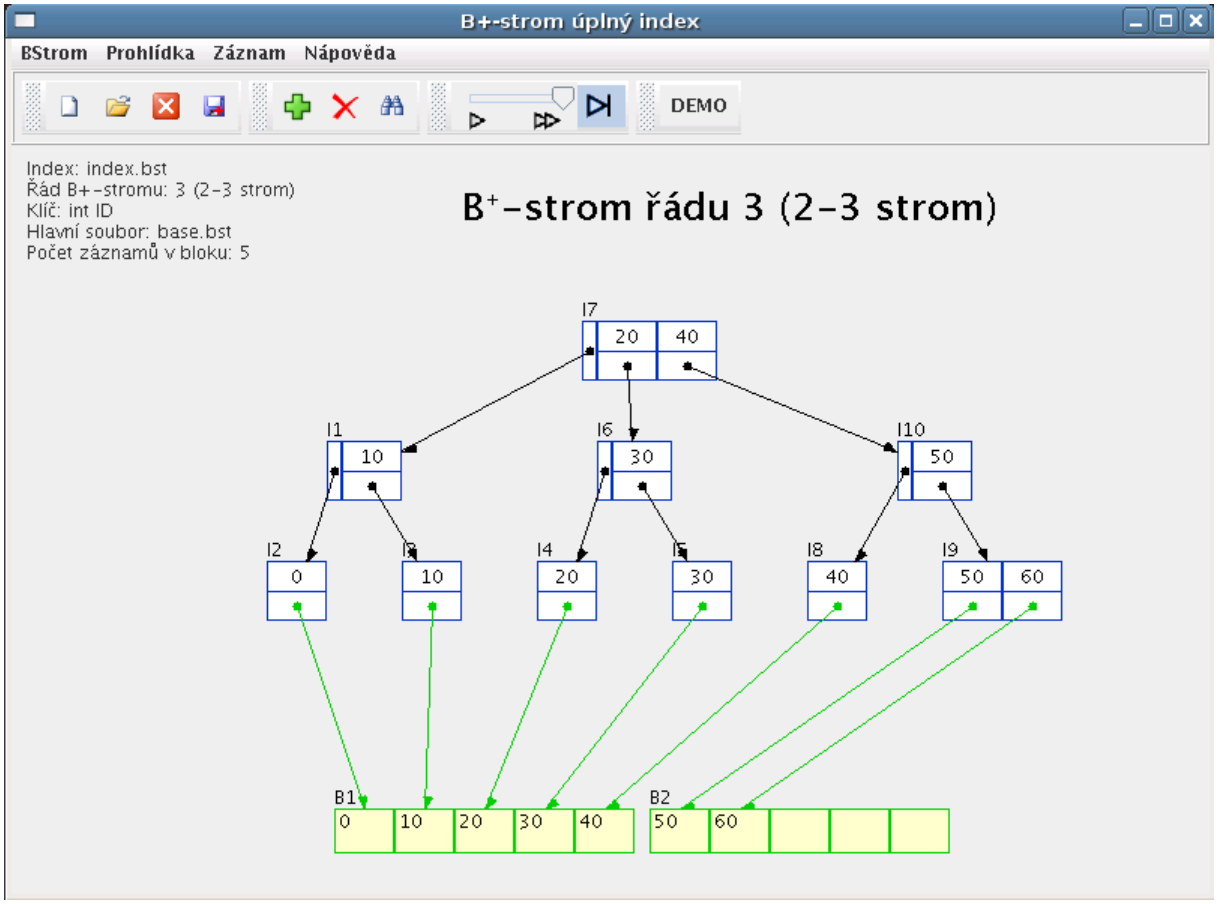
Obr. A3: Obrázek běžící aplikace

Je vhodné mít v jeden okamžik spuštěnou pouze jednu instanci aplikace, protože v případě, že by byly spuštěné dvě instance a každá by měla Soubor s úplným indexem, který by se skládal z *index.bst* a *base.bst*, pak by tyto dvě aplikace zároveň měnily stejné soubory a docházelo by k nepředvídatelnému chování aplikace.

Vzhledem k tomu, že je Java multiplatformní, běží aplikace i na jakémkoliv operačním systému, na kterém je nainstalováno prostředí Java Runtime Edition. V době psaní diplomové práce jsou společností Sun podporovány následující operační systémy:

- Windows,
- Linux,
- Solaris.

Na následujícím obrázku je zachycena aplikace běžící na operačním systému Linux.



Obr. A4: Běžící aplikace na OS Linux

Na všech operačních systémech má aplikace vzhled téměř stejný. Pro spuštění se také používá stejný spustitelný *jar* soubor.

Pro spuštění desktop aplikace pod OS Linux je vhodné použít spustitelný soubor *start.sh*. Pokud tento soubor nelze spustit, potom je třeba ho označit jako spustitelný pomocí příkazu:

```
chmod +x start.sh
```

I. Ovládání aplikace

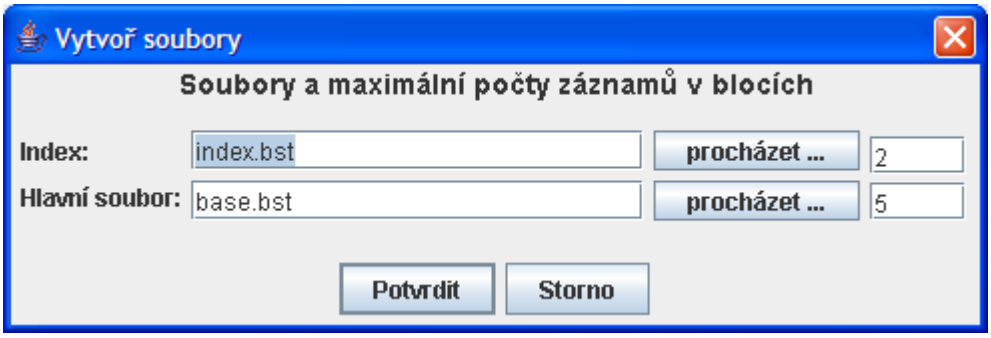
Pro ovládání aplikace je k dispozici menu, panel nástrojů a klávesové zkratky. V menu jsou všechny operace, které lze s aplikací provádět kromě řízení rychlosti animace. To je možné provádět pouze v panelu nástrojů.

Menu BStrom

Generování náhodného indexu (Ctrl-G)

Tato operace vytvoří strom a vygeneruje náhodně záznamy, které vloží do stromu. Postupně se zobrazí tři dialogy pro zadání vstupních údajů.

V prvním dialogu se zadá název indexu a hlavního souboru a maximální počet záznamů v blocích těchto souborů.

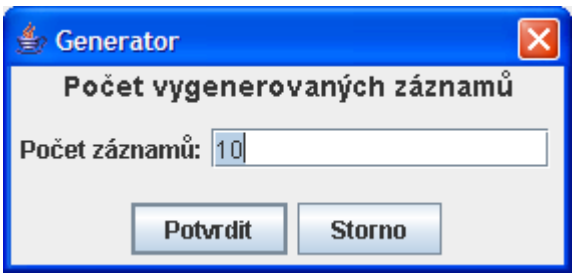


Obr. A5: Dialog pro tvorbu indexu a hlavního souboru

V dalším dialogu se vybere typ klíče. K dispozici jsou dvě varianty: Buď bude klíč typu Integer, nebo typu String.

Obr. A6: Dialog pro výběr typu klíče

V posledním dialogu se zadá počet vygenerovaných záznamů. Maximální počet vygenerovaných záznamů je 15 záznamů.

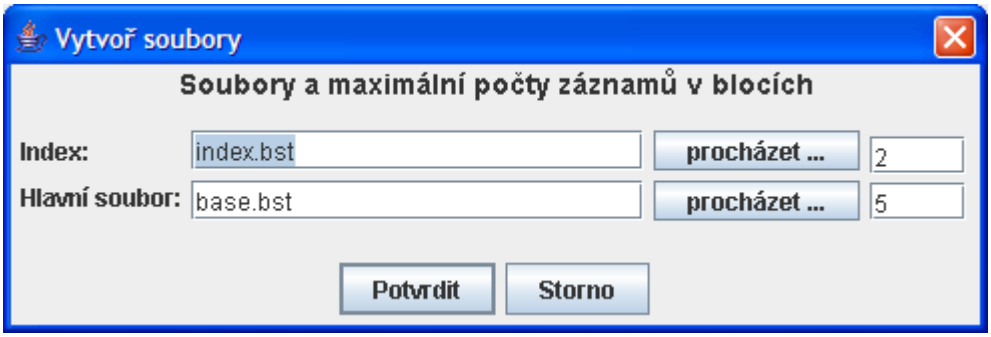


Obr. A7: Dialog pro zadání počtu vygenerovaných záznamů

Vytvoření nového Souboru s úplným indexem (Ctrl-N)

Tato operace vytvoří nový prázdný Soubor s úplným indexem. Postupně se zobrazí dva dialogy pro zadání vstupních hodnot. Tyto dialogy jsou stejné jako první dva dialogy v operaci generování Souboru s úplným indexem.

V prvním dialogu se zadají názvy souborů indexu a hlavního souboru a dále maximální počty záznamů v blocích souborů.



Obr. A8: Dialog pro tvorbu indexu a hlavního souboru

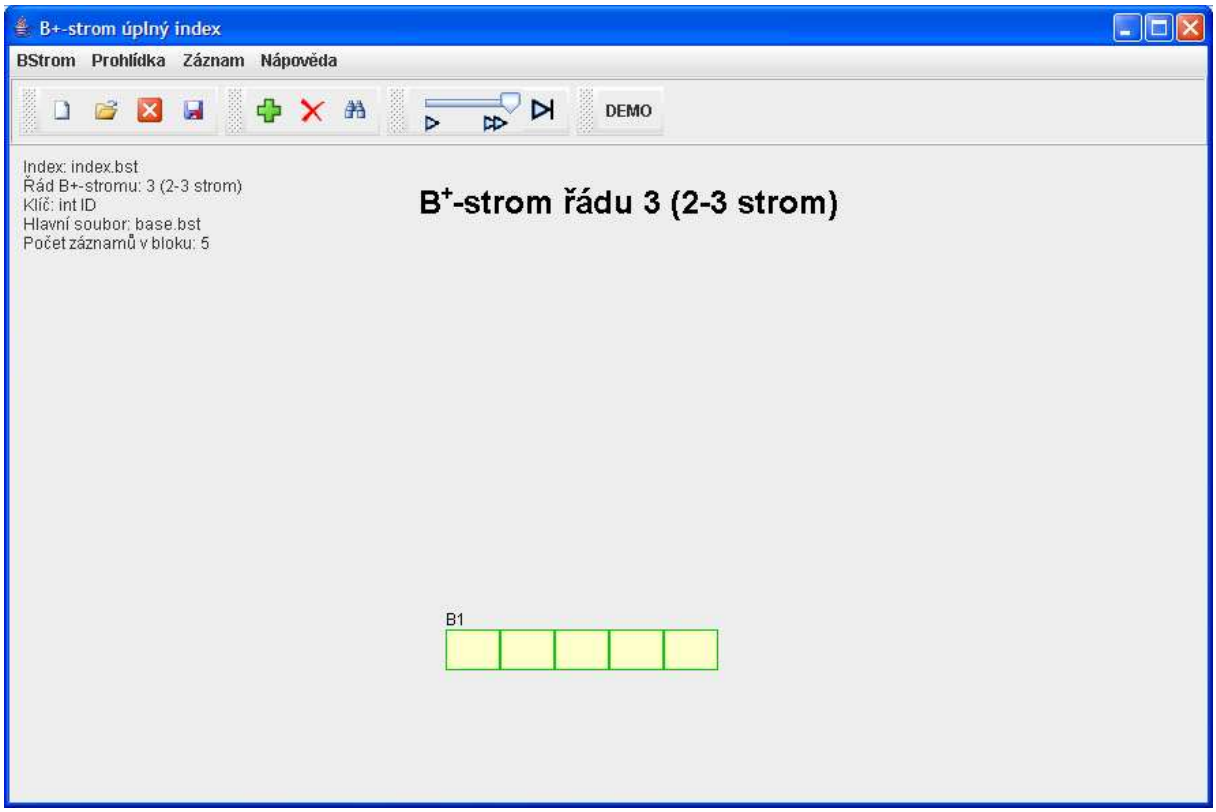
V druhém dialogu se vybere typ klíče.

Obr. A9: Dialog pro výběr typu klíče

Po stisknutí tlačítka Potvrdit se vytvoří prázdný Soubor s úplným indexem. V horní části obrazovky je zobrazen řád B^+-stromu. V horní levé části obrazovky je podrobnější popis, který obsahuje:

- název indexového souboru,
- řád B^+-stromu,
- typ klíče,
- název hlavního souboru,
- maximální počet záznamů v bloku hlavního souboru.

V dolní části obrazovky je blok B1, který je prvním blokem alokovaným v hlavním souboru.

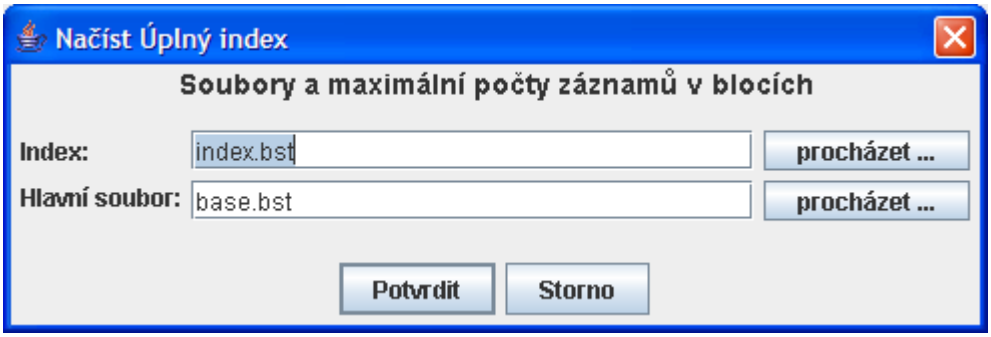


Obr. A10: Nový Soubor s úplným indexem

Načtení stávajícího Souboru s úplným indexem (Ctrl-O)

Tato operace načte index a bázový soubor ze stávajících souborů.

Zobrazí se dialog pro vstup souborů. Pro přímý výběr souborů je možné použít tlačítka „procházet …“.



Obr. A11: Dialog pro načtení souborů

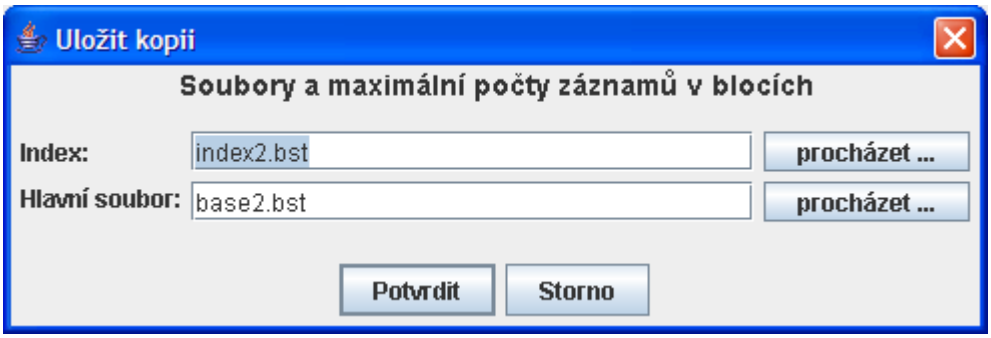
Zrušení stávajícího Souboru s úplným indexem (Ctrl-W)

Tato operace zruší (a uloží) Soubor s úplným indexem.

Uložení kopie Souboru s úplným indexem (Ctrl-S)

Tato operace je částečným ekvivalentem operace „Uložit jako“ s tím rozdílem, že se stávající soubory uloží. Provedou se následující operace na disku:

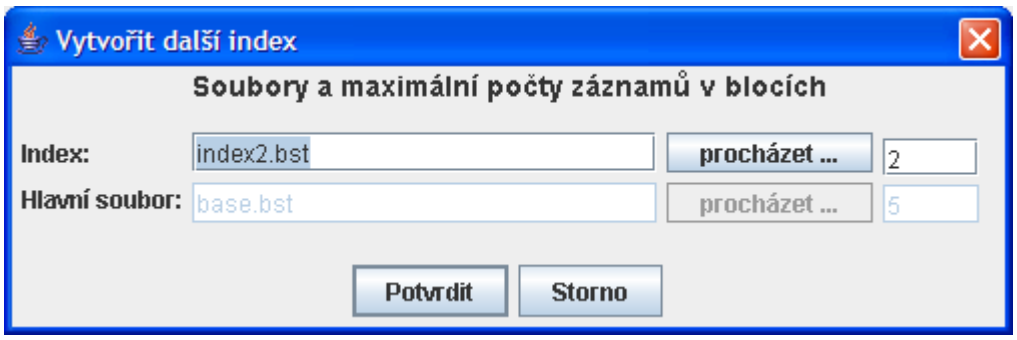
- stávající Soubor s úplným indexem se uloží,
- uložené soubory se zkopírují do nového umístění,
- Soubor s úplným indexem z nového umístění se otevře.



Obr. A12: Dialog pro uložení kopie

Vytvoření druhého přístupového indexu (Ctrl-I)

Tato operace vytvoří druhý přístupový index. Nejprve se vybere v dialogu, který je zobrazen na následujícím obrázku, nový indexový soubor.



Obr. A13: Dialog pro vytvoření dalšího přístupového indexu

Poté se zobrazí dialog pro výběr typu klíče.

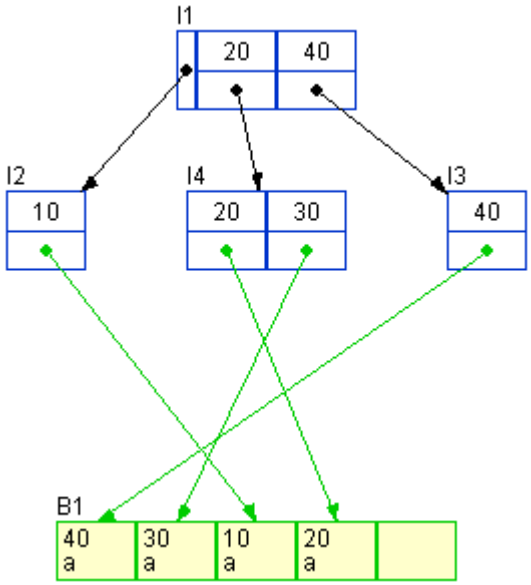
Obr. A14: Dialog pro výběr typu klíče

Při tvorbě dalšího přístupového indexu se může narazit na situaci, kdy jsou klíče v bázovém souboru duplicitní. V indexu ovšem ale tato situace nastat nemůže, proto se odkazy na duplicitní klíče v bázovém souboru jednoduše ignorují.

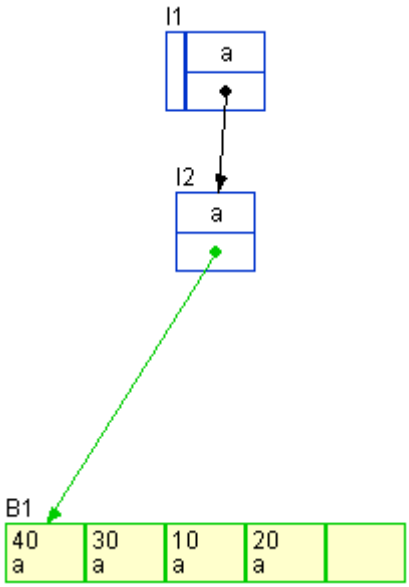
Na obrázku A15 je vytvořen přístupový index podle klíče typu Integer. Záznamy v bázovém souboru jsou:

- 10, a
- 20, a
- 30, a

Při tvorbě druhého přístupového indexu podle klíče typu String musí být klíče unikátní. Tato situace je znázorněna na obrázku A16, kde dojde k tomu, že v indexu je pouze jeden klíč *a*, který ukazuje na záznam „10, a“ v bázovém souboru. Ostatní záznamy v bázovém souboru, které obsahují stejný klíč, se ignorují.



Obr. A15: Soubor s úplným indexem s klíčem typu Integer



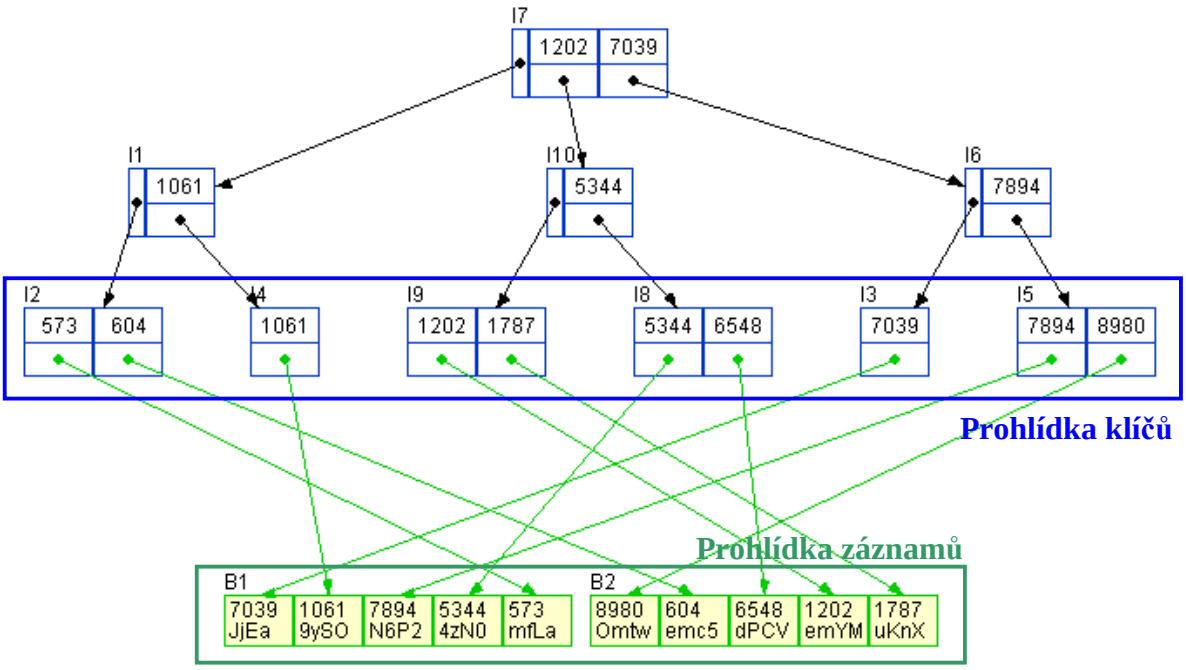
Obr. A16: Soubor s úplným indexem s klíčem typu String

Ukončení aplikace (Ctrl-X)

Tato operace ukončí aplikaci. Stávající Soubor s úplným indexem se uloží.

Menu Prohlídka

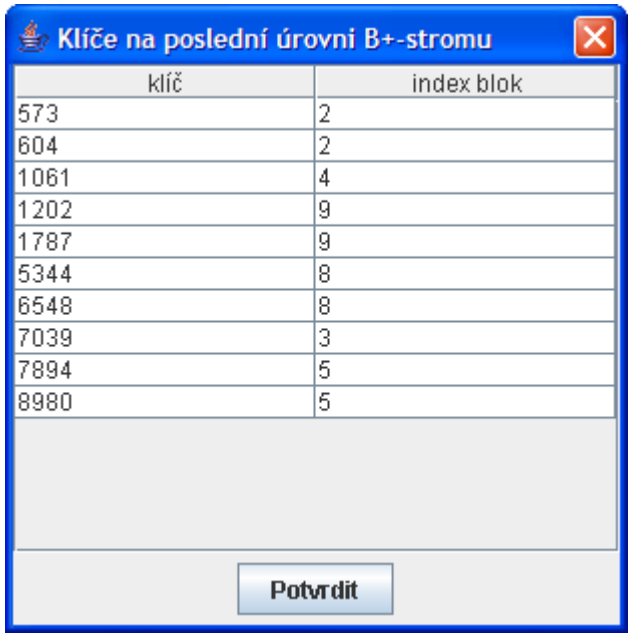
Na následujícím obrázku je výchozí Soubor s úplným indexem pro prohlídku. Na tomto příkladě se nyní prezentuje prohlídka klíčů a prohlídka záznamů.



Obr. A17: Výchozí Soubor s úplným indexem pro prohlídku

Prohlídka klíčů

Na následujícím obrázku je prohlídka klíčů na poslední logické úrovni B^+-stromu. Klíče jsou uspořádány vzestupně. U každého klíče je informace o tom, ve kterém bloku indexového souboru se klíč nachází.



Obr. A18: Příklad prohlídky klíčů

Prohlídka záznamů

Na obrázku 11 je prohlídka záznamů hlavního souboru. Každý záznam obsahuje následující informace:

- atribut typu Integer,
- atribut typu String,
- číslo bloku v bázovém souboru,
- umístění v bloku.

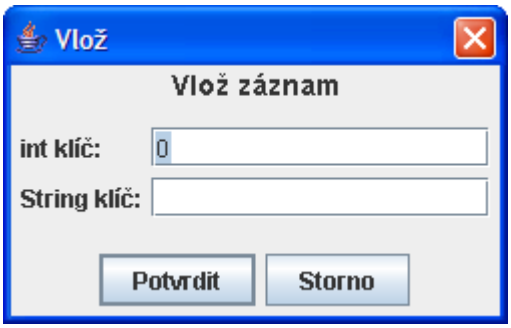
Obr. A19: Příklad prohlídky záznamů

Menu Záznam

Vložení nového záznamu (Insert)

Tato operace vloží nový záznam do Souboru s úplným indexem. Záznam obsahuje 2 atributy:

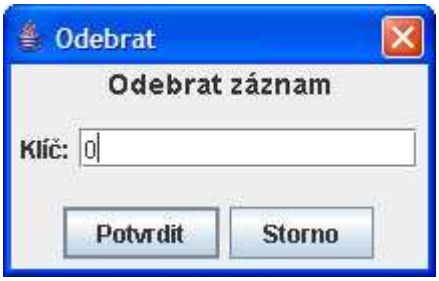
- atribut typu Integer,
- atribut typu String.



Obr. A20: Dialog pro vložení záznamu

Odebrání záznamu (Delete)

Tato operace odebere záznam ze Souboru s úplným indexem. Zadává se hodnota klíče, která se má odebrat ze Souboru s úplným indexem. Pokud je tedy klíč typu Integer, pak se zadává hodnota typu Integer. Pokud je klíč typu String, pak dialog očekává vstup hodnoty typu String.



Obr. A21: Dialog pro odebrání záznamu

Hledání záznamu (Ctrl-F)

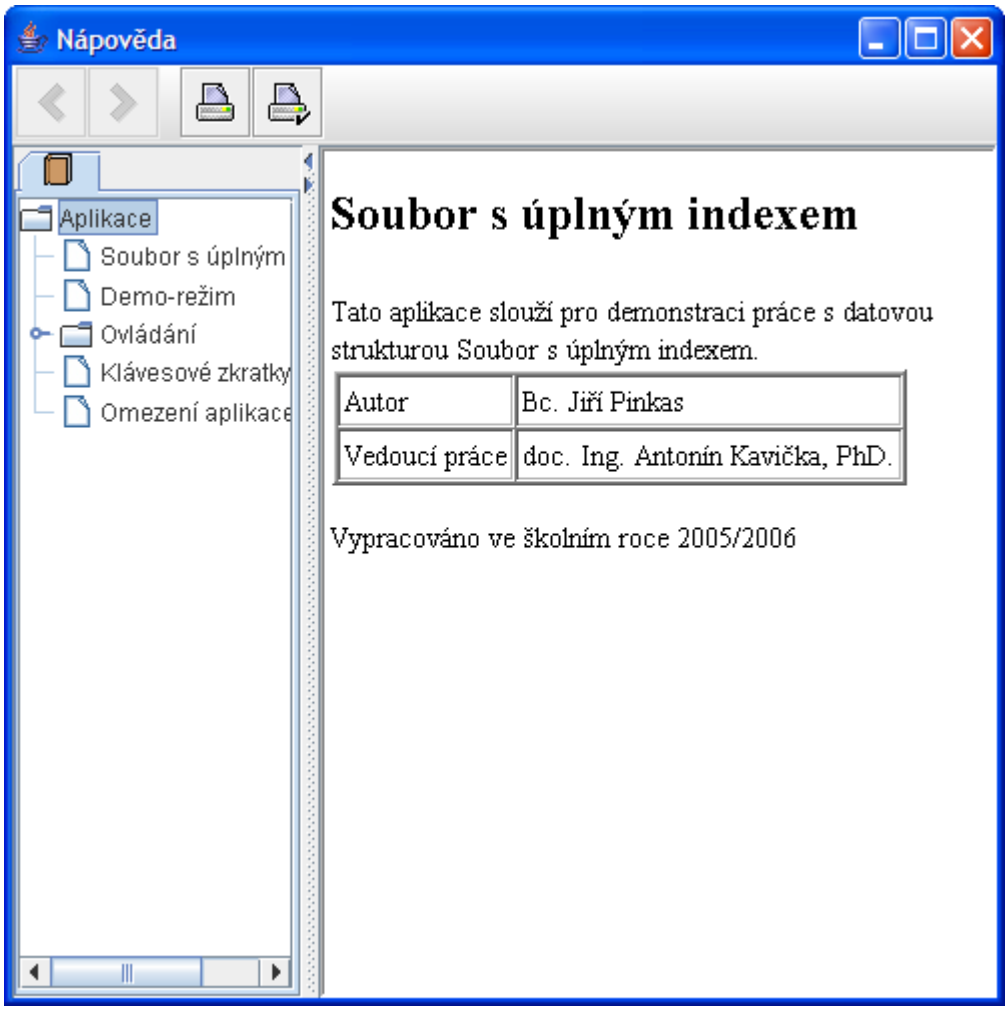
Tato operace vyhledá záznam v Souboru s úplným indexem. Stejně jako v dialogu „Odeber …“ se zadává pro nalezení hodnota zvoleného klíče.

Obr. A22: Dialog pro hledání záznamu

Menu Nápověda

Uživatelská nápověda

V nápovědě je v elektronické podobě uživatelská nápověda a základní informace o aplikaci a Souboru s úplným indexem.



Obr. A23: Uživatelská nápověda

Informace o programu

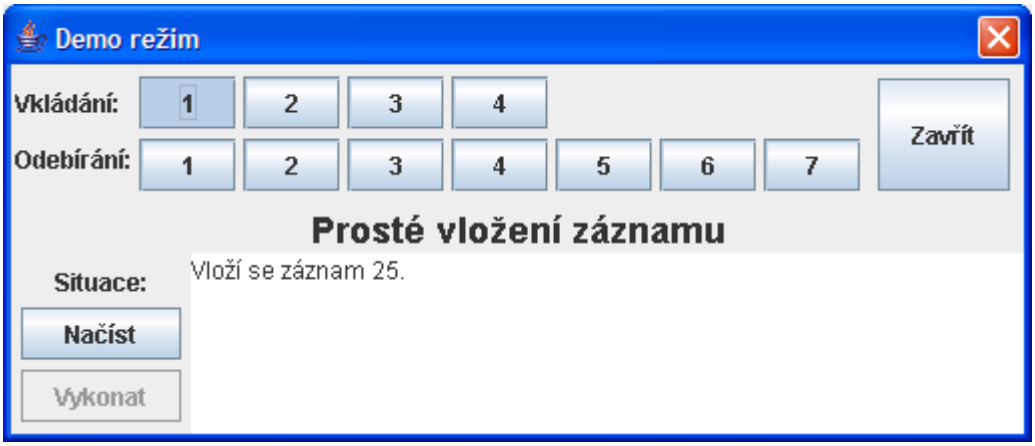
Posledním dialogem je krátký souhrn informací o aplikaci v dialogu „O programu“.

Obr. A24: Informace o programu

II. Demo režim

Aplikace obsahuje demo-režim, který se aktivuje kliknutím na tlačítko DEMO. V tomto režimu jsou připraveny různé situace, které mohou nastat při vkládání, resp. odebírání v Souboru s úplným indexem. Demo-režim se vypne kliknutím na tlačítko Zavřít.

Ovládání hlavní aplikace je v demo-režimu značně omezeno. V menu je přístupná pouze nápověda. V panelu nástrojů je k dispozici panel pro ovládání rychlosti animace.



Obr. A25: Ovládání demo-režimu

Pro vkládání do Souboru s úplným indexem jsou připraveny 4 situace:

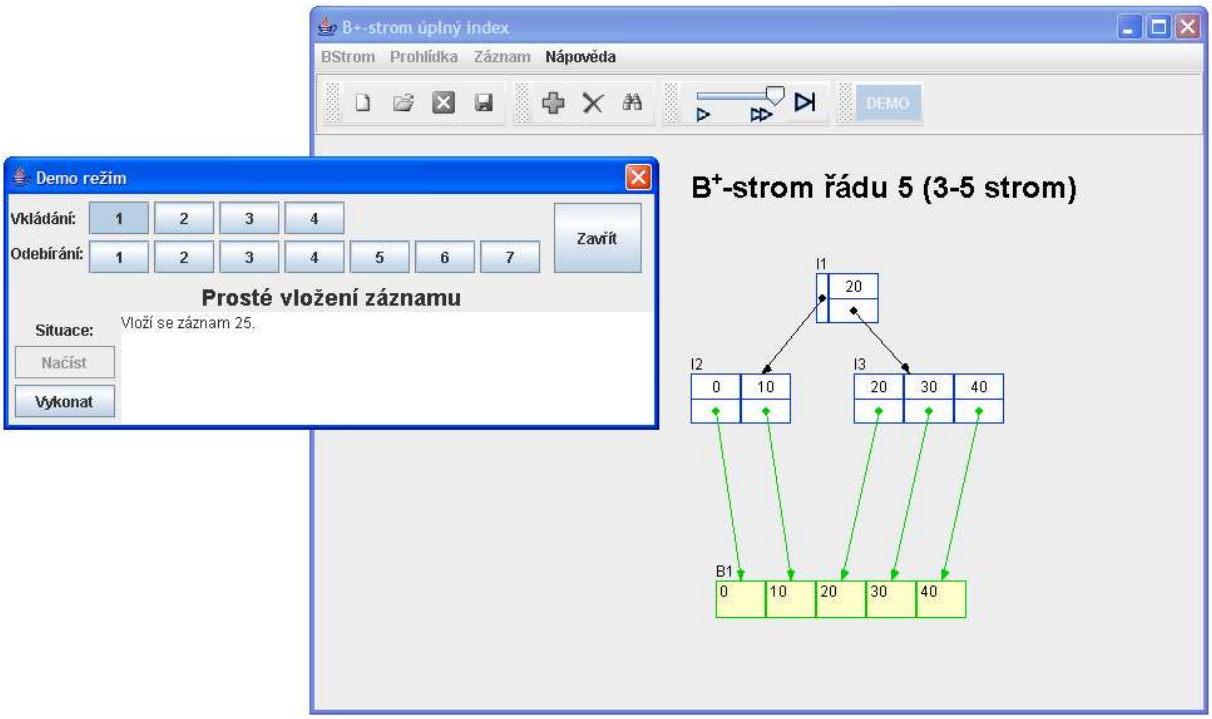
1. Vložení záznamu bez reorganizace struktury
2. Vložení záznamu s přetečením bloku a následnou alokací nového bloku na poslední logické úrovni
3. Vložení záznamu s přetečením bloku a následnou alokací nového bloku na poslední i vyšší úrovni
4. Vložení záznamu, které si vynutí alokaci nových bloků, která je ukončena vytvořením nového kořene B^+-stromu a s tím zvýšením výšky stromu

Pro odebírání ze Souboru s úplným indexem je připraveno 7 modelových situací:

1. Odebrání záznamu bez reorganizace struktury
2. Odebrání záznamu, které způsobí podtečení bloku a přesun záznamů z levého logicky sousedního bloku bez další reorganizace
3. Odebrání záznamu, které způsobí podtečení bloku a přesun záznamů z pravého logicky sousedního bloku bez další reorganizace
4. Odebrání záznamu, které způsobí podtečení bloku a následné sloučení 2 sousedních bloků
5. Odebrání záznamu, které způsobí podtečení bloku a následné sloučení 2 sousedních bloků, které způsobí další reorganizaci na vyšší úrovni (přesun záznamu z pravého logicky sousedního bloku)
6. Odebrání záznamu, které způsobí podtečení bloku a následné sloučení 2 sousedních bloků, které způsobí další reorganizaci na vyšší úrovni (přesun záznamu z levého logicky sousedního bloku)
7. Odebrání záznamu, které způsobí řetězové slučování bloků až ke kořenu, což vyústí v dealokaci kořene a snížení výšky stromu

Tlačítkem Načíst se načte příslušná modelová situace a tlačítkem Vykonat se poté spustí. Ke každé situaci je také krátký textový popis, který popisuje co se stane při spuštění příslušné modelové situace.

Na následujícím obrázku je příklad spuštěného demo-režimu. Jedná se o první modelovou situaci vkládání do Souboru s úplným indexem. B^+-strom je řádu 5 (3-5 strom).



Obr. A26: Příklad spuštěného demo-režimu

III. Klávesové zkratky

Pro rychlé ovládání programu je k dispozici sada klávesových zkratek pro nejčastější operace prováděné v aplikaci.

Příkaz	**Popis funkce příkazu**
Ctrl-G	Náhodné generování nového stromu
Ctrl-N	Vytvoření nového prázdného stromu
Ctrl-O	Načtení stromu ze souborů
Ctrl-W	Zrušení aktuálního stromu
Ctrl-S	Uložení kopie stromu
Ctrl-I	Vytvoření dalšího přístupového indexu
Ctrl-X	Ukončení aplikace
Insert	Vložení záznamu
Delete	Odebrání záznamu
Ctrl-F	Zpřístupnění záznamu

Tabulka A1 – Klávesové zkratky pro ovládání aplikace

IV. Omezení aplikace

Aplikace je zaměřena na názornou demonstraci Souboru s úplným indexem. Proto jsou stanovena určitá omezení, která zabezpečují přehlednost vytvořených stromů.

Omezení aplikace jsou následující:

- Maximální počet záznamů v B^+-stromu je 15 záznamů
- Rozmezí, ve kterém jsou klíče typu Integer: <–9999, 9999>
- Klíče typu String musí mít maximálně 4 znaky
- Maximální velikost bloku indexového souboru je: 2, 4, 6, 8, nebo 10 záznamů
- Maximální velikost bloku bázového souboru je: 1 až 10 záznamů

Příloha B: Obsah CD média

Cesta	**Popis**
\aplikace	Aplikace, kterou je možné spustit pomocí *spustit.bat*
\aplikace_web	Aplikace, kterou je možné umístit na web. Na webu musí běžet PHP server
\aplikace_ui	Aplikace Úplný index, která neobsahuje export animace do grafické reprezentace stromu
\analyza_web	Videa s animací appletů, které byly použity při analýze
\diagramy_trid	Plné diagramy tříd grafické reprezentace stromu
\text	Text diplomové práce ve formátu pdf