UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2011 Bc. Tomáš Michek

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Datové struktury pro uchovávání geografických dat

Bc. Tomáš Michek

Diplomová práce

2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Tomáš MICHEK**
Osobní číslo:	**I09374**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Datové struktury pro uchovávání geografických dat**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V úvodní části práce je nutné provést přehled problematiky datových struktur specializovaných na uchovávání geografických dat (k-D stromy, prioritní vyhledávací stromy, quad stromy, oktálové stromy, grid soubory apod.) a jejich kategorizaci. Pro zmíněné datové struktury je dále potřebné přehledově přiblížit problematiku bodového a intervalového vyhledávání. Primárním cílem diplomové práce je realizace efektivních vzorových implementací vybraných datových struktur výše uvedeného typu, provedení jejich srovnání a doporučení ohledně jejich nasazení dle typu řešené aplikace. Pro účely testování zkoumaných datových struktur se využijí reálná geografická data z vybrané části území České republiky.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

CORMEN, H. A KOL. Introduction to algorithms. Boston, MIT Press, 2001. LEWIS, H. R., DENENBERG, L. Data structures and their algorithms. Berkley, Adison-Wesley, 1997. GOODRICH, M.T., TAMASSIA, R. Algorithm Design. Hoboken (NJ), John Wiley & Sons, 2002. SAMET, H. Foundations of Multidimensional and Metric Data Structures, San Francisco (CA), Morgan Kaufmann Publishers, 2006.

Vedoucí diplomové práce: **doc. Ing. Antonín Kavička, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **27. října 2010**

Termín odevzdání diplomové práce: **20. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 3. listopadu 2010

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.

V Pardubicích dne 13. 5. 2011

Bc. Tomáš Michek

Poděkování

Rád bych poděkoval vedoucímu práce prof. Ing. Antonínu Kavičkovi, PhD. za zapůjčené materiály, cenné rady a čas, který mi během zpracování práce věnoval.

ANOTACE

Diplomová práce se zabývá problematikou vybraných datových struktur uchovávajících geografická data. V podpůrné aplikaci byly vytvořeny jejich vzorové implementace.

Základní operace pracující nad datovými strukturami byly otestovány a na základě jejich porovnání byla pro jednotlivé struktury doporučena vhodná oblast použití.

KLÍČOVÁ SLOVA

datové struktury, geografická data, 2D strom, range strom, prioritní vyhledávací strom, quad strom, grid soubor

TITTLE

Data structures for storage of geographic data

ABSTRACT

This thesis deals with selected data structures storing geographic data. Sample implementations of data structures are created in the support program.

Basic operations on data structures were tested. Based on their comparison, an area of application was suggested for each data structure.

KEYWORDS

data structures, geographic data, 2D tree, range tree, priority search tree, quad tree, grid file

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Cílem diplomové práce bylo provést průzkum v oblasti datových struktur pro uchovávání geografických dat. Datovou strukturu si lze představit jako nástroj, který slouží nejen k ukládání dat, ale také k jejich následnému vyhledávání. Výběrem vhodné datové struktury pro danou problematiku lze zajistit, že provádění základních operací bude dostatečně efektivní.

Tato práce popisuje datové struktury, jež vycházejí z abstraktního datového typu tabulka, ve kterém jsou uchovávané prvky opatřeny jednoznačnými klíčovými hodnotami. V případě geografických dat se jedná o multidimenzionální klíč, který tvoří hodnoty souřadnic pro zeměpisnou šířku a zeměpisnou délku. Porovnány byly následující datové struktury:

- 2D strom,
- prioritní vyhledávací strom,
- range strom,
- quad strom,
- grid soubor.

Čtyři prvně jmenované patří mezi hierarchické stromové struktury a využívají pouze operační paměť, naopak grid soubor vycházející z blokově orientovaného souboru pracuje i s externí pamětí v podobě pevného disku. Filozofie všech zmíněných struktur byla popsána na množině vzorových dat, které odpovídají městům České republiky a jejich souřadnicím.

Jako součást diplomové práce byla vytvořena také podpůrná aplikace, která obsahuje vzorové implementace jednotlivých struktur. Pomocí ní byly otestovány základní operace pracující nad těmito strukturami. Nad rámec samotného zadání byla navíc vytvořená aplikace použita i ke zpracování případové studie.

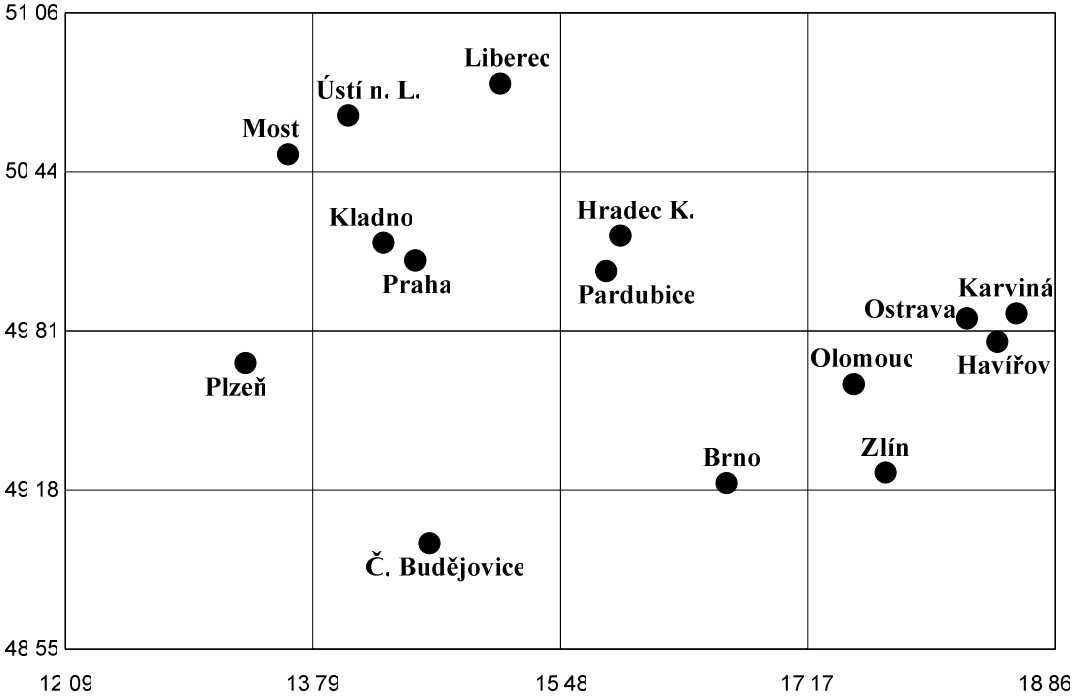
2. Vzorová data

Pro vysvětlení základních principů u implementovaných datových struktur byla zvolena reálná geografická data, která odpovídají patnácti největším městům České republiky a jejich souřadnicím [8].

Jejich seznam je uveden na obrázku 1, kromě zeměpisných souřadnic a názvů obsahuje také zkratky, které budou v dalších kapitolách používány. Geografické rozložení těchto měst zachycuje obrázek 2.

Praha (PHA) 50.09 14.42	**Brno (BRN)** 49.20 16.60	**Ostrava (OVA)** 49.83 18.28
Plzeň (PLZ) 49.75 13.38	**Olomouc (OLO)** 49.59 17.25	**Liberec (LIB)** 50.77 15.06
Hradec K. (HKR) 50.21 15.83	**Č. Budějovice (CEB)** 48.97 14.47	**Ústí n. L. (UNL)** 50.66 14.03
Pardubice (PCE) 50.04 15.78	**Havířov (HAV)** 49.80 18.44	**Zlín (ZLN)** 49.23 17.67
Kladno (KLA) 50.15 14.10	**Most (MST)** 50.50 13.64	**Karviná (KAR)** 49.85 18.54

Obrázek 1: Seznam vzorových dat, zdroj [autor]



Obrázek 2: Geografická poloha vzorových dat, zdroj [autor]

3. Seznámení s jednotlivými datovými strukturami

Tato kapitola má za cíl přiblížit čtenáři vybrané datové struktury. Pro snadnější pochopení byly základní operace k jednotlivým datovým strukturám zpracovány graficky v programu Microsoft Office Visio 2007, k čemuž byla využita vzorová data z předcházející kapitoly. Kromě toho byly jednotlivé operace popsány i pomocí pseudokódu, který v některých případech doplňují vysvětlující komentáře.

3.1. 2D strom

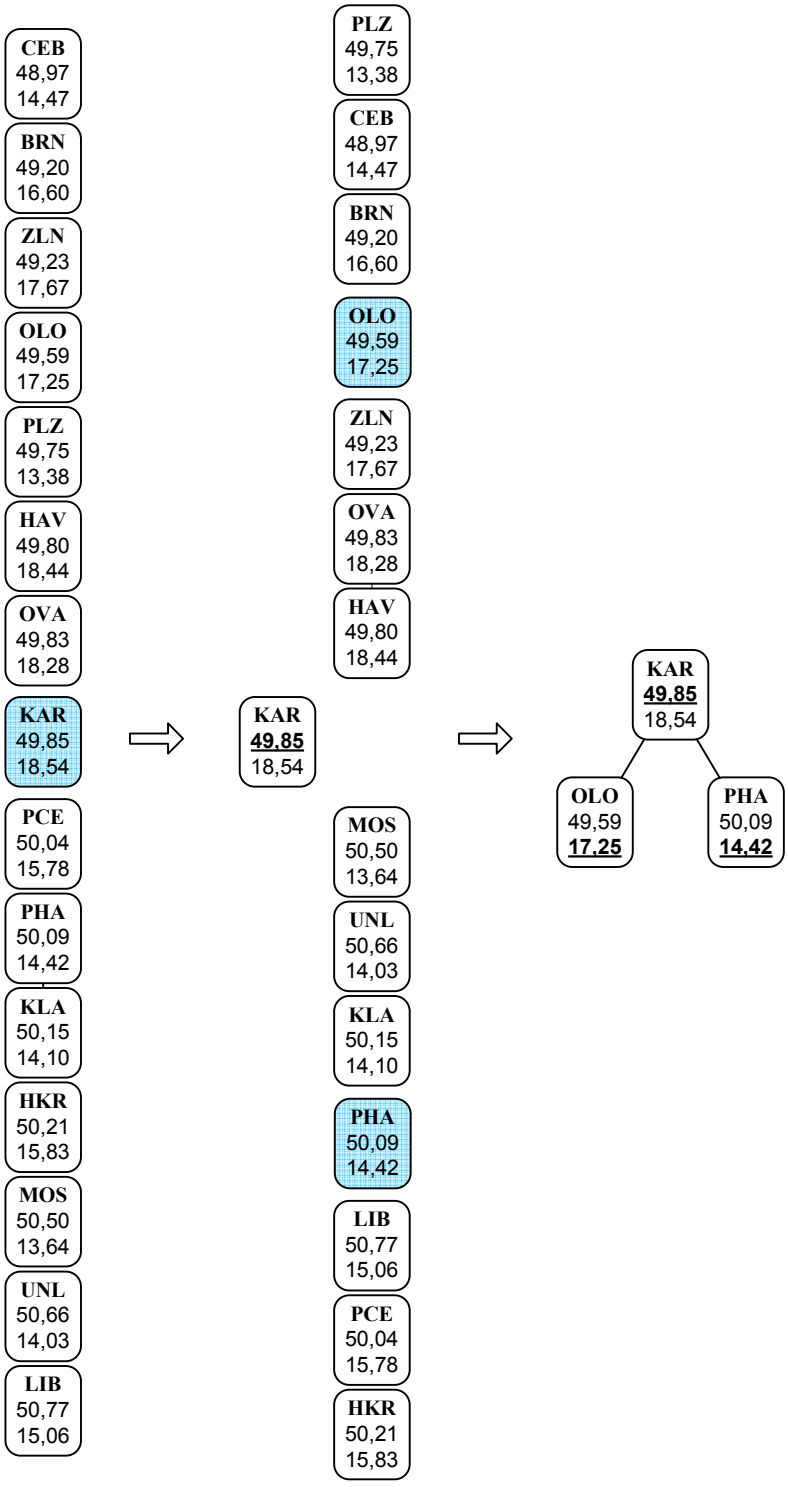
Zobecněním 2D stromu je datová struktura označovaná jako *k*D strom, která slouží k ukládání *k*-dimenzionálních dat. Vychází přitom z principů binárního vyhledávacího stromu, přičemž platí, že dělení *k*D prostoru se v každé úrovni realizuje pouze podle hodnoty jedné souřadnice. Konkrétně u 2D stromu, který pracuje s dvourozměrnými daty, to znamená, že se dělení na liché úrovni realizuje podle hodnot první souřadnice a na sudé úrovni podle hodnot druhé souřadnice. [1][4][5]

3.1.1. Vybudování struktury

V případě, že jsou vstupní data známa předem, lze vybudovat stromovou strukturu vyváženou. Pro vzorový příklad bylo rozhodnuto, že na liché úrovni bude 2D prostor dělen podle hodnoty zeměpisné šířky a na sudé úrovni podle hodnoty zeměpisné délky.

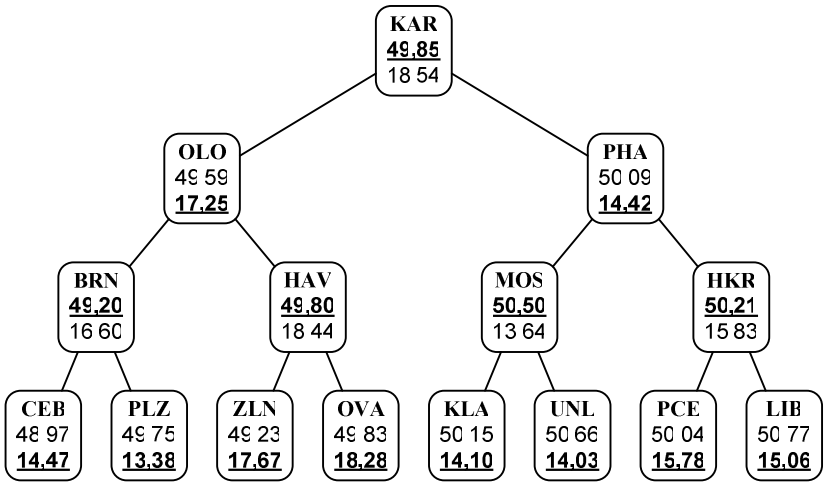
Nejprve se tedy vstupní data seřadí podle zeměpisné šířky a určí se medián, jemuž odpovídající prvek se stane kořenem stromu. Zbývající prvky jsou rozděleny na dvě části. Prvky s hodnotou zeměpisné šířky nižší než je hodnota zeměpisné šířky kořene tvoří jeho levý podstrom, zbývající prvky tvoří pravý podstrom. Obě části se seřadí podle zeměpisné délky a pro každou část se opět určí medián, jemu odpovídající prvek se následně stane potomkem kořene.

Graficky je tento postup zachycen na obrázku 3.



Obrázek 3: Princip budování 2D stromu, zdroj [autor]

Výše popsaný postup se rekurzivně opakuje pro levý i pravý podstrom až do okamžiku, kdy po rozdělení zbude v každé skupině prvků pouze jediný prvek, jenž se poté umístí na pozici listu stromu. Po dokončení odpovídá stav struktury situaci, která je znázorněna na obrázku 4.



Obrázek 4: Vybudovaný 2D strom, zdroj [autor]

Pseudokód

Vybuduj(↓prvky)

```
jestliže prvky = neexistuje || Pocet(prvky) = 0 pak
  Konec;
jinak
  pocet := Pocet(prvky);
  SeradPodleX(prvky);
  median := Pocet(prvky) / 2;
  koren := prvky[median];
  // vybudování podstromů kořene z rozdělených prvků
  Vybuduj(prvky[0 .. median - 1], koren, 1, true);
  Vybuduj(prvky[median + 1 .. Pocet(prvky) – 1], koren, 1, false);
```

Vybuduj(↓prvky, ↓predek, ↓uroven, ↓levyPodstrom)

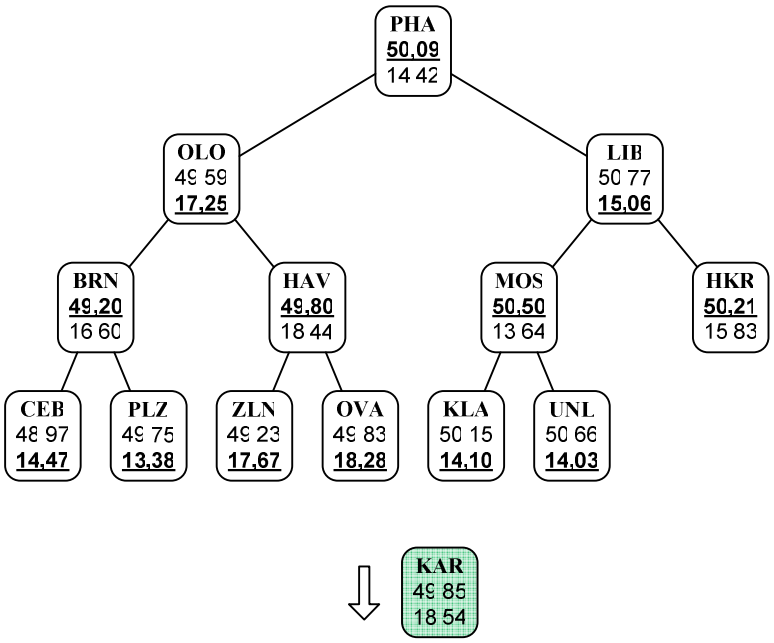
```
jestliže Pocet(prvky) = 0
  pak Konec;
jinak
  uroven := uroven + 1;
  median := Pocet(prvky) / 2;
  // seřazení prvků podle příslušné úrovně
  jestliže uroven = lichá pak
    SeradPodleX(prvky);
  jinak
    SeradPodleY(prvky);
```

```
// vybudování podstromů pro daného předka
jestliže levyPodstrom pak
  LevySyn(predek) :=  prvky[median];
  Vybuduj(prvky[0 .. median - 1], LevySyn(predek), uroven, true);
  Vybuduj(prvky[median + 1 .. Pocet(prvky) – 1], LevySyn(predek), uroven, false);
jinak
  PravySyn(predek) :=  prvky[median];
  Vybuduj(prvky[0 .. median - 1], PravySyn(predek), uroven, true);
  Vybuduj(prvky[median + 1 .. Pocet(prvky) – 1], PravySyn(predek), uroven, false);
```

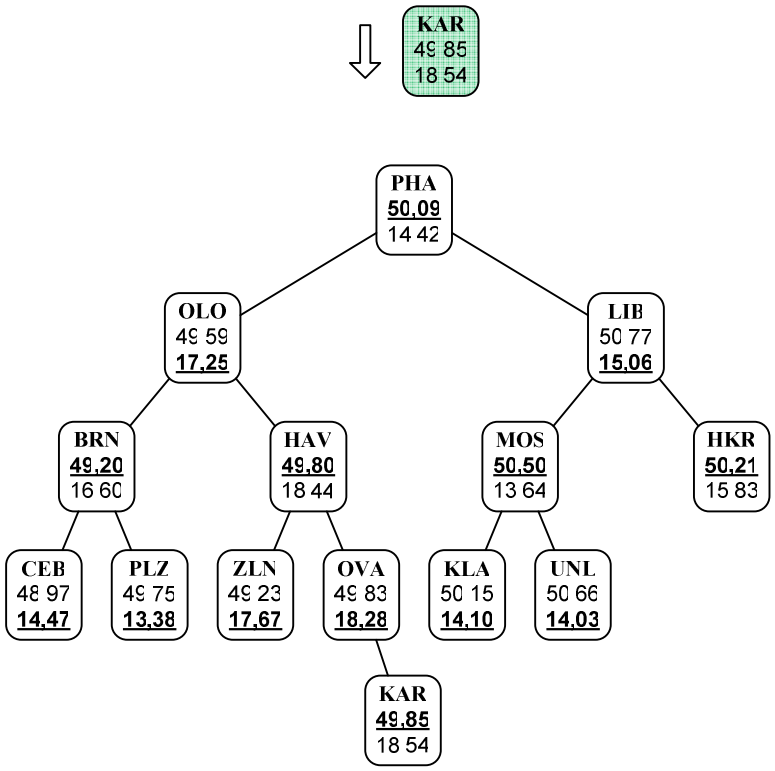
3.1.2. Vložení prvku

Vkládaný prvek traverzuje od kořene stromu směrem dolů na příslušnou pozici. Při průchodu stromem se v každém vrcholu rozhodne, kterým směrem má traverzování pokračovat. Toto rozhodnutí se realizuje porovnáním hodnot té souřadnice, která je pro danou úroveň diskriminátorem. V případě, že hodnota porovnávané souřadnice vkládaného prvku je nižší než hodnota odpovídající souřadnice aktuálního vrcholu, pokračuje traverzování do levého podstromu, v opačném případě pokračuje do pravého podstromu.

Obrázek 5 ukazuje stav před vložením prvku *Karviná*, výsledný stav 2D stromu po vložení zachycuje obrázek 6.



Obrázek 5: Vložení prvku do 2D stromu (stav před vložením), zdroj [autor]



Obrázek 6: Vložení prvku do 2D stromu (stav po vložení), zdroj [autor]

Pseudokód

Vloz(↓prvek)

```
pocet := pocet + 1;
jestliže koren = neexistuje pak
  koren := prvek;
jinak
  uroven := 1;
  pomocny := koren;
  while(true)
    // zvolení směru traverzování v případě liché úrovně
    // po průchodu stromem se nový prvek vloží na pozici listu
    jestliže uroven = lichá pak
      jestliže SouradniceX(pomocny) ≤ SouradniceX(prvek) pak
        jestliže PravySyn(pomocny) = neexistuje pak
          PravySyn(pomocny) := prvek;
          KonecCyklu;
        jinak pomocny := PravySyn(pomocny);
      jinak
        jestliže LevySyn(pomocny) = neexistuje pak
```

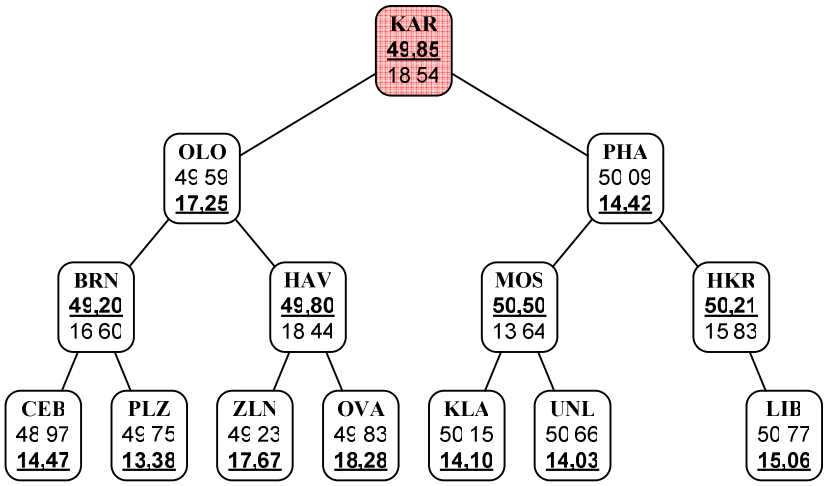
```
        LevySyn(pomocny) := prvek;
        KonecCyklu;
      jinak pomocny := LevySyn(pomocny);
  // zvolení směru traverzování v případě sudé úrovně
  // po průchodu stromem se nový prvek vloží na pozici listu
  jinak
    jestliže SouradniceY(pomocny) ≤ SouradniceY(prvek) pak
      jestliže PravySyn(pomocny) = neexistuje pak
        PravySyn(pomocny) := prvek;
        KonecCyklu;
      jinak pomocny := PravySyn(pomocny);
    jinak
      jestliže LevySyn(pomocny) = neexistuje
        pak LevySyn(pomocny) := prvek;
        KonecCyklu;
      jinak pomocny := LevySyn(pomocny);
  uroven := uroven + 1;
```

3.1.3. Odebrání prvku

Odstranění prvku se zadanými souřadnicemi je složitější než jeho vložení. Pro snazší vysvětlení lze operaci odebírání prvku z 2D stromu rozdělit podle pozice, na níž se odebíraný prvek ve stromové struktuře nachází, na tyto tři případy:

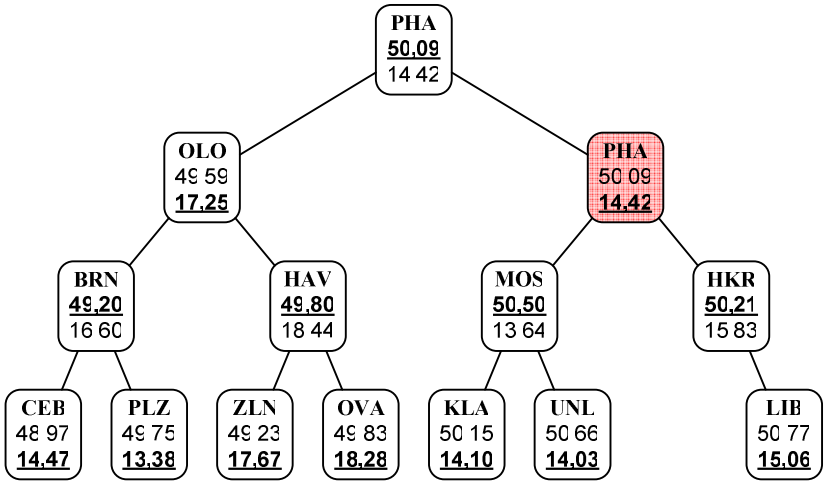
- je-li odebíraný prvek listem stromu, lze ho přímo odstranit,
- existuje-li pravý podstrom odebíraného prvku, nalezne se v něm pomocí operace *NajdiMin* zástupce, kterým se odebíraný prvek nahradí, následně se z původní pozice rekurzivně odebere tento zástupce,
- existuje-li pouze levý podstrom odebíraného prvku, nalezne se v něm pomocí operace *NajdiMax* zástupce, zbývající postup je totožný s předchozím bodem.

Obrázek 7 definuje stav stromové struktury, ze které se odebere prvek se souřadnicemi zeměpisné šířky = 49,85 a zeměpisné délky = 18,54 (*Karviná*).



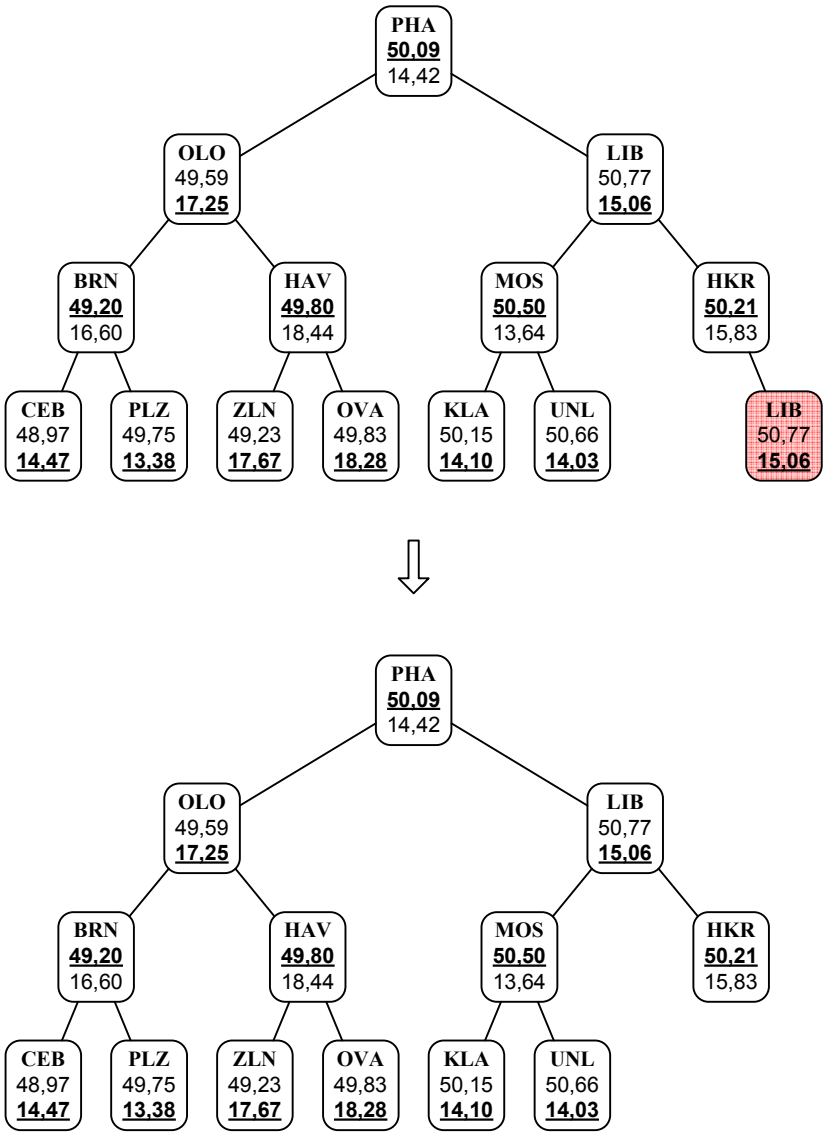
Obrázek 7: Odebrání prvku z 2D stromu (stav před odebráním), zdroj [autor]

Protože existuje pravý podstrom tohoto prvku, nalezne se v něm zástupce odebíraného prvku. Odebíraný prvek nachází v první (liché) úrovni stromové struktury, a proto hledá operace *NajdiMin* prvek s nejnižší hodnotou zeměpisné šířky. Zástupcem je zvolen prvek *Praha* (obrázek 8).



Obrázek 8: Odebrání prvku z 2D stromu (hledání zástupce), zdroj [autor]

Prvek *Praha* musí být ze své původní pozice rekurzivně odebrán. Protože leží ve druhé (sudé) úrovni, hledá operace *NajdiMin* v pravém podstromu zástupce podle nejnižší hodnoty zeměpisné délky. Nalezeným zástupcem pro prvek *Praha* je prvek *Liberec*, který leží na pozici listu, a tudíž bude z této pozice přímo odebrán. Tento krok společně s výslednou podobou struktury zachycuje obrázek 9.



Obrázek 9: Odebrání prvku z 2D stromu (stav po dokončení), zdroj [autor]

Pseudokód

Odeber(↓x, ↓y, ↑prvek)

```
pomocny := koren;
uroven := 1;
while(true)
  jestliže pomocny = neexistuje pak
    return neexistuje;
  // po dotraverzování k prvku s vyhovujícími souřadnicemi, dojde k odebrání
  jestliže SouradniceX(pomocny) = x & SouradniceY(pomocny) = y pak
    OdeberPrvek(pomocny, uroven);
    pocet--;
```

```
    KonecCyklu;
  // zvolení směru traverzování v případě liché úrovně
  jestliže uroven = lichá pak
    jestliže SouradniceX(pomocny) ≤ x pak
      pomocny := PravySyn(pomocny);
    jinak pomocny := LevySyn(pomocny);
  // zvolení směru traverzování v případě sudé úrovně
  jinak
    jestliže SouradniceY(pomocny) ≤ y pak
      pomocny := PravySyn(pomocny);
    jinak pomocny := LevySyn(pomocny);
      uroven := uroven + 1;
return pomocny;
```

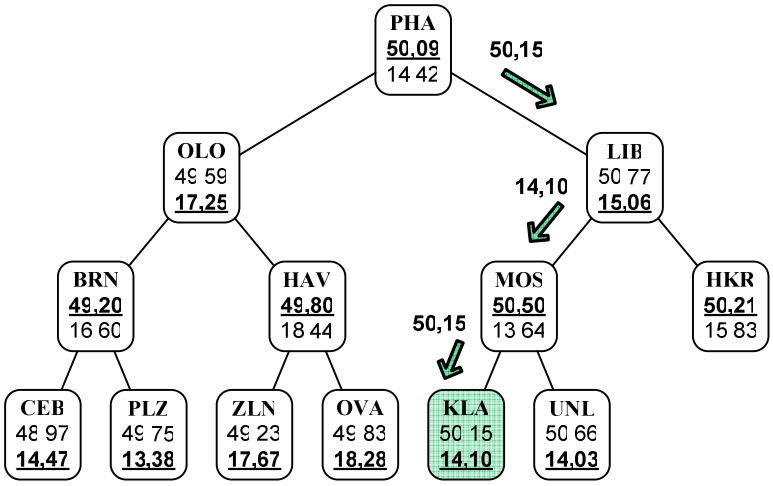
OdeberPrvek(↓prvek, ↓uroven)

```
// je-li odebirany prvek kořenem, dojde k jeho odstranění
jestliže PravySyn(prvek) = neexistuje & LevySyn(prvek) = neexistuje pak
  prvek := neexistuje;
// existuje-li pravý podstrom, je v něm nalezen zástupce
jinak jestliže PravySyn(prvek) ≠ neexistuje pak
  zastupce := neexistuje;
  urovenZast := 0;
  zastupce := NajdiMin(PravySyn(prvek), zastupce, uroven = lichá, uroven + 1,
                       urovenZast);
  // rekurzivní odebrání zástupce z původní pozice a nahrazení odebíraného prvku
  OdeberPrvek(zastupce, urovenZast);
  vrchol := zastupce;
// existuje-li pouze levý podstrom, je v něm nalezen zástupce
jinak
  zastupce := neexistuje;
  urovenZast := 0;
  zastupce := NajdiMax(LevySyn(prvek), zastupce, uroven = lichá, uroven + 1,
                       urovenZast);
  // rekurzivní odebrání zástupce z původní pozice a nahrazení odebíraného prvku
  OdeberPrvek(zastupce, urovenZast);
  vrchol := zastupce;
```

3.1.4. Hledání prvku

Operace hledání prvku ve 2D stromu podle zadaných souřadnic vychází ze stejného principu, který využívala také operace pro vložení prvku. Strukturou se traverzuje od kořene stromu směrem dolů, přičemž pro každý vrchol jsou porovnány jeho souřadnice se souřadnicemi hledanými. V případě rovnosti obou souřadnic je hledaným prvkem aktuální vrchol a algoritmus končí, v opačném případě se rozhodne o směru, kterým má hledání pokračovat. Pokud nebyl nalezen prvek vyhovující zadaným souřadnicím a současně bylo dotraverzováno k listu, lze jednoznačně říci, že hledaný prvek ve struktuře neexistuje.

Princip hledání prvku, kde zeměpisná šířka = 50,15 a zeměpisná délka = 14,10 (*Kladno*), je vysvětlen na obrázku 10.



Obrázek 10: Hledání prvku ve 2D stromu, zdroj [autor]

Pseudokód

```
Najdi(↓x, ↓y, ↑prvek)
  pomocny := koren;
  uroven := 1;
  while(true)
    jestliže pomocny = neexistuje pak
      return neexistuje;
    // porovnání hledaných souřadnic se souřadnicemi vrcholu
    jestliže SouradniceX(pomocny) = x & SouradniceY(pomocny) = y pak
      return pomocny;
    // zvolení směru traverzování v případě liché úrovně
    jestliže uroven = lichá pak
```

```
    jestliže SouradniceX(pomocny) ≤ x pak
      pomocny := PravySyn(pomocny);
    jinak pomocny := LevySyn(pomocny);
  // zvolení směru traverzování v případě sudé úrovně
  jinak
    jestliže SouradniceY(pomocny) ≤ y pak
      pomocny := PravySyn(pomocny);
      jinak pomocny := LevySyn(pomocny);
  uroven := uroven + 1;
```

3.1.5. Intervalové hledání

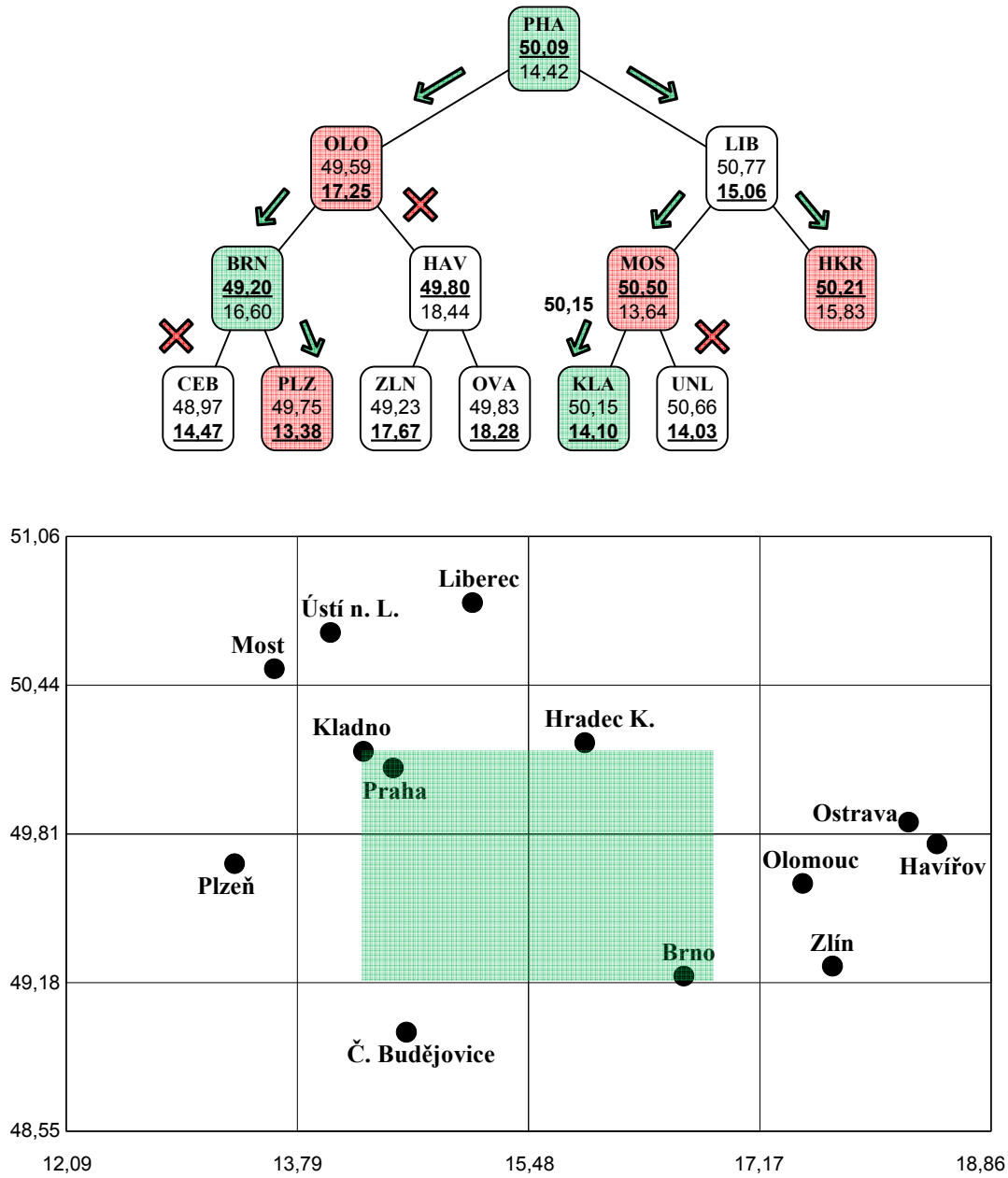
Hledání všech prvků z obdélníkového segmentu zadaného dvěma body lze pro 2D strom realizovat jednoduchým algoritmem. Stejně jako u hledání jednoho konkrétního prvku, vychází tento algoritmus z traverzování stromem od kořene směrem dolů. Pro každý prvek se testuje, zda do zadaného obdélníkového segmentu patří, či nikoliv. Pokud prvek v daném segmentu leží, aplikuje se na něj zvolená akce.

Navíc se v každém vrcholu rozhodne, zda má smysl pokračovat v prohledávání levého a pravého podstromu. Přijetí tohoto rozhodnutí vychází z podmínek:

- aktuální vrchol leží v liché úrovni:
    - je-li zeměpisná šířka vrcholu větší než minimální zeměpisná šířka v hledaném segmentu, pokračuje hledání do levého podstromu,
    - není-li zeměpisná šířka vrcholu větší než maximální zeměpisná šířka v hledaném segmentu, pokračuje hledání do pravého podstromu,
- aktuální vrchol leží v sudé úrovni:
    - je-li zeměpisná délka vrcholu větší než minimální zeměpisná délka v hledaném segmentu, pokračuje hledání do levého podstromu,
    - není-li zeměpisná délka vrcholu větší než maximální zeměpisná délka v hledaném segmentu, pokračuje hledání do pravého podstromu.

Princip intervalového hledání ukazuje obrázek 11, vyhledávány jsou všechny prvky z obdélníkového segmentu, kde:

- zeměpisná šířka = 49,20 až 50,15,
- zeměpisná délka = 14,10 až 17,00.



Obrázek 11: Intervalové hledání ve 2D stromu, zdroj [autor]

Pseudokód

Při prvním volání funkce je parametr *vrchol* roven kořenu 2D stromu a parametr *uroven* hodnotě *0*.

```
NajdiInterval(↓x1, ↓y1, ↓x2, ↓y2, ↓vrchol, ↓uroven, ↓akce)
  uroven := uroven + 1;
  jestliže vrchol ≠ neexistuje pak
    // test, zda vrchol leží v hledaném segmentu
    jestliže x1 ≤ SouradniceX(vrchol) & SouradniceX(vrchol) ≤ x2
     & y1 ≤ SouradniceY(vrchol) & SouradniceY(vrchol) ≤ y2 pak
        akce(vrchol);
    // rozhodnutí, zda má smysl pokračovat hledáním v levém podstromu
    jestliže (uroven = lichá & x1 < SouradniceX(vrchol))
     || (uroven = sudá & y1 < SouradniceY(vrchol)) pak
        NajdiInterval(x1, y1, x2, y2, LevySyn(vrchol), uroven, akce);
    // rozhodnutí, zda má smysl pokračovat hledáním v pravém podstromu
    jestliže (uroven = lichá & x2 ≥ SouradniceX(vrchol))
     || (uroven = sudá & y2 ≥ SouradniceY(vrchol)) pak
       NajdiInterval(x1, y1, x2, y2, PravySyn(vrchol), uroven, akce);
```

3.2. Range strom

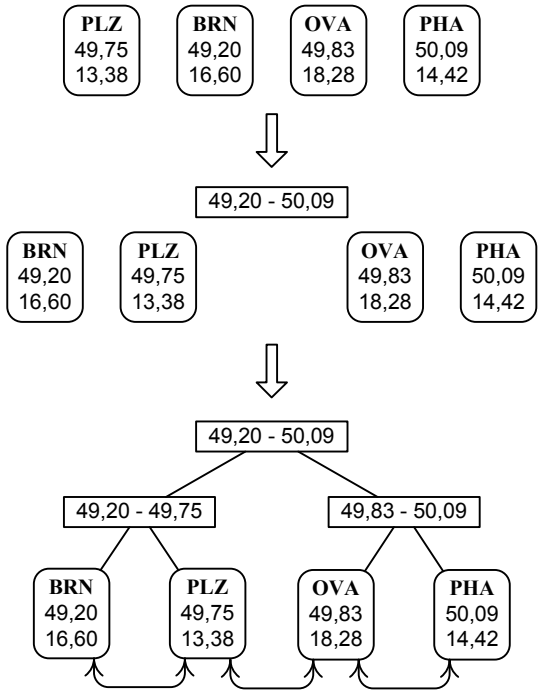
Datová struktura range strom se vyznačuje tím, že se plnohodnotné prvky nacházejí pouze na úrovni listů, kde jsou mezi sebou zřetězeny a zároveň seřazeny podle jedné z dimenzí. Ostatní vrcholy stromu tvoří navigační strukturu a mají uloženu informaci o intervalu, do kterého patří všichni jejich potomci. Celá struktura se skládá z jednoho hlavního stromu, jenž umožňuje hledání pouze podle první dimenze, a několika stromů druhé dimenze. Počet stromů druhé dimenze se rovná počtu navigačních vrcholů v hlavním stromu, z jeho navigačních vrcholů také lze do jednotlivých stromů druhé dimenze přeskočit. [2][7]

3.2.1. Vybudování struktury

Obdobně jako u 2D stromu lze i pro range strom vybudovat vyváženou strukturu, pokud jsou vstupní data k dispozici předem. Vzhledem k tomu, že se kromě hlavního stromu pro první dimenzi buduje také několik stromů pro druhou dimenzi, je metoda pro vybudování struktury složitější, než tomu bylo u 2D stromu. Pro vzorový příklad byla za první dimenzi zvolena zeměpisná šířka a za druhou dimenzi zeměpisná délka, princip budování stromové struktury je poté následující.

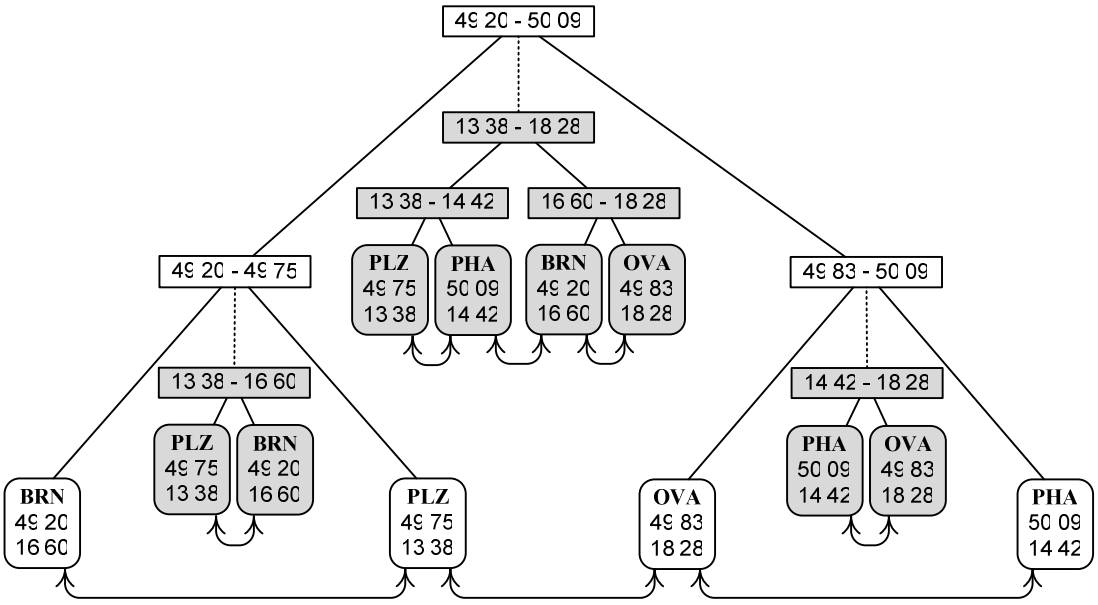
V případě, že je pro budování předáno více prvků než jeden, vznikne navigační vrchol hlavního stromu. Protože za první dimenzi byla zvolena zeměpisná šířka, uloží si tento vrchol nejmenší možný interval zeměpisné šířky, jenž pojme všechny prvky. Ty se následně seřadí podle první dimenze a rozdělí se na poloviny. První polovina bude tvořit levý podstrom navigačního vrcholu, druhá polovina pravý podstrom. Tento postup se cyklicky opakuje až do okamžiku, kdy zbude po rozdělení pouze jeden prvek. Ten se stane listem range stromu, kde se společně s dalšími listy (plnohodnotnými prvky) zřetězí. Každý navigační vrchol ve stromu první dimenze navíc obsahuje ukazatel na strom druhé dimenze, jehož vybudování se realizuje obdobným způsobem podle zeměpisné délky vždy po vytvoření nového navigačního vrcholu v hlavním stromu.

Graficky přibližuje princip této operace obrázek 12, na kterém se pro přehlednost realizuje pouze budování hlavního stromu pro první dimenzi. Oproti 2D stromu je výsledná struktura rozsáhlejší, a proto se pro tento ukázkový příklad používá menší počet prvků.



Obrázek 12: Princip budování range stromu, zdroj [autor]

Po připojení stromů druhé dimenze ke všem navigačním vrcholům hlavního stromu vypadá výsledná struktura následovně (obrázek 13).



Obrázek 13: Vybudovaný range strom, zdroj [autor]

Pseudokód

Vybuduj(↓prvky)

```
jestliže prvky = neexistuje || Pocet(prvky) = 0
  pak Konec;
jinak
  pocet := Pocet(prvky);
  // vybudování stromu první dimenze, jeho vrchol se stane kořenem
  koren := VybudujStrom(prvky, neexistuje, true);
```

VybudujStrom(↓prvky, ↓prvekDruhaDimenze, ↓dimenzeX, ↑prvek)

```
pomocny := VybudujPodstrom(prvky, neexistuje, neexistuje, dimenzeX);
// případné spojení vrcholu podstromu s prvkem opačné dimenze
jestliže prvekDruhaDimenze ≠ neexistuje pak
  DruhaDimenze(pomocny) := prvekDruhaDimenze;
return pomocny;
```

VybudujPodstrom(↓prvky, ↓predek, ↓predchozi, ↓dimenzeX, ↑prvek)

```
pomocny := VytvorNovyPrvek();
jestliže Pocet(prvky) ≥ 2 pak
  // nastavení intervalu pro navigační prvek
  jestliže dimenzeX = true pak
    SeradPodleX(prvky);
```

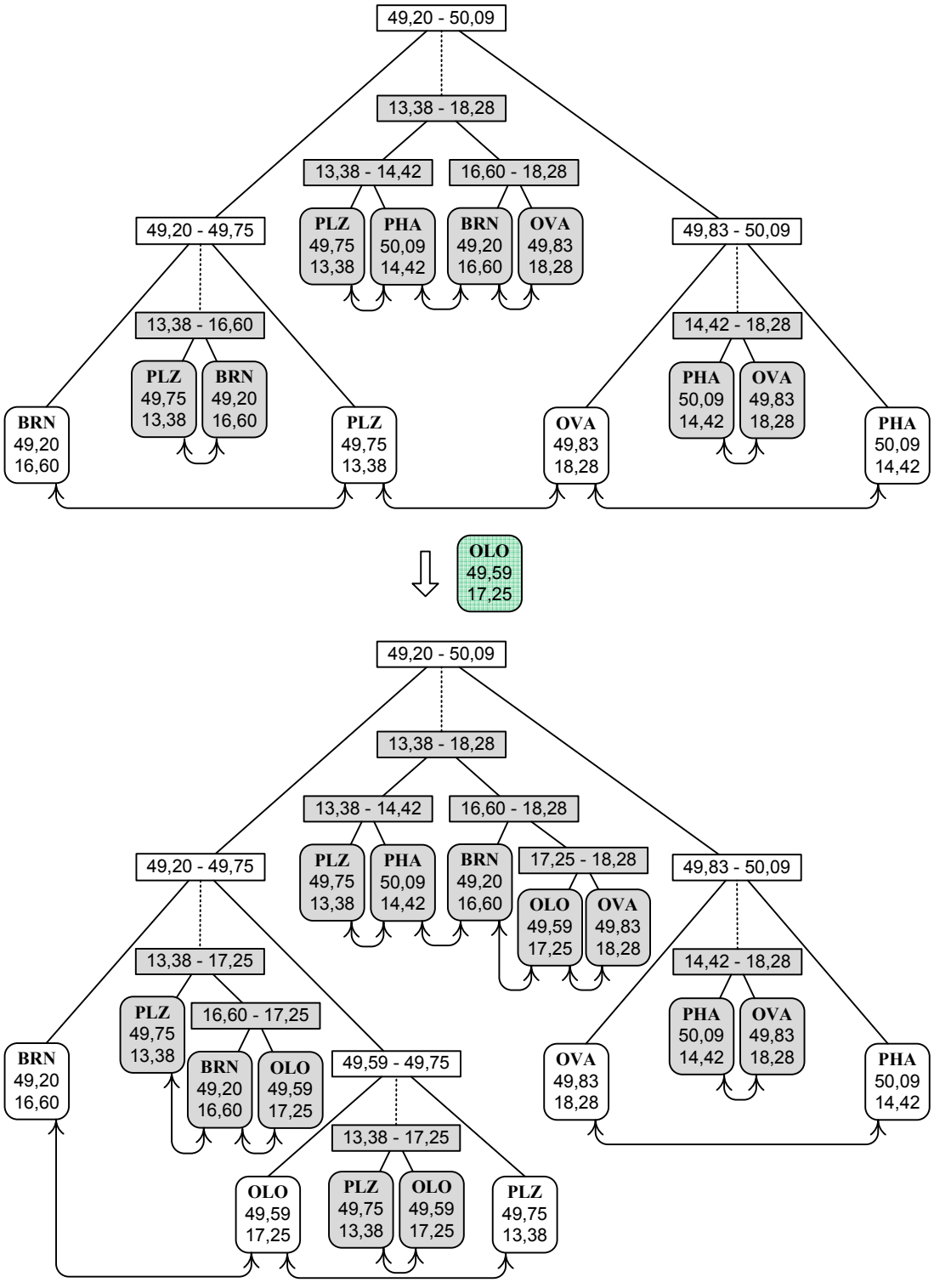
```
      SouradniceX(pomocny) := SouradniceX(prvky[0]);
      SouradniceY(pomocny) := SouradniceX(prvky[Pocet(prvky) – 1]);
    jinak
      SeradPodleY(prvky);
      SouradniceX(pomocny) := SouradniceY(prvky[0]);
      SouradniceY(pomocny) := SouradniceY(prvky[Pocet(prvky) – 1]);
    // vytvoření seznamů prvků pro levý a pravý podstrom, do nichž se prvky rozdělí
    prvkyL := VytvorSeznamPrvku();
    prvkyP := VytvorSeznamPrvku();
    RozdelPrvky(prvky, dimenzeX, prvkyL, prvkyP);
    // vybudování podstromů z rozdělených prvků
    LevySyn(pomocny) := VybudujPodstrom(prvkyL, pomocny, predchozi, dimenzeX);
    PravySyn(pomocny) := VybudujPodstrom(prvkyP, pomocny, predchozi, dimenzeX);

  jinak
    // byl-li pro budování předán jediný prvek, stane se plnohodnotným prvkem
    pomocny := prvky[0];
    Platny(pomocny) := true;
    // zřetězení prvků na úrovni listů (plnohodnotné prvky)
    jestliže pomocny ≠ neexistuje pak
      Dalsi(predchozi) := pomocny;
      Predchozi(pomocny) := predchozi;
  // pro navig. vrchol ve stromu první dimenze je vybudován strom druhé dimenze
  jestliže dimenzeX = true & Platny(pomocny) = false pak
    DruhaDimenze(pomocny) := VybudujStrom(prvky, pomocny, false);
  Otec(pomocny) := predek;
  return pomocny;
```

3.2.2. Vložení prvku

Obdobně jako u 2D stromu spočívá princip vkládání prvku v traverzování stromovou strukturou směrem od kořene k listům. Při průchodu stromem se v každém navigačním vrcholu kontroluje jeho interval, a pokud leží vkládaný prvek mimo tento interval, dojde k jeho rozšíření. Pokud jsou synové také navigačními vrcholy, rozhodne se podle jejich intervalů, zda má traverzování pokračovat do levého či pravého podstromu. Při dosažení plnohodnotného prvku (listu) traverzování skončí a dosažený prvek se nahradí novým navigačním vrcholem, kterému se přidělí dva synové - vkládaný prvek a prvek, k němuž

bylo dotraverzováno. Oba synové leží na úrovni listů a jsou plnohodnotnými prvky, tudíž se zařadí do zřetězeného seznamu. Pro každý navigační vrchol hlavního stromu, který byl při traverzování navštíven, se vkládaný prvek vloží i do příslušného stromu druhé dimenze.

Tento postup znázorňuje obrázek 14, kde se do původní struktury vkládá prvek *Olomouc*.



Obrázek 14: Vložení prvku do range stromu, zdroj [autor]

Pseudokód

Při prvním volání je parametr *vrchol* roven kořenu stromu a *dimenzeX* hodnotě *true*.

```
Vloz(↓prvek, ↓vrchol, ↓dimenzeX)
  jestliže vrchol = neexistuje pak
    vrchol := prvek;
    pocet := pocet + 1;
  jinak
    // pro navigační vrchol se zkontroluje interval a zvolí se směr traverzování
    jestliže Platny(vrchol) = false pak
      KontrolaSirkyIntervalu(prvek, vrchol, dimenzeX);
      jestliže dimenzeX = true pak
        Vloz(prvek, DruhaDimenze(vrchol), false);
        jestliže Platny(PravySyn(vrchol)) = false
        & SouradniceX(PravySyn(vrchol)) ≤ SouradniceX(prvek) pak
          Vloz(prvek, PravySyn(vrchol), dimenzeX);
        jinak
          Vloz(prvek, LevySyn(vrchol), dimenzeX);
      jinak
        jestliže Platny(PravySyn(vrchol)) = false
        & SouradniceX(PravySyn(vrchol)) ≤ SouradniceY(prvek) pak
          Vloz(prvek, PravySyn(vrchol), dimenzeX);
        jinak
          Vloz(prvek, LevySyn(vrchol), dimenzeX);
    // po dosažení listu je vytvořen navigační vrchol
    // z vkládaného prvku a původního listu se vybuduje jeho podstrom
    jinak
      pomocny := Otec(vrchol);
      prvky := VytvorSeznamPrvku();
      prvky.Vloz(prvek);
      prvky.Vloz(vrchol);
      jestliže pomocny = neexistuje pak
        koren := VybudujPodstrom(prvky, pomocny, neexistuje, dimenzeX);
      jinak jestliže vrchol = LevySyn(pomocny) pak
        LevySyn(pomocny) := VybudujPodstrom(prvky, pomocny, Predchozi(vrchol),
                                            dimenzeX);
```

jinak

PravySyn(pomocny) := VybudujPodstrom(prvky, pomocny, Predchozi(vrchol), dimenzeX);

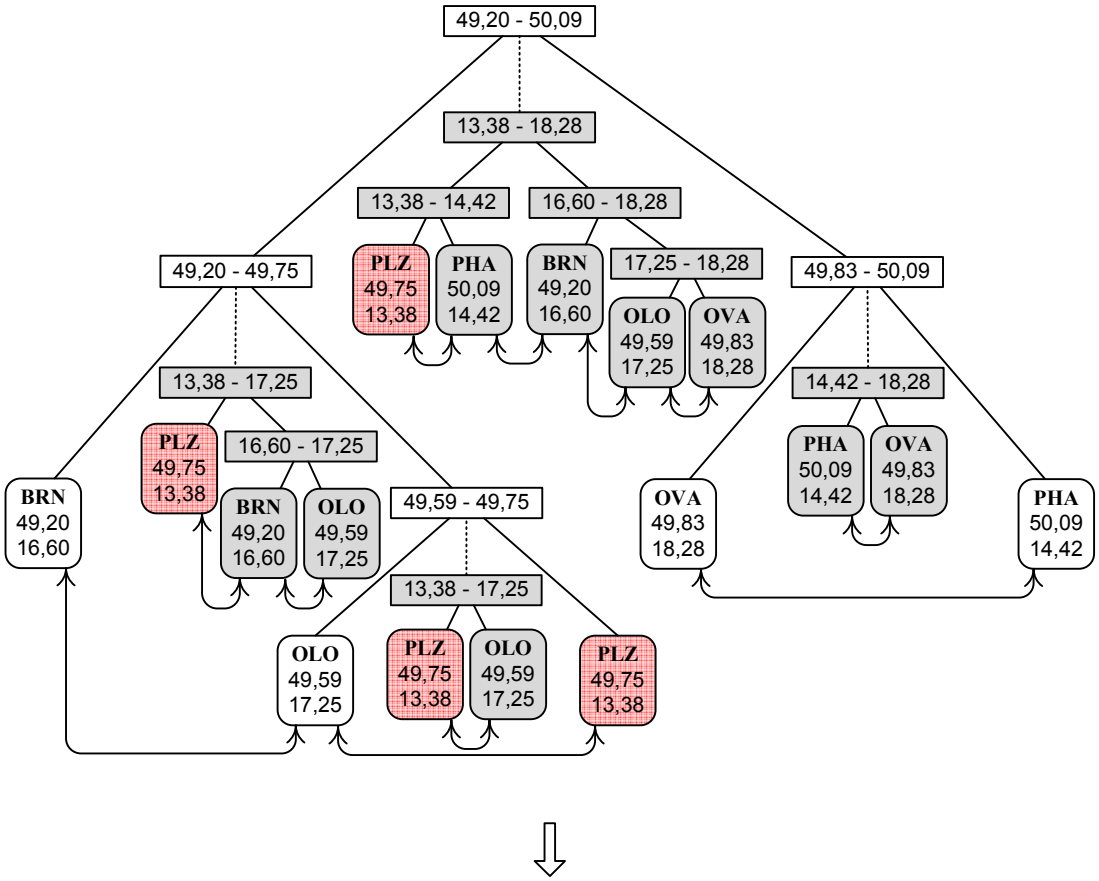
jestliže dimenzeX = true pak

pocet := pocet + 1;

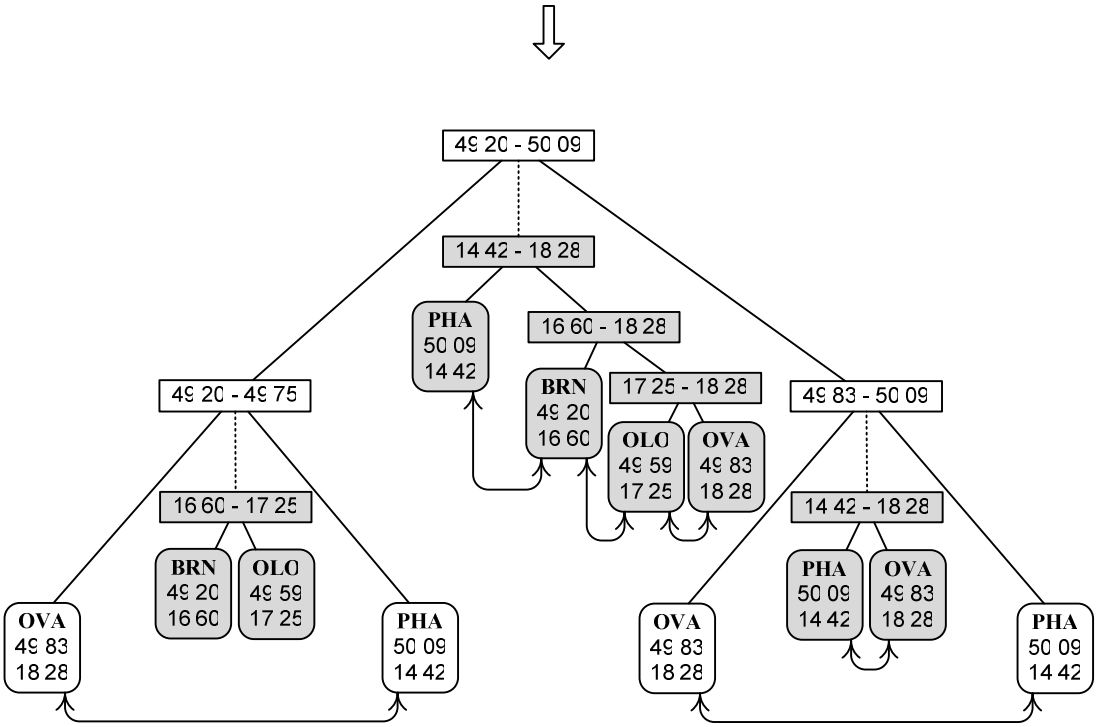
3.2.3. Odebrání prvku

Princip odebrání prvku se zadanými souřadnicemi je poměrně jednoduchý. Odebíraný prvek se odstraní ze stromu první dimenze a také ze všech stromů druhé dimenze, ve kterých se vyskytuje. Následně se bratr odebíraného prvku přemístí na pozici svého otce. Ve specifickém případě, kdy struktura obsahuje pouze jediný (odebíraný) prvek, jenž se nachází na pozici kořene stromu, dojde k odstranění prvku tím, že se do kořene stromu přiřadí hodnota *neexistuje*.

Stav stromu před odebráním prvku se souřadnicemi zeměpisné šířky = 49,75 a zeměpisné délky = 13,38 (*Plzeň*) ukazuje obrázek 15, výslednou situaci po odebrání obrázek 16.



Obrázek 15: Odebrání prvku z range stromu (stav před odebráním), zdroj [autor]



Obrázek 16: Odebrání prvku z range stromu (stav po odebrání), zdroj [autor]

Pseudokód

Odebíraný prvek je nejprve nalezen a předán pomocí atributu *vrchol*. Parametr *dimenzeX* se rovná hodnotě *true*, jestliže byl prvek nalezen ve stromu první dimenze, v opačném případě se tento parametr rovná hodnotě *false*.

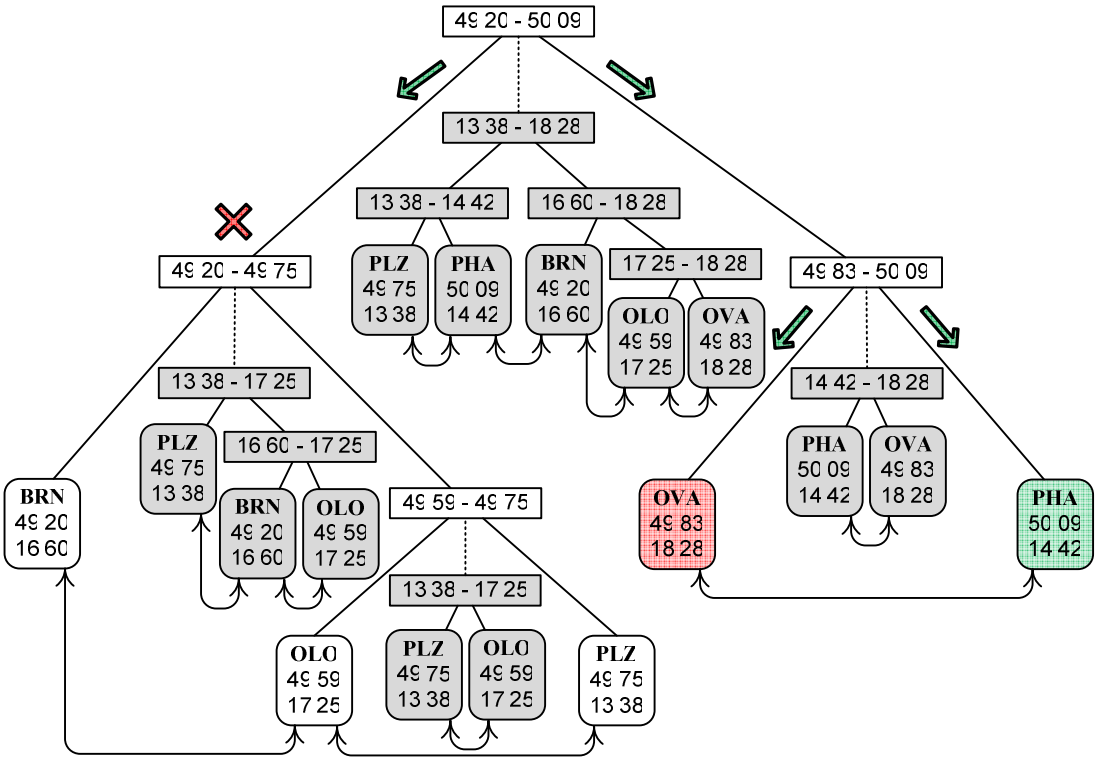
```
OdeberPrvek(↓vrchol, ↓dimenzeX)
 // je-li vrchol kořenem stromu, odebere se
  jestliže Otec(vrchol) = neexistuje pak
    jestliže dimenzeX = true pak
      koren := neexistuje;
    jinak
      DruhaDimenze(DruhaDimenze(vrchol)) := neexistuje;
  // je-li otec kořenem stromu, stane se kořenem stromu bratr odebíraného prvku
  jinak jestliže Otec(Otec(vrchol) = neexistuje pak
    jestliže vrchol = LevySyn(Otec(vrchol)) pak
      jestliže dimenzeX = true pak
        koren := PravySyn(Otec(vrchol));
      jinak
        DruhaDimenze(DruhaDimenze(Otec(vrchol))) := PravySyn(Otec(vrchol));
```

```
        DruhaDimenze(PravySyn(Otec(vrchol))) := DruhaDimenze(Otec(vrchol));
    jinak
      jestliže dimenzeX = true pak
        koren := LevySyn(Otec(vrchol));
      jinak
        DruhaDimenze(DruhaDimenze(Otec(vrchol))) := LevySyn(Otec(vrchol));
        DruhaDimenze(LevySyn(Otec(vrchol))) := DruhaDimenze(Otec(vrchol));
  // pokud vrchol ani jeho otec nejsou kořenem stromu, přesune se bratr
  // odebíraného prvku na pozici otce a případně se zmenší intervaly navig. prvků
  jinak
    jestliže Otec(vrchol) = LevySyn(Otec(Otec(vrchol) pak
      jestliže vrchol = LevySyn(Otec(vrchol)) pak
        LevySyn(Otec(Otec(vrchol))) := PravySyn(Otec(vrchol));
        Otec(PravySyn(Otec(vrchol))) := Otec(Otec(vrchol));
        ZmensiInterval(PravySyn(Otec(vrchol)), dimenzeX);
      jinak
        LevySyn(Otec(Otec(vrchol))) := LevySyn(Otec(vrchol));
        Otec(LevySyn(Otec(vrchol))) := Otec(Otec(vrchol));
    jinak
      jestliže vrchol = LevySyn(Otec(vrchol)) pak
        PravySyn(Otec(Otec(vrchol))) := PravySyn(Otec(vrchol));
        Otec(PravySyn(Otec(vrchol))) := Otec(Otec(vrchol));
      jinak
        PravySyn(Otec(Otec(vrchol))) := LevySyn(Otec(vrchol));
        Otec(LevySyn(Otec(vrchol))) := Otec(Otec(vrchol));
        ZmensiInterval(LevySyn(Otec(vrchol)), dimenzeX);
```

3.2.4. Hledání prvku

Hledání prvku se zadanými souřadnicemi probíhá pouze ve stromu první dimenze, který je organizován podle zeměpisné šířky. Prohledávání začíná od kořene stromu směrem k listům. V každém navigačním vrcholu se testuje, zda interval vrcholu obsahuje hledanou hodnotu zeměpisné šířky. V případě záporné odpovědi hledání skončí, protože lze jednoznačně určit, že hledaný prvek se v range stromu nenachází. V opačném případě se pokračuje do obou podstromů aktuálního vrcholu. Při dosažení plnohodnotného prvku (listu stromu) se porovnají jeho souřadnice se souřadnicemi hledanými.

Princip hledání prvku se zeměpisnou šířkou = 50,09 a zeměpisnou délkou = 14,42 je znázorněn na obrázku 17.



Obrázek 17: Hledání prvku v range stromu, zdroj [autor]

Pseudokód

Najdi(↓x, ↓y, ↑prvek)

```
// je-li kořen platným vrcholem, porovnají se souřadnice a hledání skončí
jestliže koren ≠ neexistuje & Platny(koren) = true
  pak jestliže SouradniceX(koren) = x & SouradniceY(koren) = y pak
    return koren;
  jinak
    return neexistuje;
// v opačném případě je strom prohledáván traverzováním směrem k listům
pomocny := koren;
while(true)
  jestliže pomocny = neexistuje pak
    return neexistuje;
  // je-li levý syn platným vrcholem, porovnají se jeho souřadnice
  jestliže LevySyn(pomocny) ≠ neexistuje & Platny(LevySyn(pomocny)) = true
  & SouradniceX(LevySyn(pomocny)) = x & SouradniceY(LevySyn(pomocny)) = y pak
```

```
    return LevyPotomek(pomocny);
  // je-li pravý syn platným vrcholem, porovnají se jeho souřadnice
  jestliže PravySyn(pomocny) ≠ neexistuje & Platny(PravySyn(pomocny)) = true
  & SouradniceX(PravySyn(pomocny))=x & SouradniceY(PravySyn(pomocny))=y pak
    return PravySyn(pomocny);
  // jinak se rozhodne, kterým směrem pokračovat
  jestliže LevySyn(pomocny) ≠ neexistuje
  & Platny(LevySyn(pomocny)) = false& SouradniceX(LevySyn(pomocny)) ≤ x
  & x ≤ SouradniceY(LevySyn(pomocny)) pak
    pomocny := LevyPotomek(pomocny);
  jinak jestliže PravySyn(pomocny) ≠ neexistuje
  & Platny(PravySyn(pomocny)) = false & SouradniceX(PravySyn(pomocny)) ≤ x
  & x ≤ SouradniceY(PravySyn(pomocny)) pak
    pomocny := PravyPotomek(pomocny);
  jinak return neexistuje;
```

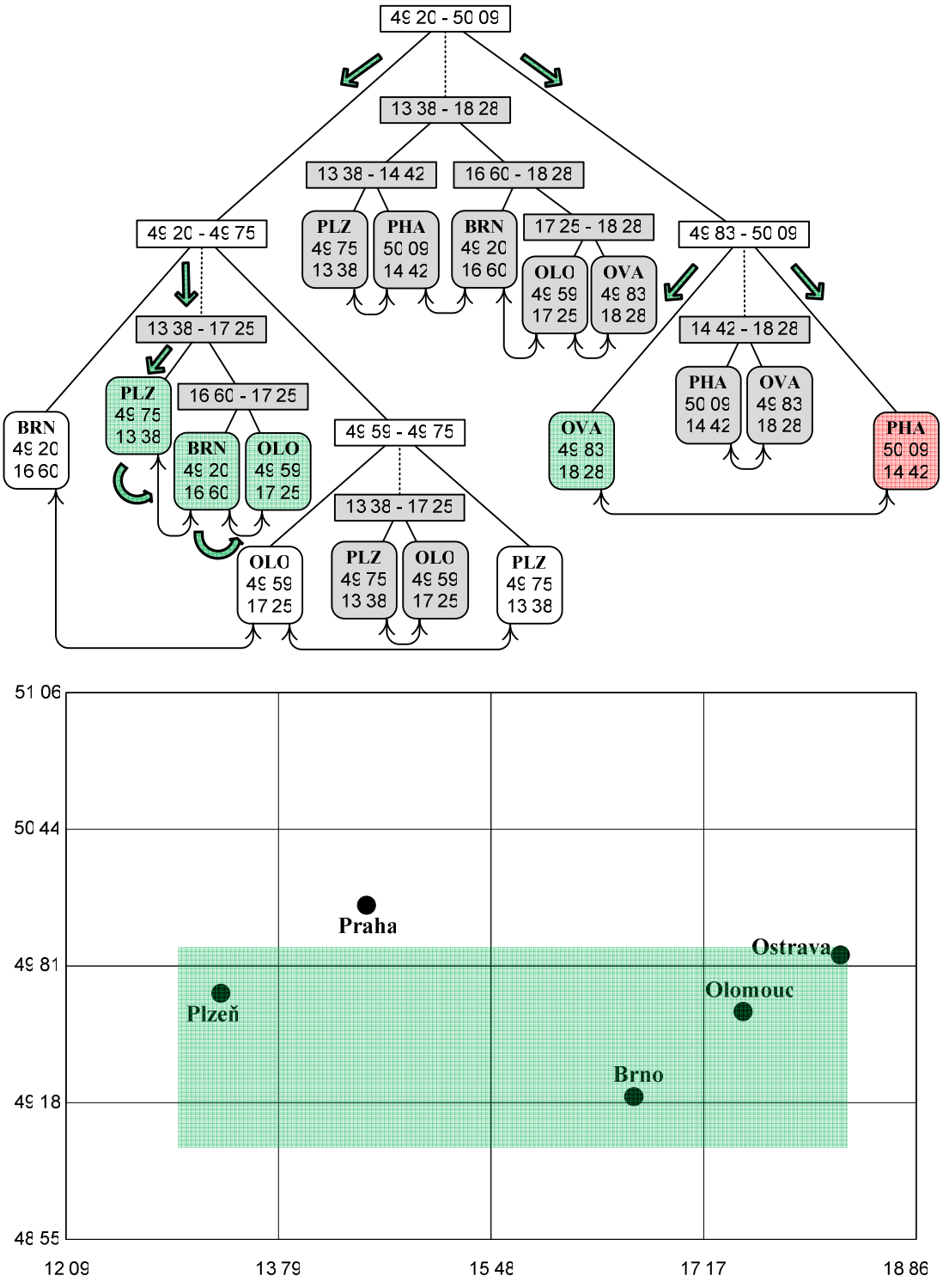
3.2.5. Intervalové hledání

Pro vyhledání prvků v obdélníkovém segmentu se využívá nejen hlavní strom, ale také stromy druhé dimenze. Hledání opět začíná od kořene stromu směrem k listům. Při dosažení plnohodnotného prvku (listu stromu) se zkontrolují jeho souřadnice, a pokud spadají do hledaného segmentu, vykoná se na aktuálním prvku zadaná akce. Při traverzování přes navigační vrcholy se rozhoduje, jakým způsobem v hledání pokračovat. Uplatňují se přitom následující pravidla:

- jedná-li se o navigační vrchol ve stromu první dimenze a celý jeho interval patří do daného segmentu, pokračuje hledání do příslušného stromu druhé dimenze,
- jedná-li se o navigační vrchol ve stromu druhé dimenze a celý jeho interval patří do daného segmentu, provede se prohlídka celého podstromu (při ní se dotraverzuje k nejlevějšímu listu podstromu a projde se celý zřetězený seznam),
- pro navigační vrcholy ve stromu první i druhé dimenze navíc platí, že pokud do daného segmentu patří pouze část z jejich intervalu, pokračuje hledání levým i pravým podstromem,
- neplatí-li žádná výše uvedená podmínka, hledání dále nepokračuje.

Obrázek 18 definuje strukturu, v níž se prohledává obdélníkový segment, kde:

- zeměpisná šířka = 49,00 až 49,83,
- zeměpisná délka = 13,00 až 18,28.



Obrázek 18: Intervalové hledání v range stromu, zdroj [autor]

Pseudokód

Při prvním volání funkce se parametr *vrchol* rovná kořenu range stromu a parametr *dimenzeX* hodnotě *true*.

```
NajdiInterval(↓x1, ↓y1, ↓x2, ↓y2, ↓vrchol, ↓dimenzeX, ↓akce)
  jestliže vrchol ≠ neexistuje pak
    // při dosažení plnohodnotného prvku se zkontrolují jeho souřadnice
    jestliže Platny(vrchol) = true pak
      jestliže SouradniceX(vrchol) = x & SouradniceY(vrchol) = y pak
        akce(vrchol);
    // v opačném případě je zvolen směr hledání
    jinak
      jestliže dimenzeX = true pak
        // patří-li celý interval do hledaného rozsahu, hledá se ve druhé dimenzi
        jestliže x1 ≤ SouradniceX(vrchol) & SouradniceY(vrchol) ≤ x2 pak
          NajdiInterval(x1, y1, x2, y2, DruhaDimenze(vrchol), !dimenzeX, akce);
        // patří-li část intervalu do hledaného rozsahu, pokračuje se oběma syny
        jinak jestliže SouradniceX(vrchol) ≤ x1 || x2 ≤ SouradniceY(vrchol) pak
          NajdiInterval(x1, y1, x2, y2, LevySyn(vrchol), dimenzeX, akce);
          NajdiInterval(x1, y1, x2, y2, PravySyn(vrchol), dimenzeX, akce);
      jinak
        // patří-li celý interval do hledaného rozsahu, provede se prohlídka podstromu
        jestliže y1 ≤ SouradniceX(vrchol) & SouradniceY(vrchol) ≤ y2 pak
          Prohlidka(vrchol, akce);
        // patří-li část intervalu do hledaného rozsahu, pokračuje se oběma syny
        jinak jestliže SouradniceX(vrchol) ≤ y1 || y2 ≤ SouradniceY(vrchol) pak
          NajdiInterval(x1, y1, x2, y2, LevySyn(vrchol), dimenzeX, akce);
          NajdiInterval(x1, y1, x2, y2, PravySyn(vrchol), dimenzeX, akce);
```

3.3. Prioritní vyhledávací strom

Prioritní vyhledávací strom slouží k uchovávání dvourozměrných dat, přičemž se jedná o strukturu vycházející z kombinace binárního vyhledávacího stromu a binární haldy. V níže uvedeném příkladě platí pravidla binárního vyhledávacího stromu pro souřadnice zeměpisné šířky a pravidla binární hlady (konkrétně tzv. max-haldy) pro souřadnice zeměpisné délky.

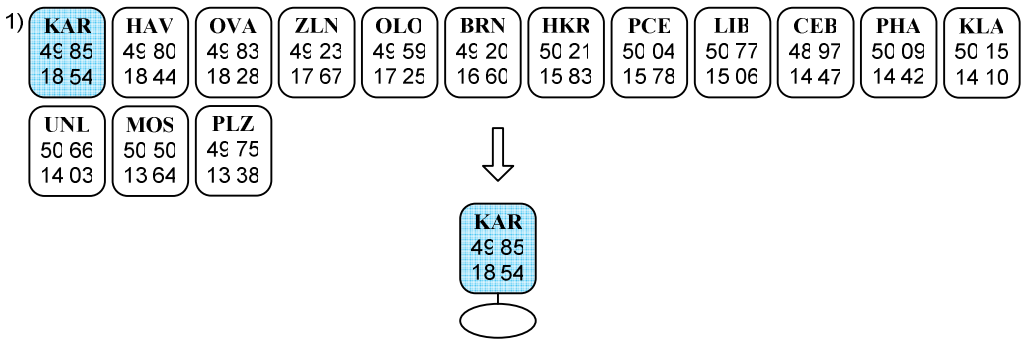
Na rozdíl od klasického binárního vyhledávacího stromu však nejsou prvky rozdělovány přímo podle hodnoty zeměpisné šířky otce, ale podle hodnoty jeho atributu *hranice*. Důvodem pro zavedení tohoto atributu byla možnost vybudování vyvážené struktury, pokud jsou vstupní data předem známa. V případě nezavedení atributu *hranice* by totiž mohla nastat situace, že prvek s nejvyšší zeměpisnou délkou, který se umístí do kořene stromu, by mohl zároveň být prvkem s nejvyšší nebo nejnižší zeměpisnou šířkou. Při následném dělení prvků by tak došlo k situaci, že by se všechny zbylé prvky vložily pouze do jednoho podstromu kořene, což by znemožnilo vybudování vyvážené stromové struktury. Hodnota atributu *hranice* se volí takovým způsobem, aby zbylé prvky rozdělila na dvě stejně velké části. [1][2][6][11]

3.3.1. Vybudování struktury

Nejsou-li vstupní data k dispozici předem, je struktura budována průběžně pomocí opakovaného volání operace pro vložení prvku. Pokud však jsou vstupní data předem známá, lze vybudovat vyváženou stromovou strukturu. V tomto konkrétním případě se při organizaci prvků podle hodnot zeměpisných délek dodržují pravidla max-haldy, postup pro vybudování prioritního vyhledávacího stromu je tedy následující.

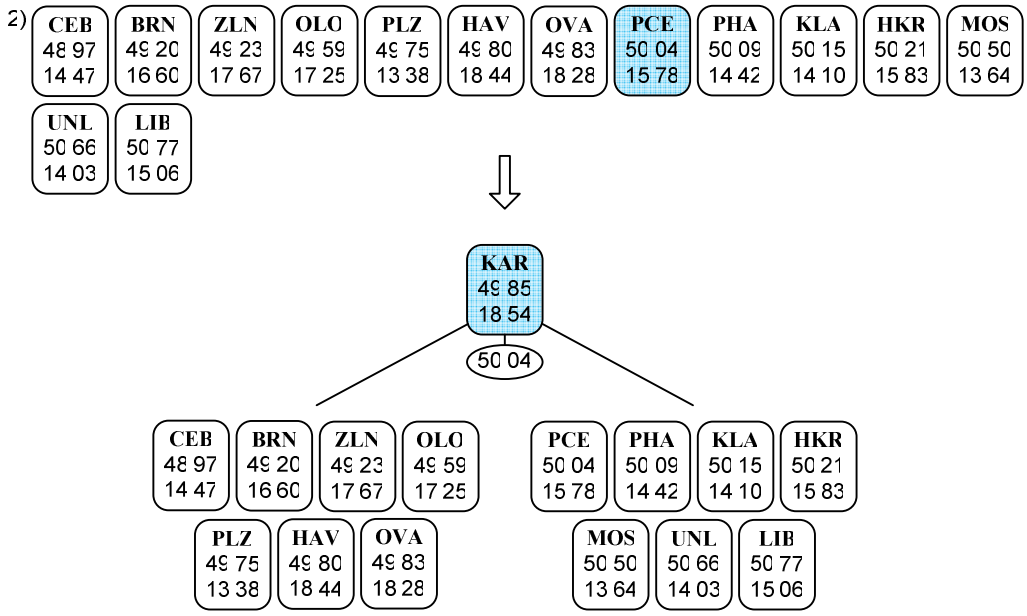
V prvním kroku dojde k seřazení všech prvků podle zeměpisné délky a prvek s její nejvyšší hodnotou stane kořenem prioritního vyhledávacího stromu. Ve druhém kroku se zbylé prvky seřadí podle zeměpisné šířky a nalezne se medián. Prvky ležící před mediánem se vloží do levého podstromu, zbylé prvky do pravého podstromu. Navíc hodnota zeměpisné šířky mediánu určuje hodnotu atributu *hranice* pro kořen stromu.

Graficky vysvětlují princip budování stromu následující obrázky.



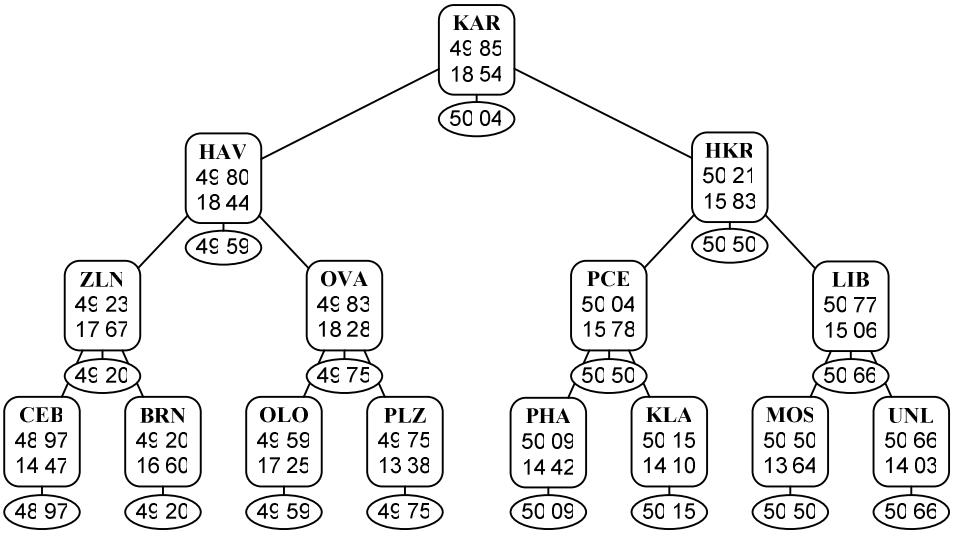
Obrázek 19: Princip budování prioritního vyhledávacího stromu (první krok), zdroj [autor]

V prvním kroku byl kořenem stromu zvolen prvek *Karviná* (obrázek 19). Ve druhém kroku se ostatní prvky seřadí podle hodnot zeměpisné šířky a dojde k jejich rozdělení do dvou skupin. Zároveň se podle hodnoty zeměpisné šířky mediánu nastaví hodnota atributu *hranice* (obrázek 20).



Obrázek 20: Princip budování prioritního vyhledávacího stromu (druhý krok), zdroj [autor]

Předchozí postup se pro obě rozdělené skupiny prvků cyklicky opakuje do okamžiku, než zbude v každé skupině pouze jeden prvek, který se stane listem prioritního vyhledávacího stromu. Výsledný stav struktury po vybudování vykresluje obrázek 21.



Obrázek 21: Vybudovaný prioritní vyhledávací strom, zdroj [autor]

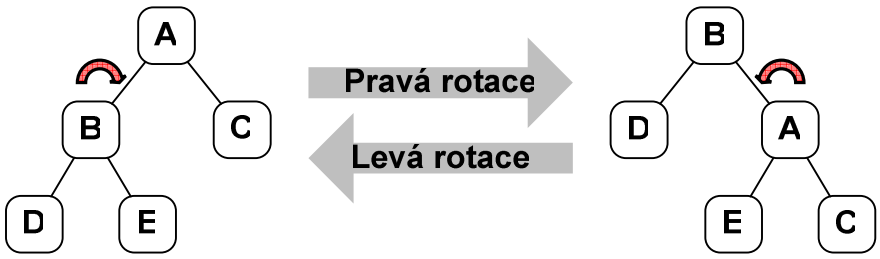
Pseudokód

```
Vybuduj(↓prvky)
  jestliže prvky = neexistuje || Pocet(prvky) = 0 pak
    Konec;
  jinak
    pocet := Pocet(prvky);
    SeradPodleY(prvky);
    // poslední prvek v seznamu se stane kořenem a ze seznamu se odebere
    koren := prvky[Pocet(prvky) – 1];
    Odeber(prvky, Pocet(prvky) – 1);
    // určí se hodnota atributu hranice a případně se vybudují podstromy
    jestliže Pocet(prvky) = 0 pak
      Hranice(koren) := SouradniceX(koren);
    jinak
      SeradPodleX(prvky);
      median := Pocet(prvky) / 2;
      Hranice(koren) := SouradniceX(prvky[median]);
      VybudujPodstrom(prvky[0 .. median - 1], koren, true);
      VybudujPodstrom(prvky[median .. Pocet(prvky) – 1], koren, false);

VybudujPodstrom(↓prvky, ↓predek, ↓levyPodstrom)
  jestliže Pocet(prvky) = 0
    pak Konec;
  jinak
    // poslední prvek v seznamu se stane potomkem předka a ze seznamu se odebere
    SeradPodleY(prvky);
    jestliže levyPodstrom = true pak
      LevySyn(predek) = prvky[Pocet(prvky) – 1];
    jinak
      PravySyn(predek) = prvky[Pocet(prvky) – 1];
    Odeber(prvky, Pocet(prvky) – 1);
    // určí se hodnota atributu hranice a případně se vybudují podstromy
    jestliže Pocet(prvky) = 0 pak
      jestliže levyPodstrom = true pak
        Hranice(LevySyn(predek)) := SouradniceX(LevySyn(predek));
      jinak
        Hranice(PravySyn(predek)) := SouradniceX(PravySyn(predek));
```

```
jinak
  SeradPodleX(prvky);
  median := Pocet(prvky) / 2;
  jestliže levyPodstrom = true pak
    Hranice(LevySyn(predek)) := SouradniceX(LevySyn(prvky[median]));
    VybudujPodstrom(prvky[0 .. median - 1],  LevySyn(predek), true);
    VybudujPodstrom(prvky[median .. Pocet(prvky) – 1], LevySyn(predek), false);
  jinak
    Hranice(PravySyn(predek)) := SouradniceX(PravySyn(prvky[median]));
    VybudujPodstrom(prvky[0 .. median - 1],  PravySyn(predek), true);
    VybudujPodstrom(prvky[median .. Pocet(prvky) – 1], PravySyn(predek), false);
```

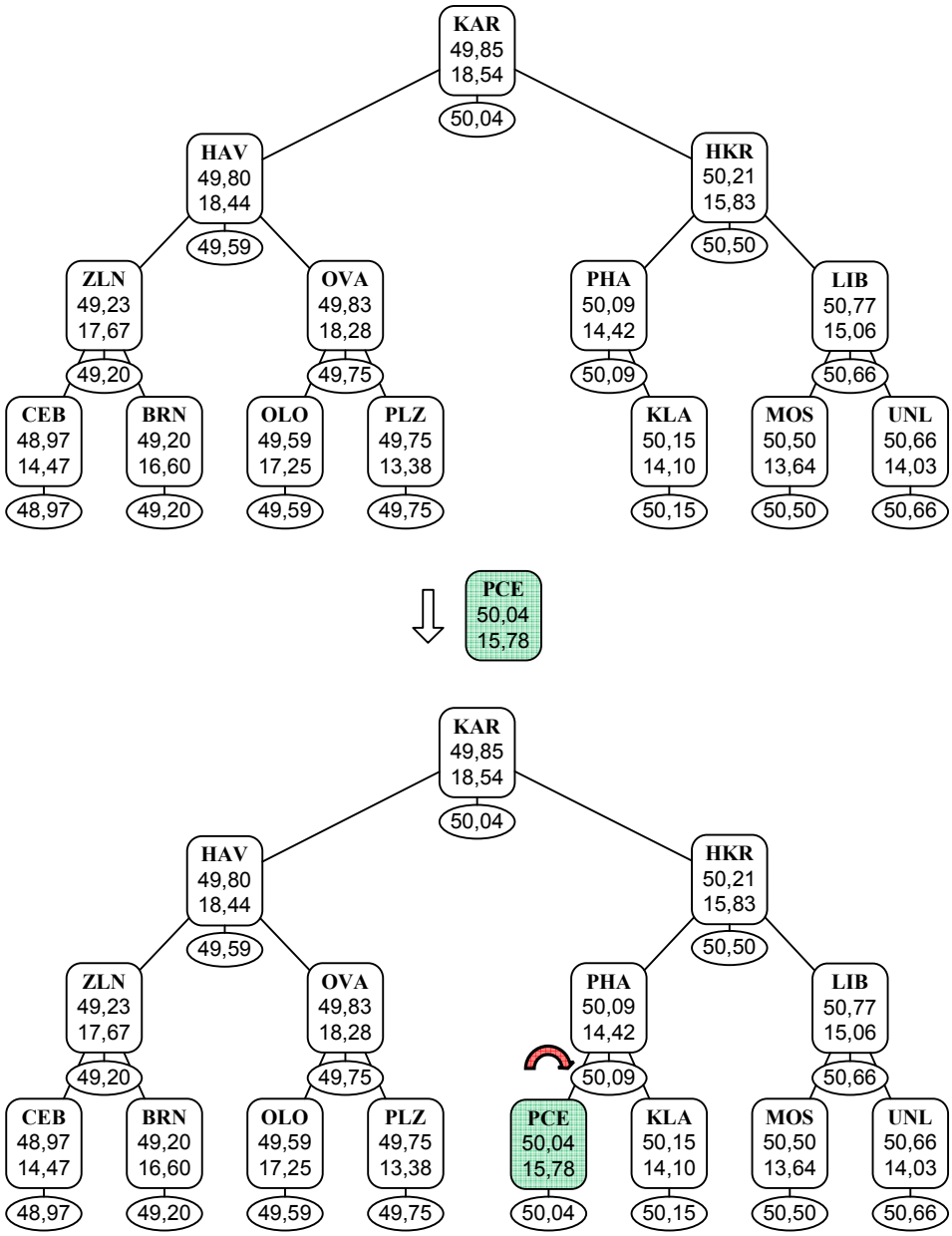
3.3.2. Vložení prvku

Vložení prvku do prioritního vyhledávacího stromu lze rozdělit do dvou fází. V té první traverzuje vkládaný prvek směrem od kořene stromu dolů na příslušnou pozici. Při jeho průchodu strukturou se v každém vrcholu porovnává hodnota zeměpisné šířky s atributem *hranice* a na základě tohoto porovnání se zvolí následující směr traverzování. Druhá fáze má za cíl zajistit korektní umístění prvku dle pravidel binární max-haldy. Pokud je hodnota zeměpisné délky u rodiče nižší než hodnota zeměpisné délky u vkládaného prvku, provádějí se jednoduché levé nebo pravé rotace (obrázek 22), dokud není pozice vkládaného prvku vyhovující.



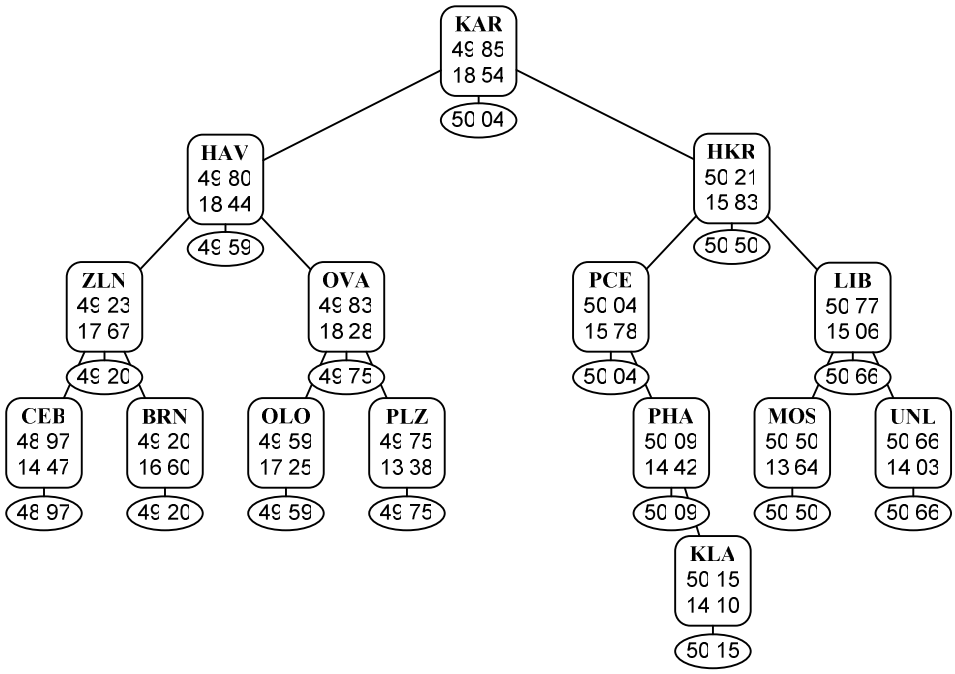
Obrázek 22: Princip jednoduchých rotací, zdroj [autor]

Obrázek 23 zachycuje stav struktury před vložením prvku *Pardubice*, který následně traverzuje stromovou strukturou směrem od kořene dolů na příslušné místo.



Obrázek 23: Vložení prvku do prioritního vyhledávacího stromu (stav před vložením), zdroj [autor]

Z obrázku je patrné, že umístění vloženého prvku (*Pardubice*) a jeho rodiče (*Praha*) nevyhovuje haldovému uspořádání. Musí se tedy provést jednoduchá pravá rotace, jak naznačuje šipka. Po jejím vykonání se stane rodičem vloženého prvku prvek *Hradec Králové* a tato situace již pravidlům haldového uspořádání vyhovuje, což graficky dokazuje obrázek 24.



Obrázek 24: Vložení prvku do prioritního vyhledávacího stromu (stav po vložení), zdroj [autor]

Pseudokód

Vloz(↓prvek)

```
pocet := pocet + 1;
Hranice(prvek) := SouradniceX(prvek);
jestliže koren = neexistuje pak
  koren := prvek;
jinak
  // traverzování na příslušnou pozici dle principu binárního vyhledávacího stromu
  pomocny := koren;
  while (true)
    // traverzování vlevo
    jestliže SouradniceX(prvek) < Hranice(pomocny) pak
      jestliže LevySyn(pomocny) = neexistuje pak
        LevySyn(pomocny) := prvek;
        KonecCyklu;
      jinak
        pomocny := LevySyn(pomocny);
    // traverzování vpravo
    jinak
      jestliže PravySyn(pomocny) = neexistuje pak
        PravySyn(pomocny) := prvek;
```

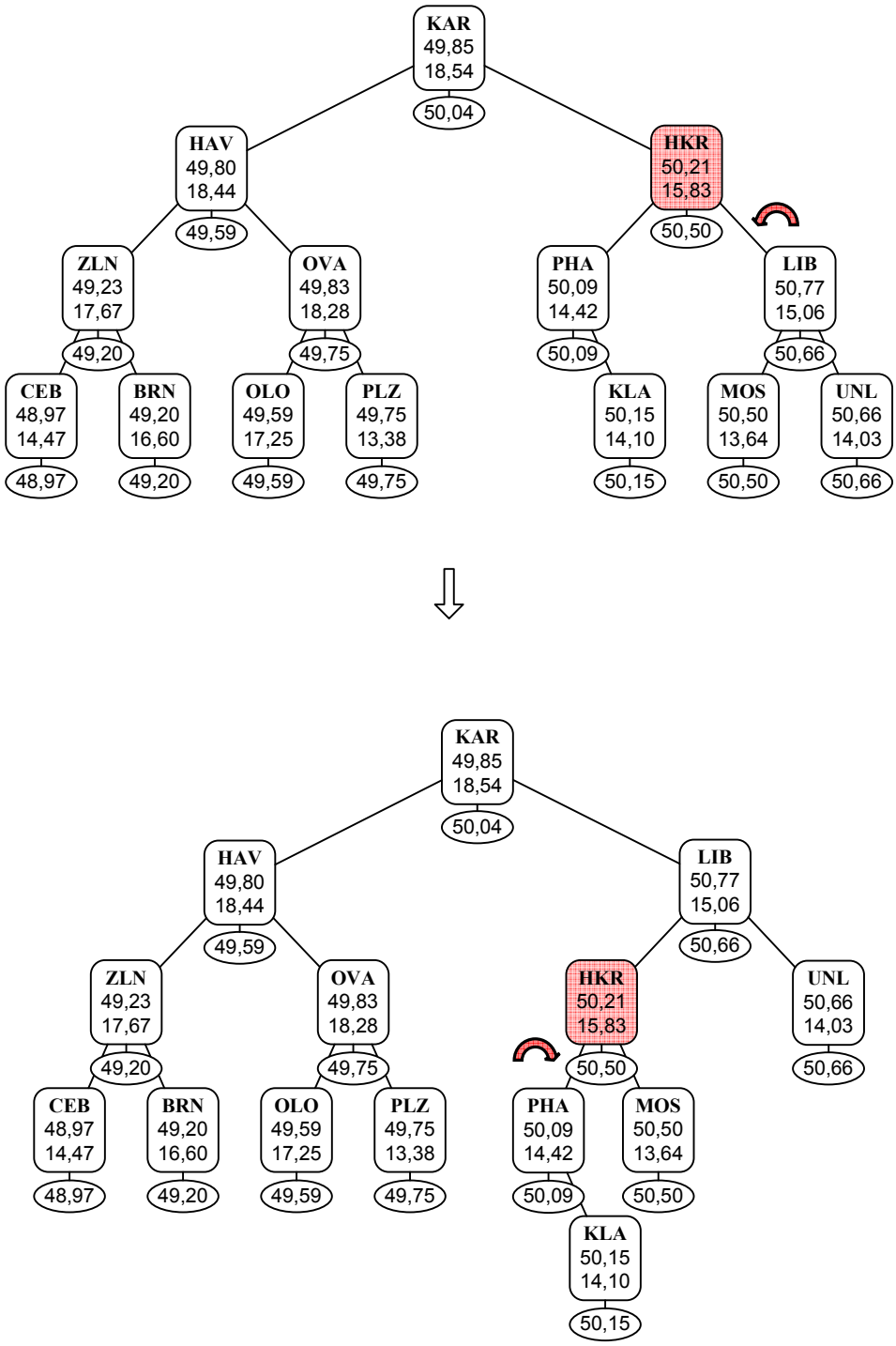
```
        KonecCyklu;
      jinak
        pomocny := PravySyn(pomocny);
  // případné provádění rotací, dokud zeměpisné délky nevyhovují max-haldě
  while(true)
    jestliže SouradniceY(prvek) ≤ SouradniceY(Otec(prvek)) pak
      KonecCyklu;
    jestliže prvek = LevySyn(Otec(prvek)) pak
      PravaRotace(Otec(prvek));
    jinak
      LevaRotace (Otec(prvek));
```

3.3.3. Odebrání prvku

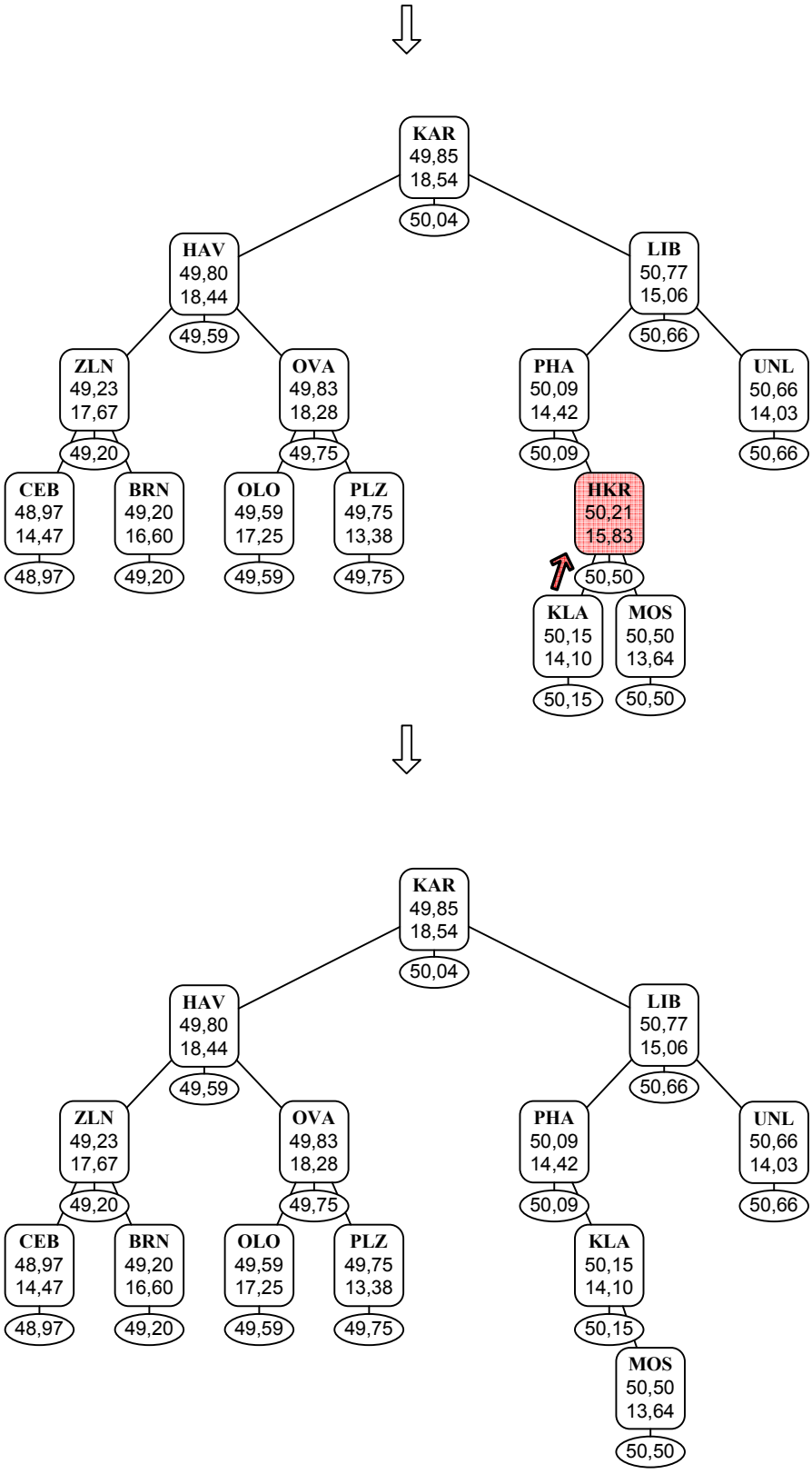
Při odebírání prvku z prioritního vyhledávacího stromu podle zadaných souřadnic mohou nastat tři různé případy:

- je-li odebíraný prvek listem stromu, lze ho přímo odebrat,
- jsou-li synové odebíraného prvku listy stromu, dojde k odstranění prvku tím, že se nahradí synem s vyšší hodnotou zeměpisné délky,
- v ostatních případech se odebíraný prvek přemísťuje pomocí jednoduchých rotací směrem od kořene k listům (rotace jsou přitom vždy prováděny tak, aby se otcem odebíraného prvku stal syn, který má vyšší hodnotu zeměpisné délky), dokud nejsou synové odebíraného prvku listy stromu, poté již lze aplikovat předchozí pravidlo.

Odstranění prvku z prioritního vyhledávacího stromu je graficky znázorněno na obrázku 25, kde se z výchozí struktury odebírá prvek se souřadnicemi zeměpisné šířky = 50,21 a zeměpisné délky = 15,83 (*Hradec Králové*). Pomocí jednoduchých levých a pravých rotací se tento prvek přemísťuje do pozice, kdy se jeho synové stanou listy stromu. Následně dojde k odstranění prvku tím, že se nahradí tím synem, jenž má vyšší hodnotu zeměpisné délky (obrázek 26).



Obrázek 25: Odebrání prvku z prioritního vyhledávacího stromu (rotování prvku), zdroj [autor]



Obrázek 26: Odebrání prvku z prioritního vyhledávacího stromu (stav po odebrání), zdroj [autor]

Pseudokód

```
Odeber(↓x, ↓y, ↑prvek)
  odebirany := Najdi(x, y);
  jestliže odebirany = neexistuje pak
    return neexistuje;
  pocet := pocet – 1;
  while (true)
    // je-li je odebíraný prvek listem, odstraní se
    jestliže LevySyn(odebirany) = neexistuje && PravySyn(odebirany) = neexistuje pak
      jestliže odebirany = LevySyn(Otec(odebirany)) pak
        LevySyn(Otec(odebirany)) := neexistuje;
      jinak
        PravySyn(Otec(odebirany)) := neexistuje;
      return odebirany;
    // jsou-li synové odebíraného prvku listy stromu, dojde k jeho nahrazení
    jinak jestliže MaxHloubkaPodstromu((odebirany) ≤ 1 pak
      jestliže LevySyn(odebirany) ≠ neexistuje & PravySyn(odebirany) ≠ neexistuje pak
        jestliže SouradniceY(LevySyn(odebirany)) > SouradniceY(PravySyn(odebirany))
          pak Nahradit(odebirany, LevySyn(odebirany));
        jinak
          Nahradit(odebirany, PravySyn(odebirany));
      jinak jestliže LevySyn(odebirany) = neexistuje pak
        Nahradit(odebirany, PravySyn(odebirany
      jinak
        Nahradit(odebirany, LevySyn(odebirany));
      return odebirany;
    // nejsou-li synové odebíraného prvku listy stromu, bude prvek „prototován“
    jinak
      jestliže LevySyn(odebirany) ≠ neexistuje & PravySyn(odebirany) ≠ neexistuje pak
        jestliže SouradniceY(LevySyn(odebirany)) > SouradniceY(PravySyn(odebirany))
          pak PravaRotace(odebirany);
        jinak
          LevaRotace(odebirany);
      jinak jestliže LevySyn(odebirany) = neexistuje pak
        LevaRotace(odebirany);
      jinak
        PravaRotace(odebirany);
```

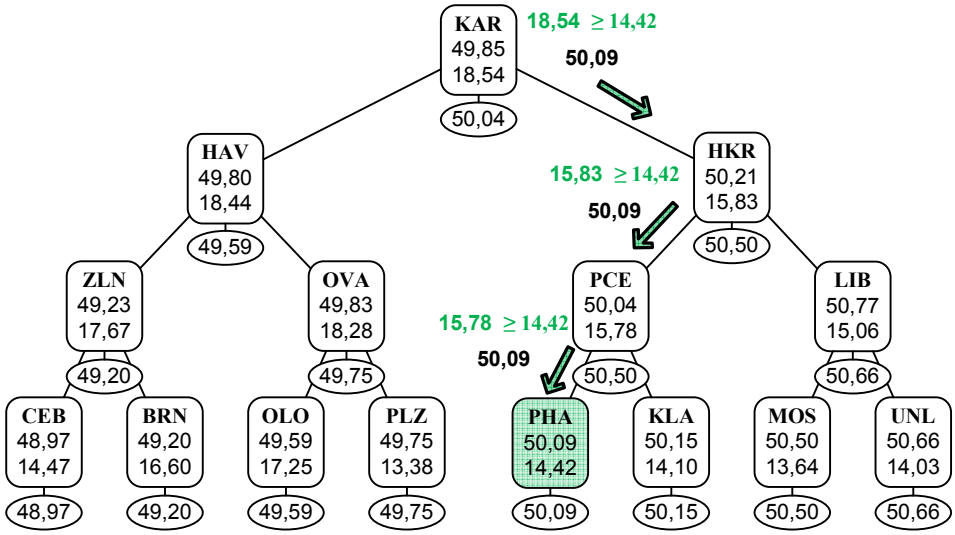
3.3.4. Hledání prvku

Hledání prvku se zadanými souřadnicemi začíná od kořene prioritního vyhledávacího stromu směrem k listům. Během traverzování stromovou strukturou se při průchodu každým vrcholem porovnají jeho souřadnice se souřadnicemi hledanými. V případě shody je hledaným prvkem aktuální prvek a algoritmus končí. Jinak se rozhodne o tom, zda má hledání pokračovat a pokud ano, zvolí se směr dalšího hledání. Pro přijetí tohoto rozhodnutí slouží následující pravidla:

- je-li hodnota zeměpisné délky aktuálního vrcholu menší než hledaná zeměpisná délka, hledání končí,
- jinak:
    - je-li hodnota atributu *hranice* aktuálního vrcholu větší než hledaná zeměpisná šířka, pokračuje hledání do levého podstromu,
    - jinak pokračuje hledání do pravého podstromu.

Pokud je dotraverzováno k listu stromu, a přesto není nalezen vyhovující prvek, lze jednoznačně určit, že ve stromové struktuře se hledaný prvek nenachází.

Postup při hledání prvku se zeměpisnou šířkou = 50,09 a zeměpisnou délkou = 14,42 ukazuje obrázek 27.



Obrázek 27: Hledání prvku v prioritním vyhledávacím stromu, zdroj [autor]

Pseudokód

```
Najdi(↓x, ↓y, ↑prvek)
  pomocny := koren;
  while(true)
    jestliže pomocny = neexistuje pak
      return neexistuje;
    // porovnání souřadnic aktuálního prvku se souřadnicemi hledanými
    jestliže SouradniceX(pomocny) = x && SouradniceY(pomocny) = y pak
      return pomocny;
    // kontrola, zda má smysl pokračovat v hledání
    jestliže SouradniceY(pomocny) > y pak
      return neexistuje;
    // pokud ano, je zvolen směr, kterým se má hledat
    jestliže Hranice(pomocny) > x pak
      pomocny := LevyPotomek(pomocny);
    jinak
      pomocny := PravyPotomek(pomocny);
```

3.3.5. Intervalové hledání

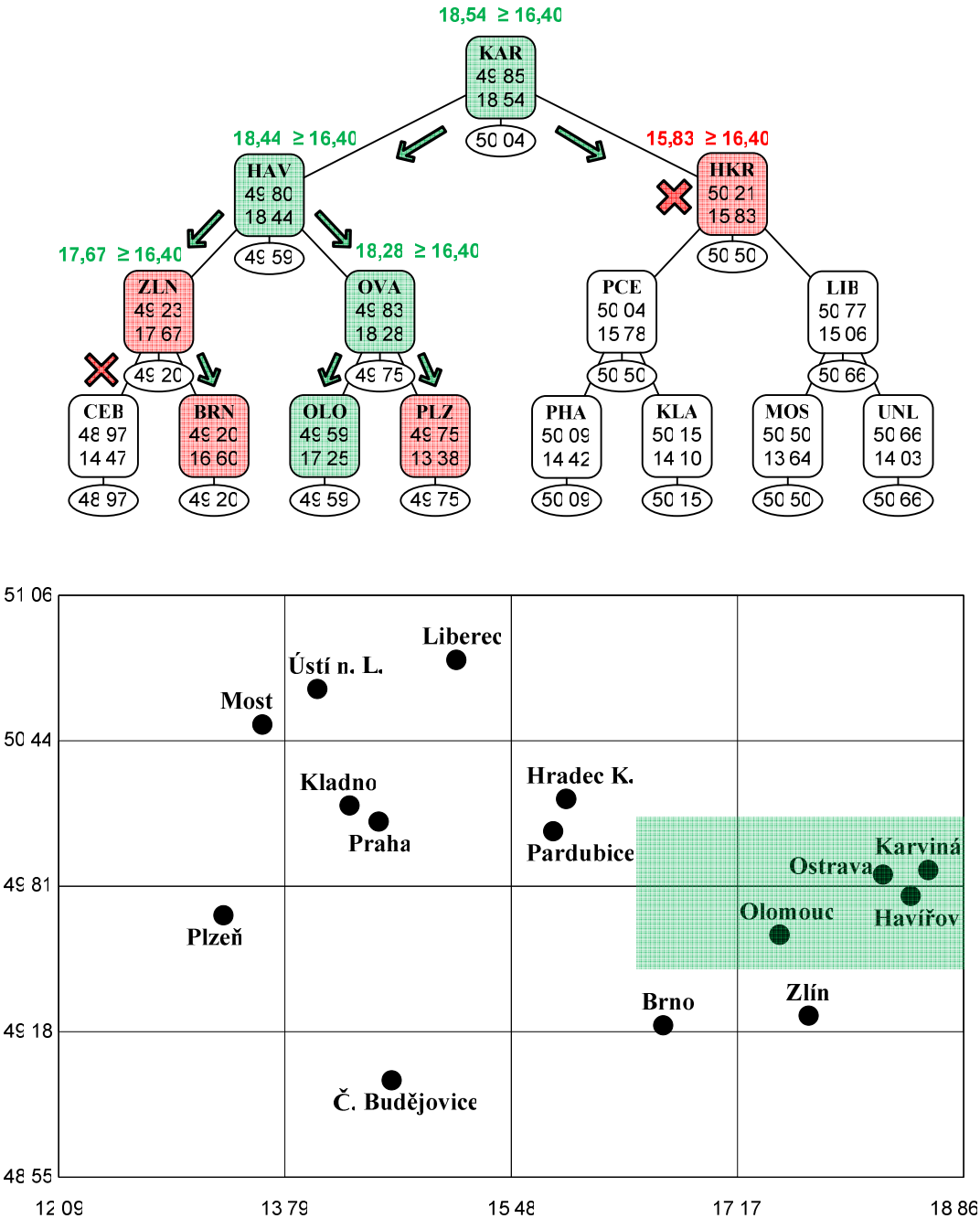
Hledání prvků v obdélníkovém segmentu opět vychází z traverzování prioritním vyhledávacím stromem směrem od kořene k listům. Pro každý prvek se přitom testuje, zda do zadaného obdélníkového segmentu patří, na vyhovující prvky se aplikuje zadaná akce. V každém vrcholu se navíc rozhoduje, zda se má pokračovat prohledáváním levého a pravého podstromu. Toto rozhodnutí vychází z pravidel:

- je-li zeměpisná délka aktuálního vrcholu menší než minimální zeměpisná délka v hledaném segmentu, hledání nepokračuje,
- neplatí-li první pravidlo a současně je zeměpisná šířka aktuálního vrcholu větší než minimální zeměpisná šířka v hledaném segmentu, pokračuje hledání do levého podstromu,
- neplatí-li první pravidlo a současně není zeměpisná šířka aktuálního vrcholu větší než maximální zeměpisná šířka v hledaném segmentu, pokračuje hledání do pravého podstromu,

- neplatí-li žádná výše uvedená podmínka, hledání končí.

Obrázek 28 vysvětluje graficky princip vyhledávání všech prvků z obdélníkového segmentu, kde:

- zeměpisná šířka = 49,45 až 50,12,
- zeměpisná délka = 16,40 až 18,86.



Obrázek 28: Intervalové hledání v prioritním vyhledávacím stromu, zdroj [autor]

Pseudokód

Při prvním volání této funkce se parametr *vrchol* rovná kořenu prioritního vyhledávacího stromu.

```
NajdiInterval(↓x1, ↓y1, ↓x2, ↓y2, ↓vrchol, ↓akce)
  jestliže vrchol ≠ neexistuje pak
    // kontrola, zda prvek patří do zadaného segmentu
    jestliže x1 ≤ SouradniceX(vrchol)  &&  SouradniceX(vrchol) ≤ x2
    & y1 ≤ SouradniceY(vrchol)  &&  SouradniceY(vrchol) ≤ y2 pak
      akce(vrchol);
    // rozhodnutí, zda se má pokračovat prohledáváním levého podstromu
    jestliže LevySyn(vrchol) ≠ neexistuje & y1 ≤ SouradniceY(vrchol)
    & x1 < Hranice(vrchol) pak
      NajdiInterval(x1, y1, x2, y2, LevySyn(vrchol), akce);
    // rozhodnutí, zda se má pokračovat prohledáváním pravého podstromu
    jestliže LevySyn(vrchol) ≠ neexistuje & y1 ≤ SouradniceY(vrchol)
    & Hranice(vrchol) ≤ x1 pak
      NajdiInterval(x1, y1, x2, y2, PravySyn(vrchol), akce);
```

3.4. Quad strom

Pro datovou strukturu quad strom existuje více možných implementací. Tato kapitola představuje tu nejzákladnější, při níž se dělení území na jednotlivé kvadranty realizuje metodou půlení intervalu. Někdy se datová struktura s tímto typem implementace označuje také jako pseudoquad strom. Vyznačuje se tím, že plnohodnotné prvky se nacházejí pouze na úrovni listů, obdobně jako tomu bylo u range stromu. Ostatní vrcholy stromu tvoří navigační strukturu, přičemž každý navigační vrchol má schopnost dělit region na čtyři oblasti – severozápadní, severovýchodní, jihozápadní, jihovýchodní kvadrant. Tyto čtyři kvadranty sousedí s daným vrcholem, a pokud se v některém z nich nachází více než jeden prvek, je tento kvadrant dále cyklicky dělen. Cyklické dělení regionu na kvadranty pokračuje do doby, než zbude v každém kvadrantu maximálně jeden prvek. [1][2][4]

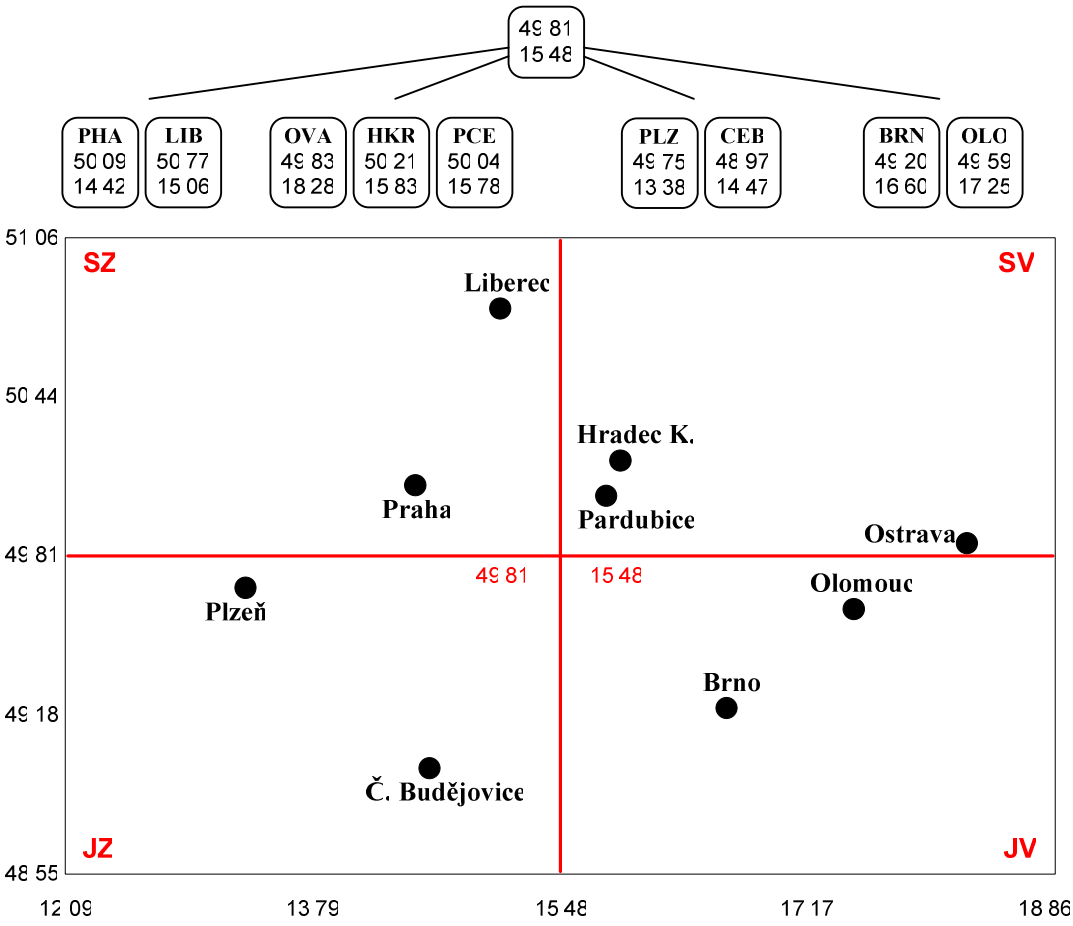
3.4.1. Vybudování struktury

Na rozdíl od předchozích struktur má psedoquad strom jednu významnou odlišnost. Podoba vybudované struktury pro danou množinu prvků je totiž vždy stejná, bez ohledu na

to, zda jsou prvky známy předem nebo zda se struktura buduje průběžně pomocí opakovaného volání operace pro vložení prvku. Tuto vlastnost zajišťuje fakt, že dělení území na jednotlivé kvadranty se realizuje právě metodou půlení intervalu.

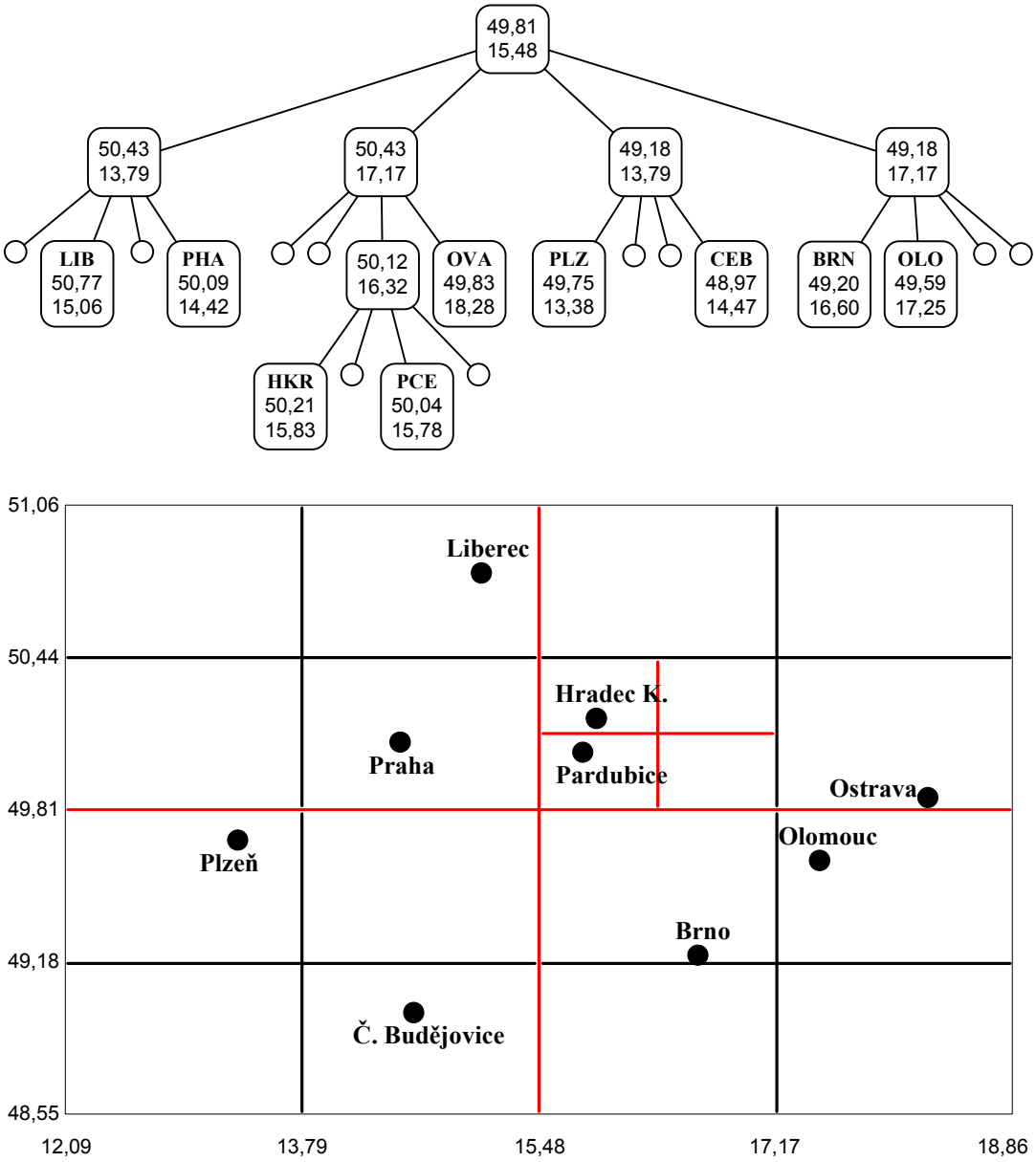
Jak již bylo zmíněno, princip vybudování stromové struktury spočívá v cyklickém dělení regionu na kvadranty a toto dělení pokračuje cyklicky do okamžiku, než zbude v každém kvadrantu jediný prvek. Ve specifickém případě, kdy je pro vybudování struktury předán jediný prvek, se žádné dělení nerealizuje a plnohodnotný prvek se stane přímo kořenem stromu. V ostatních případech se pomocí metody půlení intervalu rozdělí území na čtyři stejně velké oblasti, čímž je definován střed tohoto dělení. Do kořene stromu se následně vloží nový navigační vrchol se čtyřmi pozicemi pro potomky (pozice některých potomků mohou zůstat prázdné) a se souřadnicemi, které odpovídají středu dělení. Pro kvadranty, v nichž se stále nachází více prvků než jeden, dělení cyklicky pokračuje.

Princip budování quad stromu s využitím metody půlení intervalu přibližuje obrázek 29.



Obrázek 29: Princip budování quad stromu, zdroj [autor]

Z obrázku je patrné, že se v jednotlivých kvadrantech nachází stále více než jeden prvek, v dělení se tedy musí cyklicky pokračovat. Výsledný stav quad stromu po vybudování zachycuje obrázek 30.



Obrázek 30: Vybudovaný quad strom, zdroj [autor]

Pseudokód

Vybuduj(↓prvky)

jestliže prvky = neexistuje || Pocet(prvky) = 0 pak

Konec;

jinak

```
pocet := Pocet(prvky);
// pro jeden prvek není nutné dělit region na kvadranty, prvek se stane kořenem
jestliže Pocet(prvky) = 1 pak
  koren := prvky[0];
// v opačném případě se region na kvadranty rozdělí
jinak
  koren := VytvorNovyPrvek();
  NastavSouradniceNavigacnihoPrvku(koren);
  VybudujPodstrom(koren, prvky);
```

VybudujPodstrom(↓vrchol, ↓prvky)

```
// vytvoření seznamů, které odpovídají kvadrantům – do nich se prvky rozdělí
prvkySZ := VytvorSeznamPrvku();
prvkySV := VytvorSeznamPrvku();
prvkyJZ := VytvorSeznamPrvku();
prvkyJV := VytvorSeznamPrvku();
RozdelPrvky(vrchol, prvkySZ, prvkySV, prvkyJZ, prvkyJV);
// kontrola severozápadního kvadrantu (seznam prvkySZ)
// je-li v SZ kvadrantu jediný prvek, stane se potomkem aktuálního vrcholu
jestliže Pocet(prvkySZ) = 1 pak
  SynSZ(vrchol) :=  prvkySZ[0];
// je-li v SZ kvadrantu více prvků, vznikne navig. vrchol a dělení cyklicky pokračuje
jinak jestliže Pocet(prvkySZ) > 1 pak
  SynSZ(vrchol) :=  VytvorNovyPrvek();
  NastavSouradniceNavigacnihoPrvku(SynSZ(vrchol);
  VybudujPodstrom(SynSZ(vrchol), prvkySZ prvky);
// stejný postup se provede i pro tři zbývající kvadranty (tedy pro SV, JZ, JV)
…
```

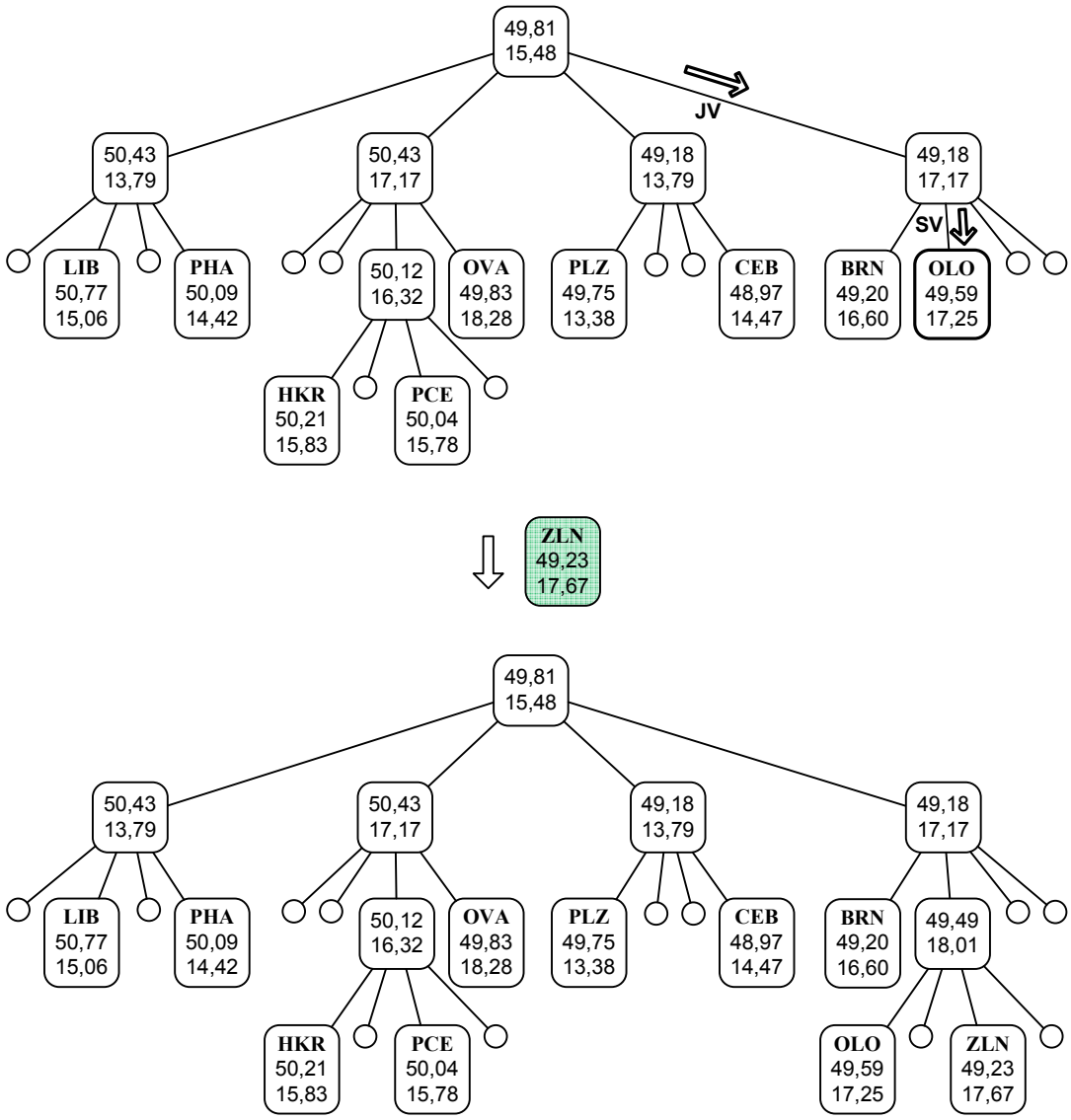
3.4.2. Vložení prvku

Při vkládání nového prvku do quad stromu traverzuje tento prvek stromovou strukturou směrem od kořene na příslušnou pozici. V každém navigačním vrcholu se určí, ve kterém kvadrantu vkládaný prvek leží, čímž se zároveň určí další směr traverzování. Po dosažení úrovně listů mohou nastat dvě situace:

- bylo-li dotraverzováno na pozici, která je neobsazena, lze na ni vkládaný prvek vložit,

- bylo-li dotraverzováno na pozici, na které se již nachází plnohodnotný prvek, vznikne na této pozici nový navigační vrchol, který rozdělí příslušný region na čtyři kvadranty a jeho syny se následně stanou vkládaný prvek a prvek, k němuž bylo dotraverzováno.

Vložení prvku *Zlín* do původní struktury přibližuje obrázek 31. Vzhledem k tomu, že traverzování skončilo na pozici, kde se již nachází prvek *Olomouc*, vytvoří se nový navigační vrchol, jehož potomky se stanou oba plnohodnotné prvky.



Obrázek 31: Vložení prvku do quad stromu, zdroj [autor]

Pseudokód

Vloz(↓prvek)

pocet := pocet + 1;

```
jestliže koren = neexistuje pak
  koren := prvek;
jinak
  pomocny := koren;
  // pro navig. vrcholy se volí směr podle kvadrantu, do něhož vkládaný prvek patří
  while (Platny(pomocny) = false)
    // traverzování do severozápadního kvadrantu
    jestliže SouradniceX(pomocny) ≤ SouradniceX(vrchol)
    & SouradniceY(pomocny) > SouradniceY(vrchol) pak
      jestliže SynSZ(pomocny) = neexistuje pak
        SynSZ(pomocny) := prvek;
        Konec;
      jinak
        pomocny := SynSZ(pomocny);
    // traverzování do severovýchodního kvadrantu
    jinak jestliže SouradniceX(pomocny) ≤ SouradniceX(vrchol)
    & SouradniceY(pomocny) ≤ SouradniceY(vrchol) pak
      jestliže SynSV(pomocny) = neexistuje pak
        SynSV(pomocny) := prvek;
        Konec;
      jinak
        pomocny := SynSV(pomocny);
    // traverzování do jihozápadního kvadrantu
    jestliže SouradniceX(pomocny) > SouradniceX(vrchol)
    & SouradniceY(pomocny) > SouradniceY(vrchol) pak
      jestliže SynJZ(pomocny) = neexistuje pak
        SynJZ(pomocny) := prvek;
        Konec;
      jinak
        pomocny := SynJZ(pomocny);
    // traverzování do jihovýchodního kvadrantu
    jinak
      jestliže SynJV(pomocny) = neexistuje pak
        SynJV(pomocny) := prvek;
        Konec;
      jinak
        pomocny := SynJV(pomocny);
  // bylo-li dotraverzováno k plnohodnotnému prvku, je nutné dělení kvadrantu
```

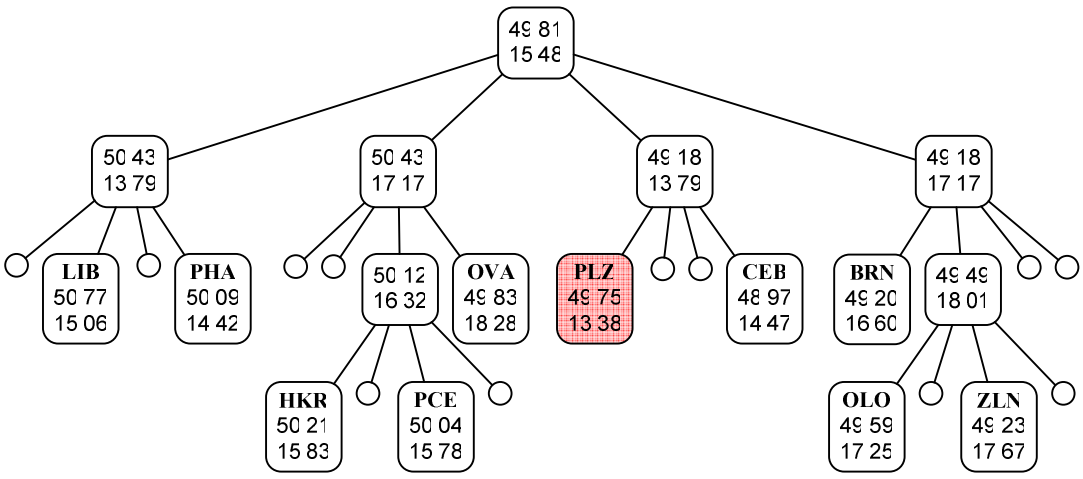
```
// vytvoření seznamu pro prvky, z nichž se vybuduje podstrom navig. vrcholu
prvky := VytvorSeznamPrvku();
prvky.Vloz(prvek);
prvky.Vloz(pomocny);
novy := VytvorNovyPrvek();
Nahradit(Otec(pomocny), novy);
NastavSouradniceNavigacnihoPrvku(novy);
VybudujPodstrom(novy, prvky);
```

3.4.3. Odebrání prvku

Před odebrání prvku z quad stromu dojde na základě zadaných souřadnic k jeho nalezení. Při následném odstranění prvku mohou nastat tři různé případy:

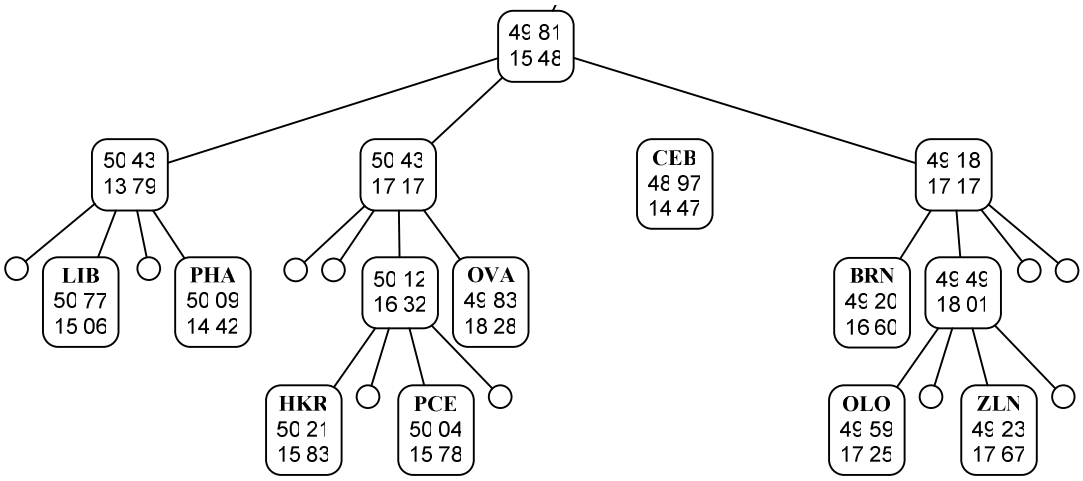
- je-li odebíraný prvek kořenem stromu, lze ho z kořene přímo odebrat,
- není-li odebíraný prvek kořenem stromu a současně má za sourozence minimálně jeden navigační vrchol nebo alespoň dva plnohodnotné prvky, lze ho přímo odebrat (původní pozice, na níž se prvek nacházel, zůstane prázdná),
- není-li odebíraný prvek kořenem stromu a současně má pouze jediného sourozence, jímž je plnohodnotný prvek, dojde k odstranění prvku a jeho sourozenec se přemístí na pozici otce.

Princip odebrání prvku ukazuje obrázek 32, na němž je z původní struktury odebírán prvek se souřadnicemi zeměpisné šířky = 49,75 a zeměpisné délky = 13,38 (*Plzeň*).



Obrázek 32: Odebrání prvku z quad stromu (stav před odebráním), zdroj [autor]

Vzhledem k tomu, že prvek *Plzeň* má jediného sourozence, kterým je plnohodnotný prvek (*České Budějovice*), dojde nejen k odstranění odebíraného prvku, ale také k nahrazení otce sourozencem (obrázek 33).



Obrázek 33: Odebrání prvku z quad stromu (stav po odebrání), zdroj [autor]

Pseudokód

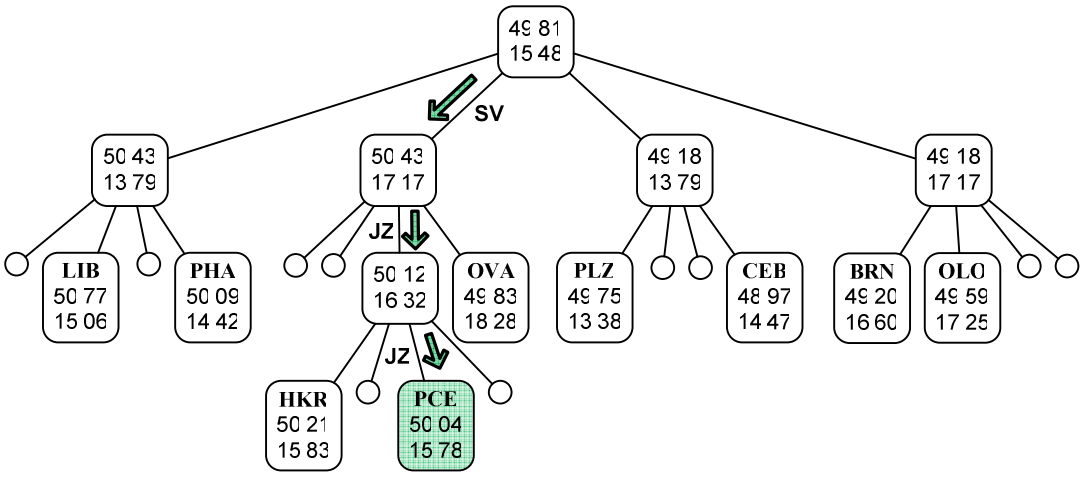
```
Odeber(↓x, ↓y, ↑prvek)
  odebirany := Najdi(x, y);
  jestliže odebirany ≠ neexistuje pak
    // je-li odebíraný prvek kořenem, je z kořene odstraněn
    jestliže Otec(odebirany) = neexistuje pak    // prvek byl kořenem
      koren := neexistuje;
    // nemá-li odebíraný prvek jen jediného plnohodnotného sourozence, odstraní se
    jinak jestliže PocetPlatnychSouroz(odebirany) ≠ 1 pak
      // zjištění pozice odebíraného prvku vzhledem k otci a jeho následné odebrání
      jestliže SynSZ(Otec(odebirany)) = odebirany pak
        SynSZ(Otec(odebirany)) := neexistuje;
      jinak jestliže SynSV(Otec(odebirany)) = odebirany pak
        SynSV(Otec(odebirany)) := neexistuje;
      jinak jestliže SynJZ(Otec(odebirany)) = odebirany pak
        SynJZ(Otec(odebirany)) := neexistuje;
      jinak
        SynJV(Otec(odebirany)) := neexistuje;
    // má-li odebíraný prvek jediného plnohodnotného sourozence, nahradí se otec
    jinak
      sourozenec := neexistuje;
```

```
    // nalezení sourozence a současné zrušení odebíraného prvku
    jestliže SynSZ(Otec(odebirany)) ≠ neexistuje pak
      jestliže SynSZ(Otec(odebirany)) = odebirany pak
        SynSZ(Otec(odebirany)) := neexistuje;
      jinak sourozenec := SynSZ(Otec(odebirany));
    jestliže SynSV(Otec(odebirany)) ≠ neexistuje pak
      jestliže SynSV(Otec(odebirany)) = odebirany pak
        SynSV(Otec(odebirany)) := neexistuje;
      jinak sourozenec := SynSV(Otec(odebirany));
    jestliže SynJZ(Otec(odebirany)) ≠ neexistuje pak
      jestliže SynJZ(Otec(odebirany)) = odebirany pak
        SynJZ(Otec(odebirany)) := neexistuje;
      jinak sourozenec := SynJZ(Otec(odebirany));
    jestliže SynJV(Otec(odebirany)) ≠ neexistuje pak
      jestliže SynJV(Otec(odebirany)) = odebirany pak
        SynJV(Otec(odebirany)) := neexistuje;
      jinak sourozenec := SynJV(Otec(odebirany));
    // nahrazování otce sourozencem, než získá sourozenec nového sourozence
    while(PocetPlatnychSouroz(sourozenec) = 0 & Otec(sourozenec ≠ neexistuje)
      pak Nahradit(Otec(sourozenec), sourozenec);
  pocet := pocet – 1;
return odebirany;
```

3.4.4. Hledání prvku

Vyhledávání prvku podle zadaných souřadnic opět začíná směrem od kořene stromu k listům. V každém navigačním vrcholu se zjistí, ve kterém z jeho kvadrantů hledaný prvek leží a podle toho se určí následující směr hledání, traverzování stromovou strukturou končí při dosažení listu. Následně se zkontroluje, zda se na této pozici nachází prvek s hledanými souřadnicemi. Pokud ano, je prvek vrácen, v opačném případě se hledaný prvek ve stromové struktuře nevyskytuje.

Hledání prvku se zeměpisnou šířkou = 50,04 a zeměpisnou délkou = 15,78 je znázorněno na obrázku 34.



Obrázek 34: Hledání prvku v quad stromu, zdroj [autor]

Pseudokód

```
Najdi(↓x, ↓y, ↑prvek)
  pomocny := koren;
  while(true)
    jestliže pomocny = neexistuje pak
      return neexistuje;
    // při dosažení listu (plnohodnotného prvku) dojde k porovnání a hledání končí
    jestliže Platny(pomocny) = true pak
      jestliže SouradniceX(pomocny) = x && SouradniceY(pomocny) = y pak
        return pomocny;
      jinak
          return neexistuje;
    // jinak se rozhodne, do kterého kvadrantu má hledání pokračovat
    jinak
      jestliže SouradniceX(pomocny) ≤ x && SouradniceY(pomocny) > y pak
        pomocny := SynSZ(pomocny);
      jinak jestliže SouradniceX(pomocny) ≤ x && SouradniceY(pomocny) ≤ y pak
        pomocny := SynSV(pomocny);
      jestliže SouradniceX(pomocny) > x && SouradniceY(pomocny) > y pak
        pomocny := SynJZ(pomocny);
      jinak
        pomocny := SynJV(pomocny);
```

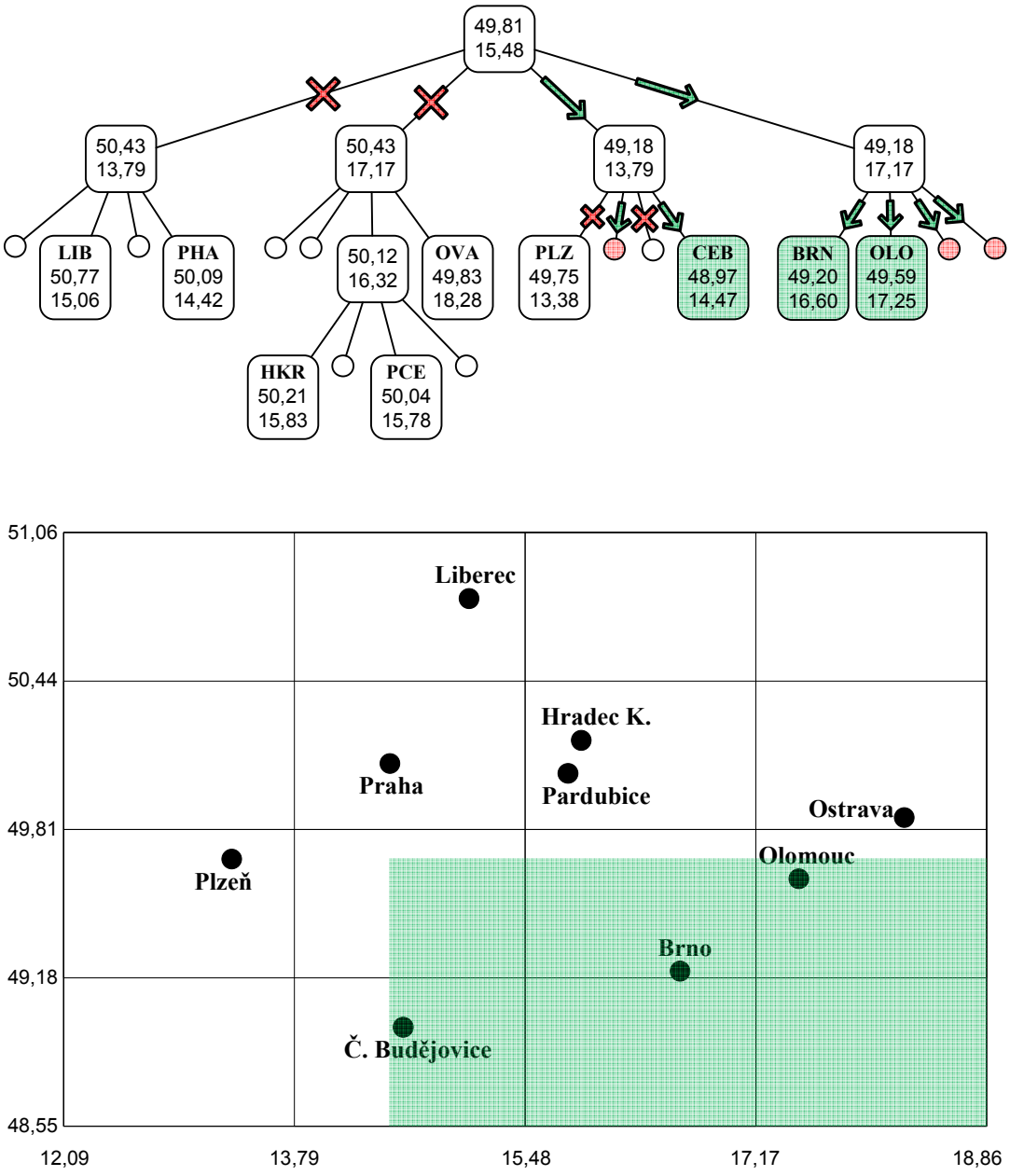
3.4.5. Intervalové hledání

Obdobně jako při hledání jednoho konkrétního prvku vychází se i při prohledávání obdélníkového segmentu z principu traverzování strukturou směrem od kořene stromu k listům. Při průchodu navigačním vrcholem se rozhoduje o tom, které jeho kvadranty (podstromy) má smysl dále prohledávat. Přijetí tohoto rozhodnutí vychází z následujících pravidel:

- severozápadní kvadrant se prohledává, pokud:
    - zeměpisná šířka aktuálního vrcholu není větší než maximální zeměpisná šířka v hledaném segmentu a současně zeměpisná délka aktuálního vrcholu je větší než minimální zeměpisná délka v hledaném segmentu,
- severovýchodní kvadrant se prohledává, pokud:
    - zeměpisná šířka aktuálního vrcholu není větší než maximální zeměpisná šířka v hledaném segmentu a současně zeměpisná délka aktuálního vrcholu není větší než maximální zeměpisná délka v hledaném segmentu,
- jihozápadní kvadrant se prohledává, pokud:
    - zeměpisná šířka aktuálního vrcholu je větší než minimální zeměpisná šířka v hledaném segmentu a současně zeměpisná délka aktuálního vrcholu je větší než minimální zeměpisná délka v hledaném segmentu,
- jihozápadní kvadrant se prohledává, pokud:
    - zeměpisná šířka aktuálního vrcholu je větší než minimální zeměpisná šířka v hledaném segmentu a současně zeměpisná délka aktuálního vrcholu není větší než maximální zeměpisná délka v hledaném segmentu.

Princip intervalového hledání ukazuje obrázek 35, na němž se hledají všechny prvky z obdélníkového segmentu, kde:

- zeměpisná šířka = 48,55 až 49,75,
- zeměpisná délka = 14,42 až 18,86.



Obrázek 35: Intervalové hledání v quad stromu, zdroj [autor]

Pseudokód

NajdiInterval(↓x1, ↓y1, ↓x2, ↓y2, ↓vrchol, ↓akce)

jestliže vrchol ≠ neexistuje pak

jestlize Platny(vrchol) = true pak

// při dosažení listu (plnohodnotného prvku) dojde k porovnání

jestliže x1 ≤ SouradniceX(vrchol) & SouradniceX(vrchol) ≤ x2

& y1 ≤ SouradniceY(vrchol) && SouradniceY(vrchol) ≤ y2 pak

akce(vrchol);

jinak

```
// jinak se rozhodne, kterými kvadranty má hledání pokračovat
jestliže SouradniceX(vrchol) ≤ x2  &&  SouradniceY(vrchol) > y1  pak
  NajdiInterval(x1, y1, x2, y2, SynSZ(vrchol), akce);
jestliže SouradniceX(vrchol) ≤ x2  &&  SouradniceY(vrchol) ≤ y2  pak
  NajdiInterval(x1, y1, x2, y2, SynSV(vrchol), akce);
jestliže SouradniceX(vrchol) > x1  &&  SouradniceY(vrchol) > y1  pak
  NajdiInterval(x1, y1, x2, y2, SynJZ(vrchol), akce);
jestliže SouradniceX(vrchol) > x1  &&  SouradniceY(vrchol) ≤ y2  pak
  NajdiInterval(x1, y1, x2, y2, SynJV(vrchol), akce);
```

3.5. Grid soubor

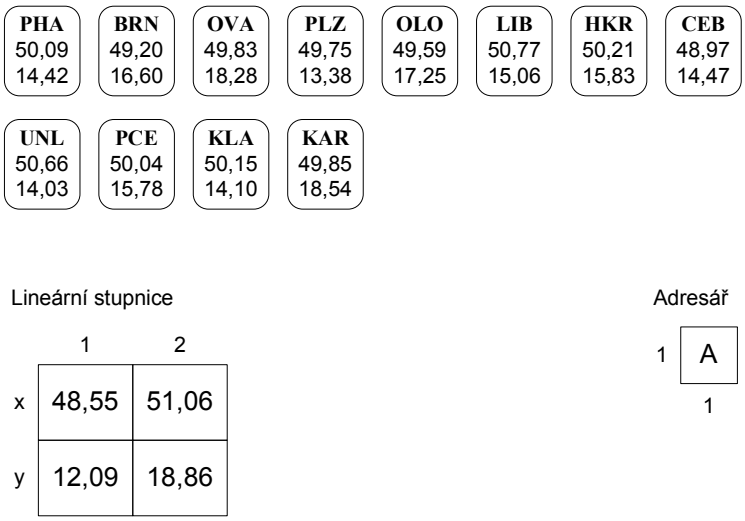
Datová struktura grid soubor se výrazně odlišuje od předchozích hierarchických struktur. Jedná se o dvouúrovňovou datovou strukturu, která se skládá z blokového souboru a řídicí struktury. Řídicí struktura slouží k řízení přístupu do blokového souboru a lze ji dále rozdělit na lineární stupnici a adresář. [1][12]

3.5.1. Vybudování struktury

Grid soubor sice nepatří mezi hierarchické struktury a není ho tedy třeba vyvažovat, přesto však není vhodné budovat strukturu postupným vkládáním prvků v náhodném pořadí, pokud jsou vstupní data známá předem. Mohla by totiž nastat situace, že například první prvek se vloží do bloku A, druhý prvek do bloku B, třetí prvek do bloku A, atd. V takovém případě by docházelo ke zbytečným blokovým přenosům, čímž by se budování struktury výrazně zdržovalo. Operace pro vybudování struktury proto zajišťuje, že není během budování nutné načíst do paměti žádný z bloků vícekrát. Princip spočívá v tom, že pokud převyšuje počet prvků kapacitu jednoho bloku, dojde k rozdělení prvků podle budoucí příslušnosti k jednotlivým blokům. Současně se rozdělí také lineární stupnice a adresář. Dělení se cyklicky opakuje až do okamžiku, kdy lze celou skupinu prvků po rozdělení umístit do jediného bloku, v té chvíli se vytvoří nový blok souboru a všechny prvky, které do něho náleží, se vloží najednou.

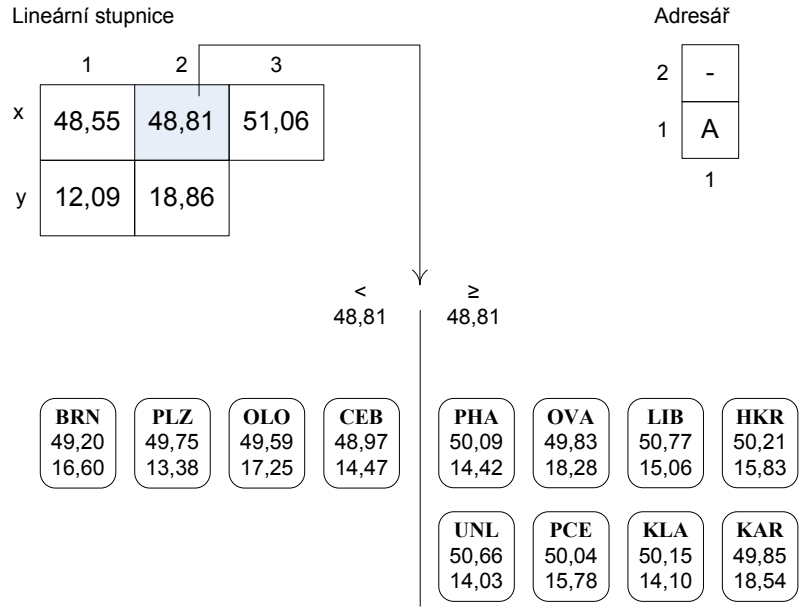
Výše popsaný postup je graficky vysvětlen na následujícím příkladě, pro ukázku byla maximální kapacita jednoho bloku souboru nastavena na tři záznamy (v praxi se však volí kapacita několikanásobně vyšší). Na obrázku 36 je uveden seznam prvků, z nichž má být datová struktura vybudována. Při inicializaci grid souboru se vytvoří také jeho první blok

blokově orientovaného souboru a odkaz na něj se vloží do adresáře, zároveň se v lineárních stupnicích nastaví mezní hodnoty pro dané území (v tomto příkladě se jedná o souřadnice ohraničující území České republiky).



Obrázek 36: Princip budování grid souboru (první krok), zdroj [autor]

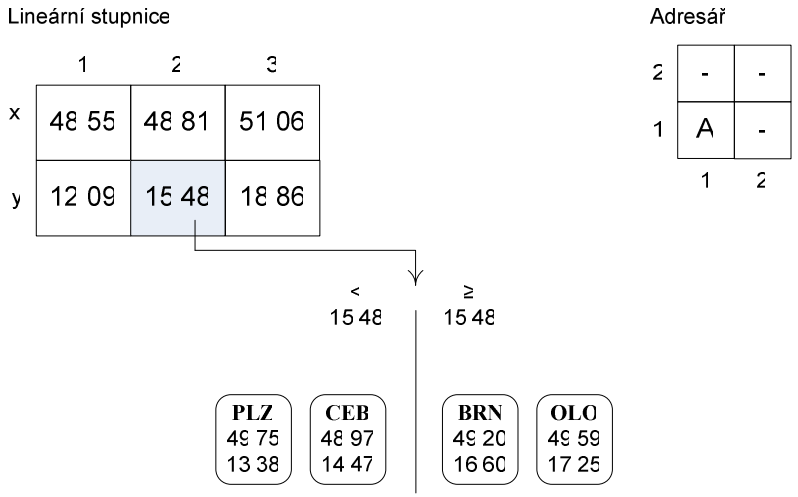
Protože počet prvků převyšuje kapacitu jednoho bloku, následuje rozdělení první lineární stupnice a také adresáře. Prvky se rozdělí do dvou skupin podle nově přidané hodnoty do lineární stupnice (obrázek 37).



Obrázek 37: Princip budování grid souboru (druhý krok), zdroj [autor]

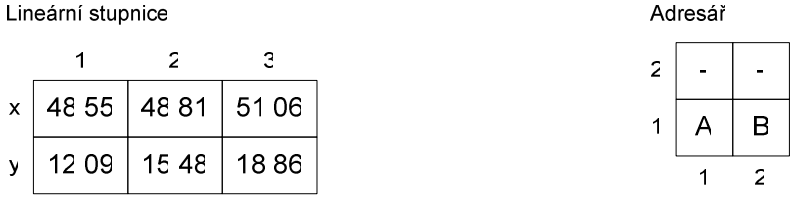
Vzhledem k tomu, že se i po rozdělení nachází v obou skupinách více prvků, než povoluje kapacita jednoho bloku souboru, cyklicky se v dělení pokračuje. Tentokrát se však dělení

realizuje podle druhé lineární stupnice. Pro prvky, které se po předchozím dělení nacházely vlevo od „čáry“, je tento postup graficky zachycen na obrázku 38.



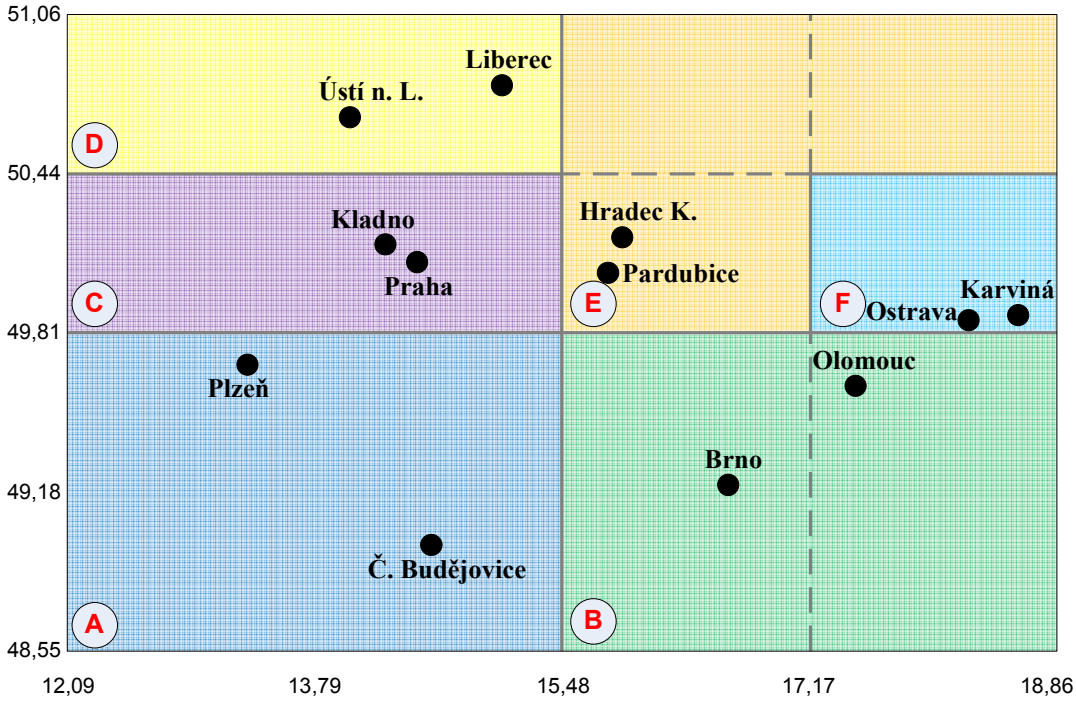
Obrázek 38: Princip budování grid souboru (třetí krok), zdroj [autor]

Nyní již počty prvků v obou skupinách po dělení vyhovují kapacitě bloku a mohou se tedy vložit do souboru. Prvky *Plzeň* a *České Budějovice* se umístí do předem připraveného bloku *A*. Pro prvky *Brno* a *Olomouc* se vytvoří nový blok *B*, následně se na příslušnou pozici v adresáři vloží jeho odkaz (obrázek 39).



Obrázek 39: Princip budování grid souboru (aktualizace adresáře), zdroj [autor]

Pro prvky, které se nacházely na obrázku 37 vpravo od „čáry“ a nebyly dosud do grid souboru vloženy, se aplikuje totožný postup. Výslednou podobu grid souboru po dokončení budování ukazuje obrázek 40.



Lineární stupnice

	1	2	3	4
x	48,55	48,81	50,44	51,06
y	12,09	15,48	17,17	18,86

Adresář

	1	2	3
3	D	E	F
2	C	E	F
1	A	B	B

Obrázek 40: Vybudovaný grid soubor, zdroj [autor]

Pseudokód

Kromě prvků pro vybudování struktury jsou pomocí parametrů *x1*, *y1*, *x2*, *y2* předány souřadnice ohraničující celé území (u výše uvedeného příkladu se jednalo o souřadnice vymezující území České republiky).

```
Vybuduj(↓prvky, ↓x1, ↓y1, ↓x2, ↓y1)
  jestliže prvky = neexistuje || Pocet(prvky) = 0 pak
    Konec;
  pocet := Pocet(prvky);
  stupniceX := VytvorNovouStupnici(x1, x2);
  stupniceY := VytvorNovouStupnici(y1, y2);
  adresar := VytvorNovyAdresar();
  soubor := VytvorNovySoubor();
  VybudujBlok(prvky, x1, y1, x2, y2, 1);
```

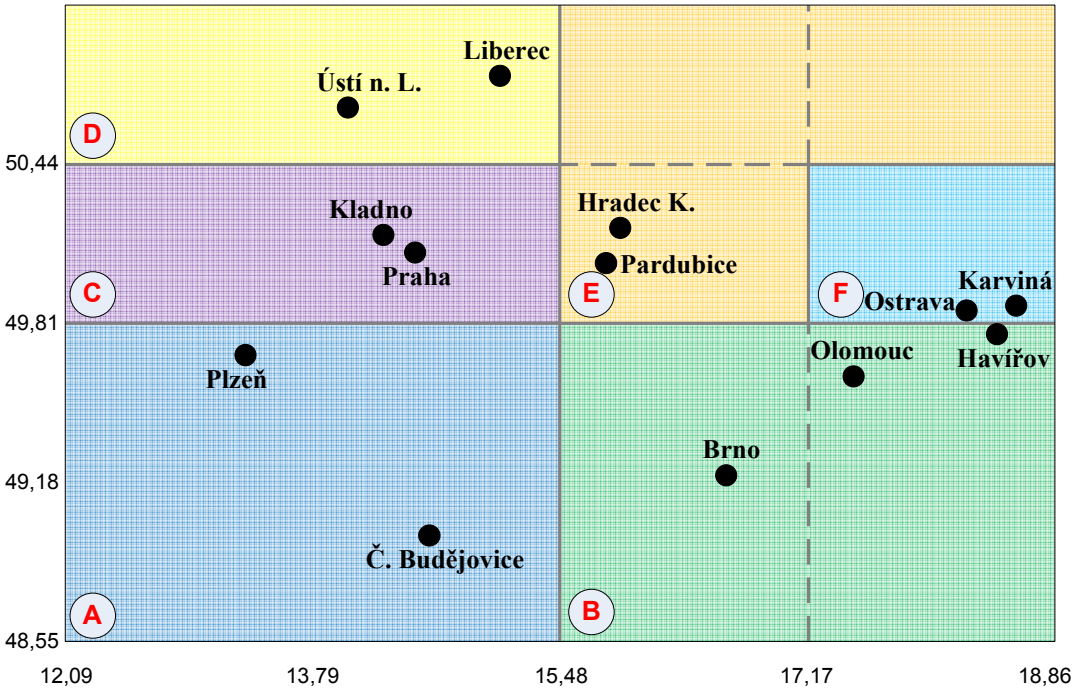
```
VybudujBlok(↓prvky, ↓x1, ↓y1, ↓x2, ↓y1, ↓cisloBloku)
  jestliže prvky = neexistuje || Pocet(prvky) = 0 pak
    Konec
  // zjištění indexů v lineárnách stupnicích, na nichž se nacházejí zadané hodnoty
  stupX1 := DejIndexHodnoty(stupniceX, x1);
  stupY1 := DejIndexHodnoty(stupniceY, y1);
  stupX2 := DejIndexHodnoty(stupniceX, x2);
  stupY2 := DejIndexHodnoty(stupniceY, y2);
  // v adresáři se podle předchozích indexů nastaví pro zadané území číslo bloku
  for i from stupX1 to stupX2 – 1 do
    for j from stupY1 to stupY2 – 1 do
      NastavCisloBloku(adresar, cisloBloku, i, j);
  // kontrola počtu prvků, pokud nepřevyšují kapacitu bloku, dojde k vložení
  jestliže Pocet(prvky) ≤ PocetZaznamuVBloku(soubor) pak
    jestliže (cisloBloku > 1) pak
      VytvorNovyBlok(soubor);
    for i  from 0 to Pocet(prvky) – 1 do
      VlozZaznam(soubor, cisloBloku, prvky[i]);
  // v případě, že počet prvků převyšuje kapacitu bloku, dojde k jejich rozdělení
  jinak
    // porovnání délky stupnic a rozšíření kratší z nich (při rovnosti stupniceX)
    jestliže DelkaStupnice(stupniceX) ≤ DelkaStupnice(stupniceY) pak
      // rozšíření stupniceX, rozšíření adresáře a následné rozdělení prvků
      RozsirStupnici(stupniceX, stupX2);
      RozsirAdresar(adresar, stupX2 – 1, stupY2 – 1, true);
      prvkyNadHranici := VytvorSeznamPrvku();
      // zjištění hranice, podle které se prvky rozdělí
      hranice := DejHodnotu(stupniceX, stupX2);
      RozdelPrvky(prvky, prvkyNadHranici, hranice, true);
      // prvky pod hranicí se vloží do původního bloku
      VybudujBlok(prvky, x1, y1, hranice, y2, cisloBloku);
      // pro prvky nad hranicí se vytvoří nový blok
      cisloNovehoBloku := MaximalniCisloBloku(adresar) + 1;
      VybudujBlok(prvkyNadHranici, hranice, y1, x2, y2, cisloNovehoBloku);
    jinak
      // rozšíření stupniceY, rozšíření adresáře a následné rozdělení prvků
      RozsirStupnici(stupniceY, stupY2);
      RozsirAdresar(adresar, stupX2 – 1, stupY2 – 1, false);
```

```
prvkyNadHranici := VytvorSeznamPrvku();
// zjištění hranice, podle které se prvky rozdělí
hranice := DejHodnotu(stupniceY, stupY2);
RozdelPrvky(prvky, prvkyNadHranici, hranice, false);
// prvky pod hranicí se vloží do původního bloku
VybudujBlok(prvky, x1, y1, y1, hranice, cisloBloku);
// pro prvky nad hranicí se vytvoří nový blok
cisloNovehoBloku := MaximalniCisloBloku(adresar) + 1;
VybudujBlok(prvkyNadHranici, x1, hranice, x2, y2, cisloNovehoBloku);
```

3.5.2. Vložení prvku

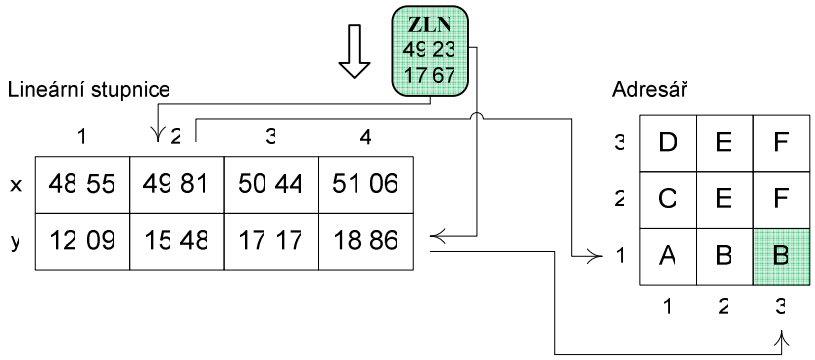
Operace pro vložení prvku do grid souboru vychází z jednoduchého principu. Pomocí lineárních stupnic a adresáře se zjistí, do kterého bloku vkládaný prvek patří. V případě, že není tento blok plně obsazen, dojde k vložení prvku. V opačném případě musí být nejprve příslušný blok rozdělen.

Obrázek 41 zachycuje stav struktury před vložením prvku *Zlín*.



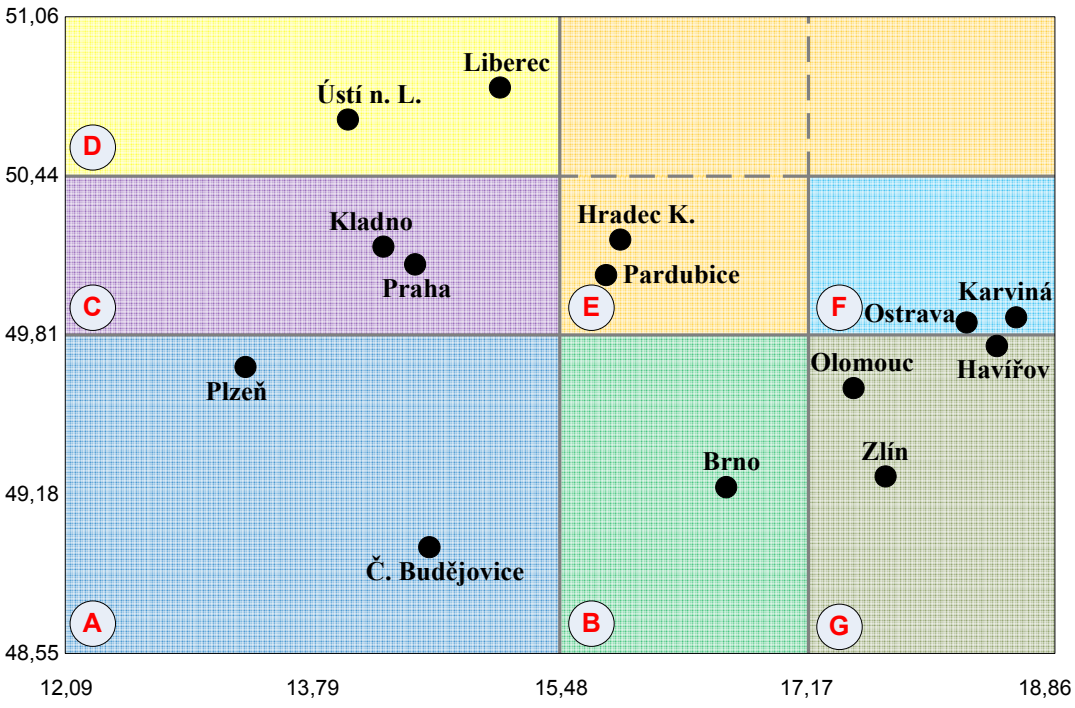
Obrázek 41: Vložení prvku do grid souboru (stav před vložením), zdroj [autor]

V prvním kroku je pomocí lineárních stupnic a adresáře zjištěno, že vkládaný prvek náleží do bloku *B* (obrázek 42).



Obrázek 42: Vložení prvku do grid souboru (nalezení bloku pro vložení)

Vzhledem k plné obsazenosti bloku *B* následuje jeho rozdělení a vytvoření nového bloku *G*. Protože se již blok *B* vyskytuje v adresáři na více pozicích, není nutné fyzicky dělit lineární stupnice a adresář, pouze se využije jejich předchozí rozdělení. Výsledný stav grid souboru po vložení prvku *Zlín* zobrazuje obrázek 43.



Lineární stupnice

	1	2	3	4
x	48,55	48,81	50,44	51,06
y	12,09	15,48	17,17	18,86

Adresář

3	D	E	F
2	C	E	F
1	A	B	G
	1	2	3

Obrázek 43: Vložení prvku do grid souboru (stav po vložení), zdroj [autor]

Pseudokód

Vloz(↓prvek)

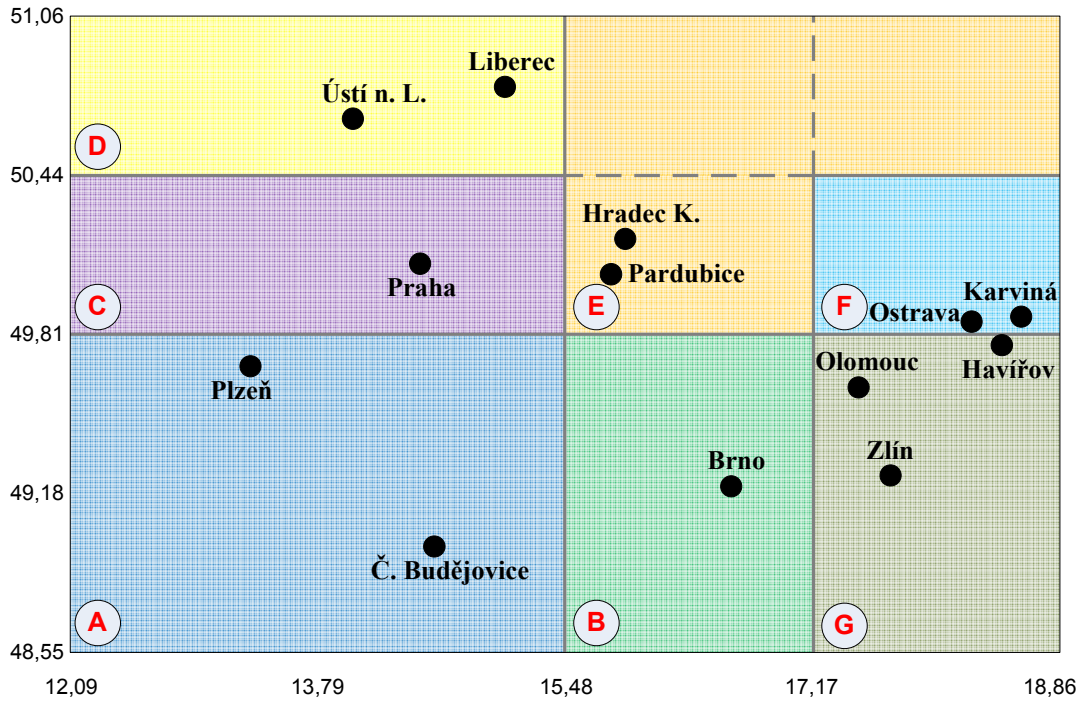
```
pocet := pocet + 1;
// zjištění indexů ve stupnicích, na nichž se nacházejí nejbližší vyšší hodnoty
stupX := DejIndexVyssiHodnoty(stupniceX, SouradniceX(prvek));
stupY := DejIndexVyssiHodnoty(stupniceY, SouradniceY(prvek));
cisloBloku := DejCisloBloku(adresar, stupX - 1, stupY - 1);
// je-li v daném bloku místo, dojde ke vložení prvku (jinak se vrátí hodnota false)
vlozeno := VlozZaznam(soubor, cisloBloku, prvek);
jestliže vlozeno = false pak
  // získání všech záznamů z daného bloku
  zaznamy := DejZaznamy(soubor, cisloBloku);
  // cyklické dělení, dokud všechny prvky patří do stejné oblasti (bloku)
  while(true)
    jestliže DelkaStupnice(stupniceX) ≤ DelkaStupnice(stupniceY) pak
      RozsirStupnici(stupniceX, stupX);
      RozsirAdresar(adresar, stupX – 1, stupY – 1, true);
      jestliže SouradniceX(prvek) ≥ DejHodnotu(stupniceX, stupX) pak
        // náleží-li vkládaný prvek do bloku nad dělením, zvýší se hodnota indexu
        stupX := stupX + 1;
    jinak
      RozsirStupnici(stupniceY, stupY);
      RozsirAdresar(adresar, stupX – 1, stupY – 1, false);
      jestliže SouradniceY(prvek) ≥ DejHodnotu(stupniceY, stupY) pak
        // náleží-li vkládaný prvek do bloku nad dělením, zvýší se hodnota indexu
        stupY := stupY + 1;
    // podle uložených indexů se zjistí, kolik záznamů bude patřit do nového bloku
    pocetOblast := PocetPrvkuVOblasti(zaznamy, stupX - 1, stupY - 1, stupX, stupY);
    jestliže pocetOblast < Pocet(zaznamy) pak
      // nebudou-li patřit všechny prvky do nového bloku, cyklus končí
      KonecCyklu;
  // vytvoření nového bloku
  novyBlok := VytvorNovyBlok(soubor);
  NastavCisloBloku(adresar, novyBlok, stupX - 1, stupY - 1);
  // projití všech původních záznamů a případné přesunutí do nového bloku
  x1 := DejHodnotu(stupniceX, stupX - 1);
  x2 := DejHodnotu(stupniceX, stupX);
```

```
y1 := DejHodnotu(stupniceY, stupY - 1);
y2 := DejHodnotu(stupniceY, stupY);
for i  from 0 to Pocet(zaznamy) – 1 do
  jestliže x1 < SouradniceX(zaznamy[i]) && SouradniceX(zaznamy[i]) ≤ x2
  & y1 < SouradniceY(zaznamy[i]) & SouradniceY(zaznamy[i]) ≤ y2 pak
    ZrusZaznam(soubor, cisloBloku, i);
    VlozZaznam(soubor, novyBlok, zaznamy[i]);
// nakonec dojde k umístění vkládaného prvku
VlozZaznam(soubor, novyBlok, prvek);
```

3.5.3. Odebrání prvku

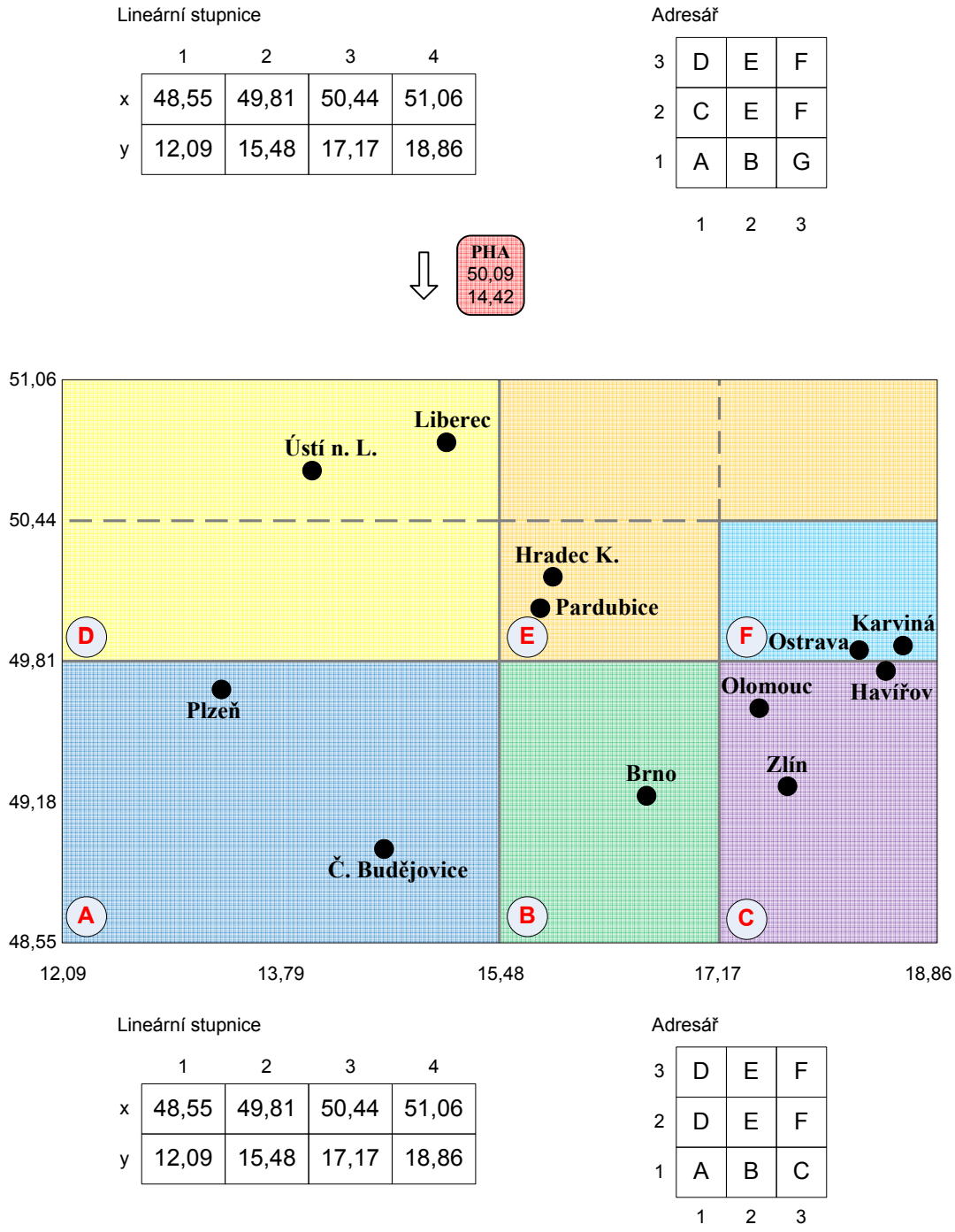
Při odebrání prvku z grid souboru mohou nastat dvě situace. Nachází-li se v bloku, z něhož se prvek odebírá alespoň jeden další prvek, provede se pouze odstranění odebíraného prvku. V opačném případě dojde navíc k připojení území, z něhož se prvek odebere, k bloku sousedního území. Do vyprázdněného bloku se poté přemístí všechny záznamy z posledního bloku, který je následně uvolněn a velikost celého souboru se tím sníží.

Výchozí stav grid souboru před odebráním prvku se souřadnicemi zeměpisné šířky = 50,09 a zeměpisné délky = 14,42 (*Praha*) definuje obrázek 44.



Obrázek 44: Odebrání prvku z grid souboru (stav před odebráním), zdroj [autor]

Protože se v bloku *C* nachází pouze odebíraný prvek, připojí se po jeho odstranění daná oblast k sousednímu bloku *D*. Následně se do vyprázdněného bloku *C* přesunou všechny prvky z posledního bloku *G*, který je uvolněn, čímž se celý soubor zmenší. Po provedení tohoto postupu odpovídá stav grid souboru situaci na obrázku 45.



Obrázek 45: Odebrání prvku z grid souboru (stav po odebrání), zdroj [autor]

Pseudokód

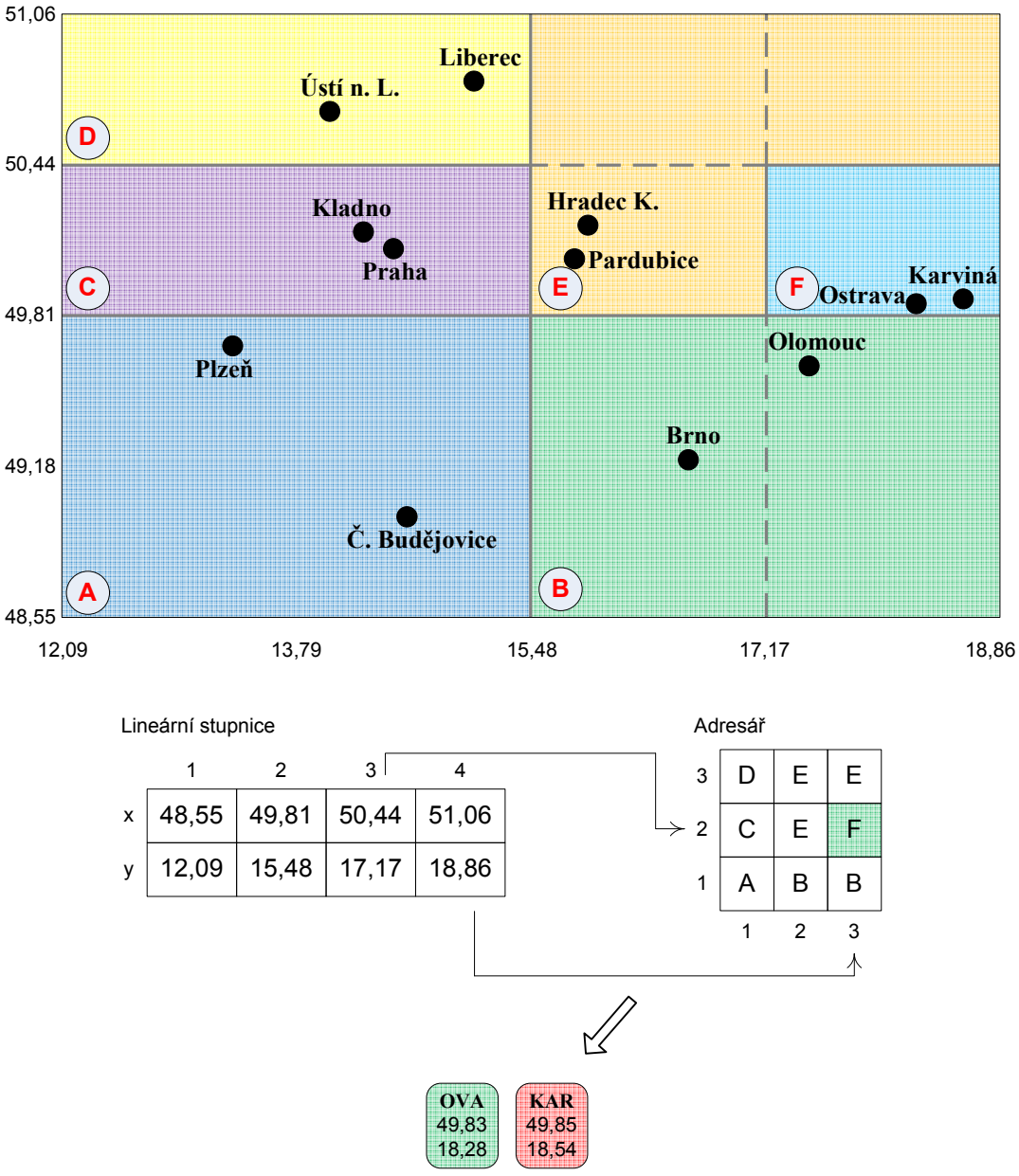
```
Odeber(↓x, ↓y, ↑prvek)
  // zjištění indexů ve stupnicích, na nichž se nacházejí nejbližší vyšší hodnoty
  stupX := DejIndexVyssiHodnoty(stupniceX, x);
  stupY := DejIndexVyssiHodnoty(stupniceY, y);
  cisloBloku := DejCisloBloku(adresar, stupX - 1, stupY - 1);
  // získání všech záznamů z daného bloku
  zaznamy := DejZaznamy(soubor, cisloBloku);
  odebirany := neexistuje;
  blokUvolnen := false;
  // projití záznamů a odebrání prvku s vyhovujícími souřadnicemi
  for i  from 0 to Pocet(zaznamy) – 1 do
    jestliže zaznamy[i] ≠ neexistuje & SouradniceX(zaznamy[i]) = x
    & SouradniceY(zaznamy[i]) = y pak
      pocet := pocet – 1;
      odebirany :=  zaznamy[i];
      // zrušení i-tého záznamu (byl-li záznam jediným v bloku, vrací operace true)
      blokUvolnen := ZrusZaznam(soubor, cisloBloku, i);
  // byl-li prvek v bloku jediným (došlo ke zmenšení souboru), zaktualizuje se adresář
  jestliže blokUvolnen = true pak
    // nalezení sousedního bloku, který si připojí území po vyprázdněném bloku
    sousedniBlok := DejCisloSouseda(cisloBloku);
    // nahrazení výskytů čísla vyprázdněného bloku číslem sousedního bloku v adresáři
    NahradCislaBloku(adresar, cisloBloku, sousedniBlok);
    // navíc nahrazení výskytů čísla posledního bloku číslem vyprázdněného bloku
    NahradCislaBloku(adresar, DejPocetBloku(soubor) + 1, cisloBloku);
  return odebirany;
```

3.5.4. Hledání prvku

Při hledání prvku podle zadaných souřadnic se nejprve pomocí lineárních stupnic a adresáře zjistí, ve kterém bloku se může hledaný prvek vyskytovat. Následně se porovnají souřadnice všech prvků v daném bloku se souřadnicemi hledanými, v případě úspěchu je vyhovující prvek vrácen. Nalezení bloku, který má být prohledán, se skládá ze dvou kroků. V prvním kroku se pro lineární stupnice zjistí indexy rovných nebo nejmenších vyšších hodnot, než jsou hodnoty hledaných souřadnic. Konkrétně pro první lineární stupnici se jedná o index shodné nebo nejmenší vyšší hodnoty oproti hodnotě

hledané zeměpisné šířky, pro druhou stupnici se jedná o index shodné nebo nejmenší vyšší hodnoty oproti hledané zeměpisné délce. Ve druhém kroku se tyto indexy sníží o jedničku, čímž se určí pozice adresáře, která odkazuje na hledaný blok.

Graficky tento princip znázorňuje obrázek 46, kde se vyhledává prvek se zeměpisnou šířkou = 49,83 a zeměpisnou délkou = 18,28. Po zjištění odpovídající bloku dojde ke kontrole všech jeho prvků a hledaným souřadnicím vyhovuje prvek *Ostrava*.



Obrázek 46: Hledání prvku v grid souboru, zdroj [autor]

Pseudokód

```
Najdi(↓x, ↓y, ↑prvek)
  // zjištění indexů ve stupnicích, na nichž se nacházejí nejbližší vyšší hodnoty
  stupX := DejIndexVyssiHodnoty(stupniceX, x);
  stupY := DejIndexVyssiHodnoty(stupniceY, y);
  cisloBloku := DejCisloBloku(adresar, stupX - 1, stupY - 1);
  // získání všech záznamů z daného bloku
  zaznamy := DejZaznamy(soubor, cisloBloku);
  // projití záznamů a případné vrácení vyhovujícího prvku
  for i  from 0 to Pocet(zaznamy) – 1 do
    jestliže zaznamy[i] ≠ neexistuje & SouradniceX(zaznamy[i]) = x
    & SouradniceY(zaznamy[i]) = y pak
      return zaznamy[i];
  // nebyl-li vyhovující prvek nalezen, struktura ho neobsahuje
  return neexistuje;
```

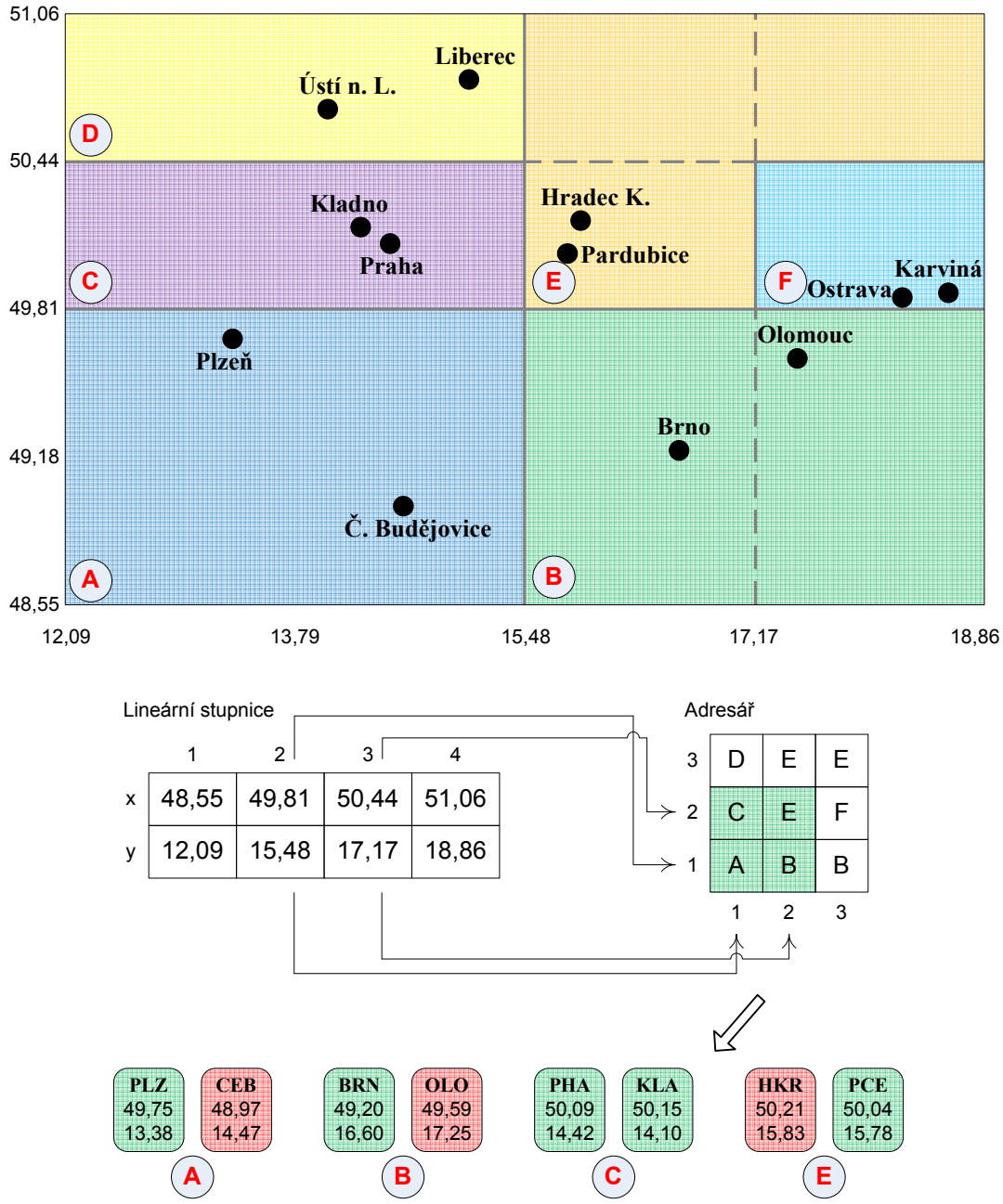
3.5.5. Intervalové hledání

Hledání všech prvků v grid souboru, které se nacházejí v obdélníkovém segmentu zadaném dvěma body, vychází ze stejného principu, jako hledání jednoho konkrétního prvku. Rozdíl spočívá pouze v tom, že pro obě lineární stupnice se nalezne dvojice indexů. Pomocí nich se následně v adresáři určí množina odkazů na bloky, jejichž prvky budou testovány na příslušnost k hledanému segmentu.

Postup pro intervalové hledání vysvětluje graficky obrázek 47, na němž se prohledává obdélníkový segment, kde:

- zeměpisná šířka = 49,20 až 50,20,
- zeměpisná délka = 12,09 až 16,60.

Pomocí lineárních stupnic a adresáře se zjistí, že prohledány mají být všechny prvky z bloků *A*, *B*, *C* a *E*. Porovnáním hledaných souřadnic se souřadnicemi všech prvků z uvedených bloků se zjistí, že hledanému segmentu vyhovují prvky *Plzeň*, *Brno*, *Praha*, *Kladno* a *Pardubice*.



Obrázek 47: Intervalové hledání v grid souboru, zdroj [autor]

Pseudokód

NajdiInterval(↓x1, ↓y1, ↓x2, ↓y2, ↓vrchol, ↓akce)

```
// zjištění indexů ve stupnicích, na nichž se nacházejí nejbližší vyšší hodnoty
stupX1 := DejIndexVyssiHodnoty(stupniceX, x1);
stupX2 := DejIndexVyssiHodnoty(stupniceX, x2);
stupY1 := DejIndexVyssiHodnoty(stupniceY, y1);
stupY2 := DejIndexVyssiHodnoty(stupniceY, y2);
// zjištění množiny bloků, které je nutné prohledat
```

```
for i  from stupX1 - 1 to stupX2 – 1 do
  for j  from stupY1 - 1 to stupY2 – 1 do
    cisloBloku := DejCisloBloku(adresar, i,  j);
    zaznamy := DejZaznamy(soubor, cisloBloku)
        // projití všech záznamů v bloku a případné vykonání akce
        for k from 0 to Pocet(zaznamy) – 1 do
          jestliže zaznamy[k] ≠ neexistuje & x1 ≤ SouradniceX(zaznamy[k])
          & SouradniceX(zaznamy[k]) ≤ x2 & y1 ≤ SouradniceY(zaznamy[k])
          & SouradniceY(zaznamy[k]) ≤ y2 pak
            akce(zaznamy[k]);
```

4. Podpůrná aplikace

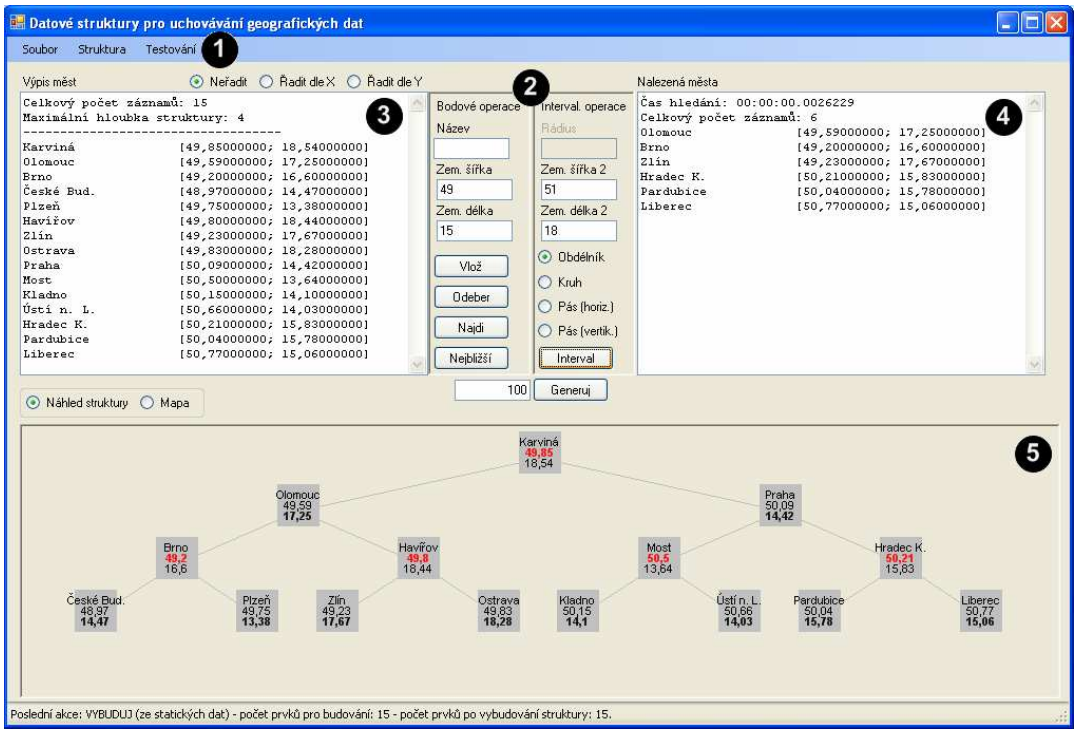
4.1. Uživatelská dokumentace

4.1.1. Základní popis

Podpůrná aplikace byla vytvořena v programovacím jazyce C# (platforma .NET) ve vývojovém prostředí Microsoft Visual Studio 2008. Jedná se o desktopovou aplikaci, pro jejíž spuštění je nutné mít v počítači nainstalovaný Microsoft .NET Framework verze 3.5.

4.1.2. Uživatelské prostředí

Obrázek 48 zobrazuje uživatelské prostředí aplikace. V levé horní části okna se nachází hlavní menu (1). Uprostřed horní poloviny formuláře jsou umístěny volby umožňující provádění základních operací nad datovými strukturami (2). Okno pro textový výpis všech prvků struktury leží na levé straně formuláře (3), podobné okno na pravé straně slouží pro výpis nalezených prvků (4). Zobrazení prvků na mapovém podkladu nebo vykreslení organizace datové struktury zajišťuje panel nacházející se ve spodní části okna (5).



Obrázek 48: Prostředí podpůrné aplikace, zdroj [autor]

4.1.3. Hlavní menu

Hlavní menu aplikace se nachází na horní liště (obrázek 49).

Obrázek 49: Hlavní menu aplikace, zdroj [autor]

Uživatel má k dispozici následující možnosti:

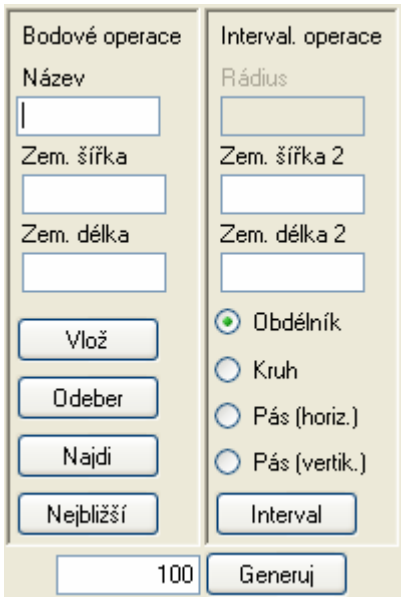
- menu *Soubor*:
    - volba *Načíst* umožňuje načtení dříve uloženého stavu datové struktury,
    - volba *Uložit* zajistí uložení aktuálního stavu datové struktury pro pozdější načtení,
    - volba *Načíst z TXT (staticky)* vybuduje vyváženou datovou strukturu z prvků, které zvolený soubor obsahuje,
    - volba *Načíst z TXT (dynamicky)* vybuduje datovou strukturu z prvků, které zvolený soubor obsahuje, pomocí opakovaného volání operace pro vložení prvku,
    - volba *Konec* zavře aplikaci,
- menu *Struktura*:
    - volby *2D strom*, *Range strom*, *Prioritní vyhledávací strom*, *Quad strom* a *Grid soubor* provedou změnu datové struktury,
    - volba *Zachovat data* určuje, zda se při použití předchozí volby vybuduje nová datová struktury ze stávajících prvků nebo zda zůstane prázdná,

- volba *Nastavení* zobrazí dialogové okno umožňující nastavit volitelné parametry aplikace (bližší popis v kapitole 4.1.5.),

- menu *Testování*:
  - volba *Testy* vyvolá dialogové okno, ve kterém lze testovat základní operace nad jednotlivými datovými strukturami (bližší popis v kapitole 4.1.6.),
  - volba *Případová studie* otevře dialogové okno týkající se zpracované případové studie (bližší popis v kapitole 4.1.7.).

4.1.4. Další volby

Uprostřed horní poloviny formuláře se nacházejí volby umožňující provádění základních operací nad datovými strukturami (obrázek 50).



Obrázek 50: Další volby aplikace, zdroj [autor]

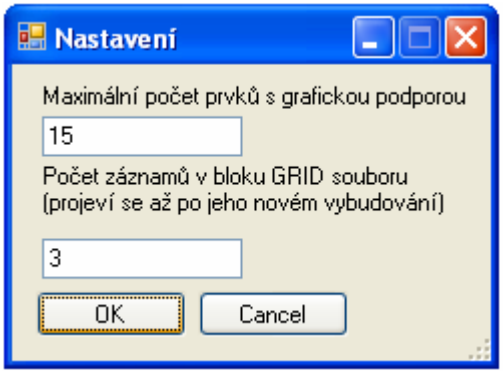
Ve spodní části je umístěno tlačítko *Generuj*, které náhodně vygeneruje zadaný počet prvků a vybuduje z nich aktuálně zvolenou datovou strukturu. Zbylé volby se dělí do dvou částí, vlevo leží tlačítka pro bodové operace, vpravo tlačítka pro intervalové operace. Následující přehled popisuje jejich funkci:

- tlačítko *Vlož* umístí do aktuální datové struktury nový prvek se zadaným názvem a souřadnicemi,

- tlačítko *Odeber* odstraní z aktuální datové struktury prvek se zadanými souřadnicemi,
- tlačítko *Najdi* nalezne v aktuální datové struktuře prvek se zadanými souřadnicemi,
- tlačítko *Nejbližší* nalezne v aktuální datové struktuře prvek, který se nachází nejblíže k zadaným souřadnicím,
- přepínače *Obdélník*, *Kruh*, *Pás (horiz.)*, *Pás (vertik.)* zajistí změnu používaného typu intervalového vyhledávání,
- tlačítko *Interval* vyhledává v aktuální datové struktuře všechny prvky, které odpovídají zadanému intervalu.

4.1.5. Dialogové okno Nastavení

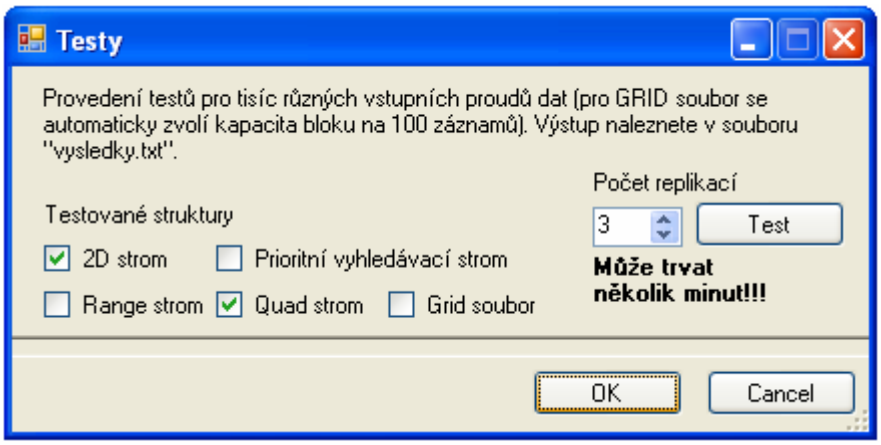
Podobu dialogového okna sloužícího ke změně volitelných parametrů aplikace ukazuje obrázek 51. Nastavit lze maximální počet prvků, pro které se datová struktura zobrazuje graficky, a také kapacitu jednoho bloku grid souboru, což se projeví až při jeho dalším vybudování. Po stisknutí tlačítka *OK* se provedené změny uloží.



Obrázek 51: Dialogové okno Nastavení, zdroj [autor]

4.1.6. Dialogové okno Test

Na obrázku 52 je zachyceno dialogové okno sloužící pro testování základních operací nad jednotlivými datovými strukturami. Před provedením testu lze zvolit datové struktury, které mají být do testu zahnuty, a také počet replikací. Kliknutím na tlačítko *Test* se spustí testování, které může v závislosti na zvolených datových strukturách a zvoleném počtu replikací trvat několik minut. Po dokončení se výsledky testu uloží do textového souboru *testy.txt* ve složce *Vystup*.



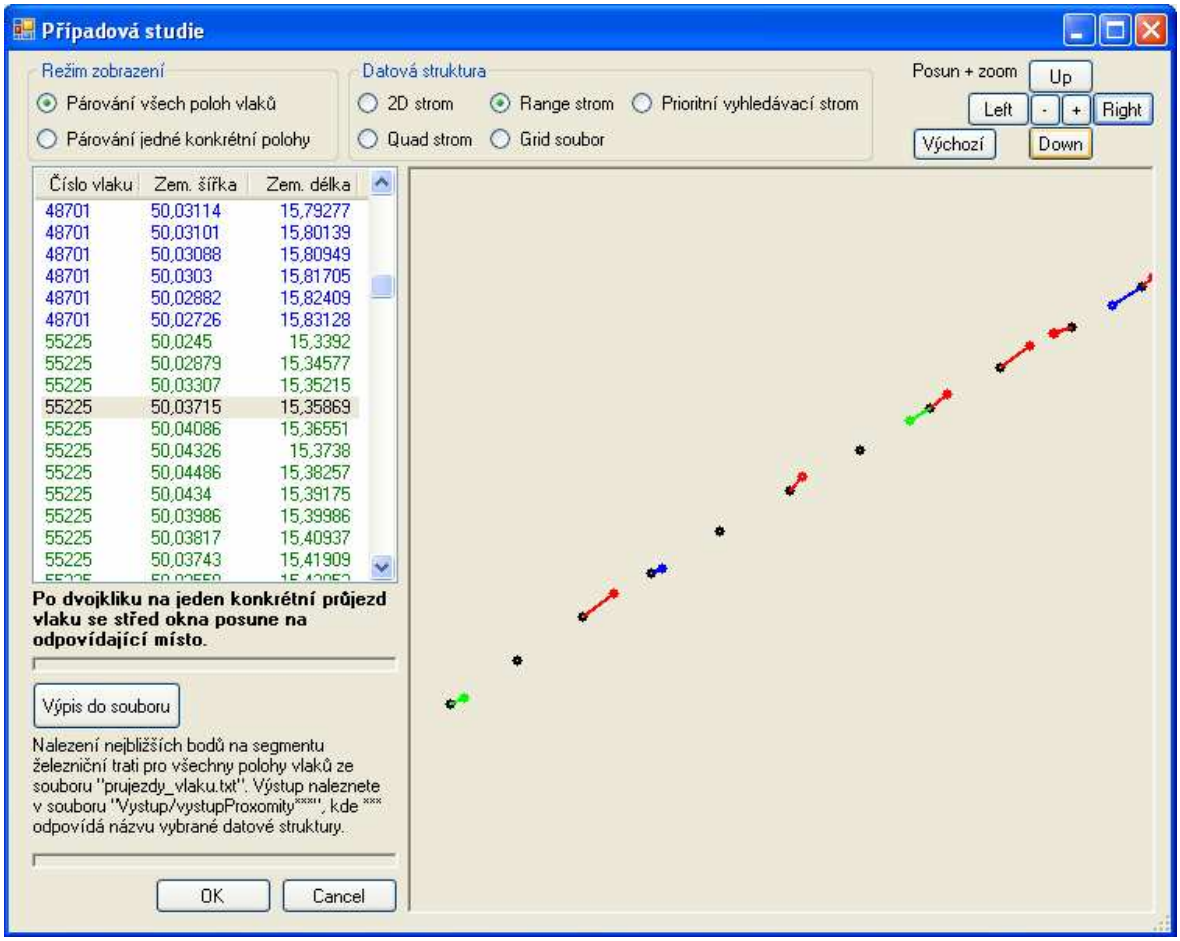
Obrázek 52: Dialogové okno Testy, zdroj [autor]

4.1.7. Dialogové okno Případová studie

Dialogové okno pro zpracovanou případovou studii slouží k hledání nejbližšího hektometrovníku v daném železničním segmentu pro zadané polohy odpovídající průjezdům vlaků. Bližší informace o případové studii uvádí kapitola 6. V levé horní části okna je umístěn přepínač módu zobrazení, který určuje, zda dojde k zobrazení nejbližšího hektometrovníku pro všechny polohy vlaků nebo jen pro jednu konkrétní polohu. V pravé horní části okna jsou umístěna tlačítka pro zoomování a posouvání se po mapě, uprostřed nahoře se pak nachází přepínače pro zvolení datové struktury, pomocí které se nejbližší hektometrovníky vyhledávají. Ihned po načtení formuláře dojde k vybudování zaškrtnuté struktury, do níž se vloží všechny hektometrovníky z daného úseku železniční trati. Současně se v levé části dialogového okna (v seznamu s průjezdy) vypíší všechny polohy vlaků, které jsou odlišeny barevně podle čísla vlaku. Výběr polohy z tohoto seznamu se realizuje dvojklikem myší nad vybranou polohou. Je-li aktivní mód pro vykreslování jediné polohy, zobrazí se tato poloha společně s nejbližším nalezeným hektometrovníkem včetně odpovídajících souřadnic. Je-li aktivní mód pro vykreslení všech poloh vlaků ležících ve zobrazovaném okně, souřadnice se kvůli přehlednosti nezobrazují. Jednotlivé polohy jsou odlišeny barevně i na mapovém podkladu, barvy samozřejmě odpovídají polohám v seznamu průjezdů.

Kromě grafického zpracování existuje také možnost exportu výsledků do textového souboru. K tomu slouží tlačítko *Výpis do souboru*, po jehož stisknutí se všechny polohy vlaků společně s nejbližšími nalezenými hektometrovníky uloží do textového souboru *vystupProxomity*.txt* (ve složce *Vystup*), kde * odpovídá názvu zvolené datové struktury.

Podobu dialogového okna pro případovou studii ukazuje obrázek 53.



Obrázek 53: Dialogové okno Případová studie, zdroj [autor]

4.2. Základní popis tříd

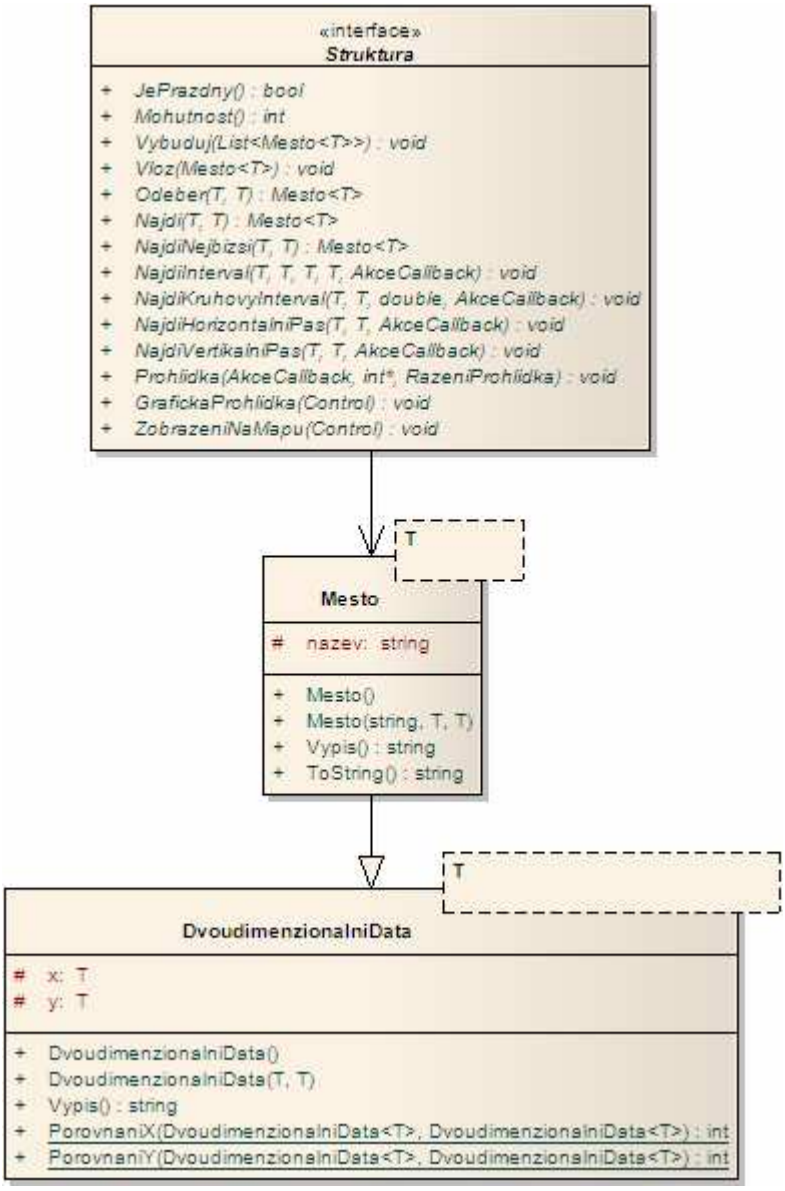
Tato kapitola stručně přibližuje implementaci jednotlivých datových struktur, přičemž obsahuje pouze základní popis tříd. Detailní popis s vysvětlením jednotlivých metod je uveden v programátorské dokumentaci (příloha B).

4.2.1. Třída pro dvoudimenzinální data a rozhraní datové struktury

Vzhledem k tomu, že všechny datové struktury uchovávají data se dvěma klíči, byla vytvořena třída *DvoudimenzionalniData* s atributy *x* a *y*. Tyto atributy představují klíče a mohou být libovolného porovnatelného datového typu (tzn. typ, který implementuje rozhraní *IComparable)*. Od třídy *DvoudimenzionalniData* je pomocí dědičnosti odvozena třída *Mesto*, která navíc definuje atribut *nazev*.

Z důvodu možnosti měnit za běhu typ datové struktury bylo navrženo rozhraní *Struktura*, které implementují všechny datové struktury uvedené v následujících kapitolách.

Graficky je návrh tříd *DvouDimenzionalniData* a *Mesto* a rozhraní *Struktura* zachycen na obrázku 54.



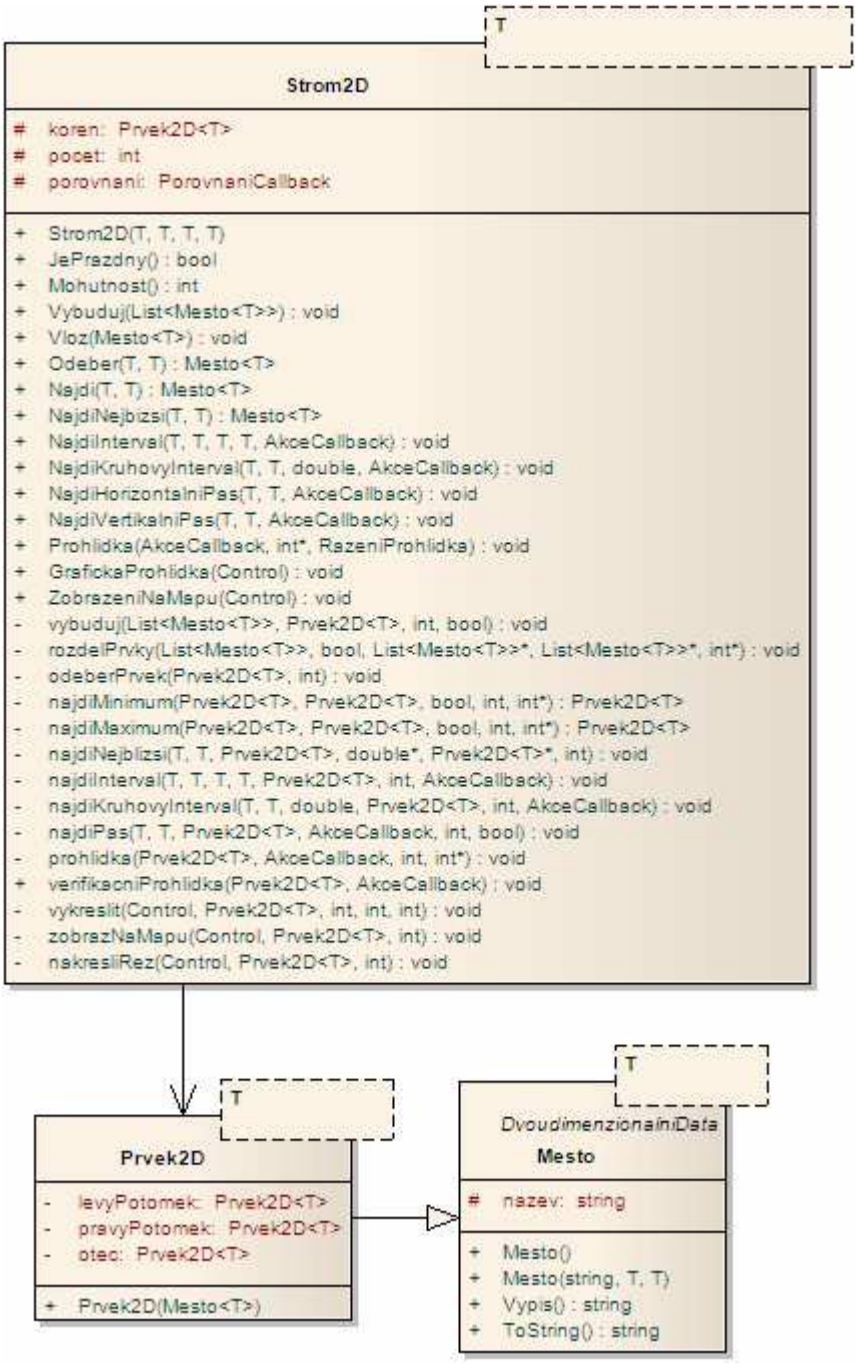
Obrázek 54: Třída pro dvoudimenzinální data a rozhraní datové struktury, zdroj [autor]

4.2.2. Třídy pro 2D strom

Pro datovou strukturu 2D strom byla navržena třída *Strom2D* a pro prvky, se kterými strom pracuje, třída *Prvek2D*. Pro třídu *Strom2D* byly definovány atributy *koren*, *pocet* a *porovnani*, což je delegát, který umožňuje porovnat souřadnice dvou prvků, přestože

datové typy souřadnic nejsou předem známé. Třídě *Prvek2D* byly přiděleny atributy *levyPotomek*, *pravyPotomek* a *otec*, což znamená, že každému vrcholu jsou přiděleny dva synové a otec. Jedná se tedy o filozofii vycházející z principů odvozených od binárního vyhledávacího stromu.

Návrh tříd pro 2D strom popisuje obrázek 55.



Obrázek 55: Třídy pro 2D strom, zdroj [autor]

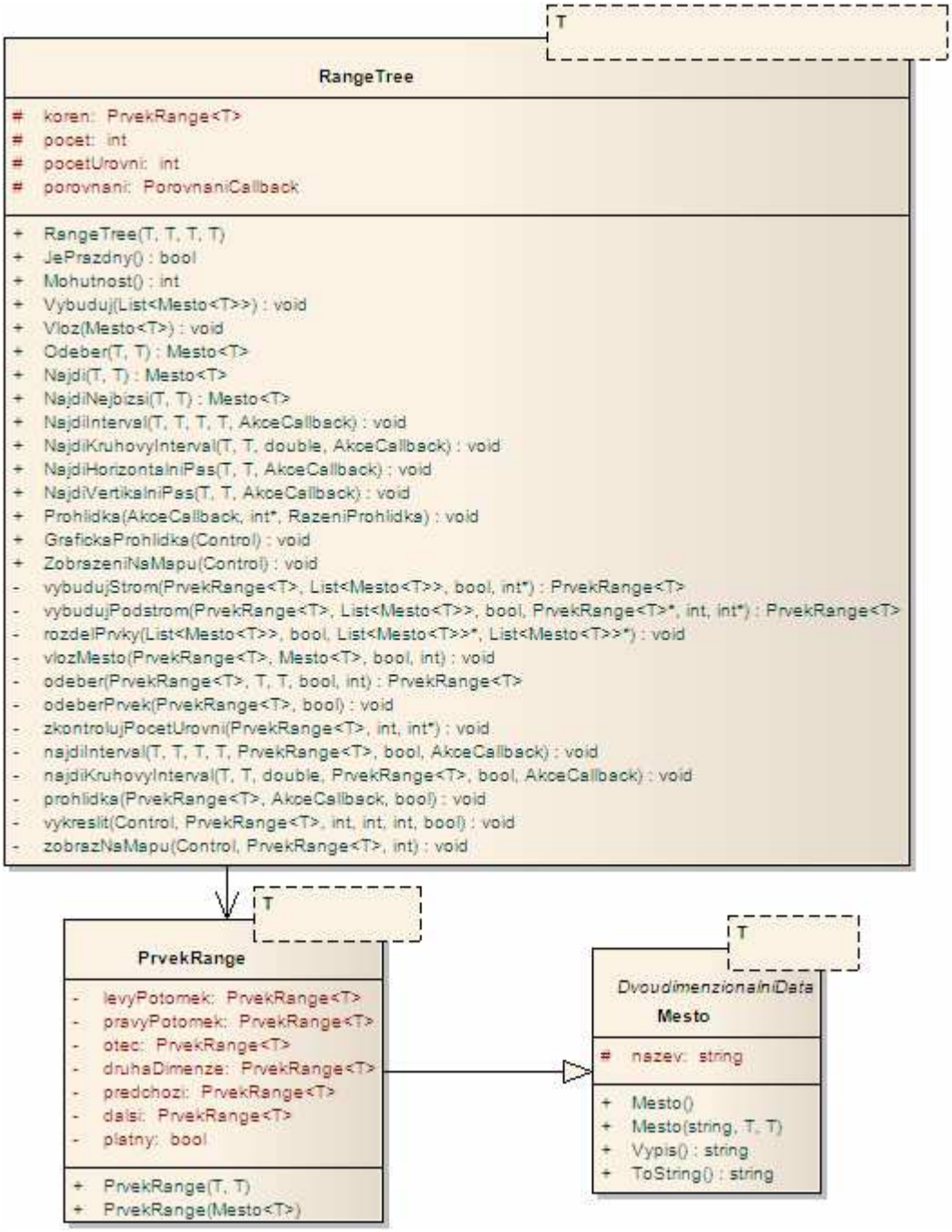
4.2.3. Třídy pro range strom

Pro datovou strukturu range strom byly implementovány třídy *RangeTree* a *PrvekRange*. Třída *RangeTree* obsahuje všechny atributy, které měla i třída *Strom2D*. Při provádění prohlídky se v každé struktuře zjišťuje a vypisuje také její maximální hloubka. Na rozdíl od všech ostatních stromových struktur se však při prohlídce range stromu neprohledává celý strom, ale po dosažení nejlevějšího plnohodnotného prvku se prochází pouze zřetězený seznam listů, což znemožňuje průběžné zjišťování hloubky stromu. Z toho důvodu byl třídě *RangeTree* přidán atribut *pocetUrovni*, ve kterém je maximální hloubka struktury trvale uložena. Třídě *PrvekRange* byly kromě atributů *levyPotomek*, *PravyPotomek* a *otec* přiděleny také atributy *predchozi* a *dalsi*, které slouží ke zřetězení prvků v seznamu listů, atribut *platny* slouží k odlišení plnohodnotných prvků od navigačních vrcholů, které navíc využívají i atribut *druhaDimenze*, jenž slouží k uložení odkazu na strom opačné dimenze.

Navíc se navigační vrcholy vyznačují tím, že neuchovávají klíčové souřadnice *x* a *y*, ale rozsah intervalu v rámci jedné ze souřadnic, všechny prvky v jejich podstromu poté spadají do tohoto intervalu. Pro hodnoty v atributech *x* a *y* u navigačních vrcholů tedy platí:

- leží-li navigační vrchol ve stromu první dimenze (dimenze *x*):
    - hodnota atributu *x* se rovná hodnotě *xmin* (minimální hodnota *x*-ové souřadnice v podstromu),
    - hodnota atributu y se rovná hodnotě x*max* (maximální hodnota *x*-ové souřadnice v podstromu),
- leží-li navigační vrchol ve stromu druhé dimenze (dimenze *y*):
    - hodnota atributu *x* se rovná hodnotě *ymin* (minimální hodnota *y*-ové souřadnice v podstromu),
    - hodnota atributu *y* se rovná hodnotě *ymax* (maximální hodnota *y*-ové souřadnice v podstromu).

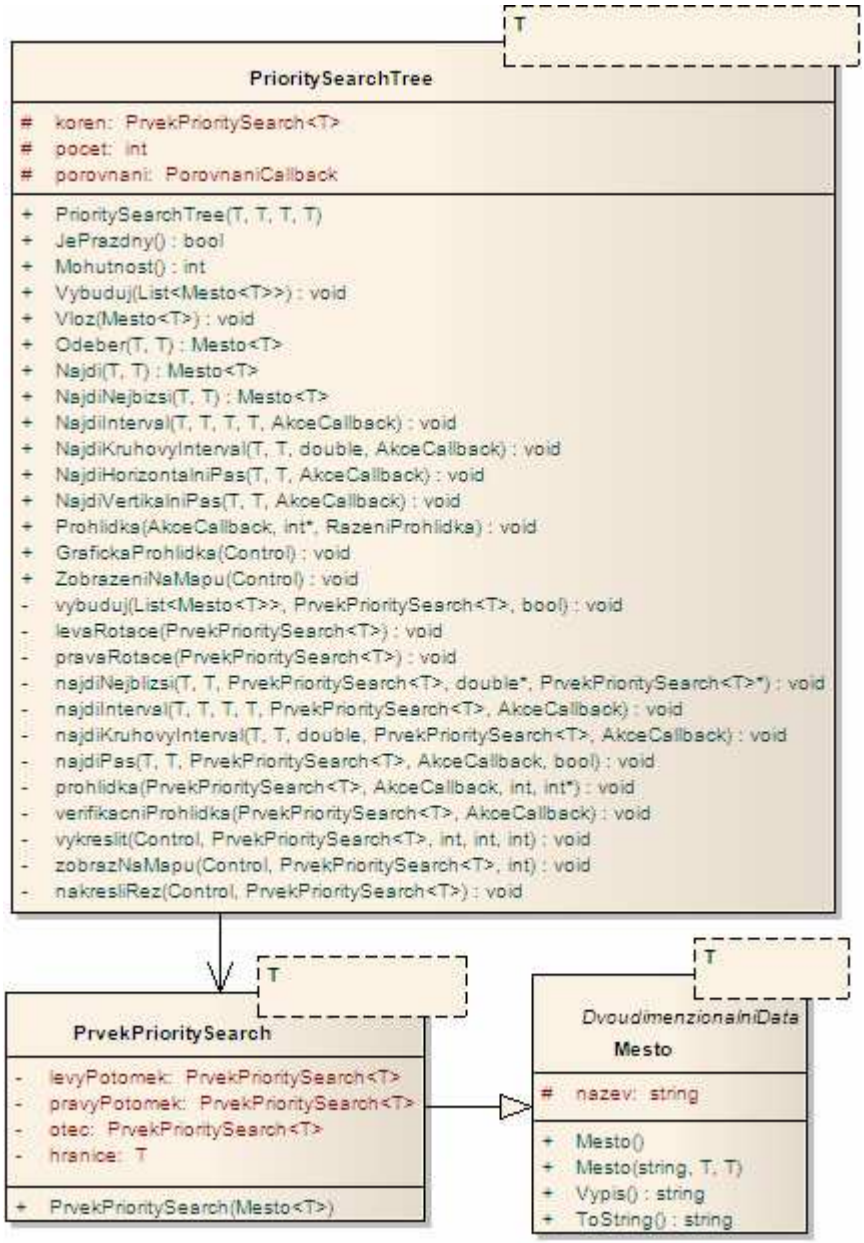
Návrh tříd pro prioritní vyhledávací strom je zobrazen na obrázku 56.



Obrázek 56: Třídy pro range strom, zdroj [autor]

4.2.4. Třídy pro prioritní vyhledávací strom

Třídy *PrioritySearchTree* a *PrvekPrioritySearch* představující prioritní vyhledávací strom zachycuje obrázek 57. Jedná se o téměř stejný návrh jako v případě 2D stromu, jediný rozdíl spočívá v přidaném atributu *hranice* u třídy *PrvekPrioritySearch*. Tento atribut umožňuje vybudovat vyváženou stromovou strukturu.

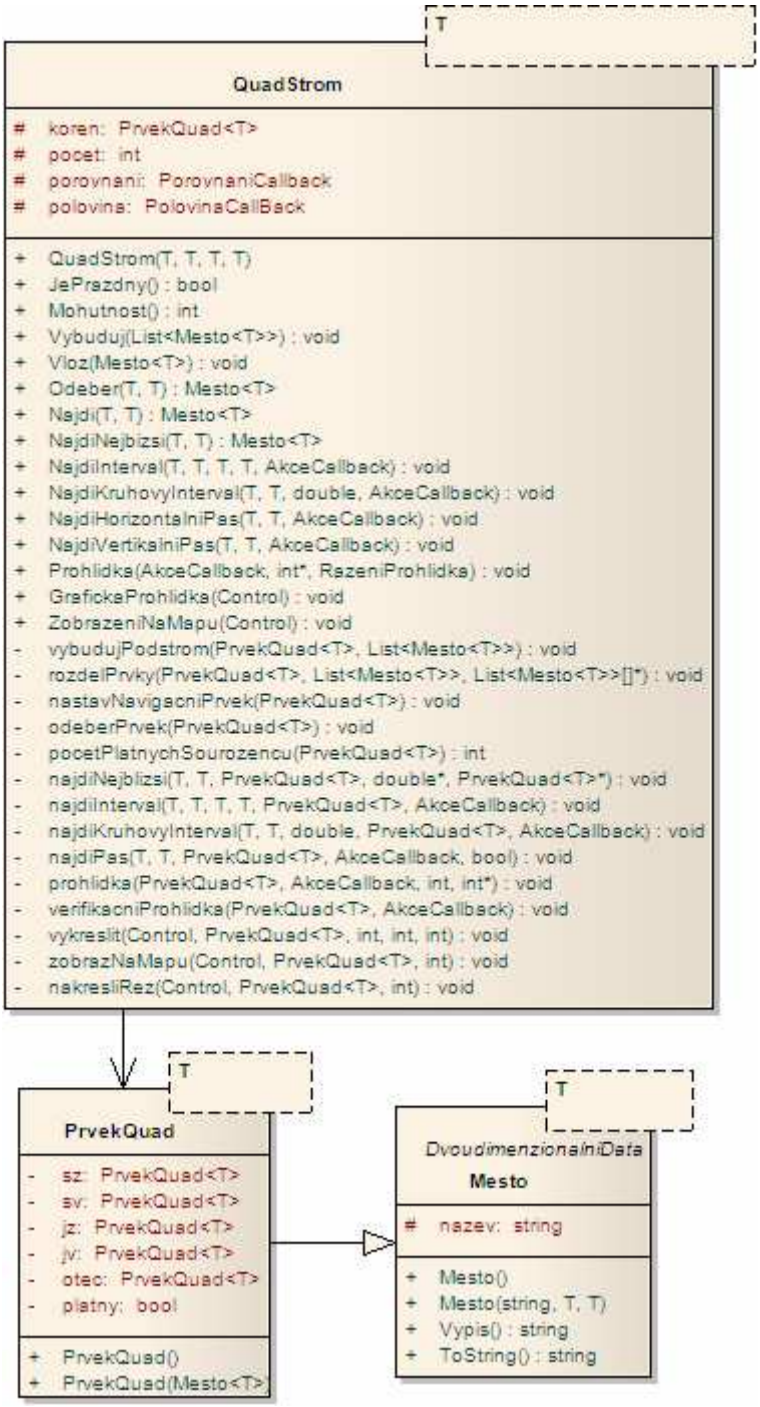


Obrázek 57: Třídy pro prioritní vyhledávací strom, zdroj [autor]

4.2.5. Třídy pro quad strom

Datová struktura quad strom se od předchozích struktur odlišuje tím, že každý vrchol má přiřazeny čtyři syny. Kromě atributů pro tyto čtyři syny (*sz*, *sv*, *jz*, *jv*) byly třídě *PrvekQuad* přiděleny také atributy *otec* a *platny*, jenž odlišuje plnohodnotné vrcholy od navigačních. Třída *QuadStrom* obsahuje atributy *koren*, *pocet*, *porovnani* a *polovina*, což je delegát umožňující dělit území na poloviny, i když datové typy souřadnic nejsou předem známy.

Graficky popisuje třídy pro quad strom obrázek 58.

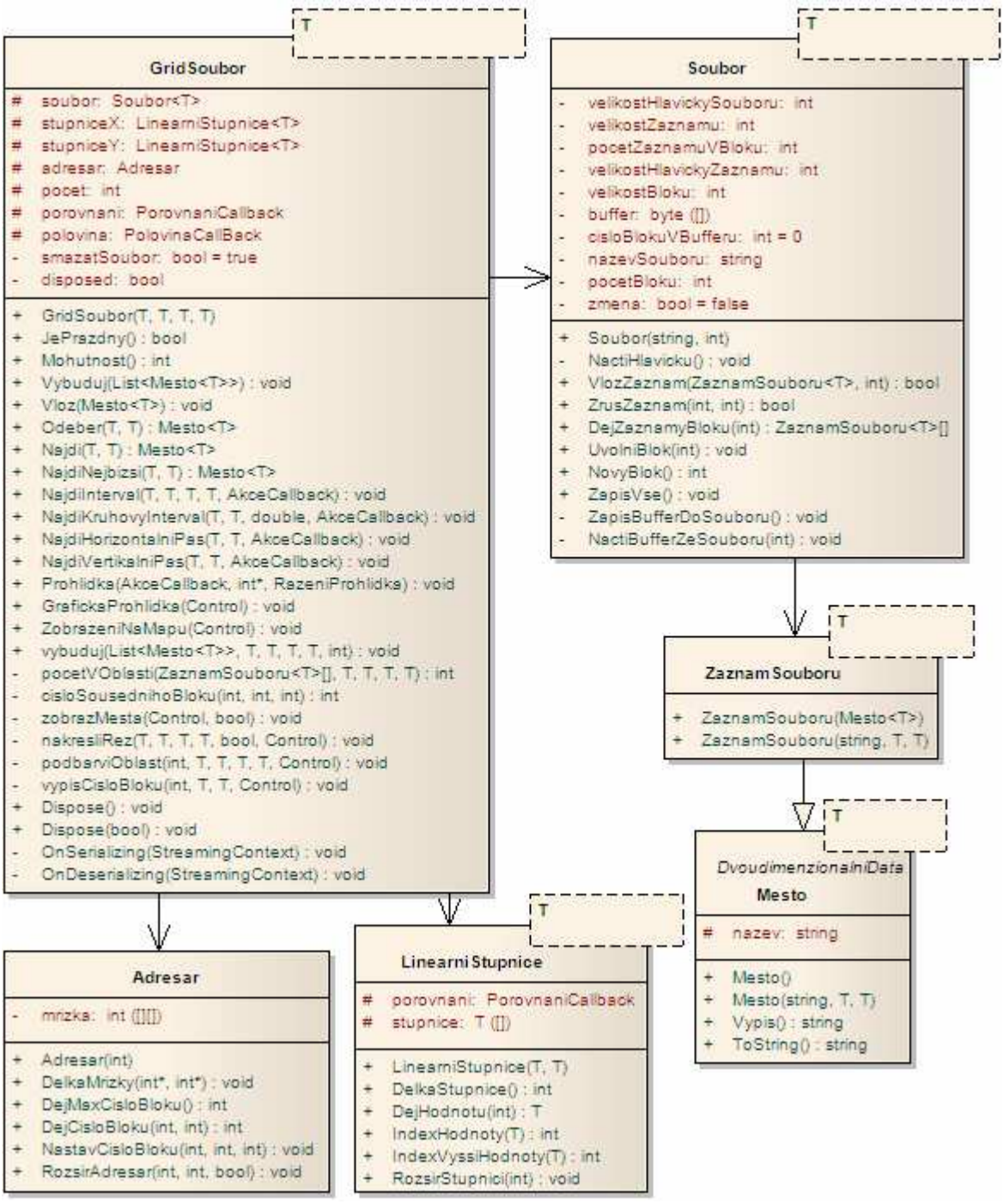


Obrázek 58: Třídy pro quad strom, zdroj [autor]

4.2.6. Třídy pro grid soubor

Datová struktura grid soubor se od předchozích stromových struktur výrazně liší, čemuž odpovídá i návrh jednotlivých tříd (obrázek 59). Třída *LinearniStupnice* definuje atribut *stupnice*, který odpovídá jednorozměrnému poli s hodnotami libovolného datového typu, jenž implementuje rozhraní *IComparable*. Pro třídu *Adresar* byl definován atribut *mrizka*,

který odpovídá dvourozměrnému poli celočíselných hodnot, které odpovídají číslům bloků v souboru. Hlavní třída *GridSoubor* obsahuje mimo jiné atributy *stupniceX*, *stupniceY*, *adresar* a *soubor*, první tři zmíněné atributy slouží jako přístupová struktura do souboru. Třída *Soubor* slouží pro manipulaci s blokově orientovaným souborem, v němž jsou uloženy záznamy odpovídající třídě *ZaznamSouboru*. Dále má třída *Soubor* přidělen atribut *buffer*, pomocí něhož se do operační paměti načte vždy jeden blok souboru.



Obrázek 59: Třídy pro grid soubor, zdroj [autor]

5. Porovnání datových struktur

Testování se uskutečnilo pro všechny vybrané datové struktury, tedy pro 2D strom, range strom, prioritní vyhledávací strom, quad strom a grid soubor, u kterého byla zvolena kapacita jednoho bloku na 2000 záznamů. Pro předem známá statická data i pro dynamické proudy dat se porovnávaly následující operace:

- vybudování struktury,
- vložení prvku do struktury,
- odebrání prvku ze struktury,
- hledání prvku se zadanými souřadnicemi,
- hledání nebližšího prvku k zadaným souřadnicím,
- intervalové hledání v obdélníkovém segmentu,
- intervalové hledání v kruhovém segmentu,
- intervalové hledání v tzv. pruhovém segmentu.

Testy byly provedeny na množině reálných dat odpovídajících obcím České republiky, k dispozici bylo celkem 16119 záznamů. Testování každé operace se na této množině dat zopakovalo celkem tisíckrát, přičemž pořadí jednotlivých záznamů bylo pro každé opakování zvoleno náhodně. Celková doba pro každou operaci byla změřena pomocí třídy *StopWatch* a její operace *ElapsedTicks*, která slouží pro velmi přesné měření a počítá počet tzv. tiků, jež během měření uběhly. [9]

Pro vyšší přehlednost byla vždy pro danou operaci označena nejvyšší naměřená hodnota za časový úsek odpovídající 100%. Hodnoty pro tuto operaci u ostatních datových struktur byly podle poměru naměřených hodnot také převedeny na procenta.

5.1. Testy a jejich výsledky

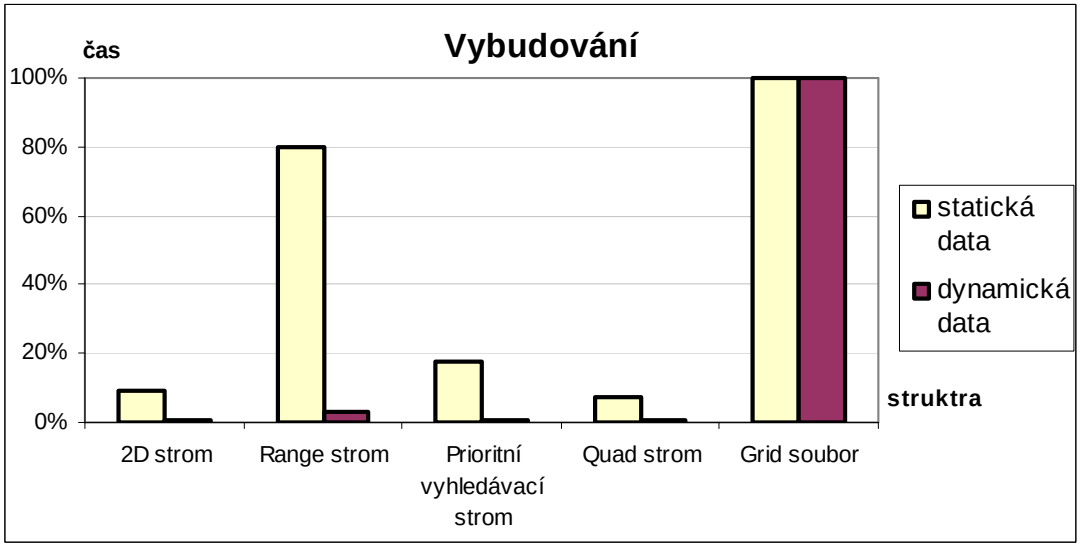
Výsledky všech měření ukazuje tabulka 1, pro každou operaci je navíc zvýrazněna nejnižší hodnota.

		2D strom	Range strom	Prioritní vyhledávací strom	Quad strom	Grid soubor
Předem známá data	**Vybudování**	8,98%	80,02%	17,70%	**7,05%**	100%
	Vložení	**1,29%**	22,07%	4,01%	2,95%	100%
	Odebrání	2,82%	18,34%	4,43%	**2,22%**	100%
	Hledání prvku	**1,19%**	2,85%	1,38%	1,60%	100%
	Hledání nejbližšího prvku	3,41%	4,05%	1,72%	**1,43%**	100%
	Intervalové hledání (obdélník)	**19,57%**	35,36%	20,33%	25,30%	100%
	Intervalové hledání (kruh)	**9,06%**	23,42%	12,95%	17,71%	100%
	Intervalové hledání (pás)	5,24%	**4,59%**	6,15%	5,90%	100%
Dynamický proud dat	**Vybudování**	**0,04%**	2,84%	0,16%	0,10%	100%
	Vložení	**1,38%**	24,84%	3,89%	2,85%	100%
	Odebrání	2,86%	28,88%	4,28%	**2,18%**	100%
	Hledání prvku	**1,26%**	3,52%	1,94%	1,69%	100%
	Hledání nejbližšího prvku	4,82%	5,03%	2,69%	**1,44%**	100%
	Intervalové hledání (obdélník)	**21,02%**	39,42%	23,75%	24,77%	100%
	Intervalové hledání (kruh)	**12,36%**	25,02%	15,40%	18,30%	100%
	Intervalové hledání (pás)	6,10%	**4,91%**	5,69%	5,84%	100%

Tabulka 1: Výsledky testů, zdroj [autor]

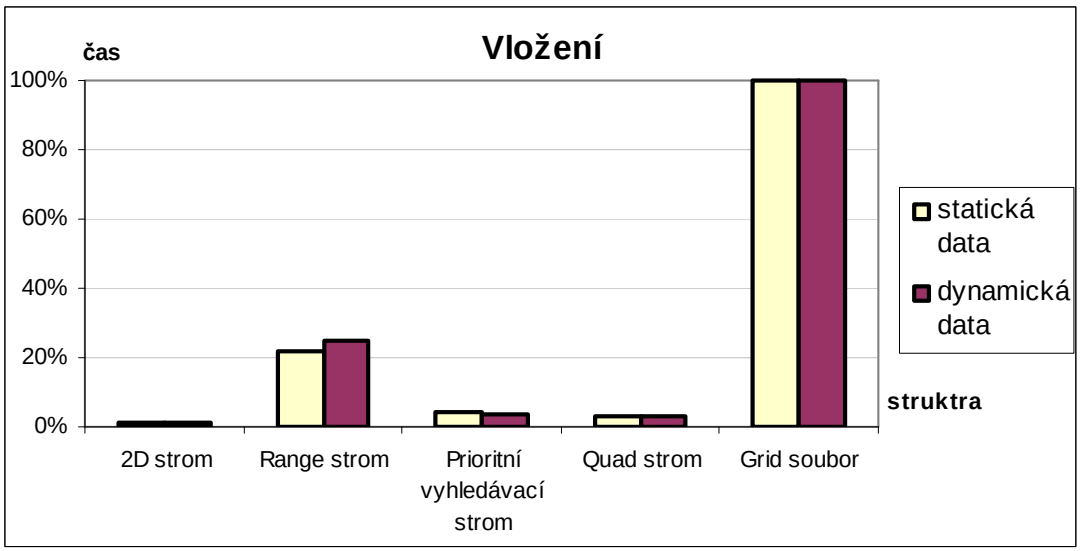
5.2. Grafické vyhodnocení testů

Pro lepší představu byly naměřené hodnoty z předchozí tabulky zpracovány také do grafů (obrázek 60 až obrázek 67). Výsledky jednotlivých operací pro předem známá data i pro dynamické proudy dat byly vloženy do jednoho společného grafu.



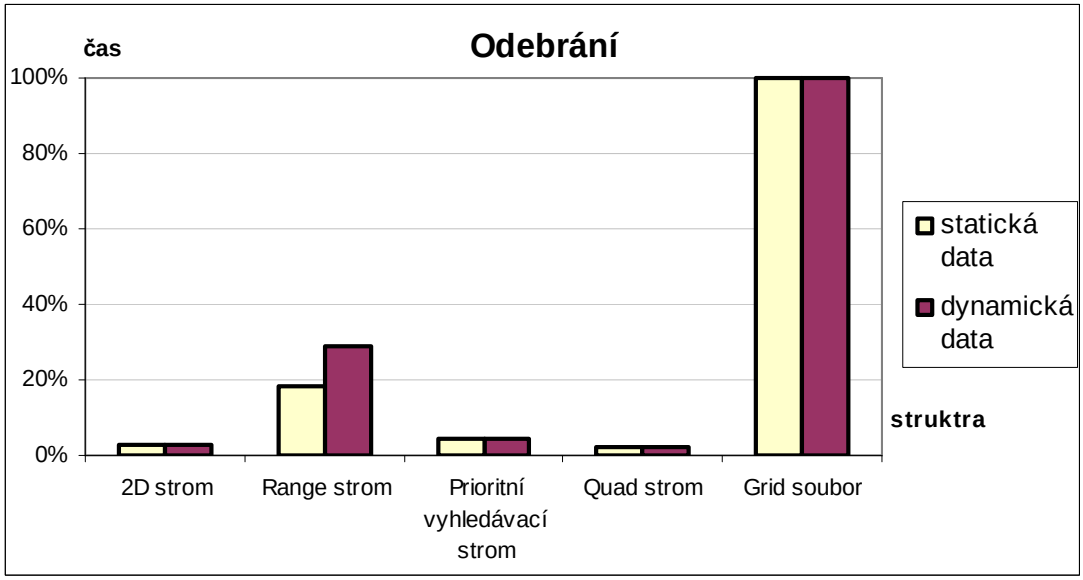
Obrázek 60: Výsledky testu (vybudování struktury), zdroj [autor]

Z obrázku je patrné, že nejméně časové náročné bylo vybudování quad stromu pro statická data a 2D stromu pro dynamická data. Naopak nejvíce času zabralo budování grid souboru, u dynamických dat se navíc projevilo zdržování způsobené opakovaným načítáním bloků do paměti, což dokazuje výrazný nepoměr k časům naměřeným pro ostatní struktury.



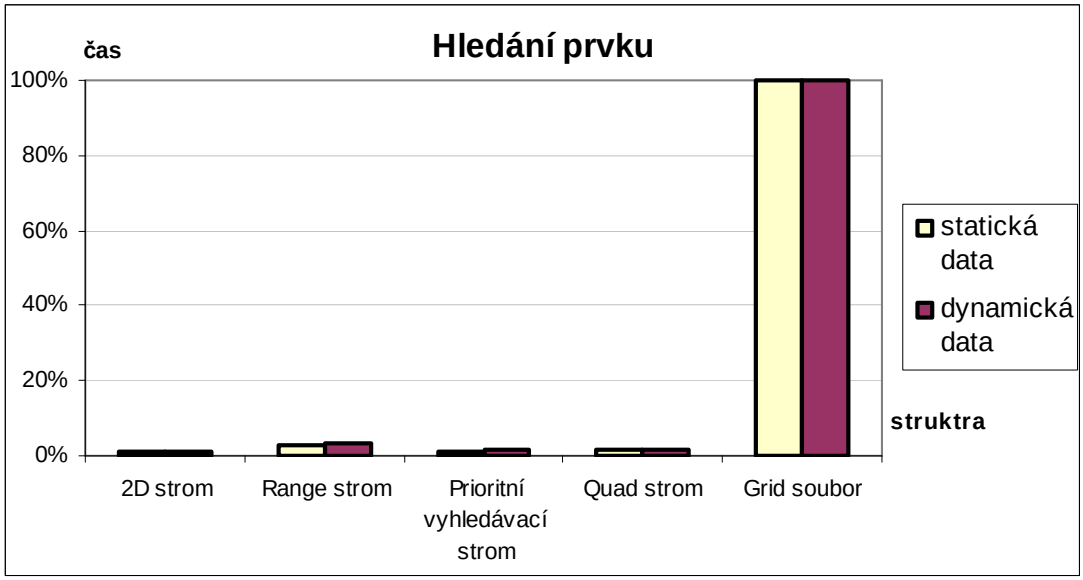
Obrázek 61: Výsledky testu (vložení prvku do struktury), zdroj [autor]

Měření času při vkládání prvku do struktury dopadlo nejpříznivěji u 2D stromu, nejhůře dopadl naopak grid soubor. Mezi čtyřmi stromovými strukturami vyčnívá vyšší časovou náročností range strom, což lze přisuzovat skutečnosti, že se prvek nevkládá pouze do hlavního stromu, ale také do stromů druhých dimenzí.



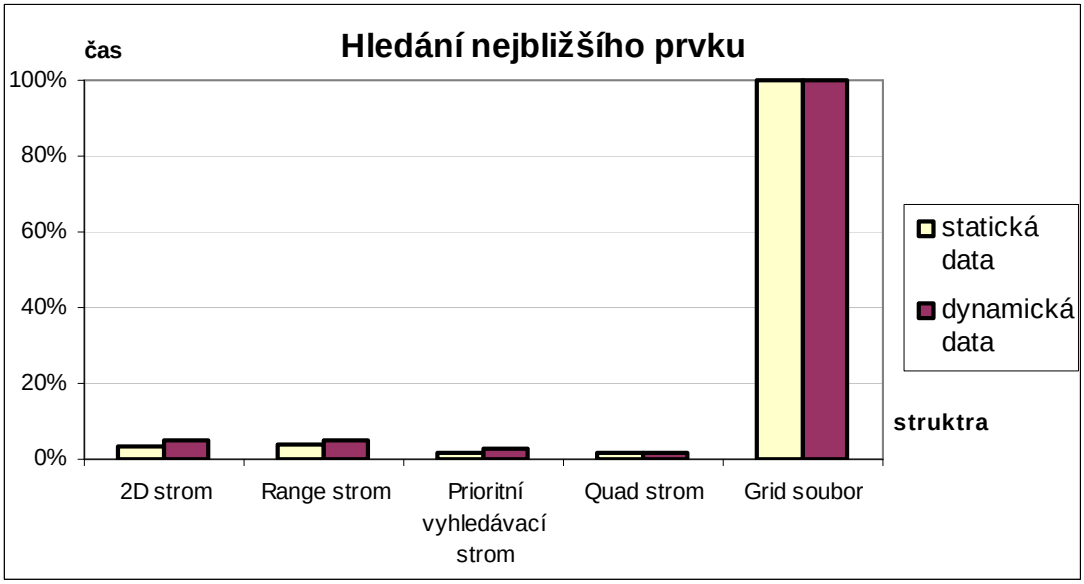
Obrázek 62: Výsledky testu (odebrání prvku ze struktury), zdroj [autor]

U odebírání prvku ze struktury dosáhl nejlepšího výsledku quad strom, u kterého není nutné hledat zástupce odebíraného prvku (2D strom) ani provádět jednoduché rotace (prioritní vyhledávací strom).



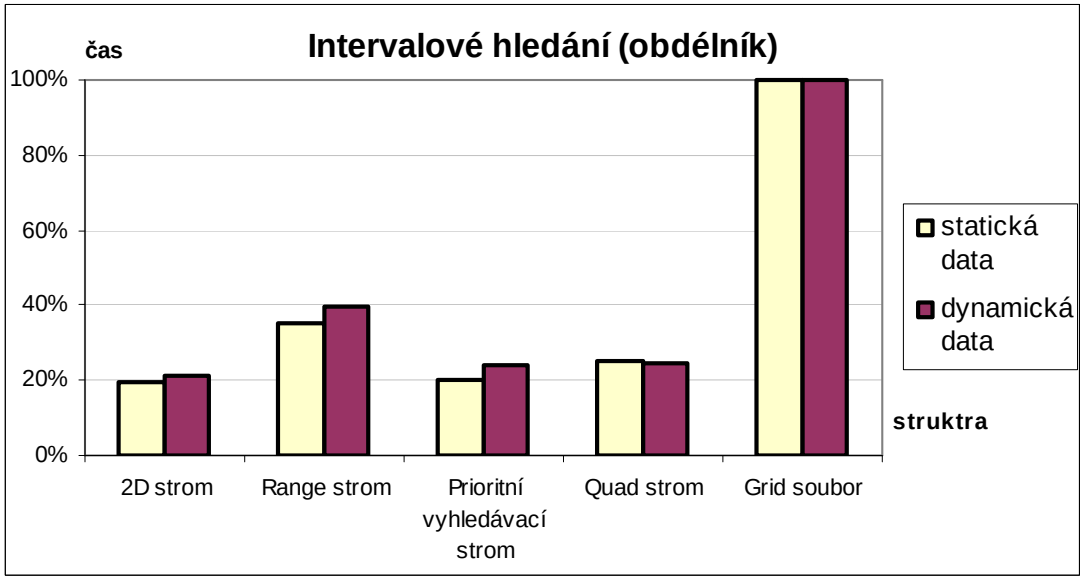
Obrázek 63: Výsledky testu (hledání prvku), zdroj [autor]

Při hledání prvku podle zadaných vykázal nejlepší výsledky 2D strom. Výrazně nejvyšší čas byl naměřen u grid souboru, což nebylo nijak překvapující, neboť bylo nutné nejprve načíst do paměti blok souboru a vzhledem k jeho kapacitě porovnat až 2000 prvků.



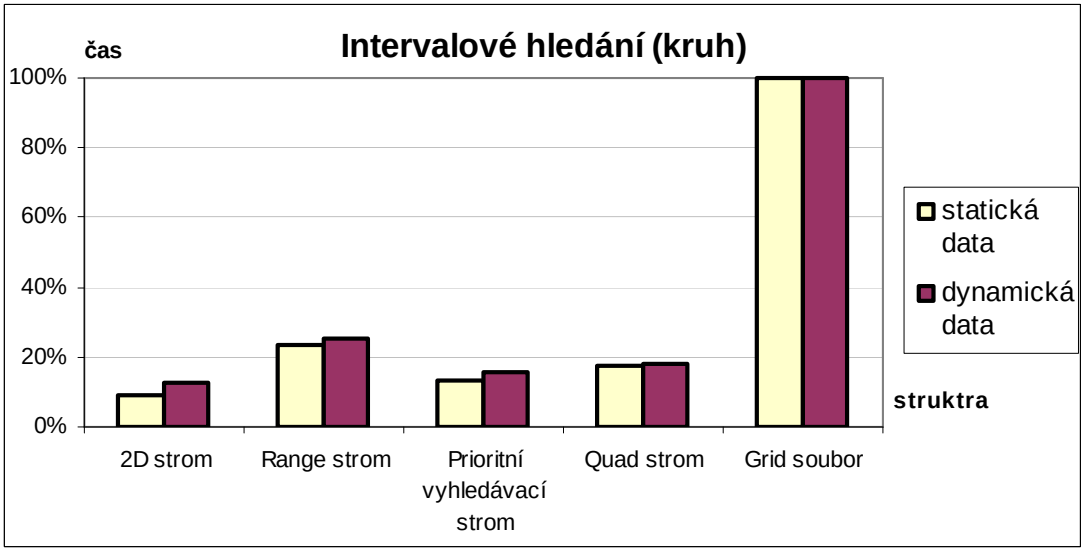
Obrázek 64: Výsledky testu (hledání nejbližšího prvku), zdroj [autor]

Pro hledání nejbližšího prvku k zadaným souřadnicím se ukázala jako nejvhodnější datová struktura quad strom. U statických dat příliš nezaostal ani prioritní vyhledávací strom.



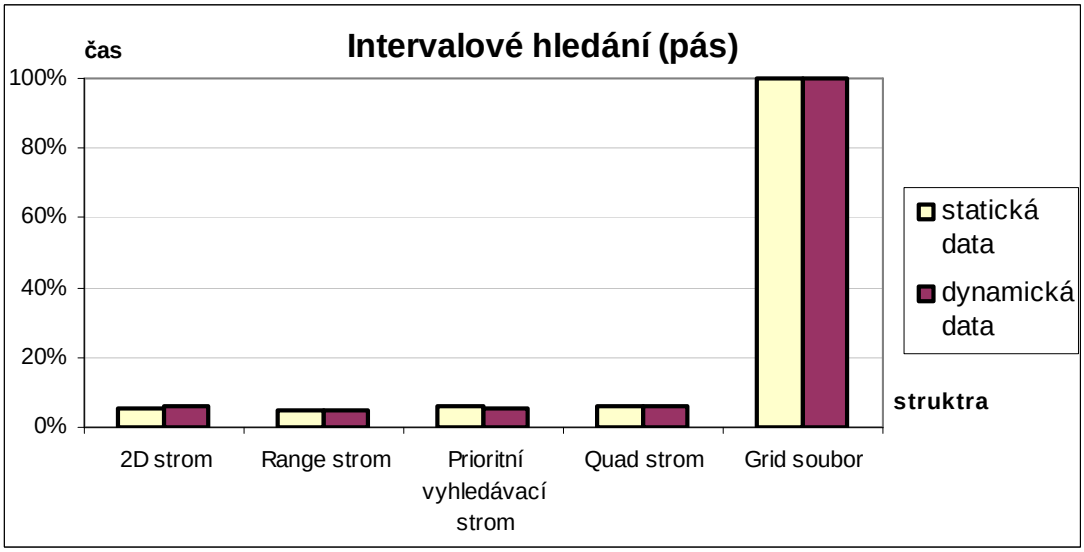
Obrázek 65: Výsledky testu (hledání v obdélníkovém segmentu), zdroj [autor]

Při prohledávání obdélníkového segmentu byl nejnižší čas neměřen pro 2D strom, prioritní vyhledávací strom vykázal pouze nepatrně vyšší hodnoty.



Obrázek 66: Výsledky testu (hledání v kruhovém segmentu), zdroj [autor]

Při hledání všech prvků z kruhového segmentu dopadly výsledky velmi podobně jako u předcházející operace.



Obrázek 67: Výsledky testu (hledání v pruhovém segmentu), zdroj [autor]

Poslední testovanou operací bylo tzv. pruhové hledání, které je vymezeno dvojicí hodnot pro zeměpisnou šířku nebo dvojicí hodnot pro zeměpisnou délku. Nejpříznivěji dopadl výsledek u datové struktury range strom, který pro tuto operaci využívá zřetězený seznam listů.

5.3. Zhodnocení testů

Z provedených testů lze odvodit, že datová struktura grid soubor dosáhla výrazně horších výsledků než zbylé stromové struktury. Možným řešením by samozřejmě bylo zmenšení kapacity bloku, což by například u hledání konkrétního prvku znamenalo načítání menšího bloku do operační paměti a následné procházení menšího počtu záznamů z tohoto bloku. Zároveň by však snížením maximální kapacity bloku došlo k ještě výraznějšímu zvýšení časové náročnosti u operace pro vybudování struktury, obzvláště v případě, kdy nejsou vstupní data známa předem.

Pro aplikace, u kterých jsou vstupní data známa předem a hlavní důraz je kladen na operace pro hledání konkrétního nebo nejbližšího prvku, lze doporučit datové struktury 2D strom, quad strom a případně také prioritní vyhledávací strom. Posledně jmenovaný se sice vyznačuje vyšší náročností při budování struktury, což je způsobeno jedním řazením prvků navíc vždy při hledání mediánu, ale při zmíněných operacích pro hledání nijak výrazně nezaostává.

V případě, že by hlavním požadavkem aplikace bylo intervalové hledání, lze za nejvhodnější datovou strukturu označit 2D strom, který dosáhl nejlepších výsledků pro hledání v čtvercovém i kruhovém segmentu, jen nepatrně horší byl prioritní vyhledávací strom. Strukturu range strom lze doporučit pouze v případě, že by nejčastější operací bylo tzv. pruhové hledání.

Za předpokladu, že by operace pro vyhledávání nebyly podstatné a důraz by byl kladen hlavně na rychlé vybudování struktury, vložení a odebrání prvku, lze doporučit datovou strukturu 2D strom nebo quad strom. Budování struktury z předem známých statických dat je rychlejší v případě quad stromu, obdobně jako odebírání prvku, což způsobuje fakt, že není nutné hledat zástupce odebíraného prvku jako ve 2D stromu. Naopak vložení prvku proběhne rychleji ve 2D stromu.

6. Případová studie

Na Katedře softwarových technologií Univerzity Pardubice probíhal výzkum zabývající se určením nejbližšího bodu na daném segmentu železniční trati pro GPS souřadnice, které odpovídají poloze vlakové soupravy. Vzhledem k tomu, že zaměření výzkumu bylo příbuzné k tématu datových struktur pro uchovávání geografických dat, vznikla nad rámec samotného zadání diplomové práce specifická úloha, která poskytla možnost využít implementované datové struktury pro reálný příklad z praxe.

Cíl této úlohy spočíval v hledání nejbližšího hektometrovníku na úseku železniční trati mezi stanicemi *Pardubice - Černá za Bory* a *Starý Kolín* pro jednotlivé polohy, které odpovídaly GPS souřadnicím průjezdů vlaků *CityElefan*. Formát podkladových dat pro případovou studii ukazuje obrázek 68 (polohy hektometrovníků) a obrázek 69 (polohy průjezdů).

	A	B	C	D	E	F	G	H
1	**TUDU**	**hektometr**	**E**			**N**		
2	150118	300.0	15	50	9.646	50	1	34.442
3	150118	300.1	15	50	4.876	50	1	35.477
4	150118	300.2	15	50	0.107	50	1	36.512
5	150118	300.3	15	49	55.338	50	1	37.548
6	150118	300.4	15	49	50.568	50	1	38.583
7	150118	300.5	15	49	45.799	50	1	39.618
8	150118	300.6	15	49	41.029	50	1	40.653
9	150118	300.7	15	49	36.259	50	1	41.688

Obrázek 68: Vzorek dat ze souboru s polohami hektometrovníků, zdroj [autor]

	A	B	C	D	E	F	G
1	CISLO_VLAKU	ZEM_SIRKA	ZEM_DELKA	RYCHLOST	AZIMUT	VOZIDLO	DATUM_CA
57	48701	50,0303	15,81705	64	104	91547123022	14.02.2011 05:01:55
58	48701	50,02882	15,82409	64	108	91547123022	14.02.2011 05:02:25
59	48701	50,02726	15,83128	65	108	91547123022	14.02.2011 05:02:55
60	55225	50,0245	15,3392	81	48	91547123017	14.02.2011 00:17:26
61	55225	50,02879	15,34577	80	43	91547123017	14.02.2011 00:17:56

Obrázek 69: Vzorek dat ze souboru s polohami průjezdů vlaků CityElefant, zdroj [autor]

Úloha byla zpracována pro všechny datové struktury uvedené v předchozích kapitolách, pro efektivní vyhledávání nejbližšího bodu k zadaným souřadnicím byla dodatečně implementována operace *ProximitySearch*, jejíž popis pro jednotlivé struktury je uveden níže (pomocí testů byla v předchozí kapitole také porovnávána). Případová studie byla navíc rozšířena i o grafickou podporu (obrázek 53 v kapitole 4.1.7.)

6.1. Hledání nejbližšího bodu ve 2D stromu

Princip pro operaci *ProximitySearch* u 2D stromu vychází z filozofie použité pro intervalové vyhledávání ve 2D stromu (kapitola 3.1.5). Na počátku se za nejbližší vrchol považuje kořen stromu, jehož vzdálenost od zadaných souřadnic se uloží do globální proměnné. Následně se stromová struktura prochází pomocí prohlídky, přičemž při nalezení nového bližšího vrcholu se aktualizuje hodnota globální proměnné s uloženou vzdáleností a samozřejmě se zapamatuje aktuálně nejbližší vrchol. Ze znalosti zadaných souřadnic a aktuální nejnižší vzdálenosti lze snadno určit čtvercový segment, v němž se může nacházet ještě bližší prvek. Prvky nacházející se vně tohoto segmentu není nutné kontrolovat a jím odpovídající větve stromu se tedy neprocházejí.

Pro zvýšení efektivity se během prohlídky v každém vrcholu rozhodne, který z podstromů se zkontroluje dříve. Jako první se prohledá podstrom, do kterého spadá poloha odpovídající hledaným souřadnicím.

6.2. Hledání nejbližšího bodu v range stromu

Pro nalezení nejbližšího bodu využívá range strom pouze strom první dimenze, který je v tomto konkrétním případě organizován podle zeměpisné šířky. Prohledávání začíná od kořene stromu směrem k listům a v každém navigačním vrcholu se volí směr dalšího traverzování (podle totožné strategie jako u hledání prvku v kapitole 3.2.4.). Po dosažení úrovně listů se za nejbližší považuje prvek, ke kterému bylo dotraverzováno. Následně se do globální proměnné uloží vzdálenost mezi tímto prvkem a hledanými souřadnicemi.

Ve druhé fázi algoritmu se prochází obousměrně zřetězený seznam listů od prvku, k němuž bylo dotraverzováno, směrem dopředu a následně také směrem dozadu. Ze znalosti zadaných souřadnic a aktuálně nejnižší vzdálenosti lze určit interval zeměpisné šířky, v němž se může vyskytovat bližší prvek než aktuálně nalezený. Díky skutečnosti, že jsou prvky ve zřetězeném seznamu seřazeny právě podle hodnot zeměpisné šířky, není nutné procházet celý seznam. V okamžiku dosažení prvku, který leží mimo vypočtený interval zeměpisné šířky, se v procházení zřetězeného seznamu plnohodnotných prvků dále nepokračuje. Navíc v případě nalezení nového bližšího prvku se sníží hodnota nejmenší vzdálenosti, čímž se zúží také prohledávaný interval.

6.3. Hledání nejbližšího bodu v prioritním vyhledávacím stromu

Vyhledávání nejbližšího prvku k zadaným souřadnicím začíná u prioritního vyhledávacího stromu směrem od kořene k listům. Na začátku se opět považuje za nejbližší bod vrchol odpovídající kořenu stromu a jeho vzdálenost od hledaných souřadnic se uloží do globální proměnné. Následně se stromová struktura prochází pomocí prohlídky, přičemž nejprve se navštíví ten podstrom, do kterého podle principů binárního vyhledávacího stromu patří hledaná hodnota zeměpisné šířky. Při nalezení bližšího prvku se aktualizuje hodnota vzdálenosti v globální proměnné.

Obdobně jako v předešlých případech lze vypočítat čtvercový interval, který má smysl dále prohledávat. Zbylé větve stromu lze na základě pravidel uvedených u intervalového hledání (kapitola 3.3.5) vynechat.

6.4. Hledání nejbližšího bodu v quad stromu

Filozofie operace *ProximitySearch* u quad stromu je prakticky totožná jako u 2D stromu. Rozdíl spočívá pouze v tom, že prováděná prohlídka pokračuje do čtyř směrů odpovídajících jednotlivým kvadrantům. Z důvodu efektivnosti se nejprve se navštíví kvadrant, jehož území obsahuje hledané souřadnice. Poté oba kvadranty, které s ním sdílejí společnou hranu, a nakonec zbylý čtvrtý kvadrant.

6.5. Hledání nejbližšího bodu v grid souboru

Při hledání nejbližšího bodu v grid souboru se nejprve pomocí lineárních stupnic a adresáře určí číslo bloku, do něhož geograficky patří hledané souřadnice. Prohledání tohoto bloku by však nebylo dostatečné, neboť může nastat situace, kdy se hledaný bod nachází v okrajové části tohoto bloku a v tom případě může nejbližší bod odpovídat i prvku ze sousedního bloku. Z toho důvodu se v adresáři naleznou také čísla všech sousedních bloků.

Ve druhém kroku se vypočítá vzdálenost mezi zadanými souřadnicemi a všemi prvky z dříve určených bloků. Nejbližšímu bodu samozřejmě odpovídá prvek, u něhož byla hodnota vypočítané vzdálenosti nejmenší.

7. Závěr

Cílem diplomové práce bylo provést seznámení s vybranými datovými strukturami pro uchovávání geografických dat, následně realizovat jejich vzorové implementace a pomocí nich provést srovnání a doporučení jejich nasazení podle typu řešené aplikace.

V teoretické části práce byla vysvětlena filozofie datových struktur 2D strom, range strom, prioritní vyhledávací strom, quad strom a grid soubor na malém vzorku reálných dat z území České republiky.

V implementační části byla v programovacím jazyce C# (platforma .NET) vytvořena desktopová aplikace, která obsahuje vzorové implementace pro všechny výše uvedené struktury. Během fáze ladění byl aktivní verifikační mechanismus, který kontroloval nejen správné počty prvků v datové struktuře před a po provedení jednotlivých operací, ale také korektnost umístění všech prvků v rámci dané struktury. Aplikace pracovala nejprve s malou množinou reálných dat, na které byly vysvětleny základní principy jednotlivých datových struktur, a následně i s větší množinou obsahující 16119 záznamů.

Právě na této větší množině dat, která obsahovala zeměpisné souřadnice obcí v České republice, byly pro jednotlivé datové struktury provedeny testy základních operací. Na základě jejich vyhodnocení byla doporučena vhodná oblast použití pro jednotlivé struktury. Na závěr byla zpracována případová studie zabývající se hledáním nejbližšího hektometrovníku na železniční trati pro zadanou polohu vlaku, tato studie slouží jako vhodná ukázka možného využití diplomové práce v praxi.

8. Použitá literatura a ostatní zdroje

[1] SAMET, Hanan. *Foundations of multidimensional and metric data structures*. San Francisco : Morgan Kaufmann, 2006. 1024 s. ISBN 978-0-12-369446-1.

[2] MEHTA, Dinesh P.; SAHNI, Sartaj. *Handbook of Data Structures and Applications*. London : Chapman & Hall, 2004. 1392 s. ISBN 1-58488-435-5.

[3] LEWIS, Harry. R.; DENENBERG, Larry. *Data structures and their algorithms*. Reading : Addison-Wesley, 1991. 509 s. ISBN 978-0673397362.

[4] GOODRICH, Michael T.; TAMASSIA, Roberto. *Algorithm design*. Hoboken : John Wiley & Son, 2002. 708 s. ISBN 0-471-38365-1.

[5] BRABEC, František. *SAND Internet Browser* [online]. 2007 [cit. 2011-03-29]. K-D Tree Demo. Dostupné z WWW: <http://donar.umiacs.umd.edu/quadtree/points/kdtree.html>.

[6] BRABEC, František. *SAND Internet Browser* [online]. 2007 [cit. 2011-03-29]. Priority Search Tree Demo. Dostupné z WWW: <http://donar.umiacs.umd.edu/quadtree/points/priority.html>.

[7] STEFANOV, Emil. *EmilStefanov.net* [online]. c2011 [cit. 2011-04-03]. Multi-Dimensional Range Search Tree. Dostupné z WWW: <http://www.emilstefanov.net/Projects/RangeSearchTree.aspx>.

[8] *Stavebnictví, architektura, bydlení - eStav.cz* [online]. 2011 [cit. 2011-03-30]. Největší města ČR. Dostupné z WWW: <http://www.estav.cz/stat/top/mesta-obyv-cz.asp>.

[9] *GeoNames* [online]. 2010 [cit. 2011-04-03]. Download server. Dostupné z WWW: <http://download.geonames.org/export/dump/>.

[10] *Astrology Pacific* [online]. 2011 [cit. 2011-03-30]. Databáze měst. Dostupné z WWW: <http://astrolot.cz/city/!city.html>.

[11] KAVIČKA, Antonín. *Vyhledávací stromy*. Elektronické sylaby přednášek k předmětu Datové struktury a algoritmy. Pardubice, 2010.

[12] KAVIČKA, Antonín. *Grid file*. Elektronické sylaby přednášek k předmětu Datové struktury a algoritmy. Pardubice, 2010.

Příloha A – CD s vytvořenou aplikací

Pro korektní spuštění vytvořené podpůrné aplikace je nutné rozbalit celý obsah souboru *DiplomovaPrace.rar* do adresáře s oprávněním ke čtení a k zápisu. V rozbalené složce se nacházejí následující položky:

- *DiplomovaPrace.exe* – spustitelný program k vytvořené aplikaci,
- *nastaveni.txt* – textový soubor obsahující informaci o počtu prvků, pro něž je podporován grafický výstup, a informaci o velikosti bloku pro grid soubor,
- *vzorovaData.txt*, *mestaCZ_SK.txt* a *mestaCZ.txt* – textové soubory s množinami vzorových dat, z nichž lze ve vytvořené aplikaci vybudovat datové struktury,
- *prujezdy_vlaku.txt* a *zeleznice_hektometry.txt* – textové soubory s daty pro zpracovanou případovou studii,

- *readMe* – textový soubor s popisem vytvořené aplikace a přiložených souborů,
- *Strom2D, RangeTree, PrioritySearchTree, QuadStrom a GridSoubor* – složky s uloženými vzorovými datovými strukturami, které je možné v aplikaci načíst z menu *Soubor – Načíst…*(stavy těchto datových struktur odpovídají obrázkům z teoretické části diplomové práce – kapitola 3),
- *ProudyTest* – složka s textovými soubory, které se v aplikaci využívají při provádění ukázkového testu,
- *Vystup* – složka, do které se ukládají výsledky ukázkových testů a výstupy z případové studie,
- *Temp* – složka pro dočasné uložení blokově orientovaného souboru, jenž je potřebný pro práci s grid souborem.

Příloha B – programátorská dokumentace

Tato příloha popisuje jednotlivé třídy, které byly implementovány pro potřeby podpůrné aplikace. Stručný popis včetně diagramů tříd již byl uveden v kapitole 4.2., zde je popsán význam jednotlivých metod.

Třída DvoudimenzionalniData

Třída *DvoudimenzionalniData* představuje objekt se dvěma klíči libovolného datového typu, který implementuje rozhraní *IComparable*.

Atributy třídy *DvoudimenzionalniData*:

- *x* – první klíčová hodnota,
- *y* – druhá klíčová hodnota.

Metody třídy *DvoudimenzionalniData*:

- *DvoudimenzionalniData(T x, T y)* – konstruktor třídy, který nastaví klíčové hodnoty podle předaných parametrů,
- *PorovnaniX(DvoudimenzionalniData<T> a, DvoudimenzionalniData<T> b)* – metoda porovnávající dvě předané instance této třídy podle klíče *x*,
- *PorovnaniY(DvoudimenzionalniData<T> a, DvoudimenzionalniData<T> b)* – metoda porovnávající dvě předané instance této třídy podle klíče *y*,
- *Vypis()* – metoda, která vrací řetězec vytvořený spojením hodnot obou klíčů.

Třída Mesto

Třída *Mesto* je odvozena pomocí dědičnosti od předchozí třídy a představuje objekt se zeměpisnými souřadnicemi a názvem.

Atribut třídy *Mesto*:

- *nazev* – řetězec obsahující název daného města.

Metody třídy *Mesto*:

- *Mesto(string nazev, T x, T y)* – konstruktor třídy, podle předaných parametrů nastaví obě souřadnice a název,
- *Vypis()* – metoda, která vrací řetězec tvořený názvem a hodnotami obou souřadnic,
- *ToString()* – metoda, která vrací řetězec obsahující pouze název.

Rozhraní Struktura

Rozhraní *Struktura* definuje metody, které musí být implementovány pro každou datovou strukturu.

Metody definované rozhraním *Struktura*:

- *JePrazdny()* – metoda vracející booleovskou hodnotu, pokud je struktura prázdná, vrací hodnotu *true*, v opačném případě *false*,
- *Mohutnost()* – metoda, která vrací celočíselnou hodnotu odpovídající počtu prvků ve struktuře,
- *Vybuduj(List<Mesto<T>> prvky)* – metoda, která ze zadaných prvků předaných parametrem *prvky* vybuduje novou strukturu,
- *Vloz(Mesto<T> uzel)* – metoda, která do aktuální struktury vloží nový prvek předaný pomocí parametru *uzel*,
- *Odeber(T x, T y)* – metoda, která z aktuální struktury odebere prvek se zadanými souřadnicemi *x* a *y*, tento prvek poté vrací jako návratovou hodnotu,
- *Najdi(T x, T y)* – metoda, která v aktuální struktuře vyhledá prvek se zadanými souřadnicemi *x* a *y*, následně tento prvek vrací jako návratovou hodnotu,
- *NajdiNejbizsi(T x, T y)* – metoda, která vrací jako návratovou hodnotu prvek ležící nejblíže k zadaným souřadnicím x a y,
- *NajdiInterval(T x1, T y1, T x2, T y2, AkceCallback akce)* – metoda, která provede zvolenou akci nad každým prvkem ze zadaného obdélníkového intervalu,

- *NajdiKruhovyInterval(T x1, T y1, double radius, AkceCallback akce)* – metoda, která provede zvolenou akci nad každým prvkem ze zadaného kruhového intervalu,
- *NajdiHorizontalniPas(T x1, T x2, AkceCallback akce)* – metoda, která provede zvolenou akci nad každým prvkem z intervalu, který vymezuje dvojice klíčových *x*-ových hodnot,
- *NajdiVertikalniPas(T y1, T y2, AkceCallback akce)* – metoda, která provede zvolenou akci nad každým prvkem z intervalu, který vymezuje dvojice klíčových *y*-ových hodnot,
- *Prohlidka(AkceCallback akce, out int maxHloubka, RazeniProhlidka razeni)* – metoda, která vykoná zvolenou akci nad všemi prvky datové struktury v pořadí definovaném parametrem *razeni*, pomocí parametru *maxHloubka* vrací maximální hloubku struktury,
- *GrafickaProhlidka(Control control)* – metoda, která zobrazí aktuální stav struktury v grafické podobě na komponentu předanou parametrem *control*,
- *ZobrazeniNaMapu(Control control)* – metoda, která zobrazí všechny prvky struktury v mapovém rozložení na komponentu předanou parametrem *control*.

Třída Prvek2D

Třída *Prvek2D* slouží k vytváření instancí prvků, které jsou uchovávány v datové struktuře 2D strom.

Atributy třídy *Prvek2D*:

- *levyPotomek* – odkaz na prvek stejného typu, který představuje levého syna,
- *pravyPotomek* – odkaz na prvek stejného typu, který představuje pravého syna,
- *otec* – odkaz na prvek stejného typu, který představuje otce.

Metoda třídy *Prvek2D*:

- *Prvek2D(Mesto<T> mesto)* – konstruktor třídy, na základě objektu předaného pomocí parametru *mesto* vytvoří novou instanci této třídy.

Třída Strom2D

Třída *Strom2D* představuje datovou strukturu 2D strom. Implementuje rozhraní *Struktura*, a tudíž obsahuje všechny metody definované tímto rozhraním, z toho důvodu jsou zde popsány pouze zbylé metody.

Atributy třídy *Strom2D*:

- *koren* – prvek typu *Prvek2D* odpovídající kořenu stromu,
- *pocet* – celočíselná hodnota s informací o počtu uložených prvků ve struktuře,
- *porovnani* – delegát, který umožňuje porovnat hodnoty souřadnic dvou prvků typu *Prvek2D*, přestože datové typy těchto souřadnic nejsou předem známé.

Metody třídy *Strom2D*:

- *Strom2D()* – konstruktor, který vytvoří novou prázdnou strukturu,
- *vybuduj(List<Mesto<T>> prvky, Prvek2D<T> predek, int uroven, bool levy)* – metoda, která zajistí vybudování podstromu z prvků předaných v parametru *prvky*, booleovský parametr *levy* určuje, zda se vytvořený podstrom připojí k předkovi pomocí odkazu na levého nebo pravého syna,
- *rozdelPrvky(List<Mesto<T>> prvky, bool dimenzeX, out List<Mesto<T>> lprvky, out List<Mesto<T>> pprvky, out int median)* – metoda, která podle zadané dimenze rozdělí prvky předané parametrem *prvky* do seznamů *lprvky* a *pprvk*y, pomocí parametru *median* navíc vrací index hraničního prvku,
- *odeberPrvek(Prvek2D<T> prvek, int uroven)* – metoda, která provede odstranění předaného prvku ze struktury, parametr *uroven* označuje úroveň, v níž se odebíraný prvek nachází,
- *najdiMinimum(Prvek2D<T> vrchol, Prvek2D<T> min, bool dimenzeX, int uroven, ref int urovenMin)* – pomocná metoda, která hledá ve stromu s kořenem odpovídajícím parametru *vrchol* prvek s minimální hodnotou klíče pro zadanou dimenzi,

- *najdiMaximum(Prvek2D<T> vrchol, Prvek2D<T> max, bool dimenzeX, int uroven, ref int urovenMax)* – pomocná metoda, která hledá ve stromu s kořenem odpovídajícím parametru *vrchol* prvek s maximální hodnotou klíče pro zadanou dimenzi,

- *najdiNejblizsi(T x, T y, Prvek2D<T> vrchol, ref double nejblizsiVzdalenost, ref Prvek2D<T> nejblizsiVrchol, int uroven)* – metoda, která nalezne ve stromu s kořenem odpovídajícím parametru *vrchol* prvek, jenž leží nejblíže k zadaným souřadnicím *x* a *y*, následně tento prvek vrací pomocí parametru *nejblizsiVrchol*,

- *najdiInterval(T x1, T y1, T x2, T y2, Prvek2D<T> vrchol, int uroven, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v obdélníkovém intervalu, jenž je vymezen souřadnicemi *x1*, *y1*, *x2* a *y2*,

- *najdiKruhovyInterval(T x, T y, double radius, Prvek2D<T> vrchol, int uroven, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v kruhovém intervalu, jenž určuje střed se souřadnicemi [*x*, *y*] a parametr *radius*,

- *najdiPas(T min, T max, Prvek2D<T> vrchol, AkceCallback akce, int uroven, bool DimenzeX)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v tzv. pruhovém intervalu, jenž je určen dvojicí klíčových hodnot *min* a *max* (parametr *DimenzeX* určuje, zda se jedná o *x*-ové nebo *y*-ové klíče),

- *prohlidka(Prvek2D<T> vrchol, AkceCallback akce, int uroven, ref int maxUroven)* – metoda provádějící pre-order prohlídku, vykoná zvolenou akci nad všemi prvky ze stromu s kořenem odpovídajícím parametru *vrchol*, pomocí parametru *maxUroven* vrací maximální dosaženou hloubku stromu,

- *verifikacniProhlidka(Prvek2D<T> vrchol, AkceCallback akce)* – pomocná metoda využívaná pouze ve fázi ladění, provádí obdobnou činnost jako předchozí metoda, ale navíc zobrazuje varování při nekorektním umístění prvků ve struktuře,

- *vykreslit(Control control, Prvek2D<T> vrchol, int uroven, int x, int y)* – metoda, která graficky zobrazí prvek odpovídající parametru *vrchol* na komponentu předanou parametrem *control* (střed kreslení na komponentě určují hodnoty *x* a *y*),

- *zobrazNaMapu(Control control, Prvek2D<T> prvek, int uroven)* – metoda, která vykreslí prvek odpovídající parametru *vrchol* na mapový podklad, jemuž odpovídá komponenta předaná parametrem *control*,
- *nakresliRez(Control control, Prvek2D<T> prvek, int uroven)* – pomocná metoda využívaná předchozí metodou, zobrazí na mapový podklad řez, který odpovídá rozdělení prvků v datové struktuře.

Třída PrvekRange

Třída *PrvekRange* slouží k vytváření instancí prvků, které jsou uchovávány v datové struktuře range strom.

Atributy třídy *PrvekRange*:

- *levyPotomek* – odkaz na prvek stejného typu, který představuje levého syna,
- *pravyPotomek* – odkaz na prvek stejného typu, který představuje pravého syna,
- *otec* – odkaz na prvek stejného typu, který představuje otce,
- *druhaDimenze* – odkaz na prvek stejného typu, který leží ve stromu opačné dimenze,
- *predchozi* – odkaz na prvek stejného typu, který představuje předchůdce ve zřetězeném seznamu listů,
- *dalsi* – odkaz na prvek stejného typu, který představuje následníka ve zřetězeném seznamu listů,
- *platny* – booleovský příznak určující, zda se jedná o plnohodnotný nebo navigační vrchol.

Metody třídy *PrvekRange*:

- *PrvekRange(T x, T y)* – konstruktor třídy, který na základě souřadnic předaných pomocí parametrů *x* a *y* vytvoří nový navigační vrchol range stromu,
- *PrvekRange(Mesto<T> mesto)* – konstruktor třídy, který na základě objektu předaného pomocí parametru *mesto* vytvoří nový plnohodnotný vrchol range stromu.

Třída RangeTree

Třída *RangeTree* představuje datovou strukturu range strom. Implementuje rozhraní *Struktura*, a tudíž obsahuje všechny metody definované tímto rozhraním, zde jsou proto popsány pouze ostatní metody.

Atributy třídy *RangeTree*:

- *koren* – prvek typu *PrvekRange* odpovídající kořenu stromu,
- *pocet* – celočíselná hodnota s informací o počtu uložených prvků ve struktuře,
- *pocetUrovni* – celočíselná hodnota uchovávající hodnotu maximální hloubky stromu,
- *porovnani* – delegát, který umožňuje porovnat hodnoty souřadnic dvou prvků typu *PrvekRange*, přestože datové typy těchto souřadnic nejsou předem známé.

Metody třídy *RangeTree*:

- *RangeTree()* – konstruktor, který vytvoří novou prázdnou strukturu,
- *vybudujStrom(PrvekRange<T> prvekDruhaDimenze, List<Mesto<T>> prvky, bool dimenzeX, ref int maxUroven)* – metoda, která zajistí vybudování stromu z prvků předaných v parametru *prvky*, booleovský parametr *dimenzeX* určuje, zda se vytvoří strom první nebo druhé dimenze, pro případ budování stromu druhé dimenze se navíc v parametru *prvekDruhaDimenze* předá prvek, pomocí něhož se vybudovaný strom připojí k hlavnímu stromu,
- *vybudujPodstrom(PrvekRange<T> predek, List<Mesto<T>> prvky, bool dimenzeX, ref PrvekRange<T> predchozi, int uroven, ref int maxUroven)* – pomocná metoda, která se volá rekurzivně a ze zadaných prvků (v parametru *prvky*) vybuduje pro danou dimenzi podstrom předka, parametr *predchozi* odkazuje na aktuálně poslední prvek zřetězeného seznamu listů, k němuž se následně připojí listy vytvářeného podstromu,
- *rozdelPrvky(List<Mesto<T>> prvky, bool dimenzeX, out List<Mesto<T>> lprvky, out List<Mesto<T>> pprvky)* – metoda, která podle zadané dimenze rozdělí prvky předané parametrem *prvky* do seznamů *lprvky* a *pprvk*y,

- *vlozMesto(PrvekRange<T> predek, Mesto<T> mesto, bool DimenzeX, int uroven)* – metoda, která podle zadané dimenze vloží nový prvek předaný parametrem *mesto* do stromu s kořenem odpovídajícím parametru *predek*,

- *odeber(PrvekRange<T> vrchol, T x, T y, bool DimenzeX, int uroven)* – metoda, která provede podle zadané dimenze odstranění prvku se souřadnicemi *x* a *y* ze stromu, jehož kořen odpovídá parametru *vrchol*,

- *odeberPrvek(PrvekRange<T> prvek, bool DimenzeX)* – pomocná operace, která fyzicky odebere zadaný prvek ze struktury tím, že otce odebíraného prvku nahradí sourozencem odebíraného prvku,

- *zkontrolujPocetUrovni(PrvekRange<T> vrchol, int uroven, ref int maxUroven)* – pomocná metoda, která při odstranění prvku ze struktury zkontroluje, zda nedošlo ke snížení maximální hloubky stromu a případně aktualizuje atribut *pocetUrovni*,

- *najdiInterval(T x1, T y1, T x2, T y2, PrvekRange<T> vrchol, bool dimenzeX, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v obdélníkovém intervalu, jenž je vymezen souřadnicemi *x1*, *y1*, *x2* a *y2*,

- *najdiKruhovyInterval(T x, T y, double radius, PrvekRange<T> vrchol, bool dimenzeX, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v kruhovém intervalu, jenž určuje střed se souřadnicemi [*x*, *y*] a parametr *radius*,

- *prohlidka(PrvekRange<T> vrchol, AkceCallback akce, bool dimenzeX)* – metoda provádějící prohlídku, dotraverzuje k nejlevějšímu listu stromu, jehož kořen odpovídá parametru *vrchol*, a následně projde zřetězený seznam se všemi plnohodnotnými prvky, nad kterými vykoná zvolenou akci (během fáze ladění navíc kontroluje korektnost umístění prvků ve struktuře),

- *vykreslit(Control control, PrvekRange<T> vrchol, int uroven, int x, int y, bool dimenzeX)* – metoda, která graficky zobrazí prvek odpovídající parametru *vrchol* na komponentu předanou parametrem *control* (střed kreslení na komponentě určují hodnoty *x* a *y*), booleovský parametr *dimenzeX* určuje, zda se jedná o prvek hlavního stromu či nikoliv,

- *zobrazNaMapu(Control control, PrvekRange<T> prvek, int uroven)* – metoda, která vykreslí prvek odpovídající parametru *vrchol* na mapový podklad, jemuž odpovídá komponenta předaná parametrem *control*.

Třída PrvekPrioritySearch

Třída *PrvekPrioritySearch* představuje prvek, se kterým pracuje datová struktura prioritní vyhledávací strom.

Atributy třídy *PrvekPrioritySearch*:

- *levyPotomek* – odkaz na prvek stejného typu, který představuje levého syna,
- *pravyPotomek* – odkaz na prvek stejného typu, který představuje pravého syna,
- *otec* – odkaz na prvek stejného typu, který představuje otce,
- *hranice* – atribut sloužící k rozdělení prvků podle pravidel binárního vyhledávacího stromu.

Metoda třídy *PrvekPrioritySearch*:

- *PrvekPrioritySearch(Mesto<T> mesto)* – konstruktor třídy, který na základě objektu předaného pomocí parametru *mesto* vytvoří nový prvek.

Třída PrioritySearchTree

Třída *PrioritySearchTree* odpovídá datové struktuře prioritního vyhledávacího stromu. Implementuje rozhraní *Struktura*, a tudíž obsahuje všechny metody definované tímto rozhraním, zde jsou proto popsány pouze zbylé metody.

Atributy třídy *PrioritySearchTree*:

- *koren* – prvek typu *PrvekPrioritySearch*, který odpovídá kořenu stromu,
- *pocet* – celočíselná hodnota s informací o počtu uložených prvků ve struktuře,

- *porovnani* – delegát, který umožňuje porovnat hodnoty souřadnic dvou prvků typu *PrvekPrioritySearch*, přestože datové typy těchto souřadnic nejsou předem známé.

Metody třídy *PrioritySearchTree*:

- *PrioritySearchTree()* – konstruktor, který vytvoří novou prázdnou strukturu,
- *vybuduj(List<Mesto<T>> prvky, PrvekPrioritySearch<T> predek, bool levy)* – metoda, která zajistí vybudování podstromu z prvků předaných v parametru *prvky*, booleovský parametr *levy* určuje, zda se vytvořený podstrom připojí k předkovi pomocí odkazu na levého nebo pravého syna,
- *levaRotace(PrvekPrioritySearch<T> prvek)* – metoda, která provede nad zadaným prvkem jednoduchou levou rotaci,
- *pravaRotace(PrvekPrioritySearch<T> prvek)* – metoda, která provede nad zadaným prvkem jednoduchou pravou rotaci,
- *najdiNejblizsi(T x, T y, PrvekPrioritySearch<T> vrchol, ref double nejblizsiVzdalenost, ref PrvekPrioritySearch<T> nejblizsiVrchol)* – metoda, která nalezne ve stromu s kořenem odpovídajícím parametru *vrchol* prvek, jenž leží nejblíže k zadaným souřadnicím *x* a *y*, následně tento prvek vrací pomocí parametru *nejblizsiVrchol*,
- *najdiInterval(T x1, T y1, T x2, T y2, PrvekPrioritySearch<T> vrchol, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v obdélníkovém intervalu, jenž je vymezen souřadnicemi *x1*, *y1*, *x2* a *y2*,
- *najdiKruhovyInterval(T x, T y, double radius, PrvekPrioritySearch<T> vrchol, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v kruhovém intervalu, jenž určuje střed se souřadnicemi [*x*, *y*] a parametr *radius*,
- *najdiPas(T min, T max, PrvekPrioritySearch<T> vrchol, AkceCallback akce, bool DimenzeX)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v tzv. pruhovém intervalu, jenž je

určen dvojicí klíčových hodnot *min* a *max* (parametr *DimenzeX* určuje, zda se jedná o *x*-ové nebo *y*-ové klíče),

- *prohlidka(PrvekPrioritySearch<T> vrchol, AkceCallback akce, int uroven, ref int maxUroven)* – metoda provádějící pre-order prohlídku, vykoná zvolenou akci nad všemi prvky ze stromu s kořenem odpovídajícím parametru *vrchol*, pomocí parametru *maxUroven* vrací maximální dosaženou hloubku stromu,
- *verifikacniProhlidka(PrvekPrioritySearch<T> vrchol, AkceCallback akce)* – pomocná metoda využívaná pouze ve fázi ladění, provádí obdobnou činnost jako předchozí metoda, ale navíc zobrazuje varování při nekorektním umístění prvků ve struktuře,
- *vykreslit(Control control, PrvekPrioritySearch<T> vrchol, int uroven, int x, int y)* – metoda, která graficky zobrazí prvek odpovídající parametru *vrchol* na komponentu předanou parametrem *control* (střed kreslení na komponentě určují hodnoty *x* a *y*),
- *zobrazNaMapu(Control control, PrvekPrioritySearch<T> prvek, int uroven)* – metoda, která vykreslí prvek odpovídající parametru *vrchol* na mapový podklad, jemuž odpovídá komponenta předaná parametrem *control*,
- *nakresliRez(Control control, PrvekPrioritySearch<T> prvek)* – pomocná metoda využívaná předchozí metodou, zobrazí na mapový podklad řez, který odpovídá rozdělení prvků v datové struktuře.

Třída PrvekQuad

Třída *PrvekQuad* slouží k vytváření instancí prvků, které jsou uchovávány v datové struktuře quad strom.

Atributy třídy *PrvekQuad*:

- *sz* – odkaz na prvek stejného typu, který odpovídá synovi ležícímu v severozápadním kvadrantu,
- *sv* – odkaz na prvek stejného typu, který odpovídá synovi ležícímu v severovýchodním kvadrantu,

- *jz* – odkaz na prvek stejného typu, který odpovídá synovi ležícímu v jihozápadním kvadrantu,
- *jv* – odkaz na prvek stejného typu, který odpovídá synovi ležícímu v jihovýchodním kvadrantu,
- *otec* – odkaz na prvek stejného typu, který představuje otce,
- *platny* – booleovský příznak určující, zda se jedná o plnohodnotný nebo navigační vrchol.

Metody třídy *PrvekQuad*:

- *PrvekQuad()* – bezparametrický konstruktor třídy, vytvoří nový prvek s defaultními hodnotami atributů,
- *PrvekQuad(Mesto<T> mesto)* – konstruktor třídy, který na základě objektu předaného pomocí parametru *mesto* vytvoří nový plnohodnotný prvek.

Třída QuadStrom

Třída *QuadStrom* představuje datovou strukturu quad strom. Implementuje rozhraní *Struktura*, a tudíž obsahuje všechny metody definované tímto rozhraním, zde jsou proto popsány pouze ostatní metody.

Atributy třídy *QuadStrom*:

- *koren* – prvek typu *PrvekQuad* odpovídající kořenu stromu,
- *pocet* – celočíselná hodnota s informací o počtu uložených prvků ve struktuře,
- *porovnani* – delegát, který umožňuje porovnat hodnoty souřadnic dvou prvků typu *PrvekQuad*, přestože datové typy těchto souřadnic nejsou předem známé,
- *polovina* – delegát, který umožňuje rozdělit území podle jeho souřadnic na poloviny, přestože datové typy souřadnic nejsou předem známé.

Metody třídy *QuadStrom*:

- *QuadStrom()* – konstruktor, který vytvoří novou prázdnou strukturu,
- *vybudujPodstrom(PrvekQuad<T> vrchol, List<Mesto<T>> prvky)* – metoda, která z prvků předaných parametrem *prvky* vybuduje podstrom s kořenem odpovídajícím prvku *vrchol*,
- *rozdelPrvky(PrvekQuad<T> vrchol, List<Mesto<T>> prvky, out List<Mesto<T>> sz, out List<Mesto<T>> sv, out List<Mesto<T>> jz, out List<Mesto<T>> jv)* – metoda, která rozdělí prvky předané parametrem *prvky* do seznamů *sz*, *sv*, *jz* a *jv*, rozdělení se realizuje podle geografického umístění vzhledem k prvku *vrchol*,
- *nastavNavigacniPrvek(PrvekQuad<T> prvek)* – pomocná metoda, která nastaví předanému navigačnímu vrcholu hodnoty souřadnic,
- *odeberPrvek(PrvekQuad<T> odebirany)* – metoda, která provede fyzické odstranění předaného prvku ze struktury,
- *pocetPlatnychSourozencu(PrvekQuad<T> prvek)* – pomocná metoda, která vrací celočíselnou hodnotu odpovídající počtu plnohodnotných sourozenců zadaného prvku (je-li mezi sourozenci také navigační vrchol, vrací hodnotu *-1*),
- *najdiNejblizsi(T x, T y, PrvekQuad<T> vrchol, ref double nejblizsiVzdalenost, ref PrvekQuad<T> nejblizsiVrchol)* – metoda, která nalezne ve stromu s kořenem odpovídajícím parametru *vrchol* prvek, jenž leží nejblíže k zadaným souřadnicím *x* a *y*, následně tento prvek vrací pomocí parametru *nejblizsiVrchol*,
- *najdiInterval(T x1, T y1, T x2, T y2, PrvekQuad<T> vrchol, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v obdélníkovém intervalu, jenž je vymezen souřadnicemi *x1*, *y1*, *x2* a *y2*,
- *najdiKruhovyInterval(T x, T y, double radius, PrvekQuad<T> vrchol, AkceCallback akce)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v kruhovém intervalu, jenž určuje střed se souřadnicemi [*x*, *y*] a parametr *radius*,

- *najdiPas(T min, T max, PrvekQuad<T> vrchol, AkceCallback akce, bool DimenzeX)* – metoda, která prohledá strom s kořenem odpovídajícím parametru *vrchol* a provede zvolenou akci nad každým prvkem ležícím v tzv. pruhovém intervalu, jenž je určen dvojicí klíčových hodnot *min* a *max* (parametr *DimenzeX* určuje, zda se jedná o *x*-ové nebo *y*-ové klíče),

- *prohlidka(PrvekQuad<T> vrchol, AkceCallback akce, int uroven, ref int maxUroven)* – metoda provádějící prohlídku struktury, vykoná zvolenou akci nad všemi prvky ze stromu s kořenem odpovídajícím parametru *vrchol*, pomocí parametru *maxUroven* vrací maximální dosaženou hloubku stromu,

- *verifikacniProhlidka(PrvekQuad<T> vrchol, AkceCallback akce)* – pomocná metoda využívaná pouze ve fázi ladění, provádí obdobnou činnost jako předchozí metoda, ale navíc zobrazuje varování při nekorektním umístění prvků ve struktuře,

- *vykreslit(Control control, PrvekQuad<T> vrchol, int uroven, int x, int y)* – metoda, která graficky zobrazí prvek odpovídající parametru *vrchol* na komponentu předanou parametrem *control* (střed kreslení na komponentě určují hodnoty *x* a *y*),

- *zobrazNaMapu(Control control, PrvekQuad<T> prvek, int uroven)* – metoda, která vykreslí prvek odpovídající parametru *vrchol* na mapový podklad, jemuž odpovídá komponenta předaná parametrem *control*,

- *nakresliRez(Control control, PrvekQuad<T> prvek, int uroven)* – pomocná metoda využívaná předchozí metodou pro navigační prvky, zobrazí na mapovém podkladu řez, kterým dělí navigační prvek území na jednotlivé kvadranty.

Třída LinearniStupnice

Třída *LineraniStupnice* je využívána datovou strukturou grid soubor. Společně s adresářem tvoří navigační strukturu nad blokově orientovaným souborem.

Atributy třídy *LinearniStupnice*:

- *stupnice* - jednorozměrnému pole obsahující hodnoty libovolného porovnatelného datového typu,

- *porovnani* – delegát, který umožňuje porovnat hodnoty ve stupnici, přestože jejich datový typ není předem definovaný.

Metody třídy *LinearniStupnice*:

- *LinearniStupnice(T min, T max)* – konstruktor třídy, který vytvoří v atributu *stupnice* pole o velikosti dvou prvků s hodnotami *min* a *max*,
- *DelkaStupnice()* – metoda, která vrací celočíselnou hodnotu odpovídající délce pole v atributu *stupnice*,
- *DejHodnotu(int index)* – metoda, která vrací hodnotu nacházející se v atributu *stupnice* na zadaném indexu,
- *IndexHodnoty(T hodnota)* – metoda, která v atributu *stupnice* nalezne zadanou hodnotu a jako návratovou hodnotu vrátí celočíselný index, na němž se tato hodnota vyskytuje,
- *IndexVyssiHodnoty(T hodnota)* – obdoba předchozí metody, pouze vrací index nejnižší vyšší hodnoty, než kterou obsahuje parametr *hodnota*,
- *RozsirStupnici(int indexZaDelenim)* – metoda, která zvýší délku pole uloženého v atributu *stupnice*, na pozici zadaného indexu vloží novou dopočítanou hodnotu.

Třída Adresar

Třída *Adresar* je využívána datovou strukturou grid soubor. Společně s lineárními stupnicemi tvoří navigační strukturu nad blokově orientovaným souborem.

Atribut třídy *Adresar*:

- *mrizka* – dvourozměrné pole (matice) obsahující celočíselné hodnoty odpovídající číslům bloků v blokově orientovaném souboru.

Metody třídy *Adresar*:

- *Adresar(int cisloBloku)* – konstruktor třídy, který vytvoří mřížku s jedinou hodnotou odpovídající číslu prvního bloku v souboru (parametr *cisloBloku*),

- *DelkaMrizky(out int x, out int y)* – metoda, která vrací pomocí parametrů *x* a *y* informaci o velikosti matice v atributu *mrizka*,
- *DejMaxCisloBloku()* – metoda, která prohledá mřížku a v návratové hodnotě vrací celočíselnou hodnotu odpovídající nejvyššímu číslu bloku,
- *DejCisloBloku(int indexX, int indexY)* – metoda, která vrací celočíselnou hodnotu nacházející se v atributu *mrizka* na zadaném indexu,
- *NastavCisloBloku(int cisloBloku, int indexX, int indexY)* – metoda, která nastaví hodnotu *cisloBloku* na zadaný index v atributu *mrizka*,
- *RozsirAdresar(int indexX, int indexY, bool dimenzeX)* – metoda, která za zadaným indexem rozšíří podle dané dimenze matici uloženou v atributu *mrizka*.

Třída ZaznamSouboru

Třída *ZaznamSouboru* slouží k vytváření instancí záznamů, které jsou uchovávány v blokově orientovaném souboru.

Metody třídy *ZaznamSouboru*:

- *ZaznamSouboru(Mesto<T> mesto)* – konstruktor třídy, který na základě objektu předaného pomocí parametru *mesto* vytvoří nový záznam souboru,
- *ZaznamSouboru(string nazev, T x, T y)* – konstruktor třídy, který vytvoří nový záznam souboru na základě zadaných parametrů.

Třída Soubor

Třída *Soubor* představuje blokově orientovaný soubor, který pro svou činnost využívá datová struktura grid soubor.

Atributy třídy *Soubor*:

- *velikostHlavickySouboru* – celočíselná informace o velikosti hlavičky souboru,
- *velikostHlavickyZaznamu* – celočíselná informace o velikosti hlavičky záznamu,

- *velikostZaznamu* – celočíselná hodnota odpovídající velikosti záznamu,
- *pocetZaznamuVBloku* – celočíselná informace o tom, kolik záznamů se vejde do jednoho bloku souboru,
- *velikostBloku* – celočíselná hodnota odpovídající velikosti bloku,
- *buffer* – atribut, pomocí něhož se načítá zvolený blok souboru do operační paměti,
- *cisloBlokuVBufferu* – celočíselná informace o tom, který blok souboru je právě načten pomocí bufferu v operační paměti,
- *nazevSouboru* – řetězec obsahující cestu a název souboru,
- *pocetBloku* – celočíselná informace o počtu bloků v souboru,
- *změna* – booleovský příznak určující, zda došlo k provedení změny v bloku, jenž je aktuálně načten v bufferu.

Metody třídy *Soubor*:

- *Soubor(string nazev, int pocetZaznamuVBloku)* – konstruktor třídy, který vytvoří novou instanci souboru se zadaným názvem a nastaví maximální počet záznamů v bloku podle parametru *pocetZaznamuVBloku*,
- *NactiHlavicku()* – metoda, která načte hlavičku blokově orientovaného souboru,
- *VlozZaznam(ZaznamSouboru<T> zaznam, int blokVlozeni)* – metoda, která se pokusí vložit zadaný záznam do bloku odpovídajícímu parametru *blokVlozeni*, v případě neúspěchu (je-li blok plný) vrací hodnotu *false*,
- *ZrusZaznam(int blokVymazani, int zaznamVymazani)* – metoda, která ze zadaného bloku odstraní záznam nacházející se na pozici *zaznamVymazani*, při vyprázdnění a uvolnění bloku vrací hodnotu *true*,
- *DejZaznamyBloku(int blok)* – metoda, která vrací všechny záznamy nacházející se v zadaném bloku,
- *NovyBlok()* – metoda, která vytvoří nový blok, v návratové hodnotě vrací číslo odpovídající tomuto bloku,

- *NactiBufferZeSouboru(int cisloBloku)* – metoda, která načte do bufferu blok se zadaným číslem,
- *ZapisBufferDoSouboru()* – metoda, která zapíše do souboru blok, který je aktuálně načten v bufferu (pouze v případě, že byla v tomto bloku provedena změna),
- *UvolniBlok(int cisloBloku)* – metoda, která uvolní blok se zadaným číslem, čímž dojde ke snížení velikosti celého souboru,
- *ZapisVse()* – metoda volaná při ukončení programu, zapíše provedené změny do souboru.

Třída GridSoubor

Třída *GridSoubor* představuje datovou strukturu grid soubor. Implementuje rozhraní *Struktura*, a tudíž obsahuje všechny metody definované tímto rozhraním, zde jsou proto popsány pouze ostatní metody.

Atributy třídy *GridSoubor*:

- *soubor* – blokově orientovaný soubor,
- *stupniceX* – lineární stupnice obsahující *x*-ové hodnoty,
- *stupniceY* – lineární stupnice obsahující *y*-ové hodnoty,
- *adresar* – adresář s mřížkou, která obsahuje čísla bloků v souboru,
- *pocet* – celočíselná hodnota s informací o počtu uložených prvků ve struktuře,
- *porovnani* – delegát, který umožňuje porovnat hodnoty souřadnic dvou prvků , přestože datové typy souřadnic nejsou předem známé,
- *polovina* – delegát, který umožňuje rozdělit území podle jeho souřadnic na poloviny, přestože datové typy souřadnic nejsou předem známé,
- *smazatSoubor* – pomocný booleovský příznak určující, zda má být blokově orientovaný soubor ponechán na disku či nikoliv,

- *disposed* – pomocný booleovský příznak určující, zda již byla instance této třídy zrušena.

Metody třídy *GridSoubor*:

- *GridSoubor()* – konstruktor, který vytvoří novou prázdnou strukturu,
- *vybuduj(List<Mesto<T>> prvky, T x1, T y1, T x2, T y2, int cisloBloku)* – metoda, která z prvků předaných parametrem *prvky* vybuduje strukturu, prvky se vloží do bloku s číslem *cisloBloku*,
- *pocetZaznamuVOblasti(ZaznamSouboru<T>[] zaznamy, T x1, T y1, T x2, T y2)* – metoda zjišťující, kolik záznamů ze všech předaných (parametr *zaznamy*) spadá do oblasti vymezené souřadnicemi *x1*, *y1*, *x2* a *y2*, celočíselný výsledek vrací v návratové hodnotě,
- *cisloSousednihoBloku(int cisloBloku, int indexX, int indexY)* – metoda, která vrací číslo sousedního bloku pro blok s číslem *cisloBloku*,
- *zobrazMesta(Control control, bool vcetneSouradnic)* – metoda, která zobrazí všechny prvky na mapový podklad, jemuž odpovídá komponenta předaná parametrem *control* (příznak *vcetneSouradnic* určuje, zda se mají zobrazovat i souřadnice prvků),
- *nakresliRez(T x1, T y1, T x2, T y2, bool plny, Control control)* – pomocná metoda, která zobrazí na mapovém podkladu řezy odpovídající rozdělení prvků ve struktuře,
- *podbarviOblast(int barva, T x1, T y1, T x2, T y2, Control control)* – pomocná metoda, která různými barvami podbarví oblasti, jež vzniknou po vytvoření řezů předchozí metodou,
- *vypisCisloBloku(int cisloBloku, T x1, T y1, Control control)* – pomocná metoda, která vyznačí v každé oblasti příslušné číslo bloku,
- *Dispose()* – destruktor třídy,
- *Dispose(bool disposing)* – destruktor třídy volaný z předchozí metody.

Třída BaseLib

Třída *BaseLib* představuje pomocnou knihovnu s funkcemi pro výpočetní operace a pro práci s textovými soubory. Obsahuje pouze statické metody.

Metody třídy *BaseLib*:

- *NactiTxt(string nazev, out double minZemSirka, out double maxZemSirka, out double minZemDelka, out double maxZemDelka, bool bezDuplicit)* – statická metoda, která načte obsah textového souboru se zadaným názvem, v návratové hodnotě vrací seznam měst nalezených v tomto souboru (příznak *bezDuplicit* určuje, zda v seznamu smí být některé prvky vícekrát či nikoliv), pomocí atributů *minZemSirka*, *maxZemSirka*, *minZemDelka* a *maxZemDelka* informuje o hranicích území, do něhož města patří,
- *PrepocetXnaMapu(Control control, double x, double MinX, double MaxX, double MinY, double MaxY)* – statická metoda, která pro komponentu *control* převede klíčovou hodnotu *x* na mapovou souřadnici,
- *PrepocetYnaMapu(Control control, double y, double MinX, double MaxX, double MinY, double MaxY)* – statická metoda, která pro komponentu *control* převede klíčovou hodnotu y na mapovou souřadnici.

Třída DoubleDelegaty

Třída *DoubleDelegaty* představuje pomocnou třídu s funkcemi, jež umožňují pracovat se souřadnicemi prvků, přestože datové typy těchto souřadnic nejsou předem známé. Třída obsahuje pouze statické metody.

Metody třídy *DoubleDelegaty*:

- *PrevodNaDouble()* – statická metoda vracející delegát, který umožňuje převést hodnotu předem neznámého datového typu na hodnotu typu *double*,
- *Porovnani()* – statická metoda vracející delegát, který umožňuje porovnávání dvou hodnot předem neznámého datového typu tím, že je při porovnávání převede na typ *double*,

- *Polovina()* - statická metoda vracející delegát, který umožňuje vypočítat polovinu ze dvou hodnot předem neznámého datového typu tím, že tyto hodnoty při výpočtu převede na typ *double*.