Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Datové struktury pro uchovávání multidimenzionálních dat

Bc. Libor Dvořák

Diplomová práce

2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra softwarových technologií
Akademický rok: **2008/2009**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Libor DVOŘÁK**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Datové struktury pro uchovávání multidimenzionálních dat**

Zásady pro vypracování:

V úvodní části práce je nutné provést přehled problematiky datových struktur specializovaných na uchovávání multidimenzionálních 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:

1.CORMEN, H. A KOL. Introduction to algorithms. Boston, MIT Press, 2001.

2.LEWIS, H. R., DENENBERG, L. Data structures and their algorithms. Berkley, Adison-Wesley, 1997.

3.GOODRICH, M.T., TAMASSIA, R. Algorithm Design. Hoboken (NJ), John Wiley & Sons, 2002.

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

Datum zadání diplomové práce: **31. října 2008**

Termín odevzdání diplomové práce: **22. května 2009**

Univerzita Pardubice Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

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

V Pardubicích dne 4. listopadu 2008

Prohlašuji:

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

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

V Pardubicích dne 21. 8. 2009

Bc. Libor Dvořák

Poděkování

Děkuji svému vedoucímu práce, doc. Ing. Antonínu Kavičkovi Ph. D., za cenné rady a zapůjčené materiály.

ANOTACE

Práce se zabývá analýzou a implementací vybraných datových struktur uchovávající multidimenzionální data převážně 2-rozměrná.

V naprogramované aplikaci je testována složitost jednotlivých operací a navrhnuto doporučení oblasti použití.

KLÍČOVÁ SLOVA

Datové struktury; multidimenzionální data; *k*-d strom; prioritní vyhledávací strom; quad strom; oktálový strom; grid file

TITLE

Data structures for storage of multidimenzional data

ANNOTATION

This dissertation is about analyse and implementation selected data structures stores multidimenzional data mainly two-dimensional.

The operation complexity is tested in the programmed application and there is propounded recommendation area of application.

KEYWORDS

Data structures; multidimenzional data; *k*-d tree; priority search tree; quadtree; octree; grid file

Obsah

1 Úvod ... 13

1.1 Multidimenzionální data ... 14

1.2 Datové struktury pro uchovávání multidim. dat ... 15

2 Analýza vybraných datových struktur ... 16

2.1 *k*-d strom ... 16

2.1.1 Definice ... 16

2.1.2 Konstrukce ... 16

2.1.3 Najdi minimum ... 23

2.1.4 Odebrání prvku ... 26

2.1.5 Vyhledávání ... 29

2.2 Prioritní vyhledávací strom ... 35

2.2.1 Definice ... 35

2.2.2 Konstrukce ... 35

2.2.3 Přidání prvku ... 40

2.2.4 Odebrání prvku ... 44

2.2.5 Vyhledávání ... 45

2.3 Quad strom ... 48

2.3.1 Konstrukce ... 48

2.3.2 Přidání prvku ... 50

2.3.3 Odebrání prvku ... 53

2.3.4 Využití číslicového stromu ... 55

2.4 Oktálový strom ... 56
2.5 Grid file ... 57
2.5.1 Konstrukce ... 58
3 Návrh tříd ... 61
3.1 Třídy pro binární strom ... 61
3.2 Třídy pro 2-d strom ... 61
3.3 Třídy pro prioritní vyhledávací strom ... 62
3.4 Třídy pro quad strom ... 63
3.5 Třídy pro oktálový strom ... 64
3.6 Třídy pro grid file ... 64
3.6.1 Třídy pro blokový soubor ... 64
3.6.2 Třídy pro grid file ... 65
4 Implementace tříd a GUI aplikace ... 66
5 Testování ... 69
5.1 Testování č. 1 ... 69
5.2 Testování č. 2 ... 70
5.3 Testování č. 3 ... 70
5.4 Testování č. 4 ... 71
5.5 Výsledky testování ... 73
6 Závěr ... 74

Seznam obrázků

Obr. 1) Geografická reprezentace vzorových dat 15

Obr. 2) Konstrukce 2-d stromu - mapa - krok 1..... 18

Obr. 3) Konstrukce 2-d stromu – stromová struktura - krok 1..... 18

Obr. 4) Konstrukce 2-d stromu - mapa - krok 2..... 19

Obr. 5) Konstrukce 2-d stromu – stromová struktura - krok 2..... 19

Obr. 6) Konstrukce 2-d stromu - mapa - krok 3..... 20

Obr. 7) Konstrukce 2-d stromu – stromová struktura - krok 3..... 20

Obr. 8) Konstrukce 2-d stromu – stromová struktura - krok 4..... 21

Obr. 9) Přidání prvku do 2-d stromu – stromová struktura - krok 1..... 22

Obr. 10) Přidání prvku do 2-d stromu – stromová struktura - krok 2..... 23

Obr. 11) Hledání minima ve 2-d stromu – stromová struktura..... 24

Obr. 12) Hledání minima ve 2-d stromu – stromová struktura - krok 1..... 24

Obr. 13) Hledání minima ve 2-d stromu – stromová struktura - krok 2..... 25

Obr. 14) Hledání minima ve 2-d stromu – stromová struktura - krok 3..... 26

Obr. 15) Odebrání prvku z 2-d stromu – stromová struktura..... 27

Obr. 16) Odebrání prvku z 2-d stromu – stromová struktura – krok 1 28

Obr. 17) Odebrání prvku z 2-d stromu – stromová struktura – krok 2 29

Obr. 18) Rozsahové vyhledávání ve 2-d stromu – mapa..... 30

Obr. 19) Vyhledávání ve 2-d stromu - stromová struktura – krok 1 30

Obr. 20) Vyhledávání ve 2-d stromu - stromová struktura - krok 2..... 31

Obr. 21) Vyhledávání ve 2-d stromu - stromová struktura - krok 3..... 32

Obr. 22) Vyhledávání ve 2-d stromu - stromová struktura - krok 4..... 33

Obr. 23) Vyhledávání ve 2-d stromu - stromová struktura - krok 5..... 33

Obr. 24) Konstrukce PST – mapa – krok 1 36

Obr. 25) Konstrukce PST – stromová struktura – krok 1 36

Obr. 26) Konstrukce PST – mapa – krok 2 37

Obr. 28) Konstrukce PST – mapa – krok 3 38

Obr. 27) Konstrukce PST – stromová struktura – krok 2 38

Obr. 29) Konstrukce PST – stromová struktura – krok 3 39

Obr. 30) Vložení do PST A – stromová struktura – krok 2 41

Obr. 31) Vložení do PST A – stromová struktura – krok 2 41

Obr. 32) Vložení do PST A – stromová struktura – krok 4 42

Obr. 33) Vložení do PST B – stromová struktura – krok 2 43

Obr. 34) Vložení do PST B – stromová struktura – krok 3 43

Obr. 35) Odebrání z PST – stromová struktura – krok 1 44

Obr. 36) Odebrání z PST – stromová struktura – krok 2 45

Obr. 37) Vyhledávání v PST – mapa – krok 1 46

Obr. 38) Vyhledávání v PST – stromová struktura – krok 1 46

Obr. 39) Vyhledávání v PST – stromová struktura – krok 1 47

Obr. 40) Vyhledávání v PST – stromová struktura – krok 1 47

Obr. 41) Konstrukce QT – mapa – krok 1 49

Obr. 42) Konstrukce QT – strom– krok 1 49

Obr. 43) Konstrukce QT – mapa – krok 2 49

Obr. 44) Konstrukce QT – mapa – krok 3 50

Obr. 45) Přidání prvku do QT - mapa - krok 1 51

Obr. 46) Přidání prvku do QT - stromová struktura - krok 1 51

Obr. 47) Přidání prvku do QT - mapa - krok 2 52

Obr. 48) Přidání prvku do QT - stromová struktura - krok 2 52

Obr. 49) Odebrání prvku z QT - mapa - krok 1 53

Obr. 50) Odebrání prvku z QT - stromová struktura - krok 1 54

Obr. 51) Odebrání prvku z QT - stromová struktura - krok 2 54

Obr. 52) Odebrání prvku z QT - mapa - krok 2 54

Obr. 53) QT - Využití číslicového stromu 55

Obr. 54) QT - přidání prvku do číslicového stromu 56

Obr. 55) Oktálový strom (zdroj [9]) 57

Obr. 56) Grid file - data - krok 1 58

Obr. 57) Grid file – sektory – krok 1 58

Obr. 58) Grid file - data - krok 2 59

Obr. 59) Grid file – sektory – krok 2 59

Obr. 60) Grid file - data - krok 3 60

Obr. 61) Grid file - data - krok 3 60

Obr. 62) Návrh binárního vyhledávacího stromu 61

Obr. 63) Návrh tříd pro 2-d strom 62

Obr. 64) Návrh tříd pro prioritní vyhledávací strom 62

Obr. 65) Návrh tříd pro quad strom 63

Obr. 66) Návrh tříd pro oktálový strom 64

Obr. 67) Návrh tříd pro grid file 65

Obr. 68) Výsledná testovací GUI aplikace 66

Obr. 69) GUI aplikace - navigační panel 67

Obr. 70) GUI aplikace – rychlé rozsahové hledání 67

Obr. 71) GUI aplikace – výběr struktury 68

Obr. 72) GUI aplikace – rychlé vkládání 68

Obr. 73) Testování - předpoklad testu 4 72

Seznam tabulek

Tab. 1) Vzorová 2-rozměrná geografická data 14

Tab. 2) Vzorová 2-rozměrná upravená data 14

Tab. 3) Konstrukce 2-d stromu - data - krok 1 17

Tab. 4a, 4b) Konstrukce 2-d stromu - data - krok 2 19

Tab. 5a-5d) Konstrukce 2-d stromu - data - krok 3 20

Tab. 6) Konstrukce PST – data – krok 1 36

Tab. 7a,7b) Konstrukce PST – data – krok 2 37

Tab. 8a-8d) Konstrukce PST – data – krok 3 38

Tab. 9) QT - převod do binárních souřadnic 55

Tab. 10) Výsledky testu č. 1 69

Tab. 11) Výsledky testu č. 2 70

Tab. 12) Výsledky testu 3 71

Tab. 13) Výsledky testu 4 72

Seznam grafů

Graf. 1) Výsledky testů 1 a 2 70

Graf. 2) Výsledky testu 3 71

Graf. 3) Výsledky testu 4 72

1 Úvod

V dnešní době je potřeba nástroj, kterým by se efektivně ukládaly a zpracovávaly objekty reálného světa. Tento nástroj si můžeme představit jako relační tabulku, která však není vždy vhodná pro realizaci uživatelských požadavků. Účelem vhodného výběru datových struktur je umožnit efektivní vyhodnocování prostorových dotazů, mezi něž patří např. dotaz na obsah dané oblasti, dotaz na průnik dvou oblastí, dotaz na nalezení nejbližších sousedů objektu apod. V prostředí databází, jejichž data jsou založena na záznamech, je snadné si představit, že záznam je vlastně bodem vícerozměrného prostoru [3].

V této práci se budu zabývat pěti datovými strukturami, které slouží pro uchovávání vícerozměrných dat. Cílem práce je analýza, implementace a doporučení těchto struktur z hlediska aplikace.

Z hlediska zkoumaného prostoru budete toto rozdělení na 1) hlavní 2D a 2) pouze ukázkový 3D prostor.

1. *k*-d strom, quad strom, prioritní vyhledávací strom, grid file
2. oktálový strom

Z hlediska paměťové reprezentace bude toto rozdělení na 1) operační paměť počítače a 2) externí paměť v podobě pevného disku.

1. *k*-d strom, prioritní vyhledávací strom, quad strom, oktálový strom
2. grid file

K analýze všech struktur byla použita velmi malá množina dat, a to množina měst z okolí Pardubic, která obsahuje 15 prvků.

V naimplementované aplikaci jsou pak použity tři typy dat.

1. vzorová (stejná jako v analýze)
2. reálná z ČR (bankomaty České Spořitelny)
3. generovaná

1.1 Multidimenzionální data

Všeobecně jsou výrazem *multidimenzionální data* myšlena data, která obsahují jednoznačné informace o objektu, jeho rozměr (popř. velikost, rozsah, hranice) a/nebo jeho umístění v prostoru. Tyto informace jsou většinou reprezentovány vektory a objekty multimenzionálních dat leží v *k*-dimenzionálním prostoru, kde *k* je přirozené číslo určující počet rozměrů.

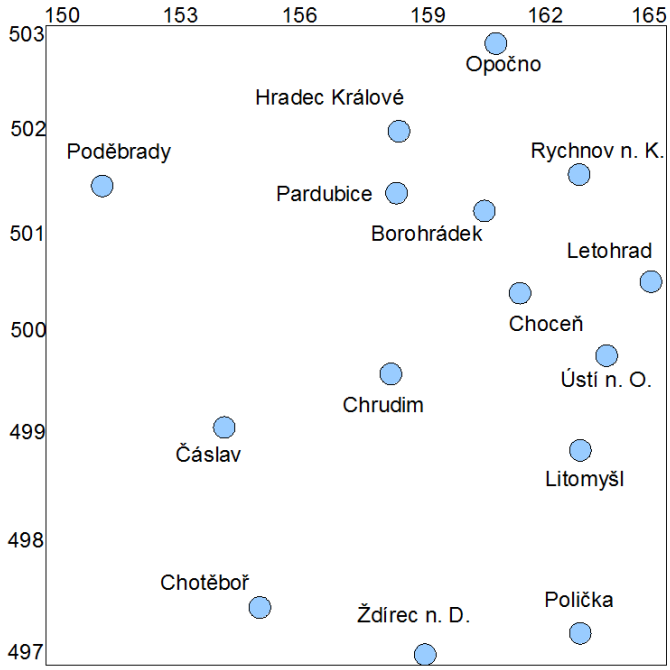
Příklad 2-rozměrných geografických dat je uveden v tabulce Tab. 1 (zdroj vlastní), která obsahuje seznam vybraných měst v okolí Pardubic společně s jejich zeměpisnou šířkou a délkou. Data dále upravíme dle tabulky Tab. 2 pro lepší přehlednost na mapě a budeme používat již označení *x* a *y* pro souřadnice. Na těchto ukázkových datech bude vysvětlen základní princip probíraných datových struktur. Geografické rozložení těchto dat je zobrazeno na obrázku Obr. 1 (zdroj vlastní).

Město	Zem. Šířka[°S]	Zem. Délka[°V]
Borohrádek	50,09	16,09
Čáslav	49,91	15,39
Hradec Králové	50,21	15,83
Choceň	50,00	16,22
Chotěboř	49,72	15,67
Chrudim	49,95	15,79
Letohrad	50,04	16,50
Litomyšl	49,87	16,31
Opočno	50,27	16,11
Pardubice	50,04	15,78
Poděbrady	50,14	15,12
Polička	49,71	16,26
Rychnov nad Kněžnou	50,16	16,28
Ústí nad Orlicí	49,97	16,39
Ždírec nad Doubravou	49,70	15,81

Tab. 1) Vzorová 2-rozměrná geografická data

Město	Y	X
Borohrádek	500,9	160,9
Čáslav	499,1	153,9
Hradec Králové	502,1	158,3
Choceň	500,0	162,2
Chotěboř	497,2	156,7
Chrudim	499,5	157,9
Letohrad	500,4	165,0
Litomyšl	498,7	163,1
Opočno	502,7	161,1
Pardubice	500,4	157,8
Poděbrady	501,4	151,2
Polička	497,1	162,6
Rychnov nad Kněžnou	501,6	162,8
Ústí nad Orlicí	499,7	163,9
Ždírec nad Doubravou	497,0	158,1

Tab. 2) Vzorová 2-rozměrná upravená data



Obr. 1) Geografická reprezentace vzorových dat

1.2 Datové struktury pro uchovávání multidim. dat

Datové struktury pro uchovávání mulitidimenzionálních dat jsou nástrojem podporující vyhledávání a modifikování (vkládání, změnu, vymazání) v multidimenzionálních datech.

Datové struktury pro uchovávání mulitidimenzionálních dat jsou také v některých literaturách známy jako multidimenzionální přístupové metody, prostorové přístupové metody nebo také prostorové indexové struktury.

Je také důležité rozlišovat uchovávání bodových a prostorových multidimenzionálních dat. Bodová multidimenzionální data jsou body ve dvou nebo více dimenzích. Prostorová multidimenzionální data jsou prostorové objekty jako přímky, polygony nebo vícedimenzionální mnohostěny.

Předmětem zkoumání této práce jsou právě datové struktury pro uchovávání bodových multidimenzionálních dat s hlavním zaměřením na 2D prostor.

2 Analýza vybraných datových struktur

V této kapitole provedu teoretický rozbor vybraných pěti datových struktur. Představeny budou základní operace konstrukce struktury, přidání prvku, odebrání prvku a vyhledávaní prvku.

Většina operací je doplněna algoritmem, který je popsán mým vlastním zjednosušeným pseudokódem vycházejícím z programovacího jazyka C#. Pseudokód obsahuje klíčová slova a pojmenované operace v anglickém jazyce a jejich význam je samovysvětlující. V některých případech je doplněn komentář následovaný za značkou „//“.

2.1 *k*-d strom

K-d strom slouží k ukládání bodových *k*-dimenzionálních dat. *K*-d strom je vlastně binární strom, kde každý jeho prvek je *k*-dimenzionální bod. Dělení *k*D prostoru se provádí podle vkládaných dat. V 1D je tímto dělícím prvek bod, ve 2D přímka a ve 3D plocha. Dělení *k*D prostoru se provádí střídavě podle všech jeho *k* souřadnic. V této kapitole bylo použito zdrojů [1], [7], [10].

2.1.1 Definice

- každý prvek *k*-d stromu obsahuje *k*-dimenzionální záznam a má přidělen tzv. diskriminátor (rozlišovací znak) $j \in (0,1,..,k-1)$
- pro každý prvek s klíčem x a s diskriminátorem j platí : jakýkoliv prvek v levém podstromu s klíčem y splňuje podmínku $y_j < x_j$ a jakýkoliv prvek v pravém podstromu s klíčem y splňuje podmínku $y_j \geq x_j$
- kořenový prvek má výšku stromu v rovnu 0 a diskriminátor j roven 0, všechny prvky ve výšce v mají diskriminátor roven (v mod k)
- každý prvek, který není listem, dělí prostor na dva podprostory

2.1.2 Konstrukce

Nejdříve si zvolíme počáteční souřadnici *s*, podle které budeme prostor dělit. Zadaná data seřadíme a prohledáme podle zvolené souřadnice, nalezneme medián a nazveme ho klíčem *a*. Tento prvek bude tvořit kořen stromu *r*.

Následně všechny prvky mající zvolenou souřadnici *s* menší než klíč *a* budou tvořit levý podstrom kořenu *r*. Zcela intuitivně budou zbylé prvky zařazeny do pravého podstromu.

Rekurzivně procházíme levý a pravý podstrom aktuálního prvku a cyklicky měníme souřadnice, podle kterých se prvky dělí do dalších podstromů.

Příklad

Na upravených vzorových datech si nyní ukážeme konkrétní postup při konstrukci 2-d stromu z předem známých dat.

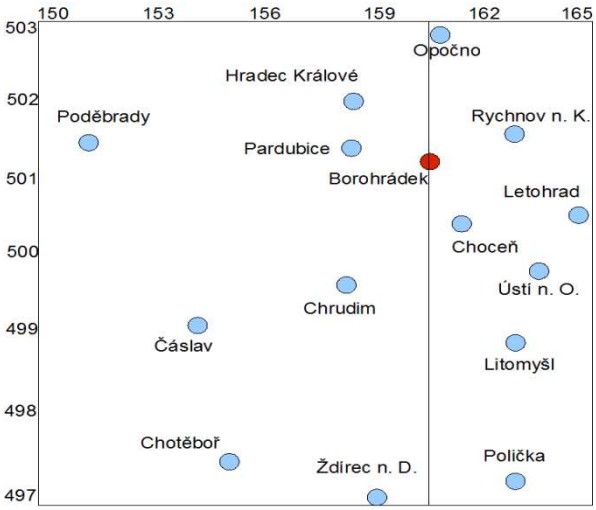
Krok 1

Zvolíme počáteční souřadnici *x*. Zadaná data seřadíme podle této souřadnice a nalezneme medián, tím je město Borohrádek. Tento prvek bude tvořit kořen stromu. Tuto situaci zachycuje tabulka Tab. 3 (zdroj vlastní).

Město	Y	X
Poděbrady	501,4	151,2
Čáslav	499,1	153,9
Chotěboř	497,2	156,7
Pardubice	500,4	157,8
Chrudim	499,5	157,9
Ždírec nad Doubravou	497,0	158,1
Hradec Králové	502,1	158,3
Borohrádek	500,9	160,9
Opočno	502,7	161,1
Choceň	500,0	162,2
Polička	497,1	162,6
Rychnov nad Kněžnou	501,6	162,8
Litomyšl	498,7	163,1
Ústí nad Orlicí	499,7	163,9
Letohrad	500,4	165,0

Tab. 3) Konstrukce 2-d stromu - data - krok 1

Na obrázku Obr. 12 (zdroj vlastní) můžeme vidět grafické dělení prostoru podle zvolené souřadnice *x* a nalezeného mediánu.



Obr. 2) Konstrukce 2-d stromu - mapa - krok 1

Nalezený medián je vložen do prázdné stromové struktury jako kořen stromu (Obr. 3)

Obr. 3) Konstrukce 2-d stromu – stromová struktura - krok 1

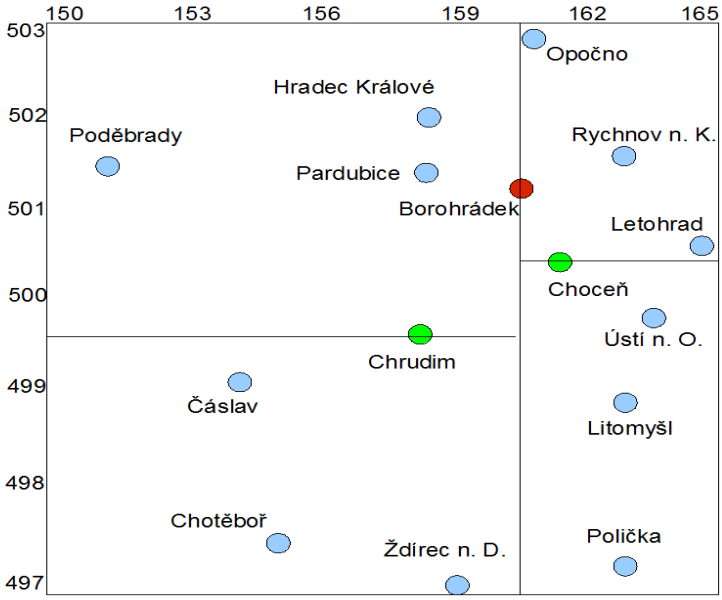
Krok 2

Prvky, které patří do levého podstromu kořene, zařadíme do nové množiny. Seřadíme je podle *y*-ové souřadnice a nalezneme medián. Stejný postup opakujeme s prvky pravého podstromu (Tab. 4). Výsledné rozdělení prostoru je vidět na obrázku Obr. 4 a výsledná stromová struktura na obrázku Obr. 5.

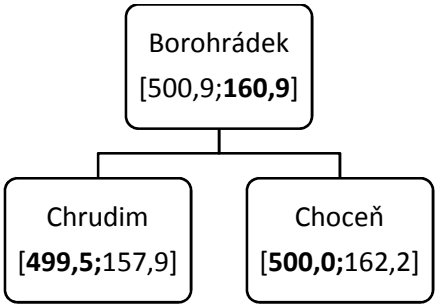
Město	Y	X
Ždírec nad Doubravou	497,0	158,1
Chotěboř	497,2	156,7
Čáslav	499,1	153,9
Chrudim	499,5	157,9
Pardubice	500,4	157,8
Poděbrady	501,4	151,2
Hradec Králové	502,1	158,3

Město	Y	X
Polička	497,1	162,6
Litomyšl	498,7	163,1
Ústí nad Orlicí	499,7	163,9
Choceň	500,0	162,2
Letohrad	500,4	165,0
Rychnov nad Kněžnou	501,6	162,8
Opočno	502,7	161,1

Tab. 4a, 4b) Konstrukce 2-d stromu - data - krok 2



Obr. 4) Konstrukce 2-d stromu - mapa - krok 2



Obr. 5) Konstrukce 2-d stromu – stromová struktura - krok 2

Krok 3

Dělení nyní probíhá zcela intuitivně podle *x*-ové souřadnice. Výsledky tohoto kroku jsou zobrazeny v tabulce Tab. 5 a obrázkách Obr. 6 a Obr. 7.

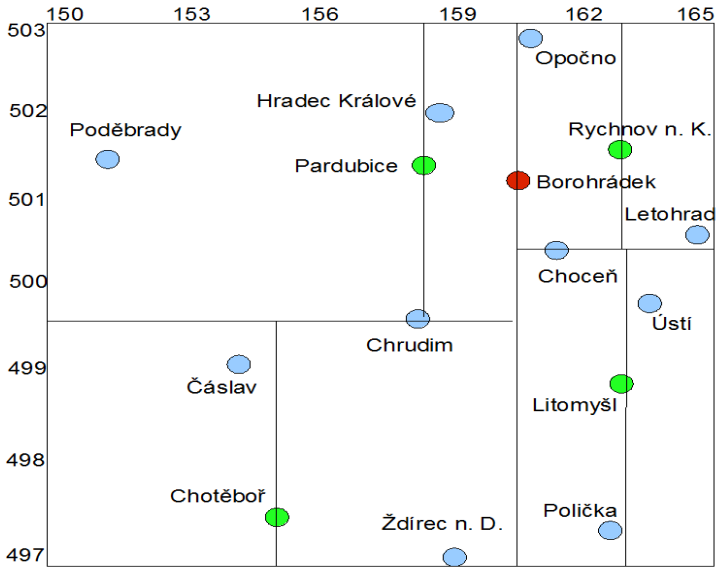
Město	Y	X
Čáslav	499,1	153,9
Chotěboř	497,2	156,7
Ždírec nad Doubravou	497,0	158,1

Město	Y	X
Polička	497,1	162,6
Litomyšl	498,7	163,1
Ústí nad Orlicí	499,7	163,9

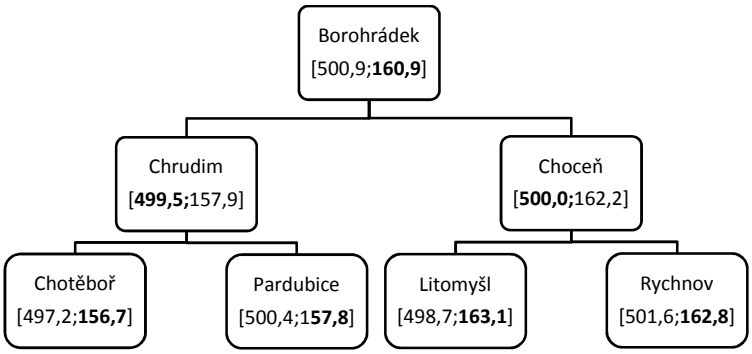
Město	Y	X
Poděbrady	501,4	151,2
Pardubice	500,4	157,8
Hradec Králové	502,1	158,3

Město	Y	X
Opočno	502,7	161,1
Rychnov nad Kněžnou	501,6	162,8
Letohrad	500,4	165,0

Tab. 5a-5d) Konstrukce 2-d stromu - data - krok 3



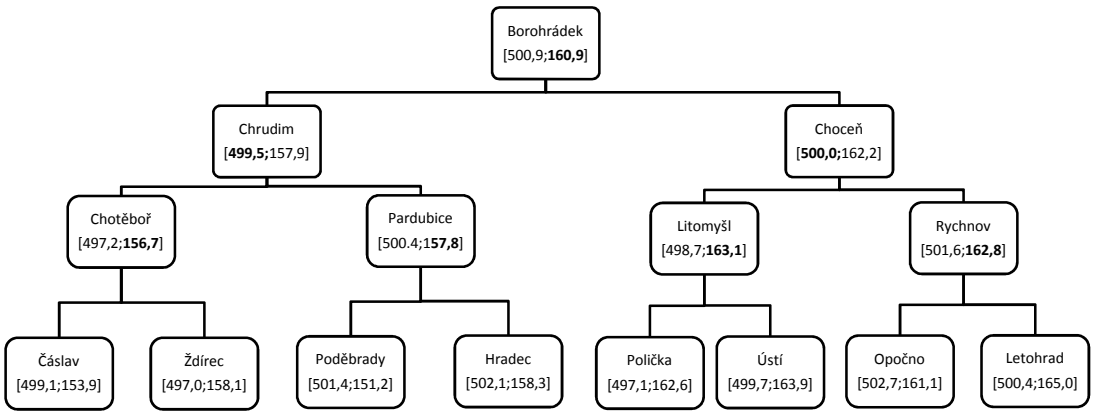
Obr. 6) Konstrukce 2-d stromu - mapa - krok 3



Obr. 7) Konstrukce 2-d stromu – stromová struktura - krok 3

Krok 4

Nyní každý podstrom obsahuje pouze jeden prvkem, tedy dělení zde končí a zbylé prvky jsou do stromu vloženy jako listy. Konečnou stromovou strukturu zobrazuje obrázek Obr. 8.



Obr. 8) Konstrukce 2-d stromu – stromová struktura - krok 4

Algoritmus

```
Root kdConstruction(data,diskriminator)
{
  //získání mediánu
        median = GetMedian(data, diskriminator);
        node = median;
        data.Remove(median);
        //rozdělení dat na dvě poloviny
        LH = GetLeftHalf(data);
        RH = GetRightHalf(data);
        //rekurzivní volání konstrukce pro podstromy
        TR = kdConstruction(LH, !diskriminator);
        TL = kdConstruction(RH, !diskriminator);
        node.AddLeftChild(TL);
        node.AddRightChild(TR);
        return node;
}
```

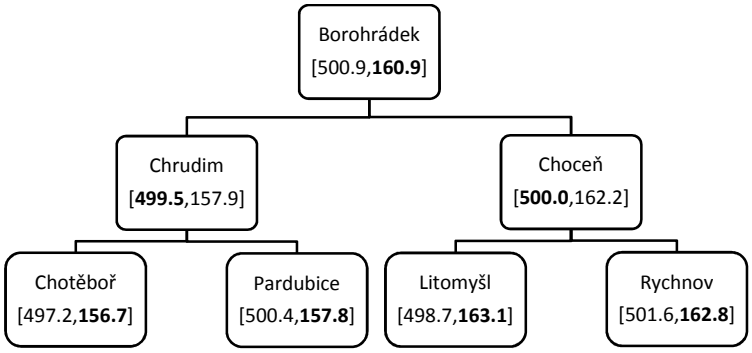
2.1.2.1 Přidání prvku

Nově přidávaný prvek *p* traverzuje stromem dolů na příslušnou pozici. Na každé úrovní stromu je jeho klíč *a*, který je daný aktuální hodnotou diskriminátoru, porovnán s klíčem *a* aktuálního prvku *q*. Dle výsledku porovnání rekurzivně pokračujeme levým nebo pravým synem prvku *q*. Vkládaný prvek je vždy vložen jako nový list stromové struktury.

Příklad

Krok 1

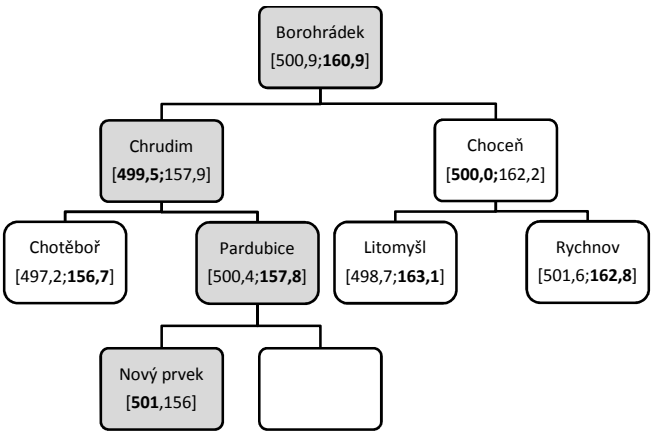
Mějme výchozí situaci na obrázku Obr. 9. Přidáváme prvek *Nový prvek* se souřadnicemi např. [501;156].



Obr. 9) Přidání prvku do 2-d stromu – stromová struktura - krok 1

Krok 2

Porovnáváním nejdříve *x*-ové souřadnice na 1. úrovni stromu, dále *y*-ové na 2. úrovni a naposled opět *x*-ové na 3. úrovni, přidávaný prvek protraverzuje na příslušnou pozici nového listu. Na obrázku Obr. 10 jsou vyznačeny prvky stromu, které byly při vkládání navštíveny.



Obr. 10) Přidání prvku do 2-d stromu – stromová struktura - krok 2

Algoritmus

```
kdNode kdInsert(kdNode start, kdNode input, int d)
{
        if(start.data == input.data) return // duplicitní prvek
//pokračuj levou větví
        else if (input.data.d < start.data.d)
                start.LeftChild = insert(start.LeftChild, input, d mod k)
        else
//pokračuj pravou větví
start.RightChild = insert(start.RightChild, input, d mod k)
        return start
}
```

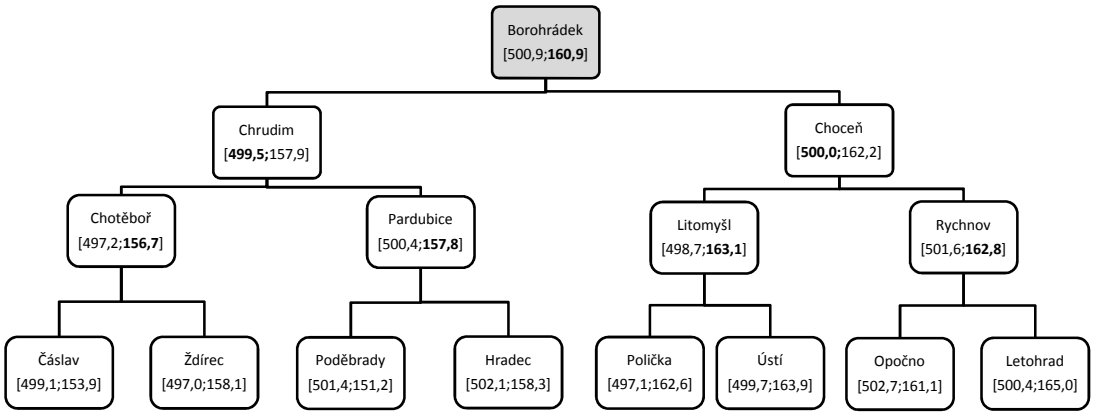
2.1.3 Najdi minimum

Ještě než popíši operaci *odebrání prvku* (kap 2.1.4), musím zde představit pomocnou operaci *najdi minimum*, která je v operaci *odebrání* zásadní.

Pro zahájení hledání je potřeba počáteční prvek *p*, diskriminátor podle kterého budeme hledat minimum *dmin* a diskriminátor prvku, na kterém se nacházíme *d*. Princip si ukážeme na konkrétním příkladě.

Příklad

Mějme výchozí stromovou strukturu danou obrázkem Obr. 11. Počátečním prvek *p* je město Borohrádek, jeho diskriminátor *d* je 0, hledat budeme minimum v hodnotách *x*-ových souřadnic, tedy *dmin* je roven 0 také.



Obr. 11) Hledání minima ve 2-d stromu – stromová struktura

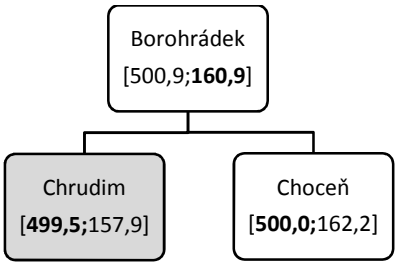
Definujme si nejdříve dvě podmínky, s kterými budeme dále pracovat.

Podmínka 1: $dmin = d$

Podmínka 2: $p.leftChild \neq null$

Krok 1

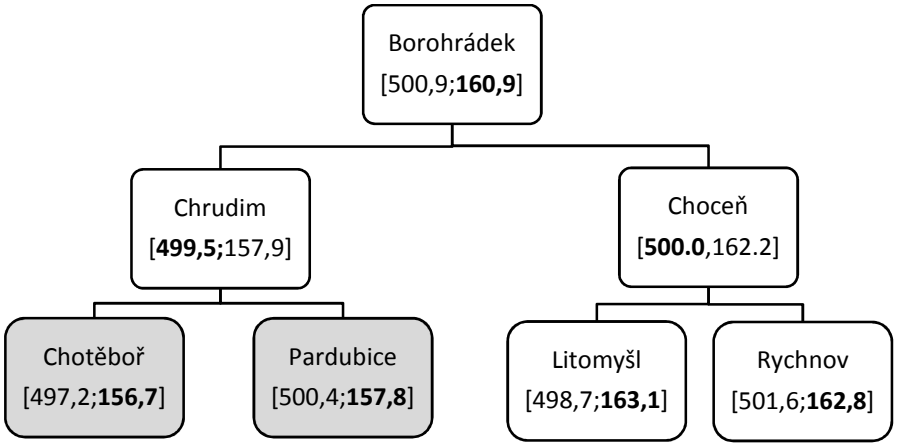
Porovnáváme hodnotu diskriminátoru *dmin* aktuálního prvku p s hodnotou zadaného diskriminátoru *d*. Daná podmínka 1 platí, porovnáme tedy podmínku 2, zda má prvek p levého syna. Podmínka 2 je také splněna a přecházíme na levého syna prvku p. Část stromové struktury potřebnou pro tento krok zobrazuje obrázek Obr. 12.



Obr. 12) Hledání minima ve 2-d stromu – stromová struktura - krok 1

Krok 2

Aktuálním prvek *p* se stává město Chrudim. Nesplňuje podmínku 1, hodnotu příslušící zadanému diskriminátoru *d* si uložíme do pomocné proměnné min = [157,9] a v dalším kroku prohledáváme obě větve prvku *p*. Výřez stromové struktury zachycuje obrázek Obr. 13.

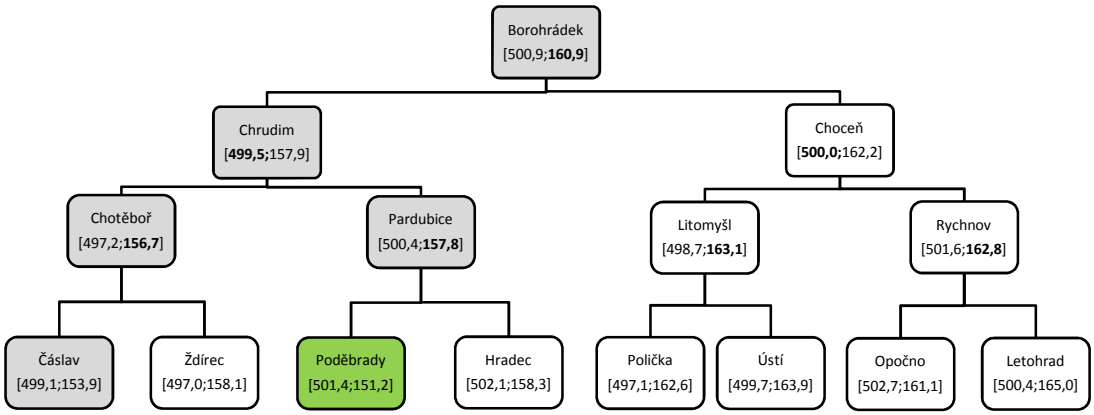


Obr. 13) Hledání minima ve 2-d stromu – stromová struktura - krok 2

Krok 3

Projdeme nejprve levou větev a aktuálním prvkem *p* je město Chotěboř. Splňuje podmínku 1 i 2. Pokračujeme tedy jeho levým synem, kterým je město Čáslav. Tento prvek nesplňuje podmínku 1 a je listem, tudíž hledání v této větvi končí. Hodnotu proměnné min nahradíme aktuálně nižší nalezenou, tedy min = [153,9].

Pokračujeme na pravou větev, kde se aktuálním prvkem *p* stává město Pardubice. Prvek *p* splňuje podmínku 1, postupujeme tedy levou jeho větví na prvek Poděbrady, který se stává aktuálním prvkem *p*. Prvek *p* nesplňuje podmínku 1 a je zároveň listem. Hledání zde končí a proměnná min je přepsána nalezenou nižší hodnotou, tedy nalezené minimum je rovno 151,2. Výslednou situaci, navštívené prvky a nalezený prvek s minimální hodnotou v *x*-ové souřadnici zachycuje obrázek Obr. 14.



Obr. 14) Hledání minima ve 2-d stromu – stromová struktura - krok 3

Algoritmus

```
kdFindMin(p, dmin, d)
{
        //porovnej diskriminátory
        if (dmin == d) then
        {
                if (p.leftChild == null) return p.data
                //pokračuj levou větví
                else return kdFindMin(p.leftChild, dmin, d mod k)
        }
        else
        {
                d = d mod k
                //projdi obě větve
                return min(p.data, kdFindMin(p.leftChild, dmin, d),
        kdFindMin(p.rightChild,dmin, d)
        }
}
```

2.1.4 Odebrání prvku

Odebrání prvku je u *k*-d stromu poněkud složitější. Využívá předešlou operaci *hledání minima* (kap. 2.1.3).

Princip

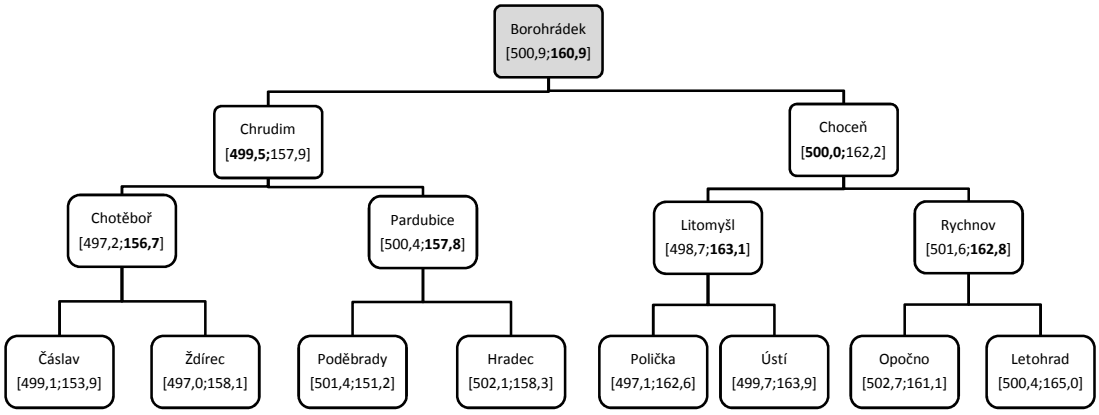
- chceme odstranit prvek *p*, který má souřadnice *x*, *y*
- nalezneme prvek *p*
    - pokud je prvek *p* listem je nahrazen hodnotou NULL

- jinak nalezneme náhradní prvek *q* se souřadnicemi *a, b*
- souřadnice prvku *p* jsou nahrazeny souřadnicemi prvku *q*
- rekurzivně odebereme prvek *p* s novými souřadnicemi

- k nalezení prvku *q* použijeme operaci *hledání minima*

Příklad

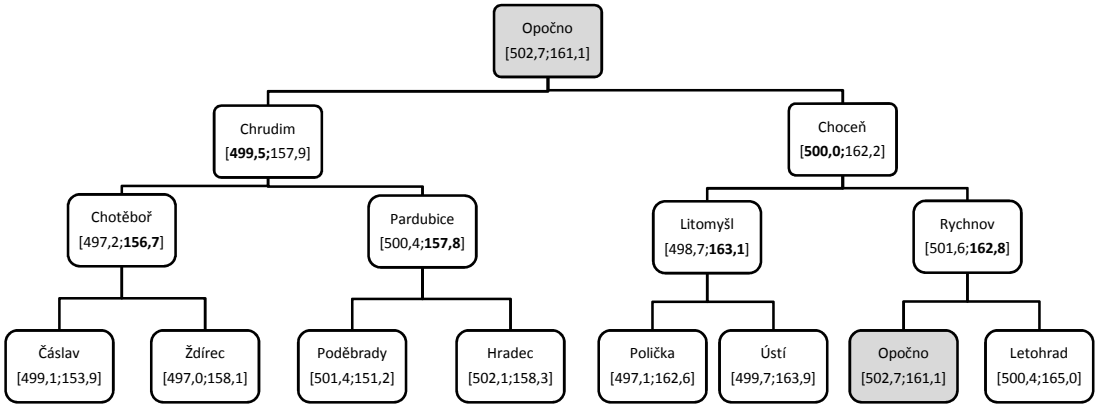
Mějme výchozí situaci na obrázku Obr. 15 a požadujeme odstranit prvek se souřadnicemi [500.9,160.9].



Obr. 15) Odebrání prvku z 2-d stromu – stromová struktura

Krok 1

Nalezený prvek *p*, kterým je město Borohrádek, není listem. Aplikujeme na něj tedy operaci *hledání minima*. Výsledkem hledání náhradního prvku *q* je město Opočno. Následuje výměna nejen souřadnic, ale i veškerých informací, které prvky nesou. Situaci po tomto kroku vystihuje obrázek



Obr. 16) Odebrání prvku z 2-d stromu – stromová struktura – krok 1

Krok 2

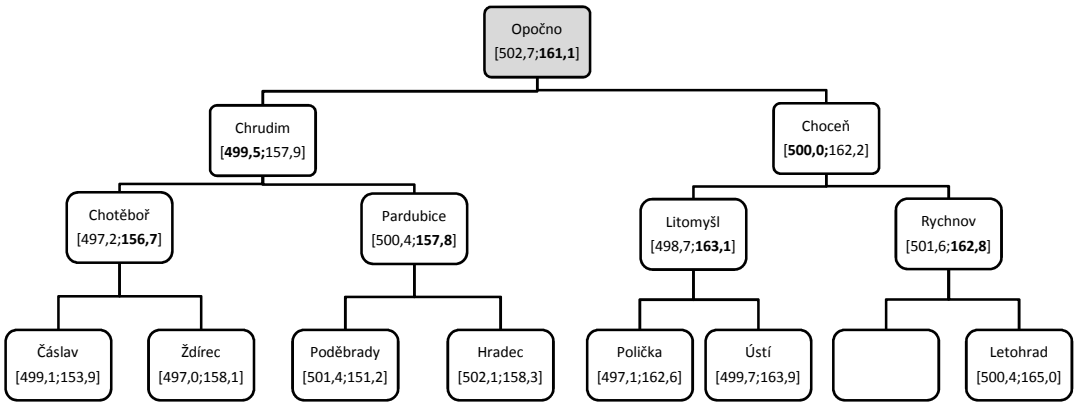
Dále rekurzivně voláme metodu o*dstranit prvek* na pravého syna (město Choceň) aktuálního prvku *p* s novými souřadnicemi získanými od prvku *q*.

Výchozí prvek hledání *v* je město Choceň a hledáme prvek *p*, který má původní hledané souřadnice [502,7;161,1]. Porovnáme *y*-ovou souřadnici prvku *v* (500.0) s hledanou hodnotou (protože prvek *v* má diskriminátor *y*). Hledaná hodnota je větší, hledaný prvek *p* bude tedy ležet napravo od prvku *v*. Přejdeme tedy na pravého syna prvku *v*, kterým je město Rychnov a který se automaticky stane výchozím prvek *v*.

Prvek *v* má diskriminátor *x*, porovnáme tedy hledanou hodnotu *x* (161.1) s *x*-ovou hodnotou (162.8) prvku *v*. Hledaná hodnota je menší, hledaný prvek bude tedy ležet vlevo od prvku *v*.

Přecházíme na prvek Opočno. Výsledkem porovnání hodnot jsme našli hledaný prvek *p* k odstranění se souřadnicemi [502.7,161.1]. Jelikož je prvek listem, je nahrazen hodnotou NULL. Pokud by nalezený prvek nebyl listem, hledal by se opět náhradní prvek *q* pomocí operace *hledání minima*.

Výsledkem je tedy obr. , kde byl odstraněn prvek Borohrádek a na jeho kořenové místo se přesunul prvek Opočno.



Obr. 17) Odebrání prvku z 2-d stromu – stromová struktura – krok 2

Algoritmus

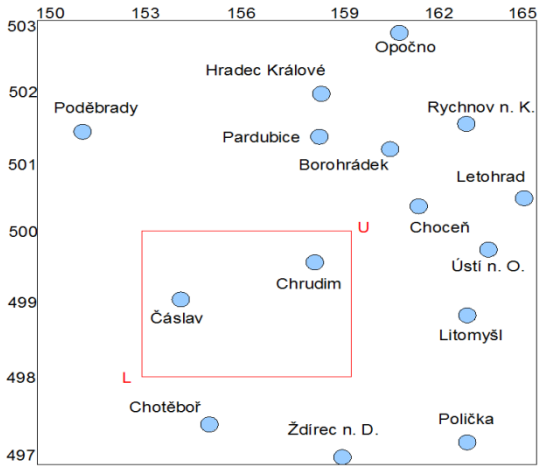
```
//výchozí prvek v, hledaný prvek h, diskriminátor d
kdNode kdRemove(v, h, d)
{
        //najdi prvek k odstranění
        if (h.d < v.d) v.leftChild = kdRemove(v.leftChild, h, d mod k)
        if (h.d > v.d) v.rightChild = kdRemove(v.rightChild, h, d mod k)
        else
        {
                If (v.isLeaf) return NULL;
                //hledej minimum
                If (v.rightChild != null)
                        v.data = kdFindMin(v.rightChild, d, d mod k)
                Else
                {
                        v.data = kdFindMin(v.leftChild, d , d mod k)
                        v.leftChild = null
                }
                v.rightChild = kdRemove(v.rightChild, v.data, d mod k)
        }
        return v
}
```

2.1.5 Vyhledávání

Představíme si zde tzv. *rozsahové vyhledávání (Range searching).* Ve dvourozměrném prostoru bude tímto rozsahem obdélník určený pouze dvěma body. Tyto body si označíme *L* (z ang. lower) a *U* (z ang. upper). Princip si vysvětlíme na konkrétním příkladu.

Příklad

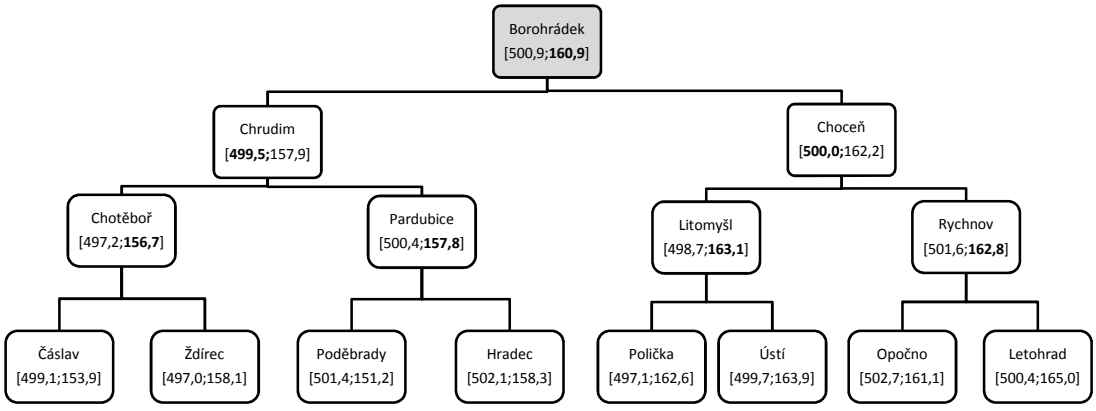
Mějme základní obrázek Obr. 18 s požadovaných rozsahem označeným červeným obdélníkem a vyznačené body L[498;153] a U[500;159].



Obr. 18) Rozsahové vyhledávání ve 2-d stromu – mapa

Krok 1

Všechny prvky už jsou ve stromě vloženy dle obrázku Obr. 19, začínáme tedy kořenem stromu, tím je město Borohrádek. K vyhledávání budeme potřebovat ještě vstupní diskriminátor, který je v případě kořene roven nule (dělení začíná podle osy *x*), a nějakou doplňující operaci *o*, která bude nalezené prvky zpracovávat.



Obr. 19) Vyhledávání ve 2-d stromu - stromová struktura – krok 1

Začíná porovnání všech hodnot klíčů aktuálního prvku p z hodnotami ohraničujících bodů *L* a *U*.

Podmínka 1 :

$$L_x \leq P_x \leq U_x \ (153 \leq 160{,}9 \leq 159)$$

$$L_y \leq P_y \leq U_y \ (498 \leq 500{,}9 \leq 500)$$

Pokud by vyhovovaly obě podmínky, byl by prvek *p* z požadovaného rozsahu a byl by předán operaci *o* ke zpracování.

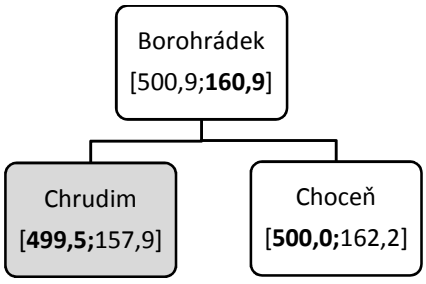
Následuje porovnání hodnoty klíče vyhovující pouze příslušnému diskriminátoru *d* (v tomto případě hodnota klíče *x*).

Podmínka 2: $L_d < P_d (153 < 160{,}9)$

Podmínka 3: $P_d \leq U_d (160{,}9 \leq 159)$

Krok 2

Prvek *p* vyhovuje podmínce 2 a proto se ve stromové struktuře přesouváme pouze na levého syna. Obrázek Obr. 20 zachycuje pouze část stromové struktury podstatnou pro tento krok.



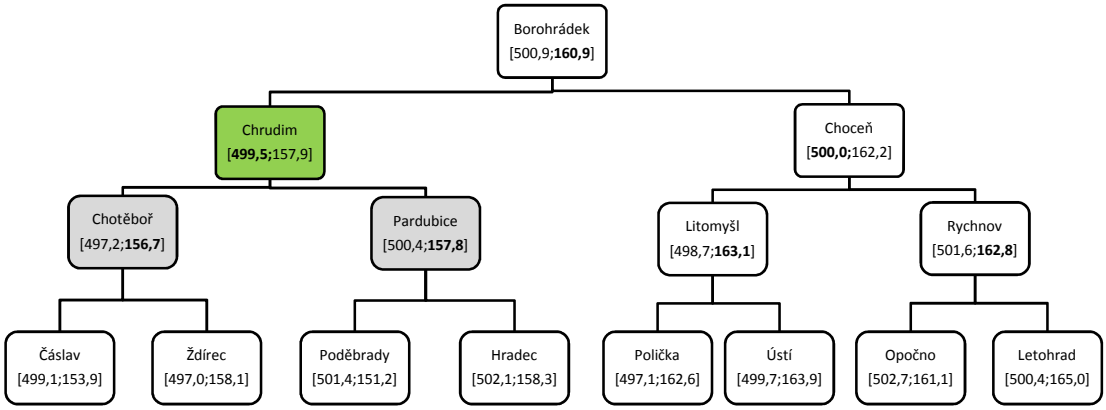
Obr. 20) Vyhledávání ve 2-d stromu - stromová struktura - krok 2

Krok 3

Prvkem *p* se stává město Chrudim. Vyhovuje podmínce 1, tedy je hledaným prvkem z rozsahu a je předán operaci *o* ke zpracování. Prvek *p* vyhovuje i oběma následujícím podmínkám a prohledávání pokračuje v obou větvích (Obr. 21).

Věta

Pokud prvek *p* vyhovuje podmínce 1, pak současně vyhovuje oběma podmínkám 2 a 3, a prohledávání pokračuje v obou jeho větvích.



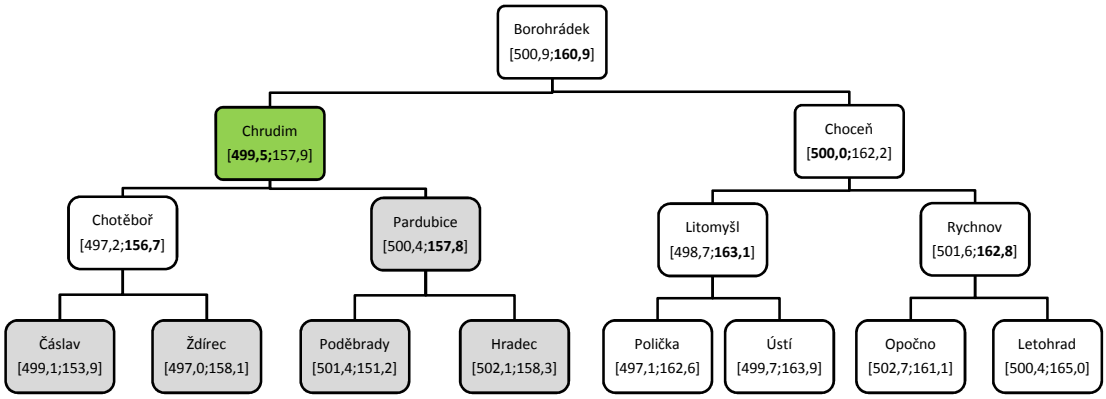
Obr. 21) Vyhledávání ve 2-d stromu - stromová struktura - krok 3

Krok 4

Projdeme nejdříve levou větev, tedy prvkem *p* se stává město Chotěboř, který nevyhovuje podmínce 1, není tedy hledaným prvkem. Prvek *p* ale vyhovuje podmínkám 2 a 3, tedy prohledávání pokračuje v kroku 5 v obou jeho větvích.

U pravé větve se prvkem *p* stává město Pardubice, nevyhovuje podmínce 1, ale vyhovuje podmínkám 2 a 3. V kroku 5 budeme prohledávat obě jeho větve.

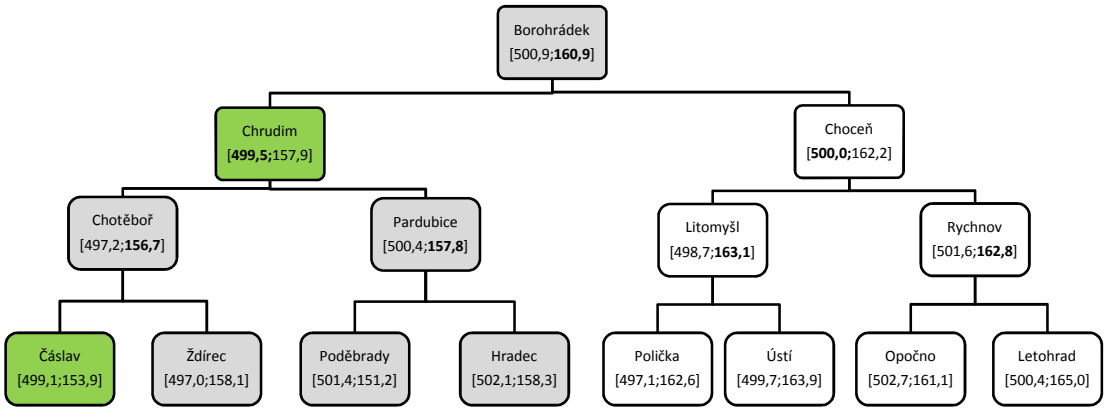
Následující situaci procházení stromové struktury zachycuje obrázek Obr. 22.



Obr. 22) Vyhledávání ve 2-d stromu - stromová struktura - krok 4

Krok 5

V posledním kroku porovnáváme čtyři prvky pouze již na podmínku 1. Vyhovuje pouze město Čáslav a je posledním hledaným prvek z rozsahu. V pěti krocích jsme prošli celou levou větev 2-d stromu a nalezli dva prvky vyhovující požadovanému rozsahu zadaného dvěma body *L* a *U*. Výsledek hledání, nalezené a navštívené prvky zachycuje obrázek Obr. 23



Obr. 23) Vyhledávání ve 2-d stromu - stromová struktura - krok 5

Algoritmus

```
kdRangeSearch(L, U, d, p, o)
{
        //zda prvek vyhovuje rozsahu
        if (L_i ≤ Key_i(p) ≤ U_i) then o(p)..... for i = 0,...,k − 1
        K = Key_d(p)
        //projdi levou větev
        if (L_d < K) then kdRangeSearch(L, U, d + 1 mod k, p.leftChild, o)
        //projdi pravou větev
        if (K ≤ U_d) then kdRangeSearch(L, U, d + 1 mod k, p.rightChild, o)
}
```

2.1.5.1 Dynamický přístup

Pokud operace *vyhledávání* nebude prioritní operací v příslušné aplikaci a operace *přidej* nebo *odeber* budou velmi časté, mluvíme zde o tzv. *dynamickém přístupu*. *K*-d strom se bude stávat čím dál více nevyváženým. Poté každá operace bude trvat mnohem delší dobu a použití aplikace bude méně efektivní.

Otázkou je tedy jestli zvolit nějakou techniku pro udržení *k*-d stromu ve vyváženém stavu po každé operaci nebo provést kompletní přebudování stromu. Klasické stromové rotace jsou zde nepoužitelné, protože prvky jsou řazeny ve více dimenzích. Zbývá tedy druhá možnost, a to sledování nevyváženosti *k*-d stromu a v případě překročení nad zvolenou mez, provést kompletní *prohlídku* struktury. Na vytvořenou množinu všech prvků aplikovat operaci *konstrukce* (kap. 2.1.2) a vytvořit nový vyvážený *k*-d strom.

Toto platí pro všechny následující stromové datové struktury a proto již otázku dynamického přístupu nebudu v dalších kapitolách zmiňovat.

2.2 Prioritní vyhledávací strom

Prioritní vyhledávací strom (z angl. Priority Search Tree dále jen PST) je kombinací binárního vyhledávacího stromu[1] na *x*-ové souřadnici a binární haldy[2] na *y*-ové souřadnici. PST slouží pro uchovávání dvourozměrných dat. V této kapitole bylo využito hlavně zdrojů [1], [2], [5] a [12].

2.2.1 Definice

- každý prvek má maximálně dva syny
- levý podstrom uzlu obsahuje pouze klíče *x*-ové souřadnice menší než je klíč mediánu uchovaný v tomto uzlu
- pravý podstrom uzlu obsahuje pouze klíče *x*-ové souřadnice větší nebo rovné klíči mediánu tohoto uzlu
- klíč *y*-ové souřadnice uzlu je vždy větší nebo roven klíči jeho podstromů

2.2.2 Konstrukce

Varianta A

Data seřadíme podle *y*-ové souřadnice a kořenem zvolíme prvek *p* s nejvyšší hodnotou. Zbytek prvků seřadíme podle *x*-ové souřadnice a rozdělíme na polovinu. Hodnota prvku, který je mediánem v tomto uspořádání, se uloží do prvku *p*. Tyto dvě poloviny se stanou levým a pravým podstromem prvku *p*. Tento postup se rekurzivně opakuje hledáním prvku s nejvyšší hodnotou *y* v každé polovině, dokud obsahuje alespoň jeden prvek.

Příklad

Krok 1

Data seřadíme podle *y*-ové souřadnice a vybereme prvek s největší hodnotou. Tím je město Opočno, které bude tvořit kořen stromu (Tab. 6).

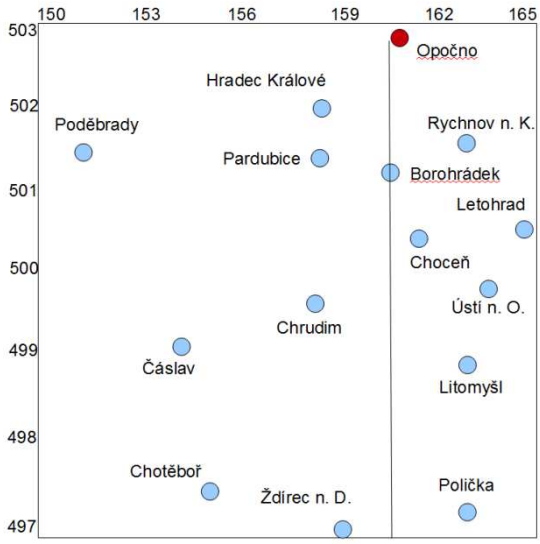
[1] http://cs.wikipedia.org/wiki/Binární_vyhledávací_strom

[2] http://cs.wikipedia.org/wiki/Binární_halda

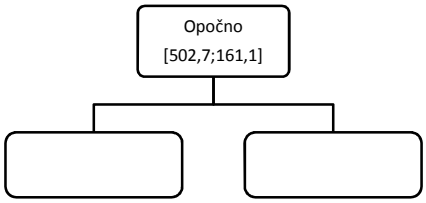
Město	Y	X
Ždírec nad Doubravou	497,0	158,1
Polička	497,1	162,6
Chotěboř	497,2	156,7
Litomyšl	498,7	163,1
Čáslav	499,1	153,9
Chrudim	499,5	157,9
Ústí nad Orlicí	499,7	163,9
Choceň	500,0	162,2
Letohrad	500,4	165,0
Pardubice	500,4	157,8
Borohrádek	500,9	160,9
Poděbrady	501,4	151,2
Rychnov nad Kněžnou	501,6	162,8
Hradec Králové	502,1	158,3
Opočno	502,7	161,1

Tab. 6) Konstrukce PST – data – krok 1

Zbylá data seřadíme podle *x*-ové souřadnice, nalezneme medián a data rozdělíme na polovinu. Mediánem je město Borohrádek (Obr. 24).



Obr. 24) Konstrukce PST – mapa – krok 1



Obr. 25) Konstrukce PST – stromová struktura – krok 1

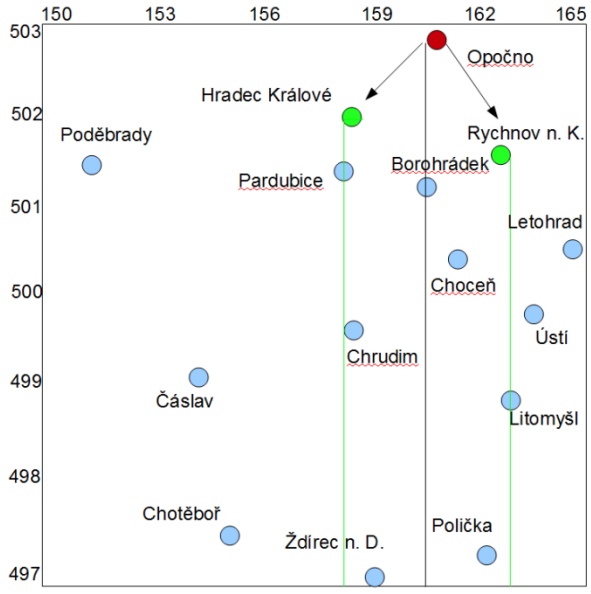
Krok 2

Data v každém podstromu seřadíme podle *y*-ové souřadnice a vybereme prvky s nejvyšší hodnotou. Těmi jsou města Hradec Králové a Rychnov nad Kněžnou (Tab. 7).

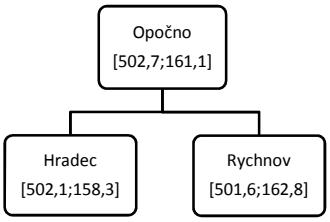
Město	Y	X
Ždírec nad Doubravou	497,0	158,1
Chotěboř	497,2	156,7
Čáslav	499,1	153,9
Chrudim	499,5	157,9
Pardubice	500,4	157,8
Poděbrady	501,4	151,2
Hradec Králové	502,1	158,3

Město	Y	X
Polička	497,1	162,6
Litomyšl	498,7	163,1
Ústí nad Orlicí	499,7	163,9
Choceň	500,0	162,2
Letohrad	500,4	165,0
Borohrádek	500,9	160,9
Rychnov nad Kněžnou	501,6	162,8

Tab. 7a,7b) Konstrukce PST – data – krok 2



Obr. 26) Konstrukce PST – mapa – krok 2



Obr. 27) Konstrukce PST – stromová struktura – krok 2

Krok 3

Data v každém ze 4 podstromů seřadíme podle *y*-ové souřadnice a vybereme prvky s nejvyšší hodnotou. Těmi jsou města Poděbrady, Pardubice, Borohrádek a Letohrad.

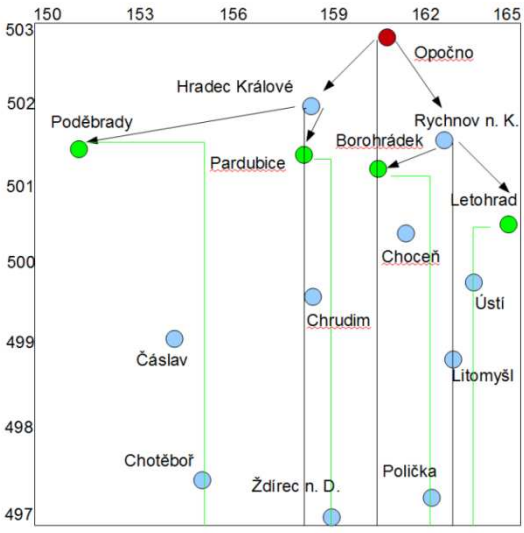
Město	Y	X
Chotěboř	497,2	156,7
Čáslav	499,1	153,9
Poděbrady	501,4	151,2

Město	Y	X
Polička	497,1	162,6
Choceň	500,0	162,2
Borohrádek	500,9	160,9

Město	Y	X
Ždírec nad Doubravou	497,0	158,1
Chrudim	499,5	157,9
Pardubice	500,4	157,8

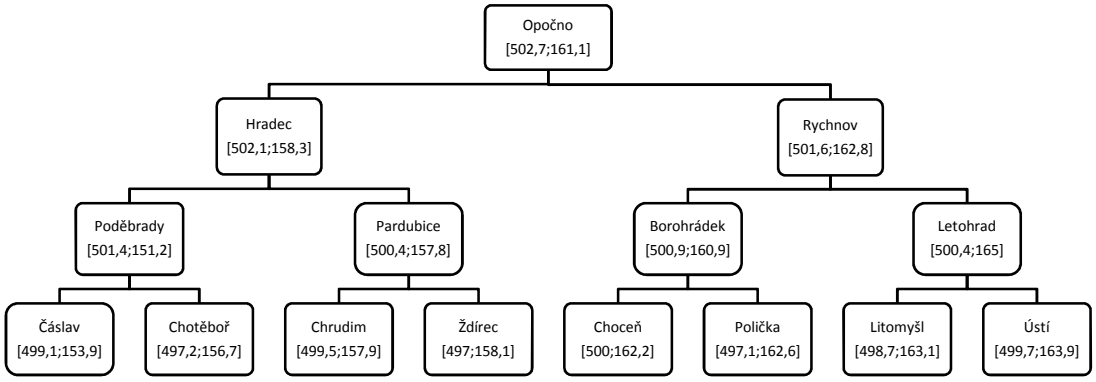
Město	Y	X
Litomyšl	498,7	163,1
Ústí nad Orlicí	499,7	163,9
Letohrad	500,4	165,0

Tab. 8a-8d) Konstrukce PST – data – krok 3



Obr. 28) Konstrukce PST – mapa – krok 3

Zbylé prvky seřadíme podle *x*-ové souřadnice a rozdělíme na polovinu. Nyní nám vznikne 8 nových podstromů. Jelikož každý z těchto 8 podstromů obsahuje již pouze jeden prvek, dělení zde končí a PST je vytvořen (Obr. 29).



Obr. 29) Konstrukce PST – stromová struktura – krok 3

Algoritmus

```
PSTNode buildPST(data)
    {
            //najdi prvek s největší hodnotou y
            node = getHighestY(data);
            //najdi medián
            node.median = getMedian(data);
            LH = getLeftHalf(data);
            RH = getRightHalf(data);
            //buduj levý podstrom
            TL = buildPST (LH);
            //buduj pravý podstrom
            TR = buildPST(RH);
            node.AddLeftChild(TL);
            node.AddRightChild(TR);
            return node;
    }
```

Nevýhodou této varianty je, že na ní nelze jednoduše navázat operacemi *přidej prvek* nebo *odeber prvek*. Důvodem je fakt, že v jakémkoliv uzlu stromu je uložena hodnota prvku, který je mediánem pro všechny následující potomky. Aby byla zachována filozofie této popsané konstrukce, muselo by se prohledávat velké množství dat kvůli novému nalezení mediánu, což by bylo značně neefektivní.

Naopak výhodou této konstrukce je vyvážený strom.

Varianta B

Odstranit nevýhodu předešlé varianty, lze určením si předem minimalní a maximální hodnoty *x*-ové souřadnice a v každém prvku neuchovávat hodnotu prvku, který je mediánem, ale polovinu z předem stanoveného rozsahu. Na tuto variantu lze později velmi jednodušše navázat dalšími operacemi.

Nevýhodou této varianty je možnost značně nerovnoměrného rozložení prvků v předem stanovených mezí, což bude vést k nevyváženému stromu.

2.2.3 Přidání prvku

Představím zde dvě varianty přidání prvku. Varianta A vychází z datové struktury Treap[3], která má stejný princip uložení prvků jako PST. Varianta B je specifická přímo pro PST. Obě varianty lze použít po operaci konstrukce (kap. 2.2.2) pouze v případě vytvoření variantou B.

Varianta A

- prvek postupně traverzuje stromem od kořene k listu na základě porovnávání *x*-ových souřadnic vkládaného a aktuálního prvku
- prvek je uložen jako list a z pohledu *x*-ové souřadnice je uložen správně, následuje úprava podle *y*-ové souřadnice
- pomocí operací levé a pravé rotace prvek traverzuje zpět nahoru stromem, pokud je potřeba

Varianta B

- pokud má vkládaný prvek *v* vetší hodnotu *y*-ové souřadnice než prvek aktuální *a*, jsou veškerá data prvků vyměněna
- vkládání pokračuje rekurzivně dolů stromem podle toho, do které poloviny vkládaný prvek *v* patří dle *x*-ové souřadnice

[3] Treap - http://en.wikipedia.org/wiki/Treap

Příklad - Varianta A

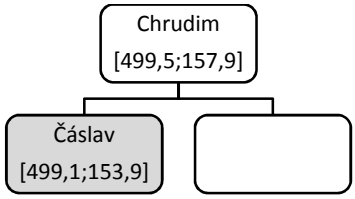
Mějme prázdnou strukturu PST a náhodně nám přichází 3 prvky (např. Chrudim, Čáslav, Choceň).

Krok 1

Příchází prvek Chrudim [499,5;157,9] a je vložen jako kořen stromu.

Krok 2

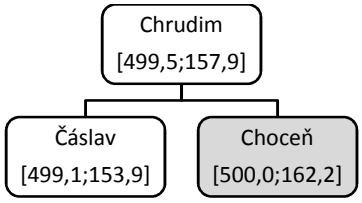
Přichází prvek Čáslav [499,1;153,9]. Protraverzuje na pozici listu jako levý syn aktuálního kořenového prvku. Prvek je uložen korektně, není tedy potřeba úprava podle *y*-ové souřadnice (Obr. 30).



Obr. 30) Vložení do PST A – stromová struktura – krok 2

Krok 3

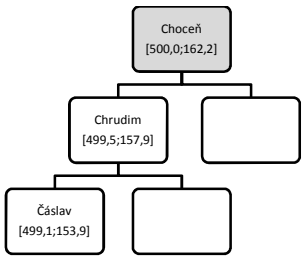
Vkládaným prvkem je Choceň [500;162,2]. Protraverzuje na pozici listu jako pravý syn aktuálního kořenového prvku.



Obr. 31) Vložení do PST A – stromová struktura – krok 2

Krok 4

Následuje úprava podle *y*-ové souřadnice. Na vložený prvek je aplikována operace *levá rotace*. Výsledek operace je zobrazen na obrázku



Obr. 32) Vložení do PST A – stromová struktura – krok 4

Algoritmus

```
AddNode(node)
{
        pom = root
        // úprava podle x-ové souřadnice
         while (konec == false)
        {
            if (pom.median <= node.x)
            {
                if (pom.haveRightChild()) pom = pom.GetRightChild();
                else
                {
                    pom.AddRightChild(prvek);
                    prvek.AddParent(pom);
                    konec = true;
                }
            }
            else
            {
                if (pom.haveLeftChild()) pom = pom.GetLeftChild();
                else
                {
                    pom.AddLeftChild(prvek);
                    prvek.AddParent(pom);
                    konec = true;
                }
            }
          }

        // úprava podle y-ové souřadnice
        while ((prvek.GetParent() != null) && (prvek.GetParent().y <
        prvek.y)
          {
            if (prvek.GetParent().GetRightChild() != null)
                if(prvek.GetParent().GetRightChild().x == prvek)
                    LeftRotation(prvek.GetParent());
                else RightRotation(prvek.GetParent());
            else RightRotation(prvek.GetParent());

          }
}
```

Příklad 2 – Varianta B

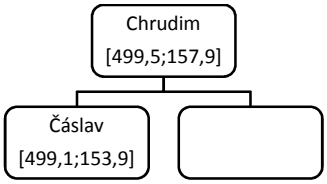
Mějme prázdnou strukturu PST a náhodně nám přichází 3 prvky (např. Chrudim, Čáslav, Choceň). Prostor máme napevno vymezen *x*-ovými souřadnicemi xmin a xmax [150;162].

Krok 1

Příchází prvek Chrudim[499,5;157,9] a je vložen jako kořen stromu.

Krok 2

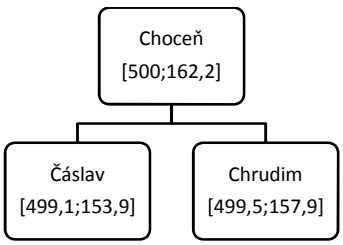
Přichází prvek Čáslav[499,1;153,9]. Jelikož má *y*-ovou souřadnici menší než kořenový prvek Chrudim, nedochází k výměně dat. Získáme xmiddle = (xmin + xmax) / 2. Po porovnání *x*-ové souřadnice vkládaného prvku a xmiddle vložíme prvek jako levého syna aktuálního prvku (Obr. 33).



Obr. 33) Vložení do PST B – stromová struktura – krok 2

Krok 3

Vkládaným prvkem je Choceň [500;162,2] a jeho *y*-ová souřadnice je větší než souřadnice akutálního prvku, kterým je kořenový prvek Chrudim [499,5;157,9]. Dochází k výměně dat mezi prvky. Aktuálním prvkem je nyní je nyní Choceň a vkládaným Chrudim. Zjistíme xmiddle [156]. Vkládaný prvek je vložen jako pravý syn aktuálního prvku (Obr. 34).



Obr. 34) Vložení do PST B – stromová struktura – krok 3

2.2.4 Odebrání prvku

Varianta A

U této varianty bylo čerpáno ze zdroje [13].

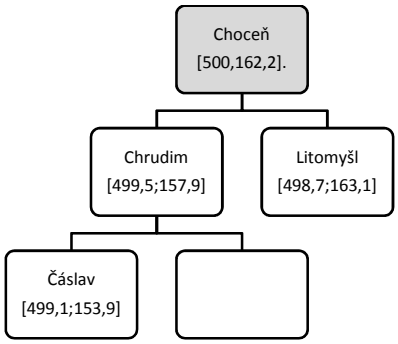
Zásady pro odebírání prvku ze stromu :

- je-li prvek listem stromu, pak je ze stromu rovnou odebrán
- jsou-li synové prvku listy stromu, pak je prvek odebrán a nahrazen synem s vyšší hodnotou *y*-ové souřadnice
- pokud neplatí ani jedno výše uvedené pravidlo, je prvek pomocí rotací přemisťován od kořene směrem k listům, dokud neplatí jedna z výše uvedených podmínek

Příklad

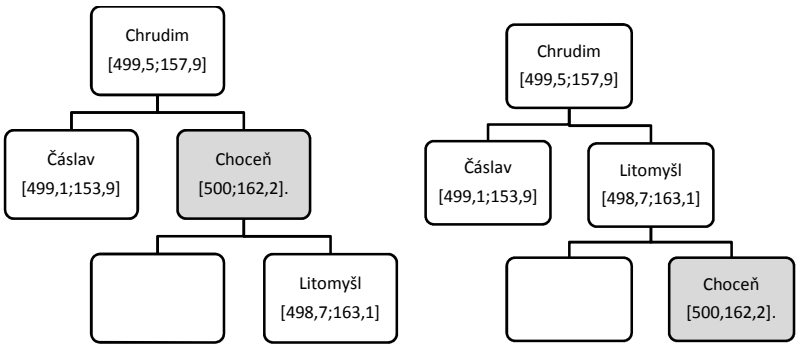
Krok 1

Mějme k dispozici výchozí stromovou strukturu na obrázku Obr. 35. Požadujeme odebrání kořenového prvku Choceň.



Obr. 35) Odebrání z PST – stromová struktura – krok 1

Pomocí pravé rotace a nahrazením jeho synem dostaneme prvek na pozici listu odkud je následně odstraněn (Obr. 36).



Obr. 36) Odebrání z PST – stromová struktura – krok 2

Algoritmus

```
PSTRemove(P)
{
        // Přemístění prvku na pozici listu
        While (RightChild <> Leaf AND LeftChild <> Leaf)
{
                If (LeftChild.Y < RightChild.Y) LeftRotation
                Else RightRotation
}
        // Odebrání
        If (LeftChild.Y< RightChild.Y) Replace(P, RightChild)
        Else Replace(P,LeftChild)
}
```

2.2.5 Vyhledávání

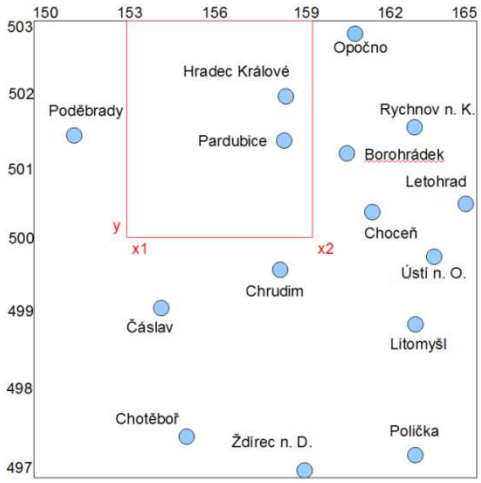
Ukážeme si tzv. vyhledávání ze tří stran (three-sided range query) nebo-li NajdiVšechnyVRozsahu(x1,x2,y). Postupujeme dolů stromem podobným způsobem jako u 1-rozměrného vyhledávaní pro rozsah [x1,x2]. V prohledávání podstromu prvku *p* pokračujeme pouze pokud *p*(y) >= y.

Podmínka 1 P(y) >= y

Podmínka 2 x1 <= P(x) <= x2

Příklad

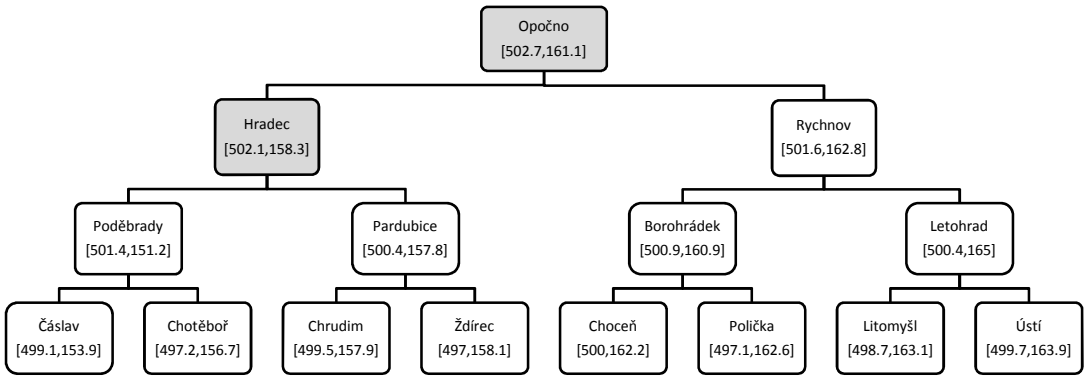
Mějme výchozí situaci na Obr. 37. Požadujeme vyhledání všech prvků, které mají *x*-ovou souřadnici v rozmezí 153 <= P(*x*) <= 159 a *y*-ovou 500 <= P(*y*).



Obr. 37) Vyhledávání v PST – mapa – krok 1

Krok 1

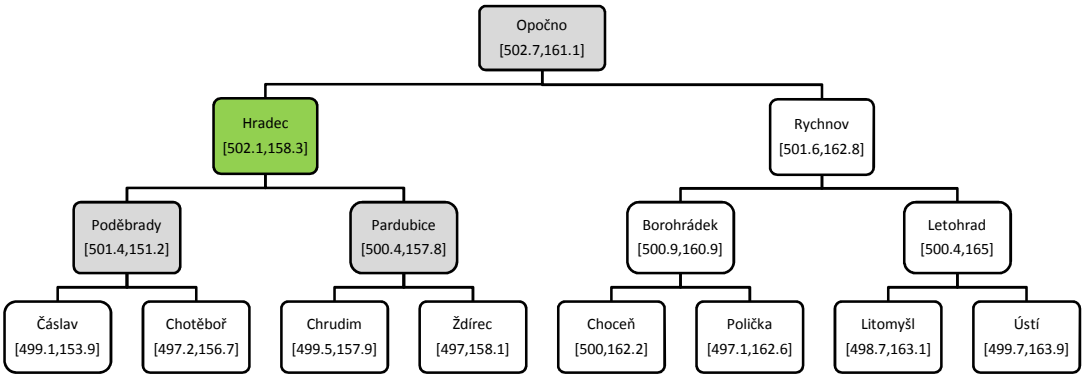
Začneme kořenovým prvkem, kterým je Opočno. Splňuje podmínku 1, ale nesplňuje podmínku 2, není tedy hledaným prvkem. Hledání pokračuje levým synem (Obr. 38).



Obr. 38) Vyhledávání v PST – stromová struktura – krok 1

Krok 2

Aktuálním prvek je Hradec. Splňujeme obě podmínky, je tedy prvním nalezeným prvkem. Vyhledávání pokračuje v obou jeho větvích (Obr. 39).



Obr. 39) Vyhledávání v PST – stromová struktura – krok 1

Krok 3

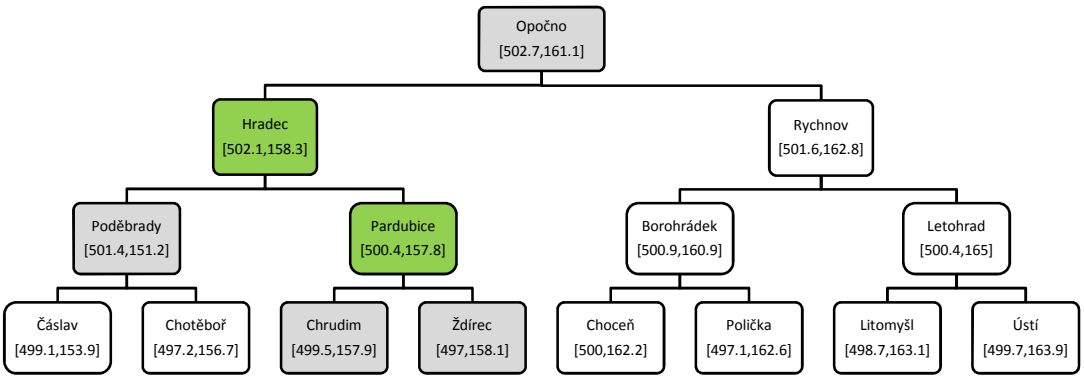
Aktuální prvkem se stává město Poděbrady. Nesplňuje podmínku 1, hledání v této větvi tedy končí.

Krok 4

Aktuálním prvkem se stává město Pardubice. Splňuje obě podmínky a je tedy hledaným prvkem. Hledání pokračuje v obou jeho větvích.

Krok 5

Ani jeden ze synů nesplňuje podmínku 1, hledání končí, nalezli jsme dva prvky, celkem navštívili prvků šest (Obr. 40).



Obr. 40) Vyhledávání v PST – stromová struktura – krok 1

Algoritmus

```
PSTSearch(x1,x2,y1,p)
{
        If (p(y)<y1) OR (p == null) return;
        If (x1 <= p(x) <= x2) OutputList.Add(p);
        If (x1<=p(x)) PSTSearch(x1,x2,y1,p.LeftChild)
        If (p(x) <= x2) PSTSearch(x1,x2,y1,p.RightChild)
}
```

2.3 Quad strom

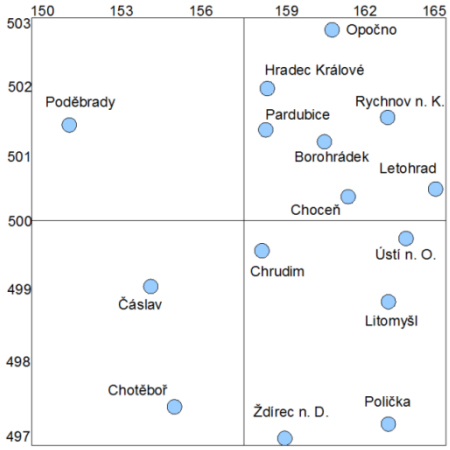
Datová struktura quad strom se používá výhradně k organizaci dat se dvěma nezávislými klíči jako prostorovými koordináty. Každý prvek stromu, který není listem, má čtyři syny. Prvek *p* stromu *t* reprezentuje čtvercový region *r*, čtyři synové prvku *p* reprezentují severovýchodní, jihovýchodní, jihozápadní a severozápadní kvartál regionu *r*. Pokud region *r* obsahuje více než jeden prvek, musí být rekurzivně dále dělen, dokud neobsahuje právě jeden prvek. V této kapitole bylo využito zdrojů [1], [3] a [8].

2.3.1 Konstrukce

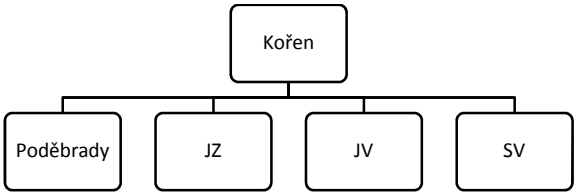
Existuje více variant quad stromu. Představím pouze tu nejzákladnější a nejjednodušší s rovnoměrným dělením prostoru na čtyři stejné kvartály. Nevýhodou této varianty je nutnost uchovávání informací o příslušném regionu v každém uzlu stromu, který není listem.

Krok 1

Prostor daný vzorovýma datama dělíme na čtyři rovnoměrné kvartály podle Obr. 41. Pokud jeden z kvartálů obsahuje pouze jeden prvek, je tento prvek vložen do stromu jako list (v tomto případě město Poděbrady), jinak je vytvořen syn kořene, který nese pouze informaci o příslušném regionu (Obr. 41).



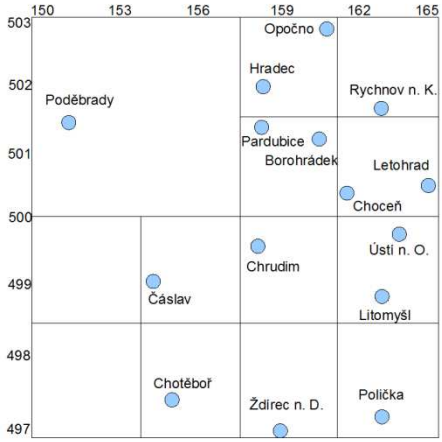
Obr. 41) Konstrukce QT – mapa – krok 1



Obr. 42) Konstrukce QT – strom– krok 1

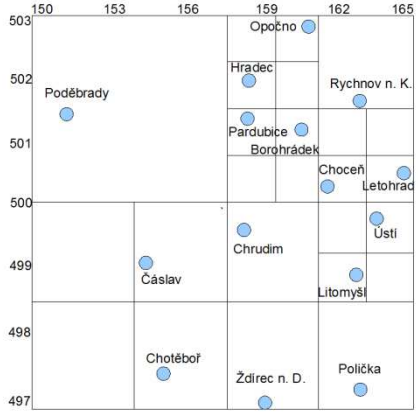
Krok 2

Dělení dále probíhá rekurzivně pouze na prvcích, která nenesou žadná data.

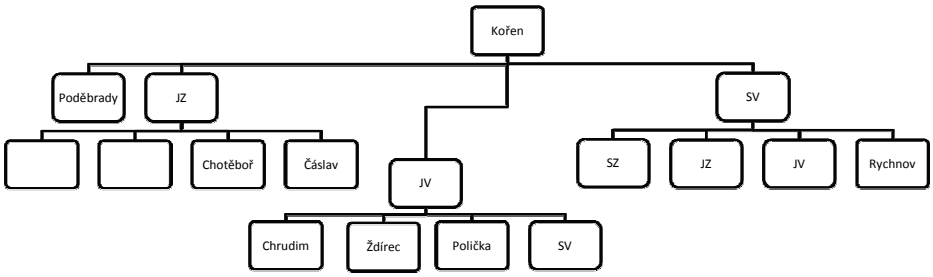


Obr. 43) Konstrukce QT – mapa – krok 2

Krok 3



Obr. 44) Konstrukce QT – mapa – krok 3



Obr. 1: Konstrukce QT – strom – krok 2

2.3.2 Přidání prvku

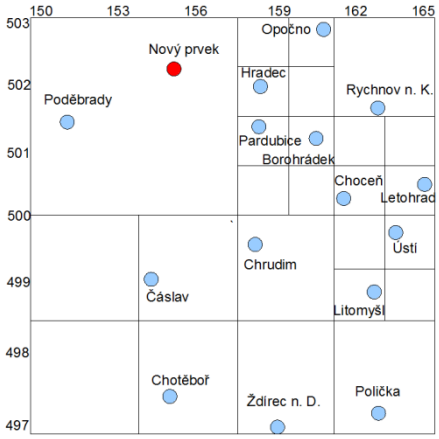
Je-li je region *r*, do kterého vkládáme prvek *p*, prázdný, je prvek *p* vložen přímo do regionu *r*.

Obsahuje-li region *r*, do kterého vkládáme prvek *p*, již nějaký prvek *q*, jsou regionu *r* přidáni čtyři syni (kvartály *k1* až *k4*) a prvek *q* se posouvá na pozici jednoho ze synů.

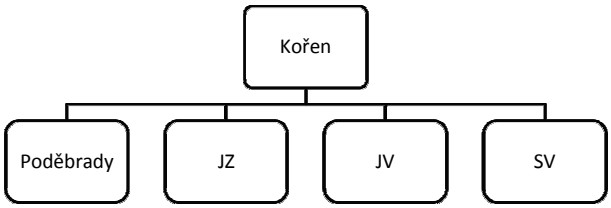
Příklad

Krok 1

Vyjdeme z výsledku operace *konstrukce* quad stromu (Obr. 45). Budeme vkládat *Nový prvek*. V tomto příkladu si zobrazíme pouze část stromové struktury, která je pro ukázku operace *přidání prvku* podstatná (Obr. 46).



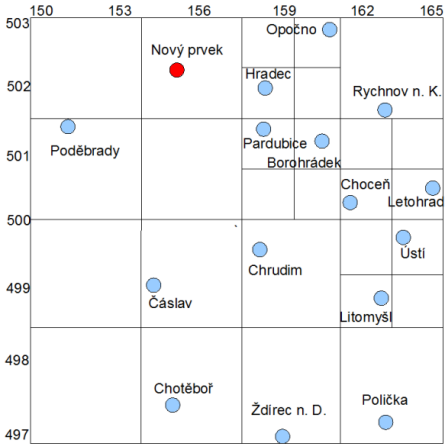
Obr. 45) Přidání prvku do QT - mapa - krok 1



Obr. 46) Přidání prvku do QT - stromová struktura - krok 1

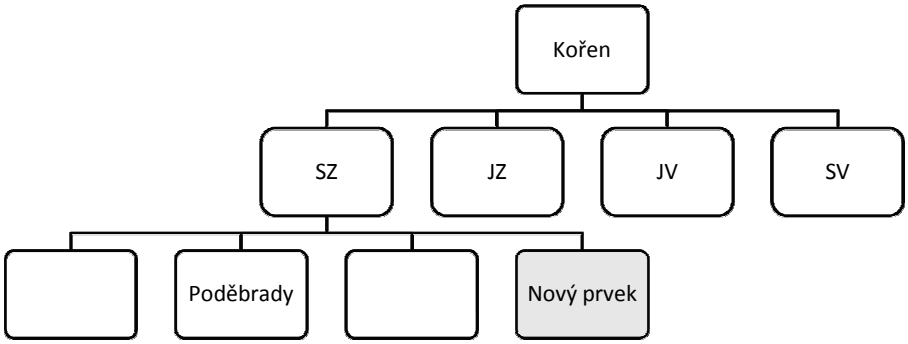
Krok 2

Region, do kterého jsme vložili *Nový prvek*, rozdělíme na čtyři kvartály podle obrázku Obr. 47.



Obr. 47) Přidání prvku do QT - mapa - krok 2

Ve stromové struktuře přejdeme do severozápadního kvartálu na prvek Poděbrady. Přidáme mu čtyři syny a přesuneme jej do příslušného nového kvartálu. Určíme severovýchodní kvartál, do kterého N*ový prvek* spadá, a vložíme prvek na pozici čtvrtého syna aktuálně rozvětveného prvku.



Obr. 48) Přidání prvku do QT - stromová struktura - krok 2

Algortimus

```
AddNode(Node startNode, Node insertNode)
{
        if (startNode.isNotEmpty())
                //rozdělení regionu
                Split(startNode)
                nextNode = ChooseLeaf(startNode, insertNode.getKey())
                AddNode(nextNode, insertNode)
        else if (startNode.isLeaf())
                startNode = insertNode
        else
                nextNode = ChooseLeaf(startNode,insertNode.getKey())
```

```
                AddNode(nextNode, insertNode)
}
Split(Node node)
{
        p = node.getKey()
        node.deleteKey()
        leaf = ChooseLeaf(node, p)
        leaf.setKey(p)
}
//určení nového kvartálu
ChooseLeaf(Node node, Point p)
        return region
```

2.3.3 Odebrání prvku

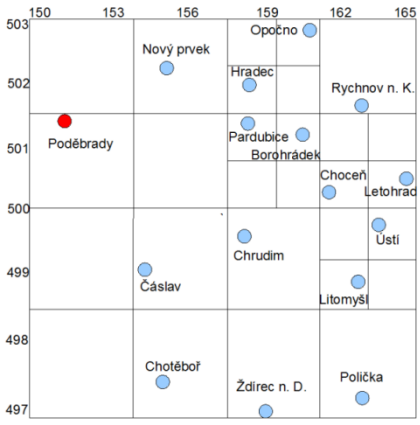
Definice

- odebíraný prvkem je vždy listem
- má-li odebíraný prvek *p* více než jednoho bratra, je tento prvek *p* odebrán a struktura se nemění
- má-li odebíraný prvek *p* jednoho bratra, je prvek *p* odebrán a bratr je vyměněn se svým otcem

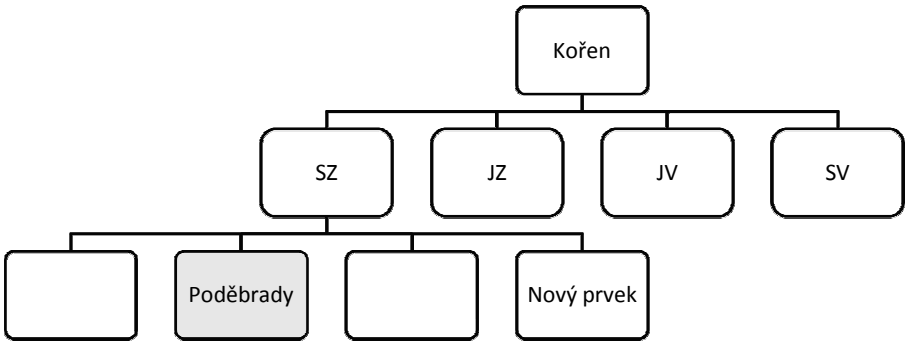
Příklad

Krok 1

Mějme výchozí situaci z předchozí operace *přidání prvku* a požadujeme nyní odebrání prvku Poděbrady (Obr. 49 a Obr. 50).



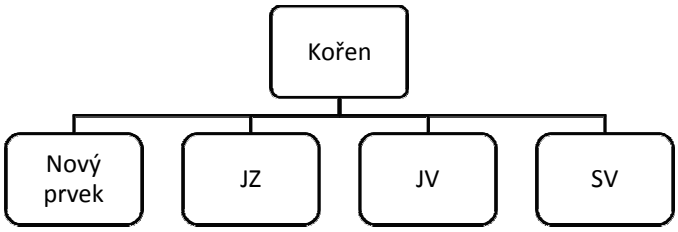
Obr. 49) Odebrání prvku z QT - mapa - krok 1



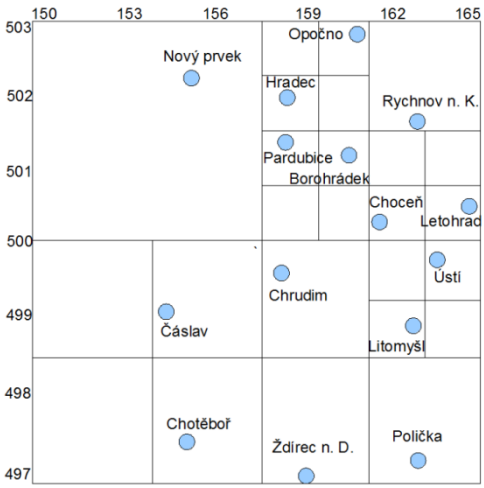
Obr. 50) Odebrání prvku z QT - stromová struktura - krok 1

Krok 2

Odebíraný prvek má pouze jednoho bratra, prvek Poděbrady je odebrán a jeho bratr je vyměněn s otcem (Obr. 51). Výsledkem je sloučeni kvartálů do jednoho reginu (Obr. 52).



Obr. 51) Odebrání prvku z QT - stromová struktura - krok 2

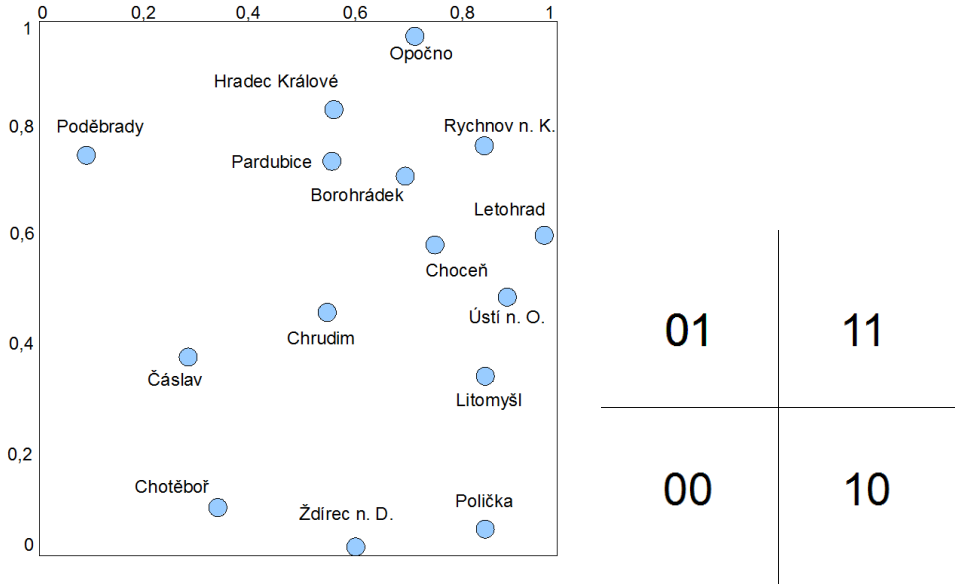


Obr. 52) Odebrání prvku z QT - mapa - krok 2

2.3.4 Využití číslicového stromu

Tento způsob odstraňuje nevýhodu předešlé varianty, kdy si uzel stromu, který nese pouze informaci o regionu, nemusí tuto informaci pamatovat.

Souřadnice prvků převedeme na desetinná čísla, kde $0 <= x < 1$ a $0 <= y < 1$. Tato desetinná čísla dále převedeme do binární podoby (Tab. 9). Poté každý podstrom bude reprezentován jedním ze čtyř označení 00, 01, 10 nebo 11.



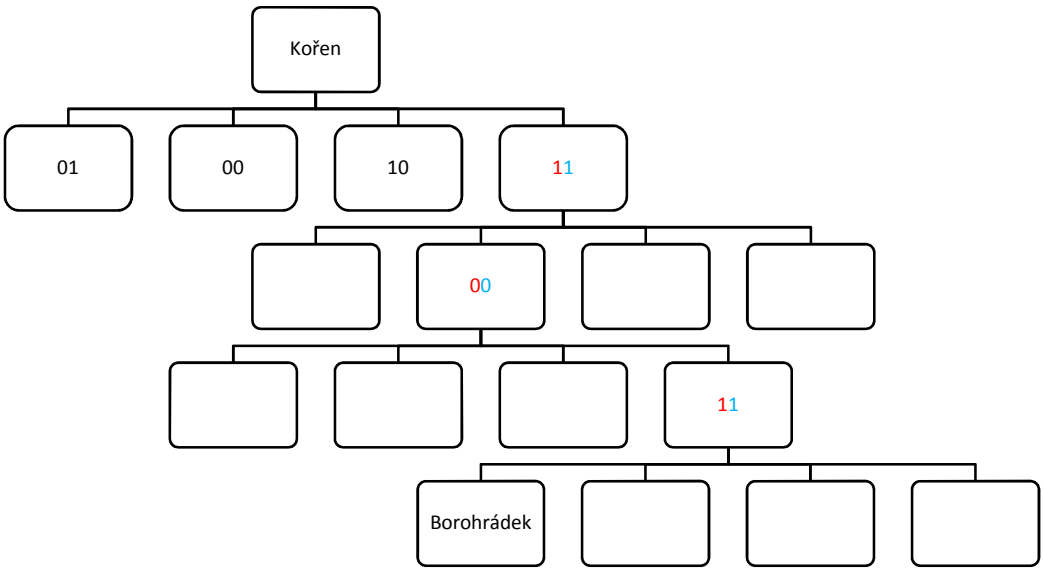
Obr. 53) QT - Využití číslicového stromu

Město	Y[10]	X[10]	Y[2]	X[2]
Ždírec nad Doubravou	0,00000000	0,49640288	0,00000000	0,01111111
Polička	0,01724138	0,82014388	0,00000100	0,11010001
Chotěboř	0,03448276	0,39568345	0,00001000	0,01100101
Litomyšl	0,29310345	0,85611511	0,01001011	0,11011011
Čáslav	0,36206897	0,19424460	0,01011100	0,00110001
Chrudim	0,43103448	0,48201439	0,01101110	0,01111011
Ústí nad Orlicí	0,46551724	0,91366906	0,01110111	0,11101001
Choceň	0,51724138	0,79136691	0,10000100	0,11001010
Letohrad	0,58620690	0,99280576	0,10010110	0,11111110
Pardubice	0,58620690	0,47482014	0,10010110	0,01111001
Borohrádek	0,67241379	0,69784173	0,10101100	0,10110010
Poděbrady	0,75862069	0,00000000	0,11000010	0,00000000
Rychnov nad Kněžnou	0,79310345	0,83453237	0,11001011	0,11010101
Hradec Králové	0,87931034	0,51079137	0,11100001	0,10000010
Opočno	0,98275862	0,71223022	0,11111011	0,10110110

Tab. 9) QT - převod do binárních souřadnic

Příklad

Vložíme město Borohrádek, které má po převedení do binární soustavy tyto souřadnice [0,10101100; 0,10110010]. Pro přehlednost obě souřadnice spojíme a každá dvojice bude představovat jednu úroveň stromu **1**1**00**1**1**0**1**1**0**1**0**0**1**00. Vkládání si ukážeme na přesnosti čtyř desetinných míst (tedy **1**1**00**1**1**01) na Obr. 54.



Obr. 54) QT - přidání prvku do číslicového stromu

První syn kořene, nesoucí označení 01, reprezentuje množinu všech prvků, jejichž binární vyjádření souřadnic začíná 0 pro x a 1 pro y. Nebo-li v jiném slova smyslu : množinu všech prvků, jejichž souřadnice jsou v rozmezí $0 < x < 0{,}5$ a $0{,}5 < y < 1$ v desetinné podobě souřadnic.

2.4 Oktálový strom

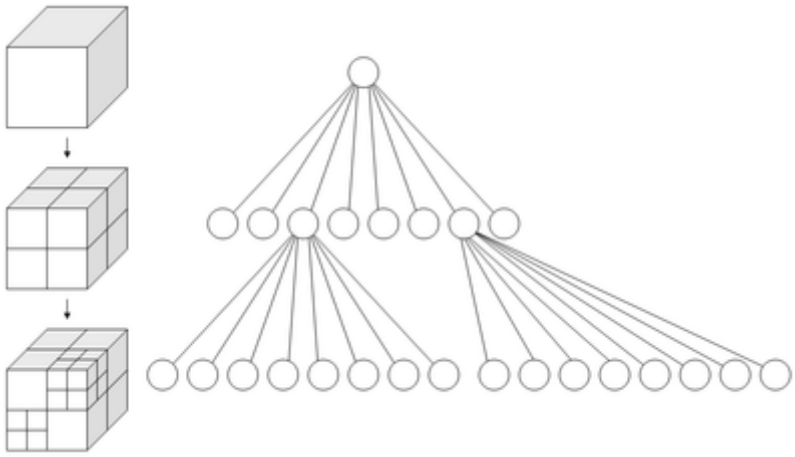
Velmi krátce představím oktálový strom (angl. octal tree, octne), jelikož je takřka totožný s předešlým uvedeným quad stromem. Rozdílem je jejich oblast použití, kdy oktálový strom je zaměřen na 3D prostor namísto 2D.

Z praktického hlediska má oktálový strom velmi velkou využitelnost v oblasti počítačových her. Umožňuje rozdělením 3D prostoru např. vykreslit pouze tu část

scény, která je viditelná z pohledu kamery. Tím se rapidně sníží datový tok, který je posílán na grafickou kartu, počet snímků/s se logicky zvýší a obraz bude plynulejší.

Definice

- každý prvek, který není listem, obsahuje osm synů
- data jsou uložena pouze v listech
- prvky, které nejsou listy, nesou informaci pouze o příslušném regionu
- dělí prostor na osm rovnoměrných krychlí



Obr. 55) Oktálový strom (zdroj [9])

2.5 Grid file

Grid file je dvouúrovňová datová struktura, která obsahuje:

- řídící strukturu (angl. grid directory), kterou si nazveme *Adresář*
- množinu datových bloků (angl. bucketů), které si nazveme *Sektory*

Adresář se dále dělí také na dvě části:

- dynamické k-rozměrné pole (angl. grid array), které si nazveme *Mřížka*
- *k* 1-rozměrných polí (angl. linear scales), které si nazveme *Stupnice*

2.5.1 Konstrukce

Jelikož je princip konstrukce této datové struktury složitější než všechny předešlé, vysvětlím princip přímo na příkladě a obrázkách. Jelikož pracujeme ve 2D prostoru máme k dispozici dvě 1-rozměrné stupnice *Sx*, *Sy* a 2-rozměrnou mřížku *M*.

Příklad

Mějme prázdnou strukturu a přichází nám postupně 5 prvků. Sektory nastavíme pro ukázkové účely na maximální počet prvků roven 2.

Krok 1

Vkládáme prvek Borohrádek. Stupnice *Sx* a *Sy* budou obsahovat dva prvky určující minimum a maximum v daném rozměru. Indexy v těchto stupnicích ukazují do mřížky *M* na pozici [1,1], kde je uloženo číslo sektoru, kde se bude nacházet námi vkládaný prvek (Obr. 56 a Obr. 57).

Stupnice Sx a Sy

	0	1	2	3	4	5	6
Sx	150	165					
Sy	497	503					

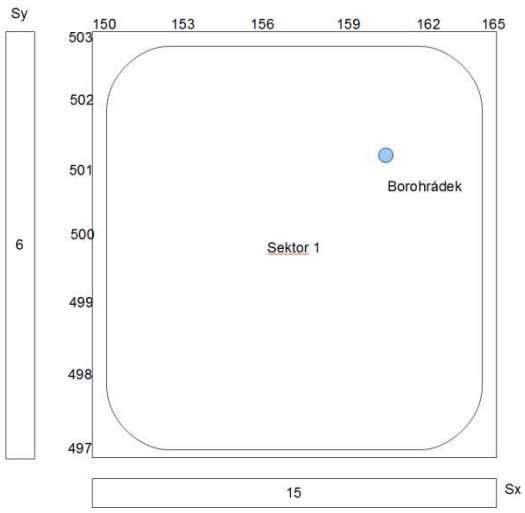
Mřížka M

M	1	2	3	4
1	1			
2				

Sektory

1
Borohrádek

Obr. 56) Grid file - data - krok 1



Obr. 57) Grid file – sektory – krok 1

Krok 2

Vkládáme prvek Čáslav [499.1,153.9]. Ve stupnicích *Sx* a *Sy* vyhledáme nejnižší větší hodnoty odpovídající souřadnicím. Indexy *Sx*[1] a *Sy*[1] nás odkáží do mřížky *M* na pozici [1,1], kde po přesunutí na sektor 1 zjistíme, že obsahuje pouze jeden prvek. Do sektoru tedy přidáme náš vkládaný prvek. Ve stupnicích ani v mřížce se nic nemění (Obr. 58 a Obr. 59).

Stupnice Sx a Sy

	0	1	2	3	4	5	6
Sx	150	165					
Sy	497	503					

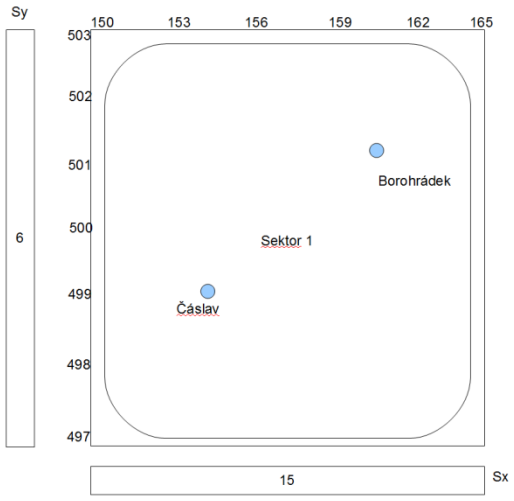
Mřížka M

M	1	2	3	4
1	1			
2				

Sektory

1
Borohrádek Čáslav

Obr. 58) Grid file - data - krok 2



Obr. 59) Grid file – sektory – krok 2

Krok 3

Vkládáme prvek Hradec Králové [502.1,158.3]. Stejným postupem najdeme sektor 1 a zjistíme, že jich obsahuje maximální počet prvků. Nezbývá než rozdělit prostor podle zvolené souřadnice (např. *x*), přidat prvek do stupnice *Sx*, mřížky *M* a založit nový sektor. Zároveň musí dojít k reorganizaci prvků v sektorech, kde se prvek Borohrádek musí přesunout do nově vytvořeného sektoru 2.

Stupnice Sx a Sy

	0	1	2	3	4	5	6
Sx	150	157,5	165				
Sy	497	503					

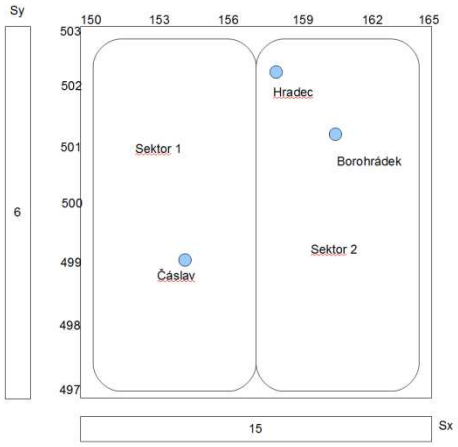
Mřížka M

y/x	1	2	3	4
1	1	2		
2				

Sektory

1	2
Čáslav	Borohrádek
	Hradec

Obr. 60) Grid file - data - krok 3



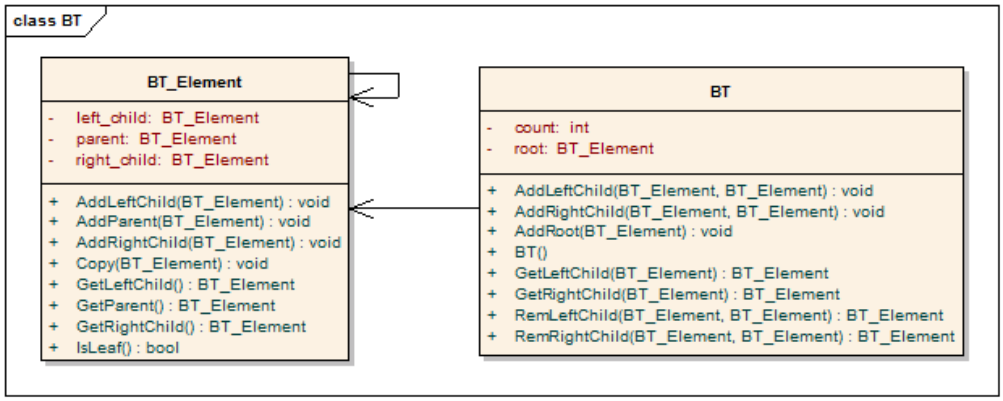
Obr. 61) Grid file - data - krok 3

3 Návrh tříd

U všech stromových struktur (tedy kromě grid file) jsou vždy navrženy dvě abstraktní datové struktury. Jedna pro prvek stromu a druhá hlavní, která tento prvek využívá a obsahuje veškeré operace příslušné dané struktuře. Dále jsou navrženy zděděné třídy, které pracují s již konkrétními daty.

3.1 Třídy pro binární strom

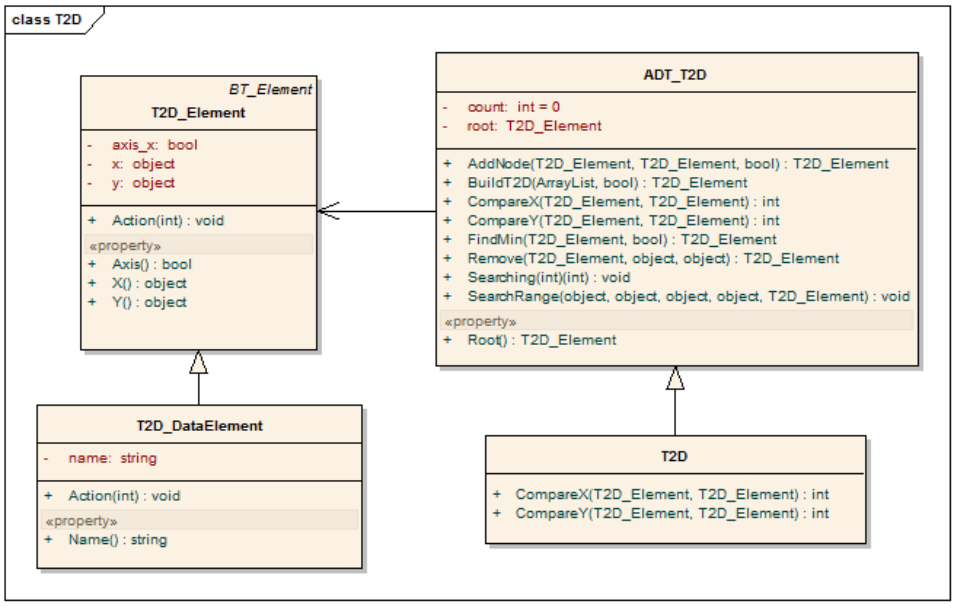
Jelikož *k*-d strom a prioritní vyhledávací strom mají svou strukturu založenou na binárním vyhledávacím stromu, je dobré si navrhnout nejdříve třídu pro prvek binárního stromu *BT_Element* a poté samotnou třídu *BT* obsahující kořen stromu, který je typu *BT_Element*, a příslušné operace pro binární vyhledávací strom. (Obr. 62)



Obr. 62) Návrh binárního vyhledávacího stromu

3.2 Třídy pro 2-d strom

Byla navržena abstraktní třída *T2D_Element*, která představuje prvek 2-d stromu a je potomkem dříve vytvořené třídy pro binární vyhledávací strom *BT_Element*, a hlavní abstraktní třída *ADT_T2D*, která obsahuje veškeré operace pro práci s tímto prvkem. Dále třída *T2D_DataElement*, která je potomkem třídy *T2D_Element* a obsahuje již veškerá uživatelská data. Poslední třídou, která je potomkem třídy *ADT_T2D*, je třída *T2D*, která pomocí polymorfismu, přepisuje virtuální metody pro porovnávání jednotlivých prvků již s konkrétními daty (Obr. 63).

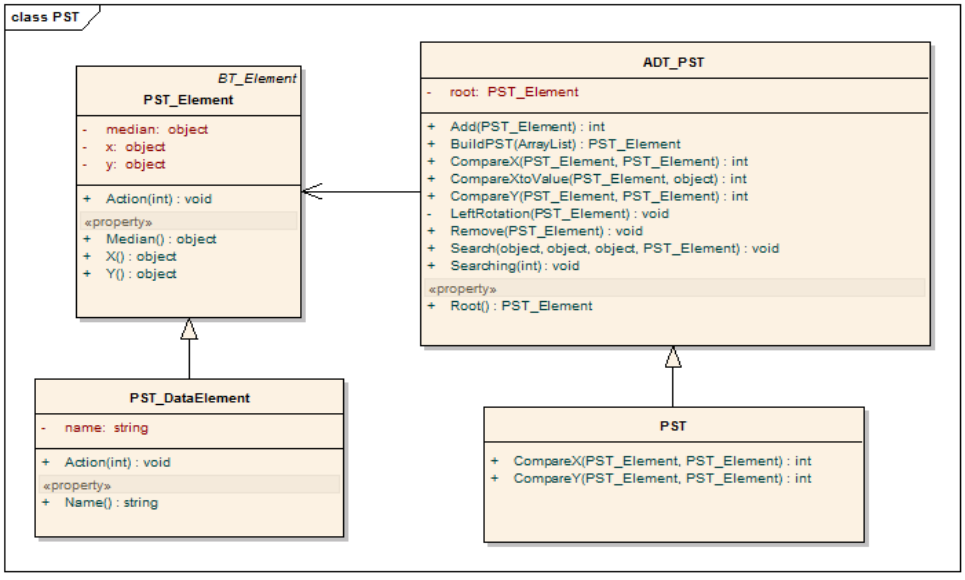


Obr. 63) Návrh tříd pro 2-d strom

3.3 Třídy pro prioritní vyhledávací strom

Takřka totožný způsob (jako u předešlého 2-d stromu) vytvoření tříd pro prioritní vyhledávací strom zachycuje Obr. 64.

Základní dvě abstraktní třídy *PST_Element* a *ADT_PST* a odvozené *PST_DataElement* a *PST*.



Obr. 64) Návrh tříd pro prioritní vyhledávací strom

3.4 Třídy pro quad strom

Jelikož u quad stromu již nelze využít základní třídy pro binární strom, byly navrženy nové s ohledem na využitelnost i u oktálového stromu.

Třída *Point* – obecná třída pro bod v *k*-rozměrném prostoru

Třída *QT_Bin_Point* – potomek třídy *Point* obohacený o atributy konverze do desítkové a binární soustavy, tato třída je potřebná pro využití číslicového stromu

Třída *Base_Element* – základní stavební prvek pro quad strom i oktálový strom, určuje počet synů ve stromě

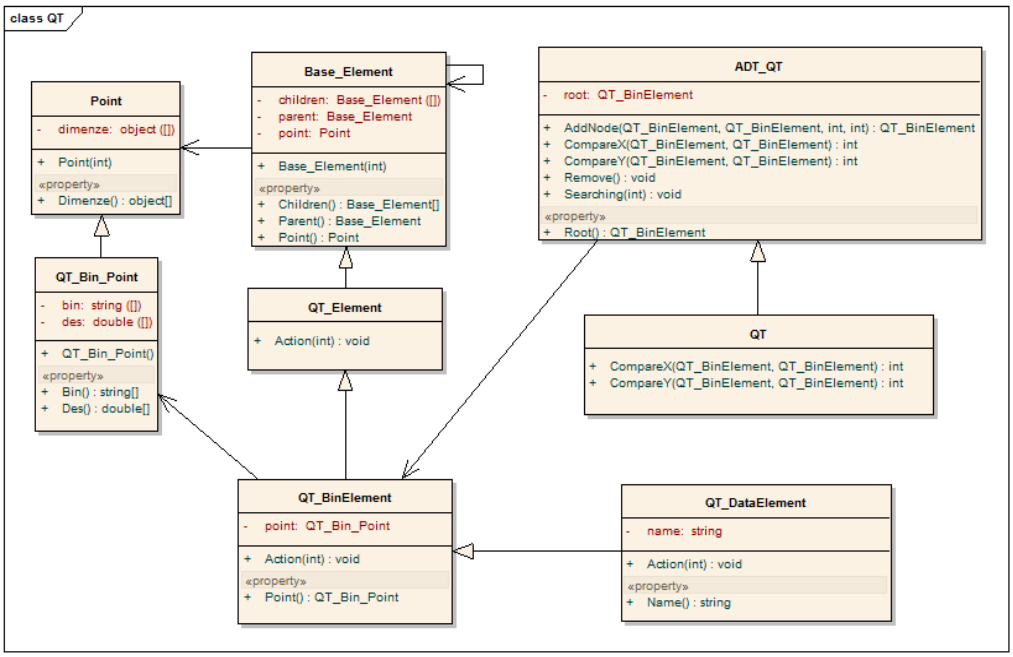
Třída *QT_Element* – potomek třídy *Base_Element* specifický pouze pro quad strom a určující počet synů ve stromě na čtyři

Třída *QT_BinElement* – prvek quad stromu již s využitím číslicového stromu

Třída *QT_DataElement* – třída pro prvek quad stromu již s konkrétnimi daty

Třída *ADT_QT* – hlavní abstraktní třída se všemi operacemi quad stromu

Třída *QT* – potomek třídy *ADT_QT* předefinovávající metody na porovnání prvků



Obr. 65) Návrh tříd pro quad strom

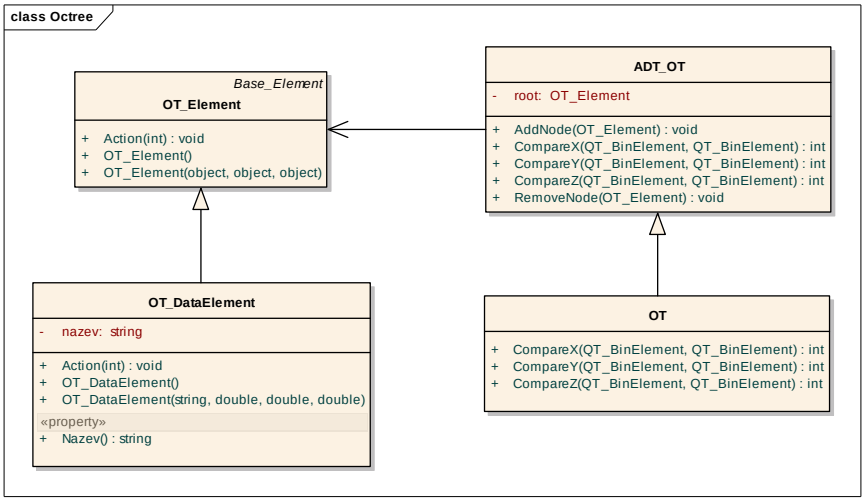
3.5 Třídy pro oktálový strom

Třída O*T_Element* – potomek třídy *Base_Element* určující počet 8 synů ve stromě

Třída *OT_DataElement* – třída pro prvek již s konkrétnimi daty

Třída *ADT_OT* – hlavní abstraktní třída se všemi operacemi

Třída *OT* – potomek třídy *ADT_OT* předefinovávající metody na porovnání prvků



Obr. 66) Návrh tříd pro oktálový strom

3.6 Třídy pro grid file

Byly navrženy třídy potřebné pro práci s blokovým přenosem dat z externí paměti do operační paměti a zpět, dále třídy potřebné pro samotný princip grid file.

3.6.1 Třídy pro blokový soubor

Třída Control – kontrolní blok pro buffer

Třída Buffer – buffer pro přenášený blok záznamů do paměti

Třída RidiciBlok – řídící blok pro správu záznamů v externí paměti

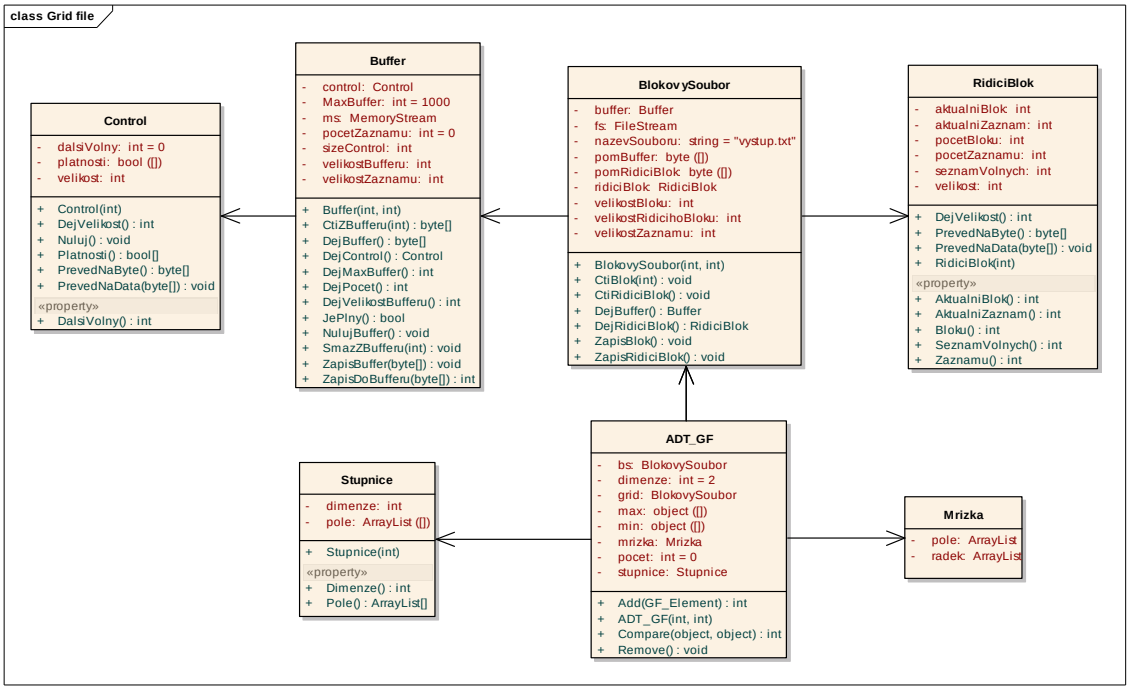
Třída BlokovySoubor – hlavní třída pro práci s blokovým souborem

3.6.2 Třídy pro grid file

Třída Stupnice – třída uchovávající *k* 1-rozměrných stupnic

Třída Mrizka – třída uchovávající potřebnou *k*-rozměrnou mřížku

Třída ADT_GF – hlavní abstraktní třída pro grid file



Obr. 67) Návrh tříd pro grid file

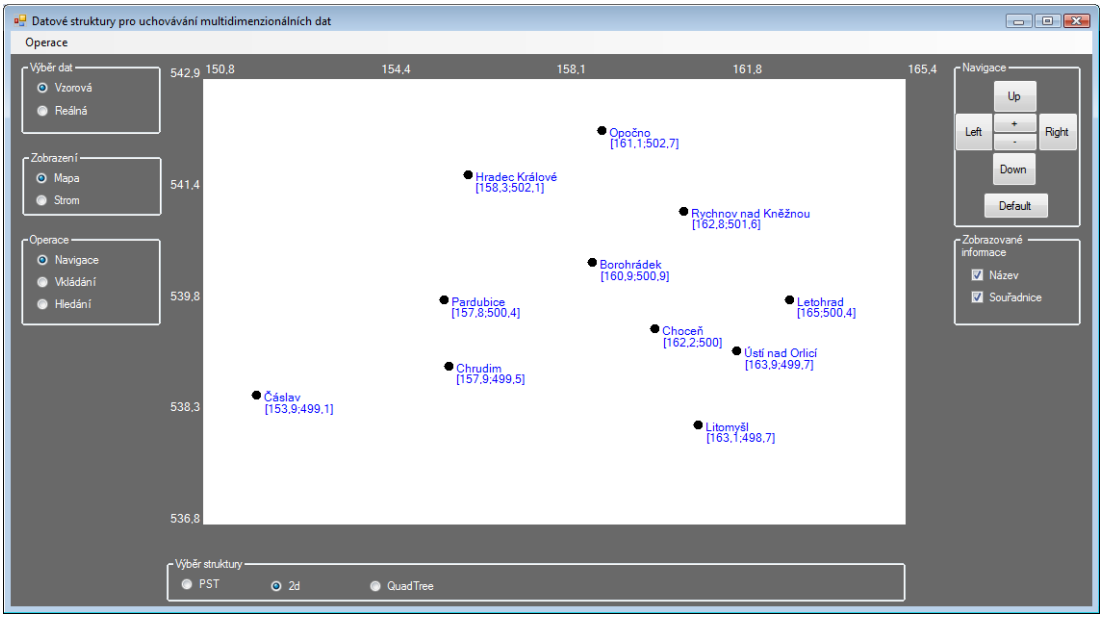
4 Implementace tříd a GUI aplikace

Samotné třídy i výsledná testovací GUI aplikace byly vytvořeny v programovacím jazyce C# v technologii .NET ve vývojovém prostředí Visual Studio 9.

Ve výsledné aplikaci byly použity tři typy dat. A to vzorová data, na kterých byla prezentována veškerá analýza datových struktur. Dále reálná data, která obsahují GPS souřadnice bankomatů České spořitelny. Z důvodů malé množiny reálných dat, která obsahuje přibližně 1200 záznamů, byl vytvořen pro testovací účely ještě jeden typ dat, a to náhodně generovaná.

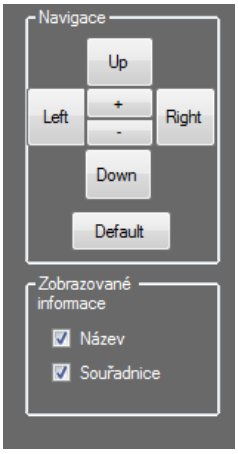
Základní stromové struktury *k*-d strom, prioritní vyhledávací strom a quad strom byly použity v aplikaci vizuálně. Tedy na vzorových datech lze u nich zobrazit jak rozložení na mapě ve 2D prostoru, tak výslednou stromovou strukturu (mimo quad stromu). Na reálných datech lze zobrazit pouze mapu.

Datové struktury grid file a oktálový strom jsou implementovány pouze kódově a z časových důvodů nebyly úplně dokončeny.



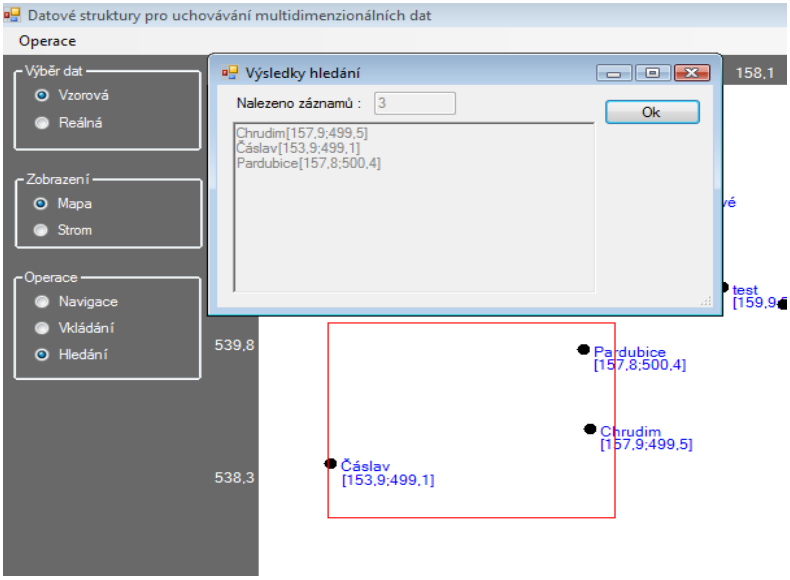
Obr. 68) Výsledná testovací GUI aplikace

Mapové rozložení lze pomocí navigačního panelu přibližovat, oddalovat a posouvat všemi čtyřmi směry. Lze také zobrazit nebo skrýt dodatečné informace u prvků (Obr. 69).



Obr. 69) GUI aplikace - navigační panel

V levém panelu Operace lze po přepnutí na položku Hledání otestovat rozsahové vyhledávaní u k-d stromu a PST. Po vybrání oblasti hledání na mapovém podkladu (od levého dolního rohu k hornímu pravému rohu) se automaticky zobrazí okno s nalezenými prvky (Obr. 70). U PST je omezující u y-ové souřadnice pouze dolní hrana (kvůli zvolenému vyhledávání ze tří stran).

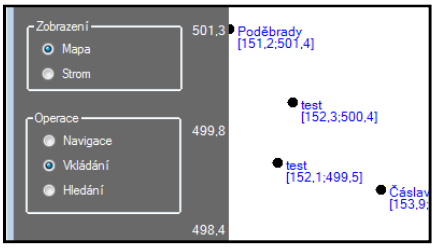


Obr. 70) GUI aplikace – rychlé rozsahové hledání

Potřebnou datovou strukturu lze zobrazit pomocí dolního panelu (Obr. 71). Na zvolenou strukturu je aplikována operace *prohlídka* a všechna data jsou vykreslena. V závislosti na zvolené volbě v panelu Zobrazení se zobrazí buď rozvržení dat v prostoru nebo stromová struktura (mimo quad stromu).

Obr. 71) GUI aplikace – výběr struktury

V levém panelu Operace lze také využít rychlého vkládání testovacích dat. Pouhým klikáním do mapového prostoru se vkládají testovací body (Obr. 72)



Obr. 72) GUI aplikace – rychlé vkládání

Operace přidej prvek, odeber prvek a hledání lze zadat i přímo ručně konkrétními údaji přes horní menu Operace, kde se vyvolá příslušné dialogové okno pro zadání dat.

5 Testování

Testování proběhlo na všechny čtyři základní operace : Konstrukce z předem známých dat, Přidání prvku, Odebrání prvku a Hledání prvků v příslušném rozsahu.

Hledisko časové složitosti bylo zvoleno následovně :

- každý test byl několikrát opakován se stejnými daty
- byl zaznamenán průměrný čas operace v milisekundách
- byl vytvořen poměr těchto časů pro každou testovanou strukturu
- nejrychlejší operace pro danou strukturu dostala označení 1 a ostatní v příslušném poměru k ní

Toto hledisko časové složitosti bylo zvoleno z důvodů běhu programu na případném odlišném hardwaru a softwaru.

Byly použity dva typy dat :

- reálná
- generovaná

Do testování byly zahrnuty tři základní datové struktury : *k*-d strom, prioritní vyhledávací strom a quad strom.

5.1 Testování č. 1

Testování proběhlo na reálných datech, kdy bylo použito všech 1175 záznamů. U *k*-d stromu byla použita operace *konstrukce* (kap. 2.1.22.1.2), u prioritního vyhledávacího stromu obě varianty operace *konstrukce* (kap. 2.2.2) a u quad stromu operace *přidání prvku s využitím číslicového stromu* (kap. 2.3.4), jelikož popsaná varianta quad stromu není závislá na pořadí a znalosti dat předem a tudíž dává vždy stejné výsledky.

100 pokusů	2-d	pst A	pst B	qt
reálná 1175	1	2,2	2,2	1

Tab. 10) Výsledky testu č. 1

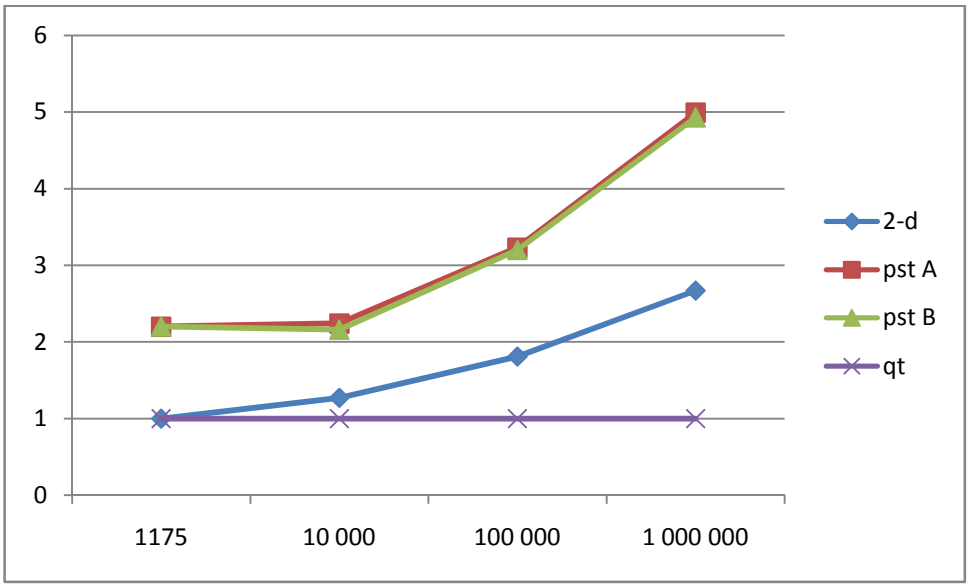
5.2 Testování č. 2

Počet záznamů z reálných dat je přeci jenom nedostačující, proto byly vygenerovány tři skupiny náhodných dat a každá operace byla 10krát opakována (Tab. 11).

10 pokusů	2-d	pst A	pst B	qt
generovaná 10 000	1,27	2,24	2,16	1
generovaná 100 000	1,81	3,23	3,2	1
generovaná 1 000 000	2,67	4,99	4,93	1

Tab. 11) Výsledky testu č. 2

Z výsledku testů je zřejmé, že u quad stromu odpadá časová režie spojená se zpracováním předem známých dat, tudíž podává nejrychlejší výsledky. Obě varianty u PST se téměř neliší. Celkové porovnání testů č. 1 a 2 přináší Graf. 1.



Graf. 1) Výsledky testů 1 a 2

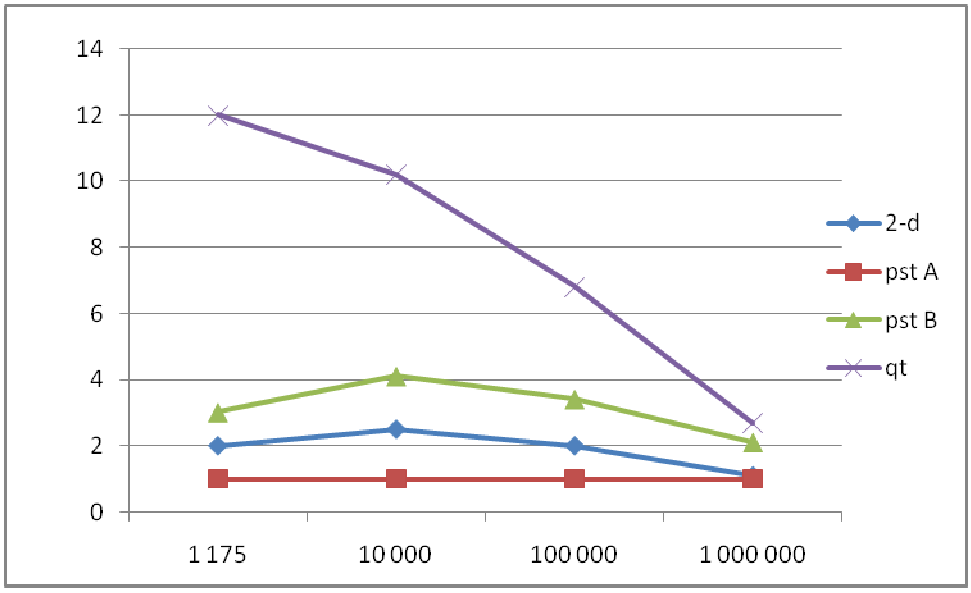
5.3 Testování č. 3

Předpokladem tohoto testování jsou prázdné struktury a neznalost dat předem. Data přicházejí postupně v náhodném pořadí. Testována je u *k*-d stromu operace *přidání prvku* (kap. 2.1.2.1), u prioritního vyhledávacího stromu obě varianty *přidání prvku* (kap. 2.2.3) a u quad stromu operace *přidání prvku s využitím číslicového stromu* (kap. 2.3.4).

Výsledky testy jsou zobrazeny v tabulce Tab. 12 a grafu Graf. 2. Zde strukturám *k*-d strom a PST odpadla režie spojená ze zpracováním dat předem a výsledné operace jsou u nich mnohem rychlejší než u quad stromu. Cenou za tuto rychlost budou jistě hodně nevyvážené stromy. U quad stromu je tato rychlost způsobena vytvářením velkého počtu prázdných regionů. S rostoucí výškou stromu se ale rozdíly postupně stírají.

10 pokusů	2-d	pst A	pst B	qt
reálná 1175	2	1	3	12
generovaná 10 000	2,5	1	4,1	10,2
generovaná 100 000	2	1	3,4	6,8
generovaná 1 000 000	1,09	1	2,09	2,68

Tab. 12) Výsledky testu 3

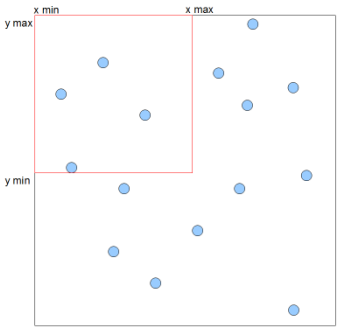


Graf. 2) Výsledky testu 3

5.4 Testování č. 4

Předpokladem toho testování bude provedení testu č. 2 (nezapočítává se do měření), tedy naplnění struktur z předem známých dat, a následné rozsahové vyhledávání. Strukturu quad strom musíme z tohoto testu vyřadit, jelikož implementace rozsahového vyhledávání není k dispozici.

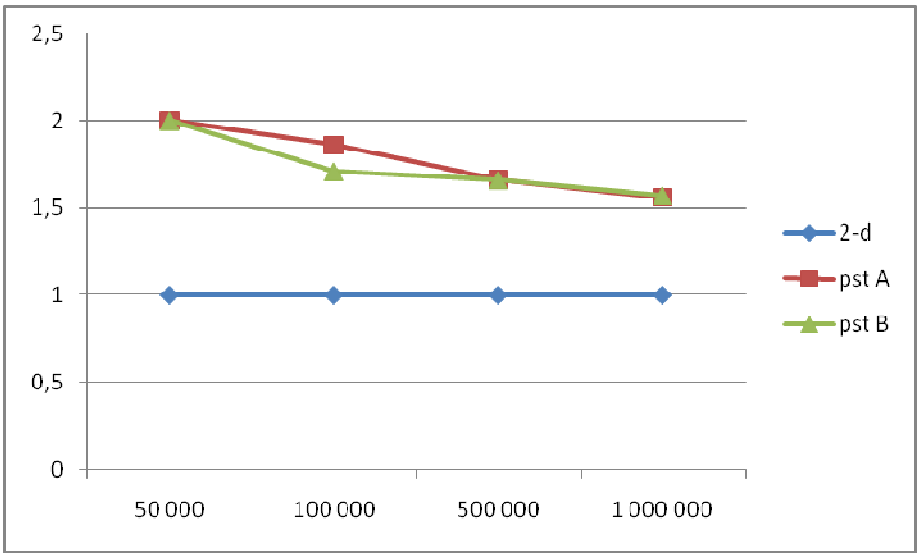
V testu byly použity operace *vyhledávání* (pro 2-d strom kap. 2.1.5 a pro prioritní vyhledávací strom kap. 2.2.5). Protože PST využívá pouze vyhledávání ze tří stran, byla pro vyhledávání vždy zvolena oblast levého kvadrantu, kde čtvrtou potřebnou omezující hranou bude maximální rozsah *y*. Vyhledávat se tedy bude přibližně čtvrtina počtu záznamů (Obr. 73).



Obr. 73) Testování - předpoklad testu 4

10 pokusů	2-d	pst A	pst B
generovaná 50 000	1	2	2
generovaná 100 000	1	1,86	1,71
generovaná 500 000	1	1,66	1,66
generovaná 1 000 000	1	1,56	1,57

Tab. 13) Výsledky testu 4



Graf. 3) Výsledky testu 4

5.5 Výsledky testování

Pokud bychom uvažovali doporučení datové struktury pro aplikaci, jejíž hlavním cílem je uchování předem známých dat a prioritní operací bude vyhledávání, lze na základě teoretických znalostí a testů č. 4 doporučit datovou strukturu *k*-d strom, která po vytvoření vyváženého stromu bude poskytovat velmi dobrou odezvu na bodové i rozsahové vyhledávání.

Pokud by byla potřeba aplikace, kde by byl kladen důraz na velmi rychlé vkládání prvků a operace hledání by zde nebyla podstatná, lze na základě znalostí a výsledků testu č. 3 doporučit datovou strukturu PST a její variantu A na vkládání prvků, která vychází z datové struktury Treap. V případě, že by počet vkládaných prvků byl až v řádech miliónů, lze doporučit (opět na základě testu č. 3) všechny 3 testované datové struktury, kdy se poměr časových výsledky téměř shoduje.

Pokud by byla potřeba aplikace, kde by byl v 1.fázi kladen důraz na velmi rychlé vkládání a ve 2.fázi (která by bezprostředně nenásledovala) velmi rychlé vyhledávání, lze doporučit ty datové struktury jako v předešlém odstavci s tím, že před 2.fází by došlo ke kompletní prohlídce zvolené datové struktury a z vybraných dat by byl vytvořen nový *k*-d strom. Platilo by tedy to, co v prvním odstavci této kapitoly.

6 Závěr

Cílem této práce bylo představit vybrané datové struktury pro uchovávání multidimenzionálních dat, implementovat je ve zvoleném programovacím jazyce a navrhnout oblast použití.

Na velmi malém počtu testovacích dat byl v analytické části ukázán princip vybraných pěti datových struktur (k-d strom, prioritní vyhledávací strom, quad strom, oktálový strom a grid file).

V jazyce UML byly poté navrženy třídy pro pozdější implementaci.

Implementační část zahrnovala naprogramování datových struktur podle vytvořených tříd a vytvoření základní GUI aplikace. Tato aplikace využívá jak testovací data z analýzy, tak reálná geografická data ČR. Bohužel z časových důvodů a podle mě velkého rozsahu zadání nebyly zcela dokončeny struktury oktálový strom a grid file.

V testovací části byla z důvodů nevyhovujícího malého počtu reálných dat použita i data náhodně generovaná. Na základě testů byla doporučena oblast použití datových struktur.

Datovou strukturu grid file bych svým velkým rozsahem v kombinaci s blokovým souborem navrhl jako téma pro samostatnou práci, která by jistě pokryla rozsah práce bakalářské.

Seznam použitých zdrojů

1. T. GOODRICH, Michael, TAMASSIA, Roberto. *Algorithm Design*. [s.l.] : John Wiley & Sons, Inc. , 2002. ISBN 0-471-38365-1.
2. MCCREIGHT, Edward. *Priority Search Trees* [online]. 1985 [cit. 2009-08-20]. Dostupný z WWW: <http://www.cs.princeton.edu/courses/archive/spr04/cos423/handouts/priority%20search%20trees.pdf>
3. POKORNÝ, Jaroslav . *Prostorové datové struktury a jejich použití k indexaci prostorových objektů* [online]. [2009] [cit. 2009-08-20]. Dostupný z WWW: <http://gis.vsb.cz/GIS_Ostrava/GIS_Ova_2000/Sbornik/Pokorny/Referat.htm>.
4. W. MOORE, Andrew . *An intoductory tutorial on kd-trees trees* [online]. 1991 [cit. 2009-08-20]. Dostupný z WWW: <http://www.autonlab.org/autonweb/14665/version/2/part/5/data/moore-tutorial.pdf?branch=main&language=en>.
5. J. RADWIN, Michael. *Priority Search Tree Demo trees* [online]. 1997 [cit. 2009-08-20]. Dostupný z WWW: <http://www.cs.brown.edu/courses/cs252/misc/proj/src/Spr96-97/mjr/>.
6. D. BROWN, Amalia. *Design and Analysis of Multidimensional Data Structures* [online]. 1994 [cit. 2009-08-20]. Dostupný z WWW: <http://www.tdr.cesca.es/TESIS_UPC/AVAILABLE/TDX-0207107-173857//01ADB01de01.pdf>.
7. *Introduction to kd-trees trees* [online]. [cit. 2009-08-20]. Dostupný z WWW: <http://www.cse.iitb.ac.in/~sharat/previous/courses/spatial/cmsc420/pbasic.pdf>.
8. PELIKÁN, Josef, TOBOLA, Petr. *Datové struktury pro prostorové vyhledávání* [online]. 2003 [cit. 2009-08-20]. Dostupný z WWW: <http://www.fi.muni.cz/~sochor/PA010/Slajdy/SpaceSearch.pdf>.
9. *Octree* [online]. 2009 [cit. 2009-08-20]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Octree>.
10. *K-d tree* [online]. 2009 [cit. 2009-08-20]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Kd-tree>.
11. *Treap* [online]. 2009 [cit. 2009-08-20]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Treap>.
12. CECILIE KJELDSEN, Linda. *An Application of the Priority Search Tree* [online]. 2005 [cit. 2009-08-20]. Dostupný z WWW: <http://www.it.hiof.no/prosjekter/hoit/html/nr2_05/linda.html>.
13. KAVIČKA, Antonín. *Treap - sylaby přednášek předmětu „Datové struktury a algoritmy*.