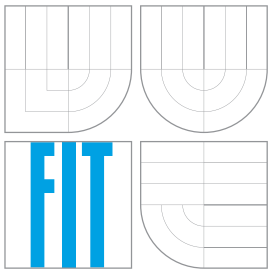
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DEMONSTRACE INDEXAČNÍHO ALGORITMU KD STROM A JEHO DERIVÁTŮ

DEMONSTRATION OF AN INDEXING ALGORITHM KD TREE AND ITS DERIVATIVES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ FOUKAL

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. DUŠAN KOLÁŘ,

BRNO 2016

Abstrakt

Tato práce se zabývá rozborem, návrhem a implementací aplikace pro výuku indexačního algoritmu k-D Strom a jeho derivátů. Dále se zabývá testováním aplikace, popisem jejího použití a příklady nad konkrétními daty. Aplikace také umožňuje krokování algoritmů a jejich zobrazení v grafickém uživatelském rozhraní. Při vývoji byl kladen důraz na spustitelnost aplikace ve všech majoritních internetových prohlížečích.

Abstract

This thesis deals with analysis, designing and implementation of an application for teaching of indexing algorithm kD Tree and its derivates. Also, it deals with testing the application, describing of its use and including examples with exact data. Application as well offers a possibility of running of algorithms step-by-step and their visualization with a graphical user interface. During development, the emphasis was on executability of the application in all major web browsers.

Klíčová slova

indexovací algoritmus, k-D Strom, adaptivní k-D Strom, quadtree, demonstrace, Javascript, CSS, HTML5, podpora výuky, Selenium

Keywords

indexing algorithm, kD Tree, adaptive kD Tree, quadtree, demonstration, JavaScript, CSS, HTML5, support of education, Selenium

Citace

FOUKAL, Tomáš. Demonstrace indexačního algoritmu kD strom a jeho derivátů. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kolář Dušan.

Demonstrace indexačního algoritmu kD strom a jeho derivátů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Dušana Koláře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Tomáš Foukal
17. května 2016

Poděkování

Poděkování patří panu doc. Dr. Ing. Dušanu Kolářovi za odborné vedení, cenné rady a ochotu.

© Tomáš Foukal, 2016.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod 3

2 k-D Strom 4
2.1 Algoritmus . . . 4
2.2 Operace . . . 4
2.2.1 Konstrukce stromu . . . 5
2.2.2 Vyhledávání . . . 5
2.2.3 Přidávání bodů . . . 5
2.3 Popis k-D Stromů implementovaných v aplikaci . . . 5
2.3.1 k-D Strom . . . 6
2.3.2 Adaptivní k-D Strom . . . 7
2.3.3 Quadtree . . . 7

3 Návrh 10
3.1 Požadavky . . . 10
3.2 Programovací jazyk . . . 10
3.2.1 Volba aplikačního rámce . . . 10
3.3 Grafické uživatelské rozhraní . . . 11
3.3.1 Limity pravého pole se zobrazením stromu . . . 12
3.4 Objektově orientovaný návrh . . . 12

4 Implementace 14
4.1 HTML prvky . . . 14
4.1.1 Canvas . . . 14
4.1.2 Div . . . 15
4.2 JavaScript objekty a podstatné metody . . . 15
4.2.1 Objekt Node . . . 15
4.2.2 Objekt Zone . . . 15
4.2.3 Objekt KDTree . . . 16
4.2.4 Objekt GUI . . . 17
4.2.5 Manipulační programy událostí . . . 17

5 Testování 18
5.1 Testování pomocí nástroje Selenium . . . 18
5.1.1 Nástroj Selenium . . . 18
5.1.2 Vytvořené testy . . . 19
5.2 Zátěžové testování . . . 19
5.2.1 Výsledky . . . 19

5.2.2 Testovací platforma 19

6 Použití aplikace 21
6.1 Exportování dat do souboru 21
6.2 Nahrávání souborů s daty do aplikace 21
6.3 Barevné zvýraznění při dělení zón 22

7 Příklady 23
7.1 Syntetické příklady 23
7.1.1 Sinusoida 23
7.1.2 Shluk bodů v jednom místě 25
7.1.3 Shluky bodů ve dvou místech 26
7.1.4 Rovnoměrné rozložení bodů 27
7.1.5 Degradace stromu a jeho přestavba 29
7.2 Příklady na reálných datech 30
7.2.1 Rovnoměrně osídlená oblast 30
7.2.2 Nerovnoměrně osídlená oblast 32
7.2.3 Obličej Lenny 34

8 Závěr 36

Literatura 37

Přílohy 38
Seznam příloh 39

A Obsah CD 40

Kapitola 1

Úvod

Tento dokument se zabývá tvorbou online aplikace, která umožní demostraci algoritmu k-D Strom a jeho derivátů. Tato aplikace by měla sloužit především k praktické grafické ukázce, jak jednotlivé algoritmy pracují. Toho bude docíleno pomocí interaktivity s uživatelem programu, který si bude moci zadávat vlastní vstupy, postupovat jednotlivými algoritmy krok po kroku a zobrazovat si potřebné informace k pochopení problému tvorby k-D Stromu.

Podobné aplikace lze již na internetu nalézt, jedná se však většinou o implementace, které se soustředí jen na jeden druh k-D Stromu s nízkou interaktivitou (data volena náhodně, algoritmus vykonán najednou, nemožnost rozebrat algoritmus na jednotlivé kroky). Tyto nedostatky se pokusí nahradit výsledek této bakalářské práce, který bude popsán v následujícím dokumentu.

Největší přínos a také důvod, proč jsem si toto téma vybral je, že se výstup práce nezaloží do knihovny, ale bude skutečně využíván. Aplikace bude používána pro výuku předmětů na FIT VUT, takže na konec splní účel, kvůli kterému byla vytvořena. Nebude se jednat o „podesáté“ vypsanou bakalářskou práci, ale o ucelený, využívaný nástroj.

Kapitola 2 se zabývá obecným popisem problematiky k-D Stromů, popisuje některé operace nad ním a zmiňuje se o jeho derivátech, které byly do aplikace začleněny.

3. kapitola se věnuje návrhu aplikace, hodnotí požadavky na aplikaci, volí implementační jazyk, jeho aplikační rámec a knihovny a analyzuje grafické uživatelské rozhraní.

Kapitola číslo 4 popisuje implementaci programu, nejdůležitější HTML prvky, JavaScript objekty a jejich metody.

Kapitola 5 popisuje testování aplikace pomocí nástroje Selenium a zátěžové testování programu.

6. kapitola vysvětluje použití aplikace, věnuje se práci se soubory a barevnému zvýrazňování pro výukové účely.

V 7. kapitole lze najít konkrétní příklady použití aplikace a jejich popis s příslušnými obrázky.

Závěrečná kapitola rekapituluje záměr práce, hodnotí výsledky práce a navrhuje další možnosti vývoje aplikace.

Kapitola 2

k-D Strom

k-D Stromy (nebo také kD-trees, k-D-trees, nebo k-d trees) [2] byly představeny Jonem Bentleyem v roce 1975. Jedná se o datovou strukturu pro ukládání bodů ve vícedimenzionálním prostoru. Práce ve více dimenzích se tedy odráží na názvu „k-D Strom“, který zobecňuje názvy typu „2-D Strom“, „3-D Strom“, atd.

Při jejich tvorbě je vstupem množina bodů v k-dimenzionálním prostoru. Tento prostor je poté dělen nadrovinami na poloprostrory. Tyto nadroviny jsou často rovnoběžné s jednou z os prostoru, objevují se však i algoritmy, ve kterých jsou nadroviny vedeny tak, aby co nejoptimálněji rozdělovaly daný prostor (tento způsob se odráží ve vyšší složitosti algoritmu pro výběr vektoru). Těmito způsoby tedy rozdělíme daný prostor do menších buněk, které obsahují malý (nebo nulový) počet vstupních bodů. Poté lze přistupovat k jednotlivým bodům na základě jejich pozice, což zvyšuje rychlost při následném vyhledávání dat.

2.1 Algoritmus

k-D Strom patří do skupiny binárních stromů. Pří tvorbě stromu dělí nadrovina daný k-D prostor na dva poloprostory. Hrana nadroviny musí být vedena skrz vhodně zvolený index. V základní verzi algoritmu k-D Strom se v daném indexu nachází bod, který se stává uzlem k-D Stromu, vzniklé dva poloprostory lze brát jako budoucí podstromy vycházející z tohoto uzlu. V adaptivní verzi je hranou dělící poloroviny pouze přímka, která bodem procházet nemusí a proto jsou body až listovými uzly stromu. Zároveň se v jedné polorovině budou nacházet uzly, které mají hodnotu v dané dimenzi nižší, než má uzel, přes který vedla dělící nadrovina. V druhé polorovině budou uzly s vyšší hodnotou.

Toto dělení prostoru se provádí tak dlouho, dokud se nedosáhne požadovaného počtu uzlů v polorovině. Osa, podle které je dělící nadrovina vedena, může být určena několika způsoby. Častým (a v této práci i implementovaným) způsobem je střídání jednotlivých os (Round–robin), může se však volit i osa, která je nejdelší, nebo se osy mohou v určitém poměru střídat. Pravidelné střídání dělících dimenzí je ale ze subjektivního hlediska nejintuitivnější.

2.2 Operace

Následující sekce se zabývá jednotlivými operacemi nad k-D Stromem – popisuje algoritmy, zmiňuje problémy a nabízí některá řešení. Pro lepší představu jsou operace doplněny o pseudokód.

2.2.1 Konstrukce stromu

Jak již bylo zmíněno výše, probíhá samotná konstrukce stromu dělením prostoru pomocí nadrovin. Jak ale zvolit správný bod, kterým bude nadrovina procházet? Nabízí se několik možností. Lze použít hodnotu, která je nejbližší průměrné hodnotě v dané dimenzi, můžeme bod zvolit náhodně, nebo například použít medián v dané dimenzi.

Následující algoritmus vytváří strom rekurzivně pomocí posledně zmíněné možnosti.

```
uzel kDStrom(PoleBodu body, int pocetDimenzi, int aktualniDimenze)
    aktualniDimenze ++;
    aktualniDimenze = aktualniDimenze % pocetDimenzi;

    Seřaď body od nejnižšího po nejvyšší podle dané dimenze
    Najdi medián z body v aktualniDimenze
    Vytvoř uzel uzel a vlož do něj medián

    uzel.levyPodstrom = kDStrom(body nižší, pocetDimezi, aktualniDimenze);
    uzel.pravyPodstrom = kDStrom(body vyšší, pocetDimezi, aktualniDimenze);

    Vrať uzel
```

Algoritmus 1: Rekurzivní vytvoření k-D Stromu

Takto vytvořený strom je vyvážený.

2.2.2 Vyhledávání

Při vyhledávání se začíná v kořenovém uzlu. Známe hodnotu vyhledávaného uzlu v jednotlivých dimenzích a ty porovnáváme s hodnotami uzlů v procházených dimenzích. Stejně, jako u binárních stromů se buď vydáváme levým, nebo prvým směrem, podle toho, zda je hodnota vyšší, nebo nižší, než v právě procházeném uzlu.

2.2.3 Přidávání bodů

Přidávání bodů se provádí stejně, jako u jiných binárních stromů. Postupně procházíme strom od kořene a v dané dimenzi se rozhodujeme, zda se vydáme levým, nebo pravým podstromem. Jakmile se dostaneme do listového uzlu, rozhodneme, kam nový uzel přidáme, a uzel vytvoříme. Zároveň nastavujeme odkazy z rodičovskéhu uzlu na potomka a zpět. Je nutné připomenout, že při procházení pravidelně měníme dimenze, které porovnáváme. Takovéto přidávání uzlů může strom vyvést z rovnováhy a způsobit jeho nevyváženost. Při přílišné degradaci stromu je vhodné ho celý, nebo jeho degradovanou část, vytvořit znovu.

2.3 Popis k-D Stromů implementovaných v aplikaci

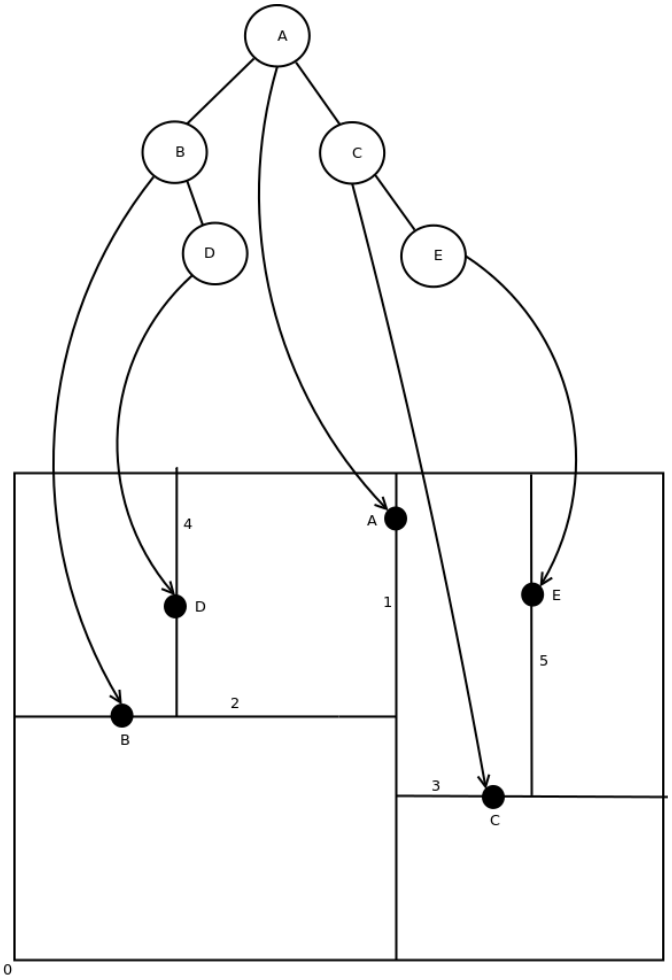
V aplikaci byly implementováni tři zástupci algoritmů pro dělení roviny. Byly voleny tak, aby se jeden od druhého v určitém ohledu zásadně lišil. Následující podkapitoly vysvětlují postup algoritmu, který stromy tvoří a také rozdíly mezi jednotlivými druhy.

2.3.1 k-D Strom

Jedná se o nejzákladnější algoritmus k-D Strom. Hrana dělící roviny je volena jako bod, který má hodnotu rovnu (nebo hodnotu blízkou) mediánu v dané dimenzi. Pokud se medián nachází mezi dvěma body (v tom případě se nachází přesně v polovině), je dělící index určen náhodně v jednom z těchto dvou bodů v poměru 1:1.

Příklad

Na obrázku 2.1 můžeme vidět ukázku typického k-D Stromu. Data jsou vložena v dvoudimenzionálním prostoru. Rovina, která bude dělena, se před započetím dělení rozkládá v celé ploše prostoru. V prvním kroku je rovina rozdělena na dvě poloroviny, rovnoběžně s osou Y. Dělící úsečka je v diagramu označena číslicí 1. Zároveň je vytvořen uzel A, který bude i uzlem kořenovým. Dále se dvě vzniklé oblasti rozdělí rovnoběžně s osou X, usečky (2, 3) jsou vedeny skrz body B a C. Strom se rozšiřuje o další dva uzly. Vznikají tím čtyři oblasti, z nichž pouze dvě obsahují body a dvě jsou prázdné. V dalším kroku algoritmu jsou dvě zbylé neprázdné roviny opět rozděleny (4, 5). Všechny oblasti jsou prázdné, strom je plně vytvořen a algoritmus končí.



Obrázek 2.1: Příklad vytvořeného k-D Stromu

2.3.2 Adaptivní k-D Strom

Algoritmus adaptivního k-D Stromu se od algoritmu základního k-D Stromu liší tím, že není nutné, aby dělící čára procházela bodem. To má za důsledek, že uzly (kromě listových uzlů) jsou dělené oblasti a ne body, jako ve výše zmíněné verzi algoritmu. Body jsou uloženy v listových uzlech. Tato verze nabízí možnost stanovit počet listových uzlů, které budou obsaženy v jedné oblasti. V aplikaci však bylo navrženo, že v jedné oblasti se mohou nacházet maximálně dva body.

Příklad

Obrázek 2.2 je ukázkou adaptivního stromu. Před započetím algoritmu jsou body v jediné oblasti. Tato oblast je tedy kořenovou oblastí stromu. Oblast je v prvním kroku rozdělena na dvě podoblasti podle osy Y. Tím vznikají ve stromu dva nové uzly. První obsahuje pět bodů, druhý tři. Dále jsou podoblasti opět roděleny, přičemž v několika vzniklých oblastech (1, 2, 3) již bylo dosáhnuto požadovaného počtu uzlů v oblasti. Tyto části tedy nebudou dále děleny. Jednu z oblastí je však nutné naposledy rozdělit, protože obsahuje tři body. Vznikají oblasti 4 a 5, algoritmus končí. V tomto případě si můžeme všimnout, že pokud by byla první dělící úsečka vhodněji zvolena, mohla rozdělit kořenovou oblast na dvě podoblasti, přičemž každá z nich by obsahovala čtyři body. Stačily by tedy jen další dva kroky algoritmu pro dokončení dělení. Strom by byl navíc vyvážený.

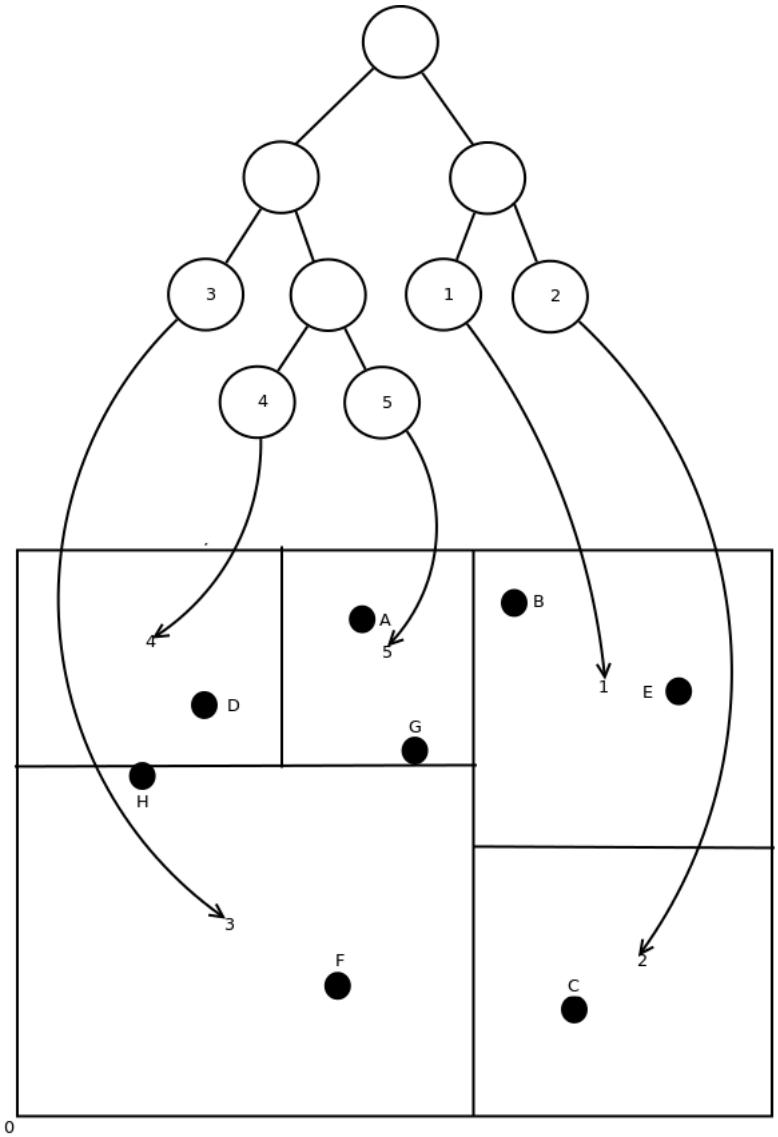
2.3.3 Quadtree

Quadtree, jak už název napovídá, nerozděluje oblasti do dvou, ale do čtyř podoblastí. V základní verzi toho algoritmu není nutné vyhledávat nejlepší index, na kterém se prostor rozdělí, nemusí ani procházet žádným bodem. Jako dělící index se jednoduše určí polovina v každé dimenzi. Existují však i verze (např. „point quadtree“), které (podobně jako u k-D Stromu) využívají k určení polohy dělících čar body. Stejně jako u adaptivního algoritmu, může být určeno, kolik bodů se maximálně v jedné oblasti může nacházet. V aplikaci byla implementována verze, kdy je tato hodnota stanovena na maximálně jeden bod.

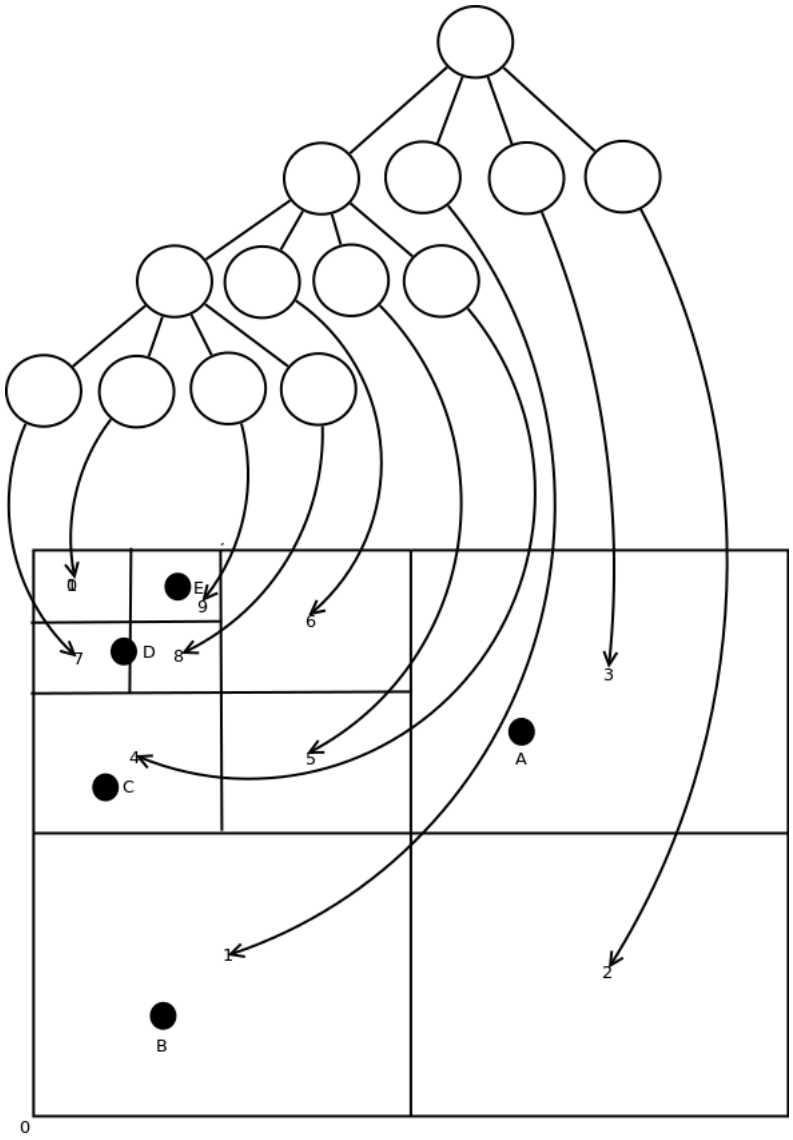
Existují i vice než dvoudimenzionální verze toho algoritmu, například ve třech dimenzích se můžeme setkat s názvem „octree“ – zde se používají tři indexy dělení a vzniká osm nových prostorů. Tento algoritmus se hojně využívá v 3D grafice [5] (např. počítačových hrách).

Příklad

Na obrázku 2.3 můžeme vidět vytvořený quadtree. Hlavní vlastností tohoto algortimu je, že nehledě na rozložení bodů v ploše, dělí plochu vždy na čtyři stejně velké oblasti. Tato vlastnost je z příkladu velmi zřejmá, můžeme si například všimnout oblasti 2, která je po prvním dělení úplně prázdná. V každém kroku vznikají ve stromu čtyři další uzly reprezentující oblasti. V závislosti na rozložení bodů může být tedy strom značně nevyvážený (tak jako v příkladu). Algoritmus dělení končí v okamžiku, kdy každá oblast obsahuje pouze jeden bod.



Obrázek 2.2: Příklad vytvořeného adaptivního stromu



Obrázek 2.3: Příklad vytvořeného quadtree

Kapitola 3

Návrh

Následující kapitola se zabývá návrhem celé aplikace. Budou v ní vyhodnoceny požadavky na aplikaci, kritéria, které musí aplikace splnit a další informace pro tvorbu programu.

3.1 Požadavky

Práce byla zadána jako bakalářská práce, jejímž výsledkem má být výuková aplikace. Již z tohoto faktu vyplývá, že aplikace by měla být intuitivní a jednoduchá na ovládání. Tím pádem bude i člověk, který není s problematikou k-D Stromů plně seznámen, schopen pochopit základní vlastnosti implementovaných algoritmů.

Aplikace by měla fungovat na rozličných platformách a prohlížečích. Zároveň by nemělo být třeba mít nainstalovaný žádný speciální software nebo rozšíření pro prohlížeč. Program by měl být z uživatelského pohledu rychlý a bezproblémový. Další důležitou vlastností je maximální interaktivita, uživateli by měla nabízet odpovědi na otázky typu „Co se stane, když udělám...“

Aplikace bude mít možnost provádět jednotlivé algoritmy krok po kroku, případně se stejnými daty provádět každý z druhů a porovnávat výsledné stromy. Dále také data uložit a později vytvořený soubor zpět nahrát a znovu interpretovat.

3.2 Programovací jazyk

Pro implementaci budou využity standardní webové technologie jako $HTML5, CSS3, \ldots$ Při tvorbě projektu se budeme maximálně snažit využít element *Canvas*, který byl navrhnut pro vykreslování grafiky za pomoci skriptování. Pro implementaci interaktivity a logiky práce s k-D Stromy bude využit jazyk *JavaScript*. Jedná se o multiplatformní, interpretovaný, objektově orientovaný jazyk, který je velmi vhodný pro naše užití, protože je akceptován všemi majoritními internetovými prohlížeči.

3.2.1 Volba aplikačního rámce

Angular

V současnosti velmi rozšířený aplikační rámec, vhodný pro tvorbu komplexních webových aplikací. Je podporován společností Google a může se pochlubit velkou uživatelskou základnou i komunitou. Aplikační rámec využívá velmi abstraktní prvky, jejichž implementaci nemusí být snadné pochopit.

Total.js

Jedná se o aplikační rámec postavený na *Node.js*. Má kvalitně zpracovanou dokumentaci a spoustu příkladů pro použití v konkrétních situacích. Je vhodný pro aplikace, kde počítání neprobíhá po stažení na straně klienta, ale přímo na serveru. Proto je vhodný zvláště pro serverové aplikace.

jQuery

Knihovna s otevřeným zdrojovým kódem s rozsáhlou komunitní podporou, kvalitní dokumentací a velkým množstvím návodů. Počet dostupných návodů napomáhá i začátečníkovi v jazyce *JavaScript* pochopit způsob programování, propojení jazyka s *HTML*, apod. Můžeme pominout cenový aspekt. Lze ho využít zdarma a jeho zdrojový kód je otevřený.

Ve fázi návrhu projektu byla pro ulehčení tvorby zvolena knihovna *jQuery* kvůli její jednoduchosti a silné podpoře komunity. Tanto závěr se však může během vývoje změnit. Je totiž otázkou, jak nám mohou aplikační rámce ve vývoji podobné aplikace pomoci. Předpokládá se, že při tvorbě aplikace bude využito minimum grafických elementů. Pro tvorbu logiky by v tom případě mohl stačit samotný programovací jazyk bez jakýchkoli rozšíření.

3.3 Grafické uživatelské rozhraní

Jak vyplývá z výše uvedených požadavků, aplikace má být jednoduchá, pochopitelná. K této vlastnosti velkým dílem přispívá grafické rozhraní.

Již při načtení samotné stránky vidíme, že rozhraní je bez zbytečných informací nebo prvků – viz. obrázek 3.1. Skládá se z šesti prvků:

1 Prvního horního panelu pro volbu požadovaného algoritmu

2 Druhého horního panelu pro ovládání algoritmu, který se dále skládá z:

- *random node + step* – náhodné přidání nového bodu a vykonání jednoho kroku algoritmu
- *random node* – náhodné přidání bodu
- *play* – spuštění automatického vykonávání algoritmu krok po kroku
- *step* – vykonání jednoho kroku algoritmu
- *build tree* – vykonání algoritmu až do konce
- *clear tree* – vymazání vytvořených struktur, připravení dat pro nový běh algoritmu

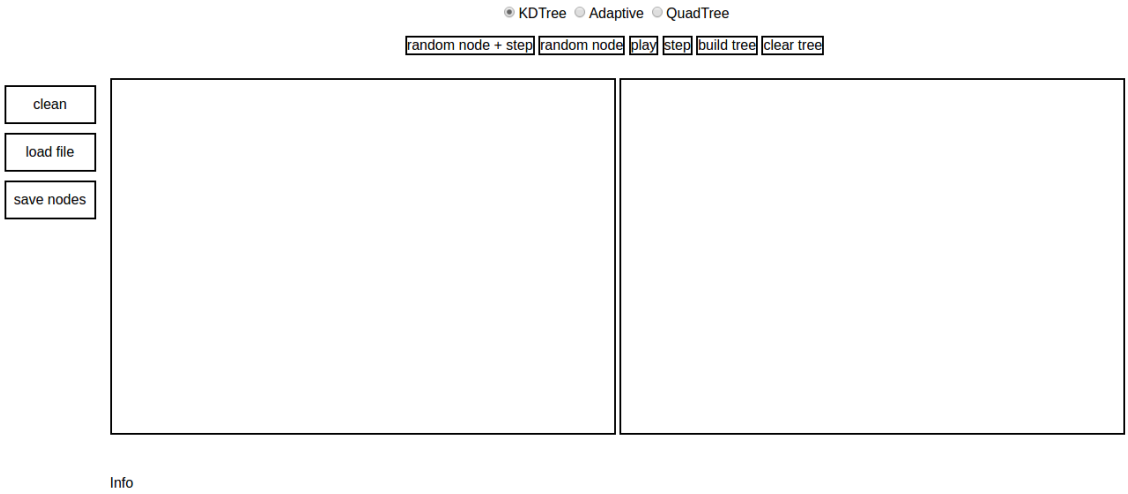
3 Levého panelu pro práci s daty, který se skládá z:

- *clean* – smazání všech dat na stránce
- *load file* – načtení souboru od uživatele s daty
- *save file* – uložení dat do souboru a nabídnutí uživateli soubor stáhnout

4 Levého, klikatelného pole se zobrazením dat

5 Pravého pole se zobrazením vytvářeného stromu

6 Spodního pole s výpisem informací



Obrázek 3.1: Aplikace po načtení stránky

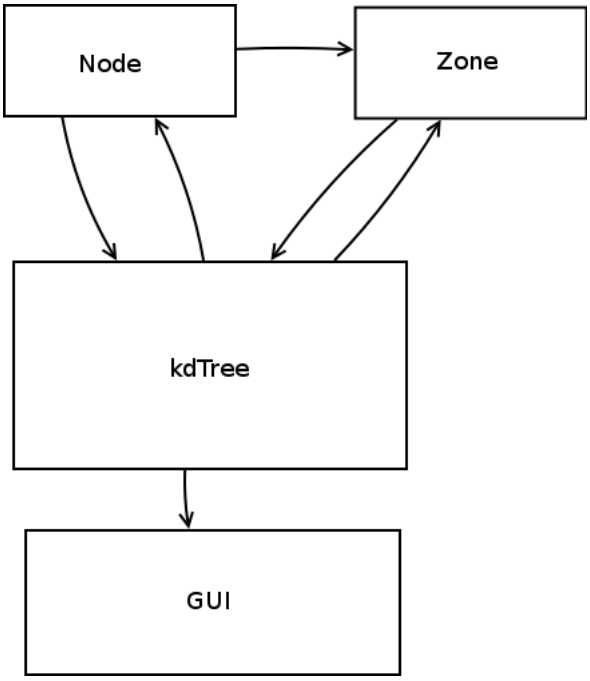
3.3.1 Limity pravého pole se zobrazením stromu

Pravé pole se zobrazením vytvářeného stromu má limity, které když jsou přesaženy, není toto pole vytvářený strom schopno zobrazit. Pro korektní zobrazování stromu může strom dosáhnout maximální výšky devět uzlů. To pro vyvážený strom znamená, že se může skládat až z 2^8 uzlů. Tato hodnota je pro výukové účely plně dostačující. U stromů s více uzly nebo s výraznou degradací je pak více informačně hodnotné levé, klikatelné pole.

3.4 Objektově orientovaný návrh

Objektově orientovaný návrh zahrnuje čtyři primární části (viz. obrázek 3.1):

- *kdTree* – nejdůležitější objekt, uchovává logiku programu, komunikuje s ostatními objekty a částečně je řídí
- *Zone* – objekt reprezentující oblasti, které jsou vytvářeny při dělení plochy, uchovává data o oblasti a odkazy na jednotlivé uzly, které se v zóně nacházejí
- *Node* – objekt zastupující uzel ve stromu, obsahuje informace o uzlu, jeho polohu v prostoru, odkazuje na své rodičovské uzly a na své potomky
- *GUI* – objekt, zprostředkovává komunikaci s uživatelem aplikace



Obrázek 3.2: Schéma objektově orientovaného návrhu

Kapitola 4

Implementace

Jak již bylo výše zmíněno, aplikace byla vyvíjena s využitím standardních webových technologií. Důraz byl kladen na rychlost běhu aplikace. Následující podkapitoly popisují nejpodstatnější prvky, které jsou kostrou programu.

4.1 HTML prvky

4.1.1 Canvas

Tento prvek je využíván pro vykreslování grafiky. Je úzce spjat se skriptovacím jazykem *JavaScript*, pro vykreslení grafiky na *Canvas* je třeba skriptu. Obsahuje metody pro vykreslování jednoduchých objektů jako je čára, kruh, mnohoúhelník, nebo text. Dokáže však vykreslit i vložený rastrový obrázek.

Práce s prvkem je rychlá a jednoduchá, například pro vykreslení čáry z bodu se souřadnicemi x_1, y_1 do bodu x_2, y_2 stačí čtyři příkazy:

```
context.beginPath();
context.moveTo(x1, y1);
context.lineTo(x2, y2);
context.stroke();
```

Algoritmus 2: Vykreslení čáry do *Canvasu*

Pro jeho vytvoření se využívá HTML element typu *Canvas*. Důležitým parametrem, který byl v aplikaci využit, je *z-index* :. Tímto parametrem se stanoví, kde se canvas nachází na ose Z, to znamená překrytí jednotlivých prvků *Canvas*. Tato vlastnost byla s úspěchem využita, v levém klikacím poli. Při načtení stránky jsou totiž v této oblasti vytvořeny dva prvky *Canvas*, přičemž na *Canvas* v popředí jsou vykreslovány dělící čáry a na vzdálenějším prvku typu *Canvas* jsou barvou vyznačovány oblasti, které se rozdělily. O obarvování dělených oblastí více v kapitolách Použití a Příklady.

Na stránce se nachází i třetí prvek typu *Canvas* a to na pravém poli vykreslujícím tvořený strom. Je zde využit pro vykreslování čar spojujících jednotlivé uzly.

4.1.2 Div

Prvek, který má (podobně jako *Canvas*) v programu velmi důležitou grafickou roli. I když se jedná o *tag* bez sémantického významu [3], může mu být skrze *CSS* přidělen grafický vzhled.

Těchto vlastností je s úspěchem využívno při vykleslování bodů, které si uživatel zadává do levého interaktivního pole. Při kliknutí do toho prostoru je na souřadnicích kliknutí vytvořen *tag div*, který má jako parametr *class* nejen třídu určující jeho vlastnosti ve stylovém předpisu, ale i hodnotu parametru id nově vzniknutého bodu. Toto je použito při kliknutí na už vytvořený bod, z tagu se zjistí id kliknutého bodu a program zareaguje vypsáním informací o bodu do spodního informačního pole a také zvýrazněním bodu v pravém poli s vytvářeným stromem.

Dále je tag *div* využit v pravém stromovém poli pro vykreslování uzlů stromu. Například při každém kroku algoritmu k-D Strom je vytvořen *div*, který má parametr *class* nastaven na stejnou hodnotu jako parametr id použitého uzlu. Tím je logicky propojen se značkami *div* v levém klikatelném poli a je možné ho v případě potřeby zvýraznit.

4.2 JavaScript objekty a podstatné metody

4.2.1 Objekt Node

Objekt reprezentující uzel. Při vytváření objektu typu *Node* je třeba stanovit polohu bodu v rovině (parametry x_in a y_in) a dále pak id bodu (parametr id_in). Hodnoty se pak uloží do vnitřních proměnných a je možné k nim přistupovat pomocí příslušných metod. Další vnitřní proměnné jsou x_second a y_second, určující polohu bodu v pravém stromovém poli, a proměnné $parent$, $left$ a $right$, které po nastavení vytváří strukturu stromu.

4.2.2 Objekt Zone

Objekt reprezentující oblast. Pro vytvoření zástupce tohoto objektu je třeba zadat začátek oblasti v ose x, konec oblasti v ose x, začátek oblasti v ose y, konec oblasti v ose y, barvu oblasti, dimenzi, podle které bude oblast dělena, a dále pak začátek oblasti v pravém stromovém poli, konec oblasti v pravém stromovém poli a stupeň zanoření. Všechny tyto informace jsou uloženy do vnitřních proměnných a jsou dále využívány při dělení oblastí, jejich vykreslování a obarvování (více informací v kapitole Příklady). Navíc obsahuje seznam bodů, které se v oblasti nachází.

Důležitými metodami jsou tyto:

- $check\_if\_in\_zone(x, y)$ – metodě jsou zaslány souřadnice a ta ověří, jestli se souřadnice nachází v dané zóně (oblasti)
- $zone\_contain\_\{one, two\}$ – zjišťuje, kolik bodů se v zóně nachází (důležité pro algoritmus adaptive a quadtree)
- $zone\_empty$ – určuje, zda je zóna prázdná (důležité pro algoritmus k-D Strom)
- $color\_zone$ – obarvuje zónu příslušnou barvou

4.2.3 Objekt KDTree

Nejdůležitější část programu. Obsahuje logiku, která komunikuje s objekty uzlů a zón, nastavuje je, distribuuje uzly do zón a komunikuje s objektem grafického uživatelského rozhraní. Při vytvoření tohoto objektu se jako jediný parametr předává odkaz na objekt *GUI*.

Vnitřními proměnnými jsou:

- *tree_type* – implicitně nastavená na $KDTREE\_SET$, určující, který z algoritmů se bude provádět
- *zones* – pole obsahující odkazy na objekty zón, které se v programu nachází
- *nodes* – pole obsahující odkazy na objekty vytvořených uzlů
- *node_count* – proměnná, ze které nově vytvořené uzly získávají své id

Za zmínku stojí také nejdůležitější metody:

- *initialize_KDTree* – vytváří a nastavuje první zónu pokrývající celou plochu levého interaktivního pole, v případě potřeby volá metodu objektu GUI pro vykreslení zóny
- *load_info_and_write* – volána při kliknutí na bod v klikacím poli, připravuje informační výpis a volá metody *GUI* pro jeho zobrazení
- *build* – spuštěna při kliknutí na tlačítko *buildtree*, volá metodu pro vytváření stromu, dokud není strom plně vytvořen
- *process_nodes* – nastavuje vnitřní proměnné uzlů (*parent*, $left$, *right*), tím logicky vytváří strom
- $\{quadtree, median, adaptive\}\_step$ – metody vykonávající jeden krok algoritmu quadtree, k-D Strom nebo jeho adaptivní verzi
- *step* – zjišťuje, která metoda pro vykonání kroku z výše zmíněných bude vykonána, využívá vnitřní promeěnnou tree_type
- $create\_related\_zones\_\{KDTree, adaptive, quadtree\}$ – dělí zónu a vytváří nové zóny podle indexu, nebo uzlu, který je předán, nově vytvořené zóny nastavuje, plní body a volá metodu objektu *GUI* pro jejich vykreslení
- $find\_index\_to\_cut$ – hledá a vrací v zadané dimenzi index, kde bude dělená zóna rozdělena
- $find\_median\_node$ – vyhledává v předaném poli index, který má hodnotu mediánu, v případě, že se medián nachází mezi dvěma indexy, volí jeden z nich náhodně v poměru 1:1
- *sort_by_dimension* – řadí pole uzlů podle zadané dimenze od nejmenšího po největší
- *all_zones_contain_one*, *all_zones_contain_two*, *all_zones_empty* – sada metod zjišťujících, zda všechny zóny obsahují jeden, dva, nebo žádné uzly, tyto metody se využívají pro ukončení algoritmů
- $find\_zone\_where\_add$ – metoda volaná při vytvoření nového bodu, zjišťuje, do které zóny bude bod spadat
- *create_new_node* – metoda volaná pro vytvoření nového uzlu, vytváří nový objekt typu *Node*, nastavuje ho a přidává ho do příslušné zóny

4.2.4 Objekt GUI

Objekt *GUI* zastává roli pro interpretaci algoritmů a komunikaci s uživateli. Obsahuje proměnné reprezentující kontexty pro generování výstupů na *Canvas*. *Context*1 a *Context*2 pro levé klikací pole, *Context*3 pro pravé pole tvorby stromu.

Metody, které zajišťuji zobrazování grafických informací, jsou:

- $draw\_info\_adaptive$ – vypisuje do spodního informačního panelu id a souřadnice bodu, na který uživatel kliknul
- $draw\_info\_kdtree$ – vypisuje do spodního informačního panelu id a souřadnice bodu, dále pak předka uzlu, levého a pravého potomka
- $draw\_info\_error$ – vypisuje úspěšnost operace nahrání souboru s body do aplikace
- $draw\_tree\_zone$ – metoda využívaná u adaptivní verze a u quadtree, vykresluje do pravého stromového pole vytvořené zóny a vztahy mezi nimi
- $draw\_tree\_node$ – metoda využívaná u k-D Stromu, vykresluje uzly stromu do pravého pole a propojuje je příslušnými čarami
- $draw\_line\_quadtree$ – vykresluje úsečky dělící prostor do levého klikacího pole pro algoritmus quadtree, jedná se o dvě čáry, které jsou k sobě vždy kolmé
- $draw\_line\_\{adaptive, kdtree\}$ – vykresluje čáru, která dělí prostor levého klikacího pole, pro algoritmy kdtree a adaptive, informace o poloze a délce úsečky metoda zjišťuje ze zóny, která je dělena a do metody předána
- $initialize\_GUI$ – inicializuje objekt GUI, nastavuje vnitřní proměnné a prvky typu *Canvas* do výchozího stavu
- $clear\_GUI$ – maže vytvořené body a čistí prvky *Canvas* od jakýchkoli vykreslených objektů

4.2.5 Manipulační programy událostí

Pro implementaci klikatelných tlačítek byla využita metoda *.click*. Tato metoda je součástí knihovny *jQuery*. Na každý klikatený objekt byla v kódu navěšena tato metoda, která při kliknutí na element vykoná anonymní funkci. V této funkci je část programu vykonávající potřebnou akci. Například na element s id $step\_forward$ byl navěšen poslouchač událostí, který při kliknutí na element vyvolá anonymní funkci volající metodu objektu *KDTree* *step*().

Kapitola 5

Testování

Po vytvoření aplikace došlo na fázi testování. Následující kapitola se zabývá tím, jak testování probíhalo. Již během tvorby testovací sady bylo zjištěno několik chyb, které byly následně opraveny. Samotné testy pak neodhalily žádnou další, pouze potvrdily bezproblémovou funkčnost programu.

Testovací sada je celá přidána na přiložené CD.

5.1 Testování pomocí nástroje Selenium

Hlavní část testování byla provedena pomocí aplikace Selenium (http://www.seleniumhq.org/).

5.1.1 Nástroj Selenium

Selenium [6] je multiplatformní nástroj vyvinutý v jazyce *Java*, využívaný pro automatické testování webových aplikací. Grafické rozhraní pro program Selenium lze stáhnout jako rozšíření do prohlížeče. V tomto prostředí lze nahrávat určitou aktivitu na internetové stránce a poté ji zpětně interpretovat. Testy se tedy mohou vytvářet například následovně:

Otevři stránku www.fit.vutbr.cz;
Klikni na element s id „search“;
Napiš do elementu s id „search“ text „IMS“;
Klikni na element „input.submit“;
Ověř zobrazení výsledků vyhledávání;

Algoritmus 3: Vytvoření jednoduchého testu v prostředí Selenium

Takto vytvořený test může být uložen, popřípadě exportován do jednoho z jazyků, podporujících Selenium (*C#*, *Java*, *Python*2, *Ruby* [6]) a interpretován, třeba i na vzdáleném serveru.

Tímto způsobem lze vytvořit sadu testů simulujících chování uživatele, který využívá aplikaci. Lze tak i zajistit, že každý z elementů na stránce byl alespoň jednou využit.

5.1.2 Vytvořené testy

Vytvořené testy se nacházejí na přiloženém CD. Byly uchovány v, pro Selenium nativním, formátu *HTML*. Lze je tedy interpretovat jednoduchým nahrátím do *SeleniumIDE*.

V rámci testování aplikace bylo vytvořeno 28 testovacích případů. Testovány byly všechny klikatelné objekty a správné reakce aplikace. Pozornost byla zaměřena na jejich správnou funkcionalitu. Tímto způsobem testovací sada pokryla kritérium pokrytí pro testování GUI *Event Coverage* (vyžaduje, aby testovací sada obsahovala vykonání všech událostí minimálně jednou). Dále byly testovány některé důležité přechody mezi dvojcemi, například přechody mezi jednotlivými druhy algoritmů a kontroly správného chování aplikace. Díky tomu testovací sada alespoň částečně pokryla kritérium pokrytí *Event – iteraction Coverage*. Ke kompletnímu pokrytí tohoto kritéria však nedošlo, protože jednotlivé události jsou na ostatních ve většině případů nezávislé a proto by bylo vytváření úplné sady bezpředmětné a ověřeny byly jen případy, které to vyžadují.

Při vytváření testů byly nalezeny dvě chyby v programu, které byly následně opraveny. Po vytvoření úplné sady testovacích případů se přistoupilo k úplnému testování aplikace. Všech 28 případů bylo vyhodnoceno hodnocením *test pass*, tedy úspěšností testů. Celá sada proběhla úspěšně, lze tedy prohlásit, že aplikace se chová podle požadavků, které jsou na ni kladeny.

5.2 Zátěžové testování

Zátěžové testování bylo prováděno zadáním hraničních dat. Rozměry pole pro zadávání dat jsou 570 pixelů v ose X a 400 pixelů v ose Y. Do prostoru s těmito rozměry lze naskládat maximálně 9 315 bodů. Byl proto vytvořen soubor pro import bodů maximálně pokrývající tuto plochu a nastavující výše zmíněný počet bodů. Soubor je uložen na přiloženém CD pod názvem „zatez“.

Pro měření času stráveného vykonáváním programu byl využit *JavaScript Profiler* prohlížeče [4] *Google Chrome* (https://www.google.com/chrome/browser/desktop/index.html).

5.2.1 Výsledky

Samotné nahrávání zátěžového souboru trvalo 4 593 milisekund, skončilo úspěšně. Dále pak byla vyvolána funkce pro vytvoření celého stromu při provádění algoritmu k-D Strom. Tato akce od jejího začátku po úplné dokončení trvala 13 873 milisekund. Pro adaptivní verzi algoritmu byla naměřena hodnota 18 157 milisekund. Hodnota pro algoritmus quadtree byla 82 581 milisekund. Všechny algoritmy skončily úspěšným vytvořením stromu.

Tyto hodnoty se zdají být dosti vysoké, každopádně je třeba si uvědomit, že se jedná o hraniční a nejnáročnější možný případ. Například u souboru „rovnomerne“ byly naměřené hodnoty 110, 98 a 115 milisekund. Import dat z toho souboru vytvořil 211 bodů. Pro úplné porovnání je záhodno ještě uvést hodnoty, které byly naměřeny pro vykonání jednoho jediného kroku algoritmů po nahrání dat ze zátěžového souboru (216 ms, 442ms, 459ms) a jednoho kroku, po kterém algoritmus skončil (18 ms, 24 ms, 26 ms).

5.2.2 Testovací platforma

Stroj, na kterém byly testy prováděny, má následující parametry:

- Operační systém – Linux Mint 17.3 Rosa
- Verze jádra – 3.19.0-32-generic
- CPU – Intel Core i5-3210M, 3100 MHz, cache: 3072 KB
- Architektura – 64 bit
- RAM – 3838 MB

Využitá verze prohlížeče *Google Chrome* byla 50.0.2661.94 (64-bit) s *JavaScript* enginem *V8*.

Kapitola 6

Použití aplikace

Tato kapitola se zabývá použitím aplikace. Rozebírá možnosti, které program nabízí, popisuje funkce, které nebyly ve výše uvedených kapitolách zmíněny.

6.1 Exportování dat do souboru

Aplikace nabízí možnost uložit data do souboru. Tato akce je vyvolána při kliknutí na tlačítko v levém sloupci *save nodes*. Soubor je následně uživateli nabídnut ke stažení.

Formát souboru byl navrhnut s ohledem na maximální jednoduchost a čitelnost vytvořeného souboru. V prvním kroku jsou nahrány koordináty bodů. Ty jsou vypisovány do proměnné reprezentující řetězec souboru, který bude vytvořen. Formát zapisování je ve tvaru:

---

hodnota souřadnice na ose x {mezera} hodnota souřadnice na ose y {konec řádku}

Algoritmus 4: Formát zapisování souřadnic bodů do souboru

Takto vytvořený řetězec je poté nahrán do soubru a nabídnut ke stažení. Soubor samotný se tedy v podstatě skládá ze dvou sloupců, přičemž první sloupec reprezentuje souřadnice na ose x, a druhý souřadnice na ose y. Každý řádek pak reprezentuje jeden bod v prostoru.

Takto vytvořený soubor je jednoduchý na pochopení i úpravu, může být jednoduše generován a upravován.

6.2 Nahrávání souborů s daty do aplikace

Dále aplikace nabízí nahrát soubor s daty a vložit data do plochy pomocí takového vstupu. To může být vyvoláno kliknutím na tlačítko *load file*, nacházejícím se v levém sloupci.

Aplikace je kdykoli připravena takový soubor zpracovat. Nahrátá data se pak spojí s daty, které se již v aplikaci nachází. Při duplicitě bodů se však na klikací ploše bude nacházet pouze jeden z těchto bodů. Formát souboru, který je nahráván, je přešně popsán v podkapitole „Exportování dat do souboru“. Pokud soubor tyto požadavky nesplňuje, není aplikací přijmut a v dolní informační části je zobrazen chybový výpis „Wrong input file format“. I když však soubor požadavky splňuje, je vyžadováno, aby byla data v přesném rozmezí velikosti plochy, do které bude vkládána. To v praxi znamená, že minimální index na ose x může být 0,

maximální 570. Na ose y jsou tyto hodnoty stanoveny na 0 a 400. Pokud jsou tato pravidla porušena nebo pokud se v souboru nachází jakýkoli nevalidní znak, je opět zobrazen chybový výpis „Wrong input file format“.

Při správném formátu souboru aplikace vypisuje hlášení „Correct input file format“ a interpretuje data ze souboru na levou klikací plochu. Dále je možné s daty pracovat, jako s obyčejnými body.

6.3 Barevné zvýraznění při dělení zón

Aplikace je určena k výuce algoritmů, které tvoří stromy. Jednou z vlastností je vyváženost. Touto vlastnotí se však zabývají jiné kapitoly. Tato kapitola se zabývá schopností aplikace tuto vlastnost zobrazit. K tomu bylo využito barevného zvýraznění pole, ve kterém probíhá rozdělování.

Při vykonání jednoho kroku jakéhokoli algoritmu jsou děleny oblasti. Pro zvýraznění této akce je pak celá oblast, která byla dělena, obarvena. Při každém rozdělení se vychází z barvy předchozí a následující barva je o něco více modrá. Počáteční zóna je bílá a s pokračujícím rozdělováním zóny více a více tmavnou a získávají až sytě modrou barvu. Tím dochází ke zvýraznění oblastí, které jsou rozdělovány nejvíce a oblasti, které byly děleny málo, jsou světlé. Již zběžným pohledem na zóny můžeme určit místa, která strom vyvádějí z rovnováhy – jsou tmavší, než okolní. Také můžeme jednoduše zjistit, že je strom vyvážený – všechny zóny mají stejný odstín modré.

Kapitola 7

Příklady

Kapitola Příklady je částí, která dokumentuje použití aplikace na konkrétních příkladech. Vstupní soubory jsou uloženy na přiloženém CD, kdokoli, kdo přijde s touto prací do styku, je schopen si sám příklady interpretovat. Výběr příkladů probíhal tak, aby byly vybrány typické situace a také situace demonstrující vlastnosti algoritmů. Většinou jsou vytvořené stromy vyvážené, obrázky vytvořených stromů z pravé stromové oblasti aplikace jsou proto přiloženy jen v případech, kdy je vyváženost porušena.

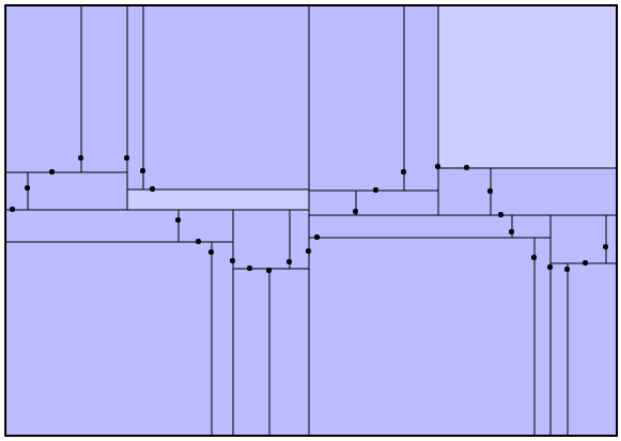
7.1 Syntetické příklady

Následující podkapitola popisuje příklady, které nejsou podloženy realnými daty. Jedná se však o zajímavé případy, stojící za zmínku.

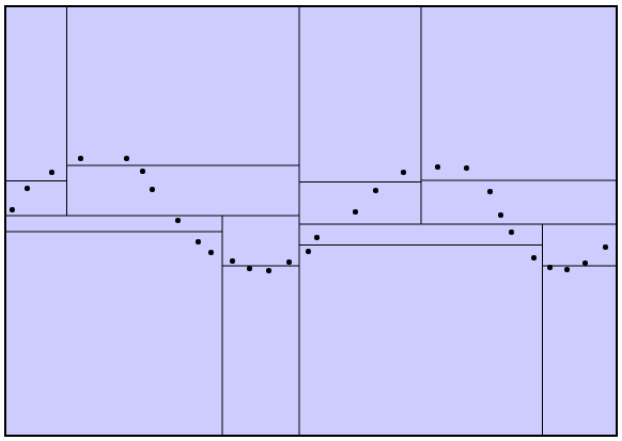
7.1.1 Sinusoida

Tento příklad vychází ze souboru „sinusoida“.

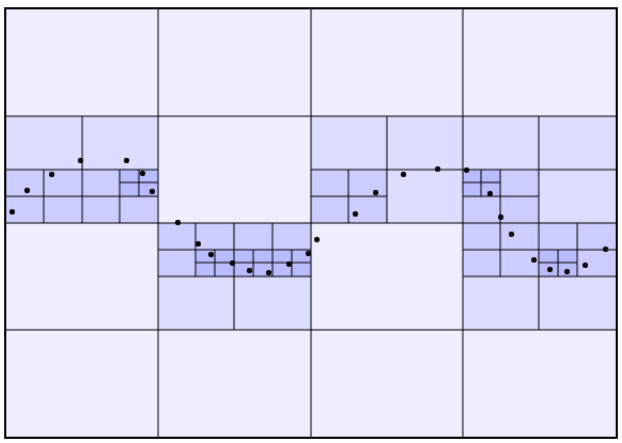
Data jsou vložena do plochy v přibližném tvaru sinusiody, bodů je 29. Data pokrývají téměř celý rozsah osy X, na ose Y jsou soutředěny obzvláště do středu. O tom vypovídá i vzniklé dělení pomocí algoritmů k-D Strom a jeho adaptivní verze. Dělení na ose X je rovnoměrně rozdistribuováno do celého rozsahu, zatímco na ose Y se soutředí do středu (obrázky 7.1 a 7.2). U algoritmu quadtree je plocha nejříve rozdělena rovnoměrně a poté se dělení soustředí do oblastí, které jsou blíže k bodům (obrázek 7.3).



Obrázek 7.1: Data ve tvaru sinusoidy zpracovaná algoritmem k-D Strom



Obrázek 7.2: Data ve tvaru sinusoidy zpracovaná adaptivním algoritmem k-D Strom

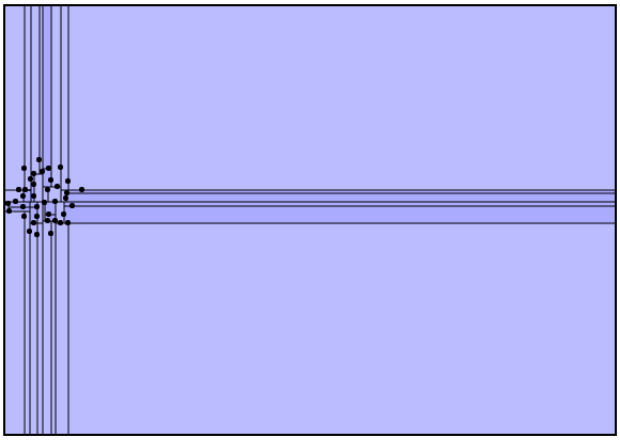


Obrázek 7.3: Data ve tvaru sinusoidy zpracovaná algoritmem quadtree

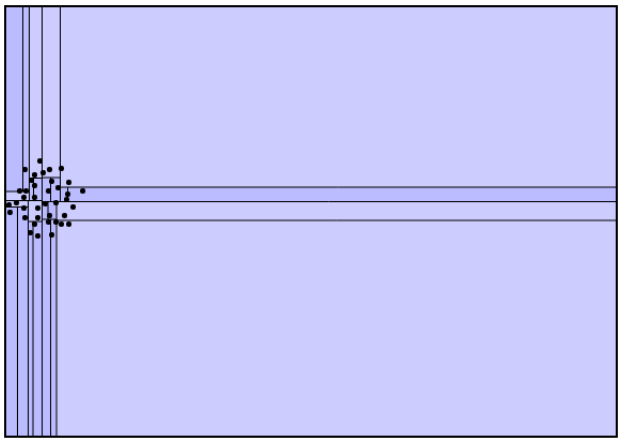
7.1.2 Shluk bodů v jednom místě

Data pro tento příklad můžeme najít v souboru „shluk1“.

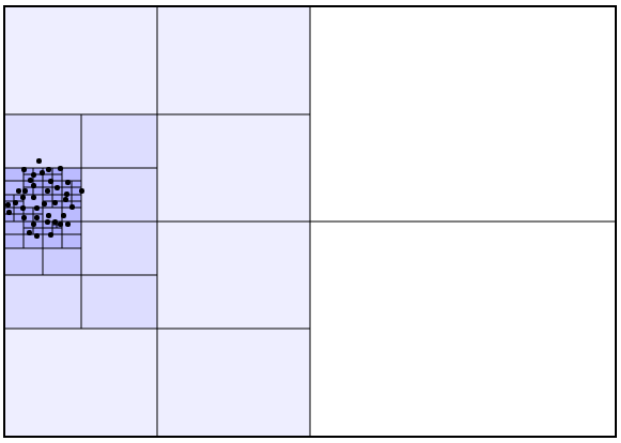
Po nahrátí souboru je vytvořeno 39 bodů, které se soustředí v jedné oblasti a vytvářejí shluk. Jak lze vidět z obrázků 7.4 a 7.5, dělení zón se po vytvoření stromu prvními dvěma algoritmy soustředí do střední oblasti shluků , vnější oblasti jsou dělení více ušetřeny. Poslední algoritmus se pak postupně dělením k shluku přibližuje a nejvzdálenější oblasti nejsou děleny vůbec, protože se zde nenachází žádný bod (obrázek 7.6).



Obrázek 7.4: Data ve shluku zpracovaná algoritmem k-D Strom



Obrázek 7.5: Data ve shluku zpracovaná adaptivním algoritmem k-D Strom

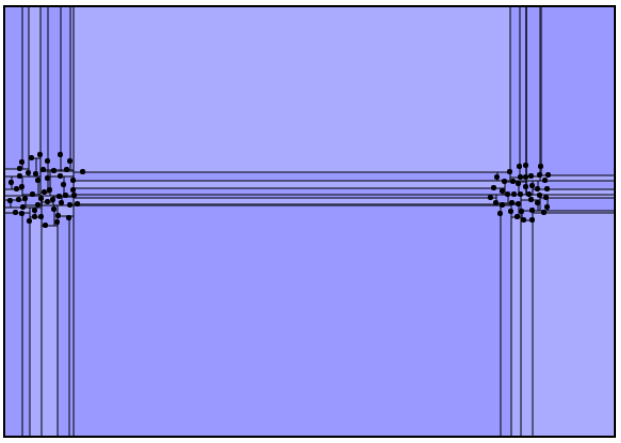


Obrázek 7.6: Data ve shluku zpracovaná algoritmem quadtree

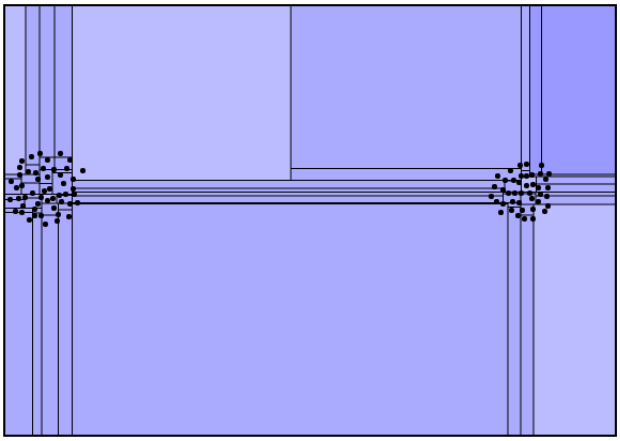
7.1.3 Shluky bodů ve dvou místech

Body z toho příkladu byly exportovány do souboru „shluk2“.

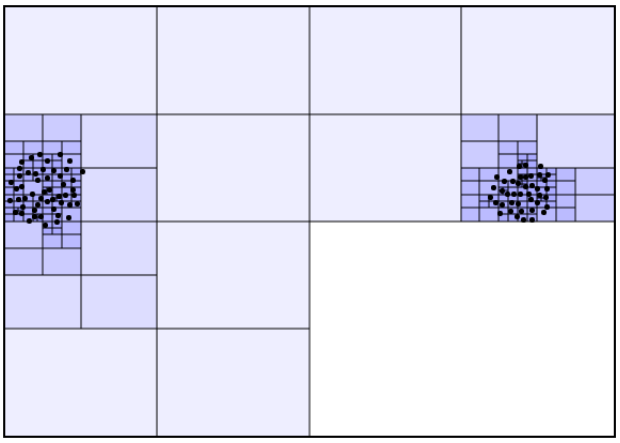
V tomto případě bylo vytvořeno 93 bodů. Výsledek vytvoření stromů je velmi podobný, jako v předchozím případě, dělení se soustředí na indexech shluků bodů, u quadtree v oblastech shluků (obrázky 7.7, 7.8 a 7.9).



Obrázek 7.7: Data ve dvou shlucích zpracovaná algoritmem k-D Strom



Obrázek 7.8: Data ve dvou shlucích zpracovaná adaptivním algoritmem k-D Strom

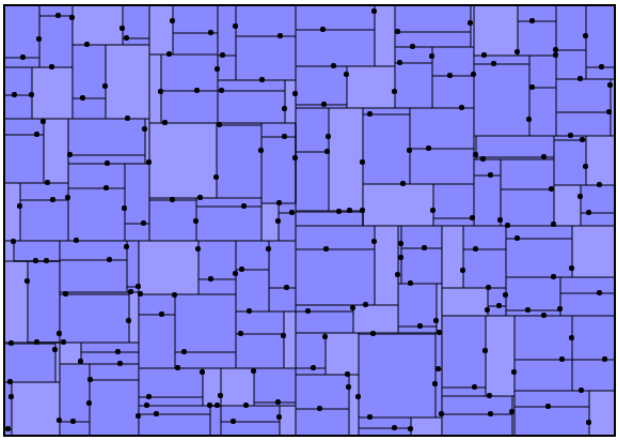


Obrázek 7.9: Data ve dvou shlucích zpracovaná algoritmem quadtree

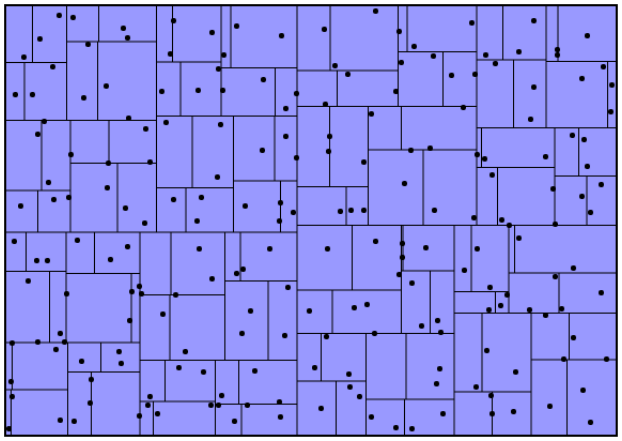
7.1.4 Rovnoměrné rozložení bodů

Pro tento příklad byla použita data ze souboru „rovnomerne“.

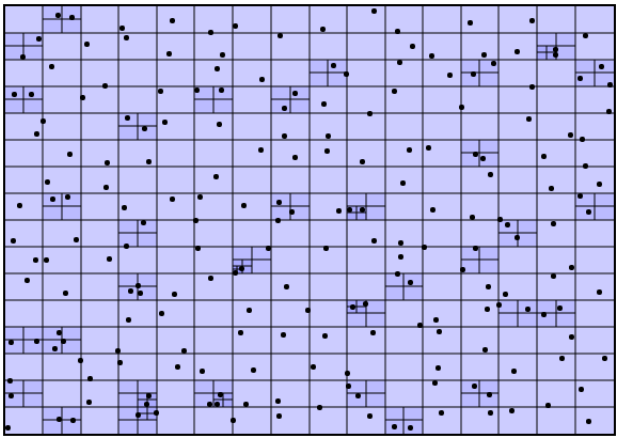
Při tomto příkladu bylo vytvořeno 211 náhodně generovaných bodů, rovnoměrně rozložených v prostoru. Jak můžeme vidět z obrázku po vytvoření stromu, dělení zón je rozprostřeno na celou plochu. Rozdíly mezi velikostmi vzniklých oblastí nejsou tak markantní, jako u předchozích případů. Barva rozdílně obarvených oblastí se liší maximálně o jeden odstín modré (viz. obrázky 7.10 a 7.12). Zajímavý je v tomto případě adaptivní algoritmus, který dokáže dělení přizpůsobit tak, že mají všechny oblasti stejnou barvu. Strom je v takovém případě ideálně vyvážen (viz. obrázek 7.11).



Obrázek 7.10: Data rovnoměrně rozložená v prostoru zpracovaná algoritmem k-D Strom



Obrázek 7.11: Data rovnoměrně rozložená v prostoru – adaptivní algoritmus k-D Strom

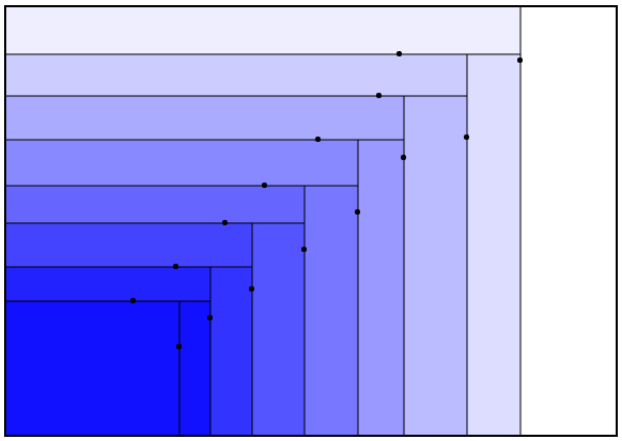


Obrázek 7.12: Data rovnoměrně rozložená v prostoru zpracovaná algoritmem quadtree

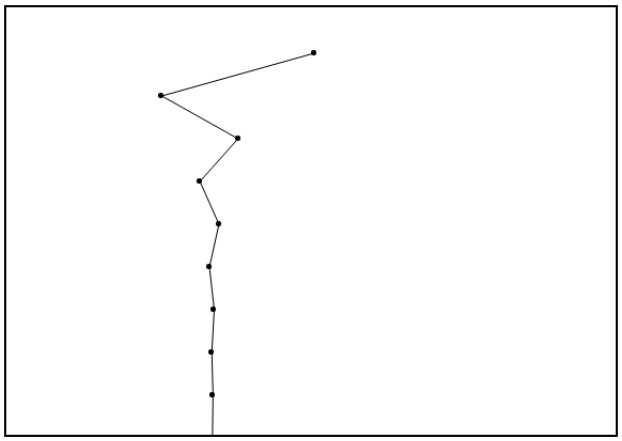
7.1.5 Degradace stromu a jeho přestavba

Následující příklad se od ostatních liší. Pro načtení dat nebylo využito nahrátí souboru, ale levé klikatelné pole. Následující příklad má demonstrovat schopnost aplikace zvýraznit degradaci stromu a také možnost takový strom přeskládat a vyvážit.

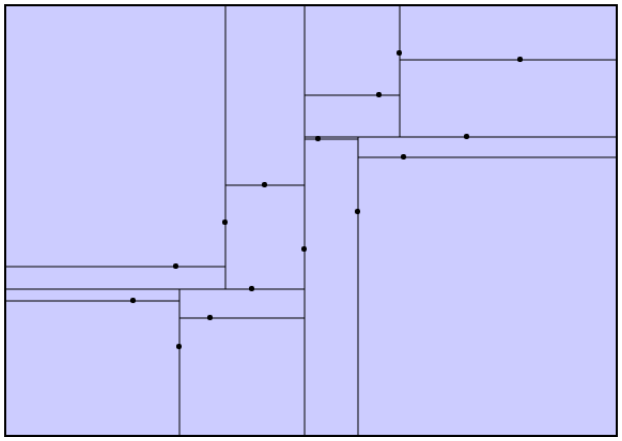
Na obrazku 7.13 vidíme, že byly postupně přidávany body do jedné oblasti. Po přidání každého z bodů byl vykonán jeden krok algoritmu pro tvorbu stromu. Tak vznikal silně degradovaný strom (obrázek 7.14). Všimněme si zvláště sytě modré barvy v oblasti největší degradace. Po přidání patnácti bodů bylo rozhodnuto strom úplně přestavět. Vytvořený strom byl tedy vyčištěn kliknutím na tlačítko *clear tree* a znovu vystavěn pomocí tlačítka *build tree*. Obrázek 7.15 zobrazuje výsledek těchto akcí. Odstín modré je ve všech oblastech stejný. Strom je přestavěn a plně vyvážen (obrázek 7.16).



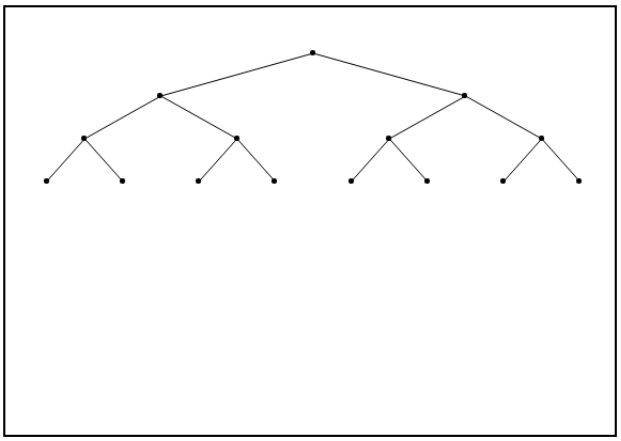
Obrázek 7.13: Silně zdegradovaný k-D Strom



Obrázek 7.14: Silně zdegradovaný k-D Strom – pravé pole pro zobrazení stromu



Obrázek 7.15: k-D Strom po přestavbě jeho zdegradované verze



Obrázek 7.16: k-D Strom po přestavbě jeho zdegradované verze – pravé pole pro zobrazení stromu

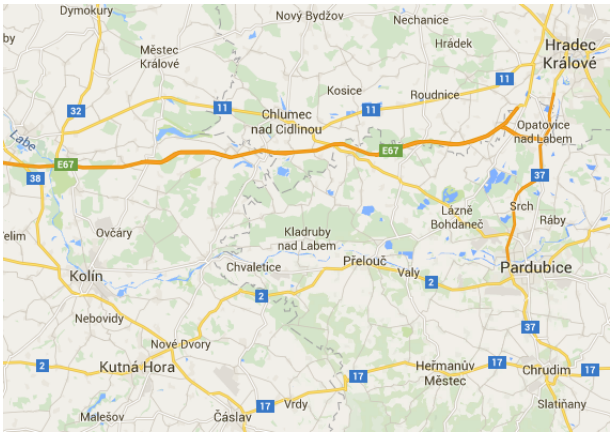
7.2 Příklady na reálných datech

Následující příklady čerpají data z reality a ukazují, jak lze implementované algoritmy využít v takových případech.

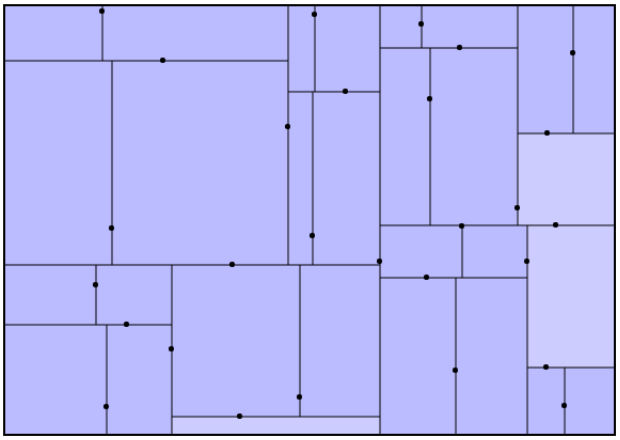
7.2.1 Rovnoměrně osídlená oblast

Data pro tento příklad byla získána z oblasti mezi Hradcem Králové a Kolínem (obrázek 7.17, zdroj https://www.google.cz/maps). Jedná se o rovinatou oblast s rovnoměrným lidským osídlením. Jako vstupní data byla brána jednotlivá lidská sídla vyznačená na mapě. Data byla uložena do souboru „mapacechy“.

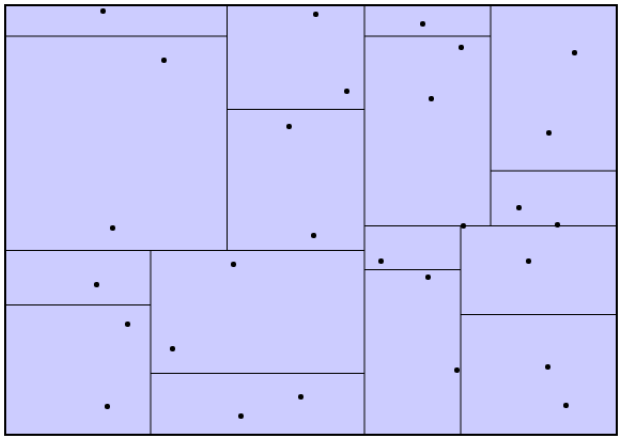
Jako vstup máme 28 bodů pseudo-rovnoměrně rozložených do prostoru. U algoritmů můžeme vidět podobné výsledky, jako u předcházejícího příkladu s rovnoměrným rozložením (obrázky 7.18, 7.19 a 7.20). Však i data jsou si dosti podobná, tento výsledek se již dal očekávat.



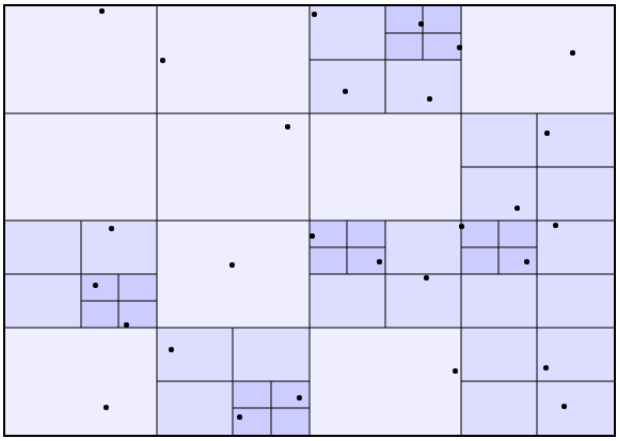
Obrázek 7.17: Mapa oblasti mezi Hradcem Králové a Kolínem



Obrázek 7.18: Mapa převedená do reprezentace aplikace, použit algoritmus k-D Strom



Obrázek 7.19: Mapa převedená do reprezentace aplikace – adaptivní algoritmus k-D Strom

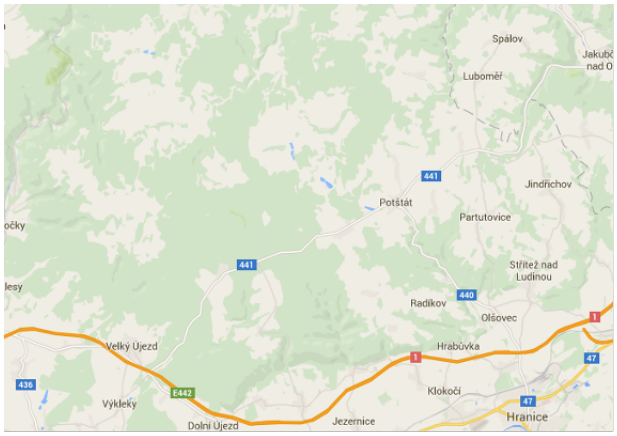


Obrázek 7.20: Mapa převedená do reprezentace aplikace, použit algoritmus quadtree

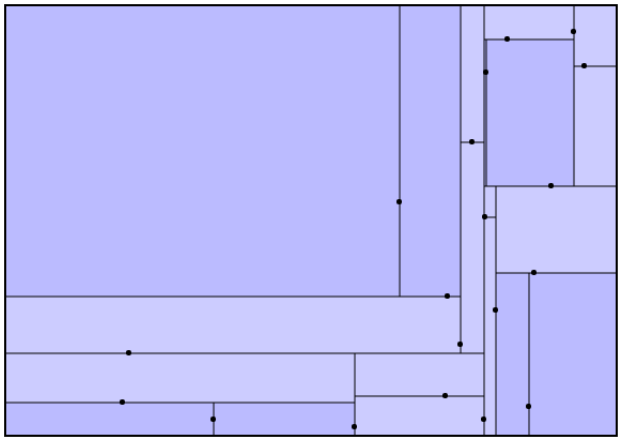
7.2.2 Nerovnoměrně osídlená oblast

Data pro následující příklad pocházejí pochází z oblasti u města Hranice v okrese Přerov. Na mapě (obrázek 7.21) můžeme vidět velkou částečně zalesněnou řídce obydlenou oblast (jedná se o vojenský újezd Libavá). Tato oblast náhle přechází do hustěji obydlené části kolem města Hranice. Mapa z této oblasti posloužila pro získání vstupních dat (uložených v souboru „mapamorava“) pro následující příklad.

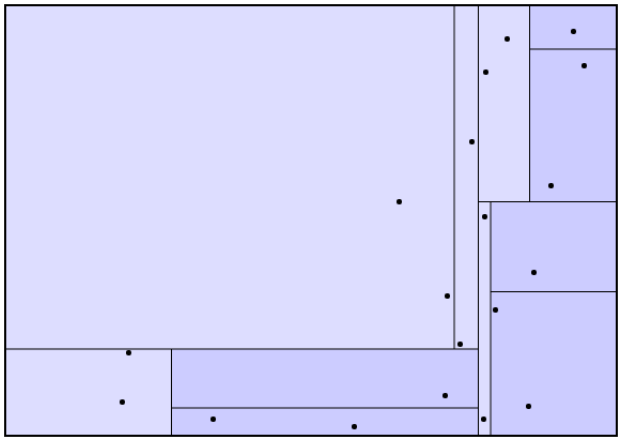
Podobně jako u předcházejících shluků dat vidíme intezivnější dělení na indexech a v oblastech s více body (obrázky 7.22, 7.23 a 7.24). V oblasti vojenského újezdu nedochází k žádnému dělení. Takto vzniklé stromy by se správnými daty mohly bez problému sloužit pro uložení informací o pozici obcí třeba i v rámci celé České republiky.



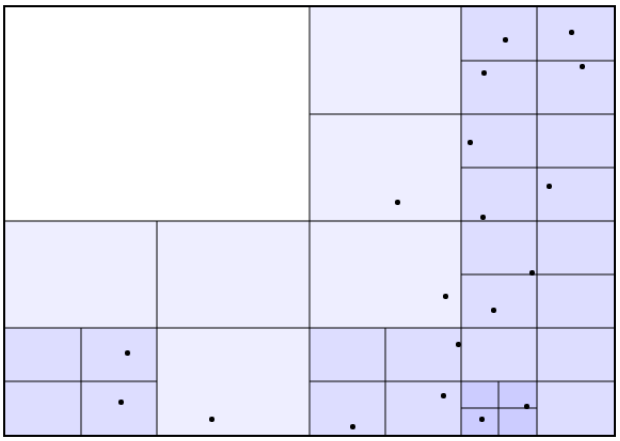
Obrázek 7.21: Mapa oblasti vojenského újezdu Libavá a města Hranic



Obrázek 7.22: Mapa převedená do reprezentace aplikace, použit algoritmus k-D Strom



Obrázek 7.23: Mapa převedená do reprezentace aplikace – adaptivní algoritmus k-D Strom



Obrázek 7.24: Mapa převedená do reprezentace aplikace, použit algoritmus quadtree

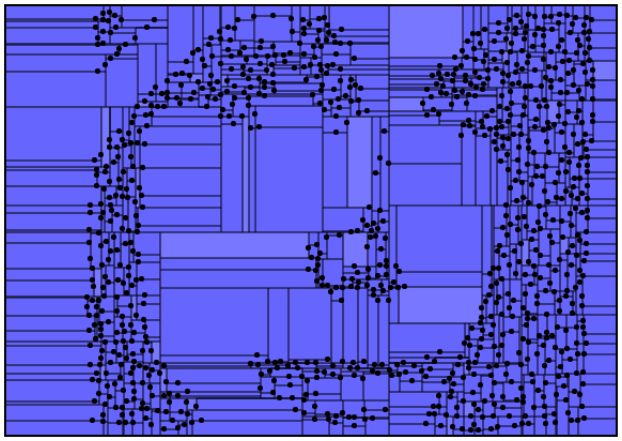
7.2.3 Obličej Lenny

Poslední příklad vychází ze známého obrázku Lenna, který už od roku 1973 slouží jako testovací obrázek [1]. Na obrázku 7.25 můžeme vidět původní obrázek (jedná se pouze o obličejovou část obrázku – vybráno kvůli lehkému rozeznání lidským okem po převedení do aplikace). Další obrázek (7.26) je výsledkem převedení původního obrázku do pouze černé a bílé prahováním [7]. Z tohoto obrázku pak vychází data, která byla uložena do souboru „lenna“.

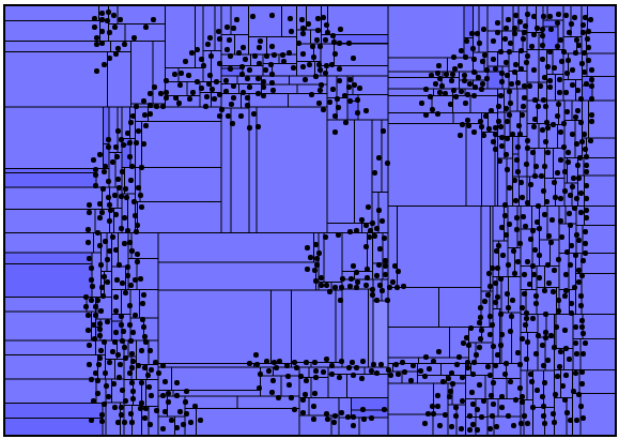
Body se soustředí do černých oblastí, zatímco bílé oblasti vůbec žádné body neobsahují. Obdobně jako v případech se shluky dat nebo nerovnoměrným osídlením dochází k velkému rozdělování oblastí v tmavých částech obrázku (7.27 a 7.28). Zajímavý je algoritmus quadtree, který díky vlastnostem algoritmu barevně zvýrazňuje oblasti s velkým počtem bodů a tím obrázku dodává na optické rozpoznatelnosti (obrázek 7.29).

Obrázek 7.25: Výchozí obrázek v odstínech šedé

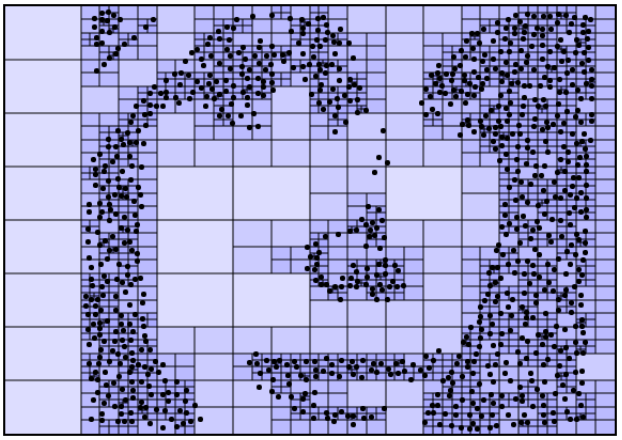
Obrázek 7.26: Obrázek transformován pomocí prahování na černou a bílou



Obrázek 7.27: Obrázek převedený do reprezentace aplikace, použit algoritmus k-D Strom



Obrázek 7.28: Obrázek převedený do reprezentace aplikace – adaptivní algoritmus k-D Strom



Obrázek 7.29: Obrázek převedený do reprezentace aplikace, použit algoritmus quadtree

Kapitola 8

Závěr

Cílem této bakalářské práce bylo vytvořit aplikaci podporující výuku a pochopení problematiky indexačních algoritmů k-D Strom, jeho adaptivní verze a quadtree.

Daný cíl byl splněn, v aplikaci lze demonstrovat klíčové vlastnosti algoritmů, aplikace je interaktivní a funguje bez problému na všech majoritních prohlížečích.

Aplikace reaguje dostatečně rychle na vnější požadavky. I uživatel, který není s problematikou plně seznámen je schopen ji ovládat a pochopit. Díky osmi předpřipraveným datovým sadám si může interpretovat speciální případy. Popis jednotlivých souborů a výstupů jejich zpracování umožňuje lepší porozumění příkladům. Několik ukázek nad reálnými daty pomáhá pochopit využití indexačních algoritmů v praxi. Obrázky doplňující výklad umožňují udělat si představu o aplikaci bez nutnosti spuštění samotného programu.

Práce mi rozšířila rozhled jak v řešené problematice, tak i v oblasti webových technologií. Dala mi zkušenost s vývojem ucelené aplikace s účelem, nad kterým se bylo třeba zamyslet. Implementace musela probíhat s ohledem na daný účel.

V rozvoji programu by se dalo pokračovat a to zejména implementací dalších indexačních algoritmů za využití již připravených objektů a metod. Dalším zajímavým prvkem by byla možnost udělat v algoritmu krok zpět. Dalo by se také rozšířit zpracování souborů, aplikace by mohla přijímat data ve více než dvou dimenzích.

Literatura

[1] BBC News Online: Playboy centrefold photo shrunk to width of human hair. [Online; navštíveno 12.5.2016].
URL http://www.bbc.co.uk/news/technology-19260550

[2] Bertino, E.; OOI, B. C.; Sacks-Davis, R.; aj.: Indexing Techniques for Advanced Database Systems . Kluwer Academic Publishers, 1997, ISBN 0-7923-9985-4.

[3] Dušan Janovský: Div a span: neutrální HTML tagy. [Online; navštíveno 17.5.2016].
URL http://www.jakpsatweb.cz/div-span.html

[4] Google: Chrome Browser. [Online; navštíveno 12.5.2016].
URL https://www.google.co.uk/intl/en/chrome/browser/desktop/

[5] Meagher, D.: High-speed image generation of complex solid objects using octree encoding. Září 27 1995, eP Patent 0,152,741.
URL http://www.google.com/patents/EP0152741B1?cl=en

[6] Selenium: Selenium - Web Browser Automation. [Online; navštíveno 11.5.2016].
URL http://www.seleniumhq.org/

[7] Španěl, M.; Beran, V.: Obrazové segmentační techniky [online]. FIT VUT v Brně, 2005-10-12 [cit. 2016-05-12], [Online; navštíveno 12.5.2016].
URL http://www.fit.vutbr.cz/~spanel/segmentace/

Přílohy

Seznam příloh

A Obsah CD 40

Příloha A

Obsah CD

- Složka „bp_tex_kody“ se zdrojovým tvarem písemné zprávy
- Složka „doc“ s vygenerovanou dokumentací k programu
- Složka „zdrojove_kody“ s:
  - Složka „js“ s knihovnou jQuery a souborem „main.js“ se skriptem aplikace
  - Složka „priklady“ s příklady souborů pro nahrání do aplikace
  - Složka „selenium_tc“ se soubory obsahujícími jednotlivé testovací případy pro Selenium
  - Soubor „index.html“ s hlavní stranou aplikace
  - Soubor „css.css“ se styly
- Soubor „projekt.pdf“ s písemnou zprávou