Univerzita Pardubice
Fakulta elektrotechniky a informatiky

# Návrh a implementace softwarového nástroje pro generování železniční sítě s využitím Oracle Spatial a MapViewer

Bc. Adam Ostrožlík

Diplomová práce

2018

**Univerzita Pardubice**
**Fakulta elektrotechniky a informatiky**
Akademický rok: **2017/2018**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Adam Ostrožlík** |
| Osobní číslo: | **I16235** |
| Studijní program: | **N2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **Návrh a implementace softwarového nástroje pro generování železniční sítě s využitím Oracle Spatial a MapViewer** |
| Zadávající katedra: | **Katedra softwarových technologií** |

Zásady pro vypracování:

V teoretické části práce budou popsány možnosti uchovávání multidimenzionálních dat se zaměřením na sítové topologie.
Praktická část bude zaměřena na vypracování softwarového nástroje pro generování železniční sítě. To zahrnuje návrh a implementaci příslušných algoritmů, které z bázových dat vybudují model infrastruktury železniční sítě. Data modelu budou uloženy v databáze Oracle Spatial s využitím technologie Network Data Model. Pro vizualizaci a ověření vygenerovaného modelu železniční infrastruktury bude využit nástroj MapViewer

Rozsah grafických prací:

Rozsah pracovní zprávy: **60 stran**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

**BRYLA, B.,LONEY,K. Mistrovství v Oracle Databáze 11g. Boston, Computer Press Books, 2009.**
**LEWIS, H. R., DENENBERG, L. Data structures and their algorithms. Berkley, Adison-Wesley, 1997.**

Vedoucí diplomové práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **30. října 2017**
Termín odevzdání diplomové práce: **18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

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

V Pardubicích dne 15. listopadu 2017

Prohlášení autora

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

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 18. 05. 2018

Bc. Adam Ostrožlík

## PODĚKOVÁNÍ

V úvodu bych rád poděkoval vedoucímu práce panu Ing. Janu Fikejzovi, Ph.D. za odborné vedení v celém průběhu vykonávání tohoto díla.

## ANOTACE

Tato diplomová práce se zabývá multidimenzionálními datovými strukturami a databázovými systémy, které ukládání těchto typů struktur podporují. Hlavním cílem praktické části je z dostupných souborů vygenerovat model železniční sítě, jejíž podoba je následně uložena do databáze jako prostorová data. Data uložená v databázi se následně zobrazují pomocí dostupných technologií MapBuilder a MapViewer do aplikace s grafickým uživatelským rozhraním JavaFX.

## KLÍČOVÁ SLOVA

Oracle; MapBuilder; MapViewer; Železnice; Datové struktury; Spatial; Spatial databases; Prostorová data

## TITLE

Design and implementation of a software tool to generate of rail network model using Oracle Spatial and MapViewer

## ANNOTATION

This diploma thesis deals with multidimensional data structures and database systems which support storing of these types of structures. The main goal is to generate a model of the railway network from the available files, the form of which is subsequently stored in the database as spatial data. Data stored in the database is then displayed using the available MapBuilder and MapViewer technologies into a JavaFX graphical user interface application.

## KEYWORDS

Oracle; MapBuilder; MapViewer; Railway; Data structures; Spatial; Spatial databases; Multidimensional data

# OBSAH

# SEZNAM ZKRATEK A ZNAČEK

| | |
|---|---|
| API | Application Programming Interface |
| BLOB | Binary Large Object |
| CAD | Computer Aided Design |
| DBF | DataBase File |
| DBMS | Database Management System |
| DDL | Data Definition Layer |
| DML | Data Manipulation Layer |
| GPS | Global Positioning System |
| IT | Information Technology |
| MBR | Minimum Bounding Rectangle |
| MBV | Minimum Bounding Volume |
| OLAP | Online Analytical Processing |
| ONDM | Oracle Network Data Model |
| SQL | Structured Query Language |
| SRID | Spatial Reference Identifier |
| TDNU | Traťový definiční nadúsek |
| TUDU | Traťový definiční úsek |
| WKB | Well-Known Binary |

# SEZNAM OBRÁZKŮ A ILUSTRACÍ

# SEZNAM TABULEK

# ÚVOD

Systém ohodnocování úseků železničních kolejí neboli staničení je nepřetržitě se vyvíjejícím odvětvím železniční sítě, které se doposud stále rozvíjí a upravuje. Neustálými změnami vznikají různé nesrovnalosti, jež jsou především způsobeny skutečností, že některé koleje byly budovány později, což zapříčinilo vznik celé řady datových anomálií.

Podle Niklause Wirtha lze program chápat jako celek obsahující datové struktury a algoritmy. Primárním cílem této práce je vybudování vícevrstvého modelu železniční sítě nad vhodnými datovými strukturami a návrh a implementace algoritmů, které se snaží minimalizovat dopad nekonzistentních dat a různých výjimek na konečnou podobu tohoto modelu.

Cílem teoretické části je představit vybrané datové struktury pro práci s multidimenzionálními daty se zaměřením na síťové topologie a s nimi rovněž související databázové systémy, které podporují ukládání prostorových dat ať již implicitně, anebo pomocí rozšiřujících modulů. Posledním cílem teoretické části je popsat aplikaci, jež je předmětem praktického segmentu práce.

Praktická část se zaměřuje na návrh a implementaci softwarového nástroje s grafickým uživatelským rozhraním pomocí programovacího jazyka Java s využitím technologie JavaFX. Očekávanou přidanou hodnotou uvažované aplikace je zautomatizování procesu vybudování modelu železniční sítě z bázových dat. Vybudovaný model železniční infrastruktury je následně možné exportovat do vhodné databáze a pomocí dostupných technologií ji zobrazit uživateli.

## Konvence

V této práci se používají následující typografické konvence:

„**`Tučný styl písma Courier New o velikosti 12`**“

Slouží k vyznačení názvů datových typů, tabulek a hlaviček metod ve zdrojovém kódu.

„`Styl písma Courier New o velikosti 12`“
Slouží k vyznačení názvu proměnných, sloupců a těl metod zdrojového kódu.

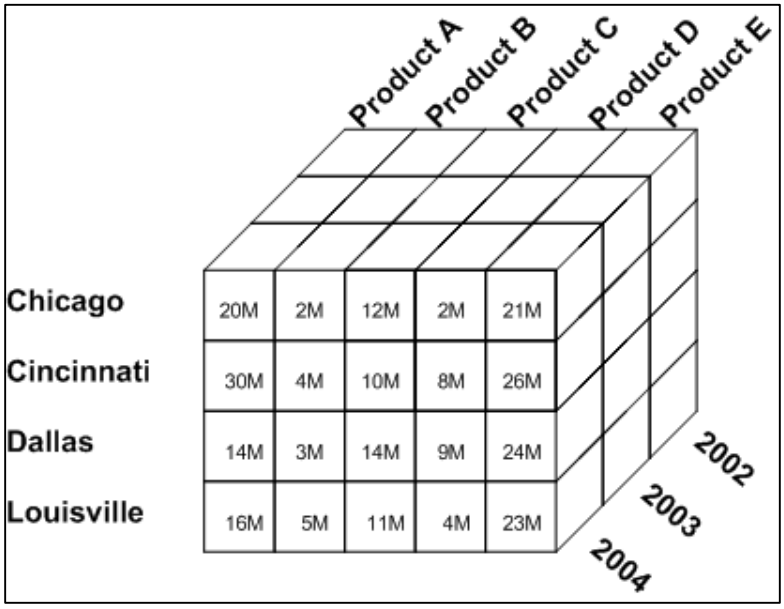
„**Tučný styl písma Times New Roman o velikosti 12**“
Slouží k vyznačení položek seznamu, nespadají-li položky do konvencí uvedených výše.

„Modrý podtržený text písma Times New Roman o velikosti 12“
Slouží ke zvýraznění URL odkazu.

# 1 MULTIDIMEZIONÁLNÍ DATA A STRUKTURY

Úvodem diplomové práce bude čtenář nejdříve seznámen s problematikou multidimenzionálních dat. S touto datovou reprezentací se pravděpodobně setkal každý, aniž by o tom ve skutečnosti věděl. Příkladem mohou být dnes tak oblíbené navigační systémy, kde se pomocí správných algoritmů a datové reprezentace optimalizují náhledy do mapových podkladů. Tato forma uložení dat se samozřejmě týká mnoha dalších oblastí. Jedním odvětvím, které mimochodem také souvisí s navigačními systémy, jsou prostorové databáze, jež umožňují ukládat vícerozměrná data (např. úsečka, bod a polygon). V oboru databází se čtenář může setkat i s pojmem multimediální databáze, které umožňují uchovávat údaje (např. text, obrázky a grafické objekty) a mimoto zachycovat i vztahy mezi nimi. Dalšími příklady jsou též oblasti grafických systémů, herního programování, CAD[1] systémů, robotiky, zpracování obrazu a mnoho dalších, kde jedna dimenze k uložení údajů nestačí, anebo by byla velmi neefektivní.

Aby byla také uvedena nějaká forma definice, tak lze říci, že multidimenzionální datové struktury se zabývají reprezentací množiny bodů. Tyto body mohou odpovídat pozicím či objektům v prostoru. Chybou je však chápat dimenzi pouze ve smyslu vyjádření pozice v prostoru, neboť za rozměr lze prohlásit libovolný atribut nějakého objektu. Obrázek 1 představuje příklad tří dimenzí, kde každá z nich odpovídá jedné vlastnosti. Tento příklad znázorňuje četnost prodaných produktů v jednotlivých městech za každý uplynulý rok. Této reprezentaci v podobě kostky se odborně říká OLAP[2]:




**Obrázek 1 - Příklad multidimenzionálních dat [1]**

[1] Computer Aided Design – počítačem podporované kreslení
[2] Online analytical processing – rozšíření dvojrozměrného tabulkového uspořádání tak, že každá datová dimenze je uložena v jedné ose kostky

Nejdůležitějším kritériem pro vícerozměrné struktury jsou jednoznačně operace zabývající se vyhledáváním dat. Více dimenzí často znamená nárůst paměťové náročnosti, a tak je klíčové, aby se vybrala pouze ta data, která jsou potřeba dle aktuálně stanovených požadavků. To souvisí i s efektivním ukládáním dat na médium tak, aby datová reprezentace byla co nejvíce úsporná při zachování všech informací. Tento proces lze přirovnat k bezztrátové kompresi dat, která pomocí algoritmů zajistí úspornější datovou reprezentaci takovým způsobem, aby bylo možné data následně zrekonstruovat bez ztráty jakékoli informace. Pro rychlejší přístup k údajům je možné použít vhodnou datovou strukturu, anebo mechanismus pomocných indexových struktur, který s sebou nese však nevýhodu v podobě duplicitního zápisu (index + data) a větších požadavků na paměťové médium. Jak lze spatřit dle informací uvedených výše, pro vývojáře je velmi náročné vytvořit multidimenzionální strukturu, neboť musí zvážit všechny možnosti a rizika, která se při implementaci mohou vyskytnout.

## 1.1 Datové struktury pro multidimenzionální data

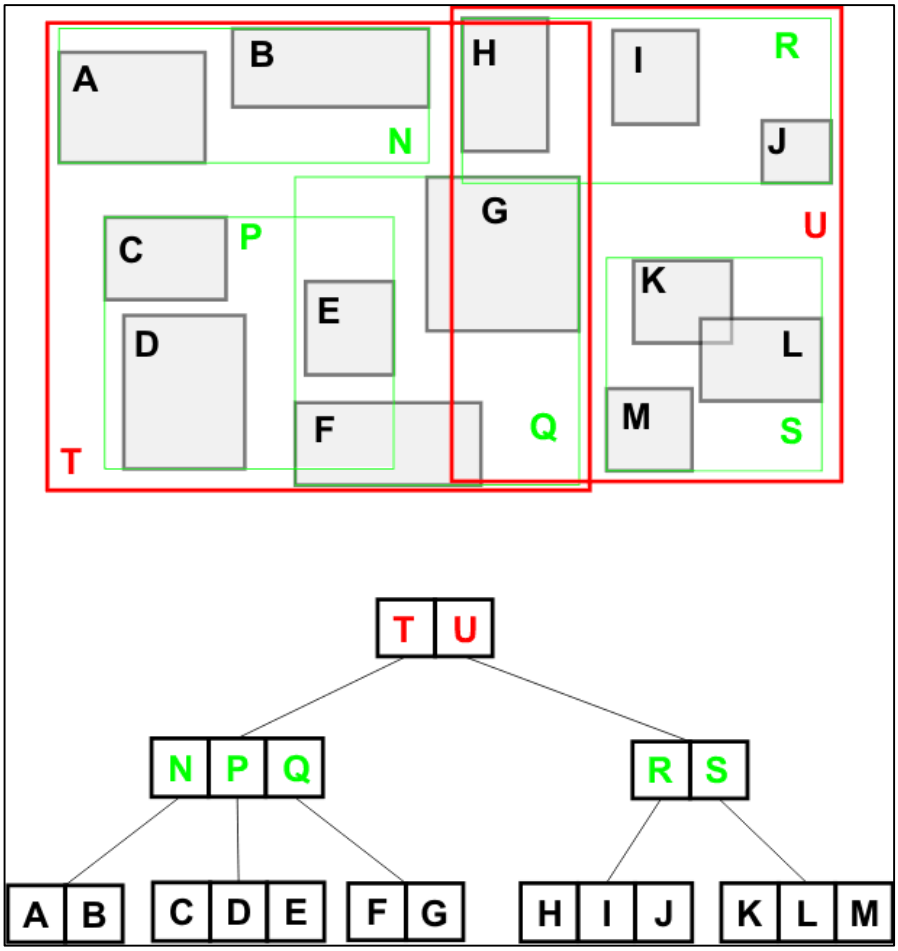
V následujících podkapitolách budou přestaveny jednotlivé datové struktury, které lze využít pro nejenom samotné uchování vícerozměrných dat, ale i pro základní operace jako přidávání, odebírání a samozřejmě hlavně vyhledávání údajů. Důraz bude kladen především na struktury vhodné pro práci se síťovou topologií. Datové struktury budou čtenáři představeny velmi obecně tak, aby pochopil základní problematiku. Podrobněji se lze seznámit se strukturami v publikaci Foundations of multidimensional and metric data structures [2].

### 1.1.1 R-tree

Datová struktura R-strom je jedna ze struktur pro ukládání vícerozměrných dat. Základní charakteristiky jsou velmi podobné struktuře B+ stromu a jsou jimi například vyváženost a minimální počet potomků kořene, který musí být větší či roven dvěma. R-strom využívá techniku dělení zabraného prostoru na oblastí zvané MBR[3]. Tato obdélníková oblast musí být co nejmenší pro zvolené geometrické útvary. Největším obdélníkem bude disponovat vždy kořen, neboť právě kořen je vstupní branou to této datové struktury. Ostatní uzly, jež nejsou listy, uchovávají informace o identifikaci a MBR potomka. Podle těchto vytvořených indexů se lze dostat postupně až k listům, kde jsou uloženy skutečné geometrické útvary. Rovněž lze zde spatřit podobnost se strukturou B+ stromů, protože obě struktury uchovávají data až v listech a ostatní uzly tvoří index pro přístup k listům. Důležitým parametrem při budování je limit potomků pro každý uzel, který nesmí být přesažen. Samotné vybudování struktury je nejjednodušší provádět statickým

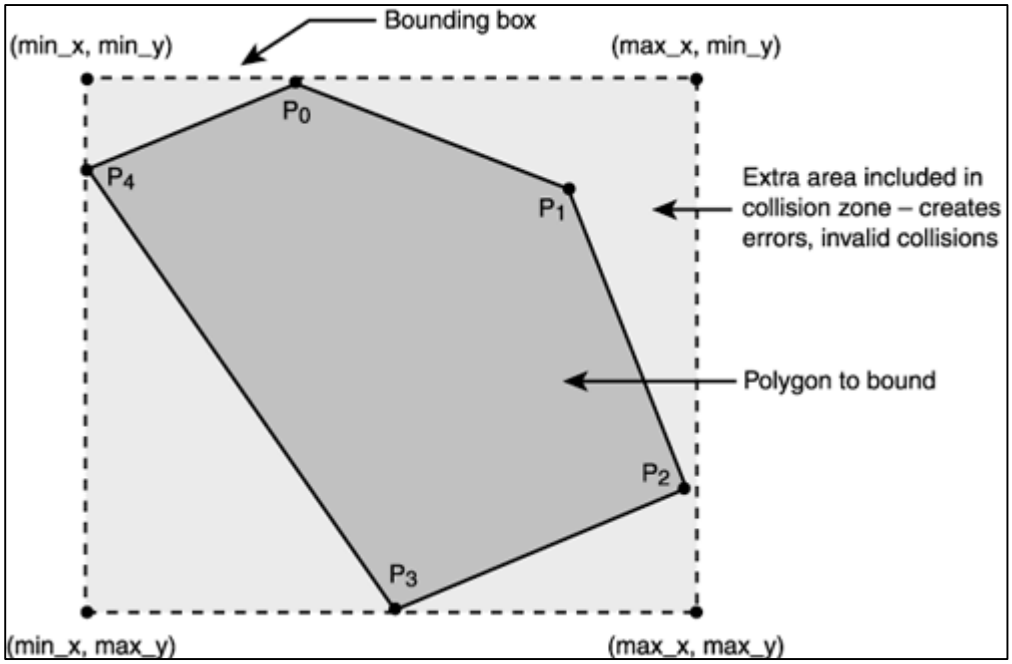
[3] Minimum bounding rectangle – minimální ohraničující obdélník

způsobem, avšak pouze za předpokladu, že data jsou rovněž statická. Po splnění tohoto předpokladu lze strom vybudovat odspodu směrem ke kořeni. Pokud se data mění, pak je nutné strom budovat obtížnějším způsobem. Na obrázku 2 je znázorněn příklad datové struktury R-strom:




**Obrázek 2 - Znázornění datové struktury R-tree [11]**

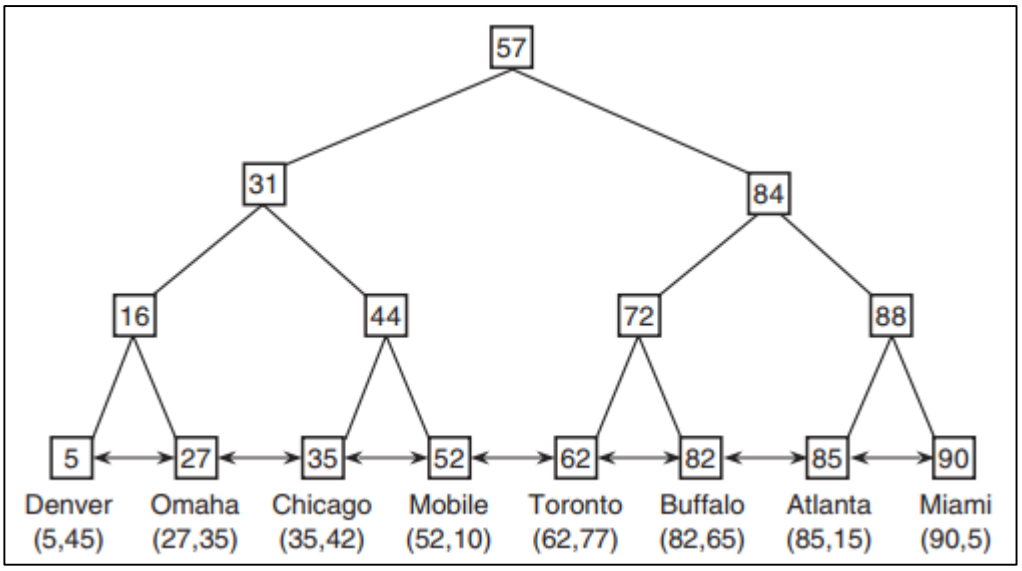
Pro pochopení, jak může takový MBR vypadat, je znázorněn obrázek 3. Obrázek sice vznikl pro účely programování počítačových her, kde se mimochodem řeší i kolize mezi jednotlivými objekty, ale jako příklad naprosto dostačuje. Na obrázku si je možné všimnout tmavší plochy představující cílený geometrický útvar, v tomto případě polygon. Jako minimální ohraničující obdélník se musí zvolit taková obdélníková oblast, která je nejmenší možná a zároveň obsahuje všechny body tohoto polygonu (vyplněna světlejší šedivou barvou). Tuto oblast lze popsat několika způsoby. První informace musí být vždy počátek nejčastěji levého horního rohu a následuje informace buď o výšce a šířce obdélníkové oblasti, anebo souřadnice protějšího bodu úhlopříčky. Správou volbou vyjádření rozměrů MBR se může docílit snadnější forma optimalizace pro speciální případy některých aplikací.




**Obrázek 3 - Znázornění minimálního ohraničující obdélníku MBR [19]**

### 1.1.2 Range tree

Datová struktura s názvem Range tree má velmi podobné rysy se strukturou B+ tree. Obě tyto struktury uchovávají data až v listech stromu, jsou vyvážené (všechny listy jsou ve stejné úrovni neboli hloubce), vnitřní uzly slouží jako index k nalezení listů a jsou binárními stromy (uzly mají maximálně 2 potomky). Pro usnadnění vyhledávání v této datové struktuře jsou listy mezi sebou zřetězeny a seřazeny podle jedné z dimenzí multidimenzionálního klíče. Co se týče vlastností, tak se tato datová struktura řadí mezi ty rychlejší při vyhledávání dat (např. než Quad-tree a K-D tree). Ovšem kde jsou výhody, tam jsou i nevýhody a Range tree se konkrétně potýká s vyššími nároky na paměť. Výsledkem všech těchto charakteristik je datová struktura, která umožňuje uchovávat a nalézt body dle zvoleného rozsahu. Vyhledávání je však možné pouze podle jedné dimenze a to takové, podle které jsou seřazena data v seznamu listů. Pro potřebu vyhledávat v dimenzi další se doplňuje struktura o další stromy. Obrázek 4 znázorňuje 1D Range tree:



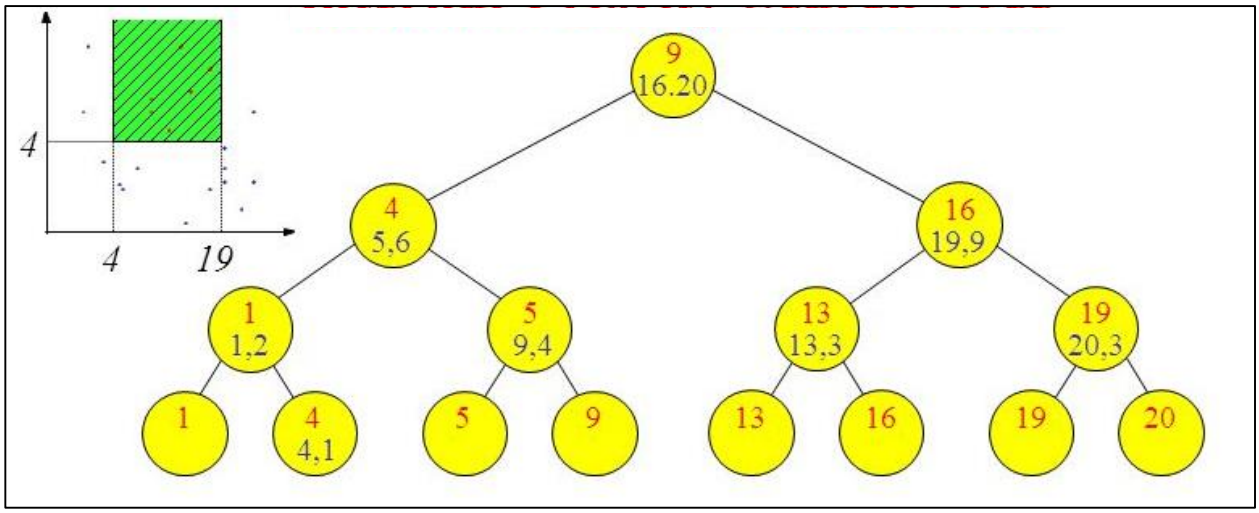

**Obrázek 4 - Příklad jednodimenzionální stuktury Range tree [2]**

Vyhledávání v Range tree zobrazeném na obrázku 4 spočívá ve stanovení požadovaného rozsahu. Je-li zvolen např. rozsah [28:62], pak pomocí indexu je přistoupeno k prvku s označením „Chicago“ – prvek, jehož první dimenze je větší nebo rovna hodnotě 28. Ve zřetězeném seznamu je poté dosaženo posledního prvku, pro který platí, že hodnota první dimenze je menší nebo rovna hodnotě 62 – hledaným posledním prvkem je tedy hodnota „Toronto“. Výsledkem je rozsah prvků mezi těmito dvěma městy.

### 1.1.3 Priority Search Trees

Prioritní vyhledávací strom je datová struktura, jež je rozšířením obyčejné prioritní fronty. Hlavním účelem tohoto rozšíření je nejenom umožnit uchovávat body ve dvou dimenzích, ale i snížit asymptotickou složitost vyhledávání prvků.

Prvky jsou ve stromu seřazeny dle priority a zároveň klíče. Aby tyto dva prvky bylo možné zkombinovat, používá se implementace pomocí hybridní prioritní fronty v kombinaci s binárním vyhledávacím stromem. Na obrázku 5 je znázorněn příklad prioritního vyhledávacího stromu, kde uzel je složen z klíče (nahoře) a priority (dole). Jsou-li tyto hodnoty přirovnány k bodům, tak klíč představuje souřadnici X a priorita souřadnici Y.




**Obrázek 5 - Ukázka prioritního vyhledávacího stromu [4]**

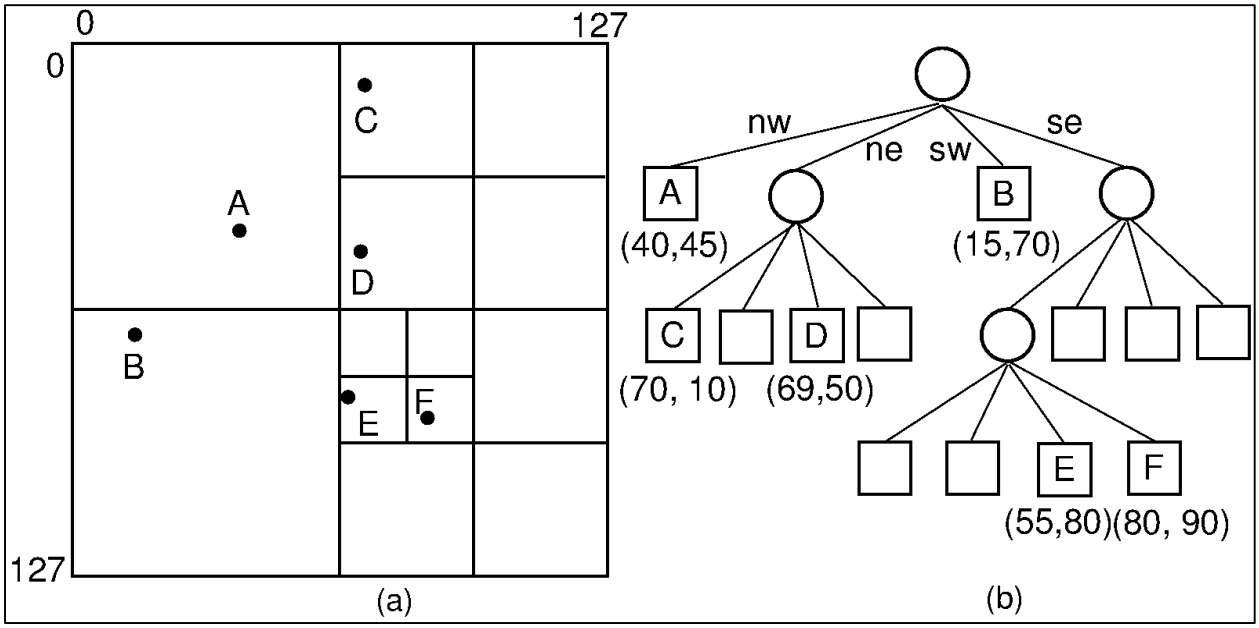
### 1.1.4 Quadtree

Datová struktura Quadtree se vyznačuje tím, že každý uzel, jenž není listem, má přesně čtyři potomky. Rozdělení na právě čtyři potomky se používá z důvodu separace prostoru na čtyři kvadranty nebo regiony. Zatímco vnitřní uzly se starají o rozdělení daného prostoru, tak v samotných listech se pak nachází cílová data. Rovněž je možné zpozorovat podobnost s jinými datovými strukturami, které využívají techniky dělení informací na přístupový index a data. Často se čtenář setká v literaturách s označením kvadrantů podle světových stran – severovýchodní, severozápadní, jihovýchodní a jihozápadní.

Algoritmus spočívá v postupném dělení oblasti na kvadranty. Cílem je dosáhnout takového stavu, kdy v každém kvadrantu bude maximálně jeden prvek (bod). Nastane-li tedy situace, kdy při rozdělení oblasti na kvadranty se v některém z nich vyskytuje více bodů, pak je tento kvadrant dělen znovu a znovu, dokud se nedocílí podmínka přítomnosti maximálně jednoho bodu v kvadrantu.

Zajímavostí je, že samotné kvadranty nemusí přesně odpovídat rozměrově identickým čtvercovým útvarům. Kvadranty mohou být rovněž v podobě obdélníků či jiných náhodných tvarů. Důvodem volby různých podob kvadrantů je čistě optimalizační hledisko. Optimalizace lze dosáhnout mimochodem i povolením kvadrantu obsahovat více než jeden bod. To znamená, že se kvadranty striktně nedělí, když obsahují více než jeden bod, ale tato hranice je nastavena libovolně dle požadavků. Tato technika se používá tehdy, tvoří-li sousedící body nějaký logický celek využitelný pro konkrétní aplikaci.

Obrázek 6 znázorňuje rozdělení prostoru postupně na kvadranty čtvercových útvarů a následnou hierarchickou reprezentaci. Za povšimnutí stojí fakt, že kvadranty s body A a B není již třeba dělit, protože splňují podmínku maximálně jednoho bodu v daném kvadrantu.



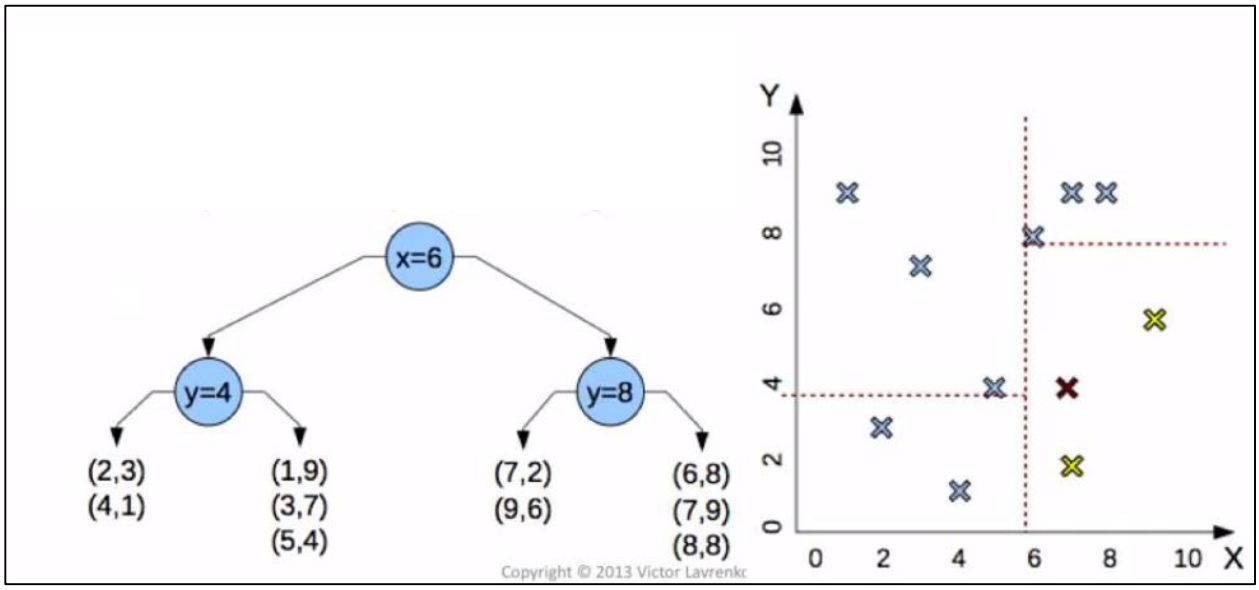

**Obrázek 6 - Ukázka datové struktury Quadtree [6]**

### 1.1.5 K-D tree

Datová struktura K-D strom je velmi podobná struktuře binárního stromu. Stejně jako u předchozích struktur i tato struktura uchovává data pouze v listech. Vnitřní uzly pouze uchovávají informace o rozdělení prostoru. Tento prostor se nerozděluje do kvadrantů, jak tomu bylo u Quadtree, ale používá se mnohem sofistikovanější způsob. V této datové struktuře je prostor dělen způsobem, při kterém se postupně zpracovávají jednotlivé osy tak, že se za bod dělení volí medián souřadnic bodů v podintervalu vzniklém rozdělením předchozího intervalu. Tato definice může vypadat příliš složitě, ale aplikace je velmi jednoduchá, jak je znázorněno na obrázku 7.

Vstupní data na obrázku 7 tvoří souřadnice bodů. Strukturu lze tedy nazvat K-2 strom. Celý postup spočívá v opakování operací v této posloupnosti:

- zvolení náhodné dimenze,
- nalezení mediánu
- a rozdělení dat.




**Obrázek 7 - Zobrazení datové struktury K-D strom (K-2 strom) [7]**

V prvním kroku byla zvolena dimenze X, takže medián nabyl hodnoty 6 (svislá čára). To rozdělilo prostor na dvě poloviny. V dalším kroku je zvolena náhodná dimenze, podle které se bude prostor dále rozdělovat. Jelikož byla předtím použita dimenze X, tak by bylo vhodné, nikoli nutné, použít dimenzi Y. Pro každou polovinu je nalezen odpovídající medián pro osu Y a následně je rozdělen prostor na polovinu vodorovnou čarou. Jak je možné spatřit, tak zde došlo ke třem dělením. Četnost dělení čistě závisí na požadavcích konkrétní aplikace. Obvykle se však rozděluje tak, aby v každém segmentu byl maximálně jeden bod.

V praxi je možné se setkat s různými konvencemi pojmenovávání této datové struktury. Příkladem je parametr K, jenž označuje počet dimenzí. Ostatně je to logické, neboť je-li dosazeno za K například číslo 2, pak lze přímo z názvu přečíst „Dvou dimenzionální strom“. To však není správný přístup, protože podle literatury a konvencí se dodržuje pravidlo, že počet dimenzí označuje právě parametr D, pravděpodobně kvůli shodnému počátečnímu písmenu ve slově „Dimension“.

## 1.2 Multidimenzionální databáze

Relační databázový model používá dvoudimenzionální strukturu, která se skládá z řádků a sloupců, jež jsou součástí tabulky představující skutečnou entitu. Relační databáze se může jevit jako jednoduchá a úsporná organizace dat, nicméně veškeré její výhody se ztrácejí ve chvíli, kdy se nasadí do produkčního prostředí a spouští se uživatelské dotazy. Takovéto dotazy vyžadují spojení několika tabulek, což je velice netriviální záležitostí z důvodu zajištění správných podmínek napojení dat pomocí cizích klíčů. Proto si s tímto úkolem běžný koncový uživatel neví rady a je nucen se obrátit na odborníka z oboru IT, který mu dokáže sestavit efektivní dotazy

v jazyce SQL. Když jsou spuštěny dotazy DML[4] či DDL[5], konkrétně INSERT[6], DELETE[7] a ALTER TABLE[8], tak nesprávné použití těchto příkazů může mít vážné důsledky v produkční databázi, které není leckdy snadné navrátit do původní podoby.

Multidimenzionální databáze přináší úplně odlišný pohled na uložená data. Nejčastěji lze toto uspořádání popsat jako vícerozměrné pole, kde každá buňka v tomto poli má individuální hodnotu přístupnou pomocí několika indexů. Přijatelným příkladem je bezpochyby obrázek 1 na začátku kapitoly 1, kde je znázorněna krychle představující multidimenzionální data, která popisují četnost prodaných výrobků podle konkrétního města a roku. Ke konkrétnímu údaji (buňce v krychli) je možné přistoupit pomocí kombinace indexů rok, město a produkt. Schopnost reprezentovat data na vyšší úrovni pomocí těchto tří atributů je velmi důležitá pro multidimenzionální databáze a charakterizuje jejich velký potenciál. Důležité je však mít na mysli fakt, že počet dimenzí není omezen. Tento atribut je čistě na nás, nicméně je žádoucí zvolit co možná nejmenší počet, protože s narůstajícím počtem rozměrů roste samozřejmě i složitost celého systému.

### 1.2.1 Výhody multidimenzionální databáze

Je-li srovnán relační databázový systém se systémem vícerozměrným, tak má druhý zmíněný značné výhody nad prvním. Jednou z mnoha výhod je seskupení souvisejících informací v jedné dimenzi. Na obrázku 1 jsou například všechny informace o roku seřazené v jedné dimenzi, což umožňuje velmi rychlou operaci sčítání nebo přibližné porovnání s předchozími roky.

Multidimenzionální struktury poskytují vyšší úroveň organizace než relační tabulka. Samotná struktura představuje inteligentnější pohled na data, protože právě tyto pohledy jsou vloženy přímo do struktury jako dimenze na rozdíl od umístění do polí. Pokud by byla zkonstruována fiktivní relační tabulka pro ukázková data, pak by její podoba mohla vypadat následovně:

[4] Data manipulation layer – operace pro práci se řádky tabulky
[5] Data definition layer – operace pro vytvoření a úpravu struktury tabulek
[6] INSERT – operace pro vkládání záznamů do tabulky
[7] DELETE – operace pro mazání záznamů v tabulce
[8] ALTER TABLE – jedna z operací DDL pro úpravu struktury tabulky

**Tabulka 1 - Vzorová data pro obrázek 1**

| Město | Četnost prodeje | Rok | Produkt |
|---|---|---|---|
| Chicago | 10000 | 2004 | B |
| Dallas | 15000 | 2004 | A |
| Dallas | 10000 | 2005 | A |
| Chicago | 20000 | 2005 | B |
| Dallas | 12000 | 2004 | B |

Tato tabulka toho v podstatě mnoho neříká. Je možné se dozvědět pouze to, že jsou zde čtyři atributy a nějaké záznamy. Bohužel již není možné odvodit, jestli spolu údaje nějakým způsobem souvisí. Tento fakt zůstává s relačními databázemi skrytý. To se však netýká obrázku 1, kde lze jednoduše určit vztah mezi jednotlivými dimenzemi. Proto tou nejdůležitější a nejviditelnější výhodou při práci s multidimenzionálními daty je jakýsi přirozený pohled na strukturu údajů podle dimenzí, kde každá dimenze představuje pohled na data z jednoho pomyslného směru. Další důležitou výhodou je rovněž zamezení duplikace údajů. V tabulce si lze všimnout, že dochází k duplikaci dat, vlivem které budou paměťové nároky na médium růst rychleji. V tabulce je pak možné zpozorovat duplikaci údajů o městech a produktech, kde se v každém záznamu mění pouze rok a četnost prodejů. To je velice neefektivní přístup datové reprezentace, který se pak podepíše ve svižnosti systému. Za povšimnutí stojí také skutečnost, že při ukládání tří dimenzí na obrázku 1 je vyžadováno celkem čtyř atributů, aby bylo možné třírozměrný prostor korektně vyjádřit. Relační databáze by tuto skutečnost řešily složitými SQL dotazy, které by vedly k netriviálnímu napojení tabulek.

Uvedená výhoda v inteligenci reprezentace u multidimenzionálních databází není jedinou výhodou. Mezi další lze zařadit vylepšenou datovou prezentaci a navigaci, jednoduchou údržbu a zvýšený výkon.

### 1.2.2 Spatial data

Všechny databázové systémy, které budou níže představeny, podporují uložení prostorových dat. Tuto funkcionalitu mají v sobě již zakomponovanou (dalo by se říci v jádře databázového systému), anebo je dostupná prostřednictvím rozšíření. Účelem prostorových informací je určit nejenom geografickou polohu na Zemi. Mapy se týkají dat pro uložení souřadnic či topologie. Právě s jedním takovým využitím prostorových dat se setká člověk v navigačních systémech či jakékoli formě digitálního zobrazení libovolné mapy. Zde figurují souřadnice

zeměpisné délky a šířky, které určují jednoznačnou polohu na Zemi. To využívá systém GPS [9]pro určování polohy. Více o tomto systému se lze dočíst v referenci [12].

### 1.2.3 Databázový systém MySQL

MySQL databázový systém pro řízení báze dat (DBMS[10]) využívá relační databázový model. MySQL byl vytvořen švédskou firmou MySQL AB. V roce 2008 byl tento systém koupen společností Sun Microsystems, jež je dceřinou společností firmy Oracle Corporation. MySQL se řadí mezi open source databáze, tedy takový druh softwaru, který je k dispozici zdarma včetně zdrojového kódu. To by bylo k základnímu představení MySQL vše. Nyní budou představeny základy o tom, jakým způsobem tento systém podporuje multidimenzionální data.

MySQL zakomponovává podporu prostorových dat v podobě rozšíření s názvem OpenGIS Geometry Model. Toto rozšíření předepisuje každému objektu v prostoru povinné atributy, kterými jsou:

- definice souřadnicového systému (SRID[11]), ve kterém jsou souřadnice objektu definovány
- a objekt, který je instancí nějaké geometrické třídy používané tímto rozšířením.

Zjednodušeně řečeno je vynuceno používat předepsané geometrické třídy, které jsou k dispozici a zároveň uvádět souřadnicový systém. Mezi další zajímavé atributy patří kombinace vlastností zvaných interiér, exteriér a hranice. Exteriérem se rozumí prostor, který je vně objektu, interiér je pak opak, tedy prostor, který objekt vyplňuje. Hranici lze rozumět jako rozhraní mezi exteriérem a interiérem. Další užitečnou vlastností je MBR, který se používá pro vyhledávání jednotlivých geometrických objektů s využitím datové struktury R-strom. Tato datová struktura zastává roli při indexování prostorových dat. Základní geometrická třída s názvem **`Geometry`** představuje abstraktní třídu, jež je kořenem hierarchického uspořádání ostatních tříd popisujících jednotlivá grafická primitiva, která používají i ostatní databázové systémy pro uchovávání vícerozměrných dat.

[9] Global Positioning Systém – systém pro určování polohy na zemi pomocí satelitů
[10] Database Management Systém – systém pro řízení báze dat
[11] Spatial reference identifier – identifikátor souřadnicového systému

Třídy jsou následující:

- **`Point`** – Třída představující bod v prostoru.
- **`Curve`** – Abstraktní třída představující křivku. Třída se dělí dále na jednotlivá primitiva:
  - **`Line`**
  - a **`LinearRing`**.
- **`Surface`** – Abstraktní třída definující plochu, kde základním grafickým primitivem je třída **`Polygon`**.
- **`GeometryCollection`** – Třída obsahující posloupnosti objektů uvedených výše jako například **`MultiPoint`** a **`MultiCurve`**.

Následně budou přestaveny jednoduché způsoby, kterými je programátor schopen jednotlivé geometrické třídy inicializovat. Z důvodu rozsáhlé problematiky zde nebude uveden vyčerpávající seznam všech možností, ale pouze několik, se kterými se vývojář setká nejčastěji. Základním prvkem je samozřejmě inicializace bodu, která se provádí uvedením dvou souřadnic, které se neoddělují čárkou, ale mezerou:

```
POINT(30 25)
```

Pro definici geometrické třídy **`LineString`**, která je předkem třídy **`Line`** a **`LinearRing`**, se používá posloupnost několika bodů za sebou. Jednotlivé body se mezi sebou oddělují čárkou. Pro souřadnice bodů platí stejná pravidla jako výše, tedy oddělování mezerou:

```
LINESTRING(30 25, 0 0, 10 15, 25 12)
```

Posledním příkladem je třída **`GeometryCollection`**, která umožňuje ukládat jednotlivé grafické objekty do kolekce pro zachování vztahu mezi nimi. Na příkladu je znázorněna kolekce dvou bodů a jedné křivky:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30),
LINESTRING(15 15, 20 20))
```

Dalším zajímavým způsobem, kterým lze inicializovat data, je použití techniky zvané WKB[12]. Tato metoda se hojně využívá při práci s binárními proudy dat, kdy se geometrický objekt uloží do binárního formátu databáze typu BLOB[13]. Příklad uložení bodu se souřadnicemi (1, -1) vypadá následovně:

```
0101000000000000000000F03F000000000000F0BF
```

Při pohledu na tuto sekvenci znaků často vzniká neporozumění, jak jsou vlastně data, konkrétně bod (1, -1), vyjádřena. Nejedná se o nic jiného než jen o napojení jednotlivých údajů za sebou. Význam jednotlivých bajtů popisuje tato tabulka:

**Tabulka 2 - Vysvětlení struktury WKB reprezentace**

| Komponenta | Velikost | Data |
|---|---|---|
| Pořadí bajtů | 1 bajt | 01 |
| Typ WKB | 4 bajty | 01000000 |
| Souřadnice X | 8 bajtů | 000000000000F03F |
| Souřadnice Y | 8 bajtů | 000000000000F0BF |

Tabulka dokázala jednoznačně určit smysl jednotlivých bajtů v uvedené posloupnosti. Pro úplnost ještě bude vysvětlen význam komponent týkajících se pořadí bajtů a typu WKB. Pořadí bajtů neboli endianita udává, v jakém pořadí se uloží jednotlivé bajty číselného datového typu. Možnostmi jsou little-endian a big-endian. Typ WKB představuje kód, který jednoznačně identifikuje typ geometrického objektu.

Příklad vytvoření indexu nad sloupcem pro prostorová data:

```
CREATE TABLE geometry (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

[12] Well-Known Binary – binární reprezentace geometrických objektů
[13] Binary large object – databázový datový typ pro ukládání binárních dat

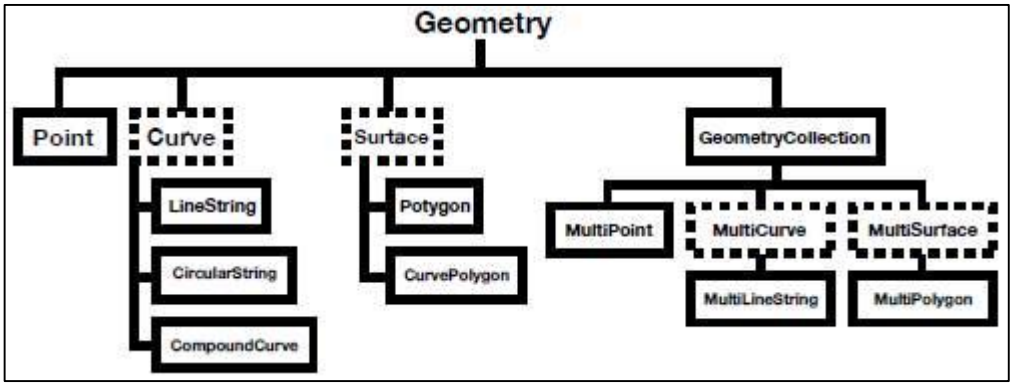
### 1.2.4 Databázový systém MS SQL

Dalším databázovým systémem podporující ukládání vícerozměrných dat je systém od firmy Microsoft, který s podporou této funkcionality přišel až v roce 2008. Tato databáze podporuje následující prostorové datové typy, které v sobě obsahují celkem šestnáct geometrických tříd, z toho jedenáct inicializovatelných:

- **`Geometry`**
- a **`Geography`**.

Datový typ **`Geometry`** uchovává data, která jsou založena na plochém neboli Euklidovském souřadnicovém systému. Tento typ slouží k uložení souřadnic X a Y, které mohou reprezentovat přímky, body a polygony ve dvojrozměrném prostoru. Datový typ **`Geography`** slouží pro uchovávání zeměpisné šířky a délky geografického souřadnicového systému, a tak se využívá v kombinaci s GPS technologiemi.

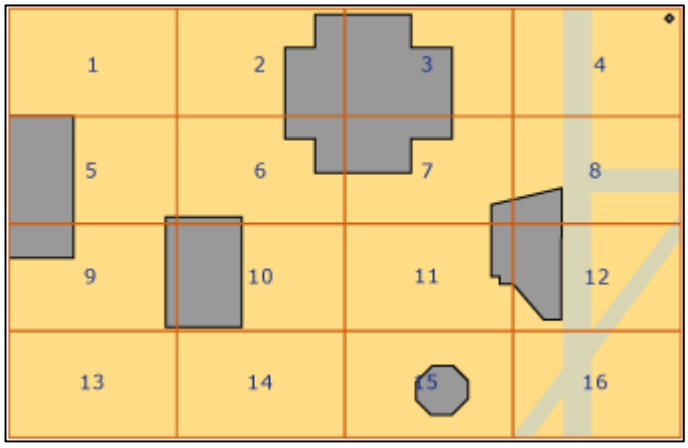
Oba tyto zmíněné datový typy podporují celkem deset datových tříd, které se málo liší od tříd uvedených u databázového systému MySQL v předchozí kapitole 1.2.3. Zbývající jedna třída se nedá použít ve spojení s datovým typem **`Geometry`**. Jedná se o objekt typu **`FullGlobe`**, který lze použít pouze u typu **`Geography`**. Kompletní hierarchie podporovaných tříd typu **`Geometry`** je znázorněna na obrázku 8:




**Obrázek 8 - Hierarchie podporovaných prostorových tříd [20]**

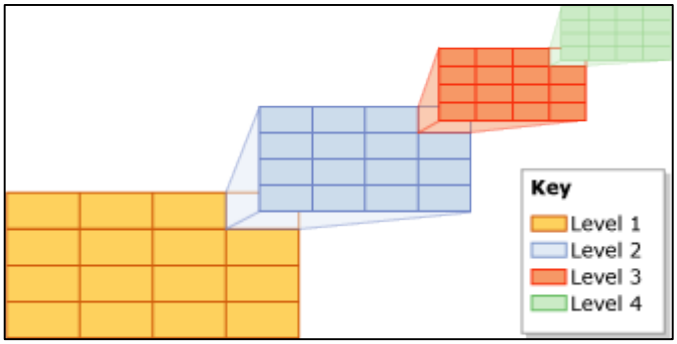
Co se týče indexování prostorových dat, tak to je prováděno pouze na těch sloupcích tabulky, které jsou určeny pro uložení prostorových objektů. Tím se liší od některých systémů, které indexují přímo celou tabulku, což může být značně neefektivní. Prostorové indexy jsou v tomto databázovém systému budovány za použití struktury B-stromu. Před samotným vytvářením indexu provádí systém rutinu zvanou jako rovnoměrná dekompozice prostoru. Tento proces rozdělí prostor do čtyř mřížek neboli úrovní označených jako layer1, layer2, layer3 a

layer4. Tyto úrovně se generují postupně tím způsobem, že se vezme nejvyšší úroveň (layer1) a pro každou sekci (buňku) mřížky z vyšší úrovně se provede operace rozložení na další mřížku o stejných či odlišných rozměrech. Výhodou použití čtyř úrovňového rozdělení prostoru je větší flexibilita, neboť index je založen na více úrovních než na jediné úrovni pokrývající celý prostor. Mřížka může vypadat potom následovně:




**Obrázek 9 - Zobrazení dekompozice prostoru při indexování [13]**

Na obrázku 10 je znázorněna podoba rozdělení prostoru na čtyři úrovně mřížek:




**Obrázek 10 - Zobrazení vrstev při dekompozici prostoru [13]**

Tento proces vytváření čtyř úrovní může být samozřejmě optimalizován vhodným zvolením rozměrů jednotlivých mřížek. Microsoft SQL nabízí celkem tři úrovně, které lze libovolně kombinovat pro jednotlivé mřížky na odlišných úrovních. Čím větší rozměr je zvolen, tím nastane jemnější rozdělení prostoru. Výchozí jemností je MEDIUM na všech úrovních. Podporovanými úrovněmi jsou:

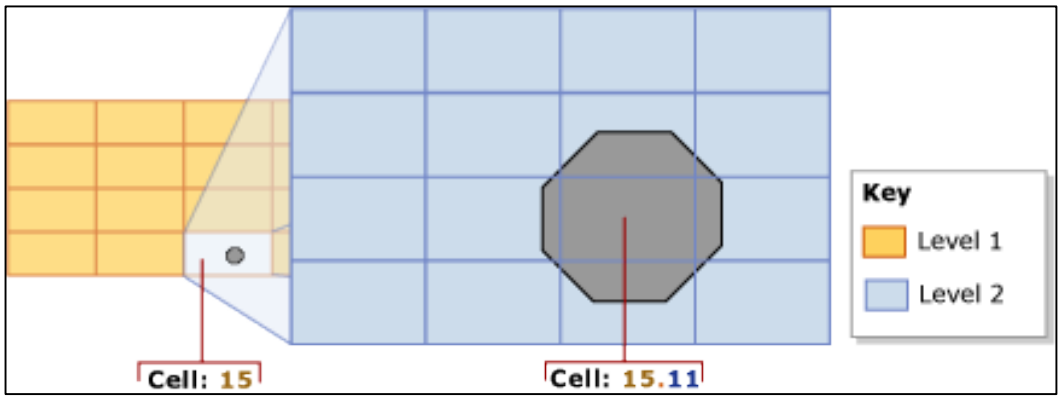
**Tabulka 3 - Tabulka jemností mřížek při dekompozici prostoru**

| Klíčové slovo | Rozměr | Počet buněk |
|---|---|---|
| LOW | 4 x 4 | 16 |
| MEDIUM | 8 x 8 | 64 |
| HIGH | 16 x 16 | 256 |

Po vytvoření čtyř úrovní mřížek následuje proces zvaný teselace. Ten má za úkol vzít postupně každý geometrický objekt a určit, které buňky jednotlivých mřížek spadají do plochy tohoto objektu. Ovlivněné buňky se označují jako dotknuté buňky. Tyto nalezené buňky jsou výstupem procesu teselace a jsou uloženy do indexu. Pro tento proces existují pravidla, která celou úlohu optimalizují nastavením početního limitu dotknutých buněk pomocí tří stanovených vlastností:

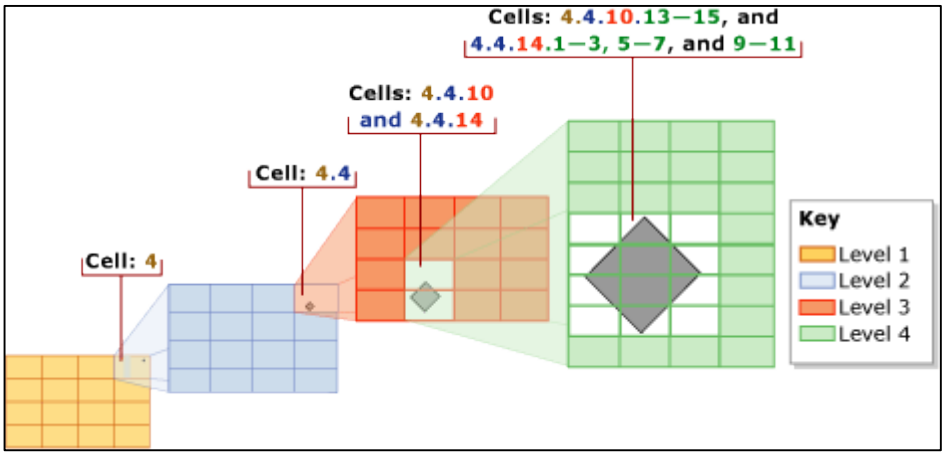
- **The covering rule** – Je-li buňka celá zakrytá objektem nebo jeho částí, pak tato buňka není podrobena teselačnímu procesu v dalších úrovních.
- **The cells-per-object rule** – Na nižších úrovních, než je úroveň 1, se nesmí překročit stanovený limit dotčených buněk.
- **The deepest-cell rule** – Do indexu se zaznamenávají dotčené buňky pouze z nejnižších úrovní. Rodičovské buňky jsou tedy ignorovány.

Znázorněný obrázek 11 představuje první z těchto tří pravidel. Z definice plyne, že buňka, která je celá zahalená, se dále nerozkládá. Tudíž buňka se souřadnicemi (15, 11) se přímo uloží do indexu.




**Obrázek 11 - Pravidlo The covering rule při teselaci [13]**

Další obrázek znázorňuje třetí pravidlo, jehož definice říká, že se vždy použijí dotknuté buňky z nejnižší úrovně.




**Obrázek 12 - Pravidlo the deepest-cell rule při teselaci [13]**

Proces teselace se samozřejmě liší pro oba základní datové typy. Pro prostorový datový typ **`Geometry`** se používá Geometry grid tessellation a pro geografický datový typ **`Geography`** se používá Geography grid tesselation.

Příklad vytvoření indexu v systému MS SQL je následující:

```
CREATE SPATIAL INDEX SIndx SpatialTable geometry col2
   ON SpatialTable(geometry column)
   USING GEOMETRY GRID
   WITH (
    BOUNDING BOX = ( xmin=0, ymin=0, xmax=500, ymax=200 ),
    GRIDS = (LOW, LOW, MEDIUM, HIGH),
    CELLS PER OBJECT = 64,
    PAD_INDEX  = ON);
```

Další informace o tomto databázovém systému a jeho schopnostech se zacházením s multidimenzionálními daty lze najít v referenci [13].

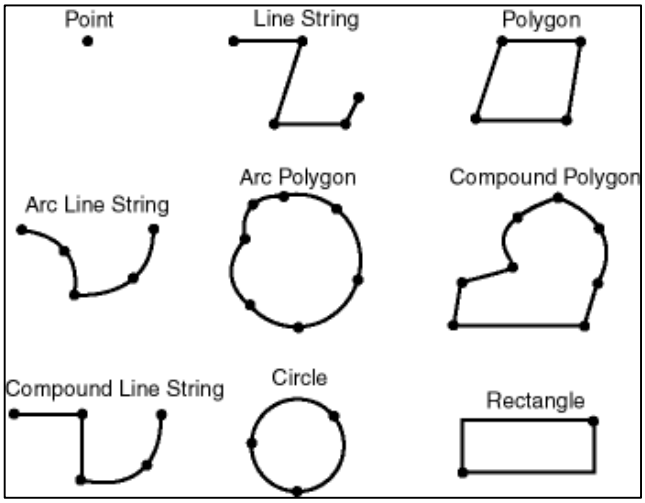
### 1.2.5 Databázový systém Oracle

Databázový gigant Oracle nesmí zcela jistě chybět v tomto seznamu databází podporujících prostorová data. Tomuto systému bude věnováno více odstavců už jen kvůli tomu, že je používán v praktické části této diplomové práce. Při popisu aplikace v teoretické části bude také vysvětleno několik fakt týkajících se prostorových dat v této databázi. Technologiím MapViewer a MapBuilder, jež jsou rovněž předmětem této diplomové práce, bude věnována samostatná kapitola.

Databáze Oracle používá pro práci s prostorovými daty službu zvanou Oracle Spatial and Graph. Tato služba se skládá z několika funkcí, procedur, datových typů a datových modelů, které podporují práci s prostorovými daty. Funkce umožňují prostorová data ukládat, vybrat a analyzovat efektivním způsobem přímo v databázi Oracle. Vše, co se týká prostorových dat, obohacuje databázi konkrétně o tyto prvky:

- schéma MDSYS, které předepisuje syntaxi a sémantiku prostorových datových typů,
- indexovací mechanismus prostorových dat,
- operace, funkce a procedury pro dotazy nad prostorovými daty,
- operace pro optimalizaci,
- síťový i topologický datový model
- a podpora použití GeoRasteru.

Spatial and Graph podporuje objektově-relační model pro reprezentaci geometrie, kterou uchovává ve vektorovém datovém typu s názvem **`SDO_GEOMETRY`**. V reprezentaci tabulky se tedy nachází jeden nebo více sloupců s tímto datovým typem pro uložení prostorových informací o každém záznamu. Tyto vektory mohou představovat několik podporovaných grafických primitiv, mezi které patří například **`Point`**, **`LineString`** či **`Polygon`**. Bod je jednouchý, neboť je popsán pouze dvěma souřadnicemi, které nejčastěji představují zeměpisnou šířku a délku. Druhý zmíněný slouží pro definici posloupnosti přímek a polygon slouží pro vytvoření mnohoúhelníku. Geometrické útvary se velmi podobají těm, které jsou popsané v předchozích kapitolách o ostatních databázových systémech. Na obrázku 13 jsou zobrazeny veškeré podporované tvary:

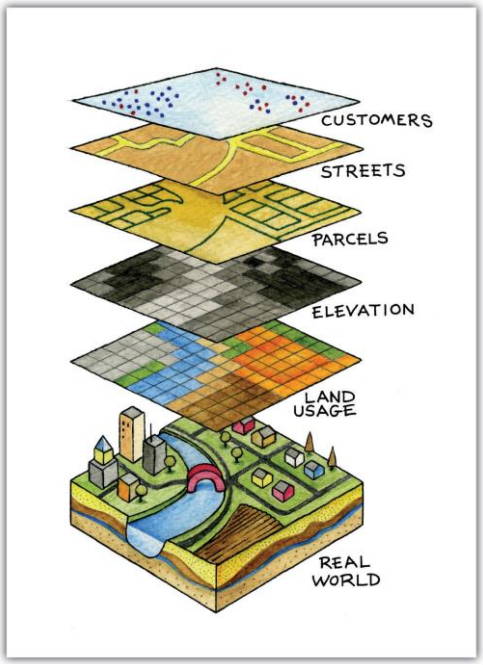



**Obrázek 13 - Podporované geometrický útvary v databázi Oracle Spatial [14]**

Dále bude představen datový model. Datový model popisuje strukturu a formát dat v informačních systémech a určuje vztahy mezi těmito jednotlivými prvky navzájem. Datový model Oracle Spatial and Graph se skládá z následujících prvků:

- **Element** – Element představuje použitý geometrický tvar, kterým může být pouze bod, přímka nebo polygon. Každá souřadnice v elementu je datově reprezentována jako dvojice souřadnic X a Y. Zjednodušeně řečeno si lze element představit jako základní stavební blok, ze kterého jsou následně skládány složitější geometrické útvary.
- **Geometrie** – Geometrie představuje kolekci několika elementů, tvořící tak nějaký komplexnější geometrický útvar. Kolekce může být buď homogenní (všechny elementy jsou stejné, např. množina bodů), anebo heterogenní (elementy jsou odlišné, např. bod a polygon).
- **Souřadnicový systém** – Souřadnicový systém je způsob, jakým se popisuje jednoznačná poloha na Zemi. Těchto systémů existuje celá řada, ale pouze tyto podporuje Oracle Spatial:
    - **Kartézský systém –** Určuje polohu pomocí os X, Y a případně i Z (pro třetí dimenzi) vzhledem ke stanovenému počátku.
    - **Geodetický systém** – Určuje polohu pomocí zeměpisné šířky a délky.
    - **Projekční systém** – Určuje polohu na Zemi pomocí kartézských souřadnic, kde Země je transformována na plochu v jedné rovině.
    - a **lokální systém** – Lokální souřadnice, které se nevztahují k určení polohy na Zemi, ale k určení různých objektů například v CAD systémech.
- **Tolerance** – Umožňuje definovat úroveň přesnosti prostorových dat., která řeší například nepřesnosti operací s reálnými čísly. Stručně řešeno určuje tolerance to, jestli jsou dva body s minimální odchylkou stále na stejném místě. V geodetických systémech (definovaných pomocí zeměpisné šířky a délky) představuje tolerance počet metrů (například 10 znamená toleranci 10 metrů). V ostatních systémech vyjadřuje tolerance počet jednotek, které jsou spjaty s konkrétním souřadnicovým systémem.
- **Vrstva** – Vrstva není nic jiného než kolekce geometrií, které spolu souvisí a popisují jednu skutečnost. Příkladem může být to, že mapový podklad se může skládat z několika vrstev, kde jedna zobrazuje silnice, druhá popisuje četnost obyvatelstva a třetí zobrazuje teplotní

mapu. Vícevrstvou mapu znázorňuje obrázek 14, kde se celkový pohled na mapu skládá z celkem pěti vrstev, kde každá popisuje jednu skutečnost:




**Obrázek 14 - Ukázka vícevrstvé mapy [15]**

Další důležitou úlohou při práci s prostorovými daty v Oracle Spatial je samotné indexování, které je prováděno pomocí datové struktury R-strom (kapitola 1.1.1). Zajímavostí je, že v souvislosti s vymezením hranic prostoru u této datové struktury se pro 3D prostor nepoužívá MBR, ale MBV[14] Toto indexování je velmi výhodné v praktické části aplikace, kde se vytváří dotazy na obdélníkovou plochu a ta je v podobě obrázku navrácena do zobrazovacího zařízení.

Datovým typem pro ukládání prostorových dat je typ s názvem **SDO_GEOMETRY**. Tento datový typ se skládá z několika atributů:

```
SDO_GTYPE          NUMBER
SDO_SRID           NUMBER
SDO_POINT          SDO_POINT_TYPE
SDO_ELEM_INFO      SDO_ELEM_INFO_ARRAY
SDO_ORDINATES      SDO_ORDINATE_ARRAY
```

Těchto pět informací kompletně popisuje jeden objekt v prostoru. Jednotlivé údaje mají následující význam:

- **SDO_GTYPE** – Jedinečný číselný identifikátor určující typ geometrického objektu podle tabulky 4, kde údaj, který je vyžadován typem **SDO_GOMETRY**, je v posledních třech

[14] Minimum Bounding Volumes

sloupcích. Lze si všimnou, že dimenze se odlišuje počáteční cifrou a poslední cifra určuje typ geometrického tvaru:

**Tabulka 4 - SDO_GTYPE identifikátory**

| Číselný identifikátor | Popis | 2D | 3D | 4D |
|---|---|---|---|---|
| 1 | Bod | 2001 | 3001 | 4001 |
| 2 | Přímka | 2002 | 3002 | 4002 |
| 3 | Polygon | 2003 | 3003 | 4003 |
| 4 | Heterogenní kolekce bodů, přímek či polygonů | 2004 | 3004 | 4004 |
| 5 | MultiPoint – více bodů | 2005 | 3005 | 4005 |
| 6 | MultiLineString – více přímek | 2006 | 3006 | 4006 |
| 7 | MultiPlygon – více polygonů | 2007 | 3007 | 4007 |

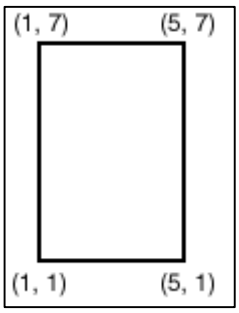
- **`SDO_SRID`** – Hodnota jednoznačně identifikující souřadnicový systém. Je-li hodnota NULL, poté není asociován s objektem žádný souřadnicový systém. Přijatelné hodnoty lze nalézt v referenci [17].
- **`SDO_POINT`** – Tento atribut ukládá souřadnice X, Y a Z. Používá se pouze v případě, kdy se v prostoru nachází jen body (například bodová reprezentace grafického tělesa). V jiných případech se doporučuje nastavovat na NULL a pro vyjádření geometrií použít následující dva parametry.
- **`SDO_ELEM_INFO`** – Tento atribut nabývá tří číselných hodnot, které určují způsob reprezentace dat uložených v atributu **`SDO_ORDINATES`**. Jednotlivé číselné hodnoty mají tento význam:
    - **`SDO_STARTING_OFFSET`** – Určuje první pozici prvku v **`SDO_ORDINATES`**. Indexuje se od čísla jedna a většinou nabývá hodnoty právě jedna = první hodnota v poli.
    - **`SDO_ETYPE`** – Identifikuje typ útvaru. Používá hodnoty z tabulky, kterou lze nalézt v referenci [16], kde ke každému útvaru je rovněž přiřazen parametr třetí.
- **`SDO_ORDINATES`** – Tento atribut představuje pole hodnot, které jsou souřadnicemi jednotlivých bodů tvořících požadovaný geometrický útvar. Atribut se používá pouze v případě definování předchozího atributu v tomto seznamu.

Příklad vytvoření tabulky s jedním sloupcem určeným pro prostorová data vypadá následovně:

```
CREATE TABLE states
  (
     description  VARCHAR2(30),
     geometry     SDO_GEOMETRY
  );
```

Výsledkem následujícího dotazu vznikne jednoduchý obdélník znázorněný na obrázku 15:

```
INSERT INTO SPATIAL_TABLE VALUES(
  SDO_GEOMETRY(
    2003,  -- 2D polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- 1003 = obdélník
definovaný levým dolním a pravým horním rohem
    SDO_ORDINATE_ARRAY(1,1, 5,7) -- definice 2 požadovaných
bodů dle požadavku typu 1003
  )
);
```




**Obrázek 15 - Znázornění příkladu vytvoření SDO_GEOMETRY [16]**

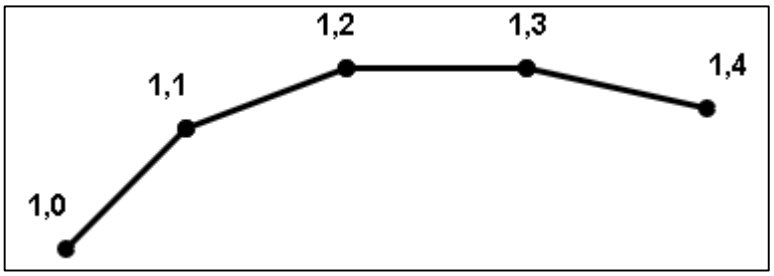
# 2 ÚVOD DO PROBLEMATIKY MODELU ŽELEZNIČNÍ SÍTĚ

Jak již bylo zmíněno v úvodu, tak tato diplomová práce navazuje na práci od absolventa Ing. Radka Prokeše, který své dílo zhotovil v roce 2011. Vzhledem k tomu, že zmíněný absolvent provedl rozsáhlou analýzu vstupních dat a problému sestavení modelu železniční sítě, tak zde budou uvedeny ve stručnosti základní fakta, která se přímo týkají této rozšiřující práce a jsou pro pochopení této problematiky klíčová. Většina informací se tedy bude odkazovat na zmíněnou závěrečnou práci pod referencí [18].

## 2.1 Základní pojmy

Nyní budou představeny základní pojmy, se kterými se čtenář dále setká jak v teoretické, tak i praktické části diplomové práce. S ostatními pojmy je možné se setkat v uvedené diplomové práci v referenci [18]. Základními pojmy jsou:

- **Hektometrovník** – Jednotka staničení. Staničení je systém pro označení kolejí podle jejich vzdálenosti od hlavního města Prahy, dále pak ve směru k okresním městům, dále směrem k menším městům a obcím. Hektometrovník je označením pro staničení v intervalu 100 metrů. Ukázka staničení pomocí hektometrovníku je znázorněna na obrázku 16 s jednou kolejí:




**Obrázek 16 - Staničení s hektometrovníky [autor]**

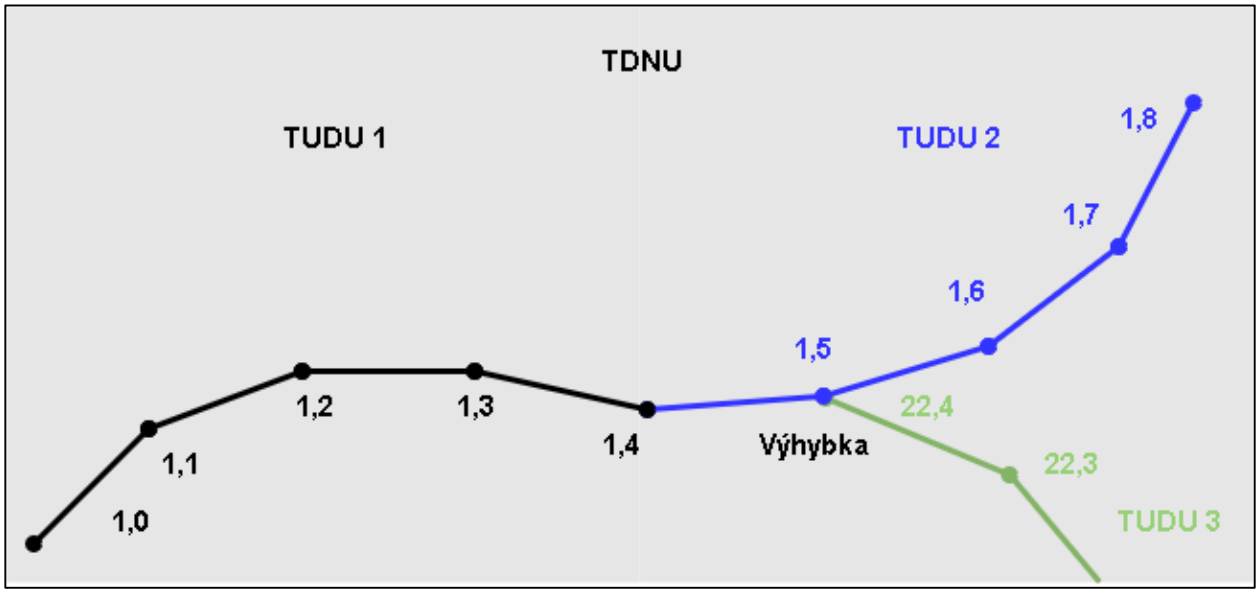
- **TUDU** – Traťový definiční úsek je logickým celkem některé posloupnosti navzájem navazujících hektometrovníků. Obrázek 16 lze prohlásit za jeden TUDU.

- **TDNU** – Traťový definiční nadúsek je logickým celkem posloupnosti na sebe navzájem navazujících TUDU.

- **Výhybka** – Na výhybce dochází ke sbíhání či rozcházení kolejí.

Na obrázku 17 lze znázornit všechny zmíněné pojmy. Je možné si povšimnout tří definičních úseků, které jsou seskupeny do jednoho logického nadúseku TDNU. Za zmínku je určitě zajímavé uvést fakt, že mezi hodnotami hraničních hektometrovníků některých TUDU vzniká velký skokový rozdíl. Důvodem je skutečnost způsobená tím, co bylo zmíněno u pojmu

staničení, a to tedy problémem vzniklým vlivem různého počátku při výstavbě železnice. Fiktivním příkladem může být střet kolejí, kde první počátek by byl v Praze a druhý v Pardubicích. Výstavba skončí v bodě střetu s nějakou hodnotou hektometrovníku, od které se budou hodnoty ostatních hektometrovníků v protějším směru neracionálně snižovat. Vždy je tudíž nutné s hodnotami hektometrovníků spojit rovněž počátek, od kterého bylo staničení zahájeno. Zelená kolej na obrázku je právě tím případem, kdy se začala kolej budovat z jiného směru, a proto skončila na výhybce s hodnotou odlišnou, než mají ostatní hektometrovníky. V praktické části aplikace bylo zjištěno, že se TDNU většinou nevětví, jak je znázorněno na obrázku, nicméně i s větvením je třeba počítat a přizpůsobit tomu sestavovací algoritmus.




**Obrázek 17 - Zobrazení modelu železniční sítě [autor]**

## 2.2 Popis datových souborů

Dle informací z diplomové práce, na kterou tato navazuje, byly poskytnuty datové soubory od společnosti SŽDC-TÚDC. Tyto soubory netřeba analyzovat v této práci, neboť je předpokládáno, že jsou dostačující pro vygenerování modelu železniční sítě. Datové soubory byly dle informací dodány ve formátu DBF[15]. Struktura toho souboru reflektuje tabulku v databázi, neboť obsahuje hlavičky jednotlivých sloupců a k nim příslušná data v podobě řádků. Z důvodu stručnosti budou v následujícím seznamu uvedeny pouze ty soubory včetně důležitých atributů, které byly nezbytné pro vykonání praktické aplikační části. Pro kompletní popis jednotlivých souborů lze nahlédnout do reference 18. Použitými soubory jsou:

[15] DataBase File – formát používaný pro import a export dat v databázových systémech

- **hmhkp.dbf** – Nejrozsáhlejší soubor ze všech ostatních, který obsahuje data o všech hektometrovnících v železniční síti. Atributy, které byly nezbytně nutné pro vybudování modelu železniční infrastruktury, jsou uvedeny v tabulce 5:

**Tabulka 5 - Popis atributů souboru s hektometrovníky**

| Název atributu | Význam pro hektometrovník |
|---|---|
| TUDU | Přítomnost v konkrétním TUDU |
| KHMEK | Vzdálenost v kilometrech |
| KV | Umístění na koleji (K) či výhybce (V) |
| LON | Souřadnice zeměpisné délky |
| LAT | Souřadnice zeměpisné šířky |

- **sr70wgs.dbf** – Tento soubor obsahuje seznam všech stanic a zastávek. Následující atributy byly vybrány pro potřeby aplikace. Bystrý čtenář si jistě všimne dvou GPS souřadnic a položí si otázku, proč nestačí pouze jeden pár. Tomu lze odpovědět tak, že to bylo nutné řešení pro potřeby aplikace. První dvě lze označit za korektní, druhý pár jako náhradní, není-li první pár k dispozici. Práce s těmito dvěma páry souřadnic bude popsána v samostatné kapitole zabývající se algoritmem importu dat. Vybranými atributy jsou:

**Tabulka 6 - Popis atributů souboru se stanicemi**

| Název atributu | Význam pro stanici |
|---|---|
| NAZEV | Název stanice |
| KM_POL | Vzdálenost v kilometrech |
| TUDU | Přítomnost v konkrétním TUDU |
| DPM_LON | Souřadnice zeměpisné délky |
| DPM_LAT | Souřadnice zeměpisné šířky |
| LON | Souřadnice zeměpisné délky |
| LAT | Souřadnice zeměpisné šířky |

- **tdnu_17_0.dbf** – Tento datový soubor umožňuje seskupit jednotlivá TUDU do větších celků TDNU. Obsahuje tudíž údaje o jednotlivých nadúsecích. Pro potřeby aplikace byly vybrány následující atributy:

**Tabulka 7 - Popis atributů souboru s nadúseky**

| Název atributu | Význam pro nadúsek |
|---|---|
| TDNU | Označení nadúseku |
| TUDU | Traťový definiční úsek v tomto nadúseku |
| POR | Pořadí traťového definičního úseku v tomto nadúseku |

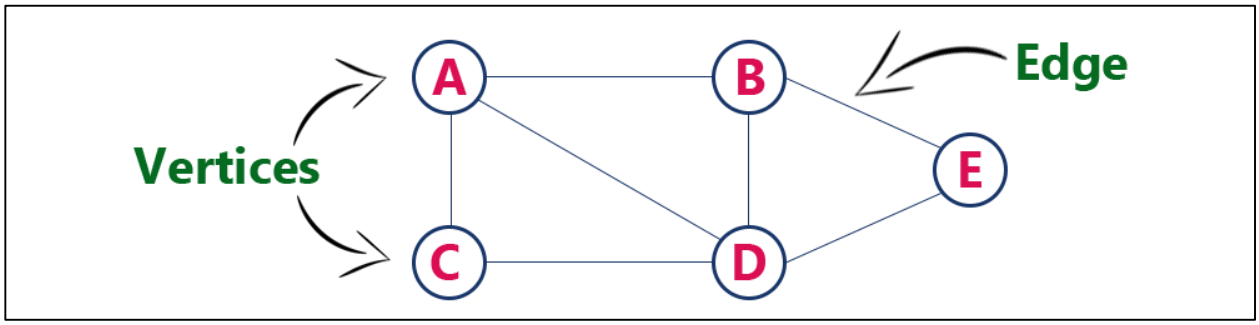
## 2.3 Datová struktura pro uchování topologie modelu železniční sítě

Do výsledné datové struktury je nutné ukládat stanice a hektometrovníky. Dle analýzy plyne, že stanice a hektometrovníky budou představovat uzly. Každý uzel bude mít alespoň jednoho souseda. Z těchto faktů plyne, že za nejvhodnější datovou strukturu je možné použít ADT Graf.

ADT Graf odráží binární relaci v množině a je bipartitní (nehomogenní) datovou strukturou. To znamená, že prvky grafu nejsou tvořeny pouze kolekcí objektů jednoho datového typu, protože pro plnohodnotnou reprezentaci ADT Grafu je nutné použít datové typy pro vrcholy a pro hrany.

Co se týče přístupu k prvkům ADT Grafu, tak byl použit vrcholově orientovaný přístup. To znamená, že vstupní branou do této datové struktury jsou vždy vrcholy, přičemž zvolený vrchol může být libovolný. Pro zajištění konstantní asymptotické složitosti je použit přístup pomocí hash kódu. Ten zajísti velmi rychlý přístup ke konkrétnímu vrcholu v této datové struktuře.

Obrázek 18 znázorňuje strukturu ADT Graf pro pět vrcholů (Vertices) A,B,C,D,E a sedm hran (Edge):




**Obrázek 18 - Znázornění datové struktury ADT Graf [19]**

Uspořádání grafu se skládá, jak již bylo zmíněno v odstavci výše, ze dvou datových typů. Pro datový typ **`Hrana`** je nutné uchovávat informace o incidentních vrcholech. **`Hrana`** musí mít vždy dva vrcholy, které spojuje. Datový typ **`Vrchol`** uchovává oproti hraně více informací, neboť tvoří pomyslnou křižovatku, kde se může sbíhat velké množství hran. Odpovědností vrcholu je tudíž evidovat seznam incidentních hran. Pro zachování co nejuniverzálnější podoby této datové struktury se v praktické části pracuje s generickými datovými typy jak pro vrcholy, tak i pro hrany. Následující ukázky zdrojového kódu vystihují vnitřní uspořádání grafu, ve kterém **`V`** označuje generický datový typ pro vrcholy a **`E`** pro hrany.

**Datový typ Vrchol:**

```
private class Vertex {
    private final V data; // data vrcholu
    private final Collection<Edge> edges; // seznam hran
}
```

**Datový typ Hrana:**

```
private class Edge {
    private final Vertex v1; // vrchol 1
    private final Vertex v2; // vrchol 2
    private final E data; // data hrany
}
```

**Kolekce pro rychlý přístup k vrcholům:**

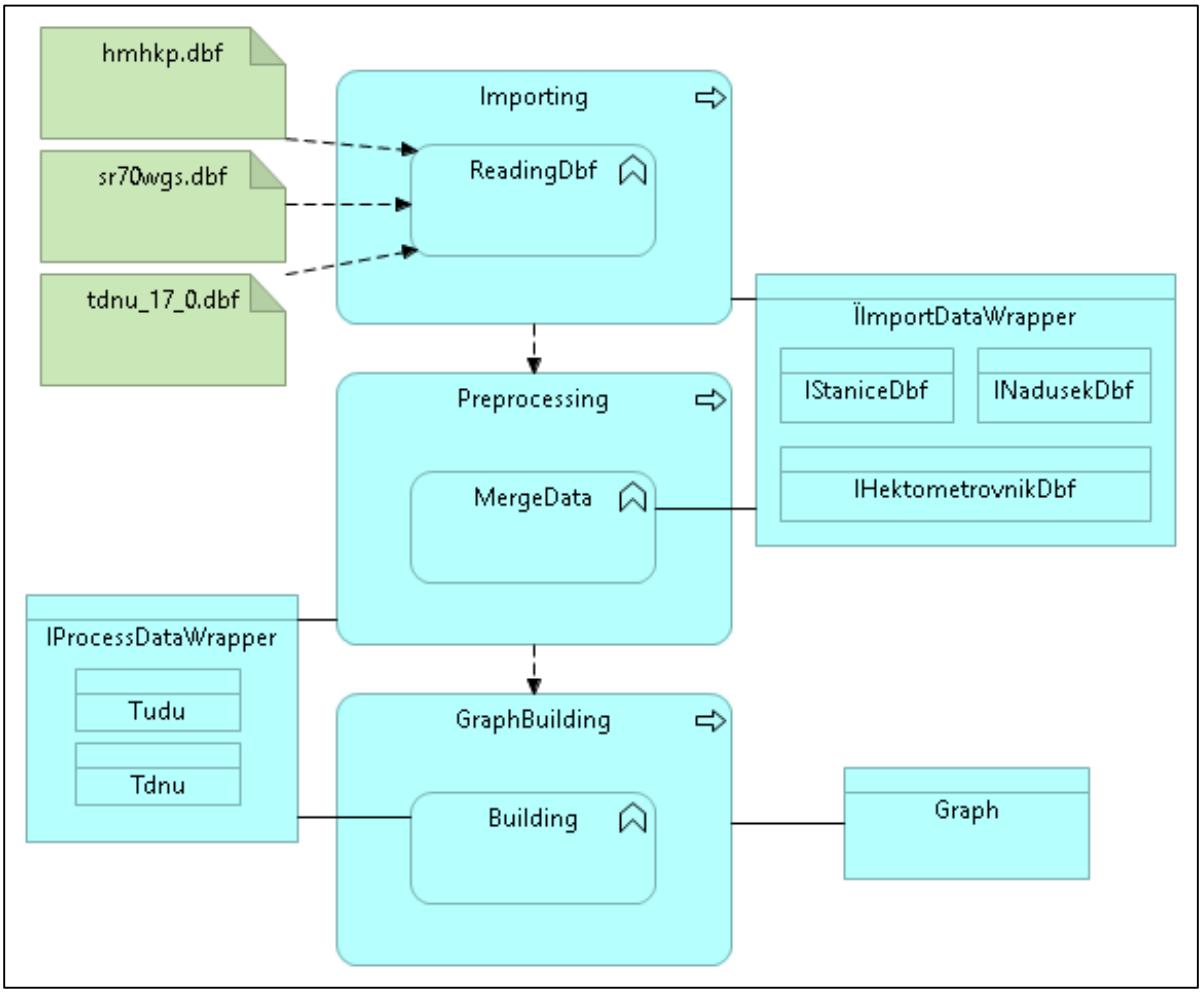
```
// klíčem jsou data vrcholu (hektometrovník / stanice)
private final HashMap<V, Vertex> verticies;
```

# 3 NÁVRH A IMPLEMENTACE APLIKACE

V úvodu bude velmi stručně popsán algoritmus pro generování modelu železniční sítě. Z důvodu rozsáhlých operací byl algoritmus rozdělen do tří po sobě navazujících procesů, kde každý z nich má za úkol provést jednu konkrétní činnost. Tato činnost přebírá nějaký vstup a na základě tohoto vstupu generuje výstup, jenž je zdrojem dat pro další fázi. Tento postup byl zvolen z jednoho prostého důvodu. Pokud je některá fáze upravena, tak tato modifikace neovlivní korektnost výpočtů následujících procesů. Zmíněnými třemi operacemi jsou:

- proces importování dat,
- proces slučování dat
- a proces sestavování modelu železniční sítě.

Na obrázku 19 je znázornění jednotlivých procesů a datových tříd, které jsou mezi nimi předávány. Diagram bude znázorněn v souladu s jazykem pro popis softwarové architektury Archimate verze 3.0. Diagram má tuto podobu:




**Obrázek 19 - Diagram algoritmu generování modelu železniční sítě [autor]**

Na první pohled se může zdát diagram jako velmi složitý, ale ve skutečnosti je jednoduchý. Počátkem celého procesu je existence tří zobrazených souborů (artefaktů), které jsou předány do aplikačního procesu importování dat. Aplikační proces načítání využívá funkce čtení DBF souborů pro načtení dat z artefaktů. Výstupem prvního procesu je datový objekt, který obsahuje načtené hektometrovníky, stanice a nadúseky. Načtená data jsou uložena v objektu s datovým typem **`IImportDataWrapper`** a jsou jako celek předána následující fázi pro zpracování, kterou je proces slučování dat nazvaný jako Preprocessing. Ten pomocí funkce s názvem **`MergeData`** sloučí data do logických celků, které jsou uloženy ve formě traťových definičních úseků a nadúseků v datovém objektu typu **`IProcessDataWrapper`**. Pomocí tohoto objektu je možné sestrojit konečnou datovou strukturu ADT Graf, k čemuž slouží další aplikační proces s názvem GraphBuilding. Podobně pojmenovaná funkce **`Building`** vytvoří datový objekt typu **`Graph`**, který je výstupem celého algoritmu sestavování modelu železniční sítě.

## 3.1 Proces importování dat

Proces importování dat je jednoznačně klíčovou fází, neboť vznikne-li nějaká forma nekonzistence dat již v tomto prvním kroku, pak mohou nastat problémy v ostatních procesech. Proto je nutné provádět určitou formu kontroly konzistence již v prvopočátku algoritmu sestavení modelu železniční sítě.

### 3.1.1 Načítání DBF souborů

Klíčovými daty pro import jsou datové soubory popsané v kapitole 2. Tyto soubory jsou v binárním formátu DBF, který je nutné rozložit na hlavičku a na užitečná data. Pro tyto účely byla použita dostupná open-source knihovna dbf-reader, která poskytuje pár základních tříd pro načítání DBF souborů. Třídy této knihovny lze popsat v následujícím seznamu:

- **`DbfRowProcessor`** – Umožnuje zpracovávat jednotlivé řádky DBF souboru.
- **`DbfRowMapper<T>`** – Přetvoří řádek na programátorem zvolený datový typ T pomocí metody **`T mapRow(Object[])`**.
- **`DbfProcessor`** – Nabízí statické metody, jejichž argumenty zahrnují cestu k danému souboru DBF a poté jedno z uvedených dvou rozhraní výše v seznamu, které definuje způsob práce se souborem DBF.

Jak je patrné z předchozího seznamu, tak pro tuto aplikaci je nejvhodnější využít možnost mapování jednotlivých řádků na vlastní datové typy, které představují jednotlivé prvky modelu železniční sítě (hektometrovníky, stanice a nadúseky). Pro potřeby grafického uživatelského

rozhraní je však vhodné využít rovněž třídu **`DbfRowProcessor`**, která je použita čistě pro zjištění počtu řádků souboru. Tato informace totiž umožní zobrazit uživateli pokrok v načítání jednotlivých datových souborů. Celá operace načtení jednoho souboru tedy spočívá v získání počtu řádků, ze kterého se následně určí procentuální nárůst pokroku po zpracování jednoho záznamu. Tento pokrok je zvyšován při mapování jednoho řádku na zvolený datový typ. Klíčový zdrojový kód pro tuto činnost vypadá následovně:

```
class BaseMapper<T> implements DbfRowMapper<T> {
    protected final IProgressUpdatable progressUpdatable;
    protected final double incrementValue;
    @Override
    public final T mapRow(Object[] objects) {
        T data = doMap(objects);
        progressUpdatable.addProgress(incrementValue);
        return data;
    }
    protected abstract T doMap(Object[] objects);
}
```

Jak je možné si povšimnout, tak se jedná o abstraktní třídu, která byla použita z důvodu dodržování čistého kódu, neboť pro všechny potomky jsou tyto informace identické. Třída obsahuje dva základní atributy. První s názvem `progressUpdatable` představuje referenci na komponentu grafického uživatelského rozhraní a druhý atribut představuje procentuální nárůst pokroku po zpracování jednoho řádku. Hlavní činnost lze spatřit v první metodě `mapRow`, ve které se pomocí metody potomka `doMap` získá požadovaný objekt z jednoho řádku. Následuje aktualizace pokroku o danou hodnotu v proměnné `incrementValue`.

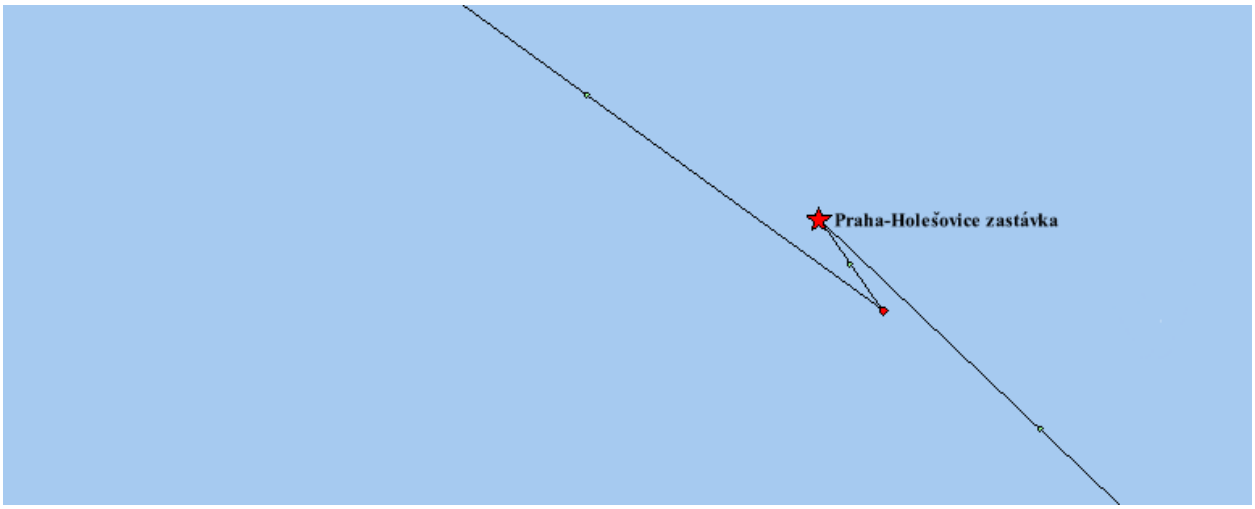
### 3.1.2 Nestandardní situace při načítání dat

Prvním problémem je samotný princip mapování. Jak si lze povšimnout z parametrů mapovacích metod, tak pracují se všemi načtenými daty pod stejným datovým typem **`Object`**. To je vcelku logické řešení, protože při načítání binárních dat není možné určit datový typ jednotlivých hodnot, a tak je nevyhnutelné použít techniku přetypování na požadovaný datový typ. Proto bylo nutné provést analýzu dat v jednotlivých datových souborech pomocí některého z dostupných komerčních či nekomerčních tabulkových procesorů. Z tabulkového procesoru bylo možné jednoznačně určit datový typ jednotlivých hodnot a rovněž index sloupců. Index sloupce je potřeba proto, že řádek představuje pole objektů a k jednotlivým objektům se přistupuje pomocí přesného určení pozice v daném poli. Tento přístup má, sic za cenu explicitního určení indexu, velkou výhodu v podobě nízké přístupové asymptotické složitosti.

Druhým problémem, který se týká rovněž mapování dat, bylo určení kódování znaků u názvu stanic. To se týká převádění jednotlivých bajtů na řetězec, kdy je nutné specifikovat i to, v jakém kódování byl tento řetězec uložen. Řešení tohoto problému bylo vcelku jednoduché. Za účelem nalezení vhodného kódování znaků byl vytvořen jednotkový test (Unit test), který vypsal na konzoli název stanice ve všech podporovaných kódováních. Jak je vidět na obrázku 20, tak s testováním i ostatních stanic bylo nalezeno správné kódování s označením **IBM852**. Výstup jednotkového testu byl následující:

```
IBM437: Chrást nad Sázavou
IBM500: äÇÊµËÈ□>/À□ëµ:/Î?Í□□□
```

byly pro znázornění modelu nepřijatelné. Jak je patrné z obrázku 21, tak nepřesné souřadnice stanice způsobily nežádoucí zalomení tratě. V tomto případě se stanice podle hodnot hektometrovníků nachází za červeným puntíkem směrem zleva doprava, ale špatné geografické souřadnice však způsobily následující problém:



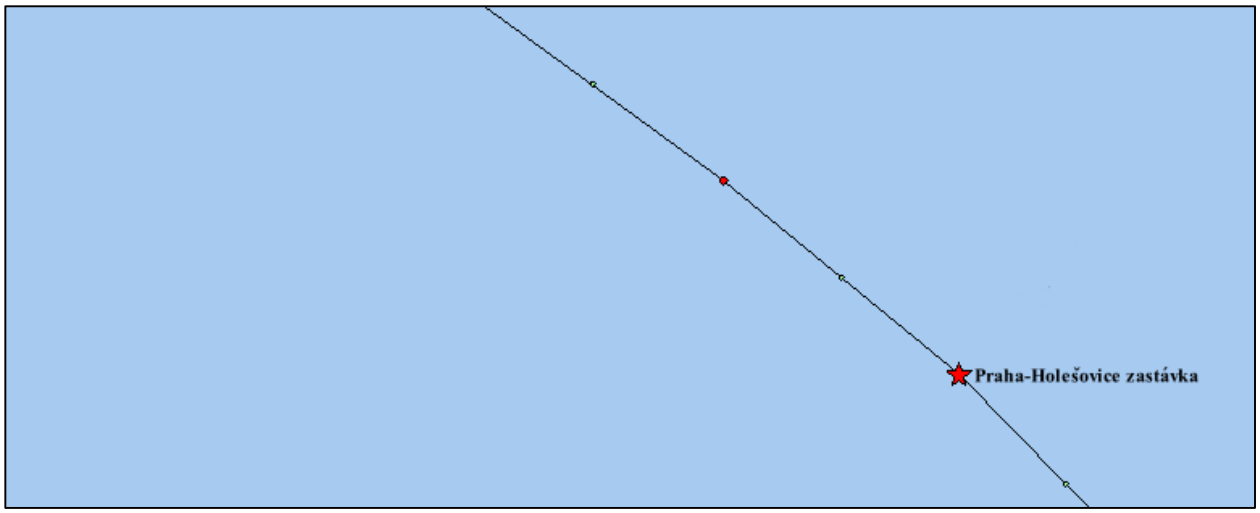

**Obrázek 21 - Zobrazení špatných souřadnic stanic [autor]**

Pro řešení tohoto problému bylo potřeba provést analýzu vstupních dat. Při analýze se zjistilo, že soubor se stanicemi obsahuje dva páry souřadnic. Prvním pokusem korekce bylo tedy použití tohoto druhého páru souřadnic. Toto řešení však vedlo k problému, že se některé stanice nacházely vedle afrického kontinentu, což by bylo pro stanice modelu železniční sítě Čech zcela nepřijatelné. Středem zájmu zkoumání se tedy staly tyto africké zastávky. Nalezení řešení spočívalo opět v provedení analýzy vstupních dat, kde byly objeveny prázdné hodnoty u zmíněných stanic. Na obrázku 22 si lze všimnout dvou výskytů chybových řádků. Co se týče prvního zvýrazněného řádku se stanicí Droužkovice, tak zde bylo řešení použití nekorektních, ale alespoň dostupných souřadnic. U dalšího zvýrazněného řádku však nastává problém, protože všechny souřadnice jsou nevyplněné. Proto je zapotřebí tyto stanice nezahrnovat do modelu železniční sítě, protože je nelze vnést do vícerozměrných dat. Problémová vstupní data vypadala následovně:

| | | | | |
|---|---|---|---|---|
| Peruc | 50,335933210 | 13,958818880 | 50,332818670 | 13,961232830 |
| Litvínov město | 50,591353050 | 13,613974850 | 50,591365900 | 13,613921650 |
| Meziměstí | 50,626684660 | 16,242312590 | 50,626519900 | 16,242388200 |
| Dobříkov u Chocně | 49,995779200 | 16,136249940 | 49,995771570 | 16,136371630 |
| Telce | 50,311905220 | 13,971680280 | 50,311862860 | 13,971873630 |
| Droužkovice | 0,000000000 | 0,000000000 | 50,428506700 | 13,420786600 |
| Březno u Chomutova | 50,395877620 | 13,421027160 | 50,395894750 | 13,420855400 |
| Březno u Chomutova, kolejiště DNT | 0,000000000 | 0,000000000 | 0,000000000 | 0,000000000 |
| Klobuky v Čechách | 50,298281310 | 13,996593960 | 50,299284270 | 13,996444050 |
| Libočany | 50,330340500 | 13,515441780 | 50,330342540 | 13,515444730 |
| Hynčice | 50,624329790 | 16,288205880 | 50,624313390 | 16,288779590 |
| Vraný | 50,309652510 | 14,000682390 | 50,309646390 | 14,000720200 |
| Holetice | 50,385507890 | 13,452850930 | 50,385489760 | 13,453174030 |

**Obrázek 22 - Ukázka neplatných dat se stanicemi [autor]**

Je-li shrnuta kompletní korektura problému se stanicemi, tak ideálním případem jsou stanice, které mají dostupné a zároveň správné souřadnice. Těchto stanic je naštěstí veliké procento. Poté jsou stanice, kde muselo dojít k použití druhých, ale nekorektních souřadnic. To způsobuje fakt, že v modelu železniční sítě se bude nacházet malé procento stanic, které budou vytvářet nerovnosti jako na obrázku 21. Toto řešení bylo zvoleno jako menší zlo, než aby byla celá stanice algoritmem zamítnuta. Ostatní stanice, které nemají žádné dostupné souřadnice, jsou kompletně zamítnuty, neboť je nelze zobrazit. Je-li aplikována tato oprava na data z obrázku 21, pak je dosaženo následujícího výsledku na obrázku 23:




**Obrázek 23 - Výsledek korektury souřadnic problémových stanic [autor]**

Při importování bylo nutné rovněž určit, co se stane se stanicemi, které neměly žádný záznam traťového definičního úseku. U takových stanic by bylo složité určit, do které části železniční mapy spadají. Po přezkoumání názvů stanic bylo však zjištěno, že se jedná převážně o virtuální stanice, které by nemělo smyl v modelu železniční sítě zobrazovat.

### 3.1.3 Ukládání načtených dat

Načtená data jsou ukládána do jednoho společného objektu datového typu **`IImportDataWrapper`**. Ten obsahuje chytré kolekce pro efektivní ukládání načtených objektů. Všechny kolekce jsou typu **`HashMap<String, Collection<T>>`**. Použití tohoto typu struktury bylo zapříčiněno požadavkem na velmi rychlý přístup k velkému množství dat pomocí převedení klíče na hash kód, což zajistí konstantní asymptotickou složitost. Klíče jsou u všech struktur tvořeny textovým řetězcem, ale každý řetězec může odpovídat odlišnému atributu. Pomocí klíče je možné získat kolekci načtených objektů. Hektometrovníky a stanice jsou seskupeny dle TUDU, což umožňuje snadnější slučování příbuzných hektometrovníků se stanicemi. Nadúseky jsou seskupeny dle TDNU, takže se podle tohoto atributu seskupují příbuzné

traťové definiční úseky, což umožňuje jejich snadnější propojení. Tabulka 8 určuje, jakými hodnotami je parametrizována tato datová struktura pro jednotlivé prvky železniční sítě:

**Tabulka 8 - Tabulka parametrizace struktury HashMap**

| Prvek železniční sítě | String | T | Typ kolekce |
|---|---|---|---|
| Hektometrovník | TUDU | IHektometrovnikDbf | LinkedList |
| Stanice | TUDU | IStaniceDbf | LinkedList |
| Nadúsek | TDNU | INadusekDbf | TreeSet |

Za povšimnutí jistě stojí typ kolekce u nadúseků, kde je použita datová struktura **`TreeSet<T>`**. Tato, jak z názvu plyne, hierarchická struktura, představuje množinu, která nepovoluje duplicitní prvky a umožňuje rychlý přístup vlivem seřazování prvků. Podmínky seřazení lze měnit pomocí třídy **`Comparator<T>`**. Pro účely této aplikace se nadúseky ve skupině řadí podle atributu udávající pořadí traťového definičního úseku v nadúseku. To umožní jednoznačné napojení, neboť pomocí tohoto atributu lze napojovat úseky jednoduše za sebou. Seřazená data v rámci jednoho TDNU mohou vypadat následovně:

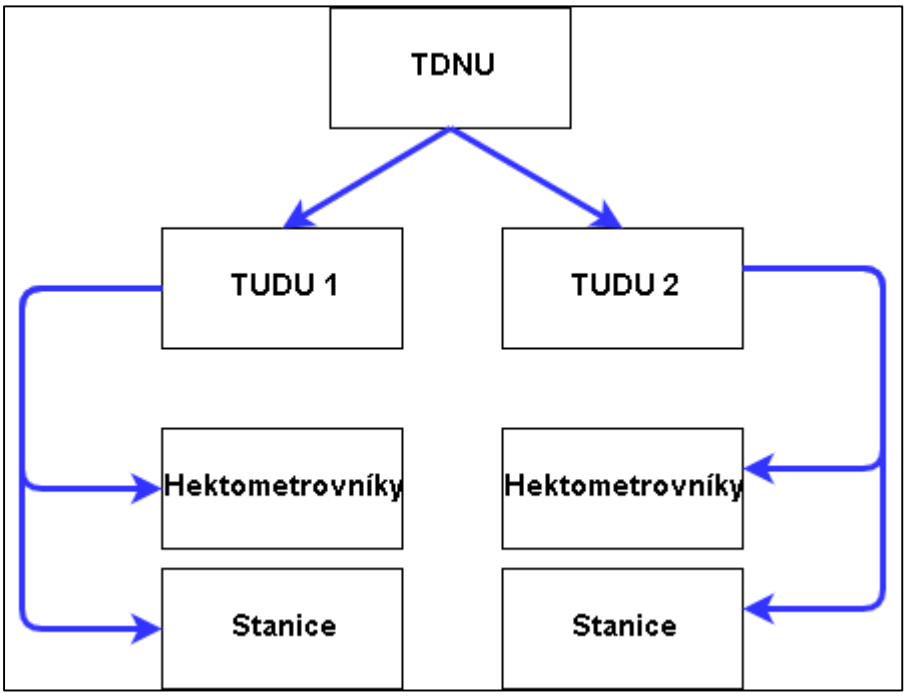
**Tabulka 9 - Ukázka složení TUDU objektu**

| TUDU | Pořadí |
|---|---|
| 0801B1 | 1 |
| 0801BA | 2 |
| 080121 | 3 |
| 0801C1 | 4 |

Co se týče pojmenování, tak může čtenáře napadnout otázka, proč jsou u některých názvů přítomny předpony tvořené písmenem velké I. Tento způsob tvoření názvů je jedna z možných konvencí pro pojmenování rozhraní. Celá aplikace je založena na technice programování proti rozhraní pro zajištění co největší nezávislosti programu na konkrétní implementaci třídy. Tato technika zaručuje velmi snadnou cestu, jak modifikovat stávající kód bez ovlivnění ostatních tříd (modularita kódu) a zároveň limituje objekt na zveřejnění pouze důležitých metod. Kód jako takový vede samozřejmě i k úspoře a znovupoužitelnosti. V předchozím odstavci je rozhraním třída **`Comparator<T>`**, která předepisuje metodu **`int compare(T,T)`**, kde její implementace závisí již na samotném programátorovi. Ten jen musí dodržet navrácení nějaké číselné hodnoty.

## 3.2 Proces slučování dat

Proces slučování (merging) je druhou fází algoritmu pro sestavování modelu železniční sítě. Principem slučování dat je logicky seskupit prvky železniční sítě do hierarchické struktury. To poslouží k rychlému vyhledávání a zároveň získávání příbuzných dat. Na obrázku 24 lze zpozorovat uspořádání železničních prvků do hierarchické podoby. Kořen je tvořen traťovým definičním nadúsekem (TDNU), který následně obsahuje seřazené traťové definiční úseky (TUDU). Jednotlivé traťové definiční úseky obsahují příbuzné hektometrovníky a stanice. Ukázka uspořádání vypadá na obrázku 24 následovně:




**Obrázek 24 - Hierarchické uspořádání prvků železnice [autor]**

Třída, jež zpracovává tento proces slučování, se nazývá **`MergeData`**. Tato třída zpracuje nejdříve všechny TUDU, ke kterým přiřadí příslušné hektometrovníky a stanice. Následně je sestavován TDNU. Tyto údaje jsou znovu uloženy v datové struktuře **`HashMap<String, T>`**, která umožňuje velmi rychlý přístup pomocí hashovaného klíče. Klíč je tvořen identifikátorem traťového definičního úseku a nadúseku podobně, jak je to popsáno v tabulce 8.

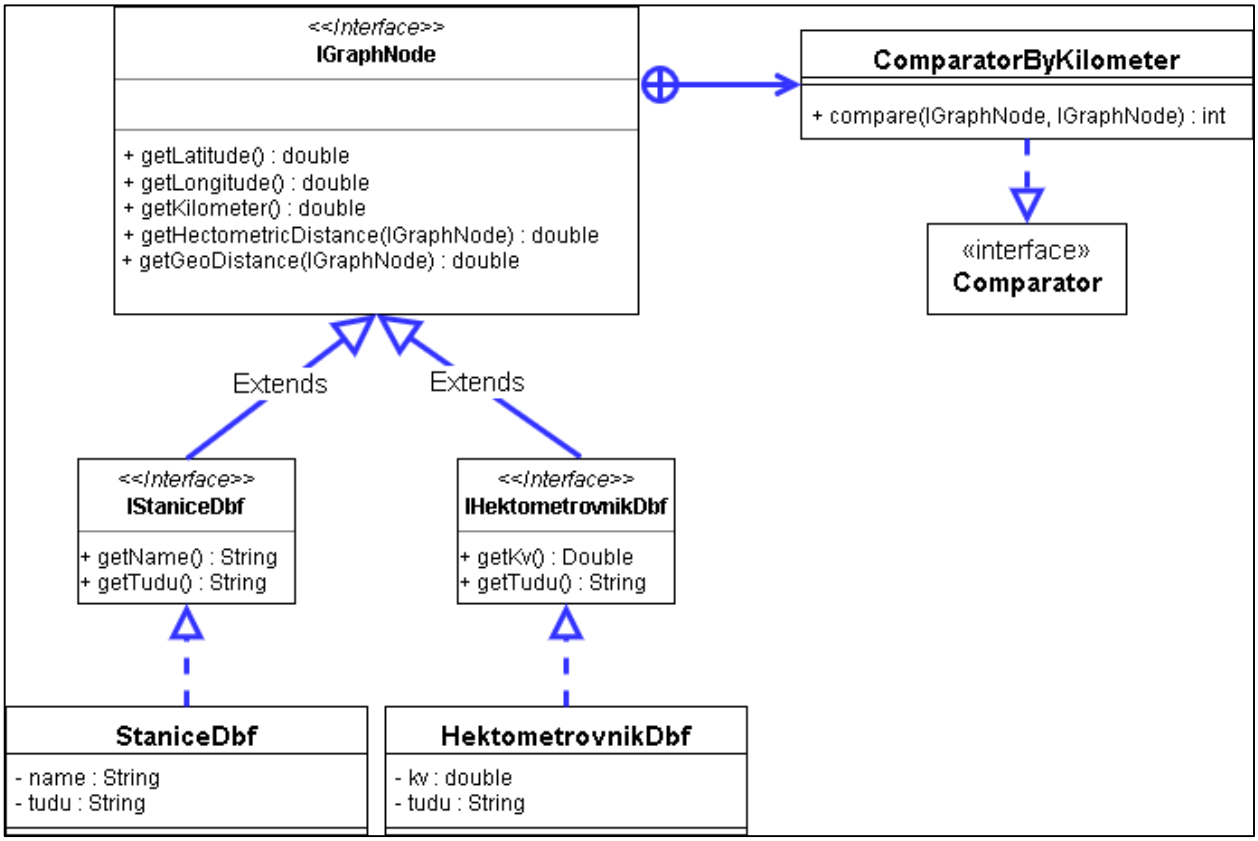
Tímto lze popis celého procesu ukončit. Očividně se jedná o velmi jednoduchou fázi algoritmu, která pouze seskupí příbuzná data. Více zajímavější je následující fáze popisující finální sestavování grafu, ve kterém se musí řešit celá řada situací pro vygenerování korektních výsledků.

## 3.3 Proces sestavování modelu železniční sítě

Proces sestavování je posledním procesem při generování modelu železniční sítě. Úkolem tohoto procesu je správně vložit hektometrovníky a stanice do datové struktury ADT Graf tak, aby byly mezi těmito prvky (v grafu vrcholy) korektně vytvořeny hrany. Co zcela jistě přichází na mysl je otázka, jak se chovat ke stanicím a hektometrovníkům zcela stejně. Odpověď se nachází v podkapitole 3.3.1.

### 3.3.1 Homogenizace hektometrovníků a stanic

Jelikož je datová struktura ADT Graf implementována pro práci s jedním datovým typem vrcholů, pak je nutné nějakým způsobem homogenizovat hektometrovníky a stanice do jednoho společného datové typu. K tomu poslouží jednoduchá technika programování proti rozhraní. Toto rozhraní se nazývá **`IGraphNode`** a předepisuje základní metody, které by měl každý prvek v síti obsahovat. Atributy, které jsou společné pro všechny prvky, jsou zeměpisná šířka a délka a hodnota vzdálenosti hektometrovníku. Třídy **`IHektometrovnikDbf`** a **`IStaniceDbf`** musí implementovat toto rozhraní. Obrázek 25, znázorňující UML diagram, zobrazuje uspořádání v podobě analytických tříd:




**Obrázek 25 - Diagram kooperujících tříd s IGraphNode [autor]**

Z diagramu je patrné využívání techniky programování proti rozhraní. Jak hektometrovníky, tak i stanice mají svá rozhraní, kde každé toto rozhraní dědí z **`IGraphNode`**. Uvedené rozhraní, jehož účelem je sjednotit přístup k oběma prvkům železniční sítě, předepisuje následující metody:

- **`String getLatitude()`** – Tato metoda vrací zeměpisnou šířku.
- **`String getLongitude()`** – Tato metoda vrací zeměpisnou délku.
- **`double getKilometer()`** – Tato metoda vrací hodnotu hektometrovníku.

Další dvě metody jsou označovány jako výchozí (default method). Výchozí metody jsou novinkou Java verze 1.8 a nabízí v samotném rozhraní nejenom definovat hlavičku metody, ale napsat i její implementaci. V těle této výchozí metody existuje přístup pouze k metodám, které jsou definovány v rozhraní. Proto u obou metod jak pro výpočet vzdálenosti podle hektometrovníků, tak i podle GPS souřadnic, stačí používat předchozí tří metody rozhraní. Důvodem výpočtu vzdálenosti mezi hektometrovníky (kdy je vzdálenost teoreticky vždy 100 metrů) je, že při sestavování modelu železniční sítě se vlivem nekorektních dat mohly spojit hektometrovníky či stanice, které od sebe uchovávají vzdálenost odlišnou sta metrů, a tak je i výpočet této vzdálenosti vhodný. Výpočty vzdálenosti však neovlivní algoritmus, neboť slouží pouze jako informativní položka při popisu hrany. Rozhraní obsahuje navíc vnitřní třídu **`ComparatorByKilometer`**, která slouží jako komparátor pro prvky s tímto rozhraním podle hodnot jejich kilometrů.

### 3.3.2 Sestavování jednotlivých TUDU

Prvním úkolem při generování modelu železniční sítě je sestavit grafy jednotlivých traťových definičních úseků. Vstupem do tohoto subprocesu je seznam všech traťových definičních úseků, které vznikly v předchozím procesu slučování dat. Pro každý tento TUDU vznikne graf, a tak výstupem subprocesu je kolekce všech grafů. Postup při mapování datového typu **`ITudu`** na ADT Graf je následující:

- vložení dat do struktury podporující seřazení,
- průměrování prvků na stejném kilometru
- a sestavení grafu.

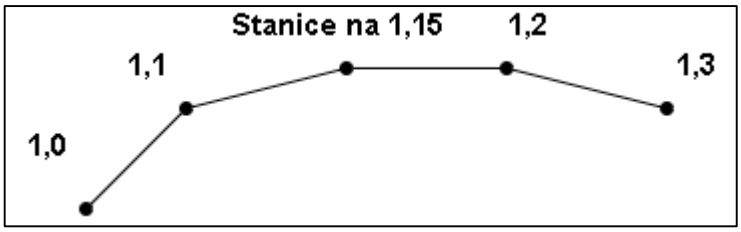
Prvním krokem je vložení všech hektometrovníků a stanic v daném traťovém definičním úseku do vhodné datové struktury, která bude zachovávat pořadí prvků dle zvoleného kritéria

uspořádání. Jako vhodná datová struktura byla zvolena struktura typu **`TreeMap<Double, List<IGraphNode>>`**. Klíčem je číselný datový typ reálné přesnosti, který představuje hodnoty kilometrů. Podle klíče struktura seskupuje prvky železniční sítě, které se nachází na daném kilometru. Důvodem uchování více prvků k jedné hodnotě vzdálenosti je situace, že se na jednom místě nachází více prvků. Tato událost ve skutečnosti nastává z důvodu existence souběžných kolejí. Proto bylo rozhodnuto, že se tyto souběžné koleje budou sjednocovat do jedné. Tuto operaci lze provádět pouze pro prvky, které jsou ve stejném TUDU a zároveň se nachází na stejné hodnotě kilometru. Proto je nutné provést proces průměrování jako první krok při sestavování grafu z traťového definičního úseku. Algoritmicky se tedy provádí činnost, ve které se vyberou ze struktury všechny klíče, jejichž hodnoty v podobě seznamů obsahují více než jeden počet prvků. Celý proces průměrování lze znázornit v tabulce 10. Průměrováním první dvojice hektometrovníků vznikne jeden hektometrovník, jehož GPS souřadnice jsou výsledkem aritmetického průměru zeměpisné šířky a délky. Co se týče druhé skupiny, tak v tomto případě je zapotřebí upřednostnit stanici před hektometrovníkem, aby nedošlo k absenci této stanice ve finálním grafu. Zároveň není v tomto případě vhodné provádět průměrování GPS souřadnic, protože potom by došlo k úpravě polohy stanice, a to by byl nežádoucí jev. Tabulka vypadá následovně:

**Tabulka 10 - Průměrování prvků železniční sítě**

| Klíč | Hodnoty | Operace |
|---|---|---|
| 1,4 | hektometrovník | Průměr obou hektometrovníků |
| | hektometrovník | |
| 1,5 | hektometrovník | Upřednostnění stanice bez úprav |
| | stanice | |

Po operaci průměrování jsou v datové struktuře seřazeny železniční prvky podle hodnot, na kterém kilometru se nachází a v každé skupině je pouze jeden prvek. V následujícím kroku se provádí vkládání prvků do datové struktury ADT Graf a vytváření hran mezi těmito prvky. Tím, že jsou prvky seřazeny podle kilometrů, je docíleno správného napojení tak, jak jdou prvky v železnici za sebou. Obrázek 26 zobrazuje výsledek procesu napojení prvků v rámci jednoho traťového definičního úseku:



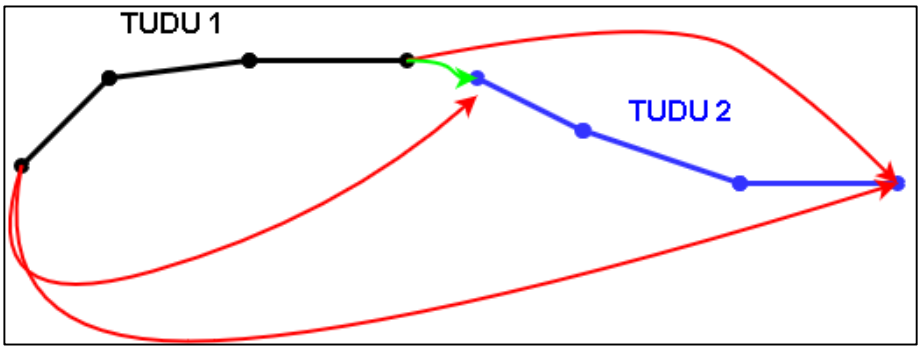

**Obrázek 26 – ATD Graf pro jeden traťový definiční úsek [autor]**

### 3.3.3 Sestavování jednotlivých TDNU

Propojení všech traťových definiční úseků v rámci jednoho nadúseku je nesložitějším krokem při generování modelu železniční sítě. Algoritmus byl několikrát změněn, neboť se ve vygenerovaném modelu nacházely vždy nějaké výjimky, které zhoršily výsledný dojem a přesnost.

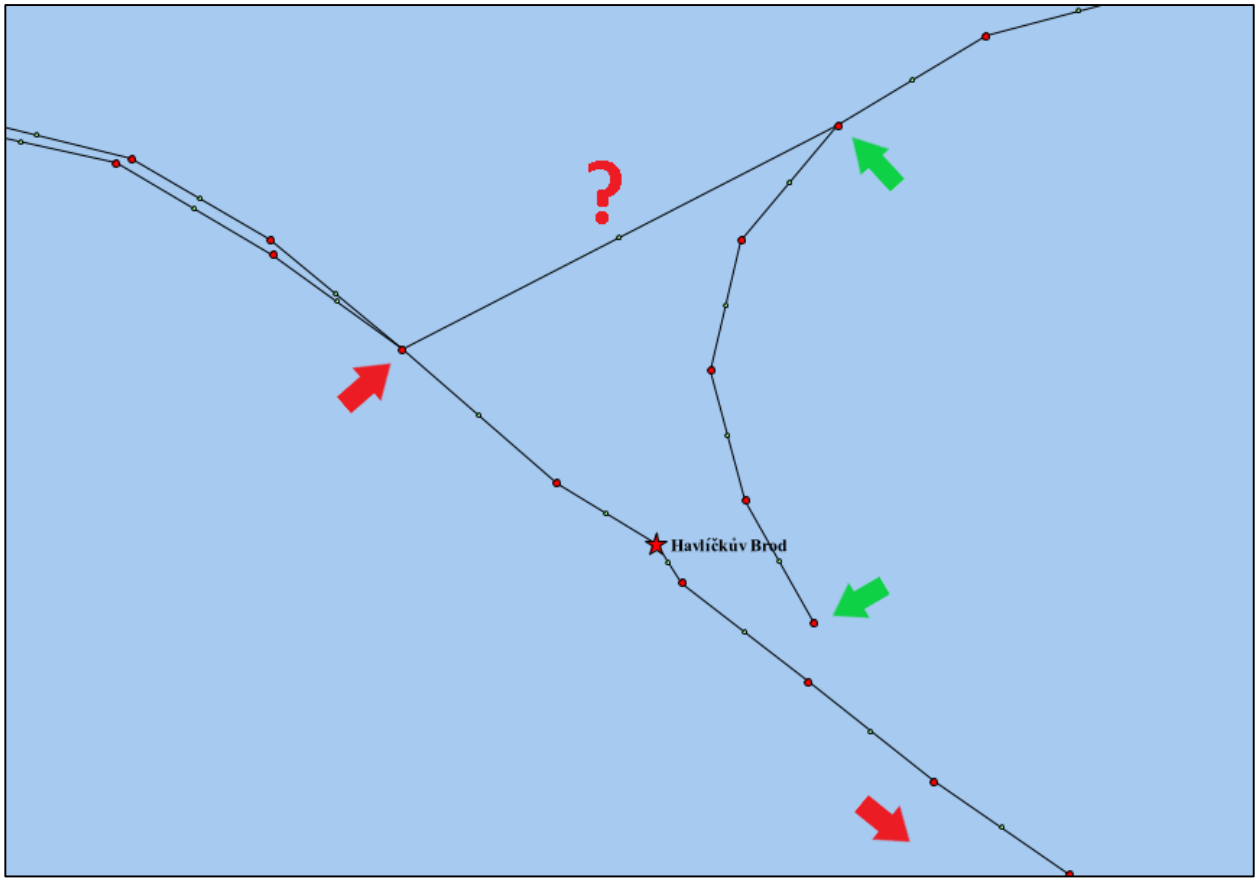
Celá situace propojování TUDU byla zjednodušena bázovými daty, která určovala pořadí traťových definičních úseků v rámci daného nadúseku. Proto každý algoritmus vychází z postupného propojování úseků podle jejich pořadí. Pokud by neexistoval žádný záznam o pořadí úseků, tak by byla celá situace komplikovanější, neboť by se pořadí muselo určit na základě GPS souřadnic a kilometrů, což by však nemuselo vždy vést ke správnému výsledku.

První algoritmus, který ve většině případech přinášel správné výsledky, byl založený na propojování úseků podle krajních hektometrovníků a stanic. Jednalo se o jednoduchý postup, kdy se předchozí úsek propojil s následujícím v takovém místě, kde okrajové hektometrovníky byly od sebe co nejméně vzdáleny. Pro určení vzdálenosti bylo potřeba pracovat s GPS souřadnicemi, neboť se nebylo možné spoléhat na hodnoty hektometrovníků, jak bylo uvedeno v předchozích kapitolách. Tato nespolehlivost hektometrovníků zapříčinila vzniku techniky, kdy bylo zapotřebí zkontrolovat všechny možné kombinace napojení krajních uzlů a určení těch, jejichž vzdálenost byla nejmenší ze všech. Na obrázku 27 představují červené linky ty uzly, které byly shledány jako nevhodné k propojení a zelená linka ty, které naopak propojeny byly:




**Obrázek 27 - Napojení TUDU mezi sebou [autor]**

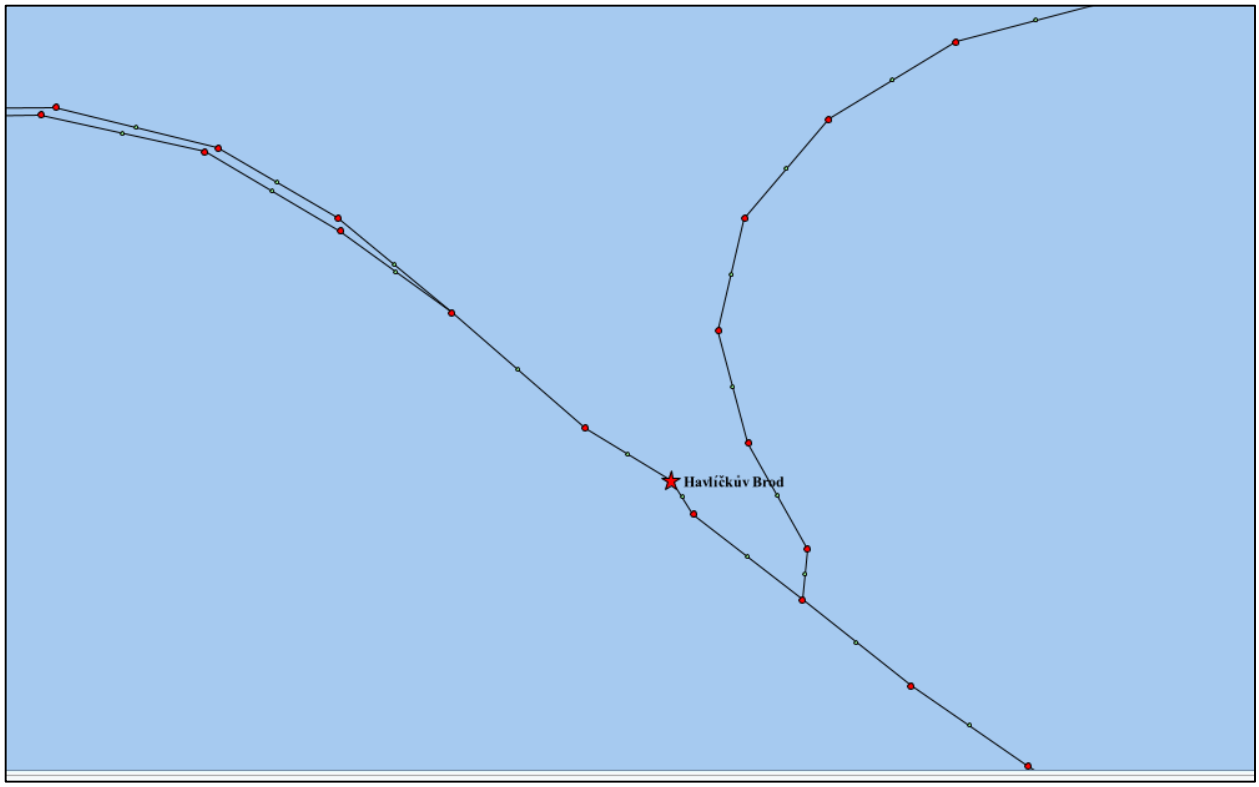
Toto se však ukázalo býti nevhodným řešením, a to z toho důvodu, že ne všechny traťové definiční úseky na sebe takto ideálně navazují ve svých počátečních či koncových uzlech. Tato skutečnost vedla ke vzniku několika nepřesně napojených úseků, což zkazilo výsledný dojem při pohledu na model železniční sítě a u těchto případů vedla pozorovatele k otázce, je-li vůbec možné takovou strukturu někde ve skutečném světě zpozorovat. Tento problém nesprávného napojení lze znázornit na obrázku 28, kde se aplikoval algoritmus na propojení dvou úseků oddělených šipkami (červená šipka vpravo znamená, že krajní bod úseku je mimo obrázek). Nastalo zde ke zmíněnému problému, kdy úsek označený zelenými šipkami má být napojen zhruba do středu úseku označeného červenými šipkami. Jelikož se ale pracuje pouze s koncovými uzly, tak je vytvořena nekorektní hrana (označena červeným otazníkem), která je nejkratší možnou mezi koncovými uzly. Takový útvar je však nereálný. Proto bylo vhodné algoritmus upravit tak, aby k těmto případům nedocházelo:




**Obrázek 28 - Špatné napojení TUDU [autor]**

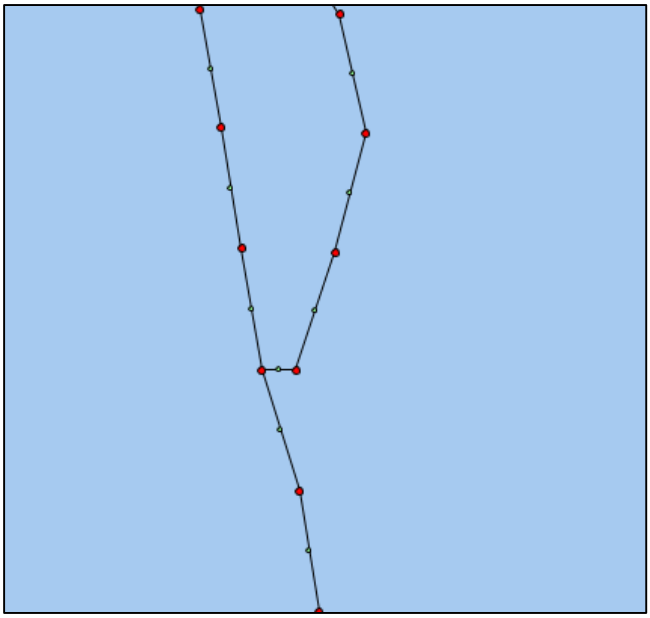
Kvůli problémům popsaným v předchozím odstavci se zvážilo použití algoritmu označovaného jako „Hrubá síla“. Tento postup spočíval ve vypočítání všech možných kombinací napojení jednotlivých uzlů označovaného jako „každý s každým“ (kartézský součin). To vedlo k nezávislosti vůči koncovým bodům a napojení vždy těch, které jsou k sobě nejblíže. Tento

algoritmus se tedy dokázal přizpůsobit situaci na obrázku 28, kdy se jeden úsek nenapojoval do středu dalšího úseku. Je-li zvážen problém z předchozího obrázku 28, tak výsledkem tohoto algoritmu je následující výstup na obrázku 29:



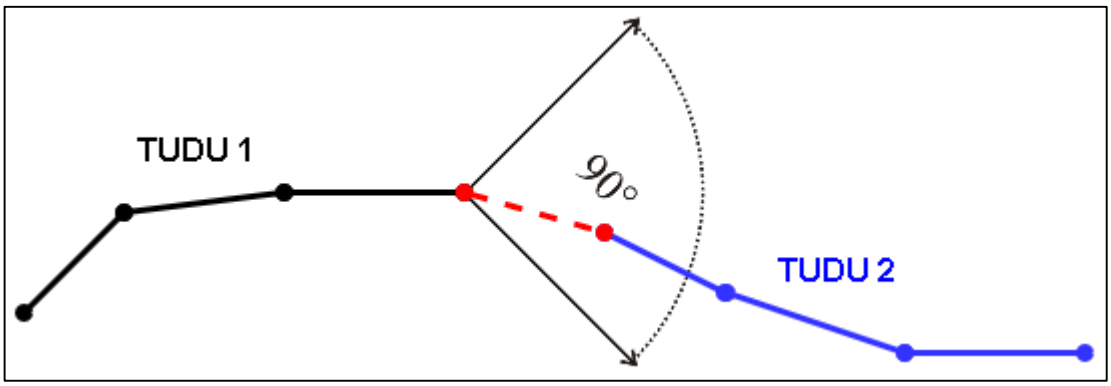

**Obrázek 29 - Kartézský součin TUDU [autor]**

Tento algoritmus je podle obrázku 29 vskutku lepší než předchozí, protože v tomto případě spojil vhodnějším způsobem traťové definiční úseky. Problémem však je, že ani tento postup nepřináší dokonalé výsledky celoplošně, neboť v některých situacích je napojení nejbližších bodů nežádoucí. To zahrnuje hlavně ty úseky, které jsou přibližně vedle sebe, anebo se různě kříží. Na obrázku 30 je znázorněn problém spojení dvou úseků, kde se algoritmicky správně vytvořila hrana mezi nejbližšími uzly, nicméně je tak ostrá vůči předchozím úsečkám, že ve skutečnosti nemůže existovat. Obrázek 30 tuto situaci znázorňuje:



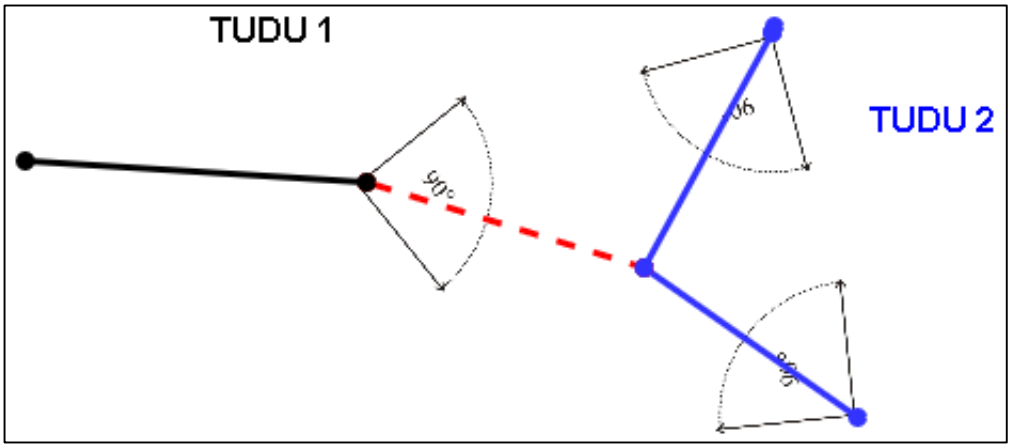
**Obrázek 30 - Ostré napojení TUDU [autor]**

Proto bylo zapotřebí modifikovat stávající algoritmus „Hrubá síla“ tak, aby k tomuto problému nedocházelo. Z prostých dat nelze získat žádné užitečné informace, které by vedly k řešení této situace. Proto na scénu přicházela poctivá matematika, která pomocí určení úhlu mezi vytvořenou a sousední hranou určila, jestli lze danou hranu přijmout či ne. Modifikace se netýkala pouze kontroly úhlu. Jelikož při zamítnutí ostré hrany je nutné vyhledat hranu další, která by byla kandidátem pro další kontrolu úhlu, tak je nutné tyto hrany uchovávat v nějaké vhodné kolekci. V nemodifikované verzi tohoto algoritmu kolekce potřeba není, protože se vždy uchovává pouze a jen ta nejkratší hrana. Hrany jsou v dané kolekci samozřejmě uspořádané podle délky od nejkratší po nejdelší a postupně se prochází, dokud není nalezena ta, která splňuje podmínku vhodného úhlu. Pokud se žádná taková hrana nenalezne, což by se teoreticky nemělo stát, tak se na spojení úseků aplikuje nemodifikovaná verze tohoto algoritmu, která nezahrnuje kontrolu úhlů. Jako vhodné úhlové rozmezí byla zvolena hodnota 90 stupňů, která se aplikuje na hranu následujícím způsobem:

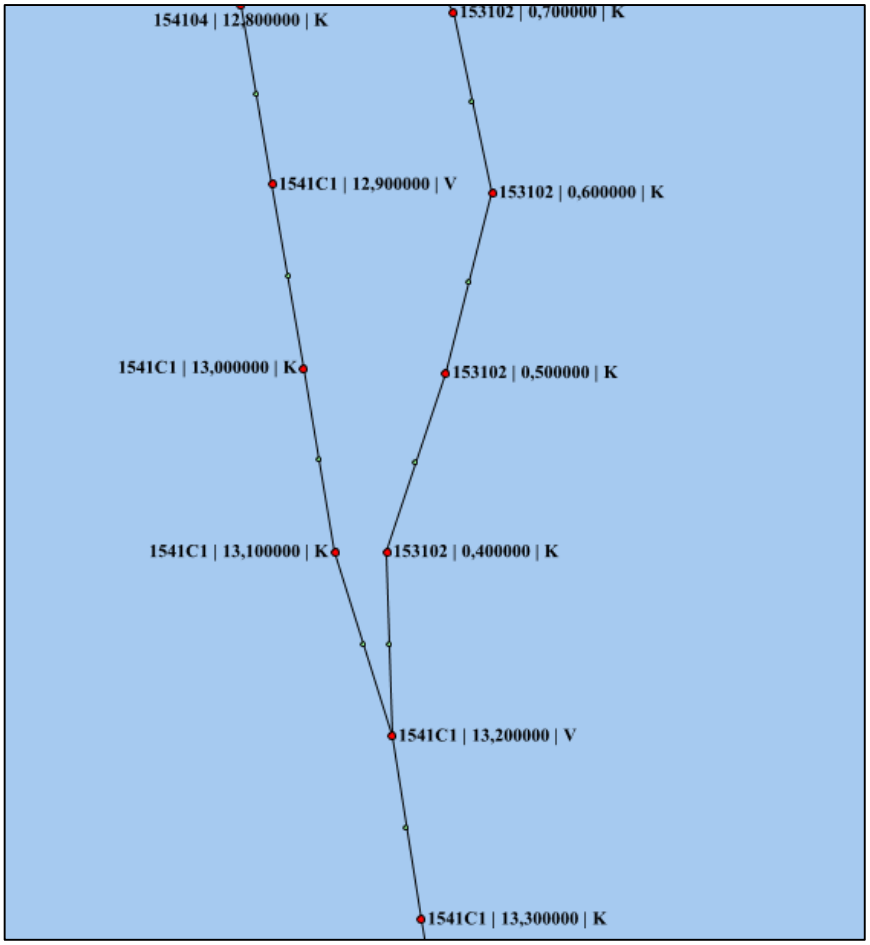



**Obrázek 31 - Vylepšený hrubý algoritmus [autor]**

K určení úhlu jsou však zapotřebí dva vektory, a proto dalším problémem bylo určení, ze kterých hran se mají tyto vektory vytvořit. Prvním vektorem je jednoznačně vytvořená hrana mezi traťovými definičními úseky. Další vektory musely vzniknout jednoznačně z krajních bodů zkoumané vytvořené hrany, kdy se tento krajní bod spojil s předchozím a vznikl vektor další. Problémem však bylo, mezi kterými vektory porovnávat úhel. Jelikož chybí informace o tom, který uzel sousedí se kterým, tak musel být vyhotoven pomocný seznam pro každý ze dvou definičních úseků. Tento seznam byl následně seřazen podle hodnot kilometrů, aby bylo možné vůbec získat jednotlivé sousedy pro zhotovení úsečky. Celá situace mohla vypadat následovně na obrázku 32, kde jsou pro porovnání k dispozici hned tři vektory. Prvním vektorem k porovnávání je jednoznačně červená úsečka představující možnou hranu mezi traťovými definičními úseky. Druhým vektorem mohou být celkem tři úsečky (černá a dvě modré), což vedlo k otázce, jaké úsečky zvolit. Po prozkoumání problému bylo však zvoleno řešení, které spočívá v porovnání všech úhlů a určení, které vektory vyhovují. Pokud počet vyhovující vektorů je větší nebo roven počtu nevyhovujících, tak potom lze tuto hranu přijmout. To by platilo i na obrázku 32 znázorněném níže, kdy jediný nevyhovující úhel vznikne z vektoru modrého horního a červeného. Tento způsob není zcela dokonalý, protože může nastat situace, kdy žádné napojení nebude vyhovovat podmínce úhlů. Proto při nenalezení žádného možného spojení bylo rozhodnuto o použití nemodifikovaného algoritmu „Hrubé síly“ jako záchranného způsobu napojení úseků. Díky tomu může v modelu železniční sítě vzniknout několik nesprávně napojených traťových definičních úseků. Tento modifikovaný algoritmus však záplatuje většinu takovýchto situací a zajistí vhodné napojení u majority případů jako například u toho, který je znázorněn na obrázku 33:




**Obrázek 32 - Hledání vhodných vektorů [autor]**




**Obrázek 33 - Opravené napojení úseků včetně popisu hektometrovníků [autor]**

K porovnání jednotlivých algoritmů slouží tabulka 11, kde špatná hrana znamená hranu, jejíž délka je podle GPS souřadnic mimo rozmezí 0–150 metrů:

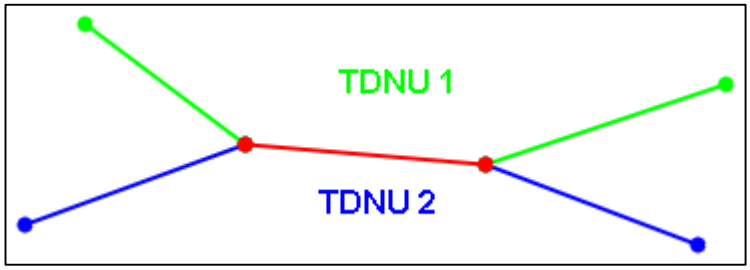
**Tabulka 11 - Statistika algoritmů napojování TUDU**

| Algoritmus | Počet špatných hran | Počet hran celkem | Špatné hrany % |
|---|---|---|---|
| Napojení krajních bodů | 136 | 93896 | 0,14 % |
| Hrubá síla | 61 | 93896 | 0,06 % |
| Hrubá síla s kontrolou úhlů | 61 | 93896 | 0,06 % |

### 3.3.4 Napojení TDNU mezi sebou

Cílem předchozího kroku v předešlé podkapitole bylo sjednotit traťové definiční úseky do nadúseku. V předchozí práci absolventa Radka Prokeše bylo použito mechanismu s pomocí takzvaných super tras. Super trasa definovala právě napojení jednotlivých TDNU mezi sebou. V rámci této práce však bylo zvoleno řešení, které se super trasám zcela vyhýbá, a proto jsou i ignorovaný pomocné datové soubory s nimi spojené. Pro eliminaci super tras bylo použito

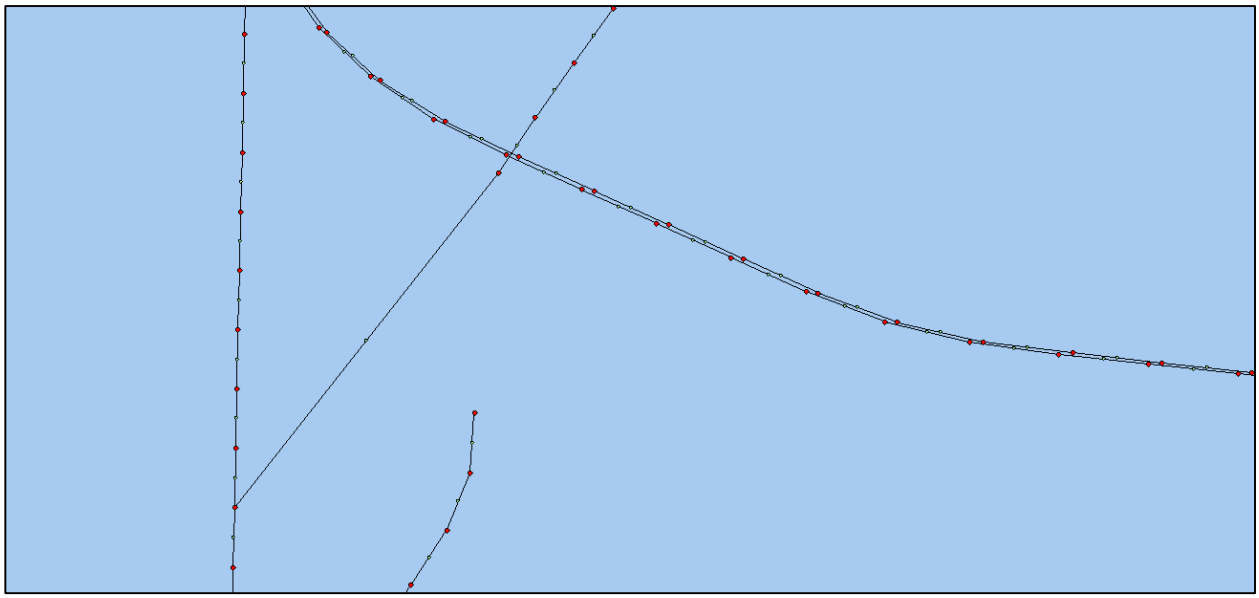
možnosti hash kódů jednotlivých hektometrovníků a stanic, kdy tento hash kód představuje unikátní hodnotu tvořenou GPS souřadnicemi. K použití hash kódu bylo přistoupeno potom, co byla odhalena skutečnost, že se nadúseky mezi sebou kříží. Zjednodušeně řečeno, pokud má nadúsek nějaký traťový definiční úsek v sobě obsažený, pak tento definiční úsek se nachází v minimálně jednom dalším nadúseku. Na obrázku 34 jsou znázorněny dva nadúseky (zelený a modrý) a červený traťový definiční úsek, kde se tyto nadúseky kříží. Proto byl algoritmus celkem jednoduchý, neboť provedl mezi těmito nadúseky pouze operaci sjednocení, aby bylo zamezeno duplicitním úsekům v místě křížení. Operaci sjednocení nebylo obtížné vyřešit, protože o vše se staralo použití zmíněného hash kódu, který ve správné datové struktuře zamezil vytváření duplicitních záznamů. Obrázek znázorňující napojení vypadá následovně:




**Obrázek 34 - Napojení TDNU mezi sebou [autor]**

### 3.3.5 Problém s některými TUDU

Vlivem nedostatečných bázových dat vznikl problém s některými traťovými definičními úseky. Jak bylo zmíněno v předchozích kapitolách, tak správnost vygenerování modelu železniční sítě závisí na tom, aby byla dodržena hierarchie záznamů a nevznikly žádné osamocené listy. Nesmí tedy nastat situace, kdy některý traťový definiční úsek nemá přiřazený nadúsek. Tento stav bohužel v této aplikaci nastává a jeho důsledkem je vznik děr (nespojitá kolej) v modelu. Právě kvůli těmto dírám nelze v některých případech správně napojit definiční úseky mezi sebou a vzniká situace znázorněná na obrázku 35, kde lze zpozorovat zmíněnou díru v modelu železniční sítě, vlivem které vznikla velmi dlouhá hrana, která je ve výsledném zobrazení nežádoucí. Obrázek 35 znázorňuje zmíněnou situaci:



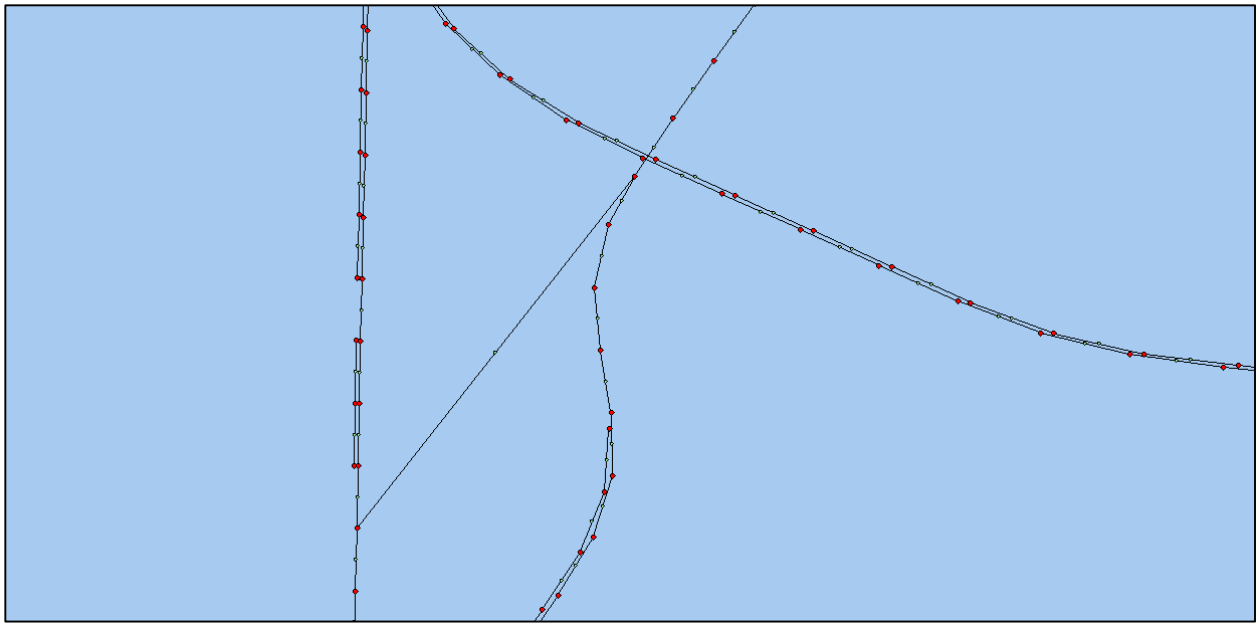
**Obrázek 35 - Vznik děr při absenci definičního úseku pro daný nadúsek [autor]**

Problémem je, že daný definiční úsek nespadá do žádného nadúseku, a tudíž nemůže být podroben standardnímu procesu sestavování grafu. Ten spoléhá právě na sestavování modelu železniční sítě při splnění hierarchické struktury. Ignorování těchto definičních úseků je však hloupost, protože budou v modelu chybět informace a zároveň na vzniklé díry není radost pohledět. Proto byl vymyšlen mechanismus zakomponování těchto úseků do výsledného grafu. Výhodou tohoto řešení je jednoznačně zajištění informační úplnosti, neboť v grafu nechybí ani jeden hektometrovník. Nevýhodou je fakt, že jelikož není možné tyto úseky zahrnout do standardního procesu sestavení, tak tento proces opravy musí nastat až po vygenerování kompletního grafu. Zároveň algoritmus nemusí vždy ideálně fungovat, protože mohou být samozřejmě výjimky.

Jako řešení tohoto problému byly použity opět možnosti hash kódu. Kroky algoritmu lze popsat následujícími kroky:

- sestavení grafu z problémového definičního úseku,
- získání počátečního a koncového uzlu úseku podle největší a nejmenší hodnoty kilometru,
- prohledání celé datové struktury ADT Graf a získání nejbližších hektometrovníků pro oba krajní uzly nalezené v předchozím kroku,
- vložení problémového nadúseku do modelu železniční sítě a vytvoření hran mezi nalezenými hektometrovníky z předchozích dvou kroků – využití hash kódů
- a odstranění dlouhých hran.

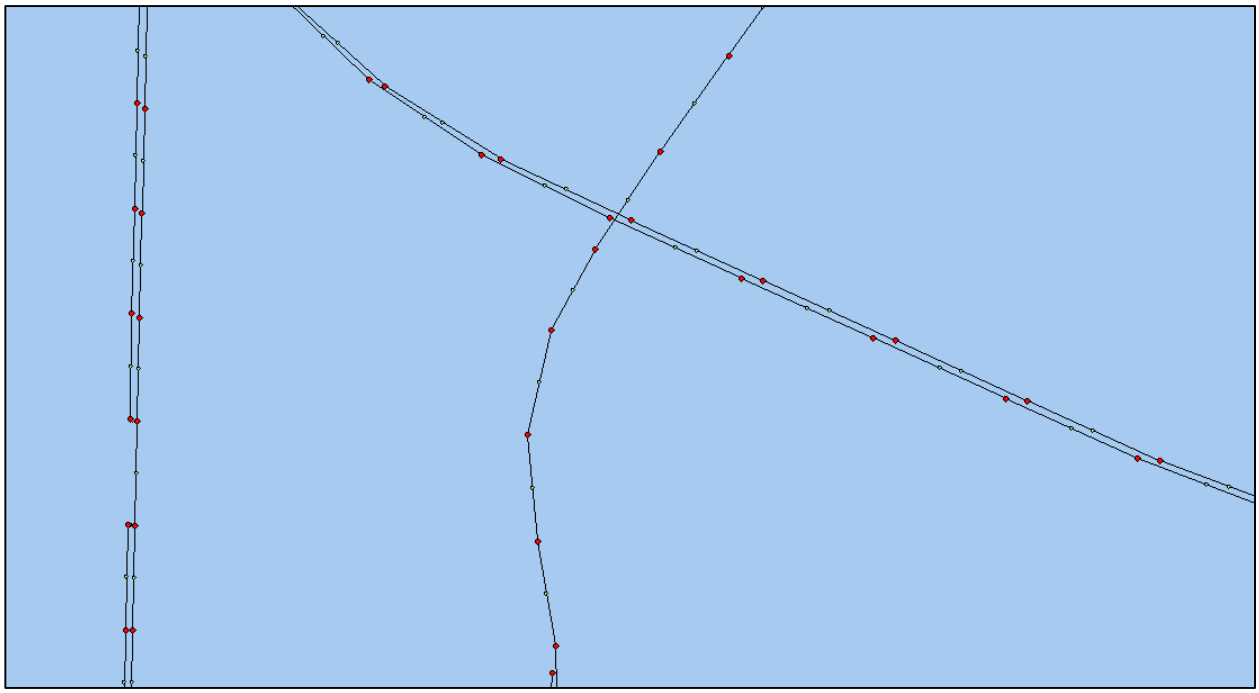
Pokud by nebyl uvažován poslední krok týkající se odstranění dlouhých hran, pak by výstup vypadal následovně:



**Obrázek 36 - Oprava díry v nadúseku [autor]**

V kroku předposledním je zmíněno využití hash kódů. To umožňuje, jak bylo zmíněno v předchozích odstavcích, vložení dat do grafu a zamezení vzniku duplicit.

V posledním kroku se odstraňují velmi dlouhé hrany, jejichž délka je větší než 150 metrů. Odstraňování hran zde probíhá z toho důvodu, že při standardním generování modelu železniční sítě vznikly velmi dlouhé hrany vlivem děr zapříčiněných absencí definičních úseků u některých nadúseků. Proto jsou uzly, ve kterých začíná a končí díra, vhodnými kandidáty pro překontrolování hran a odebrání těch, které nejsou korektní. Kontrola probíhá tak, že se pro všechny čtyři nalezené body (krajní body díry a krajní body definičního úseku bez nadúseku) zkontrolují hrany na existenci těch, které jsou příliš dlouhé. Pokud je některá taková hrana nelezena, pak je zařazena do kolekce pro odstranění a na konci jsou všechny hrany z této kolekce odstraněny ze seznamu hran příslušného vrcholu. Jelikož je graf vrcholově orientovaný, tak díky absenci seznamu hran není nutné jakýkoli takovýto seznam aktualizovat. K odstranění zcela postačí aktualizace seznamu hran u obou incidentních vrcholů, což je operace s konstantní asymptotickou složitostí. To znamená, že se jedná o velmi efektivní opravu, neboť se k datům přistupuje opět pomocí hash kódu. Výsledná oprava děr s odstraněním příliš dlouhých hran vypadá následovně na obrázku 37:



**Obrázek 37 - Výsledné opravení díry v nadúseku [autor]**

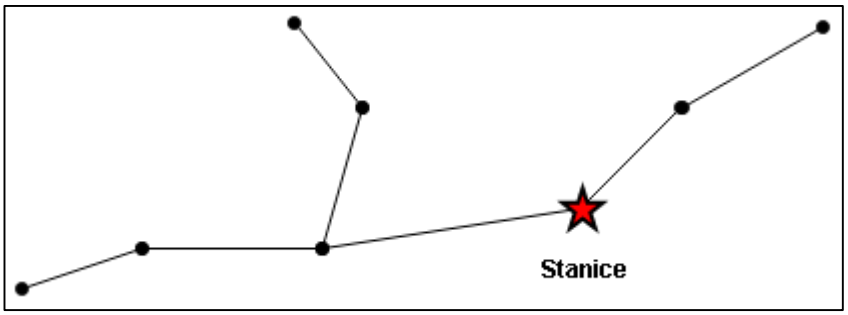
Co se týče zhodnocení celkové správnosti algoritmu sestavení modelu železniční sítě, tak samozřejmě existují výjimky, které nebylo možné ošetřit z důvodu absence či chybovosti bázových dat, které lze algoritmicky velmi těžce nahradit. Nicméně technikami uvedenými výše bylo docíleno co možná nejnižšího dopadu nekorektnosti bázových dat na finální model železniční sítě.

### 3.3.6 Sestavení zjednodušeného modelu

Z důvodu velkého množství dat ve výsledném modelu (přibližně 95 000 bodů a 360 000 hran) bylo výpočetně a časově velmi náročné tato data načíst a zpracovat službou MapViewer k zobrazení požadovaného výstupu. Proto bylo zapotřebí celou strukturu grafu nějakým způsobem datově zjednodušit. Základní myšlenka představovala zobrazení zjednodušené kostry modelu železniční sítě při malém přiblížení, protože v oddáleném pohledu na mapu nejsou detaily okem viditelné, což je možné spatřit na znázornění modelu v kapitole 3.5.1. Princip je možné přirovnat při vykreslování objektů v počítačových hrách, kdy objekty vzdálenější než určitá mezní hodnota jsou převedeny na objekt s daleko hrubší trojúhelníkovou sítí, což ho činí výpočetně nenáročným. Proto bylo rozhodnuto vytvořit dva grafy, kde první představuje graf na micro vrstvě obsahující veškerá data a druhý, nacházející se na vrstvě macro, je zjednodušením micro grafu.

Ještě před samotným psaním algoritmu se vyskytla otázka, jaká pravidla platí pro zjednodušení modelu železniční sítě. Vstupem do algoritmu zjednodušení struktury mohly být

údaje znázorněné na obrázku 30, kde si lze povšimnout jednoho uzlu s více jak dvěma hranami (dále označovány jako křižovatky) a jedné stanice:

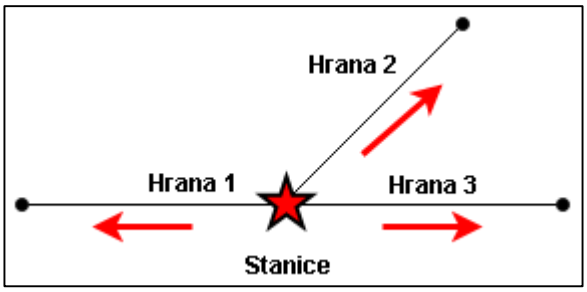



**Obrázek 38 - Pohled na část modelu železniční sítě [autor]**

Z tohoto obrázku vyplývají jasná pravidla při zjednodušování modelu železniční sítě. První pravidlo představuje nutnost zachovat všechny křižovatky a stanice, neboť tyto údaje nesmí nikdy chybět. Druhé pravidlo říká, že veškeré hektometrovníky, které nejsou ani stanicí, ani křižovatkou, lze podrobit zjednodušení. Na základě těchto dvou pravidel byl sestaven algoritmus popsaný v následujícím odstavci. Tato dvě pravidla by ovšem nestačila, protože by model z obrázku 38 končil vždy pouze v křižovatkách, anebo stanicích. Proto bylo nutné přidat pravidlo třetí, které říká, že klíčovým bodem, jenž je nutné rovněž zachovat, je konec tratě (uzel s pouze jednou hranou).

Nejtěžším úkolem bylo určit algoritmus, který převede plnohodnotný model železniční sítě ve formě datové struktury ADT Graf na zjednodušenou formu. Jelikož procházení samotné kolekce vrcholů není vhodným způsobem procházení modelu železniční sítě, tak bylo zapotřebí zvolit odlišný způsob procházení. Jako vhodným způsobem se ukázalo býti zvolení náhodného vrcholu grafu, který je označen jako počáteční vrchol pro procházení. Tento vrchol nesmí ale být jen tak obyčejný, ale musí to být právě křižovatka, stanice, anebo konec trati. Důvodem omezení počátečního vrcholu na tento typ hektometrovníku je fakt, že zjednodušený model se musí budovat vždy od bodu, který je součástí jak zjednodušeného, tak i kompletního grafu. Jinak řečeno se hrany budují mezi křižovatkami, stanicemi a konci tratě, a proto i počáteční vrchol musí být některým z těchto typů vrcholů. Samotný počáteční vrchol by ale nestačil, protože z něj nelze určit

informaci, kterým směrem pokračovat, proto se s vrcholem uvádí i hrana vyjadřující směr procházení. V prvopočátku algoritmu je do fronty vložen tento vrchol se všemi možnými směry:

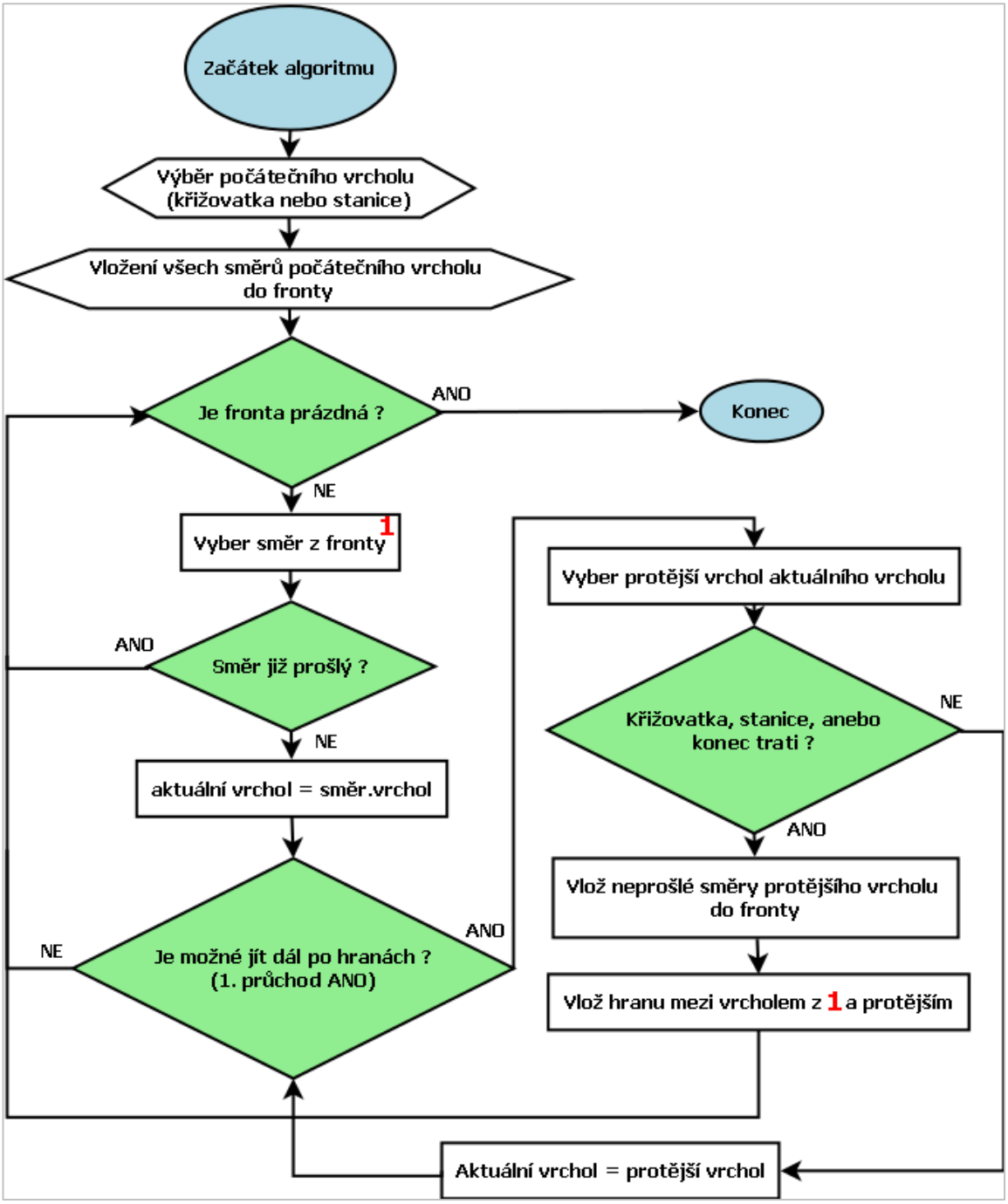



**Obrázek 39 - Princip procházení při zjednodušování sítě [autor]**

**Tabulka 12 - Inicializace fronty podle obrázku 39**

| Vrchol | Hrana |
|---|---|
| Stanice | Hrana 1 |
| Stanice | Hrana 2 |
| Stanice | Hrana 3 |

Kroky algoritmu jsou znázorněny na následujícím diagramu:




**Obrázek 40 - Diagram zjednodušení železniční sítě [autor]**

Princip algoritmu je tedy prostý. Od počátečního vrcholu se prochází jeden směr, dokud se nenarazí na křižovatku, stanici, anebo konec trati. Pokud se na takový vrchol narazí, pak je mezi vybraným z fronty a aktuálním protějším vytvořena hrana, která zredukuje všechny prošlé hrany mezi těmito vrcholy do jedné. Z protějšího vrcholu se vyberou všechny hrany, které dosud nebyly

prošlé a vloží se do fronty. Algoritmus pokračuje, dokud jsou ve frontě některé záznamy dostupné. Výsledná redukce modelu železniční sítě z obrázku 38 je znázorněna na obrázku 41:


**Obrázek 41 - Výsledek zjednodušení železniční sítě [autor]**

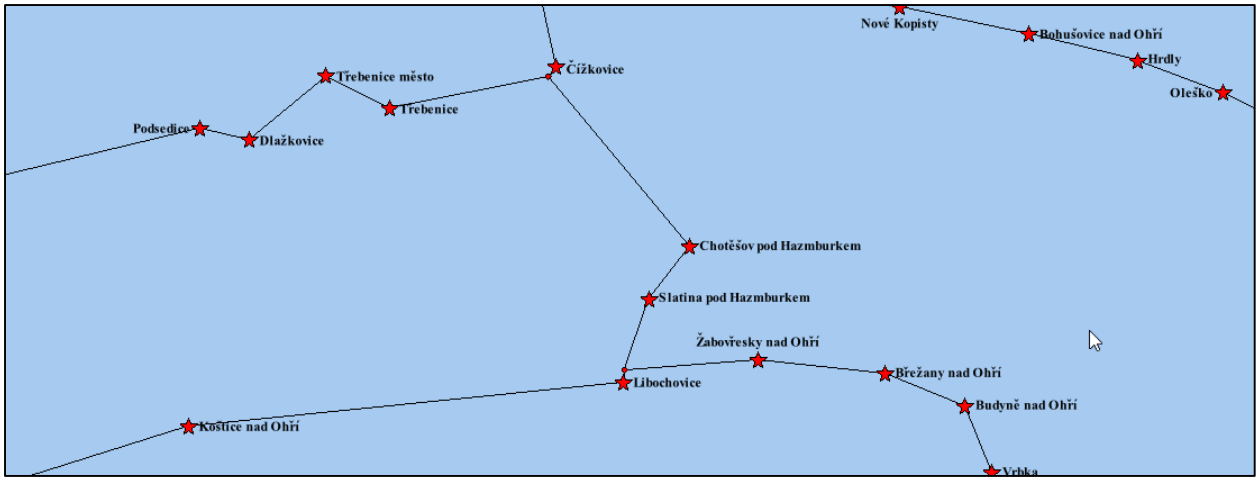
Jak si lze povšimnout, tak z celkového počtu osmi vrcholů a sedmi hran na obrázku 38 byl model zredukován na pět vrcholů a čtyři hrany. V kompletním modelu železniční síti jsou ale čísla zajímavější:

**Tabulka 13 - Tabulka uspořených dat zjednodušeného modelu železniční sítě**

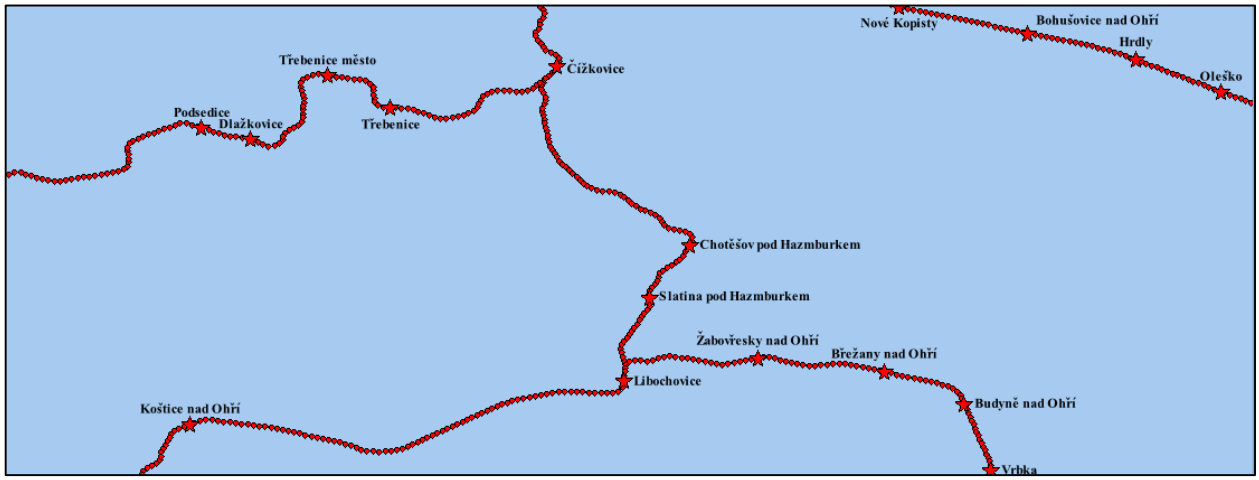
| | Vrcholy | Hrany |
|---|---|---|
| **Kompletní model železniční sítě** | 95374 | 95526 |
| **Zredukovaná model železniční sítě** | 3252 | 3608 |
| **Procento uspořených dat** | 96,6 % | 96,22 % |

Redukcí takového množství dat vznikla naděje na zobrazení větší rozlohy modelu železniční sítě s přijatelnou časovou náročností. Při testování zabralo zobrazení celého modelu 4820 milisekund (méně než pět sekund), zatímco při zobrazení kompletního modelu bez zjednodušení zabralo několik jednotek minut, což by uživatele, ale i programátora, značně iritovalo.

Na obrázcích 42 a 43 si lze všimnou porovnání dvou shodných úseků modelu železniční sítě, kde první obrázek představuje zjednodušený model vhodný pro zobrazení z větší výšky a na druhém detailní model pro prozkoumání jednotlivých hektometrovníků:




**Obrázek 42 - Zjednodušená železniční síť [autor]**



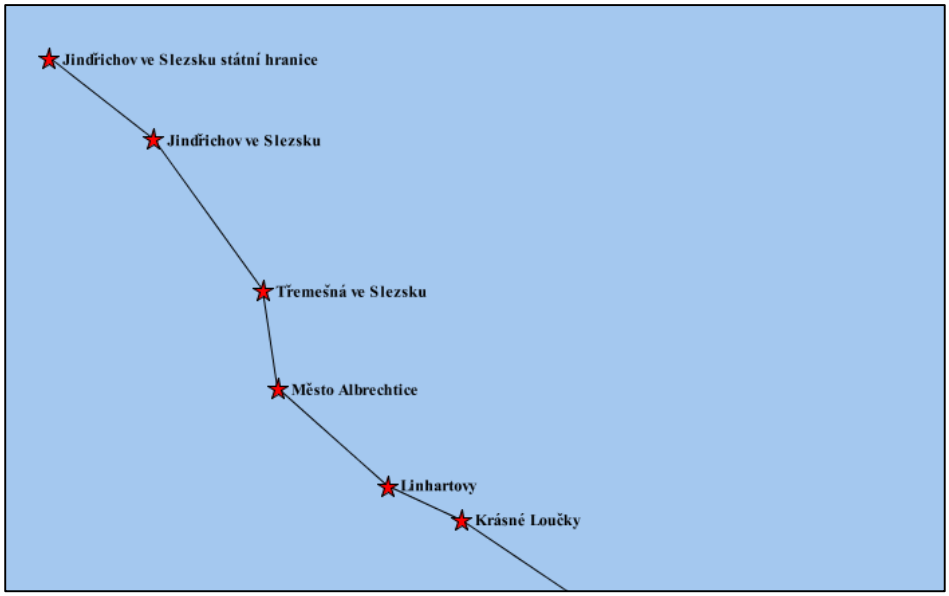

**Obrázek 43 - Detailní železniční síť [autor]**

Výsledkem byla micro a macro vrstva, které dohromady tvořily dvouvrstvý model železniční sítě. Úkolem micro vrstvy bylo uchovat co největší detaily modelu železniční sítě. Oproti tomu vrstva macro měla za úkol uchovat data zjednodušená.

### 3.3.7 Osamocené nadúseky

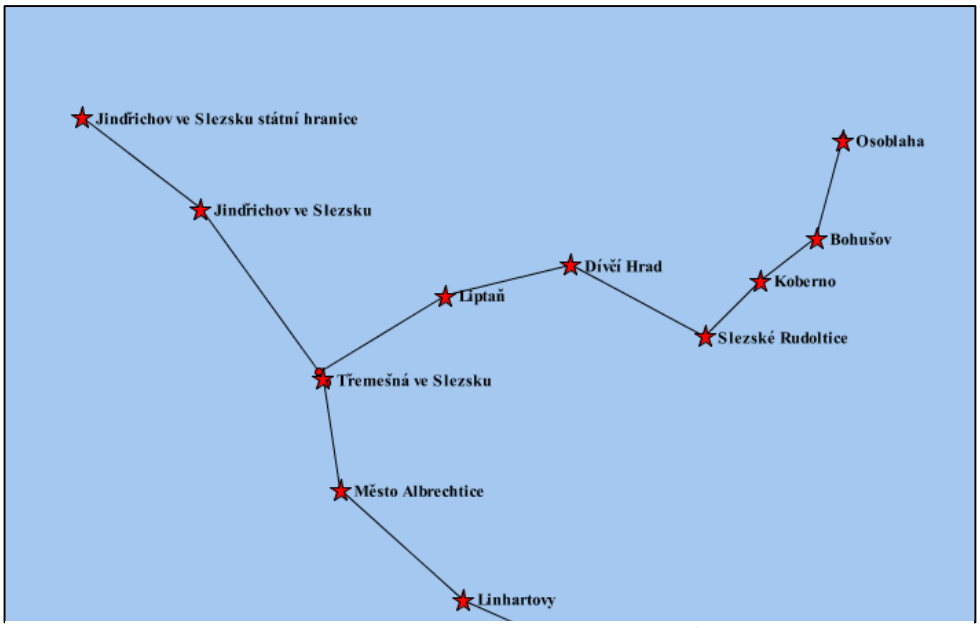
V pozdější fázi diplomové práce bylo zjištěno, že existuje několik nadúseků (okolo 8), u kterých neexistoval průnik s nějakým dalším nadúsekem. Vznikly takzvané osamocené nadúseky, které zapříčinily nepřesnost modelu železniční sítě. Tato chyba se projevovala hlavně ve zjednodušeném modelu železniční sítě, kde tyto nadúseky zcela chyběly, neboť se k nim nebylo

možné přes nějakou hranu dostat. Celý problém je zobrazen na obrázku 44, kde pravá větev železnice zcela chybí:




**Obrázek 44 - Chybějící nadúseky v železniční síti [autor]**

Postup pro nápravu tohoto problému byl zvolen takový, který zahrnuje identifikaci osamocených nadúseků a spojení s modelem železniční sítě vhodně vygenerovanou hranou. Jelikož je model rozsáhlý, tak by bylo velmi výpočetně náročné vyhledávat nejkratší hranu mezi všemi možnými kombinacemi. Proto bylo rozhodnuto vedle samotné kolekce vrcholů evidovat datovou strukturu R-strom. Tato dvourozměrná datová struktura umožnila vyhledat vrcholy ve stanoveném obdélníkovém rozsahu a nalézt pouze ty vrcholy, které se nacházely v nejbližším okolí. Tyto nalezené vrcholy pak byly podrobeny sestavování hran spolu s vrcholy osamoceného nadúseku a ta hrana, která byla nejvhodnější (stejná pravidla jako na obrázku 32), se mezi těmito nadúseky vytvořila. Tím došlo ke spojení, které zapříčinilo zamezení absence nadúseku v grafu modelu železniční sítě:




**Obrázek 45 - Oprava osamocených nadúseků [autor]**

## 3.4 Exportování dat

Exportování dat do databáze je zajisté klíčovým krokem a zároveň předpokladem pro využití služeb MapBuilder a MapViewer. Aby bylo možné použít tabulku těmito službami, tak musí obsahovat minimálně jeden sloupec pro prostorová data typu **`SDO_GEOMETRY`**.

Tato práce se však drží standardu zvaného Oracle Network Data Model (dále jen ONDM[16]), který předepisuje strukturu tabulek pro pozdější analýzu prostorových dat bez nutnosti použití aplikace třetí strany. Tento model poskytuje následující možnosti:

- uložení prostorových dat do databáze,
- databázové procedury a funkce pro udržování a analýzu prostorových dat
- a analýza prostorových dat přes aplikační rozhraní Java.

### 3.4.1 Struktura tabulek v aplikaci

Samotné tabulky mohou mít libovolný název, ale musí splňovat konkrétní strukturu v závislosti na datech, která se v ní uchovávají. Tabulky jsou dostupné pro uzly, spojení, cesty a spojení cest. V této aplikaci se však pracuje pouze se strukturou pro uzly a spojení, neboť cesty jsou pro účel této práce irelevantní. Není-li předepsaná struktura tabulek dodržena, pak není možné výše popsané funkce využívat. Nic nebrání předepsanou strukturu obohatit o vlastní objekty, ať už jde o sloupce, index apod. Vlastní sloupce jsou však pro ONDM neviditelné, a tak slouží pouze

[16] Oracle Network Data Model – Systém ukládání prostorových dat v databázi Oracle

pro vlastní interní účely. V této práci je každá vrstva (0 – micro; 1 - macro) vyjádřena dvojící tabulek **NODES** a **LINKS**.

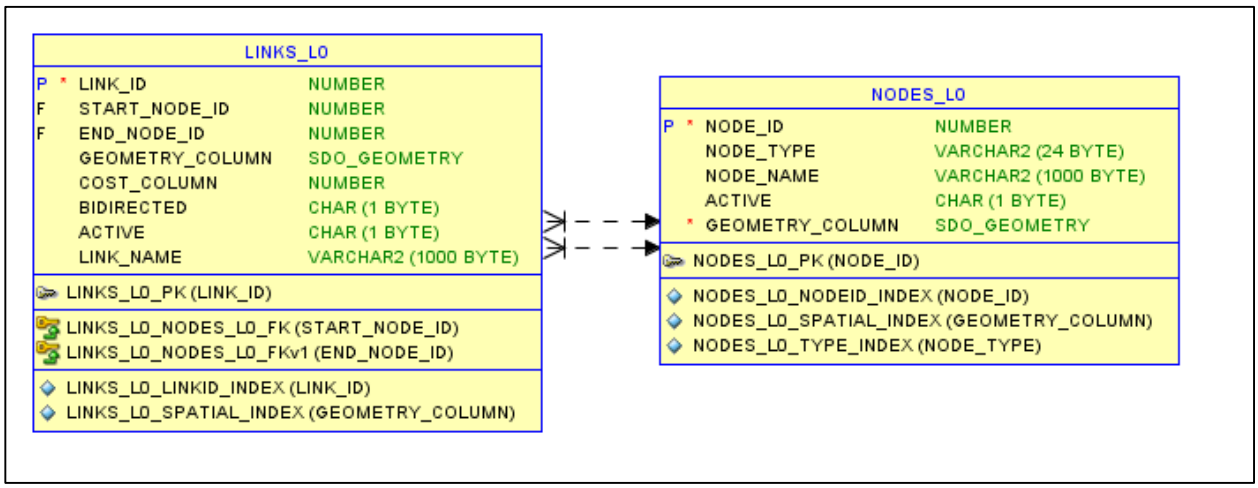
Pro ukládání hektometrovníků a stanic je využívána tabulka **NODES**. ONDM předepisuje pouze jediný atribut, který nesmí v tabulce chybět a je jím primární klíč NODE_ID. Jedná se o jedinečný číselný identifikátor uzlu v prostorových datech. Unikátnost identifikátoru je zajištěna již na úrovní aplikace, kdy se každému uzlu při vytvoření nastaví tento údaj pomocí statické proměnné. Statická proměnná ošetřuje i situaci, kdy dochází ke spojení dvou grafů mezi sebou, neboť při vytvoření nového grafu nedochází k vynulování této proměnné, ale pokračování v posledním údaji. ONDM dále nabízí doplňkové atributy, které slouží pro další operace. Mezi nepovinné atributy využívané v této práci patří:

- GEOMETRY_COLUMN – Uchování prostorových dat **SDO_GEOMETRY**.
- ACTIVE – Aktivace uzlu (neaktivní uzel není použit při analýze prostorových dat).
- NODE_NAME – Textové údaje o uzlu.
- NODE_TYPE – Údaj o typu uzlu, v této aplikaci pro rozlišení hektometrovníků a stanic.

K ukládání hran slouží tabulka s názvem **LINKS**, jež vyžaduje hned tři povinné atributy. Jedná se o jedinečný identifikátor hrany LINK_ID a dva cizí klíče START_NODE_ID a END_NODE_ID odkazující na tabulku **NODES**, které představují informaci, mezi kterými vrcholy se daná hrana nachází. Dalšími nepovinnými atributy, které jsou využívány v této práci, jsou následující:

- GEOMETRY_COLUMN – Uchování prostorových dat **SDO_GEOMETRY**.
- COST_COLUMN – Ohodnocení hrany (v této práci GPS vzdálenost mezi incidentními vrcholy).
- BIDIRECTED – Indikátor (ne)orientovaného grafu (v této práci vždy neorientovaný).
- ACTIVE – Aktivace hrany (neaktivní hrana není využita při analýze prostorových dat).
- LINK_NAME – Textové údaje o hraně.

Výsledný relační model používaný v této aplikaci je znázorněn níže na obrázku 46:




**Obrázek 46 - Relační databázový model použitých tabulek [autor]**

### 3.4.2 Odstraňování záznamů

Výchozí chování aplikace je takové, že před každým exportem dat do databáze je nejdříve provedeno smazání všech tabulek. Díky provázanost tabulek pomocí cizích klíčů je znemožněno provést operaci `TRUNCATE`. Není to nicméně zcela nedosažitelný úkol, ale vedl by jistě ke komplexnějšímu řešení v podobě zakazování a povolování integritních omezení, což by mohlo celý proces exportování dat do databáze více ztížit. První pokus k nalezení řešení vedl k otestování příkazu `DELETE FROM NODES/LINKS`. Jelikož se v tabulkách nachází desítky tisíc záznamů, tak celá operace vyústila ve velmi zdlouhavé mazání dat, neboť procházela jednotlivé řádky, což vedlo k časové náročnosti až několika jednotek minut. To by byl velmi nežádoucí jev při práci s aplikací. Jako nejmenší zlo bylo proto zvoleno řešení v podobě zakázání integritních omezení jako výchozí stav. Tímto se ztrácí kontrola existence cizích klíčů, nicméně za cenu velmi rychlého smazání rozsáhlé tabulky.

### 3.4.3 Indexy krokem k úspěchu

Pro ještě větší rychlost přístupu k datům z důvodu občasné filtrace záznamů dle konkrétních atributů bylo rozhodnuto o použití indexů. Indexy s sebou nesou však nevýhodu v podobě zdvojeného zápisu při DML operacích (vkládání, mazání a aktualizace údajů). V této práci tento handicap nebyl shledán jako nevýhoda, neboť se nejedná o systém s častými změnami údajů. Data jsou exportována do tabulek pouze jednou a dále nedochází k žádné změně.

Co se týče tabulky **`NODES`**, tak zde je přítomen jeden pomocný index. Ten se týká atributu `NODE_TYPE`. Důvodem vytvoření indexu pro tento atribut je právě existence údajů různého typu

v jedné tabulce. Pokud bude chtít uživatel přistoupit údajům konkrétního typu, tak bude příjemně překvapen rychlostí vrácení výsledků.

Příkladem může být příkaz pro vyhledání všech záznamů z tabulky **NODES**, jež jsou jen stanicemi (typ = 2= stanice). Jak je možné spatřit, tak se neprovedlo prohledání celé tabulky, což by vedlo k velmi výpočetně nákladné operaci, ale pomocí indexu se vyhledaly všechny záznamy odpovídající podmínce typu stanic a pomocí jedinečného interního databázového identifikátoru řádku `ROWID` se k jednotlivým záznamům přistoupilo konstantní asymptotickou složitostí. Tento index je nádhernou ukázkou datové struktury Range-tree z kapitoly 1.1.2. Příklad je následující:

```
SELECT * FROM NODES_L0 WHERE NODE_TYPE = 2;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |     1 |  4364 |     1   (0)| 00:00:09 |
|   1 |  TABLE ACCESS BY INDEX ROWID| NODES          |     1 |  4364 |     1   (0)| 00:00:09 |
|*  2 |   INDEX RANGE SCAN           | NODES_LAYER_IDX |     1 |       |     1   (0)| 00:00:09 |
-------------------------------------------------------------------------------------------
```

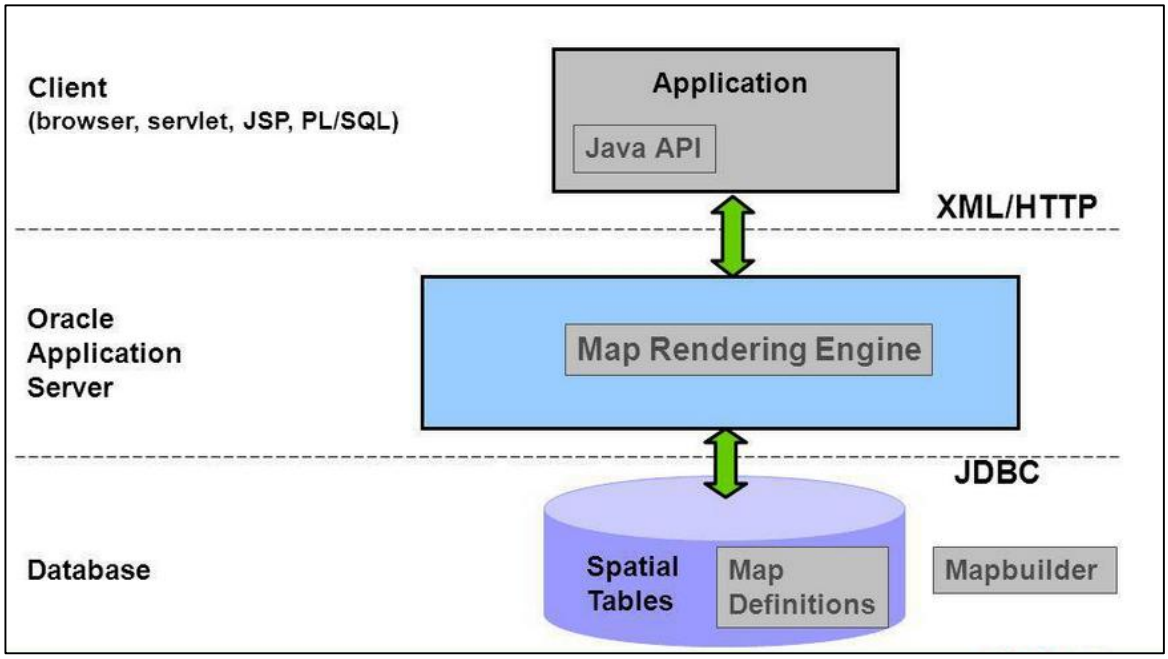
**Obrázek 47 - Exekuční plán příkazu nad tímto obrázkem [autor]**

### 3.4.4 Potvrzování transakcí

Z důvodu co největší konzistence dat bylo rozhodnuto o provádění manuálního potvrzování či zamítání změn v databázi až po dokončení celé operace exportování dat. Uživatel má tak možnost vidět, zdali proběhl export dat do databáze úspěšně a může rozhodovat o potvrzení či zamítnutí těchto změn. Problém transakčních databází je však ten, že transakce by neměla být příliš dlouhá, neboť transakční log se neustále zvětšuje, vlivem čehož se potvrzování či zamítnutí změn časově prodlužuje. Je však nežádoucí, aby se provádělo potvrzování změn postupně, neboť vznikne-li například chyba v půlce exportování dat, pak jsou data v databázi sice konzistentní, ale na roztříštěný model železniční sítě není pěkný pohled. Proto byl zvolen postup potvrzování změn až na úplném konci vkládání záznamů, avšak za cenu zdlouhavého čekání na potvrzení změn v řádech několika minut.

## 3.5 Nástroje MapBuilder a MapViewer

V předchozí diplomové práci byl vytvořen vlastní zobrazovací mechanismus pro grafickou interpretaci sestaveného grafu. Toto řešení je náročné na implementaci, ale velmi rychlé na zpracování, protože výpočty probíhají v operační paměti počítače. Proč ale implementovat něco, co již někdo jiný kvalitně implementoval, a tak v této práci bylo pro účely zobrazování sestaveného grafu využito nástrojů poskytovaných společností Oracle. Těmito nástroji jsou Oracle MapBuilder a MapViewer. Následující podkapitoly popíší tyto nástroje velmi stručně. Budou představeny pouze ty možnosti, které se týkají přímo aplikace, jež je předmětem vyhotovení praktické části této diplomové práce. Architektura je znázorněna na obrázku 48:




**Obrázek 48 - Architektura služby MapViewer [21]**

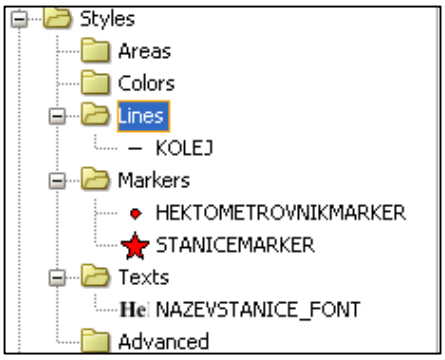
### 3.5.1 Oracle MapBuilder

Nástroj MapBuilder slouží k definici a stylování mapových podkladů (umožňuje vytvořit metadata). Veškeré potřebné kroky, které jsou zapotřebí, jsou následující:

- vytvoření připojení k databázi,
- definice stylů pro vykreslování,
- definice mapových vrstev
- a definice finální mapy.

První záležitostí je vytvoření připojení k databázi. Jedná se o velmi jednoduchý krok, kde je potřeba dodržet pouze přítomnost databáze Oracle.

Druhou fází je definice jednotlivých stylů pro vykreslování konkrétních geometrických útvarů. MapBuilder umožňuje vytvořit barvy, body, čáry, oblasti a textové styly. Je-li například uložen ve sloupci **SDO_GEOMETRY** tvar obdélníku, pak lze definovat styly pro jeho čtyři body a čáry, které tyto body spojují. Na obrázku 49 jsou znázorněny styly, které byly použity pro stylování mapových podkladů v této aplikaci. První je definice stylu pro znázornění koleje. Jedná se o prostou černou čárku, která má spojovat hektometrovníky a stanice. Další je definice stylů takzvaných markerů neboli bodů. Zde je použit styl pro vyznačení hektometrovníku v podobě červeného puntíku a stanice v podobě červené hvězdy. Posledním stylem je formát textu pro zobrazení názvu stanice. V nástroji MapBuilder vypadají styly následovně:



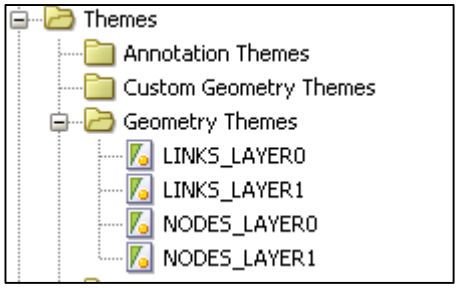

**Obrázek 49 - Definice stylů v nástroji MapBuilder [autor]**

Dalším krokem je vytvoření jednotlivých mapových podkladů (vrstev) pro každý sloupec **SDO_GEOMETRY**. Stručně řečeno je zapotřebí spojit každý sloupec zmíněného datového typu s jednotlivými styly, čímž vznikne jedna mapová vrstva. Jak je znázorněno na obrázku 14, tak mapa se může skládat z více vrstev, například vrstva pro silnice, dálnice, vrstevnice, teplotní mapy a další. Ukázkovým příkladem je definice mapového podkladu v této aplikaci. Na obrázku 51 si lze povšimnout dvou stylů, které definují vzhled mapového podkladu spjatého s jedním sloupcem typu **SDO_GEOMETRY**. Jednotlivá mapování budou popsána směrem dolů, neboť tento směr představuje pořadí vykreslování jednotlivých prvků modelu železniční sítě (čím níže prvek je, tím má vyšší prioritu). Důležité však je rozlišit jednotlivé body mezi sebou, tedy určit, který bod je hektometrovník a stanice. K tomu slouží v tabulce sloupec s názvem NODE_TYPE, který určí, pro které body se má určený styl použít. Výčet prvků sloupce NODE_TYPE je následující:

**Tabulka 14 - Výčet prvků sloupce NODE_TYPE**

| NODE_TYPE | Prvek |
|---|---|
| 1 | Hektometrovník |
| 2 | Stanice |

Jelikož je v této práci pracováno s dvěma rozdílnými hustotami modelu železniční sítě, tak musí existovat dva mapové podklady pro hrany a vrcholy (vrstva micro (0) a macro (1)):




**Obrázek 50 - Přehled vrstev železniční sítě [autor]**

Například vykreslení vrstvy vrcholů s názvem NODES_LAYER* spočívá v zobrazení hektometrovníků a následně stanic. Tato posloupnost byla zvolena tak, stanice nebyly překryty hektometrovníky, což by kazilo výsledný dojem ze zobrazeného výstupu. Výsledná podoba spojení stylů s konkrétními prvky modelu železniční sítě je znázorněna na obrázku 51:

| Rendering | | Labeling | |
|---|---|---|---|
| Columns: | | Column: | |
| Style: | HEKTOMETROVNIKMARKER | Style: | |
| Query: | (NODE_TYPE = 1 AND layer = 0) | Function: | -1 |
| Columns: | | Column: | NODE_NAME |
| Style: | STANICEMARKER | Style: | NAZEVSTANICE_FONT |
| Query: | (NODE_TYPE = 2 AND layer = 0) | Function: | 1 |

**Obrázek 51 - Definice mapového podkladu pro vrcholy [autor]**

Vykreslení vrstvy hran LINKS_LAYER* spočívá v prostém zobrazení úsečky, jak zobrazuje následující definice na obrázku 52:

| Rendering | | Labeling | |
|---|---|---|---|
| Columns: | | Column: | |
| Style: | KOLEJ | Style: | |
| Query: | (layer = 0) | Function: | -1 |

**Obrázek 52 - Definice mapového podkladu pro hrany [autor]**

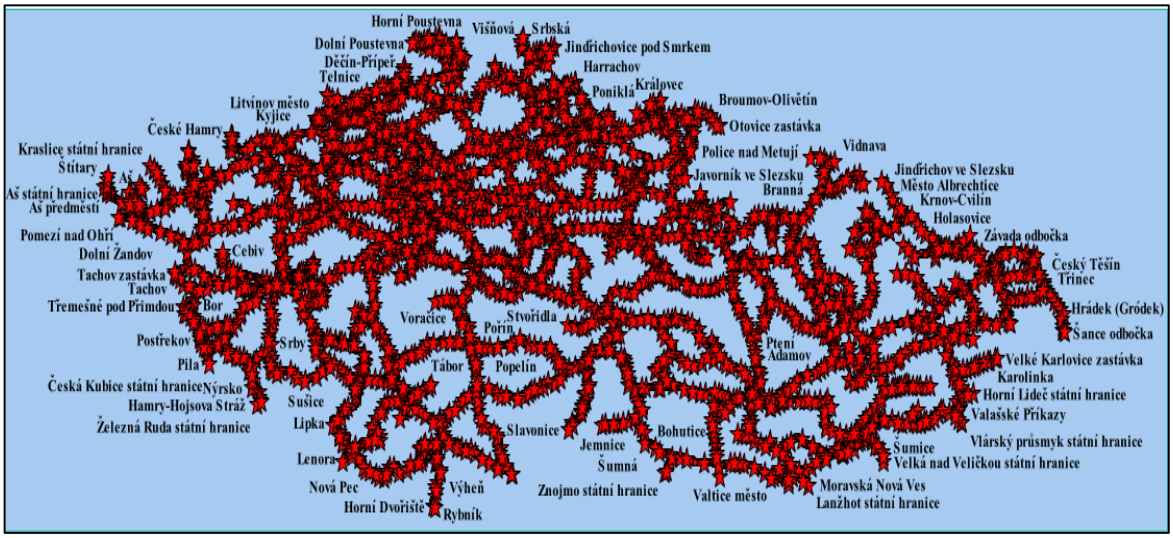
Posledním krokem je definice mapy složené z několika mapových vrstev. Tato mapa se nazývá jako základní (Base map) a zapouzdřuje jednotlivé vrstvy, mezi kterým se přepíná podle aktuální hodnoty přiblížení. V této aplikaci se používají celkem dvě vrstvy, mezi kterými se přepíná podle aktuálního přiblížení. Při vzdáleném pohledu na mapu se používá hrubá vrstva macro a při bližším pohledu jemnější vrstva micro pro zobrazení maxima detailů. Hodnoty

přiblížení, ve kterém dochází při přepínání mezi vrstvami LAYER0 (micro) a LAYER1 (macro) byly zvolena tak, aby vykreslení detailní vrstvy LAYER0 netrvalo déle než pět sekund. Definice základní mapy aplikaci vypadá následovně:

| Theme Name | Theme Properties | |
|---|---|---|
| LINKS_LAYER1 | Minimum Scale: | 1 |
| | Maximum Scale: | 0,025 |
| NODES_LAYER1 | Minimum Scale: | 1 |
| | Maximum Scale: | 0,025 |
| LINKS_LAYER0 | Minimum Scale: | 0,025 |
| | Maximum Scale: | |
| NODES_LAYER0 | Minimum Scale: | 0,025 |
| | Maximum Scale: | |

**Obrázek 53 - Definice základní mapy s názvem ZELEZNICNI_SIT_BASE [autor]**

Jak u definic jednotlivých vrstev, tak i u základní mapy je umožněno vývojáři nahlédnout na požadovaný výstup. Vrstva umožňuje nahlédnout pouze na data týkající se této vrstvy, zatímco základní mapa na výstupu zobrazí výslednou mapu složenou z několika zvolených vrstev Výstup je znázorněn na obrázku 54:




**Obrázek 54 - Náhled na mapu v nástroji MapBuilder [autor]**

Cílem nástroje MapBuilder je uložit vytvořená metadata do databáze. Jelikož k výstupům tohoto nástroje není možné přistupovat z vlastní aplikace, tak je nutné využít službu MapViewer,

která pomocí svého API[17] zpřístupňuje funkce pro zobrazení výsledné mapy. O tomto nástroji v podkapitole 3.5.2.

### 3.5.2 Oracle MapViewer

Klíčovou službou je právě MapViewer, která se stará o zobrazení mapových podkladů prostorových dat v databázi Oracle. Předpokladem pro využívání této služby je přítomnost aplikačního serveru, na kterém je tato komponenta nainstalována. Pro zachování maximální možné kompatibility byl využit aplikační server Oracle WebLogic od stejnojmenné společnosti Oracle.

Je-li komponenta nasazena na aplikačním serveru, tak jediným krokem, který je dále třeba, je definice takzvaného datového zdroje. V datovém zdroji je třeba definovat přístup k databázi, jež hostí prostorová data. Bohužel grafické webové rozhraní nenabízí intuitivní způsob změny konfiguračních souborů, a tak je nutné manuálně vepsat potřebné položky, které mohou mít následující podobu:

```
<map data source name="spatial"
      jdbc host="fei-sql-zd.upceucebny.cz"
      jdbc sid="ee11"
      jdbc port="1521"
      jdbc user="st43312"
      jdbc password="!mypassword"
      jdbc mode="thin"
      number of mappers="3"
      allow jdbc theme based foi="false"
      editable="false"
 />
```

Zobrazená konfigurace obsahuje následující položky:

- **jdbc_host** – URL k připojení k databázi.
- **jdbc_sid** – Síťový identifikátor.
- **jdbc_port** – Databázový port.
- **jdbc_user** – Uživatel databáze.
- **jdbc_password** – Heslo uživatele (počáteční vykřičník znamená, že heslo bude při dalším spuštění MapViewer převedeno na hash kód.

[17] Application Programming Interface – rozhraní pro programování aplikací

- **jdbc_mode** – Režim JDBC ovladače.
- **number_of_mappers** – Maximální počet požadavků v jednom čase.
- **allow_jdbc_theme_based_foi** – Režim kompatibility při problémech s JDBC.
- **editable** – Povolení editace mapy.

Po dokončení konfigurace a restartování je možné využívat službu MapViewer pomocí možných aplikačních rozhraní. Podporovaným aplikačním rozhraním jsou knihovny jazyka JavaScript, které umožňují zobrazovat mapu ve webovém prohlížeči. Zastaralou možností je využívat knihovnu mvclient.jar, která umožňuje pracovat s touto službou z Java aplikace. Tato možnost je využívána i v této aplikaci, jelikož je rozhraní vytvořeno v knihovně JavaFX.

Využívání služby není složité a postačí k tomu jen správné nastavení příslušného objektu `MapViewer` a zavolání metody `java.awt.Image run()`. Ta navrátí obrázek mapy vygenerovaný dle nastavených parametrů v uvedeném objektu.

## 3.6 Popis balíčků aplikace pro generování modelu sítě

V následujících kapitolách budou představeny třídy důležitých balíčků, které jsou použity v praktické části této diplomové práce. Ke konkrétnímu balíčku budou popsány jednotlivé třídy a na závěr bude znázorněn vygenerovaný diagram návrhových tříd pomocí nástroje vývojového prostředí IntelliJ IDEA.

### 3.6.1 Balíček Importing

Úkolem tohoto balíčku je shromáždit veškeré třídy a rozhraní potřebné k provedení importování dat do aplikace. Jedná se prosté operace načtení dat z DBF souborů do připravených kolekcí.

Jak bylo uvedeno v předchozích kapitolách, tak bylo využito techniky programování proti rozhraní, aby nebyly algoritmy závislé na konkrétní implementaci třídy a bylo tak zajištěna modularita celého systému.

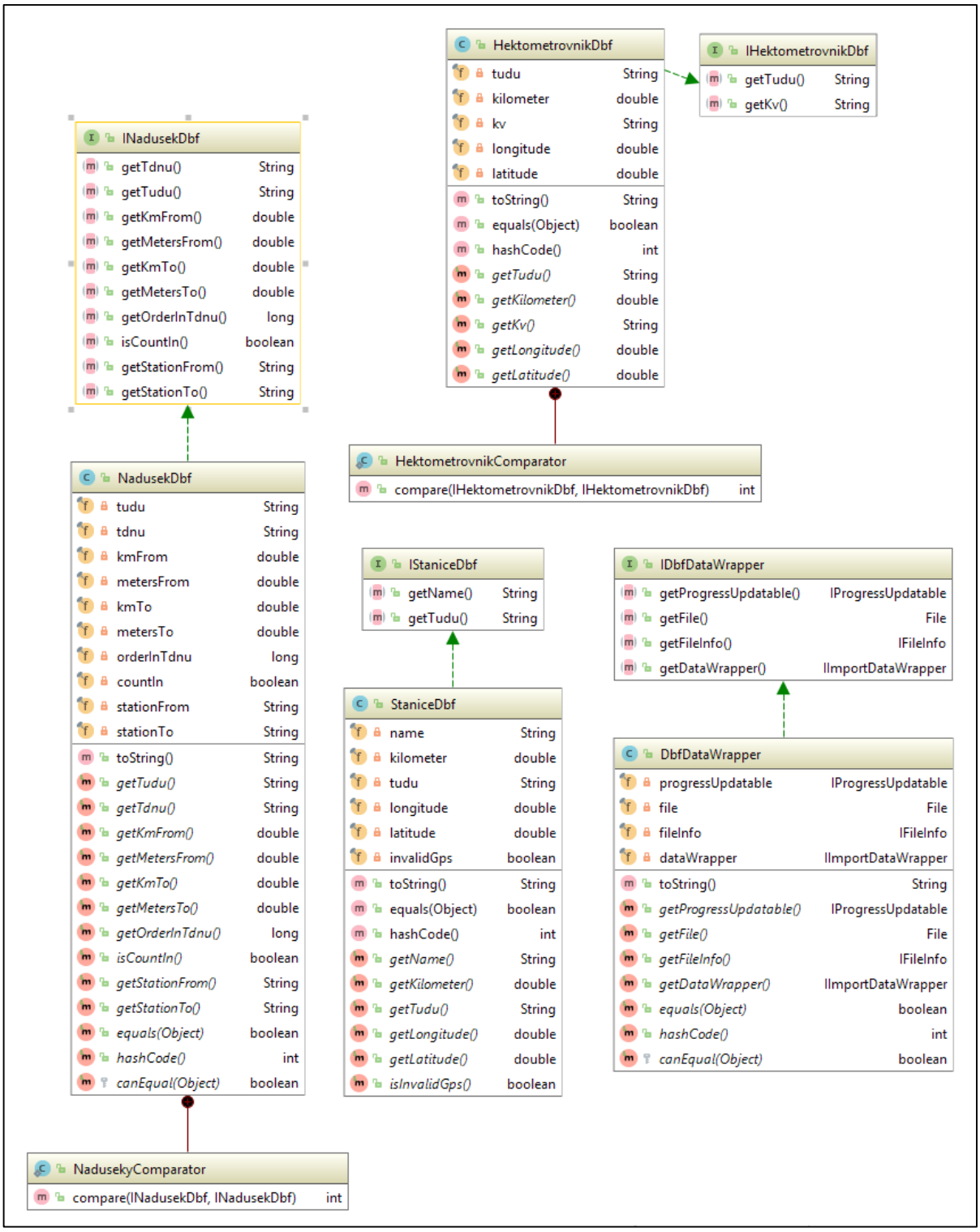
Pro jednotlivé soubory DBF jsou vytvořeny rozhraní a k nim odpovídající třídy, kde instance těchto tříd představuje jeden načtený řádek. Co se názvosloví týče, tak v celé aplikaci jsou datové třídy v balíčku zvaném Models a jejich rozhraní v balíčku Interfaces (rozhraní začínají vždy písmenem „I“) Datovými třídami jsou:

- **HektometrovnikDbf** (**IHektometrovnikDbf**) – Objekt hektometrovníku.
- **StaniceDbf** (**IStaniceDbf**) – Objekt stanice.
- a **NadusekDbf** (**INadusekDbf**) – Objekt nadúseku.

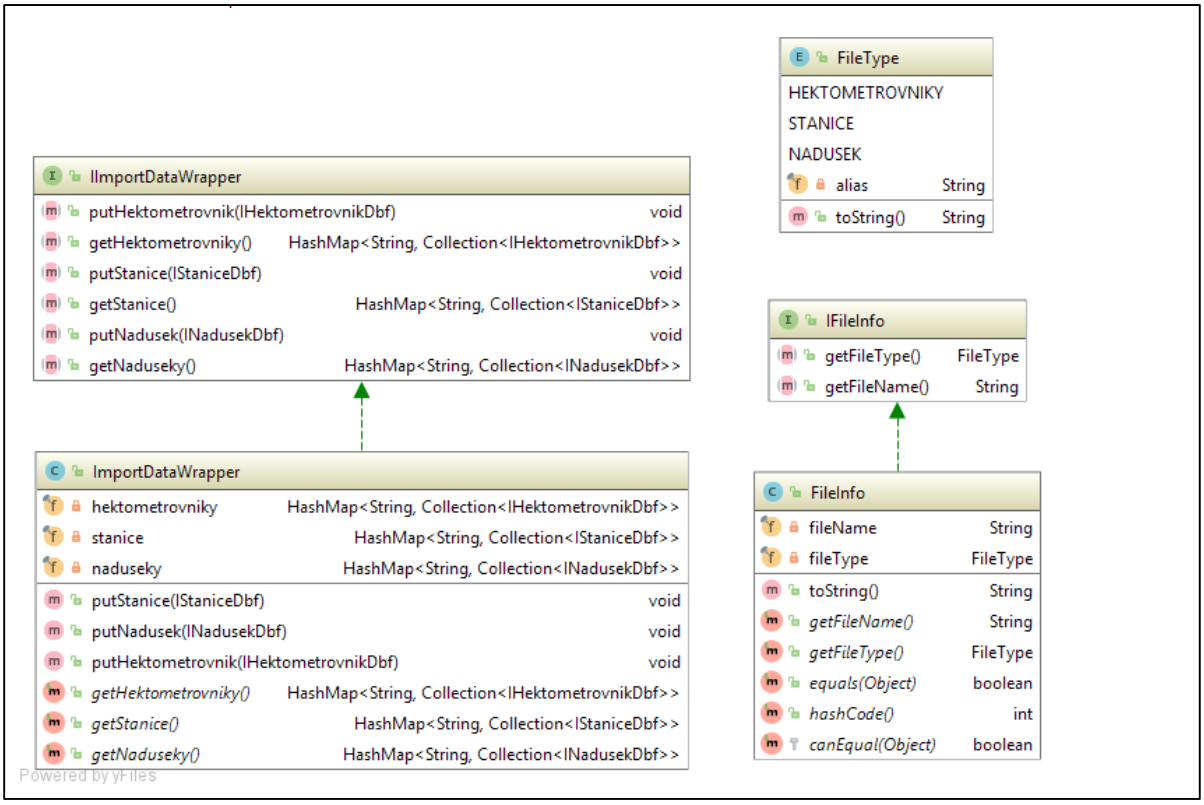
Hlavní třída pro načítání DBF souborů se nazývá **ReadingDbf**. Ta poskytuje jednu statickou metodu **void read(IDbfDataWrapper)**, která na základě předaného objektu získá potřebné informace pro načtení souboru a získané hodnoty uloží do objektu **IImportDataWrapper**. V balíčku se tedy nachází tyto další třídy:

- **DbfDataWrapper** (**IDbfDataWrapper**) – Informace k určení, který soubor se má načítat.
- **FileType** – Výčtový typ definující druhy souborů (hektometrovníky, stanice a nadúseky).
- **FileInfo** (**IFileInfo**) – Zapouzdřuje **FileType** a název souboru rovněž pro potřeby grafického uživatelského rozhraní.
- **ImportDataWrapper** (**IImportDataWrapper**) – Zapouzdřuje načtená data.

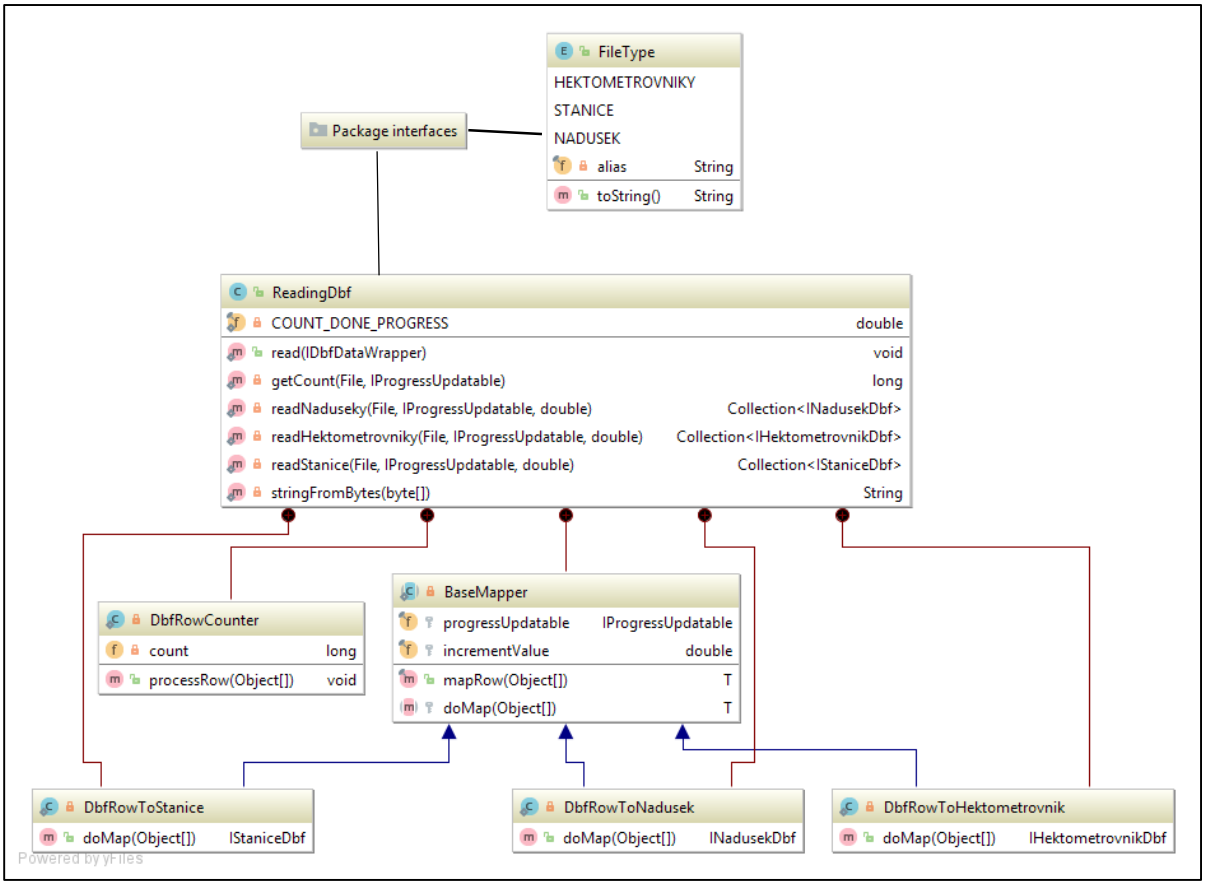
Třída **ReadingDbf** rovněž obsahuje sadu vnitřních tříd pro zjednodušení práce při načítání souborů a zajištění větší modularity. Samotný diagram návrhových tříd musí být rozdělen na několik stránek z důvodu velkého rozsahu. Třída **ReadingDbf** bude zobrazena samostatně. Nejprve budou znázorněny jednotlivé implementace pomocných tříd a rozhraní:




**Obrázek 55 - Diagram tříd balíčku Importing, část 1 [autor]**




**Obrázek 56 - Diagram tříd balíčku Importing, část 2 [autor]**




**Obrázek 57 - Diagram všech vazeb balíčku Importing [autor]**

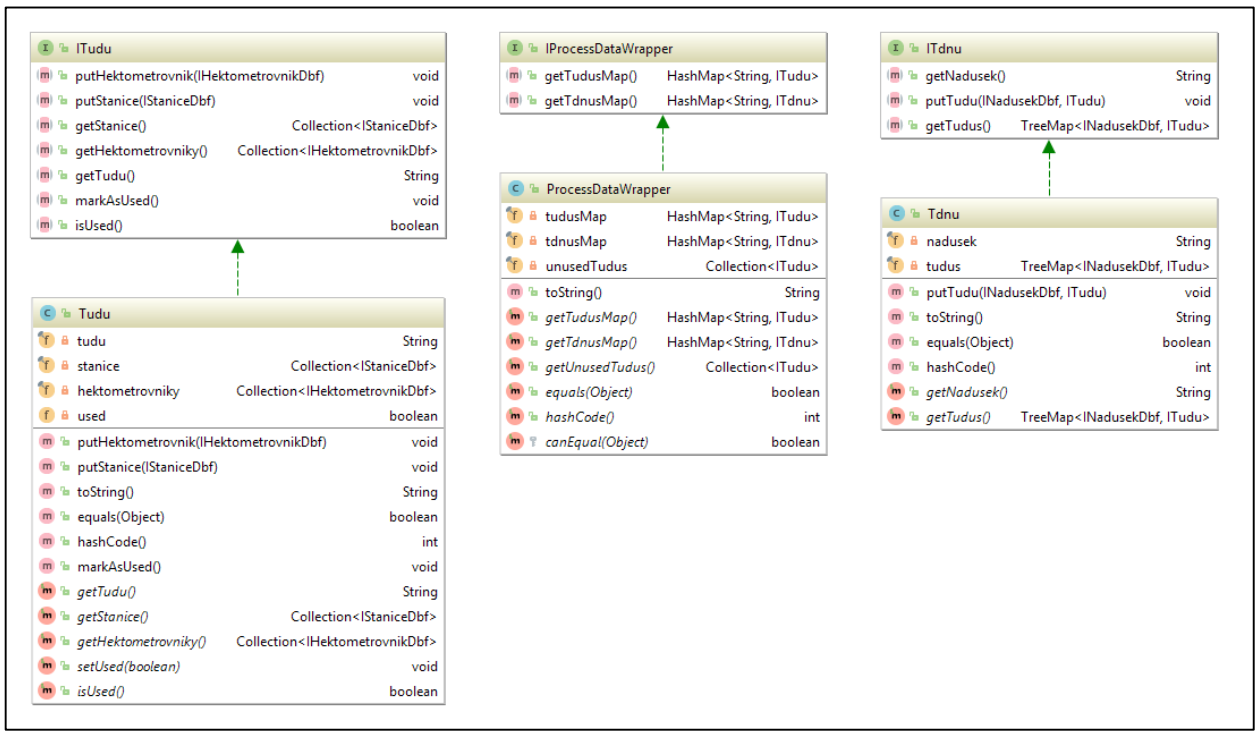
### 3.6.2 Balíček Preprocesing

Úkolem tříd v tomto balíčku je sestavit z importovaných dat logické celky. Příkladem je seskupení všechn hektometrovníků a stanic podle hodnoty traťového definičního úseku. To nadále umožní jednoduchou práci s těmito objekty, neboť lze jednoznačně určit, kam přesně údaje patří.

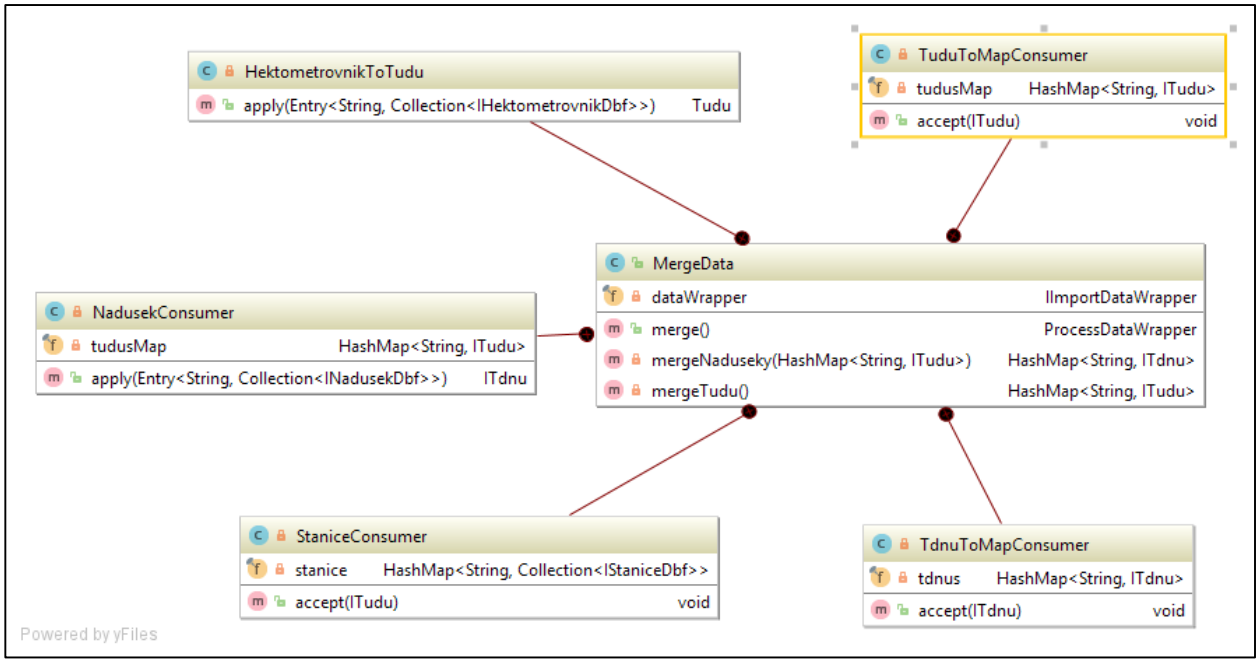
Mezi hlavní datové třídy patří:

- **`Tudu`** (**`ITudu`**) – datová třída pro seskupení hektometrovníků a stanic pod jeden úsek
- **`Tdnu`** (**`ITdnu`**) – datová třída nadúseku pro seskupení souvisejících **`ITudu`**.

Hlavní třída, která pomocí statických operací provádí seskupení dat, se nazvá **`MergeData`**. Jejím úkolem je jak pro úseky, tak i nadúseky, seskupit jednotlivá data. Musí být však zachována posloupnost zpracování dat tak, aby se nejdříve vytvořily objekty **`ITudu`**, která seskupuje nejdříve hektometrovníky a stanice pro všechny dostupné traťové definiční úseky a dále tyto úseky přiřadí jednotlivým nadúsekům. Vlivem nedokonalosti dat tato třída zjišťuje i úseky, které nepatří do žádného nadúseku. Všechny tyto informace se uloží do obalové třídy s názvem **`ProcessDataWrapper`** (**`IProcessDataWrapper`**). Řešení traťových definičních úseků bez nadúseků bylo vysvětleno v kapitole 3.3.5. Diagram návrhových tříd bude rovněž rozdělen do dvou částí, kde první zobrazí datové třídy a jejich rozraní a druhá hlavní třídu **`MergeData`**:




**Obrázek 58 - Diagram tříd balíčku Preprocessing [autor]**



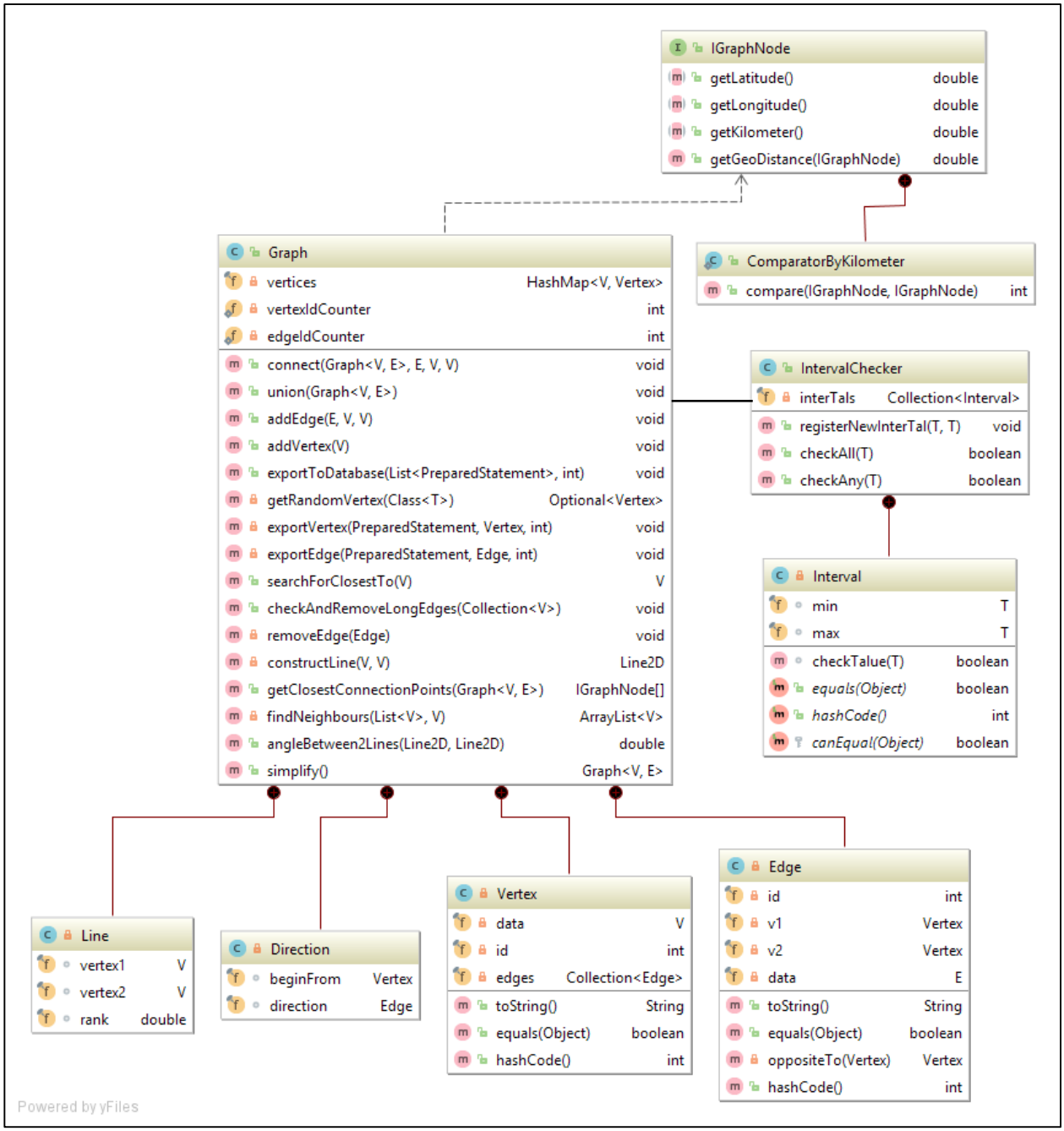

**Obrázek 59 - Diagram třídy MergeData balíčku Preprocessing [autor]**

### 3.6.3 Balíček GraphBuilding

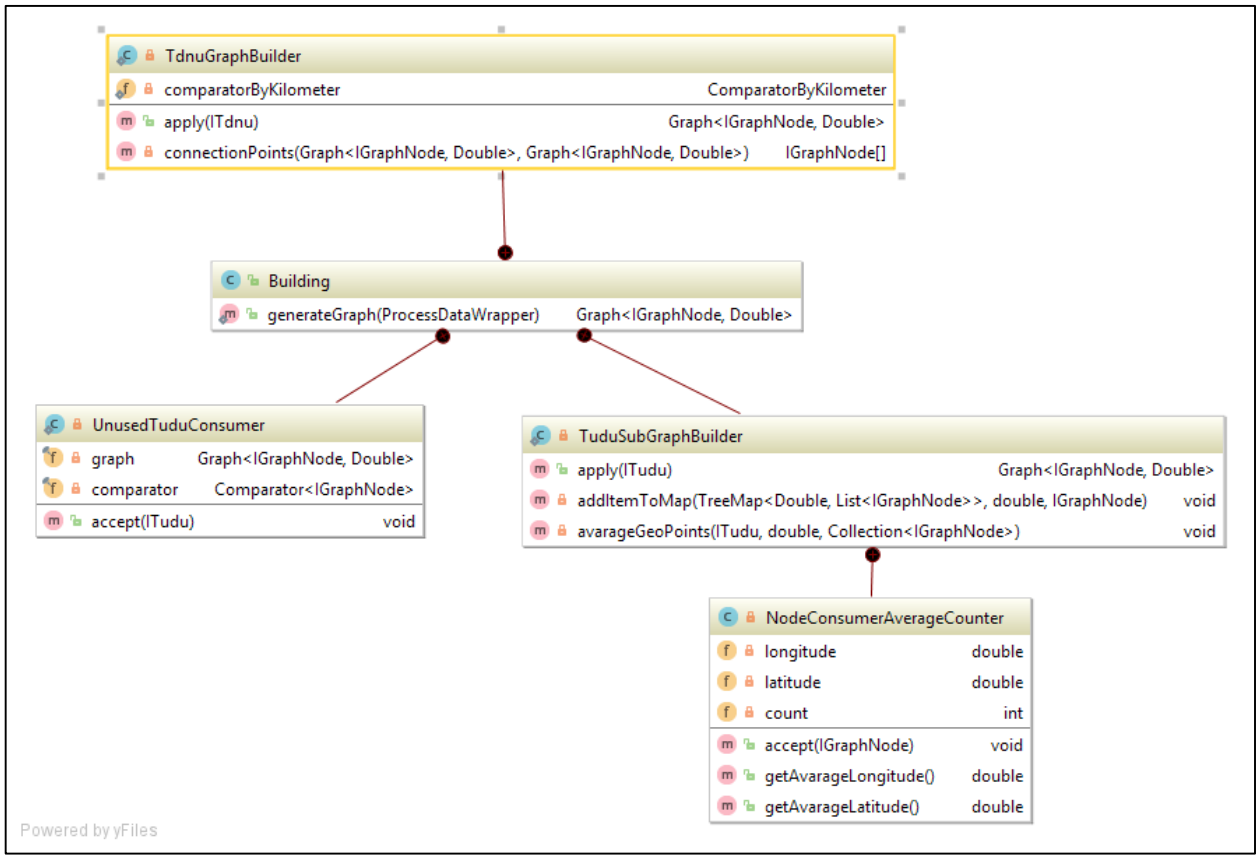
Třídy balíčku GraphBuilding mají na starosti všechny operace týkající se sestavování modelu železniční sítě. Hlavní třídou pro sestavování modelu je třída s názvem **`Building`**. Tato třída pomocí jedné statické metody **`Graph<IGraphNode, Double> generateGraph(ProcessDataWrapper)`** vytvoří datovou strukturu typu **`Graph<IGraphNode, Double>`**, která představuje vygenerovaný model železniční sítě. Při generování se používají tři vnitřní třídy implementující rozhraní **`Consumer`** a **`Function`** pro možnost práce s objekty pomocí proudů (Stream), jež jsou novinkou v Java verzi 8. Vnitřní třídy slouží pro:

- vygenerování grafu z jednotlivých traťových definičních úseků,
- spojení traťových definičních úseků v jeden graf představující nadúsek
- a spojení grafů nadúseků v jeden kompletní graf.

Hlavní rutině napomáhá řada doplňkový tříd. Rozhraní **`IGraphNode`** homogenizuje hektometrovníky a stanice tak, aby k těmto prvkům byl sjednocen přístup (dále v kapitole 3.3.1). Čistě doplňkovou třídou je ta s názvem **`IntervalChecker`**, jejíž úkolem je provádět kontrolu, spadá-li hodnota do všechn zaregistrovaných intervalů. Tato třída se používá pro kontrolu úhlů při operacích popsaných v kapitole 3.3.3. Diagram návrhových tříd bude rovněž rozdělen na podpůrné třídy a na hlavní třídu **`Building`** provádějící hlavní rutinu sestavování:



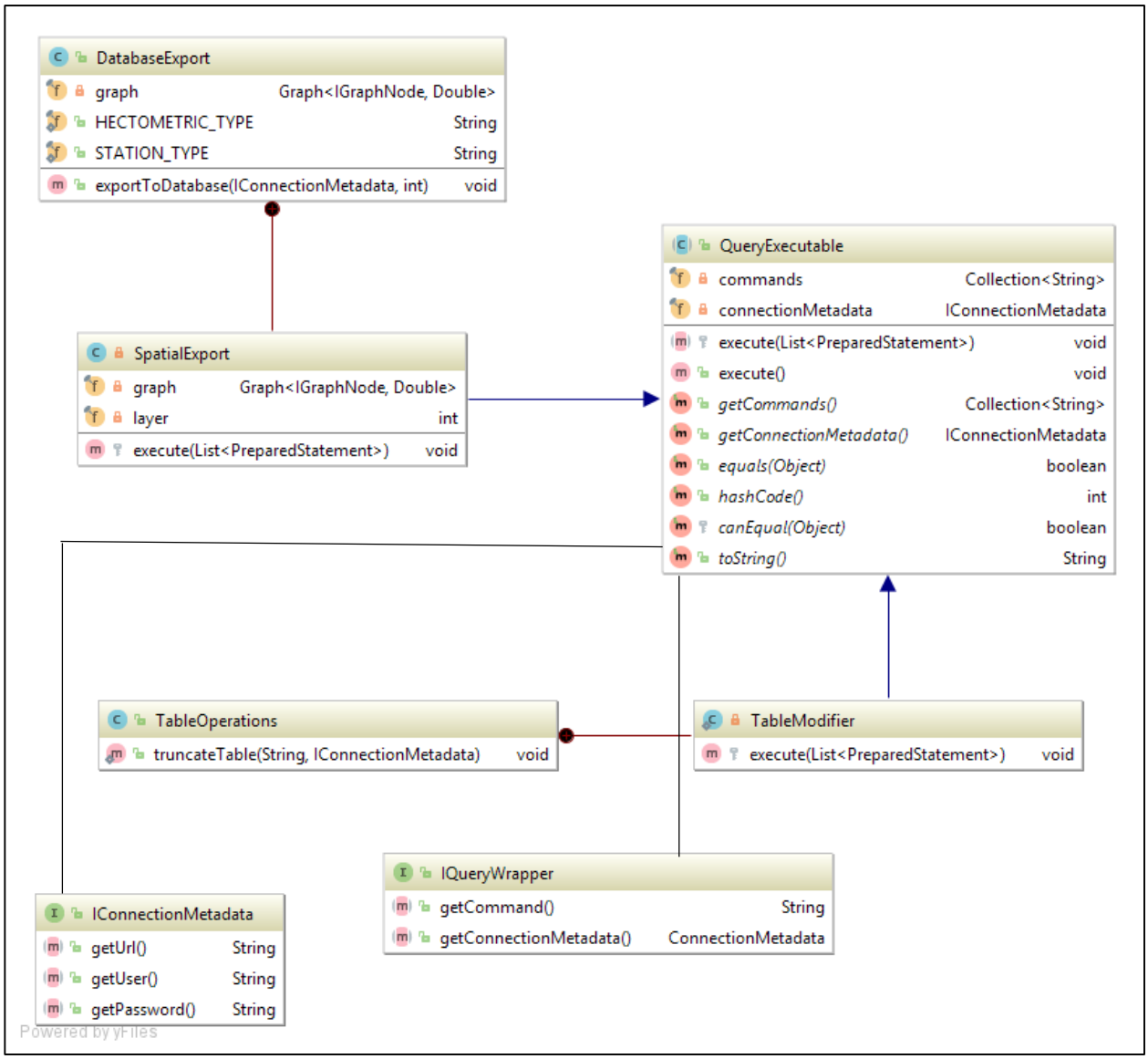

**Obrázek 60 - Diagram tříd balíčku GraphBuilding [autor]**




**Obrázek 61 - Diagram třídy Building balíčku GraphBuilding [autor]**

### 3.6.4 Balíček Exporting

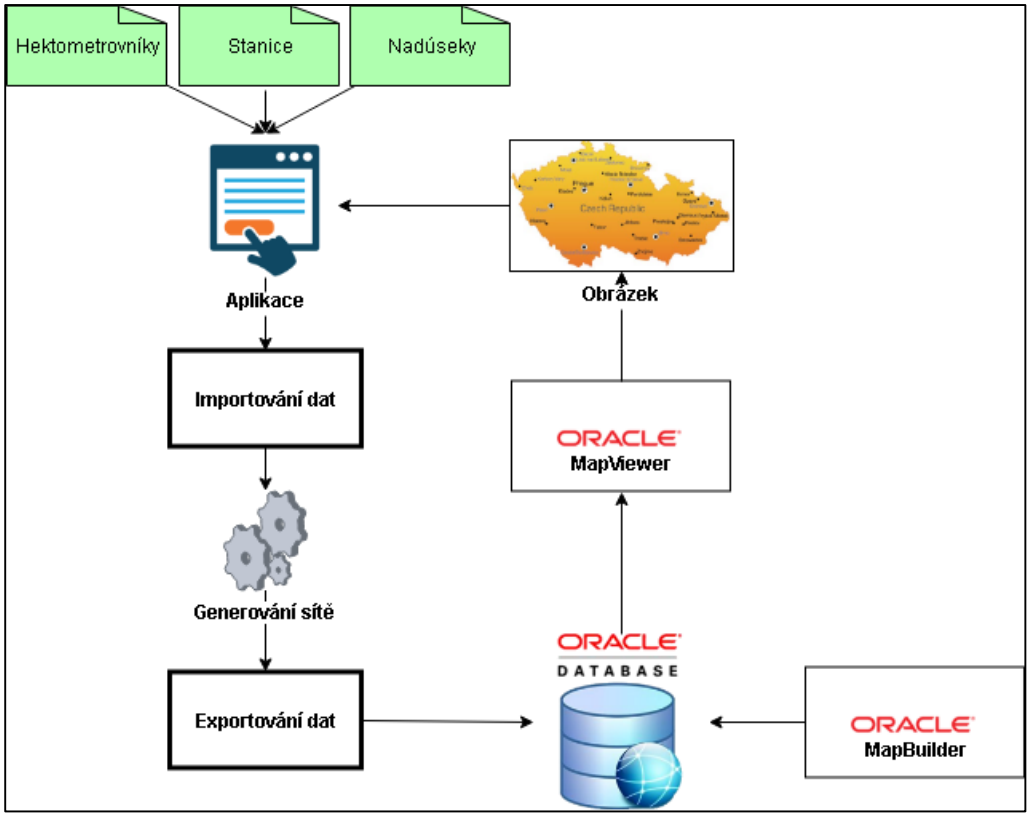
Úkolem tříd uvnitř balíčku Exporting je exportování sestaveného modelu železniční sítě do databáze Oracle. Hlavní třída **`DatabaseExport`** pomocí statické metody **`void exportToDatabase(IConnectionMetadata, int layer)`** exportuje datovou strukturu **`Graph`** do databáze na macro či micro vrstvu. První parametr **`IConnectionMetadata`** definuje metadata připojení k databázi. Jedinou doplňkovou třídou, jež je součástí tohoto balíčku, je třída **`TableOperations`**, která poskytuje statickou metodu **`void truncateTable(String table, IConnectionMetadata)`**, jejíž úkolem je provést smazání tabulky podle názvu udávaném v parametru table. Diagram návrhových tříd vypadá následujícím způsobem:



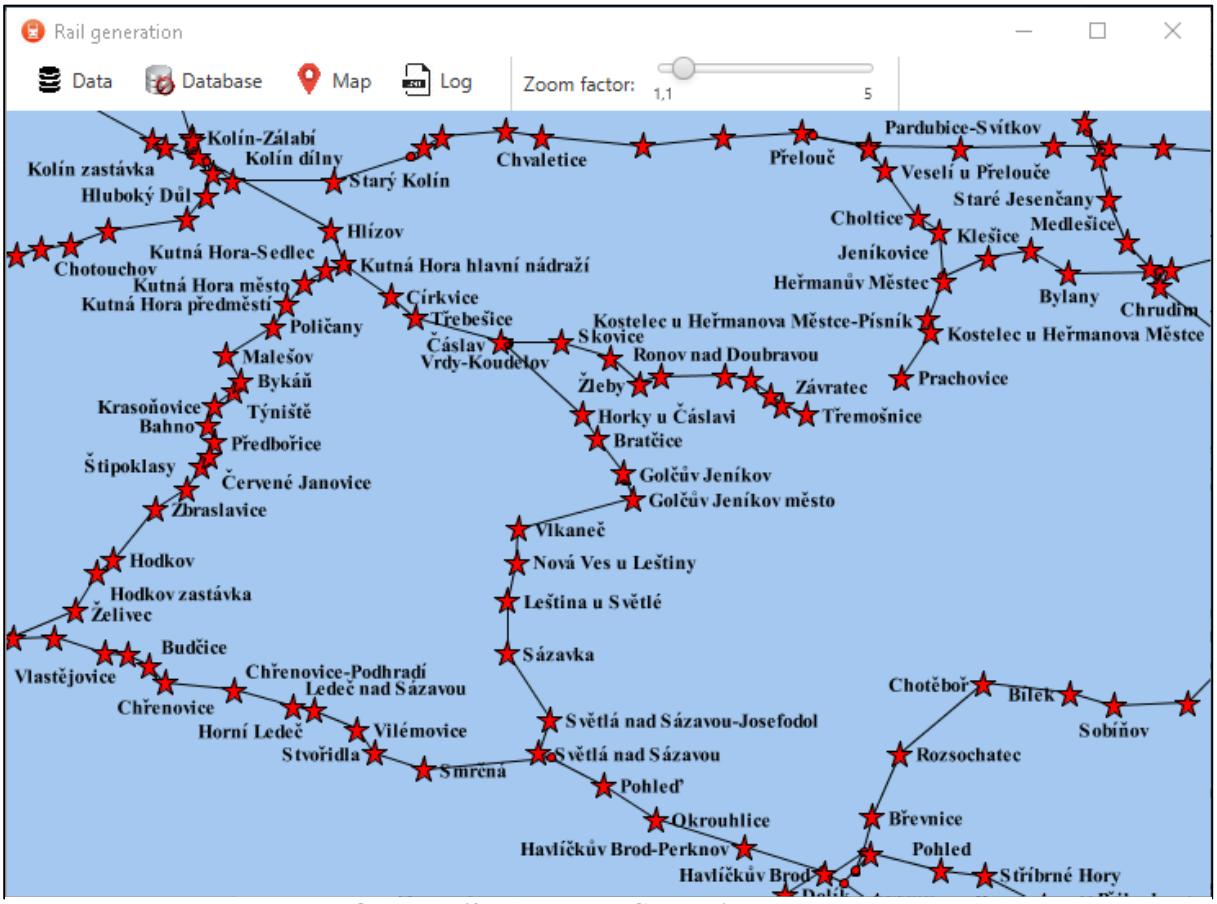

**Obrázek 62 - Diagram tříd balíčku Exporting [autor]**

## 3.7 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní bylo rozvrženo do horní lišty obsahující několik menu položek a indikátory stavu. Zbylý prostor byl vyhrazen pro zobrazení mapového podkladu. Responzivita je řešena tím způsobem, že prostor pro menu je vždy stejně vysoký, ale proměnlivě široký. Prostor pro zobrazení mapy je přizpůsobitelný ve všech směrech. Pro úplně pochopení, které procesy a technologie se v aplikaci používají, slouží obrázek 64. Ihned pod tímto diagramem se nachází pohled na grafické uživatelské rozhraní aplikace:




**Obrázek 64 - Diagram procesů a technologií aplikace [autor]**




**Obrázek 63 - Náhled na GUI aplikace [autor]**

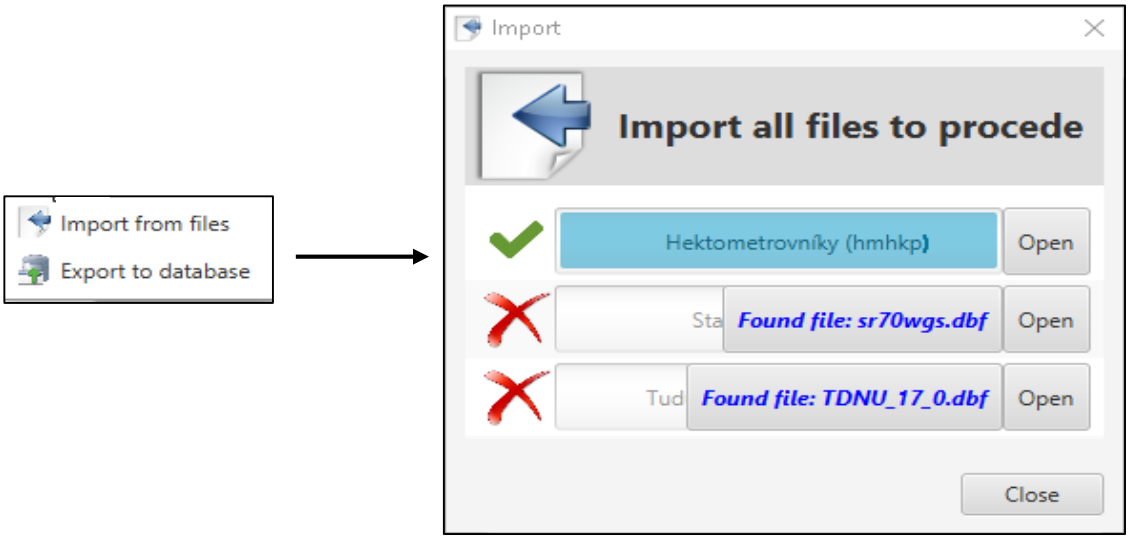
Horní menu se skládá z následujících položek:

- **Data** – Slouží pro importování a exportování dat.
- **Database** – Možnosti pro připojení, odpojení k databázi a potvrzení či zamítnutí změn.
- **Map** – Operace pro práci s mapou týkající se úpravy poměru mapy, uložení do obrázku a obnovy mapy.
- **Zoom factor** – Interaktivní ovládání hodnoty přiblížení (otáčením kolečka myši).
- **a Log** – Možnosti pro práci s logovacími nástroji.

### 3.7.1 Položka Data

Toto menu slouží pro importování a exportování dat. Při importování je zobrazeno dialogové okno pro vybrání všech požadovaných souborů. Jsou-li všechny soubory načtené a je stisknuto tlačítko close, pak se v pozadí ve vlastním vlákně začnou provádět tři zmíněné fáze generování modelu železniční sítě, který je uložen do paměti do doby, než je graf regenerován dalším importem. Importování dat poskytuje doplňkové a optimalizační funkcionality zpříjemňující tuto činnost:

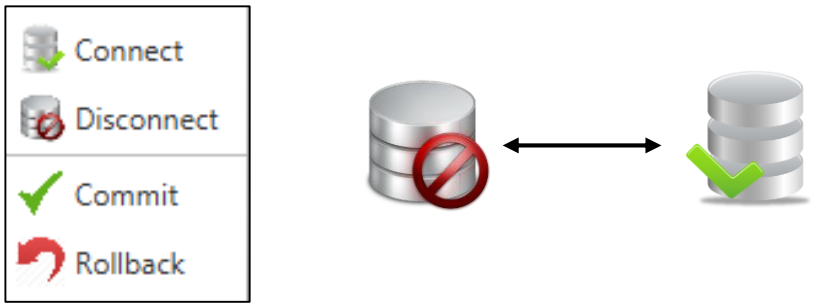
- Je-li otevřen dialog pro importování dat a nic není změněno, pak se graf samozřejmě negeneruje.
- Nachází-li se některé soubory ve stejném adresáři, pak je uživateli poskytnuta možnost tyto soubory načíst bez nutnosti je vyhledávat v souborovém systému na disku. Dostupnost této funkce je podmíněno neměnnými názvy souborů, jimiž jsou ty popsané v kapitole 2.2:




**Obrázek 65 - Možnosti menu Data s náhledem [autor]**

### 3.7.2 Položka Database

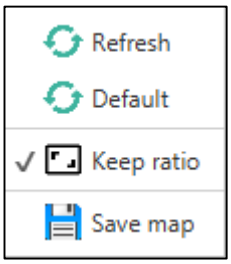
Toto menu slouží pro práci s databází. Umožňuje funkce připojení a odpojení k databázi, a také potvrzení či zamítnutí provedených změn. Připojení k databázi probíhá v hlavním vláknu aplikace, protože je to operace výchozí pro ostatní služby. Za zmínku stojí jistě dynamická ikona v hlavním menu měnicí se v závislosti na připojení k databázi. Grafická reprezentace vypadá následovně:




**Obrázek 66 - Možnosti menu Database [autor]**

### 3.7.3 Položka Map

Toto menu slouží pro ovládání hlavní komponenty pro zobrazení mapy. Veškeré tyto funkce kromě možnosti uložení mapy jsou prováděny ve vlastním vláknu, a to tím způsobem, že je-li již prováděna některá operace pracující se službou MapViewer, tak žádná jiná činnost s mapou kromě uložení nejde provést. Tento limit je dán neměnným exkluzivním přístupem k objektu pro práci se službou MapViewer. Možnosti pro práci s mapou jsou následující:



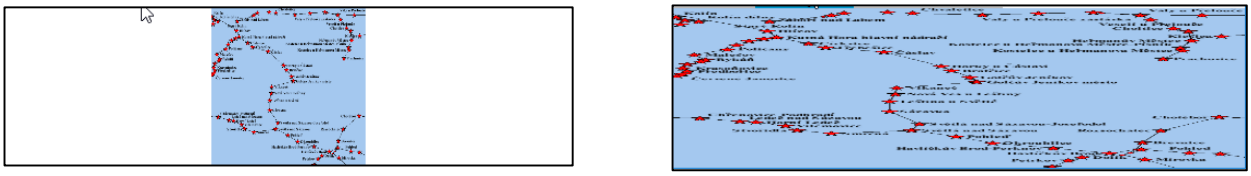

**Obrázek 67 - Možnosti menu Map [autor]**

Vysvětlení k jednotlivým položkám:

- **Refresh** – Načte znovu mapu s polední hodnotou přiblížení. Každá aktualizace znamená přizpůsobení služby MapViewer aktuální dimenzi zobrazovací plochy, což vždy znamená vrácení obrázku ve správném poměru nezávisle na nastavení položky „Keep ratio“.
- **Default** – Výchozí pohled na mapu při prvním spuštění aplikace (výchozí pohled zabraňuje delšímu načítání celé rozlohy modelu železniční sítě). Výchozí hodnoty byly zjištěny nástrojem MapBuilder.

- **Keep ratio** – Při změně okna je zachován poměr obrázku (je-li zaškrtnuto). V opačném případě se obrázek roztáhne maximálně do rozměru aktuální velikosti okna (může nastat deformace). Efekt je viditelný pouze při změně rozměru okna.
- **Save map** – Umožňuje uložit načtený obrázek mapy do souboru.

Ukázka vlivu nastavení položky „Keep ratio“ (vlevo ANO, vpravo NE):



**Obrázek 68 - Funkce úpravy poměru obrázku [Autor]**

### 3.7.4 Položka Log

Toto menu slouží pro práci s logovacími nástroji. Logovací mechanismus je v aplikaci implementovaný od základu, aby vyhověl vlastním požadavkům na aplikaci. Rozlišeny jsou tyto typy logovacích úrovní:

- INFORMATION (informační logy),
- INFORMATION_DIALOG (logy pro zobrazení informačního dialogu),
- EXCEPTION (výjimky, které nezpůsobí pád aplikace),
- CRITICAL (výjimky způsobující pád aplikace),
- a UDENTIFIED (ostatní logy).

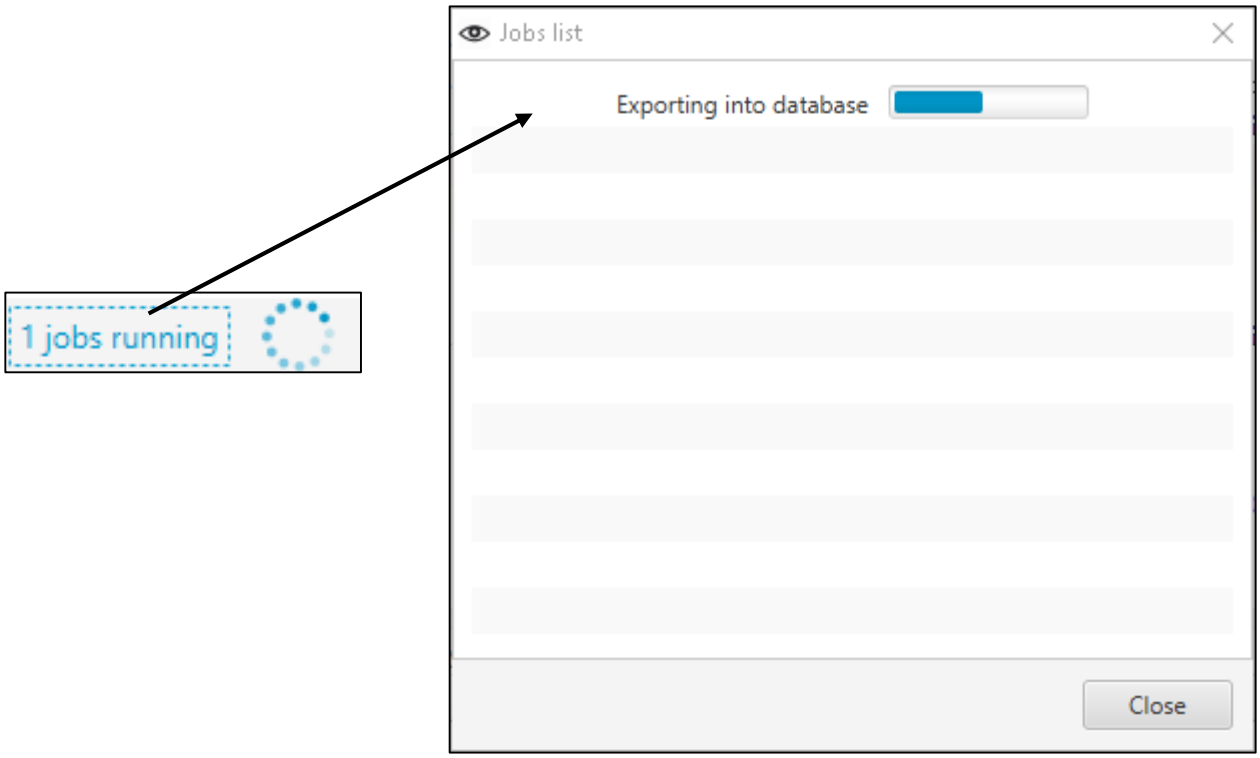
Možnosti, které se skrývají v tomto menu, jsou následující. První možnost zobrazí hierarchický náhled na všechny události v aplikaci seřazené od nejnovějšího po nejstarší. Po kliknutí na nejnižší úroveň (na konkrétní událost) se zobrazí informační dialog o této události. Další možnost se týká zobrazování dialogů výjimek. Pro některé uživatele mohou být tyto dialogy nežádoucí, a tak je možné je vypnout. Nezávisle na aktivaci této volby se při jakékoli výjimce zobrazuje v horní liště aplikace blikající trojúhelník, nabízející náhled na problém zobrazením dialogu výjimek. Možnosti vypadají následovně:


**Obrázek 69 - Možnosti menu Log [autor]**

### 3.7.5 Položka prací

Jelikož je většina operací spouštěna v separátním vlákně, tak je zapotřebí zavést nějakou formu koordinace těchto vláken. Proto byl vyvinut mechanismus třídy **`JobsMonitor`**. Jedná se o návrhový vzor Singleton (jediná instance objektu v aplikaci), která uchovává seznam všech běžících vláken, umožňuje je registrovat do procesu běhu a registruje rutiny po dokončení jednotlivých nebo všech vláken. Jednou takovou rutinou, spouštějící se po doběhnutí každého vlákna, je zobrazování indikátoru počtu běžících vláken. Později bylo rozhodnuto o změně terminologie, kdy vlákna byla přejmenována na práci (job). Práce představuje nějakou činnost, která se může skládat z několika vláken, a tak indikátor zobrazuje počet prací. Většinou jsou v aplikaci použity práce s jedním vláknem, ale je možné se setkat i s větším počtem v rámci jednoho „jobu“. Takovou operací je například paralelní běh mazání obou tabulek, ale i paralelní

běh exportování obou grafů (kompletní a zjednodušená verze). Na položku indikátoru počtu prací je možné kliknout a zobrazit právě prováděné práce. Práce exportování se skládá z mazání a exportování dat. Aby nenastalo k situaci, kdy se zaregistrují dvě stejné práce (například rollback a commit databáze), tak každý prostředek (databáze, mapa a další) má svůj identifikátor, pod kterým je zaregistrován do monitoru vláken. Pokud se má provést nějaká databázová operace a v seznamu prací již některá příbuzná existuje, tak je tato příchozí operace odmítnuta, aby nedošlo ke konfliktům a byla zajištěno vzájemné vylučování vláken ke sdílenému prostředku. Jak je možné vidět na obrázku, tak polovina pokroku značí, že mazání tabulek bylo již vykonáno a aktuálně se provádí exportování dat do databáze:




**Obrázek 70 - Zobrazení běžících činností (prací) aplikace [autor]**

# 4 ZÁVĚR

Cílem teoretické části práce byl popis vybraných datových struktur vhodných pro ukládání prostorových dat. V sekci databázových systémů byly popsány systémy podporující ukládání multidimenzionálních údajů a to primárně ty, které mají majoritní podíl na trhu.

Cílem praktické části práce bylo navrhnout a implementovat algoritmy, jež umožnily vybudovat dvouvrstvý model železniční sítě odrážející neorientovaný graf. Zhodnotí-li se celkový vývoj praktické části diplomové práce, tak obecně lze konstatovat, že se bylo nutné vypořádat s velkým počtem problémů, které byly zapříčiněny nedostatečnou konzistencí bázových dat, anebo zvoleným postupem pro vygenerování modelu železniční sítě. Ke konkrétním problémům bylo nutné přistupovat odlišným způsobem v závislosti na dopadu konkrétní chyby na vytvořenou či stále vytvářející se datovou strukturu.

Z kapitoly 3 plyne, že v bázových datech existuje velké množství výjimek, které algoritmus buď musí vyřešit nějakou vhodnou alternativou, anebo tuto výjimku v konečném modelu zcela zamítnout. Do obou řešení spadá například problém týkající se načítání stanic (popsáno v kapitole 3.1.2), kde existovaly záznamy nejen se špatnými, ale i nulovými GPS koordináty, a proto bylo nutné navrhnout takové řešení, které bude mít co nejmenší dopad na korektnost výsledných dat. Bylo rovněž zapotřebí najít vhodné řešení mnohem závažnějších problémů týkajících se například absence informace o traťovém definičním úseku u některých záznamů, anebo hledání vhodnějšího napojení úseků s dodatečnou kontrolou úhlů. Přes velké snahy o korekce dat pomocí algoritmů se však nepodařilo odstranit všechny výjimky modelu železniční sítě. Tyto výjimky však tvoří tak malé procento, že výsledná úspěšnost vytvořeného modelu je 99 %.

V závěru práce bylo dokázáno, že použití dostupných technologií MapViewer a MapBuilder zjednodušilo rutinu zobrazování modelu železniční sítě do grafického uživatelského rozhraní. Díky těmto dostupným nástrojům nebylo nutné vyvíjet vlastní zobrazovací mechanismus.

Jak plyne z množství úprav algoritmů pro přizpůsobování se jednotlivým nesrovnalostem, tak lze konstatovat, že systém staničení je stále se rozvíjející systém, který v současné době prochází velkým množstvím úprav. Cíle jak teoretické, tak i praktické části práce, byly splněny v celém rozsahu.

# 5 REFERENCE


[1] Using Discoverer Plus with a multidimensional data source. Moved [online]. Copyright © [cit. 16.01.2018]. Dostupné z: https://docs.oracle.com/cd/B14099_19/bi.1012/b13915/i_olap_chapter.htm.

[2] **SAMET, Hanan**. *Foundations of multidimensional and metric data structures*. Boston: Elsevier/Morgan Kaufmann, c2006. [cit. 16.01.2018]. ISBN 9780123694461.

[3] **McCREIGHT, Edward M.** *Priority Search Trees* SIAM J. Comput., Vol. 14, No. 2, pp. 257–276, May 1985. [cit. 17.01.2018].

[4] **ARGE, Lars**. *I/O-Algorithms* [online]. Copyright © 2018 SlidePlayer.com Inc. [cit. 17.01.2018]. Dostupné z: http://slideplayer.com/slide/5092102.

[5] **FINKEL, R. A. a J. L. BENTLEY** (1974). *Quad Trees A Data Structure for Retrieval on Composite Keys*. Acta Informatica. Springer-Verlag. [cit. 17.01.2018].

[6] *15.3. The PR Quadtree — CS3 Data Structures & Algorithms. OpenDSA* [online]. Copyright © Copyright 2016 by OpenDSA Project Contributors and distributed under an MIT license. [cit. 17.01.2018]. Dostupné z: https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/PRquadtree.html

[7] kNN.15 K-d tree algorithm - YouTube. *YouTube* [online]. [cit. 17.01.2018]. Dostupné z: https://www.youtube.com/watch?v=Y4ZgLlDfKDg

[8] Comparison of Relational and Multi-Dimensional Database Structures. alphadevx.com [online]. Copyright © 2001 [cit. 19.01.2018]. Dostupné z: http://www.alphadevx.com/a/36-Comparison-of-Relational-and-Multi-Dimensional-Database-Structures

[9] MySQL :: MySQL 5.7 Reference Manual :: 11.5 Spatial Data Types. MySQL :: Developer Zone [online]. Copyright © 2018, Oracle Corporation and [cit. 22.01.2018]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/spatial-types.html

[10] **GUTTMAN, Antonín**: *R-Trees: A Dynamic Index Structure for Spatial Searching*, Proc. 1984 ACM SIGMOD International Conference on Management of Data, pp. 47-57. ISBN 0-89791-128-8

[11] Computational Geometry Lab – *Index, R-tree* [online]. [cit. 22.01.2018]. Dostupné z: http://www.cglab.ca/~cdillaba/comp5409_project/R_Trees.html

[12] **OSTROŽLÍK, Adam.** *Zobrazení a monitorování senzorů* [online]. Pardubice, 2016 [cit. 2018-01-22]. Dostupné z: http://hdl.handle.net/10195/64915. Bakalářská práce. Univerzita Pardubice. Vedoucí práce Ing. Karel Šimerda.

[13] Spatial Indexing Overview. Object. Moved [online]. Copyright © 2018 Microsoft [cit. 22.01.2018]. Dostupné z: https://technet.microsoft.com/en-us/library/bb964712(v=sql.105).aspx.

[14] Spatial Concepts. Moved [online]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/spatl/spatial-concepts.html#GUID-CD6ABC34-1CB4-476D-ACB1-BDA07026C206.

[15] Multiple Layer Analysis. *Site not found · GitHub Pages* [online]. [cit. 23.01.2018]. Dostupné z: https://saylordotorg.github.io/text_essentials-of-geographic-information-systems/s11-02-multiple-layer-analysis.html.

[16] Spatial Data Types and Metadata. *Moved* [online]. Copyright © [cit. 23.01.2018]. Dostupné z: https://docs.oracle.com/html/B10826_01/sdo_objrelschema.htm.

[17] Coordinate Systems (Spatial Reference Systems). *Moved* [online]. Copyright © [cit. 23.01.2018]. Dostupné z: https://docs.oracle.com/html/B10826_01/sdo_cs_concepts.htm#i890025

[18] **PROKEŠ, Radek**. *Generování železniční sítě: Diplomová práce*. Univerzita Pardubice – Fakulta Informačních technologií, 13. 9. 2011.

[19] Game Programming Gurus. *Yaldex JavaScript Editor* [online]. Dostupné z: http://www.yaldex.com/games-programming/0672323699_ch08lev1sec10.html

[20] Introduction to SQL Server Spatial Data - Simple Talk. *Redgate Software - Tools For SQL Server, .NET, & Oracle* [online]. Dostupné z: https://www.red-gate.com/simple-talk/sql/t-sql-programming/introduction-to-sql-server-spatial-data/

[21] *SlidePlayer - Chargez et faites partager vos présentations PowerPoint* [online]. Dostupné z: http://slideplayer.fr/slide/1173118/3/images/4/MapViewer+Architecture.jpg

[22] **KOTHURI, Ravi, Albert GODFRIND a Euro BEINAT**. *Pro Oracle Spatial for Oracle database 11g*. 2nd ed., New ed. Berkeley, CA: Apress, 2007. ISBN 9781590598993.

# 6 PŘÍLOHY

**Příloha A** – *Ukázka algoritmu zjednodušení modelu železniční sítě*

```
public Graph<V,E> simplify() throws NullPointerException {
    // prepare collections
    final HashSet<Integer> processedEdges = new HashSet<>();
    final Queue<Direction> toScan = new LinkedList<>();

    // pick random station vertex
    final Optional<Vertex> vertexOpt = getRandomVertex(IStaniceDbf.class);
    final Vertex vertex = vertexOpt.get();

    // initial data for cycle below
    vertex.edges.forEach(edge -> toScan.add(new Direction(vertex, edge)));

    // new graph object
    final Graph<V,E> simpledGraph = new Graph<>();

    // until data is present
    while (!toScan.isEmpty()) {
        // pick one direction
        final Direction direction = toScan.poll();
        // finish indicator
        boolean canWalk = true;
        // pick current direction data
        Vertex current = direction.beginFrom;
        Edge directionEdge = direction.direction;

        // if edge has been process already, ignore this direction
        if (processedEdges.contains(directionEdge.id)) continue;

        while (canWalk) {
            // pick opposite vertex
            final Vertex opposite = directionEdge.oppositeTo(current);
            //...
            if (opposite.edges.size() != 2 || opposite.data instanceof IStaniceDbf) {
                // register edge as processed
                processedEdges.add(directionEdge.id);
                opposite.edges.stream()
                        .filter(edge -> !processedEdges.contains(edge.id))
                        .forEach(edge -> toScan.add(new Direction(opposite, edge)));
                Double cost = direction.beginFrom.data.getGeoDistance(opposite.data);
                simpledGraph.addEdge((E)cost, direction.beginFrom.data, opposite.data);
                canWalk = false;
            } else {
                // register edge as processed
                processedEdges.add(directionEdge.id);
                // continue in walking on path if cycle is not done
                current = opposite;
                for (Edge edge : opposite.edges) {
                    if (edge != directionEdge) {
                        directionEdge = edge;
                        break;
                    }
                }
            }
        }
    }

    // restore id
    simpledGraph.vertices.values().stream()
        .forEach(v -> v.id = this.vertices.get(v.data).id);

    return simpledGraph;
}
```

**Příloha B** – *Ukázka algoritmu průměrování prvků železniční sítě*

```
private void avarageGeoPoints(
        final ITudu tudu, final double km,
        final Collection<IGraphNode> graphNodes) {
    final Optional<IGraphNode> station = graphNodes.stream()
            .filter(node -> node instanceof IStaniceDbf)
            .findAny();    // 2 or more stations cannot be at one place

    if (station.isPresent()) {
        // station is present - return station
        graphNodes.clear();
        graphNodes.add(station.get());
    } else {
        // otherwise merge hectometric into single one
        boolean turnout = graphNodes.stream()
                .map(node -> ((IHektometrovnikDbf) node).getKv())
                .distinct()
                .anyMatch(kv -> kv.equals("V"));
        NodeConsumerAverageCounter averageCounter = new NodeConsumerAverageCounter();
        graphNodes.forEach(averageCounter);
        graphNodes.clear();
        graphNodes.add(new HektometrovnikDbf(
                tudu.getTudu(), km, turnout ? "V" : "K",
                averageCounter.getAvarageLongitude(), averageCounter.getAvarageLatitude()
        ));
    }
}
```

**Příloha C (1. část)** – *Ukázka algoritmu hledání nejbližších uzlů včetně korekce úhlů*

```
public IGraphNode[] getClosestConnectionPoints(final Graph<V, E> to) {

    // preparation
    // collections for neighbours
    final LinkedList<V> list1 = new LinkedList<>();
    final LinkedList<V> list2 = new LinkedList<>();
    // fill neighbours
    vertices.values().stream().map(vertex -> vertex.data).forEach(list1::add);
    to.vertices.values().stream().map(vertex -> vertex.data).forEach(list2::add);
    final IGraphNode.ComparatorByKilometer comparator = new IGraphNode.ComparatorByKilometer();
    // sort neighbours by kilometer
    list1.sort(comparator);
    list2.sort(comparator);

    // create lines and find closest verticies for the purpose of finding any suitable angle
    // lines are sort by rank = GPS distance
    final TreeSet<Line> lines = new TreeSet<>(Comparator.comparingDouble(o -> o.rank));
    double min = Integer.MAX_VALUE;
    V found1 = null, found2 = null;
    for (final Vertex v1 : vertices.values()) {
        for (final Vertex v2 : to.vertices.values()) {
            double temp;
            if ((temp = v1.data.getGeoDistance(v2.data)) < min) {
                min = temp;
                found1 = v1.data;
                found2 = v2.data;
            }
            lines.add(new Line(v1.data, v2.data, temp));
        }
    }

    // create iterval checker
    final IntervalChecker<Double> angleChecker = new IntervalChecker<>();
    angleChecker.registerNewInterTal(135d, 225d);

    // for each found generated line
    for (final Line line : lines) {
        // find neighbours of vertices
        final ArrayList<V> neighbours1 = findNeighbours(list1, line.vertex1);
        final ArrayList<V> neighbours2 = findNeighbours(list2, line.vertex2);
        final Collection<Line> neighboursLines = new LinkedList<>();
        /// create lines for neighbours with vertex1
        for (final V neighbour : neighbours1) {
            final double rank = neighbour.getGeoDistance(line.vertex1);
            neighboursLines.add(new Line(line.vertex1, neighbour, rank));
        }
        /// create lines for neighbours with vertex2
        for (final V neighbour : neighbours2) {
            final double rank = neighbour.getGeoDistance(line.vertex2);
            neighboursLines.add(new Line(line.vertex2, neighbour, rank));
        }
        int accept = 0;
        int deny = 0;
```

**Příloha C (2. část) -** *Ukázka algoritmu hledání nejbližších uzlů včetně korekce úhlů*

```
        // calculate angles for each neighbour
        for (final Line neighbour : neighboursLines) {
            // find common vertex
            final V intersect = line.intersect(neighbour);
            // calculate angle
            double angle = angleBetween2Lines(
                    constructLine(intersect, line.opposite(intersect)),
                    constructLine(intersect, neighbour.opposite(intersect)));
            if (angleChecker.checkAny(angle)) {
                ++accept;
            } else {
                ++deny;
            }
        }
        if (accept >= deny) {
            return new IGraphNode[] {line.vertex1, line.vertex2};
        }
    }
    // if no line was found by angle, return vertices by each-to-each algorithm
    return new IGraphNode[] {found1, found2};

}
```

**Příloha D** – *Přiložené CD*

Obsah přiloženého CD:

- zdrojový kód aplikace (spustitelný projekt v prostředí IntelliJ IDEA),
- spustitelný JAR soubor aplikace,
- podpůrné soubory pro běh aplikace,
- a elektronická podoba práce ve formátu PDF.