

UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Komparace výkonu Oracle Database a grafové  
databáze Neo4j při grafových úlohách

Bc. Vít Přenosil

Diplomová práce  
2014

Univerzita Pardubice  
Fakulta elektrotechniky a informatiky  
Akademický rok: 2013/2014

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Vít Přenosil**  
Osobní číslo: **I11406**  
Studijní program: **N2646 Informační technologie**  
Studijní obor: **Informační technologie**  
Název tématu: **Komparace výkonu Oracle Database a grafové databáze Neo4j při grafových úlohách**  
Zadávající katedra: **Katedra softwarových technologií**

### Z á s a d y   p r o   v y p r a c o v á n í :

Cílem diplomové práce je porovnat výkon databázového systému Oracle (verze 11g nebo 12c) oproti čistě grafové v databázi Neo4j. Těžiště testů bude ležet na testování typických grafových operací. Výstupem praktické části bude aplikace (desktopová nebo webová), která bude sloužit ke spouštění a modifikaci předpřipravených testů a k vizualizaci výsledků jednotlivých testů. Teoretická část probere problematiku reprezentace grafových struktur v klasických relačních databázích. Bude provedena rešerše existujících řešení. Dále se teoretická část zaměří na představení existujících grafových databázích, především se zaměří na to, jakým způsobem jsou v jednotlivých grafových databázích grafy fyzicky reprezentovány a jakým způsobem probíhá samotné dotazování nad grafy.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

Robinson, Ian, Emil Eifrem, Jim Webber. Graph Databases. Sebastopol, CA [u.a.: O'Reilly Et Associates, 2013. ISBN 978-1-4493-5626-2.  
Steven S. Skiena. The Algorithm Design Manual (Second Edition). New York, USA, Springer, 2008. ISBN 978-1-84800-069-8.  
Kothuri, Ravi, Albert Godfrind, and Euro Beinat. Pro Oracle Spatial for Oracle Database 11g. Berkeley, CA: Apress, 2007. ISBN 978-1-59059-899-3.  
Kyte, Thomas. Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions. [Berkeley, Calif.]: Apress, 2010. ISBN 978-1-4302-2946-9.  
<http://www.neo4j.org/>  
<http://www.oracle.com/cz/products/database/overview/index.html>

Vedoucí diplomové práce:

**Ing. Tomáš Váňa**

Katedra informačních technologií

Datum zadání diplomové práce:


**31. října 2013**

Termín odevzdání diplomové práce:

**16. května 2014**



prof. Ing. Simeon Karamazov, Dr.  
děkan



L.S.



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

V Pardubicích dne 15. listopadu 2013

## **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 21. 8. 2014

Vít Přenosil

## **Poděkování**

Rád bych poděkoval svému vedoucímu práce za pomoc s jakýmkoliv problémem. Dále bych rád poděkoval své rodině, že mě podporovala po celou dobu mého studia.

## **Anotace**

Tématem diplomové práce je porovnání výkonu databázového systému Oracle (verze 11g) oproti čistě grafové databázi Neo4j (verze 2.0.3) při grafových úlohách. Cílem práce je naprogramovat aplikaci pro spouštění a úpravu předpřipravených testů. Dále bude třeba vizualizovat výsledky jednotlivých testů. Testování se bude týkat typických grafových operací (Problém obchodního cestujícího, nejkratší cesta a minimální kostra grafu).

## **Klíčová slova**

Graf, grafová databáze, relační databáze, komparace, test

## **Title**

Comparison of the performance of Oracle Database and graph database Neo4j for graph tasks

## **Annotation**

The topic of diploma thesis is to compare the performance of database system Oracle (version 11g) against pure graph database Neo4j (version 2.0.3) in the context of solving graph problems. The goal is to program the application for running and modifying pre-built tests. The next necessary step is to visualize the results of particular tests. Testing will cover the typical graph operations (travelling salesman problem, the shortest path and minimum cost spanning tree).

## **Keywords**

Graph, graph database, relational database, comparison, test

## Obsah

<b>Seznam zkratek .....</b>	<b>9</b>
<b>Seznam obrázků .....</b>	<b>10</b>
<b>Seznam tabulek .....</b>	<b>11</b>
<b>Úvod .....</b>	<b>13</b>
<b>1 Graf.....</b>	<b>14</b>
1.1 Co je graf .....	14
1.2 Důležité pojmy z oblasti grafů .....	15
1.3 Typy grafů .....	15
1.4 Operace s datovou strukturou graf .....	17
1.5 Typické grafové operace .....	17
<b>2 Grafové databáze .....</b>	<b>22</b>
2.1 Charakteristika grafových databází .....	22
2.2 Neo4j .....	24
2.3 OrientDB .....	25
2.4 FlockDB .....	26
2.5 AllegroGraph .....	27
2.6 Titan .....	28
2.7 Souhrnný přehled grafových databází .....	29
<b>3 Reprezentace grafových struktur v relačních databázích .....</b>	<b>30</b>
3.1 Typy technik .....	30
3.2 Existující řešení .....	30
3.2.1 DB2 Spatial Extender .....	30
3.2.2 PostGIS .....	31
3.2.3 Oracle Spatial .....	32
3.3 Oracle Network Data Model .....	33
3.3.1 Datový model .....	34
3.3.2 Vytváření grafu .....	36
3.3.3 Další grafové operace .....	37
3.3.4 Java API pro Oracle Network Data Model .....	38
3.4 Souhrnný přehled vybraných řešení .....	38
<b>4 Návrh a implementace aplikace pro testování grafových operací .....</b>	<b>40</b>

4.1 Požadavky na systém.....	40
4.2 Obecný návrh.....	40
4.2.1 Grafické rozhraní.....	40
4.2.2 Generátor grafů.....	41
4.3 Implementace testovacího systému.....	41
4.3.1 Použité technologie.....	41
4.3.2 Propojení NetBeans a Neo4j.....	42
4.3.3 Propojení NetBeans a Oracle Spatial.....	42
4.3.4 Adresářová struktura.....	42
4.3.5 Grafické prostředí.....	43
4.3.6 Generátor grafů.....	45
4.3.7 Převodník grafů z Neo4j do Oracle Spatial.....	46
4.4 Před prvním spuštěním testovací aplikace.....	47
4.5 Ovládání aplikace.....	47
<b>5 Testování.....</b>	<b>48</b>
5.1 Zadávání testovacích scénářů .....	48
5.2 Testovací data .....	49
5.3 Hledání nejkratší cesty mezi dvěma uzly .....	49
5.3.1 Spuštěné testy pro Dijkstrův algoritmus z Java API rozhraní obou databází .....	50
5.3.2 Vyhodnocení naměřených údajů .....	54
5.3.3 Spuštěné testy pro vlastní implementaci Dijkstrova algoritmu .....	55
5.3.4 Vyhodnocení naměřených údajů .....	59
5.4 Úloha obchodního cestujícího .....	60
5.4.1 Spuštěné testy pro vlastní implementaci 2aproximačního algoritmu .....	60
5.4.2 Vyhodnocení naměřených údajů .....	64
5.5 Hledání minimální kostry grafu .....	65
5.5.1 Spuštěné testy pro vlastní implementaci Kruskalova algoritmu .....	65
5.5.2 Vyhodnocení naměřených údajů .....	69
<b>Závěr .....</b>	<b>70</b>
<b>Použité zdroje.....</b>	<b>71</b>
<b>Příloha A – kompaktní disk obsahující zdrojové kódy aplikace.....</b>	<b>73</b>



## Seznam zkratek

ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CPU	Central Proccessing Unit
CRUD	Create, Read, Update and Delete
DDR	Double Data Rate
E-R	Entity-Relationship
ERD	Entity-relationship diagram
GeoTIFF	Georeferenced Tag Image File Format
GIMP	GNU Image Manipulation Program
GIS	Geographic information system
GNU	rekurzivní zkratka „GNU 's Not Unix!“
GPL	General Public License
GPS	Global Positioning System
HTML	HyperText Markup Language
JDBC	Java Database Connectivity
JPG	Joint Photographic Experts Group
MVRB	Multi Value Red Black
NASA	National Aeronautics and Space Administration
NetCDF	Network Common Data Form
NOSQL	Not Only SQL (nebo také no SQL)
OCI	Oracle Call Interface
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PDF	Portable Document Format
PNG	Portable Network Graphics
RAM	Random-Access Memory
RB	Red Black
RDF	Resource Description Framework
REST	Representational State Transfer
SATA	Serial ATA (Advanced Technology Attachment)
SPARQL	rekurzivní zkratka „SPARQL Protocol And RDF Query Language“
SQL	Structured Query Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

## Seznam obrázků

Obrázek 1 - Použití grafu pro zobrazení rodinných vztahů .....	14
Obrázek 2 – Použití grafu pro znázornění úlohy Sedm mostů města Královce .....	15
Obrázek 3 – Graf znázorňující dopravní komunikaci.....	16
Obrázek 4 – Hledání nejkratší cesty pomocí Dijkstrova algoritmu.....	18
Obrázek 5 – Řešení úlohy obchodního cestujícího pomocí zařazovacího algoritmu .....	20
Obrázek 6 – Minimální kostra grafu určená Kruskalovým algoritmem .....	21
Obrázek 7 – E-R diagram relační databáze vztahů osob .....	30
Obrázek 8 – Vztah komponent v Oracle Network Data Model [3] .....	34
Obrázek 9 – Grafické rozvržení aplikace.....	41
Obrázek 10 – Adresářová struktura .....	43
Obrázek 11 – Grafické prostředí pro testování Neo4j databáze.....	44
Obrázek 12 – Grafické prostředí pro testování Oracle Spatial databáze .....	45
Obrázek 13 – Graf doby trvání řešení Dijkstrova algoritmu z Java API pro graf s 5 000 uzly.....	50
Obrázek 14 – Graf využití RAM a vytížení CPU v průběhu řešení Dijkstrova algoritmu z Java API pro graf s 5 000 uzly .....	51
Obrázek 15 – Graf doby trvání řešení Dijkstrova algoritmu z Java API pro graf se 7 500 uzly.....	52
Obrázek 16 – Graf využití RAM a vytížení CPU v průběhu řešení Dijkstrova algoritmu z Java API pro graf se 7 500 uzly.....	52
Obrázek 17 – Graf doby trvání řešení Dijkstrova algoritmu z Java API pro graf s 10 000 uzly.....	53
Obrázek 18 – Graf využití RAM a vytížení CPU v průběhu řešení Dijkstrova algoritmu z Java API pro graf s 10 000 uzly .....	54
Obrázek 19 – Graf doby trvání řešení vlastní implementace Dijkstrova algoritmu pro graf s 5 000 uzly .....	55
Obrázek 20 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Dijkstrova algoritmu pro graf s 5 000 uzly .....	56
Obrázek 21 – Graf doby trvání řešení vlastní implementace Dijkstrova algoritmu pro graf se 7 500 uzly .....	57
Obrázek 22 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Dijkstrova algoritmu pro graf se 7 500 uzly .....	57
Obrázek 23 – Graf doby trvání řešení vlastní implementace Dijkstrova algoritmu pro graf s 10 000 uzly .....	58
Obrázek 24 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Dijkstrova algoritmu pro graf s 10 000 uzly .....	59
Obrázek 25 – Graf doby trvání řešení vlastní implementace 2aproximačního algoritmu pro graf s 50 uzly .....	60
Obrázek 26 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace 2aproximačního algoritmu pro graf s 50 uzly .....	61

Obrázek 27 – Graf doby trvání řešení vlastní implementace 2aproximačního algoritmu pro graf se 100 uzly.....	62
Obrázek 28 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace 2aproximačního algoritmu pro graf se 100 uzly.....	62
Obrázek 29 – Graf doby trvání řešení vlastní implementace 2aproximačního algoritmu pro graf se 150 uzly.....	63
Obrázek 30 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace 2aproximačního algoritmu pro graf se 150 uzly.....	64
Obrázek 31 – Graf doby trvání řešení vlastní implementace Kruskalova algoritmu pro graf s 500 uzly .....	65
Obrázek 32 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Kruskalova algoritmu pro graf s 500 uzly.....	66
Obrázek 33 – Graf doby trvání řešení vlastní implementace Kruskalova algoritmu pro graf se 750 uzly.....	67
Obrázek 34 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Kruskalova algoritmu pro graf se 750 uzly .....	67
Obrázek 35 – Graf doby trvání řešení vlastní implementace Kruskalova algoritmu pro graf s 1 000 uzly.....	68
Obrázek 36 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Kruskalova algoritmu pro graf s 1 000 uzly.....	69

## Seznam tabulek

Tabulka 1 – Přehled grafových databází.....	29
Tabulka 2 – Přehled prostorových rozšíření relačních databází.....	39
Tabulka 3 – Konfigurace přenosného počítače použitého při testování .....	48
Tabulka 4 – Vypočítané hodnoty statistických proměnných pro graf s 5 000 uzly při řešení úlohy Dijkstrovým algoritmem z Java API.....	51
Tabulka 5 – Vypočítané hodnoty statistických proměnných pro graf se 7 500 uzly při řešení úlohy Dijkstrovým algoritmem z Java API.....	53
Tabulka 6 – Vypočítané hodnoty statistických proměnných pro graf s 10 000 uzly při řešení úlohy Dijkstrovým algoritmem z Java API.....	54
Tabulka 7 – Vypočítané hodnoty statistických proměnných pro graf s 5 000 uzly při řešení úlohy vlastní implementací Dijkstrova algoritmu .....	56
Tabulka 8 – Vypočítané hodnoty statistických proměnných pro graf se 7 500 uzly při řešení úlohy vlastní implementací Dijkstrova algoritmu .....	58
Tabulka 9 – Vypočítané hodnoty statistických proměnných pro graf s 10 000 uzly při řešení úlohy vlastní implementací Dijkstrova algoritmu .....	59
Tabulka 10 – Vypočítané hodnoty statistických proměnných pro graf s 50 uzly při řešení úlohy vlastní implementací 2aproximačního algoritmu .....	61
Tabulka 11 – Vypočítané hodnoty statistických proměnných pro graf se 100 uzly při řešení úlohy vlastní implementací 2aproximačního algoritmu .....	63

Tabulka 12 – Vypočítané hodnoty statistických proměnných pro graf se 150 uzly při řešení úlohy vlastní implementací 2aproximačního algoritmu .....	64
Tabulka 13 – Vypočítané hodnoty statistických proměnných pro graf s 500 uzly při řešení úlohy vlastní implementací Kruskalova algoritmu .....	66
Tabulka 14 – Vypočítané hodnoty statistických proměnných pro graf se 750 uzly při řešení úlohy vlastní implementací Kruskalova algoritmu .....	68
Tabulka 15 – Vypočítané hodnoty statistických proměnných pro graf s 1 000 uzly při řešení úlohy vlastní implementací Kruskalova algoritmu .....	69

## Úvod

V dnešním světě databázových systémů lze stále častěji sledovat vývoj nových typů databází, jejichž datový model již není založen na relačním modelu dat. Tato diplomová práce se zabývá grafovými databázemi, jejichž datový model je odvozen od datové struktury grafu. Dá se říci, že každý databázový systém disponuje určitými přednostmi. Tyto přednosti lze využít při řešení konkrétních problémů. Je proto možné položit si otázku, do jaké míry lze nahradit určitý typ databázového systému jiným typem.

Grafové databáze jsou v současné době všude kolem nás a to zejména v oblasti webových aplikací. Tyto databáze vynikají především svou flexibilitou, vysokým výkonem při práci s propojenými daty a nenáročností na pochopení proti relačním databázím. Mezi typické úlohy, které velmi efektivně řeší grafové databáze, patří hledání maximálního průtoku v síti (vodní, elektrické, datové), hledání nejkratší cesty z bodu A do bodu B (GPS navigace) a hledání minimální či maximální kostry grafu (obchodní doporučovací systémy). Mnoho lidí ani nemá tušení, v jakých oblastech jsou dnes grafové databáze používány. Příkladem mohou být sociální sítě (Twitter, Facebook), systém správy dat (Cisco), geografické systémy, neurochirurgické systémy a doporučovací systémy.

Tato práce si klade dva základní cíle. Za prvé prozkoumat problematiku současných grafových databází a jejich alternativních reprezentací v relačním světě. Za druhé otestovat výkon čistě grafové databáze Neo4j, v porovnání s výkonem relační databáze Oracle, při práci nad grafem.

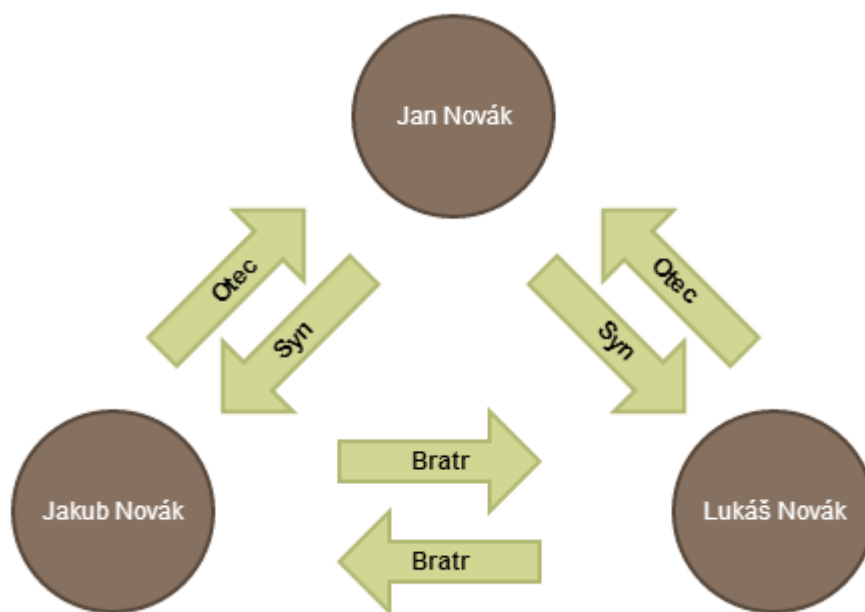
V rámci diplomové práce se čtenář nejprve seznámí se základními pojmy teorie grafů včetně principů některých algoritmů. Dále dojde k představení vybraných grafových databází a jejich nejdůležitějších vlastností. Pro porozumění uchování grafových struktur v relačních databázích budou představeny základní techniky a někteří zástupci databázových systémů, zabývajících se danou oblastí. Dále dojde k uvedení požadavků na testovací aplikaci a samotnou implementaci. Nakonec budou obě databáze testovány při řešení vybraných algoritmů, což umožní porovnání jejich výkonu.

# 1 Graf

Pokud se mluví o tématu grafových databází, je nejprve nutné vysvětlit pojem graf. Grafy se zabývá teorie grafů, což je matematická disciplína.

## 1.1 Co je graf

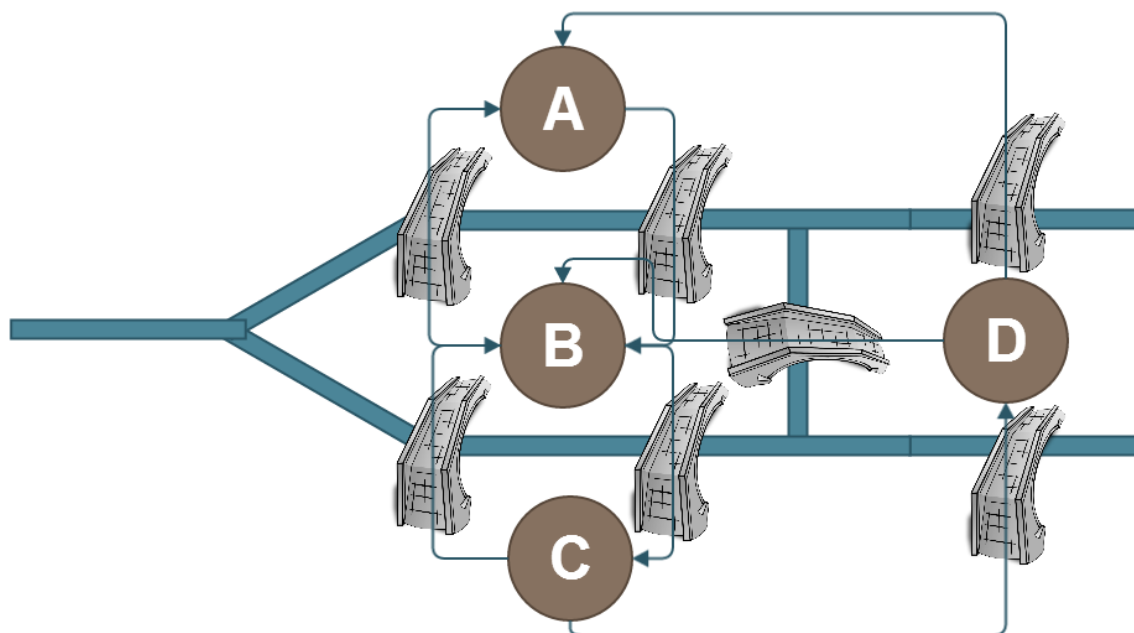
V tomto případě není správné mluvit o grafu ve smyslu grafické interpretace určitých výsledků, ale o grafu jako datové struktuře. Mezi prvky datové struktury graf patří vrcholy (někdy též uzly) a hrany, které spojují právě dva uzly (jež mohou být totožné). Hrany mohou být orientované a neorientované. Výhodou grafu je velká obecnost, pojednání o tom, co přesně znázorňují vrcholy a hrany je určeno až konkrétním řešeným problémem. Vrcholy zpravidla vykreslují libovolné objekty. Hrany slouží k zobrazení jakýchkoliv vztahů mezi objekty. Popsané prvky můžeme vidět na následujícím příkladě – zobrazení rodinných vztahů na obrázku 1. Jako vrcholy jsou zde zobrazeny konkrétní osoby. U vrcholů si lze také všimnout vlastnosti označení představující jméno člověka. Hrany znázorňují vztahy mezi osobami. U těchto hran lze vidět, že mají směr a označení, které popisuje vztah mezi dvěma konkrétními lidmi.



Obrázek 1 - Použití grafu pro zobrazení rodinných vztahů

V praxi se grafy využívají zpravidla při modelování a studiích různých oblastí (v informatice, dopravě, logistice a dalších). Mezi nejznámější grafové problémy patří hledání nejkratší cesty mezi dvěma uzly, které se může řešit například pomocí Dijkstrova algoritmu. Historicky první grafovou úlohu – Sedm mostů města Královce (Königsberg) – řešil pomocí grafů matematik Leonhard Euler. Vyobrazení tohoto problému je možné vidět na obrázku 2. Úkolem bylo zjistit, zda lze všechny mosty přejít tak, abychom každý použili

pouze jednou. Leonhard Euler pomocí zjednodušeného zobrazení úlohy jako grafu dokázal, že to možné není.



Obrázek 2 – Použití grafu pro znázornění úlohy Sedm mostů města Královce

## 1.2 Důležité pojmy z oblasti grafů

V této kapitole popíšeme vybrané důležité grafové pojmy, nutné k pochopení dále popisovaných algoritmů. Jedná se o následující:

- 1) kružnice – označuje takový graf, který se skládá z jediné uzavřené posloupnosti propojených vrcholů (příklad lze najít na obrázku 1, pokud je tento graf procházen jedním vybraným směrem);
- 2) Hamiltonovská kružnice – označuje prohlídku grafu, kdy každý jeho uzel je navštíven právě jednou s výjimkou uzlu výchozího, který je rovněž uzlem cílovým;
- 3) kostra grafu – označuje jakoukoliv cestu, která spojuje všechny uzly v grafu.

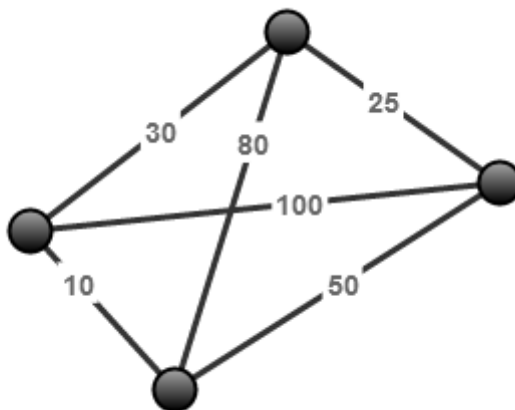
## 1.3 Typy grafů

To, jaký typ grafu bude zpracováván, má dopad na volbu použité datové struktury k znázornění grafu. A také tím dojde k zúžení množiny algoritmů dostupných pro analýzu grafu. Nyní budou popsány základní typy grafů. Grafy lze dělit na základě několika kritérií. Mezi základní hlediska dělení patří následující:

- orientovanost grafu – lze rozdělit na orientovaný (je takový graf, jehož hrany mají označený směr, což umožňuje určit výchozí a cílový vrchol hrany, příklady je

možné vidět na obrázcích 1 a 2) a neorientovaný graf (je takový graf, jehož hrany nemají vyznačený směr, příkladem je graf na obrázku 3);

- váženost grafu – je možné rozdělit na ohodnocený (každé hraně či vrcholu je přiřazena číselná hodnota, která může mít například význam délky cesty jako na obrázku 3) a neohodnocený graf (hranám ani vrcholům nejsou přiřazeny žádné hodnoty, nejkratší cestou je na takovém grafu nejmenší počet hran mezi dvěma vrcholy, příklad neohodnoceného grafu lze vidět na obrázku 2);
- hustota grafu – lze rozdělit na hustý (takový graf, kde jsou jakékoliv dva vrcholy mezi sebou spojeny hranou, jak je možné vidět na obrázku 3) a řídký graf (kde nemusí být jakékoliv dva vrcholy mezi sebou spojeny hranou, příklad je k nalezení na obrázku 2);
- označenost grafu – je možné rozdělit na označený (každému vrcholu je přiřazen jedinečný název nebo identifikátor, pomocí kterého dojde k odlišení od ostatních vrcholů jako na obrázku 2) a neoznačený graf (neexistuje žádné jedinečné označení vrcholů – obrázek 3).
- úplnost grafu – lze rozdělit na úplný (každé 2 vrcholy v grafu jsou spojeny hranou, jak je možné vidět na obrázku 1 a 3) a neúplný (všechny dvojice vrcholů v grafu nejsou spojeny hranou – viz obrázek 2, kde vrcholy A a C nejsou spojeny hranou);



**Obrázek 3 – Graf znázorňující dopravní komunikaci**

Z hlediska fyzické reprezentace lze grafy rozdělit do 4 skupin:

- 1) vrcholově statické struktury – operace související se vkládáním a odebráním vrcholů, nejsou realizovatelné, nebo je třeba při jejich provádění prohledat průměrně  $n$  (počet vrcholů grafu) vrcholů;
- 2) vrcholově dynamické struktury – tyto typy grafů jsou vhodné pro dynamickou práci vzhledem k jejich vrcholům;



- 3) hranově statické struktury - operace související se vkládáním a odebíráním hran, nejsou realizovatelné, nebo je třeba při jejich provádění prohledat průměrně  $m$  (počet hran grafu) hran;
- 4) hranově dynamické struktury – tyto typy grafů jsou vhodné pro dynamickou práci vzhledem k jejich hranám.

#### 1.4 Operace s datovou strukturou graf

Mezi základní grafové operace lze zařadit všechny úkony týkající se vytváření, úpravy a prohlídky grafu. Pro všechny datové struktury jsou společné operace vytvoření, zrušení, dotaz na prázdnotu a počet prvků ve struktuře. U většiny ostatních operací se využívá parametr klíč. Klíč slouží k jednoznačnému určení prvku v grafu. Nyní budou uvedeny a stručně popsány základní operace:

- prohlídka grafu (vstupními parametry jsou typ, počátek a akce) – z typů lze vybrat buď prohlídku vrcholovou, nebo hranovou a pro dále je možné zvolit prohlídku do šířky nebo do hloubky; prohlídka zajistí návštěvu všech prvků zvoleného typu (uzel či hrana) v grafu; jako počátek prohlídky lze zvolit libovolný prvek;
- vložení prvku (vstupním parametrem je vrchol či hrana) – slouží pro vkládání vrcholu či hrany do grafu;
- odebrání prvku (vstupním parametrem je vrchol či hrana) – slouží pro odebrání určitého vrcholu či hrany z grafu;
- nalezení prvku (vstupním parametrem je klíč pro nalezení prvku a výstupním parametrem nalezený vrchol či hrana) – slouží k vyhledání vrcholu či hrany na základě zadaného klíče;
- zpřístupnění následníků nebo předchůdců (vstupním parametrem je prvek, ke kterému se operace vztahuje, výstupem je množina prvků následníků či předchůdců) – tyto operace lze provádět pouze nad orientovaným grafem;
- zpřístupnění incidenčních prvků (vstupním parametrem je prvek, ke kterému se operace vztahuje, výstupem je množina incidenčních prvků) – tato operace nalezne všechny prvky přímo navazující na daný prvek (incidenční prvky).

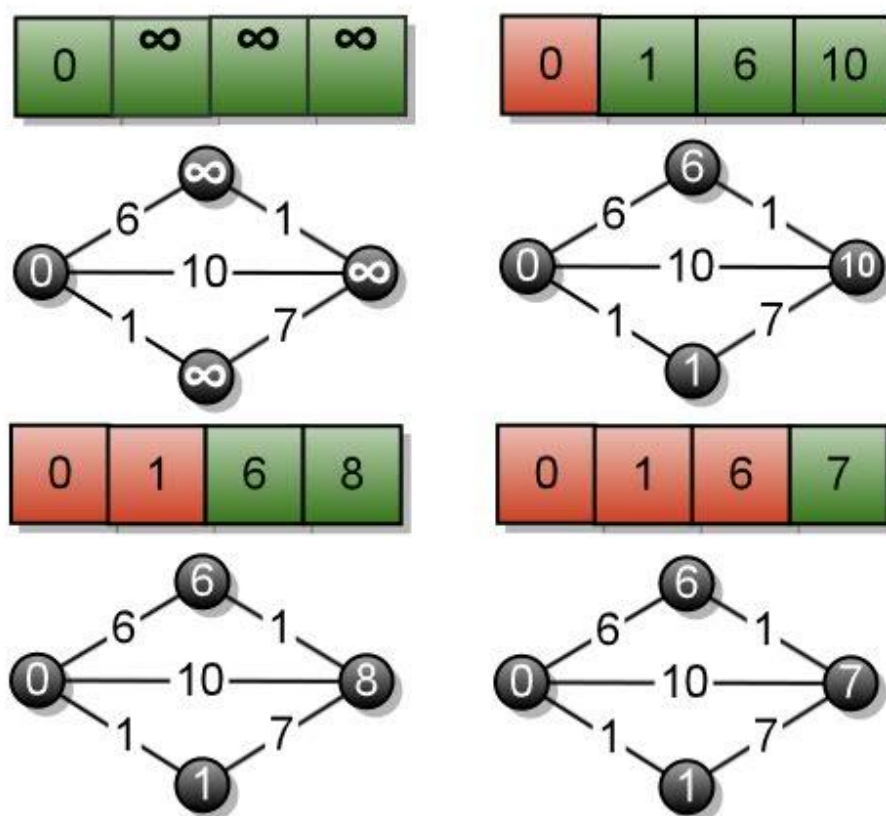
#### 1.5 Typické grafové operace

V oblasti grafů lze najít kromě základních operací manipulujících s datovou strukturou graf i další často používané operace. V této kapitole budou uvedeny 3 vybrané.

Prvním a zároveň nejčastěji řešeným problémem v oblasti grafů je nalezení nejkratší cesty mezi dvěma uzly hranově ohodnoceného grafu. Hledání nejkratší cesty v grafu mohou komplikovat pouze výskyty cyklů záporné délky. Pokud však takový případ nenastane, je možné úlohu řešit v polynomiálním čase pomocí různých algoritmů. Výhodou je, že

většina praktických úloh hrany záporné délky neobsahuje (např.: nalezení nejkratší cesty mezi 2 městy). Pro příklad lze uvést 3 základní algoritmy řešící problém nalezení nejkratší cesty:

- 1) Dijkstrův (viz obrázek 4) – jedná se o nejefektivnější algoritmus pro vyhledání nejkratší cesty ze zvoleného uzlu do všech ostatních v grafu; Dijkstrův algoritmus pracuje nad prioritní frontou; v každém kroku dochází k výběru prvku s nejvyšší prioritou (nejmenší vzdáleností od startovního uzlu); v prvním kroku má počáteční uzel ohodnocení 0 a všechny ostatní nekonečno; dále začne zpracovávat všechny potomky vybraného uzlu (kteří již nebyli zpracováni v některém z předchozích kroků), tak že zjistí, zda jejich ohodnocení není menší než hodnota aktuálního uzlu plus ohodnocení hrany vedoucí k danému potomkovi; pokud tak nastane, dojde k přehodnocení uzlu v prioritní frontě a jejímu přeuspořádání; algoritmus končí v okamžiku vyprázdnění prioritní fronty; jediným omezením algoritmu je možnost použití pouze pro grafy obsahující výhradně hrany s kladným ohodnocením;



Obrázek 4 – Hledání nejkratší cesty pomocí Dijkstrova algoritmu

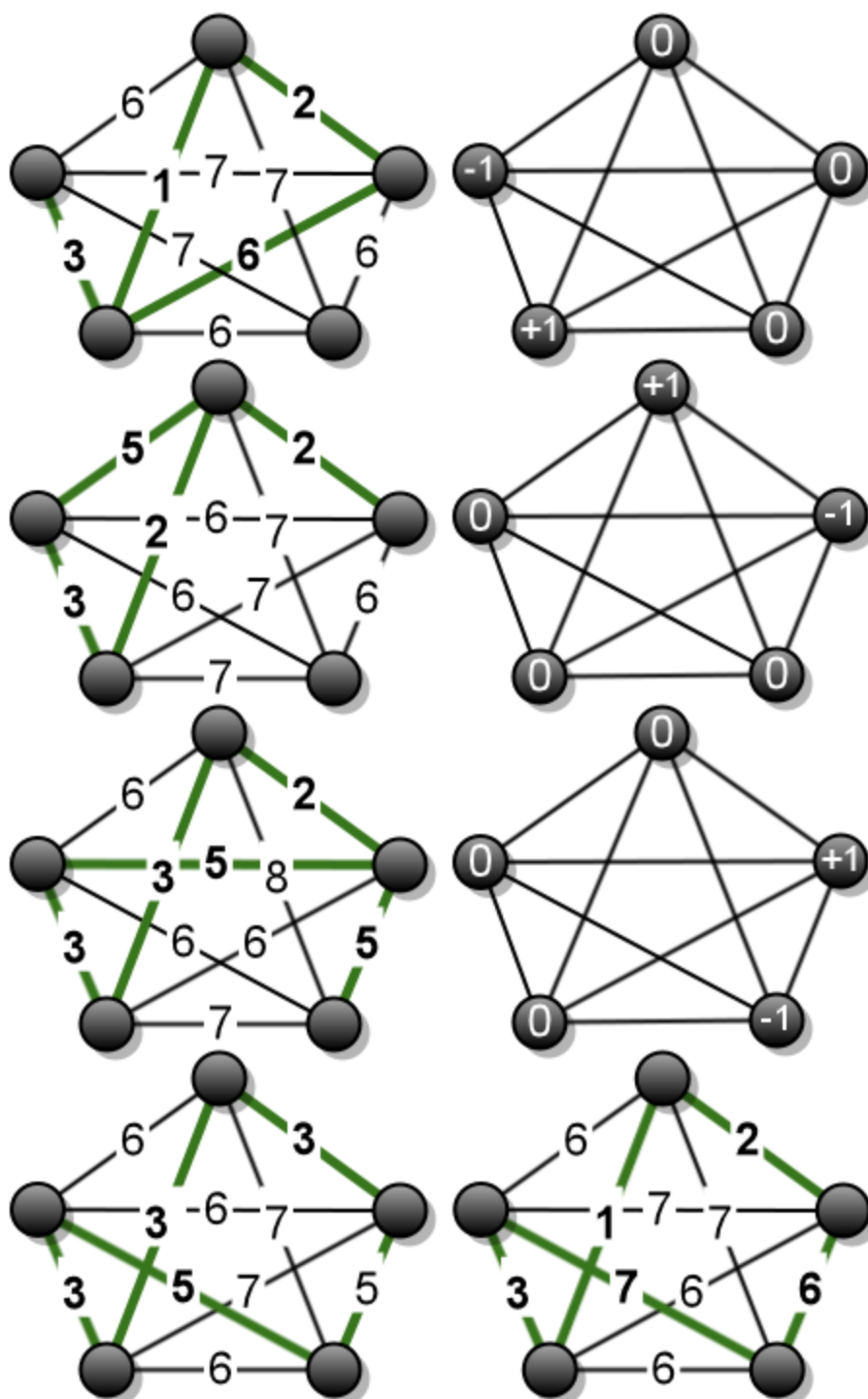
- 2) Bellman-Fordův – základní operací pro tento algoritmus je relaxace; během ní dochází ke zpracování dvou uzlů a hrany, která je spojuje; pokud je součet vzdálenosti zdrojového uzlu a hrany menší než cílového, pak se vzdálenost cílového přepočítá a jako předchůdce cílového uzlu se označí zdrojový uzel; v opačném případě se nic nestane; pokud dojde k provedení relaxace hran p (počet

uzlů v grafu minus 1) krát, pak musí být nalezeny všechny nejkratší cesty; lze ověřit existenci cyklu záporné délky tím, že dojde k opakovanému spuštění relaxace pro všechny hrany; pokud kterákoliv z relaxací změní ohodnocení nějakého uzlu, pak graf obsahuje cyklus záporné délky;

- 3) Floyd-Warshallův – základem tohoto algoritmu je matice délek; v matici na pozicích  $(i, j)$  se vždy nachází vzdálenost mezi dvěma uzly čísla  $i$  a  $j$ ; tato matice má na začátku na diagonále nuly; na pozicích, kde existuje přímá cesta má ohodnocení hran mezi uzly a na pozicích, kde se nachází mezi dvěma uzly ještě jiné pak hodnotu nekonečno; v následujících krocích se matice přepočítává, jak postupně dochází ke zpřístupnění dalších uzlů; hlavní výhoda tohoto algoritmu je implementační jednoduchost, v podstatě se jedná o tři vnořené cykly; pro zjištění cesty mezi dvěma uzly je nutné vytvořit matici předchůdců; pokud je nutné zjistit cestu mezi uzly  $i$  a  $j$ , pak dojde k přesunutí na pozici  $(i, j)$  v matici předchůdců, kde se nachází označení předchůdce uzlu na hledané cestě; tento postup je nutné opakovat, dokud nebude předchůdce uzlu  $i$ .

Prakticky stejně se řeší i problém opačný – nalezení nejdelší cesty – kde stačí pouze otočit znaménka ohodnocení u všech hran grafu.

Nyní bude představena další grafová úloha. Frekventovaným problémem při práci s úplným hranově ohodnoceným grafem je Úloha obchodního cestujícího (anglicky Travelling salesman problem). Cílem je najít nejkratší cestu pro obchodníka cestujícího z určitého místa v grafu. Jeho úkolem je navštívit zadaná obchodní místa (vrcholy grafu) právě jednou a nakonec se vrátit na místo odkud vyjel. Tento problém souvisí s hamiltonovskou kružnicí. Úloha obchodního cestujícího je v praxi řešena především heuristickými algoritmy. Nevýhoda těchto řešení je v tom, že většina algoritmů dává buď přibližné řešení, nebo může skončit neúspěchem. Jedním z algoritmů, který může skončit neúspěchem je i algoritmus hledající podgraf s minimálním ohodnocením, jež má právě jedna hledaná kružnice ze všech kružnic v grafu. V praxi lze použít například zařazovací algoritmus (viz obrázek 5). Jedná se o heuristický algoritmus, který může skončit neúspěchem. Nejdříve jsou hrany seřazeny v neklesající posloupnosti, dále je vybrán diskrétní podgraf celého grafu a postupně jsou přidávány hrany s nejnižším ohodnocením. A to tak, aby nevznikla více než jedna kružnice. Pokud všechny hrany tvoří jednu kružnici, pak je v původním grafu nalezena kružnice s nejmenším ohodnocením. Pokud k tomuto nedojde, jsou upraveny hodnoty hran. Provede se snížení ohodnocení o 1 u všech hran podgrafu, které sousedí s vrcholem stupně 1. Současně je zvýšeno ohodnocení o 1 u všech hran sousedících s vrcholem stupně větší než 2. Dále budou opakovány předchozí kroky, dokud není nalezena hamiltonovská kružnice. Pokud výpočet trvá neúměrně dlouho, pak algoritmus pro konkrétní graf nedokáže nalézt řešení a uživatel musí jeho činnost ukončit sám. [2]

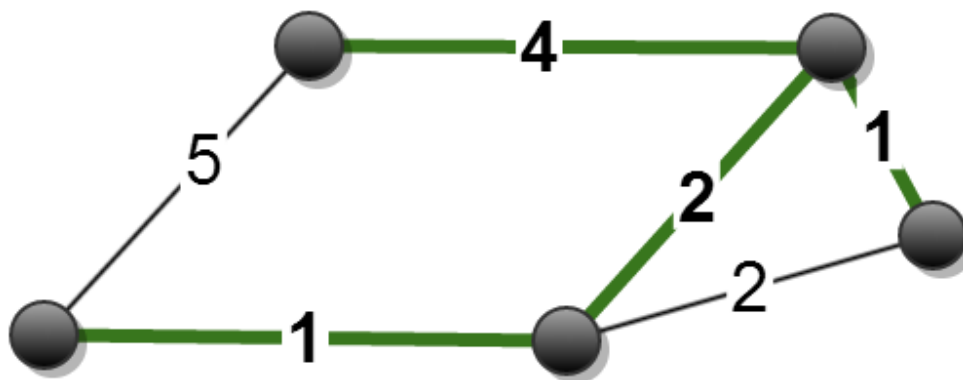


Obrázek 5 – Řešení úlohy obchodního cestujícího pomocí zařazovacího algoritmu

Jako zástupce algoritmů, které dávají přibližné řešení Úlohy obchodního cestujícího, bude popsán 2aproximační algoritmus. V prvním kroku je zkonstruována minimální kostra grafu (viz obrázek 6). Dále se provede průchod grafem do hloubky pouze nad hranami minimální kostry. Ze seznamu vrcholů prohlídky budou vyřazeny opakující se vrcholy. Tudíž je ponechán pouze jejich první výskyt v tomto seznamu. Zbýlé vrcholy určují řešení Úlohy

obchodního cestujícího. Hodnota cesty oproti optimálnímu řešení se maximálně zdvojnásobí, pokud v grafu platí trojúhelníková nerovnost. [17]

Jako poslední bude uveden problém nalezení minimální kostry na hranově ohodnoceném grafu. Cílem je popsat, jak spojit všechny vrcholy grafu pomocí hran s nejmenším ohodnocením (viz obrázek 6). V praxi lze tímto typem úloh řešit například problém propojení měst nejkratší cestou elektrického vedení. Jednou z možností, jak řešit úlohu nalezení minimální kostry grafu je Kruskalův algoritmus, který bude následně popsán. Na začátku jsou všechny hrany grafu seřazeny vzestupně, dle jejich ohodnocení. Za druhé budou vybrány hrany s nejmenším ohodnocením ze seřazeného seznamu. Pokud při přidání hrany vznikne v grafu kružnice, hranu je zahozena a použita další v pořadí. Druhý krok je opakován, dokud vyznačené hrany nevytvoří kostru grafu. [18]



Obrázek 6 – Minimální kostra grafu určená Kruskalovým algoritmem

## 2 Grafové databáze

Jak už název napovídá, grafová databáze uchovává datovou strukturu graf a umožňuje s ní i dále pracovat.

### 2.1 Charakteristika grafových databází

Databáze zabývající se grafy lze rozdělit na dvě hlavní oblasti:

- 1) grafové databáze – technologie používané zejména jako transakční on-line grafové úložiště (OLTP), obvykle přímo přístupné z aplikace běžící v reálném čase;
- 2) grafové výpočetní stroje – technologie používané především pro off-line grafovou analýzu, typicky prováděná jako série kroků v dávce; tyto výpočetní stroje je možné zařadit do stejné kategorie jako jiné technologie pro analýzu velkého objemu dat a on-line analytického zpracování (OLAP).

Dalším způsobem dělení grafového prostoru je pohled na grafové modely. Z tohoto hlediska existují 3 druhy grafových modelů:

- 1) graf vlastností – obsahuje uzly a vztahy mezi uzly; uzly i vztahy mají vlastnosti, které jsou tvořeny dvojicí klíč - hodnota; vztahy jsou pojmenované a mají směr (určen počáteční a koncový uzel); v grafu vlastností mohou mít uzly i vztahy označení (tzv. label), pomocí kterého je lze rozřadit do skupin;
- 2) hypergrafy – jedná se o zobecněný model grafu, ve kterém lze jedním vztahem propojit libovolný počet uzlů; hypergrafy jsou užitečné především tam, kde je požadováno zaznamenat vazby m:n (např.: pokud dva lidé vlastní tři stejné televizory); ovšem při použití hypergrafů dochází ke ztrátě určitého detailu ve vztazích (např.: oba lidé vlastní televizor, ale ve svém pokoji ho má pouze jeden a převážně ho používá); lze tedy říci, že hypergrafy jsou obecnějším modelem grafu než grafy vlastností;
- 3) triples úložiště (trojice) – označují se také jako datová struktura subjekt-predikát-objekt; triples používají jako úložiště RDF dokumenty; tento typ úložiště byl navrhnut pro získávání informací z internetu a predikování dalších znalostí na základě již získaných; ovšem tato myšlenka předpokládala, že RDF statementy budou obsaženy ve standardních HTML stránkách, kde by popisovaly obsah stránky formou srozumitelnou strojům; triples nejsou nativními grafovými databázemi, protože neposkytují přirozený index sousednosti; úložiště triples ukládá poznatky jako samostatné trojice, což nám při ukládání umožňuje horizontálně škálovat, ale brání nám v rychlém procházení vztahů; pro dotazování nad RDF dokumenty slouží dotazovací jazyk SPARQL.

Grafové databáze slouží jako nejlepší úložiště pro specifický druh dat, jež jsou označovány jako propojená data. Taková data vyžadují, abychom nejdříve pochopili, jakým způsobem

jsou propojená. Je tedy nutné pojmenovat spojení mezi ukládanými objekty. Grafové databáze patří mezi on-line systémy správy dat s CRUD operacemi, které odhalují datový model grafu. Obvykle jsou tvořeny pro použití s transakčními systémy (OLTP). Grafové databáze se tedy optimalizují s ohledem na transakční výkon. Důležitým kritériem při jejich vytváření je provozní dostupnost a udržení transakční integrity. Předtím než budou představeny konkrétní grafové databáze, můžeme uvést jednu důležitou vlastnost. Některé grafové databáze používají jako nativní úložiště graf, který je optimalizován pro ukládání a správu grafů. Ovšem grafové databáze mohou také využívat jako nativní úložiště relační, objektovou či databázi jiného typu. V takovém případě dochází při přístupu do takových úložišť k serializaci grafových dat. Mezi nejsilnější argumenty pro přechod na grafové databáze patří tyto tři výhody:

- 1) vysoký výkon – při práci s propojenými daty se obrovsky zvýší výkon oproti použití relačních nebo NOSQL databází; zatímco u relačních databází se při dotazování nad propojenými daty zhoršuje výkon při nárůstu dat v datovém úložišti, u grafových databází zůstává výkon dotazů relativně konstantní; tato výhoda vzniká díky lokalizaci dotazů (grafová databáze prochází pouze část dat); doba provádění dotazu je tedy úměrná velikosti zkoumané oblasti dat v grafu;
- 2) velká flexibilita datového modelu – data jsou spojována dle domény, ve které se pracuje; oproti jiným databázovým technologiím, které vyžadují před započítím modelování znalost problematiky, grafové databáze se snaží dát prostor vývojáři k postupnému objevování a pochopení problémů struktury a schéma modelované problematiky; grafy jsou přirozeně aditivní, což znamená, že je možné přidávat nové druhy vztahů, uzlů a nové podgrafy do existující struktury bez narušení správnosti existujících dotazů či funkčnosti nadstavbové aplikace; z důvodu flexibility grafového modelu není nutné předem dopodrobna modelovat zpracovávanou doménu; následkem aditivní povahy grafů je provádění méně změn v běžící databázi, čímž se sníží náklady na údržbu;
- 3) dobrá obratnost modelu – jedná se o schopnost vyvíjet datový model po krocích společně se zbytkem aplikace, v souladu s moderními iterativními a přírůstkovými softwarovými metodami; grafové databáze postrádají jakékoliv schéma, chybí jim tedy určitý druh schematicky orientovaných mechanismů pro správu dat, který je známý z relačního světa; tato vlastnost grafových databází není považována za nevýhodu, ale umožní nám bezprostřední a průhlednější řízení dat; pro agilní a testově řízený vývoj softwaru se dnes více hodí grafové než relační databáze, agilní vývoj nám dovoluje vrátit se do kteréhokoliv kroku vývoje aplikace při změně požadavků v byznysovém prostředí.

Pro shrnutí tedy lze říci, že nativní grafové databáze nejsou do značné míry závislé na indexech. Graf totiž sám o sobě poskytuje přirozený index sousednosti. V nativní grafové databázi je možné pomocí přímého spojení procházet mezi sousedními uzly. Lze tedy v takovém grafu navštívit až miliony uzlů za sekundu na rozdíl od spojování dat přes

globální index, který je o několik řádů pomalejší. Grafové databáze jsou dnes jasnou volbou, při vytváření sociálních sítí, doporučujících systémů, systémů správy dat, geografických systémů, neurochirurgických systémů a dalších. [1]

## 2.2 Neo4j

Typickým zástupcem grafových databází je Neo4j. Datovým modelem této databáze je graf vlastností. Jedná se o databázi s volitelným použitím datového schématu. V Neo4j je tedy možné pracovat bez jakéhokoli schématu. Od verze 2.0 lze nepovinně zavést použití datového schématu pro zlepšení výkonu, či z důvodu snadnějšího modelování. Realizace této volby nám usnadňuje práci po celou dobu, kdy je využívána jedna z uvedených výhod, aniž by nás schéma jakkoliv omezovalo. Je možné vybrat si ze dvou druhů schémat:

- 1) indexy – vytvořením indexu zvýšíme výkon při jakémkoliv prohledávání uzlů v databázi; jakmile je jednou určena, která vlastnost uzlu se má indexovat Neo4j bude s rozvojem grafu udržovat tento index aktuální; výkon každé operace, která bude vyhledávat uzly pomocí indexovaných vlastností, se rapidně zvýší; indexy ovšem nejsou ihned po zadání příkazu pro vytvoření k dispozici; po zadání příkazu je sice ovládání ihned vráceno uživateli, ovšem index se teprve začne vytvářet v pozadí běhu databáze; jakmile dojde k dokončení procesu tvorby indexu, Neo4j ho začne využívat při dotazování; pokud se v průběhu vytváření indexu cokoliv pokazí, databáze ho pro zrychlení dotazů nepoužívá; takový index lze odstranit a znovu vytvořit; záznam o chybách při vytváření se objeví v log souboru databáze; další možností jak sledovat korektnost stavu indexů je pomocí API rozhraní databáze, v současné verzi Neo4j není možné sledovat stav indexů pomocí dotazovacího jazyku databáze;
- 2) omezení – omezení určují pomocí podmínek, jak by měla data v databázi vypadat; jakákoliv změna, která by porušila předepsaný stav je zakázána; díky omezením dokáže Neo4j pomoci uživateli udržet databázi čistší; výhodou je, že se o všechna omezení nemusí starat aplikace; v současné verzi je k dispozici pouze omezení na jedinečné hodnoty.

Neo4j je nativní grafová databáze. Pro ukládání dat tedy využívá datovou strukturu graf. Mezi její další vlastnosti patří spolehlivost, plná podpora ACID (všechny transakce splňují požadavky na atomicitu, konzistenci dat, izolovanost a trvalost provedených změn). Jedná se o odolnou a rychlou databázi. Dále je rozsáhle škálovatelná, až na několik miliard uzlů, vztahů či vlastností. Při distribuci na více počítačů má Neo4j vysokou dostupnost. Při práci s touto grafovou databází lze využívat dva dotazovací jazyky:

- 1) Cypher – je navržen, aby byl dobře čitelný a jednoduchý na pochopení pro vývojáře, databázové specialisty a obchodníků účastnících se databázových projektů; jeho jednoduchost vyplývá ze skutečnosti, že funguje v souladu s intuitivním způsobem popisu grafů pomocí diagramů; dotazování probíhá formou žádostí na databázi o návrat dat splňujících předepsaný vzor; jedná se tedy o



deklarativní dotazovací jazyk; zápis vzoru, podle kterého budou nalezena data, jsou uvedena pomocí ASCII artu;

- 2) Gremlin – tento dotazovací jazyk byl navržen pro grafové databáze, které pracují s grafem vlastností; ovšem může také pracovat s grafy uloženými v RDF souborech; Gremlin poskytuje základní operace s grafem (CRUD), dále zajišťuje transakční integritu; lze pomocí něj objevovat strukturu grafu, se kterým je pracováno a dále tento graf analyzovat; Gremlin pracuje v krocích (tzv. steps); každý krok je ve skutečnosti odkaz na tzv. pipeline (potrubí); potrubí jsou tvořeny spojením tzv. pipe (rour); pomocí roury lze transformovat data na vstupu různými způsoby (rozdělit, sloučit, filtrovat, opakovat určitou operaci nad daty či definovat vlastní operace); spojené roury jsou zpracovávány sekvenčně, výstup z jedné roury slouží jako vstup do další roury v pořadí; Gremlin je velmi efektivní při práci nad nativní grafovou databází; velkou výhodou je možnost definovat vlastní rozšíření jazyka přímo v Gremlinu, či v Java API; tento dotazovací jazyk lze ovládat pomocí konzole, přes OrientDB Studio nebo Java API (Gremlin poskytuje podporu pro programovací jazyk Java); vývoj tohoto programovacího jazyka zajišťuje komunita jménem Tinkerpop. [6]

Neo4j pracuje s grafem vlastností. Všechny vztahy jsou v této grafové databázi orientované. Procházení grafu v Neo4j znamená navštívit uzly a vztahy propojující uzly. Ve většině případů je prozkoumána pouze část grafu (tzv. subgraf), ve které se nacházejí pro nás zajímavé uzly a vztahy. Na základní úrovni grafu lze volit mezi prohlídkou do šířky či do hloubky. Do Neo4j lze snadno přistupovat přes pohodlné rozhraní REST nebo objektově orientovaného Java API rozhraní. Neo4j je grafová databáze vytvořená a vyvíjená společností Neo Technology v jazyce Java. Jedná se o Open Source databázi pro nekomerční účely nebo firmy do 3 zaměstnanců a ročního zisku 100 000 \$. Pro firmy s větším ziskem nebo více než 3 zaměstnanci jsou dostupné placené verze s profesionální podporou. V současnosti představuje nejpoužívanější grafovou databázi na světě. Mezi největší firmy využívající Neo4j patří Cisco, Adobe, HP a Huawei. Nejnovější vydání Neo4j nese označení 2.0.

## 2.3 OrientDB

OrientDB nelze striktně zařadit mezi grafové databáze. Tato databáze disponuje charakteristikami grafových i dokumentových databází. OrientDB je v podstatě dokumentová databáze, ve které jsou vztahy realizovány přímým spojením mezi záznamy stejně jako v grafové databázi. Grafy uložené v databázi jsou zpracovávány nativně (řídí se standardem společnosti TinkerPop Blueprints pro grafové databáze). V OrientDB je možné pracovat bez datového schématu, s datovým schématem či v kombinovaném režimu. Tato grafová databáze je orientovaná na rychlost, dědí nejlepší vlastnosti a koncepty z dokumentových, grafových a objektových databází. Jedná se o databázi s velkou rychlostí zápisu (pokud běží na běžném počítači, dokáže uložit až 150 000 záznamů za sekundu). Nespornou předností je možnost projít část, či celý graf v několika málo

milisekundách. Další výhodou OrientDB je silný zabezpečovací systém, založený na přidělování oprávnění konkrétním uživatelům či na základě rolí. Dále podporuje kromě jiných dotazovacích jazyků i SQL s určitými rozšířeními, což představuje velkou výhodu pro uživatele přecházející z relačních databází. OrientDB využívá pro vkládání a vyhledávání uzlů vlastní MVRB-Tree algoritmus. Jedná se o kombinaci RB-Tree a B+Tree algoritmu. MVRB-Tree algoritmus dovoluje uložit několik hodnot namísto jedné v každém uzlu vyhledávacího stromu. Výsledkem je snížení času na polovinu u operací vkládání nebo aktualizace uzlů při využití RB-Tree algoritmu. Dále MVRB-Tree umožňuje rychlé vyhledávání uzlů. Důležitou vlastností OrientDB je podpora ACID transakcí. Pokud je u OrientDB třeba zvětšit průchodnost sítě, stačí přidat nový server, který se připojí na stávající distribuovaný serverový klastr. Pokud u OrientDB dojde k selhání jednoho uzlu serveru, serverový cluster přerozdělí zatížení napříč dostupnými uzly. Všichni klienti napojení na porouchaný uzel jsou přepnuti na jiný dostupný uzel serveru. OrientDB je naprogramovaná v jazyku Java, jedná se tedy o platformově nezávislou databázi. Požadavkem Light verze databáze na paměť je pouze 1 MB pro server. Další výhodou je nezávislost OrientDB na externích knihovnách či softwaru třetích stran. OrientDB vyvíjí firma Orient Technologies jako Open Source projekt. [10]

## 2.4 FlockDB

Jedná se o distribuovanou grafovou databázi, která ukládá orientovaný graf ve formě seznamů sousednosti. Při ukládání hran (neboli vztahů) zaznamenává FlockDB 4 informace. Zapisuje unikátní ID zdrojového a cílového uzlu, dále pozici vztahu (slouží pro řazení vztahů v dotazu, používá se např. časové razítko) a stav vztahu (používán pro označení hrany, zda má být zachována, odstraněna či archivována). Vztahy jsou ukládány pro oba směry průchodu vztahu (např. A následuje B, B je následováno A). Tato technika je používána z důvodu rychlosti získávání vztahů mezi uzly. Pro vztahy platí, že zdrojový a cílový uzel je neměnný, ovšem pozici a stav vztahu, lze kdykoliv upravit. Pro FlockDB je typická podpora:

- 1) velmi frekventovaných operací vkládání, odebírání a úpravy prvků grafu;
- 2) potenciálně složitých výpočetních dotazů;
- 3) stránkování výsledných množin dotazů obsahujících miliony záznamů;
- 4) „archivace“ hran grafu a jejich pozdější obnova;
- 5) horizontální škálování včetně replikací;
- 6) online migrace dat;
- 7) rezistence proti chybám v databázi.

Naopak FlockDB primárně nepodporuje:

- 1) dotazy pro průchod velkou částí grafu;

- 2) automatickou migraci dat pro horizontální škálování.

FlockDB se nesnaží řešit tolik problémů jako jiné grafové databáze (např. Neo4j). Hlavním záměrem databáze je horizontální škálování dat, poskytnutí nízké latence operací uživatelům a zajištění vysoké propustnosti prostředí. Proto se FlockDB hodí především pro online webové systémy. Jedná se o Open Source databázi vyvinutou společností Twitter. Sociální síť Twitter používá FlockDB pro ukládání uživatelů jako uzlů a řízení vztahů mezi nimi (kdo koho následuje, kdo koho blokuje, kdo si co oblíbil). Pro představu, v dubnu roku 2010 FlockDB cluster Twitteru obsahoval přes 13 miliard hran a dokázal obstarat přes 20 tisíc zápisů do databáze a 100 tisíc čtení z databáze za sekundu. [7] FlockDB poskytuje API pro programovací jazyky Scala, Java a Ruby. Za nevýhodu lze považovat nutnost nejprve nainstalovat externí knihovny pro chod databáze – sbt a thrift. [8]

## 2.5 AllegroGraph

Grafový model AllegroGraph je triples úložiště, ukládá tedy propojená data do RDF souborů. Každá informace je uložena ve formě triples (trojice), subjekt-predikát-objekt. Návrh databáze byl uzpůsoben, aby splňoval standardy W3C pro Resource Description Framework. RDF soubory disponují elegantním formalismem pro popis grafů. Tyto soubory lze použít i pro uložení klasických relačních databází, stromů či jiných informací popsateľných v XML. Mezi hlavní vlastnosti AllegroGraph databáze patří:

- 1) vysoký výkon – databáze byla navržena pro maximální rychlost zpracování dotazů;
- 2) trvalost databáze;
- 3) podpora ACID transakcí;
- 4) úplná a rychlá obnova databáze do stavu před záchytným bodem;
- 5) úplná podpora souběžného čtení více uživateli;
- 6) téměř úplná podpora souběžného zápisu více uživateli;
- 7) online zálohování, návrat do bodu obnovy v čase, replikace, rychlé uvedení do pohotovostního režimu;
- 8) uživatelsky definované, dynamické nebo automatické indexování všech nově potvrzených záznamů;
- 9) pokročilé fulltextové indexování;
- 10) optimalizované využití paměti – automatické řízení zdrojů pro všechny procesory a disky z důvodu maximalizace výkonu databáze;
- 11) trojí úroveň zabezpečení díky bezpečnostním filtrům.

AllegroGraph je výborná při využití jako OLTP webová databáze. Pro prohlídku grafu poskytuje AllegroGraph vlastní grafický prohlížeč s názvem Gruff. AllegroGraph je poskytován zdarma pouze s omezením na počet uložených trojic, komerční verze nemá žádná omezení. Jako dotazovací jazyk lze použít SPARQL, RDFS++, Prolog a dotazy skrze programovací API. AllegroGraph implementuje statický a dynamický analyzátor dotazu. Statický analyzátor zjistí před zpracováním dotazu, jaké indexy se použijí. Dynamický analyzátor zkoumá až při zpracovávání dotazu, jaké indexy se použijí. Dynamický analyzátor poskytuje lepší informace, zároveň je však pomalejší než statický. Dotazování do databáze pomocí SPARQL probíhá na základě načítání dat z úložiště dle požadovaného vzoru (podobně jako Cypher). Pro všechny trojice odpovídající vzoru vytvoří dotaz odkaz na všechny proměnné trojice. AllegroGraph poskytuje API pro programovací jazyky Java, Python, Ruby, Perl, C#, Clojure and Common Lisp. O vývoj této grafové databáze se stará firma Franz, Inc., která se proslavila vytvořením programovacího jazyku Lisp. Mezi klienty používající AllegroGraph patří např.: Americká armáda, Citi bank, Novartis, NASA, General Electric, Ford a Adobe. [9]

## 2.6 Titan

Pro grafy, které vyžadují diskový prostor a výpočetní kapacitu nad rámec zvládnutelnosti jedním databázovým strojem, slouží grafová databáze Titan. Jedná se tedy o distribuovanou databázi, která se z pohledu uživatele chová jako jeden databázový stroj. Ve skutečnosti se jedná o několik počítačů propojených do clusteru. Z důvodu růstu velikosti dat a uživatelské základny podporuje Titan pružné škálování. Dále tato grafová databáze poskytuje architekturu připojitelných indexů (ElasticSearch a Lucene indexů). Datovým modelem této databáze je graf vlastností. Pro Titan jsou typické tyto vlastnosti:






- 1) podpora velkého počtu souběžných transakcí;
- 2) transakční kapacita Titanu je škálovatelná dle počtu počítačů v clusteru;
- 3) distribuce a replikace dat z důvodu většího výkonu a lepší odolnosti vůči chybám;
- 4) podpora ACID transakcí;
- 5) podpora pro geografické, či fulltextové vyhledávání nad velkým počtem vrcholů a hran;
- 6) nativní podpora průchodu grafem pomocí jazyka Gremlin;
- 7) mnoho možností konfigurací na grafové úrovni pro zlepšení výkonu databáze;
- 8) z důvodu eliminace problému super uzlů poskytuje vrcholově orientované indexování grafu;
- 9) optimalizovaná disková reprezentace z hlediska rychlého přístupu k datům a efektivního využití úložiště.

Velkou výhodou Titanu je rozšíření množiny předností databáze díky využití jedné z podkladové administrační databáze Cassandra nebo HBase. Jedná se o Open Source databázi s možností využití placené podpor, kterou vyvíjí společnost Aurelius. Na základě standardizace Blueprints podporuje generické API pro programovací jazyk Java. [11]

## 2.7 Souhrnný přehled grafových databází

V této kapitole je pro přehlednost uvedena tabulka srovnání představených grafových databází.

**Tabulka 1 – Přehled grafových databází**

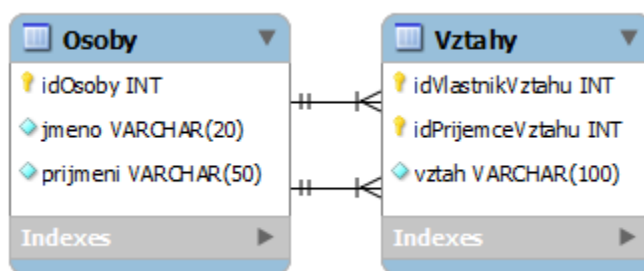
Grafová databáze	Neo4j	OrientDB	FlockDB	AllegroGraph	Titan
Datový model	Graf vlastností	Graf vlastností	Seznamy sousednosti	Triples	Graf vlastností
Dotazovací jazyk	Cypher, Gremlin, programovací API	Gremlin, modifikované SQL, programovací API	programovací API	SPARQL, RDF++, API	Gremlin, programovací API
Oblast zaměření databáze	Grafově orientované úlohy	Orientace na databáze s rychlým zápisem a propracovaným zabezpečením	Orientace na vysokou propustnost a nízkou latenci databáze	Pro systémy náročné na výkon databáze	Ukládání velkých objemů dat
Transakční podpora	Plná podpora ACID transakcí	Plná podpora ACID transakcí	Nepodporuje ACID transakce	Podpora ACID transakcí	Podpora ACID transakcí
Zvláštní specifikace	Vysoká spolehlivost	Využití MVRB-Tree algoritmu	Ukládání každého vztahu pro oba směry průchodu	Poskytuje analyzátor dotazů	Pouze distribuovaná databáze
Licence	Open Source	Open Source	Open Source	Closed Source	Open Source
Logo					

### 3 Reprezentace grafových struktur v relačních databázích

Relační databáze organizují data do tabulek. Práce s tabulkami je dnes pro většinu lidí velmi intuitivní. Grafové databáze poskytují více volnosti v tom, jaká data budou uložena v jednotlivých uzlech a hranách. V této kapitole je tedy zkoumána otázka: jak nejlépe uchovat velmi obecná data grafové databáze v tabulkách relační databáze.

#### 3.1 Typy technik

Nejjednodušším způsobem, jak ukládat grafová data do relační databáze, je vytvoření dvou tabulek. Lze hovořit o čistě relační reprezentaci. Zároveň se jedná o řešení, které má nejvíce nevýhod ze všech dále uvedených řešení. První tabulka této databáze bude obsahovat uzly grafu a druhá jeho hrany. Nevýhodou je samozřejmě malá flexibilita úložiště a časově náročné dotazování při procházení grafem. Malá flexibilita se projeví, například když jsou postupně ukládány uzly či hrany a každý nový prvek má novou vlastnost. Pokaždé je tedy třeba upravit strukturu tabulky uzlů či hran. Na obrázku 7 je možné vidět příklad jednoduché databáze vztahů osob v relační databázi (typicky uchovávané v grafové databázi).



Obrázek 7 – E-R diagram relační databáze vztahů osob

#### 3.2 Existující řešení

V relačních databázích lze lépe pracovat s grafovými strukturami především díky různým rozšířením. V této kapitole budou přestavena některá z těchto rozšíření.

##### 3.2.1 DB2 Spatial Extender

Jako první je uvedeno rozšíření Spatial Extender pro relační databázový systém DB2 od IBM. Jedná se o placenou Closed Source databázi. DB2 Spatial Extender slouží k ukládání a správě geografických prvků (cokoliv v reálném světě u čeho lze určit polohu umístění). Umožňuje také analyzovat a generovat nové prostorové informace o těchto prvcích. Využívá se především v kombinaci s firemními daty pro lepší rozhodování (např. o umístění kancelářských budov, vymezení velikosti záplavové zóny apod.). DB2 Spatial Extender implementuje typy a funkce definované ISO SQL/MM (SQL Multimedia and Application Packages) a Open Geospatial konsorciem. Tato databáze navíc disponuje možností instalace bezplatného geoprohlížeče, který umožňuje zobrazit informace volitelnou symbolikou z libovolné tabulky s prostorovými informacemi z databáze. DB2

Spatial Extender se snaží především o rozšíření základních informací o objektech o prostorovou informaci, typicky souřadnice objektu, formou odvozeného sloupečku. V praxi se u vybraných objektů v databázi zaznamenává informace o jejich poloze v určitém souřadnicovém systému, či formou adresy objektu. Ovšem na základě adresy lze získat pomocí geocodingu nebo GPS lokace zeměpisnou šířku a délku objektu. Ukládání prostorových dat v DB2 Spatial Extender typicky probíhá následujícím způsobem:

- 1) volba zdroje dat – z již existujících dat je vybrán soubor informací, na základě kterých bude vytvářen sloupeček s prostorovými daty;
- 2) transformace dat – souřadnice, případně adresu transformuje pomocí připravených funkcí databáze do některého z mezinárodních systémů souřadnic; vznikne nám tedy údaj obsahující zeměpisnou délku a šířku ve vybraném souřadnicovém systému;
- 3) zavedení nové informace – transformovaný údaj je přidán buď jako nový sloupeček do existující tabulky ke konkrétnímu záznamu, případně lze vytvořit novou tabulku obsahující odkaz na konkrétní záznam společně s jeho souřadnicemi;
- 4) registrace v systémovém katalogu – tento krok je nutný pro zpřístupnění prostorových informací propojovacím klientům GIS systémů; jsou využity tabulky `St_Geometry_Columns` a `ST_Spatial_Reference_Systems`; do první z nich je zaznamenána existence nového sloupečku s prostorovými informacemi, dále bude zadán název tabulky, jméno sloupečku, datový typ, ve kterém je informace uložena (typicky `ST_POINT`) a ID zvoleného mezinárodního systému souřadnic dle tabulky `ST_Spatial_Reference_Systems` (uloženy všechny poskytované souřadnicové systémy).

Samotný DB2 Spatial Extender neposkytuje pokročilé analytické prostorové funkce, avšak umožňuje propojit databázi s pokročilými GIS systémy, které tyto funkce přinášejí. Toto rozšíření také umožňuje vytvoření vlastních analytických funkcí díky poskytovaným základním geografickým funkcím (např. určení vzdálenosti mezi dvěma souřadnicemi). Veškeré funkce rozšíření databáze jsou přístupné skrz volání balíčku s názvem `db2gse`. [15]

### 3.2.2 PostGIS

Mezi představitele rozšíření svobodných databází pod licencí GNU GPL lze zařadit PostGIS pro objektově-relační databázi PostgreSQL. Jak již název napovídá, v tomto rozšíření je možné pracovat s geometrickými objekty, není zde ovšem možné vytvářet logické celky (např. sítě). PostGIS disponuje kromě základních funkcionalit databáze PostgreSQL schopností pracovat s geometrickými, geografickými, rastrovými a dalšími typy objektů. Další výhodou jsou také funkce, operátory a indexová vylepšení vztahující se k rozšiřujícím prostorovým typům. PostGIS rovněž poskytuje vlastní pokročilé importní a exportní nástroje prostorových a geometrických dat z různých standardizovaných formátů souborů (GeoTiff, NetCDF, PNG, JPG a dalších). Velkým

přínosem je také implementace standardů Simple Features vytvářených konsorciem Open Geospatial. [12] Pro naše účely je nejzajímavější podpora práce se sítěmi (neboli grafy), kterou zajišťuje především modul pgRouting. Tato složka PostGIS poskytuje rovněž funkce pro standardní grafovou analýzu pouze nad geometrickými sítěmi (např. nalezení nejkratší cesty, řešení problému obchodního cestujícího a další). Vytváření samotné sítě probíhá v několika krocích:

- 1) konstrukce tabulky hran – tato tabulka musí povinně obsahovat několik sloupečků (ID hrany, geometrie hrany, ID počátečního uzlu, ID koncového uzlu, ohodnocení hrany);
- 2) tabulku hran naplněna daty;
- 3) použití funkce `pgr_createTopology` – základní verze této funkce vyžaduje pouze 2 parametry (název tabulky hran, vzdálenost tolerance se kterou se zjišťuje protínání hran); předpokladem použití základní verze funkce je předepsané pojmenování sloupečků geometrie hrany, ID hrany, ID počátečního uzlu a ID koncového uzlu; pokud funkce ukončí činnost správně, vrátí hodnotu TRUE; dojde k vytvoření a naplnění tabulky uzlů na základě průniků hran; tabulka uzlů je automaticky pojmenována jako „název tabulky hran“ + „\_vertices\_pgr“; každý záznam této tabulky bude obsahovat 6 údajů (ID uzlu, počet hran odkazujících se na tento vrchol, indikátor možného problému s vytvořeným vrcholem, počet hran odkazujících se na vrchol jako počáteční, počet hran odkazujících se na vrchol jako koncový a geometrie vrcholu); v tabulce hran dojde k doplnění hodnoty ID počátečního a koncového uzlu; funkce nám také vytvoří indexy nad sloupcečky ID hrany, geometrie hrany, ID počátečního a koncového uzlu z důvodu rychlejšího přístupu k informacím.

Na vytvořené síti lze pomocí funkce `pgr_analyzeGraph` analyzovat správnosti údajů prvků sítě. Mezi hlavní výhody pgRouting patří:

- 1) možnost modifikovat data a atributy různými klienty – QGIS, JDBC, ODBC a přímo přes pgSQL;
- 2) změny v datech se projevují okamžitě, nedochází k žádnému předkalkulování;
- 3) náklady na průchod prvkem sítě lze vypočítat dynamicky pomocí SQL navíc zdrojová data pro výpočet mohou pocházet z více polí či tabulek.

V době tvorby diplomové práce bylo nejnovější verzí PostGIS vydání 2.0.5 a modulu pgRouting 2.0. [13]

### 3.2.3 Oracle Spatial

K rozšířením relačních databází patří také Oracle Spatial řešení. Tento modul je dostupný pouze v placené Enterprise edici Oracle databáze. Jedná se především o množinu funkcí a



procedur pro usnadnění práce s prostorovými daty. Jednou z částí tohoto modulu je i Oracle Network Data Model, který slouží pro práci s grafy. [14]

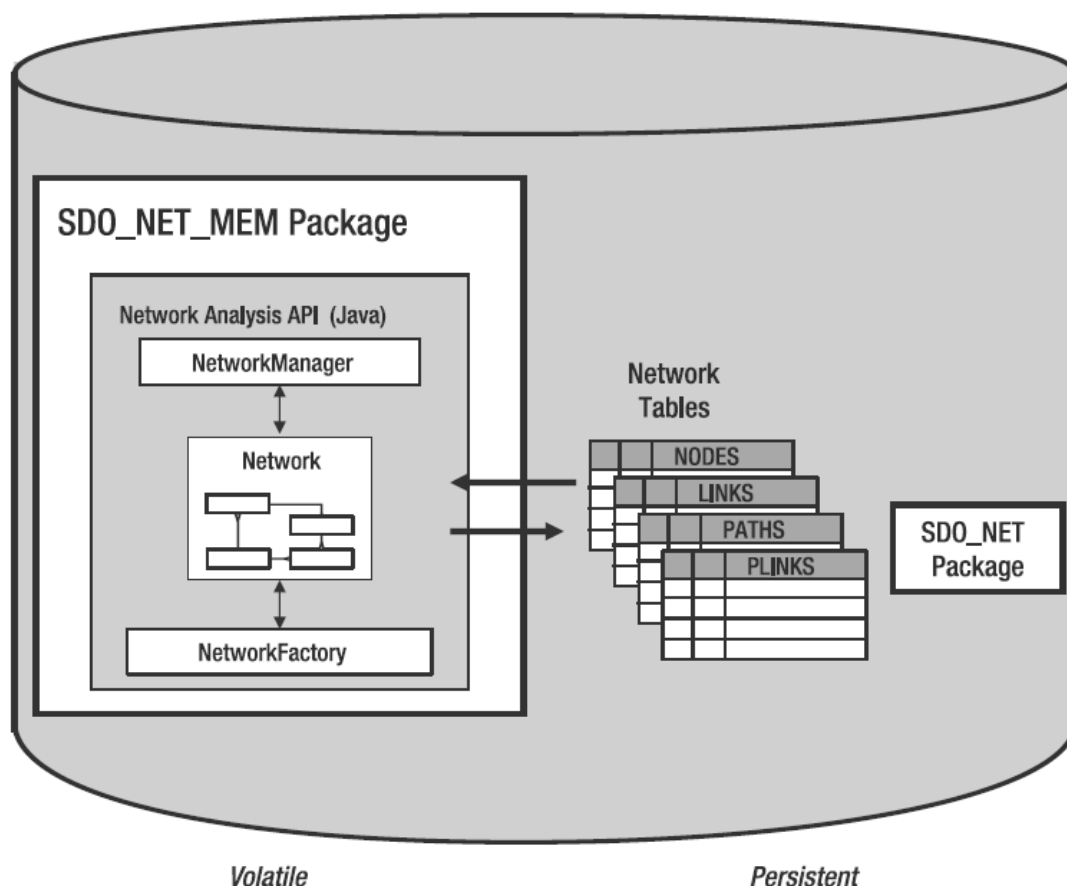
### 3.3 Oracle Network Data Model

Jedná se o rozšíření Oracle databází, dále bude toto rozšíření popsáno pro verzi databáze Oracle 11g. Jak už název napovídá, jedním z hlavních úkolů této nadstavby bylo pokrýt oblast modelování cest a křižovatek. Oracle se proto v souvislosti s tímto modulem nezmiňuje o grafech, ale říká, že jeho cílem je podpora vytváření sítí. Proto bude dále v textu pro graf používáno i označení síť. Umožňuje zaznamenávat kromě orientovanosti a ohodnocení i geometrické údaje všech prvků grafu. Zajímavostí je možnost ukládat i jakékoliv cesty v grafu ve speciální tabulce. Cesta je v podstatě sekvence uzlů a hran mezi dvěma uzly grafu, přičemž mezi těmito dvěma uzly může vést i více cest. Cesty mohou být jednoduché či složité. U jednoduchých platí, že každá hrana v seznamu hran je při průchodu cestou navštívena pouze jednou. Další předností je možnost záznamu více typů ohodnocení nad jedním prvkem grafu (např. délka cesty, čas průjezdu cestou a kvalita cesty). V danou chvíli lze však použít pouze jedno vybrané ohodnocení. V Oracle Network Data Model lze také využívat různá omezení, týkající se vlastností prvků v grafu (např. cesty sjízdné pouze pro osobní automobily).

Toto rozšíření zabývající se grafovými databázemi se skládá ze čtyř komponent:

- 1) datový model – tvořený množinou tabulek sloužících pro permanentní ukládání grafů do databáze;
- 2) SDO\_NET – balíček SQL funkcí pro definici a údržbu grafů;
- 3) Java API funkce – určené k analýze grafů (sítí); tyto funkce pracují nad dočasnou kopií grafu získanou z permanentní relační databáze; výsledky analýzy (např. vypočtená nejkratší cesta) a také změny grafu mohou být zapsány zpět do trvalého úložiště grafu v relační databázi;
- 4) SDO\_NET\_MEM – balíček PL/SQL funkcí a procedur určených k analýze grafů; důležité ovšem je, že tento balíček využívá Java API rozhraní, které pracuje nad dočasně vytvořeným grafem, uloženým v paměti virtuálního stroje Javy.

Vzájemné vztahy popsaných komponent je možné vidět na obrázku 8.



Obrázek 8 – Vztah komponent v Oracle Network Data Model [3]

### 3.3.1 Datový model

Nyní bude podrobněji popsána složka datového modelu. Jakýkoli graf v Oracle Network Data Model je obvykle tvořen 5 tabulkami, přičemž funkční graf musí tvořit alespoň 3 z nich. Pojmenování jednotlivých tabulek je libovolné. Ovšem musí dodržovat stanovenou strukturu. Je nutné, aby obsahovali minimální počet určitých sloupců, přičemž některé sloupce musí být označeny předdefinovaným výrazem. Mezi výhody patří možnost použít pohledy místo tabulek při vytváření grafu. Nyní budou všechny tabulky popsány podrobněji:

- 1) nodes (uzly) – tato tabulka uchovává informace o všech uzlech v síti; každý uzel má své unikátní ID uchovávané v povinném sloupečku s názvem `NODE_ID` (toto pojmenování není volitelné), zároveň se jedná o primární klíč tabulky; další sloupce v této tabulce není nutné vytvářet a jsou tedy volitelné; prostorové sítě využívají sloupeček typu `SDO_GEOMETRY` určující souřadnice uzlu, jeho název je volitelný; dalším parametrem v tabulce uzlů je číselný údaj ohodnocení uzlu, určuje hodnotu uzlu a jeho název je volitelný; v případě neuvedení sloupce ohodnocení má každý uzel hodnotu 0; další dva sloupce se využívají v hierarchických sítích; první určuje úroveň uzlu v hierarchii a musí se označovat jako `HIERARCHY_LEVEL`; druhý určuje ID rodičovského uzlu, jeho povinné pojmenování je `PARENT_NODE_ID`; funkce pro síťovou analýzu využívají

sloupeček s povinným názvem ACTIVE, který může nabývat hodnot 'Y', nebo 'N'; při vykonávání síťové analýzy nám určuje, zda je uzel aktivní a lze ho tedy do ní zahrnout; v případě neuvedení tohoto sloupečku v tabulce uzlů je každý uzel aktivní; poslední dva sloupečky s povinným označením NODE\_NAME a NODE\_TYPE slouží pro pojmenování uzlu a jeho typu, přičemž tyto vlastnosti jsou libovolné; tabulka uzlů může dále obsahovat jiné libovolné sloupečky;

- 2) links (hrany) – tato tabulka uchovává informace o všech hranách v síti; primárním klíčem je sloupec LINK\_ID jehož pojmenování není volitelné, označuje číslo (ID) hrany v síti; dvojice sloupců START\_NODE\_ID a END\_NODE\_ID určují ID uzlu, ze kterého hrana vychází a do kterého uzlu ústí; pojmenování těchto sloupečků je předepsané; další sloupce v této tabulce není nutné vytvářet a jsou tedy volitelné; prostorové sítě využívají sloupeček typu SDO\_GEOMETRY určující tvar hrany, jeho název je volitelný; dalším parametrem v tabulce hran je číselný údaj ohodnocení hrany, určuje cenu průchodu hranou a jeho název je volitelný; navíc je možné vytvořit více sloupců ohodnocení hrany, ovšem funkce síťové analýzy budou používat pro své potřeby sloupec ohodnocení uvedený v tabulce metadat; v případě neuvedení sloupce ohodnocení má každá hrana ohodnocení 1; pro orientované sítě je používán sloupeček BIDIRECTED; určuje nám, zda je hrana průchozí oběma směry; může nabývat hodnot 'Y', nebo 'N'; pokud je síť orientovaná a zároveň není tento sloupeček definován, pak jsou všechny hrany průchozí pouze směrem z počátečního do koncového uzlu; funkce pro síťovou analýzu využívají sloupeček s povinným názvem ACTIVE, který může nabývat hodnot 'Y', nebo 'N'; při vykonávání síťové analýzy nám určuje, zda je hrana aktivní (viditelná pro uživatele) a lze jí tedy do této analýzy zahrnout; v případě neuvedení tohoto sloupečku je každá hrana aktivní; v hierarchických sítích je využíván sloupec s daným pojmenováním PARENT\_LINK\_ID, slouží k určení rodičovské hrany; pomocí sloupečku LINK\_LEVEL s pevně stanoveným pojmenováním lze určit prioritu hrany; poslední dva sloupečky s povinným označením LINK\_NAME a LINK\_TYPE slouží pro pojmenování hrany a jejího typu, přičemž tato jména hran jsou libovolné; tabulka hran může dále obsahovat jiné libovolné sloupečky;
- 3) metadata (název pohledu nad tabulkou metadat je USER\_SDO\_NETWORK\_METADATA) – tuto tabulku uživatel databáze nevytváří, ale pouze do ní přidává nové sítě, případně edituje údaje o síti; v této tabulce lze nalézt názvy všech vytvořených grafů; záznam o každé síti obsahuje 20 údajů; je možné zde zvolit název sítě (řetězec o maximální délce 24 znaků), unikátní ID sítě; zde lze také určit typ sítě; dále zde nastane možnost rozhodnutí, zda se bude jednat o čistě logickou, či geometrickou síť; volí se také typ použité geometrie (SDO\_GEOMETRY, LRS\_GEOMETRY nebo TOPO\_GEOMETRY); dále se označí počet hierarchických úrovní sítě (při zadání 1 nebude síť hierarchická); při rozdělení sítě na více oddílů je nám umožněno zadat počet oddělených částí a název tabulky uchovávající informaci o rozdělení sítě; dále je

možné zvolit orientovanost hran v síti; samozřejmě je také nezbytné zadat názvy tabulek uzlů, hran, cest a hrany cest; dále je nám v metadatech umožněno specifikovat názvy sloupců z tabulek uzlů a hran, které slouží jako sloupce ohodnocení těchto prvků; tabulka metadat slouží rovněž pro pojmenování sloupců určujících geometrii uzlů, hran a cest; díky této tabulce je také možné nad jednou skupinou tabulek postavit více různých sítí s různými sloupečky ohodnocení prvků (např. v jedné síti může jako ohodnocení hran sloužit jejich délka, v jiné čas průchodu hranou);

- 4) paths (cesty) – tabulky cesty a hrany cest jsou volitelné a používají se pouze, pokud je požadováno zaznamenat cesty vypočtené síťovou analýzou (většinou pomocí Java API funkcí síťové analýzy); slouží tedy k možnému rychlejšímu zjišťování nejkratší cesty mezi dvěma uzly; 5 následujících sloupečků této tabulky patří mezi povinné, navíc musí uživatel zachovat jejich přesné pojmenování; PATH\_ID určuje unikátní číslo zaznamenané cesty, zároveň se jedná o primární klíč tabulky; sloupeček START\_NODE\_ID slouží pro označení ID počátečního uzlu cesty; END\_NODE\_ID určuje ID koncového uzlu cesty; sloupeček COST udává celkové ohodnocení cesty; sloupec SIMPLE nabývá hodnot 'Y', nebo 'N' a vyjadřuje, zda je cesta jednoduchá či komplexní; prostorové sítě využívají sloupeček typu SDO\_GEOMETRY určující tvar cesty, jeho název je volitelný; tvar cesty je tvořen kombinací tvarů všech hran v cestě; poslední dva sloupečky s povinným označením PATH\_NAME a PATH\_TYPE slouží pro pojmenování cesty a jejího typu, přičemž toto pojmenování je libovolné; tabulka cest může dále obsahovat jiné libovolné sloupečky;
- 5) path links (hrany cest) – tato tabulka slouží k zaznamenání seznamu hran tvořících určitou cestu; primární klíč tvoří sloupečky PATH\_ID a LINK\_ID; tato unikátní kombinace označuje ID cesty a ID hrany nacházející se na dané cestě; sloupeček SEQ\_NO označuje pořadí hrany v posloupnosti hran dané cesty; tabulka hran cest může dále obsahovat jiné libovolné sloupečky.

### 3.3.2 Vytváření grafu

Nejjednodušším grafem, který lze v Oracle Network Data Model definovat je graf tvořený tabulkou uzlů (pouze sloupeček NODE\_ID) a hran (pouze sloupečky LINK\_ID, START\_NODE\_ID, END\_NODE\_ID). Takový graf vyjadřuje logickou síť bez ohodnocení. Je možné definovat 3 hlavní způsoby, jak zkonstruovat graf pomocí balíčku funkcí a procedur SDO\_NET:

- 1) automatické vytváření zavoláním jedné procedury – pro automatické vytvoření sítě je použita buď procedura CREATE\_SDO\_NETWORK pro konstrukci prostorových sítí, nebo procedura CREATE\_LOGICAL\_NETWORK pro konstrukci logických sítí; obě procedury vyžadují zadání několika argumentů určujících parametry sítě; existují 2 možnosti jak zavolat uvedené procedury; při volbě první varianty je nutné uvést 4 argumenty a to název, počet úrovní hierarchie,

orientovanost sítě, a zda budou uzly ohodnocené; dojde k vygenerování tabulek a zapsání informací o hranově ohodnocené síti do metadat; druhou možností je zavolat proceduru až s 12 argumenty; druhá možnost pracuje s již vytvořenými tabulkami, které se nalinkují do metadat; první 3 argumenty jsou stejné jako u předchozí varianty procedury, dále jsou určeny názvy tabulek uzlů, hran, cest a hran cest; dále se zadávají názvy sloupců určujících typ geometrie a ohodnocení prvků; pokud není určen název sloupce geometrie, pak databáze předpokládá název GEOMETRY; ovšem pokud není zadán název sloupce ohodnocení, pak se předpokládá, že žádné ohodnocení neexistuje; při automatickém vytváření sítě je nutné dát pozor na správnost dokončení vytvářecí procedury; pokud běh procedury předčasně skončí, pak dojde k vytvoření pouze části sítě, a proto je nutné před opětovným spuštěním procedury smazat vytvořenou část sítě voláním procedury DROP\_NETWORK;

- 2) manuální vytváření – oproti automatickému vytváření je časově náročnější, ovšem umožňuje větší flexibilitu při vytváření sítě (především kontrolu nad fyzickým úložištěm, lepší správa tabulkového prostoru a dělení tabulek); uživatel databáze musí při manuálním vytváření sám aktualizovat metadata a kontrolovat konzistenci údajů v tabulkách proti metadatům; postup při manuálním vytváření je následující; za prvé jsou vytvořeny tabulky uzlů, hran, případně cest a hran cest s předepsanými názvy sloupců; v druhém kroku budeme vytvářet síť se zvolenými parametry přidáním záznamu do tabulky metadat; poslední krok není povinný, ovšem doporučuje se provést; jedná se o kontrolu správnosti sítě pomocí validačních funkcí Oraclu; hlavní validační funkce VALIDATE\_NETWORK s parametrem název sítě vrací TRUE pokud je síť v pořádku, NULL pokud taková síť neexistuje a v případě jiné chyby vrací řetězec s její diagnostikou;
- 3) manuální vytvoření nad pohledy tabulek – princip stejný jako 2) bod s tím rozdílem, že síť je vytvářena na základě pohledů nad existujícími tabulkami; pohledů se využívá především z důvodu možnosti využití zavedených tabulek, které nemají pojmenované sloupce dle požadavků Oracle Network Data Model.

V Oracle Network Data Model jsou navíc k dispozici funkce pro ověření správnosti vytvořeného grafu (sítě) dle předepsaných pravidel Oraclu.

### 3.3.3 Další grafové operace

Při práci s grafem v Oracle Network Data Model uživatel může využít dalších funkcí z balíčku SDO\_NET. Pro odstranění celé sítě slouží procedura DROP\_NETWORK s parametrem název sítě. Odstraní záznam o síti z metadat a zároveň všechny tabulky (případně pohledy), která síť tvoří. Pokud je požadována pouze redefinice sítě, pak je výhodnější smazat záznam o síti z metadat a následně vložit nový s požadovanými parametry.

Další skupinou jsou funkce sloužící ke zjištění všech informací o síti vždy s argumentem název sítě. Pokud vznikne požadavek zavolat všechny tyto zjišťovací funkce najednou,

použije se procedura SHOW\_NET\_DETAILS. Do této skupiny lze také zařadit funkce pro zjištění podrobností o jednotlivých uzlech (argumentem je vždy název sítě a ID uzlu).

Jinou skupinou jsou procedury pro zjištění výskytu zvláštností v síti, jako jsou izolované uzly nebo hrany bez počátečního či koncového uzlu. Tato skupina funkcí vyžaduje zadání argumentu názvu sítě. Ovšem takovýmto zvláštnostem lze předejít přidáním různých omezení nad tabulkami uzlů a hran.

### **3.3.4 Java API pro Oracle Network Data Model**



Balíčky Java API funkcí umožňují jednak správu grafů tak i jejich analýzu. Je nutné poznamenat, že všechny změny nejprve probíhají nad grafem uloženým v paměti, a nikoliv přímo nad databází. Celé Java API určené pro grafovou oblast lze rozdělit na tři hlavní oblasti tříd funkcí:

- 1) Network, Node, Link and Path – tato třída umožňuje ukládat a starat se o údržbu sítě a prvků sítě; také lze díky ní zjistit komplexní informace o síti včetně metadat;
- 2) Network Manager – tato třída poskytuje veškerou síťovou analýzu, pracuje systémem načtení celé sítě z databáze do paměti; po provedení změn (nalezení nových cest) umožňuje zápis změn zpět do databáze; základní funkcí tohoto modulu je nalezení cesty mezi dvěma uzly; dále samozřejmě poskytuje složitější analytické funkce jako nalezení nejkratší cesty mezi dvěma uzly, nalezení nejbližšího souseda, nalezení všech uzlů v určitém okruhu dle vzdálenosti, řešení problému obchodního cestujícího, nalezení minimální kostry grafu a další; mimo jiné nám tato třída také umožňuje definovat omezení analyzovaného prostoru sítě;
- 3) Network Factory – tato třída umožňuje vytvářet síť a jednotlivé síťové prvky.

## **3.4 Souhrnný přehled vybraných řešení**

V této kapitole je pro přehlednost uvedena tabulka srovnání představených rozšíření databází pro ukládání prostorových dat.

**Tabulka 2 – Přehled prostorových rozšíření relačních databází**

Rozšíření	DB2 Spatial Extender	PostGIS	Oracle Spatial
Základní databáze	DB2	PostgreSQL	Oracle Enterprise Edition
Typ základní databáze	Relační databáze	Objektově-relační databáze	Relační databáze
Dotazovací jazyky	SQL	SQL, pgSQL	SQL, PL/SQL
Oblast zaměření rozšíření	Možnost ukládání a správy prostorových dat	Orientace na zpracování a analýzu vektorových i rastrových dat	Možnost pracovat s většinou existujících prostorových dat a provádění základních i pokročilých analýz nad těmito daty
Zvláštní specifikace rozšíření	Souřadnice ukládány formou zeměpisné šířky a délky	Pokročilé importní a exportní nástroje prostorových dat ze standardizovaných formátů souborů	Ukládání cest získaných analýzou sítě
Licence	Closed Source	Open Source	Closed Source
Logo	DB2 Spatial Extender		

## 4 Návrh a implementace aplikace pro testování grafových operací

Tato kapitola se zabývá rozbořem požadavků na systém a jeho následnou implementací.

### 4.1 Požadavky na systém

Na základě konzultace s vedoucím práce byly definovány následující požadavky na aplikaci pro testování typických grafových operací v databázích Oracle a Neo4j:

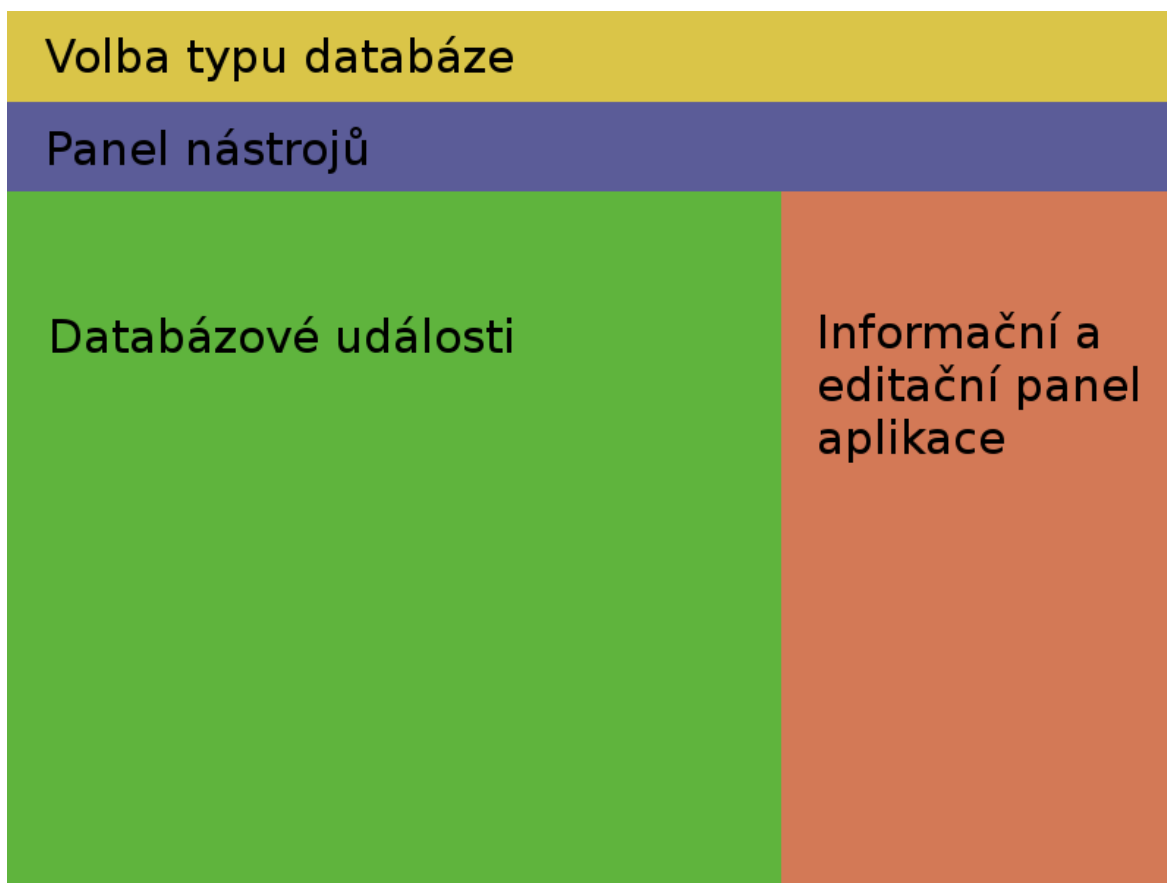
- 1) bude se jednat o desktopovou či webovou aplikaci, přičemž programovací jazyk je volitelný;
- 2) aplikace bude umožňovat generovat grafy s náhodnými uzly a hranami pro Neo4j databázi;
- 3) v rámci části aplikace určené pro Oracle databázi poskytne možnost konverze aktuálně načteného grafu v Neo4j do Oraculu a to z důvodu využití stejné množiny dat při testování obou databází;
- 4) aplikace bude umožňovat řešit 3 vybrané typické grafové úlohy v obou zkoumaných databázích – řešení úloh bude v praktické části implementováno studentem, v případě existence předpřipravených řešení úloh v rozhraních databází, bude možné tato řešení v rámci aplikace taktéž spouštět a testovat;
- 5) pro účely testování bude aplikace poskytovat možnost sledovat 3 ukazatele – čas potřebný pro řešení zadané úlohy, vytížení procesoru počítače a využití paměti RAM v průběhu řešení úlohy.

### 4.2 Obecný návrh

#### 4.2.1 Grafické rozhraní

Grafické rozhraní aplikace by mělo být přehledné a intuitivní. Dle těchto zásad byla navržena následující struktura komponent rozhraní (viz obrázek 9). Okno aplikace bylo rozděleno do 4 částí. Horní část bude obsahovat nástroj pro volbu aktuálně používané databáze. Prostřední úsek aplikace by měl obsahovat lištu nástrojů pro ovládání zvolené databáze. Největší prostor bude vyhrazen oknu pro textový výpis všech databázových událostí a jejich výstupů. Poslední částí aplikace by měl být panel sloužící jako informační ukazatel aktuálního stavu programu společně s možností editace parametrů testování. Při návrhu grafického rozhraní byl použit grafický editor GIMP. Pro interakci programu s uživatelem se jeví jako vhodná volba použití základních typů formulářů.





Obrázek 9 – Grafické rozvržení aplikace

#### 4.2.2 Generátor grafů

Důležitým předpokladem pro testování databázových úloh je možnost spouštět úlohy nad bázemi dat různých velikostí. Navíc je vhodné, aby se tyto báze týkaly stejné oblasti problematiky. Na základě těchto předpokladů došlo k vyhledání dostupnýchází dat pro Neo4j na internetu. Bylo nalezeno několik poskytnutýchází, například na webových stránkách Neo4j. Ovšem v rámci stejné problematiky se nepodařilo vyhledat 3 báze dat s různou velikostí. Z tohoto důvodu se předpokládá vytvoření vlastního generátoru databází pro Neo4j. Úkolem bude testovat databáze při zpracování 3 různých grafových úloh. Z tohoto důvodu se předpokládá možnost generovat grafy v Neo4j z 3 různých oblastí problematik.

### 4.3 Implementace testovacího systému

Na základě požadavků lze říci, že desktopový typ aplikace je pro účely testování výkonu databázových systémů dostačující. Dále budou popsány údaje týkající se implementace testovacího systému.

#### 4.3.1 Použité technologie

Samotná implementace programu bude provedena v programovacím jazyce Java. Hlavním důvodem této volby byla existence programovacího API pro jazyk Java u obou zkoumaných databází. Dalším důvodem pro výběr jazyka Java byla dobrá znalost tohoto

jazyka a tudíž možnost rychlejší implementace testovací aplikace. Na základě dobré znalosti bylo zvoleno i vývojové prostředí NetBeans verze 7.2.1. Desktopová aplikace bude koncipována jako formulářová aplikace využívající Java Swing komponenty.

Jako testovaná verze databázového serveru Neo4j byla vybrána komunitní verze 2.0.3, kterou lze používat zdarma. Získání je možné ze zdroje [5]. Pro testování Oracle Spatial rozšíření byl zvolen databázový server Oracle 11g verze Enterprise edition release 11.2.0.1.0. Tuto verzi lze zdarma stáhnout a používat pouze pro studijní účely, pro stažení je nutné být registrován na stránkách společnosti Oracle [19]. Je nutné použít Enterprise edici, protože žádná jiná edice neobsahuje rozšíření Oracle Spatial. Detailní popis instalací těchto systémů lze nalézt na [5] a [19]. Obě testované databáze při testování fungovaly jako lokální databázové servery na lokálním přenosném počítači. [4]

#### **4.3.2 Propojení NetBeans a Neo4j**

Pokud je nainstalovaný a spuštěný databázový server Neo4j, pak existují 2 možnosti, jak ovládat konkrétní databázi z programovacího prostředí NetBeans v jazyce Java. Prvním způsobem je využití připravených knihoven, které lze získat na manuálových stránkách NetBeans [20]. Tyto knihovny ve formátu .jar poté stačí v NetBeans běžným způsobem připojit ke konkrétnímu Java projektu. Druhou možností je vytvoření vlastních knihoven pro práci s Neo4j, případně připojení neoficiálních knihoven poskytnutých jinými vývojáři. V rámci diplomové práce jsem zvolil první variantu. Funkce poskytnuté v připojených byly pro potřeby aplikace dostačující.

#### **4.3.3 Propojení NetBeans a Oracle Spatial**

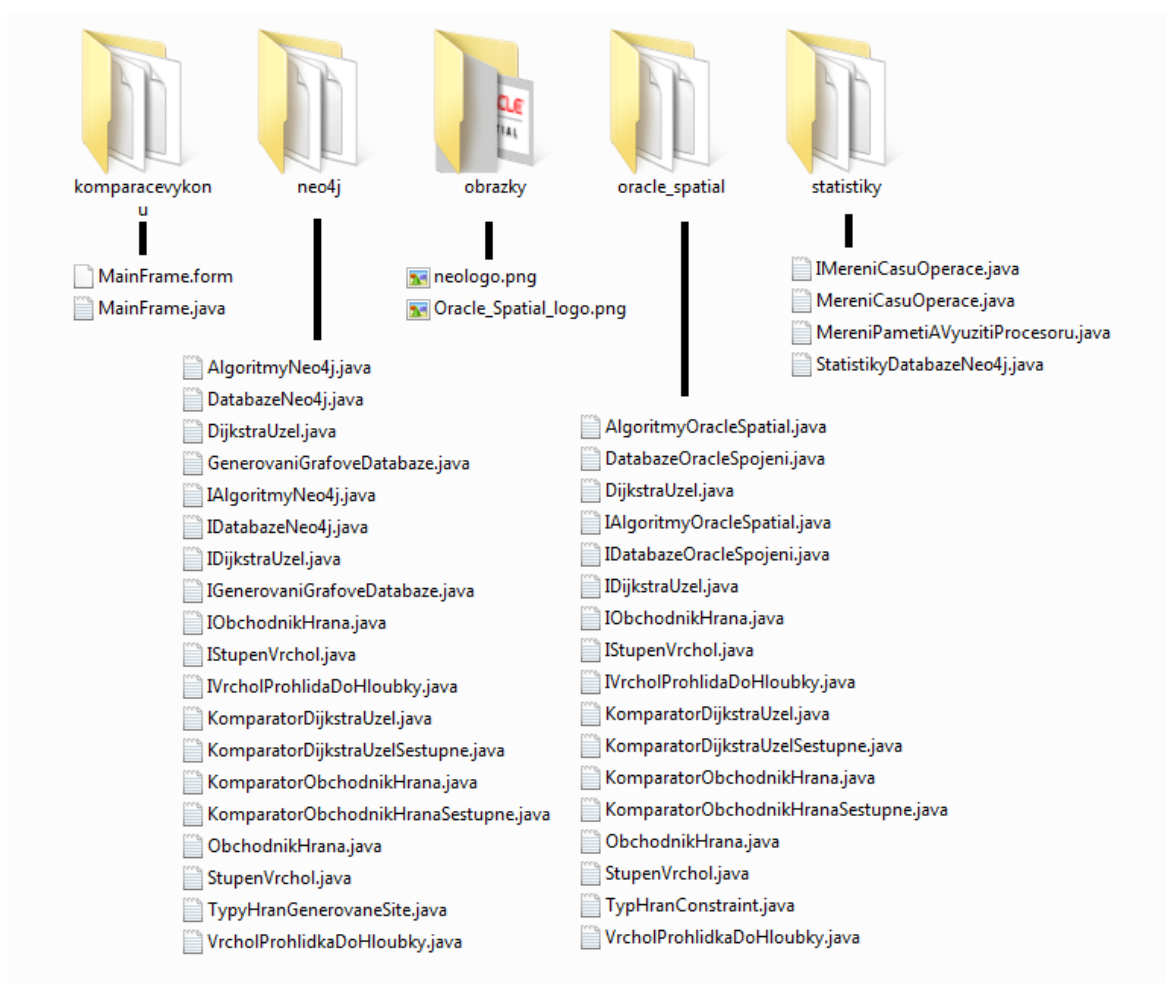
Po instalaci databázového serveru Oracle Enterprise existují 2 možnosti, jak pracovat s databází využívající rozšíření Oracle Spatial v jazyce Java pod programovacím prostředím NetBeans [21]. První možností je připojení k databázi přes Thin JDBC spojení, přes tzv. lehkého klienta. Druhou alternativou je využití OCI klienta. Pokud je zvolena první možnost, pak je nutné nejprve získat JDBC ovladač například z webových stránek společnosti Oracle. Po získání souboru s příponou .jar bude tento soubor klasicky připojen v rámci projektu jako knihovna. V databázi již stačí zavolat funkci *getConnection()* třídy *DriverManager*, která dle zadaných parametrů – umístění databáze, přihlašovacího jména a hesla – vytvoří funkční spojení s databází. V případě druhé možnosti je postup pro vytvoření spojení obdobný, nejdříve bude nainstalován Oracle databázový klient. Následně OCI ovladač používá nativní knihovny nainstalovaného Oracle databázového klienta pro komunikaci s databází.

#### **4.3.4 Adresářová struktura**

Na obrázku 10 lze vidět celou adresářovou strukturu projektu. Třídy a další objekty byly strukturovány do balíčků, které budou nyní podrobněji popsány:

- 1) „komparaceVykonu“ – balíček obsahující spouštěcí třídu aplikace;

- 2) „neo4j“ – balíček zahrnující třídy pro spojení s databází Neo4j, současně také třídy obsahující implementace grafových algoritmů a nakonec pomocné i obslužné třídy týkající se provádění algoritmů;
- 3) „obrazky“ – balíček obsahující obrázky použité v aplikaci;
- 4) „oracle\_spatial“ – balíček zahrnující třídy pro spojení s databází Oracle, obsahuje také další třídy týkající se prací s touto databází a to základní a pomocné třídy obstarávající řešení grafových algoritmů;
- 5) „statistiky“ – balíček obsahující třídy sloužící k měření vybraných parametrů náročnosti úloh.

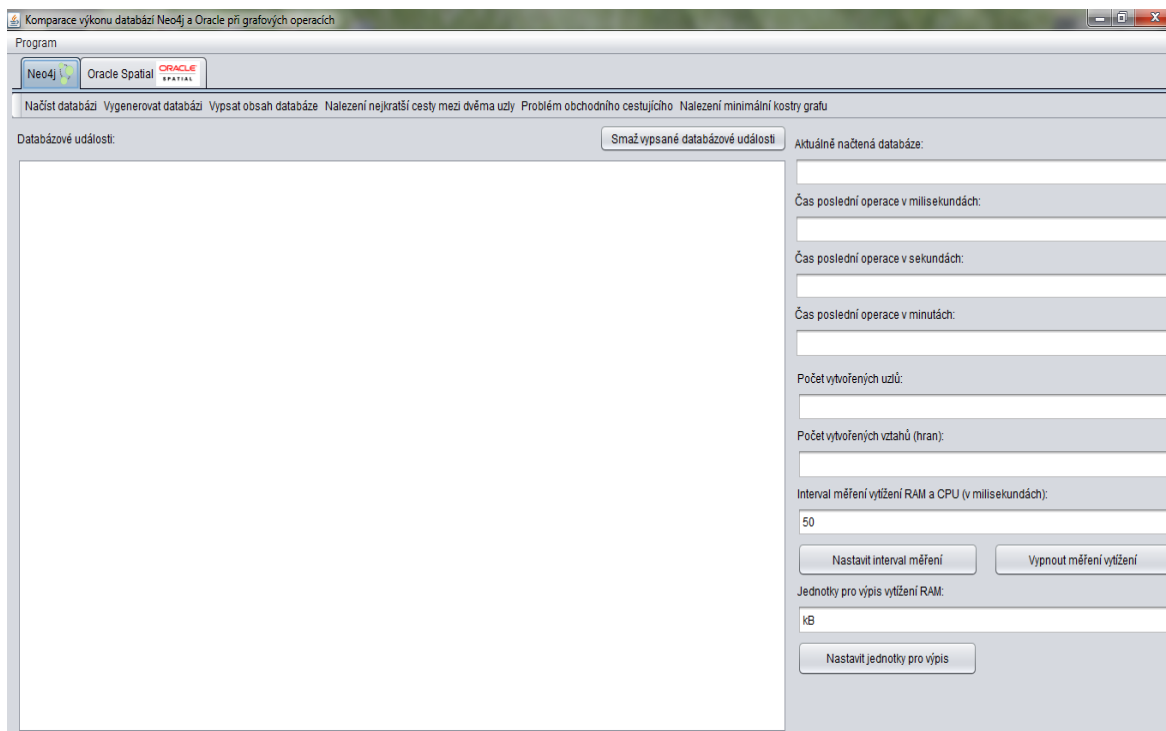


Obrázek 10 – Adresářová struktura

#### 4.3.5 Grafické prostředí

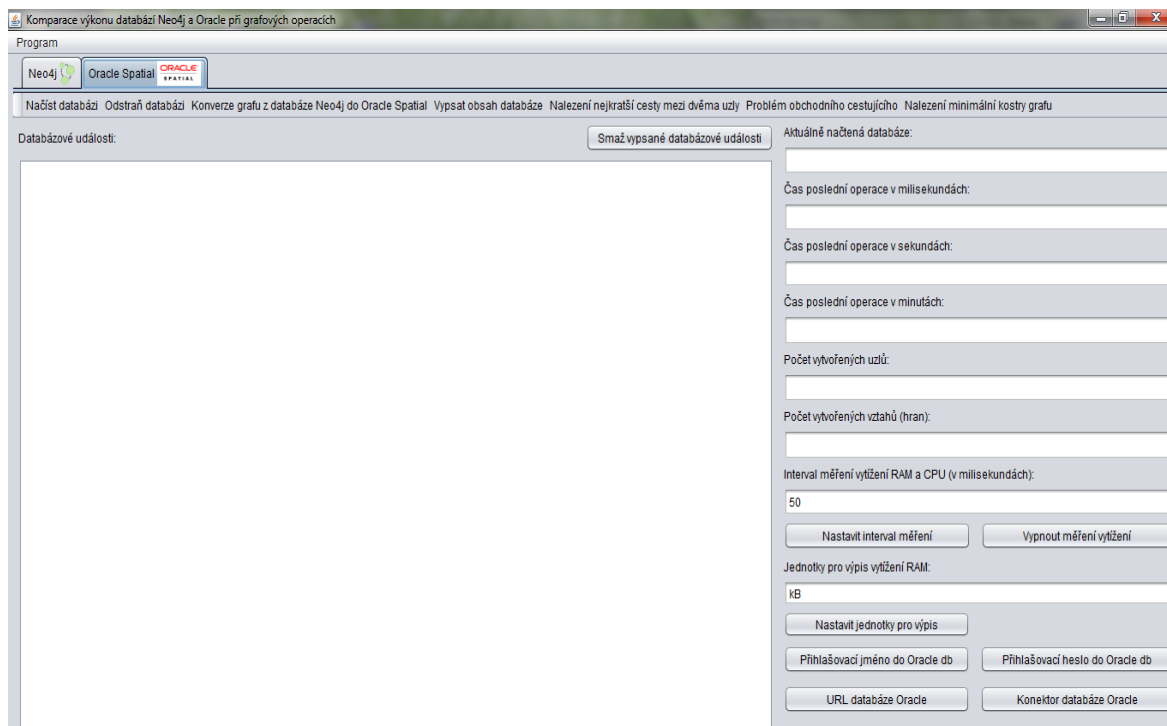
Při vytváření uživatelského rozhraní bylo snahou dodržet rozložení komponent dle návrhu grafického rozhraní. Horní část zůstala oproti návrhu nezměněna a obsahuje možnost ukončení aplikace a volbu aktuálně používaného okna aplikace dle vybrané databáze. Obě okna shodně obsahují v levé spodní části prostor pro výpis všech ukončených událostí databáze. Obsah zbývajících částí aplikace se liší dle aktuálně testované databáze. Na úvod

je popsána část aplikace určená pro databázi Neo4j (viz obrázek 11). Prostřední část okna aplikace obsahuje veškeré možnosti pro výběr, generování a výpis databáze a spouštění předpřipravených algoritmů. V pravém dolním úseku aplikace lze nalézt označení aktuálně načtené databáze, čas poslední operace a počet uzlů i hran v databázi. Navíc byla do této části umístěna informace o hodnotě intervalu pro měření využití RAM a vytížení CPU, jednotky, ve kterých bude měřeno vytížení a možnosti tato nastavení upravit.



**Obrázek 11 – Grafické prostředí pro testování Neo4j databáze**

Nyní bude popsána část určená pro Oracle Spatial databázi (viz obrázek 12). V prostředním úseku okna aplikace oproti Neo4j části chybí možnost generovat databázi, ovšem přibýly zde možnosti odstranit databázi a zkonvertovat databázi z Neo4j. Pravý dolní úsek aplikace obsahuje navíc oproti Neo4j části možnost editace nastavení připojení k databázi Oracle. Veškerý výběr možností a zadávání vstupů od uživatele je řešeno přes dialogová okna. Konkrétně bylo využito dialogových oken různých typů z Java třídy JOptionPane.



Obrázek 12 – Grafické prostředí pro testování Oracle Spatial databáze

#### 4.3.6 Generátor grafů

Na základě požadavků byl vytvořen generátor grafových databází pro Neo4j. Tento generátor umožňuje generovat grafy z 3 různých oblastí. Je tedy možné získat grafy silničních sítí, železničních sítí a leteckých sítí. Generátor dle oblasti vybrané problematiky pojmenuje názvy vlastností a typy uzlů i hran. Dále určí hodnotu minimálně a maximálně generovaného ohodnocení hran. Generátor dále umožňuje zvolit mezi 3 typy generovaných grafů – graf bez omezení, úplný graf a graf s propojenými všemi uzly. Uživatel také může při generování zvolit velikost grafu, dle počtu uzlů, umístění v rámci souborového systému a jeho název. Po zjištění všech potřebných údajů je princip generování následující. Každý uzel se skládá z vygenerovaného jména a automaticky inkrementovaného id v rámci grafu. Hrany se skládají z vygenerované hodnoty ohodnocení, automaticky inkrementovaného id v rámci grafu a odkazů na uzly, které hrana spojuje. Při veškerém generování bylo využito generátoru třídy Random. Nyní bude uvedena část kódu aplikace důležité funkcionality. Uvedený Java kód zajišťuje nastavení proměnných pro generování dle oblasti zvolené problematiky.

```
switch (typSite) {
    case 1:
        labelUzel = "Stanice";
        nazevVlastnosti = "Nazev stanice";
        labelHrana = TypyHranGenerovaneSite.KOLEJ;
        minVzdalenost = 2;
        maxVzdalenost = 30;
        break;
    case 2:
        labelUzel = "Krizovatka";
        nazevVlastnosti = "Oznaceni krizovatky";
```

```

        labelHrana = TypyHranGenerovaneSite.SILNICE;
        minVzdalenost = 1;
        maxVzdalenost = 7;
        break;
    case 3:
        labelUzel = "Letiste";
        nazevVlastnosti = "Nazev letiste";
        labelHrana = TypyHranGenerovaneSite.KORIDOR;
        minVzdalenost = 200;
        maxVzdalenost = 30000;
        break;
    default:
        labelUzel = "Stanice";
        nazevVlastnosti = "Nazev stanice";
        labelHrana = TypyHranGenerovaneSite.KOLEJ;
        minVzdalenost = 2;
        maxVzdalenost = 30;
        break;
}

```

#### 4.3.7 Převodník grafů z Neo4j do Oracle Spatial

Převodník grafových databází slouží pro překlopení vybrané grafové databáze z Neo4j do Oracle Spatial. Umožňuje nám tedy získat graf z Neo4j a dále ho v Oracle databázi používat. Obě databáze nepracují se stejným principem uchování uzlů a hran. Při konverzi tedy může dojít ke ztrátě určitých informací. V první řadě bude ztracena informace, jakého typu je konvertovaný uzel, pokud je takový uzel v Neo4j označen více typy. Dále je možné přijít o alternativní sloupec ohodnocení hran. V Neo4j je totiž vybíráno, jaká vlastnost hran bude použita jako ohodnocení až při požadavku na konkrétní analýzu grafu. Ovšem v Oracle Spatial je zvoleno, jaký údaj má obsahovat ohodnocení hran již při tvorbě grafu. Samotný převodník dále kontroluje, zda se již databáze se stejným jménem v Oracle Spatial nevyskytuje. Dále je nutné kontrolovat počet konvertovaných uzlů a hran. Neo4j totiž umožňuje uchovávat počet uzlů a hran dle rozsahu datového typu Long, ovšem Oracle Spatial pracuje pouze s rozsahem datového typu Integer. Uživatel tedy musí při konverzi zvolit vlastnost uzlů určující jejich názvy a vlastnost hran určující jejich ohodnocení. Nyní bude uvedena část kódu aplikace důležité funkcionality. Uvedený Java kód zajišťuje konverzi všech hran s vybranou vlastností ohodnocení z Neo4j do Oracle Spatial.

```

Iterator<org.neo4j.graphdb.Relationship> iteratorHran =
vsechnyVztahyNea.iterator();

while (iteratorHran.hasNext()) {
    aktualniHrana = iteratorHran.next();
    typ = aktualniHrana.getType().name();
    vychoziId = (int) aktualniHrana.getStartNode().getId();
    ciloveId = (int) aktualniHrana.getEndNode().getId();
    vychoziUzel = logicalNetwork.getNode(vychoziId);
    cilovyUzel = logicalNetwork.getNode(ciloveId);
    novaHranaOracle = NetworkFactory.createLogicalLink((int)
aktualniHrana.getId(), null, vychoziUzel, cilovyUzel,
Double.parseDouble(aktualniHrana.getProperty(vlastnostOhodnoceniHrany).toString()));
    novaHranaOracle.setType(typ);
    logicalNetwork.addLink(novaHranaOracle);
}

```

#### 4.4 Před prvním spuštěním testovací aplikace

Pro spuštění testovací aplikace je nutné provést následující kroky. Nainstalovat databázový server Neo4j ve verzi 2.0.3, jiné verze totiž nebyly testovány a není zaručena jejich kompatibilita s testovací aplikací. Instalační balíček lze získat na webových stránkách společnosti Neo Technology, Inc. [5]. Dále nainstalovat, případně mít k dispozici spuštěný databázový server Oracle 11g Enterprise Edition. Nakonec je nutné v testovacím programu v části Oracle Spatial nastavit přihlašovací údaje do Oracle databáze. Provedení těchto kroků by mělo uživateli umožnit používat testovací aplikaci v plné míře.

#### 4.5 Ovládání aplikace

Nyní dojde k představení práce se samotnou aplikací pro testování. Celý program je koncipován jako aplikace ovládaná myší s možností zadávat vstupy přes klávesnici. V každé části programu, Neo4j i Oracle Spatial se nachází okolo 10 možností voleb. Konfigurační volby spojení s databází a měření vytížení jsou editovány pouze přes tlačítka v pravé dolní části aplikace. Panel nástrojů v horní části aplikace obsahuje všechny dostupné volby oddělené velkými písmeny na začátku jejich označení. Nástroje nelze využít v jakékoliv době běhu aplikace. Pro výpis obsahu databáze či spuštění některého z nabízených algoritmů, je nutné mít načtený graf, nad kterým lze řešit vybraný problém. V případě nepřehlednosti, či potřeby mít k dispozici pouze poslední databázové události, lze smazat všechny předchozí pomocí tlačítka „Smaž vypsane databázové události“. Lze říci, že došlo k testování veškerých voleb, které lze v rámci aplikace zadat a jejich funkčnost byla odladěna.

## 5 Testování

V této kapitole bude představena práce s aplikací při zadávání testovacích scénářů. Dále bude uveden výběr testovacích dat. Nakonec dojde k samotnému testování a vyhodnocení výsledků.

### 5.1 Zadávání testovacích scénářů

Jako příklad bude použito testování databází při řešení úlohy nalezení nejkratší cesty mezi dvěma uzly. Před samotným testováním jsou ukončeny všechny aplikace, které by mohli zkreslit měření využití RAM a vytížení CPU. Předpokladem je existence 3 předpřipravených různě velkýchází dat ze stejné oblasti problematiky. Před samotným testováním budou nastaveny v obou částech aplikace jednotky, v kterých bude probíhat měření využití RAM a požadovaná velikost intervalu měření využití RAM i vytížení CPU. Postup při testování je následující:

- 1) Neo4j – na začátek je načtena první báze dat; dále bude spuštěno „Nalezení nejkratší cesty mezi dvěma uzly“ v části pro Neo4j; bude zvolen jeden z nabízených algoritmů, přes vstupy bude zadáno id výchozího a cílového uzlu, dle počtu uzlů v databázi; dále je vybrán typ hran, přes které lze cestovat a název vlastnosti hran, která určuje jejich ohodnocení; nakonec je zadán počet opakování stejné úlohy pro požadovaný počet měření; výstupy jsou zaznamenávány v libovolném tabulkovém procesoru pro pozdější zpracování; zadávání se opakuje pro ostatní typy algoritmů; po otestování všech typů algoritmů je báze dat změněna a přichází na řadu opakování výše popsaného postupu;
- 2) Oracle Spatial – po zaznamenání všech měření v Neo4j části aplikace se pokračuje v části Oracle Spatial; při spouštění algoritmu jsou opakovány všechny volby dle postupu pro Neo4j; rozdíl je pouze ve vynechání volby vlastnosti hran, která určuje ohodnocení; tato volba v části Oracle Spatial se provádí již při konverzi báze dat z Neo4j.

V části Neo4j i Oracle Spatial jsou k dispozici různé množiny algoritmů řešících vybrané problémy z důvodu neexistence některých z nich v Java API pro Neo4j. Testovat se ovšem budou pouze algoritmy, které jsou k dispozici v obou částech aplikace. Veškeré testování bude probíhat na přenosném počítači s konfigurací dle tabulky 3.

**Tabulka 3 – Konfigurace přenosného počítače použitého při testování**

Lenovo ThinkPad Edge E530 3259-HHG	
Procesor	Intel Core i3 2328M Sandy Bridge
Taktovací frekvence procesoru	2,2 GHz
Počet jader procesoru	2
Počet vláken jádra procesoru	2
Paměť RAM	4 GB DDR3 SDRAM, 1 600 MHz
Pevný disk	500 GB SATA
Operační systém	Microsoft Windows 7 Home Premium 64bitový



## 5.2 Testovací data

Jak již bylo zmíněno v kapitole 5, výkon databází bude testován pomocí 3 úloh: hledání nejkratší cesty mezi dvěma uzly, úloha obchodního cestujícího a hledání minimální kostry grafu. Pro každou z těchto úloh je z důvodu testování vygenerována speciální množina grafových databází. Velikost množiny bude navržena s ohledem na časovou složitost algoritmu řešícího daný problém. Nyní je již možné popsat, jaké grafy budou vygenerovány pro konkrétní algoritmy:

- 1) hledání nejkratší cesty mezi dvěma uzly – v praxi se lze setkat s problémem nalezení nejkratší cesty po železnici mezi 2 body z důvodu minimalizace nákladů vynaložených na přepravu, proto se bude tato úloha řešit nad grafy železničních sítí; tuto úlohu lze řešit nad jakýmkoliv grafem, nevyžaduje tedy žádné speciální omezení; časová složitost v případě řešení Dijkstrovým algoritmem bude v nejhorším případě kvadratická [16]; pro účely testování budou vygenerovány 3 grafy o velikostech 5 000, 7 500 a 10 000 uzlů;
- 2) úloha obchodního cestujícího – tento problém často řeší automobilové navigační zařízení, či logistické systémy, proto bylo navrženo řešit tuto úlohu nad grafy silničních sítí; pro tuto úlohu se budou používat pouze úplné grafy; nejlepší algoritmy v současné době řeší tento problém časově s exponenciální složitostí, přičemž vyřešit se podařilo nejvýše problém, do kterého řádově vstupovalo 100 000 uzlů [22]; z důvodu rychle rostoucí časové náročnosti společně s velikostí grafu budou pro účely testování vygenerovány 3 grafy o velikostech 50, 100 a 150 uzlů;
- 3) hledání minimální kostry grafu – pokud existuje požadavek na minimalizaci nákladů na přepravu zásilek leteckou dopravou, pak je nutné řešit problém nalezení minimální kostry grafu; proto bylo navrženo řešit tuto úlohu nad grafy leteckých sítí; tato úloha má pouze jediné omezení a to, že jí lze řešit pouze nad grafy se všemi propojenými uzly; pokud se tento problém řeší Kruskalovým algoritmem, pak bude časová složitost nejhůře logaritmická [16]; pro účely testování budou vygenerovány 3 grafy o velikostech 500, 750 a 1 000.

Veškeré grafy se následně zkonvertují do Oracle Spatial databáze. Všechny vygenerované grafy použité při testování se nachází na přiloženém kompaktním disku v adresáři „testovaci\_data“.

## 5.3 Hledání nejkratší cesty mezi dvěma uzly

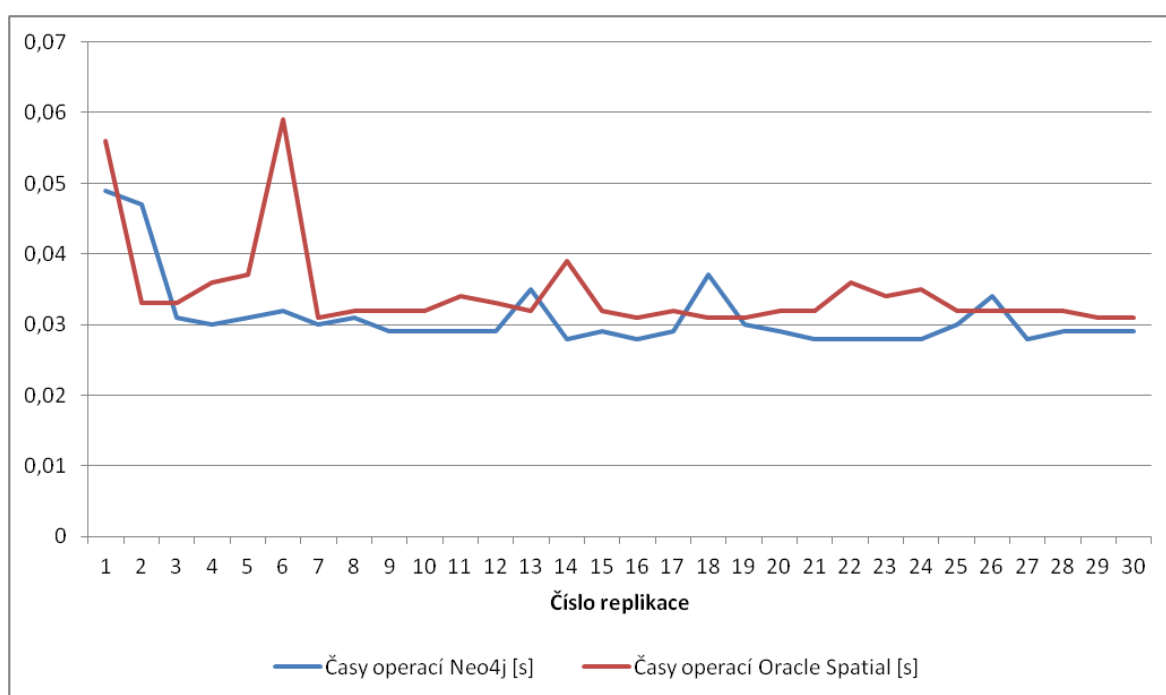
Pro vyhledání nejkratší cesty mezi 2 uzly existují v Neo4j i Oracle Spatial části aplikace připravené 2 možnosti řešení. Jako první bude testována implementace Dijkstrova algoritmu poskytnutého v Java API rozhraní obou databází. Následně bude otestováno řešení úlohy vlastní implementací Dijkstrova algoritmu. Testování bude probíhat vždy v 30 replikacích z důvodu potřeby výpočtu statistických veličin, umožňujících vytvářet přesnější závěry.

### 5.3.1 Spuštěné testy pro Dijkstrův algoritmus z Java API rozhraní obou databází

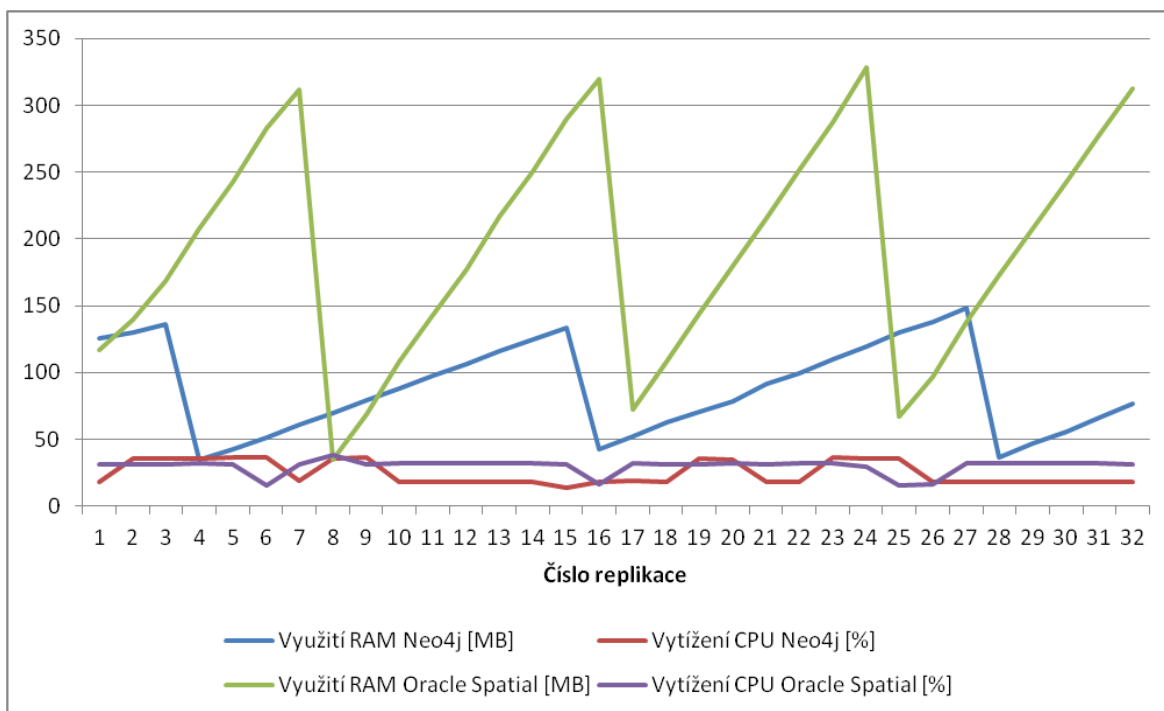
Každý uvedený graf i tabulka bude obsahovat výsledek testování nad danou množinou dat pro oba databázové systémy. Pro stejné množiny dat, pouze rozdílné databáze, byl vždy zadán stejný výchozí a cílový uzel cesty. Výstupy veškerého testování budou zpracovány v tabulkovém procesoru Excel a uloženy na přiložený kompaktní disk do složky „testovani\_vystupy“.

U všech následujících grafů platí, že osa x označuje aktuální číslo replikace. Jednotky osy y jsou zvlášť označeny pro každé konkrétní měření v legendě grafu.

Jako první probíhá testování nad množinou dat s 5 000 uzly.



Obrázek 13 – Graf doby trvání řešení Dijkstrova algoritmu z Java API pro graf s 5 000 uzly

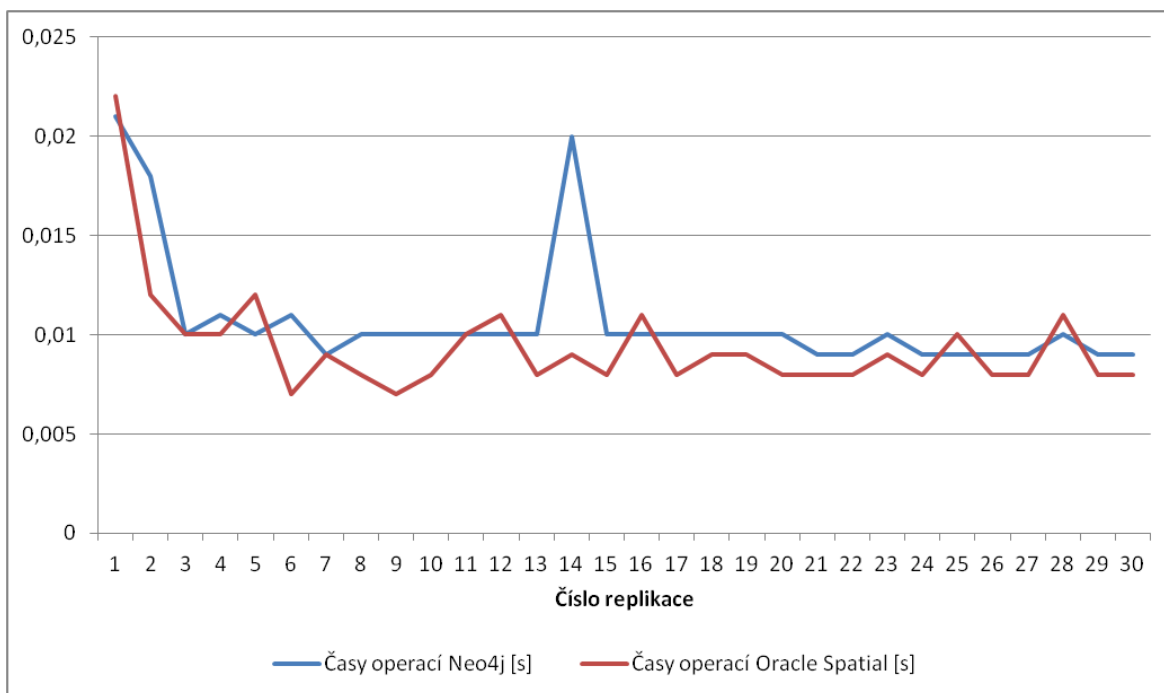


Obrázek 14 – Graf využití RAM a vytížení CPU v průběhu řešení Dijkstrova algoritmu z Java API pro graf s 5 000 uzly

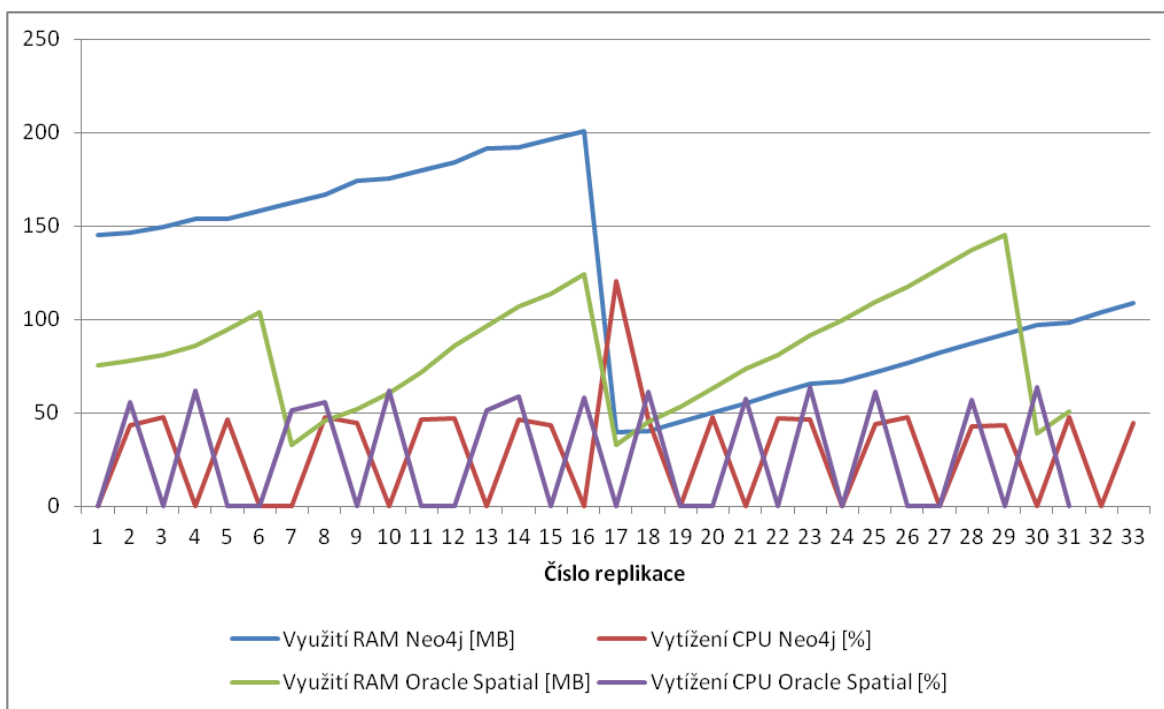
Tabulka 4 – Vypočítané hodnoty statistických proměnných pro graf s 5 000 uzly při řešení úlohy Dijkstrovým algoritmem z Java API

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
Naměřených hodnot	30	32	32	30	32	32
Průměr	0,03	88,18	24,64	0,03	193,13	29,90
Maximum	0,05	148,44	36,54	0,06	328,71	37,95
Medián	0,03	83,59	18,42	0,03	193,75	31,62
Směrodatná odchylka	0,00	34,22	8,65	0,01	83,31	5,45

Pokračujeme testováním nad množinou dat se 7 500 uzly.



**Obrázek 15 – Graf doby trvání řešení Dijkstrova algoritmu z Java API pro graf se 7 500 uzly**

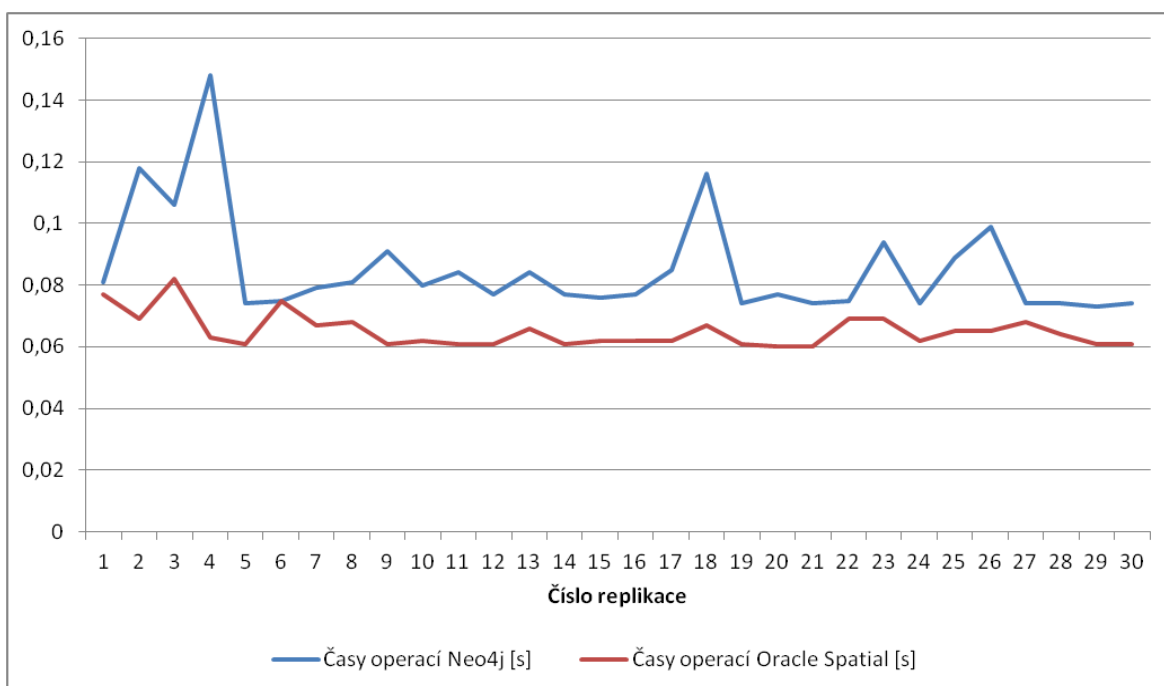


**Obrázek 16 – Graf využití RAM a vytížení CPU v průběhu řešení Dijkstrova algoritmu z Java API pro graf se 7 500 uzly**

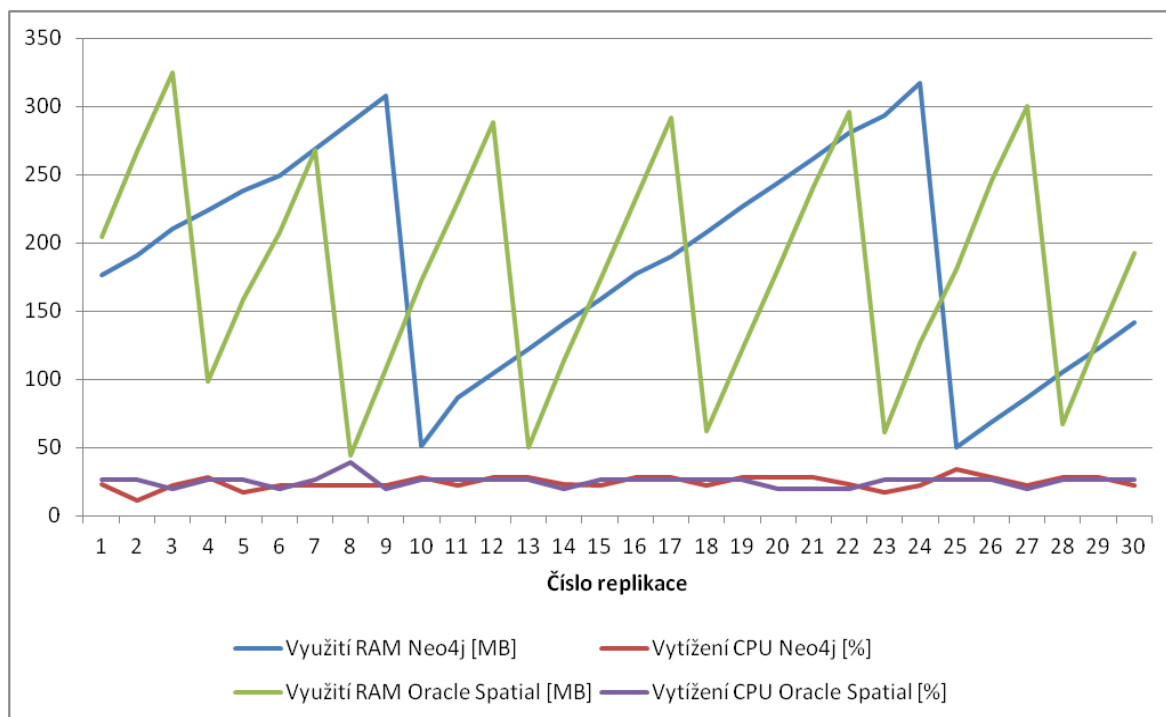
**Tabulka 5 – Vypočítané hodnoty statistických proměnných pro graf se 7 500 uzly při řešení úlohy Dijkstrovým algoritmem z Java API**

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
<b>Naměřených hodnot</b>	30	33	33	30	31	31
<b>Průměr</b>	0,01	120,30	30,01	0,01	83,03	26,42
<b>Maximum</b>	0,02	200,85	120,44	0,02	145,28	63,75
<b>Medián</b>	0,01	108,64	43,26	0,01	81,20	0,00
<b>Směrodatná odchylka</b>	0,00	52,77	27,34	0,00	30,52	29,23

Nakonec probíhá testování nad množinou dat s 10 000 uzly.



**Obrázek 17 – Graf doby trvání řešení Dijkstrova algoritmu z Java API pro graf s 10 000 uzly**



Obrázek 18 – Graf využití RAM a vytížení CPU v průběhu řešení Dijkstrova algoritmu z Java API pro graf s 10 000 uzly

Tabulka 6 – Vypočítané hodnoty statistických proměnných pro graf s 10 000 uzly při řešení úlohy Dijkstrovým algoritmem z Java API

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
Naměřených hodnot	30	30	30	30	30	30
Průměr	0,09	186,29	24,28	0,07	181,31	24,88
Maximum	0,15	316,70	33,54	0,08	324,32	39,09
Medián	0,08	190,13	22,51	0,06	181,02	26,14
Směrodatná odchylka	0,02	78,79	4,43	0,01	82,56	3,92

### 5.3.2 Vyhodnocení naměřených údajů

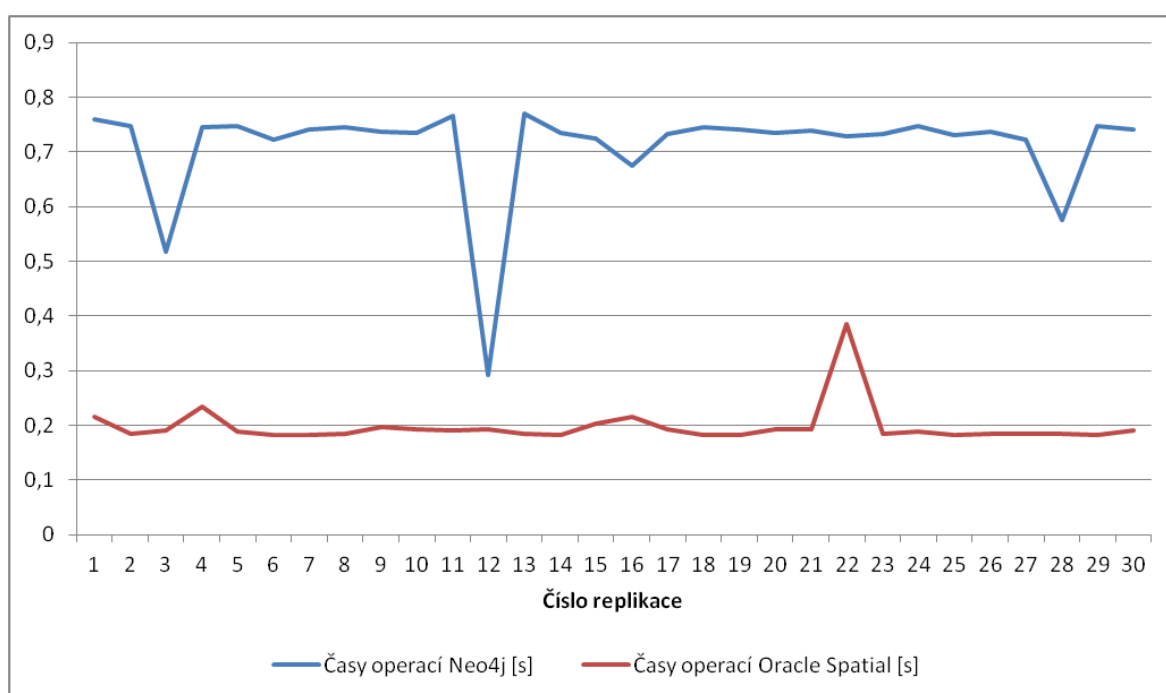
Nejprve je vyhodnoceno testování na základě výsledků běhů Dijkstrova algoritmu z Java API rozhraní obou databází. Přičemž se vychází z obrázků grafů s čísly 13, 14, 15, 16, 17, 18 a tabulek 4, 5, 6. Lze pozorovat, že rychlost provedení algoritmu je u obou databází stejná až na největší množinu dat o velikosti 10 000 uzlů. Při testování této množiny byla Neo4j databáze průměrně o 0,02 sekundy pomalejší. V případě využití RAM lze říci, že větší rozdíl se objevil pouze při testování množiny o velikosti 5 000 uzlů. V uvedeném případě bylo u databáze Oracle využití RAM průměrně o 105 MB vyšší. Z měření bylo zjištěno, že volba databáze nemá v tomto případě vliv na vytížení CPU, které se průměrně pohybovalo okolo 26 %.

### 5.3.3 Spuštěné testy pro vlastní implementaci Dijkstrova algoritmu

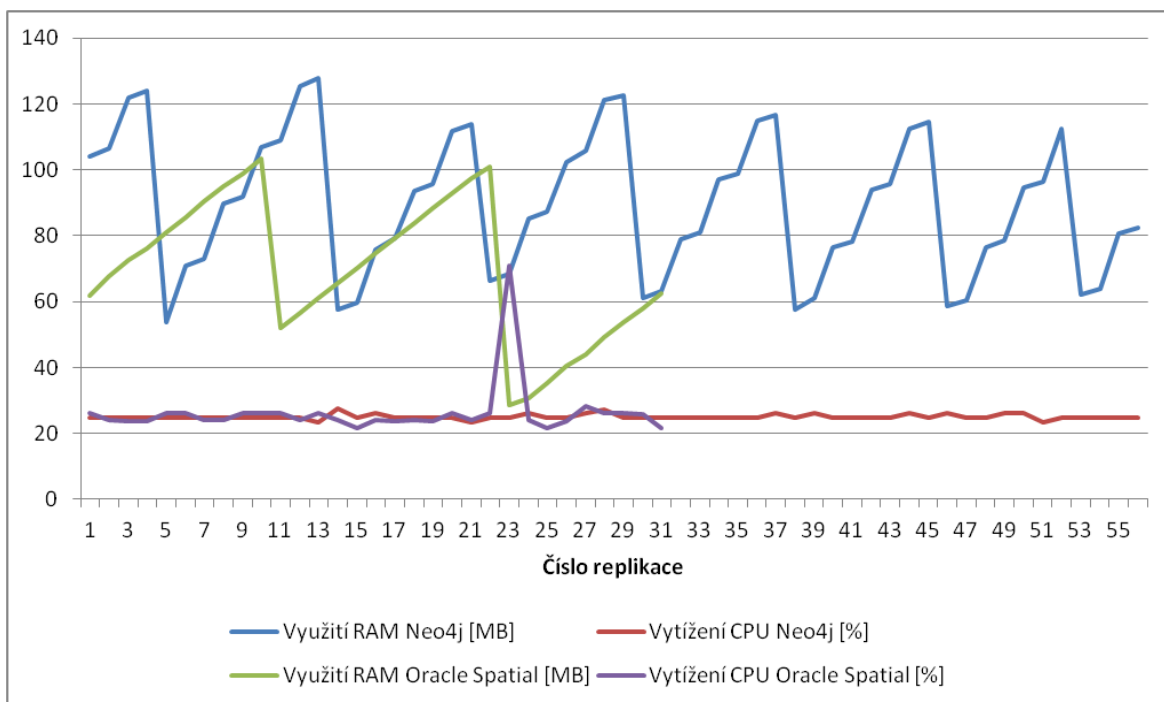
Následující grafy zobrazují testování obou databází při hledání nejkratší cesty pomocí vlastní implementace Dijkstrova algoritmu. Každý uvedený graf i tabulka bude obsahovat výsledek testování nad danou množinou dat pro oba databázové systémy. Pro stejné množiny dat, pouze rozdílné databáze, byl vždy zadán stejný výchozí a cílový uzel cesty. Výstupy veškerého testování budou zpracovány v tabulkovém procesoru Excel a uloženy na přiložený kompaktní disk do složky „testovani\_vystupy“.

U všech následujících grafů platí, že osa x označuje aktuální číslo replikace. Jednotky osy y jsou zvlášť označeny pro každé konkrétní měření v legendě grafu.

Jako první probíhá testování nad množinou dat s 5 000 uzly.



Obrázek 19 – Graf doby trvání řešení vlastní implementace Dijkstrova algoritmu pro graf s 5 000 uzly



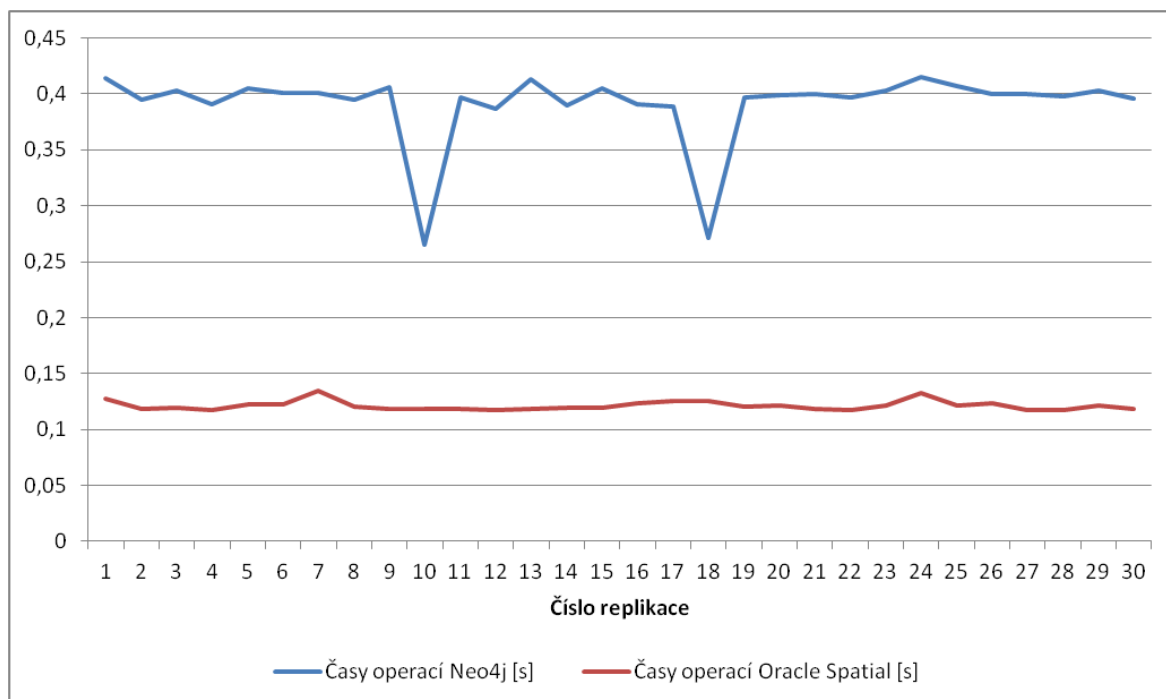
Obrázek 20 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Dijkstrova algoritmu pro graf s 5 000 uzly

Tabulka 7 – Vypočítané hodnoty statistických proměnných pro graf s 5 000 uzly při řešení úlohy vlastní implementací Dijkstrova algoritmu

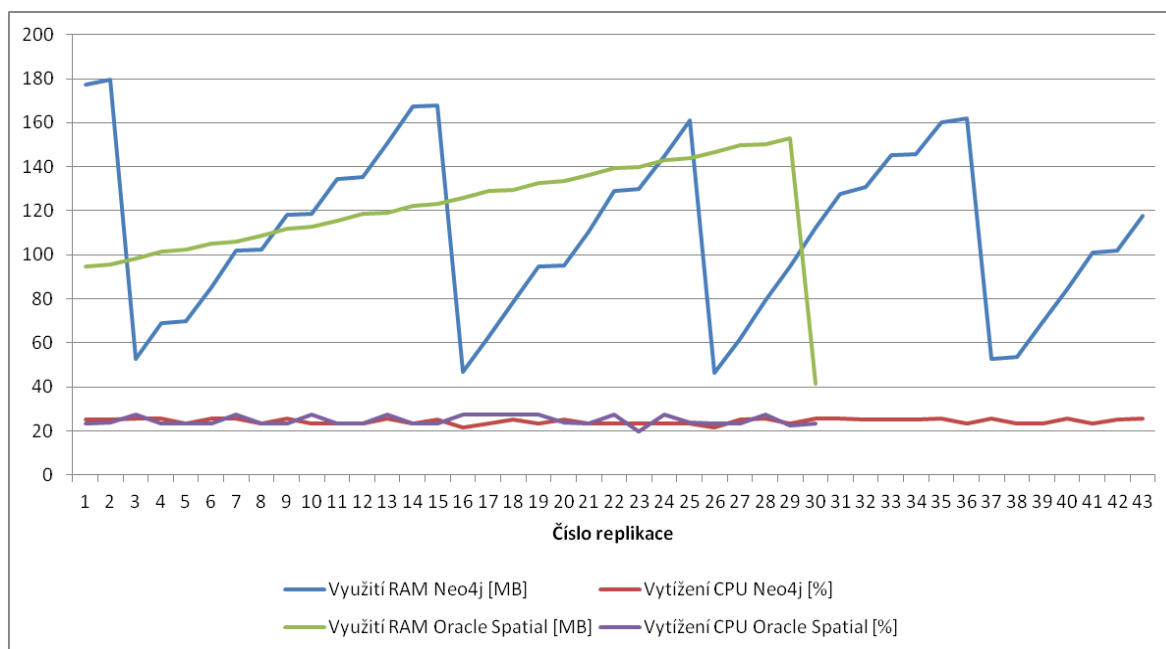
	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
Naměřených hodnot	30	56	56	30	31	31
Průměr	0,71	89,65	24,95	0,20	69,58	26,18
Maximum	0,77	127,80	27,38	0,39	103,45	70,74
Medián	0,74	90,76	24,71	0,19	70,10	23,95
Směrodatná odchylka	0,09	21,43	0,75	0,04	21,11	8,27

Pokračujeme testováním nad množinou dat se 7 500 uzly.





**Obrázek 21 – Graf doby trvání řešení vlastní implementace Dijkstrova algoritmu pro graf se 7 500 uzly**

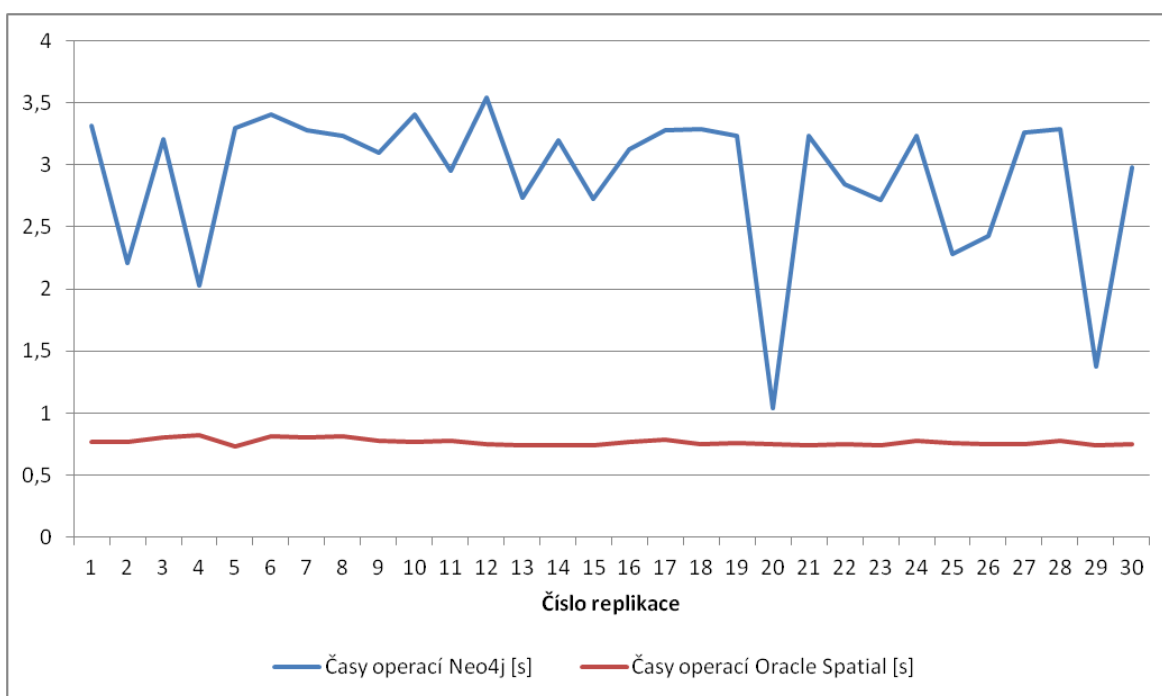


**Obrázek 22 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Dijkstrova algoritmu pro graf se 7 500 uzly**

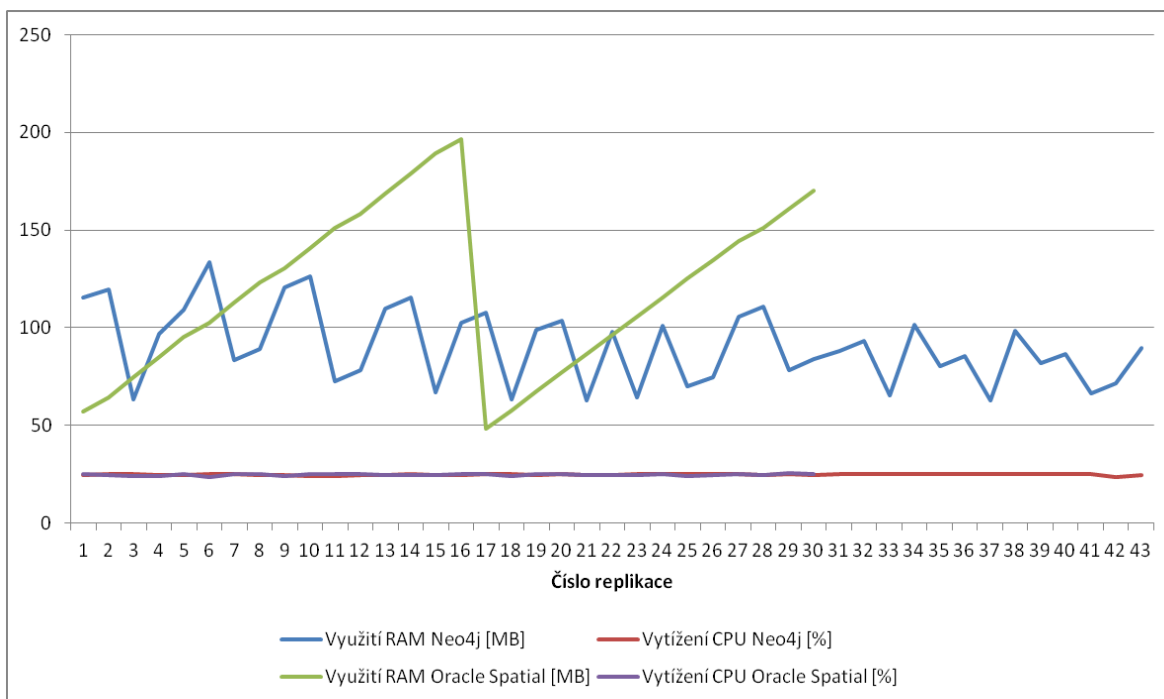
**Tabulka 8 – Vypočítané hodnoty statistických proměnných pro graf se 7 500 uzly při řešení úlohy vlastní implementací Dijkstrova algoritmu**

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
<b>Naměřených hodnot</b>	30	43	43	30	30	30
<b>Průměr</b>	0,39	109,97	24,39	0,12	120,97	24,74
<b>Maximum</b>	0,42	179,36	25,46	0,14	152,90	27,54
<b>Medián</b>	0,40	110,42	25,34	0,12	122,60	23,50
<b>Směrodatná odchylka</b>	0,03	38,48	1,14	0,00	22,73	2,13

Nakonec probíhá testování nad množinou dat s 10 000 uzly.



**Obrázek 23 – Graf doby trvání řešení vlastní implementace Dijkstrova algoritmu pro graf s 10 000 uzly**



**Obrázek 24 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Dijkstrova algoritmu pro graf s 10 000 uzly**

**Tabulka 9 – Vypočítané hodnoty statistických proměnných pro graf s 10 000 uzly při řešení úlohy vlastní implementací Dijkstrova algoritmu**

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
<b>Naměřených hodnot</b>	30	43	43	30	30	30
<b>Průměr</b>	2,91	90,63	24,76	0,77	118,85	24,75
<b>Maximum</b>	3,54	133,41	25,11	0,82	196,34	25,67
<b>Medián</b>	3,20	89,05	24,86	0,76	119,19	24,86
<b>Směrodatná odchylka</b>	0,59	19,30	0,32	0,03	41,74	0,48

### 5.3.4 Vyhodnocení naměřených údajů

Nyní se vyhodnocuje testování na základě výsledků běhů vlastní implementace Dijkstrova algoritmu. Přičemž se vychází z obrázků grafů s čísly 19, 20, 21, 22, 23, 24 a tabulek 7, 8, 9. Z měření lze pozorovat, že rychlost provádění algoritmu je vyšší u databáze Neo4j. Současně se rozdíl průměrně stráveného času na zpracování algoritmu v obou databázích společně s objemem zpracovávaných dat postupně zvětšuje (konkrétně nabývá hodnot 0,48->1,64->4,5 sekund). V případě využití RAM lze říci, že žádný výrazný rozdíl se v průběhu testování mezi databázemi neobjevil. Zjištěné rozdíly průměrných hodnot byly pouze v řádu jednotek MB. Z měření bylo zjištěno, že volba databáze nemá v tomto případě vliv na vytížení CPU, které se průměrně pohybovalo okolo 25 %.

## 5.4 Úloha obchodního cestujícího

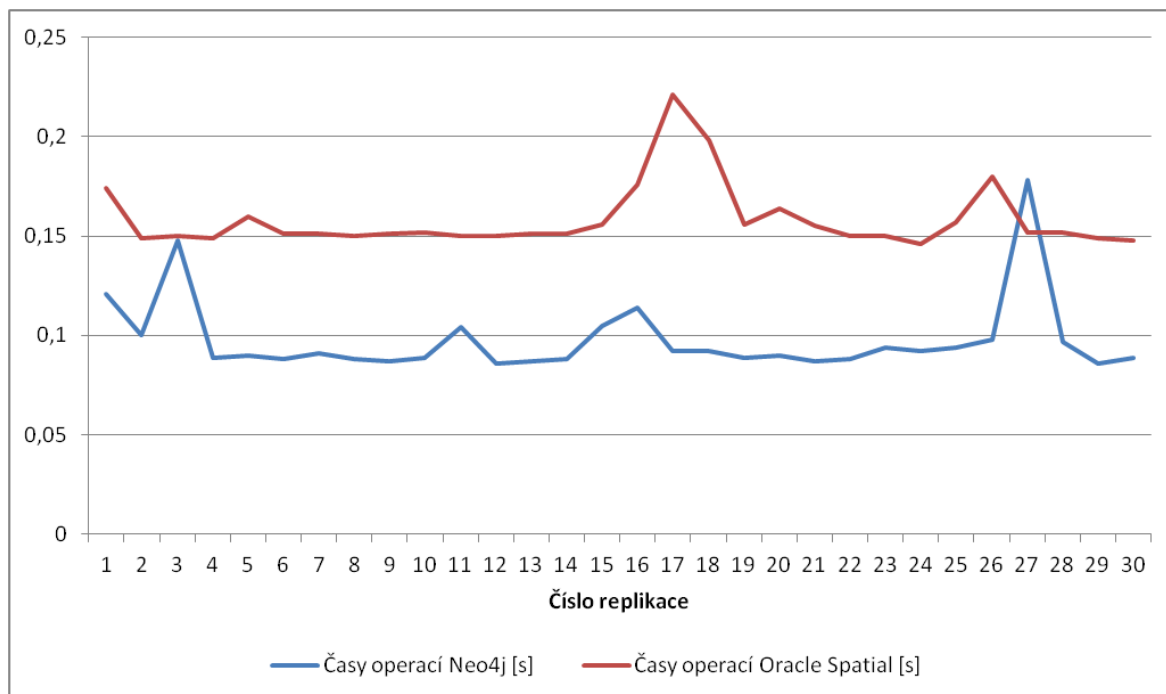
V této kapitole je uvedeno, jaké algoritmy jsou použity pro řešení úlohy obchodního cestujícího při testování databází. Také si bude možné prohlédnout průběh testování a zjistit vyhodnocení výsledků.

### 5.4.1 Spuštěné testy pro vlastní implementaci 2aproximačního algoritmu

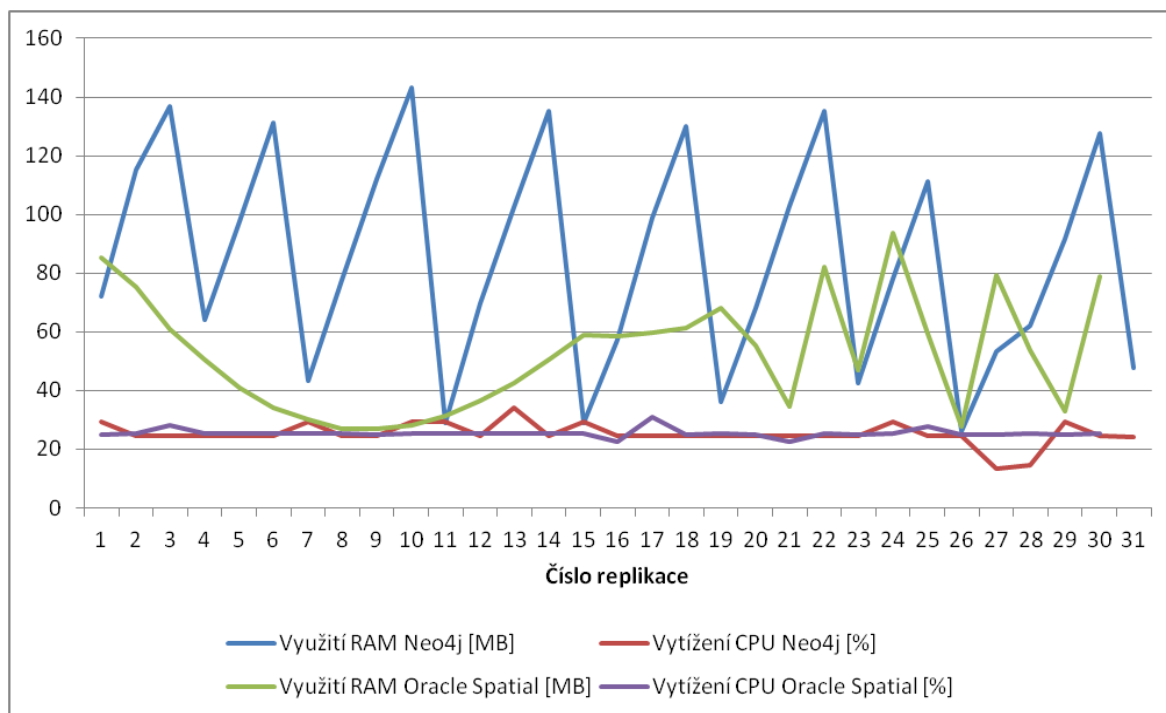
Pro řešení úlohy obchodního cestujícího existují v Neo4j i Oracle Spatial části aplikace připravené 2 možnosti řešení. První možností je zařazovací algoritmus, který vrací přesné řešení úlohy obchodního cestujícího, ovšem nemusí přinést řešení v konečném čase. Tato možnost tedy z tohoto důvodu nebyla testována. Druhou možností je 2aproximační algoritmus, který vrací řešení této úlohy s určitou tolerancí nepřesnosti, ovšem vždy k němu dojde v konečném čase. Bude testováno pouze řešení úlohy vlastní implementací 2aproximačního algoritmu. Testování bude probíhat vždy v 30 replikacích z důvodu potřeby výpočtu statistických veličin, umožňujících vytvářet přesnější závěry. Každý uvedený graf i tabulka bude obsahovat výsledek testování nad danou množinou dat pro oba databázové systémy. Výstupy veškerého testování budou zpracovány v tabulkovém procesoru Excel a uloženy na příložený kompaktní disk do složky „testovani\_vystupy“.

U všech následujících grafů platí, že osa x označuje aktuální číslo replikace. Jednotky osy y jsou zvlášť označeny pro každé konkrétní měření v legendě grafu.

Jako první probíhá testování nad množinou dat s 50 uzly.



Obrázek 25 – Graf doby trvání řešení vlastní implementace 2aproximačního algoritmu pro graf s 50 uzly

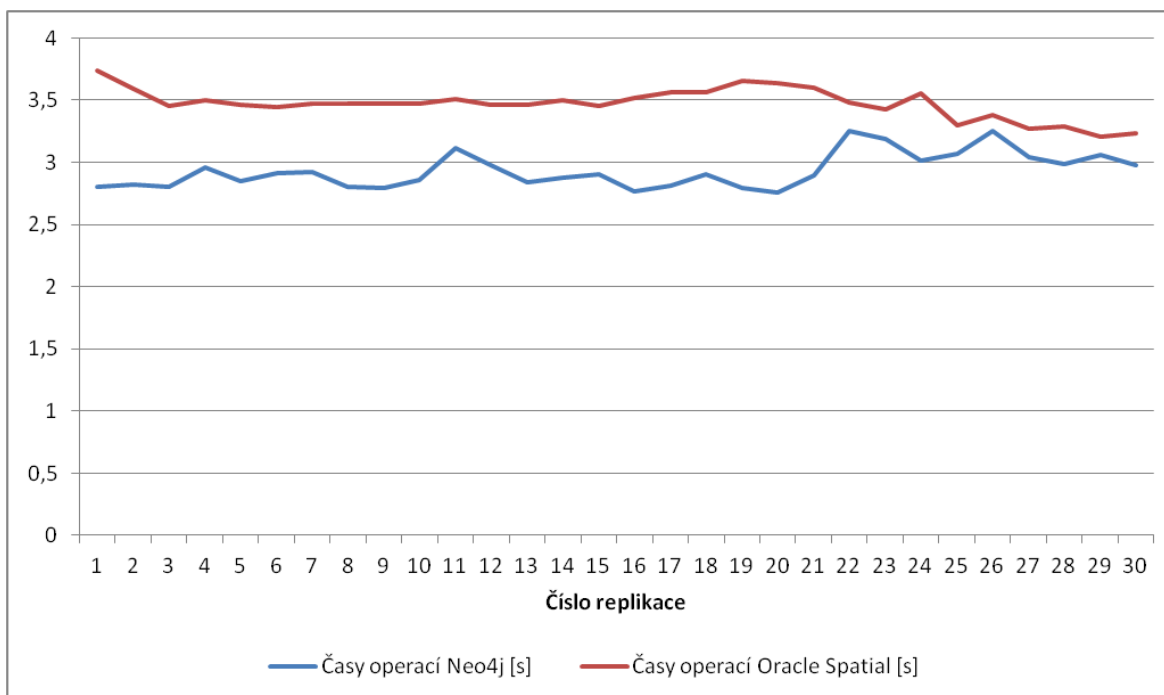


Obrázek 26 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace 2aproximačního algoritmu pro graf s 50 uzly

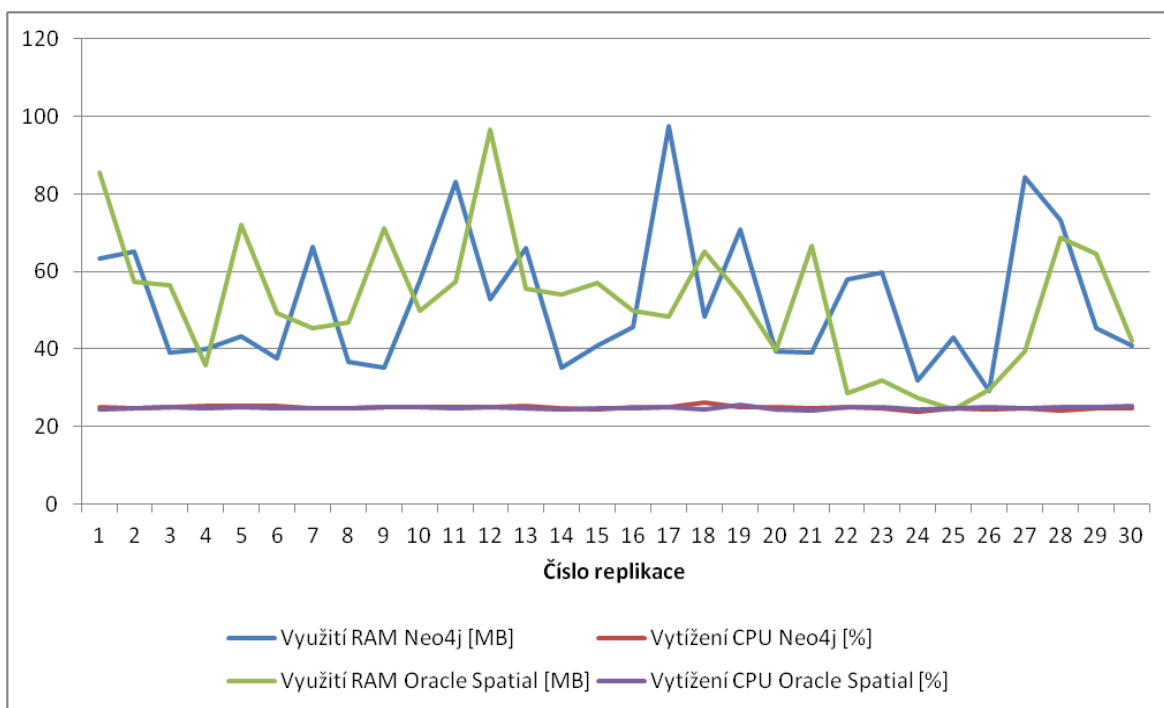
Tabulka 10 – Vypočítané hodnoty statistických proměnných pro graf s 50 uzly při řešení úlohy vlastní implementací 2aproximačního algoritmu

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
Naměřených hodnot	30	30	30	30	30	30
Průměr	0,10	85,99	25,25	0,16	52,33	25,33
Maximum	0,18	143,21	34,09	0,22	93,58	30,73
Medián	0,09	84,80	24,54	0,15	52,21	25,16
Směrodatná odchylka	0,02	36,32	3,97	0,02	19,27	1,43

Pokračujeme testováním nad množinou dat se 100 uzly.



**Obrázek 27 – Graf doby trvání řešení vlastní implementace 2aproximačního algoritmu pro graf se 100 uzly**

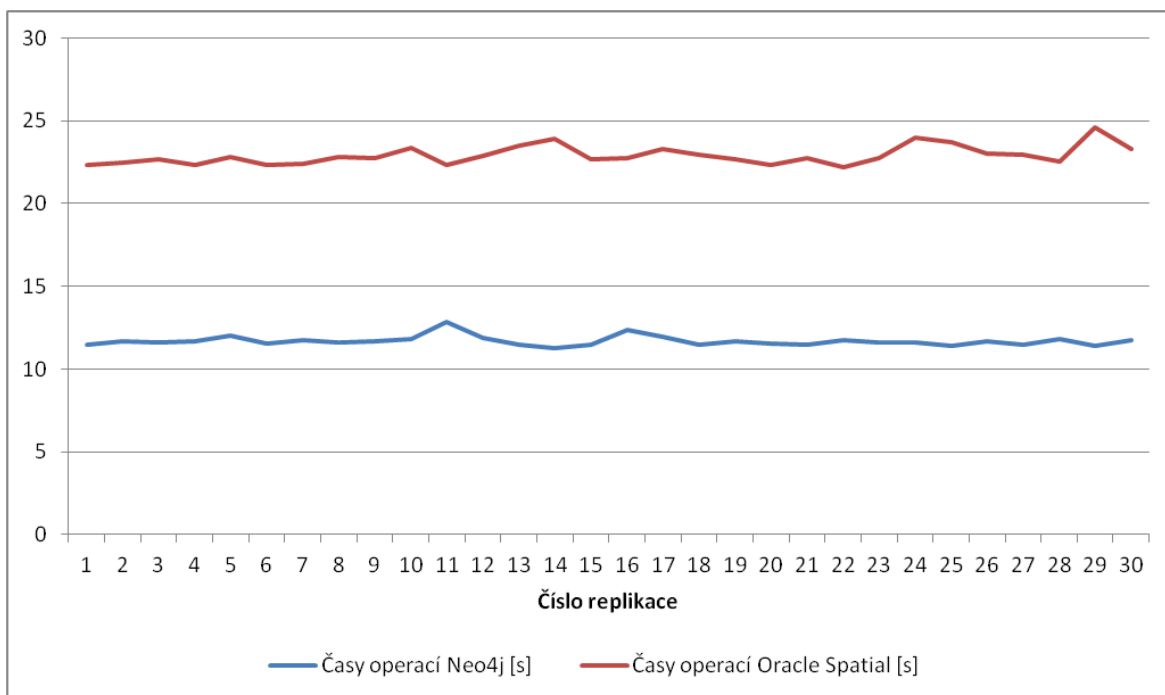


**Obrázek 28 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace 2aproximačního algoritmu pro graf se 100 uzly**

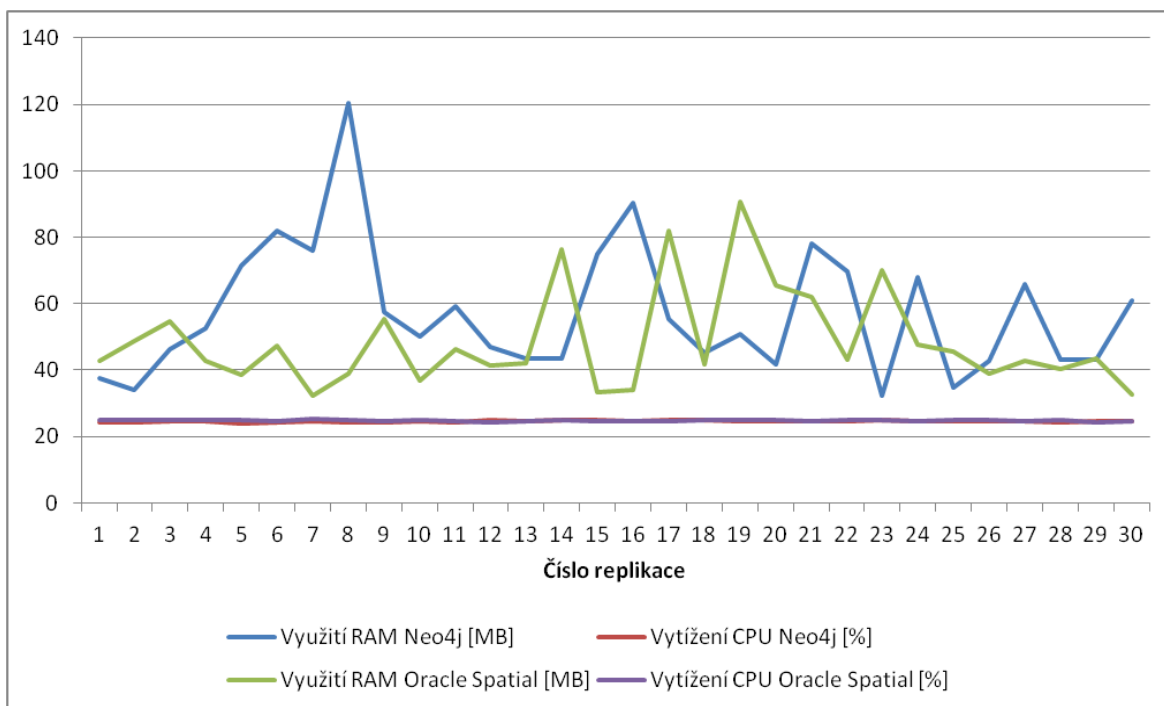
**Tabulka 11 – Vypočítané hodnoty statistických proměnných pro graf se 100 uzly při řešení úlohy vlastní implementací 2aproximačního algoritmu**

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
<b>Naměřených hodnot</b>	30	30	30	30	30	30
<b>Průměr</b>	2,93	52,29	24,87	3,47	52,33	24,79
<b>Maximum</b>	3,25	97,50	26,14	3,73	96,49	25,61
<b>Medián</b>	2,90	45,47	24,98	3,47	51,95	24,83
<b>Směrodatná odchylka</b>	0,14	17,15	0,44	0,12	16,75	0,29

Nakonec probíhá testování nad množinou dat se 150 uzly.



**Obrázek 29 – Graf doby trvání řešení vlastní implementace 2aproximačního algoritmu pro graf se 150 uzly**



Obrázek 30 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace 2aproximačního algoritmu pro graf se 150 uzly

Tabulka 12 – Vypočítané hodnoty statistických proměnných pro graf se 150 uzly při řešení úlohy vlastní implementací 2aproximačního algoritmu

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
Naměřených hodnot	30	30	30	30	30	30
Průměr	11,69	57,28	24,60	22,91	48,65	24,80
Maximum	12,85	120,31	25,13	24,58	90,62	25,22
Medián	11,64	51,74	24,62	22,77	43,07	24,84
Směrodatná odchylka	0,30	19,24	0,26	0,56	14,62	0,19

#### 5.4.2 Vyhodnocení naměřených údajů

Následuje vyhodnocení testování na základě výsledků běhů vlastní implementace 2aproximačního algoritmu. Přičemž se vychází z obrázků grafů s čísly 25, 26, 27, 28, 29, 30 a tabulek 10, 11, 12. Z měření lze pozorovat, že rychlost provádění algoritmu je vyšší u databáze Neo4j. Současně se rozdíl průměrně stráveného času na zpracování algoritmu v obou databázích společně s objemem zpracovávaných dat postupně zvětšuje (konkrétně nabývá hodnot 0,06->0,54->11,22 sekund). V případě využití RAM lze říci, že větší rozdíl se objevil pouze při testování množiny o velikosti 50 uzlů. V uvedeném případě bylo u databáze Neo4j využití RAM průměrně o 33,66 MB vyšší. Z měření bylo zjištěno, že volba databáze nemá v tomto případě vliv na vytížení CPU, které se průměrně pohybovalo okolo 25 %.



## 5.5 Hledání minimální kostry grafu

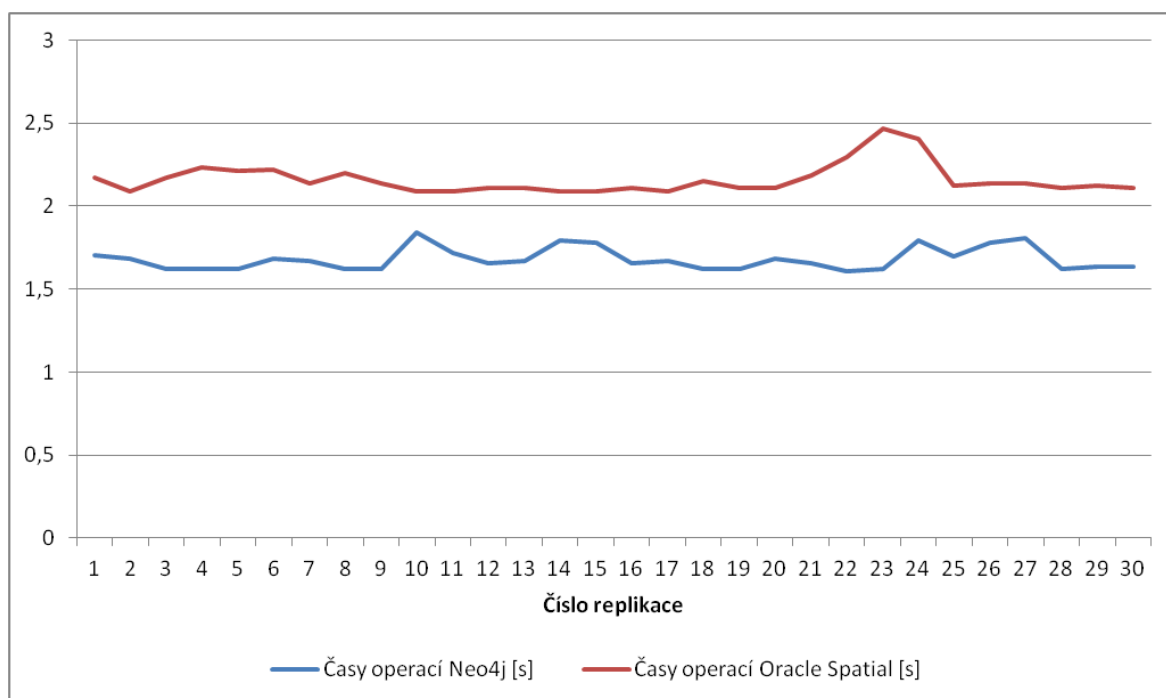
V této kapitole je uvedeno, jaké algoritmy jsou použity pro hledání minimální kostry grafu při testování databází. Také si bude možné prohlédnout průběh testování a zjistit vyhodnocení výsledků.

### 5.5.1 Spuštěné testy pro vlastní implementaci Kruskalova algoritmu

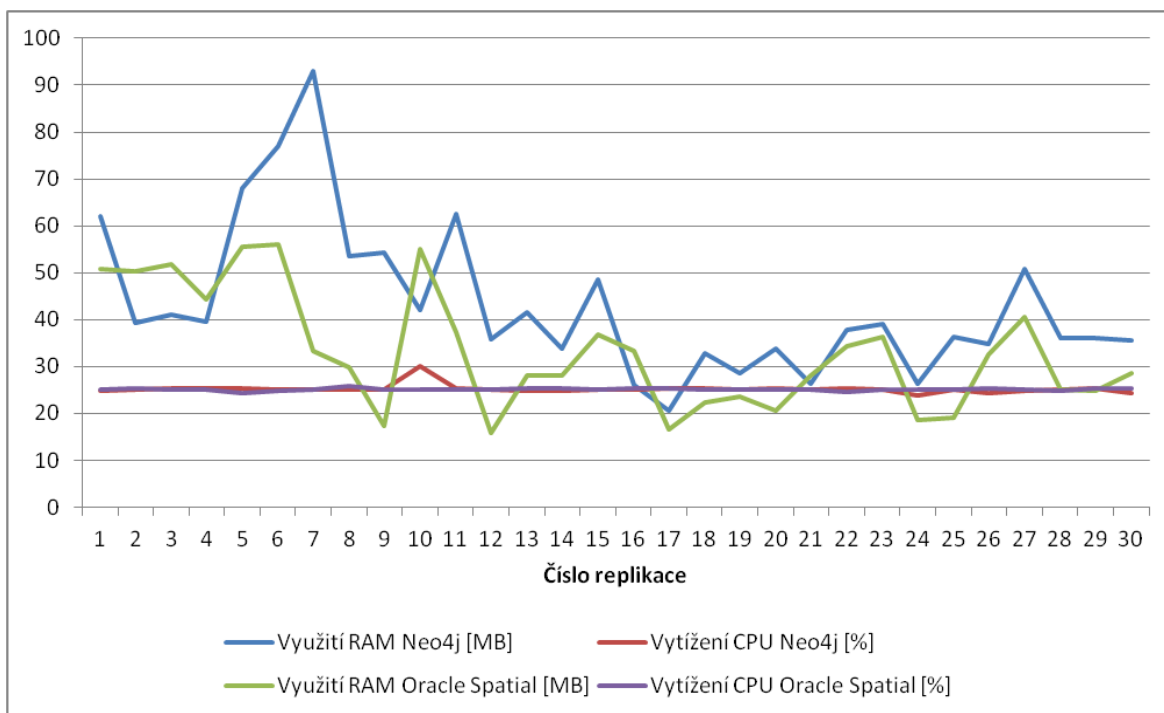
Pro řešení úlohy hledání minimální kostry grafu je v Neo4j i Oracle Spatial části aplikace připraven 1 společný algoritmus. Bohužel z Java API rozhraní obou databází obsahuje předpřipravené řešení této úlohy pouze Oracle Spatial. Dojde tedy k otestování vlastní implementace Kruskalova algoritmu. Testování bude probíhat vždy v 30 replikacích z důvodu potřeby výpočtu statistických veličin, umožňujících vytvářet přesnější závěry. Každý uvedený graf i tabulka bude obsahovat výsledek testování nad danou množinou dat pro oba databázové systémy. Výstupy veškerého testování budou zpracovány v tabulkovém procesoru Excel a uloženy na přiložený kompaktní disk do složky „testovani\_vystupy“.

U všech následujících grafů platí, že osa x označuje aktuální číslo replikace. Jednotky osy y jsou zvlášť označeny pro každé konkrétní měření v legendě grafu.

Jako první probíhá testování nad množinou dat s 500 uzly.



Obrázek 31 – Graf doby trvání řešení vlastní implementace Kruskalova algoritmu pro graf s 500 uzly

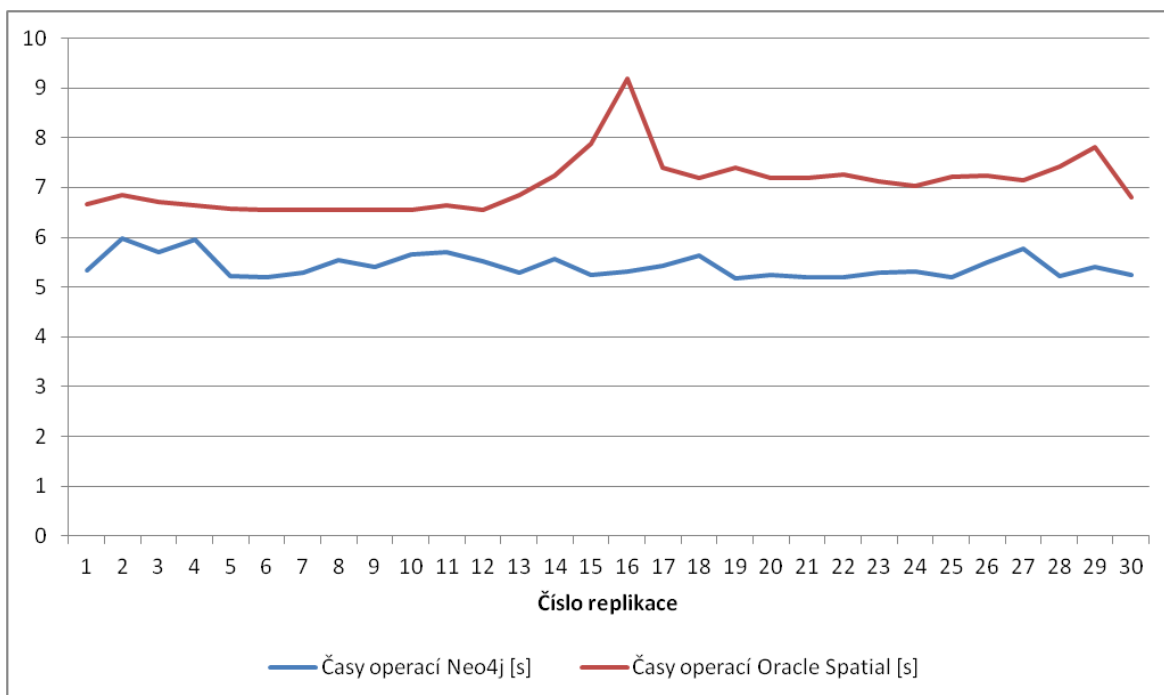


**Obrázek 32 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Kruskalova algoritmu pro graf s 500 uzly**

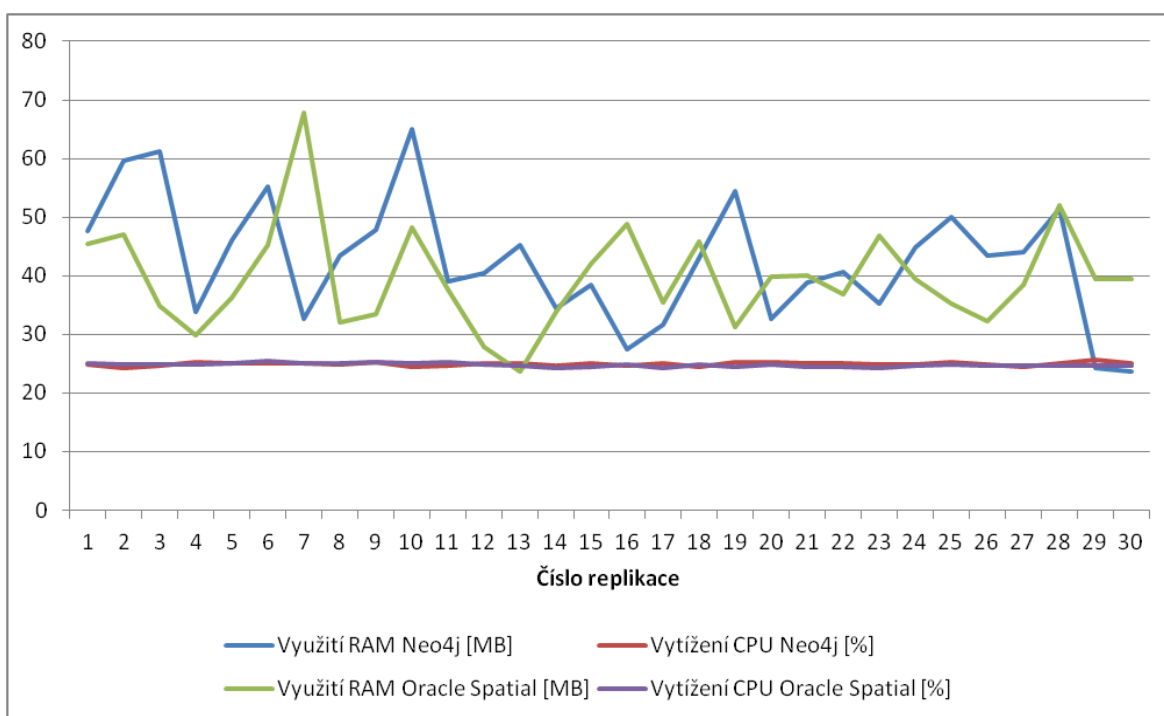
**Tabulka 13 – Vypočítané hodnoty statistických proměnných pro graf s 500 uzly při řešení úlohy vlastní implementací Kruskalova algoritmu**

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
Naměřených hodnot	30	30	30	30	30	30
Průměr	1,68	43,10	25,18	2,16	33,19	25,13
Maximum	1,84	93,00	29,97	2,47	56,12	25,90
Medián	1,66	38,42	25,07	2,13	31,25	25,14
Směrodatná odchylka	0,07	15,88	0,94	0,09	12,30	0,26

Pokračujeme testováním nad množinou dat se 750 uzly.



**Obrázek 33 – Graf doby trvání řešení vlastní implementace Kruskalova algoritmu pro graf se 750 uzly**

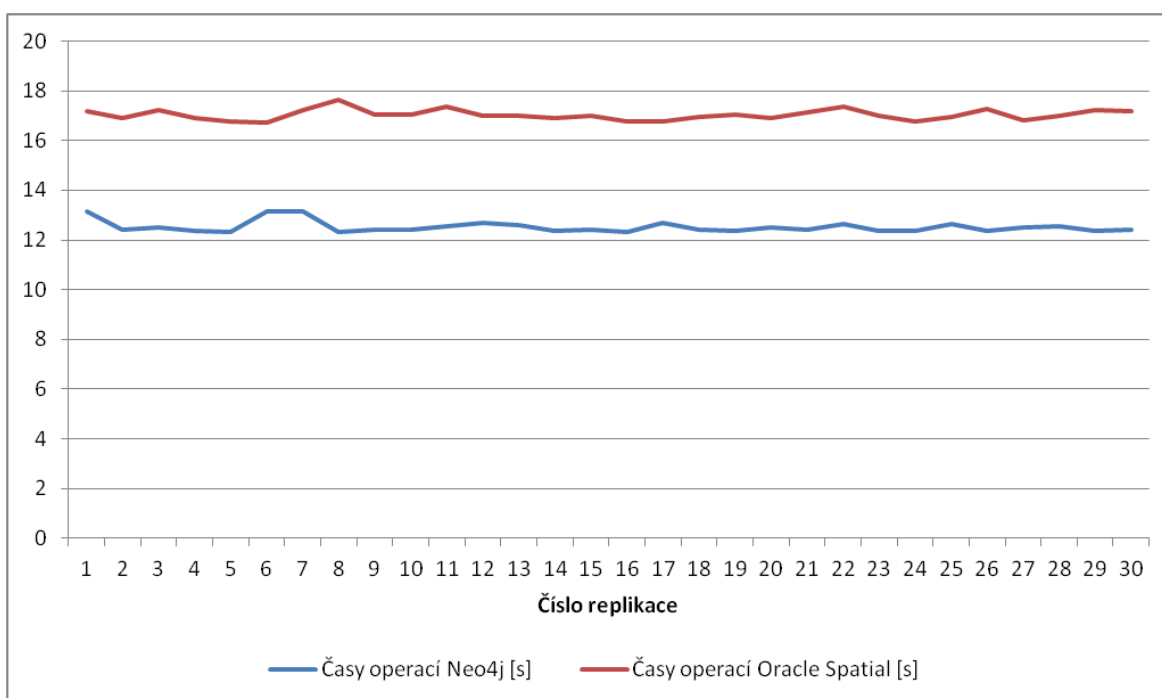


**Obrázek 34 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Kruskalova algoritmu pro graf se 750 uzly**

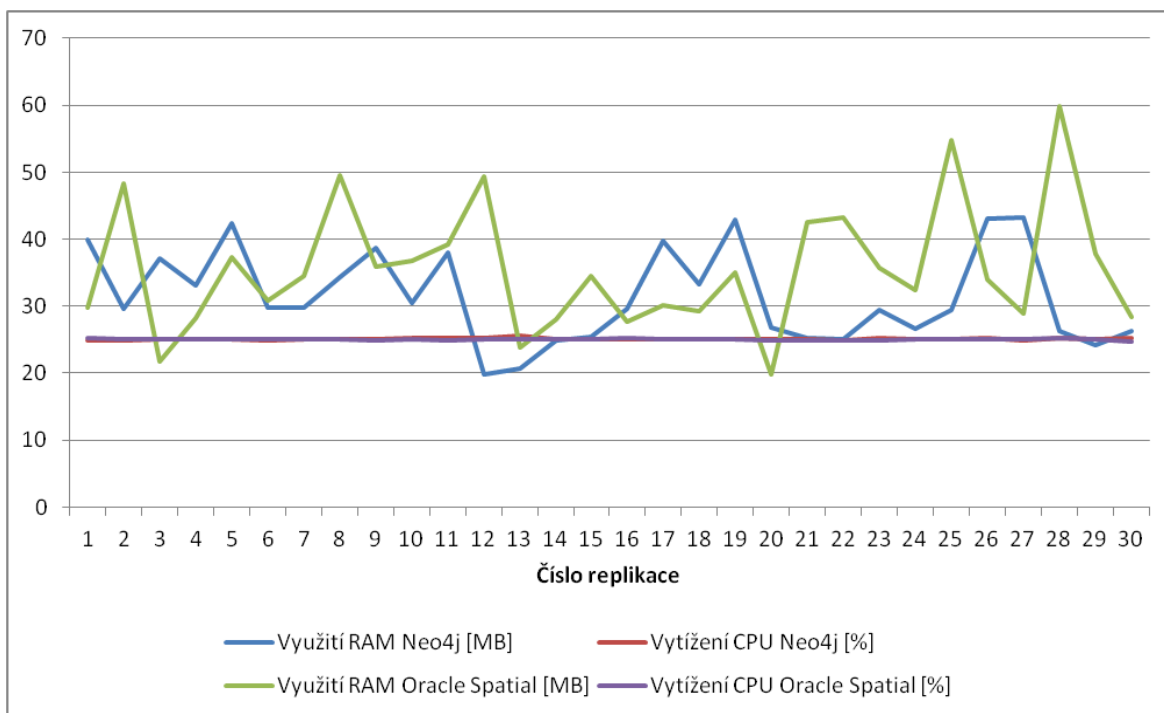
**Tabulka 14 – Vypočítané hodnoty statistických proměnných pro graf se 750 uzly při řešení úlohy vlastní implementací Kruskalova algoritmu**

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
<b>Naměřených hodnot</b>	30	30	30	30	30	30
<b>Průměr</b>	5,42	42,53	24,93	7,07	39,56	24,77
<b>Maximum</b>	5,98	64,96	25,57	9,18	67,74	25,50
<b>Medián</b>	5,33	43,15	24,96	7,08	39,00	24,78
<b>Směrodatná odchylka</b>	0,23	10,22	0,28	0,54	8,45	0,29

Nakonec probíhá testování nad množinou dat s 1 000 uzly.



**Obrázek 35 – Graf doby trvání řešení vlastní implementace Kruskalova algoritmu pro graf s 1 000 uzly**



Obrázek 36 – Graf využití RAM a vytížení CPU v průběhu řešení vlastní implementace Kruskalova algoritmu pro graf s 1 000 uzly

Tabulka 15 – Vypočítané hodnoty statistických proměnných pro graf s 1 000 uzly při řešení úlohy vlastní implementací Kruskalova algoritmu

	Neo4j			Oracle Spatial		
	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]	Časy operací [s]	Využití RAM [MB]	Vytížení CPU [%]
Naměřených hodnot	30	30	30	30	30	30
Průměr	12,53	31,52	25,09	17,03	35,57	25,04
Maximum	13,17	43,27	25,56	17,63	59,93	25,29
Medián	12,43	29,72	25,08	16,98	34,45	25,03
Směrodatná odchylka	0,23	6,80	0,13	0,21	9,35	0,11

### 5.5.2 Vyhodnocení naměřených údajů

Nyní dojde k vyhodnocení testování na základě výsledků běhů vlastní implementace Kruskalova algoritmu. Budeme vycházet z obrázků grafů s čísly 31, 32, 33, 34, 35, 36 a tabulek 13, 14, 14. Z měření lze pozorovat, že rychlost provádění algoritmu je vyšší u databáze Neo4j. Současně se rozdíl průměrně stráveného času na zpracování algoritmu v obou databázích společně s objemem zpracovávaných dat postupně zvětšuje (konkrétně nabývá hodnot 0,48->1,64->4,5 sekund). V případě využití RAM lze říci, že žádný výrazný rozdíl se v průběhu testování mezi databázemi neobjevil. Zjištěné rozdíly průměrných hodnot byly pouze v řádu jednotek MB. Z měření bylo zjištěno, že volba databáze nemá v tomto případě vliv na vytížení CPU, které se průměrně pohybovalo okolo 25 %.

## Závěr

V této práci byly nejprve objasněny vybrané pojmy z oblasti teorie grafů. Dále došlo k popisu a přehlednému porovnání předností a nedostatků zvolených grafových databází. Stejným způsobem byly popsány vybrané relační databáze umožňující uchovávání grafů společně s představením jejich kladů a záporů.

Na základě teoretických poznatků a konzultací s vedoucím práce došlo k návrhu systému pro testování, který byl později implementován. Hlavní předností je nezávislost na bázi dat pro účely testování. Tohoto cíle se podařilo dosáhnout díky implementaci vlastního generátoru bází dat z různých oblastí reálných problematik. Veškeré požadavky na systém, pro účely testování, byly splněny.

Testování proběhlo na 4 úlohách. Z toho 3 vycházely z algoritmů vlastní implementace a 1 z algoritmů Java API obou databází. V případě použití algoritmů z Java API rozhraní databází nebyl zjištěn významný rozdíl v rychlosti provedení algoritmu ani využití RAM. Při použití algoritmů vlastní implementace, se objeví patrné rozdíly mezi oběma databázemi a to v rychlosti zpracování. Na základě testování lze říci, že databáze Neo4j zpracovává algoritmy vlastní implementace rychleji. Přičemž rozdíl v rychlosti se zvětšuje současně s nárůstem prvků používané báze dat. Dále testování ukázalo, že obě databáze při zpracování algoritmů vlastní implementace nevykazují významné rozdíly ve využití RAM. Pokud se nějaké rozdíly objevily, pak se průměrně jednalo o jednotky MB. Z pohledu vytížení CPU, dosahovaly obě testované databáze obdobných výsledků. Také se ukázalo, že v případě volby databáze pro zpracování menší datové množiny v řádu desítek uzlů nejsou mezi oběma databázemi patrné velké rozdíly ve výkonu.

Nejsložitějším problémem byla implementace konverze grafu z Neo4j do Oracle Spatial. Díky práci na tomto tématu došlo k prohloubení mých znalostí zejména v oblasti grafových databází.

Výsledky testování by mohli v budoucnu posloužit jako podklad při rozhodování, jakou databázi zvolit při uchovávání a následné práci s konkrétní bází grafových dat. Jako velmi zajímavé se do budoucna jeví možnost doimplementování dalších algoritmů řešících problém minimální kostry grafu (např. Jarníkův, Borůvkův a kontraktivní algoritmus). A to především z důvodů testování databází v rámci řešení jednoho problému různými způsoby. Samotnou aplikaci by bylo také možné rozšířit o algoritmy z jiné oblasti problematiky např. hledání maximálního toku v sítích.

## Použité zdroje

- [1] **Ian, Robinson; Emil Eifrem; Jim Webber:** Graph Databases. Sebastopol: O'Reilly Media, Inc., 2013. ISBN: 978-1-4493-5626-2.
- [2] **Steven, S. Skiena:** The Algorithm Design Manual. Second edition. New York: Springer, 2008. ISBN: 978-1-84800-069-8.
- [3] **Kothuri, Ravi; Albert Godfrind; Euro Beinat:** Pro Oracle Spatial for Oracle Database 11g. Berkeley: Apress, 2007. ISBN: 978-1-59059-899-3.
- [4] **Thomas, Kyte:** Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions. Berkeley: Apress, 2010. ISBN: 978-1-4302-2946-9.
- [5] Neo4j: The World's Leading Graph Database [online]. Neo Technology, Inc., c2014 [cit. 2014-04-16]. Dostupné z WWW: <<http://www.neo4j.org/>>.
- [6] TinkerPop: GitHub [online]. GitHub, Inc., c2014 [cit. 2014-04-16]. Dostupné z WWW: <<http://github.com/tinkerpop/>>.
- [7] Twitter/flockdb: GitHub [online]. GitHub, Inc., c2014 [cit. 2014-04-16]. Dostupné z WWW: <<https://github.com/twitter/flockdb>>.
- [8] Introducing FlockDB: Twitter Blogs [online]. Twitter, Inc., c2014 [cit. 2014-04-16]. Dostupné z WWW: <<https://blog.twitter.com/2010/introducing-flockdb>>.
- [9] AllegroGraph RDFStore Web 3.0's Database [online]. Franz Inc., c2014 [cit. 2014-04-16]. Dostupné z WWW: <<http://franz.com/agraph/allegrograph/>>.
- [10] OrientDB: Orient Technologies – OrientDB Distributed Graph DatabaseOrient Technologies; OrientDB Distributed Graph Database [online]. Orient Technologies LTD, c2014, poslední aktualizace March 31, 2014 [cit. 2014-04-16]. Dostupné z WWW: <<http://www.orienttechnologies.com/orientdb/>>.
- [11] Thinkaurelius/titan Wiki: GitHub [online]. GitHub, Inc., c2014 [cit. 2014-04-16]. Dostupné z WWW: <<https://github.com/thinkaurelius/titan/wiki>>.
- [12] PostGIS: Spatial and Geographic Objects for PostgreSQL [online]. PostGIS Project Steering Committee, c2001, poslední aktualizace 2014/03/31 [cit. 2014-05-01]. Dostupné z WWW: <<http://postgis.net/>>.
- [13] pgRouting Project: Open Source Routing Library [online]. pgRouting Community, c2007 [cit. 2014-05-04]. Dostupné z WWW: <<http://pgrouting.org/>>.

- [14] **Daniel Geringer:** Developing Network Analysis Applications Using the Oracle Spatial Network Data Model. April 29, 2010, [cit. 2014-04-16]. Dostupný z WWW:  
<[http://download.oracle.com/otndocs/products/spatial/pdf/osuc2010\\_presentations/osuc2010\\_ndm\\_geringer.pdf](http://download.oracle.com/otndocs/products/spatial/pdf/osuc2010_presentations/osuc2010_ndm_geringer.pdf)>.
- [15] **International Business Machines Corporation:** IBM DB2 Spatial Extender: User's Guide and Reference. Version 8. IBM Corp., 1998, [cit. 2014-05-05]. Dostupný z WWW:  
<<ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/pdf/letter/db2sbe80.pdf>>.
- [16] **Mička, Pavel:** Algoritmy.net [online]. 2008, poslední modifikace 2014 [cit. 2014-04-26]. Dostupné z WWW: <<http://www.algoritmy.net>>.
- [17] **Večerka, Arnošt:** Grafy a grafové algoritmy [online]. 2007 [cit. 2014-04-26]. Dostupné z WWW:  
<[http://phoenix.inf.upol.cz/esf/ucebni/Grafy\\_a\\_grafove\\_algoritmy.pdf](http://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf)>.
- [18] **Bečvář, Milan:** Modelování v grafové nerelační databázi. Plzeň, 2013. Dostupné z WWW:  
<[https://otik.uk.zcu.cz/bitstream/handle/11025/7596/diplomova\\_prace.pdf](https://otik.uk.zcu.cz/bitstream/handle/11025/7596/diplomova_prace.pdf)>.  
Diplomová práce. Západočeská univerzita v Plzni. Vedoucí práce Ing. Roman Moučka, Ph.D.
- [19] Oracle: Hardware and Software, Engineered to Work Together [online]. Oracle Corporation and/or its affiliates, c2014, 8/8/14 11:11 AM [cit. 2014-08-12]. Dostupné z WWW: <<http://www.oracle.com/index.html>>.
- [20] The Neo4j Manual v2.0.4 - [online]. Neo Technology, Inc., c2014, 2014-07-07 18:48 [cit. 2014-08-12]. Dostupné z WWW:  
<<http://docs.neo4j.org/chunked/2.0.4/index.html>>.
- [21] Working and connecting with databases [online]. Oracle Corporation and/or its affiliates, c2013, 2013-11-19 [cit. 2014-08-12]. Dostupné z WWW:  
<[http://docs.oracle.com/cd/E40938\\_01/doc.74/e40142/work\\_dbases.htm#NBDAG1790](http://docs.oracle.com/cd/E40938_01/doc.74/e40142/work_dbases.htm#NBDAG1790)>.
- [22] **Pavel Houser:** Trable obchodního cestujícího: CIO Business World.cz. 11. 5. 2012, [cit. 2014-08-12]. Dostupný z WWW: <<http://businessworld.cz/cio-bw-special/trable-obchodniho-cestujiciho-8954>>.



## **Příloha A – kompaktní disk obsahující zdrojové kódy aplikace**

Příložený kompaktní disk obsahuje testovací aplikaci, její zdrojové kódy, datové soubory použité při testování a text diplomové práce ve formátu PDF.