Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Vyhledávání expertů zadaného oboru z bibliografických vědeckých databází

Plzeň 2015 Radek Šmolík

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 12. května 2015

Radek Šmolík

Abstract

Searching experts the assign field of bibliographic an scientific databases.

The aim of this work is to create a program that searches the list of publications in *Web of Science* database.

The work shows the possibilities of searching text in documents by different methods and describes basic methods of working with text in natural languague.

My solution is based on using open source Apache Lucene library that creates index of data and then allows searches. The input query can use ontology and text lemmatization.

From the results is created citation network of authors. This citation network is ranked and evaluated with the algorithm PageRank and displayed to user like a page in the web browser.

Citation network of authors is also evaluated with Indegree algorithm and the results are compared with the PageRank algorithm.

Obsah

1 Úvod

Cílem této práce je vytvoření aplikace, která na základě dotazu specifikující téma, vyhledá v databázi *Web Of Science* seznam publikací, vázající se k danému tématu. Z autorů nalezených publikací se vytvoří citační síť, na kterou se dále aplikují algoritmy hodnocení prestižnosti jednotlivých autorů, určující pořadí zobrazení ve výsledcích.

Výstupem práce je aplikace, umožňující uživateli vyhledat nejprestižnější autory a publikace, zabývající se hledanou tématikou.

K vytvoření programu je použit programovací jazyk *JAVA* ve verzi *Enterprise Edition* a jedná se o webovou aplikaci. Základními kroky programu jsou indexace zdrojových dat, vyhledávání relevantních výsledků v indexu dat na základě různě modifikovaného dotazu a v závislosti na citacích z dat, výpočet ohodnocení jednotlivých výsledků.

Výsledky programu jsou zobrazeny uživateli ve webovém prohlížeči formou seznamu autorů s publikacemi.

2 Vyhledávání vzorků v textu

Hledání přesného vzorku textu v dokumentech patří mezi nejběžnější způsoby hledání. Jako uživatelé počítačů se s ním setkáváme dnes a denně. Např. máme před sebou otevřený dokument a potřebujeme v něm nalézt slovo, či větu.

Vyhledávání textu je operace, jejíž výsledkem je, zda hledaný text (vzorek) je součástí prohledávaného textu a pokud ano, zajímá nás, kde se nalezená informace nachází. Pro porovnání způsobu vyhledávání, použitého v této práci (kapitola 3), jsou v následující podkapitole 2.1 uvedeny základní metody nalezení vzorku v textu.

2.1 Algoritmy vyhledávání vzorku v textu

V následujících kapitolách bude vysvětleno několik algoritmů vyhledávání. Aby bylo možné je spolu vzájemně porovnat, bude pro každý algoritmus ukázáno vyhledávání stejného vzorku textu na stejných datech [1].

Prohledávaný text:

```
abccbabcabbcaabccbabcbbbabc
```

Vzorek:

```
abccbabcbbb
```

2.1.1 Hrubá síla

Algoritmus hledání hrubou silou patří mezi ty nejjednodušší, co se týče implementace. Hledaný vzorek textu je postupně porovnáván se všemi pozicemi v prohledávaném textu. Neprobíhá zde žádné předzpracování vzorku textu, ani prohledávaného textu. Velkou nevýhodou je časová náročnost algoritmu, která se odvíjí od velikosti prohledávaných dat a vzorku.

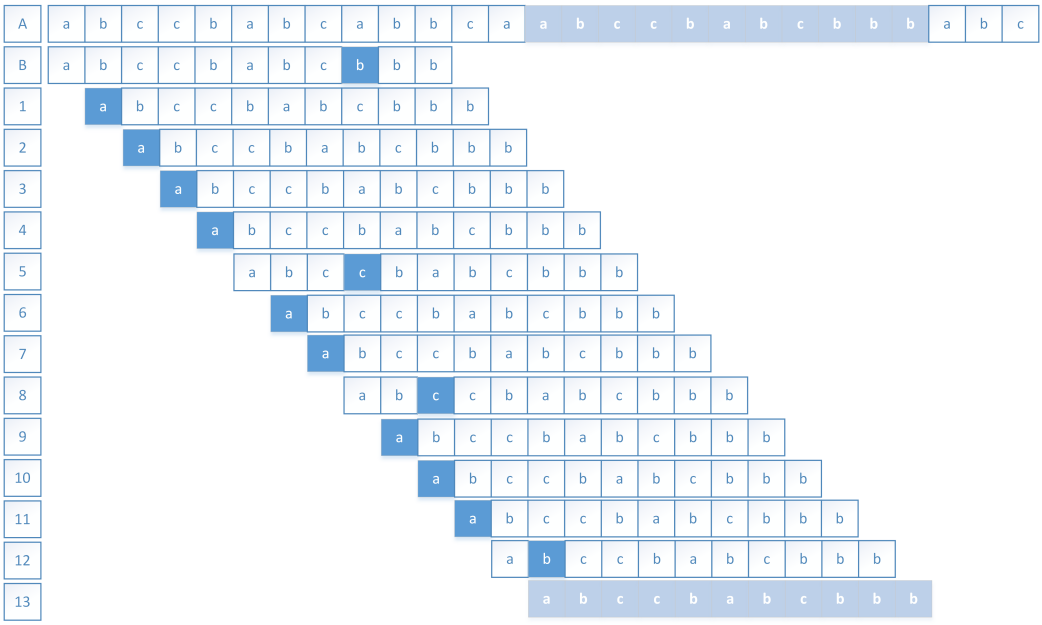
Nejhorší možný případ asymptotické složitosti[1] algoritmu hrubé síly pro

- m – délka vzorku
- n – délka prohledávaného textu

je $O(mn)$, avšak průměrná asymptotická složitost algoritmu je rovna $O(n)$.

Ostatní algoritmy vyhledávání se snaží tuto asymptotickou složitost a tedy i dobu nalezení vzorku textu co nejvíce snížit. Používají k tomu techniky pro předzpracování prohledávaného textu i samotného vzorku textu. Předzpracováním se zkoumá struktura textu a dle něj je vytvořen odpovídající vyhledávací algoritmus, více v následujících podkapitolách 2.1.2, 2.1.3.

Ukázka vyhledávání vzorku v textu pro algoritmus hrubé síly na obrázku 2.1 je převzat z [1] a následně kompletně dopracován.



Obrázek 2.1: Ukázka algoritmu hrubá síla

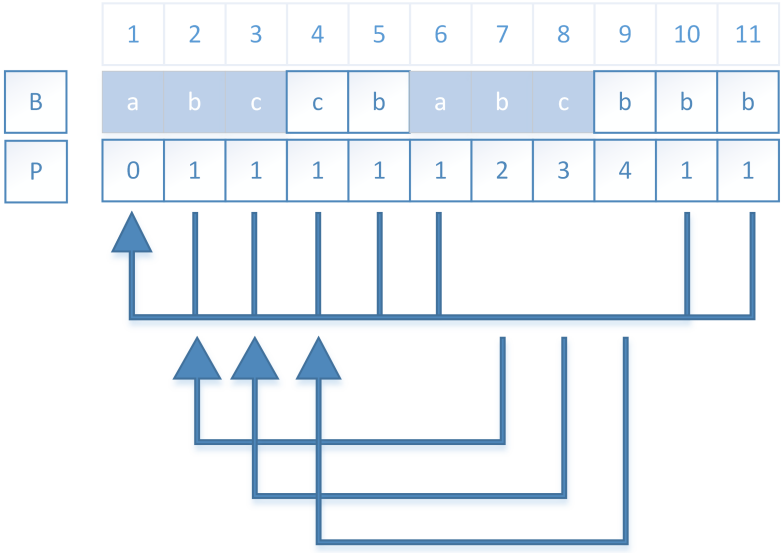
[1] Asymptotická složitost udává závislost algoritmu na velikosti vstupních dat, tj. jak bude algoritmus rychlý pro určitou množinu vstupních dat.

Na obrázku 2.1 je na pozici A prohledávaný text, pozice B je hledaný vzorek textu. Porovnávat se začínají první znaky textu a vzorku zleva. V tomto případě jsou oba znaky a, je shoda a porovnávají se znaky o jednu pozici vpravo. V případě neshody porovnávaných znaků dochází k posuvu vzorku o jednu pozici doprava a porovnávání s textem začíná opět od prvního znaku vzorku. Na obrázku jsou znaky neshody tmavě zvýrazněny. Dojde-li se až na konec vzorku a všechny jeho znaky se s prohledávaným textem shodují, máme nalezenu shodu.

Jak je vidět na příkladu 2.1, nalezení vzorku v prohledávaném textu vyžadovalo celkem *13* krát posunutí vzorku. V podkapitole 2.2 je uvedeno srovnání s dalšími algoritmy vyhledávání.

2.1.2 Knuth-Morris-Prattův algoritmus

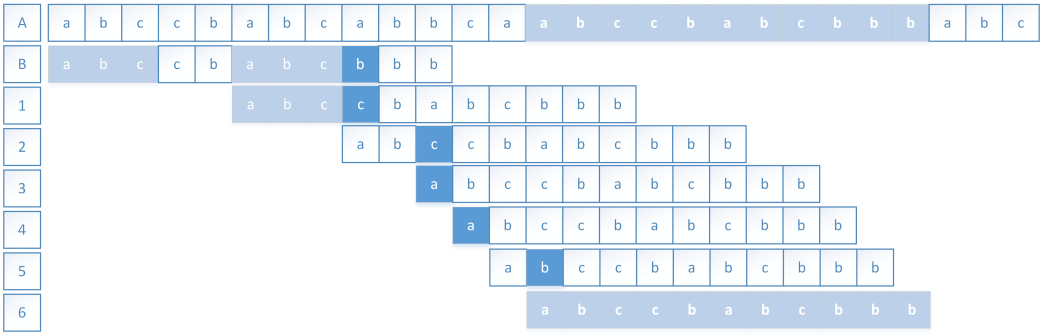
KMP (Knuth-Morris-Prattův) algoritmus se snaží na rozdíl od hrubé síly neprohledávat pozice, které již byly prohledány. V případě neshody se vzorek neposouvá o jeden znak, jak tomu bylo u hrubé síly, ale dochází k posunu vzorku doprava o tolik pozic, kolik znaků je v příponě prohledávaného textu a předponě vzorku shodných. Na obrázku 2.2 [1] je ukázka výpočtu posuvu vzorku v případě nalezení neshody znaků.



Obrázek 2.2: Ukázka výpočtu posuvu KMP algoritmu

Řádka B obsahuje vzorek textu a P obsahuje posuvy v případě neshody znaků při porovnávání. Pro vysvětlení si vezmeme znak b na $7.$ pozici vzorku. Algoritmus hrubé síly by v případě neshody vzal vzorek a posunul by jej z původní pozice o jednu pozici vpravo a pokračoval s porovnáváním od jeho prvního znaku. Algoritmus KMP však posune vzorek na jeho $2.$ pozici, neboť z předzpracování víme, že pro aktuální porovnávanou pozici existuje ve vzorku stejná předpona a přípona znaků. Právě proto dochází k posuvu až na $2.$ pozici vzorku, neboť minulý znak a byl již kontrolován a uznán jako shoda, takže jeho opětovné porovnávání by ztrácelo význam.

Na obrázku 2.3 je dopracovaná ukázka vyhledávání KMP algoritmu z [1].



Obrázek 2.3: Ukázka KMP algoritmu

V případě KMP algoritmu je nutné k naleznutí vzorku v textu provést celkem 6 krát posuv vzorku, což je v porovnání s algoritmem hrubé síly o více než polovinu méně.

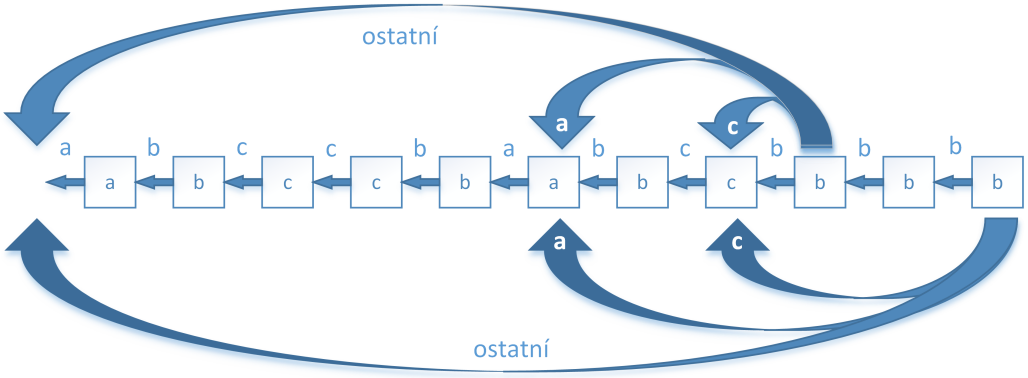
Časová složitost algoritmu je $O(m+n)$, kdy samotné prohledávání textu trvá $O(n)$ a předzpracování, tj. výpočet posuvu vzorku v případě neshody je $O(m)$.

2.1.3 Boyer-Moore algoritmus

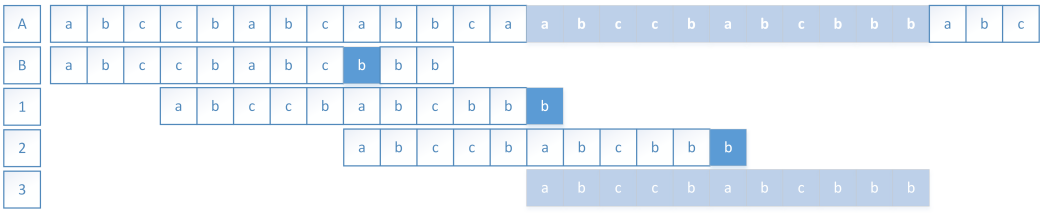
BM (Boyer-Moore) algoritmus se od hrubé síly liší v tom, že již neprohledává úseky textu, ve kterém se vzorek nemůže vyskytovat. Z toho vyplývá zmenšení celkového počtu porovnávání vzorku textu s prohledávaným textem a dojde k urychlení jeho nalezení. Algoritmus hrubá síla a Knuth-Morris-Prattův při hledání prohledává znaky textu vzorku zleva doprava. Tento algoritmus právě naopak, tedy zprava doleva. V případě nalezení odlišného znaku dochází k posunu vzorku doprava.

Velikost posunu vzorku je řízena funkcí *shift(j,x)*, kde j je pozice ve vzorku textu a x je znak abecedy a říká, o kolik znaků je možné při neshodě posunout vzorek, aby nedošlo k přeskočení možného výskytu vzorku v prohledávaném textu. Konkrétněji jde o číslo, udávající kolik pozic vlevo od porovnávané pozice vzorku *(j)* je ve vzorku nejbližší výskyt znaku x z prohledávaného textu. Pokud se znak x ve vzorku nalevo od pozice j nenachází, dochází k posunu vzorku o počet pozic rovnající se počtu znaků vzorku a porovnávání začíná znovu od poslední pozice znaku vzorku textu.

Obrázek 2.4 ukazuje posuv na pozicích neshody j z příkladu na obrázku 2.5. Z důvodu přehlednosti nejsou v obrázku znázorněny všechny možné posuvy pro každou pozici znaku. Jsou zobrazeny pouze pozice, související s tímto příkladem. Shoda znaku je znázorněna vodorovnou šipkou vlevo a posuvem o jednu pozici. Neshodu znaků zobrazují šipky s posuvem na nejbližší výskyt právě toho znaku, který zapříčinil neshodu. Nenalezení tohoto znaku ve vzorku způsobí posun celého vzorku.



Obrázek 2.4: Funkce shift BM algoritmu



Obrázek 2.5: Ukázka BM algoritmu

Vzorek B na obrázku 2.5 je umístěn na počátek prohledávaného textu. Následně se začínají porovnávat znaky od jeho konce. V případě shody se

porovnává další následující znak, resp. předchozí znak ve vzorku a prohledávaném textu. V případě neshody dochází k posunu vzorku vpravo o počet míst, určených dle funkce 2.4.

Použitím *Boyer-Mooreova* algoritmu je pro nalezení vzorku textu zapotřebí provést jen *3* posuvy. To je výrazně méně, než bylo potřeba u předchozích algoritmů. Jejich porovnání je uvedeno v následující podkapitole 2.2.

Nejhorší možný případ asymptotické složitosti algoritmu je *O(mn)*, průměrná složitost *O(n/m)* a platí, že při vyhledávání dlouhých vzorků je doba vyhledávání kratší, než u vyhledávání kratších vzorků. Je to dáno tím, že v případě neshody dochází k posunu o větší část vzorku a právě velikost posunu udává, o kolik méně je dále prováděno porovnání znaků.

2.2 Srovnání metod vyhledávání

Tabulka 2.1 porovnává výše zmíněné algoritmy vyhledávání na vzorovém příkladu z podkapitoly 2.1.

Algoritmus	**Max. složitost**	**Prům. složitost**	**Počet posuvů**	**Počet porovnání**
Hrubá síla	O(mn)	O(n)	13	38
KMP	O(m+n)	O(m+n)	6	31
BM	O(mn)	O(n/m)	3	16

Tabulka 2.1: Srovnání algoritmů vyhledávání

Z tabulky 2.1 je vidět značná úspora počtu porovnání *Boyer-Mooreova* algoritmu, především vůči triviálnímu algoritmu hrubé síly. To neplatí samozřejmě pro všechny případy vyhledávání. V případě vyhledávání krátkého vzorku textu, může dojít k výraznému zmenšení rozdílu těchto metod. V případě vyhledávání pouze jednoho znaku, metody si budou v počtu porovnávání rovny. Vždy tedy záleží na konkrétních případech, jaké velikosti vzorku textu se dají při vyhledávání očekávat a dle toho vhodně zvolit příslušnou metodu vyhledávání. Pokud se dá předpokládat vyhledávání dlouhých částí textu, je vhodnější volit *BM* algoritmus a naopak, pro krátké vzorky textu metodu hrubé síly, kdy odpadá vyšší složitost implementace. Je zde samozřejmě také potřeba zvážit náročnost implementace algoritmů vyhledávání, která je pro algoritmy *KMP* a *BM* náročnější.

3 Indexace

Z metod uvedených v kapitole 2 vyplývá, že pokud máme soubory malých velikostí, hledání vzorku textu nebude dělat problémy. Vezmeme-li ale v úvahu soubor o velikosti v řádech GB (GigaByte) a více, mohlo by velice snadno nastat, že hledání vzorku textu již nebude tak rychlé. Tyto základní algoritmy prohledávání textu jsou tedy silně závislé na délce prohledávaného textu (velikosti souboru). Abychom mohli takto veliké soubory prohledávat v rozumném čase, využívá se technika zvaná indexace, při které vzniká index dat.

Vyhledávání pak probíhá ve vytvořeném indexu. Data, která nebudou zaindexována, nemůžeme při vyhledávání nijak najít. A proto volba, která data budou při tvorbě indexu indexována, silně ovlivňuje výsledky vyhledávání.

3.1 Invertované soubory

Invertovaný soubor obsahuje uspořádaný[1] seznam termů[2], odkazy na umístění termů v dokumentu a jejich četnost výskytu (obrázek 3.1). Odkazy invertovaného souboru ukazují na počátek seznamu, obsahující souřadnice dokumentu odkazujícího termu. Souřadnice reprezentují např. kapitoly, odstavce nebo věty dokumentu, kde se daný term vyskytuje. Vyhledávání dat v invertovaných souborech pak probíhá metodou půlení intervalu, která má oproti sekvenčnímu vyhledávání ($0(n)$) složitost $O(log_2 n)$.

3.1.1 Invertované soubory – tvorba

Sekvenčním procházením dokumentu získáme jednotlivá slova, které před vložením do souboru indexu porovnáme se slovníkem nevýznamových slov (3.2). K jednotlivým slovům se uloží také jejich pozice (resp. odkaz na jejich umístění), viz obrázek 3.1 A).

Dalším krokem je uspořádání seznamu nalezených slov (3.1 B)).

[1] Seznam seřazený podle klíčových slov.

[2] Vzorek textu, at' už jednoslovný, či více slovný se nazývá term. Někdy ho můžeme označovat jako klíčové slovo.

Term	Ukazatel/č.dokum.
Počítač	10
Hardware	15
Inteligence	12
Software	11
Indexace	20
Vyhledávání	31
.	.
.	.
.	.
Indexace	14
Software	13
.	.
.	.
.	.

A)

Term	Ukazatel/č.dokum
Hardware	15
Indexace	20
Indexace	14
Inteligence	12
Počítač	10
Software	13
Software	11
Vyhledávání	31
.	.
.	.
.	.
.	.
.	.
.	.

B)

Term	Frekvence	Ukazatel/č.dokum.
Hardware	1	15
Indexace	2	14, 20
Inteligence	1	12
Počítač	1	10
Software	2	11,13
Vyhledávání	1	31
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.

C)

Obrázek 3.1: Struktura invertovaného souboru

Posledním krokem je odstranění duplicit termů (3.1 C)), neboť jeden term se může v dokumentu vyskytovat vícekrát (= frekvence termů). Výsledkem je uspořádaný seznam termů, kde jednotlivé termy odkazují na svoji pozici v dokumentu.

3.2 Slovník nevýznamových slov

Každý dokument, ať už je zaměřený na jakoukoli tématiku, obsahuje vždy značné množství slov, které se dokola opakují a mají tedy malou vypovídající hodnotu pro daný dokument. Tyto slova mají v dokumentech podíl 20% až 30% [1]. Tzn., že zpracováváme v každém dokumentu 20% až 30% obsahu navíc a přitom se tento obsah nijak výrazně napříč dokumenty nemění. Není to tedy zanedbatelné číslo. V případě, že tento *"nežádoucí"* obsah dokumentu odstraníme, dojde k výrazné úspoře prostředků při jeho zpracování. Slovům, tvořící tento nežádoucí obsah, se říká *nevýznamová slova* a jejich odstraněním dokument neztrácí na své vypovídající hodnotě. Jedná se především o spojky, předložky nebo zájmena. Obsah slovníku nevýznamových slov je pro každý jazyk samozřejmě odlišný, nelze tedy používat globální slovníky při zpracovávání dokumentů. Výhodou tohoto principu je možnost vytvořit si vlastní slovník pro daný dokument při jeho zpracovávání. Není nutné používat předem známé slovníky nevýznamových slov, ale při prohledávání dokumentu a nalezení frekventovaných slov říci, že se jedná o nevýznamové slovo a dále s ním nepracovat. Tato metoda je samozřejmě náročnější, než-li když máme předem slovník nevýznamových slov, ale daný slovník je vždy tvořen na konkrétní dokument a odpadá závislost už jen např. na použitém jazyku dokumentu.

Nevýznamová slova se odstraňují z dokumentů, či textů při jejich zpracování a dále již nedochází k jejich zaindexování, dochází k redukci velikosti indexu a s tím posléze související i rychlejší nalezení odpovědi pro dotaz nad indexem. V některé literatuře můžeme narazit na pojem *stopslovo* (*stopword*), které je alternativou k nevýznamovému slovu.

3.3 Lemmatizace

Při zpracovávání textu v přirozeném jazyce nastane snadno případ, kdy např. pro slovo *volat* nedostaneme při hledání žádný výsledek, ale přitom dokument může obsahovat slova podobné, např. *volající*, *volaný*, apod. Pro tyto případy se využívá tzv. *lemmatizátor*, který převádí slova na jejich základní tvary, kterým se říká lemmata. Může k tomu využít několik způsobů. Prvním z nich je využití slovníku, kde pro konkrétní slovo jsou uvedeny jeho další možné tvary. Je jasné, že tyto slovníky musejí být obsahově velmi rozsáhlé. Druhou možností je využití algoritmické lemmatizace, kdy se o získání lemmatu postará program. Lemmatizace je opět, jako v případě slovníku nevýznamových slov (3.2) silně závislá na konkrétním jazyce dokumentu. Vždy je potřeba mít ten správný slovník nebo správný algoritmus pro získání základního tvaru slova, neboť tvorba slov a jejich dalších tvarů není pro všechny jazyky totožná a pro každý jazyk se řídí jinými pravidly.

Využít lemmatizaci lze již v průběhu vytváření indexu nebo až při vyhodnocování dotazu. V prvním případě se k jednotlivým termům indexu naleznou příslušná lemmata a ty se zaindexují. Dochází k redukci velikosti indexu, ale na druhou stranu se z dokumentu vytrácí konkrétní kontext, význam textu. Pokud využijeme lemmatizaci při vyhodnocování dotazu, budeme ke slovu z dotazu připojovat jeho další tvary a dojde tak k nárůstu velikosti dotazu.

3.4 Stemming

Na rozdíl od lemmatizace nehledáme slova základních tvarů, ale kořeny slov. Algoritmu, hledající kořeny slov se říká *stemmer*. Základní princip hledání kořene je v odstranění koncovky a předpony slova. Jelikož tvorba slov je opět závislá na konkrétním jazyku, tak i algoritmus stemmeru není pro každý jazyk totožný.

3.5 Taxonomie

Taxonomie je kolekce termů, mající mezi sebou vazbu. Kolekci si lze představit jako strom, kde nejvyšší úroveň je dělena do odvětví, ve kterém se cyklicky mohou jednotlivé položky dále členit do dalších pododvětví. Každý

term nižší něž nejvyšší úrovně má pak rodiče a potomka. Rodič jednotlivých termů pak symbolizuje kategorii, kam term patří (zobecňuje jej) a potomek daný term více konkretizuje. Více viz následující příklad, kde je ukázána stromová struktura taxonomie:

Elektronika

- *Počítače*
  - *Notebooky*
  - *Stolní počítače*
  - *Servery*

3.6 Ontologie

Ontologie je strukturou shodná s taxonomií, jen se liší v reprezentaci vztahu jednotlivých termů. Vztah už není reprezentován pouze jako rodič – potomek, ale může být libovolný. Ontologie tedy obsahuje kolekci termů a definici vztahů, popisující vazbu mezi jednotlivými termy.

3.7 Tezaurus

Tezaurus je slovník, obsahující seznamy ekvivalentních slov (synonym). Kvalita hledání synonym je tedy silně závislá na kvalitě (obsahu) použitého slovníku. Zde kvalitní slovník znamená, že obsahuje rozsáhlou množinu slovních pojmů a jejich synonym. S tím souvisí i velikost takového slovníku, která nebude v případě rozsáhlého slovníku zanedbatelná.

Protože pracujeme s přirozeným jazykem, může nastat, že použitím lemmatizace dvou absolutně významově rozdílných slov dostaneme stejný základ slova nebo podobně u stemmingu dostaneme stejný kořen slov. Je tedy nutné v těchto případech znát i daný kontext, abychom mohli určit správný význam slov.

Techniky, popsané v kapitolách výše, jsou využívány knihovnami *Apache Lucene*[5] a *Stanford CoreNLP*[6], použitých v této práci.

4 Tvorba dotazu

Vyhledávat nemusíme pouze jednoduché dotazy typu: hledej slovo *"počítač"*, ale lze vyhledává různá slovní spojení. Aby to bylo možné, využívá se následující techniky.

Boolovský model

Pro sestavení složitějšího dotazu, neobsahující pouze jeden term, lze využít Booleovské operátory (*AND*, *OR*, *XOR*, *NOT*)

Výsledkem může být např. dotaz:

```
systém souborů AND Windows
```

kde se požaduje hledání výskytu slov *systém souborů* tam, kde se vyskytuje slovo *Windows*. V případě neznalosti priority operátorů lze využít závorek *(,)*.

Krom výše uvedených operátorů mohou mít Booleovské modely další operátory a to tzv. *proximitní operátory* [1], pomocí nichž lze stanovit vzdálenosti výskytu termů v dokumentech. V tabulce 4.1 jsou uvedeny příklady proximitních operátorů.

Výraz	**Dokumenty, které obsahují ...**
A adjacent B	term A následovaný termem B.
A (n) words B	term A následovaný termem B v maximální vzdálenosti n slov.
A sentence B	termy A a B ve stejné větě.
A paragraph B	termy A a B ve stejném odstavci.

Tabulka 4.1: Proximitní operátory

V Booleovském modelu lze při tvorbě dotazu použít i primitivní regulární výrazy. V tabulce 4.2 [1] jsou popsány pro ukázku některé regulární výrazy a jejich významy.

Znak	**Význam**	**Příklad**	**Výstup**
.	Libovolný znak	.	a, x, z
$*$	Libovolný počet výskytu znaku, včetně nulového	ab*	a, ab, abb, ...
$+$	Libovolný počet výskytu znaku, kromě nulového	ab+	ab, abb, ...
[ ]	Libovolný jeden znak ze závorek	[abc]	a, b, c
[-]	Rozsah znaků v závorce	[a - d]	a, b, c, d

Tabulka 4.2: Booleovský model – regulární výrazy

5 Významnost dokumentů

Výše popsané postupy jsou snadno představitelné, hledáme-li výraz v konkrétním článku nebo dokumentu. Položíme dotaz a dostaneme výsledek a pozici, kde jej můžeme najít.

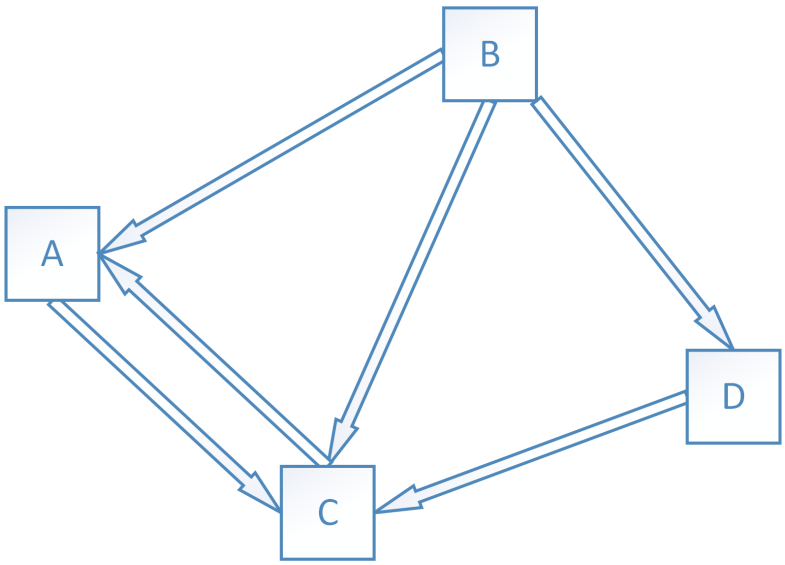
Nastane-li ale případ, kdy máme těchto dokumentů mnohokrát více, hledáme tedy napříč velkou množinou dokumentů, můžeme na náš dotaz dostat veliké množství výsledků, ve kterých se dá jen velmi těžko orientovat. Uvážíme-li navíc, že dokumenty jsou spolu často provázány různými citacemi, nastane situace, kdy se množina výsledků může zvětšit právě o tyto odkazované dokumenty, neboť mohou také obsahovat námi hledaný pojem. Abychom ale určili významnost dokumentů, potřebujeme nějaký systém, který nám řekne, jaký výsledek je ten správný, resp. co nejpřesněji odpovídá našemu dotazu. Existují techniky, zabývající se touto problematikou, které hodnotí dokumenty v závislosti na jejich popularitě nebo na jejich prestižnosti a jsou popsány v následujících kapitolách.

5.1 Indegree

Indegree je velmi jednoduchou reprezentací hodnocení popularity dokumentů. Spočívá v množství odkazů, vedoucí na daný dokument. Pořadí popularity dokumentů pak nepřímo úměrně odpovídá počtu vstupních odkazů daného dokumentu.

Na obrázku 5.1 je uveden příklad čtyř dokumentů *(A, B, C, D)* a jejich vzájemné provázanosti.

Dokument C je odkazován největším počtem ostatních dokumentů a je tedy tímto algoritmem vyhodnocen jako nejpopulárnější.



Obrázek 5.1: Indegree

5.2 PageRank

Jedná se o algoritmus hodnocení důležitosti webových stránek, ale dá se převést i na problém hodnocení prestižnosti dokumentů. Velikost hodnocení má pak vliv na pozici dokumentu ve výsledcích vyhledávání. Hodnocení daného dokumentu nespočívá pouze na závislosti vstupních odkazů, jak tomu bylo u algoritmu *indegree*, ale na samotném hodnocení dokumentů, které daný dokument odkazují. Pokud máme tedy významný dokument, odkazující na další dokumenty, pak tyto dokumenty získávají na důležitosti právě díky vysokému hodnocení odkazujícího dokumentu.

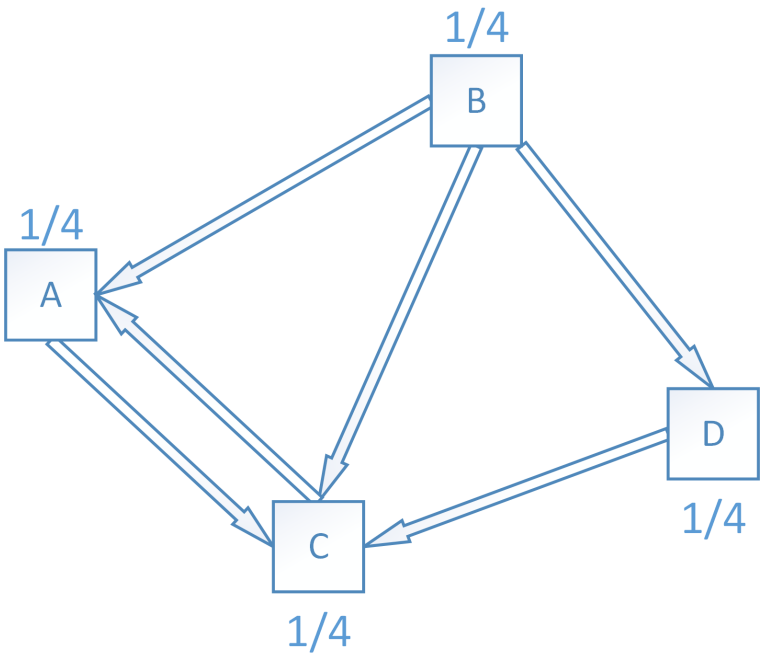
Ukážeme si srovnání pořadí dokumentů po aplikaci *PageRanku* na stejných datech (obrázek 5.1), jako v případě *indegree*.

Základem výpočtu hodnoty *PageRanku* pro dokument je následující iterační vzorec č. 1, [4].

$$PR_{t(i+1)} = \frac{(1-d)}{m} + d \cdot \sum_{1}^{m} \frac{PR_{t(i)}}{O_{t(i)}} \qquad (1)$$

- *PR t(i+1)* – hodnota PageRanku dokumentu iterace *i+1*
- *PR t(i)* – hodnota PageRanku dokumentu iterace *i*
- *O t(i)* – počet výstupních odkazů na další dokumenty
- *d* – damping faktor, nabývá hodnot *<0,1>* a značí pravděpodobnost pokračování na dalším odkazu aktuálního dokumentu
- *m* – počet dokumentů

V prvním fázi výpočtu je všem dokumentům přidělena stejná *PageRank* hodnota. Vezmeme-li, že součet všech hodnot dokumentů se rovná jedné, potom platí, že každý dokument má výchozí hodnotu *PageRanku* rovnu *1/m*, [2], tedy v našem konkrétním případě *1/4*, viz obrázek 5.2.



Obrázek 5.2: PageRank – počáteční hodnoty

Druhým krokem je vytvoření matice *P* s odkazy mezi dokumenty, kde *O(X, Y)* je odkaz ze stránky *X* na stránku *Y* s hodnotou *1/O t(X)*. Matice má obecný tvar viz tabulka 5.1.

O(A,A)	O(A,B)	...	O(A,m)
O(B,A)	O(B,B)	...	O(B,m)
...	...	...	...
O(m,A)	O(m,B)	...	O(m,m)

Tabulka 5.1: P matice odkazů – obecný tvar

Matice P pro náš konkrétní případ viz tabulka 5.2.

0	0	1	0
1/3	0	1/3	1/3
1/2	1/2	0	0
0	0	1	0

Tabulka 5.2: P matice odkazů – konkrétní příklad

V případě existence dokumentu, neodkazujícího na žádné jiné dokumenty, dochází v matici v příslušné řádce dokumentu k nahrazení nulových hodnot hodnotami rovnající se *1/m*. V tomto konkrétním případě se zde žádný takový dokument nenachází, ale kdyby ano, hodnoty dokumentu v matici by nabývaly čísel *1/4*.

Z matice P se dále dle vzorce č. 2 a pro damping faktor $d = 0{,}85$ [4] vypočte matice M, uvedená v tabulce 5.3.

$$M = \frac{(1-d)}{m} + d \cdot P \qquad (2)$$

0,0375	0,0375	0,8875	0,0375
0,32083	0,0375	0,32083	0,32083
0,8875	0,0375	0,0375	0,0375
0,0375	0,0375	0,8875	0,0375

Tabulka 5.3: M matice

Pro jednotlivé iterace výpočtu pak platí vzorec č. 3.

$$PR_{t(i+1)} = PR_{t(i)} \cdot M \quad (3)$$

Iterační výpočet končí tehdy, konvergují-li hodnoty *PageRanku* jednotlivých dokumentů. V tabulce 5.4 je uvedeno několik prvních iterací výpočtu, ze které vyplývají dva významné dokumenty, konkrétně dokumenty A a C.

Iterace	**A**	**B**	**C**	**D**
0	1/4	1/4	1/4	1/4
1	0,3208	0,0375	0,5333	0,1088
2	0,5015	0,0375	0,4129	0,0481
3	0,399095	0,0375	0,5153	0,0481
4	0,486103	0,0375	0,4282	0,0481
..	..	..	..	..
10	0,461201	0,0375	0,453157	0,0481

Tabulka 5.4: Tabulka iterací výpočtu PageRanku

Pořadí dokumentů po 10.iteraci je v následující tabulce 5.5.

Pořadí	**Hodnota PageRank**	**Dokument**
1	0,4612	A
2	0,4531	C
3	0,0481	D
4	0,0375	B

Tabulka 5.5: Hodnocení dokumentů – PageRank

Srovnání vypočteného pořadí dokumentů z metod 5.1 a 5.2 ukazuje tabulka 5.6.

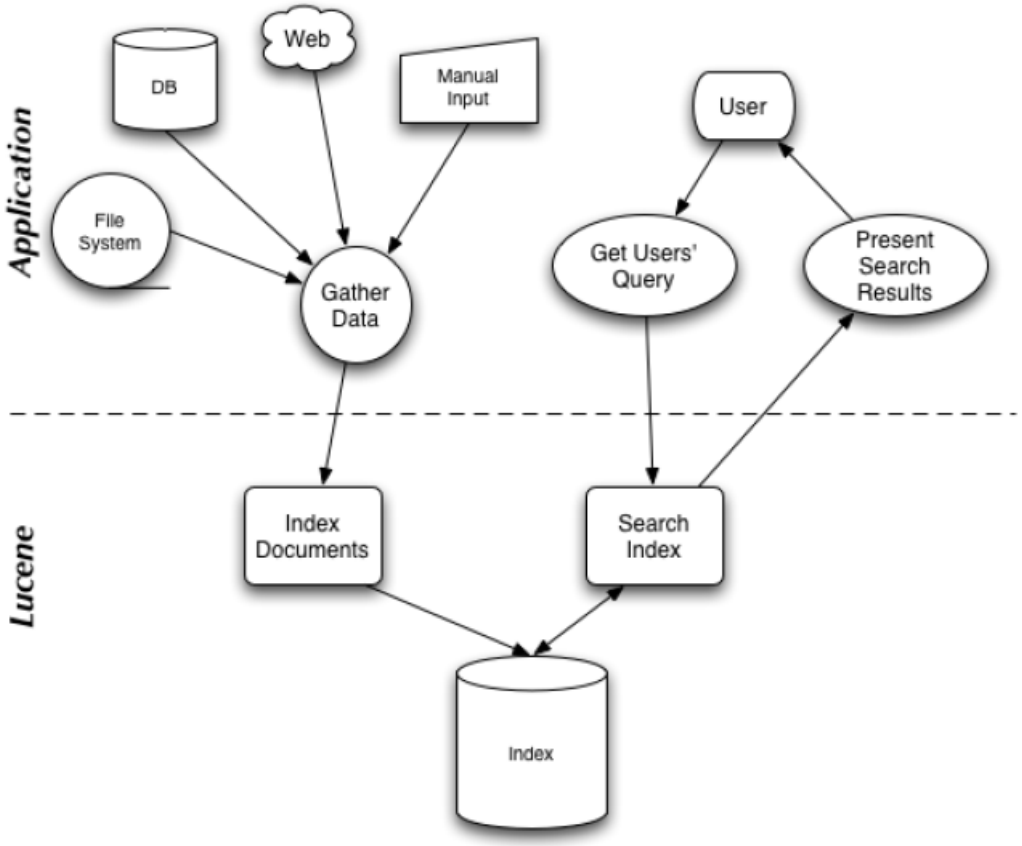
Dokument/Pořadí	**PageRank**	**Indegree**
A	1	2
B	4	4
C	2	1
D	3	3

Tabulka 5.6: Srovnání pořadí metod Indegree a PageRank

Obě metody nám pro tento příklad daly trochu odlišné výsledky, kdy *Indegree* nejvíce upřednostnilo dokument *C*, zatím co *PageRank* označil jako nejlepší dokument *A*. *Indegree* nám poskytuje pouze pořadí dokumentů, ale využitím *PageRanku* dostaneme ohodnocení každého dokumentu, ze kterého lze vzájemným porovnáním určit stupeň jejich významnosti. Např. srovnáním dokumentů *C* a *D* dostáváme výsledek, že dokument *C* je pro *PageRank* přibližně *9,5*-krát zajímavějším.

6 Lucene

Lucene je open source[1] knihovna, psaná v jazyce JAVA, umožňující vysoce výkonné vyhledávání a indexaci dat. Na obrázku 6.1 [3] je znázorněn vztah mezi Lucene a aplikací.



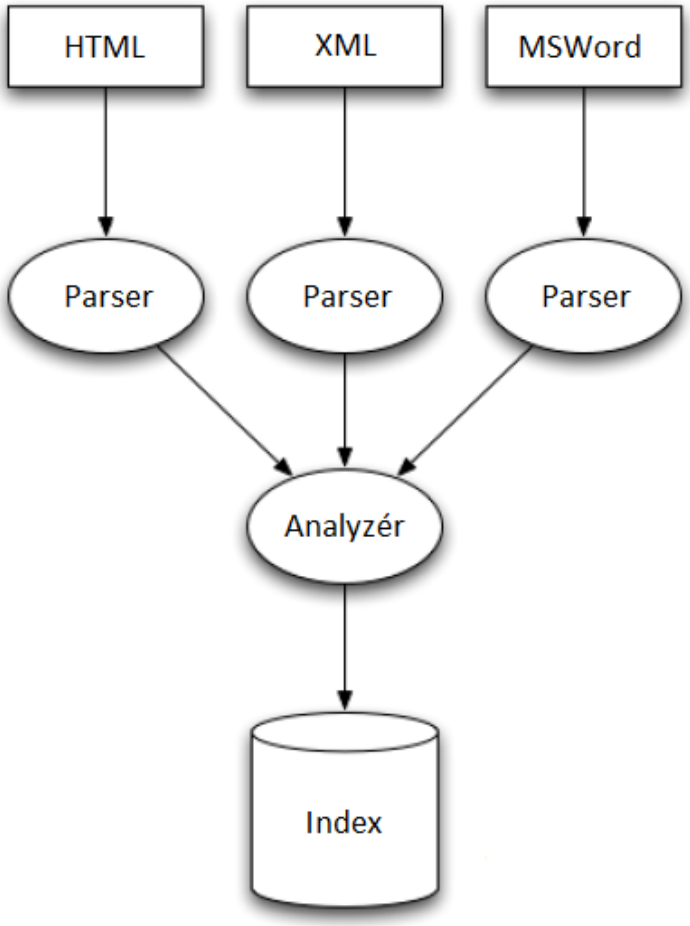
Obrázek 6.1: Vztah Lucene k aplikaci

Aplikace poskytne Lucene data, která jsou knihovnou naindexována. Uživatel položí dotaz, který je předán Lucene a následně dochází k jeho vyhledání v indexu dat. Aplikace pak už jen zobrazí nalezené výsledky poskytnuté knihovnou.

[1]Software s volně přístupným zdrojovým kódem.

6.1 Indexace

Průběh indexace dat je znázorněn na obrázku 6.2 [3]. Lucene umožňuje indexaci dat různých formátů, jako např. *XML*[2], *HTML*[3], *MSWord*[4] a dalších. Parser získává ze vstupních dat text (tagy, slova), která jsou dále zpracovávána analyzérem. Analyzér může slova porovnávat se slovníkem nevýznamových slov (3.2), převádět na malá či velká písmena a další [5]. Takto zpracovaná data knihovna uloží do indexu ve formě invertovaného souboru (podkapitola 3.1).



Obrázek 6.2: Lucene – indexace

[2]Extensible Markup Language
[3]HyperText Markup Language
[4]Microsoft Word

Vytvořený index lze ukládat dvěma způsoby:

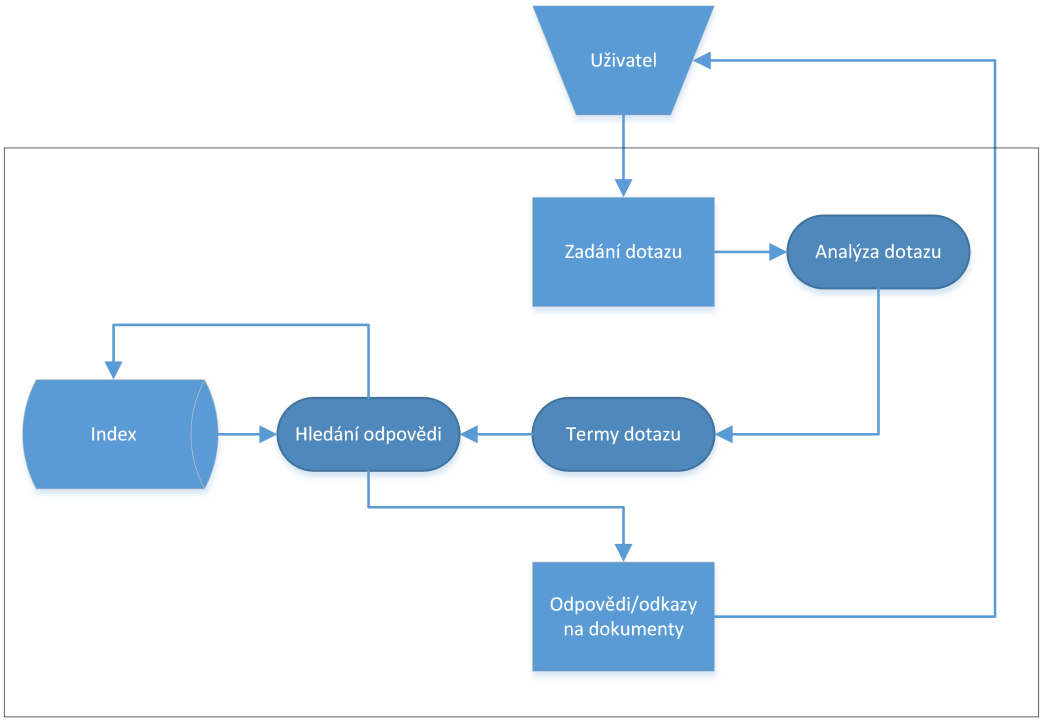
- na pevný disk
- do operační paměti

Každá možnost s sebou nese výhody i nevýhody. V případě ukládání indexu do paměti je nalezení požadované položky rychlejší, ale v případě velkého množství dat, může nastat nedostatek operační paměti. Navíc se index při každém startu aplikace musí do paměti opětovně nahrát. Uložením indexu na pevný disk nejsme omezeni velikostí operační paměti a odpadá nutnost předčítání indexu před startem aplikace. Daní za to může být pomalejší nalezení položky v indexu.

Lucene umožňuje zkombinovat oba způsoby ukládání indexu, kdy jako primární paměť je využívána operační paměť a v případě její nedostatečné velikosti je index přesunut na pevný disk. Operační paměť se pak dá označit jako tzv. vyrovnávací paměť, která urychluje práci s daty a omezuje častý přístup k pevnému disku.

6.2 Vyhledávání

Dotaz, pro který chceme najít odpovědi, *Lucene* nejprve analyzuje. Při analýze dotazu se vytváří termy dotazu a obdobně jako při indexaci se využívá nevýznamových slov, převodu na malá/velká písmena a pokud je třeba, využije se Booleovských operátorů (kapitola 4). Upravený dotaz se následně hledá v indexu dat metodou půlení intervalu.



Obrázek 6.3: Lucene – vyhledávání

Odpovědí je seznam dokumentů, odpovídající položenému dotazu, které jsou poskytnuty uživateli (obrázek 6.3). Knihovna umožňuje hledání odpovědi v indexu omezit různými filtry. Pokud máme např. data obsahující časové údaje, můžeme nastavit filtr vyhledávání tak, aby vyhledával položky jen ze zvoleného omezeného časového období. Uživatel tak dostane pouze výsledky z tohoto období.

6.3 Nevýznamová slova

Knihovna umožňuje při indexaci, tak i při vyhledávání identifikaci nevýznamových slov (podkapitola 3.2), se kterými dále nepracuje. To obstarává již zmíněný analyzér. V případě nutnosti provádět další operace se slovy, které knihovna *Lucene* nepodporuje, je možnost implementace vlastního analyzéru s danou funkčností.

Seznam nevýznamových slov, které knihovna využívá je uveden v následující tabulce 6.1.

a	*an*	*and*	*are*
as	*at*	*be*	*but*
by	*for*	*if*	*in*
into	*is*	*it*	*no*
not	*of*	*on*	*or*
such	*that*	*the*	*their*
then	*there*	*these*	*they*
this	*to*	*was*	*will*
with			

Tabulka 6.1: Lucene – nevýznamová slova

7 Praktická část

Aplikace je navržena jako webová aplikace, psaná v programovacím jazyce *Java EE* (Enterprise Edition) s využitím vývojového prostředí *Eclipse Java EE* využívající framework *PrimeFaces*[10] starající se o *GUI* (Graphics User Interface) aplikace. Jelikož aplikace využívá několik knihoven, bylo pro projekt využit nástroj *Apache Maven*[11], který se stará o snadnější konfiguraci a sestavení celého projektu.

Celá práce stojí na několika hlavních bodech:

- indexace dat
- položení a tvorba dotazu
- vyhledání výsledků
- hodnocení nalezených výsledků

Na obrázku 7.1 je uveden vývojový diagram aplikace, kde je možné vidět jednotlivé části programu a postup, jak samotné vyhledávání probíhá. Jednotlivé kroky algoritmu jsou popsány dále v této kapitole.

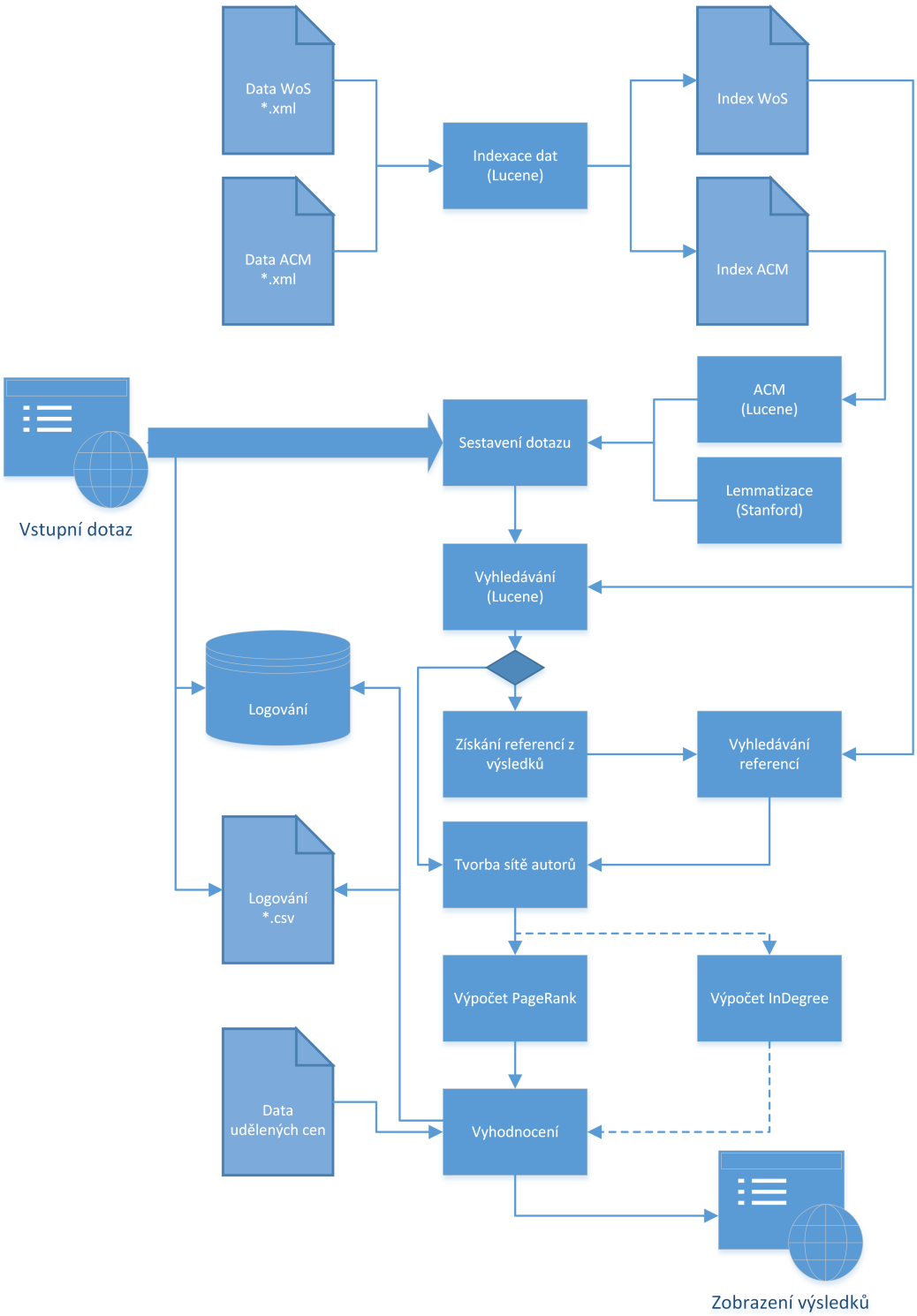
Pro indexaci dat a vyhledávání v nich byla použita knihovna *Apache Lucene*[5], neboť z hlediska výkonnosti nebylo možné provádět hledání triviálními způsoby, jako je např. metoda *brute force*, zmíněná v 2.1.1.

7.1 Vstupní data

Web of Science

Jedná se o citační databázi, obsahující informace o vědeckých článcích, jako např. jejich abstrakty, citace nebo i jejich texty. Databáze obsahuje data vědeckých pracích od roku 1945. Je přístupná přes online portál[1] nebo ji lze využívat offline, ve formě *xml* souboru, jako je tomu v případě této práce.

[1] http://isiknowledge.com/



Obrázek 7.1: Vývojový diagram aplikace

Vstupem této práce je použita část dat databáze *WoS*, konkrétněji část z období od roku 1995 do 2007 . Data jsou uložena v *xml* souboru, se strukturou viz příloha A.1. Data obsahují celkem 149348 záznamů.

ACM ontologie

Pro využití ontologie jsou použita data ACM (Association for Computing Machinery). Data jsou uložena v *xml* a mají strukturu dat viz příloha A.2. Obsahují celkem 4600 záznamů. Pro indexaci a vyhledávání se zde používá také knihovna *Apache Lucene*.

ACM ontologie se zabývá konkrétně oblastí výpočetní techniky. Je dostupná online [8] nebo ve formě XML souboru [9] a obsahuje následující oblasti:

- *General and reference*
- *Hardware*
- *Computer systems organization*
- *Networks*
- *Software and its engineering*
- *Theory of computation*
- *Mathematics of computing*
- *Information systems*
- *Security and privacy*
- *Human-centered computing*
- *Computing methodologies*
- *Applied computing*
- *Social and professional topics*
- *Proper nouns: People, technologies and companies*

Výše vypsané oblasti se dále dělí na další podoblasti. Jako příklad vezmeme *Hardware*, který má podoblasti *Hardware test* a *Integrated circuits*.

Podoblast *Hardware* se dále dělí na:

- *Fault models and test metrics*
- *Memory test and repair*
- *Hardware reliability screening*

S využitím ontologie můžeme při hledání pojmu *Hardware test* různě upravovat náš dotaz. Můžeme ho zobecnit, kdy k vyhledávanému pojmu připojíme oblast, do které náš pojem patří, v tomto případě *Hardware* a jeho podoblast *Integrated circuits*, nebo můžeme dotaz více konkretizovat, kdy vyhledáváme v podoblastech našeho pojmu, tedy v *Fault models and test metrics*, *Memory test and repair* a *Hardware reliability screening*.

Udělené ceny

Pro hodnocení významnosti autorů jsou použita data, obsahující seznam cen a autorů, kteří obdrželi významné ceny v určitých oblastech. Data jsou uložena v samostatných *csv* souborech jednotlivých ocenění.

- *ACM Fellows*[2] – udělující se za inovace ve výpočetní technice
- *ACM Turing Award*[3] – označována jako Nobelova cena výpočetní techniky
- *ACM SIGMOD Edgar F. Codd Innovation Award*[4] – udělující se za přínosy v oblasti správy dat a databázových systémů
- *ISI* [5] *Highly Cited*[6] – autoři s nejčastěji citovanými publikacemi

[2]http://fellows.acm.org/
[3]http://amturing.acm.org/
[4]http://www.sigmod.org/sigmod-awards/
[5]The Institute for Scientific Information
[6]http://www.isihighlycited.com/

Data udělených cen v této práci jsou ze stejného časového období jako *WoS* data. Vzhledem k principu udělování cen, kdy ceny za články se udělují zpětně, nikoliv v období jejich publikace, by bylo vhodné využít data o udělených cenách z časově posunutého období (oproti vstupním datům). Při tvorbě této práce ale byla dostupná pouze data ze shodného časového období, nicméně aplikace umožňuje tyto data kdykoliv v budoucnu aktualizovat (viz podkapitola 8.4).

Lemmatizace

Pro lemmatizaci slov je použita knihovna *Stanford CoreNLP*[6]. Knihovna neumí pracovat s českým jazykem, ale vzhledem k *WoS* datům, které jsou dostupné v anglickém jazyce, je tento problém bezpředmětný.

7.2 Indexace

Ve vstupních datech byly vybrány následující položky, pro které je vytvářen index. (Struktura dat viz příloha A.1).

- *recid* – číslo publikace
- *source_title* – název publikace
- *item_title* – název článku
- *primary_author* – primární autor
- *author* – sekundární autoři
- *keyword* – klíčová slova
- *ref* – reference na další publikace
- *bib_id* – rozsah stránek v publikaci, rok vydání publikace
- *primarylang* – jazyk publikace
- *doctype* – typ publikace
- *abstract* – abstrakt publikace
- *email_name* – jméno autora následujícího emailu
- *email_addr* – email autora
- *rp_author* – jméno autora následující adresy
- *rp_address* – adresa autora
- *rp_organization* – organizace autora
- *rp_state* – stát autora
- *rp_street* – ulice adresy autora
- *rp_city* – město adresy autora
- *rp_country* – země autora
- *rp_zip* – poštovní směrovací číslo adresy

O proces indexace se stará knihovna *Apache Lucene*, která k indexaci *xml* souboru využívá knihovny *Digester*[7], což je vlastně parser *xml*, kterému řekneme, jaké *xml* tagy potřebujeme zpracovat, on je zpracuje a poskytne dále knihovně *Lucene*, která z nich vytvoří index a uloží jej. V případě této práce bylo zvoleno ukládání indexu na pevný disk. Odpadají tak možné komplikace při využití operační paměti (podkapitola 6.1) a navíc, výhoda rychlejšího nalezení položky v indexu je vzhledem k dalším operacím (tvorba sítě a její hodnocení), trvající o poznání déle, minoritní.

Obdobně jako index pro vstupní data se vytváří index pro data *ACM*. Zde se ale navíc provádí předzpracování dat souboru, nebot' tak jak jsou data dostupná, obsahují nepovolené znaky a *xml* tagy, které *Lucene* neumí zpracovat, více viz tabulka 7.1.

Nevyžádaný znak	**Nahrazeno**
&	AND
rdf:	prázdným řetězcem
skos:	prázdným řetězcem

Tabulka 7.1: Nahrazení znaků v datech ACM

Pro data *ACM* (struktura viz příloha A.2) jsou indexované následující položky:

- *concept* – číslo oblasti
- *prefLabel* – název oblasti
- *altLabel* – alternativní název oblasti
- *broader* – číslo nadoblasti
- *narrower* – číslo podoblasti

V prvním kroku tedy dojde k vytvoření dvou indexů, obsahující naindexovaná data z databáze *WoS* a data *ACM* ontologie. Velikost dat *WoS* je 539MB a *ACM* dat 1,24MB. Vytvořením indexu se data zredukovala na velikost 115MB (o *21%*) pro data *WoS* a pro data *ACM* pouze na 290kB (o *23%*).

7.3 Tvorba dotazu

Jak bylo popsáno v kapitole 4 i zde se využívá při tvorbě dotazu Booleovského modelu, kdy se dotaz sestavuje z několika částí. K dotazu se připojují nalezené lemmata a *ACM* oblasti a vzniká tak nový, rozšířený dotaz.

Ukázka tvorby dotazu:

Vyhledáváme:

```
operating system
```

ACM poskytne:

```
os
monitor in operating system
MS-DOS
realtime systems
windows operating system
```

Lemmatizátor poskytne:

```
operating, operate
monitor, monitors
system, systems
windows, window
```

Výsledný dotaz pouze s využitím ACM vypadá následovně:

```
(operating system) OR (os) OR (monitor in operating system) OR
(MS-DOS) OR (realtime systems) OR (windows operating system)
```

Přidá-li se navíc lemmatizace, dotaz bude upraven následovně:

```
( ((operating OR operate) AND (system OR systems) ) OR
                        (os) OR
((monitor OR monitors) AND in AND (operating OR operate)
              AND (system OR systems) ) OR
                        (MS-DOS) OR
          (realtime AND (systems OR system) ) OR
((windows OR window) AND (operating OR operate)
              AND (system OR systems)) )
```

Dotaz s využitím lemmatizace a ontologie značně nabývá na své velikosti. Není ale pravidlem, že využitím těchto technik dosáhneme nalezení přesnějších výsledků. Můžeme jimi do dotazu zavést klamné pojmy, nesouvisející s kontextem a tím negativně ovlivnit výsledek vyhledávání. Příkladem může být lemma pro slovo *windows*, které lemmatizátor určil jako *window*. Lemma je určené sice správně, ale z hlediska významu slova v konkrétním kontextu úplně chybně.

7.4 Vyhledávání

Ze zvolených položek při indexaci dat (podkapitola 7.2) je zřejmé, ve který lze vyhledávat položené dotazy. Uživatel tedy může vyhledávat např. podle titulu článku, autora, jeho emailu nebo může položit dotaz na určitou lokalitu a v odpovědi dostane všechny publikace pocházející z dané lokality. K vyhledávání se využívá knihovnou *Lucene* předem připravený index. Vyhledávání v indexu má na starosti stejná knihovna.

Vyhledávají se články v závislosti na položeném dotazu, který prochází různým předzpracováním (viz podkapitola 7.3). Výsledky vyhledávání však nejsou brány jako finální a provádí se další rozšiřující vyhledávání. Toto vyhledávání zohledňuje vzájemné reference článků a autorů, protože na položený dotaz může lépe vyhovovat jiný článek, který sice neobsahuje daná klíčová slova která hledáme, ale věnuje se stejné tématice a byl by tak v případě nevyužití vzájemných referencí nenalezen.

V prvním kole tedy dojde k nalezení článků, obsahující klíčová slova. Z těchto výsledků se získají reference na další články, které se následně vyhledají a přidají do množiny výsledků. Další hledání referencí ve výsledcích neprobíhá. Vyhledává se tedy jen jedna úroveň referencí. Vyhledáváním dalších úrovní referencí by mohlo do množiny výsledků zanést již nerelevantní výsledky.

Výsledek vyhledávání se skládá ze seznamu objektů, resp. článků, obsahující všechny potřebné informace, které byly obsaženy ve vstupním souboru *WoS*.

7.4.1 Tvorba sítě

Z nalezených článků program vybere všechny autory, které budou reprezentovat uzly vytvářené sítě. Ke každému uzlu se připojí články z výsledků vyhledávání, které daný autor publikoval. Po vytvoření všech uzlů sítě následuje jejich vzájemné propojení, vytvoření hran sítě. Hrana mezi uzly je vytvořena tehdy, pokud jeden autor odkazuje na druhého v některé ze svých publikací.

Vznikne tak síť autorů, kde každý autor (uzel) obsahuje své publikace a reference na další autory.

7.4.2 Hodnocení výsledků

Z takto vytvořené sítě se dále provádí tvorba matice sousednosti, která je základem pro výpočet hodnot *PageRanku*. Matice sousednosti obsahuje hodnotu *1* tam, kde jeden uzel odkazuje na další. V případě, že uzel má více odkazů na další uzly, dojde k úpravě matice tak, že původní hodnota *1* je dělena celkovým počtem odkazujících odkazů daného uzlu. Dalším krokem je zohlednění případné existence uzlu, který nemá žádné odkazy na další uzly, tedy v matici obsahuje samé *0*. Nulové hodnoty se nahradí číslem, rovnající se podílu *1* a celkovému počtu uzlů. Z takto připravené matice se dle vzorce č. 2 z podkapitoly 5.2 provede finální úprava matice, viz následující ukázka části programového kódu:

```
private final double ALPHA = 0.85;
private final double ONE = 1.0;

...

private double[][] createFinalyMatrix(double[][] matrix){

 double factor = ( ONE - ALPHA ) *
    ( ONE / ((double)matrix.length) );

 for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix.length; j++) {

   matrix[i][j] = ( ALPHA * matrix[i][j] ) + ( factor );

  }
 }

 return matrix;
}
```

Na závěr se provádí ověření správnosti hodnot matice, kdy součet hodnot jednotlivých řádek musí být roven *1*.

Takto vytvořená matice je vstupem pro výpočet hodnot *PageRanku*. Jedná se o iterační výpočet, kdy je v počáteční iteraci zvolena stejná výchozí hodnota *PageRanku* pro každého autora. Jedná se o vektor hodnot, mající defaultní hodnoty *1 / počet autorů*, více viz podkapitola 5.2. S počátečním vektorem se dle vzorce č. 3 (podkapitola 5.2) zahájí iterační výpočet , viz následující programový kód, kde je ukázka výpočtu jedné iterace *PageRanku*:

```
private final double ZERO_INIT = 0.0;

...

private double[] computeOneIteration(double[] lastVector){

 double[] nextVector = new double[lastVector.length];
 double tmpVal = ZERO_INIT;

 for (int i = 0; i < inputMatrix.length; i++) {
  for (int j = 0; j < inputMatrix.length; j++) {

   tmpVal += ( lastVector[j] * inputMatrix[i][j] );

  }

  nextVector[i] = tmpVal;
  tmpVal = ZERO_INIT;
 }

 return nextVector;
}
```

Vzhledem k možné náročnosti výpočtu (při velkém počtu výsledků) se ukončovací podmínka iteračního algoritmu kontroluje pouze po každé *10.* iteraci. Pro splnění ukončovací podmínky algoritmu musí být rozdíl Euklidovských norem dvou po sobě jdoucích vektorů menší než *Epsilon*, které bylo stanoveno na *0,0000001*.

Ukončením iteračního výpočtu dostaneme vektor hodnot, odpovídající hodnotám *PageRanku* pro jednotlivé uzly (autory). Hodnoty z vektoru jsou přiřazeny autorům a následně řazeny dle jejich velikosti.

Algoritmus je schopen určit také hodnoty popularity (*indegree*) jednotlivých autorů. Využívá k tomu vytvořenou síť autorů pro algoritmus *PageRank*, ve které počítá pro jednotlivé autory počty vstupních odkazů. Dle těchto hodnot pak probíhá řazení jednotlivých autorů. Hodnoty popularity ve finálním programu nebyly využity, slouží pouze pro srovnání s algoritmem *PageRank*, viz podkapitola 7.5.

7.4.3 Ocenění autorů

Pro každého autora, který se dostal do finálního seznamu výsledků jsou zjišťovány další informace, ohledně jeho získaných cen z různých oblastí. Vzhledem k dostupným datům se pro autory zjišťuje, zda neobdrželi některou z následujících cen:

- ACM Fellows
- ACM Turing Award
- ACM SIGMOD Edgar F. Codd Innovation Award
- ISI Higly Cited

V případě, že autor některou z výše uvedených cen obdržel, dojde k zobrazení informace u výpisu autora, viz obrázek 8.12. Dostupná data jsou ze stejného období, jako vstupní data *WoS* a jsou uloženy v samostatných *CSV* souborech (podkapitola 7.1), které zpracovává jednoduchý parser.

Udělené ceny autorů dále slouží k ověření funkčnosti aplikace, viz podkapitola 7.6.

7.4.4 Příklad vyhledávání

V tabulce 7.2 je uvedeno několik příkladů dotazů s počtem nalezených výsledků pro různá nastavení vyhledávání.

Dotaz/Počet publikací	**Standardní nastavení**	**Využití lemmatizace**	**Využití ontologie**
operating system	276	278	7496
computer vision	3715	3715	16898
database application	137	137	180
database systems	1543	1949	12446
mobile devices	155	199	579
websites	102	198	104
database queries	221	674	221
system Windows	32	63	713
document structure	99	99	104
system security	366	366	1635

Tabulka 7.2: Příklady vyhledávání

Počet výsledků v případě využití lemmatizace nijak dramaticky nenarůstá. Lemmatizace vytváří pro slova jejich lemmata, která jsou v tomto případě většinou jednotná/množná čísla slov, která jsou přidána do vstupního dotazu. Vzhledem k počtu (jednotky) lemmat tak vstupní dotaz nijak výrazně nenarůstá. Opakem je ale využití *ACM* ontologie, kdy pro vstupní dotaz jsou nalezeny další pojmy z ontologie. Nové pojmy se připojují ke vstupnímu dotazu a v závislosti nastavení ontologie jej zobecňují, či konkretizují. Tzn. budou se vyhledávat odpovědi pro více dotazů a dostaneme větší množinu výsledků, viz podkapitola 7.3.

7.5 Srovnání metod hodnocení autorů

Výsledky vyhledávání metodou *PageRank* a *Indegree* byly srovnávány dvěma způsoby. Vždy bylo z výsledků hodnocení vzato prvních x nejlepších pozic, obsahující ohodnocené autory.

První způsob srovnání metod spočívá v porovnávání pořadí jednotlivých autorů napříč metodami. Tabulka 7.3 ukazuje procentuelní shodu pořadí autorů. Např. v prvních *5-ti* nejlepších výsledcích se pro dotaz *operating system* shodovalo pořadí *37,5%* autorů.

	Počet prvních pozic					
Dotaz	**5**	**10**	**20**	**30**	**40**	**50**
operating system	37,5	17,6	11,3	8,3	4,5	3,1
database application	30	38,2	23,9	8,8	7,9	7,9
database system	80	70	62,5	52,7	48,9	45,9
mobile devices	0	7,6	5,9	2,6	2,5	2,5
websites	0	7,6	14,3	9,6	6,1	6,1
database qieries	33,3	18,8	9,3	6,1	9,25	7
system Windows	21,2	22,6	21,8	21,8	21,8	21,8
document structure	20	12,1	18,7	21,8	15,9	18,9
system security	80	33,3	12,5	11	6,9	4,5
Průměr [%]	33,57	25,32	20,02	15,86	13,75	13,01

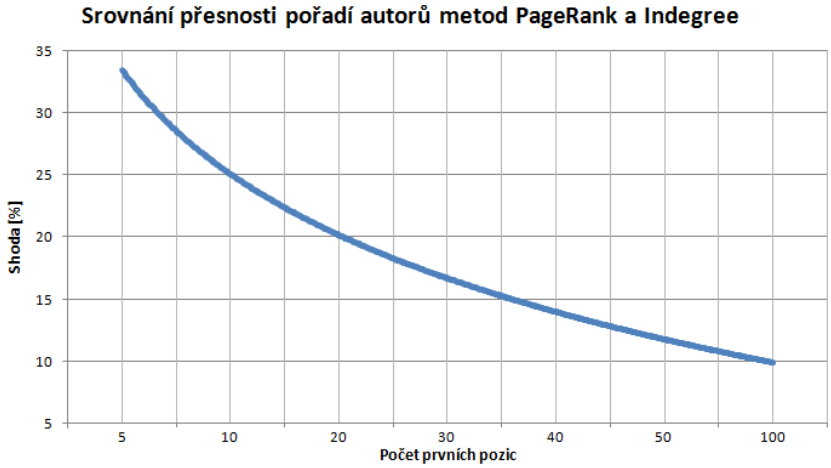
Tabulka 7.3: Shoda pořadí autorů ve výsledcích metod PR a Indegree

Druhá metoda srovnává výskyt shodných autorů ve výsledcích jedné a druhé metody bez ohledu na jejich pořadí. Tabulka 7.4 ukazuje procentuelní podíl shodných autorů pro několik prvních podmnožin výsledků hodnocení. Např. pro dotaz *operating system* a *5* prvních hodnocení se ve výsledcích obou metod vyskytuje *95,2%* shodných autorů.

	Počet prvních pozic					
Dotaz	**5**	**10**	**20**	**30**	**40**	**50**
operating system	95,2	98,8	98	98,1	98,4	99,9
database application	96	97,45	100	100	100	100
database system	95,2	97,4	99,9	99,9	99,9	100
mobile devices	93,6	96,4	97	100	100	100
websites	44,4	70	83,8	96,6	100	100
database qieries	52,4	98,7	97,4	99,7	100	100
system Windows	97,6	99,8	100	100	100	100
document structure	95	90,7	99,8	99,9	100	100
system security	95,2	90,5	96,4	99,9	99,9	100
Průměr [%]	84,96	93,31	96,92	99,34	99,8	100

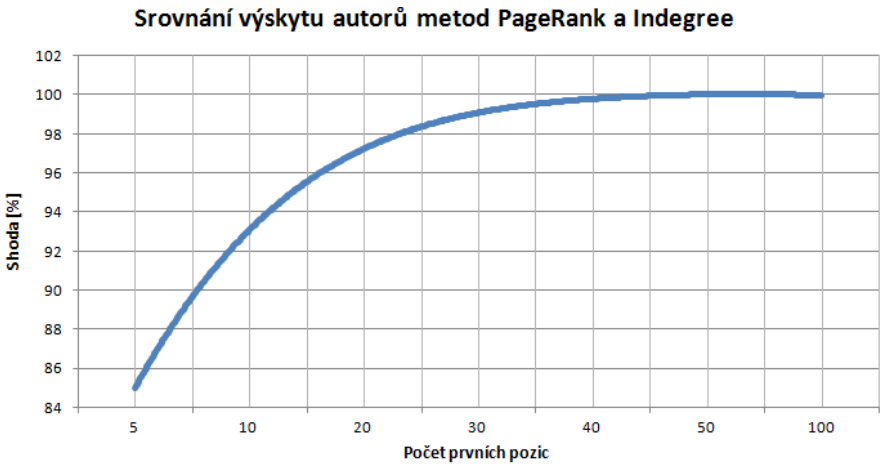
Tabulka 7.4: Výskyt stejných autorů ve výsledcích metod PR a Indegree

Na obrázku 7.2 je znázorněn průměrný průběh shody pořadí autorů s narůstajícím počtem výsledků (z tabulky 7.3). Graf znázorňuje, že metody se v hodnocení autorů s jejich narůstajícím počtem více rozcházejí.



Obrázek 7.2: Graf srovnání metod PR a Indegree – shoda pořadí autorů

Graf na obrázku 7.3 zobrazuje tabulku 7.4 a je z něj vidět, že metody pro prvních několik výsledků nevyhodnotí stejné autory jako ty nejlepší. S narůstajícím počtem výsledků se samozřejmě množina hodnocených autorů srovnává.



Obrázek 7.3: Graf srovnání metod PR a Indegree – shoda výskytu autorů

Porovnáním metod způsobem, jak je popsaný v podkapitole 7.6 pro *PageRank*, vyšel algoritmus *PageRank* jako lepší volbou pro využití v této práci, kdy významnější autoři byli řazeni v žebříčku výše.

7.6 Ověření relevance výsledků

Pro ověření, zda aplikace vyhledává na dotazy správné odpovědi, bylo využito ceny *ACM Fellows*. Seznam udělených cen má aplikace v souboru, viz podkapitola 7.4.3 a v případě, že autor danou cenu obdržel, je u něj zobrazena ve výpisu výsledků. Data, využívané aplikací obsahují seznam autorů a cen, neobsahují však informace, za co danou cenu autor obdržel. Aby bylo možné ověřit relevantnost výsledků vyhledávání, je potřeba položit aplikaci dotaz odpovídající tématice, které se oceněný autor věnuje. Pro tyto účely posloužil online seznam[7] udělených cen *ACM Fellows*, kde mimo samotný seznam obdarovaných autorů jsou i informace o oblasti, za kterou tuto cenu autor získal. Z informací náhodně vybraných autorů bylo vytvořeno několik dotazů, viz tabulka 7.5, týkající se oblasti jejich zájmu. Tyto dotazy byly následně aplikací vyhledány a proběhla kontrola, zda se ve výsledcích vyskytl příslušný autor, mající vazbu s tímto dotazem. V tabulce 7.5 je uvedeno několik příkladů ověření správnosti vyhledávání.

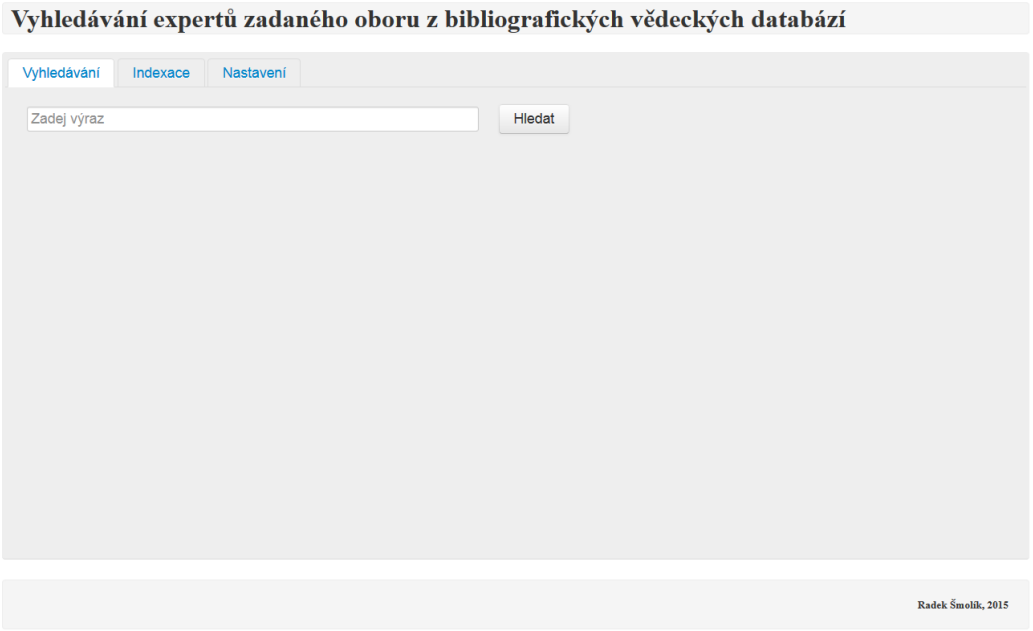
Dotaz	**Oceněný autor**	**Pozice ve výsledcích**
medical image analysis	Jain Anil K	14
error detection	Bose Bella	15
DBMS	Hellerstein Joseph M	7
management in wireless networks	Akyildiz Ian F	1
mobile network	Chlamtac Imrich	5
information retrieval	Croft Bruce W	17
approximation algorithms	Motwani Rajeev	2
real-time system	Shin Kang G	3
parallel and distributed computing	Kedem Zvi	10
internet applications	Paxson Vern	2

Tabulka 7.5: Ověření relevance vyhledávání

[7] http://awards.acm.org/fellow/year.cfm

7.7 Grafické rozhraní aplikace

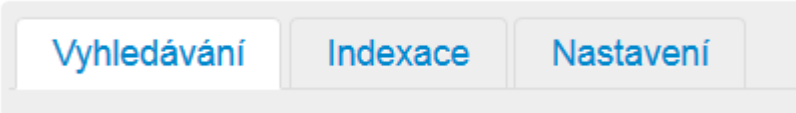
Prezentační vrstva aplikace je jednoduchá webová stránka (viz obrázek 7.4), vytvořena frameworkem *Primefaces*. Webová stránka je v jednoduchém a přehledném stylu. Vzhledem k aktuální době, kdy se hojně využívají mobilní zařízení, je stránka navržena jako responzivní, tj. je optimalizována pro všechny druhy zařízení (počítač, tablet, mobilní telefon, atd.). Prvky stránky se dynamicky upravují, v závislosti na jakém zařízení je zobrazována. To přináší komfort pro uživatele při práci s aplikací.



Obrázek 7.4: Úvodní stránka aplikace

Pro jednodušší ovládání aplikace, aniž by bylo nutné přepínat mezi více stránkami, bylo zvoleno využití záložek na jedné stránce (obrázek 7.5). Jednotlivé záložky pak obsahují příslušné funkce:

- *Vyhledávání* – vyhledávání v indexu a zobrazování výsledků vyhledávání
- *Indexace* – informace o indexu a možnost jeho vytvoření
- *Nastavení* – nastavení vyhledávání (ACM, lemmatizace, reference)



Obrázek 7.5: Záložky stránky

7.8 Databáze a logování

Aplikace má ve výchozí složce (podkapitola 8.4) uložen soubor databáze *log.db* a logu *log.csv*. Do těchto souborů se ukládají výrazy, které uživatel hledá. Záznamy jsou uloženy s časovou značkou a číslem, udávající, kolik výsledků se pro daný výraz našlo. Další využití databáze by bylo možné k doplňování nebo napovídání při psaní dotazu nebo by se data mohla využít pro vytvoření různých žebříčků, např. nejhledanější položky, apod.

Jako databáze je využita *SQLite*[8] se strukturou uvedenou v následující tabulce 7.6.

ID (int)	QUERY (string)	COUNT_RESULT (int)	DATE (datetime)

Tabulka 7.6: Struktura databáze pro logování

7.9 Lokalizace aplikace

Všechny texty grafického rozhraní aplikace jsou umístěny společně v jednom souboru *resources.properties*. To umožňuje velice snadnou lokalizaci aplikace do jakéhokoli jazyka, která spočívá v pouhém přeložení obsahu tohoto souboru.

[8]http://www.tutorialspoint.com/sqlite/sqlite_java.htm

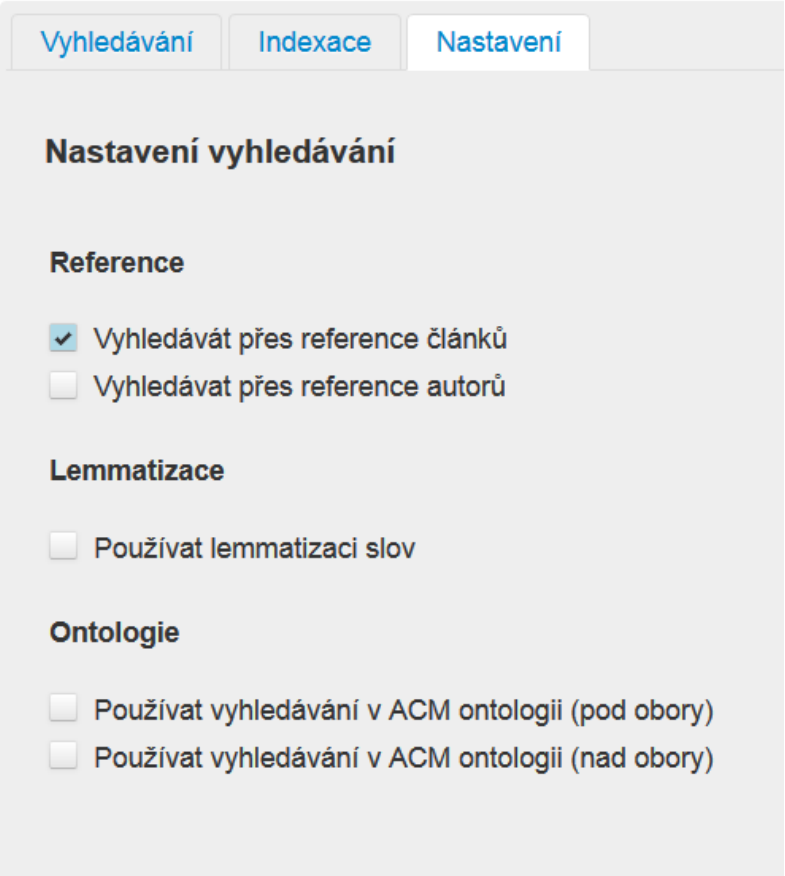
8 Uživatelská příručka

8.1 Ovládání aplikace

Pro ovládání aplikace jsou využity záložky na stránce, ve kterých se ovládá vyhledávání, indexace dat a nastavení aplikace.

8.1.1 Záložka nastavení

Na záložce nastavení lze ovlivnit chování aplikace při vyhledávání výsledků. Na obrázku 8.1 jsou vidět volitelné položky:



Obrázek 8.1: Záložka nastavení

Reference

- *Vyhledávat přes reference článků* – vyhledávání dalších článků, které jsou uvedeny v referencích nalezeného článku
- *Vyhledávat přes reference autorů* – vyhledávání dalších článků, které publikoval autor nalezeného článku

Lemmatizace

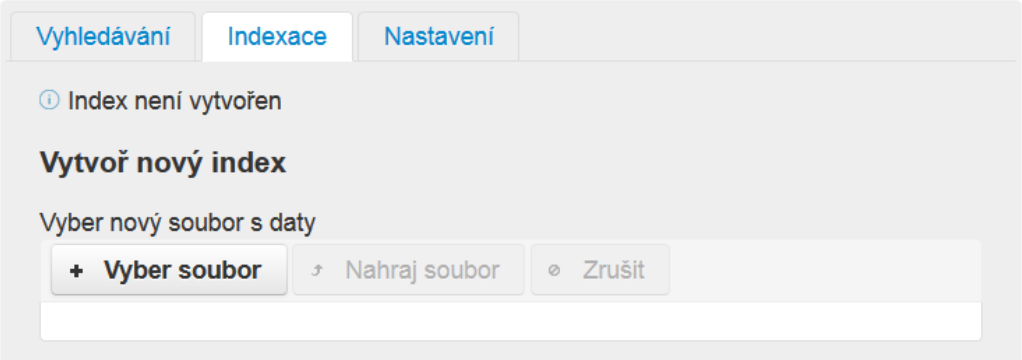
- *Používat lemmatizaci slov* – dotaz je rozšířen o nalezená lemmata

Ontologie

- *Používat vyhledávání v ACM ontologii (pod obory)* – připojení konkrétních ACM oblastí hledaného pojmu
- *Používat vyhledávání v ACM ontologii (nad obory)* – připojení obecných ACM oblastí hledaného pojmu

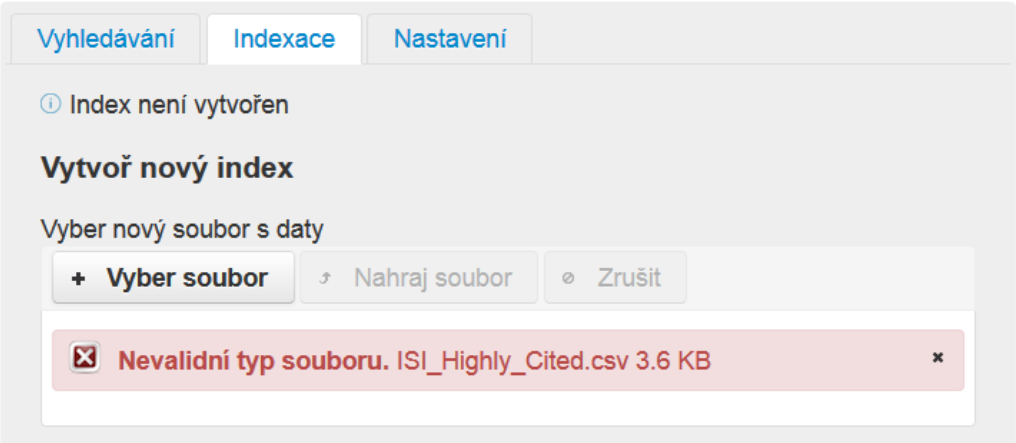
8.1.2 Záložka indexace

Záložka indexace zobrazuje informaci (obrázek 8.2), zda je, či ještě není vytvořen index z *WoS* dat.



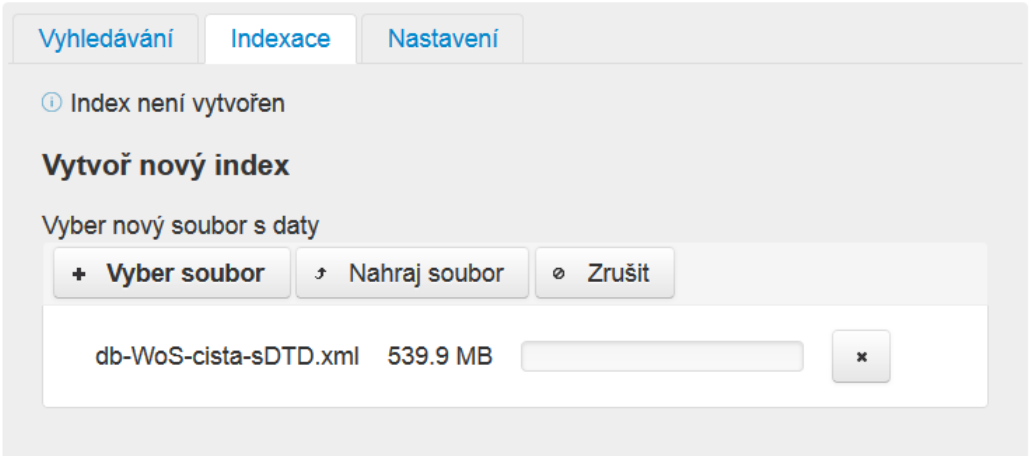
Obrázek 8.2: Záložka indexace

Dále obsahuje uploader souboru, který umožňuje vybrat tlačítkem *Vyber soubor xml* soubor s *WoS* daty. Uploader povoluje vybrat pouze jeden soubor, který kontroluje na základě přípony *xml*, takže nedovolí zvolení jiného typu souboru (obrázek 8.3). Jiné *xml* data jsou validována až na úrovni indexace dat, kdy probíhá kontrola správnosti struktury dat.



Obrázek 8.3: Záložka indexace – validace souboru

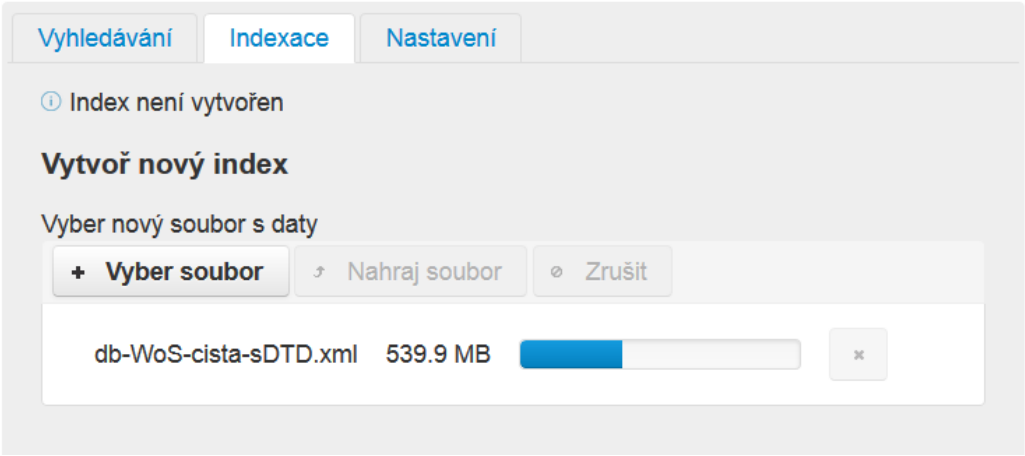
Výběrem správného typu souboru dojde k jeho zobrazení v oblasti uploaderu, viz obrázek 8.4.



Obrázek 8.4: Záložka indexace – výběr souboru

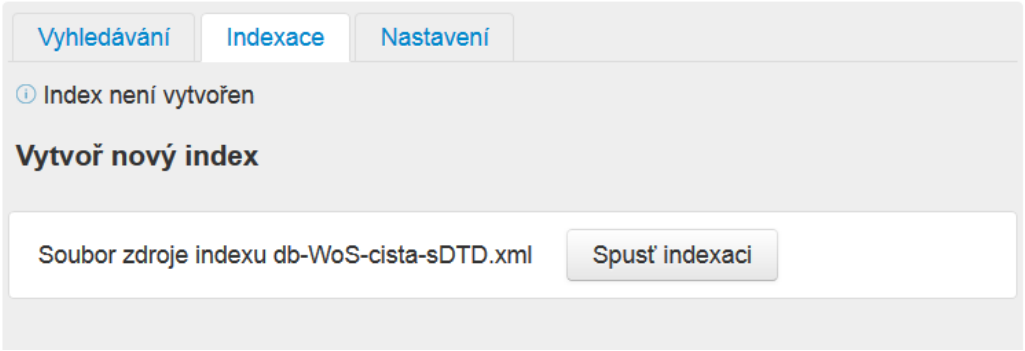
Vzhledem k možnosti nahrávat velké množství dat, resp. velké *xml* soubory, je uživateli notifikován průběh nahrávání souboru přes *"progres bar"*, viz obrázek 8.5.

Nahrávání souboru je spuštěno tlačítkem *Nahraj soubor* v liště uploaderu.



Obrázek 8.5: Záložka indexace – upload souboru

Nahráním souboru dojde k zobrazení tlačítka *Spust' indexaci* (obrázek 8.6), které jak název napovídá, spustí indexaci nahraného souboru a vytvoří jeho index do výchozí složky aplikace (viz podkapitola 8.4).

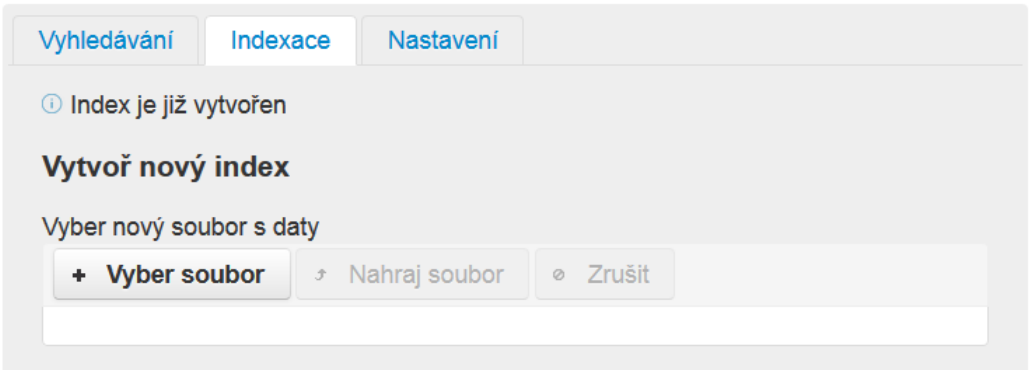


Obrázek 8.6: Záložka indexace – soubor nahrán

Doba indexace je závislá na velikosti indexovaných dat. Její dokončení je notifikováno bublinou v pravém horním rohu aplikace, viz obrázek 8.7. V případě vyskytnutí chyby (např. poškozená vstupní data) v průběhu indexace je stejnou formou vypsána informace o vyskytnutí chyby.

Úspěšně dokončená indexace způsobí změnu stavu informace o vytvoření indexu z *Index není vytvořen* na *Index je již vytvořen*, viz obrázek 8.8.

Obrázek 8.7: Notifikace – indexace dokončena

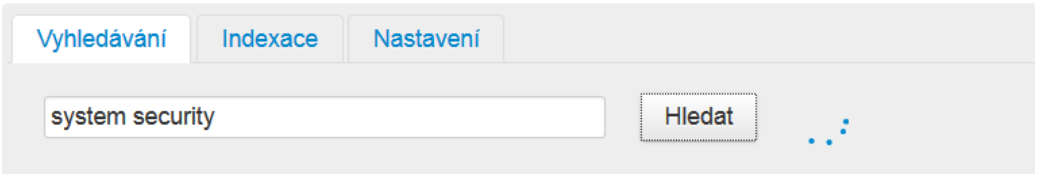


Obrázek 8.8: Záložka indexace – existující index

V případě, že dojde ke změně vstupních *WoS* dat, aplikace umožní výše popsaným způsobem nahrání nového souboru dat a jeho indexaci. Dojde tak k aktualizaci stávajícího indexu o nová data.

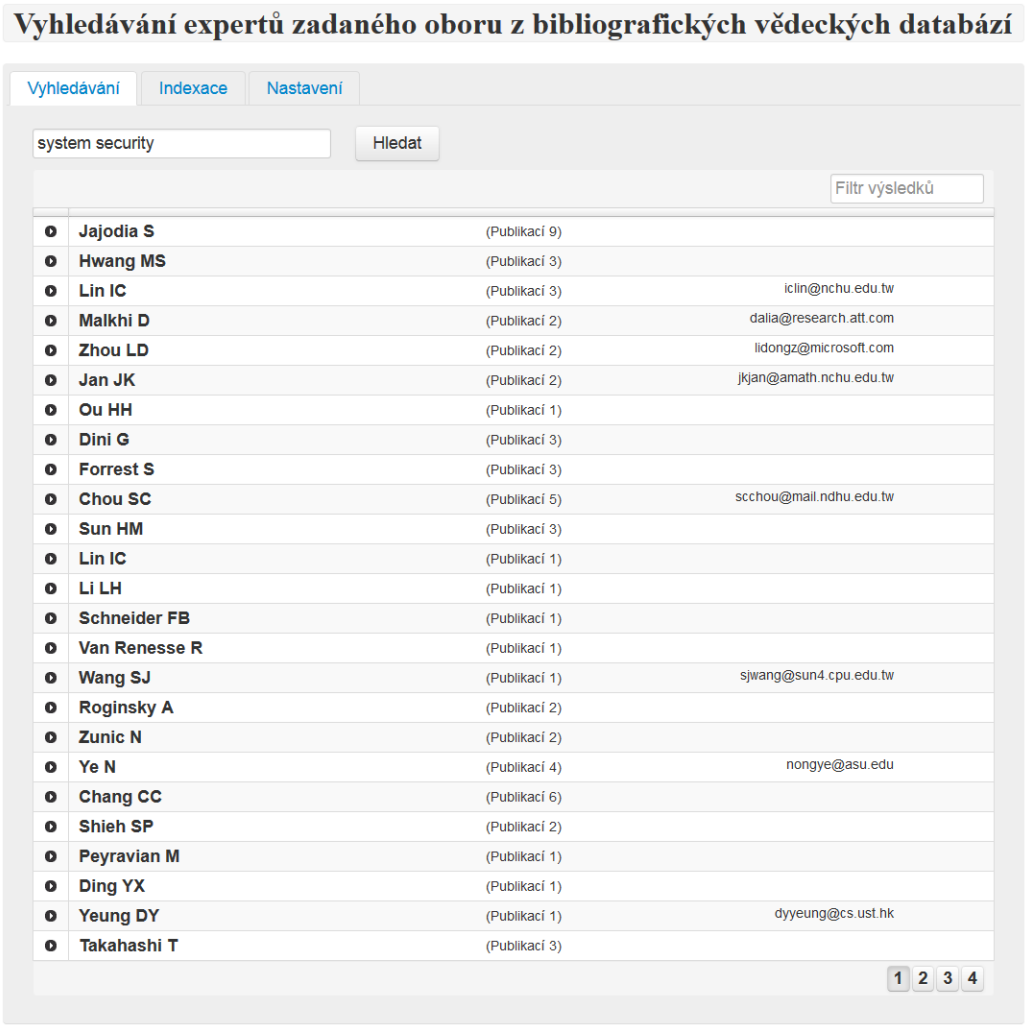
8.1.3 Záložka vyhledávání

Záložka vyhledávání (obrázek 7.4) je nastavena jako defaultní záložka aplikace a umožňuje zadáním výrazu do vyhledávajícího políčka a potvrzením tlačítka *Hledat* vyhledat výsledky k položenému výrazu. Vyhledávání a následné vyhodnocování výsledků může v závislosti na jejich počtu chvilku trvat. Aby měl uživatel od prostředí aplikace odezvu, že se něco děje, zobrazuje se mu vedle tlačítka *Hledat* animace, informující o probíhajícím hledání, viz obrázek 8.9.



Obrázek 8.9: Záložka vyhledávání – animace

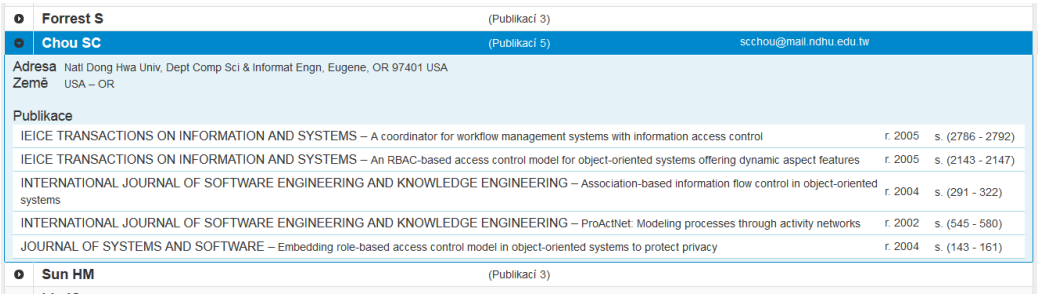
Po dokončení vyhledávání jsou na stránce zobrazeny výsledky vyhledávání formou seznamu autorů, viz obrázek 8.10.



Obrázek 8.10: Záložka vyhledávání – seznam položek

Každá řádka znamená jednu položku výsledku a obsahuje příjmení a zkratku jména autora, počet jeho publikací a email, je-li dostupný. Více informací o daném autorovi lze zobrazit kliknutím na ikonku šipky v levé části příslušné řádky autora. Dojde k rozbalení dané řádky (viz obrázek 8.11) a zobrazení dalších dostupných informací o autorovi. Zobrazuje se zde v případě dostupnosti v datech jeho adresa, ulice nebo země a dále tabulka autorovo publikací, obsahující název publikace, rok vydání a rozsah stránek, kde se naše hledaná informace v dané publikaci nachází.

Vzhledem k velkému počtu výsledků je seznam autorů rozdělen do stránek, obdobně jak tomu u jiných vyhledávacích stránek bývá. Počet zobrazovaných výsledků, neboli velikost jedné stránky je nastavena v ohledu na dnes velmi časté rozlišení monitorů (1920x1080 bodů). Jednoduché zobrazení výsledků se tak vejde na jednu obrazovku a omezí se tím nutnost dalšího posouvání stránky. Přepnutí na další stránku výsledků lze v pravé dolní části obrazovky, jak je vidět např. na obrázku 8.10.



Obrázek 8.11: Záložka vyhledávání – detail položky

Stránka umožňuje ve výsledcích další vyhledávání, respektive filtrování dle zvoleného kritéria. Filtrovat lze přes:

- *jméno autora*
- *příjmení autora*
- *adresu autora*
- *zemi autora*
- *stát autora*
- *město autora*
- *email autora*

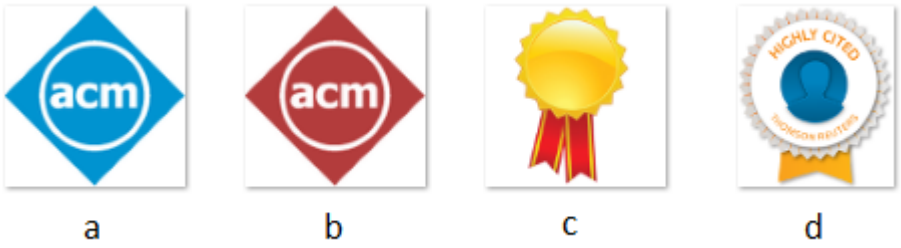
To nám umožní např. z výsledků vyfiltrovat pouze autory konkrétní země. Filtr výsledků je umístěn do pravého horního rohu seznamu publikací, viz obrázek 8.10. Filtrování výsledků probíhá v reálném čase, tedy zároveň s psaním vyhledávajícího kritéria na klávesnici.

V případě, že autor získal významnou cenu, je na jeho řádce ve výsledcích zobrazena ikonka dané ceny, viz obrázek 8.12.

Obrázek 8.12: Záložka vyhledávání – detail ocenění autora

Možné ikonky pro konkrétní ocenění jsou na obrázku 8.13, kde:

- a = ACM Fellow
- b = ACM Turing Award
- c = ACM SIGMOD Edgar F. Codd Innovation Award
- d= ISI Highly cited



Obrázek 8.13: Ikonky dostupných ocenění

Ukázka responzivnosti stránky, kdy dochází k přeuspořádání některých prvků je zobrazena na obrázku 8.14.

V případě, že se uživatel pokusí vyhledat odpověď na dotaz, ale nebyl zatím vytvořen žádný index *WoS* dat, dojde k oznámení tohoto faktu uživateli formou bubliny v pravém horním rohu aplikace, viz obrázek 8.15 s příslušnou hláškou.

Obrázek 8.14: Náhled responzivního vzhledu

Obrázek 8.15: Notifikace nevytvořeného indexu WoS

Stejným způsobem je řešena notifikace v případě, že uživatel v nastavení zvolí možnost využívat *ACM* ontologii a aplikace nemá její potřebný index. Objeví se hláška s textem *Není vytvořen index pro ACM, bude vyhledáváno bez ACM.*

Jak nahrát *ACM* index do aplikace je popsán v podkapitole 8.4.

8.2 Chybová hlášení

Seznam možných chyb při používání aplikace:

- *Není vytvořen index dat* – v případě pokusu vyhledat výsledky bez předem vytvořeného indexu dat
- *Nevalidní typ souboru* – v případě nahrání jiného než *xml* souboru pro indexaci dat
- *Při indexaci se vyskytla chyba (nevalidní data, zamčený soubor write.lock)* – v případě indexace souboru s poškozeným/jiným obsahem
- *Index dat poškozen (příčina soubor write.lock)* – v případě pokusu vyhledat výsledky když je index dat je poškozen
- *Není vytvořen index pro ACM, bude vyhledáváno bez ACM* – v případě vyhledávání výsledků s využitím ontologie, ale bez příslušného indexu ACM
- *Chyba v konfiguraci uploadDirectory* – špatná konfigurace defaultní složky aplikace

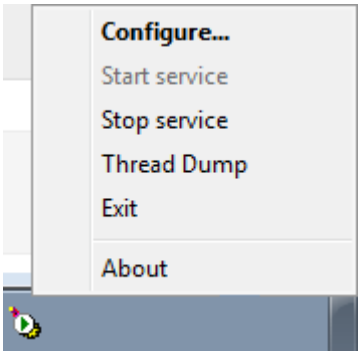
8.3 Nasazení aplikace

Pro spuštění aplikace budeme potřebovat prostředí *Java JRE*(Java Runtime Environment) nebo *JDK*(Java Development Kit)[1] a server *Apache Tomcat 7.0*[2] jako instalační verzi *Windows Service Installer* pro jeho snadnější konfiguraci.

Nejprve nainstalujeme *Java JRE* či *JDK* pomocí průvodce. Při instalaci serveru *Apache Tomcat* je pouze zapotřebí vybrat cestu k nainstalovanému *JRE, resp. JDK*, jinak nás zde nečekají žádné složité konfigurace. Po nainstalování serveru dojde k jeho automatickému spuštění. To lze ověřit ve webovém prohlížeči zadáním adresy:

```
localhost:8080
```

Zobrazí se nám domovská stránka serveru. Pokud se server automaticky nespustí, lze ho spustit manuálně přes položku *Start service* v oznamovací oblasti Windows (obr. 8.16).



Obrázek 8.16: Menu serveru Apache Tomcat

Aplikaci, resp. balíček *app.war* nakopírujeme do složky *webapps* a provedeme restart serveru. Složka *webapps* se nachází v adresáři nainstalovaného serveru.

Př. *C:/Program Files/Apache Software Foundation/Tomcat 7.0/webapps*

[1] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[2] https://tomcat.apache.org/download-70.cgi

Restart serveru se provede zastavením a opětovném spuštění serveru položkami *Stop service* a *Start service* (obr. 8.16).

Po restartu serveru se ve složce *webapps* vytvoří složka se shodným názvem jako je název *war* balíčku aplikace, obsahující soubory aplikace. Nyní je aplikace dostupná na webové adrese

`http://localhost:8080/app/faces/Index.xhtml`

kde *app* je název odpovídající názvu *war* balíčku aplikace.

V případě, že nechceme aplikaci provozovat pouze lokálně, potřebujeme mít k dispozici *J2EE* aplikační server. Na server pak jen nahrajeme *war* balíček aplikace a server se o vše ostatní postará sám. Nemusíme již provádět konfiguraci popsanou v této kapitole.

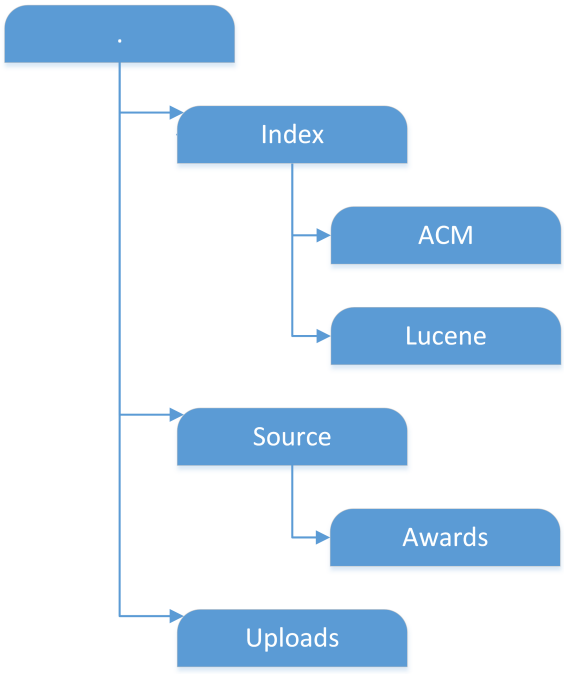
8.4 Konfigurace aplikace

Aplikace používá pevný disk k ukládání dat. Je nutné před jejím použitím zvolit výchozí adresář, který aplikace bude využívat. Volba adresáře se provede v konfiguračním souboru aplikace *web.xml* umístěného v adresáři *webapps/app/WEB-INF*. Parametru s názvem *uploadDirectory* určíme hodnotou cestu k výchozímu adresáři:

```
<context-param>
    <param-name>uploadDirectory</param-name>
    <param-value> C:/tmp_dir/</param-value>
</context-param>
```

Defaultní hodnota výchozího adresáře aplikace je: *C:/tmp_dir/*

Po konfiguraci a spuštění aplikace ve webovém prohlížeči dojde k vytvoření výchozí složky a adresářové struktury, potřebné k práci aplikace (obrázek 8.17).



Obrázek 8.17: Adresářová struktura aplikace

- *Index* – obsahuje složky indexů dat ontologie a vstupních dat *Web of Science*
- *Source/Awards* – *csv* soubory s údaji o udělených cenách
- *Uploads* – složka pro upload souboru z GUI aplikace
- . – obsahuje soubor s logováním aplikace a soubor databáze s logy

Dostupná paměť

V konfiguraci serveru *Tomcat*, položka *Configure* viz obrázek 8.16 v záložce *Java* zvýšíme velikost paměti *Maximum memory pool* na hodnotu minimálně *512MB*. Bez zvýšení paměti narazíme při používání aplikace na nedostatek paměti (*Java heap space*).

Ontologie

Pro možnost využívat ontologii v aplikaci je zapotřebí rozbalit do složky výchozího adresáře *./Index/ACM/* data, obsažená v archivu *ACM.zip*. Jedná se o předem vytvořený index dat ontologie.

Udělené ceny

Pro funkčnost zjišťování udělených cen autorů je zapotřebí rozbalit archiv *Awards.zip* do složky *./Source/Awards/*. Archiv obsahuje soubory:

- *ACM_Fellows.csv*
- *ACM_Turing_Award.csv*
- *Codd_award.csv*
- *ISI_Highly_Cited.csv*

V souborech jsou jména autorů, kterým byla za svoji činnost udělena příslušná cena.

Index dat

Před prvním použitím aplikace je nutné vytvořit index dat. Soubor s daty *db-WoS-cista-sDTD.xml* vybereme tlačítkem *Vyber soubor* a nahrajeme na server tlačítkem *Nahraj soubor*. Průběh uploadu souboru je možné sledovat na progress baru. Po nahrání souboru se tlačítkem *Spust' indexaci* zahájí indexování nahraného souboru dat. V závislosti na velikosti dostupných dat a výkonu počítače bude indexace dat chvilku trvat (řádově jednotky minut).

V případě rychlého zprovoznění aplikace je možné využít již předem vytvořeného indexu *WoS* dat, který je dostupný v archivu *Index.zip*, který nahrajeme do složky *./Source/Index/Lucene/*.

Soubory konfiguračních dat

Výše zmíněné soubory, potřebné pro konfiguraci aplikace jsou k práci přiloženy v adresáři *files*, který obsahuje i samotný *war* balíček aplikace.

9 Závěr

Cílem této práce bylo vytvořit program, schopný vyhledávat experty a publikace zadaného oboru z bibliografických vědeckých databází a prezentovat je uživateli.

Program využívá data databáze *Web of Science*, pro které vytváří index, ve kterém následně vyhledává odpovědi na příslušné dotazy. Při zpracování dat se provádí eliminace nevýznamových slov, využívá se lemmatizace a ontologie, aby byly výsledky vyhledávání co nejrelevantnější. Pro hodnocení vyhledaných expertů se využívá algoritmus *PageRank*, který experty řadí dle jejich prestižnosti. Výsledky jsou vizualizovány jako webová stránka v prohlížeči, kde se program také ovládá.

Program je schopen v rozumném čase vyhledat experty, provést hodnocení jejich prestižnosti a zjistit, zda jim nebyla udělena některá z významných cen. Ověření relevantnosti výsledků vyhledávání bylo provedeno na několika příkladech, viz podkapitola 7.6 a bylo potvrzeno, že program vyhledává experty daného oboru a relevantně je hodnotí. V případě dotazování velmi obecného pojmu, může vyhledávání a vyhodnocování programu trvat déle, nebot' bude docházet k vyhodnocování velkého objemu dat. Tomuto problému by bylo vhodné se v případě pokračování v této práci věnovat a čas vyhledávání obecných dotazů co nejvíce snížit. Vzhledem k charakteru aplikace je možné jako další rozšíření navrhnout mobilní aplikaci, která bude prezentovat výsledky vyhledávání poskytnuté serverem.

Seznam obrázků

Seznam tabulek

Literatura

[1] Pokorný Jaroslav, Snášel Václav, Kopecký Michal, *Dokumentografické Informační Systémy*, Karolinum, 2005, ISBN: 80-246-1148-1.

[2] Christopher D.Manning, Prabhakar Raghavan, Hinrich Schütze, *An Introduction to Information Retrieval*, Cambridge University Press, 2009.

[3] Otis Gospodnetic, Erik Hatcher, *Lucene in Action*, Manning Publications Co., 2005, ISBN: 1-932394-28-1.

[4] Lili Lin, Zhuoming Xu, Ying Ding, Xiaozhong Liu, *Finding topic-level experts in scholarly networks*, Akademiai Kiado, 2013.

[5] *Apache Lucene Documentation*, [28.4.2015]
`https://lucene.apache.org/core/5_1_0/index.html`.

[6] *The Stanford Natural Language Processing*, [28.4.2015]
`http://nlp.stanford.edu/software/corenlp.shtml`.

[7] *Apache Commons Digester*, [28.4.2015]
`https://commons.apache.org/proper/commons-digester/`.

[8] *ACM Digital Library*, [28.4.2015]
`http://dl.acm.org/ccs_flat.cfm`.

[9] *ACM Digital Library, xml*, [28.4.2015]
`http://dl.acm.org/ft_gateway.cfm?id=2371137&ftid=1290922&dwn=1`.

[10] *PrimeFaces JSF Framework Documentation*, [28.4.2015]
`http://www.primefaces.org/documentation`.

[11] *Apache Maven Documentation*, [28.4.2015]
`https://maven.apache.org/`.

[12] Pavel Herout, *Java a XML*, Koop, 2007, ISBN: 978-80-7262-307-4.

A Seznam příloh

A.1 Ukázka struktury dat *WoS*

```
<REC inst_id="9" recid="146783686" hot="yes"
      sortkey="3134521239" timescited="50"
          sharedrefs="0" inpi="false">
 <item issue="146783685" recid="146783686"
      coverdate="200506" sortkey="3134521239"
          refkey="3593923" dbyear="2005">
  <ut>000232597900001</ut>
  <i_ckey>KELL0083050037AC</i_ckey>
  <i_cid>0108321202</i_cid>
  <source_title>ACM COMPUTING SURVEYS</source_title>
  <source_abbrev>ACM COMPUT SURV</source_abbrev>
  <item_title>Lowering the barriers to programming:
     A taxonomy of programming environments and
        languages for novice programmers</item_title>
  <sq>10785J0</sq>
  <bib_id>37 (2): 83-137 JUN 2005</bib_id>
  <bib_pages begin="83" end="137" pages="55" />
  <bib_issue year="2005" vol="37"/>
  <doctype code="@">Article</doctype>
  <editions full="SCI"/>
  <languages count="1">
      <primarylang code="EN">English</primarylang>
  </languages>
  <authors count="2">
      <primaryauthor>Kelleher, C</primaryauthor>
      <author key="5408058">Pausch, R</author>
  </authors>
  <emails count="1">
      <email>
          <name>Kelleher, C</name>
           email_addr>caitlin+@cs.cmu.edu</email_addr>
      </email>
  </emails>
  <keywords count="8">
      <keyword>design</keyword>
      <keyword>languages</keyword>
      <keyword>human factors</keyword>
      <keyword>human-computer interaction</keyword>
```

```
    <keyword>computer science education</keyword>
    <keyword>literacy</keyword>
    <keyword>learning</keyword>
    <keyword>problem solving</keyword>
</keywords>
<reprint>
    <rp_author>Kelleher, C</rp_author>
    <rp_address>Carnegie Mellon Univ, Dept Comp Sci,
      5 Forbes Ave, Pittsburgh,15213 USA</rp_address>
    <rp_organization>Carnegie Mellon
      Univ</rp_organization>
    <rp_suborganizations count="1">
    <rp_suborganization>Dept Comp
      Sci</rp_suborganization>
</rp_suborganizations>
   <rp_street>5 Forbes Ave</rp_street>
    <rp_city>Pittsburgh</rp_city>
    <rp_state>PA</rp_state>
    <rp_country>USA</rp_country>
    <rp_zips count="1">
      <rp_zip location="AP">15213</rp_zip>
    </rp_zips>
</reprint>
<research_addrs count="1">
    <research>
      <rs_address>Carnegie Mellon Univ, Dept Comp Sci,
          Pittsburgh, PA 15213 USA</rs_address>
      <rs_organization>Carnegie Mellon
          Univ</rs_organization>
      <rs_suborganizations count="1">
          <rs_suborganization>Dept Comp
          Sci</rs_suborganization>
      </rs_suborganizations>
      <rs_city>Pittsburgh</rs_city>
      <rs_state>PA</rs_state>
      <rs_country>USA</rs_country>
      <rs_zips count="1">
          <rs_zip location="AP">15213</rs_zip>
      </rs_zips>
    </research>
</research_addrs>
```

```
  <abstract avail="Y" count="1">
      <p>Since the early 1960's, researchers have built
        a number of programming languages and ...</p>
  </abstract>
  <refs count="3">
      <ref>146783687</ref>
      <ref>146783688</ref>
      <ref>146783689</ref>
  </refs>
  </item>
</REC>
```

A.2 Ukázka struktury dat *ACM*

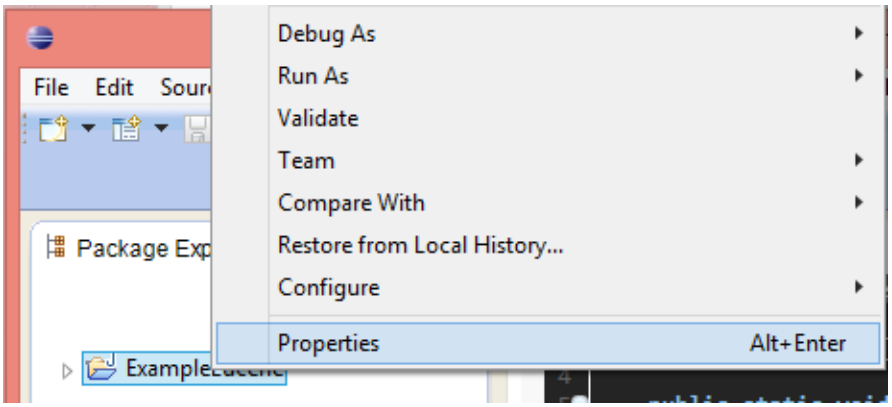
```
<skos:Concept rdf:about="#10002952" xml:lang="en">
  <skos:prefLabel xml:lang="en">
    Data management systems
  </skos:prefLabel>
  <skos:altLabel xml:lang="en">
    database
  </skos:altLabel>
  <skos:altLabel xml:lang="en">
    database management systems
  </skos:altLabel>
  <skos:altLabel xml:lang="en">
    database management system
  </skos:altLabel>
  <skos:altLabel xml:lang="en">
    databases
  </skos:altLabel>
  <skos:altLabel xml:lang="en">
    data management system
  </skos:altLabel>
  <skos:altLabel xml:lang="en">
    dbms
  </skos:altLabel>
  <skos:related rdf:resource="#10003018"/>
  <skos:inScheme rdf:resource="http://totem.semedica.
    com/taxonomy/The ACM Computing Classification
      System (CCS)"/>
  <skos:broader rdf:resource="#10002951"/>
  <skos:narrower rdf:resource="#10002953"/>
  <skos:narrower rdf:resource="#10002971"/>
  <skos:narrower rdf:resource="#10003190"/>
  <skos:narrower rdf:resource="#10003197"/>
  <skos:narrower rdf:resource="#10003212"/>
  <skos:narrower rdf:resource="#10003219"/>
  <skos:narrower rdf:resource="#10003400"/>
</skos:Concept>
```

A.3 Konfigurace knihoven v Eclipse IDE

V případě, že se rozhodneme pracovat se zdrojovým kódem aplikace a nevyužijeme dostupný funkční projekt, máme dvě možnosti, jak programový kód zkompilovat a spustit. Do projektu můžeme manuálně přidat všechny potřebné knihovny nebo využít nástroje *Apache Maven*, který knihovny stáhne a nainstaluje za nás. Pro ukázku použijeme *Eclipse IDE*.

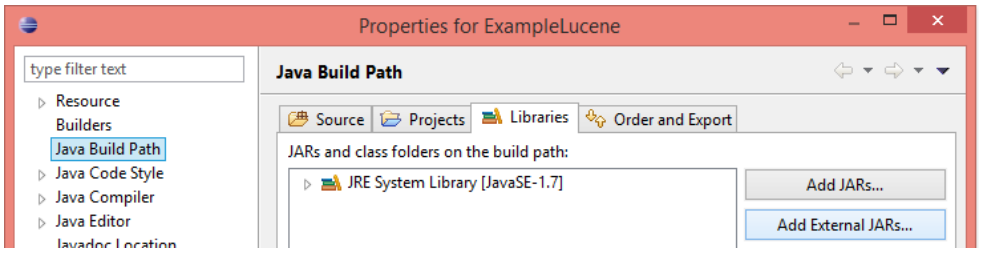
Manuální

V prostředí *Eclipse* si nad projektem vybereme možnost *Properties*, viz obrázek 9.1.



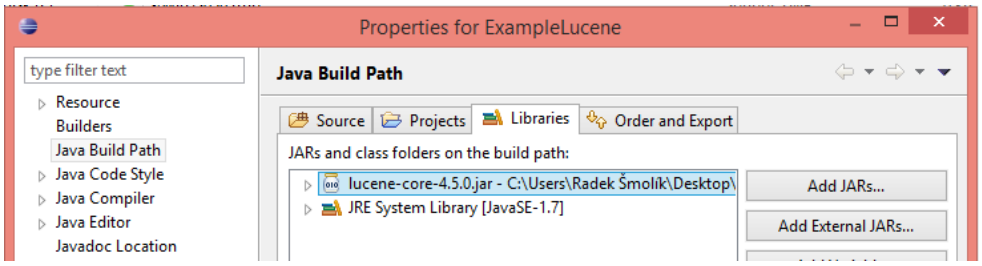
Obrázek 9.1: Vlastnosti projektu

V levém menu okna vybereme položku *Java Build Path* a v záložce *Libraries* volbu *Add External JARs*, obrázek 9.2.



Obrázek 9.2: Přidání knihovny do projektu

V nově otevřeném okně najdeme umístění přidávané knihovny a vybereme *jar* soubor knihovny.

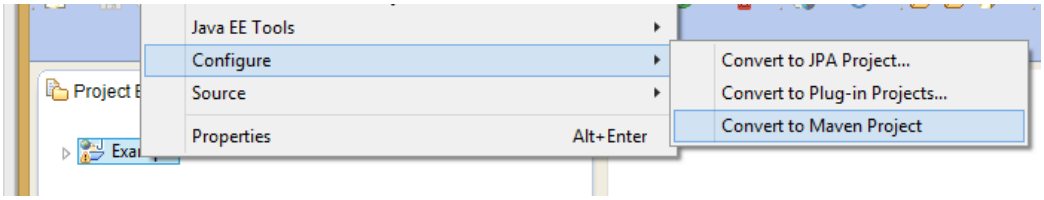


Obrázek 9.3: Java Build Path

V okně knihoven 9.3 máme zobrazenu nově přidanou knihovnu. Takto přidáme všechny potřebné knihovny projektu.

Využití Apache Maven

Abychom mohli využít nástroje *Maven*, je zapotřebí zkonvertovat projekt na *Maven* projekt. V projektu přes položku *Configure* zvolíme možnost *Convert to Maven Project*, viz obrázek 9.4.



Obrázek 9.4: Vytvoření Maven projektu

Po převodu projektu dojde k vytvoření souboru *pom.xml*, sloužícího k přidávání knihoven do aplikace. Pro přidání knihovny do projektu je zapotřebí do souboru *pom.xml* vložit následující informace o knihovně:

```
<dependency>
        <groupId>org.apache.lucene</groupId>
        <artifactId>lucene-core</artifactId>
        <version>4.10.1</version>
</dependency>
```

- *groupId* – jméno balíčku knihovny
- *artifactId* – název *jar* knihovny
- *version* – verze knihovny

Při kompilování projektu si *Maven* z konfiguračního souboru *pom.xml* zjistí, jaké knihovny projekt vyžaduje a automaticky provede jejich stažení a přidání do projektu.

Dostupnost knihoven a jejich verzí je možné vyhledat online[1] v *Maven* repositáři.

[1] http://mvnrepository.com