ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ
KATEDRA ŘÍDICÍ TECHNIKY

Kontextové prehľadávanie textu

Bakalárska práca

2006

Vypracoval: Rastislav Biroščák

Vedúci práce: Ing. Richard Šusta Ph.D.

Prehlásenie

Prehlasujem, že som svoju bakalársku prácu vypracoval samostatne a použil som iba podklady (literatúru, projekty, software atď.) uvedené v priloženom zozname literatúry.

V Prahe dňa .....26.5.2006..... .....Birošťák.....

podpis

České vysoké učení technické v Praze, fakulta elektrotechnická

Katedra řídicí techniky Školní rok:2004/2005

Zadání bakalářské práce

Student: Rastislav Biroščák

Obor: Kybernetika a měření

Název tématu: Kontextové prohledávání textu

Zásady pro vypracování:

1. Prostudujte otázku hledaní okolí slova.
2. Navrhněte vhodnou strukturu potřebných indexů.
3. Vyzkoušejte navrženou strukturu pomocí realizace jednoduché verze systému.

Seznam odborné literatury: Dodá vedoucí práce

Vedoucí bakalářské práce: Ing. Richard Šusta, Ph.D.

Datum zadání bakalářské práce: únor 2005

Termín odevzdání bakalářské práce: 20. 1. 2006

Prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-

v zast.

Prof. Ing. Vladimír Kučera, DrSc.
děkan

Anotácia

V tejto bakalárskej práci sa zaoberáme IR (Information Retrieval) systémami, presnejšie TR (Text Retrieval) z hľadiska indexácie a vyhľadávania vo väčšom množstve dokumentov, za účelom rýchleho vyhľadávania v texte z kolekcie dokumentov. Uvádzame prehľad metód, ktoré sa používajú pri spracovaní textu dokumentov z dôvodu rýchleho vyhľadávania.

Pri návrhu IR systému sme použili knižnicu DotLucene pre C# (port Java Lucene open-source projekt z dielne Jakarta Apache). Výsledkom návrhu vznikol vyhľadávaci nástroj, vytvarajúci Lucene index. Ten je možno zdielať so všetkými vyhľadávacími nástrojmi založenými na ktoromkoľvek porte Lucene API.

Annotation

In this bachelor thesis we deal with IR systems, mainly TR from the point of view of indexing and searching in bigger amount of documents, in order to search the text faster from a collection of documents. We introduce list of methods that are used in processing document texts so as to search in the text faster.

When designing IR system, DotLucene library (C# port Java Lucene open-source project of the Jakarta Apache) was used. The result of the design was a search tool *Rexo* creating Lucene index. It is possible to share it with all search tools based on any of the Lucene ports.

Poďakovanie

Chcel by som poďakovať Ing. Richardu Šustovi, PhD. za odborné vedenie bakalárskej práce.

Velká vďaka patrí mojej manželke a priateľom, pretože počas štúdia a písania bakalárskej práce mi poskytovali zázemie a duševnú podporu.

Obsah

ÚVOD 3

1. PREHĽAD V OBLASTI TEXTOVÝCH INFORMAČNÝCH SYSTÉMOCH 4

1.1. INFORMAČNÉ SYSTÉMY 4

1.2. ALGORITMY PRE VYHĽADÁVANIE V TEXTE 5

1.2.1. ELEMENTÁRNY ALGORITMUS 7

1.2.2. METÓDY S PREDSPRACOVANÍM HĽADANÉHO SLOVA 7

1.2.2.1. Metódy s predspracovaním slova so súmerným vyhľadávaním: 7

1.2.2.2. Metódy s predspracovaním slova s protismerným vyhľadávaním: 8

1.2.3. METÓDY S PREDSPRACOVÁNÍM TEXTU – INDEXOVÉ METÓDY 8

1.2.3.1. Implementácia indexových systémov 8

1.2.3.2. Invertovaný index 9

1.2.3.3. Zoznam dokumentov 11

1.3. METÓDY S PREDSPRACOVANÍM TEXTU A HĽADANÉHO SLOVA – SIGNATUROVÉ METÓDY 11

1.3.1. REŤAZOVÉ A VRSTVENÉ KÓDOVANIE 12

1.3.2. VYHLADAVANIE SLOVA POMOCOU INDEXU A SIGNATURY 12

2. NÁVRH INFORMAČNÉHO SYSTÉMU 13

2.1. NÁVRH LOGICKEJ ŠTRUKTÚRY INDEXU S VYUŽITÍM LUCENE 14

2.1.1. FIELDY 15

2.1.2. LUCENE DOKUMENT 15

2.1.3. SEGMENT 17

2.1.4. INDEX 18

2.1.5. DVE MOŽNÉ KOMPOZÍCIE LUCENE INDEXU 18

2.1.5.1. Mnohosúborová štruktúra indexu 18

2.1.5.2. Štruktúra zlúčených index súborov 20

2.2. VOĽBA INVERTOVANÉHO INDEXU 21

2.2.1. NÁZVY FIELDOV: 21

2.2.2. SLOVNÍK TERMOV 23

2.2.3. FREKVENCIA VÝSKYTU TERMOV 23

2.2.4. POZÍCIA TERMOV 24

2.3. INDEXUJEME DOKUMENTY DO INDEXU 24
2.3.1. ZÁKLADNÉ TRIEDY PRE INDEXOVANIE 24
2.3.1.1. IndexWriter 25
2.3.1.2. Directory 25
2.3.1.3. Analyzer 25
2.3.1.4. Document 26
2.3.1.5. Field 26
2.3.2. ANALÝZA TEXTU 27
2.3.3. VYTVÁRANIE INDEXU – INDEXWRITER 29
2.3.4. OPTIMALIZÁCIA VYTVORENÉHO INDEXU 30
2.4. VYHĽADÁVANIE 31
2.4.1. ZÁKLADNE TRIEDY PRE VYHĽADÁVANIE 31
2.4.1.1. IndexSearcher 31
2.4.1.2. Term 32
2.4.1.3. Query 32
2.4.1.4. TermQuery 32
2.4.1.5. Hits 33
2.4.2. SPRACOVANIE QUERY – PARSOVANIE 34
2.4.3. LUCENE SCORING 35
2.4.4. VYHĽADÁVANIE PODĽA QUERY – INDEXSEARCHER 36
2.4.5. HIGHLIGHTER - ZOBRAZENIE OKOLIA HĽADANÉHO VÝRAZU 37
3. REALIZÁCIA INFORMAČNÉHO SYSTÉMU 39
3.1. REXO – NÁŠ VYTVORENÝ IR SYSTÉM 39
3.1.1. SCHOPNOSTI NÁŠHO IR SYSTÉMU 40
3.2. EXPERIMENT 41
ZÁVER 44
LITERATÚRA 45

Úvod

Súčasná doba je poznamenaná neustálym hľadaním, získavaním a vytváraním nových informácií a dát. Ich množstvo sa každodenne zväčšuje a už nie je obmedzené, ako v minulosti, veľkosťou pamäťových médií. To postupne spôsobuje stratu prehľadu o tom, čo kde je presne umiestnené v tom nesmiernom záplave dát.

Väčšina týchto informácií a dát je vytváraná, sprostredkovaná a uchovávaná v textovej podobe. A práve kvôli tomu, sa kladie dôraz na techniku spracovania textových dokumentov.

V tejto bakalárskej práci sa zaoberáme optimálnym vyhľadávaním v textových súboroch. Túto oblasť popisujú informačné systémy, ktoré vyhľadávajú informácie v texte na základe požiadaviek pre nich špecifikovaných .

1. Prehľad v oblasti textových informačných systémoch

V úvodnej časti práce sú opísané druhy informačných systémov v závislosti na ich zameraní. Ďalej sú stručne rozobrané základné metódy, ktoré sa využívajú v informačnom procese pri vyhľadávaní. Hlavnú časť informácií som čerpal z literatúry [1] a internetových stránok.

1.1. Informačné systémy

Information Retrieval (IR) – je Informačný systém systém, ktorý umožňuje účelné usporiadanie zberu, uchovanie, spracovanie a poskytovanie informácií.

Text Retrieval (TR) – ide o IR systém využívaný k získavaniu informácií z textových, webových dokumentov na základe dotazu.

Informácia – je odraz poznaného alebo predpokladaného obsahu skutočnosti. Možno ju definovať vždy iba vo vzťahu k systému, pre ktorý je určená.

Informačný proces – je proces vzniku informácií, ich zobrazenia vo forme údajov, uchovania, spracovania, poskytovania a využitia.

Najdôležitejšie IR systémy predstavujú:

1) databázové systémy – systémy využívajúce databázy a v nich uložené dáta, ktoré sú zoradené v súboroch. Systémy na vyhľadávanie informácií udržujú súbory údajov a spracuvávajú dotazy, na základe ktorých identifikujú a vrátia záznamy, ktoré sú relevantné pre daný dotaz. Vrátenie záznamu závisí od definovania podobnosti, ktorá môže byť napríklad určená porovnaním hodnôt atribútov priradených konkrétnemu záznamu a dotazu.

2) dokumentografické vyhľadávacie systémy – dáta sú uložené v textových dokumentoch. Pri vyhľadávaní sa zadá hľadaný výraz alebo slovné spojenie a výsledkom hľadania je množina záznamov, ktorá obsahuje daný výraz, či slovné spojenie. Oproti

databázovým, kde je informácia štrukturálne usporiadaná podľa daného typu dát, tak tento druh systému má menšiu štruktúrovateľnosť informácií.

3) faktografické systémy pre riadenie – rozšírený databázový systém o funkcie, ktoré mu umožňujú manipulovať s informáciami uloženými v databáze tak, aby mohol vykonávať určitú riadiacu činnosť.

4) systémy pre podporu rozhodovania – systém, ktorý obsahuje všetky tri predošlé.

5) expertné systémy – používa databázu a rozhodovací aparát, vytvorený na základe znalostí, ktoré sú mu dodané.

Databázové systémy a systémy pre riadenie spracovávajú štruktúrované údaje často vo forme číselných údajov. Vyhľadávajúce dokumentografické systémy a expertné systémy spracovávajú údaje v prirodzenom jazyku. Vyhľadávajúce systémy vyhľadávajú dokumenty a expertné systémy vyhľadávajú fakta potrebné k odpovedi na zadaný dotaz.

Našou úlohou bude zamerať sa na dokumentografické vyhľadávacie systémy, ktoré sa orientujú na vyhľadávanie v dokumentoch.

1.2. Algoritmy pre vyhľadávanie v texte

Metódy na vyhľadávanie slova v texte sa nezaoberajú iba tým, či sa hľadané slovo v texte vyskytuje, ale aj tým, či sa dané slovo v texte nachádza. Môžme ich rozdeliť do dvoch skupín:

Prvú skupinu tvoria algoritmy, ktoré prehľadávajú samotný text, čo môže byť dosť časovo náročné.

Druhá skupina predstavuje metódy, ktoré v úvodnej fáze predspracujú text do takej formy, ktorá umožňuje rýchlejšie vyhľadávanie. Formy, ktoré charakterizujú text môžu byť:

- **index** - ktorý predstavuje zoznam významných prvkov s odkazom do pôvodného textu.
- **signatúra** – t.j. reťazec príznakov indikujúci prítomnosť významných prvkov v texte.

Ďalším kritériom, ktorým môžeme klasifikovať vyhľadávajúce metódy je to, či metóda predspracuváva text alebo iba dané slovo alebo oboje.

Metóda vyžaduje		Predspracovávanie textu	
		nie	áno
Predspracovávanie slov	nie	I	III
	áno	II	IV

- Elementárny algoritmus (I) – nevyžaduje predspracovanie textu ani hľadaného slova.

- Metódy (II), ktoré pre dané hľadané slovo vytvoria najprv vyhľadávajúci stroj, ktorý potom uskutočňuje vyhľadávanie.

- Indexové metódy (III) – sú také, ktoré pre daný text, v ktorom sa bude vyhľadávať vytvoria index (usporiadaný zoznam slov s odkazmi na ich umiestnenie v texte).

- Signaturové metódy (IV) – vytvoria pre hľadané slovo a pre prehľadávaný text reťazce bitov (signatúry), ktoré charakterizujú ako text, tak i hľadané slovo. A prehľadávanie sa uskutočňuje porovnávaním signatúr.

1.2.1. Elementárny algoritmus

Algoritmus postupne prikladá a porovnáva hľadané slovo vo všetkých pozíciách textu.

```
int elemnt_search(char *P, char *S)
{
      extern int Text[T];
      extern int Slovo[V];
      int m = 0;
      int I, J =0;
      boolean FIND, OK = false;

      while ((not FIND ) and (I <= T - V )) {
            I ++;
            J= 0;
            OK = true;
            while (OK and (J < V)){
            if (Text[I+J] == Slovo[J+1]){
                  J++;
            }else OK = false;
                  FIND = true;
}
```

1.2.2. Metódy s predspracovaním hľadaného slova

Tieto metódy predtým ako začnú vyhľadávať, predspracujú si hľadané slovo, čím získajú vyhľadávajúci stroj, ktorý je potom použitý pre vyhľadávanie.

1.2.2.1. Metódy s predspracovaním slova so súmerným vyhľadávaním:

1) Knuth-Morris-Prattov algoritmus (KMP)
   - vyhľadávanie jedného slova
2) Aho_Corasickovej algoritmus
   - vyhľadávanie konečnej množiny slov
3) Konečný automat
   - pre vyhľadávanie nekonečnej množiny slov

1.2.2.2. Metódy s predspracovaním slova s protismerným vyhľadávaním:

1) Boyer-Moorevov algoritmus
   - vyhľadávanie jedného slova
2) Commentz-Walterovej algoritmus
   - vyhľadávanie konečnej množiny slov
3) Konečný automat
   - pre reverzovaný regulárny výraz

1.2.3. Metódy s predspracováním textu – indexové metódy

Pred vyhľadávaním predspracováva text, ale nerobí predspracovanie hľadaného slova.

Využíva indexový súbor a index, ktorý je usporiadanou množinou, obsiahnutou v texte. U každého slova v indexe sú uvedené odkazy do textu, ktoré ukazujú, kde sa príslušné slovo nachádza. Indexový súbor obsahuje slová tvoriace index s identifikáciou ich výskytu v texte.

1.2.3.1. Implementácia indexových systémov

Indexový systém sa skladá z:

- **invertovaného indexu**, v ktorom sú uložené informácie o prístupu k jednotlivým dokumentom,
- **sekvenčného súboru**, kde sú uložené vlastné dokumenty.

Možné implementácie:

- použitie **invertovaného súboru**,
- použitie **zoznamu dokumentov** ku každému kľúčovému slovu súradnicový systém s ukazovateľmi na pozíciu v texte.

1.2.3.2. Invertovaný index

Na inverovaný index sa pozrieme dôkladnejšie, keďže tento typ indexu bude zvolený za základ nášho IR systému.

Vrátenie záznamu je založené na určení jeho podobnosti s dotazom a keďže v textovo-založených systémoch sú záznamy neštrukturované a rozhodnutie o vrátení môže závisieť od obsahu týchto záznamov, muselo by sa pri porovnávaní prechádzať celým textom. Väčšinou sú však texty veľmi dlhé a takáto operácia by bola potom príliš nákladná.

Preto prv, než sa skutočne porovnáva, transformujú sa informačné požiadavky do formálnych dotazov a záznamy do indexovej reprezentácie. Obyčajne sú uložené záznamy reprezentované indexovými termínmi, nazývanými indexové vektory. Aby sa rozlíšila dôležitosť jednotlivých termínov pridávajú sa niekedy váhy. Dotazy sú takisto reprezentované váženými alebo neváženými vektormi termínov.

V praktických systémoch sa často navyše využívajú boolovské operátory na určenie vzťahu medzi termínami. Pomocou operátoru *and* môžeme kombinovať termínové frázy, pomocou *or* určiť synonymá a operátor *not* sa obyčajne využíva v kombinácii s *and* ako zužovanie významu určitého termínu. Indexová reprezentácia sa implementuje pomocou invertovaných súborov, t.j. pre každý termín je vytvorený samostatný index s identifikátormi alebo adresami záznamov v ktorých sa nachádza. Invertovanie sa robí nasledovne:

Celý súbor je najprv reprezentovaný ako dvojrozmerné pole indexovaných záznamov, kde každý riadok reprezentuje záznam a každý stĺpec špecifikuje priradenie konkrétneho termínu danému záznamu (tabuľka 1).

Toto pole sa invertuje (transponuje, tabuľka 2).

Riadky invertovaného súboru sa použijú na zisťovanie záznamov relevantných pre daný dotaz.

	Termín 1	Termín 2	Termín 3	Termín 4
Záznam 1	1	1	0	1
Záznam 2	0	1	1	1
Záznam 3	1	0	1	1
Záznam 4	0	0	1	1

tab. 1

	Záznam 1	Záznam 2	Záznam 3	Záznam 4
Termín 1	1	0	1	0
Termín 2	1	1	0	0
Termín 3	0	1	1	1
Termín 4	1	1	1	1

tab. 2

Riadok 2 invertovaného súboru indikuje, že druhý termín je priradený záznamom 1 a 2, ale nie záznamom 3 a 4.

Manipulácia s takýmto súborom je jednoduchá:

- Pre vektorový dotaz, obsahujúci napríklad termíny (T_i, T_j, T_k), sa sčítajú po stĺpcoch i-ty, j-ty a k-ty riadok. Ak sa tento riadok zostupne usporiada, obsahuje záznamy usporiadané podľa relevantnosti.

- Pre boolovský dotaz sa postupuje podobne, napríklad pre operátor and, (T_i and T_j), sa vypíšu všetky záznamy R_k také, že na i-tom riadku a k-tom stĺpci a na j-tom riadku a k-tom stĺpci je v matici invertovaného súboru jednotka.

Výhodou invertovaných súborov je možnosť vyhodnocovať boolovské výrazy len pomocou jednoduchých operácií na jeho riadkoch, nemusí sa pristupovať k hlavným textovým súborom. Hlavná nevýhoda boolovských dotazov spočíva v nemožnosti

usporiadania podľa očakávanej relevantnosti a v nemožnosti rozumne ohraničiť množstvo záznamov vrátených systémov. Formulovanie boolovských dotazov je príliš zložité pre bežných užívateľov.

1.2.3.3. Zoznam dokumentov

Podobne ako pri invertovaných súboroch i tu sú jednotlivé dokumenty očíslované a ku každému kľúčovému slovu je priradený zoznam čísel dokumentov, v ktorých sa toto slovo vyskytuje, viď tabuľku. U tohto zoznamu sa často využíva kódovanie, ktorá spočíva v odvodení *id* dokumentu z *id* dokumentov, ktoré ho predchádzajú v zozname. Napríklad v našej tabuľke sa *slovo3* vyskytuje v dokumentoch *1, 3, 6*

Čísla v druhom stĺpci predstavujú čísla dokumentov1, 2, 3.

Slovo	Dokument
slovo1	1
slovo2	1,2,3
slovo3	2,3
slovo4	1,3

tab 1

1.3. Metódy s predspracovaním textu a hľadaného slova – signaturové metódy

Signatúra je bitový reťazec *Sx*, dĺžky *m,* ktorý je priradený textovému reťazcu *x*. Zobrazenie h: x $\rightarrow$Sx, Sx$\in\{0,1\}$. Každý bit signatúry S vyjadruje nejakú vlastnosť textového reťazca. Pri zvolení vhodného zobrazenia *h*, môžeme porovnávaním signatúry textu St a signatúry hľadaného slova Ss zistiť, či dané slovo je obsiahnuté v texte.

Signatúry jednotlivých termínov môžeme vytvoriť tak, že

- každému termínu priradíme jeden bit v signatúre, alebo
- signatúru termínu vytvoríme pomocou hašovacej funkcie.

Zo signatúr jednotlivých termínov vytvoríme signatúru dokumentov reťazovým alebo vrstveným kódovaním.

1.3.1. Reťazové a vrstvené kódovanie

U **reťazového kódovania** sú signatúry jednotlivých termínov zreťazené.

- záznam: $z = (a_1, a_2, ...a_n)$
- signatúra záznamu: $h(z) = h(a_1) . h(a_2) ... h(a_n)$

Pri **vrstvenom kódovaní** sú signatúry termínov použitých v dokumente logicky sčítané bit po bite.

- v dokumente *d* sú slová: $a_1, a_2, ...a_n$
- $h(d) = h(a_1)$ or $h(a_2)$ or ... or $h(a_n)$

1.3.2. Vyhľadávanie slova pomocou indexu a signatúry

Výsledkom porovnania signatúr je iba zistenie, či text dané slovo obsahuje, ale nedáva nám informáciu o tom, kde sa slovo v texte nachádza.

Preto je dobré kombinovať signatúru s indexom. Pripojiť k signatúre zoznamy ukazovateľov do textového reťazca. Príznakom v signatúre už potom nie je jediný bit, ale ukazovateľ na začiatok reťazca ukazovateľov alebo nič, ak tento reťazec neexistuje v danom dokumente.

2. Návrh informačného systému

Predpokladajme, že máme veľké množstvo dokumentov a chceme byť schopní nájsť dokument, v ktorém sa nachádza slovo, alebo fráza, ktorú práve potrebujeme pochopiť, či použiť v práci. Dosť naivné by bolo začať prehľadávať sekvenčne všetky súbory a pokúsiť sa nájsť hľadanú frázu. Ak by sa jednalo o desiatku dokumentov zdalo by sa nám to zvládnuteľné, ale čo v prípade,že sa jedná o stovky dokumentov, ktoré pri dnešných kapacitách diskov veľakrát už také množstvo obsahujú.

To je ta chvíľa, kedy sa vynorí potreba indexovať dokumenty a pomocou indexu zrýchliť vyhľadávanie. Zvyšovanie kapacít diskov tlačí mnohých užívateľov, hľadať riešenie tohto problému v indexovaní. Stačí len, keď skúsime vyhľadávať zabudnutý email v Outlooku napr. podľa názvu. Pri dostatočnom počte nevymazaných emailoch si nejaké tie sekundy počkáme. Omnoho lepšie na tom budeme keď na svoju poštu použijeme emailového klienta zabudovaného v prehliadači Opera, kde sa ukladajú emaily, záložky do indexov a zadaním hľadaného výrazu sa výsledok zobrazí okamžite.

V tejto kapitole bakaláskej práce pojednávame o návrhu informačného systému (tzv. IR), ktorého úlohou bude vyhľadanie hľadaného výrazu a identifikácia blízkeho okolia v dokumetoch jeho výskytu.

Východiskom pri návrhu IR systému bol volne šíriteľný (open source) project Lucene. Ide o vysoko výkonnú, dostupnú IR knižnicu, ktorá umožňuje vývojárovi využiť v svojej aplikácií jej schopnosti rýchlo a účinne indexovať a vyhľadávať v zaindexovaných textových dokumentoch hľadaný výraz. *Lucene* je členom rodiny Apache Jakarta projektov, ktoré sú šírené voľne pod Apache Software licenciou.

V podkapitolách sa oboznámime s vytváranim a využívanim invertovaného Lucene indexu. Najprv si rozoberieme Lucene index z logického pohľadu. Ten nám priblíži ako funguje základná štruktúra Lucene indexu pri jeho napĺňaní dokumentami a vyhľadávaní v

ňom V ďalšej podkapitole sa pozrieme na *Lucene - invertovaný index*, ktorý bude použitý pre indexáciu dokumentov.

Potom bude následovať *indexácia dokumentov,* kde sa bližšie zoznámime s jednotlivými fázami indexovania dokumentov. Tento postup – indexovanie nám dáva výhodu pri vyhľadávaní, pretože umožňuje zrýchliť prístup k informáciam, keďže nemusíme pri každom vyhľadávaní otvárať nespočetné množstvo dokumentov.

Stačí len pristupovať k indexu, ktorý je veľkosťou menší pri dobre zvolenej kompresii dát, než celkové množstvo dokumentov a zároveň obsahuje informácie o presnej pozícii hľadaného výrazu v dokumetoch, čím sa obmedzí doba prehľadávania celého textu v dokumente.

Poslednou časťou tejto kapitoly bude *vyhľadávanie v texte,* očakávaná funkčnosť IR systému, a to schopnosť získavať informácie z veľkého množstva dokumentov na základe požadovaného výrazu v čo najrýchlejšom čase z predpripraveného spracovania dokumentov v indexe, ktorý sme vytvorili na začiatku.

2.1. Návrh logickej štruktúry indexu s využitím Lucene

Pohľad na Lucene index môžeme znázorniť následovne:

Index → Segmenty → Lucene Dokumenty → Fields

Index sa skladá z niekoľko na sebe nezávislých *segmentov*, z ktorých každý obsahuje presne daný počet *Lucene Dokumentov*, tie zas predstavujú v RAM pamäti do určitej miery obraz textových dokumentov, ktoré sú indexované. Pred tým, než sa nejaký textový dokument zaindexuje, je potrebné použiť analýzu, pomocou ktorej zistíme ktoré dáta dokumentu korešpondujú s ktorým typom *fieldu* a zároveň extrahujeme stopové termy, ktoré sú pre vyhľadávanie relevantné. *Lucene Dokument* sa skladá zo sekvencie *fieldov*, ktoré predstavujú dáta dokumentu.

Kvôli môžnému zahmleniu významu, prípadne menšej zrozumiteľnosti sme ponechali pôvodný pojem *field*, ktorý si vysvetlíme neskoršie.

2.1.1. Fieldy

Každý *Lucene Dokument* (viac kapitola 2.1.2) obsahuje jeden alebo viacero typov fieldov. Každý *Lucene Dokument* (viac kapitola 2.1.2) obsahuje jeden alebo viacero typov *fieldov*. Každý field odpovedá kúsku dát (slová, emaily, čísla ...) z *Lucene Dokumentu*. To ako sa z textu dokumentu transformujú jednotlivé slová, znaky, emaily, či iné entity prítomné v texte dokumentu do *fieldov*, tak to záleží jednak na nastavení typu fieldu (pozri kapitola 2.3.1.5) a na analýze textu *Analyzátorom* (viac v kapitole 2.3.1.3).

Každý typ *fieldu*, ktorý ponúka Lucene sa ďalej delí na:

- neinvertovateľné – nie sú prevedené do invertovaného formátu, ale sa ukladajú do indexu doslovne (napr. url, telefónne číslo),
- invertovateľné – všetky tie, ktoré je možné uložit ako invertované.

Oba druhy fieldov sa uložené do indexu, prvé ako neinvertované a druhé ako invertované.

Typ fieldu	Analyzovaný	Indexovaný	Uložený	Použitie
Keyword		áno	áno	Telefónne číslo, URL, osobné mená
UnIndexed			áno	PDF, HTML, ak nie sú potrebné pre vyhľadávanie
UnStored	áno	áno	áno	Názvy a obsah dokumentov
Text	áno	áno	áno	Názvy a obsah dokumentov

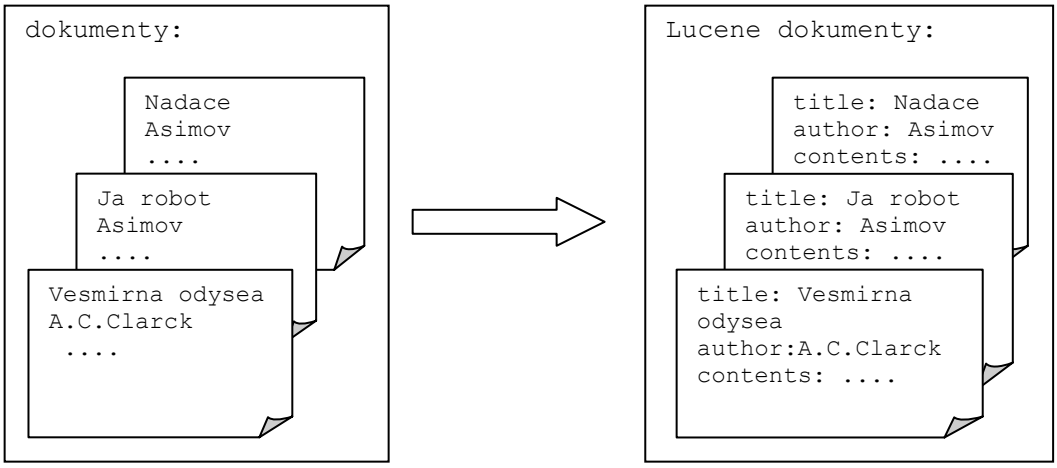
tab. 2.1.1

2.1.2. Lucene Dokument

Keď indexujeme, vytvárame inštanciu triedy *Lucene Dokumentu.* Rozlišujeme slovo *dokument* od *Lucene Dokument*, prvé slovo vyjadruje textový dokument typu DOC, RTF,

PDF, atď., kdežto Lucene Dokument predstavuje v pamäti RAM analyzovaný obsah textového dokumentu vo forme čistého textu, ktorý chystáme zaindexovať.

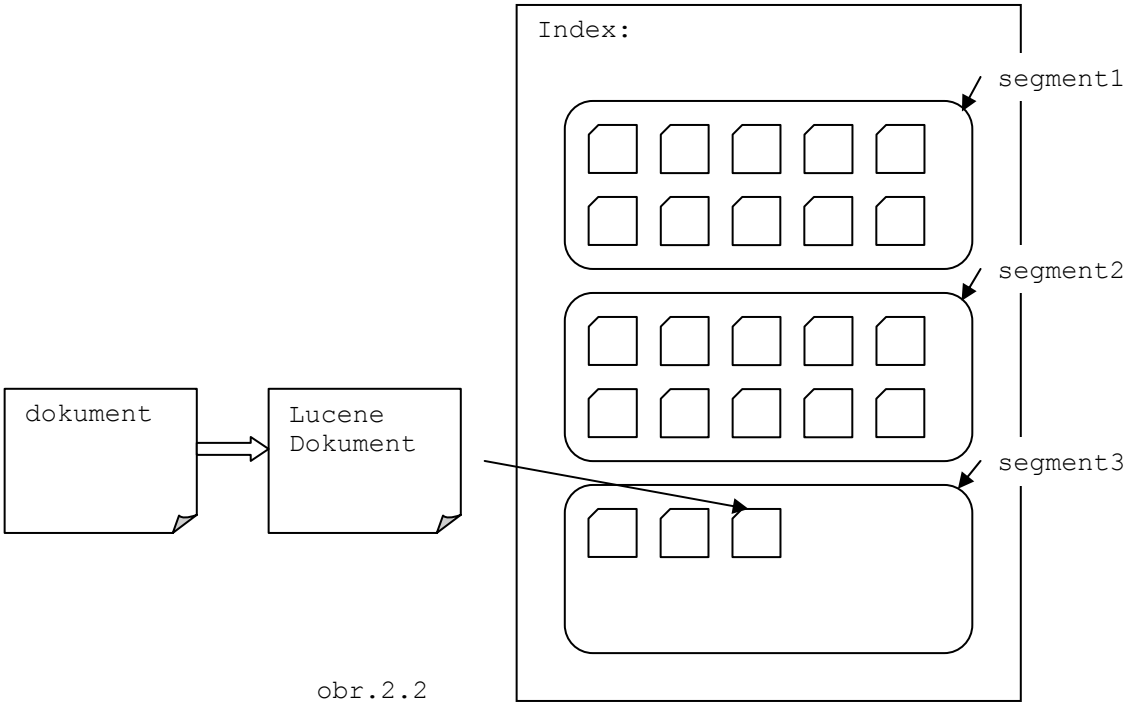
Po vytvorení triedy Lucene Document ju naplníme fieldami, ktoré obsahujú meno fieldu a jeho textovú hodnotu termu. Ako vidno z obrázka obr.2.1 Lucene dokument sa naplní podľa definovaných fieldov, ktorým sa priradí im prináležiaca hodnota. Podľa obr.2.1 sa v Lucene Dokumente pre 1. textový súbor do fieldu *title* priradí názov daného dokumentu: Nadace a podobne sa naplia ostatné fieldy podľa charakteru informácie, ktorú obsahujú.



obr.2.1

2.1.3. Segment

Výsledný Lucene dokument uložený v RAM pamäti sa stane súčasťou otvoreného segmentu. V situácií keď už existuje segment, ktorý ešte nie je úplne zaplnený, sa daný nový Lucene Dokument pridá k ostaným Lucene Dokumentom prítomným v segmente. Každý segment obsahuje kolekciu Lucene dokumentov, ich počet závisí od parametra, ktorý je možno nastaviť podľa potreby. Defaultne nastavená hodnota je 10 Lucene Dokumentov na 1 segment.



obr.2.2

Pridanie Lucene Dokumentu k ostatným v segmente je proces samotnej indexácie.

Použitie segmentov nám umožňuje rýchlo pridať nový Lucene Dokument do indexu a to tak, že ho pridáme do vytvoreného segmentu, ktorý po naplnení periodicky zlučujeme s ostatnými exitujúcimi segmentami v indexe. Tento proces zabezpečí dostatočnú výkonnosť, pretože minimalizuje fyzickú modifikáciu indexu, zmenšuje počet prístupov na disk.

Po zaindexovaní sa Lucene Dokument stane súčasťou segmentu. Tento segment nesie so sebou informácie o indexovaných Lucene Dokumentoch v indexových súboroch

(bližšie o nich v kapitole 2.2), ktoré uchovávajú záznamy o indexovaných dokumentoch, fieldoch, termoch, ďalej výskyt a pozíciu termov v dokumente a sú vzájomne poprepájané odkazmi na záznamy, ktoré obsahujú.

2.1.4. Index

Zaplnený zaindexovaný segment sa pridá do poradia segmentov, ktoré sú súčasťou indexu. Takže na index sa môžeme pozerať ako na sekvenciu segmentov, ktoré obsahujú informácie o zaindexovaných Lucene Dokumentoch (pozri obr.2.2).

2.1.5. Dve možné kompozície Lucene indexu

Lucene podporuje dve štruktúry indexu:

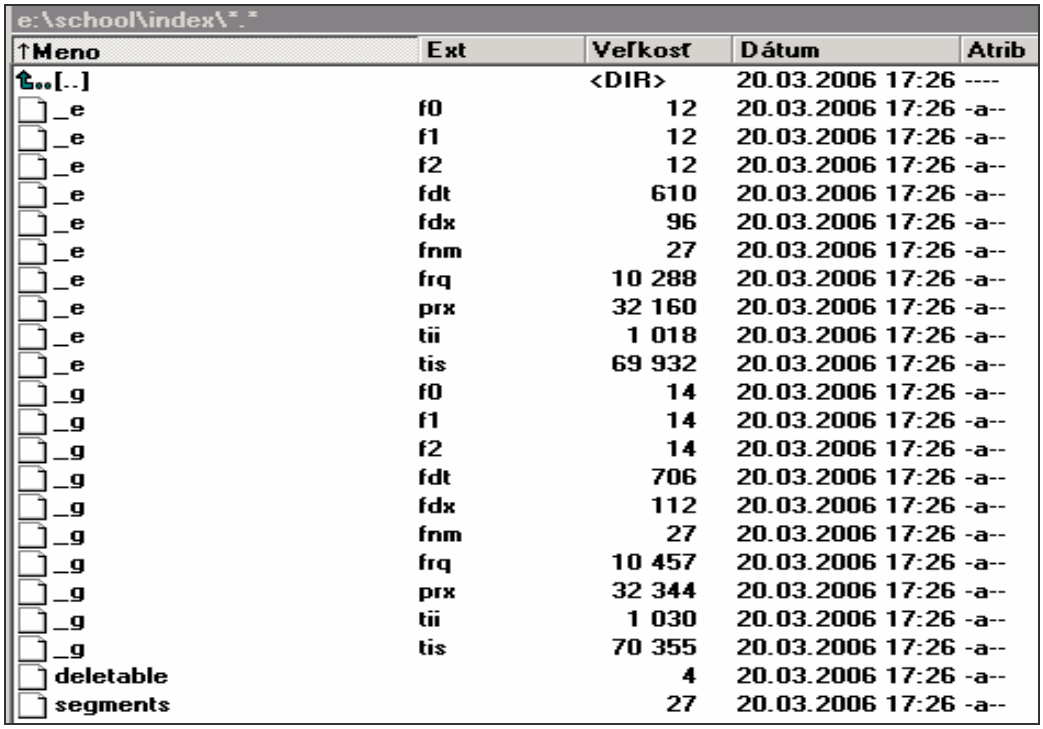
- mnohosúborové (multifile) indexy
- zlúčené (compound) indexy

Rozdiel medzi nimi je v štruktúre. Každý index sa sladá z viacerých segmetov, ktoré sú charakterizované 7 indexovými súbormi. Oproti mnohosúborovej štrukútre indexu, kde sa s indexovými súbormi pre všetky segmenty nerobí žiadna úprava, sa pri zlúčenej štruktúre jednotlivé indexové súbory zlúčia do jedného indexového súboru.

2.1.5.1. Mnohosúborová štruktúra indexu

Lucene index pozostáva z jedného alebo viacerých segmentov a každý segment je tvorený príslušnými indexovými súbormi. Každý indexový súbor zachytáva prestný typ informácie nevyhnutný pre Lucene index. V prípade modifikácie alebo odstránenia iným systémom, než je postavený na Lucene API môže dôjsť k nekompaktibilite indexu a jedinou opravou je reindexácia z originálnych súborov. Na druhej strane, je možné pridanie

nových dokumentov pomocou Lucene do indexu bez nejakého narušenia konzistencie indexu.



e:\school\index*.*

↑Meno	Ext	Veľkosť	Dátum	Atrib
[..]		<DIR>	20.03.2006 17:26	----
_e	f0	12	20.03.2006 17:26	-a--
_e	f1	12	20.03.2006 17:26	-a--
_e	f2	12	20.03.2006 17:26	-a--
_e	fdt	610	20.03.2006 17:26	-a--
_e	fdx	96	20.03.2006 17:26	-a--
_e	fnm	27	20.03.2006 17:26	-a--
_e	frq	10 288	20.03.2006 17:26	-a--
_e	prx	32 160	20.03.2006 17:26	-a--
_e	tii	1 018	20.03.2006 17:26	-a--
_e	tis	69 932	20.03.2006 17:26	-a--
_g	f0	14	20.03.2006 17:26	-a--
_g	f1	14	20.03.2006 17:26	-a--
_g	f2	14	20.03.2006 17:26	-a--
_g	fdt	706	20.03.2006 17:26	-a--
_g	fdx	112	20.03.2006 17:26	-a--
_g	fnm	27	20.03.2006 17:26	-a--
_g	frq	10 457	20.03.2006 17:26	-a--
_g	prx	32 344	20.03.2006 17:26	-a--
_g	tii	1 030	20.03.2006 17:26	-a--
_g	tis	70 355	20.03.2006 17:26	-a--
deletable		4	20.03.2006 17:26	-a--
segments		27	20.03.2006 17:26	-a--

obr.2.3

Indexové súbory, ktoré patria do to istého segmentu zdieľajú spoločné pomenovanie ale rozdielnu príponu. Ako je vidno z obr.2.3, máme index, ktorý má dva segmenty s pomenovaním *_e.** a *_g.**, ale ma rodielne prípony pre indexové súbory (**.fdt, *.fdx, *.tis* ... viac o nich v kapitole 2.2). Jedine súbory *segment* a *deletable* nemajú príponu.

V prvom z nich, v súbore *segment* sú uložené mená všetkých existujúcich segmentov. Pred prístupom k nejakému zaindexovanému dokumentu v indexe, sa Lucene poradí so súborom *segment*, ktoré indexové súbory je potrebné otvoriť a prečítať, pretože vie, že pre každý segment uložený v indexe existujú indexové súbory s rovnakým menom a odlišnou príponou, pomocou ktorých môže pristupovať náhodne do indexovaného súboru.

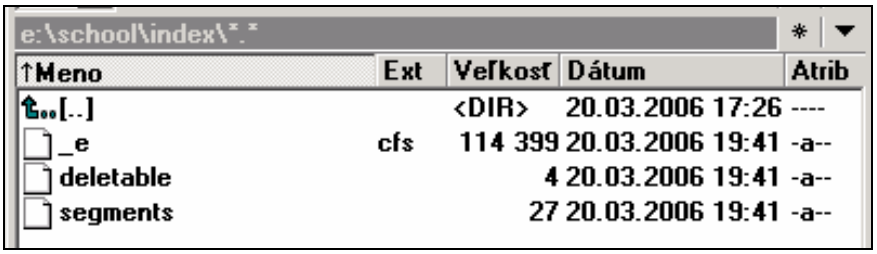
Súbor "deletable" obsahuje mená súborov, ktoré nie sú dlhšiu dobu využívané, ale ktorý nemôže byť vymazaný. Toto je potrebné pre Win32m, kde súbor nesmie byť vymazaný pokiaľ je ešte otvorený. Na iných platformách súbor obsahuje nulový bajt.

2.1.5.2. Štruktúra zlúčených index súborov

Keď sme písali o mnohosúborovej štruktúre indexov, povedali sme, že počet indexových súborov závisí na počtu indexovaných fieldov prítomných v indexe. Taktiež sme zmienili, že sú vytvárané nové segmenty, keď pridávame dokumenty do indexu. Pretože segment pozostáva z množiny indexových súborov, výsledkom môže byť veľký počet indexových súborov v adresári indexu. Preto sa objavuje otázka, či práca s tak veľkým množstvom súborom nezefektívni prácu Operačného Systému (OS)?

Pri dostatočne veľkom počte indexových súborov to môže byť problém, pretože OS povoľuje len určitý počet otvorených súborov v jednom okamihu. V prípade mnohosúborovej štruktúry indexu by pri vačšom počte indexov mohol počet otvorených indexových súborov prekročit maximálne povolené množstvo otvorených súborov. Preto Lucene má defaultne nastavenú štruktúru zlúčených index súborov.

Zlúčený index pozostáva z jedného súboru s príponou *.*cfs* (pozri obr.2.4), v ktorom sú zlúčené všetky index súbory.



obr.2.4

Namiesto otvorených 10 súborov u mnohosúborového indexu, Lucene musí pri prístupe k zlúčenému indexu otvoriť iba dva súbory: *e.cfs* a *segments* (Súbor *deletable* nie je potrebné čítať počas indexovania a vyhľadávania), čím spotrebováva menej systémových zdrojov. Zlúčený index redukuje počet indexových súborov, ale koncept segmentov, dokumentov, fieldov a termov stále zostáva, ako to bolo u mnohosúborového indexu. Rozdiel je v tom, že *zlúčený index* obsahuje jeden *.*cfs* súbor na segment, zatiaľ čo každý segment v *mnohosúborovom indexe* obsahuje sedem medzi sebou prepojených indexových

súborov. Výsledný zlúčený index zastrešuje jednotlivé indexové súbory do výsledného jediného *.*cfs* súboru (v našom prípade ide o súbor *_e.cfs*) .

2.2. Voľba invertovaného indexu

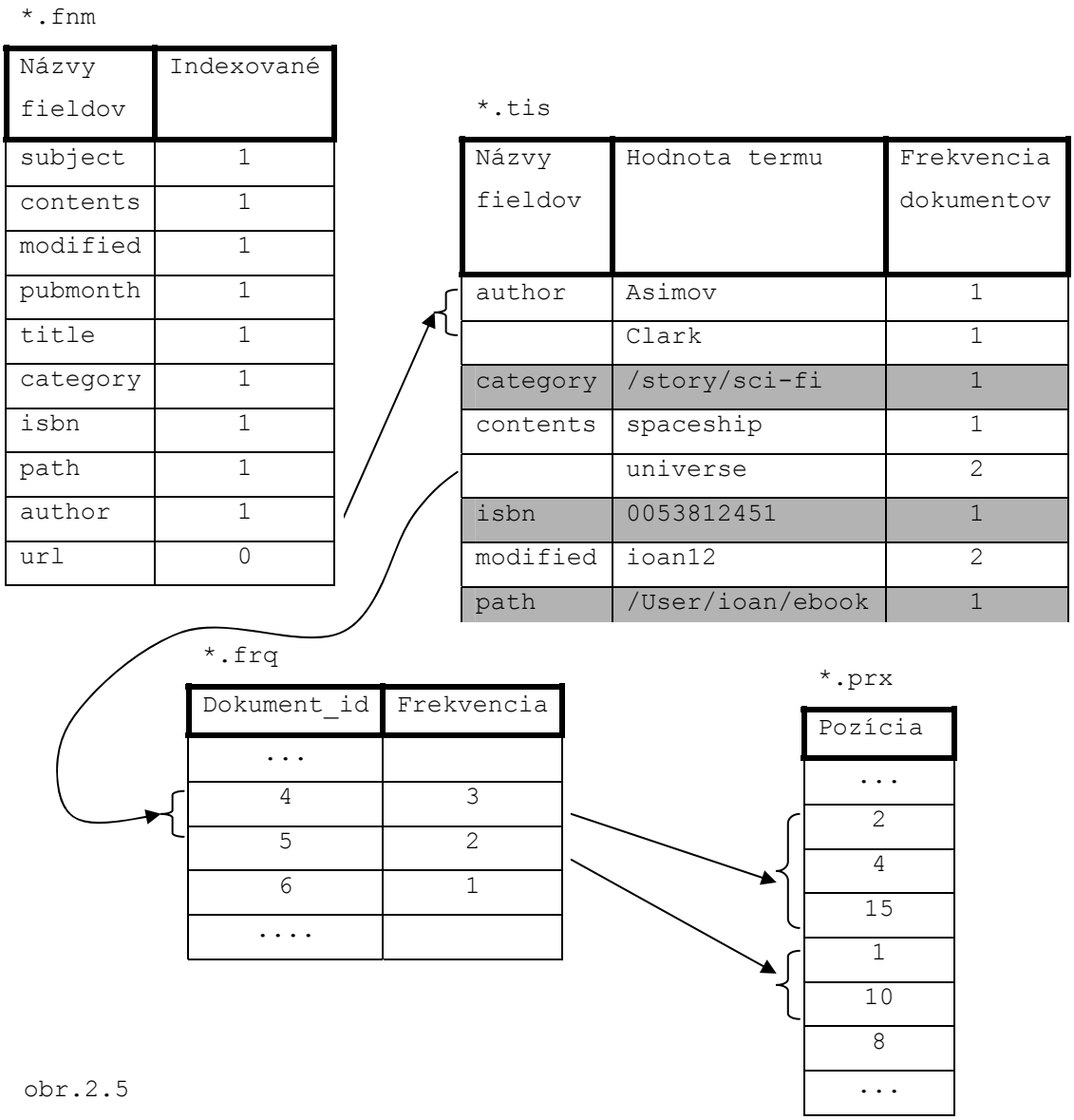
Možnosť použitia boolovských výrazov pomocou jednoduchých operácií na riadkoch invertovaných súborov je určite jedným z dôvodov prečo Lucene index používa invertovaný index.

Lucene index patrí do rodiny invertovaných indexov, ktoré boli spomínané v 1. kapitole u indexovaných metód. Pre pripomenutie ide o invertovanú indexovú štruktúru mapujúcu umiestnenie slov v dokumente alebo v sade dokumentov, poskytujúcu fulltext vyhladávanie. Súbor s invertovaným indexom obsahuje pre každé slovo zoznam odkazov na všetky dokumenty, v ktorých sa vyskytuje. Plne invertovaný index dodatočne obsahuje informáciu o tom, kde v dokumente sa slova vyskytujú.

Lucene indexu pozostáva z niekoľkých segmentov a každý z nich je tvorený indexovými súbormi (**.fnm, *.tis, *.frq, *.prx,* ...), ktoré sú poprepájané medzi sebou odkazmi. Časť z týchto súborov znázorňuje obr.2.5**.**

2.2.1. Názvy fieldov:

Názvy fieldov (*.fnm) **-** súbor s príponou **.fnm (field name)* obsahuje mená fieldov, ktoré sú použíté v dokumente. Každý field je označený, či je indexovaný, v tom prípade je najnižší bit jednička, v prípade, že field nie je indexovaný, vtedy je najnižší bit rovný nule a je iba uložený do indexu. Poradie mien fieldov je určené počas indexovania a nie je nevyhnutne abecetne zoradené. Pozícia fieldu v súbore je použítá k prepojeniu s normalizačným súborom (**.f[0..9]*)



obr.2.5

Uloženie fieldov (****.fdx a *.fdt***)

Uloženie údajov o fieldoch reprezentujú dva súbory. Prvý súbor ****.fdx (field index)*** obsahuje pre každý dokument ukazovateľ k jeho dátam. Je používaný k nájdeniu polohy vo vnútri súboru fieldových dát (**.fdt field data files*) z jednotlivého dokumentu. Pretože obsahuje pevnú dĺžku údajov, je preto možné využiť náhodný prístup. Druhý súbor ****.fdt (field data)*** obsahuje uložené fieldy z každého dokumentu.

2.2.2. Slovník termov

Slovník termov je reprezentovaný dvoma súbormi (**.tis a *.tii*).
Term predstavuje základnú jednotku v texte dokumentu.

Informácie o Terme ****.tis (term infos)***, kde sa ukladajú všetky indexovateľné termy v segmente. Termy sú zoradené abecedne najprv podľa mien fieldov a až potom podľa hodnoty fieldu. Každý term obsahuje frekvenciu jeho výskytu v dokumentoch, čiže počet dokumentov obsahujúci daný term a to iba pre daný segment. V prípade, že máme definovaný field, ktorý sa neindexuje napr. url, tak tento typ fieldu nie je prítomný, uložený v tomto súbore. Pre každý term uložený v **.tis* súbore, obsahuje súbor **.frq* odkazy na každý dokument, ktorý obsahuje daný term. Podľa obr.2.5 term *universe* má frekvenciu výskytov v dvoch dokumentoch, preto sa odkazuje v súbore **.frq* na dva dokumenty: dokument s *id 4* a dokument s *id 5*.

Súbor ****.tii (term info index)*** má podobnú štruktúru ako súbor **.tis*. Obsahuje všetky atribúty súboru **.tis* a naviac ešte pozície kde sa dané atribúty v súbore **.tis* presne nachádzajú. Je konštruovaný tak, aby bol prítomný vo fyzickej pamäti RAM, aby bol umožnený rýchly prístup k záznamom termu uložených v súbore **.tis*.

2.2.3. Frekvencia výskytu termov

Početnosť výskytu termov v každom dokumente je zaznamenaná v súbore ****.frq***, kde pre každý dokument je uvedený počet výskytov termu v danom dokumente. Pre každý dokument zaznamenaný v súbore **.frq* sú v súbore **.prx* (*term position*) uložené záznamy o presnej pozícií jednotlivých termov v danom dokumente. Podľa obr.2.5 sa v našom prípade term "universe" vyskytuje v dvoch dokumentoch. V prvom dokumente s *id 4* sa nachádza tri-krát a to na pozícií 2, potom sa znovu objavuje ako 4-ty term v dokumente a

nakoniec sa ešte objavuje ako 15-ty v poradí termov. V druhom dokumente s *id 5* sa vyskytuje iba dva-krát a to na pozícii 1 a 10 v texte dokumentu.

2.2.4. Pozícia Termov

Súbor ***.prx** ako už bolo priblížené v príklade predchadzajúcej kapitoly uchováva záznam o pozícií daného termu v dokumente. Informácia o pozícii sa využíva pri dotazoch ako sú phrase queries a span queries.

2.3. Indexujeme dokumenty do indexu

V tejto kapitole rozoberieme postupne všetky kroky potrebné k indexovaniu dokumentov, k tomu využíva Lucene svoje základné triedy, ktoré si v stručnosti rozoberieme v následujúcej kapitole.

2.3.1. Základné triedy pre indexovanie

Pre indexovanie dokumentov používa Lucene týchto 5 hlavných tried:

- *IndexWriter*
- *Directory*
- *Analyzer*
- *Document*
- *Field*

V následujúci podkapitolách si bližšie vysvetlíme, čo znamenajú a akú úlohu zohrávajú pri vytváraní Lucene indexu.

2.3.1.1. IndexWriter

IndexWriter je ústrednou komponentou indexovacieho procesu. Táto trieda vytvára nový index a pridáva ďalšie dokumenty do existujúceho idexu. Môžeme sa pozerať naň ako na objekt, ktorý má práva zápisu a prístup k indexu, umožňuje modifikáciu indexu, ale nedáva nám možnosť čítania a vyhľadávania v indexe.

2.3.1.2. Directory

Trieda *Directory* reprezentuje umiestnenie Lucene indexu. Ide o abstraktnú triedu, má dve implementácie *FSDirectory* (File System Directory) a *RAMDirectory*. Prvá *FSDirectory* sa používa v prípade, že chceme zapisovať Lucene index na pevný disk. Druhou implementáciou triedy *Directory* je *RAMDirectory*, ktorá ako už sám názov naznačuje sa používa, keď je potrebný rýchly prístup k indexu. Jeho využitie je obmedzené veľkosťou indexu, aby ho bolo možné celý nahrať do operačnej pamäte RAM. Samozrejme výkonostný rozdiel medzi oboma variantami je menej viditeľný, keď je Lucene používaný na OS, ktorý ukladá súbory do vyrovnávacej pamäte. Jediný rozdiel medzi nimi je v perzistentnosti indexu, ktorý sa u *RAMDirectory* po ukočení práce stráca.

2.3.1.3. Analyzer

Pred vytvorením indexu je potrebné text dokumentu analyzovať. Na to sa používa *Analyzátor,* ktorý má dozor nad extrahovaním tokenov z textu, ktoré budú indexované a ktoré budú vyňaté. Ak obsah, ktorý sa má indexovať nie je čistý text, je potrebné ho najprv naň konvertovať a až následne je možná indexácia. Lucene ponúka štyri druhy analyzátorov:

Analyzátor	Funkcia
WhitespaceAnalyzer	vynecháva prázdne miesta
SimpleAnalyzer	oddelí od textu neabecedné znaky a upraví všetky písmená na malé.
StopAnalyzer	oddelí od textu neabecedné znaky, upraví všetky písmená na malé a odstráni stopové slová
StandardAnalyzer	tokenizácia založená na sofistikovanej gramatike, ktorá rozpozná e-mailové adresy, skratky, Čínsko-Japonské znaky, taktiež odstráni stopové slová a zmení písmená na malé

tab. 2.3.1

2.3.1.4. Document

Presnejšie ide o *Lucene Dokument*, o ktorom sme sa už zmienili v kapitole 2.1.2. Ide o akýsi virtuálny dokument, ktorý reprezentuje dáta v skutočnom indexovanom dokumente. *Dokument* obsahuje fieldy, ktoré predstavujú časti textového dokumentu (slovo, e-mailovú sdresu, ...) alebo metadáta spojené s dokumentom. Iba metadáta typu: autor, titul, dátum sú revelantné pre Lucene, tie sú potom indexované a uložené separátne v svojich fieldoch.

2.3.1.5. Field

Lucene ponúka štyri typy fieldov:

Keyword – nie je analyzovaný, ale je indexovaný a uložený v indexe doslovne. Tento typ je vhodný pre fieldy, ktorých pravá hodnota by mohla byť bez zmien zachovaná. Jedná sa napr. o URL, systémovú cestu súboru, dátum, číslo kreditnej karty, atď.

UnIndexed – neanalyzované, neindexované, ale ich hodnota je uložená v indexe bez zmien. Tento typ je vhodný pre fieldy, ktoré potrebujeme zobraziť s výsledkom vyhľadávania (napr. URL, primárny kľúč z databázy, ...), ale ktorých hodnotu nikdy nebudeme vyhľadávať priamo dotazom. Pretože pôvodna hodnota tohto typu fieldu sa ukladá do indexu, nie je vhodný pre field s veľkou hodnotou.

UnStored – je opakom *UnIndexed.* Tento typ je analyzovaný a indexovaný ale nie je uložený v indexe. Je vhodný pre indexovanie väčšieho množstva textu, ktorý nie je potrebné získavať v jeho originálnej forme.

Text – je analyzovaný, indexovaný a zároveň i uložený. To naznačuje, že fieldy tohto typu môžu byť vyhľadávané z indexu.

Všetky fieldy pozostávaju z páru: meno a hodnota. Vyhľadávanie v indexe potom môžeme filtrovať podľa daného typu *fieldu*.

Pohľad na rozdielnosť typov fieldov ich charakteristík a použitia je znázornený v tabuľke 2.1.1:

Typ fieldu	Analyzovaný	Indexovaný	Uložený	Použitie
Keyword		áno	áno	Telefónne číslo, URL, osobné mená
UnIndexed			áno	PDF, HTML, ak nie sú potrebné pre vyhľadávanie
UnStored	áno	áno	áno	Názvy a obsah dokumentov
Text	áno	áno	áno	Názvy a obsah dokumentov

tab. 2.3.2

2.3.2. Analýza textu

Analýza u Lucene, je proces prevodu textu na jeho najzákladnejšiu indexovú reprezentáciu, *termy.*

Aby sme mohli indexovať dáta, musíme najprv z dokumentov vyextrahovať čistý text vo formáte, ktorý je schopný Lucene spracovať. Akonáhle máme pripravený čistý text pre indexáciu a vytvorený Lucene Dokument naplnený fieldami, môžme začať indexovať.

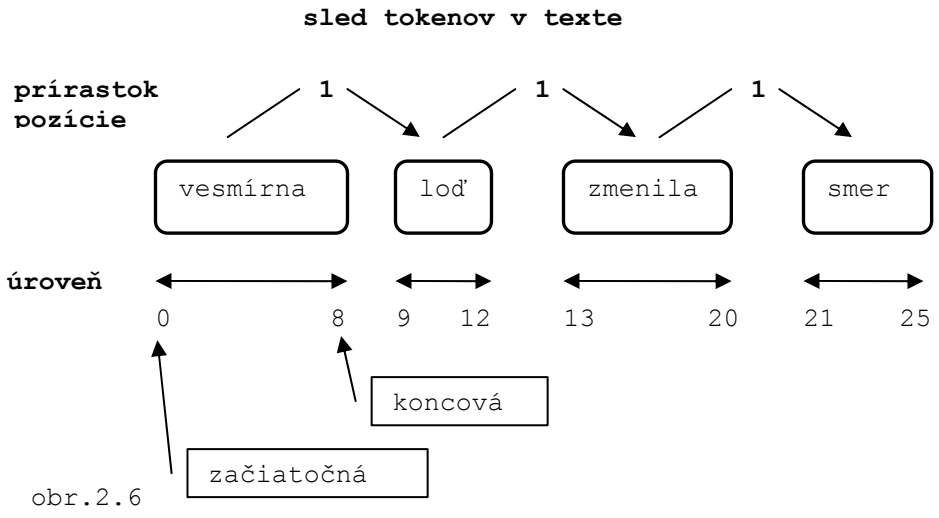
Pri analýze Lucene najprv zistí vhodnosť dát, ktoré sa budú indexovať. Aby sa to mohlo urobiť, je potrebné extrahovať slová z textu a vykonať niekoľko operácií na nich. Je žiadúce odstrániť všetky často frekventované bezvýznamné slová, znaky z textu, ako sú stopové slová (napr. *pre anglické texty: a, an, the, in, on* ....) a taktiež pre vyhľadávanie irevelantné interpunkčné znamienka.

Hlavným cieľom Lucene je umožniť získavanie informácií. Keďže dôraz je kladený na vyhľadávanie slov v texte, je nutné preto vedieť ktoré slová to sú. Ktoré sú dôležité pre vyhľadávanie a analýzou textu počas indexovania, extrahovať ich do termov. Tieto termy sú základnými stavebnými blokmi pri vyhľadávaní.

Lucene ponúka štyri typy analýzatorov, ktoré sme si priblížili v kapitole 2.3.1.3.

Hlavným výstupom procesu analýzy je sled tokenov. Počas indexovania, fieldy určené na tokenizáciu sú spracované špecifickým analyzátorom a každý token je zapísaný do indexu ako term. Tento rozdiel medzi tokenmi a termami sa môže na prvý pohľad zdať nejasný. Objasníme si to na príklade:

Každý token reprezentuje jednotlivé slovo v texte. Token nesie so sebou hodnotu slova a taktiež nejaké metadáta: začiatočnú a koncovú úroveň v texte, typ znaku a prírastok pozície (pozri obr.2.6). Začiatočná úroveň je poloha znaku v originálnom texte, kde text tokenu začína a koncová úroveň je poloha hneď po poslednom znaku textu tokenu. Keď je text tokenizovaný, pozícia tokenu sa vždy určuje relatívne vzhľadom na predchádzajúci token a je zaznamenaná hodnotou prírastku pozície.



obr.2.6

Po analýze textu počas indexovania, každý token je uložený do indexu ako term. Prírastok pozície je jedinou meta-hodnotou pripojenou k tokenu, ktorá sa prenáša spolu s

ním do indexu. Začiatočná a koncová úroveň sa vyhodí, pretože sa používa iba počas analýzy.

2.3.3. Vytváranie indexu – Indexwriter

Potom čo vstupné dáta boli analyzované, môžu sa uložiť do indexu. Lucene ukladá analyzované dáta do štruktúry známej pod menom *invertovaný index* (pozri kapitolu 2.2). Táto štruktúra umožňuje účinné využitie miesta na disku, čím dovoľuje rýchle vyhľadávanie. Invertovaný index používa extrahované termy z vstupných dokumentov ako vyhľadávacie kľúče namiesto práce priamo s dokumentami. Inými slovami, namiesto toho, aby sme sa pokúsili získať odpoveď na otázku "ktoré slová obsahuje daný dokument?", táto štruktúra odpovedá na otázku "ktoré dokumenty obsahujú dané slovo?".

Lucene vytvára index pomocou objektu triedy *IndexWriter*. Konštrukciou tohto objektu sa predávajú parametre adresár, analyzátor, boolovska hodnota.

```
Analyzer analyzer = new StandardAnalyzer();
IndexWriter writer = new IndexWriter(index_dir, analyer, true);
```

Adresár predstavuje miesto, kde sa uloží vytvorený index, presnejšie indexové súbory. Predspracovanie textu zabezpečuje analyzátor (pozri kapitolu 2.3.1.3); v našom príklade ide o *StandardAnalyzer*. Posledným parametrom je boolovska hodnota, ktorá pri nastavení na hodnotu *true*, indikuje vytvorenie nového indexu.

Následujúci kód demoštruje indexáciu dokumentu s fieldom typu *contents*:

```
Document doc = new Document();
doc.add(Field.Text("contents"; "Nadace je epická science
                                                fiction ...));
doc.add(Field.Keyword("isbn"; 80-204-0931-9");
writer.addDocument(doc);
```

Pomocou inštancie *analyzer* predanej *IndexWriter* sa analyzuje iba field *contents*, no do indexu sa uložia oba *contents* i *isbn*. V prípade, že nejaký dokument potrebuje špeciálnu analýzu, analýzer môže byť špecifikovaný pre daný dokument:

```
writer.addDocument(doc, analyzer);
```

Field.Keyword indexovaný field nie je analyzovaný. Field *isbn* je indexovaný ako jednoduchý term s hodnoutou presne takou, akú obsahuje. Keď je dokument zaindexovaný, tak nie je rozdiel medzi termom *isbn* a termom *contents,* ktorý bol naviac ešte analyzovaný.

2.3.4. Optimalizácia vytvoreného indexu

Optimalizácia indexu je proces, kedy zlučujeme mnohonásobné indexové súbory spolu v snahe redukovť ich počet a tak minimalizovať čas potrebný na čítanie indexu pri vyhľadávaní.

Pri optimalizácií indexu splynú všetky indexové segmenty do jedného. Táto optimalizácia prebieha pri volaní metódy `optimize()` na objekte `IndexWriter`.

Pričom tento proces optimalizácie indexu nemá vplyv na rýchlosť indexovania, ale iba na rýchlosť vyhľadávania v indexe. Prínosom vo vyhľadávaní je fakt, že s optimalizovaným indexom Lucene potrebuje otvoriť a spracovať menej súborov ako pri neoptimalizovanom indexe.

2.4. Vyhľadávanie

Vyhľadávanie využitím Lucene API je sofistikovaný mechanizmus, ktorý vyhovuje vyhľadávacím potrebám, ako je vrátenie najviac relevantných dokumentov zoradených podľa ich relevantnosti a dodanie výsledku neuveriteľne rýchlo. Táto kapitola pokrýva obvyklý spôsob vyhľadávania v texte použitím Lucene.

2.4.1. Základne triedy pre vyhľadávanie

Základné vyhľadávacie prostedie, ktoré Lucene poskytuje je nenáročné podobne ako pre indexovanie. Iba niekoľko tried je potrebných pri vykonávaní základných vyhľadávacích operácií:

- *IndexSearcher*
- *Term*
- *Query*
- *TermQuery*
- *Hits*

V Nasledujúcich podkapitolách si ozrejmíme ich použitie a význam pri vyhľadávaní použitím Lucene API, ktorej sú súčasťou.

2.4.1.1. IndexSearcher

IndexSearcher vyhľadáva to, čo *IndexWriter* indexuje. Môžeme si ho predstaviť ako triedu, ktorá otvára index v móde určenom iba na čítanie (read only mode).

Ponúka niekoľko vyhľadávacích metód, niektoré z nich sú implementáciou abstraktnej rodičovskej triedy *Searcher*, najjednoduchšia z nich zoberie jednoduchý *Query* objekt ako parameter a vráti výsledky vyhľadávania ako *Hit* objekt. Typický príklad takejto metódy vyzerá nasledovne:

```
IndexSearcher isearcher = new IndexSearcher (
                                FSDirectory.get("index",false));
Query query = new TermQuery (new Term("contents", "vesmir"));
Hits hits = isearcher.search(q);
```

2.4.1.2. Term

Term je najmenšou základnou časťou indexu, tým pádom aj najzákladnejšou jednotkou vyhľadávania. Podobne ako Fieldový objekt pozostáva z dvojice elementov a to z mena fieldu a hodnoty daného fieldu. Objekty Termov sú tiež zahrnuté do indexovacieho procesu. Sú tvorené interne v Lucene, takže nemusíte o nich rozmýšľať počas indexovania. Počas vyhľadávania, môžeme vytvoriť objekty Term a použiť ich spolu s TermQuery(pozri kapitola 2.4.1.4) :

```
Query query = new TermQuery(new Term("contents", "vesmir"));
Hits hits = isearcher.search(q);
```

Tento kód požaduje od Lucene nájsť všetky dokumenty, ktoré obsahujú slovo *vesmir* vo fiede nazvanom *contents.* TermQuery objekt je odvodený z abstraktnej rodičovskej triedy Query, preto ho možeme priradiť objektu typu *Query*.

2.4.1.3. Query

Lucene prichádza s niekoľkými konkretnými podtriedami *Query*. Zatiaľ sme sa zaoberali a uvádzali sme iba najzákladnejšiu Lucene *Query* a to *TermQuery.* Ďalšie typy *Query* sú *BooleanQuery, PrefixQuery a* tie sú odvodené od svojej rodičovskej triedy *Query.* Bližšie si ich rozoberieme v kapitole 2.4.2, kde si popíšeme aj zvláštnu triedu *QueryParser*, ktorá dokáže vyhľadávaný dotaz analyzovať a na základe typu dotazu použiť ten typ *Query*, ktorý zodpovedá danému dotazu.

2.4.1.4. TermQuery

TermQuery je najzákladnejším typom *Query*, podporovaná Lucene. Je to jedna z prvotných typov *Query,* ktorá vyhľadáva v indexe hľadaný dotaz v tvare jedného termu:

```
Term term = new Term("contents", "lucene"ˇ);
Query query = new TermQuery(term);
```

Všetky dokumenty, ktoré majú slovo *lucene* v *contents* fielde, vráti *TermQuery* ako výsledok hľadania. Pri použití *TermQuery* nie je použitý analyzátor, ktorý by daný dotaz upravil na tvar s malými písmenami, keďže v indexe sú termy zaindexované iba v tvare malých písmen.

2.4.1.5. Hits

Trieda *Hits* je jednoduchý kontajner odkazov na výsledok vyhľadávania, ktorým sú dokumenty, ktoré zodpovedajú zadanému dotazu.

Inštancii *Hits* náležia štyri metódy zobrazené v tabuľke 2.4.1. Dokumenty, ktoré sú výsledkom hľadania majú vypočítané skóre (viac v kapitole 2.4.3) väčšie než 0 podľa vzorca uvedeného v kapitole 2.4.3. Podľa tohto skóre sú dokumenty zoradené v kolekcii *Hits*.

metódy Hits	**návratová hodnota**
length()	počet dokumentov prítomných v colekcii *Hits*
doc(n)	inštancia n-tého Lucene Dokumentu spomedzi dokumentov s najvyšším skóre
id(n)	ID n-tého dokumentu spomedzi dokumentov s najvyšším skóre
score(n)	normalizované skóre (odvodené od dokumentu s najvyšším skóre) n-tého dokumentu garantujúce , že upravené skóre bude väčšie než 0 a menšie alebo rovné 1.

tab. 2.4.1

Kvôli výkonu vyhľadávania objekty *Hits* nenahrávavajú všetky dokumenty, ktoré zachytávajú daný dotaz, ale iba malú časť z nich v danom čase. Prvých 100 dokumentov je automaticky získaných a uložených do vyrovnávacej pamäte. Pre užívateľa je zvyčajne postačujúce, aby výsledkom vyhľadávania bolo len niekoľko dokumentov s najväčším skóre, ktoré najviac zodpovedajú danému dotazu.

2.4.2. Spracovanie query – parsovanie

Na vyhľadávacie aplikácie sú požadované dve vlastnosti: sofistikovaná syntaktická analýza dotazovaného výrazu a prístup k dokumentom, ktoré majú byť výsledkom hľadania.

Lucene obsahuje zaujímavú vlastnosť, ktorá analyzuje dotazovaný výraz skrze triedu *QueryParser* . Tá dokáže rozanalyzovať a spracovať zložený dotaz (napr. "+vermirna -lod" AND "planeta OR mesiac") do jedného objektu *Query* a potom s ním pracovať pri vyhľadávaní. Primárnym účelom *QueryParseru* je schopnosť spracovať aj takýto zložený dotaz človekom veľakrát zadaný.

QueryParser potrebuje pri svojej činnosti *analyzator* pre rozkúskovanie zadaného dotazu na časti, tak aby sa dali vyhľadávať medzi termami zaindexovanými v indexu.

```
Query query = QueryParser("+vermirna -lod" AND "planeta OR mesiac"),
                                           "contents",
                                           new SimpleAnalyzer());
```

Všetky termy obsiahnuté v dotaze sú analyzované separátne. *SimpleAnalyzer*(pozri kapitola 2.3.1.3) vynechá z termov neabecedné znaky a upraví veľkosť písme na malé, aby bolo možné ich porovnávať s termami z indexu, pretože všetky termy sú indexované v indexe s malými písmenami. V našom príklade sa vyhľadajú dokumenty, ktoré obsahujú termy *vesmirna* a zároveň termy *planeta* alebo *mesiac*, avšak nesmú obsahovať term *lod*.

Lucene si pri vyhľadávaní vytvára objekt *Query*, ktorému sa priradí hľadaný výraz. Ten je však potrebné analýzovať, aby bolo možné špecifikovať o aký druh dotazu ide a následne vybrať najvhodnejšiu implementáciu *Query*:

TermQuery – object tohto typu *QueryParser* vracia, v prípade, že dotazovaný výraz pozostáva z jediného slova, odpovedajúci termu v indexe.

BooleanQuery - object tohto typu *QueryParser* vracia, v prípade, že dotazovaný výraz pozostáva z viacerých združencých termov. Združovanie je vytvárané pomocou oblých zátvoriek: ( ) a indikátoru zákazu (-), vyžadovania (+) a logických operátorov (AND, OR, NOT). V prípade, že vyhľadávny dotaz *malý pes* sa skladá z viacerých termov

bez logických operátorov, či indikátorov, tak sa vyhodnotí ako následujúci výraz *malý OR pes OR (malý AND pes)*.

PrefixQuery - object tohto typu *QueryParser* vracia, v prípade, že dotazovaný výraz pozostáva z termu začínajúceho sa na špecifický reťazec a je ukončený hviezdičkou (*). (napr. pomo* = pomoc, pomalý,...)

2.4.3. Lucene scoring

Lucene využíva systém skórovania výsledkov pri vyhľadávaní v dokumentoch. Pre výpočet skóre pre každý dokument používa Lucene následujúci vzorec obr.2.7:

$$\sum_{(t\ z\ q)} tf(t\ z\ d).idf(t).boost(t.field\ z\ d).lengthNorm(t.field\ z\ d)$$

obr. 2.7

(t – term; d – dokument; q – query; f.field – meno fieldu)

Popis faktorov je zapísaný v tab.2.4.3:

Faktor	Popis
tf(t z d)	Frekvenčný faktor termu (t) v dokumente (d) tf(term z dokumentu).
idf (t)	Inverzia frekvencia termu (t) v danom dokumentu.
boost(t.field z d)	Field boost faktor, nastavený počas indexovania.
lengthNorm(t.field z d)	Normalizačná hodnota fieldu, dáva počet termov vo vnútri fieldu. Táto hodnota je vypočítaná počas indexovania a uložená v indexe.

tab. 2.4.3

Skóre je vypočítavané pre každý vyhľadávaním získaný dokument *(d)*. Ak má niektorý z týchto dokumentov skóre väčšie než 1, tak sa prevedie normalizácia vzhľadom na jeho skóre.

Boost faktory sú zabudované do rovnice kvôli tomu, aby umožňovali ovplyvniť vplyv dotazu alebo fieldu na skóre. Lucene používa *boosting*, je to schopnosť zvýrazniť dôležitosť niektorých dokumentov, či fieldov v týchto dokumentov oproti iným. Výsledok *boostingu* má vplyv potom prí vyhľadávaní. Napríklad ak chceme vo výsledku vyhľadávania boli na prvých miestach firemne emaily ako dôležitejšie než ostatné emaily, tak sa nastaví *boost factor* dokumentu s väčšou hodnotou, ako vidno z časti kódu:

```
Document doc = new Document();
doc.add(Field.Keyword("senderEmail", getSenderEmail()));
doc.add(Field.Text("senderName", getSenderName()));
doc.add(Field.Text("subject", getSubject()));
doc.add(Field.Unstored("body", getBody()));
if (getSenderDomain().endsWithIgnoreCase("vodafone.com"))
      doc.setBoost(1.5);
else if (getSenderDomain().endsWithIgnoreCase("hotmail.com"))
      doc.setBoost(0.1);
writer.addDocument(doc);
```

Keď indexujeme emaily poslané zamestnancami domény *vodafone.com*, nastavíme *boost faktor* na 1,5, ktorý je vačší než je defaultne nastavený 1. V prípade, že sa jedná o email *hotmail.com*, tak boostaktor je nastavený na nízku hodnotu 0,1 .

Boost field *boost(t.field z d)* faktor sa vytvára počas indexovania dokumentov a charakterizuje dôležítosť jednotlivých fieldoch v dokumentoch, ktorá sa využíva pri vyhľadávaní, kedy sú zvýhodnené niektoré fieldy oproti iným.

Aj dotaz môže mať vplyv na výsledok skóre dokumentu a to pomocou *queryNorm* faktor. Jej vplyv je viditeľný viac u mnohočlenných dotazov.

2.4.4. Vyhľadávanie podľa query – Indexsearcher

Prvým krokom v procese vyhľadávaia je zadanie dotazu, ktorý chceme vyhľadať v indexe. V prípade, jednoslovného zadaného dotazu, predstavuje daný dotaz jeden term. Ak sa jedná o zložený dotaz, pozostávajúci z viacerých slov, prípadne slov spojených pomocou logických operátorov, či indikátormi, tak sa daný dotaz rozdelí pomocou *QueryParser* na niekoľko dotazov a tie sa vyhodnotia daným typom *Query*. Následuje porovnávanie termu

zastupujúci zadaný dotaz, či časť z dotazu, s termami uloženými v indexových súboroch indexu. Tento proces porovnávania sa uskutočnuje Lucene pomocou triedy *IndexSearcher*.

Pri inicializácií objektu triedy *IndexSearcher* sa konštruktóru predá parameter *dir,* získaný pomocou inštancie *FSDirectory*, reprezentujúcej umiestnenie nášho indexu:

```
Directory dir = FSDirectory.getDirectory( path, false);
IndexSearcher searcher = new IndexSearcher(dir);
```

Posledný argument metody *getDirectory()* oznamuje, že index na danom mieste existuje a ideme ho otvoriť.

```
Query q = QueryParser("vesmir", "contents", new SimpleAnalyzer());
Hits hits = searcher.search(q);
```

Po vytvorení inštancie *IndexWriter* zavoláme metódu *search()*, aby vykonala hľadanie termu *vesmir* v indexe.

2.4.5. Highlighter - zobrazenie okolia hľadaného výrazu

Highlighter je sofistikované a flexibilná utilita na zobrazenie a zvýraznenie okolia hľadaného výrazu spolu so zvýraznením výrazu.

Highlighter obsahuje tri hlavné časti: *Fragmenter, Scorer* a *Formatter*. (korešpondujú s Java prostredím).

Táto utilita vyžaduje, aby sme mu poskytli nielen text a skóre z *Hits* k zvýrazneniu, ale aj *TokenStream.* Analyzer produkuje *TokenStream* (viac v kapitole 2.3.2). Pre správne fungovanie *Highlighteru* je potrebné, aby termy v *Query* potrebujú zachytiť tokeny vysielané z *TokenStreamu.* Každý token vysielaný z *TokenStremu* obsahuje informáciu o pozícií v dokumente. Táto informácia ukazuje, kde v originálnom texte začína a končí token.

Highlighter rodelí originálny text na fragmentý použijúc *Fragmenter*. Ten rozdelí originálny text na rovnaké časti s defaultnou dĺžkou 100 znakov.

Úlohou *QueryScorer* je zoradiť fragmenty. Používa termy z dotazov typu: *TermQuery, BooleanQuery, PhraseQuery*, extrahuje ich a váži ich podľa ich *boost*

faktorov. Aby sme mohli použiť *Query* pre *QueryScorer* musia byť prepísané pomocou metódy *rewrite()* všetky jej typy do *BooleanQuery*.

Nakoniec *Formatter* zvýrazní term v texte. Zabudovany *SimpleHTMLFormatter* použije začiatocný a koncový HTML tag na zvýraznenie termu v texte. *Highlighter* používa oba *SimpleHTMLFormatter* a *SimpleFragmenter* ako východzie nastavenie. Pre každý term, ktorý sa zvýrazní, *Formatter* vráti token skóre. Toto skóre, vzniknuté pomocou *QueryScorer*, je odvodené z *boost faktor* dotazovaného člena nášho termu. A môže byť použité na ovplyvnenie zvýraznenie založené na dôležitosti termu.

3. Realizácia informačného systému

Nakoniec pristúpime k samotnému vytvoreniu nášho IR systému pre indexovanie a vyhľadávanie v textových dokumentoch. Kde sa pozrieme na náš *Rexo IR systém*, ktorý je výsledkom nášho návrhu. Pozrieme sa na jeho možnosti pri zobrazení výsledkov vyhľadávania, vplyv nastavených p

Experiment s naším *Rexo IR systémom* . Ako vplýva veľkosť indexovaných dát na veľkosť jeho indexu.

3.1. Rexo – náš vytvorený IR systém

Realizáciou nášho návrhu IR systému vznikla aplikácia *Rexo.* Táto naša aplikácia používa knižnicu *Lucene.Net.dll* ako základ pre vytváranie a komunikáciu s invertovaným indexom, analýzu textu pred indexovaním, spracovanie hľadaného dotazu a následné poskytnutie výsledku, v akom dokumente na akej pozícií sa daný hľadaný výraz nachádza, Výstupom hľadania je farebne zvýraznený nami hľadaný dotaz v texte na základe pozície tokenov, ktorá je uložená v našom indexe. Na zvýraznenie dotazu v texte bola použitá knižnica *Highlighter.Net.dll.* Táto knižnica využíva informáciu o pozícií tokenov z indexu a zároveň zobrazuje iba tie najrelevantnejšie časti textu v závislosti na ich skóre.

Výsledná aplikácia pre indexovanie a vyhľadávanie v texte bola naprogramovaná v jazyku C# vo Visual Studio 2005 Framework NET 2 s použitím knižnice *Lucene.Net.dll* (verzia Lucene.net-1.4.3.final-004) a k nej kompaktibilnej *Highlighted.Net.dll* (verzia Highlighter.Net-1.4.0.RC1-001). Pre bezproblemovy chod aplikácie je potrebné mať naištalovaný MS Framwork NET 2.

3.1.1. Schopnosti nášho IR systému

Index vytváraný *Rexom* používa jeden ***typ fieldu***:

Field.Text("contents", slovo) – analyzuje, indexuje a ukladá každé *slovo* z textu do indexu a priradí mu meno fieldu *contents*, ktoré je smerodatné pri vyhľadávaní.

Field.Text("path", C:\dokumenty\txt\Asimov.txt) – field pod meno *path* indexuje systémovú cestu dokumentu do indexu.

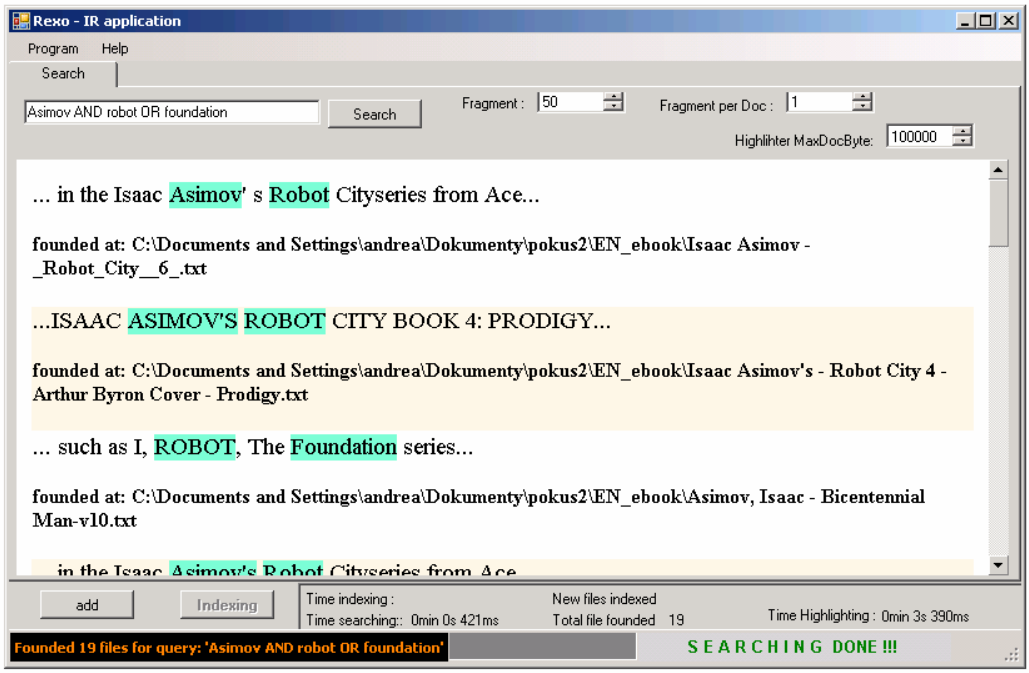
Využíva ***štruktúru zlúčených indexových súborov*** do jedného index súboru, výsledkom je jednak lepšia manipulácia ako s vačším počtom indexových súborov, ako i menšia náročnosť na systém, pri otváraní indexových súborov (viac pozri kapitola 2.1.5.2).

Rexo IR systém ***indexuje*** zatiaľ ***súbory typu *.txt***, pre rozšírenie na indexovanie ďalších typov textových dokumentov by bolo potrebné vytvoriť pre ne handlery, ktoré by získali čistý text na indexáciu, prípadne ešte metadáta, ktoré by boli indexované pod iným názvom fieldu.

Štruktúra Lucene indexu umožňuje ***updatovanie indexu*** novými dokumentami, bez narušenia štruktúry, či konzistencie dát v indexe a preto nie je potrebné vytvárať viacero indexov, ale postačí jediný.

Umožňuje vyhľadávanie boolovských výrazov pomocou boolovskych operátorov napríklad:

Asimov AND robot	ekvivalentom je výraz:	*+Asimov +robot*
Asimov AND NOT odysea	<=>	*+Asimov -odysea*
Asimov OR Clark	<=>	*Asimov Clark*
Asimov AND (robot OR foundation)	<=>	*Asimov (+robot + foundation)*



Rexo ***zvýrazní výsledok vyhľadávania*** využitím knižnice *Highlihter,* vložením tagu *slovo* do časti textu, kde sa dané *slovo* nachádza a uloží tento fragment textu do html súboru, ktorý sa zobrazí uživateľovi vo *Rexovom* webovom prehliadači.

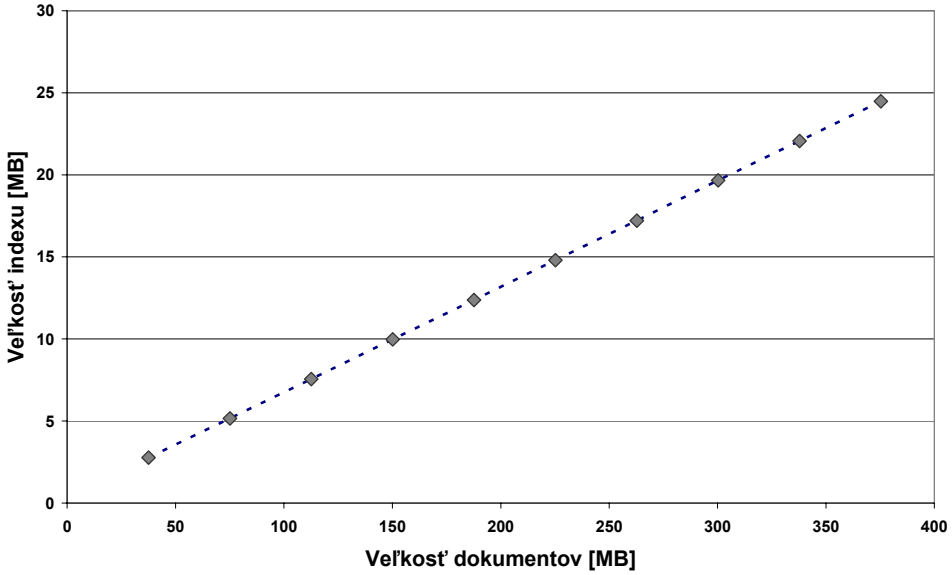
Určite zaujímavou vlastnosťou indexu vytvoreným *Rexom* je jeho ***kompaktibilita s indexami*** vytvorenými vyhľadávacími nástrojmi, ktoré využívajú akýkoľvek port Lucene.

3.2. Experiment

Po vytvorení aplikácie *Rexo* nás zaujímalo aké sú časové požiadavky pre indexovanie a vyhľadávanie, ich závislosť na veľkosti dokumentov, indexu. Preto sme otestovali pár závislostí a pokúsili sa ich vyhodnotiť.

Závislosť veľkosti indexu od veľkosti dokumentov:

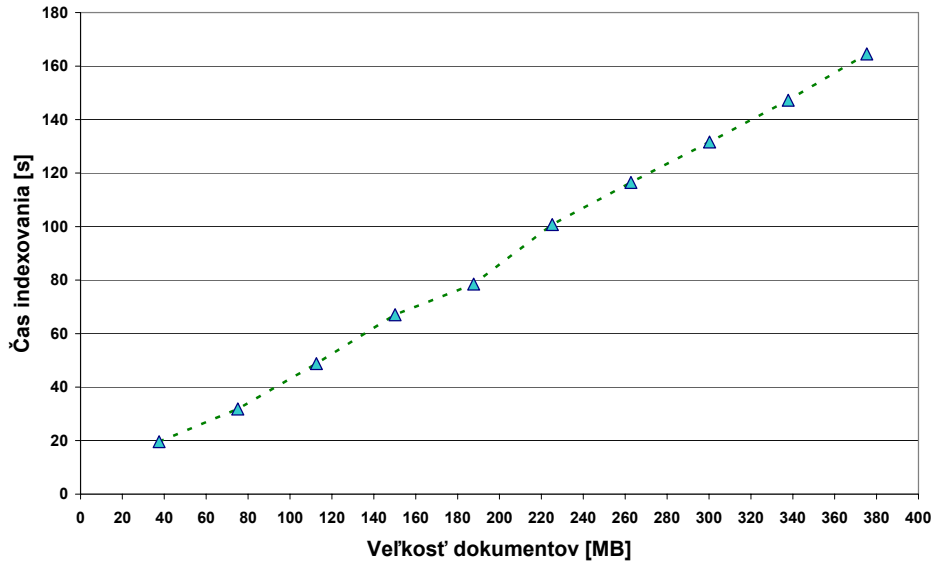
Z grafu 1 je vidno, že závislosť je približne lineárna s koeficientom úmernosti 1/15. Čo znamená, že veľkosť indexu v porovnaní s veľkosťou dokumentov je 15 krát menšia.



graf 1

Závislosť času indexovania od veľkosti dokumentov:

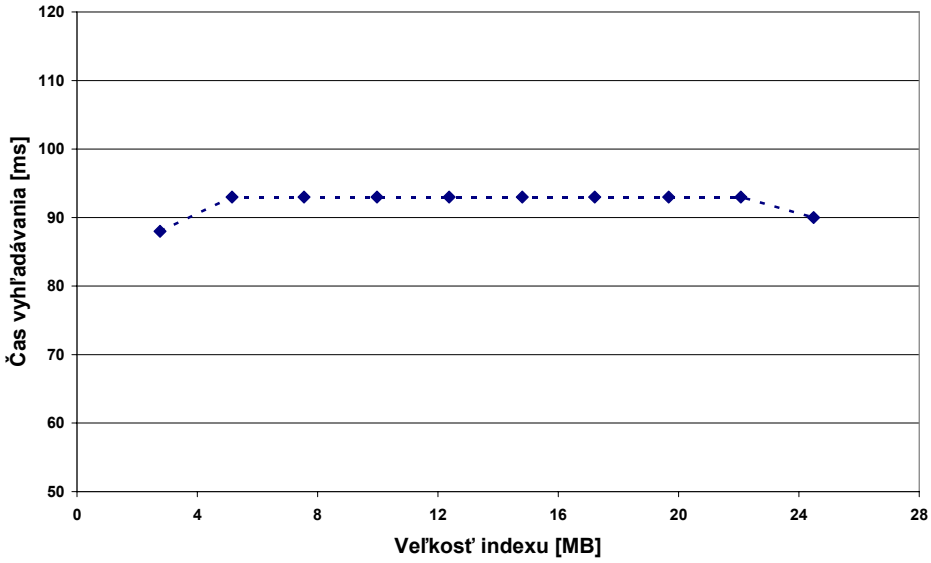
Dĺžka indexovania je tiež priamoúmerná veľkosti dokumentov ako je možné vidieť z *grafu 2*.



graf 2

Závislosť času vyhľadávania od veľkosti indexu:

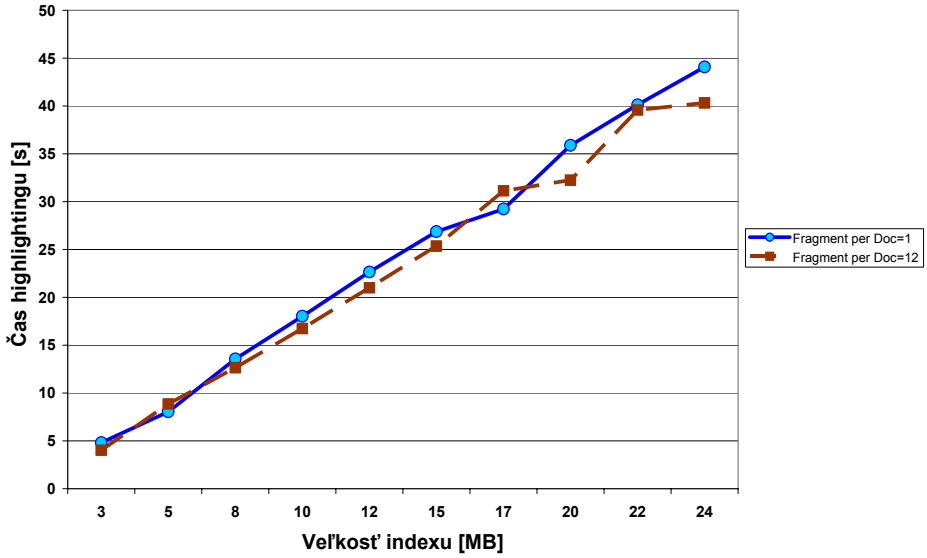
Veľkosť indexu sa neprejavila na dĺžke vyhľadávania; rýchlosť vyhľadávania bola rádovo iba 100 milisekúnd.



graf 3

Závislosť času zobrazovania od veľkosti indexu:

Samotná dĺžka zobrazenia výsledkov pomocou highlightingu bola závislá od veľkosti indexu priamoúmerne. Koeficient úmernosti sa nemenil ani zvyšovaním počtu fragmentov na dokument.



Záver

Problematika získavania textových dát, informácií je dosť široká, s tým sme sa stretli pri študovaní materiálov zaoberajúcich sa IR systémami. Určite to bude jej dôležitosťou pre stále rastúci dopyt po informáciách a ich zhromažďovaní.

Naším cieľom bola otázka kontextového prehľadávania v texte. Tejto problematike sme sa snažili porozumieť a následne navrhnúť pre jej vyriešenie IR systém, ktorý by bol schopný zobraziť okolie hľadaného dotazu. Jedným z riešení bola možnosť využiť schopnosti knižnice Lucene, ktorá poskytuje dosť rozsiahly prostriedok pre vytváranie aplikácií použitím jej špecifického invertovaného indexu, schopnosti zachytit pozíciu tokenu v texte, uchovať ho v ňom a následne možnosť jej získania pri vyhľadávaní.

Výsledkom nášho snaženia bola aplikácia *Rexo*, ktorá je schopná indexovať a vyhľadávať dotaz a zobrazovať jeho okolie v textových súboroch (*.txt), ktorý obsahuje neformátovaný čistý text. Rozšírenie tejto aplikácie pre ďalšie formátované textové súbory je možné pomocou parsingu (analýzy) týchto dokumentov a zaindexovaní jednotlivých informácií do rozdielnych typov fieldov, tým by sme naviac získali možnosť vyhľadávať aj podľa typov fieldov. Lucene knižnica nie je ukončený projekt a jeho rozšírenia do mnohých iných programovacích jazykov (C++, C#, Perl, Python, ...) sú toho dôkazom. Poskytuje možnosť stemovania, leminizovania, filtrovanie vyhľadaných dotazov a mnoho ďalších funkcionalít pre zefektívnenie vyhľadávania v textových dokumentoch.

Literatúra

[1] Melichar B.: Textové informačné systémy, Vydavatelství ČVUT 1994.

[2] Gospodnetic O., Hatcher E.: Lucene in Action, Manning 2005.

[3] Baeza-Yates R., Ribeiro-Neto Berthier: Moder Information Retrieval, ACM Press 1999.

[4] http://lucene.apache.org/

[5] http://sourceforge.net/projects/dotlucene/

[6] http://dotlucene.net/

[7] http://www.dcs.gla.ac.uk/Keith/Preface.html

[8] http://ir.dcs.gla.ac.uk/wiki/InformationRetrieval

Zoznam príloh:

CD s elektronickou podobou tohoto dokumentu, programem *Rexo IR application.*