FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce z KIV/IR

Implementace vlastního systému automatické indexace a vyhledávání dokumentů

Zdeněk Častorál
A19N0026P
zcastora@students.zcu.cz

30. 5. 2020

Obsah

1 Zadání

Cílem semestrální práce je implementovat komplexní systém automatické indexace a vyhledávání dokumentů s využitím hotových knihoven pro preprocessing.

Systém po předchozím předzpracování zaindexuje zadané dokumenty a poté umožní vyhledávání nad vytvořeným indexem. Vyhledávání je možné zadáním dotazu s logickými operátory AND, OR, NOT a s použitím závorek. Výsledek dotazu by měl vrátit top x (např. 10) relevantních dokumentů seřazených dle relevance.

Semestrální práce se bude testovat indexací dokumentů a vyhledáváním relevantních dokumentů nad vytvořeným indexem. K evaluaci bude použit evaluační skript, proto je nutné, aby projekt obsahoval třídy z projektu Interface.

Semestrální práce musí umožňovat zaindexování poskytnutých dat a dat stažených na cvičení (1. cvičení Crawler).

1.1 Minimální nutná funkčnost

Implementace semestrální práce musí nutně obsahovat:

- tokenizaci,
- preprocessing (stopwords remover, stemmer/lemmatizer),
- vytvoření in-memory invertovaného indexu,
- tf-idf model,
- cosine similarity,
- vyhledávání pomocí dotazu (top x výsledků seřazených dle relevance),
- vyhledávání s pomocí logických operátorů AND, OR, NOT,
- podporu závorek pro vynucení priority operátorů.

1.2 Nadstandardní funkčnost

Nadstandardní funkčnosti semestrální práce jsou například:

- file-based index,
- pozdější doindexování dat (přidání nových dat do existujícího indexu),
- ošetření např. HTML tagů,
- detekce jazyka dotazu a indexovaných dokumentů,
- vylepšení vyhledávání,
- vyhledávání frází,
- vyhledávání v okolí slova,
- více scoring modelů,

- indexování webového obsahu,
- další předzpracování normalizace,
- GUI/webové rozhraní,
- napovídání keywords,
- podpora více polí pro dokument,
- CRUD indexovaných dokumentů,
- zvýraznění hledaného textu v náhledu výsledků,
- dokumentace psaná v TEXu,
- implementace dalšího modelu (použití sémantických prostorů).

2 Analýza

V rámci analýzy byla aplikace rozdělena na následující hlavní části:

- předzpracování (preprocessing),
- vytvoření invertovaného indexu (zaindexování dokumentů),
- booleovské vyhledávání,
- vektorové vyhledávání,
- příkazový interpret.

2.1 Předzpracování (preprocessing)

Před samotnou indexací dokumentu je nutné jej předzpracovat. Na kvalitě předzpracování závisí celková kvalita vyhledávání. Předzpracování obecně obsahuje následující hlavní části:

- tokenizace – rozdělení vstupu na jednotlivé tokeny,
- vyjmutí stop slov – odstranění předem definovaných slov,
- stemming a lematizace – převedení slov do jejich základního tvaru nebo jejich kořenu.

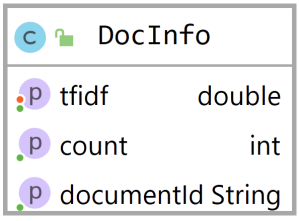
2.2 Invertovaný index

Invertovaný index je datová struktura používaná pro fulltextové vyhledávání v rozsáhlých kolekcích dokumentů. Obvykle se jedná o seřazenou kolekci předzpracovaných slov, kde ke každému slovu (resp. termu) je přiřazen seznam dokumentů, ve kterých se slovo vyskytuje.

V rámci každé položky v seznamu dokumentů v invertovaném indexu jsou uchovávány další informace:

- **`documentId`** – identifikátor dokumentu,
- **`count`** – počet opakování daného slova v příslušném dokumentu (TF - term frequency),
- **`tfidf`** – složka TF-IDF odpovídající danému slovu v invertovaném indexu pro daný dokument (`documentId`).

Položka v seznamu dokumentů je reprezentována třídou `DocInfo` (viz obrázek 2.1).



Obrázek 2.1: Položka `DocInfo` v invertovaném indexu

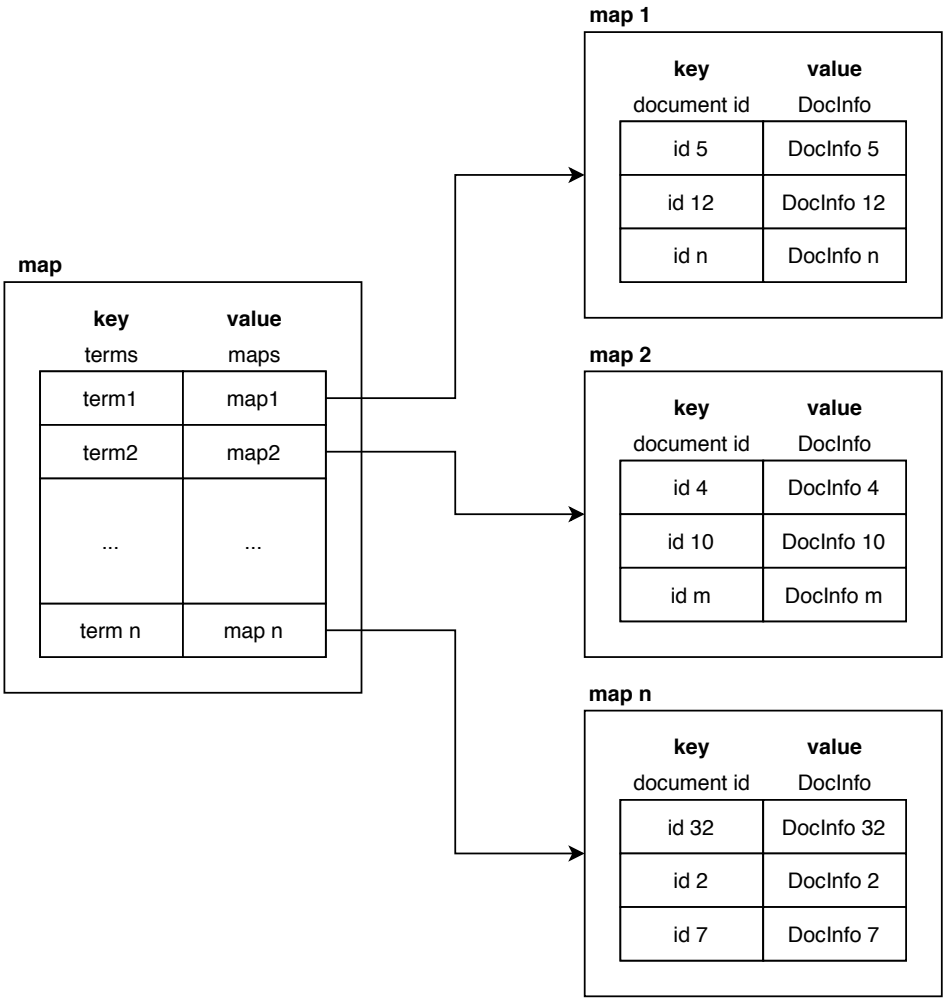
2.2.1 Reprezentace invertovaného indexu

Invertovaný index lze realizovat dvěma základními variantami:

- *maticí*,
- *spojovou strukturou (mapou)*.

První variantou je realizace invertovaného indexu prostřednictvím *matice* o rozměrech $|T| \times |D|$, kde T je množina termů a D je množina dokumentů. Pro každý term je v matici uložen seznam položka `DocInfo` pro každý dokument. Toto řešení je značně neekonomické, jelikož je nutné alokovat paměť pro celý rozsah matice, tzn. i pro dokumenty, které daný term neobsahují.

Vhodnějším řešením je použít spojovou strukturu či mapu, kde pro každý term jsou ukládány pouze ty dokumenty, ve kterých se daný term vyskytuje. Toto řešení zabírá podstatně méně paměti než předchozí řešení a také vyhledávání je rychlejší (vzhledem k tomu, že nemusíme procházet celou matici). V rámci této semestrální práce byla použita *mapa*, kde *klíčem* je daný term a *hodnotou* je další mapa, jejíž *klíčem* je identifikátor dokumentu a *hodnotou* položka `DocInfo` (viz obrázek 2.2). Tato struktura byla použita z důvodu rychlejšího prohledávání v položkách pro daný term, oproti použití spojového seznamu.



Obrázek 2.2: Realizace invertovaného indexu

2.3 Vektorový model vyhledávání

V rámci semestrální práce byl realizován *vektorový model vyhledávání*. Jedná se o implementaci ohodnoceného vyhledávání, kde každému dokumentu je přiřazeno skóre relevance k danému dotazu. Dokumenty jsou poté seřazeny dle tohoto skóre sestupně.

Každý dokument je reprezentován vektorem $\vec{d_i} = (d_{i1}, d_{i2}, ..., d_{im})$ reálných čísel v m-dimenzionálním prostoru, tzn. $\vec{d_i} \in R^m$, kde i je identifikátor dokumentu. Proměnná $m = |V|$, kde V je slovník kolekce dokumentů, každá složka d_{ij} vektoru reprezentuje váhu j-tého slova v i-tém dokumentu. Dotaz je také dokument, tzn. lze jej reprezentovat odpovídajícím vektorem.

Nejprve je nutné vypočítat vektory pro dokumenty, včetně dotazu (viz kapitola 2.3.1), poté je nutné tyto vektory porovnat, tzn. zjistit jejich podobnost. Čím je podobnost dokumentu s dotazem vyšší, tím je dokument k danému dotazu relevantnější. Jednou z možností porovnání vektorů je použití *kosinové podobnosti* (viz kapitola 2.3.2).

2.3.1 Výpočet vektorů

Nejprve je nutné vypočítat *weighted term frequency* (vážená frekvence termu) v daném dokumentu. *Term frequency* (frekvence termu) $tf_{t,d}$ termu t v dokumentu d označuje, kolikrát se term t vyskytl v dokumentu d. *Weighted term frequency* je možné vypočítat podle následujícího vzorce:

$$wf_{t,d} = 1 + \log tf_{t,d}$$

pro $tf_{t,d} > 0$, jinak $wf_{t,d} = 0$. Tato hodnota udává důležitost termu v dokumentu.

Poté je nutné vypočítat tzv. *inverse document frequency* k zohlednění důležitosti termu v celé kolekci. *Document frequency* df_t termu t označuje počet dokumentů v celé kolekci, ve kterých se term t vyskytl. *Inverse document frequency* se vypočítá následovně:

$$idf_t = \log \frac{N}{df_t},$$

kde N je celkový počet dokumentů v kolekci. Čím více se term t v dokumentech vyskytl, tím bude jeho idf_t nižší. Tato hodnota udává důležitost termu v celé kolekci.

Vynásobením $wf_{t,d}$ váhy a idf_t váhy získáme *tf-idf váhu* $w_{t,d}$ termu t v dokumentu d. Tyto váhy jsou poté jednotlivými složkami vektorů dokumentů.

2.3.2 Kosinová podobnost

Mějme dotaz, který je popsán vektorem $\vec{q}$ a dokument, který je popsán vektorem $\vec{d}$. Kosinová podobnost pro tyto dva vektory se vypočítá podle následujícího vzorce:

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}|| \cdot ||\vec{d}||} = \frac{\sum_{i=1}^{m} q_i d_i}{\sqrt{\sum_{i=1}^{m} q_i^2} \sqrt{\sum_{i=1}^{m} d_i^2}},$$

kde

- q_i je *tf-idf* váha termu i v dotazu q,
- d_i je *tf-idf* váha termu i v dokumentu d.

2.4 Booleovský model vyhledávání

Součástí této semestrální práce je také *booleovský model vyhledávání*. Získané dokumenty s použitím tohoto modelu jsou *neohodnocené* (slovo se v dokumentu vyskytlo nebo ne). Vyhledávání prostřednictvím tohoto modelu umožňuje použití operátorů `AND`, `OR`, `NOT` a vynucení priority těchto operátorů pomocí závorek.

Dotazy s využitím logických operátorů musí být v této aplikaci zadávány v *infixové notaci* (operand operátor operand). Parsování tohoto typu dotazu je realizováno s využitím knihovny *Apache Lucene*. Ta se stará o převod dotazu do *prefixové notace* (operátor operand operand) a tento dotaz vrací jako instanci třídy `Query`. Následně je rekurzivně z tohoto dotazu vytvořen strom, kde každý z listů odpovídá termu a jeho rodičem je booleovský operátor. Pro každý z listů je získán seznam instancí `DocInfo` a podle booleovského operátoru, který je společným rodičem daných uzlů, je proveden buď průnik (operátor `AND`), sjednocení (operátor `OR`), nebo doplněk (operátor `NOT`) daných seznamů. Takto se postupuje od listů až ke kořeni stromu, kde je poté uložen výsledek dotazu jako seznam instancí `DocInfo`.

3 Implementace

Aplikace byla implementována v jazyce `Java` verze `11` s využitím knihovny `Apache Lucene` pro parsování dotazu s logickými operátory. Jedná se o konzolovou aplikaci.

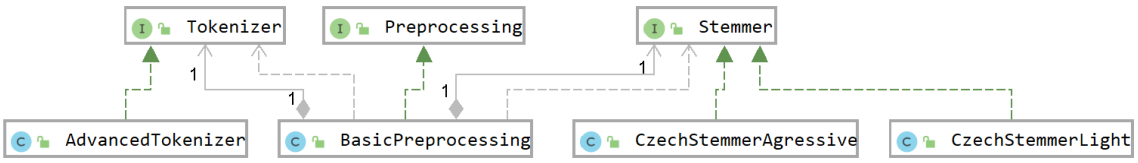
3.1 Struktura aplikace

Aplikace je rozdělena do balíků (`package`) a tříd (`class`). Hlavní balíky a třídy aplikace jsou:

- balík `counter` – obsahuje třídy k výpočtu hodnot TF-IDF a skóre dokumentů při použití vektorového modelu vyhledávání,
- balík `data` – obsahuje třídy s potřebnými daty,
- balík `preprocessing` – obsahuje třídy zajišťující předzpracování dokumentů,
- balík `search` – obsahuje třídy pro vyhledávání v dokumentech,
- balík `utils` – obsahuje pomocné třídy (např. pro vstupy ze souborů a výstupy do souborů),
- třída `App` – hlavní třída programu, obsahuje spustitelný bod aplikace,
- třída `Index` – třída reprezentující index,
- třída `Shell` – třída sloužící jako příkazový interpret.

3.2 Předzpracování (preprocessing)

Předzpracování bylo implementováno v souladu s analýzou uvedenou v kapitole 2.1. Na obrázku 3.1 je uveden UML diagram balíku `preprocessing`. Předzpracování je možno nakonfigurovat podle potřeb. Ve třídě `Index` se vytváří instance třídy `BasicPreprocessing`, kde je možné prostřednictvím parametrů zvolit konfiguraci předzpracování.



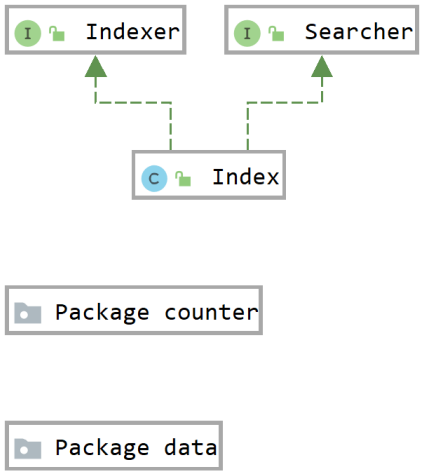
Obrázek 3.1: UML diagram balíku `preprocessing`

3.3 Indexace dokumentů

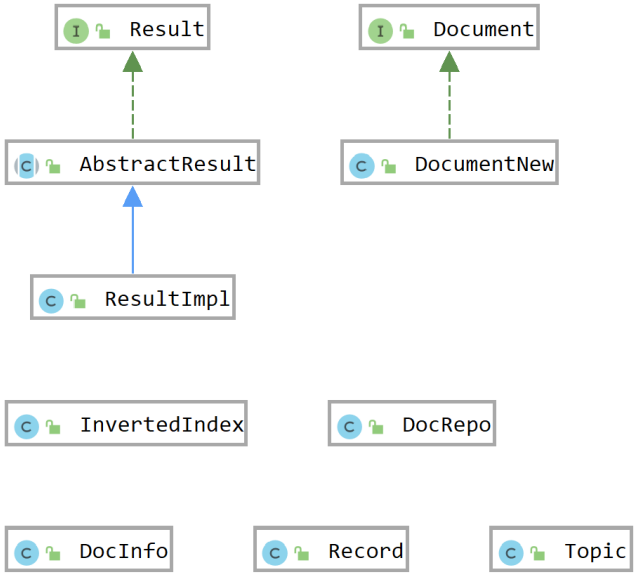
Dokumenty jsou ukládány do invertovaného indexu, který je reprezentován mapou (viz kapitola 2.2). Konkrétně je použita třída `HashMap<T>`, která používá *hashování* klíčů, díky kterému má vyhledávání podle klíče v této mapě složitost $O(1)$.

V rámci šetření paměti jsou v mapě ukládány pro každý term pouze ty dokumenty, které daný term obsahují. Vyšší rychlosti vyhledávání je dosaženo předpočítáváním hodnot

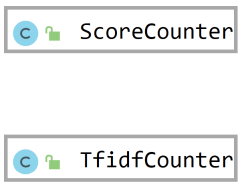
TF-IDF jednotlivých dokumentů pro dané termy již při indexaci dokumentů. Vektor *idf* je rovněž vypočítán již při indexaci, stejně jako *normy* (velikosti) vektorů jednotlivých dokumentů. Na obrázcích 3.2, 3.3 a 3.4 jsou uvedeny UML diagramy tříd zabývajících se indexováním dokumentů.



Obrázek 3.2: UML diagram – indexování



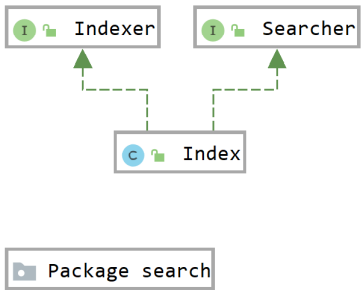
Obrázek 3.3: UML diagram – indexování (package `data`)



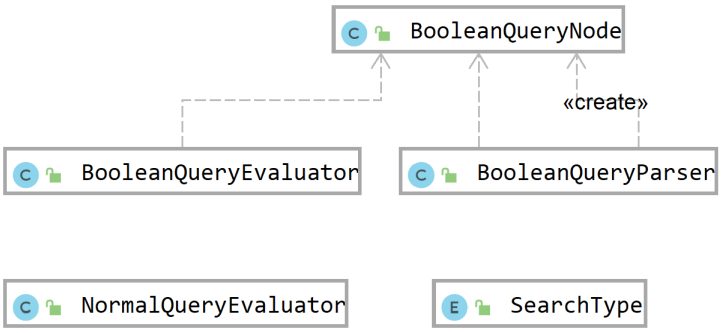
Obrázek 3.4: UML diagram – indexování (package `counter`)

3.4 Vyhledávání v dokumentech

Aplikace umožňuje jak ohodnocené vyhledávání pomocí *vektorového modelu*, tak neohodnocené vyhledávání pomocí *booleovského modelu*. Při vyhledávání prostřednictvím booleovského modelu je možné použít logické operátory `AND`, `OR`, `NOT` a vynutit prioritu operátorů pomocí závorek. Booleovský dotaz musí být v aplikaci zadán v *infixové notaci* (operand operátor operand). Při ohodnoceném vyhledávání je výpočet kosinové podobnosti dokumentu s dotazem prováděn pouze pro dokumenty, které obsahují daný term z dotazu. Díky tomu je vyhledávání rychlejší a je ušetřena paměť, oproti variantě počítání kosinové podobnosti všech dokumentů s dotazem. UML diagramy tříd zabývajících se vyhledáváním jsou uvedeny na obrázcích 3.5 a 3.6.



Obrázek 3.5: UML diagram – vyhledávání (package `counter`)



Obrázek 3.6: UML diagram – vyhledávání (package `search`)

3.5 Nadstandardní rozšíření

V rámci semestrální práce byly realizovány následující nadstrandardní rozšíření:

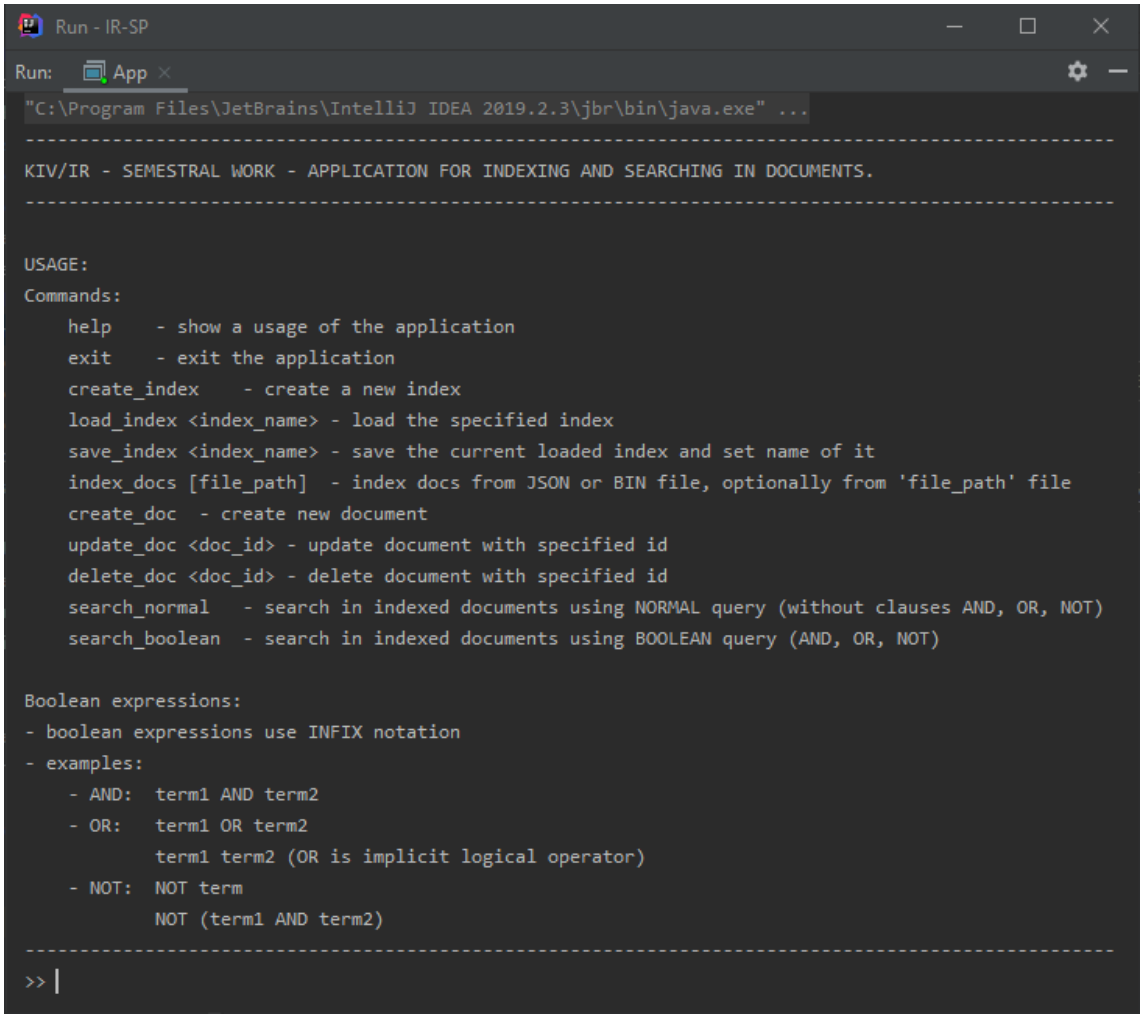
- *file-based index* – index je možné uložit do souboru a následně jej z tohoto souboru načíst,
- *pozdější doindexování dat* – do existujícího indexu, který již obsahuje množinu dokumentů, je možné zaindexovat další množinu dokumentů,
- *CRUD indexovaných dokumentů* – aplikace umožňuje vytvořit nový dokument a následně jej zaindexovat, upravit existující dokument v indexu a odstranit dokument z indexu,
- *dokumentace psaná v TEXu.*

4 Uživatelská příručka

V této kapitole je popsáno spuštění a ovládání aplikace.

4.1 Spuštění aplikace

Pro spuštění aplikace je nutné mít nainstalované `JRE` alespoň verze `8`. Po spuštění aplikace se v konzoli vypíše nápověda k použití aplikace, kde je zobrazen výčet všech příkazů (viz obrázek 4.1).



Obrázek 4.1: Spuštění aplikace

4.2 Ovládání aplikace

Aplikace je konzolová, ovládá se přes příkazovou řádku. Příkazy, které lze v aplikaci použít, jsou:

- `help` – zobrazí nápovědu k použití aplikace,
- `exit` – ukončí aplikaci,
- `create_index` – vytvoří nový index,
- `load_index <`*`název indexu`*`>` – načte index ze souboru,
- `save_index <`*`název indexu`*`>` – uloží index do souboru,
- `index_docs [`*`cesta k indexu`*`]` – zaindexuje dokumenty z JSON nebo BIN souboru (pokud nebude zadána cesta k indexu, použije defaultní cestu `./data/my_testing_data.json`),
- `create_doc` – vytvoří nový dokument,
- `update_doc <`*`id dokumentu`*`>` – upraví existující dokument specifikovaný zadaným id,
- `delete_doc <`*`id dokumentu`*`>` – odstraní dokument z indexu specifikovaný zadaným id,
- `search_normal` – vyhledá dotaz v zaindexovaných dokumentech – použije vektorový model vyhledávání,
- `search_boolean` – vyhledá dotaz v zaindexovaných dokumentech – použije booleovský model vyhledávání (povoleny logické operátory `AND`, `OR`, `NOT`).

4.2.1 Zaindexování dokumentů

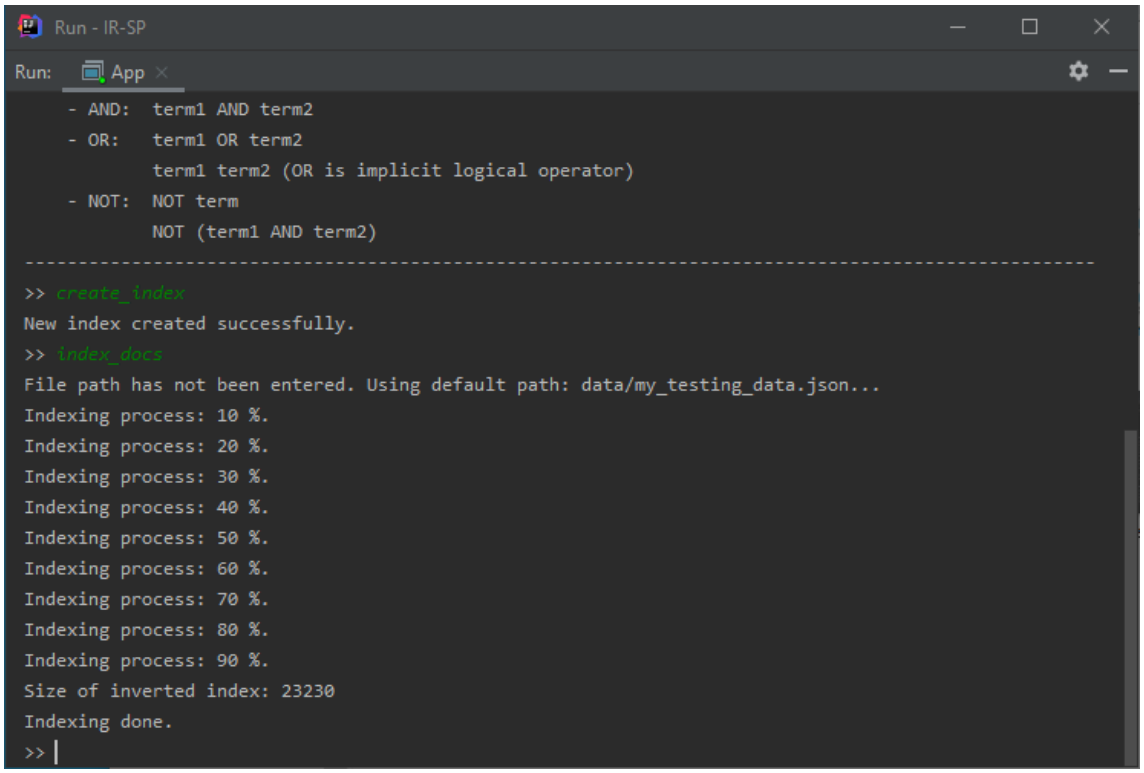
Pro zaindexování dokumentů musí být v paměti načtený index. Načtení indexu lze provést dvěma způsoby:

1. příkazem `create_index`, který vytvoří nový prázdný index a načte jej do paměti,
2. příkazem `load_index`, který načte do paměti dříve vytvořený index ze souborového systému.

Po načtení indexu je možné do něj indexovat dokumenty. To lze rovněž provést dvěma způsoby:

1. příkazem `index_docs [`*`cesta k indexu`*`]`, který načte dokumenty z JSON nebo BIN souboru specifikovaného v parametru a následně tyto dokumenty zaindexuje do aktuálně načteného indexu (pokud nebude zadána cesta k souboru, použije se defaultní hodnota `./data/my_testing_data.json`, viz obrázek 4.2),
2. příkazem `create_doc`, který vytvoří nový dokument v aktuálně načteném indexu (podrobněji v kapitole 4.2.3).

Aktuálně načtený index lze uložit do souboru příkazem `save_index <`*`název indexu`*`>`.



Obrázek 4.2: Zaindexování dokumentů – defaultní cesta

4.2.2 Vyhledávání v dokumentech

V dokumentech uložených v aktuálně načteném indexu můžeme vyhledávat dvěma způsoby:

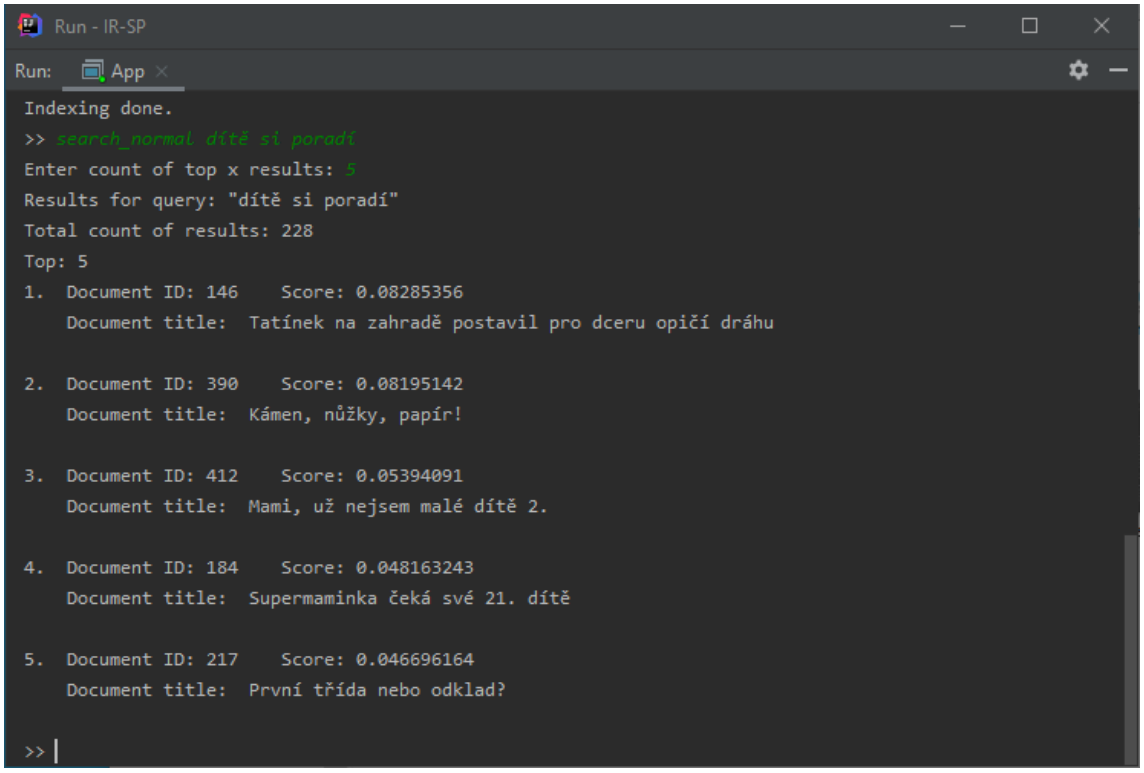
1. příkazem `search_normal` – vyhledá dotaz v zaindexovaných dokumentech prostřednictvím vektorového modelu (toto vyhledávání nepodporuje logické operátory),
2. příkazem `search_boolean` – vyhledá dotaz v zaindexovaných dokumentech prostřednictvím booleovského modelu (podporuje logické operátory `AND`, `OR`, `NOT`).

Dále je uživatel dotázán na počet nejlepších výsledků, který chce vrátit. Vyhledávání prostřednictvím vektorového modelu je vizualizováno na obrázku 4.3.

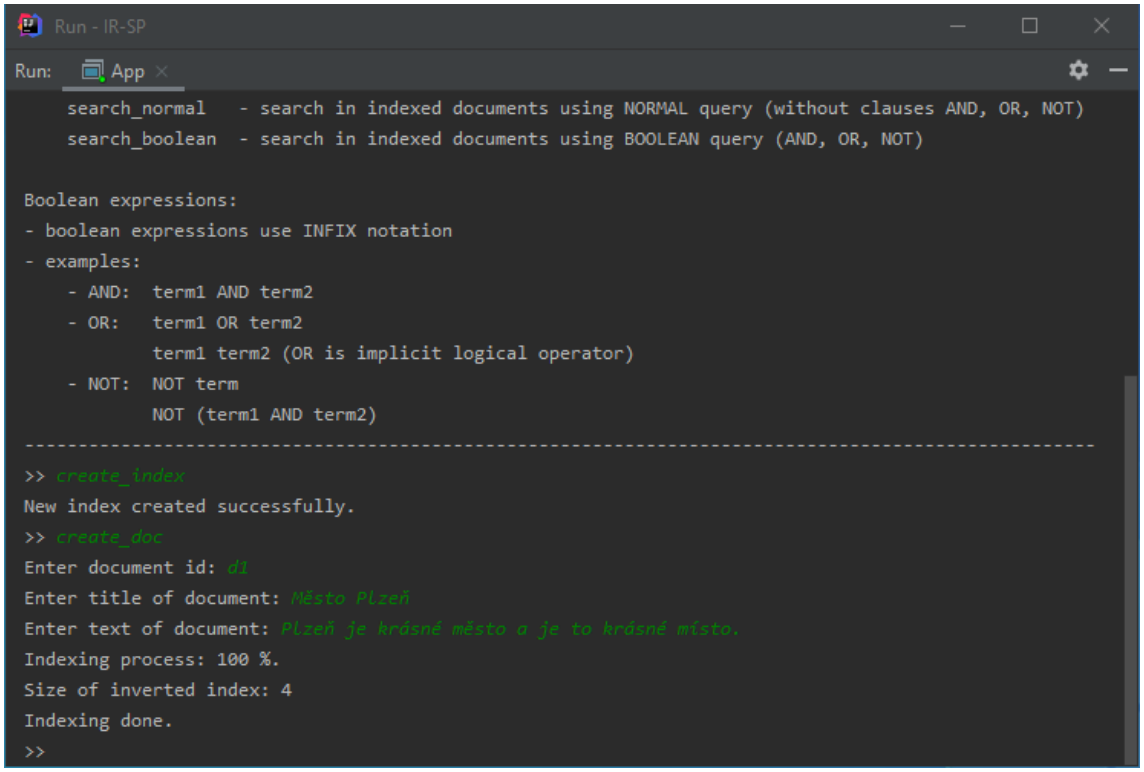
4.2.3 Vytvoření, úprava, odstranění dokumentu

V aktuálně načteném indexu je možné vytvořit nový dokument, upravit existující dokument, či odstranit dokument z indexu. K tomuto účelu slouží příkazy:

- `create_doc` – vytvoří nový dokument v aktuálním indexu (viz obrázek 4.4),
- `update_doc <id dokumentu>` – upraví existující dokument specifikovaný zadaným id v aktuálním indexu,
- `delete_doc <id dokumentu>` – odstraní dokument z aktuálního indexu specifikovaný zadaným id.



Obrázek 4.3: Vyhledávání prostřednictvím vektorového modelu



Obrázek 4.4: Vytvoření nového dokumentu v aktuálním indexu

5 Závěr

5.1 Dosažené výsledky

Pro různé konfigurace sestavení dotazu vyhledávání jsou výsledky evaluace různé. Záleží, přes která textová pole v evaluačním skriptu `TestTrecEval` se vyhledává. Vyhledávání je defaultně nastaveno na top 3000 výsledků. Zde uvádím hodnoty `map` pro různé konfigurace:

- pole `title` – $map = 0.1640$,
- pole `description` – $map = 0.1687$,
- pole `title`, `description` – $map = 0.1955$,
- pole `title`, `description`, `narrative` – $map = 0.2378$ (viz výsledky evaluace 5.1).

Výsledky evaluace 5.1: Pole `title`, `description` a `narrative`

num_q	all	50
num_ret	all	150000
num_rel	all	762
num_rel_ret	all	709
map	all	0.2378
gm_ap	all	0.1053
R-prec	all	0.2478
bpref	all	0.2358
recip_rank	all	0.4186
ircl_prn.0.00	all	0.4654
ircl_prn.0.10	all	0.4192
ircl_prn.0.20	all	0.3616
ircl_prn.0.30	all	0.3116
ircl_prn.0.40	all	0.2934
ircl_prn.0.50	all	0.2621
ircl_prn.0.60	all	0.2205
ircl_prn.0.70	all	0.1809
ircl_prn.0.80	all	0.1416
ircl_prn.0.90	all	0.0867
ircl_prn.1.00	all	0.0646
P5	all	0.2680
P10	all	0.2460
P15	all	0.2240
P20	all	0.1970
P30	all	0.1600
P100	all	0.0778
P200	all	0.0490
P500	all	0.0241
P1000	all	0.0133

5.2 Zhodnocení

Cílem této semestrální práce bylo implementovat komplexní systém automatické indexace a vyhledávání dokumentů. Realizovaná aplikace je schopná předzpracovat a zaindexovat

dokumenty, ve kterých je možné následně vyhledávat prostřednictvím vektorového modelu či booleovského modelu. Vyhledávání prostřednictvím booleovského modelu umožňuje použití logických operátorů AND, OR, NOT a také je možné vynutit prioritu těchto operátorů prostřednictvím závorek.

Minimální funkčnost popsaná v kapitole 1.1 byla splněna. Z nadstandardního rozšíření (viz kapitola 1.2) jsem realizoval file-based index, pozdější doindexování dat, CRUD indexovaných dokumentů a dokumentaci psanou v TEXu. Při implementaci této semestrální práce jsem se nesetkal s většími problémy.

Semestrální práce z mého pohledu splňuje požadavky zadání.