

Semestrální práce z předmětu KIV/IR

Indexer

Zdeněk Valeš

26.5. 2020

1 Zadání

V jazyce Java vytvořte indexer, který bude schopný indexovat zadané dokumenty a následně nad nimi provádět vyhledávání. K realizaci práce použijte připravená rozhraní.

2 Analýza

Aplikace se skládá ze dvou částí. První část tvoří jádro, které indexuje dokumenty a vyhledává na základě dotazu, druhou část tvoří jednoduché grafické rozhraní pro práci s indexerem. Jádro aplikace se skládá z textového preprocesoru, indexu a vyhledávače.

2.1 Preprocessing

V průběhu pre-processingu textu dojde k jeho rozdělení na tokeny, vyřazení stop-slov a konečně k převodu do základní formy. Před rozdělením na tokeny jsou všechna písmena v textu převedena na malá a text je zbaven diakritiky. Součástí tokenizace je také odstranění kontrakcí v anglickém textu, respektive jejich převedení na celá slova. Regulární výrazy použité k detekci kontrakcí na svém GitHubu¹ publikoval uživatel pauli31.

Stop-slova jsou slova, jež typicky patří k nejběžnějším v jazyce, ale samotnému textu nedávají přílišný význam a jejich indexace je tedy zbytečná. Seznam stop-slov je pro každý jazyk jiný a neexistuje jeden univerzální. Protože jsem v průběhu cvičení stahoval anglický text, pre-processor primárně pracuje s anglickými stop-slovy². Aby bylo možné provést TREC evaluaci, je k dispozici také soubor s českými stop slovy. Algoritmus odstranění stop-slov spočívá v iterování přes text rozdělený na tokeny a v případě, že daný token odpovídá některému ze stop-slov, je z výsledného seznamu tokenů odstraněn.

Stop-slova jsou uložena v 'nezákladní' formě (nejsou tedy zpracována stemmerem, nebo lematizérem) a proto se jejich vyhledání a odstranění z textu provádí ještě před jeho převedením na základní tvar.

Token lze na základní tvar převést stematizací, nebo lematizací. Lemmatizace je o něco složitější a i když vrací gramaticky správný základní tvar (stemmer vrací kmen), není tento potřeba a proto jsem se rozhodl použít stemmer, místo lematizéru.

2.2 Index

Datová struktura pro ukládání zpracovaných dokumentů se nazývá index. Nejjednodušší formou indexu je tzv. incidenční matice o rozměrech $|D| \times |T|$, kde D je množina uložených dokumentů a T množina termů. V incidenční matici je pak každé dvojici dokument-term přiřazena hodnota 1 nebo 0, pokud se daný term v dokumentu nachází nebo ne.

Tento způsob ukládání je značně neekonomický, protože většina hodnot v matici je 0 a ukládat gigabyty nul není žádoucí. Řešením je tzv. invertovaný index, což je struktura udržující seznam termů a pro každý term, seznam dokumentů, ve kterých se daný term

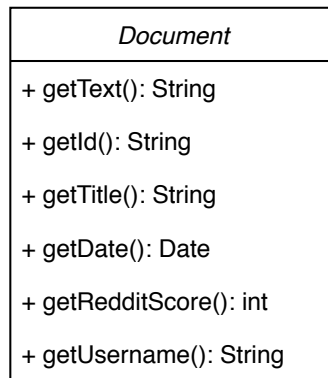
¹<https://gist.github.com/pauli31/3dce15096d87d8f32015ae519b32d418>

²<https://gist.github.com/sebleier/554280>

vyskytuje – tzv. postings list. Výhodou invertovaného indexu oproti incidenční matici je mimo jiné také možnost uložit počet výskytů termu v dokumentu, čehož jsem v práci využil při počítání TF-IDF vah.

2.2.1 Dokument

Kromě výskytů termů v dokumentu jsou také uložena jeho metadata. Základními metadaty, které vycházejí s poskytnutého rozhraní, jsou ID, text, titulek a datum. Protože jsem pracoval s komentáři ze stránky Reddit.com³, ve kterých jeden dokument představoval jeden komentář k příspěvku, přidal jsem k metadatům i skóre (suma souhlasných a nesouhlasných reakcí) a přezdívku autora komentáře. Rozhraní reprezentující metadata dokumentu je zobrazeno na obrázku 1.



Obrázek 1: UML diagram rozhraní reprezentující dokument

2.2.2 TF-IDF váhy

Na dokument je, v rámci indexu nahlíženo jako na množinu výskytů termů a konkrétní pozice termu v dokumentu není zohledněna. Tento přístup se nazývá *bag of words model* a je založen na předpokladu, že dva dokumenty reprezentované podobnými množinami termů si budou také podobné.

Jedním z klíčových ukazatelů podobnosti je počet výskytů $TF_{t,d}$, termu t v dokumentu d . Ukazatel TF nicméně nezohledňuje důležitost termů a na všechny nahlíží jako na stejně rovné. Řešením je přidání váhy *inverse document frequency*, definované jako $IDF_t = \log \frac{N}{df_t}$. Tato přiřazuje vzácnějším termům vyšší váhu a naopak běžným termům vyskytujícím se ve velkém množství dokumentů váhu nižší. Výsledná TF-IDF váha termu t v dokumentu d je definovaná jako $tf-idf_{t,d} = tf_{t,d} \times idf_t$.

Pro zefektivnění vyhledávání jsou v indexu také uloženy předpočítané IDF váhy termů, TF-IDF váhy termů v dokumentech a normy TF-IDF vektorů pro jednotlivé dokumenty. Původní verze programu tyto počítala během vyhledávání, což vedlo na velmi dlouhou dobu zpracování dotazu při TREC vyhodnocení (jednotky až desítky minut).

³<https://old.reddit.com/r/politics/>

2.3 Vyhledávání v indexu

Vyhledávání v indexu pracuje ve dvou režimech: boolean retrieval a ranked retrieval. Boolean retrieval pracuje s booleovskými dotazy a vrací dokumenty, které přesně odpovídají dotazu. Oproti tomu ranked retrieval pracuje s výše zmíněnými TF-IDF vahami, na základě kterých počítá podobnost dotazu s dokumentem a vrací dokumenty s nejvyšší podobností.

2.3.1 Boolean retrieval

Metoda vyhledávání pomocí booleovských dotazů je založena na vytvoření stromu dotazu, získání postings seznamů pro jeho listy a následného vypočtení průniku seznamů, podle booleovského operátoru, který je rodičem uzlů. Takto se postupuje od listů stromu až ke kořeni, kde je obsažen finální výsledek v podobě množiny dokumentů odpovídající dotazu.

Aby bylo možné efektivně počítat průniky postings seznamů, je nutné aby tyto měly stejné řazení. Typicky se používá řazení podle ID dokumentu, nicméně ID dokumentu je v tomto případě řetězec, proto jsem seznamy seřadil podle zahashovaného ID.

2.3.2 Ranked retrieval

Vyhledávání pomocí podobnosti je založena na interpretaci dotazu jako dokumentu a následném výpočtu podobnosti s ostatními dokumenty. Dotaz je tedy, stejně jako dokumenty, převeden na vektor TF-IDF vah a podobnost mezi ním a dokumentem d je vyjádřena jako kosinus úhlu, který mezi sebou tyto vektory svírají. Tato metoda výpočtu podobnosti se nazývá kosinová podobnost. Funkce kosinus je zvolena, protože pro totožné vektory (svírající úhel 0°) vrací 1 a pro naprosto odlišné (úhel 90°) vrací 0.

Pro vektory \vec{q} a \vec{d} je kosinus úhlu α , který mezi sebou svírají, definováno jako:

$$score_1(\vec{q}, \vec{d}) = \cos \alpha = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| \cdot |\vec{d}|} \quad (1)$$

Možnou optimalizací tohoto výpočtu je použití tzv. relativní kosinové podobnosti. Vektor dotazu \vec{q} je nahrazen vektorem \vec{Q} , který se skládá pouze z hodnot 0 a 1 (1 pro každou ne-nulovou složku \vec{q}). Tím se skalární součin ve jmenovateli zjednoduší na součet (pro ne-nulové položky \vec{Q}) a celý vzoreček má tvar:

$$score_2(\vec{Q}, \vec{d}) = \frac{\vec{Q} \cdot \vec{d}}{|\vec{d}|} \quad (2)$$

Takto získaná hodnota sice již neudává kosinus úhlu mezi vektory, nicméně platí, že:

$$score_2(\vec{Q}, \vec{d}_1) \geq score_2(\vec{Q}, \vec{d}_2) \Leftrightarrow score_1(\vec{q}, \vec{d}_1) \geq score_1(\vec{q}, \vec{d}_2) \quad (3)$$

a proto lze tento zjednodušený vztah použít k výpočtu podobnosti dotazu a dokumentu.

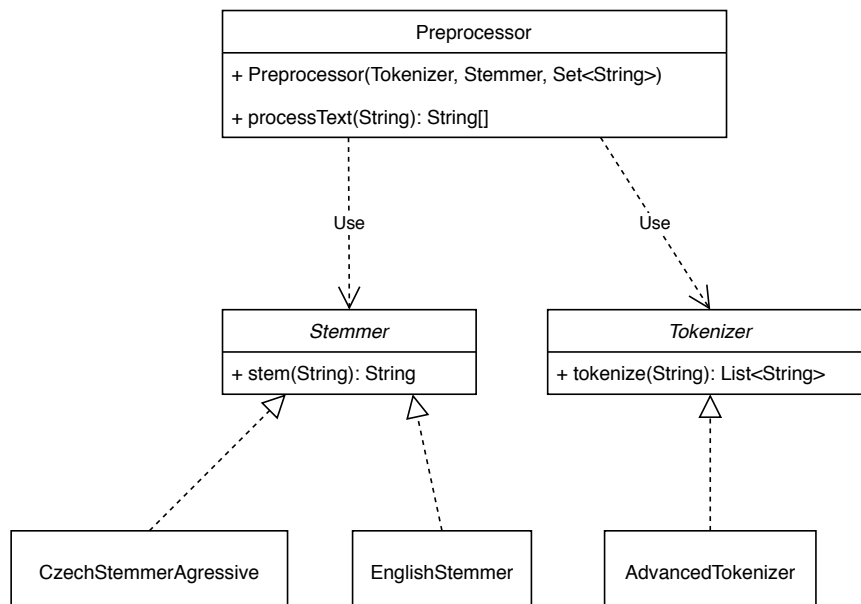
Při používání této metody typicky nechceme vrátit skóre každého dokumentu, ale jen k nejrelevantnějších dokumentů. Protože k je typicky výrazně menší než celkový počet dokumentů, není nutné provádět řazení nad všemi výsledky, ale stačí např. prvních 10. K tomu je použita halda do které jsou v průběhu výpočtu ukládány skóre jednotlivých dokumentů a po ukončení výpočtu je vybráno k nejvyšších.

3 Popis implementace

Indexer byl implementován jako desktopová aplikace v jazyce Java, skládající se z jádra a grafického uživatelského rozhraní. Jádro aplikace je rozděleno mezi balíky `core`, `data` a `preprocess`. K implementaci byla využita rozhraní ze zadání.

3.1 Preprocessing

Logika předzpracování textu je obsažena ve třídě `Preprocessor`. Tato třída tokenizuje a stemuje text pomocí dodaných tříd implementujících rozhraní `Tokenizer` a `Stemmer`, tak jak je znázorněno na obrázku 2. Tím je zajištěna pohodlná rozšiřitelnost předzpracování.



Obrázek 2: Třídy zajišťující předzpracování textu

Tokenizaci textu zajišťuje třída `AdvancedTokenizer`, která vychází převážně z prací na cvičeních. Tokenizace textu je řešena následujícími regulárními výrazy:

- `(http[s]?://[\\p{L}\\d:/.?=&+*'-_]+)` - detekce HTTP odkazů
- `(\\p{L}[\\p{L}\\d:/?=&+*'-]+)` - detekce slov obsahující písmena, čísla a znaky

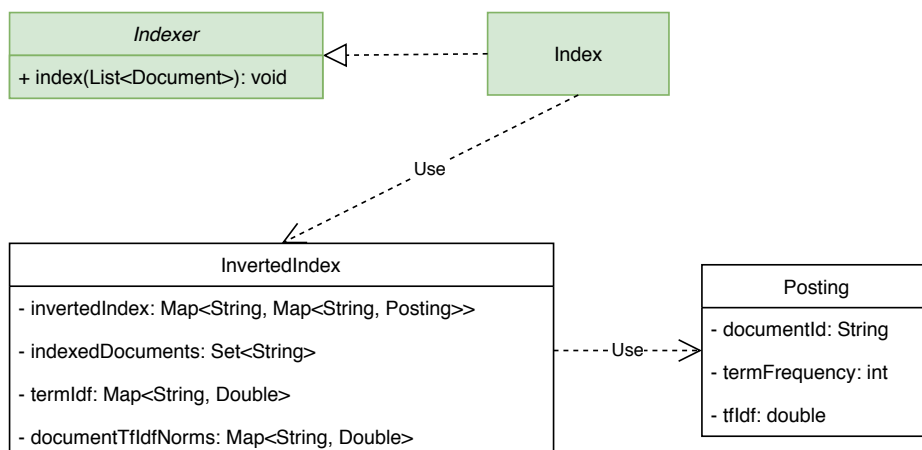
- `(\d[\d*x.]+)` - detekce čísel a číselných výrazů
- `<.*?>` - detekce HTML tagů
- `([\p{Punct}])` - detekce interpunkce

Tyto jsou spojeny operátorem `|` do jednoho regulárního výrazu, kterým je text tokenizován.

Stemování českého textu zajišťuje třída `CzechStemmerAgressive` vytvořená během cvičení. Stematizátor anglického textu `EnglishStemmer`, jež implementuje Porterův algoritmus, byl převzat z GitHub⁴ repozitáře CoreNLP.

3.2 Indexování

Třídy použité k indexování dokumentů jsou znázorněny na diagramu 3, zeleně označené jsou třídy dodané se zadáním. Samotná data jsou držena ve třídě `InvertedIndex`, která reprezentuje invertovaný index. Kromě indexovaných dokumentů jsou zde uloženy také ID všech indexovaných dokumentů, předpočítané IDF hodnoty termů a normy TF-IDF vektorů dokumentů. Většina dat je ukládána do hashovacích map z důvodu snadného čtení.



Obrázek 3: Třídy zajišťující indexování dokumentů

Ve třídě `InvertedIndex` se také nachází metody pro přepočítání TF-IDF vah termů, jež by měly být volané po indexování (aby byl index konzistentní). Atribut `invertedIndex` obsahuje vnořenou `Map`, jež umožňuje v seznamu postings vyhledávat pomocí dokument ID a není tedy nutné procházet celý seznam.

Aplikace podporuje ukládání indexu do souboru. V takovém případě je instance `InvertedIndex` serializována a uložena do zadaného souboru. Nevýhodou tohoto přístupu je závislost na

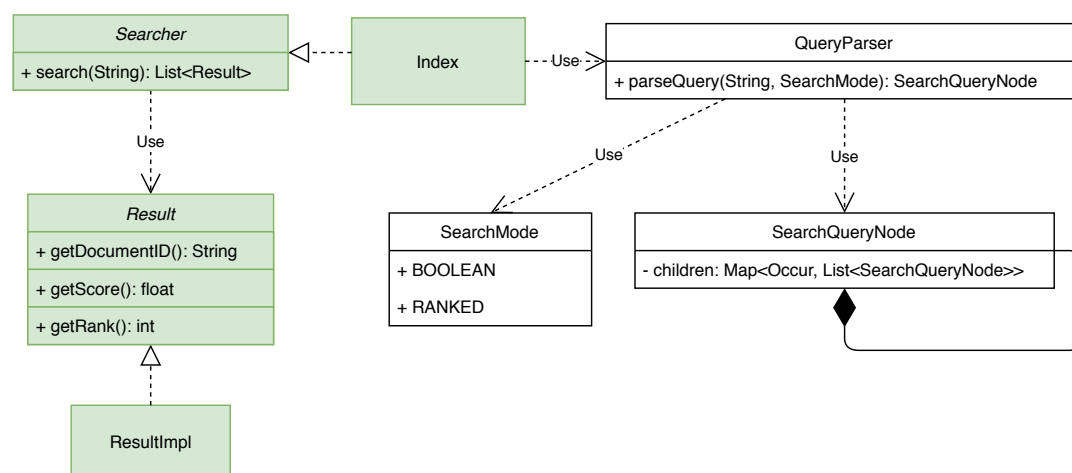
⁴<https://github.com/stanfordnlp/CoreNLP/blob/master/src/edu/stanford/nlp/process/Stemmer.java>

implementaci a pokud ve třídě `InvertedIndex` dojde ke změně, staré soubory s indexem již nejsou kompatibilní.

Dokumenty do indexu lze také načíst ze souboru, nicméně protože je tato funkcionality určena pro data, která jsem sebral crawlingem stránky Reddit během cvičení, je podporován pouze specifický formát JSON. Soubor musí obsahovat pole JSON objektů, jež mají atributy `username`, `timestamp`, `score` a `text`. Parsování JSON souborů je implementováno ve třídě `RPolDocumentReader`.

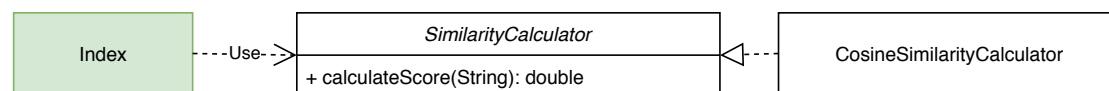
3.3 Vyhledávání

Vyhledávání nad indexem je prováděno skrze dodané rozhraní `Searcher`. Diagram 4 popisuje strukturu tříd, zajišťujících rozparsování dotazu a následné vyhledání relevantních dokumentů.



Obrázek 4: Třídy zajišťující vyhledání dokumentů v indexu

Aplikace podporuje dva režimy vyhledávání (boolean a ranked). Tyto jsou reprezentovány enumem `SearchMode`, jež slouží k výběru parsovací metody a také k výběru vyhledání dokumentů. Zpracování dotazů (pro oba režimy) je implementováno ve třídě `QueryParser`. Vnitřně je k parsování použita knihovna Lucene. Třída `SearchQueryNode` je rekurzivním objektem použitým k reprezentaci booleanových dotazů stromem. V případě boolean query jsou akceptovány pouze operátory (velkým písmem) `AND`, `OR`, `NOT` a závorky.



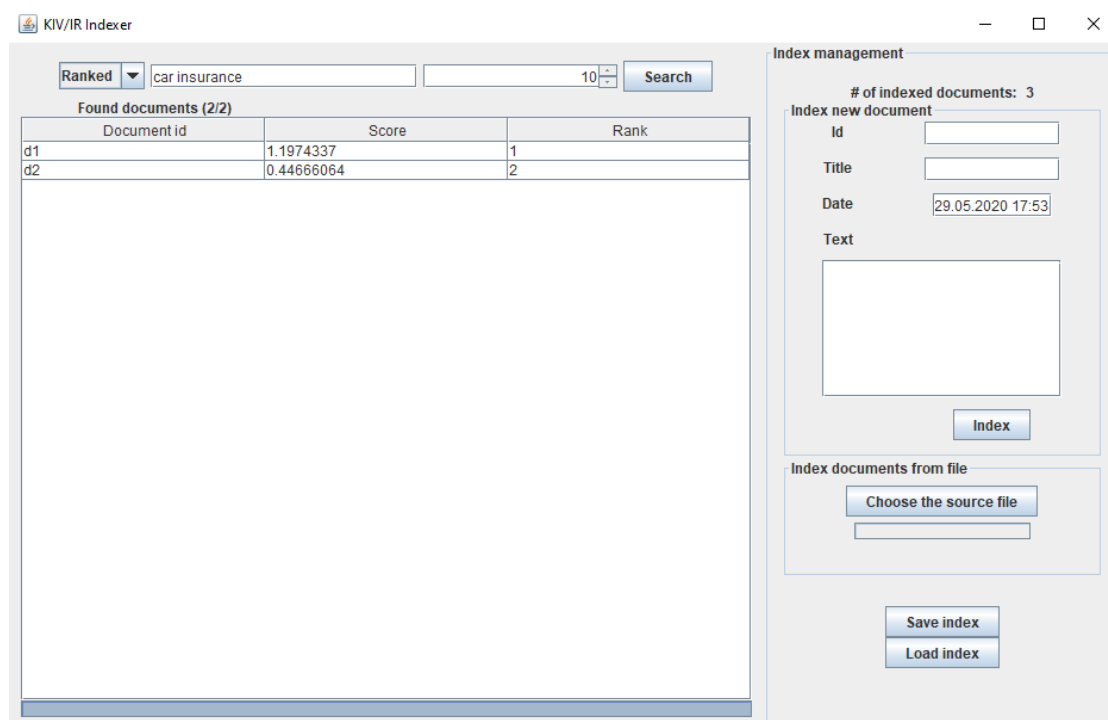
Obrázek 5: Třídy pro počítání ranked retrieval

V případě vyhledávacího režimu ranked je k výpočtu skóre použita metoda kosi-

nové podobnosti. Diagram 5 obsahuje rozhraní `SimilarityCalculator`, protože původní záminka byla implementovat více modelů podobnosti. Nicméně z časových důvodů bylo od tohoto upuštěno a jedinou implementací je tedy třída `ConsineSimilarityCalculator`, jež využívá model kosinové podobnosti.

3.4 Uživatelské rozhraní a jeho ovládání

Uživatelské rozhraní bylo implementováno pomocí frameworku Java Swing. Teto jsem si vybral kvůli jeho jednoduchosti a znalosti z předchozích prací. Celé rozhraní, které je zobrazeno na obrázku 6, je rozděleno do dvou hlavních částí. Těmi jsou vyhledávací panel (vlevo) a panel pro ovládání indexu (vpravo).



Obrázek 6: Ukázka grafického rozhraní aplikace

3.4.1 Ovládání indexu

Panel pro ovládání indexu obsahuje kontrolní prvky pro indexování jednotlivých dokumentů, indexování dokumentů ze souboru a pro uložení indexu do souboru a jeho případné načtení.

Pro indexování jednotlivých dokumentů je potřeba vyplnit pole `Id`, `Title` a `Text` (všechna jsou povinná) a následně kliknout na tlačítko `Index`. V případě úspěchu se počet indexovaných dokumentů v horní části panelu zvýší o jedna, v opačném případě je zobrazena chybová hláška.

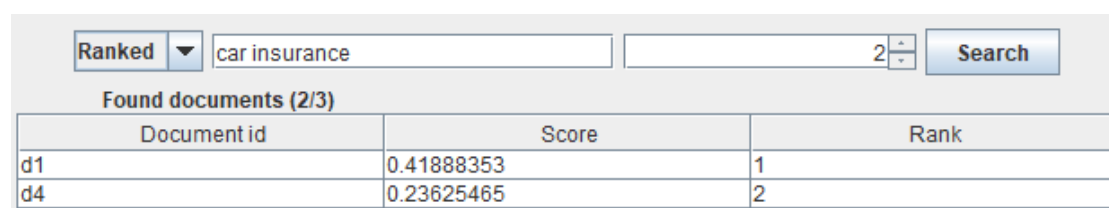
Indexování dokumentů ze souboru je určeno pro načítání dat, které jsem získal během prvních cvičení. Výběr souboru a jeho následné rozparsování a oindexování lze provést kliknutím na tlačítko *Choose the source file*. Protože indexování velkého počtu dokumentů může trvat delší čas, je pod tlačítkem umístěna komponenta pro sledování průběhu indexování.

Index je možné uložit do souboru kliknutím na tlačítko *Save index*. Jeho opětovné nahrání je možné kliknutím na tlačítko *Load index*.

3.4.2 Vyhledávání

Vyhledávací panel se skládá z vyhledávacího formuláře a tabulky pro zobrazení výsledků. Jak již bylo řečeno, vyhledávač umí pracovat ve dvou režimech - ranked a boolean retrieval. Režim vyhledávání lze vybrat komponentou v levé straně vyhledávacího formuláře. Omezení počtu výsledků lze nastavit číselným výběrem. Vyhledávání lze spustit kliknutím na tlačítko *Search*. Podobně jako u indexování, i tato akce může trvat nějakou dobu a proto je ve spodní části vyhledávacího panelu (pod tabulkou s výsledky) umístěn progress bar, kterým lze sledovat postup vyhledávání.

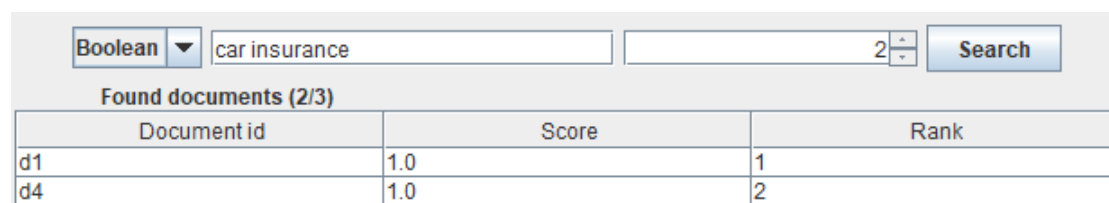
Příklady zobrazení výsledků v tabulce pro jednotlivé režimy jsou na obrázcích 7 a 8. V případě boolean retrieval je hodnota skóre 1, protože dokument buď přesně odpovídá dotazu (1), nebo neodpovídá a není tedy uveden ve výsledcích. Nad tabulkou je umístěn také indikátor počtu vrácených dokumentů (top k) a všech nalezených.



The screenshot shows a search interface with a dropdown menu set to 'Ranked', a search box containing 'car insurance', and a result count of 2. Below the search bar, it says 'Found documents (2/3)'. A table displays the results:

Document id	Score	Rank
d1	0.41888353	1
d4	0.23625465	2

Obrázek 7: Výsledky ranked retrieval



The screenshot shows a search interface with a dropdown menu set to 'Boolean', a search box containing 'car insurance', and a result count of 2. Below the search bar, it says 'Found documents (2/3)'. A table displays the results:

Document id	Score	Rank
d1	1.0	1
d4	1.0	2

Obrázek 8: Výsledky boolean retrieval

Protože předzpracování textu je schopno detekovat HTML tagy, je možné podle nich i vyhledávat. To je znázorněno na obrázku 9. Pro tento příklad byl zaindexován zdrojový kód jednoduché HTML stránky obsahující pouze titulek a text. Tento dokument byl následně vrácen vyhledávačem na dotaz obsahující pouze HTML značky.

Ranked 2

Found documents (1/1)

Document id	Score	Rank
htmlidoc	1.1036272	1

Obrázek 9: Vyhledávání podle HTML tagů

4 Závěr

Vyhledávač byl úspěšně naimplementován. Funkcionalita byla otestována jednotkovými a několika manuálními testy. TREC evaluací byl dosažen MAP 0.1378. V rámci práce jsem také zaindexoval data získaná na cvičení a zkusil v nich vyhledávat.

Případná vylepšení by mohla zahrnovat ukládání indexu do standardizovaného formátu (například JSON), čímž, by se odstranila závislost na implementaci. Dále by se také mohlo doimplementovat více modelů podobnosti. Protože komentáře příspěvků ze stránky Reddit, která jsem indexoval, obsahují vlastní skóre, bylo by dobré rozšířit možnosti vyhledávání o pole dokumentů, čímž by bylo umožněno vyhledávat komentáře obsahující určité výrazy a filtrovat je dosaženým skóre. Dotaz by pak mohl mít podobu například: `text:"Trump and Russia" score:">30"`.