|  |
| --- |
| FIT ČVUT, BI-VWM |
| Projektová dokumentace |
| Rozšířený Boolovský Model |

|  |
| --- |
| Lukáš Rynt, Martin Šír  1.1.2021 |

Obsah

[1. Popis projektu 1](#_Toc71129714)

[2. Způsob řešení 1](#_Toc71129715)

[2.1. Invertovaný index 1](#_Toc71129716)

[2.2. Zpracování dotazů 2](#_Toc71129717)

[2.3. Zpracování dotazů 2](#_Toc71129718)

[3. Implementace 2](#_Toc71129719)

[3.1. Použité technologie a knihovny 2](#_Toc71129720)

[3.2. Instalační instrukce 3](#_Toc71129721)

[4. Příklad výstupu 3](#_Toc71129722)

[5. Experimentální sekce 3](#_Toc71129723)

[6. Diskuse 3](#_Toc71129724)

[7. Závěr 3](#_Toc71129725)

1. Popis projektu

Cílem projektu je vytvořit implementaci rozšířeného boolovského modelu ukládání dat (tj. preprocessing a indexování) spolu s možností dotazování z GUI.

Vstupem je boolovský dotaz a výstupem je seznam databázových dokumentů odpovídající dotazu v klesajícím pořadí podobnosti (relevance).

1. Způsob řešení

Aplikace při spuštění serveru nejprve načte všechny soubory a postupně je stemmuje (tedy redukuje slova na společný základ, odstraňuje duplicity a stop slova) a tokenizuje (zařazuje zredukované základy slov do pole). V další fázi se iterují jednotlivá slova (teď už vlastně termy se kterými pracujeme) a počítá se jejich frekvence v kolekci. Tato frekvence se poté mapuje na termy a jména souborů, ze kterých pochází.

* 1. Invertovaný index

V další fázi se vytváří samotný invertovaný index (v naší implementaci se jedná o mapu, kde jsou termy namapovány na pole obsahující jméno souboru, ze kterého term pochází a jeho váhu). Váha se pak počítá pomocí schématu tf-idf. Mějme kolekci dokumentů a termy . Pro tyto potřeby jsou nejprve spočítány hodnoty (což je počet dokumentů v kolekci, ve kterých se daný term vyskytuje alespoň jednou). Poté můžeme spočítat hodnoty (normalizovaná frekvence termu z dokumentu vztažená na celou kolekci) následovně:

Kde je frekvence termu v dokumentu a je nejvyšší frekvence termu v celé kolekci.

Následně spočítáme (inverzní dokumentovou frekvenci) a to jako:

Výslednou váhu termu potom spočítáme jako

S takto vypočtenými vahami máme všechno potřebné k vytvoření invertovaného indexu, který si server uloží ve formě JSONu do souboru pro použití při dalším spuštění. Zároveň si server drží invertovaný index v paměti a čeká na požadavky od klienta.

* 1. Zpracování dotazů

Server přijímá dotazy ve formě textových řetězců. Ty následně parsuje na jednotlivá slova a logické spojky (program zpracovává jak formát logických operátorů typu “&&” nebo “||” ale i textový formát “and”, “or”). Výsledkem je pole tokenů obsahující logické operátory a jednotlivá slova.

V další fázi je z tohoto pole tokenů vytvořen AST (Abstract Syntax Tree), kde jednotlivé nody stromu představují logický operátor nebo samotný výraz. Tento strom je tvořen rekurzivně zpracováním pole tokenů. V rámci zpracování jsou brány v potaz také priority operátorů.

* 1. Zpracování dotazů

Samotný strom se ve výsledku zpracovává rekurzivně. Při vyhodnocení listů stromu – tedy samotných výrazů – jsou vráceny hodnoty v invertovaném indexu včetně váhy a umístění souboru.

Při vyhodnocení AND operátorů se postupuje podobně jako při zatřiďování v merge sortu. Vzhledem k tomu, že výrazy v listech jsou vzestupně třízeny podle jmen souborů kde se nachází, můžeme tuto strategii využít. Postupně zvyšujeme index levého nebo pravého souboru tak, abychom byli stále na stejné úrovni. Indexy odpovídají číslu souborů. V momentě, kdy se indexy rovnají, přidáme nový záznam do výsledku. Váha toho záznamu bude počítána následujícím způsobem (a jsou popořadě váhy levého a pravého výrazu, které byly rekurzivně zpracovány dříve):

Vyhodnocování OR operátoru probíhá podobně s tím rozdílem, že v případě, že se indexy neshodují tak je stejně přidáme do výsledku. Další věc, která se zde liší je samotné počítání váhy při shodě. Ta se počítá tímto způsobem:

1. Implementace
   1. Použité technologie a knihovny

Pro implementaci projektu byl zvolen jazyk Javascript. Projekt je rozdělen na část serveru a klienta, kde klient využívá framework React.js a server Node.js. Program využívá pro svou činnost různé balíčky (knihovny) pro práci s komunikací mezi klientem a serverem, kromě těchto využívá jmenovitě balíček [Natural](https://www.npmjs.com/package/natural), který se stará o zpracování stemmování/lemmatizace textu a následnou tokenizaci. Aplikace nevyužívá žádnou specializovanou databázi a přijímá soubory ve file systému počítače s uloženým textem.

* 1. Instalační instrukce

1. Příklad výstupu
2. Experimentální sekce
3. Diskuse
4. Závěr