FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z KIV/IR

Implementace komplexního IR systému

Dominik Zappe – A23N0011P
(zapped99@students.zcu.cz)

Květen 2024

Obsah

1 Zadání

Standardní zadání

Implementace vlastního systému automatické indexace a vyhledávání dokumentů.

Minimální nutná funkčnost semestrání práce pro získání 15 bodů (a tedy potenciálně zápočtu):

Tokenizace, preprocessing (stopwords remover, stemmer/lemmatizer), vytvoření in-memory invertovaného indexu, tf-idf model, cosine similarity, vyhledávání dotazem vrací top x výsledků seřazených dle relevance (tj. vektorový model - vector space model), vyhledávání logickými operátory AND, OR, NOT (booleovský model), podrobná dokumentace (programátorská i uživatelská), podpora závorek pro vynucení priority operátorů.

Semestrální práce **musí umožňovat** zaindexování dat stažených na cvičení (1. bodované cvičení Crawler) a libovolných dalších dat ve stejném formátu. Obě sady dat je možné zaindexovat nezávisle na sobě.

Semestrální práce **musí umožňovat** zadávat dotazy z GUI nebo CLI (command line interface) a při zadávání dotazů je možno vybrat index a model vyhledávání (vector space model vs. boolean model). Výsledky vyhledávání **obsahují i celkový počet dokumentů**, které odpovídají zadanému dotazu.

Semestrální práce **musí umožňovat** zaindexování evaluačních dat, vyhledávání v nich a spuštění evaluace. Pokud pracujete pouze s daty v Angličtině, není evaluace povinná (jelikož nemáme evaluační data pro Angličtinu)

Nadstandardní funkčnost(lze získat až dalších 15 bodů)

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

- indexování webového obsahu - zadám web, program stáhne data a rovnou je zaindexuje do existujícího indexu (2b)
- další předzpracování normalizace
- GUI/webové rozhraní
- napovídání keywords
- podpora více polí pro dokument (např. datum, od do)
- zvýraznění hledaného textu v náhledu výsledků
- vlastní implementace parsování dotazů bez použití externí knihovny
- implementace dalšího modelu (použití sémantických prostorů, doc2vec, Transformers - BERT) atd. (1-5b podle náročnosti)
- podpora více jazyků
- jakékoliv další užitečné vylepšení, které vás napadne

Pokud není uvedeno jinak, dostanete za každou nadstandardní funkčnost 1 bod za funkční řešení, 2 body v případě "hezkého"řešení (efektivnější, nebo v jiných ohledech lepší než běžné)

Nejlepší práce (nejefektivnější/nejlepší výsledky vyhledávání) získá další bonusový bod. V případě že budou výsledky hodně podobné, může získat bonusový bod více prací.

2 Implementace

Většina aplikace je implementována v jazyce C++, pouze crawler a detekce jazyka je implementována v jazyce Python. Zdrojový adresář /src/cpp_indexer je rozdělen do čtyř podadresářů – data, eval, gui a index. Tato dokumentace bude rozdělena do podkapitol, které více méně odpovídají tommuto rozdělení zdrojového adresáře.

2.1 Načtení a předzpracování dat

Data byla „crawlována“ z webové stránky `https://zaklinac.fandom.com/wiki/ZaklnaÄ‰_Wiki` – je tedy pevně předpokládaný formát vstupních dat na základě struktury, která je dána crawlerem. Očekávaný formát je *JSON*, který má klíče *title*, *toc*, *h1*, *h2*, *h3* a *content*. Na každém klíči se nachází pole stringů. „Nacrawlovaná“ data ve správném formátu je možné stáhnout z `https://drive.google.com/drive/folders/1jFOxtOwzfwmeruiqLLb7ncQo08mDbBEM?usp=sharing`.

Samotné předzpracování probíhá následovně – nejprve je vstupní text převeden na malé znaky. Následně jsou ošetřeny HTML tagy (ačkoliv by se v dokumentech žádné neměly vyskytovat – ošetřeno již crawlerem). Posléze je text tokenizován vlastním tokenizerem, který mimo tokenizování také nasčítává poziční index jednotlivých tokenů. Tokenizer mimo jiné navíc řeší speciální tokeny jako datumy, hypertextové odkazy, časy, apod. Navíc se tokenizer zbavuje interpunkce a bílých znaků. Číslice se snaží řešit roztrháním tokenu na slovní a číselné části a čísla se také později indexují. Po tokenizaci by normálně ještě docházelo k odstranění stop slov, ale z důvodu pozičního indexu je tato funkčnost nevyužita. Následně se provádí *lemmatizace* (případně *stemming*). Na závěr je ještě možnost odstranění duplicitních tokenů, což je ale v souvislosti indexace nežadoucí, a tedy tato funkčnost je nevyužita.

2.2 Indexace

V práci je implementována komplexní třída reprezentující *indexer*. Tento indexer obsahuje *kolekci* tokenizovaných dat, *cache* původních dokumentů, *klíčová slova* pro GUI našeptávač, *index* samotný (dvojí – indexují se zvlášť nadpisy a obsahy dokumentů), předpočítané *normy* jednotlivých dokumentů (opět dvojí) a v neposlední řadě *poziční mapu slov*, která slouží jako poziční index. Index je implementován jako invertovaný seznam – tedy mapa slov, kde na konkrétním klíči je uložena hodnota *idf* daného slova a pole ID dokumentů, ve kterých se dané slovo vyskytuje a jaké skóre *tf-idf* s daným dokumentem má.

Samotná indexace probíhá tak, že se dokumenty předzpracují a uloží se do jak v původní, tak v předzpracované formě do cache. Následně se napočítají jednotlivá *tf-idf* skóre a

předpočítají se *normy* dokumentů. Při indexaci se navíc provolává Python kód pro detekci jazyka a do cache dokumentů je navíc ukládán detekovaný jazyk daných dokumentů.

2.3 Vyhledávání

Při vyhledávání je nejprve předzpracován dotaz samotný.

Vektorové vyhledávání

Nejprve je vypočítáno skóre *tf-idf* daného dotazu pomocí napočítání *tf* a následné pronásobení uloženým *idf* z indexu. Následným užitím kosínovy podobnosti je vypočítána podobnost jednotlivých dokumentů se zadaným dotazem (dotaz je vyhledaván v nadpisech a obsazích separátně). Následně je na základě zadaného parametru převážena hodnota nadpisu (1.5 × váha nadpis + váha obsahu) do jednoho výsledného skóre. Nyní jsou v základní verzi výsledky seřazeny na základě skóre od nejrelevantější po ty nejméně relevantní a výsledky jsou oříznuty na předem určený počet (parametrem určený počet). V rozšířené verzi se ještě provádí hledání v blízkosti slova, kde se na základě pozičního indexu skáče po slovech z dotazu v dokumentech a v jejich okolí se hledají v pozičním indexu ostatních slova z dotazu. Tímto způsobem jsou výsledky před řazením ještě filtrovány. Navíc když se zvolí blízkost rovna jedné, jedná se již o hledání konkrétní fráze.

Booleovské vyhledávání

Před samotným vyhledáváním je nutné správně předzpracovat zadaný dotaz. K tomu slouží vlastní implementace parseru booleovských dotazů, který dokáže vyhodnotit, zda je vůbec dotaz validní, a když ano, tak ho převádí pro pozdější jednoduché zpracování do *postfixového zápisu*.

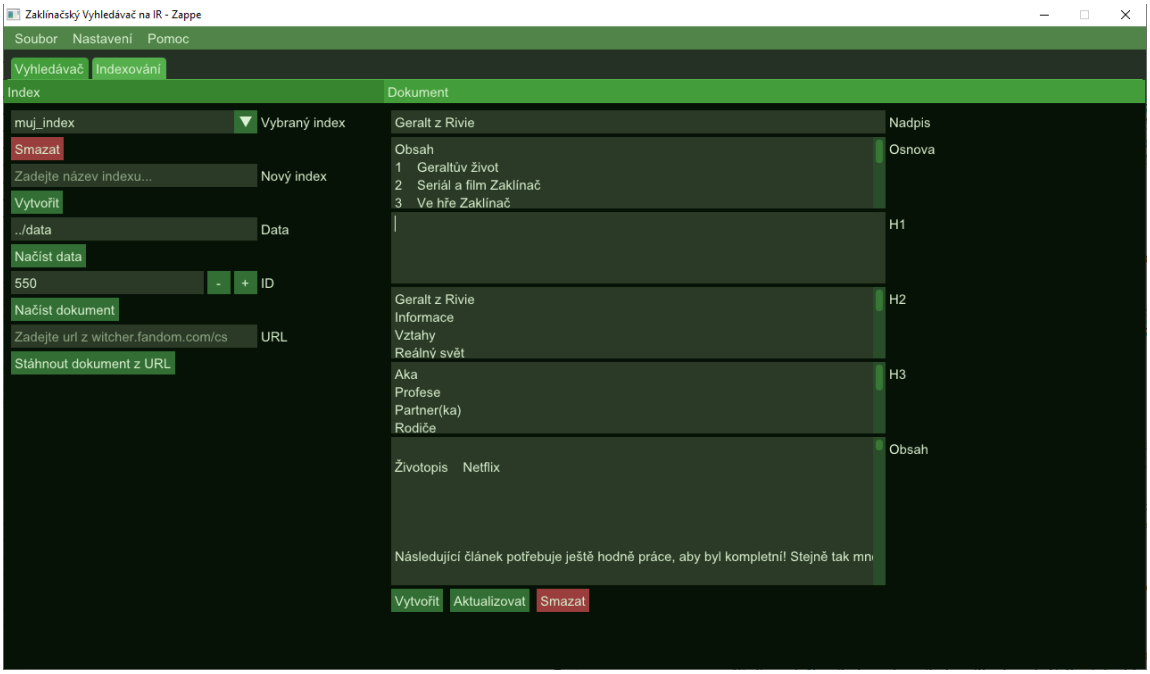
Samotné vyhledávání tedy očekává booleovský výraz v postfixovém zápisu a je tedy možné vyhledávání realizovat pomocí zásobníku. Když se najde operátor *AND* udělá se množinový průnik dvou posledních výsledků v zásobníku, když se najde operátor *OR* udělá se množinové sjednocení dvou poslední výsledků v zásobníku, když se najde operátor *NOT* provádí se množinový rozdíl všech dokumentů a posledního výsledku v zásobníku a když se najde obyčejné slovo, vyhledá se v indexu a výskyty tohoto slova se ukládají na zásobník. Důležité je upozornit, že když se slovo v indexu nevyskytuje, je stále nutné na zásobník uložit prázdnou množinu.

2.4 Grafické uživatelské rozhraní

GUI je postavené na knihovně *imgui* – `https://github.com/ocornut/imgui`. Jedná se o minimalistickou knihovnu, která dovoluje pracovat s grafikou na nízké úrovni (OpenGL). V hlavním okně je klasicky menu nabízející vypnutí aplikace, nastavení okna a „o aplikaci“. Okno mimo jiné obsahuje dvě hlavní záložky – záložka pro vyhledávání a záložka pro indexování.

Záložka indexování

Tato záložka slouží pro tvorbu indexu, je možné existující indexy smazat, vytvořit nové a načíst data do vybraného indexu. Dále je možné si podle ID získat určitý dokument a pomocí formuláře v okně ho jednoduše upravit. Je také možné si nový dokument vytvořit, či již existující (opět na základě ID) smazat. Dodatečnou možností je naindexovat webový obsah, ale pouze z indexované stránky – `https://zaklinac.fandom.com/wiki/ZaklnaÄ‰_Wiki`. Vše popsané je vidět na Obrázku 1

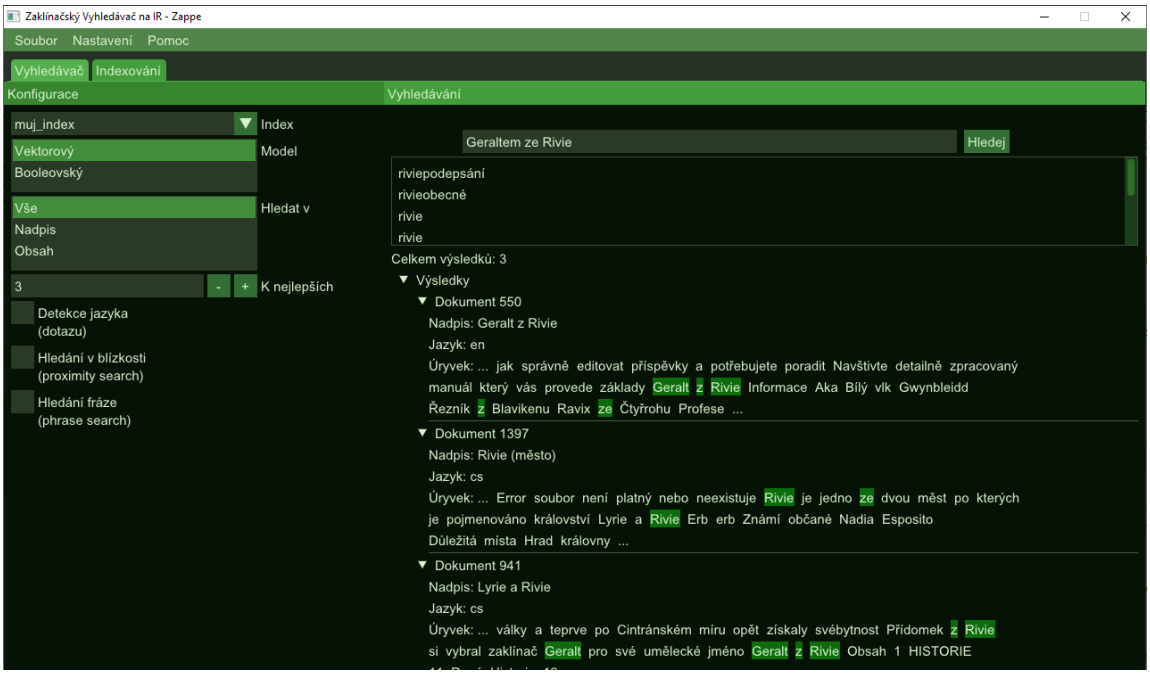


Obrázek 1: Ukázka GUI – záložka indexování

Záložka vyhledávač

Tato záložka slouží jako hlavní okno aplikace. Zde je možné si vybrat index, ve kterém se vyhledává a následně parametrizovat vyhledávání – je možné si vybrat model, kterým se hledá, fieldy ve kterých se hledá a další možnosti jako např. zobrazení K nejlepších výsledků atp.

V pravé části tohoto okna se nachází samotný vyhledávač, který je obdařen našeptávačem klíčových slov a zobrazuje dynamicky tvořené úryvky z relevantních dokumentů, ve kterých mimo jiné zvýrazňuje vyhledávaná slova – viz Obrázek 2



Obrázek 2: Ukázka GUI – záložka vyhledávač

2.5 Seznam implementovaných nadstandardních funkčností

File-based index

Při spuštění je možné použít příznak „--file-based“, který zapříčiní to, že se indexy nenačítají ze složky /index, ale z /index_file_based. Veškerá funkčnost – tj. indexace a vyhledávání – má své file-based varianty a toto řešení by mělo lehce zpomalit běh programu, ale výrazně ušetřit paměť RAM za běhu.

Pozdější doindexování dat - přidání nových dat do existujícího indexu

Tato funkčnost již byla zmíněna v podkapitole o záložce indexování. Jsou implementovány všechny čtyři funkce *CRUD* – Create, Read, Update, Delete. Všechny operace je možné používat z GUI – záložka indexace – viz Obrázek 1.

Ošetření HTML tagů

Tato funkčnost opět již byla zmíněna v předchozích kapitolách.

Detekce jazyka dotazu a indexovaných dokumentů

Kvůli této funčknosti není aplikace čistě C++, neboť detekce jazyka je implementována v Pythonu. Kvůli této skutečnosti je implementován vlastní „exec()“, který zachytává standardní výstup Pythonvské aplikace a parsuje si ho uvnitř C++ aplikace. Detekce jazyka dokumentů je vidět na Obrázku 2.

Lepším řešením by bylo z Pythonvských aplikací udělat server a dotazovat se ho přes API, ale to již bylo nadrámec časových možností.

Vyhledávání v okolí slova

Tato funkčnost byla již zmiňována v kapitole o vyhledávání, ve zkratce se jedná o vylepšení, které je možné uplatnit díky pozičnímu indexu. Checkbox pro tuto možnost vyhledávání je vidět na Obrázku 2.

Vyhledávání frází (i stop slova)

Kvůli této funkčnosti není užito mazání stop slov. Tato funkčnost je pouze speciálním případem vyhledávání v okolí slova (okolí rovno jedné). Checkbox je opět vidět na Obrázku 2.

Indexování webového obsahu - zadám web, program stáhne data a rovnou je zaindexuje do existujícího indexu

Tato funkčnost již opět byla zmíněna dříve v této dokumentaci. Toto konkrétní vylepšení však není dokonalé – nedá se zadat libovolný web, ale pouze indexovaná fandom wikipedie. Textové pole k této funkcionalitě je vidět na Obrázku 1.

GUI/webové rozhraní

Bylo zvoleno GUI, opět byla v této dokumentaci věnována celá kapitola tomuto vylepšení – viz Obrázky 1 a 2.

Napovídání keywords

Klíčová slova jsou uložena v čase indexování přímo do instance objektu indexeru. Následně tento set využívá grafické uživatelské rozhraní, které hledá právě psané slovo jako substring známých klíčových slov. V GUI je navíc možné kliknout myší na jedno z napovídaných slov a bude tak automaticky doplněno. Seznam s klíčovými slovy je v GUI vidět na Obrázku 2.

Zde je možným vylepšením do budoucna vlastní implementace datové struktury *TRIE*, mělo by to zásadně urychlit vyhledávání oproti užitému setu.

Podpora více polí pro dokument (např. datum, od do)

Jak již bylo dříve zmíněno, je dovoleno hledat pouze v nadpisech, pouze v obsahu nebo v obojím zároveň, kde se pak dává větší váha skóre nadpisu. Opět je funkčnost vidět v GUI na Obrázku 2.

Zvýraznění hledaného textu v náhledu výsledků

Tato funkčnost již byla zmíněna v textu dokumentace – využívá pozičního indexu, který je počítán přes tokeny. Zvýrazněný text je vidět v pravé části Obrázku 2.

Vlastní implementace parsování dotazů bez použití externí knihovny

Je implementováno vlastní předzpracování jak obecného dotazu, tak i booleovského dotazu.

3 Evaluace a výsledky

Evaluace byla provedena na dodaných datech ze zadání – *trec eval*. Bylo testováno více různých případu předzpracování a vyhledávání – konkrétně hlavně lemmatizace a stemming.

Z Tabulky 1 je zřejmé, že lemmatizace je lepší než stemming. Dává to pro češtinu i smysl, neboť v si myslím, že v češtině jsou kořeny důležitější, než pouhé okrajování koncovek. Zároveň si myslím, že tento výsledek může být způsoben i implementací stemmeru, kterou jsem pro C++ našel – možná není ideální.

Zároveň je také z Tabulky 1 vidět, že nejvyšší dosažený MAP (*Mean Average Precision* je 9.2 %, což mi osobně přijde nízké – můj IR systém není v žádném ohledu dokonalý.

Tabulka 1: Porovnání výsledků lemmatizace a stemmingu v kombinaci s různými částmi dotazu

Část dotazu	**MAP**
Lemmatizace	
Title + Description	0.0801
Title	0.0924
Description	0.0707
Stemming	
Title + Description	0.0160
Title	0.0221
Description	0.0143

4 Uživatelská příručka

Aplikace je převážně napsaná v jazyce C++, pro její spuštění tedy není žádná prerekvizita, neboť odevzdaný zip obsahuje přeloženou aplikaci. Pro správný běh indexování webového obsahu a detekci jazyka je však nutné mít nainstalovaný Python 3+ (konkrétně byla testována nejméně verze Python 3.7+).

Pro vlastní překlad je nutné mít nainstalované standardní knihovny jazyka C++ (součást překladače), konkrétně standard C++20. Pro překlad je dále nutné mít nainstalovaný CMake ve verzi alespoň 3.20.

4.1 Překlad

V kořenovém adresáři projektu se nachází soubor CMakeLists.txt, který zajišťuje hladký průběh překladu za pomocí nástroje CMake (verze 3.20+).

Překlad je možné uskutečnit následujícím příkazem

```
C:\SP\build>cmake ../
C:\SP\build>cmake --build . --config Release
```

```
user@pc:/SP/build$ cmake -DCMAKE_BUILD_TYPE=Release ../
user@pc:/SP/build$ make
```

Z důvodu užívání Python skriptů za běhu programu je dále nutné si nainstalovat i závislosti do Pythonu za pomocí standardního manažera balíčků pip.

```
C:\SP>pip install -r requirements.txt
```

```
user@pc:/SP$ pip3 install -r requirements.txt
```

Při překladu na Linuxu jsem narazil na problém ve WSL (Windows Subsystem Linux), kdy se mi přidávala podpora jak *Wayland*, tak *x11* protokolů, ale moje grafické knihovny (*GLFW* + *GLEW*) se nechtěli moc přátelit s *Waylandem*; musel jsem tedy tuto podporu vypnout v CMakeLists.txt. Pokud používáte primárně *Wayland* protokol, na svém stroji zakomentujte řádky 62 a 63 v CMakeLists.txt.

4.2 Spuštění

Spustitelné exe (resp. elf) soubory se nacházejí v adresáři /bin/.

Při vlastním překladu se spustitelné soubory nacházejí v adresáři /build/Release (pro platformu Windows), resp. /build/ (pro platformu Linux).

Pozor, při vlastním překladu mohou později za běhu selhat relativní cesty! Všude v programu se počítá se spouštěním z adresáře /bin/, tedy zanoření v jedné složce oproti kořenové složce projektu!

Aplikace navíc očekává dodatečné vstupní parametry z příkazové řádky, pro nápovědu je možné použít příznak „--help“. Dostupné další parametry (příznaky) jsou:

- --file-based
- --no-lang-detect
- --lemma
- --stem

Jak již názvy napovídají, --file-based přepíná běh do tzv. file-based indexu; --no-lang-detect vypíná detekci jazyka dokumentů při indexaci; --lemma je defaultní stav a není nutné ho uvádět, ale zapíná lemmatizaci; --stem přepíná z lemmatizace na stemming.

5 Závěr

V rámci semestrální práce byl implementován komplexní IR systém, který obsahuje mnoho nadstandardních funkčností – např. proximity search, phrase search, dynamická tvorba snippetů atd. Při evaluaci byla odhalena poměrně nízká hodnota MAP (Mean Average Precision) – 9.2 %. Navíc při evaluaci je program poměrně pomalý a je možné, že je v kódu někde zanesená chyba nepředávání referencí a tedy v paměti možná vznikají zbytečné kopie.

Do budoucna by se dal IR systém vylepšit, aby dosahoval lepších výsledků při evaluaci. Navíc by se kód dal profilovat (např. zapomocí nástroje *perf*), aby se odhalilo úzké hrdlo a program se významně urychlil.