ZRÁVA K PROJEKTU

I-1: BOOLEOVSKÝ MODEL

ADAM SKLUZÁČEK

POPIS PROJEKTU

Standardní booleovský model je jedním z prvních modelů pro získávání informací, který je pro svou jednoduchost v některých oblastech dodnes využívaný. Vychází z booleovské logiky a teorie množin, kdy každý dokument i booleovský dotaz je chápán jako množina slov (termů). Za relevantní dokument je pak považován ten, který obsahuje všechna/některá/žádná (v závislosti na použitých logických operátorech) slova obsažená v dotazu.

Pro efektivní zjištění zdali je dané slovo obsaženo v dokumentu, je vytvořena binární (true/false) matice, ve které řádky představují vektor slova (tedy znázorňují, které dokumenty obsahují dané slovo) a sloupce představují vektor dokumentu (znázorňují, která slova se vyskytují v daném dokumentu). Protože jednotlivé dokumenty obsahují většinou pouze zlomek z celé slovní zásoby daného jazyka (například 1%), je tato matice extrémně řídká (obsahuje 99% nul). Pro efektivnější uložení je tedy použit invertovaný index, ve kterém jsou vektory slov obsahující seřazený seznam dokumentů, ve kterých se dané slovo vyskytuje. Jelikož jsou v invertovaném seznamu seřazeny jak slova, tak id dokumentů, lze v něm při použití správných datových struktur velice rychle vyhledávat.

Abychom omezili velikost použité slovní zásoby (a tím zlepšili časovou i paměťovou složitost vyhledávání v invertovaném indexu) projde každý dokument nejprve tzv. preprocesingem, který z dokumentu odstraní mimo jiné interpunkci a bezvýznamová slova (např. členy). Následně jsou pak slova stemmována, tak abychom z významově stejných slov v různých tvarech získali jejich přibližný základ. Po preprocesingu je slovní zásoba dokumentů zmenšená např. na třetinu.

Cílem projektu, je kromě implementace standardního booleovského modelu také implementace jednoduchého algoritmu pro vyhledávání v textu (např. Knuth-Morris-Pratt algoritmus) a následné porovnání rychlosti vyhodnocení dotazu těchto dvou přístupů viz experimentální sekce.

ZPŮSOB ŘEŠENÍ

DATOVÁ SADA

Jako datovou sadu jsem použil anglicky psané články. Datová sada obsahuje celkem 1000 článků, každý o velikosti 1kB až 8kB (celkem 2.9MB) v čistě textové podobě. Témata článků jsou velmi různorodá (přibližně 100 různých témat) např. astronomie, rybaření, New York atd.

PREPROCESING

Každý článek musí nejprve projít tzv. preprocesingem, při kterém jsou z článků odstraněny zbytečně přebývající bílé znaky, interpunkční znaménka, čísla apod. Dále jsou z článků odstraněna bezvýznamová slova – tzv. stopwords. Jedná se především o členy, předložky, přivlastňovací přídavná jména apod., která jsou zejména v angličtině velmi častá. V angličtině se tedy jedná o slova jako the, his, in atd. V projektu jsem použil seznam obsahující 174 bezvýznamových slov.

STEMMING

Během preprocesingu jsou významová slova stemmována – z každého slova je získán jeho přibližný základ za účelem zmenšení slovní zásoby při zachování stejného či podobného významu. Např. slova connect, connected, connecting mají stejný význam a pro snížení slovní zásoby bychom je chtěli všechny převést na jedno slovo connect.

Ve svém projektu jsem si pro implementaci vybral Porter stemming algorithm, který pracuje na principu transformace sufixů slov, které splňují určité podmínky. Např. slova, která obsahují sufix “ing” a zároveň obsahují vice, než dvě samohlásky jsou transformována na slova bez sufixu “ing”. Tedy slovo “skiing” je transformováno na slovo “ski”, naopak slovo “sing” toto pravidlo nezmění, protože nesplňuje podmínku alespoň dvou samohlásek. Algoritmus má celkem 64 obdobných pravidel aplikovaných postupně v pěti krocích, např. slovo generalization je postupně transformováno následujícím způsobem: generalization -> generalize -> general -> gener. Původní slovní zásobu tento stemmer redukuje přibližně na třetinu.

PARSOVÁNÍ BOOLEOVSKÉHO DOTAZU

Pro schopnost vyhodnocovat libovolné booleovské dotazy včetně uzávorkování, převádím booleovský dotaz ze standardní infixové notace do postfixové notace (reverse polish notation), ve které jsou operátory za příslušnými operandy a priority operátorů jsou jednoznačně vyjádřený samotným zápisem i bez závorek např. dotaz “a^!b” je v postfixové notaci zapsán jako “a b ! ^”. Pro převod z infixové do postfixové notace používám obodobu Shunting-yard algoritmu Edsgera Dijskry. Vyhodnocení dotazu pak probíhá standardním způsobem v závislosti na použitém vyhledávacím algoritmu (sekvenční průchod nebo invertovaný index).

SEKVENČNÍ PRŮCHOD

Pro vyhledávání sekvenčním průchodem v textu jsem naimplementoval Knuth-Morris-Pratt algoritmus. Představuje vylepšení brute force algoritmu pro hledání v textu, který v případě neshody textu a hledaného slova začíná porovnávání vždy na pozici posunuté o jedna doprava od začátku předchozího porovnávání nehledě na to, kolik porovnání při něm proběhlo. KMP algoritmus využívá tabulku prefixů a na základě předchozích porovnání může začátek příštího porovnávání posunout o více pozic, čímž je snížen celkový počet porovnání a tím zlepšená časová složitost v porovnání s brute force algoritmem.

INVERTOVANÝ INDEX

Jak už bylo v úvodu nastíněno, binární matice obsahující vektory slov v řádcích a vektory dokumentů ve sloupcích je extrémně řídká. V projektu proto používám invertovaný index, který má následující strukturu:

SLOVO POČET_DOKUMENTŮ_OBSAHUJÍCÍ_SLOVO -> SEŘAZENÝ_SEZNAM_ID_DOKUMENTŮ

Například tento zápis:

CCTV 1 -> 642
CD 13 -> 8 44 193 228 411 453 510 594 653 795 808 923 970

Reprezentuje, že slovo “cctv” je obsaženo pouze v jednom dokumentu s ID 642 a slovo “cd” je obsaženo celkem v 13 dokumentech s ID 8, 44, 193 atd.

Jelikož jsou v invertovaném indexu jak slova, tak ID dokumentů seřazeny, můžeme při použití vhodných datových struktur vyhledávat v logaritmickém (případně amortizovaně konstatním) čase. Já jsem pro uložení invertovaného indexu použil datové struktury mapa a množina (tedy červeno-černé stormy), které umožňují jak vkládání tak vyhledávání v logaritmickém čase.

POZIČNÍ INDEX

Ze struktury invertovaného indexu vyplývá, že nám poskytuje pouze informaci o výskytu slova v dokumentu, ale nevyčteme z něj kolikrát a kde se dané slovo v dokumentu nachází. K tomu nám slouží poziční index s následující strukturou:

SLOVO POČET_DOKUMENTŮ -> ID_DOKUMENTU POČET_VÝSKYTŮ : SEZNAM_POZIC

Například tento zápis:

CENSOR 1 -> 79 2 : 583 587

Reprezentuje, že slovo “censor“ je obsaženo v jednom dokumentu s ID 79, ve kterém se vyskytuje celkem dvakrát na pozicích 583 a 587 (pozice 583 znamená 583. slovo v dokumentu).

Poziční index nám tedy umožňuje dotazovat se nejen na to, který document obsahuje dané slovo, ale také kolikrát a na jakých pozicích. Díky tomu můžeme rozšířit funkčnost standardního booleovského modelu tak, aby seřadil výsledné dokumenty sestupně na základě počtu výskytů hledaného slova, nebo zvýraznil pozici hledaného slova v dokumentech.

IMPLEMENTACE

Při implementaci jsem použil programovací jazyk C++, kompilátor GCC a textový editor Atom.

KNIHOVNY

Při implementaci jsem nevyužil žádné knihovny, které by řešili některou z větších částí projektu (např. preprocesing nebo stemming). Hojně jsem však využíval standardní knihovnu C++, především kontejnery ze standardní knihovny šablon – vektor, mapu, množinu a zásobník.

Pro tvorbu webové aplikace jsem použil knihovnu Wt (web toolkit), což je open-source webové rozhraní pro tvorbu webových aplikací v C++.

Pro jednoduché procházení všech dokumentů ve složce s kolekcí dokumentů jsem použil knihovnu dirent.h.

POŽADAVKY NA BĚH

- Kompilátor jazyka C++ podporující standard C++14
- CMake verze 2.6 nebo vyšší (Požadavek knihovny Wt)
- Knihovnu C++ booost verze 1.46.1 nebo vyšší (Požadavek knihovny Wt)
- Knihovnu Wt verze 4.0.3 nebo vyšší

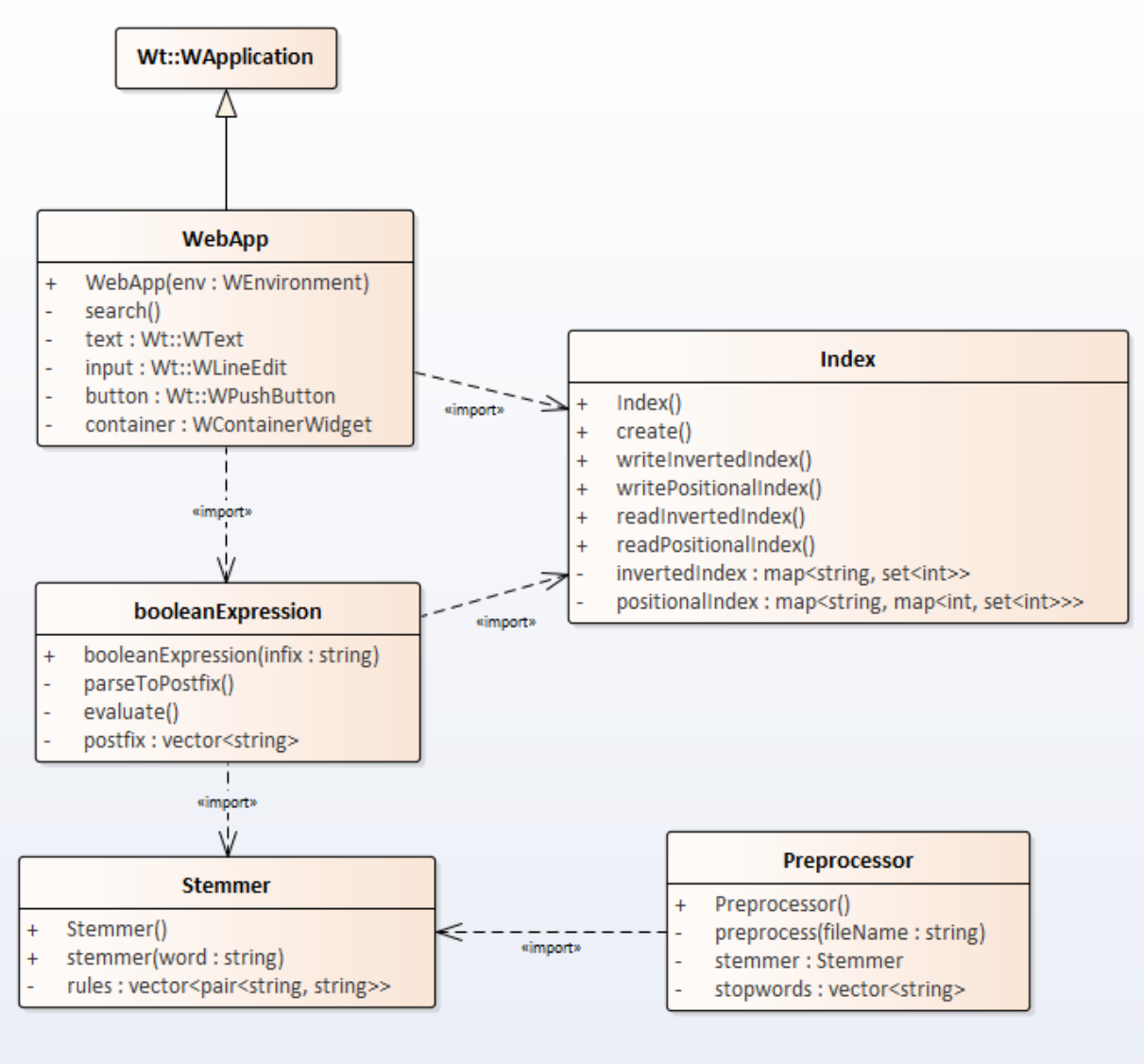
Po kompilaci (ručně nebo pomocí příkaz make) je při spouštění potřeba specifikovat, na jaké a adrese a portu se má webová aplikace spustit. Po spuštění doporučeným způsobem:

./WEBAPP –DOCROOT . –HTTP-ADDRESS 0.0.0.0 –HTTP-PORT 8080

Bude aplikace ve webovém prohlížeči dostupná na loopback adrese 127.0.0.1:8080.

DIAGRAM TŘÍD

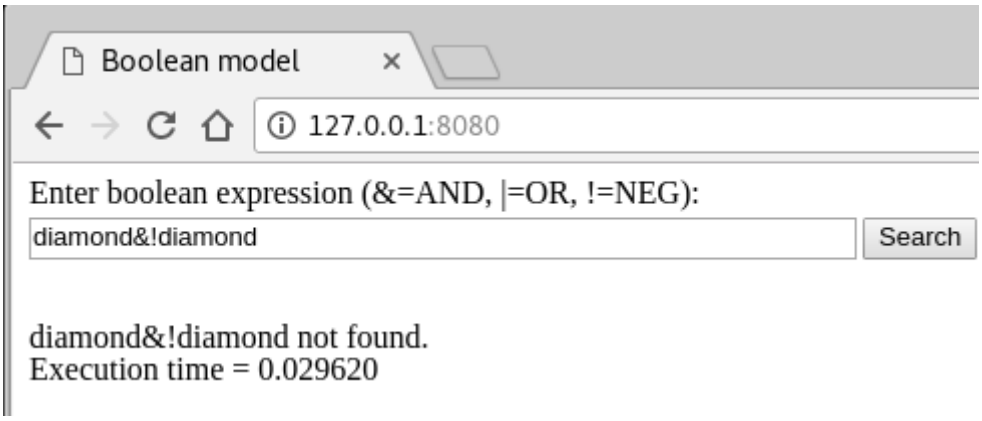
Pro lepší představu o návrhu přikládám diagram jednotlivých tříd obsahující pouze ty nejdůležitější metody a členské proměnné.



Obrázek 1: Diagram tříd

PŘÍKLAD VÝSTUPU

Obrázek 2: Ukázka aplikace



Obrázek 3: Ukázka aplikace

EXPERIMENTÁLNÍ SEKCE

Měření času probíhá pomocí funkce gettimeofday() z knihovny sys/time.h, pro větší přesnost je každé měření zopakováno 5x, následně vypočítán průměr a nakonec je rychlost vyhledávání KMP algoritmem dána do poměru s rychlostí invertovaného indexu.

MALÝ VSTUP

Vstup = "addict&man"

ČÍSLO MĚŘENÍ	INVERTOVANÝ INDEX [S]	KNUTH-MORRIS-PRATT [S]
1	0.036765	0.184415
2	0.050880	0.182237
3	0.036185	0.175777
4	0.035959	0.176595
5	0.036479	0.180632
PRŮMĚR:	**0.039887**	**0.179811**
	INVERTOVANÝ INDEX RYCHLEJŠÍ:	**4.5X**

VELKÝ VSTUP

Vstup =
"((!diamond&gold)&(!money|coin)&(christmas|art|car)&(glass|cook)&!garden"

ČÍSLO MĚŘENÍ	INVERTOVANÝ INDEX [S]	KNUTH-MORRIS-PRATT [S]
1	0.161943	1.051256
2	0.160009	1.062191
3	0.156423	1.079794
4	0.157478	1.048978
5	0.136962	1.070932
PRŮMĚR:	**0.154563**	**1.062630**
	INVERTOVANÝ INDEX RYCHLEJŠÍ O:	**6.8X**

DISKUZE

Z výsledků měření vyplývá, že pro vstup se dvěma termy dokáže invertovaný index zpracovat dotaz přibližně 4.5x rychleji než KMP algoritmus a pro větší vstup s 10 termy dokonce 6.8x rychleji. Jsem přesvědčený, že s rostoucí kolekcí dokumentů by se tento rozdíl ještě navyšoval ve prospěch invertovaného indexu, ale již na mé kolekci o velikosti 2.9MB, je tento rozdíl v rychlosti dobře patrný.

ZÁVĚR

Závěrem si myslím, že jak implementace, tak následný experiment porovnání booleovského modelu se sekvenčním průchodem (KMP algoritmem) dopadly úspěšně a splnily účel této semestrální práce. Prostor pro zlepšení vidím zejména v rozhraní webové aplikace, ale i přes tuto mírnou vadu na kráse jsem s výsledky spokojený.

ZDROJE

Datová sada:
https://archive.org/details/7000EnglishArticlesPBN

Stopwords:
https://www.ranks.nl/stopwords

Porter stemming algorithm:
https://tartarus.org/martin/PorterStemmer/def.txt

Reverse Polish notation
https://en.wikipedia.org/wiki/Reverse_Polish_notation

Shunting-yard algorithm:
https://en.wikipedia.org/wiki/Shunting-yard_algorithm

Knuth-Morris-Pratt algorithm:
https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm

Knihovna Wt:
https://www.webtoolkit.eu/wt