VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ODHALOVÁNÍ PLAGIÁTŮ

PLAGIARISM IDENTIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAKUB MENŠÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Menšík Jakub**
Obor: Informační technologie
Téma: **Odhalování plagiátů**
Plagiarism Identification
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte existující metody pro odhalování plagiátů se zaměřením na studentské práce.
2. Seznamte se s dosavadními výsledky v oblasti automatického zpracování přirozeného jazyka, které je možné použít k odhalení doslovných překladů.
3. Navrhněte a implementujte systém, který pomocí rozsáhlé lokální množiny textových dat, případně dotazů na webové vyhledávače, dokáže identifikovat plagiáty.
4. Vyhodnoťte výsledky systému na reprezentativním vzorku dat.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle domluvy s vedoucím

Pro udělení zápočtu za první semestr je požadováno:

- funkční prototyp systému

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je navrhnout a implementovat systém pro odhalování plagiátů na základě rozsáhlé množiny textových dat.

Abstract

The aim of this work is to design and implement system for plagiarism identification based on a huge dataset.

Klíčová slova

odhalování plagiátorství, n-tice, hash, indexace

Keywords

plagiarism identification, n-gram, hash, indexing

Citace

MENŠÍK, Jakub. *Odhalování plagiátů*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Smrž Pavel.

Odhalování plagiátů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Jakub Menšík
17. května 2017

Poděkování

Chtěl bych poděkovat vedoucímu panu doc. Pavlu Smržovi za odborné vedení a pomoc s prací.

Obsah

1 Úvod **3**

2 Analýza problému **4**
2.1 Druhy plagiátorství . . . 4
2.2 Možnosti vyhledávání plagiátů . . . 5
2.2.1 Vnější odhalování plagiátů . . . 5
2.2.2 Vnitřní odhalování plagiátů . . . 5
2.2.3 Vícejazyčné odhalování plagiátů . . . 6
2.3 Obcházení systémů pro odhalování plagiátů . . . 8
2.3.1 Homografy . . . 8
2.3.2 Zabránění rozpoznávaní slov . . . 8
2.3.3 Úprava textu . . . 8
2.4 Zpracování dokumentu . . . 9
2.4.1 Normalizace dokumentu . . . 9
2.4.2 Tvorba hashů . . . 9
2.5 Indexace dat . . . 11
2.5.1 B+ strom . . . 11
2.5.2 AVL strom . . . 11
2.5.3 Hašovací tabulka . . . 12

3 Použité technologie **13**
3.1 Jazyk C/C++ . . . 13
3.2 Hašovací tabulka Sparsepp . . . 13
3.3 Hašovací funkce xxHash . . . 13
3.4 Knihovna Command Line Parameters Parser . . . 14

4 Návrh **16**
4.1 Klient . . . 16
4.1.1 Předzpracování a normalizace dokumentu . . . 16
4.1.2 Výběr n-tic . . . 17
4.1.3 Hašování řetězců . . . 17
4.1.4 Vyhodnocení shody . . . 17
4.2 Server . . . 18
4.2.1 Indexace dat . . . 19
4.2.2 Vyhledávání hashů . . . 19

5 Implementace **21**
5.1 Klient . . . 21
5.1.1 Zpracování argumentů . . . 21
5.1.2 Připojení k serverům . . . 22
5.1.3 Zpracování dokumentů . . . 22
5.1.4 Komunikace se serverem . . . 22
5.1.5 Vyhodnocení a nalezení plagiátů . . . 23
5.2 Server . . . 23
5.2.1 Zpracování argumentů . . . 23
5.2.2 Navázání komunikace s klientem . . . 24
5.2.3 Komunikace s klientem . . . 24
5.2.4 Ukončení serveru . . . 24

6 Testování a vyhodnocení **25**
6.1 Datové sady pro testování systému . . . 25
6.1.1 Wikipedia dumps . . . 25
6.1.2 Data z CommonCrawlu . . . 25
6.1.3 PAN korpus . . . 25
6.1.4 Data ze Seznamu . . . 26
6.2 Porovnání se systémem Theses.cz . . . 26
6.3 Testování systému . . . 26
6.4 Zhodnocení systému . . . 27

7 Závěr **28**

Literatura **29**

Přílohy **31**

A Obsah přiloženého DVD **32**

B Manuál **33**
B.1 Instalace . . . 33
B.2 Spuštění . . . 33

C Plakát **35**

Kapitola 1

Úvod

S rychle se zvyšujícím množstvím textových dokumentů na internetu, snadnou dostupností a možností kopírování těchto dokumentů, roste důležitost odhalování plagiátorství. Odhalení plagiátů však není triviální záležitostí a pro efektivní vyhledávání je nutné použití algoritmů a datových struktur, které dokáží pracovat s velkým množstvím dat.

Existuje více možností, jak porovnávat dokumenty a hledat duplicity. Dají se hledat přesné kopie, sémanticky podobné dokumenty, dokumenty se mohou rozdělit na několik částí a tyto části prohledat každou zvlášť. V této práci si ukážeme porovnávání dokumentů na základě rozdělení textu do odstavců a následným vyhodnocením těchto odstavců.

Nejprve se v této práci podíváme na analýzu problému v kapitole 2, vysvětlíme si, co pojem „plagiát“ znamená, ukážeme si různé druhy plagiátorství, dále si ukážeme různé přístupy k odhalování plagiátů. Podíváme se na to, jakými způsoby se dají systémy pro odhalování plagiátů obcházet, rozebereme si možnosti, jakými lze normalizovat dokument. Nakonec se podíváme na možné datové struktury, do kterých můžeme indexovat data.

V kapitole 3 si představíme použité technologie a knihovny. Řekneme si, proč je systém napsán právě v jazyce C, dále si popíšeme použitou hašovací tabulku a hašovací funkci. Nakonec si přiblížíme použité knihovny.

Kapitola 4 se podrobněji zabývá návrhem systému. Rozdělíme si systém na programy typu „server“ a programy typu „klient“. Ke každému programu si řekneme, jak funguje a jaké má úkoly.

Kapitola 5 nese název implementace. Podíváme se na to, jakým způsobem jsou programy server a klient implementovány a jak mezi sebou komunikují.

V kapitole 6 si představíme datové sady, na kterých byl systém testován, porovnáme nás systém s jiným systémy pro odhalování plagiátů a zhodnotíme celý systém.

Kapitola 2

Analýza problému

Plagiátorství můžeme podle [6] definovat jako znovupoužití nápadu, práce, procesu nebo výsledku jiného autora bez uvedení odkazu na originální zdroj. V této kapitole se zaměříme na druhy plagiátorství a možnosti, jak se dají vyhledávat plagiáty. Dále se podíváme na to, jak se nejčastěji obcházejí systémy pro odhalování plagiátů a na zpracování dokumentu. Nakonec si představíme možné datové struktury pro indexaci dat.

2.1 Druhy plagiátorství

Pojem plagiátorství nezahrnuje pouze představení díla jiného autora jako svého vlastního, pod pojmem plagiátorství si můžeme představit:

- přeložení textu z cizího jazyka
- kopírování nápadu
- koupení práce a vydávání za svou vlastní
- znovupoužití vlastní práce
- kopírování textu bez citování autora

Množství moderních technologií pro snadné překladání jazyků otevírá možnosti pro *překládaní cizích textů* a vydávání za své vlastní bez uvedení zdroje, neboli *cross language plagiarism* (CLP). Odhalování takových plagiátů není jednoduché, přesto existuje několik metod, které tento problém řeší, viz 2.2.3.[10]

Pokud jde o *kopírování cizích nápadů*, autor se snaží použít jiná slova pro prezentování cizí práce. Odhalit tento způsob plagiátorství se dá pomocí sémantické analýzy textu.

Čím dál častěji se setkáváme s možností *zakoupení hotové práce*, a to hlavně díky vznikajícím stránkám, které nabízejí vytvoření práce za finanční odměnu.

Ač se to může zdát na první pohled zvláštní, i *znovupoužití vlastní práce* může být bráno jako plagiátorství, a to v momentě, kdy autor vydává práci jako nový výtvor.[5]

Nás bude zajímat hlavně poslední možnost, a to *kopírování textu* či jeho úpravy, jako například změna pořadí slov, převzetí části textu, nebo dokonce převzetí celé práce. Takovým dokumentům se říká *near-duplicates*.

2.2 Možnosti vyhledávání plagiátů

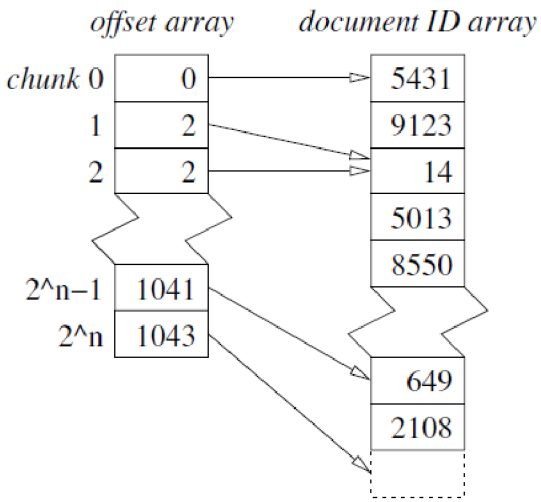
Je celá řada možností, jak vyhledávat plagiáty. Tyto možnosti se dají rozdělit do dvou skupin, a to externí odhalování plagiátů a vnitřní odhalování plagiátů. V této sekci si obě skupiny přiblížíme a dále rozdělíme. Navíc si podrobněji popíšeme odhalování plagiátu ve vícejazyčných dokumentech.

2.2.1 Vnější odhalování plagiátů

Výzkum v oblasti automatického odhalování plagiátů se zaměřuje zejména na vnější odhalování plagiátů. To spočívá v porovnávání vstupních dokumentů s velkou kolekcí lokálních dat. Tato kolekce dat je ve většině případů tvořena otisky (z anglického *fingerprints*) a cestami k souborům, ve kterých se otisky vyskytují.

Data jsou indexována do tzv. forward indexu nebo do tzv. inverted indexu. *Forward index* ukládá ke každému souboru seznam otisků, které se v něm nacházejí. *Inverted index* naopak ukládá ke každému otisku seznam souborů, ve kterých se otisk nachází, viz obrázek 2.1. My budeme využívat inverted index.

Do vnějšího odhalování plagiátů patří také porovnávání dvojice dokumentů, to je však neefektivní řešení pro velkou kolekci dokumentů.



Obrázek 2.1: Datová struktura pro inverted index, převzato z [5].

2.2.2 Vnitřní odhalování plagiátů

Hlavní myšlenkou vnitřního odhalování plagiátů je najít části textu, které nezapadají svým stylem do zbytku dokumentu. Systémy tedy nejsou závislé na velké kolekci lokálních dat. Tyto systémy však vychází z jedné zásadní podmínky, a to že autor práce napsal většinu textu. Principem vnitřního odhalování plagiátů je rozdělení textu na menší segmenty (například věty) a na základě těchto segmentů se vytvoří funkce, která popisuje autorův styl. Následně se v této funkci hledají kritické hodnoty, které detekují plagiáty.[7]

2.2.3 Vícejazyčné odhalování plagiátů

V tomto případě plagiátorství je zdrojový materiál přeložen z jednoho jazyka do druhého a vydáván jako vlastní, bez uvedení citace. Odhalení tohoto typu plagátorství je velmi obtížné. Jedním z hlavních problémů je nahrazení slov jejich synonymy, neboli slovy stejného významu. K tomu se může podle [4] použít EuroWordNet Thesaurus [1], což je mnohojazyčná databáze slov a jejich vztahů pro většinu evropských jazyků, například angličtinu, italštinu, španělštinu, dánštinu, němčinu, francouzštinu, češtinu a další. Obsahuje množiny synonym, tzv. *synsety*, a vztahy mezi nimi. Každému synsetu je přiřazen unikátní index. Jazyky jsou propojeny tak, že každý synset v jednom jazyce má stejný index jako synset v jazyce druhém. Aby toto mohlo fungovat, musí být slova převedena na jejich základní gramatický tvar. K tomu slouží lemmatizace, o které bude řeč v sekci 2.4.1. Celý tento proces je znázorněn na obrázku 2.2. U jazyků s velkou ohybností slov je lemmatizace obtížnější. Po lemmatizaci je vhodné odstranit „stopslova“, což je v oblasti počítačového zpracování přirozeného jazyka označení pro slova, které nenesou žádný sémantický význam, například spojky nebo předložky. Zároveň tím zmenšíme množství dat pro porovnávání.

Jednou z metod, která se zabývá odhalováním plagiátů ve vícejazyčných korpusech, je MLPlag metoda [4]. Tato metoda využívá vyhodnocování na základě jednotlivých slov a jejich umístění v dokumentu. Pokud si představíme situaci, kde je nějaké slovo či jeho synonymum na začátku jednoho dokumentu a na konci jiného dokumentu, nelze toto slovo brát jako opsané. Tato metoda definuje množinu pozic $C_w(R,S)$ ze slova w podle následujícího vzorce:

$$\begin{aligned} C_w(R,S) = \{a :\ & a \in <1, N_R>,\ b \in <1, N_S>, \\ & w = w_{R,a},\ w = w_{S,b},\ |\frac{a}{N_R} - \frac{b}{N_S}| < \omega\}, \end{aligned} \tag{2.1}$$

kde slovo w z dokumentu R je opsáno z dokumentu S. V tomto vzorci značíme slovo w na pozici a z dokumentu R jako $w_{R,a}$ a podobně slovo w na pozici b z dokumentu S jako $w_{S,b}$. Konstanta N_S je celkový počet slov v dokumentu S. Konstanta ω reprezentuje relativní velikost okna, ve kterém bereme dvě slova $w_{R,a}$ a $w_{S,b}$ jako dostatečně blízká.

Aby se odstranil vliv různé velikosti dokumentů, normalizujeme pozici slov do intervalu $<0,1>$, kde nula znamená, že je slovo na začátku a jednička znamená, že je na konci. Například pro relativní velikost okna $\omega = 20\,\%$, slovo $w_{R,a}$ na pozici $a/N_R = 0,7$ je označeno jako plagiát pouze pokud je stejné slovo $w_{S,b}$ na pozici $b/N_S \in <0{,}5; 0{,}9>$.

Nyní můžeme definovat vzorec pro frekvenci výskytu opsaného slova w jako $P_w(R,S)$ jako:

$$P_w(R,S) = |C_w(R,S)| \cdot |C_w(S,R)|, \tag{2.2}$$

kde $|C_w(R,S)|$ reprezentuje počet pozic opsaných slov w z dokumentu R v dokumentu S. Dále definujeme frekvenci výskytu slova w v dokumentu R jako $F_w(R)$. Pomocí těchto dvou definic jsou vytvořeny dvě míry podobnosti, a to symetrická podobnostní míra $\text{MLPlag}_{\text{SYM}}$ a asymetrická podobnostní míra $\text{MLPlag}_{\text{ASYM}}$. Symetrická míra podobnosti mezi dokumenty R a S je definována jako:

$$sim(R,S) = \frac{\sum\limits_{w \in R \cap S} \alpha_w^2 \cdot P_w(R,S)}{\sqrt{\sum\limits_{w \in R} \alpha_w^2 \cdot F_w^2(R) \cdot \sum\limits_{w \in S} \alpha_w^2 \cdot F_w^2(S)}} \tag{2.3}$$

Tento vzorec vyjadřuje symetrickou míru mezi dvěma dokumenty, kde α_w je váha spojená s výskytem slova w. Výsledek je v intervalu $<0,1>$.

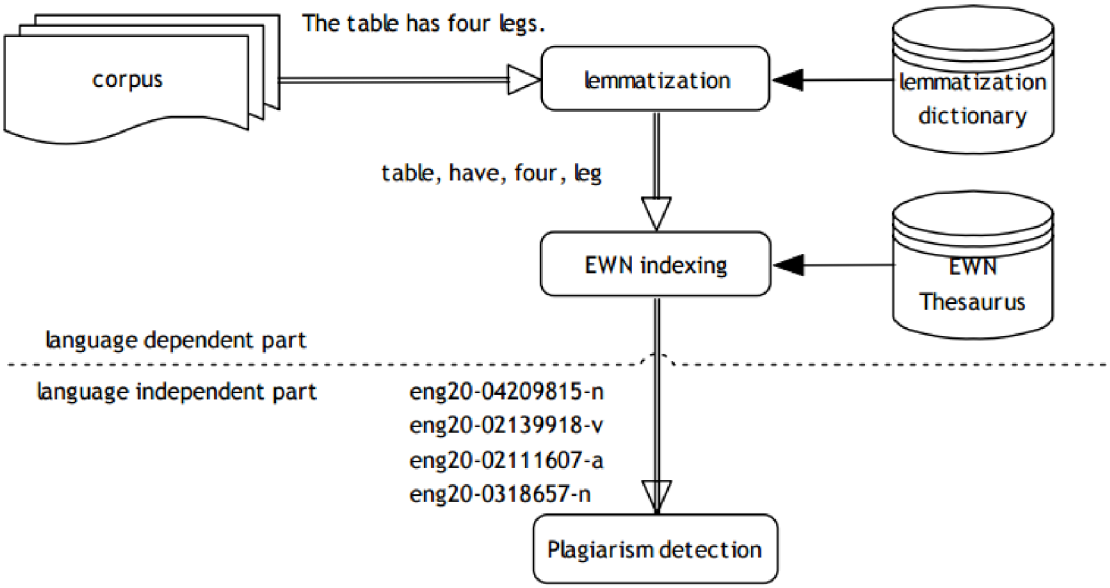
U asymetrické míry podobnosti musíme definovat podmnožinu míry dokumentu R jako podmnožinu dokumentu S, rovnice vypadá následovně:

$$subset(R,S) = \frac{\sum\limits_{w \in R \cap S} \alpha_w^2 \cdot P_w(R,S)}{\sum\limits_{w \in R} \alpha_w^2 \cdot F_w^2(R)}, \tag{2.4}$$

kde α_w je váha spojená s výskytem slova w. Nyní můžeme definovat míru podobnosti jako maximum obou asymetrických mír podle vzorce:

$$sim(R,S) = \max\{subset(R,S), subset(S,R)\} \tag{2.5}$$

Pokud je podobnost $sim(R,S)$ větší než jedna, nastavíme ji na 1. Tím zachováme podobnost v intervalu $<0,1>$.[4][10]



Obrázek 2.2: Předzpracování vícejazyčného korpusu, rozdělené do dvou částí - lemmatizace a EWN indexování, převzato z [4].

2.3 Obcházení systémů pro odhalování plagiátů

Existuje spousta způsobů, jak lze obejít systémy pro odhalování plagiátů. Tato podkapitola rozebírá některé způsoby a nabízí možné řešení, jak jim lze zabránit.

2.3.1 Homografy

V Unicode můžeme najít značné množství znaků, které pocházejí z různých druhů písma, mají odlišné escape sekvence, přesto vypadají na první pohled stejně nebo dokonce naprosto totožně, viz 2.1. Takové znaky nazýváme homografy. Tento problém se netýká pouze samostatných znaků, ale také předzpracovaných zkratek a slov z více znaků.

Jednou z možností, jak homografy odhalit, je zjistit, zda všechny znaky v dané práci jsou napsány stejným druhem písma. Další možností by bylo převést všechny znaky do stejného písma.[5]

Písmo latinkové	Escape sekvence	Písmo cyrilické	Escape sekvence
a	U+0061	а	U+0430
c	U+0063	с	U+0441
e	U+0065	е	U+0435
p	U+0070	р	U+0440

Tabulka 2.1: Tabulka na oko stejně vypadajících znaků v písmu latinkovém a písmu cyrilickém.

2.3.2 Zabránění rozpoznávaní slov

V dnešní době obsahuje málokterá práce pouze prostý text, většinou se odevzdávají soubory typu PDF (z anglického *Portable Document Format*). U takových souborů je potřeba nejprve vyextrahovat prostý text, a až pak ho zpracovat. Problém však nastává v případě, že někdo místo textu použije například oskenované stránky, a tím zabrání extrakci textu a odhalení plagiátu.

Dalším problémem mohou být znaky psány bílou barvou místo mezer. Při extrakci tak vznikají dlouhá slova, která jsou jen težko porovnatelná s originálem.

V těchto případech je nejlepším řešením použití OCR neboli optického rozpoznávání znaků (z anglického *Optical Character Recognition*). Tato technologie umožňuje převod dokumentu z digitální obrazové do textové podoby, se kterou lze následně pracovat jako s prostým textem. Tato metoda je založena na porovnávání rozložení bodů předlohy s jednotlivými písmeny z databáze [3].[5]

2.3.3 Úprava textu

Asi nejvíce používanou metodou, jak obejít systémy pro odhalování plagiátů, je změna pořadí slov a nahrazení slov synonymy.

Pokud jde o změnu pořadí slov, nejlepším protiopatřením je při vytváření n-tic nebrat ohled na pořadí slov. Pokud však autor nahradí slova synonymy, je odhalení plagiátů obtížnější. Nejvhodnějším řešením je vynechání slova nebo více slov v rámci n-tice, viz 4.1.2. Tato metoda je však náročna na paměť.[5]

2.4 Zpracování dokumentu

V této sekci si popíšeme, jak vypadá zpracování dokumentů od zpracování textu až po vytvoření hashe. Hashe se vytvářejí z n-tic slov a slouží hlavně pro zmenšení paměťových nároků na systém.

2.4.1 Normalizace dokumentu

Normalizace dokumentu je nezbytnou části celého systému. Jelikož jen zřídka dostaneme soubor s prostým textem, musíme se umět vypořádat se soubory, které obsahují různé značky. Mezi takové soubory patří například PDF či HTML soubory. Pod značkami si můžeme představit například HTML tagy `<doc>` nebo `<p>`. Systém by měl být schopen sám zjistit, o jaký typ souboru se jedná, a podle toho odstranit nechtěné značky a pracovat pouze s prostým textem.

Pokud nám odstranění značek nestačí a chceme text ještě více normalizovat, můžeme text lemmatizovat. Lemmatizace znamená převedení slova na základní gramatický tvar, například slovo „počítačích“ převedeme na „počítač“. Některá slova však mají více významů, což může být pro lemmatizaci problém. Stematizace zase využívá odstranění předpon a přípon pro nalezení kořene slova. Dalším krokem normalizace by mohlo být odstranění interpunkčních znamének a zmenšení velkých písmen na malá. Tabulka synonym zase nabízí nahrazování slov za jejich synonyma. Stopslova, která nemají velkou váhu, můžeme při zpracování přeskočit. V této práci odstraníme pouze značky, interpunkční znaménka a zmenšíme velká písmena na malá. Žádný další normalizační krok nevyužijeme a budeme pokračovat s prostým textem.

2.4.2 Tvorba hashů

Dříve jsme zmínili dva pojmy, jedním z nich byl pojem otisk a druhým byl pojem hash. Na první pohled to může vypadat, že pojmy znamenají to samé, ale je mezi nimi malý rozdíl. Když se bavíme o otiscích, bavíme se na úrovni dokumentů, ale když se bavíme o hashích, bavíme se o konkrétních číslech, které jsou výstupem z hašovací funkce.

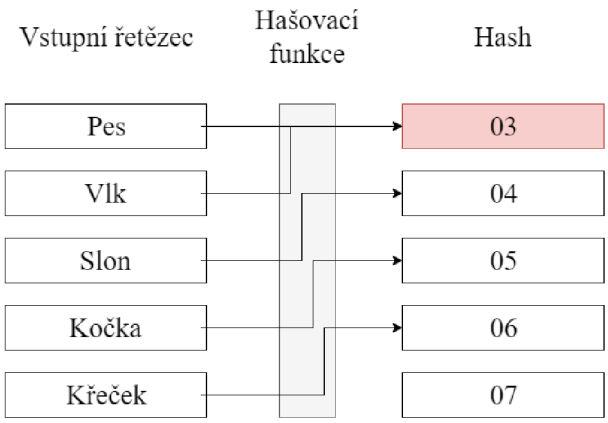
Existuje mnoho algoritmů pro tvorbu hashů, liší se v použité hašovací funkci, délce řetězce, počtu extrahovaných řetězců a metodě pro výběr řetězce. V této sekci to dál podrobněji rozebereme.[14]

Hašovací funkce se používá ke generování hashů, což jsou číselné hodnoty, které reprezentují řetězce. Hašovací funkce mají dva hlavní parametry, které jsou pro nás důležité. Je to rychlost a odolnost vůči kolizi. Kolize nastává v momentě, kdy pro dva různé vstupy nastane stejný výstup, viz obrázek 2.3. Takové situace by vedly ke špatnému vyhodnocení plagiátů.

Pokud bychom chtěli hašovací funkce rozdělit do skupin, pak by to byly kryptografické a nekryptografické hašovací funkce. Kryptografické hašovací funkce nabízejí určitou bezpečnostní garanci. Je velmi obtížné najít dvě vstupní hodnoty, u kterých by došlo ke kolizi. Zároveň je téměř nemožné zjistit předlohu, tzn. že funkce jsou jednosměrné. Daň za větší bezpečnost je menší rychlost těchto funkcí. Mezi nejoblíbenější hašovací funkce patří MD5, která se používá například pro kontrolu integrity souborů nebo ukládání hesel. Další oblíbenou funkcí je SHA1, kterou využívá například systém Git pro zjištění, zda došlo ke změně dat nebo ne. Obě tyto funkce, MD5 i SHA1, byly v minulosti prolomeny a nedají se považovat za bezpečné. Nekryptografické hašovací funkce se snaží vyhnout kolizím, ale bezpečnost není na takové úrovni, jako u kryptografických funkcí. Jejich velkým plusem

je však rychlost. Mezi nejlepší nekryptografické hašovací funkce patří xxHash, která je použita i v tomto systému, viz 3.3. Další oblíbenou funkcí je CityHash. Porovnání několika hašovacích funkcí naleznete v obrázku 3.4.

Je taky důležité, jak velké hashe chceme vytvořit. Naše použitá funkce xxHash nabízí 32 bitové a 64 bitové hashe, funkce CityHash nabízí 32, 64, 128 a 256 bitové hashe. My si v této práci vystačíme s 64 bitovými hashi.



Obrázek 2.3: Jednoduchá hašovací funkce, která vytvoří hash na základě délky řetězce. V hashi s hodnotou 03 nastává kolize, kdy vstupní řetězec vlk i pes dostávají po hašování stejnou hodnotu.

Délka řetězce k hašování nám ovlivní spoustu věcí. Čím kratší řetězce budeme mít, tím je větší šance nalezení falešné shody. Kdybychom hašovali například pouze dvojice slov, našli bychom podobnost téměř u všech dokumentů. Čím delší řetězce budeme mít, tím je šance nalezení plagiátu menší. Je potřeba najít ideální střed. Výzkumy se v tomhle ohledu liší. Některé tvrdí, že nejlepší jsou řetězce ze tří až pěti slov, jiné zase vyzdvihují řetězce z šesti až sedmi slov. V naší práci se osvědčily řetězce ze sedmi slov.

Počet řetězců je počet extrahovaných řetězců z dokumentu pro vytvoření otisku. Jsou dvě možnosti používání počtu řetězců. První používá předem určený počet řetězců, avšak většina systémů používá proměnný počet řetězců, který se rovná počtu řetězců ve zpracovávaném dokumentu.

Metody pro výběr řetězců nám udávají, které řetězce budeme hašovat. Existuje řada metod pro výběr řetězců. Jednou z nich je rozdělení dokumentu na náhodné podřetězce fixní délky, ale tím nemusíme vybrat důležité části, a proto tato metoda není vhodná. Další metodou je rozdělení dokumentu na věty, ale věty mají většinou velmi odlišnou délku, což může vést ke špatným výsledkům. Třetí metodou je rozdělení dokumentu na n-tice po sobě jdoucích slov, což je nejvíce využívaná metoda, kterou budeme používat i my. Spočívá v tom, že tvoříme hashe pro po sobě jdoucí slova, které se vzájemně překrývají, viz 4.1.2. Další zajímavou metodou je tvorba řetězců v závislosti na frekvenci jejich výskytu. Řetězcům, které se, vyskytují méně se přidělí větší hodnota, a naopak řetězcům, které se vyskytují často, se přidělí malá hodnota. Tato metoda se používá spíše při hledání sémanticky podobných textů.[14]

2.5 Indexace dat

Pro ukládání hashů potřebujeme využít takovou datovou strukturu, která umožňuje ukládání dat jako páru <klíč, hodnota>. Takových datových struktur existuje velké množství a je důležité, abychom vybrali tu, která je pro nás nejlepší. Chceme vybrat především takové datové struktury, které efektivně pracují s pamětí a dokáží rychle vkládat a hledat prvky.

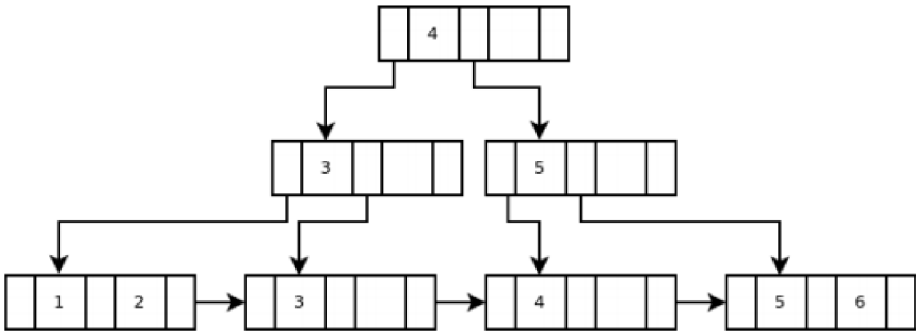
Nejjednodušší datovou strukturou je pole párů <klíč, hodnota>. Taková struktura je však nevyhovující, protože v nejhorším případě bychom pro nalezení prvku museli projít celé pole. V této sekci se podíváme na složitější datové struktury, které by mohly být v našem systému použity.[12]

2.5.1 B+ strom

B+ strom je stromová struktura, která vychází z B-stromu. Hlavním rozdílem je, že B+ strom má hodnoty uložené pouze v listech. Každý uzel stromu obsahuje pole klíčů a pole ukazatelů na další uzel, viz 2.4. Požadavky na B+ strom mohou být shrnuty do čtyř bodů:

- kořen stromu má nejvíc N potomků
- každý uzel kromě kořenu má nejvíc N a nejméně $N/2$ potomků
- data jsou uložena pouze v listech
- všechny listy mají stejnou úroveň, jsou ve stejné hloubce

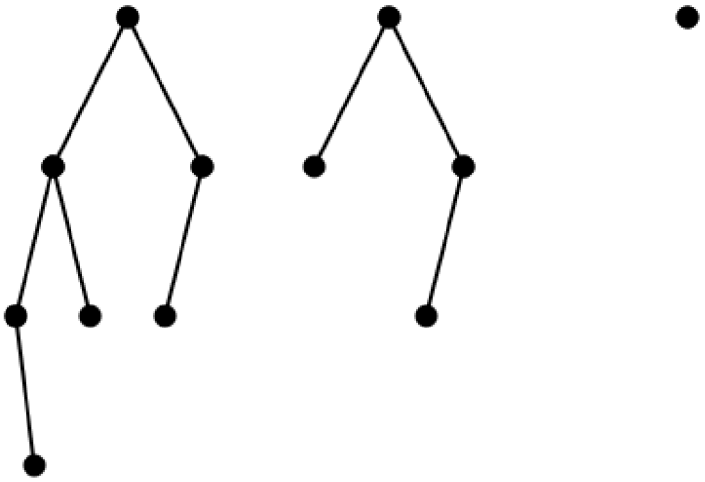
Tato struktura je vždy vyvážená a vyhledávání v ní má logaritmickou složitost.[9][12]



Obrázek 2.4: Ukázka B+ stromu, převzato z [9].

2.5.2 AVL strom

Pokud po stromu nebudeme vyžadovat, aby byl vyvážený, ale budeme vyžadovat, aby se u každého vrcholu lišily jejich hloubky o jedna, vznikne nám AVL strom. Každý vyvážený strom je zároveň také AVL stromem. Hloubka AVL stromu je logaritmická. Příklad AVL stromu na obrázku 2.5.[8]



Obrázek 2.5: Příklady jednoduchých AVL stromů.

2.5.3 Hašovací tabulka

Hašovací tabulka je taková struktura, která spojuje klíč s potřebnou hodnotou. Klíč je hodnota, která je vypočítaná hašovací funkcí. Rozsah hašovací funkce určuje velikost hašovací tabulky. Do hašovací tabulky je možné dávat přímo hodnoty nebo ukazatele na hodnoty. Hašovací tabulka je určena především pro rychlé vyhledávání. Při vhodně zvolené velikosti tabulky a při použití správné hašovací funkce má algoritmus složitost shora omezenou na O(1).[9][12]

Z výše jmenovaných datových struktur je pro náš systém hašovací tabulka nejvíce vyhovující.

Kapitola 3

Použité technologie

Tato kapitola se zaměřuje na použité technologie, knihovny, jejich popis a vysvětluje v čem je pro tuto práci výhodná.

3.1 Jazyk C/C++

Systém byl nejdříve vyvíjen v jazyce Perl, ale vznikaly komplikace při zpracování větších dat. Následoval jazyk Python, ale ani tento jazyk nevyřešil problém s paměťovou náročností. Nakonec se systém implementoval v jazyce C[1] s využitím některých C++ knihoven, především pro zjednodušení práce s textovými řetězci a ukládání většího množství dat do datových struktur. Po přechodu na jazyk C/C++ se výrazně zmenšily paměťové nároky.

3.2 Hašovací tabulka Sparsepp

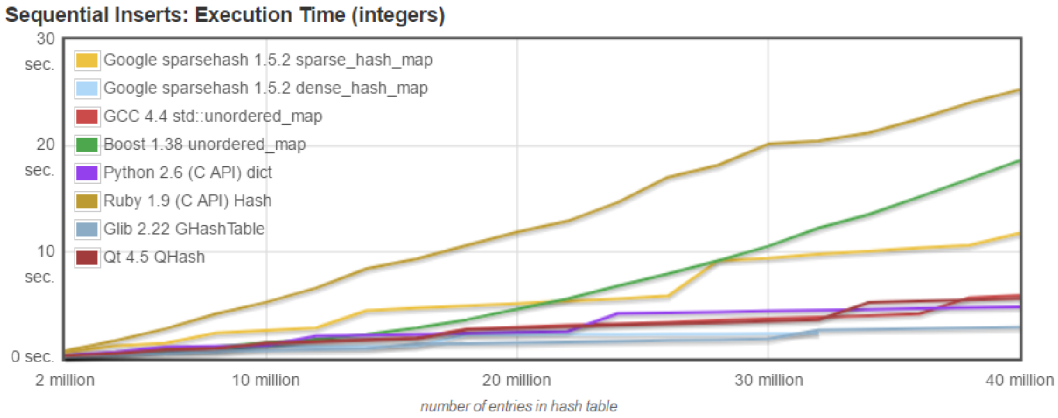
Pro náš systém potřebujeme použít hašovací tabulku, která má velmi nízkou paměťovou náročnost. Sparsepp[2] je rychlá hašovací tabulka, která pracuje velmi efektivně s pamětí. Vychází z hašovací tabulky Sparsehash[3]. Porovnání hašovacích tabulek najdeme na obrázcích 3.1, 3.2 a 3.3. Naše hašovací tabulka Sparsepp tam sice není, můžeme se však podívat, jak si vede Sparsehash, ze které Sparsepp vychází, a kterou vylepšuje.

Potřebujeme ukládat dva typy dat. Musíme ukládat hashe a k nim seznam identifikátorů dokumentů, ve kterých se hash nachází, a zároveň musíme ukládat identifikátory dokumentů a k nim absolutní cesty, kterými se k dokumentům dostaneme.

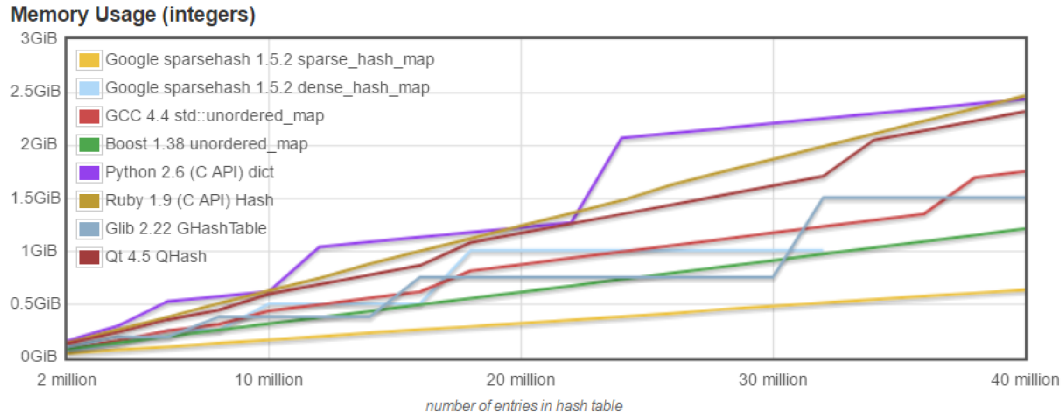
3.3 Hašovací funkce xxHash

Pokud chceme v našem systému dosáhnout co nejlepších výsledků, je velmi důležitá nízká míra kolize u hašovací funkce. Nesmíme zapomenout ani na rychlost. Hašovací funkce xxHash[4] splňuje oba požadavky. Jedná se o nekryptografickou hašovací funkci, která nabízí tvorbu 32 bitových a 64 bitových hashů. Abychom měli co nejmenší míru kolize, budeme vytvářet 64 bitové hashe. Porovnání několika hašovacích funkcí v grafu 3.4.

[1] Více o jazyce C a C++ na: http://www.cprogramming.com/
[2] Knihovna Sparsepp dostupná z: https://github.com/greg7mdp/sparsepp
[3] Knihovna Sparsehash dostupná z: https://github.com/sparsehash/sparsehash
[4] Knihovna xxHash dostupná z: https://cyan4973.github.io/xxHash/



Obrázek 3.1: Porovnání hašovacích tabulek při sekvenčním vkládání čísel do hašovací tabulky, převzato z [13].

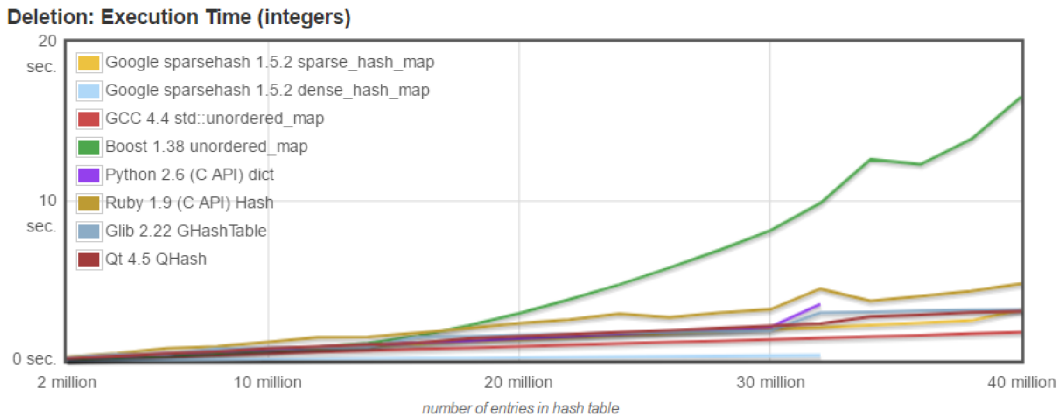


Obrázek 3.2: Porovnání využití paměti hašovacích tabulek při ukládání velkého množství čísel, převzato z [13].

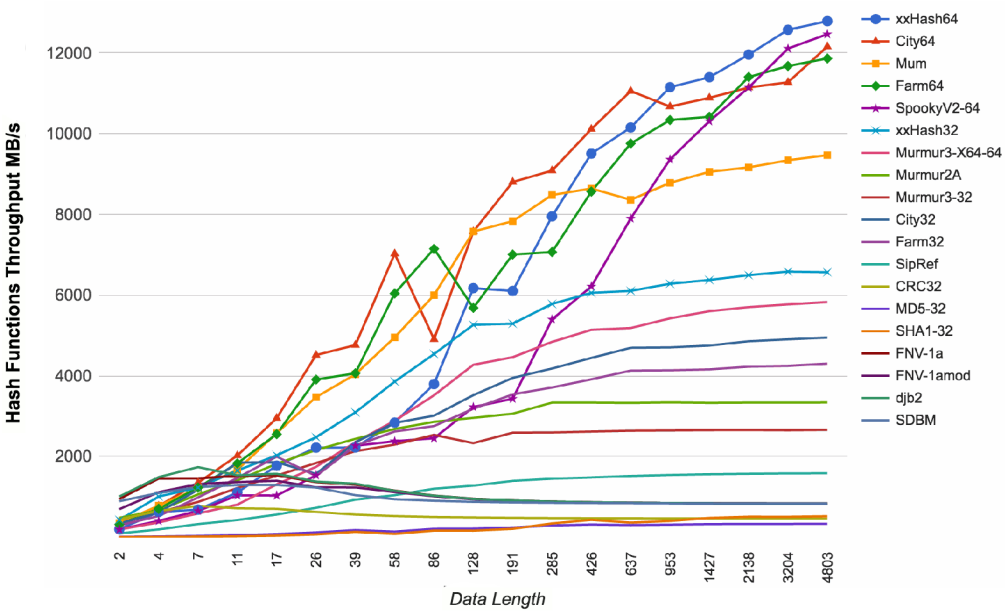
3.4 Knihovna Command Line Parameters Parser

Command Line Parameters Parser, zkráceně CLPP[5], je knihovna, která usnadňuje práci s parametry zadanými na vstupu. Knihovna je v jazyce C++ a vychází částečně z Boost knihovny.

[5]Knihovna CLPP dostupná z: https://github.com/makerbot/clp-parser



Obrázek 3.3: Porovnání hašovacích tabulek při mazání čísel z hašovací tabulky, převzato z [13].



Obrázek 3.4: Porovnání několika hašovacích funkcí, převzato z [11].

Kapitola 4

Návrh

V této kapitole si ukážeme návrh výsledného systému, který musí být schopen odhalit plagiáty na základě rozsáhlé množiny textových dat. Takový systém musí být schopen nejen plagiáty vyhledávat, ale také indexovat všechny dokumenty, které dostane na svém vstupu. Většina takových systémů vychází z tzv. *chunking algoritmu*, viz algoritmus 1, který je převzat z [5]. Indexováním nových dokumentů se bude rozšiřovat množina textových dat pro porovnávání.

V návrhu budeme vycházet z kapitoly 2, kde jsme si ukázali možné přístupy k odhalování plagiátů. Náš systém by měl tedy zvládnout zpracovat dokumenty, indexovat je a odhalit plagiáty, viz obrázek 4.1. Splnit všechny tyto úkoly by bylo pro jeden proces náročné, proto si rozdělíme procesy na server a klient.

Server-klient je síťová architektura, kde spolu komunikují dva programy přes počítačovou síť. Klient většinou iniciuje komunikaci tím, že žádá server o nějaké služby. V našem systému bude klient zpracovávat a normalizovat dokumenty, bude vytvářet hashe a následně se bude dotazovat serveru, zda už takový hash obsahuje. Server bude hashe uchovávat, indexovat a dávat klientovi informace o hashích. Klient nakonec zpracuje informace získáné ze serveru, vyhodnotí je a najde plagiáty.

4.1 Klient

Nejprve se podíváme na návrh klienta. Budeme vycházet hlavně ze sekce 2.4.1, kde jsme si rozebrali zpracování dokumentu, což je jeden z hlavních úkolů klienta. Na ukázkové větě „Lorem ipsum dolor sit amet, consectetuer adipiscing elit nulla, est aenean placerat.“ si ukážeme zpracování textu od normalizace až po vytvoření hashe. Dalším důležitým úkolem je vyhodnocení výsledků.

4.1.1 Předzpracování a normalizace dokumentu

Na začátku běhu programu máme kolekci dokumentů. Tato kolekce může obsahovat různé typy souborů a je nutné s každým pracovat jinak, abychom získali prostý text, což je cílem této fáze. Tento systém dokáže pracovat se soubory typu PDF, HTML a se soubory, které neobsahují žádné speciální značky.

4.1.2 Výběr n-tic

Důležitou věcí je určit si délku n-tic. Po testování a porovnávání výsledků jsme zvolili délku n-tice na sedm. Ukažme si na ukázkové větě jednotlivé kroky výběru n-tic. Naše věta vypadá po normalizaci následovně:

lorem ipsum dolor sit amet consectetuer adipiscing elit nulla est aenean placerat

V našem systému vybíráme pouze slova delší než tři znaky. Zbavíme se tak předložek, spojek a dalších krátkých slov, čímž zvyšujeme šanci na odhalení plagiátorství založeného na vložení slov do původního textu. Z naší věty nyní můžeme vytvořit všechny možné sedmice. Získáme:

lorem ipsum dolor amet consectetuer adipiscing elit
ipsum dolor amet consectetuer adipiscing elit nulla
dolor amet consectetuer adipiscing elit nulla aenean
amet consectetuer adipiscing elit nulla aenean placerat

Abychom zvýšili odolnost navrženého systému proti přeskupování slov, seřadíme slova abecedně. Dále vložíme mezi slova separátor a vytvoříme řetězce, které jsou připraveny k hašování. Výsledné řetězce:

adipiscing-amet-consectetuer-dolor-elit-ipsum-lorem
adipiscing-amet-consectetuer-dolor-elit-ipsum-nulla
adipiscing-aenean-amet-consectetuer-dolor-elit-nulla
adipiscing-aenean-amet-consectetuer-elit-nulla-placerat

4.1.3 Hašování řetězců

Jak už bylo zmíněno v sekci 3.3, v našem systému používáme hašovací funkci xxHash, která vytváří 64 bitové hashe. Řetězce z našeho příkladu po hašování vypadají následovně:

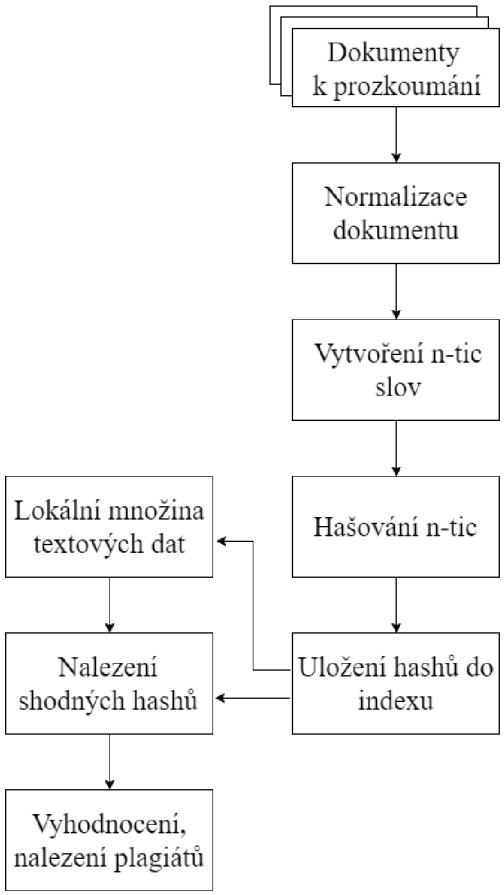
68f44d87228528c6
45e53ff75acbe4b2
0eec711c5326165c
62a5afd678480e01

4.1.4 Vyhodnocení shody

Hashe vytváříme postupně. Když je hash vytvořený, odešle se na server, který hash vyhodnotí a pokud najde shodu, vrátí absolutní cesty k souborům, ve kterých se hash vyskytuje. Jakmile klient dostane informace o všech hashích ve zkoumaném odstavci, vypočítá shodu s pro každý soubor, který obsahuje stejné hashe, podle vzorce:

$$s = \frac{n_{R,S}}{N_R}, \tag{4.1}$$

kde R je zkoumaný dokument, S je dokument z lokální množiny dat, $n_{R,S}$ je počet hashů, které má zkoumaný odstavec z dokumentu R společné s dokumentem S. N_R je počet všech hashů v daném odstavci z dokumentu R. Vypočtené hodnoty jsou v intervalu $<0, 1>$, kde nula znamená, že dokumenty jsou naprosto odlišné, a naopak jedna znamená, že dokumenty jsou totožné. Uložíme si všechny vypočtené hodnoty pro každý dokument z lokální množiny dat. Když prozkoumáme všechny odstavce, zůstanou nám uložené cesty k dokumentům z lokální množiny dat a shody s odstavci ze zkoumaného dokumentu.



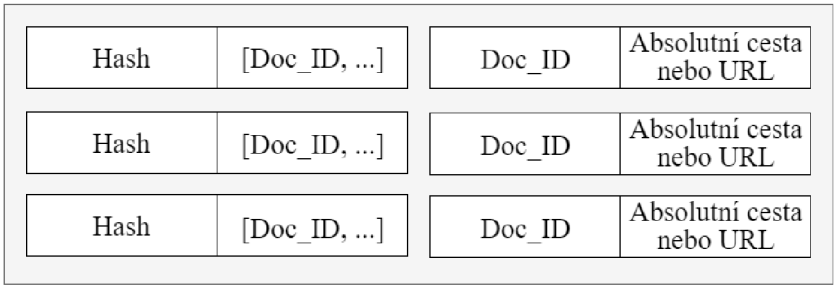
Obrázek 4.1: Návrh systému od zpracování dokumentů až po odhalení plagiátů.

4.2 Server

Hlavním úkolem serveru je indexace a vyhledávání hashů v hašovací tabulce. V této podsekci si oba problémy více přiblížíme.

4.2.1 Indexace dat

Jak už bylo zmíněno v kapitole 3, budeme indexovat dva typy dat. První je indexace hashů a identifikátorů dokumentů, ve kterém se hash nachází. Druhým typem dat, který potřebujeme ukládat, jsou identifikátory dokumentů a absolutní cesty k daným dokumentům nebo URL, pokud se jedná o indexování dat z wikipedie nebo HTML dokumentu. U indexace obou typů dat využijeme hašovací tabulku Sparsepp, viz 3.2. Výsledný index najdeme na obrázku 4.2.



Obrázek 4.2: Výsledná podoba indexu.

4.2.2 Vyhledávání hashů

Pokud chceme pouze indexovat data pro rozšíření lokální množiny dat, k vyhledávání vůbec nedojde. Pokud však chceme hledat plagiáty, musíme hashe v hašovací tabulce vyhledávat. V momentě, kdy přijde od klienta dotaz na hash, server se podívá do hašovací tabulky a vyhledá stejný hash. Pokud najde, podívá se do hašovací tabulky s absolutními cestami a vrátí klientovi všechny absolutní cesty k souborům, ve kterých se hash nachází. Následně uloží hash do hašovací tabulky pro rozšíření lokální množiny pro budoucí vyhledávání.

Algorithm 1 Chunking algorithm

1: **for each** source document d **do**
2: split the document into words
3: form word n-grams (*chunks*) from each n adjacent words
4: **for each** chunk **do**
5: compute the $fingerprintf$ of this chunk
6: add the (f, d) pair to the *list of pairs*
7: **end for**
8: **end for**
9: from the *list of pairs*, construct the *inverted index* mapping each fingerprint to the list of document IDs
10: **for each** suspicious document d **do**
11: **for each** source document s_i **do**
12: set the counter of common fingerprints $c(s_i)$ to 0
13: **end for**
14: split the suspicious document into *words*
15: form word n-grams (*chunks*) from each n adjacent words as we did with the source documents
16: denote $|d|$ the number of chunks in d
17: **for each** chunk of s_i **do**
18: compute the $fingerprintf$ of this chunk
19: look up the fingerprint in the *inverted index*, giving the *list of source document IDs* L
20: **for each** source document $s_i \in L$ **do**
21: $c(s_i) \leftarrow c(s_i) + 1$
22: **end for**
23: **end for**
24: **for each** source documents_i **do**
25: output the asymetricsimilarity $a(d, s_i) = \frac{c(s_i)}{|d|}$
26: **end for**
27: **end for**

Kapitola 5

Implementace

V této kapitole se podrobněji podíváme na samotnou implementaci systému. Jak jsme si již řekli v kapitole 4, náš systém je rozdělen na programy server a klient. Jedná se o konzolové aplikace, které zpracovávají dokumenty, respektive ukládají hashe do indexu. Obě aplikace jsou spouštěny pomocí skriptů v jazyce Python. Při spuštění serverů se vytvoří tzv. screen[1], což je terminálový multiplexer, který nám umožňuje vytvořit libovolný počet virtuálních obrazovek na jednom terminálu. Screen se nám hodí pro přehlednost, a také kvůli jednoduššímu ukončení serverů, viz 5.2.4. Při spuštění klientů se screenu nevyužívá.

5.1 Klient

Tato podkapitola se zabývá jednotlivými částmi implementace klienta. Nejprve se podíváme na zpracování argumentů, následuje připojení k serverům, poté se podíváme na implementaci zpracování dokumentů, komunikaci se serverem, vyhodnocení a nalezení plagiátů.

5.1.1 Zpracování argumentů

Po spuštění klienta je nejdříve nutné zpracovat argumenty. K tomu slouží knihovna Command Line Parameters Parser, viz 3.4. U klienta je možné nastavit několik argumentů, z nichž všechny volitelné. Možné argumenty[2] jsou:

- -b – určuje binární soubor, který se má spustit
- -i – určuje vstupní složku, tzn. složku, ve které jsou nachystané dokumenty ke zpracování
- -o – určuje výstupní složku, tzn. složku, kam se uloží vyhodnocení po prozkoumání vstupních dokumentů
- -d – určuje, zda se budou vstupní dokumenty pouze indexovat, nebo se budou hledat i plagiáty
- -p – určuje, na jakém portu bude program běžet

[1] Více o příkazu screen na https://www.root.cz/clanky/screen-jeste-mocnejsi-terminal/.

[2] Všechny argumenty mají nastaveny výchozí hodnoty, které je možné změnit ve startovacím skriptu, což je startClient.py.

- -s – určuje soubor s názvy serverů, na kterých se spustí program server
- -c – určuje soubor s názvy serverů, na kterých se spustí program klient

5.1.2 Připojení k serverům

Jakmile jsou argumenty zpracovány, klient projde složku s dokumenty ke zpracování a vybere ty, které má za úkol zpracovat on a vloží je do fronty. To zjistí podle svého ID[3]. Následně se klient připojí k serverům a pošle serverům zprávu, zda mají hashe pouze indexovat, nebo o nich posílat klientům informace potřebné k odhalení plagiátů. Následně se vytvoří vlákna. Optimální počet vláken byl během testování stanoven na dva. Nyní máme vše připraveno a můžeme začít se zpracováváním dokumentů.

5.1.3 Zpracování dokumentů

Z předchozího odstavce už víme, že klient je *vícevláknový*. Každé vlákno obsahuje smyčku, ve které vyjme z fronty dokument ke zpracování, zjistí jakého je dokument formátu a podle toho zavolá funkci k normalizaci. Vlákno se ukončí v momentě kdy je fronta s dokumenty prázdná.

Nejjednodušší je normalizace čistě textových dokumentů, u kterých nemusíme odstraňovat žádné značky a stačí se zaměřit na odstranění interpunkce. Pokud jde o dokument typu PDF nebo jde o data stažená z Wikipedie či Seznamu, je nutné odstranit značky. Pro odstranění značek ze souborů typu PDF použijeme program pdftotext, který nám vrátí pouze text. U dat z Wikipedie musíme značky odstranit sami. Při zpracování nás zajímají dva HTML tagy a to `<doc>` a `<p>`. Když narazíme na začátek dokumentu, který poznáme tagem `<doc>`, uložíme si URL stránky, což je atribut tagu `<doc>`. Pak hledáme tagy `<p>`, které určují odstavce. Jeden odstavec z Wikipedie je tedy roven textu mezi tagy `<p>` a `</p>`. Data, stažená ze Seznamu, jsou uložena ve speciálním formátu, který má v každém dokumentu oddělené odstavce a zpracování není obtížné. Jakmile získáme normalizovaný odstavec, zjistíme, zda splňuje minimální délku odstavce, která je nastavena na patnáct slov. Pokud odstavec nemá patnáct slov, je zahozen. Pokud má patnáct až třicet slov, připojí se k dalšímu odstavci. Pokud máme odstavec s alespoň třiceti slovy, zavoláme funkci `processParagraph`, která odstavec dále zpracuje.

Funkce `processParagraph` má za úkol vytvořit n-tice slov, z těchto n-tic vytvořit pomocí hašovací funkce hashe a ty odeslat na server. Serverů je několik, a každý z nich má přidělený interval hashů, které má zpracovávat. Je tedy nutné spočítat, na který server se má hash poslat.

5.1.4 Komunikace se serverem

Komunikace se serverem se liší podle toho, zda má server hashe pouze indexovat, nebo se budou hledat plagiáty. V druhém případě je komunikace obtížnější a výrazně zpomalí běh programu. Komunikace začíná v obou případech stejně. Klient odešle serveru hash, následně odešle URL (v případě zpracovávání dat z Wikipedie) nebo absolutní cestu dokumentu. Pokud se jedná pouze o indexaci, komunikace v rámci právě zpracovávaného hashe končí a začíná se zpracovávat další hash. Pokud hledáme plagiáty, klient musí počkat, až mu server pošle o hashi informace. Nejdříve server pošle informaci o počtu dokumentů, které daný hash obsahují. Následně klient ve smyčce přijímá URL nebo absolutní cesty k dokumentům

[3]ID si každý klient spočítá pomocí souboru s názvy serverů, na kterých se procesy klient spustí.

a ty si ukládá do mapy. K jednotlivým URL a cestám k dokumentům si klient taky ukládá počet hashů, které mají společné.

5.1.5 Vyhodnocení a nalezení plagiátů

Do této části se klient dostane pouze v případě, že chceme hledat plagiáty. Vyhodnocení má dvě části, a to vyhodnocení odstavců a vyhodnocení celého dokumentu.

Vyhodnocení odstavců probíhá pokaždé po tom, co se zpracuje nějaký odstavec. Klient spočítá pro daný odstavec podobnost podle vzorce 4.1. Pokud podobnost překročí nastavený práh podobnosti[4], je odstavec uložen a spolu s ním i řetězec jedniček a nul, který určuje, jaká slova byla opsána. Pokud je například v řetězci na pozici s indexem tři jednička, je čtvrté slovo opsáno. Takto uložíme všechny odstavce s překročeným prahem podobnosti pro celý zpracovávaný dokument.

Jakmile je celý dokument zpracován, přichází na řadu vyhodnocení dokumentu. Nejprve si spočítáme celkovou podobnost zkoumaného dokumentu a všech dokumentů z lokální množiny textových dat, které obsahují alespoň jeden stejný hash - tyto dokumenty máme uloženy zároveň s počtem stejných hashů. Tuto podobnost spočítáme podle vzorce 4.1. Následně zjistíme, jestli mají dokumenty uložený nějaký odstavec s překročeným prahem podobnosti. Pokud ano, jsou tyto odstavce vypsány, spolu s vypočítanou podobností, a jsou v nich vyznačeny velkými písmeny slova, která byla opsána.

5.2 Server

Tato podkapitola se zabývá jednotlivými částmi implementace serveru. Nejprve se podíváme na zpracování argumentů, následuje navázání komunikace s klientem, komunikace s klientem a ukončení serveru.

5.2.1 Zpracování argumentů

Stejně jako u klienta, i u serveru se po spuštění nejdříve zpracují argumenty pomocí knihovny Command Line Parameters Parser, viz 3.4. Narozdíl od klienta však nejsou všechny argumenty volitelné. Je nutné programu říct, zda se budou servery zapínat nebo vypínat. Možné argumenty[5] jsou:

- start | stop – určuje, zda se servery zapnou nebo vypnou, povinný argument
- -b – určuje binární soubor, který se má spustit
- -o – určuje výstupní složku, tzn. složku, kam se uloží vyhodnocení po prozkoumání vstupních dokumentů
- -p – určuje, na jakém portu bude program běžet
- -s – určuje soubor s názvy serverů, na kterých se spustí program server

[4] V této práci je práh podobnosti u odstavců nastaven na 0,2.

[5] Kromě argumentu start | stop mají všechny argumenty nastaveny výchozí hodnoty, které je možné změnit ve startovacím skriptu, což je startServer.py.

5.2.2 Navázání komunikace s klientem

Jakmile jsou argumenty zpracovány, načte server data z lokální množiny textových dat a čeká na připojení klientů. Když se klient připojí, server vytvoří nové vlákno, které bude mít na starost komunikaci s tímto klientem. Nejprve přijde zpráva, zda-li má server hashe pouze indexovat nebo o nich posílat informace zpět klientovi.

5.2.3 Komunikace s klientem

Komunikace s klientem probíhá ve smyčce. Nejprve server přijme hash, který převede na *unsigned long*, následně přijme URL nebo absolutní cestu k dokumentu, ve kterém se hash nachází. Z této URL nebo absolutní cesty server vytvoří hash pomocí hašovací funkce, čímž vytvoří ID dokumentu.

Server se následně podívá do hašovací tabulky s hashi n-tic, jestli už tam daný hash s daným ID dokumentu není. Pokud tam není, vložíme ID dokumentu do seznamu ID dokumentů u daného hashe. Pokud hledáme plagiáty, tak si tento seznam ID dokumentů uložíme do prozatimní proměnné, abychom nemuseli v další části programu znova vyhledávat v hašovací tabulce.

Poté se server podívá do druhé hašovací tabulky, a to s ID dokumentu a jejich URL nebo absolutní cestou. Pokud tam ID dokumentu není, vložíme dvojici ID dokumentu a URL nebo absolutní cestu. Pokud bychom chtěli data pouze indexovat, vracíme se na začátek.

Pokud chceme hledat plagiáty, musíme klientovi posat všechny URL a absolutní cesty, ve kterých se daný hash nachází. Tyto URL a absolutní cesty máme uložené v seznamu v prozatimní proměnné z dřívějšího kroku. Nejprve klientovi pošleme počet prvků seznamu, a pak mu ve smyčce pošleme všechny prvky.

5.2.4 Ukončení serveru

Ukončení serverů se provádí pomocí skriptu v jazyce Python. Tento skript má za úkol ukončit screen, ve kterém běží proces server. Při ukončení screenu pošle screen procesům, které na něm běží, signál k ukončení. Když proces server tento signál dostane, zavolá se funkce `serialize`, která uloží obě hašovací tabulky do složky určené pro výstup.

Kapitola 6

Testování a vyhodnocení

Tato kapitola obsahuje vyhodnocení systému. Ukážeme si, na jakých datových sadách byl tento systém testován, dále systém porovnáme s jinými systémy pro odhalování plagiátů. Nakonec celkově zhodnotíme systém.

6.1 Datové sady pro testování systému

Pro testování tohoto systému bylo použito několik datových sad. Nejprve probíhalo testování na datech z Wikipedie, následně na datech z CommonCrowlu, z PAN korpusu a na datech ze Seznamu.

6.1.1 Wikipedia dumps

Wikipedia[1] nabízí zdarma stažení všech stránek v několika formátech. V našem případě bylo testováno na datech ve formátu Wikipedia Dump. Česká wikipedia, se kterou bylo testováno, má necelých 6 GB dat. Z těchto dat je při zpracování potřeba odstranit značky.

6.1.2 Data z CommonCrawlu

Jedná se o lokální sadu dat[2] z CommonCrawlu, která obsahuje anglická data o velikosti 47 GB. Tato sada dat byla využita pro testování zpracování většího množství dat.

6.1.3 PAN korpus

PAN korpus[3] je volně dostupný korpus určený pro testování systémů pro odhalování plagiátů. Korpus obsahuje značné množství originálních i podezřelých dokumentů, a navíc obsahuje dvojice dokumentů, které byly označeny jako plgiát, což nám značně usnadní kontrolu výsledků.

[1] Data z Wikipedie ke stažení z: https://dumps.wikimedia.org/

[2] Lokální sada dat z CommonCrawlu dostupná na lokálních serverech na: /mnt/data/commoncrawl/CC-2016-40/vert.

[3] PAN korpus je ke stažení z: http://www.uni-weimar.de/medien/webis/events/pan-15/pan15-web/plagiarism-detection.html

6.1.4 Data ze Seznamu

Jedná se o lokální data[4] z českého webu. Data jsou stažena ze Seznamu a obsahují 557 GB dat. Testování s touto sadou však probíhalo se zmenšenou verzí a to s 50 GB dat. Pro indexaci těchto dat je vhodné rozdělit hlavní soubor, který obsahuje všechna data, na několik menších souborů, aby se zpracování rozdělilo mezi všechny klienty.

6.2 Porovnání se systémem Theses.cz

Theses.cz [2] je webově založený systém pro odhalování plagiátů se zaměřením na závěrečné práce, který je provozován Masarykovou univerzitou. Tento systém se vyvíjí od roku 2006, kdy byl použit prototyp postavený na Oracle databázi. Poté systém začal přiřazovat k identifikátorům dokumentů sekvenci slov, pětic. Pro vypočítání podobnosti se začal používat *chunking algorithm*, viz 1. V roce 2008 měl systém naindexováno 250 000 dokumentů, ze kterých bylo vytvořeno 600 000 000 dvojic <dokumentID, n-tice>, z čehož bylo 445 000 000 unikátních n-tic. Výkon tohoto systému začal s přibývajícími daty klesat, a tak se od roku 2008 začlo pracovat na vylepšování tohoto systému.

Stejně jako tento systém, Theses.cz patří také do skupiny vnějších systémů pro odhalování plagiátů. Používá lokální množinu textových dat a vychází z chunking algoritmu. Text je převeden na slova, z nichž se vybírají pouze slova delší než tři znaky. Následně se vytvářejí pětice slov, ze kterých se pomocí hašovací funkce MD5 vytvoří hashe o délce 30 bitů. Systém počítá asymetrickou podobnost a následně vyznačí uživateli opsané části.

Hlavní datovou strukturou je inverted index, který se skládá ze dvou polí, viz 2.1. První pole mapuje otisky na ID dokumentů, druhé pole obsahuje seřazené pole ID dokumentů pro otisk s hodnotou 0, seřazené pole ID dokumentů pro otisk s hodnotou 1 atd.

Narozdíl od tohoto systému, Theses.cz používá jinou topologii připojení serverů. Theses.cz má jeden hlavní proces a desítky až stovky podřadných procesů, které se všechny připojí jak na hlavní server, tak i sami na sebe, viz 6.1.[5]

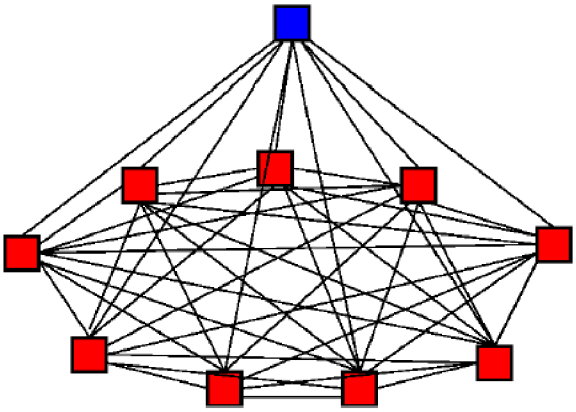
Oba systémy vycházejí ze stejných algoritmů, Theses.cz je však dlouho vyvíjeným systémem a je tedy mnohem propracovanější.

6.3 Testování systému

Systém byl od počátku testován především na datech z Wikipedie. Probíhalo několik různých testů. Nejprve se testovalo se staršími a aktuálními daty z Wikipedie, kdy bylo za úkol najít stránky, které se od starší verze změnily. Wikipedia nabízí k nahlédnutí změny vůči starší verzi, díky tomu jsme mohli snadno zjisit, jak testy dopadly. Tento systém měl úspěšnost kolem 84 %.

Následně jsem vytvořil sadu testovacích dokumentů, která byla naplněna náhodně vygenerovaným textem. Do tohoto textu byly vkládány věty z Wikipedie. Při vkládání dlouhých vět nebo více vět za sebou měl systém vysokou úspěšnost, pokud byly vloženy samostatné věty různě do dokumentu, byla úspěšnost malá. Pokud se ale u zpracování takových dokumentů zmenšila délka n-tic, úspěšnost rostla.

[4] Lokální data stažená ze Seznamu dostupná na: /mnt/minerva1/nlp/corpora_datasets/monolingual/czech/seznam2017



Obrázek 6.1: Topologie propojení uzlů v systému Theses.cz, modře je zvýrazněn hlavní proces, červeně jsou označeny podřadné procesy, převzato z [5].

6.4 Zhodnocení systému

Systém by se dal zhodnotit několika způsoby. Můžeme hodnotit zpracování dokumentů od vytvoření n-tic po vytvoření hashů, můžeme hodnotit úspěšnost vyhledávání plagiátů a nebo můžeme hodnotit rychlost systému.

Zpracování dokumentů, vytvoření n-tic a hashů zvládá náš systém dobře, umí zpracovat více typů souborů a používá velmi rychlou hašovací funkci. Úspěšnost vyhledávání plagiátů byla při testování velká, hodně záleželo na nastavení délky n-tic a dalších normalizačních prvků. Rychlost zpracování dokumentů a celého systému není příliš velká, může za to hlavně pomalejší komunikace mezi servery a klienty.

Kapitola 7

Závěr

Cílem této práce bylo prostudovat metody pro odhalování plagiátů se zaměřením na studentské práce, seznámit se s dosavadními výsledky v oblasti automatického zpracování přirozeného jazyka a navrhnout a implementovat systém pro odhalování plagiátů na základě rozsáhlé množiny textových dat. Výsledný systém je schopen zpracovávat a indexovat dokumenty více typů a hledat v nich plagiáty.

V porovnání s jinými systémy náš systém dosahuje srovnatelných výsledků v úspěšnosti vyhledávání, v budoucnu by však bylo vhodné systém zrychlit a zmenšit paměťové nároky.

Výstupem systému je soubor obsahující porovnání zdrojového a podezřelého dokumentu, jsou vypsány odstavce s podobností nad 20 % a je v nich velkými písmeny vyznačen opsaný text. V budoucnu by se dal systém rozšířit o uživatelské rozhraní, kde bude vyznačení opsaných částí vyřešeno přehledněji.

Literatura

[1] *Global WordNet Association: EuroWordNet*. 2001, [online]. [cit. 10.05.2017].
URL http://www.illc.uva.nl/EuroWordNet/

[2] Czech National Archive of Graduate Theses. 2008-2017, [online]. [cit. 11.05.2017].
URL http://theses.cz/

[3] Balvínová, A.: OCR. In: *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)*. 2003, praha : Národní knihovna ČR, 2003. [online]. [cit. 2017-04-26].
URL
http://aleph.nkp.cz/F/?func=direct&doc_number=000000218&local_base=KTD

[4] Ceska, Z.; Toman, M.; Jezek, K.: (2008) Multilingual Plagiarism Detection. In: *Dochev D., Pistore M., Traverso P. (eds) Artificial Intelligence: Methodology, Systems, and Applications*. AIMSA, 2008, lecture Notes in Computer Science, vol 5253. Springer, Berlin, Heidelberg.

[5] Kasprzak, J.: *Distributed Systems for Discovering Similar Documents*. Doctoral theses, dissertations, Masaryk University, Faculty of Informatics, Brno, 2015 [cit. 2017-04-26], [online].
URL http://is.muni.cz/th/1885/fi_d_b1/

[6] Khoshnavataher, K.; Zarrabi, V.; Mohtaj, S.; aj.: Developing Monolingual Persian Corpus for Extrinsic Plagiarism Detection Using Artificial Obfuscation—Notebook for PAN at CLEF 2015. In *CLEF 2015 Evaluation Labs and Workshop – Working Notes Papers, 8-11 September, Toulouse, France*, editace L. Cappellato; N. Ferro; G. Jones; E. San Juan, CEUR-WS.org, Září 2015, ISSN 1613-0073.

[7] Kuznetsov, M.; Motrenko, A.; Kuznetsova, R.; aj.: Methods for Intrinsic Plagiarism Detection and Author Diarization—Notebook for PAN at CLEF 2016. In *CLEF 2016 Evaluation Labs and Workshop – Working Notes Papers, 5-8 September, Évora, Portugal*, editace K. Balog; L. Cappellato; N. Ferro; C. Macdonald, CEUR-WS.org, sep 2016, ISSN 1613-0073.
URL http://ceur-ws.org/Vol-1609/

[8] Mareš, M.; Valla, T.: Vyhledávácí stromy - Recepty z programátorské kuchařky. december 2013, [online]. [cit. 04.05.2017].
URL https://ksp.mff.cuni.cz/kucharky/vyhledavaci-stromy/

[9] Pekař, T.: *Vyhledávání duplicitních textů*. bakalářská práce, FIT VUT v Brně, Brno, 2015.

[10] Poulston, A.; Stevenson, M.; Bontcheva, K.: Topic Models and n-gram Language Models for Author Profiling—Notebook for PAN at CLEF 2015. In *CLEF 2015 Evaluation Labs and Workshop – Working Notes Papers, 8-11 September, Toulouse, France*, editace L. Cappellato; N. Ferro; G. Jones; E. San Juan, CEUR-WS.org, Září 2015, ISSN 1613-0073.

[11] Pranckevičius, A.: *More Hash Function Tests · Aras' website*. [online]. [cit. 28.04.2017].
URL http://aras-p.info/blog/2016/08/09/More-Hash-Function-Tests/

[12] Robenek, D.; Platos, J.; Snásel, V.: Efficient In-memory Data Structures for n-grams Indexing. In *Proceedings of the Dateso 2013 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Pisek, Czech Republic, April 17, 2013*, *CEUR Workshop Proceedings*, ročník 971, editace V. Snásel; K. Richta; J. Pokorný, CEUR-WS.org, 2013, s. 48–58.
URL http://ceur-ws.org/Vol-971/paper21.pdf

[13] Welch, N.: *Hash Table Benchmarks*. [online]. [cit. 02.05.2017].
URL http://incise.org/hash-table-benchmarks.html

[14] Yang, H.; Callan, J.: Near-duplicate detection by instance-level constrained clustering. In: *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2006.

Přílohy

Příloha A

Obsah přiloženého DVD

Práce obsahuje přiložené DVD se všemi soubory potřebnými ke spuštění systému. Dále jsou na DVD zdrojové soubory k písemné práci v LaTeXu. Soubory a adresáře na DVD:

- **lib** - složka s knihovnami clp_parser a sparsepp
- **input** - výchozí složka pro vstup
- **output** - výchozí složka pro výstup
- **tex** - složka se zdrojovými soubory pro vytvoření písemné práce
- **server.c** - zdrojový kód pro program server
- **client.c** - zdrojový kód pro program klient
- **startServer.py** - spouštěcí skript pro program server v Pythonu
- **startClient.py** - spouštěcí skript pro program klient v Pythonu
- **servers.txt** - seznam serverů, na kterých se má spustit program server
- **clients.txt** - seznam serverů, na kterých se má spustit program klient
- **xxhash.*** - soubory pro hašovací funkci xxHash
- **Makefile**
- **poster.png**
- **README.txt**

Příloha B

Manuál

Tento systém je konzolová aplikace rozdělená na procesy typu server a klient. V tomto manuálu si ukážeme, jak aplikaci nainstalovat a spustit.

B.1 Instalace

Aplikace je určena pro Linux. Kromě běžně dostupných nástrojů je nutné mít nainstalovanou knihovnu Boost. Aplikace byla testovaná s knihovnou Boost verze 1.54.0, měla by však fungovat i s nižšími verzemi od 1.50.0.

Přiložené DVD obsahuje soubor Makefile, který slouží ke kompilaci aplikace. Program se zkompiluje po zadání příkazu **make**.

B.2 Spuštění

Jakmile zkompilujeme aplikaci pomocí příkazu make, můžeme aplikaci spustit. Nejprve je nutné spustit server. Možné argumenty serveru:

- start | stop – určuje, zda se servery zapnou nebo vypnou, povinný argument
- -b – určuje binární soubor, který se má spustit
- -o – určuje výstupní složku, tzn. složku, kam se uloží vyhodnocení po prozkoumání vstupních dokumentů
- -p – určuje, na jakém portu bude program běžet
- -s – určuje soubor s názvy serverů, na kterých se spustí program server

Příklady spuštění serveru:
python3 startServer.py start -s servers.txt
python3 startServer.py start -b "./server" -o "./output" -p 11420

Příklady vypnutí serveru:
python3 startServer.py stop -s servers.txt
python3 startServer.py stop -b ./server -p 11420

Když jsou servery spuštěny, můžeme spustit klienty. Možné argumenty klienta:

- -b – určuje binární soubor, který se má spustit
- -i – určuje vstupní složku, tzn. složku, ve které jsou nachystané dokumenty ke zpracování
- -o – určuje výstupní složku, tzn. složku, kam se uloží vyhodnocení po prozkoumání vstupních dokumentů
- -d – určuje, zda se budou vstupní dokumenty pouze indexovat, nebo se budou hledat i plagiáty
- -p – určuje, na jakém portu bude program běžet
- -s – určuje soubor s názvy serverů, na kterých se spustí program server
- -c – určuje soubor s názvy serverů, na kterých se spustí program klient

Příklady spuštění serveru:
python3 startClient.py
python3 startClient.py -p 11420 -i ./input -d

Příloha C

Plakát

ODHALOVANÍ PLAGIÁTŮ

CÍLE PRÁCE

Cílem této práce bylo prostudovat metody pro odhalování plagiátů se zaměřením na studentské práce, seznámit se s dosavadními výsledky v oblasti automatického zpracování přirozeného jazyka a navrhnout a implementovat systém pro identifikaci plagiátů na základě rozsáhlé množiny textových dat.

POPIS VÝSLEDNÉHO SYSTÉMU

Výsledný systém je schopen zpracovávat a indexovat dokumenty více typů a hledat v nich plagiáty. Výstupem systému je soubor obsahující porovnání zdrojového a podezřelého dokumentu, jsou vypsány odstavce s podobností nad 20% a je v nich velkými písmeny vyznačen opsaný text.

VYHODNOCENÍ SYSTÉMU

V porovnání s jinými systémy tento systém dosahuje srovnatelných výsledků v úspěšnosti vyhledávání, v budoucnu by však bylo vhodné systém zrychlit a zmenšit paměťové nároky.

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ