VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Informační systém pro rozpoznávání plagiovaných dokumentů

Information System for a Document plagiarism Detection

2015 Pavel Kolář

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Pavel Kolář**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Informační systém pro rozpoznávání plagiovaných dokumentů
Information System for a Document plagiarism Detection

Zásady pro vypracování:

Cílem této práce je vytvořit informační systém, který bude umět vyznačit rozsah podobnosti vstupního dokumentu s dokumenty uloženými ve službě vyvinuté na Katedře informatiky. V rámci práce bude kladen důraz zejména na uživatelské rozhraní.

Výsledný systém bude mít následující funkce:
1. Možnost registrace uživatele do systému.
2. Vytvoření rozhraní pro vyhledávání podobných dokumentů.
3. Bude možné vyznačit stejné části dvou dokumentů.
4. Vygenerování zprávy, která shrne, kolik částí ze vstupního dokumentu je plagiováno.
5. Porovnání výsledného systému s jinými systémy umožňujícími porovnání dvou dokumentů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Rád bych poděkoval vedoucímu mé bakalářské práce, panu Ing. Radimu Bačovi, Ph.D., za jeho trpělivost a poskytnuté rady, bez kterých by tato práce nevznikla.

Abstrakt

Práce se zabývá detekcí plagiovaných nebo podobných dokumentů a způsoby jak takovou podobnost mezi dvěma dokumenty zobrazit. Cílem bylo vyvinout webovou aplikaci která uživateli bude umožňovat podobné dokumenty hledat a na základě různých metrik graficky vyznačit společné části mezi dvěma texty. Aplikace byla vyvíjená na platformě .NET a byl použit webový aplikační framework ASP.NET MVC. V teoretické části popisuju, co plagiarismus je a popisuji různé způsoby jak lze takovou detekci řešit. V praktické řeším konkrétní popis algoritmů a zvolených postupů spolu s jednoduchou analýzou projektu. Ke konci použité algoritmy testuji na zkušebních datech a jejich průběhy zapisuju do grafů, z kterých vyvozuji závěr.

Klíčová slova: plagiarismus, podobnost, informační systém, webová aplikace, detekce

Abstract

My thesis describes ways of detecting plagiarized or similar documents and displaying such similarity to the user. Main goal was to develop web application that would allow user to search for similar documents and based on various methods display this similarity in a graphical way. Application was developed using Microsoft's .NET platform specifically for the web part of the project I was using web application framework ASP.NET MVC that implements model-view-controller pattern. The theoretical part explains what plagiarism actually is and describes various ways how to approach computer based detection of plagiarized documents. The practical part focuses on description of used algorithms and techniques accompanied by basic software analysis. In the end of my thesis I am conducting benchmarks of the algorithms and from their performance I plot graphs from which I deduct my conclusions.

Keywords: plagiarism, similarity, information system, web application, detection

Seznam použitých zkratek a symbolů

LCS	–	Longest common substring - nejdelší společná část dvou textů
CP	–	Cosínová podobnost - metrika určující podobnost mezi dvěma vektroy

Obsah

Seznam obrázků

1 Úvod

V mé bakalářské práci se zabývám detekci podobných anebo plagiovaných dokumentů. Cílem je naimplementovat webovou aplikaci, která bude umožňovat uživateli pohodlně porovnávat a vyhledávat podobné dokumenty v databázi odevzdaných prací studentů všb. K vyhledání a určení míry podobnosti využívám službu vyvinutou na Katedře informatiky, ke které mi byl umožněn přístup. Služba určuje míru podobnosti, ale neřeší už uživatelskou část, která by uživateli zobrazila podobnost dokumentů ve formě grafického porovnání, anebo tisku zprávy o porovnání. Právě způsoby jak dva dokumenty porovnat a podobnost zobrazit uživateli, je to, na co se ve své práci zaměřuji.

Kapitola 2 je věnována obecnému úvodu do problému plagiátorství, popisuje, co to znamená něco plagiovat. V další kapitole 3 popisuji různé přistupy a metody k detekci plagiátu. Kapitola 4 se zabývá už praktickou stránkou detekce, a to popisem mojí implementace jednotlivých algoritmů spolu s jejich teorií. Také popisuje jaké algoritmy jsem nejdříve zvažoval a pro které jsem se na základě testů a poznatků při studování jejich teorie nakonec rozhodl. Taky popisuji postup jakým je text zvýrazňen uživateli. Konec kapitoly je věnován teorii o redukci textu, konrétně stemmingu a odstraňování stop slov. Další kapitola 5 je popis vlastního řešení, kde popisuji specifikace aplikace, architekturu, jednoduchou analýzu a detaily implementace. Kapitola 7 je věnovaná testům paměťové a časové náročnosti vybraných algoritmů spolu s jejich grafy.

2 Plagiarismus

Plagiarismus z latinské slova plagiarius "únosce, svůdce, lupič".[1] Je čin, při kterém vydáváme cizí dílo za své. Kdykoli parafrazujeme, shrneme, použijeme myšlenky, nápady, slova, fráze, anebo celé věty z práce někoho jiného, tak je třeba uvést odkaz v textu. Nestačí pouze bibliografické citace na konci práce, jak si většina lidí myslí. Je třeba si taky dávat pozor, že i když třeba neúmyslně, odkaz v textu špatně uvedeme, tak je to také považováno za plagiarismus.[2]

2.1 Co se považuje za plagiarismus

- Prohlašení cizí práce za svou vlastní
- Použití slov a nebo myšlenek někoho cizího bez uvedení zdroje
- Neuvedení jedné a nebo obou uvozovek při citaci
- Uvedení špatného zdroje citace
- Úprava původního textu bez uvedení zdroje

[10]

2.2 Odkaz v textu

Označujeme části textu naší práce, které nejsou naše, anebo pro jejiž napsání jsme využili myšlenky někoho jiného. Je to jakási forma tagu, která určuje, co jsme si vypůjčili od ostatních. Existují zaběhlé a čtenáři známe způsoby, jak takový odkaz provést. Nelze říct, že je jeden lepší než durhý, spíše využívají jiných pravidel a norem. Mezi nejznámější patří: Modern Language Association(MLA), American Psychological Association (APA), Harvard a Oxford.[7]

2.3 Kdy musíme uvádět citaci

Zdroje, které musí být citované nejsou jen texty, ale také umění, grafika, hudba, počítačový program, nápady, myšlenky, obrázky, grafy, diagramy, data, anebo jakákoli jiná forma komunikace či nahrávacího média. Na druhou stranu nemusíme citovat nebo uvádět autora pro fakta, které jsou běžně známé. Jsou to fakta, známe všemi, anebo téměř všemi čtenáři z cílové skupiny, pro kterou svou práci píšeme např. pokud napíšeme, že *Země je kulatá*, tak nemusíme uvádět jak jsme na to přišli, protože je to považováno za běžně známý fakt i když třeba byl potvrzen složitým vědeckým experimentem. Je patrné, že nelze úplně jasně určitě, co lze považovat za běžný fakt. Je třeba si dávat pozor při posuzování jestli něco běžně známý fakt skutečně je, a není to tak jen z našeho pohledu jako člověka znalého v oblasti, o které píšeme. Když jsme na pochybách, je lepší vždy zdroj uvést.[18]

2.4 Následky plagiarismu

Plagiarismus jako takový, není trestný čin, pokud se nejedná o pořušení patentových zákonů. Je, ale považován za velice nekalou praktiku a obzvláště na akademicé půdě je brán velice vážně. Podle závažnosti může například znamenat neuznání předmětu, ukončení studia nebo rozvázání pracovního poměru spolu se ztrártou reputace a důvěryhodnosti.

2.5 Typy plagiátů

Hranice mezi plagiarismem a legitimní prací není vždy úplně černobílá, i tak, ale můžeme plagiarismus rozdělit do kategorií podle závažnosti, které nám umožňí plagiarismus lépe odhalit nebo se mu vyvarovat.

- Klon - Úplné okopírovaní cizí práce bez jakékoli upravy a vydávání za svou
- Copy&Paste - Využívání velkých částí textu z jednoho zdroje
- Find&Replace - Nahrazení jen klíčových slov a některých vět
- Remix - Parafráze z více zdrojů přepsané tak aby vypadali jako autentický text
- Recycle - Autor hojně využívá své předešlé práce bez uvedení zdroje
- Hybrid - Autor kombinuje bezchybně citované zdroje s necitovanými
- Mashup - Autor "polepí"text z několika zdrojů
- 404 Error - Odkazy na zdroje jsou nefunkční a nebo úplně vymyšlené
- Aggregator - Obsahuje správně citované zdroje, ale žádný autorův originální text

[9]

3 Detekce

Detekce plagiarismu je proces, který se zabývá vyhledávaní plagiátu v pracích nebo dokumentech. Rozšíření počítačů a hlavně internetu umožnilo velice jednoduše plagiovat práce ostatních. S kopírovaním se většinou setkávámě na akademické půdě, kde se píší různé práce, zprávy a vědecké listy. Tento problém se však může vyskytovat prakticky ve všech oborech.

Detekce může být prováděna buď ručně, nebo za pomocí počítačového programu. Ruční detekce je extrémně pracná, vyžaduje perfektní paměť a trpělivost. Stává se téměř nemožnou, když je třeba porovnat větší množství dokumentů. Počítač tyto problémy nemá, nikdy se neunaví, má perfektní paměť. Nedělá mu problém porovnávat stovky, ne-li tisíce dokumentů, ve kterých je mnohem větší šance, že se plagiát objeví. Poté, toto podezření poté může vyhodnotit člověk.

3.1 Metody

Všechny druhy metod, jak plagiarismus odhalit, se dají rozdělit do dvou kategorií.

3.1.1 Externí

Porovnává podezřelý dokument s kolekcí dokumentů, které jsou považovaný za originální a jejím ukolem je nalést dokumenty, jehož text je podobný textu ve vstupním dokumentem do takové míry, že ho považujeme za podezřelý.

3.1.2 Intristická

Analyzuje samotný text, aniž by ho s něčím porovnávala a snaží se nalést náhlou změnu ve stylu psaní, která může značit, že text je plagiovaný. Tato metoda vrací velké množsví falešných výsledků a poslední slovo má vždy člověk, který detekci provádí.

3.2 Přístupy

3.2.1 Fingerprinting

Je asi nejrozšířenější metodou. Tato metoda z porovnávaného textu vybere na základě algoritmu tzv. n-gramy, což jsou po sobě jdoucí prvky, v našem případě slova z porovnávaného textu, které se pak porovnávají s n-gramy spočítaných z kolekce dokumentů, ve kterých hledáme shodu. Tuto metodu využívá služba vyvinutá na Katedře informatiky, kterou jsem ve své práci používal.

3.2.2 String matching

Za pomocí string matching se snažíme najít průnik dvou textů, které jsou stejné mezi porovnávanými dokumenty.

3.2.3 Bag of words

Z textu se spočítá jeden nebo více vektorů, např. podle počtu výskytů jednotlivých slov, a pomocí těchto vekterů se počítají různé metriky. Jedna z nejjednodušších počítá úhel mezi vektory, čím je úhel menší, tím jsou si texty podobnější.

3.2.4 Citation analysis

Je jediný přístup, který nespoléhá na podobnost dvou textů, ale hledá podobnost v citacích, z toho vyplývá, že je vhodný spíše pro vědecké práce, které citace hojně využívají.

3.2.5 Stylometrie

Snaží se zjistit unikátní styl autorova psaní a hledá části textu, které se od tohoto stylu výrazně liší. [3]

4 Popis použitých algoritmů

Jakmile nám služba vyvinutá na Katedře informatiky najde podobnost se vstupním dokumentem, tak řešíme jak takovou podobnost uživateli zobrazit, aby mohl sám posoudit jestli se jedná o plagiát. To řeším dvěma způsoby, oba zvýrazňují části textu, ale každý podle jiného kritéria.

4.1 String matching

Chtěl jsem najít nejdelší společnou část dvou textů, a to ne jen jednu, ale všechny části od určité délky, tj. že budu ignorovat společná slova, těch by bylo určitě spousta, ale budu se zaměřovat jen na společné věty a více slov za sebou. V informatice je to celkem známy problém, anglicky se jmenuje Longest Common Substring Problem, dále už jen LCS. Existují dva přístupy jak tento problém řešit, jeden za pomocí dynamického programování s náročnosti $\Theta(nm)$, druhý pomocí použití suffix tree s náročností $\Theta(n + m)$. Přirozeně jsem se začal zajímat o řešení s využití suffix tree.[15]

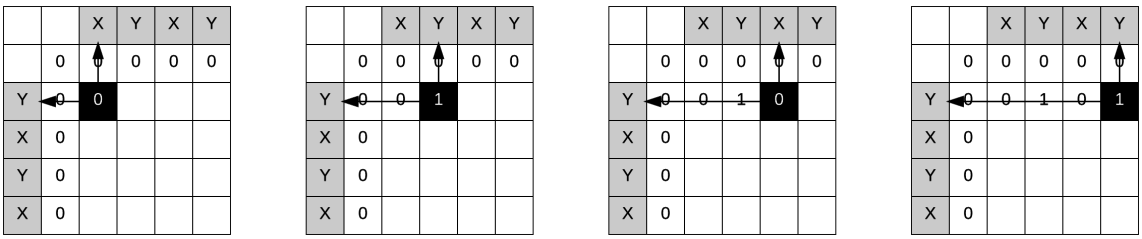
Příklad 4.1
Pojem LCS se bude dále vyskytovat poměrně často, zde je příklad jak LCS pro dvě věty vypadá. Mějme dvě věty *A programming language is a formal constructed language designed to communicate instructions to a machine...*, *The description of a programming language is usually split into the two components...* při bližším pohledu zjistíme že jejich nejdelší společná část je *programming language is*. ■

4.1.1 Suffix tree

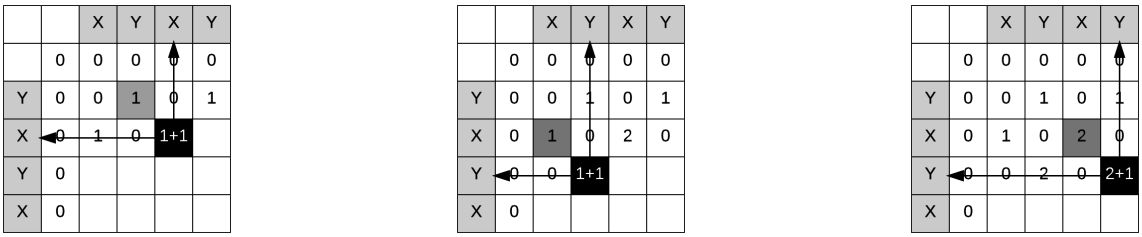
Suffix tree je datová struktura, která se používá pro uložení textu, anebo obecně znaků z konstatně velké abecedy. Obsahuje všechny podslova vstupního textu uložené ve formě pozice a počtu znaků za touto pozicí. Zajímavostí je, že poměrně nedávno (1995) byl objeven algoritmus, který je schopen strom sestrojit v lineárním čase, tzv. Ukkonen's algorithm. Nad takovým stromem je poté, možno provádět různé operace v extrémně krátkém čase, oproti textu uloženém jako pole znaků. Chtěl jsem tuto datavou strukturu použít pro hledání LCS, což je možné provést v čase $\Theta(n+m)$, oproti ostatním algoritmům, které jsou toho schopné v nejlepším případě v $\Theta(nm)$. Takové zrychlení je na úkor paměťové náročnosti, protože algoritmus využívá paměťově náročnou strukturu, jako je strom pro uložení dat, s kterými se pracuje. Výsledná velikost takového stromu odpovídá asi 20 násobku velikosti vstupního textu.[16] To znamená, že pro naše účely je tento přístup nepoužitelný. Pro představu, strom pro text o 4000 slovech, by zabíral v paměti místo přes jeden gigabajt. V našem projektu pracujeme s texty mnohem většími. O tom jsem se přesvědčil v testech v kapitole 7. Pro test jsem využil už naimplementovaný suffix strom[13], protože implementace takového stromu, spolu s algoritmem pro hledání LCS, je mimo rozsah této prace.

4.1.2 Dynamické programování

Tento přístup si zaznamenává data do matice, namísto do stromové struktury. Je mnohem méně paměťově náročný, za to má větší časovou složitost. Princip fungování algoritmu je poměrně jednoduchý. V první kroku si vytvoříme dvojrozměrné celočíselné pole velikosti o jedno větší, v každé dimenzi, jako je délka příslušného textu. Pro ilustraci si zvolíme text $XYXY$ a $YXYX$. Na pozice kde se i nebo j rovnají nule, napíšeme nulu. To nám zaplní celý první sloupec a řádek nulama. Dále postupně procházíme naše pole a díváme se jestli na pozici řádku, která odpovídá pozici znaku v první porovnávaném textu a na pozici sloupce, která odpovídá pozici znaku v druhém porovnávaném textu se nachází stejný prvek. Pokud ano, tak píšeme prozatím 1 pokud ne tak 0. Tento postup lze vidět na obrázku 1. Nastane-li situace, že při nalezeních shody máme v levém horním políčku tj. $[i-1, j-1]$ (zobrazeno šedě) číslo, tak k němu přičítáme jedničku. Tyto situace jsou zobrazeny na obrázku 2. Tímto postupem dostaneme výslednou matici na obrázku 3, ze které lze vyčíst společný text, v našem případě to je YXY a XYX.[15]

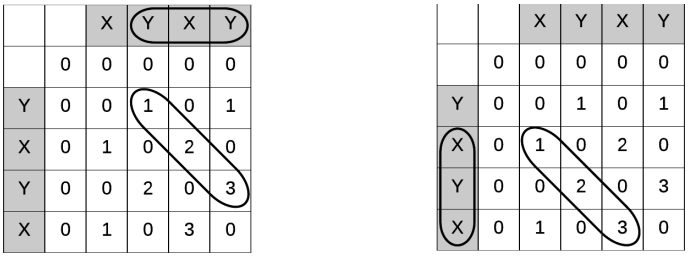


Obrázek 1: Pokud jsou znaky stejné píšeme jedna



Obrázek 2: Přičítáme jedna k hodnotě na pozici $[i-1, j-1]$

Po implementaci a otestování tohoto algoritmu jsem ale zjistil, že nalezení LCS v rozsáhlých textech trvá řádově několik sekund. To dávalo poměrně smysl, pokud máme dva průměrně dlouhé dokumenty, každý o délce 40000 znaků, tak nám vzniká matice o 1.6milardách prvků, kde se každý prvek musí spočítat. Na algoritmu nebylo už co vylepšovat, tak jsem se zaměřil na data, ve kterých shodu hledám, jestli nenaleznu prostor pro zlepšení. Problém je v tom, že shodu hledám na znak přesně, tj. že sloupce a řádky matice skládám z jednotlivých znaků textu. Pokud, ale matici sestavíme ne z jednotlivých znaků, ale z celých slov, tak nám výsledný počet prvků výrazně klesá. Počet slov v 40000 znacích dlouhém textu je ≈ 7000, takže matici o velikosti 1.6milard prvků se nám



Obrázek 3: Nalezení LCS ve výsledné matici

podařilo snížit na necelých 50 milionů. Po této úpravě je algoritmus schopný najít LCS v řádech milisekund.

4.2 Bag of words

String matching je vhodný na odhalování části dvou textů, které jsou si absolutně totožné, tj. když náš potenciální plagiátor vykopíroval část textu a použil ho ve svém dokumentu bez jakékoli úpravy. Stačí ale, aby v použitém textu zaměnil jedno nebo více slov, a už takhle plagiovaný text neodhalíme. Na tento typ problému budeme muset použít přístup Bag of Words, konkrétně metodu hledání cosínové podobnosti, dále už jen *CP*.

4.2.1 Cosínová podobnost

Je metrika, při které měříme podobnost dvou vektorů na základě úhlu mezi nimi. Toto měření je nezávislé na velikosti daných vektorů, zajímá nás jen jejich orientace. Cosinus $0°$ je jedna a pro jakýkoli jiný úhel je menší. Dva orientací totožné vektory mají $CP = 1.0$, dva na sebe kolmé vektory, tj. pod úhlem $90°$ mají $CP = 0.0$ a dva diametrálně odlišné vektory, tj. úhel mezi nimi je $180°$ mají $CP = -1.0$. Naše vstupní vektory budou mít vždy kladné hodnoty, takže náš výsledek nebude nikdy záporný a bude vždy spadat do intervalu $< 0, 1 >$, dále v textu si ukážeme proč tomu tak je.

Cosinus dvou vektorů se dá vyvodit ze vzorce pro skalární součin $a \cdot b = ||a||||b||cos\theta$, výsledný vzorec potom je

$$similarity = cos\theta = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

[5][19]

Máme matematický vzorec jak takovou vzdálenost vypočítat, ale naše vstupy jsou dva texty a né číselné vektory, je tedy jasné, že budeme muset text nějakým způsobem převést do číselné podoby. To uděláme tak, že spočítáme četnost slov v textu z této četnosti, pak vytovoříme náš výsledný vektor. Tady vidíme, proč naše vektory nemohou mít záporné hodnoty, protože žádné slovo se nám v textu nemůže vyskytovat záporně krát.

Příklad 4.2
Mějme dva vstupní texty *Dnes svítí slunce, ale zítra asi bude pršet* a *Dnes bude asi pršet, ale zítra už bude krásně* Zjistíme si seznam unikátních slov z obou textů a spočítáme počet jejich výskytů v obou textech.

dnes	1	1
svítí	1	0
slunce	1	0
ale	1	1
zítra	1	1
bude	1	2
pršet	1	1
už	0	1
krásně	0	0

Výsledný vektor pro první text je $[1, 1, 1, 1, 1, 1, 1, 1, 0, 0]$ a pro druhý $[1, 0, 0, 1, 1, 1, 2, 1, 1, 1]$. Jejich podobnost je ≈ 0.74 [11] ■

Tento algoritmus se mi podařilo udělat ještě flexibilnější tím, že jsem do něho zapojil stemming a stop slova, viz. kapitola 4.4. Předtím než věty začnu porovnávat, tak z nich odstraním stop slova a jednotlivá slova převedu na jejich kmen. Takže náš potenciální plagiátor může větu opsat tak, že změní slovosled, dokonce slova i jinak vyskloňuje, a my, i tak poznáme, že jsou věty podobné.

Příklad 4.3
Věty *Se stoupající teplotou odpor narůstal* a *Narust odporu způsobila stoupající teplota* jsou pro nás velice podobné $CP \approx 0.7$. ■

Existuje spousta jiných metrik, které jsou dokonalejší, ale jsou náročnější na výpočet a to si nemůžeme dovolit, protože takových výpočtů pro jeden dokument budeme provádět spousty. *CP* nám vrací úroveň podobnosti dvou textů v rozmezí $0.0 - 1.0$, ale má slabinu v tom že pro velké texty je tento algoritmus úplně nevhodný. Kdyby jsme jednoduše spočítali *CP* pro dva dokumenty, tak by jsme dostávali vždy podobnost velice blízkou nule. I kdyby tomu tak nebylo a podobnost pro dva velké dokumenty by jsme dokázali vyčíslit, tak by jsme jen dostali informaci, kterou už nám poskytuje služba vyvinutá na Katedře informatiky, která tohle už řeší a určitě mnohem sofistikovaněji. V čem zase *CP* vyniká je, že dokáže velice rychle určit shodu mezi dvěma krátkými texty, i když mají jiný slovosled, nebo úplně zaměněné některé slovo. Musíme tedy vymyslet postup, jak z dvou velkých dokumentů vyextrahovat menší části, které už pomocí *CP* budeme mezi sebou moc porovnávat.

4.2.2 Kvantování textů

Zvažoval jsem několik způsobů, jak takové rozdělení provést. Ten nejpřesnější by byl, že bych vybral všechny možné souvislé části na úrovni slov z obou textů. Co tím myslím si ukážeme na následujícím příkladu. Představme si text o 4 slovech a to *A B C D* způsoby,

kterými můžeme vybrat všechny souvislé části z tohoto textu jsou znázorněny v tabulce 4

	A	B	C	D	
1	A				
2		B			
3			C		4
4				D	
5	A	B			
6		B	C		3
7			C	D	
8	A	B	C		2
9		B	C	D	
10	A	B	C	D	1

Obrázek 4: Možnosti jak vybrat souvislý text

Možnosti jako *AC*,*BD* apod. nevybíráme, protože by se nejednalo o souvislou část. Při bližším pohledu na tabulku můžememe vypozorovat vztah pro výpočet, kolik takových výběrů existuje. Máme větu o čtyřech slovech a počet výběrů pro takovou větu je $4+3+2+1$, když si vztah zobecníme a budeme mít větu o n slovech, tak můžeme napsat, že počet výběrů bude $(n)+(n-1)+(n-2)+(n-3)$, to odpovídá vztahu $\frac{(n+1)n}{2}$, což je vzorec pro součet prvních n členů aritmetické posloupnosti. Postup odvození vztahu nejspíše není úplně matematicky přesný, ale vzorec jsem poté ověřoval na několika příkladech a fungoval.

$\frac{(n+1)n}{2} \approx n^2$ z toho vyplývá, že kdybychom porovnávali mezi sebou dva texty, tak bychom museli provést $\approx n^2m^2$ porovnání na *CP*, kde n je počet slov v prním textu a m v textu druhém. Tento přístup je sice nejpřesnější, ale počítat ho v rozumném čase není možné.

Nakonec jsem zvolil kompromis, a to tak, že si každý dokument rozdělím na jeho věty a ty mezi sebou porovnávám. Tím drasticky omezím počet porovnání, které musím provést a také mezi sebou budeme porovnávat právě ty části textu, které mají největší šanci, že jsou opsané. Tím mám na mysli, že když se někdo snaží přepsat text, tak parafrázuje právě jednotlivé věty. Pro představu, pro porovnání dvou textů každý o 1000 slovech, by jsme prvním způsobem museli provést $\approx 25 * 10^10$ porovnání za to, když mezi sebou porovnáváme jednotlivé věty, tak porovnání je už jen $\approx 62 * 10^6$. To je pokles o 99.98% oproti porovnávání všech možností.

4.3 Zvýraznění společného textu

Našli jsme společné části dvou dokumentů a tyto části chceme barevně zvýraznit. Víme, na které pozici společný text začíná, a na které končí. Takových dvojic může být několik, podle toho kolik zvýraznění chceme v textu provést. Na obrázku 5 můžeme vidět text *lorem ipsum*, ve kterém chceme zvýraznit dvě části textu, a to *rem* a *psu*. Pozice zvýraznění pro část *rem* začíná na pozici 2 a končí na pozici 4, to můžeme zapsat jako $[2, 4]$ a pro část *psu* $[7, 9]$. Zvýraznění provádíme tak, že konrétní čast textu obalíme do HTML tagů. Jak jde vidět na obrázku 6 text *rem* jsme obalili do fiktivních tagů, ale v této chvíli nastává problém. Pozice pro zvýraznění druhé části textu *psu* $[7, 9]$ přestává být platná, z $[7, 9]$ se stalo $[18, 20]$ obrázek 6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
L	O	R	E	M		I	P	S	U	M																			

Obrázek 5: Neotagovaný text

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
L	O	<	T	A	G	>	R	E	M	<	/	T	A	G	>		I	P	S	U	M								

Obrázek 6: Text kdy je část *rem* obalená do tagů

Budeme tedy muset použít pomocné pole, které bude uchovávat informaci, kde se znaky z původního textu momentálně nachází. Mapuje nám původní pozice na pozice nové. Jak takové pole vypadá můžeme vidět na obrázku 7.

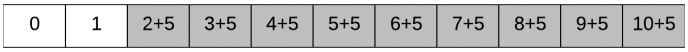
0	1	7	8	9	16	17	18	19	20	21

Obrázek 7: Výsledné pole

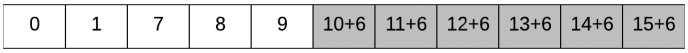
Výpočet pole je znázorněn na obrázcích 8 až 10. Ke všem prvkům nacházejících se za pozicí, na kterou byl tag vložen, přičteme delku počátečního tagu ve znacích. V našem případě počáteční tag je <*tag*>, který má délku 5 znaků, takže k prvkům na indexu 2 a výše přičteme hodnotu 5. To samé provedeme pro ukončovací tag </*tag*>, který má délku 6 znaků. Dostaneme tabulku na obrázku 7, která mapuje index z původního textu na nové tak, že ignoruje námi vložené tagy.

0	1	2	3	4	5	6	7	8	9	10

Obrázek 8: Počáteční pole



Obrázek 9: Počítání pozic pro začáteční tag



Obrázek 10: Počítání pozic pro ukončovací tag

4.4 Redukce textu

4.4.1 Stemming

Je proces, při kterém se slovo převádí na jeho kmen. Algoritmy, které toto řeší jsou vždy specializované na konrétní jazyk. Existují dva nejpoužívanější přístupy. První je, že algoritmus používá slovník obsahující kmen pro všechna slova z konkrétního jazyka. Pro některé jazyky jako např. angličtina, by byl takový slovník poměrně malý, ale pro jazyky jako čeština, by jeho velikost už byla poměrně značná. Další nevýhodou je, že pokud konrétní slovo ve slovníku nemáme, tak ho nejsme schopni převést. Druhý přístup je, že algoritmus ze slova odstraňuje morfologické koncovky a předpony, např. v angličtině by jsme mohli odstraňovat koncovku *ing*. V češtině může ale nastat několik situací, které jsou složité ošetřit, např. kmen slova může končit na morfologickou koncovku, takže stemmer by ji nesprávně odstranil. Ve většině případů nám tedy stemmer vrací ne úplně gramaticky správný kmen. To, ale pro naše účely stačí, důležité je, aby se podobná slova namapovala vždy na stejný kmen.[20][8]

4.4.2 Stop slova

V češtině máme slova, která nenesou samy o sobě žádný význam. Jedná se například o předložky, spojky a některá další slova. Těmto slovům nepřikládají velký význam, nebo je úplně ignorují internetové vyhledávače při indexování stránek. Příklady takových slov mohou být: aby, ale, nebo, bude, jak, jaké, pak, po, pod, podle. Ve svém projektu tyto stop slova odstraňuji pomocí regulérního výrazu.[17]

4.4.3 Normalizace

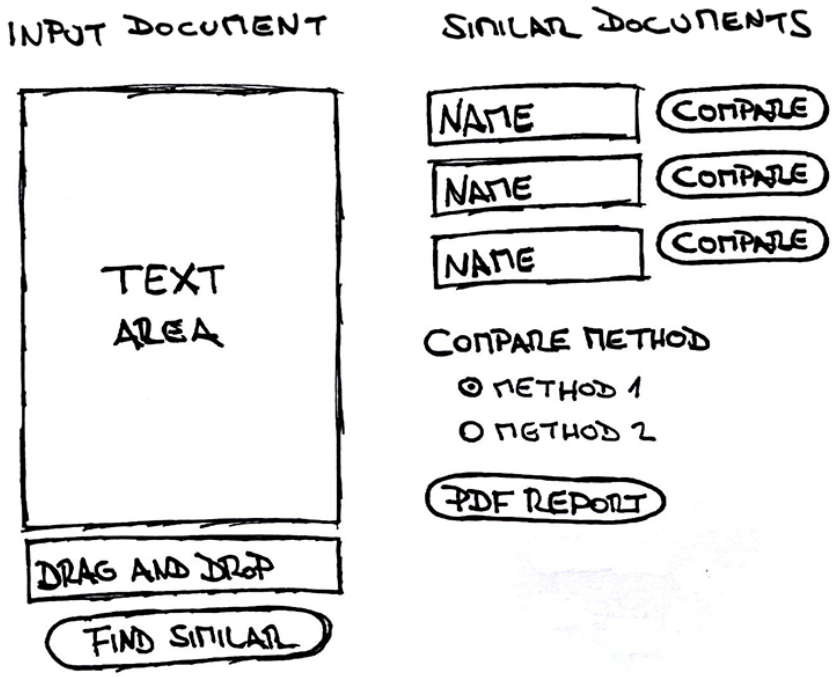
Za ideálních podmínek by byl porovnávaný text jen slova oddělené jednou mezerou, tomu tak ale většinou není. Text vrácený ze služby vyvinuté na Katedře informatiky je plný vícenásobných mezer, nových řádků a tabulátorů. Tyto znaky je třeba nahradit jen jednou mezerou a odstranit počáteční a koncové mezery. To lze velice jednoduše dosáhnout pomocí regulérních výrazů, části textů které odpovídají výrazu `\t|\n|\r` se nahradí mezerou, poté tento výraz `[ ]{2,}` nahradí všechny více násobné mezery jen jednou. Dále ještě text převedeme na malá písmena.

5 Popis vlastního řešení

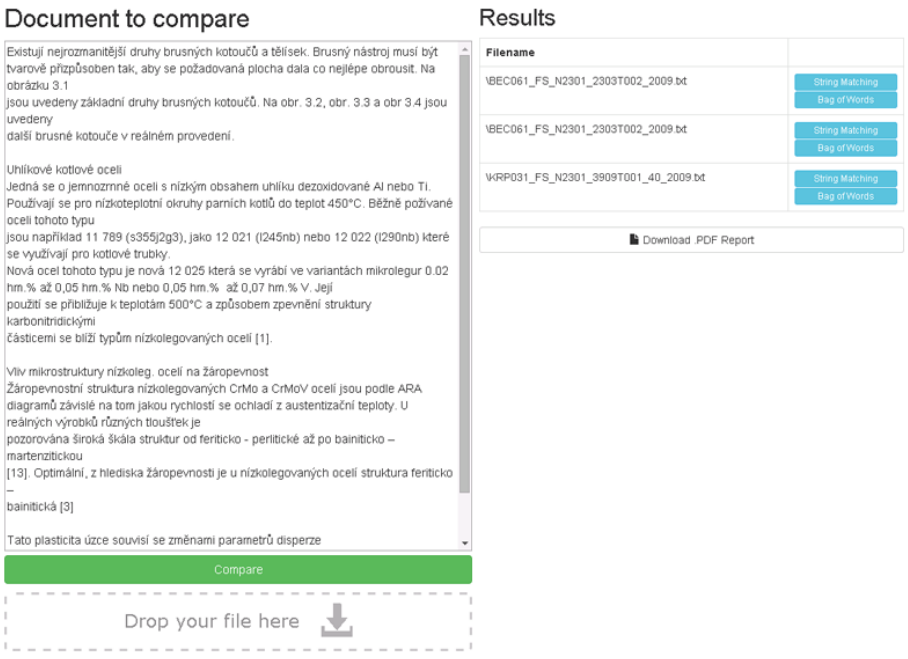
5.1 Specifikace

Aplikace bude umožňovat jednoduše hledat podobné dokumenty a v nich pak zobrazit na základě různých metrik společné části textů, u kterých je podezření na plagiarismus. Aplikaci budou moc využívat uživatelé s platným jménem a heslem. Pro jejich ověření budu využívat školní LDAP systém. Byla mi poskytnuta třída, která ověření údajů proti LDAP řeší.

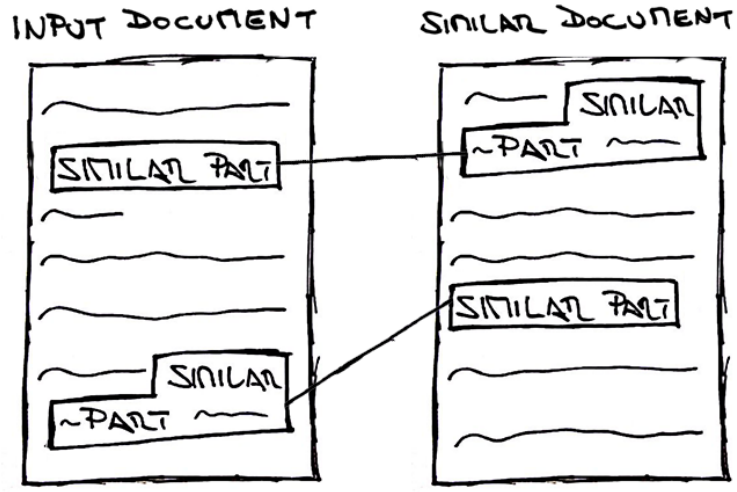
Po přihlášení se uživatel dostane do hlavního okna aplikace obrázek 11, kde může zadat text k porovnání. Po odeslání se mu zobrazí podobné dokumenty, ve kterých si bude moct zobrazit společné části na základě různých metrik, nebo si stáhnout výslednou zprávu o porovnání ve formátu pdf. Při výběru způsobu porovnání je uživatel přesměrován na obrazovku, která je rozdělena na půl obrázek 13. V levé části uživatel vidí svůj dokument a v pravé dokument, který je mu podobný. Mezi dokumenty jsou barevně vyznačeny podobné části



Obrázek 11: Návrh hlavního okna aplikace



Obrázek 12: Hlavní okno aplikace



Obrázek 13: Návrh okna pro porovnání dvou dokumetnu

Obrázek 14: Porovnání dvou dokumnetů

5.1.1 Vstupy

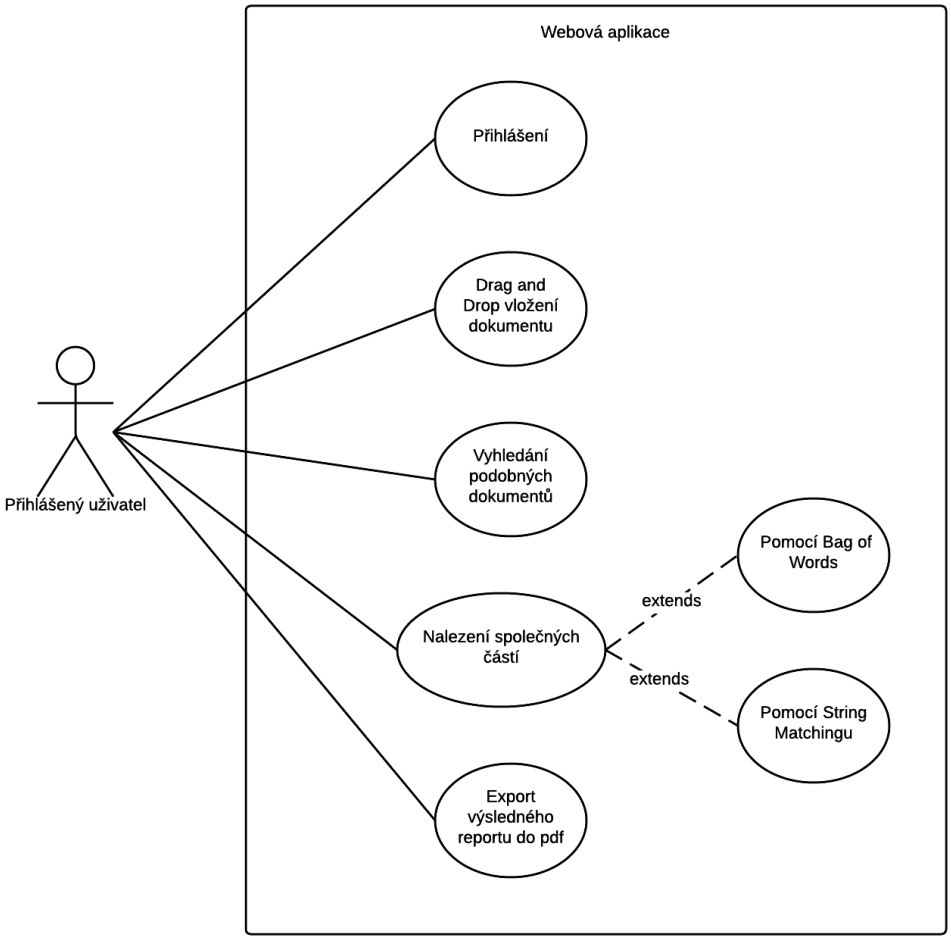
Uživatel na hlavní stránce naší aplikace bude moc vepsat, nebo pomocí drag and drop funkce vložit dokument ve formě textového souboru.

5.1.2 Výstupy

Po zaslání dokumentu se mu zobrazí tabulka s podobnými dokumenty a možností je porovnat na dvou úrovních, a to pomocí String Matching nebo Bag of Words. Po výběru přístupu porovnání je uživatel přesměrován na stránku, kde jsou vyznačene společné části dvou dokumentů. Na pravé straně uživatel vidí text svého vloženého dokumentu a na levé text dokumentu nalezeného v naší databázi. Na obou dokumentech jsou vyznačeny společné části pomocí stejné barvy. Pro pohodlnější hledání konkrétní části uživatel může kliknout na obarvený text v levém dokumentu a pravý se mu automaticky zascrolluje na odpovídající část. Uživatel si také může stáhnout zprávu o porovnání ve formátu .pdf.

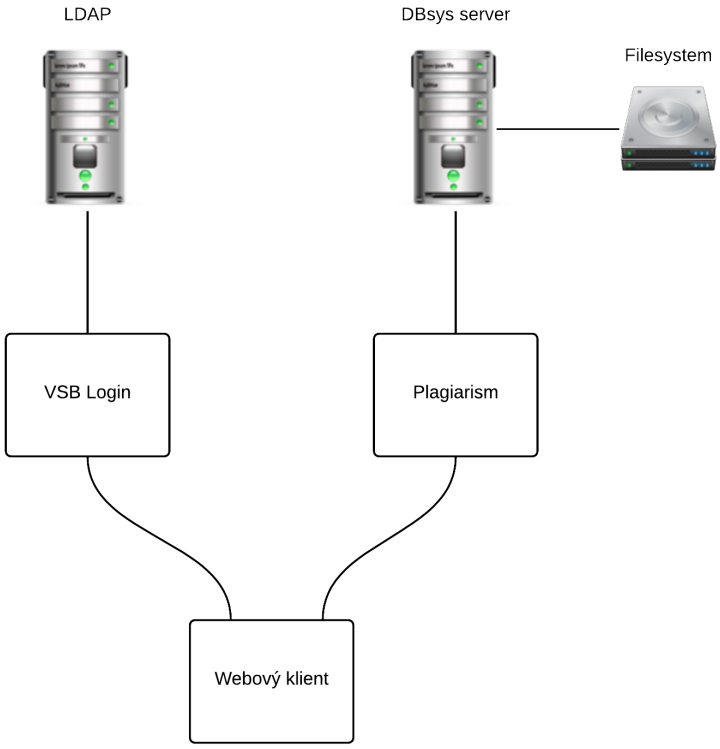
5.1.3 Use Case diagram

- Přihlášení: Přihlášení za pomocí LDAP systému do naší aplikace.
- Drag and Drop: Možnost přetáhnout textový dokument do okna naší aplikace a vyextrahování jeho obsahu do textového pole pro vstupní dokument.
- Vyhledání podobných dokumentů: Pokud existují jeden nebo více podobných dokumentů tak se uživateli zobrazí ve formě seznamu.
- Nalezení společných částí: Barvené zvýraznění podobných částí podle konrétní metriky uživateli
- Export do pdf: Export závěrečné zprávy do pdf



Obrázek 15: Use case diagram

5.2 Architektura



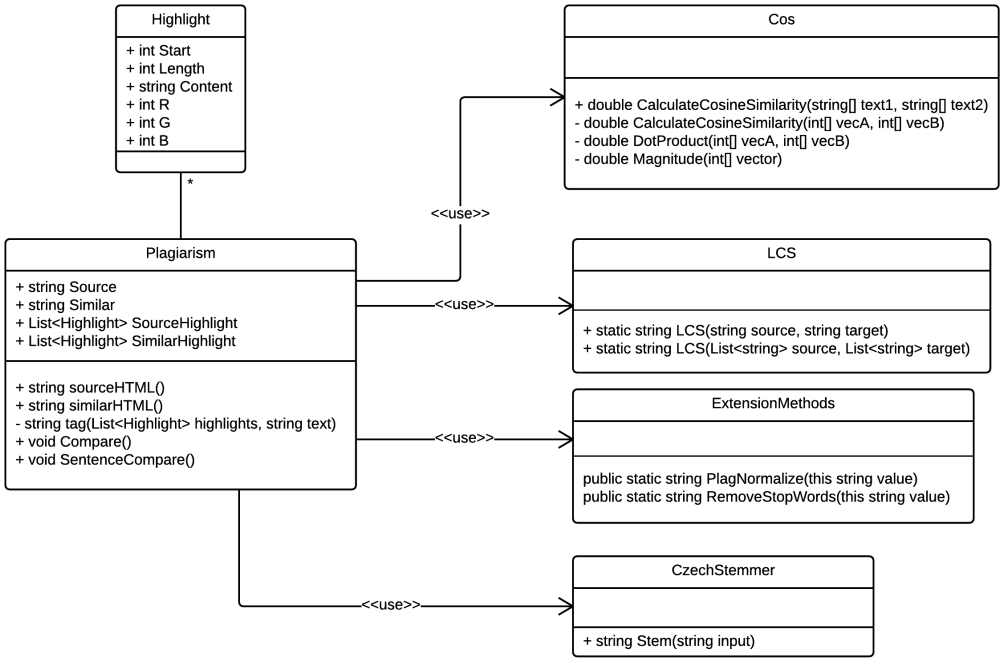
Obrázek 16: Rozložení

Projekt se skládá z doménové vrstvy, ve které je umístěna veškerá business logika včetně části, která komunikuje se službou vyvinutou na Katedře informatiky. Služba vyhledává podobné dokumenty ve file systému a určí míru podobnosti jako počet nalezených n-gramů, který spolu s absolutní cestou k dokumentu vrací.

5.3 Analýza

5.3.1 Třídní diagram

Zde vidíme zjednodušený třídní diagram, na kterém je zobrazena nejdůležitější třída *Plagiarism*, která obsahuje text obou dokumentů spolu s informací, které části textu je třeba zvýraznit. Tuto informaci generuje, buď metoda *Compare()*, nebo *SentenceCompare()*, záleží, kterým způsobem chceme texty porovnávat. Metoda naplní list objektu *Highlight* ve kterých je obsažena barva a pozice konrétního zvýraznění. Poté, si můžeme nechat vygenerovat otagovaný text pro zobrazení na webové stránce pomocí metod *SourceHTML()* a *SimilarHMTL()*. Třída ke svému fungovaní využivá několik dalších tříd, které jsou znázorněny v diagramu pomocí vazby se slovem «use».

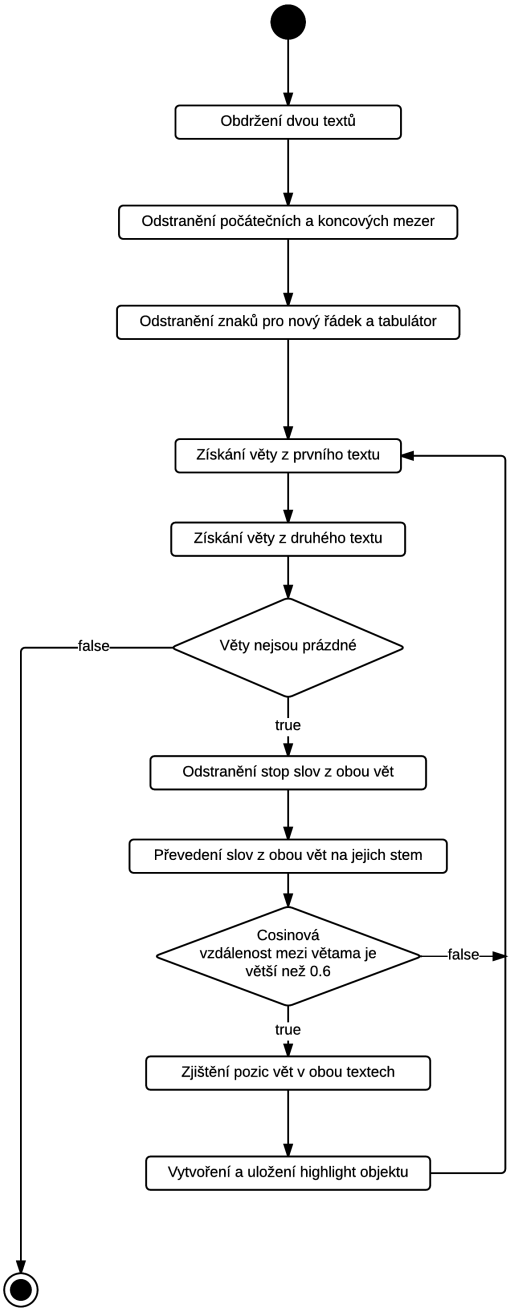


Obrázek 17: Zjednodušený třídní diagram

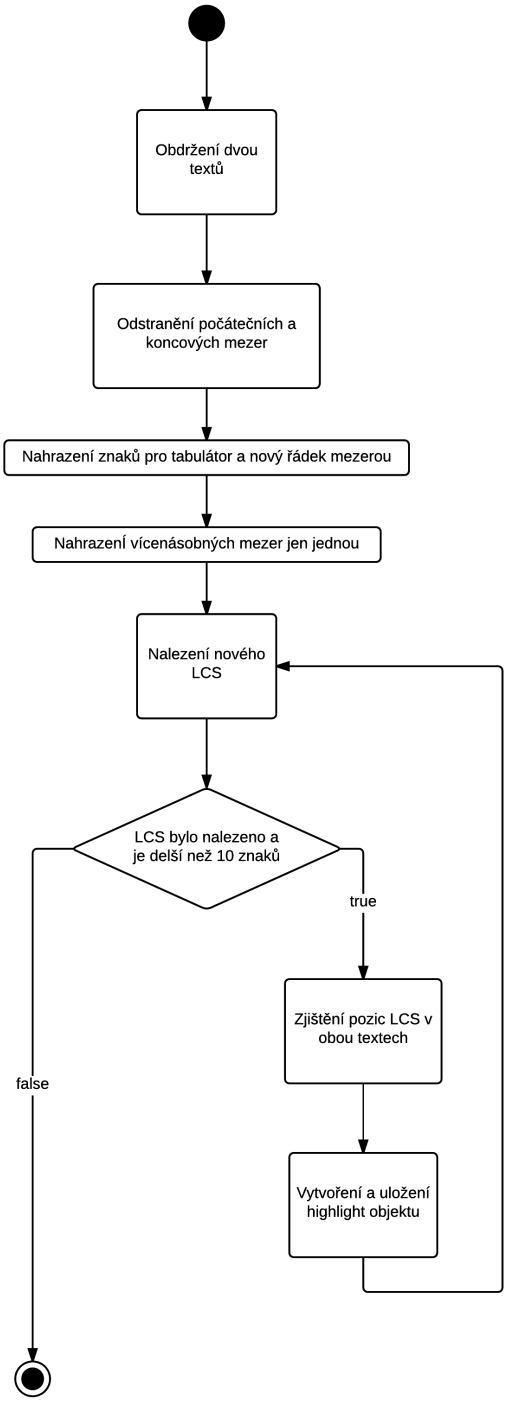
5.3.2 Aktivitní diagram

Na obrázku 18 vidíme aktivitní diagram popisující průběh nalezení společných částí za pomocí Bag of Words. Ze vstupních textů odstraníme vícenásobné mezery a nepotřebné znaky, poté si texty rozdělíme na věty. Odstraníme z nich stop slova a zbývající slova převedeme na jejich kořen. Tyto věty mezi sebou porovnáme a zjistíme jestli je jejich $CP > 0.5$ pokud ano, tak si uložíme pozici této věty vzhledem ke vstupnímu textu a proces opakujeme dokud jsme mezi sebou neporovnali všechny věty.

Obrázek 19 popisuje postup nalezení společných částí pomocí String Matching. Vstupní texty upravíme stejně, jako při hledání za pomocí Bag of Words, včetně stemmingu a odstranění stop slov, a poté hledáme všechny LCS, které jsou dělší než 10 znaků. Při nalezení LCS si uložíme pozici nalezené části vzhledem ke vstupnímu textu.



Obrázek 18: Nalezení společných částí pomocí Bag of Words



Obrázek 19: Nalezení společných částí pomocí String Matching

5.4 Implementace

K implementaci webové aplikace jsem využil ASP.NET framework využívající MVC vzor. Jako vývojové prostředí jsem používal Visual Studio 2012. Samotný vývoj aplikace jsem prováděl na vzdálené ploše Windows 7 běžící na serveru dbsys, na kterou jsem se připojoval za pomocí VPN. Pluginy které jsem v mém projektu využíval:

- jQuery - Framework usnadňující práci s javascriptem
- Codaxy.WkHtmlToPdf - Třida která umožňuje export ASP.NET MVC View do pdf
- jQCloud - jQuery knihovna vykreslující word cloud obrázek 20
- StemmersNet - Stemmer podporující několik jazyků včetně češtiny

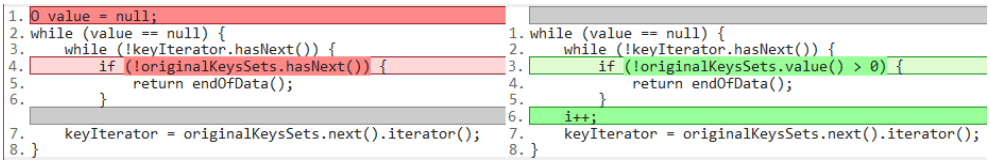
Obrázek 20: Ukazka word cloudu pro text lorem ipsum

6 Ostatní systémy

Snažil jsem se najít podobné systémy pro porovnávaní dvou dokumentů, které pro hledání společných části využívají nějaké pokročilejší techniky, z kterých bych se mohl něco přiučit nebo je mohl zakomponovat do svého projektu. V tomhle ohledu jsem neměl moc štěstí, podařilo se mi najít jen tyto řešení:

6.1 Diff utility

Diff je příkaz z unixových systémů, který se používá pro zjištění rozdílu mezi dvěma texty. Je obvzláště užitečný pro porovnávání zdrojových kodu, využívá ho například známá webová služba pro verzování zdrojových kodu Github, používá obdobu příkazu diff pro zobrazení rozdílu mezi dvěmi verzemi kodu obrázek 21 [4]



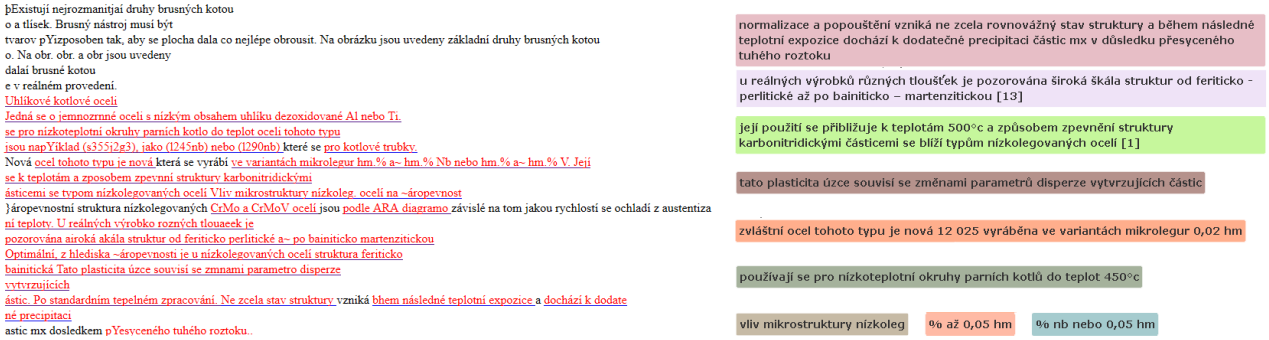
Obrázek 21: Zobrazení změny pomocí diff utility

6.2 Dupli Checker

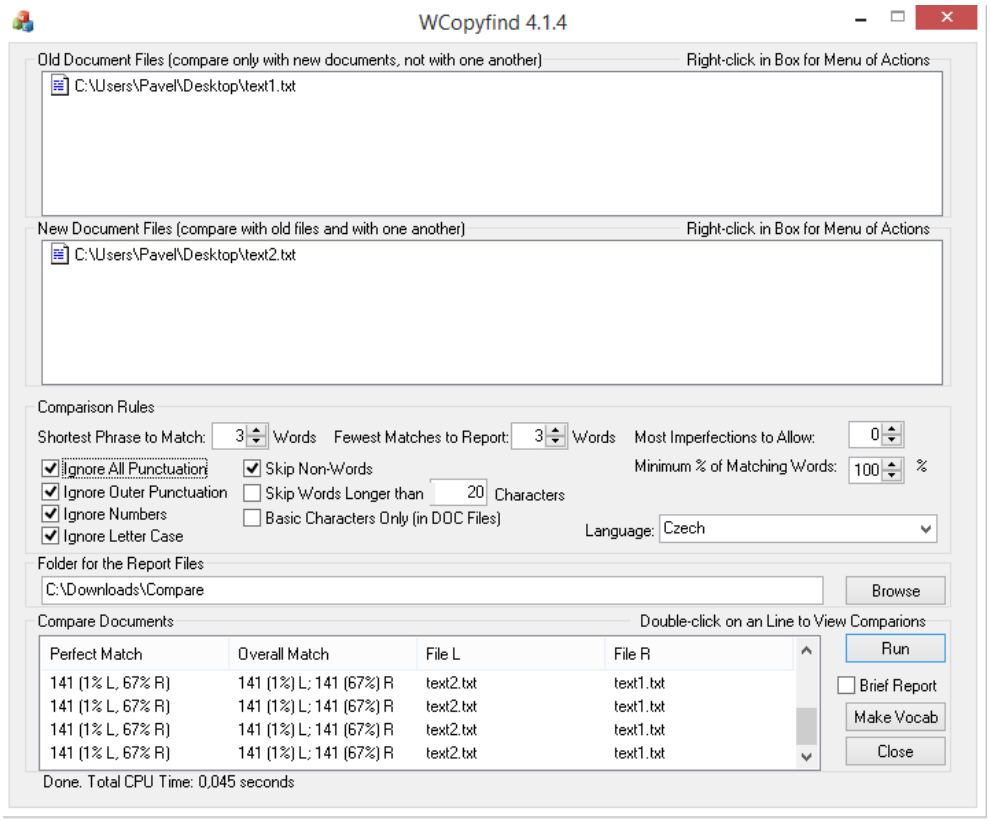
Je webová aplikace podobná mému projektu, až na to, že má problémy s češtinou a nevrací vždy největší společnou část dvou textů. Podle toho co se dá vyčíst z informací na jejich stránkách, text dělí na jednotlivé věty, které mezi sebou blíže nespecifikovaným algoritmem porovnává. Moje aplikace při využití string matchingu nalezne stejný počet společných částí, až na to, že moje jsou větší. [6]

6.3 WCopyfind

Asi nejpovedenější aplikace pro desktop s poměrně rozsáhlým nastavením 23. Můžeme programu zadat cestu do složky s autentickými dokumenty, nebo obecně dokumenty proti kterým chceme porovnávat. Má podporu dokumentů v českém jazyce i když je po porovnání zobrazí špatně, ale samotné porovnání provede správně. Je velice rychlá a má 64bit verzi, která je ještě o něco rychlejší. Na stránkách není přesně popsáno jakým způsobem se porovnání provádí, ale je open-source, takže princip funkce je možný vyčíst z kodu, o to jsem se pokoušel, ale bezúspěšně. Při testování dokumentů, které používám v mé aplikaci, dosahuji velice podobných výsledků. Na obrázku 22 je porovnání výstupu z WCopyfind a mé aplikace. [12]



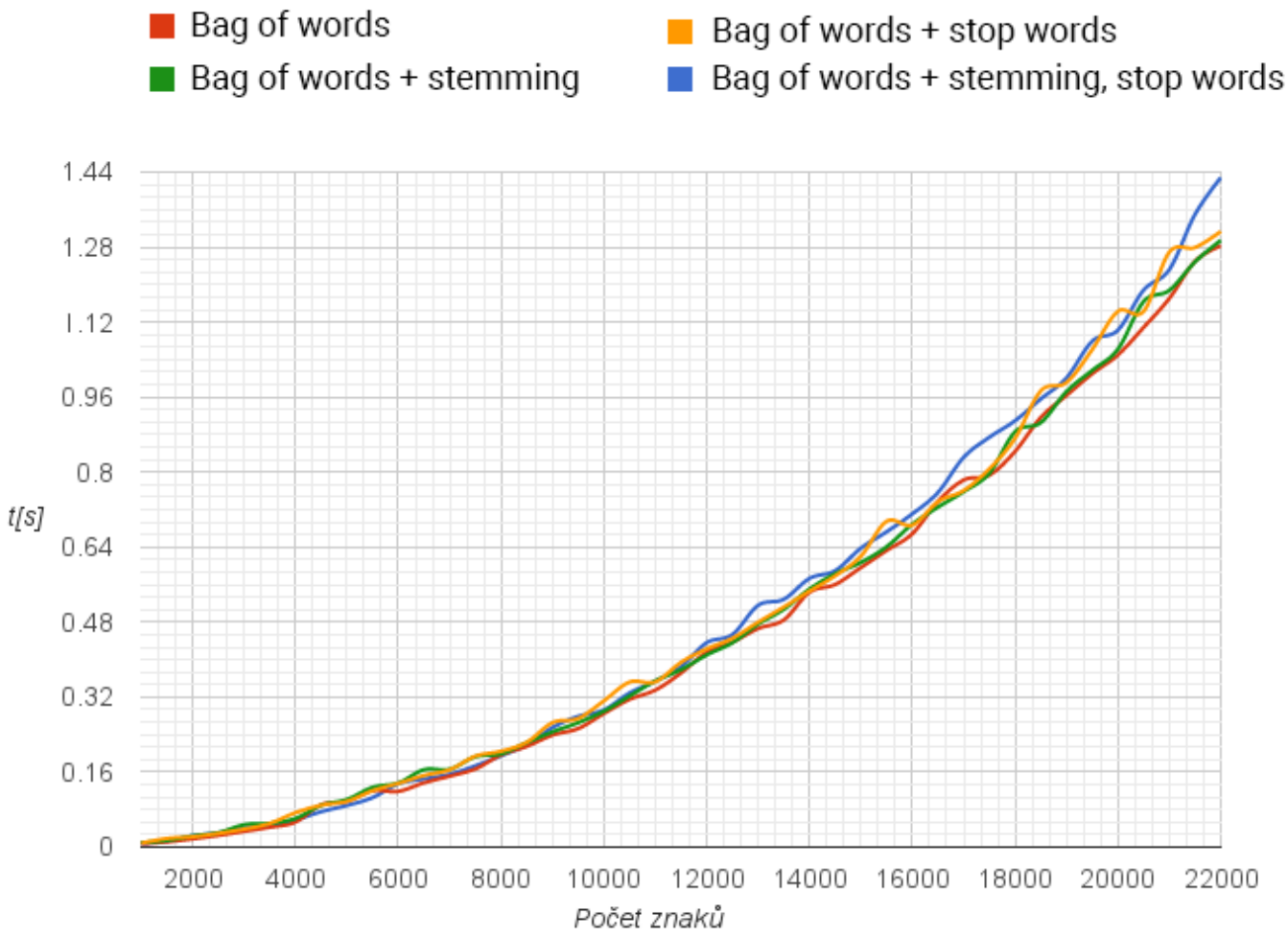
Obrázek 22: Porovnání výstupu z mé aplikace a WCopyfind



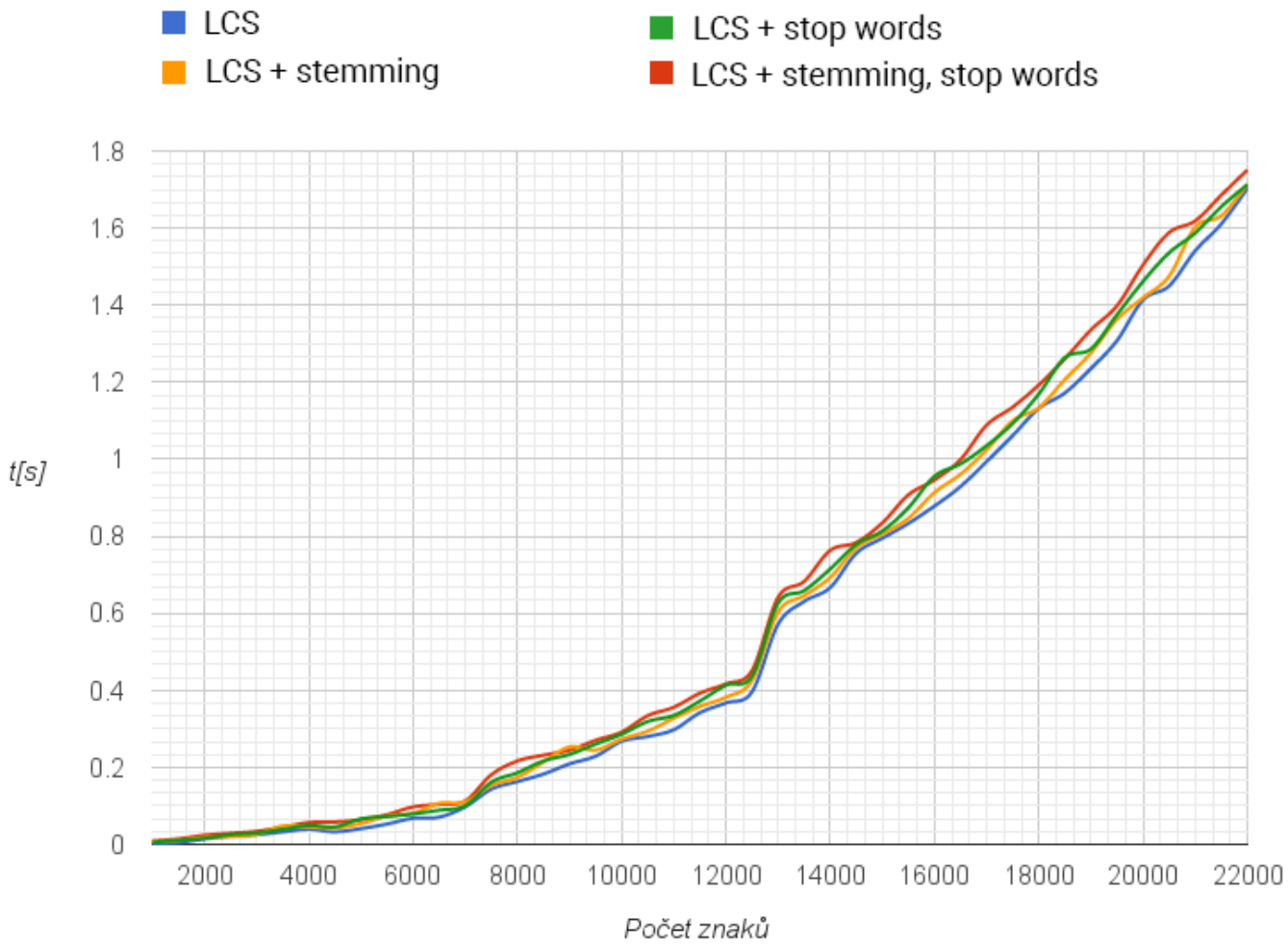
Obrázek 23: Hlavní okno aplikace WCopyfind

7 Testy

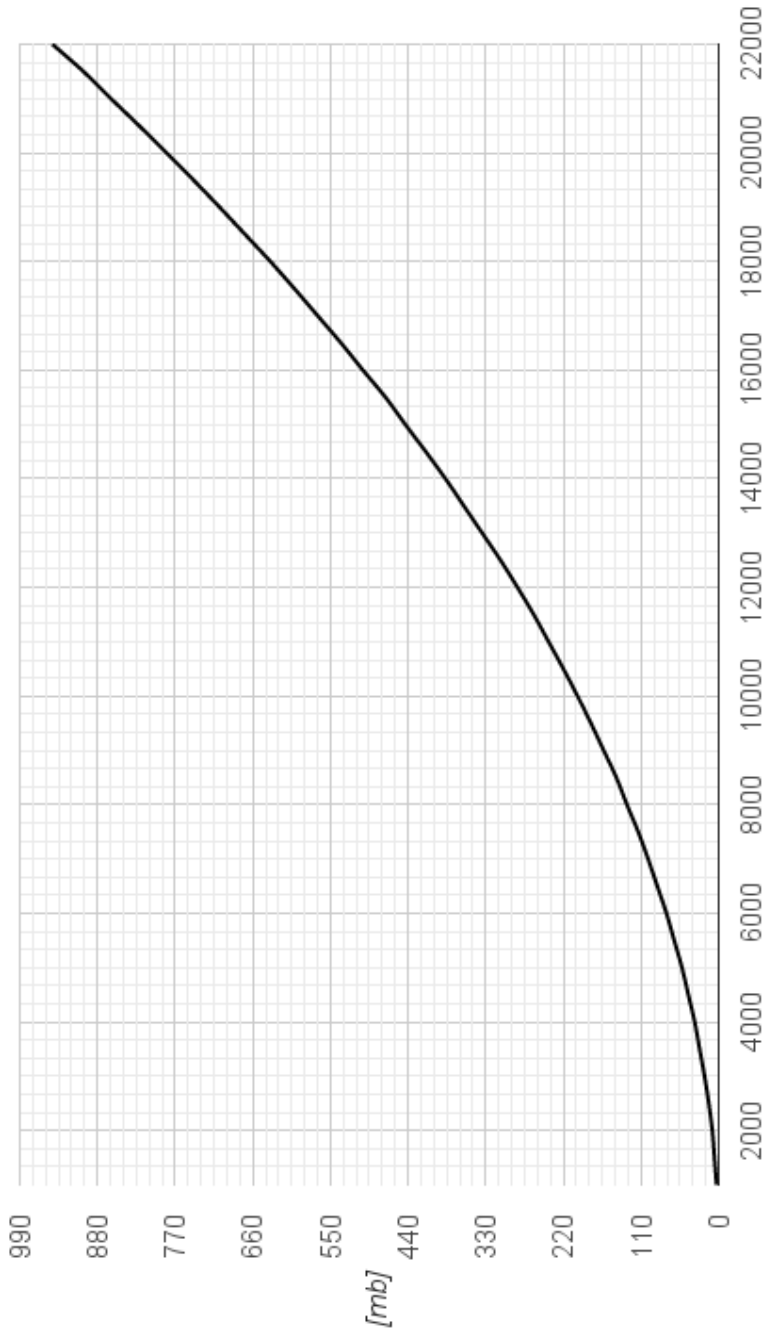
Při testování algoritmů jsem se zajímal o vztah mezi délkou textu a časovou náročností. Při porovnání grafů 24 a 25 můžeme vidět, že přístup Bag of Words je nepatrně rychlejší než String Matching. U obou grafů vidíme, že výsledný trend je složen ze čtyř téměř totožných průběhů, a to z průběhu pro samotný algoritmus, použití stop slov, použití stemmingu a použití jak stop slov, tak i stemmingu. Zjistili jsme, že použití technik pro redukci textu naší výslednou časovou náročnost ovlivňují jen zanedbatelně. Přibližné využití paměti u obou přístupu nepřesáhlo 30mb. Za to algoritmus pro suffix tree využíval až 1GB paměti. Na grafu 26 červený průběh ukazuje jak dlouho trva takový suffix tree vytvořit a modrý za jak dlouho v něm bylo nalezeno LCS. Můžeme vidět, že když už máme suffix tree vyvořený, tak hledání v něm je extremně rychlé. Černý průběh je celkový čas potřebný k nalezení LCS, tj. součet času k vytvovření suffix tree a nalezení LCS. Test byl prováděn na dvou různých českých textech, každý o délce 22000 znaků, kde jsem pro každých 500 znaků měřil délku běhu algoritmu. Program jsem testoval na počítači s procesorem i5 - 2430m 2.40 GHz, RAM o velikosti 4GB a SSD diskem.



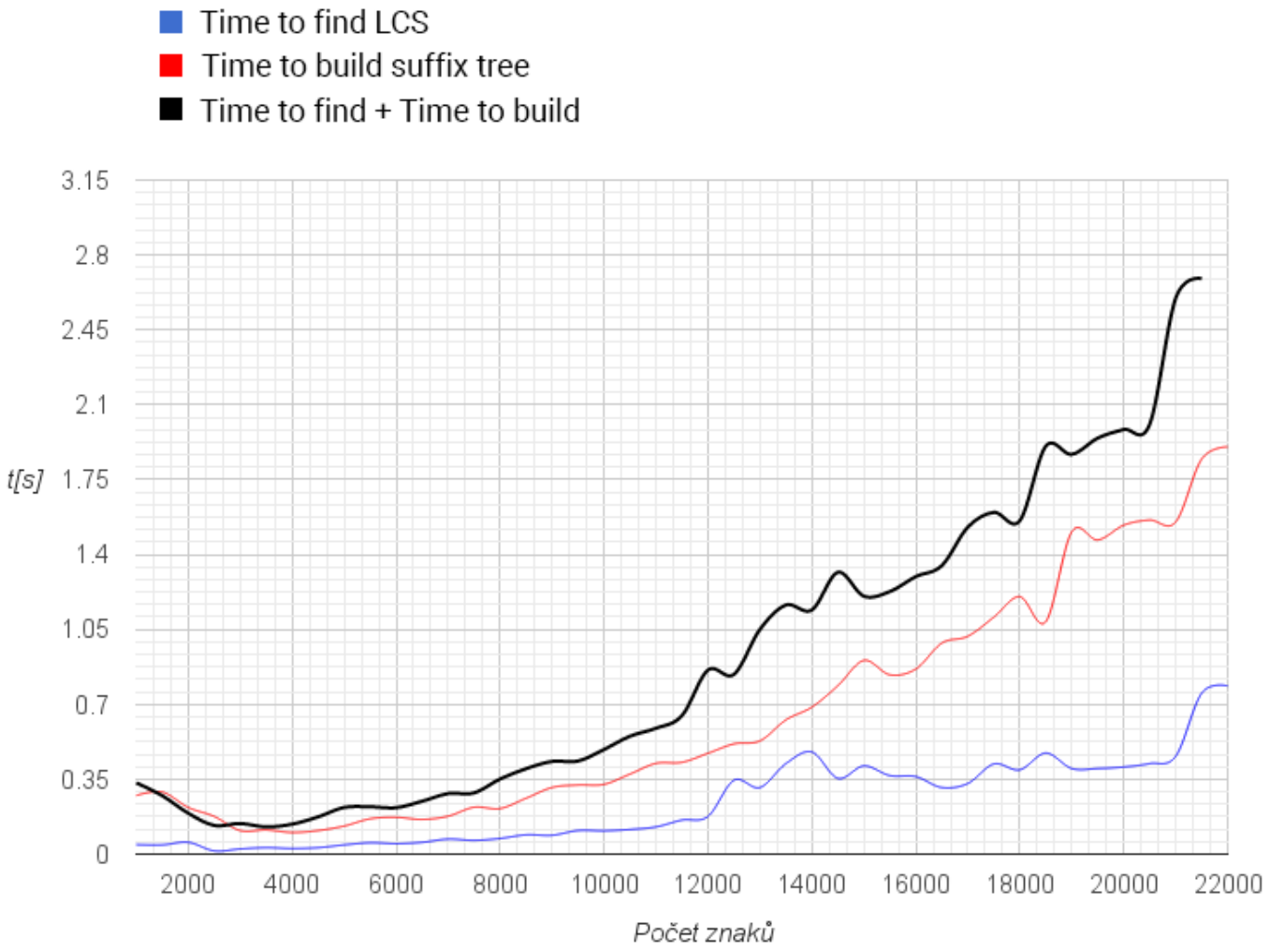
Obrázek 24: Závislost délky textu na čase pro Bag of Words



Obrázek 25: Závislost délky textu na čase pro String Matching



Obrázek 26: Paměťová náročnost pro Suffix Tree



Obrázek 27: Časová náročnost pro Suffix Tree

8 Závěr

Hlavním cílem této práce bylo vytvořit webovou aplikaci pro hledání podobných, nebo plagiovaných dokumentů.Rozpoznání takových dokumentů není úplně jednoznačný proces. Neexistuje program, který by na vstupu dostal dokument a jeho výstupem by byla odpověď jestli se jedná o plagiát či nikoli. Vždy takový dokument porovnáváme vzhledem k něčemu, pokud možno k databázi dokumentů, které jsou považovány za autentické. Mezi nimi se poté snažíme hledat shody a podobnosti pomocí různě sofistikovaných přístupů, od těch nejjednoduších jako ty, které hledají doslovnou shodu a nebo porovnávají citace až po komplexní algoritmy, které dokáži určit unikátní styl autorova psaní. Všechny tyto přístupy usnadňují práci člověku, který se plagiát snaží najít a jsou pro něho jakýmsi ukazatelem, na které práce by se měl zaměřit, poslední slovo při rozhodování jestli něco plagiát je, má vždy právě on.

Jak jsem se již zmínil, přístupů existuje spousta. Já se zaměřil právě na ty nejjednodušší, které tento problém řeší. Konrétně dva, první, který hledá doslovnou shodu a druhý, který hledá podobné věty mezi dvěma texty. Po nastudování teorie obou přístup a zvážení několika způsobů jak se dají řešit, se mi podařilo algoritmy naimplementovat. Musel jsem je několikrát upravovat a hledat v nich prostor pro zlepšení, protože aplikace klade velké nároky na rychlost, zvláště při větších textech. Musím říct, že po několika úpravách se mi podařilo dosáhnout i s použitím takto základních přistupů velice dobrých výsledků. Zvláště přístup pro porovnávaní vět, dokáže odhalit podobné věty, které by člověku mohli uniknout.

V práci jsem popsal postup tvorby mé aplikace od teorie, specifikace požadavků až po samotný popis důležitých algoritmů spolu s jednoduchou analýzou. Ke konci jsem provedl testy nad zkušebními daty a výsledky zakreslil do grafu.

Při tvrobě této práce jsem se dozvěděl spostu nových informací z oboru, který mi byl dosud neznámý. Téma pro mě bylo zajímavé a v budoucnu bych se chtěl pokusit o implementaci pokročilejších přístupů jak plagiarismus odhalovat.

Pavel Kolář

9 Reference

[1] Online etymology dictionary, . URL http://www.etymonline.com/index.php?allowed_in_frame=0&search=plagiarism&searchmode=none.

[2] What is plagiarism, . URL http://www.lib.usm.edu/legacy/plag/whatisplag.php.

[3] Plagiarism detection. URL https://en.wikipedia.org/wiki/Plagiarism_detection.

[4] Diff checker. URL https://www.diffchecker.com/.

[5] Skalární součin. URL http://www.matematika.cz/skalarni-soucin.

[6] Dupli checker. URL http://www.duplichecker.com/comparison.

[7] How to use internal citations. URL http://www.wikihow.com/Use-Internal-Citations.

[8] Stemming. URL http://en.wikipedia.org/wiki/Stemming.

[9] The plagiarism spectrum: Tagging 10 types of unoriginal work. URL http://www.turnitin.com/assets/en_us/media/plagiarism_spectrum.php.

[10] What is plagiarism. URL http://www.plagiarism.org/plagiarism-101/what-is-plagiarism/.

[11] Bill Bell. Example of cosine similarity, November 2009. URL http://stackoverflow.com/a/1750187.

[12] Lou Bloomfield. Wcopyfind software. URL http://www.duplichecker.com/comparison.

[13] Niall Gallagher. Concurrent-trees, January 2015. URL https://code.google.com/p/concurrent-trees/.

[14] Jon Galloway. *Professional ASP.NET MVC 4*. Wrox, 2012. ISBN 978-1118348468.

[15] GeeksforGeeks. Dynamic programming | set 29 (longest common substring). URL http://www.geeksforgeeks.org/longest-common-substring/.

[16] GeeksforGeeks. Suffix tree application 5 - longest common substring, January 2015. URL http://www.geeksforgeeks.org/suffix-tree-application-5-longest-common-substring-2/.

[17] Michal Janík. Česká stop slova(stop words), March 2009. URL http://www.michaljanik.cz/oblibene/stop-slova.

[18] University of California. The meaning and prevention of plagiarism. URL http://cai.ucdavis.edu/plagiarism.html.

[19] Christian S. Perone. Machine learning :: Cosine similarity for vector space models (part iii), September 2013. URL http://blog.christianperone.com/?p=2497.

[20] Cambridge University Press. Stemming and lemmatization, April 2009. URL http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.