Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Milan Freml, David Chalupa, Marek Mego,
Peter Mindek, Michal Noskovič a Matej Sabo

Podpora kontroly plagiarizmu

Tímový projekt I.

Študijné programy: Informačné systémy, Softvérové inžinierstvo
Vedúca projektu: Mgr. Daniela Chudá, PhD.
Ak. rok: 2009/10

Obsah

1 Úvod do problematiky

1.1 Čo je plagiarizmus?

Plagiarizmom sa nazýva využívanie cudzích prác, obrázkov alebo myšlienok bez toho, aby sa na ne autor patrične odvolával. Je to teda prisvojenie si časti cudzej práce. S plagiátorstvom sa dnes bežne stretávame hlavne v školstve, no neobchádza ani iné oblasti každodenného života. Postoj k nemu je v dnešnej spoločnosti čoraz menej tolerantný. Keďže sa s plagiátorstvom stretávame najčastejšie v školstve, jedná sa najmä o kopírovanie častí cudzích bakalárskych alebo diplomových prác bez odvolania sa na zdroj. S nástupom internetu sa možnosti získavania informácií rozšírili, čo paradoxne do veľkej miery napomohlo k rozšíreniu fenoménu plagiátorstva.

Autor, ktorý čerpá myšlienky z iných prác, má povinnosť uviesť k zdroju týchto myšlienok pôvodného autora a článok alebo prácu, z ktorej čerpal. To môže urobiť viacerými spôsobmi. Najčastejšie sa používajú úvodzovky na vymedzenie citovaného textu, poznámka pod čiarou a použitá literatúra, ktorá sa väčšinou uvádza na konci práce.

V spoločnosti existuje snaha autorov plagiátorov odhaliť. Táto snaha však naráža na tvorivosť a vynaliezavosť plagiátorov. Tí sa snažia cudzie myšlienky formulovať tak, aby prípadný čitateľ, respektíve osoba, ktorá prácu bude kontrolovať, nemala pochyby o tom, že práca je originálna a všetky myšlienky pochádzajú od autora práce.

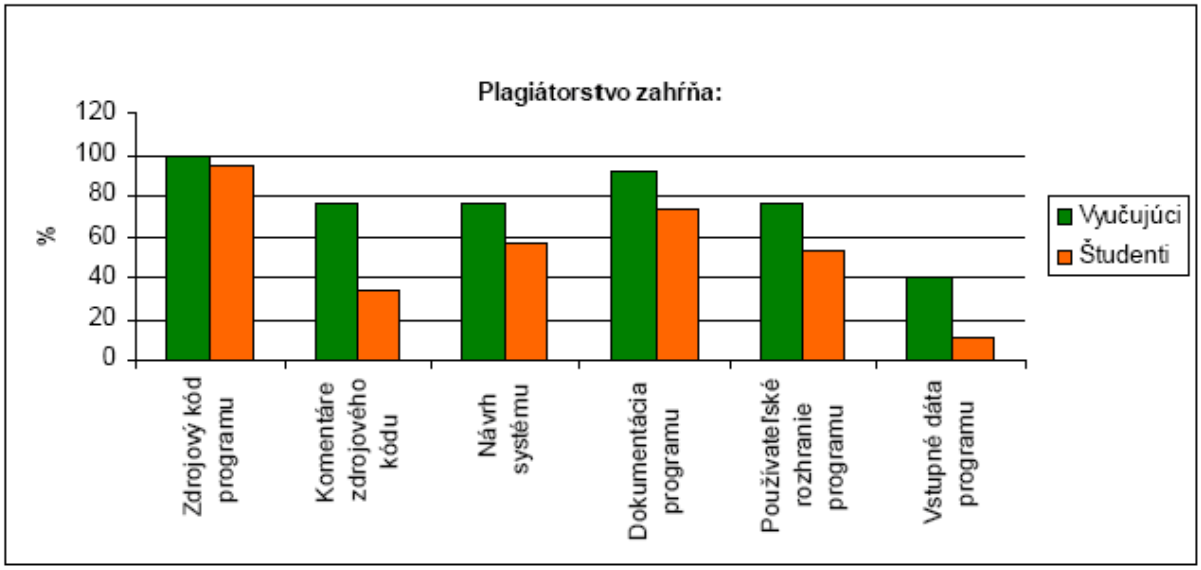
Za týmto účelom vzniklo veľa systémov a programov, ktoré majú za úlohu kontrolovať práce alebo zdrojové kódy a hľadať v nich podobnosť. Ak je podobnosť medzi dvoma dokumentmi nad určitou percentuálnou hranicou, sú tieto dokumenty označené ako podozrivé a je na učiteľovi, aby urobil následnú kontrolu. Niektoré programy mu môžu túto kontrolu značne uľahčiť, a to tým, že graficky vyznačia v dokumentoch časti, ktoré sa nachádzajú v oboch prácach.

1.2 Formy plagiarizmu a ich špecifiká

1.2.1 Zdrojové kódy

Používanie cudzích zdrojových kódov bez odkazovania sa na pôvodného autora sa zaraďuje taktiež medzi plagiátorstvo. Jedná sa totiž o použitie cudzích myšlienok, ktoré nie sú vyjadrené vo forme textu, ale vo forme zdrojového kódu. K softvérovým projektom sa často viažu aj textové dokumenty, ktoré obsahujú napríklad analýzu, dokumentáciu alebo návod na používanie výsledného programu. K týmto dielam, ktoré sú nedielnou súčasťou práce autora, sa taktiež vzťahuje jeho duševné vlastníctvo a nemôžu byť použité bez uvedenia zdroja.

Na obr. 1 môžeme vidieť výsledok ankety, ktorú vykonali Bianka Kováčová a Pavol Humay vo svojich diplomových prácach, v ktorých sa zaoberajú plagiátorstvom. Z nej je vidieť, čo študenti a profesori zahŕňajú pod pojem plagiátorstvo.



Obr. 1: Anketa na tému plagiátorstva [1]

Z prieskumu je zrejmé, že väčšina respondentov pokladá zdrojový kód za veľmi dôležitý a je potreba sa naň pri odvodených prácach odvolávať.

Zisťovanie podobnosti v zdrojových kódoch je veľmi náročná úloha, nakoľko existuje veľa metód, pomocou ktorých sa dá pravdepodobnosť na odhalenie podobnosti znížiť. Jednoduché plagiáty dokáže odhaliť aj osoba, ktorá porovnáva dva súbory a dokáže na 100% povedať, že sa jedná o plagiáty, cez zložitejšie až po tie najzložitejšie techniky, ktoré sa dajú odhaliť až za pomoci sofistikovaných nástrojov.

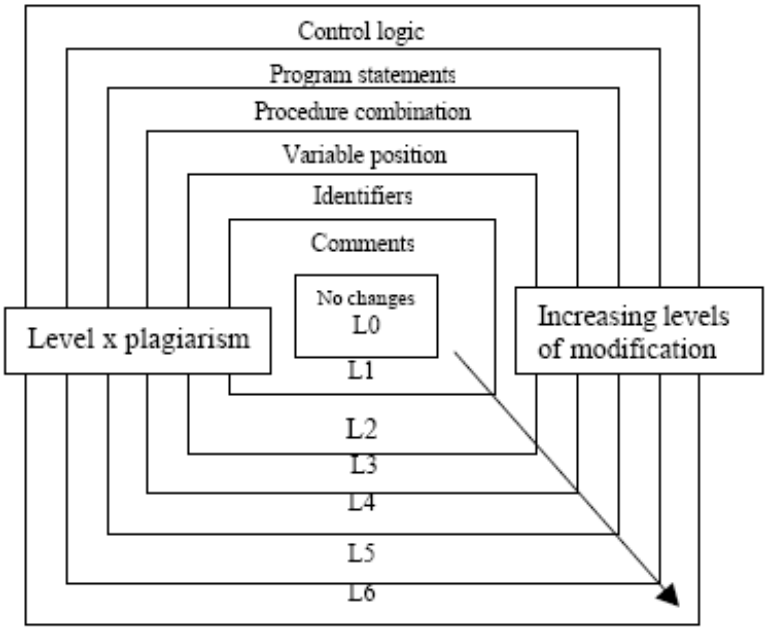
Všetky tieto techniky sa snažia pozmeniť pôvodný zdrojový kód tak, aby bol na pohľad rozdielny, ale výsledok programu bol nezmenený. To znamená, že pôvodný aj odvodený program na rovnaký vstup dajú rovnaký výstup.

Podľa Paula Clougha [Clough, 2000] môžeme zmeny v zdrojových kódoch rozdeliť podľa ich zložitosti na:

1. Žiadna zmena
2. Zmena komentárov
3. Zmena identifikátorov
4. Zmena pozície premenných
5. Kombinácia procedúr a funkcií
6. Správanie programu
7. Kontrola logiky programu

Predošlý zoznam sa dá graficky zobraziť pomocou prehľadného grafu, ktorý je zobrazený na obr. 2.

[1] Bianka Kováčová: *Podobnosť a rozpoznanie opísaného programu.* Diplomová práca, 2009.



Obr. 2: Vykonané zmeny v zdrojových kódoch [Clough, 2000]

Z obrázka je zrejmé, že metód na maskovanie plagiátorstva je viac a líšia sa v zložitosti a v čase, ktorý je potrebný na ich zapracovanie.

Najmenej namáhavé sú postupy L1 a L2. Tu stačí v práci pozmeniť alebo vymazať len komentáre pôvodného autora. Taktiež je možné premenovať názvy premenných a funkcií, ktoré program používa. V prípade premenných plagiátori často menia ich typ. Napríklad prepíšu premennú, ktorá bola pôvodne deklarovaná ako celočíselný typ, na typ s desatinnou čiarkou. Medzi tieto jednoduché spôsoby môžeme taktiež zaradiť pridávanie medzier, tabulátorov, prázdnych riadkov do zdrojového kódu. Postupy, ktoré patria do tejto skupiny sú teda najjednoduchšie a preto je aj ich odhalenie najmenej zložité. Na ich odhalenie netreba často ani zložitý nástroj, ale stačí len manuálna kontrola.

Ďalšou skupinou zmien, sú zmeny, ktoré už vyžadujú zložitejšie prostriedky na odhalenie. Do tejto skupiny sa dajú zaradiť postupy L3 a L4. Jednou z možností je napríklad nahradenie podmienok za while cykly alebo naopak. Tento postup je znázornený na nasledujúcej ukážke.

```
3   if ($premenna){
4       doThis();
5   }
6
7   while ($premenna){
8       doThis();
9       break;
10  }
```

Obr. 3: Nahradenie podmienok za cykly

Jednou z možností, ako oklamať osobu, ktorá kontroluje prácu je nahradenie volaní funkcie samotnou implementáciou funkcie. Týmto sa stáva odvodená práca neprehľadnejšia a má samozrejme viacej riadkov. Do tejto skupiny môžeme taktiež zaradiť nahradenie matematických konštánt zápisom cez matematické operátory +,-,*,/. Konštantu 6 je možné zapísať napríklad ako 1*2+5-1.

Doteraz spomenuté postupy si nevyžadovali veľkú námahu a množstvo času na zrealizovanie. Preto aj ich odhalenie je viac menej nenáročné. Ostatné postupy z obrázku číslo 2 si už vyžadujú určitú námahu a na vytvorenie plagiátu je potrebné väčšie množstvo času. Sem môžeme zaradiť napríklad premiešanie poradia vzájomne nezávislých príkazov, blokov vo vetvách switchu, alebo zámenou poradia konštrukcie if else. Pokiaľ chceme spraviť kód neprehľadný, je možné doň pridať zbytočný kód, ktorý sa buď nevykoná, alebo jeho vykonanie nebude mať vplyv na výsledok

programu. Takéto sofistikované metódy majú za následok, že program, ktorý má hľadať podobnosť medzi zdrojovými kódmi, túto podobnosť nenájde, alebo bude podobnosť veľmi malá, čo nebude zo strany učiteľa vnímané ako plagiát.

1.2.2 Slovenské texty

S problematikou plagiátorstva sa častejšie spájajú texty ako programové kódy a preto je plagiátorstvo textov aj viacej rozšírené. V dnešnej dobe má človek obrovské množstvo zdrojov, z ktorých môže čerpať informácie. Za týmto účelom vzniklo na internete veľa webových stránok, ktoré ponúkajú už vypracované referáty, alebo za určitý poplatok je majiteľ stránky ochotný vypracovať danú prácu. Tento systém je využívaný najmä na stredných školách, kde sa problém plagiarizmu rieši menej ako na vysokých školách. Samozrejme všetky práce sa líšia kvalitou a prevedením, ktorým chcel autor zatajiť, že väčšina jeho práce nie je pôvodná.
Takéto práce môžeme rozdeliť podľa zdrojov z ktorých vznikli na:

- Kopírovaný plagiát
  Autor odvodenej práce v pôvodnej práci len zmenil meno pôvodného autora a názov samotnej práce. Tento spôsob môžeme zaradiť medzi najjednoduchšie formy plagiarizmu a preto sa takéto práce aj najľahšie odhaľujú či už za pomoci programu na to určeného, alebo učiteľa.
- Modifikovaný plagiát
  V tomto diele je zmenených viacej častí. Okrem mena samotného autora a názvu práce sa v texte zmenia formulácie, poprípade sa pomení poradie kapitol. Do práce sa pridajú zdroje z internetu, najčastejšie z elektronických encyklopédií ako je napríklad Wikipédia. Do práce je možné pridať ďalšie obrázky, alebo niektoré existujúce obrázky vymazať.
- Rešeršný plagiát
  V tomto diele sú použité myšlienky z viacerých zdrojov, ktoré sa týkajú problematiky diela. Samotné dielo je poskladané z viacerých originálnych prác iných autorov.

Odhaľovanie plagiátov v písanom texte

V nasledujúcom odseku sú popísané spôsoby, akými sa dá odhaliť plagiátorstvo v písaných textoch aj bez pomoci elektronických nástrojov. Každý jeden z nás má svoj špecifický slovník, ktorý používa pri písaní prác. Pomocou tejto vlastnosti sa dá jednoduchým spôsobom odhaliť práca, ktorá s veľkou pravdepodobnosťou nie je originálna, alebo obsahuje časti z iných prác. Medzi tieto vlastnosti patrí:

1. Použitie slov
   Každý jeden z nás používa svoj slovník a používa špecifické slová, ktoré nemusia používať ostatní. Analýzou týchto slov sa dá ľahko zistiť či je autorom textu samotný študent.
2. Zmena slov
   Tento bod má súvis s bodom 1. Ak študent vo svojej práci začne používať výrazy, ktoré predtým nepoužíval, je možné že daný text niekde prevzal, nakoľko človek často nemení svoj zaužívaný slovník a je pre každého viac menej jedinečný.
3. Celistvosť textu
   Tento bod hovorí o tom, či je text jeden celok, alebo niektoré jeho časti do textu nezapadajú.
4. Interpunkcia a gramatické chyby
   Pokiaľ majú dve rozdielne práce rovnaké gramatické chyby, alebo je v oboch v špecifických častiach alebo slovách vynechaná interpunkcia je veľmi pravdepodobné, že jedna z prác je skopírovaná.
5. Stavba viet
   Tak ako má každý jeden autor špecifickú slovnú zásobu, používa aj vlastné konštrukcie viet, v ktorých svoju slovnú zásobu používa. Autor môže používať veľmi dlhé vety, alebo naopak krátke vety, v ktorých zhrnie svoju myšlienku. Ak autor vo svojich prácach dlhodobo používa

jeden štýl písanie viet, je málo pravdepodobné, že ho bezdôvodne v inej práci zmení. Indikátorom plagiátu môže byť aj výskyt jednotlivých slov, ktoré sa v práci nachádzajú. Či už sa jedná o počet jednotlivých slov, alebo ich špecifické umiestnenie.

Vyššie spomenuté body samozrejme neplatia vždy a pre všetky práce. Je to len pomôcka pre učiteľa a rýchly spôsob, akým môže nájsť podozrivé práce. Niektoré z týchto spôsobov napríklad nemôžu fungovať na prácach, pri ktorých má väčšia skupina za úlohu napísať prácu na rovnakú tému. Vo väčšine prác sa budú vyskytovať rovnaké špecifické pojmy, ktoré s prácou súvisia a bez nich nie je možné prácu napísať. Medzi tieto pojmy sa dajú zaradiť:

- Mená
- Dátumy
- Geografické lokácie
- Špecifické pojmy z problematiky, ktorou sa práca zaoberá

Naopak, existujú slová, ktorých výskyt v práci môže opravujúcemu indikovať, že autor práce ho prebral z inej práce. Tieto slová sa nazývajú hapax legomena [Clough, 2000] slová. Jedná sa o slová, ktoré sú v práci použité práve raz.

Parafrázovanie

"Parafrázovanie je lingvistická operácia, ktorá spočíva vo vyjadrení toho istého obsahu, ako má východiskový výraz, konštrukcia alebo výpoveď, inými výrazovými prostriedkami." [FILIT] Medzi najčastejšie druhy parafrázovania zaraďujeme:

- Použitie synoným vo vete - Niektoré kľúčové slová sú nahradené ich synonymami.
  Pôvodná veta:
  Bol brutálne napadnutý a zavraždený.
  Pozmenená veta:
  Muž bol brutálne napadnutý a zabitý.
- Zmenenie typu vety
  Pri tomto spôsobe sa do pôvodnej vety vkladajú takzvané transition slová ako napríklad ale, takže, pretože, kvôli. Pridanie týchto slov často zmení typ vety.
  Pôvodná veta:
  Technológia môže spôsobiť katastrofu.
  Pozmenená veta:
  Katastrofa je spôsobená kvôli technológiám.
- Zmenenie slovosledu vo vete
  V originálnej vete sa pomení poradie slov, tak aby nová veta mala rovnaký význam.
  Pôvodná veta:
  Technológie môžu zlepšiť kvalitu života ak si dôkladne plánujeme budúcnosť.
  Pozmenená veta:
  Ak si dôkladne plánujeme budúcnosť, technológie nám môžu zlepšiť kvalitu života [Clough, 2000].

1.3 Prehľad prístupov k detekcii plagiarizmu

1.3.1 Prehľad metód spracovania zdrojových kódov

Pre všetky metódy platí, že lepšie výsledky vykazujú po predspracovaní kódu, za čo môžeme považovať jeho tokenizáciu. Tokenizácia môže byť implementovaná rôznymi spôsobmi a výsledok predspracovania kódu má významný vplyv na určenie podobnosti.
Metódy spracovania zdrojového kódu môžeme rozdeliť podľa prístupu k zdrojovému kódu na:

- Štrukturálne – metódy hľadajúce v kóde rovnaké štrukturálne prvky
- Neštrukturálne – metódy hľadajúce jednoducho podobnosti textu zdrojových kódov

Medzi štrukturálne metódy môžeme zaradiť napríklad hľadanie najväčšieho spoločného podgrafu.
Neštrukturálne metódy ďalej môžeme rozdeliť na:

- Štandardné
- Metódy pre porovnávanie textov

Do štandardných metód na porovnávanie zdrojových kódov môžeme zaradiť Rabinov-Karpov algoritmus, prípadne Greedy String Tiling.
Na zisťovanie plagiarizmu v zdrojových kódoch sa však po predspracovaní často používajú aj metódy na spracovanie textov. Ich prehľad sa nachádza v nasledujúcej kapitole.

1.3.2 Prehľad metód spracovania slovenských textov

Zdeněk Češka uvádza rozdelenie metód pre detekciu plagiarizmu v textových dokumentoch na základe dvoch hlavných kritérií [Češka, 2008]. Podľa zložitosti metódy rozdeľujeme nasledovne:

- *Povrchné* – Metódy nevyužívajú žiadne lingvistické pravidlá, pracujú iba s pôvodným textom.
- *Štrukturálne* – Metódy sa snažia čiastočne porozumieť dokumentu.

Druhé kritérium predstavuje počet spracovaných dokumentov. Na základe spomínaného kritéria môžeme metódy zaradiť do nasledujúcich štyroch kategórií:

- *Jednotlivé* – Metódy spracovávajú v rovnakom čase iba jeden dokument.
- *Párové* – Metódy vykonávajú spracovanie dvoch dokumentov naraz.
- *Multidimenzionálne* – Metódy spracovávajú aspoň tri dokumenty naraz.
- *Korpálne* – Metódy spracovávajú naraz celú skúmanú vzorku.

Pozrime sa detailnejšie na rozdelenie podľa prvého kritéria. Povrchné metódy spravidla pracujú priamo s dokumentom, ktorý skúmame. Štrukturálne metódy naopak pred samotným spracovaním textu vykonávajú predspracovanie. V slovenskom jazyku toto predspracovanie pozostáva z dvoch činností:

1. *Odstránenie stop-slov* – Existujú slová, ktoré tvoria veľkú časť dokumentov v prirodzenom jazyku, ale principiálne neovplyvňujú význam textu. Spomínané slová nazývame stop-slovami a môžeme ich jednoducho vylúčiť z textu.
2. *Lematizácia* – Pod pojmom lematizácie rozumieme prevod slova na základný tvar

Druhé kritérium rozdeľuje metódy na základe počtu dokumentov, ktoré spracovávame naraz. Vzhľadom k tomu, že si kladieme za cieľ skúmať podobnosť, prirodzenými metódami sú predovšetkým metódy párové, z ktorých si predstavíme napr. metódu 3-gramy alebo LCS. Vydáme sa ale aj ku korpálnej metóde latentnej sémantickej analýzy.

2 Metódy a prístupy k detekcii plagiarizmu

2.1 Predspracovanie zdrojových kódov a dokumentácie

2.1.1 Stop-slová a lematizácia v slovenských textoch

Predspracovanie slovenských textov pre potreby bližšej analýzy textu prebieha v dvoch fázach.
Prvú fázu tvorí odstránenie tzv. stop slov, sémanticky nevýznamných slov. Sú to zväčša krátke slová, predložky, spojky, častice prípadne samostatné číslice.

V druhej časti, nazývanej lematizácia, sa každé slovo zmení na jeho slovníkový tvar prípadne na koreň pôvodného slova. Túto transformáciu možno dosiahnuť niekoľkými spôsobmi.
Veľmi jednoduchá ale účinná metóda je použitie databázy všetkých slov v slovníku a ich príslušných možných tvarov. Následne možno spárovať ľubovoľné slovo obsiahnuté v slovníku s jeho základným tvarom. Problém nastane v prípade slov, ktoré nie sú obsiahnuté v slovníku. Sú to prevažne nové slová, mená, priezviská, názvy firiem. Koreň slova z takýchto slov je možné získať iba algoritmickou detekciou, ktorá však nemusí byť vždy presná. Ďalším problémom môže byť nutnosť rýchleho prístupu do databázy a vyhľadávanie v nej. Dôležitý je práve fakt, že samotný slovník slovenského jazyka obsahuje približne 150000 slov v základnom tvare, ktoré majú ďalších približne 5 miliónov tvarov, čo nie je málo. Možným úskalím sa javí aj samotná dostupnosť a prítomnosť spomínanej databázy v systémoch, kde je lematizácia potrebná. Aj zo spomenutých dôvodov je snaha vyvinúť algoritmické riešenie, ktoré nie je založené len na úplnosti a dostupnosti databázy všetkých slov.
Potrebné riešenie existuje v anglickom jazyku implementované ako Porter-ov algoritmus. Tento prístup je založený na odstraňovaní známych prípon zo slov. V slovenčine je však tento spôsob takmer nemožný prípadne veľmi neefektívny, pretože by bolo treba odvodiť všetky pravidlá na ohýbanie slov.
V dokumente [LM] je však navrhnutý zaujímavý spôsob riešenia uvedeného problému pomocou "cross-over" algoritmu. Tento algoritmus predpokladá, že máme k dispozícii databázu všetkých slovenských slov v ich základnom tvare a ďalej reprezentatívnu vzorku slov a ich tvarov po skloňovaní rozložených na slabiky.
Algoritmus funguje na základe podobného skloňovania slov. K zadanému slovu sa vždy vyberie slovo z databázy s rovnakou koncovkou. Ďalej sa zistí základný tvar vybraného slova, pričom sa sleduje zmena koncovky slova pri tomto prechode a následne sa algoritmus snaží vykonať podobnú transformáciu aj na zadanom slove. Výsledné slovo sa nakoniec skontroluje v databáze všetkých slov v základnom tvare. Ak sa také slovo našlo, predpokladáme, transformácia prebehla úspešne. Kvalita výsledku samozrejme závisí od správneho výberu slova s rovnakým skloňovaním, čo však možno zefektívniť ak dané slovo bude mať napríklad ten istý rod. Samozrejme výsledok nie je vždy korektný, známe sú napríklad problémy pri jednoslabičných slovách, pri zmene základu slova pri skloňovaní a iné. Dôležité však je, že testovaním [LM] sa ukázalo, že správnosť výsledku pri použití spomenutého algoritmu je vyše 90%. Táto hodnota je veľmi dobrá aj vzhľadom nato, že nepotrebujeme databázu všetkých slov a ich skloňovaní.

Príklad predspracovania textu:

1 vyradenie stop slov z textu
   Mama mi povedala, aby som kúpil 10 rožkov a priniesol aspoň jednu limonádu.
   Mama povedala kúpil 10 rožkov priniesol aspoň jednu limonádu.
2. úprava slov na základný tvar
   Mama povedať kúpiť 10 rožok priniesť aspoň jedna limonáda.

2.1.2 Tokenizácia zdrojového kódu

Tokenizácia je proces, pri ktorom sa snažíme inštrukcie pôvodného kódu nahradiť inými symbolmi tak, aby sme ich význam zovšeobecnili. To znamená, že viacerým pôvodným symbolom môžeme priradiť jeden a ten istý symbol (token) . Týmto spôsobom zabezpečíme, že ak plagiátor zmení určitú časť kódu za inú funkčne ekvivalentnú časť, tak po predspracovaní oboch kódov tokenizáciou získame identické výsledky. Pri tokenizáciu časti kódu "while x>3;" môžeme podľa stupňa generalizácie získať napríklad takéto výsledky :

- while VARIABLE > 3
- while
- CYCLE_begin

Je teda zrejmé, že samotný úspech tokenizácie závisí aj od zvoleného stupňa generalizácie.
Nespornou výhodou tokenizácie je skutočnosť, že takýmto spôsobom môžeme úspešne porovnávať zdrojové kódy, ktoré sú naprogramované inými programovacími jazykmi. V každom jazyku existuje istá forma podmienok, cyklov, metód a podobne. Tieto prvky sa pri tokenizácii prevedú na jeden, pre všetky jazyky rovnaký symbol.
Pre lepšie pochopenie možných tokenov je uvedená nasledujúca tabuľka, kde sa jednotlivým identifikátorom v zdrojovom kóde priradí symbol, token.

Identifikátor v zdrojovom kóde	Token
Int, float, double, boolean, byte, long, short	NUMBER
String, StringBuffer, StringBuilder, char	STRING
while, do-while, for, repeat-until, if then, else	BLOCK
<, >, <=,>=, ==, *!=*	COMPARATOR
ľubovoľná premenná	VARIABLE
+, -, *, /,	OPERATOR

2.2 Porovnávanie zdrojových kódov

2.2.1 Rabinov-Karpov algoritmus

Rabinov–Karpov algoritmus patrí medzi klasické prístupy k riešeniu problému vyhľadávania podreťazcov. V prvom kroku vypočítame hash reťazca i podreťazca, tým sa konvertuje úloha porovnávania reťazcov na úlohu porovnávania hashov s kontrolou zhody reťazcov. Následne iterujeme nad reťazcom a počítame hashe pre všetky podreťazce typu 1..m, 2..m+1, atď. Porovnávame hashe. V prípade zhody porovnávame samotné reťazce. Algoritmus môžeme vyjadriť v nasledujúcom pseudokóde.

Algoritmus 2.2.1 – Rabinov-Karpov algoritmus [Karp, 1987]

```
RabinKarp(string s[1..n], string sub[1..m])
  hsub = hash(sub[1..m]); hs = hash(s[1..m])
  for i=1..(n-m+1)
    if (hs == hsub)
      if (s[i..i+m-1] == sub)
        return i
      hs = hash(s[i+1..i+m])
  return not found
```

2.2.2 Metóda Greedy String Tiling

Greedy String Tiling predstavuje metódu na porovnávanie tokenizovaných zdrojových kódov, v ktorej implementácii sa spravidla využíva Rabinov-Karpov algoritmus. Cyklicky iterujeme nad tokenmi, využívame tzv. zaznačovanie tokenov, na začiatku každej iterácie sú všetky tokeny v reťazcoch A a B nezaznačené. Pre všetky neoznačené tokeny v A a B potom hľadáme čo najdlhšiu sekvenciu zhodných tokenov. Vytvoríme množinu takýchto sekvenciu, ktorej tokeny následne zaznačíme. Pokračujeme ďalšou iteráciou. Algoritmus špecifikujeme nasledujúcim pseudokódom.

Algoritmus 2.2.2 – Greedy String Tiling [Wise, 1993]

```
GreedyStringTiling(String A, String B)
  tiles = {};
  do
    maxmatch = Max;
    matches = {};
    Forall unmarked tokens Aa in A
      Forall unmarked tokens Bb in B
        j = 0;
        while (Aa+j == Bb+j && unmarked(Aa+j) && unmarked(Bb+j))
          j++;
          if (j == maxmatch)
            matches = matches + match(a, b, j);
          else if (j > maxmatch)
            matches = {match(a, b, j)};
            maxmatch = j;
          Forall match(a, b, maxmatch) in matches
            For j = 0..(maxmatch- 1)
              mark(Aa+j);
              mark(Bb+j);
            tiles = tiles + match(a, b, maxmatch);
  while (maxmatch > M);
  return tiles;
```

2.3 Porovnávanie slovenských textov

2.3.1 Metóda N-gramov

Jednu z najjednoduchších a paradoxne aj najúčinnejších metód porovnávania fráz v textoch predstavuje metóda N-gramov. Pod N-gramom budeme rozumieť N-ticu slov, ktoré v texte nasledujú bezprostredne za sebou.

Uvažujme napr. metódu 3-gramov. Pri porovnávaní dvoch súborov vtedy vyberieme 1., 2. a 3. slovo z obidvoch súborov a porovnáme. Celý proces opakujeme nad všetkými možnými dvojicami 3-gramov, ktoré v daných súboroch existujú (vrátane prekrývajúcich sa 3-gramov).

Porovnanie dvoch N-gramov možno najjednoduchšie vykonať prostredníctvom naivnej metódy, t.j. porovnávať frázy postupne po znakoch.

2.3.2 Najdlhší spoločný podreťazec

Problém najdlhšieho spoločného podreťazca dvoch reťazcov (angl. Longest Common Substring, ďalej len LCS) patrí medzi typické aplikácie dynamického programovania. Algoritmus 2.3.1 popisuje, ako je

využívaná tabuľka – dvojrozmerné pole `back` na memorizáciu výsledkov pre i-tu pozíciu prvého a j-tu pozíciu druhého reťazca znakov. Rekurzívna formulácia algoritmu je nasledujúca [LCS]:

$$c[i,j] = \begin{cases} 0 & ak\ i = 0 \vee j = 0 \\ c[i-1, j-1] + 1 & ak\ i,j > 0 \wedge x_i = y_j \\ \max\{c[i.j-1], c[i-1,j]\} & inak \end{cases} \quad (1)$$

Algoritmus 2.3.1 – Hľadanie LCS pomocou dynamického programovania „zdola nahor“ [LCS]

```
LCS(X,Y)
        m <- LENGTH[X]; n <- LENGTH[Y];
        for i <- 1 to m, do c[i,0] <- 0;
        for j <- 0 to n, do c[0,j] <- 0;
        back <- c;

        for i <- 1 to m, do
                for j <- 1 to n do
                        if (x[i] = y[j])
                                c[i,j] <- c[i-1, j-1]+1;
                                back[i,j] <- "UP&LEFT";
                        else if (c[i-1,j] >= c[i,j-1])
                                c[i,j] <- c[i-1,j];
                                back[i,j] <- "UP";
                        else
                                c[i,j] <- c[i,j-1];
                                back[i,j] <- "LEFT";
        return c and back
```

Metóda najdlhšieho spoločného podreťazca vykazuje pre detekciu plagiarizmu relatívne nízke percentuálne ohodnotenia podobnosti, preto je potrebné dbať na to, aby bola nastavená dostatočne nízka hranica, po ktorej prekročení systém vyhlási súbory za podobné. Spomínaná hranica sa môže pohybovať približne okolo 3%.

2.3.3 Frekvencie slov v dokumentoch

Ďalší prístup k detekcii podobnosti v slovenských textoch predstavuje počítanie frekvencií, s ktorými sa jednotlivé slová nachádzajú v skúmaných textoch. Jednoduchú metódu, založenú na porovnávaní týchto početností, nazývame metódou TF.

Modifikáciu metódy TF môžeme získať tak, že vylúčime slová, ktoré sa vyskytujú v danej vzorke veľmi často, uvažujeme následne iba frekvencie výskytov vzácnejších slov, čo pri rovnakej téme článkov vedie na rozdiel od LCS práve k vysokým podobnostiam. Jedná sa o metódu inverznej frekvencie slov v dokumentoch (angl. Inverse Document Frequency, skr. IDF).

2.3.4 Latentná sémantická analýza

V nasledujúcich riadkoch sa oboznámime s metódou, ktorá už využíva zložitejší prístup k detekcii plagiarizmu v dokumentoch. Latentná sématická analýza sa, ako alternatívna metóda, zakladá na reprezentácii vzorky dokumentov pomocou lineárneho modelu. Koncepciu metódy a jej využiteľnosť predstavil Zdeněk Češka [Češka, 2008].

Fáza predspracovania v metóde LSA

Predspracovanie prirodzeného jazyka prebieha v LSA rovnako, ako sme opísali v časti 2.1.1, zmažú sa stop-slová, vykoná sa lematizácia. Extrahujeme z dokumentov frázy ako N-gramy, pričom odporúčaná hodnota N sa pohybuje približne v intervale od 5 do 10. Následne sa zo vzorky odstránia frázy, ktoré sa nachádzajú iba v jednom dokumente. Naopak, časté frázy odstraňujeme vtedy, keď ich zastúpenie v dokumentoch presahuje určitú hranicu. Ostatné frázy sa použijú na tvorbu matice A.

Model dokumentov a jeho vyjadrenie pomocou matice

Budeme mať maticu A rozmeru n x m. Jej riadky popisujú model, ktorý určuje, ktoré frázy dokument reprezentovaný i-tym stĺpcom matice obsahuje. V poradí j-ty riadok matice reprezentuje frázu j. Potom ohodnotíme prvky matice hodnotami podľa nasledujúceho vzťahu [Češka, 2008]:

$$a[i,j] = \begin{cases} \frac{1}{2} + \frac{PF[i,j]\log\left(\frac{|n|}{DF[j]}\right)}{2\max_i\{PF[i,j]\}log(|n|)} & ak\ fráza\ j\ je\ v\ dokumente\ i \\ 0 & inak \end{cases} \quad (2)$$

kde PF[i,j] znamená frekvenciu výskytu frázy j v dokumente i, DF[j] znamená počet dokumentov, v ktorých sa nachádza fráza j a |n| je počet skúmaných dokumentov. Zložitosť vzorca pramení z požiadavky, aby sa koeficienty matice pohybovali v intervale <0.5;1>, čím sa dosiahne výraznejší "odstup" od nuly, ktorá reprezentuje absenciu frázy v danom dokumente.

Algoritmus singulárnej dekompozície matíc

Definujme vlastné číslo štvorcovej matice M ako taký skalár λ, že platí:

M**x** = λ**x**, (3)

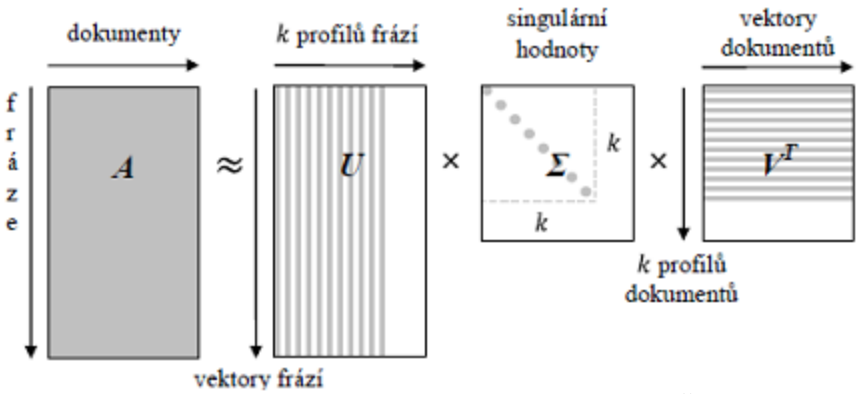
pre nejaký vektor **x**, ktorý nazývame vlastným.

Nech A je teraz obdĺžniková matica rozmeru n x m (napr. naša matica modelu dokumentov). Uvažujme matice AA^T a A^TA, kde operátor ^{T} značí transpozíciu matice. Uvedené matice sú obidve evidentne štvorcové (rozmerov m x m, resp. n x n). Pod singulárnou dekompozíciou matice A rozumieme súčin:

$A = U \Sigma V^T$. (4)

Postulujme, že matice U a V majú k spoločných vlastných čísel. Matica Σ rozmeru k x k obsahuje spomínané vlastné čísla AA^T a A^TA, nazývame ich singulárnymi číslami matice A. Matice U rozmeru n x k a V rozmeru m x k potom v stĺpcoch obsahujú vlastné vektory, prislúchajúce k maticiam AA^T, resp. A^TA. Spomínané vektory nazývame takisto singulárnymi vektormi matice A.

Interpretácia singulárnej dekompozície vo vzťahu k LSA je zobrazená na nasledujúcom obrázku. Matice U, resp. V obsahujú vo svojich stĺpcoch vektory - profily jednotlivých fráz, resp. dokumentov.



Obr. 4: Singulárna dekompozícia v kontexte LSA [Češka, 2008]

Metódu singulárnej dekompozície možno implementovať viacerými spôsobmi. Niekoľko z nich priamo ponúka vedecká knižnica pre jazyk C – GNU Scientific Library [GSL].

Výpočet korelačnej matice podobnosti dokumentov

Profily, ktoré získame singulárnou dekompozíciou matice dokumentov nám ešte o hľadanej podobnosti priamo nič nepovedia. Preto je potrebné pristúpiť k záverečnej fáze LSA. V prvom kroku vynásobíme jednotlivé profily dokumentov zodpovedajúcimi singulárnymi číslami, čo môžeme zapísať pomocou maticového súčinu:

$B = \Sigma V^T$. (5)

Stĺpce matice B teraz musíme normalizovať, t.j. každý stĺpcový vektor musíme vynásobiť skalárom tak, aby jeho veľkosť bola rovná 1. Vznikne nám tak matica ||B||. Položme:

$S^* = ||B||^T ||B||$. (6)

Uvedeným výpočtom už získame maticu, ktorej prvky už popisujú podobnosti medzi jednotlivými dokumentmi vo vzorke. Vzhľadom k využitiu filtra vo fáze predspracovania sú hodnoty ale skreslené, preto musíme vykonať nasledujúci výpočet, ktorým získame finálnu korelačnú maticu podobnosti dokumentov S:

$$S(X,Y) = S^*(X,Y)\sqrt{\frac{|ph_{orig}(X)||ph_{orig}(Y)|}{|ph_{red}(X)||ph_{red}(Y)|}} \quad (7)$$

V uvedenej rovnici ph_{orig} predstavuje počet pôvodných fráz a ph_{orig} počet fráz, ktoré v dokumente zostávajú po redukcii.

2.4 Alternatívne prístupy k detekcii plagiarizmu

Kontrola zdrojových kódov – inštrukcie

Tokenizácia zdrojového kódu by mohla prebiehať na úrovni jednotlivých príkazov – každý príkaz by bol jeden token. Toto by mohlo byť realizované prevodom zdrojového kódu do jazyku symbolických inštrukcií, prípadne do postupnosti pseudoinštrukcií, sada ktorých by bola navrhnutá špeciálne pre tento účel. Pri tomto prevode by boli cykly a podmienky zapísané pomocou skokov, podobne, ako je to pri kompilovaní programu do strojového kódu. Vďaka tomu by sa zvýšila schopnosť programu odhaľovať techniky používané plagiátormi, akými sú napríklad prepísanie while cyklov na for cykly a podobne. Ďalšou možnosťou je nahradenie volaní lokálnych funkcií ich implementáciou, v prípade,

že je to možné. Na takto predspracovanom zdrojovom kóde by sa mohlo vykonať porovnanie pomocou ľubovoľnej metódy na porovnávanie textov – jednotlivé pseudoinštrukcie by boli spracovávané rovnako ako slová v základnom tvare. Bol by však kladený dôraz predovšetkým na rovnaké postupnosti inštrukcií, pretože rovnaké skupiny inštrukcií s rôznym poradím pravdepodobne neznamenajú rovnaké zdrojové kódy. Samotné inštrukcie by neobsahovali dáta, ktoré majú pri kontrole plagiarizmu veľkú výpovednú hodnotu (až na niekoľko špeciálnych prípadov).

Kontrola zdrojových kódov – graf

Ďalšou metódou kontroly by mohlo byť vytvorenie grafovej reprezentácie kontrolovaného zdrojového kódu. Vrcholy grafu by tvorili dátové štruktúry a bloky kódu, hrany by predstavovali vzťahy medzi týmito entitami. V takomto grafe by sa hľadal najväčší spoločný podgraf, pričom by sa brali do úvahy taktiež typy jednotlivých vrcholov. Typmi sa myslí napríklad funkcia, štruktúra, if-podmienka, for-cyklus, a podobne.

String-blurring algoritmus

V našom systéme chceme experimentovať s naším vlastným algoritmom, ktorý sme nazvali String-blurring. Jeho podstatou je vynechanie všetkých bielych znakov z textu (medzery, tabulátory, konce riadkov, a pod.) a uloženie jednotlivých znakov do jednorozmerného poľa. Každý znak je v tomto poli reprezentovaný svojou číselnou hodnotou. Pole je následne „rozmazané“ – hodnota každého prvku poľa je nahradená hodnotou váženého priemeru hodnôt okolitých prvkov, pričom váhy sú určené normálnym (gaussovským) rozdelením.
Z dvoch takto predspracovaných polí sa vyberú všetky možné prekrývajúce sa podpostupnosti, ktoré sa od seba odčítajú. Vo výsledkoch tvorených rozdielom hodnôt podpostupností sa následne vyhľadá najdlhšia podpostupnosť prvkov, ktorých absolútna hodnota je menšia ako zadaný prah. Dĺžka tejto podpostupnosti je výstupom algoritmu a tvorí index podobnosti daných súborov.
Okrem prahu podobnosti, na základe ktorého sa vyhľadávajú najdlhšie podpostupnosti, je možné meniť silu rozmazania – počet susedných prvkov, ktoré tvoria vážený priemer novej hodnoty prvku poľa.
Algoritmus je pomerne odolný voči malým a početným zmenám textu, ktoré by potenciálne dokázali oklamať algoritmus Longest common substring. Relevantnosť výsledkov však klesá so silou rozmazania, ktorá presiahne určitú hodnotu. Táto hodnota musí byť zistená experimentálne.

3 Analýza existujúcich riešení

Pre lepšie pochopenie vlastností, aké má budúca aplikácia poskytovať sa analyzovali niektoré v súčasnosti používané nástroje na detekciu plagiarizmu. Menovite SIDPlag, Jplag, SIM, MOSS, YAP, Plades a Sherlock. Niektoré sú primárne určené na zdrojové kódy, iné na porovnávanie textov. Všetky nástroje sme testovali na rovnakých vstupoch.

Slovenské texty sme získali na základe predchádzajúcich prác kolegov od Daniely Chudej. Pri zdrojových kódoch sa na chvíľu zastavíme. V kapitole 1.2 sú spomenuté základné princípy vytvárania plagiátov z originálnych zdrojových kódov. V nasledujúcej tabuľke je uvedená metodika pre tvorbu fiktívnych plagiátov za účelom testovania už existujúcich programov na detekciu plagiátorstva ako i vytváraného nástroja v tomto projekte. Týmto spôsobom sa zabezpečia objektívne a porovnateľné výsledky. V tabuľke sú zobrazené štyri úrovne plagiátorstva pričom každý nasledujúci zahŕňa aj zmeny predošlého plagiátu.

amatér	premenovanie premenných, prehodenie funkcií, metód, zmena typu premenných, zmena odsadenia, pridávanie prázdnych riadkov, medzier, zmena komentárov
študent	Prepis podmienok na while cykly a naopak, nahradenie volanie funkcií ich implementáciou, nahradenie konštánt, define
pokročilý	prepis jednoduchých operácii na vlastné funkcie, zmena poradia vzájomne nezávislých inštrukcií, blokov vo switch-i, poradie if {} else {}, čiastočné pridanie a odobratie zbytočného kódu a úprava podmienok pomocou zbytočného kódu
guru	pridanie zbytočného kódu za každú operáciu, drobná zmena funkcionality

3.1 Podobnosť zdrojových kódov

Táto kapitola je zameraná na nástroje primárne určené na podobnosť zdrojových kódov, to však neznamená, že sa nedajú použiť na detekciu plagiarizmu v slovenských textoch. Túto skutočnosť naznačujú aj niektoré dosiahnuté výsledky.

3.1.1 SIDPlag

Implementácia: Je programovaný v jazyku Java.

Algoritmus: Ideou algoritmu podľa dokumentácie je rozloženie si problému na dve časti. V prvej časti nástroj tokenizuje kód, tak aby zamenil všetky jednoducho zameniteľné prvky kódu na jeden všeobecný symbol. V ďalšej časti je použitý voľne dostupný algoritmus na porovnávanie textov Greedy String Tiling. Z uvedeného vyplýva, že zaujímavou časťou riešenia je práve časť tokenizácie. Program používa základné symboly pre tokenizáciu ako sú: zámena premenných, cyklov či zrušenie nepodstatných častí. Takýto druh tokenizácie by bol dostačujúci, ak by boli používateľovi poskytnuté možnosti pre pridávanie vlastných symbolov, čo je však len ťažko realizovateľné.

Podmienky použitia: Pre použitie SIDPlag-u musí používateľ mať nainštalovanú JRE v bližšie nešpecifikovanej verzii. Jedná sa voľne dostupnú aplikáciu, nie je nutná registrácia.

Použitie: Program SIDplag , slúži na porovnávanie zdrojových kódov jazyku java a C. Ďalej program umožňuje aj porovnávanie bežného textu, čo vyplýva z implementačného algoritmu programu, avšak o tejto funkcii nikde nie je zmienka.

Vizualizácia: SIDPlag zobrazuje podobnosti dvojíc súborov v prehľadnej tabuľke. Je možné vybrať konkrétnu dvojicu a zobraziť ich obsah vedľa seba. Tiež farebne vyznačuje rovnaké časti.

Klady:

- pri relatívne jednoduchom algoritme sa dajú získať akceptovateľné a vierohodné výsledky
- prehľadné zobrazenie výsledkov

Zápory:

- používateľovi nie sú poskytnuté možnosti pre pridávanie vlastných symbolov pre tokenizáciu
- program sa dá teda ľahko obísť potencionálnym plagiátorom, pridaním zbytočného kódu
- problém, ak plagiátor vkladá do kódu nič nevykonajúce, zbytočné časti implementácie
- program až príliš zaujatý voči plagiátorom, keď udával mieru podobnosti vyššiu ako 100%,
- táto situácia nastala vtedy, keď program chybne vyhodnotil podobnosť stringov (531%)
- nutnosť zadávania zdrojového adresára pri každom novom porovnávaní
- nekorektné určovanie veľkosti okna, pri príliš dlhej ceste k súborom je potrebné ručne okno zväčšovať, inak sú nedostupné tlačidlá

Výsledky testovania:
Výsledky sú dobré pri jednoduchých algoritmoch. Pri pridávaní zbytočných inštrukcií však strácajú na relevantnosti.

C

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	86.71%
program1 – program1 (plagiát študentský)	71.79%
program1 – program1 (plagiát pokročilý)	nedetekované
program1 – program1 (plagiát „guru“)	nedetekované
program2 – program2 (plagiát amatérsky)	chybové hlásenie
program2 – program2 (plagiát študentský)	chybové hlásenie
program2 – program2 (plagiát pokročilý)	chybové hlásenie
program2 – program2 (plagiát „guru“)	chybové hlásenie

Java

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	59%
program1 – program1 (plagiát študentský)	7.38%
program1 – program1 (plagiát pokročilý)	0.17%
program2 – program2 (plagiát amatérsky)	1.26%
program2 – program2 (plagiát študentský)	1.34%
program2 – program2 (plagiát pokročilý)	0.63%
program2 – program2 (plagiát „guru“)	0.63%

Celkový dojem: Čo sa týka myšlienky riešenia problému, nedá sa povedať, že je zlá ale ani najlepšia. Používateľské rozhranie je vporiadku, až na zmienený problém so zadávaním adresára. Rýchlosť systému je akceptovateľná vzhľadom na objem porovnávaní.

3.1.2 JPlag

Jplag sa špecializuje na odhaľovanie plagiátorstva v zdrojových kódoch. Podporuje jazyky Java, C#, C, C++ a Scheme, ale dá sa použiť aj pri hľadaní podobnosti v normálnom texte. Program slúži na odhaľovanie plagiátorstva medzi študentami, ale taktiež aj na nelegálnom kopírovaní softvéru.

Výrobca píše, že JPlag bol už viackrát použitý v majetkových súdnych prípadoch, v ktorých bol expertmi použitý ako dôkazový materiál a svedectvo.

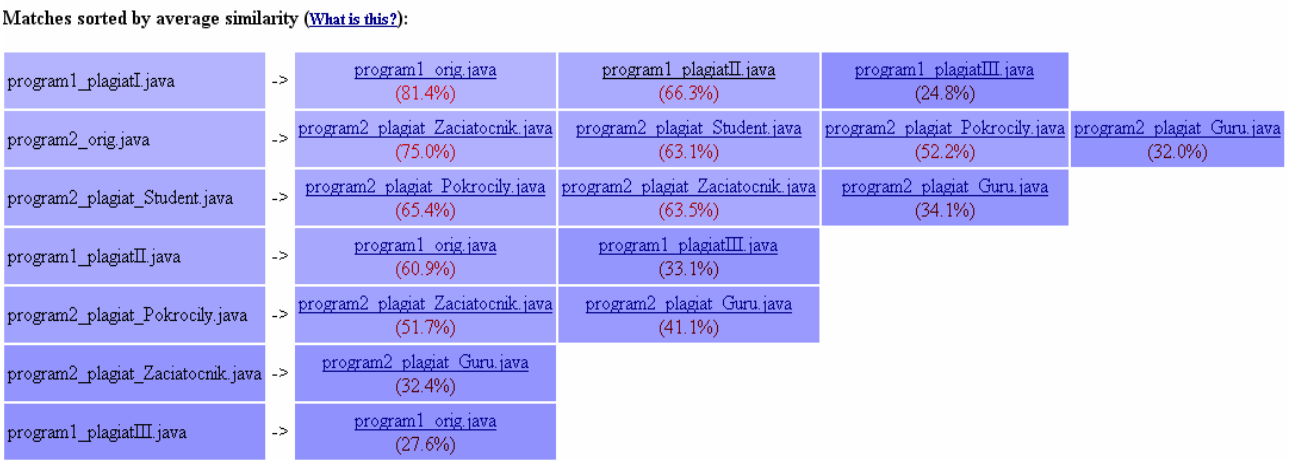
Implementácia: Program je naprogramovaný v jazyku Java, má klient-server architektúru.

Algoritmus: JPlag transformuje zdrojový kód programu do postupností tokenov. Každý token sa dá chápať ako jeden znak. Pre jazyk Java zverejnili tvorcovia tohto programu celkovo 39 rôznych tokenov. Po transformácii na tokeny ich následne porovnáva a hľadá zhody pomocou algoritmu GST. Teda hľadá podobné reťazce zložené z týchto tokenov.
GST porovnávanie prebieha po pároch. Pri každom porovnávaní sa snaží JPlag pokryť jeden prúd tokenov podreťazcom druhého čo najlepšie ako sa dá. Percento prúdu tokenov ktoré sa dá pokryť druhým je hodnota podobnosti.
Predpríprava, teda premena programu na reťazce tokenov je jediný proces JPlag-u, ktorý je závislý od programového jazyka. Implementácie pre jazyky Java a Scheme obsahujú úplný parser, zatiaľ čo modul pre jazyk C obsahuje iba skener.

Podmienky použitia: Pre použitie Jplagu musí byť používateľ zaregistrovaný. Registrácia je na základe formulára s overením identity na základe webového sídla školy, ktoré ukazuje email žiadateľa, študenti k Jplagu nemajú prístup pre prípadné zneužitie.

Použitie: Program má intuitívne grafické rozhranie a ponúka rôzne nastavenia. Funguje ako webová služba, preto je jednoducho dostupný pre užívateľa z ľubovolného miesta. Užívateľ si vyberie ktoré súbory sa majú kontrolovať a tie sa následne prenesú na server, kde sa okontrolujú. Testovanie je rýchle, rádovo niekoľko minút na 100 programov s niekoľko sto riadkovým kódom. Program hľadá podobnosti iba vo zvolených programoch, nekontroluje programy na Internete. Výsledok hľadania sa užívateľom prehľadne zobrazí vo forme html kódu. Užívateľ si taktiež môže nastaviť zložitosť prehľadávania, miesto kam sa majú výsledky uložiť, alebo napríklad aj akú podobnosť má brať JPlag ako podozrivú.

Vizualizácia: Jplag vizualizuje podobné časti kódu ich jednoduchým farebným zvýraznením. Na obrázku 5 sú znázornené výsledky nášho testovania. Na jednotlivé súbory je možné kliknúť, čím sa zobrazia dvojice zdrojových kódov súborov vedľa seba so zvýraznenou podobnosťou. Takto sa dá jednoducho zistiť ktoré časti súborov sú si vzájomne podobné. JPlag taktiež zobrazuje počet zhodujúcich sa tokenov v jednotlivých segmentoch, podľa čoho vyrátava celkové percento zhody.



Obr. 5.: Výsledky testovania v programe Jplag na vzorke programov v jazyku Java

Klady:

- intuitívne rozhranie
- škálovateľnosť nastavení

- klient-server architektúra
- veľmi dobé výsledky pre Java súbory
- prehľadná vizualizácia výsledkov
- archivácia predošlých testovaní
- pri zobrazení podľa priemernej podobnosti sa nevyskytli falošné plagiáty

Zápory:

- zložitejšie získavanie softvéru
- nutnosť byť učiteľom/zamestnancom v oblasti školstva
- slabšia úspešnosť pri súboroch v jazyku C
- podpora iba pre jazyky Java, C#, C, C++ a Scheme

Výsledky testovania:

JPlag dosahoval veľmi dobré výsledky pri odhaľovaní plagiátov v jazyku Java, no horšie výsledky v jazyku C. To môže byť spôsobené tým, že pre Javu je implementovaný spomínaný úplný parser, zatiaľ čo pre C iba skener.
Jednoduchšie plagiáty v jazyku Java odhalil na 50-81% podobnosť, ale aj pri prepracovanejších plagiátoch dosahovala podobnosť približne aspoň 30%, čo je dobrá úspešnosť. V jazyku C boli odhalené iba jednoduchšie plagiáty, ostatné ostali nedetekované.
Dobrým výsledkom taktiež je, že pri zobrazení výsledkov podľa priemernej podobnosti (average similarity) neboli nájdené falošné plagiáty, teda vyššie percento podobnosti pri nesúvisiacich programoch. Táto podobnosť pri algoritme maximálnej podobnosti (maximum similarity) dosahovala v niektorých prípadoch skoro až 50%, čo je veľmi nežiadúce. Na druhej strane je možné, že táto podobnosť vznikla z dôvodu použitia rovnakej metodiky pri tvorbe plagiátov, alebo podobného charakteru programov, keďže boli písané rovnakým autorom.

C

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	77%/68%
program1 – program1 (plagiát študentský)	26%/18%
program1 – program1 (plagiát pokročilý)	nedetekované
program1 – program1 (plagiát „guru“)	nedetekované
program2 – program2 (plagiát amatérsky)	54%/46%
program2 – program2 (plagiát študentský)	nedetekované
program2 – program2 (plagiát pokročilý)	nedetekované
program2 – program2 (plagiát „guru“)	nedetekované

Java

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	77%/86%
program1 – program1 (plagiát študentský)	64%/57%
program1 – program1 (plagiát pokročilý)	45%/25%
program1 – program1 (plagiát „guru“)	41%/20%
program2 – program2 (plagiát amatérsky)	75%/74%
program2 – program2 (plagiát študentský)	62%/63%
program2 – program2 (plagiát pokročilý)	49%/55%
program2 – program2 (plagiát „guru“)	35%/29%

Celkový dojem: veľmi užitočný nástroj, ktorý dosahuje prekvapivé výsledky pre zdrojové kódy v jazyku Java, no horšie pre jazyk C. Nevýhodami sú obmedzenia iba na vybrané jazyky a dostupnosť iba pre pedagógov, ktorým je určení. Oproti týmto nevýhodám ponúka program viaceré výhody, vďaka čomu je veľmi užitočný pri odhaľovaní plagiátov a šetrení času.

3.1.3 SIM

Implementácia: SIM je open-source textová aplikácia implementovaná v jazyku C, k dispozícii pre MS-DOS.

Algoritmus: SIM využíva tokenizáciu zdrojového kódu, pričom v tokenizovaných kódoch hľadá najdlhší spoločný podreťazec [Grune, 1989].

Podmienky použitia: Jedná sa o voľne stiahnuteľnú aplikáciu pod licenciou GNU GPL, nie je nutná registrácia.

Použitie: Textové rozhranie poskytuje porovnanie množiny zdrojových kódov, percentuálna podobnosť je určovaná relatívne – súbor 1 obsahuje X% obsahu súboru 2, opačne vychádza iná hodnota. Pre porovnávanie sú k dispozícii rôzne EXE súbory, pre každý jazyk jeden.

Vizualizácia: Systém zobrazí percentuálnu podobnosť súborov, vizualizáciu zhodných častí podporuje, ale absentujú v nej prvky zvýraznenia.

Klady:

- poskytuje relevantnú funkcionalitu
- výhoda príkazového riadku - GUI "nezavadzia"
- výsledok je čistý text, výsledky sa dajú relatívne ľahko využiť ďalej
- podpora relatívne slušnej škály jazykov a dokonca aj paradigiem - C, Pascal, Lisp, Java, …
- implementovaný v C, veľmi rýchly
- podporuje aj porovnávanie čistého textu

Zápory:

- chaoticky vyzerajúca prezentácia dát pre bežného používateľa
- slabá vizualizácia, súvisiaca s absenciou GUI
- pre každý jazyk má systém jeden EXE súbor, čo predstavuje otáznu koncepciu z hľadiska rozšíriteľnosti
- porovnávanie komentárov si vyžaduje spúšťať porovnanie súborov ako textu

Výsledky našich testov:

C

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	95%/91%
program1 – program1 (plagiát študentský)	nedetekované
program1 – program1 (plagiát pokročilý)	nedetekované
program1 – program1 (plagiát „guru“)	nedetekované
program2 – program2 (plagiát amatérsky)	61%/55%
program2 – program2 (plagiát študentský)	nedetekované
program2 – program2 (plagiát pokročilý)	nedetekované
program2 – program2 (plagiát „guru“)	nedetekované

Java

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	88%/76%
program1 – program1 (plagiát študentský)	5%/5%
program1 – program1 (plagiát pokročilý)	nedetekované
program1 – program1 (plagiát „guru“)	nedetekované
program2 – program2 (plagiát amatérsky)	80%/80%
program2 – program2 (plagiát študentský)	nedetekované
program2 – program2 (plagiát pokročilý)	nedetekované
program2 – program2 (plagiát „guru“)	nedetekované

Delphi

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	99%/98%
program1 – program1 (plagiát študentský)	nedetekované
program1 – program1 (plagiát pokročilý)	nedetekované
program1 – program1 (plagiát „guru“)	nedetekované

Celkový dojem: Solídny nástroj, ktorý spĺňa základné požiadavky, pričom spracovanie vzorky prebieha veľmi rýchlo. Jeho nedostatkom je predovšetkým nedostatočná schopnosť zachytiť zložitejšie plagiátorské prístupy a otázna je taktiež jeho využiteľnosť pre ďalšie jazyky (vhodnejšie by bolo využitie externých definícií štruktúr jazyka). Výhodou je podpora viacerých paradigiem programovania, čo ide ruka v ruke so širším dosahom aplikácie (netestujú sa len jazyky s inou syntaxou, ale aj inou koncepciou).

3.1.4 MOSS

Implementácia: MOSS je klient-server aplikacia, klient je implementovaný ako Perl skript.

Algoritmus: Keďže porovnávanie prebieha na serveri a autori algoritmus nezverejnili kvôli potencionálnemu zneužitiu študentmi, nevieme určiť ako samotné porovnávanie prebieha. Autori sa však odkazujú na článok, k dispozícii na [MOSS], kde sú spomenuté niektoré algoritmy na detekciu plagiarizmu.

Podmienky použitia: Pre použitie aplikácie je nutné sa registrovať, registrovať sa môže ktokoľvek. Po poslaní mailu na základe inštrukcií príde automatická odpoveď, v ktorej sa nachádza samotný klient.

Použitie: Pre používateľa OS typu Unix/Linux, ktorý má rád jednoduché textové aplikácie veľmi príjemné, keďže perl je štandardnou súčasťou týchto OS. Je potrebné len spustiť dodaný skript s parametrom pre súbory, ktoré sa majú otestovať.
Po uploadovaní súborov na server sa na serveri vykoná kontrola a následne klient dostane http link s prezentáciou výsledkov. Výsledky sú na serveri uložené ďalších 14 dní, potom sú automaticky zmazané. Zhody sú zobrazené prehľadne podľa súborov, usporiadané zostupne podľa počtu zhodných riadkov.
Dá sa použiť na detekciu plagiarizmu v širokej palete programovacích jazykov. Pre slovenské texty je však trochu obmedzený, berie do úvahy totiž iba ASCII kódovanie.

Vizualizácia: Okrem percentuálnej zhody sa dajú zobraziť súbory vedľa seba, kde je farbami jasne označené, ktoré časti sa zhodujú a ktoré sú odlišné (ak je prehodené poradie segmentov, farby zhodných segmentov sú rovnaké).

Klady:

- použitie je vcelku jednoduché a prehľadné, skript zvládne použiť naozaj každý, kto má k dispozícii stroj s vyššie spomenutým OS
- možnosť porovnávať celé adresáre ako programy, teda jeden program na jeden adresár (oceníme v prípade rozsiahlejších projektov)
- možnosť na vloženie tzv. *basefile* = obsahuje kód, o ktorom vyučujúci predpokladá, že ho budú zdrojové súbory obsahovať (riešenie špecifickej úlohy)
- pri použití vhodných parametrov je možné určiť počet prípadov, kedy výskyt zhodného segmentu kódu ešte považujeme za plagiarizmus a kedy za štandardnú súčasť všetkých kódov, čím sa dá vyhnúť použitiu basefile a súčasne zlepšiť výpovednú hodnotu výsledkov
- dobrá prezentácia výsledkov

Zápory:

- "nepodpora" iného ako ASCII kódovania, pre použitie na slovenské texty je to značne

obmedzujúce.

- nevedomosť o použitých algoritmoch na detekciu
- slabá detekcia prípadov, kedy bol kód zmenený trochu viac ako len premenovanie premenných, čo dokazujú aj výsledky testov
- pre bežného používateľa môže byť problém v použití klienta, keďže väčšina používateľov dokáže použiť len programy s GUI

Výsledky testovania: Pri programových kódoch vie odhaliť čisté kopírovanie, prípadne jednoduché zmeny, pri sofistikovanejšom plagiarizme však výsledky nie sú dobré. Napríklad pri doplňovaní podmienok o zbytočné klauzule, prípadne prepísaní typov cyklov tieto nevie detekovať. Toto je možné vidieť aj v prezentovaných testovacích výsledkoch. Pri použití na textové súbory odhalí aj tie, kde je prehodené poradie slov vo vete a niektoré vety sú navyše, ale ich počet nesmie prevyšovať 1-2 v danom segmente.

programy JAVA, C, Delphi:
časová náročnosť: 2.525s, z toho 2 sekundy upload súborov na server

slovenské texty:
časová náročnosť: 8.788s, z toho 6 sekúnd upload súborov na server

C

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	82%/78%
program1 – program1 (plagiát študentský)	nedetekované
program1 – program1 (plagiát pokročilý)	nedetekované
program1 – program1 (plagiát „guru“)	nedetekované
program2 – program2 (plagiát amatérsky)	57%/53%
program2 – program2 (plagiát študentský)	nedetekované
program2 – program2 (plagiát pokročilý)	nedetekované
program2 – program2 (plagiát „guru“)	nedetekované

Java

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	75%/87%
program1 – program1 (plagiát študentský)	57%/59%
program1 – program1 (plagiát pokročilý)	9%/5%
program2 – program2 (plagiát amatérsky)	80%/79%
program2 – program2 (plagiát študentský)	33%/31%
program2 – program2 (plagiát pokročilý)	14%/14%
program2 – program2 (plagiát „guru“)	4%/2%

Delphi

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	93%/92%
program1 – program1 (plagiát študentský)	56%/53%
program1 – program1 (plagiát pokročilý)	32%/28%
program1 – program1 (plagiát „guru“)	27%/22%

Celkový dojem: MOSS je veľmi dobrý nástroj pre testovanie plagiátov zdrojových kódov. Najlepšie sa uplatní v prípade jednoduchého skopírovania kódu, prípadne ľahkých modifikácií. Má výbornú podporu pre vysoký počet jazykov. Nevýhodou je nižšia rozlišovacia schopnosť sofistikovanejších plagiátov a taktiež pre Windows používateľa aj implementačné prostredie klienta.

3.1.5 YAP

YAP (Yet Another Plague) je systém pre detekciu plagiarizmu v programových kódoch (stránka autora naznačuje, že by do istej miery mal fungovať aj s anglickými textami). Dostupné sú verzie YAP1, YAP2 a YAP3.

Implementácia: Implementácia systému YAP je rozdelená do dvoch častí – tokenizácia a samotné vyhľadávanie plagiátov. Časť tokenizácie je spoločná pre všetky 3 verzie a je implementovaná ako shell skript.
YAP1 je implementovaný ako Bourne-shell skript využívajúci rôzne štandardné aplikácie, primárne sdiff (slúži na porovnávanie textových súborov). Zvyšné dve verzie sú vytvorené v jazyku Perl. Pre dosiahnutie vyššej rýchlosti porovnávania sú algoritmy verzií YAP2 a YAP3 implementované v jazyku C.

Algoritmus: YAP2 využíva Heckelov algoritmus, pri YAP3 je to Running Karp-Rabin, Greedy String Tiling algoritmus (Michael J. Wise: String Similarity via Greedy String Tiling and Running Kapr-Rabin Matching). Verzia YAP3 vďaka použitiu RKR-GST dokáže zdetegovať plagiáty aj so zmeneným poradím podreťazcov.

Podmienky použitia: YAP 1,2,3 je voľne stiahnuteľný zo stránky autora. Komerčné použitie je podmienené získaním súhlasu autora.

Použitie: Detekcia plagiarizmu prebieha v dvoch krokoch. Prvým je tokenizácia textov. Táto je relevantná iba pre programové kódy, pretože odstraňuje všetky symboly, ktoré nie sú súčasťou slovníka pre daný programovací jazyk. Tokenizácia sa dá opísať nasledovne [YAP]:

- komentáre a textové konštanty sú odstránené
- všetky písmená sú dekapitalizované
- rôzne synonymá sú nahradené ich bežnou formou
- ak je to možné, funkcie/procedúry sú rozvité v poradí volania
- všetky symboly, ktoré nie sú súčasťou slovníka pre daný programovací jazyk,sú odstránené

Tokenizácia má teda za cieľ previesť zdrojové kódy na čo najvšeobecnejšiu formu (v ideálnom prípade takú, že neautorský program bude zhodný s jeho originálom, aj napriek drobným zmenám v neautorskej verzii). Je zaujímavé všimnúť si bod o nahrádzaní funkcií ich implementáciou. Nahrádzanie je vykonané v poradí, v akom sa funkcie navzájom volajú, čo umožňuje pracovať aj volaniami funkcií z iných funkcií. Dôležitá je poznámka, že nahradenie prebehne, iba ak je to možné (príkladom, kde to možné nie je, sú rekurzívne funkcie – naivný prístup nahrádzania by viedol v zacykleniu programu, ktorý by sa snažil nahradiť volanie
funkcie implementáciou, ktorá by túto funkciu volala). Vďaka odstráneniu všetkých symbolov, ktoré nie sú súčasťou jazyka, systém efektívne oddeľuje dáta a samotný kód. Ignorovanie dát je logické. Dá sa totiž očakávať, že rovnaké zadanie bude viesť k riešeniam s obsahujúcim rovnaké dáta, aj keď nepôjde o plagiarizmus.

Fáza porovnávania vychádza zo súborov so symbolmi, ktoré boli vytvorené v predchádzajúcej fáze. Porovnávané sú potom dvojice týchto súborov. Systémy YAP1, YAP2 a YAP3 sa líšia iba v tejto fáze – tokenizácia je spoločná pre všetky tri verzie programu.

Vizualizácia: YAP je implementovaný ako konzolová aplikácia a jeho jediným výstupom je textový súbor s výsledkami.

Klady:

- jednoduchosť použitia
- tokenizácia
- nahradenie funkcií ich implementáciou

- použité algoritmy

Zápory:
- obmedzenie na Linux/Unix
- malá prenositeľnosť spôsobená (na dnešnú dobu) exotickou implementáciou
- pre bežného používateľa môže byť problém v použití klienta, keďže väčšina používateľov dokáže použiť len programy s GUI

Výsledky testovania: Systém YAP sa nám napriek vynaloženému úsiliu nepodarilo skompilovať, a preto sme ho nemohli otestovať.

Celkový dojem: Strohé, nekompromisné a rýchle riešenie hľadania plagiátov v programoch, jednoducho rozšíriteľné o schopnosť rozlišovať ďalšie programovacie jazyky vďaka oddelenej tokenizácii vstupných súborov.

3.2 Podobnosť slovenských textov

3.2.1 PlaDeS

Vznikol v rámci tímového projektu študentov Tomáša Hlatkého a Michala Kompana pod vedením Daniely Chudej.

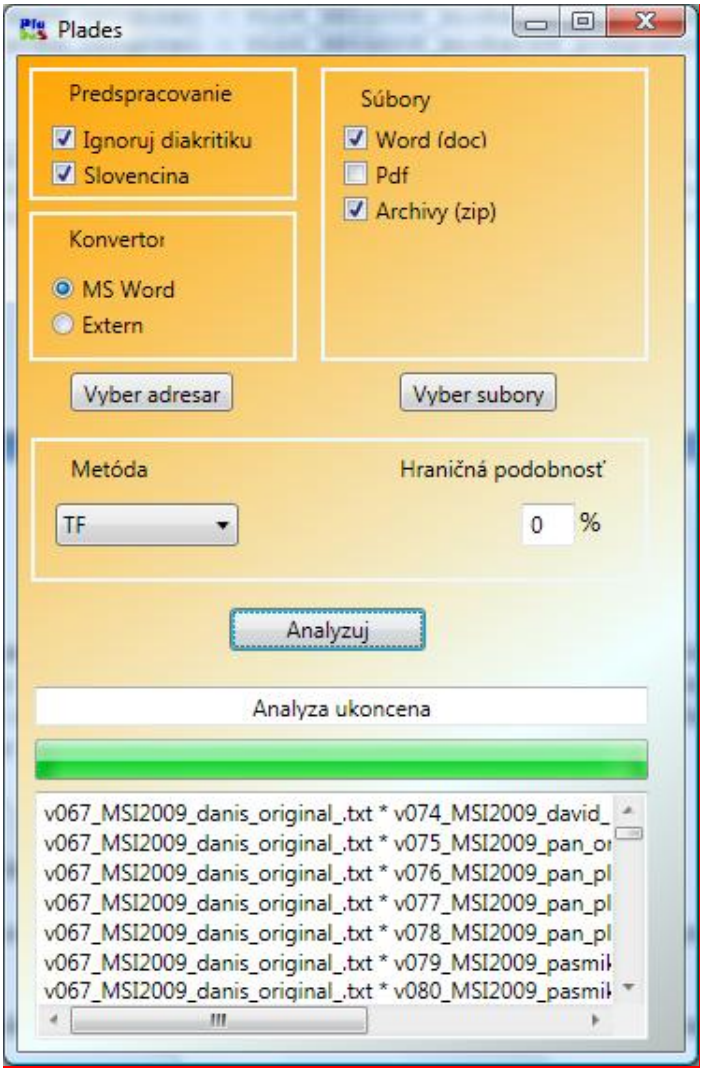
Implementácia: Jedná sa o aplikáciu určenú pre operačný systém Windows. Program je nutné inštalovať a pre beh je potrebný .NET 3.5 framework.

Algoritmus:
Aplikácia v sebe implementuje 4 druhy analýzy textu.
1. 3-gramy
2. najdlhšia spoločná podpostupnosť [LCS]
3. inverzná frekvencia slov v dokumente [IDF]
4. frekvencia výskytu slov [TF]

Podmienky použitia: Aplikácia je voľne prístupná na webe.

Použitie: Programu je možné nastaviť hraničnú podobnosť ako aj možnosť vypnúť diakritiku pri porovnávaní. Podľa slov autorov to nemá výrazný vplyv na výsledky porovnávania. Pri porovnávaní slovenských textov je možné z textov odstrániť stop slová a previesť lematizáciu, čo urýchľuje samotnú analýzu.
Program podporuje súbory vo formátoch pdf a doc, respektíve zip archívy. Je možné prehľadávať adresáre, v ktorých sa budú porovnávať všetky súbory, ako aj vyberať požadované súbory.
Program má jednoduché používateľské prostredie, ktoré podľa mňa bohato postačuje na prácu s programom. Ovládacie prvky sú logicky oddelené a nie je teda problém program ovládať. Program vizuálne informuje používateľa o vykonávanej činnosti čo osobne považujeme za veľmi kladný krok.
Pri spustení programu sa na obrazovke objaví splash screen, počas ktorého sa zrejme nahrávajú z disku konfiguračné súbory.
Pri skúške programu nastala chyba pri vyberaní súborov na analýzu. Program spadol ak nebol spustený s administrátorskými právami. Pri ďalšom spustení program fungoval už správne.

Vizualizácia:



Obr. 6.: Rozhranie programu Plades

Klady:

- Príjemné používateľské prostredie
- Možnosť nastavenia parametrov a metódy detekcie
- Možnosť vybrať na otestovanie konkrétne súbory alebo celý adresár
- Rýchlosť, spracovanie a analýza 60 doc súborov trvala 12 minút

Zápory:

- Nepodporuje obyčajné textové súbory

Výsledky testovania:

```
Metóda : 3gramy
Originál                        - plagiát                              Podobnosť
v122_MSI2008_michalek_original - v123_MSI2008_michalek_plagiatcopy      96%
v122_MSI2008_michalek_original - v124_MSI2008_michalek_plagiatmodify    75%
v122_MSI2008_michalek_original - v125_MSI2008_michalek_plagiatresers    77%

Metóda : LCS
Originál                        - plagiát                              Podobnosť
v122_MSI2008_michalek_original - v123_MSI2008_michalek_plagiatcopy      10%
```

```
v122_MSI2008_michalek_original - v124_MSI2008_michalek_plagiatmodify     8%
v122_MSI2008_michalek_original - v125_MSI2008_michalek_plagiatresers     9%

Metóda : IDF
Originál                       - plagiát                              Podobnosť
v122_MSI2008_michalek_original - v123_MSI2008_michalek_plagiatcopy     100%
v122_MSI2008_michalek_original - v124_MSI2008_michalek_plagiatmodify     0%
v122_MSI2008_michalek_original - v125_MSI2008_michalek_plagiatresers     0%

Metóda : TF
Originál                       - plagiát                              Podobnosť
v122_MSI2008_michalek_original - v123_MSI2008_michalek_plagiatcopy     100%
v122_MSI2008_michalek_original - v124_MSI2008_michalek_plagiatmodify    98%
v122_MSI2008_michalek_original - v125_MSI2008_michalek_plagiatresers    94%

Metóda : 3gramy
Originál                       - plagiát                              Podobnosť
v095_MSI2008_kozisek_original  - v096_MSI2008_kozisek_plagiatcopy       98%
v095_MSI2008_kozisek_original  - v097_MSI2008_kozisek_plagiatmodify     65%
v095_MSI2008_kozisek_original  - v098_MSI2008_kozisek_plagiatresers     83%

Metóda : LCS
Originál                       - plagiát                              Podobnosť
v095_MSI2008_kozisek_original  - v096_MSI2008_kozisek_plagiatcopy       11%
v095_MSI2008_kozisek_original  - v097_MSI2008_kozisek_plagiatmodify      8%
v095_MSI2008_kozisek_original  - v098_MSI2008_kozisek_plagiatresers     11%

Metóda : IDF
Originál                       - plagiát                              Podobnosť
v095_MSI2008_kozisek_original  - v096_MSI2008_kozisek_plagiatcopy        0%
v095_MSI2008_kozisek_original  - v097_MSI2008_kozisek_plagiatmodify      0%
v095_MSI2008_kozisek_original  - v098_MSI2008_kozisek_plagiatresers      0%

Metóda : TF
Originál                       - plagiát                              Podobnosť
v095_MSI2008_kozisek_original  - v096_MSI2008_kozisek_plagiatcopy      100%
v095_MSI2008_kozisek_original  - v097_MSI2008_kozisek_plagiatmodify     97%
v095_MSI2008_kozisek_original  - v098_MSI2008_kozisek_plagiatresers     99%
```

Celkový dojem: Program PlaDeS sa radí medzi jednoduchšie programy, no aj napriek tomu ponúka veľmi dobré výsledky pri hľadaní podobností v súboroch. Má v sebe implementované pokročilé algoritmy na hľadanie zhôd, pričom každý jeden z nich má výhody ako aj nevýhody. Program sa jednoducho obsluhuje a práca s ním je príjemná.

3.2.2 Sherlock

Sherlock je program, ktorý hľadá podobnosti v textových dokumentoch.

Implementácia: Open source, standalone, textová aplikácia, k dispozícii aj sherlock prepísaný do Javy [Sherlock].

Algoritmus: Na kontrolu využíva hashovaciu funkciu. Tá priradí bloku slov číslený odtlačok.

Podmienky použitia: Sherlock je voľne prístupný na stiahnutie

Použitie:

- ovládanie pomocou prepínačov

- hraničná podobnosť
- počet slov na vytvorenie signatúry
- skladá sa z jedného súboru
- výstup do konzoly alebo súboru
- je aj súčasťou BOSSu

Klady:

- veľmi rýchly, 69 textových súborov, každý ~ 20kB za zlomok sekundy

Zápory:

- nefunkčné podaktoré prepínače, ak je potreba otestovať 124 súborov, nutnosť všetky vypísať
- vypisuje len percentuálnu zhodu medzi súbormi, žiadne bližšie informácie

Výsledky testovania:

C

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	22%
program1 – program1 (plagiát študentský)	nedetekované
program1 – program1 (plagiát pokročilý)	nedetekované
program1 – program1 (plagiát „guru“)	nedetekované
program2 – program2 (plagiát amatérsky)	8%
program2 – program2 (plagiát študentský)	nedetekované
program2 – program2 (plagiát pokročilý)	nedetekované
program2 – program2 (plagiát „guru“)	nedetekované

Java

Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	55%
program1 – program1 (plagiát študentský)	8%
program1 – program1 (plagiát pokročilý)	5%
program2 – program2 (plagiát amatérsky)	8%
program2 – program2 (plagiát študentský)	4%
program2 – program2 (plagiát pokročilý)	4%
program2 – program2 (plagiát „guru“)	nedetekované

Delphi

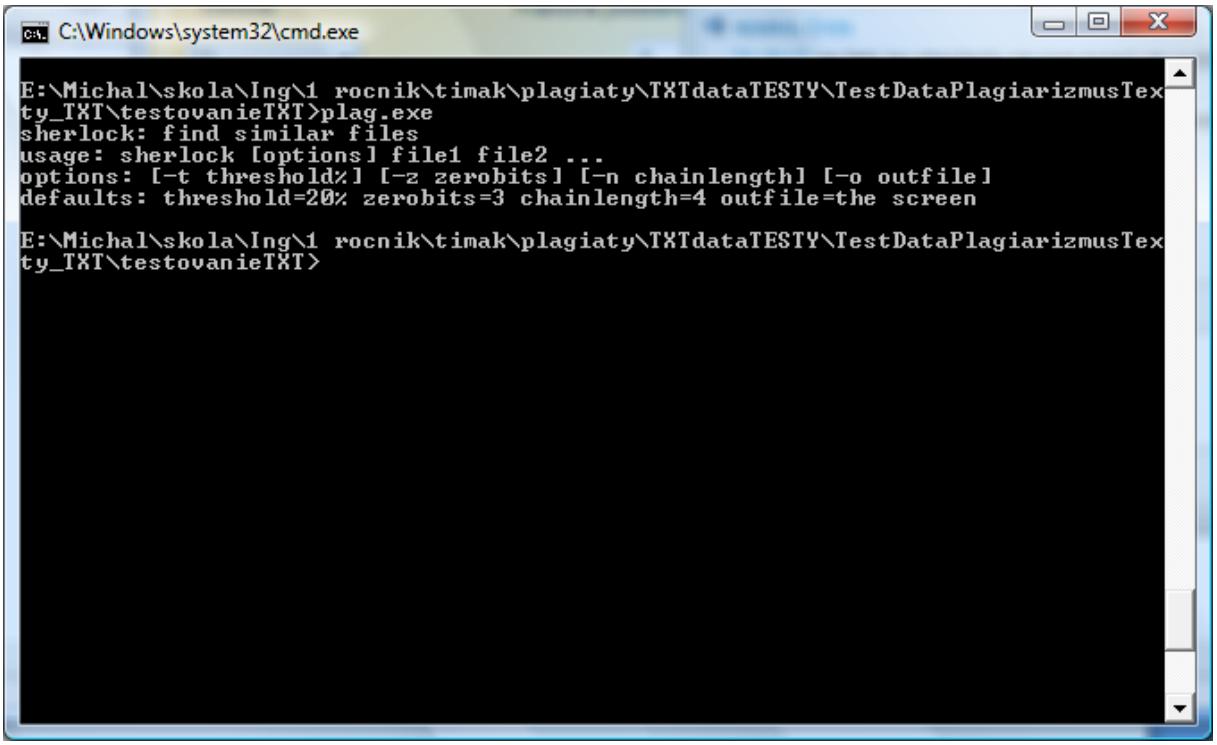
Originál – plagiát	Podobnosť
program1 – program1 (plagiát amatérsky)	16%
program1 – program1 (plagiát študentský)	12%
program1 – program1 (plagiát pokročilý)	8%
program1 – program1 (plagiát „guru“)	8%

Texty

Originál – plagiát	Podobnosť
v122_MSI2008_michalek_original – v123_MSI2008_michalek_plagiatcopy	100%
v122_MSI2008_michalek_original – v124_MSI2008_michalek_plagiatmodify	85%
v122_MSI2008_michalek_original – v125_MSI2008_michalek_plagiatresers	67%

Originál – plagiát	Podobnosť
v095_MSI2008_kozisek_original – v096_MSI2008_kozisek_plagiatcopy	99%
v095_MSI2008_kozisek_original – v097_MSI2008_kozisek_plagiatmodify	78%
v095_MSI2008_kozisek_original – v098_MSI2008_kozisek_plagiatresers	78%

Vizualizácia:



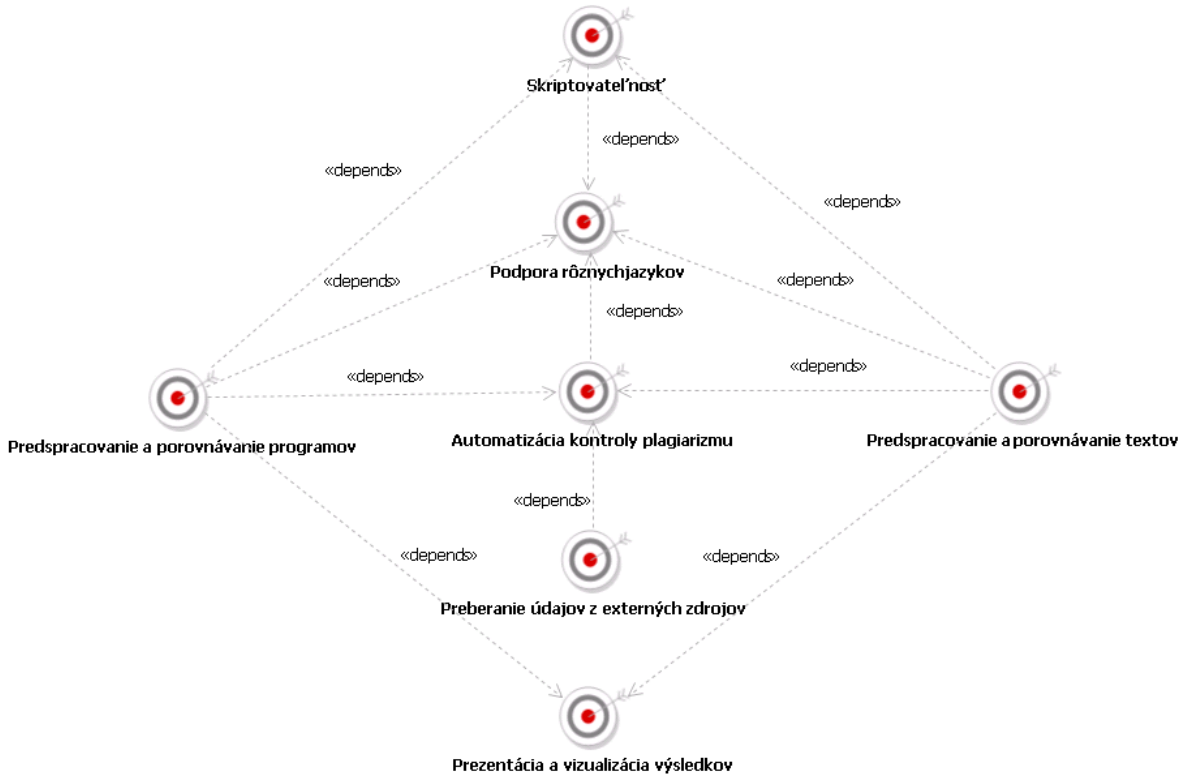
Obr. 7. Rozhranie programu Sherlock

Celkový dojem: Program Sherlock je jednoduchá konzolová aplikácia, ktorá napriek svojej jednoduchosti podáva veľmi dobré výsledky pri porovnávaní textových súborov. Pôvodná verzia sa distribuuje prostredníctvom zdrojového kódu, ktorý je potrebné skompilovať, čo môže odradiť niektorých používateľov. Nakoľko je program určený na porovnávanie textov, nie je vhodné ho používať na porovnávanie zdrojových kódov, nakoľko dáva nepresné výsledky.

4 Požiadavky na systém

4.1 Ciele projektu

Hlavné ciele nášho produktu sú zobrazené na obrázku 8. Tieto ciele spolu úzko súvisia, čo na obrázku predstavujú ich spojenia. Vo väčšine prípadov je úspešnosť jedného ciela závislá na úspešnosti iného, no v niektorých prípadoch ide o vzájomnú závislosť.



Obr. 8.: Diagram cieľov projektu

- **Predspracovanie a porovnávanie programov**
  Produkt bude porovnávať zdrojové kódy rôznych programov a hľadať podobnosti. Zdrojový kód pred samotným porovnávaním predspracuje podľa charakteristiky daného jazyka a jednotlivé inštrukcie zamení za tokeny. Samotné porovnávanie už môže teoreticky ďalej prebiehať nezávisle na type pôvodného jazyka kódu.
- **Predspracovanie a porovnávanie textov**
  Ďalším cieľom je vyhľadávanie plagiátov medzi textami v bežnom jazyku. Produkt sa bude zameriavať najmä na slovenský jazyk, ale neskôr bude možné rozšíriť funkcionalitu aj o porovnávanie textov v anglických a ďalších jazykoch. Pred samotným porovnávaním sa text taktiež predpripraví, čím sa odstránia prázdne znaky a stop slová. Tieto dva prvé ciele patria medzi najdôležitejšie a od nich sa odvíjajú ostatné.

- **Podpora rôznych jazykov**
  Tento cieľ nadväzuje na predchádzajúce ciele. Produkt bude vedieť pracovať so zdrojovými kódmi viacerých jazykov, aby sa dosiahla čo najvyššia využiteľnosť. Taktiež bude možné spomínané rozšírenie o podporu textov v cudzích jazykoch.
- **Skriptovateľnosť**
  Skriptovateľnosť priamo súvisí s podporou viacerých jazykov. Podľa skriptov sa bude ovládať prevažne predspracovanie kódu a textu. Skriptovaním sa zabezpečí jednoduché rozšírenie programu, prípadne opravenie existujúcich chýb v algoritmoch. Vďaka tomu nebude nutné pri pridávaní a čiastočnom menení funkcionality zasahovanie do kódu programu, a tak budú môcť užívatelia, prípadne iní študenti vylepšovať tento produkt.
- **Preberanie údajov z externých systémov**
  Údaje z Internetu, informačných systémov a sietí môžu výrazne zlepšiť úspešnosť odhaľovania plagiátov.
- **Prezentácia a vizualizácia výsledkov**
  Keďže výsledok testovania plagiátov nie je konečný, ale vo väčšine prípadov je nutné ručné kontrolovanie programov a textov používateľom produktu, je veľmi dôležité aby výsledky testovania boli zrozumiteľné pre používateľa. Dobrá vizualizácia výsledkov môže výrazne ušetriť jeho čas.
- **Automatizácia kontroly plagiarizmu**
  Tento cieľa úzko súvisí s ostatnými cieľmi a funkcionalitou produktu. Produkt bude automaticky zisťovať jazyk kódu, alebo textu, predpripráví ho, získa údaje z externých zdrojov a zobrazí výsledky, ktoré bude aj vizualizovať.

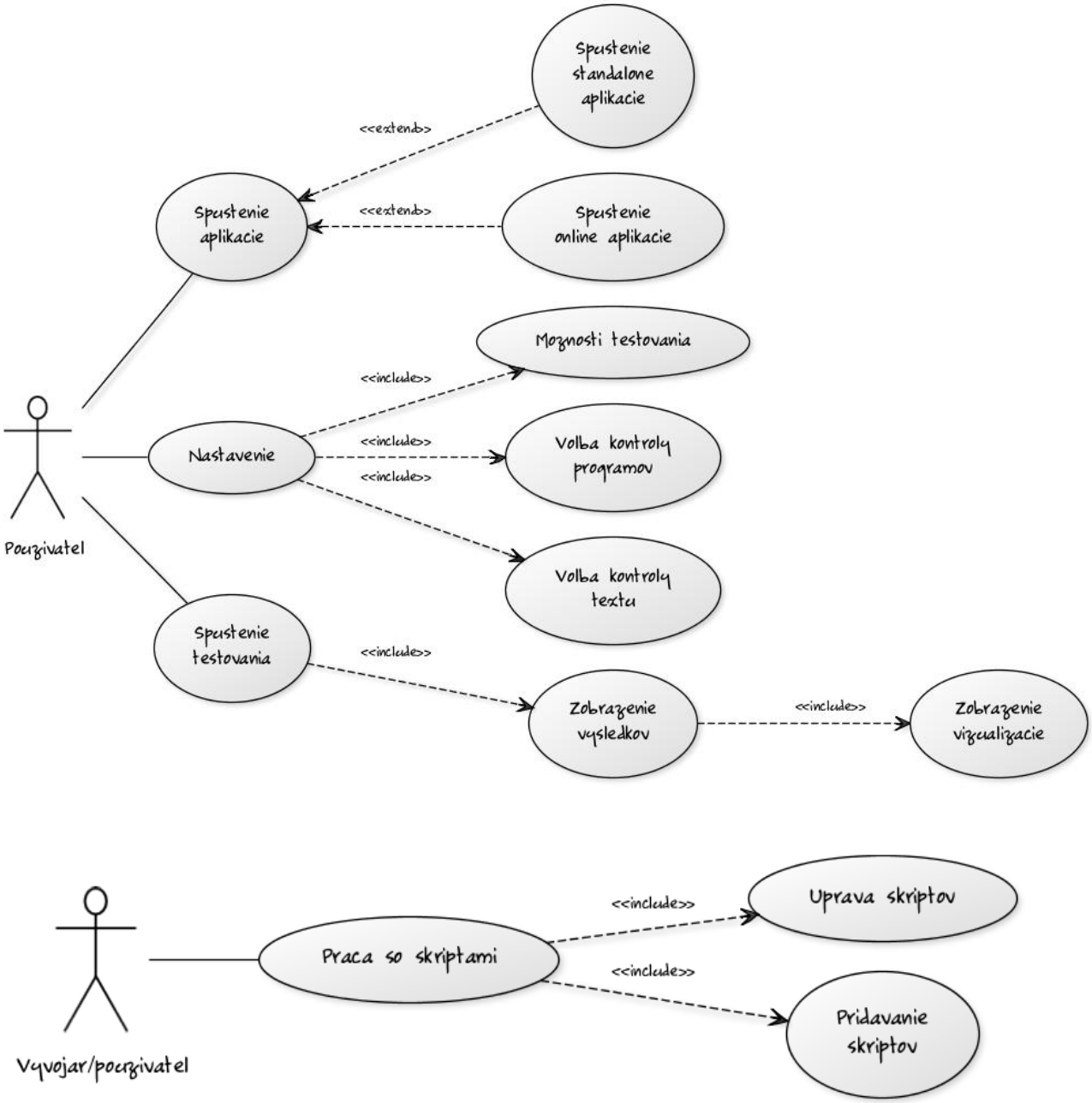
4.2 Analýza požiadaviek

Program by mal umožňovať kontrolovať zdrojové kódy (minimálne jazyky Java a C) a slovenské texty a vyhľadávať v nich plagiáty. Predspracovanie vstupných súborov (predovšetkým programových zdrojových kódov) by mala byť skriptovateľná, aby bolo možné jednoducho pridávať podporu pre ďalšie programovacie jazyky. Skripty by mali zabezpečovať celkové predspracovanie a svoj výstup poskytnúť systému na kontrolu podobnosti.
Dáta na kontrolu by mali byť získavané z viacerých zdrojov, konkrétne z pevného disku, databázy, a informačného systému (AIS, Moodle). Podporovanými vstupnými formátmi okrem textových súborov by mal byť minimálne formát Microsoft Word, prípadne aj Portable Document Format (PDF).
Aplikácia by mala ponúkať prehľadné používateľské rozhranie, ktoré by umožňovalo nastavovať parametre jednotlivých porovnávacích algoritmov, určovať súbory na kontrolu, a pod. Malo by byť možné určiť, čo s čím porovnávať – jeden dokument s množinou dokumentov, každý dokument s každým a pod.
Ďalšou funkciou by mala byť prehľadná vizualizácia výsledkov. Minimálne by to malo byť farebné zvýraznenie podozrivých častí vstupných súborov, aby tieto mohli byť následne podrobené manuálnej kontrole. Systém by mal taktiež poskytovať grafické zobrazenie štatistiky porovnávania.
Keďže porovnávanie textov (predovšetkým pre veľký počet vstupných súborov) je časovo náročná operácia, porovnávacie algoritmy by mali byť optimalizované tak, aby čo najefektívnejšie využívali hardvér počítača. Mala by byť implementovaná podpora multiprocesorových systémov (spracovávanie textov vo viacerých samostatných vláknach).

4.3 Model prípadov použitia

Na obrázku 9. sú zobrazené prípady použitia produktu a hráči ktorý vystupujú. So systémom bude pracovať v prvom rade používateľ. Keďže sa nejedná o systém so zložitou sieťovou architektúrou, ale skôr o samostatnú aplikáciu, nebude nutná jeho dodatočná údržba. Preto v prípadoch použitia

nevystupuje administrátor. Uvažujeme iba používateľa a prípadného vývojára, ktorý by chcel produkt rozšíriť pomocou skriptov.



Obr. 9.: Diagram modelu prípadov použitia

- **Spustenie aplikácie**
  Užívateľ pred prácou s produktom, musí najprv samotnú aplikáciu spustiť. Na tento prípad použitia nadväzujú nasledujúce dva, podľa toho o aký druh aplikácie sa jedná.
- **Spustenie standalone aplikácie**
  Užívateľ spustí aplikáciu na svojom počítači, kde ju má uloženú. Jedná sa o bežnú aplikáciu a po jej spustení sa zobrazí okno s jednotlivými položkami. Jednotlivými tlačítkami ovláda funkcionalitu aplikácie.
- **Spustenie online aplikácie**

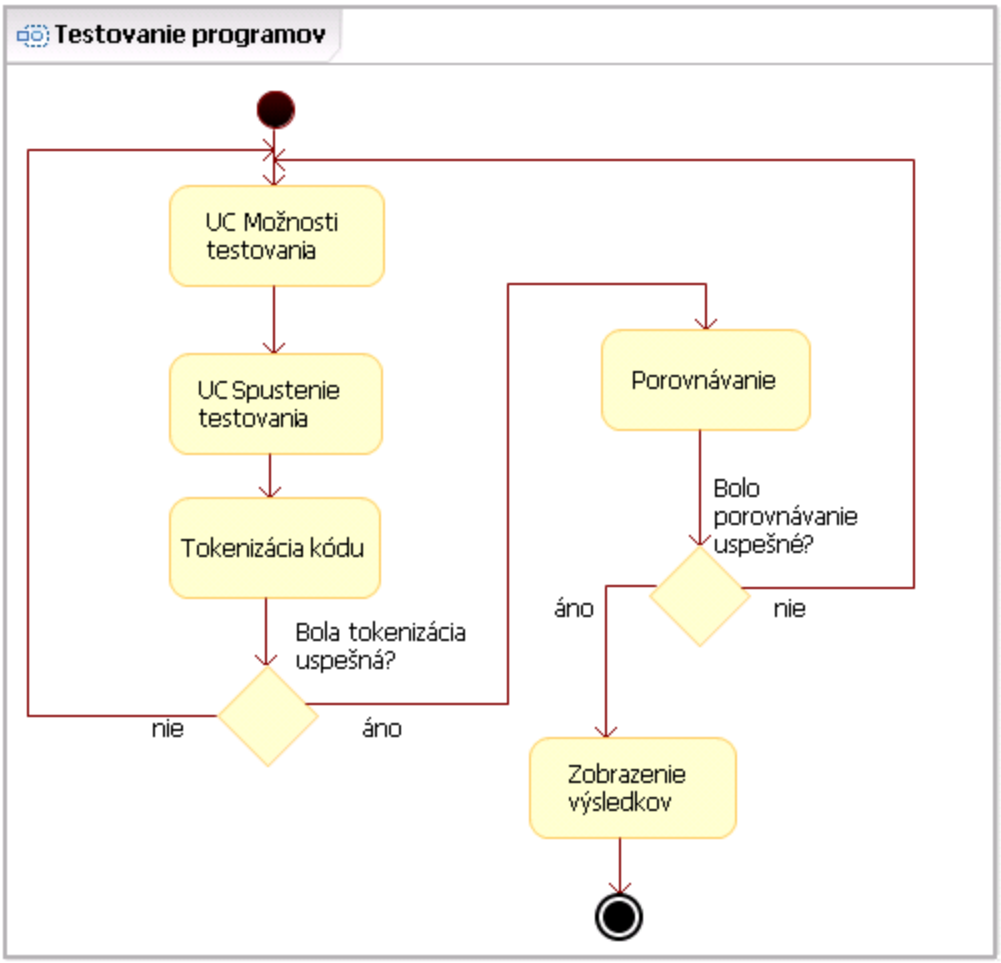
Pokiaľ užívateľ nemá aplikáciu vo svojom počítači, môže ju spustiť aj online pomocou webového rozhrania. Aplikácia sa mu zobrazí v okne prehliadača a jej funkcionalita bude rovnaká ako v predošlom prípade. Po práci s aplikáciou jednoducho zavrie okno, alebo záložku v prehliadači.

- **Nastavenie**
  Užívateľ si bude môcť prispôsobiť program. Vybrať si aký algoritmus ma program používať, ako chce mať zobrazené výsledky, kam ich ukladať a hlavne musí vybrať súbory, ktoré sa majú testovať.
- **Možnosti testovania**
  Užívateľovi sa po voľbe druhu testovania, teda či chce testovať programy alebo texty, zobrazí nastavenie tohto testovania. Pre oba druhy testovania budú niektoré spoločné nastavenia programu.
- **Voľba kontroly programov**
  Užívateľ vyberie možnosť kontrolovania zdrojových kódov programov. Po tejto voľbe sa mu zobrazí výber algoritmu, nastavenie hranice podobnosti pre podozrivé programy, formulár pre výber vstupných a výstupných zložiek a ďalšie.
- **Voľba kontroly textu**
  Pri voľbe testovania dokumentov bude taktiež môcť si voliť z dostupných algoritmov pre testovanie a predprípravu, alebo vybrať jazyk v akom je dokument napísaný, pokiaľ je dostupný.
- **Spustenie testovania**
  Po nastavení aplikácie a testovania zvolí užívateľ začatie testovanie. Počas testovania sa bude zaznamenávať a zobrazovať trvanie testu, odhadovaný čas pre dokončenie a stavový riadok s percentuálnym dokončením testu.
- **Zobrazenie výsledkov**
  Užívateľovi sa zobrazia výsledky testovania, ktoré sa taktiež uložia do zvolenej zložky na počítači, pre neskoršiu prácu s nimi. Budú to prevažne dvojice porovnávaných súborov s percentuálnymi, alebo inými zhodami podľa algoritmu a zvýraznenie jednotlivých podobných fragmentov.
- **Zobrazenie vizualizácie**
  Pre lepšiu prehľadnosť si užívateľ vyberie možnosť vizualizácie výsledkov jednou z dostupných metód. Táto vizualizácia sa zobrazí a taktiež uloží vo forme obrázku po dokončení testovania.
- **Práca so skriptami**
  Program bude ponúkať možnosť práce so skriptami. V hlavnom menu programu užívateľ zvolí kliknutím na príslušné tlačítko túto voľbu. Pri práci s online aplikáciou to bude bezpečnostne ošetrené z dôvodu neželaných zásahov do existujúcej funkcionality.
- **Úprava skriptov**
  Pri úprave skriptov sa mu zobrazí ich zoznam a užívateľ si vyberie daný skript a zvolí si či ho chce zmeniť, prípadne zmazať. Pri zmene sa mu zobrazí okno s textom, kde ho môže upravovať a ukladať zmeny. Po dokončení práce zavrie formulár voľbou návratu do menu.
- **Pridávanie skriptov**
  Pridávanie skriptov bude podobné ako ich úprava. Pri tejto voľbe sa užívateľovi zobrazí prázdne okno, kde môže napísať daný skript. Nad ním si vyberie názov skriptu a počas práce bude môcť ukladať zmeny. Po napísaní skriptu sa podobne ako pri úprave skriptov vráti späť do menu.

4.4 Procesný model

Testovanie programov

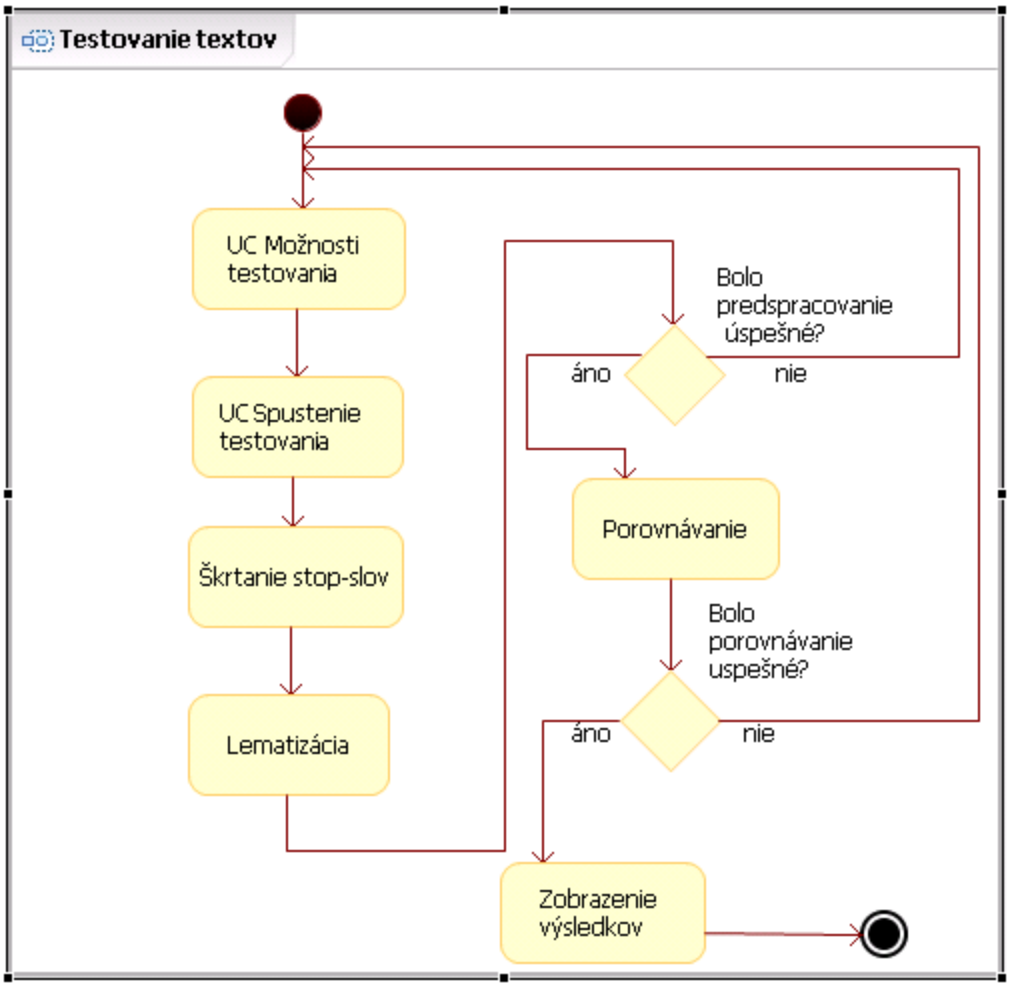
Na obrázku 10 je zobrazený model procesu testovania súborov so zdrojovým kódom. Na začiatku si užívateľ prispôsobí nastavenie testovania a následne ho spustí. Program potom najprv tokenizuje kód súborov a potom ich porovnáva. Ak pri jednom z týchto procesov došlo k chybe, program ju zobrazí a vráti sa naspäť. Na konci zobrazí výsledky testovania.



Obr. 10. Proces testovania programov.

Testovanie textov

Na obrázku 11 je zobrazený model procesu testovania súborov obsahujúcim text. Podobne ako pri testovaní programov si užívateľ najprv nastaví spôsob testovania a potom spustí samotné testovanie. Program potom najprv škrtá stop-slová a ostatné slová upravuje na základný tvar. Ak všetko prebehlo úspešne pokračuje porovnávaním a na konci zobrazí výsledok testovania. Ak došlo k chybe vráti sa na začiatok.



Obr. 11. Proces testovania textov.

5 Architektonický návrh riešenia

Architektúra systému je navrhnutá tak, aby bola možné systém jednoducho rozširovať o podporu nových programovacích jazykov (v prípade kontroly zdrojových kódov), prirodzených jazykov (kontrola textu), prípadne formátov vstupných súborov. Znázornená je na obr. 12.

Jadro (CORE)

Úlohou jadra je sprostredkovávanie komunikácie medzi ostatnými modulmi. Jadro by malo byť čo najjednoduchšie a jeho funkcionalita by mala byť čo najmenšia.

Graphical User Interface

Modul Graphical User Interface (GUI) má za úlohu získavať vstupy od používateľa a zobrazovať výstupy systému. Jeho funkcionalita by mala byť obmedzená na minimum, a všetky dáta by mu mali byť poskytnuté inými modulmi.

Command Line Interface

Command Line Interface (CLI) slúži na ovládanie aplikácie z príkazového riadku. Správanie programu je v takomto prípade ovplyvňované zadanými parametrami.

Combiner

Úlohou modulu Combiner je vytvárať dvojice vstupných súborov, ktoré budú prostredníctvom jadra odoslané modulu Parser manager na kontrolu podobnosti. Dvojice sa budú vyberať z množiny načítaných vstupných súborov na základe pravidiel, ktoré modulu Combiner poskytne modul Compare manager.

Compare manager

Compare manager definuje pravidlá, na základe ktorých sa vytvárajú dvojice súborov určených na kontrolu vzájomnej podobnosti. Vytvárané dvojice súborov sú postupne posielané modulu Parse manager. Základnou funkcionalitou modulu je vytvorenie všetkých usporiadaných dvojíc danej množiny vstupných súborov. Táto funkcionalita môže byť rozšírená o rôzne iné pravidlá vytvárania dvojíc, ktoré môžu byť užitočné pri špeciálnych prípadoch použitia systému.

Parse manager

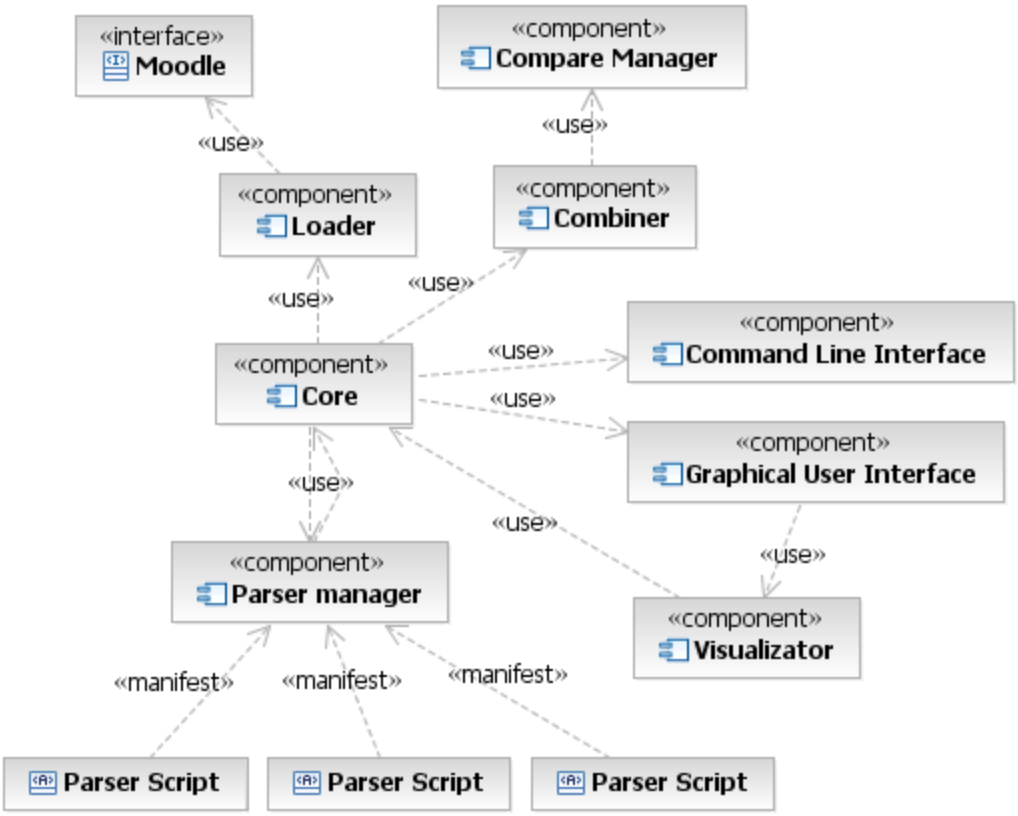
Tento modul spravuje skripty, ktoré definujú pravidlá predspracovania vstupných súborov. Vstupom pre parse manager je dvojica vstupných súborov, na ktoré je aplikovaný daný parser skript. Úlohov parser skriptov je predspracovať vstupné súbory pred aplikovaním porovnávacieho algoritmu. Porovnávacie algoritmy sú súčasťou modulu Parser manager.

Loader

Úlohou tohto modulu je načítanie textu z rôznych formátov súborov. Súbory sú získavané z pevného disku, databáz a rôznych externých systémov. Výstupom modulu je množina textových súborov, ktoré budú spracovávané systémom.

Visualizator

Vizualizačný modul má za úlohu zobraziť prostredníctvom GUI modulu porovnanie vstupných súborov, ktoré boli vyhodnotené ako podobné. Môže taktiež obsahovať vytváranie štatistík porovnávania. Dáta na vizualizáciu mu poskytne modul Core.



Obr. 12. Architektúra systému

Literatúra

[CDB] Bernstein, D. J.: CDB – Constant Database. Dostupné na internete: <http://cr.yp.to/cdb.html> [cit. 22-10-2009]

[Clough, 2000] Clough, P.: Plagiarism in natural and programming languages: an overview of current tools and technologies. *Department of Computer Science, University of Sheffield* (2000), p. 1-31.

[Češka, 2008] Češka, Z.: Využití moderních přístupů pro detekci plagiátů. In: Informačné technológie - aplikácie a teória. 2008.

[FILIT] Piaček, J., Kravčík, M.: Otvorená filozofická encyklopédia. Dostupné na internete: <http://ii.fmph.uniba.sk/~filit/fvp/parafraza.html>. [cit. 22-10-2009]

[Grune, 1989] Grune, D., Huntjens, M.: Detecting Copied Submissions in Computer Science Workshops. *Vakgroep Informatica, Faculeit Wiskunde & Informatica*, 1989.

[GSL] GSL - GNU Scientific Library - GNU Project - Free Software Foundation (FSF). Dostupné na internete: <http://www.gnu.org/software/gsl/>. [cit. 22-10-2009]

[LCS] Thanh Dao: The Longest Common Substring with Maximal Consecutive. September 2005. Dostupné na internete: <http://www.codeproject.com/KB/recipes/lcs.aspx>. [cit. 22-10-2009]

[LM] Krajčí, S., Laclavík, M., Novotný, R., Turlíková L: The tool Morphonary: Using of word lemmatization in processing of documents in Slovak. Institute of Computer Science, Univerzita Pavla Jozefa Šafárika.

[MOSS] Schleimer, S., Wilkerson, D. S., Aiken, A.: Winnowing: Local Algorithms for Document Fingerprinting. Marec 2003 [cit. 30-10-2009]. Dostupné na internete: <http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf>.

[Karp, 1987] Karp, R., Rabin, M.: Efficient Randomized Pattern-Matching Algorithms, *IBM Journal of Research and Development* 31(2), pp. 249–260 (1987).

[Sherlock] Demonstration of Sherlock - Plagiarism. Dostupné na internete: <http://www.ics.heacademy.ac.uk/resources/assessment/plagiarism/demo_sherlock.html>. [cit. 22-10-2009]

[Wise, 1993] Wise, M:String Similarity via Greedy String Tiling and Running Karp-Rabin Matching. Dostupné na internete: <http://www.pam1.bcs.uwa.edu.au/~michaelw/ftp/doc/RKR_GST.ps>. [cit. 22-10-2009]

[YAP] Wise, Michael J.: Plagiarism Detection – YAP. Dostupné na internete: <http://www.pam1.bcs.uwa.edu.au/~michaelw/YAP.html>. [cit. 22-10-2009]