VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Předzpracování zdrojových kódů pro účely detekce plagiátů

Source Code Preprocessing for Plagiarism Detection

2015 Jan Havlas

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Havlas**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Předzpracování zdrojových kódů pro účely detekce plagiátů
Source Code Preprocessing for Plagiarism Detection

Zásady pro vypracování:

Cílem práce je analýza, návrh a realizace aplikace pro předzpracování zdrojových kódů pro účely detekce plagiátů. Výsledkem předzpracování bude tokenizovaný dokument, použitelný pro některou z metod pro porovnání dokumentů.
1. Seznamte se obecně s problematikou plagiátorství a softwarového plagiátorství.
2. Seznamte se s metodami předzpracování zdrojových kódů.
3. Vyberte vhodnou metodu předzpracování s ohledem na další porovnávání dokumentů.
4. Metodu naimplementujte a otestujte na vybrané kolekci dat.

Seznam doporučené odborné literatury:

[1] Alzahrani S., Salim N. and Abraham A., Understanding Plagiarism Linguistic Patterns, Textual Features and Detection Methods, IEEE Transactions on Systems Man and Cybernetics: Applications and Reviews, IEEE, USA, 2011.
[2] Goel S., Rao D. et. al., Plagiarism and its Detection in Programming Languages. http://www.deepak-rao.info/Resources/Plagiarism.pdf

Dále dle pokynů vedoucího bakalářské práce.

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

Vedoucí bakalářské práce: **RNDr. Eliška Ochodková, 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

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 na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez jejich podpory by tato práce nevznikla.

Abstrakt

Tato bakalářská práce se zabývá plagiátorstvím a softwarovým plagiátorstvím. Cílem je pochopit, co tyto pojmy znamenají a obecně se seznámit s jejich problematikou. Dalším krokem je navrhnout a implementovat aplikaci, která by předzpracovala zdrojové kódy programů. Výstupem bude tokenizovaný nebo n-gramový dokument, který se použije pro následnou detekci plagiátu. Aplikace bude pracovat se zdrojovými kódy v C# a kromě tokenizovaného výstupu bude mít navíc možnost tokenizovaný kód s druhými porovnat. Posledním krokem bude tuto aplikaci otestovat na vybrané kolekci dat. V tomto případě se jedná o projekty studentů VŠB a následně se provede vyhodnocení.

Klíčová slova: předzpracování, porovnávání, zdrojový kód, detekce, tokenizace, levenshtein, ngram, plagiátorství, plagiát, softwarové plagiátorství, vizualizace výsledků

Abstract

This work deals with plagiarism and plagiarism software. The objective is to understand what these terms mean and generally become familiar with this issue. The next step is to design and implement an application that would preprocess source codes of programs. The output will be tokenized or n-gram document, which is used for subsequent detection of plagiarism. The application will work with source code in C# and besides tokenized output will also have the opportunity tokenized code to compare with others. The last step will be to test this application on the selected data collection. In this case it is the student projects of VSB and then will be evaluation.

Keywords: Preprocessing, Comparison, Source code, Detection, Tokenization, Levenshtein, Ngram, Plagiarism, Plagiarism software, Visualization of results

Seznam použitých zkratek a symbolů

GST	–	Greedy String Tiling
RKR-GST	–	Running Karp-Rabin Greedy String Tiling
YAP	–	Yet another Plague
ČSN	–	Česká soustava norem
MOSS	–	Measure of Software Similarity
PlaDeS	–	Plagiarism Detection System

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Plagiátorství je způsob, jak si usnadnit práci nebo se obohatit na cizí práci. Dopouští se ho stále více lidí. V dnešní době není problém si nechat zaplatit nebo naopak někomu zaplatit za vyhotovení práce. Plagiátorismu se ve výsledku dopouští jak osoba, která vytvořila poptávku, tak i člověk, který vyhotovil řešení. Za studentské plagiátorství a jeho dokázání hrozí dotyčnému disciplinární řízení a možné vyloučení ze školy. Při usvědčení a prokázání hrozí odebrání titulu.

Cílem práce je analýza, návrh a realizace aplikace pro předzpracování zdrojových kódů pro účely detekce plagiátů. Výsledkem předzpracování bude tokenizovaný dokument použitelný pro některou z metod pro porovnání dokumentů. Aplikace bude, kromě zmíněného výstupu, navíc disponovat možností porovnat tokenizované nebo předzpracované kódy mezi sebou a zobrazit, jak moc jsi jsou podobné.

První kapitola odpovídá na otázku, co je to plagiátorství a popisuje blíže jeho rozdělení do příslušných podkapitol.Druhá kapitola popisuje softwarové plagiátorství. Popis modifikací kódu, kterými se snaží hříšníci zamaskovat plagiátorství, metody pro předzpracování a pro analýzu podobnosti. Vizualizaci výsledků a zmínění se o již existujících nástrojích pro detekci plagiátů. Třetí kapitola popisuje realizaci vlastní aplikace, prvotní analýzu, návrh a implementaci, použité metody pro předzpracování zdrojových kódů. Použité technologie, ovládání aplikace a systémové požadavky pro bezproblémový běh aplikace, třídní diagram pro přehled jednotlivých tříd, proměnných a metod používané v aplikaci. Čtvrtá kapitola popisuje testování aplikace na kolekci zdrojových kódů. Poslední kapitola s názvem „Závěr“ popisuje dosažené výsledky a splnění podmínek zadané práce.

2 Plagiátorství

2.1 Co je to plagiátorství

Odpověď na otázku, co je to plagiátorství, si můžeme zjednodušit tím, že si odpovíme na otázku, co je to plagiát. Plagiát popisuje norma ČSN ISO 5127-2003 jedná se o „představení duševního díla jiného autora půjčeného nebo napodobeného v celku nebo z části, jako svého vlastního" [1]. Autor díla, které bylo označené za plagiát, se tedy dopustil plagiátorství. Vydával cizí dílo za své, a to i přesto, že autorem nebyl on sám.

2.2 Druhy plagiátorství

V podkapitole druhy plagiátorství si představíme nejrozšířenější a často používané formy plagiátorství.

- Metoda "Ctrl+C"a "Ctrl+V"- bez uvědomění autora díla je obsah převzat a vydáván za své vlastní dílo.
- Použité texty z více zdrojů zformulovat do samostatného díla, kdy obsah citací přesahuje obsah vlastního textu. Použití určité pasáže z cizího díla nebo ji i předem nedostatečně, nebo špatně parafrázovat bez uvedení zdroje před vložením do své práce.
- Špatně použité citace, neuvedení všech autorů. Může se jednat např. o zapomenuté vložení jednoho z autorů.

Představili jsme si pár základních problémů. Na straně jedné se může jednat o úmyslné kopírování a na straně druhé se může jednat o chyby neúmyslné. V dalších podkapitolách si popíšeme podrobněji jednotlivé druhy a jak se jich můžeme dopustit.

2.2.1 Co je to neúmyslné plagiátorství

Mohlo se stát, že při uvádění zdrojů, autorů se mohlo na jednoho z autorů zapomenout. Při uvádění citace je nutné dodržovat i její normu ČSN ISO 690:2011 [2]. Může se tedy stát, že uvedeme zdroj ve špatném formátu. Další možnost může byt nedostatečná nebo špatně provedená parafráze, když se původní myšlenka rozchází s aktuálně upravenou. Následně k tomu může být vynechán zdroj, odkud jsme čerpali. Jedná se tedy o chyby zaviněné nedbalostí, neznalostí a autor se jich dopustil především neúmyslně.

2.2.2 Co je to úmyslné plagiátorství

Osoba se vědomě dopouští, anebo dopustila plagiátorství. Důvodem, proč se toho dopouští, může být např. jedna z možností uvedená v seznamu.

- Usnadnění si práce a tím ušetření si času s vymýšlením vlastního řešení.
- Neschopnost realizovat, popsat vlastními slovy danou práci.

- Nepochopení zadání a tedy neschopnost vypracovat danou práci.
- Lenost - okopíruje cizí dílo s vidinou ušetření času při jeho realizaci.
- Vytíženost - ví, že nestíhá a tedy okopíruje větší část práce s tím, že mu nezbývá jiná možnost. Snaží se zachránit situaci.

V dnešní době je velmi populární si obstarat už vyhotovenou práci. Nechat si práci vypracovat spolužákem nebo skupinou, která se o tyto věci stará. Může se jednat o projekty, úlohy nebo i dokonce seminární a bakalářské práce. Existují webové stránky, skupiny lidí, které se specializují na tyto služby a nabízejí hotová řešení. Zaručují, že práce jsou kvalitně zpracované, že jsou ve většině případu rychle vyhotovené a samozřejmě za přiměřenou peněžní částku ihned k dispozici.

2.2.3 Co je to kryptomnézie

Kryptomnézie se dopouští lidé především nevědomě [3]. Dochází k tomu, že člověk se s danou myšlenkou setkal v minulosti a nyní k ní přistupuje jako k vlastní. Pravdou však zůstává to, že autorem dané myšlenky nebyla tatáž osoba. Tento problém lze z větší části vyřešit tak, že si před samotnou prací uděláme někde na disku složku nebo soubor, kde odkazy na nalezená díla budeme ukládat. Následně, pokud si nebudeme jistí, že daná myšlenka už nezazněla v jiném díle, si danou myšlenku můžeme i s danou informaci zpětně dohledat. Pokud se však jedná o vzpomínky, myšlenky s kterými jsme se mohli setkat před několika lety, může se stát, že i zde vycházíme už z nějakého díla, ale dohledání už je téměř nemožné.

2.2.4 Co je to auto-plagiátorství

Jedná se o tzv. „auto-plagiátorství“ a nebo také „self-plagiátorství“. Plagiátorství, kterému se může vystavovat neúmyslně přímo autor díla. Dopouští se ho tím, že v konkrétní práci se odkazuje na práce z let minulých, bez potřebné reference na ní. Autor tak může ukazovat na to, že daný problém má řešení a že už byl úspěšně v minulosti vyřešen. Následný čtenář pak může mít problém takové dílo dohledat, ověřit si ho. Autor tedy odkazuje na dřívější práci bez uvedení zdroje.

2.3 Prevence

Prevencí je nebrat plagiátorství na lehkou váhu, uvádět korektně zdroje a při používání citací dodržovat její normu. Při parafrázování a přebírání cizích části textů jednoznačně označit, kde začíná a kde taky končí daný text. Správně označit a uvést do zdrojů autora a odkud jsme čerpali (článek, literatura). Promyslet si dopředu o čem budeme psát a popsat to nejlépe vlastními slovy a citace používat jen v omezeném počtu. Vyhotovené dílo neskládat jen z textu cizích děl bez vlastního přičínění. Citace používat jen v omezeném počtu.

2.4 Theses.cz

Theses je systém, který se stará o kontrolu studentských prací, hlavně tedy bakalářských a diplomových prací. Má na starost odhalovat plagiátory v řadách studentů. Tento systém vznikl na Masarykově univerzitě, kde je i nadále vyvíjen a vylepšován. Podílí se i na dalších projektech, které jsou uvedeny v tomto seznamu.

- Odevzdej.cz - zaměřuje se na plagiáty v seminárních prácích.
- Repozitar.cz - slouží jako repositáři vědeckých prací.
- PravyDiplom.cz - ověřování pravosti čísla diplomu.

Theses byl založen roku 2008. Aktuálně se do projektu připojilo 43 škol z celé ČR, které využívají tento systém [4]. Přispívají k tomu, aby se nestávalo jako v minulosti, že studenti obhajovali už vyhotovené práce z let minulých.

3 Softwarové plagiátorství

3.1 Co je to softwarové plagiátorství

Definice 3.1 *Softwarový plagiát je program, který byl vytvořen na základě jiného programu po provedení menšího počtu úprav [5].*

Je-li možné pomocí jednoduchých operací transformovat program na program jiný, jedná se s větší pravděpodobností o kopii. Tento program by měl být dále prozkoumán a vyhodnocen, zda se jedná opravdu o kopii a tedy plagiát. Pro zjištění, zda se jedná o plagiát, je však nutno rozumět danému programovacímu jazyku, jeho syntaxi. Na řadu zde přicházejí aplikace, které zhodnotí, rozeberou zdrojové kódy těchto programů a zobrazí, jak moc si jsou kódy podobné. Na výstupu zobrazí hodnocení a např. zvýrazní podobné, nebo stejné části kódu. Postup pro odhalení softwarového plagiátů lze popsat v jednotlivých krocích:

1. Předzpracování zdrojového kódu.
2. Použití metody, která porovnává míru podobnosti mezi dvěma kódy.
3. Vizualizace výsledků - Zobrazení výsledků, tak aby bylo na první pohled vidět, které kódy jsou si podobné a jak hodně, např. v procentech.

3.2 Modifikace kódu

Modifikací zdrojového kódu rozumíme to, že kód byl upraven s úmyslem zamaskovat stopy plagiátorství. Jednotlivými úpravami změníme v nejjednodušších případech vzhled a v složitějších to může znamenat změnění struktury kódu. Může se jednat o přejmenování názvu tříd, metod, proměnných buď jinými názvy nebo překladem do jiného jazyku. Další případ může být vypouštění nebo přidávání nadbytečných a nepoužívaných metod, které se v kódu nikde nevolají a podobně. Na začátek si popíšeme, o jaké úpravy kódu se může jednat.

1. Kód nebyl nijak upraven, jedná se o kopii originálu, což je patrné na první pohled.
2. Z kódu byly odstraněny, upraveny nebo naopak přidány nové komentáře. Byly přidány prázdné řádky, tím se myslí, že na daných řádcích se nenachází žádná slova nebo znaky. Potom se může jednat o přidání mezer tzv. „bílých znaků“, mezer tabulátorem v počtu větším než jedna. V tom případě osoba, která kód napsala, se snaží vyvolat dojem, že se jedná o zcela jiný kód. Jedná se o úpravy, které mění formát, vzhled kódu.
3. Přejmenování různých názvů proměnných nebo názvů metod, identifikátorů. Může se využít např. anglických názvů metod, proměnných a přeložit je do češtiny a změnit tím znatelně obsah kódu. To vše stále bez znalosti programovacího jazyka.

4. Změny provedené úpravou pořadí řádků nebo posloupností metod v kódu. První se vloží metoda „B“ namísto metody „A“ a metoda „C“, která se nacházela na konci, se vloží na začátek. Proměnné, které se deklarovaly v metodě se vloží na začátek a budou se deklarovat jako globální. Může se změnit obsah metody, kdy v dané metodě, pokud je to možné zpřehází se řádky. Místo přístupu public se budou používat private, protected, samozřejmě jen tam, kde to lze. Místo datového typu int použít float nebo double a tím přetypovávat na jiné datové typy. Pro představu je připravena ukázka, jak by se kódy mohly lišit:

```
public class Program
{
    public static void Print()
    {
        int number1 = 3;
        int number2 = 5;
        string str1 = "a␣=␣";
        string str2 = "\nb␣=␣";
        Console.WriteLine(str1 + number1 + str2 + number2);
    }

    public static void Main(/*string[] args*/)
    {
        Print();
        Console.ReadKey();
    }
}
```

Výpis 1: Program v jazyce C# - původní kód

```
class Program
{
    private static void Main(/*string[] args*/)
    {
        Tiskni();
        Console.ReadKey();
    }

    private static void Tiskni()
    {
        string retezec1 = "a␣=␣";
        float cislo2 = 5;
        string retezec2 = "\nb␣=␣";
        float cislo1 = 3;
        Console.WriteLine(retezec1 + cislo1 + retezec2 + cislo2);
    }
}
```

Výpis 2: Program v jazyce C# - upravený kód

5. Změna cyklu: např. vyměníme **while** cyklus za **do while** nebo cyklus **for**. Mohou se řešit dílčí operace v jednotlivých metodách namísto jedné metody, která by obsahovala všechny operace. Pro každou složitější operaci vytvořit novou metodu, která by danou operaci prováděla a v části, kde se operace nachází, ji nahradíme voláním metody. Jde nám tedy o to, abychom nahradili část kódu za nově vytvořenou metodu, která by se volala na místě, kde dřív daná metoda byla. Zde je už nutné mít alespoň základní programátorské znalosti.

6. Při řešení funkce můžeme změnit styl řešení, ale zachovat její funkcionalitu. Můžeme v třídě vytvořit další metodu s dvěma parametry vracející výsledek a volat ji při stisknutí tlačítka. V neposlední řadě se vytvoří další třída, která by vykonávala jen tuto operaci. Další možnost je založit nový projekt s názvem číselné operace. Podobných možností a způsobů, jak řešit zadanou úlohu, problematiku, je mnoho.

7. Z kódu se odstraní nepotřebné části, popřípadě si je nadeklarujeme jinde ve vytvořeném projektu. Do kódu jsou přidávany nadbytečné časti, které nejsou v kódu nijak dosažitelné. Jedná se o metody, které nejsou v programu volány, používány. Tento druh úprav lze v kódu už jen těžko dohledat. Kód mohl být změněn k nepoznání, je pouhým okem nerozpoznatelný od původního kódu, a to i dokonce ve většině případů i detekčních programů. Z těchto úprav vyplývá, že daná osoba je programátorem a ví, jak kód upravit. Mění obsah, strukturu, ale to co má program udělat, udělá velice podobným způsobem. Může se jednat o přepsání kódu, např. z jazyku Java do C#, případně jiného.

3.3 Metody pro předzpracování kódu

3.3.1 Předzpracování

Předzpracování je proces, kterým kód upravujeme, zjednodušujeme a odstraňujeme z něj nepotřebné a nedůležité části, které neovlivní v žádném případě jeho původní význam. Nejedná se o úpravy, které by měly za výsledek změnu chodu programu. Díky úpravám můžeme kód lépe a efektivněji porovnávat s jinými kódy za předpokladu, že byl kód menším počtem úprav pozměnen. Druhá výhoda je ta, že kód po předzpracování bude následně při porovnávání rychleji zpracováván. Pro představu uvádíme možné úpravy, kterými se dá kód upravit:

- Odstranění prázdných míst, mezer mezi slovy, avšak výhodnější je tyto mezery odstranit jen tam, kde jich je více než jedna vedle sebe. Prázdné místo tedy odstraníme jen pokud je počet mezer >1. Za mezeru považujeme i vytvořenou mezeru tabulátorem.

- Odstranění prázdných řádků, které neobsahují žádný kód. V určitých případech se odstraňuje i řádkování. Tento způsob by byl výhodný, pokud bychom už dále kód neměli v plánu upravovat a chtěli mít výstupný kód na jednom řádku.

- Odstranění přebytečných částí z kódu, které nejsou pro porovnávání důležité. Můžeme ignorovat některé znaky nebo posloupnost znaků o kterých víme, že při porovnávání nevyužijeme.
- Odstranění nebo úprava komentářů. Na jedné straně se nejedná přímo o programovou část kódu, ale některé detekční programy je přesto využívají při detekci plagiátů. Pokud tedy komentáře neodstraníme, je nutné je upravit: odstranit mezery, převést všechny znaky na malé, odstranit diakritiku atd.
- Převést všechny znaky na malé. Důvod může být např. zamezit přehlížení stejných názvů, kdy jednou se metoda může jmenovat jako „NačtiSoubor“ a v dalších případech „načtisoubor, Načtisoubor, načtiSoubor“. Ve výsledku tedy získame při porovnávání jen jedinou možnou variantu „načtisoubor“.
- Odstranění řádků, které začínají znakem #, označují pre-procesorové direktivy, za znakem # je název direktivy. Direktivy jsou v programovacím jazyku zpracovávany programovacím jazykem, kompilátorem.
- Odstranění řádků, které slouží k snadnějšímu přístupu k jinak nepřístupnám metodám. V programovacím jazyku C# by to bylo např. direktiva „using System.IO;“. Using nám umožňuje přístup k určitému prostoru jmen. Pro představu uvádíme ukázku v tabulce 1, kde v prvním případě na začátku kódu vložíme:
  - using System.Text.RegularExpressions;
  - using System.IO;
  - using System;

Bez použití direktivy	S použitím direktivy
System.Text.RegularExpressions.Regex(...);	Regex(...);
System.IO.File.Exists(...)	File.Exists(...)
System.Console.WriteLine()	Console.WriteLine(...)

Tabulka 1: Ukázka výhod použití direktivy v jazyku C#
Na první pohled v tabulce 1 si lze všimnout rozdílu v délce příkazu. Pokud tedy nevyužijeme direktivy using, museli bychom opakovaně při každém přístupu vypisovat její cestu.

- Přepsání obsahu řetězců za jeden znak, který ho bude reprezentovat.
- Přepsání jednotlivých znaků („char a = ’a’; “ na „char a = ’;“) a podobně.
- Převedení všech čísel na nulu.
- Nahrazení volání metod jejím obsahem: „tělem“. Zamezí se tomu, že osoba, která kód upravila, mohla změnit posloupnost metod. Další možností je otestování, jestli se metody vyskytované v zdrojovém kódu volají a pokud ne, tak je z kódu odstranit.

Po provedení uvedených úprav můžeme na takto upravený kód použít jednu z metod pro porovnávání podobnosti. Při porovnávání máme pak větší šanci na odhalení plagiátu, avšak je nutné vybrat metodu, u které víme, že má nejlepší možné výsledky. Pro pochopení procesu předzpracování jsem připravil na ukázku kód, který není předzpracovaný 3 a v druhé ukázce kód po předzpracování 4.

```
using System;
using System.Text.RegularExpressions;
using System.IO;

namespace Test
{
    class Program
    {
        private static void meTodaA(int N, char type) // metoda vypisujici hodnoty od 1 do N
        {
            for
                (int i =    1;      i <= N; i++)
            {
                if (type == 'S' && i % 2 == 0) // cisla jsou suda
                    Console.WriteLine(i);
                else if (type == 'L' && i % 2 != 0) // cisla jsou licha
                    Console.WriteLine(i);
                else if (type != 'L' && type != 'S') // pokud neni 'S' ani 'L' vypis vsechna cisla
                    Console.WriteLine(i);
            }
        }
        // hlavni metoda, ktera se stara o nacteni znaku z konzole a zobrazeni vysledku
        private static  void Main(string[] args)
        {
            Console.WriteLine("Pro␣vypis␣sudych␣cisel␣zadejte␣-␣'S',␣pro␣liche␣-␣'L'");
            char type =                 Convert.ToChar(Console.ReadLine()); // pouze char 'L', 'S'
                bude bran v potaz
            Console.Clear();
            Console.WriteLine("Zadejte␣cislo,␣ktere␣bude␣reprezentovat␣maximalni␣hodnotu:");
            int maximum =                 Convert.ToInt32(Console.ReadLine());
            Console.Clear();
#pragma warning disable
            if (type != 'S' &&
                                                                    type != 'L')
                type = '-';
#pragma warning restore
            string rsltr = "Vystup␣v␣rozmezi␣od␣1␣do␣" + maximum + "␣typu␣" + type;

            Console.WriteLine(rsltr);
            meTodaA(maximum, type);

            Console.ReadKey();
        }
    }
}
```

Výpis 3: Program v jazyce C# - kód před předzpracováním

```
namespace test
{
class program
{
private static void metodaa(int n, char type)
{
for
(int i =0;i <= n; i++)
{
if (type == ' && i % 0 == 0)
console.writeline(i);
else if (type == ' && i % 0 != 0)
console.writeline(i);
else if (type != ' && type != ')
console.writeline(i);
}
}
private static void main(string[] args)
{
console.writeline(");
char␣type␣=convert.tochar(console.readline());
console.clear();
console.writeline(");
int maximum =convert.toint0(console.readline());
console.clear();
if (type != ' &&
type != ')
type = ';
string rsltr = "␣+␣maximum␣+␣" + type;
console.writeline( rsltr );
metodaa
(
maximum,
type
);
console.readkey();
}
}
}
```

Výpis 4: Program v jazyce C# - kód po předzpracování

3.3.2 N-gramy

Definice 3.2 *"N-gram je definován jako sled **n** po sobě jdoucích položek z dané posloupnosti."[6].*

N-gram může být sekvencí slov nebo písmen. Využití má jak v normálním textu, tak i pro zdrojový kód. U n-gramů, které se používají u zdrojového kódu je výhodné používat posloupnost slov. Na zdrojový kód se pak nenahlíží jako na celek, ale rozdělí se na tzv. n-gramy skládající se ze slov. Když si tedy vezmeme dva předpřipravené soubory v n-gram podobě, můžeme je porovnat, jak moc jsou si podobné. Zda jsou n-gramy obsažené

v prvním souboru obsažené i v druhém a v jakém počtu. Na základě této podobnosti jsme pak schopni se rozhodnout, zda jsou si dva soubory podobné. Při vytváření n-gramů je potřeba dosadit za **n** číslo, která určuje jeho délku.

- n = 1 se nazývá „unigram“
- n = 2 se nazývá „bigram“
- n = 3 se nazývá „trigram“
- n > 4 se nazývá „čtyř-gram“, „pěti-gram“, atd.

Je tedy nutné vybrat tu nejlepší možnou variantu. Můžeme se rozhnodnout na základě testování, porovnávání výsledků u vytvořených n-gramů. Např. při délce n<3 bude docházet k vyhledávání příliš mnoha stejných dvojic, ale s větší pravděpodobností falešné detekce, anebo n>5, kdy při porovnání může zase docházet k přehlížení menších dvojic a plagiát nemusí být detekován. Vytvořením n-gramů slouží k eliminaci základních praktik, které slouží k zamaskování plagiátu. Bereme tedy v potaz, že kód mohl být upraven tak, aby se snížila míra odhalení. Změní se tedy část kódu nebo se změní posloupnost kódu. N-gramy se snaží tento problém řešit, místo porovnáváni zdrojového kódu jako jeden velký řetězec jej rozdělí do menších částí. Můžeme zjistit počet opakování nalezených shod mezi dvěma porovnávanými soubory.

3.3.3 Tokenizace

Na úvod si řekneme, co se skrývá pod slovem token. Token je řetězec délky jednoho nebo více znaků, kterým nahrazujem původní slovo, znak.Tokenizace značí proces, který část kódu nebo slova převede na tokeny. Tokenizace slouží k zjednodušení, k zobecnění určitých částí kódu, které se často opakují. Při tokenizaci zdrojového kódu je nutné znát jeho sémantiku, charakteristiku programovacího jazyka proto, aby se dokázal daný zdrojový kód co nejlépe zoptimalizovat, zjednodušit. Díky těmto úpravám můžeme následně porovnávat shodu mezi soubory s větší pravděpodobností odhalení plagiátu. Využívají je např. aplikace Yap3, Plague, JPlag zmíněné v kapitole 3.6. Zaměřují se na porovnáváni zdrojového kódu mezi soubory. Každé slovo a znak je zaměněn za token reprezentující jejich sémantický význam. Tyto skupiny tokenů jsou následně používané v post-processingu „post-zpracování“ tokenu buď parserem (analyzátorem) nebo jinou funkcí v programu. Při tokenizaci je výhodné seřazovat tokeny do jednotlivých skupin, např. interpunkcí, funkcí, parametrů. Připravil jsem v tabulce 2 seznam, pár příkladů jak probíhá tokenizace na jednotlivých slovech, znacích daného programovacího jazyka. Může se jednat o slovo a nebo přímo znak, který je následně nahrazen obecnější formou. Pokud se však jedná o specifické slovo, znak se opatří bloky „[]“, které obsahují uvnitř daný text.

Před tokenizaci	Po tokenizaci
int, string, float, ...	[VALUE_TYPE]
+=,-=, ...	[ASSIGMENT]
{	[BLOCK_START]
}	[BLOCK_END]
>, <,>=,<=	[COMPARE]
true, false	[LITERAL]
string str;	[ASSIGN][IDT]
private void RenameFunc()	[ACCESSIBILITY][VOID][FUNC]
int a = 5;	[VALUE_TYPE][IDT][ASSIGMENT][NUMBER]

Tabulka 2: Ukázka tokenizace

Zobrazená tokenizace slouží pouze jako ukázka možného řešení tokenizace. Nyní si předvedeme ukázku tokenizace na kódu 5.

```
// http :// www.tutorialspoint.com/csharp/csharp_while_loop.htm
using System;

namespace Loops
{

    class Program
    {
        static void Main(string[] args)
        {
            /* local variable definition */
            int a = 10;

            /* while loop execution */
            while (a < 20)
            {
                Console.WriteLine("value of a: {0}", a);
                a++;
            }
            Console.ReadLine();
        }
    }
}
```

Výpis 5: Program v jazyce C# - kód před předzpracováním

Nyní si zobrazíme výstup kódu v tokenizované podobě:
[NAMESPACE][IDT][BLOCK_START_NAMESPACE][CLASS][IDT]
[BLOCK_START_CLASS][MODIFIER][VOID][MAIN][FUNC][BLOCK_START_PARAM]
[VALUE_TYPE][VALUE_TYPE][BLOCK_END_PARAM][BLOCK_START_MAIN]
[VALUE_TYPE][IDT][ASSIGMENT][NUMBER][BLOCK_START_LOOP][FUNC:WRITE]
[BLOCK_START_PARAM][STRING][BLOCK_END_PARAM][IDT][UNARY]
[BLOCK_END_LOOP][FUNC:READ][BLOCK_END_MAIN][BLOCK_END_CLASS]
[BLOCK_END_NAMESPACE]

3.4 Metody pro analýzu podobnosti

3.4.1 Porovnávání řeťězců

Při porovnávání řetězců se zdrojový kód bere jako celek. Před samotným porovnáváním se provede předzpracování, kdy se kód převede do čitelnější, přehlednější podoby. Předzpracování, bylo popsáno v podkapitole 3.3.1. Při porovnávání řetězců se vybere algoritmus, který vyhledává podobné podřetězce. Pokud kód přesáhne určitou míru podobnosti, je kód považován za podobný. Jedná se o nejjednodušší metodu pro porovnávání řetězců, ale na porovnání podobnosti mezi zdrojovými kódy se nehodí. Plagiátoři mohli zdrojový kód upravit, změnit pojmenování parametrů, názvy metod, přidat kód navíc nebo ho i zkrátit. Následné vyhodnocení podobnosti pak ztrácí smysl. Avšak pro výpočet míry podobnosti mezi dvěma řetězci lze použít např. algoritmy [7], kde za **x**, **y** dosadíme řetězce, které chceme mezi sebou porovnat.

1. Dice míra podobnosti:
$$D(x, y) = \frac{2|x \cap y|}{|x \cup y|}$$

2. Cosinová míra podobnosti:
$$Cos(x, y) = \frac{\sum_i (x_i, y_i)}{\sqrt{\sum_i (x_i)^2}\sqrt{\sum_i (y_i)^2}}$$

3.4.2 Porovnávání tokenů

Porovnávání tokenů pracuje na podobném principu, jako tomu bylo u porovnávání řetězců, viz kapitola 3.4.1. U porovnávání tokenů ale kromě úprav, kterým se říká předzpracování, se kód převádí na tzv. tokeny. Převádění kódu na tokeny se nazývá tokenizace, viz kapitola 3.3.3. Po tokenizaci máme tedy kód upravený tak, že dokážeme detekovat plagiát, i když byly učiněny kroky pro jeho zamaskování. V dalším kroku se provede podobný algoritmus, který byl použit u porovnávání podřetězců s tím rozdílem, že se vyhledají podobné posloupnosti tokenů. Na konci se vyhodnotí míra podobnosti mezi porovnávanými kódy.

3.4.3 Greedy String Tiling

Greedy String Tiling (GST) byl vytvořen roku 1993. Autor algoritmu je Michael J. Wise. Veškeré informace o tomto algoritmu si lze přečíst v dokumentu [8]. Algoritmus využívají v dnešní době např. aplikace na detekci plagiátů SIDPLAG, YAP zmíněné v podkapitole Existující nástroje 3.6. Jedná se o algoritmus, který provádí porovnávání mezi dvěma řetězci. Porovnává shodu mezi řetězci a může být použit i pro porovnávání zdrojových kódů. Využívá tokenizaci pro převedení kódu do obecnější podoby. Algoritmus porovnává nejdelší možnou podobnost řetězců, tokenů mezi např. originálem a kontrolovaným kódem. Zjišťuje do jaké míry jsou si dva řetězce podobné na základě porovnávání společných posloupností podřetězců. Pokud dojde ke shodě tokenů mezi porovnávaným

kódem a originálem, takto shodný kód se už dále nebude porovnávat. Existuje zde pravidlo, že se může s řetězcem, který se shoduje s jiným řetězcem, pracovat jen jednou a přejde se na další, aby se předešlo zacyklení. Při porovnávání se nehledí na pořadí řádků, tím se předejde případu, kdy programátor změní pořadí, aby znesnadnil jeho odhalení. Při odkazování na dva řetězce, kratší z nich bude označen jako vzor, zatímco delší bude označen jako textový řetězec. Časová složitost algoritmu je O(n^3). Pro zobrazení podobnosti mezi porovnávanými kódy si označíme první řetězec jako A a druhý B. Dále se zde nachází „tiles“ označující množinu a „match“ list obsahující shody potvrzené v řetězci. Další parametr je „length“ označující délku shody, dále „a“ určující pozici v řetězci „A“ a pozici „b“ v řetězci „B“. Následně můžeme spočítat míru podobnosti mezi těmito dvěma řetězci pomocí vzorce uvedeného v dokumentu [8]

$$sim(A, B) = \frac{2 \times coverage(tiles)}{(|A| + |B|)}$$

$$coverage(tiles) = \sum match(a, b, length) \in tiles$$

Výsledkem bude výsledná míra podobnosti mezi těmito dvěma řetězci.

3.4.4 Running Karp-Rabin Greedy String Tiling

Running Karp-Rabin Greedy String Tiling (RKR-GST) je rozšířením algoritmu GST o Running Karp Rabin algoritmus. Karp-Rabin byl vytvořen roku 1987 a autory jsou Richard. M. Karp a Michael. O. Rabin. Jako jeho předchůdce GST byl algoritmus vytvořen pro vyhledávání podřetězců. Podrobnější popis algoritmu se nachází v [8].

3.4.5 Levenshteinova vzdálenost

Jedná se o algoritmus, který porovnává dva řetězce [9]. Pro usnadnění si označíme řetězce písmeny A a B. Pomocí Levenshteinova algoritmu se vypočítá, kolik úprav je potřeba provést, aby mohl být řetězec A přetransformován na řetězec B. Mezi tyto operace patří přidání „Insertion“, odebrání „Deletion“ nebo použití substituce „Substitution“ na každém znaku v daném řetězci. Algoritmus dostal jméno po jeho autorovi Vladimíru Levenshteinovi, který definoval tuto vzdálenost roku 1965. Levnshteinovu vzdálenost vypočítáme podle:

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} max(i,j) & if min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & otherwise \end{cases}$$

Obrázek 1: Výpočet Levenshteinovy vzdálenosti

Levenshteinovu vzdálenost se vyplatí používat na porovnávání kratších řetězců. Při počítání vzdálenosti u větších řetězců se potýká algoritmus s nárůstem výpočetního výkonu a tedy se doporučuje používat na menší řetězce. Je to z toho důvodu, že je potřeba

projít dva řetězce, kde se délka prvního plus délka druhého použije k sestrojení pole. Pole může následně nabývat obrovské velikosti. Následná práce s daným polem a daty, které obsahuje může mít za následek zpomalení a tedy prodloužení doby výpočtu vzdálenosti.

Příklad 3.1
Pro ukázku jak se řeší dva řetězce pomocí Levenshteinovy metody si uvedeme příklad.

$$A = "Dneskajehezky." B = "Dnesbylohezky."$$

Jak algoritmus řetězce zpracuje si zobrazíme na obrázku 2.

	ε	D	n	e	s		b	y	l	o		h	e	z	k	y	.
ε	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
D	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
e	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
s	4	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12
k	5	4	3	2	1	1	2	3	4	5	6	7	8	9	9	10	11
a	6	5	4	3	2	2	2	3	4	5	6	7	8	9	10	10	11
	7	6	5	4	3	2	3	3	4	5	5	6	7	8	9	10	11
j	8	7	6	5	4	3	3	4	4	5	6	6	7	8	9	10	11
e	9	8	7	6	5	4	4	4	5	5	6	7	6	7	8	9	10
	10	9	8	7	6	5	5	5	5	6	5	6	7	7	8	9	10
h	11	10	9	8	7	6	6	6	6	6	6	5	6	7	8	9	10
e	12	11	10	9	8	7	7	7	7	7	7	6	5	6	7	8	9
z	13	12	11	10	9	8	8	8	8	8	8	7	6	5	6	7	8
k	14	13	12	11	10	9	9	9	9	9	9	8	7	6	5	6	7
y	15	14	13	12	11	10	10	9	10	10	10	9	8	7	6	5	6
.	16	15	14	13	12	11	11	10	10	11	11	10	9	8	7	6	**5**

Obrázek 2: Ukázka Levenshteinovy vzdálenosti

Výsledek se tedy rovná pěti. Pro výpočet podobnosti použijeme Jaccardovu[1] vzdálenost. Za parametry „A, B“ dosadíme jednotlivé řetězce a „LD“ nám vrátí výslednou Levenshteinovou vzdálenost.

$$sim(A, B) = 1 - \frac{LD(A, B)}{max(|A|, |B|)}$$

[1] http://en.wikipedia.org/wiki/Jaccard_index

Po dosazení do vzorce dostaneme.

$$sim(Dneskajehezky., Dnesbylohezky.) = 1 - \frac{LD(Dneskajehezky., Dnesbylohezky.)}{max(|Dneskajehezky.|, |Dnesbylohezky.|)}$$

Pokud spočítáme, dostáváme se k výsledku.

$$1 - \frac{5}{max(16, 16)} = 0.6875$$

■

3.4.6 Winnowing

Winnowing algoritmus patří do skupiny, která porovnává dva řetězce na principu vyhledávání společných podřetězců. Podřetězce jsou převedeny do hash podoby. Winnowing pracuje s otisky K-gramů, které reprezentují původní dokument. Rozděluje dokument na podřetězce, které jsou převedeny do hash podoby. Podřetězec musí splňovat dvě vlastnosti, aby mohl být vybrán.

1. Detekuje společné podřetězec, pokud je délka podřetězce alespoň stejně velká jako hodnota **t**.
2. Nedetekuje společné podřetězce, které jsou kratší, než hodnota **k**.

Hodnotu **t** a **k** přiřazuje uživatel. Podmínkou je, že vždy platí.

$$k \leq t$$

Vyhýbáme se vybírání společných podřetězců, u kterých nejsou splněné podmínky uvedené v předchozích bodech. Čím vyšší je **k**, tím jistější si můžeme být, že shoda mezi dokumenty není náhodou. Avšak čím vyšší hodnota **k**, tím menší šance u předem upraveného dokumentu k odhalení. Je tedy nutné vybrat takovou délku „k“ abychom se tomuto problému vyhnuli. V krátkosti si popíšeme jak tento algoritmus funguje:

1. Načtení řetězce obsahujícího text.
2. Provést předzpracování, může se jednat o odstranění nadbytečných mezer, převedení všech znaků v řetězci na malé a podobně.
3. Převedení obsahu řetězce na k-gramy.
4. Převedení obsahu řetězce do hash podoby popsané v dokumentu [10].
5. Uspořádání hash hodnot do úseků délky w.

   $$w = t - k + 1$$

   Za úsek se považují okna „windows“.

6. Vybírání otisků - v každém okně se vybere minimální hodnota hashe. Pokud existuje více jak jeden hash s minimální hodnotou, vybere se hodnota nacházející se v poli nejvíce vpravo. Následně se uloží hodnoty všech hashů jako otisk daného dokumentu.

Informace o tomto algoritmu jsem čerpal z dokumentu Local Algorithms for Document Fingerprinting [10]. Nachází se zde detailnější popis algoritmu.

3.5 Vizualizace výsledků

Vizualizace slouží k zobrazení výsledků např. po testování podobnosti mezi soubory. Výsledky zobrazuje tak, aby je bylo jednoduché pochopit. Je nutné odlišit případy, kde jsou si porovnávané soubory mezi sebou podobné s jinými od těch, které podobné nejsou. Na ty, které jsou si výrazně podobné se následně zaměří osoba, která kontrolu nad danými soubory prováděla. Daná osoba rozhodne, zda se jedná, anebo nejedná o plagiát. Máme na vybranou, jak výsledek kontrolorovi reprezentovat. Pro představu jsem v seznamu vypsal dva možné způsoby:

- Zobrazit výsledky v seznamu.
- Zobrazit výsledky pomocí histogramu.

3.5.1 Zobrazení výsledků v seznamu

Index	Název souboru	Datum poslední úpravy	Podobnost %	Podrobnosti
0	Soubor1	1.1.2001	0...100	Zobrazit
1	Soubor2	1.2.2001	0...100	Zobrazit

Tabulka 3: Vizualizace výsledků v seznamu

Zobrazení může být provedeno způsobem uvedeným v tabulce 3. Jedná se o metodu zobrazení v seznamu. První kolonka by se nazývala Index, která nám bude zobrazovat, v jakém pořadí byly soubory nahrány. Druhá nám zobrazuje název souboru nebo projektu. Třetí nám ukazuje, kdy byl daný soubor naposledy upraven. Další sloupec se jmenuje podobnost. U podobnosti si můžeme určit, jak by se hodnotily jednotlivé případy podobnosti. Každá skupina by měla přiřazenou barvu, aby byla na první pohled lehce odlišitelná od ostatních.

- 0-25% (šedá barva)

  Kód se nepodobá porovnávanému vzorku.

- 25-45% (zelená barva)

  Kód se začíná podobat porovnávanému vzorku, ale stále se nemusí jednat o plagiát.

- 45-80% (oranžová barva)

  Kód se už stává podezřelý, je nutné prozkoumat jeho obsah.

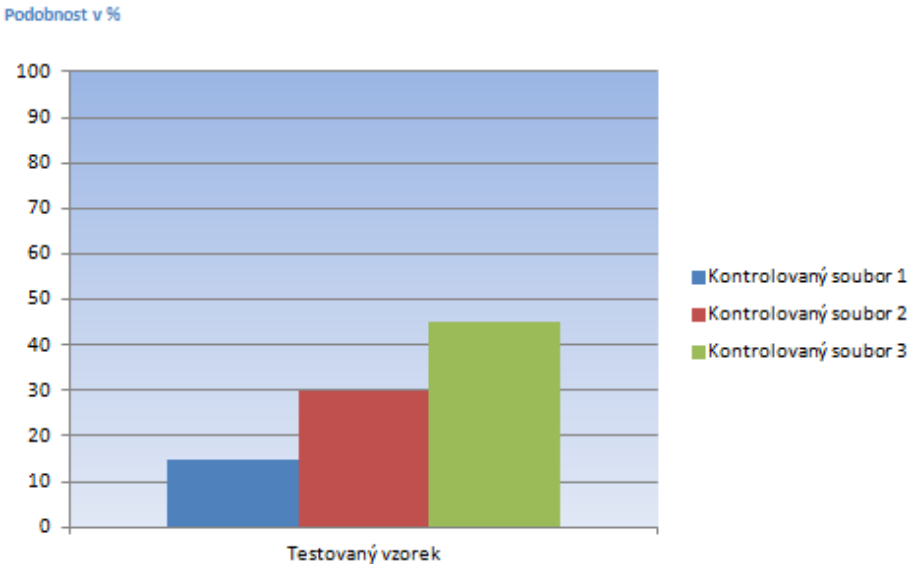
- 80-100% (červená barva)

  V posledním případě, pokud nestala chyba pří porovnáváni, jedná se o plagiát.

V posledním sloupci se nachází Podrobnosti. Po kliknutí na "Zobrazit"se otevře možnost zobrazení podrobného výpisu. Mohlo by se jednat o zobrazení, kdy na levé straně by byl originál a na pravé straně kontrolovaný soubor. Shodné části kódu by byly vyznačeny. Výsledky daných souborů by se na základě % shody seřadily vzestupně. Je nutné podotknout, že pokud program zobrazí, že se soubor podobá jinému, je na kontrolorovi rozhodnout, zda se jedná o plagiát.

3.5.2 Zobrazení výsledků pomocí histogramu

Dalším příkladem vizualizace je histogram, kde výška sloupců znázorňuje procentuální shodu, maximální hodnota značí sto procent a počet sloupců určuje počet testovaných souborů. Popřípadě můžeme použít obyčejný XY graf, který by měl na ose X soubory a na ose Y míru podobnosti s originálem. Pro pochopení přikládám obrázek 3, jak by histogram měl vypadat v případě použití u vizualizace výsledků porovnávaní.



Obrázek 3: Ukázka Histogramu

3.6 Existující nástroje

V této podkapitole se zaměřím na známé nástroje, které dokáží v zdrojových kódech zjistit, zda se jedná o plagiát. Snažil jsem se stručně popsat a ujasnit, jak nástroje kód zpracují, jaké detekční metody používají, kdy vznikly a které programovací jazyky podporují.

- **JPlag**

  Jplag[2] program byl vytvořen jako studentský projekt už roku 1996. Autorem byl G. Malphol. Podporuje jazyky Java, C, C++, C#. Zdrojový kód se na začátku tokenizuje. Následně se pomocí upravené metody GST provádí porovnávání dvou řetězců.

- **MOSS**

  MOSS[3] je ve vývoji už od roku 1994. Autorem je Alex Aiken. Podporuje širokou škálu programovacích jazyků. Pro představu zmíním C, C++, Java, Javascript, Visual basic, C#, Pascal, Python, Perl, atd. Pro porovnávání podobnosti kódu se používá winnowing.

- **Sherlock**

  Sherlock vytvořen roku 1994. Autor programu byl původně Rob Pike. Program se skládal ze dvou částí **sig** a **comp**. Sig se staral o vygenerování signatury a ukládal je do souboru. Comp se staral o následné porovnávání signatur mezi soubory a hlásil, zobrazoval míru podobnosti. Po spojení dvou nástrojů vznikl Sherlock[4]. Program se při vzniku hlavně zaměřoval na porovnávání Java kódů a normálního textu.

- **SIDPlag**

  SIDPlag[5] podporuje programovací jazyky C++ a Java. Zdrojový kód se na začátku, před porovnáváním tokenizuje. Pro porovnávání používá algoritmus Greedy String Tiling.

- **YAP**

  YAP[6] převádí zdrojový kód na posloupnost tokenů. Pro porovnávání používá RKR-GST. Obsahuje tokenizátor pro programovací jazyky C, Pascal a LISP. První zmínky o programu jsou z roku 1992.

Na závěr bych zmínil, že výsledné rozhodnutí zda se jedná o plagiát, záleží na osobě, která kontrolu prováděla. Je nutné si uvědomit, že při zadaném tématu, které je pro velké množství studentů stejný, bude míra podobnosti vyšší, než kdyby si mohl téma každý zvolit sám.

[2] `https://jplag.ipd.kit.edu/`
[3] `http://www3.nd.edu/~kwb/nsf-ufe/1110.pdf`
[4] `http://sydney.edu.au/engineering/it/~scilect/sherlock/`
[5] `http://www2.fiit.stuba.sk/~chuda/plagiarism/SIDPlag.html`
[6] `http://luggage.bcs.uwa.edu.au/~michaelw/YAP.html`

4 Realizace aplikace

Cílem aplikace je předzpracovat zdrojové kódy do tokenizované a n-gram podoby. Aplikace na vstupu dostane zdrojovové kódy, které má za úkol předzpracovat a výstupem bude tokenizovaný dokument. Na tento dokument pak bude moci být použita jedna z možných metod uvedených v kapitole 3.3.1 pro detekci plagiátu. Při realizaci aplikace jsem se zaměřil i na možnost otestovat tokenizovaný nebo předzpracovaný zdrojový kód mezi dalšími kódy načtenými v aplikaci. Je tedy možné otestovat podobnost zdrojových kódů a zjistit míru podobnosti mezi porovnávanými kódy.

4.1 Analýza

Aplikace bude mít možnost načítat soubory po jednom, nebo načíst obsah adresáře a v poslední řadě načíst projekt. Při načítání jednotlivých souborů se provedou základní úpravy a čištění kódů před uložením do paměti programu. Tento proces se nazývá předzpracování. Následně se provede tokenizace a výstupem bude tokenizovaný dokument.

4.2 Návrh

Aplikace má za úkol předzpracovat zdrojové kódy. Upravit kód nejlépe tak, aby byl odolný vůči základním plagiátorským praktikám. Výstupem bude tokenizovaný dokument pro možnost dalšího využití k odhalení plagiátů. Možností navíc bude porovnání tokenizovaných zdrojových nebo předzpracovaných kódů s jinými, které budou načteny v aplikaci. Pro porovnávání kódů mezi sebou je možno porovnávat soubor každý s každým nebo vybrat a potvrdit vzorový soubor, podle kterého se bude porovnávat. V prvním kroku, už při načítání souborů, se daný kód předzpracuje než se uloží do paměti programu. Mezi tyto úpravy patří:

1. Odstranění komentářů.
2. Odstranění prázdných řádků a mezer, ke kterým patří i tabulátorem přidané mezery.
3. Odstranění z kódu řádek importujících různé knihovny, jedná se o using.
4. Odstranění řádků začínající znakem #, který naznačuje, že se zde vyskytuje pragma.
5. Nahrazení řetězců za znak ".
6. Nahrazení znaků za jeden znak '.
7. Úprava čísel na defaultní hodnotu 0.
8. Převedení všech znaků na malé.

V druhém kroku bude na výběr mezi výpisem tokenizovaného kódu nebo porovnání kódu. Při výběru výpisu zdrojového kódu v tokenizované podobě bude kód upraven a výstup uložen do nově vytvořené složky v adresáři s aplikací. Následně se zobrazí umístění složky. Pokud se vybere možnost porovnávat kód, provede se porovnávání a výsledkem bude zobrazení v seznamu. Pro porovnávání kódů bude na výběr podobnost n-gramů nebo Levenshteinovy vzdálenosti. U n-gram podobnosti se kontrola bude provádět na předzpracovaném zdrojovém kódu. U Levenshteinovy vzdálenosti bude na výběr mezi předzpracovaným zdrojovým kódem nebo tokenizovaným kódem. V obou případech se ukládá na disk do složky soubor s požadovaným výstupem, pokud bude vyžadováno.

4.3 Implementace

Program je implementován pro programovací jazyk C# a předzpracovává zdrojové kódy pouze v jazyku C#. Pokud by bylo třeba, nebyl by problém rozšířit aplikaci o další programovací jazyky. Na začátku se do aplikace načtou požadované zdrojové kódy, řeší se přes metodu AddAndPrepareFile, které volají metody LoadFromFile, LoadFolder, HandleLoadingProject

- AddAndPrepareFile - přidá do paměti programu požadovaný zdrojový kód, součástí je i jeho předzpracování.
- LoadFromFile - pro načítání jednotlivých souborů. Volá na konci metodu AddAndPrepareFile, které předá informace o souboru.
- LoadFolder - zjišťuje obsah složky a předává metodě AddAndPrepareFile přijatelné soubory, které může načíst.
- HandleLoadingProject - stará se o sjednocení vícero souborů do jednoho, zpracovává .sln soubor a na konci předává informaci o výsledném souboru.

Data se ukládají do paměti programu formou seznamu. Obsah s informacemi a daty obstarává třída StoredFiles. Ukládá vstupní, předzpracovaný kód, dále obsahuje kód převedený do n-gramů a tokenizovaný kód. O předzpracování se starají třídy PreProcess a HandleNumbers.

- PreProcess - stará se o hlavní část předzpracování, které bylo zmíněno v kapitole 4.2.
- HandleNumbers - převádí veškerá validní čísla ve zdrojovém kódu na základní formát a tím je nula. Vrací následně takto upravený kód zpět.

Dále aplikace se skládá z tříd, které se starají o tokenizaci, n-gramy a Levenshteinovy vzdálenosti. O tokenizaci se stará třída Token a Tokenizer. Hlavní části třídy Token jsou jednotlivé typy tokenů, s kterými pracuje Tokenizer a taky struktura tokenu.
Třída Tokenizer zpracovává na vstupu zdrojový kód a výstupem je jeho tokenizovaná podoba.

Třída N-gram obsahuje metody pro vygenerování n-gramů pro příslušný zdrojový kód a kromě toho výpočet podobnosti mezi dvěma soubory. Pro výpočet je použit Diceova míra zmíněná v podkapitole 3.4.1.
Třída Levenshtein obsahuje metody pro vypočítání Levenshteinovy vzdálenosti mezi dvěma soubory zmíněné v podkapitole 3.4.5 a možnost vypočítaní následné podobnosti pomocí Diceovy míry.

Třída Colors vrací zpět barvu.

1. 0-10% = šedá
2. 11-34% = zelená
3. 35-74% = oranžová
4. 75-100% = červená

Na základě hodnoty, která vyšla při porovnávání, se vybere ta hodnota, která splňuje podmínku z seznamu a vrátí zpět příslušnou barvu. V seznamu se následně barevně rozliší jednotlivé případy a osoba, která kontroluje tyto soubory, bude mít snadnější práci.

Třída MyException obsahuje chybové zprávy nebo výjimky, na které reagujeme chybovými zprávami.

Třída ChartControl má na starost zobrazovat výsledek. Stará se o vizualizaci výsledků v režimu jedna ku jedné.

Třída ShowDiff obsahuje metodu, která má na starost zvýraznění kódů, které si jsou podobné. Tato funkce funguje pouze u porovnávání n-gramů mezi dvěma soubory.

4.4 Ovládání

4.4.1 První spuštění

Je nutné mít nainstalovaný .NET Framework 4.5 nebo vyšší. Prvním krokem je zkompilovat kód a mít spustitelný soubor SCPLAG.exe. Pro zkompilování kódu je potřeba mít nainstalované Visual Studio 2013. Potom otevřít soubor SCPLAG.sln, který je uvnitř složky s projektem a nechat sestavit spustitelný soubor. Následně se nechá spustit jako správce a pokračovat se bude dále viz. 4.4.2

4.4.2 Seznámení s aplikací

Po spuštění aplikace, se nacházíme v menu, které se nazývá „Načti“. Mezi dalšími lze vidět „Zpracovat“, „Zobrazit“ a „Nastavení“. Pro ovládání aplikace se stačí seznámit se čtyřmi hlavními částmi, z kterých se aplikace skládá.

Načíst - zde je na výběr jedna ze tří možností.

1. Soubor - načíst jednotlivé soubory.
2. Složka - načíst obsah složky. Podporovaný formát je s koncovkou cs.
3. Projekt - načte projekt. Vybere se projekt s koncovkou sln.

Po načtení všech potřebných souborů pokračujeme do záložky „Zpracovat“.

Zpracovat - na výběr je možnost si nechat vypsat výstup kódu nebo provést kontrolu pomoci n-gram podobnosti nebo Levenshteinovy vzdálenosti. Po ukončení kontroly je uživateli zobrazena hláška, že akce byla dokončena. Dále se zde zobrazí podobnost mezi soubory v procentech, při kliknutí následně na „Náhled“ se v seznamu aplikace přepne do menu „Zobrazit“.

Zobrazit - na levé straně je kód, s kterým se porovnává, a na pravé straně porovnávaný kód. Jsou zobrazené v jednom okně pro přehlednost.

Nastavení - v okně máme možnost na výběr z těchto možných akcí.

- Možnost označit konkrétní načtený soubor za vzorový.
- Omezit načítaní souboru, nastavit minimální v kB a maximální velikost v MB souborů, které se budou moci načítat do paměti programu.
- Omezit počet souborů, které program dovolí načíst.
- Možnost, na kolik desetinných míst si nechat zaokrouhlit výsledek.
- Vyčistit paměť, resetovat program.

4.5 Systémové požadavky

Systémové požadavky nutné pro bezproblémový běh programu.

1. .NET Framework 4.5.
2. OS Windows XP, 7, 8, 8.1 a x64.
3. 4GB RAM a více za předpokladu testování většího množství dat.

4.6 Třídní diagram

Třídní diagram je zobrazený na obrázku 4 v příloze. V této podkapitole si popíšeme jednotlivé třídy, které se nacházejí v aplikaci.

MainMenu - hlavní třída slouží především ke komunikaci mezi formulářem a jednotlivými objekty nacházejícím se v něm. Stará se navíc o výpis výstupních souborů.

ChartControl - třída, starající se o zobrazení výsledku po porovnávání v Chart-u. Obsahuje metodu:

1. UpdateChart - aktualizuje hodnoty, provede se opětovné vykreslení a vypsání nových hodnot.

PreProcess - třída, starající se o základní úpravy kódu, která je popsaná zde 4.2.
Obsahuje metody:

1. RemoveComments - odstraňuje nalezené komentáře v kódu a to řádkové i blokové.
2. RemoveWhiteSpaces - odstraní mezery pokud se vyskytují 2 a více vedle sebe.
3. RemoveEmptyLines - řádky, které neobsahují žádný text se odstraňují.
4. RemoveTabSpaces - odstraní mezery, které se přidávají klávesou tab.
5. Start - spustí proces předzpracování.

HandleNumbers - třídá, starající se o všechny čísla obsažená ve zdrojovém kódě.
Obsahuje metody:

1. NormalizeNumbers - metoda, která upraví všechna nalezená čísla int na 0 a čísla s desetinou hodnotou na **0.0**.
2. ReplaceNumbers - metoda, která změní číslo na defaultní hodnotu. V tomto případě na 0.
3. GetNextChar - vrací zpět další znak. Pokud se jedná o mezeru, přeskočí se na další znak.
4. Output - obsahuje upravený kód, který po provedené úpravě čísel vrací zpět.

Colors - třída, starající se o vrácení příslušné barvy. Stará se o to, že na základě podobnosti, které jsem si určil, mi vrátí příslušnou barvu. Obsahuje jedinou metodu a ta se jmenuje GetColor, která vrací barvu na základě uvedené číselné hodnoty. Hranice, jakou barvu bude vracet, je pevně zadaná.

Token - třída, starající se především o strukturu tokenu pro tokenizátor. Obsahuje typy tokenů a metody. Obsahuje metody:

1. IsExceptionStatement - pokud je token typu Try, Catch vrací true. V opačném případě false.
2. IsSelectionStatement - pokud je token typu Condition, Switch, Case vrací true. V opačném případě false.

Tokenizer - třída, starající se o tokenizaci zdrojového kódu.
Obsahuje metody:

1. GetCodeBetween - vrací podřetězec podle počátečního a koncového znaku.
2. GetCurrentBlock - vrací aktuální typ bloku.
3. GetFunction - kontroluje zda splňuje podmínky metody, funkce a pokud ano vrací zpět metodu.
4. HasBlock - zjišťuje, jestli následující znak označuje začátek bloku. Vrací true nebo false.
5. MethodInProgress - vrací true nebo false podle toho, jestli se jedná o metodu, ale není ještě celá.
6. ReturnBlockName - vrací název bloku, ale tentokrát jako řetězec.
7. ReturnBodyBracket - vrací obsah podřetězce ohraničený mezi bloky.
8. ReturnFuncType - zjišťuje, jestli lze funkce blíže specifikovat a vrátí její konkrétní typ a nebo nechá základní.
9. ReturnToken - existují dva typy, první kontroluje, jestli se jedna o znak a druhý slovo, za které by šlo je nahradit tokenem.
10. ReturnStringToSpecificChar - vrací řetězec od aktuální pozice až po znak, který se zadá.
11. SpecialCaseNeedToClose - metoda, která uměle přidává uzavíraní bloků tam, kde chybí.
12. Start - metoda, která tokenizuje zdrojový kód a vrací jeho kolekci tokenů.
13. Tokenize - metoda vrací kolekci tokenů. Předpřipravuje tokenizaci. Skládá se z pěti částí.
    (a) Podmínka „else if“ se upraví na „if“
    (b) Upraví se zdrojový kód tak, aby se vyvaroval chybám při tokenizaci. Jedná se o úpravu před vkládáním umělých bloků.
    (c) Zavolá se metoda Start, která provede tokenizaci kódu.
    (d) Po tokenizaci se projede kolekce tokenů a pojmenují se konce bloků.
    (e) Projede se kolekce tokenů a při zjištění, že se jedná o funkci, se zkontroluje, jestli obsahuje parametry a vypíšou se.
14. TokenizeBlocks - metoda, která se zavolá po tokenizaci. Stará se o pojmenování konců bloků.

Levenshtein - třída, která má na starost Levenshteinovou vzdálenost a operace s ní spjaté. Obsahuje metody:

1. GetDistance - vrátí vzdálenost mezi dvěma porovnávanými soubory.
2. GetSimilarity - vrací podobnost, kterou získal ze vzdálenosti.
3. GetSimilarityInPct - převádí výslednou podobnost na procenta.

N-grams - třída, starající se o vygenerování n-gramů pro daný zdrojový kód a operace s nimi spjaté. Obsahuje metody:

1. GenerateNgrams - parametry pro délku, bigram, trigram. Vrací složený řetězec.
2. nGramFrequency - vrací slovník dvou porovnávaných souborů s počtem opakujících se výrazů.
3. nGramSimilarity - porovnává zdrojové kódy dvou souborů a vrací jejich podobnost.
4. nGramSimilarityInPct - převádí výslednou podobnost na procenta.

HandleLoadingProject - třída, starající se o načítání projektů do paměti programu. Obsahuje metody:

1. CanAddFile - metoda, která vrací true nebo false. Vrací true v případě, že se nenachází v seznamu.
2. CanLoadCSFile - metoda, která kontroluje, jestli jsou splněny podmínky pro načtení cs souboru. Metoda vrací true nebo false.
3. Clear - smaže obsah seznamu, který obsahuje aktuální seznam **cs** souborů.
4. GetMergedFile - vrací cestu k souboru, který byl v případě úspěšnosti vytvořen.
5. IsFileOK - obsahuje parametr, obsah souboru, který se bude kontrolovat. Vrací true nebo false.
6. LoadProject - je hlavní metodou v třídě. Sloučí jednotlivé **cs** soubory do jednoho hlavního souboru. Vrací zpět cestu k souboru.
7. ParseCSProjFile - metoda, která čte informace ze souboru **cs**.
8. ParseSlnFile - metoda, která čte informace ze souboru **sln**.
9. SortByLength - vrací seřazený obsah seznamu. Na výběr možnost vzestupně nebo sestupně.

5 Testování na vybrané kolekci dat

Cílem práce bylo předzpracování kódů a výstupem být tokenizovaný dokument. Aplikace kromě výstupu tokenizovaného kódu dokáže tento kód porovnat s ostatními kódy načtenými v aplikaci. V této kapitole se zaměříme na testování podobnosti. Při testování můžeme použít jednu ze tří metod pro porovnávání.

1. N-gram podobnost na předzpracovaném kódu.
2. Levenshteinova vzdálenost na předzpracovaném kódu.
3. Levenshteinova vzdálenost na tokenizovaném dokumentu (kódu).

Při testování vzdálenosti pomocí Levenshteinova algoritmu, kdy zdrojový kód v tokenizované podobě obsahoval více jak 2500 tokenů, porovnávací doba vzrostla z řádu sekund na několik minut. Při porovnávání velkého množství zdrojových kódů (projektů), se pak doba porovnávání značně protáhla. Z tohoto důvodu jsem do aplikace přidal možnost zobrazit na konci porovnávaný čas, který zobrazí jak dlouho porovnávání trvalo. V dalším případě se mi stalo, že při porovnávání projektů se mi zobrazila hláška „OutOfMemoryException“. Je tedy otázkou, jestli by pak nebylo lepší využít v budoucnu jinou metodu pro porovnávání. Tato hláška se zobrazila při porovnávání kódů, přičemž jeden z nich překročil hranici 9500 tokenů.

5.1 Testovací data

Mezi testovací data patří projekty studentů. Zobrazeny jsou v tabulce 4.

Název projektu	Počet
Člověče nezlob se	4
Pexeso	6
Piškvorky	5
Miny	6
Lodě	4

Tabulka 4: Počet testovaných zdrojových kódů

Kromě studentských projektů jsem do testování zařadil i dva soubory, které mě zaujaly tím, že obsahovaly praktiky plagiátorství. Byly to zdrojové kódy, které se nacházely v jednom adresáři s dalšími, tedy nebylo možné zjistit podrobnější informace. Mezi praktiky, které se zde vyskytovaly patří:

1. Přejmenování atributů, jmen metod, tříd a všeho dalšího, co bylo možné.
2. Přidávání komentářů.
3. Přidávání metod, které se nevyskytovaly v původním kódu.

Oba soubory se nacházejí v kolekci testovaných souborů. V tabulce 5 lze vidět, kdy byl soubor naposledy upraven, název souborů a jejich aktuální velikost v kB.

Název souboru	Datum poslední úpravy	velikost
MinesClass.cs	15.5.2011 17:23	29kB
MinyKod.cs	6.5.2010 15:36	28kB

Tabulka 5: Zobrazený detailnější popis dvou plagiátů

Po předzpracování a otestovaní jsem došel k výsledkům uvedeným v tabulce 6.

Testování souborů	Metoda porovnávání	Podobnost (%)
MinesClass vs. MinyKod	N-gram podobnost	54
MinesClass vs. MinyKod	Levenshteinova vzdálenost typ 1	76
MinesClass vs. MinyKod	Levenshteinova vzdálenost typ 2	96

Tabulka 6: Zobrazení výsledků po testování podobnosti

Jak lze vidět, základní metoda, která porovnává na principu n-gramů zobrazila míru podobnosti 54%, což už při této hodnotě začíná být podezřelé. Avšak pokud by se jednalo např. o projekty, které měly pevně zadané téma, je možné, že se stále jedná o false detekci. Levenshteinova vzdálenost pro předzpracovaný kód následně zobrazila 76% a pokud se použil tokenizovaný kód podobnost překročila hranici 90%. Zdrojové kódy jsou si tedy podobné.

Nyní se vrátíme k projektům. Projekty mají původní název a na konci připsané „projekt a číslo“ v kterém pořadí byly načteny a porovnávaný. Pro představu.

- Login1234 Projekt 1.
- Login4321 Projekt 2.

Při testování jsem následně použil zkrácený název č. 1, č. 2 až č. n. Shoda mezi jednotlivými projekty v kategorii (Člověče nezlob se, Piškvorky, Pexeso, Miny, Lodě) byla vesměs v rozmezí od 0 do 50%. Dva z nich však měli podobnost vyšši než 50%.

1. Projekt **Pexeso** - porovnání č. 3 vs. č. 1 metodou n-gram podobnosti se shoduje 62%. Výsledky jsou zobrazené v tabulce 9. Při porovnávání metodou Levenshteina, ale shoda vyšla 40%. V tomto případě se spíš jedná o chybu. Při kontrole zdrojového kódu jsem nedošel k závěru, že by si byly podobné.
2. Projekt **Miny** - porovnání č. 5 vs. č. 6 a č. 6 vs. č. 5 se shoduje 99-100%. Výsledky jsou zobrazené v tabulce 10. V tomto případě se jedná o kopii. Nebyly ani použity žádné plagiátorské techniky.

Pří testování jsem se zaměřil na n-gram podobnost na předzpracovaném kódu a Levenshteinovy vzdálenosti na tokenizovaném kódu.

Číslo projektu	Metoda porovnávání	Podobnost (%)
č. 1 vs. č. 2, 3, 4	N-gram podobnost	33 - 16 - 5
č. 2 vs. č. 1, 3, 4	N-gram podobnost	4 - 2 - 3
č. 3 vs. č. 1, 2, 4	N-gram podobnost	30 - 32 - 8
č. 4 vs. č. 1, 2, 3	N-gram podobnost	8 - 7 - 5
č. 1 vs. č. 2, 3, 4	Levenshteinova vzdálenost	17 - 38 - 47
č. 2 vs. č. 1, 3, 4	Levenshteinova vzdálenost	17 - 8 - 12
č. 3 vs. č. 1, 2, 4	Levenshteinova vzdálenost	38 - 8 - 46
č. 4 vs. č. 1, 2, 3	Levenshteinova vzdálenost	47 - 12 - 46

Tabulka 7: Výsledky testů projektů Člověče nezlob se

Název projektu	Metoda porovnávání	Podobnost (%)
č. 1 vs. č. 2, 3, 4, 5	N-gram podobnost	8 - 7 - 11 - 10
č. 2 vs. č. 1, 3, 4, 5	N-gram podobnost	13 - 12 - 11 - 13
č. 3 vs. č. 1, 2, 4, 5	N-gram podobnost	14 - 16 - 11 - 17
č. 4 vs. č. 1, 2, 3, 5	N-gram podobnost	7 - 5 - 4 - 4
č. 5 vs. č. 1, 2, 3, 4	N-gram podobnost	16 - 13 - 13 - 9
č. 1 vs. č. 2, 3, 4, 5	Levenshteinova vzdálenost	35 - 44 - 43 - 37
č. 2 vs. č. 1, 3, 4, 5	Levenshteinova vzdálenost	35 - 46 - 21 - 48
č. 3 vs. č. 1, 2, 4, 5	Levenshteinova vzdálenost	44 - 46 - 30 - 48
č. 4 vs. č. 1, 2, 3, 5	Levenshteinova vzdálenost	43 - 21 - 30 - 23
č. 5 vs. č. 1, 2, 3, 4	Levenshteinova vzdálenost	37 - 48 - 48 - 23

Tabulka 8: Výsledky testů projektů Piškvorky

Název projektu	Metoda porovnávání	Podobnost (%)
č. 1 vs. č. 2, 3, 4, 5, 6	N-gram podobnost	17 - 31 - 13 - 4 - 9
č. 2 vs. č. 1, 3, 4, 5, 6	N-gram podobnost	18 - 10 - 6 - 2 - 1
č. 3 vs. č. 1, 2, 4, 5, 6	N-gram podobnost	62 - 13 - 13 - 5 - 8
č. 4 vs. č. 1, 2, 3, 5, 6	N-gram podobnost	10 - 10 - 8 - 4 - 9
č. 5 vs. č. 1, 2, 3, 4, 6	N-gram podobnost	4 - 1 - 3 - 3 - 5
č. 6 vs. č. 1, 2, 3, 4, 5	N-gram podobnost	4 - 3 - 2 - 3 - 3
č. 1 vs. č. 2, 3, 4, 5, 6	Levenshteinova vzdálenost	45 - 40 - 43 - 25 - 12
č. 2 vs. č. 1, 3, 4, 5, 6	Levenshteinova vzdálenost	45 - 28 - 34 - 34 - 17
č. 3 vs. č. 1, 2, 4, 5, 6	Levenshteinova vzdálenost	40 - 28 - 46 - 12 - 6
č. 4 vs. č. 1, 2, 3, 5, 6	Levenshteinova vzdálenost	43 - 34 - 46 - 16 - 8
č. 5 vs. č. 1, 2, 3, 4, 6	Levenshteinova vzdálenost	25 - 34 - 12 - 16 - 36
č. 6 vs. č. 1, 2, 3, 4, 5	Levenshteinova vzdálenost	12 - 17 - 6 - 8 - 36

Tabulka 9: Výsledky testů projektů Pexeso

Název projektu	Metoda porovnávání	Podobnost (%)
č. 1 vs. č. 2, 3, 4, 5, 6	N-gram podobnost	16 - 21 - 32 - 15 - 15
č. 2 vs. č. 1, 3, 4, 5, 6	N-gram podobnost	33 - 21 - 27 - 24 - 24
č. 3 vs. č. 1, 2, 4, 5, 6	N-gram podobnost	14 - 9 - 20 - 12 - 12
č. 4 vs. č. 1, 2, 3, 5, 6	N-gram podobnost	14 - 10 - 13 - 8 - 8
č. 5 vs. č. 1, 2, 3, 4, 6	N-gram podobnost	11 - 12 - 11 - 14 - 100
č. 6 vs. č. 1, 2, 3, 4, 5	N-gram podobnost	11 - 12 - 11 - 14 - 100
č. 1 vs. č. 2, 3, 4, 5, 6	Levenshteinova vzdálenost	26 - 50 - 48 - 46 - 46
č. 2 vs. č. 1, 3, 4, 5, 6	Levenshteinova vzdálenost	26 - 34 - 19 - 39 - 39
č. 3 vs. č. 1, 2, 4, 5, 6	Levenshteinova vzdálenost	50 - 34 - 41 - 48 - 48
č. 4 vs. č. 1, 2, 3, 5, 6	Levenshteinova vzdálenost	48 - 19 - 41 - 35 - 35
č. 5 vs. č. 1, 2, 3, 4, 6	Levenshteinova vzdálenost	46 - 39 - 48 - 35 - 99
č. 6 vs. č. 1, 2, 3, 4, 5	Levenshteinova vzdálenost	46 - 39 - 48 - 35 - 99

Tabulka 10: Výsledky testů projektů Miny

Název projektu	Metoda porovnávání	Podobnost (%)
č. 1 vs. č. 2, 3, 4	N-gram podobnost	8 - 12 - 8
č. 2 vs. č. 1, 3, 4	N-gram podobnost	8 - 8 - 9
č. 3 vs. č. 1, 2, 4	N-gram podobnost	8 - 5 - 5
č. 4 vs. č. 1, 2, 3	N-gram podobnost	13 - 17 - 19
č. 1 vs. č. 2, 3, 4	Levenshteinova vzdálenost	44 - 46 - 45
č. 2 vs. č. 1, 3, 4	Levenshteinova vzdálenost	44 - 47 - 30
č. 3 vs. č. 1, 2, 4	Levenshteinova vzdálenost	46 - 47 - 34
č. 4 vs. č. 1, 2, 3	Levenshteinova vzdálenost	45 - 30 - 34

Tabulka 11: Výsledky testů projektů Lodě

6 Závěr

6.1 Zhodnocení

Cílem práce bylo vytvořit aplikaci, která by předzpracovala zdrojové kódy a výstupem byl tokenizovaný dokument. Aplikace je napsána v programovacím jazyku C# a dokáže zpracovávat zdrojové kódy v C#. Zpracovává jak jednotlivé soubory, má i možnost načíst projekt jako jeden soubor. Při testování jsem se setkal s možností, že projekt odkazoval i na soubory, které se zde nenacházely. V tomto případě se zobrazí chybová hláška. Výstupem aplikace je předzpracovaný n-gramový a nebo tokenizovaný dokument. Aplikace navíc dokáže pomocí Levenshteinovy vzdálenosti nebo podobnosti n-gramů porovnat zdrojové kódy mezi sebou. Výsledkem je tedy aplikace, která dokáže zdrojové kódy předzpracovat a výstup mít v podobě tokenizovaného dokumentu, ale dokáže i porovnat zdrojové kódy mezi sebou a zobrazit míru podobnosti. Na konec bych chtěl zmínit funkci, která dokáže nahradit volání metod jejím obsahem „tělem“. Aplikace v nastavení má možnost tuto funkci zapnout, ale tato funkce nebyla dostatečně otestována a tudíž se jí nedoporučuje používat. Nebyla z tohoto důvodu ani zmiňována v implementaci, ale do budoucna by se na tuto funkci mohlo zaměřit a dostatečně ji otestovat, doladit nedostatky a začít ji používat.

6.2 Rozšíření aplikace

Do budoucna by se mohla aplikace dále rozšířit o funkce, vylepšení uvedené v tomto seznamu:

- U předzpracování doladit možnost přepsání volaných metod jejím tělem. Zmínka je v závěru „Zhodnocení“.
- Ignorovat prázdné metody nebo metody, které nejsou volány a nepracovat s nimi.
- Přidat možnost ukládat už tokenizované dokumenty do databáze, kdy by se kontrolovaly zdrojové kódy už uložené v databázi.
- Přidat podporu i pro další programovací jazyky.
- Přidat další metody pro detekci plagiátů.

Jan Havlas

7 Reference

[1] Plagiátorství - Infogram, [online]. 18.4.2015, Dostupný z WWW: *http://www.infogram.cz/findInSection.do?sectionId=1115&categoryId=1168*

[2] Citační norma ČSN ISO 690:2011, [online]. 18.4.2015, Dostupný z WWW: *https://sites.google.com/site/novaiso690/*

[3] Kryptomnézie, [online]. 18.4.2015, Dostupný z WWW: *http://en.wikipedia.org/wiki/Cryptomnesia*

[4] Theses.cz, [online]. 18.4.2015, Dostupný z WWW: *http://theses.cz/*

[5] Parker and Hamblen, [online]. 30.4.2015, Dostupný z WWW: *http://luggage.bcs.uwa.edu.au/~michaelw/YAP.html*

[6] N-gram, [online]. 18.4.2015, Dostupný z WWW: *http://cs.wikipedia.org/wiki/N-gram*

[7] Salha Alzahrani, Naomie Salim, Ajith Abraham, *Understanding Plagiarism Linguistic Pattern, Textual Features and Detection Methods*, IEEE Transactions on Systems, Man, and Cybernetics, 2012.

[8] Michael J. Wise, *Running Karp-Rabin Matching and Greedy String Tiling*, Basser Department of Computer Science, University of Sydney, ISBN 0867586699, 1993.

[9] Levenshtein-Algorithm, [online]. 18.4.2015, Dostupný z WWW: *http://www.levenshtein.net/*

[10] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken *Winnowing: Local Algorithms for Document Fingerprinting*, Computer Science Division, UC Berkeley, 2003.

[11] Gestaltismus, [online]. 18.4.2015, Dostupný z WWW: *http://cs.wikipedia.org/wiki/Gestaltismus*

A Třídní diagram

Obrázek 4: Třídní diagram aplikace

B Adřesářová struktura přiloženého CD

1. Aplikace - adresář obsahující archív s názvem SCPLAG.zip. Archív obsahuje zdrojové kódy, soubory aplikace. Před začátkem používání aplikace je nutné sestavit spustitelný **exe** soubor viz postup v podkapitole 4.4.1.

2. Zdrojove kody - adresář obsahující zdrojové kódy, projekty studentů, se kterými byla aplikace testována.