VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

HLEDÁNÍ PLAGIÁTŮ V DYNAMICKÝCH PROGRAMOVACÍCH JAZYCÍCH

SEARCHING FOR PLAGIARISM IN DYNAMIC PROGRAMMING LANGUAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAKUB ŠKUNDA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

146482

Ústav: Ústav informačních systémů (UIFS)
Student: **Škunda Jakub**
Program: Informační technologie
Specializace: Informační technologie

Název: **Hledání plagiátů v dynamických programovacích jazycích**
Kategorie: Překladače
Akademický rok: 2022/23

Zadání:

1. Seznamte se s ANTLR a dalšími nástroji pro generování abstraktního syntaktického stromu (AST) pro daný zdrojový kód.
2. Identifikujte a popište různé plagiátorské techniky v různých programovacích jazycích a studujte možnost jejich detekce.
3. Podle pokynů vedoucího navrhněte nástroj na detekci podobnosti dvou zadaných projektů ve dvou vybraných dynamických programovacích jazycích. Uvažujte i možnost současného provádění detekcí na řadě projektů.
4. Navržený nástroj implementujte a uživatelsky testujte pro alespoň 2 jazyky (např. PHP a Python 3) a všechny identifikované plagiátorské techniky.
5. Implementovaný nástroj porovnejte s existujícími nástroji a nastiňte další možný vývoj nástroje.

Literatura:

- Ondřej Krpec: Rozpoznání plagiátů zdrojového kódu v jazyce PHP, bachelor thesis, Brno University of Technology, Faculty of Information Technology, 2015
- Terence Parr: *The Definitive ANTLR 4 Reference*. 2nd Edition, Pragmatic Bookshelf, 2013

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Křivka Zbyněk, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 18.10.2022

Abstrakt

Cielom tejto práce bolo vytvoriť nástroj na vyhľadávanie plagiátov v dynamických programovacích jazykoch. Tento nástroj bol následne testovaný na projektoch v predmete Pricípy programovacích jazykov a OOP na Fakulte informačných Technológií VUT v Brně.

Abstract

The goal of this work was to create a tool for searching for plagiarism in dynamic programming languages. This tool was subsequently tested on projects in the subject Principles of Programming Languages and OOP at the Faculty of Information Technologies BUT.

Klíčová slova

Plagiátorstvo, odhalovanie plagiátov, dynamické programovacie jazyky, PHP, Python

Keywords

Plagiarism, plagiarism detection, dynamic programming languages, PHP, Python

Citace

ŠKUNDA, Jakub. *Hledání plagiátů v dynamických programovacích jazycích.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Hledání plagiátů v dynamických programovacích jazycích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Škunda
8. května 2023

Poděkování

Rád by som sa poďakoval vedúcemu práce pánovi Ing. Zbyňkovi Křivkovi, Ph.D za odbornú pomoc a rady pri riešení tejto práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Plagiátorstvo predstavuje významný problém v akademickej sfére, ktorý si v posledných rokoch vynútil čoraz väčšiu pozornosť. Táto bakalárska práca sa zameriava na špecifický aspekt plagiátorstva v oblasti dynamických programovacích jazykov. Aj keď je odhalenie plagiátu veľmi náročný proces, je nesmierne dôležité zaoberať sa ním, aby sa zabezpečila integrita akademických prác a presadzovalo spravodlivé hodnotenie študentov.

Pre plné pochopenie tejto problematiky si v kapitole 2 postupne predstavíme tento problém, pozrieme sa na jednotlivé techniky plagíatorstva v zdrojovom kóde a uvedieme si pri nich pár príkladov. Zároveň sa pozrieme na v súčasnosti veľmi populárne AI a nebezpečenstvo spojené s ním, predstavíme si v súčasnosti najrozšírenejší model strojového učenia `GPT` a niektoré populárne nástroje, ktoré ho používajú.

V kapitole 3 sa pozrieme na už existujúce nástroje na vyhľadávanie plagiátov, fázy v ktorých pracujú a algoritmy, ktoré využívajú. Pri jednotlivých algoritmoch sa taktiež pozrieme na to, akú majú časovú zložitosť a predstavíme si ukážky ako pracujú.

V kapitole číslo 4 si predstavíme nástroje na tvorbu abstraktného syntaktického stromu a možnosti tokenizácie kódu v jazykoch ako sú Python či PHP.

V ďalšej kapitole sa pozrieme na návrh nástroja na detekciu plagiátov, bližšie sa pozrieme na jednotlivé fázy, v ktorých nástroj bude pracovať a rozoberieme si konkrétne možnosti implementácie.

V kapitole číslo 6 sa nachádza popis výslednej implementácie nástroja, teda pretavenie návrhu do výsledného zdrojového kódu. Predstavíme si štruktúru nástroja a popíšeme jednotlivé moduly, ktoré sa tu vyskytujú. Ďalej si v jednotlivých bodoch načrtneme možný vývoj nástroja do budúcna.

V kapitole číslo 7 si priblížime testovanie daného nástroja a výsledky jeho používania. Ďalej sa pozrieme bližšie na porovnanie nástroja s už existujúcimi nástrojmi na detekciu plagiátov.

Kapitola 2

Plagiátorstvo

Predtým, ako bude bližšie priblížená táto problematika, je nutné si najprv predstaviť, čo plagiátorstvo vlastne je.

2.1 Plagiátorstvo

Plagiátorstvo je definované ako forma krádeže intelektuálneho vlastníctva, ktorá sa prejavuje v tom, že niekto použije cudzie dielo, alebo jeho časť bez súhlasu autora, alebo bez uvedenia pôvodu [18]. Ide o porušenie autorských práv, ktoré je trestné a môže mať závažné dôsledky pre páchateľa. Plagiátorstvo sa môže vyskytovať v rôznych formách, ako napríklad v kopírovaní častí textu z iných zdrojov bez uvedenia zdroja, v použití obrázkov alebo grafov bez súhlasu autora, alebo v použití hudby alebo videa bez súhlasu autora. Plagiátorstvo je vážny problém, ktorý môže mať negatívny vplyv na reputáciu páchateľa a môže viesť aj k trestnoprávnym sankciám.

Medzi časté dôvody plagiátorstva patria:

- Nedostatočné schopnosti.
- Zlý časový manažment.
- Zlá schopnosť písať úlohy alebo odborné texty.
- Problém písania textov v cudzom jazyku.
- Nedostatočná citácia.
- Zlé vedenie.
- Zlá znalosť toho, čo predstavuje plagiátorstvo, alebo akademická integrita.
- Príliš veľký dôraz na propagáciu samého seba.

2.2 Plagiátorstvo v zdrojovom kóde

Plagiátorstvo v zdrojovom kóde je taktiež forma krádeže intelektuálneho vlastníctva a je nelegálne.

V tejto práci sa zameriam hlavne na túto problematiku na akademickej pôde, kde je tento problém veľmi aktuálny. Študentov, ktorí plagiátostvo podstúpia postihnú disciplinárne opatrenia, či dokonca vylúčenie zo školy.

Preto je potrebné, aby sa študenti, ktorí používajú cudzí zdrojový kód vo svojich prácach uistili, že majú súhlas od autora alebo vlastníka kódu a že ho uvedú ako zdroj vo svojej práci.

2.3 Techniky plagiátorstva v zdrojovom kóde

Keďže v tejto práci sa zameriavame na hľadanie plagiátorstva na akademickej pôde, konkrétne na Fakulte informačných Technológií VUT v Brně, sme obmedzení iba na práce na tejto fakulte. Porovnávanie jednotlivých zdrojových kódov teda prebieha iba medzi jednotlivými študentmi a nie s prácami mimo našej fakulty.

Existuje niekoľo techník, ktoré môžu byť použité pri plagiátorstve v zdrojovom kóde [15]:

- kontrola komentárov
- zmena odriadkovania, odsadenia, medzier
- zmena názvu premenných a funkcií
- zmena datových typov
- nahradenie konštrukcie inou, ekvivalentnou k nej
- pridávanie nadbytočných príkazov
- prehodenie príkazov

Komentáre

Keďže komentáre nemajú na beh programu žiadny vplyv, často sa stáva, že študenti ich pri plagiátorstve prehliadajú. Analýza komentárov preto môže byť použitá ako jasný dôkaz v prípade, že sa zdrojový kód označí ako plagiát. Napríklad ak je v dvoch zdrojových kódoch rovnaká pravopisná chyba, je takmer jasné, že sa jedná o plagiát.

Biele znaky

V zdrojovom kóde sa nachádza veľké množstvo bielych znakov, či už pri oddelovaní identifikátorov, či operátorov. Študenti si často môžu myslieť, že pridávanie medzier, alebo nových riadkov do kódu im pomôže zamaskovať plagiátorstvo. Z tohto dôvodu je vhodné všetky biele znaky zo zdrojového kódu odstrániť a nechať ich iba tam, kde je to nutné.

Názvy premenných

Zmena názvu premenných, či funkcií je taktiež veľmi rozšírená stratégia na zakrytie plagiátorstva. Na predídenie tomuto typu plagiátorstva je potrebné kód rozložiť na tokeny a porovnávať medzi sebou jednotlivé typy tokenov, nie iba názvy jednotlivých premenných.

Datové typy

V niektorých programovacích jazykoch, ktoré niesú dynamicky typované, je možné meniť datové typy. Napríklad číselných datových typov existuje niekoľko. Preto je dobré datové typy zhlukovať, tj. ak by bol použitý datový typ `unsigned int` a `long int`, bolo by ideálne prekonvertovať ich na spoločný datový typ.

Konštrukcie

V programovacích jazykoch je viac spôsobov ako zapísať rovnaký príkaz. Napríklad príkaz inkrementácie môžme zapísať ako `x = x + 1`, ale aj ako `x += 1`.

Pridávanie a prehadzovanie príkazov

Pridávanie či prehadzovanie príkazov je taktiež veľmi rozšírený spôsob plagiátorstva. Na jeho odhalenie existujú tzv. metriky, ktoré nám ukážu, ako moc sa jednotlivé kódy líšia.

2.4 Používanie AI

V dnešnej dobe sa používanie umelej inteligencie (AI) stáva čoraz populárnejším a rozšírenejším v rôznych oblastiach, vrátane programovania a vývoja softvéru. Z nedávneho vývoja v oblasti AI ťaží mnoho priemyselných odvetví. Aj keď mnoho z nich pozitívne ovplyvňuje našu spoločnosť, objavuje sa stále viac a viac prípadov, kedy bola táto technológia zneužitá. Ide hlavne o oblasť vzdelávania, kde vzniklo množstvo nástrojov, ktoré predstavujú vysoké riziko pre podporu plagiátorstva.

ChatGPT

V súčasnosti je najpoužívanejšia AI `ChatGPT`[1], od spoločnosti `OpenAI`. Aj keď bola táto AI spustená iba na konci novembra 2022, v januári 2023 už dosiahla 100 miliónov užívateľov a stala sa tak najrýchlejšie rastúcou webovou aplikáciou v histórii. Model, ktorý používa `ChatGPT` sa nazýva `GPT-3` a v dnešnej dobe je používaný už v mnohých aplikáciách[2]. Niektoré z týchto aplikácií sami seba prezentujú ako "AI nástroje pre písanie vysokoškolských prác". V článku o nebezpečenstve používania AI [22] autori spomínajú, že študenti dávajú AI najčastejšie úlohy typu:

- doplňovanie otázok
- parafrázovanie uvedeného textu

Prieskum [1] ukázal, že študenti sa často spýtajú AI otázku, ktorej odpoveď iba upravia a bez použitia akéhokolvek zdroja túto odpoveď prehlásia za svoju.

Autori práce [22] ďalej uvádzajú, že AI dokáže parafrázovať zadaný text s oveľa nižšou podobnosťou, ako človek. Zatial čo parafrázovanie od človeka označil nástroj na vyhľadávanie plagiátov ako pozitívnu zhodu, parafrázu od AI nie.

[1] https://openai.com/blog/chatgpt
[2] https://gpt3demo.com/map

Spoločnosť `OpenAI` aktuálne pracuje na modeli `GPT-4`, ktorý by mal byť podľa výzkumu [4] omnoho viac tvorivý pri riešení problémov ako model `GPT-3`. Tento model bol trénovaný na omnoho väčšom datasete, ktorý zahŕňal obrázky a text. Tento model je však rovnako ako predtým trenovaný na dátach iba do roku 2021.

Github Copilot

Medzi ďalšie rozšírené AI patrí `Github Copilot`[3]. Copilot bol vydaný v júni 2022 a je to nástroj na automatické dopĺňanie funkcií, ktorý takisto ako `ChatGPT` používa model `GPT-3`. V užívateľskom rozhraní, ktoré je poskytované je možné kliknúť na tlačítko "next", a tým zobraziť ďalšie riešenie, ktoré s riešením poskytnutým predtým nemá nič spoločné.

Príklady generovania riešenia nástrojom Github Copilot[22]:

```
def max_sum_slice(xs):
    if not xs:
        return 0

    current_max = 0
    max_so_far = 0

    for x in xs:
        current_max = max(0, current_max + x)
        max_so_far = max(max_so_far, current_max)

    return max_so_far
```

Výpis 2.1: Prvé automaticky generované riešenie

```
def max_sum_slice(xs):
    best_sum, best_start, best_end = 0, None, None

    for i, x in enumerate(xs):
        if best_start is None or best_sum < x + best_sum:
            best_sum = x + best_sum
            best_start = i
            best_end = i
        elif best_sum > x + best_sum:
            best_sum = x + best_sum
            best_end = i

    return best_sum, best_start, best_end
```

Výpis 2.2: Ďalšie automaticky generované riešenie

[3] https://copilot.github.com/

Ako je možné vidieť uvedené príklady majú dosť odlišnú štruktúru. Dokonca nástroj na detekciu plagiátov MOSS určil ich podobnosť ako veľmi nízku.

Trénovanie modelu GPT

Trénovanie modelu `GPT` (Generative Pre–trained Transformer) prebieha v dvoch fázach: predtrénovanie a ladenie.

Vo fáze predtrénovania je model vystavený obrovskému množstvu textu, aby sa naučil štruktúru a gramatiku jazyka. Na základe tohto učenia sa model naučí predpovedať, ktoré slovo bude nasledovať.

V druhej fáze sa model učí riešiť špecifické úlohy, pri ktorých sa upravujú jednotlivé váhy modelu, aby sa dosiahla lepšia účinnosť pre konkrétnu úlohu.

Pri riešení jednotlivých úloh model často replikuje obsah z trénovacích dát. Ak tieto dáta obsahujú chránené diela ako sú knihy, alebo iné publikácie, môže dôjsť k porušeniu autorských práv, ktoré si osoba ktorá tento model používa nemusí uvedomiť.

Avšak vďaka rýchlemu vývoju AI existuje aj množstvo nástrojov, ktoré sa zameriavajú na detekciu plagiátov v kóde. Tieto nástroje sú založené na algoritmoch strojového učenia, ktoré dokážu porovnávať a analyzovať kód, aby identifikovali potenciálne kopírovanie alebo zmeny, ktoré boli vykonané na kóde bez pripísania zdroja.

Existujú aj iné AI nástroje, ktoré sú používané na detekciu plagiátov v akademickom prostredí, ako napríklad Turnitin. Tieto nástroje sú schopné analyzovať textové dokumenty a porovnávať ich s existujúcimi zdrojmi, aby identifikovali podobnosti alebo plagiátorstvo.

Je však dôležité si uvedomiť, že AI nástroje na detekciu plagiátov nie sú dokonalé a môžu viesť k falošným pozitívam alebo negatívam. Preto je dôležité, aby ich používanie bolo kombinované s manuálnou kontrolou a zodpovednosťou zo strany používateľov.

Kapitola 3

Detekcia plagiátorstva v zdrojovom kóde

Existuje niekoľko nástrojov a algoritmov, ktoré sa používajú na detekciu plagiátorstva v zdrojovom kóde. Niektoré z nich si v nasledujúcich podkapitolách predstavíme.

3.1 Nástroje na vyhľadávanie plagiátov

V súčasnosti existuje niekoľko nástrojov, ktoré sú špeciálne navrhnuté na detekciu plagiátorstva v zdrojovom kóde. Tieto nástroje sú veľmi dôležité pre akademické inštitúcie, pretože študenti často kopírujú a modifikujú kódy bez správneho citovania zdrojov. V tejto sekcii si postupne predstavíme najznámejšie nástroje na detekciu plagiátov v zdrojovom kóde.

MOSS (Measure of Software Similarity)

MOSS je nástroj na automatické vyhľadávanie plagiátov v zdrojových kódoch programovacích jazykov. Bol vyvinutý v roku 1994 na Standford University a odvtedy sa stal široko používaným nástrojom pre detekciu plagiátov v akademickom prostredí. Tento nástroj je schopný rozoznať plagiáty v programovacích jazykoch, ako sú C, C++, Java, Python a mnoho ďalších [3].

MOSS používa techniku známu ako odtlačky dokumentov – Winnowing. Snímanie odtlačkov dokumentu je technika na detekciu kópií, ktorá sa vyhýba naivnému porovnávaniu podreťazcov. Namiesto toho je pre každý dokument vopred vypočítaná množina hash (odtlačok prsta) a namiesto toho sa porovnávajú odtlačky každého dokumentu, čo drasticky znižuje celkový počet porovnaní. Bližšie si techniku winnowingu predstavíme v kapitole 3.2.

JPlag (Java Programming Language Automatic Grader)

JPlag je nástroj na porovnávanie zdrojových kódov. Je vytvorený v jazyku Java. Hlavnou výhodou JPlagu je, že je open-source [16]. Bol vytvorený hlavne pre hľadanie podobností medzi projektmi na akademickej pôde. Rozoznáva plagiáty v jazykoch Java, C#, C, C++, Python 3, Go, Rust, Kotlin, Swift, Scala, Scheme a EMF. Taktiež vie rozoznávať plagiátorstvo v texte [14].

JPlag pracuje v dvoch fázach:

1. tokenizácia – všetky programy ktoré sú porovnávané sú zparsované a vytvoria sa z nich tokeny

2. tokeny sú porovnávané v pároch pre určenie podobnosti každého páru. Behom každého porovnávania sa JPlag snaží pokryť jeden token z tokenov v páre tokenmi z druhého páru. Následné percento tokenov, ktoré je možné prekryť je percento podobnosti.

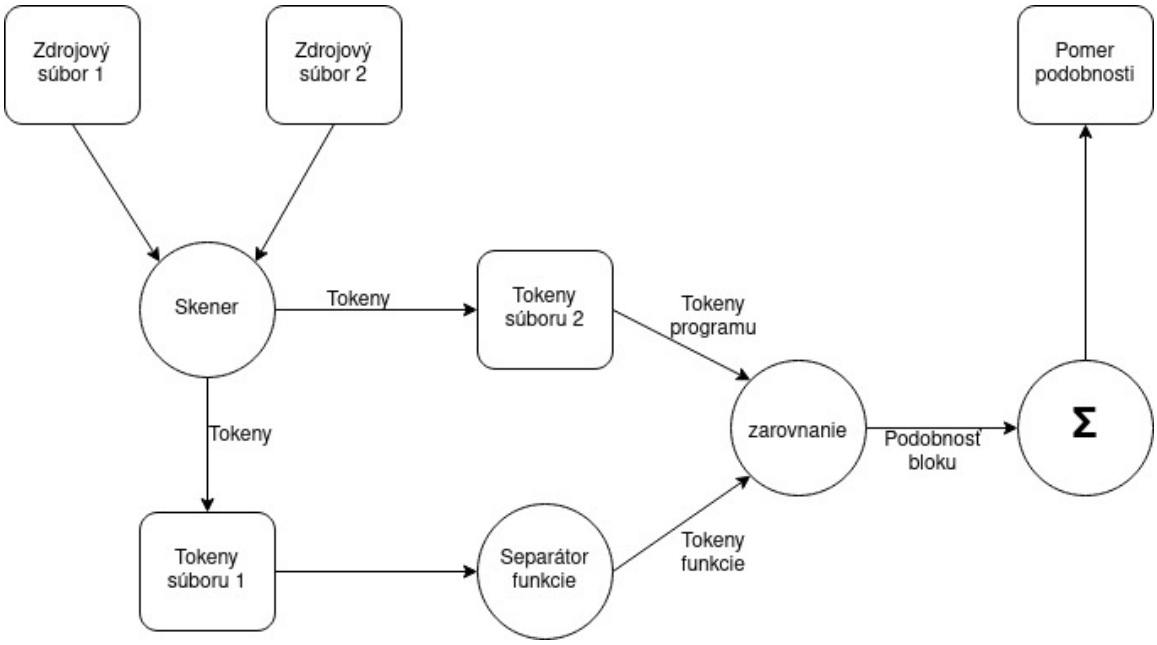
V druhej fáze sa používa algoritmus známy ako Greedy String Tiling. Tento algoritmus si bližšie predstavíme v kapitole 3.2.

SIM

SIM testuje lexikálnu podobnosť v prirodzenom jazyku, ako aj v programovacích jazykoch ako sú C, C++, Java, Pascal, Modula-2, LISP, ... Vyvoril ho v roku 1989 Dick Grune a je implementovaný v jazyku C [7].

SIM sa používa hlavne na:

- detekovanie duplicitného kódu v dlhších softwarových projektoch
- detekovanie plagiátov v softwarových projektoch a vedeckých prácach



Obrázek 3.1: Diagram funkčnosti detekoru SIM (prevzaté z [6])

SIM používa na určenie podobnosti jednotlivých súborov metriky, ktoré určujú podobnosť medzi jednotlivými reťazcami. Jedná sa o komplexný detektor, ktorý dokáže detekovať aj zmeny názvu premenných a funkcií, preusporiadanie príkazov a funkcií v kóde.

Sherlock

Sherlock je nástroj na vyhľadávanie plagiátov, ktorý je dostupný online a pomáha detekovať plagiáty v zdrojových kódoch a rôznych typoch dokumentov, ako napríklad PDF súbory a webové stránky [9]. Sherlock používa hlavne algoritmy na porovnávanie textu. Najprv jednotlivé súbory rozdelí na tokeny a potom vytvára hashované n-gramy. Následne sú niektoré z hashovaných hodnôt náhodne vyľúčené s cieľom vykonať rýchlejšiu a spoľahlivejšiu detekciu. Nakoniec Sherlock vypíše zoznam jednotlivých výsledkov obsahujúci percentuálnu podobu súborov.

Tento nástroj aktuálne podporuje väčšinu objektovo orientovaných jazykov, hľavne však Javu, pre ktorú boli vytvorené špecifické optimalizácie.

Sherlock ďalej umožňuje:

- hľadať plagiáty v rôznych jazykoch
- integráciu s rôznymi platformami, ako napríklad Google alebo Microsoft OneDrive
- možnosť porovnávať dokument so zdrojmi na internete

Plaggie

Plaggie je open sourcový nástroj na vyhľadávanie plagiátov. Plaggie má podobnú funkcionalitu ako JPlag, rovnako ako on používa algoritmus Greedy String Tiling, avšak je potrebné ho mať lokálne nainštalovaný. Plaggie podporuje iba jazyk Java [2].

Tento nástroj umožňuje napríklad:

- nastaviť percentuálnu hranicu pre detekciu plagiátu
- vytvárať vlastné databázy zdrojov

Turnitin

Turnitin je voľne dostpný vysoko uznávaný nástroj na kontrolu plagiátorstva, ktorý sa používa v akademickom prostredí. Tento nástroj umožňuje študentom, učiteľom a iným inštitúciám overiť autentickosť akademickej práce. Jeho hlavným účeľom je zabrániť plagiátorstvu a podporiť akademickú čestnosť [11].

Turnitin funguje na základe algoritmu, ktorý porovnáva text odovzdaného dokumentu s oborovskou databázou zdrojov, ktorá obsahuje knihy, časopisy, články webových stránok a iné. Výsledkom tohto porovnávania je podrobná správa o podobnosti, ktorá zobrazuje percentuálnu zhodu medzi overovaným textom a zdrojmi, z ktorých mohol byť text skopírovaný. Okrem detekcie priameho plagiátorstva dokáže Turnitin identifikovať aj takzvanú parafrázu, teda prepisovanie pôvodného textu s cieľom zakryť skutočný zdroj.

Je to jeden z mála nástrojov, ktorý využíva algoritmy založené na umelej inteligencii. Napriek všetkým týmto výhodám, má však aj niekoľko obmedzení. Ak sa zdroj nenachádza v databáze, Turnitin ho nieje schopný odhaliť. Turnitin taktiež neslúži na detekciu plagiátorstva v zdrojovom kóde.

3.2 Algoritmy na detekciu plagiátov

V priebehu rokov, boli vyvinuté rôzne algoritmy na detekciu plagiátov. Tieto algoritmy umožňujú porovnávanie pôvodnej práce so zdrojmi a výsledkom je percentuálna zhoda. Ak je táto zhoda vysoká, existuje podozrenie na plagiátorstvo a autor musí predložiť dodatočné dôkazy o tom, že práca je jeho vlastným dielom.

Existuje množstvo rôznych algoritmov na detekciu plagiátov, ktoré sa líšia svojimi metódami a úrovňou presnosti. Niektoré z týchto algoritmov používajú heuristické metódy na hľadanie podobností v texte, zatiaľ čo iné používajú strojové učenie na zlepšenie presnosti. V tejto bakalárskej práci sa budeme zameriavať na niektoré z najpoužívanejších algoritmov na detekciu plagiátov a porovnáme ich účinnosť a presnosť.

Winnowing

Winnowing je algoritmus na zistenie podobnosti medzi dvoma textami. Funguje tak, že rozdelí text na tzv. k-gramy, kde k je vopred stanovené číslo. Pre každý k-gram následne vytvorí hashovaciu funkciu. Hashovacie funkcie sa používajú na zníženie velkosti dát a určenie podobnosti medzi nimi. Nakoniec sa porovnávajú hodnoty hashovacích funkcií k-gramov v oboch textoch a ak sa zhodujú znamená to, že texty sú podobné [17].

Použitie winnowingu na detekciu plagiátov je efektívnejšie ako porovnávanie celých textov, pretože sa porovnávajú iba k-gramy.

Winnowing má však aj niekoľko obmedzení. Napríklad ak sú k-gramy v oboch textoch zoradené inak, algoritmus ich nemusí detekovať. Tiež môže byť náchylný na falošné pozitívne výsledky, keďže k-gramy môžu byť podobné aj náhodou. Napriek týmto obmedzeniam winnowing zostáva efektívnym nástrojom na detekciu plagiátov.

Príklad winnowingu

- Vstupný text:

  ```
  A do run run run, a do run run
  ```

- Odstránenie nepdstatných častí:

  ```
  adorunrunrunadorunrun
  ```

- Sekvencia 5-gramov odvodená z textu:

  ```
  adoru dorun orunr runru unrun nrunr runru unrun nruna runad unado nador
  adoru dorun orunr runru unrun
  ```

- Hypotetická sekvencia hashov 5-gramov:

  ```
  77 74 42 17 98 50 17 98 8 88 67 39 77 74 42 17 98
  ```

- Okná hashov o velkosti 4:

  ```
  (77, 74, 42, 17) (74, 42, 17, 98) (42, 17, 98, 50) (17, 98, 50, 17)
  (98, 50, 17, 98) (50, 17, 98, 8) (17, 98, 8, 88) (98, 8, 88, 67)
  (8, 88, 67, 39) (88, 67, 39, 77) (67, 39, 77, 74) (39, 77, 74, 42)
  (77, 74, 42, 17) (74, 42, 17, 98)
  ```

- Odtlačky vybraté winnowingom:

  `17, 17, 8, 39, 17`

- Odltačký spárované s 0-base pozičnou informáciou

  `[17,3] [17,6] [8,8] [39,11] [17,15]`

Algoritmus winnowing má časovú zložitosť $O(nw)$, kde n je dĺžka vstupu a w je šírka okna, ktoré sa používa pri vytváraní hashu z bloku textu. Najhoršia možná časová zložitosť algoritmu winnowing však môže byť až $O(n^2)$.

Greedy String Tiling

Greedy String Tiling je algoritmus, ktorý sa používa na porovnávanie dvoch textových reťazcov a hľadanie najdlhších opakujúcich sa podreťazcov v nich. Tento algoritmus pracuje tak, že prechádza oba reťazce znak po znaku a hladá najdlhší možný podreťazec, ktorý sa opakuje v oboch reťazcoch. Ak takýto podreťazec nájde, označí ho a pokračuje v hľadaní ďalších opakujúcich sa podreťazcov od miesta, kde skončil. Tento proces pokračuje dovtedy, dokým nieje celý reťazec prehľadaný.

Príklad algoritmu Greedy String Tiling:

Dané sú vstupné reťazce P a T:

- P = c a a b a a d
- T = b a a d c a a a a b a a

Algoritmus vyberie podreťazec `a a b a a` ako zhodu P a T radšej ako dva podreťazce `c a a` a `b a a d`, pretože prvý spomenutý má dĺžku 5 a zvyšné dva dĺžku 3 a 4.

Spomenutý algoritmus má prinajhoršom časovú zložitosť $O(n^3)$ [21]. Upravená verzia tohto algoritmu – Running-Karp-Rabin Greedy String Tiling, ktorá sa snaží zlepšiť rýchlosť pri porovnávaní dlhších reťazcov má taktiež rovnakú časovú zložitosť ako pôvodná verzia.

Levenshteinova vzdialenosť

Levenshteinova vzdialenosť je metrika na meranie podobnosti medzi dvoma reťazcami alebo slovami. Bola vymyslená v roku 1965 Vladimirom Levenshteinom.

Levenshteinova vzdialenosť medzi dvoma reťazcami je daná minimálnym počtom operácií potrebných k transformácii jedného reťazca na druhý. Operáciami sa v tomto prípade rozumie nahradenie, vloženie alebo zmazanie jedného znaku. Túto techniku je možné uplatniť na dva zdrojové kódy bez akejkolvek predchádzajúcej analýzy.

Ako príklad si môžme uviesť slová `kitten` a `sitting`. Ak v slove `sitting` nahradíme prvé písmeno písmenom `k`, písmeno `i` písmenom `e` a odstránime posledné písmeno dostaneme dva rovnaké reťazce. Tým pádom je Levenshteinova vzdialenosť 3.

Časová zložitosť tohto algoritmu je $O(m*n)$, kde m a n sú dĺžky porovnávaných reťazcov.

Halsteadove metriky

Halsteadove metriky sú súbor metrík, ktoré boli navrhnuté v roku 1977 americkým vedcom Mauriceom H. Halsteadom na meranie zložitosti a kvality softvéru. Halsteadove metriky sú založené na analýze zdrojového kódu programu, kde sa zohľadňujú dva základné atribúty programu: operátory a operandy. Operátory predstavujú akcie, ktoré sa vykonávajú v programe a operandy sú prvky, na ktorých sa jednotlivé akcie vykonávajú.

Majme štyri premenné:

- n_1 : počet jedinečných operátorov
- n_2 : počet jedinečných operandov
- N_1 : celkový počet operátorov
- N_2 : celkový počet operandov

Z nich vieme vytvoriť jednotlivé metriky:

- programový slovník : $(n) = n_1 + n_2$
- programová dĺžka : $(N) = N_1 + N_2$
- predpokladaná programová dĺžka : $(N^) = n_1 * log_2(n_1) + n_2 * log_2(n_2)$
- celková zložitosť : $(E) = N_1 * log_2(n_1) * N_2 * log_2(n_2)$
- programový objem : $(V) = N * log_2(n)$
- programové úsilie : $(D) = E/V$
- programová rychlosť : $(L) = V/E$

Výhodou týchto metrík je, že na základe jednotlivých výsledkov dokážu zohľadniť podobnosť zdrojových kódov bez hlbšieho porovnávania.

Príklad Halsteadových metrík:

Majme jednoduchý kód v jazyku Python [8]:

```
num = int(input("Enter a number: "))
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp /= 10
if num == sum:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")
```

Jednotlivé metriky pre tento kód budú nasledovné: $N_1 = 30$, $N_2 = 19$, $n_1 = 17$, $n_2 = 7$, $N = 49$, $n = 24$, $V = 224,663$, $E = 1729$, $D = 7.8721$, $L = 0.127$.

Tokenizácia

Tokenizácia je proces rozdelenia textu na jednotlivé slová alebo frázy, ktoré sa nazývajú tokeny. Cieľom tokenizácie je získať jednotlivé slová alebo frázy z textu, aby sa dali ďalej spracovávať.

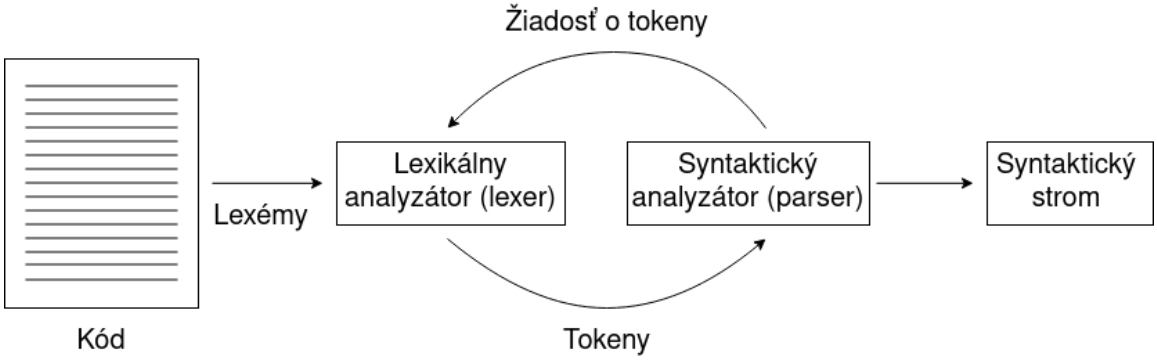
Existuje niekoľko spôsobov tokenizácie. Pre účely jazykového modelovania sa často používa tzv. medzerová tokoenizácia, kde sa text rozdelí na jednotlivé tokeny podľa medzier medzi nimi. Tokenizácia je dôležitý krok v procese prípravy dát.

Tokenizácia pri zdrojovom kóde znamená proces rozdelenia zdrojového kódu na jednotlivé časti. Tieto tokeny majú svoj typ, hodnotu, poprípade rôzne iné parametre. Príkladom typu tokenov môžu byť kľúčové slová – napríklad if, else, identifikátory, operátory a ďalšie elementy zdrojového kódu.

Kapitola 4

Nástroje na tvorbu abstraktného syntaktického stromu

Vytvorenie abstraktného syntaktického stromu zo zdrojového kódu je veľmi užitočné, pretože nám umožňuje ignorovať formátovanie kódu a sústrediť sa len na dôležité syntaktické prvky. Rovnako môžeme obmedziť prítomnosť niektorých tokenov, ako sú napríklad medzery alebo tabulátory.



Obrázek 4.1: Proces generovania syntaktického stromu

V tejto kapitole sa pozrieme na niektoré nástroje na tvorbu abstraktného syntaktického stromu a tokenizáciu zdrojového kódu.

4.1 ANTLR

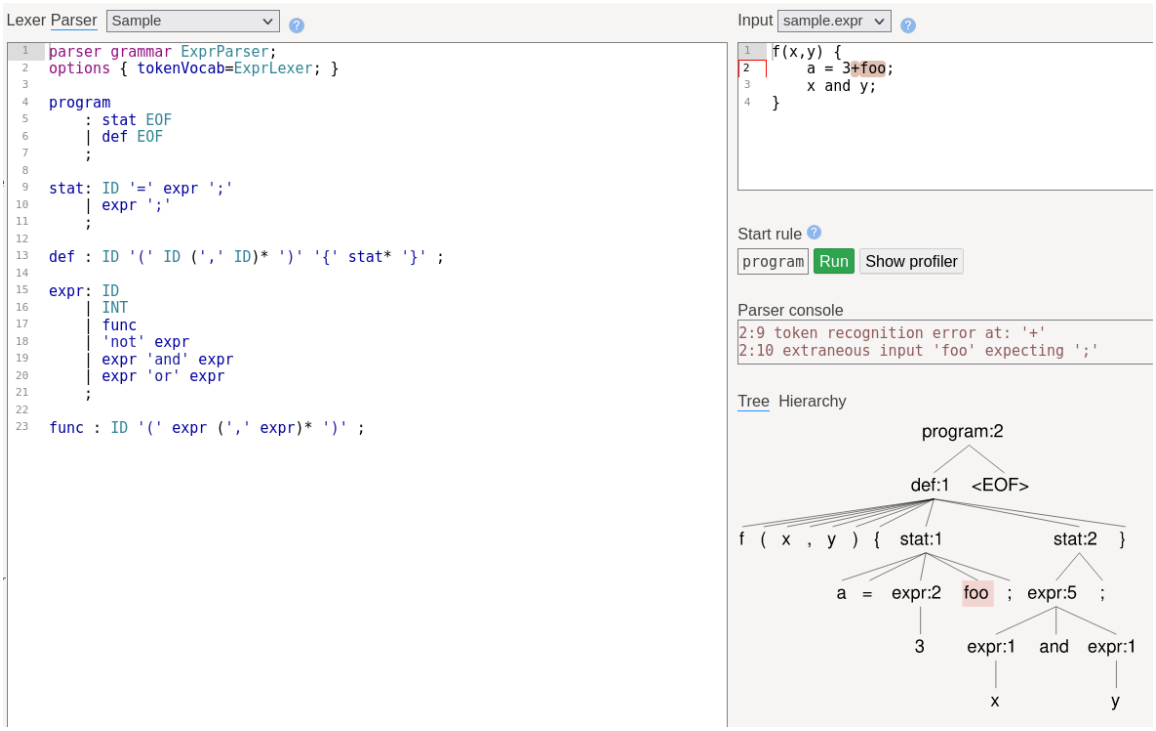
ANTLR – ANother Tool for Language Recognition v preklade ďalší nástroj na rozpoznávanie jazyka, je nástroj na tvorbu lexikálnych a syntaktických analýz pre rôzne jazyky. Bol prvýkrát vyvinutý v roku 1989 Terencom Parrom a jeho vývoj pokračuje dodnes. Tento nástroj je implementovaný v Jave, ale je možné ho používať vo viacerých jazykoch [13]. ANTLR je veľmi obľúbený a široko používaný hlavne kvôli jednoduchosti vývoja a čitateľnej a zrozumitelnej forme gramatiky. Medzi ďalšie výhody patria: silná typová kontrola, ktorá znižuje chyby počas kompilácie, schopnosť spracúvať komplexné jazyky a generovať efektívne parsre, ktoré dokážu zvládať zložité konštrukcie a výrazy a jednoduchosť pridá-

vania vlastností do vygenerovaných parserov prostredníctvom závislostí.

ANTLR pracuje vo viacerých fázach:

1. Vytvorenie gramatiky v podobe textového súboru.
2. Na základe gramatiky sa vygeneruje lexer a parser, ktorý kontroluje či je kód v danom jazyku napísaný správne podľa zadaných pravidiel.
3. Parser prechádza kód pričom delí reťazce na lexémy.
4. Z lexém sa vytvorí abstraktný syntaktický strom.

ANTLR je možné použiť v programovacích jazykoch ako napríklad C, C#, Java, JavaScript, Python, Go, PHP, ...



Obrázek 4.2: Príklad použitia nástroja ANTLR

Na obrázku 4.2 je zobrazené použitie nástroja ANTLR. Gramatika je definovaná pomocou formátu, ktorý zahŕňa zvlášť pravidlá pre lexer a zvlášť pre parser. Pravidlá pre lexer začínajú veľkým písmenom, zatiaľ čo pravidlá pre parser malým písmenom. Lexer pravidlá definujú, ako má byť vstupný text rozdelený na tokeny. Sú často definované pomocou regulárnych výrazov. Parser pravidlá definujú syntaktickú štruktúru jazyka pomocou pravidiel, ktoré sú aplikované na jednotlivé tokeny. Pri parseri je dôležité takzvané štartovacie pravidlo, ktoré signalizuje, kde má začať analýza vstupného textu. Štartovacie pravidlo, presnejšie jeho lavá strana, taktiež slúži ako koreň abstraktného syntaktického stromu a zahŕňa celý rozsah analyzovanej vety modelového jazyka.

Vytvorený abstraktný syntaktický strom je hierarchická datová štruktúra, ktorá reprezentuje syntaktickú štruktúru vstupného textu podľa pravidiel gramatiky.

Vo vytvorenom AST každý uzol predstavuje jednu časť syntaxe – napríkľad kľúčové slovo, premennú alebo funkciu. Vzťahy medzi uzlami reprezentujú, ako sú tieto časti syntaxe prepojené v rámci celého kódu.

4.2 Python tokenizer

V jazyku Python existuje modul `tokenize`, ktorý je súčasťou štandardnej knižnice a používa sa na tokenizáciu zdrojového kódu. Poskytuje spôsob, ako rozdeliť zdrojový kód na jednotlivé tokeny, ako sú napríklad kľúčové slová, identifikátory, literály a symboly.

Modul `tokenize` poskytuje radu funkcií pre prácu s tokenmi, ako napríklad `tokenize.tokenize()` a `tokenize.generate_tokens()`. Tieto funkcie vracajú tzv. iterátor [5].

V jazyku Python je odporúčané používať na tokenizáciu modul `tokenize`, keďže je implementovaný v jazyku C a je veľmi efektívny.

```
1 (type=59 (ENCODING), string='utf-8', start=(0, 0), end=(0, 0))
2 (type=1 (NAME), string='print', start=(1, 0), end=(1, 5))
3 (type=53 (OP), string='(', start=(1, 5), end=(1, 6))
4 (type=3 (STRING), string="'Hello, world!'", start=(1, 6), end=(1, 20))
5 (type=53 (OP), string=')', start=(1, 20), end=(1, 21))
6 (type=4 (NEWLINE), string='', start=(1, 21), end=(1, 22))
7 (type=6 (DEDENT), string='', start=(2, 0), end=(2, 0))
8 (type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0))
```

Výpis 4.1: Výstup po použití funkcie `tokenize` na príkaz `print('Hello, world!')`

Tento výstup obsahuje typ tokenu, jeho hodnotu a informáciu na ktorom riadku a stĺpci začal, a na ktorom končí. Tieto informácie sú veľmi užitočné a v nasledujúcich kapitolách si popíšeme ich využitie.

4.3 Tokenizácia v jazyku PHP

Funkcia `token_get_all()` je užitočný nástroj na tokenizáciu PHP kódu. Umožňuje rozklad PHP kódu na sériu tokenov, ktoré môžu byť následne analyzované.

Pri využití tejto funkcie stačí zadať vstupný reťazec, ktorý obsahuje zdrojový kód v jazyku PHP. Funkcia potom rozdelí zadaný vstup na tokeny a vráti ich vo forme poľa, v ktorom sa nachádza typ tokenu, jeho hodnota a pozícia v kóde. Jednotlivé typy tokenov je možné následne previesť na názvy pomocou funkcie `token_name()`[1].

[1] https://www.php.net/manual/en/function.token-get-all.php

```
T_OPEN_TAG, <?php ,1
T_ECHO, echo, 1
:
T_CONSTANT_ENCAPSED_STRING, 'Hello, world!', 1
;
T_CLOSE_TAG, ?>, 1
```

Výpis 4.2: Výstup po použití funkcie `token_get_all()` na príkaz `echo 'Hello, world!'`

Na tomto výstupe môžme vidieť, že obsahuje podobné informácie ako výstup funkcie `tokenize`, a to typ tokenu, jeho hodnotu a číslo riadku, na ktorom sa nachádza. Niektoré typy tokenov, ako napríklad `:` a `;` však obsahujú iba hodnotu tokenu. Informácie ako číslo riadku je možné doplniť podľa predchádzajúceho tokenu, keďže pred týmito tokenmi sa často nachádzajú iné tokeny a väčšinou nebývajú na začiatku nového riadku.

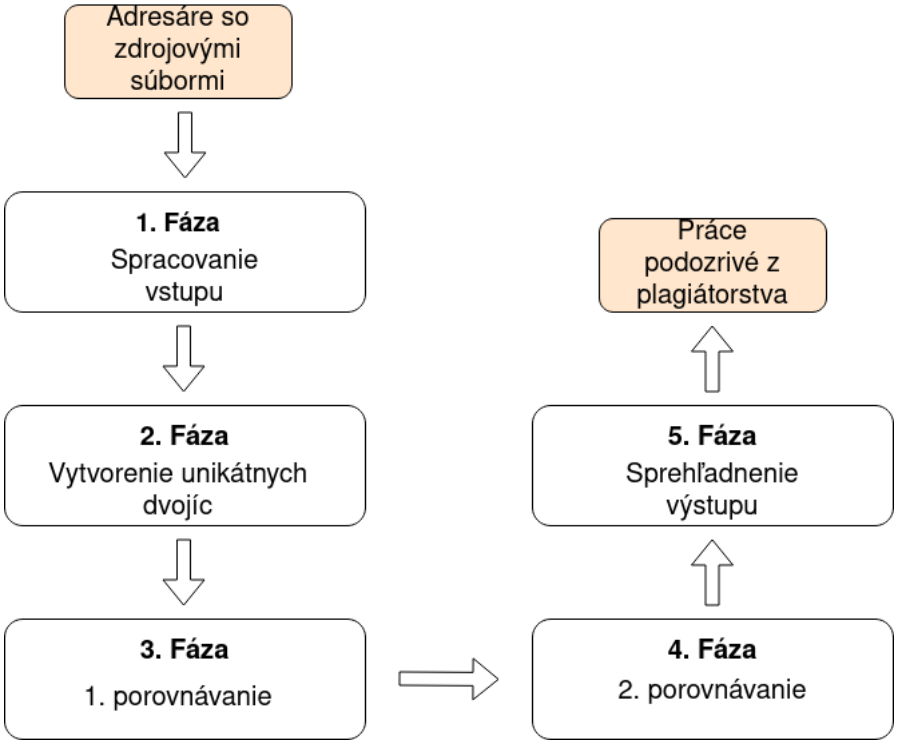
Kapitola 5

Návrh výslednej aplikácie

V tejto kapitole sa pozrieme na návrh výslednej aplikácie. Aplikáciu som sa rozhodol vytvoriť v jazyku Python, keďže mám s ním väčšie skúsenosti ako s ostatnými programovacími jazykmi.

Aby bola aplikácia interpretovateľná na školskom serveri Merlin, bude potreba, aby bola spustitelná interpretom Python verzie 3.10.

Na výslednú aplikáciu bolo viacero požiadavok. Aplikácia má pracovať v moduloch, ktoré sú na sebe nezávislé, takže implementáciu bude potrebné rozdeliť na viacero častí. Ďalej má aplikácia kontrolovať plagiáty vo viacerých programovacích jazykoch. Taktiež by malo byť možné, aby aplikácia bolá schopná prevádzať detekciu súčasne na viacerých projektoch.



Obrázek 5.1: Návrh výslednej aplikácie

Návrh aplikácie 5.1 je inšpirovaný predchádzajúcou bakalárskou prácou Ondreja Krpca [10]. Pre oba návrhy bolo prioritou, aby aplikácia bola ľahko rozšíriteľná a preto pracujú v moduloch. Jednotlivé moduly boli zvolené podobne, bol však pridaný modul v ktorom sa výsledky transformujú na prehľadnejší formát. Návrh samotných modulov však už prebiehal nezávisle od vyššie spomínaného návrhu. Na obrázku 5.1 sa nachádzajú jednotlivé moduly, v ktorých aplikácia bude pracovať.

Jednotlivé fázy si bližšie priblížime v nasledujúcich kapitolách.

5.1 Spracovanie argumentov

Keďže výsledkom práce bude konzolová aplikácia, bude potrebné vytvoriť parser argumentov. Jednotlivé argumenty boli zvolené nasledovne: `input` – adresár s projektmi z aktuálneho roka, `oldinput` – adresár s projektmi z minulých rokov, `first`, `second`, `third`, `fourth`, `fifth` – vykonávanie iba zvolenej fázy. Celý prehľad argumentov sa nachádza v prílohe A.

5.2 Spracovanie vstupu

Po spracovaní argumentov sa začne spracúvať vstup, tj. jednotlivé projekty. Postupne prebehne načítanie zdrojového súboru, jeho spracovanie, rozloženie na tokeny a uloženie všetkých potrebných informácií pre ďalšie časti do výstupného súboru. Výsledné dáta sú uložené do výstupného súboru.

Pri navrhovaní aplikácie padla zároveň požiadavka, skúsiť implementovať tzv. whitelist, na ktorom by boli časti kódu, ktoré sa zdieľajú na prednáškach, alebo sú vypísané v študíjnych oporách. Ide napríklad o regexy. Tento whitelist by bolo možné implementovať hneď v úvodnej časti pri tokenizácií, kde by sa z tokenov vyfiltrovali určité tokeny podľa vopred zadaného súboru.

Rovnako sa môže tento model použíť aj iba na samotné vytváranie výstupného súboru. Ako formát výstupného súboru bol najprv zvolený JSON, avšak nakoniec sa použil formát `msgpack`. `Msgpack` je binárny formát, ktorý umožňuje rýchle a kompaktné ukladanie dát. Tento formát umožňuje ukladanie informácií vo forme štruktúr, rovnako ako JSON. Naviac `msgpack` má v porovnaní s formátom JSON rýchlejšie serializačné a deserializačné časy. Prieskum [20] ukázal, že zatiaľ čo súbor vo formáte JSON mal veľkosť 359 bajtov, rovnaký súbor vo formáte `msgpack` mal veľkosť iba 231 bajtov.

5.3 Vytvorenie unikátnych dvojíc

Pretože cieľom je odhaliť všetky možné prípady plagiátov, je potrebné zaistiť, aby sa každý dokument porovnal s každým iným, aby sa zabezpečilo, že sa žiadny dokument neprehliadne. To sa docieli v tejto fáze tým, že sa zo všetkých názvov projektov vygenerujú unikátne kombinácie.

Požiadavkou na nástroj bolo taktiež, aby bolo možné kontrolovať plagiáty aj formou porovnávania so staršími projektmi. Tu však stačí projekty porovnávať iba s novšími, a tým zamedziť, aby študenti kopírovali staršie projekty.

5.4 Porovnávanie

Ako je vidieť v obrázku 5.1, táto fáza sa vo výslednom návrhu vyskytuje dvakrát.

V prvom prípade je potrebné vytriediť projekty, ktoré sú z plagiátorstva skutočne podozrivé. V predchádzajúcej kapitole bol vidieť nárast počtu dvojíc pri počte projektov. V predmete IPP, pre ktorý je táto aplikácia primárne robená, je počet študentov približne 500. Počet unikátnych dvojíc je teda približne 125000. Na toto porovnávanie sa použijú algoritmy, ktoré niesú výpočetne až tak náročné. V predchádzajúcej implementácii sa použili algoritmy Halsteadova metrika a Levenshteinov algoritmus.

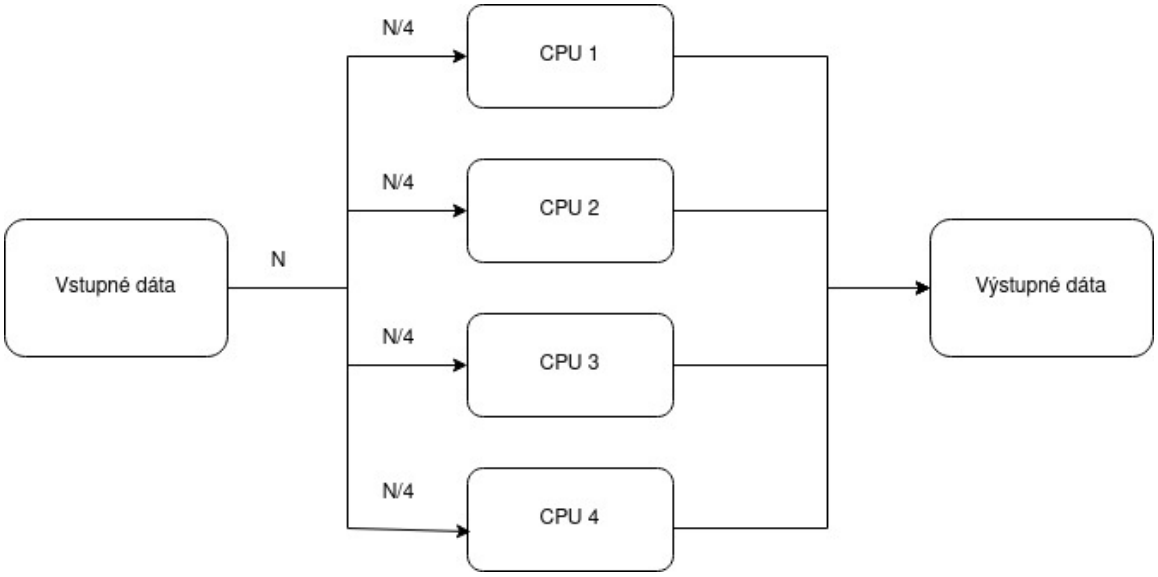
Druhá fáza porovnávania bude slúžiť na presnejšie vyhladávanie plagiátov. V tejto fáze sa môže použiť algoritmus, ktorý je výpočetne náročnejší, keďže sa nepredpokladá, že ostane veľa podozrivých dvojíc.

5.5 Súčasné prevádzanie detekcií

Problém súčasného prevádzania detekcií som sa rozhodol riešiť Python multiprocessingom.

Je to technika, ktorá umožňuje spustenie viacerých procesov súčasne v rámci jedného programu. Táto vlastnosť sa dá využiť na zvýšenie výkonu programu, keďže sa tým využijú viaceré jadrá procesora. Multiprocessing je implementovaný v Pythone prostredníctvom modulu `multiprocessing`. Tento modul poskytuje niekoľko spôsobov, ako vytvárať a spúšťať paralelné procesy, napríklad pomocou funkcie `Process` [19].

Medzi jednotlivými procesmi je možné zdieľať dáta prostredníctvom zdieľaných pamätí, alebo pomocou komunikačných kanálov, akým je napríklad `pipe`.



Obrázek 5.2: Príklad multiprocessingu

Po úspešnom dokončení všetkých procesov sa výsledky spoja a je možné ich uložiť do jednej premennej, s ktorou sa dá ďalej pracovať.

Kapitola 6

Implementácia

Ako už bolo načrtnuté v návrhu, výsledný nástroj sa bude skladať z 5 častí. V nasledujúcej kapitole si predstavíme jednotlivé časti a ich implementáciu.

6.1 Popis jednotlivých častí nástroja

V súbore `controller.py` sa nachádza hlavná časť výslednej aplikácie. V nej sa postupne volajú zvyšné moduly nevyhnutné pre správne fungovanie nástroja.

Spracovanie argumentov

V súbore `arg_parser.py` sa nachádza trieda `ArgumentParser`, ktorá slúži na spracovanie argumentov. V tejto triede sa uložia všetky potrebné informácie slúžiace k behu programu. Ak bol zadaný neplatný argument, alebo bola zadaná neplatná kombinácia argumentov, propaguje sa výnimka a program sa ukončí. Prehľad jednotlivých argumentov sa nachádza v prílohe A. Taktiež všetky výstupné súbory z nasledujúcich fáz sú ukladané do samostatného adresára.

Prvá fáza – vytvorenie súboru s dátami

Po spracovaní argumentov sa začne spracovanie jednotlivých projektov a ich uloženie do štruktúry. Obsah danej štruktúry je zvolený podobne, ako vo svojej práci zvolil Ondrej Krpec [10], kde pre každý projekt je vytvorený jeden objekt s názvom podľa prihlasovacieho mena autora projektu. Do tohto objektu je ďalej uložená cesta k súboru a pole `files`, do ktorého je spracovaný každý súbor v danom adresári.

Pre každý súbor je postupne uložený jeho názov, do poľa `tokens` sú ukladané tokeny ako polia skladajúce sa z typu tokenu, jeho hodnoty a čísla riadku, na ktorom sa nachádza. Čísla riadkov sú ukladané kvôli spätnej dohladateľnosti pri určovaní, v ktorých miestach sú jednotlivé dvojice zdrojových kódov podobné. Ďalej sú do poľa `halstead_blocks` ukladané operátory a operandy celého zdrojového kódu pre neskoršie použitie Halsteadovych metrík, rovnako aj do pola `function_halstead_blocks` operátory a operandy jednotlivých funkcií. Do pola `levenshtein_blocks` sa ukladajú typy jednotlivých tokenov pre počítanie Levenshteinovej vzdialenosti.

Vytváranie tejto štruktúry prebieha v súbore `phase_1.py`, kde sa buď volá tokenizátor pre PHP projekty uložený v súbore `php_tokenizer.py`, alebo tokenizátor pre Python projekty, nachádzajúci sa v súbore `python_tokenizer.py`, podľa zvoleného parametra.

Výsledný súbor v tejto fáze nie je uložený ako JSON, ako bolo spomenuté v návrhu, ale vo formáte `msgpack`[1]. Keďže nie je potrebné, aby tieto dáta boli čitateľné pre ľudí, tak bol zvolený spomenutý formát, ktorý používa binárny formát na reprezentáciu dát. Taktiež sa z dôvodu zmenšenia dát rozdelil výsledný súbor z tejto fázy na dva súbory.

Prvý súbor obsahuje dáta potrebné pre tretiu fázu, a to výpis operátorov a operandov z celého zdrojového kódu, ďalej výpis operátorov a operandov pre jednotlivé funkcie, levenshteinove bloky, a pole `comments`, kde sú uložené komentáre. Taktiež štruktúra tohto súboru sa generuje podľa zadaných parametrov, teda ak bol zvolený iba parameter `-levenshtein`, súbor obsahuje iba levenshteinove bloky.

```
{
    "xlogin00": {
        "path": "Cesta k suboru",
        "dir": "Nazov adresaru",
        "files": [
            {
                "filename": "Nazov suboru",
                "content": {
                "halstead_blocks": [
                    Vypis operatorov a operandov celeho zdrojoveho kodu
                ],
                "function_halstead_blocks" :[
                    Vypis operatorov a operandov jednotlivych funkcii
                ]
                "levenshtein_blocks": [
                    [Pole s typmi tokenov]
                ],
                "comments": [
                    Vypis vsetkych komentarov v zdrojovom kode
                ]
            }
            }
        ]
    }
}
```

Výpis 6.1: Štruktúra súboru vytvoreného v prvej fáze

Druhý súbor obsahuje iba jednotlivé tokeny a je vytvorený až po tretej fáze, avšak tiež triedou `DirectoryWorker` v súbore `phase_1.py`. Taktiež dáta uložené v tomto súbore niesú generované pre každý projekt, ale iba pre projekty, ktoré sa porovnávajú v štvrtej fáze.

[1] https://msgpack.org/index.html

```
{
    "xlogin00": {
        "path": "Cesta k suboru",
        "dir": "Nazov adresaru",
        "files": [
            {
                "filename": "Nazov suboru",
                "content": {
                "tokens": [
                    [
                        Typ tokenu,
                        Hodnota tokenu,
                        Cislo riadku v kode
                    ],
                ]
            }
            }
        ]
    }
}
```

Výpis 6.2: Štruktúra súboru vytvoreného po tretej fáze

Výsledkom tejto fázy je teda `msgpack` súbor s dátami o jednotlivých projektoch a CSV súbor, ktorý obsahuje prihlasovacie meno autora a informáciu o tom, či je daný projekt aktuálny, teda z tohto roku, alebo je už starší, teda z minulých rokov. Tento súbor je dôležitý pre vytváranie dvojíc, keďže projekty z minulých rokov nepotrebujeme kontrolovať medzi sebou.

```
    Project Name,Project Type
    xlogin01,new
    xlogin02,new
    xlogin03,new
```

Výpis 6.3: CSV súbor s informáciou o aktuálnosti projektu

Druhá fáza – vytvorenie unikátnych dvojíc

V súbore `pairs.py` prebieha generovanie unikátnych dvojíc. Najskôr sa načítajú dáta z prvej fázy z CSV súboru a následne sa pomocou knižnice `itertools` generujú unikátne dvojice. Ako už bolo povedané skôr, nové projekty sú párované medzi sebou, zatiaľ čo staršie projekty sú párované iba s novými. Výsledné dvojice sú uložené ako objekt iterátor[2] a následne sa zapisujú do CSV súboru pomocou modulu `csv`[3].

[2] https://wiki.python.org/moin/Iterator
[3] https://docs.python.org/3/library/csv.html

```
xlogin01,xlogin02
xlogin01,xlogin03
xlogin02,xlogin03
```

Výpis 6.4: Štruktúra výsledného CSV súboru v druhej fáze

Tretia fáza – filtrovanie dvojíc

V tejto fáze programu prebieha prvé porovnávanie, ktoré má za cieľ vyfiltrovať dvojice, ktoré niesú podozrivé z plagiátorstva. Jadro tejto fázy sa nachádza v súbore `phase_3.py`, kde sa najprv načítajú dáta z prvej fázy a dvojice z druhej fázy, ktoré sa podľa počtu procesov rozdelia. Následne sa pre každý pár počítajú jednotlivé metriky zvolené pri vstupe.

V súbore `comments.py` sa nachádza jednoduchá metrika pre určenie podobnosti komentárov dvojice projektov. Táto metrika sa nazýva TF-IDF a jej výsledkom je percentuálna podobnosť komentárov oboch zdrojových súborov. Podobnosť sa počíta podľa takzvanej kosínusovej vzdialenosti dvoch polí, v ktorých sú uložené komentáre oboch projektov. Táto metrika však dokáže byť sama o sebe veľmi nepresná, pretože ak plagiátor vymaže všetky komentáre, teda jeden z projektov neobsahuje žiaden komentár, je vrátená nulová podobnosť.

V súbore `halstead.py` sa nachádzajú Halsteadove metriky, ktoré sú počítane pre celý projekt, ako aj pre každú funkciu zvlášť. Následne sú výsledky týchto metrík porovnané a výsledkom je podobnosť zdrojových súborov v percentách.

V súbore `levenshtein.py` je implementovaná Levenshteinova vzdialenosť. Pri implementácii som použil funkciu `distance` z modulu `Levenshtein`[4], pretože táto funkcia bola neporovnateľne rýchlejšia ako vlastná implementácia. Vstup je rozdelený do viacerých polý, kde maximálna veľkosť jedného pola je 255. Následne sa pre každé pole sčíta výsledok už spomínanej funkcie `distance`, ktorý udáva najmenšiu vzdialenosť na zmenenie jedného bloku, aby bol rovnaký ako druhý blok. Výsledkom tejto metriky je taktiež percentuálna podobnosť dvojice zdrojových súborov.

Po vypočítaní všetkých metrík sa jednotlivé výsledky pre každú dvojicu sčítajú a vydelia sa počtom použitých metrík. Ako výstup tejto fázy bol taktiež zvolený CSV súbor.

```
xlogin01,xlogin02,percentualna podobnost
xlogin01,xlogin03,percentualna podobnost
xlogin02,xlogin03,percentualna podobnost
```

Výpis 6.5: Štruktúra CSV súboru z tretej fázy

Štvrtá fáza – detailné porovnávanie

Ďalšou časťou programu je detailné porovnávanie. V tejto časti sa taktiež najprv načítajú dáta z prvej a tretej fázy a následne sa rozdelia podľa počtu procesov rovnako ako v tretej fáze. Ak mal pár zhodu v predchádzajúcom porovnávaní väčšiu ako 60%, tak sa na danom páre spustí detailnejšie porovnanie.

[4] https://pypi.org/project/python-Levenshtein/

Ako algoritmus pre detailné porovnávanie som zvolil vyššie spomenutý `Greedy String Tiling`. Keďže je tento algoritmus výpočetne veľmi náročný, musel som modul, v ktorom sa nachádza implementovať pomocou `Python/C API`[5]. Táto implementácia sa nachádza v súbore `greedy_string_tiling.c` a pri testovaní bola približne päťkrát rýchlejšia, ako implementácia v jazyku Python. Výstup tohto algoritmu je pole datových štruktúr tuple, v ktorých sú uložené postupne: index tokenu v prvom porovnávanom projekte, index tokenu v druhom porovnávanom projekte a dĺžka zhody. Následne sa podľa indexu tokenov zistí súbor, v ktorom sa daná zhoda nachádza a presné riadky, ktoré sú zhodné. Výstupom tejto časti je CSV súbor, v ktorom sú uložené názvy adresárov oboch projektov, percentuálna podobnosť projektov a mená súborov a riadky, ktoré boli zhodné.

```
1    xlogin01, xlogin02, percentualna podobnost, [zhodne casti]
2    xlogin01, xlogin03, percentualna podobnost, [zhodne casti]
3    xlogin02, xlogin03, percentualna podobnost, [zhodne casti]
```

Výpis 6.6: Štruktúra CSV súboru z tretej fázy

Súčasné vykonávanie detekcií a ukladanie stavov

V tretej a štvrtej časti bakalárskej práce sa program zaoberá spracovaním veľkého množstva dvojíc projektov. Pre zefektívnenie a skrátenie behu programu bolo nevyhnutné zaviesť paralelné vykonávanie detekcií. Tento účel bol dosiahnutý vďaka knižnici `multiprocessing`[6], ktorá poskytuje jednoduchý spôsob vytvárania a riadenia paralelných procesov v jazyku Python. Prístup využívajúci paralelné vykonávanie detekcií umožňuje programu efektívnejšie využiť dostupné zdroje viacjadrových procesorov, čo vedie k značnému zrýchleniu celého procesu spracovania dvojíc projektov.

Na začiatku oboch spomínaných fáz programu sa najskôr definuje funkcia, ktorá bude vykonávaná paralelne. Potom sa dvojice projektov rozdelia do skupín, zodpovedajúcich počtu procesov. V každom procese je následne zavolaná táto funkcia, ktorá dostane ako argumenty zoznam dvojíc na spracovanie a tiež dáta z prvej fázy. Po dokončení všetkých procesov sa výsledky z jednotlivých procesov zlúčia a následne sa s nimi pokračuje ďalším spracovaním. Základný počet procesov bol zvolený podľa počtu CPU, ktoré má k dispozícii testovacie prostredie, pomocou funkcie `cpu_count()` z modulu `multiprocessing`.

Vzhľadom k tomu, že jednou z požiadaviek na nástroj bolo umožnenie prerušenia detekcií v tretej a štvrtej fáze, bolo nevyhnutné navrhnúť spôsob ukladania stavov jednotlivých procesov. Riešením tohto problému bolo vytvorenie súboru pre každý proces po prerušení behu programu, v ktorom sa uložia jeho stavy. Okrem toho bolo potrebné ukladať aj zdieľanú premennú, ktorá sa používa na zobrazenie celkového pokroku spracovania dvojíc projektov. Zdieľaná premenná bola taktiež použitá pre ukladanie výsledkov algoritmu. Pretože zapisovanie do zdielaných premenných nemôže prebiehať naraz, musel byť taktiež použitý takzvaný zámok, ktorý zosynchronyzoval zapisovanie do zdielaných premenných. Následne je možné po zadaní parametru `resume` pokračovať vo vykonávaní detekcií.

Pre ukladanie dát bol zvolený formát `pickle`[7], ktorý je vhodný na ukladanie stavov programu. Pickle ukladá dáta v binárnom formáte, čo je veľmi efektívne z hľadiska rýchlosti

[5] https://docs.python.org/3/c-api/index.html
[6] https://docs.python.org/3/library/multiprocessing.html
[7] https://docs.python.org/3/library/pickle.html

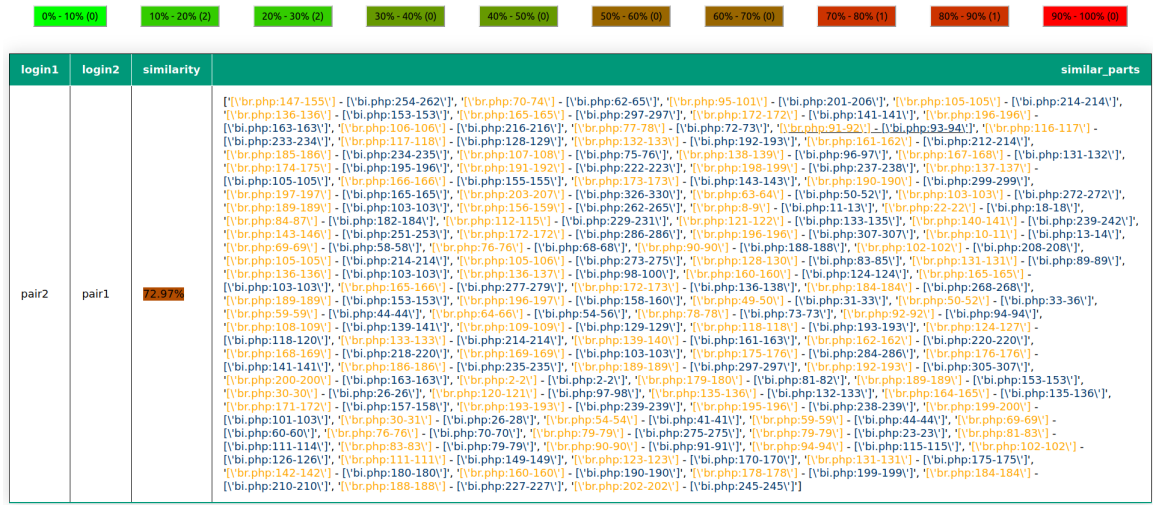
čítania a zápisu stavov. Medzi niektoré ďalšie výhody knižnice patrí jej schopnosť ukladať rôzne typy objektov a jednoduchosť jej použitia. Avšak jednou z nevýhod je nekompatibilita s inými programovacími jazykmi, čo znamená, že je vhodná iba pre použitie v jazyku Python.

Piata fáza – sprehľadnenie výstupu

V poslednej časti programu sa pre lepší prehľad štýluje výsledný CSV súbor zo štvrtej fázy na HTML súbor. Na načítanie súboru a prácu s ním je použitá knižnica Pandas, ktorá je veľmi výkonná a zároveň poskytuje široké možnosti spracovania CSV súboru pomocou štruktúry, ktorý sa nazýva `Dataframe`. Tento `Dataframe` je vlastne tabuľka, s ktorou sa dá rýchlo a efektívne pracovať[12].

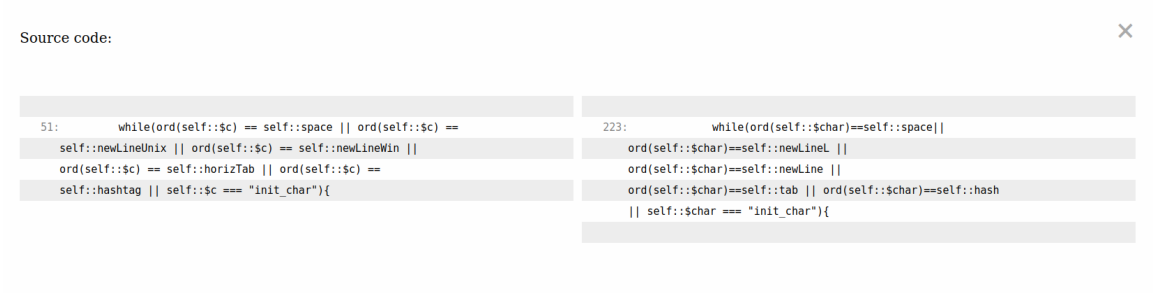
Po načítaní vstupného súboru a jeho transformácii na `Dataframe` sa táto tabuľka zoradí podľa stĺpca similarity a ďalej sa pracuje už iba s prvými päťdesiatimi hodnotami. Pre každú zhodu v stĺpci similar_parts sa načítajú dané riadky zo súboru, ale ostanú skryté.

Výsledný HTML súbor vyzerá nasledovne:



Obrázek 6.1: Výsledný HTML súbor

Na vrchu súboru sú pre lepšiu prehľadnosť tlačítka v intervaloch 10% a v zátvorke je pri nich počet dvojíc. Po kliknutí na dané tlačítko sa zobrazia všetky dvojice z intervalu.



Obrázek 6.2: Zobrazenie dvojice zo stĺpca `similar_parts`

Ďalej je možné kliknúť na jednotlivé páry v stĺpci `similar_parts`, kde sa po kliknutí zjaví modálne okno, ktoré obsahuje dané riadky zo súbora.

Okrem HTML súboru sa taktiež v tejto časti vytvoria dva grafy pomocou knižnice `matplotlib`. V týchto grafoch je zobrazený výsledok tretej a štvrtej fázy v podobe počtu dvojíc v jednotlivých intervaloch.

Logovanie

Nástroj obsahuje taktiež modul na logovanie, ktorý zaznamenáva dôležité informácie o priebehu programu. Logovanie poskytuje prehľad o začiatku, priebehu a dokončení jednotlivých fáz, čo značne zjednodušuje sledovanie a analýzu výkonnosti nástroja. Okrem toho, v prípade výskytu chýb, sú tieto informácie zaznamenané do logu, čo uľahčuje ich následné riešenie a ladenie.

Modul logovania je implementovaný v súbore `logger.py`. V tomto súbore je taktiež implementovaná funkcia, pomocou ktorej sa meria pokrok spracovávania v tretej a štvrtej fáze programu.

6.2 Možné rozšírenia

Nástroj bol navrhnutý tak, aby bolo jednoduché pridať doň jednotlivé rozšírenia. Existuje veľa rôznych možností, ako tento nástroj zdokonaliť. Niektoré z nich si predstavíme v nasledujúcich bodoch:

- Pridávanie jazykov – ako už bolo spomenuté vyššie, podpora jednotlivých jazykov sa nachádza v separátnych moduloch. Pre pridanie nového jazyka teda stačí, aby sa vytvoril nový modul s triedou, ktorá má rovnaké metódy ako triedy `PhpTokenizer` a `PythonTokenizer`.

- Pridávanie algoritmov – algoritmy sa rovnako ako tokenizátory pre jednotlivé jazyky nachádzajú v separátnych moduloch. Pre pridanie nového algoritmu teda stačí jeho implementáciu pridať či už do triedy `FilterTesting`, alebo `DeepTesting`, podľa toho, či chceme aby bol daný algoritmus použitý pre filtovanie dvojíc, ktoré niesú podzrivé z plagiátorstva, alebo na hĺbkové porovnávanie.

- Pridanie nástroja ANTLR – pridanie nástroja ANTLR by dokázalo znížiť požiadavky na prostredie, v ktorom sa nástroj používa. Keďže momentálne nástroj používa funkciu na tokenizáciu z jazyka PHP, taktiež vyžaduje, aby bol tento jazyk nainštalovaný na zariadení. Pri pridávaní ďalších jazykov by teda mohol vzniknúť problém, že dané zariadenie bude musieť mať všetky tieto jazyky nainštalované. Tomu by sa dalo vyhnúť použitím nástroja ANTLR, ktorý na svoj beh potrebuje iba gramatiky jazykov, ktoré sa dajú nájsť na ich Github stránke[8].

- Pridanie strojového učenia – rovnako ako nástroj Turnitin, aj nástroj navrhnutý v tejto bakalárskej práci, by mohol obsahovať algoritmus, ktorý je schopný odhaliť parafrázovanie jednotlivých funkcii, či dokonca celého kódu. Táto vlastnosť by sa dala dosiahnuť

[8] https://github.com/antlr/grammars-v4

pridaním modelu, ktorý by bol natrénovaný na dátach jednotlivých projektov z predošlých rokov. V jazyku Python existuje množstvo knižníc, ktoré by sa na toto rozšírenie dali použiť, ako napríklad `scikit-learn`[9].

- Pridanie informačného systému – Integrácia s existujúcim informačným systémom (napr. školským alebo univerzitným) by umožnila ľahší prístup k relevantným dátam, ako sú napríklad zdrojové kódy študentských prác, informácie o študentoch a ich predchádzajúcich prácach. Toto rozšírenie by uľahčilo automatické overovanie podobnosti kódu a zvýšilo efektivitu procesu hodnotenia. Pridanie informačného systému by tiež umožnilo sledovať vývoj študentských prác počas semestra a automaticky generovať reporty s výsledkami analýz plagiátorstva.
- Porovnanie s výsledkami testov – zvýšiť presnosť tohto nástroja by mohlo porovnávanie s výsledkami testov, podľa ktorých sa projekty hodnotia. To by sa dalo docieliť pridaním parametra, do ktorého sa zadá cesta k súboru v ktorom sú uložené výsledky jednotlivých testov.
- Upravenie už existujúceho vizuálneho porovnania – ak sa nachádza v stĺpci similar_parts mnoho zhôd, tak sa existujúca HTML stránka stáva neprehľadnou. Možnosťou by bolo inšpirovanie sa vizualizáciou nástroja JPlag, ktorý zobrazuje celé zdrojové kódy a zvýrazňuje podobné časti. Taktiež by pre lepšiu analýzu bolo možné do HTML súbora pridať grafy vygenerované v piatej fáze.

[9] https://scikit-learn.org/stable/

Kapitola 7

Testovanie

Po úspešnej implementácii všetkých častí nástroja sme prešli k testovaniu. Testovanie prebiehalo na projektoch z predmetu Princípy programovacích jazykov a OOP (IPP), ktorý je vyučovaný na Fakulte informačných Technológií VUT v Brně. Testy prebiehaly na severy Merlin s operačným systémom CentOS Linux 7, na ktorom je nainštalovaný interpret jazyka Python 3.10 a rovnako aj jazyka PHP 8.1. Do tohto testovacieho prostredia však bolo potrebné stiahnuť Python 3 knižnice, ktoré sa tu nenachádzajú. Ide hlavne o knižnice, ktoré urýchlujú proces spracovávania dát.

7.1 Testovanie na projektoch v predmete IPP

Jedným z mojich cielov v tejto práci bolo vytvoriť nástroj v predstihu, aby sa stihol vyskúšať na projektoch v predmente IPP. To sa čiastočne podarilo, keďže nástroj bol úspešne spustený na prvom projekte, ktorý bol Analyzátor kódu IPPcode v PHP. Testovanie druhého projektu – Interpretu XML reprezentácie kódu napísaného v Pythone verzie 3 však narazilo na niekoľko nečakaných problémov, ktoré zabránili jeho úspešnému testovaniu. Kvôli zákonu o ochrane osobných údajov mal prístup k zdrojovým kódom od študentov iba vedúci tejto bakalárskej práce, čo tiež trochu sťažilo testovanie.

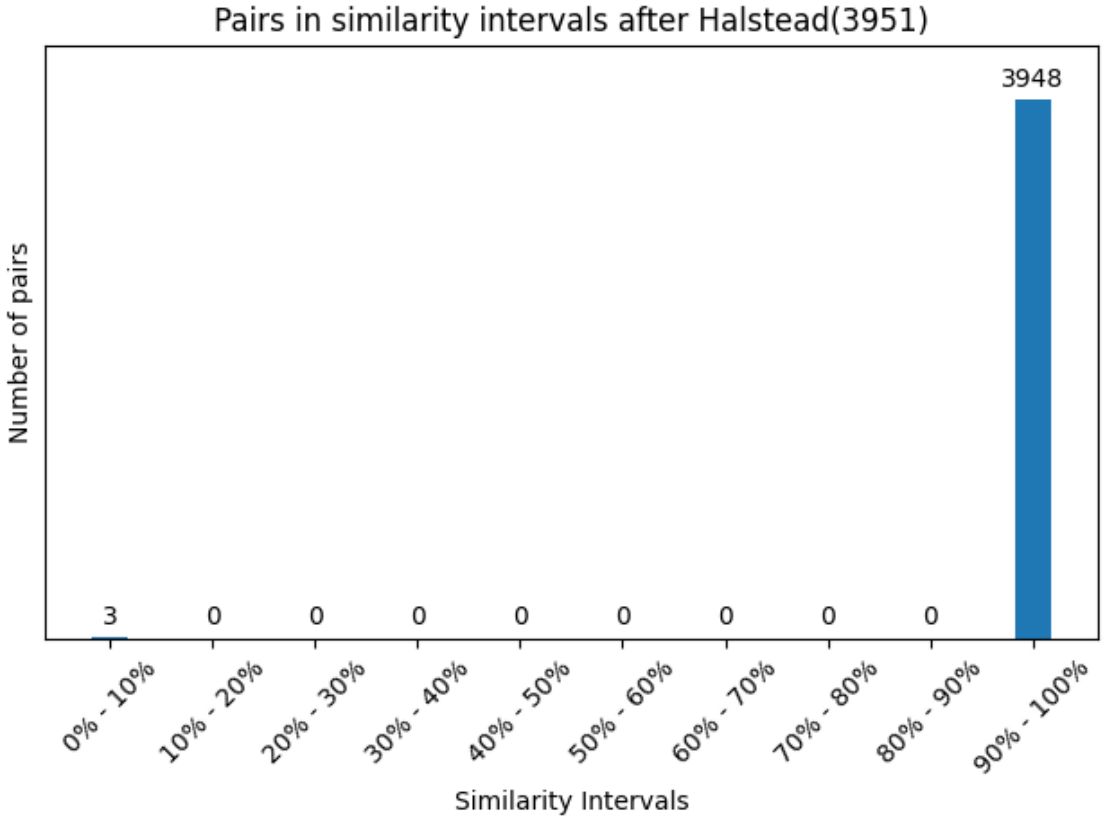
Testovanie PHP zdrojových kódov

Pri testovaní PHP súborov sa nevyskytli žiadne závažné problémy. Program sa správal podľa očakávania a dokonca pomohol odhaliť 3 plagiáty.

V tomto testovaní bol ešte použitý JSON formát súboru s dátami, keďže prvý projekt nebol až tak dlhý, a preto sa všetky dáta zmestili do jednej JSON štruktúry. Implementácia Levenshteinovej vzdialenosti sa ukázala ako účinná, keďže dokázala vyfiltrovať dvojice, ktoré vôbec neboli podozrivé. Ako je však vidieť na grafe 7.1, použietie samotnej Halsteadovej metriky na filtrovanie dvojíc nieje vhodné, keďže takmer všetky páry boli určené ako podobné, s podobnosťou 90 % a viac.

Tretia fáza testovania mala za úlohu odstrániť z ďalšieho testovania dvojice, ktoré neboli podozrivé z plagiátorstva, aby sa predišlo porovnávaniu každého páru v nasledujúcej fáze. Keďže Halsteadova metrika nesplnila svoju úlohu a odfiltrovala iba 3 páry, do štvrtej fázy

prešlo zbytočne príliš veľa párov. V dôsledku toho sme museli v nasledujúcej fáze porovnať väčší počet párov, čo zvýšilo časovú náročnosť testovania.

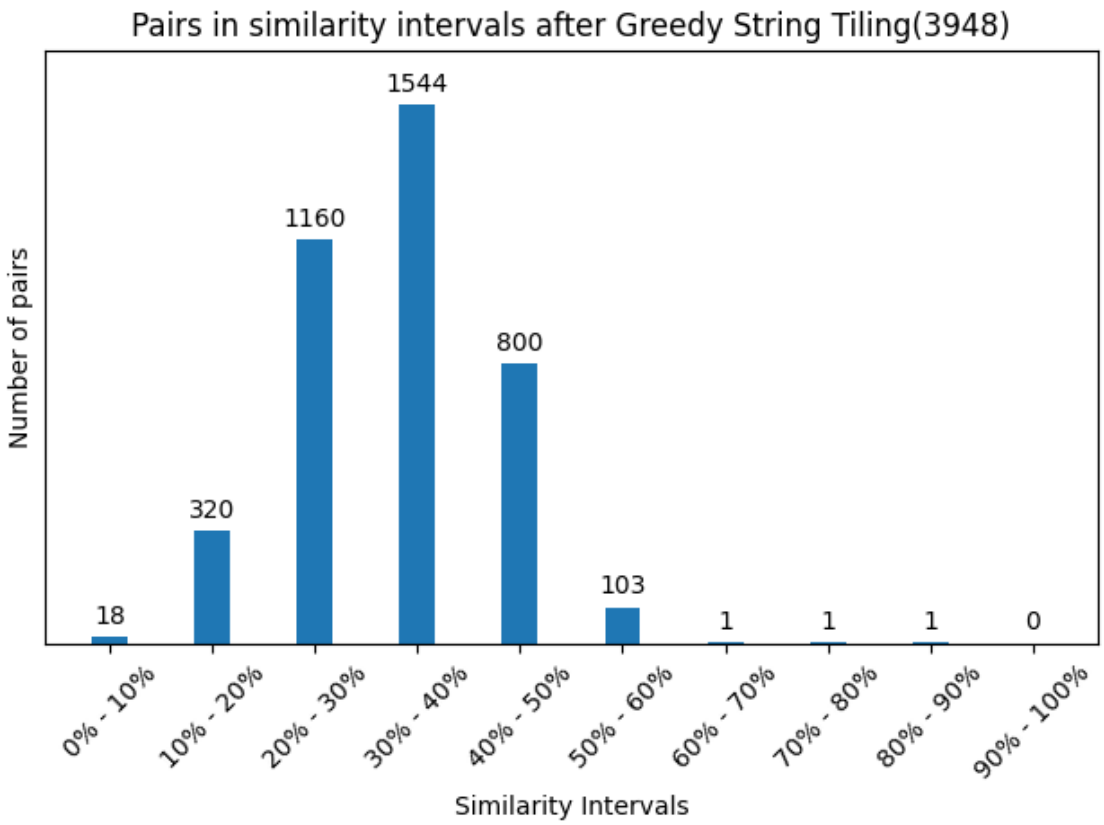


Obrázek 7.1: Graf podobnosti po tretej fáze behu programu

Aj keď použitie algoritmu Greedy String Tiling je časovo náročnejšie ako predchádzajúce algoritmy, ukázalo sa, že funguje na detekciu podobnosti zdrojových kódov výrazne lepšie. Na grafe 7.2 je vidieť, že podobnosť vyššiu ako 50 % má iba malé percento dvojíc z celkového počtu a podobnost vyššiu ako 60 % majú iba 3 dvojice.

V tejto fázi projektu bolo veľkým problémom zrýchlenie porovnávania. Keďže predchádzajúca implementácia v jazyku Python 3 nebola dostatočne rýchla, bolo potrebné vytvoriť Python 3 modul napísaný v jazyku C. To sa docielilo pomocou Python C API rozšírenia, v ktorom bol algoritmus niekoľkonásobne rýchlejší.

Podľa výsledkov v tejto fáze bolo možné povedať, že za to, aby dvojica bola prehlásená za podozrivú, musí mať prinajmenšom zhodu 60-70 %. Ak aj dvojica podozrivá je, neznamená to však hneď, že je aj plagiátom. Stále je potrebná bližšia analýza, kvôli ktorej bol vytvorený HTML výstup.



Obrázek 7.2: Graf podobnosti po štvrtej fáze behu programu

7.2 Testovanie Python 3 zdrojových kódov

Testovanie Python 3 zdrojových súborov však neprebiehalo tak hľadko, ako PHP súborov. Vyskytlo sa hneď viacero problémov, keďže druhý projekt z predmetu IPP bol omnoho mohutnejší ako prvý. Hneď v prvej fáze tak vznikol problém, že JSON štruktúra bola príliš veľká na to, aby mohla byť uložená, keďže JSON súbor môže mať maximálnu veľkosť iba 1 GB[1]. To sa vyriešilo tým, že súbor sa rozdelil na 2 časti – dáta potrebné pre tretiu fázu a dáta potrebné pre štvrtú fázu. Taktiež bol súbor uložený už vyššie spomínaným formátom `msgpack`, ktorý zaberá omnoho menej pamäte ako formát JSON. Taktiež musel byť obmedzený počet dvojíc, ktoré sa dostanú do štvrtej fázy. To sa docielilo volitelným argumentom, ktorý ma základnú hodnotu nastavenú na 50.

Aj keď sa pri testovaní Python 3 kódu vyskytlo viacero chýb, a kvôli nim nemohol byť testovaný na druhom projekte v predmete IPP, nakoniec bola väčšina chýb opravená a nástroj bol doladený na anonymizovaných projektoch, ktoré boli z plagiátorstva podozrivé. Nástroj je však na mohutnejších projektoch veľmi pomalý a preto sa neodporúča púšťať pre veľa projektov. Nakoniec boli pomocou nástroja odhalené 2 plagiáty, ktoré potvrdil aj

[1] https://dev.mysql.com/blog-archive/how-large-can-json-documents-be/

nástroj JPlag. Nástroj JPlag naviac našiel dva ďalšie plagiáty, ktoré tento nástroj neodhalil, keďže bolo príliš pomalé spustiť ho na všetkých projektoch.

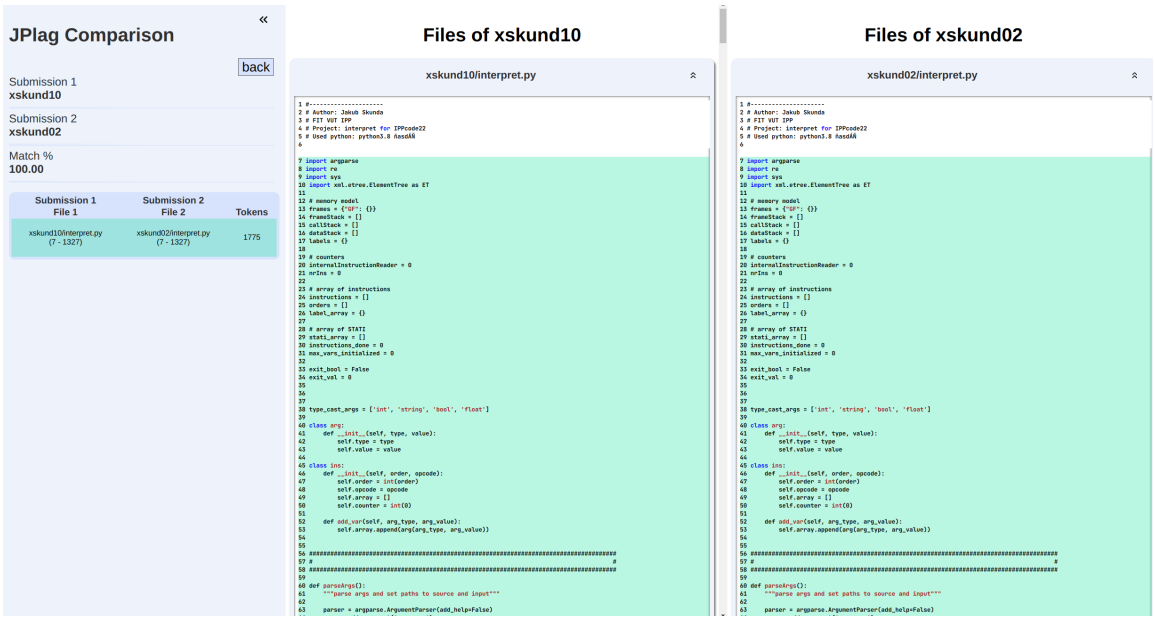
7.3 Porovnanie s už existujúcimi nástrojmi

JPlag

Nástroj vytvorený v tejto bakalárskej práci je s nástrojom JPlag veľmi podobný. Oba nástroje používajú takzvaný Greedy String Tiling algoritmus na detekciu plagiátov. Taktiež vstupné parametre nástroja vytvoreného v tejto bakalárskej práci a nástroja JPlag sú veľmi podobné[2]. Oba nástroje sú navrhnuté tak, aby boli ľahko rozšíriteľné, čo umožňuje pridávanie nových metód a podpory pre ďalšie jazyky bez veľkých zmien v kóde.

JPlag však obsahuje podporu pre omnoho viac programovacích jazykov, zatiaľ čo mnou vytvorený nástroj podporuje iba jazyky Python a PHP, a je vytvorený v jazyku Java, ktorý je väčšinou rýchlejší ako jazyk Python.

Výstupom nástroja JPlag je zip súbor, ktorý obsahuje informácie o podobnosti projektov. Po nahratí tohto súboru na stránku[3] sa zobrazí vizuálny rozdiel medzi projektmi.



Obrázek 7.3: Zobrazenie dvoch rovnakých súborov nástrojom JPlag

MOSS

Narozdiel od nástroja vytvoreného v tejto bakalárskej práci, ktorý používa algoritmus Greedy String Tiling, MOSS používa na detekciu plagiátov algoritmus založený na takzvaných odtlačkoch. MOSS taktiež nie je open-source projekt, takže nie je jednoducho rozšíriteľný, keďže k nemu nemá prístup každý. Pre použitie nástroja MOSS je potrebná registrácia.

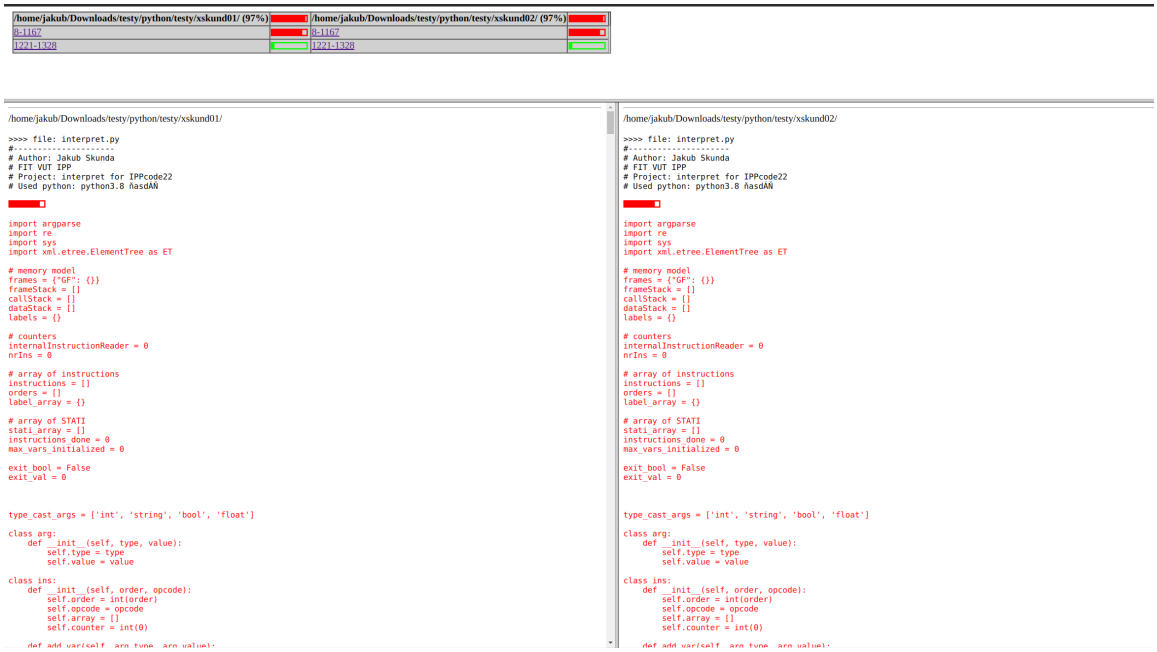
[2] https://github.com/jplag/JPlag
[3] https://jplag.github.io/JPlag/

MOSS podporuje veľké množstvo programovacích jazykov, avšak stále nepodporuje niektoré rozšírené jazyky ako je napríklad PHP. Je však vytvorený v jazyku C, ktorý umožňuje omnoho rýchlejšie spracovávanie zdrojových kódov a určovanie podobnosti medzi nimi ako jazyk Python.

Výstupom nástroja MOSS je taktiež html súbor, ktorý zobrazuje perecentuálnu podobnosť medzi zdrojovými kódmi a kde sú taktiež zvýraznené konkrétne podobné časti kódov.

MOSS môže byť naviac integrovaný do externých systémov prostredníctvom svojho API.

Generovanie výstupu nástrojom MOSS prebieha tak, že najprv sa na server pošlú zdrojové kódy súborov, medzi ktorými chceme hľadať plagiát. Následne ich server spracuje a pošle späť url, na ktorej sa nachádzajú výsledky. Preto je aj použitie nástroja MOSS omnoho jednoduchšie ako nástroja vytvoreného v tejto bakalárskej práci a rovnako aj nástroja JPlag, keďže na celé vyhodnotenie nám stačilo stiahnuť skript a zadať cestu k súborom. Tento nástroj sa však neoplatí používať pre väčšie množstvo projektov, pretože čakanie na odpoveď servera je príliš dlhé.



Obrázek 7.4: Zobrazenie dvoch rovnakých súborov nástrojom MOSS

Kapitola 8

Záver

Cieľom tejto práce bolo navrhnúť a implementovať nástroj na vyhľadávanie plagiátov v zdrojových kódoch skriptovacích jazykov. Zadanie považujem za splnené, nástroj bol otestovaný a boli pomocou neho nájdené plagiáty v predmete Princípy programovacích jazykov a OOP.

V kapitole 4.1 sme si teda predstavili nástroj ANTLR a ďalšie nástroje na tokenizáciu zdrojového kódu. Druhý bod zadania bol popísať jednotlivé techniky plagiátorstva. Tie sme si popísali v kapitole 2, kde sme sa taktiež pozreli na hrozbu umelej inteligencie v tomto odvetví. Návrh nástroja bol popísaný v kapitole 5 a jeho implementácia v kapitole 6. Tu sme si detailnejšie predstavili jednotlivé zdrojové súbory a ich funkciu. V kapitole 7 sú predstavené výsledky testovania na projektoch k predmetu IPP a to v programovacích jazykoch PHP 8.1 a Python 3. V rovnakej kapitole sa taktiež nachádza porovnanie s už existujúcimi nástrojmi.

V kapitole 6 je taktiež v pár bodoch predstavený možný vývoj nástroja do budúcna, ktorý spočíva hlavne v rozšírení funkcionality, pridaním podpory pre viac jazykov, pridaním ďalších algoritmov na odhalovanie plagiátov alebo prepracovaním vizuálizácie rozdielu medzi zdrojovými kódmi.

Literatura

[1] Abdelhamid, M., Azouaou, F. a Batata, S. A Survey of Plagiarism Detection Systems: Case of Use with English, French and Arabic Languages. *CoRR*. 2022, abs/2201.03423. Dostupné z: https://arxiv.org/abs/2201.03423.

[2] Ahtiainen, A., Surakka, S. a Rahikainen, M. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. *ACM International Conference Proceeding Series*. Únor 2006, sv. 276. DOI: 10.1145/1315803.1315831.

[3] Aiken, A. *MOSS (Measure of Software Similarity)* [https://theory.stanford.edu/~aiken/moss/]. Accessed 2023-05-06.

[4] Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E. et al. *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. 2023.

[5] Dupuy, M. *Tokenize: Tokenizer for Python source* [online]. 2023 [cit. 2023-03-20]. Dostupné z: https://docs.python.org/3/library/tokenize.html.

[6] Gitchell, D. a Tran, N. Sim: a utility for detecting similarity in computer programs. In: *Technical Symposium on Computer Science Education*. 1999.

[7] Grune, D. a Huntjens, M. Het detecteren van kopie\(:en bij informatica-practica. *Informatie (in Dutch)*. Nov 1989, sv. 31, č. 11, s. 864–867.

[8] Hariprasad, T., Vidhyagaran, G., Seenu, K. a Thirumalai, C. Software complexity analysis using halstead metrics. In: *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. 2017, s. 1109–1113. DOI: 10.1109/ICOEI.2017.8300883.

[9] Joy, M. *Sherlock - Plagiarism Detection Software* [online]. 2014 [cit. 2023-03-20]. Dostupné z: https://warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock/.

[10] Krpec, O. *Rozpoznani plagiatu zdrojoveho kodu v jazyce PHP*. Brno, CZ, 2015. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/88521.

[11] Meo, S. A. a Talha, M. Turnitin: Is it a text matching or plagiarism detection tool? *Saudi Journal of Anaesthesia*. Apr 2019, sv. 13, Suppl 1, s. S48–S51. DOI: 10.4103/sja.SJA_772_18.

[12] Mrázek, V. *Nástroje pro pokročilou manipulaci s daty Pomocí knihovny Pandas.* 2022. Dostupné z: https://moodle.vut.cz/pluginfile.php/434073/course/section/41540/07-pandas/07-pandas.pdf.

[13] Parr, T. *ANTLR Reference Manual* [online]. University of San Franciso, march 2004 [cit. 2023-03-20]. Dostupné z: https://www.antlr3.org/share/1084743321127/ANTLR_Reference_Manual.pdf.

[14] Prechelt, L., Malpohl, G. a Phlippsen, M. *JPlag: Finding plagiarisms among a set of programs* [online]. Fakult at fur Informatik Universit at Karlsruhe D-76128 Karlsruhe, Germany, march 2000 [cit. 2023-03-19]. Dostupné z: https://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf.

[15] Salplachta, B. P. *Aplikace pro odhalovani plagiatu.* Brno, CZ, 2009. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis-file/8910/8910.pdf.

[16] Sağlam, T. *JPlag Wiki* [online]. 2022 [cit. 2023-03-19]. Dostupné z: https://github.com/jplag/JPlag/wiki.

[17] Schleimer, S., Wilkerson, D. S. a Aiken, A. *Winnowing: Local Algorithms for Document Fingerprinting* [online]. March 2003 [cit. 2023-03-19]. Dostupné z: https://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf.

[18] Stepchyshyn, V. a Nelson, R. S. *Library plagiarism policies.* Assoc. of College & Resrch Libraries, 2007. 65 s. ISBN 978-0-8389-8416-1.

[19] Sundar, K. *A beginners guide to Multi-Processing in Python* [online]. 2021 [cit. 2023-03-19]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/04/a-beginners-guide-to-multi-processing-in-python/.

[20] Sági Kazár, M. *PHP serialization benchmarks* [https://github.com/sagikazarmark/php-serialization-bench]. 2017.

[21] Wise, M. J. *String Similarity via Greedy String Tiling and Running Karp Rabin Matching* [online]. 1993 [cit. 2023-03-19]. Dostupné z: https://www.researchgate.net/profile/Michael_Wise/publication/262763983_String_Similarity_via_Greedy_String_Tiling_and_Running_Karp-Rabin_Matching/links/59f03226aca272a2500141f4/String-Similarity-via-Greedy-String-Tiling-and-Running-Karp-Rabin-Matching.pdf.

[22] Xiao, Y., Chatterjee, S. a Gehringer, E. A New Era of Plagiarism the Danger of Cheating Using AI. In: *2022 20th International Conference on Information Technology Based Higher Education and Training (ITHET).* 2022, s. 1–6. DOI: 10.1109/ITHET56107.2022.10031827.

Příloha A

Manuál k programu

Stiahnutie potrebných knižníc

```
pip install python-Levenshtein scikit-learn setuptools pandas matplotlib
msgpack
```

Popis stiahnutých knižníc

`python-Levenshtein`	Knižnica, z ktorej sa používa funkcia distance v implementácii Levenshteinovej vzdialenosti
`scikit-learn`	Knižnica, z ktorej je použitá implementácia TF–IDF algoritmu
`setuptools`	Knižnica, pomocou ktorej sa prekladá C modul
`pandas`	Knižnica používaná v piatej fáze na spracovanie CSV súborov
`matplotlib`	Knižnica používaná v piatej fáze na generovanie grafov
`msgpack`	Knižnica, v ktorej je definovný formát na ukladanie dát

Preloženie Python API kódu napísaného v C

```
python3.10 setup.py build_ext --inplace
```

Príklady použitia

Všobecný:
python3.10 controller.py --input INPUT_FILE --PROGRAMMING_LANGUAGE --METRICS

Na priložených testovacích dátach:
python3.10 controller.py --input ../test/PHP/ --php --halstead --levenshtein
python3.10 controller.py --input ../test/Python/ --python --levenshtein

Vstupné parametre

`-h, --help`	zobrazí manuál programu
`--first`	zapne iba prvú fázu programu
`--second`	zapne iba druhú fázu programu
`--third`	zapne iba tretiu fázu programu
`--fourth`	zapne iba štvrtú fázu programu
`--fifth`	zapne iba piatu fázu programu
`--python`	porovnávanie python zdrojových súborov
`--php`	porovnávanie php zdrojových súborov
`--input INPUT`	Adresár s novými projektmi na porovnávanie
`--oldinput OLDINPUT`	Adresár so starými projetmi na porovnávanie
`--common COMMON`	adresár so súbormi, ktoré sa neporovnávajú
`--phase1_data_out NAME`	názov súboru z prvej fázy s dátami (**phase1_data_out**)
`--phase1_name_out NAME`	názov súboru z prvej fázy s názvom projektov (**phase1_name_out**)
`--phase1_token_out NAME`	názov súboru z prvej fázy s tokenmy súborov (**phase1_token_out**)
`--phase2_out NAME`	názov výstupného súboru z druhej fázy (**phase2_out**)
`--phase3_out NAME`	názov výstupného súboru z tretej fázy (**phase3_out**)
`--phase4_out NAME`	názov výstupného súboru zo štvrtej fázy (**phase4_out**)
`--phase4_in NAME`	názov vstupného súboru do štvrtej fázy (**phase4_in**)
`--phase5_out NAME`	názov výstupného súboru z piatej fázy (**phase5_out**)
`--output_folder NAME`	adresár na ukladanie výstupov (**súbor so zdrojovými kódmi**)
`--proc PROC`	počet procesov, na ktorých beží porovnávanie (**počet CPU**)
`--resume`	pokračuje vo vykonávaní, ak existujú potrebné súbory
`-n N`	maximálny počet párov spracovávaných v 4. fáze (**50**)
`--levenshtein`	použije levenshteinovu metriku v 3. fáze
`--halstead`	použije halsteadovu metriku v 3. fáze
`--comments`	použije metriku porovnávania komentárov v 3. fáze

Příloha B

Obsah odovzdaného média