TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Kevin Kaar 143055 IAPB

JAVA-SPETSIIFILINE PLAGIAADITUVASTUS BAITKOODI PÕHJAL

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kevin Kaar

21.05.2017

Annotatsioon

Antud töö raames analüüsitakse, kas ja kuidas oleks võimalik kasutada Java baitkoodi plaagiaadi tuvastamiseks ning kuidas erineks antud lahendus hetkel kasutuselolevatest. Töö koosneb kasutuselolevate ja baitkoodi põhjal loodavast plagiaadituvastussüsteemide analüüsist, nende omavahelisest võrdlusest ja viimase realisatsioonist, olenemata, kas omavahel võrreldavad tööd on kättesaadavad lähte- või juba kompileeritud koodina. Realiseeritakse prototüüp, mis omab võimalust täiendamiseks ja edasi arendamiseks. Kogu koodibaas on avalikustatud Githubis aadressil https://github.com/kevinkaar/bytecode-plagiarism-detection.git.

Lõputöö on kirjutatud eesti keeles ning koosneb 44 leheküljest, 7 peatükist, 22 joonisest, 3 tabelist.

Abstract

Java-specific plagiarism detection using bytecode

The purpose of this thesis is to anaylse whether and how to use Java bytecode to detect plagiarism and how would it differ from current solutions. The thesis consists of three parts: analysis of existing and Java bytecode plagiarism detection systems, comparison of those two and realization, no matter whether the accessible file is source- or already compiled code. The realization of plagiarism detection using bytecode is not fully implemented and there is still room for development. Whole codebase is publicly available in Github at https://github.com/kevinkaar/bytecode-plagiarism-detection.git.´

The thesis is written in Estonian and consists of 44 pages, 7 chapters, 22 figures, 3 tables.

Lühendite ja mõistete sõnastik

All-to-all	Kattuvuste võrdlemiseks võrreldakse kõiki meetodeid kõigi teises klassis olevate meetoditega
Git	Versioonihaldussüsteem, peamiselt kasutuses programmeerijate poolt
Github	„Keskus“, erinevate giti repositooriumidele, kus saab hoida erinevaid projekte
Java	Programmeerimiskeel, mille baitkoodi põhjal antud töö käigus plagiaati tuvastatakse
Java Baitkood	Instruktsioonid Java virtuaalmasinale programmi käivitamiseks, asub failis laiendiga .class [1]
Java Language Specification	Ametlik Java programmeerimiskeele spetsifikatsioon, mis kirjeldab keelt ja selle eripärasid
Java virtuaalmasin (Java Virtual Machine)	Võimaldab käivitada Java programme [2]
Javac	Java Compiler - java kompilaator, mis kompileerib lähtekoodist baitkoodi, et JVM koodist käivitada oskaks [3]
Javap	Java Print – programmm mis võimaldab kompileeritud Java .class laiendiga faili lugeda ja kuvada sealt vastava faili baitkoodi [4]
Jplag	Populaarne plagiaadituvastussüsteem [5]
Karp-Rabin Algorithm	Sõnede kokkulangemist kontrolliv algoritm, koostatud Michael O.Rabin’i ja Richard M.Karp poolt aastal 1987 [6]
K-gram	Paljude plagiaadituvastussüsteemide, nt. MOSS esimene samm, kus jagatakse dokumendi sisu fikseeritud pikkusega alamsõnedeks (k-gram’ideks) [7]
Kuhi	Inglise keeles stack – arvutiteadustes kasutatav andmetüüp, mis hoiab endas erinevate elementide kollektsiooni. 2 peamist operatsiooni: *push* (lisab elemendi kollektsiooni) ja *pop* (võtab elemendi kollektsioonist). Toimub *LIFO* (*Last In First Out*) põhimõttel – mis viimati lisati, tuleb esimesena välja
Käsurida	Programm operatsioonisüsteemis, mille abil on võimalik käivitada teisi süsteemis olevaid programme
Lähtekood	Kood, mida arendaja kirjutab programme luues
Mappima	Koodi ümber *mappima* – viima koodi ühelt kujult teisele, et seda oleks lihtsam lugeda või edaspidi kasutada
MOSS	*Measure Of Software Similarity* – populaarne plagiaadituvastussüsteem

Plagiaat	Võõra teose või selle osade avaldamine enda omadena [8]
Sõrmejälg	Plagiaadituvastussüsteemides väljavalitud räsiväärtused mille abil tuvastada plagiaati
UI	User Interface – liides, mille teel toimub kasutaja ja süsteemi vaheline suhtlus
Voog	Inglise keeles *stream* - sisaldab järjestikke objekte (enamasti baite), millele saab ligi pääseda järjestikuliselt [9]
Winnowing	Algoritm leidmaks efektiivsemalt dokumendist sõrmejälgi, et tuvastada plagiarismi [10]

Sisukord

1 Sissejuhatus 10
2 Plagiaadituvastus praegu 12
2.1 Üldine põhimõte 12
2.1.1 Sisu jagamine k-gram'ideks 12
2.1.2 K-gram'idel räsifunktsiooni kasutamine 12
2.1.3 Sõrmejälgede valimine 13
2.2 Teisi plagiaadituvastus tehnikaid 14
2.3 Kokkuvõte 14
3 Java baitkood 15
3.1 Lähtekoodist baitkood 15
3.2 Baitkood 15
4 Plagiaadituvastus baitkoodi põhjal ning selle eelised ja puudused 21
4.1 Plagiaadituvastus baitkoodi põhjal 21
4.2 Baitkoodi puudused plagiaadituvastamisel 22
4.3 Baitkoodi eelised plagiaadituvastamisel 22
5 Realisatsioon 24
5.1 Tehnoloogia valik 24
5.2 Realisatsioon 24
5.2.1 Realisatsioon 1 – muudetud muutujanimed 25
5.2.2 Realisatsioon 2 – muudetud instruktsioonide järjekord 27
5.2.3 Realisatsioon 3 – üleliigne kood 28
6 Valideerimine tegelike tudengite tööde põhjal 31
6.1 Katse 1 – Võrdluse all MOSS plagiaadituvastussüsteemi põhjal 84% kattuvusega klassifailid 31
6.2 Katse 2 – Võrdluse all MOSS plagiaadituvastussüsteemi põhjal 77% kattuvusega klassifailid näide 1 33
6.3 Katse 3 - Võrdluse all MOSS plagiaadituvastussüsteemi põhjal 77% kattuvusega klassifailid näide 2 34
6.4 Kokkuvõte 35
7 Kokkuvõte 37

Jooniste loetelu

Joonis 1: POJO Human 18
Joonis 2: POJO Human baitkood 19
Joonis 3: Realisatsiooni sarnaste lähtefailide näide 1 25
Joonis 4: Realisatsiooni sarnaste lähtefailide näide 2 25
Joonis 5: Realisatsiooni sarnaste failide näite CSV baitkood 26
Joonis 6: Realisatsiooni sarnaste failide näite tulemus 26
Joonis 7: Realisatsiooni erinevate järjekordade näide 1 27
Joonis 8: Realisatsiooni erinevate järjekordade näide 2 27
Joonis 9: Realisatsiooni erinevate järjekordade näite baitkood CSV 28
Joonis 10: Realisatsiooni erinevate järjekordade näite tulemus 28
Joonis 11: Realisatsiooni erinevate järjekordade näite CSV sorteeritud 28
Joonis 12: Realisatsiooni erinevate järjekordade näite tulemus sorteeritud 28
Joonis 13: Realisatsiooni üleliigse koodi näide 1 29
Joonis 14: Realisatsiooni üleliigse koodi näide 2 29
Joonis 15: Realisatsiooni üleliigse koodi näite CSV baitkood 29
Joonis 16: Realisatsiooni üleliigse koodi näite tulemus 30
Joonis 17: Reaalsed tudengite tööd katse 1 töö 1 32
Joonis 18: Reaalsed tudengite tööd katse 1 töö 2 32
Joonis 19: Reaalsed tudengite tööd katse 2 töö 1 33
Joonis 20: Reaalsed tudengite tööd katse 2 töö 2 34
Joonis 21: Reaalsed tudengite tööd katse 3 töö 1 34
Joonis 22: Reaalsed tudengite tööd katse 3 töö 2 35

Tabelite loetelu

Tabel 1: Baitkoodi instruktsioonid [5], [13] 16
Tabel 2: Baitkoodi eesliited [5] 17
Tabel 3: Baitkoodi järelliited [13] 17

1 Sissejuhatus

Ajastul, kus praktiliselt kõik on kõigile internetis kättesaadav, on väga raske vahet teha, kas väidetav autor on ka tegelik autor või on materjal võetud hoopis kuskilt mujalt ja lihtsalt enda nimi alla kirjutatud. Eelnev on ka väga levinud probleem erinevates ülikoolides – kuidas peaks õppejõud saama aru, kas tudeng on ise oma koduse ülesande lahendanud, või hoopiski kelleltki kopeerinud ja enda nime alt esitanud?

Autor, olles ise juba kolmandat aastat tudeng Tallinna Tehnikaülikoolis, on märganud, et tihtipeale leiavad aset olukorrad, kus tudeng A küsib tudengilt B õppematerjale selleks, et vaadata, kuidas on antud kodutöö tehtud ja sealt õppida. Tudeng B pahaaimamata jagab oma tööd, mille tudeng A väikeste muudatustega õppejõule edastab.

Tänapäeval on laialt levinud üpriski vanad programmikoodi plagiaadituvastussüsteemid, nt TTÜ's kasutusel olevad MOSS[1] – 1994 ja Jplag[2] – 1996, mis töötavad k-gram'ide toel teksti *hashimise* teel. Lisaks k-gram'idele on kasutusel ka teisi tehnikaid, kus „sõrmejäljena" ei kasutata mitte k-grame vaid hoopiski fikseeritud pikkusega sõnesid, mis algavad kindlate „võtmesõnadega" [10].

Antud töö eesmärgiks on luua plagiaadituvastussüsteem, mis võimaldaks ära tunda sarnast koodi programmeerimiskeele Java raames, seejuures lähenedes plagiaadi tuvastamisele veidi teise nurga alt, kui seda hetkel tehakse. Antud töös plagiaadi tuvastamiseks kasutatakse Java-spetsiifilist lähenemist, kasutades Java baitkoodi, seda vajadusel ka sorteerides.

Autor on arvamusel, et selline lähenemine võib kasuks tulla, sest kui varasemate plagiaadituvastussüsteemide puhul võib tulemus muutuda ainult „naiivsete" muudatustega Java koodis nt. muutujanimede muutmine, siis baitkoodi puhul nii lihtsalt

[1] https://theory.stanford.edu/~aiken/moss/
[2] https://jplag.ipd.kit.edu/

ei lähe – muutujanimede muutmisel baitkood ei muutu. Selle tulemusel on võimalik tuvastada eriti naiivselt esitatud „kopeeritud ja kleebitud“ tööd.

Prototüüp realiseeritakse täiesti nullist, käsurea rakendusena, ja on suuresti eksperimentaalne tegevus, teadmata, kas eelnimetatud süsteem reaalselt ka positiivseid tulemusi toob.

Prototüübi realiseerimisel on autor seadnud endale eesmärgid:

- realiseerida prototüüp, mis võimaldaks tuvastada naiivselt kopeeritud ja kleebitud koodi Java baitkoodi põhjal;
- prototüüp peab töötama kasutades sisendina nii Java lähtekoodi kui ka kompileeritud koodi.

2 Plagiaadituvastus praegu

Programmikoodi plagiaadituvastus praegu on enamikel süsteemidel üpriski sarnane. Järgnevalt tutvustame plagiaadituvastuse üldisi põhimõtteid, mille alusel töötab näiteks MOSS, väikeste muudatustega *winnowing* algoritmis [10].

2.1 Üldine põhimõte

Üks varasemaid algoritme, mis avastati taolisel viisil töötavat on *Karp-Robini* sõnede kokkulangevuse tuvastamise algoritm, kui sõnedest hakati efektiivsema saranasuse tuvastamiseks räsiväärtusi arvutama [6], [10].

2.1.1 Sisu jagamine k-gram'ideks

Esimene samm plagiaadituvastamiseks on dokumendi sisu jagamine fikseeritud pikkusega tekstis järjest tulevateks alamsõnedeks (edaspidi *k-gram*), kus k, ehk alamsõne pikkus on kasutaja poolt vabalt valitav [10]. Enne teksti juppideks jagamist kaotatakse tekstist üleliigsed sümbolid, (nt tühikud, punktid, komad jms) ja asendatakse suured tähed väikestega. *K-gram'e* tuleb peaaegu sama palju kui on olemasolevas tekstis tähti (v.a viimased k-1 tähte).

Näiteks, kui tekst on „*Hello world!*“, millest asendades ebavajalikud sümbolid saame „*helloworld*“ ja *k-grami* pikkus oleks näiteks 5 (k = 5), siis saame kõikideks võimalikeks 5-gram'deks: „*hello, ellow, llowo, lowor, oworl, world*“.

2.1.2 K-gram'idel räsifunktsiooni kasutamine

Iga eelnevas sammus leitud *k-grami* peal kasutatakse räsifunktsiooni ja arvutatakse vastava k-grami räsi.

Näiteks eelneva „*Hello world*!“ näite puhul võiks olla tulemuseks „32, 45, 23, 18, 64, 78“

2.1.3 Sõrmejälgede valimine

Järgmiseks sammuks on eelnevalt arvutatud räsidest välja valida dokumendi sõrmejäjed. Enamasti valitakse selleks vaid väike osa kõikidest *k-gramidest*, kuna *k-gram'e* on pikema teksti puhul väga palju. Üheks populaarseks viisiks on valida kõik räsid, mis on *0 mod p*, kus p on fikseeritud suurusega number.

Näiteks *0 mod 4*, kus siis p = 4 annaks eelneva näite puhul sõrmejäljeks „32, 64“.

Eelnevalt nimetatud sõrmejälgede leidmise viisil on siiski ka negatiivseid külgi. Nimelt selline meetod ei garanteeri, et kokkulangemisi dokumentide vahel üldse leitaks, kuna kokkulangev *k-gram* dokumentide vahel avastatakse ainult olukordades, kus vastav räsi on *0 mod p*. Niisiis kõik kokkulangevused, mille räsi väärtus ei ole *0 mod p* jäävad leidmata.

Eelneva probleemi lahendamiseks on välja töötatud *winnowing* algoritm, mille kohaselt luuakse arvutatud räsidest fikseeritud pikkusega (pikkus on kasutaja poolt valitud) „aknad“, kust igast „aknast“ valitakse minimaalne räsiväärtus, mis ei ole veel sõrmejäljena salvestatud. Kui minimaalseid räsiväärtusi on rohkem kui üks, siis salvestatakse sõrmejäljena parempoolne.

Tuues näiteks peatükis 2.1.2 „*Hello world!*“ näite puhul arvutatud *k-gram* räsi väärtused „32, 45, 23, 18, 64, 78“, ja aknad pikkusega kolm, saame akendeks: (32, 45, **23**), (45, 23, **18**), (23, 18, 64), (18, 64, 78), kust *winnowing* algoritmiga saaksime sõrmejälgedeks „23 18“.

Tõhus *winnowing* algoritm ei salvesta mitte ainult sõrmejälgi, vaid ka positsiooni, juhuks, kui hiljem peaks olema vajadus tuvastada kokkulangemiste asukohta. Hea näide olukorrast, kus seda vaja minna võiks oleks olukord, kus tahetakse kasutajale ka *UI*'s näidata, kus täpsemalt kokkulangemine tekkis.

Näiteks salvestusviis [sõrmejälg, positsioon] autori näite puhul oleks [23, 2], [18, 3].

2.2 Teisi plagiaadituvastus tehnikaid

K-gram'ide teel sõrmejälgede leidmine ei ole olnud ainuke viis plagiaadituvastuseks. Sõnesi sõrmejälgedeks võib valida ka lausete, lõikude või fikseeritud pikkusega sõnede hulgast, mis algavad mingite kindlate „võtmesõnadega". Sellisel lähenemisel aga on omad probleemid – iga implementatsioon on suuresti spetsiifiline ühe andmetüübi suhtes. Näiteks kui keskendutakse ingliskeelsele tekstile ja sõrmejälgede leidmiseks kasutatakse erinevaid lauseid, siis seda sama süsteemi kasutada sõrmejälgede leidmiseks mõnest programmeerimiskeelest, mis inglise keelega ei sarnane absoluutselt, (nt *Prolog*[1] ja *C*[2]) osutuks väga keeruliseks. Isegi kui proovitakse sõrmejälgi leida kahest tekstilisest andmefailist, võib mõistliku lause leidmine sõrmejäljeks olla kaheldav, kuna andmefail võib vabalt koosneda näiteks tabelitest [10].

2.3 Kokkuvõte

Olemasolevad enim kasutuselolevad ja tõhusaimad plagiaadituvastussüsteemid kasutavad sisendite kattumiste leidmiseks *k-gram'e*, mis leiab kattumisi ning töötab väga hästi. Üheks suureks boonuseks *k-gram*'idel on see, et selle abil saab protsessida igasuguseid erinevaid andmetüüpe, alustades tavalise tekstiga, lõpetades näiteks erinevate programmeerimiskeelte koodidega. Kuid lisaks *k-gram'idele* on siiski vaja kasutada ka mingisugust muud meetodit, et aru saada, mis ja kus täpsemalt kattus, näiteks *winnowing* algoritm.

Võttes arvesse eelnevalt nimetatud asjaolusid, on näha, et praegune metoodika plagiaadi tuvastamiseks on tõhus ja universaalne erinevate andmetüüpide lõikes. Kuid kui vaadata näiteks ühte kindlat andmetüüpi, Java koodi ja selle kompileerimisel tulenevat baitkoodi, tekib küsimus, et äkki on viimase abil, spetsiaalselt Java raames, võimalik plagiaadituvastust jooksutada tõhusamalt ja optimaalsemalt.

[1] http://groups.engin.umd.umich.edu/CIS/course.des/cis400/prolog/prolog.html
[2] https://www.tutorialspoint.com/cprogramming/c_overview.htm

3 Java baitkood

Järgnevalt ülevaade Java baitkoodist – kuidas saada lähtekoodist baitkood, mis on Java baitkood ning selle omapärad.

3.1 Lähtekoodist baitkood

Et saada kasutaja poolt kirjutatud koodist baitkood, on vaja lähtekoodi failid (laiendiga *.java*[1]) kompileerida baitkoodi failideks (laiendiga *.class*[2]). Class laiendiga failid sisaldavadki kompilaatori poolt koostatud baitkoodi instruktsioone *JVM'le*, mis on saadud eelnevalt lähtekoodi kompileerides. Lähtekoodi kompileerimiseks baitkoodi kasutatakse programmi *javac*[3].

3.2 Baitkood

Java baitkood on tavaline Java lähtekood, mis on kompileeritud „masinkeelde", et *Java Virtual Machine*[4] (edaspidi *JVM*) seda käivitada oskaks. Kui *JVM* laeb class faili, loeb viimane sisse ühe voo (*streami*) baitkoode (järjestikuseid instruktsioone) iga meetodi kohta, mis on kompileeritud klassis. Meetodi baitkood käivitatakse alles siis, kui vastav meetod on programmi töö jooksul välja kutsutud [11].

Kõik arvutused *JVM's* toimuvad kuhjas (*stack*[5]) ja seetõttu tuleb kõik lükata kuhja, enne kui saab teha erinevaid arvutusi. Seetõttu opereerivad ka baitkoodi instruktsioonid peamiselt kasutades kuhja.

[1] https://fileinfo.com/extension/java
[2] https://fileinfo.com/extension/class
[3] https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/tooldocs/win32/javac.html
[4] https://www.javatpoint.com/internal-details-of-jvm
[5] https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

Baitkoodi instruktsioonid jagunevad mitmesse erinevasse gruppi, millest mõningad on järgnevalt koos seletustega välja toodud:

Tabel 1: Baitkoodi instruktsioonid [1], [12]

Grupp	**Näide baitkoodi instruktsioonist**	**Kirjeldus**
Muutujate laadimine ja salvestamine	*aload*_0, *istore*	• *aload*_0 – laeb viite lokaalsete muutujate tabelist indexiga 0 kuhja. • *istore* – salvestab integer väärtus muutujasse
Aritmeetika ja loogika	*ladd, fcmpl*	• *ladd* – liidab 2 long tüüpi väärtust • *fcmpl* – võrdleb 2 float tüüpi väärtust
Tüübiteisendused	*i2b, d2i*	• *i2b* – teisendab int tüüpi väärtuse byteks • *d2i* – teisendab double tüüpi väärtuse integeriks
Objektide loomine ja manipulatsioon	*new, putfield*	• *new* – loob uue objekt • *putfield* – omistab objekti väljale uue väärtuse
Kuhjas opereerimised	*swap, dup2*	• *swap* – vahetab 2 pealmist väärtust kuhjas • *dup2* – duplikeerib 2 pealmist kuhja väärtust
Instruktsioonide ümber juhtimised	*ifeq, goto*	• *ifeq* – tingimuslause, kui väärtus 0 siis *true*, muidu *false* • *goto* – instruktsioonide suunamine teise instruktsiooni juurde (näiteks continue java koodis)
Meetodite väljakutsed ja tagastamised	*invokespecial, areturn*	• *invokespecial* – kutsutakse objektis meetodit välja, tulemus salvestatakse kuhja • *areturn* – tagastab viite meetodist

Täpsemalt ja rohkem instruktsioone koos kirjeldustega siin [12].

Nagu eelmisest näitest näha võis, on paljudel instruktsioonidel ees- ja järelliited, vastavalt sellele, mis andmetüüpi muutujate kallal parasjagu opereeritakse. Näiteks *iadd* liidab kaks täisväärtust(*int*) tüüpi arvu, aga *fadd* liidab kaks ujukomaarvu(*float*).

Toon välja baitkoodis esinevad ees- ja järelliited:

Tabel 2: Baitkoodi eesliited [1]

Eesliide	**Kirjeldus**
i	*integer* (32-bitine täisarv)
l	*long* (64-bitine täisarv)
s	*short* (16-bitine täisarv)
b	*byte* (bait)
c	*character* (sümbol)
f	*float* (ujukomaarv[1] 32bit)
d	*double* (ujukomaarv 64bit)
z	*boolean* (tõeväärtus)
a	viide objektile

Tabel 3: Baitkoodi järelliited [12]

Järelliide	**Näide**	**Kirjeldus**
_0	*aload*_0	Laeb viite lokaalsele muutujale indexiga 0 kuhja
_1	*aload*_1	Laeb viite lokaalsele muutujale indexiga 1 kuhja

Järelliiteid on veel, näiteks load ja store instruktsioonide puhul „_2“, „_3“ ja konstantide puhul „_n“ järelliite n maksimaalne väärtus erineb olenevalt tüübist [1].

Nüüd, kus üldine arusaam baitkoodist ja selle instruktsioonidest on olemas, järgneb väike näide reaalse koodi ja baitkoodi põhjal.

Eeldame, et meil on üks lihtne *POJO*[1], *Human*, millel on ainult üks väli – selle inimese pikkus, ehk *length*. Sellele väljale ligi pääsemiseks ja manipuleerimiseks on objektile

[1] https://et.wikipedia.org/wiki/Ujukomaarv

lisatud ka meetodid *getLength*() ja *setLength*() (edaspidi *getter* ja *setter*[2]). Eelnevalt nimetatud *POJO* Java koodis näeks välja järgnev:

Joonis 1: POJO Human

```
public class Human {

    private long length;

    public long getLength() { return length; }

    public void setLength(long length) { this.length = length; }
}
```

Käivitades käsurealt käsklus „*javac Human.java*“ kompileeritakse vastav klass ja selle tulemusena pannakse meile kokku kompileeritud kujul *Human.java* fail – *Human.class*, mis sisaldab nüüd baitkoodi.

Käskluse kirjeldus:

- *javac* – käsurida tunneb selle järgi ära, et tahame kasutada *Java compiler* programmi
- *Human.java* – faili nimi, mida tahame kompileerida

Et näha loetaval kujul kompileeritud *Human.class* faili sisu (baitkoodides) käivitame käsurealt käskluse „*javap –c Human.class“*.

Käskluse kirjeldus:

- *javap* – käsurida tunneb selle järgi ära, et tahame kasutada Java Print programmi;
- -c – sellega ütleme käsureale, et soovime kuvada koodi „lahti võetuna“, baitkoodi instruktsioonidena;
- *Human.class* – jällegi vastava faili nimi, mille sisu soovime kuvada.

[1] Plain Old Java Object – Tavaline Java koodis paikenv objekt, mis ei sisalda koodi erinevatelt raamistikelt – https://spring.io/understanding/POJO

[2] https://en.wikipedia.org/wiki/Property_(programming)

Javap käskluse tulemusena väljastatakse meile järgnev:

Joonis 2: POJO Human baitkood

```
Compiled from "Human.java"
public class Human {
  public Human();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public long getLength();
    Code:
       0: aload_0
       1: getfield      #2                  // Field length:J
       4: lreturn

  public void setLength(long);
    Code:
       0: aload_0
       1: lload_1
       2: putfield      #2                  // Field length:J
       5: return
}
```

Esimese asjana jääb baitkoodist silma, et baitkoodis on näha vaikekonstruktor[1], mida koodis justkui ei ole. Nagu on kirjutatud *Java Language Specification*'is [13] – kui klassil ei ole ühtegi konstruktori deklaratsiooni, siis luuakse automaatselt *default constructor* ilma parameetrite ja *throws*[2] kirjeldusteta.

Analüüsides toimuvat näiteks meetodis *getLength()* on näha, et meetodil on kolm baitkoodi instruktsiooni:

- *aload*_0 – lükkab väärtuse lokaalsete muutujate tabelist, kohal indexiga 0, kuhja. Kohal indexiga 0 konstruktoritel ja objekti meetoditel on alati *this*[3] viide;
- *getfield* – saab objektilt välja väärtuse;
- *lreturn* – tagastab väärtuse meetodist, kusjuures „l" returni ees tähistab seda, et tagastatav väärtus on *long* tüüpi, nagu eelnevalt mainitud.

Eelnevast näitest on näha, et Java koodi kompileerides saame hästi defineeritud baitkoodi, mida autori arvates võiks kasutada ka plagiaadi tuvastamiseks Java koodis.

[1] http://www.rapidprogramming.com/questions-answers/What-is-a-default-constructor-in-Object-Oriented-Programming--826

[2] Käsklus Javas, mis võimaldab erindeid edasi välja kutsuvale meetodile visata

[3] https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

Nagu enamus asjadel, on ka baitkoodi põhisel tuvastusel omad puudused ja eelised praeguste plagiaadituvastussüsteemide ees.

4 Plagiaadituvastus baitkoodi põhjal ning selle eelised ja puudused

Järgnevalt selgitab autor prototüübis loodava plagiaadituvastusalgoritmi tööpõhimõtteid ning toob välja mõningad puudused ja eelised baitkoodi põhjal plagiaadi tuvastamisel, mis on siis kasuks või hoopiski kahjuks, tuvastamaks Java koodist plagiaati.

4.1 Plagiaadituvastus baitkoodi põhjal

Baitkoodi põhjal plagiaadituvastuse realiseerimiseks on autor mõelnud kasutada järgnevat loogikat.

- Omavahel võrreldakse alati kahte klassi, *all-to-all* põhimõttel, st et ühe klassi igat meetodit võrreldakse iga teise klassi meetodiga.
- Omavahelisel võrdlusel ei kasutata vaikekonstruktoreid ja tavalisi *gettereid* ning *settereid*, et vältida valesid tulemusi.
- Klassi meetoditest on loodud baitkoodi põhjal andmestik, kus iga meetod on eraldiseisev.
- Olles kätte saanud iga meetodi kohta vajamineva andmestiku, võrreldakse andmestikke järgneval kolmel põhimõttel:
    - võrreldakse omavahel järjestikuste baitkoodi instruktsioonide kattumist, kui kattumine on suur, on oht, et on avastatud plagiaat;
    - võrreldakse omavahel tähestikuliselt sorteeritud baitkoodide instruktsioonide kattumist – sellega on võimalik välistada näiteks tehete järjekord koodis, kuid sorteerimine võib siiski osutada ka valeinfot lõpptulemuses;
    - vaadatakse, milline on väiksema instruktsioonide arvuga meetodi kattuvuse suhe suurema suhtes – sellega on võimalik vältida olukorda,

kus meetoditesse on lisatud asjatut koodi, mis muudel juhtudel võiks tekitada valeinfot.

- Võttes arvesse eelnevaid näitajaid peaks olema võimalik tuvastada koodides vähemalt naiivselt kopeeritud ja kleebitud ning seejärel esitatud tööd.

4.2 Baitkoodi puudused plagiaadituvastamisel

- Kuna tegemist on tervete meetodite omavaheliste kattumiste otsimisega, on keeruline hiljem tuvastada, millised osad meetoditest omavahel täpsemalt kattusid.
- Üleliigse koodi lisamine meetoditesse lisab baitkoodi instruktsioone, mis võivad kattumise otsimisel tulemust mõjutada.
- Kuna baitkoodist ei ole kuidagi näha, mis meetodeid / klasse konkreetse töö tegemiseks kasutati, võib tekkida olukordi, kus totaalselt erinevad meetodid (mis teevad isegi totaalselt erinevat asja) võivad baitkoodi puhul kattuda ja osutuda plagiaadiks.
- Koodis näiteks liitmistehte järjekorra muutmine muudab baitkoodi instruktsioonide järjekorda, mis samuti võib mõjutada lõpptulemust.
- Erinevad *getterid*, *setterid* ja *konstruktorid*, millel on praktiliselt identsed instruktsioonid, võivad näidata erinevaid kokkusattuvusi kohtades, kus neid tegelikult ei tohiks töö raames arvesse võtta.

4.3 Baitkoodi eelised plagiaadituvastamisel

- Muutujanimede muutmine ei muuda baitkoodi instruktsioone – see on väga oluline boonus baitkoodi puhul, mis võimaldab tuvastada just sellist „naiivset kopeeritud-kleebitud“ koodi, kus tudeng ei ole ilmselgelt ise mitte midagi teha proovinud.
- Plagiaadi tuvastamiseks ei ole vaja ligipääsu lähtekoodile – piisab juba kompileeritud koodist.
- Java baitkood on juba olemuselt hästi defineeritud kujul, mis annab lootuse, et ehk ei pea räsifunktsioone koodi peal kasutama.

- Sõrmejälgede räsiväärtuste põhjal plagiaadituvastamise mõte on see, et plagiaat võib esineda ükskõik millises programmi osas ja plagieeritud võib olla ainult üks meetod klassis. Kui baitkoodi põhjal teha aga igast meetodist üks andmerida, saavutame loodetavasti vähema pingutusega parema tulemuse kui räsiväärtuste teel – leiame üles ümber tõstetud meetodid ilma *winnowing* algoritmi ja räsideta.

Nagu eelnevast loetelust näha, on baitkoodi kasutamisel plagiaadituvastamiseks nii positiivseid kui ka negatiivseid külgi, mida tuleb realisatsiooni käigus arvesse võtta ja võimalikult palju negatiivseid punkte lahenduses elimineerida.

5 Realisatsioon

Selles peatükis tuuakse välja eelnevalt välja toodud kirjelduse põhjal plagiaadituvastussüsteemi prototüübi realisatsioon ja peatükis 4.2 välja toodud baitkoodi negatiivsete külgede elimineerimine kattuvuste leidmisel.

5.1 Tehnoloogia valik

Antud töö käigus kasutuselolev tehnoloogia on programmeerimiskeel Java, sest autor tunneb ennast selles kõige kogenenuma ja kindlamana.

5.2 Realisatsioon

Prototüübi realiseerimist alustasin kõigepealt uurimisega, kuidas oleks kõige lihtsam ja mugavam jooksutada väliseid käsurea protsesse Java koodis ja kätte saada lähtekoodist baitkood. Samuti tuli arvestada sellega, et baitkood oleks hiljem lihtne ümber *mappida* Java objektideks, et mugavalt viimaseid koodis kasutada. Väikese analüüsi ja uurimise teel avastasin, et võimalusi on üpris palju, näiteks *ProcessBuilder*[1], *Apache Commons Exec*[2] ja ühe eesti startup'i[3], *ZeroTurnaround'i*[4], vabavaraliseks kasutamiseks loodud projekt nimega *zt-exec*[5]. Prototüübi realiseerimisel otsustasin kasutada viimast, *zt-exec'it*, peamiselt hea ja lihtsalt arusaadava *Github'i* dokumentatsiooni tõttu.

Olles leidnud tööks vajalikud vahendid, sai autor hakata prototüüpi ehitama, mis esialgu töötaks lihtsamate stsenaariumide korral. Sarnasuste leidmisel kasutatakse meetodite puhul *all-to-all* võrdlust otse baitkoodi põhjal, ilma instruktsioonide peal

[1] https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html
[2] https://commons.apache.org/proper/commons-exec/
[3] https://www.forbes.com/sites/natalierobehmed/2013/12/16/what-is-a-startup/#1e89452d4044
[4] https://zeroturnaround.com/
[5] https://github.com/zeroturnaround/zt-exec

räsifunktsioone kasutamata. Sarnasuse tuvastamiseks arvutab programm järjestike baitkoodi instruktsioonide kattuvuse suhte kogu instruktsioonide hulgast.

5.2.1 Realisatsioon 1 – muudetud muutujanimed

Järgnevalt toob autor välja näite, kus kasutatakse prototüüpi leidmaks plagiaati kahe väga sarnase klassi puhul, kus muudetud on ainult muutujanimesid.

Joonis 3: Realisatsiooni sarnaste lähtefailide näide 1

```
public class TestIdentical1 {

    private int foo;

    private long bar;

    public TestIdentical1() {
    }

    public TestIdentical1(int foo, long bar) {
        this.foo = foo;
        this.bar = bar;
    }

    public long getFooPlusBar() {
        return foo + bar;
    }
}
```

Joonis 4: Realisatsiooni sarnaste lähtefailide näide 2

```
public class TestIdentical2 {

    private int length;

    private long width;

    public TestIdentical2() {
    }

    public TestIdentical2(int length, long width) {
        this.length = length;
        this.width = width;
    }

    public long getLengthPlusWidth() {
        return length + width;
    }
}
```

Vaadates peale eelnevale kahele lähtekoodi näitele, on näha, et kõik klassimeetodid teevad täpselt sama asja, neil on sama palju meetodeid, neil on samad tingimuslaused, konstruktorid ja väljundid. Ainuke erinevus nendel kahel klassil on see, et väljade nimed on ära muudetud ja seega kogu klassi ulatuses on muutujanimed erinevad.

CSV[1] tekstifaili formaadis näeb eelnevate meetodite klasside kompileerimisel saadud baitkood välja järgnev:

Joonis 5: Realisatsiooni sarnaste failide näite CSV baitkood

```
classId,method,bytecode
KLASSID1,public TestIdentical1(),aload_0 invokespecial return
KLASSID1,"public TestIdentical1(int, long)",aload_0 invokespecial aload_0 iload_1 putfield aload_0 lload_2 putfield return
KLASSID1,public long getFooPlusBar(),aload_0 getfield i2l aload_0 getfield ladd lreturn
KLASSID2,public TestIdentical2(),aload_0 invokespecial return
KLASSID2,"public TestIdentical2(int, long)",aload_0 invokespecial aload_0 iload_1 putfield aload_0 lload_2 putfield return
KLASSID2,public long getLengthPlusWidth(),aload_0 getfield i2l aload_0 getfield ladd lreturn
```

Nagu varasemalt on autor ka maininud ja eelnevalt välja toodud pildist näha on, siis muutujanimede muutmine baitkoodi ei muuda ja mõlema klassi baitkoodid on identsed, mille alusel võiks julgelt väita, et tegemist on võimaliku plagiaadiga.

Käivitades autori loodud prototüüp eelnevalt näidatud sisendfailidega, saame programmi väljundiks:

Joonis 6: Realisatsiooni sarnaste failide näite tulemus

```
Comparing classes with ID's: KLASSID1 and KLASSID2
Methods (public TestIdentical1()) and (public TestIdentical2()) instructions match 3/3 - Shorter 100% in longer
Methods (public TestIdentical1()) and (public TestIdentical2(int, long)) instructions match 2/9 - Shorter 66,67% in longer
Methods (public TestIdentical1()) and (public long getLengthPlusWidth()) instructions match 1/7 - Shorter 33,33% in longer
Methods (public TestIdentical1(int, long)) and (public TestIdentical2()) instructions match 2/9 - Shorter 66,67% in longer
Methods (public TestIdentical1(int, long)) and (public TestIdentical2(int, long)) instructions match 9/9 - Shorter 100% in longer
Methods (public TestIdentical1(int, long)) and (public long getLengthPlusWidth()) instructions match 2/9 - Shorter 28,57% in longer
Methods (public long getFooPlusBar()) and (public TestIdentical2()) instructions match 1/7 - Shorter 33,33% in longer
Methods (public long getFooPlusBar()) and (public TestIdentical2(int, long)) instructions match 2/9 - Shorter 28,57% in longer
Methods (public long getFooPlusBar()) and (public long getLengthPlusWidth()) instructions match 7/7 - Shorter 100% in longer
```

Kuna programm jooksutab meetodite puhul *all-to-all comparison'it*, siis nagu ka tulemusest näha, on kokku võrreldud meetodeid üheksa korda, kus igat ühe klassi meetodit on võrreldud iga teise klassi meetodiga. *Instructions match* suhe on arvutatud vastavalt sellele, kui palju on ühes meetodis järjestikkusi baitkoodi instruktsioone samas

[1] https://www.computerhope.com/jargon/c/csv.htm

järjekorras nagu seda on teises meetodis. Niisiis *2/9* tähendab, et ühes meetodis on kaks instruktsiooni, mis esinevad teise meetodi üheksast instruktsioonist täpselt samas järjekorras. Nagu programmitöö tulemuselt välja võib lugeda, on mõlemas klassis baitkoodi põhjal kolm omavahel identset meetodit, millest üks on aga tühi konstruktor. Kuivõrd konstruktoreid ei ole mõtet teiste meetoditega võrrelda (vähemalt ilma argumentideta konstruktoreid), siis realiseeris autor programmis alampiirangu, mis välistab vaikekonstruktoriga võrdlused. Käivitades peale piirangut samade tulemustega programmi, saame viis võrdlust vähem ja ainult kaks plagiaadiohtlikku meetodit.

5.2.2 Realisatsioon 2 – muudetud instruktsioonide järjekord

Nagu on välja toodud peatükis 4.2, siis baitkoodi üheks puuduseks on see, et muutes näiteks erinevate arvutuste järjekorda lähtekoodis, muutub ka järjekord baitkoodis, mis võib omakorda muuta lõpptulemust.

Järgnevalt on välja toodud näide, kus klassis olevad meetodid teevad täpselt sama asja, kuid baitkoodid ei ühildu, kuna järjekord lähtekoodis on erinev.

Joonis 7: Realisatsiooni erinevate järjekordade näide 1

```
public class TestChangeOfOrder1 {

    private int foo;

    public long getFooPlusFour() {
        return foo + 4;
    }
}
```

Joonis 8: Realisatsiooni erinevate järjekordade näide 2

```
public class TestChangeOfOrder2 {

    private int bar;

    public long getFourPlusBar() {
        return 4 + bar;
    }
}
```

Käivitades prototüüpi näitena toodud failidega, saame *CSV* faili nende baitkoodid, mis on identsed, kuid erinevad järjekorras:

Joonis 9: Realisatsiooni erinevate järjekordade näite baitkood CSV

```
classId,method,bytecode
klass_ID1,public long getFooPlusFour(),aload_0 getfield iconst_4 iadd i2l lreturn
klass_ID2,public long getFourPlusBar(),iconst_4 aload_0 getfield iadd i2l lreturn
```

Kuna programm otsib järjestikku asetsevaid instruktsioone, mis on samasugused, saame programmi lõpptulemiks:

Joonis 10: Realisatsiooni erinevate järjekordade näite tulemus

```
Comparing classes with ID's: klass_ID1 and klass_ID2
Methods (public long getFooPlusFour()) and (public long getFourPlusBar()) instructions match 3/6
```

Kuigi meetodite tulemid on identsed, on näha, et baitkoodi instruktsioonide järjekorrad on erinevad, mis tähendab, et baitkoodid ei ühildu üks ühele ja kattuvuse suhteks tagastab programm ainult kolm instruktsiooni kuuest.

Baitkood instruktsioonide järjekorra parandamiseks on autor otsustanud kasutada tavalist tähestiku järgi sorteerimist, mis annab *CSV* kujul meetodite baitkoodiks hoopiski sellise tulemuse:

Joonis 11: Realisatsiooni erinevate järjekordade näite CSV sorteeritud

```
classId,method,bytecode
klass_ID1,public long getFooPlusFour(),aload_0 getfield i2l iadd iconst_4 lreturn
klass_ID2,public long getFourPlusBar(),aload_0 getfield i2l iadd iconst_4 lreturn
```

Ja nagu *CSV* failile peale vaadates arvata võib, on programmi väljund täpselt selline nagu oodatud:

Joonis 12: Realisatsiooni erinevate järjekordade näite tulemus sorteeritud

```
Comparing classes with ID's: klass_ID1 and klass_ID2
Methods (public long getFooPlusFour()) and (public long getFourPlusBar()) instructions match 6/6
```

5.2.3 Realisatsioon 3 – üleliigne kood

Üleliigse koodi lisamine lähtefaili toodab üleliigseid instruktsioone baitkoodis, mis jällegi mõjutab lõpptulemust.

Järgnevalt on toodud näide üleliigse koodi mõjust baitkoodi instruktsioonidele ja ühest võimalikust lahendusest, et üleliigne kood ei mõjutaks liialt prototüübi tulemust.

Joonis 13: Realisatsiooni üleliigse koodi näide 1

```
public class TestArbitraryCode1 {

    private int foo;

    public long getFooPlusFour() {
        return foo + 4;
    }
}
```

Joonis 14: Realisatsiooni üleliigse koodi näide 2

```
public class TestArbitraryCode2 {

    private int bar;

    public long getBarPlusFourAndSout() {
        System.out.println(bar);
        return bar + 4;
    }
}
```

Nagu näidetelt näha, on kood täpselt samasugune, kuid ühele meetodile on lisatud *System.out.println()*, mis väljastab muutuja *bar* väärtuse konsooli. Eelnevate meetodite baitkood *CSV* failis näeb välja järgnev:

Joonis 15: Realisatsiooni üleliigse koodi näite CSV baitkood

```
classId,method,bytecode
klass_ID1,public long getFooPlusFour(),aload_0 getfield iconst_4 iadd i2l lreturn
klass_ID2,public long getBarPlusFourAndSout(),getstatic aload_0 getfield invokevirtual aload_0 getfield iconst_4 iadd i2l lreturn
```

Nagu baitkoodist näha võib, klapivad samasuguste koodijuppide (*return bar + 4*) baitkoodi instruktsioonid üks ühele, kuid ühel meetodil on lisaks ka *System.out.println()*, mis tekitab üleliigseid instruktsioone ja võrdlus ei ole enam nii mugav kui eelnevate näidete puhul.

Et üleliigne kood liialt tulemust ei ohustaks, on võimalik kasutusele võtta suhe lühema meetodi järjestike instruktsioonide arvu asumise kohta suuremas meetodis. Näide programmi tulemusest:

Joonis 16: Realisatsiooni üleliigse koodi näite tulemus

```
Comparing classes with ID's: klass_ID1 and klass_ID2
Methods (public long getFooPlusFour()) and (public long getBarPlusFourAndSout()) instructions match 6/10 - Shorter 100% in longer
Methods (public long getFooPlusFour()) and (public long getBarPlusFourAndSout()) SORTED instructions match 3/10 - Shorter 50% in longer
```

Programmi töö tulemusest on näha, et baitkoodi sorteerimine selle näite puhul teeb tulemuse ebausaldusväärseks ja ebatäpseks. Vaadates aga tulemust, mille saame ilma sorteerimata, on näha, et kattuvuse suhe on *6/10*, ja lühema meetodi instruktsioonid asuvad 100%'liselt samas järjestuses suurema meetodi instruktsioonide seas. Kui ühe meetodi instruktsioonid on pea täielikult teise meetodi sees, on ka oht, et tegemist võib olla plagiaadiga.

6 Valideerimine tegelike tudengite tööde põhjal

Olles läbi proovinud erinevaid teoreetilisi variante ja täiendanud prototüüpi vastavalt, proovis autor antud lahendust olemasolevate Tallinna Tehnikaülikooli programmeerimisainete erinevate tudengite tööde peal. Käesolev jaotis annab ülevaate saadud tulemustest ja veidi statistikast.

Järgnevalt toob autor välja mõningad näited saadud tulemustest, kasutades loodud baitkoodi põhjal plagiaadituvastust. Kuna prototüüp võrdleb kõiki meetodeid kõikide meetoditega, on väljundis palju ridu, mis meid antud näidete puhul ei huvita, seega toob autor välja vaid vastava näitega seonduva rea(d) väljundist.

6.1 Katse 1 – Võrdluse all MOSS plagiaadituvastussüsteemi põhjal 84% kattuvusega klassifailid

Esimese võrdlusena võrreldakse kahte klassifaili, kus on mõningad *getterid* ja *setterid*, *konstruktorid* ja üks klassi põhiline, sünkroniseeritud[1] meetod *getPassengers(...)*.

[1] https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html

Joonis 17: Reaalsed tudengite tööd katse 1 töö 1

```
public synchronized ArrayList<Passenger> getPassengers(int size, String location, String direction,
        String identifier, Schedule schedule) {
    ArrayList<Passenger> passengersTemp = new ArrayList<Passenger>();
    List<String> tempStops;
    Schedule scheduleTemp = availableSchedules.stream().filter(s -> s == schedule).findFirst().get();
    int tempCount = 0;
    if (tempCount < size) {
        tempStops = scheduleTemp.getStops();
        int temp = tempStops.indexOf(location);
        if (direction == "+") {
            tempStops = tempStops.subList(temp + 1, tempStops.size());
        } else {
            tempStops = tempStops.subList(0, temp - 1);
        }

        for (int k = 0; k < passengers.size(); k++) {
            if (!passengers.isEmpty()) {
                if (tempStops.contains(passengers.get(k).getDestination())
                        && passengers.get(k).getLastTrain() != identifier) {
                    tempCount++;
                    passengersTemp.add(passengers.get(k));
                    if (tempCount == size) {
                        passengers.removeAll(passengersTemp);

                        return passengersTemp;
                    }
                }
            }
        }
    }
    passengers.removeAll(passengersTemp);
    return passengersTemp;
}
```

Joonis 18: Reaalsed tudengite tööd katse 1 töö 2

```
public synchronized ArrayList<Passenger> getPassengers(int size, String location, String direction,
        String identifier, Schedule schedule) {
    ArrayList<Passenger> passengers1 = new ArrayList<Passenger>();
    List<String> stops;
    Schedule schedule1 = schedules.stream().filter(s -> s == schedule).findFirst().get();
    int count = 0;
    if (count < size) {
        stops = schedule1.getStops();
        int index = stops.indexOf(location);
        if (direction == "forwards") {
            stops = stops.subList(index + 1, stops.size());
        } else {
            stops = stops.subList(0, index - 1);
        }

        for (int k = 0; k < passengers.size(); k++) {
            if (!passengers.isEmpty()) {
                if (stops.contains(passengers.get(k).getDestination())
                        && passengers.get(k).getLastTrain() != identifier) {
                    count++;
                    passengers1.add(passengers.get(k));
                    if (count == size) {
                        passengers.removeAll(passengers1);

                        return passengers1;
                    }
                }
            }
        }
    }
    passengers.removeAll(passengers1);
    return passengers1;
}
```

Kuna meetodeid on klassides mitmeid ja võrreldakse kõiki meetodeid kõikidega, on tulemuses väga palju ridu, mis meid hetkel ei huvita. Tulemusena toob autor välja vaid vastavate meetodite võrdlustulemuse, mis näeb välja järgnev:

Methods (public synchronized java.util.ArrayList<passenger.Passenger> getPassengers(int, java.lang.String, java.lang.String, java.lang.String, schedule.Schedule)) and (public synchronized java.util.ArrayList<passengers.Passenger> getPassengers(int, java.lang.String, java.lang.String, java.lang.String, schedule.Schedule)) ***instructions match 101/101 - Shorter 100% in longer***.

Võrreldes antud failide lähtekoode on ilmselge, et tegemist on lihtsalt kopeeritud ja kleebitud koodiga, mida kinnitab meile ka baitkoodide võrdlustulemus. Kõik 101 instruktsiooni kattuvad mõlemal meetodil, kusjuures baitkood ei ole isegi sorteeritud.

6.2 Katse 2 – Võrdluse all MOSS plagiaadituvastussüsteemi põhjal 77% kattuvusega klassifailid näide 1

Järgnevalt toob autor välja kahe klassifaili puhul kaks näidet, mis pealtnäha tunduvad samasugused plagiaadid nagu eelnevas näites, kus väga palju vaeva peale koodi ümber tõstmise, pole nähtud. Selle näite puhul plagiaadituvastus baitkoodi põhjal enam nii kindlaid tulemusi, kui varasema näite puhul, ei anna.

Joonis 19: Reaalsed tudengite tööd katse 2 töö 1

```
private void checkTimetable() {
    if (this.getTimetable() == null) {
        this.searchTimetable();

    } else if (this.stationIndex == this.timetable.getAllStations().size() - 1) {
        this.timetablesDone++;
        if (controller.isWorking || this.timetablesDone < this.minimumTimetableToDo) {
            this.stationIndex = 0;
            this.searchTimetable();
        } else {
            System.out.println("Rong ID = " + this.getID() + " lõpetas töö");
            this.IsTimetableFinished = true;
            this.timetable = null;
        }

    }
}
```

Joonis 20: Reaalsed tudengite tööd katse 2 töö 2

```
private void checkSchedule() {
    // If train has no schedule, find one.
    if (this.getSchedule() == null) {
        this.findSchedule();

    // If at the end station of the schedule, reset station index and find next schedule, IF THE DAY IS NOT OVER!
    } else if (this.stationIndex == this.schedule.getStations().size() - 1) {
        this.schedulesCompleted++;
        if (controller.working || this.schedulesCompleted < this.minSchedulesToComplete) {
            this.stationIndex = 0;
            this.findSchedule();
        // If the work day is over and the Train is in the end Station of the current Schedule, unload Passengers and turn off the Train.
        } else {
            System.out.println("Train #" + this.getID() + " finished " + this.schedulesCompleted + " schedules and shuts down.");
            this.scheduleDone = true;
            this.schedule = null;
        }

    }
}
```

Lastes eelnevad meetodid loodud prototüübist läbi, saame järgneva tulemuse:

Methods (private void checkTimetable()) and (private void checkSchedule()) ***instructions match 47/61 - Shorter 83,93% in longer***.

Eelnev näide näitab väga hästi, et kõigest *System.out.println()'i* ühe välja väärtuse lisaks välja printimine muudab programmi tulemust päris suurel määral. Kusjuures tööl number kaks on ainult viis instruktsiooni rohkem (ainsad erinevad instruktsioonid kahe meetodi vahel) kui tööl number üks.

6.3 Katse 3 - Võrdluse all MOSS plagiaadituvastussüsteemi põhjal 77% kattuvusega klassifailid näide 2

Järgnevalt veel üks näide samadest töödest, kus väikesed muudatused koodis rikuvad prototüübi tulemust veelgi.

Joonis 21: Reaalsed tudengite tööd katse 3 töö 1

```
private void drive(Station nextStation, Railroad railroadToUse) {
    // Calculate the time to sleep to mimic driving.
    int timeToTravel = this.schedule.getSelectedStationTravelTime(nextStation);
    System.out.println("#" + this.getID() + ": " + this.currentStation + " --> " + nextStation
        + " [" + timeToTravel + "ms] " + " | Unloaded: " + this.passengersUnloaded + " | Loaded:" + this.passengersLoaded);
    try {
        Thread.sleep(timeToTravel);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    // After successful driving, increase station Index, to let others know the current position in the Schedule.
    this.stationIndex++;

    // Next station becomes current Station.
    this.currentStation = nextStation;

    // Notify other Trains that this Railroad is no longer in use, let others drive.
    this.freeRailroad(railroadToUse);
}
```

Joonis 22: Reaalsed tudengite tööd katse 3 töö 2

```
private void drive(Station nextStation, Railway railWayToUse) {

    int timeToTravel = this.timetable.getSelectedStationTravelTime(nextStation);
    System.out.println(this.getID() + ": " + this.currentStation + " " + nextStation + " aeg: "
        +  timeToTravel + " Maha laaditud: " + this.travelersOffTrain + " Peale laaditud: " + this.travelersToTrain);
    try {
        Thread.sleep(timeToTravel);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    this.stationIndex++;
    this.currentStation = nextStation;
    synchronized (railWayToUse) {
        this.freeRailway(railWayToUse);
    }
}
```

Jällegi kaks pealtnäha väga sarnast tööd, kuid väikeste muudatuste tulemusena on instruktsioonides ikka üht-teist erinevat. Prototüübi väljundiks nende meetodite võrdluse puhul on järgnev:

Methods (private void drive(Station, Railway)) and (private void drive(Station.Station, Railroad.Railroad)) ***instructions match 35/74 - Shorter 54,69% in longer***.

Eelnevast näitest on näha, et praktiliselt identsete meetodite baitkoodi instruktsioonid ei kattu nii suures ulatuses kui võiks arvata.

6.4 Kokkuvõte

Autor, olles proovinud loodud prototüüpi mitmete erinevate tudengite tööde peal, julgeb väita, et enamike tööde peal, kus plagiaadituvastussüsteemi MOSS poolt väljastatud kattuvuse protsent oli suur, on loodud programmi kattuvus ka pigem üle poole. Sellest võiks järeldada, et tegemist on plagiaadiohtu olukordadega.

Testides prototüüpi tudengite tööde peal selgus kaks suuremat miinust:

- väiksemad, seetõttu „lihtsamad“ meetodid saavad suure kattuvusprotsendi, kuigi vahel teevad suurel määral üldsegi erinevat asja;
- lihtsad asjad nagu näiteks klassimuutujate poole põõrdumine ja nendega toimetamine lisab baitkoodi instruktsioone, mis autori mõeldud meetodi puhul langetab oluliselt instruktsioonide kattuvust ja mõjutab tulemust.

Samuti selgus tudengite tööde põhjal, et baitkoodi sorteerimine enamasti ei mõjuta tulemust paremuse poole. Seda seetõttu, et kui sorteerida instruktsioonid ja ühel meetodil on näiteks *fcmpl* instruktsiooni kaks ja teisel kolm korda, siis järjestikune

instruktsioonide kattumine katkeb ja tulemus on sellevõrra kehvem. Lisaks eelnevale süvendab baitkoodi sorteerimine olukordi, kus meetodid teevad totaalselt erinevaid asju, kuid kattuvus sorteerimise tõttu suureneb.

7 Kokkuvõte

Antud töö eesmärgiks oli luua plagiaadituvastussüsteem, mis võimaldaks Java baitkoodi põhjal ära tunda erinevate programmeerimistööde puhul plagiaati. Töö oli suuremas osas eksperimentaalne, teadmata kas antud viis üldse sobib plagiaadituvastamiseks või mitte ja kuidas üleüldse baitkoodi põhjal otsustada, et tegemist võiks olla plagiaadiga.

Analüüsi tulemusena leidis autor, et loodaval plagiaadituvastussüsteemil on nii positiivseid kui ka negatiivseid külgi juba olemasolevate süsteemide ees ning prototüübi loomine tasub kindlasti proovimist. Prototüübi loomisel seadis autor endale kolm peamist eesmärki:

- realiseerida prototüüp, mis võimaldaks tuvastada naiivselt kopeeritud ja kleebitud koodi Java baitkoodi põhjal;
- prototüüp peab töötama kasutades igat operatsioonisüsteemi;
- prototüüp peab töötama kasutades nii Java lähtekoodi kui ka kompileeritud koodi.

Viimased kaks eesmärki on autor suutnud täielikult realiseerida. Rääkides esimesest eesmärgist, siis võib öelda, et loodud prototüüp on võimeline väga täpselt hindama plagiaati naiivselt ümber tõstetud meetodite puhul, kus on muudetud meetodi ja muutujanimesid. Raskemaks läheb aga olukord, kui on lisatud üleliigset koodi, mis põhjustab lisainstruktsioone kuskile meetodi keskele. Kuna autor valis kattuvuse tuvastamiseks järjestikuste instruktsioonide võrdluse, siis lisainstruktsioonid kuskil meetodi keskel lõhuvad instruktsioonide järjestikkuse ja muudavad oluliselt tulemust.

Antud plagiaadituvastussüsteemi on plaanis edasi arendada, mõeldes välja veidi töökindlama kattuvuse tuvastamise viisi, mis loodetavasti toob ka paremaid tulemusi.

Kasutatud kirjandus

[1] Java bytecode. https://en.wikipedia.org/wiki/Java_bytecode [WWW] (07.05.2017).

[2] Inside Java : The Java Virtual Machine. http://www.javacoffeebreak.com/articles/inside_java/insidejava-jan99.html [WWW] (18.05.2017).

[3] javac - The Java Compiler. https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/tooldocs/win32/javac.html [WWW] (12.05.2017).

[4] javap - The Java Class File Disassembler. http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javap.html [WWW] (12.05.2017).

[5] Prechelt L., Malpohl G., Philippsen M., Finding Plagiarisms among a Set of Programs with JPlag". http://www.jucs.org:8080/ujs/jucs/Journal/Volume%208/Issue_8_11/finding_plagiarisms_among_a/Prechelt_L.html;internal&action=buildframes.action&Parameter=1494275390043&ctx=eKS [WWW] (08.05.2017).

[6] Rabin-Karp Algorithm. https://brilliant.org/wiki/rabin-karp-algorithm/ [WWW] (18.05.2017).

[7] Myles, G., Collberg, C., k-gram Based Software Birthmarks, 2005 (07.05.2017).

[8] Plagiaat. https://et.wikipedia.org/wiki/Plagiaat [WWW] (07.05.2017).

[9] SRFI 41: Streams. https://srfi.schemers.org/srfi-41/srfi-41.html [WWW] (18.05.2017).

[10] Schleimer S., Wilkerson D. S., Aiken, A., Winnowing: Local Algorithms for Document Fingerprinting. https://www.researchgate.net/publication/2840981_Winnowing_Local_Algorithms_for_Document_Fingerprinting [WWW] (18.05.2017).

[11] B. Venners, Bytecode basics. www.javaworld.com/article/2077233/core-java/bytecode-basics.html [WWW] (12.05.2017).

[12] Java bytecode instruction listing. https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings [WWW] (18.05.2017).

[13] Chapter 8. Classes. http://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html#jls-8.8.9 [WWW] (12.05.2017).

Lisa 1 – katses 6.1 välja toodud katse meetodite baitkoodid

Kuna katse tulemusena saadud baitkoodi instruktsioonid olid identsed, ei hakka autor kõiki 101'te instruktsiooni topelt välja tooma ja toob ühe näitena välja mõlema meetodi baitkoodi, kusjuures iga instruktsioon on eraldatud komaga.

new, dup, invokespecial, astore, aload_0, getfield, invokevirtual, aload, invokedynamic, invokeinterface, invokeinterface, invokevirtual, checkcast, astore, iconst_0, istore, iload, iload_1, if_icmpge, aload, invokevirtual, astore, aload, aload_2, invokeinterface, istore, aload_3, ldc, if_acmpne, aload, iload, iconst_1, iadd, aload, invokeinterface, invokeinterface, astore, goto, aload, iconst_0, iload, iconst_1, isub, invokeinterface, astore, iconst_0, istore, goto, aload_0, getfield, invokevirtual, ifne, aload, aload_0, getfield, iload, invokevirtual, checkcast, invokevirtual, invokeinterface, ifeq, aload_0, getfield, iload, invokevirtual, checkcast, invokevirtual, aload, if_acmpeq, iinc, aload, aload_0, getfield, iload, invokevirtual, checkcast, invokevirtual, pop, iload, iload_1, if_icmpne, aload_0, getfield, aload, invokevirtual, pop, aload, areturn, iinc, iload, aload_0, getfield, invokevirtual, if_icmplt, aload_0, getfield, aload, invokevirtual, pop, aload, areturn

Lisa 2 – Katses 6.2 välja toodud katse meetodite baitkoodid

Rasvas kirjas instruktsioonid on *checkSchedule()* meetodis ainsad, mis erinevad meetodist *checkTimetable()*.

Instruktsioonid	
checkTimetable()	**checkSchedule*()***
aload_0	aload_0
invokespecial	invokespecial
ifnonnull	ifnonnull
aload_0	aload_0
invokespecial	invokespecial
goto	goto
aload_0	aload_0
getfield	getfield
aload_0	aload_0
getfield	getfield
invokevirtual	invokevirtual
invokeinterface	invokeinterface
iconst_1	iconst_1
isub	isub
if_cmpne	if_cmpne
aload_0	aload_0
dup	dup
getfield	getfield
iconst_1	iconst_1
iadd	iadd
putfield	putfield
aload_0	aload_0
getfield	getfield
getfield	getfield
ifne	ifne
aload_0	aload_0

getfield	getfield
aload_0	aload_0
getfield	getfield
if_icmpge	if_icmpge
aload_0	aload_0
iconst_0	iconst_0
putfield	putfield
aload_0	aload_0
invokespecial	invokespecial
goto	goto
getstatic	getstatic
new	new
dup	dup
invokespecial	invokespecial
ldc	ldc
invokevirtual	invokevirtual
aload_0	aload_0
invokespecial	invokespecial
invokevirtual	invokevirtual
ldc	ldc
invokevirtual	invokevirtual
Invokevirtual	**aload_0**
invokevirtual	**getfield**
aload_0	**invokevirtual**
iconst_1	**ldc**
putfield	**invokevirtual**
aload_0	Invokevirtual
aconst_null	invokevirtual
putfield	aload_0
return	iconst_1
	putfield
	aload_0
	aconst_null

	putfield
	return

Lisa 3 – Katses 6.3 välja toodud katse meetodite baitkoodid

Instruktsioonid	
drive(Station nextStation, Railroad railroadToUse) - töö 1	**drive(Station nextStation, Railway railWayToUse) - töö 2**
aload_0	aload_0
getfield	getfield
aload_1	aload_1
invokevirtual	invokevirtual
istore_3	istore_3
getstatic	getstatic
new	new
dup	dup
invokespecial	invokespecial
ldc	aload_0
invokevirtual	invokespecial
aload_0	invokevirtual
invokespecial	ldc
invokevirtual	invokevirtual
ldc	aload_0
invokevirtual	getfield
aload_0	invokevirtual
getfield	ldc
invokevirtual	invokevirtual
ldc	aload_1
invokevirtual	invokevirtual

aload_1	ldc
invokevirtual	invokevirtual
ldc	iload_3
invokevirtual	invokevirtual
iload_3	ldc
invokevirtual	invokevirtual
ldc	aload_0
invokevirtual	getfield
ldc	invokevirtual
invokevirtual	ldc
aload_0	invokevirtual
getfield	aload_0
invokevirtual	getfield
ldc	invokevirtual
invokevirtual	invokevirtual
aload_0	invokevirtual
getfield	iload_3
invokevirtual	i2l
invokevirtual	invokestatic
invokevirtual	goto
iload_3	astore
i2l	aload
invokestatic	invokevirtual
goto	aload_0
astore	dup
aload	getfield
invokevirtual	iconst_1
aload_0	iadd
dup	putfield
getfield	aload_0
iconst_1	aload_1
iadd	putfield
putfield	aload_2

aload_0	dup
aload_1	astore
putfield	monitorenter
aload_0	aload_0
aload_2	aload_2
invokespecial	invokespecial
return	aload
	monitorexit
	goto
	astore
	aload
	monitorexit
	aload
	athrow
	return