TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

**Mikko Maran**

# Õpilaste lahendusprogrammide sarnasuse kontrollija

**Bakalaureusetöö (9 EAP)**

Juhendaja Reimo Palm

Tartu 2021

## Õpilaste lahendusprogrammide sarnasuse kontrollija

**Lühikokkuvõte:**

Bakalaureusetöö eesmärgiks oli luua tarkvaralahendus programmeerimise algkursustel õppijate esitatud lahendusprogrammide sarnasuse kontrollimiseks. Loodav tarkvara abistab õppejõude õppijate omavahelise plagieerimise tuvastamisel. Lõputöö käigus valmis töölauarakendus, mille abil saab võrrelda Pythoni programmide sarnasust ning tuvastada omavahel sarnased programmid. Töös kirjeldatakse rakenduse loomise protsessi ja selleks kasutatud tehnoloogiaid ning selgitatakse rakenduse toimimist. Seejärel analüüsitakse ja antakse hinnang valminud tarkvarale. Kokkuvõttes leiti, et loodud tarkvara andis programmide võrdlemisel enamjaolt soovitud tulemusi, kuid leiti ka aspekte, mida saaks parendada.

**Võtmesõnad:**

Lähtekoodi plagiaat, lähtekoodi sarnasuse tuvastamine, programmeerimiskursused

**CERCS:** P175 Informaatika, süsteemiteooria

## Source code similarity detector for checking programs submitted by students

**Abstract:**

The purpose of this bachelor's thesis was to develop a software solution for checking the similarity of programs submitted by students in introductory programming courses. The software would aid teachers in the process of detecting plagiarism amongst students. A desktop application, which compares the source codes of Python programs to detect the most similar programs, was created. In this thesis a description of the development process and used technological tools is given. The functionality of the developed software is also explained. Finally, a review of the application is conducted to evaluate the quality of the application. In conclusion, the software gave sufficiently good results comparing students' programs, although some possibilities for improvements were also found.

**Keywords:**

Source code plagiarism, source code similarity detection, programming courses

**CERCS:** P175 Informatics, systems theory

# Sisukord

## Mõisted ja terminid

**Klasterdamine** (ingl *clustering*) on objektide grupeerimine sellisel viisil, et objektid, mis kuuluvad samasse gruppi ehk klastrisse, on üksteisega sarnasemad kui teistesse klastritesse kuuluvate objektidega. [1]

**Lekseem** (ingl *lexical token*) on programmikeele alfabeedi ühest või mitmest märgist koosnev string, mis kokkuleppeliselt esitab mingit tähenduse elementaarüksust. [2]

**Märgistuskeel** (ingl *markup language*) on vorminduskeel, mis on mõeldud teisendama toorteksti struktureeritud dokumentideks, protseduurilise ja deskriptiivse märgistuse lisamise teel toorteksti sisse. [3]

**Parsimine** (ingl *parsing*) on keeleüksuse süntaktilise struktuuri määramine ta lahutamisega alamüksusteks ja seoste leidmisega nende alamüksuste vahel. [4]

**Pistikmoodul** (ingl *plugin*) on lisandprogramm, mis on hõlpsalt installeeritav ja laiendab senise programmi võimalusi. [5]

**Plagiaat** (ingl *plagiarism*) on teise isiku loodud teose või selle osa avaldamine oma nime all, ka võõraste (teaduslike) seisukohtade esitamine nende allikaile viitamata. [6]

**Tagasüsteem** (ingl *back-end*) on kasutajale nähtamatu rakenduse osa, mis tegeleb andmete hoiustamise ning töötlemisega ja käsitseb kasutajaliidese kaudu antud sisendit. [7]

---

[1] https://www.iso.org/obp/ui#iso:std:iso-iec:tr:29119:-11:ed-1:v1:en:term:3.1.22

[2] http://www.eki.ee/dict/its/index.cgi?Q=lekseem&F=M&C06=et&C10=1

[3] https://stats.cyber.ee/term/3209-markup-language

[4] https://akit.cyber.ee/term/3627-parsing#t_3627

[5] https://akit.cyber.ee/term/2437-plugin-3

[6] https://www.eki.ee/dict/ekss/index.cgi?Q=plagiaat&F=M

[7] https://whatis.techtarget.com/definition/front-end

# Sissejuhatus

Plagieerimine on akadeemilises maailmas olnud pikalt aktuaalseks teemaks ning üheks suurimaks murekohaks. Hoolimata sellest, et plagieerimist taunitakse ning selle tõkestamiseks võetakse kasutusele järjest rohkem meetmeid, ei ole selle praktiseerimine siiski vähenemas [1].

Programmeerimisalaste tööde puhul on õppijate suhtumine akadeemilisse petturlusse leebem, kui see on näiteks esseede kirjutamise puhul [2]. See tähendab aga, et plagieerimine on programmeerimisülesannete puhul levinumaks probleemiks kui muud tüüpi ülesannetes. Niinimetatud teadmiste vahetamisel põhinev plagiaat, kus õppijad teadlikult jagavad oma lahenduskäike või koguni terveid koodilõike üksteisega, on IT-tudengite seas üks levinumaid plagieerimise viise [3]. Samuti suurendab probleemi see, et internetis on olemas paljude tüüpülesannete lahenduste koodinäited, mida on lihtne leida ja kasutada.

COVID-19 pandeemia tõttu on 2020. aasta algusest alates suurem osa õppetööst toimunud veebi teel. See on andnud õppijatele soodsamad võimalused akadeemilise petturlusega tegelemiseks [4], mistõttu on vajadus plagieerimist tõkestavate meetodite jaoks veelgi suurem. Kuna aga kodustes tingimustes õppetööd läbi viies on oluliselt keerulisem pidada järelevalvet õpilaste üle, siis on plagieerimist ennetavate meetodite kasutamine raskendatud.

Eelnevalt kirjeldatu viitab sellele, et kuna plagieerimise ennetamine on sageli keeruline, siis alternatiivseks abinõuks plagieerimisega võitlemisel on tegeleda plagiaadijuhtumite tuvastamise ning asjaosaliste karistamisega. Kui suudetakse tuvastada järjest suurem osa toimuvatest plagiaadijuhtumitest, on tõenäolisem, et väheneb üldine akadeemilise petturluse praktiseerimine, sest õppijad on teadlikud tõsistest tagajärgedest, mis kaasnevad plagieerimisega vahele jäämisega.

Bakalaureusetöö eesmärk on luua tarkvaralahendus programmeerimisainetes õppijate esitatud lahendusprogrammide sarnasuse kontrollimiseks, et seeläbi tuvastada potentsiaalseid plagiaadijuhtumeid. Loodav tarkvara on abivahendiks õppejõududele ning mõeldud eelkõige programmeerimise algkursustel kasutamiseks. Esimene õppeaine, kus selle kasutamist proovitakse, on Tartu Ülikooli sissejuhatav programmeerimisaine nimega „LTAT.03.001 Programmeerimine“ [5]. Rakenduse eduka toimimise korral saab selle kasutusele võtta ka teistel programmeerimise algkursustel.

Tartu Ülikoolis õpetatavatel programmeerimisalastel kursustel õpib korraga sadu tudengeid. Suurima õppijate arvuga programmeerimisaine ongi „Programmeerimine“, milles osaleb igal sügissemestril üle 300 tudengi. Isegi rühmadeks jaotatuna on õpilaste esitatud tööde parandamine õppejõudude jaoks väga aeganõudev tegevus. Kui veel lisaks lahenduste õigsuse kontrollimisele peavad juhendajad töid käsitsi omavahel võrdlema akadeemilise petturluse tuvastamiseks, siis suurendab see veelgi nende töökoormust. Selle lõputöö käigus loodav tarkvara õpilaste lahendusprogrammide sarnasuse automaatseks kontrollimiseks aitabki vähendada manuaalse plagiaadi kontrollimise mahtu.

Kursusel õppiva mitmesaja õpilase jaotamine väiksemateks rühmadeks teeb küll lihtsamaks kursuse haldamise, kuid tegelikkuses vähendab see oluliselt tõenäosust, et potentsiaalsed plagiaadijuhtumid tuvastatakse. Juhendajad kontrollivad esitatud lahendusi vaid enda rühma siseselt, kuid töö originaalautor ning plagieerija ei pruugi üldse olla samas rühmas. Seetõttu võivad manuaalsel plagiaadi kontrollimisel paljud plagiaadijuhtumid tähelepanuta jääda. Loodav tarkvara aitab ka seda probleemi lahendada, kuna esitatud tööde sarnasuse kontroll tehakse mitte rühmasiseselt, vaid kõikide kursusel osalejate lahenduste peal.

Bakalaureusetöö koosneb neljast peatükist. Esimeses analüüsitakse olemasolevat plagiaadi-tuvastustarkvara ning meetodeid plagiaadi tuvastamiseks programmide lähtekoodides. Teises peatükis kirjeldatakse autori loodud rakendusele seatud nõudeid, selle loomisel kasutatud tehnoloogiad ning rakenduse kasutajaliidese- ja sarnasuse analüüsi toimimist. Kolmandas peatükis analüüsitakse valminud rakenduse vastavust seatud nõuetele ning antakse hinnang valminud rakenduse kvaliteedile. Viimases peatükis tuuakse välja rakenduse edasiarendamise võimalused.

# 1 Olemasoleva tarkvara analüüs

Selles peatükis analüüsitakse olemasolevat plagiaadituvastustarkvara. Kirjeldatakse plagiaadituvastusrakenduste liigitamist, nendes kasutatavaid meetodeid plagiaadi tuvastamiseks ning probleeme, mis sageli esinevad nende rakenduste kasutamisel õpilaste programmide lähtekoodide võrdlemiseks.

## 1.1 Plagiaadi tuvastamise tarkvara liigitamine

Olemasoleva plagiaadituvastustarkvara kohta on tehtud ülevaatlikke kokkuvõtteid [6, 7, 8]. Autorid pakuvad nende liigitamiseks välja erinevaid viise. Mozgovoy [6] jagab plagiaadi tuvastamise tarkvara kahte suuremasse klassi: võrguühendusega ning võrguühenduseta plagiaadituvastustarkvara.

Võrguühendusega süsteemid viivad läbi plagiaadikontrolli otsides etteantud töö sisu põhjal originaaltöid, mida on potentsiaalselt plagieeritud, internetis olevatest andmebaasidest ning otsingumootorite otsingutulemuste hulgast [6]. Need on üldjuhul loodud universaalsete lahendustena ehk kontrollimiseks saab ette anda ükskõik missuguse teksti kujul oleva teose. See tähendab, et kuna võrguühendusega süsteemid ei ole kohandatud spetsiaalselt programmide lähtekoodide kontrollimiseks, siis ei pruugi nende kasutamine sel eesmärgil anda piisavalt täpseid tulemusi. Samuti on võrguühendusega plagiaadituvastustarkvara kasutamine plagiaadi kontrollimise meetmena programmeerimise algkursustel pigem väheefektiivne. Põhjuseks on see, et tavaliselt püütakse ülesanded koostada piisavalt unikaalsetena, et poleks võimalik leida täpselt samasuguste ülesannete lahendusi internetist. Sellepärast on veebis olevate lahenduste plagieerimine õppijate seas ka pigem harvaesinev probleem. Võrguühendusega plagiaadi tuvastamise tarkvarast on tuntumad Turnitini [9] ning Grammarly [10] poolt pakutavad lahendused.

Võrguühenduseta tarkvara seevastu teostab kontrolli vaid sisendiks antavate tööde hulgas, võrreldes neid omavahel ning filtreerides nende hulgast välja sarnased tööd [6]. Programmeerimise algkursustel on sellise plagiaadikontrollimistarkvara kasutamine eelistatud võrguühendusega tarkvarale. Üldiselt ongi plagiaadi kontrollimise eesmärk tuvastada õppijate omavahelist plagieerimist, kuna see on praktikas märksa levinum, kui internetis olevate lahenduste plagieerimine. Sageli pole ka ülesannete lahendustes üldse lubatud veebimaterjalide kasutamine, seega sel juhul, isegi nendele materjalidele viidates, oleks tegemist ülesande nõuete rikkumisega. Olemasolevatest võrguühenduseta lahendustest, mis on

loodud just programmide lähtekoodide võrdluseks, on enim kasutatavad JPlag [11], MOSS [12], Sherlock [13] ning YAP3 [14]. Antud lõputöö käigus valmiva tarkvara puhul on tegemist samuti võrguühenduseta plagiaadituvastajaga.

Kokkuvõtteks saab öelda, et võrguühendusega plagiaadituvastustarkvara on mõeldud selleks, et tuvastada internetist plagieerimist, ning võrguühendusta tarkvara selleks, et tuvastada esitatud tööde autorite omavahelist plagieerimist. Programmeerimiskursustel kasutamiseks on efektiivsem võrguühenduseta plagiaadituvastustarkvara, sest üldiselt toimub plagieerimine suuremal määral just õppijate vahel, mitte niivõrd internetimaterjalide põhjal.

## 1.2 Lähtekoodi plagiaadi tuvastamiseks kasutatavad meetodid

Plagiaadi tuvastamine programmide puhul põhineb lähtekoodide vahelise sarnasuse leidmisel. Mida sarnasemad on kaks koodi, seda tõenäolisem on, et ühe programmi autor on teise autori programmi plagieerinud. Tavaliselt ei esitata plagieerimisel originaalprogrammiga täpselt samasugust programmi, vaid enne tehakse selles teatavaid muudatusi. Plagiaadi tuvastamisel sarnasuse võrdlemiseks sobiva meetodi valimine sõltub aga olulisel määral sellest, millisel viisil on neid muudatusi koodis tehtud. Järgnevalt kirjeldatakse plagieerimisel kasutatavaid võtteid ning meetodeid nende võtete tuvastamiseks.

Joy jt [7] toovad välja, et programmi koodi muutmist saab jaotada kaheks. Esimeseks viisiks on programmi leksiline muutmine ehk koodis väikeste muudatuste tegemine selliselt, et programmi töövoog ning struktuur jäävad lõppkokkuvõttes samaks. Selliste muudatuste näideteks on:

- muutujanimede muutmine;
- kommentaaride lisamine, eemaldamine või ümbersõnastamine;
- tühikute ja tühjade ridade eemaldamine või lisamine;
- väiketähtede muutmine suurtähtedeks või vastupidi;
- täpitähtede asendamine.

Joy jt järgi on teine viis programmi koodi muutmiseks struktuuriline muutmine, mis tähendab, et muudetakse programmi ülesehitust ning seetõttu muutub ka selle töövoog, kuid programmi väljund jääb sel juhul siiski samaks. Struktuursete muudatuste näideteks on [7]:

- koodiridade või -plokkide järjekorra muutmine;
- võtmesõnade või tsüklite asendamine samaväärsete alternatiividega;
- operaatorite ja operandide järjekorra muutmine.

Leksiliste muudatuste tegemine on kõige lihtsam viis originaalkoodi muutmiseks, kuna see ei nõua põhjalikku arusaama originaalautori poolt kasutatud programmeerimisvõtetest, sest keskendutakse peamiselt vaid programmi koodi välimuse muutmisele. Seetõttu on loomulik, et leksiliste muudatuste tegemine on eriti just algajate programmeerijate seas kõige levinum viis koodi muutmiseks. Seda toetab ka Marttila bakalaureusetöö pealkirjaga „Programmide sarnasuse määratlemine ühe programmeerimisülesande näitel“ [15], milles analüüsiti Tartu Ülikooli algajatele mõeldud programmeerimise e-kursuse “Tehnoloogia tarbijast loojaks” osalejate ühe ülesande lahenduste sarnasust. Sarnaste lahenduste puhul analüüsiti, missuguseid plagieerimisvõtteid võib olla kasutatud. Valdavalt olid seal välja toodud just leksiliste muudatuste alla kuuluvad võtted.

Joy jt [7] väidavad, et struktuursete muutmisviiside kasutamine üldjuhul eeldab, et plagieerija saab originaalautori lahendusest aru, sest ta peab veenduma, et programmi ülesehituse muutmisel jääb programm korrektselt tööle. Algajatele programmeerijatele mõeldud kursustel on ülesanded tavaliselt lihtsa loomuga, mis tähendab, et sageli on ühe ülesande lahendamiseks piiratud arv erinevaid viise. See aga tähendab, et ka struktuursete muudatuste tegemiseks on vaid loetud arv võimalusi. Kui sellise ülesande puhul suudab plagieerija siiski teha ka struktuurseid muudatusi, siis põhimõtteliselt on ta ülesande ideest aru saanud ning sel juhul oskaks plagieerija ilmselt ka iseseisvalt ülesande ära lahendada ja tegelikkuses puudub vajadus kellegi teise lahendust maha kirjutada. Seetõttu on algajate programmeerijate seas plagieerimisel struktuursete muudatuste tegemine pigem vähelevinud.

Struktuursetel muudatustel põhinevat plagieerimist saab tuvastada kasutades programmi atribuutide loendamisel baseeruvaid meetodeid, millest kirjutavad Karnalim jt [8]. Sel puhul loendatakse kokku näiteks programmides esinevate muutujate, operaatorite, operandide, võtmesõnade, tühikute, kommentaaride jm elementide arvud ning programmide vaheline sarnasus leitakse selle põhjal, kui sarnased on samade atribuutide esinemissagedused programmides. Atribuutide esinemissagedused üldjuhul ei muutu, kui programmi struktuuri muuta, sellepärast on see meetod sobilik struktuursetel muudatustel põhineva plagieerimise tuvastamiseks. Seevastu leksiliste muudatuste tegemine mõjutab tavaliselt ka programmi atribuutide esinemissagedusi. Praktikas ei ole atribuutide loendamise meetodid aga sageli

eriti efektiivsed, sest harva esineb juhte, kus muudetakse ainult programmi struktuuri ning leksilisi muudatusi sealjuures ei tehta.

Kuna leksiliste muudatuste tegemine reeglina ei muuda programmi struktuuri, siis on selliste muudatustega toime pandud plagiaadi tuvastamiseks paslik kasutada programmide struktuuri võrdlemise meetodeid. Karnalim jt [8] väitel põhinevad need meetodid enamasti sõne-, puu- või graafivõrdlusel, millest enim kasutatakse sõnevõrdlust. See on üldiselt teostatud kahes osas: kõigepealt parsitakse programmide koodid lekseemide (ingl *lexical tokens*) jadaks ning seejärel kasutatakse valitud sõnevõrdlusalgoritmi, et leida kõikide lekseemideks teisendatud lahenduskoodide vahelised sarnasused. Siis leitakse eeldefineeritud lävendist suurema sarnasusega programmide paarid, mis loetakse plagiaadiks. Struktuuri võrdlemise meetodid on keerukamad kui tunnuste võrdlemise omad, kuid üldjuhul annavad ka täpsemaid tulemusi [8].

Karnalim jt sõnul [8] on struktuuri võrdlemise meetodid plagiaadituvastusprogrammides ka kõige laialdasemalt kasutusel. Selliseid meetodeid kasutavad näiteks JPlag [11], mis sooritab sõnevõrdlust algoritmiga *Running Karp-Rabin Greedy String Tiling* [16], ning Sherlock [13], mis kasutab sõnevõrdluseks kombinatsiooni pikima ühisjada (ingl *longest common subsequence*) [17] algoritmist ning Levenshteini kaugusest [18]. Viimast on kasutatud ka antud töö käigus loodud tarkvaras programmide sarnasuse tuvastamisel.

## 1.3 Probleemid õpilaste lahendusprogrammide sarnasuse võrdlemisel

Olemasolevad plagiaadi tuvastamise meetodid on erineva otstarbega. Nende meetodite kasutamise efektiivsus sõltub sellest, millises kontekstis ja mis eesmärgil plagiaadi tuvastamist tehakse. Järgnevalt analüüsitakse olemasolevate plagiaadituvastussüsteemide kasutamist õpilaste lahendusprogrammide võrdlemiseks ning seejuures tuuakse välja levinumad probleemid.

Õppeaines „Programmeerimine“ [5] kasutatakse õppijate lahenduste esitamiseks veebikeskkonna Moodle pistikmoodulit VPL (Virtual Programming Lab) [19]. See võimaldab õpilaste lahenduste õigsuse automaatset hindamist lahenduste esitamisel, mistõttu on see väga kasulik vahend programmeerimiskursustel kasutamiseks. Samuti on VPL moodulis sisseehitatud plagiaadikontrollija, mis ongi ainus hetkel aines „Programmeerimine“ kasutusel olev lahenduste sarnasuse kontrollimise viis käsitsi kontrolli kõrval. Samas on õppejõudude kogemus näidanud, et VPL-i plagiaadikontrollija ei anna piisavalt täpseid tulemusi. Sageli tuvastab

see sarnasteks selliseid programmipaare, mida käsitsi üle vaatamisel ei saaks kuidagi lugeda plagiaadiks, ning samas jätab tähelepanuta programmipaare, mis manuaalsel kontrollimisel tunduvad sarnased.

Kuna suur osa plagiaadituvastusprogramme kasutab lähtekoodide võrdlemiseks struktuuri võrdlevaid meetodeid, siis tehakse sageli enne lähtekoodide võrdlemist nende eeltöötlus, et vähendada võrdlusalgoritmi ajakulu. Selline eeltöötlus tähendab tavaliselt programmi koodi muutmist mingisuguseks lekseemide jadaks. Lekseemideks teisendamise eesmärk on alles jätta ainult võrdlusalgoritmi jaoks vajalik info programmi kohta, mis tähendab, et paratamatult läheb kaduma informatsiooni programmide kohta. Joy jt [7] väidavad, et lihtsamate ülesannete lahendusprogrammide puhul võib see aga põhjustada liigset programmide üldistamist. See tähendab, et koodide lekseemideks teisendamise tulemusel võib juhtuda, et võrdlemisel loetakse sarnaseks ka programmid, mis tegelikult ei olnud sarnased.

VPL-i plagiaadikontrollija puhul ilmneb tõenäoliselt sarnane probleem. Rodríguez-del-Pino jt [19] seletavad VPL-i plagiaadituvastaja tööpõhimõtet. Selle eeltöötlusfaasis tehakse programmide leksiline analüüs, mille käigus muudetaksegi programmide koodid lekseemideks. Seejärel toimub lekseemide filtreerimine, kus jäetakse välja analüüsi jaoks ebaolulised lekseemid. Selle etapi käigus tõenäoliselt toimubki programmide liigne üldistamine, mis annab ebatäpseid lõpptulemusi.

Tartu Ülikoolis õpetatavatel algajate programmeerimiskursustel on kasutatavaks programmeerimiskeeleks valdavalt Python [20], seda ka õppeaines „Programmeerimine“. Paraku ei võimalda kõik olemasolevad tarkvaralahendused Pythoni lähtekoodide võrdlemist ning lahendused, mis seda võimaldavad, ei ole sageli sobilikud mitmesaja lähtekoodi samaaegseks võrdlemiseks.

Üheks tarkvaralahenduseks, millega saab Pythoni lähtekoodides plagiaati tuvastada, on CodeGrade [21], mis on olemas nii iseseisva rakendusena kui pistikmoodulina mitmetes õppesüsteemides kasutamiseks kaasa arvatud Moodle’is. Tegemist on sarnase tarkvaraga nagu VPL ehk see on mõeldud õppijate lahenduste hindamise automatiseerimiseks ning sealjuures sisaldab ka automaatset plagiaadituvastussüsteemi. Selle tarkvara miinuseks on aga kättesaadavus, kuna tegemist on tasulise lahendusega, kui näiteks VPL on vabavara. Moodle’i VPL-mooduli plagiaadikontrollijaga saab samuti Pythoni lähtekoodides plagieerimist tuvastada, kuid eelnevalt kirjeldati selle kasutamisega seonduvaid probleeme. Vabavaralistest rakendustest on laialdasemalt kasutusel veel JPlag [11] ning MOSS [12]. Mõlema

puhul on miinuseks see, et tegemist on üsna vanade lahendustega – need on loodud vastavalt aastatel 2004 ja 1997 – ning vahepeal ei ole neid ka oluliselt uuendatud. Samuti on rakenduste puuduseks veel kasutamise mugavus, kuna tegemist on käsureaprogrammidega.

Olemasoleva plagiaadituvastustarkvara analüüsist selgub, et ei leidu piisavalt head lahendust, mis oleks sobilik kasutamiseks algajatele mõeldud programmeerimiskursustel, kus kasutatavaks programmeerimiskeeleks on Python. See tähendab, et on vaja uut plagiaadikontrollimistarkvara, mis lahendaks kirjeldatud probleemid.

# 2 Loodud rakenduse kirjeldus

Bakalaureusetöö käigus loodi töölauarakendus[8], millega saab kontrollida programmeerimisainetes õppijate poolt esitatud programmide lähtekoodide omavahelist sarnasust, et seeläbi tuvastada potentsiaalseid plagieerimisjuhtumeid. Selles peatükis kirjeldatakse autori loodud rakendust üksikasjalikult. Tutvustatakse rakenduse loomisel järgitud nõudeid ja kasutatud tehnoloogiaid ning valminud rakenduse kasutajaliidest ja sarnasuse analüüsi toimimist.

## 2.1 Nõuded rakendusele

Olemasolevate tarkvaralahenduste analüüsimine käigus tulid välja olulised aspektid, mida peaks jälgima uue sarnasuse kontrollija loomisel, eelkõige lähtudes sellest, et see on mõeldud algajate programmeerijate lahenduskoodide võrdlemiseks. Järgnevalt on nende aspektide ning õppejõudude soovituste põhjal kokkuvõtlikult esitatud nõuded uuele tarkvarale.

1. Lähtekoodide sarnasuse võrdlemise algoritmi põhifookuseks on tuvastada leksilistel muudatustel põhinevat plagieerimist.
2. Lähtekoodide eeltöötlemisel tohib algsest informatsioonist eemaldada minimaalse osa.
3. Rakendus peab suutma võrrelda programmeerimiskeeles Python kirjutatud programmide lähtekoode.
4. Rakendus peab analüüsima ZIP-faili kujul olevat sisendit, mis genereeritakse Moodle'i VPL pistikmooduli poolt, kusjuures tuleb arvestada genereeritava ZIP-faili sisese kaustastruktuuriga.
5. Rakendus peab teostama sarnasuse kontrolli paarikaupa kõigi sisendiks antud lähtekoodide vahel.
6. Rakendus näitab sarnasuse analüüsi tulemusena leitud sarnaste lahendusprogrammide paare ning võimalusel grupeerib leitud paarid suurematesse gruppidesse.
7. Rakenduse analüüsi tulemustest on võimalik välja lugeda leitud sarnaste lahenduste autorid.
8. Sarnasuse analüüs kuni 300 õppija esitusega, kus iga õppija esituse hulgas on keskmiselt kolme eri ülesande lahendusfailid, ei tohi võtta kauem aega kui 10 minutit.

Uue tarkvaralahenduse loomisel on püütud järgida kõiki nimetatud nõudeid.

[8] Rakenduse lähtekoodi repositoorium on ära toodud lisas 1.

## 2.2 Kasutatud tehnoloogiad

Selles alapeatükis on kirjeldatud loodud rakenduse arendamisel kasutatud tehnoloogilisi vahendeid ning põhjendatud nende valimist. Kasutatavate tehnoloogiate valimisel piiranguid ei olnud. Järgnevalt kirjeldatakse lähemalt seitset olulisemat kasutatud tehnoloogiat.

### 2.2.1 Java

Loodud rakenduse tagasüsteem (ingl *back-end*) on kirjutatud programmeerimiskeeles Java ning kasutatud on Java arenduskomplekti (JDK) versiooni 11 [22]. Java on juba aastaid olnud üks enim kasutatavaid programmeerimiskeeli maailmas. PYPL ehk PopularitY of Programming Language indeks näitab programmeerimiskeele populaarsust selle järgi, kui palju Google'i päringuid tehakse vastava programmeerimiskeele õpetuste otsimiseks [23]. Selle näitaja järgi on Java praeguse seisuga Pythoni järel teisel kohal.

Java puhul on tegemist objektorienteeritud programmeerimiskeelega, mis tähendab, et Javas kirjutatud programmi koodi saab jagada päriselu objektide või abstraktsete funktsionaalsete üksuste põhjal klassideks ning defineerida klassides nende objektide kasutamise loogika. Java objektorienteeritud lähenemine võimaldas loodud sarnasuse kontrollija tagasüsteemi funktsionaalsuse järgi lihtsasti struktureerida.

### 2.2.2 JavaFX

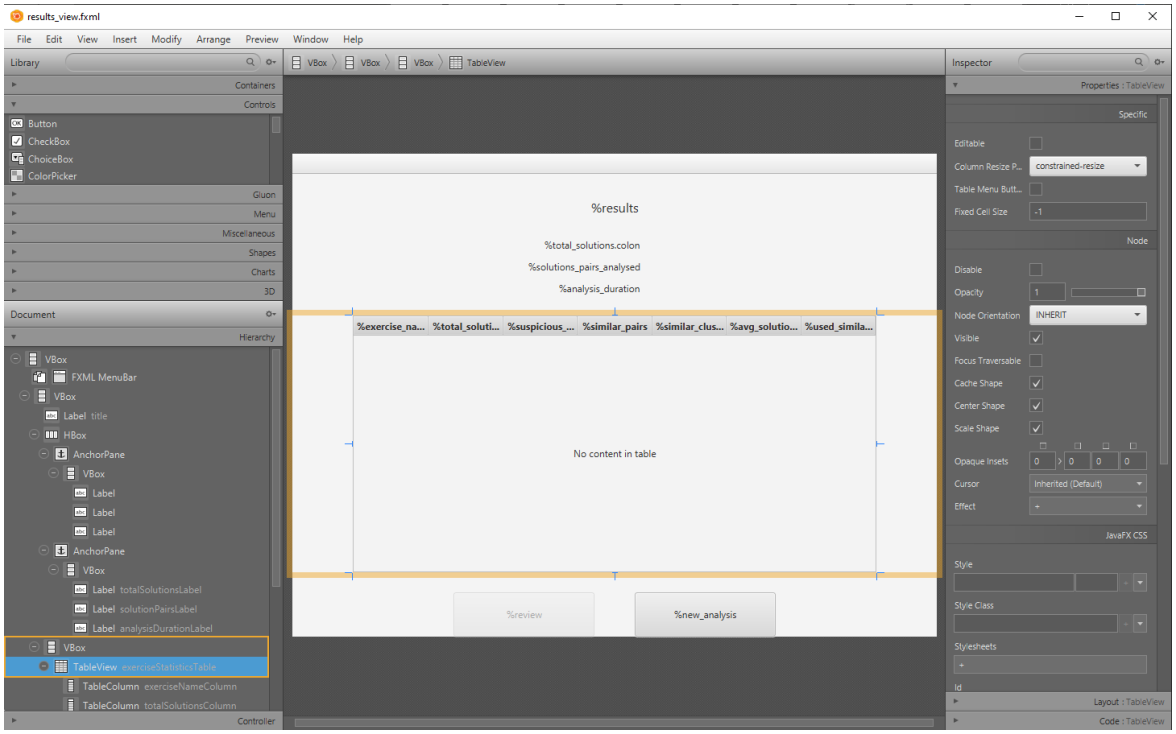
Rakenduse kasutajaliides on loodud platvormil JavaFX (versioon 11) [24], mis on eelkõige töölauarakenduste graafiliste kasutajaliideste loomiseks mõeldud platvorm, kuid selle abil saab luua ka veebirakendusi. Antud rakendus sai loodud töölauarakendusena, kuna vajadus veebirakenduse loomiseks puudus, sest rakendus ei kasuta veebiühendust.

JavaFX võimaldab graafilise kasutajaliidese loomist Java tagasüsteemiga. Kasutajaliidese loomiseks on olemas kõik üldlevinud graafilise kasutajaliidese elemendid nagu nupud, tekstiväljad, raadionupud, tabelid, menüüd jpm.

### 2.2.3 Scene Builder

JavaFX'ga programmiliselt (ingl *programmatically*) kasutajaliidese disainimine on üsna tülikas tegevus. Rakendus tuleb iga kord käivitada, et näha koodi kirjutamise tulemusel toimunud visuaalseid muudatusi kasutajavaates. Samuti võtab puhtalt koodi kirjutamise teel kasutajaliidese loomine võrdlemisi palju aega. Selle protsessi mugandamiseks on olemas ka

ise JavaFX platvormil loodud rakendus nimega Scene Builder [25]. Joonisel 1 on näidatud rakenduse Scene Builder kasutamist graafilise kasutajaliidese loomiseks.




Joonis 1. Näide Scene Builderis kasutajavaate loomisest

Scene Builder võimaldab kasutajaliidese vaadete või väiksemate komponentide disainimist ja loomist *drag-and-drop* põhimõttel ehk saab lohistada erinevaid graafilise kasutajaliidese elemente loodavasse vaatesse ning neid seal otse muuta, ilma et peaks kirjutama ühtegi rida koodi. Samuti näeb muudatuste tegemisel korraga ka vaate visuaalset muutumist, ilma et peaks rakendust eraldi käivitama, mis kiirendab kasutajaliidese disainimise protsessi.

### 2.2.4 FXML

Scene Builderiga kasutajaliidese komponentide loomisel genereeritakse FXML-fail, mida saab Java koodiga lihtsasti laadida rakenduse graafilises kasutajaliideses kasutamiseks. FXML on XML-keele baasil loodud märgistuskeel, mis on mõeldud JavaFX'i kasutaja-liideste loomise lihtsustamiseks. FXML-fail sisaldab infot kasutajaliidese graafiliste elementide omavahelise hierarhia, asetuse jm atribuutide kohta [26]. Joonisel 2 on toodud näide FXML-faili hierarhilisest struktuurist.

```
<AnchorPane fx:id="root" minHeight="0.0" minWidth="0.0" >
    <children>
        <TabPane tabClosingPolicy="ALL_TABS">
            <tabs>
                <Tab fx:id="codeTab">
                    <content>
                        <AnchorPane minHeight="0.0" minWidth="0.0">
                            <children>
                                <WebView fx:id="webView"/>
                            </children>
                        </AnchorPane>
                    </content>
                </Tab>
            </tabs>
        </TabPane>
    </children>
</AnchorPane>
```

Joonis 2. Näide FXML-i struktuurist

Igale FXML-failile saab lisada kontrollerklassi, kus on implementeeritud FXML-failis defineeritud kasutajaliidese elementide kasutamise loogika rakenduse töötamise ajal. Näiteks kui FXML-failis on defineeritud nupp, siis on kontrollerklassis võimalik määrata, mis juhtub selle nupu vajutamisel. Seega saab hoida kasutajaliidese struktuuri FXML-failides ning toimimise loogika eraldi kontrollerklassides.

### 2.2.5 CSS

JavaFX'i graafiliste elementide kujundust saab muuta programmiliselt, kuid see tähendaks, et koodis on segamini rakenduse kasutamise loogika ja kujundus. Programmeerimise heade tavade kohaselt on aga soovitatav need üksteisest eraldi hoida, sest see parandab koodi loetavust. JavaFX'i puhul on seda mugav teha, sest on võimalik kasutada JavaFX CSS-i [27]. Tavaline CSS (Cascading Style Sheets) [28] on põhiliselt veebilehtede kujundamiseks kasutatav märgistuskeel. CSS-failidesse kirjutatakse kasutajaliidese elementide välimust ja paigutust määravad reeglid. JavaFX CSS ongi loodud CSS-i põhjal ning seda on modifitseeritud nii, et saaks seda kasutada ka JavaFX'i graafiliste elementide kujundamiseks.

### 2.2.6 Jython

Loodud rakendus kasutab lähtekoodide eeltöötlemiseks programmeerimiskeeles Python kirjutatud skripte. Selleks, et käivitada Pythoni koodi ilma, et rakendus peaks sõltuma lisaks Javale ka Pythoni versioonist ning seeläbi vähendada rakenduse kettaruumi vajadust, on kasutatud Java pistikmoodulit nimega Jython (versioon 2.7.2) [29]. See võimaldab käitada Pythoni koodi Java virtuaalmasinal, mis tähendab, et puudub vajadus Pythoni versiooni installeerimiseks.

### 2.2.7 Gradle

Rakenduse arendamiseks on kasutatud erinevaid tehnoloogiaid ja tööriistu ning nende kasutamise lihtsustamiseks ja omavaheliseks ühendamiseks on kasutatud rakenduse loomise protsesse automatiseerivat tööriista nimega Gradle [30]. Selle abil saab automatiseerida selliseid tegevusi nagu rakenduse lähtekoodi kompileerimist, testimist ning pakendamist terviklikuks rakenduseks. Samuti võimaldab see lihtsasti installeerida rakenduse arendamisel kasutatavaid tööriistu: lisades faili „build.gradle“ vastava tööriista nime ja versiooni, laadib Gradle selle automaatselt alla ning tööriist ongi kasutamiseks valmis. Gradle'i abil on genereeritud ka rakenduse JAR-fail. See sisaldab kõiki rakenduse tööks vajalikke pakette ja teeke ning JDK versiooni 11 või uuema olemasolul on võimalik selle JAR-faili kaudu rakendus käivitada.

## 2.3 Kasutajaliides

Järgnevalt kirjeldatakse rakenduse graafilise kasutajaliidese toimimist tavapärase rakenduse kasutamise töökäigu põhjal. Kasutajaliidese loomisel lähtuti sellest, et rakenduse kasutamine oleks võimalikult lihtne, seega on selle erineva funktsionaalsuse hulka püütud minimeerida. Samuti on rakendus disainitud selliselt, et selle kasutamist saaks hõlpsalt integreerida praegu programmeerimise algkursustel kasutusel olevate vahenditega.

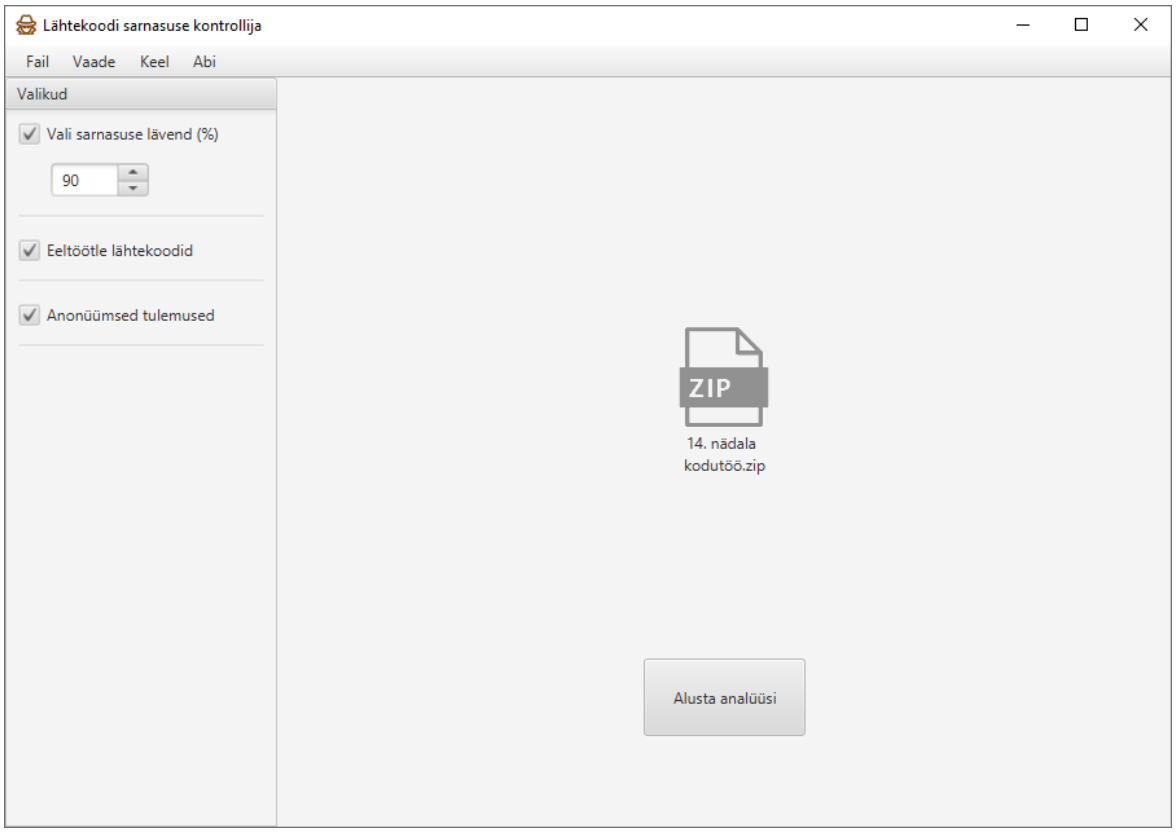
Rakenduse kasutajaliidese saab jagada kolmeks kasutajavaateks: peavaade, tulemuste vaade ja koodivaade. Vaated on loodud rakendusega SceneBuilder, millega on genereeritud kõigi kolme vaate jaoks FXML-failid. Rakenduse käivitamisel ning vaadete vahetamisel laaditakse kasutajaliidese aknasse FXML-failiga defineeritud vaate sisu. Järgnevates alapeatükkides kirjeldatakse neid kolme kasutajavaadet ning ka rakenduse menüüriba lähemalt.

### 2.3.1 Peavaade

Rakenduse käivitamisel avaneb peavaade (vt joonis 3), kus toimub sarnasuse analüüsi, mida lähemalt kirjeldatakse alapeatükis 2.4, seadistamine ja käivitamine. Joonisel 3 on näha peavaate vasakus ääres menüü „Valikud“, kus on sarnasuse analüüsi kohandamiseks kolm valikuvõimalust, mida on lähemalt kirjeldatud tabelis 1.

Tabel 1. Valikute menüü seaded

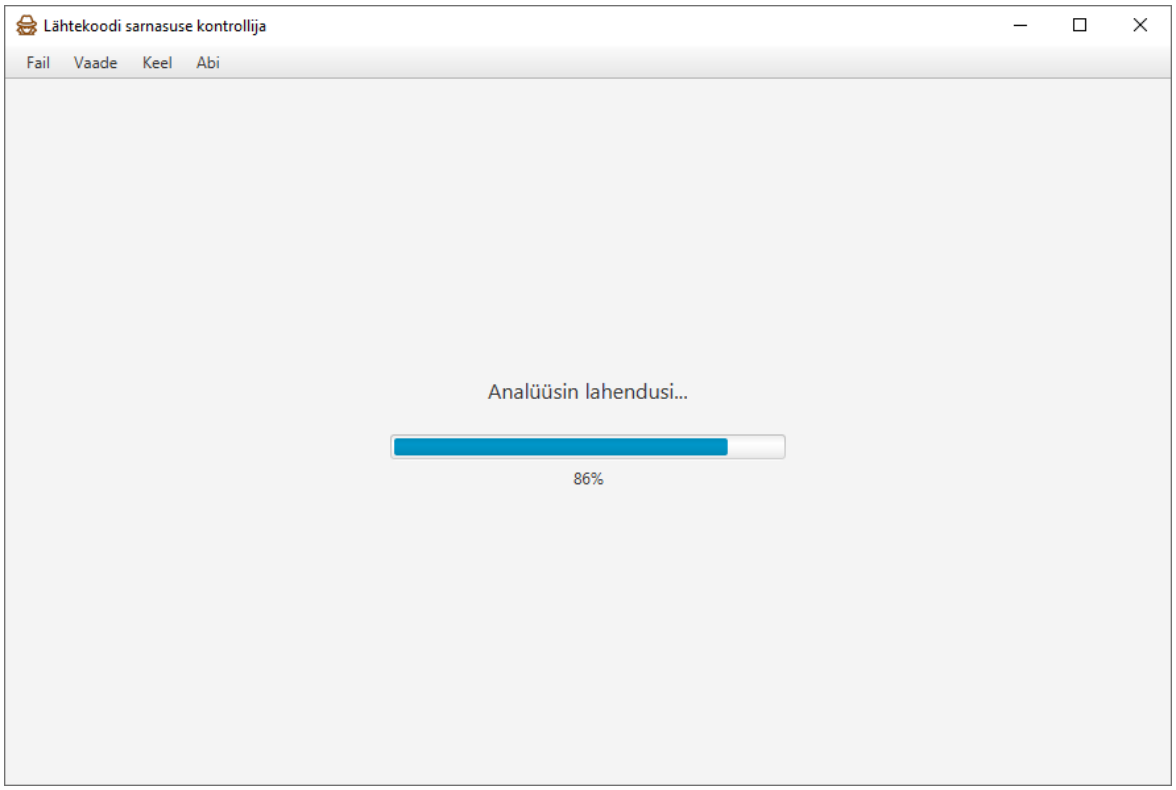
| | |
|---|---|
| **Vali sarnasuse lävend (%)** | Kasutaja saab valida analüüsis kasutatava sarnasuse lävendi protsendina. Seda valikut mitte valides arvutatakse sarnasuse lävend automaatselt sisendiks antavate programmide lähtekoodide keskmise pikkuse põhjal. |
| **Eeltöötle lähtekoodid** | Kasutaja saab valida, kas lähtekoodid eeltöödeldakse enne analüüsi alustamist või kasutatakse analüüsil eeltöötlemata lähtekoode. |
| **Anonüümsed tulemused** | Kasutaja saab valida, kas koodivaates näidatakse sarnaste programmide autorite nimesid või autorite unikaalseid 6-kohalisi kasutajanumbreid. |




Joonis 3. Rakenduse peavaade

Sarnasuse analüüsi seadistamise järel peab valima sarnasuse kontrolliks antavaid programme sisaldava ZIP-faili. Kuna Moodle'i VPL moodulit kasutatakse laialdaselt programmeerimise algkursustel ülesannete lahenduste esitamise keskkonnana ja see võimaldab

lihtsasti alla laadida esitatud lahendused ühe ZIP-failina, siis on rakendus loodud töötama just VPL-i poolt genereeritud ZIP-faili kujul oleva sisendiga. ZIP-faili sisemist kaustastruktuuri kirjeldatakse lähemalt jaotises 2.4.1. Rakenduse kasutamiseks ongi vaja esmalt alla laadida Moodle'ist VPL-vahendi kaudu esitatud programmide ZIP-fail ning see programmile sisendiks anda. Sisendfaili valimise järel saab alustada programmide sarnasuse analüüsimise protsessi, vajutades nupule „Alusta analüüsi" (vt joonis 3).



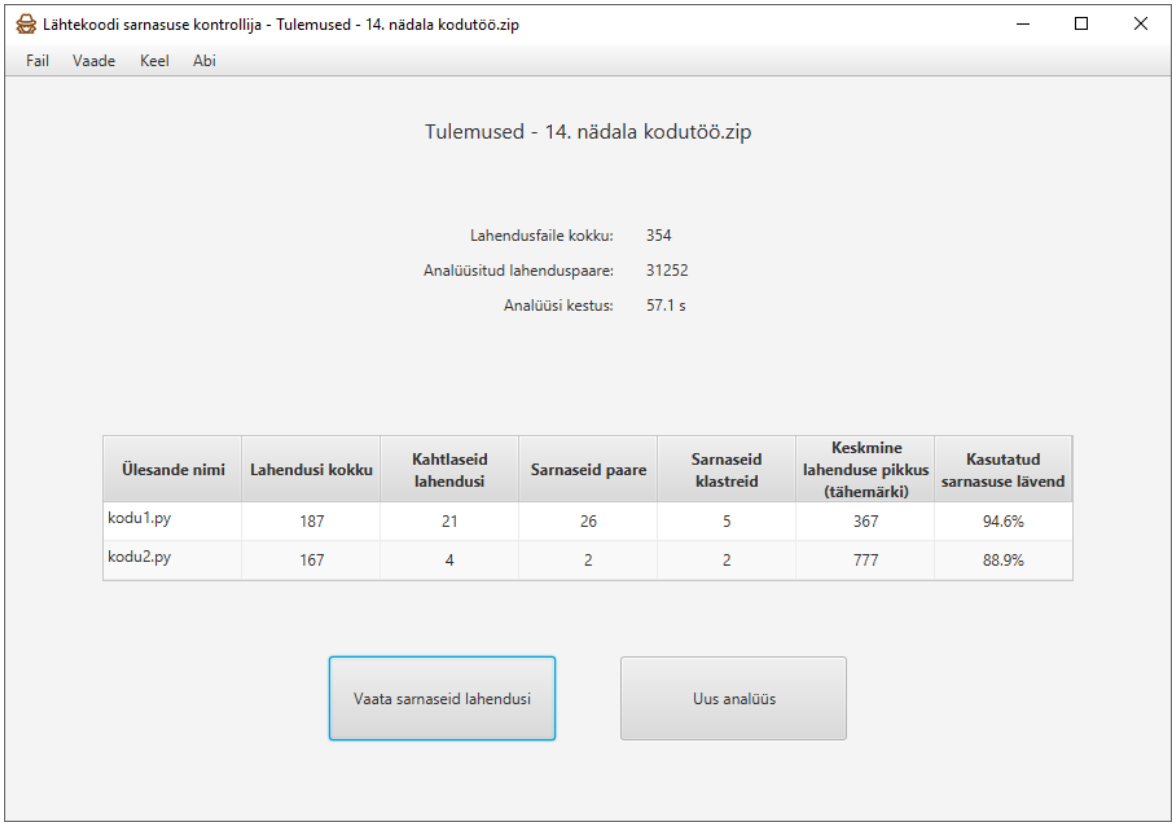

Joonis 4. Sarnasuse analüüsi edenemise vaade

Sarnasuse analüüsi alustamisel ilmub analüüsi edenemise vaade (vt joonis 4), kust on võimalik jälgida analüüsi edasikulgu ning see annab infot, missugune sarnasuse analüüsi etapp parasjagu töötab.

### 2.3.2 Tulemuste vaade

Sarnasuse analüüsi edukal lõppemisel avaneb tulemuste vaade (vt joonis 5), kus on välja toodud statistikat analüüsi kohta ning tulemusi kirjeldav tabel. Statistika sisaldab infot selle kohta, kui palju oli analüüsitud lahendusfaile kokku, mitu lahenduspaaride kombinatsiooni läbi vaadati ning kui kaua analüüs kestis. Tulemuste tabel näitab ülesannete kaupa informatsiooni analüüsi tulemuste kohta. Tabeli seitsmes tulbas on välja toodud järgnev info (vt joonis 5):

1. ülesande nimi;
2. mitu erinevat autorit esitas ülesandele oma lahenduse;
3. mitu esitatud lahendust luges sarnasuse analüüs kahtlasteks;
4. mitu sarnast paari analüüs leidis;
5. mitu klastrit moodustus leitud paaridest;
6. ülesandele esitatud lahenduste lähtekoodide keskmine pikkus tähemärkides;
7. analüüsis programmipaaride võrdlemisel kasutatud sarnasuse lävend.



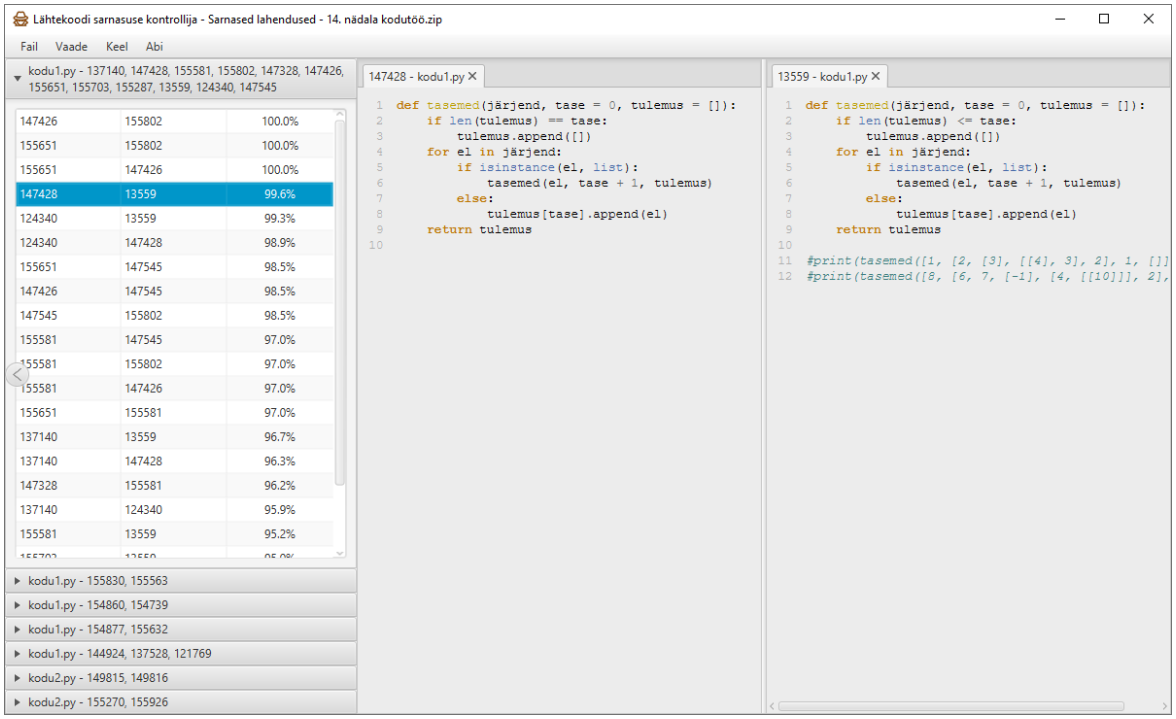

Joonis 5. Tulemuste vaade

Tulemuste vaate alumises ääres on kaks nuppu (vt joonis 5). „Uus analüüs“ viib kasutaja tagasi peavaatesse, kus on võimalik teha uus analüüs erineva seadistusega või uute sisendfailidega. „Vaata sarnaseid lahendusi“ avab uues aknas koodivaate, kus saab lähemalt uurida leitud klastreid ja sarnaseid paare ning näha lahenduste lähtekoode.

### 2.3.3 Koodivaade

Koodivaate vasakus ääres (vt joonis 6) on ära toodud rakenduse analüüsi käigus leitud klastrid, mille nimed koosnevad ülesande nimetusest ning sellesse klastrisse kuuluvate autorite nimedest (anonüümsete tulemuste valimisel autorite kasutajanumbritest). Klastri peale

vajutades avaneb selle all klastrisse kuuluvate sarnaste paaride loetelu, mis on sorteeritud paaride sarnasuse järgi kahanevalt. Loetelus paari peal topeltklõpsates avanevad lahenduste eeltöötlemata kujul olevad lähtekoodid koodiakendes. See võimaldab analüüsi käigus sarnaseks loetud programmid manuaalselt üle kontrollida. Korraga saab avada ka rohkem kui ühe paari lähtekoodid.




Joonis 6. Koodivaade

Koodi süntaksi värviliselt esiletõstmine on tehtud Pythoni pistikmooduli Pygments [31] abiga, mis genereerib etteantud lähtekoodi põhjal HTML-faili, kuhu on lisatud märgistuskeelega CSS värviline vormindus. Loodud HTML-fail on kuvatud JavaFX-elemendiga „WebView“, mis võimaldab HTML-faile sarnaselt veebibrauseritele näidata.

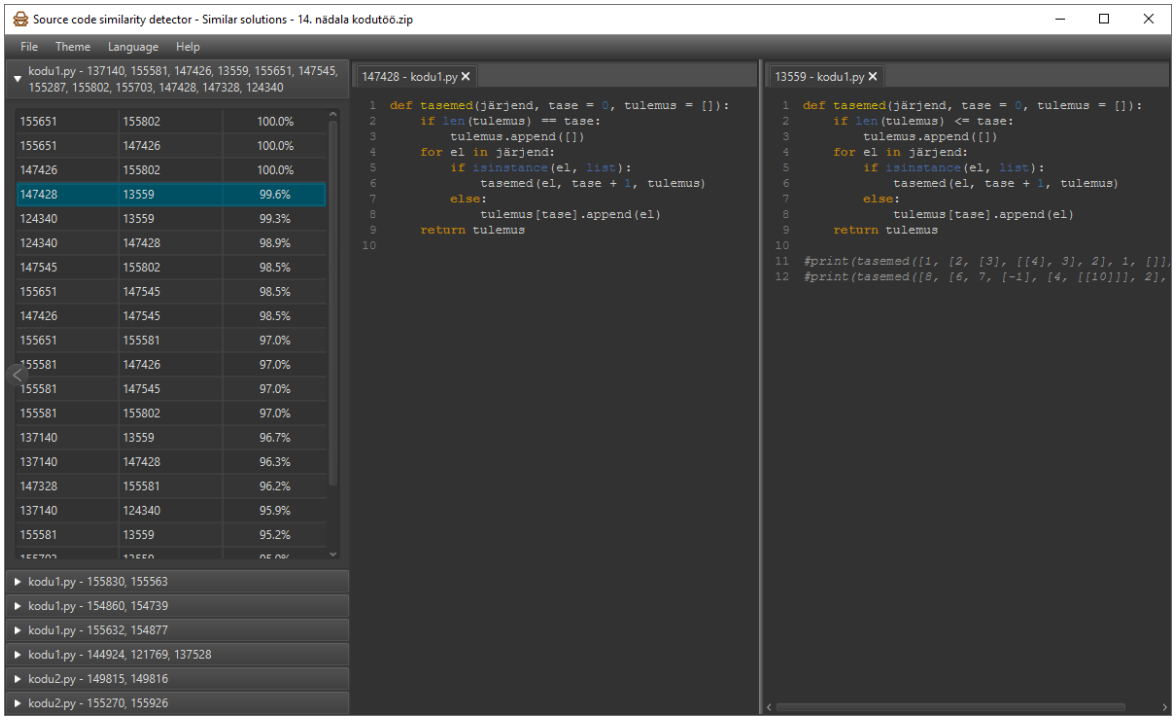
### 2.3.4 Menüüriba

Rakenduse menüüribal (vt joonis 7) on neli menüüd, mille funktsioone on kirjeldatud tabelis 2. Menüüdes pole palju erinevaid funktsioone, sest põhilised tegevused saab ära teha otse kasutajavaadetes. Lisafunktsionaalsusena on menüüdes juures võimalus vahetada rakenduse kuvakeelt ning kasutajaliidese taustavärvi. Joonisel 8 on näidatud rakenduse kasutajaliidest tumeda taustaga ning ingliskeelsena.

Tabel 2. Rakenduse menüüriba funktsioonide kirjeldus

| | |
|---|---|
| **Fail** | Menüüst saab rakenduse sulgeda ning koodivaates olles lisandub menüüsse ka võimalus sulgeda kõik avatud koodiaknad. |
| **Vaade** | Võimaldab valida kasutajaliidese heleda ja tumeda taustaga vaate vahel. Vaikimisi on kasutajaliides heleda taustaga. |
| **Keel** | Saab muuta rakenduse kuvakeelt kas eesti- või ingliskeelseks. Vaikimisi on rakendus eestikeelne. |
| **Abi** | Saab lugeda rakenduse kohta käivat teavet. |


Joonis 7. Rakenduse menüüriba



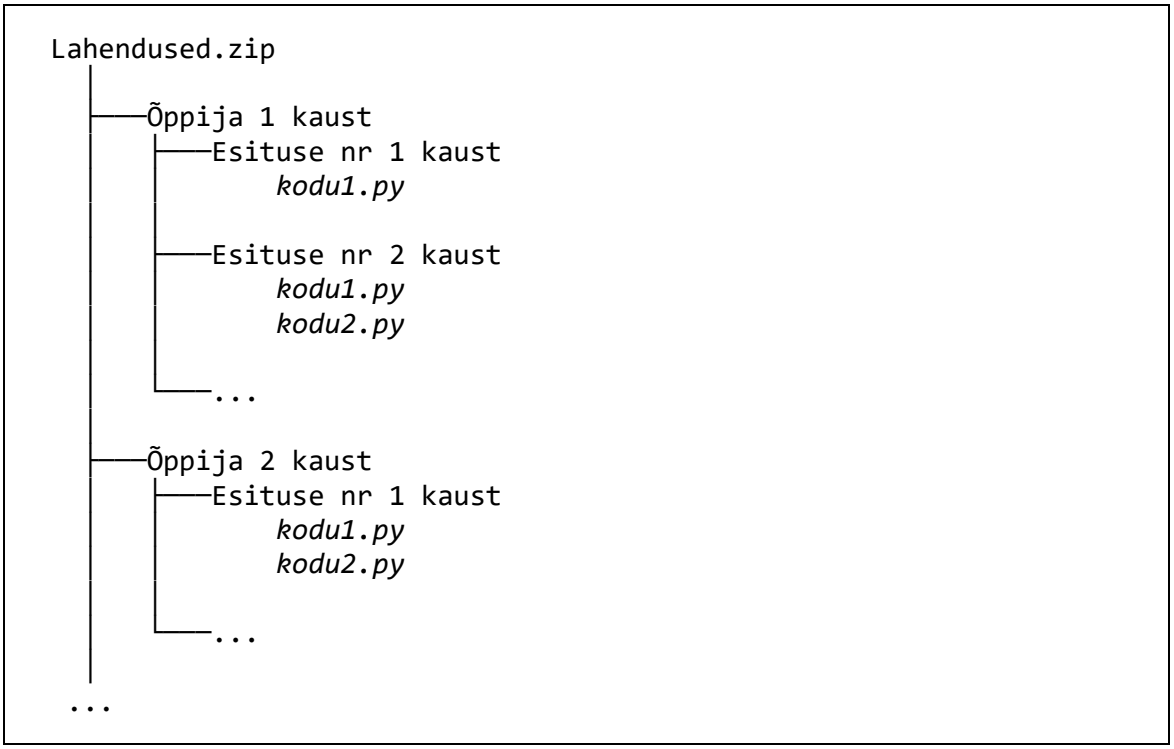

Joonis 8. Rakenduse kasutajaliides tumeda taustaga ja ingliskeelsena

## 2.4 Programmide sarnasuse analüüsimine

Peatükis kirjeldatakse rakenduse tagasüsteemis programmikoodide sarnasuse analüüsi käigus toimuvaid protsesse. Need protsessid saab jagada kolmeks: sisendi töötlemine, programmide sarnasuse võrdlemine ja sarnaste programmide klasterdamine. Järgnevates alapunktides kirjeldatakse neid kolme protsessi lähemalt.

### 2.4.1 Sisendi töötlemine

Sarnasuse analüüsi käivitamisel on esimene tegevus kasutaja antud sisendfaili töötlemine. Sisendiks saab anda Moodle'i VPL-mooduli poolt genereeritud ZIP-faili, kus on kõik õppijate esitatud lahendused ülesannetele. ZIP-faili struktuuri kirjeldab joonis 9. Jooniselt on näha, et see sisaldab iga õppija kohta kausta, mille sees on üks või enam esituse kausta (ülesannetele saab lahendusi esitada mitu korda), kus on ülesannete lahendusteks esitatud programmid. Programmeerimise algkursustel on ühes esitatavas töös keskmiselt kolm ülesannet ning iga ülesande lahenduseks esitatakse üks fail, mille nimi on ette määratud. See tähendab, et kui näiteks kodutöös on kolm ülesannet, siis igas esituse kaustas on kuni kolm faili, mis on tavaliselt nimedega „kodu1.py", „kodu2.py" ja „kodu3.py". Tegelikkuses sisaldavad õppijate kaustad rohkem alamkaustasid ja faile, kuid analüüsi jaoks on vajalikud vaid õppijate viimase esituse kaustas olevad lahendusfailid.




Joonis 9. Lahenduste ZIP-faili struktuuri näide

Lahenduste ZIP-failist eraldatakse iga õppija viimases esituses olevad lahendusfailid, mis jagatakse ära ülesannete kaupa. Kui kasutaja valis peavaates ka lähtekoodide eeltöötlemise valiku, siis tehakse selles etapis ka lahendusprogrammide lähtekoodide eeltöötlus Pythoni skriptiga, mis eemaldab koodist tühjad read ning kommentaarid. Nende eemaldamine on põhjendatud sellega, et tühjad read ning kommentaarid ei mõjuta kuidagi programmi tööd selle käivitamisel. Eeltöödeldud lähtekoodid salvestatakse uute failidena, et alles jääks ka originaalfailid, mida hiljem kasutatakse koodivaates näitamiseks.

Juhul kui kasutaja ei valinud peavaates lahenduste sarnasuse võrdlemisel kasutatavat sarnasuse lävendit, siis arvutatakse sisendfailide töötlemise käigus ka see. Sarnasuse lävend arvutatakse iga ülesande kohta eraldi järgneva valemi järgi:

$$\text{sarnasuse lävend} = 0.98^{\,0.01x}$$

Valemis tähistab *x* ülesande lahenduskoodide keskmist pikkust tähemärkides. Tegemist on eksponentsiaalse kahanemise valemiga [32], mille järgi väheneb sarnasuse lävend lahenduste keskmise pikkuse suurenemisel. Lävendiks saadakse arv vahemikus (0; 1).

Sarnasuse lävendi arvutamiseks kasutatava valemi valimisel oli põhiline kriteerium see, et lühemate lahendustega ülesannete puhul oleks lävend kõrgem ja pikemate puhul madalam. Põhjuseks on see, et lühemate ja lihtsamate ülesannete lahendamiseks on tavaliselt vähem erinevaid võimalusi. See tähendab, et selliste ülesannete puhul on suurem tõenäosus, et võib esineda juhuslikke sarnasusi lahenduskoodides, kuid seetõttu võivad need ka omavahelisel võrdlemisel saada üsna suure sarnasuse. Sellepärast ongi lühemate lähtekoodide võrdlemisel mõistlikum kasutada suuremaid sarnasuse lävendeid, et mingil määral välja filtreerida juhuslikust kokkulangevusest põhjustatud sarnasus. Eksponentsiaalse kahanemise valem täidab kirjeldatud tingimusi sarnasuse lävendi arvutamisel hästi. Lisaks on selle valemi puhul hea veel see, et mida suuremaks läheb valemis muutuja *x* väärtus, seda aeglasemalt väheneb sarnasuse lävendi väärtus. See tähendab, et pikemate lahenduskoodidega ülesannete puhul ei jõua lävendi väärtus nii madalale, et loetaks põhjendamatult palju programme omavahel sarnaseks. Valemis kasutatavate konstantide valimine toimus põhiliselt erinevate konstantide katsetamise läbi. Lõpuks said valitud sellised konstandid, mille põhjal arvutatud sarnasuse lävendid andsid autori hinnangul kõige täpsemaid sarnasuse analüüsi tulemusi.

Sisendprogrammide töötlemisel ei parsita lähtekoode lekseemideks, mida teevad paljud olemasolevad plagiaadituvastussüsteemid, vaid jäetakse need sõne kujule. Lekseemideks

teisendamise eelis on, et programmide võrdlus lekseemide põhjal võtab üldjuhul vähem aega, kui sõne kujul olevate lähtekoodide võrdlemine. Samas on lekseemideks teisendamisel oht kaotada ära liiga palju informatsiooni programmi kohta ning lõppkokkuvõttes võib see sarnasuse analüüsi tulemuste täpsust langetada (seda probleemi kirjeldati lähemalt ka jaotises 1.3). Autorile pakubki seejuures huvi, kas lähtekoode sõnedena võrreldes on võimalik saada täpsemaid tulemusi kui lekseemide võrdlemise põhjal ning kas see ka õigustab analüüsi pikemat tööaega.

### 2.4.2 Programmide sarnasuse võrdlemine

Pärast sisendi töötlust toimub programmide sarnasuse võrdlemine, mis tehakse ülesannete kaupa. Selle käigus võrreldakse ülesande kõiki lahendusfaile omavahel paarikaupa, et leida nende hulgast sarnased paarid. Selline paarikaupa võrdlemine on ruutkeerukusega, sest tuleb võrrelda $n \cdot m$ programmipaari ($n$ on võrreldavate programmide arv). See tähendab, et kui ülesandele on esitatud 300 lahendust, siis tuleb sarnasuse võrdlemise käigus kokku võrrelda 44850 lahenduste paari. Seetõttu on see etapp ka sarnasuse analüüsi protsessidest kõige ajakulukam.

Programmipaari võrdlemine põhineb kahe sõnena esitatud lähtekoodi vahelise Levenshteini kauguse leidmisel. Levenshteini kauguse mitteformaalne definitsioon on järgnev: „Kahe sõne vaheline Levenshteini kaugus on vähim tähemärgimuudatuste arv, mida on vaja teha selleks, et ühte sõne teiseks muuta“ [18]. Võimalikeks muudatusteks on tähemärgi lisamine, kustutamine ja asendamine. Levenshteini kauguse leidmise algoritmi baasimplementatsioonis on iga tehtava muudatuse hinnaks 1, mis tähendab, et ühe muudatuse tegemisel suureneb Levenshteini kaugus ehk muudatuste koguhind 1 võrra. Rakenduses kasutatavas implementatsioonis kehtivad muudatuste hindade kohta järgnevad lisareeglid:

1. Jutumärkide ( " ) ja ülakoma ( ' ) omavahel vahetamise hinnaks on 0.2;
2. Väike- ja suurtähtede vahetamise hinnaks on 0.2;
3. Eesti tähestiku eritähtede (õ, ä, ö, ü, š, ž) vahetamine inglise tähestiku alternatiivide või numbritega on hinnaga 0.2;
4. Tühiku lisamine või eemaldamine on hinnaga 0.5.

Loetletud muudatused on ühed lihtsamini läbi viidavad leksilised muudatused, mida lähtekoodi muutmisel teha saab, ilma et programmi töökäik oleks mõjutatud. Seetõttu on nendele muutmistele antud ka väiksemad kaalud. Selle tulemusel peaks kirjeldatud muudatusi kasutades plagieeritud lähtekoodide võrdlemisel tulema Levenshteini kaugus väiksem kui baas-

implementatsiooni puhul. Nende muudatuste puhul võib küll väita, et sageli väljendavad need hoopis autori erinevat stiili, mitte niivõrd plagieerimisvõtteid. Samas on rakenduse eesmärk leida omavahel sarnased programmid, mis võivad potentsiaalselt olla plagieeritud, kuid lõpliku otsuse selle osas, kas tegemist on plagiaadiga või mitte, teeb ikkagi rakenduse kasutaja. Enne selle otsuse tegemist saab ta üle vaadata lahenduste koodid ning siis otsustadagi, kas tegemist on autorite stiili erinevusega või siiski ühe autori plagieerimiskatsega.

Maksimaalseks võimalikuks Levenshteini kauguseks kahe sõne vahel on pikema sõne pikkus tähemärkides ning see on võimalik vaid sel juhul, kui sõned on täiesti erinevad. Juhul kui aga sõned on identsed, on kauguseks 0. Levenshteini kauguse leidmise algoritm on samuti ruutkeerukusega, kuna minimaalse muutmiskauguse leidmiseks tuleb läbi proovida tähemärgi muudatused $n \cdot m$ erinevas kohas ($n$ ja $m$ on vastavalt esimese ja teise sõne pikkused tähemärkides).

Rakenduses kasutatavas algoritmi implementatsioonis on tööaja vähendamiseks tehtud kaks optimeerimist. Esiteks eemaldatakse võrreldavatest sõnedest nende ühine algus- ja lõpposa. Näiteks sõnede „sepistama“ ja „sepitsema“ puhul jääb ühise algus- ja lõpposa eemaldamisel sõnedest järele „sta“ ja „tse“. Levenshteini kaugus tuleb aga siiski mõlema sõnepaari võrdlemisel sama. Vahe seisneb aga selles, et lühemate sõnede omavahel võrdlemine tehakse oluliselt kiiremini, sest algoritm on ruutkeerukusega. Teiseks on lisatud muudatuste koguhinna lävend, mis saadakse korrutades paari pikema lähtekoodi pikkuse sarnasuse lävendiga. Kui algoritmi töö käigus ületab tehtud muudatuste koguhind selle lävendi, siis katkestatakse algoritmi töö, sest sarnasuse lävendi põhjal ei loetaks neid programme nagunii enam sarnaseks.

Levenshteini kaugus põhimõtteliselt näitab, kui erinevad on kaks sõne ehk teisisõnu, mida suurem on kahe sõne vaheline Levenshteini kaugus, seda erinevamad need sõned on. Selleks, et saada Levenshteini kaugusest programmide vahelist sarnasust, on kasutatud järgnevat valemit:

$$\text{sarnasus}(A, B) = 1 - \frac{\text{lev}(A, B)}{\max(|A|, |B|)}$$

Valemis kasutatavad tähised tähendavad järgnevat:

- $\text{sarnasus}(A, B)$ – programmide $A$ ja $B$ vaheline sarnasus (väärtus lõigus [0; 1])
- $\text{lev}(A, B)$ – programmide $A$ ja $B$ vaheline Levenshteini kaugus
- $\max(|A|, |B|)$ – programmide $A$ ja $B$ lähtekoodidest pikema pikkus tähemärkides

Valemi järgi saadud sarnasust võrreldakse seejärel sarnasuse lävendiga. Kui programmipaari sarnasus on suurem kui sarnasuse lävend või sellega võrdne, siis loetakse paar sarnaseks ning salvestatakse sarnaste paaride listi. Sellisel viisil võrreldakse kõiki programmipaare ning leitakse nende hulgast sarnased paarid.

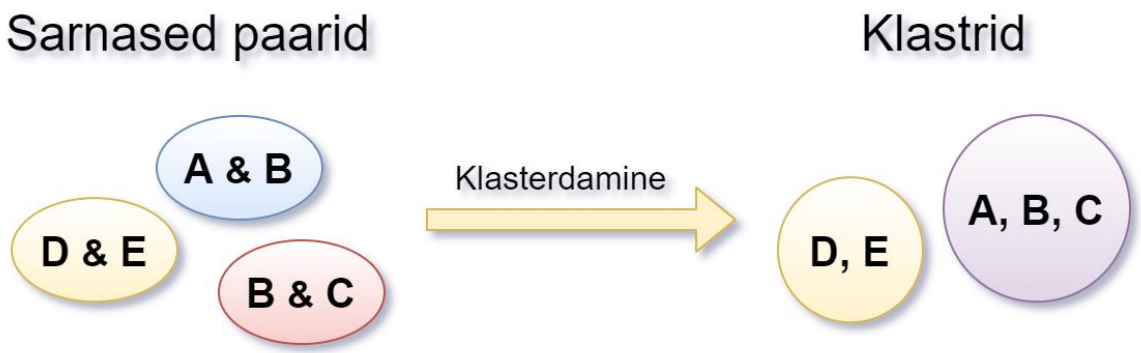
### 2.4.3 Sarnaste programmide klasterdamine

Sarnaste programmipaaride leidmise järel toimub leitud paaride põhjal programmide klasterdamine. Selle eesmärgiks on grupeerida sarnased programmid paaridest suurematekss gruppideks, et oleks võimalik tuvastada õppijate grupid, kes võivad olla omavahel potentsiaalselt plagieerinud, sest plagieerimine ei toimu alati vaid kahe õppija vahel.

Klasterdamine toimub selliselt, et leitud sarnaseid paare vaadatakse listis ühekaupa läbi ning iga paari puhul toimitakse järgneva loogika järgi.

1. Kui kumbki paaris olev programm ei kuulu veel ühtegi olemasolevasse klastrisse või ühtegi klastrit veel ei ole, siis luuakse uus klaster ja lisatakse sinna need kaks programmi.
2. Kui üks programm juba kuulub mingisse klastrisse, siis lisatakse sellesse klastrisse ka teine programm.
3. Kui mõlemad programmid juba kuuluvad klastritesse, kuid need klastrid on erinevad, siis ühendatakse need klastrid kokku üheks klastriks.

Klasterdamise üldine põhimõte seisneb selles, et kui on programmid A, B ja C ning A ja B on üks sarnane paar ning B ja C teine sarnane paar, siis A, B ja C kuuluvad kõik samasse klastrisse. Joonis 10 illustreerib seda, kuidas klasterdamise tulemusel moodustuvad sarnastest programmipaaridest klastrid.




Joonis 10. Sarnaste paaride põhjal lahendusprogrammide klasterdamine
(A, B, C, D ja E tähistavad erinevaid lahendusprogramme)

Klastrite moodustamisega lõpeb tagasüsteemis programmide sarnasuse analüüsimine. Edasi kuvatakse juba tulemusi rakenduse kasutajaliideses, millest kirjutati täpsemalt ülalpool.

# 3 Tulemuste analüüs

Lõputöö käigus valmis töölauarakendus õppijate esitatud lahendusprogrammide sarnasuse tuvastamiseks. Antud peatükis esmalt analüüsitakse valminud rakenduse vastavust seatud nõuetele. Seejärel hinnatakse loodud rakenduse kvaliteeti, testides seda reaalsete lahendusprogrammidega. Viimaks analüüsitakse ka sarnasuse võrdlemiseks kasutatud algoritmi eeliseid ja puuduseid.

## 3.1 Rakenduse vastavus seatud nõuetele

Selles alapeatükis uuritakse, kas enne arendustegevuse alustamist seatud nõuded rakendusele said täidetud. Nõuetele vastavust kontrollitakse peatükis 2.1 esitatud punktide kaupa.

**Lähtekoodide sarnasuse võrdlemise algoritmi põhifookuseks on tuvastada leksilistel muudatustel põhinevat plagieerimist. –** Leksiliste muudatuste tegemine üldiselt ei muuda programmi struktuuri ning seetõttu kasutatakse tavaliselt sellistel muudatustel põhineva plagieerimise tuvastamiseks programmi struktuuri võrdlemise meetodeid. Need meetodid põhinevad sageli sõnevõrdlusalgoritmide kasutamisel. Loodud rakenduse sarnasuse võrdlemise algoritm baseerub Levenshteini kaugusel, mis on samuti sõnevõrdlusalgoritm, seega sobib rakenduse algoritm hästi leksilistel muudatustel põhineva plagiaadi tuvastamiseks.

**Lähtekoodide eeltöötlemisel tohib algsest informatsioonist eemaldada minimaalse osa. –** Programmide eeltöötlemise faasis eemaldatakse lähtekoodidest vaid tühjad read ja koodis olevad kommentaarid (vt peatükk 2.4.1). Nendel ei ole programmis funktsionaalset otstarvet ehk nad ei mõjuta kuidagi programmi töökäiku. Seetõttu ei lähe nende eemaldamisel lähtekoodist kaduma sarnasuse võrdlemise jaoks olulist informatsiooni.

**Rakendus peab suutma võrrelda programmeerimiskeeles Python kirjutatud programmide lähtekoode. –** Sarnasuse võrdlemise algoritm võrdleb lähtekoode sõne kujul, seega saaks antud algoritmiga võrrelda tegelikult ükskõik missuguses programmeerimiskeeles kirjutatud lähtekoode. Programmide eeltöötlus tuleb aga teha vastavalt programmeerimiskeele reeglitele, seega on iga lisatava keele jaoks vaja eraldi implementeerida lähtekoodide eeltöötlemise loogika. Rakenduse praeguses versioonis ongi seda tehtud vaid programmeerimiskeele Python jaoks.

**Rakendus peab analüüsima ZIP-faili kujul olevat sisendit, mis genereeritakse Moodle'i VPL pistikmooduli poolt, kusjuures tuleb arvestada genereeritava ZIP-faili sisese kaustastruktuuriga. –** Rakendus ongi loodud töötama just VPL-i poolt genereeritava

lahenduste ZIP-faili kujul oleva sisendiga. Peatükis 2.4.1 kirjeldati sisendi töötlemist ja kasutamist lähemalt.

**Rakendus peab teostama sarnasuse kontrolli paarikaupa kõigi sisendiks antud lähtekoodide vahel.** – Sarnasuse kontroll tehakse ülesannete kaupa ning võrreldakse igale ülesandele esitatud lahendusprogramme omavahel paarikaupa (vt peatükk 2.4.2).

**Rakendus näitab sarnasuse analüüsi tulemusena leitud sarnaste lahendusprogrammide paare ning võimalusel grupeerib leitud paarid suurematesse gruppidesse. –** Kasutajaliidese koodivaatest on võimalik vaadata analüüsi käigus leitud sarnaseid lahendusprogrammide paare (vt peatükk 2.3.3) ning samuti on nende paaride põhjal sarnased programmid klasterdatud (vt peatükk 2.4.3).

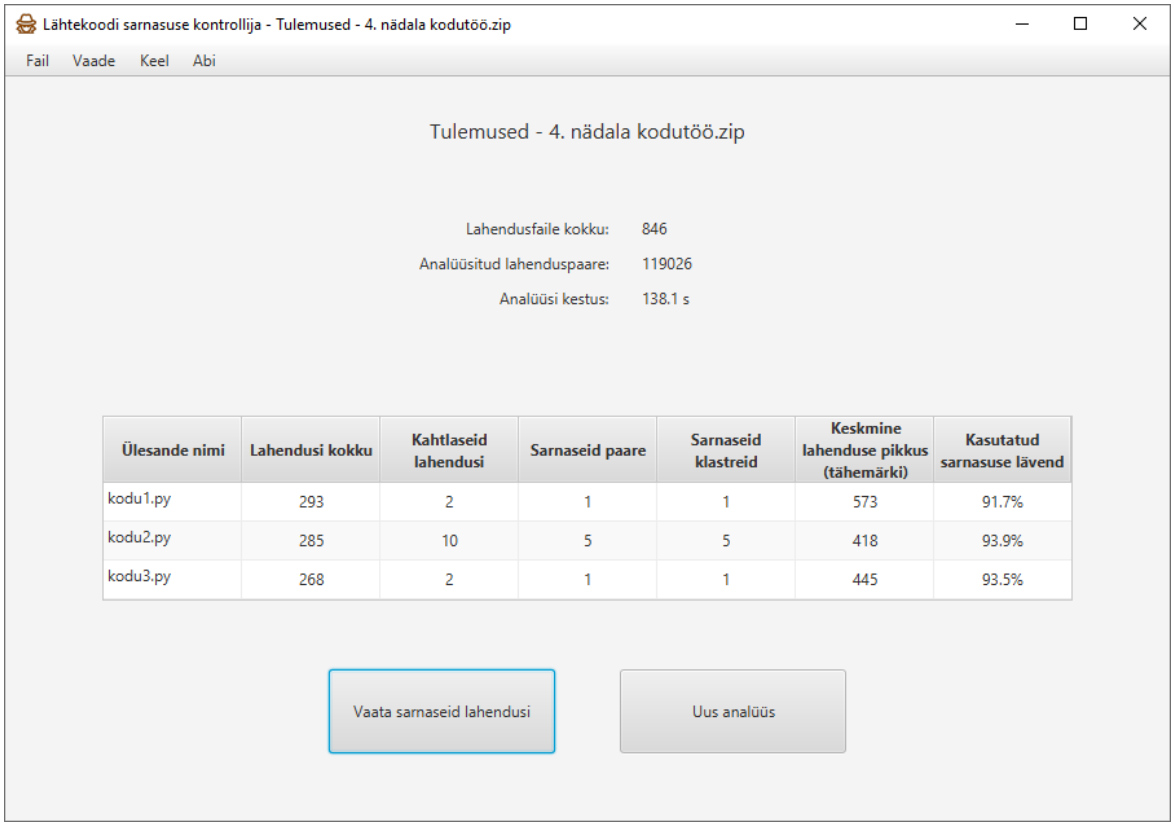
**Rakenduse analüüsi tulemustest on võimalik välja lugeda leitud sarnaste lahenduste autorid.** – Rakenduse koodivaates näidatakse tuvastatud sarnaste programmipaaride autorite nimesid (vt peatükk 2.3.3) ning samuti on klastrite nimedest võimalik välja lugeda sinna kuuluvate programmide autorite nimed.

**Sarnasuse analüüs kuni 300 õppija esitusega, kus iga õppija esituse hulgas on keskmiselt kolme eri ülesande lahendusfailid, ei tohi võtta kauem aega kui 10 minutit. –** Sarnasuse analüüsi ajakulu testiti õppeaine „Programmeerimine“ ühele kodutööle esitatud lahendustega. Kodutöö koosnes kolmest ülesandest ning igale ülesandele oli lahenduse esitanud ligi 300 õpilast. Joonisel 11 on näidatud sarnasuse analüüsi tulemused selle kodutöö lahenduste võrdlemisel. Analüüs võttis aega 138 sekundit ehk 2.3 minutit, seega 4 korda kiiremini kui maksimaalse ajakulu limiidiks seatud 10 minutit.

Sama kodutöö lahendustega testiti ka rakenduse põhimälu (RAM) ja protsessoriaja (CPU) kasutust. Seda tehti programmi SysGauge [33] abil, millega saab protsesside ressursikulu monitoorida. Joonisel 12 on väljavõte programmist SysGauge rakenduse mälukasutuse kohta. Maksimaalne mälukasutus antud kodutöö ülesannete analüüsimisel oli 0.5 GB ning keskmine 0.4 GB, mis on suhteliselt madal, sest tänapäeva arvutitel on põhimälu enamasti vähemalt 4 GB. Joonisel 13 on SysGauge'i poolt mõõdetud rakenduse protsessoriaja kasutus. Keskmiselt kasutas rakendus 13% ning maksimaalselt 17% protsessori täisvõimsusest. Testimisel kasutatava arvuti protsessor oli Intel® Core™ i5-8500[9], mis on 6-tuumaline

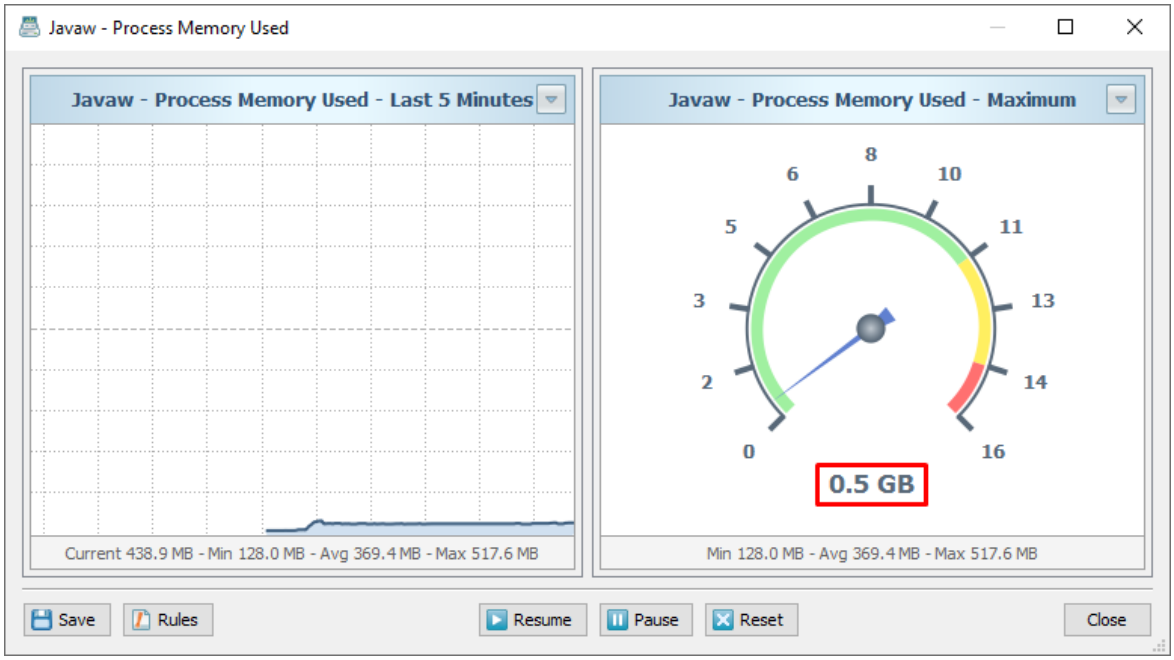
[9] https://cpu.userbenchmark.com/SpeedTest/447884/IntelR-CoreTM-i5-8500-CPU---300GHz

protsessor taktsagedusega 3.00 GHz. Võib öelda, et ka protsessorikasutus rakenduse poolt on üsna madal ning ka aeglasemate protsessorite puhul ei tohiks jõudlusprobleeme tekkida.

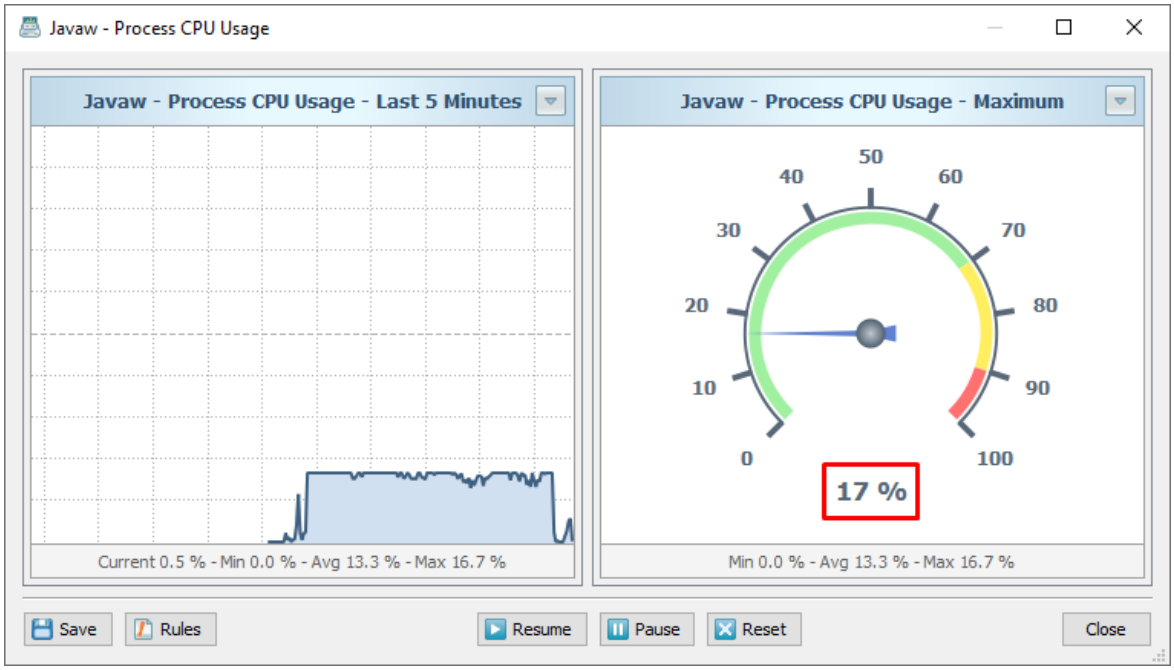



| Ülesande nimi | Lahendusi kokku | Kahtlaseid lahendusi | Sarnaseid paare | Sarnaseid klastreid | Keskmine lahenduse pikkus (tähemärki) | Kasutatud sarnasuse lävend |
|---|---|---|---|---|---|---|
| kodu1.py | 293 | 2 | 1 | 1 | 573 | 91.7% |
| kodu2.py | 285 | 10 | 5 | 5 | 418 | 93.9% |
| kodu3.py | 268 | 2 | 1 | 1 | 445 | 93.5% |

Joonis 11. Sarnasuse analüüsi tulemused 3 × 300 lahendusega




Joonis 12. Rakenduse mälukasutus 3 × 300 lahendusega testimisel




Joonis 13. Rakenduse protsessoriaja kasutus 3 × 300 lahendusega testimisel

Nõuete analüüsist järeldub, et kõik eelnevalt seatud nõuded on valminud rakenduses täidetud. Samuti on rakenduse ressursikulu küllaltki väike, seega jõudlusprobleeme antud rakenduse kasutamisel ei tohiks esineda.

## 3.2 Rakenduse sarnasuse tuvastamise kvaliteedi hindamine ühe ülesande lahendusprogrammide põhjal

Valminud rakendus on mõeldud praktiliseks kasutamiseks programmeerimise algkursustel õppejõude abistava vahendina plagieerimise tõkestamisel. Seetõttu on vajalik selle testimine reaalselt õppijate poolt esitatud programmide peal, et kindlaks teha, kas rakenduse sarnasuse analüüs annab ka praktilisel kasutamisel soovitud tulemusi ning sobib programmeerimiskursustel kasutamiseks.

### 3.2.1 Testimise metoodika

Rakenduse testimist viidi läbi algajatele mõeldud Pythoni programmeerimiskursuse „Tehnoloogia tarbijast loojaks“ ühe ülesande lahendustega. Tegemist on samade programmidega, mida Marttila bakalaureusetöös [15] kasutati programmide sarnasuse määratlemiseks kursuse mentorite (ehk õppejõudude) poolt. Ülesande nimeks oli *Jukebox* ning Marttila töös on selle ülesande kirjeldus esitatud selliselt:

„Ada tahab valida plaadiautomaadist laulu ja uurib, milliseid laule masin mängib. Muusikapalad on kirjas failis, kus iga laul on eraldi real.

Koostada programm, mis

- küsib kasutajalt failinime (kasutaja sisestab failinime koos laiendiga, nt *jukebox.txt*);
- loeb sisestatud nimega failist andmed;
- näitab kõiki laule koos järjekorranumbritega (alates 1);
- küsib kasutajalt, mitmendat laulu ta soovib (kasutaja sisestab alati täisarvu);
- väljastab ekraanile vastavalt valitud arvule muusikapala.“ [15:16]

Marttila sõnul valiti ülesanne *Jukebox* seetõttu, et kuna tegemist oli 5. õppenädala ülesandega, siis oli see juba piisavalt keeruline ülesanne, et võimalike erinevate lahenduste arv oleks piisavalt suur ning õppijad saaks enda programmeerimisoskusi näidata. Mentorite poolt analüüsimiseks valiti välja kõigi esitatud lahenduste hulgast 6 sarnast programmipaari. Paaride valimisel prooviti leida olemuselt üksteisest võimalikult erinevad paarid. Valitud paarid nimetati nende muutujanimede ja struktuuri sarnasuste ning erinevuste põhjal järgnevalt:

1. “sarnaste muutujanimede, kuid erineva struktuuriga paar;
2. sarnaste muutujanimede ja struktuuriga paar;
3. identsete muutujanimede ja erineva struktuuriga paar;
4. osalt erinevate muutujanimede ja osalt identsete ridadega paar;
5. sarnaste muutujanimede ja identse struktuuriga paar;
6. erinevate muutujanimede ja struktuuriga paar.“ [15:20]

Põhjus, miks käesoleva bakalaureusetöö käigus loodud rakenduse kvaliteedi testimiseks needsamad programmid valiti, on see, et Marttila töös on kursuse „Tehnoloogia tarbijast loojaks“ 11 mentorit andnud nendele programmipaaridele omapoolse hinnangu selle kohta, kuivõrd on nende paaride puhul tegemist plagiaadiga. Rakenduse testimise eesmärk on kindlaks teha, kas rakenduse sarnasuse analüüsi tulemused lähevad kokku kursuse mentorite arvamustega ehk kas rakendus loeb sarnasteks programmipaarid, mida mentorid pidasid plagieerituks, ning erinevaks paarid, mis ka mentorite arvates ei olnud plagieeritud. Seeläbi saab anda rakenduse toimimisele adekvaatse hinnangu.

### 3.2.2 Testimise tulemused

Analüüsi all olid eelmises alapeatükis toodud 6 programmipaari programmid. Kokku oli 11 erinevat programmi, sest üks programm esines kahes paaris. Need lahendusprogrammid anti rakendusele analüüsi sisendiks. Programmide võrdlemisel kasutati sarnasuse lävendiks kõige madalamat võimalikku – 1% – sest siis leitakse kõikide sisendprogrammide vahelised sarnasused. See on vajalik selleks, et teada saada ka rakenduse poolt leitavad sarnasused vähem sarnaste programmide vahel, sest rakenduse tulemustes näidatakse vaid sarnasuse lävendist kõrgema sarnasusega programmipaare. Rakenduse poolt leitud sarnasused ja mentorite hinnangud analüüsitud kuuele programmipaarile on ära toodud tabelis 3.

Tabel 3. Analüüsitud programmipaaridele rakenduse poolt leitud sarnasused ning mentorite hinnangud plagieerimise esinemise kohta nende paaride puhul. Mentorite hinnangu tulba andmed pärinevad Marttila bakalaureusetööst [15].

| **Programmipaar** | | **Rakenduse poolt leitud sarnasus (%)** | **Mentorite hinnang selle kohta, kas on tegemist plagiaadiga (Jah/Ei)** |
|---|---|---|---|
| **1** | Sarnaste muutujanimede, kuid erineva struktuuriga paar | 49.6 | Ei |
| **2** | Sarnaste muutujanimede ja struktuuriga paar | 90.8 | Jah |
| **3** | Identsete muutujanimede ja erineva struktuuriga paar | 46.5 | Ei |
| **4** | Osalt erinevate muutujanimede ja osalt identsete ridadega paar | 74.9 | Jah |
| **5** | Sarnaste muutujanimede ja identse struktuuriga paar | 95.4 | Jah |
| **6** | Erinevate muutujanimede ja struktuuriga paar | 45.0 | Ei |

Tabelist 3 on näha, et programmipaare 1, 3 ja 6 ei pidanud mentorid plagiaadiks. Rakenduse võrdluses said kõik need kolm paari sarnasuseks alla 50% (vastavalt 49.6%, 46.5% ja 45.0%), mis viitab, et tegemist on rakenduse analüüsi põhjal üsna erinevate programmidega. Nende kolme paari puhul seega ühtib mentorite hinnang rakenduse analüüsi tulemustega. Tavapärase võrdluse käigus neid programmipaare rakenduse tulemustes ilmselt üldse ei kajastatakski, kuna sellise lühema ülesande puhul ei oleks mõistlik sarnasuse võrdlemisel kasutada nii madalat lävendit nagu 45%, sest see tooks palju valepositiivseid tulemusi.

Mentorite arvates oli programmipaaride 2, 4 ja 5 puhul tegemist plagiaadiga. Rakendus leidis paaridele 2 ja 5 sarnasuseks vastavalt 90.8% ja 95.4%. Nii kõrged sarnasuse protsendid näitavad, et tegemist on omavahel väga sarnaste programmidega, mis võivad ka tõenäoliselt olla plagieeritud.

o7 - jukebox.py

```
muutuja = input("Palun sisestage failinimi: ")
fail = open(muutuja, encoding = "UTF-8")
laulud = []
for rida in fail:
    laul = rida
    if laul[-1:] == "\n":
        laul = laul[:-1]
    laulud.append(laul)

print("Muusikapalade valik:")
for i in range(len(laulud)):
    print(str(i + 1) + ".", laulud[i])

muutuja2 = int(input("Valige laulu järjekorranumber: ")) - 1
print("Mängitav muusikapala on " + str(laulud[muutuja2]))

fail.close()
```

o22 - jukebox.py

```
failinimi = input("Palun sisestage failinimi: ")

fail = open(failinimi, encoding="UTF-8")
laulud = []

for rida in fail:
    laul = rida
    if laul[-1:] == "\n":
        laul = laul[:-1]
    laulud.append(laul)
fail.close()

print("Muusikapalade valik:")
for i in range (0, len(laulud)):
    print(str(i + 1) + ".", laulud[i])
number = input("\nValige laulu järjekorranumber: ")

pala = laulud[int(number) - 1]
print("Mängitav muusikapala on", pala + ".")
```

Joonis 14. Paar nr 4 - osalt erinevate muutujanimede ja osalt identsete ridadega paar

Paari 4 puhul andis analüüs sarnasuseks 74.9%, mis ei ole nii kõrge kui paaride 2 ja 5 puhul. Ilmselt on põhjuseks see, et 4. paari programmide struktuur on osaliselt erinev ja rakenduse sarnasuse võrdlemise algoritm on struktuursete muudatuste suhtes üsna tundlik. Joonisel 14 on toodud paari nr 4 programmide lähtekoodid. Jooniselt on näha, et parempoolse lahenduse koodis on võrreldes vasakpoolsega liigutatud ettepoole faili sulgemise käsk ning samuti on juurde lisatud muutuja *pala*, mis vasakpoolses lahenduses puudub. Kuna ka osad muutujanimed on programmides erinevad, siis lõppkokkuvõttes ei anna sarnasuse võrdlus väga kõrget sarnasuse protsenti. Samas näitab 74.9 protsendiline sarnasus siiski, et tegemist on pigem sarnaste kui erinevate programmidega, kuid selle põhjal ei saaks kindlalt väita, et tegemist võib olla plagiaadiga.

### 3.2.3 Kokkuvõte

Rakenduse testimisel 6 erineva lahendusprogrammiga loeti kõik programmipaarid, mida mentorid ei pidanud plagieerituks, ka rakenduse poolt erinevateks. Kolmest paarist, mida mentorid pidasid plagieerituks, hindas rakendus väga sarnaseks kaks. Kolmanda paari puhul oli rakenduse leitud sarnasus oluliselt väiksem, kuid selle põhjal sai siiski väita, et programmid on sarnased. Lõppkokkuvõttes kattusid viie paari puhul kuuest rakenduse analüüsi tulemused üsna täpselt mentorite hinnangutega ning viimase paari puhul langes rakenduse analüüsi tulemus ka pigem kokku mentorite hinnanguga, kuid natuke vähemal määral. Testimise tulemused näitavad, et rakenduse sarnasuse analüüs andis soovitud tulemusi.

Ühe ülesande lahenduste peal testimine annab aga rakenduse toimimise kohta vaid põgusa ülevaate. Programmeerimiskursustel on läbi nädalate ülesanded erilaadsed ning ühe ülesande raames ei pruugi ka kõik erinevad plagieerimisvõtted olla esindatud. Selleks, et valminud rakenduse toimimist paremini hinnata, tuleks seda veel põhjalikumalt testida. Kõige põhjalikuma hinnangu rakenduse kohta saaks ilmselt alles siis, kui õppejõud seda enda igapäevatöö käigus reaalselt kasutama hakkaksid.

## 3.3 Sarnasuse võrdlemise algoritmi analüüs

Rakenduses teostati lahendusprogrammide sarnasuse võrdlemine algoritmiga, mis põhineb sõne kujul olevate lähtekoodide vahelise Levenshteini kauguse leidmisel. Alapeatükis 2.4.2 kirjeldati selle algoritmi teostust detailselt. Käesolevas peatükis analüüsitakse kasutatud Levenshteini kauguse leidmise algoritmi häid ja halbu omadusi lähtekoodide võrdlemisel ning eraldi tuuakse välja ka lähtekoodide sõne kujul võrdlemise eelised ja puudused.

### 3.3.1 Levenshteini kauguse kasutamine lähtekoodide sarnasuse võrdlemiseks

Rakenduses kasutatava sarnasuse võrdlemise algoritmi puhul on hea see, et programmid, mis on struktuurilt väga sarnased või identsed ja mis erinevad põhiliselt vaid leksiliselt (näiteks muutujanimede poolest), saavad omavahelisel võrdlusel alati suure sarnasuse. Kuna Levenshteini kauguse leidmise implementatsioonis on mõnede levinumate leksiliste muutmiste hindasid vähendatud, siis mõjutavad need muudatused programmide vahelist muutmiskaugust vähe. Seetõttu on ka lõpptulemuseks saadud programmide vaheline sarnasus üldiselt üsna suur.

Samas mõjutavad struktuursed erinevused programmide vahelist sarnasust üsna palju. Kuna kahe programmi sarnasuse leidmiseks võrreldakse mõlemat lähtekoodi tervikuna, mitte ei otsita näiteks lähtekoodidest sarnaseid koodilõike, siis on struktuurilt erinevate programmide vaheline Levenshteini kaugus alati suur ning seetõttu nende sarnasus väike. Struktuurilt erinevate programmide puhul on aga sageli üldse keeruline otsustada, kas nende vaheline suur leksiline sarnasus viitaks ka plagieerimisele, sest erineva struktuuriga programmide puhul ei erine mitte ainult nende programmide välimus, vaid ka nende töökäik. Näiteks koodis ridade ümbertõstmine tähendab, et nendel ridadel olevad instruktsioonid tehakse programmi käivitamisel teises järjekorras kui algselt.

Struktuurilt erinevate programmide plagiaadiks pidamise keerulisust ilmestab ka see, et eelmises peatükis analüüsitud programmipaaride puhul otsustasid kursuse mentorid kõikide erineva struktuuriga paaride korral, et tegemist ei ole plagieeritud programmidega (vt tabel 3). Seevastu programmipaarid, mis mentorite poolt plagiaadiks loeti, olid kas identse või sarnase struktuuriga. Ei saa küll järeldada, et erinev struktuur programmide puhul tähendaks seda, et programmid on kindlasti iseseisvalt tehtud ja mitte plagieeritud. Üldjuhul see aga siiski pigem vähendab tõenäosust, et tegemist oleks plagiaadiga. Nagu juba peatükis 1.2 väideti, kasutavad õpilased plagieerimisel struktuurseid muudatusi ka oluliselt vähem kui

leksilisi, sest struktuuri muutmine on keerulisem tegevus. Nendel põhjustel ongi struktuursetel muudatustel põhineva plagieerimise tuvastamine jäetud rakenduse analüüsis tähelepanuta ning keskendutud on vaid leksilistel muudatustel põhineva plagieerimise tuvastamisele.

### 3.3.2 Lähtekoodide võrdlemine sõne kujul

Rakenduse sarnasuse tuvastamise algoritm käsitleb programmi lähtekoodi tervikliku sõnena. Esmapilgul võib see tunduda vale lähenemisena programmide võrdluseks. Samas on programmeerimise algkursustel lahendatavad ülesanded enamjaolt lihtsa loomuga ja sageli nõuavad õppijate poolt vaid ühe funktsiooni kirjutamist, eriti on see nii just Pythoni kursuste puhul. Sel juhul ei ole vale arvestada tervet lähtekoodi ühe tervikuna, kuna ühte funktsiooni saab pidada terviklikuks objektiks.

Programmeerimise algkursustel on kasutatavaks programmeerimiskeeleks tihti Python. Pythoni koodis puuduvad muutujate ning funktsioonide tüüpide deklareerimised, erinevalt paljudest teistest programmeerimiskeeltest – näiteks Java koodis on deklareerimine vajalik. See tähendab, et kahe Pythoni lähtekoodi puhul peaks nendevaheline süntaktiline kokkulangevus olema väiksem kui sel juhul, kui need programmid oleks Javas kirjutatud. Seega, kui kahte Pythoni programmi sõne kujul võrrelda ning nendevaheline sarnasus tuleb üsna suur, on ka suurem tõenäosus, et need ei pruugi olla iseseisvalt tehtud. Sõne kujul lähtekoodide võrdlemine peaks järelikult paremini toimima selliste programmeerimiskeelte koodide puhul, kus ei kasutata tüüpide deklareerimist, ning on natuke vähem efektiivne nende keelte puhul, kus tüübideklareerimine on vajalik.

Lähtekoodide sõne kujul võrdlemise eeliseks on ka see, et on võimalik üsna lihtsasti teha universaalne sarnasuse kontrollija, mida on võimalik paljude erinevate programmeerimiskeelte puhul kasutada. Iga programmeerimiskeele jaoks on vaja vaid eraldi implementeerida vastava keele lähtekoodi eeltöötlemine. Võimalik on ka lähtekoodide eeltöötlusfaas üldse välja jätta, mis juhul on universaalse lahenduse tegemine veelgi lihtsam, kuid mingisugune koodide eeltöötlemine on üldiselt siiski vajalik, et saada sarnasuse analüüsis paremaid tulemusi.

Sõnepõhise võrdluse kõige märgatavamaks puuduseks on võrdlusalgoritmi suur ajakulu. Olemasolevates plagiaadituvastusprogrammides teisendatakse sageli eeltöötluse käigus lähtekoodid lekseemideks, sageli just sel põhjusel, et vähendada võrdluse aega. Peatükis 3.1

kirjeldatud 300 õppija lahenduste analüüsimine võttis bakalaureusetöö käigus loodud tarkvaraga, mis teostab sõnepõhist võrdlust, aega 138 sekundit. Samade lahenduste võrdlemine näiteks VPL-i plagiaadituvastussüsteemiga [19], kus programmid teisendatakse eelnevalt lekseemideks, võtab aega vaid mõne sekundi, mis kinnitab seda, et sõnepõhine võrdlus on oluliselt aeglasem kui lekseemide põhine võrdlemine. Loodud rakenduse sõnepõhise võrdluse ajakulu tuleneb suuresti sellest, et kasutusel oleva Levenshteini kauguse leidmise algoritm on ruutkeerukusega. Ajakulu vähendamiseks oleks võimalik võtta kasutusele mõni teine sõnevõrdlusalgoritm, mis teostab võrdlust efektiivsemalt.

Rakenduse sarnasuse võrdlemise algoritmi analüüsist järeldub, et rakenduse ettenähtud viisil kasutamisel – programmeerimise algkursustel õppijate lahenduste võrdlemisel – toimib algoritm võrdlemisi hästi. Siiski leidub kasutatud algoritmi puhul ka aspekte, mida annaks parandada.

# 4 Võimalikud edasiarendused

Rakenduse põhifunktsionaalsus sai nõuetekohaselt valmis, kuid funktsionaalsuse täiendamiseks tekkis mõningaid ideid, mille elluviimiseks antud bakalaureusetöö käigus aega ei jätkunud. Järgnevalt tuuakse välja mõned rakenduse edasiarendamise võimalused ning peatüki lõpus antakse ka mõned ideed selle kohta, mida võiks antud teema puhul tulevikus lähemalt uurida.

Kõikides Tartu Ülikoolis õpetatavates programmeerimise algkursustes ei kasutata programmeerimiskeelena ainult Pythonit. Mõnes kursuses on kasutusel ka Java, seega olekski üheks võimalikuks rakenduse edasiarenduseks lisada juurde ka Java lähtekoodide kontrollimise tugi. Siis oleks rakendust võimalik kasutada rohkemates programmeerimiskursustes lahenduste sarnasuse kontrollimiseks.

Bakalaureusetöö käigus loodud tarkvara kontrollib vaid lahendusprogrammide omavahelist sarnasust. Kuigi levinum ongi just õppijate omavaheline plagieerimine, siis võib vähemal määral esineda ka veebist leitavate materjalide põhjal plagieerimist. Võimalik oleks rakenduse analüüsimise protsessi kaasata ka veebimaterjalide läbivaatamine, et seeläbi tuvastada esitatud programmidega sarnaseid materjale internetist. Sel juhul kataks rakendus laiemat spektrit võimalikest plagieerimise viisidest.

Rakenduse praeguses versioonis on võimalik analüüsi tulemusel leitud sarnaseid programme kõrvuti vaadata ning lähtekoodide süntaks on seejuures värviliselt esile toodud. Selleks, et programmide omavaheline võrdlemine oleks aga veelgi mugavam, võiks sarnaste paaride kõrvuti vaatamisel esile tuua ka nendevahelised erinevused. Seda saaks teostada selliselt nagu paljudes versioonihaldustarkvarades, kus kahe eri versiooni vahelised erinevused on tavaliselt värviliselt esile toodud. See teeks mugavamaks tuvastatud sarnaste paaride käsitsi üle kontrollimise protsessi.

Viimaks saaks tulevastes töödes lähemalt uurida, kuidas lähtekoodide sõne- ja lekseemide põhjal võrdlemine üksteisest erinevad ning mis on nende omavahelised eelised ja puudused. Paljudes olemasolevates plagiaadituvastusprogrammides tehakse just lekseemide põhjal võrdlemist, kuid bakalaureusetöö käigus loodud tarkvara võrdleb lähtekoode sõnepõhiselt. Eelkõige saaks uurida seda, kas lekseemideks teisendamine on põhjendatud vaid kiirema analüüsi saavutamisega või annab see mingil määral ka täpsemaid tulemusi kui sõnepõhine võrdlus.

# 5 Kokkuvõte

Plagieerimine on akadeemilises maailmas esinev murettekitav probleem, mille tõkestamine on vajalik, kuid sageli ei ole seda lihtne teha. Üheks enamlevinud plagieerimise tõkestamise viisiks on plagieeritud tööde tuvastamine ning nende tööde autorite karistamine. Programmeerimisülesannete puhul on plagieerimise piirid aga natuke hägusamad, kui muudes valdkondades, seega on nende puhul plagiaadi eristamine mõnevõrra keerulisem. Samuti õpib programmeerimiskursustel sageli sadu tudengeid, seega on üsna mahukas töö kontrollida kõigi esitatud lahenduste puhul, et need oleks ka iseseisvalt tehtud. Bakalaureusetöö eesmärgiks oli luua tarkvara, mis kontrolliks õppijate poolt esitatud lahendusprogrammide omavahelist sarnasust ning abistaks sellega õppejõude plagieerimise tuvastamisel programmeerimise algkursustel.

Enne sarnasuse kontrollimise tarkvara loomist tutvuti esmalt olemasoleva plagiaadituvastustarkvaraga. Analüüsiti nendes rakendustes kasutatavaid meetodeid programmide sarnasuse võrdlemiseks ning probleeme, mis esinevad olemasoleva tarkvara puhul õpilaste lahenduste kontrollimisel. Selle põhjal seati paika nõuded loodava rakenduse jaoks.

Töö käigus valmis töölauarakendus, mis kontrollib sisendiks antavate Pythoni programmide omavahelist sarnasust ning tuvastab seeläbi sisendprogrammide hulgast kõige sarnasemad programmipaarid ja sarnaste programmide klastrid. Töös kirjeldati rakenduse loomise protsessi ning selleks kasutatud tehnoloogilisi vahendeid. Samuti seletati rakenduse kasutajaliidese ning programmide sarnasuse analüüsimise protsessi toimimist.

Valminud rakenduse kvaliteedi hindamiseks testiti seda programmeerimise algkursuse „Tehnoloogia tarbijast loojani“ ühe ülesande kuue lahenduspaariga, mille sarnasust olid eelnevalt hinnanud selle kursuse õppejõud. Rakendusel lasti analüüsida neid kuut lahenduspaari. Seejärel võrreldi rakenduse poolt saadud tulemusi õppejõudude hinnangutega. Viie paari puhul kuuest kattusid rakenduse tulemused üsna täpselt õppejõudude hinnangutega. See andis rakenduse kvaliteedi kohta positiivset tagasisidet. Selleks, et aga täielikumat ülevaadet rakenduse toimimise kvaliteedi kohta saada, oleks seda tarvis tulevikus põhjalikumalt testida. Kõige parema tagasiside rakenduse kohta saaks aga alles siis, kui programmeerimise õppejõud selle praktilist kasutamist oma igapäevatöös proovivad.

## Viidatud kirjandus


[1] Curtis G. J., Tremayne K. Is plagiarism really on the rise? Results from four 5-yearly surveys. *Studies in Higher Education*, 2019, 1-11. https://doi.org/10.1080/03075079.2019.1707792

[2] Cook B., Sheard J., Carbone A., Johnson C. Academic integrity perceptions regarding computing assessments and essays. *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 2014, 107-114. https://doi.org/10.1145/2632320.2632342

[3] Sheard J., Dick M., Markham S., Macdonald I., Walsh M. Cheating and plagiarism: perceptions and practices of first year IT student. *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education*, 2002, 183-187. https://doi.org/10.1145/544414.544468

[4] Gregory J. L. COVID-19 Elevating the Problem of Plagiarism: The Implied Social Contract of Academic Integrity. *Delta Kappa Gamma Bulletin*, 2020, vol 87(1), 18-23.

[5] Tartu Ülikooli Õppeinfosüsteem (ÕIS II) Programmeerimine (6 EAP) LTAT.03.001. https://ois2.ut.ee/#/courses/LTAT.03.001/details (13.04.2021)

[6] Mozgovoy M. Desktop Tools for Offline Plagiarism Detection in Computer Programs. *Informatics in Education*, 2006, vol 5(1), 97-112.

[7] Joy M., Luck M. Plagiarism in programming assignments. *IEEE Transactions on Education*, 1999, vol 42(2), 129-133. https://doi.org/10.1109/13.762946

[8] Karnalim O., Simon, Chivers W. Similarity Detection Techniques for Academic Source Code Plagiarism and Collusion: A Review. *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*, 2019, 1-8. https://doi.org/10.1109/tale48000.2019.9225953

[9] Turnitin. Turnitin Similarity | Robust Plagiarism Checker. https://www.turnitin.com/products/similarity (13.04.2021)

[10] Grammarly. Plagiarism Checker by Grammarly. https://www.grammarly.com/plagiarism-checker (13.04.2021)

[11] JPlag. http://jplag.ipd.kit.edu/ (13.04.2021)

[12] Aiken A. Moss - A System for Detecting Software Similarity. https://theory.stanford.edu/~aiken/moss/ (13.04.2021)

[13] University of Warwick, Department of Computer Science Sherlock - Plagiarism Detection Software. https://warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock (13.04.2021)

[14] Wise M. YAP3: Improved Detection Of Similarities In Computer Program And Other Texts. *ACM SIGCSE Bulletin*, 1996, vol 28. https://doi.org/10.1145/236462.236525

[15] Marttila H.-K. Programmide sarnasuse määratlemine ühe programmeerimisülesande näitel. Tartu, 2020. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=69754&year=2020 (01.05.2021)

[16] Wise M. String Similarity via Greedy String Tiling and Running Karp-Rabin Matching. *Unpublished Basser Department of Computer Science Report*, 1993.

[17] Paterson M., Dančik V. Longest common subsequences. *Proceedings of 19th International Symposium on Mathematical Foundations of Computer Science, number 841 in Lecture Notes in Computer Science*. Switzerland: Springer, 1994, 127-142. https://doi.org/10.1007/3-540-58338-6_63

[18] Grashchenko S. Levenshtein Distance Computation. Baeldung. https://www.baeldung.com/cs/levenshtein-distance-computation (29.04.2021)

[19] Rodríguez-del-Pino J. C., Rubio-Royo E., Hernández-Figuero Z. J. A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. Departament of Informática y Sistemas University of Las Palmas de Gran Canaria, 2012.

[20] Python documentation. https://www.python.org/doc (15.04.2021)

[21] CodeGrade How to check for plagiarism in Python source code. https://www.codegrade.com/blog/how-to-check-for-plagiarism-in-python-source-code (15.04.2021)

[22] Oracle Corporation JDK 11 Documentation. https://docs.oracle.com/en/java/javase/11/ (20.04.2021)

[23] PYPL PopularitY of Programming Language. https://pypl.github.io/PYPL.html (21.04.2021)

[24] Oracle Corporation JavaFX. https://openjfx.io/ (14.04.2021)

[25] Scene Builder. https://gluonhq.com/products/scene-builder/ (14.04.2021)

[26] Oracle Corporation Introduction to FXML. https://docs.oracle.com/javase/8/javafx/api/javafx/fxml/doc-files/introduction_to_fxml.html (20.04.2021)

[27] Oracle Corporation JavaFX CSS Reference Guide. https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html (21.04.2021)

[28] Tutorialspoint What is CSS? https://www.tutorialspoint.com/css/what_is_css.htm (26.04.2021)

[29] Jython. https://www.jython.org/ (14.04.2021)

[30] Gradle Build Tool. https://gradle.org/ (14.04.2021)

[31] Brandl G., Chajdas M. Pygments. https://pygments.org/ (22.04.2021)

[32] Ledwith J. Exponential Function and Decay. https://www.thoughtco.com/exponential-decay-definition-2312215 (26.04.2021)

[33] Flexlense. SysGauge - System Monitor. https://www.sysgauge.com/ (03.05.2021)

# Lisad

## I. Rakenduse lähtekoodi repositoorium

Rakenduse lähtekood on üleval GitHubi repositooriumis veebiaadressil https://github.com/mikkomaran/Source-code-similarity-detector. Samuti on sealt võimalik alla laadida rakenduse viimase versiooni JAR-fail, mida saab käivitada, kui on installeeritud JDK versioon 11 või uuem.

## II. Litsents

**Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks**

Mina, Mikko Maran,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   **Õpilaste lahendusprogrammide sarnasuse kontrollija**,

   mille juhendaja on Reimo Palm,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

*Mikko Maran*

***07.05.2021***