TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Toom Lõhmus

Programm funktsiooni ajalise keerukuse leidmiseks

Bakalaureusetöö (9 EAP)

Juhendaja: Reimo Palm

Tartu 2018

Programm funktsiooni ajalise keerukuse leidmiseks

Lühikokkuvõte:

Käesoleva bakalaureusetöö raames valmis Java-programm funktsiooni ajalise keerukuse leidmiseks. Programm loodi eesmärgiga lihtsustada programmeerimisülesannete hindamist olukorras, kus hinne sõltub lahenduse ajalisest keerukusest. Käesolev töö annab ülevaate programmi kasutusvõimalustest ning dokumenteerib programmi seadistamise ja kasutamise Moodle'i keskkonnas esitatud lahenduste jaoks. Töös on välja toodud funktsiooni käivitamise ning ajalise keerukuse leidmise jaoks kasutatavad meetodid.

Võtmesõnad:

Tarkvara, ajaline keerukus, Java (programmeerimiskeel), funktsioonide lähendamine

CERCS: P175 (Informaatika, süsteemiteooria)

The Program for Determining the Time Complexity of a Function

Abstract:

The purpose of this Bachelor's thesis is to develop a Java program to determine time complexity of a function. The program was created with the idea of simplifying the evaluation of programming exercises, where the result is based on time complexity of the solution. This Bachelor's thesis provides an overview of functionalities of the program and documents the configuration and usage of the program for solutions submitted in online learning environment Moodle. The study describes the methods used for executing functions and finding time complexity.

Keywords:

Software, time complexity, Java (programming language), approximation of functions

CERCS: P175 (Informatics, systems theory)

Sisukord

1. Sissejuhatus

Tartu Ülikoolis kasutatakse paljudes arvutiteaduse instituudi ainetes õpiväljundite kontrollimiseks programmeerimisülesandeid. Kõige suurema kaaluga on programmeerimisülesanded programmeerimise õpetamisele keskenduvates ainetes, näiteks „Programmeerimine“, „Objektorienteeritud programmeerimine“ ning „Programmeerimiskeeled“. Aga programmeerimisülesandeid esineb ka teistes ainetes, näiteks „Algoritmid ja andmestruktuurid“, „Diskreetne matemaatika I“ ja „Operatsioonisüsteemid“.

Tavaliselt küsitakse tudengitelt etteantud nõuetele vastava funktsiooni (või meetodi) kirjutamist. Ülesande püstituses antakse ette funktsiooni signatuur – nimi, sisendparameetrid ning tagastusväärtuse tüüp. Lisaks on mõne ülesande jaoks kasutusel standardsisendi ning standardväljundi või failide kaudu andmevahetus.

Lahenduse kontrollimiseks kuluvat aega aitab lühendada automaatkontroll. Programmeerimise õpetamisele keskenduvatel ainetel on üldjuhul automaatkontrollid juba olemas, nende koostamise ja arendamisega on tegelenud Karl-Aksel Puulmann [1], Aivar Annamaa [2], Kaspar Papli [3] ja teised. Automaatkontrolli puudumise korral on vaja esitatud lahenduse hindamiseks selle kood hoolikalt läbi vaadata, mis on ajamahukas.

Kasutusel olev Moodle'i lisandmoodul VPL (*Virtual Programming Lab* [4]) võimaldab standardseadistusega lihtsamaid sisendi ja väljundi kontrolle, aga algoritmide hindamisel on oluline ka ajaline keerukus. Kuna lisandmoodul VPL võimaldab funktsiooni sisendi ja väljundi kontrolli, siis käesolev töö ei keskendu programmi korrektsuse testimisele. Töö käigus valminud programm on mõeldud automaatkontrolli teise sammuna: esmalt jooksutatakse ühiktestid (*unit test*), seejärel käivitatakse ajalise keerukuse leidmise programm.

Käesoleva töö käigus valmis keeles Java kirjutatud programm, mis käivitab etteantud programmifaili ning mõõdab selles leiduvate funktsioonide tööaega erinevate sisendite korral. Programmi väljundiks on käivitusajad iga vaadeldud sisendi suuruse kohta, samuti tagastatakse leitud lähend ajalisele keerukusele. See programm on disainitud käivitama Moodle'i VPL-harjutuse kaudu esitatud koodi. Programmi lähtekood on saadaval versioonihaldussüsteemi *git* kaudu [5]. Samuti on seal olemas seadistuse juhend projekti ülesseadmiseks oma kohalikus arvutis.

Ajaline keerukus on defineeritud kui funktsioon, mis seab andmemahule vastavusse algoritmi täitmiseks vajamineva sammude arvu [6]. Käesoleva töö käigus valminud programmiga leitakse funktsiooni tegeliku ajalise keerukuse lähend. Selleks mõõdetakse funktsiooni tööaega erinevate sisendite korral ning tööaegadest leitakse keerukusfunktsioon, mis on lähedane funktsiooni ajalisele keerukusele.

Tartu Ülikoolis alustatakse programmeerimise õpetamist keeltest Python ja Java. Käesolev töö lähtub aine „Algoritmid ja andmestruktuurid“ ootustest ajalise keerukuse leidmise programmile, kuna selles aines pole automaatkontrolle eriti palju kasutatud ning selles aines on oluline fookus algoritmi ajalisel keerukusel. Selles aines on peamine programmeerimiskeel Java, aga algoritmide näited on sageli keeles Python, mis võimaldab esitada kompaktsemat koodi. Seetõttu peab ajalise keerukuse leidmise programm toetama nii keeles Java kui ka keeles Python kirjutatud funktsioone. Käesolevas töös on mõiste Java-funktsioon kasutusel staatilise meetodi tähenduses.

Aine „Algoritmid ja andmestruktuurid“ kodutöödele on varasemalt automaatkontroll koostatud ainult mõne rühma variandile vastava praktikumijuhendaja poolt. Erinevad ülesanded erinevatel rühmadel toovad kaasa suurema praktikumijuhendajate koormuse, aga ka raskendavad üldise automaatkontrolli kasutamist. Siiski ei ole kursuse korralduse muutmine

käesoleva töö seisukohast vajalik. Küll aga tähendab erinev küsitud funktsiooni signatuur vajadust lisada Moodle'i keskkonda eraldi VPL-harjutus iga signatuuri kohta.

Käesoleva kirjaliku töö põhiosa on jaotatud nelja peatükki. Peatükk 3 annab ülevaate programmeerimisülesannete ja automaatkontrolli kasutamisest Tartu Ülikoolis. Peatükk 4 kirjeldab funktsioonide käivitamise jaoks kasutatavaid meetodeid. Peatükis 5 selgitatakse keerukusklassi valimisel rakendatavaid võtteid. Peatükk 6 sisaldab programmi seadistamise võimalusi ning ülesseadmise ja kasutamise juhendit.

2. Töö eesmärk

Töö eesmärgiks on kirjutada programm, mis lisab programmeerimisülesannete automaatkontrollile ajalise keerukuse mõõtmise. Seni on kasutatud sisend-väljund-testimist, aga fookus pole olnud programmi töökiirusel. Koostatud programmiga saab kohe teada, kui ülesande lahenduseks esitatud funktsiooni ajaline keerukus on oluliselt erinev oodatavast ajalisest keerukusest.

Selle töö käigus koostatud programm mõõdab ajalist keerukust funktsiooni käivitamise teel, lähtekoodi analüüsimist ega elementaaroperatsioonide lugemist ei toimu. Katseliselt määratud ajaline keerukus on suure tõenäosusega lähedane funktsiooni tegelikule ajalisele keerukusele. Kuna nii tudengil kui ka õppejõul on võimalik automaatset hindamist uuesti käivitada, ei ole mõõtmistulemuste võimalik varieerumine probleem.

Suuremate võimalustega automaatkontrollist on kasu nii õppijal kui ka õppejõul. Õppijal tekib võimalus vähese vaevaga proovida, millised muudatused koodis mõjutavad tööaega. Samuti on vahetult pärast esitamist saadaval esialgne hinnang lahendusele. Kui esialgne hinnang leiab, et tööaeg on oodatavast suurem, siis on õppijal lihtsam muudatusi teha: vahetult pärast esitamist on veel meeles, kuidas lahendus peaks toimima.

Õppejõu vaatepunktist tähendab võimsam automaatne hindamine väiksemat töövaeva. Kui käivitusaja mõõtmine toimub juba esitamisel, siis ei ole õppejõul vaja seda oma arvutis uuesti läbi teha. Samuti aitab objektiivne aja mõõtmine vähendada rühmadevahelisi hindamise erinevusi. Mõõtmistulemuste põhjal on võimalik koostada edetabel ning disainida ülesandeid, kus on eesmärgiks saavutada programmi väiksem tööaeg.

3. Praegune olukord

3.1 Programmeerimise õpetamisele keskenduvad ained

Tartu Ülikooli informaatika õppekavas on peamised programmeerimise õpetamisele pühenduvad ained „Programmeerimine“, „Objektorienteeritud programmeerimine“, „Programmeerimisharjutused“ ning „Programmeerimiskeeled“. Selliseid aineid on rohkem ning need võivad olla mõeldud ka mitteinformaatikutele. Nendes ainetes saadakse oluline osa tudengi lõpphindest programmeerimisülesannete lahenduste eest.

Nimetatud ainetes on suuremal või vähemal määral kasutuses automaatkontroll, mis võrdleb programmi väljundit oodatava väljundiga. Ained „Programmeerimisharjutused“, „Programmeerimise alused“ ning „Programmeerimise alused II“ on disainitud tuginema täielikult automaatsele hindamisele. Teistes nimetatud ainetes on automaatkontrolli hinne ainult esialgne – lahendus vaadatakse ka käsitsi üle. Sellistes ainetes leidub ka ülesandeid, milles automaatkontrolli ei kasutata, näiteks graafilise kasutajaliidese koostamine. Automaatkontrollil on kaasas mälu- ja ajapiirang, mida ei või ületada, aga tegelikku tööaega ega mälukasutust ei näidata.

Programmeerimise algkursuste automaatkontrollides esineb ka lahenduse staatilist analüüsi. Selle jaoks on koostatud teek, mis võimaldab näiteks kontrollida, kas lahenduses on kasutatud tsüklit, rekursiooni või on defineeritud mitu erinevat funktsiooni [7]. Selline analüüs on kasutusel olukorras, kus oodatav programmi struktuur on rangelt ette antud ning peab kasutama ainult lubatud lahendusvõtteid.

3.2 Programmeerimisülesandeid kasutavad ained

Tartu Ülikoolis leidub ka aineid, kus ainult osa lõpphindest tuleb programmeerimisülesannete lahenduste eest. Sellistes ainetes võib programmeerimine olla vaid lisaülesanne boonuspunktide jaoks, aga võib ka olla vajalik osa õpiväljundite kontrolliks. Näiteks ainetes „Matemaatiline maailmapilt“ ning „Diskreetne matemaatika I“ annavad programmeerimisülesanded võimaluse vaadeldud teemat ise järele proovida, aga ei mõjuta punktisummat olulisel määral. Nendele ülesannetele tavaliselt automaatkontrolli ei ole koostatud, sest lahendajaid pole eriti palju.

Käesoleva töö fookus on aine „Algoritmid ja andmestruktuurid“. Selles aines on programmeerimisülesannete osakaal lõpptulemusest 40%. Kasutusel on kaheksa suurt kodutööd, kus tuleb realiseerida etteantud algoritm. See on esimene aine, kus lisaks korrektsusele on oluline ka lahenduse ajaline keerukus. Nendele ülesannetele üldiselt automaatkontrolli ei ole, aga praktikumijuhendaja võib testid koostada oma rühma ülesannete variandile. Selles aines vähendaks ühtsed automaatkontrolliga ülesanded oluliselt juhendajate koormust. Aines „Algoritmid ja andmestruktuurid“ tuleb kõik programmid koostada keeles Java, millele on mugav lisada ühikteste, kasutades teeki JUnit [8].

3.3 Programmeerimisülesande automaatkontrolli ülesehitus

Selles peatükis nimetatud kursustes toimub automaatkontrolliga ülesannete lahenduste esitamine Moodle’i keskkonnas VPL-harjutuse kaudu. Lahenduse kontrollimiseks saadetakse esitus käivitusserverisse koos kõigi vajaminevate failidega. VPL-harjutuse seadistuses määratakse failiga *vpl_evaluate.sh*, kuidas lahendust kontrollida. Selle skripti töö tulemusena genereeritakse fail *vpl_execution*, mis seejärel jooksutatakse. Fail *vpl_execution* väljastab standardväljundisse kindlas formaadis kommentaare, mida Moodle’i keskkonnas näidatakse. Samuti on võimalik väljastada hinde kommentaar, mis seatakse lahenduse esialgseks hindeks Moodle’i keskkonnas.

Kompileeritavate keelte korral on faili *vpl_evaluate.sh* ülesandeks ka esituse kompileerimine. Pärast kompileerimist jooksutatakse fail *vpl_execution*, mis saab kompileeritud faile kasutada ning nende alusel tulemuse väljastada.

4. Kasutatavad meetodid

4.1 Üldine lähenemine

Funktsiooni ajalise keerukuse leidmiseks on mitu lähenemist. Esiteks on võimalik analüüsida funktsiooni lähtekoodi ning leida, mitu korda iga lause (*statement*) käivitatakse. Seda saab teha nii ainult halvima juhu kui ka iga käsuvoo jaoks. [9] Selle eeliseks on võimalus leida täpne operatsioonide arvu sõltuvus iga sisendiklassi kohta – see tähendab, et näiteks tingimuslausete jaoks vaadeldakse iga võimalikku käsuvoogu. Samuti saab ligikaudse tulemuse leida juba enne funktsiooni valmimist, mis võimaldab varakult valida, millist algoritmi kasutada. Peamiseks puuduseks selle lähenemisega on asjaolu, et seda on väga keeruline automaatselt teha – vaadeldav funktsioon võib teisi funktsioone välja kutsuda, mille mõju ajalisele keerukusele pole teada.

Teiseks saab mõõta, kui kaua funktsiooni töö kestab etteantud argumendi korral. Sisendargumendi suuruse ning kulunud aja alusel on leitav tegelikule ajalisele keerukusele lähedane keerukusfunktsioon. Selle eeliseks on automatiseerimise lihtsus, funktsiooni saab käivitada genereeritud andmetega. Selle lähenemise puuduseks on asjaolu, et funktsiooni tööaeg varieerub erinevate käivituste korral. Kulunud aega mõjutab arvuti koormus mõõtmise käigus, võrguliiklus, kettapöörduste arv, prügikoristus (*garbage collection*) ning veel hulk tegureid [10].

Kolmandaks võimaluseks on lisada vaadeldava programmi lähtekoodi väljakutsed, mille järgi saab kokku lugeda, mitu ühikoperatsiooni läbi viidi. See lähenemine on levinud programmi profiilide kogumise (*profiling*) osana. [11] Nende väljakutsete kaudu saab teada, kui kaua mingi koodiosa aega võttis, aga käesoleva töö kontekstis vajab see lähenemine liiga palju vaeva võrreldes teise lähenemisega.

Käesoleva töö käigus koostatud programm leiab ajalise keerukuse programmi tööaja mõõtmise kaudu. Funktsioone käivitatakse erineva suurusega sisendite korral ning leitakse keskmine kuluv aeg iga kasutatud sisendi suuruse kohta. Selle lähenemisega saab teada funktsiooni ajalise keerukuse keskmisel juhul (*average-case complexity*). Käivitamiste lõpus on olemas andmepunktid, mis koosnevad sisendi suurusest (täisarvuna) ja kulunud ajast (täisarvuna millisekundites). Nende alusel leitakse vähimruutude meetodil lähedane funktsioon.

4.2 Java koodi käivitamine

Java on kompileeritav keel – programmi käivitamiseks on vaja failid laiendiga .java kompileerida baitkoodiks, failideks laiendiga .class [12]. Kuna käesoleva töö käigus koostatud programm on samuti keeles Java, siis on testimisele kuuluva funktsiooni käivitamiseks vajalikud vahendid keele standardvahendites olemas.

Koostatud programm kompileerib esitatud faili ning loeb selles sisalduva klassi mällu. Dünaamiliseks ligipääsuks kompileeritud klassile on kasutusel Java-keele komponent *reflection*, mis võimaldab programmi tööajal klasside sisu analüüsida ja muuta [13]. Seejärel otsitakse klassist meetod, mis vastab programmi seadetes määratud meetodi nimele. Kui sama nimega meetodeid on mitu, kasutatakse esimest. Aega mõõdetakse meetodi käivitamisest meetodi väljakutse lõpetamiseni.

4.3 Pythoni koodi käivitamine

Python on interpreteeritav keel, koodi käivitamiseks on vaja Pythoni interpretaatorit, aga koodi kompileerida pole vaja. Koostatud programm saab käivitada .py laiendiga faile uue

protsessina. Kui testimisele kuulub terve Pythoni fail, siis saab sellesama faili käivitada. Kui aga on vaja funktsiooni kontrollida, siis on vaja esitatud fail mõnest teisest .py laiendiga failist importida.

Selle jaoks genereeritakse sobilik Pythoni fail, mis impordib esitatud faili, käivitab selles defineeritud funktsiooni ning mõõdab selle funktsiooni tööaega. Koostatud fail käivitatakse uue protsessina. Samuti mõõdab protsessi tööaega töö käigus koostatud Java-programm.

Mõõdetud tööaegade erinevus näitab, kui suur on ajakulu, mis ei sõltu kontrollitavast funktsioonist. See erinevus tuleneb protsessi loomisele kuluvast ajast ning Pythoni faili importimisele kuluvast ajast. Kui esitatud Pythoni fail lisaks funktsiooni defineerimisele seda ka ise käivitab ilma kontrollita, kas seda faili käivitatakse kui skripti [14], siis kulub aega ka sellele. Samuti toimub Pythoni andmestruktuuride koostamine sisendi suuruse alusel Pythoni protsessis, mis töö käigus valminud programmi vaatepunktist kuulub ka käivitusaja sisse.

Funktsiooni sisendparameetri väärtuse suurus edastatakse protsessi käivitamisel käsureaparameetrina. Sisendi teisendamiseks otsitakse failist *datagen.py* funktsioon `getInput`, mis teisendab parameetri suuruse reaalseks väärtuseks, näiteks suuruse 100 teisendab listiks, mis sisaldab 100 elementi. Vaikimisi, faili *datagen.py* puudumise korral, kasutatakse suuruse arvulist väärtust ka reaalse sisendväärtusena. Vaadeldava funktsiooni käivitamiseks genereeritakse ning käivitatakse fail *python_runner.py* kujul

```
import FAILINIMI as source
import time
import sys
try:
    import datagen
    input_value = datagen.getInput(int(sys.argv[1]))
except ImportError:
    input_value = int(sys.argv[1])
start_time = (time.time())
source.FUNKTSIOONINIMI(input_value)
end_time = (time.time())
print(1000*(end_time-start_time))
```

Igal käivitamisel kirjutatakse faili seadetes määratud failinimi ning funktsiooninimi koodinäites esinevate `FAILINIMI` ning `FUNKTSIOONINIMI` asemele.

4.4 Funktsiooni sisendväärtuste koostamine

Ajalise keerukuse leidmiseks funktsiooni käivitamise kaudu on vaja valida sobivad sisendväärtused, millega seda funktsiooni käivitada. Kui sisendiks on täisarv, siis saab kasutada sedasama arvu. Teistsuguse sisendi andmetüübi korral on esmalt vaja teisendada parameetri suurus parameetri väärtuseks.

Funktsiooni tööaega mõõdetakse vastavusena sisendparameetri suurusest, mis on täisarv. Kui vaadeldav funktsioon tahab teist tüüpi sisendit, siis kasutatakse failides *DataGen.java* ning *datagen.py* defineeritud funktsiooni/meetodit, mis tagastab etteantud suurusega sobivat tüüpi väärtuse. Näide koos täiendava kirjeldusega on esitatud programmi kirjelduse peatükis, jaotises 6.4.

4.5 Etteantud testikomplekti kasutav programm

Töö käigus valminud programm võimaldab ka sisendi ja väljundi testimist standardsisendi ning standardväljundi kaudu, aga funktsiooni korrektsuse kontroll ei ole selle töö fookus. Sellise testitava programmi korral suunatakse etteantud fail standardsisendiks ning kontrollitakse standardväljundi vastavust etteantud vastusefaili sisuga. Testikomplektis on ka fail,

mis annab teada, mida lugeda sisendfaili suuruseks. Standardsisendit kasutav lahendus on mõeldud olukorraks, kus tahetakse käivitada tervet programmi – Pythoni korral tervet faili, ning Java korral *main*-meetodit.

Iga etteantud testi korral toimub tööaja mõõtmine nagu ka genereeritud sisendiga funktsioonidega. Samas on ajalise keerukuse katseliseks leidmisega vaja kümneid andmepunkte, et tulemusel oleks usaldusväärsust. Väikese andmemahu puhul on oht, et juhuslik tööaja varieerumine mõjutab oluliselt leitava keerukusfunktsiooni kuju. Seetõttu vähem kui 10 erineva sisendandmete suuruse väärtuse korral ajalise keerukuse leidmist ei alustatagi. Siiski saab endiselt teada tööaja iga andmekomplekti kohta.

5. Ajalise keerukuse määramine

5.1 Algoritmi ajaline keerukus

Algoritmi ajaline keerukus kirjeldab, kui kiiresti see algoritm töötab. Kuna töö kiirus sõltub mitmesugustest teguritest nagu protsessori kiirus ning salvestusseadme kiirus, vaadeldakse aja või kiiruse võrdlemise asemel sammude arvu. Algoritmi ajaline keerukus on hinnang algoritmi elementaarsammude arvule vastavalt sisendi suurusele. Elementaarsammuks nimetatakse tegevust, mille sooritamiseks kulub konstantne hulk aega. [15]

Ajalise keerukuse vaatlemisel eristatakse halvima, parima ning keskmise juhu ajalist keerukust. Parim juht kirjeldab tavaliselt olukorda, kus on vaja minimaalselt tööd teha, näiteks järjendist elemendi otsimine, kui otsitav element on esimesel positsioonil. [15] Kuna algoritmi kasutamisel tavaliselt ei rakendu parim juht, siis tuginetakse keerukuse analüüsimisel algoritmide keskmise juhu ajalisele keerukusele.

5.2 Regressioonianalüüs

Regressioonianalüüs on statistiline meetod, mis üritab leida sõltuva muutuja ning sõltumatute muutujate omavahelist suhet. Käesolevas töös kasutatakse regressioonianalüüsi tüüpidest lineaarregressiooni ning selle laiendust polünomiaalset regressiooni. Lineaarregressioon seosega `Y = a + bX` sisaldab parameetreid `a` ja `b`, mis kirjeldavad sõltuva muutuja `Y` väärtust sõltumatu muutuja `X` alusel. [16] Selle seose korral `a` on vabaliige ning `b` on sirge tõus. Polünomiaalne regressioon lisab lineaarregressioonile üksliikmeid juurde, näiteks ruutregressiooni korral on seos kujul `Y = a + bX + cX^2`, lisandunud on ruutliikme kordaja `c`.

Regressiooniprobleemi lahendamine tähendab regressiooniseose parameetrite väärtuste leidmist, eesmärgiga vaadeldavat sõltumatu ning sõltuva muutuja seost võimalikult täpselt kirjeldada. Parameetrite sobivuse hindamisel kasutatakse tavaliselt vähimruutude meetodit. Vähimruutude meetodi korral minimeeritakse andmepunktide ning ennustatud seose väärtuste ruutude summat. Erinevusi mõõdetakse mööda vertikaaltelge ning need tõstetakse ruutu, et vältida erimärgiliste hälvete korral üksteise kustutamist. [17]

5.3 Käivitusaegadest ajalise keerukuse leidmine

Funktsiooni käivitamise tulemusena saab töö käigus valminud Java-programm teada hulga andmepunkte, mis sisaldavad sisendi suuruse ning tööaja informatsiooni. Kõrvale jäetakse kõik punktid, mille korral tööaeg on alla ühe millisekundi kas sellel punktil või mistahes suurema sisendi suurusega punktil. Sellised punktid ei sisalda usaldusväärset informatsiooni, juhuslik tööaja varieerumine mõjutab neid liiga suurel määral.

Valminud programm vaatleb võimalike keerukusfunktsioonidena järgnevaid seoseid sisendparameetri suurusega `n`:

1) logaritmfunktsioon `log(n)`,
2) lineaarfunktsioon `n`,
3) linearitmiline seos `n*log(n)`,
4) ruutfunktsioon `n^2`,
5) kuupfunktsioon `n^3`,
6) eksponentfunktsioon `2^n`.

Regressioonikordajate leidmiseks kasutatakse teeki Commons Math [18], mis võimaldab leida polünomiaalse regressiooni parameetrite väärtused. Polünomiaalne regressioon laiendab lineaarregressiooni, mistõttu sobivad selle jaoks leitud parameetrid ka lineaarregressiooni jaoks.

Töö käigus valminud programm kasutab teisendusi, et rakendada polünomiaalset regressiooni ka logaritm- ning eksponentfunktsiooni hindamiseks. Kasutusel on kolm teisendust:

1) logaritmfunktsiooni jaoks võetakse sisendi suurustest logaritm,
2) linearitmilise seose jaoks jagatakse tööaeg logaritmiga tööajast,
3) eksponentfunktsiooni jaoks võetakse tööajast logaritm.

Linearitmilise seose tuvastamiseks kasutatakse väikse osakaaluga ka korrektsemat teisendust, kus tööaega jagatakse logaritmiga sisendi suurusest. Testimisel saadud tulemuste kohaselt see tavaliselt klassifitseerimise tulemust ei paranda, aga selle olemasolu võimaldab programmi käitumist paremini jälgida.

Polünomiaalse seose korral on andmed juba sobival kujul, teisendada pole vaja. Teisendus võimaldab kasutada polünomiaalset regressiooni ka mittepolünomiaalse funktsiooni sobivuse hindamiseks. Seejärel saab analüüsida regressiooni kaudu leitud parameetreid ning nende alusel valida kõige tõenäolisema keerukusklassi. Polünomiaalse seose tuvastamiseks kasutatakse kolmanda astme polünomiaalset regressiooni, mis sisaldab ka kuup- ja ruutliiget. Eelnevalt nimetatud mittepolünomiaalsete funktsioonide uurimiseks sooritatud teisenduste korral on lineaarregressioon ehk esimese astme polünomiaalne regressioon piisav.

5.4 Regressiooniparameetrite alusel keerukusklassi valimine

Regressiooniparameetrite tõlgendamise jaoks viidi läbi suurusjärgus 100 testi, kasutades Java funktsioone, mis sooritavad täisarvude liitmist sobilik arv kordi, et ajaline keerukus oleks mõni vaadeldav keerukusklass. Testimisel kasutatud funktsioonid on saadaval koos programmi lähtekoodiga, failides *java/Example_1.java* kuni *java/Example_5.java*. Väiksemal määral kasutati ka sarnaseid Pythoni funktsioone, aga nende korral esines sageli tavalisest palju suuremaid tööaegu, mille tõttu regressiooni sobivus oli väiksem. Seadistused valiti selliselt, et ühe jooksutamise käigus sooritataks mõõtmisi vähemalt ühe minuti jooksul.

Tulemuste analüüsimisel selgus, et logaritmfunktsiooni jaoks sooritatud teisenduse korral annab sobiva vastavuse ainult logaritmilise keerukusega funktsioon. Seevastu logaritmilise keerukuse funktsioon sobis hästi kõigi kasutatud teisenduste korral. Sellel põhjusel tuginetakse logaritmilise keerukuse tuvastamisel ainult ühe teisenduse alusel saadud vastavusele.

Kui logaritmilist keerukust ei tuvastatud, siis järgmisena vaadeldakse eksponentsiaalset keerukust. Selgus, et eksponentsiaalset keerukust kirjeldab hästi ainult eksponentsiaalfunktsiooni teisendus. Lisatingimusena kontrollitakse enne eksponentfunktsiooni kasuks otsustamist, et suurim sisendandmete suurus oleks väiksem kui 100. Praeguse arvutusvõimsusega ei ole võimalik, et eksponentfunktsioon lõpetaks töö nii suure sisendi suuruse korral.

Programmis kasutatud teek Commons Math leiab regressiooniparameetreid Levenberg-Marquardt'i algoritmi [19] abil. See on iteratiivne algoritm ning programmis kasutatakse seda iteratsioonide limiidiga 10000. Iteratsioonide arvu ülempiir aitab tuvastada olukorda, kus regressioonimudel ei vasta andmetele. Tulemuste kohaselt selle limiidiga algoritm tavaliselt ei lõpeta tööd olukorras, kus linearitmilist teisendust rakendati polünomiaalse

keerukusega funktsioonilt saadud andmetel. Seda asjaolu kasutatakse ära, et eristada linearitmilist keerukust: kui parameetrite leidmine õnnestub, siis on suur tõenäosus, et tegu ongi sellise keerukusega.

Allesjäänud keerukusklasside eristamiseks leitakse kolmanda astme polünomiaalse regressiooni kordajad. Kui ruutliikme ja kuupliikme kordajad on negatiivsed, siis on tegu lineaarse keerukusega. Kui ruutliikme kordaja on positiivne ning kuupliikme oma pole, antakse vastuseks ruutkeerukus. Ülejäänud juhtudel väljastatakse kuupkeerukus, kõrgema astme seoseid ei vaadelda. Üliväikesed positiivsed kordajad võivad tuleneda ka mõõtmisvigadest, mistõttu loetakse positiivseks arvud, mis on suuremad kui 1E-10.

Vastuseks antud keerukusklass ei pruugi olla alati õige. Programmi testimise käigus leiti, et kõige rohkem vigu tekib linearitmilise seose ning lineaarse seose eristamisel. Täpsuse hindamiseks jooksutati programmi 50 korda samade seadetega iga vaadeldava keerukusklassi jaoks. Tulemuste alusel leiti, et linearitmilise keerukusega funktsioon klassifitseeritakse enamasti linearitmiliseks (66% juhtudest) mõnikord lineaarseks (24%) ning harva ka ruutkeerukusega funktsiooniks (8%). Samuti esineb vastupidist kattumist: lineaarse keerukusega funktsioon määratakse enamasti lineaarseks (60%), aga sageli hoopis linearitmiliseks (40%).

Eksponentsiaalse keerukusega ning logaritmilise keerukusega funktsioonid klassifitseeriti kõige täpsemalt, nende tuvastamisel ei esinenud kattumisi teistsuguste keerukusklassidega. Ruut- ja kuupkeerukusega funktsioonid määratleti samuti ilma vigadeta, aga esines linearitmilise seose määramist ruutfunktsiooniks.

6. Programmi kirjeldus

6.1 Programmi üldine kuju

Käesoleva töö käigus valmis programm, mis kompileeritakse käivitamiseks failiks *time-complexity-1.0.jar*. Seda faili saab käivitada käsuga `java -jar time-complexity-1.0.jar`. Programmi lähtekood on saadaval versioonihaldussüsteemi *git* kaudu [5]. Programm kirjutati keeles Java, kasutades Java versiooni 1.8.

Projekti struktuuri haldamiseks on kasutusel rakendus Apache Maven [20]. See lihtsustab projekti jaoks vajalike teekide ülesseadmist ning sisaldab seadistust käivitamiskõlbuliku *jar*-faili ehitamiseks käsuga `mvn package`.

6.2 Programmi tööprotsess Java funktsiooni näitel

Keeles Java on kõik funktsioonid ka meetodid. Käesolevas töös tähendab Java funktsioon staatilist meetodit. Java funktsiooni käivitamiseks on vajalik JDK olemasolu – see sisaldab Java koodi kompileerimiseks vajalikke vahendeid.

Kõigepealt kompileeritakse sisendfail. Kui faili ei leitud või kompileerimine ebaõnnestus, siis väljastatakse veateade. Siin vaadeldavas näites eeldatakse, et faili *DataGen.java* ei leidu, samuti on seadistuste failis ainult failinimi ning funktsiooninimi määratud. Kuna sisendi suuruse alusel ei genereerita teistsugust sisendväärtust, kasutatakse sisendina sisendi suuruse arvulist väärtust.

Seejärel laaditakse kompileeritud klass mällu ning otsitakse sellest etteantud nimega funktsioon. Kui sellise nimega funktsiooni ei leidu, väljastatakse veateade. Edasi leitakse topeltkahendotsinguga, millise sisendini on mõistlik aega mõõta. Selle jaoks käivitatakse vaadeldav funktsioon erinevate sisenditega, kuni kulunud tööaeg jääb 1500 ning 2500 millisekundi vahele.

Pärast seda mõõdetakse ühtlaste vahedega veel 100 andmepunkti kuni leitud suurima sisendini. Seejärel alustatakse ajalise keerukuse määramist. Selleks kasutatakse peatükis 5 kirjeldatud meetodeid ning lõpuks väljastatakse Moodle'i formaadis kommentaar oletatava ajalise keerukusega, näiteks: `Comment :=>> Oletatav ajaline keerukus: O(n^2).`

6.3 Programmi seadistamine

Koostatud programmi seadistamine toimub faili *config.properties* kaudu. Selles failis on seaded kujul `nimi=väärtus`. Väärtust `1` kasutatakse tõese väärtusena ning väärtust `0` väära väärtusena. Käivitamiseks peavad olema seadistatud vähemalt järgnevad omadused:

- *source.file*: Jooksutatava faili nimi koos laiendiga .java või .py. Vastavalt faililaiendile otsitakse seda faili kaustas, mis on määratud võtmega *source.java* või *source.python*, mis vaikimisi viitavad samale kaustale, kus on käivitatav *jar*-fail.
- *function.name*: selle funktsiooni nimi, mille ajalist keerukust otsitakse.

Lisaks on võimalik erinevate parameetritega juhtida programmi tööd:

- *source.java*: Kaust, millest programm otsib laiendiga .java faile. Vaikimisi on selle väärtus praegune kaust.
- *source.python*: Kaust, millest programm otsib laiendiga .py faile. Vaikimisi on selle väärtus praegune kaust.
- *function.goal.time*: Soovitud funktsiooni tööaeg millisekundites suurima sisendi suuruse korral. Vaikimisi on selle väärtus `2000`.

- *function.goal.offset*: Suurima sisendi suuruse leidmise jaoks soovitud tööaja väärtuse lubatud kõrvalekalle. Vaikimisi on selle väärtus `0.25`, seega vaikimisi otsitakse sisendi suurust, millega tööaeg jääb `0.75*2000=1500` ning `1.25*2000=2500` millisekundi piiridesse.
- *function.n.min* ja *function.n.max*: Võimaldab piirata, millises vahemikus otsitakse suurima sisendi suurust, millega oleks määratud tööaeg. See vahemik on vaikimisi nullist kuni suurima Java *int*-täisarvuni (`0x7fffffff`).
- *point.count*: Mitu mõõtmist teha leitud suurima suuruseni. Mõõtmised sooritatakse ühtlaselt vahemikus *function.n.min* ja *function.n.max* vahel. Vaikimisi sooritatakse `100` sellist mõõtmist.
- *point.only*: Kui selle väärtus on `1`, siis suurima sisendi leidmist ei toimu, selle asemel tehakse ainult ühtlased mõõtmised vahemikus *function.n.min* ja *function.n.max*. Vaikimisi on selle väärtus `0`.

Samuti on olemas seadistused millega määrata, millist väljundit programm näitab:

- *output.printprogress*: Kui see on `1`, siis väljastatakse pärast iga funktsioonikäivitust selle käivituse aeg. Vaikimisi on selle väärtus `0`.
- *output.printtimes*: Kui see on `1`, siis väljastatakse pärast funktsioonikäivitusi mõõtmistulemused sorteeritult sisendi suuruse alusel. Vaikimisi on selle väärtus `1`.
- *output.regression*: Kui see on `1`, siis väljastatakse tulemuste alusel leitud regressioonikordajad iga teisenduse jaoks. Kordajate esimene väärtus on vabaliige, teine lineaarliige ja nii edasi. Vaikimisi on selle väärtus `0`.
- *output.predictions*: Kui see on `1`, siis väljastatakse leitud regressioonikordajate alusel leitud ennustatud väärtused ja tegelikud väärtused. Samuti väljastatakse hälvete ruutude summa. Kuna kasutusel on erinevad teisendused, siis ei ole erinevate teisenduste kaudu saadud hälvete summad võrreldavad. Seda väljundit saab kasutada analüüsimiseks, milline ajaline keerukus andmete kohaselt kõige paremini sobib. Vaikimisi on selle väärtus `0`.

Etteantud testikomplekti kasutamiseks on vaja järgnevaid seadistusi:

- *mode=manual*: See annab teada, et tuleb kasutada testikomplekti. Vaikimisi kasutatakse sisendi genereerimist.
- *source.tests*: Kaust, millest programm otsib testjuhte. Testjuhud on kirjeldatud failidega *inputN.txt*, *metaN.txt* ja *outputN.txt*, kus N on testjuhu number, mis algab arvust 1. Faili *inputN.txt* sisu suunatakse testitava programmi standardsisendisse, fail *metaN.txt* sisaldab rida `input_size=X`, kus X on selle testjuhu sisendi suuruse arvuline väärtus. Käivitamisel kontrollitakse, kas funktsioon kirjutab standardväljundisse samad read, mis on failis *outputN.txt*.

6.4 Funktsiooni sisendväärtuse genereerimine

Kui funktsioon, mille ajalist keerukust vaadeldakse, kasutab sisendväärtusena midagi muud kui täisarvu (Java *long*, Pythoni *int*), siis on vaja funktsiooni, mis teisendab sisendi suuruse sobivat tüüpi väärtuseks. Java-meetodi teisenduse jaoks saab kasutada faili *DataGen.java*, kus on defineeritud meetod signatuuriga `public static Object getInput(long n)`. Pythoni funktsiooni jaoks otsitakse funktsiooni `def getInput(n: int) -> object` failist *datagen.py*. Kumbki fail peab kasutamiseks leiduma kaustas, millest vastava laiendiga faile otsitakse (vaikimisi jooksev kataloog).

Faili *datagen.py* näide, mis koostab listi, kus on *n* elementi:

```
def getInput(n: int) -> object:
    return list(range(n))
```

Faili *DataGen.java* näide, mis koostab järjendi, kus on *n* juhuslikku täisarvu:

```
import java.util.concurrent.ThreadLocalRandom;

public class DataGen {
    public static Object getInput(long n) {
        return generateLongArray(n, 0, Long.MAX_VALUE);
    }

    private static long[] generateLongArray(long n, long min, long max) {
        ThreadLocalRandom random = ThreadLocalRandom.current();
        long[] array = new long[Math.toIntExact(n)];
        for (int i = 0; i < n; i++) {
            array[i] = random.nextLong(min, max);
        }
        return array;
    }
}
```

6.5 Programmi ülesseadmine Moodle'i keskkonnas

Moodle'i keskkonnas on programmeerimisülesannete jaoks kasutusel ülesandetüüp VPL-harjutus. Töö käigus valminud programmi kasutamiseks on vaja laadida programmi *jar*-fail ning *config.properties* „Täpsemate seadete" juurest leitavate „Käivituse failide" nimekirja. Samuti tuleb „Jooksutamisel vajaminevate failide" juures märgistada linnukesega *config.properties* ning *jar*-fail. Käivituse failide juures tuleb faili nimega *vpl_evaluate.sh* lisada käsud *jar*-faili käivitamiseks. Selleks sobib näiteks failisisu

```
#!/bin/bash
base64 -d time-complexity-1.0.jar.b64 > time-complexity.jar
echo "#!/bin/bash" > vpl_execution
echo "java -jar time-complexity.jar" >> vpl_execution
chmod +x vpl_execution
```

Lisaks on soovituslik lisada „Nõutud failide" seadistuse kaudu failis *config.properties* antud nimega fail, milles on etteantud nimega funktsioon. Näiteks seadetega

```
source.file=Peafail.java
function.name=start
```

sobib fail *Peafail.java* sisuga

```
public class Peafail {
    public static void start(long n) {
        // Funktsiooni keha
    }
}
```

Sisendi genereerimiseks tuleb üles laadida ning jooksutamisel vajaminevaks märkida vastavalt *datagen.py* või *DataGen.java*. Samuti tasub üle vaadata, et „Käivituse seadetes" oleks kontrollimine lubatud.

Etteantud testkomplekti kasutamiseks tuleb lisada käivitamise failide hulka failid *inputN.txt*, *metaN.txt* ja *outputN.txt*, kus N on testjuhu number, mis algab arvust 1. Neid faile otsitakse seadistusega *source.tests* määratud kaustast ning need failid on jooksutamisel vajalikud. Failide sisu selgitatakse seadistuse *source.tests* kirjelduses, jaotises 6.3.

6.6 Programmi käivitamine kohalikus arvutis

Oma arvutis käivitamiseks on vaja kompileerida projekt Maveni toega arenduskeskkonnas või käsuga `mvn package`. Selle tulemusena valmib *jar*-fail, mida saab käivitada käsuga `java -jar time-complexity-1.0.jar`. Sarnaselt Moodle'i keskkonnas käivitamisele on ka siin vajalik fail *config.properties* samas kaustas, kus *jar*-fail.

7. Kokkuvõte

Käesoleva bakalaureusetöö tulemusena valmis funktsiooni ajalise keerukuse leidmise programm. Seda saab kasutada keeltes Java ning Python koostatud funktsioonide ajalise keerukuse leidmiseks nii oma arvutis kui ka Moodle'i käivitusserveris VPL-harjutuse kaudu.

Programm leiab funktsiooni ajalise keerukuse ligikaudselt, kogudes andmeid funktsiooni tööaja mõõtmistest erinevate sisendite korral. Funktsiooni tööaja varieerumise tõttu võidakse väljastada ka tegelikkusest erinev keerukus.

Üks olulisemaid programmi edasiarendamise võimalusi on keerukuse leidmise täpsuse suurendamine. Seda saab teha kasutades mitut erinevat arvutusmudelit või teeki tööaegadest keerukuse arvutamiseks. Samuti võib andmeid eelnevalt töödelda, et eristada käivitusi, mille jaoks kulus palju rohkem aega kui mõnel käivitusel, mis kasutas isegi suuremat sisendit.

Samuti on võimalik ühildada funktsiooni tööaja mõõtmine funktsiooni korrektsuse kontrolliga. Valminud programm teostab ainult kõige lihtsamat funktsiooni väljundi kontrolli. Moodle'i keskkonnas on seda mõistlik teha näidislahenduse tulemusega võrdlemise kaudu.

Lisaks saab uurida algoritmi mälukasutuse sõltuvust sisendi suurusest – mahulist keerukust. Mälukasutuse hulka arvestatakse nii sisendi jaoks kasutatav ruum kui ka suurim algoritmi töö käigus vajaminev mäluhulk.

8. Viidatud kirjandus

[1] K.-A. Puulmann, „Pythoni teek programmeerimisülesannete automaatseks testimiseks," 2014. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=40971&year=0. [Vaadatud 08 05 2018].

[2] A. Annamaa, „Python Grader," https://github.com/aivarannamaa/python-grader. [Vaadatud 11 05 2018].

[3] K. Papli, „Python Grader," https://github.com/kspar/python-grader. [Vaadatud 27 04 2018].

[4] „Moodle virtual programming lab," http://vpl.dis.ulpgc.es/. [Vaadatud 22 04 2018].

[5] T. Lõhmus, „ProditorMagnus/timecomplexity," https://github.com/ProditorMagnus/timecomplexity.

[6] J. Penjam, „Keerukusteooria elemente," 09 12 2007. http://www.cs.ioc.ee/~jaan/teorinf/konspekt/loeng9.pdf. [Vaadatud 22 04 2018].

[7] K. Papli, „wheels/rattad.py," https://github.com/kspar/wheels/blob/master/rattad.py. [Vaadatud 07 05 2018].

[8] „JUnit 4 / About," https://junit.org/junit4/. [Vaadatud 08 05 2018].

[9] D. Zindros, „A Gentle Introduction to Algorithm Complexity Analysis," https://discrete.gr/complexity/. [Vaadatud 03 05 2018].

[10] D. Bridge, „Measuring and Analysing Algorithm Complexity," http://www.cs.ucc.ie/~dgb/courses/toc/handout23.pdf. [Vaadatud 03 05 2018].

[11] M. Kruse, „Perfrewrite. Program Complexity Analysis via Source Code Instrumentation," 07 09 2014. https://arxiv.org/pdf/1409.2089.pdf. [Vaadatud 25 04 2018].

[12] B. Venners, „The Java class file lifestyle," 01 07 1996. https://www.javaworld.com/article/2077193/core-java/the-java-class-file-lifestyle.html. [Vaadatud 08 05 2018].

[13] G. McCluskey, „Using Java Reflection," 01 1998. http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html. [Vaadatud 24 04 2018].

[14] „29.4. __main__ — Top-level script environment," https://docs.python.org/3/library/__main__.html. [Kasutatud 24 04 2018].

[15] V. S. Adamchik, „Algorithmic Complexity," 2009. https://www.cs.cmu.edu/~adamchik/15-121/lectures/Algorithmic%20Complexity/complexity.html. [Vaadatud 08 05 2018].

[16] Investopedia, „Regression," https://www.investopedia.com/terms/r/regression.asp. [Vaadatud 09 05 2018].

[17] Yale University Department of Statistics, „Linear Regression," http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm. [Vaadatud 09 05 2018].

[18] The Apache Software Foundation, „Commons Math: The Apache Commons Mathematics Library," http://commons.apache.org/proper/commons-math/. [Vaadatud 09 05 2018].

[19] M. I. A. Lourakis, „A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar,“ 11 02 2005. https://www.researchgate.net/publication/239328019_A_Brief_Description_of_the_Levenberg-Marquardt_Algorithm_Implemened_by_levmar. [Vaadatud 10 05 2018].

[20] Apache Software Foundation, „Welcome to Apache Maven,“ https://maven.apache.org/. [Vaadatud 09 05 2018].

Lisad

I. Lähtekood

Programmi lähtekood on saadaval aadressil:

https://github.com/ProditorMagnus/timecomplexity

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Toom Lõhmus**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Programm funktsiooni ajalise keerukuse leidmiseks,
(*lõputöö pealkiri*)

mille juhendaja on Reimo Palm
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **14.05.2018**