TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Romet Vinnal

Automaatkontrolli logide parsimine ja interaktiivne visualiseerimine aine Algoritmid ja andmestruktuurid näitel

Bakalaureusetöö (9 EAP)

Juhendajad:
Tõnis Hendrik Hlebnikov, MSc
Ahti Põder, PhD

Tartu 2025

Automaatkontrolli logide parsimine ja interaktiivne visualiseerimine aine Algoritmid ja andmestruktuurid näitel

Lühikokkuvõte:

Bakalaureusetöö eesmärk oli luua lokaalne ja terviklik veebirakendus, mis võimaldab Tartu Ülikooli kursuse Algoritmid ja andmestruktuurid automaatkontrolli logifailide struktureerimist, analüüsi ja visualiseerimist. Väljatöötatud süsteem hõlmab kolme peamist komponenti: parsimistoru, analüüsikihti ja visualiseerimismoodulit. Rakendus on realiseeritud Flaski, Pandase ja Vega-Altairi teekide abil ning võimaldab õppejõududel laadida üles logifaile ning analüüsida saadud tulemusi interaktiivsete diagrammide kaudu veebibrauseris. Loodud süsteem suurendab hindamisandmete analüüsivõimalusi, aitab kaasa kursuse sisulisele täiustamisele ning pakub võimalusi edasiseks arendamiseks ja laiendamiseks.

Võtmesõnad: Automaatkontroll, parsimine, veebirakendus, andmeanalüüs

CERCS: P175 Informaatika, süsteemiteooria

Parsing and Interactive Visualisation of Automated Grading Logs: A Case Study on the Course Algorithms and Data Structures

Abstract:

The objective of this bachelor’s thesis was to develop a local and integrated web application for structuring, analysing, and visualising the automated grading log files of the University of Tartu's course Algorithms and Data Structures. The developed system consists of three main components: a parsing pipeline, an analysis layer, and a visualisation module. The application was implemented using the Flask framework, the Pandas library, and the Vega-Altair visualisation tools, enabling instructors to upload log files and analyse the resulting data through interactive charts in a web browser. The created system enhances the possibilities for assessment data analysis, supports the continuous improvement of the course content, and provides a foundation for future development and expansion.

Keywords: Automated assessment, parsing, web application, data analysis

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Automaatne ülesannete hindamine on muutunud oluliseks õppeprotsessi osaks moodsa programmeerimisõppe kontekstis. Kursuste suure osalejate arvu ja korduvate ülesannete tõttu on vajalik digitaalne süsteem, mis suudab esitused kiiresti testida, hinnata ja tagasisidestada. Ainuüksi hindetabelite väljastamisest ei piisa, sest hindamistorustik salvestab märkimisväärses koguses olulist teavet – esituste ajatemplid, testide tulemused, punktigrupid, kompileerimise logid, mis võimaldavad õppijate käitumismustreid ja õppeprotsessi efektiivsust hinnata oluliselt põhjalikumalt.

Käesoleva töö eesmärk on luua terviklahendus, mis omandab, töötleb ja visualiseerib aine Algoritmide ja andmestruktuuride automaatkontrolli logifailide andmed, võimaldades õppejõududel saada ülevaade esituskäitumise mustritest kursusel, hinnata ülesannete raskusastet punktigrupi tasandil ning pakkuda alusandmeid hindemudelite ja õpet parendavate sekkumiste arendamiseks.

Töö koosneb andmetorustikust, mis teisendab heterogeensed logifailid struktureeritud andmestikuks, analüüsikihist, mis arvutab kursuse võtmemõõdikud ning visualiseerimismoodulist tulemuste interaktiivsete graafikute esitamiseks veebipõhises keskkonnas.

1. Taust ja varasemad lahendused

1.1 Automaatkontroll ja nende logid

Automaatne programmeerimisülesannete hindamise ajalugu ulatub 1960. aastatesse. [1] Programmeerimisülesanded on arvutiteaduse õppekava keskne osa ning nende mõjusaks kasutamiseks tuleb nii ülesanded kui ka hindamine läbimõeldult kavandada. [2] Käsitsi hindamine võib olla aeganõudev ning õppejõule koormav. Suur tudengite arv nõuab rohkem hindajaid, millega võib kaasneda subjektiivsus ja ebaühtlane hindamine. Seetõttu on üha enam vaja automaatseid hindamisvahendeid. [3]

Automaatkontrollid analüüsivad koodi süntaksivigade leidmiseks, kontrollivad lahenduse korrektsust ja koodistiili, muuhulgas hindavad programmeerimisoskust ja probleemi-lahendamisevõimet ning annavad kiiret tagasisidet nii õppejõule kui ka tudengile. Lisaks ei sea automaattööriistad oma efektiivsuse poolest piirangut õppejõu antavate ülesannete hulgale. [2] Automaatkontrollid võimaldavad iseseisvat ülesannete hindamist, annavad tudengile kiire tagasiside ja lubavad vajadusel tööd uuesti esitada. Selline lähenemine tõstab üliõpilaste rahulolu ja aitab neil paremaid tulemusi saavutada. [3] Eelnevast tulenevalt on ka selle töö aluseks oleva aine puhul automaatkontrollidel kandev roll.

Aine Algoritmid ja andmestruktuurid (ainekood LTAT.03.005) kodutööde kontrolliks kasutatakse Moodle'i keskkonnas olevaid automaatkontrolle. Automaatkontrollid väljastavad iga tudengi tehtud esituse kohta logifailid. Logifaile on õppejõul võimalik alla laadida vastava kodutöö esituse hindetabelist, vajutades nupule „Laadi alla kõik esitused."

Tulemuseks on ZIP fail joonises 1 toodud struktuuriga:

```
kodutööd.zip/
├── eesnimi_perenimi_id_moodle_id/
│   └── aasta-kuu-päev-tund-minut-sekund/
│       └── esitus.java
│   └── aasta-kuu-päev-tund-minut-sekund.ceg/
│       └── aasta-kuu-päev-tund-minut-sekund.ceg/compilation.txt
│       └── aasta-kuu-päev-tund-minut-sekund.ceg/execution.txt
│       └── aasta-kuu-päev-tund-minut-sekund.ceg/grade.txt
```

Joonis 1. Automaatkontrolli väljastatud logifailide struktuur.

Automaatkontrolli poolt väljastatavad andmed on salvestatud faili *execution.txt*, sisaldades seega kõige rohkem andmeid. *Execution.txt* fail sisaldab infot automaatkontrolli käivituse kohta ehk väljastatud hinnet, automaatkontrolli käivitusaega, lahenduse jooksmisaega ning koguhinde moodustavad punktigruppe. Samuti saab kaustanimest välja lugeda ajahetke, millal tudeng oma lahenduse kontrollimiseks üles laadis. Lisas 1 on toodud ka näide *execution.txt* faili sisust.

1.2 Varasemad lahendused ja puudused

Aine Algoritmid ja andmestruktuurid Moodle'i lehel on iga kodutöö kohta hindetabelid, mille ridades on kajastatud esituse teinud tudengid ning veergudes tudengite nimed, viimase esituse aeg, esituste arv, hinne, hindaja ning hindamise aeg.

Kirjeldatud tabel on hindamise seisukohast piisavalt praktiline, kuid ei kajasta konkreetse kodutöö võtmemõõdikuid. Parema ja efektiivsema tagasiside saamiseks võiksid andmed olla visualiseeritud viisil, mis annaks kodutöö tulemustest kokkuvõtlikuma ülevaate. Selleks võiksid õppejõule nähtavad olla diagrammid ja graafikud, kus erinevate näitajate pinnalt saaks teha kaalukamaid järeldusi. Näiteks võiks esituste koguarv anda aimu kodutöö keerukusest ning automaatkontrolli kasutusintensiivsusest. Kodutöö esitamise osalust saaks aga hinnata esituse teinud üliõpilaste arvu põhjal, mida saaks omakorda võrrelda teiste kodutööde kontekstis. Automaatkontrolli ülemäärase kasutamise tuvastamiseks võiks vaadata ka suurimat esituste arvu tudengi poolt.

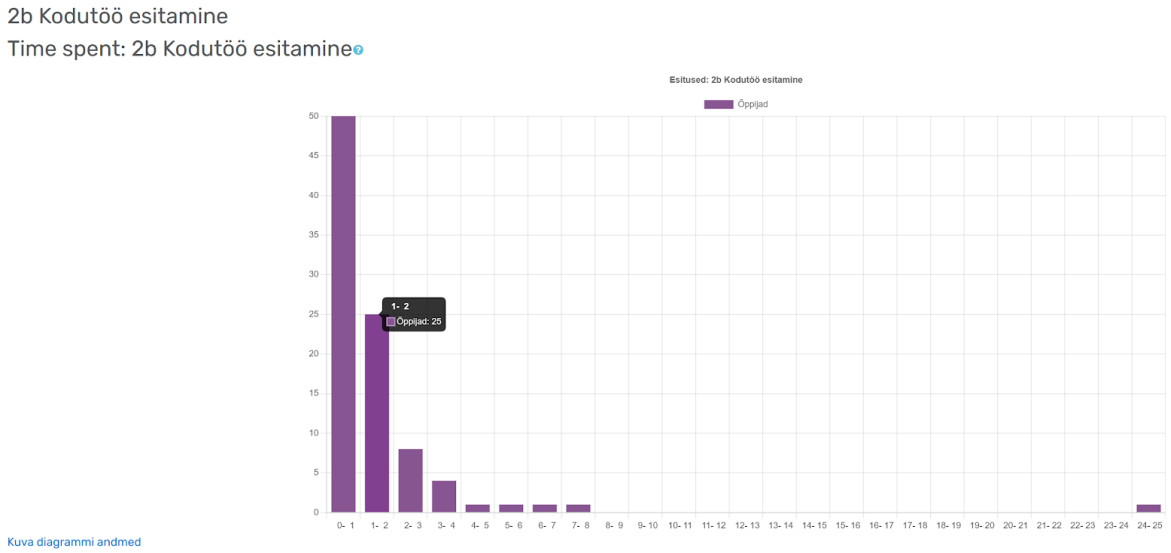
Samuti võiksid tabelid ja graafikud anda aimdust tudengite esitamisharjumustest ja üldistest tendentsidest. Õppejõule võiks töö tulemusena tekkida ülevaade sellest, millal tudengid kõige enam esitusi teevad ja millal tehakse viimased esitused lõplikuks hindamiseks. Tudengite hinnete ja esituste arvu seos saab ilmestada esituste kvaliteeti ning seda, kui palju on soovitud tulemuse jaoks esitusi tehtud. Samuti võiks kajastuda tudengite hinded punktigruppide kaupa ning näiteks ka see, kas ja kui palju lahendatakse vabatahtlikke ülesandeid.

Kui hindetabelis valida konkreetne tudeng, kuvatakse menüüs „Eelmine esituste loetelu" all kaks graafikut. Esimene on joondiagramm, mille abstsissteljel on esituse järjekorranumber ning mille ordinaatteljel on esituse Java faili sümbolite arv. Eeldatavasti illustreerib diagramm esitatud töö mahu muutumist esituste vahel. Teine on aga tulpdiagramm, mis jaotab

esitustevahelise aja tööperioodideks ehk abstsissteljel on tööperioodi järjekorranumber ning ordinaatteljel on tööperioodi ajakulu.

Mõlema graafiku puhul on võimalik näha diagrammis kasutatavaid andmeid tabelina, kuid puudu on hõlpsasti mõistetavad pealkirjad, andmete kirjeldused ning üheselt mõistetavad interaktiivsed annotatsioonid.

Hindetabelis esituste arvu päise peale vajutades kuvatakse ka tulpdiagramm, kus esitustevaheline aeg tundides on vastavusse seatud selles tulbas olevate õpilaste arvuga.



Joonis 2. Algoritmide ja andmestruktuuride 2024. aasta 2. kodutöö b variandi ajavahemike diagramm.

Joonisel kujutatud diagrammi on keeruline lugeda ning see pole kuigi informatiivne. Diagrammi pealdisest ei selgu, milliseid andmeid on kujutatud, mis on antud graafiku eesmärk ning milliseid järeldusi selle põhjal teha saab. Samuti puuduvad telgedel päised ning ka tulba kohal hõljudes ilmuv annotatsioon ei anna vaatajale piisavat infot graafikust mõistmiseks.

Moodle'i graafikutel põhineva tagasiside lakoonilisus ning esitatud andmete vähene selgus olid peamised ajendid käesoleva töö loomisel.

2. Metoodika ja tehniline lahendus

Selles peatükis kirjeldatakse lahenduse kavandamist ja teostamist tehnilisest vaatenurgast. Alustuseks antakse ülevaade teoreetilistest alustest, millele töö tugineb, sealhulgas parsimise põhimõtetest ja andmete analüüsi alustest. Seejärel tutvustatakse valitud tehnoloogiaid ning põhjendatakse nende sobivust ülesande lahendamiseks. Lisaks kirjeldatakse süsteemi arhitektuuri, töövoogu ning olulisemaid komponente nagu parsimistoru, analüüsikiht ja visualiseerimismoodul. Peatüki lõpus käsitletakse realiseerimise tehnilisi detaile ja tehtud valikute mõju süsteemi töökindlusele ja laiendatavusele.

2.1 Teoreetilised lähtekohad

2.1.1 Parsimine

Parsimise ehk süntaktilise analüüsi all mõistetakse sümbolite jadade grammatikareeglite põhjalist analüüsi. Arvutiteaduses tähendab parsimine formaalset analüüsi, kus arvuti jagab lause või sõnad grammatilisteks osadeks ja moodustab sellest struktuuri ehk parsimispuu, näidates sõnadevahelisi süntaktilisi ja semantilisi suhteid.

Parsimise algoritme saab eristada kolmel moel:

1. Otsingustrateegia järgi:
    - sügavuti otsing (*depth-first*);
    - laiuti otsing (*breadth-first*).
2. Struktuuri loomise suuna järgi:
    - alt üles (*bottom-up*), alustades sõnadest ja liikudes puu tipuni;
    - ülevalt alla (*top-down*), alustades puu tipust ja liikudes sõnadeni.
3. Sisendi töötlemise järjekorra järgi:
    - tavaliselt vasakult paremale;
    - vahel paremalt vasakule või keskelt väljapoole (levinud kõnetuvastuses).

Antud töö kontekstis on kasutatud Lark teegi LARL(1) parserit, mis toimib sügavuti ja loob struktuuri alt üles.

Alt üles (*bottom-up*) parsimisel kasutatakse grammatikareegleid vastupidises suunas (paremalt vasakule) ja parsitakse pinu abil järgnevate sammudega:

- pinusse lisatakse järjest sümboleid;

- kui pinu tipus olevad sümbolid vastavad mõne reegli paremale poolele, eemaldatakse need ja asendatakse reegli vasaku poolega (*reduce* - taanda);
- kui pinu tipus sobivaid sümboleid pole, lisatakse sinna järgmine sisendsümbol (*shift* - nihuta).

Meetodi sammudest tulenevalt nimetatakse alt üles parsereid sageli ka *shift-reduce* ehk nihuta-taanda parseriteks.

Võtmeküsimusteks alt üles parsimisel on nihutamise ja taandamise hetke ära tajumine ning millise reegli järgi taandamine läbi viia. Reeglite äratundmise jaoks kasutatakse lõplikku automaati, mis tunneb ära, millised sümbolid pinus moodustavad sobiva reegli (*handle*). [4]

2.1.2 Andmete analüüsi ja visualiseerimise alused

Andmete analüüsi ja visualiseerimise protsess on käesolevas töös keskse tähtsusega, kuna automaatkontrolli logidest saadud andmed tuleb muuta kiiresti mõistetavateks ja analüüsitavateks graafilisteks esitusteks. Selle protsessi elluviimiseks on oluline määratleda selged nõuded tööriistadele, mida andmete töötlemiseks ja esitamiseks kasutatakse.

Töös seatud funktsionaalsed nõuded visualiseerimisteekidele olid järgmised:

- teek võimaldab luua interaktiivseid graafikuid;
- teek pidi pakkuma deklaratiivset või minimaalset käsitsi kohandamisega nõudvat programmeerimisliidest, et vähendada graafikute koostamise keerukust.

Tehniliste nõuete kohaselt:

- teek pidi toetama Pandase DataFrame andmestruktuure kui peamisi andmeallikaid;
- teek pidi olema hõlpsasti ühildatav brauseripõhise veebirakendusega, eelistatult JSON vormingus;
- graafikute lõplik joonistamine pidi toimuma brauseri poolel, vältides serveripoolset keerukat pildigeneratsiooni.

Visualiseerimisprotsessi kavandamisel oli oluline mõista kahe programmeerimisparadigma, deklaratiivse ja imperatiivse, põhimõttelisi erinevusi. Deklaratiivne lähenemine keskendub soovitud tulemuse kirjeldamisele ilma samm-sammulise tegevusjuhiseta – kasutaja määrab, milliseid andmeid ja millises vormis soovitakse näha, jättes visuaali koostamise tehnilised üksikasjad teegi ülesandeks. Seevastu imperatiivses lähenemises peab kasutaja kirjeldama täpselt kõik sammud, kuidas andmetest graafik kujundada.

Kuna töö eesmärk on luua esituste andmetest kiiresti ning süsteemselt selged ja interaktiivsed visuaalid liigse käsitsi seadistamiseta, oli deklaratiivne lähenemine kõige sobivam. See võimaldas toetada brauseripõhist graafikute joonistamist, vähendada serveri töökoormust ning tagada lahenduse edasine laiendamine.

2.2 Kasutatud tehnoloogiad ja nende valik

Selles alapeatükis kirjeldatakse, milliseid tarkvaratehnoloogiaid on käesoleva töö realiseerimiseks kasutatud ning miks just need tehnoloogiad valiti. Valikuprotsessi keskmes oli vajadus töökindla, laiendatava ja mõõduka ressursikasutusega lahenduse järele, mis sobituks automaatkontrolli logide töötlemise ja visualiseerimise konteksti. Eraldi käsitletakse programmeerimiskeelt Python ja tema andmetöötluse raamistikke, veebirakenduse loomiseks valitud Flask raamistikku, parsimisteeki Lark, andmeanalüüsi tööriista Pandas ning visualiseerimiseks kasutatud Vega-Altairi ja VegaEmbed skripti kombinatsiooni. Samuti tutvustatakse kasutajaliidese loomisel rakendatud Bootstrapi raamistikku, mille eesmärk oli tagada aknasuurusele reageeriv ja ühtlane kujundus. Iga valitud tehnoloogia roll ja tugevused selgitatakse lähtuvalt töö eesmärkidest ja lahendatavatest probleemidest.

2.2.1 Python

Automaatkontrolli esitustest info välja lugemiseks on kasutatud programmeerimiskeele Pythoni[1] versiooni 3.9. Selle valiku põhjuseks on Pythoni laialdane kasutus andmeteaduse valdkonnas ning rikkalik ökosüsteem andmetöötluseks sobivate teekidega. Python on koos R-keelega üks enim kasutatavaid programmeerimiskeeli andmeteaduses 2016. aasta seisuga. [5] Versioon 3.9 pakub head ühilduvust tööks vajalike teekidega.

Töös on kasutatud mitmeid olulisemaid Pythoni andmetöötluse teeke, sealhulgas NumPy ja Pandas.

2.2.2 Flask

Serveripoolne rakendus on loodud Flask teegiga. Flask pakub kerget veebirakenduse raamistikku, millel on sisseehitatud URL-marsruutimine, staatiliste varade jagamine ja Jinja mallimootor.[2] Käesolevas lahenduses kasutatakse Flaski peamiselt REST-stiilis rakendusliidese (edaspidi API) loomiseks, mille kaudu edastatakse Altair teegi graafikute

[1] https://www.python.org/
[2] https://flask.palletsprojects.com/en/stable/

JSON-spetsifikatsioonid ning kursuse analüütilised andmestikud. Raamistik on valitud peamiselt selle kihilise ülesehituse ja minimaalse baasfunktsionaalsuse tõttu, mis sobib hästi väikese jalajäljega andmetorustiku teenindamiseks. Tänu kihilisusele saab vajaduse korral lisada funktsioone samm-sammult ning käivitamiseks pole suurt konfiguratsiooni tarvis.

2.2.3 Lark

Lark[3] on avatud lähtekoodiga Pythoni parsimisteek, mille arhitektuur keskendub ergonoomilisusele, jõudlusele ja modulaarsele ülesehitusele. Teek eraldab deklaratiivse grammatika programmikoodist, rõhudes loetavusele. Kasutaja seadistamise vaeva vähendatakse põhimõttega, et parser määrab optimaalsed töörežiimid ise ning ehitab alati parsimispuu, võimaldades süntaktilist teavet töödelda etapiviisiliselt[4].

Automaatkontrolli *execution.txt* logifailid moodustavad struktureeritud, kuid varieeruva tekstide voo, mille korrektne tõlgendamine eeldab kontekstivaba grammatikat. Lark teeki eelistati just oma kasutamislihtsuse ja sisse ehitatud LALR(1) parseri tõttu, võimaldades eelkompileeritud parseriga töödelda faile lineaarse ajakompleksusega. Teegi Transformer liides vähendab koodi mahtu, sest võimaldab parsimispuu kergesti teisandada lihtsasti töödeldavasse struktuuri vaid vajaliku infoga.

2.2.4 Pandas

Andmetöötluse ja analüüsi keskseks tööriistaks on kasutatud Pythoni teeki Pandas[5], mis võimaldab struktureeritud andmete tõhusat käitlemist. Pandas pakub kahte peamist andmestruktuuri: Series[6] ja DataFrame[7], mis võimaldavad tabelandmete laadimist, filtreerimist, transformeerimist ja agregeerimist.

Teek sobib hästi nii väikeste kui ka suuremahuliste andmehulkade käsitlemiseks ning võimaldab mitmesuguseid operatsioone, nagu puuduolevate väärtuste töötlemine, ajas järjestatud andmete sorteerimine, mitmetasandiline grupeerimine ja statistilise näitajate arvutamine. Tänu laiale funktsioonidevalikule on Pandas andmeteaduse ja masinõppe töövoogudes üks enim kasutatavaid töövahendeid.

[3] https://lark-parser.readthedocs.io/en/latest/
[4] https://lark-parser.readthedocs.io/en/latest/philosophy.html#design-choices
[5] https://pandas.pydata.org/docs/index.html
[6] https://pandas.pydata.org/docs/user_guide/dsintro.html#series
[7] https://pandas.pydata.org/docs/user_guide/dsintro.html#dataframe

Käesolevas töös kasutatakse Pandast esmase andmetöötluse tarbeks logifailidest struktureeritud andmestike koostamiseks, esituste, hinnete ja punktigruppide analüüsimiseks ning visualiseerimiseks suunatud andmeväljade ettevalmistamiseks.

Pandas võimaldab andmeid tõhusalt ühendada ja ümber kujundada, mis on vajalik edasiste sammude, näiteks visualiseerimise (Vega-Altair abil) või masinõppeks kasutatava sisendi koostamise jaoks. Teegi kasutamine muudab töövoo selgemaks, aitab tulemusi korrata ja tagab andmeanalüüsi õigsuse.

2.2.5 Vega-Altair ja VegaEmbed

Andmete visualiseerimiseks on kasutatud Pythoni vabavaralist teeki Vega-Altair, mille esmane väljalase ilmus 2016. aastal.[8] Altair põhineb Vega ja Vega-Lite visualiseerimisgrammatikal, mille abil teisendatakse andmed automaatselt JSON-vormingusse. Käesolevas töös visualisatsioonid joonistatakse veebilehel VegaEmbed[9] skripti abil, mis võimaldab interaktiivsete graafikute paindlikku manustamist ja kuvamist veebibrauseris.

Altair on deklaratiivne statistilise visualiseerimise teek, mille tugevuseks on lihtne ja loogiline programeerimisliides. Visualiseerimisreeglid määratakse koodis deklaratiivselt, mitte samm-sammulise protsessina. [6] Graafiku loomisel tuleb määrata andmeallikana Pandas andmeraam ning määratleda visualiseeritavad veerud vastavalt telgedele.

Selline lähenemine toetab paindlikku ja taaskasutatavat töövoogu, kus andmed ja visualiseerimisloogika on lahutatud graafiku koostamisest. Altair võimaldab määratleda visuaalseid parameetreid, mis aitavad kasutajal andmetest selgemalt aru saada, s.h tuvastada mustreid, anomaaliaid või trende. Interaktiivsus saavutatakse tänu VegaEmbed skripti võimekusele kuvada graafikuid koos tööriistadega, mis võimaldavad näiteks andmepunktide esiletõstmist või filtreerimist.

2.2.6 Bootstrap

Veebiliidese kujundamisel rakendatakse Bootstrapi (v. 5) komponent- ja abiklassidel põhinevat CSS-raamistikku. Bootstrap sisaldab reageeriva paigutuse ruudustikku, valmis UI-komponente (nt navigeerimisribad, kaardid, märguanded) ning utiliidiklasse, mille abil on võimalik kiiresti tagada ühtne visuaalne stiil eri ekraanisuurustel.[10] Lahenduses kasutatakse Bootstrapi põhiliselt

[8] https://altair-viz.github.io/index.html
[9] https://vega.github.io/vega/usage/#embed
[10] https://getbootstrap.com/docs/5.3/getting-started/introduction/

struktuursete elementide ja kujundusutiliitide tarbeks. Raamistik on valitud tänu laialdasele brauseritoele ja madalale õppimiskünnisele, mis võimaldab keskenduda andmete visualiseerimisele ilma märkimisväärset aega stiilide käsitsi loomisele kulutamata.

2.3 Süsteemi arhitektuur

Süsteemi arhitektuur on kavandatud kihilise andmetorustikuna, mis ühendab endas logifailide struktureerimise, analüüsi ja tulemuste veebipõhise esitamise. Kolm põhilist funktsionaalset kihti – parsimiskiht, analüüsikiht ja visualiseerimismoodul – on seotud Flask-põhise veebiserveri kaudu, mis vahendab nii failide üleslaadimist kui ka rakendusliidese (API) kaudu andmevahetust.

Projekti struktuur:

```
grade-analysis/
├── analysis.py
├── app.py
├── chart_api.py
├── data_store.py
├── execution_grammar.py
├── helpers.py
├── parse_homework.py
├── parser_lark.py
├── utils.py
├── templates/
│  └── dashboard.html
│  └── uploads.html
├── static/
│  └── chart_endpoints.js
│  └── datepicker.js
│  └── other.js
├── requirements.txt
├── README.md
```

Joonis 3. Projekti failistruktuur.

Järgnevalt on kirjeldatud arhitektuuri töövoona kodutöö üleslaadimisest visualiseeringuteni.

2.3.1 Parsimistoru

Töövoog algab siis, kui õppejõud käivitab faili *app.py*. Server alustab tööd arvuti sisemisel aadressil ja pordil, milleks on vaikimisi http://localhost:5000. See tähendab, et rakendus on kättesaadav kasutaja enda arvutis, porti 5000 kuulava veebiserveri kaudu.

Pärast rakenduse edukat käivitamist saab kasutaja avada brauseris üleslaadimisvormi, mille kaudu saadetakse ZIP-fail serverisse. Moodle'ist pärit kodutöö automaatkontrolli logi ZIP

arhiivi üleslaadimise järel suunatakse see automaatselt andmetorustikku, kus kontrollitakse faililaiendit ja käivitatakse funktsioon *parse_homework(zip_path, batch_size=70)*.

Funktsioon tuvastab igast tudengi kaustast faili *execution.txt* ning jagab need vaikesuurusega partiideks paralleelseks töötlemiseks. Iga partii läbib järgmised sammud: esmalt puhastatakse failid regulaaravaldistega üleliigsest mürast, seejärel parsitakse need Lark teegi LARL(1)[11] parseriga kasutades faili struktuuri kirjeldavat formaalset grammatikat. Aine Algoritmid ja andmestruktuurid automaatkontrolli logifaili *execution.txt* grammatika on toodud lisas 2.

Parsitud puustruktuur teisendatakse Lark teegi Transformeri[12] abil sõnastikuks, mis sisaldab ajatemplit, hinnet ja punktigruppe. Tudengi anonüümsus tagatakse räsides tudengi kaustanimed SHA-256 algoritmiga, eemaldades kõik isikut tuvastatavad andmed.

Töötluse tulemuseks saadakse kaks Pandase teegi DataFrame objekti ehk andmeraami: *submission_df*, kuhu koondatakse kõigi esituste metaandmed, ning *point_group_df*, mis sisaldab punktigrupi tasemel infot. Mõlemad andmeraamid laaditakse failis *data_store.py* asuvasse mälupõhisesse andmehoidlasse, et kõik API-marsruudid ja graafikufunktsioonid neile ligi pääseks. Selline lahendus välistab kordusparsimise igal päringul, vähendades serverikoormust.

2.3.2 Analüüsikiht

Analüüsikiht realiseerib mõlema eelkirjeldatud andmeraami põhised mõõdikud failis analysis.py. Näiteks funktsioon *general_data()* arvutab kokku tudengite arvu, esituste hulga ja parima hinde, *submission_count_timeline()* loob graafiku, mis näitab esituste kumuleerumist esitamisaja vältel, ring- ja kastdiagrammid võimaldavad uurida nii punktigruppe kui ka hinde paranemist esituste arvu lõikes. Kõik need funktsioonid tagastavad Altairi Chart[13] alaobjekti, mis serialiseeritakse Vega-Lite JSON vormingusse.

2.3.3 Visualiseerimismoodul

Flaski veebiserver avab iga graafiku jaoks eraldi API otspunkti, kasutades MethodView-põhist klassihierarhiat failis *chart_api.py*. MethodView on Flaski klassipõhine vaadete süsteem, mis võimaldab määratleda erinevad HTTP-meetodid (GET, POST) eraldi meetoditena klassi sees, muutes koodi struktuurseks ja paremini hallatavaks. Marsruudi */api/<graafiku_nimi>* kujul

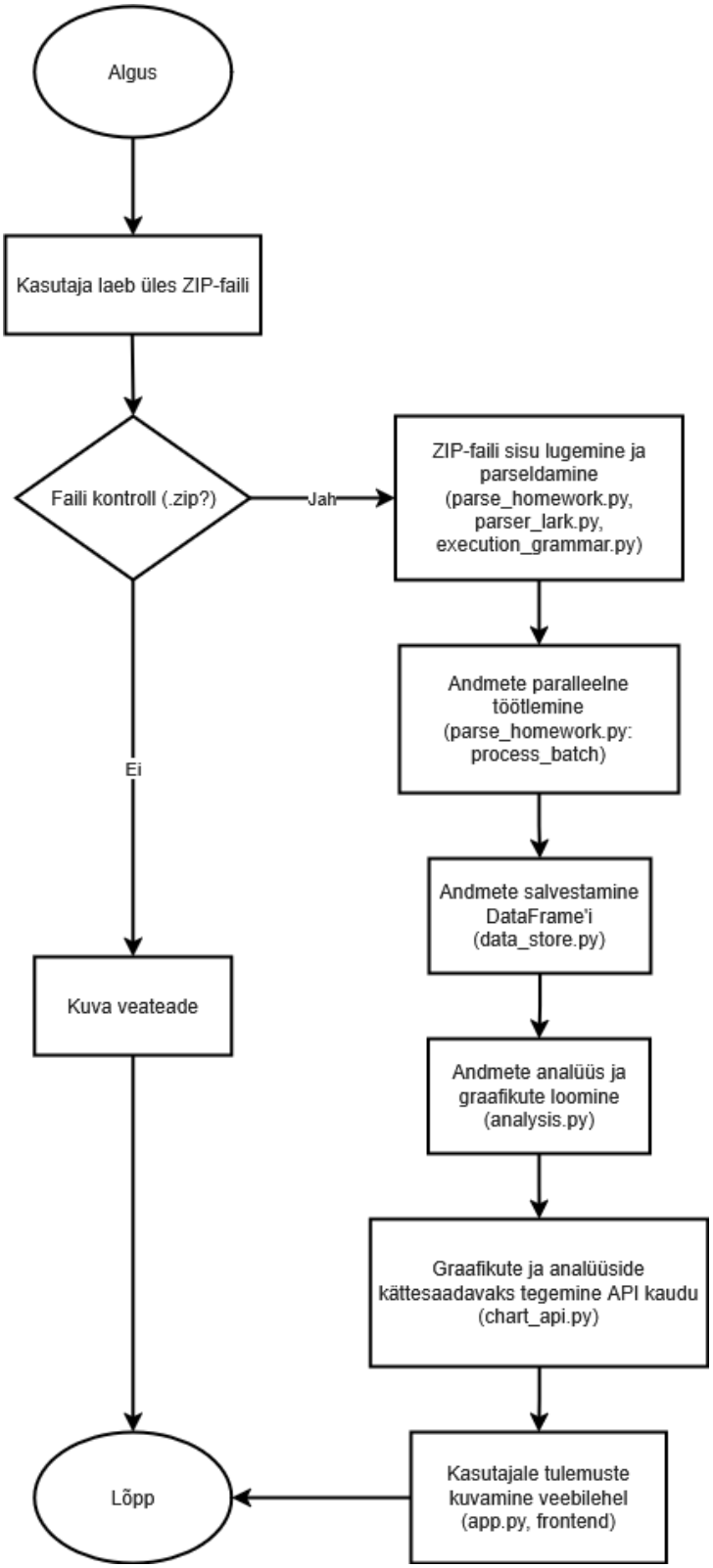
[11] https://lark-parser.readthedocs.io/en/latest/parsers.html#lalr-1
[12] https://lark-parser.readthedocs.io/en/latest/visitors.html#lark.visitors.Transformer
[13] https://altair-viz.github.io/user_guide/generated/toplevel/altair.Chart.html

saadetud GET-päringule vastatakse vastava Altairi funktsiooni tulemi serialiseeritud kujuga. Selline disain võimaldab uut diagrammi lisada üksnes Altairi funktsiooni ja ühe marsruudi-registratsiooni juurde kirjutamisega, ilma et ülejäänud arhitektuuri oleks vaja muuta.

Kasutajaliides suunab õppejõu pärast failitöötluse lõpuleviimist lehele *dashboard.html*, kus JavaScripti fail *chart_endpoints.js* saadab iga diagrammi konteineri kohta API-päringu. Iga eduka päringu vastuse puhul tagastatud Vega-Lite JSON joonistatakse brauseris VegaEmbed skripti abil, mis pakub interaktiivsust, nagu legendi filtreerimine, andmepunktide detailvaade ja suurendamine. Paigutuse ja värviskeemi ühtlustamiseks kasutatakse Bootstrap 5 raamistikku ja kohandatud CSS-klassi.

Kõnealune arhitektuur võimaldab serveril keskenduda andmetöötlusele, jättes diagrammide joonistamise brauserile; tulemusena kohandub süsteem koormuse kasvuga paremini ja kasutaja kogeb väiksemat latentsust.

Kokkuvõttes moodustavad teadlikult eraldatud kihid – parsimistorustik, analüüsikiht ja visualiseerimismoodul – struktureeritud arhitektuuri, mille võib vajaduse korral laiendada reaalajas andmetöötluseks või masinõppe-põhiseks hindeprognoosiks, muutmata seejuures senist komponente siduvat Flask-liidest.



Joonis 4. Rakenduse töövoo diagramm.

2.4 Realiseerimise detailid

Selles alapeatükis käsitletakse süsteemi praktilist teostust ja tehnilisi valikuid. Kirjeldatakse, kuidas on korraldatud veebiliidese ja serveri vaheline suhtlus, millised REST-stiilis API-otspunktid on kasutusel ning kuidas nende kaudu andmeid edastatakse. Tutvustatakse ka parsimissüsteemi täiustamist, rõhutades formaalse grammatika kasutuselevõttu ja paralleel-töötluse rakendamist, mis võimaldas tõsta töötlustoru töökindlust ja jõudlust. Lisaks antakse ülevaade andmete ettevalmistamisest visualiseerimiseks, et tagada sujuv ja tõhus graafikute kuvamine brauseris.

2.4.1 API päringud ja nende selgitused

Järgnevas tabelis on toodud rakenduse peamised REST-põhised API-otspunktid, mille kaudu veebiliides suhtleb tagarakendusega. Päringud jagunevad kahte tüüpi:

- GET – kasutatakse üksnes andmete lugemiseks, mis ei muuda serveri olekut;
- POST – kasutatakse failide üleslaadimiseks ja serverisiseste toimingute käivitamiseks, mis muudavad olekuid (näiteks kodutöö üleslaadimiseks ja parsimisprotsessi käivitamiseks või andmete puhastamiseks).

API punkt	**Punkti selgitus**	**Päringu tüübid**	**Õnnestunud päring**	**Ebaõnnestunud päring**
/upload	*GET* – tagastab ZIP-faili üleslaadimise vormi. *POST* – võtab kasutajalt ZIP-arhiivi & käivitab parsimise.	GET, POST	**GET 200** – kuvatakse üleslaadimisleht. **POST 200** – fail töödeldakse & kasutaja suunatakse dashboard.html lehele.	**POST 400** – „Please upload a valid zip file“ (vigane arhiiv). **POST 400** – „Error processing zip file: <veateade>“ (parsimisviga).
/api/general	Tagastab kodutöö üldised mõõdikud (tudengite arv, esituste hulk, parim hinne jne).	GET	**200** – JSON-objekt; brauser joonistab üldtabeli.	**404** – „Failed to fetch general data.“
/api/date_range	Annab andmestiku minimaalse ja maksimaalse ajatempliga kuupäeva – kasutatakse filtris.	GET	**200** – JSON {start, end}.	**404** – „No date range available.“
/api/hw_name	Tagastab aktiivse kodutöö nime	GET	**200** – kodutöö nimi tekstina.	**404** – „Homework name not found.“

	pealkirjades kasutamiseks.			
/api/purge	Kustutab üleslaaditud ZIP-failid ning lähtestab rakenduse mäluseisu.	POST	**200** - /uploads puhastatud ja andmed nullitud.	**500** – „Failed to clear uploads.“ (konkreetne veateade logis).

Tabel 1. Põhiliste API punktide selgitused.

Analoogselt on tehtud ka iga graafiku jaoks oma reegel, mis alati kasutab **GET** meetodit ja tagastab graafiku või veakoodi. Joonisel 5 on viimase hinde ajavahemike sektordiagrammi näitel rakendusliidese reegel.

```
app.add_url_rule(
  "/api/last_grade_time_pie_chart",
  view_func=LastGradeTimeRangePieChartApi.as_view("last_grade_time_pie_chart")
)
```

Joonis 5. Hinde ajavahemike sektordiagrammi graafiku API reegel.

2.4.2 Parsimissüsteemi täiustamine

Automaatkontrolli *execution.txt* logide varasem töötlustoru tugines ainult regulaaravaldistele ja järjestikusele failide läbivaatamisele. Kuigi selline lähenemine võimaldas kiiresti eraldada ajatemplid, hinded ja punktigrupid, osutus see hapraks. Väiksemgi muudatus faili vormingus tekitas parsimisvigu, mida oli keerukas parandada, koodi oli keeruline hooldada ja kogu ZIP-arhiiv töödeldi vaid ühe lõimega, mistõttu suuremate ZIP-failide käitlemine oli aeganõudev.

Sisendfaili mahu kasv ja struktuuri muutlikkus tekitasid vajaduse töökindlama ja paindlikuma lahenduse järele. Uues arhitektuuris defineeriti logivorming formaalse Lark grammatikaga, mis kirjeldab täpselt kõiki *execution.txt* struktuurielemente. Grammatikast genereeritud LALR(1) parser tagab, et muudatused logiväljundis nõuavad üksnes reeglite ajakohastamist, mitte kogu parsimiskoodi ümber kirjutamist. Larki Transformer liides teisendab parsimispuu kujule, millest kaks põhilist Pandase andmeraami luua.

Parandamaks andmetoru läbilaskevõimet, rakendati partii-põhist paralleeltöötlust. ZIP-arhiivist eraldatakse tudengikaustad vaikesättega kuni 70-elemendilisteks plokkideks, mida töötlevad eraldi protsessijad Pythoni sisseehitatud moodulist concurrent.futures. Nõnda jaotub

koormus mitmele tuumale, välditakse üksikprotsessi pudelikaela ja vähendatakse suurte arhiivide töödeldavust mitmekordselt.

Tulemuseks on süsteem, mille vigade taluvus on oluliselt kõrgem ja hooldatavus märgatavalt parem kui ainuüksi regulaaravaldisi kasutavas lahenduses. Uuenduse tagajärjel süsteem kohandub proportsionaalselt masina protsessorituumade arvuga ning loob stabiilse aluse edasiseks laiendamiseks.

2.4.3 Andmete visualiseerimiseks ette valmistamine

Enne kui Vega-Altair teek saab graafikud brauseris joonistada, läbivad parsimistorust pärit andmed ühtse ettevalmistuskihi failis *analysis.py*. Kõik töötlused algavad kahe mälus hoitava Pandase andmeraami laadimisega – *submission_df* (esituste metaandmed) ja *point_group_df* (punktigrupi taseme tulemused). Andmehoidlast *data_store* funktsiooniga *get* saadud raamid filtreeritakse kuupäeva- ja muu kasutajapõhise valiku alusel abifunktsiooniga *filter_dataframe()*, vältides tarbetuid ümberarvutusi iga API-päringu korral. Kõik funktsioonid tagastavad otse Altair Chart objekti või selle kihistatud variandi. Selline korraldus tähendab, et brauserisse jõuavad Vega-Lite JSON vormingus kirjeldused ning tagarakendus ei joonista pilte ega tabeleid, vaid edastab ainult minimaalse informatsiooni.

2.4.4 Visualiseerimisteegi valik

Visualiseerimisteegi valik tugines eelnevalt määratletud funktsionaalsetele ja tehnilistele nõuetele, mille eesmärk oli tagada lahenduse sobivus loodud süsteemi arhitektuuriga ning toetada kasutusmugavat ja informatiivset andmeesitust.

Nõuete valguses võrreldi mitut levinud visualiseerimisteeki. Laialdases kasutuses olevad Matplotlib[14] ja sellel põhinev Seaborn[15] toetavad Pandase andmestruktuure, kuid keskenduvad peamiselt staatiliste graafikute loomisele ja eeldavad serveripoolset joonistamist, mistõttu nende kasutamine oleks vastuolus brauseripõhise graafikute joonistamise ja interaktiivsuse nõudega. Plotly[16] ja Bokeh[17] võimaldavad küll interaktiivseid visualiseeringuid, kuid nende rakendusliidesed on imperatiivse lähenemise ja nõuvavad rohkem käsitsi juhtimist ning integreerimine brauseripõhisesse rakendusse oleks raskendatud. Dash, mis toetub Plotly

[14] https://matplotlib.org/
[15] https://seaborn.pydata.org/
[16] https://plotly.com/python/
[17] https://bokeh.org/

teegile, on mõeldud pigem keerukamate serveripõhiste rakenduste loomiseks ja oleks seetõttu ülemäära mahukas valik.

Kõige paremini vastas seatud kriteeriumitele Altair. Altair toetab otseselt Pandase andmeraame, võimaldab oma deklaratiivse rakendusliidese kaudu lihtsasti luua Vega-Lite põhiseid graafikuspetsifikatsioone ning toetab graafikute edastamist JSON vormingus brauserisse, kus need kuvatakse VegaEmbed skripti abil. Seetõttu oli Altairi kasutamine selle rakenduse jaoks tehniliselt ja funktsionaalselt sobivaim lahendus.

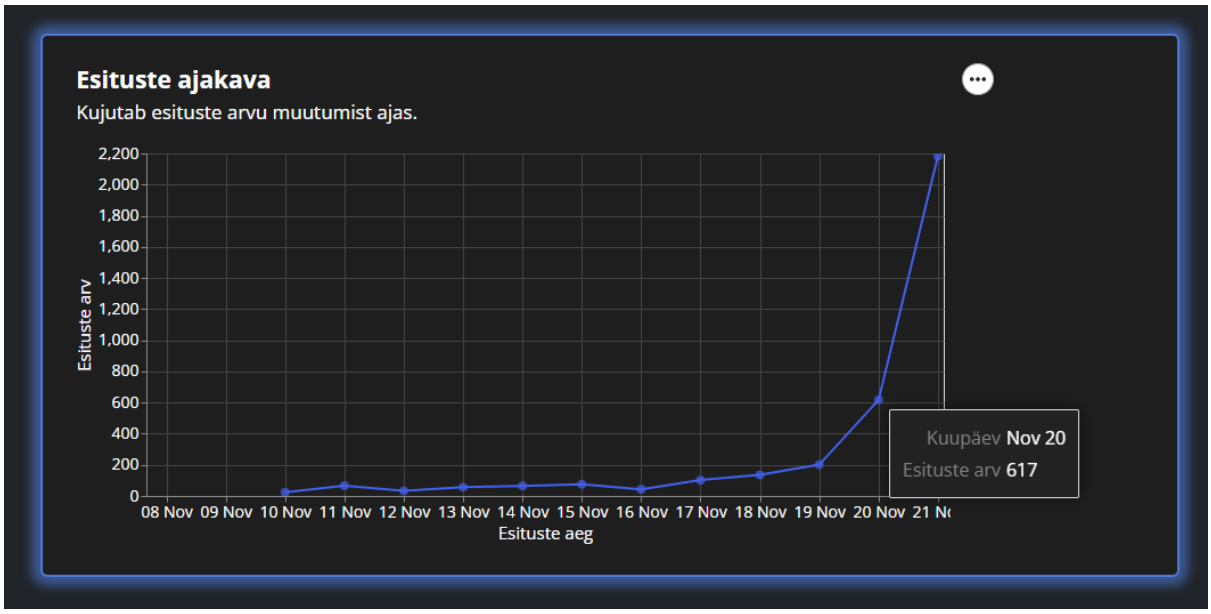
3. Tulemused ja analüüs

Peatükis hinnatakse valminud süsteemi väljundit ning näidatakse, kuidas interaktiivsed graafikud võiksid aidata õppejõul mõtestada automaatkontroll kirjetest pärit andmeid 2024. aasta kursuse Algoritmid ja andmestruktuurid viienda kodutöö näitel. Pakutakse vaatenurki esituskäitumisele ja hinnete kujunemisele. Peatüki lõpus tehakse ettepanekud võimalike edasiarenduste kohta, näiteks reaalajas andmeuuendused ning erinevate kodutööde võrdlemine.

3.1 Visualiseeringud ja kasutatavus

Peatükis on välja toodud mõned valminud visualiseeringud ning nende kirjeldused, mille abil õppejõud saab tuvastada kodutööga seotud trende ja iseärasusi, et õppetööd täiustada. Visualiseeringute andmestikuks on kasutatud 2024. aasta Algoritmide ja andmestruktuuride viienda kodutöö logifaile ajavahemikus 10/11/2024 – 21/11/2024 (esitusperiood).

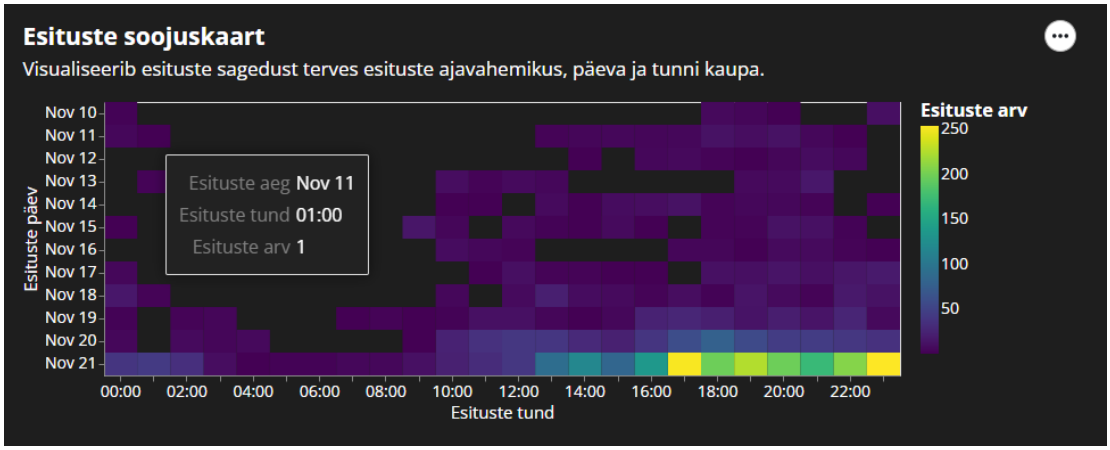
3.1.1 Esituste ajakava joondiagrammina



Joonis 6. Esituste aja joondiagramm.

Kujutatakse esituste arvu muutumist ajas. Abstsissteljel on esituste kuupäevad ja ordinaatteljel on esituste arv. Kui kasutaja liigutab hiire kursori punkti kohale, ilmuvad annotatsioonid, mis kuvavad konkreetse punkti andmeid – kuupäeva ja esituste arvu sellel hetkel. Graafikut saab sisse ja välja suumida, et tihedamaid piirkondi lähemalt uurida ning hiirega lohistades vasakule ja paremale nihutada. Diagramm annab ülevaatliku kokkuvõtte, millised esitustrendid konkreetse kodutöö puhul olid ning võimaldab tuvastada esitusserveri koormustipud.

3.1.2 Esituste soojuskaart



Joonis 7. esituste aja soojusdiagramm.

Ülevaatlik visualiseering esitussagedustest päeva ja kellaaja kaupa. Abstsissteljel on esituse täistund ning ordinaatteljel on esituste kalendripäevad. Kursori asetamisel konkreetse ruudu kohale kuvatakse annotatsioon, mis sisaldab esituse aega, esituse tundi ning esituste arv. Värvid on valitud esituste laialt varieeruva arvu illustreerimiseks. Graafik toob esile, millal automaatkontrolli server ressursse kõige enam vajab ning millal tudengid eelistavad töötada – antud juhul on tipptunnid õhtul vahetult enne tähtaega.

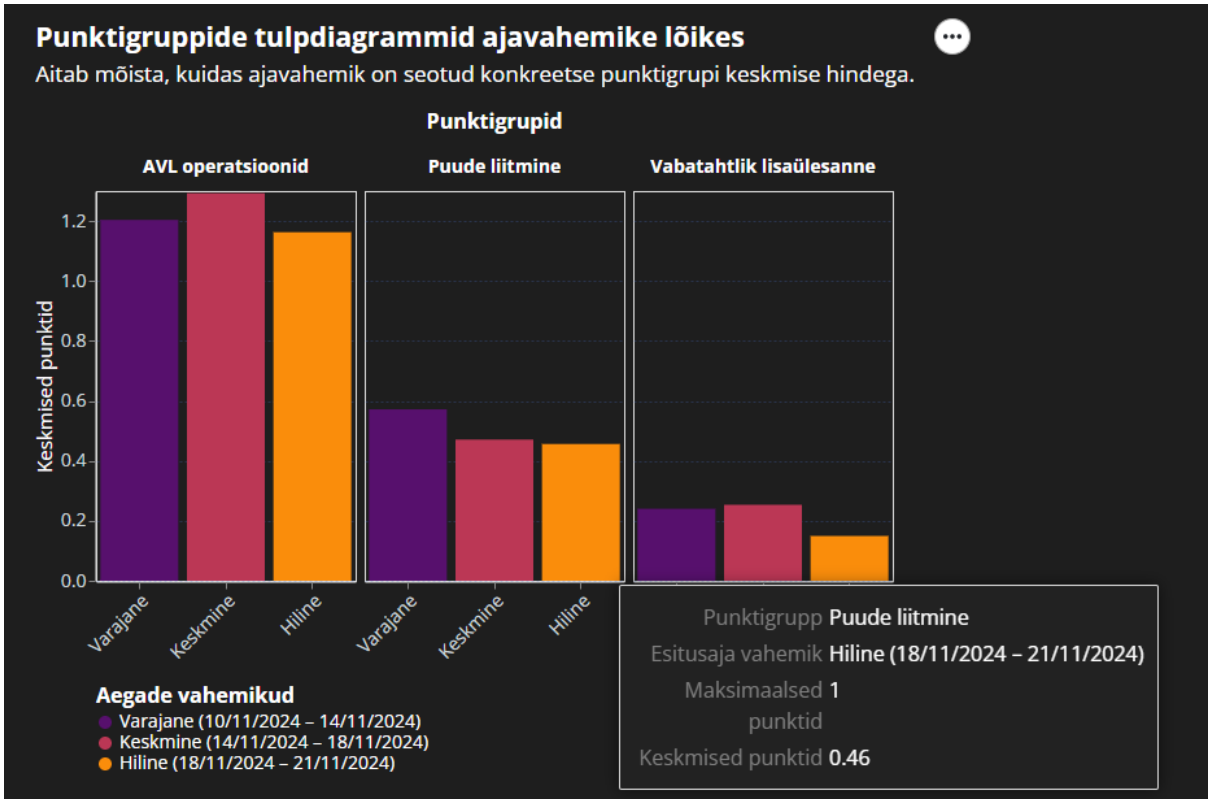
3.1.3 Viimaste esituste ajavahemike jaotus sektordiagrammina

Joonis 8. Esituste ajavahemike sektordiagramm.

Esitusperiood on jaotatud kolmeks võrdseks osaks: varajane, keskmine, hiline. Sektori raadius kujutab suhtelist tudengite arvu. Erinevalt eelmistest graafikutest näitab, millal tudengid oma

töö lõplikuks hindamiseks esitavad. Hiirega sektori kohal hõljudes kuvatakse konkreetset ajavahemikku ning selles sektoris töö esitanud tudengeid. Graafiku legend kuvab kõikide esitusaegade vahemikku koos vastava värviga.

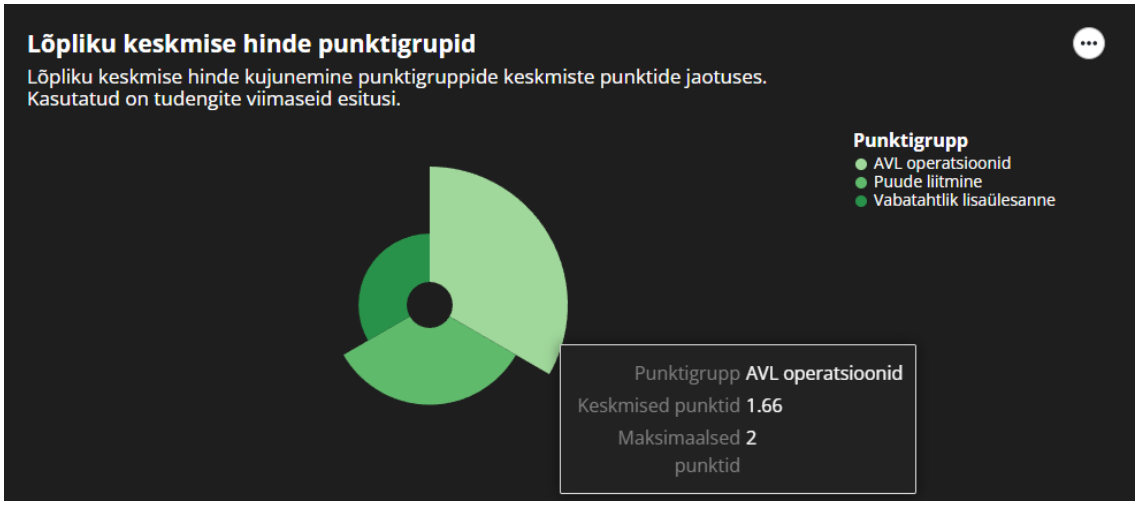
3.1.4 Punktigruppide keskmised punktid algusaja lõikes



Joonis 9. Punktigruppide keskmiste punktide tulpdiagramm

Esitusperiood on jaotatud kolmeks võrdseks osaks. Tulpdiagrammil on toodud iga punktigrupi keskmine vastavalt esitusperioodi ajavahemikule. Annotatsioonid näitavad punktigruppi, esitusaja vahemikku, maksimaalseid punkte ja keskmisi punkte. Võimaldab luua seoseid alustusaja ja punktigrupi vahel, mis muidu jääks üldise tulemuse taha varjatuks. Lisaks veel selgitab, kas mingisugune konkreetne hinnatav ülesanne eeldab varasemat alustamist.

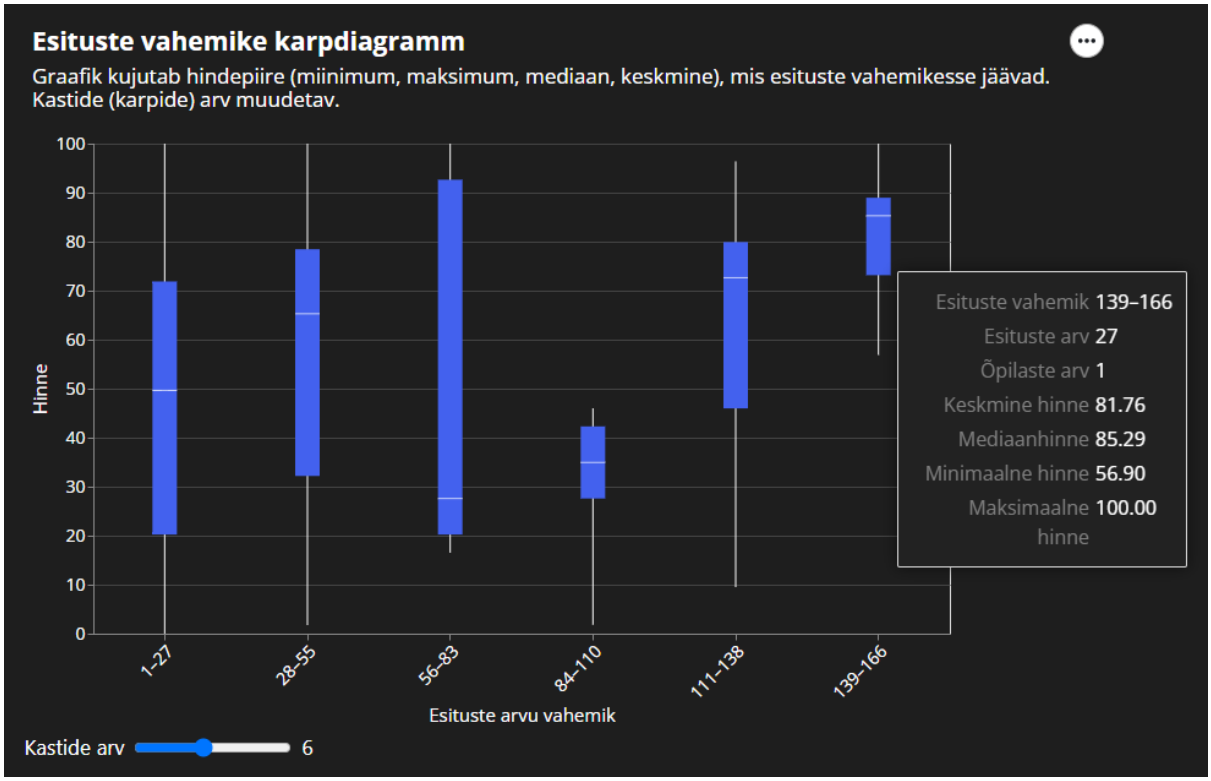
3.1.5 Lõpliku keskmise hinde moodustavate punktigruppide sektordiagramm



Joonis 10. Lõpliku keskmise hinde moodustavad punktigrupid.

Sektordiagramm moodustatakse vastavalt punktigruppidele töös. Iga sektor vastab ühele hinnatavale punktigrupile ning selle laius peegeldab grupi keskmist panust tudengite lõpliku hinde moodustumisele. Annotatsioonid kuvavad konkreetset punktigruppi, selle punktigrupi keskmisi punkte ning võrdluseks sellele ka maksimaalseid punkte. See aitab hinnata, milline punktigrupp panustab enim lõpphinde kujunemisse. Selle näitegraafiku puhul võib järeldada, et kõige suuremaks hindemoodustajaks on „AVL operatsioonid“ ülesanne.

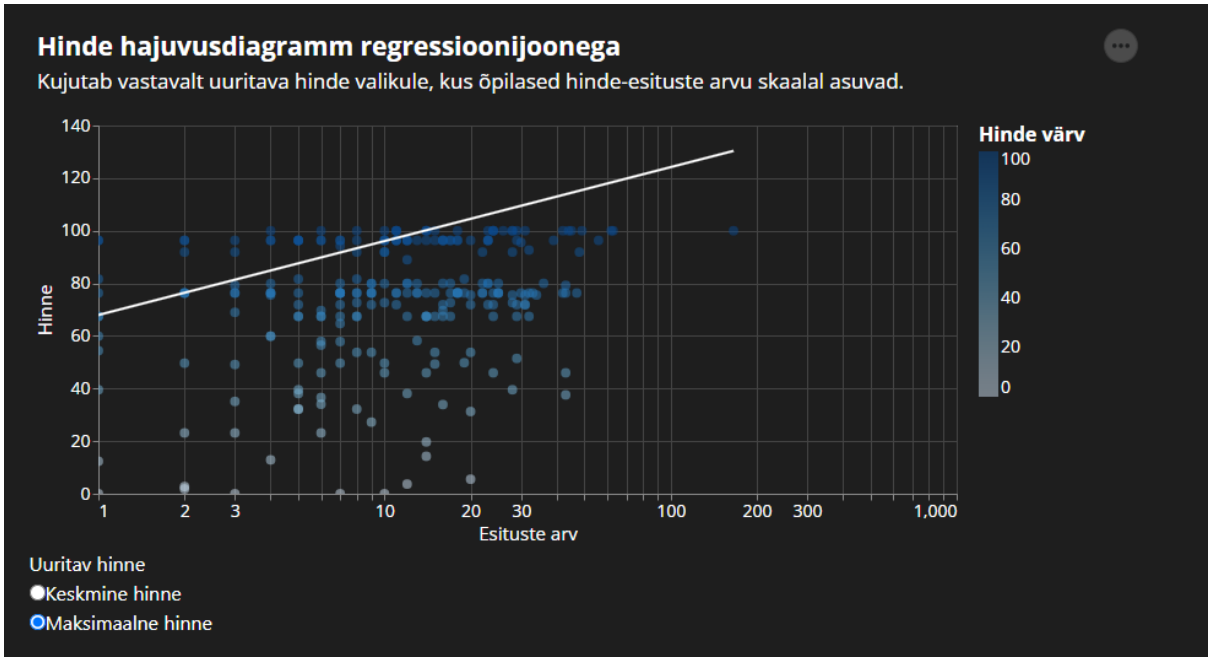
3.1.6 Esituste arvu vahemike karpdiagramm



Joonis 11. Esituste arvu vahemike karpdiagramm.

Diagrammi abstsissteljel on esituste arvude vahemikud, ordinaatteljel on esituste saavutatud hindeskaala. Iga esitus on omale külge saanud „järjekorranumbri“ ning on selle alusel kasti määratud. Kastide arv on liuguriga reguleeritav, vaikesäte 6. Iga „kast“ või „karp“ kujutab Y-teljel hinnete alam- ja ülemkvartiili, valge vertikaalne joon ulatub minimaalsest hindest maksimaalse hindeni selles grupis ning valge horisontaalne joon kujutab grupi mediaanhinnet. Kasti kohal hiirega hõljudes kuvatakse annotatsioonid, kus on kõik seda kasti puudutav info: esituste vahemik, esituste arv, tudengite arv, esituste keskmine hinne, mediaanhinne, minimaalne ja maksimaalne hinne. Selline diagramm aitab hinnata, milline esitamisstrateegia korreleerib kõrgemate ja stabiilsemate hinnetega ning kus riskipiirkonnad (lai hajuvus, madal mediaan) tegelikult paiknevad. Selle konkreetse kodutöö diagrammi puhul võiks arvata, et suur esituste arv korreleerub kõrge hindega. Asjaolu, et diagrammi kolmes viimases kastis on ainult üks tudeng, kes tegi rohkelt esitusi, ei luba siiski teha põhjapanevat järeldust. Kui valdav enamik esitab oma kodutöö näiteks vahemikus 1-55 esitust, siis tasub kaaluda, kui palju esitusi on mõistlik ühe kodutöö kohta lubada, pidades silmas serveriressursside kokkuhoidu.

3.1.7 Hinde hajuvusdiagramm



Joonis 12. Hinde ja esituste arvu vaheline hajuvusdiagramm.

Hajuvusdiagramm seob abstsisstelje esituste arvu ja ordinaattelje all valitava hindemõõdikuga – kas tudengi esituste keskmise või maksimaalse hinde. Punktide värv tumeneb koos hinde tulemuse kasvuga ning horisontaaltelg on loodud logaritmilise skaalaga, et ühes vaates mahutada nii paari esitusega tudengid kui ka kümnete esituste tegijad. Graafikule joonistub ka regressioonijoon, mis näitab hinde ja esituste arvu vahelist korrelatsiooni. Lisaks on graafik suumitav ja telgi saab liigutada, et mõnda punkti lähemalt uurida. Selle konkreetse kodutöö graafiku puhul on maksimaalne hinne ja esituste arv positiivselt korreleeritud, aga keskmiste hinnete puhul on joon negatiivne. See viitab võimalusele, et tudengid on esitanud madala kvaliteediga lahendusi ja kasutanud automaatkontrolli enda kasuks ära.

3.2 Võimalikud edasiarendused

Selles peatükis käsitletakse võimalikke edasiarendusi. Pakutud edasiarendused põhinevad vajadustel, mis ilmnesid lahenduse esialgsel kasutamisel ja testimisel. Lisaks tuuakse välja arendustega kaasnevad arhitektuurilisi muudatusi, tehnilisi eripärasid ja võimalikke ressursi-vajadusi.

3.2.1 Reaalajas andmeuuendus

Praegu töötab loodud süsteem staatiliste ZIP-arhiividega, mis tuleb käsitsi Moodle’ist alla laadida ja veebirakendusse üles laadida. Selline töövoog on sobiv väiksemahuliste analüüside jaoks, kuid piirab süsteemi kasutusmugavust ja paindlikkust suuremate kursuste või reaalajas monitooringu vajaduse korral.

Üheks võimalikuks edasiarenduseks oleks süsteemi täiustamine nii, et andmete parsimine ja visualiseerimine toimuksid automaatselt reaalajas, ilma manuaalse failide üleslaadimise vajaduseta. See tähendaks, et rakendus suudaks iseseisvalt ühenduda Moodle'i APIga või muude serveripõhiste andmeallikatega, laadida ja töödelda uusi logisid ajakava alusel (nt iga 5 või 10 minuti järel.)

Tehniliselt eeldaks see järgmisi muudatusi.

- Moodle’i andmepäringu API arendamine;
- ajastatud taustatöötluse lisamine Flaski rakendusele;
- vajadusel autentimis- ja sessioonihalduse mehhanismide juurutamine, et ligipääs Moodle’ile oleks turvaline;
- muudatused parsimistorus, mis võimaldaks andmete järkjärgulist täiendamist, mitte ainult täielikku ZIP-arhiivi korraga töötlemist.

Sellise täiustuse elluviimine võimaldaks õppejõul jälgida kodutööde esitamise ja hindamise protsessi reaalajas, aidates kiiresti tuvastada näiteks viimase hetke esituste kuhjumist, serveri koormustippe või võimalikku süsteemi tõrkeid.

Samas tuleb arvestada, et reaalajas andmetöötluse juurutamine suurendab süsteemi keerukust ja nõuab põhjalikumat veahaldust, näiteks katkestatud ühendused Moodle’iga või ebatäielikud andmed ning võib vajada ressursimahukamat serverikeskkonda.

3.2.2 Mitme kodutöö toetamine

Praegune süsteem eeldab, et korraga analüüsitakse ja visualiseeritakse ühe kodutöö andmeid. Kuigi see lähenemine on lihtne ja tõhus väikese töömahuga juhtumite jaoks, piirab see lahenduse ulatuslikumat kasutamist, eriti õppeainetes, kus semestri jooksul on mitmeid kodutöid või erinevaid kodutöö variante.

Mitme kodutöö toe lisamine võimaldaks:

- laadida korraga mitme kodutöö automaatkontrolli logifailid;

- eraldada andmed kodutööde kaupa, säilitades iga kodutöö jaoks oma andmeraamid ja visualiseeringud;
- võimaldada õppejõul valida kasutajaliideses, millise kodutöö andmeid parajasti analüüsida või võrrelda mitme kodutöö tulemusi omavahel.

Tehniliselt eeldaks see järgmisi muudatusi:

- parsimistoru ja andmestruktuuride ümberkujundamist selliselt, et toetada mitut andmestikku paralleelselt;
- andmete eristamine näiteks kodutöö ID või nime alusel;
- API ja veebiliidese kohandamine, et kasutaja saaks valida analüüsitava kodutöö, näiteks rippmenüü alusel;
- võrdlusgraafikute ja koondtabelite loomine, mis kombineeriks andmeid erinevatest kodutöödest.

Selline täiendus avardaks oluliselt süsteemi praktilist kasutust, võimaldades õppejõul analüüsida kursuse kulgu tervikuna, tuvastada näiteks, kas üliõpilaste esituskäitumine muutub aja jooksul või kas mõni kodutöö osutub järjest teistest raskemaks. Samas toob mitme kodutöö toetamine kaasa vajaduse hoolikama andmehalduse järele ning võib suurendada mälunõudlust serveripoolse töötlemise käigus.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli arendada terviklik lahendus Tartu Ülikooli aine Algoritmid ja andmestruktuurid automaatkontrolli logifailide töötlemiseks ja analüütiliseks visualiseerimiseks. Moodle'i hindetabelid annavad küll üldise hinnangu, kuid ei võimalda piisavalt detailselt analüüsida tudengite esituskäitumist ega hinnata ülesannete raskusastet. Automaatkontrolli logifailides leiduv lisainfo – ajatemplid, punktigrupid ja vahetulemused – võimaldab pakkuda õppetöö arendamiseks oluliselt sisukamat ja detailsemat tagasisidet.

Arendatud süsteem moodustab ühtse terviku, mis koosneb parsimistorust, analüüsikihist ja visualiseerimismoodulist ning on realiseeritud lõppkasutajale mõeldud veebirakendusena. Logifailide töötlemine toimub Larki formaalse grammatika abil, mis tagab suure töökindluse ja kohandatavuse logivormingu muutumisel. Pandase teegi toel viiakse läbi andmete töötlemine ja võtmemõõdikute arvutamine. Visualiseerimiseks kasutatakse Vega-Altairi teeki, mille abil luuakse interaktiivsed ja dünaamilised graafikud, mis kuvatakse kasutaja brauseris VegaEmbed skripti vahendusel. Kõik süsteemi kihid on ühendatud kergekaalulise Flask-põhise veebirakenduse kaudu, mis võimaldab õppejõul lihtsalt ja mugavalt andmeid üles laadida, töödelda ning tulemusi sirvida.

Töös loodud lahendus on hooldatav ja laiendatav. Tulevikus on võimalik süsteemi täiendada mitme kodutöö võrdleva analüüsiga ja reaalajas andmete uuendamise funktsionaalsusega. Töö tulemuseks on lokaalne, iseseisvalt töötav süsteem, mis muudab seni vähekasutatud logiandmed praktiliseks tööriistaks, toetades õppetöö läbipaistvust, hinnangute objektiivsust ja kursuse järjepidevat arendamist.

Viited

[1] J. Hollingsworth (1960). Automatic graders for programming classes. *Communications of the ACM* 3, 10, lk 528–529.

[2] Sidhidatri Nayak, Reshu Agarwal ja Sunil Kumar Khatri. (2022) Automated Assessment Tools for grading of programming Assignments: A review. International Conference on Computer Communication and Informatics, väljaanne jaanuar 25-27.

[3] M. Messer et al. (2024) Automated Grading and Feedback Tools for Programming Education: A Systematic Review. ACM Transactions on Computing Education. Köide 24, väljaanne 1, 19.02.2024, lk 1-43.

[4] P. Sharma et al. (2013) Parsing Techniques: A Review. International Journal of Advanced Research in Engineering and Applied Sciences. Köide 2, väljaanne 10, lk 65-76.

[5] G. Piatetsky (2016) R, Python Duel As Top Analytics, Data Science software – KDnuggets 2016 Software Poll Results. KDnuggets. 6.06.2016. https://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html.

[6] J. VanderPlas *et al*. (2018). Altair: Interactive Statistical Visualizations for Python. *Journal of Open Source Software*, 3(32), 1057. https://doi.org/10.21105/joss.01057.

Lisad

```
2024-11-05 19:26:37+02:00
Kodu4Test.java ab086aa2abf94907

<|-- -[ Korrektsus ] --|>
<|-- -Korrektus 1 ... OK --|>
<|-- -[ Juhuslikud ] --|>
<|-- -Juhuslik, n = 6 ... OK --|>
<|-- -[ Efektiivsus ] --|>
<|--
-Aeg 1 ... FAIL
Sisendmassiivi pikkus: 100000
Viga kohtadel 12599, 12600: 19912316927 > 20001033553
--|>
<|-- -[ EDETABEL ] --|>
<|-- Edetabelisse pääsemiseks pead edukalt läbima kõik testid! --|>
Point groups:
Korrektsus: 2.2857142857142856/3.0 (16.0/21.0)
Juhuslikud: 0.08333333333333333/3.0 (1.0/36.0)
Efektiivsus: 0.0/2.0 (0.0/3.0)
<|--
-Punktid:
-Korrektsus: 22.86 / 30
-Juhuslikud: 0.83 / 30
-Efektiivsus: 0 / 20
--|>

Grade :=>> 23.6905
Cleared leaderboard score
```

Lisa 1. Automaatkontrolli logide fiktiivne *execution.txt* fail.

```
execution_file_grammar = r"""
start: header (point_groups | time_section | grade_section | junkline)*

header: timestamp filename
timestamp: /\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}\+\d{2}:\d{2}/
filename: /[^\n]+/

point_groups: "Point groups:" (group_name ":" points_info)*

group_name: /[^\n:]+/

points_info: points_score "(" points_weighed ")"
points_score: points "/" points_max
points_weighed: weighed_points "/" weighed_max

points: NUMBER
points_max: NUMBER
weighed_points: NUMBER
weighed_max: NUMBER

!time_section: "Aeg:" ms_time "ms"
ms_time: NUMBER

grade_section: GRADE_HEADER grade_value
GRADE_HEADER: /Grade\s*:=>>/
grade_value: NUMBER

junkline: /.+/ -> skip_line

%import common.WS
%import common.NUMBER

%ignore WS
"""
```

Lisa 2. Automaatkontrolli logide *execution.txt* faili kirjeldav grammatika.

Projekti lähtekood, dokumentatsioon ja juhised käivitamiseks on kättesaadavad GitHubi hoidlas: https://github.com/rometv/grade-analysis.

Lisa 3. Programmi lähtekood.

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Romet Vinnal ,
(autori nimi)

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

   Automaatkontrolli logide parsimine ja interaktiivne visualiseerimine aine Algoritmid ja andmestruktuurid näitel ,
   (lõputöö pealkiri)

   mille juhendajad on Tõnis Hendrik Hlebnikov ja Ahti Põder ,
   (juhendajate nimed)

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada Tartu Ülikooli digitaalarhiivi kuni autoriõiguse kehtivuse lõppemiseni;

2. annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, 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.

Romet Vinnal

15.05.2025