TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Karolin Konrad

Paisktabelialgoritmide läbimängu automaatse hindaja loomine

Bakalaureusetöö (9 EAP)

Juhendajad: Ahti Põder,
Tõnis Hendrik Hlebnikov

Tartu 2023

Paisktabelialgoritmide läbimängu automaatse hindaja loomine

Lühikokkuvõte:

Tartu Ülikooli kursusel „Algoritmid ja andmestruktuurid“ õpetatakse klassikalisi andmestruktuure ja nendega seotud algoritme. Üks õpetamise viise on lasta tudengitel läbi teha algoritmide sammsammuline läbimäng. Et seda ei peaks tegema paberil oleks vaja keskkonda, kus tudeng saab läbimängu teha ja mis hindaks läbimängu korrektsust automaatselt. Töös valmis lahendus paisktabelialgoritmide läbimängu automaatseks hindamiseks. Selle osana valmis ka käsurealt käivitatava programmi prototüüp, kus kasutaja saab läbimänguülesannet lahendada.

Võtmesõnad:

Paisktabel, algoritmid

CERCS: P175 Informaatika, süsteemiteooria

Creating an automatic evaluator for the playthrough of hash table algorithms

Abstract:

One of the topics in the course “Algorithms and data structures” is classical data structures and algorithms related to them. One way of teaching this is to have students play through the algorithms step-by-step. To avoid having them do it on paper, an environment is needed where the playthroughs can be done and where each playthrough is automatically evaluated for correctness. In this thesis the solution for the automatic evaluation of the playthrough of hash table algorithms was created. As part of this, a command line interface was also created, where the user can solve the playthrough task.

Keywords:

Hash table, algorithms

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Kursusel „Algoritmid ja andmestruktuurid“ õpetatakse klassikalisi andmestruktuure ja nendega seotud algoritme. Üks õpetamise viise on lasta tudengitel läbi teha algoritmide sammsammuline läbimäng. Siiani tehakse seda paberil, millega kaasneb palju ebavajalikku lisatööd nii tudengile kui õppejõule. Neid raskusi vähendaks oluliselt sobiv keskkond, kus tudeng saab läbimängu teha ja mis hindaks läbimängu korrektsust automaatselt.

Õpetatavatest andmestruktuuridest oli eesmärgiks realiseerida paisktabelialgoritmide läbimängu süsteemi tagarakenduse osa ja käsureaprogrammist kasutajaliides tagarakenduse testimiseks. Ülesande sisendid genereeritakse Voitsehhovski loodud paisktabelialgoritmide sisendite genereerijaga [1], mis antakse süsteemile ette tekstifailis.

Esimeses peatükis on selgitus, miks on automaatne hindaja vajalik ning milliseid ülesandeid see hindab. Teises peatükis on kirjeldus töökäigu erinevatest etappidest ning lähem tutvustus sellest, mis oli oluline planeerimisel ja millised raskused ilmnesid realiseerimisel ning kuidas programmi testiti. Lisaks saab lähema tutvustuse ka realiseerimisel kasutatud programmeerimismustrite kohta ning põhjendusi nende valikus. Töö viimasest peatükist saab detailse ülevaate valminud lahendusest klasside kaupa ja leiab ettepanekuid, kuidas võiks süsteemis edasi arendada.

1. Probleemi analüüs

Sellest peatükis saab ülevaate probleemist, millele lahendus loodi. Lisaks segitusi, miks on vaja automaatset hindajat ja mis on paisktabelialgoritmid. Siin on ka hinnatavate ülesannete kirjeldused.

1.1 Automaatse hindaja vajalikkus

“Algoritmid ja andmestruktuurid” (LTAT.03.005) [2] on Tartu Ülikoolis informaatika õppekaval kohustuslik õppeaine. Seal tutvustatakse muuhulgas andmestruktuuride olemust, õpetatakse klassikalisi andmestruktuure ja nendega seotud algoritmide tööpõhimõtteid. Klassikalised andmestruktuurid on järjendid, paisktabelid, puud ja graafid. Lisaks algoritmide realiseerimisele arvutis peavad tudengid õppima algoritmide tööd, nii-öelda läbi mängima samm-sammult, nii nagu arvuti seda teeks. See aitab kaasa algoritmi sügavamale mõistmisele ning aitab hindamisel näha, kas tudeng on algoritmi töö põhimõttest ja nüanssidest aru saanud.

Siin töös on loodud hindaja paisktabelialgoritmide läbimängudele. Paisktabeli mõistet selgitab Jüri Kiho [3]. Paisktabel on massiiv, kus elemendid jaotuvad ühtlaselt massiivi kõikide ridade vahel. Igale elemendile on paisktabelis kindel koht, mis leitakse paiskfunktsiooniga. Tänu sellele on paisktabelist elemendi otsimine, lisamine ja eemaldamine konstantse keerukusega. Paisktabel võib olla lahtise adresseerimisega või välisahelatega.

Paisktabeliga seonduvate algoritmide õpetamise ja hindamise lihtsustamiseks on vaja automaatset hindajat ja keskkonda, kus nende läbimänge sooritada. Kuna selline keskkond ja hindaja puudub, siis tehakse algoritmide läbimänge paberil.

Läbimäng paberil nõuab tudengilt palju lisatööd, mis pole vajalik algoritmi mõistmiseks. Näiteks massiivialgoritmide puhul peab kogu massiivi elemendid järgmise algoritmi sammu tegemiseks uuesti maha kirjutama. Nii suure hulga lisatööga võivad kergesti tekkida hooletusvead. Paberil tehtud vigu on tudengil raske parandada ja nii läheb palju energiat hoopis muudele probleemidele, mitte algoritmi läbimängule.

Õpilaste paberil kujutatud läbimänge on ka õppejõududel raskem kontrollida. Lahendused võivad erineda käekirja ja vormistuse poolest ning segasematest töödest aru saamine pikendab oluliselt hindamisele kuluvat aega. Lisaks võib olla raske aru saada, kas tegu on näpuveaga ja tudeng tegelikult oskab teemat või ta päriselt ei saa aru. Automaatne hindaja annab

ühtse kujutusviisi, mis teeb läbimängu jälgimise lihtsamaks ja vähendab õppejõu ajakulu kontrollimisel, kus ta peaks läbimängu ise samm-sammult läbi tegema.

1.2 Töövahendite valikud

Automaatne hindaja on plaanis kasutusele võtta programmeerimisülesannete automaatkontrollile suunatud DeepMOOC veebikeskkonnas [4]. Integreerimise täpsed nõuded pole veel teada, kuna platvorm on veel arendamisel. Seepärast siin töös sellega ei tegeleta.

Hindaja arenduses oli valida kahe programmeerimiskeele vahel. Alates 2023. aastast toimub veebiplatvormi DeepMOOC tagarakenduse arendus keeles Kotlin [5]. Java ja Kotlin kompileeruvad mõlemad Java baitkoodiks [6]. Kasutades ühte neist keeltest lihtsustab see hilisemat hindaja integreerimist DeepMOOCi. Kuna Javaga olin juba tuttav, siis valisin programmi arendamiseks just Java, et mitte kulutada aega uue keele õppimiseks.

Töös kasutatakse Nikolai Voitsehhovski [1] loodud ülesannetele sisendi loomise programmi, millega saab genereerida valitud raskusastmega ülesande sisendeid. Need on sama või väga sarnase raskusastmega, kuid piisavalt erinevaid, et lahendust ei saaks kopeerida. Sisendeid saab genereerida ühe-või mitmekaupa, mida automaatne hindaja oskab tekstifailist lugeda.

1.3 Hinnatavad ülesanded

Kursuse materjalide [7] järgi on paisktabeli operatsioonid, mida kursusel õpetatakse, elemendi lisamine, elemendi eemaldamine ja elemendi otsimine. Neid rakendatakse kahel erineval paisktabeli tüübil, milleks on lahtise adresseerimisega paisktabel ja välisahelatega paisktabel. Lisaks paisktabeli operatsioonidele õpetatakse veel järjestamise kimbumeetodit ja järjestamise positsioonimeetodit. Need on sorteerimisalgoritmid, mis põhinevad välisahelatega paisktabelil.

Eraldi hindajat ei ole loodud elemendi otsimise ülesandele. Seda eraldi kontrollida pole vajadust, sest see on põhiline osa kõigis paisktabeli operatsioonides. Ilma selle mõistmiseta, on teisi operatsiooni võimatu õigesti teha. Hindajat vajavaid ülesande tüüpe on kokku neli, mida on täpsemalt kirjeldatud järgnevates alapeatükkides. Info ülesannete sisu kohta pärineb juhendajatelt ja kursuse materjalidest (vt [7]).

Lisamise ja eemaldamise ülesanded

Tavalises lisamise ülesandes saab tudeng ette paisktabeli, mis on kas tühi või mõnede elementidega, ja järjendi elementidega, mis tuleb ükshaaval tabelisse lisada. Ülesande koostaja poolt on määratud kompesamm ja paiskfunktsioon. Kompesamm on enamasti üks, kui tegu on lineaarse kompimisega, kuid on ka teisi kompimise viise, näiteks ruutkompimine. Neid viise automaatne hindaja ei pea kasutama, sest õpilastelt neid ei küsita ning realiseerimine oleks liigselt keerukas. Paiskfunktsioonina on ülesannetes kasutatud jääkpaiskamist, mis on räsi arvutamine elemendi jäägiga jagamisel paisktabeli pikkusega. See eeldab, et paisatavad elemendid on täisarvud. Teoorias võib paiskfunktsioon olla ükskõik milline. Oluline on paiskfunktsiooni puhul vaid see, et selle abil tabelisse paisatud elemendid jaotuksid tabelis ühtlaselt.

Eemaldamise ülesandes on tudengile antud paisktabel, eemaldatavad elemendid, kompesamm ja paiskfunktsioon. Tavapärane paiskfunktsioon on siin samuti jääkpaiskamine. Määratud elemendid tuleb etteantud järjestuses paisktabelist ükshaaval eemaldada ning järgnev kompejada uuesti tabelisse paisata. Seda võib teha kahel viisil. Esimene on võtta terve kompejada järjest üles ja panna siis järjest tagasi, see on igal juhul õige lähenemine. Teine viis on võtta elemente ükshaaval üles ja panna kohe tagasi. See on õige lineaarsel kompimisel, aga mitte ruutkompimisel. Läbimänguslaididel [7], mis on kursuse üks põhilisi õppematerjale, on kompejada läbimist esitatud just nii. Sellepärast eeldab automaatne hindaja kompejada läbimist ka samal viisil, et element paisatakse tagasi kohe peale üles võtmist. See on piisav kursusel küsitavate ülesannete korrektseks lahendamiseks.

Voitsehhovski programmist [1] genereeritav sisend on ühine lisamise ülesandele ja eemaldamise ülesandele. Programmi väljund on järjend elementidest, mis lisamise ülesande korral tuleb paisktabelisse täpselt selles järjekorras sisestada. Eemaldamise ülesandeks on eemaldada element, mis on lisatavate elementide jadas märgitud tärniga. Sisendi genereerimise programmiga saadud ülesanded on mõeldud tegemiseks lahtise adresseerimisega paisktabelil jääkpaiskamisega ja lineaarse kompimisega, kus kompesamm on alati üks. Midagi peale lisatavate elementide genereeritavas sisendis kirjas ei ole. Lisamise puhul on algne paisktabel alati tühi, kuid eemaldamiseks tuleb eelnevalt elemendid sisendist selles järjekorras paisktabelisse lisada. Ülesanne genereeritakse ülesande koostaja määratud raskusparameetri

järgi. Lisamise operatsiooni raskusparameetriks võttis Voitsehhovski lisamiste ajal tekkivate põrgete summa. Eemaldamise operatsiooni raskusparameetriks võeti kompejada läbimisel ümber paigutatavate elementide arv ehk nende elementide arv kompejadast, mis ümberpaiskamisel satuvad uuele kohale.

Kimbumeetodil sorteerimise ülesanded

Kimbumeetodil järjestamise ülesandes saab tudeng ette ainult arvude järjendi, mis tuleb järjestada. Põhiliselt kasutatakse meetodit positiivsete reaalarvude järjestamiseks. Esmalt peab leidma elementide arvu m ning vähima elemendi a ja suurima elemendi b sorteeritavast järjendist. Element b peab olema natuke suurem maksimaalsest väärtusest ja selle võib teatud piires ise valida. Nende põhjal tuleb esitada korrektne paiskfunktsioon, mis kimbumeetodis on

$$h(k) = \left\lfloor \frac{k-a}{b-a} * m \right\rfloor,$$

kus k on paisatav element. Loodud programmis tuleb tudengil sisestada a, b ja m, mille põhjal hindaja loob paisktabeli. Tudeng peab arvud tõstma sisendist ükshaaval paisktabelisse õigesse kimpu nii, et arvud igas kimbus oleks järjestatud. Päris implementatsioonis sorteeritakse kimbud mõne lineaarse keerukusega sorteerimisalgoritmiga, kuid läbimängus on piisav, kui arv sisestatakse kimbus õigele kohale. Kui see on tehtud, tuleb elemendid tõsta järjest paisktabelist tulemusjärjendisse.

Voitsehhovski programm [1] genereerib sisendiks vaid positiivsetest reaalarvudest järjendi, mis tuleb järjestada. Sisendi raskusparameeter on paiskamise etapi lõpuks tühjaks jäänud kimpude arv. Kuna paisktabeli pikkus on sisendi elementide arv, siis iga tühi kimp tähistab ühte arvu, mis paisati mõnda teise kimpu, kus juba oli vähemalt üks arv. Seega ajakulukas operatsioon on selles ülesandes paiskamine mittetühja kimpu.

Positsioonimeetodil sorteerimise ülesanded

Positsioonimeetodi algandmeteks on ainult sorteeritav järjend. Esmalt peab tudeng leidma, millise süsteemi elemendiga on tegemist ja looma paisktabeli vastavalt elementide hulga suurusele, mis antud süsteemis ühel positsioonil olla saab. Näiteks kümnendsüsteemi korral

oleks paisktabeli suurus 10, sest kümnendsüsteemi arvude positsioonid koosnevad numbritest 0 kuni 9. Ülesande lahendamisel paisatakse arve paisktabelisse ja sealt välja nii mitu korda kui on järjendi suurimas arvus positsioone. Paiskamist tehakse hetkel vaadataval positsioonil oleva elemendi järgi. Oluline on igal paiskamisel säilitada arvude järjestus, et pärast viimast tsükli läbimist oleks elemendid positsioonide kaupa järjestatud.

Voitsehhovski loodud programmiga genereeritavad sisendid on täisarvujärjendid. Positsioonimeetodil sorteerimise ülesandel on kaks raskusparameetrit: suurima elemendi positsioonide arv ja pikim ahela pikkus igal tsükli kordusel [1].

2. Töökäik

Töökäik koosnes kolmest etapist, milleks olid planeerimine, realiseerimine ja testimine. Järgmise etapi juurde liikumiseks pidi eelmine olema valmis, kuid ühes etapis ilmnenud probleemi korral tuli liikuda tagasi eelmisesse või isegi algusesse. Järgnevalt on kirjeldatud iga etappi põhjalikumalt.

2.1 Planeerimine

Planeerimisel tuli mõelda, kuidas programmi ehitada nii, et ülesande lahendamiseks jääks kasutajale piisavalt suur vabadus, kuid et oleks rohkem tuge kui tühjal paberilehel. Oluline on, et kasutajale võimalikes tegevustes ja nähtavas infos ei ole viiteid sellele, kuidas ülesannet lahendada. Tudengite tehtavate vigade kohta andsid täpsemat infot juhendajad. Näiteks selgus, et tudengid ei tee alati vahet ahelpaiskamisega paisktabelil ja lahtise adresseerimisega paisktabelil ning ei tea alati, mis on jääkpaiskamine.

Lisaks peab automaatne hindaja vähendama mahukat tööd, mis ei ole oluline algoritmist arusaamise näitamiseks. Näiteks, kui tudeng on viinud läbi muudatuse tabelis, mida algoritm sellel sammul teeb, siis ei tohiks temalt nõuda kogu ülejäänud massiivi käsitsi ümber kopeerimist nagu seda on vaja teha paberil. Tudeng peab saama oma vigu ka parandada, mida ta töö tegemise ajal märkab. Hindamisel pole nii oluline, et tudeng läbimängul ühtegi viga ei teeks, vaid algoritmi töö põhimõtetest arusaamine.

Läbimängu kujutamisel olid eeskujuks läbimänguslaidid, mis on kasutusel kursuse õppematerjalidena [7]. Seal on näha, milliste sammude näitamist läbimängus oodatakse. Samuti tuli analüüsida kõiki ülesande tüüpe ja leida neis ühiseid jooni. Selgus, et kõik ülesanded näevad välja piisavalt sarnased, et neid saab lahendada sama kasutamise süsteemiga. Kõigil on sisendiks massiiv, kus on elemendid, mida tuleb paisktabelisse lisada või on vaja kuhugi ajutiselt elemente tõsta.

2.2 Realiseerimine

Realiseerimise põhifookus oli hästi toimival tagarakendusel, mis oleks lihtsalt laiendatav ja kasutatav erinevat tüüpi kasutajaliidestega. Selle tarvis läks kasutusse kaks programmeerimismustrit: käsu programmeerimismuster ja strateegia programmeerimismuster. Programmeerimismustrite valikul kasutasin artikleid veebilehelt Refactoring Guru [8].

Kasutaja sammude edastamiseks hindajale on kasutatud käsu programmeerimismustrit (ingl. *command pattern*). Käsu programmeerimismustri artiklis [9] on selgitatud, et see muster võimaldab muuta kasutaja käsud programmis objektideks, mida saab anda meetoditele argumentideks ja koodis talletada. See teeb käskude tagasivõtmise implementeerimise väga lihtsaks, mis on loodava programmi juures oluline funktsionaalsus.

Ülesanded on implementeeritud strateegia programmeerimismustri põhimõttel (ingl. *strategy design pattern*). Artiklis [10] on kirjeldatud, et mustri peamine eesmärk on teha strateegia implementatsioon käitusajal väljavahetatavaks. See on hea probleemide puhul, kus on palju sarnaseid klasse, mis teevad ühte asja pisut erinevalt. Sarnane oleks šablooni meetod (ingl. *template method*), kus alamklassid katavad üle algoritmist muutuva osa. Kahe mustri peamine erinevus on toimimise tasemes, kus šablooni meetod on staatiline kuid strateegia meetod ei ole. Antud juhul on ülesannete vahelisi erinevusi piisavalt palju, et kasulikumaks osutus strateegia meetod, mis laseb tulevikus ka ülesannet töö ajal kergemini vahetada.

Raskusi valmistas sellise paisktabeli implementatsiooni loomine, mida saaks kasutada nii lahtise adresseerimisega kui välisahelatega paisktabeli ülesande lahendamisel, määramata kindlalt kummaga on tegemist. Lahtise adresseerimisega paisktabelis on igal real ainult üks arv, kuid ahelpaiskamisega paisktabelis võib ühel real olla mitu arvu. Lahendus on aga lihtne, kui kasutaja sisestab ainult konkreetsed indeksid, kuhu milline arv läheb. Sellisel juhul on paisktabel alati topelt järjend, mida käsitletakse lahtise adresseerimisega paisktabelina. Kui ühele reale lisatakse mitu elementi, siis on paisktabel ahelpaiskamisega.

Raskusi valmistas ka ülesannete kontrollpuude realiseerimine (vt joonised 2-5). Erinevalt lisamisülesandest pole eemaldamisülesandes võimalik läbimängu seisule peale vaadates hinnata, mis seisus algoritm parasjagu on ja mis oleks järgmine õige samm. Kuna sarnane probleem on ka kimbumeetodi ja positsioonimeetodi klassis, siis oli vaja ülesande klassis kuidagi järge pidada, mis staadiumis läbimäng on. Oli kaalumisel kuidas seda täpselt teha, kuid lõpuks töötas kõige paremini, kui hoida igas ülesande klassis selleks vastavat loendurit.

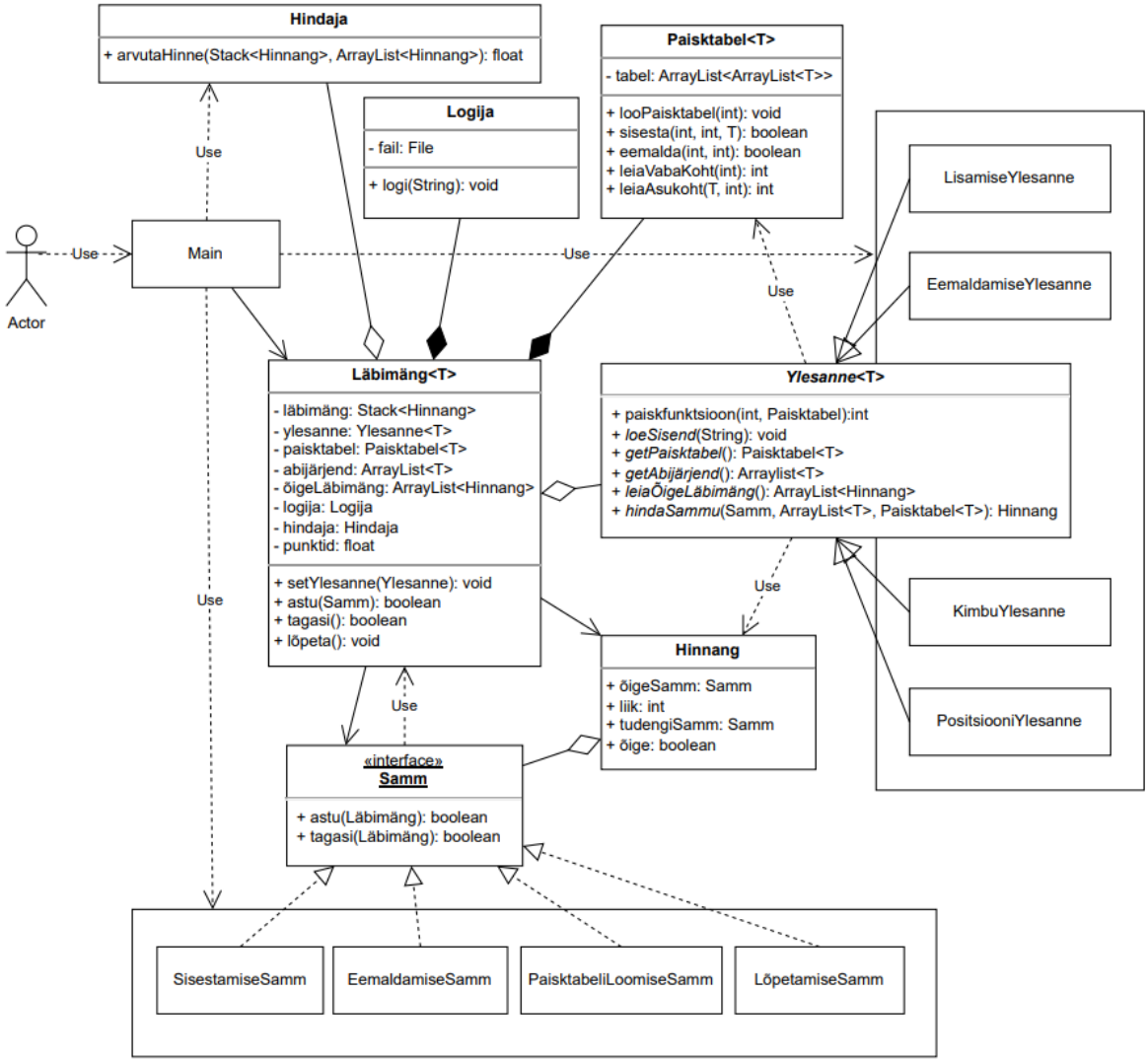
2.3 Testimine

Testimise eesmärk oli kontrollida oodatud funktsionaalsuste tööd kasutaja vaatenurgast musta kasti meetodil. Ülesande sisendid genereerisin Voitsehhovski programmiga kasutades programmi vaikeväärtuseid. Katsetatud teste on kirjeldatud järgmises lõigus.

Kõikide võimalike sammude läbi katsetamine aitas näha, kas operatsioonid töötavad õigesti. Kasutajale on saadaval sisestamine, eemaldamine, tagasivõtmine ja kahe ülesande alguses ka paisktabeli loomine. Vaja oli katsetada eraldi iga erineva käsu tagasivõtmist ja tagasivõtmist läbimängu alguses ja lõpus. See näitab, kas käsu programmeerimismuster on implementeeritud korrektselt. Iga ülesande tüübiga tuli teha läbi täiesti õige läbimängu, mõne tagasivõtmisega õige läbimängu, mõne veaga läbimängu ja ilma ühegi sammuta läbimängu. Kõigil juhtudel oli oluline programmi toimimise ootuspärasus.

3. Programmi kirjeldus

Valminud programmi kood koosneb seitsmest klassist ja kahest abstraktsest klassist, millel mõlemal on neli alamklassi. Programmi keskne klass on Läbimäng, mis on kontekstiks ülesande läbimängule. Seal kasutatakse abstraktse klassi Ylesanne defineeritud meetodeid käsurealiidesega Main klassist programmi testimiseks. Joonisel 1 on näha UML klasside diagramm, mis kujutab klasside vahelisi seoseid ja olulisemaid meetodeid ja välju. Järgnevalt on kirjeldatud iga klassi põhjalikumalt ja viimases alapeatükis on mõtteid edasiarendusteks.



Joonis 1. UML klasside diagramm.

3.1 Klass Paisktabel

Klass Paisktabel hoiustab läbimängus paisktabeli rolli täitvat andmestruktuuri ja meetodeid sellega opereerimiseks. Paisktabelit kujutav andmestruktuur, mida koodis ja tekstis edaspidi nimetatakse tabeliks, on järjendite järjend ArrayList<ArrayList<T>>. Tabeli sisemine andmetüüp on Java üldtüüp, et toetada paisktabeli klassi kasutamist erinevate andmetüüpidega ülesannetes. Kui lisamise ja eemaldamise ülesanded on eranditult täisarvudega, siis kimbumeetod on rakendatud just reaalarvudel ja positsioonimeetod võib olla rakendatud ka sõnedel. Järjendite järjend peab tabel olema selleks, et ülesande lahendaja saaks elemente paisktabelisse ka valesti lisada. Näiteks peaks õpilane saama lisada samasse ritta mitu elementi nagu tehakse välisahelatega paisktabelis isegi kui paisktabel peab olema lahtise adresseerimisega paisktabel.

Paisktabeli loomine käib kahes osas. Konstruktoris luuakse tabeli järjend ja meetodis *looPaisktabel* täidetakse tabel tühjade järjenditega kogu paisktabeli pikkuses. Tabelisse elementide lisamiseks ja sealt eemaldamiseks on klassis meetodid *sisesta* ja *eemalda*. Nendega saab lisada või siis vastavalt eemaldada elemendi tabelis täpselt etteantud kohas, rakendamata seejuures paisktabeli loogikat. Selle loogika peab läbi käima tudeng oma peas ja siis vastava sisestuse või eemalduse tegema otse tabelis õigel kohal. Lahtise adresseerimisega paisktabeli põrgete loogika jäljendamiseks on klassis meetodid *leiaVabaKoht* ja *leiaAsukoht*. Esimene leiab antud räsist alates vaba koha indeksi tabelist. Teine leiab elemendi asukoha tabelis. Need on vajalikud, et arvestada põrgetega. Kumbki meetod ei tee tabelis ühtegi muudatust.

Meetod *tagasi* tagastab *true* kui sammude massiiv on tühi või sai just tühjaks. Selle järgi saab kasutajaliides teha otsuse minna uuesti paisktabeli parameetrite küsimise juurde, kui ülesandel peaks seda vaja olema.

3.2 Klass Samm

Kasutajaliidesest liiguvad kasutaja käsud Läbimängu klassi käsu programmeerimismustri põhimõttel (ingl. *command pattern*). Kasutaja sisestatud käsust tehakse objekt, mida saab edasi anda ja salvestada. See muudab ka käsu tagasivõtmise implementeerimise lihtsaks. Olemasolevad käsud ehk läbimängu sammud on sisestamine, eemaldamine, paisktabeli loomine ja algoritmi lõpetamine.

SisestamiseSamm sisestab arvu abijärjendist vastavalt indeksilt paisktabelisse vastavale reale ja indeksile. EemaldamiseSamm töötab vastupidi ehk eemaldab paisktabelist vastavalt realt ja indeksilt arvu ja tõstab selle abijärjendisse vastavale indeksile. PaisktabeliLoomiseSamm initsialiseerib paisktabeli antud pikkusega. LõpetamiseSamm lõpetab algoritmi töö ja kutsub välja hindaja. Sisestus- ja eemaldussammu parameetrites pole elementi vaid indeksid, et mitte piirata elementide tüüpe, mida sammuga sisestada saab. Kasutajaliides ei saa teada millised andmetüübid kasutusel on. Loodud viisil võivad elemendid olla ükskõik mis tüüpi.

3.3 Klass Ylesanne

Ylesanne on abstraktne klass, kus on defineeritud ülesannetega seotud meetodid, mida Läbimängu klass kasutab. Iga alamklass implementeerib meetodeid erinevalt. Struktuur on tuntud kui strateegia muster (ingl. *strategy design pattern*).

Jääkpaiskamisega räsi arvutamiseks on klassis Ylesanne meetod *paiskfunktsioon*, sest see on enamikel ülesannetel sama. Vajadusel saab alamklassides meetodi üle katta nagu on tehtud kimbumeetodi ja positsioonimeetodi klassides. See meetod on ainus mitteabstraktne meetod Ylesande klassis. Kokku on kaheksa abstraktset meetodit, mis alamklassides peavad defineeritud olema. Meetod *loeSisend* loeb etteantud failist vastavalt ülesande sisendi ja salvestab saadud andmed klassimuutujatesse. Igal ülesandel võib failis olla erinevaid andmeid organiseeritud erinevas formaadis. Faili nimi antakse Main klassist ülesande initsialiseerimisel. Seal tuleb ka jälgida, et iga ülesanne saab sisendiks õige faili.

3.4 Klass Läbimäng

Klass Läbimäng on kontekst, mis koondab kogu loogikat. Selles hoitakse läbimängu jooksev paisktabel ja abijärjend, mille peal sammud teevad muudatusi *astu* ja *tagasi* meetodites.

Kasutajaliidese suhtlus käib läbi selle klassi. Seal määratakse ülesanne, kust kasutatakse meetodeid sammude kontrollimiseks ja ülesande alustamiseks. Meetodid *getPaisktabel* ja *getAbijärjend* tagastavad sisendist saadud andmete põhjal ülesandele vastavad paisktabeli ja abijärjendi algseisud. Sammu kontrollimine käib meetodiga *hindaSammu*, mille parameetriteks on samm, paisktabel ja abijärjend. Meetod tagastab klassi Hinnang isendi, mis

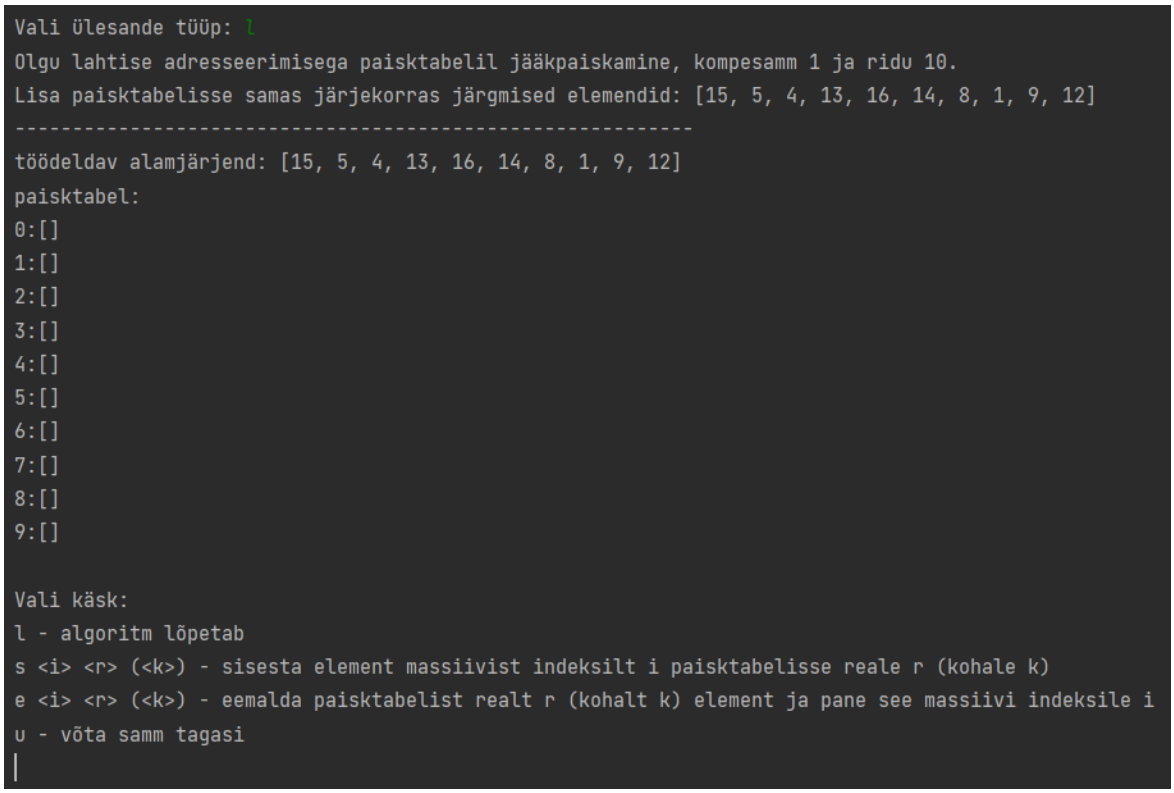
salvestatakse pärast hinde arvutamiseks. Õige samm leitakse paisktabeli ja abimassiivi hetkeseisu järgi. Igal ülesandel on selleks erinev implementatsioon.

Hinde arvutamine toimub Hindaja klassis, kus selleks on meetod *arvutaHinne*, mis tagastab tudengi läbimängu ja õige läbimängu põhjal saadud punktide protsendi. Töö osana valmis esialgne hindamisalgoritm, mis loeb iga õige sammu eest ühe punkti ja jagab selle õige läbimängu kogu sammude arvuga. See pole lõplik versioon, sest seda peavad otsustama õppejõud omavahel, kuidas täpselt hinne peaks kujunema. Hindaja algoritm on tõstetud eraldi klassi, et seda oleks lihtne muust programmist sõltumatult välja vahetada. Meetodi väljakutse on klassi Läbimäng *lõpeta* meetodis, kus tagastatud punktid logitakse ja salvestatakse Läbimängu klassis muutujasse, kust need on kättesaadavad programmi eluea lõpuni.

3.5 Klass Main

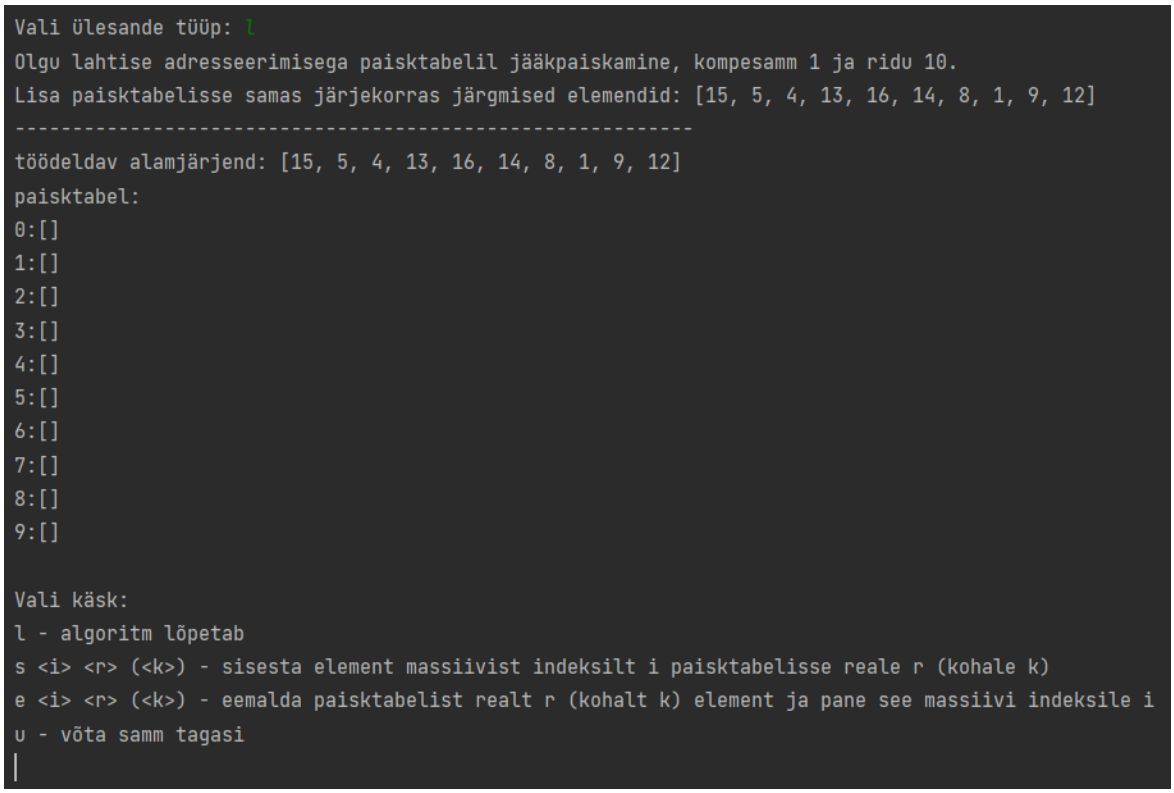
Klassis Main on kasutajaliidese käsurea prototüüp. Kasutaja tehtavad käsud antakse Läbimängule edasi klassi Samm isenditena. Klass Main on kergesti asendatav mõne graafilise liidesega, sest tegelikult ei ole vahet kuidas käsk kasutajalt saadakse. Main klassis luuakse Läbimängu isend millele antakse ette uus Hindaja ja uus Ylesanne.

Programmi käivitades on kasutajal võimalus valida nelja erineva ülesande vahel, mida lahendada, või programmist väljuda. Iga ülesande lõpetamisel kuvab programm hinde ja uuesti need valikud. Valikuid on näha joonisel 2. Saadaval on lisamis-, eemaldamisülesanne ning kimbu-ja positsioonimeetodil sorteerimisülesanded.

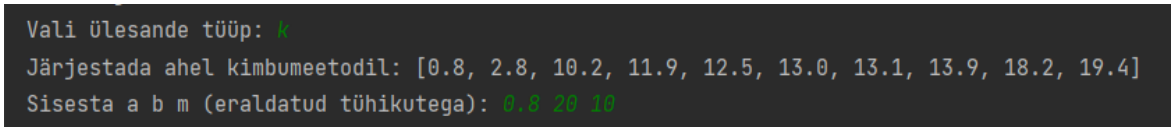


Joonis 2. Käsureal ülesande valimine.

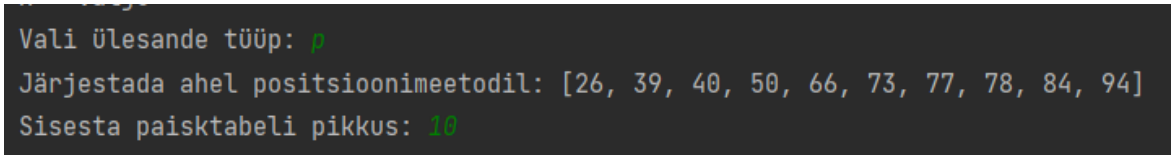
Ülesande valikul kuvab programm kasutajale valitud ülesande kirjelduse ning abijärjendi ja paisktabeli nende algseisudes ning jääb ootama kasutaja järgmist käsku. Sellest on näide joonisel 3, kus on lahendamiseks valitud lisamisülesanne. Kimbumeetodi ja positsioonimeetodi korral küsitakse lisaparameetreid nagu on näha joonisel 4 ja 5.



Joonis 3. Lisamisülesande algseis.

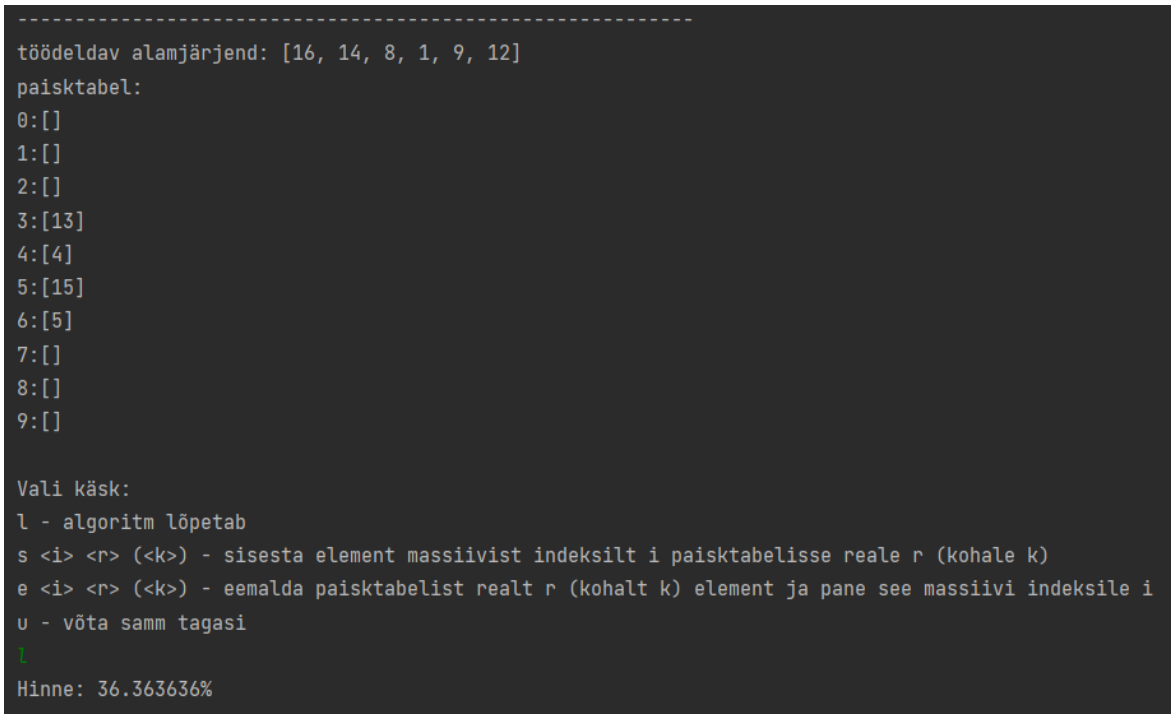


Joonis 4. Kimbumeetodil sorteerimise ülesandes valemi parameetrite küsimine.



Joonis 5. Positsioonimeetodil sorteerimise ülesandes paisktabeli pikkuse küsimine.

Ülesannete lahendamiseks on kasutajal alati neli käsku. Neid on näha joonisel 3 ja 6. Sisestamise ja eemaldamise käsud mõlemad võtavad argumentideks indeksi abijärjendist ja rea paisktabelist. Kolmas argument on indeks paisktabeli reas. See on vaikimisi null ja pole vajalik käsus täpsustada selleks, et käsurealt kasutamine oleks mugavam. Käsust luuakse klassi Samm isend, milleks on alati vaja kolme argumenti. Kui ülesande lahendaja on kindel et läbimäng on lõppenud, siis tuleb valida käsk “algoritm lõpetab” ning programm kuvab ülesande hinde. Näidet näeb joonisel 6.



Joonis 6. Lisamisülesanne on lõpetatud poole pealt.

3.6 Klass Logija

Läbimängus hoitakse Logija klassi isendit, mis loob initsialiseerimisel uue tekstifaili ja meetodiga *logi* lisab sinna ridu. Sellega logitakse detailne ülevaade läbimängust. Kirja läheb ülesande kirjeldus ja iga sammu kohta paisktabeli ja abimassiivi seisud, kasutaja tehtud samm, läbimängu olek ja hinnang kas samm oli õige. Juhul kui samm oli vale, on kaks lisa rida sõnaga “viga” ja õige sammuga. Läbimängu lõpus logitakse ka hinne protsentides.

Hindamise oluline osa on logid, kuhu peab näha jääma kogu läbimäng. Logitakse iga tehtud samm. Kui samm on vale, siis peaks selle logis märkima ja ka õige sammu ära näitama. Logisse peavad jääma näha ka tagasi võetud sammud, mille nägemine võib olla õppejõule vajalik hindamisel. Algoritmi antud hinne ei pruugi olla piisava kaaluga. Õppejõud või tudeng võib soovida lahenduse üle vaadata. Samm kontrollitakse ja logitakse kohe peale sammu tegemist, sest tagasi võetud sammud ei jää lõpptulemuses näha.

3.7 Klass Hinnang

Hindaja kasutab punktide arvutamiseks läbimängu ajal tudengi sammudest koostatud pinu ja läbimängu alguses arvutatud õigete sammude järjendit. Mõlemad andmestruktuurid koosnevad klassi Hinnang objektidest, mis on ühe sammu ja selle sammuga seotud andmete hoidmiseks. Klassis Hinnang on tudengi tehtud samm, läbimängu olek sammu tegemise hetkel, õige samm sellel hetkel ja tudengi sammu vastavus õigele sammule.

Maksimumpunktide leidmiseks on vaja algoritmiliselt leida kõik õige läbimängu sammud. Sellest tudengi sammu kontrollimiseks ei piisa, kuna õige läbimängu sammud ei arvesta paisktabeli ja abimassiivi tegelikku seisu, mis võib olla erinev. Viga ühel sammul võib muuta paisktabeli seisu nii, et paisktabeli seis on natuke erinev kui oleks pidanud ja sellest on järgnevad õiged sammud ka natuke erinevad. Sellepärast tahame õige sammu leida läbimängu ajal. Nii ei karistata tudengit varem tehtud vea eest korduvalt. Lisaks on sammule vaja kohe hinnangut, et selle saaks logisse märkida. Logisse on vaja lisada hinnang ka tagasi võetud sammude kohta, mida logitakse jooksvalt.

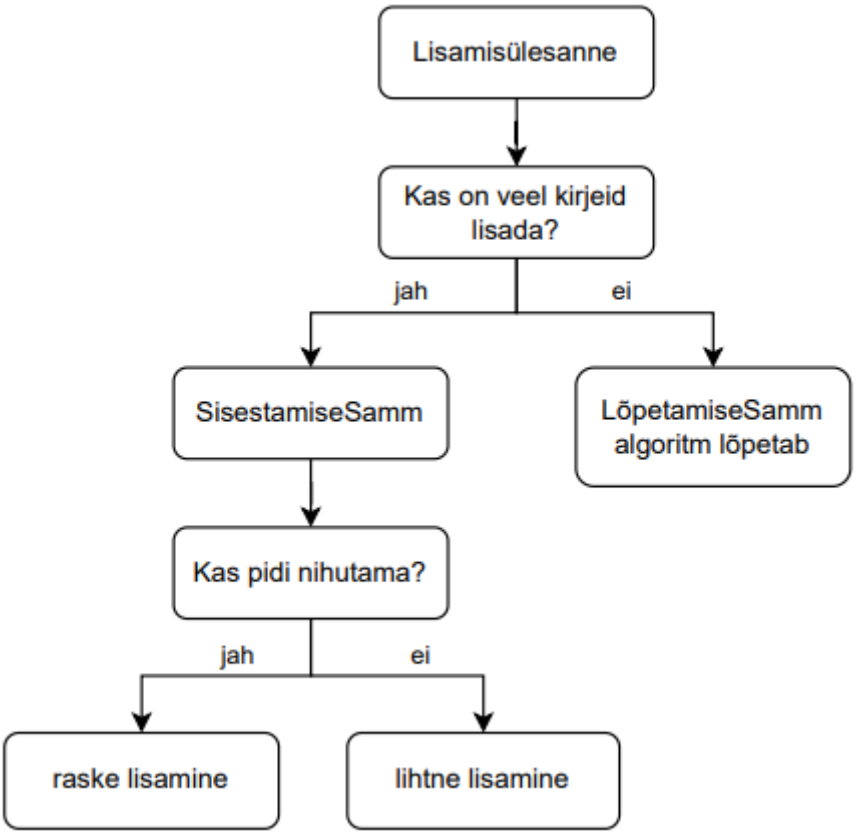
Hinnangu klassis on *õigeSamm* ja *tudengiSamm*, mis on mõlemad klassi Samm isendid. Väljal *õige* hoitakse mõlema sammu võrdsuse väärtust ehk teisisõnu tõeväärtust sellest kas tudengi samm on õige. Väljal *olek* on salvestatud läbimängu seis sammu tegemise hetkel. See on ühtlasi ka olek õige sammu leidmise hetkel. Kokku on kuus erinevat olekut:

- raske lisamine,
- lihtne lisamine,
- paisktabeli loomine,
- algoritm lõpetab,
- eemaldamine,
- kustutamine.

Olekute salvestamine hinnangu juurde on vajalik, et anda hindamisalgoritmile lisainformatsiooni sammu olulisuse kohta. Seda infot saab kasutada hindamisel lugedes kokku mitu sammu iga olekuga tehti ja mitu oli vaja teha. Olenevalt olekust on sammudel erinev kaal. Sama tüüpi samm võib olla erinevatel tingimustel erineva kaaluga. Siin töös on käsitletud kõiki olekuid sama kaaluga, kuna aine täpse hindamise otsustavad õppejõud.

3.8 Hinnangute leidmine

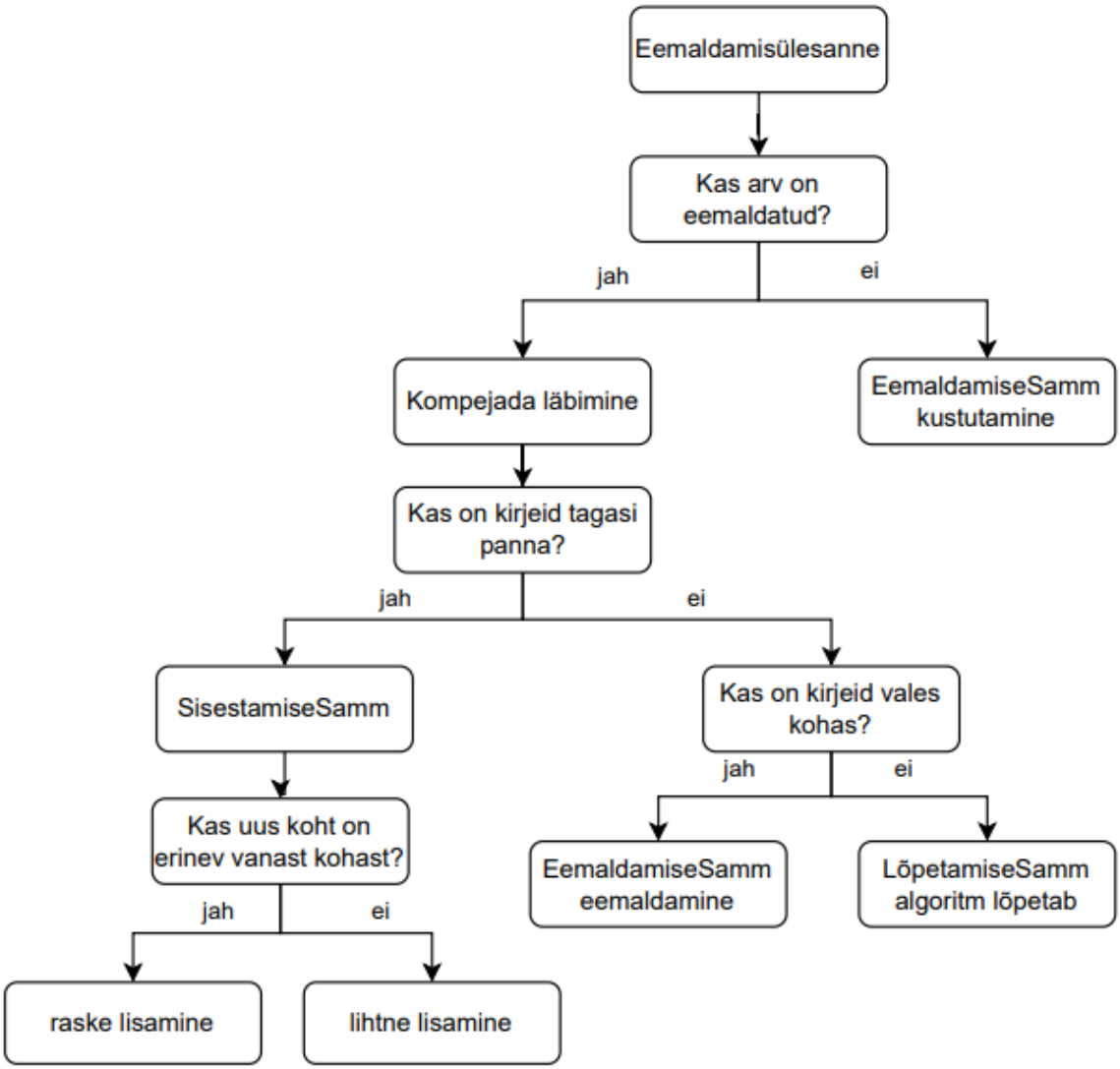
Klassi Ylesanne meetod *hindaSammu* leiab läbimängu oleku ja õige sammu ning tagastab need klassi Hinnang objektina. Lisamise ülesande sammu ja oleku leidmist kujutav skeem on joonisel 7. Kui on veel elemente, mida paisktabelisse lisada, siis on järgmine õige samm abijärjendi esimese elemendi lisamine tabelisse. Kui lisamisel toimub põrkeid, on tegu raske lisamisega. Ilma põrgeteta lisamise korral on tegu lihtsa lisamisega. Kui elemente enam tabelisse lisada pole, siis on algoritmi töö lõppenud ja õige samm on LõpetamiseSamm.



Joonis 7. Lisamisülesande kontrollgraaf.

Eemaldamisülesande õige sammu ja oleku leidmise skeemi näeb joonisel 8. Kui sisend on tühi, siis on järgmine õige samm ülesandes määratud elemendi kustutamine tabelist. Siis on õige samm Eemaldamise samm ja olek on kustutamine. Sealt edasi tuleb paisktabel korrastada õigesti. Selleks peab välja otsima, millised arvud on vaja ümber tõsta ja siis järge pidada nende tõstmistel. Kustutatud element pole tegelikult kustutatud ja hoitakse abijärjendis viimasel kohal. Selle järgi, et abijärjendis on vähemalt üks element, teab programm, et ülesandes nõutud arv on eemaldatud ja liigub edasi kompejada läbimise etappi. Selle järgi, kas abijärjendis on rohkem kui üks element teab programm kas järgmisena on vaja kompejadas

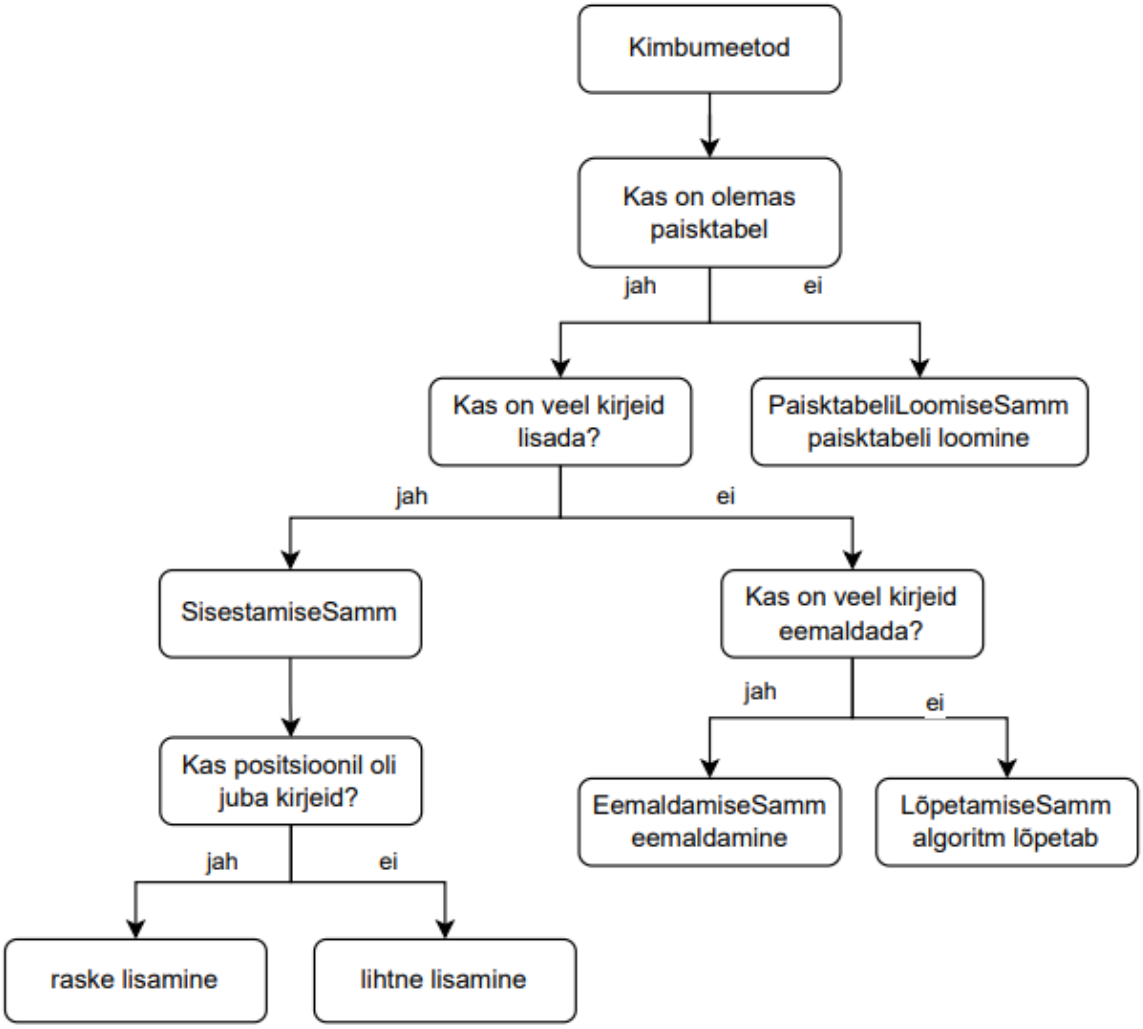
element üles võtta või juba üles võetud element tagasi paisata. Tagasi paiskamisel ehk lisamisel on raske lisamine juht kus arv paisatakse teisele kohale paisktabelis kui ta enne oli. Lihtne lisamine tähendab, et arv asukohta ei muutnud. EemaldamiseYlesande klassis on muutuja, mis hoiab sarnaselt paisktabeli tavalisele implementatsioonile järge indeksil, mille juures kompejada läbimine pooleli on. Kui see indeks jõuab tühja kohani või tagasi kompejada alguse indeksini, on järgmine õige samm LõpetamiseSamm.



Joonis 8. Eemaldamisülesande kontrollgraaf.

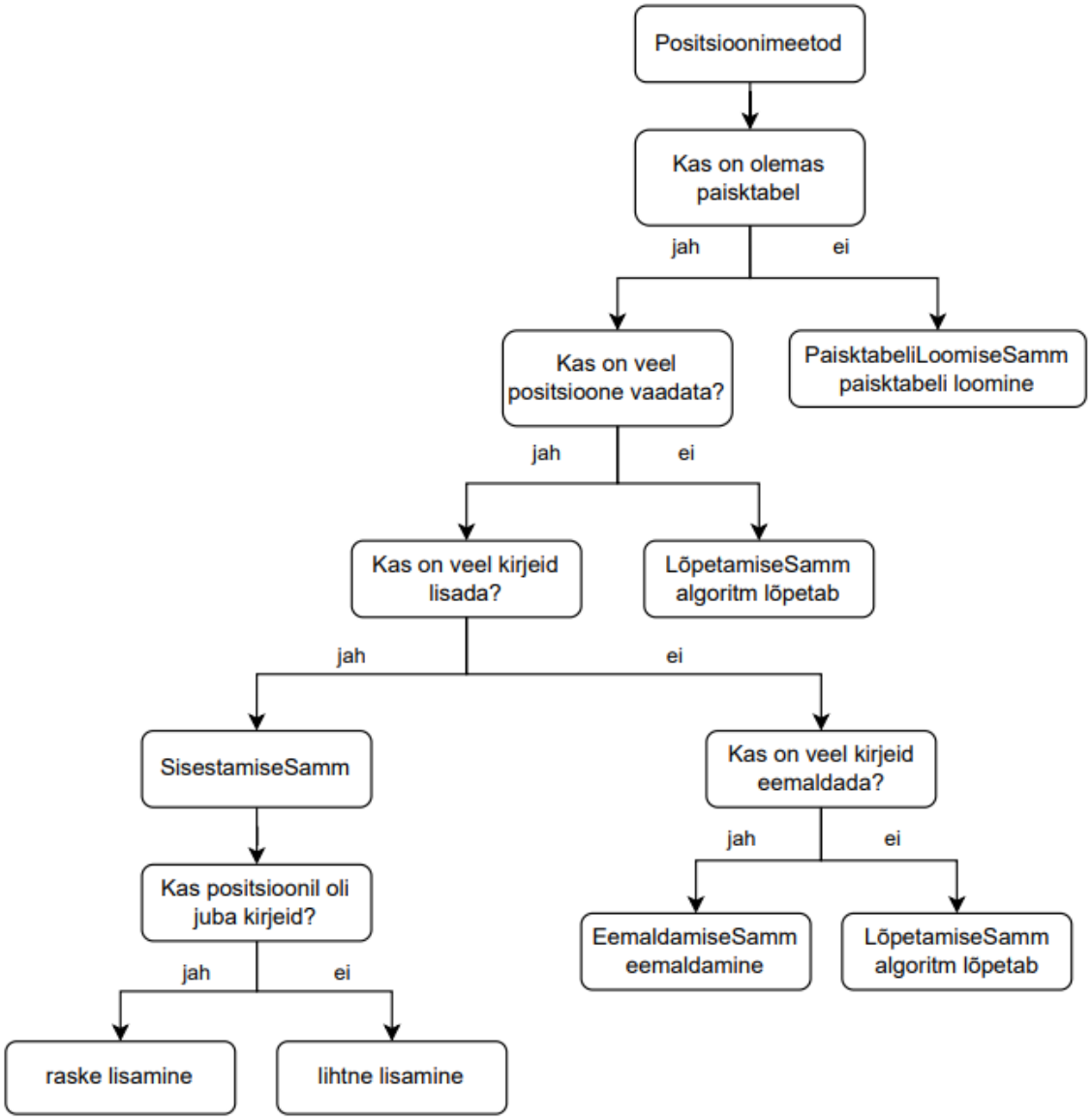
Järjestamise kimbumeetodi õige sammu ja oleku leidmist illustreerib joonis 9. Esmalt tuleb ülesande lahendajal luua paisktabel. Selleks küsitakse paisktabeli parameetreid. Kui paisktabel on olemas, minnakse elementide lisamise etappi, mis kehtib seni kuni abijärjend saab

tühjaks. Siin kontekstis on raske lisamine siis kui paisktabeli real juba oli eelnevalt kirje ja lihtne lisamine on lisamine tühjale reale. Kui abijärjend on korra tühjaks saanud, siis on järgmine etapp elementide eemaldamine tagasi abijärjendisse. Siin tuleb koodis meeles pidada, et nüüd on arvude eemaldamine. Muidu hakkab abijärjendi täitumisel jälle kehtima tingimus elementide lisamiseks. Kui paisktabelist on kõik arvud eemaldatud, on läbimäng jõudnud lõpetatud olekusse.



Joonis 9. Kimbumeetodil sorteerimisülesande kontrollgraaf.

Eelnevaga sarnane süsteem on ka positsioonimeetodil, kuid nüüd tuleb lisamise ja eemaldamise etapid teha läbi mitu korda ja pidada järge mitmendat korda tsüklit läbitakse. Positsioonimeetodi kontrollgraafi näeb joonisel 10.



Joonis 10. Positsioonimeetodil sorteerimisülesande kontrollgraaf.

3.9 Edasiarendused

Hetkel pole programm veel kõlblik hindamisel kasutamiseks. Peamine põhjus on sobiva hindamise valemi puudumine Hindaja klassis, kuid lisaks sellele on valminud programmil veel mitmeid puuduseid, mis vajaksid täiendamist.

Valminud käsurealiides on prototüüp kasutajaliidesele. Päris liides saab tehtud siis, kui see programm integreeritakse ka teisi läbimängualgoritme käitavasse keskkonda. Graafiline keskkond võiks olla selline, kus saab elemente abijärjendi ja paisktabeli vahel lohistada. See

muudaks kasutamist intuitiivsemaks ja samal ajal välistab vead, mida kasutaja saaks teha käske trükkides kogemata või meelega, lootes süsteemi üle kavaldada. Näiteks kaks tühikut järjest, mis muudab sisendi loetamatuks, või indeks, mis on massiivi/tabeli piiridest väljas.

Teine arenduskoht oleks laiendada programmi uute paisktabeliülesannete variatsioonidega. Valminud programmis on ülesandeklassid vaid Voitsehhovski programmiga loodavate sisendite lahendamiseks, kuid programm on võimeline hindama ka teistsuguste sisenditega ülesandeid. Näiteks on puudu lisamise ülesanne, kus ülesandes antud paisktabelisse on juba mõned arvud lisatud või kus saab anda ette teistsuguse kompesammu. Saaks ka teha eemaldamise ülesande, kus peab eemaldama mitu elementi mitte ainult ühe. Seda saab teha luues uue klassi Ylesanne alamklassi, mis oskab lugeda keerukamat sisendit.

Kokkuvõte

Töös valmis lahendus paisktabelialgoritmide läbimängu automaatseks hindamiseks. Selle osana valmis ka käsurealiides, kus kasutaja saab käske andes lahendada läbimänguülesannet. Süsteem salvestab käsud võimaldades ka nende tagasivõtmist. Salvestatud sammude põhjal arvutab algoritm hinde protsentides. Kogu läbimäng logitakse tekstifaili vajadusel üle vaatamiseks. Loodud lahendus automatiseerib kursusel "Algoritmid ja andmestruktuurid" õpetatavate paisktabelialgoritmide läbimängu hindamise.

Töökäik koosnes kolmest etapist: planeerimine, realiseerimine ja testimine. Disainis oli rõhuasetus piisavalt üldistatud toimival süsteemil, mis toetaks piisavalt erinevaid paisktabeli ülesandeid. Selleks kasutasin käsu ja strateegia programmeerimismustrit. Täielikult valmis tarkvarast on veel puudu graafiline kasutajaliides ja reaalne hinde arvutamise valem. Lisaks on vaja süsteem integreerida DeepMOOC veebiplatvormi, et seda saaks hindamiseks kasutada.

Viidatud kirjandus

[1] Voitsehhovski, Nikolai. 2022. „Input Generation for Hashtable Algorithms“, TÜ arvutiteaduse instituudi bakalaureusetöö. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=74478 (03.08.2023)

[2] Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) ülevaade. (24.04.2023). https://ois2.ut.ee/#/courses/LTAT.03.005/details (02.08.2023)

[3] Kiho, Jüri. 2003. “Algoritmid ja andmestruktuurid”. Tartu Ülikooli Kirjastus, lk 45

[4] DEEPMOOC platvormi arendamine. https://comserv.cs.ut.ee/ati_thesis_offers/datasheet.php?id=73312&year=2021 (07.04.2023)

[5] Tender, Märt. 2023. “DeepMOOC platvormile tagarakenduse arendamine”, TÜ arvutiteaduse instituudi bakalaureusetöö. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=77057 (03.08.2023)

[6] Kotlini lähtekoodihoidla. https://github.com/JetBrains/kotlin/blob/master/ReadMe.md (27.07.2023)

[7] Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) materjalid. (2023). https://moodle.ut.ee/course/view.php?id= 182 (03.08.2023)

[8] Refactoring and design Patterns. Refactoring.guru. https://refactoring.guru/ (04.08.2023)

[9] Command. Refactoring.guru. 2022. https://refactoring.guru/design-patterns/command (03.08.2023)

[10] Strategy. Refactoring.guru. 2022. https://refactoring.guru/design-patterns/strategy (03.08.2023)

Lisad

I. Lähtekood

Loodud hindaja lähtekoodi leiab failist „hindaja_lähtekood.zip“.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Karolin Konrad

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose Paisktabelialgoritmide läbimängu automaatse hindaja loomine, mille juhendajad on Ahti Põder ja Tõnis Hendrik Hlebnikov, 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.

Karolin Konrad

09.08.2023