TARTU ÜLIKOOL

Arvutiteaduse instituut

Informaatika õppekava

Markus Michelis

Kahendpuu- ja kuhjaalgoritmide läbimängija ja hindaja

Bakalaureusetöö (9 EAP)

Juhendajad: Ahti Põder,

Kristo Väljako

Tartu 2025

Kahendpuu- ja kuhjaalgoritmide läbimängija ja hindaja

Lühikokkuvõte:

Bakalaureusetöö eesmärk oli arendada Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ jaoks interaktiivne rakendus kahendotsimispuu, AVL-puu ja kuhja algoritmide läbimängu simuleerimiseks. Loodud programm võimaldab kasutajal algoritme samm-sammult läbimängida ning logib automaatselt kõik tehtud vead. See loob võimaluse asendada senise aeganõudva kirjaliku kontrolltöö koostamise ja kontrollimise protsessi. Rakendus on üles ehitatud viisil, mis võimaldab selle hilisemat integreerimist e-õppekeskkonda DeepMOOC, pakkudes lahendust nii algoritmide õppimiseks kui ka hindamiseks.

Võtmesõnad: Algoritmid ja andmestruktuurid, kahendotsimispuu, AVL-puu, kuhi, õpitarkvara

CERCS: P175 Informaatika, süsteemiteooria

Playthrough simulator for binary tree and heap algorithms

Abstract:

The aim of this bachelor's thesis was to develop an interactive application for the University of Tartu course “Algorithms and Data Structures” to simulate the step-by-step execution of binary search tree, AVL tree, and heap algorithms. The developed program allows users to interactively work through these algorithms while automatically logging all mistakes made during the process. This provides a foundation for objective and efficient assessment, replacing the currently time-consuming process of preparing and grading written tests. The application is designed to be integrated into the DeepMOOC e-learning platform, offering a solution for both learning and evaluating algorithmic understanding.

Keywords: Algorithms and data structures, binary search tree, AVL tree, heap, educational software

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus ... 4
1. Taust ... 5
    1.1 Probleem ... 5
    1.2 Sarnased lahendused ... 5
    1.3 Töövahendid ... 6
2. Terminid ... 7
3. Realiseeritud algoritmide kirjeldused ... 8
    3.1 Järjend kahendotsimispuuks ... 8
    3.2 Eemaldamine kahendotsimispuust ... 8
    3.3 Lisamine AVL-puusse ... 9
    3.4 Eemaldamine AVL-puust ... 10
    3.5 Järjendi kuhjastamine ... 10
    3.6 Kuhjameetod ... 11
4. Funktsionaalsed ja visuaalsed nõuded ... 12
    4.1 Üldised nõuded programmile ... 12
    4.2 Nõuded seoses eelnevate töödega ... 13
5. Rakendus ja edasiarendus ... 14
    5.1 Kontrollitud vead läbimängul ... 14
    5.2 Andmestruktuuride käsitlemine ja kuvamine ... 15
    5.3 Järjend kahendotsimispuuks ... 16
    5.4 Eemaldamine kahendotsimispuust ... 16
    5.5 Lisamine AVL-puusse ... 18
    5.6 Eemaldamine AVL-puust ... 19
    5.7 Järjendi kuhjastamine ... 20
    5.8 Kuhjameetod ... 20
    5.9 Edasiarenduse võimalused ... 21
Kokkuvõte ... 22
Viidatud kirjandus ... 23
Lisad ... 24
    1. Lähtekood ... 24
    2. Litsents ... 25

Sissejuhatus

Tartu Ülikoolis õpetatav aine „Algoritmid ja andmestruktuurid“ on informaatika õppekavale kohustuslik, mispärast osaleb sellel ainel korraga väga palju tudengeid. Aines käsitletakse olulisemaid andmestruktuure, milleks on massiivid, paisktabelid, puud ja graafid. Peamiseks algoritmide töö õpetamisviisiks on hetkel läbimänguslaidid ja hindamiseks kasutatakse kirjalikku kontrolltööd. Suure osalejate arvu tõttu osutub tööde koostamine ja hindamine üpris keeruliseks. Seetõttu tuleks luua nii õppimise kui ka hindamise jaoks mingi digitaalne süsteem, mis võimaldab tudengil algoritmide tööd harjutada ning hiljem sooritada ka seal kontrolltöö.

Bakalaureusetöö eesmärgiks on luua interaktiivne läbimängija kahendotsimispuude, AVL-puude ja kuhjade jaoks, mis logib automaatselt kõik tehtud vead hilisemaks hindamiseks. Selline rakendus aitab õppejõududel säästa aega kontrolltööde genereerimisel ja hindamisel. Samuti lubab programm tudengitel harjutada kursusel käsitletavaid algoritme potentsiaalselt kiirendades õppeprotsessi.

Töö on jaotatud neljaks suuremaks peatükiks. Peatükis 1 kirjeldatakse täpselt, milles seisneb probleem, analüüsitakse olemasolevaid lahendusi ning tuuakse välja põhjused, miks need ei ole piisavad. Lisaks on toodud välja kasutatud töövahendid. Peatükis 2 on edasise töö lugemiseks vajalikud mõisted välja toodud. Peatükk 3 sisaldab kõigi realiseeritud algoritmide kirjeldusi, mida loodud läbimängija sisaldab. Peatükis 4 on välja toodud kõik nõuded, mille järgi hakati rakendust arendama. Peatükis 5 antakse ülevaade rakenduse toimimise loogikast ning kasutamisest. Samuti analüüsitakse edasiarenduse võimalusi. Lisas 1 on saadaval programmi lähtekood.

1. Taust

1.1 Probleem

Hetkel puudub Tartu Ülikooli aines „Algoritmid ja andmestruktuurid“ algoritmide läbimängimise võimalus digitaalselt ning pole võimalik anda tudengile kiiret tagasisidet. Õppekava eeldab, et tudeng omandab olulisemate algoritmide täieliku mõistmise. Varasemalt on juba loodud läbimängijad massiivi-, paisktabeli- ja graafialgoritmidele. Siin töös keskendutakse kahendotsimispuu-, AVL-puu- ja kuhjaalgoritmidele. Probleem seisneb kirjalike tööde kontrollimises ja sisendite genereerimises. Kirjalik algoritmide kujutamine on aeganõudev nii tudengile kui ka õppejõule, mistõttu on vajalik alternatiiv, mis aitaks säästa aega ja oleks kasutajasõbralik. Kuigi internetist on võimalik leida mõningaid algoritmide visualiseerijaid, ei ole need piisavalt mugavalt kasutatavad ega võimalda automaatset vigade logimist ja interaktiivset andmestruktuuri muutmist.

Sisendite genereerimine on varasemalt realiseeritud Renno Seppa lõputöös [1], kus genereeritakse sisendeid vastavalt numbrilisele raskusastmele. Siinses lõputöös arendati interaktiivne tööriist, mis võimaldab tudengitel kahendpuu- ja kuhjaalgoritme läbimängida vastavalt sisendi järgi ning õppejõul tudengit automaatselt hinnata, mis on realiseeritud vigade logimisena. Läbimängijat kasutades on võimalik tudengile anda kohest tagasisidet ja lihtsustada arusaamist algoritmi toimimisest. Lisaks avardab see tööriist võimalusi algoritmide õpetamiseks ja hindamiseks e-õppekeskkonnas DeepMOOC, mille osaks see programm tulevikus saab.

1.2 Sarnased lahendused

Mitmed lahendused on juba loodud algoritmide visualiseerimiseks ja mõnel määral läbimängimiseks. Selles peatükis analüüsitakse neist populaarsemaid.

San Francisco Ülikoolis loodud visualiseerija [2] on andmestruktuuri muutuste jälgimiseks üpris hea, kuid läbimängimiseks see hästi ei sobi. Kahendpuude ja kuhjastamisega on võimalus elementide üksikuks lisamiseks ja eemaldamiseks, kuid algse puu või kuhja struktuuri loomist pole võimaldatud. Kuhjameetodi visualiseerimine toimib sarnaselt siin töös valminud lahendusega, kuid kõigi algoritmide ühine probleem on see, et kasutaja ei saa ise reaalselt midagi andmestruktuurides liigutada ega muuta.

Singapuri Ülikoolis loodud VisuAlgo [3] keskkonnas on olemas vastavalt siin töös loodud läbimängijale kõigi algoritmide visualiseerijad peale kuhjameetodi. Algoritmidele ei saa jällegi ette anda sisendeid ja kasutajal ei ole võimalik midagi struktuurides ise muuta. Ehk tegu on puhtalt visualiseerimistööriistaga. AVL-puude puhul olid samade väärtustega elemendid lisatud ühe tipu kohale, mis ei ole kooskõlas selle töö lahendusega. Selle visualiseerija heaks küljeks see, et algne kahendotsimispuu ja kuhi olid juba suvaliste väärtustega ette loodud.

Olemasolevate lahenduste kõige suuremaks puuduseks on esiteks see, et kasutaja ei saa ise tippe lisada ega liigutada. Kui läbimängu tahta hinnata, siis on vaja lubada kasutajal kindlasti ise tippe lisada, eemaldada ja andmestruktuuris muutuseid teha, ehk hetkel on loodud vaid algoritmide visualiseerijad. Teiseks ei luba veebikeskkonnad ka oma sisendeid lisada, mis on oluline nii hindamiseks kui ka õppimiseks.

1.3 Töövahendid

Programm on arendatud Java programmeerimiskeeles, kasutades graafilise kasutajaliidese loomiseks JavaFX raamistikku. Valik langes Java ja JavaFX kasuks mitmel põhjusel.

Esiteks on rakendus loodud osaks DeepMOOC keskkonnast, mida arendatakse Kotlin programmeerimiskeeles [4]. Kuna Java ja Kotlin on kompileeritavad Java virtuaalmasina baitkoodiks, siis on Java koodi hiljem lihtne olemasolevasse Kotlini põhisesse keskkonda juurde lisada.

Teiseks toetab valitud JavaFX raamistik varasemate tööde ühte keskkonda integreerimist. Näiteks on Tartu Ülikooli varasema lõputööna valminud graafialgoritmide läbimängija [5], mille arendusel valiti programmeerimiskeeleks Java ja kasutajaliidese raamistikuks JavaFX. See võimaldab hiljem erinevate algoritmide läbimängijate ühildamise ühte keskkonda teha võimalikult valutuks ning muudab ka edasise arenduse tõhusamaks.

2. Terminid

Raskusparameeter tähistab sisendi lahendamise raskust vastava numbrilise parameetrina [1].

Kahendpuu (ingl *binary tree*) on andmestruktuur, kus igal tipul on ülimalt üks ülemtipp ja mitte enam kui kaks alamtippu [6].

Kahendotsimispuu (ingl *binary search tree*) on kahendpuu, mis on tühi või kus iga ülemtipu vasaku alluva väärtus on väiksem ja parema alluva oma suurem kui ülemtipu väärtus [6].

AVL-puu (ingl *AVL tree*) on kahendotsimispuu, mis on tühi või tema vasaku ja parema alampuu kõrguste vahe on ülimalt üks ning mõlemad need alampuud on AVL-puud [6]. Lühend AVL on tuletatud selle andmestruktuuri loojate Adelson-Velski ning Landise nimede initsiaalidest.

Kompaktne kahendpuu (*ingl compact binary tree*) saadakse kahendpuust, kus kõik lehttipud on samal tasemel, eemaldades null või rohkem lehttippu viimase taseme lõpust [6].

Kahendkuhi (ingl *binary heap*) on kompaktne kahendpuu, kus iga tipu võtmeväärtus on alamtippude omast suurem või sellega võrdne [6].

Allaviimise operatsioon tähendab puus ülemtipu vahetamist oma alluvatega seni kuni tal ei ole enam alluvaid või ta on mõlemast oma alluvast vastavalt, kas suurem või väiksem olenevalt reeglist [6].

Pöördkuhi (ingl *min-heap*) on osaliselt järjestatud kahendpuu, mille iga tipu võtmeväärtus on alamtippude omast väiksem või sellega võrdne [6].

3. Realiseeritud algoritmide kirjeldused

Käesolev peatükk annab teoreetilise ülevaate kuue algoritmi kohta, mida loodud tööriist käsitleb. Samade algoritmide kohta on varasemalt loodud lõputöö „Sisendite genereerimine puude ja kuhjade algoritmidele" Renno Seppa poolt [1]. Sisendeid genereeritakse vastavalt raskusparameetrile [1], millest tuleneb ka läbimängu keerukus.

3.1 Järjend kahendotsimispuuks

Järgnev algoritmi kirjeldus põhineb kursuse „Algoritmid ja andmestruktuurid" Moodle'i materjalide [7] põhjal. Järjendi iga element töödeldakse ükshaaval, alustades järjendi elemendist indeksil 0, lisades esimene element puu juurtipu väärtuseks. Iga järgnevat elementi võrreldakse olemasolevate puu tippudega järgmiselt: kui element on väiksem, liigutakse puu vasakusse harru, vastupidisel korral paremasse. Kui element on võrdne võrreldavaga siis liigutakse paremasse harusse. Element lisatakse kahendotsimispuu tipuks kui jõutakse harru, mida ei ole veel olemas. Algoritm jätkab kuni järjend on tühi.

Selle algoritmi raskusparameetriks peab olema järjendi elementide arv, kuna algoritmi keerukus sõltub võrdluste arvust iga elemendi lisamisel [1].

3.2 Eemaldamine kahendotsimispuust

Jüri Kiho õpiku põhjal [6]: kui eemaldatav tipp on puu lehttipp, siis saab selle lihtsalt puust välja jätta. Kui tipul T on ainult üks järglane, siis saab eemaldada T ja asendada ta oma ainsa järglasega. Kui aga eemaldataval tipul T on olemas mõlemad järglased, siis peab eemaldama tipu T parema haru väikseima väärtusega tipu V ning seejärel asendama tipu T väärtuse tipu V omaga. Kui juhtub, et tipp V ei olnud lehttipp, siis ei eemaldata veel puust tippu V, vaid korratakse sama algoritmi tipus V seni kuni eemaldatava tipu väikseim tipp on lehttipp või parem alluv on ainus alluv.

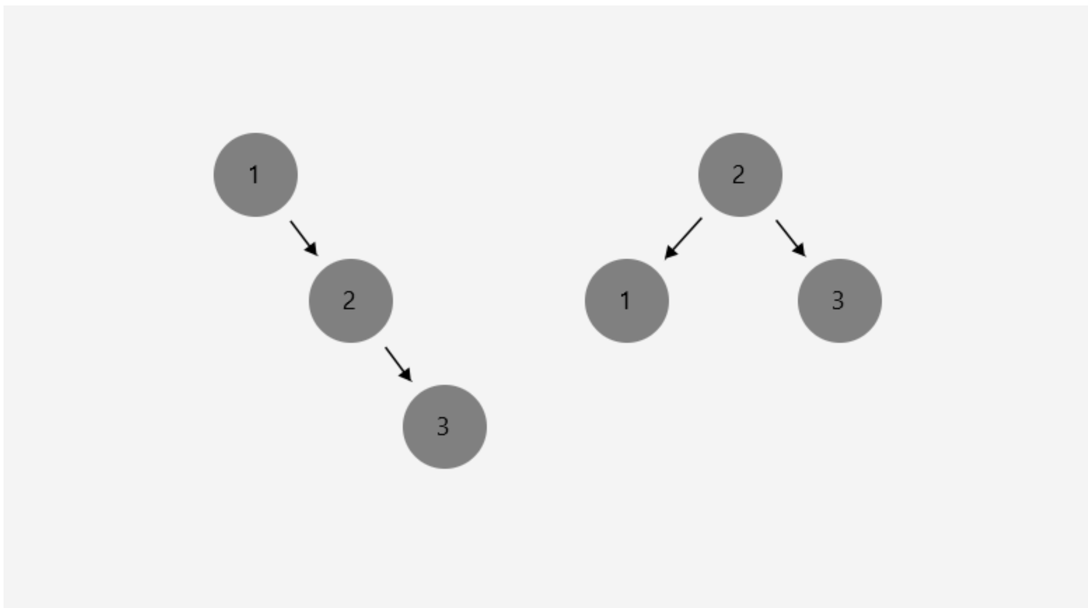
Siin on raskusparameetri väärtus 1-5 vastavalt eemaldatava elemendi keerukusele [1].

Elementide lisamine ja eemaldamine kahendotsimispuus on keskmise ajalise keerukusega $O(h)$, kus h on puu kõrgus [6]. Elementide eemaldamisel ja lisamisel muutub aga harude kõrguste vahe ebaproportsionaalseks ning seega puu kõrgus ebavajalikult suureks. Halvimal juhul on vaja sooritada puu kõrgusega võrdne arv võrdlusoperatsiooni. Selle probleemi

lahendab AVL-puu, mis säilitab harudevahelise kõrguste tasakaalu. Tänu tasakaalustamisele on AVL-puus elemendi lisamise ja eemaldamise keskmine ajaline keerukus $\Theta(log\ n)$, kus n tähistab puu elementide arvu.

3.3 Lisamine AVL-puusse

Element lisatakse AVL-puusse samamoodi nagu seda tehti kahendotsimispuusse, vahe on aga selles, et nüüd peab kontrollima lisatud tipu iga ülema juures tema alampuude kõrguste vahet kuni juurtipuni [7]. Kui alampuude kõrgus peaks erinema mingis tipus rohkem kui ühe võrra siis on vaja sooritada tasakaalustusoperatsioon, milleks võib olla parem-, vasak-, paremvasak- või vasakparempööre [7]. Ühe elemendi lisamisel tehakse maksimaalselt üks nimetatud pööretest [7]. Selle pöörde sooritamine tagab, et kahendotsimispuu iga tipu alampuude kõrguste vahe oleks väiksem kui 2. Joonisel 1 on kujutatud vasakpööre vasakpoolse puu tipus 1, kus tipp 2 saab uueks juurtipuks ja tipp 1 lisatakse tipu 2 vasakuks alluvaks.



Joonis 1. AVL-puu vasakpööre.

Raskusparameeter sõltub siin sooritatavate pöörete ja lisatavate elementide arvust, sest iga lisatud tipu ülemate juures on vaja otsustada, kas ja milline pööre sooritada [1].

3.4 Eemaldamine AVL-puust

Elemendi eemaldamine toimib samuti samamoodi nagu kahendotsimispuus. Erinevus on jällegi puu tasakaalustamises. Pärast tipu eemaldamist on vaja sooritada tasakaalukontroll eemaldatud tipust kuni puu juurtipuni ja vajadusel sooritada ka tasakaalustamised pöörete abil [7]. Erinevalt AVL-puusse lisamisega peab eemaldamisel kontrollima kõrguste vahesid ülemates ka pärast esimese pöörde sooritamist, sest eemaldamine võib põhjustada tasakaaluhäireid mitmes järjestikuses tipu tasemes kuni juurtipuni [7].

Raskusparameetriks sobib siin sooritavate pöörete arv, sest eemaldatud tipu iga ülema juures on vaja otsustada, kas ja milline pööre sooritada [1].

3.5 Järjendi kuhjastamine

Järjendi kuhjastamiseks on erinevaid variante, kuid aines „Algoritmid ja andmestruktuurid“ kasutatakse vaadeldakse järjendit kompaktse kahendpuuna ning kuhjastatakse mullina allaviimise operatsiooni abil [7]. Alustades järjendi lõpust, peab leidma töödeldava elemendi ülema ja võrdlema seda tema alluvatega, kui ülem on väiksem vähemalt ühest oma alluvast, siis vahetatakse ta nendest suuremaga [7]. Jätkates sama loogikat kõigi järjendi elementidega, selliselt, et kui ülemat T vahetatakse mingi elemendiga, siis vahetatakse (allaviimise operatsioon) elementi T oma uute alluvatega edasi kuni ta on kas lehttipp või suurem mõlemast alluvast. Tulemuseks on järjend, milles iga element vastab kuhja tingimusele. Kuhjastades tingimusel, et vahetakse ülemtipp väikseima elemendiga oma alluvatest, moodustub pöördkuhi.

Siin sobib raskusparameetriks elementide võrdluste arv kuhjastamisel, sest mida rohkem võrdlusoperatsioone sooritatakse seda keerukam kuhjastamine on [1].

3.6 Kuhjameetod

Järgnev lõik tugineb J. Kiho õpikule [6]. Kuhjameetod on parimal juhul ajalise keerukusega $\Theta(n)$ ja halvimal juhul $\Theta(n \, log \, n)$, kus n on järjendi elementide arv. Meetodi realiseerimiseks tuleb kõigepealt kuhjastada järjend, mille ajaline keerukus on $\Theta(n)$. Pärast kuhjastamist vahetatakse kuhja suurim element järjendi viimase elemendiga. Seejärel vahetatakse kuhja juurtipus olev suurim element järjendi viimase elemendiga ning eemaldatakse see aktiivsest kuhjast. Kuhja omaduste säilitamiseks rakendatakse juurele allaviimise operatsiooni. Protsessi korratakse – igal sammul eemaldatakse suurim element ja taastatakse kuhi – kuni kogu järjend on sorteeritud mittekahanevas järjestuses.

Raskusparameeter on jällegi elementide võrdluste arv nagu ka järjendi kuhjastamisel, sest see on keerukaim operatsioon kuhjameetodil [1].

4. Funktsionaalsed ja visuaalsed nõuded

Eeltööna oli enne iga algoritmi läbimägija kirjutamise alustamist vaja paika panna kindlad nõuded algoritmi väljanägemisele ja piirangud tudengile lubatud liigutuste kohta puu ja kuhja struktuurides. Järgnevas peatükis on kirjeldatud neid nõudeid.

4.1 Üldised nõuded programmile

Põhieesmärgiks oli luua programm, millega saaks ennekõike asendada kirjaliku kontrolltöö aines „Algoritmid ja andmestruktuurid“ virtuaalsega. Esiteks tähendab see, et kasutajale peab jätma kahendpuu muudatustes üpris suure vabaduse: simuleerimaks kirjalikku tööd, mõningate eranditega. Näiteks järjendi lugemisel kahendotsimispuuks ei lasta kasutajal olemasoleva puu struktuuri muuta, sest selline viga tähendaks algoritmi mittetundmist ja järelikult võib eeldada, et kasutaja teeb läbimängu käigus ka muid vigu. Teiseks, kasutaja hindamiseks peab arvestama tema poolt tehtavate vigadega ja need logima. Hindamise juures on oluline, et kõik kasutaja poolt tehtud vead logitakse ja samal ajal ei öeldaks ette korrektseid samme algoritmi töös.

Iga algoritmi läbimängija peab kasutama sisendit vastavalt raskusparameetrile, mis genereeritakse sisendite genereerimise programmis [1]. See tähendab, et sisend igale algoritmile peab vastama täpselt antud programmi väljundile.

Kõik algoritmid peaks visuaalselt sarnased välja nägema, et suurendada kasutajasõbralikkust. Kindlasti peab olema kasutajale näha hetkel töödeldav sisendjärjend ning vajadusel hetkel töödeldav element. Samuti võiks kuvatud olla juhend tegevustest, mis on kasutajal lubatud teha, et ta saaks keskenduda algoritmi läbimängule ja ei peaks mõistatama, mida miski nupp või tegevus teha võiks.

Kasutajale kuvatud puu peab olema ilusa struktuuriga, aga samas peab lubama tal liigutada puu tippe. See tähendab, et iga tipp peab olema oma kindlal tasemel, et eristada ülemtippe alamtippudest. Tippe peab saama liigutada, et kasutaja saaks endale mugavalt puu struktuuri laiemaks venitada. Tipu vasakut alluvat ei tohi lasta liigutada oma ülemusest paremale ega ka vastupidi ja samuti mitte ka kõrgemale. Pärast suuremat struktuuri muutust peab puu struktuuri uuesti ilusaks kuvama, et ei tekiks tippude tasemete ja alluvate eristamisel segadust. Selline puu visuaalne kuvamine tagab, et kasutaja eristab kogu kahendpuu struktuuri muutmise vältel kõiki tippe.

Kasutajal peab olema võimalus minna sammu võrra tagasi eelnevalt kontrollitud puu olekusse. See on oluline esiteks sellepärast, et paberi peal on alati selline võimalus olemas. Teiseks, kui tehakse näiteks vale lisamine või eemaldatakse vale element, siis peab olema võimalus ilma aega kulutamata minna tagasi eelnevasse puu struktuuri ja oma viga parandada.

4.2 Nõuded seoses eelnevate töödega

Et hilisem läbimängijate kokku integreerimine oleks sujuvam võiks graafiline liides sarnaneda juba olemasolevatele. Kuna hetkel on graafiline liides loodud vaid graafialgoritmide läbimängijale [5], siis oli mõistlik võtta kujunduse eeskuju just sealt. Samuti sarnaneb graafi struktuur kahendpuu omale kõige enam võrreldes teiste seni loodud läbimängijatega nagu näiteks paisktabeli või massiivi omadega.

Graafilise liidese põhiliseks nõudeks võeti see, et ta oleks graafi algoritmide läbimängijaga sarnane. See tähendab, et kasutajaliideses elementide paigutus ja puu kuvamine peaks olema sarnase kujundusega. Esiteks soodustab see tudengil intuitiivset arusaamist erinevatest läbimängijatest, kui ta on ühte juba kasutanud, teiseks võiks see hilisemal programmide DeepMOOC keskkonda tõstmisel soodustada rakenduste ühtlustamist.

Rakenduse struktuur ja failijaotus püüti hoida samuti sarnane graafialgoritmide läbimängijale eelnevalt nimetatud põhjusel. See hõlmas põhiliselt rakenduse sarnast arhitektuuri ja logifailide hoiustamist. Selline lähenemine lihtsustab hilisemat rakenduste kokku tõstmist ja lisaks vähendades vajadust suuremate arhitektuuriliste muudatuste järele.

5. Rakendus ja edasiarendus

Iga algoritmi puhul pidi arvestama nende eripäradega. Samuti tuli välja mõelda moodus, kuidas programm puid käsitleb ja vigu kontrollib. Järgnevas peatükis kirjeldatakse, milliste vigadega oli vaja arvestada, kuidas programm taustal toimib, rakenduse kasutamist ning kirjeldatakse võimalikke edasiarendusi. Läbimängija lähtekoodi link on leitav Lisas 1.

5.1 Kontrollitud vead läbimängul

Erinevate andmestruktuuride korral on vaja kontrollida puu olekuid erinevatel hetkedel ja erinevate kontrollidega vigade eristamiseks. Vigu eristatakse suures plaanis andmestruktuuride kaupa.

Kahendotsimispuus kontrollitakse puu olekut peale iga elemendi lisamist või eemaldamist. Kõige olulisem on säilitada vajalik kahendotsimispuu struktuur, kui see kaotatakse, siis loetakse seda suurimaks võimalikuks veaks, misjärel peab sooritama terve elemendi lisamise või eemaldamise töö uuesti. Teiseks võimalikuks veaks on see, et konstrueeritakse vale kahendotsimispuu. Sellisel juhul jätkatakse läbimängu uue kasutaja poolt loodud puuga.

AVL-puid peaks kontrollima samadel tingimustel, mis olid kahendotsimispuualgoritmidel. Erinevus tekib puu struktuuri kontrollimisel. Siin on vaja kontrollida ka AVL-puu struktuurile vastavust. See tähendab, et kui kasutaja poolt loodud puu ei vasta AVL-puu tingimustele, siis peab kogu lisamise või eemaldamise uuesti sooritama, sama kehtib ka kui puu ei ole enam kahendotsimispuu. Kui lisamise või eemaldamise tulemusena on loodud vale AVL-puu struktuur, siis viga logitakse ja kasutajal lastakse uus operatsioon teha tema poolt loodud uue puuga.

Kuhjade korral pole enam kahendotsimispuu tingimusi vaja kontrollida, nüüd on potentsiaalseteks veakohtadeks kompaktse kahendpuu struktuur, kuhjameetodi puhul õige elemendi töötlemine, kuhjameetodi samm ning elementide vahetamine allaviimise teel. Kompaktse kahendpuu kontrollimiseks saab iga elemendi lisamisel valesse kohta logida vea ja lasta kasutajal uuesti lisada sama element kuhja. Kuhjameetodi rakendamisel peab alati töötlema kuhja viimase elemendi, seega lastakse kasutajal töödelda vaid kuhja viimast elementi, vastasel korral logitakse viga. Kuhjameetodi sammu kontrolliks vaadatakse, kas kuhjast on eemaldatud juur korrektselt ja kuhi vastab kuhja tingimustele, kui mitte siis peab

selle sammu uuesti sooritama. Tipu allaviimisel oli kaks võimalust, kuidas kontrolli sooritada: kas kontrollida, et mõlemad vahetatavate tippude alamkuhjad on kuhjad või kontrollida kuhja tingimust ainult pärast kogu kuhjastamist ehk pärast iga tipu allaviimist. Siin töös valiti esimene variant kuna teise valiku korral võib kasutaja vahetada tippe selliselt, et algoritmi ajaline keerukus muutuks ebasoodsaks. Ehk iga vale tipu allaviimisel logitakse viga.

5.2 Andmestruktuuride käsitlemine ja kuvamine

Kolme andmestruktuuri jaoks loodi kaks eri klassi, milleks olid *Kahendotsimispuu* ja *Kuhi*. AVL-puu funktsioonid lisati *Kahendotsimispuu* klassi, kuna ainus erinevus nendes andmestruktuurides seisnes kahendotsimispuu tasakaalustamises. Lisaks sisaldab klass *Kahendotsimispuu* välja *juurtipp* klassist *Tipp* ja klass *Kuhi* järjendit kuhja elementidest täisarvudena.

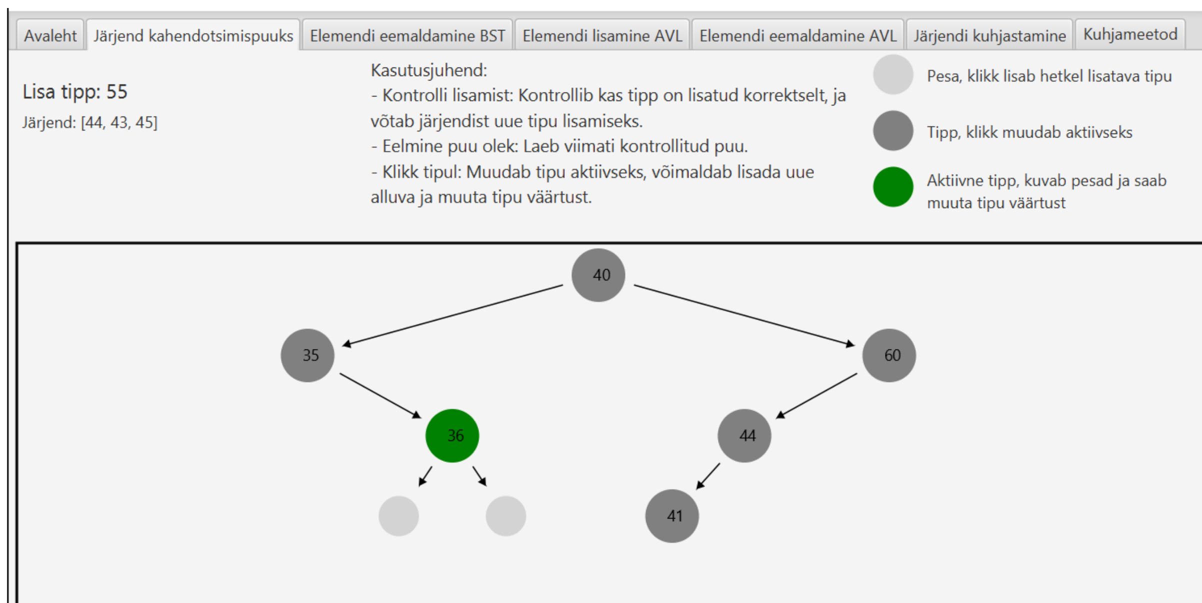
Klassis *Tipp* on väljad tipu väärtuse, vasaku ja parema alluva ning visuaalse tipu jaoks. Visuaalne tipp sisaldab siin viidet klassi *VisuaalneTipp* objektile, mis on loodud tippude ekraanil kuvamiseks. Selleks, et puu kuvamine õnnestuks, lisati tavapärasele *Tipp* klassile ka väli *tase*, mis aitab kindlustada, et alamtipp ei satuks visuaalselt oma ülemusest kõrgemale. Samuti on ka kuhja jaoks eraldi väli *indeks*, mis sisaldab elemendi indeksit kuhja järjendis.

Algoritmi töö läbimängimiseks ja kontrollimiseks on loodud igale algoritmile eraldi kontroller klass. Kontrolleris toimub andmestruktuuri kuvamine, tippude ja nuppude funktsionaalsus ning vigade logimine.

Iga algoritmi korral hoitakse sarnaselt kolme puud/kuhja korraga mälus: eelneva seisuga, visuaalne ja korrektne. Eelneva seisuga puu/kuhi on selleks, et hoida meeles viimati kontrollitud struktuuri, ehk kui kasutaja vajutab „Lae eelnev…“ siis kuvatakse kasutajale see struktuur. Visuaalne tähistab puud või kuhja, mida kasutaja on ekraanil loonud ja muutnud. Korrektne puu/kuhi sisaldab struktuuri, kus on vajalik operatsioon juba tehtud ning sellega kontrollitakse visuaalse puu/kuhja korrektsust. Selle programmi puhul ei ole kolme struktuuri korraga mälus hoidmine probleemiks, kuna see on mõeldud õppimiseks ja hoitavates andmestruktuurides ei ole korraga palju elemente.

5.3 Järjend kahendotsimispuuks

Sellel läbimängul loetakse vastavalt sisendile kasutajale ette järjend, millest hakatakse kahendotsimispuud konstrueerima. Kasutaja hakkab elemente järjest kahendotsimispuusse lisama ja peab saama lisada uue elemendi järjendist kõigi võimalike tippude alluvaks, välja arvatud juurtipu millel on üks kindel positsioon. See garanteerib, et kasutaja saab lisada elemendi valesse kohta, mis ei vasta kahendotsimispuu reeglitele nagu on näidatud joonisel 1, kus saab hetkel lisada tipu 55 valitud tipu 36 alluvaks.

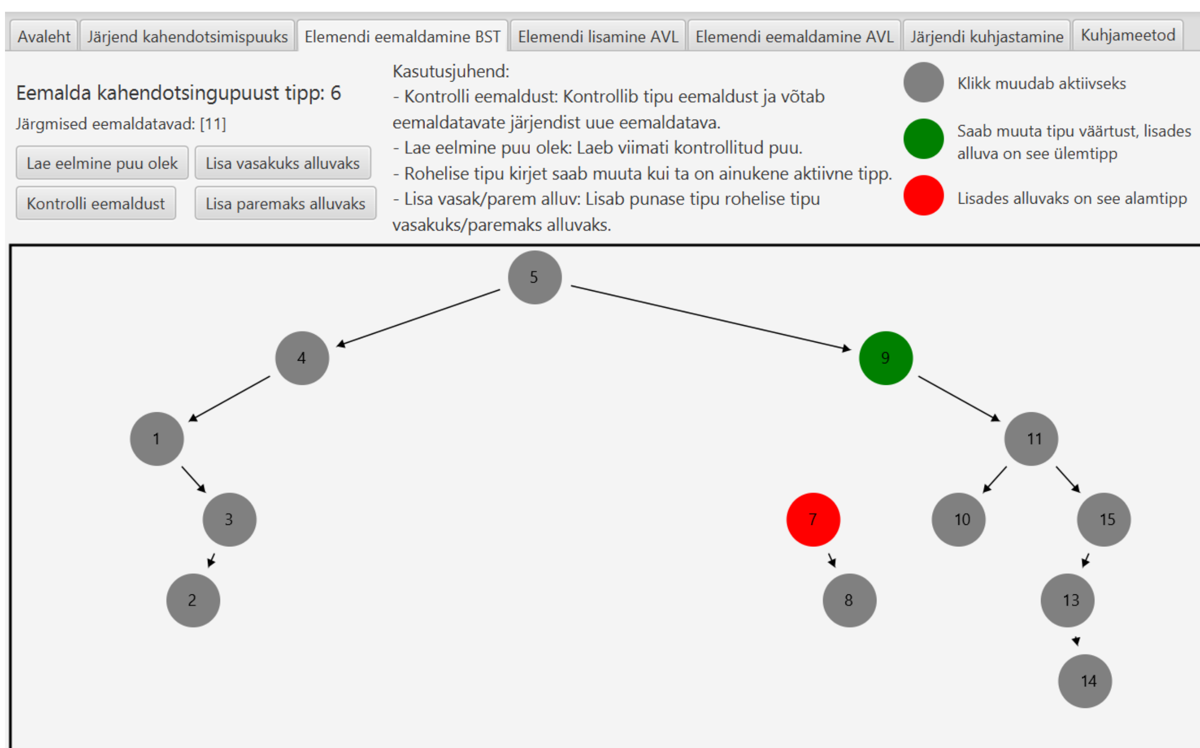


Joonis 2. Järjend kahendotsmispuuks läbimäng.

Samuti lubatakse muuta tippude väärtuseid eesmärgiga, et algoritmi tööga mitte tuttaval kasutajal oleks rohkem võimalusi eksida. Peale tipu lisamist on kasutajal võimalus samm tagasi võtta, kontrollida lisamist või muuta edasi tipu kirjeid. See peaks soodustama vigade tegemist, sest tegelikult algoritmis pole peale tipu lisamist enam midagi teha vaja.

5.4 Eemaldamine kahendotsimispuust

Elemendi eemaldamisel peab kasutaja eemaldama puust märgitud tipu, kuid lubatakse eemaldada kahendotsimispuust ükskõik milline element, välja arvatud kahe alluvaga element ja juur, sest nende eemaldamine ei vastaks algoritmi tööle ning see logitakse veana. Kasutajal on veel kaks võimalust, mida tippudega teha: muuta tipu kirjeid valides ühe tippu või lisada mingi alampuu mingi tipu alluvaks. Võimalus lisada ükskõik millist alampuud mingi tipu alluvaks, kui vastava alluva kohal juba alluvat pole, muutes sellega puu struktuuri, peaks simuleerima umbes sama olukorda nagu paberil algoritmi tööd näidates. Joonisel 2 on kujutatud selline olukord, kus puust on eemaldatud tipp 6 ning järgnevalt õige samm oleks lisada tipu 9 vasakuks alluvaks tipp 7 ja seejärel eemaldust kontrollida.

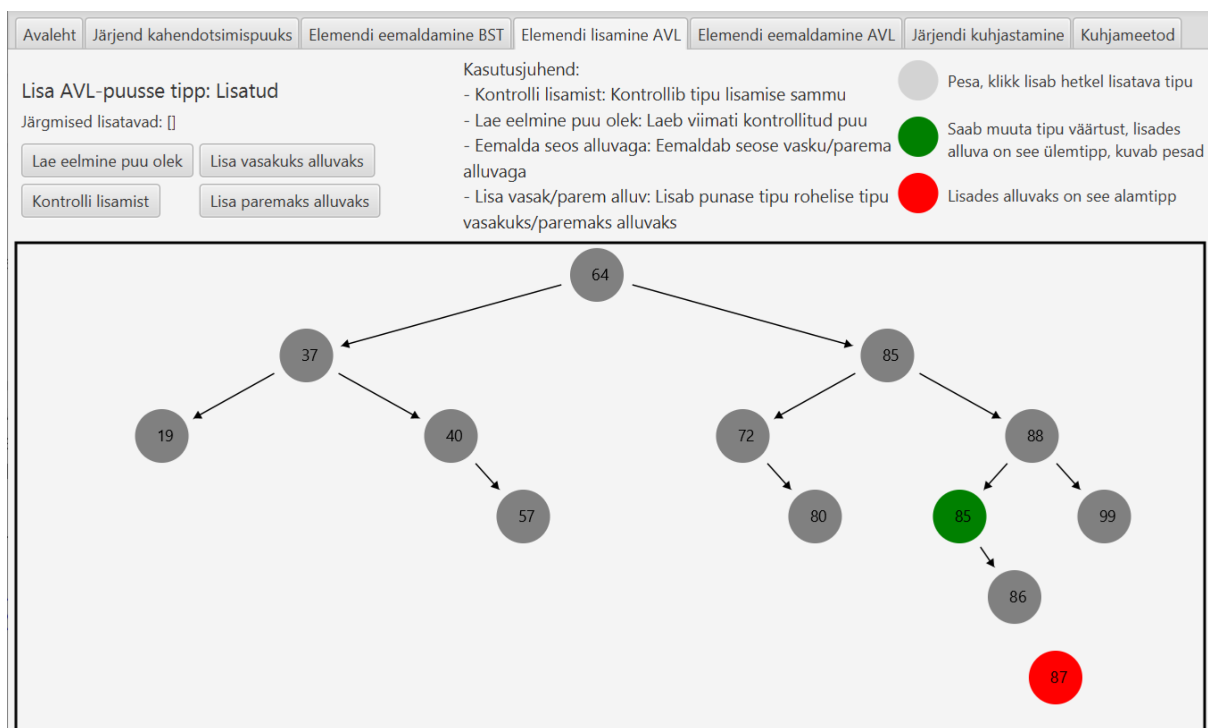


Joonis 3. Eemaldamine kahendotsimispuust läbimäng.

Sellele lahendusele korrektne alternatiiv oleks muuta tippude kirjeid ja eemaldada puust tipp, mille väärtus hetkel on 8. Peale mingi elemendi eemaldust on kasutajal ka alati võimalus võtta samm tagasi – nupu „Lae eelmine olek“ näol – selleks, et vale eemalduse korral oleks võimalik kiiresti eemalduse algolekusse tagasi liikuda.

5.5 Lisamine AVL-puusse

Kasutajale on, vastavalt sisendile ülesande loomisel, korrektse struktuuriga AVL-puu ette antud. Vastavalt lisatavatele elementidele hakatakse neid ühekaupa puusse lisama. Kasutajal on jällegi võimalik lisatav tipp ükskõik millise tipu alluvaks lisada. Kahendotsimispuudega võrreldes on nüüd erinevuseks pöörete tegemise võimalus ja sellega puu struktuuri täielik muutmine. Et pöördeid visuaalselt lihtsustada, saab eemaldada ülemtipu ja alamtipu vahel seoseid ja need pärast vastavalt algoritmile teiste tippudega kokku ühendada. Joonisel 3 on kujutatud olukord, kus puusse on lisatud tipp 87 vastavalt algoritmile ja eemaldatud sellega seos. Nüüd peaks sooritama vasakpöörde tipus 85, pärast mida on 87 korrektselt lisatud.

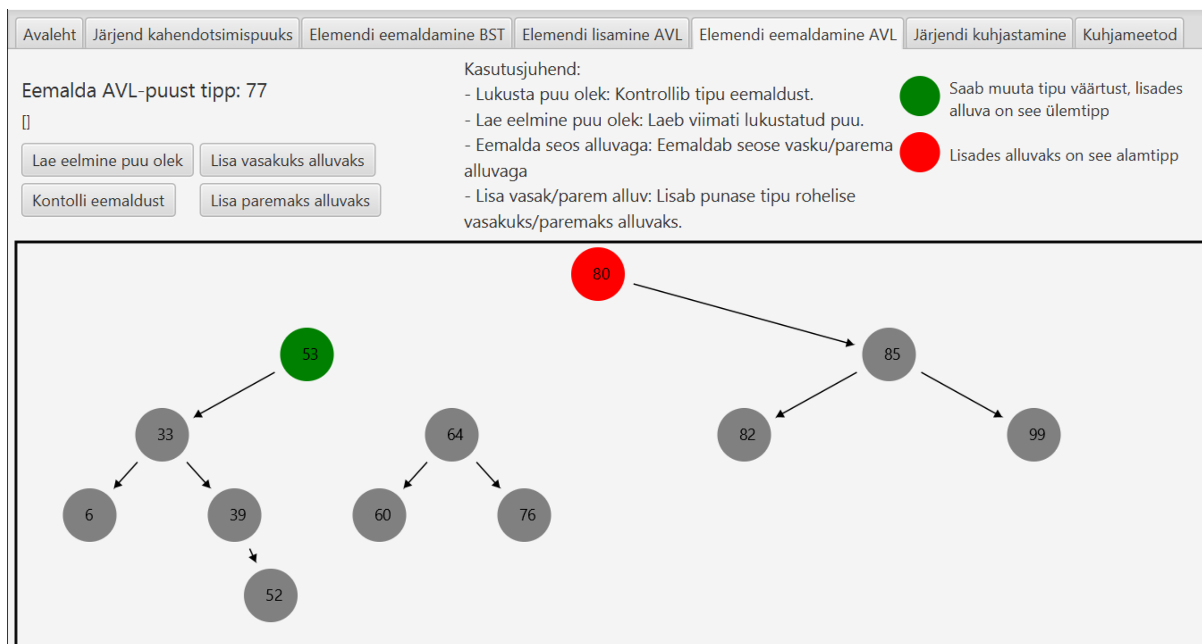


Joonis 4. AVL-puu lisamine läbimäng.

Lisamisel on võimalus iga tipu kirjet muuta ja lisaks ka pöördeid sooritada ilma seoseid eemaldamata. See annab kasutajale erinevaid võimalusi, kuidas ülesannet visuaalselt lahendada sarnaselt paberil tehtava tööga. Kui elemendi lisamine ja vajalikud pöörded on sooritatud, tuleb kontrollida lisamist, misjärel võetakse lisatavate elementide järjendist uus element või kui järjend on tühi, siis lõpetatakse läbimäng.

5.6 Eemaldamine AVL-puust

Siin on samuti vastavalt sisendile loodud algne AVL-puu, kust kasutaja hakkab elemente eemaldama. Sarnaselt AVL-puusse lisamisega on siin võimalikeks operatsioonideks tippude seoste eemaldamine, tipu lisamine alluvaks ja tipu kirje muutmine. Puu struktuuri saab muuta ükskõik milliseks, tagamaks paberi peal läbimängu sarnasuse. Joonisel 4 on välja toodud olukord, kus puust on juba eemaldatud tipp 77, mis oli juurtipp ja selle asemele pandud vastavalt algoritmile tipp 80. Nüüd tuleks kasutajal lisada korrektseks lahenduseks 80 tipu 53 alluvaks ja 64 tipu 80 alluvaks. Pärast sellist pööret on puu uueks juurtipuks 53.

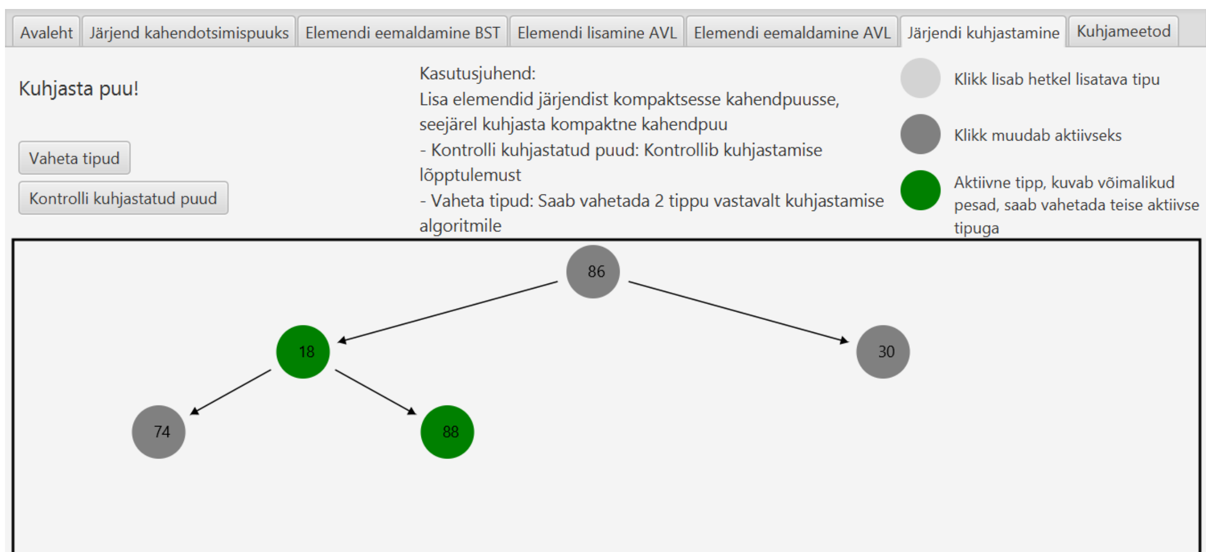


Joonis 5. AVL eemaldamine läbimäng.

Kuna puu struktuuris saab kõike muuta, siis on jällegi tagatud sarnane läbimäng paberi peal lahendatava kontrolltööga. Pöörete sooritamisel võib juhtuda, et tipud kattuvad visuaalselt. Sellisel juhul on võimalus tippe liigutada, et tagada struktuurist ühene arusaam.

5.7 Järjendi kuhjastamine

Enne kui hakatakse puu struktuuris muutusi tegema peab kasutaja looma sisendjärjendist kompaktse kahendpuu. Selle puu loomisel lubatakse lisada tipp ainult vastavalt algoritmile esimesele vabal kohale puus, kui üritatakse valesse kohta tipp lisada, siis käsitletakse seda veana. Pärast kompaktse kahendpuu loomist on vaja puu struktuur kuhjastada tippude allaviimise teel operatsiooniga „Vaheta tipud“. Joonisel 5 on kujutatud olukord, kus järjend on juba loetud kompaktseks kahendpuuks. Järgnevalt peab vastavalt algoritmile viima tipu 18 alla ning seejärel tipu 86.



Joonis 6. Järjendi kuhjastamise läbimäng.

Rakendades tippude allaviimist kontrollitakse enne iga vahetust, kas tipu mõlemad harud vastavad kuhja tingimustele. Kontrollides nii iga tipu vahetust tagab, et saadav kuhi on ühene ja kasutaja ei tohi teha ühtegi vale liigutust algoritmi läbimängus.

5.8 Kuhjameetod

Siin kuvatakse kasutajale ette sisendjärjend ja sellest loodud kuhja struktuurile vastav puu ehk kasutaja ei pea enam ise kuhja looma, sest seda on juba eelmise läbimängu käigus kontrollitud. Vastavalt algoritmile peab kasutaja hakkama järjest kuhja viimaseid tippe töödelduks märkima, millega lukustatakse element sorteeritava massiivi lõppu ja eemaldatakse puust. Pärast elemendi töötlemist peab taas kuhja struktuuri taastama juurtipu allaviimise teel. Joonisel 6 on

kujutatud olukord, kus tipud 8 ja 10 on eelnevalt töödeldud ning märgitud massiivis roheliseks. Järgnevalt peaks kasutaja viima juure, milleks on hetkel 4, vahetamiste teel alla.

Joonis 7. Kuhjameetodi läbimäng.

Kui kasutaja peaks kontrollima sammu enne kui kuhja struktuur on taastatud või on töödeldud vale element, siis loetakse see veaks ja elemendi töötluse peab uuesti sooritama.

5.9 Edasiarenduse võimalused

Valminud rakenduse eesmärk on seda tulevikus kasutada kontrolltööde hindamiseks programmeerimise õppeks arendatavas e-õppekeskkonnas DeepMOOC. Hetkel on hindamise osa programmis tehtud logifailina, milles on kõik kasutaja poolt tehtud vead ja ka korrektsed sammud. Tulevikus, kui rakendus integreeritakse DeepMOOC keskkonda, saab luua nende vigade põhjal hindamismaatriksi, mille põhjal peale läbimängu ka hinde anda. Samuti võib olla vajalik lõplikul hindamisel suuremate vigade korral läbimäng koheselt lõpetada, mille korral praegu nõutakse vajaliku sammu kordamist.

Kuna praegu keskendutakse läbimängu käigus rohkem hindamisele ning tudengile ei anta väga palju tagasisidet, siis on üks võimalus lisada ka eraldi harjutus- või õppimisrežiim. Harjutamise käigus võiks kasutaja vigade kohta võimalikult palju tagasisidet saada, sest praegu, selleks et kasutajale vähem infot õige sammu kohta anda, seda ei tehta. Lisaks on harjutamise mõttes võimalus lisada ka uusi algoritme sarnase koodiga, mis töö käigus on valminud. Näiteks AVL-puude puhul oleks kasulik eraldi erinevate pöörete harjutamine, kus kasutajale antakse ülesandeks sooritada kindel pööre nõutud tipus.

Kokkuvõte

Töös arendati interaktiivne rakendus Tartu Ülikooli ainele „Algoritmid ja andmestruktuurid“, mille eesmärk on võimaldada tudengil iseseisvalt läbi mängida kuut algoritmi, keskendudes kahendotsimispuu, AVL-puu ja kuhja andmestruktuuridele. Iga algoritmi läbimäng põhineb eelnevalt genereeritud sisendil, mille alusel hinnatakse tudengi arusaama algoritmi tööpõhimõtetest. Programm logib automaatselt kõik kasutaja tehtud vead, võimaldades õppejõul hinnata tudengi tööd tõhusamalt.

Loodud rakendus aitab vähendada senise kirjaliku kontrolltöö koostamise ja hindamise ajakulu ning loob aluse hindamisprotsessi osaliseks või täielikuks automatiseerimiseks. Kasutajaliides järgib varasemate sarnaste tööde struktuuri, mis toetab programmi mõistetavust ning loob eeldused rakenduse hilisemaks integreerimiseks DeepMOOC e-õppekeskkonda. Visuaalne ühtlus ja läbimängija loogiline ülesehitus võimaldavad rakendust kasutada nii õppimiseks kui ka hindamiseks suurel kursusel.

Töö tulemusel loodi toimiv lahendus, mille põhjal saab arendada täiendavaid läbimängijaid teistele algoritmidele. Edasisteks võimalusteks nähakse funktsionaalsuse laiendamist, õpiosade lisamist ning automaatse hindamise täiendamist, et toetada paremini digitaalset õppeprotsessi.

Viidatud kirjandus

[1] Renno Sepp. Sisendite genereerimine puude ja kuhjade algoritmidel Tartu Ülikooli arvutiteaduse instituudi bakalaureuse lõputöö, 2024. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=80109 (13.05.2025)

[2] D. Galles. Data Structure Visualizations, University of San Francisco. https://www.cs.usfca.edu/~galles/visualization/Algorithms.html. (11.05.2025)

[3] S. Halim, VisuAlgo, National University of Singapore. https://visualgo.net/en. (11.05.2025)

[4] Märt Tender. DeepMOOC platvormile tagarakenduse arendamine Tartu Ülikooli arvutiteaduse instituudi bakalaureuse lõputöö, 2023. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=77057 (06.05.2025)

[5] Erik Presnov. Graafialgoritmide läbimängija ja hindaja Tartu Ülikooli arvutiteaduse instituudi bakalaureuse lõputöö, 2024. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=79681 (06.05.2025)

[6] Jüri Kiho. Algoritmid ja andmestruktuurid. Tartu Ülikooli Kirjastus 2003. https://dspace.ut.ee/server/api/core/bitstreams/9b9bf761-f89d-4eed-aee0-ebf0aedaa721/content (13.05.2025)

[7] Tartu Ülikooli aine „Algoritmid ja andmestruktuurid" (LTAT.03.005) Moodles olevad materjalid. (13.05.2025). URL: https://moodle.ut.ee/course/view.php?id=182.

Lisad

1. Lähtekood

Algoritmide läbimängija lähtekood on leitav Github repositooriumis aadressil

https://github.com/markusmi1/BST-AVL-Heap-Algorithm-Simulator

2. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Markus Michelis, annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose Kahendpuu- ja kuhjaalgoritmide läbimängija ja hindaja, mille juhendajad on Ahti Põder ja Kristo Väljako, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

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 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.

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

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

Markus Michelis

15.05.2025