TARTU ÜLIKOOL
Loodus- ja täppisteaduste valdkond
Arvutiteaduse instituut
Informaatika õppekava

Renno Sepp

Sisendite genereerimine puude ja kuhjade algoritmidele

Bakalaureusetöö (9EAP)

Juhendajad: Ahti Põder, PhD
Tõnis Hendrik Hlebnikov

Tartu 2024

Sisendite genereerimine puude ja kuhjade algoritmidele

Lühikokkuvõte:
Antud töö eesmärk on luua programm, mis abistaks aines "Algoritmid ja andmestruktuurid" sisendite genereerimist. Programm suudab koostada sisendeid puu- ja kuhjaalgoritmidele. Sisendite genereerimisel on võimalik määrata soovitud sisendi lahendamise keerukuse tase ning andmestruktuuri iseloomustavaid parameetreid. Programmi abil on võimalik õppejõududel efektiivselt genereerida suures koguses ühtlase raskustasemega sisendeid, mis ei kordu ning sobivad nii harjutamiseks kui ka teadmiste kontrollimiseks. Täpsemalt on programmiga võimalik luua sisendeid kahendotsimispuude, AVL-puude ja kuhjade algoritmidele.
Võtmesõnad:
Algoritmid ja andmestruktuurid, puud, kahendotsimispuud, AVL-puud, kuhjad, õpitarkvara
CERCS: P175 Informaatika, süsteemiteooria

Input generation for tree and heap algorithms

Abstract:

The main goal of this thesis work is to create a program that assists in generating inputs for the course “Algorithms and Data Structures”. The program is capable of composing inputs for tree and heap algorithms. During the generation of inputs, it is possible to specify the desired complexity level of the input solution and the parameters characterizing the data structure. The program enables instructors to efficiently generate a large amount of inputs with a consistent level of difficulty, which are non-repeating and suitable for both practice and knowledge testing. Specifically, the program can create inputs for binary search tree, AVL-tree, and heap algorithms.

Keywords:

Algorithms and Data Structures, trees, binary search trees, AVL-trees, heaps, educational software

CERCS: P175 I Informatics, systems theory

Sisukord

Sissejuhatus

Mitmed Tartu Ülikooli loodus- ja täppisteaduste valdkonna õppekavad hõlmavad kursuseid, mis keskenduvad erinevate algoritmide mõistmisele ja rakendamisele. Üheks selliseks kursuseks Tartu Ülikoolis on „Algoritmid ja andmestruktuurid“, mille eesmärgiks on süvendada tudengite programmeerimisalast algoritmilist mõtlemist ning arendada praktilisi oskusi algoritmide rakendamiseks [1]. Hetkel puudub selles õppeaines tõhus vahend algoritmide sisendülesannete automaatseks genereerimiseks, mis raskendab ülesannete variatsiooni ja sisendülesannete ühtlast raskustaseme tagamist.

Selle bakalaureusetöö peamine eesmärk on optimeerida puude ja kuhjade algoritmidele suunatud sisendülesannete koostamist nimetatud kursuse jaoks. Töö tulemusena loodud programm võimaldab automaatselt genereerida erineva keerukustasemega sisendeid, mis lihtsustab õppejõudude tööd üliõpilastele ülesannete koostamisel ja hindamisel. Lisaks ülesannete loomisele on programmiga võimalik tagada ühtlane sisendite keerukus.

Bakalaureusetöö tulemusena loodud Java programm võimaldab sisendeid koostada kuuele algoritmile: järjendi lugemine kahendotsimispuuks, kahendotsimispuust elementide eemaldamine, AVL-puusse elementide lisamine ning puust elementide eemaldamine, järjendi kuhjastamine ja järjendi sorteerimine kuhjameetodiga. Programm sisaldab lihtsat graafilist kasutajaliidest, mis tagab kiire ja efektiivse kasutajakogemuse.

Töö on jaotatud kolmeks peatükiks. Esimene peatükk võtab üldiselt kokku töö eesmärgi ning kasutatud tehnoloogia ja metoodikad. Teises peatükis on välja toodud printsiibid, mille järgi sisendülesannete genereerimise algoritme koostatakse, mis on jaotatud kolmeks osaks: kahendotsimispuud, AVL-puud ja kuhjad. Igas osas selgitatakse algoritmide eripärasi ning sisendi genereerimise algoritmi koostamise protsessi. Kolmandas peatükis on esitatud kasutusjuhend, mis annab ülevaate programmi tööst ning selgitab, kuidas seda kasutada.

1 Teoreetiline lahenduse ülevaade

Peatükis selgitatakse lõputöö eesmärki ning tutvustatakse selle raames välja töötatud algoritmide praktilist väärtust. Käsitletakse, miks just need algoritmid osutusid valikuks, ja antakse ülevaade kasutatud metoodikatest ning tehnoloogiatest, mis aitasid eesmärki saavutada.

1.1 Töö eesmärk

Käesoleva lõputöö peamine eesmärk on välja töötada algoritmid, mis genereerivad sisendülesandeid Tartu Ülikooli kursusele “Algoritmid ja andmestruktuurid”. Selle töö raames loodud algoritmid keskenduvad kahendotsimispuudele, AVL-puudele ning kuhjadele. Loodavad algoritmid võimaldavad kasutajal genereerida soovitud hulgal sisendeid, võimaldades määrata sisendile omadusi, nagu elementide arv, sisendi lahendamise keerukus, eemaldatavate või lisatavate elementide arv. Samuti on oluline, et genereeritavad sisendid oleks erinevad, kuid säilitaksid samad seatud omadused. Selliste algoritmide abil on võimalik luua suures koguses erinevaid sisendülesandeid, mille lahendamiseks peab näitama võrdväärseid oskusi. Need algoritmid täiendavad nii õppematerjalide kui ka teadmiste kontrolli mitmekesisust. Lisaks saab algoritmide kasutamine vähendada kursuse raames õppekorralduse koormust, pakkudes automaatset ja efektiivset viisi sisendite genereerimiseks.

1.2 Valitud algoritmid

See töö keskendub kahendotsimispuude, AVL-puude ja kuhjade algoritmide sisendite genereerimisele. Käesolevas lõputöös on iga teema puhul hoolikalt kaalutud "Algoritmid ja Andmestruktuurid" kursuse raames käsitletud algoritme, et valida need, mille sisendi genereerimine on manuaalselt keerukas, kuid mis on õppeaines laialdaselt kasutusel. Välistatud on algoritmid, mis on mõne käsitletud algoritmi osaülesanded, näiteks kahendotsimispuus elemendi leidmine, mis on vajalik oskus ka kahendotsimispuust elemendi eemaldamisel.

1.3 Kasutatud metoodikad

Sisendite genereerimise algoritme arendades võeti arvesse nii lahendusalgoritmide eripärasid kui ka andmestruktuuride omadusi. Oluliseks teguriks sisendi genereerimisel on raskusparameeter, mis võimaldab määratleda sisendi lahendamise keerukust. Esialgu analüüsiti lahendusalgoritmide samme, et kindlaks teha, mis konkreetselt suurendab lahendamiskäigu keerukust iga algoritmi puhul. Seejärel uuriti andmestruktuuride mustreid, et leida sobivad lähenemised sisendite genereerimiseks. Neid lähenemisi võrreldi ajalise keerukuse, sisendite variatsiooni ja selguse alusel. Valituks osutus kõige optimaalsem lähenemine, mis toetas lõputöö eesmärkide saavutamist. Lõpuks kontrolliti genereeritud sisendite vastavust etteantud omadustele ja raskusparameetrile.

1.4 Kasutatud tehnoloogiad

Algoritmid programmeeriti Java keeles, peamiselt seetõttu, et seda on kasutatud varasemates sarnast eesmärki täitvates lõputöödes, mis võimaldab tulevikus potentsiaalselt ühendada need tööd ühtseks süsteemiks [2], [3], [4]. Lisaks on Java ka „Algoritmid ja Andmestruktuurid“ kursuse raames eelistatud programmeerimiskeel, mis toetab aine raames kasutatud tehnoloogia ühtsust. Graafilise kasutajaliidese loomiseks kasutati Swingi, kuna see pakub head ühilduvust kasutatud integreeritud arenduskeskkonnaga. Sellise lihtsa graafilise kasutajaliidese raamistiku kasutamine võimaldas keskenduda töö raames peamiselt algoritmide väljatöötamisele.

2 Algoritmide sisendite genereerimine

Peatükis kirjeldatakse algoritmide loomise protsessi. Kõigepealt seletatakse lahti peamised põhimõtted, mille alusel genereerimisalgoritme loodi ning seejärel on iga loodud algoritmi kohta alampeatükk. Igas alampeatükis tehakse esmalt ülevaade lahendusalgoritmist ning seejärel võetakse kokku töö käigus loodud algoritmi loomisprotsess ja põhiideed. Kõik kirjeldatud algoritmid on kokku koondatud ühte GitHub'i hoidlasse[1]. Algoritmide kirjeldamise aluseks on võetud "Algoritmid ja Andmestruktuurid" Moodle'i materjalid [5] ning algoritmide töökäik on selgitatud J. Kiho õpiku näidete põhjal [6].

2.1 Algoritmide koostamise põhimõtted

Iga genereerimisalgoritm on välja töötatud lähtudes vastavast lahendusalgoritmist. Lahendusalgoritmi töökäigu tingimuste alusel on välja töötatud raskusparameetrid, mis peegeldavad nõutavat oskustaset sisendülesande lahendamiseks. Lisaks raskusparameetrile sisaldavad iga algoritm sisendväärtused ka täiendavaid parameetreid, mis kirjeldavad genereeritavate sisendite struktuuri. Sellisteks parameetriteks võivad olla näiteks elementide koguarv, eemaldatavate või lisatavate elementide hulk. Andmestruktuuri eripärade ja raskusparameetrite põhjal on loodud algoritmid, mis suudavad süstemaatiliselt genereerida soovitud hulga erinevaid, kuid ühtlase raskustasemega sisendülesandeid. Ülesanne loetakse erinevaks isegi juhul, kui lahenduse sammud on samad, kuid elementide väärtused erinevad. Selline lähenemine võimaldab genereerida väga palju erinevaid järjendeid, säilitades nende raskustaseme. Oluline on ka, et programmeeritud algoritm on suuteline genereerima mõistliku ajaga soovitud koguse kursuse raames kasutusel olevate omadustega sisendeid. See tähendab, et algoritmide koostamisel oli oluliseks eesmärgiks, et programm genereerib mõistliku ajaga kuni tuhat 20-elemendilist sisendit.

2.2 Kahendotsimispuud

See peatükk pakub põhjalikku ülevaadet kahest kahendotsimispuu algoritmist. Esialgu tutvustatakse kahendotsimispuude peamisi omadusi. Seejärel on lahti seletatud mõlema algoritmi töökäik kasutades pseudokoodi ja illustreerivaid jooniseid. Peale algoritmide selgitust käsitletakse nende algoritmide sisendeid genereerivaid meetodeid. Mõlema algoritmi sisendite genereerimise kohta on selgitus selle loogikast ja koostamisprotsessist, mis võimaldab lugejal süvitsi mõista algoritmide ülesehitust ja funktsioone.

2.2.1 Mõisted

Kahendotsimispuu on andmestruktuur, mis organiseerib andmed hierarhiliselt, võimaldades kiireid otsingu-, sisestus- ja kustutamisoperatsioone. Iga puu tipp sisaldab võtit. Kahendotsimispuu põhireegli kohaselt peab igas tipus oleva elemendi väärtus olema suurem-võrdne kõigist tema vasakus alampuus olevatest elementidest ja väiksem-võrdne kõigist tema paremas alampuus olevatest elementidest. See unikaalne omadus tagab elemendi kiire leidmise, sest igal sammul välistatakse pooled potentsiaalsed asukohad, järgides lihtsat suurem-või-väiksem otsinguloogikat.

[1] Programmi lähtekood https://github.com/RennoSepp/sisendi_algoritmid.git

2.2.2 Järjendi lugemine kahendotsimispuuks

Lahendusalgoritm algab esimese elemendi valimisega järjendist, mis määratakse puu juurtipuks. Seejärel lisatakse iga järgnev element puusse vastavalt kahendotsimispuu põhiprintsiibile: kui lisatav element on väiksem kui vaadeldava tipu kirje, liigutakse tipu vasakusse harusse, kui aga suurem, liigutakse paremasse harusse. Kui jõutakse tühja harusse, lisatakse element sinna. Protsess jätkub kõigi järjendi elementidega.

```
Antud:     n – järjendi pikkus, mis tuleb lugeda kahendotsimispuuks
Tagastab:  tipp – kahendotsimispuuks loetud järjendi juurtipu

tipp = null
if järjend is empty:
    return tipp

for element in järjend:
    uus_tipp.info = element
    if tipp == null:
        tipp = uus_tipp
    else:
        while true:
            if uus_tipp.info < tipp.info:
                if tipp.v == null:
                    tipp.v = uus_tipp
                    break
                else:
                    tipp = tipp.v
            else:
                if tipp.p == null:
                    tipp.p = uus_tipp
                    break
                else:
                    tipp = tipp.p
```

Sellise algoritmi sisendite genereerimise algoritmi raskusparameeter on kõikideks lisamisteks vajaminevate võrdluste summa. Raskusparameetrile lisaks saab sisendile määrata ka elementide arvu, mis aitab kaasa sarnaste sisendite loomisele. Seejärel genereerib algoritm soovitud hulga erinevaid selliseid järjendeid, kus elementide sisselugemisel kahendotsimispuuks tehakse täpselt sisendis määratud arv võrdlusi.

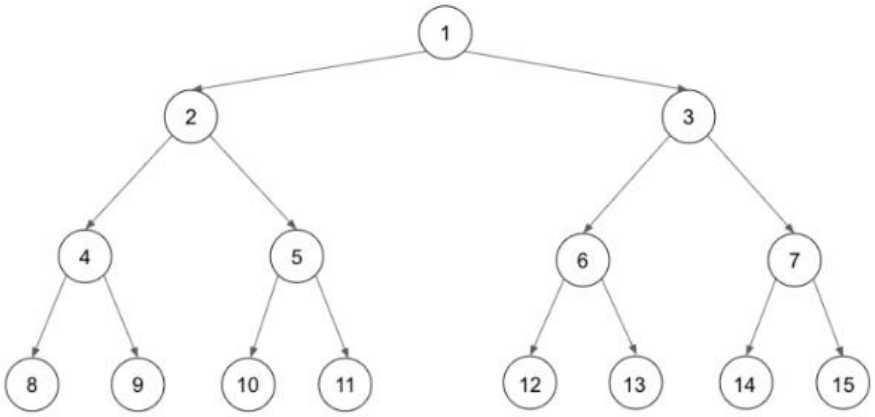
Kuna järjendi elementide arv ning määratav võrdluste arv sisendis on tihedalt seotud, on nende vahel leitud seos, et juhendada programmi kasutajat valima kohast võrdluste arvu peale elementide arvu määramist.

Algoritmi arendamisel kaaluti sisendparameetrina ka puu kõrguse lisamist, kuid algoritmi koostamisel osutus, et parameeter ei mõjutanud ülesande keerukust ning kitsendas erinevate võimalike genereeritavate sisendite arvu. Lisaks kaaluti algoritmi koostamisel n-elemendiliste mudelite loomist. See lähenemine oleks katsetanud kõiki erinevaid järjestusi järjendi elementidega 1,2,…,n. Nii oleks saanud mudelites arvud 1,2,…,n asendada suvaliste arvudega, mis vastaksid samale suurusjärjestusele. Hoolimata, et sellised mudelid saab kuni teatud järjendi pikkustele ette defineerida, on lähenemine siiski faktoriaalse ajalise keerukusega ning alternatiivne lähenemine osutus efektiivsemaks.

Programmi implementeeritud algoritmi loomine on fokuseeritud võrdluste arvule, mida iga järgmise elemendi lisamisega on võimalik teha. Ehk esimese elemendi lisamisel tuleb teha 0 võrdlust. Teise elemendi lisamisel tuleb teha 1 võrdlus. Kolmanda elemendi lisamisel on võimalik teha kas 1 või 2 võrdlust. Nii on võimalik algoritmiliselt meeles pidada võimalikku võrdluste arvu ning nende seast juhuslikult valida positsioone nii, et lähenetakse sisendis määratud võrdluste arvule. Juhul, kui võrdluste arv või järjendi elementide arv ületab sisendis seatud piire, saab algoritm itereerida tagasi ning valida vastavalt uue positsiooniga arvu.

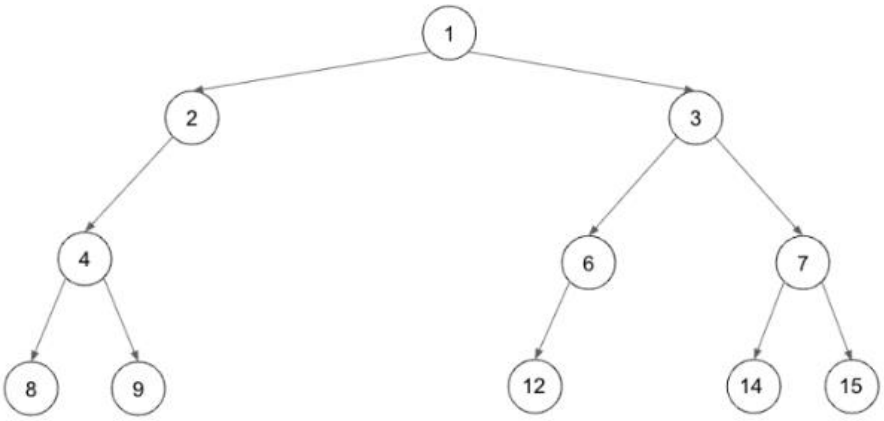
Kuna algoritm on koostatud nii, et valminud järjend on ehitatud täiskahendotsimispuu skeletile (joonis 1), on võimalik defineerida järjendis olevate elementide suuruse järjestus ning see asendada juhuslike arvudega samas suuruse järjestuses.

Täis-kahendpuu indeksite skelett



Kahendotsimispuu elementide suuruse järjestus: [8, 4, 9, 2, 10, 5, 11, 1, 12, 6, 13, 3, 14, 7, 15]

Juhuslikult koostatud kahendotsimispuu lähtudes täis-kahendotsimispuu indeksitest



Juhuslikult genereeritud kahendotsimispuu elementide suuruse järjestus: [8, 4, 9, 2, 1, 12, 6, 3, 14, 7, 15]

Joonis 1. Genereeritud kahendotsimispuu elementide suuruse järjestuse loogika

```
Antud:        n – elementide arv järjendis
              raskusparameeter – kahendotsimispuu koostamiseks tehtav
              võrdluste arv

Tagastab:     järjendi, mille kahendotsimispuuks lugemisel tuleb teha
              täpselt raskusparameetri jagu võrdlusi

//Alustame nii, et indeksite järjendis on vaid positsioon 0 (juurtipu
positsioon)
kahendotsimispuu_tühjad_indeksid = [0]
tagastatav_järjend = []
tagastatava_järjendi_võrdlused = 0 //raskusparameetri hindaja

//Järjend koostatakse täiskahendotsimispuu indeksitest
def leia_tagastatav_järjend():
    if tagastatav_järjend.length == n and
    tagastatava_järjendi_võrdlused == raskusparameeter:
        return tagastatav_järjend
    //Kui üks omadustest on rikutud, lähme sammu võrra tagasi
    else if tagastatav_järjend.length > n or
    tagastatava_järjendi_võrdlused > raskusparameeter:
        return
    else:
        lisatav = juhuslik(kahendotsimispuu_tühjad_indeksid)
        tagastatav_järjend.add(lisatav)
        //Kuna lähtume täiskahendotsimispuu indeksitest, siis saame
lihtsalt leida elemendi kõrguse puus
        tagastatava_järjendi_võrdlused += kõrgus(lisatav)
        //Lisame uute tühjade positsioonide indeksid lähtudes
        täiskahendotsimispuu positsioonidest
        kahendotsimispuu_tühjad_indeksid.add(indeks(lisatav)*2+1,
        indeks(lisatav)*2+2)
        //Protsess jätkub, kuni järjend on leitud või kõik võimalused
on läbi käidud
        leia_tagastatav_järjend()


//tagastatav_järjendi leidmisel teeb programm suuruse järjestuse
lähtudes täiskahendotsimispuu positsioonidest.
//Suuruste järjestusest lähtuvalt genereeritakse n suvalist arvu ning
määratakse vastavatele positsioonidele.
```

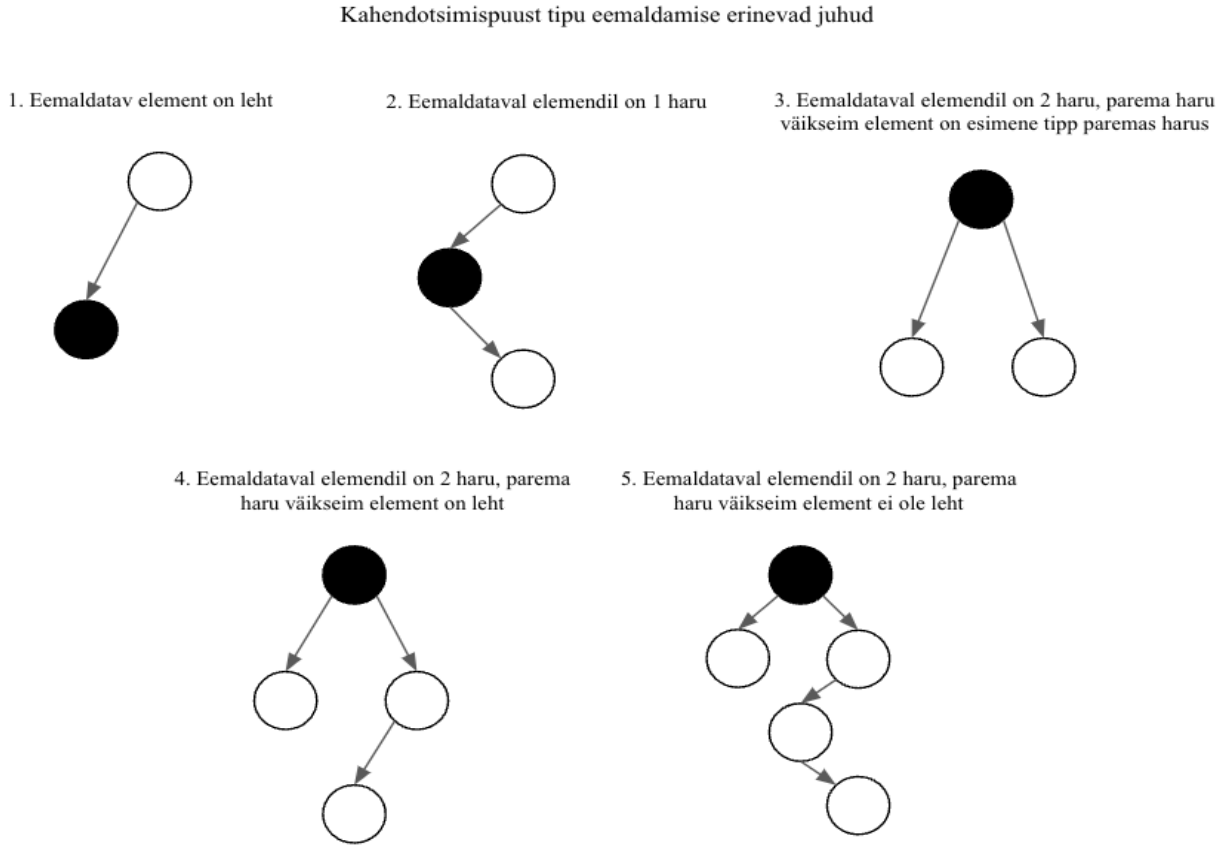
2.2.3 Elementide eemaldamine kahendotsimispuust

Elemendi eemaldamiseks kahendotsimispuust tuleb esmalt leida puust eemaldatava kirjega tipp. Kui tipp on leht, saab tipu eemaldada ilma asenduseta. Kui tipul on üks haru, tuleb tipu eemaldamisel asendada tipp oma haruga. Kui tipul on kaks haru, tuleb leida paremas harus väikseima kirjega tipp, eemaldada see ning asendada eemaldatava tipu kirje leitud väikseima elemendi kirjega. Sellisel juhul on võimalik, et tekib eemaldamiste ahel.

```
Antud:      eemaldatav – eemaldatav kirje
            tipp – kahendotsimispuu juurtipp

Tagastab:   tipp – kahendotsimispuu juurtipp, millest on
            eemaldatud element

vanem = null
vaadeldav_tipp = tipp
while vaadeldav_tipp != null and vaadeldav_tipp.info != eemaldatav:
    vanem = vaadeldav_tipp
    if eemaldatav < vaadeldav_tipp.info:
        vaadeldav_tipp = vaadeldav_tipp.v
    else:
        vaadeldav_tipp = vaadeldav_tipp.p

if vaadeldav_tipp == null:
    return tipp
if vaadeldav_tipp.v == null:
    if vanem == null:
        tipp = vaadeldav_tipp.p
    else if vaadeldav_tipp == vanem.v:
        vanem.v = vaadeldav_tipp.p
    else:
        vanem.p = vaadeldav_tipp.p
else if vaadeldav_tipp.p == null:
    if vanem == null:
        tipp = vaadeldav_tipp.v
    else if vaadeldav_tipp == vanem.v:
        vanem.v = vaadeldav_tipp.v
    else:
        vanem.p = vaadeldav_tipp.v
else:
    muutuja_vanem = vaadeldav_tipp
    muutuja = vaadeldav_tipp.p
    while muutuja.v != null:
```

```
        muutuja_vanem = muutuja
        muutuja = muutuja_vanem.v
    vaadeldav_tipp.info = muutuja.info
    if muutuja == muutuja_vanem.v:
        muutuja_vanem.v = muutuja.p
    else:
        muutuja_vanem.p = muutuja.p
    return tipp
```

Kuna tipu eemaldamise keerukus sõltub tema alluvate struktuurist, siis raskusparameeter peab sõltuma kirjest, mida eemaldatakse. Sisendülesande loomise algoritm saab sisendiks puu tippude arvu, eemaldamist vajavate elementide arvu ning raskusparameetri, mis on määratletud viie võimaliku eemaldamise stsenaariumi põhjal (joonis 2).



Joonis 2. Kahendotsimispuust elemendi eemaldamise 5 erinevat võimalust

Algoritmi koostamise fookus on luua määratud elementide arvuga järjend, kus esineksid eelmainitud omadustega tipud, et nende eemaldamisel määratud järjekorras tuleks eemaldada vastavate omadustega tippe nii, et eemaldamiste keerukuste kogusumma on võrdne määratud raskusparameetriga.

Kuna töö fookuseks on luua ühetaolise raskusega sisendeid läbimänguülesannete lahendamiseks kursuse “Algoritmid ja Andmestruktuurid” raames, on omaduste määramisele seatud piirid, et

tagada kiire ja efektiivne lahendus, mis tagastaks soovitud omadustega järjendid. Algoritmi sisendparameetritele on määratud kaks täpsustust: järjendi elementide arv peab olema vähemalt 5 ning sisendis määratav eemaldatavate elementide arv saab olla maksimaalselt 1/3 elementide arvust.

Algoritm genereerib etteantud tippude arvuga järjendi ning määrab igale tipule tema tüübi vahemikust 1-5. Vastavalt juhuslikule kombinatsioonile arvudest vahemikus 1-5, mille summa on võrdne raskusparameetriga, eemaldatakse järjendist vastava 1-5 omadusega tipp. Seejärel kalkuleeritakse tippudele uuesti tüübid vastavalt ühest viieni. Seda protsessi korratakse kuni määratud arv elementide eemaldamisega on saavutatud täpselt määratud raskusaste. Seejärel tagastab programm algse järjendi ning eemaldatavate elementide järjendi.

```
Antud:   n – elementide arv järjendis
         eemaldatavaid – mitu elementi järjendist eemaldatakse
         raskusparameeter – eemaldamiste raskustüüpide summa

Tagastab: eemaldatavate elementide järjend, kahendotsimispuu

//Esimese sammuna leitakse kombinatsioonid, kuidas saab eemaldatavate
elementidega kokku raskusparameetri. Iga eemaldatava elemendi
raskusväärtus saab olla vahemikus 1-5 sõltuvalt eemaldatava elemendi
positsioonist puus.
kombinatsioonid = leia_kombinatsioonid(eemaldatavad,
raskusparameeter)

def järjend_eemaldatavatega(kombinatsioonid):
    //Kahendotsimispuuks luuakse suvaline järjend n
    elemendiga.
    kahendotsimispuu = juhuslik_kahendotsimispuu(n)
    eemaldatavad_raskused =
    vali_juhuslik_kombinatsioon(kombinatsioonid)
    while eemaldatavad_raskused.length != 0:
        //Algoritm väärtustab iga puu tipu väärtusega 1-5 vastavalt
        //1 – leht
        //2 – ühe alluvaga tipp
        //3 – kahe alluvaga tipp (paremal alluval ei ole vasakut
        haru)
        //4 – kahe alluvaga tipp (parema alluva vasaku haru väikseim
        element on leht)
        //5 – kahe alluvaga tipp (parema alluva vasaku haru väikseim
element ei ole leht))
        järjendi_väärtustus = väärtusta_tipud(kahendotsimispuu)

        //Vali suvaline element kombinatsioonist ning eemalda vastava
väärtustusega tipp järjendist vastavalt samalt positsioonilt
```

```
        eemaldatav = eemaldatavad_raskused.pop(juhuslik)
        järjend.remove(järjendi_väärtustused[eemaldatav])
        eemaldatavad_elemendid.add(eemaldatud_element)

//Kuna tõenäosus on suur, et iga väärtustusega element esineb
järjendis on tõenäoline iga tsükliga leida sisend. Kui aga
sisendparameetrid on rikutud, alustatakse tsüklit uue eemaldatavate
elementide väärtustusega uuesti. Kui sisendit kolme erineva
kombinatsiooniga ei leita, on juhuslikult sattunud väga äärmuslik puu
ning tsüklit alustatakse uuesti uue puuga.
```

2.3 AVL-puud

Peatükis seletatakse lahti AVL-puu definitsioon ja selle andmestruktuuri peamised omadused. Seejärel keskendutakse kahele peamisele operatsioonile: elementide lisamine ja eemaldamine AVL-puus. Algul seletatakse lahti operatsioonide toimimise põhimõtted ning seejärel tutvustatakse algoritmi, mis loob nende operatsioonide jaoks sobivaid sisendeid.

2.3.1 Mõisted

AVL-puu on kahendotsimispuu, kus iga tipu vasaku ja parema haru kõrguste vahe on maksimaalselt üks. See rangelt järgitud tasakaalustatuse nõue tagab, et puu kõrgus hoitakse logaritmilises seoses sisalduvate tippude arvuga, mis võimaldab kiireid otsingu-, lisamis- ja eemaldamisoperatsioone. AVL-puud kasutavad elementide lisamisel ja eemaldamisel pöördeid, et säilitada oma tasakaalustatud struktuur.

2.3.2 Elementide lisamine AVL-puusse

Elementide lisamine AVL-puusse toimib sarnaselt elemendi lisamisele kahendotsimispuusse. Erisus tekib peale kirje lisamist. Peale elemendi lisamist on oluline kontrollida puu tasakaalu iga lisatud tipu vanema tipu juures, liikudes puus üles kuni juurtipuni. Kui leitakse tipp, mis ei ole tasakaalus, tehakse vastavas alampuus pöördeid nii, et taastada AVL-puule omane tasakaal. Pöördeid on nelja tüüpi: vasakpööre, parempööre, vasakparempööre ja paremvasakpööre. Iga selline rotatsioon mitte ainult ei taasta tasakaalu, vaid säilitab ka kahendotsimispuu omadused, tagades, et kõik tipud vasakul on väiksemad ja paremal suuremad kui juurtipp.

```
Antud:    kirje – lisatav element
          tipp – AVL puu juurtipp
Tagastab: tipp – AVL puu juurtipp, millele on lisatud element

def lisa(tipp, kirje):
    if tipp == null:
        uus_tipp = kirje
        return uus_tipp
```

```
    if kirje < tipp.info:
        tipp.v = lisa(tipp.v, kirje)
    else:
        tipp.p = lisa(tipp.p, kirje)

    uuenda tippu:
        tipp.kõrgus = 1 + max(kõrgus(tipp.v), kõrgus(tipp.p))
        tipp.tasakaal = kõrgus(tipp.v) - kõrgus(tipp.p)

    if tipp.tasakaal > 1 and kirje < tipp.v.info:
        return paremPööre(tipp)
    if tipp.tasakaal < -1 and kirje > tipp.p.info:
        return vasakPööre(tipp)
    if tipp.tasakaal > 1 and kirje > tipp.v.info:
        tipp.v = vasakPööre(tipp.v)
        return ParemPööre(tipp)
    if tipp.tasakaal < -1 and kirje < tipp.p.info:
        tipp.p = paremPööre(tipp.p)
        return vasakPööre(tipp)
    return tipp
```

Sellise algoritmi keerukus sõltub tehtavatest pööretest. Kuna iga pöörde puhul tuleb hoolikalt jälgida pöördest mõjutatud tippude alluvaid, mis on tihti väga erinevad, on raskusparameetriks seatud lihtsalt pöörete arv. Elementide arvu suurendamisega saab tagada ühtlase sisendite keerukuse. Antud lähenemine annab võimaluse raskusparameetri seada maksimaalselt lisatavate elementide arvuks. Sellisel juhul tuleb teha pööre iga lisatava elemendi korral.

Sellest tulenevalt on sisendparameetriteks elementide arv ning raskusparameeter. Algoritm genereerib suvalise määratud pikkusega järjendi ning tasakaalustab selle. Seejärel hindab algoritm iga tipu tasakaalu ning lisab tasakaaluga 1 ja -1 tipud ühte järjendisse. Kuna iga tipu lisamine peab tekitama pöörde, valitakse tekitatud 1 ja –1 koosnevast järjendist suvaline tipp, ning liigutakse tasakaalupunkti otsides lehte. Sellele lehele tipu lisamisel on kindel, et tekib pööre. Algoritm lisab puule juhuslikult kirje, mille väärtus on leitud lehe kirje +1 või leitud lehe kirje –1. Nii korratakse protsessi, kuni määratud arv kirjeid on lisatud nii, et raskusparameeter on rahuldatud.

```
Antud:   n – tippude arv
         raskusparameeter – mitu pööret tuleb lisamistega teha

Tagastab: AVL-puu ja lisatavad elemendid

def leia_lisatavad_elemendid():
        //Genereeri juhuslik n-elemendiline AVL-puu
        AVLpuu = genereeri_juhuslik_AVL_puu(n)
```

```
        for i in range(raskusparameeter):
            //Funktsioon käib puu läbi keskjärjestuses ning märgib
            kõik tipud, mille tasakaal on 1 või -1 ühte järjendisse
            tasakaalutud = leia_tasakaalutud(AVLpuu)

            //Valime tasakaalude seast suvalise tipu ning
            leiame sellest tipust alla liikudes lehe, kuhu
            elementi lisades läheb puu tasakaalust välja
            tasakaalutuLeht =
            leia_tasakaalutu_leht(juhuslik(tasakaalud))
            //Valime juhuslikult, kas element lisatakse leitud lehe
            vasakusse või paremasse harusse, väärtustades lisatava
            elemendi kas +1 või -1-ga
            lisatav_element = juhuslik(tasakaalutu_leht +1,
            tasakaalutu_leht -1)
```

2.3.2 AVL-puust elemendi eemaldamine

AVL-puust elemendi eemaldamise esimene samm on sarnane kahendotsimispuust elemendi eemaldamisele. Erisus tekib peale eemaldamist järgnevas tasakaalukontrollis, kus alates eemaldamiskohast kuni juureni kontrollitakse iga tipu tasakaalustatust. Kui mõne tipu tasakaal on suurem kui 1 või väiksem kui –1, siis tehakse vajalikud pöörded, et taastada tasakaal.

```
Antud:      eemaldatav – eemaldatav element
            tipp – AVL puu juurtipp
Tagastab:   tipp – AVL puu juurtipp, millest on eemaldatud
            element

def eemalda(tipp, eemaldatav):
    if tipp == null:
        return tipp

    if eemaldatav < tipp.info:
        tipp.v = eemalda(tipp.v, eemaldatav)
    else if eemaldatav > tipp.info:
        tipp.p = eemalda(tipp.p, eemaldatav)
    else:
        if tipp.v == null or tipp.p == null:
            if tipp.v != null:
                ajutine = tipp.v
            else:
                ajutine = tipp.p
            if ajutine == null:
```

```
                ajutine = tipp
                tipp = null
            else:
                tipp = ajutine
        else:
            ajutine = min(tipp.p)
            tipp.info = ajutine.info
            tipp.p = eemalda(tipp.p, ajutine.info)
    if tipp == null:
        return tipp

    tipp.kõrgus = 1 + max(Kõrgus(tipp.v), Kõrgus(tipp.p))
    tasakaal = kõrgus(tipp.v) - kõrgus(tipp.p)

    if tasakaal > 1 and kõrgus(tipp.v.v) >= kõrgus(tipp.v.p):
        return paremPööre(tipp)
    if tasakaal > 1 and kõrgus(tipp.v.v) < kõrgus(tipp.v.p):
        tipp.v = vasakPööre(tipp.v)
        return paremPööre(tipp)
    if tasakaal < -1 and kõrgus(tipp.p.p) >= kõrgus(tipp.p.v):
        return vasakPööre(tipp)
    if tasakaal < -1 and kõrgus(tipp.p.p) < kõrgus(tipp.p.v):
        tipp.p = paremPööre(tipp.p)
        return vasakPööre(tipp)
    return tipp
```

AVL-puust elemendi eemaldamise sisendi genereerimisalgoritmi koostamise puhul ei võetud enam arvesse kahendotsimispuust elemendi eemaldamise keerukust, mis sai lahti seletatud ja lahendatud eelmises peatükis. Seega sarnaselt elemendi lisamisele AVL-puusse, on ka elemendi eemaldamise peamine keerukus säilitada puu tasakaalustatud olek. Erinevalt lisamisest, mis vajab puu tasakaalustamiseks ühte pööret, võib eemaldamine nõuda mitmeid pöördeid puu juurtipu poole liikudes, et säilitada AVL-puu omadused.

Sellele ideele tuginedes on sisendi genereerimise algoritmi sisendparameetriteks AVL-puu tippude arvu, eemaldatavate elementide arv ja raskusparameeter. Algoritmi loomisel on otstarbekas alustada juhuslikult koostatud AVL-puust, sest kindlate omaduste ja struktuuriga puu loomine piiraks sisendite mitmekesisust. Samuti on tõenäoline, et juhuslikult genereeritud puudes on suur hulk tippe, mille eemaldamine nõuab pöördeid, mis teeb algoritmi alustamise juhusliku AVL-puuga sobilikuks lähenemiseks.

Lähtudes AVL-puu struktuurist ja tasakaalujaotusest saab tegelikult mitmete elementide eemaldamise siduda ühe selle puu lehttipu eemaldamisega. Algoritm uurib kõiki puu lehttippe, et hinnata, mitu pööret on vajalik iga lehe eemaldamisel puu tasakaalu taastamiseks. Seejärel seob algoritm puu tipud lehtedega, mille kõrgus vastava elemendi eemaldamisel muutub. Lähtudes keerukusparameetrist määratakse, mitu pööret on järgmise elemendi eemaldamisel vajalik.

Elementide hulgast valitakse juhuslik tipp, mille eemaldamisel on vaja teha etteantud arv pöördeid. Element eemaldatakse ning protsessi korratakse, kuni kõik sisendparameetrid on täidetud.

```
Antud: n – tippude arv
       eemaldamisi – eemaldatavaid elemente kokku
       raskusparameeter – mitu pööret eemaldamistega kokku
       tehakse

Tagastab: AVL puu (järjendi kujul) ja järjendi eemaldatavaid
          elemente

def leia_eemaldatavad_elemendid:
    //Genereeri juhuslik n-elemendiline AVL-puu
    AVLpuu = genereeri_juhuslik_AVL_puu(n)
    while raskusparameeter > 0:
         //Funktsioon hindab iteratiivselt iga lehttipu
         eemaldamisel puu tasakaalu taastamiseks tarvilikku
         pöörete arvu
         lehed = leia_lehed(AVLpuu)
         eemaldamise_pöörete_arv = leia_pöörete_arv(lehed)
         //Kuna iga tipu eemaldamine on seotud ühe haru
         lühenemisega, mille saab teisendada ümber lehe
         eemaldamiseks, jaotame kõik järjendi elemendid
         vastavate lehttippude alla lähtudes kahendotsimispuu
         elemendi eemaldamise põhimõtetest
         eemaldamise_pöörete_arv =
         JaotaLehtede_harudesse(AVLpuu.tipud())
         //Valime juhuslikult, vastava pöörete arvuga lehe hulgast
         elemendi ning eemaldame selle
         eemaldatav_element = juhuslik(eemaldamise_pöörete_arv)
         raskusparameeter = raskusparameeter -
         eemaldamise_pöörete_arv[eemaldatav_element]
```

2.4 Kuhjad

Peatükk algab ülevaatega kuhja omadustest. Seejärel käsitletakse kahte olulist kuhjade algoritmi – järjendi kuhjastamine ning sorteerimine kuhjameetodil. Algul seletatakse lahti algoritmide tööpõhimõtted ning seejärel tehakse ülevaade sisendi genereerimise algoritmidest. Lisaks on mõlema algoritmi kohta esitatud pseudokood nii sisendülesannete koostamise kui ka lahendamise kohta.

2.4.1 Mõisted

Kuhjad on puupõhised andmestruktuurid, mida kasutatakse andmete prioritiseerimiseks, kiirendades juurdepääsu kõige olulisematele elementidele. Kuhjad jagunevad maksimum- ja

miinimum-kuhjadeks, kus maksimum-kuhjas on iga vanemelemendi väärtus suurem või võrdne ja miinimum-kuhjas väiksem või võrdne oma alluvate väärtustega. Selles töös on algoritmid loodud maksimum-kuhjade algoritmidele. Miinimumkuhjade algoritmide tööpõhimõte on sama.

2.4.2 Järjendi kuhjastamine

Kuhjastamise algoritmi eesmärk on vahetada järjendi elementide positsioone nii, et tagastatav andmestruktuur oleks kuhja omadustega. Kuhjastamine toimub alustades järjendi lõpust ja liikudes alguse poole. Iga elemendi puhul kontrollitakse, kas see vastab kuhja omadustele. Kui vanemelemendi väärtus on laste omast väiksem, vahetatakse elemendid. See protsess tagab, et kõik kuhja tipud vastavad maksimum-kuhja nõuetele, kus iga vanem on oma lastest suurem. Elementide vahetamine toimub rekursiivselt, kuni kõik elemendid vastavad kuhja omadustele.

```
Antud:    järjend
Tagastab: kuhjastatud järjend

def kuhjasta(järjend):
    n = järjend.length
    for i = n/2 - 1 to 0 step -1:
        korrastaKuhi(järjend, n, i)

    def korrasta_kuhi(järjend, n, i):
        suurim = i
        vasak = 2*i + 1
        parem = 2*i + 2

        if vasak < n and järjend[vasak] > järjend[suurim]:
            suurim = vasak
        if parem < n and järjend[parem] > järjend[suurim]:
            suurim = parem
        if suurim ≠ i:
            vaheta(järjend[i], järjend[suurim])
            korrasta_kuhi(järjend, n, suurim)
```

Selle algoritmi puhul tekib raskusparameeter kuhjastamisel tehtavate elementide vahetuste koguarvust. Lisaks saab sisendparameetriga määrata ka järjendi pikkuse. Elementide arvu põhjal on võimalik iga järjendi lõpust alguse poole vaadeldava elemendi kohta määrata positsioonile maksimaalne võimalik vahetuste arv. Maksimaalsetest vahetustest võetakse juhuslik osa nii, et vahetuste kogusumma oleks võrdne raskusparameetriga.

Järgmise sammuna töötab algoritm järjendi läbi tagantpoolt ettepoole. Kui osahulgal määratud väärtused näitavad, et vastaval positsioonil peaks toimuma vahetus, väärtustab algoritm juhuslikult ühe või mõlemad lapsed suuremaks vaadeldavast tipust. Vastavalt varasemalt määratud vahetuste arvule korratakse seda protsessi. Igal sammul tehakse ka vahetus kuhjale omasel viisil. Kui aga positsioonil vahetust toimuda ei tohi, väärtustab algoritm tipu suuremaks enda lastest. Seda protsessi jätkatakse kuniks jõutakse järjendi algusesse.

Seejärel määratakse elementide suuruse järjestus ning genereeritakse juhuslikud arvud, mis asendatakse vastavas suuruse järjestuses loodud järjendi elementidele. Kõige lõpuks viiakse elemendid tagasi oma algsetele positsioonidele ning tagastatakse järjend.

```
Antud: n – elementide arv
       raskusparameeter – mitu võrdlust tuleb kuhjastamisel teha

Tagastab: järjendi, mille kuhjastamiseks tuleb teha täpselt määratud
          arv võrdlusi

def kuhjastatavate_sisendite_genereerimine():
    //Arvutame välja mitu võrdlus on igale positsioonile võimalik
    maksimaalselt teha.
    max_võrdlusi = arvutame_max_arvu_võrdusi_igale_positsioonile(n)
    //Arvutame iga positsiooni maksimaalsetest võrdlustest osahulga
    juhuslikult nii, et kogu osahulkade summa oleks võrdne
    raskusparameetriga.
    genereeritavad_võrdlused = raskusparameetri_osa(max_võrdlusi)

    //Ehitame vastava kuhja sisendi
    kuhi = koostamePaarid(n*[0,i])  //i on positsiooni indeks, algne
elementide väärtustus on 0
    for i in range(n):
    //Fikseerime ära vaadeldava positsiooni vasaku ja parema alluva
        vasak = 2 * i + 1
        parem = 2 * i + 2
        if genereeritavad_võrdlused[i] == 0:
        //Kui vahetusi teha ei tule, määrame vaadeldava elemendi
        suuremaks oma alluvatest
            kuhi[indeks][0] = max(kuhi.vasak, kuhi.parem)+1
        else:
            for j in range(genereeritavad_võrdlused[i]):
                //Kui tuleb teha vahetus, valime suvaliselt, kumb
                alluvatest on suurem (või mõlemad) ning väärtustame
                need vastavalt
                suurem_haru = juhuslik(3)
                //Funktsioon muudab vastavalt juhuslikult
                genereeritud harule kirjete väärtused
                ning vahetab nende positsioonid
                suurenda_alluvad_ja_vaheta_kohad(suurem_haru)

//Kui kuhi on valmis, sorteerime elemendid võtmete järgi suuruse
järjestusse
```

```
//Genereerime suvalised n numbrit ja lisame need vastavalt suuruse
järjestuses positsioonidele
//Kuna hoidsime ka indekseid alles, viime elemendid õigetele
indeksitele ja tagastame järjendi
```

2.4.3 Sorteerimine kuhjameetodil

Kuhjasorteerimine on sorteerimismeetod, mis kasutab kuhja andmestruktuuri elementide järjestamiseks. Alguses teisendatakse järjend maksimum-kuhjaks, mis asetab suurima elemendi järjendi algusesse. Seejärel vahetatakse suurim element järjendi viimasega, asetades selle õigesse järjekorda lõppu ja jäetakse edaspidi puutumata. Ülejäänud järjendi jaoks taastatakse kuhja omadused, välja arvatud juba paika pandud suurim element. See protsess kordub, vähendades kuhja suurust ühe elemendi võrra iga kord, kuni kõik elemendid on sorteeritud ja paigutatud järjendi lõppu. Kuhja sorteerimise efektiivsus tuleneb suurte elementide kiirest liikumisest ülespoole ja väiksemate allapoole, võimaldades kiiret eemaldamist ja paigutamist.

```
Antud:      järjend - kuhjastatud järjend, sorteerimiseks
Tagastab:   sorteeritud järjend

def sorteeri_kuhi(järjend):
    n = järjend.length

    for i = n - 1 to 1 step -1:
        vaheta(järjend[0], järjend[i])
        korrasta_kuhi(järjend, i, 0)

def korrasta_kuhi(järjend, n, i):
    suurim = i
    vasak = 2*i + 1
    parem = 2*i + 2

    if vasak < n and järjend[vasak] > järjend[suurim]:
        suurim = vasak
    if parem < n and järjend[parem] > järjend[suurim]:
        suurim = parem
    if suurim ≠ i:
        vaheta(järjend[i], järjend[suurim])
        korrasta_kuhi(järjend, n, suurim)
```

Ka selle genereerimisalgoritmi puhul on raskusparameetriks järjendi sorteerimisel kuhja tingimuste säilitamiseks tehtavate võrdluste arv. Kuid erinevalt kuhjastamisest, kus saab iga elemendi kohta kindlaks teha maksimaalse võrdluste arvu, on kuhja sorteerimisel maksimaalse võimaliku võrdluste arvu määratlemine väga keeruline. See tuleneb asjaolust, et iga element võib kuhja tippu sattuda mitu korda. Sellest tulenevalt saab iga elemendi positsioon mõjutada tehtavate

võrdluste arvu ning koguvõrdluste arvu saab defineerida vaid kogu kuhja sorteerimisel terviklikult. Ka kuhja sammsammuline ehitamine, kus kuhjale lisatakse elemente ühekaupa ja jälgitakse võrdluste arvu suurenemist, osutus ajaliselt keerukaks. Selle lähenemisega ei õnnestunud leida ka mustrit, mis võiks muuta algoritmi raskusparameetri tagamise lihtsamaks.

Ainus omadus, mis kuhja meetodil sorteerimisel kindlasti säilib, on suuremate arvude liikumine alati kuhja tipu poole, kuid ka see ei mõjuta oluliselt sorteerimiseks kuluvate võrdluste arvu, mistõttu ei ole põhjust hakata kuhjastatud järjendeid sellest lähtuvalt ehitama. Seetõttu otsustati selles töös sorteerimiseks kuhjad genereerida iteratiivselt katsetusmeetodil.

3 Programmi kasutusjuhised

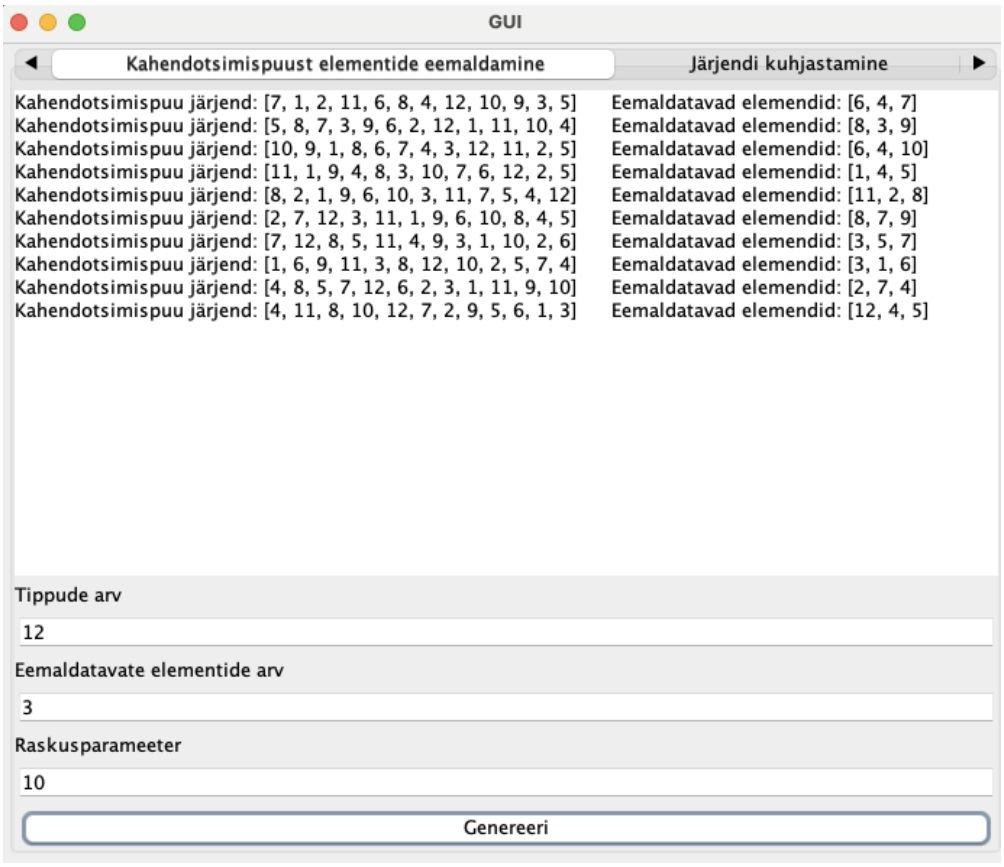
See peatükk annab ülevaate, kuidas töö käigus loodud programmi efektiivselt kasutada ja millised on selle peamised funktsioonid. Lahti on seletatud erinevate algoritmide kasutamisvõimalused ning lisatud ka kasutusnäide.

3.1 Programmi üldise kasutamise põhimõtted

Programm on realiseeritud Java failina, mida käivitades avaneb Swiftiga loodud intuitiivne kasutajaliides. Kasutajatel on võimalus navigeerida kuue erineva vahelehe vahel, kus igaüks keskendub konkreetsele, töö käigus välja töötatud algoritmile. Iga vahelehe esimene sisendväli võimaldab kasutajal määrata elementide või tippude arvu, millele järgneb raskusparameetri valik. Mõne algoritmi puhul on ka vajalik sisestada täiendavaid tingimusi. Programmi peamine eesmärk on pakkuda kasutajatele kiiret ja lihtsat ligipääsu algoritmide poolt genereeritud sisenditele, mistõttu ei ole keerukamatele erinditele suurt tähelepanu pööratud. Juhul, kui sisend ei vasta nõuetele, võib osutuda vajalikuks programm taaskäivitada. Täpsed sisendparameetrite vahemikud on defineeritud programmis.

3.2 Programmi kasutamise näide

Joonisel 3 illustreeritakse kahendotsimispuu elemendi eemaldamise sisendi genereerimise protsessi. Kasutaja on valinud menüüst konkreetse algoritmi vaate ja määranud tippude arvuks kümme. Nagu vastava algoritmi kirjelduses mainitud, on optimaalne eemaldatavate elementide arv valida alla 1/3 kogu tippude arvust, tagamaks programmi efektiivse toimimise. Seega on kasutaja valinud eemaldatavate elementide arvuks 3. Raskusparameetriks on määratud 10, et tippude eemaldamise protsess varieeruks. Programm on genereerinud vastavalt nendele nõuetele kümme sobivate parameetritega sisendit. Igal real kujutatavad järjendid vastavalt kahendotsimispuud ning kolme eemaldamist vajavat tippu.



Joonis 3. Programmi näide kahendotsimispuust elementide eemaldamise sisendi genereerimisest

Kokkuvõte

Käesoleva töö eesmärk sai täidetud. Töö käigus koostati kuuele kursusel "Algoritmid ja Andmestruktuurid" käsitletavale algoritmile sisendülesandeid genereerivad algoritmid. Algoritmid said koostatud kahendotsimispuude, AVL-puude ja kuhjade algoritmidele. Iga algoritmi jaoks sai defineeritud spetsiifiline raskusparameeter, mis võimaldab määrata genereeritavate sisendite keerukuse. Järjendi lugemisel kahendotsimispuuks ja kuhjade puhul kujunes raskusparameetriks algoritmis teostatav võrdluste arv ning AVL-puusse elementide lisamisel ja eemaldamisel oli oluline tasakaalu taastamiseks vajalike pöörete arv.

Lisaks raskusparameetritele võimaldab iga algoritm määrata ka sisendi elementide arvu ning osade puhul saab kasutaja kontrollida ka konkreetsete lisamiste või eemaldamiste hulka. See võimaldab täpsemat kontrolli sisendite genereerimise üle vastavalt kasutaja vajadustele.

Lisaks algoritmidele loodi ka lihtne kasutajaliides, mis võimaldab kasutajal katsetada erinevaid algoritme. Kasutajaliides võimaldab määrata erinevaid parameetreid ning genereerib vastavalt kasutaja valikule kümme sisendit vastavate parameetritega. See võimaldab kasutajal kiiresti ja mugavalt katsetada töö käigus loodud algoritme.

Üks võimalus programmi edasi arendada, on laiendada sisendite genereerimise algoritme veel teistele lahendusalgoritmidele või ühildada loodud algoritmid mõne varasemalt sarnase eesmärgiga kirjutatud töö algoritmidega. Lisaks saab välja töötatud algoritme kasutada õppeprogrammi loomiseks, mis võimaldab õpilastel kursuse "Algoritmid ja Andmestruktuurid" raames interaktiivselt harjutada algoritmide kasutamist erineva keerukusega sisendite peal.

Viidatud kirjandus

[1] Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) üldinfo ÕISis. (05.02.2024). URL: https://ois2.ut.ee/#/courses/LTAT.03.005/details.

[2] Samuel Johannes Pitko. Massiivialgoritmide sisendite genereerimine. TÜ arvutiteaduse instituudi bakalaureusetöö. 2021.

[3] Uku Hannes Arismaa. Graafialgoritmide sisendite genereerimine. TÜ arvutiteaduse instituudi bakalaureusetöö. 2022.

[4] Nikolai Voitsehhovski. Input Generation for Hashtable Algorithms. TÜ arvutiteaduse instituudi bakalaureusetöö. 2022.

[5] Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) moodle’s olevad materjalid. (02.02.2024). URL: https://moodle.ut.ee/course/view.php?id=182.

[6] Jüri Kiho. Algoritmid ja andmestruktuurid. Tartu Ülikooli Kirjastus, 2003.

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Renno Sepp, annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose "Sisendite genereerimine puude ja kuhjade algoritmidele" mille juhendajad on Ahti Põder ja Tõnis Hendrik Hlebnikov 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 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.

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.

Renno Sepp
15.05.2024