TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Samuel Johannes Pitko

Massiivialgoritmide sisendite genereerimine

Bakalaureusetöö (9 EAP)

Juhendaja: Ahti Põder, PhD

Tartu 2021

Massiivialgoritmide sisendite genereerimine

Lühikokkuvõte:
Bakalaureusetöö eesmärgiks oli luua programm aines „Algoritmid ja andmestruktuurid“ käsitlevate massiivialgoritmide sisendite genereerimiseks, mis lihtsustaks õppejõul ülesannete koostamist ning võimaldaks luua õppeprogrammi aines käsitletud algoritmide läbimängimise harjutamiseks. Töö käigus valmis lihtsa graafilise kasutajaliidesega Java programm, millega saab viiele algoritmile genereerida etteantud parameetrite alusel sisendeid.

Võtmesõnad: Algoritmid ja andmestruktuurid, Java, sorteerimine, õpitarkvara

CERCS: P175 Informaatika, süsteemiteooria

Input generation for array algorithms

Abstract:
The purpose of the Bachelor's thesis was to create a program that would allow to generate input arrays for various array algorithms addressed in „Algorithms and data structures“ course. In the process a Java program with a simple graphical user interface was created, which allows generating input arrays for five different algorithms.

Keywords: Algorithms and data structures, Java, sorting, educational software

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Tartu Ülikooli informaatika erialal on olulisel kohal algoritmid ning neid õpetatakse mitmes kohustuslikus aines. Üheks selliseks aineks on „Algoritmid ja andmestruktuurid“, mille eesmärgiks on anda esmane kogemus põhiliste andmestruktuuridega seonduvate algoritmide realiseerimisel [1]. Töö vajadus tuleneb sellest, et hetkel ei eksisteeri aine õppejõul vahendit, millega genereerida kindlate omadustega algoritmide sisendeid, mida algoritme käsitlevate ülesannete koostamisel kasutada.

Lisaks selles töös käsitletud sisendite genereerimisele on tulevikus plaanitud luua ka algoritme käsitlevate ülesannete lahendamise programm, mis tugineks ülesannete genereerimisel selle töö käigus valminud programmile. Õppeprogrammiga saaksid aine läbijad erinevate algoritmide läbimängimist interaktiivselt harjutada ning lahendada ka õppejõu poolt neile määratud ülesandeid. Täiendavalt lubaks programm õppejõul tudengite lahendusi automaatselt kontrollida ja hinnata.

Bakalaureusetöö on rakenduslik ning selle eesmärk on luua Java programm aines „Algoritmid ja andmestruktuurid“ käsitlevate algoritmide sisendite genereerimiseks. Kasutusmugavust arvestades luuakse programmile ka lihtne graafiline kasutajaliides. Programm võimaldaks aine õppejõul automaatset ülesannete genereerimist.

Esimeses peatükis tutvustatakse ainet „Algoritmid ja andmestruktuurid“ ning aines läbitud teemasid. Teises peatükis selgitatakse bakalaureusetöö eesmärki ning uurimismeetodeid ja antakse ülevaade töös kasutatud tehnoloogiatest. Kolmandas peatükis kirjeldatakse töö käigus koostatud algoritme ja nende valmimise protsessi. Viimases peatükis saab lugeda programmi kasutamiseks loodud kasutusjuhendit.

1 Aine „Algoritmid ja andmestruktuurid“

„Algoritmid ja andmestruktuurid“ (LTAT.03.005) on 6-ainepunktilise mahuga õppeaine, mis on informaatika bakalaureuse eriala kohustuslikus osas. Soovitatud on ainet läbida eriala teisel õppeaastal. Aine eesmärgiks on „anda süstemaatiline sissejuhatus põhilistesse andmestruktuuridesse ja nendega seonduvatesse algoritmidesse ning esmane kogemus nende realiseerimisel“ [1]. Antud aine on ka eeldusaineks kahele teisele informaatika eriala õppeainele - Keeletehnoloogia (LTAT.01.002) ja Tehisintellekt (LTAT.01.003).

Tartu Ülikooli õppeinfosüsteemis on välja toodud järgnevad õpiväljundid, mida aine läbinud tudeng peaks valdama [1]:

1. saab aru algoritmi keerukuse mõistest, oskab algoritmi keerukust empiiriliselt ja lihtsamatel juhtudel ka analüütiliselt hinnata ning võrrelda algoritmide töökiirust keerukushinnangute alusel;
2. tunneb põhilisi andmestruktuure ja nendega seonduvaid klassikalisi algoritme, nende kasutustingimusi ja eesmärke ning saab aru nende algoritmide tööpõhimõttest;
3. oskab neid andmestruktuure ja algoritme arvutis realiseerida, kasutada ning eesmärgist lähtuvalt sobivamaid andmestruktuure ja algoritme praktikas eelistada.

Üheks aine põhiliseks teemaks on algoritmide ajaline keerukus, selle liigid ja põhiomadused [1]. Sellega seoses käsitletakse ka erinevaid sorteerimismeetodeid, sest need on üks lihtsamaid viise ajalise keerukuse ja selle erinevate juhtude õpetamiseks. Kuna asjade käsitsi läbi tegemise oskus on hea viis näha, et teemast on aru saadud, siis on tudengitelt eeldatud klassikaliste sorteerimismeetodite läbimängimise oskus. Sarnaselt on 1. kontrolltöös tudengitelt nõutud teatud algoritmi läbimängimist etteantud massiivil, ehk algoritmi töö käigus toimuvate massiivi muutuste välja kirjutamist.

Algoritmide läbimängimisel on aga olulisel kohal algoritmile antud sisend. Sisendi valikust sõltub suuresti läbimängimisel tehtava töö maht ning mõnel juhul võib see olla eeldatust väiksem. Hetkel ei eksisteeri vahendit sisendite genereerimiseks ning seda tuleb teha käsitsi, mis võib osutuda ajamahukaks.

2 Lahenduse ülevaade

Peatükis selgitatakse bakalaureusetöö eesmärki ning uurimismeetodeid ja antakse ülevaade töös kasutatud tehnoloogiatest.

2.1 Töö eesmärk

„Algoritmid ja andmestruktuurid“ aines käsitletud algoritmide harjutamiseks on plaanitud luua õppeprogramm, mis võimaldaks tudengitel interaktiivselt nimetatud algoritmide tööd läbimängida. Loodava õppeprogrammi jaoks on aga tarvilik, et selle kasutaja saaks määrata soovitud „raskusastme“, mis oleks vastavuses tehtava töö määraga, ehk algoritmi poolt tehtavate erinevat liiki sammude arvuga. Sellest tulenebki töö eesmärk - luua algoritmid, millega genereerida soovitud keerukusega sisendeid.

Valminud algoritmid vähendaksid ka aine õppejõu ajakulu ülesannete koostamisel, sest võimaldaksid genereerida ülesannete jaoks ekvivalentseid massiive: mitte samasuguseid, aga sama raskeid. Seetõttu on programmi lihtsamaks kasutamiseks loodud ka naiivne graafiline kasutajaliides.

2.2 Metoodika

Kõigepealt uuritakse käsitletud sorteerimismeetodeid, et välja selgitada, kuidas sõltub nendes tehtav erinevate sammude, näiteks tsükli iteratsioonide, arv algoritmile antud sisendist. Seejärel koostatakse igale valitud meetodile omaette programm, millele saab parameetrina ette anda eelmainitud sammude arvu ning mille tulemusena saaks soovitud pikkuses arvumassiivi selliselt, et algoritmi käitamisel ka täpselt see arv samme tehakse.

2.3 Kasutatud tehnoloogiad

Alampeatükis kirjeldatakse töö käigus kasutatud tehnoloogiaid.

2.3.1 Java

Java on tuntud objektorienteeritud programmeerimiskeel, mida kasutatakse Tartu Ülikoolis mitmete ainete õpetamisel, sealhulgas ka „Algoritmid ja andmestruktuurid“. Java üheks tugevaks eeliseks on selle mitmeplatvormilisus - Java programme on võimalik käitada kõigis levinud operatsioonisüsteemides.

Programmi loomisel otsustati kasutada Javat, sest hiljem valmiv õppeprogramm kirjutatakse samuti Javas, ning sama programmeerimiskeele kasutamine võimaldab

paremat ühilduvust programmide vahel.

2.3.2 Apache Maven

Maven on Apache poolt loodud avatud lähtekoodiga tööriist programmide koostamise automatiseerimiseks [2], mida kasutatakse laialdaselt Java projektide haldamiseks. Tööriist võimaldab muuhulgas vajalike teekide allalaadimist, lähtekoodi kompileerimist, automaattestide käivitamist ning kompileeritud koodi pakendamist. Maveni käivitamine käib läbi ülesannete (ingl *task*), mis võimaldab kasutajal käivitada vaid neid töö etappe, mida parasjagu vaja. Lisaks võimaldab Apache Maven pakendatud programmi või teegi üleslaadimist ühisesse hoidlasse (ingl *repository*), lastes teistel Maveni kasutajatel valminud programmi enda projektides lihtsasti kasutada.

Algoritmide koostamisel otsustati kasutada Mavenit, sest see on Java programmides kõige enam kasutatav ning töö autoril oli tööriistaga varasem kogemus. Mavenit on kasutatud bakalaureusetöös koodi automaattestimiseks ja valminud algoritmide kogumi GitHubi Maveni hoidlasse üleslaadimiseks, võimaldades õppeprogrammi arendajal seda enda programmis kasutada.

3 Algoritmide koostamine

Peatükk annab ülevaate töö käigus koostatud algoritmidest ja nende valmimise protsessist. Kogu programmi kood on kättesaadav ka GitHub'i keskkonnas[1]. Sorteerimismeetodite algoritmide kirjeldustes on lähtutud õppeaine Moodle'i [3] materjalidest ning algoritmide selgitused on refereeritud J. Kiho õpikust [4].

3.1 Lähenemine

Sorteerimismeetodite sisendi genereerimise algoritmide koostamisel alustati iga meetodi poolt tehtava töö mahu kvantifitseerimisega, ehk niinimetatud raskusparameetri valikuga. Seejärel realiseeriti sorteerimismeetodid Javas modifitseeritud kujul selliselt, et meetodi käitamisel oleks võimalik tehtud sammude arvu mõõta. Viimaks koostati meetodite sisendite genereerimiseks algoritmid, lähtudes antud sorteerimismeetodi koodi staatilisest analüüsist ja aine Moodle'is asuvatest läbimänguslaididest [3]. Algoritmide korrektsus sai tagatud automaattestidega, kus mõõdeti meetodi poolt tehtud sammude arvu saadud sisendil ning võrreldi seda eeldatud tulemusega, ehk generaatorile antud parameetriga.

Valminud sisendi genereerimise algoritmid ei genereeri otse massiive, vaid neile on ette antud kasvavas järjestuses massiiv, mida algoritm soovitud keerukusega sorteerimismeetodi sisendi saamiseks segab. Oluline on seejuures märkida, et segamine toimub elementide positsioonide alusel elementide väärtusi arvestamata, mistõttu ei ole toetatud korduvate elementidega massiivid. Algoritmide läbimängimise oskuse demonstreerimist antud kitsendus ei mõjuta. Lisaks koostati juhusliku kasvavas järjekorras arvumassiivi genereerimiseks eraldi meetod.

3.2 Pistemeetod

Pistemeetod (ingl *insertion sort*) on üks lihtsamaid sorteerimismeetodeid, mis on lühikeste massiivide puhul ka suhteliselt kiire, kuid muutub oma ajalise ruutkeerukuse tõttu pikematel massiividel kiiresti ebaefektiivseks.

3.2.1 Pistemeetodi algoritm

Pistemeetod jagab etteantud massiivi virtuaalselt kaheks osaks - sorteerimata ja sorteeritud. Iteratiivselt võetakse vaatluse alla esimene element sorteerimata osast ning otsitakse sellele sobib koht sorteeritud osas, kuhu see paigutada. Vaadeldava elemendi sobivasse kohta „pistmise“ järel suureneb massiivi sorteeritud osa ühe võrra. Pärast viimase sorteerimata elemendi pistmist lõpetab algoritm töö ning

[1] Programmi lähtekood asub aadressil `https://github.com/spitko/algorithms`

massiiv on sorteeritud. Pistemeetodi ajaline keerukus on nii keskmisel kui halvimal juhul $\Theta(n^2)$ [4]. Algoritmi tööd on kujutatud joonisel 1.

```
   Antud : Massiiv a pikkusega n
 1 for i := 1 to n − 1 do
 2 |   j := i;
 3 |   while j > 0 and a_{j−1} > a_j do
 4 |   |   a_j ↔ a_{j−1};
 5 |   |   j := j − 1;
 6 |   end
 7 end
```

Joonis 1. Pistemeetodi algoritm

3.2.2 Pistemeetodi sisendi generaator

Raskust määravaks parameetriks sai valitud sorteerimisel tehtavate pistmiste arv - kusjuures olukorda, kus vaatluse all olev element kohta ei vaheta, pistmiseks ei loeta. Seega on vähim võimalik pistmiste arv 0, kui etteantud massiiv on juba sorteeritud, ja suurim võimalik arv võrdne massiivi elementide arvuga n.

```
   Antud : Sorteeritud massiiv a pikkusega n, pistmiste arv p
 1 for i := n − 1 to 1 do
 2 |   r := juhuslik arv vahemikus 0 ⩽ r < i;
 3 |   if r < p then
 4 |   |   p := p − 1;
 5 |   |   j := juhuslik arv vahemikus 0 ⩽ j < i;
 6 |   |   x := a_j;
 7 |   |   while j < i do
 8 |   |   |   a_j := a_1;
 9 |   |   |   j := j + 1;
10 |   |   end
11 |   |   a_i := x;
12 |   end
13 end
```

Joonis 2. Pistemeetodi sisendi genereerimise algoritm

Koostatud algoritm (joonis 2) on olemuselt sarnane pistemeetodi algoritmile, aga

tagurpidine. Algoritmi sisenditeks on sorteeritud massiiv a ja tehtavate pistmiste arv p. Esiteks läbitakse massiivi vastupidises suunas. Igal tsükli sammul on juhus, et pistetakse juhuslikult valitud eelnev element vaatluse all olevale positsioonile. Eelnimetatud tegevus toimub tõenäosusega $\frac{p}{i}$, kus p on pistmiste arv ja i on vaadeldava elemendi indeks. Pistmise toimumisel väheneb tehtavate pistmiste arv p ühe võrra.

3.3 Valikumeetod

Valikumeetod (ingl *selection sort*) on lihtsa olemusega sorteerimismeetod, mille ajaline keerukus on kõikidel juhtudel $\Theta(n^2)$.

3.3.1 Valikumeetodi algoritm

Valikumeetod on sarnane pistemeetodile, kuid olemuselt veelgi aeglasem. Nagu ka pistemeetodis, jagatakse massiiv sorteeritud ja sorteerimata osaks, kusjuures sorteeritud osa on alguses tühi [5]. Seejärel otsitakse sorteerimata massiivi osast vähim element ning lisatakse see sorteeritud osa lõppu. Tegevust korratakse, kuni kogu massiiv on sorteeritud. Valikumeetodi ajalise keerukuse hinnang on nii halvimal, keskmisel kui ka parimal juhul $\Theta(n^2)$ [5]. Algoritmi tööd on kujutatud joonisel 3.

```
   Antud : Massiiv a pikkusega n
 1 for i := 0 to n − 2 do
 2 |   min := i;
 3 |   for j := i + 1 to n − 1 do
 4 |   |   if a_j < a_min then
 5 |   |   |   min := j;
 6 |   |   end
 7 |   end
 8 |   if i <> min then
 9 |   |   a_min ↔ a_i;
10 |   end
11 end
```

Joonis 3. Valikumeetodi algoritm

3.3.2 Valikumeetodi sisendi generaator

Raskusparameetriks valiti sorteerimisel tehtavate vahetuste arv. Vähim võimalik vahetuste arv on 0, kui massiiv on juba sorteeritud, ja suurim võimalik tehtavate

vahetuste arv võrdne massiivi elementide arvuga n.

```
   Antud : Sorteeritud massiiv a pikkusega n, vahetuste arv v
1  for i := n − 1 to 1 do
2  |  r := juhuslik arv vahemikus 0 ⩽ r < i;
3  |  if r < v then
4  |  |  v := v − 1;
5  |  |  j := juhuslik arv vahemikus 0 ⩽ j < i;
6  |  |  a_i ↔ a_j;
7  |  end
8  end
```

Joonis 4. Valikumeetodi sisendi genereerimise algoritm

Algoritmi (joonis 4) põhiosas läbitakse parameetrina antud massiivi paremalt vasakule, alustades viimasest elemendist. Igal tsükli sammul on tõenäosus $\frac{v}{i}$, kus v on vahetuste arv ja i on elemendi indeks, et vaadeldav element vahetatakse suvalise eelneva elemendiga. Selleks valitakse juhuslik arv vahemikust $[0, i)$ ning võrreldakse seda vahetuste arvuga. Kui vahetus toimub, siis väheneb tehtavate vahetuste arv ühe võrra.

3.4 Mullimeetod

Mullimeetod on lihtne sorteerimisalgoritm, mille üheks eeliseks on selle suutlikkus kiiresti tuvastada olukord, kui massiiv on juba sorteeritud.

3.4.1 Mullimeetodi algoritm

Algoritm (joonis 5) koosneb kahekordsest tsüklist, kus massiivi läbitakse korduvalt ning vahetatakse omavahel kõrvutiasetsevad elemendid, kui need on suuruse poolt vales järjestuses [6]. Massiiv loetakse sorteerituks ning algoritm lõpetab töö, kui massiivi läbimise käigus ei tehta ühtegi vahetust. Parimal juhul on mullimeetodi ajaline keerukus lineaarne, kuid halvimal ja keskmisel juhul on ajaline keerukus $\Theta(n^2)$ [6]. Loengumaterjalides küll nimetatud meetodit täpsemalt ei käsitleta, kuid Moodle'is [3] on siiski välja toodud mullimeetodi (ingl *bubble sort*) läbimänguslaidid.

```
Antud : Massiiv a pikkusega n
1 repeat
2 |   s = false;
3 |   for i := n − 1 to 1 do
4 |   |   if a_{i−1} > a_i then
5 |   |   |   a_{i−1} ↔ a_i;
6 |   |   |   s = true;
7 |   |   end
8 |   end
9 until not s;
```

Joonis 5. Mullimeetodi algoritm

3.4.2 Mullimeetodi sisendi generaator

Algoritmi parameetriks valiti sorteerimisel tehtavate massiivi läbimiste arv. Parimal juhul, kui massiiv on juba sorteeritud, läbitakse mullimeetodis massiivi üks kord ning lõpetatakse seejärel töö. Halvimal juhul tehakse massiivi pikkusega võrdne arv läbimisi. Kuna sorteerimisel liigutatakse elemente massiivi lõpust eespoole, siis liiguvad väikesed elemendid massiivi lõpus kiiresti oma kohale. Suured elemendid massiivi alguses liiguvad lõpupoole aeglasemalt - ühel massiivi läbimisel ühe koha võrra edasi. Seega on mullimeetodis tehtavate läbimiste arv otseses vastavuses oma lõpp-positsioonist kõige rohkem eespool asuva elemendi kaugusega selle lõpp-positsioonist. Ehk kui selleks kauguseks $(i_1 - i_2)$ on n, siis tehakse mullimeetodis kokku $n + 1$ läbimist, kus viimasel läbimisel ühtegi vahetust ei tehta.

Algoritmi (joonis 6) idee põhineb asjaolul, et l läbimiste arvuga tuleb teha vähemalt üks $l-1$ kaugusega vahetus. Sellise kaugusega vahetusi saab aga teha maksimaalselt $n-l+1$. Seepärast alustab algoritm niinimetatud maksimaalse kaugusega vahetuste arvu v leidmisega.

Seejärel läbitakse tsüklis parameetrina antud massiivi vasakult paremale. Igal tsükli sammul, kui kõiki maksimaalse kaugusega vahetusi ei ole veel tehtud, on tõenäosus $\frac{v}{n-i+l-1}$, et vaadeldav element vahetatakse indeksil $i + l - 1$ asuva elemendiga. Juhul kui eelnimetatud vahetust ei toimu, valitakse juhuslik element poollõigult $[i, \min(i + l - 1, n))$ ning vahetatakse see vaadeldava elemendiga. Kuna poollõik sisaldab ka elementi i, siis on võimalik, et tsükli sammul vahetust ei tehta.

Antud : Sorteeritud massiiv a pikkusega n, läbimiste arv l

```
1  v := 0;
2  while v < 1 do
3  |  for i := 1 to n − l + 1 do
4  |  |  r := juhuslik arv poollõigult [0, l);
5  |  |  if r == 0 then
6  |  |  |  v := v + 1;
7  |  |  end
8  |  end
9  end
10 for i := 0 to n do
11 |  max := i + l − 1;
12 |  r := juhuslik arv poollõigult [0, n − max);
13 |  if v > 0 and r < v then
14 |  |  v := v − 1;
15 |  |  a_i ↔ a_max;
16 |  end
17 |  else
18 |  |  j := juhuslik arv poollõigult [i, min (max, n));
19 |  |  a_i ↔ a_j;
20 |  end
21 end
```

Joonis 6. Mullimeetodi sisendi genereerimise algoritm

3.5 Kuhjameetod

Kuhjameetodiga sorteerimine koosneb kahest sammust, kuhjastamisest ja kuhjast suurima elemendi võtmisest. Kuna viimasel sammul ei sõltu tehtavate sammude arv enam sisendist, on sisendi genereerimisel keskendutud ainult kuhjastamise osale. Kuhjastamisel on mitmeid erinevaid implementatsioone, siin jaotises on käsitletud niinimetatud alt-üles varianti, kus kahendpuu tippude töötlust tehakse puu tasemete kaupa alt ülespoole liikudes.

3.5.1 Kuhjastamise algoritm

Massiivi kuhjastamine (ingl *heapify*) on massiivi elementide ümbertõstmine nii, et massiivi kujutamisel kompaktse kahendpuuna kehtib kuhja tingimus. Kuhjaks nimetatakse kompaktset kahendpuud, milles ühegi tipu kirje pole suurem kui

tema ülemuse kirje. Massiivi kuhjastamiseks vaadeldakse kõigepealt massiivi kui kompaktset kahendpuud. Seejärel rakendatakse puu tippudel tasemete kaupa järjest kuhjaparanduse operatsiooni, ehk liigutatakse liiga väike kirje kuhjas allapoole õigele kohale. Kuhjastamise ajaline keerukus on kõikidel juhtudel $\Theta(n)$. Algoritmi tööd on kujutatud joonisel 7.

Antud : Massiiv a pikkusega n

```
1 for i := (n/2) − 1 to 0 do
2 |   heapDown (a, i);
3 end
```

Joonis 7. Kuhjastamise algoritm

3.5.2 Kuhjastamise sisendi generaator

Raskusparameetriks valiti tehtavate allaviimiste arv. Vähim võimalik allaviimiste arv on 0, kui massiiv on juba kuhjastatud, ning suurim allaviimiste arv $\frac{n}{2}$, kus n on massiivi pikkus.

Antud : Massiiv a pikkusega n, allaviimiste arv v

```
1 shuffle (a);
2 heapify (a);
3 T := v juhuslikult valitud alluvatega tippu (poollõigult [0, n/2) );
4 foreach a_i ∈ T do
5 |   j := juhuslikult valitud tipu a_i alluv;
6 |   a_i ↔ a_j;
7 end
```

Joonis 8. Kuhjastamise sisendi genereerimise algoritm

Algoritm (joonis 8) alustab sisendis antud massiivi segamisega ja siis kuhjastab massiivi. Seejärel valitakse alluvatega tippude hulgast kasvavas järjestuses v juhuslikku tippu, kus v on sisendiks antud allaviimiste arv. Valitud tippude kirjed vahetatakse juhuslikult valitud alluvatega.

3.6 Kiirmeetod

Kiirmeetod (ingl *quicksort*) on jaga-ja-valitse (ingl *divide-and-conquer*) põhimõttel algoritm - esialgne ülesanne jagatakse kaheks lihtsamaks ülesandeks, mis omakorda

rekursiivselt samal viisil lahendatakse. Ülesande kaheks jagamisel valitakse massiivist üks kindel element ning jagatakse kõik massiivi elemendid kahte ühisosata alamjärjendisse selliselt, et ühte jääks valitud elemendist väiksemad, teise aga suuremad (või võrdsed) elemendid. Jagamist kordatakse rekursiivselt, kuni kõik osad on sorteeritud, millega osutub sorteerituks ka kogu massiiv. Element, mille põhjal elemente jaotada, on vabalt valitav ning sellest valikust sõltub suuresti ka algoritmi ajalise keerukuse hinnang. Loengutes on käsitletud meetodi varianti, kus selleks elemendiks valitakse alati alamjärjendi esimene element. Kiirmeetodi ajalise keerukuse hinnang on keskmisel juhul $\Theta(n \log n)$, kuid ebasoodsamal juhul $\Theta(n^2)$.

Kuigi kiirmeetodil tehakse võrdse pikkusega massiivide puhul jaotamisi alati ühe palju, mõjutab algoritmi sisend siiski tehtavate jaotuste tulemusena saadud alamjärjendite võrdsust. Ebavõrdsemate alamjärjendite puhul suureneb omakorda algoritmi rekursioonipuu maksimaalne sügavus. Rekursioonipuu sügavus ei mõjuta küll tehtavate jaotuste arvu, aga mõjutab siiski töö käigus tehtavate võrdlusoperatsioonide arvu ning seega kaudselt ka algoritmi läbimängimise raskust. Läbimängimisel tehtavate sammude arv jääb aga siiski samaks.

Sisendi genereerimise algoritmi koostamiseks oleks siis saanud lähtuda kas rekursiooni sügavusest või tehtavate võrdluste arvust, kuid mõlemad lähenemised osutusid liialt keerukaks ning jäid seetõttu tööst välja. Siiski on võimaldatud kiirmeetodi jaoks juhusliku massiivi genereerimine, mis on meetodi läbimängimise kontekstis piisav lahendus, sest elementide järjestus läbimängimise käigus tehtud sammude, ehk jaotuste arvu ei mõjuta.

3.7 Valiku kiirmeetod

Valiku kiirmeetod on algoritm k väikseima elemendi leidmiseks massiivis. Algoritmi tulemusena ei ole massiiv tingimata sorteeritud, vaid soovitud arv vähimaid elemente on toodud juhuslikus järjestuses massiivi algusesse.

3.7.1 Valiku kiirmeetodi algoritm

Valiku kiirmeetod on sarnane kiirmeetodile. Erinevalt kiirmeetodist antakse meetodile lisaks parameetrina arv k, mis tähistab kui mitu vähimat elementi massiivist leida ($k + 1$ elementi). Algoritmis läbitakse sarnaselt kiirmeetodile massiivi rekursiivselt, kuid valiku kiirmeetodi sammus tehakse vaid üks rekursiivne kutse.

3.7.2 Valiku kiirmeetodi sisendi generaator

Raskusparameetriks valiti tehtavate jaotuste arv, ehk kui mitu korda valiku kiirmeetodi algoritmi käitmise jooksul rakendatakse massiivil lahkme järgi ümbertõstmise

protseduuri. Kuid kuna valiku kiirmeetodile antakse parameetrina lisaks ka leitavate vähimate elementide arv, siis on ka generaatoril kokku kaks parameetrit. Optimaalset sisendi genereerimise algoritmi valiku kiirmeetodile ei õnnestunud meetodi keerukuse tõttu luua. Koostati siiski algoritm, mis etteantud massiivi korduvalt juhuslikku järjekorda segab ning igal iteratsioonil kontrollib valiku kiirmeetodiga saadud permutatsioonil tehtud jaotuste arvu.

Joonisel 9 on kujutatud 10-elemendilise massiivi juhuslikul segamisel saadud jaotuste arvude (j) tõenäosused protsentides, $k+1$ elemendi ettetoomisel. Tõenäosuste leidmiseks genereeriti kõik 10-elemendilise massiivi permutatsioonid ning seejärel kontrolliti valiku kiirmeetodi poolt tehtud jaotuste arvu saadud permutatsioonidel. On näha, et jaotuste arvu kasvades väheneb sobiva massiivi leidmise tõenäosus kiiresti. Programmi testimisel selgus aga, et ka kõige haruldasemad juhud, näiteks ühe elemendi ettetoomine üheksa jaotamisega leiab programm vaid mõne sekundiga. Seega on valminud algoritm probleemi lahendamiseks täiesti piisav.

	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
k=0	20.000	36.579	28.054	11.832	3.012	0.475	0.046	0.002	0.000
k=1	10.000	17.734	30.572	26.555	11.802	2.904	0.402	0.029	0.001
k=2	10.000	13.845	22.547	26.466	18.458	7.072	1.455	0.151	0.006
k=3	10.000	12.833	21.342	23.477	19.028	9.932	2.930	0.433	0.025
k=4	10.000	12.778	20.942	22.993	18.256	10.349	3.845	0.776	0.062
k=5	10.000	13.591	20.914	23.057	17.827	9.813	3.778	0.920	0.099
k=6	10.000	15.496	21.519	23.299	17.083	8.698	3.067	0.740	0.099
k=7	10.000	19.266	23.763	23.146	15.149	6.397	1.838	0.385	0.056
k=8	10.000	27.877	31.550	19.750	7.919	2.274	0.518	0.099	0.014

Joonis 9. Valiku kiirmeetodi tõenäosused protsentides (n=10)

3.8 Ühildus- ehk põimemeetod

Põimemeetod (ingl *merge sort*) põhineb sarnaselt kiirmeetodile jaga-ja-valitse printsiibile. Erinevalt kiirmeetodile jagatakse (alam)järjendid rekursiivselt kaheks võrdse pikkusega (või siis ühe võrra erineva pikkusega) pooleks, sõltumata elementide väärtusest. Pärast jagamist ühendatakse ehk põimitakse kõik alamjärjendite paarid, mille tulemusena saab ka esialgne massiiv sorteeritud. Kuna algoritmi jagamise protseduuri tulemusena saadakse alati võimalikult võrdsed pooled, siis on ilmne, et algoritmi rekursioonipuu sügavus on $\Theta(\log n)$. Ning kuna igal tasemel tehtav töö (jaotamine ja põimimine) on $\Theta(n)$, siis on ühildusmeetodi ajaline keerukus kokku $\Theta(n \log n)$.

Kuna põimemeetodis tehakse võrdse pikkusega massiivide puhul jagamisi alati ühe palju, sõltumata elementide järjestusest, siis sorteerimismeetodi läbimängimise kontekstis ei mõjuta algoritmile antud sisend tehtavate sammude arvu. Seega ei osutunud põimemeetodi korral vajalikuks koostada eraldi algoritmi ning sisendi saamiseks piisab juhusliku massiivi genereerimisest.

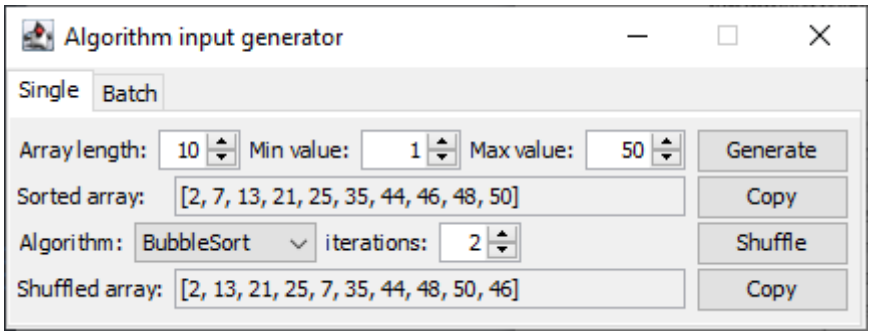
4 Programmi kasutusjuhend

Programm on käivitatava *jar*-faili kujul. Käivitatava programmi saamiseks lähtekoodist on vaja, et arvutis oleks installeeritud Maven ning seejärel lähtekoodi kaustas käsureal sisestada käsk „mvn package“. Programm tekib seejärel kausta *target*, nimega *algorithms-1.4.1.jar*. Kompileeritud programm ja programmi lähtekood on leitavad tööga kaasa antud *zip*-arhiivis.

Programm koosneb kahest vahelehest. Esimesel vahelehel (joonis 10) on võimaldatud üksikute massiivide genereerimine, teisel vahelehel (joonis 11) saab esimesel vahelehel määratud parameetritega genereerida suuremaid massiivide komplekte.

4.1 Ühe massiivi genereerimine

Kõigepealt tuleb valida soovitud massiivi pikkus ning arvude vahemiku alumine ja ülemine piir. Seejärel nuppu „Generate“ vajutades genereeritakse juhuslik sorteeritud massiiv järgmisele reale ning saadud massiivi on võimalik nuppu „Copy“ vajutades kopeerida lõikelauale. Järgmisel real on võimalik valida saadud massiivi segamiseks kasutatav algoritm ja täpsustada selle parameetreid. Parameetrite valik ja lubatud väärtuste vahemik muutub automaatselt algoritmi vahetamisel. Nuppu „Shuffle“ vajutades käitatakse taustal valitud sisendi genereerimise algoritm antud parameetritega ning saadud tulemus ilmub järgmisele reale, kust seda on jällegi võimalik kopeerida.

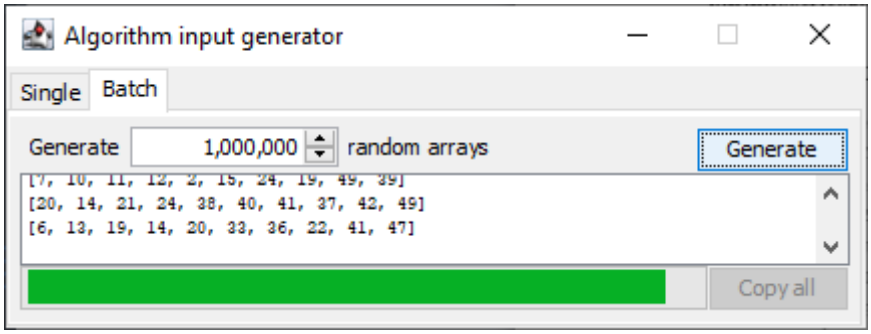


Joonis 10. Programmi esimene vaheleht

4.2 Massiivide komplekti genereerimine

Esiteks tuleb määrata esimesel vahelehel sobivad parameetrid ning seejärel liikuda „Batch“ vahelehele. Uuel vahelehel saab veel lisaks määrata soovitud massiivide koguse. Nuppu „Generate“ vajutades alustatakse massiivide genereerimist, mis ilmuvad ka järk-järgult alumisse lahtrisse. Lahtri all on ka edenemisriba, kust on

võimalik jälgida protsessi kulgemist. Töö lõppemisel saab genereeritud massiivide kogumi lihtsasti kopeerida lõikelauale, vajutades all paremal asuvat „Copy all“ nuppu.



Joonis 11. Programmi teine vaheleht

Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli luua võimalus aines „Algoritmid ja andmestruktuurid“ käsitlevate algoritmide sisendite genereerimiseks. Töö eesmärk sai täidetud ja algoritmid koostati viiele meetodile - mullimeetod, kuhjastamine, valikumeetod, pistemeetod ja valiku kiirmeetod. Igale meetodile üritati luua võimalikult optimaalne algoritm, mis suudaks vähima võimaliku ajaga genereerida sobiva sisendi. Suure osa algoritmide koostamise töömahust moodustas ka selleks vajalik eeltöö, milles uuriti iga käsitletud algoritmi korral selle tööpõhimõtet ja eripärasid, et välja selgitada, kuidas mõjutab algoritmile antud sisend selle poolt tehtud töö määra. Lisaks loodi valminud algoritmide kasutamise lihtsustamiseks programmile lihtne graafiline kasutajaliides.

Programm loodi Java programmeerimiskeeles ning programmi pakendamiseks kasutati Mavenit, võimaldades teistel huvilistel valminud programmi enda projektides kasutada.

Programmi üheks võimalikuks edasiarenduseks on koostada sisendi genereerimise algoritmid ka teistele algoritmidele, näiteks erinevate paisktabeli operatsioonide või paisktabelil sorteerimise jaoks. Lisaks on võimalik töö jooksul valminud algoritme kasutada õppeprogrammi loomiseks, mis võimaldaks aine „Algoritmid ja andmestruktuurid“ läbijatel algoritmide läbimängimist interaktiivselt harjutada.

Viidatud kirjandus

[1] *Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) ülevaade*. (21.04.2021). 2020. URL: https://ois2.ut.ee/#/courses/LTAT.03.005/details.

[2] *Maven*. (01.01.2021). URL: https://maven.apache.org/index.html.

[3] *Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) materjalid*. (10.12.2020). 2020. URL: https://moodle.ut.ee/course/view.php?id=182.

[4] Jüri Kiho. *Algoritmid ja andmestruktuurid*. Tartu Ülikooli Kirjastus, 2003. ISBN: 9789985567678.

[5] *Selection Sort Algorithm*. (10.12.2020). URL: https://www.tutorialspoint.com/data_structures_algorithms/selection_sort_algorithm.htm.

[6] *Bubble Sort Algorithm*. (10.12.2020). URL: https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm.

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Samuel Johannes Pitko**,
*(*autori nimi)

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

   Massiivialgoritmide sisendite genereerimine,
   *(*lõputöö pealkiri)

   mille juhendaja on Ahti Põder,
   *(*juhendaja nimi)

   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.

Samuel Johannes Pitko
07.05.2021