TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Hiie Vill

Graafialgoritmide visualiseerimine

Bakalaureusetöö (9 EAP)

Juhendaja: Härmel Nestra, PhD

Tartu 2016

Graafialgoritmide visualiseerimine

Lühikokkuvõte:

Bakalaureusetöö raames loodi programm graafialgoritmide tööpõhimõtte õpetamiseks aines „Algoritmid ja andmestruktuurid“. Programm genereerib vabalt valitud sisendandmete põhjal algoritmi sammsammulise läbimängu slaididena. Töö kirjeldavas osas antakse ülevaade visualiseeritavatest algoritmidest, võrreldakse loodud rakendust olemasolevate lahendustega ja tuuakse välja programmi kasutusjuhend, puudujäägid ja edasiarendusvõimalused.

Võtmesõnad:

Graafialgoritmid, graafide visualiseerimine

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Visualizing graph algorithms

Abstract:

This thesis introduces a program written as part of the thesis and with the aim of teaching the graph algorithms taught in the course „Algorithms and Data Structures“. The program generates slides of all algorithms step by step. The written part includes descriptions of all implemented algorithms, comparisons of this program to other existing solutions for visualizing graph algorithms, a manual for using the program, shortcomings of the program and ideas for further development of the program in the future.

Keywords:

Graph algorithms, visualizing graphs

CERCS:

P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Aines „Algoritmid ja andmestruktuurid“ kasutatakse puu- ja järjendialgoritmide õpetamiseks automaatselt genereeritud slaide. Graafialgoritme õpetati seni loengus ja praktikumis näiteid tahvlile tehes ning loengumaterjalides on ka algoritmide kirjeldused, kuid seletustega sammsammulisi algoritmide läbimänge neile materjalidena ei leidu.

Selle bakalaureusetöö eesmärk on luua programm, mis võimaldaks aine „Algoritmid ja andmestruktuurid“ lugejatel vabalt valitud sisendandmetega, milleks on graafi tipud, servad ning algtipp, millest algoritmi rakendama hakatakse; genereerida analoogiliselt puu- ja järjendialgoritmidega slaide üheteistkümne õpetatava graafialgoritmi sammsammulisest läbimängust. Genereeritavate slaidide eesmärk on aine kuulajatele algoritmide tööpõhimõtet selgeks teha, mitte üliõpilaste teadmisi kontrollida ega neile algoritmide rakendamist õpetada.

Programm sisaldab olemasolevate graafialgoritmide visualiseerimislahenduste häid omadusi, näiteks algoritmi sammude kirjeldusi ning tippude asukoha ja kaarte kumeruse muutmist, kuid lisab sammudele eestikeelsed kirjeldused ja võimaluse algoritmi läbimängu automaatselt slaididena genereerida. Lisaks on selle töö raames kirjutatud programm erinevalt teistest lahendustest loodud just aine „Algoritmid ja andmestruktuurid“ õpetamisele mõeldes, mistõttu toetab see parajasti selles aines õpetatavaid graafialgoritme.

Töö esimene peatükk sisaldab taustainfot: töös ja programmis kasutatavate mõistete definitsioone ning kõikide visualiseeritavate graafialgoritmide kirjeldusi.

Töö teises osas antakse ülevaade olemasolevatest lahendustest graafialgoritmide visualiseerimiseks ja nende erinevustest selle töö raames loodud programmiga.

Töö kolmandas peatükis on loodud programmi kirjeldus ja installeerimis- ja kasutusjuhendid. Tuuakse välja programmi puudujäägid ja edasiarendusvõimalused tulevikuks.

Lisadena on kaasas programmi lähtekood ja litsents töö avaldamiseks. Lisatud failid sisaldavad nii kirjutatud koodifaile kui ka programmi tutvustavat *readme* faili ja kahte näidisena mõeldud sisendfaili.

1. Taustainfo

1.1 Graafidega seonduvad mõisted

Järgnevalt on ära toodud kõik graafide ja graafialgoritmidega seonduvad mõisted sel kujul, kuidas neid töö raames käsitletakse. Enamik mõisteid selles peatükis on defineeritud sel kujul, nagu nad esinevad raamatus „Introduction to Algorithms“ [1:1168-1171].

Suunatud ehk *orienteeritud graafiks G* nimetatakse paari (*V*, *E*), kus *V* on mittetühi lõplik hulk ning *E* on binaarne relatsioon hulgal *V*. *Mittesuunatud* ehk *orienteerimata* graafiks G nimetatakse paari (*V*, *E*), kus *E* koosneb järjestamata tipupaaridest. *V* elemente kutsutakse *tippudeks*, E elemente (järjestatud tipupaarid) suunatud graafide puhul *kaarteks*, mittesuunatud graafide puhul *servadeks*. Kui algoritm toetab nii suunatud kui suunamata graafe, siis kasutatakse serva nime. Kui suunatud graafis *G* leidub n serva (v_i, *v*), kus *i* = 1…n, siis tipu *v* *sisendastmeks* loetakse arvu *n*. Kui suunatud graafis *G* leidub *n* serva (*v*, v_i), kus *i* = 1…*n*, siis tipu *v* *väljundastmeks* loetakse arvu *n*.

Teeks pikkusega *k* tipust *v* tippu *w* graafis *G* = (*V*, *E*) nimetatakse sellist tippude järjendit $\langle v_0, v_1, \ldots, v_k \rangle$, mille korral $v = v_0$, $w = v_k$ ning iga *i* puhul, kus *i* = 1, 2, …, *k*, kuulub (v_{i-1}, v_i) graafi servade hulka. Öeldakse, et tee *sisaldab* tippe $v_0, v_1, \ldots, v_k$ ja servi (v_0, v_1), (v_1, v_2), …, (v_{k-1}, v_k). Tee pikkuseks loetakse tees sisalduvate servade arvu. Kui suunatud graafis leidub tee tipust v tippu *w*, öeldakse, et *v* on *w eellane* ja *w* on *v järglane* [2]. Suunatud graafi puhul nimetatakse *tsükliks* teed $\langle v_0, v_1, \ldots, v_k \rangle$, nii et $v_0 = v_k$ ning tee sisaldab vähemalt ühte serva. Mittesuunatud graafi puhul peavad lisaks kõikidel servadel lähte- ja sihttipud erinema. Kui graafis leidub mõni tsükkel, nimetatakse seda graafi *tsükliliseks*, vastasel juhul *atsükliliseks*. Mittesuunatud graafi nimetatakse *sidusaks*, kui graafi igast tipust leidub tee kõikidesse teistesse tippudesse. Vastasel korral nimetatakse graafi *mittesidusaks*. Suunatud graafi nimetatakse *tugevalt sidusaks*, kui graafi igast tipust v_i leidub tee igasse tippu v_j ja vastupidi.

Graafi $G' = (V', E')$ nimetatakse graafi $G = (V, E)$ *alamgraafiks*, kui $V' \subseteq V$ ja $E' \subseteq E$. Graafi *tugevalt sidusateks komponentideks* nimetatakse graafi tugevalt sidusaid alamgraafe, millele ei saa tugevalt sidusust säilitades tippe juurde lisada. *Kaalutud graafideks* nimetatakse graafe, mille iga servaga on vastavuses kaal, tüüpiliselt reaalarv. Selle graafi servade kohta võib öelda, et need on *kaalutud servad* [1:591]. *Eeldusgraafiks* nimetatakse suunatud atsüklilist graafi, mille kõik tipud on vastavusse viidud positiivsete arvude ehk *hindadega*. Eeldusgraafi võib vaadelda kui projekti, graafi tippe kui töid ning hindu kui vastavatele töödele kuluvat aega [2]. Tipu *varaseimaks lõpuajaks* nimetatakse varaseimat võimalikku aega, mil tipuga tehtav töö valmis saab, *hiliseimaks algusajaks* aga kõige hilisemat aega, mil tipu töö alustada võib, nii et kogu projekti varaseim lõpuaeg ei suurene [3:94-95]. *Kriitiline töö* on tipp, mille hiliseima algusaja ja hinna summa on võrdne tema varaseima lõpuajaga. *Kriitiline tee* on tee $\langle v_0, v_1, \ldots, v_k \rangle$, kus v_0 on eellasteta tipp, v_k on järglasteta tipp ja iga *i* korral, kus *i* = 1, 2, …, *k*, on tipp v_i kriitiline [3:96].

Topoloogiline järjestamine on graafi *G* = (*V*, *E*) tippude järjestamine sel moel, et kui graafis esineb serv *E* = (*v*, *w*), siis tipp *v* on järjestuses eespool tippu *w* [1:612].

Graafi *minimaalne toespuu* on puu, mis sisaldab kõiki graafi *G* = (*V*, *E*) tippe ning mille servade kaalude summa on minimaalne[1:624]. Toespuu kujutab endast graafi *G* alamgraafi. Mittesidusa graafi puhul kasutatakse terminit *minimaalne toesmets*.

Algtipp on tipp, mille algoritm saab parameetrina ette ja mis võetakse läbimisalgoritmidel esimeseks jooksvaks tipuks.

Serva $e = (v, w)$ *otstippudeks* nimetatakse tippe v ja w [4]. *Silmusteks* nimetatakse selliseid servi, mille otstipud langevad kokku [4]. Töös on kasutatud kaare $e = (v, w)$ otstippude v ja w tähistamiseks tavapärase algtipu ja lõpptipu asemel vastavalt termineid *lähtetipp* ja *sihttipp*, vältimaks segadust esimese tipuga, millele algoritmi rakendatakse ning mida siin algtipuks kutsutakse.

1.2 Graafialgoritmid

Tartu Ülikooli informaatika õppekava aines „Algoritmid ja andmestruktuurid“ õpetatakse 11 graafialgoritmi:

1. graafi laiuti läbimine;
2. graafi tippude töötlemine sügavuti eesjärjestuses;
3. graafi tippude töötlemine sügavuti lõppjärjestuses;
4. Primi algoritm;
5. Kruskali algoritm;
6. Dijkstra algoritm;
7. Floyd-Warshalli algoritm;
8. topoloogiline järjestamine lõppjärjestuse kaudu;
9. topoloogiline järjestamine Kahni algoritmiga;
10. eeldusgraafi analüüs;
11. Kosaraju algoritm.

See nimekiri kajastab ainult selle bakalaureusetöö raames visualiseeritavaid algoritme, tegelikult õpetatakse näiteks ka Bellman-Fordi algoritmi, mille visualiseerimine ei olnud aga tarvilik.

1.2.1 Graafi laiuti läbimine

Olgu kaaludeta või konstantsete kaaludega graafil $G = (V, E)$ valitud algtipp v_0. Algtipp märgitakse töödelduks ja kõik selle tipuga seotud servad lisatakse servade järjekorda. Kuni servade järjekord pole tühi, korratakse järgmist:

1) järjekorrast valitakse ja eemaldatakse esimene serv $e = (v, w)$;

2) kui serva üks otstipp on töötlemata, märgitakse see töödelduks ja järjekorda lisatakse kõik äsja töödeldud tipuga seotud servad, mille teine otstipp on töötlemata. Vastasel juhul ei tehta midagi.

Laiuti läbimise algoritmi abil leitakse graafi tippude läbimise järjekord, alustades tipust v_0, ning graafi laiuti läbimise puu [5:176]. Erinevalt viidatud originaalist on siin kirjeldatud ja visualiseerimisel kasutatud algoritmi servapõhist versiooni, mille puhul hoitakse järjekorras servi, mitte tippe. Laiuti läbimise leiutas 1950. aastatel Edward F. Moore [1:623].

1.2.2 Graafi tippude töötlemine sügavuti eesjärjestuses

Olgu kaaludeta või konstantsete kaaludega graafil $G = (V, E)$ valitud algtipp v_0. Algtipp märgitakse töödelduks ja kõik selle tipuga seotud servad lisatakse servade magasini. Kuni servade magasin pole tühi, korratakse järgmist:

1) magasinist valitakse ja eemaldatakse esimene serv $e = (v, w)$;

2) kui serva üks otstipp on töötlemata, märgitakse see töödelduks ja magasini lisatakse äsja töödeldud tipuga seotud servad. Vastasel juhul ei tehta midagi.

Sügavuti eesjärjestuses läbimise algoritmi abil leitakse graafi sügavuti eesjärjestuses tippude läbimise järjekord, alustades tipust v_0, ning graafi sügavuti läbimise puu. Erinevalt viidatud originaalist on siin kirjeldatud ja visualiseerimisel kasutatud algoritmi servapõhist versiooni, mille puhul hoitakse magasinis servi, mitte tippe. Nii tippude sügavuti eesjärjestuses kui lõppjärjestuses töötlemise algoritmid on tehtud iteratiivselt, mitte rekursiivselt, et nad oleks võimalikult sarnased teiste algoritmidega ning üliõpilastel neid kergem mõista ja võrrelda oleks. Graafi sügavuti läbimise algoritm on olnud kasutusel 1950. aastate lõpust [1:623].

1.2.3 Graafi tippude töötlemine sügavuti lõppjärjestuses

Olgu kaaludeta või konstantsete kaaludega graafil $G = (V, E)$ valitud algtipp v_0. Algtipp märgitakse töödelduks ning kõik temaga seotud servad lisatakse servade magasini. Algoritmi töö käigus tehakse järgmist:

1) magasinist valitakse ja eemaldatakse esimene serv $e = (v, w)$;

2) kui tipp w on töötlemata, lisatakse magasini selle tipuga seotud servad. Vastasel juhul ei tehta midagi;

3) kui tipul w on töötlemata järglasi, lisatakse vastavad servad magasini. Kui ei, siis märgitakse w töödelduks ja vastav serv eemaldatakse magasinist.

Sügavuti lõppjärjestuses läbimise algoritmi abil leitakse graafi sügavuti lõppjärjestuses tippude läbimise järjekord, alustades tipust v_0, ning graafi sügavuti läbimise puu [1:604]. Erinevalt viidatud originaalist on siin kirjeldatud ja visualiseerimisel kasutatud algoritmi servapõhist versiooni, mille puhul hoitakse magasinis servi, mitte tippe.

1.2.4 Primi algoritm

Olgu $G = (V, E)$ kaalutud mittesuunatud sidus graaf., millel on valitud algtipp v_0. Kõik tipuga v_0 seotud servad lisatakse eelistusjärjekorda, väiksema kaaluga servad ees. Kuni servade eelistusjärjekord pole tühi, toimitakse järgnevalt:

1) servade eelistusjärjekorrast eemaldatakse esimene serv;

2) kui serva üks otstippudest oli külastamata, siis lisatakse serv e tekkinud puusse, külastamata otstipp loetakse külastatuks ning eelistusjärjekorda lisatakse kõik selle tipuga seotud servad, mille teine otstipp on külastamata.

Algoritmi tulemusena leitakse graafi minimaalne toespuu [1:634, 5:219-220]. Algoritmi töötas välja Vojtěch Jarník aastal 1930, kuid hiljem avaldas selle aastal 1957 uuesti Robert C. Prim [1:642].

1.2.5 Kruskali algoritm

Kaalutud mittesuunatud graafi $G = (V, E)$ kõik servad lisatakse kaalu põhjal servade eelistusjärjekorda, väiksema kaaluga servad ees. Graafi iga tipp moodustab omaette puu. Kuni eelistusjärjekord ei ole tühi, toimitakse järgnevalt:

1) servade eelistusjärjekorrast eemaldatakse esimene serv $e = (v, w)$;

2) kui v ja w kuuluvad erinevatesse puudesse, siis ühendame need serva e lisamisega üheks puuks. Kui v ja w kuuluvad samasse puusse, ei tehta midagi.

Kui tegu on sidusa graafiga, siis leitakse algoritmi tulemusena graafi minimaalne toespuu, vastasel juhul minimaalne toesmets [1:631, 5:221]. Kruskali algoritmi leiutas Joseph B. Kruskal aastal 1956 [1:642].

1.2.6 Dijkstra algoritm

Dijkstra algoritm leiab mittenegatiivsete kaaludega suunatud graafi $G = (V, E)$ kõikide tippude kaugused valitud algtipust v_0 ja vastavad lühimad teed algtipust iga tipuni. Iga tipuga seatakse vastavusse kaugus algtipust. Algtipul on see alguses 0, ülejäänud tippudel ∞. Nende kauguste põhjal pannakse tipud eelistusjärjekorda. Seejärel toimitakse järgnevalt, kuni eelistusjärjekord tühjaks saab:

1) eelistusjärjekorrast eemaldatakse esimene tipp v;

2) iga v naabertipu jaoks arvutatakse uus kaugus algtipust tipu v kaudu. Arvutatavaks kauguseks on v ja vastava naabertipu vahelise kaare kaalu ja v kauguse algtipust summa. Kui uus kaugus on väiksem kui senine, siis kaugust uuendatakse.

Algoritmi töö lõpuks on iga tipuga vastavuses olev kaugus algtipust vähim võimalik [1:658-661]. Tulemusena leiti ka kauguste puu, mis kujutab vastavaid lühimaid teid algtipust iga tipuni. Dijkstra algoritmi avaldas Edsger W. Dijkstra aastal 1959 [1:682].

1.2.7 Floyd-Warshalli algoritm

Floyd-Warshalli algoritm leiab kõikide tippude kaugused üksteisest suunatud kaalutud graafis, milles ei esine negatiivseid tsükleid. Koostatakse külgnevusmaatriks M, mille ridade ja veergude arv on n, kus n on tippude arv, ja iga i ja j korral, kus i ja j on täisarvud lõigus $[1, n]$, vastab $M[i][j]$ kaugusele tipust i tippu j. Alguses täidetakse arvudega ainult selliste tippude lahtrid, mille vahel on serv, ülejäänutesse kirjutatakse ∞. Maatriksi peadiagonaal täidetakse nullidega. Seejärel tehakse järgnevalt:

1) iga $k = 1, 2, \ldots, n$ puhul, kus n on tippude arv, tehakse järgnevalt:

- fikseeritakse maatriksi k-s tulp ja k-s veerg;
- iga maatriksi lahtri $M[i][j]$ väärtust uuendatakse, juhul kui $M[i][k] + M[k][j] < M[i][j]$, kus $i = 1, 2, \ldots, n$ ja $j = 1, 2, \ldots, n$. Uueks väärtuseks saab sel juhul $M[i][k] + M[k][j]$.

Algoritmi tulemuseks on külgnevusmaatriks, mille iga lahter $M[i][j]$ tähistab vähimat teepikkust tipust v_i tippu v_j [1:695-697]. Floyd-Warshalli algoritmi avaldas 1962. aastal Robert W. Floyd, kuid see põhineb Stephen Warshalli 1962. aastal avaldatud teoreemil [1:706].

1.2.8 Topoloogiline järjestamine lõppjärjestuse kaudu

Olgu $G = (V, E)$ suunatud kaaludeta atsükliline graaf [1:612]. G läbitakse alates juhuslikult valitud algtipust v_0 sügavuti lõppjärjestuses ning kirjutatakse välja tekkinud lõppjärjestus. Kuniks graafis leidub veel külastamata tippe, valitakse juhuslikult uus algtipp, läbitakse uuesti sügavuti lõppjärjestuses ning lisatakse tekkinud järjestus eelnevale lõppjärjestusele. Kui tekkinud lõppjärjestus ümber pöörata, on tulemuseks graafi G topoloogiline järjestus. Seda algoritmi kirjeldas esimesena Donald E. Knuth aastal 1968 [1:623]. Käesoleva töö esitamise aegsest versioonis ei valita algtippu juhuslikult, vaid võetakse sügavuti läbimise esmakordse rakendamise korral argumendina ning edaspidi valitakse iga kord uueks algtipuks juhuslikult üks tipp, mille sisendaste on 0.

1.2.9 Topoloogiline järjestamine Kahni algoritmiga

Algoritm graafi topoloogilise järjestuse leidmiseks Kahni algoritmiga töötab järgnevalt:

1) iga tipuga seatakse vastavusse temasse sisenevate kaarte arv;

2) valitakse suvaliselt üks selline tipp v_x, millele vastab arv 0. Tipp v_x kirjutatakse tekkivasse järjestusse;

3) kõikide selliste tippudega, millesse viis kaar tipust v_x, vastavuses olevat arvu vähendatakse ühe võrra.

Järjest korratakse samme 2-3, kuni kõik tipud on läbi käidud. Tekkinud järjestus on graafi G topoloogiline järjestus [2]. Seda algoritmi kirjeldas esimesena Arthur Kahn aastal 1962 [6].

1.2.10 Eeldusgraafi analüüs

Olgu $G = (V, E)$ suunatud kaaludeta atsükliline graaf, mille kõikidele tippudele on määratud positiivsed hinnad, mis kujutavad endast tipu läbimiseks kuluvat aega. Eeldusgraafi analüüsi käigus leitakse graafi G iga tipu kohta varaseim lõpetamisaeg ja hiliseim algusaeg. Ühtlasi leitakse kogu projekti varaseim lõpetamisaeg ning kriitilised teed. Topoloogilises järjestuses leitakse iga tipu varaseim lõpetamisaeg järgnevalt:

1) kui tipul eellasi ei ole, siis võrdub tipu varaseim lõpetamisaeg tipu hinnaga;

2) kui tipul on eellasi, siis võrdub tipu varaseim lõpetamisaeg tipu hinna ja tipu eellaste suurima varaseima lõpuaja summaga.

Kogu projekti varaseim lõpuaeg on võrdne tippude maksimaalse varaseima lõpuajaga. Tagurpidi topoloogilises järjestuses leitakse iga tipu hiliseim algusaeg järgnevalt:

1) kui tipul järglasi ei ole, võrdub tipu hiliseim algusaeg projekti varaseima lõpuaja ja tipu hinna vahega;

2) kui tipul on järglasi, võrdub tipu hiliseim algusaeg tema järglaste minimaalse hiliseima algusaja ja tipu hinna vahega.

Kriitilistele teedele jäävad kõik tipud, mille hiliseima algusaja ja hinna summa on võrdne vastava tipu varaseima lõpuajaga [2].

1.2.11 Kosaraju algoritm

Kosaraju algoritm leiab suunatud kaaludeta graafi $G = (V, E)$ kõik tugevalt sidusad komponendid. Kuni leidub külastamata tippe, tehakse järgnevalt:

1) valitakse suvaliselt algtipp ja rakendatakse sellele sügavuti lõppjärjestuses läbimist;

2) kirjutatakse välja sügavuti läbimise lõppjärjestus.

Iga sügavuti läbimise tulemusena tekkinud lõppjärjestus lisatakse eelmise läbimise käigus tekkinud lõppjärjestusele, nii et tulemuseks on järjestus, mis sisaldab graafi kõiki tippe. Seejärel tekitatakse pööratud kaartega graaf, tipud märgitakse jälle mittekülastatuks ning tekkinud lõppjärjestus keeratakse tagurpidi. Kuni leidub veel tippe, mis pole üheski tugevalt sidusas komponendis, tehakse järgnevat:

1) valitakse esimene tipp tagurpidi lõppjärjestusest, mis pole veel üheski tugevalt sidusas komponendis;

2) sellest tipust alates teostatakse graafi läbimist ning kõik läbitud tipud liidetakse esimesega üheks tugevalt sidusaks komponendiks [1:616-617].

Kosaraju algoritmi leiutas 1978. aastal Rao S. Kosaraju [1:623]. Sarnaselt topoloogilisele järgjestamisele lõppjärjestuse kaudu ei valita esitatud versioonis sügavuti läbimiseks algtippu

juhuslikult, vaid esimesel korral antakse argumendina kaasa ning järgmistel kordadel valitakse uueks algtipuks juhuslikult üks tipp sisendastmega 0.

2. Olemasolevad visualiseerimisvõimalused

2.1 Visualisatsioonid veebis

Internetis leidub erinevate eesmärkide ja realisatsioonidega algoritmide visualiseerimissaite, millest mõned haakuvad osaliselt loodud programmi eesmärkide ja omadustega. Kolm asjakohasemat visualiseerimissaiti on järgnevad:

1. Müncheni Ülikooli algoritmide visualiseerimissait [7]. Sellel saidil on algoritmide läbimängude juures kõige põhjalikumad sammsammulised kirjeldused. Siin on võimalik ise graafe koostada ja kaartele kaale määrata ning seejärel sammhaaval läbimängu teha. Paraku on vajalikest algoritmidest olemas ainult Kruskali, Primi, Floyd-Warshalli ja Dijkstra algoritmid.

2. San Francisco Ülikooli algoritmide visualiseerimissait [8]. Sellel saidil on kõige rohkem aines „Algoritmid ja andmestruktuurid“ õpetatavaid graafialgoritme kujutatud – kõik peale eeldusgraafi analüüsi ja Kosaraju algoritmi – ning algoritmide läbimäng on visuaalselt hea. Samas ei kirjeldata igal sammul piisavalt, mida tehakse. Graafide servad ja kaalud genereeritakse näiliselt juhuslikult, aga kindlasti on sisse kodeeritud, millistest tippudest millistesse kaari minna saab. Erinevaid graafe on seega valikus palju, kuid valida saab vaid kahe eri tippude arvu vahel ja tippude paigutus on samuti fikseeritud. Muuta ei saa ka kaalude numbreid, mis on alati lõigus [1, 9], nii et näiteks negatiivsete kaaludega graafe toetava Floyd-Warshalli algoritmi seda eripära demonstreerida ei saa. Lisaks kirjutatakse sügavuti otsingul välja ainult eesjärjestus, nii et ees- ja lõppjärjestuse erinevust ei selgu. Üks selle saidi erinevus kõikidest teistest leitud saitidest on see, et siin on võimalik tippe ja servi kujutada mitte ainult graafina, vaid ka külgnevusmaatriksi ja järjendina.

3.VisuAlgo [9]. Ka sellel saidil on alamlehtede „Graph Traversal“, „Min Spanning Tree“ ja „SS Shortest Paths“ all olemas enamik algoritme: Kruskali, Dijsktra, Primi ja Kosaraju algoritmid, laiuti ja sügavuti läbimine ja topoloogiline järjestamine lõppjärjestuse kaudu, ehkki sügavuti läbimisel pole ei ees- ega lõppjärjestust välja toodud. Sellel saidil saab ise graafe luua ja läbimänge teha, aga kõrval on koodinäidis olemas. Kirjeldusi läbimängudel tehtavast ei ole, seega algoritmide algusest peale õppimiseks ei ole see kõige parem variant, küll aga iseenda kontrollimiseks siis, kui algoritmi tööpõhimõte juba selgeks õpitud.

2.2 Programmid

Lisaks veebisaitidele leidub algoritmide visualiseerimiseks internetist allalaetavaid programme, nagu näiteks:

1) Graph Drawing tool [10]. See programm sisaldab vaid Primi, Krsukali ja Dijsktra algoritme ning sügavuti ja laiuti otsinguid;

2) Animal Algorithm Visualization System [11]. Selles programmis on olemas Primi, Kruskali, Dijkstra, Floyd-Warshalli ja Kosaraju algoritmid ning sügavuti ja laiuti otsingud. Siin on algoritmidel konkreetsed koodinäited ka kõrval, aga saab ise graafe luua ja nende põhjal

algoritme jooksutada. On võimalik ka sammsammulisi kirjeldusi juurde lisada ning koodinäiteid eemaldada.

2.3 Loodud programmi võrdlus olemasolevate visualiseerimisvõimalustega

Ehkki leidub saite, mis õpetavad erinevate algoritmide tööpõhimõtet, ei leidu saiti, mis kõiki aines „Algoritmid ja andmestruktuurid“ käsitletavaid graafialgoritme õpetaks, vaid eri algoritmide jaoks on sobilikemad olemasolevad visualiseeringud eri veebisaitidel. Primi, Kruskali ja Dijkstra algoritmide visualiseerimise kohta leidub rohkem erinevaid näiteid kui teiste algoritmide kohta. Eeldusgraafi analüüsist ega graafi tippude sügavuti töötlemisest, nii et lõppjärjestuses tulemus välja kirjutataks, head visualiseeringut ei leidugi. Lisaks ei ole enamikul eksisteerivatel saitidel võimalik ise algandmeid valida, vaid saab ainult ühe kuni mitme etteantud näite vahel valida ja nende põhjal läbimängu teha. Näiteks alapeatükis 2.1 välja toodud saitidest ei saa ise sisendandmeid valida San Francisco saidil. Mitte ühelgi saidil pole ka võimalust automaatselt läbimänguslaide genereerida. Lisaks mainitud saitidele leidub selliseid algoritmide õpetamiseks mõeldud saite, mis ei võimalda üldse visualiseerimist, vaid on tekstilisel kujul, koodinäidetega või konkreetsete näidisgraafide piltidega. Visualiseerimist võimaldavatest saitidest leidub nii selliseid, mis on küll mõeldud algoritmide õpetamiseks, ent millel jääb midagi vajaka (pole piisavalt seletusi, mida tehakse, või kontrollitakse ainult seda, kas inimene juba algoritmi mõistab), nagu näiteks Texase Ülikooli saidil [12], kui ka visualiseeringuid, mis on tehtud hoopis muul otstarbel, mitte algoritmi tööpõhimõtte kujutamiseks. Ühestki eeltoodud saidist ei ole antud juhul kasu, sest neist ei piisa ei tudengitele õppimiseks ega praktikumijuhendajale oma sisenditega näidiste koostamiseks.

Käesoleva töö raames loodud programm kombineerib olemasolevate visualiseeringute head küljed ning lisab läbimänguslaidide genereerimise ja sammudele eestikeelsed kirjeldused läbimängu. Selle programmiga on võimalik ise sisendandmeid valida ja servi kumerdada. Viimast ei ole ühegi senise programmiga ega ühelgi saidil võimalik teha, kuid mõnel saidil olid servad juba automaatselt kumerad, ehkki just sellistel saitidel, kus sisendandmeid valida ei saanud. Kui võrrelda käesoleva töö raames loodud programmi teiste mainitud programmidega, siis esimesega on võimalik graafe ka failis kirjeldada.

3. Programm

Programm on kirjutatud OCamlis ning kasutatakse graafikamoodulit Graphics ning sõnede töötlemise moodulit Str, mis on programmi Makefile’i kompileerimisreeglisse sisse kirjutatud ja mis peaks iga OCamli distributsiooniga kaasas olema. Rohkem OCamli väliseid teeke ei kasutatudki, küll aga muud tarkvara, milleks on GNU make, MikTexis sisalduvad mpost ja epstopdf ning PDF toolkit.

3.1 Programmi kirjeldus

Programmi abil saab ise saab ära märkida graafi tipud ja olemasolul nende hinnad, graafi servad ja olemasolul nende kaalud ja suunad; algtipu, millest algoritmi läbimäng algab, ja soovi korral tippude koordinaadid. Graafide sisestamisest on põhjalikumalt kirjutatud alapeatükis 3.3.1.

Lisaks tippude koordinaatide märkimisele sisendandmete sisestamisel saab nende asukohti hiirega lohistades pildil muuta, kuniks nad musta kastiga märgitud pildi raamidesse jäävad. Ka servade kuju on võimalik hiirega lohistades muuta sirgest kuni poolringini. Servade puhul tuleb ise jälgida, et nad pildilt välja ei jääks.

Graafi tippe ja servi kujutatakse viie erineva värvitooniga, kusjuures tippude värvitoonid on servade omast pisut heledamad, kuid teistega võrreldes märgatavalt sarnased. Vaatlemata tippe ja servi ehk selliseid, milleni algoritmi töö käigus üldse jõutud ei ole, kujutatakse punasena. Vaadeldavaid tippe ja servi ehk parajasti jooksvaid tippe ja servi kujutatakse kollasena. Kui mõni tipp või serv on parajasti valitud, näiteks vahetult enne töötlemist või teistest vaadeldavatest tippudest ja servadest eristamiseks, siis kuvatakse neid oranžiga. Oranžiga kuvatakse ka eeldusgraafi töö lõppedes kriitilisi teid, kuvamaks tulemust, kuid eristamaks neid tippe ja servi vaadeldud tippudest ja servadest. Vaadeldavad tipud ja servad ehk sellised, mille algoritm on läbinud ja millega enam ei tegelda, märgitakse roheliseks. Mõne algoritmi puhul, näiteks Primi ja Kruskali, puhul kasutatakse ka halli tooni selliste servade jaoks, mida on küll valitud, kuid pole saanud tekkivasse puusse lisada. Kosaraju algoritmi puhul märgitakse halliks ka juba leitud sidusad komponendid. Seda samuti ülejäänud vaadeldud tippudest ja servadest eristamiseks, kuid siin oranži värvi algoritmi töö käigus veel kasutatakse ning ka arusaadavuse huvides näis hall sobilikum variant olevat.

Programmiaknas kuvatakse lisaks graafile ka musta kasti, mille piiridesse graaf jääma peaks, et graaf slaididel teksti ei kataks, ja läbimängu korral graafi kohal tippude või servade nimekirju.

Iga algoritmi puhul antakse ette algtipp: koodi muutes tuleb see ise funktsioonis Programm.main ära märkida, graafi sisendfailist lugemise korral võtab programm algtipuks esimesena neljandal real kirjeldatud tipu. Algtippu nõudvad algoritmid on laiuti ja sügavuti läbimised, Primi algoritm, Dijkstra algoritm, topoloogilise järjestuse leidmine lõppjärjestuse abil ja Kosaraju algoritm. Viimased kaks ei nõua klassikaliselt algtippu, vaid valitakse juhuslik tipp, kuid siinse realisatsiooni järgi alustatakse kummagi algoritmi käigus tippude sügavuti lõppjärjestuses töötlemist algtipust ning kui mõni tipp töötlemata jäi, alustatakse uuesti juhuslikust tipust, mille sisendaste on null, minimeerides nõnda uuestialustamiste arvu.

Kohe pärast sisendandmete kättesaamist teostatakse programmi töö alguses kontroll, kas sisendgraaf vastab algoritmi poolt esitatud nõuetele ning kas graaf ise on sobilik. Mittesobilikud graafid on näiteks sellised, millel leidub mitu sama nimega tippu, millel leidub kahe tipu vahel rohkem kui üks serv (suunatud graafide puhul on kahe tipu vahel kaks kaart lubatud, kui need on vastupidiste suundadega). Graafikontroll on vajalik algoritmide töö ootuspärase kulgemise jaoks. Selle käigus kontrollitakse ka näiteks seda, et graafi servad oleks kaaludega või kõik kaaludeta (Primi, Kruskali, Dijkstra ja Floyd-Warshalli algoritmid nõuavad, et graafi servad oleks kõik kaaludega, ülejäänud algoritmide puhul just ei tohi kaale olla) ja tipud kõik kas hindadega või hindadeta (eeldusgraafi analüüsi puhul peavad kõikidel tippudel hinnad olema, ülejäänud algoritmide puhul ei tohi hindu olla). Mitte nõuetele vastava serva või tipu leidmise korral lõpetab programm töö asjakohase veateatega. Läbimisalgoritmide, Primi algoritmi ja Dijkstra algoritmi puhul kontrollitakse ka seda, et graaf oleks sidus. Õigupoolest kontrollitakse seda, kas valitud algtipust leidub tee kõikidesse graafi tippudesse. Lisaks kontrollitakse, et kõik kaalud oleks mittenegatiivsed (välja arvatud Floyd-Warshalli algoritmil) ja kõik hinnad positiivsed. Kuvamise esteetilisuse nimel kontrollitakse ka, et kõikide tippude nimed oleks pikkusega üks tähemärk ning et servade kaalud jääksid lõiku [-99, 99].

Slaidide genereerimise käigus koostatakse OCamlis automaatselt MetaPosti fail, millesse kirjutatakse vajalik kood programmiaknas oleva pildi ja nimekirjade ning algoritmi sammukirjelduste kujutamiseks MetaPosti abil. Iga algoritmisamm eraldatakse MetaPosti koodis, nii et neid hiljem slaididena kuvada saaks. Seejärel, parameetriks MetaPosti fail, kutsutakse OCamlist välja käsureakäsk „mpost“, mille abil luuakse abil MetaPostist .mps failid, mille arv võrdub slaidide arvuga. Pärast seda kasutatakse iga tekkinud .mps faili peal käsureakäsku „epstopdf“, mis muudab iga .mps faili üheslaidiliseks PDFiks. Siis liidetakse „pdftk“ abil kõik tekkinud PDF failid üheks PDFiks kokku. Kuna see võtab veidi aega, siis

prinditakse töö käigus konsooliaknasse iga slaidi järel, mitu slaidi mitmest on juba PDFiks tehtud. Slaidide eduka genereerimise lõpus kustutatakse kõik vaheetappidel tekkinud slaidid jooksvast kaustast uuesti ära.

3.2 Installeerimisjuhend

Operatsioonisüsteemidest toetab programm Windowsi ja Linuxit, testitud on Windows 8, Ubuntu 16.04 ja SuSE 13.1 peal.

Programmi tööks vajalik tarkvara koos installeerimisjuhistega Windowsis ja Ubuntus:

1. OCaml. Windowsis tuleb selleks laadida alla ja installeerida sobiv versioon aadressilt https://www.typerex.org/ocpwin.html ning kui seda automaatselt ei tehtud, siis lisada bin kaust keskkonnamuutujatesse. Linuxis piisab käsust „sudo apt-get install ocaml“.

2. GNU make. Windowsis on vajalik installida MinGW või Cygwin, Linuxis on see juba olemas.

3. Metapost. Windowsis peab installima MikTexi aadressilt http://miktex.org/download ning vajadusel lisama bin kausta (vaikimisi C:\Program Files\MiKTeX 2.9\miktex\bin\x64) keskkonnamuutujatesse. Linuxis on vajalikud 3 TeX Live paketti, mis on saadavad käskudega „sudo apt-get install texlive-binaries“, „sudo apt-get install texlive-metapost“ ja „sudo apt-get install texlive-font-utils“.

4. PDF toolkit. Windowsis saab aadressilt https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/ tarkvara alla laadida, Linuxis käsuga „sudo apt-get install pdftk“.

Ainult läbimängu tegemiseks piisab ka esimesest kahest, kuid slaidide genereerimise jaoks on vaja ka MetaPosti failist PDFiks tegemise vahendeid.

3.3 Kasutusjuhend

Järgnev kasutusjuhend on väga detailne ja kirjeldab üksikasjalikult graafide sisestamist, läbimängu sooritamist ja slaidide genereerimist. Lühem kasutusjuhend on leitav failis readme.md.

3.3.1 Graafide sisestamine

Sisendandmeid on võimalik ette anda kahel viisil. Üks võimalus on kirjeldada sisendandmed (graafi tipud, servad ja algtipp) tekstifailis järgneva formaadi järgi:

- 1. real: Algoritm: [Laiuti|SygavutiEes|SygavutiLopp|Prim|Kruskal|Dijkstra|FloydWarshall|TopoLopp|TopoKahn|Eeldusgraaf|Kosaraju]
- 2. real: Tippe: {tippude arv}, hindadega: [true|false]
- 3. real: Servi: {servade arv}, kaaludega: [true|false], suundadega: [true|false]
- järgneval *n*-l real, kus *n* on tippude arv: {tipu nimi} <, [{tipu x-koordinaat}|-], [{tipu y-koordinaat}|-] <, [{tipu hind}|-]> >
- järgneval *m*-l real, kus *m* on servade arv: {1. tipu nimi}, {2. tipu nimi} <, [{tipu kaal}|-]>

Loogeliste sulgudega väli tuleb asendada sobiva arvu või tähemärgiga, kandiliste sulgude puhul tuleb valida üks püstkriipsuga eraldatud valikutest ning nurksulud tähistavad valikulist välja, mis tähendab, et nende vahel oleva võib lisada, kuid võib ka ära jätta.

Tühikuid võib vahele jätta, kuid reavahetusi mitte. Komad, koolonid, tippude arv ja servade arv peavad kindlasti õiged olema. Kui valikulised väljad (koordinaadid, hind, kaal) tühjaks jätta või sinna "-" kirjutada, valitakse suvaliselt sobiv arv. Algtippu nõudvate algoritmide puhul võetakse algtipuks esimene failis kirjeldatud tipp. Näited on failides Andmed.txt ja Andmed2.txt.

Tippude arv, servade arv, tipu x-koordinaat ja tipu y-koordinaat peavad olema mittenegatiivsed täisarvud, tipu hind positiivne täisarv, serva kaal peab olema täisarv ning tipu nimi, 1. tipu nimi ja 2. tipu nimi pikkusega 1 tähemärk.

Graafis peab olema vähemalt üks tipp, servade arv pole piiratud. Lubatud ei ole samanimelised tipud ega silmused (servad tipust iseendasse). Kaks serva kahe tipu vahel on lubatud vaid juhul, kui tegu on suunatud graafiga ning need servad on vastupidiste suundadega. Servade tipunimed peavad kuuluma eelnevalt defineeritud tippudele. Kirjeldatud graaf peab kindlasti vastama valitud algoritmile, mida antud graafi peal läbi mängima hakatakse, ehk rahuldama algoritmi sisendilt nõutavaid tingimusi. Graafi kontrollitakse programmi töö alguses ning iga nõuetele mitte vastava graafi sisestamise korral lõpetab programm töö veateatega.

Alternatiivselt on võimalik sisendandmeid muuta mooduli Programm funktsioonis main. Igale algoritmile on kirjutatud neli sobivat näidisgraafi, kõik kujul „nt{algoritm}[1-4]“. On ka näidisgraafid yksTipp ja yksTippHinnaga, mis kujutavad endast ühe hinna või hinnata tipu ja mitte ühegi servaga graafi. Kõikide etteantud näidisgraafide puhul peab algtipuks olema valitud tipp A. Lisaks on võimalik ise koodis graafi luua, kasutades selleks meetodeid Graafika.looGraaf, Graafika.looTipud ja Graafika.looServad. Graafika.looTipud saab argumendiks tipuandmete listi kujul ({tipu nimi}, {tipu x-koordinaat}, {tipu y-koordinaat}, [None|Some {tipu hind}]). Graafika.looServad saab argumendiks servaandmete listi kujul ({1. tipu nimi}, {2. tipu nimi}, [None|Some {serva kaal}], [true|false]), kus viimane tõeväärtus käib serva suunatuse kohta. Graafika.looGraaf saab arumendiks tippude listi ja servade listi.

3.3.2 Läbimäng

Kõigepealt on vaja Makefile’iga samas kaustas käivitada käsurealt käsk „make“, mis koostab käivitatava faili tulem.exe. Windowsis saab seejärel programmi käivitada käsuga „tulem.exe“, Linuxis „./tulem.exe“.

Programmi käivitades ilmub aken kirjeldatud graafiga. See on mõeldud vaheetapina slaidide genereerimisel, et veenduda graafi paigutuse sobivuses ning graafi soovi kohaselt ümber paigutada – tippude asukohad võisid juba sisendandmetes kirjas olla, kuid servi saab kumerdada vaid siin. Kuna see on vaid vaheetapp, siis ei ole siin visuaalsele perfektsusele rõhku pandud. Graafi tipud ja servad on kujutatud eri värvides, mille tähendused on seletatud alapeatükis 3.1.

Programmiaknas hiirega tippudele klikkides ning hiirt all hoides ja lohistades on võimalik tippude asukohti muuta. Samal põhimõttel on võimalik muuta servade kumerust sirgest kuni poolringini. Jälgida tuleb, et graaf üleni musta kastiga piiratud alasse jääks, et graaf slaidide genereerimisel pildile jääks ega teksti kataks. Tippe ega kaale ei ole niigi võimalik piiridest välja tõmmata, kuid kumerate servadega ja hindadega võib nii juhtuda.

Vajutades klahvi ’n’, saab algoritmi läbimängu teostada ning iga klahvivajutusega ühe sammu edasi minna, et näha järgmisi samme ning neid graafi paigutamisel arvesse võtta. Pildi kohal kuvatakse algoritmile kohaseid tippude ja/või servade nimekirju ning konsooliaknasse prinditakse tehtava sammu seletus. Klahvi ’b’ abil saab tagasi läbitud sammude juurde minna. Ka tagasisamme sooritades prinditakse vastava sammu tekst konsooliaknasse.

3.3.3 Slaidide genereerimine

Kui graaf on meelepäraselt paigutatud, on võimalik klahvile ’s’ vajutades alustada automaatset slaidide genereerimist. Programm loob slaidide genereerimise käigus jooksvasse kataloogi ajutisi faile, kuid programmi eduka töö korral kustutab need pärast kasutamist ka ära. Slaidide genereerimine lõppeb teatega konsooliaknas „PDF valmis.“, misjärel programmiaken sulgub ning programm lõpetab oma töö. Tulemusena ilmub jooksvasse kataloogi vastava algoritmi nimega PDF. Kui sellise nimega fail juba kaustas oli, kirjutatakse see üle.

3.3.4 Programmi töö lõpetamine

Programmi töö lõpetamiseks on kolm võimalust. Klahviga ’q’ lõpetatakse sündmuste ootamine ja sellega lõpetab programm iseenesest töö. Teine võimalus on sulgeda programmiaken ristist, kuid Graphics teegi iseärasuste tõttu lõppeb see Windowsis akna hangumisega ega pole mugavaim viis programmi tööd lõpetada. Linuxis võib nii talitada küll.

Slaide genereerides peaks programm ootuspärase töö puhul pärast PDFi genereerimist lõpetama iseenesest. Kui slaidide genereerimine oli edukas, lõpetatakse töö tavapäraselt. Kui slaidide genereerimine mingil põhjusel ebaõnnestus, siis lõpetab programm töö asjakohase veateatega.

3.4 Kasutatud tehnoloogiad ja nende valik

Keele ja teekide valikul lähtusin eelkõige nende sobivusest ülesannete lahendamiseks ning nende pakutavatest võimalustest. Arvesse on võetud ka installeerimise lihtsuse, et kasutatav tarkvara sõltuks võimalikult vähe teistest teekidest, et nii Windowsis kui Linuxis programmi hõlbus kasutada oleks ega lisateeke installima peaks. Ehkki see oli eesmärk, ei olnud valitud keele sisseehitatud funktsioonidega võimalik kogu programmi tööks vajalikku realiseerida, nii et paar teeki tuleb sellegipoolest juurde installida, kuid vähem kui alternatiivsetel variantidel oleks pidanud. Määrav oli ka keele ja/või teegi selgeksõppimise ja kasutamise lihtsus, et võimalikult kergesti minimaalse töötava versioonini jõuda.

3.4.1 Keelevalikul kaalutud variandid

Töö autor soovis kasutada just funktsionaalprogrammeerimiskeeli nii nendes kirjutamise kogemuse kui nende tugevate külgede tõttu, milleks on näiteks lühike kood. Seega jäi valik Haskelli ja OCamli vahele. Kuna olemasolev süsteem puu- ja järjendialgoritmide visualiseerimiseks on tehtud OCamlis, oleks saanud seda laiendada. Samas ei ole OCaml puhas funktsionaalse programmeerimise keel, vaid võimaldab ka imperatiivset programmeerimist, millega on autor rohkem kokku puutunud. Just funktsionaalprogrammeerimise parema õppimise eesmärgil sai esialgu valitud Haskell. Kuna OCamlis õnnestus aga kiiremini minimaalne töötav versioon valmis saada, samas kui Haskellis esines probleeme tippude koordinaatide muudetavusega, siis jäi lõplikuks valikuks siiski OCaml.

3.4.2 Graafikateegi valikul kaalutud variandid

Haskellis alustades kasutas autor graafikateeki Gloss [13]. Selle valiku ajend oli just selle kasutamise lihtsus võrreldes OpenGLiga, mille peale Gloss ise ehitatud on. Ehkki Gloss pakub vähem võimalusi kui OpenGL, oleks nendest antud töö jaoks piisanud ning algeline versioon graafide kuvamisest sai loodud. Gloss jäi kõrvale aga Haskellist loobumise ja OCamli kasuks otsustamise tõttu.

OCamlisse on sisse ehitatud moodul Graphics, mille abil on võimalik joonist kuvada ning hiire- ja klahvisündmuseid käsitleda. Lisaks saab OCamliga kasutada kõrvalisi graafikateeke, nagu näiteks OCaml-Xlib [14]. Teegi valikul jäi määravaks kasutumise lihtsus ja teegi sõltuvuste arv. Linuxis ei ole lisateekide installimine keeruline, kuid kuna programm peab toetama ka

Windowsi, siis võinuks lisateekide installimine keeruliseks ja ajakulukaks minna nii autorile kui kõigile, kes seda kasutada sooviks. Kuna Graphics moodul on juba OCamliga kaasas ning seega hõlpsasti kasutatav, aga iga muu graafikateegi jaoks oleks pidanud lisateeke installima, siis jäi valik Graphics teegi peale. Ehkki vaid põhilisemaid visualiseerimisfunktsioone sisaldav (näiteks sündmustele reageerimisel pidi palju ise implementeerima ja hiire mingi elemendi peal asumist pidi kontrollima arvutuste abil, mitte mõne olemasoleva funktsiooniga), näis see esialgu rahuldavat kõiki programmi vajadusi. Vastupidine selgus alles hiljem, sellest on täpsemalt kirjutatud alapeatükis 3.5.

3.4.3 Pildi salvestamiseks kaalutud variandid

Slaidide genereerimise jaoks oli vaja Graphics mooduli abil joonistatud aknast pilt mingil kujul kätte saada ja salvestada, et siis slaidile edasi kanda. Kuna Graphics moodulis ei ole võimalik programmiakna sisu üheski levinud formaadis salvestada, ainult funktsioonide Graphics.get_image ja Graphics.dump_image abil värvimaatriks salvestada, oli tarvis leida muu lahendus. Üks variant oli kasutada mõnda teist programmeerimiskeelt ning vastav värvimaatriks näiteks JPG formaadis salvestada. Javas on selline asi võimalik ning eksisteerib teek OCaml-Java, mille abil saab OCamlist Java funktsioone välja kutsuda [15]. Selle teegi töölesaamisega esines aga probleeme, nii et see jäi programmi kirjutamisel kaasamata.

Teine variant oli kasutada teeki CamlImages, mille abil on võimalik programmiakna sisu salvestada levinumates formaatides, näiteks JPG, PNG, PS [16]. Sel teegil on aga ka liiga palju sõltuvusi: Findlib, OMake ning valitud formaadi jaoks veel eraldi teek, näiteks PNG jaoks libpng ja JPG jaoks libjpeg [17].

Kolmas kaalutud variant oli teek GraphicsPDF, millega saab programmiakna sisu otse PDFiks teha. Kuna see teek kasutab ka CamlPDFi, siis saab sellega ka pildil fonti muuta. Graphics mooduliga teksti stiili ja suurust muuta ei saa, samuti ei toeta see reavahetusi ega täpitähti, mistõttu pilti salvestades või kohe PDFiks tehes oleks pidanud enne tekstiprobleeme lahendama. GraphicsPDFi puhul oli kaks puudujääki. Esiteks sai selle abil teha pildist vaid ühe PDFi (katsetades ilmnes, et vastav funktsioon kutsuti välja alles pärast Graphics.close_graph funktsiooni, mis juba akna sulgeb). Vaja oli aga salvestada mitu seisu. Teiseks on ka GraphicsPDFil palju sõltuvusi: OcamlFind (Findlib) ja CamlPDF, millest viimane sõltub veel omakorda teegist Zlib [18, 19].

Viimane variant oli viia kuvamine hoopis mõnda teise keelde. Selle teostamiseks oleks tulnud kõikide kuvatavate elementide kuvamise funktsioonid osaliselt või täielikult teises keeles ümber kirjutada. Kuna puu- ja järjendislaidide pildid olid genereeritud MetaPostiga ning selle abil tehtud pilti saanuks lihtsasti PDFiks teha, siis jäi valituks see kui kokkuvõttes parim variant.

3.4.4 Slaidide genereerimiseks kaalutud variandid

Slaidide genereerimiseks kaaluti kolme võimalust, mis kõik ka realiseeritud said.

OCamli abil pildi kujutamiseks vajalik MetaPosti fail temp.mp loodud, sai programmist käivitada käsurealt käsku mpost temp.mp, mis lõi igast slaidist omaette faili laiendiga .mps. Sellistest failidest sai omakorda käsu epstopdf abil PDF formaadis faile luua. Kuna epstopdf võtab argumendiks vaid ühe .mps faili korraga, oli üks võimalus pärast .mps failide loomist need automaatselt ühte faili kokku kirjutada ning sellele käsku epstopdf rakendada [20]. See variant oli küll kiire, kuid sellega jäid kõik slaidid pärast esimest nihkesse, sest loodud MetaPosti koodis ei ole vasakul all nurgas (0,0), vaid negatiivsete koordinaatidega punkt. Teise versioonina loodi igast .mps failist eraldi PDF ning seejärel liideti käsu pdftk abil tekkinud PDFid üheks tulemfailiks kokku. Selle variandi ainus puudus oli aeglus võrreldes eelmisega: kuna epstopdf osutus ülejäänud operatsioonidest aeglasemaks ning esimese variandi korral pidi

seda tegema ühe korra, teise variandi puhul nii mitu korda, kui mitu slaidi tekkinud tulemfailis oli, siis kulus sellele sõltuvalt algoritmist ja sisendandmetest mõni sekund kuni mõni minut.

Kolmas variant ei rakenda käsku epstopdf üldse ning kasutab slaidide genereerimiseks LaTeXit. Kõigepealt luuakse iga slaidi kohta üks .mps fail nagu teisteski variantides, kuid lisaks koostatakse OCamlis LaTeXi fail temp.tex, mis kaasab kõik .mps failid, millest igaüks kujutab endast ühte slaidi. Seejärel saab käsu pdflatex temp.tex abil koostada PDF formaadis fail. See variant on küll kiirem kui teine ning ilma nihkes slaididideta, kuid selle puhul ei kata pildid täpselt slaide. Kui esimese ja teise variandi puhul sai igast pildist omaette slaid, siis sellega on lisaks pildile igal slaidil veel vaba ruumi.

Kuna teise variandi puhul ei ole aeglus niivõrd segav ning slaidide genereerimise juures polegi oluline, et programm tingimata 1-2 sekundiga oma töö ära teeks, siis on teine variant esimese ja kolmandaga võrreldes kõige parem, sest selle puhul katab pilt terve slaidi üleni ning ei pea tegelema pildi slaidi mõõtmetele sobivaks skaleerimisega. Võrdluseks, kolmanda variandi puhul oleks olnud tarvis näiteks Floyd-Warshalli algoritmi läbimängu pilte 0,7-kordselt kuvada, et need slaidile ära mahuks.

3.5 Programmi puudused

Programmi nähtavaim puudus on see, et slaidide genereerimisel ei õnnestunud fonti sobivaks saada, mistõttu slaididel ei kuvata õigesti ei täpitähti ega isegi mitte tühikuid. Visuaalsuse huvides ei tasu kindlasti programmi slaidide genereerimiseks kasutada, kuni täpitähtede kuvamine korda pole saadud.

Programmil on mitu puudust, mis tulenevad Graphics mooduli piiratusest ning avalduvad programmiakna puhul, kuid mitte slaididel. Programmiakna suurust saab küll muuta, kuid siis ei saa tippude asukohti ega servade kumerust muuta. Samuti ei saa Windowsis akent ristist korralikult sulgeda. Sulgemiseks on loodud ka teisi võimalusi, ent ristist sulgemist ei saa ei eemaldada ega parandada. Graphics moodul sisaldab küll funktsioone set_font ja set_text_size, kuid tegelikult on need jäänud mooduli lähtekoodis implementeerimata, mistõttu programmiaknas kuvatava teksti suurust ja fonti muuta ei saa [21]. Ka reavahetusi ega teksti automaatset jagamist mitmele reale (inglise keeles *wrap*) ei ole võimalik teha. Saanuks kontrollida tähemärkide arvu ja vastavalt akna laiusele õiges kohas uue tekstireaga alustada, kuid kuna algoritmide kirjeldustekst kuvatakse niikuinii konsooliaknas ning programmiaknas vaid tippude ja servade nimekirjad, mis on mõistliku tippude ja servade arvu korral lühikesed ega ulatugi üle ääre, siis jäi see omadus lisamata. Pealegi on ka aknaservast üle minemise korral võimalik programmiakent suurendada ja siiski teksti näha.

Kuna Graphics moodul toetab vaid täisarvulisi nurki, kuid väga suure raadiusega servade kuvamisel on tingimata vajalik täpsemini arvutada, siis kuvatakse igal juhul ringjoone ühele kraadile vastav osa, mitte väiksem. See aga tähendab, et servi vaid pisut kumerdades võib joonistatud serv tippudest teiselt poolt läbi minna.

Suure ringjoone raadiusega servade puhul võib ka juhtuda, et ületatakse MetaPosti lubatud 1,4 meetrit [22].

Suunatud graafide puhul võib leiduda ka silmuseid, kuid kuna selle kuvamist loodud programm ei toeta, siis programm selliste servadega graafe ei aktsepteeri.

Kui klahviga ’n’ läbimängu tehes algoritmi viimase sammuni jõuda ja naasta, siis ei ole võimalik enam slaide genereerida ega uuesti viimase sammuni minna, vaid eelviimaseni.

3.6 Probleemid lahendamisel

Lahendamisel tekitas raskusi tehnoloogiate valik ning algne töötama saamine. Sellele kulus ka teiste alamülesannetega võrreldes kõige rohkem aega, seda enam, et suur osa ajast oli töö tulemuse mõttes ebaproduktiivne, kuna paljud tehnoloogiad ei sobinud, mis selgus aga alles katsetamise käigus.

Kirjaliku osa tegemisel tekitas probleeme see, et nii algoritmidest kui graafiga seonduvatest mõistetest on erinevaid ja kohati isegi üksteisega vastuolulisi versioone, nii et tuli valida antud programmi ja algoritmide realisatsioonide kohaselt sobivaimad ja tihti mitmest eri algoritmiversioonist ja definitsioonist sobivat kombineerida, et õige versioon saaks.

Kuna OCamli graafikamoodulis Graphics on vaid mõned elementaarsemad funktsioonid, siis pidi palju vajalikke funktsioone ise leiutama. Näiteks oli noolte ja kumerate servade kuvamise ja hiiresündmustele reageerimise jaoks tarvis väga palju koordinaate arvutada, mis ei käinud aga üle jõu ja oli autorile isegi meelepärane. Sellegipoolest oleks mõne rohkem arendatud graafikateegiga graafika poolele mõnevõrra vähem vaeva ja aega kulunud.

Töö kõige raskemaks osaks osutus aga slaidide genereerimine. See eeldas ka uute tehnoloogiatega kurssi viimist ja katsetamisi ning selle taha autor ka osaliselt takerdus – tulemus ei ole siiani päris selline, nagu ta võiks olla. Näiteks on endiselt probleeme pildi slaidile paigutamise ja tekstiga, mis tuleb edaspidi lahendada.

3.7 Mida uuesti tegemise korral muuta?

Kui oleks tarvis praeguste teadmistega kogu projekti uuesti teha, siis valiks autor kindlasti teise graafikateegi, millel oleks rohkem võimalusi. Algoritmide läbimängude realiseerimiseks oli OCaml piisavalt hea variant, kuid sobivat graafikateeki ei pruugigi leiduda – enamikul on veel sõltuvusi teistest teekidest ning ka need ei pruugi pakkuda kõiki funktsioone kuvamiste mugavaks realiseerimiseks. Seega kõige kindlam oleks kolida kogu graafikapool üleüldse mõnesse teise keelde, seda enam, kui kasutajaliides ka teha. Kuna töö autor on kõige tuttavam Java ja selle graafikateegiga JavaFX, siis üritaks ta veel Ocamlit ja Javat ühendada, nii et kogu loogika pool jääks OCamlisse, kuid kogu graafika pool Javasse.

3.8 Edasiarendamisvõimalused

Kõigepealt võiks likvideerida olemasolevad puudused. Selleks peaks saama korda fondi, et slaididel täpitähti ja tühikuid kuvada ning teksti suurust muuta. Samuti peaks võimaldama tipust iseendasse minevaid servi ning kasutama MetaPostis kumerate servade kuvamiseks mitte osa ringjoonest, vaid joonistama serv kolme arvutatava punkti abil. Graphics teegis oli võimalik kaart joonistada vaid ringjoone võrrandi abil ning seetõttu sai mõlemale ühine funktsioon, kuid MetaPostis on ka teisi võimalusi.

Seni hoiustatakse koordinaate ja kaarte andmeid täisarvudena, arvutusteks teisendatakse ujukomaarvudeks ning uuesti salvestamiseks tagasi täiearvudeks. Seetõttu näiteks kumerate servadega seotud tippu lohistades kaarte kumerus väheneb ajapikku. Visuaalse täpsuse ja kasutusmugavuse huvides võiks kõiki arvulisi muutujaid hoidagi ujukomaarvudena, vältimaks teisendamist ja sellega potentsiaalsete ebatäpsuste teket.

Sisestatud graafide puhul kontrollitakse enne nende vastavust algoritmile: sidusust, kaalude ja hindade olemasolu, kaalude märke ja suunatust. Edaspidi võiks teostada ka tsüklilisuse kontrolli, sest topoloogilistel järjestamiste ja eeldusgraafi analüüsi puhul peab graaf olema atsükliline ning Floyd-Warshalli puhul ei tohi leiduda negatiivseid tsükleid.

Kui suunatud graafis leiduvad kaared (*v*, *w*) ja (*w*, *v*), siis võiks need kasutusmugavuse huvides kohe kumeralt kuvada, et need kahe silma vahele ei jääks ja seetõttu slaididel sirgetena kattuma ei jääks.

Praegu on algoritmid realiseeritud, kasutades lihtsuse mõttes vaid andmestruktuure List, Array ja Hashtbl. Mõne algoritmi pseudokoodide järgi tuleks aga kasutada näiteks järjekordi ja magasine. Efektiivsema ja originaalilähedasema lahenduse saamiseks võiks teha sobilikud asendused.

Lisaks võiks mõnda algoritmi muuta, näiteks Dijkstra algoritm Primi ja Kruskali algoritmiga analoogiliselt servapõhiseks teha. Hetkel on algoritmid üksteisest kohati väga erinevad ning nende kergemaks õppimiseks võiks kõik ühte moodi tehtud olla, olgugi et mitte klassikaliste variantidena tehtud.

Sügavuti lõppjärjestuse abil topoloogilise järjestuse leidmise ja Kosaraju algoritmid võiks seevastu just pigem originaalilähedaseks muuta, vahetades algtipust alustamise ja nullise sisendastmega tipu valimise uueks algtipuks selle juhusliku valimise vastu. Slaidide genereerimise poolest on hea, kui sügavuti läbimist alustatakse just sellistest tippudest, et ühe läbimisega võimalikult paljude tippudeni jõuda, kuid algoritmide õpetamise seisukohast oleks parem kasutada algoritme originaalsel kujul ilma sisendastmete leidmise lisasammuta.

Kokkuvõte

Töö käigus loodi üheteistkümne eri graafialgoritmi tööpõhimõtte õpetamiseks programm, mis loeb failist sisendandmeid, võimaldab tekkinud aknas hiirega tippe paigutada ja servi kumerdada ning koostab automaatselt slaidid, nii et igal slaid kujutab üht algoritmi sammu ning sisaldab pilti graafist sel sammul ja tekstilist kirjeldust, mida algoritmi sellel sammul tehakse. Sammuseletuste põhjalikkus, tippude ja servade eri värvides kuvamine vastavalt nende vaadeldavusele ning kõikide nõuetele vastavate sisendgraafide toetamine tagavad algoritmide tööst ühese ja vajaliku arusaamise.

Ehkki loodud programm ei ole täiuslik ning edasiarendusvõimalusi jätkub eelkõige korraliku kasutajaliidese loomise, graafikateegi vahetamise ja visuaalsete paranduste näol, on see piisav, et vajalikke slaide genereerida ja algoritmide tööpõhimõtet selgeks teha.

Töö oli suures osas praktiline. Töö praktilise osa tegi keeruliseks, aga samas ka huvitavaks OCamli mooduli Graphics funktsionaalsuste vähesus, mistõttu oli vaja palju koordinaatide arvutamise ja kuvamise funktsioone ise kirjutada. Enim valmistas raskusi tehnoloogiate valik, töölesaamine ja üksteisega sidumine.

Kasutatud materjalid

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, „Introduction to algorithms“, Third Edition, The MIT Press, 2009

[2] http://kodu.ut.ee/~nestra/mat/inf/p/aa/08s/s/graafid.pdf, viimati vaadatud 09.05.2016

[3] J. Kiho, „Algoritmid ja andmestruktuurid“, kolmas trükk, Tartu Ülikooli Kirjastus, 2003

[4] A. Buldas, P. Laud, J. Villemson, „Graafid“, 2003

[5] K. Mehlhorn, P. Sanders, „Algorithms and Data Structures“, Springer, 2008

[6] https://en.wikipedia.org/wiki/Topological_sorting, viimati vaadatud 12.05.2016

[7] https://www-m9.ma.tum.de/Allgemeines/GraphAlgorithmen, viimati vaadatud 12.05.2016

[8] https://www.cs.usfca.edu/~galles/visualization/Algorithms.html, viimati vaadatud 12.05.2016

[9] http://visualgo.net/, viimati vaadatud 12.05.2016

[10] http://people.engr.ncsu.edu/mfms/Software/GDR-1.2.9/index.html, viimati vaadatud 11.05.2016

[11] http://www.algoanim.net/, viimati vaadatud 11.05.2016

[12] https://www.me.utexas.edu/~jensen/ORMM/methods/unit/network/subunits/mst_spt/mst_demo/mst1.html, viimati vaadatud 12.05.2016

[13] https://hackage.haskell.org/package/gloss, viimati vaadatud 11.05.2016

[14] https://github.com/dmsh/ocaml-xlib, viimati vaadatud 07.05.2016

[15] http://www.ocamljava.org/, viimati vaadatud 07.05.2016

[16] http://pauillac.inria.fr/camlimages/, viimati vaadatud 07.05.2016

[17] https://bitbucket.org/camlspotter/camlimages/src/9430c29c6ff13b4f550d12a739b59081128df1ad/INSTALL.rst?at=default&fileviewer=file-view-default, viimati vaadatud 07.05.2016

[18] https://github.com/johnwhitington/graphicspdf, viimati vaadatud 07.05.2016

[19] http://www.coherentpdf.com/ocaml-libraries.html, viimati vaadatud 07.05.2016

[20] ftp://ftp.funet.fi/pub/TeX/CTAN/support/epstopdf/epstopdf.man1.pdf, viimati vaadatud 11.05.2016

[21] https://github.com/ocaml/ocaml/blob/trunk/otherlibs/graph/text.c, viimati vaadatud 11.05.2016

[22] https://www.tug.org/docs/metapost/mpman.pdf, viimati vaadatud 11.05.2016

Lisad

I. Lähtekood

Lähtekood on saadaval lisana ning aadressil https://github.com/hiievill/Algoritmid. Lisatud failid sisaldavad OCamlis kirjutatud koodifaile, ülevaatlikku tutvustusfaili readme.md ja kahte näidisena mõeldud sisendfaili Andmed.txt ja Andmed2.txt.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Hiie Vill**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose **Graafialgoritmide visualiseerimine**,

   mille juhendaja on Härmel Nestra,

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **12.05.2016**