Tartu Ülikool
Arvutiteaduse instituut
Informaatika õppekava

Erik Presnov

Graafialgoritmide läbimängija ja hindaja

Bakalaureusetöö (9 EAP)

Juhendaja: Ahti Põder, PhD

Tartu 2024

Graafialgoritmide läbimängija ja hindaja

Lühikokkuvõte:

Töö eesmärk on aine "Algoritmid ja andmestruktuurid" jaoks graafialgoritmide läbimängimise ja hindamise programmi loomine. Praeguseni on läbimängu tehtud paberil, mille miinuseks on kontrollimise raskus ning raske ülesande ülesseadmine ja joonise korrashoid. Loodud programmi abil näeb kasutaja peale vea tegemist kohe, kus viga on ning saab selle parandada, et edasine läbimäng selle vea tõttu valesti ei läheks. Lisaks sellele säästab loodud programm õppejõu aega graafialgoritmide õpetamisel ja algoritmide läbimängude kontrollimisel.

Võtmesõnad: Algoritmid ja andmestruktuurid, graafid, õpitarkvara

CERCS: P175 Informaatika, süsteemiteooria

Playthrough simulator for graph algorithms

Abstract:

The purpose of this thesis is to create an algorithm playthrough tool for the course "Algorithms and data structures". Up until now, the playthrough of the algorithms has been done on paper, which has the downsides of being hard to check, hard to set up and difficult to keep the drawing clean. With the help of the created tool, the user will be able to immediately see that a mistake has been made and fix it, so the further playthrough of the algorithm does not get spoiled. In addition to that, the created tool helps teaching assistants save time during the teaching and grading process.

Keywords: Algorithms and data structures, graphs, educational software

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

“Algoritmid ja andmestruktuurid” (edaspidi AA) kursusel õpetatakse enamlevinud andmestruktuure ja nendega teostatavaid algoritme. Iga algoritmi käsitletakse nii teoreetiliselt kui ka praktiliselt. Praktiline pool tähendab algoritmi käigu samm-sammulist läbimängimist mingi sisendiga. Õpetatavad teemad jagunevad neljaks suuremaks osaks: massiivid, paisktabelid, puud ja graafid. Pikemas perspektiivis on eesmärk luua iga teema jaoks sisendite genereerimise programm ja läbimängimise programm. Seni on valminud paisktabelialgoritmide läbimängija [1] ning sisendi generaatorid paisktabelialgoritmidele [2], graafialgoritmidele [3] ning massiivialgoritmidele [4].

Selle töö eesmärk on luua graafialgoritmide läbimängimiseks ja automaatseks hindamiseks prototüüp. See programm on vajalik, sest praeguseni on tehtud algoritmide läbimängimist paberil, mille miinuseks on raske korrektsuse kontroll ja palju tehnilist kirjutamist. Loodud programmiga säästab kasutaja aega ülesande ülesseadmisega ning saab kohest tagasisidet tehtud vigade korral, mis ei lase vigadel kuhjuda. Lisaks sellele teeb loodud programm igas algoritmis kasutatava andmestruktuuri haldamise automaatseks, mis omakorda säästab kasutaja aega ning laseb kasutajal keskenduda läbimängitava graafialgoritmi sisule.

Esimeses peatükis analüüsitakse olemasolevaid sarnaseid lahendusi ning nende puudujääke. Teises peatükis antakse ülevaade käsitletavatest algoritmidest. Kolmandas peatükis on kirjeldatud programmi nõuded. Neljandas peatükis kirjeldatakse läbimängu käiku programmis, läbimängu korrektsuse hindamist ning erinevaid veatüüpe. Viimases peatükis kirjeldatakse loodud programmi testimist.

1. Sarnaste lahenduste analüüs

Praeguse seisuga puudub avalik keskkond või tarkvara, millega saaks mugavalt graafialgoritmide läbimängimist teha. Natuke lahjem variant läbimängimis keskkonnast on visualiseerija ning neid leidub graafialgoritmidele päris palju.

Visualisatsioone saab vaadata näiteks VisuAlgo kodulehel[1] aga selle halvaks küljeks on algoritmide puudumine. Näiteks ei ole eeldusgraafi analüüsi, Floyd-Warshalli algoritmi ning sügavuti läbimisel ei saa valida ees- või lõppjärjestust. Lisaks sellele ei saa olemasolevaid algoritme jooksutada enda sisenditel vaid visualiseerija pakub sisendiks ainult enda genereeritud graafi. Viimaseks murekohaks on see, et visualiseerija näitab potentsiaalselt mitut sisemiselt kasutatavat andmestruktuuri vaheldumisi ning siis on nende jooksvat seisu keeruline jälgida.

Teine koht, kus saab visualisatsioone vaadata on Varssavi Ülikoolis loodud visualiseerijas[2], kuid ka sellel on puudusi. Selles on olemas ainult neli algoritmi ning nagu ka eelmise visualiseerijaga ei saa sügavuti läbimisel valida ees- ja lõppjärjestuse vahel. Selle visualiseerija eelised on järgmised. Esiteks saab kasutaja ise sisendi anda. Teiseks see, et sisemiselt kasutatav andmestruktuur on kogu aeg nähtav. Hea on ka see, et järgmist sammu näidatakse peale nupu vajutust ehk kasutaja saab praegust seisu uurida nii kaua kui vaja ning siis edasi liikuda. Kasulik on ka see, et igal sammul on põhjalik kirjeldus juures, miks mingi otsus või arvutus tehakse.

Kolmas koht, kus saab visualisatsioone vaadata on San Francisco Ülikoolis loodud visualiseerijas[3], aga ka sellel on puudusi. Puudub eeldusgraafi analüüs ning kasutaja ei saa algoritme enda sisendiga jooksutada vaid saab alati sisendiks mingi genereeritud graafi. Veel üks miinus selle visualiseerija juures on see, et mõned andmestruktuurid on päris arusaamatult või ebaintuitiivselt kuvatud, vt lisa 1. Selle visualiseerija puhul on hea see, et visualisatsiooni saab vaadata automaatselt, ise visualisatsiooni kiirust valides või ise igat sammu edasi klikkides.

Kõigil eelnevalt analüüsitud visualiseerijatel on mingi kombinatsioon ja raskusaste järgmistest puudustest.

- Algoritmide puudumine, kõik AA kursusel õpitavad algoritmid ei ole kaetud.

[1] https://visualgo.net/en
[2] https://www.mimuw.edu.pl/~erykk/algovis/graphs.html
[3] https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

- Andmestruktuuri(de) ebaselge visualiseerimine või visualiseerimise puudumine.
- Kasutaja ei saa sisestada enda graafe ehk visualiseerija genereerib mingi suvalise graafi, mis ei pruugi olla väga hea sisend mingile kindlale algoritmile.

Visualiseerijad annavad üldise ülevaate algoritmi tööst kuid algoritmi omandamise vaatenurgast ei ole visualisatsiooni vaatamine nii efektiivne, kui ise algoritmi töö läbi mängimine. Selletõttu on vaja keskkonda, kus kasutaja (tudeng) saab ise valitud graafialgoritmi tööd läbimängida. Lisaks sellele saavad õppejõud loodud programmi abil kergemini käsitletavaid graafialgoritme õpetada.

2. Mõisted ja terminid

Suunamata graaf on paar $G = (V, E)$, kus V on mittetühi tippude hulk ning E servade hulk, mille elementideks on hulga V kaheelemendilised alamhulgad [5].

Suunatud graaf on paar $G = (V, E)$, kus V on mittetühi tippude hulk ning E kaarte hulk, mille elementideks on hulga V järjestatud kaheelemendilised alamhulgad [5].

Kaalutud graaf on graaf G, mille igale kaarele (või servale) vastab reaalarv, selle kaare (või serva) kaal [5].

Sidusa suunamata graafi $G = (V, E)$ **toeseks** ehk **toespuuks** nimetatakse sidusat tsükliteta graafi (V, T), kus $T \subseteq E$. Graafi sidusate komponentide **toeste hulka** nimetatakse graafi **toesmetsaks** [6].

Tipp u on **intsidentne** servaga e kui tipp u kuulub servale e [5].

Tippe u ja v nimetatakse **naabertippudeks** kui nad on servaga ühendatud [5].

Kaare (u, v) tippu u nimetatakse **algtipuks** ja tippu v **lõpptipuks** või **järglaseks** [5].

Järjekord on järjestatud andmestruktuur, kust võetakse elemente välja samas järjestuses nagu need lisati [6].

Magasin on järjestatud andmestruktuur, kust võetakse esimesena välja sinna viimasena lisatud element [6].

Eelistusjärjekord on andmestruktuur, kus igal **elemendil (kirjel)** on mingi **väärtus (võti)**, mille järgi elemendid on järjestatud [6].

Muutuvate eelistustega eelistusjärjekord on eelistusjärjekord, kus kirjete võtmeid on töö ajal võimalik muuta [6].

Graafis G nimetatakse **ahelaks** tippudest $v_0, v_1, .., v_k$ koosnevat järjendit, kus iga kaks järjestikust tippu on servaga (või kaarega) ühendatud [5].

Graafi G nimetetakse **sidusaks**, kui iga kahe tipu korral leidub neid tippe ühendav ahel [5].

Graafi **sidusus komponentideks** nimetatakse graafi eraldiseisvaid osi, mis omaette on sidusad graafid [5].

3. Algoritmide tutvustus

Järgnevalt tutvustatakse lühidalt käsitletud algoritme ning kirjeldatakse, mis omadustega graafi mingi kindla algoritmi jaoks vaja on ja mis on selle algoritmi puhul raskusparameeter.

Raskusparameeter on mingi numbriline väärtus (või mitu), mis kirjeldab kui raske (see tähendab kui palju tööd tuleb teha) mingit algoritmi antud graafil läbi mängides. Näiteks nagu Uku Hannes Arismaa lõputöös [2] kirjutas, võiks Kahni algoritmi puhul olla raskusparameetriteks kaarte arv ja tippude arv. Algoritmidele, mida ei käsitletud Uku Hannes Arismaa lõputöös [2], pakutakse välja, mis võiks olla raskusparameeter.

Järgnevad algoritmide kirjeldused tuginevad Jüri Kiho õpikule [6].

3.1 Laiuti läbimine

Laiuti läbimise idee on töödelda graafi mingist algtipust alates laiuti ehk tasemete kaupa, see tähendab hoides töödeldava tipu kaugust (siin algoritmis on kauguseks lühima tee kaarte arv algtipust töödeldavasse tippu) võimalikult väiksena. See tähendab, et esimesena töödeldakse algtipp, siis tipud kaugusel 1, seejärel tipud kaugusel 2 jne.

Algoritmi töö käik on järgmine. Tipu järglased lisatakse järjekorda ning tipp märgitakse töödelduks. Järjekorrast võetakse töötlemiseks järgmine tipp. Algoritm lõpetab töö kui järjekord saab tühjaks.

Laiuti läbimisel võiks olla sisendi genereerimisel raskusparameetriteks tippude arv ning nn heade ja halbade kaarte arv. Kaart loetakse *heaks*, kui kaare lõpptipp on võrreldes kaare algtipuga kaugemas tasemes, see tähendab, et seda kaart on kerge töödelda. Kaart loetakse *halvaks*, kui kaare lõpptipp on võrreldes kaare algtipuga samas või varasemas tasemes, sest siis on oht teha viga.

3.2 Sügavuti läbimine

Sügavuti läbimise idee on liikuda algtipust võimalikult sügavale ehk kaugele. Erinevalt laiuti läbimisest, kus praegu töödeldava tipu järglased lähevad järjekorra lõppu, siis sügavuti läbimisel pannakse kõik töötlemata järglased magasini (sisuliselt tööjärjekorra algusesse).

Algoritmi töö käik on järgmine. Tipu järglased lisatakse magasini ning tipp märgitakse töödelduks (eesjärjestuses) või ootele (lõppjärjestuses). Magasinist võetakse töötlemiseks järgmine tipp. Juhul kui tegu on lõppjärjestuses läbimisega siis tuleb õigel hetkel (siis kui

kõik järglased saavad töödelduks) muuta tipu seis ootavast töödelduks. Algoritm lõpetab töö kui magasin saab tühjaks.

Sügavuti läbimisel võiks olla raskusparameetriteks tippude arv, kaarte arv ning kaarte arv, mis viivad juba töödeldud või töötlusmagasinis olevasse tippu.

3.3 Primi algoritm

Primi algoritm leiab graafi ühe minimaalse kaaluga toespuu. Algoritmi tööpõhimõte on järgmine. Alustatakse suvalisest tipust a (võib alustada suvalisest, sest toespuu peab sisaldama igat tippu). Kõik töödeldava tipuga intsidentsed servad lisatakse eelistusjärjekorda. Töödeldav tipp märgitakse töödelduks. Eelistusjärjekorrast võetakse vähima kaaluga serv e. Kui serva e lisamine tekitab tsükli, siis seda serva jooksvalt ehitatavasse toespuusse ei lisata. Kui serva e lisamine ei tekita tsüklit (see tähendab, et üks serva otstippudest on töötlemata), siis lisatakse serv e toespuusse ning serva töötlemata otstipp on järgmine töödeldav tipp. Algoritm lõpetab töö kui eelistusjärjekord saab tühjaks.

Primi algoritmi sisend peab olema sidus, see tähendab suvalisest tipust on võimalik jõuda igasse teise tippu. See on vajalik sellepärast, et juhul kui graafis on rohkem kui üks sidususkomponent ning esimene (suvaliselt valitud) tipp on ühes sidususkomponendis, siis algtipuga intsidentseid servi valides ei ole võimalik jõuda teise sidususkomponenti, seega ei ole võimalik mittesidusale graafile Primi algoritmi rakendada. Primi algoritmi puhul võiks olla raskusparameetrid tippude ja servade arv. Paberil läbimängides tuleb eelistusjärjekorda hallata manuaalselt.

3.4 Kruskali algoritm

Kruskali algoritm leiab graafi ühe minimaalse kaaluga toespuu (või toesmetsa). Algoritmi tööpõhimõte on järgmine. Graafi kõik servad lisatakse eelistusjärjekorda. Igat graafi tippu käsitletakse omaette puuna, seega kogu graafi käsitletakse kui ühetipuliste puude metsa. Eelistusjärjekorrast hakatakse servi välja võtma. Kui serv ühendab kahte erinevat puud, siis lisatakse serv toespuusse (või toesmetsa). Juhul kui serv ühendab samas puus olevaid tippe, siis seda toesesse (või toesmetsa) ei lisata, sest see tekitaks tsükli, mis on vastuolus puu definitsiooniga. Algoritm lõpetab töö kui eelistusjärjekord saab tühjaks.

Kruskali algoritmi puhul võiks olla raskusparameetriks kaarte arv ja tsükli tekkimise võimaluste arv.

3.5 Dijkstra algoritm

Dijkstra algoritm leiab graafis lühimad teed (siin algoritmis on tegu kaalutud graafiga seega lühim tee tähendab ahela $a \rightarrow ... \rightarrow x$ minimaalset kaalu, kus ahela $a \rightarrow ... \rightarrow x$ kaal on kõikide ahelasse kuuluvate kaarte summa) algtipust a kõikidesse teistesse tippudesse. Algoritmi tööpõhimõte on järgmine. Kontrollitakse kõiki praegu töödeldavast tipust väljuvaid kaari. Juhul kui kaar viib juba töödeldud tippu, ei tehta midagi. Juhul kui kaar viib tippu, mis on avastamata, lisatakse kaare lõpptipp muutuvate eelistustega eelistusjärjekorda (võtmega kaugus praegusesse tippu + kaare kaal). Juhul kui kaar viib tippu, mis on juba eelistusjärjekorras olemas, siis kontrollitakse, kas tee läbi praeguse tipu, see tähendab kaugus praegusesse tippu + kaare kaal on lühem kui parim senine tee (kirje praegune võti eelistusjärjekorras). Kui on, siis teostatakse antud kirje eelistuse muutmine tööjärjekorras Q (kui Q on realiseeritud kuhjana, siis kuhja parandus).

Dijkstra algoritmi sisend peab olema positiivsete kaaludega graaf. Positiivsed kaalud on vajalikud sellepärast, et algoritmi korrektsus põhineb eeldusel, et juba töödeldud tipu kaugus ei saa väheneda. Juhul kui graaf sisaldab negatiivsete kaaludega servi, siis võib tekkida olukord, kus leiame lühima tee tippu x ning hiljem avastame, et kuskilt mujalt on võimalik veel samasse tippu x jõuda, kasutades negatiivse kaaluga serva, mis võib anda tulemuseks väiksema kaalu, kui see, mille algoritm algselt leidis. Selle algoritmi puhul on raskusparameetriteks tippude arv, kaarte arv ja eelistusjärjekorra paranduste arv [2].

3.6 Floyd-Warshalli algoritm

Floyd-Warshalli algoritm leiab graafis lühimate teede pikkused kõikide tippude vahel. Algoritm töötab järgmiselt. Fikseeritakse tipp a ning selle tipu kohta vaadatakse läbi kõikvõimalikud tipupaarid (b, c) kus $b \in V \backslash \{a\}$ ja $c \in V \backslash \{a\}$ ning tehakse kontroll, kas tee $b \rightarrow ... \rightarrow a \rightarrow ... \rightarrow c$ on parem kui $b \rightarrow ... \rightarrow c$. Juhul kui on, siis tehakse kaugusmaatriksis vastav parandus. Sama protseduuri korratakse iga tipu jaoks ehk iga tipp peab olema a rollis.

Selle algoritmi puhul on raskusparameetriteks tippude arv, kaarte arv ja kaugusmaatriksi paranduste arv [2].

3.7 Bellman-Fordi algoritm

Bellman-Fordi algoritm leiab graafis lühimad teed algtipust a kõikidesse teistesse tippudesse. Algoritm töötab järgmiselt. Iga tipu t kohta määratakse algselt kauguseks lõpmatus ehk kaugus sinna tippu on teadmata. Graafi kõik servad kogutakse järjekorda. Järjekorrast

võetakse serv e. Kui serva e lõpptipp t on kaugusega lõpmatus ehk teadmata, siis uuendatakse selle tipu kaugust. Kui serva e lõpptipu t kaugus ei ole lõpmatus, siis kontrollitakse, kas tee lõpptippu t on serva e kasutades lühem kui praeguseni lühim tee, kui on, siis parandatakse kaugust. Algoritm töötab seni, kuni läbitakse kõik servad ning ühtegi kauguse parandust ei tehta.

Selle algoritmi puhul on raskusparameetriteks tippude arv, kaarte arv ja kauguste puu kõrgus [2], välimise tsükli läbimiste arv. Selleks, et välimise tsükli läbimiste arvu raskusparameetrina kasutada, tuleb kaarte töötlemise järjekord fikseerida, sest sõltuvalt kaarte töötlemise järjekorrast, tuleb välimist tsüklit läbida 1 kuni $n - 1$ korda, kus n on graafi tippude arv.

3.8 Kahni algoritm

Kahni algoritm leiab graafi ühe topoloogilise järjestuse. Algoritmi tööpõhimõte on järgmine. Iga tipu kohta loetakse kokku, mitmel kaarel on antud tipp lõpptipuks ehk mis on selle tipu sisendaste. Kõik tipud, mille sisendaste on null, lisatakse järjekorda. Seejärel võetakse järjekorrast järgmine töödeldav tipp. Töödeldakse kõik sellest väljuvad kaared, see tähendab vähendatakse väljuva kaare lõpptipu sisendastet, ning tipp ise märgitakse töödelduks. Juhul kui mingi kaare töötlemise puhul jõuab selle lõpptipu sisendaste nullini, lisatakse kaare lõpptipp järjekorda. Algoritm lõpetab töö kui järjekord saab tühjaks.

Kahni algoritmi sisend peab olema tsükliteta, sest vastasel juhul ei saa ühtegi tippu, mis tsüklis on, töödelda. Kui graafis leidub tsükkel, siis saab järjekord tühjaks, kuid saadud topoloogiline järjestus ei sisalda kõiki tippe. Selle algoritmi puhul on raskusparameetriteks kaarte arv ja tippude arv [2].

3.9 Eeldusgraafi analüüs

Eeldusgraafi analüüsi eesmärk on leida eeldusgraafis kogu graafi ehk projekti varaseim lõpuaeg, iga tipu ehk töö varaseim lõpuaeg ja hiliseim algusaeg ning kriitilised tipud. Algoritmi töö käik on järgmine. Leitakse graafi suvaline topoloogiline järjestus. Läbitakse tipud topoloogilises järjestuses ja leitakse nende varaseimad lõpuajad. Selleks võetakse eellaste seast kõige hiljem lõpetav tipp ning tema ajale liidetakse juurde tipu enda tööaeg. Juhul kui tipul ei ole eellasi, siis tema varaseim lõpuaeg on tema enda tööaeg. Siit saadakse ka kogu projekti varaseim lõpuaeg. Seejärel läbitakse tipud vastupidiselt topoloogilisele järjestusele ja leitakse nende hiliseimad algusajad. Selleks võetakse järglaste seast kõige varem alustav tipp ning tema ajast lahutatakse tipu enda tööaeg. Juhul kui tipul ei ole järglasi,

siis lahutatakse tipu tööaeg maha kogu projekti varaseimast lõpuajast. Nüüd leitakse kriitilised tipud. Selleks kontrollitakse, kas tipu hiliseima algusaja ja tipu enda tööaja summa on võrdne varaseima lõpuajaga, kui on, siis tipp on kriitiline. Töö kriitilisus tähendab, et selle tööga peab alustama kohe kui võimalik, muidu ei ole võimalik kogu projekti leitud varaseimaks lõpuajaks valmis saada..

Eeldusgraafi analüüsi saab teostada graafil, millel leidub topoloogiline järjestus (see tähendab graafis puuduvad tsüklid). See on vajalik selleks, et juhul kui graafis oleks mingi tsükkel (näiteks $a \rightarrow b \rightarrow a$) siis tekiks tsükliline sõltuvus. See tähendaks seda, et töö a tegemiseks on vaja ära teha töö b, mis omakorda sõltub tööst a. Lisaks sellele, peavad tipu tööajad olema positiivsed. Vastasel juhul tekiks olukord, kus mingi tööga peab tulevikus alustama selleks, et see õigeks ajaks valmis saada (valemi $TipuAlgusAeg + TipuTööAeg = TipuLõppAeg$ puhul näiteks 13 + (-3) = 10). Eeldusgraafi analüüsi puhul on raskusparameetriteks kaarte arv, tippude arv ning keskmine tipu järglaste arv ning tipu eellaste arv.

4. Nõuded programmile ja programmi kasutamine

Eesmärgiks oli luua graafialgoritmide läbimängimiseks programm. Nõuded loodavale programmile saab jagada kaheks: visuaalseteks ja funktsionaalseteks.

4.1 Visuaalsed nõuded

Programmi loomisel oli esimene visuaalne nõue see, et erinevate algoritmide läbimängud näeksid visuaalselt võimalikult sarnased välja. See on vajalik selleks, et programm oleks võimalikult kasutajasõbralik. Kui juhend ära lugeda ja aru saada, kuidas programmi kasutada, näiteks laiuti läbimise läbimängimiseks, siis võiks teiste algoritmide läbimäng programmi kasutamise mõttes olla juba intuitiivne.

Päris ühetaoliseks läbimänge visuaalselt teha ei saa, sest olenevalt algoritmist on graafid erinevate omadustega, näiteks suunatud või suunamata, servad kaaludega või ilma, tipud kaaludega või ilma, ja kasutavad erinevaid andmestruktuure, näiteks järjekord, eelistusjärjekord, magasin, paisktabel, maatriks. Mõne algoritmi (näiteks minimaalse kaaluga toespuu leidmise algoritmid) puhul on ka oluline see, milline serv valiti toesesse ja milline mitte, seega läbimängu käigu jooksul muudetakse mõnede servade värvi vastavalt tehtud otsustele.

Teine visuaalne nõue oli see, et andmestruktuurid näeksid võimalikult lihtsad ja arusaadavad välja. Eesmärk on see, et andmestruktuuri nime ja kirjete struktuuri järgi on lihtne aru saada, mida mingi kirje tähendab.

4.2 Funktsionaalsed nõuded

Esimeseks funktsionaalseks nõudeks oli see, et kasutaja opereerib algoritmis kasutatava andmestruktuuriga nii abstraktselt kui võimalik. See tähendab, et enamus algoritmide puhul saab kasutaja asju kasutatavasse andmestruktuuri ainult lisada (klõpsates kaarele või nupule) ja eemaldada (nupp kasutatava andmestruktuuri kõrval). Juhul kui kasutusel on midagi muud peale magasini või järjekorra, näiteks Dijkstra algoritmi puhul eelistusjärjekord tippudest või Kruskali algoritmi puhul eelistusjärjekord servadest, teeb programm andmestruktuuri sisemise töö automaatselt ära. See on hea seetõttu, et graafialgoritmi harjutamisel ei pea andmestruktuuri haldamisega ise tegelema, mis elimineerib vea tegemise ohu, näiteks kuhja loomisel või parandamisel.

Teiseks funktsionaalseks nõudeks oli see, et programm logib iga läbimängu kohta järgneva info: mis graafiga, millist algoritmi läbi mängiti ja iga kasutaja poolt tehtud samm ning

vigase sammu puhul ka see, mis valesti oli. Selline funktsionaalsus on vajalik selleks, et juhul kui kasutaja jääb mingi läbimänguga hätta või tahab lihtsalt õppejõult täpsustust millegi kohta küsida, siis antud logi põhjal peaks olema väga kerge see antud seis, kus viga või küsimus tekkis, rekonstrueerida.

Kolmas nõue oli see, et kasutaja saab läbimängu teha suvalise, mingite raskusparameetrite järgi genereeritud, sisendiga või anda algoritmile sisendiks endaloodud graafi.

4.3 Programmi kasutamine

Programm laseb kasutajal valida soovitud algoritmi ehk vahetada erinevate vahelehtede vahel. Peale soovitud algoritmi (vahelehe) valimist kuvatakse kasutajale väike tutvustus (algoritmi nimi, kasutatav(ad) andmestruktuur(id) ning lühikirjeldus algoritmi töökäigust). Seejärel saab kasutaja nupu vajutusega graafi kuvada. Kuvatav graaf on mingite kindlate omadustega (suunatud või mitte, kaalutud või mitte), olenevalt kasutaja valitud algoritmist. Programmis paigutatakse algselt kõik tipud üksteise peale ning kasutaja saab need enda soovi järgi ümber liigutada. Selline lähenemine valiti selle pärast, et suvalise graafi paigutamine tasandile nii, et servade ristumisi on võimalikult vähe, on keeruline ülesanne ning selle lõputöö skoobist väljas. Vähene servade ristumine on vajalik selleks, et tegevusi graafil oleks kergem jälgida. Selline lähenemine on ka selle poolest hea, et juhul kui kaks kasutajat teevad sama graafiga sama algoritmi läbimängu, siis ei ole eriti tõenäoline, et mõlemad paigutavad graafi sama moodi. Seejärel saab kasutaja graafi nupu vajutusega “lukku panna” ning siis algoritmi läbimänguga pihta hakata, mille käik sõltub valitud algoritmist.

5. Graafialgoritmide läbimäng

“Algoritmid ja andmestruktuurid” kursusel õpitud graafialgoritme on tavaliselt paberil läbi mängitud. Selle miinuseks on lahenduse kontrollimise raskus ja joonise segaseks muutumise oht. Juhul kui avastatakse mitu sammu tagasi tehtud viga ja proovitakse joonisel või mingis andmestruktuuris midagi tagantjärgi parandada, võib asi kiiresti väga segaseks minna. Tihti avastatakse viga alles siis, kui jõutakse algoritmi läbimänguga lõpuni ning lõpptulemus erineb oodatust. Sellisel juhul on keeruline tuvastada, kas tegu oli mingi lihtsa arvutusveaga, mis rikkus tulemuse ära, või oli tegu mingi algoritmilise veaga.

5.1 Läbimängu kontrollimine

Lahenduse korrektsuse kontrollimiseks on sisuliselt kaks erinevat varianti.

- Läbimängu jooksul koostada list kõigist kasutaja poolt tehtud sammudest. Siis lasta arvutil sama sisendiga sama algoritm läbi teha ning selle sammud meelde jätta ning seejärel kahte sammude järjendit omavahel võrrelda.
- Läbimängu käigus jooksvalt kontrollida tehtud samme. Olenevalt algoritmist toimub kontroll natuke erinevatel ajahetkedel ja natuke erinevate objektidega, aga üldine idee on see, et praegu töödeldava tipu või kaarega peavad kõik vajalikud sammud korrektselt tehtud olema, ennem kui programm laseb kasutajal edasi minna.

Kuna paljude algoritmide puhul kasutatakse praegusel või eelmistel sammudel saadud tulemusi järgnevates sammudes, siis on esimene lähenemine halb. Juhul kui kasutaja teeb vea, näiteks viga arvutamisel või tehakse mingi operatsioon vale elemendiga, ning saadud tulemust kasutatakse hiljem mingis arvutuses või otsuses, siis võib lõpptulemus oodatust täiesti erinev olla. Lisaks sellele võib kasutajal jääda mingi samm üldse tegemata, see viiks läbimängu seisu olukorda, millesse ei tohiks jõuda ning nüüd kahe järjendi võrdlemisel on ühest üks samm puudu. Mis tähendab, et kõik sammud peale vahele jäänud sammu loetakse valeks. Lisaks sellele, võiks kohene tagasiside aidata kasutajal aru saada, mis valesti läks ning algoritmi paremini omandada.

Seega programm kontrollib peale igat kasutaja tehtud sammu, kas see oli õige. Juhul kui see on õige, lastakse kasutajal läbimängu jätkata. Juhul kui kasutaja tegi midagi valesti või jättis midagi tegemata, siis teavitab programm kasutajat, ning ei lase edasi liikuda ennem, kui viga saab parandatud või kõik vajalikud sammud tehtud.

5.2 Läbimängu võimalikud vead

Algselt oli plaanis jaotada vead kaheks erinevaks kategooriaks: lihtne ja kriitiline. Sellise jaotuse mõte oli see, et lihtne viga on midagi sellist, mis ei riku läbimängu tulemust ära ning kriitiline viga on midagi sellist, mis teeb läbimängu kohe katki. Näiteks Dijkstra algoritmis jäetakse praegu töödeldavast tipust üks väljuv kaar töötlemata. Võib juhtuda, et lühim tee töötlemata jäänud kaare lõpptipu realiseerub just selle sama töötlemata kaare kaudu. Sellisel juhul leiaks algoritm vale tulemuse.

Plaanis oli lihtsa vea puhul lasta läbimängu jätkata ning kriitilise vea korral kasutajat teavitada ning nõuda lahenduse parandamist. Selgus, et selliseid lihtsaid vigu, mille puhul võiks läbimängu jätkata, ei ole. Seega programm teavitab kasutajat iga veaga. See lähenemine on hea ka läbimängu korrektsuse tagamisel, sest programm ei lase kasutajal minna võimatusse seisu.

Mõned algoritmid küsivad kasutajalt kirjalikku sisendit, näiteks Dijkstra algoritmi puhul peab kasutaja iga töödeldava kaare korral sisestama kaare lõpptipu kauguse. Sellistes olukordades loetakse veaks ainult õiges formaadis antud vale sisend. See tähendab, et kui küsitakse tipu kaugust (täisarv) ning kasutaja sisestab näiteks sõne või tühiku või midagi muud sellist, mida ei saa täisarvuna interpreteerida, siis seda ei loeta veaks. Juhul kui oodatud kaugus on näiteks 7 ja kasutaja sisestab 9, ehk õiget tüüpi (täisarvulise) aga vale väärtuse, siis see loetakse veaks.

6. Valideerimine

Järgnevalt analüüsitakse ja põhjendatakse loodud programmi vastavust peatükis 4 välja toodud nõuetele.

Võrreldes näiteks Floyd-Warshalli algoritmi (vaata lisa 2) ja sügavuti eesjärjestuses läbimist (vaata lisa 3) läbimängu samal graafil selgub, et visuaalselt on need küllaltki sarnased. Floyd-Warshalli algoritm leiab tippude kaugused (kaarte kaalude summa) algtipust, seega on kaared kaaludega ehk iga kaare juures on mingi numbriline väärtus. Teine erinevus on see, et Floyd-Warshalli põhiline töö toimub graafi paremal asuvas kauguste maatriksis.

Teine visuaalne nõue oli see, et andmestruktuurid näevad võimalikult lihtsad välja. Näiteks Dijkstra algoritmis (vaata lisa 4) kui andmestruktuur on eelistusjärjekord (programmis kuhi) ja kirje on $E\colon 7$ siis on arusaadav, et tipp E on järjekorras kaugusega (eelistusega) 7.

Algoritmi läbimängu käigus saab kasutaja kasutatavasse andmestruktuuri elemente lisada ja eemaldada. Olenevalt algoritmist toimub lisamine mingile kindlale nupule vajutamisega (vaata lisa 5) või kaarele vajutades. Juhul kui algoritm (kõik peale eeldusgraafi analüüsi ja Floyd-Warshalli) nõuab andmestruktuurist eemaldamist, siis saab seda teha andmestruktuuri kõrval oleva nupuga (vaata lisa 6).

Lisaks sellele peab eelistusjärjekord sisemise töö automaatselt ära tegema (võtmeparandus kuhjas, elemendi paigutamine õigesse kohta). Lisad 7 ja 8 näitavad kuhja seisu enne ja pärast võtmeparandust. Algselt on kuhjas kirje $C\colon 3$. Seejärel töödeldakse tipp B ehk leitakse uus lühim tee tippu C ning peale tipu B töötlemist on kuhjas tipu C võtmeparandus (ja potentsiaalne elementide ümberpaigutamine) automaatselt tehtud. Lisad 9 ja 10 näitavad kuhja seisu enne ja pärast uute elementide lisamist. Peale uute kirjete ($BC\colon 1$ ja $BE\colon 6$) lisamist paigutatakse need automaatselt kuhjas õigetele kohtadele.

Järgmine nõue oli logimine. Lisa 11 näitab logi formaati. Logis on välja toodud graaf, millega algoritmi läbi mängiti ning seejärel kasutaja poolt tehtud sammud. Igal real on kirjas sammu number, mis tegevus tehti (näiteks “Lisan tipu X järjekorda”, “Kontrollin tippu X” vms) ning seejärel vea staatus. Rida lõppeb korrektse sammu puhul sõnaga “KORRAS” ning vigase sammu puhul sõnaga “VIGA”. Vigase sammu puhul on järgmisel real koos sama sammu numbriga kirjas, mis valesti tehti või mis tegemata jäi.

Viimane nõue oli see, et kasutaja saab suvalise, mingite raskusparameetrite järgi genereeritud, graafiga algoritme läbi mängida või anda sisendiks enda graaf. See osa tööst jäi

realiseerimata, sest praeguses graafialgoritmide sisendite generaatoris [2] on implementeeritud genereerimine nelja erineva algoritmi jaoks. Ülejäänud kuuele algoritmile käsitsi sisendite kirjutamine võtaks liiga palju aega ning automaatse lahenduse implementeerimine läheb selle töö skoobist välja. Praeguse lahendusega on kaasas kolm erinevate omadustega graafi, millega saab kõiki algoritmide läbimänge testida. Kaasas olevad graafid on:

- Suunatud graaf, mille servad on kaaludega.
- Suunatud graaf, mille servad on kaaludeta.
- Suunatud graaf, mille tipud on kaaludega (eeldusgraafi analüüsi jaoks).

Juhul kui algoritm kasutab gaafi, mis on suunamata, siis võetakse esimene või teine graaf, vastavalt sellele, kas servad peavad olema kaaludega või mitte, ning puuduvad kaared genereeritakse juurde. See tähendab, et kui näiteks esimeses graafis on kaar AB siis programm teeb automaatselt kaare BA juurde.

Kuna tegu on prototüübiga, siis testimine tehti autori ja mõningal määral ka juhendaja poolt. Asjaolu, mis takistas suuremamahulist kasutajapoolset testimist on see, et AA kursus toimub sügissemestril, aga lõputöö valmis kevadsemestril. Loodud prototüüpi on plaanis 2024 sügissemestril AA kursuse õpilastega testida ning vastavalt saadud tagasisidele kohendada.

Kokkuvõte

Töö eesmärgiks oli luua graafialgoritmide läbimängija ja hindaja. Selleks loodi graafilise kasutajaliidesega prototüüp, mis võimaldab kasutajal "Algoritmid ja andmestruktuurid" kursusel õpetatavaid graafialgoritme läbi mängida ning kohest tagasisidet saada. Lisaks sellele saavad loodud programmi abil õppejõud mugavamalt käsitletavaid algoritme õpetada.

Edasiarendusena saab mõne algoritmi puhul läbimängu käiku kasutajale mugavamaks teha. Kuna kasutaja mugavus ei olnud programmi loomisel nõue siis sellele eriti palju tähelepanu ei pööratud kuid mõne algoritmi (näiteks Floyd-Warshalli algoritm) läbimäng nõuab kasutajalt väga palju sisendeid.

Viidatud kirjandus

[1] Karolin Konrad. Paisktabelialgoritmide läbimängu automaatse hindaja loomine. Tartu Ülikooli arvutiteaduse instituudi bakalaureuse lõputöö, 2023. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=77834 (04.03.2024)

[2] Nikolai Voitsehhovski. Paisktabelialgoritmide sisendite genereerimine. Tartu Ülikooli arvutiteaduse instituudi bakalaureuse lõputöö, 2022. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=74478 (04.03.2024)

[3] Uku Hannes Arismaa. Graafialgoritmide sisendite genereerimine. Tartu Ülikooli arvutiteaduse instituudi bakalaureuse lõputöö, 2022. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=74594 (03.12.2023)

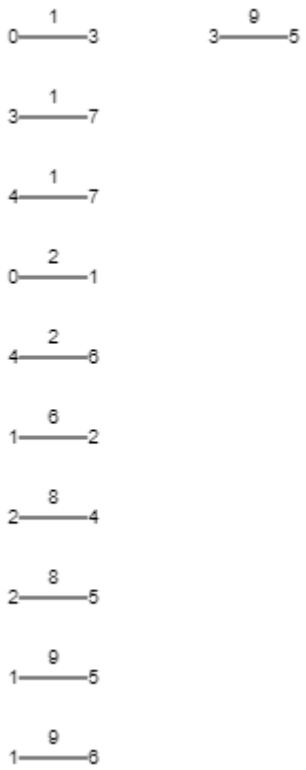
[4] Samuel Johannes Pitko. Massiivialgoritmide sisendite genereerimine. Tartu Ülikooli arvutiteaduse instituudi bakalaureuse lõputöö, 2021. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=72137 (04.03.2024)

[5] Reimo Palm. Diskreetse matemaatika elemendid. Tartu Ülikooli Kirjastus 2009. https://kodu.ut.ee/~reimo_p/teosed/dme/dme.pdf (03.12.2023)

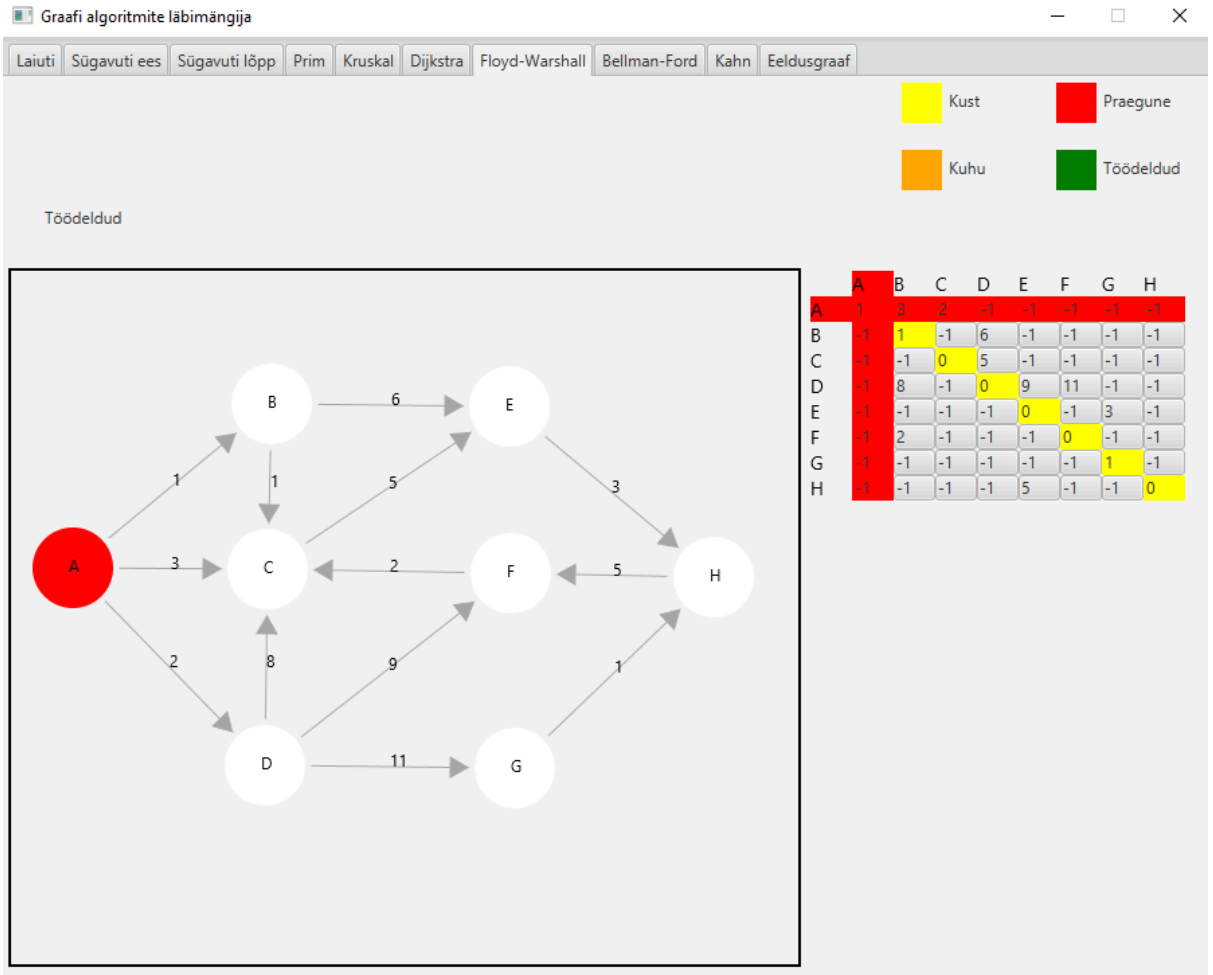
[6] Jüri Kiho. Algoritmid ja andmestruktuurid. Tartu Ülikooli Kirjastus 2003. https://moodle.ut.ee/pluginfile.php/76788/mod_resource/content/4/LOENG2011/ads2003.pdf (03.12.2023)

Lisad

1. Kuhja kujutamine San Francisco ülikoolis loodud visualiseerijas

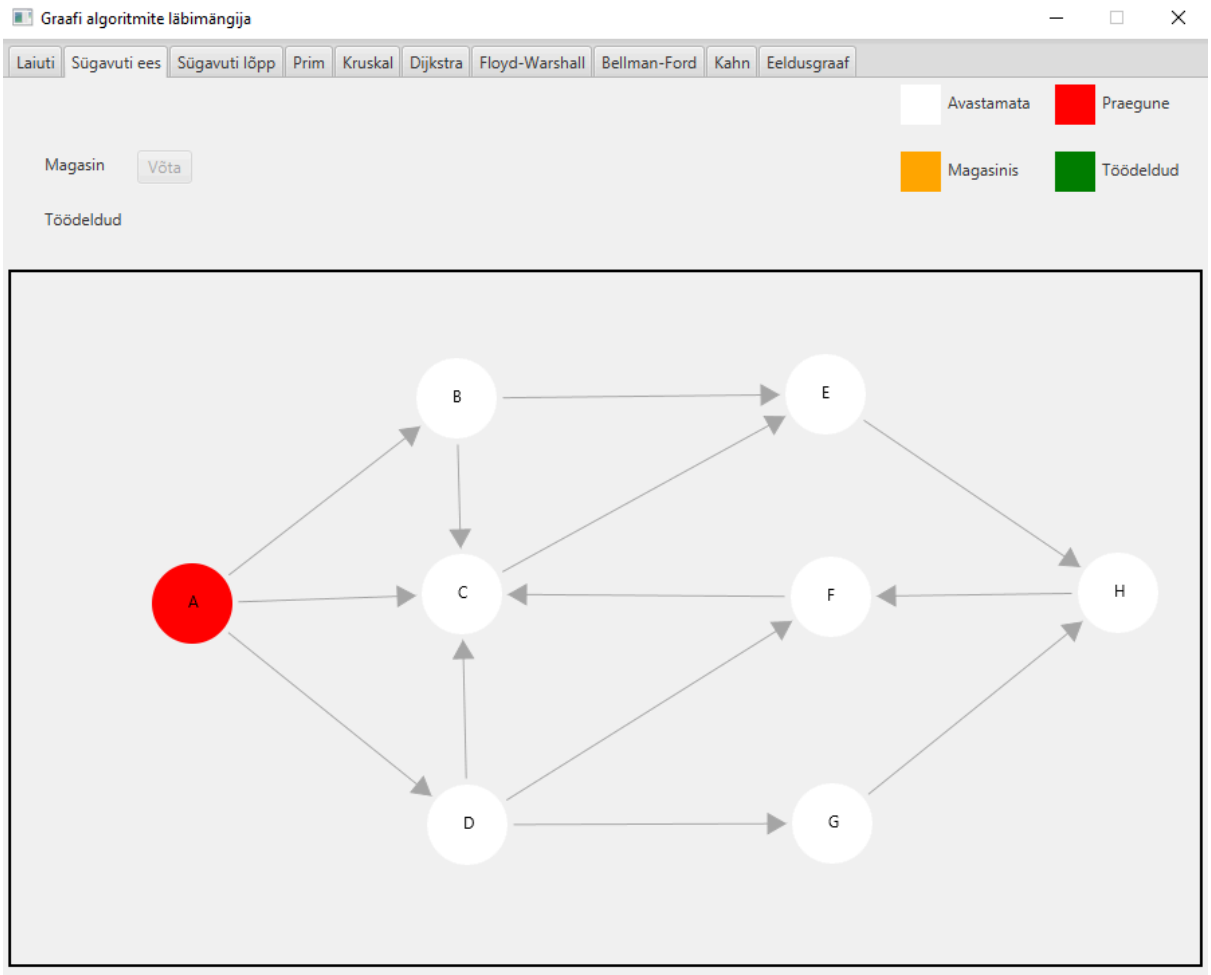


2. Floyd-Warshalli algoritmi vaade

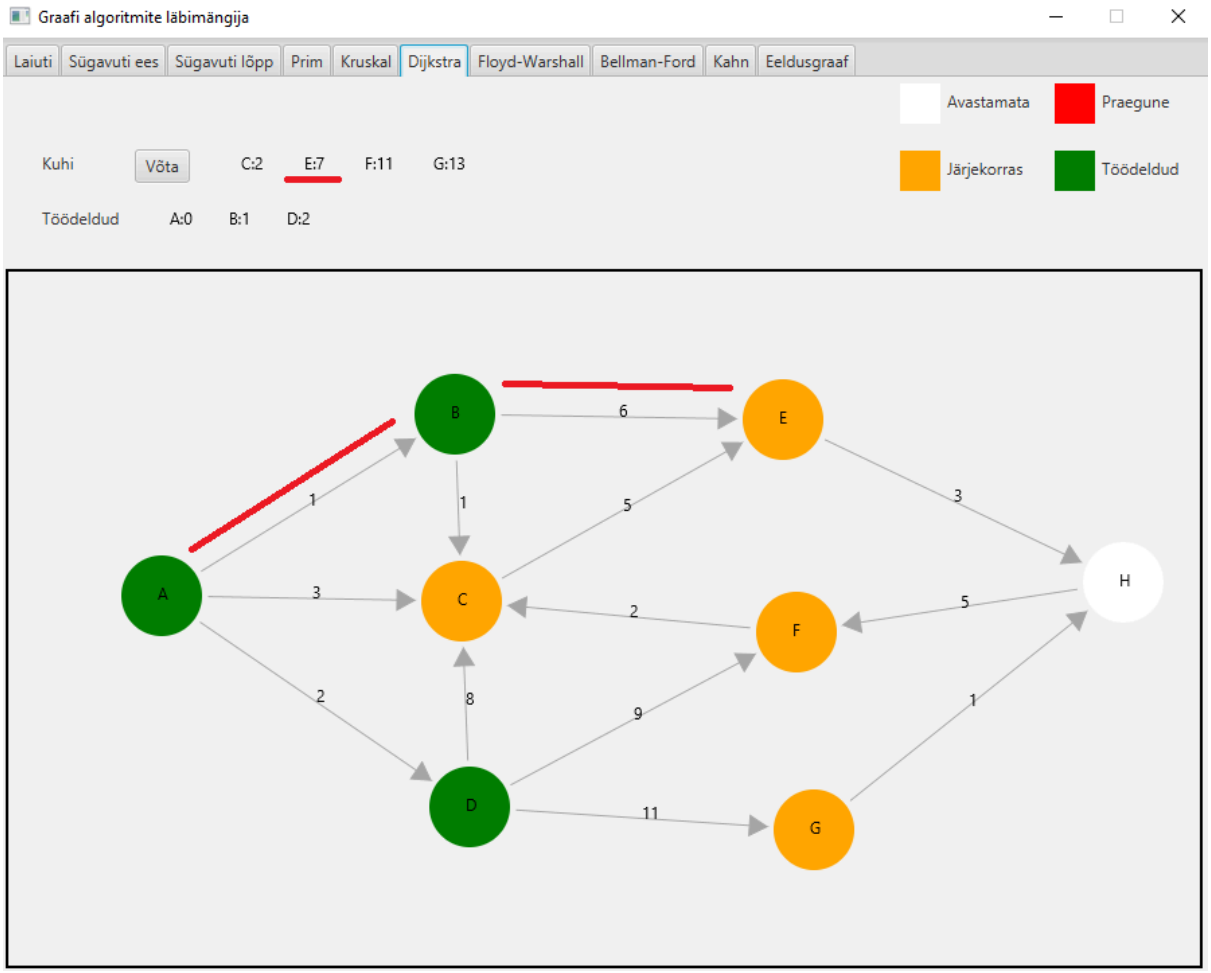


	A	B	C	D	E	F	G	H
A	1	3	2	-1	-1	-1	-1	-1
B	-1	1	-1	6	-1	-1	-1	-1
C	-1	-1	0	5	-1	-1	-1	-1
D	-1	8	-1	0	9	11	-1	-1
E	-1	-1	-1	-1	0	-1	3	-1
F	-1	2	-1	-1	-1	0	-1	-1
G	-1	-1	-1	-1	-1	-1	1	-1
H	-1	-1	-1	-1	5	-1	-1	0

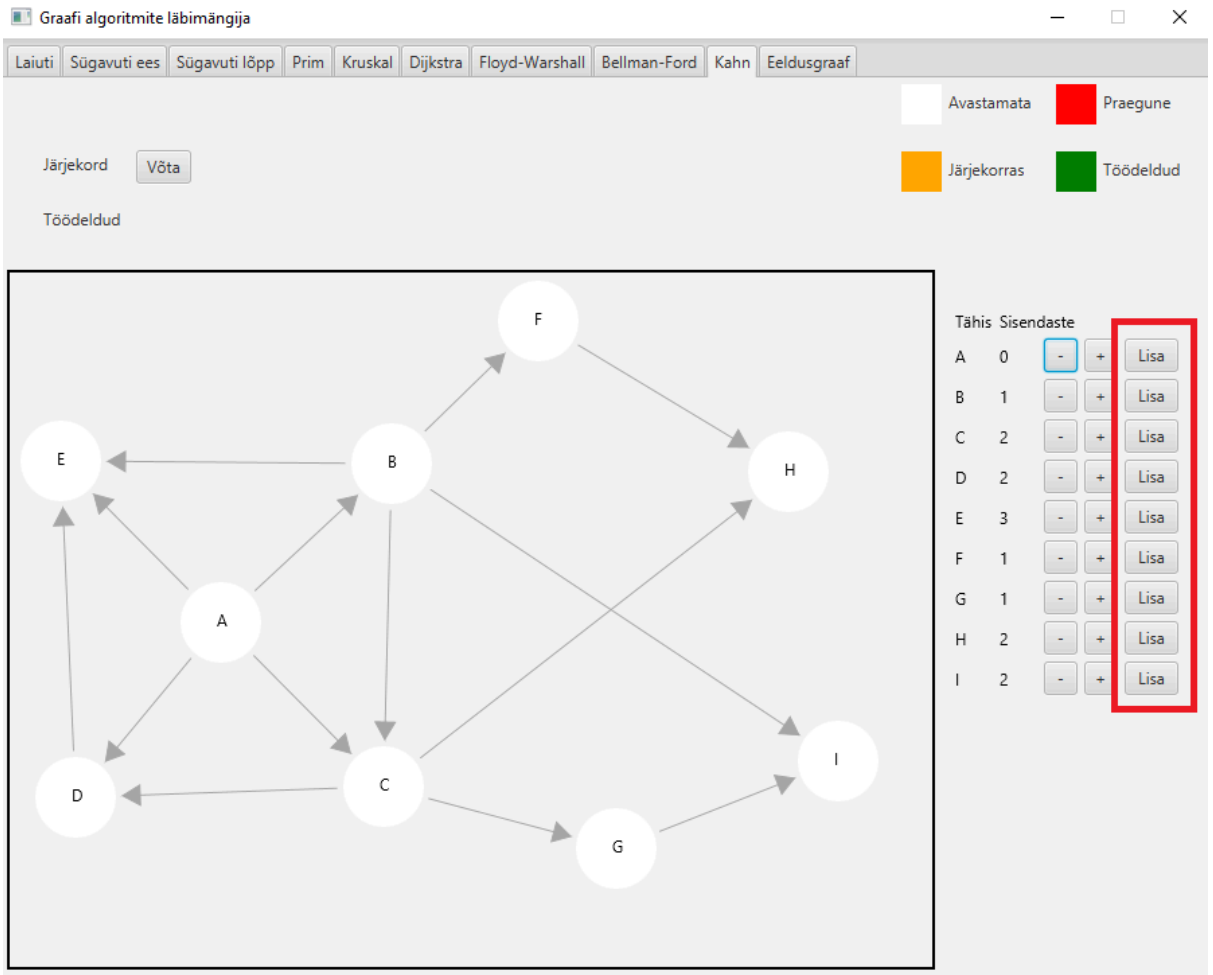
3. Sügavuti eesjärjestuses läbimise vaade



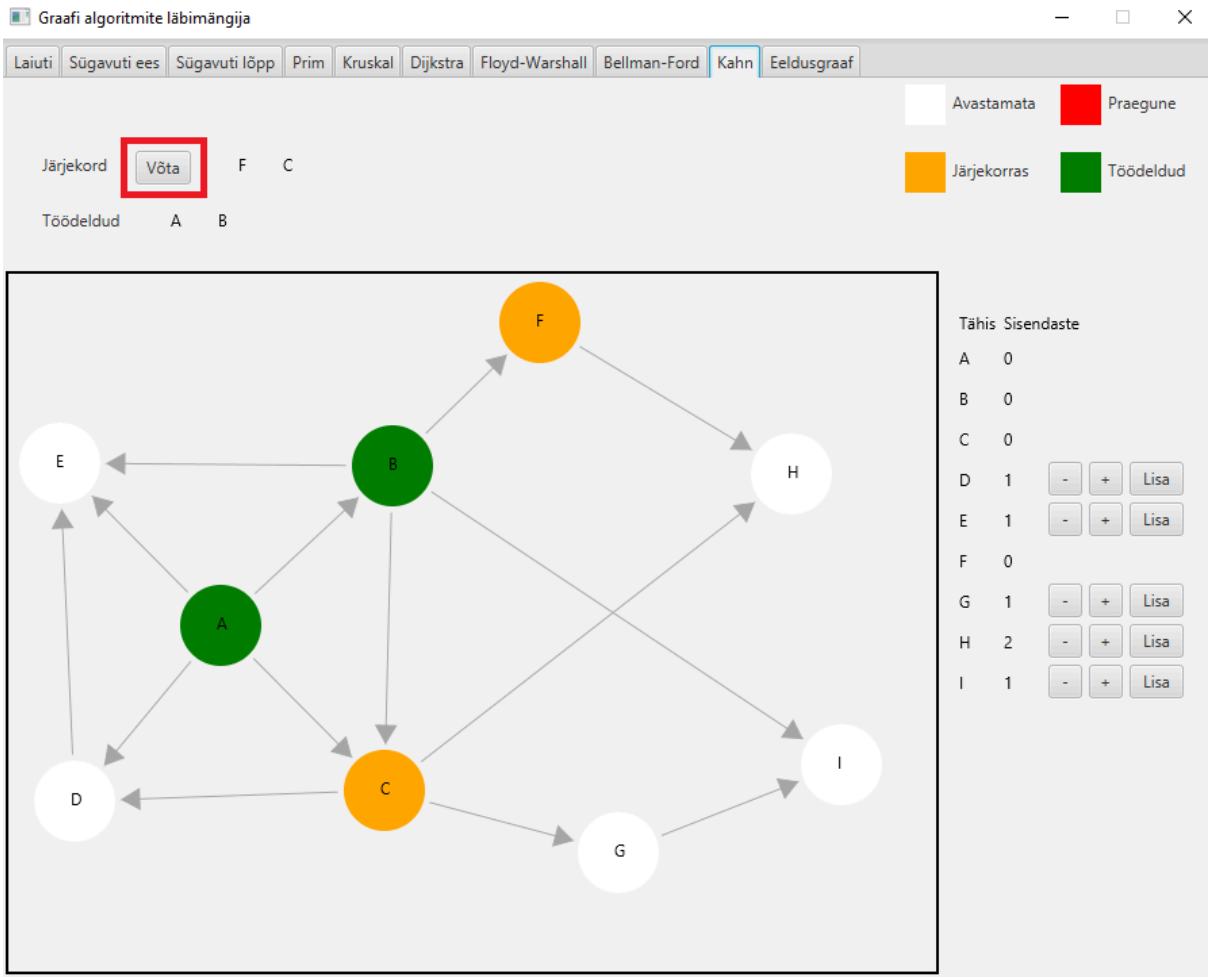
4. Kuhja kujutamine Dijkstra algoritmi töö käigus



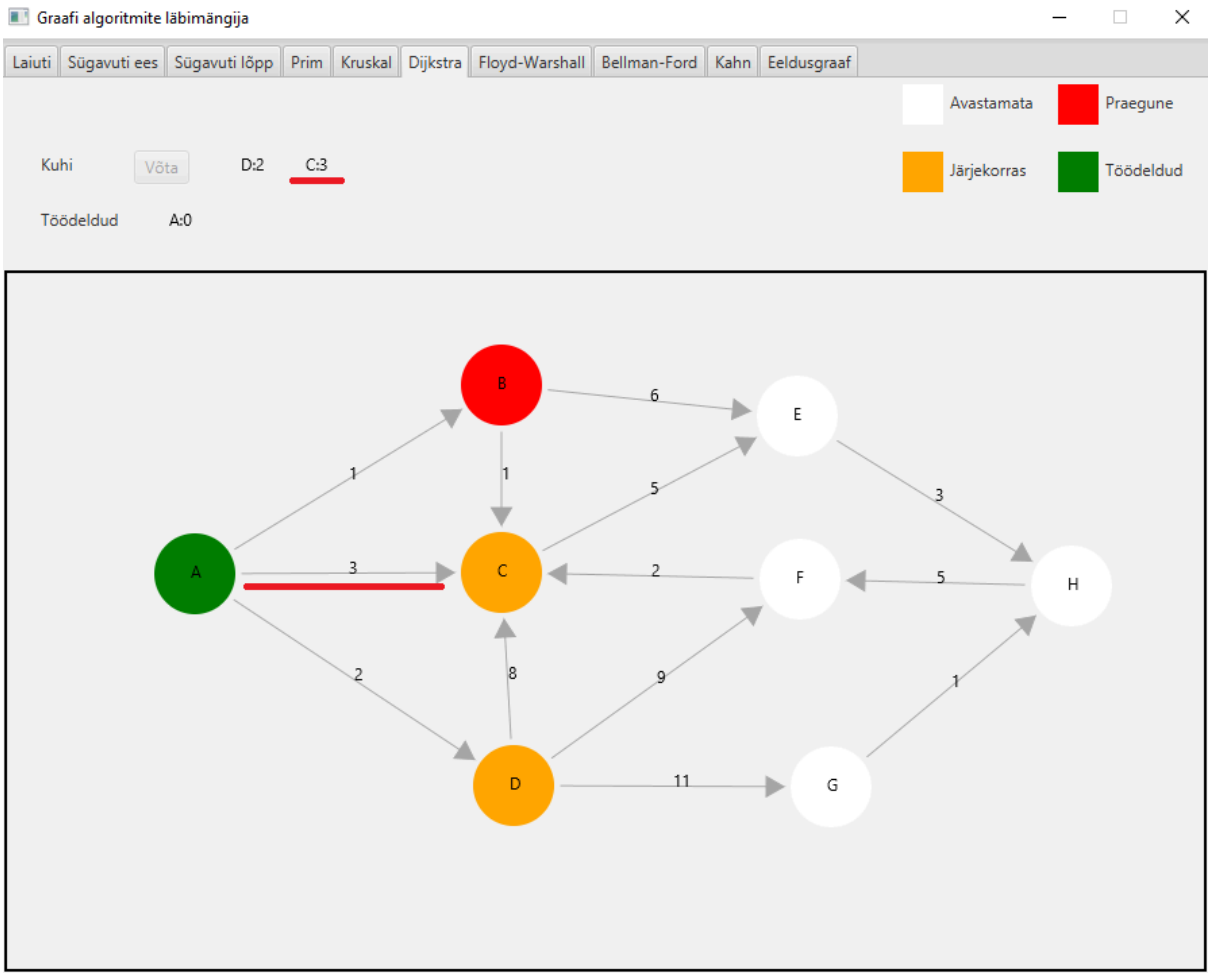
5. Järjekorda lisamise nupp Kahni algoritmi vaates



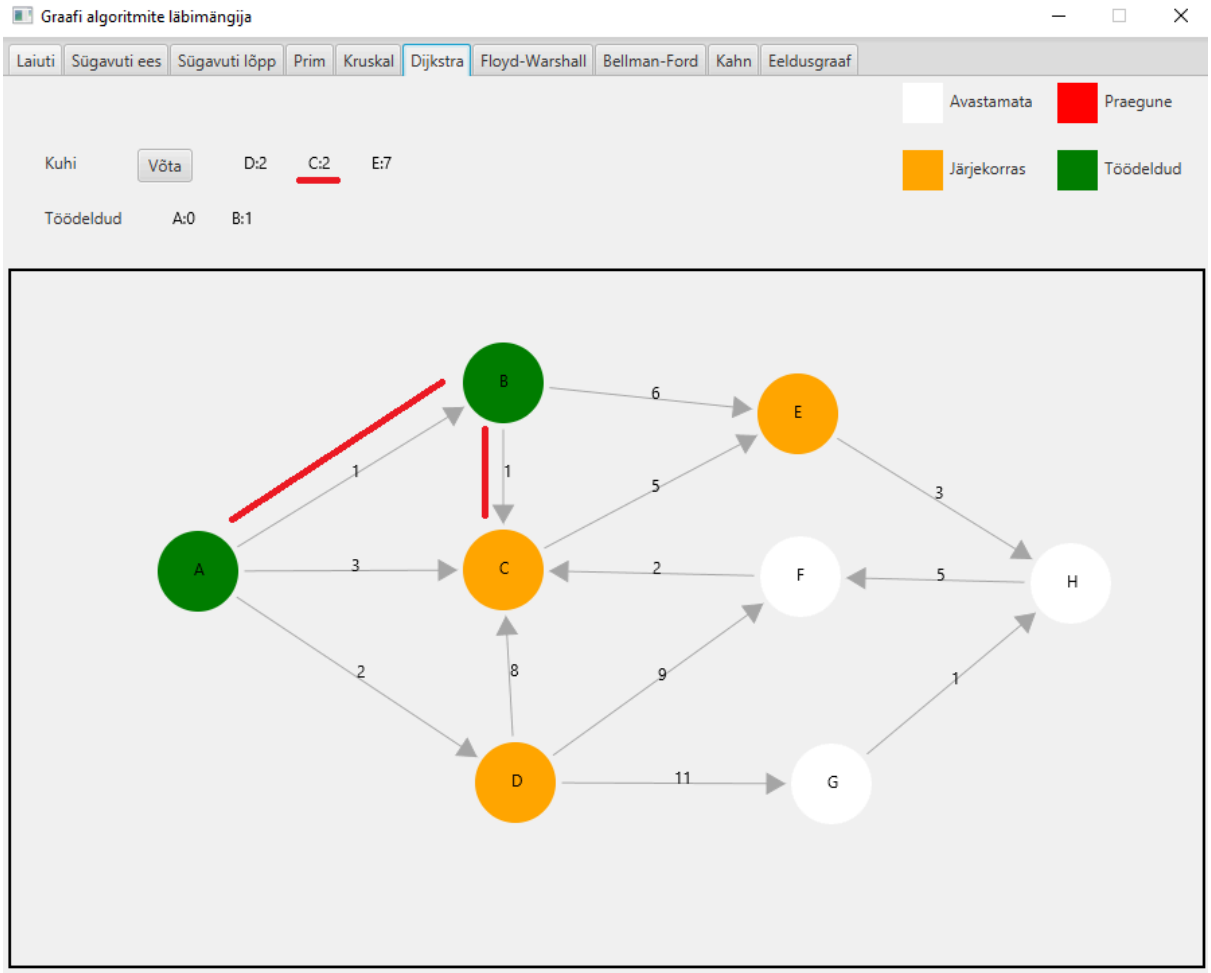
6. Andmestruktuurist elemendi võtmise nupp Kahni algoritmi vaates



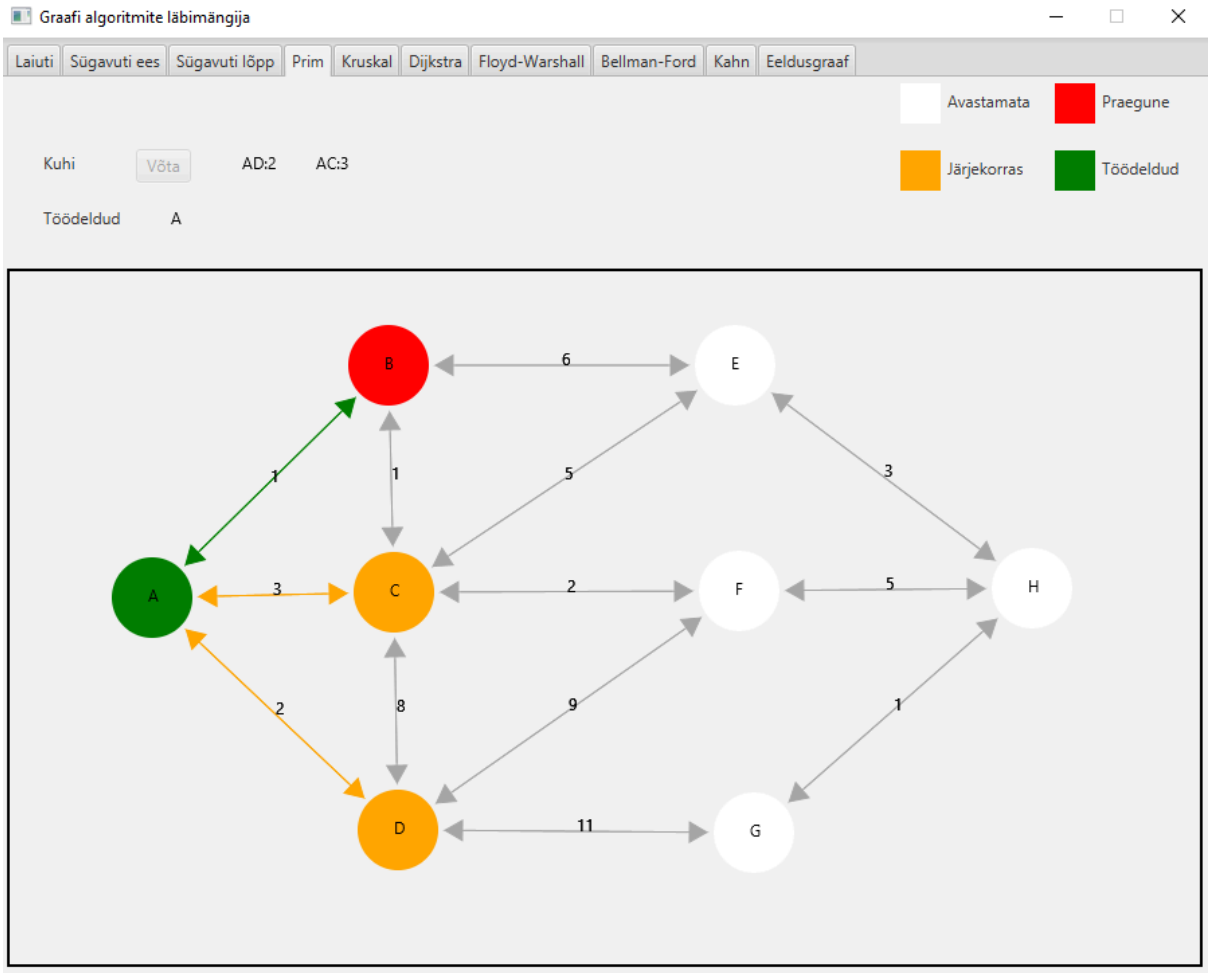
7. Kuhja seis enne võtmeparandust Dijkstra algoritmis



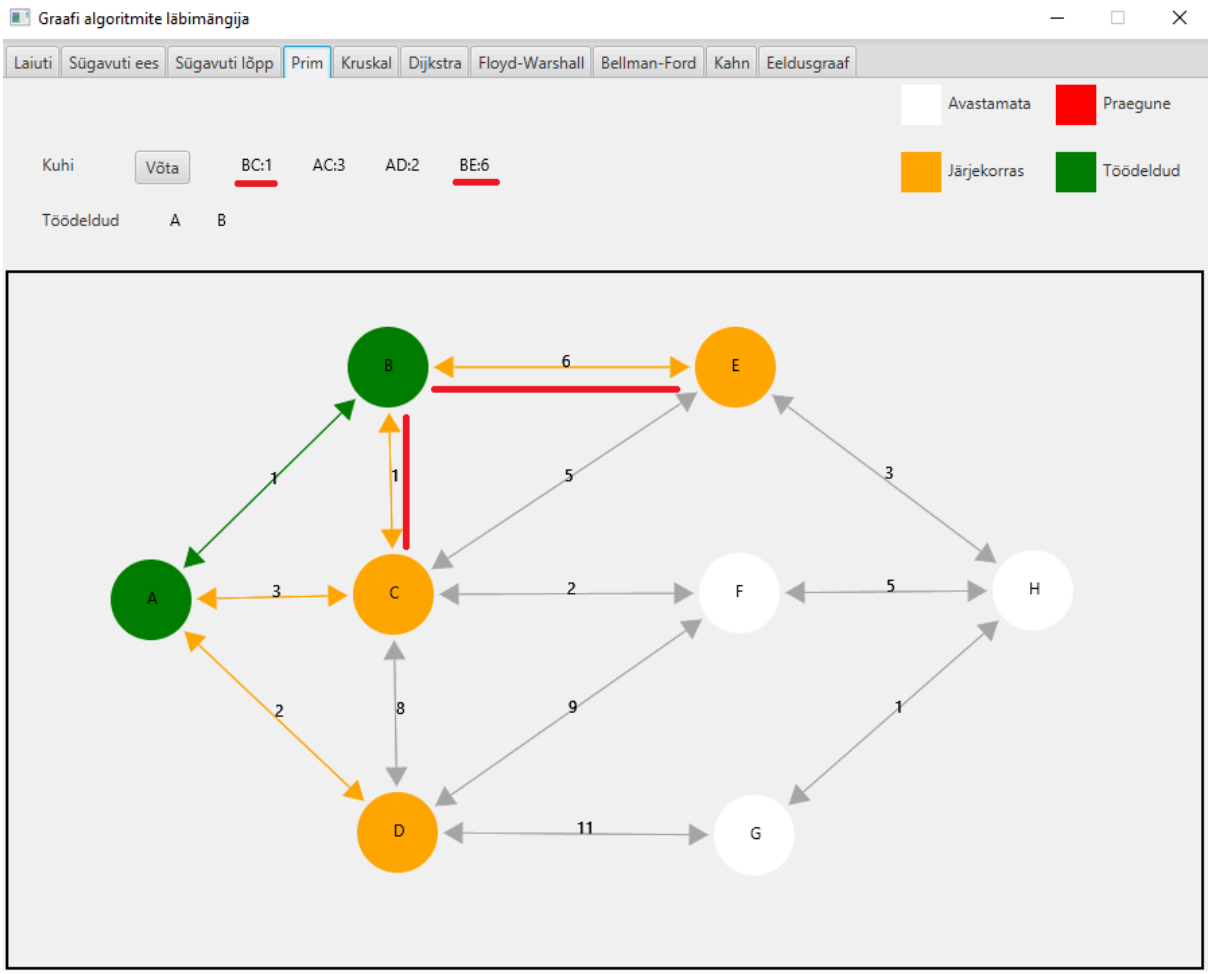
8. Kuhja seis peale võtmeparandust Dijkstra algoritmis



9. Kuhja seis enne uute kirjete lisamist Primi algoritmis



10. Kuhja seis pärast uute kirjete lisamist Primi algoritmis



11. Laiuti läbimisel tekkinud logi 3 veateatega

```
p edge 8 13
e A B
e A C
e A D
e B C
e B E
e C E
e D C
e D F
e D G
e E H
e F C
e G H
e H F

1       : Lisan tipu B järjekorda. KORRAS
2       : Lisan tipu C järjekorda. KORRAS
3       : Kontrollin tippu A. VIGA
3       : Kõik järglased ei ole töödeldud.
4       : Lisan tipu D järjekorda. KORRAS
5       : Kontrollin tippu A. KORRAS
6       : Võtsin järjekorrast järgmise tipu B. KORRAS
7       : Lisan tipu E järjekorda. KORRAS
8       : Kontrollin tippu B. KORRAS
9       : Võtsin järjekorrast järgmise tipu C. KORRAS
10      : Lisan tipu E järjekorda. VIGA
10      : Lõpptipp E on juba töödeldud või andmestruktuuris.
11      : Kontrollin tippu C. KORRAS
12      : Võtsin järjekorrast järgmise tipu D. KORRAS
13      : Lisan tipu F järjekorda. KORRAS
14      : Lisan tipu G järjekorda. KORRAS
15      : Kontrollin tippu D. KORRAS
16      : Kontrollin tippu F. VIGA
16      : Tipp F ei ole praegu töödeldav
17      : Võtsin järjekorrast järgmise tipu E. KORRAS
18      : Lisan tipu H järjekorda. KORRAS
19      : Kontrollin tippu E. KORRAS
20      : Võtsin järjekorrast järgmise tipu F. KORRAS
21      : Kontrollin tippu F. KORRAS
22      : Võtsin järjekorrast järgmise tipu G. KORRAS
23      : Kontrollin tippu G. KORRAS
24      : Võtsin järjekorrast järgmise tipu H. KORRAS
25      : Kontrollin tippu H. KORRAS
```

12. Lähtekood

Loodud käivitatav programm, programmi käivitusjuhend, programmi poolt kasutatavad graafid (teksti failidena) ning programmi lähtekood on leitav järgmiselt lingilt https://github.com/ErikPresnov/GraphGrader.

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Erik Presnov,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Graafialgoritmide läbimängija ja hindaja**, mille juhendaja on Ahti Põder, 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 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

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.

Erik Presnov
15.05.2024