TARTU ÜLIKOOL
Loodus- ja täppisteaduste valdkond
Arvutiteaduse instituut
Informaatika õppekava

Uku Hannes Arismaa

Graafialgoritmide sisendite genereerimine

Bakalaureusetöö (9 EAP)

Juhendaja: Ahti Põder, PhD

Tartu 2022

Graafialgoritmide sisendite genereerimine

Lühikokkuvõte:

Töö eesmärk on aine „Algoritmid ja andmestruktuurid“ õpetamist abistava programmi loomine. Programm genereerib graafialgoritmidele sisendeid nii, et genereeritud sisendid oleks algoritmi tööd sisendil läbimängivale tundengile võimalikult sarnase raskusega. Nii saab aine „Algoritmid ja andmestruktuurid“ õppejõud kiiresti genereerida tööde ja harjutuste jaoks tudengitele erinevaid sisendeid. Programm koosneb meetoditest, mis genereerivad sisendeid Kahni, Dijkstra, Bellmann-Fordi ning Floyd-Warshalli algoritmidele. Programm aitab tudengitel tulevikus paremini graafialgoritme omandada.

Võtmesõnad: Algoritmid ja andmestruktuurid, graafid, õpitarkvara

CERCS: P175 Informaatika, süsteemiteooria

Generating inputs for graph algorithms

Abstract:

The purpose of the Bachelor's thesis is to create a program to aid in the teaching of the course „Algorithms and data structures“. The program generates inputs for graph algorithms so that multiple inputs could be generated such that they pose a similar challenge to the student who is tasked with demonstrating the work of the algorithm with the given input. Using this program the professors can quickly generate inputs for the students. The program has methods for generating inputs for the Kahn algorithm, the Dijkstra algorithm, the Bellmann-Ford algorithm and the Floyd-Warshall algorithm. This program is one step of many on the path of imporving the learning process of graph algorithms for students.

Keywords: Algorithms and data structures, graphs, educational software

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Aine „Algoritmid ja andmestruktuurid“ jaoks luuakse hetkel paralleelselt mitmeid digilahendusi, mis lähevad kasutusse keskses platvormis DEEPMOOC, mille arendamine alles käib [1]. Üks nendest digilahendustest on tudengitele harjutamis- ja kontrollimiskeskkonna loomine.

Töö eesmärk on luua graafialgoritmide ülesannete genereerimiseks ning nende lahenduste hindamiseks metoodika ning programm aine „Algoritmid ja andmestruktuurid“ jaoks.

Aine „Algoritmid ja andmestruktuurid“ üheks õpiväljundiks on, et aine läbinud tudeng oleks tuttav põhiliste andmestruktuuride ning algoritmidega [2]. Kontrollimaks, kas see õpiväljund on saavutatud, on ainesKontrolltööd. Aine kontrolltöödes, mis on näha aine moodle'i lehel [3], esineb tihti ülesanne, kus on etteantud andmestruktuur andmetega ning tudeng peab läbi mängima algoritmi töö sellel andmestruktuuril. Hetkel peab kontrolltöö koostaja iga töö jaoks tegema uue sisendi ning kontrollima, kuidas algoritm selle peal käitub ning seejärel sisendit vajadusel kohendama. Sisendeid genereeriv algoritm võimaldaks selle osa tööst automatiseerida. Kui vastav algoritm leidub, võimaldab see määrata ülesande täpse raskuse ning genereerida ülesande koheselt ilma õppejõu aega kulutamata. Kuna genereeritud ülesanded on sama raskusega, oleks ka võimalik igale tudengile genereerida kontrolltöös oma ülesanne, vältimaks spikerdamist. Sarnase programmi massiivialgoritmide jaoks on kirjutanud Pitko oma lõputöös [4].

Lahenduste hindamise programm võimaldab läbi mängimise osa kontrolltöödest ning hindamisest teha elektrooniliseks. Määrates, kui palju tööd (näiteks arvutusi, võrdlusi) teatud sisendil algoritmi läbi mängimine võtab, saab läbimängus tehtud näpuvea korral võrrelda, kas tehti rohkem või vähem tööd võrreldes algse sisendi töömahuga. Programm annaks tudengitele võimaluse harjutada algoritme enne kontrolltööd ning anda neile jooksvalt tagasisidet.

Töö koosneb kolmest peatükist. Esimeses kirjeldatakse sisendite genereerimise protsessi. Teises selgitatakse graafidega seotud mõisteid, graafialgoritmide tööd ning sisendite genereerimist. Viimases peatükis on juhend valminud programmi kasutamiseks ning programmi töö ülevaade.

1 Lahenduse ülevaade

Peatükis kirjeldatakse, kuidas läheneti töö koostamisele ning programmi arendamisele. Selgitatakse raskusparameetri mõistet.

1.1 Sisendi genereerimise printsiibid

Iga genereeritav sisend peab vastama nõuetele, et tema genereerimisest soovitud kasu oleks. Esimene nõue on, et sisendil oleks võimalik algoritmi jooksutada (näiteks Dijkstra algoritmi puhul tuleb hoida kaarte pikkused positiivsed). Teine nõue on, et sisendit saaks genereerida raskusparameetrist (või mitmest): muutujast, mille saab programmile ette anda, mille põhjal genereeritakse sisend. Raskusparameeter peaks määrama, kui keeruline on või kui palju tööd peab tudeng mingi algoritmi läbimängimisel tegema. Samadel raskusparameetrite väärtustel genereeritud sisendite tegelik keerukus peab varieeruma võimalikult vähe. Kolmandaks, programm peab väljastama võimalikult palju erinevaid sisendeid. Minimaalselt oleks see piisavalt sisendeid, et iga tudeng saaks kontrolltöös läbi mängida erinevat sisendit. Ideaalis väljastaks programm iga etteantud raskusparameetri väärtuse (või mitme) korral kõik sobiva raskusega sisendid võrdse või vähemalt mingi tõenäosusega. Neljandaks, eriti graafide puhul, on iga sisendi puhul vaikimisi raskusparameetriks väljundi loetavus (ja esteetilisus). Kuna neljas nõue nõuab teatud sisendite välistamist, siis on see vastuolus kolmandaga. Siin töös on rõhk pigem kolmandal nõudel ja ei kaaluta visuaalset vaatepunkti. Graafide visualiseerimise kohta on kirjutanud Tiganik oma lõputöös [5].

1.2 Algoritmide valikust

Uuriti aine „Algoritmid ja andmestruktuurid“ loengutes käsitletud algoritme, mis esinesid aine ülesannetes. Töös käsitlemiseks valiti need algoritmid, mille puhul sisendiks ei kõlba lihtsalt igasugune kombinatsioon tippudest ja kaartest (ja pikkustest). Algoritm vastab nõudele siis, kui sellele on olemas hea raskusparameeter või kui sisendiks oleva graafi puhul on oluline mõni graafi omadustest. Selle kriteeriumi tõttu jäävad käsitlemata sügavuti läbimine, laiuti läbimine ning minimaalse toespuu leidmise algoritmid.

1.3 Sisendi genereerimise protsess

Iga algoritmi puhul uuriti, millised võiks olla selle raskusparameetrid ning millised nõuded peavad kehtima sisendiks olevale graafile. Kui olid leitud sobivad raskusparameetrid, sai hakata programmeerima sobivat sisendit genereerivat meetodit. Meetodi leidmisel kontrolliti meetodi töö korrektsust suvalistel sisenditel, reeglina tippude arvuga vahemikust $(1, 20)$, graafi nõuetele vastavates kaarte arvude vahemikes ning raskusparameetrite vahemikes. Lõpuks loodud meetodid dokumenteeriti.

2 Algoritmidele sisendite genereerimine

Iga algoritmi kohta on kolm osa. Kõigepealt lühike kirjeldus, mida algoritm teeb. Seejärel detailsem sõnastus, kuidas algortim seda teeb ehk kuidas algoritmi realiseerida. Viimaks osa algoritmile sisendite genereerimisest. Kõigepealt tutvustan teemaga seotud termineid.

2.1 Graafidest üldiselt

Järgnevad on Kiho [6] kasutatud mõistetest olulisimad. *Graaf* on defineeritud kui mittetühi hulk tippe ja hulk *kaari*: tippude järjestatud paare [7] (tähistatakse $G = (V, E)$). Kaar on defineeritud kahe järjestatud tipu järgi. Nendest esimene on *algustipp* ning teine *lõpptipp*. Kui tipp on kaare algus- või lõpptipp, on see kaar antud tipuga *intsidentne* [7]. Lõpptipp on algustipu *naaber* st tipp on (kaaripidi, kus see tipp on algustipp) ühendatud oma naabritega. Kui hakata mingist tipust (*eelkäija*) kaari mööda minema ja lõpuks mingi tipu (*järeltulija*) juures peatuda, moodustavad läbitud tipud *tee*. Kui lõpetatakse samas tipus, kust alustati, on tegu *tsükliga*. Kaarega võib olla seotud *pikkus*. Selle järgi saab arvutada *tee pikkuse*: kõigi läbitud kaarte pikkuste summa. Kahe tipu vaheline *kaugus* on neid ühendava kõige lühema tee pikkus.

Graaf on *tugevalt sidus*, kui graafi igast tipust leidub tee igasse teise tippu [7]. Graaf on *nõrgalt sidus*, kui igast tipust igasse teise tippu tee moodustamiseks on vahel vaja kaare lõpptipust minna kaare algustippu [7]. Graafi nimetatakse tavaliselt *puuks*, kui graaf on sidus ja tsükliteta [7]. Palm [7] defineerib puu *suunamata graafide* puhul, kus kaare lõpptipust saab minna algustippu. Kuna graafide genereerimiseks kasutatakse siin töös suunatud graafe, kasutatakse siin natuke erinevat puu definitsiooni. Puu on tsükliteta graaf, mille mingist tipust (*juurtipp*, algoritmide puhul tihti algoritmi lähtetipp) saab moodustada tee igasse teise tippu. Puu tipu sügavus on kaarte arv tees lähtetipust antud tipuni [3]. Puu kõrgus on suurim puu tipu sügavus [3]. *Kauguste puu* on graaf, mis koosneb mingi alusgraafi tippude hulgast ning selle alusgraafi selliste kaarte alamhulgast, mis on vajalikud lähtetipust igasse teise tippu viivate lühimate teede moodustamiseks.

2.2 Kahni algoritm

Kahni algoritmi eesmärk on leida graafi topoloogiline järjestus. *Topoloogiline järjestus* on graafi tippude järjestus, kus iga kaare algustipp on järjestuses enne selle kaare lõpptippu [6]. Topoloogilise järjestuse olemasolu mingi graafi puhul annab garantii, et selles järjekorras tippe läbides ei läbi me kunagi tippu, millesse viiks kaar läbimata tipust [6]. Topoloogiline järjestus leidub parajasti siis, kui graafis puuduvad tsüklid [6].

2.2.1 Kahni algoritmi realiseerimine

Näidis realisatsioon on toodud joonisel 1. Realiseerimise idee leiab aine „Algoritmid ja andmestruktuurid“ materjalidest [3]. Iga graafi tipu v kohta peetakse meeles (tippu esitava kirje väljas *sisendaste*) selle *sisendastet*: kaarte arvu, mille lõpptipuks see tipp on [7] (read 2 - 5). Tuleb teha teine andmestruktuur A, kus hoida läbimata tippe, mille sisendastmeks on 0. Seejärel tuleb ükshaaval hakata tippe, mille sisendaste on 0, läbima (rida 8). Läbimise käigus lisatakse tipp tagastatava järjestuse lõppu (rida 10) ning iga kaare, mille algustipuks see tipp on, lõpptipu sisendastet vähendatakse ühe võrra (read 11 - 14). Lõpuks eemaldatakse see tipp andmestruktuurist ja minnakse andmestruktuuris järgmise tipu juurde või lõpetatakse töö.

Antud : Tsükliteta graaf $G = (V, E)$
Tagastab : Topoloogiline järjestus T

```
1  T := ∅;
2  foreach v ∈ V do
3      v.sisendaste := 0;
4  foreach (v_i, v_j) ∈ E do
5      v_j.sisendaste++;
6  A := ∅;
7  A ⇐ {v | v.sisendaste = 0};
8  while A pole tühi do
9      t ⇐ A ;
10     T ⇐ t;
11     foreach (t, v_j) ∈ E do
12         v_j.sisendaste −−;
13         if v_j.sisendaste=0 then
14             A ⇐ v_j;
```

Joonis 1. Kahni algoritm

2.2.2 Kahni algoritmi sisendi genereerimine

Kuna algoritmis tehakse kõike iga kaare ja iga tipu korral võibki sisendi keerukuse parameetriteks määrata kaarte ja tippude arvud. Sisendi genereerimisel peab silmas pidama, et väljund oleks tsükliteta ning nõrgalt sidus. Nõrk sidusus on oluline, kuna tudengi läbimänguoskust kontrollides pole mittesidusatel graafidel algoritmi jooksutamise oskus eriti oluline. Alati saab genereerida kaks või enam väiksema tippude arvuga graafi ja esitada need ühe sisendina. Vastupidine on märgatavalt tülikam.

Tuleb välja, et parajasti siis, kui graaf on tsükliteta, saab järjestada kõik selle tipud nii, et graafi kaared viivad ainult järjekorras eespool olevatest tippudest tagapool olevatesse. Nii on realiseeritud sisendi generaator joonisel 2. Selleks loome järjestatud tippude hulga V, esialgu tühja hulga E kaarte jaoks ning sobivate potentsiaalsete kaarte hulga E_p (read 1-3). Valime suvaliselt potentsiaalstest kaartest kaari nii, et graaf oleks kindlasti nõrgalt sidus (read 4-13). Selleks valitakse tipp indeksiga *pivot*, millega hakatakse järjekorras kõrvuti olevaid kaari ühendama. Kaare suund sõltub sellest, kummal pool tippu indeksiga *pivot* ühendatav tipp järjekorras on. Lõpuks valime suvaliselt ülejäänud kaared (rida 15).

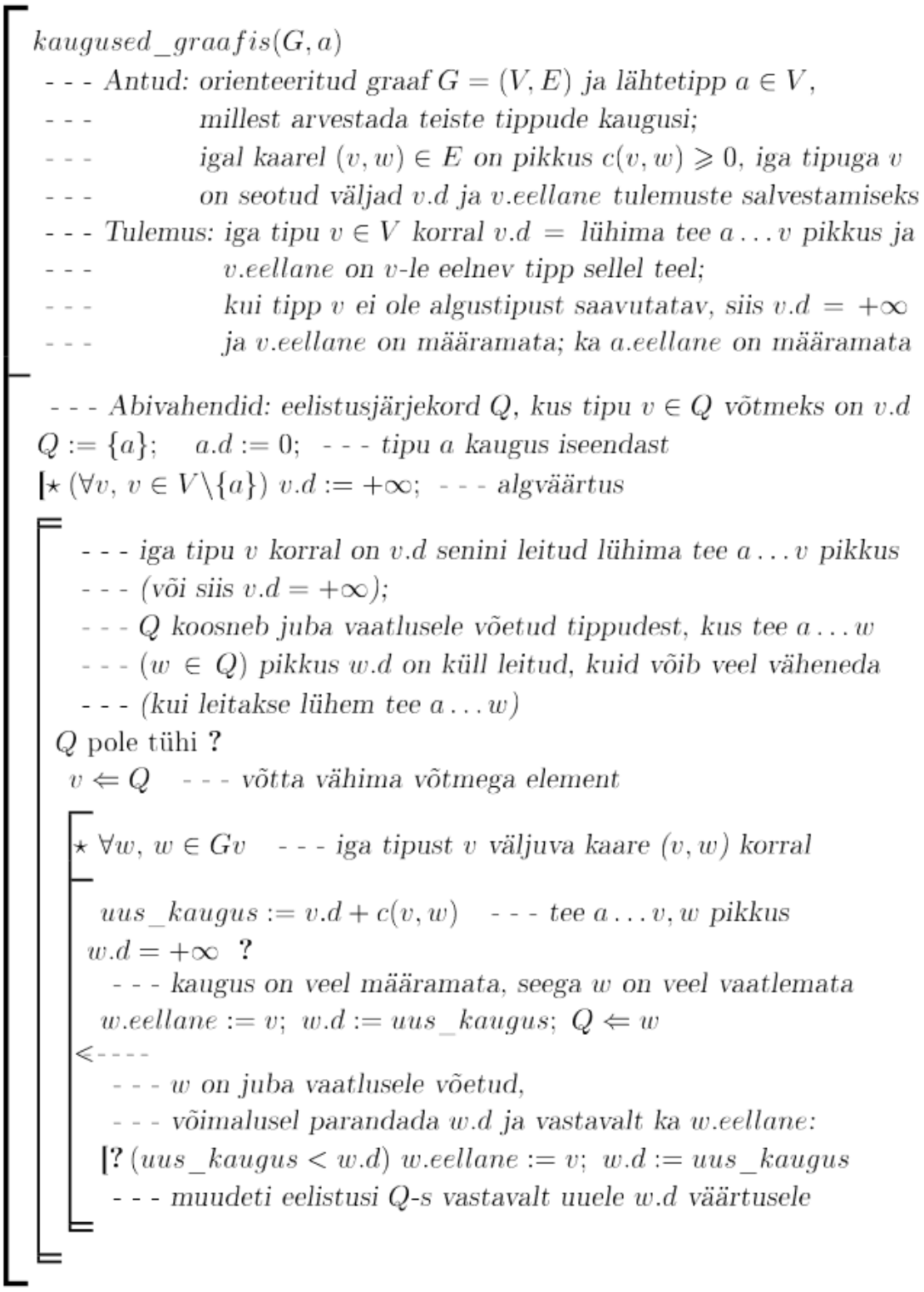
Antud : Tippude arv n ning kaarte arv m, kus $\frac{n(n-1)}{2} \geq m \geq n-1$
Tagastab : Topoloogiliselt järjestatav sidus graaf G tippude arvuga n ning kaarte arvuga m.

```
1  V := {v_0, ..., v_{n-1} | ∀i, v_i.läbitud :=false } ;
2  E := ∅ ;
3  E_p := {(v_i, v_j) | i < j} ;
4  pivot := suvalise v_i indeks;
5  v_pivot.läbitud :=true;
6  for j := 0 to n − 1 do
7      v_i ⇐ suvaline tipp hulgast V, mis on kõrvuti läbitud tipuga;
8      if i >pivot then
9          E ⇐ suvaline kaar, mille lõpptipp on v_i ning algustipp on läbitud;
10     else
11         E ⇐ suvaline kaar, mille algustipp on v_i ning lõpptipp on läbitud;
12     v_i.läbitud :=true;
13     V ⇐ v_i;
14 for i := n − 1 to m do
15     E ⇐ suvaline kaar hulgast E_p;
```

Joonis 2. Kahni algortmi sisendi genereerija

2.3 Dijkstra algoritm

Algoritm leiab kõikide tippude kaugused lähtetipust (graafis, kus kõikide kaarte pikkus on positiivne) [3].



Joonis 3. Dijkstra algoritmi realiseering Kiholt [6]

2.3.1 Dijkstra algoritmi realiseerimine

Kiho [6] realiseerib algoritmi joonisel 3 järgmiselt. Iga tipu kohta peetakse meeles selle *teadaolev kaugus*: hetkel teadaoleva lühima tee pikkust lähtetipust sellesse tippu. Lähtetipu teadaolevaks kauguseks seatakse 0 ning iga teise tipu teadaolevaks kauguseks $+\infty$. Võetakse kasutusele eelistusjärjekord, kuhu lisatakse lähtetipp. Eelistusjärjekorras on elemendi võtmeks selle teadaolev kaugus. Kuni eelistusjärjekorras on tippe, eemaldatakse sealt vähima teadaoleva kaugusega element v. Iga tipu w puhul, kuhu viib tipust v kaar, kontrollitakse, kas selle teadaolev kaugus on veel $+\infty$. Kui on, siis muudetakse teadaolev kaugus tipu v kauguse ning kaare (v, w) pikkuse summaks ning lisatakse tipp w eelistusjärjekorda. Vastasel juhul kontrollitakse, kas tippu v ning kaart (v, w) kaudu saadakse lühem kaugus, kui praegune tipu w teadaolev kaugus. Kui saadakse, siis teadaolev kaugus muudetakse ning parandatakse tipu w paiknemist eelistusjärjekorras. Kui eelistusjärjekord on tühi, on iga tipu teadaolev kaugus selle kaugus lähtetipust üldises mõttes. Selles realiseeringus on kasutatud välja *eellane*, mida siin ei käsitleta.

2.3.2 Dijkstra algoritmi sisendi genereerimine

Algoritmi töö käigus käib eelistusjärjekorrast läbi iga tipp (mis on algtipust saavutatav) ning peale sealt eemaldamist käiakse läbi igast tipust väljuvad kaared. Kui kaare ja tipu kauguste summa on väiksem märgitud kaugusest, muudetakse kaugust ja parandatakse tipu positisooni eelistusjärjekorras. Sellest tulenevad sisendi genereerimise kolm raskusparameetrit: tippude arv, kaarte arv ning eelistusjärjekorras tipu paiknemise parandamiste arv.

Dijkstra algoritmi sisendi genereerimine on realiseeritud joonistel 4, kus genereeritakse kaared, ja 5, kus genereeritakse kaarte pikkused.

Esmalt genereeritakse tipud ja kaared, mis põhjustavad tipu eelistusjärjekorda lisamist või seal positsiooni parandamist. Parandamisteks saab kasutada ainult kaari, mille algustipp on järjekorras lõpptipust tagapool, muidu proovitaks parandada eelistusjärjekorrast eemaldatud tipu kaugust. Genereerimist saab alustada luues soovitud pikkusega tippude järjestuse V, mis algab lähtetipust v_0, esialgu tühja parandusi põhjustavate kaarte hulga E, ning genereerides kõik kaared, mis saavad parandamist põhjustada (hulka E_p) (read 1-3). Et lähtetipust leiduksid teed teistesse tippudesse, peab garanteerima, et iga tipp, peale lähtetipu, oleks vähemalt ühe kaare lõpptipp (read 4-5). Seejärel tuleb suvaliselt valida ülejäänud kaared (rida 6).

Seejärel genereeritakse kaared (rida 7), mis ei põhjusta parandusi. Need on jagatud kahte hulka vastavalt sellele, kas algustipp on lõpptipust järjekorras eespool (E_v) või tagapool (E_s) (read 8-15). Nii genereerides võib juhtuda, et mingi lõpptipu jaoks järjekorras eespoolseim algustipp ei tohiks põhjustada selle lõpptipu eelistusjärjekorda lisamist, kuna pole vastavas hulgas. Selle vältimiseks tuleb iga lõpptipu v_i puhul kontrollida, millises hulgas eespoolseima algtipuga kaar on, ning vajadusel vahetada see kaar õiges

hulgas oleva kaare vastu (read 16-22).

```
Antud     : Tippude arv n, kaarte arv m, kus n − 1 ≤ m ≤ n(n − 1) ning
            parandamiste arv p, kus p ≤ m − n + 1
Tagastab : n tipuga hulk V, p + n − 1 kaarega hulk E ning kokku
            m − p − n + 1 kaarega hulgad E_v, kus algustippude indeksid on
            väiksemad lõpptippudest, ning E_s, kus algustippude indeksid on
            suuremad lõpptippudest
1  V := {v_0, ..., v_{n−1}} ;
2  E := ∅ ;
3  E_p := {(v_i, v_j) | i < j} ;
4  for i := n − 1 to 0 do
5      E ⇐ suvaline kaar hulgast E_p, mille lõpptipp on v_i;
6  E ⇐ p + n − 1 suvalist kaart hulgast E_p
7  E_p := E_p ∪ {(v_i, v_j) | j < i};
8  E_v := ∅;
9  E_s := ∅ ;
10 for i := 0 to m − p − n + 1 do
11     (v_i, v_j) ⇐ suvaline kaar hulgast E_p;
12     if i<j then
13         E_v ⇐ (v_i, v_j);
14     else
15         E_s ⇐ (v_i, v_j);
16 foreach v_i ∈ V do
17     (v_j, v_i) ⇐ vähima j väärtusega kaar hulgast E;
18     (v_k, v_i) ⇐ vähima k väärtusega kaar hulgast E_v;
19     if j > k then
20         (v_j, v_i) ⇔ (v_k, v_i);
21     E ⇐ (v_j, v_i);
22     E_v ⇐ (v_k, v_i);
```

Joonis 4. Dijkstra algoritmi sisendi kaarte genereerija

Lõpuks määratakse kaarte pikkused (väljas (v_1, v_2).*pikkus*). Alapeatüki lõpuni genereeritakse suvalisi arve alati vahemikust $(0, 21)$. Pikkuste määramisel alustatakse Dijkstra algoritmi väljundi - tippude kauguste (v.*kaugus*) - genereerimisest ning seejärel hakatakse servi läbima ning väärtustama algoritmist vastupidises järjekorras.

Joonisel 5 toodud algoritmis genereeritakse esmalt kauguste puu lähtetipust v_0. Läh-

tetipu kauguseks seatakse 0 (rida 1). Iga järjekorras järgmise tipu v_i puhul seatakse selle kaugus lähtetipust suvalise arvu võrra suuremaks järjekorras eelmisest tipust v_{i-1} (rida 4). Selleks, et see kaugus oleks saavutatav kauguste puud kaudu, määratakse puus oleva kaare (v_i, v_j) pikkus kaare lõpp- ja algtipu kauguste vaheks (rida 5).

Seejärel genereeritakse pikkus kaartele (v_i, v_j), mille lõpptipp on eelistusjärjekorrast eemaldatud enne algustippu ($j < i$) (read 6-7). Nende pikkuste puhul on oluline, et pikkus ei võimaldaks lõpptipu teadaolevat kaugust parandada. Kuna algustipu kaugus on alati suurem lõpptipu kaugusest, määratakse kaare pikkuseks lihtsalt suvaline arv.

Ülejäänud kaarte pikkuste genereerimist alustatakse järjekorras tagumise algustipuga kaartest (v_i, v_j). Kaarte puhul, mis põhjustavad parandamise (hulgast E), muudetakse lõpptipu kaugus paranduse eelseks väärtuseks (liites parandatud väärtusele suvalise arvu (rida 10)) ning muudetakse kaare pikkus vastavaks (rida 11). Kaarte puhul, mis parandamist ei põhjusta (hulgast E_v), määratakse pikkuseks summa lõpptipu ja algustipu vahest ning suvalisest arvust, mis peab olema suurem mujal suvaliselt genereeritavast arvust (siin vahemikust $(20, 41)$), et seda kaart mööda kindlasti saavutataks suurem teadaolev kaugus, kui mõnda eelmist teed mööda (rida 13).

Antud : $V, |V| = n$, $E, |E| = p + n - 1$ ning
$E_v, E_s, \forall (v_i, v_j) \in E_s, i > j, \forall (v_i, v_j) \in E_v, i < j, |E_v| + |E_s| = p + n - 1$
Tagastab : Raskusparameetritele n, m, p vastav graaf, millel saab läbi mängida Dijkstra algoritmi

1 v_0.*kaugus* $:= 0$;
2 **for** $i := 1$ **to** n **do**
3 $\quad (v_i, v_j) :=$ *suurima lõpptipu indeksiga kaar hulgast* E;
4 $\quad v_i$.*kaugus* $:= v_{i-1}$.*kaugus* $+$ *suvaline arv vahemikust* $(0, 21)$;
5 $\quad (v_i, v_j)$.*pikkus* $:= v_j$.*kaugus* $- v_i$.*kaugus*;
6 **foreach** $(v_i, v_j) \in E_s$ **do**
7 $\quad (v_i, v_j)$.*pikkus* $:=$ *suvaline arv vahemikust* $(0, 21)$;
8 **for** $i := n - 1$ **to** 0 **do**
9 $\quad$ **foreach** $(v_i, v_j) \in E$ **do**
10 $\quad\quad v_j$.*kaugus* $:= v_j$.*kaugus* $+$ *suvaline arv vahemikust* $(0, 21)$;
11 $\quad\quad (v_i, v_j)$.*pikkus* $:= v_j$.*kaugus* $- v_i$.*kaugus*;
12 $\quad$ **foreach** $(v_i, v_j) \in E_v$ **do**
13 $\quad\quad (v_i, v_j)$.*pikkus* $:=$
$\quad\quad\quad v_j$.*kaugus* $- v_i$.*kaugus* $+$ *suvaline arv vahemikust* $(20, 41)$;
14 $E := E \cup E_v \cup E_s$;

Joonis 5. Dijkstra algoritmi sisendi kaarte väärtustaja

2.4 Bellmann-Fordi algoritm

Loengu materjalidest [3] saame teada, et Dijkstra algoritm ei tööta, kui kaarte pikkused võivad olla negatiivsed. Selleks juhuks on Bellmann-Fordi algoritm. Algoritmi saab kasutada negatiivsete tsüklite tuvastamiseks.

```
parandada_kaugused(G, a)
 - - - Antud: orienteeritud graaf G = (V, E) ja lähtetipp a ∈ V,
 - - -        millest arvestada teiste tippude kaugusi;
 - - -        igal kaarel (v, w) ∈ E on pikkus c(v, w), iga tipuga v
 - - -        on seotud väljad v.d ja v.eellane tulemuste salvestamiseks
 - - - Tulemus: iga tipu v ∈ V korral, kui eksisteerib lühim tee a ... v,
 - - -          siis v.d = lühima tee a ... v pikkus ja
 - - -          v.eellane on v-le eelnev tipp sellel teel;
 - - -          kui tipp v ei ole algustipust saavutatav, siis v.d = +∞
 - - -          ja v.eellane on määramata; ka a.eellane on määramata

a.d := 0;  - - - tipu a kaugus iseendast

⋆ ∀v, v ∈ V\{a}

  v.d := +∞;  - - - algväärtus

⋆ |V| − 1

  ⋆ ∀(v, w), (v, w) ∈ E  - - - iga kaare (v, w) korral

    uus_kaugus := v.d + c(v, w)  - - - tee a ... v, w pikkus
     - - - võimalusel parandada w.d ja vastavalt ka w.eellane:
    [? (uus_kaugus < w.d) w.eellane := v; w.d := uus_kaugus
```

Joonis 6. Bellmann-Fordi algoritmi realiseering Kiholt [6].

2.4.1 Bellmann-Fordi algoritmi realiseerimine

Kiho [6] realiseerib algoritmi kahes osas. Üks osa (toodud joonisel 6) määrab graafis tippude kaugused lähtetipust. Teine osa kontrollib, kas esimeses osas leitud kaugused on

korrektsed.

Iga tipu kohta peetakse meeles selle teadaolev kaugus. Esimeses algoritmi osas seatakse lähtetipu teadaolevaks kauguseks 0 ning iga teise tipu kauguseks $+\infty$. Seejärel käiakse kõik kaared läbi üks kord vähem, kui on tippude arv. Igal läbikäimisel peatutakse iga kaare juures ning kontrollitakse, kas kaare algustipu teadaoleva kauguse ja kaare pikkuse summa on väiksem kaare lõpptipu teadaolevast kaugusest. Kui on, siis parandatakse lõpptipu teadaolevat kaugust: seatakse see kaare algustipu teadaoleva kauguse ja kaare pikkuse summaks. Teises osas käiakse iga kaar veelkord läbi ja kontrollitakse sama asja. Kui ikka tehakse parandusi, on graafis tsükkel, millele vastavate kaarte läbimisel on pikkuste summa negatiivne (*negatiivne tsükkel*), seega seda tsüklit mitu korda läbides, saab selles olevate (ja mõnede teiste) tippude kaugust kahandama jäädagi.

2.4.2 Bellmann-Fordi algoritmi sisendi genereerimine

Algoritmi töö käigus käiakse iga kaar läbi tippude arv korda. Mõnel läbimisel tehakse tipu teadaoleva kauguse parandus. Kuna kaarte läbimise järjekord pole fikseeritud, on võimalik, et algoritmi läbi mängiv tudeng leiab negatiivsete tsükliteta sisendi puhul lõplikud kaugused kaarte esimesel läbimisel ning teisel läbimisel näeb, et kaugused enam ei muutu. See tähendab, et paranduste arv (arvestamata teadaolevate kauguste muutusi väärtuselt $+\infty$) võib olenevalt kaarte järjekorrast olla negatiivsete tsüklite puudumisel alati 0. Seega on sobilikeks raskusparameetriteks tippude arv, kaarte arv ning kauguste puu kõrgus. Kauguste puu kõrgus on ühtlasi tudengi jaoks maksimaalne välimise tsükli läbimiste arv, kuna järgmisel kõikide kaarte läbimisel parandusi ei toimu. See kaarte arv määrab lisaks ära, kui raske on tudengil graafile peale vaadates leida kaarte läbimisjärjekorda, et algoritmi läbi mängimine kahe välimise tsükli läbimisega lõpetada.

Esmalt genereeritakse kõiks tipud V ning kõik potentsiaalsed kaared E_p (read 1-3). Seejärel konstrueeritakse kauguste puu. Kõigepealt kasutatakse esimest l kaart sirge tee moodustamiseks, mis annab puule kõrguse (read 5-7). Seejärel ühendatakse puusse ülejäänud tipud, valides kaari nii, et need ühendaksid uut ühendatavat tippu v_i olemasoleva puuga ilma, et puu kõrgus muutuks (read 8-11).

Seejärel genereeritakse puu kaarte pikkused. Iga pikkus on suvaline arv vahemikust $(-10, 10)$. Nende pikkuste põhjal arvutatakse tippude kaugused lähtetipust (read 12-17).

Lõpuks genereeritkse ülejäänud kaared ning nende pikkused, valides suvaliselt kasutamata kaartest. Kaare pikkus peab olema selline, et seda mööda ühegi tipu kaugust parandada ei saaks. Selleks leitakse kaare tippude kauguste vahe ning liidetakse sellele suvaline arv vahemikust $(0, 21)$ (read 18-21).

Antud : Tippude arv n, kaarte arv m, kus $n(n-1) \geq m \geq n-1$ ning kauguste puu sügavus $l \leq n-1$

Tagastab : Raskusparameetritele vastav graaf, millel saab jooksutada Bellmann-Fordi algoritmi ning kus puuduvad negatiivsed tsüklid

1 $V := \{v_0, ..., v_{n-1}\}$;
2 $E := \emptyset$;
3 $E_p := \{(v_i, v_j) \mid i \neq j\}$;
4 v_0.***sügavus*** $:= 0$;
5 **for** $i := 0$ **to** l **do**
6 $\quad E \Leftarrow (v_i, v_{i+1}) \Leftarrow E_p$;
7 $\quad v_{i+1}$.***sügavus*** $:= v_i$.***sügavus*** $+ 1$;
8 **for** $i := l + 1$ **to** n **do**
9 $\quad (v_j, v_i) \Leftarrow$ ***suvaline kaar hulgast*** E_p, ***kus*** $j < i$ ***ning*** v_j.sügavus $\neq l$;
10 $\quad v_i$.***sügavus*** $:= v_j$.***sügavus*** $+ 1$;
11 $\quad E \Leftarrow (v_j, v_i)$;
12 v_0.***kaugus*** $:= 0$;
13 **for** $i := 1$ **to** n **do**
14 $\quad (v_j, v_i) \Leftarrow E$;
15 $\quad (v_j, v_i)$.***pikkus*** $:=$ ***suvaline arv vahemikust*** $(-10, 10)$;
16 $\quad v_i$.***kaugus*** $:= v_j$.***kaugus*** $+ (v_j, v_i)$.***pikkus***;
17 $\quad E \Leftarrow (v_j, v_i)$;
18 **for** $i := 0$ **to** $m - n + 1$ **do**
19 $\quad (v_i, v_j) \Leftarrow$ ***suvaline kaar hulgast*** E_p;
20 $\quad (v_i, v_j)$.***pikkus*** $:= v_j$.***kaugus*** $- v_i$.***kaugus*** $+$ ***suvaline arv vahemikust*** $(0, 21)$;
21 $\quad E \Leftarrow (v_i, v_j)$;

Joonis 7. Bellmann-Fordi algoritmi negatiivsete tsükliteta sisendi genereerija

Kui sisendis esineb negatiivne tsükkel, peab tudeng igal kaarte läbimisel mingi kaare kaugust parandama. Siin on sobilikuks raskusparameetriks (lisaks tippude ja kaarte arvule) tippude arv, mille teadaolev kaugus saab igal tsükli läbimisel muutuda, kuna need tipud moodutavad see osa graafist, millele tudeng peab mõtlema peale esimest läbimist.

Joonisel 8 genereeritav sisend koosneb kahest suurest ja ühest väiksest osast. Graafi osa, mida negatiivne tsükkel ei mõjuta on esimesed $n - u$ tippu ning nende vahelised kaared (hulgast E_p). Graafi osa, mida mõjutab on viimased u tippu ning nende vahelised kaared (hulgast E_{p3}). Kolmas osa on esimest kahte osa ühendav osa: kaared hulgast E_{p2}. Kõigepealt genereeritakse iga osa minimaalse hulga kaartega. Esimese osa puhul tähendab see kauguste puu genereerimist (read 6-11), teise puhul negatiivse tsükli (read 16-21) ning ülejäänud tippe sellega siduvate kaarte genereerimist (read 22-23) ning kol-

manda puhul suvalise kaare genereerimist (read 12-14). Lõpuks genereeritakse ülejäänud kaared (read 25-31).

Esmalt genereeritakse vajalikud struktuurid: tippude hulk V, tulemuskaarte hulk E, negatiivse tsükli suurus t ning kaarte hulgad, kust saab kaari suvaliselt valima hakata (E_p ja E_{p2}) (read 1-4). Selleks, et programm töötaks, peab kontrollima, kas üldse on tippe, mida negatiivne tsükkel ei mõjuta (rida 5). Kui on, saab genereerida suvalise kauguste puu suvaliste kaarte pikkustega vahemikust $(-10, 10)$ (read 6-11) ning suvalise kahte osa ühendava kaare suvalise pikkusega vahemikust $(-10, 10)$ (read 12-14).

Seejärel genereeritakse negatiivse tsükliga osa kasutades hulga E_{p3} kaari. Alustuseks genereeritakse negatiivne tsükkel. Selleks hoitakse muutujas s meeles tsükli senist pikkust (rida 16), et osataks väärtustada viimase kaare pikkust. Seejärel genereeritakse t-tipune tee kaartest pikkustega vahemikust $(-10, 10)$ (read 17-19). Nüüd tee viimane tipp ühendatakse tee esimese tipuga (rida 20), et leiduks tsükkel ning kaare pikkus valitakse selline, et tsükli pikkus jääks vahemikku $(-21, 0)$ (rida 21). Seejärel genereeritakse suvalised kaared ülejäänud tippude ühendamiseks tsükliga (read 22-23).

Ülejäänud kaarte genereerimiseks, pannakse kõik kasutamata kaared hulka E_p (rida 24), kust valitakse suvalisi kaari, kuni $|E| = m$ (rida 25). Kui kaar kuulub esimesse osasse, on oluline, et see ei põhjustaks seal negatiivset tsüklit ning selle pikkus arvutatakse sarnaselt negatiivse tsüklita Bellmann-Fordi algoritmiga (rida 28). Kui ei kuulu, siis kaare pikkus läbimängu ei mõjuta ning valitakse suvaliselt vahemikust $(-10, 10)$ (rida 30).

Antud : Tippude arv n, kaarte arv m ning kaugusega $-\infty$ tippude arv u
Tagastab : Raskusparameetritele vastav graaf, millel saab jooksutada Bellmann-Fordi algoritmi ning sisaldab negatiivset tsüklit

```
1  V := {v_0, ..., v_{n-1}} , E := ∅ ;
2  t := suvaline arv vahemikust (1, u + 1);
3  E_p := {(v_i, v_j) | i < j < n − u} ;
4  E_p2 := {(v_i, v_j) | i < n − u, j ≥ n − u};
5  if n > u then
6      v_0.kaugus := 0;
7      for i := n − 1 − u to 0 do
8          (v_j, v_i) ⇐ suvaline kaar hulgast E_p, mille lõpptipp on v_i;
9          (v_j, v_i).pikkus := suvaline arv vahemikust (−10, 10);
10         v_i.kaugus := v_j.kaugus + (v_j, v_i).pikkus;
11         E ⇐ (v_j, v_i);
12     (v_i, v_j) ⇐ suvaline kaar hulgast E_p2;
13     (v_i, v_j).pikkus := suvaline arv vahemikust (−10, 10);
14     E ⇐ (v_i, v_j) ; // Edaspidi öeldakse selle 3 rease ploki (read
        12-14) kohta kaare (v_i, v_j) valimine ja väärtustamine
15 E_p3 := {(v_i, v_j) | i ≥ n − u, j ≥ n − u, i ≠ j};
16 s := 0;
17 for i := n − u to n − u + t − 1 do
18     Hulgast E_p3 valitakse ja väärtustatakse kaar (v_i, v_{i+1}); // vt rida 14
19     s := s + (v_i, v_{i+1}).pikkus;
20 E ⇐ (v_{n−u+t−1}, v_{n−u}) ⇐ E_p3 ;
21 (v_{n−u+t−1}, v_{n−u}).pikkus := −s − suvaline arv vahemikust (0, 21);
22 for i := n − u + t to n do
23     Hulgast E_p3 valitakse ja väärtustatakse kaar (v_j, v_i), kus n − u ≤ j < i;
       // vt rida 14
24 E_p := E_p ∪ E_p2 ∪ E_p3;
25 for k := n to m do
26     (v_i, v_j) ⇐ suvaline kaar hulgast E;
27     if i < n − u and j < n − u then
28         (v_i, v_j).pikkus :=
            v_j.kaugus − v_i.kaugus + suvaline arv vahemikust (1, 21);
29     else
30         (v_i, v_j).pikkus := suvaline arv vahemikust (−10, 10);
31     E ⇐ (v_j, v_i);
```

Joonis 8. Bellmann-Fordi algoritmi negatiivsete tsüklitega sisendi genereerija

2.5 Floyd-Warshalli algoritm

Algoritmi eesmärk on leida negatiivsete tsükliteta graafis kaugused kõikide tippude vahel. Algoritmi võib ka kasutada negatiivsete tsüklite leidmiseks.

2.5.1 Floyd-Warshalli algoritmi realiseerimine

Algoritm põhineb kaugustemaatriksi pideval täiendamisel [6]. n-tipulise graafi *kaugusmaatriks* on $n \times n$ maatriks $A = (a_{ij})$, kus a_{ij} sisaldab mingi tee, tipust v_i tippu v_j, pikkust või juhul, kui sellise tee olemasolu on veel selgumisel, väärtust $+\infty$ [6]. Algoritm on realiseeritud joonisel 9.

```
kauguste_maatriks(A, n)
 - - - Antud: n-tipulise graafi tippude naabrusmaatriks
 - - -          A = (a_ij, i = 1, 2, ... , n, j = 1, 2, ... , n)
 - - - Tulemus: A on teisendatud tippudevaheliste kauguste maatriksiks

⋆ t = 1, 2, ... , n

   - - - maatriksi A elemendiks a_ij on i-nda tipu ja j-nda tipu vaheline
   - - - kaugus mööda teid, mis läbivad vähemalt ühte tippudest
   - - - nr. 1, 2, ... , t − 1

  ⋆ i = 1, 2, ... , n
  ⋆ j = 1, 2, ... , n

     - - - korrigeerida a_ij, kui tee i-ndast tipust j-ndasse tippu läbi
     - - - t-nda tipu on lühem senini leitud lühimast ühendusteest nende
     - - - tippude vahel
    [? (a_it + a_tj < a_ij) a_ij := a_it + a_tj
```

Joonis 9. Floyd-Warshalli algoritmi realiseering Kiholt [6]

2.5.2 Floyd-Warshalli algoritmi sisendi genereerimine

Kuna algoritmi igas tsüklis käiakse läbi kõik tipud, on tippude arv peamine raskusparameeter. Teiseks raskusparameetriks on kaarte arv - antud kaugusmaatriksis täidetud lahtrite arv, kuna tudengi jaoks on kergem teha parandusi, kus osaleb väärtus $+\infty$. Viimaseks raskusparameetriks on paranduste (edukate korrigeerimiste, kus ei osale väärtus $+\infty$) arv.

Antud : Tippude arv n, kaugusmaatriksisse lisamata kaarte arv m
Tagastab : Raskusparameetritele n ja m vastav negatiivsete tsükliteta kaugusmaatriks A

```
1  V := v_0, ..., v_n;
2  A := (a_ij)_{i=1,...,n,j=1,...,n};
3  E_p := {(v_i, v_j) | i ≠ j};
4  a_ij := 0, kui i = j, muidu +∞;
5  v_0.kaugus := 0;
6  for i := 1 to n + 1 do
7      if m > 0 then
8          (v_j, v_i) ⇐ suvaline kaar hulgast E_p;
9          (v_j, v_i).pikkus := suvaline arv vahemikust (-10,10);
10         v_0.kaugus := v_j.kaugus + (v_j, v_i).pikkus;
11         if j > 0 then
12             a_ji := (v_j, v_i).pikkus;
13             m − −;
14 while m > 0 do
15     (v_j, v_i) ⇐ suvaline kaar hulgast E_p;
16     (v_j, v_i).pikkus := v_i.kaugus − v_j.kaugus + suvaline arv vahemikust (0,20);
17     if j > 0 and i > 0 then
18         a_ji := (v_j, v_i).pikkus;
19         m − −;
```

Joonis 10. Floyd-Warshalli algoritmi suvalise sisendi genereerija

Algoritmi raskuse tõttu valmis töö raames ainult genereerimise metoodika (joonis 11), kus genereeritakse suvalisi sisendeid fikseeritud tippude ja kaarte arvuga (joonis 10), kuni leitakse sobiva paranduste arvuga sisend. Paranduste arv leitakse joonisel 11 modifitseeritud Floyd-Warshalli algoritmi abil, mis töö käigus loeb kokku tehtud parandused (read 7-12).

Sisendi genereerimine töötab sarnaselt Floyd-Warshalli algoritmi sisendi negatiivsete

tsükliteta osa genereerimisega. Erinevuseks on, et siin pole sisendilt vaja eeldada sidusust. Selleks, et saavutataks mittesidusaid sisendeid, kasutatakse genereerimisel liigset tippu v_0, millega intsidentseid kaari kaugusmaatriksisse A ei kanta.

Kõigepealt genereeritakse vajalike tippude hulk V, tagastatav kaugusmaatriks A ning potentsiaalselt graafi lisatavate kaarte hulk E_p (read 1-3). Maatriksi A elemendid väärtustatakse (rida 4). Kui element a_{ij} asub peadiagonaalil ($i = j$), määratakse sellele väärtus 0, kuna tipu kaugus iseendast on 0. Kui element asub kusagil mujal, siis saab sellele väärtus $+\infty$, kuna kaart (teed) tipust v_i tippu v_j pole veel genereeritud. Seejärel genereeritakse kauguste puu tipust v_0 (read 5-13), kusjuures sobivate kaarte pikkused lisatakse kaugusmaatriksisse (rida 12) ning seda tehes vähendatakse kaarte arvu m (rida 13) näitamaks, et ollakse ühe kaare võrra lähemal eesmärgile. Lõpuks genereeritakse ülejäänud kaared (read 14-19) selliste pikkustega, et ei moodustuks negatiivseid tsükleid (rida 16) ning sobivad kaared lisatakse kaugusmaatriksisse (rida 18).

Antud : Tippude arv n, kaugusmaatriksisse lisamata kaarte arv m ning paranduste arv p

Tagastab : Raskusparameetritele n, m ja p vastav negatiivsete tsükliteta kaugusmaatriks A

```
1  p_c := 0;
2  A := (a_ij);
3  while p_c ≠ p do
4      Genereerida suvaline raskusparameetritele n ja m vastav kaugusmaatriks A
         joonisel 10 toodud meetodiga;
5      A_enneFW := A;
6      p_c := 0;
7      for k := 1 to n + 1 do
8          for i := 1 to n + 1 do
9              for j := 1 to n + 1 do
10                 if a_ij > a_ik + a_kj and a_ij ≠ +∞ then
11                     a_ij := a_ik + a_kj;
12                     p_c++;
13     A := A_enneFW;
```

Joonis 11. Floyd-Warshalli algoritmi sisendi genereerija

3 Programmi kasutamine

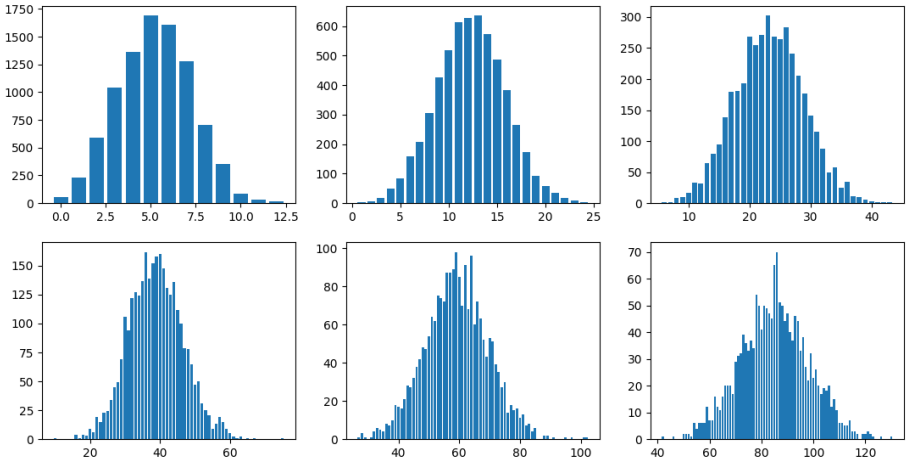
Peatükis selgitatakse, kuidas peaks programmi kasutama ning mida programm täpselt teeb. Programm on praegusel kujul mõeldud peamiselt sisendite genereerimise demonstreerimiseks ning praktiliseks kasutamiseks on tõenäoliselt vajalik kasutajal juurde programmeerida talle oluline lisafunktsionaalsus.

3.1 Programmi üldine kasutamine

Programm on tavaline *java*-fail, mida saab käivitada ning muuta. Programmile tuleb käivitamisel käsurea argumentidena anda algoritmi nime esitäht, millele sisendit soovitakse: (K)ahni algoritm, (D)ijkstra algoritm, (B)ellmann-Fordi algoritm, (n)egatiivse tsükliga Bellmann-Fordi algoritm, (F)loyd-Warshalli algoritm. Tähele peab järgnema tühikutega eraldatult vajalik kogus raskusparameereid. Valest arvust või vales vahemikus sisestatud käsureaargumentidest kasutajat teavitatakse. Sobiva väljakutse korral väljastab programm käivitamisekeskkonda sisendiks oleva graafi DIMACS-formaadis, mille seletuse leiab allikast [8], või Floyd-Warshalli algoritmi korral CSV-formaadis. Väljastatud sisendil on tippude numbrid valitud suvaliselt ning ei ole seotud sisendi genereerimisel kasutatud indeksitega va Floyd-Warshalli sisendite puhul, kus tippude numbreid ei väljastata.

3.2 Floyd-Warshalli algoritmi sisendite genereerimise erisused

Kuna genereerimine põhineb juhuslikusel, pole Floyd-Warshalli algoritmile sisendite genereerimisel võimalik täpselt määrata, kui kaua mingite raskusparameetrite järgi sisendi genereerimine aega võtab või kas selline sisend üldse on olemas. Soovitatav on enne raskusparameetrite valimist kontrollida, millise paranduste arvuga kaugusmaatrikse valitud tippude arvu ja kaarte arvu korral genereeritakse. Joonisel 12 on toodud joonisega 10 analoogse Pythoni programmi genereeritud sisendite jaotused 4-9 tipuliste sisendite korral. Igal graafikul on x-teljeks paranduste arv ning y-teljeks 1 sekundi jooksul genereeritud vastava paranduste arvuga sisendite arv. Sisendid on genereeritud maksimaalse kaarte arvuga. Väiksema paranduste arvuga sisendite genereerimiseks on soovitatav vähendada kaarte arvu. On tõenäoline, et suurendades vahemikke, kust genereeritakse kaarte pikkused, genereeritakse rohkem suurema paranduste arvuga sisendeid.



Joonis 12. Genereeritud sisendite jaotused

Kokkuvõte

Töös koostati algoritmid, mille abil on graafialgoritmidele võimalik genereerida sisendeid vastavalt etteantud raskusparameetritele, ning nende algoritmide realiseering programmis. Kahni algoritmile loodi sisendeid tippude ja kaarte arvu põhjal. Dijkstra algoritmile loodi sisendeid tippude, kaarte ja paranduste arvu põhjal. Bellmann-Fordi algoritmile loodi sisendeid tippude ja kaarte arvu ning kauguste puu kõrguse põhjal, kui sisendis ei tohtinud olla negatiivset tsüklit, ning tippude, kaarte ja negatiivse tsükli mõjualas olevate tippude arvu põhjal, kui negatiivne tsükkel pidi sisendis leiduma. Floyd-Warshalli algoritmile loodi sisendite generaator, mis genereeris suvalisi kaugusmaatrikse etteantud tippude ning kaarte arvu järgi, kuni leidis maatriksi, millel Floyd-Warshalli algoritmi käitades tehti etteantud arv parandusi, ning tagastas selle maatriksi. Koostatud programm demonstreeris iga graafialgoritmi sisendi genereerimist.

Töö esialgsest eesmärgist jäi puudu lahenduste hindamise programmi koostamine. Hindamise programmi koostamiseks fikseeriti iga algoritmi puhul raskusparameetrid, mis aitavad sellise programmi koostamisel võrrelda erinevaid tudengite poolt tehtud vigu numbriliselt. Floyd-Warshalli algoritmi sisendeid saab genereerida kohe paranduste arvu järgi. Sellist lähenemist võimaldava algoritmi leidmise korral on võimalik, et sisendeid saaks genereerida suuremast raskusparameetrite vahemikust ning kiiremini.

Aines „Algoritmid ja andmestruktuurid“ käsitletakse veel palju teisi algortme, millest osadele pole veel sisendi generaatoreid tehtud. Need generaatorid oleksid sama kasulikud, kui töös leitud. Ühelegi algoritmile pole veel tehtud lahenduste hindamise programmi ega harjutamiskeskkonda. Graafialgoritmide puhul on vaja leida ja kohandada graafide visualiseerimise meetodeid, et tudengitele oleks võimalik anda ülesandeid graafi, mitte ploki numbrite kujul. Kõik need lahendused peaksid valmima DEEPMOOC platvormi arendamise raames.

Viidatud kirjandus

[1] *DEEPMOOC platvormi arendamine*. (09.05.2022). URL: https://comserv.cs.ut.ee/ati_thesis_offers/datasheet.php?id=73312&year=2021.

[2] *Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) üldinfo ÕISis*. (25.04.2022). URL: https://ois2.ut.ee/#/courses/LTAT.03.005/details.

[3] *Tartu Ülikooli aine „Algoritmid ja andmestruktuurid“ (LTAT.03.005) moodle's olevad materjalid.* (12.01.2022). URL: https://moodle.ut.ee/course/view.php?id=182.

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

[5] Sander Tiganik. *Kaksiksidusate graafide visualiseerija*. TÜ arvutiteaduse instituudi bakalaureusetöö. 2015.

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

[7] Reimo Palm. *Diskreetse matemaatika elemendid*. Tartu Ülikooli Kirjastuse trükikoda, 2009. ISBN: 9985-4-0354-1.

[8] *Satisfiability Suggested Format. Center for Discrete Mathematics Theoretical Computer Science (DIMACS)*. 1993, lk. 8.

Lisad

I. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Uku Hannes Arismaa**,
*(*autori nimi)

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

   Graafialgoritmide sisendite genereerimine,
   *(*lõputöö pealkiri)

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

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

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

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

Uku Hannes Arismaa
10.05.2022