UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Pavle M. Gavrilović

ALGORITMI ZA GENERISANJE LISTI NEIZOMORFNIH GRAFOVA

master rad

Beograd, 2022.

UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Pavle M. Gavrilović

ALGORITMI ZA GENERISANJE LISTI NEIZOMORFNIH GRAFOVA

master rad

Beograd, 2022.

Mentor:

dr Miodrag Živković, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Filip Marić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Vesna Marinković, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: 30. 9. 2022.

Naslov master rada: Algoritmi za generisanje listi neizomorfnih grafova

Rezime: U radu je opisan i realizovan algoritam za kanonizaciju grafova koji je prikazan u radu [1]. Taj algoritam je dalje upotrebljen za realizaciju nekoliko algoritama za generisanje kataloga povezanih grafova.

Pored toga opisan je i metod za uspešno generisanje kataloga neizomorfnih objekata, kao i jedan veoma efikasan algoritam za generisanje svih stabala sa n čvorova.

Ključne reči: grafovi, kanonizacija, izomorfizam, stabla

Sadržaj

Glava 1

Uvod

Grafovi i njihova svojstva predstavljaju jednu od najznačajnijih oblasti istraživanja u računarstvu i šire. Popularnost grafova prouzrokovana je činjenicom da se veliki broj raznorodnih pojava i problema mogu modelovati tako jednostavnim formalizmom.

Zaista, intuitivno, graf je ništa više do crteža tačaka od kojih su pojedine povezane dužima, ali upravo u toj jednostavnosti odnosno velikom stepenu apstraktnosti leži njihova moć - da opišu gotovo svaki vid veze među nekim objektima. Tako, na primer, grafovi mogu modelovati mrežu saobraćajnica, gde bi tačke predstavljale gradove, opštine ili države dok bi duži bile magistrale koje ih spajaju. Recimo u hemiji, svi molekuli, odnosno atomi i njihove kovalentne ili jonske veze, mogu se modelovati grafovima. Isto tako, grafovi mogu modelovati i veze među planetama, zvezdama ili galaksijama, te se zaista vidi sveopšta prisutnost grafova, od reprezentacije najsitnijih čestica do prikaza čitavog univerzuma. Naravno, ne samo prirodne, već i razne društvene interakcije, sociološke pojave i odnosi, se mogu predstaviti grafom.

Dodatno, grafovi se mogu i „obogatiti" tako što se tačkama pridružuje neka oznaka ili boja, a slično se može učiniti i sa dužima koje povezuju tačke. Tako se, na primer, mogu razlikovati atomi ugljenika, vodonika i kiseonika, prilikom predstavljanja nekog organskog jedinjenja ili, pak, ako se traži najkraći put od jednog mesta do drugog, dužima se mogu dodeliti brojevi koji bi predstavljali udaljenosti mesta (kuća, gradova, država...) koja se modeluju.

Pored ovog, slikovitog, prikaza gde sve grafovi mogu naći svoju primenu, važno je istaći da se mnogi algoritamski ili matematički problemi mogu svesti na poznate „grafovske" probleme te tako lako rešiti. Zbog toga je interesovanje za izučavanje

grafova, njihovih svojstava kao i algoritama vezanih za njih, neprestano prisutno u naučnoj zajednici.

Često je, u tim istraživanjima, od koristi imati listu grafova koji imaju određena svojstva - svojevrsni „katalog" grafova određenog tipa. Ono što je značajno jeste da lista ne sadrži „duplikate". Oni mogu nepotrebno da smetaju i oduzimaju vreme prilikom izučavanja te liste. Nije teško zamisliti neku „skupu" operaciju koja se mora izvršiti nad grafovima, u tom slučaju, jasno je da postoji želja da se izbegne njeno izvršavanje nad grafovima koji su isti. Upravo analiziranjem nekih kolekcija grafova mogu se možda uočiti neka nova svojstva i izvući novi zaključci o toj kolekciji. Sa druge strane, imati dostupnu kolekciju grafova može poslužiti i za brzo pronalaženje kontraprimera nekim hipotezama.

Kako je ovo problem kojim su se mnogi, sada već decenijama, bavili, razvijen je veliki broj algoritama za njegovo rešavanje. Verovatno najpoznatiji alat za generisanje grafova jeste *geng*, koji se nalazi u okviru programa *nauty* [9]. Takođe, postoje i drugi programi, kao i oni programi koji su specijalizovani za generisanje samo grafova sa određenim svojstvima.

Fokus ovog rada je na algoritmima za generisanje povezanih neusmerenih grafova[1] i njihova implementacija. Pored algoritama za generisanje listi povezanih grafova, prikazuje se i algoritam za svođenje grafa na *kanonski* oblik, koji omogućava izbegavanje pojave „duplikata" u listama.

1.1 Osnovni pojmovi

U nastavku su opisani osnovni pojmovi vezani za grafove, a koji se spominju i koriste u radu. Pojmovi koji su uže vezani za neki konkretni algoritam su navedeni i objašnjeni u glavi koja se tiče tog algoritma.

Graf je uređeni par $G = (V, E)$ gde je:

- V konačan neprazan skup objekata koji se najčešće nazivaju **čvorovi** (*eng. vertices*),
- $E \subseteq \{\{x, y\} \mid x, y \in V \wedge x \neq y\}$ skup **grana** (*eng. edges*), odnosno neuređenih parova čvorova.

U nekim situacijama može biti od interesa da se dozvoli $x = y$ tj. prisustvo petlji (čvor je povezan sam sa sobom). Takođe, na ovaj način definisan graf je **neusmeren**

[1] Objašnjenje pojmova dato u nastavku

(*eng. undirected*), dok ako bi grane bile definisane kao uređeni parovi tada je u pitanju **usmeren** (*eng. directed*) graf.

Stepen čvora v jeste broj grana koje taj čvor povezuju sa ostalim čvorovima u grafu. **Regularan** (*eng. regular*) graf je onaj kod koga su svi čvorovi istog stepena.

Put (*eng. walk*) od v_1 do v_k je niz čvorova $v_1, v_2, \ldots, v_k$ povezanih granama $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$. **Ciklus** (*eng. cycle*) je takav put kome se početni i završni čvor poklapaju. Čvor u je **dostižan** iz čvora v ako postoji put[2] od v do u. **Neusmereni oblik** usmerenog grafa jeste isti taj graf kada se njegove grane posmatraju kao „neusmerene". Graf je **povezan** (*eng. connected*) ako u svom neusmerenom obliku sadrži put između bilo koja dva čvora.

Stablo (*eng. tree*) je povezan graf koji ne sadrži cikluse.

Kada je u pitanju reprezentacija grafova, najčešće se susreću sledeća dva vida reprezentacije. Prvi je **matrica povezanosti** ili **matrica susedstva** (*eng. adjacency matrix*) grafa. Za graf $G = (V, E)$ gde je $|V| = n$ i $V = \{v_1, v_2, \ldots, v_n\}$, matrica povezanosti je kvadratna matrica $A = (a_{ij})$ reda n, za koju važi:

- $a_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$
- $a_{ij} = 0 \Leftrightarrow (v_i, v_j) \notin E$

Kako je fokus ovog rada na generisanju neusmerenih grafovima bez petlji, treba istaći da su matrice povezanosti takvih grafova simetrične i isključivo sadrže nule na svojoj glavnoj dijagonali[3]. Mana ovog načina predstavljanja grafova jeste što, bez obzira na broj grana prisutnih u grafu, uvek se barata sa celom $n \times n$ matricom.

Ako se uklone eksplicitno predstavljene nepostojeće grane, odnosno nule u matrici povezanosti, dolazi se do drugog vida reprezentacije grafa - **liste povezanosti** ili **liste susedstva** (*eng. adjacency list*). Dakle, svaki čvor ima listu njemu susednih čvorova, te se ceo graf predstavlja jednim nizom (dužine n) listi različitih dužina (u zavisnosti od broja suseda svakog čvora).

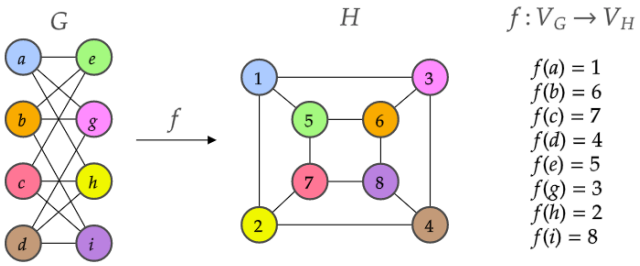
Dva grafa $G = (V_G, E_G)$ i $H = (V_H, E_H)$ su **izomorfna** (*eng. isomorphic*) ako postoji bijektivno preslikavanje $f : V_G \to V_H$, tako da za svaki par čvorova $(u, v) \in V_G \times V_G$ važi

$$(u, v) \in E_G \Leftrightarrow (f(u), f(v)) \in E_H.$$

Primer dva izomorfna grafa, kao i funkcije koja vrši preslikavanje jednog skupa čvorova u drugi je dat na slici 1.1.

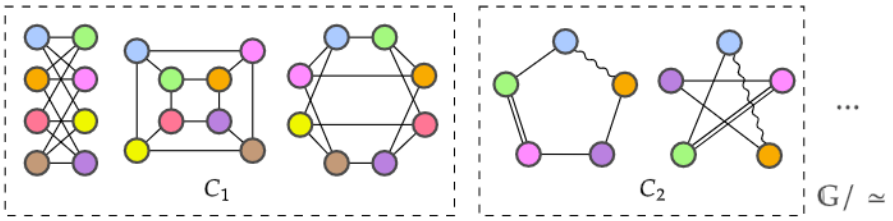
[2]put mora da poštuje usmerenje grana, kada se radi o usmerenom grafu

[3]kroz čitav rad se misli na glavnu dijagonalu, ako nije naglašeno suprotno



Slika 1.1: Primer izomorfnih grafova

Izomorfni grafovi predstavljaju one „duplikate" koje je, kako je već istaknuto, potrebno izbeći prilikom generisanja liste svih grafova. Pošto *biti izomorfan* predstavlja relaciju ekvivalencije, u oznaci $\simeq$, odatle sledi da se svi grafovi mogu podeliti ili grupisati u klase ekvivalencije (videti sliku 1.2). U svakoj klasi je moguće izabrati jedan graf za predstavnika cele klase. Taj predstavnik se proglašava za **kanonski oblik** ili **kanonsku formu** - $Canon(G)$ grafa G.



Slika 1.2: Primeri klasa ekvivalencije na skupu svih grafova

Koji graf će biti izabran za predstavnika neke klase ekvivalencije tj. proglašen za kanonski oblik grafova te klase je odluka koja se može doneti u zavisnosti od konkretnog problema koji se rešava. Ta odluka može olakšati ili čak omogućiti rešavanje problema, te ima smisla posvetiti joj pažnju prilikom pristupanja potrazi za rešenjem. U svakom slučaju, kanonski graf[4] - $Canon(G)$ mora da zadovoljava sledeća svojstva:

- $Canon(G)$ postoji i jedinstven je za svaki graf G
- $Canon(G) = g \cdot G$ za neko $g \in S_n$
- $Canon(g \cdot G) = Canon(G)$ za svako $g \in S_n$

gde je n broj čvorova grafa G, a S_n skup svih permutacija skupa od n elemenata.

[4]onaj koji je proglašen za kanonski oblik njemu izomorfnih grafova

Glava 2

Svođenje grafova na kanonski oblik

Kao što je već istaknuto u prethodnoj glavi, prilikom generisanja liste grafova cilj je izbeći pojavu „duplikata", odnosno izomorfnih grafova. Da bi se to učinilo, neophodno je da postoji način da se utvrdi da li su dva grafa izomorfna.

Problem ispitivanja izomorfnosti dva grafa je jedan od retkih algoritamskih problema čija vremenska složenost, još uvek, nije poznata. Naime, nije otkriven nijedan algoritam koji bi ovaj problem rešio u polinomijalnom vremenu, ali sa druge strane, nije poznato ni da li je ovaj problem NP-kompletan. Do sada najbolje vremensko ograničenje dao je Laslo Babai (*mađ. László Babai*), dokazavši da je gornje vremensko ograničenje za ovaj problem kvazi-polinomijalno $2^{O(\log^c(n))}$ [2]. Za praktične potrebe, ipak su najbolji programi koje su razvili Brenden Mekej (*eng. Brendan McKay*) i Adolfo Piperno - *nauty i Traces* [9]. Tu su još neki programi, kao što su *bliss* [6], *conauto* [7], *saucy* [3].

Jedan od načina da se utvrdi da li su dva grafa izomorfna jeste svođenjem na njihove kanonske oblike, koji se potom mogu jednostavno uporediti. U ovom radu je prikazan jedan takav algoritam, čija se implementacija kasnije koristi prilikom generisanja listi povezanih neizomorfnih grafova u glavi 3.

Algoritam za svođenje konačnih neusmerenih grafova na kanonski oblik opisan u radu [1] je bio namenjen prvenstveno za kanonizaciju jako regularnih grafova,[1] ali se može upotrebiti i za neusmerene grafove koji ne sadrže petlje i maksimalno imaju jednu granu između bilo koja dva čvora.

[1] za jako regularan graf, pored toga što je regularan (str. 3), važi dodatno da postoje dva cela broja λ i μ takvi da:

- svaka dva susedna čvora imaju λ zajedničkih suseda
- svaka dva ne-susedna čvora imaju μ zajedničkih suseda

Neka su $A = (a_{ij})$ i $B = (b_{ij})$ dve različite $(0,1)$-matrice reda n. Matrica A je veća od matrice B, u oznaci $A \succ B$, ako je n^2-dimenzioni vektor $(a_{11} \dots a_{1n} a_{21} \dots a_{2n} \dots a_{nn})$ *leksikografski*[2] veći od vektora $(b_{11} \dots b_{1n} b_{21} \dots b_{2n} \dots b_{nn})$. Na osnovu ovog poretka, može se definisati **maksimalna forma** matrice A na sledeći način:

$$\widetilde{A} = \max_{g \in S_n} gAg^{-1},$$

gde je n broj čvorova matrice A, dok je g element skupa S_n koji predstavlja skup svih permutacija skupa od n elemenata[3]. Permutacija g određuje jednu matricu permutacije P_g koja razmešta vrste, ako se primeni sa leve strane matrice A. Kako je A matrica povezanosti grafa, potrebno je da veze između čvorova ostanu nepromenjene, te se na matricu povezanosti sa desne strane mora primeniti matrica permutacije P_g^{-1} ili ekvivalentno,[4] P_g^T, kako bi kolone zadovoljile željeni razmeštaj vrsta.

U suštini, maksimalna forma se dobija promenom redosleda vrsta zajedno sa kolonama matrice A tako da vektor $(a_{11} \dots a_{1n} a_{21} \dots a_{2n} \dots a_{nn})$ bude maksimalan leksikografski gledano.

Ako A predstavlja matricu povezanosti nekog grafa G, tada graf $\widetilde{G}$, čija je matrica povezanosti upravo maksimalna forma matrice A tj. $\widetilde{A}$, ima sva svojstva kanonskog grafa (str. 4).

Za dalje potrebe uvodi se sledeća notacija:

$$\mathrm{Min}_k(A) = (a_{ij}),\ i,j = 1,2,\dots,k,$$

$$\mathrm{Min}_{k,r}(A) = \begin{bmatrix} & & & a_{1,r} \\ & \mathrm{Min}_k(A) & & \vdots \\ & & & a_{k,r} \\ a_{r,1} & \cdots & a_{r,k} & a_{r,r} \end{bmatrix}$$

Dakle, $\mathrm{Min}_k(A)$ je podmatrica matrice A reda k, dok je $\mathrm{Min}_{k,r}$ podmatrica matrice A reda $k+1$ koja predstavlja proširenje $\mathrm{Min}_k(A)$ vrstom i kolonom r.

Matrica A je **monotonična** (eng. *monotonic*)[5], ako za bilo koje $k = 1,2,\dots,n$, važi:

$$\mathrm{Min}_k(A) = \max_{k \le r \le n} \mathrm{Min}_{k-1,r}(A)$$

[2]poredak koji određuje redosled reči u rečniku

[3]bilo koja bijekcija $g : A \to A$ jeste permutacija skupa A. Skup svih takvih permutacija se označava sa S_A, dok u slučaju da je $A = \{1,2,\dots,n\}$ za neki prirodan broj n, koristi se oznaka S_n

[4]kako su permutacione matrice ortogonalne tj. $PP^T = I$ važi $P^{-1} = P^T$

[5]razlikovati od pojma monotona matrica: kvadratna matrica A, reda n, je monotona *akko* $Av \ge 0$ za svako $v \in R^n$

Matrica gAg^{-1}, $g \in S_n$ koja jeste monotonična, predstavlja **monotoničnu formu** matrice A.

Slično, matrica $\mathrm{Min}_s(A)$ takva da za $k = 1, 2, \ldots, s$ $(s < n)$ važi, kao i malopre:

$$\mathrm{Min}_k(A) = \max_{k \le r \le n} \mathrm{Min}_{k-1,r}(A)$$

se zove **monotonični minor** matrice A.

Jasno je da su sve glavne podmatrice[6] monotoničnog minora, a samim tim i monotonične matrice, i same monotonične.

Svi ovi pojmovi su uvedeni kako bi se formulisalo centralno tvrđenje rada [1], koje je osnov za odabrani algoritam svođenja matrice na kanonski oblik.

Teorema 2.1. *Ako je A simetrična (0,1)-matrica sa nulama na dijagonali, njena maksimalna forma je monotonična.*

Dokaz. Pretpostavimo suprotno, da maksimalna forma $\widetilde{A}$ matrice A nije monotonična. Neka je $\mathrm{Min}_k\left(\widetilde{A}\right)$, $k \le n$ monotonični minor $\widetilde{A}$, ali takav da važi $\mathrm{Min}_{k,r}(\widetilde{A}) \succ \mathrm{Min}_{k+1}(\widetilde{A})$, za neko r, $k+1 < r \le n$.

Može se zaključiti da postoji neko t, $t < k$, takvo da je $a_{i,r} = a_{i,k+1}$ za $1 \le i \le t$, ali da je $a_{t+1,r} > a_{t+1,k+1}$. Da nije tako, odnosno da je $a_{i,r} = a_{i,k+1}$ za $1 \le i \le k$, zbog simetričnosti matrice $\widetilde{A}$ i $a_{r,r} = a_{k+1,k+1} = 0$ (nule na dijagonali), sledilo bi da je $\mathrm{Min}_{k,r}(\widetilde{A}) = \mathrm{Min}_{k+1}(\widetilde{A})$, što ne odgovara polaznoj pretpostavci.

Neka je $g \in S_n$ transpozicija[7] brojeva $k+1$ i r. Tada za matrice $g\widetilde{A}g^{-1}$ i $\widetilde{A}$ važi da su im prvih t vrsta iste, ali da je $(t+1)$-a vrsta matrice $g\widetilde{A}g^{-1}$ veća od $(t+1)$-e vrste matrice $\widetilde{A}$, samim tim je $g\widetilde{A}g^{-1} \succ \widetilde{A}$, što je kontradikcija sa pretpostavkom da je $\widetilde{A}$ maksimalna forma matrice A. □

Kako su matrice povezanosti neusmerenih grafova upravo simetrične (0,1)-matrice sa nulama na dijagonali, teorema 2.1 omogućava da se potraga za maksimalnom formom matrice svede na generisanje monotoničnih formi matice A i traženje maksimalne među njima.

Ako je $V^{(k)} = (i_1, \ldots, i_k)$ *parcijalna permutacija*[8] od k različitih brojeva $(0 \le k \le n)$ odnosno k čvorova i_s, $1 \le i_s \le n$, neka je:

[6] matice koje se dobijaju brisanjem i-te vrste i kolone početne matrice

[7] permutacija koja menja mesta samo dva elementa, dok ostale fiksira

[8] susreće se i termin *r-permutacija*. Neka je S skup od n elemenata i neka je r prirodan broj, takav da je $r \le n$. Tada je *r-permutacija* skupa S jedna r-torka koju čine međusobno različiti elementi skupa S

$$\mathrm{Min}_{V^{(k)}}(A) = \begin{bmatrix} a_{i_1,i_1} & \cdots & a_{i_1,i_s} & \cdots & a_{i_1,i_k} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i_s,i_1} & \cdots & a_{i_s,i_s} & \cdots & a_{i_s,i_k} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i_k,i_1} & \cdots & a_{i_k,i_s} & \cdots & a_{i_k,i_k} \end{bmatrix}$$

Dakle, $V^{(k)}$ određuje jednu podmatricu, $\mathrm{Min}_{V^{(k)}}(A)$, matrice A. Jasno je da za svako $V^{(k)}$ postoji $g \in S_n$ tako da $\mathrm{Min}_{V^{(k)}}(A) = \mathrm{Min}_k(gAg^{-1})$.

Neka je sa $T(k)$ je označen skup svih parcijalnih permutacija[9] $V^{(k)}$, takvih da je $\mathrm{Min}_{V^{(k)}}(A)$ monotonični minor matrice gAg^{-1}, za neko $g \in S_n$ i neka je

$$T = \bigcup_{i=0}^{n} T(i)$$

unija svih tih skupova. Neka je $V^{(k)} = (i_1, \ldots, i_k)$ i $V^{(k+1)} = (i_1, \ldots, i_k, i_{k+1})$, gde je $i_{k+1} \in \{1, 2, \ldots, n\} \backslash V^{(k)}$. U ovom slučaju $V^{(k)}$ je **sadržana** u $V^{(k+1)}$, odnosno $V^{(k+1)}$ je **proširenje** $V^{(k)}$. Na osnovu definicije monotoničnih minora, ako je $V^{(k+1)} \in T(k+1)$, tada permutacija $V^{(k)}$, koja je sadržana u permutaciji $V^{(k+1)}$, odgovara monotoničnom minoru $\mathrm{Min}_k(\mathrm{Min}_{k+1}(A))$, te sledi da je $V^{(k)} \in T(k)$. Obrnuto, ako je $V^{(k)} \in T(k), k < n$, može se naći permutacija (nekad i više od jedne) $V^{(k+1)} \in T(k+1)$, koja je proširenje permutacije $V^{(k)}$.

Dakle, može se zaključiti da je skup T parcijalno uređen skup u odnosu na gore opisanu relaciju između permutacija $V^{(k)}$ i $V^{(k+1)}$ i formira jedno stablo sa korenom $T(0) = \emptyset$, gde se $V^{(k)}$ nalazi na udaljenosti k od korena. Iz konstrukcije ovog stabla, sledi da su permutacije $V^{(n)} \in T(n)$ listovi stabla T i da svaka matrica koju oni određuju, $\mathrm{Min}_{V^{(n)}}(A)$, jeste jedna monotonična forma matrice A. Takođe, važi i obrnuto, za svaku monotoničnu formu, $\widehat{A}$, matrice A, može se pronaći list $V^{(n)} \in T(n)$ takav da je $\mathrm{Min}_{V^{(n)}}(A) = \widehat{A}$.

Podstablo stabla T čiji je koren $V^{(k)}$ označeno je sa $T(V^{(k)})$, dok je skup proširenja $V^{(k)}$ koja pripadaju stablu T označen sa $Q(V^{(k)})$.

Algoritam za pronalaženje svih monotoničnih formi matrice A se sastoji od konstrukcije opisanog stabla T i potom određivanja maksimalne matrice $\mathrm{Min}_{V^{(k)}}$ od svih matrica koje su određene na osnovu listova. Proces konstrukcije stabla se sastoji u postepenom otkrivanju proširenja $Q(V^{(k)})$ za svaku permutaciju $V^{(k)} \in T(k)$ polazeći od $T(0)$, tj. praznog skupa.

[9] u nastavku, ako je jasno da se radi o parcijalnoj permutaciji na osnovu oznake, reč „parcijalna" se izostavlja

Najzanimljiviji i najzahtevniji deo ovog algoritma jeste pronalaženje skupa proširenja $Q(V^{(k)})$ za permutacije $V^{(k)} \in T(k)$. Za matrice povezanosti neusmerenih grafova, ovo se može obaviti na efikasan način. Neka je A simetrična matrica reda n sa nulama na dijagonali. Prema definiciji monotoničnog minora, da bi bio ispunjen uslov $\{V^{(k)}, r\} \in Q(V^{(k)})$, gde je $V^{(k)} \in T(k)$ i $V^{(k)} = (i_1, \ldots, i_k)$, neophodno je da važi sledeće:

$$\begin{bmatrix} & a_{i_1,r} \\ \mathrm{Min}_{V^{(k)}}(A) & \vdots \\ & \\ a_{r,i_1} \cdots\cdots\cdots & a_{r,r} \end{bmatrix} = \max_{t \in \{1,2,\ldots,n\} \setminus V^{(k)}} \begin{bmatrix} & a_{i_1,t} \\ \mathrm{Min}_{V^{(k)}}(A) & \vdots \\ & \\ a_{t,i_1} \cdots\cdots\cdots & a_{t,t} \end{bmatrix}$$

Kako je A, kao što je već istaknuto, simetrična i $a_{i,i} = 0$ za svako $1 \leq i \leq n$, prethodni uslov je ekivalentan sledećem:

$$(a_{i_1,r}, \ldots, a_{i_k,r}) = \max_{t \in \{1,2,\ldots,n\} \setminus V^{(k)}} (a_{i_1,t}, \ldots, a_{i_k,t}),$$

gde je uređenje leksikografsko. Vidi se da je za pronalaženje proširenja $V^{(k)}$ dovoljno posmatrati i maksimizovati samo poslednju kolonu matrice $\mathrm{Min}_{\{V^{(k)},t\}}$ gde je $t \in \{1, 2, \ldots, n\} \setminus V^{(k)}$ i to bez elementa koji pripada dijagonali, jer je on svakako 0. Algoritam, koji je formalno opisan u nastavku, koristi tu činjenicu tako što u svakoj iteraciji na matricu $\mathrm{Min}_{V^{(k)}}$ dodaje novu kolonu[10] koja ima što je moguće više jedinica na svom početku, odnosno pri vrhu.

Neka $R(V^{(k)})$ označava skup čvorova $r \in \{1, 2, \ldots, n\} \setminus V^{(k)}$ takvih da je $\{V^{(k)}, r\} \in Q(V^{(k)})$. Pored toga, sa W_i je označen skup brojeva $j \in \{1, 2, \ldots, n\} \setminus V^{(k)}$ takvih da je $a_{i,j} = 1$. Dakle, W_i se smanjuje u svakoj iteraciji odnosno što se ide dublje u stablo T. Može se primetiti da je za $V^{(0)} = \emptyset$ skup svih skupova W_i

$$W = \bigcup_{i=1}^{n} W_i$$

zapravo lista povezanosti matrice A.

Neka je $R_0 = \{1, 2, \ldots, n\} \setminus V^{(k)}$, R_s se induktivno određuje po sledećem principu:

$$R_s = \begin{cases} R_{s-1}, & \text{ako je } R_{s-1} \cap W_{i_s} = \emptyset \\ R_{s-1} \cap W_{i_s}, & \text{ako je } R_{s-1} \cap W_{i_s} \neq \emptyset \end{cases}$$

[10] drugim rečima, bira čvor koji je susedan što većem broju čvorova sa početka uređenog skupa $V^{(k)}$

Teorema 2.2. *Ako je skup R_s konstruisan na gore opisan način za $V^{(k)} = \{i_1, \ldots, i_k\}$, $V^{(k)} \in T(k)$, onda R_s sadrži brojeve $j \in \{1, 2, \ldots, n\} \setminus V^{(k)}$, za koje važi:*

$$(a_{i_1,j}, \ldots, a_{i_s,j}) = \max_{t \in \{1,2,\ldots,n\} \setminus V^{(k)}} (a_{i_1,t}, \ldots, a_{i_s,t}).$$

Dokaz. Očigledno je da je tvrđenje tačno za $s = 0$. Pod pretpostavkom da tvrđenje važi za $s = p - 1$, potrebno je pokazati da važi i u slučaju $s = p$. Drugim rečima, potrebno je pokazati da R_p sadrži samo one brojeve $j \in R_{p-1}$, za koje je:

$$a_{i_p,j} = \max_{t \in R_{p-1}} a_{i_p,t}.$$

Međutim, iz konstrukcije sledi da

- R_p sadrži isključivo one brojeve $j \in R_{p-1}$ za koje je $a_{i_p,j} = 1$, ako oni postoje;
- $R_p = R_{p-1}$, ako je $a_{i_p,j} = 0$ za svako $j \in R_{p-1}$.

□

Iz teoreme 2.2 za $s = k$ sledi da je $R_k = R(V^{(k)})$. Dakle, moguće je na ovaj način otkriti sva proširenja skupa $V^{(k)}$, i to računajući ne više od k preseka među delovima vrsta matrice A. Može se zapaziti i da, ako se dogodi da je $|R_s| = 1$, tada je $R_{s+1} = R_s$, te je onda $R(V^{(k)}) = R_s$. Sama implementacija, čiji je pseudokod dat u algoritmu 1, blisko prati opisani postupak pronalaženja proširenja skupa $V^{(k)}$.

U najgorem slučaju, algoritam 1 mora da generiše $n!$ matrica tj. stablo T ima $n!$ listova. Ovo se lako vidi na primeru kompletnog grafa, jer je svaka permutacija čvorova kompletnog grafa maksimalna.

U algoritmu 1 se može videti da je povratna vrednost *kod* kanonskog oblika grafa. Ovo je učinjeno da bi se olakšala upotreba algoritma prilikom generisanja liste neizomorfnih grafova (više o tome u glavi 3). Ekvivalentno, ako je cilj samo svesti graf na kanonski oblik, funkcija može da vrati bilo matricu povezanosti, bilo listu povezanosti kanonskog oblika grafa[11].

Pomoću algoritma 1 lako se dolazi do algoritma za proveru da li je dati graf u kanonskom obliku ili ne. Potrebno je željeni graf svesti na njegov kanonski oblik i uporediti da li se početni graf razlikuje od svog kanonskog oblika.

[11] implementirane su pomoćne metode konverzije između svih ovih reprezentacija

Algoritam 1 Algoritam za svođenje grafa na kanonski oblik

Ulaz Graf $G = (V, E)$ predstavljen listom povezanosti LP
Izlaz Kod kanonskog grafa $\widetilde{G}$

```
1:  function ReduceToCanonicalForm(LP)
2:      n ← |V|
3:      l ← 1
4:      Q(V^(l−1)) ← [{1}, {2}, ..., {n}]        ▷ Lista svih skupova proširenja V^(0) = ∅.
5:      while l < n do
6:          for all V^(l) ∈ Q(V^(l−1)) do
7:              R ← {1, 2, ..., n} \ V^(l)
8:              for all i ∈ V^(l) do
9:                  W_i ← LP[i] \ V^(l)
10:                 if R ∩ W_i ≠ ∅ then
11:                     R ← R ∩ W_i
12:                     if |R| = 1 then
13:                         break
14:             for all r ∈ R do
15:                 Q(V^(l)).add(V^(l) ∪ {r})
16:         l ← l + 1
17:     naći V^(n)_max tj. maksimalni element liste Q(V^(n−1))
18:     return kod grafa koji odgovara V^(n)_max
```

Primer 2.0.1. *Kako bi se olakšalo razumevanje ovog algoritma u nastavku je dat konkretan primer grafa sa četiri čvora, čija je matrica povezanosti:*

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

U nastavku su prikazani koraci svođenja ovog grafa na jednu od njegovih monotoničnih formi uz prikaz matrica koje su indukovane permutacijama $V^{(k)}$. Ove matrice se ne formiraju u toku rada algoritma, ali su zgodne za ilustraciju dokle se stiglo pre nego što se stigne do listova stabla.

Počinje se sa $V^{(0)} = \emptyset$ i traga se za $R(V^{(k)})$ odnosno proširenjima permutacije V^0. U ovom slučaju to je svaki čvor iz skupa čvorova $\{1, 2, 3, 4\}$, jer svi oni, pošto graf nema petlji, određuju istu matricu:

$$\begin{bmatrix} 0 \end{bmatrix}$$

Sada je portebno odrediti proširenja za ove četiri permutacije $V^{(1)}$. Dakle, u opštem slučaju, $V^{(k)}$ sadrži čvorove koji su već „uzeti" i njima se ne može proširivati sam $V^{(k)}$. Neka je $V^{(1)} = (3)$, ostali slučajevi $V^{(1)}$ se izvršavaju na isti način. Za proširenja određuju se skupovi R_s tako što se krene od $R_0 = \{1,2,3,4\} \setminus V^{(1)} = \{1,2,4\}$, jer kako je već rečeno, $V^{(1)}$ sadrži čvor 3 i on ne može biti proširenje. Od preostalih čvorova potrebno je naći one koji su povezani sa čvorom 3, za to se koriste skupovi W_i. W_i sadrži čvorove $j \in \{1,2,3,4\} \setminus V^{(k)}$ za koje važi da su povezani sa čvorom i. Dakle, radi se presek čvorova koji nisu još „uzeti" (R_0) sa onima koji su povezani sa čvorom 3. Kako je samo jedan čvor u $V^{(1)}$ vrši se provera samo jednog preseka. Dobija se $R_1 = \{2,4\}$ i ovo su proširenja koja daju $V^{(2)} = (3,2)$ i $V^{(2)} = (3,4)$ odnosno matrice:

$$\begin{array}{c} \\ 3 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 3 & 2 \end{array} \\ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} \\ 3 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{array}$$

Sada se postupak ponavlja. Za $V^{(2)} = (3,2)$ odrede se $R_0 = \{1,4\}$ i $W_3 = \{2,4\}$. Zatim se, kao i malopre, gleda da li neki od preostalih čvorova ima zajedničku granu sa čvorom 3 tj. $R_1 = R_0 \cap W_2$ jer bi ovo dovelo „jedinicu" u „maksimalan" položaj u matrici, a potom od čvorova iz R_1 da li neko ima zajedničku granu i sa čvorom 2. Konkretno, ovde je $R_1 = \{4\}$ jednočlan, pa nema potrebe računati $R_2 = R_1 \cap W_2$. Analogno se postupa sa $V^{(2)} = (3,4)$. Za $V^{(3)} = (3,2,4)$ odnosno $V^{(3)} = (3,4,2)$ dobijaju se matrice:

$$\begin{array}{c} \\ 3 \\ 2 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 3 & 2 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} \\ 3 \\ 4 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 3 & 4 & 2 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}$$

Kako je ostao samo jedan čvor polaznog grafa kao moguće proširenje, on se može odmah dodati. Dobijaju se monotonični oblici u podstablu generisanom polaznim čvorom 3:

$$\begin{array}{c} \\ 3 \\ 2 \\ 4 \\ 1 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 3 & 2 & 4 & 1 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} \\ 3 \\ 4 \\ 2 \\ 1 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 3 & 4 & 2 & 1 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Nakon što se odrede svi listovi, odnosno sve monotonične matrice izabere se ona koja je maksimalna. Za polaznu matricu iz ovog primera maksimalna forma se dobija za dve permutacije njenih čvorova, a to su:

$$\begin{array}{c|cccc} & 4 & 2 & 3 & 1 \\ \hline 4 & 0 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{array} \qquad \begin{array}{c|cccc} & 4 & 3 & 2 & 1 \\ \hline 4 & 0 & 1 & 1 & 1 \\ 3 & 1 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{array}$$

Glava 3

Generisanje liste povezanih neizomorfnih grafova

Nakon opisa i prikaza algoritma za svođenje grafa na kanonski oblik u drugoj glavi, odnosno sada kada je poznat način utvrđivanja da li je proizvoljan graf u kanonskom obliku ili ne, može se pristupiti problemu generisanja liste povezanih grafova koja ne sadrži izomorfne grafove.

Pre toga, uvodi se još jedan vid reprezentacije grafova. Naime, grafovi se mogu predstaviti pomoću n-torki. Ove n-torke se mogu definisati na više načina. U glavi 2 je već korišćeno to da se graf sa n čvorova može predstaviti jednom n^2-torkom koja se dobija nadovezivanjem vrsta njegove matrice povezanosti. Isto tako, mogle su se nadovezivati, na primer, kolone.

S obzirom da su matrice povezanosti neusmerenih grafova, koji ne sadrže petlje, simetrične i da imaju nule na dijagonali, one su potpuno određene elementima iznad ili ispod dijagonale. Na osnovu toga, neusmereni grafovi se mogu predstaviti torkom[1] sačinjenom, kao i malopre, nadovezivanjem vrsta, ali ovaj put samo od elemenata iznad dijagonale matrice povezanosti.

Pošto su ove torke sačinjene samo od nula i jedinica, te se mogu posmatrati kao binarni zapisi nekih brojeva, može se reći da predstavljaju **kod** (kôd) grafa[2].

Najjednostavniji način da se reši problem generisanja liste povezanih neizomorfnih grafova jeste upotreba algoritma grube sile, čiji je pseudokod dat algoritmom 2.

[1] dužine $n(n-1)/2$

[2] u nastavku, kod grafa je torka dobijena na opisan način

Ako se za svaki broj čvorova n generiše lista L_n svih kodova[3] dužine $n(n-1)/2$, jasno je da su na taj način pokriveni svi grafovi, kao i da među njima ima dosta međusobno izomorfnih grafova. Potom je potrebno proći kroz svaku od tih lista L_n i generisati listu $\mathcal{L}_n$. To podrazumeva proveru da li tekući kod iz L_n predstavlja kod povezanog grafa ili ne; kao i da li je graf sa tekućim kodom izomorfan bilo kom grafu čiji se kod već nalazi u listi $\mathcal{L}_n$, što se proverava poređenjem njihovih kanonskih oblika.

Algoritam 2 Algoritam grube sile za generisanje liste povezanih neizomorfnih grafova

Ulaz Željeni broj čvorova - n
Izlaz Lista kodova grafova sa n čvorova

```
1: function GenerateListOfGraphs(n)
2:     k ← n(n − 1)/2
3:     L_n ← V̄^k_{0,1}                          ▷ V̄^k_{0,1}-varijacije sa ponavljanjem skupa {0, 1} dužine k
4:     L_n ← ∅                                  ▷ Lista L_n je inicijalno prazna
5:     for all c ∈ L_n do
6:         if graf(c) je povezan then           ▷ Npr. DFS obilazak grafa, potom provera da li su svi čvorovi posećeni
7:             if ∀c' ∈ L_n : graf(c) ≇ graf(c') then
8:                 L_n.add(c)
9:     return L_n
```

Jasno je da ovaj algoritam jako brzo postaje neefikasan. Za početak potrebno je generisati sve varijacije sa ponavljanjem skupa $\{0,1\}$ dužine $n(n-1)/2$ odnosno dužine koda, što je $2^{n(n-1)/2}$ elemenata. Potom za svaki element tog skupa izvršiti neku vrstu obilaska grafa kako bi se utvrdila povezanost, što bi bila složenost od $O(n+e)$, gde je $e = |E|$ tj. broj grana. Nakon provere povezanosti ostaje da se proveri da li je tekući element izomorfan nekom od već dodatih u $\mathcal{L}_n$. To ispitivanje izomorfnosti dva grafa podrazumeva njihovo svođenje na kanonske oblike, a kao što je već rečeno, u najgorem slučaju to je generisanje $n!$ monotoničnih formi od kojih bi se odabrala maksimalna.

Mogu se uočiti mesta za unapređenje ovog postupka. Kako je već istaknuto na početku glave 2, i ispitivanje da li su dva grafa izomorfna je težak problem, te bi bilo dobro minimizovati broj parova grafova koje je potrebno uporediti. Trenutno se to čini za svaki kod grafa koji je „kandidat" i za svaki element liste $\mathcal{L}_n$.

[3] sve varijacije sa ponavljanjem skupa $\{0,1\}$ dužine $n(n-1)/2$

Ono što se može primetiti jeste da algoritam grube sile (algoritam 2) ni na koji način ne garantuje da će liste $\mathcal{L}_n$ ($n > 0$) sadržati kanonske kodove[4] tj. kodove grafova koji su u kanonskom obliku kakav je definisan u glavi 2. Trenutno se grafovi samo svode na kanonski oblik radi poređenja. Bilo bi prirodno zahtevati da se u svakoj listi $\mathcal{L}_n$ nađu samo kanonski grafovi, s obzirom da su to predstavnici svojih klasa ekvivalencije.

Na prvi pogled ovo deluje kao dodatni uslov koji samo komplikuje, a ne olakšava, generisanje liste $\mathcal{L}_n$, ali ako se iskoristi i činjenica da je kod kanonskog grafa leksikografski maksimalan u odnosu na kodove drugih grafova iz te klase ekvivalencije, dolazi do nestanka potrebe za poređenjem grafa koji ima tekući kod i svih grafova čiji su kodovi, do tog trenutka, u listi $\mathcal{L}_n$. Naime, svi kodovi se mogu generisati sortirani leksikografski rastuće ili opadajuće. Kako je kompletan graf[5] već u kanonskom obliku te svakako pripada listi $\mathcal{L}_n$ i ima najveći kod od svih grafova sa n čvorova, ima smisla opredeliti se za opadajući poredak. U tom slučaju se poslednja provera u algoritmu 2 (linija 7), može zameniti samo proverom da li tekući kod jeste ili nije kod nekog kanonskog grafa.

Dakle, nakon što se utvrdi da je tekući kod kanonski, nema potrebe prolaziti kroz listu $\mathcal{L}_n$, jer su svi kanonski kodovi u njoj veći od tekućeg, te se on može sa sigurnošću dodati u $\mathcal{L}_n$.

Još jedan način da se smanji obim posla prilikom generisanja liste $\mathcal{L}_n$, jeste da se ne generišu sve varijacije sa ponavljanjem skupa $\{0, 1\}$ tj. da se ne generišu svi kodovi. Kako su od interesa samo povezani grafovi, broj grana, odnosno jedinica, mora biti bar $n-1$. To što graf ima bar $n-1$ grana ne garantuje da je graf povezan, te provera da li je povezan ostaje. Pseudokod ovako blago modifikovanog algoritma je dat algoritmom 3.

U nastavku je prikazan jedan efikasniji način za rešavanje ovog problema koji bolje iskorišćava prethodno navedena zapažanja. Pre toga je dat mali uvod.

Treba istaći da postoje mnogi efikasni algoritmi koji se mogu iskoristiti za rešavanje problema generisanja kataloga grafova ili drugih objekata. Uopšteno govoreći, metodi za efikasno generisanje liste neizomorfnih kombinatornih struktura se mogu podeliti u tri tipa, mada podela nije stroga.

Najčešći tip se zasniva na postojanju kanonskog objekta u svakoj klasi izomorfnih objekata i upravo se on generiše. Ovaj metod se obično naziva *uređeno ili sistematsko*

[4] radi uprošćavanja teksta, neka je kanonski kod kod grafa u kanonskom obliku

[5] između svaka dva čvora postoji grana

Algoritam 3 Blago unapređen algoritam grube sile

Ulaz Željeni broj čvorova - n
Izlaz Lista kodova grafova sa n čvorova

```
1: function GenerateListOfGraphs(n)
2:     k ← n(n − 1)/2
3:     L_n ← sorted({x ∈ V̄^k_{0,1} | sum(x) ≥ n − 1})   ▷ sortiranje u opadajućem poretku
4:     L_n ← ∅                                          ▷ Lista L_n je inicijalno prazna
5:     for all c ∈ L_n do
6:         if graf(c) je povezan then                   ▷ Npr. DFS obilazak grafa, potom provera da li su svi čvorovi posećeni
7:             if c je kanonski kod then
8:                 L_n.add(c)
9:     return L_n
```

generisanje (*eng. orderly algorithm*), jer podrazumeva postojanje relacije uređenja među objektima. Do ovog vida generisanja listi neizomorfnih objekata došli su, nezavisno, Faradžev [4] (*rus. Игорь Александрович Фараджев*) i Rid [10] (*eng. Ronald Cedric Read*).

Drugi tip metoda je razvio Mekej [8] i može se, površno, opisati kao generisanje pomoću kanonskog postupka, nasuprot generisanju samog kanonskog oblika. Objekti se prave augmentacijom, na određeni način, manjih objekata, pri čemu se jedino objekti koji su dobijeni kanonskom augmentacijom prihvataju.

Poslednji tip su metodi zasnovani na homomorfizmu [5].

Kako se glava 2 odnosila na kanonski oblik grafa, i kako je već nagovešteno da sortirano generisanje kodova donosi benefite, u nastavku se za generisanje liste povezanih neizomorfnih grafova koristi Rid-Faradžev tip algoritma.

3.1 Rid-Faradžev tip algoritma

Kao što je već rečeno, poželjno je zarad efikasnosti izbeći poređenje grafa, koji je kandidat za $\mathcal{L}$, sa svim grafovima koji su već dodati u listu (katalog) $\mathcal{L}$. Rid u svom radu [10] opisuje uopšteni postupak za pravljenje kataloga proizvoljnih kombinatornih objekata. Deo naslova Ridovog rada jeste „*svaki je pobednik*", što se upravo odnosi na to da generisanog kandidata nema potrebe porediti sa svim objektima liste $\mathcal{L}$, jer je napravljen na takav način da sigurno nije izomorfan sa ostalima. Taj po-

stupak je opisan u nastavku, uz povremeni osvrt na konkretan problem generisanja grafova.

Neka je S skup nekih objekata ili, konkretno, grafova. Pretpostavka je da se ti objekti mogu, prema nekom parametru q, klasifikovati. Na primer, ovaj parametar q za grafove može da bude broj čvorova ili broj grana. Dakle, postoji sekvenca podskupova $S_0, S_1, S_2, \ldots$, gde je S_q skup onih objekata čija je vrednost klasifikacionog parametra upravo q. Problem se svodi na to da se za svako q sastavi lista $\mathcal{L}_q$ koja bi sadržala samo po jednog predstavnika svake od klasa ekvivalencije relacije „biti izomorfan".

Za rešavanje ovog problema potrebne su tri stvari. Prvo, potrebno je da se može definisati **kanonski objekat**, koji će biti uzet kao predstavnik svoje klase u listi $\mathcal{L}_q$. U slučaju grafova, već je prikazana jedna definicija kanonskog objekta u glavi 2, ali to svakako nije jedini način da se definiše kanonski objekat. Drugo, potrebno je da postoji **uređenje** „$\prec$" elemenata iz S_q. Opet, za grafove, u glavi 2 je korišćeno leksikografsko uređenje niski dobijenih od matrica povezanosti, ali mogla je to da bude i sama vrednost kodova grafova. Konačno, potrebno je da postoji **metod augmentacije**, čijom se primenom na elemente iz S_q generišu, sortirano u nekom poretku, elementi iz S_{q+1}.

Dakle, ovaj tip algoritama (pseudokod dat algoritmom 4) ima sledeću strukturu:

1. Početi od liste $\mathcal{L}_q$ kanonskih objekata koja je uređena u skladu sa izabranim uređenjem $\prec$. Inicijalno, $\mathcal{L}_{q+1}$ je prazna.

2. Redom se, na svaki objekat liste $\mathcal{L}_q$ primenjuje augmentacija tako da se dobija lista elemenata koji pripadaju S_{q+1}. Tokom generisanja ovih novih objekata, primenjuje se sledeće pravilo:

   Ako je objekat kanonski i ako ne prethodi trenutno poslednjem objektu iz $\mathcal{L}_{q+1}$, dodati ga u $\mathcal{L}_{q+1}$, inače ga odbaciti.

3. Kada se ovo uradi za sve objekte iz $\mathcal{L}_q$ onda je lista $\mathcal{L}_{q+1}$ kompletna.

Algoritam 4 Rid-Faradžev algoritam

```
1: procedure ORDERLY ALGORITHM
2:     L_q ← sorted({c ∈ S_q | c je kanonski})
3:     L_{q+1} ← ∅
4:     for all c̄ ∈ {augment(c) | c ∈ L_q}) do
5:         if c̄ je kanonski and c̄ ≺ L_{q+1}.last() then
6:             L_{q+1}.add(c̄)
```

Opšta struktura ovih algoritama se može formulisati i na drugi način:

1. Svaka lista $\mathcal{L}_q$ sadrži isključivo kanonske objekte, poređane prema nekom uređenju $\prec$.

2. Ako objekat kandidat, koji je nastao primenom augmentacije, može da se doda u $\mathcal{L}_{q+1}$ bez narušavanja prvog svojstva, dodati ga, u suprotnom odbaciti.

Jasno je da postoji mnogo načina da se definiše kanonski oblik ili uređenje među objektima koji se generišu. Isto tako, jasno je da efikasnost zavisi od izbora koji se naprave, a da bi postojao efikasan algoritam „uređenog" generisanja neophodno je da su ispunjeni sledeći uslovi.

Uslov 1. *Svaki kanonski objekat iz S_{q+1} se može dobiti augmentacijom bar jednog kanonskog objekta iz S_q.*

Ovo je očigledno neophodno, jer u suprotnom nikad ne bi mogla da se generiše kompletna lista $\mathcal{L}_{q+1}$ od elemenata liste $\mathcal{L}_q$, koja sadrži samo kanonske objekte. Šta više, očekuje se da više elemenata iz $\mathcal{L}_q$ „proizvede" isti kanonski objekat X.

Neka je $f(X)$ prvi u nizu elemenata $\mathcal{L}_q$ koji nakon primene augmentacione funkcije daje X. Time je definisana funkcija $f : \mathcal{L}_{q+1} \to \mathcal{L}_q$ koja mora da zadovoljava sledeći uslov

Uslov 2. *Funkcija f je monotona tj. ako važi $X, Y \in \mathcal{L}_{q+1}$ i $X \prec Y$, tada je $f(X) \preceq f(Y)$.*

Dokaz. Ako se pretpostavi suprotno, tada postoje elementi liste $\mathcal{L}_{q+1}$ X i Y za koje važi da je $X \prec Y$, ali je $f(Y) \prec f(X)$. Jasno je da ako se X ne doda u $\mathcal{L}_{q+1}$ čim je kreirano augmentacijom elementa $f(X)$, neće moći da bude dodato u kasnijim koracima algoritma. Prateći drugi korak opisanog algoritma, nakon obrade elementa $f(Y)$, Y će sigurno biti dodato u listu $\mathcal{L}_{q+1}$. Ovo znači da kada na red za augmentaciju dođe $f(X)$ i kreira se kandidat X, on neće moći da bude dodat

u $\mathcal{L}_{q+1}$, jer se u listi nalazi Y (pretpostavka je bila da je $X \prec Y$). Dakle, u ovom slučaju postojanje elementa Y u listi $\mathcal{L}_{q+1}$ blokira dodavanje elementa X, te se vidi da ako ovaj uslov nije zadovoljen, algoritam neće raditi korektno. □

Ova dva uslova nisu sasvim dovoljna za efikasan algoritam „uređenog" generisanja, pošto nije razmatran slučaj kada je $f(X) = f(Y)$. Ako su X, Y iz S_{q+1} kreirani po prvi put augmentacijom istog elementa iz $\mathcal{L}_q$ i ako je $X \prec Y$, onda je neophodno obezbediti da se element X kreira pre Y. U suprotnom, Y će se naći u $\mathcal{L}_{q+1}$ i blokiraće dodavanje elementa X kada se on kreira. Ova situacija se može izbeći dodavanjem novog uslova:

Uslov 3. *Primena funkcije augmentacije na elemente iz $\mathcal{L}_q$ generiše elemente skupa S_{q+1} u (izabranom) sortiranom poretku.*

Ovako postavljen uslov se odnosi na kreiranje svih objekata iz S_{q+1}, iako je neophodan samo za one objekte koji se dodaju listi $\mathcal{L}_{q+1}$, preostali se mogu generisati u bilo kom poretku. U praksi je često da odabrana funkcija augmentacije po svojoj prirodi generiše sve objekte na neki sistematičan, uređen, način. Stoga ovo nije uslov koji jako ograničava izbor te funkcije.

Teorema 3.1. *Dovoljan uslov da algoritam „uređenog" generisanja (Rid-Faradžev algoritam, algoritam 4) bude efikasan jeste da definicija kanonskog oblika, relacija uređenja i funkcija augmentacije zadovoljavaju prethodno navedena 3 uslova.*

Dokaz. Uslov 1 obezbeđuje da se svaka kanonska konfiguracija X iz S_{q+1} kreira bar jednom. Uslov 2 obezbeđuje da u trenutku kada se kreira X, po prvi put od $f(X)$, u listi $\mathcal{L}_{q+1}$ ne postoji element Y takav da je $f(Y) \neq f(X)$, koji je sledbenik od X i čije prisustvo u listi $\mathcal{L}_{q+1}$ bi blokiralo dodavanje elementa X. Uslov 3 obezbeđuje isto to, samo u slučaju kada je $f(X) = f(Y)$. □

Jedno opšte rešenje za uređeno generisanje torki može se videti u Ridovom radu [10] i opisano je u nastavku.

Neka je S skup vektora dimenzije n čije su vrednosti koordinata iz skupa $\{0, 1, 2, \ldots, k-1\}$. Neka je S_q skup onih vektora čiji je zbir koordinata q. Dva vektora su izomorfna, ako se permutovanjem koordinata jednog vektora, može dobiti drugi. Tada se može definisati algoritam uređenog generisanja liste $\mathcal{L}_q$, gde lista $\mathcal{L}_q$ sadrži po jednog predstavnika svake klase ekvivalencije nad elementima skupa S_q. I u ovom slučaju se može definisati kod, ovog puta opštije. Do sada je torka brojeva tumačena

kao binarni zapis nekog broja, a u ovom opštem slučaju, koordinate vektora se mogu tumačiti kao zapis nekog broja u k-arnom zapisu. Tako, na primer, za $k = 4$, $n = 6$ i $v = (1, 3, 2, 2, 0, 0)$ kod je

$$code(v) = 132200_4 = 1952_{10}$$

Neka je kanonski vektor klase ekvivalencije, kao i ranije, onaj sa maksimalnim kodom. Za uređenje se uzima opadajući niz kodova, a funkcija augmentacije se definiše na sledeći način. Za vektor $v \in \mathcal{L}_q$ krećući se zdesna naći prvu ne-nula koordinatu. Ako je ona manja od $k - 1$, prvo se formira vektor uvećavanjem te koordinate za 1. Naredne augmentacije se formiraju menjanjem pratećih nula na jedinice, krećući se sleva udesno. Tako se od $v = (1, 3, 2, 2, 0, 0)$ dobijaju $v = (1, 3, 2, 3, 0, 0)$, $v = (1, 3, 2, 2, 1, 0)$, $v = (1, 3, 2, 2, 0, 1)$. Kako je kretanje sleva udesno, jasno je da se novi kodovi generišu u opadajućem poretku, te je uslov 3 ispunjen. Da je ispunjen uslov 1 sledi iz sledeće teoreme.

Teorema 3.2. *Ako je w* kanonski vektor iz S_{q+1} i ako se poslednja ne-nula koordinata vektora w* umanji za 1 dajući vektor f(w*), onda je vektor f(w*) kanonski.*

Dokaz. U listi $\mathcal{L}_q$ postoje vektori čija se jedna koordinata razlikuje za 1 od nekog izomorfa vektora w^*, neka je taj izomorf označen sa w. Ti vektori su svakako kanonski, čim su u listi $\mathcal{L}_q$ i, kao predstavnici svoje klase, izomorfni su sa nekim vektorima koji se dobijaju smanjivanjem neke koordinate vektora w^*. Od ovih vektora neka je v^* onaj koji se javlja prvi u $\mathcal{L}_q$ tj. onaj sa najvećim kodom. Dakle, važi sledeće: $code(v^*) < code(w) \leq code(w^*)$. Ako se uporede v^* i w^* mogu se uočiti dva broja i i j koji predstavljaju najmanji, odnosno najveći indeks α za koji važi $w^*_\alpha > v^*_\alpha$ tj.

$$v^* = (\ldots\ldots v^*_i \ldots v^*_j \ldots\ldots)$$
$$w^* = (\underbrace{\ldots\ldots}_{\text{identično}} w^*_i \ldots w^*_j \underbrace{\ldots\ldots}_{w^*_\alpha \leq v^*_\alpha})$$

Ako je $w^*_i > v^*_i + 1$, neka je onda x vektor koji se dobija od w^* umanjivanjem w^*_i za 1. Ako se sa x^* označi njegov kanonski izomorf (predstavnik), onda je

$$code(v^*) < code(x) \leq code(x^*) \tag{3.1}$$

Kako se x razlikuje samo na jednoj koordinati od w^*, to znači da se x^* razlikuje samo na jednoj koordinati od nekog izomorfa w^*, isto kao i v^*. Ovo je u suprotnosti sa definicijom v^*, jer je $code(x^*) > code(v^*)$, te v^* nema najveći kod.

Neka je onda $w_i^* = v_i^* + 1$. Neka je $j > i$ i, ovaj put, neka je x vektor dobijen od $w*$ umanjivanjem w_j^* za 1. Opet važe relacije 3.1, te se dolazi do iste kontradikcije.

Dakle, mora važiti $j = i$. Ovo znači da se v^* i w^* razlikuju samo na i-oj koordinati, u suprotnom bi suma koordinata vektora v^* premašila q.

Neka je pretpostavka da i-ta koordinata **nije** poslednja ne-nula koordinata vektora w^*. Tada je

$$v^* = (\ldots\ldots \underset{i}{a}, \ldots\ldots b, 0, 0, \ldots 0)$$

$$w^* = (\underbrace{\ldots\ldots}_{\text{identično}} \underset{i}{a+1}, \underbrace{\ldots\ldots}_{\text{identično}} b, 0, 0, \ldots 0),$$

gde je b poslednja ne-nula koordinata. Neka je

$$x = (\ldots, a+1, \ldots, b-1, 0, 0, \ldots, 0).$$

Onda se vektor x razlikuje samo na jednoj koordinati od vektora w^* za 1. Opet sledi kontradikcija na osnovu nejednakosti 3.1.

Može se zaključiti da je v^* vektor dobijen umanjivanjem poslednje ne-nula koordinate vektora w^*, što je upravo vektor $f(w^*)$. Kako je v^* ujedno i kanonski vektor, time je tvrđenje dokazano. □

Iz dokazane teoreme sledi da se augmentacijom kanonskog vektora v^* dobija w^* i da $v^* = f(w^*)$.

Da bi se pokazalo da je uslov 2 zadovoljen mogu se posmatrati dva vektora y i z iz $\mathcal{L}_{q+1}$, gde je $code(y) > code(z)$. Neka je i najmanji broj takav da je $y_i > z_i$, te je prvih $i-1$ koordinata jednako.

Kako y i z imaju isti zbir koordinata $(q+1)$, jasno je da z_i ne može biti poslednja ne-nula koordinata vektora z. Ako y_i nije poslednja ne-nula kordinata od y, onda ona nema uticaj na poredak vektora $u = f(y)$ i $v = f(z)$, koji nastaju menjanjem poslednjih ne-nula koordinata vektora y odnosno z. Dakle, $code(u) > code(v)$, te u ovom slučaju, važi uslov 2.

Ako y_i jeste poslednja ne-nula koordinata od y i ako je $y_i > z_i+1$, onda umanjenje za 1 koordinate y_i ne menja činjenicu da je $y_i > z_i$, te opet važi $code(u) > code(v)$. S druge strane, ako je $y_i = z_i + 1$ onda se, zbog zbira koordinata zna da z sadrži samo jednu „jedinicu" negde desno od z_i. Zbog toga, nakon umanjenja y_i i te „jedinice", važi $u = v$, te uslov 2 i dalje važi.[6]

[6] uslov 2 ne zahteva strogu monotonost

Kako su sva tri uslova zadovoljena, sledi da se na ovaj način mogu generisati sve torke sa željenom sumom elemenata. Samim tim, sve strukture koje se mogu predstaviti pomoću torki, kao što su grafovi preko svojih kodova mogu se generisati na ovaj način.

3.2 Generisanje neizomorfnih stabala

Kako bi se realizovao prethodno opisani postupak, u konkretnom problemu generisanja povezanih grafova, potrebno je odrediti početni skup S_q elemenata čijom augmentacijom se dobijaju elementi skupa S_{q+1}, gde q predstavlja broj grana između određenog broja čvorova. Jasno je da bi se moglo krenuti od $q = 0$ i dodavati grane dok grafovi ne postanu povezani, ali ovo bi predstavljalo suviše nepotrebnog posla.

Bilo bi bolje početi od liste neizomorfnih stabala, jer ona predstavljaju povezane grafove sa najmanjim brojem grana i svako naredno dodavanje grane na stablo ostavlja graf povezanim. Na ovaj način se eliminiše i potreba da se proverava povezanost grafova u toku njihovog generisanja.

Kao što je rečeno, postoje određeni tipovi grafova za koje su poznate efikasne metode generisanja, a stabla su upravo jedna od takvih. U ovom radu je iskorišćen postupak koji je opisao Džo Savada (*eng. Joe Sawada*) u svom radu [11].

Savada prvo opisuje algoritam za generisanje korenskih stabala (*eng. rooted trees*). **Korensko stablo** jeste stablo čiji je jedan čvor izdvojen[7] i zove se **koren** stabla. Potom se taj algoritam prilagođava za generisanje slobodnih[8] stabala, tako što se jedan njihov čvor proglasi za koren kako bi se iskoristio prethodno opisani algoritam.

Čvor v je **centar stabla** ako je njegova maksimalna udaljenost od preostalih čvorova minimalna. Stablo sa jednim centrom se naziva **unicentralno**; dok je stablo sa dva centra **bicentralno**. Ako stablo ima dva centra, onda ti čvorovi moraju biti susedni.

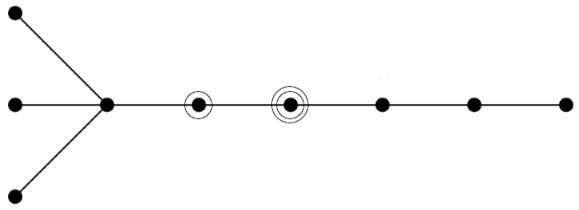
Veličina stabla (ili podstabla) jeste broj čvorova koje stablo sadrži. Čvor v je **centroid stabla** ako je njegovo najveće podstablo, koje ostaje kada se v ukloni, minimalno. Stablo može imati jedan centroid ili dva centroida. Ako ima jednog onda je **unicentroidno**; dok ako ima dva, naziva se **bicentroidno**. Čvor v je jedinstven centroid ako i samo ako je veličina najvećeg podstabla manja ili jednaka $\lfloor (n-1)/2 \rfloor$.

[7] obično se prikazuje kao vrh stabla

[8] stabla bez istaknutog čvora tj. korena

Ako je stablo bicentroidno, onda su njegovi centroidi susedni čvorovi, te uklanjanje grane između njih daje dva podstabla jednake veličine. Očigledno, ako je n neparno, onda ne postoje bicentroidna stabla.

Na slici 3.1 je prikazano stablo kod koga se razlikuju centroid (zaokružen) i centar (dvostruko zaokružen). Vidi se da uklanjanjem centroida ostaju dva stabla sa po četiri čvora, dok bi se uklanjanjem centra dobilo jedno stablo sa pet i jedno sa tri čvora. Sa druge strane, maksimalna udaljenost centra od preostalih čvorova je tri, dok za centroid to ne važi, njegova maksimalna udaljenost je četiri.



Slika 3.1: Centroid i centar stabla

Prilikom izbora čvora slobodnog stabla koji će biti koren dva prirodna kandidata su, upravo, centar stabla i centroid stabla. Ispostavlja se da korišćenje centroida, nasuprot centra, za „ukorenjavanje" stabala olakšava i omogućava njihovo efikasno generisanje. Ovo se postiže kroz dva rekurzivna algoritma, jedan za generisanje unicentroidnih stabala (algoritam 5) i jedan za generisanje bicentroidnih stabala (algoritam 6).

Algoritam 5 Algoritam za generisanje unicentroidnih stabala sa n čvorova

Ulaz Željeni broj čvorova - n i niz a dužine n
Izlaz Ispis unicentroidnih stabala

```
1: procedure UNICENTROID(t = 2, p = 1, s = 1)
2:     j, max
3:     if t = n then
4:         if p | (n − 1) then
5:             ispiši a
6:     else
7:         if s = ⌊(n − 1)/2⌋ then
8:             max ← 1
9:         else
10:            max ← a_{t−1} + 1
11:        for j ∈ {a_{t−p}, ..., max − 1, max} do
12:            a_t ← j
13:            if j = a_{t−p} and a_{t−p} = 1 then
14:                Unicentroid(t + 1, p, 1)
15:            else if j = a_{t−p} then
16:                Unicentroid(t + 1, p, s + 1)
17:            else
18:                Unicentroid(t + 1, t, s + 1)
```

Za generisanje satabala sa proizvoljnim brojem čvorova n potrebno je pokrenuti algoritam 5 za svako n, dok se, kako je već istaknuto, bicentroidna stabla mogu javiti jedino kada je n parno te se algoritam 6 pokreće samo u tom slučaju. Dakle, za parno n pokreću se oba algoritma kako bi se generisala sva stabla.

Algoritam 6 Algoritam za generisanje bicentroidnih stabala sa n čvorova

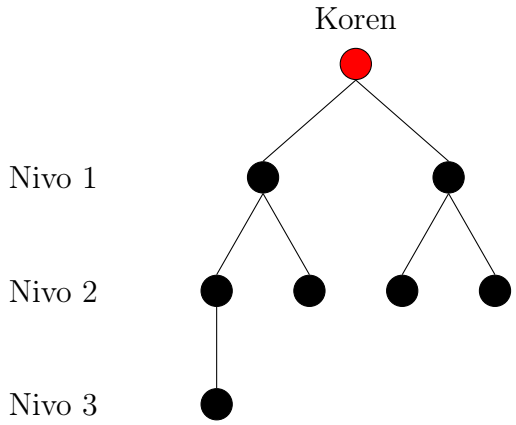
Ulaz Željeni broj čvorova - n i niz a dužine n
Izlaz Ispis bicentroidnih stabala

```
1:  procedure BICENTROID(t = 1, istiPrefiks = False)
2:      j, min
3:      if t = n then
4:          ispiši a
5:      else if t = n/2 then
6:          a_t ← 1
7:          Bicentroid(t + 1, True)
8:      else if istiPrefiks = False then
9:          if t > n/2 then
10:             min ← 2
11:         else
12:             min ← 1
13:         for j ∈ {min, min + 1, ..., a_{t−1} + 1} do
14:             a_t ← j
15:             Bicentroid(t + 1, False)
16:     else
17:         for j ∈ {a_{t−n/2} + 1, ..., a_{t−1}, a_{t−1} + 1} do
18:             a_t ← j
19:             if j = a_{t−n/2} + 1 then
20:                 Bicentroid(t + 1, True)
21:             else
22:                 Bicentroid(t + 1, False)
```

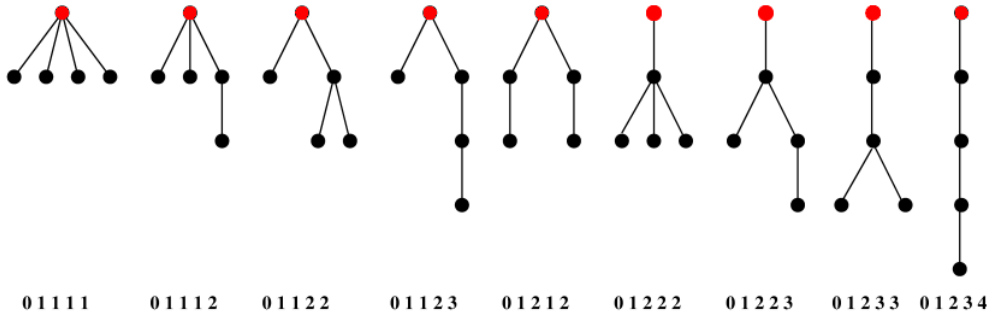
Treba istaći da reprezentacija stabla koja se koristi u Savadinom radu, nije ista kao reprezentacija koja je korišćena u ovom. Naime, zbog prirode korenskih stabala (videti sliku 3.2) moguće ih je reprezentovati nizom brojeva koji predstavljaju udaljenost svakog čvora od korena stabla, odnosno za svaki čvor se beleži na kom je „nivou" stabla. Shodno tome ova repezentacija se na engleskom naziva *level sequence*. Kako ona odgovara prefiksnom (KLD) obilasku stabla i zapisivanju udaljenosti od korena za svaki posećeni čvor, biće označena kao **KLD reprezentacija**. U konkretnom primeru sa slike 3.2, KLD reprezentacija je 01232122.

Na slici 3.3 se mogu videti korenska stabla sa pet čvorova i njihove KLD reprezentacije.

Dakle, kako bi se iskoristio ovaj algoritam za generisanje neizomorfnih stabala kao polazna tačka za generisanje svih povezanih grafova, potrebno je dobijena stabla iz KLD reprezentacije prevesti u, prethodno definisanu, kod reprezentaciju. Na sreću,



Slika 3.2: Primer korenskog stabla



Slika 3.3: Sva korenska stabla sa 5 čvorova

KLD reprezentacija se jednostavno prevodi u matricu povezanosti stabla, odakle nije teško pročitati kod.

Algoritam za prevođenje KLD reprezentacije u matricu povezanosti je dat algoritmom 7. Za realizaciju je korišćen **stek** sa svojim osnovnim operacijama: *push* i *pop*, kao i operacijom *peek*. **Push** stavlja dati element na vrh steka, ostavljajući prethodne ispod. **Pop** uklanja i vraća element koji je trenutno na vrhu, dok **peek** samo vraća element sa vrha bez uklanjanja.

Algoritam 7 Algoritam za prevođenje KLD repreznetacije stabla u matricu povezanosti

Ulaz Niz dužine n koji predstavlja KLD reprezentaciju stabla
Izlaz Matrica poveznosti reda $n \times n$

```
1: function ConvertLevelSequenceToMatrix(stablo)
2:     matrica ← [0]_{n×n}
3:     stek ← []
4:     for i ∈ {1, ..., n} do
5:         i_nivo ← i
6:         if stek nije prazan then
7:             j ← stek.peek()
8:             j_nivo ← stablo_j
9:             while j_nivo ≥ i_nivo do
10:                stek.pop()
11:                j ← stek.peek()
12:                j_nivo ← stablo_j
13:            matrica_{i,j} = matrica_{j,i} = 1
14:        stek.push(i)
15:    return matrica
```

Na kraju, u tabeli 3.1 je prikazan broj neizomorfnih stabala do 20 čvorova zajedno sa vremenom koje je potrebno da *Python* skripta generiše ta stabla. [9]

Tabela 3.1: Broj stabala čiji je broj čvorova $n \leq 20$

n	**Broj stabala sa n čvorova**	**Vreme generisanja (ms)**	**n**	**Broj stabala sa n čvorova**	**Vreme generisanja (ms)**
1	1	0.003	11	854	2.309
2	1	0.007	12	2694	4.301
3	1	0.010	13	8714	12.267
4	2	0.017	14	28640	34.192
5	3	0.028	15	95640	130.783
6	6	0.057	16	323396	656.229
7	14	0.109	17	1105335	2164.897
8	34	0.266	18	3813798	6489.727
9	95	0.485	19	13269146	22674.696
10	280	1.629	20	46509358	81271.377

[9] generisanje izvršeno na računaru koji poseduje procesor Intel® Core™ i5-10300H CPU @ 2.50GHz × 8 sa 16GB RAM memorije pod operativnim sistemom Ubuntu 20.04.4 LTS

3.3 Algoritam za generisanje grafova zasnovan na stablima

Sada kada je, na osnovu prethodnog odeljka, poznato kako generisati neizomorfna stabla sa nekim brojem čvorova, ostaje pitanje da li se i na koji način to može iskoristiti, zajedno sa algoritmom 4, da se dobije efikasniji metod za kreiranje kataloga povezanih grafova. Kako su stabla najmanji, po broju grana, povezani grafovi sa određenim brojem čvorova, uzeta su za polaznu tačku ovog ispitivanja.

Kako je već istaknuto, opšta struktura Rid-Faradžev algoritama može se opisati pomoću sledeća dva iskaza:

1. Svaka lista $\mathcal{L}_q$ sadrži isključivo kanonske objekte, poređane prema nekom uređenju $\prec$.

2. Ako objekat kandidat, koji je nastao primenom augmentacije, može da se doda u $\mathcal{L}_{q+1}$ bez narušavanja prvog svojstva, dodati ga, u suprotnom odbaciti.

Dakle, potrebno je definisati kanonsku reprezentaciju, neko uređenje i metod za augmentaciju članova liste $\mathcal{L}_q$. Kanonska reprezentacija grafa je prikazana u glavi 2 - to je maksimalna vrednost koda dobijenog od elemenata iznad dijagonale matrice povezanosti grafa nadovezivanjem njenih vrsta. Odabrano uređenje je leksikografsko opadajuće po kodovima. Osnovni problem jeste definisati postupak augmentacije.

Inicijalna zamisao je bila da se prilagodi prethodno opisani opšti postupak augmentacije elemenata iz S_q u S_{q+1} koji je dao Rid [10] (videti 3.1), zato što kod jeste jedna torka. Augmentacija bi se vršila sistematskim menjanjem 0 u 1 iza poslednje ne-nula vrednosti u kodu i potom bi se dalje pratio opisani postupak. Ispostavlja se da ovo nije moguće. Naime, kroz poređenje rezultata takvog postupka i rezultata algoritma „grube sile”[10], dolazi se do zaključka da takav način augmentacije za generisanje povezanih grafova ne zadovoljava uslov 1 za postojanje „uređenog” algoritma. Ovo se može videti na konkretnom primeru datom u tabeli 3.2.

Analizom kodova iz desne kolone, jasno je da se svi oni ne mogu generisati samo jednim menjanjem nule u jedinicu iza poslednje ne-nula vrednosti. Čak i ideja da se ne menjaju samo nule iza poslednje ne-nula vrednosti tj. poslednje jedinice, već redom, sve nule ne donosi željeni rezultat.

[10]koji sigurno generiše sve grafove

Tabela 3.2: Tabela kodova kanonskih stabala sa 6 čvorova i kanonskih grafova sa 6 čvorova i 6 grana

Kanonska stabla sa 6 čvorova	**Kanonski grafovi sa 6 čvorova i 6 grana**
11111 0000 000 00 0	11111 1000 000 00 0
11110 0001 000 00 0	11110 1001 000 00 0
11100 0011 000 00 0	11110 1000 000 01 0
11100 0010 001 00 0	11110 0001 001 00 0
11100 0001 000 00 1	11100 1010 001 00 0
11000 0100 010 01 0	11100 1010 000 01 0
	11100 1000 000 11 0
	11100 1000 000 10 1
	11100 0011 010 00 0
	11100 0010 010 01 0
	11100 0010 010 00 1
	11100 0010 001 00 1
	11000 0100 010 01 1

Kako bi se obezbedilo generisanje svih kanonskih grafova, mora se izabrati neki drugi način. Pošto je lista $\mathcal{L}_q$ lista svih kanonskih grafova, odnosno predstavnika svojih klasa ekvivalencije, jasno je da dodavanje grane ovim grafovima tj. promenom jedne nule u jedinicu na sve moguće načine generiše listu elemenata iz S_{q+1} u kojoj je bar jedan, ne nužno u kanonskom obliku, predstavnik svake klase ekvivalencije. Svođenjem ovih elemenata na kanonske oblike i otklanjanjem duplikata, sigurno se dobija lista $\mathcal{L}_{q+1}$. Pseudokod ovog algoritma je dat algoritmom 8. Ovaj algoritam, iako sličan algoritmu grube sile (algoritam 2), je u stanju da generiše grafove sa većim brojem čvorova. Očigledno to što rad algoritma polazi od stabala ima značajan uticaj. Vremena generisanja grafova do 7 čvorova su data u tabeli 3.3.

Tabela 3.3: Rezultati algoritma 8

n	**Broj grafova sa n čvorova**	**Vreme generisanja (ms)**
1	1	0.004
2	1	0.004
3	2	0.006
4	6	0.638
5	21	11.122
6	112	192.737
7	853	4716.093
8	11117	167233.579

Algoritam 8 Algoritam za generisanje kataloga povezanih grafova sa stablima

Ulaz Željeni broj čvorova - n
Izlaz Lista kodova grafova sa n čvorova

```
1:  function GenerateListOfGraphs(n)
2:      L_n, L_q ← stabla sa n čvorova          ▷ Skup svih kodova grafova i tekući „radni" skup.
3:      while L_q ≠ {K_n} do                    ▷ K_n je kompletan graf
4:          L_{q+1} ← ∅
5:          for all c ∈ L_q do
6:              for all c̄ ∈ {augment(c) | c ∈ L_q}) do
7:                  c̃ ← ReduceToCanonicalForm(c̄)
8:                  L_n.add(c̃)
9:                  L_{q+1}.add(c̃)
10:         L_q ← L_{q+1}
11:     return L_n
```

Jedna modifikacija algoritma 8 tako da se koristi ideja „uređenog" generisanja, nije se pokazala uspešnom. Naime, čak i da su članovi liste $\mathcal{L}_q$ uređeni, to ne garantuje da će njihova augmentacija, a potom svođenje na kanonski oblik dati uređenu listu. Utvrđeno je, kroz implementaciju te ideje, da dolazi do blokiranja pokušaja dodavanja nekih kanonskih grafova zato što bi narušili uređenost liste $\mathcal{L}_{q+1}$. Ostaje pitanje da li je moguće naći pogodan način augmentacije koji bi za ovako odabrani kanonski oblik davao korektne rezultate.

Kako bi se pokazala efikasnost Rid-Faradžev algoritama, implementiran je postupak koji je prethodno opisan u odeljku 4. Dakle, kako se graf može predstaviti kao torka, mogu se generisati svi grafovi menjanjem nula nakon poslednje jedinice u kodu. Ovaj postupak generiše sve grafove za zadati broj čvorova, te je potrebno na kraju odstraniti one grafove koji nisu povezani. Pseudokod ovog algoritma je dat algoritmom 9.

Algoritam 9 Rid-Faradžev algoritam za generisanje kataloga povezanih grafova

Ulaz Željeni broj čvorova - n
Izlaz Lista kodova grafova sa n čvorova

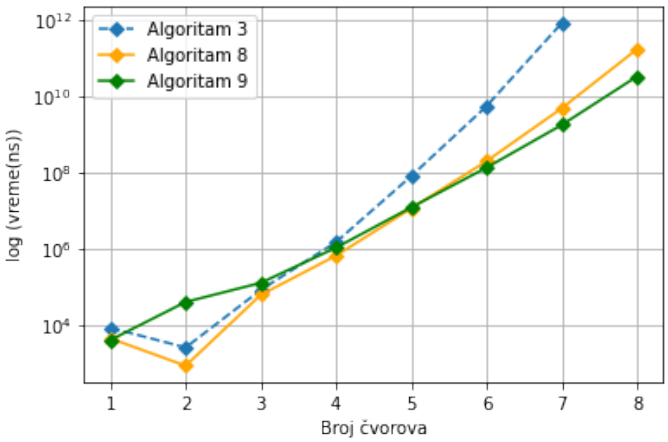
1: **function** GenerateListOfGraphs(n)
2: $\mathcal{L}_n \leftarrow \emptyset$
3: $l \leftarrow \lfloor n(n-1)/2 \rfloor$
4: $\mathcal{L}_q \leftarrow [\underbrace{0, 0 \ldots 0,}_{l}]$ ▷ Tekuća „radna" lista.
5: **while** $\mathcal{L}_q \neq \{K_n\}$ **do** ▷ K_n je kompletan graf
6: $\mathcal{L}_{q+1} \leftarrow \emptyset$
7: **for all** $c \in \mathcal{L}_q$ **do**
8: **for all** $\bar{c} \in \{\textbf{augment}(c) \mid c \in \mathcal{L}_q\})$ **do** ▷ Augmentacija menja samo nule iza poslednje jedinice.
9: **if isCanonical**($\bar{c}$) **then**
10: $\mathcal{L}_n$.**add**($\bar{c}$)
11: $\mathcal{L}_{q+1}$.**add**($\bar{c}$)
12: $\mathcal{L}_q \leftarrow \mathcal{L}_{q+1}$
13: **return** $\{g \in \mathcal{L}_n \mid g$ *je povezan*$\}$

Rezultati ovog algoritma su dati u tabeli 3.4. Može se videti značajno poboljšanje vremena izvršavanja kako raste broj čvorova, uprkos naknadnom filtriranju povezanih grafova.

Tabela 3.4: Rezultati algoritma 9

n	**Broj grafova sa n čvorova**	**Vreme generisanja (ms)**
1	1	0.004
2	1	0.039
3	2	0.123
4	6	1.035
5	21	11.832
6	112	130.414
7	853	1720.548
8	11117	32319.290

Na slici 3.4 je prikazan grafik sa vremenima izvršavanja modifikovanog algoritma grube sile (algoritam 3), algoritma sa stablima (algoritam 8) i Rid-Faradžev algoritma za generisanje grafova uz filtriranje (algoritam 9).



Slika 3.4: Poređenje vremena izvršavanja

Algoritam 9, uprkos potrebi da filtrira listu kanonskih grafova kako bi se izbacili oni koji su nepovezani, daje najbolje rezultate. Algoritmu 8, nakon što udeo broja stabala u ukupnom broju povezanih grafova počne drastično da opada, povećava se vreme izvršavanja, dok je algoritam 3, očekivano, dao najslabije rezultate. Čak je algoritam 3 bio prekinut zbog dužine izvršavanja pre nego što bi došao do rezultata za grafove sa 8 čvorova.

Glava 4

Zaključak

U radu je prikazan jedan algoritam za svođenje grafova na njihov kanonski oblik. Kanonski oblik grafa jeste jedan istaknuti graf koji je izabran da predstvalja sve međusobno izomorfne grafove. Svakako, opisni način kanonizacije grafova je podložan promenama, kao i unapređenju uz neki vid algoritma grananja uz odsecanje. Jedan vid unapređenja, baziran na teoriji grupa, jeste detektovanje automorfizama među skupovima čvorova stabla koje se generiše prilikom izračunavanja. Na taj način se može ranije odustati od generisanja pojedinih listova.

Dalje, rad prikazuje jedan okvir za sistematsko generisanje grafova i drugih struktura koje se mogu nabrojati. Razmotrena je primena tog okvira na generisanje povezanih stabala, polazeći od svih neizomorfnih stabala. Zaključeno je da opisani način implementacije ne daje ispravne rezultate, te je potrebno pronaći drugačiji metod augmentacije polaznih stabala, a potencijalno i drugačiju definicijz kanonskog oblika.

Uprkos tome, implementirani su i algoritmi koji uspešno generišu katalog neizomorfnih povezanih grafova.

Bibliografija

[1] Vladimir L. Arlazarov, Ivan I. Zuev, Anatoly V. Uskov, and Igor A. Faradzev. An algorithm for the reduction of finite non-oriented graphs to canonical form. pages 195–201, December 1974.

[2] László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015.

[3] Paolo Codenotti, Hadi Katebi, Karem A. Sakallah, and Igor L. Markov. Conflict Analysis and Branching Heuristics in the Search for Graph Automorphisms. In *25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 907–914, Washington, DC, November 2013.

[4] Igor A. Faradzev. Constructive enumeration of combinatorial objects. 1978.

[5] Thomas Gruner, Reinhard Laue, and Markus Meringer. Algorithms for group actions: homomorphism principle and orderly generation applied to graphs. January 1997.

[6] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In David Applegate, Gerth Stølting Brodal, Daniel Panario, and Robert Sedgewick, editors, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007.

[7] José Luis López-Presa, Antonio Fernández Anta, and Luis Núñez Chiroque. Conauto-2.0: Fast isomorphism testing and automorphism group computation. *CoRR*, abs/1108.1060, 2011.

[8] Brendan D. McKay. Isomorph-free exhaustive generation. In *J. Algorithms*, volume 26, page 306–324, USA, February 1998. Academic Press, Inc.

[9] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *CoRR*, abs/1301.1493, 2013.

[10] Ronald C. Read. Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations*. In B. Alspach, P. Hell, and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 107–120. Elsevier, 1978.

[11] Joe Sawada. Generating rooted and free plane trees. In *ACM Transactions on Algorithms*, volume 2, pages 1–13, January 2006.

Biografija autora

Pavle Gavrilović je rođen 1. februara 1994. u Čačku. Osnovno obrazovanje stekao je u Osnovnoj školi *Vuk Karadžić* u Čačku. Nakon čega je upisao i završio prirodno-matematički smer čačanske *Gimnazije*.

Matematički fakultet u Beogradu, studijski modul *Računarstvo i informatika* upisuje 2013. godine. Osnovne studije završava 2018. godine sa prosečnom ocenom 8.57. Potom nastavlja studiranje na master studijama na istom fakultetu i istom studijskom modulu.