Neki algoritmi za određivanje ciklusa u grafu

Biloš, Hrvoje

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1303

NEKI ALGORITMI ZA ODREĐIVANJE CIKLUSA U GRAFU

Hrvoje Biloš

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1303

NEKI ALGORITMI ZA ODREĐIVANJE CIKLUSA U GRAFU

Hrvoje Biloš

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1303

Pristupnik:	**Hrvoje Biloš (0036538635)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Mario Krnić

Zadatak: **Neki algoritmi za određivanje ciklusa u grafu**

Opis zadatka:

U ovom radu student treba osmisliti i implementirati neke algoritme za određivanje ciklusa u zadanom grafu. Nakon uvodnog teorijskog dijela, potrebno je napraviti odgovarajuće algoritme koji daju odgovor na pitanje sadrži li zadani neorijentirani, odnosno orijentirani graf, ciklus. Osim toga, potrebno je implementirati algoritam koji detektira postojanje negativnog ciklusa u orijentiranom težinskom grafu. Konačno, student treba implementirati algoritam koji broji cikluse zadane duljine u danom grafu.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

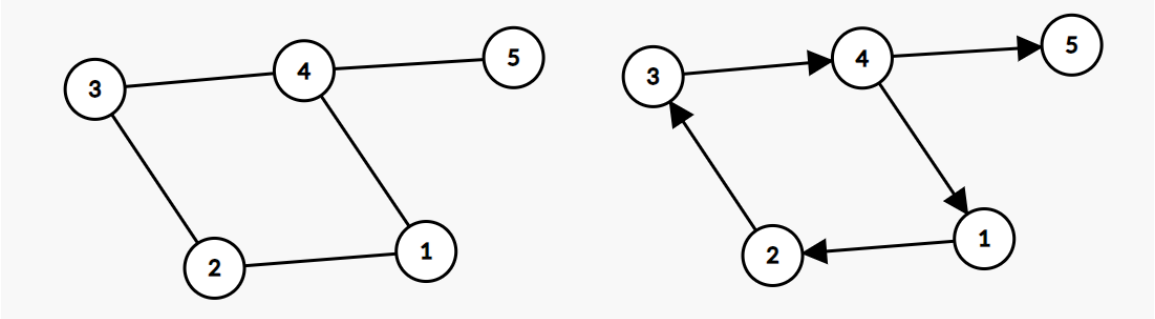
U ovom završnom radu objasnit ćemo što su grafovi, preciznije rečeno, što su jednostavni grafovi. Definirat ćemo usmjerene i neusmjerene grafove te pojam ciklusa u tim grafovima. Također ćemo opisati način zapisivanja grafova s pomoću matrica, koje su prikladne za programski pristup.

Kao glavna tema ovog rada, prikazat ćemo izvorni kod programa za algoritme koji pronalaze i ispisuju sve jedinstvene jednostavne cikluse u proizvoljnom jednostavnom grafu te detaljnije objasniti kako implementirani algoritmi pronalaze cikluse. Navedeni program napisan je u programskom jeziku Java.

1. Kratki uvod u teoriju grafova

1.1. Jednostavni graf

Jednostavni graf G sastoji se od nepraznog konačnog skupa V(G), čije elemente zovemo **vrhovi** (čvorovi) grafa G i konačnog skupa E(G) različitih dvočlanih podskupova skupa V(G) koje zovemo **bridovi**. Skup V(G) zovemo skup vrhova i ako je jasno o kojem je grafu G riječ označavat ćemo ga kraće samo s V, a skup E(G) zovemo skup bridova i označavat ćemo ga i samo s E. Formalno, ponekad ćemo pisati G = (V(G),E(G)) ili kraće još i G = (V,E) [1].



Slika 1.1 Jednostavni graf i jednostavni usmjereni graf [2].

Primjer jednostavnog grafa s 5 vrhova možemo vidjeti lijevo na slici (Slika 1.1). Naravno, postoje i složeni grafovi koji mogu imati više od jednog brida između dva ista vrha ili brid koji spaja vrh sam sa sobom. U svrhu ovog rada njima se nećemo baviti.

Osim neusmjerenih, postoje i usmjereni grafovi kojima ćemo se ovdje baviti. Primjer usmjerenog grafa možemo vidjeti desno na slici (Slika 1.1). Glavna razlika između te dvije vrste grafova jest da usmjereni graf ima strogo definiran smjer prijelaza između vrhova. Bridovi neusmjerenog grafa su poput dvosmjernih cesta, dok su lukovi usmjerenog grafa poput jednosmjernih cesta.

Važno je definirati pojam povezanosti. Graf je povezan ako postoji put između bilo koja dva vrha. Ako ne postoji, graf je nepovezan. Bavimo se samo povezanim grafovima jer, ako graf nije povezan, ekvivalentno bi bilo kao da ih tretiramo kao dva ili više različitih grafova i provodimo algoritam na svakom zasebno.

1.2. Matrica susjedstva

Nailazimo na problem kako da graf formatiramo u oblik koji je prikladan za primjenu pri programiranju. Na našu sreću, postoji takav oblik, a to je matrica susjedstva.

Označimo li vrhove zadanog grafa G s V = {1,2,...,n}, onda definiramo **matricu susjedstva** A = [ai j] kao n×n matricu čiji je element $a_{i\,j}$ jednak broju bridova koji spajaju vrh i s vrhom j. [1]. Programski možemo prikazati kao dvodimenzionalnu matricu preko koje iteriramo da izvučemo bilo koju potrebnu informaciju o zadanom grafu.

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Slika 1.2 matrice susjedstva.

Na slici iznad (Slika 1.2) prikazane su matrice susjedstva za neusmjereni i usmjereni graf prikazan na slici 1.1 (Slika 1.1). Možemo primijetiti nekoliko karakteristika matrica susjedstva jednostavnih grafova, kao što je činjenica da su svi elementi na dijagonali matrice nule. Matrice susjedstva neusmjerenih grafova su simetrične, što može smanjiti broj pretraživanja u programskoj implementaciji jer sve prijelaze možemo pronaći gledajući samo jednu polovicu matrice iznad ili ispod glavne dijagonale.

U slučaju kada radimo s težinskim grafovima, tj. grafovima čiji prijelazi imaju određenu težinu (bilo pozitivnu ili negativnu), vrijednosti brojeva u matrici susjedstva trebamo zamijeniti odgovarajućim težinama umjesto jedinicama.

1.3. Ciklus

Prije definiranja ciklusa, moramo definirati pojmove šetnje i puta jer su blisko povezani s pojmom ciklusa.

Neka je G graf. **Šetnja** u G je konačan slijed bridova oblika v_0v_1, v_1v_2, ... , $v_{m-1}v_m$, također često u oznaci $v_0 \to v_1 \to \cdots \to v_m$, u kojem su svaka dva uzastopna brida ili susjedna ili jednaka [1].

Na primjeru sa slike 1.1 (Slika 1.1) jedna moguća šetnja bi bila $5 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 1$. Kao što vidimo, pojam šetnje je preopćenit i ne daje nikakvu korisnu informaciju. Zato ćemo se fokusirati na karakteristične šetnje koje zadovoljavaju određene uvjete, kao što su staza i put.

Šetnju u kojoj su svi bridovi različiti zovemo **staza**. Ako su, uz to, i svi vrhovi v_0, v_1,..., v_m različiti (osim eventualno početni vrh v_0 i krajnji vrh v_m), onda takvu stazu zovemo **put**. Za stazu ili put kažemo da su **zatvoreni** ako je $v_0 = v_m$ [1].

Sada možemo definirati ciklus. Ciklus je zatvoreni put koji sadrži barem jedan brid. Primjer jednog ciklusa s grafa na slici 1.1 (Slika 1.1) bio bi $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$. Možda ste uočili da postoji više permutacija istog ciklusa; u prethodnom primjeru, valjan ciklus bi također bio $3 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 3$. Zato je važno napomenuti da tražimo jedinstvene cikluse, a njegove permutacije smatramo istim ciklusom.

1.3.1. Problem trgovačkog putnika

Jedan od poznatijih problema koji se bavi pronalaženjem ciklusa je problem trgovačkog putnika. Ovaj problem možemo susresti u našem svakodnevnom životu, naravno u drugačijem obliku. Problem glasi: trgovački putnik treba obići gradove i vratiti se kući, pri čemu želi prijeći što manji put tijekom svog putovanja. Ovaj problem je lako modelirati pomoću težinskog grafa. Svaki vrh predstavlja jedan grad, dok brid između vrhova predstavlja postojanje puta između ta dva grada i njegovu udaljenost. Problem se tada svodi na pronalaženje najjeftinijeg ciklusa u tom grafu, pri čemu su posjećeni svi vrhovi. Ciklus koji prolazi kroz sve vrhove grafa poznat je kao Hamiltonov ciklus.

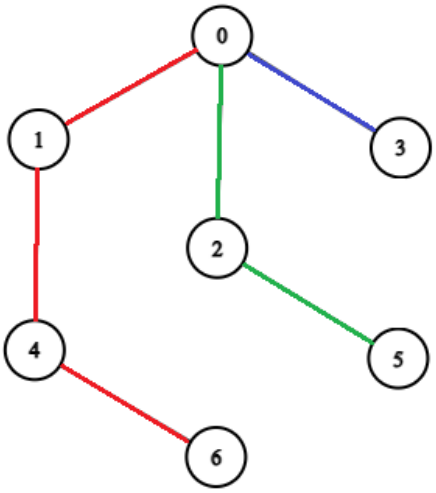
2. Algoritam za traženje ciklusa u grafu

U ovom poglavlju objasnit ćemo dva algoritma koja pronalaze sve cikluse u zadanom grafu. Prvi algoritam je depth first search (DFS), putem kojeg možemo pronaći sve cikluse u usmjerenom i neusmjerenom grafu. Drugi algoritam, koji pronalazi takozvane osnovne cikluse i njihovim kombinacijama dobiva sve ostale cikluse, može se primijeniti samo na neusmjerene grafove.

2.1. Depth-first search

U računalnoj znanosti, algoritmi pretrage imaju ključnu ulogu u pronalaženju putanja i čvorova unutar grafova, koji su temeljna struktura za predstavljanje mnogih problema u različitim domenama, od mrežnih puteva do hijerarhijskih organizacija podataka. Jedan od najosnovnijih i najčešće korištenih algoritama pretrage je algoritam pretrage u dubinu (engl. Depth First Search ili DFS) [3].

Algoritam pretrage u dubinu istražuje graf započinjući od početnog čvora, idući što je moguće dublje u svakom grananju prije nego što se vrati. Ovaj pristup omogućava detaljno istraživanje strukture grafa i pronalaženje svih mogućih putanja do određenog čvora ili do određenih uvjeta unutar grafa. DFS se implementira pomoću s stoga, bilo implicitno korištenjem rekurzije ili eksplicitno korištenjem strukture podataka stoga.



Slika 2.1 DFS nad grafom [2].

Na slici 2.1 (Slika 2.1) prikazan je jedan graf, preciznije jedno stablo. Ako uzmemo vrh 0 kao korijenski vrh i provedemo DFS algoritam nad ovim stablom, dajući prioritet vrhu s manjim indeksom (uzimajući vrh 1 prije vrha 2), u boji je prikazan slijed pretrage algoritma. Algoritam će prvo proći crvenim putem, vratiti se u korijenski vrh, proći zelenim putem, vratiti se u korijenski vrh i na kraju proći plavim putem.

Kako bismo iskoristili ovaj algoritam za traženje ciklusa, koristit ćemo strukturu podataka stog kako bismo pamtili koje smo vrhove posjetili u trenutačnoj stazi. Pri ispitivanju vrha kojeg želimo posjetiti prvo provjerimo je li vrh na dnu stoga (korijenski vrh); ako jest, onda smo pronašli ciklus, i to će nam služiti kao uvjet u ovom algoritmu pretraživanja.

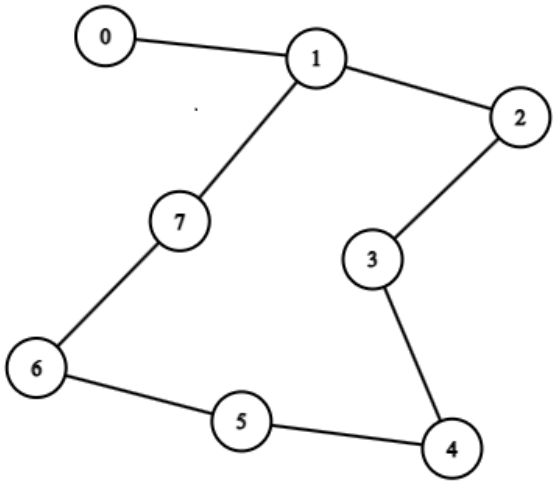
Sami tok algoritma:

Za svaki vrh u grafu provedemo depth first search algoritam. Na stog dodamo vrh i tražimo njegove susjede. Uzmemo prvog susjeda kojeg pronađemo i provjerimo uvjete: je li taj vrh već dio staze (unutar stoga) ili je li taj vrh početni vrh. U slučaju da je početni vrh, dodamo taj ciklus na listu koja sadrži sve pronađene cikluse. Kako bismo izbjegli dodavanje identičnih ciklusa, koristimo strukturu podataka set, u koji stavljamo sve vrhove koji su dio tog ciklusa i usporedimo ih. Ako već postoji ciklus koji ima iste elemente u tom setu, onda smo našli permutaciju već prije pronađenog ciklusa. U slučaju da susjed nije početni vrh i nije dio staze, dodamo ga na vrh stoga i pokrenemo potragu za susjedima za taj vrh. Ako ne pronađemo niti jednog susjeda koji zadovoljava te uvjete, tada trenutačni vrh skidamo sa stoga i nastavljamo pretragu susjeda za prethodni vrh od trenutka kada je pronađen susjed koji je skinut s vrha stoga. Zbog takvog načina rada, algoritam mora biti rekurzivan.

Pseudokod algoritma glasi ovako:

```
Za svaki vrh u Grafu
      Dodaj vrh na stog
      DFS(stog, vrh)
DFS(stog, vrh)
      Za svaki susjed od vrha stoga
            Ako susjed jednak vrh
                  Dodaj ciklus
                  Stog.pop()
            Ako stog ne sadrži susjed
                  Stog.push(susjed)
                  DFS(stog, vrh)
            Inače
                  Stog.pop()
```

Ovakav algoritam može se primijeniti na usmjerene i neusmjerene grafove jer samo gleda postojanje prijelaza između vrhova. Problem koji se može uočiti kod ovog algoritma prilikom traženja ciklusa jest da ima puno nepotrebnog ponovnog pretraživanja. Jedan od gorih slučajeva za ovakav algoritam prikazan je na slici ispod (Slika 2.2).



Slika 2.2 Loš graf za DFS [2].

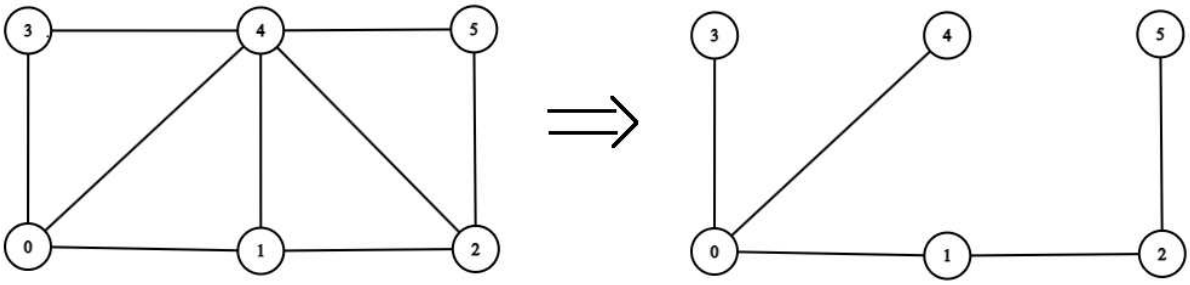
Možemo uočiti da postoji samo jedan ciklus u ovom grafu, no algoritam to ne može znati i mora provjeriti svaki vrh. Ako budemo provjeravali vrhove slijedom od 0 do 7, nakon što ispitamo vrh 1 kao korijenski, pronašli smo sve cikluse u grafu, ali algoritam će nastaviti dalje s provjerom vrhova 2, 3, ..., 6, 7, pronalazeći svaki put isti ciklus, samo u drugoj permutaciji. Za ovako mali i jednostavan graf nećemo uočiti prevelika vremena izvođenja programa, ali za grafove koji sadrže veliki broj vrhova to neće biti slučaj. Za rješavanje tog problema koristit ćemo drugi algoritam naveden u ovom radu i sljedećem potpoglavlju.

2.2. Algoritam kombinacije ciklusa

Važno je naglasiti da se ovaj algoritam može primijeniti samo na neusmjerene grafove jer ne uzima u obzir smjer prijelaza između vrhova

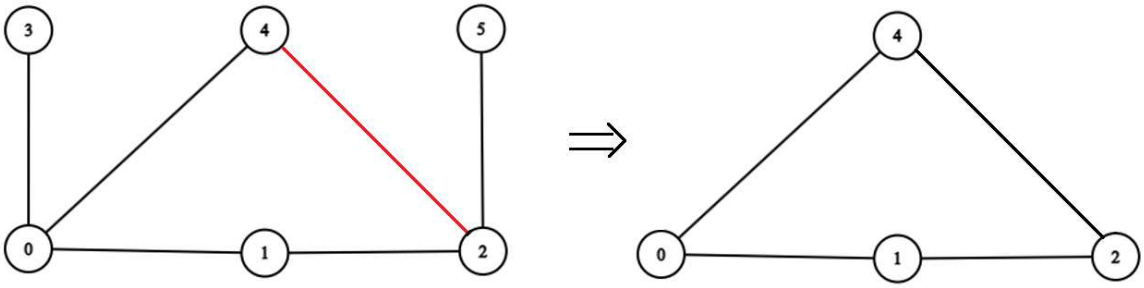
Prvi korak ovog algoritma je dobiti takozvane osnovne cikluse. Oni se dobiju tako da od početnog grafa napravimo razapinjuće stablo. Naravno, možemo dobiti različita stabla ako izmijenimo algoritam koji koristimo za njegovu konstrukciju ili promjenom početnog vrha stabla. Na našu sreću, ovo neće imati utjecaj na rezultat algoritma; samo će početak ciklusa biti neki drugi vrh. Na slici ispod (Slika 2.3) prikazan je jedan graf i jedno moguće

razapinjuće stablo koje smo dobili tako da počnemo u vrhu najmanjeg indeksa, dodamo brid svakom susjedu kojeg već nismo posjetili, i postupak ponovimo za sada novo posjećene vrhove.



Slika 2.3 Razapinjuće stablo grafa [2].

Nakon što smo dobili razapinjuće stablo, da bismo iz njega dobili naše osnovne cikluse, usporedimo stablo s početnim grafom i vidimo koji bridovi nedostaju. Zbog svojstva stabla, dodavanjem bilo kojeg od tih bridova između već dva postojeća vrha grafa garantiramo da će nastati ciklus. Postupak provedemo za svaki brid koji nedostaje i taj ciklus dodamo na našu listu ciklusa. Na slici ispod (Slika 2.4) prikazan je korak za jedan brid.



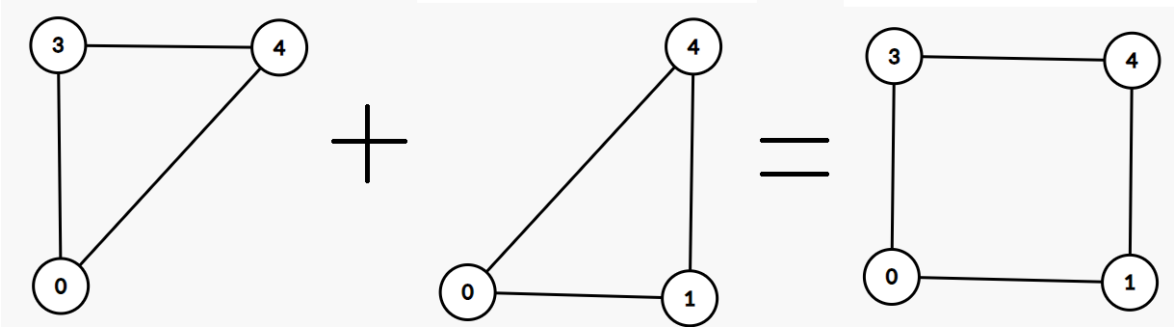
Slika 2.4 osnovni ciklus algoritma [2].

Nakon završetka tog koraka, dobili smo četiri osnovna ciklusa dodajući četiri nedostajuća brida u stablo, a to su ciklusi:

1. $0 \rightarrow 4 \rightarrow 3 \rightarrow 0$
2. $0 \rightarrow 1 \rightarrow 4 \rightarrow 0$
3. $0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 0$
4. $0 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 0$

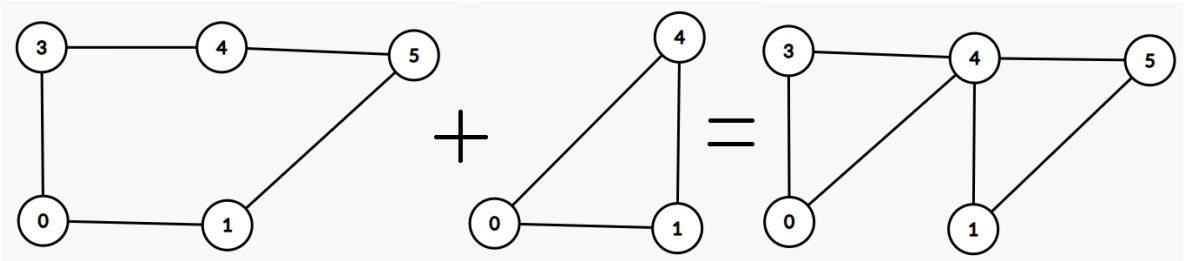
Posljednji korak je kombinacija ovih ciklusa da dobijemo sve ostale. Napravimo jedan red u koji ćemo stavljati cikluse koje nismo kombinirali. Na početku dodamo sve osnovne cikluse u taj red, naravno imamo odvojenu listu koja sadrži samo osnovne cikluse. Uzmemo jedan

ciklus iz reda i provodimo proces kombinacije njega sa svim osnovnim ciklusima; ako dobijemo novi ciklus, njega dodamo u red za čekanje. Zato nam je bilo bitno odvojiti osnovne cikluse od reda jer svaki novo-otkriveni ciklus moramo također kombinirati s osnovnim ciklusima da dobijemo druge neotkrivene cikluse. Na slici ispod (Slika 2.5) prikazan je primjer jedne takve kombinacije gdje kombiniramo ciklus 1. i ciklus 2. da dobijemo novi ciklus $0 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 0$.



Slika 2.5 kombinacija ciklusa [2].

Sam proces kombinacije neće se uvijek provesti. Važno je provesti dvije provjere tako da osiguramo da kombinacija daje ciklus, a ne nešto drugo. Prvo, ciklusi koje želimo kombinirati moraju imati najmanje jedan zajednički brid, kao što se može vidjeti na slici (Slika 2.5), gdje ciklusi dijele brid između vrhova 0 i 4. Drugi uvjet koji nam je potreban jest da, ako ciklusi imaju zajednički vrh, moraju imati barem jedan zajednički brid s tim vrhom. U slučaju da ne dijele brid, a dijele vrh, dobit ćemo nepravi ciklus jer će morati proći kroz taj vrh dva puta da ga napravi. Primjer takvog slučaja može se vidjeti na slici ispod (Slika 2.6). Ciklusi dijele brid između vrhova 0 i 1, ali imaju zajednički vrh 4 s kojim ne dijele brid, zato njihova kombinacija rezultira neispravnim ciklusom.



Slika 2.6 neispravna kombinacija [2].

Programska implementacija procesa kombinacije i provjere ovih uvjeta vrlo je jednostavna jer su sve što nam je potrebno matrice susjedstva dvaju ciklusa koje želimo kombinirati. Provjera imaju li ciklusi zajednički brid provodi se logičkom operacijom AND nad matricama ako rezultat nije nul-matrica, onda ciklusi imaju zajednički brid.

Provjera imaju li ciklusi zajednički vrh s kojim ne dijele brid provodi se tako da usporedimo rezultat logičke operacije AND. Ako rezultat operacije nad istim redovima (ili stupcima, zbog simetrije) nije samo nula, dakle ako nije nul-redak ili nul-stupac, onda kombinacija ne zadovoljava uvjet.

Na primjeru sa slike (Slika 2.6) peti redak (odnosi se na vrh 4 jer indeksi počinju od 0) matrica susjedstva ciklusa koje želimo kombinirati glasi [0 0 0 1 0 1] i [1 1 0 0 0 0]. Provođenjem logičke operacije AND dobiti ćemo [0 0 0 0 0 0].

Nakon što ciklusi ispune ta dva uvjeta, sama kombinacija ciklusa se provodi logičkim operatorom XOR (isključivo ili) nad matricama susjedstva, i rezultat će biti matrica susjedstva novog ciklusa. Jedino što nam ostaje nakon tog koraka jest provjera jesmo li već taj ciklus našli ranije.

3. Programska implementacija algoritma

Program je napisan u programskom jeziku Java. Uz glavnu klasu „Main“, potrebno je napraviti klasu „Ciklus“, koja se koristi za čuvanje podataka o pronađenim ciklusima i osiguravanje jedinstvenosti pronađenih ciklusa. Prvo ćemo objasniti klasu Ciklus.

Klasa Ciklus sadrži 5 atributa:

1. Int[][] matrica; matricu susjedstva tog ciklusa.
2. Int ID; redni broj ciklusa.
3. Set<Integer> nulLines; set indekse nul-linija matrice susjedstva kako bi ubrzali proces kombiniranje ciklusa.
4. List<Integer> cycle; lista vrhova kroz koji ciklus ide u pravilnom redoslijedu.
5. Boolean negative; zastavica koja govori jeli ciklus negativan (u težinskom grafu ako je ukupna cijena puta manja od 0).

Za klasu potrebna su nam konstruktora:

1. public ciklus(int[][] matrica, List<Integer> cycle, int ID, boolean negative) ovaj konstruktor koristimo uz algoritam kombinacije ciklusa.
2. public ciklus(List<Integer> cycle, boolean negative) ovaj konstruktor koristimo uz DFS algoritam.

Klasa sadrži 3 metode koje nisu getter ili setter.

1. void calculateNulLines(); ova metoda iterira po matrici susjedstva i napuni listu nulLines sa svim pronađenim nul-linija unutar matrice
2. public int hashCode(); ovo je override standardne metode hashCode() kako bi mogli osigurati jedinstvenost ciklusa. Hash code generiramo na osnovu sortirane liste vrhova ciklusa, tako osiguramo ako dodamo identičan ciklus generirani hash code će biti isti i onda se neće dodati u listu već pronađenih ciklusa.
3. Public String toString(); metoda koju koristimo pri ispisu svih pronađenih ciklusa, ona na temelju liste vrhova ispiše ih u finom formatu vrh1 → vrh2 → … → vrhn → vrh1.

3.1. Pokretanje i korištenje programa

Program ima mogućnost ručnog unosa bridova između vrhova ili učitavanje matrice prijelaza iz tekstualnog dokumenta „graf.txt“. Program se može pokrenuti iz terminala preko naredbe „java -jar zavrsni.jar“ u radnom direktoriju gdje se nalazi program. Pri pokretanju program pita za par ključnih svojstava željenog grafa, unesite vrijednost navedenom u zagradama pored pitanja („y“ za da, „n“ za ne) primjer odgovara može se vidjeti na slici(Slika 3.1). Prvo pita je li graf usmjeren ili neusmjeren. U slučaju da je graf neusmjeren, pita za izbor algoritma (1 za DFS ili 2 za metodu kombinacije). Nakon toga, program pita je li graf težinski. Na kraju pita želite li učitati graf preko datoteke ili ručno unijeti podatke. Strogo preporučujem da se podaci unesu preko datoteke zbog uštede vremena za unos grafova s puno vrhova.

```
C:\FER\6. Semestar\zavrsni rad\Za predaju>java -jar zavrsni.jar
Usmjereni graf? (y/n) n
algoritam 1 ili 2? (1/2) 1
Tezinski graf? (y/n) n
Unos preko file-a? (y/n) y
========================================
```

Slika 3.1 Primjer pokretanje programa.

Za unos preko datoteke, u direktoriju gdje se nalazi glavni program, trebate imati datoteku nazvanu „graf.txt“. Ako datoteka ne postoji, ručno je napravite. Unutar te datoteke potrebno je samo unijeti matricu susjedstva željenog grafa, primjer izgleda sadržaja tekstualne datoteke možete vidjet na slici ispod (Slika 3.2).

```
1  0 1 0 1 1 0
2  1 0 1 0 1 0
3  0 1 0 0 1 1
4  1 0 0 0 1 0
5  1 1 1 1 0 1
6  0 0 1 0 1 0
```

Slika 3.2 izgled sadržaja datoteke graf.txt.

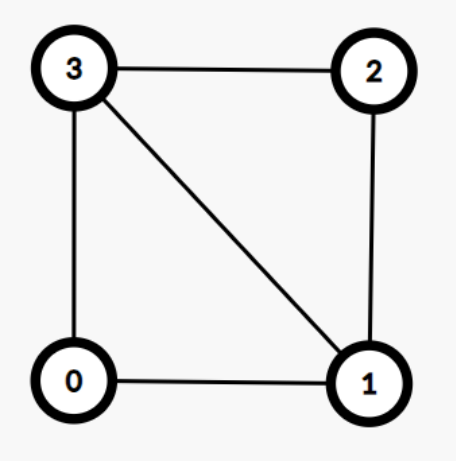
Puni ispis programa s unesenim vrijednostima iz slike (Slika 3.1) i grafom s matricom prijelaza iz slike (Slika 3.2) može se vidjeti na slici ispod (Slika 3.3).

```
Usmjereni graf? (y/n) n
algoritam 1 ili 2? (1/2) 1
Tezinski graf? (y/n) n
Unos preko file-a? (y/n) y
=======================================
Svi ciklusi:
===========================
Duljina: 3      0 -> 1 -> 4 -> 0
Duljina: 3      0 -> 3 -> 4 -> 0
Duljina: 3      1 -> 2 -> 4 -> 1
Duljina: 3      2 -> 4 -> 5 -> 2
Duljina: 4      0 -> 1 -> 2 -> 4 -> 0
Duljina: 4      0 -> 1 -> 4 -> 3 -> 0
Duljina: 4      1 -> 2 -> 5 -> 4 -> 1
Duljina: 5      0 -> 1 -> 2 -> 4 -> 3 -> 0
Duljina: 5      0 -> 1 -> 2 -> 5 -> 4 -> 0
Duljina: 6      0 -> 1 -> 2 -> 5 -> 4 -> 3 -> 0
```

Slika 3.3 pokrenuti program.

3.2. Primjeri i ispisi programa

U ovom potpoglavlju prikazat ćemo korake pronalaženja ciklusa za oba algoritma i nekoliko primjera. Za oba algoritma koristit ćemo graf prikazan na slici ispod (Slika 3.4).



Slika 3.4 graf za primjer [2].

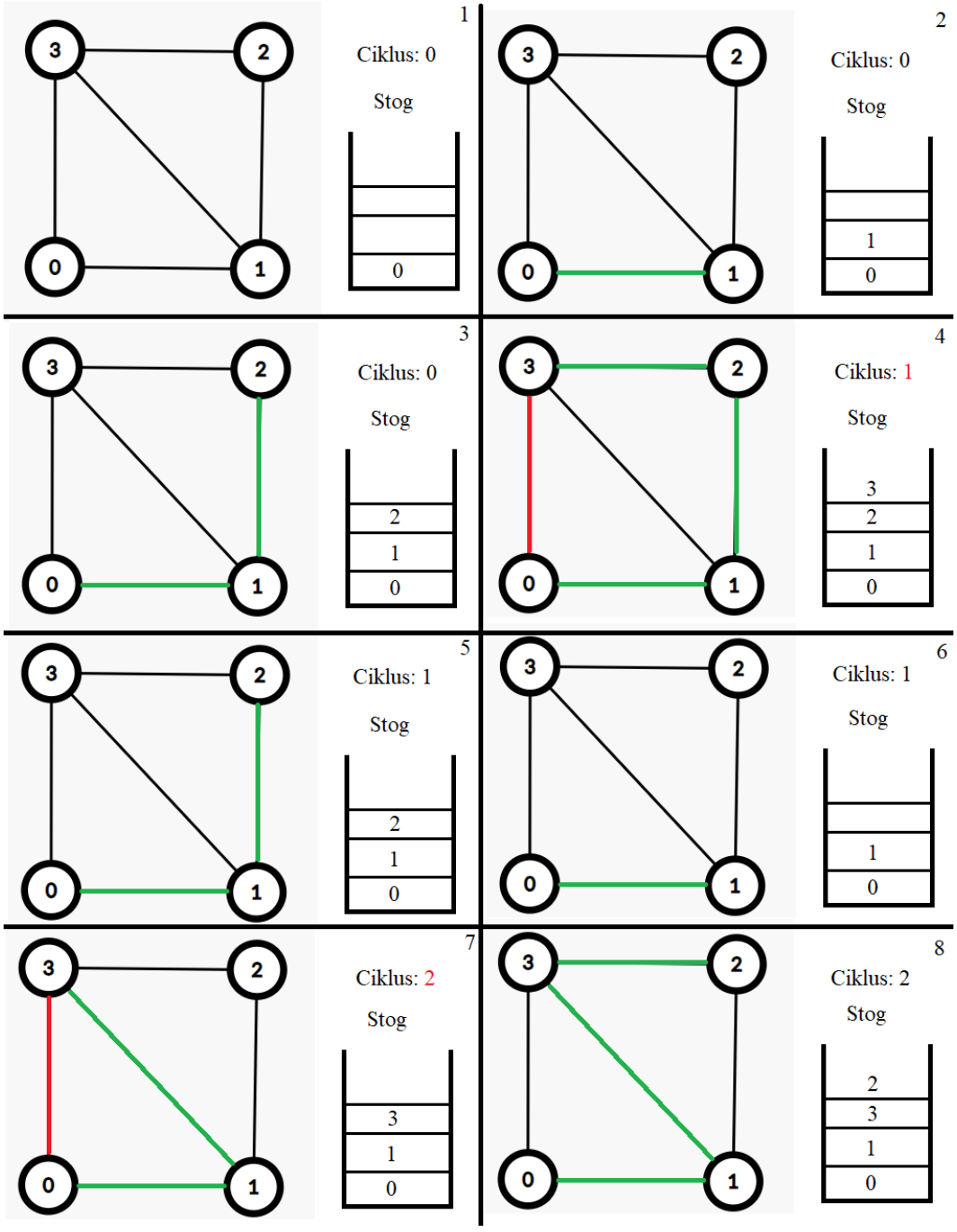
3.2.1. Rad programa pri depth-first searchu

Algoritma za svaki vrh u grafu provede pretragu tražeći ciklus koji vodi nazad do početnog vrha. Pretraga počinje iz vrha s indeksom 0, njega stavljamo na stog i gledamo sljedećeg susjeda s najnižim indeksom da nije već u stogu. Ako je početni vrh jedan od susjeda (i na stogu su više od 2 vrha) onda smo pronašli ciklus te ga dodajemo u listu ciklusa. Na slici ispod (Slika 3.5) prikazano je prvih 8 koraka algoritma, broj koraka prikazan je u gornjem desnom kutu.

Kratki opis koraka:

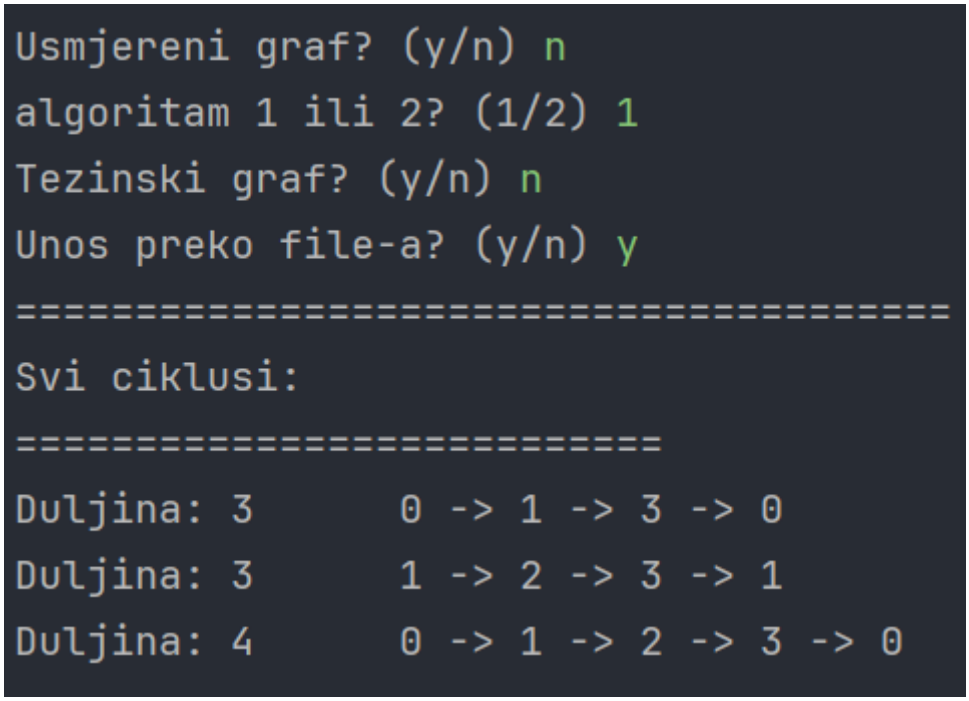
1. Početni vrh stavljen a stog
2. Prvi susjed najnižeg indeksa koji nije na stogu se posjeti i stavi na stog
3. Korak broj 2 ponavlja se dok nestane neposjećenih susjeda ili početni vrh bude jedan od susjeda.
4. Desio se slučaj da je početni vrh susjed i da u stogu ima više od 2 vrha, dodamo ciklus u listu ciklusa.
5. Nestalo susjeda koji zadovoljavaju uvjet posjete, te se trenutačni vrh skida s vrha stoga i nastavlja se pretraga od prethodnog vrha.

6. Ista situacija kao kod koraka broj 5
7. Sljedeći susjed od vrha broj 1 je vrh broj 3 te se dodaje na stog i ponovi se situacija u koraku broj 4.
8. Ista situacija kao u koraku broj 2.
9. …



Slika 3.5 Način rada DFS-a [2].

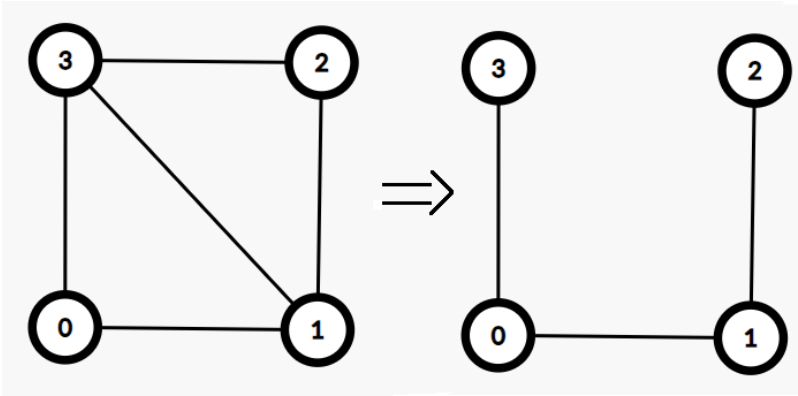
Iz prethodnog opisa koraka možemo vidjeti da se algoritam svede da ponavljanje koraka broj 2 sve dok se ne ispuni uvjet za korak broj 4, ili ako taj uvjet nije ispunjen i ne može se provesti korak broj 2, onda provedemo korak broj 5. Ovi koraci ponavljaju se sve dok ne ispraznimo stog, nakon čega se pokrene isti postupak krenuvši od sljedećeg vrha i tako za sve vrhove u grafu. Ispis programa koristeći ovaj algoritam prikazan je na slici ispod (Slika 3.6).



Slika 3.6 ispis algoritma 1.

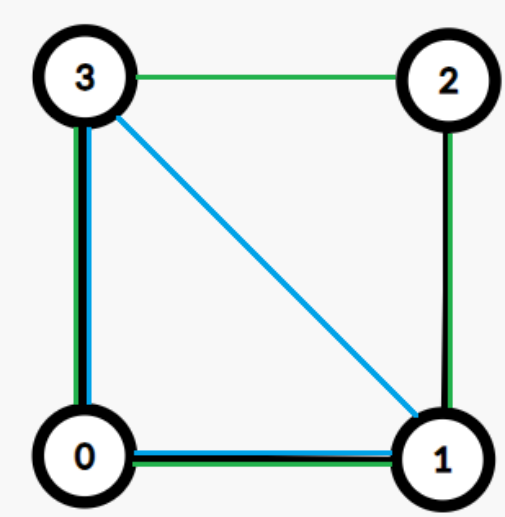
3.2.2. Rad programa pri metodi kombinacije osnovnih ciklusa

Algoritam prvo iz zadanog grafa napravi razapinjuće stablo. Graf može imati više razapinjućih stabala, ali to neće utjecati na konačni broj pronađenih ciklusa. U ovoj implementaciji koristio sam pretragu u širinu (Breath-first search, BFS) algoritam kako bih generirao stablo. Na slici ispod (Slika 3.7) prikazano je stablo koje će algoritam generirati na grafu iz slike 3.4 (Slika 3.4).



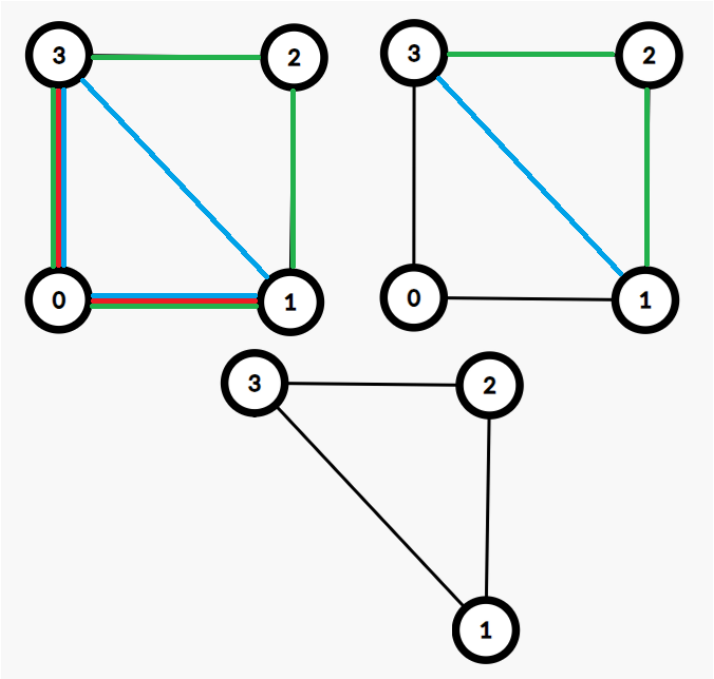
Slika 3.7 Razapinjuće stablo [2].

Nakon što algoritam generira stablo, uspoređuje ga s početnim grafom. Svi bridovi koje početni graf ima a stablo nema, njegovim dodavanjem dobiti ćemo osnovni ciklus algoritma. Iz iznad prikazane slike (Slika 3.7) vidimo da postoje dva takva brida i možemo zaključiti da ćemo imati dva osnovna ciklusa. Ti ciklusi prikazani su na slici ispod (Slika 3.8). Zelenom bojom označen je jedan osnovni ciklus, plavom drugi osnovni ciklus, crni bridovi su bridovi stabla.



Slika 3.8 Osnovni ciklusi grafa [2].

Sljedeći korak algoritam **sve osnovne cikluse** pokušava spojiti sa svakim osnovnim i novo otkrivenim ciklusom. Na slici ispod (Slika 3.9) prikazan je proces provjere ispravne kombinacije i konačne kombinacije ciklusa iz prethodne slike (Slika 3.8).



Slika 3.9 Kombinacija ciklusa [2].

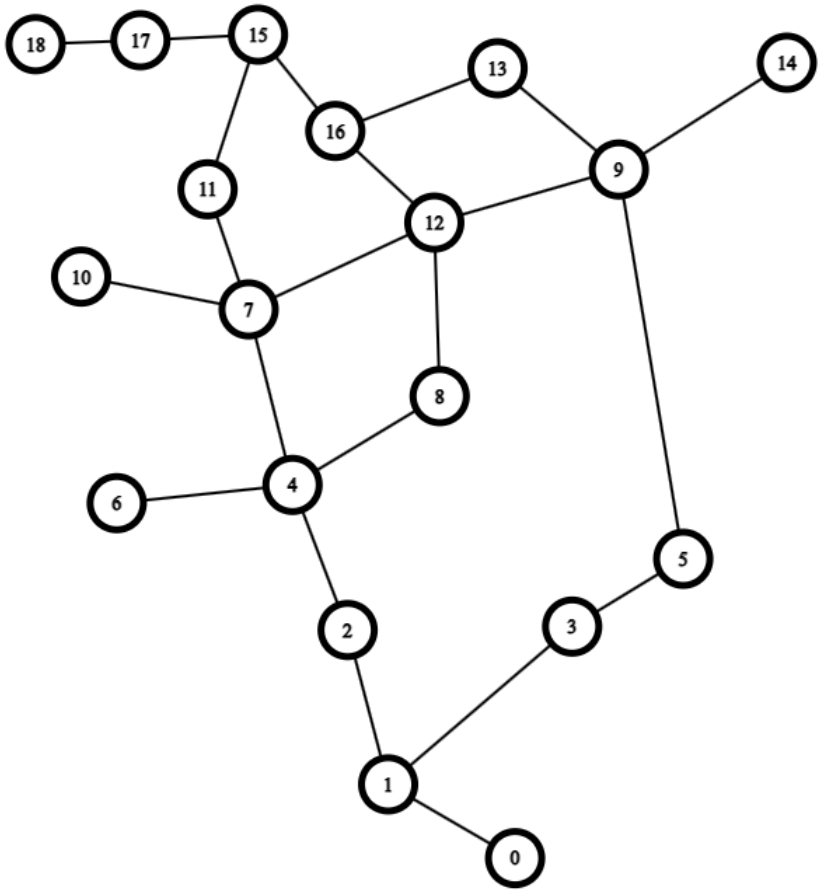
Zelenom bojom označen je prvi ciklus, a plavom bojom drugi ciklus. Crvenom bojom označeni su zajednički bridovi koje dijele ova dva ciklus te s njihovim postojanjem i provjerom uvjeta visećeg zajedničkog vrha navedenog u poglavlju 2.2 (Algoritam kombinacije ciklusa) algoritam zna da iz može kombinirati. Desni graf predstavlja tu kombinaciju. Izbacuju se zajednički bridovi tih ciklusa i ostatak tog izbacivanja je ciklus koji se može vidjeti na dnu slike. Postupak provjere se ponavlja s tim novim ciklusom i tako sve dok ne prestanemo dobivati nove cikluse. Ispis programa koristeći ovaj algoritam prikazan je na slici ispod (Slika 3.10).

```
Usmjereni graf? (y/n) n
algoritam 1 ili 2? (1/2) 2
Tezinski graf? (y/n) n
Unos preko file-a? (y/n) y
========================================
Svi ciklusi:
============================
Duljina: 3      1 -> 0 -> 3 -> 1
Duljina: 3      1 -> 2 -> 3 -> 1
Duljina: 4      2 -> 1 -> 0 -> 3 -> 2
```

Slika 3.10 Ispis algoritma 2.

3.2.3. Dodatni primjer

U ovom kratkom poglavlju prikaza je graf od 19 vrhova i ispis programa za oba algoritma navedena u ovom radu.



Slika 3.11 Graf za primjer 2 [2].

```
Usmjereni graf? (y/n) n
algoritam 1 ili 2? (1/2) 1
Tezinski graf? (y/n) n
Unos preko file-a? (y/n) y
=====================================
Svi ciklusi:
==========================
Duljina: 4      4 -> 7 -> 12 -> 8 -> 4
Duljina: 4      9 -> 12 -> 16 -> 13 -> 9
Duljina: 5      7 -> 11 -> 15 -> 16 -> 12 -> 7
Duljina: 7      7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 12 -> 7
Duljina: 7      4 -> 7 -> 11 -> 15 -> 16 -> 12 -> 8 -> 4
Duljina: 8      1 -> 2 -> 4 -> 7 -> 12 -> 9 -> 5 -> 3 -> 1
Duljina: 8      1 -> 2 -> 4 -> 8 -> 12 -> 9 -> 5 -> 3 -> 1
Duljina: 9      4 -> 7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 12 -> 8 -> 4
Duljina: 10     1 -> 2 -> 4 -> 7 -> 12 -> 16 -> 13 -> 9 -> 5 -> 3 -> 1
Duljina: 10     1 -> 2 -> 4 -> 8 -> 12 -> 16 -> 13 -> 9 -> 5 -> 3 -> 1
Duljina: 11     1 -> 2 -> 4 -> 7 -> 11 -> 15 -> 16 -> 12 -> 9 -> 5 -> 3 -> 1
Duljina: 11     1 -> 2 -> 4 -> 7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 5 -> 3 -> 1
Duljina: 13     1 -> 2 -> 4 -> 8 -> 12 -> 7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 5 -> 3 -> 1
```

Slika 3.12 Ispis algoritma 1.

```
Usmjereni graf? (y/n) n
algoritam 1 ili 2? (1/2) 2
Tezinski graf? (y/n) n
Unos preko file-a? (y/n) y
=====================================
Svi ciklusi:
==========================
Duljina: 4      8 -> 4 -> 7 -> 12 -> 8
Duljina: 4      9 -> 12 -> 16 -> 13 -> 9
Duljina: 5      15 -> 11 -> 7 -> 12 -> 16 -> 15
Duljina: 7      7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 12 -> 7
Duljina: 7      4 -> 7 -> 11 -> 15 -> 16 -> 12 -> 8 -> 4
Duljina: 8      9 -> 5 -> 3 -> 1 -> 2 -> 4 -> 7 -> 12 -> 9
Duljina: 8      1 -> 2 -> 4 -> 8 -> 12 -> 9 -> 5 -> 3 -> 1
Duljina: 9      4 -> 7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 12 -> 8 -> 4
Duljina: 10     13 -> 9 -> 5 -> 3 -> 1 -> 2 -> 4 -> 7 -> 12 -> 16 -> 13
Duljina: 10     1 -> 2 -> 4 -> 8 -> 12 -> 16 -> 13 -> 9 -> 5 -> 3 -> 1
Duljina: 11     1 -> 2 -> 4 -> 7 -> 11 -> 15 -> 16 -> 12 -> 9 -> 5 -> 3 -> 1
Duljina: 11     1 -> 2 -> 4 -> 7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 5 -> 3 -> 1
Duljina: 13     1 -> 2 -> 4 -> 8 -> 12 -> 7 -> 11 -> 15 -> 16 -> 13 -> 9 -> 5 -> 3 -> 1
```

Slika 3.13 Ispis algoritma 2.

Zaključak

Nakon detaljnog objašnjenja osnovnih pojmova kao što su matrica susjedstva, jednostavni graf i ciklus, te opisa algoritama za pronalaženje ciklusa u grafu, utvrdili smo da je problem rješiv. Prvo smo pokazali depth first search algoritam i njegovu primjenu za traženje ciklusa u grafu. Zatim smo predstavili metodu kombiniranja ciklusa kako bismo otkrili sve cikluse u grafu. Kroz programsku implementaciju riješili smo probleme duplikata ciklusa te zaključili da je manipulacija matricom susjedstva ključna za postizanje cilja traženja ciklusa. Primjeri koje smo pokazali demonstrirali su da algoritam funkcionira kako na malim, tako i na velikim grafovima, te da nema problema u pronalaženju ciklusa.

Literatura

[1] Kovačević, D., Krnić, M., Nakić, A., Pavčević, M.O. Diskretna matematika 1. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2020.

[2] https://csacademy.com/app/graph_editor

[3] https://www.geeksforgeeks.org/detect-cycle-in-a-graph/

Sažetak

Neki algoritmi za određivanje ciklusa u grafu,

Ovaj završni rad bavi se problematikom određivanja ciklusa u jednostavnom grafu. U prvom dijelu rada objašnjeni su osnovni pojmovi vezani uz grafove, uključujući definiciju jednostavnog grafa, matrice susjedstva i ciklusa. U središnjem dijelu rada predstavljena su dva algoritma za pronalaženje svih ciklusa u grafu: algoritam pretraživanja u dubinu (Depth First Search - DFS) i algoritam kombinacije ciklusa. Za svaki od ovih algoritama detaljno je opisan način rada te su prikazani pseudokod i programske implementacije. Prikazani su i konkretni primjeri koji ilustriraju primjenu ovih algoritama na različitim grafovima, čime je demonstrirana njihova učinkovitost i primjenjivost u praksi.

Ključne riječi: DFS, ciklus u grafu, teorija grafova, algoritam.

Summary

Some algorithms for determining cycles in a graph,

This thesis addresses the problem of detecting cycles in a simple graph. The first part of the thesis explains basic concepts related to graphs, including the definition of a simple graph, adjacency matrix, and cycles. The central part of the thesis presents two algorithms for finding all cycles in a graph: the Depth First Search (DFS) algorithm and the cycle combination algorithm. Each algorithm is thoroughly explained, including their operational mechanisms, pseudocode, and software implementations. Concrete examples are provided to illustrate the application of these algorithms to different graphs, demonstrating their efficiency and practical applicability.

Keywords: DFS, cycle in a graph, graph theory, algorithm.

Privitak

Za pokretanje programa potrebno je imati instaliranu Javu. Najnoviju verziju Jave možete preuzeti ovdje: https://www.oracle.com/java/technologies/downloads/. Nakon što preuzmete program i tekstualni dokument "graf.txt", pozicionirajte se preko terminala u direktorij koji sadrži te datoteke. Unutar tekstualnog dokumenta "graf.txt" unesite matricu prijelaza željenog grafa i spremite datoteku. Program se pokreće unosom sljedeće naredbe u terminal: „java -jar zavrsni.jar“. Nakon pokretanja programa, slijedite upute koje će se prikazivati u terminalu i odgovarajte na njih unosom opcija navedenih u zagradama.