Algoritmi u teoriji grafova

Buljubašić, Nena

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Nena Buljubašić

ALGORITMI U TEORIJI GRAFOVA

Diplomski rad

Voditelj rada:
doc. dr. sc. Slaven Kožić

Zagreb, studeni, 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Mojoj sestri Eni, koja me i bez riječi naučila svemu.
Roditeljima, jer je uz njih sve lakše.

Sadržaj

Uvod

Teorija grafova je grana diskretne matematike koja se bavi grafovima, a ima primjenu u različitim područjima, primjerice u računarstvu, biologiji, fizici pa čak i u socijalnim znanostima. Jedan od najpoznatijih problema, a ujedno i jedan od prvih zapisa o teoriji grafova je Eulerov problem sedam mostova Koeningsberga, objavljen 1736. Problem se sastoji od toga da se treba pronaći put kroz grad tako da se preko svakog mosta prijeđe točno jednom. Vidimo da je teorija grafova relativno mlada grana matematike koja se počinje detaljnjije istraživati tek u 18. stoljeću, a danas je njena primjena sve šira. Danas su najpoznatiji primjer upotrebe iste upravo društvene mreže koje počivaju na grafovima i upravo zbog toga su postali poznati brojni algoritmi na grafovima koji omogućuju njihovo pretraživanje i provjeravanje različitih svojstava.
Na samom početku rada, u poglavlju 1 možemo pronaći osnovne pojmove teorije grafova koji su nam potrebni za razumijevanje ostatka rada. Poglavlje prati dijelove knjige [3].
U poglavlju 2 proučavamo algoritme za detekciju ciklusa na usmjerenim grafovima o kojima se više može pročitati u člancima [6], [2] i [7]. Detekcija ciklusa je iznimno važna kod implementacije vezanih lista za osiguravanje konačnosti programa, a opisani algoritmi služe za ostvarivanje navedenoga. U poglavlju su opisana četiri algoritma, Floydov, Brentov, Gosperov i Nivaschov. U računarstvu se najčešće spominje i koristi Floydov algoritam koji zbog korištenja dvaju pokazivača na vrhove grafa, brzog i sporog, stječe popularan naziv algoritam "kornjače i zeca". Svi od navedenih algoritama imaju sličnu složenost, a zanimljivo je proučiti potpuno različite pristupe istom problemu. Naime, Floydov i Brentov algoritam imaju pristup takav da se koriste dva pokazivača koji se kreću koracima različite veličine po grafu i algoritmi će se zaustaviti ako se pokazivači nađu na istom vrhu grafa. S druge strane, Gosperov algoritam se temelji na konstruiranju podskupova niza kojeg čine vrhovi grafa i traženja istih elemenata usporedbom. Nivaschov se algoritam razlikuje od ostalih jer pretpostavlja da je skup vrhova grafa totalno uređen skup. Za svaki su algoritam dani primjeri izvođenja na konkretnom grafu.
U poglavlju 3 opisuju se rekurzivni algoritmi za pronalaženje maksimalnih klika u grafu o kojima se više može pročitati u člancima [1] i [4]. Algoritmi koje proučavamo su Bron–Kerboschov i Akkoyunluov. Opisali smo dvije verzije Bron–Kerboschovog algoritma, s pivotiranjem i bez pivotiranja. Efikasnijom se pokazala verzija s pivotiranjem jer ima ma-

nje rekurzivnih poziva. Ideja je odabrati pivotni element i tražiti maksimalne klike kojima on pripada i maksimalne klike kojima pripadaju njemu nesusjedni elementi jer vrh i njemu nesusjedni vrhovi ne mogu biti sadržani u istoj kliki koja je potpuni podgraf danog grafa. Akkoyunluov algoritam je vrlo sličan Bron–Kerboschovom, ali ima potpuno drugačiji pristup problemu te je formalnije definiran.

Poglavlje 1

Osnovni pojmovi u teoriji grafova

Definirat ćemo pojmove koji su nužni za razumijevanje sljedećih poglavlja, a detaljnjiji uvod u teoriju grafova može se pronaći u [3].

Definicija 1.0.1. *Graf G je uređeni par $G = (V, E)$, gdje je V neprazan skup čije elemente nazivamo vrhovima i E skup dvočlanih podskupova od V čije elemente nazivamo bridovima.*

Definicija 1.0.2. *Kažemo da su dva vrha v, w grafa $G = (V, E)$ susjedna ako vrijedi $\{v, w\} \in E$. Kažemo da je graf potpun ako su mu svaka dva vrha susjedna.*

Definicija 1.0.3. *Za svaki vrh x grafa $G = (V, E)$ D_x je skup svih vrhova koji nisu susjedni vrhu x, a C_x skup svih vrhova koji su susjedni vrhu x.*

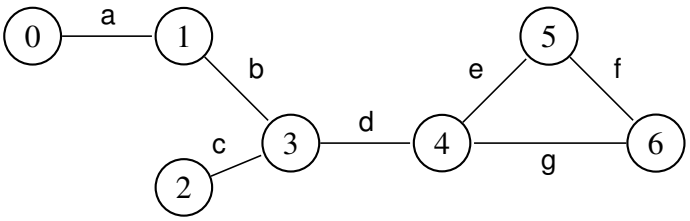
Definicija 1.0.4. *Komplement grafa $G = (V, E_1)$ je graf $G^C = (V, E_2)$ s istim skupom vrhova kao i G te vrijedi da su svaka dva vrha susjedna u G^C ako i samo ako nisu susjedna u G, tj. $e \in E_2$ ako i samo ako $e \notin E_1$.*

Definicija 1.0.5. *Šetnja W u grafu G je konačan niz $v_0 e_1 v_1 \dots e_k v_k$ vrhova v_j, $j \in \{0, \dots k\}$ i bridova e_i, $i \in \{1, \dots k\}$ pri čemu su krajevi brida e_i vrhovi v_{i-1} i v_i. Vrh v_0 nazivamo početak, a v_k kraj šetnje. Ostale vrhove nazivamo unutarnjim vrhovima. Šetnja je zatvorena ako vrijedi $v_0 = v_k$. Staza u grafu je šetnja u kojoj su svi bridovi međusobno različiti. Put je staza s međusobno različitim vrhovima. Ciklus je zatvorena staza koja sadrži barem jedan brid i unutarnji vrhovi su joj međusobno različiti.*

Definicija 1.0.6. *Neka su $G = (V_1, E_1)$ i $H = (V_2, E_2)$ grafovi takvi da vrijedi $V_2 \subseteq V_1$ i $E_2 \subseteq E_1$. Kažemo da je H podgraf grafa G i pišemo $H \subseteq G$.*

Definicija 1.0.7. *Graf $G = (V, E)$ je povezan ako su svaka dva vrha $u, v \in V$ povezana putom. Klika u grafu G je svaki povezan potpun podgraf.*

Primjer 1.0.8. *Promotrimo opisane pojmove na primjeru sljedećeg neusmjerenog grafa* $G = (V, E)$.

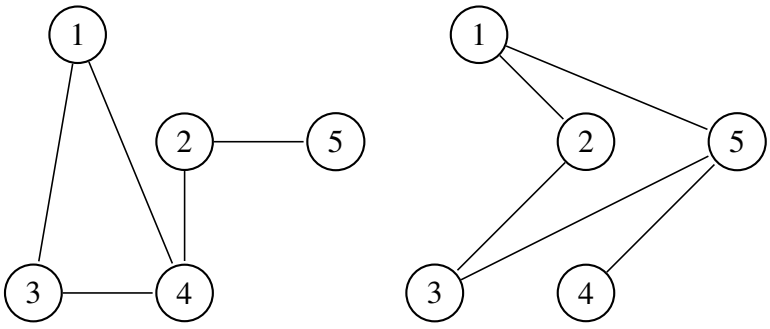


Slika 1.1: Neusmjeren graf G

Skup vrhova danog grafa je $V = \{0, 1, 2, 3, 4, 5, 6\}$*, a bridova* $E = \{\{0, 1\}, \{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 6\}, \{4, 5\}, \{5, 6\}\}$. *Radi jednostavnosti, bridove smo označili slovima* $a, \ldots, g$*, gdje je* $a = \{0, 1\}$*,* $b = \{1, 3\}$*, itd. Graf nije potpun jer primjerice vrhovi 0 i 2 nisu susjedni. Za vrh* $x = 1$ *skup susjednih vrhova je* $C_1 = \{0, 3\}$*, a nesusjednih* $D_1 = \{2, 4, 5, 6\}$.

Primjer jedne šetnje je niz vrhova i bridova $v_0e_1v_1e_2v_2 = 2, c, 3, d, 4$. *Vidimo da je ova šetnja ujedno i staza jer su svi bridovi međusobno različiti, ali i put jer su međusobno različiti i svi vrhovi. Opisana šetnja (staza) nije ciklus jer nije zatvorena. Primjer ciklusa je staza* $v_0e_1v_1e_2v_2e_3v_3 = 4, e, 5, f, 6, g, 4$.

Vrhovi 4, 5 i 6 zajedno s bridovima e, f i g čine podgraf $H = (V_1, E_1) = (\{4, 5, 6\}, \{e, f, g\})$ *grafa G. Podgraf H je potpun jer su mu svi vrhovi međusobno susjedni pa kažemo da je H klika. Pogledajmo u nastavku i primjer grafa i njegovog komplementa.*



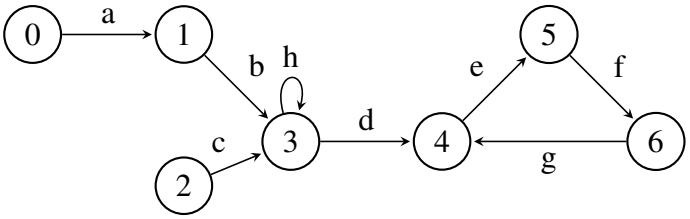
Slika 1.2: Graf G (lijevo) i njegov komplement G^C (desno)

Osim neusmjerenih grafova koji su opisani u definiciji (1.0.1), velik dio rada bit će posvećen usmjerenim grafovima.

Definicija 1.0.9. *Usmjereni graf (digraf) D je uređen par $D = (V, A)$, gdje je V neprazan skup čije elemente nazivamo vrhovima, a A skup uređenih parova $(u, v) \in V \times V$ čije elemente nazivamo lukovima. Pri tome vrh u nazivamo početak, a v kraj luka. Luk kojem su početak i kraj isti nazivamo petlja.*

Definicija 1.0.10. *Šetnja u digrafu D je niz $W = v_0 a_1 v_1 a_2 \dots a_k v_k$ čiji su članovi naizmjenično vrhovi i lukovi tako da je vrh v_{i-1} početak, a vrh v_i kraj luka a_i. Usmjerena šetnja je zatvorena ako su njezin početak i kraj isti, tj. vrijedi $v_0 = v_k$. Usmjereni ciklus je zatvorena usmjerena šetnja kojoj su svi lukovi međusobno različiti i vrhovi, osim početnog i krajnjeg.*

Primjer 1.0.11. *Promotrimo opisane pojmove na primjeru sljedećeg usmjerenog grafa.*



Slika 1.3: Usmjeren graf

Skup vrhova danog digrafa je $V = \{0, 1, 2, 3, 4, 5, 6\}$, a lukova $E = \{(0, 1), (1, 3), (2, 3), (3, 3), (3, 4), (6, 4), (4, 5), (5, 6)\}$. Radi jednostavnosti, lukove smo označili slovima $a, \dots, h$, gdje je $a = (0, 1)$, $b = (1, 3)$, itd. Primjer usmjerene šetnje u digrafu je niz $W = v_0 e_1 v_1 e 2 v_2 = 2, c, 3, d, 4$. Luk h je petlja jer mu je početak i kraj isti.

Poglavlje 2

Problem detekcije ciklusa

2.1 Opis problema i motivacija

U računarstvu se problem detekcije ciklusa grafa svodi na pronalaženje ciklusa u nizu vrijednosti iterirajuće funkcije.

Definicija 2.1.1. *Neka je $\mathcal{M}$ konačan skup koji se sastoji od m elemenata. Za svako pridruživanje $f : \mathcal{M} \to \mathcal{M}$ i proizvoljan element $a_0 \in \mathcal{M}$ definiramo niz $a_0, a_1, ...$ s*

$$a_{n+1} = f(a_n), \quad n = 0, 1, ... \tag{2.1}$$

Reći ćemo da postoji ciklus u pridruživanju f ako postoji $\lambda \geq 0$ tako da se dani niz počinje ponavljati od elementa a_λ, tj. postoji $\tau \geq 1$ takav da je

$$a_n = a_{n+\tau}, \quad za\ sve\ n \geq \lambda \geq 0. \tag{2.2}$$

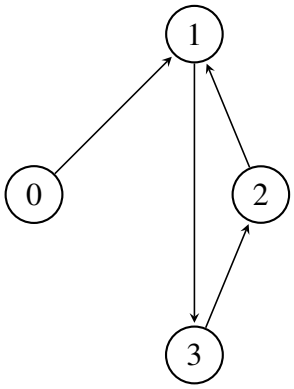
Najmanji broj τ za koji vrijedi (2.2) *nazivamo duljina ciklusa.*

U nastavku ćemo s λ označavati prvi element u danom nizu nakon kojeg se elementi počinju ponavljati. Problem detekcije ciklusa svodi se na određivanje vrijednosti λ i τ za dane f i a_0. Pridruživanje f iz definicije (2.1.1) u terminima teorije grafova ima značenje usmjerenog grafa u kojem svaki vrh ima najviše najviše jedan izlazni brid. Slijedi primjer grafa, odnosno pridruživanja s ciklusom.

Primjer 2.1.2. *U nastavku je dana tablica s vrijednostima pridruživanja f na konačnom skupu $\mathcal{M} = \{0, 1, 2, 3\}$ i odgovarajući graf.*

x	$f(x)$
0	1
1	3
2	1
3	2

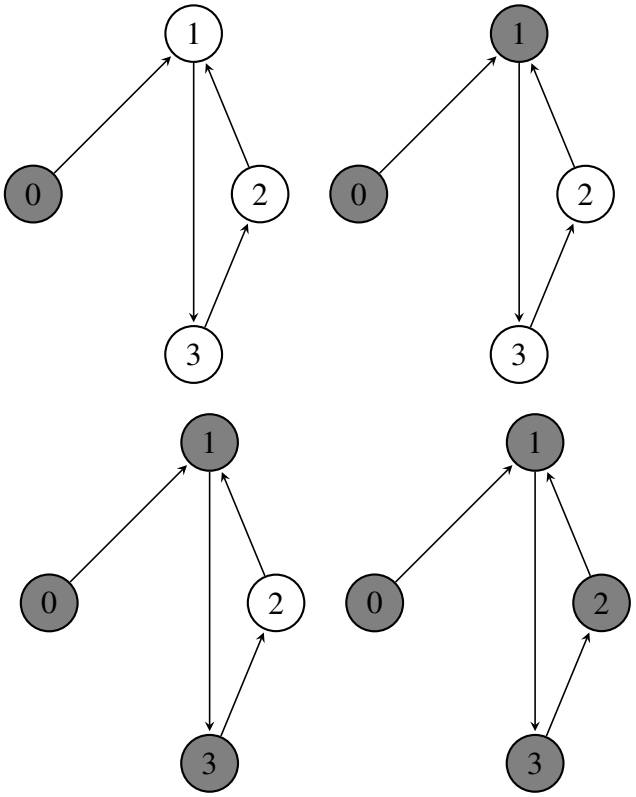
Tablica 2.1: Vrijednosti funkcije f



Slika 2.1: Graf

Za dano pridruživanje i proizvoljni element $a_0 = 0$*, definiramo niz* $0, 1, 3, 2, 1, 3, 2, 1, ...$ *kao u* (2.1.1)*. Traženi* λ *i* τ *su jednaki* $\lambda = 3 = \tau$*.*

Postoji nekoliko efikasnih algoritama za detekciju ciklusa, od kojih gotovo svi imaju različit pristup samom problemu. Za detekciju ciklusa možemo iskoristiti vrlo poznati algoritam pretraživanja po dubini grafa. Ideja je da je svaki vrh obojan nekom od triju boja: bijelom, sivom ili crnom. Na samom početku svi su vrhovi bijele boje. Za svaki vrh znamo koji su mu susjedni vrhovi, tj. poznati su nam svi bridovi grafa. Algoritam ponavljamo za svaki vrh grafa dok ne pronađemo ciklus ili dok se algoritam ne završi za sve vrhove u slučaju da ciklus ne postoji. Počnemo od proizvoljno odabranog vrha, obojimo ga u sivo te tražimo njemu susjedne vrhove koji su bijele boje te i njih bojimo u sivo. Kažemo da su sivo obojani vrhovi koji su na trenutnom putu u grafu. Zatim rekurzivno pozivamo isti algoritam na svim susjednim vrhovima. Vrh kojem su posjećeni svi susjedi i nije više dio nijednog puta u grafu bojimo u crno. Ukoliko je za vrijeme provjere susjeda neki od njih već obojan u sivo to znači da smo pronašli ciklus i algoritam se prekida s porukom da ciklus postoji. Međutim, prostorna složenost ovog algoritma je $O(V + E)$ gdje je V broj vrhova grafa, a E broj bridova pa nam je u cilju promatrati algoritme manje prostorne složenosti kako bi bili optimalni i za grafove s velikim brojem vrhova i bridova. U nastavku vidimo primjer izvođenja opisanog algoritma na grafu sa slike (2.1).



Slika 2.2: Detekcija ciklusa pretraživanjem po dubini

U danom primjeru svi vrhovi su ostali obojani u sivo jer su dio puta. Očito ciklus postoji jer iz vrha 2 u zadnjem koraku posjećujemo njegovog susjeda vrh 1 koji je već obojan u sivo pa ciklus postoji. Primijetimo da ovaj algoritam samo prepoznaje postoji li ciklus, ali ne i koji su vrhovi dio istoga. Algoritmi koje ćemo opisati u nastavku opisani su u članku [6] i promatraju povezane grafove u kojima svaki vrh ima najviše jedan izlazni brid i podrazumijeva se da stanu ako dođu do nekog vrha v takvog da $f(v)$ nije definiran te u tom slučaju kao izlaz vraćaju poruku da ciklus ne postoji.

Motivacija

Vezane liste imaju široku primjenu u računarstvu. Pomoću njih implementiraju se strukture stoga i reda, a pomoću njih možemo reprezentirati i usmjerene grafove. Uz sve to, jako su korisna struktura podataka kada moramo raditi veći broj brisanja i dodavanja podataka, a manji broj traženja po podacima. Struktura se sastoji od čvorova od koji svaki ima pokazivač na idući čvor.

Slika 2.3: Primjer vezane liste

Upravo zbog njihove široke upotrebe moramo osigurati ispravno kretanje po istima. Najčešći način kretanja po vezanoj listi dan je u pseudokodu u nastavku:

Algoritam 1 Vezana lista

Ulazni podaci: vezana lista v

```
1: a ← v.head
2: while a.next ≠ null do
3:     a ← a.next
4:     ...                                  ▷ code that depends on given task
5: end while.
```

U slučaju da postoji ciklus u danoj vezanoj listi, petlja *while* postat će beskonačna, što stvara prepreku daljnjem izvođenju programa. Stoga su algoritmi koje smo predstavili u ovom poglavlju korisni kako bismo izbjegli taj problem.

2.2 Floydov algoritam

1968. Robert W. Floyd predstavio je prvi algoritam za detekciju ciklusa poznatog pod nazivom "kornjača i zec". U algoritmu se koriste dva pokazivača koja različitom brzinom prolaze nizom danog pridruživanja koje predstavlja usmjeren graf. Na samom početku algoritma, brži pokazivač postavljamo jedan korak ispred sporijega. Sporiji pokazivač ima korake veličine 1, a brži pokazivač ima korake veličine 2. Algoritam se zaustavlja kada se pokazivači nađu na istom elementu grafa, što će se dogoditi u slučaju postojanja ciklusa u grafu. Formalno, algoritam se zasniva na sljedećoj tvrdnji: ako je zadovoljena jednakost

$$a_n = a_{2n}, \tag{2.3}$$

tada je $2n - n = n$ djeljiv s τ. S obzirom na definiciju pridruživanja f i konačnost skupa $\mathcal{M}$, lako se vidi da će u danom grafu uvijek postojati ciklus, tj. algoritam je fokusiran

na traženje ciklusa, a ne na utvrđivanje postoji li. U slučaju da graf ne sadrži ciklus, u algoritmu će se javiti pogreška jer neki od elemenata $a \in \mathcal{M}$ neće imati pridruženu vrijednost $f(a)$. Dakle, algoritam će se uvijek zaustaviti s izlaznim podatkom $\tau \geq 1$.

Algoritam 2 Floydov algoritam

Ulazni podaci: $f : \mathcal{M} \to \mathcal{M}$**, proizvoljan element** $a_0 \in \mathcal{M}$
Izlazni podatak: duljina ciklusa τ

1: $a \leftarrow a_0$, $b \leftarrow f(a)$
2: **while** $a \neq b$ **do**
3: $\quad a \leftarrow f(a)$, $c \leftarrow f(b)$, $b \leftarrow f(c)$
4: **end while**
5: $b \leftarrow f(a)$, $\tau \leftarrow 1$
6: **while** $a \neq b$ **do** $b \leftarrow f(b)$, $\tau \leftarrow \tau + 1$
7: **end while**

Uz pretpostavku da vrijedi (2.3), slijedi da je $2n = n + k\tau$, tj. $n = k\tau$ za neki prirodan broj k. Zaključujemo da mora vrijediti $k\tau \geq \lambda$ jer inače a_n ne bi bio element ciklusa. Dakle, $k \geq \frac{\lambda}{\tau}$, a najmanji takav k je upravo $k = \left\lceil \frac{\lambda}{\tau} \right\rceil$. Zaključujemo da je najmanji n takav da vrijedi (2.3)

$$\tau \left\lceil \frac{\lambda}{\tau} \right\rceil. \tag{2.4}$$

Vremenskoj složenosti algoritma, osim dijela za traženje najmanjeg n dodajemo i τ za traženje duljine ciklusa u drugoj *while* petlji. Vremenska složenost će stoga biti

$$\tau(3 \left\lceil \frac{\lambda}{\tau} \right\rceil + 1). \tag{2.5}$$

Prostorna je složenost očito $O(3) = O(1)$ jer algoritam pohranjuje vrijednosti samo 3 varijable, a, b i c, za svaki broj vrhova m, što je značajno bolja složenost od složenosti jednostavnog algoritma opisanog u prethodnom potpoglavlju koji pohranjuje cijeli graf, tj. prostorne je složenosti $O(m)$.

Primjer 2.2.1. *Pogledajmo primjenu Floydovog algoritma na grafu koji je dan pridruživanjem u sljedećoj tablici.*

x	$f(x)$
0	1
1	3
2	3
3	4
4	5
5	6
6	4

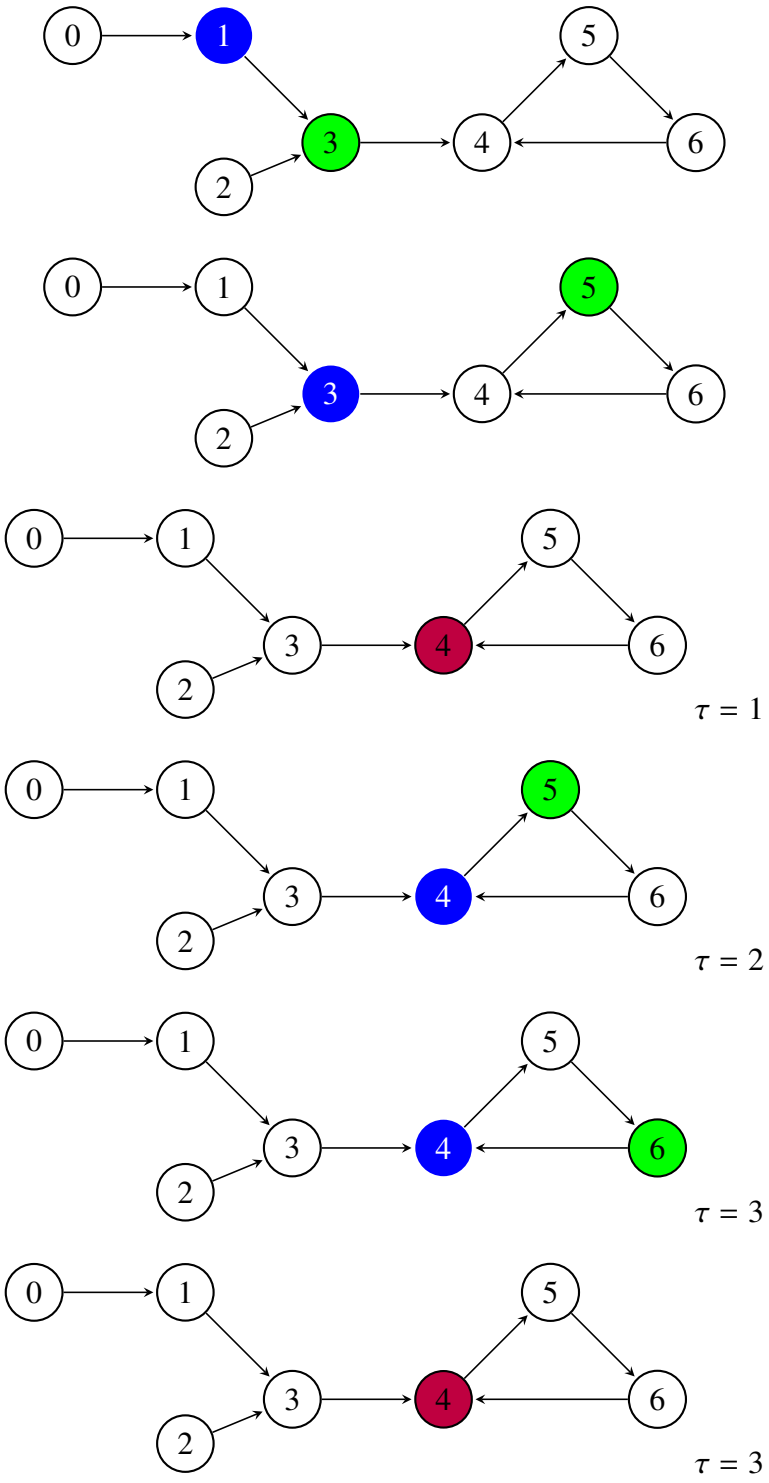
Tablica 2.2: Vrijednosti funkcije f za odabrani graf

Promatramo varijable iz algoritma: a (spori pokazivač), b (brzi pokazivač) i c (pomoćna varijabla). Osim tih varijabli, u tablici se nalazi i varijabla i koja označava broj trenutnog koraka. Za proizvoljni element $a_0 \in \mathcal{M}$ *odaberimo* $a_0 = 1$. *Pogledajmo prikaz izvođenja algoritma po koracima s danim pridruživanjem* f *i proizvoljnim elementom* a_0.

i	a	b	c	τ
0	1	3		
1	3	5	4	
2	4	4	6	
3		5		1
4		6		2
5		4		3

Tablica 2.3: Izvođenje Floyda na grafu danom pridruživanjem u (2.2)

Vidimo da se u nultom koraku inicijaliziraju varijable a i b. U koraku 2 brzi i spori pokazivač se nađu na istom elementu u grafu i tada algoritam izlazi iz petlje u liniji 2 koda (2) *te se pokazivač b pomiče za jedno mjesto, a* τ *inicijalizira na 1. U posljednja se 3 koraka traži veličina ciklusa, tj. algoritam se odvija dok b ne postane jednak a, inkrementirajući u svakom koraku* τ *za jedan. Pogledajmo sada grafički prikaz izvođenja algoritma iz tablice* (2.3). *Zelena boja označava brzi pokazivač b, plava spori pokazivač a, a ljubičasta označava trenutak kad su a i b na istom položaju.*



Slika 2.4: Primjena Brentovog algoritma na grafu iz tablice (2.2).

2.3 Brentov algoritam

Algoritam Richarda P. Brenta, 1969. predstavio je Donald Knuth. Poput Floydovog, koriste se dva pokazivača, ali u modificiranoj verziji. Brži pokazivač kreće se po indeksima grafa koji su potencija broja dva te za svaki idući položaj, sporiji pokazivač postavimo na prethodni položaj bržeg. Algoritam se odvija sve dok pokazivači ne postanu jednaki, tj. dok se ne nađu na istom elementu grafa. Naposljetku računamo duljinu ciklusa kao u Floydovom algoritmu.

Algoritam 3 Brentov algoritam

Ulazni podaci: $f : \mathcal{M} \to \mathcal{M}$**, proizvoljan element** $a_0 \in \mathcal{M}$
Izlazni podatak: duljina ciklusa τ

1: $c \leftarrow a_0$, $a \leftarrow f(a_0)$, $n \leftarrow 1$, $t \leftarrow 1$
2: **if** $a = c$ **then** return $\tau \leftarrow 1$
3: **end if**
4: **if** $n = 2t - 1$ **then** $c = a$, $t = 2t$
5: **end if**
6: $a \leftarrow f(a)$, $n \leftarrow n + 1$
7: **if** $n \geq 3t/4$ **then**
8: **if** $a \neq c$ **then** return to line 4
9: **end if**
10: **end if**
11: $\tau \leftarrow 1$, $a \leftarrow f(c)$
12: **while** $a \neq c$ **do** $a \leftarrow f(a)$, $\tau \leftarrow \tau + 1$
13: **end while**

Kako bismo objasnili složenost algoritma, definiramo funkciju $l : \mathbb{N} \to \mathbb{N}$ na sljedeći način:

$$l(n) = 2^{\lfloor \log_2 n \rfloor}. \tag{2.6}$$

Navedena funkcija pridružuje broju n najveću potenciju broja 2 koja nije veća od samog n. Dakle, vrijedi $l(n) \leq n < 2l(n)$. Definiramo

$$k = \lceil \log_2 \max\{\lambda + 1, \tau\} \rceil,$$

gdje su λ i τ definirani kao u Floydovom algoritmu. Neka je

$$n_0 = 2^k + \tau \left\lceil \frac{l(\lambda) + 1}{\tau} \right\rceil - 1. \tag{2.7}$$

Pokazat ćemo da je n_0 upravo indeks u nizu (2.1) takav da vrijedi $a_{n_0} = a_{2^k - 1}$, gdje je k neki prirodan broj takav da vrijedi $3 \cdot 2^{k-1} \leq n_0 < 2^{k+1}$.

Teorem 2.3.1. *Vrijedi*

(1) $2^k \leq n_0 < 2^{k+1}$

(2) *Za* n_0 *vrijedi* $a_{n_0} = a_{l(n_0)-1}$

(3) $\frac{3}{2}l(n_0) \leq n_0 < 2l(n_0)$.

Dokaz. Dokažimo prvo (1). S obzirom da je $\tau \geq 1$ i $\left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil \geq 1$ jer su brojnik i nazivnik pozitivni pa je i gornje cijelo pozitivan cijeli broj, slijedi da je $\tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 \geq 0$ pa zaključujemo

$$2^k \leq 2^k + \tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 = n_0.$$

Za nejednakost $n_0 < 2^{k+1}$, dokaz ćemo podijeliti na dva slučaja. Za $\tau > l(\lambda)$ slijedi

$$\tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil = \tau,$$

a po definiciji k slijedi $\tau \leq 2^k$. Dakle,

$$n_0 = 2^k + \tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 \leq 2^k + 2^k - 1 = 2 \cdot 2^k - 1 = 2^{k+1} - 1 < 2^{k+1}.$$

U drugom slučaju, za $\tau \leq l(\lambda)$ vrijedi

$$\left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil = \frac{l(\lambda)+1+\delta}{\tau} \leq \frac{l(\lambda)+\tau}{\tau} \leq \frac{2l(\lambda)}{\tau},$$

za neki pozitivan cijeli broj $\delta < \tau$. Slijedi

$$\tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil \leq \tau \frac{2l(\lambda)}{\tau} = 2l(\lambda) = 2 \cdot 2^{\lfloor \log_2 \lambda \rfloor} < 2 \cdot 2^{\lfloor \log_2 \lambda + 1 \rfloor} \leq 2 \cdot 2^k = 2^{k+1}.$$

Time smo dokazali (1).
Za (2) moramo pokazati da je $n_0 - (l(n_0) - 1)$ djeljiv s τ. Po (1) imamo da je

$$l(n_0) = 2^{\lfloor \log_2 n_0 \rfloor} = 2^k.$$

Zaključujemo sljedeće

$$n_0 - (l(n_0) - 1) = 2^k + \tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 - 2^k + 1 = \tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil,$$

a to je kao umnožak broja τ i cijelog broja djeljivo s τ. Time je dokazana i tvrdnja (2). Desna nejednakost u tvrdnji (3) očito slijedi iz jednakosti $l(n_0) = 2^k$. Za dokazati lijevu nejednakost iskoristit ćemo sljedeće dvije nejednakosti:

$$\tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 \geq \tau - 1 = 2^{\lfloor \log_2 \tau \rfloor} - 1 \geq 2^{\lceil \log_2 \tau - 1 \rceil},$$

$$\tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 \geq l(\lambda) = 2^{\lceil \log_2 \lambda + 1 \rceil - 1}.$$

Slijedi da vrijedi

$$\tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 \geq 2^{k-1}$$

pa je

$$n_0 = 2^k + \tau \left\lceil \frac{l(\lambda)+1}{\tau} \right\rceil - 1 \geq 2^k + 2^{k-1} = \frac{3l(n_0)}{2}.$$

Time smo dokazali i tvrdnju (3). □

U danom teoremu dokazano je da je n_0 vrijednost indeksa niza (2.1) koji nam je potreban kako bismo izračunali duljinu ciklusa τ, tj, to je broj koraka u algoritmu potrebnih da dođemo do jednakosti $a_{n_0} = a_{2^k-1}$, za neki k. Zatim u posljednja dva koraka algortima pronađemo duljinu ciklusa τ, ali ne računamo eksplicitno broj $n_0 - 2^k + 1$ koji dijeli τ već samo ocijenimo složenost algoritma pomoću teorema (2.3.1) i broja k na sljedeći način:

$$n_0 \geq \frac{3l(n_0)}{2} = \frac{3}{2} \cdot 2^k \geq \frac{3}{2} max\{\lambda + 1, \tau\}. \tag{2.8}$$

Dakle, složenost Brentovog algoritma je $\Omega(\tau + \frac{3}{2} max\{\lambda + 1, \tau\})$, gdje τ dolazi od posljednje *while* petlje u kojoj računamo duljinu ciklusa, a preostali dio od prvog dijela algoritma u kojem tražimo odgovarajući indeks n_0. Prostorna je složenost, kao u Floydovom algoritmu $O(1)$ jer se za svaki broj vrhova m u algoritmu koriste fiksno četiri varijable.

Primjer 2.3.2. *Pogledajmo u nastavku primjenu Brentovog algoritma na graf definiran produživanjem f iz tablice* (2.2) *s početnim proizvoljnim elementom $a_0 = 0$. Varijabla c predstavlja brzi, a varijabla a spori pokazivač.*

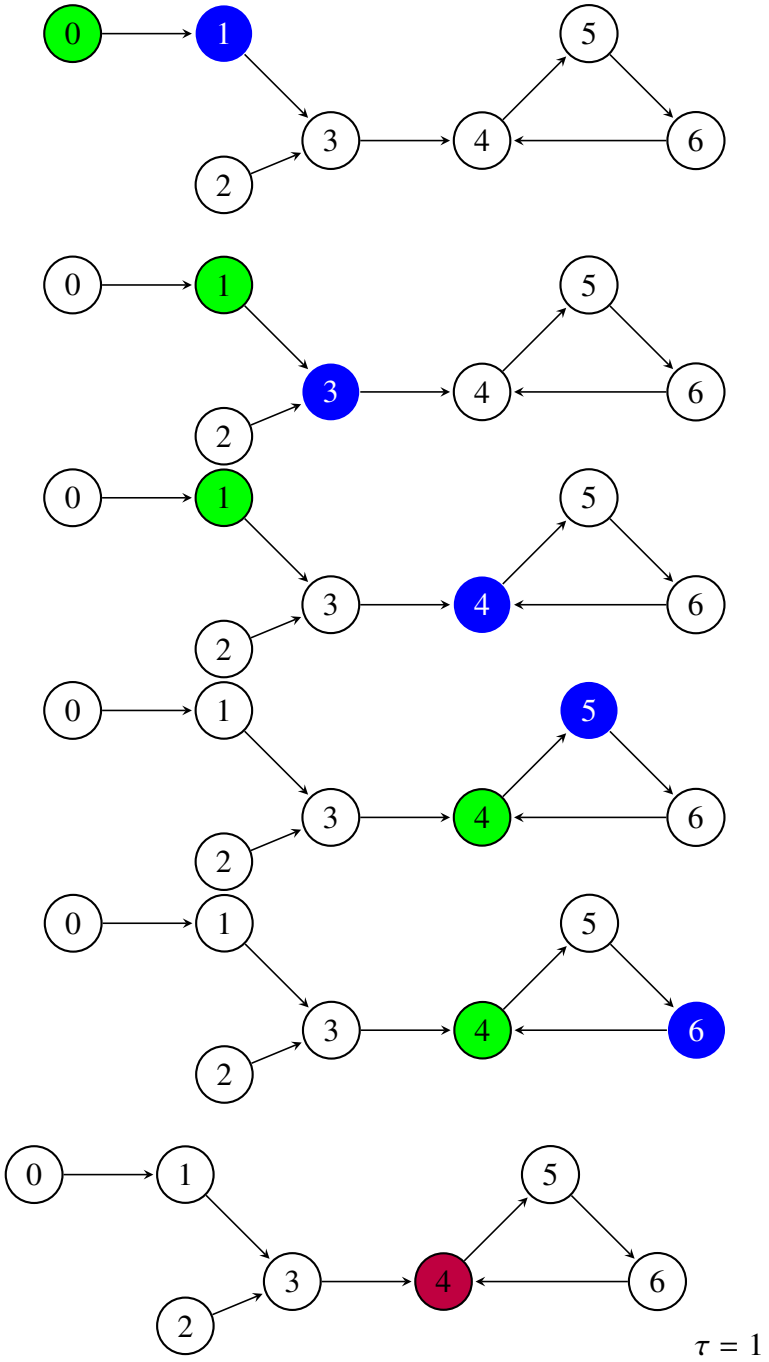
i	a	c	t	n	τ
0	1	0	1	1	
1	3	1	2	2	
2	4	1	2	3	
3	5	4	4	4	
4	6	4	4	5	
5	4	4	4	6	
7	5	4	4	6	1
8	6	4	4	6	2
9	4	4	4	6	3

Tablica 2.4: Izvođenje Brentovog algoritma na grafu iz tablice (2.2)

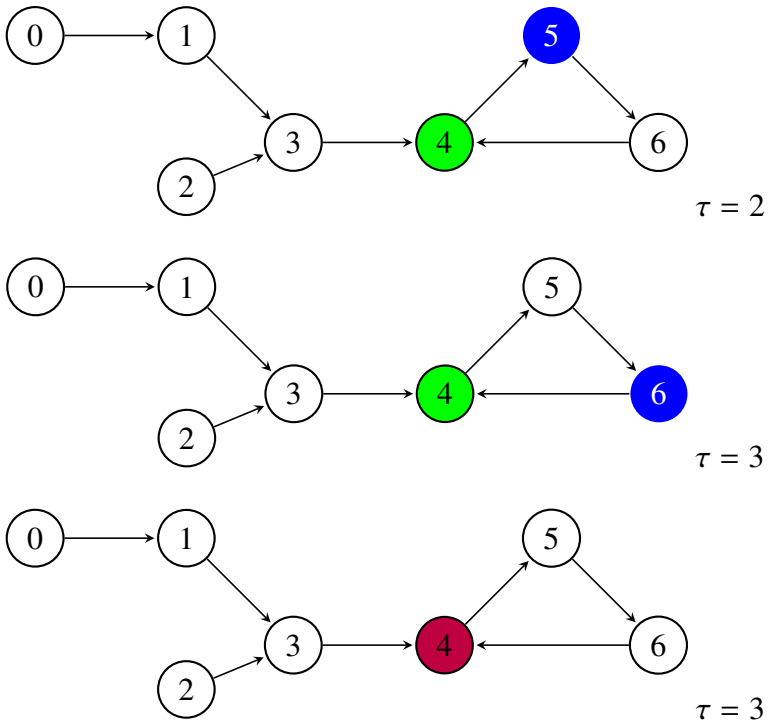
Za dani a_0 i pridruživanje f, niz (2.1) *će biti jednak:*

$$0, 1, 3, 4, 5, 6, 4, 5, 6, \ldots \tag{2.9}$$

Vidimo da brzi pokazivač c poprima vrijednosti a_0, a_1 i a_3, tj. vrijednosti a_{2^k-1}, za $k = 0, 1, 2$, što odgovara opisu algoritma. Za dani primjer vrijedi $\lambda = 3$ i $\tau = 3$ pa po (2.7) *imamo $n_0 = 6$, tj. $a_6 = a_3$ po teoremu* (2.3.1)*, a iz tablice vidimo da se algoritam zaustavio nakon točno 6 koraka upravo na toj jednakosti za $a = a_6$ i $c = a_3$. U nastavku pogledajmo grafički prikaz izvođenja algoritma na danim podacima. Boje su iste kao u Floydovom algoritmu, zelena za brzi pokazivač, plava za spori, a ljubičasta kada su pokazivači na istom položaju.*



Slika 2.5: Primjena Brentovog algoritma na grafu iz tablice (2.2).



Slika 2.6: Nastavak

2.4 Gosperov algoritam

R. W. Gosperov algoritam nije poznat poput prethodno dva opisana i jedan od zapisa o tom algoritmu pojavljuje se u knjizi Henrya S. Warrena [7], gdje je jako detaljno opisan i implementiran u programskom jeziku C, a originalno Gosper o njemu piše u članku [2]. Ideja algoritma je uspoređivati element a_n, za neki n, iz niza (2.1) s elementima skupa $M(n)$ koji je definiran na sljedeći način:

$$M(n) = \{a_{n_0}, a_{n_1}, \ldots, a_{n_m}\}, \qquad n, m > 0, \tag{2.10}$$

gdje su indeksi dani s

$$n_k = \max_{r<n}\{r \mid v_2(r+1) = k\}, \qquad k = 0, 1, \ldots \tag{2.11}$$

Vrijednost $v_2(r+1)$ predstavlja najveću potenciju broj 2 koja dijeli $r+1$. Primjerice, $v_2(5) = 1, \quad v_2(16) = 4$. Očito je da je skup $M(n)$ konačan i da sadrži maksimalno $\lfloor \log_2 n \rfloor +$ 1 elemenata te da će se od skupa $M(n+1)$ razlikovati u samo jednom elementu.

Primjer 2.4.1. *Za $n = 15$ odredimo $M(15)$. Slijedimo definiciju skupa $M(n)$:*

$$n_0 = \max_{r<n}\{r \mid v_2(r+1) = 0\} = \max\{0, 2, 4, 6, 8, 10, 12\} = 12,$$
$$n_1 = \max_{r<n}\{r \mid v_2(r+1) = 1\} = \max\{1, 5, 9, 13\} = 13,$$
$$n_2 = \max_{r<n}\{r \mid v_2(r+1) = 2\} = \max\{3, 11\} = 11,$$
$$n_3 = \max_{r<n}\{r \mid v_2(r+1) = 3\} = \max\{7\} = 7.$$

Za n_4 ne postoji rješenje jer je najmanji broj r takav da je $r+1$ djeljiv s 2^4 jednak 15, a tražimo brojeve strogo manje od 15. Dakle, $M(n) = \{a_{12}, a_{13}, a_{11}, a_7\}$.

Po opisu skupa $M(n)$ očito je da se svaki a_n uspoređuje s nekima od prethodnih elemenata niza (2.1) kako bismo pronašli ponavljanje. Binarna interpretacija algoritma je ta da indekse k promatramo u binarnom obliku i zatim tražimo a_k najbliži a_n, $k < n$, takav da $k+1$ završava s bitom 1, zatim najbliži prethodni element čiji indeks završava jednom nulom pa zatim s dvije nule itd. dok indeks ne postane veći od n. Sljedećim teoremom dokazat ćemo da skup $M(n)$ sadrži element a_r takav da je zadovoljena jednakost $a_n = a_r$, tj. dokazat ćemo ispravnost algoritma.

Teorem 2.4.2. *Neka su λ i τ kao u definiciji* (2.1.1)*. Tada postoji prirodan broj r takav da je $n = r + \tau$ i vrijedi:*

(1) *$a_r \in M(n)$ i vrijedi $a_n = a_r$,*

(2) *$\lambda + \tau \leq n < \lambda + 2\tau$.*

Dokaz. Neka je $k > 0$ takav da je $2^k \leq \tau < 2^{k+1}$. Definiramo prirodne brojeve r i n na sljedeći način:

$$r = 2^k \left\lceil \frac{\lambda+1}{2^k} \right\rceil - 1, \quad n = r + \tau. \tag{2.12}$$

Očito je $r < n$ za svaki $\tau \geq 1$. Zapišimo r u sljedećem obliku:

$$r = 2^k \left\lceil \frac{\lambda+1}{2^k} \right\rceil = 2^k 2^l s = 2^{k+l} s - 1, \tag{2.13}$$

gdje je $l \geq 0$ prirodan broj i $s = 2h + 1$ neparan prirodan broj. Slijedi

$$r = 2^{k+l} s - 1 = 2^{k+l}(2h+1) - 1 = 2^{k+l} - 1 + h2^{k+l+1} \tag{2.14}$$

pa zaključujemo $r \equiv 2^{k+l} - 1 \pmod{2^{k+l+1}}$, tj. $v_2(r+1) = k + l$. Kako bismo dokazali $r \in M(n)$, moramo još pokazati da je r najveći broj manji od n za kojeg vrijedi $v_2(r+1) = k+l$. Vrijedi

$$r + 2 \cdot 2^{k+l} \geq r + 2^{k+1} > r + \tau = n \tag{2.15}$$

pa je $r \in M(n)$. S obzirom da je $r \geq \lambda$ za svaki $k > 0$ zaključujemo $a_r = a_{r+\tau} = a_n$ pa smo time dokazali (1). Dokažimo tvrdnju (2). Vrijedi

$$\lambda = 2^k \left(\frac{\lambda + 1}{2^k} \right) - 1 \leq 2^k \left\lceil \frac{\lambda + 1}{2^k} \right\rceil - 1 = r \tag{2.16}$$

pa zbog $\tau > 0$ slijedi $\lambda + \tau \leq r + \tau = n$. Time je dokazana lijeva nejednakost tvrdnje. Iz

$$r = 2^k \left\lceil \frac{\lambda + 1}{2^k} \right\rceil - 1 < 2^k \left(\frac{\lambda + 1}{2^k} + 1 \right) - 1 = \lambda + 2^k \leq \lambda + \tau \tag{2.17}$$

zaključujemo $n = r + \tau < \lambda + 2\tau$. Time smo dokazali i desnu nejednakost tvrdnje (2). □

Ideja algoritma je da iz ulaznih podataka f i proizvoljnog elementa $a_0 \in \mathcal{M}$ gradimo skupove $M(n)$, za $n > 0$ dok ne pronađemo ciklus. Elemente skupa $M(n)$ pohranjujemo u niz T, tako da je inicijalno $T[0] = a_{n_0}$, gdje je n_0 kao u (2.11), itd. U nastavku slijedi pseudokod algoritma.

Algoritam 4 Gosperov algoritam

Ulazni podaci: $f : \mathcal{M} \to \mathcal{M}$**, proizvoljan element** $a_0 \in \mathcal{M}$
Izlazni podatak: duljina ciklusa τ

1: $a \leftarrow a_0$, $n \leftarrow 1$, $t \leftarrow 1$, $T[0] \leftarrow a_0$
2: $a \leftarrow f(a)$
3: **for** $i \leftarrow 0$ to $t - 1$ **do**
4: **if** $T[i] = a$ **then** return $\tau \leftarrow n - 2^k(1 + 2\lfloor \frac{n-2^k+1}{2^{k+1}} \rfloor) + 1, \quad k = i$
5: **end if**
6: **end for**
7: $n \leftarrow n + 1$ and $k \leftarrow v_2(n)$
8: **if** $k = t$ **then** $t = t + 1$
9: **end if**
10: $T[k] = a$ and return to line 2

Time smo opisali Gosperov algoritam koji kao rezultat vraća duljinu ciklusa τ. Prostorna složenost Gosperovog algoritma je $O(\lceil \log_2 m \rceil + 4) = O(\lceil \log_2 m \rceil)$ jer pohranjuje samo po $\lceil \log_2 m \rceil$ elemenata niza (2.1) te još 4 pomoćne varijable. Vremensku složenost vidimo iz tvrdnje 2 teorema (2.4.2), tj. jednakost $a_r = a_n$, za neki $r < n$, će se postići za neki $n < 2\tau + \lambda$. Dakle, vremenska složenost je $O(2\tau + \lambda)$. Zaključujemo da je Gosperov algoritam vrlo efikasan jer štedi vrijeme i prostor.

Primjer 2.4.3. *U tablici u nastavku pogledajmo kako se u ovisnosti o n mijenjaju varijable a i T u Gosperovom algoritmu za ulazne podatke f iz tablice* (2.2) *i* $a_0 = 0$.

n	a	$T[0]$	$T[1]$	$T[2]$
	0			
1	1	0		
2	3	0	1	
3	4	3	1	
4	5	3	1	4
5	6	5	1	4
6	4	5	6	4

Tablica 2.5: Izvođenje Gosperovog algoritma na grafu iz tablice (2.2)

Za dane f i a_0 niz (2.1) *je dan s* (2.9)*. Sada možemo primijetiti kako vrijednosti u nizu T za svaki n odgovaraju vrijednostima skupa $M(n)$:*

$$\begin{aligned} M(1) &= \{a_0\} = \{0\}, \\ M(2) &= \{a_0, a_1\} = \{0, 1\}, \\ M(3) &= \{a_1, a_2\} = \{1, 3\}, \\ M(4) &= \{a_2, a_1, a_3\} = \{3, 1, 4\}, \\ M(5) &= \{a_4, a_1, a_3\} = \{5, 1, 4\}, \\ M(6) &= \{a_4, a_5, a_3\} = \{5, 6, 4\}. \end{aligned} \tag{2.18}$$

Algoritam se zaustavlja u koraku $n = 6$ jer je zadovoljena jednakost $T[2] = a$ u liniji 4 algoritma (4) *te kao povratnu vrijednost vraća duljinu ciklusa τ.*

2.5 Nivaschov algoritam

Za razliku od prethodno opisanih algoritama koji se baziraju na sličnim idejama, ali svaki od njih ima različit način odabira indeksa elemenata koje ćemo uspoređivati, algoritam Gabriela Nivascha predstavljen 2004. drugačije pristupa problemu detekcije ciklusa što ćemo opisati u nastavku. Kako bismo primijenili algoritam, moramo postaviti restrikciju na skup $\mathcal{M}$ koji promatramo. Naime, skup $\mathcal{M}$ mora biti totalno uređen, tj. na skupu $\mathcal{M}\times\mathcal{M}$ definirano je pridruživanje $h : \mathcal{M}\times\mathcal{M} \to \{-1, 0, 1\}$ tako da za proizvoljne $a, b \in \mathcal{M}$ vrijedi:

$$h(a, b) = \begin{cases} -1, & a < b, \\ 0, & a = b, \\ 1, & a > b. \end{cases} \tag{2.19}$$

Vrijedi $h(a_\lambda, a_k) = -1$, za svaki $k = \lambda + 1, \ldots, \lambda + \tau - 2$ i $h(a_\lambda, a_{\lambda+\tau-1}) = 1$. Kako bismo proveli algoritam potreban nam je niz T, koji za svaki element pohranjuje dvije informacije:

T[i].a je vrijednost nekog elementa u nizu (2.1), a T[i].n je indeks tog elementa u nizu (2.1). Ideja algoritma je koristiti stog s kojeg ćemo dodavati i izbacivati elemente kako bismo na kraju pronašli "najmanji", tj. onaj koji je početni u ciklusu. U danom pseudokodu nije korišten stog već niz koji imitira stog, tj. u trenutku pronalaska manjeg elementa od trenutno posljednjeg u nizu, definiramo novi minimum na sljedećem mjestu u nizu. U nastavku slijedi pseudokod algoritma.

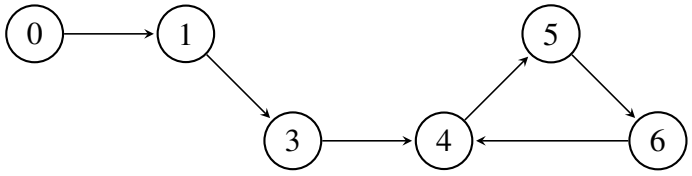
Algoritam 5 Nivaschov algoritam

Ulazni podaci: $f : \mathcal{M} \to \mathcal{M}$**, proizvoljan element** $a_0 \in \mathcal{M}$
Izlazni podatak: duljina ciklusa τ

1: $a \leftarrow a_0, n \leftarrow 0, k \leftarrow 1, T[0] \leftarrow (a_0, 0)$
2: $a \leftarrow f(a), n \leftarrow n + 1, i \leftarrow k$
3: **repeat** $i = i - 1$
4: **until** ($i \geq 0$ and $h(a, T[i].a) = -1$) ▷ uklanjamo elemente veće od a
5: **if** $h(a, T[i].a) = 0$ **then** return $\tau = n - T[i].n$
6: **end if**
7: **if** $h(a, T[i].a) = 1$ **then** define $T[i + 1].a = a, T[i + 1].n = n$ and $k = i + 2$.
8: **end if**
9: return to line 2

Očito će se algoritam zaustaviti nakon drugog pojavljivanja minimalnog elementa jer se nakon njegovog prvog pojavljivanja neće definirati nijedan novi element niza zbog uvjeta u liniji 7 koje će biti lažan za svaki novi element niza (2.1) na koji naiđemo. Stoga je vremenska složenost algoritma $O(2\tau + \lambda)$, točnije, algoritam će se zaustaviti u nekom vremenu $[\lambda + \tau, \lambda + 2\tau\rangle$ zbog gore opisanog pojavljivanja minimalnog elementa. Dakle, kada algoritam ponovo naiđe na minimalan element, uvjet u liniji 5 bit će istinit i algoritam će se zaustaviti s povratnom informacijom o vrijednosti duljine ciklusa τ. Duljina niza T se povećava za $O(\log_2 n)$ kako se n povećava pa možemo zaključiti da je prostorna složenost algoritma logaritamska te ovisi o broju koraka koji će se izvesti u algoritmu.

Primjer 2.5.1. *U nastavku promotrimo izvođenje Nivaschovog algoritma na sljedećem podgrafu grafa definiranog pridruživanjem u tablici* (2.2)*:*



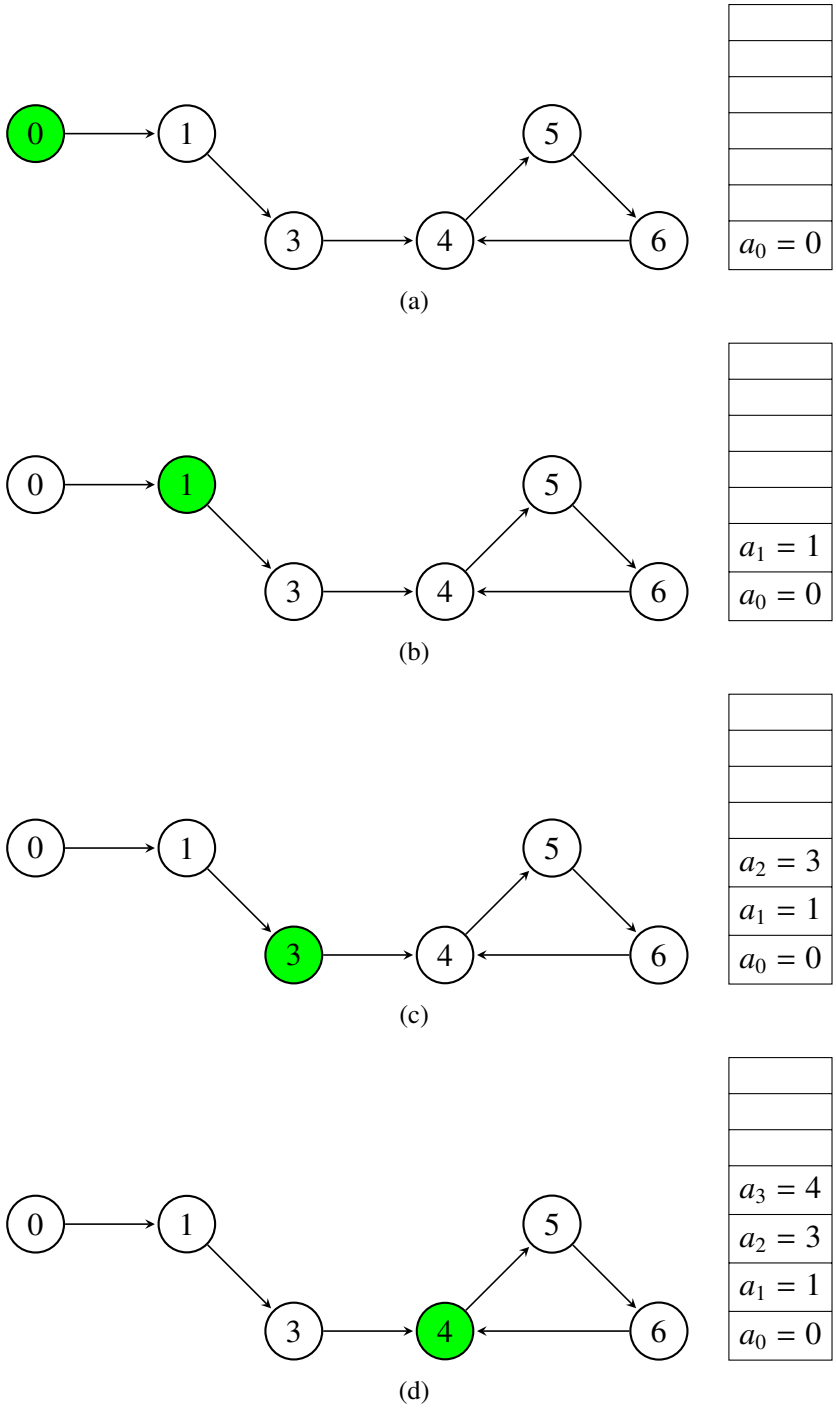
Slika 2.7: Graf

Definiramo pridruživanje $h : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$*, za* $\mathcal{M} = \{0, 1, 3, 4, 5, 6\}$*, tako da za svaki brid* (a, b) *grafa,* $a, b \in \mathcal{M}$*, vrijedi* $a < b$*. U sljedećoj tablici opisano je izvođenje algoritma za dane* f, h *i* $a_0 = 0$*:*

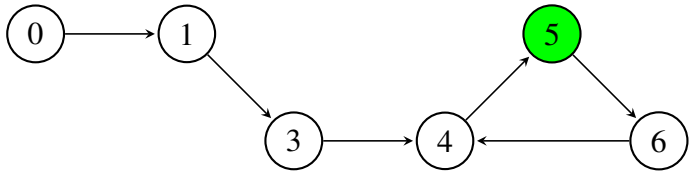
n	a	$T[0]$	$T[1]$	$T[2]$	$T[3]$	$T[4]$	$T[5]$
0	0	(0,0)					
1	1	(0,0)	(1,1)				
2	3	(0,0)	(1,1)	(2,3)			
3	4	(0,0)	(1,1)	(2,3)	(3,4)		
4	5	(0,0)	(1,1)	(2,3)	(3,4)	(4,5)	
5	6	(0,0)	(1,1)	(2,3)	(3,4)	(4,5)	(5,6)
6	4	(0,0)	(1,1)	(2,3)	(3,4)	(4,5)	(5,6)

Tablica 2.6: Izvođenje Nivaschovog algoritma na grafu (2.7)

Algoritam se zaustavio u trenutku $n = 6$ *jer vrijedi* $h(4, T[3].a) = 0$*. Stoga je povratna vrijednost algoritma jednaka* $\tau = n - T[3].n = 6 - 3 = 3$*. Pogledajmo u nastavku grafički prikaz kretanja pokazivača* a *(zelena boja) po grafu i stanje na stogu* T *u svakom koraku algoritma.*

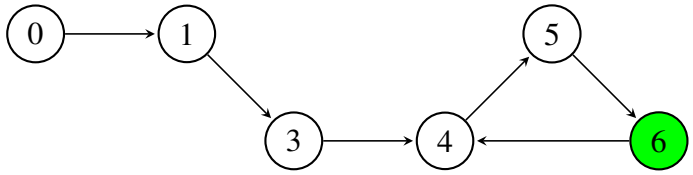


Slika 2.8: Grafički prikaz izvođenja Nivaschovog algoritma



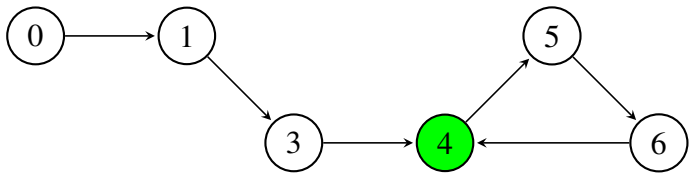
$a_4 = 5$
$a_3 = 4$
$a_2 = 3$
$a_1 = 1$
$a_0 = 0$

(e)



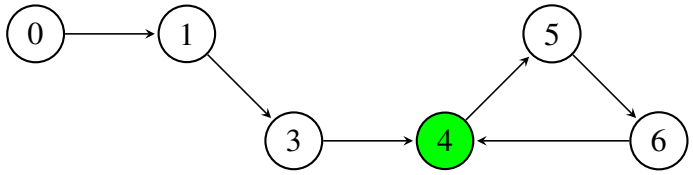
$a_5 = 6$
$a_4 = 5$
$a_3 = 4$
$a_2 = 3$
$a_1 = 1$
$a_0 = 0$

(f)



$a_5 = 6$
$a_4 = 5$
$a_3 = 4$
$a_2 = 3$
$a_1 = 1$
$a_0 = 0$

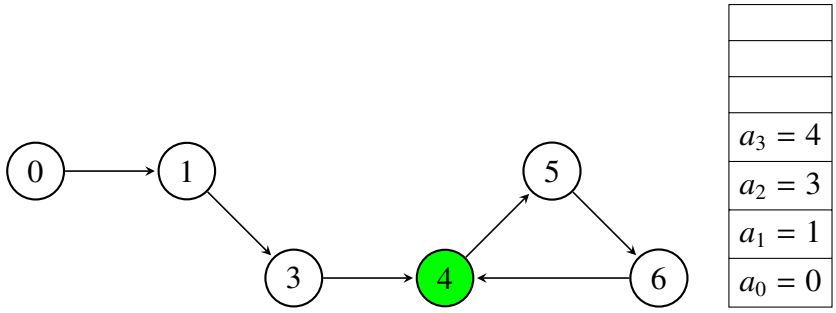
(g) $h(4, a_5) = -1$



$a_4 = 5$
$a_3 = 4$
$a_2 = 3$
$a_1 = 1$
$a_0 = 0$

(h) $h(4, a_4) = -1$

Slika 2.8



(i) $h(4, a_3) = 0$

Slika 2.8

2.6 Usporedba algoritama

U tablici u nastavku vidimo vremenske i prostorne složenosti svih opisanih algoritama.

Algoritam	Vremenska složenost	Prostorna složenost
Floyd	$O(\tau(3\left\lceil\frac{\lambda}{\tau}\right\rceil + 1))$	$O(1)$
Brent	$\Omega(\tau + \frac{3}{2}max\{\lambda + 1, \tau\})$	$O(1)$
Gosper	$O(2\tau + \lambda)$	$O(\lceil\log_2 m\rceil)$
Nivasch	$O(2\tau + \lambda)$	$O(\log_2(2\tau + \lambda))$

Tablica 2.7: Vremenske i prostorne složenosti algoritama

Svi algoritmi su sličnih vremenskih složenosti, ali u prostornoj se složenosti ističu Floydov i Brentov jer složenost ne ovisi ni o veličini grafa ni o poziciji ciklusa u grafu, a ne ovisi ni o samoj duljini ciklusa. Prema danim podacima lako je zaključiti zbog čega je Floydov algoritam najpoznatiji od navedenih. Naime, jednostavan je za implementirati i ima najbolju prostornu složenost.

Poglavlje 3

Problem maksimalne klike u grafu

3.1 Opis problema

Problem pronalaženja maksimalne klike u grafu je problem pronalaženja maksimalnih potpunih podgrafa danog grafa.

Definicija 3.1.1. *Neka je $G = (V, E)$ neusmjeren graf i $S \subseteq V$ podskup skupa vrhova grafa G. Kažemo da je $S \subseteq V$ maksimalna klika grafa G ako vrijedi:*

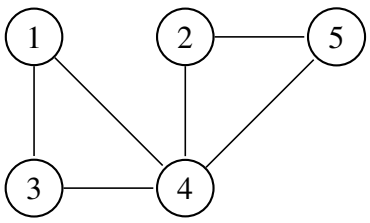
(1) *svaka dva vrha $a, b \in S$ su povezana bridom*

(2) *S nije pravi podskup nijednog skupa vrhova $S_0 \subseteq V$ za kojeg vrijedi (1).*

Problem pronalaženja maksimalne klike ekvivalentan je problemu traženja maksimalnog nezavisnog skupa, tj. komplement maksimalnog nezavisnog skupa vrhova sastojat će se od vrhova koji čine maksimalnu kliku grafa.

Definicija 3.1.2. *Za podskup $S \subset V$ vrhova grafa $G = (V, E)$ reći ćemo da je nezavisan ako nikoja dva vrha $a, b \in S$ nisu susjedna.*

Primjer 3.1.3. *Promotrimo opisane pojmove na sljedećem grafu.*



Slika 3.1: Neusmjeren graf

Maksimalne klike u ovom grafu su skupovi $\{1, 3, 4\}$ *i* $\{2, 4, 5\}$*. S druge strane, nezavisni su skupovi* $\{1, 5\}$*,* $\{1, 2\}$*,* $\{2, 3\}$ *i* $\{3, 5\}$*.*

Sada ćemo, slijedeći [3] i [1], dokazati neke osnovne rezultate o klikama.

Teorem 3.1.4. *Skup vrhova* $S \subseteq V$ *grafa* $G = (V, E)$ *je nezavisan ako i samo ako je* S *klika u grafu* G^C*.*

Dokaz. Neka je $S \subseteq V$ nezavisan skup vrhova. Tada za sve $a, b \in S$ vrijedi $ab \notin E$, a po definiciji komplementa slijedi $ab \in E^C$, za $G^C = (V, E^C)$. Analogno vrijedi za obrat.

□

Teorem 3.1.5. *Neka je* $a \in V$ *vrh grafa* $G = (V, E)$ *i* C *skup svih maksimalnih klika grafa. Za svaku maksimalnu kliku* $C \in C$ *vrijedi točno jedna od sljedeće dvije tvrdnje:*

1. *Maksimalna klika uključuje vrh* a*.*
2. *Maksimalna klika uključuje barem jedan od vrhova iz* D_a*.*

Dokaz. Pretpostavimo da je $C \in C$ klika za koju su obje tvrdnje istinite i neka je $a \in C$. Tada bi podgraf kojeg definira klika C sadržavao brid ab, za neki vrh $b \in D_a$ što je kontradikcija s $b \in D_a$. Pretpostavimo sada da nijedna tvrdnja nije istinita. Tada bi vrijedilo $C_a \subseteq C$ pa bi to bila kontradikcija s činjenicom da je C maksimalna klika jer bismo mogli u taj skup dodati vrh a koji ne bi narušio strukturu klike. □

Lako se vidi da za graf sa slike 3.1 vrijedi tvrdnja prethodnog teorema.

U ovom ćemo poglavlju opisati dva algoritma za pronalaženje svih maksimalnih klika u grafu. Prvi je, ujedno i najpoznatiji, Bron–Kerboschov algoritam predstavljen 1973. od strane dvojice nizozemskih matematičara Coenraada Bron–a i Joepa Kerboscha u članku [4]. Drugi algoritam objavljuje E. A. Akkoyunlu također 1973. u članku [1] te se smatra istim kao Bron–Kerboschov jer oba generiraju isto stablo pretraživanja, ali ipak imaju različite pristupe problemu i opise algoritama. Stoga nam je u ovom poglavlju u cilju opisati oba pristupa i usporediti ih.

3.2 Bron–Kerboschov algoritam

Postoje verzije ovog rekurzivnog algoritma bez i s pivotiranjem. Verzija bez pivotiranja provjerava sve klike, dakle uključujući i one koje nisu maksimalne pa je bolji izbor verzija s pivotiranjem koja štedi vrijeme. Naime, odabirom pivotnog vrha $u \in V$ znamo da će svaka maksimalna klika sadržavati ili u ili neki od vrhova koji mu nisu susjedni što znamo po teoremu 3.1.5. Pogledajmo oba algoritma u nastavku.

Algoritam 6 BronKerbosch1 ▷ bez pivotiranja

Ulazni podaci: $R \subseteq V$, $P \subseteq V$, $X \subseteq V$
Izlazni podatak: maksimalna klika R **ili** $\emptyset$

1: **if** $P = \emptyset = X$ **then** return R
2: **end if**
3: **for each** $v \in P$ **do**
4: BronKerbosch1($R \cup \{v\}$, $P \cap C_v$, $X \cap C_v$)
5: $P = P \setminus \{v\}$
6: $X = X \cup \{v\}$
7: **end for**

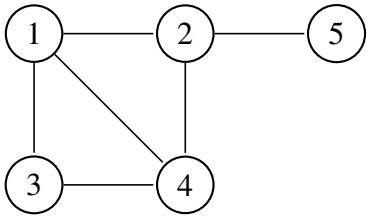
Algoritam 7 BronKerbosch2 ▷ s pivotiranjem

Ulazni podaci: $R \subseteq V$, $P \subseteq V$, $X \subseteq V$
Izlazni podatak: maksimalna klika R **ili** $\emptyset$

1: **if** $P = \emptyset = X$ **then** return R
2: **end if**
3: $u \leftarrow x$, $x \in P \cup X$ proizvoljan ▷ u je pivotni element
4: **for each** $v \in P \setminus C_u$ **do**
5: BronKerbosch2($R \cup \{v\}$, $P \cap C_v$, $X \cap C_v$)
6: $P = P \setminus \{v\}$
7: $X = X \cup \{v\}$
8: **end for**

Kako bismo pronašli sve maksimalne klike za graf $G = (V, E)$ kao ulazne podatke prvog poziva algoritma odabrat ćemo $R = \emptyset = X$ i $P = V$. Skup R predstavlja potencijalnu maksimalnu kliku u koju će se dodavati elementi daljnjim pozivanjima rekurzije. Skup P predstavlja sve vrhove koji su povezani sa svim vrhovima iz R, tj. kandidate za dodavanje u kliku. Skup X predstavlja vrhove koje ne uzimamo u obzir jer bi s trenutnim vrhovima u R vodili definiranju klike koju smo već dobili rekurzijom. Kada su P i X oba prazni, znači da smo pronašli maksimalnu kliku i vrhovi koje sadržava nalaze se u R.

Primjer 3.2.1. *Odredimo sve maksimalne klike sljedećeg grafa koristeći Bron–Kerboschov algoritam s pivotiranjem.*



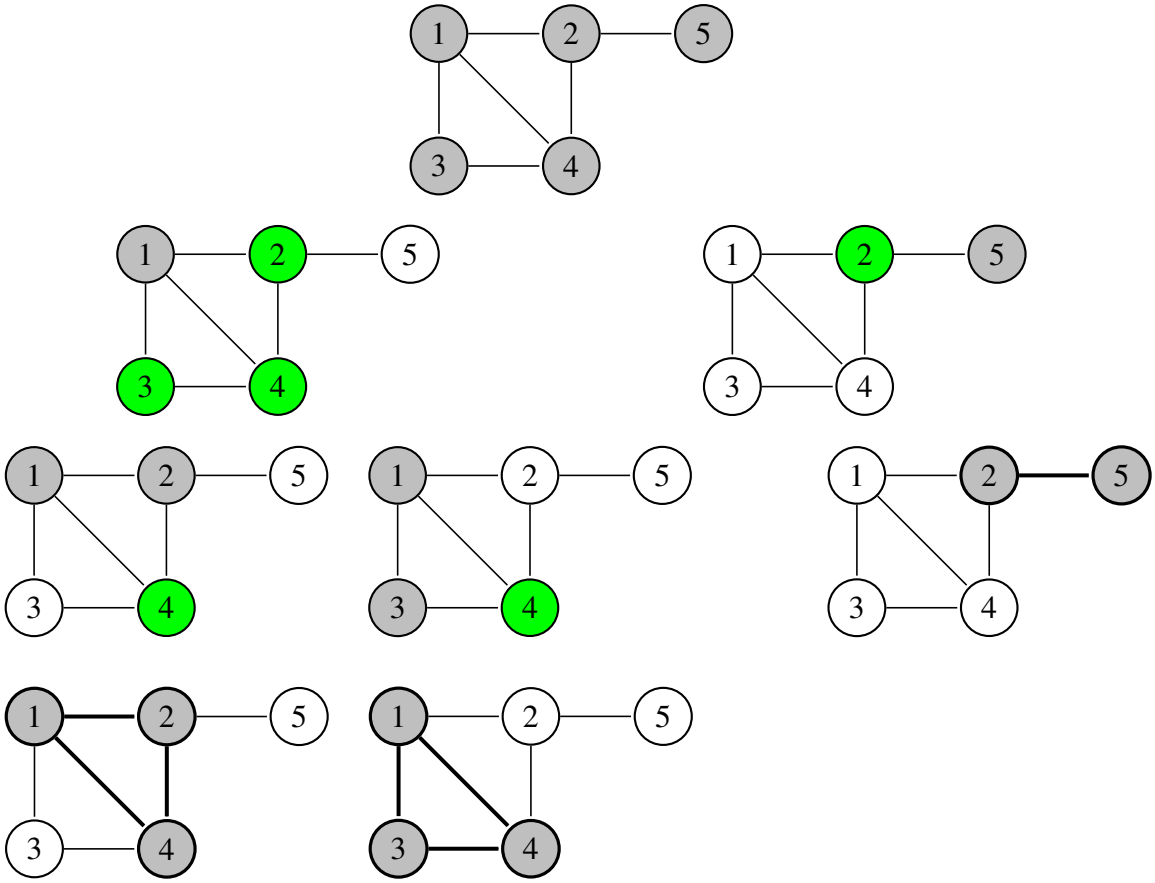
Slika 3.2: Neusmjeren graf

Pogledajmo u tablici u nastavku primjer izvođenja algoritma, tj. redoslijed pozivanja rekurzija.

BronKerbosch2(∅,{1, 2, 3, 4, 5},∅)
BronKerbosch2({1},{2, 3, 4},∅)
BronKerbosch2({1, 2},{4},∅)
BronKerbosch2({1, 2, 4},∅,∅)
BronKerbosch2({1, 3},{4},∅)
BronKerbosch2({1, 3, 4},∅,∅)
BronKerbosch2({5},{2},∅)
BronKerbosch2({5, 2},∅,∅)

Slika 3.3: Rekurzivni pozivi Bron–Kerboschovog algoritma s pivotiranjem na grafu (3.2)

Nakon inicijalnog poziva algoritma, biramo proizvoljan pivotni element $u \in P \cup X = \{1, 2, 3, 4, 5\}$. *U ovom slučaju odabrali smo* $u = 1$. *Tada je* $P \setminus C_u = \{1, 5\}$, *tj. to je skup vrhova s kojima ulazimo u* for *petlju. Za oba vrha će se algoritam rekurzivno pozvati, ponovno biramo pivotne elemente i na taj se način algoritam odvija dok ne pronađe sve maksimalne klike. Za bolje razumijevanje načina na koji opisana rekurzija pronalazi klike, pogledajmo grafički prikaz izvođenja algoritma. Sivom bojom označeni su vrhovi koji su trenutno dio skupa R, zelenom su bojom označeni kandidati za dodavanje u kliku, tj. elementi skupa P, a crvene će boje biti elementi skupa X, ukoliko ih ima.*



Slika 3.4: Izvođenje Bron–Kerboschovog algoritma s pivotiranjem

Vidimo iz slike (3.4) *da algoritam uspješno pronalazi sve maksimalne klike:* $\{1, 2, 4\}$, $\{1, 3, 4\}$, $\{2, 5\}$. *Stablo pretraživanja se podijelilo na dvije glavne grane na temelju odabranog pivotnog elementa* $u = 1$. *Lijeva grana pronalazi sve maksimalne klike koje sadrže vrh* 1, *a desna grana sve koje ga ne sadrže.*

Primjer 3.2.2. *Promotrimo za graf* (3.2) *redoslijed pozivanja rekurzija u algoritmu bez pivotiranja.*

BronKerbosch1(∅,{1, 2, 3, 4, 5},∅)
BronKerbosch1({1},{2, 3, 4},∅)
BronKerbosch1({1, 2},{4},∅)
BronKerbosch1({1, 2, 4},∅,∅)
BronKerbosch1({1, 3},{4},∅)
BronKerbosch1({1, 3, 4},∅,∅)
BronKerbosch1({2},{4, 5},{1})
BronKerbosch1({2, 4},∅,{1})
BronKerbosch1({2, 5},∅,∅)
BronKerbosch1({3},{4},{1})
BronKerbosch1({3, 4},∅,{1})
BronKerbosch1({4},∅,{1, 2, 3})
BronKerbosch1({5},∅,{2})

Slika 3.5: Rekurzivni pozivi algoritma Bron–Kerbosch1 na grafu (3.2)

Dakle, vidimo da algoritam bez pivotiranja ima puno veći broj rekurzivnih poziva i da je zbog toga algoritam s pivotiranjem efikasniji.

Po rezultatima autora Moona i Mosera koji se mogu pronaći u [5], svaki graf s n vrhova ima najviše $3^{\frac{n}{3}}$ maksimalnih klika. Stoga vremenska složenost Bron–Kerboschovog algoritma s pivotiranjem prati tu granicu i rekurzivnih će poziva biti najviše $O(3^{\frac{n}{3}})$.

3.3 Akkoyunluov algoritam

E. A. Akkoyunlu je predstavio algoritam s istim stablom pretraživanja kao i Bron–Kerboschov s pivotiranjem, ali je sam algoritam drugačiji. U nastavku rada ćemo za skup svih maksimalnih klika grafa koristiti oznaku C. Iz teorema 3.1.5 možemo zaključiti da će za svaka dva vrha a i b za koje vrijedi $D_a \subseteq D_b$ vrijediti da je a u svakoj kliki u kojoj se pojavljuje b. Za opisati algoritam potrebni su nam i pojmovi iz sljedeće definicije.

Definicija 3.3.1. *Za podskup $S \subseteq V$ skupa vrhova grafa $G = (V, E)$ definiramo skupove $L(S)$ i $E(S)$ na sljedeći način:*

$$L(S) = \{K \mid K \in C, S \cap K \neq \emptyset\} \tag{3.1}$$

$$E(S) = \{K \mid K \in C, S \subseteq K\}. \tag{3.2}$$

$L(S)$ je skup svih maksimalnih klika koje sadrže barem jedan vrh skupa S, a $E(S)$ skup svih maksimalnih klika koji sadrži sve vrhove iz S. Očito vrijedi $C = L(V)$. Za opisane skupove vrijede jednakosti iz sljedeće napomene.

Napomena 3.3.2. *1.* $L(\emptyset) = \emptyset$,

2. $L(\{x\}) = E(\{x\})$,

3. $L(S_1) \cap L(S_2) = L(S_1)$, *za* $S_1 \subseteq S_2$

4. $E(S_1) \cap E(S_2) = E(S_1 \cup S_2)$,

5. $E(\{x\}) \cap L(S) = \begin{cases} E(\{x\}), & x \in S, \\ E(\{x\}) \cap L(S \cap C_x), & \textit{inače.} \end{cases}$

6. Skup S*, u kojem za bilo koja dva elementa* $x, y \in S$ *vrijedi* $x \in C_y$*, zadovoljava sljedeću jednakost:*

$$E(S) = \begin{cases} \{S\}, & \cap_{x \in S} C_x = \emptyset, \\ E(S) \cap L(\cap_{x \in S} C_x), & \textit{inače.} \end{cases}$$

7. $L(S \cup \{x\}) = E(\{x\}) \cup (L(S) \cap L(D_x))$.

Pokažimo da vrijedi tvrdnja 5. Skup $E(\{x\})$ *uvjetuje da presjek* $E(\{x\}) \cap L(S)$ *sadrži samo maksimalne klike* K *takve da je* $x \in K$*. Ako je* $x \in S$*, skup* $L(S)$ *će po definiciji 3.1 sadržavati sve klike u kojima se nalazi vrh* x*, tj. sve elemente skupa* $E(\{x\})$ *pa jednakost vrijedi. U slučaju* $x \notin S$*, prisjetimo se teorema 3.1.5 koji kaže da svaka maksimalna klika sadrži ili odabrani vrh ili neke od njemu nesusjednih vrhova. Stoga vrh* x *može činiti kliku samo s onim vrhovima iz* S *koji su mu susjedi pa će u skupu* $L(S)$ *biti relevantne samo one klike koje sadržavaju neke od susjeda vrha* x*. Stoga* $L(S)$ *možemo reducirati na* $L(S \cap C_x)$*.*

Promotrimo sada tvrdnju 6. Uz danu pretpostavku zaključujemo da su svi vrhovi iz S *međusobno susjedni, tj. čine jednu kliku. Ukoliko ne postoji vrh* $y \in V \setminus S$ *koji je susjedan svim vrhovima iz* S*, tada slijedi da se klika* S *ne može proširiti nijednim vrhom, tj. maksimalna je i bit će jedina koju čine vrhovi iz* S*. Inače, problem možemo promatrati kao presjek samog* $E(S)$ *i skupa svih maksimalnih klika koje sadrže neki od vrhova susjednih svim vrhovima iz S,* $L(\cap_{x \in S} C_x)$*.*

U tvrdnji 7, skup $L(S \cup \{x\})$ *sadržavat će sve maksimalne klike koje uključuju neki podskup vrhova skupa* $S \cup \{x\}$*. Slijedi da će taj skup sigurno sadržavati sve maksimalne klike koje uključuju vrh* x*, tj. sve klike iz skupa* $E(\{x\})$*. Po teoremu 3.1.5 znamo da će sve maksimalne klike koje ne sadrže* x *sadržavati neke od njemu nesusjednih vrhova pa će ostatak maksimalnih klika sadržanih u* $L(S \cup \{x\})$ *biti sadržan u skupu* $L(S) \cap L(D_x)$

Ideja algoritma je pomoću gornjih jednakosti zapisati skup svih maksimalnih klika kao uniju manjih skupova koje će nam biti jednostavnije pronaći. Ideja algoritma je na stog

stavljati skupove dok ne dobijemo manje skupove u obliku $E(S)$. Na stogu će se nalaziti skupovi jednog od dva sljedeća oblika:

$$E(S') \cap (\cap_{i \in I} L(S_i)) \tag{3.3}$$

$$\cap_{i \in I} L(S_i) \tag{3.4}$$

koje ćemo zatim reducirati dok ne dođemo do željenog oblika. Dani skupovi će se uvijek sastojati od presjeka skupova oblika $L(S)$ i najviše jednog skupa oblika $E(S)$. U nastavku slijedi ideja algoritma.

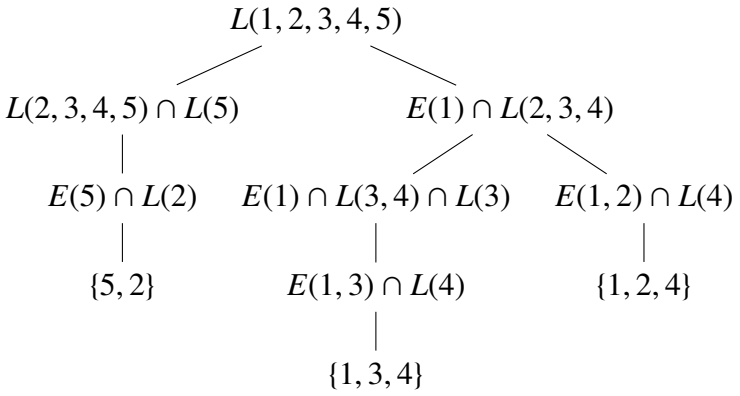
Algoritam 8 Akkoyunluov algoritam

Ulazni podaci: skup vrhova S_0 odabranog grafa, $\{C_x \mid x \in S_0\}$**,** $\{D_x \mid x \in S_0\}$
Izlazni podatak: skup svih maksimalnih klika C

1. $C = \emptyset$.
2. Stavimo $L(S_0)$ na stog.
3. (a) Ako je stog prazan, vrati C kao izlazni podatak i zaustavi algoritam.
   (b) Neka je T izraz s vrha stoga. Ako je oblika (3.3), onda je $V = S'$, a ako je oblika (3.4), onda je $V = \emptyset$.
4. T je oblika 3.3 ili 3.4. Odaberimo iz presjeka jedan od skupova oblika $L(S_k)$ (skup oblika 3.1) tako da odaberemo $k \in I$ takav da S_k ima najmanje elemenata.
5. Odaberimo $x \in S_k$ i neka je $S = S_k \setminus \{x\}$.
6. Ako je $S = \emptyset$ idi na korak 10.
7. Neka je
$$Q = \begin{cases} D_x, & V = \emptyset, \\ D_x \cap \left(\cap_{y \in V} C_y\right), & \text{inače.} \end{cases} \tag{3.5}$$
8. Ako je $Q = \emptyset$ idi na korak 10.
9. Stavi na vrh stoga familiju koja se dobije kada se u T $L(S_k)$ zamijeni s presjekom $L(S) \cap L(Q)$.
10. Neka je $J = \{j \mid j \in I, x \notin S_j\}$.
11. Ako je $J = \emptyset$ idi na korak 15.
12. Za svaki $j \in J$ izračunaj $W_j = S_j \cap C_x$.
13. Ako je $W_j = \emptyset$ za neki $j \in J$, idi na korak 3.
14. Sljedeći izraz stavi na stog:
$$E(V \cup \{x\}) \cap \left(\cap_{j \in J} L(W_j)\right) \tag{3.6}$$
    i idi na korak 3.
15. Izračunaj $P = \cap_{y \in V \cup \{x\}} C_y$.
16. Ako je $P = \emptyset$, dodaj $V \cup \{x\}$ u C i idi na korak 3.
17. Dodaj na stog $E(V \cup \{x\}) \cap L(P)$ i idi na korak 3.

Vidimo da se cijeli algoritam temelji na odvajanju kliki kojima pripada element x i onih kojima pripadaju njemu nesusjedni vrhovi.

Primjer 3.3.3. *Na slici u nastavku vidimo primjer izvođenja Akkoyunluovog algoritma na grafu* (3.2)*.*



Slika 3.6: Akkoyunluov algoritam

Vidimo da se već u prvom koraku odabirom pivotnog elementa $x = 1$ algoritam podijelio na dvije grane, lijevu koja će generirati sve maksimalne klike kojima ne pripada vrh $x = 1$ i desnu u kojima će biti sve maksimalne klike kojima pripada. U narednim se koracima promatraju elementi susjedni vrhovima skupa V iz opisa algoritma u liniji 3 pod točkom (b) i vrhu x te se sukladno tomu dodaju elementi u skup oblika 3.2. Algoritam će pronaći maksimalnu kliku kada je zadovoljen uvjet $P = \emptyset$, gdje je P presjek susjednih elemenata vrhova iz $V \cup \{x\}$, jer tada slijedi da ne postoji vrh koji možemo dodati u $V \cup \{x\}$ koji bi proširio kliku te je stoga ta klika maksimalna.

Bibliografija

[1] Eralp Abdurrahim Akkoyunlu, *The enumeration of maximal cliques of large graphs*, SIAM Journal on Computing **2** (1973), 1–6.

[2] Michael Beeler, R. W. Gosper i Richard Schroeppel, *Hakmem*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, MIT AI Memo 239 (1972).

[3] John Adrian Bondy i Uppaluri Siva Ramachandra Murty, *Graph theory with applications*, sv. 290, Macmillan London, 1976.

[4] Coenraad Bron i Joep Kerbosch, *Finding all cliques of an undirected graph (algorithm 457)*, Commun. ACM **16** (1973), 575–577.

[5] John W Moon i Leo Moser, *On cliques in graphs*, Israel journal of Mathematics **3** (1965), 23–28.

[6] A. Y. E. Nesterenko, *Cycle detection algorithms and their applications*, Fundamentalnaya i prikladnaya matematika **16** (2010), 109–122.

[7] Henry S. Warren, *Hacker's Delight*, 2nd., Addison-Wesley Professional, 2012.

Sažetak

Cilj ovog rada je opisati dio jedne grane matematike sa širokim rasponom primjena, teorije grafova. Preciznije, u radu proučavamo neke algoritme na grafovima koji svoju primjenu pronalaze u računarstvu. Prvo poglavlje rada posvećeno je osnovnim pojmovima u teoriji grafova koji su nam potrebni za daljnje razumijevanje tih algoritama. U drugom poglavlju proučavamo četiri različita algoritma za detekciju ciklusa u grafu. Valja spomenuti kako detekcija ciklusa ima istaknutu ulogu u računarstvu zbog svoje primjene na vezane liste. Opisani su Floydov, Brentov, Gosperov i Nivaschov algoritam, od kojih se najčešće koristi Floydov, poznat pod nazivom algoritam "kornjače i zeca". Treće poglavlje posvećeno je problemu pronalaska svih maksimalnih klika u grafu pomoću rekurzivnih algoritama. Opisan je Bron–Kerboschov algoritam i algoritam autora Akkoyunlua koji generiraju isto stablo pretraživanja, ali imaju drugačije postavljen pristup problemu.

Summary

The goal of this thesis is to describe a part of one branch of mathematics with wide range of applications, the graph theory. More specifically, we study several graph algorithms which are used in computer science. In the first chapter of this thesis we present definitions of the basic graph theory notions which are necessary for understanding of the aforementioned algorithms. In the second chapter we consider four cycle detection algorithms. It is worth noting that cycle detection has a significant role in computer science due to its application to linked lists. The algorithms which are described are due to Floyd, Brent, Gosper and Nivasch, and, among them, the former is the most widely used and also known as ”tortoise and hare” algorithm. The third chapter is devoted to the problem of finding all maximal cliques in graph by using recursions. The algorithms described therein are Bron–Kerbosch's and Akkoyunlu's algorithm which both generate the same search tree but have different approach to problem.

Životopis

Rođena sam 1998. godine u Splitu gdje 2017. završavam srednju školu V. gimnaziju "Vladimir Nazor". Iste godine na Prirodoslovno-matematičkom fakultetu u Splitu upisujem preddiplomski studij Matematika gdje sam 2020. stekla prvostupničku diplomu. U potrazi za novim iskustvima 2020. upisujem diplomski studij Računarstvo i matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu - Matematički odsjek.