UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 2. stopnja

Ines Meršak

NIČELNA PRISILA

Magistrsko delo

Mentor: prof. dr. Sandi Klavžar

Ljubljana, 2019

Zahvala

Rada bi se zahvalila mentorju za hitro odzivnost in dobre komentarje ter vso pomoč med pisanjem diplomske in magistrske naloge.

Najlepša hvala staršem, ki so me vsa leta izobrazbe brezpogojno podpirali in prenašali ter v meni vzbudili interes za znanost. Hvala tudi sestri, ki me je poučevala že pred malo šolo, v osnovni šoli skupaj z mano reševala enačbe, in ker me je prenašala v slabih trenutkih.

Hvala vsem bivšim kolegom, ki so mi polepšali dolge dneve na faksu, in zdajšnjim kolegom, ki me v službi neumorno sprašujejo po datumu zaključka magistrskega študija.

Največja zahvala pa mojemu fantu Juretu, za vse razlage in debate tekom študija, za podporo in boljšo voljo, ko je prihodnost (polna izpitov ali pisanja nalog) izgledala črna, za priložnosti, ki sem jih zaradi njegove spodbude in truda pridobila, pa tudi za natančen pregled tega dela in vztrajanja pri LaTeX-u za Matlabove grafe.

Kazalo

Program dela

Magistrsko delo naj vsebuje poglobljen pregled znanih rezultatov o ničelni prisili grafov. Izdelani in implementirani naj bodo algoritmi za izračun ničelne prisile. Algoritmi naj bodo tudi primerjani glede na čas izvajanja na relativno majhnih grafih.

Osnovna literatura

[2] AIM Minimum Rank – Special Graphs Work Group, *Zero forcing sets and the minimum rank of graphs*, Linear Algebra and its Applications **428** (2008) 1628–1648, doi: 10.1016/j.laa.2007.10.009.

[3] D. Amos in dr., *Upper bounds on the k-forcing number of a graph*, Discrete Applied Mathematics **181** (2015) 1–10, doi: 10.1016/j.dam.2014.08.029.

[1] A. Aazami, *Hardness results and approximation algorithms for some problems on graphs*, doktorska disertacija, University of Waterloo, 2008.

[16] S. Fallat, K. Meagher in B. Yang, *On the complexity of the positive semidefinite zero forcing number*, Linear Algebra and its Applications **491** (2016) 101–122, doi: 10.1016/j.laa.2015.03.011.

Podpis mentorja:

Ničelna prisila

Povzetek

Množica ničelne prisile grafa G je taka podmnožica vozlišč Z, za katero velja: če na začetku pobarvamo vozlišča iz Z, nato pa uporabljamo pravilo za širjenje barve, dokler se dogajajo spremembe, morajo biti na koncu pobarvana vsa vozlišča grafa G. Pri tem je pravilo širjenja barve tako, da pobarvano vozlišče u spremeni barvo soseda v natanko tedaj, ko je ta edini še nepobarvan sosed vozlišča u. Število ničelne prisile grafa G je velikost najmanjše take množice ničelne prisile. Delo obravnava ničelno prisilo nekaterih pogostih družin grafov, zgornje in spodnje meje zanjo in karakterizira grafe, ki te meje dosežejo. Obravnavane so tudi zgornje meje za nekatere produkte grafov in povezave ničelne prisile z nekaterimi drugimi grafovskimi parametri, kot sta npr. dominacijsko število in neodvisnostno število.

V sklopu dela so v C++ implementirani algoritmi za preverjanje, ali je množica res množica ničelne prisile, in za izračun števila ničelne prisile za dani graf G. Slednji so eksponentni, vendar za splošen graf (najverjetneje) ne moremo doseči polinomske časovne zahtevnosti, saj je problem NP-težek. Predstavljeni so tudi nekateri drugi rezultati kompleksnosti za probleme, ki so tesno povezani z ničelno prisilo.

Zero forcing

Abstract

Zero forcing set of graph G is a subset Z of vertices for which the following holds: if initially the vertices from Z are coloured black and we apply the colour change rule repeatedly, then at the end of the process all vertices of G should be black. Here the colour change rule is defined as: a black vertex u forces the change of colour in a white negihbour v, if v is the only white neighbour of u. The zero forcing number of a graph is the size of the smallest zero forcing set. In this work, we list and prove results for some well known families of graphs, lower and upper bounds of the zero forcing number and characterise the graphs for which these bounds are tight. Some upper bounds for various products of graphs and connections to other graph parameters, such as domination and independence number, are also given.

We implement algorithms for checking whether a set is zero forcing and for calculating the zero forcing number of a general graph in C++. The latter algorithms are exponential, however due to NP-hardness of the problem, polynomial time complexity (most likely) cannot be obtained. Some additional complexity results for closely related problems are also listed.

Math. Subj. Class. (2010): 05C99, 68Q25

Ključne besede: ničelna prisila, produkt grafov, računska zahtevnost

Keywords: zero forcing, graph product, computational complexity

1 Uvod

Pojem ničelne prisile je bil prvič predstavljen v [2] leta 2008 kot zgornja meja za problem maksimalnega koranga grafa. Kljub temu, da je problem ničelne prisile (in nekatere njegove različice) NP-težek [16, 1, 8], pa je za določene pogoste razrede grafov število lahko določljivo. V članku [2] s pomočjo ničelne prisile določijo maksimalne korange in minimalne range mnogih znanih grafov in družin, ki prej niso bili znani.

Neodvisno je bila ničelna prisila obravnavana tudi v fiziki za nadzor velikih kvantnih sistemov prek manjšega, lokalnega dela sistema [9]. Ekvivalentni pojem *propagacija* so bolj formalno uvedli v [24] leta 2008, kjer so prav tako postavili vprašanje najmanjše take podmnožice vozlišč, da bo na koncu pobarvan cel graf, problem preučili za nekaj preprostih družin grafov in število določili za uravnotežena n-terna drevesa.

Kasneje se je ničelno prisilo začelo preučevati tudi v povezavi drugimi grafovskimi problemi, kot npr. s povezano dominacijo [3], Grundyjevim dominacijskim številom [7], neodvisnostnim številom [12] in električno dominacijo [15, 5, 1]. Slednja ima še posebej zanimivo motivacijo v nadzoru električnih omrežij, saj želimo cel sistem pokriti s čim manj (dragimi) nadzornimi napravami. Nekateri izmed teh rezultatov so predstavljeni v tem delu v poglavju 7.

Ker je izračun števila ničelne prisile NP-težek problem, hkrati pa je pomembna meja za nekatere druge parametre, je bila motivacija za iskanje tesnejših zgornjih in spodnjih mej velika. V [3] so v ta namen definirali splošnejšo različico problema, *k-prisilo*, s pomočjo katere so v [3, 19] našli boljše zgornje meje, nekatere sicer držijo le za povezane grafe. S spodnjo mejo pa so se v odvisnosti od ožine grafa ukvarjali v [10, 13, 14, 19, 20].

V 2. poglavju preletimo nekatere osnovne definicije teorije grafov, ki jih bomo uporabljali v delu, nato pa definiramo pojem ničelne prisile, si ogledamo nekaj lastnosti, trivialno zgornjo in spodnjo mejo ter motivacijo za imenom "ničelna prisila". V 3. poglavju določimo število ničelne prisile za nekatere razrede grafov, kot so npr. poti, cikli in polni grafi, v 4. poglavju pa karakteriziramo grafe, za katere sta spodnja in zgornja meja iz 2. poglavja tesni.

V 6. poglavju sta zapisana algoritma za preverjanje množice in eksponentni algoritem za izračun števila ničelne prisile, katere hitrost je primerjana na naključno generiranih grafih po modelu Erdős–Rényi, implementacija pa je priložena v prilogi A. Predstavljeni so tudi nekateri rezultati glede računske zahtevnosti tega in tesno povezanih problemov. V 7. poglavju je narejen pregled boljših spodnjih in zgornjih mej za ničelno prisilo nekaterih grafov, predstavljene pa so tudi povezave z nekaterimi drugimi grafovskimi parametri, omenjenimi zgoraj.

2 Osnovne definicije

V tem poglavju najprej navedemo nekaj osnovnih definicij teorije grafov, nato pa definiramo dva temeljna pojma tega dela, množico ničelne prisile in število ničelne prisile, si ogledamo motivacijo za poimenovanje in nekaj primerov.

Graf G je definiran kot urejen par množice vozlišč, označimo jo z $V(G)$, in množice povezav $E(G)$, pri čemer so povezave pari elementov iz $V(G)$. Za potrebe tega dela bo za graf veljalo, da je končen (dovolj je, če je končna množica vozlišč) in ima neprazno množico vozlišč, nima zank (nobena povezava ne sme biti sestavljena samo iz enega vozlišča) in večkratnih povezav (elementi v $E(G)$ se ne ponavljajo). Če so elementi množice povezav urejeni pari, pravimo, da je graf *usmerjen*, v nasprotnem primeru pa *neusmerjen*. Če ni označeno drugače, bomo z besedo graf mislili neusmerjen povezan graf. *Velikost grafa* G je definirana kot število vozlišč tega grafa, označimo jo tudi kot $|G|$ ali $n(G)$. V nadaljevanju bomo, kjer je očitno, za kateri graf gre, množico vozlišč grafa označevali kar z V, množico povezav pa z E. Velikost množice povezav grafa G pa bomo označevali z $m(G)$.

Graf H je *podgraf* grafa G, če velja $V(H) \subseteq V(G)$ in $E(H) \subseteq E(G)$. Graf H je *vpet* podgraf, če velja $V(H) = V(G)$. Graf H je podgraf, *induciran* z množico vozlišč $V_H \subseteq V(G)$, če velja $E(H) = \{\{u, v\} \in E(G) \mid u \in V_H \wedge v \in V_H\}$. Zapišemo tudi $H = G[V_H]$.

Vozlišči u in v sta *sosednji*, če obstaja povezava med njima, torej $\{u, v\} \in E$, pišemo tudi $u \sim_G v$ ali kar $u \sim v$. (Pri pisanju povezav bomo ponavadi izpuščali oklepaje in vejico, namesto $\{u, v\}$ torej pišimo kar uv.) *Soseščina* $N(u)$ vozlišča u je množica tistih vozlišč, ki so mu sosednja, *zaprta soseščina* $N[u]$ poleg sosedov vsebuje še samo vozlišče u, *stopnja vozlišča* v grafu G pa je število njegovih sosedov (oziroma velikost njegove soseščine). Največjo stopnjo vozlišča v grafu G bomo označili z $\Delta(G)$, najmanjšo pa z $\delta(G)$. Kadar je očitno, za kateri graf gre, bomo pisali kar Δ in δ. Če velja $\delta(G) = \Delta(G)$, rečemo, da je graf G *regularen*.

2.1 Produkti grafov

V delu si bomo ogledali tudi rezultate za nekatere produkte grafov, zato jih najprej definirajmo in ilustrirajmo s primeri.

Definicija 2.1. *Kartezični produkt* grafov G in H, označimo z $G \,\square\, H$, je graf, katerega množica vozlišč je kartezični produkt množice vozlišč grafa G in grafa H, sosednji pa sta tisti dve vozlišči (u, v) in (u', v'), $u, u' \in V(G)$, $v, v' \in V(H)$, za kateri velja eden izmed naslednjih pogojev:

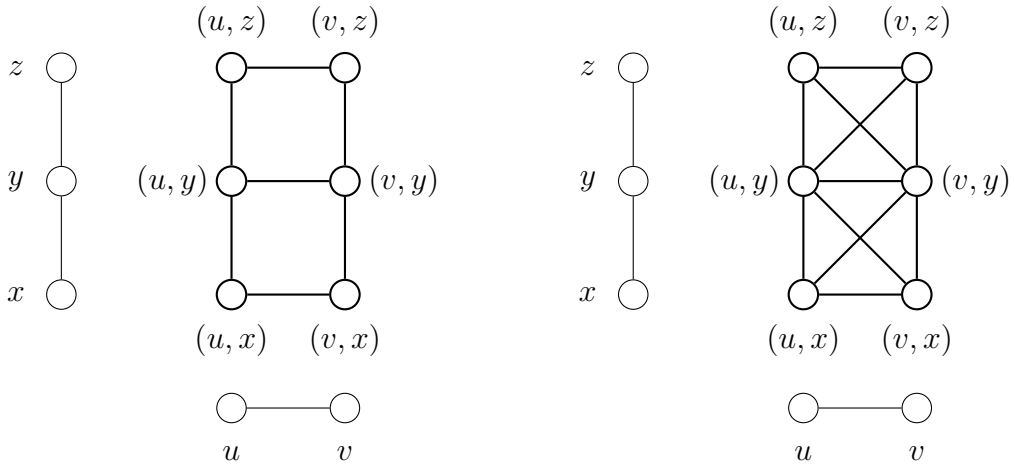
1. $u = u'$ in $vv' \in E(H)$,
2. $v = v'$ in $uu' \in E(G)$.

Definicija 2.2. *Krepki produkt* grafov G in H, označimo z $G \boxtimes H$, je graf, katerega množica vozlišč je kartezični produkt množice vozlišč grafa G in grafa H, sosednji pa sta tisti dve vozlišči (u, v) in (u', v'), $u, u' \in V(G)$, $v, v' \in V(H)$, za kateri velja eden izmed naslednjih pogojev:

1. $u = u'$ in $vv' \in E(H)$,
2. $v = v'$ in $uu' \in E(G)$,
3. $uu' \in E(G)$ in $vv' \in E(H)$.

Iz definicij opazimo, da velja $E(G \square H) \subseteq E(G \boxtimes H)$, kartezični produkt je torej vpeti podgraf krepkega. Poglejmo si nekaj primerov obeh produktov z enostavnimi družinami grafov.

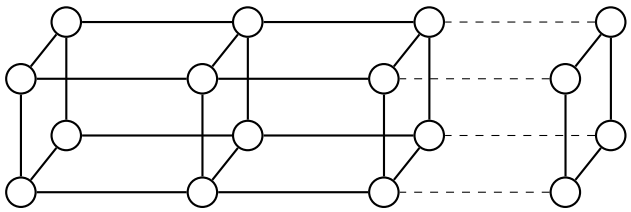
Primer 2.3. Najprej si poglejmo primera kartezičnega produkta poti $P_2 \square P_3$ in krepkega produkta poti $P_2 \boxtimes P_3$, ki sta prikazana na sliki 1.



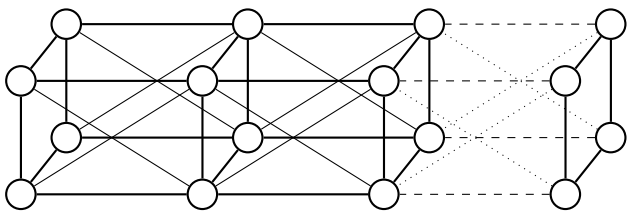
Slika 1: Na sliki sta prikazana kartezični produkt $P_2 \square P_3$ (levo) in krepki produkt $P_2 \boxtimes P_3$ (desno). Kartezični produkt označujemo z $\square$, krepki produkt pa z $\boxtimes$ ravno zaradi rezultata teh dveh produktov poti P_2 same s seboj.

Pri obeh produktih opazimo, da kot inducirani podgrafi za ustrezno izbrano podmnožico vozlišč nastopajo tri (ravno $n(P_3)$) kopije grafa P_2 in dve (ravno $n(P_2)$) kopiji grafa P_3.

Na slikah 2 in 3 sta prikazana še primera kartezičnega in krepkega produkta cikla in poti na n vozliščih, natančneje $C_4 \square P_n$ na sliki 2 in $C_4 \boxtimes P_n$ na sliki 3.



Slika 2: Prikazan je kartezični produkt $C_4 \square P_n$.

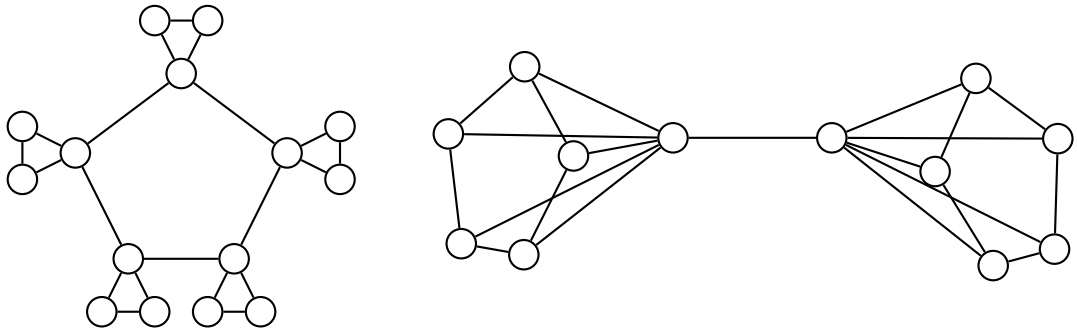


Slika 3: Prikazan je krepki produkt $C_4 \boxtimes P_n$.

Iz zgornjih definicij in primerov lahko vidimo, da sta tako kartezični kot tudi krepki produkt komutativni operaciji do izomorfnosti grafov natančno. Poglejmo si še en produkt grafov, za katerega pa komutativnost v splošnem ne velja.

Definicija 2.4. *Korona* grafov G in H, označimo jo z $G \circ H$, je graf velikosti $n(G)n(H) + n(G)$, ki ga sestavimo tako, da vzamemo eno kopijo G in $n(G)$ kopij grafa H, vsa vozlišča i-te kopije H pa povežemo z i-tim vozliščem v kopiji grafa G.

Na sliki 4 sta prikazani koroni $C_5 \circ P_2$ in $P_2 \circ C_5$.



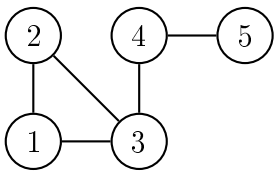
Slika 4: Na sliki sta prikazana korona $C_5 \circ P_2$ (levo) in $P_2 \circ C_5$ (desno). Že iz velikosti lahko vidimo, da grafa nista izomorfna.

2.2 Ničelna prisila

Poglejmo si osnovne definicije in lastnosti pojmov, ki so temelji tega dela ter jih demonstrirajmo na preprostih primerih.

Definicija 2.5. *Množica ničelne prisile* Z grafa G je podmnožica vozlišč tega grafa, takšna, da velja: če na začetku s črno barvo pobarvamo vsa vozlišča iz Z, potem pa sledimo pravilu, da vsako črno vozlišče u z natanko enim belim sosedom v povzroči, da vozlišče v spremeni barvo v črno (označimo $u \to v$), in postopek ponavljamo, dokler se še dogajajo kakšne spremembe, morajo biti po koncu postopka vsa vozlišča grafa G pobarvana črno.

Primer 2.6. Za graf na sliki 5 je množica ničelne prisile na primer $Z = \{1, 5\}$. Pravzaprav od podmnožic vozlišč velikosti 2 zadostujejo tiste, ki vsebujejo vozlišče 1



Slika 5: Ena izmed petih možnih množic ničelne prisile velikosti 2 grafa na sliki je množica $Z = \{1, 5\}$.

ali vozlišče 2 ter hkrati vozlišče 3 ali vozlišče 5, pa tudi podmnožica, ki vsebuje tako vozlišče 1 kot vozlišče 2. (Če se barva začne širiti iz trikotnika, potem morata

biti pobarvani vsaj dve izmed vozlišč trikotnika, če pa se začne širiti iz vozlišča 5, ki je stopnje 1, mora biti pobarvano vsaj še eno vozlišče trikotnika (1 ali 2), da bo vozlišče 3 lahko pobarvalo zadnje belo vozlišče trikotnika.) Vse možne množice ničelne prisile velikosti 2 so torej:

$$Z_1 = \{1,2\}, Z_2 = \{1,3\},\ Z_3 = \{1,5\},\ Z_4 = \{2,3\},\ Z_5 = \{2,5\}.$$

Če gledamo podmnožice vozlišč velikosti 3, pa so množice ničelne prisile vse, razen tista, ki ne vsebuje ne vozlišča 1 ne vozlišča 2, torej množica $\{3,4,5\}$. Množice ničelne prisile velikosti 4 so očitno vse možne podmnožice vozlišč velikosti 4.

Zanima nas, kakšna je najmanjša možna množica vozlišč, ki jo moramo vzeti, da bo po končanem postopku barvanja celoten graf pobarvan črno.

Definicija 2.7. *Število ničelne prisile* $Z(G)$ grafa G je definirano kot velikost najmanjše množice ničelne prisile grafa G, oziroma

$$Z(G) = \min\{|Z| \mid Z \subseteq V(G) \text{ množica ničelne prisile grafa } G\}.$$

Hitro vidimo, da je število ničelne prisile dobro definirano, saj lahko za poljuben graf vedno vzamemo vsa vozlišča, razen enega, in s tem dobimo množico ničelne prisile. Velja torej

$$Z(G) \leq n(G) - 1. \tag{2.1}$$

Po drugi strani seveda velja tudi, da moramo za zadostitev definiciji vedno imeti vsaj eno vozlišče, da lahko govorimo o množici ničelne prisile, očitno velja

$$Z(G) \geq 1. \tag{2.2}$$

Hitro lahko postavimo boljšo očitno spodnjo mejo. Če imamo graf z najmanjšo stopnjo vozlišča δ, mora biti velikost množice ničelne prisile $|Z|$ vsaj δ. V nasprotnem primeru, če bi torej na začetku pobarvali $\delta - 1$ vozlišč ali manj, bi imelo vsako od pobarvanih vozlišč stopnjo vsaj δ, torej bi imelo vsako od pobarvanih vozlišč u vsaj $\delta - (\delta - 1 - 1) = 2$ nepobarvana soseda (ker je u eno izmed pobarvanih vozlišč in ni samo svoj sosed) in se proces širjenja barve sploh ne bi mogel začeti. Velja torej spodnja meja

$$Z(G) \geq \delta. \tag{2.3}$$

Ta spodnja meja je tesna za splošen graf G brez dodatnih omejitev: če vzamemo primer poti P_n, ki ima $\delta = 1$, vidimo, da je tudi $Z(P_n)$ enak 1 (pobarvamo eno od krajišč poti), torej $Z(P_n) = \delta(P_n)$.

Smiselno se je vprašati, ali je ob neki začetni množici pobarvanih vozlišč Z, ki ni nujno množica ničelne prisile, končen rezultat postopka širjenja barve vedno enaka množica, neodvisno od vrstnega reda spreminjanja barve. Pokažimo:

Trditev 2.8 ([2, str. 1633])**.** *Končna množica pobarvanih vozlišč za neko začetno množico Z je vedno ista, ne glede na vrstni red sprememb barv.*

Dokaz. Pokažimo, da vozlišče, ki postane črno za nek vrstni red sprememb barv, lahko vedno postane črno, s pomočjo indukcije na število sprememb barv, ki so potrebne, da pobarvamo vozlišče.

Fiksirajmo Z. Recimo, da za počrnitev vozlišča u potrebujemo eno spremembo barve. To pomeni, da je u edini nepobarvan sosed enega izmed vozlišč v Z (sprememba barve, ki jo potrebujemo, je ravno barva vozlišča u). Ne glede na to, v kakšnem vrstnem redu spreminjamo barve vozlišč, bo u še vedno edini nepobarvan sosed enega izmed vozlišč v Z in bo nekoč pobarvan.

Sedaj naj naša trditev velja za vozlišča, ki potrebujejo $n-1$ sprememb barv, da postanejo črna. Recimo, da za vozlišče u potrebujemo n sprememb barv. To pomeni, da obstaja sosed v, ki potrebuje $n-1$ sprememb barv, da počrni, pri čemer bo vozlišče u po tem edini nepobarvan sosed vozlišča v. Ker smo predpostavili, da trditev drži za vozlišča, ki potrebujejo $n-1$ sprememb, torej velja, da bo vozlišče v postalo črno neodvisno od vrstnega reda sprememb. Po nekem vrstnem redu sprememb imamo torej črno vozlišče v, katerega edini nepobarvan sosed je u, ki bo torej (nekoč) postal črn. Torej tudi za vozlišče u velja, da bo postalo pobarvano neodvisno od vrstnega reda sprememb, in indukcija je zaključena. □

Oglejmo si pojem pokrivanja grafa s potmi, za katerega se bo izkazalo, da je povezan z ničelno prisilo. *Pokritje s potmi* grafa G je taka množica induciranih podgrafov poti, da vsako vozlišče grafa pripada natanko eni izmed poti. Pri tem lahko pokritje vsebuje tudi pot velikosti ena (torej eno samo vozlišče). Velikost najmanjšega pokritja s potmi nekega grafa označimo s $P(G)$. Eno izmed minimalnih pokritij s potmi grafa G na sliki 5 na strani 4 je množica $\{\{1\},\{2,3,4,5\}\}$, velja $P(G)=2$.

Če imamo dano množico ničelne prisile Z nekega grafa, lahko zapišemo zaporedje sprememb barv vozlišč, pri čemer je sprememba oblike $u \to v$ (u povzroči spremembo barve vozlišča v). Zaporedje lahko razbijemo na podzaporedja, tako da vozlišča inducirajo pot; takemu podzaporedju rečemo *veriga sprememb* in ga zapišemo kot $(u_1, u_2, \ldots, u_k)$, pri čemer velja $u_i \to u_{i+1}$ za $1 \leq i \leq k-1$. Ta podzaporedja so disjunktna (glede na vozlišča), saj vsako vozlišče povzroči spremembo v največ enem vozlišču (saj se ta zgodi, ko ima natanko enega belega soseda) in je vsako vozlišče spremenilo barvo zaradi enega vozlišča (ali pa je bilo v začetni množici Z). Povedano drugače, vsako vozlišče se v zaporedju sprememb nahaja največ dvakrat: enkrat na levi strani spremembe, drugič na desni. Poleg tega vemo, da se vsako vozlišče, ki ni v Z, v zaporedju sprememb pojavi vsaj enkrat, saj je Z množica ničelne prisile in mora biti na koncu torej pobarvan cel graf. Število verig je $|Z|$, začetno vozlišče vsake so ravno vozlišča iz Z, pri čemer za verigo sprememb štejemo tudi eno samo vozlišče iz Z, če to ne povzroči spremembe v nobenem vozlišču. Verige so torej particija vozlišč grafa, ki inducira neko pokritje s potmi. Iz tega sledi

Trditev 2.9 ([4, trditev 2.10])**.** *Za velikost minimalnega pokritja splošnega grafa G s potmi $P(G)$ velja*

$$P(G) \leq Z(G). \tag{2.4}$$

Poglejmo si, kako izgleda veriga sprememb na grafu s slike 5 iz primera 2.6, pri čemer vzemimo množico ničelne prisile $Z_3 = \{1,5\}$. Če se dogovorimo, da v primeru večih možnih povzročiteljev spremembe prevlada vozlišče z manjšo oznako, dobimo verigi $5 \to 4 \to 3$ in $1 \to 2$, ki torej inducirata še eno minimalno pokritje tega grafa poleg že omenjenega.

Če imamo dano množico ničelne prisile Z za nek graf G in v Z dodamo še kakšno vozlišče, se izkaže, da je to še vedno množica ničelne prisile grafa G.

Trditev 2.10. *Za graf G je dana množica ničelne prisile Z_1. Potem za $Z_2 \subseteq V$ velja:*

$$Z_1 \subseteq Z_2 \implies Z_2 \text{ je množica ničelne prisile.}$$

Dokaz. Imejmo množico ničelne prisile Z_1 in tako podmnožico Z_2 vozlišč G, da velja $Z_1 \subseteq Z_2$. Naj bo $z_1 = |Z_1|$ in $z_2 = |Z_2|$. Označimo zaporedje sprememb barv vozlišč, če so na začetku pobarvana vozlišča iz Z_1, takole: $u_1 \to v_1, u_2 \to v_2 \ldots$, pri čemer $u_i \to v_i$ pomeni, da je u_i povzročil spremembo barve vozlišča v_i, in i teče od 1 do $n(G) - z_1$, saj je Z_1 množica ničelne prisile in mora biti na koncu pobarvan cel graf. Vozlišča v zaporedju, ki spremenijo barvo, so enolična, torej $v_i \neq v_j$ za $i \neq j$. Če sedaj na začetku pobarvamo vozlišča iz Z_2 in gremo po zaporedju sprememb barv za Z_1, imamo za spremembo $u_i \to v_i$ dve možnosti:

- v_i še ni pobarvan in je edini bel sosed u_i (na začetku smo dodali več črnih vozlišč, torej se je število belih sosedov vozlišč kvečjemu zmanjšalo, ne povečalo), v tem primeru se zgodi sprememba barve kot originalno;
- v_i je že pobarvan, kar se lahko zgodi le, če $v_i \in Z_2 \setminus Z_1$, saj smo sledili spremembam barve za Z_1, kjer se v_i pojavi samo enkrat; sprememba barve se ne zgodi.

Druga možnost se zgodi natanko $(z_2 - z_1)$-krat, toliko sprememb torej "zavržemo", ostale pa se zgodijo kot originalno. Teh je torej $n(G) - z_1 - (z_2 - z_1) = n(G) - z_2$, na začetku smo pobarvali z_2 vozlišč, torej je zdaj skupaj pobarvanih $z_2 + n(G) - z_2 = n(G)$ vozlišč, Z_2 je torej množica ničelne prisile. □

Iz trditve 2.10 vidimo, da za dve množici ničelne prisile Z_1, Z_2 grafa G velja, da je njuna unija $Z_1 \cup Z_2$ tudi množica ničelne prisile za G. To pa ne velja za presek: če vzamemo graf K_2, z vozliščema 1 in 2, sta $Z_1 = \{1\}$ in $Z_2 = \{2\}$ množici ničelne prisile, njun presek, prazna množica, pa očitno ni množica ničelne prisile za K_2.

Velja pa tudi naslednja posledica trditve 2.10:

Posledica 2.11. *Če za graf G ne obstaja množica ničelne prisile velikosti z_1, kjer je $1 < z_1 < n(G) - 1$, potem tudi ne obstaja množica ničelne prisile velikosti $z_2 < z_1$.*

Dokaz. Fiksirajmo z_1 in $z_2 \leq z_1$. Dokažimo, da če ne velja desna stran implikacije, tudi leva ne velja. Naj torej obstaja množica ničelne prisile velikosti z_2. Po trditvi 2.10 ji lahko dodajamo vozlišča grafa G, dokler ne bo velikosti z_1, pa bo še vedno množica ničelne prisile. Sledi torej, da obstaja množica ničelne prisile velikosti z_1 in dokaz je končan. □

Kaj se zgodi z ničelno prisilo grafa, če odstranimo kakšno povezavo ali vozlišče? Če za nek graf G poznamo $Z(G)$, za njegov podgraf v splošnem ne moremo povedati, kakšna bo ničelna prisila. V poglavju 3 npr. dokažemo, da ima cikel C_n ničelno prisilo 2, polni graf K_n pa $n - 1$. V tem primeru (gledamo dovolj velike vrednosti n) za vpeti podgraf C_n velja $Z(C_n) \leq Z(K_n)$. Če pa vzamemo kolo W_n in vpet podgraf zvezdo S_n, pa velja $n - 1 = Z(S_n) \geq Z(W_n) = 3$.

Poglejmo si še, zakaj temu pojmu rečemo "ničelna prisila". Najprej definirajmo graf $\mathcal{G}(A)$ simetrične matrike A velikosti n nad poljem $\mathbb{F}$ (pišemo $A \in Sym_n(\mathbb{F})$):

$$V(\mathcal{G}(A)) = \{1, \dots n\},\ E(\mathcal{G}(A)) = \{\{i,j\} \mid a_{ij} \neq 0,\ 1 \leq i < j \leq n\}.$$

Primer 2.12. Graf, ki pripada matriki

$$A = \begin{bmatrix} 5 & -2 & 3 & 0 & 0 \\ -2 & 13 & 6.9 & 0 & 0 \\ 3 & 6.9 & 0 & 7 & 0 \\ 0 & 0 & 7 & -1.1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

je prikazan na sliki 5 na strani 4. Iz definicije vidimo, da vrednost elementov v matriki (razen neničelnosti) ni pomembna.

Sedaj lahko definiramo množico vseh (simetričnih) matrik grafa G nad poljem $\mathbb{F}$:

$$S(\mathbb{F}, G) = \{A \in Sym_n(\mathbb{F}) \mid \mathcal{G}(A) = G\}.$$

Ideja za imenom ničelne prisile je sledeča. Vzemimo matriko, ki pripada grafu G, in vektor, ki je v jedru te matrike. Črna vozlišča grafa G predstavljajo tiste koordinate v vektorju, za katere zahtevamo, da so enake nič, bela vozlišča pa so tiste koordinate, za katere (še) ni nobene zahteve. Sprememba barve i-tega vozlišča iz bele v črno pomeni, da mora i-ta koordinata vektorja biti nič. Če začnemo z množico ničelne prisile, sledi naslednja trditev

Trditev 2.13 ([2, trditev 2.3])**.** *Imejmo množico ničelne prisile Z grafa G in matriko $A \in S(\mathbb{F}, G)$ grafa G nad poljem $\mathbb{F}$. Naj bo vektor x v jedru matrike A. Če so vse koordinate, ki pripadajo vozliščem iz Z, enake 0, potem je $x = 0$.*

Ker imamo povezavo med grafom in matrikami, lahko definiramo tudi *minimalni rang* in *maksimalni korang* grafa G:

$$\mathrm{mr}^{\mathbb{F}}(G) = \min\{\mathrm{rang}(A) \mid A \in S(\mathbb{F}, G)\},$$
$$M^{\mathbb{F}}(G) = \max\{\mathrm{korang}(A) \mid A \in S(\mathbb{F}, G)\},$$

pri čemer je $\mathrm{korang}(A) = \dim(\ker(A))$. Ker za matriko A velikosti $m \times n$ nad nekim poljem $\mathbb{F}$ velja $\mathrm{rang}(A) + \mathrm{korang}(A) = n$, bosta minimalni rang in maksimalni korang očitno dosežena pri isti matriki (saj če je rang matrike A manjši od ranga matrike istih dimenzij B, mora biti korang matrike A večji od koranga matrike B). Prav tako iz te enakosti očitno sledi, da velja

$$\mathrm{mr}^{\mathbb{F}}(G) + M^{\mathbb{F}}(G) = n(G). \tag{2.5}$$

Če velja $\mathbb{F} = \mathbb{R}$, bomo oznako za polje izpuščali in pisali samo $S(G), \mathrm{mr}(G), M(G)$.

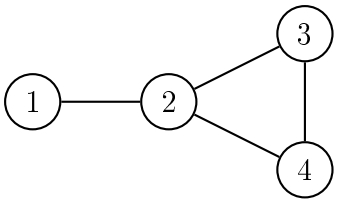
Definiramo lahko tudi *pozitivno semidefiniten minimalni rang* $\mathrm{mr}_+(G)$ grafa G nad $\mathbb{R}$, kjer upoštevamo samo tiste matrike grafa G, ki so pozitivno semidefinitne. Očitno velja $\mathrm{mr}(G) \leq \mathrm{mr}_+(G)$.

S pomočjo kličnega pokritja lahko minimalni rang grafa omejimo od zgoraj. Nek induciran podgraf velikosti n' grafa G je klika, če je izomorfen polnemu grafu na n' vozliščih (vsa vozlišča so sosednja). Množica podgrafov klik, za katero velja, da je vsaka povezava grafa G vsebovana vsaj v eni izmed klik, je *klično pokritje* grafa G. Velikost najmanjše take množice je *število kličnega pokritja* $\mathrm{cc}(G)$ grafa G. Velja [6]:

$$\mathrm{mr}(G) \leq \mathrm{mr}_+(G) \leq \mathrm{cc}(G), \tag{2.6}$$

saj lahko pozitivno semidefinitno matriko grafa dobimo kot vsoto matrik klik iz kličnega pokritja ranga 1 (in vemo, da velja $\mathrm{rang}(A+B) \leq \mathrm{rang}(A) + \mathrm{rang}(B)$). Poglejmo si preprost primer:

Primer 2.14. Naj bo graf G kot na sliki 6. Najmanjše klično pokritje je očitno sestavljeno iz klike velikosti 3 in klike velikosti 2: $\{2,3,4\}, \{1,2\}$. Eno izmed matrik,



Slika 6: Najmanjše klično pokritje grafa na sliki je sestavljeno iz klike velikosti 3 in klike velikosti 2: $\{2,3,4\}, \{1,2\}$.

ki pripada grafu G, lahko zapišemo kot vsoto matrik ranga 1, ki pripadajo tem klikam:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Matriki na desni strani sta očitno ranga 1 (število linearno neodvisnih stolpcev je enako 1), matrika na levi (A) pa je ranga 2 (drugi stolpec je linearna kombinacija prvega in tretjega) in pozitivno semidefinitna (njene lastne vrednosti so $\left\{\frac{1}{2}\left(5+\sqrt{5}\right), \frac{1}{2}\left(5-\sqrt{5}\right), 0, 0\right\}$).

Glavna motivacija za uvedbo ničelne prisile je ravno preučevanje minimalnega ranga (oziroma maksimalnega koranga) grafov. Izkaže se, da je število ničelne prisile zgornja meja za maksimalni korang, za kar potrebujemo še sledeč rezultat:

Trditev 2.15 ([2, trditev 2.2])**.** *Imejmo matriko $A \in \mathbb{F}^{n\times n}$ s korangom* $\mathrm{korang}(A) > k$*. Potem obstaja tak neničelni vektor iz jedra $x \in \ker(A)$, katerega koordinate so enake* 0 *na poljubnih k indeksih.*

Sedaj lahko dokažemo

Izrek 2.16 ([2, trditev 2.4])**.** *Imejmo množico ničelne prisile Z grafa G. Potem za poljubno polje $\mathbb{F}$ velja $M^{\mathbb{F}}(G) \leq |Z|$ in posledično*

$$M^{\mathbb{F}}(G) \leq Z(G). \tag{2.7}$$

Dokaz. S protislovjem: predpostavimo $M^{\mathbb{F}}(G) > |Z|$ in vzemimo tako matriko $A \in S(\mathbb{F}, G)$, da je korang$(A) > |Z|$. Po trditvi 2.15 obstaja tak neničelni vektor $x \in \ker(A)$, katerega koordinate so enake 0 na indeksih, pripadajočih vozliščem iz Z. Po trditvi 2.13 pa sledi, da so potem vse koordinate x enake 0, kar je protislovje z neničelnostjo x. □

3 Ničelna prisila nekaterih razredov grafov

V tem poglavju si ogledamo, kakšno je število ničelne prisile za nekatere osnovne družine grafov, kot so npr. poti, cikli ali polni grafi.

3.1 Poti

Pot dolžine n je graf z n vozlišči, od katerih sta (za $n \geq 2$) dve stopnje 1 (krajišči), preostala vozlišča pa so stopnje 2. Za poti velja:

Trditev 3.1. $Z(P_n) = 1$.

Dokaz. Če na začetku pobarvamo eno izmed krajišč, označimo ga z u, ki je torej stopnje 1, potem ta povzroči spremembo v svojem edinem sosedu v, ki je stopnje 2 in katerega sosed u je že pobarvan, zato v povzroči spremembo barve svojega edinega nepobarvanega soseda in tako dalje ... (Predstavljeno tudi na sliki 7.) Velja torej $Z(P_n) \leq 1$, saj je krajišče množica ničelne prisile. Ker je $\delta = 1$, po trivialni spodnji meji 2.3 velja $Z(P_n) = 1$. □

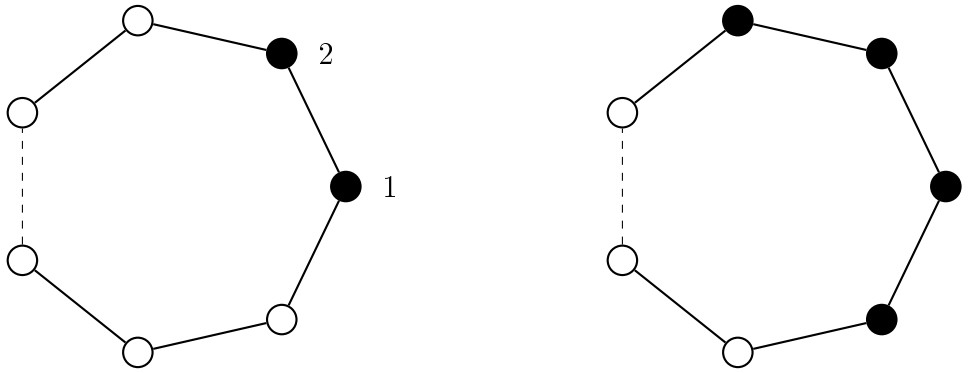
Slika 7: Prikazano je vmesno stanje širjenja črne barve v grafu P_n, če smo na začetku pobarvali samo levo krajišče. Drugo vozlišče z leve ima natanko enega nepobarvanega soseda (na sliki levo) in povzroči njegovo spremembo (na sliki desno). Širjenje se nadaljuje na enak način, dokler ni pobarvan cel graf.

3.2 Cikli

Cikel na $n \geq 3$ vozliščih C_n je graf, v katerem imajo vsa vozlišča stopnjo 2, velja torej $\delta = \Delta = 2$.

Trditev 3.2. $Z(C_n) = 2$.

Dokaz. Če za začetno množico pobarvanih vozlišč vzamemo dva soseda, bo vsak imel natanko enega nepobarvanega soseda, širjenje bo torej potekalo v obe smeri (prikazano na sliki 8), dokler ne bo cel graf pobarvan. Takšna začetna množica torej je množica ničelne prisile, velja $Z(C_n) \leq 2$. Če upoštevamo trivialno spodnjo mejo 2.3 in dejstvo, da je najmanjša stopnja v ciklu enaka 2, dobimo $Z(C_n) = 2$. □



Slika 8: Prikazano je vmesno stanje širjenja barve v grafu C_n, pri čemer sta na začetku pobarvani sosednji vozlišči 1 in 2 (prikazano na sliki levo). Vozlišče 1 lahko povzroči spremembo desnega soseda, vozlišče 2 pa levega soseda (prikazano na sliki desno). Postopek se ponavlja, dokler ni pobarvan cel graf.

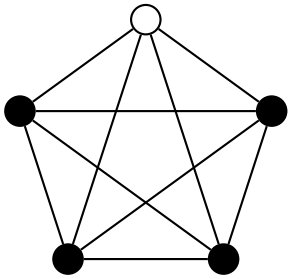
3.3 Polni grafi

Polni graf na n vozliščih K_n je graf, v katerem je vsako vozlišče sosednje vsem ostalim, vsa vozlišča so torej stopnje $n-1$.

Trditev 3.3. $Z(K_n) = n - 1$.

Dokaz. Če pobarvamo vsa vozlišča, razen enega (označimo v), je to množica ničelne prisile, saj so vsi sosedi vozlišča v črni in je v njihov edini nepobarvan sosed, torej spremeni barvo in pobarvan je cel graf. Velja torej $Z(K_n) \leq n - 1$, po spodnji meji 2.3 pa velja tudi $Z(K_n) \geq \delta = n - 1$, torej je $Z(K_n) = n - 1$. □

Na sliki 9 je prikazan polni graf na 5 vozliščih in njegova množica ničelne prisile.

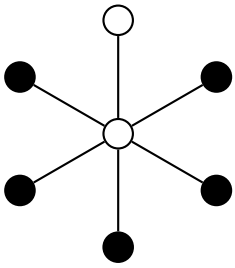


Slika 9: Prikazan je polni graf K_5 in njegova množica ničelne prisile. Čim bi pobarvali eno vozlišče manj, bi imelo vsako črno vozlišče dva nepobarvana soseda in se postopek širjenja barve ne bi začel.

3.4 Zvezde

Zvezda S_n je tak graf na $n+1$ vozliščih, ki ima n vozlišč stopnje 1 in eno vozlišče stopnje n. Je pravzaprav poseben primer polnega dvodelnega grafa, množico vozlišč

lahko torej razbijemo na dve množici: X, v kateri je vozlišče stopnje n, in Y, v kateri so vsa preostala vozlišča, ki so stopnje 1. Znotraj množic X in Y ni povezav, vsako vozlišče iz X pa je sosednje vsakemu v Y. Primer zvezde je prikazan na sliki 10.



Slika 10: Prikazana je zvezda S_6 in njena množica ničelne prisile.

Trditev 3.4. $Z(S_n) = n - 1$.

Dokaz. Za namene tega dokaza si bomo sliko zvezde predstavljali kot poln dvodelen graf $K_{1,n}$ (podobno kot na sliki 11), sredinsko vozlišče stopnje n na eni strani (množica X), na drugi pa vsa preostala vozlišča, ki so stopnje 1 (množica Y). Opazimo, da so vozlišča iz Y ekvivalentna v smislu, da ni važno, katero točno je pobarvano, saj imajo vsa isto stopnjo in istega soseda. Če črno pobarvamo vsa vozlišča v Y razen enega, bo eno izmed teh vozlišč poskrbelo za spremembo barve sredinskega vozlišča stopnje n, to pa bo poskrbelo za spremembo barve edinega vozlišča stopnje 1, ki še ni črno. Velja torej $Z(K_{1,n}) \leq n - 1$. Poskusimo najti manjšo množico ničelne prisile. Na začetku lahko pobarvamo $n - 2$ vozlišča na dva načina: lahko pobarvamo vozlišče stopnje n in $n - 3$ vozlišč stopnje 1, lahko pa pobarvamo $n - 2$ vozlišč stopnje 1. Obravnavajmo oba primera.

- Če na začetku pobarvamo vozlišče stopnje n in $n - 3$ vozlišč stopnje 1, potem črna vozlišča stopnje 1 nimajo več nobenega nepobarvanega soseda, vozlišče stopnje n pa ima še 3 nepobarvane sosede, zato se barva ne more širiti.

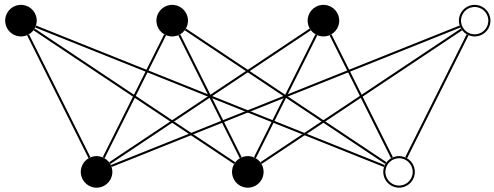
- Če na začetku pobarvamo $n - 2$ vozlišč stopnje 1, ima vsako od njih enega nepobarvanega soseda, to je vozlišče stopnje n, ki torej spremeni barvo. Sedaj ima vozlišče stopnje n še 2 nepobarvana soseda in širjenje barve se ustavi.

Torej ne glede na to, kako izberemo začetno množico velikosti $n-2$, ta ne bo množica ničelne prisile. Velja torej $Z(S_n) = n - 1$. □

3.5 Polni dvodelni grafi

Poln dvodelni graf $K_{m,n}$ je tak graf na $m + n$ vozliščih, kjer lahko množico vozlišč razbijemo na množici V_m velikosti m in V_n velikosti n tako, da znotraj vsake od množic ni povezav, vsako vozlišče iz V_m pa je sosednje vsakemu vozlišču iz V_n. Na sliki 11 je prikazan primer polnega dvodelnega graf $K_{3,4}$.

Trditev 3.5. $Z(K_{m,n}) = m + n - 2$.



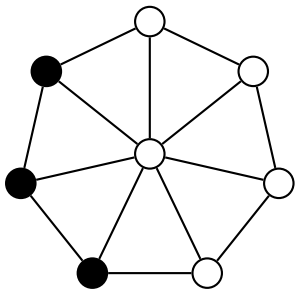
Slika 11: Prikazan je polni dvodelni graf $K_{3,4}$ in njegova množica ničelne prisile.

Dokaz. Če je $m = 1$, dobimo ravno zvezdo, $Z(K_{1,n}) = 1 + n - 2 = n - 1 = Z(S_n)$, trditev torej velja.

Zdaj si poglejmo splošen primer, torej število ničelne prisile $K_{m,n}$. Če na začetku pobarvamo $m - 1$ vozlišč iz V_m in $n - 1$ vozlišč iz V_n, lahko eno izmed pobarvanih vozlišč v V_m povzroči spremembo barve v edinem nepobarvanem vozlišču v V_n in obratno. Velja torej $Z(K_{m,n}) \leq (m-1)+(n-1) = m+n-2$. Sedaj poskusimo najti $m + n - 3$ vozlišč, ki so množica ničelne prisile. Podobno kot v posebenem primeru zvezde jih lahko izberemo tako, da pobarvamo vsa vozlišča iz V_n in $m-3$ vozlišč iz V_m, v tem primeru pobarvana vozlišča iz V_m nimajo nobenega nepobarvanega soseda, vozlišča iz V_n pa imajo tri nepobarvane sosede, zato se barva ne more razširiti. Simetrično za primer, ko pobarvamo vsa vozlišča iz V_m in $n - 3$ vozlišč iz V_n. Če na začetku pobarvamo $m - 1$ vozlišč iz V_m in $n - 2$ vozlišč iz V_n, potem imajo črna vozlišča iz V_n enega nepobarvanega soseda, ki torej spremeni barvo, pobarvana vozlišča iz V_m pa imajo 2 nepobarvana soseda, torej se širjenje ustavi. Simetrično, če na začetku pobarvamo $m - 2$ vozlišč iz V_m in $n - 1$ vozlišč iz V_n. Ne glede na to, kako izberemo vozlišča, torej ne dobimo množice ničelne prisile. Velja torej $Z(K_{m,n}) \geq m + n - 2$, trditev sledi. □

3.6 Kolesa

Kolo W_n je graf na $n + 1$ vozliščih, ki ga dobimo, če vzamemo cikel C_n, na sredino postavimo še eno vozlišče in ga povežemo z vsemi ostalimi. Najmanjša stopnja v grafu je torej $\delta = 3$, največja pa $\Delta = n$. Kolo W_7 je prikazano na sliki 12.



Slika 12: Na sliki je prikazano kolo W_7 in njegova množica ničelne prisile.

Trditev 3.6. $Z(W_n) = 3$.

Dokaz. Če na začetku pobarvamo enega izmed vozlišč stopnje 3, označimo ga z u, in njegova soseda, označimo ju z v in w, potem ima u natanko enega nepobarvanega soseda, to je sredinsko vozlišče, ki torej postane črno. Sedaj ima v natanko enega

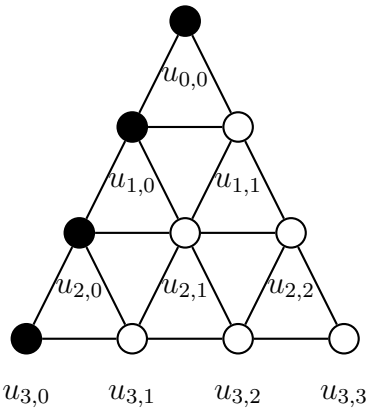
nepobarvanega soseda x, saj sta sredinsko vozlišče in u že pobarvana. Tudi x ima nato natanko enega nepobarvanega soseda, saj sta sredinsko vozlišče in v že črna. Postopek se nadaljuje, podobno se dogaja tudi z druge strani (z vozliščem w in njegovimi sosedi), na koncu so pobarvana vsa vozlišča. Velja torej $Z(W_n) \leq 3$, po trivialni spodnji meji 2.3 pa velja tudi $Z(W_n) \geq \delta = 3$, trditev sledi. □

3.7 Trikotniki

Trikotnik T_n je graf z $\frac{n(n+1)}{2}$ vozlišči. Najlažje ga definiramo rekurzivno: T_1 je graf na enem vozlišču (tudi K_1), graf T_n pa iz grafa T_{n-1} dobimo tako, da pod sliko grafa T_{n-1} dodamo n vozlišč v ravni vrsti, jih povežemo v pot, nato pa vsako vozlišče povežemo z vozliščema iz prejšnje ravni, ki imata enak indeks ali indeks, ki je za ena manjši. Zapišimo matematično: če j-to vozlišče na i-ti ravni označimo z $u_{i,j}$, potem za graf T_n velja:

$$\begin{aligned} V(T_n) &= \{u_{i,j} \mid i = 0, \ldots, n-1, \quad j = 0, \ldots, i\} \\ E(T_n) &= \{\{u_{i,j}, u_{i,j+1}\} \mid i = 1, \ldots, n-1, \quad j = 0, \ldots, i-1\} \\ &\quad \cup \{\{u_{i-1,j}, u_{i,j}\} \mid i = 1, \ldots, n-1, \quad j = 0, \ldots, i-1\} \\ &\quad \cup \{\{u_{i-1,j-1}, u_{i,j}\} \mid i = 1, \ldots, n-1, \quad j = 1, \ldots, i\} \end{aligned}$$

Za lažjo predstavo je na sliki 13 prikazan primer grafa T_4 skupaj z oznakami vozlišč.



Slika 13: Prikazan je trikotnik T_4 in ena izmed njegovih množic ničelne prisile.

Trditev 3.7 ([2, trditev 3.2])**.** *$Z(T_n) = n$.*

Dokaz. Na začetku pobarvajmo eno izmed stranic trikotnika (npr. levo, kot na sliki 13, torej vozlišča $u_{i,0}$ za $i = 0, \ldots, n-1$). Vozlišče $u_{0,0}$ ima natanko enega nepobarvanega soseda, to je diagonalni sosed $u_{1,1}$, ki torej postane črn. Nato ima $u_{1,0}$ natanko enega nepobarvanega soseda, to je $u_{2,1}$, ki spremeni barvo, in zato ima $u_{2,0}$ natanko enega nepobarvanega soseda in tako naprej ... Tako se barva razširi po celi diagonali, pobarvana so torej tudi vsa vozlišča $u_{i,1}$ za $i = 1, \ldots, n-1$. Sedaj ima $u_{1,1}$ natanko enega nepobarvanega soseda, to je $u_{2,2}$, nato ima $u_{2,1}$ natanko

enega nepobarvanega soseda $u_{3,2}$, barva se ponovno razširi po celi diagonali $u_{i,2}$ za $i = 2, \ldots, n-1$. Postopek se ponavlja, na koncu se barva razširi na vse diagonale $u_{i,j},\ i = j, \ldots, n-1$ za $j = 1, \ldots, n-1$, torej na vsa vozlišča grafa T_n. Velja torej $Z(T_n) \leq n$.

Dokažimo še enakost s pomočjo kličnih pokritij. Eno izmed kličnih pokritij T_n dobimo z vsemi "pokončnimi" podgrafi K_3, teh je

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}.$$

Upoštevamo (2.6) in dobimo $\mathrm{mr}(T_n) \leq \mathrm{cc}(T_n) \leq \frac{n(n-1)}{2}$. Ker po (2.5) velja $\mathrm{mr}(T_n) + M(T_n) = \frac{n(n+1)}{2}$, sledi

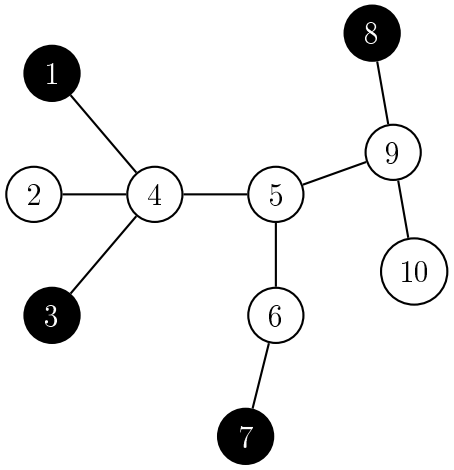
$$M(T_n) \geq \frac{n(n+1) - n(n-1)}{2} = \frac{2n}{2} = n,$$

iz (2.7) in prejšnjega odstavka dokaza pa vemo, da velja tudi $M(T_n) \leq Z(T_n) \leq n$, torej so povsod enakosti in $Z(T_n) = n$. □

3.8 Drevesa

Drevo T je graf, ki ne vsebuje ciklov (kot podgrafov). Za število povezav drevesa velja $m(T) = n(T) - 1$, poleg tega ima vsako drevo z $n(T) \geq 2$ vsaj dva lista, tj. vozlišči stopnje 1. Izkaže se, da je ničelna prisila drevesa T enaka kar velikosti minimalnega pokritja s potmi $P(T)$:

Trditev 3.8 ([2, trditev 4.2]). *$Z(T) = P(T)$.*



Slika 14: Prikazano je drevo T, katerega $P(T)$ je 4 (manj ne more biti, saj imamo eno vozlišče stopnje 4 in eno stopnje 3). Za minimalno pokritje npr. $\{\{2,4,5,6,7\},\{8,9,10\},\{1\},\{3\}\}$ je na sliki prikazana množica ničelne prisile, konstruirana kot v dokazu trditve 3.8.

Dokaz. Iz (2.4) in (2.7) sledi, da velja $P(G) \leq Z(G) \leq M(G)$. Ker za vsako drevo T velja $M(T) = P(T)$ [21], sledijo enakosti v prejšnjem izrazu in posledično $Z(T) = P(T)$.

Poglejmo si še konstruktiven dokaz z indukcijo na $P(T)$. Vzemimo minimalno pokritje s potmi nekega grafa in za vsako pot v pokritju pobarvajmo enega izmed krajišč (konstrukcija je predstavljena na sliki 14). Trdimo, da je to množica ničelne prisile. Če je $P(T) = 1$, je graf T pot, očitno velja $Z(T) = P(T) = 1$ in krajišče poti je množica ničelne prisile.

Predpostavimo, da je tako konstruirana množica res množica ničelne prisile in velja $P(T_0) = Z(T_0)$ za vse take grafe T_0, za katere je $P(T_0) \leq n - 1$. Oglejmo si graf T s $P(T) = n$. Konstruirajmo množico ničelne prisile Z grafa T prek najmanjšega pokritja s potmi kot prej in izberimo neko pot P_1 iz pokritja, ki je s preostankom drevesa T povezana samo prek ene povezave uv, recimo $v \in P_1$. (Mora obstajati, sicer nimamo drevesa.) Če začnemo širjenje barve na črnem krajišču poti P_1, postanejo črna vsa vozlišča P_1 od črnega krajišča do vključno v.

Sedaj lahko preostanek drevesa, ki ga inducirajo vozlišča $T \setminus V(P_1)$, analiziramo neodvisno od poti P_1 (edino vozlišče v, ki je povezano s preostankom grafa, je že pobarvano). Dobimo graf T_1 z minimalnim pokritjem velikosti $n-1$ (ravno pokritje T brez P_1), po indukcijski predpostavki velja, da so črna krajišča preostalih poti v pokritju množica ničelne prisile za T_1, torej postanejo vsa vozlišča iz $T \setminus V(P_1)$, tudi u, črna in se lahko barva razširi še na preostanek poti P_1. Na koncu so torej pobarvana vsa vozlišča, Z je res množica ničelne prisile. Velja $Z(T) \leq P(T)$, iz (2.4) pa sledi še enakost $Z(T) = P(T)$. □

4 Ekstremni primeri

Poglejmo si, za katere grafe je pri zgornji meji (2.1) in trivialni spodnji meji (2.2) za ničelno prisilo dosežena enakost.

Trditev 4.1 ([23, trditev 2.2])**.** *Za graf G na $n > 1$ vozliščih velja:*

$$Z(G) = n - 1 \iff G = K_n.$$

Z besedami, enakost v zgornji meji (2.1) *je dosežena natanko takrat, ko imamo polni graf na n vozliščih.*

Dokaz. Po trditvi 3.3 vemo, da velja implikacija v levo, torej

$$G = K_n \implies Z(G) = n - 1.$$

S protislovjem dokažimo še implikacijo v desno, vzemimo torej graf G, za katerega velja $Z(G) = n - 1$. Recimo, da graf G ni polni graf, torej obstaja neko vozlišče u, katerega stopnja je strogo manjša od $n - 1$. Dokažimo najprej, da iz tega sledi, da obstaja vozlišče w, ki je sosed enega izmed sosedov u, ni pa tudi sosed u-ja. Če tako vozlišče w ne bi obstajalo, bi (zaradi predpostavke povezanosti) to pomenilo, da je naš celoten graf sestavljen iz u in njegovih sosedov, je torej graf na $n = \deg(u) + 1$ vozliščih, vozlišče u pa ima stopnjo $\deg(u) = n-1$, kar je v nasprotju s predpostavko.

Množico ničelne prisile lahko sedaj sestavimo tako, da pobarvamo vse sosede u, razen enega, poleg tega pa še vsa preostala vozlišča v grafu, razen vozlišča w. Vozlišče u torej povzroči spremembo barve v edinem nepobarvanem sosedu, nato pa so pobarvana že vsa vozlišča v grafu, razen w, ki spremeni barvo zaradi enega izmed svojih sosedov. Našli smo torej množico ničelne prisile velikosti $n-2$, kar je v protislovju s predpostavko. Sledi, da je $G = K_n$. □

Trditev 4.2 ([23, ugotovitev 2.1]). *Za graf G na $n \geq 1$ vozliščih velja:*

$$Z(G) = 1 \iff G = P_n.$$

Z besedami, enakost v trivialni spodnji meji (2.2) *je dosežena natanko takrat, ko imamo pot na n vozliščih.*

Dokaz. Iz trditve 3.1 vemo, da drži implikacija v levo. Dokažimo še implikacijo v desno, vzemimo graf G na n vozliščih, za katerega velja $Z(G) = 1$. Vzemimo vozlišče u, ki predstavlja množico ničelne prisile. Če je $n = 1$, imamo torej graf P_1, trditev drži. Sicer mora u povzročiti vsaj eno spremembo barve, torej mora imeti natanko enega nepobarvanega soseda v – vozlišče u je torej stopnje 1. Če je $n = 2$, je v prav tako vozlišče stopnje 1, imamo graf P_2 in trditev drži. V nasprotnem primeru mora v povzročiti vsaj eno spremembo barve, torej ima natanko enega nepobarvanega soseda w (poleg u, ki je že črne barve) – v je torej stopnje 2. Če je $n = 3$, je vozlišče w stopnje 1, dobimo torej graf P_3. Sicer se postopek nadaljuje, v vsakem koraku ugotovimo, da mora biti vozlišče, ki je nazadnje spremenilo barvo, stopnje 1, če je pobarvan že ves graf, imamo torej pot na n vozliščih, ali pa stopnje 2, če imamo še kakšno nepobarvano vozlišče. Širjenje barve se nekoč ustavi, saj je n končen. □

Brez dokaza navedimo še naslednji rezultat:

Trditev 4.3 ([23, izrek 2.3]). *Za (ne nujno povezan) graf G velja:*

$$Z(G) = 2 \iff G \text{ je graf dveh paralelnih poti.}$$

Pri tem je G *graf dveh paralelnih poti*, če v njem obstajata dve disjunktni inducirani poti, ki pokrivata vsa vozlišča G in lahko G v ravnini narišemo kot vzporedni poti, med katerima se povezave (narisane kot ravne črte) ne križajo. En primer takega grafa je graf z dvema povezanima komponentama, pri čemer je vsaka pot.

5 Ničelna prisila produktov grafov

V tem poglavju si bomo ogledali nekaj rezultatov o številu ničelne prisile za produkte grafov. Osnova za to poglavje je predvsem [2].

Trditev 5.1. *Za kartezični produkt grafov G in H velja*

$$Z(G \,\square\, H) \leq \min\{Z(G)n(H), Z(H)n(G)\}. \tag{5.1}$$

Dokaz. Po definiciji kartezičnega produkta vemo, da graf $G \Box H$ vsebuje (kot podgrafe) $n(H)$ kopij grafa G. V vsaki kopiji G pobarvamo pripadajoča vozlišča, ki tvorijo množico ničelne prisile v osnovnem grafu G. (Natančneje, če je $U \subseteq V(G)$ množica ničelne prisile osnovnega grafa G, potem v grafu $G \Box H$ vzamemo vsa taka vozlišča, katere prva koordinata je v množici U.) To je množica ničelne prisile celotnega produkta, saj se širjenje barve zgodi v vsaki kopiji neodvisno, ker so po definiciji kartezičnega produkta med kopijami G povezave le med vozlišči z enako prvo komponento (na začetku velja, da če je pobarvano vozlišče v eni kopiji G, potem so pobarvana tudi pripadajoča vozlišča v vseh ostalih kopijah G, ko se barva začne širiti, pa si lahko predstavljamo, da vsa istoležna vozlišča v vseh kopijah G hkrati postanejo črna). Velja torej

$$Z(G \Box H) \leq Z(G)n(H),$$

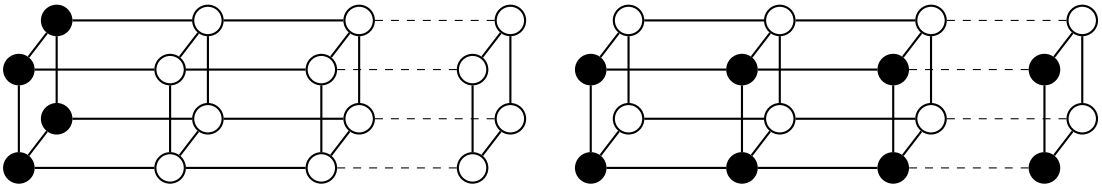
saj imamo ravno $n(H)$ kopij G.

Podobno graf $G \Box H$ vsebuje $n(G)$ kopij grafa H, množico ničelne prisile konstruiramo kot prej in dobimo

$$Z(G \Box H) \leq Z(H)n(G),$$

trditev sledi. □

Na sliki 15 sta prikazani množici ničelne prisile, konstruirani kot v dokazu zgornje trditve, za kartezični produkt $C_4 \Box P_n$.



Slika 15: Prikazan je graf $C_4 \Box P_n$ in množici ničelne prisile, konstruirani kot v dokazu trditve 5.1. Levo je množica ničelne prisile, konstruirana tako, da smo pobarvali pripadajoča vozlišča iz množice ničelne prisile grafa P_n v vseh kopijah P_n, desno pa tako, da smo pobarvali množico ničelne prisile v vsaki kopiji C_4.

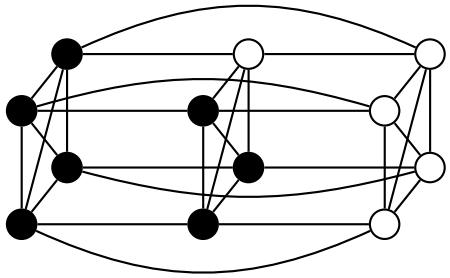
Sledi nekaj trivialnih posledic trditve 5.1.

Posledica 5.2. *Za kartezične produkte splošnega grafa G velja:*

1. $Z(G \Box P_n) \leq \min\{Z(G)n, n(G)\} \overset{n(G) \leq n}{=} n(G)$ *za* $n \geq 1$.
2. $Z(G \Box C_n) \leq \min\{Z(G)n, 2n(G)\}$ *za* $n \geq 3$.
3. $Z(G \Box K_n) \leq \min\{Z(G)n, (n-1)n(G)\}$ *za* $n \geq 2$.

Dokaz. Pri vseh točkah uporabimo zgornjo mejo (5.1), dejstvo, da so velikosti grafov P_n, C_n in K_n enake n in rezultate o njihovih ničelnih prisilah iz poglavja 3. Dodatek pri produktu s potjo P_n sledi iz trivialne spodnje meje $Z(G) \geq 1$ in predpostavke, da je $n(G) \leq n$. □

V posebnem, če imamo kartezični produkt dveh polnih grafov, lahko zgornjo mejo še malo izboljšamo. Po trditvi 5.1 je zgornja meja za ničelno prisilo grafa $K_m \,\square\, K_n$, predpostavimo tudi $2 \leq m \leq n$, enaka $\min\{(m-1)\cdot n, (n-1)\cdot m\} = \min\{mn - n, mn - m\} \overset{m\leq n}{=} n(m-1)$, torej če v vseh n kopijah grafa K_m pobarvamo vsa vozlišča razen enega. Vendar pa imamo še vedno množico ničelne prisile, če pobarvamo vsa vozlišča v eni kopiji K_n in v $m-2$ kopijah vsa, razen enega (istoležnega v vseh kopijah), kar je ravno množica ničelne prisile znotraj te kopije, zadnjo kopijo pa pustimo belo. Za lažjo predstavo je na sliki 16 prikazana tako konstruirana množica ničelne prisile za produkt $K_3 \,\square\, K_4$.



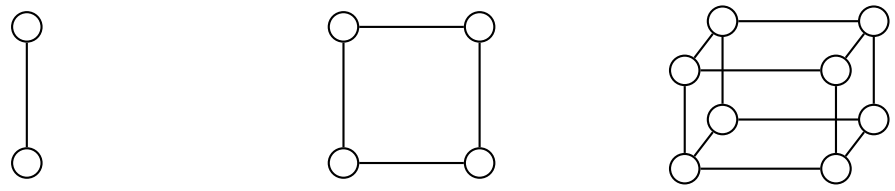
Slika 16: Prikazan je kartezični produkt $K_3 \,\square\, K_4$ in množica ničelne prisile, konstruirana po trditvi 5.3.

Poglejmo, da je to res množica ničelne prisile. Popolnoma pobarvana kopija K_n bo povzročila spremembo barve v vseh vozliščih nepobarvane kopije, razen v tistem vozlišču, ki v večini kopij ni pobarvan. Nato lahko znotraj vsake kopije K_n katerokoli pobarvano vozlišče spremeni barvo edinemu nepobarvanemu vozlišču v tej kopiji. Dokazali smo:

Trditev 5.3. *Za $n, m \geq 2$ velja*

$$Z(K_m \,\square\, K_n) \leq n + (n-1)(m-2) = mn - m - n + 2. \tag{5.2}$$

Zgornja meja (5.1) nam da tudi zgornjo mejo za hiperkocke. *Hiperkocka* velikosti $n+1$ je graf, ki ga dobimo kot kartezični produkt manjše hiperkocke velikosti n in polnega grafa velikosti 2. S simboli $Q_{n+1} = Q_n \,\square\, K_2$, pri čemer je Q_0 enak K_1. Na sliki 17 so prikazane hiperkocke Q_1, Q_2 in Q_3.



Slika 17: Na sliki so prikazane hiperkocke $Q_1 = K_2$ (levo), $Q_2 = K_2 \,\square\, K_2 \cong C_4$ (v sredini) in $Q_3 = Q_2 \,\square\, K_2 = C_4 \,\square\, K_2$ (desno).

Posledica 5.4. *Za $n \geq 1$ velja*

$$Z(Q_n) \leq 2^{n-1}. \tag{5.3}$$

Dokaz. Uporabimo splošno zgornjo mejo za kartezični produkt (5.1) in definicijo hiperkocke. Dobimo

$$Z(Q_n) = Z(Q_{n-1} \,\square\, K_2) \leq \min\{2Z(Q_{n-1}), n(Q_{n-1})\} \leq n(Q_{n-1}) = 2^{n-1},$$

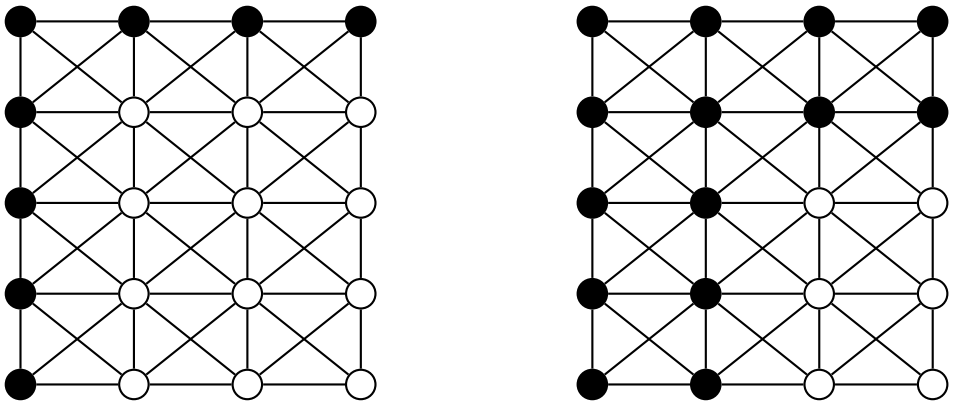
pri čemer upoštevamo še $n(Q_n) = 2^n$. □

V hiperkocki Q_n je torej dovolj, če pobarvamo eno izmed kopij manjše hiperkocke Q_{n-1} (to je očitno množica ničelne prisile, saj ima vsako vozlišče v kopiji natanko enega nepobarvanega soseda, to je istoležno vozlišče v drugi kopiji).

Poglejmo si še zgornjo mejo za krepki produkt poti.

Trditev 5.5. *Za $m, n \geq 1$ velja*

$$Z(P_m \boxtimes P_n) \leq m + n - 1. \tag{5.4}$$



Slika 18: Prikazan je krepki produkt $P_4 \boxtimes P_5$, na sliki levo njegova množica ničelne prisile, na sliki desno pa vmesno stanje širjenja barve.

Dokaz. Pobarvajmo vsa vozlišča ene izmed robnih kopij P_m in ene izmed robnih kopij P_n (prikazano na sliki 18 na primeru $P_4 \boxtimes P_5$), začetno množico pobarvanih vozlišč označimo z Z. Pobarvali smo ravno $m + n - 1$ vozlišč, saj je eno vozlišče, označimo ga z u, skupno obema kopijama. Dokažimo, da je to res množica ničelne prisile. Vozlišče u je edino, ki ima natanko enega nepobarvanega soseda (diagonalnega), ki torej spremeni barvo. Sedaj ima sosed u v kopiji P_m natanko enega nepobarvanega soseda, diagonalnega, ki spremeni barvo, in tako naprej po celotni robni kopiji P_m. Simetrično pa se zgodi tudi v robni kopiji P_n. Po koncu je pobarvana robna in še sosednja kopija P_m ter robna in sosednja kopija P_n, kot prikazano na sliki 18 desno. Vidimo, da je trenutna situacija zelo podobna začetni. Če pogledamo induciran graf iz vozlišč $V(P_m \boxtimes P_n) \setminus Z$, vidimo, da je izomorfen grafu $P_{m-1} \boxtimes P_{n-1}$, pri čemer sta trenutno pobarvani ravno robna kopija P_{m-1} in robna kopija P_{n-1}. Širjenje barve se torej rekurzivno nadaljuje kot prej, na koncu je pobarvan cel graf, začetna množica Z je res množica ničelne prisile. □

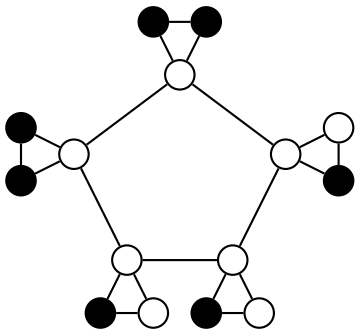
Znana je tudi zgornja meja za ničelno prisilo korone dveh splošnih grafov.

Trditev 5.6. *Za splošna grafa G in H velja*

$$\begin{aligned} Z(G \circ H) &\leq Z(H)n(G) + Z(G)n(H) - Z(G)Z(H) \\ &= Z(H)(n(G) - Z(G)) + Z(G)n(H). \end{aligned}$$

V posebnem, če $m \geq 2$, za korono polnih grafov velja $Z(K_m \circ K_n) \leq mn - 1$.

Dokaz. Vzemimo minimalno množico ničelne prisile grafa G in jo označimo z Z_G. Pobarvamo vozlišča v vseh tistih kopijah H, ki so povezane z vozliščem iz Z_G, s tem smo pobarvali $|Z_G|n(H) = Z(G)n(H)$ vozlišč, saj je Z_G minimalna množica ničelne prisile za G. V preostalih $n(G) - Z(G)$ kopijah H pa pobarvamo tista vozlišča, ki tvorijo minimalno množico ničelne prisile za H. Vse skupaj smo torej pobarvali $Z(H)(n(G) - Z(G)) + Z(G)n(H)$ vozlišč. Primer tako konstruirane množice je prikazan na sliki 19.



Slika 19: Prikazana je korona $C_5 \circ K_2$ in njegova množica ničelne prisile, konstruirana kot v zgornjem dokazu.

Poglejmo, da smo res konstruirali množico ničelne prisile. Kopije H, ki so popolnoma pobarvane, spremenijo barvo svojemu vozlišču iz kopije G. Sedaj so v G torej pobarvana vsa vozlišča iz Z_G. Ker je Z_G množica ničelne prisile za G in ker so kopije H, povezane s temi vozlišči, že črne, ima eno izmed vozlišč v G natanko enega nepobarvanega soseda v G. Ko ta počrni, se barva spremeni tudi vsem nepobarvanem vozliščem v njegovi kopiji H, saj je bila pred tem v tej kopiji pobarvana ravno množica ničelne prisile. Ker je Z_G množica ničelne prisile, se ta postopek ponavlja, dokler ni pobarvan celoten graf.

Zgornja meja za korono polnih grafov v večino primerih sledi iz dokazane splošne zgornje meje (upoštevamo osnovni rezultat o ničelni prisili za polne grafe 3.3):

$$Z(K_m \circ K_n) \leq n - 1 + (m-1)n = mn - 1, \text{ za } n \geq 2.$$

Če pa je $n = 1$, nam zaradi dejstva $Z(K_1) = 1$ splošni dokaz da $1 + (m-1) = m$ namesto v trditvi želene meje $m - 1$. Konstruirajmo množico ničelne prisile velikosti $m - 1$ za primer $K_m \circ K_1$ tako, da pobarvamo vse kopije K_1, razen ene. Pobarvane kopije K_1 povzročijo spremembo vsaka v svojem vozlišču kopije K_m, sedaj je pobarvana množica ničelne prisile v K_m. Eno izmed pobarvanih vozlišč spremeni barvo edinemu nepobarvanemu v K_m, ta pa nato povzroči še spremembo barve svoje kopije K_1. □

Izkaže se [2], da za nekatere zgornje rezultate velja enakost, natančneje za 1. točko posledice 5.2 za grafe $n(G) \leq n$, neenakosti (5.2), (5.3) in (5.4) ter poseben primer trditve 5.6. Poleg tega so podani tudi rezultati za kartezični produkt poti, cikla in polnega grafa s potjo ne glede na velikost grafa ter kartezičnega produkta cikla s polnim grafom, kar povzemamo v naslednji trditvi:

Trditev 5.7 ([2]). *Ničelna prisila za produkte nekaterih družin grafov za $m, n \geq 1$ (če ni označeno drugače) je:*

1. $Z(P_m \,\square\, P_n) = \min\{m, n\}$.
2. $Z(C_m \,\square\, P_n) = \min\{m, 2n\}$ *za* $m \geq 3$.
3. $Z(K_m \,\square\, P_n) = m$ *za* $n \geq 2$.
4. $Z(C_m \,\square\, K_n) = 2n$ *za* $m \geq 4$.

6 Izračun ničelne prisile

V tem poglavju bomo problem ničelne prisile preučili skozi stališče računske zahtevnosti: pogledali si bomo polinomski algoritem, ki preveri, ali je dana množica za dan graf res množica ničelne prisile, in s tem dokazali, da problem spada v NP. Analizirali bomo trivialen splošni algoritem za izračun ničelne prisile poljubnega grafa, si pogledali njegove potencialne izboljšave in povzeli nekaj rezultatov o NP-polnosti problema.

6.1 Predstavitve grafov in računska zahtevnost

Preden začnemo z analizo algoritmov, si oglejmo načine predstavitve grafov v računalnikih in zahtevnosti operacij pri različnih predstavitvah.

Predpostavljali bomo, da so oznake vozlišč grafa G velikosti n kar cela števila od 0 do $n-1$ (v večini programskih jezikov indeks v seznamih in podobnih podatkovnih strukturah teče od 0 naprej). Zaradi takih oznak pri delu z grafi lahko uporabljamo eno bolj preprostih podatkovnih struktur, seznam, saj si podatek o i-tem vozlišču (npr. barvo) shranimo pod indeks i. Če so oznake kaj drugega in jih bomo kasneje potrebovali, jih lahko preslikamo v števila od 0 do $n-1$ in si v seznam na i-to mesto shranimo prejšnjo oznako vozlišča.

Prvi način predstavitve je matrika sosednosti. Ta vsebuje ničle na tistih indeksih (i, j), za katere velja, da vozlišči i in j nista sosednji, in enke za tista vozlišča, ki so sosednja. Na diagonali so torej vedno ničle, saj graf ne vsebuje zank, v primeru neusmerjenega grafa pa je matrika še simetrična. Predstavimo jo s seznamom seznamov, i-ti element zunanjega seznama je i-ta vrstica matrike, torej porabimo $O(n^2)$ prostora. Kakšna je zahtevnost pogostih operacij nad grafom? Da ugotovimo, ali sta vozlišči i in j sosednji, porabimo $O(1)$, saj moramo le preveriti, ali je element v i-ti vrstici na j-tem indeksu enka, in je dostop do elementa seznama prek indeksa časovne zahtevnosti $O(1)$. Enako časovno zahtevnost ima iz istega razloga tudi dodajanje ali odstranjevanje povezave. Če želimo poiskati vse sosede vozlišča i, porabimo $O(n)$ operacij, saj moramo preveriti celotno i-to vrstico. Če želimo dodati

ali odstraniti vozlišče, potrebujemo $O(2n)$ operacij (gremo čez en stolpec in eno vrstico), pri čemer pri odstranjevanju v večini primerih porabimo še več, saj brisanje elementa iz seznama v najslabšem primeru traja $O(n)$, medtem ko je dodajanje na konec seznama amortizirano $O(1)$.

Graf lahko predstavimo tudi s seznamom sosedov: za vsako vozlišče naštejemo njegove sosede. V tem primeru prav tako uporabimo seznam seznamov, pri čemer se na indeksu i zunanjega seznama nahaja seznam sosedov vozlišča i. Pri tem porabimo $O(n+m)$ prostora, kjer je $m = m(G)$ število povezav grafa. Če nas zanima, ali sta vozlišči i in j sosednji, moramo pregledati celoten seznam sosedov vozlišča i, torej je časovna zahtevnost enaka $O(\deg i)$. Če je seznam urejen, lahko uporabimo binarno iskanje, ki ima logaritmično časovno zahtevnost, torej $O(\log \deg i)$. Odstranjevanje povezave med vozliščema i in j traja $O(2 \deg i + 2 \deg j)$, saj moramo v seznamu sosedov obeh poiskati in odstraniti drugo vozlišče, pri iskanju moramo v najslabšem primeru preiskati cel seznam sosedov, pri odstranjevanju pa ga prekopirati eno v levo. V najslabšem primeru traja celotna operacija $O(4n)$ (če je seznam sosedov urejen, pa $O(2 \log n + 2n)$). Dodajanje povezave ij traja $O(1)$, če nam vrstni red seznama sosedov ni pomemben. Če pa nam je vrstni red pomemben, lahko na seznamu sosedov i in j uporabimo binarno iskanje, da najdemo pravo mesto. V tem primeru porabimo $O(\log \deg i + \deg i)$ za eno vozlišče, saj ima dodajanje v seznam dolžine p časovno zahtevnost $O(p)$, v najslabšem primeru skupaj torej $O(2 \log n + 2n)$. Da najdemo vse sosede nekega vozlišča, moramo le pogledati na i-to mesto zunanjega seznama, torej ima operacija konstantno časovno zahtevnost. Če želimo dodati vozlišče, moramo dodati seznam z njegovimi sosedi na konec zunanjega seznama in njegovo oznako na konec nekaterih seznamov sosedov, kar traja $O(n)$. Za odstranjevanje vozlišča pa moramo odstraniti njegov seznam sosedov in vozlišče odstraniti iz vseh ostalih seznamov sosedov. Za prvo porabimo $O(n)$, za drugo pa $O(n^2)$, skupna časovna zahtevnost je torej $O(n^2)$.

Seznam sosedov lahko implementiramo tudi kot seznam množic. Če so te implementirane z zgoščenimi tabelami, to pomeni, da imajo preverjanje, ali je element v množici, dodajanje in odstranjevanje v povprečju konstantno časovno zahtevnost. Ali sta vozlišči sosednji, lahko zato ugotovimo v konstantnem času, prav toliko porabimo tudi za odstranjevanje povezave in dodajanje povezave. Dodajanje vozlišča je še vedno $O(n)$, odstranjevanje pa prav tako postane $O(n)$.

	matrika sosednosti	seznam sosedov (s)	seznam sosedov (m)
prostor	$O(n^2)$	$O(n+m)$	$O(n+m)$
$i \sim j$	$O(1)$	$O(n)$	$O(1)$
dodaj $e = ij$	$O(1)$	$O(1)$	$O(1)$
odstrani $e = ij$	$O(1)$	$O(n)$	$O(1)$
vsi sosedi i	$O(n)$	$O(1)$	$O(1)$
dodaj i	$O(2n)$	$O(n)$	$O(n)$
odstrani i	$O(n^2)$	$O(n^2)$	$O(n)$

Tabela 1: Prikazana je primerjava časovnih zahtevnosti nekaterih pogostih operacij na grafih med tremi predstavitvami.

Vse tri predstavitve lahko primerjamo v tabeli 1. Na prvi pogled se zdi, da bi bilo vedno najbolj optimalno izbrati seznam sosedov, implementiran z množicami. Vendar pa je treba upoštevati, da dodajanje, iskanje in odstranjevanje iz množice v povprečju sicer je konstantne časovne zahtevnosti, je pa ta konstanta večja kot npr. pri dodajanju na konec seznama. Tako je predstavitev najbolj smiselno izbrati glede na to, katere operacije bomo potrebovali in koliko prostora imamo na voljo oziroma kako velike grafe želimo obravnavati.

Ker se bomo dotaknili tudi težavnosti odločitvenih problemov, si poglejmo nekaj osnovnih definicij računske zahtevnosti.

Definicija 6.1.
1. Odločitveni problem Π spada v razred P, pišemo $\Pi \in \mathsf{P}$, če ga lahko rešimo z deterministič\-nim Turingovim strojem s polinomsko časovno zahtevnostjo.

2. Odločitveni problem Π spada v razred NP, pišemo $\Pi \in \mathsf{NP}$, če ga lahko rešimo z nedeterministič\-nim Turingovim strojem s polinomsko časovno zahtevnostjo.

Povedano drugače, v razred P spadajo tisti problemi, za katere poznamo algoritem, ki najde rešitev v polinomskem času. Po drugi strani v NP spadajo tisti problemi, za katere poznamo algoritem, ki v polinomskem času preveri, ali je dana rešitev pravilna. Spodnja definicija razreda NP je namreč ekvivalentna zgornji [25, izrek 7.20]:

Definicija 6.2. Problem Π je v NP, če obstaja certifikat za rešitev, polinomske dolžine glede na vhodne podatke, in obstaja polinomski algoritem (preverjevalec), ki vrne "da", če certifikat za konkreten primer problema dokaže, da je odgovor na problem "da", in "ne" sicer.

Hitro vidimo, da velja $\mathsf{P} \subseteq \mathsf{NP}$. Če za problem poznamo algoritem, ki najde rešitev v polinomskem času, potem moramo poznati tudi algoritem, ki v polinomskem času preveri pravilnost te rešitve, saj sicer ne moremo biti gotovi, da nam prvi algoritem res vrne rešitev problema. (Isto lahko vidimo tudi iz zgornjih definicij obeh razredov, saj je deterministični Turingov stroj posebna različica nedeterminističnega Turingovega stroja.)

Ali velja tudi $\mathsf{NP} \subseteq \mathsf{P}$ in posledično $\mathsf{P} = \mathsf{NP}$? To bi pomenilo, da je vsak problem, katerega rešitev lahko preverimo v polinomskem času, tudi rešljiv v polinomskem času. Odgovor na vprašanje ni znan, veliko matematikov pa predvideva, da je nikalen, saj po nekaj desetletjih preučevanja težkih problem znotraj NP, rečemo jim tudi NP-polni, znanih naj bi jih bilo več kot 3000, nihče ni uspel najti polinomskega algoritma za kateregakoli izmed njih.

Definicija 6.3. Odločitveni problem Π je NP-*poln*, pišemo $\Pi \in \mathsf{NPC}$, če velja:

1. $\Pi \in \mathsf{NP}$,

2. vsak problem $\Pi' \in \mathsf{NP}$ lahko v polinomskem času prevedemo na Π.

Polinomska prevedljivost problema Π' na problem Π pomeni, da je reševanje Π' lažje (oziroma kvečjemu polinomsko težje) od reševanja problema Π. Povedano drugače, nek polinomski algoritem, ki reši problem Π, lahko uporabimo za konstrukcijo polinomskega algoritma, ki reši Π'.

Povedali smo že, da za probleme v NPC ne poznamo polinomskih algoritmov. Če bi tak algoritem za nek NP-poln problem obstajal, bi sledilo P = NP:

Trditev 6.4. *Naj bo* $\Pi \in \mathsf{NPC}$. *Potem velja:*

$$\Pi \in \mathsf{P} \iff \mathsf{P} = \mathsf{NP}.$$

Dokaz. Implikacija v levo je očitna: če $\mathsf{P} = \mathsf{NP}$, potem $\Pi \in \mathsf{NPC} \implies \Pi \in \mathsf{NP} = \mathsf{P}$. Poglejmo še implikacijo v desno. Ker $\Pi \in \mathsf{NPC}$, pomeni, da lahko vsak problem v NP v polinomskem času prevedemo na Π. Ker $\Pi \in \mathsf{P}$, imamo polinomski algoritem za reševanje Π, torej lahko v polinomskem času rešimo tudi vse ostale probleme iz NP, torej za vsak problem iz NP sledi, da je tudi iz P. Velja torej $\mathsf{NP} \subseteq \mathsf{P}$ in skupaj z $\mathsf{P} \subseteq \mathsf{NP}$ sledi $\mathsf{P} = \mathsf{NP}$. □

Problemi, ki so NP-polni, so iz teorije grafov na primer problem vozliščnega pokritja, obstoja Hamiltonskega cikla in popolnega trirazsežnega prirejanja. Če želimo dokazati, da je nek problem NP-poln, se tega lahko lotimo s pomočjo naslednje leme:

Lema 6.5. *Naj bo* $\Pi' \in \mathsf{NPC}$, $\Pi \in \mathsf{NP}$ *in naj bo* Π' *v polinomskem času prevedljiv na problem* Π. *Potem velja* $\Pi \in \mathsf{NPC}$.

Dokaz. Ker je $\Pi' \in \mathsf{NPC}$, lahko vsak problem iz NP v polinomskem času prevedemo na Π'. Ker lahko Π' v polinomskem času prevedemo na Π, lahko po tranzitivnosti relacije prevedljivosti vsak problem iz NP prevedemo na Π v polinomskem času, torej velja $\Pi \in \mathsf{NPC}$. □

NP-polnost problema pa lahko dokažemo tudi tako, da ga zožimo na nek podproblem, za katerega vemo, da je NP-poln. (Če je podproblem težek, potem je gotovo težek tudi širši problem.) Ti dve tehniki bomo uporabili v podpoglavju 6.4.

Do zdaj smo govorili samo o težavnosti odločitvenih problemov, kaj pa, če imamo nek splošen računski problem? Formalno definiramo, da je računski problem NP-*težek*, če bi obstoj polinomskega algoritma impliciral, da je P = NP. Tako sem spadajo tudi vsi NP-polni (odločitveni) problemi. Če imamo nek optimizacijski problem, kot ga bomo srečali v tem delu, ga lahko prevedemo na odločitven problem; če je ta NP-poln, potem je originalni problem NP-težek.

6.2 Preverljivost množice ničelne prisile

Dan imamo (ne nujno povezan) graf G in neko podmnožico njegovih vozlišč $Z \subseteq V$. Najti želimo algoritem, ki preveri, ali je Z množica ničelne prisile. To naredimo tako, da simuliramo postopek širjenja barve po pravilu, da črno vozlišče povzroči spremembo barve v edinemu nepobarvanemu sosedu. Pregledamo vsa črna vozlišča, ugotovimo, katera imajo natanko enega nepobarvanega soseda, in mu spremenimo barvo. Postopek ponavljamo, dokler se še dogajajo kakšne spremembe. Če je na koncu pobarvan cel graf, je Z res množica ničelne prisile, v nasprotnem primeru pa ne.

Psevdokoda zgornjega postopka je zapisana v algoritmu 1, v funkciji PREVERI, pod njo pa sta zapisani še dve pomožni funkciji. Za graf v tem primeru izberemo

Algoritem 1 Preveri, ali je dana množica množica ničelne prisile za dan graf.

Vhod: Graf $G = (V, E)$, množica $Z \subseteq V(G)$.
Izhod: Logična vrednost `true` ali `false`, ki pove, ali je Z množica ničelne prisile za G.

```
1: function PREVERI(G, Z)
2:     for each u in Z do
3:         barva(u) ← 0                ▷ Barva 0 pomeni, da je vozlišče pobarvano.
4:     end for
5:     for each u in V \ Z do
6:         barva(u) ← 1                ▷ Barva 1 pomeni, da vozlišče ni pobarvano.
7:     end for
8:     sprememba ← true
9:     while sprememba do
10:        sprememba ← false
11:        for each u in V do
12:            if barva(u) = 0 then
13:                sosedi ← NAJDIBELESOSEDE(G, u, barva)
14:                if DOLZINA(sosedi) = 1 then      ▷ Edini nepobarvan sosed u,
15:                    barva(sosedi[0]) ← 0                  ▷ spremeni barvo.
16:                    sprememba ← true
17:                end if
18:            end if
19:        end for
20:    end while
21:    return VSIPOBARVANI(G, barva)
22: end function
23: function NAJDIBELESOSEDE(G, u, barva)
24:    sosedi ← [ ]                          ▷ Seznam belih sosedov u.
25:    for each v in N(u) do
26:        if barva(v) = 1 then
27:            DODAJ(sosedi, v)
28:        end if
29:    end for
30:    return sosedi
31: end function
32: function VSIPOBARVANI(G, barva)
33:    for each u in V do
34:        if barva(u) = 1 then return false    ▷ Obstaja nepobarvano vozlišče.
35:        end if
36:    end for
37:    return true                           ▷ Vsa vozlišča so pobarvana.
38: end function
```

predstavitev s seznamom sosedov, implementiranega s seznamom, saj želimo prihraniti na prostoru. (Če bi izbrali matriko sosednosti, bi že za grafe velikosti 25000

porabili približno 5GB spomina, tudi za redke grafe, kot je na primer pot. Če pa bi pot velikosti 25000 predstavili s seznamom sosedov, bi porabili le nekaj MB.) Poleg tega ne bomo potrebovali operacij odstranjevanja ali ugotavljati sosednosti med dvema vozliščema, zato je smiselno izbrati bolj preprosto podatkovno strukturo seznam namesto množice.

Funkcija NAJDIBELESOSEDE za dan graf G, vozlišče u in trenutno "barvanje" *barva* poišče vse nepobarvane sosede vozlišča u. Uporablja podatkovno strukturo seznam, ki ima metodo DODAJ za dodajanje elementa na konec seznama z amortizirano konstantno časovno zahtevnostjo. Funkcija VSIPOBARVANI sprejme graf G in trenutno "barvanje" *barva* ter vrne logično vrednost `true` ali `false`, odvisno od tega, ali so vsa vozlišča v grafu pobarvana.

Analizirajmo zahtevnost algoritma 1. Označimo velikosti vhodnih podatkov takole: $n = n(G), m = m(G), z = |Z|$.

Najprej si poglejmo zahtevnost pomožnih funkcij, NAJDIBELESOSEDE in VSIPOBARVANI. Časovna zahtevnost prve je $O(\deg u)$, saj pregledamo vse sosede vozlišča u, dodajanje na konec seznama pa je (amortizirane) časovne zahtevnosti $O(1)$. Časovna zahtevnost druge pa je $O(n)$, saj v najslabšem primeru (vsa vozlišča so pobarvana) pregledamo vsa vozlišča.

Sedaj se osredotočimo na glavno funkcijo PREVERI. Vrstice 2–7 imajo skupno časovno zahtevnost $O(n)$, prav tako vrstica 21. Analizirajmo operacije znotraj `while` zanke, ki poteka od vrstice 9 do 20. Za vsako vozlišče $u \in V$ preverimo, ali je pobarvano, in če je, s pomočjo pomožne funkcije pregledamo, ali ima natanko enega belega soseda. Vrstice 14–17 so konstantne, zato je časovna zahtevnost za trenutno vozlišče u v zanki $O(\deg u)$. Celotna `for` zanka (11–19) je torej $O(\sum_{u \in V} \deg u + n) = O(m + n)$ (upoštevamo lemo o rokovanju $\sum_{u \in V} \deg u = 2m(G)$).

Preštejmo še, koliko korakov naredi `while` zanka. V vsakem koraku se zgodi vsaj ena sprememba barve (v nasprotnem primeru se zanka ustavi), na začetku pa je pobarvanih z vozlišč, torej se zgodi največ $n - z$ korakov. Celotna `while` zanka ima torej časovno zahtevnost $O((n-z)(m+n))$, ker pa je v splošnem $1 \leq z \leq n-1$, dobimo $O(n(m+n))$, kar je tudi skupna časovna zahtevnost algoritma 1.

Poglejmo si še prostorsko zahtevnost. Za shranjevanje barve vozlišč porabimo $O(n)$ dodatnega prostora, za sledenje, ali se je zgodila kakšna sprememba, $O(1)$ prostora, za lokalno shranjevanje sosedov trenutnega vozlišča v vrstici 13 pa $O(\deg u)$, skupno torej največ $O(m)$. Skupna prostorska zahtevnost je torej $O(n + m)$.

(Za pregled nepobarvanih sosedov bi lahko dosegli tudi konstantno prostorsko zahtevnost: namesto funkcije NAJDIBELESOSEDE bi pregledali vse sosede in v spremenljivki hranili, ali smo do sedaj že videli kakšnega nepobarvanega. Če bi našli nepobarvano vozlišče in bi v spremenljivki imeli zapisano že neko drugo, bi prenehali s pregledom, saj smo našli več kot enega belega soseda. Če pa v spremenljivki ne bi bilo še nič shranjeno, bi si tja zapisali trenutno vozlišče. Tako bi dosegli skupno prostorsko zahtevnost $O(n)$.)

Izkaže se, da lahko algoritem za preverjanje precej izboljšamo v smislu časovne zahtevnosti. Na začetku si pripravimo tabelo, kjer bomo za vsako vozlišče beležili, katero vozlišče je povzročilo spremembo barve. Za vsako vozlišče si shranimo tudi množico belih sosedov, pri čemer upoštevamo začetni Z. Za tista črna vozlišča u, ki imajo natanko enega belega soseda v, vozlišče v zapišemo v podatkovno strukturo

vrsta, v tabeli pa označimo, da je spremembo v-ja povzročilo vozlišče u.

Če je vrsta prazna, potem se postopek širjenje barve ne more začeti, Z ni množica ničelne prisile. V nasprotnem primeru odstranimo začetni element u iz vrste, mu spremenimo barvo in preverimo, ali lahko takoj povzroči kakšno spremembo barve, torej ali ima natanko enega belega soseda v. Če ga ima, preverimo, ali spremembo barve v ni povzročilo že kakšno drugo vozlišče, in če ne, ga dodamo v vrsto in označimo, da je spremembo u povzročil v (ta pregled je zapisan v pomožni funkciji PREVERISOSEDE, saj ga uporabimo večkrat). Nato vsem sosedom vozlišča u odstranimo u iz množice belih sosedov. Če je pri tem v množici belih sosedov katerega izmed črnih vozlišč ostalo le eno vozlišče, kot prej preverimo, ali naj ga dodamo v vrsto.

Tako ponavljamo, dokler vrsta ni prazna. Nato preverimo barve vseh vozlišč, če so vsa pobarvana, je Z množica ničelne prisile. Postopek je podrobneje opisan v algoritmu 2, pri čemer uporabljamo pomožni funkciji, opisani v algoritmu 1.

V algoritmu 2 uporabljamo podatkovno strukturo množica za shranjevanje belih sosedov nekega vozlišča. Uporabimo metodi MNOZICA, ki iz dane podatkovne strukture naredi množico, in ODSTRANI, ki odstrani dan element iz množice, če pa ni danega nobenega elementa, odstrani in vrne naključen element; če danega elementa ni v množici oziroma je množica prazna, pa ne odstrani ničesar. Če množico implementiramo z zgoščenimi tabelami, je v povprečju časovna zahtevnost odstranjevanja elementa konstantna.

Poglejmo si še podatkovno strukturo vrsta. Ta deluje po principu "first in, first out": element, ki ga prvega dodamo v vrsto, bo to prvi tudi zapustil. V učinkovitih implementacijah vrste (npr. z dvojno povezanim seznamom) imata tako dodajanje kot odstranjevanje iz vrste konstantno časovno zahtevnost.

Sedaj analizirajmo zahtevnost algoritma 2 in pokažimo, da smo v primerjavi z algoritmom 1 res dosegli izboljšavo.

Trditev 6.6. *Časovna zahtevnost algoritma 2 je* $O(m+n)$.

Dokaz. Najprej si poglejmo časovno zahtevnost pomožne funkcije PREVERISOSEDE. Če je vozlišče u pobarvano in ima natanko enega belega soseda v, preverimo, ali je sprememba barve v povzročilo že katero drugo vozlišče, in če ni, dodamo v v vrsto. Celoten postopek traja konstantno časa, saj je dodajanje v vrsto operacija s konstantno časovno zahtevnostjo.

Vrstice 2–9 imajo skupno časovno zahtevnost $O(n)$, vrstica 10 pa konstantno. Analizirajmo predprocesiranje, ki se dogaja na vrsticah 11–14. Za vsako vozlišče poiščemo vse nepobarvane sosede in jih pretvorimo v množico. Vemo, da ima NAJDIBELESOSEDE časovno zahtevnost $O(\deg u)$, pretvorba v množico pa je linearna v številu elementov seznama, ki ga pretvarjamo. Skupna časovna zahtevnost vrstice 12 je torej $O(\deg u)$. V vrstici 13 kličemo funkcijo PREVERISOSEDE, za katero vemo, da je konstantne časovne zahtevnosti. Predprocesiranje ima torej skupno zahtevnost $O(m+n)$, saj za vsako vozlišče u potrebujemo $O(\deg u + 1)$.

Sedaj si poglejmo notranjost `while` zanke na vrsticah 15–23. Odstranjevanje iz vrste ima konstantno časovno zahtevnost, prav tako sprememba barve vozlišča in klic funkcije PREVERISOSEDE. Nato za vsakega soseda vozlišča u odstranimo u iz njihove množice belih sosedov, časovna zahtevnost odstranjevanja je konstantna,

Algoritem 2 Preveri, ali je dana množica množica ničelne prisile za dan graf s pomočjo podatkovne strukture vrsta.

Vhod: Graf $G = (V, E)$, množica $Z \subseteq V(G)$.
Izhod: Logična vrednost `true` ali `false`, ki pove, ali je Z množica ničelne prisile za G.

```
1: function PREVERIIZBOLJSAN(G, Z)
2:     for each u in Z do
3:         barva(u) ← 0                    ▷ Barva 0 pomeni, da je vozlišče pobarvano.
4:         povzroci(u) ← 'Z'               ▷ Označimo, da so to začetna črna vozlišča.
5:     end for
6:     for each u in V \ Z do
7:         barva(u) ← 1                    ▷ Barva 1 pomeni, da vozlišče ni pobarvano.
8:         povzroci(u) ← null
9:     end for
10:    Q ← [ ]                                          ▷ Q postane prazna vrsta.
11:    for each u in V do                                ▷ Predprocesiranje.
12:        beliSosedi(u) ← MNOZICA(NAJDIBELESOSEDE(G, u, barva))
13:        PREVERISOSEDE(barva, beliSosedi, povzroci, Q, u)
14:    end for
15:    while Q ≠ [ ] do          ▷ Spremembe se dogajajo, dokler Q ni prazna.
16:        u ← ODSTRANI(Q)                   ▷ Ko element vzamemo iz vrste,
17:        barva(u) ← 0                      ▷ zabeležimo spremembo barve.
18:        PREVERISOSEDE(barva, beliSosedi, povzroci, Q, u)
19:        for each v in N(u) do             ▷ Vsem sosedom odstranimo u
20:            ODSTRANI(beliSosedi(v), u)    ▷ iz množice belih sosedov.
21:            PREVERISOSEDE(barva, beliSosedi, povzroci, Q, v)   ▷ Preverimo,
22:        end for          ▷ ali ima kateri od njih točno enega belega soseda.
23:    end while
24:    return VSIPOBARVANI(G, barva)
25: end function

26: function PREVERISOSEDE(barva, beliSosedi, povzroci, Q, u)
27:    if barva(u) = 0 ∧ DOLZINA(beliSosedi(u)) = 1 then
28:        v ← ODSTRANI(beliSosedi(u))          ▷ v je edini bel sosed u.
29:        if povzroci(v) = null then           ▷ Če spremembe barve v še ni
30:            DODAJ(Q, v)                      ▷ povzročilo nobeno drugo vozlišče,
31:            povzroci(v) = u                  ▷ jo povzroči u.
32:        end if
33:    end if
34: end function
```

zatem pa s pomožno funkcijo PREVERISOSEDE konstantne časovne zahtevnosti ponovno preverimo, ali je pri tem ostal samo en bel sosed. En korak `while` zanke ima torej za vozlišče u časovno zahtevnost $O(\deg u + 1)$.

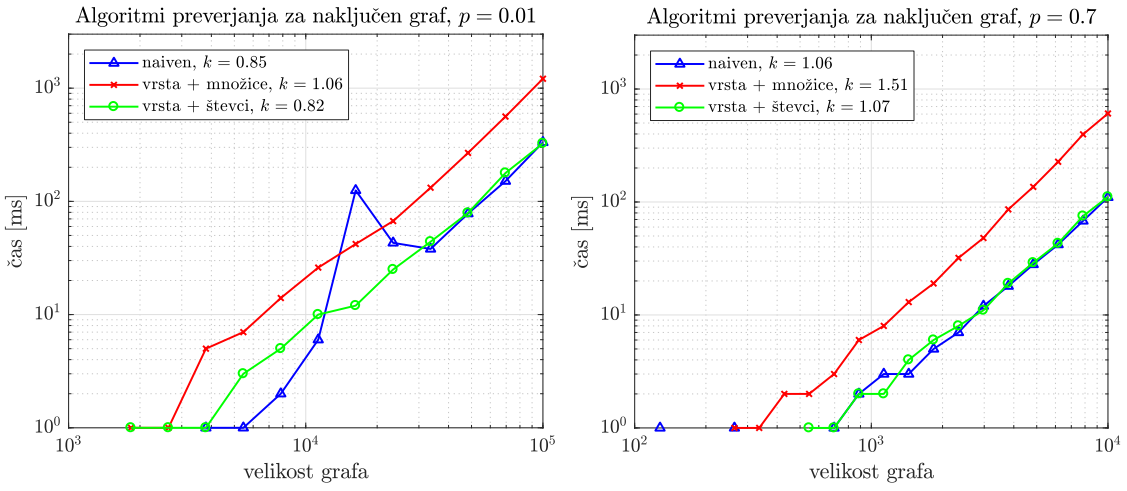
Zanka ima toliko korakov, kolikor elementov dodamo v vrsto. Ker v vrsto dodajamo še nepobarvana vozlišča (in se ta v vrsti nikoli ne ponovijo, saj s pomočjo

povzroci beležimo, ali so že dodana), jih lahko dodamo največ $n - z$, v najslabšem primeru $n - 1$, torej kar $O(n)$. Skupna časovna zahtevnost zanke je torej $O(m + n)$, tako kot tudi predprocesiranja, torej je to tudi časovna zahtevnost algoritma 2. □

Analizirajmo še prostorsko zahtevnost algoritma 2. Shranjevanje barv porabi $O(n)$ prostora, prav tako podatek, kdo povzroči spremembo barve v vozliščih, zapisan v *povzroci*. (Lahko bi si tudi samo zapisali, ali je vozlišče že dodano v vrsto, vendar s spremenljivko *povzroci* dobimo še dodaten podatek; z majhno modifikacijo lahko algoritem vrne zaporedje sprememb barve vozlišč in kdo jih je povzročil, pri tem pa ostane red prostorske zahtevnosti enak.) Za shranjevanje belih sosedov nekega vozlišča u porabimo $O(\deg u)$, skupaj torej $O(m + n)$ prostora. Ostane še vrsta, v katero vemo, da dodamo največ $O(n)$ vozlišč, torej potrebujemo toliko tudi prostora. Skupna prostorska zahtevnost algoritma 2 je torej $O(m + n)$.

Prostorsko zahtevnost bi lahko zmanjšali, če bi namesto shranjevanja nepobarvanih sosedov za vsako vozlišče shranjevali samo število. S tem bi porabili $O(n)$ prostora, skupna prostorska zahtevnost bi bila torej $O(n)$. Pri tem bi morali, po tem, ko bi v pomožni funkciji PREVERISOSEDE ugotovili, da se je števec nepobarvanih sosedov nekega vozlišča u zmanjšal na 1, poiskati, kateri sosed vozlišča u je še nepobarvan, za kar bi potrebovali $O(\deg u)$ operacij. Ker bi se to zgodilo največ $O(n)$-krat, bi skupno porabili $O(n + m)$ operacij več, kar torej ne spremeni reda časovne zahtevnosti, spremeni pa konstanto.

V prilogi A je podana C++ implementacija vseh treh algoritmov (naivnega algoritma 1 in dveh verzij algoritma 2, z množicami ali števci). Na sliki 20 je predstavljena primerjava časov izvajanja teh treh algoritmov v odvisnosti od velikosti grafa za naključno generirane grafe G po modelu Erdős–Rényi (povezavo dodamo v G z verjetnostjo p neodvisno od ostalih povezav). Levo je izbran $p = 0.01$, torej z veliko



Slika 20: Predstavljena je primerjava časov izvajanja treh algoritmov za preverjanje in njihovi približni koeficienti rasti v odvisnosti od velikosti grafa za naključno generirane grafe G po modelu Erdős–Rényi; levo je izbran $p = 0.01$, desno pa $p = 0.7$.

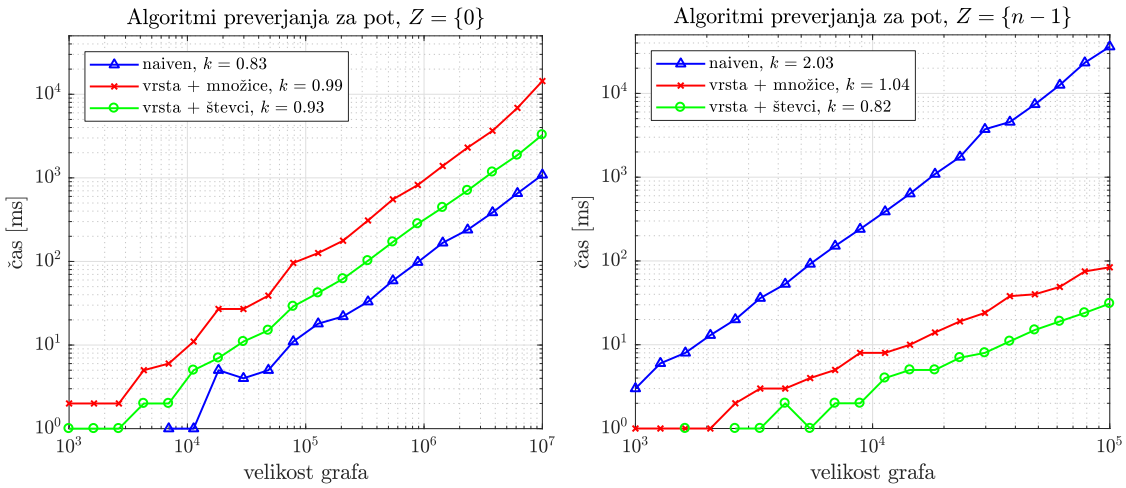
verjetnostjo generiramo "redke" grafe z malo povezavami, desno pa $p = 0.7$, torej

imajo generirani grafi veliko povezav. V obeh primerih je za preverjanje izbrana naključna množica 95% velikosti grafa G.

Izkaže se, da je na gostih grafih (slika 20 desno) zelo malo verjetno, da se bo za izbrano množico sploh začel proces širjenja barve. Vsi algoritmi tako naredijo samo eno iteracijo, vendar pa ima naiven algoritem pri tem precej manj dela (ne gradi množic ali dodatnih tabel), zato je za majhne velikosti najhitrejši, vendar ga algoritem z vrsto in števci hitro ujame. Ker je delo z množicami najbolj zamudno, je ta različica algoritma z vrsto najpočasnejša.

Pri redkih grafih (slika 20 levo) vsaj za manjše grafe (do 10000 vozlišč) še kar pogosto naletimo na množico ničelne prisile, kar pomeni tudi več dela za algoritme. Naiven algoritem se kljub temu (malo presenetljivo) v večini primerov dobro izkaže: razlog temu je majhno število (največ 2) iteracij zunanje `while` zanke algoritma 1, kar pa se verjetno zgodi zaradi prikladnega vrstnega reda sprememb barv vozlišč (naivnemu algoritmu ugaja, če se spremembe dogajajo čim bolj po vrsti od oznake 0 do $n-1$). Kljub temu na sliki vidimo skok, kjer se odreže slabše kot ostala dva: to se zgodi pri velikosti $n = 16238$, kjer naredi 11 iteracij `while` zanke.

Da bi bolje demonstrirali odvisnost naivnega algoritma od oznak vozlišč, si poglejmo še primerjavo teh treh algoritmov na poteh različnih velikosti, prikazano na sliki 21. Pri tem smo na levi strani za začetno množico vzeli vozlišče z oznako 0, na desni pa z oznako $n-1$. Na levem grafu se naiven algoritem izkaže najbolje, saj



Slika 21: Primerjava treh algoritmov za preverjanje in njihovi približni koeficienti rasti za poti v odvisnosti od velikosti; pri tem je na levi strani izbrana začetna množica $\{0\}$, na desni pa $\{n-1\}$.

je iskanje edinega belega soseda hitro (vsako vozlišče ima največ 2 soseda), poleg tega zaradi vrstnega reda sprememb algoritem naredi le 1 iteracijo `while` zanke. Na desnem grafu pa se pokaže prava časovna zahtevnost naivnega algoritma, saj mora zaradi obratnega vrstnega reda sprememb(od $n-1$ proti 0) `while` zanka narediti toliko iteracij, kolikor je velikost grafa, in odpove že za poti velikosti 10^6. Vidimo lahko, da je trend hitrosti naivnega algoritma v tem primeru kvadratičen, kar se ujema s teoretično časovno zahtevnostjo.

Opazimo tudi, da je teoretično boljša različica z množicami v praksi pravzaprav slabša. Vzrok temu je, da je povečanje števca v tabeli za 1 v jeziku C++ majhna

operacija, medtem ko je odstranjevanje iz množice večja in precej počasnejša. Dejstvo, da moramo v različici s števci poiskati edinega belega soseda, na hitrost ne vpliva preveč, saj se to zgodi zelo redko.

6.3 Eksponenten algoritem za izračun

Sedaj si poglejmo najbolj osnoven algoritem za izračun ničelne prisile. Če imamo dan splošen (ne nujno povezan) graf G in ne vemo ničesar o njegovem številu ničelne prisile $Z(G)$, lahko preiščemo vse možnosti. Za vsako podmnožico vozlišč grafa G preverimo, ali je množica ničelne prisile, in sledimo, katera je do zdaj najmanjša taka množica. Pri tem uporabimo izboljšani algoritem za preverjanje, zapisan v algoritmu 2. Postopek je opisan v algoritmu 3.

Algoritem 3 Izračuna število ničelne prisile s preiskovanjem vseh možnosti.

Vhod: Graf $G = (V, E)$.
Izhod: Število ničelne prisile $Z(G)$, $1 \leq Z(G) \leq n - 1$.

```
1: function IZRACUNAJSTEVILO(G)
2:     z ← n + 1
3:     for each Z in 2^V do      ▷ Preverimo vsako množico v potenčni množici V.
4:         if PREVERIIZBOLJSAN(G, Z) ∧ |Z| < z then
5:             z ← |Z|
6:         end if
7:     end for
8:     return z
9: end function
```

Analizirajmo zahtevnost algoritma 3. Za vsak element potenčne množice V pokličemo PREVERIIZBOLJSAN, ki ima časovno zahtevnost $O(m+n)$. Vseh elementov je 2^n, skupna časovna zahtevnost algoritma je torej $O(2^n(m+n))$. (Časovna zahtevnost bi bila še slabša, če bi uporabili osnovni algoritem za preverjanje.) Prostorska zahtevnost algoritma pa je $O(m+n)$, saj je to prostorska zahtevnost funkcije PREVERIIZBOLJSAN, ki edina porabi nekaj dodatnega prostora (ki ga po izhodu iz funkcije tudi sprosti), preostanek algoritma pa porabi $O(n)$ prostora za shranjevanje trenutne množice Z.

Razmislimo, ali lahko algoritem 3 kako izboljšamo. Hitro opazimo, da nam pravzaprav ni treba pregledati čisto vseh podmnožic. Če jih uredimo naraščajoče po velikosti, potem lahko s pregledovanjem končamo, čim najdemo eno, ki je množica ničelne prisile (ker vemo, da so vse kasnejše podmnožice, ki jih še nismo pregledali, kvečjemu večje ali enake po velikosti). Podobno, lahko jih uredimo padajoče, in s pregledovanjem končamo, čim za neko velikost ne najdemo več množice ničelne prisile. (Da potem ne obstaja kakšna manjša množica ničelne prisile, vemo zaradi posledice 2.11.)

Za oba vrstna reda obstajajo primeri, kjer moramo v najslabšem primeru še vedno pregledati vse možnosti. Prav tako sta pristopa ekvivalentna v splošnem primeru, kjer ne vemo nič o številu ničelne prisile: za grafe z majhnim številom bo sicer bolje deloval naraščajoči vrstni red, za grafe z velikim številom padajoči,

gledano skupno (čez vse grafe) pa bo število operacij zaradi simetrije enako, saj je število podmnožic velikosti i enako številu podmnožic velikosti $n-i$, $\binom{n}{i} = \binom{n}{n-i}$.

Vendar pa lahko število pregledanih možnosti v primeru padajočega vrstnega reda še malo izboljšamo. Čim za velikost i najdemo neko množico ničelne prisile, vemo, da velja $Z(G) \leq i$, zato lahko začnemo pregledovati množice velikosti $i-1$. V najslabšem primeru moramo sicer še vedno pregledati vse podmnožice neke velikosti, preden lahko nadaljujemo z manjšimi.

Še ena ideja za izboljšavo temelji na trditvi 2.10, ki pravi, da lahko neki množici ničelne prisile dodamo nekaj vozlišč, pa bo še vedno množica ničelne prisile. Začnemo s padajočim vrstnim redom in poiščemo vse množice ničelne prisile velikosti $n-2$, označimo jih z $\mathcal{Z}_{n-2}$. (Vemo, da so vse podmnožice velikosti $n-1$ množice ničelne prisile, zato začnemo kar z $n-2$.) Nato preiščemo samo tiste velikosti $n-3$, ki so podmnožice katere izmed množic v $\mathcal{Z}_{n-2}$. Nadaljujemo, dokler še najdemo kakšno množico ničelne prisile. Tako res ne izpustimo nobene množice ničelne prisile, kar nam zagotovi kontrapozicija posledice 2.11: če obstaja neka množica ničelne prisile velikosti z_2, obstaja tudi množica ničelne prisile velikosti z_1 za vsak $z_2 \leq z_1 \leq n-1$. V primerjavi s prejšnjim pristopom porabimo nekaj več prostora, ker moramo shraniti množice ničelne prisile, poleg tega ne moremo prekiniti z iskanjem, čim najdemo neko množico ničelne prisile trenutne velikosti, nam pa možnosti zato zožijo množice ničelne prisile prejšnje velikosti.

Iskalni prostor si lahko zmanjšamo tudi z binarnim iskanjem (oziroma bisekcijo). Vemo, da vedno obstaja množica ničelne prisile velikosti $n-1$. Preverimo vse množice velikosti 1. Če je katera od njih množica ničelne prisile, smo končali, saj vemo, da je $Z(G) = 1$. V nasprotnem primeru pogledamo množice velikosti $\lfloor \frac{n}{2} \rfloor$; če najdemo kakšno množico ničelne prisile, postopek iskanja ponovimo na intervalu $[1, \lfloor \frac{n}{2} \rfloor]$, v nasprotnem primeru pa na intervalu $[\lfloor \frac{n}{2} \rfloor + 1, n-1]$. Časovna zahtevnost pa je še vedno eksponentna, najslabši primer je npr. $Z(G) = \lfloor \frac{n}{2} \rfloor$, kjer prvič izberemo levi interval $[1, \lfloor \frac{n}{2} \rfloor]$, nato pa vsakič desnega: $[\lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor], [\lfloor \frac{n}{4} \rfloor + \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{2} \rfloor], \ldots$ Ocenimo število množic N, ki jih pri tem pregledamo:

$$\begin{aligned}
N &= \binom{n}{\lfloor \frac{n}{2} \rfloor} + \binom{n}{\lfloor \frac{n}{4} \rfloor} + \binom{n}{\lfloor \frac{n}{4} \rfloor + \lfloor \frac{n}{8} \rfloor} + \cdots \\
&\geq \binom{n}{\lfloor \frac{n}{4} \rfloor} = \prod_{i=1}^{\lfloor \frac{n}{4} \rfloor} \frac{n-i+1}{i} \\
&= \frac{n}{1} \frac{n-1}{2} \cdots \frac{n - \lfloor \frac{n}{4} \rfloor + 1}{\lfloor \frac{n}{4} \rfloor} \\
&\geq \left(\frac{n - \lfloor \frac{n}{4} \rfloor + 1}{\lfloor \frac{n}{4} \rfloor} \right)^{\lfloor \frac{n}{4} \rfloor} \geq \left(\frac{n - \frac{n}{4} + 1}{\frac{n}{4}} \right)^{\lfloor \frac{n}{4} \rfloor} \\
&= \left(3 + \frac{4}{n} \right)^{\lfloor \frac{n}{4} \rfloor} \geq 3^{\lfloor \frac{n}{4} \rfloor} \geq 3^{\frac{n-4}{4}} = 2^{\frac{n-4}{4 \log_3 2}} = c \cdot 2^{\frac{n}{2.5237\ldots}}.
\end{aligned}$$

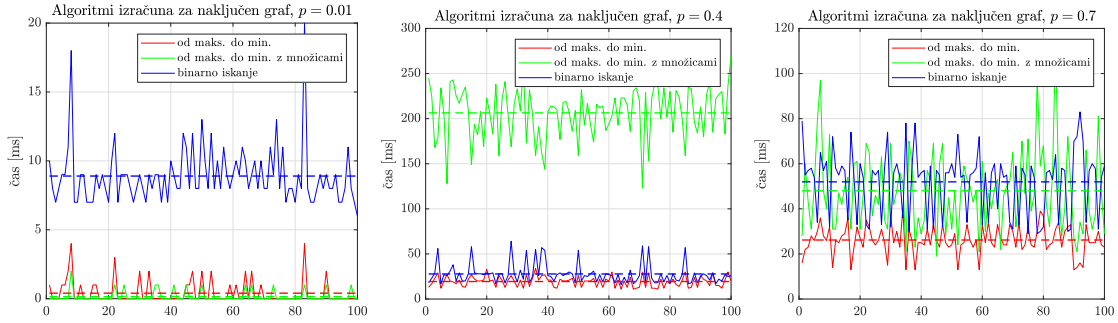
Pri tem smo uporabili $\lfloor \frac{n}{4} \rfloor \leq \frac{n}{4}$ in v konstanto c skrili $2^{\frac{-4}{4 \log_3 2}}$. Časovna zahtevnost je torej še vedno eksponentna: $\Omega((m+n) 2^{\frac{n}{2.5237\ldots}})$.

Eksperimentalno primerjajmo tri (najbolj zanimive) različice eksponentnega algoritma 3, in sicer:

- pregled množic v padajočem vrstnem redu glede na velikost,
- pregled množic v padajočem vrstnem redu glede na velikost, pri čemer pregledujemo le podmnožice množic ničelne prisile prejšnje velikosti,
- pregled z bisekcijo.

Pri tem za preverjanje, ali je množica res množica ničelne prisile, uporabimo algoritem 2, implementiran s tabelo števcev (saj se je v prejšnjem podpoglavju v splošnem izkazal za najhitrejšega), podrobnosti implementacije so ponovno podane v prilogi A.

Na sliki 22 so predstavljene primerjave časov izvajanja teh treh algoritmov za 100 naključno generiranih grafov po modelu Erdős–Rényi za različne vrednosti p (in s tem gostote povezav): od leve proti desni za $p = \{0.01, 0.4, 0.7\}$, pri čemer je bila velikost generiranih grafov fiksna, $n(G) = 15$. Opazimo, da se za redke

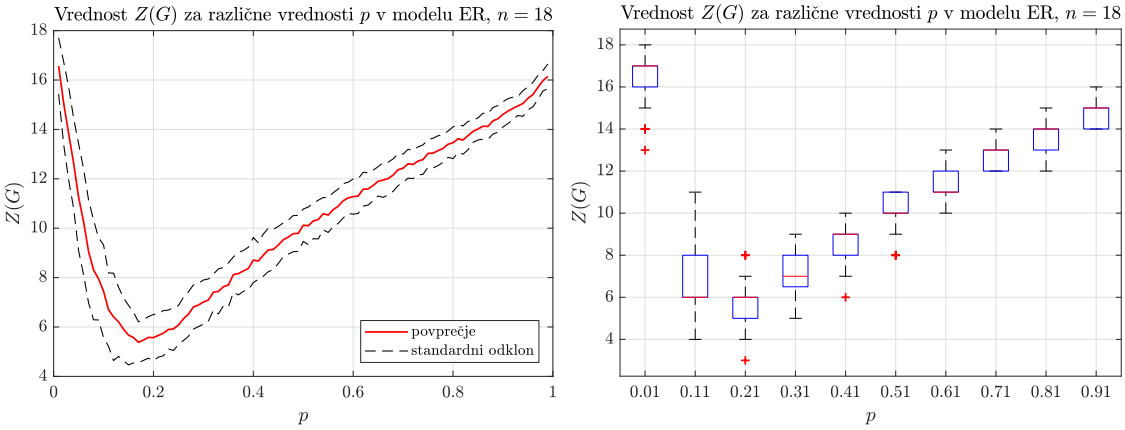


Slika 22: Primerjava časov izvajanja treh različic algoritma 3 (padajoči pregled, padajoči pregled z upoštevanjem prejšnjih množic ničelne prisile, pregled z binarnim iskanjem) za 100 naključno generiranih grafov po modelu Erdős–Rényi velikosti 15, pri čemer je p levo enak 0.01, v sredini 0.4, desno pa 0.7.

grafe, ki imajo zelo verjetno veliko izoliranih vozlišč in posledično veliko ničelno prisilo (večinoma $n-1$) najslabše obnaša binarno iskanje. To je smiselno, saj vsakič preveri sredino trenutnega intervala, ugotovi, da množica ničelne prisile te velikosti ne obstaja, in skoči v desni interval, konča pa šele, ko sreča $n-1$, torej po $\log n$ korakih. Ostala dva algoritma medtem preverita le podmnožice velikosti $n-2$, padajoči algoritem s podmnožicami še nekaj manj kot navaden, zato je najhitrejši.

Za srednje goste grafe se najslabše izkaže padajoči pregled s podmnožicami. Pri takem p je namreč veliko različnih množic ničelne prisile iste velikosti, zato ima algoritem veliko dela z generiranjem podmnožic, na koncu pa verjetno pregleda zelo podobno število podmnožic kot navaden padajoči pregled. Generiranje podmnožic bi sicer lahko bilo implementirano hitreje, če bi vse množice hranili v bitih, vendar bi se različica algoritma s podmnožicami še vedno odrezala najslabše. Binarno iskanje in navaden padajoči pregled imata podobne hitrosti, vendar ima binarno iskanje večje skoke in tudi malo večji povprečen čas.

Za goste grafe so hitrosti precej podobne, v splošnem pa se najbolje izkaže padajoča različica algoritma. Binarno iskanje ima sicer najslabši povprečni čas, ki pa se



Slika 23: Distribucija ničelne prisile za model Erdős–Rényi v odvisnosti od p za fiksno velikost grafov $n = 18$. Za vsak p je ničelna prisila izračunana za 100 naključno generiranih grafov, levo sta narisana povprečje in standardni odklon, desno pa škatla z brki.

od padajočega pregleda s podmnožicami ne razlikuje veliko, poleg tega ima slednji višje skoke.

Zanimivo je primerjati tudi porazdelitev števila $Z(G)$ v odvisnosti od p za fiksen $n(G) = 18$ (prikazano na sliki 23, uporabljena je bila različica algoritma 3 s padajočim pregledom). Ker so za zelo majhne p grafi močno nepovezani (imajo veliko komponent), je ničelna prisila velika. Vrednost števila hitro pade, že za $p = 0.1$ je v povprečju približno 7; standardni odklon je tu največji, saj 2 povezavi več lahko pomenita, da je prisila manjša za 2. Nato število $Z(G)$ počasi spet raste, saj imajo vozlišča vse več sosedov. Standardni odklon postaja vse manjši, saj z vse večjo verjetnostjo izbiramo grafe s čedalje več povezavami in je prisila vse pogosteje maksimalna (torej $n - 1$).

Prikazana je tudi škatla z brki, ki potrdi, da je odklon za začetne, manjše p večji, maksimumi in minimumi pa so bolj ekstremni.

6.4 NP-polnost

V prejšnjem podpoglavju smo si pogledali algoritem z eksponentno časovno zahtevnostjo za reševanje problema števila ničelne prisile za splošen graf. Ali lahko najdemo hitrejši, polinomski algoritem? V tem poglavju bomo naredili pregled rezultatov o kompleksnosti problema števila ničelne prisile in videli, da je NP-težek; če bi torej obstajal polinomski algoritem za naš problem, bi sledilo P = NP.

Formalno zapišimo naš optimizacijski problem, označimo ga z ZF-Opt:

$$\text{za dan graf } G\colon \min |Z|,\ Z \text{ množica ničelne prisile } G.$$

Na odločitveni problem (označimo ga z ZF-Odl) ga prevedemo, če se namesto minimizacije velikosti množice ničelne prisile vprašamo: ali za dan graf G in dano število $k \in \mathbb{N}$ obstaja množica ničelne prisile velikosti manjše ali enake k? Če je ta odločitveni problem NP-poln, potem je ZF-Opt problem NP-težek – če ne poznamo

polinomskega algoritma, ali obstaja množica ničelne prisile neke velikosti ali manj, potem tudi ne poznamo polinomskega algoritma, ki nam bi vrnil najmanjšo.

NP-polnost problema ZF-Odl so v [16] dokazali s pomočjo enega izmed problemov iskanja v grafih, natančneje hitrega mešanega iskanja. Vsi problemi iskanja v grafih so zastavljeni približno tako: v grafu obstaja *ubežnik*, ki se lahko skriva na vozliščih ali povezavah, in se lahko premika vzdolž poti v grafu, ki ne vsebujejo *iskalcev*. Iskalci se lahko na nek način, odvisen od tipa problema, premikajo po grafu in s tem povezave in vozlišča proglasijo za *preiskane*. Podobno kot pri pravem lovu na ubežnika je cilj iskalcev, da ga "stisnejo v kot", oziroma bolj formalno, da preiščejo cel graf tako, da se na koncu ubežnik ne more premakniti več nikamor. Za nek dan graf G nas zanima, koliko iskalcev moramo postaviti na vozlišča grafa, da lahko preiščejo cel graf.

Podrobneje si poglejmo, kako izgleda *hitro mešano iskanje* (prvič definirano v [26]). Ubežnik se lahko skriva na vozliščih ali povezavah, kadarkoli se lahko premakne vzdolž katerekoli poti v grafu, ki ne vsebuje iskalcev. Povezava (ali vozlišče) je *okužena*, če se tam morda skriva ubežnik; če pa vemo, da se tam ne more skrivati, je povezava (ali vozlišče) *preiskana*. Z iskalci želimo pregledati vse povezave na tak način, da se kasneje ubežnik ne more premakniti nazaj tja: po zadnji potezi morajo biti vse povezave preiskane. Z iskalcem lahko na vsakem koraku naredimo eno izmed naslednjega:

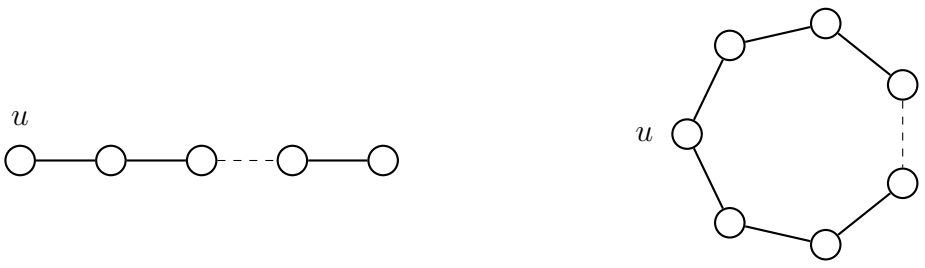
1. *zdrsnemo* iz trenutnega vozlišča u po povezavi v do vozlišča v, če je vozlišče v okuženo in so vse druge povezave, katerih je u krajišče, preiskane;

2. ga *premaknemo* na okuženo vozlišče.

Pri tem upoštevamo, da je na vsakem vozlišču lahko največ en iskalec. Povezavo proglasimo za preiskano, ko po njej zdrsi ali se na obeh krajiščih povezave nahaja iskalec. Če so preiskane vse povezave grafa, pravimo, da je graf preiskan (in ubežnik je ujet).

Za ta problem lahko sedaj definiramo število hitrega mešanega iskanja in si oglejmo njegovo vrednost na dveh preprostih primerih, poti in ciklu.

Definicija 6.7. *Število hitrega mešanega iskanja*, označimo $\mathrm{fms}(G)$, je najmanjše število iskalcev, ki jih moramo uporabiti, da lahko preiščemo cel graf G (da so po zadnji potezi preiskane vse povezave).



Slika 24: Prikazana sta grafa P_n (levo) in C_n (desno), za katera velja $\mathrm{fms}(P_n) = 1$ in $\mathrm{fms}(C_n) = 2$.

Primer 6.8. Iščemo število hitrega mešanega iskanja za pot P_n (na sliki 24 levo). Recimo, da začnemo z enim iskalcem, ki ga na začetku postavimo v levo krajišče u. Iskalec lahko zdrsi eno v desno, saj ima trenutno vozlišče samo eno okuženo povezavo, ki s tem postane preiskana. Tako lahko nadaljuje vsako potezo, saj je povezava levo od njega preiskana, desno od njega pa (edina) okužena. (Opazimo, da so povezave levo od iskalca nedosegljive za ubežnika, saj mu na poti stoji iskalec.) En iskalec je torej dovolj, da preiščemo cel graf, velja $\mathrm{fms}\,(P_n) = 1$.

Kaj pa, če postavimo enega iskalca v vozlišče u na ciklu C_n (na sliki 24 desno)? Ta ne more začeti drseti, saj ima dve okuženi povezavi. Recimo, da ga začnemo premikati po okuženih vozliščih gor. Potem povezave, ki jih je že obiskal, po definiciji niso preiskane, kar vidimo tudi iz tega, lahko ubežnik pride do njih po drugi polovici cikla. En iskalec torej ne bo dovolj, saj mu ubežnik lahko vedno pobegne "malo naprej po ciklu". Če pa dodamo še enega iskalca v vozlišče, sosednje u, je vmesna povezava preiskana. Zato lahko začne vsak iskalec drseti po ciklu v svojo smer, slej ko prej torej preiščeta vse povezave in s tem cel graf. Velja $\mathrm{fms}\,(C_n) = 2$.

Opazimo lahko, da se vrednosti v primeru 6.8 ujemata z vrednostmi števila ničelne prisile za ti dve družini grafov. Pravzaprav lahko vidimo, da je pravilo, kdaj je vozlišče preiskano, na nek način ekvivalentno pravilu za spremembo barve vozlišča: iskalec zdrsi po povezavi (iz preiskanega vozlišča u v okuženo vozlišče, ki postane preiskano) samo, če so vse ostale povezave neokužene oziroma preiskane – podobno, kot črno vozlišče u povzroči spremembo barve v nekem belem vozlišču samo, če so vsi ostali njegovi sosedi črne barve. Intuitivno se zdi, da sta števili morda povezani; brez dokaza navedimo

Izrek 6.9 ([16, izrek 3.1])**.** *Za poljuben graf G velja* $\mathrm{fms}\,(G) = Z(G)$.

Posledica 6.10. ZF-Odl *je* NP*-poln.*

Dokaz. V [26] je dokazano, da je problem hitrega mešanega iskanja (ali za dan graf G velja $\mathrm{fms}\,(G) \leq k$) NP-poln. Iz izreka 6.9 torej sledi NP-polnost problema ZF-Odl. □

V [8] je obravnavana zahtevnost izračuna ožjega problema, *povezane ničelne prisile*, pri kateri mora induciran podgraf nad vozlišči množice ničelne prisile tvoriti povezan graf.

Definicija 6.11. Število povezane ničelne prisile je definirano kot

$$Z_c(G) = \min\{|Z| \mid Z \text{ množica ničelne prisile in } G[Z] \text{ povezan}\}$$

Iz definicij je razvidno, da velja $Z(G) \leq Z_c(G)$. V splošnem iz NP-polnosti širšega problema ne sledi NP-polnost ožjega, saj zoženje morda poenostavi problem. V primeru povezane ničelne prisile pa se izkaže, da je tudi ožji problem (njegova odločitvena različica) NP-poln:

Izrek 6.12 ([8, izrek 3])**.** *Če* ZF-Odl *zožimo samo na tiste množice ničelne prisile, ki inducirajo povezan podgraf, je problem še vedno* NP*-poln.*

Poglejmo si hiter oris dokaza zgornjega izreka. Spomnimo se definicije NP-polnosti (6.3): veljati mora, da je odločitveni problem v NP in da lahko vsak problem iz NP v polinomskem času prevedemo na tega. Povezana različica ZF-Odl je v NP, saj obstaja certifikat – ta je kar množica ničelne prisile Z, velikosti manjše ali enake k – ki je polinomske dolžine glede na vhodne podatke ($O(n)$) in je preverljiv v polinomskem času (uporabimo algoritem 2 in BFS za preverjanje povezanosti induciranega podgrafa $G[Z]$).

Sedaj NP-polnost dokažemo tako, da že znan NP-poln problem ZF-Odl prevedemo na povezanega: vhodni podatek, graf $G = (V, E)$, modificiramo tako, da mu dodamo še eno vozlišče v, s katerim povežemo vsa vozlišča iz V, nato pa dodamo še dve vozlišči, ki ju povežemo z v. Nov graf označimo z G'. Sedaj trdimo, da ima G množico ničelne prisile velikosti manjše ali enake k natanko tedaj, ko ima G' povezano množico ničelne prisile velikosti manjše ali enake $k + 2$. Podrobnosti dokaza te ekvivalence so zapisane v [8]. Opazimo še, da je naša prevedba res polinomske časovne zahtevnosti glede na vhodne podatke ZF-Odl.

Poglejmo si še eno različico ničelne prisile, kjer za vhod dobimo utežen graf G. Vsakemu vozlišču priredimo neko utež s pomočjo funkcije $w\colon V(G) \to \mathbb{R}$. Namesto velikosti z množice ničelne prisile Z bomo sedaj upoštevali težo: $w(Z) = \sum_{i=1}^{z} w(u_i)$, število ničelne prisile sedaj definiramo kot

$$Z(G, w) = \min\{w(Z) \mid Z \text{ množica ničelne prisile grafa } G\}.$$

Izrek 6.13 ([1]). *Problem* ZF-Odl *je* NP*-poln za ravninske grafe z utežmi, tudi če so uteži kvečjemu* 0 *ali* 1.

Posledično sledi tudi NP-polnost uteženega problema za splošne grafe. To ni presenetljivo, saj je ZF-Odl za neutežene grafe ožji problem (predstavljamo si lahko, da imajo vse uteži vrednosti 1), za katerega vemo, da je NP-poln, torej mora tak biti tudi širši problem. Ker je bila NP-polnost uteženega problema dokazana 8 let pred neuteženim, pa je dokaz ponovno narejen s pomočjo polinomske prevedbe že znanega NP-polnega problema, usmerjenega Hamiltonskega cikla, na naš problem. (Da je problem v NP, pa pokažemo enako, kot za primer povezane ničelne prisile.)

Avtor v [1, izrek 2.3.8] za utežen problem dokaže tudi, da je za poljuben fiksen $\epsilon > 0$ NP-težko najti približek rešitve $Z(G, w)$ na $2^{\log^{1-\epsilon} n}$ natančno. Če bi torej obstajal aproksimativen algoritem s tako konstanto, bi sledilo P = NP, zato pričakujemo, da takšni aproksimativni algoritmi ne obstajajo.

Čeprav je problem ZF-Odl NP-poln, pa polinomski algoritmi morda obstajajo za specifično družino ali tip grafa. Za drevesa smo v poglavju 3 dokazali, da je število ničelne prisile enako kar velikosti minimalnega pokritja s potmi. Ker za drevesa poznamo linearen algoritem za izračun minimalnega pokritja [18], imamo zaradi trditve 3.8 torej tudi linearen algoritem za ničelno prisilo dreves.

7 Pregled ostalih rezultatov o ničelni prisili

V tem poglavju naredimo hiter pregled ostalih rezultatov, povezanih s pojmom ničelne prisile. V prvem podpoglavju se osredotočimo na boljše zgornje in spodnje meje za grafe, ki ustrezajo nekaterim pogojem glede največje in najmanjše stopnje

vozlišča, v drugem podpoglavju pa si ogledamo, kako je ničelna prisila povezana z nekaterimi drugimi parametri na grafih, kot so dominacijsko število, neodvisnostno število in velikost največjega prirejanja.

7.1 Boljša zgornja in spodnja meja

Poglejmo si posplošitev ničelne prisile, ki je bila uvedena v [3] in s pomočjo katere so bile v [3, 19] dokazane boljše zgornje meje za osnovno ničelno prisilo.

Definicija 7.1. *Število k-prisile* grafa G za pozitivno celo število k, označimo $F_k(G)$, je velikost najmanjše take množice $Z \subseteq V$, da velja: če na začetku s črno pobarvamo vozlišča iz Z, ostala pa so bela, in upoštevamo pravilo, da črno vozlišče z največ k belimi sosedi povzroči spremembo barve vseh teh, je po koncu postopka pobarvan cel graf G. Vsem takim množicam Z, kjer je po koncu širjenje barve pobarvan cel graf, pravimo množice k-prisile.

Ničelna prisila je poseben primer k-prisile za $k = 1$. Na podoben način kot (2.3) lahko sklepamo spodnjo mejo za k-prisilo:

$$F_k(G) \geq \min\{1, \delta - k + 1\}. \tag{7.1}$$

Če imamo namreč graf z najmanjšo stopnjo δ in poskusimo sestaviti množico k-prisile velikosti $\delta - k$, za neko črno vozlišče u torej vemo, da ima vsaj δ sosedov in da jih je pobarvanih največ $\delta - k - 1$, torej je najmanj $k + 1$ sosedov belih in širjenje barve se ne more začeti.

Poglejmo si vrednosti števila k-prisile za nekaj osnovnih družin grafov. Za pot očitno velja $F_k(P_n) = Z(P_n) = 1$, saj je to najmanjša možna velikost množice k-prisile. Za $k \geq 2$ je tudi za cikel dovolj, če začetna množica vsebuje samo eno vozlišče, saj ima to vozlišče dva bela soseda, ki za $k \geq 2$ oba postaneta črna, nato pa se širjenje barve nadaljuje kot pri ničelni prisili. Velja torej

$$F_k(C_n) = \begin{cases} 2, & k = 1; \\ 1, & \text{sicer.} \end{cases}$$

Za polni graf pa velja $F_k(K_n) = n - k$, saj ima neko (vsako) črno vozlišče torej k belih sosedov in jim spremeni barvo. Očitno je, da za splošen graf velja zgornja meja

$$F_k(G) \leq n(G) - k.$$

Očitno je neka množica k-prisile Z tudi množica $(k + 1)$-prisile: če lahko za začetno množico Z s pravilom, da črno vozlišče s k belimi sosedi spremeni barvo vsem tem, pobarvamo cel graf, ga lahko še toliko bolj, če ima lahko črno vozlišče iz pravila širjenja barve še enega belega soseda več. Vsaka množica k-prisile je tudi množica $(k + 1)$-prisile, obratno pa seveda ni nujno res; ker iščemo minimum in je množic k-prisile torej manj, velja

$$F_k(G) \geq F_{k+1}(G)$$

za nek graf G.

Navedimo zgornje meje za število k-prisile, dokazane v [3], in si poglejmo, kako se poenostavijo v primeru ničelne prisile.

Izrek 7.2 ([3, izrek 3.4]). *Naj bo k pozitivno celo število. Imejmo graf G z $n \geq 2$ vozlišči in naj za stopnje vozlišč velja $\Delta \geq k, \delta \geq 1$. Potem za število k-prisile velja naslednja zgornja meja*

$$F_k(G) \leq \frac{(\Delta - k + 1)n}{\Delta - k + 1 + \min\{\delta, k\}}. \tag{7.2}$$

Posledica 7.3. *Naj bo k pozitivno celo število. Imejmo graf G z $n \geq 2$ vozlišči in naj za najmanjšo stopnjo vozlišča velja $\delta \geq k$. Potem za število k-prisile velja naslednja zgornja meja*

$$F_k(G) \leq \frac{(\Delta - k + 1)n}{\Delta + 1},$$

ki je tesna.

Dokaz. Tesnost zgornje meje dobimo, če vzamemo polni graf $K_{\Delta+1}$, $\Delta \geq k$, ki je Δ-regularen graf (vsa vozlišča so stopnje Δ). Če vstavimo, dobimo

$$F_k(K_{\Delta+1}) \leq \frac{(\Delta - k + 1)(\Delta + 1)}{\Delta + 1} = \Delta - k + 1,$$

hkrati pa iz $\delta = \Delta$, $\Delta \geq k$ in (7.1) sledi

$$F_k(K_{\Delta+1}) \geq \Delta - k + 1,$$

torej velja enakost in zgornja meja je tesna. □

Posledica 7.4. *Za graf G z vsaj dvema vozliščema in najmanjšo stopnjo $\delta \geq 1$ (nimamo izoliranih vozlišč) za ničelno prisilo velja naslednja zgornja meja*

$$Z(G) = F_1(G) \leq \frac{\Delta}{\Delta + 1} n(G), \tag{7.3}$$

ki je tesna za $K_{\Delta+1}$ z $\Delta \geq 1$.

Za primera $k = 1$ in $k = 2$ pa obstaja še boljša meja, če predpostavimo povezanost grafa G. Graf je *k-povezan*, če ima več kot k vozlišč in ob odstranitvi poljubne podmnožice manj kot k vozlišč preostanek inducira povezan podgraf.

Izrek 7.5 ([3, izrek 4.4]). *Naj bo k pozitivno celo število. Imejmo k-povezan graf G z $n > k$ vozlišči in $\Delta \geq 2$. Potem velja*

$$F_k(G) \leq \frac{(\Delta - 2)n + 2}{\Delta + k - 2}, \tag{7.4}$$

meja je tesna.

Tesnost meje ponovno vidimo, če vzamemo graf $K_{\Delta+1}$ z $\Delta \geq 2$ in je $k = 1$ ali $k = 2$. Dobimo

$$F_k(G) \leq \frac{(\Delta - 2)(\Delta + 1) + 2}{\Delta + k - 2} = \frac{\Delta^2 - \Delta}{\Delta + k - 2} = \frac{\Delta(\Delta - 1)}{\Delta + k - 2},$$

za $k = 1$ torej dobimo Δ, kar vemo, da je enako $Z(K_{\Delta+1})$, za $k = 2$ pa $\Delta - 1$, kar je enako $F_2(K_{\Delta+1})$, meji sta v teh primerih tesni.

Če je $k \geq 3$, potem nam (7.2) da boljšo mejo kot (7.4):

$$\frac{(\Delta - k + 1)n}{\Delta - k + 1 + \min\{\delta, k\}} \leq \frac{(\Delta - 2)n}{\Delta - k + 1 + k} \leq \frac{(\Delta - 2)n}{\Delta + 1} \leq \frac{(\Delta - 2)n}{\Delta + k - 2} \leq \frac{(\Delta - 2)n + 2}{\Delta + k - 2},$$

pri čemer smo upoštevali trivialni neenakosti $1-k \leq -2$ (pri prvi zgornji neenakosti) in $k - 2 \geq 1$ (pri tretji zgornji neenakosti) za $k \geq 3$ ter dejstvo, da iz k-povezanosti grafa sledi $\delta \geq k$ (v nasprotnem primeru bi poiskali vozlišče z minimalno stopnjo $\delta < k$, odstranili vse njegove sosede in dobili izolirano vozlišče, induciran podgraf preostanka ne bi bil povezan, torej bi imeli protislovje s predpostavko k-povezanosti).

Posledica 7.6. *Za povezan graf G z $\Delta \geq 2$ velja*

$$Z(G) = F_1(G) \leq \frac{(\Delta - 2)n(G) + 2}{\Delta - 1}, \tag{7.5}$$

meja je tesna za $K_{\Delta+1}$, $K_{\Delta,\Delta}$ za $\Delta \geq 2$ in cikel C_n.

Dokaz. Tesnost meje za $K_{\Delta+1}$ smo dokazali zgoraj. Za $K_{\Delta,\Delta}$ dobimo

$$Z(K_{\Delta,\Delta}) \leq \frac{(\Delta - 2)2\Delta + 2}{\Delta - 1} = \frac{2(\Delta^2 - 2\Delta + 1)}{\Delta - 1} = \frac{2(\Delta - 1)^2}{\Delta - 1} = 2(\Delta - 1),$$

iz trditve 3.5 pa vemo, da je $Z(K_{\Delta,\Delta}) = 2\Delta - 2$, torej je meja tesna.

Poglejmo si še, kaj dobimo za C_n, kjer je $\Delta = 2$:

$$Z(C_n) \leq \frac{(2 - 2)n + 2}{2 - 1} = 2,$$

torej je meja tesna tudi za cikel. □

Izkaže se, da so grafi $K_{\Delta+1}, K_{\Delta,\Delta}$ in C_n edini, za katere je meja (7.5) dosežena [22, 20]. V [19] je dokazano, da lahko mejo precej izboljšamo, če izključimo te in še kakšne grafe; znebimo se lahko zadnjega člena $\frac{2}{\Delta-1}$.

Izrek 7.7 ([19, izrek 4])**.** *Za povezan graf G z največjo stopnjo $\Delta \geq 3$ velja*

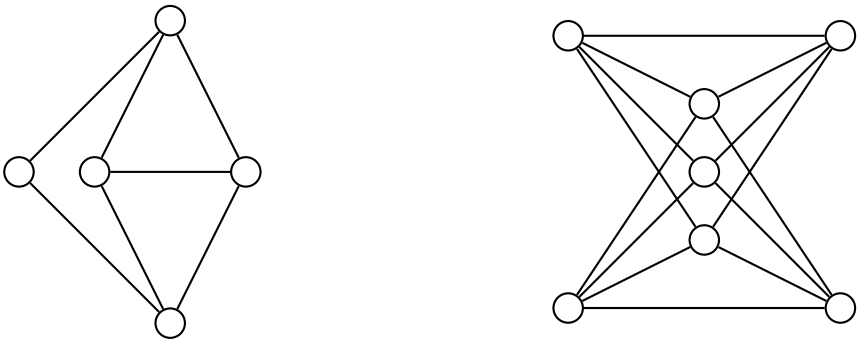
$$Z(G) \leq \frac{\Delta - 2}{\Delta - 1} n(G)$$

natanko tedaj, ko G ni eden izmed $K_{\Delta+1}, K_{\Delta,\Delta}, K_{\Delta-1,\Delta}, G_1, G_2$, kjer sta G_1 in G_2 grafa, prikazana na sliki 25.

Poglejmo še rezultate za spodnjo mejo ničelne prisile.

Trditev 7.8 ([14, izrek 3])**.** *Imejmo graf G z ožino $g \geq 5$ in najmanjšo stopnjo vozlišča $\delta \geq 2$. Potem za ničelno prisilo velja spodnja meja*

$$Z(G) \geq 2\delta - 2.$$



Slika 25: Prikazana sta grafa G_1 (levo) in G_2 (desno), za katera zgornja meja iz izreka 7.7 ne velja.

Pri tem je *ožina* dolžina najkrajšega cikla v grafu, trditev torej obravnava grafe, ki ne vsebujejo trikotnikov ali kvadratov. Ta meja je v [14] izboljšana za grafe, ki imajo kakšno prerezno vozlišče ali povezavo. V istem članku avtorji postavijo splošno domnevo glede spodnje meje:

Domneva 7.9. *Naj bo G graf z ožino $g \geq 3$ in najmanjšo stopnjo $\delta \geq 2$. Potem velja*

$$Z(G) \geq (g-3)(\delta-2)+\delta.$$

Domnevo tudi dokažejo za grafe z ožino $g > 6$ in dovolj velik δ, neodvisen od n. Za $g = 3$ dobimo $Z(G) \geq \delta$, kar je že znana spodnja meja (2.3). Za $g = 4$ je spodnja meja dokazana v [20]: za grafe brez trikotnikov dokažejo, da je spodnja meja res $Z(G) \geq 2\delta - 2$. Za $g = 5$ in $g = 6$ je domneva dokazana v [19]. V [10] jo dokažejo še za vrednosti $g \in \{7, 8, 9, 10\}$ ne glede na velikost najmanjše stopnje $\delta \geq 2$. Domneva je dokončno dokazana v [13], kjer podajo dokaz za $g \geq 5$. Poleg tega je spodnja meja tesna za nekatere družine grafov, npr. cikle in polne dvodelne grafe, pa tudi za nekatere specifične grafe, med drugimi tudi za Petersenov graf [14].

7.2 Povezave z drugimi grafovskimi parametri

Poglejmo si še, kakšne povezave ima ničelna prisila z drugimi grafovskimi pojmi, kot so npr. dominacija in neodvisna množica.

7.2.1 Dominacija

Dominacijska množica D grafa G je taka podmnožica vozlišč, da za vsako vozlišče grafa velja, da je v D ali je sosednje vozlišču iz D (pravimo, da je vsako vozlišče *dominirano*). Velikost najmanjše dominacijske množice je *dominacijsko število* $\gamma(G)$ grafa G. Če dodamo še pravilo, da je vozlišče dominirano tudi, če je edini nedominirani sosed nekega dominiranega vozlišča, in ta postopek ponavljamo, dobimo *električno dominacijo*. Motivacija zanjo je nadzor električnih omrežij, kjer želimo s čim manj nadzornimi napravami pokriti cel sistem.

Definicija 7.10. Množica $S \subseteq$ je *električna dominacijska množica*, če so po koncu apliciranja naslednjih pravil dominirana vsa vozlišča grafa:

1. vsa vozlišča iz $\bigcup_{s \in S} N[s]$ so dominirana,
2. če je vozlišče u dominirano in ima natanko enega soseda v, ki ni dominiran, potem tudi v postane dominiran.

Velikost najmanjše take množice je *električno dominacijsko število* $\gamma_P(G)$ grafa G.

Očitno je problem povezan z ničelno prisilo, saj je 2. pravilo ekvivalentno širjenju barve. Opazimo, da za graf G velja

$$S \text{ električna dominacijska množica} \iff N[S] \text{ množica ničelne prisile.}$$

Za veliko družin grafov je težko dobiti spodnjo mejo električnega dominacijskega števila (dolgi, tehnični dokazi), zato je zaželena splošna spodnja meja, ki jo dobimo s pomočjo ničelne prisile. Pri tem nam pride prav naslednja lema:

Lema 7.11 ([15, lema 2])**.** *Naj bo* $\{u_1, u_2, \dots u_k\}$ *električna dominacijska množica grafa* G *brez izoliranih vozlišč. Potem za ničelno prisilo velja*

$$Z(G) \leq \sum_{i=1}^{k} \deg(u_i).$$

Izrek 7.12 ([5, izrek 3.2])**.** *Imejmo graf* G *z vsaj eno povezavo in največjo stopnjo vozlišča* Δ*. Potem velja*

$$\gamma_P(G) \geq \left\lceil \frac{Z(G)}{\Delta} \right\rceil,$$

meja je tesna.

Dokaz. Imejmo električno dominacijsko množico $\{u_1, u_2, \dots u_k\}$, pri čemer je $k = \gamma_P(G)$. Če graf G nima izoliranih vozlišč, potem po lemi 7.11 velja

$$Z(G) \leq \sum_{i=1}^{k} \deg(u_i) \leq k\Delta = \gamma_P(G)\Delta,$$

delimo z Δ in upoštevamo, da je $\gamma_P(G)$ celo število, izrek sledi.

Če ima graf G izolirana vozlišča, potem vsako izolirano vozlišče prispeva 1 k ničelni prisili in k električni dominaciji. Če vzamemo podgraf brez izoliranih vozlišč, potem zgornja neenakost velja, ko pa izolirana vozlišča dodamo nazaj, se z vsakem dodanim izoliranim vozliščem leva stran ($Z(G)$) poveča za 1, desna pa za Δ, torej neenakost še vedno velja in rezultat drži tudi za grafe z izoliranimi vozlišči.

Meja je tesna za polne grafe K_n, saj velja $Z(K_n) = \Delta(K_n) = n-1$, za električno dominacijsko število pa je očitno, da velja $\gamma_P(K_n) = 1$, saj je vsako vozlišče sosedno vsem ostalim. □

S pomočjo te povezave avtorji v [5] določijo električno dominacijsko število in število ničelne prisile (oz. meje) za nekatere produkte znanih družin oziroma splošnih grafov, med drugim npr. ničelno prisilo za leksikografski produkt med dvema regularnima grafoma, kjer ima prvi nizko dominacijsko število, drugi pa nizko električno dominacijsko število.

Oglejmo si še en tip dominacije, za katerega se izkaže, da je povezan z ničelno prisilo. Povezana dominacijska množica grafa G je taka dominacijska množica, ki inducira povezan podgraf v grafu G. *Povezano dominacijsko število* $\gamma_c(G)$ je torej velikost najmanjše take množice za nek graf G.

Dominacijo lahko (podobno kot prisilo v 7.1) posplošimo: za k-dominacijsko množico D grafa G velja, da je vsako vozlišče, ki ni v D, sosednje vsaj k vozliščem iz D. Velikost najmanjše take množica je *k-dominacijsko število* grafa G, označimo $\gamma_k(G)$. Za $k = 1$ očitno dobimo navadno dominacijo.

Sedaj lahko definiramo posplošeno verzijo povezane dominacije: povezana k-dominacijska množica je taka k-dominacijska množica, ki inducira povezan podgraf, *povezano k-dominacijsko število* $\gamma_{k,c}(G)$ pa velikost najmanjše take množice.

Izkaže se, da velja naslednja povezava med k-prisilo in k-dominacijo:

Lema 7.13 ([3, lema 4.1])**.** *Naj bo G k-povezan graf za neko pozitivno celo število $k < n(G)$. Naj bo F najmanjša taka množica k-prisile, da je $G[V \setminus F]$ povezan podgraf. Potem je $V \setminus F$ povezana k-dominacijska množica grafa G.*

Posledično lahko dokažemo sledečo zgornjo mejo k-prisile:

Posledica 7.14 ([3, posledica 4.2])**.** *Naj bo G k-povezan graf za neko pozitivno celo število $k < n(G)$. Potem za k-prisilo in k-dominacijo tega grafa velja naslednja povezava:*

$$F_k(G) \leq n(G) - \gamma_{k,c}(G).$$

Dokaz. Naj bo F najmanjša taka množica k-prisile, da je $G[V \setminus F]$ povezan podgraf. (Taka množica k-prisile vedno obstaja, saj lahko za F vzamemo vsa vozlišča, razen enega, torej $V \setminus F$ inducira trivialen povezan podgraf.) Očitno velja $F_k(G) \leq |F|$, saj F ustreza dodatnim omejitvam povezanosti. Po lemi 7.13 je $V \setminus F$ povezana k-dominacijska množica grafa G, velja torej $\gamma_{k,c}(G) \leq |V \setminus F| = n(G) - |F|$. Če preuredimo, dobimo $|F| \leq n(G) - \gamma_{k,c}(G)$, skupaj s prvo neenakostjo torej

$$F_k(G) \leq |F| \leq n(G) - \gamma_{k,c}(G).$$

□

Za ničelno prisilo in povezano dominacijo torej sledi

Posledica 7.15. *Za povezan graf G z $n \geq 2$ vozlišči velja*

$$Z(G) = F_1(G) \leq n - \gamma_c(G),$$

meja je tesna.

Dokaz. Meja je tesna za polni graf K_n, za katerega velja $Z(K_n) = n-1$ in $\gamma_c(G) = 1$ (vemo, da je $\gamma(G) = 1$, eno vozlišče pa je trivialno povezan podgraf). □

(S pomočjo te povezave med k-prisilo in k-dominacijo, natančneje z lemo 7.13, je v [3] dokazana zgornja meja k-ničelne prisile za k-povezane grafe (7.4), navedena v prejšnjem podpoglavju.)

Še en pojem iz dominacije je *Grundyjevo dominacijsko število*: to je dolžina najdaljšega dominacijskega zaporedja za graf G, označimo $\gamma_{gr}(G)$. *Dominacijsko*

zaporedje grafa G je tako zaporedje njegovih vozlišč, da vsako vozlišče zaporedja dominira vsaj eno prej še nedominirano vozlišče. Zapisano formalno, če imamo dominacijsko zaporedje $S = (u_1, \ldots, u_k)$, potem je $\{u_1, \ldots, u_k\}$ dominacijska množica in za vsako vozlišče u_i v zaporedju mora držati

$$N[u_i] \setminus \bigcup_{j=1}^{i-1} N[u_j] \neq \emptyset.$$

Povedano drugače, vozlišče u, ki ga dodamo v zaporedje S, mora dominirati vsaj eno izmed vozlišč, ki ga ne dominira nobeno izmed prejšnjih v zaporedju, ali pa u ne sme biti dominirano, preden ga dodamo. (Tukaj smo malo priredili uporabo besede "dominira", da dodano vozlišče u dominira tudi samega sebe, kar pa ne spremeni ničesar.)

Ena izmed različic Grundyjeve dominacije je *Z-Grundyjevo dominacijsko število*, označimo $\gamma_{gr}^Z(G)$, ki je dolžina najdaljšega Z-dominacijskega zaporedja, kar pomeni, da mora vsako vozlišče v zaporedju dominirati vsaj enega do zdaj še ne dominiranega soseda. (Razlika je v tem, da vozlišče samo tukaj ne šteje.) Če zapišemo formalno, za Z-dominacijsko zaporedje $S = (u_1, \ldots, u_k)$ velja, da je $\{u_1, \ldots, u_k\}$ dominacijska množica (rekli ji bomo tudi *Z-množica*) in za vsako vozlišče u_i v zaporedju mora držati

$$N(u_i) \setminus \bigcup_{j=1}^{i-1} N[u_j] \neq \emptyset.$$

Ker je ta različica bolj omejena kot prva in iščemo najdaljše zaporedje, očitno velja

$$\gamma_{gr}^Z \leq \gamma_{gr}. \tag{7.6}$$

Izkaže se [7], da je Z-Grundyjeva dominacija povezana z ničelno prisilo (zato ima v imenu črko "Z", ki pride iz "zero forcing"):

Izrek 7.16 ([7, izrek 2.2])**.** *Za graf G brez izoliranih vozlišč velja*

$$\gamma_{gr}^Z + Z(G) = n(G).$$

Pri tem je komplement najmanjše množice ničelne prisile grafa G največja Z-množica grafa G, in obratno.

Zaradi povezave (7.6) med Grundyjevo in Z-Grundyjevo dominacijo iz zgornjega izreka sledi

Posledica 7.17. *Za graf G brez izoliranih vozlišč velja*

$$\gamma_{gr}(G) + Z(G) \geq n(G).$$

Avtorji [7] s pomočjo izreka 7.16 določijo vrednosti ničelne prisile nekaterih produktov grafov znanih družin, med drugim krepkega produkta med ciklom in potjo:

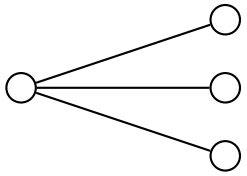
Trditev 7.18 ([7, trditev 2.5.])**.** *Za $n \geq 3$ in $m \geq 2$ velja*

$$Z(C_n \boxtimes P_m) = 2m + n - 2.$$

7.2.2 Neodvisnostno število in največje prirejanje

Izkaže se, da lahko za kubične grafe brez krempljev izboljšamo zgornjo mejo (7.5), kar sledi iz povezave ničelne prisile in neodvisnostnega števila. Poglejmo si osnovne pojme, da lahko navedemo rezultate, predstavljene v [12].

Graf je kubičen, če je 3-regularen; velja torej, da so stopnje vseh vozlišč v grafu enake 3. Graf je brez krempljev, če ne vsebuje grafa $K_{1,3}$ (prikazan na sliki 26) kot podgrafa.



Slika 26: Prikazan je graf $K_{1,3}$ (krempelj).

Neodvisna množica (vozlišč) grafa G je taka množica, da za vsak par vozlišč v njej velja, da sta neodvisna (nista soseda). Velikost največje take množice je *neodvisnostno število* grafa G, označimo z $\alpha(G)$. Podobno, *prirejanje* grafa G je taka množica povezav, ki so paroma neodvisne (nimajo skupnih krajišč). Velikost največjega prirejanja grafa G označimo z $\alpha'(G)$.

Iz zgornje meje ničelne prisile (7.5) za povezan graf sledi

Posledica 7.19. *Za povezan kubičen graf G brez krempljev velja*

$$Z(G) \leq \frac{1}{2}n(G) + 1.$$

To mejo se da še malo izboljšati, če izvzamemo graf K_4:

Izrek 7.20 ([11, izrek 3])**.** *Za povezan kubičen graf $G \neq K_4$ brez krempljev velja*

$$Z(G) \leq \frac{1}{2}n(G),$$

meja je tesna natanko tedaj, ko je G graf $C_3 \,\square\, K_2$ ali N_2 (prikazana na sliki 27).

Slika 27: Prikazana sta prizma $C_3 \,\square\, K_2$ (levo) in diamantna ogrlica N_2 (desno).

Še bolj pa se zgornjo mejo za tak tip grafa da izboljšati z naslednjim izrekom:

Izrek 7.21 ([12, izrek 5])**.** *Za povezan kubičen graf $G \neq K_4$ brez krempljev velja*

1. $Z(G) \leq \alpha(G) + 1$,
2. $Z(G) \leq \alpha'(G)$.

Meji sta tesni za prizmo $C_3 \square K_2$ in diamantno ogrlico N_2, za kateri velja $Z(G) = \alpha'(G) = \alpha(G) + 1$.

Ker iz [17] vemo, da za neodvisnostno število kubičnega grafa G brez krempljev velja $\alpha(G) \leq \frac{2}{5}n(G)$, sledi naslednja zgornja meja za ničelno prisilo tega tipa grafa:

Posledica 7.22. *Za povezan kubičen graf $G \neq K_4$ brez krempljev velja*

$$Z(G) \leq \frac{2}{5}n(G) + 1.$$

Literatura

[1] A. Aazami, *Hardness results and approximation algorithms for some problems on graphs*, doktorska disertacija, University of Waterloo, 2008.

[2] AIM Minimum Rank – Special Graphs Work Group, *Zero forcing sets and the minimum rank of graphs*, Linear Algebra and its Applications **428** (2008) 1628–1648, doi: 10.1016/j.laa.2007.10.009.

[3] D. Amos in dr., *Upper bounds on the k-forcing number of a graph*, Discrete Applied Mathematics **181** (2015) 1–10, doi: 10.1016/j.dam.2014.08.029.

[4] F. Barioli in dr., *Zero forcing parameters and minimum rank problems*, Linear Algebra and its Applications **433** (2010) 401–411, doi: 10.1016/j.laa.2010.03.008.

[5] K. F. Benson in dr., *Zero forcing and power domination for graph products*, Australasian Journal of Combinatorics **70** (2018) 221–235, `https://ajc.maths.uq.edu.au/pdf/70/ajc_v70_p221.pdf`.

[6] M. Booth in dr., *On the minimum rank among positive semidefinite matrices with a given graph*, SIAM Journal on Matrix Analysis and Applications **30** (2008) 731–740, doi: 10.1137/050629793.

[7] B. Brešar in dr., *Grundy dominating sequences and zero forcing sets*, Discrete Optimization **26** (2017) 66–77, doi: 10.1016/j.disopt.2017.07.001.

[8] B. Brimkov in I. V. Hicks, *Complexity and computation of connected zero forcing*, Discrete Applied Mathematics **229** (2017) 31–45, doi: 10.1016/j.dam.2017.05.016.

[9] D. Burgarth in V. Giovannetti, *Full control by locally induced relaxation*, Physical review letters **99** (2007) 100501, doi: 10.1103/physrevlett.99.100501.

[10] R. Davila in M. A. Henning, *The forcing number of graphs with given girth*, Quaestiones Mathematicae **41** (2018) 189–204, doi: 10.2989/16073606.2017.1376230.

[11] R. Davila in M. A. Henning, *Total forcing and zero forcing in claw-free cubic graphs*, Graphs and Combinatorics **34** (2018) 1371–1384, doi: 10.1007/s00373-018-1934-4.

[12] R. Davila in M. A. Henning, *Zero forcing in claw-free cubic graphs*, Bulletin of the Malaysian Mathematical Sciences Society (2018), doi: 10.1007/s40840-018-00705-5.

[13] R. Davila, T. Kalinowski in S. Stephen, *A lower bound on the zero forcing number*, Discrete Applied Mathematics **250** (2018) 363–367, doi: 10.1016/j.dam.2018.04.015.

[14] R. Davila in F. Kenter, *Bounds for the zero-forcing number of graphs with large girth*, Theory and Applications of Graphs **2** (2015) članek 1, 8 str., doi: 10.20429/tag.2015.020201.

[15] N. Dean in dr., *On the power dominating sets of hypercubes*, v: 14th IEEE International Conference on Computational Science and Engineering, IEEE, 2011, str. 488–491, doi: 10.1109/cse.2011.89.

[16] S. Fallat, K. Meagher in B. Yang, *On the complexity of the positive semidefinite zero forcing number*, Linear Algebra and its Applications **491** (2016) 101–122, doi: 10.1016/j.laa.2015.03.011.

[17] R. J. Faudree in dr., *On independent generalized degrees and independence numbers in K (1, m)-free graphs*, Discrete Mathematics **103** (1992) 17–24, doi: 10.1016/0012-365x(92)90035-e.

[18] D. Franzblau in A. Raychaudhuri, *Optimal Hamiltonian completions and path covers for trees, and a reduction to maximum flow*, The ANZIAM Journal **44** (2002) 193–204, doi: 10.1017/s1446181100013894.

[19] M. Gentner in D. Rautenbach, *Some bounds on the zero forcing number of a graph*, Discrete Applied Mathematics **236** (2018) 203–213, doi: 10.1016/j.dam.2017.11.015.

[20] M. Gentner in dr., *Extremal values and bounds for the zero forcing number*, Discrete Applied Mathematics **214** (2016) 196–200, doi: 10.1016/j.dam.2016.06.004.

[21] C. R. Johnson in A. L. Duarte, *The maximum multiplicity of an eigenvalue in a matrix whose graph is a tree*, Linear and Multilinear Algebra **46** (1999) 139–144, doi: 10.1080/03081089908818608.

[22] L. Lu, B. Wu in Z. Tang, *Proof of a conjecture on the zero forcing number of a graph*, Discrete Applied Mathematics **213** (2016) 233–237, doi: 10.1016/j.dam.2016.05.009.

[23] D. D. Row, *A technique for computing the zero forcing number of a graph with a cut-vertex*, Linear Algebra and its Applications **436** (2012) 4423–4432, doi: 10.1016/j.laa.2011.05.012.

[24] S. Severini, *Nondiscriminatory propagation on trees*, Journal of Physics A: Mathematical and Theoretical **41** (2008) 482002, doi: 10.1088/1751-8113/41/48/482002.

[25] M. Sipser, *Introduction to the theory of computation*, Thomson Course Technology Boston, druga izd., 2006.

[26] B. Yang, *Fast–mixed searching and related problems on graphs*, Theoretical Computer Science **507** (2013) 100–113, doi: 10.1016/j.tcs.2013.04.015.

A Priloga: implementacija algoritmov

Algoritmi so implementirani v programskem jeziku C++, koda je strukturirana takole:

- v `Graph.cpp` je implementirana podatkovna struktura grafa s seznamom sosedov, koda je zaradi jedrnatosti izpuščena;
- v `utils.cpp` so implementirane pomožne funkcije, med drugim tudi algoritmov 1 in 2, koda je zaradi jedrnatosti izpuščena;
- v `check.hpp` so implementirani algoritmi preverjanja (torej 1 in 2), pri čemer algoritem z vrsto sprejme podatkovno strukturo za sledenje belim sosedom;
- v `WhiteNeighbours.cpp` sta implementirani dve podatkovni strukturi za sledenje belim sosedom, ena z množicami, druga s tabelo števcev;
- v `zero_forcing_number.cpp` so implementirane tri različice algoritma 3 za izračun števila ničelne prisile: množice pregledujemo po velikosti padajoče, po velikosti padajoče, a samo podmnožice prejšnjih množic ničelne prisile, in s pomočjo binarnega iskanja.

`check.hpp`

```
#include "Graph.hpp"
#include "WhiteNeighbours.hpp"
#include "utils.hpp"


inline pair<bool,int> check_ZFS_naive(const Graph& graph, const vector<int>& zfs) {
    vector<int> colouring(graph.vertices_num(), WHITE);
    for (int u : zfs) {
        assert_vertex_label_correctness(graph, u);
        colouring[u] = BLACK;
    }
    bool change = true;
    int iter = 0;
    while (change) {
        change = false;
        for (int u=0; u < graph.vertices_num(); ++u) {  // for every vertex
            if (colouring[u] == BLACK) {
                auto neighbours = find_white_neighbours(graph, u, colouring);
                if (neighbours.size() == 1) {
                    int v = neighbours[0];
                    colouring[v] = BLACK;
                    change = true;
                }
            }
        }
        ++iter;
    }
    return pair<bool,int> (all_vertices_black(colouring), iter);
}

```

```cpp
template <typename T>
pair<bool,int> check_ZFS_queue(const Graph& graph, const vector<int>& zfs) {
    vector<int> colouring(graph.vertices_num(), WHITE);
    vector<int> forces(graph.vertices_num(), NOT_FORCED);
    for (int u : zfs) {
        assert_vertex_label_correctness(graph, u);
        colouring[u] = BLACK;
        forces[u] = IN_ZFS;
    }

    queue<int> q;
    T white_neighbours = T(&graph, &colouring);
    // Preprocessing
    for (int u=0; u < graph.vertices_num(); ++u) {
        if (colouring[u] == BLACK && white_neighbours.size(u) == 1) {
            int v = white_neighbours.only_white_neighbour(u);
            if (forces[v] == NOT_FORCED) {
                q.push(v);
                forces[v] = u;
            }
        }
    }

    int iter = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        colouring[u] = BLACK;

        // Check if u has only one white neighbour
        if (white_neighbours.size(u) == 1) {
            int v = white_neighbours.only_white_neighbour(u);
            if (forces[v] == NOT_FORCED) {
                q.push(v);
                forces[v] = u;
            }
        }

        // Remove u from white_neighbours set of all its neighbours and check
        // whether any of those neighbours now have only one white neighbour.
        for (int v : graph.adjacency_lists[u]) {
            white_neighbours.remove(v, u);
            if (colouring[v] == BLACK && white_neighbours.size(v) == 1) {
                int w = white_neighbours.only_white_neighbour(v);
                if (forces[w] == NOT_FORCED) {
                    q.push(w);
                    forces[w] = v;
                }
            }
        }
        ++iter;
    }

    return pair<bool,int> (all_vertices_black(colouring), iter);
}
```

Algoritme za preverjanje kličemo z eno od dveh podatkovnih struktur, implementiranih v `WhiteNeighbours`.

`WhiteNeighbours.hpp`

```
#include "Graph.hpp"


/* Implementation of a data structure for keeping track of white neighbours
using unordered sets. */
class WhiteNeighboursSet {
private:
    vector<unordered_set<int>> white_neighbours;
public:
    WhiteNeighboursSet(const Graph* graph, const vector<int>* colouring);

    void remove(int u, int v);
    int size(int u);
    int only_white_neighbour(int u);
    friend ostream& operator<<(ostream& out, const WhiteNeighboursSet& wn);
};


/* Implementation of a data structure for keeping track of white neighbours
using a vector of counts. */
class WhiteNeighboursCount {
private:
    const Graph* graph;
    const vector<int>* colouring;
    vector<int> white_neighbours_count;
public:
    WhiteNeighboursCount(const Graph* p_graph, const vector<int>* p_colouring);

    void remove(int u, int v);
    int size(int u);
    int only_white_neighbour(int u);
    friend ostream& operator<<(ostream& out, const WhiteNeighboursCount& wn);
};
```

`WhiteNeighbours.cpp`

```
#include "WhiteNeighbours.hpp"
#include "utils.hpp"

WhiteNeighboursSet::WhiteNeighboursSet(const Graph* graph,
const vector<int>* colouring) {
    for (int u=0; u < graph->vertices_num(); ++u) {
        auto neighbours = find_white_neighbours(*graph, u, *colouring);
        white_neighbours.emplace_back(neighbours.begin(), neighbours.end());
    }
}

ostream& operator<<(ostream& out, const WhiteNeighboursSet& wn) {
    for (size_t u=0; u < wn.white_neighbours.size(); ++u) {
        out << u << ": ";
        for (int v: wn.white_neighbours[u]) {
```

```
            out << v << " ";
        }
        out << "\n";
    }
    return out;
}

/* Removes v from white neighbours of u. */
void WhiteNeighboursSet::remove(int u, int v) {
    white_neighbours[u].erase(v);
}

int WhiteNeighboursSet::size(int u) {
    return static_cast<int>(white_neighbours[u].size());
}

int WhiteNeighboursSet::only_white_neighbour(int u) {
    assert(size(u) == 1 && "vertex does not have exactly one white neighbour");
    return pop(white_neighbours[u]);
}

WhiteNeighboursCount::WhiteNeighboursCount(const Graph* p_graph,
const vector<int>* p_colouring): graph(p_graph), colouring(p_colouring) {
    for (int u=0; u < graph->vertices_num(); ++u) {
        auto neighbours = find_white_neighbours(*graph, u, *colouring);
        white_neighbours_count.push_back(neighbours.size());
    }
}

ostream& operator<<(ostream& out, const WhiteNeighboursCount& wnc) {
    for (size_t u=0; u < wnc.white_neighbours_count.size(); ++u) {
        out << u << ": " << wnc.white_neighbours_count[u] << "\n";
    }
    return out;
}

/* Removes v from white neighbour count of u. */
void WhiteNeighboursCount::remove(int u, int v) {
    --white_neighbours_count[u];
}

int WhiteNeighboursCount::size(int u) {
    return white_neighbours_count[u];
}

int WhiteNeighboursCount::only_white_neighbour(int u) {
    assert(size(u) == 1 && "vertex does not have exactly one white neighbour");
    auto neighbours = find_white_neighbours(*graph, u, *colouring);
    return neighbours[0];
}
```

zero_forcing_number.cpp

```
#include "Graph.hpp"
#include "WhiteNeighbours.hpp"
```

```cpp
#include "check.hpp"
#include "zero_forcing_number.hpp"

/* Returns all subsets of the set {1, ..., n} ordered by size (from 0 to n). */
vvvi power_subset(int n) {
    vvvi subsets(n+1);
    for (int i=0; i < (1 << n); ++i) {
        vector<int> subset;
        for (int j=0; j < n; ++j) {
            if ((i & (1 << j)) != 0) {
                subset.push_back(j);
            }
        }
        subsets[subset.size()].push_back(move(subset));
    }
    return subsets;
}

/* Generates subsets of given sets which are smaller by 1. */
vvi smaller_subsets(vvi sets) {
    set<vector<int>> new_sets;
    for (auto set : sets) {
        size_t n = set.size();
        vvi curr_sets(n);
        for (size_t i=0; i < n; ++i) {
            for (size_t j=0; j < n; ++j) {
                if (j != i) curr_sets[j].push_back(set[i]);
                if (i == n-1) new_sets.insert(move(curr_sets[j]));
            }
        }
    }
    return vvi(new_sets.begin(), new_sets.end());
}

int ZFN_top_down(const Graph& graph, const vvvi& subsets) {
    int i;
    for (i = graph.vertices_num()-1; i >= 0; --i) {
        bool zfs_exists = false;
        for (auto subset : subsets[i]) {
            if (check_ZFS_queue<WhiteNeighboursCount>(graph, subset).first) {
                zfs_exists = true;
                break;
            }
        }
        if (!zfs_exists) break;
    }
    return i+1;
}

int ZFN_top_down_subsets(const Graph& graph, const vvvi& subsets) {
    int i;
    vvi eligible_subsets = subsets[graph.vertices_num()-1];
    for (i = graph.vertices_num()-1; i >= 0; --i) {
        auto it = eligible_subsets.begin();
        while (it != eligible_subsets.end()) {
            if (check_ZFS_queue<WhiteNeighboursCount>(graph, *it).first) ++it;
```

```
            else it = eligible_subsets.erase(it);
        }
        if (eligible_subsets.empty()) break;
        eligible_subsets = smaller_subsets(eligible_subsets);
    }
    return i+1;
}

int ZFN_binary_search(const Graph& graph, const vvvi& subsets) {
    int l = 1; int r = graph.vertices_num();
    for (auto subset : subsets[l]) {
        if (check_ZFS_queue<WhiteNeighboursCount>(graph, subset).first) {
            return l;
        }
    }
    while (l < r) {
        int m = static_cast<int>((l+r)/2);
        if (m == l) break;
        bool zfs_exists_mid = false;
        for (auto subset : subsets[m]) {
            if (check_ZFS_queue<WhiteNeighboursCount>(graph, subset).first) {
                zfs_exists_mid = true;
                break;
            }
        }
        if (zfs_exists_mid) r = m;
        else l = m;
    }
    return r;
}
```