

# Iskanje kromatičnega števila grafa: Projekt pri predmetu operacijske raziskave

Lonja Tjaša Tavčar in Žan Šifrer

23. februar 2017

## Najina naloga

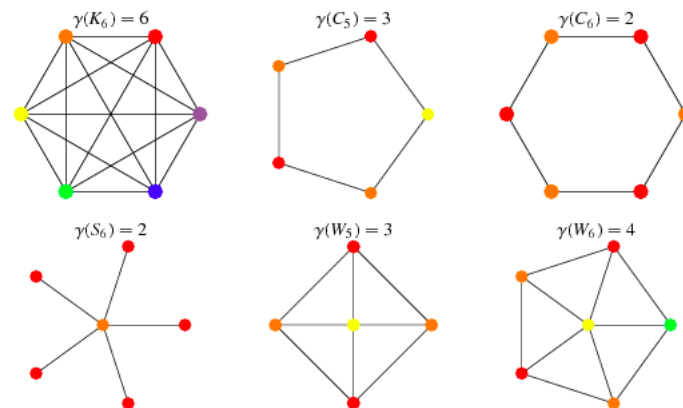
Naloga najinega projekta je generirati grafe, ki so  $k$ -barvni za  $k = 2, 4, \dots$  in poiskati njihova kromatična števila z uporabo celoštevilskega linearnega programa. Analizirati pa morava tudi število grafov, ki jih bova generirala kot  $k$ -barve, a so v resnici  $(k - 1)$ -barvni.

## Splošno o barvanju grafov

Barvanje grafa je preslikava, ki vsakemu vozlišču grafa priredi neko barvo. Vsako barvo pa označimo kar z naravnim številom. Dobro ali pravilno barvanje vozlišč je tako, da so sosednja vozlišča pobarvana z različnimi barvami. Najmanjše število  $k$ , za katerega obstaja dobro  $k$ -barvanje (minimalno barvanje) vozlišč grafa imenujemo kromatično število grafa. Označimo ga z  $\chi(G)$ . Barvanje je minimalno, ko je graf pobarvan s številom barv, ki je enako kromatičnemu številu grafa.

Za nakatere grafe kromatična števila poznamo:

- Polni graf z  $n$  vozlišči ima kromatično število  $n$
- Ciklični graf ima kromatično število 2, če ima sodo število vozlišč ali 3, če ima liho število vozlišč
- Kolo ima kromatično število 3, če ima liho število vozlišč ali 4, če ima sodo število vozlišč



Slika 1: Primeri znanih kromatičnih števil grafov

Pri iskanju kromatičnega števila upoštevamo omejitve:

- $1 \leq \chi(G) \leq n$  (pri čemer je  $n$  število vozlišč grafa)
- edini grafi, ki so lahko 1-barvni so točke
- če je  $H$  podgraf grafa  $G$  je  $\chi(H) \leq \chi(G)$
- če graf vsebuje kliko velikosti  $k$ , potem je potrebnih vsaj  $k$  barv, da pobarvamo to kliko, torej je  $\chi(G) \geq k$ .
- Če graf vsebuje zanko (torej povezavo, ki izstopa iz in vstopa v isto vozlišče), ne more imeti pravičnega barvanja.

## Algoritem za generiranje polnih obarvljivih grafov

S tem algoritmom generiramo grafe, za katere velja, da je vsako vozlišče barve  $i$  povezano z vsemi vozlišči, ki so druge barve. Torej imamo za  $k$ -obarvljiv graf največje možno število povezav. Od teh grafov lahko odstranjujemo povezave. Denimo, da naključno odstranjujemo povezave v grafu: v grafu pogledamo 2 množici vozlišč iste barve (npr. modra in rdeča vozlišča). Če se bo zgodilo, da bomo odstranili vse povezave, ki imajo eno ogljišče modro in eno rdeče, bo graf tako postal  $(k - 1)$ -obarvljiv.

**Dokaz:** Ker ne obstaja več nobena povezava, ki gre iz modrega ogljišča v rdeče, potem lahko vsa ogljišča spremenimo v modro ali obratno in tako dobimo graf obarvan s  $k - 1$  barvo. Kombinacije vozlišč za  $k$ -obarvljiv graf z  $n$  vozlišči.

```

def kombinacije(k,n):
    z=n-k
    A=matrix(1,k)
    komb=[]
    for i in range(k):
        A[0,i]=1
    if z>k:
        a=k
    else:
        a=z
    for j in range(a):
        x=matrix(1,k)
        for m in range(j+1):
            if m==0:
                x[0,m]=(z-j)
            else:
                x[0,m]=1
        komb.append(x+A)
        ali=True
        while ali:
            t=0
            desno=True
            while desno:
                if x[0,0]==x[0,t+1]:
                    t=t+1
                    if t==k-1:
                        desno=False
            else:
                desno=False
        spr2=False
        spr1=True
        for u in range(t+1,k):
            if (spr1)and not(spr2):
                if x[0,u]==0:
                    ali=False
                    spr1=False
            else:
                if (x[0,u]+2)<=x[0,t]:
                    x[0,t]=x[0,t]-1
                    x[0,u]=x[0,u]+1
                    spr2=True

```

```

        if spr2:
            komb.append(x+A)
        else:
            ali=False
    return komb

Za primer kombinacije(4,11) dobimo: [[8 1 1 1], [7 2 1 1], [6 3 1 1], [5 4 1 1],
[6 2 2 1], [5 3 2 1], [4 4 2 1], [4 3 3 1], [5 2 2 2], [4 3 2 2], [3 3 3 2]]

def pretvorivA(G):
    z = 1
    for P in G:
        for j in P:
            if j > z:
                z=j
    A=matrix(z)
    for (i,j) in G:
        A[(i-1),(j-1)]=1
        A[(j-1),(i-1)]=1
    return A

def pretvorivG(A):
    x=1
    G=[]
    for z in A:
        if x==1:
            m=len(z)
            x+=1
        for i in range(m):
            for j in range(i,m):
                if A[i,j]==1:
                    G.append(((i+1),(j+1)))
    return G

def generiraj_polni(M):
    n=0
    k=0
    for i in M:
        k = k + 1
        n = n + i
    X=matrix(k,n)
    s=0

```

```

for j in range(k):
    for z in range(M[j]):
        X[j,s]=1
        s=s+1
G=[]
for d in range(k):
    for e in range(n):
        if X[d,e]==1:
            for f in range(n):
                if X[d,f]==0:
                    G=G+[((e+1),(f+1)))]
A=pretvorivA(G)
return(A)

```

## Linearni program:

Za iskanje kromatičnega števila sva napisala celoštevilski linearni program. Najprej sva definirala binarni spremenljivki z naslednjima pomenoma:

$$y_k = \begin{cases} 1, & \text{če uporabimo barvo } k \\ 0, & \text{sicer} \end{cases}$$

in še:

$$x_{i,k} = \begin{cases} 1, & \text{če je vozlišče } i \text{ barve } k \\ 0, & \text{sicer} \end{cases}$$

V najinem primeru sta  $k$  in  $i$  števili od 1 do  $n$ , saj  $i$  predstavlja število vozlišč, ki jih je  $n$ ;  $k$  pa predstavlja število barv, ki jih uporabimo. V najslabšem primeru je vsako vozlišče svoje barve, torej uporabimo  $n$  barv in v tem primeru je  $k = n$ .

Ker iščeva najmanjše število barv za graf, mora najin program iskati minimum.

$$\min \sum_{k=1}^n y_k$$

p.p.

$$\sum_{k=1}^n x_{ik} = 1, \quad \text{za} \quad i = 1, \dots, n \quad (1)$$

$$x_{i,k} - y_k \leq 0, \quad \text{za} \quad i, k = 1, \dots, n \quad (2)$$

$$x_{i,k} + x_{j,k} \leq 1, \quad \text{kjer} \quad (i, j) \in E(G), k = 1, \dots, n \quad (3)$$

$$0 \leq x_{i,k}, y_k \leq 1 \quad (4)$$

$$x_{i,k}, y_k \in \mathbb{Z} \quad (5)$$

Pogoj (1) pomeni, da je vsako vozlišče neke barve, in da je lahko samo ene barve. Pogoj (2) pomeni, da vozlišče  $i$  dobi barvo  $k$  samo, če je barva  $k$  sploh uporabljena (torej v primeru, ko je  $y_k = 1$ ). Naslednji pogoj (3) zagotovi, da sosednji vozlišči ne smeta biti iste barve. Z zadnjima pogojema (4) in (5) pa si zagotovimo, da sta  $x$  in  $y$  binarni spremenljivki in celi števili.

### Časovna zahtevnost algoritma

Ker so  $y$  spremenljivke odvisne od  $x$ , bi časovno zahtevnost algoritma ocenila na podlagi  $x$  spremenljivk.

Denimo, da imamo graf z  $n$  vozlišči. Ko skonstruiramo spremenljivke  $x_{i,k}$  dobimo  $n^2$  spremenljivk in ker so spremenljivke enake 1 ali 0 imamo za vsako 2 možnosti, torej je vseh možnih kombinacij za  $x$   $2^{n^2}$ , če pa bi gledali vse možne kombinacije bi morali za vsako preveriti  $O(n)$  omejitev. Torej, če bi recimo sestavili celoštevilski program, ki preveri vse možnosti bi bila časovna zahtevnost  $O(n * 2^{n^2})$ . To pomeni, da bi se pri takem algoritmu izogibali grafov z velikim številom vozlišč. Sicer pa je časovna zahtevnost, seveda odvisna od sestave celoštevilskega linearne programa in je dosti manjša, vendar še vedno narašča eksponentno.

## Uporaba algoritmov za preverjanje kromatičnega števila grafa v praksi

V resničnem življenju lahko veliko problemov prevedemo na iskanje kromatičnega števila grafa, kar nam pomaga pri iskanju optimalne rešitve problema.

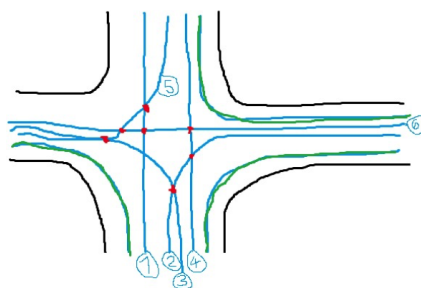
Nekaj primerov:

(1) Denimo, da imamo več radijskih postaj, ki se med sabo sporazumevajo. Postaje imamo razporejene po zemljevidu, in če sta 2 radijski postaji dovolj blizu ena drugi, se lahko motita. Torej za vozlišča uporabimo radijske postaje, povezave pa naredimo med tistimi postajami, ki so si dovolj blizu. Zdaj iščemo *vsaj koliko različnih radijskih frekvenc*

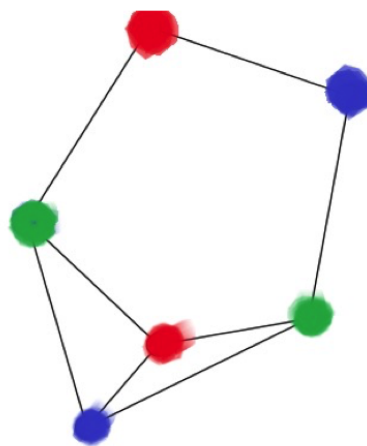
*potrebujemo*, da se naprave med seboj ne bodo motile. Različne radijske frekvence so v tem primeru barve.

(2) Malce bolj nenavaden primer, pri katerem morda sprva ne bi pomislili, da se rešuje z iskanjem kromatičnega števila, pa najdemo pri cestah. Recimo, da imamo križišče na cesti in gledamo *koliko najmanj intervalov za semaforje potrebujemo*, da bodo vsi avtomobili prevozili križišče brez trčenja. Narišemo krivulje, po katerih avtomobili potujejo-te označimo za vozlišča. Povezave v grafu dobimo tako, da smeri povežemo, če se v križišču sekata. Barve pa so faze semaforkega režima. Tiste krivulje, ki ne sekajo nobene poti lahko zanemarimo. Postopek lahko vidimo na slikah (2a, 2b, 2c, 2d)

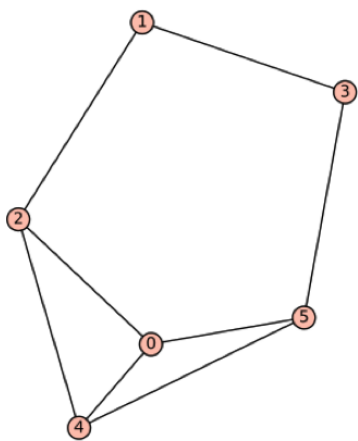
Slika 2: Problem poti v križišču



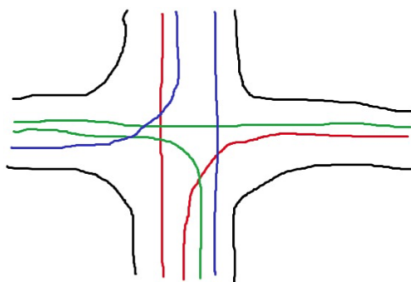
(a) Poti avtomobilov



(b) Graf

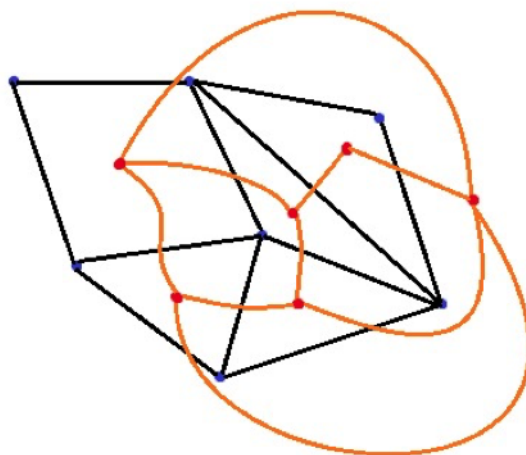


(c) Pobarvan graf



(d) Prevoz križišča





Slika 3: Barvanje zemljevidov

(3) Ko govorimo o barvanju grafov ne moremo spregledati *naloge o barvanju zemljevidov*. S tem primerom so se ukvarjali že leta 1852. Takrat je Francis Guthrie opazil, da se da regionalni zemljevid Anglije obarvati s štirimi barvami tako, da sta katerikoli dve sosednji regiji različno obarvani. Guthrie je imel prav, vendar še ni znal pravilno dokazati, zakaj njegova opažanja držijo. Za lažjo predstavo najprej prevedimo barvanje držav na barvanje točk. Iz grafa s točkami in povezavami naredimo dualni graf:

- Namesto praznih prostorov med povezavami (lic grafa) naredimo točke (zunanost grafa tudi šteje kot eno lice)
- Če sta bili dve lici sosednji (torej, bili ločeni z eno povezavo) potem med tema točkama naredimo povezavo.

Ko smo naredili dualni graf imamo v bistvu problem iskanja kromatičnega števila grafa. Poznamo pa še nekaj:

- Graf je ravninski, če ne vsebuje podgrafa  $K_5$  in  $K_{3,3}$ .

Iz tega lažje sklepamo, da je ravninski graf res največ 4-obarvljiv, vendar dokaza ne bomo navajali. Kasneje sta to dokazala Appel in Haken. Kot zanimivost pa še izrek Götzscg-a:

**Izrek 1** Vsak ravninski graf brez trikotnikov je 3-obarvljiv.

Velja pa še več: poljubno 3-barvanje nekega 4-cikla ali 5-cikla grafa se da razširiti na 3-barvanje celotnega grafa.

## Viri

- Batagelj, V. Barvanje točk grafov. Informatica, časopis za tehnologijo računalništva in probleme informatike, letnik 9, številka 1, 1985. Pridobljeno 19.2.2017
- Zarabec, Barbara. Barvanje grafov. Nauk.si. Pridobljeno 1.12.2016, s <http://www2.nauk.si/materials/>
- Weisstein Eric W. "Vertex Coloring." MathWorld-A Wolfram Web Resource. Pridobljeno 1.12.2016, s <http://mathworld.wolfram.com/VertexColoring.html>