



## Naloga 3

### Barvanje grafov

Barvanje grafov je klasičen problem iz teorije grafov, ki pa ima tudi številne praktične aplikacije. Osnovne ideje z različnimi primeri si lahko preberete na sledečih virih:

- Wikipedia
- Uvod v barvanje grafov

Za praktične aplikacije tega problema pa si lahko ogledate:

- Krajši opis primerov aplikacij
- Uporaba barvanja grafov za izdelavo univerzitetnega urnika
- Bolj poglobljen opis aplikacije barvanja v prevajalnikih

Vaša naloga je implementacija programa, ki bo na standardnem vhodu prejel neusmerjen graf in na njem izvedel enega izmed petih predpisanih algoritmov.

Prvi argument programu podaja algoritem, ki naj se izvede, in sicer:

- 2c - barvanje samo z dvema barvama
- gr - požrešni algoritem (hevrstika)
- ex - izčrpno naštevaneje
- bt - sestopanje
- dp - dinamično programiranje

Ti algoritmi bodo v nadaljevanju bolj podrobno opisani, podan pa bo tudi format sledi, ki ga morate upoštevati pri implementaciji.

**Omejitev dolžine izhoda:** Poleg argumenta, ki podaja vrsto algoritma, naj vaš program podpira še stikalo *-n*. To stikalo podaja število vrstic, ki naj jih program izpiše na standardni izhod. Če to stikalo ni podano, potem naj se izpiše celotna sled, če pa je podano *-n N*, potem naj se izpiše samo **zadnjih** *N* vrstic sledi.

Primer zagona vašega programa:

```
java Naloga3 bt -n 200
```

Zažene naj se torej sestopanje, na standardni izhod pa naj se izpiše zgolj zadnjih 200 vrstic sledi.

### Definicija problema

Podan je enostaven neusmerjen graf  $G = (V, E)$ . Množica barv naj bo kar množica celih števil  $\{0, \dots, k - 1\}$ .

Za tak graf je veljavno obarvanje grafa preslikava  $\phi : V \rightarrow \{0, \dots, k-1\}$ , za katero velja, da ima vsaka povezava različno obarvani krajišči. Formalno,  $\forall (u, v) \in E : \phi(u) \neq \phi(v)$ .

Iščemo najmanjši  $k$ , za katerega obstaja tako veljavno obarvanje. Takemu  $k$  pravimo tudi *kromatsko število grafa*  $\chi(G)$ . Vaša naloga bo, da poleg  $\chi(G)$  najdete tudi barvanje  $\phi$ , ki ustreza temu kromatskemu številu.

### Format vhoda

Na standardnem vhodu boste dobili graf, ki bo podan v sledečem formatu:

1. Najprej bosta podani dve števili  $n, m$ , prvo podaja število vozlišč grafa, drugo pa število povezav.
2. Sledi  $m$  parov števil, vsak par  $(u, v)$  predstavlja eno povezavo v grafu.

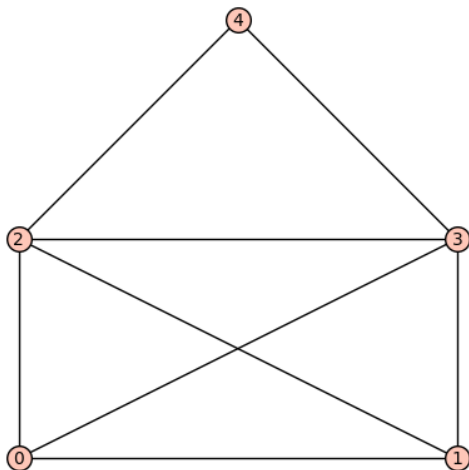
### Predpostavke pri grafih, ki jih bo vaš program obdeloval:

- množica vozlišč bo vedno:  $V = \{0, 1, \dots, n-1\}$ ,
- vsi grafi bodo povezani,
- vsaka povezava bo podana samo enkrat.

### Primer vhoda:

```
5 8
0 1
0 2
0 3
1 2
1 3
2 3
2 4
3 4
```

Ta vhod predstavlja graf (poimenujmo ga *hiška*):



### Algoritmi

Predstavili bomo 5 (zgoraj omenjenih) algoritmov, za vsakega bo predstavljena osnovna ideja ter definiran format izhoda/sledi, ki jo od vas želimo.

Prva dva algoritma imata polinomsko časovno zahtevnost, vendar pa morata za to hitrost nekaj žrtvovati. Prvi algoritem zato rešuje poenostavljeno različico barvanja in zna zgolj ugotoviti ali se da podani graf pobarvati z dvema barvama. Drugi algoritem je hevrističen, kar pomeni, da sicer poskuša vrniti čimboljše rešitev, vendar ne zagotavlja, da je ta rešitev najboljša možna (zelo pogosto tudi ni).

Zadnji trije algoritmi vedno vrnejo najboljše možno rešitev, njihova časovna zahtevnost pa je zaradi tega eksponentna.

## Dvobarvanje grafov

Za vse grafe je enostavno (beri: hitro, polinomsko) ugotoviti ali se jih da obarvati z zgolj dvema barvama. To lahko storimo z enostavnim preiskovanjem grafa v širino.

Pri preiskovanju v širino vedno začnemo z enim vozliščem (izhodiščem), to predstavlja koren preiskovalnega drevesa  $L_0 = \{0\}$ . Iz njega zgradimo množico  $L_1$ , ki predstavlja vsa vozlišča na razdalji 1 od  $L_0$ , nato zgradimo  $L_2$  v kateri so vozlišča na razdalji 2, iz nje  $L_3$ , itd. Iz  $L_i$  je enostavno zgraditi  $L_{i+1}$ , saj enostavno vzamemo vse sosedne vozlišča iz  $L_i$ , razen tistih, ki smo jih pri preiskovanju že srečali.

Za naše potrebe se najprej dogovorimo, da vedno začnemo na vozlišču 0, iz tega vozlišča pa začnemo preiskovati graf v širino po zgornjem postopku. Tak sprehod nam poda tudi poskus obarvanja grafov z dvema barvama. Množice  $L_i$  pri sodih  $i$  naj bodo obarvane z barvo 0, pri lihih  $i$  pa z barvo 1. Pri preiskovanju za vsak  $L_i$  (tako ko je generiran) preverimo, če ga lahko pobarvamo s pripadajočo barvo. Če kakšno vozlišče ustvari neveljavno barvanje, takoj javimo, da se tega grafa ne da obarvati z dvema barvama.

### Format izhoda:

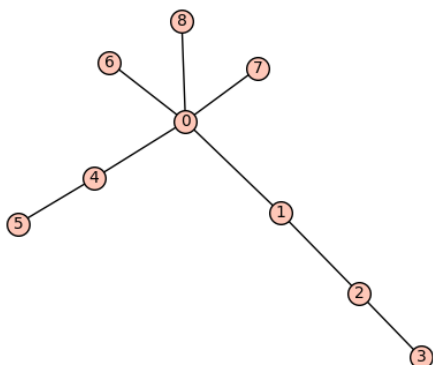
Na izhod tekom preiskovanja izpišete  $i$ , nato pa množico  $L_i$ , pri čemer uredite vozlišča naraščajoče in ločite to množico ter indeks z dvopičjem (natančni znaki razvidni iz primera). Če naletite na množico, ki je ni mogoče obarvati z eno barvo izpišete NOK (not OK) in zaključite s preiskovanjem. Če pa uspete s preiskovanjem obiskati celoten graf, pa v zadnji vrstici izpišete OK.

### Primer izhoda:

Na *hiški* je izhod:

```
0 : 0
1 : 1 2 3
NOK
```

Na spodnjem grafu:



pa je sled:

```
0 : 0
1 : 1 4 6 7 8
2 : 2 5
3 : 3
OK
```

## Požrešno barvanje

Ideja požrešnega barvanja je zelo enostavna, namreč zaporedoma se sprehodimo po celotni množici vozlišč  $V = \{0, \dots, n - 1\}$  in vsakemu vozlišču dodelimo najmanjšo možno barvo, s katero ne kršimo omejitve barvanja.

### Format izhoda:

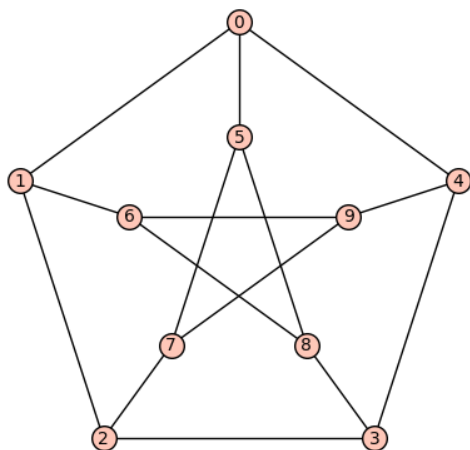
V vsaki vrstici izpišite eno vozlišče in barvo, ki je bila dodeljena temu vozlišču (ločeno z dvopičjem). Vozlišča izpisujete preprosto tako kot ste jih obiskovali - v zaporedju  $0, \dots, n - 1$ .

### Primer izhoda:

Na *hiški* dobimo takšen izhod:

```
0 : 0
1 : 1
2 : 2
3 : 3
4 : 0
```

Na poznanem Petersenovem grafu:



pa je sled:

```

0 : 0
1 : 1
2 : 0
3 : 1
4 : 2
5 : 1
6 : 0
7 : 2
8 : 2
9 : 1

```

### Izčrpno preiskovanje

Prvi algoritem, ki bo zagotovo našel optimalno barvanje grafa, bo temeljil na izčrpnem naštevanju vseh možnih preslikav  $\phi : V \rightarrow \{0, \dots, k-1\}$ . Algoritem pa bo za vsako tako preslikavo preveril, če v grafu predstavlja veljavno barvanje. Da bomo zagotovo našli minimalno obarvanje, bomo začeli s  $k = 2$  (tj. poskusimo obarvati graf z dvema barvama). Za tak  $k$  zgenerirali in preverili vse možne preslikave, takoj ko najdemo veljavno barvanje postopek ustavimo, če pa veljavnega barvanja ne najdemo, povečamo  $k$  za 1 in ponovimo postopek.

Eno preslikavo  $\phi$  lahko predstavimo z vektorjem (tabelo) dolžine  $n$ , ki za vsako vozlišče  $i$  hrani  $\phi(i)$  na poziciji  $i$ .

Pri posameznem  $k$  generirajte preslikave  $\phi$  v leksikografskem vrstnem redu, tj. vektorji, ki predstavljajo preslikavo naj bodo generirani po "abecedi".

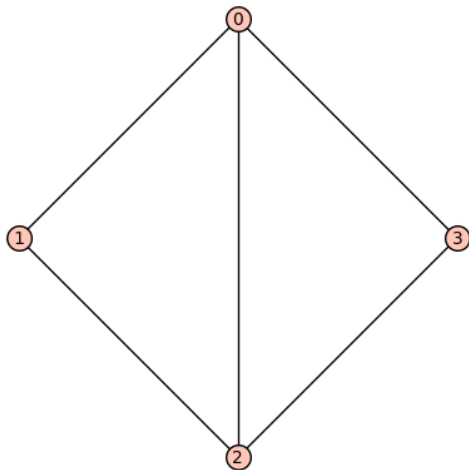
#### Format izhoda:

Najprej izpišete v eni vrstici za kateri  $k$  generirate preslikave.

Nato pa v vsaki vrstici izpišete vektor preslikave in NOK, če ta ne predstavlja veljavnega barvanja, oz. OK, če ta predstavlja veljavno barvanje. Ker se postopek zaključi takoj, ko je najdeno veljavno barvanje, bo samo v zadnji vrstici OK.

#### Primer izhoda:

Za enostaven spodnji graf:



je sled izvajanja:

```
k = 2
0 0 0 0 NOK
0 0 0 1 NOK
0 0 1 0 NOK
0 0 1 1 NOK
0 1 0 0 NOK
0 1 0 1 NOK
0 1 1 0 NOK
0 1 1 1 NOK
1 0 0 0 NOK
1 0 0 1 NOK
1 0 1 0 NOK
1 0 1 1 NOK
1 1 0 0 NOK
1 1 0 1 NOK
1 1 1 0 NOK
1 1 1 1 NOK
k = 3
0 0 0 0 NOK
0 0 0 1 NOK
0 0 0 2 NOK
0 0 1 0 NOK
0 0 1 1 NOK
0 0 1 2 NOK
0 0 2 0 NOK
0 0 2 1 NOK
0 0 2 2 NOK
0 1 0 0 NOK
0 1 0 1 NOK
0 1 0 2 NOK
0 1 1 0 NOK
0 1 1 1 NOK
0 1 1 2 NOK
0 1 2 0 NOK
0 1 2 1 OK
```

Za *hiško* pa lahko sled izvajanja najdete tukaj.

## Sestopanje

Sestopanje je najbolj klasičen pristop za reševanje težkih kombinatoričnih problemov, saj omogoča precejšnjo fleksibilnost, ugodno (tj. relativno majhno) porabo prostora, ter dokaj enostavno implementacijo.

Izhajali bomo iz implementacije izčrpnega preiskovanja, kjer smo preverjali, ali je neko obarvanje veljavno. Pri sestopanju pa ne bomo vedno imeli obarvanja vseh vozlišč, ampak bomo sistematično gradili delna obarvanja, ki jih bomo sproti preverjali za pravilnost.

### Opis algoritma

Tako kot pri izčrpnem preiskovanju, bomo najprej poskusili graf obarvati z dvema barvama, nato s tremi, itn ( $k = 2, 3, \dots$ ). Za vsak  $k$  bomo s sestopanjem preiskali, ali obstaja kakšno veljavno obarvanje. Algoritem se bo sprehajal po preiskovalnem drevesu delnih obarvanj - pri delnem obarvanju so lahko nekatera vozlišča še neobarvana.

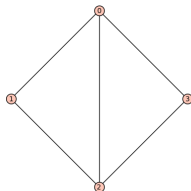
1. Vozlišča obiskujemo v vrstnem redu  $i = 0, \dots, n - 1$ .
2. Vsako vozlišče poskusimo obarvati z eno izmed barv  $c = 0, \dots, k - 1$ .
3. Če je delno obarvanje neveljavno (tj. dve vozlišči z isto barvo sta povezani), poskusimo trenutno vozlišče obarvati z naslednjo barvo.
4. Če pa je delno obarvanje veljavno, se premaknemo na naslednje vozlišče ( $i + 1$ ) in le-tega zopet poskusimo obarvati z vsemi možnimi barvami.
5. Ko smo že preizkusili vse barve, ki so na voljo, in ne najdemo veljavnega obarvanja, sestopimo (tj. vrnemo se na vozlišče  $i - 1$ ).
6. Če pa v vozlišču  $i = n - 1$  najdemo veljavno obarvanje, potem postopek iskanja takoj zaključimo, saj nam je uspelo obarvati celoten graf s  $k$  barvami.

**Optimizacija:** Hitro lahko opazimo, da je pogosto tako preiskovanje na nekaterih mestih redundantno. Npr. pri vozlišču 0, če nam ni uspelo najti nobenega obarvanja ko je  $0 \rightarrow 0$ , potem nam gotovo ne bo uspelo najti nobenega obarvanja, ko  $0 \rightarrow 1$  - barva je zgolj preimenovana. Podobno lahko ugotovimo za vozlišče 1, kjer so veljavni poskusi zgolj  $1 \rightarrow 0$  (vozlišče je enako obarvano kot 0), ter  $1 \rightarrow 1$  (vozlišče je različno obarvano kot 0).

To pravilo lahko uporabimo za poljubno vozlišče  $i$ , veljavni poskusi obarvanja so zgolj  $i \rightarrow \{0, \dots, \min(k - 1, i)\}$ .

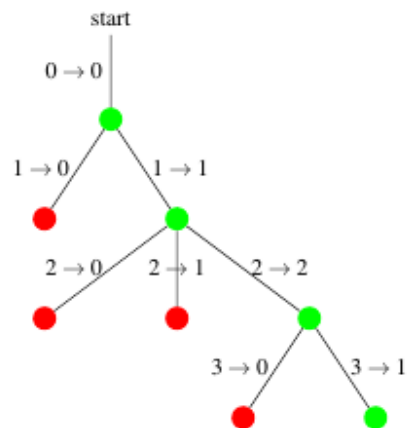
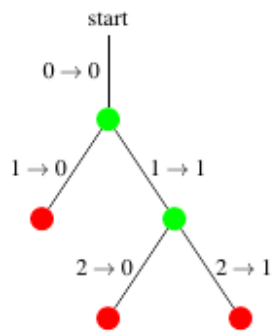
### Primer izvajanja:

Vzemimo enostaven graf (zgoraj je že bil prikazan na večji sliki):



Sled izvajanja zgornjega algoritma je drevo delnih obarvanj. Na tem grafu dobimo najprej drevo za  $k = 2$ , na katerem ne najdemo veljavnega obarvanja, pri  $k = 3$  pa veljavno obarvanje najdemo, zato se postopek takoj zaključi.

Spodnja slika ti dve drevesi prikaže. Levo drevo je za  $k = 2$ , desno drevo pa za  $k = 3$ . V zeleno obarvanih vozliščih imamo veljavno delno obarvanje, v rdečih pa neveljavno.

**Format izhoda:**

Najprej zopet v eni vrstici izpišete katero je največje število barv, ki ga dovolite (torej  $k=...$ ), potem pa izpišete vsako delno dodelitev barv ter zraven OK, če je to veljavno delno obarvanje, ter NOK, če to obarvanje ni veljavno. Zgornji dve drevesi sta izpisani takole:

```

k = 2
0 OK
0 0 NOK
0 1 OK
0 1 0 NOK
0 1 1 NOK
k = 3
0 OK
0 0 NOK
0 1 OK
0 1 0 NOK
0 1 1 NOK
0 1 2 OK
0 1 2 0 NOK
0 1 2 1 OK

```

**Dodatna primera izhoda:**

Na Petersenovem grafu je sled:



```
k = 2
0 OK
0 0 NOK
0 1 OK
0 1 0 OK
0 1 0 0 NOK
0 1 0 1 OK
0 1 0 1 0 NOK
0 1 0 1 1 NOK
0 1 1 NOK
k = 3
0 OK
0 0 NOK
0 1 OK
0 1 0 OK
0 1 0 0 NOK
0 1 0 1 OK
0 1 0 1 0 NOK
0 1 0 1 1 NOK
0 1 0 1 2 OK
0 1 0 1 2 0 NOK
0 1 0 1 2 1 OK
0 1 0 1 2 1 0 OK
0 1 0 1 2 1 0 0 NOK
0 1 0 1 2 1 0 1 NOK
0 1 0 1 2 1 0 2 OK
0 1 0 1 2 1 0 2 0 NOK
0 1 0 1 2 1 0 2 1 NOK
0 1 0 1 2 1 0 2 2 OK
0 1 0 1 2 1 0 2 2 0 NOK
0 1 0 1 2 1 0 2 2 1 OK
```

Na hiški pa je sled:

```

k = 2
0 OK
0 0 NOK
0 1 OK
0 1 0 NOK
0 1 1 NOK
k = 3
0 OK
0 0 NOK
0 1 OK
0 1 0 NOK
0 1 1 NOK
0 1 2 OK
0 1 2 0 NOK
0 1 2 1 NOK
0 1 2 2 NOK
k = 4
0 OK
0 0 NOK
0 1 OK
0 1 0 NOK
0 1 1 NOK
0 1 2 OK
0 1 2 0 NOK
0 1 2 1 NOK
0 1 2 2 NOK
0 1 2 3 OK
0 1 2 3 0 OK

```

## Dinamično programiranje

Dinamično programiranje temelji na pojmu neodvisne množice, zato najprej definirajmo kaj to je.

### Definicija neodvisne množice:

Neodvisna množica v grafu  $G = (V, E)$  je (pod)množica vozlišč  $S \subseteq V$  za katero velja  $\forall i, j \in S : (i, j) \notin E$ . Po domače - med nobenim parom vozlišč v tej množici ni nobene povezave.

Ker vemo, da med vozlišči neodvisne množice ni nobene povezave, jih lahko obarvamo z enako barvo. Oz. obratno, če gledamo pravilno obarvan graf, vsa vozlišča obarvana z enako barvo tvorijo neodvisno množico. Ta opazka nas pripelje do naslednje ideje:

1. Preiščimo vse neodvisne množice  $I \subseteq V$  (ena od teh je gotovo v optimalnem obarvanju obarvana z isto barvo).
2. Odstranimo vsa vozlišča iz  $I$  iz grafa (in pripadajoče povezave), ter rekurzivno poiščimo optimalno obarvanje preostanka grafa.
3. Optimalna rešitev je minimalno število barv preko vseh neodvisnih množic  $I$ .

Oz. bolj kompaktno:

$\chi(G) = 1 + \min(\chi(G[V \setminus I]))$ , kjer je minimum vzet preko vseh neodvisnih množic grafa  $G$ ,  $G[V \setminus I]$  pa predstavlja graf iz katerega je odstranjena množica vozlišč  $I$ .

To definicijo bi že lahko pretvorili v algoritem, ki pa bi bil (zaradi prekrivanja podproblemov) strašljivo neučinkovit. Zato bomo ubrali bolj klasično pot in z dinamičnim programiranjem tabelirali manjše rešitve in iz njih gradili bolj kompleksne rešitve. S tem bomo dobili bolj učinkovit (čeprav še vedno eksponenten) algoritem.

#### Opis algoritma:

1. Naj bo  $S_0, S_1, S_2, \dots, S_{N-1}$  seznam vseh podmnožic množice vozlišč ( $N = 2^n$ ). Te podmnožice naj bodo urejene po velikosti, torej  $i \leq j \iff |S_i| \leq |S_j|$ .
2. Tabela  $T$  naj za vsako podmnožico vozlišč hrani minimalno število barv, ki je potrebno za barvanje zgolj te podmnožice (torej če bi iz grafa izluščili samo ta vozlišča in povezave med njimi). Npr. pri "hiški" je  $T[\{0, 4\}] = 1$ , ker je množica  $\{0, 4\}$  neodvisna.
3. Nastavimo  $T[\{\}] = 0$  - tj. za barvanje prazne množice ne potrebujemo nobene barve.
4. Za vsako podmnožico  $S \in \{S_1, S_2, \dots, S_{N-1}\}$  izračunamo njeno optimalno število barv na sledeči način:
  - $T[S] = 1 + \min(T[S \setminus I])$ , kjer je  $I \subseteq S$  in  $I$  je neodvisna množica.

Ker podmnožice  $S$  obiskujemo naraščajoče po velikosti, so tudi vse vrednosti  $T[S \setminus I]$  že izračunane, ko računate minimum. To nam zagotovi pravilnost algoritma.

#### Format izhoda

Na izhod boste preprosto izpisali tabelo  $T$ , v vsaki vrstici najprej podmnožico  $S_i$ , nato pa (ločeno z dvopičjem) še vrednost  $T[S_i]$ . Podmnožico zapišite v zavitih oklepajih, elemente pa ločene z vejico in urejene naraščajoče.

#### Primer izhoda:

Za hiško dobimo sledeči izhod:

```

{} : 0
{0} : 1
{1} : 1
{2} : 1
{3} : 1
{4} : 1
{0,1} : 2
{0,2} : 2
{1,2} : 2
{0,3} : 2
{1,3} : 2
{2,3} : 2
{0,4} : 1
{1,4} : 1
{2,4} : 2
{3,4} : 2
{0,1,2} : 3
{0,1,3} : 3
{0,2,3} : 3
{1,2,3} : 3
{0,1,4} : 2
{0,2,4} : 2
{1,2,4} : 2
{0,3,4} : 2
{1,3,4} : 2
{2,3,4} : 3
{0,1,2,3} : 4
{0,1,2,4} : 3
{0,1,3,4} : 3
{0,2,3,4} : 3
{1,2,3,4} : 3
{0,1,2,3,4} : 4

```

Za Petersenov graf pa lahko vidite sled tukaj.

## Status oddaje naloge

|                      |   |
|----------------------|---|
| Število oddaj        | To je vaš 1 poskus.                         |
| Status oddaje naloge | Pri tej nalogi vam ni treba oddati ničesar. |
|                      | Pri tej nalogi ne morete oddati prispevkov. |
| Stanje ocen          | Neocenjeno                                  |
| Rok za oddajo        | nedelja, 3. junij 2018, 23:55               |

|                  |                                 |
|------------------|---------------------------------|
| Preostali čas    | Rok za oddajo naloge je potekel |
| Zadnja sprememba | -                               |
| Komentar oddaje  | ► Komentarji (0)                |

## Odziv

|            |                                  |
|------------|----------------------------------|
| Ocena      | 6,00 / 8,00                      |
| Ocenjeno v | ponedeljek, 4. junij 2018, 11:50 |

### NAVIGACIJA



#### Pregledna plošča

##### ■ Prva stran

Strani spletnega mesta


Trenutni predmet


aps2uni

Sodelujoči


Priznanja

Splošno

 O predmetu


 Potek predmeta


 Samostojno delo


 Viri in literatura


 Obvestila

 Razprave

 Opravljanje domačih nalog

 Naloga 0


 Naloga 1

 Naloga 2

 **Naloga 3**

 Skupaj izzivi

 Dodatne točke

 teorija3-rok

5. marec - 11. marec

12. marec - 18. marec

19. marec - 25. marec

26. marec - 1. april

2. april - 8. april

9. april - 15. april

16. april - 22. april

23. april - 29. april

7. maj - 13. maj

14. maj - 20. maj

21. maj - 27. maj

28. maj - 3. junij

Moji predmeti

Predmeti

#### NASTAVITVE



Skrbnišтво predmeta