UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Gregor Kikelj

PREVERJANJE KROMATSKEGA ŠTEVILA GRAFOV Z DOKAZOVALNIM POMOČNIKOM LEAN

Delo diplomskega seminarja

Mentor: prof. dr. Andrej Bauer

Ljubljana, 2024

Kazalo

Preverjanje kromatskega števila grafov z dokazovalnim pomočnikom Lean

Povzetek

TODO

Verification of chromatic number of a graph with Lean proof assistant

Abstract

TODO

Math. Subj. Class. (2020): ..., ...
Ključne besede: ..., ...
Keywords: ..., ...

1 Uvod

Raziskovanje kromatskega števila grafov predstavlja pomemben problem teorije grafov. Kromatsko število grafa nam pove najmnaj koliko barv potrebujemo za barvanje vozlišč grafa, tako da dve sosednji vozlišči nista enake barve. Ta problem ni zanimiv le matematično, ampak ima uporabe tudi pri drugih področjih, predvsem v povezavi z računalništvom. Iskanje kromatskega števila je sicer računsko težek problem v smislu, da ne poznamo algoritma, ki bi ga računal v polinomskem času v odvisnosti od števila vozlišč grafa.

V tej diplomski nalogi se osredotočimo na iskanje kromatskega šteila skupaj z dokazom da smo res našli pravo vrednost. Da to dosežemo si bomo pomagali z dokazovalnikom Lean. V prvem delu naloge natančno definiramo pojme iz teorije grafov ki jih potrebujemo za dokazovanje. Nato na kratko opišemo osnove uporabe dokazovalnika Lean ter matematične definicije konstruiramo znotraj Leana. Na koncu definiramo nekaj algoritmov za iskanje kromatskega števila in dokažemo njihovo pravilnost.

2 Osnovne definicije

Definicija 2.1. Graf je urejen par $G = (V, E)$ kjer je

- $V \neq \emptyset$ končna množica vozlišč,
- E množica povezav, kjer je vsaka povezava množica dveh različnih vozlišč iz V.

Če sta različni vozlišči u, v elementa neke povezave iz E to pišemo kot $u \sim v$ ter pravimo, da sta sosednji vozlišči in da sta krajišči povezave $uv \in E$.

Definicija 2.2. Naj bo $v \in V$. Stopnja vozlišča v je število elementov E ki vsebujejo v, oziroma z besedami število povezav, ki vsebujejo v.

- Največjo stopnjo v grafu označimo z

$$\Delta(G) = \max_{v \in V} deg(v)$$

- Najmanjšo stopnjo v grafu označimo z

$$\delta(G) = \min_{v \in V} deg(v)$$

Definicija 2.3. Definiramo oznako $[k] = \{1, 2, \ldots, k-1\}$ za množico naravnih števil manjših od k.

Definicija 2.4. Naj bo G graf in $k \in \mathbb{N}$. Preslikava $f : V(G) \to [k]$ je k barvanje grafa G če za vse $u, v \in V$ velja $u \sim v \implies f(u) \neq f(v)$.

Z besedami: barvanje grafa s k barvami je preslikava, ki vsakemu vozlišču priredi eno izmed k barv, tako da sta sosednji vozlišči vedno pobarvani z različnima barvama.

Definicija 2.5. Kromatsko število grafa G ki ga označimo kot $\chi(G)$, je najmanjše število, da obstaja $\chi(G)$ barvanje grafa G.

Lema 2.6. *Naj bo G graf. Potem kromatsko število grafa G obstaja.*

Dokaz. Dovolj je da dokažemo da obstaja neko število k za katerega obstaja k barvanje grafa G. Dokažimo da je $k = |V|$ dovolj, kjer je V množica vozlišč grafa G. Ker je V končna množica lahko vozlišča oštevilčimo kot $V = \{v_0, v_1, \ldots, v_{k-1}\}$. Poglejmo si barvanje

$$f : V \to [k], f(v_i) = i$$

Dokazati moramo $\forall v_i, v_j \in V, v_i \sim v_j \implies f(v_i) \neq f(v_j)$. Po definiciji f je dovolj dokazati $\forall v_i, v_j \in V, v_i \sim v_j \implies i \neq j$ to pa je res ker sta si povezani vozlišči vedno različni po definiciji množice povezav grafa. □

Definicija 2.7. Naj bosta G in H grafa. Pravimo da je H podgraf G če je $V(H) \subset V(G)$ in $E(H) \subset E(G)$.

Lema 2.8. *Naj bo G graf in H njegov podgraf. Potem velja $\chi(H) \leq \chi(G)$.*

Dokaz. Očitno. □

3 Lean

Lean je interaktivni dokazovalnik. Uporabljali bomo verzijo Lean 4 ki je razvita pri Microsoftu in temelji na teoriji tipov s čimer se ne bomo ukvarjali preveč ker je tema te diplomske naloge teorija grafov. Lean 4 lahko in ga bomo uporabljali tudi kot funkcijski programski jezik.

3.1 Računanje izrazov

Kot ostali običajni programski jeziki tudi Lean podpira računanje osnovnih aritmetičnih izrazov.

```
#eval 1 + 1
```

Lean nam v informacijskem oknu pove da se izraz evaluira v 2. Lean upošteva tudi vrstni red operacij na primer

```
#eval 1 + 2 * 3
```

se izračuna v 7. Če želimo uporabiti drugačen vrstni red operacij lahko to naredimo z oklepaji

```
#eval (1 + 2) * 3
```

ki se izračuna v 9.

3.2 Tipi

Tipe poznamo iz drugih programskih jezikov (npr. Python, Java) so pa v Leanu bolj pomembni in imajo nekaj več funkcionalnosti.

Osnovni tipi s katerimi se bomo največ ukvarjali so:

- `Nat` naravna števila
- `Int` cela števila
- `String` nizi
- `List` seznami

Lean načeloma tipe izrazov ugotovi sam, lahko pa jih tudi podamo. Na primer

```
#eval 2
```

se izračuna v 2, Lean pa nam pove da je tip izraza `Nat`. Če pa želimo izraz izračunati kot `Int` lahko to naredimo z

```
#eval (2 : Int)
```

kar nam potem predstavlja 2 kot celo število. Lean ponavadi namesto `Nat` uporabi oznako $\mathbb{N}$ in namesto `Int` oznako $\mathbb{Z}$, to pa lahko uporabljamo tudi mi da se program bolj ujema z matematičnim zapisom.

Poglejmo si sedaj kako definiramo funkcije

```
def f (x : ℕ) : ℕ := x + 1
```

Definirali smo funkcijo `f` ki sprejme naravno število in vrne naravno število. Funkcijo lahko poračunamo podobno kot izraze

```
#eval f 2
```

ki se izračuna v 3. Če nismo prepričani kakšen tip ima nek objekt v leanu lahko to ugotovimo z ukazom

```
#check f
```

ki nam vrne $\mathbb{N} \to \mathbb{N}$ torej je funkcija f preslikava iz naravnih števil v naravna števila.

Lahko definiramo tudi funkcije več spremenljivk

```
def sum (x: ℕ) (y: ℕ) : ℕ := x + y
```

ki ima tip $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ torej je preslikava iz naravnih števil v preslikave iz naravnih števil v naravna števila. Ker imata oba argumenta funkcije isti tip lahko definicijo skrajšamo

```
def sum (x y: ℕ) : ℕ := x + y
```

Ker lean zna sam ugotoviti tip vsote dveh naravnih števil tega ni potrebno podati je pa včasih to bolj berljivo

```
def sum (x y: ℕ) := x + y
```

Še krajše pa gre z lambda zapisom

```
λ x y:ℕ => x + y
```

3.2.1 Strukture

Podobno kot v večini programskih jezikov lahko tudi v Leanu definiramo strukture ki so ponavadi namenjene temu da združimo več podatkov v en objekt.

```
structure Tocka :=
    x: Nat
    y: Nat
```

Objekt neke strukture definiramo tako da navedemo vrednosti vseh polj

```
def izhodisce : Tocka := {x := 0, y := 0}
```

Tudi te tipe lahko uporabljamo kot argumente ali rezultate funkcij.

4 Osnovne definicije v Leanu

Pogjemo si sedaj kako smo definirali osnovne pojme iz teorije grafov v Leanu. Za berljivost bomo večino dokazov spuščali. Celotna koda je dostopna na https://github.com/grekiki2/Lean-tests. Vsekakor je v primerjavi s papirjem, boljše dokaze pregledovati znotraj urejevalnika npr. VSCode, ki nam omogoča da v vsakem delu dokaza vidimo trenutne predpostavke in željen rezultat.

```
structure Graph :=
    vertexSize : Nat
    connected: Fin vertexSize → Fin vertexSize → Pro
    connected_decidable: ∀ a b, Decidable (connected a b)
    irreflexive: ∀ n, ¬ connected n n
    symmetric: ∀ a b, connected a b → connected b a
```

Graf torej definiramo kot strukturo kjer podamo število vozlišč, izračunljivo funkcijo ki nam pove ali sta dve vozlišči povezani ter potrdilo da je relacija povezanosti irefleksivna in simetrična.

Za namene testiranja definiramo še graf K_n, ki je graf z n vozlišči in povezavami med vsemi pari vozlišč.

```
def K_n (k:Nat): Graph :=
  {vertexSize:=k, connected:=(λ x y=>x ≠ y), connected_decidable := by
    simp; intro a b;apply Not.decidable, irreflexive := by simp,
    symmetric:= by apply Ne.symm }
```

Na podoben način definiramo tudi C_n oziroma cikličen graf z n vozlišči le da moramo podati še dokaz da je $n \geq 2$.

Definiramo še tip ki predstavlja k barvanje grafa

```
def Coloring (G:Graph) (k : Nat) := Fin G.vertexSize → Fin k
```

Predstavili smo k barvanje grafa kot preslikavo iz (indeksa) vozlišča v eno izmed k barv. Definiramo še kdaj je k barvanje veljavno

```
def valid_coloring (G:Graph) {k:Nat} (coloring: Coloring G k): Prop :=
  ∀ a b, GraphConnected G a b → coloring a ≠ coloring b
```

Sledimo definiciji iz drugega poglavja. Torej z besedami, barvanje je veljavno če sta sosednji vozlišči vedno pobarvani z različnima barvama.

Ker preslikava preverja povezanost za končno število elementov se lahko uporablja za računanje. Drugače rečeno, smiselno je napisati

```
#eval valid_coloring (K_n 5) (λ x => 0)
```

in pričakovati da Lean to izračuna. Lean na žalost izračunljivosti tega izraza ne ugotovi sam, zato moramo pokazati, da je `valid_coloring` izračunjiv oz v Leanu decidable.

```
instance (G: Graph) (k:Nat) (coloring: Coloring G k): Decidable
    (@valid_coloring G k coloring)
    exact Nat.decidableForallFin _
```

Zdaj lahko torej za neko barvanje z Leanom preverimo ali je veljavno ali ne. Ker je množica barvanj grafa G s k barvami končna lahko definiramo tudi izračunljivo funkcijo ki nam za graf G pove ali obstaja k barvanje

```
def is_k_colorable (G : Graph) (k: Nat) : Prop :=
  ∃ (coloring : Coloring G k), valid_coloring G coloring
```

Tudi ta funkcija je izračunljiva ker je množica barvanj grafa G s k barvami končna.

```
instance {G : Graph} (k : Nat) : Decidable (is_k_colorable G k) :=
    Fintype.decidableExistsFintype
```

Sedaj lahko končno definiramo kromatsko število grafa G kot najmanjše število k, da velja `is_k_colorable G k`. V Leanu(oziroma bolj natančno Mathlibu) to naredimo s pomočjo funkcije `Nat.find` ki kot argumente vzame:

- `p : Nat → Prop` Izračunljiv predikat za katerega iščemo najmanjše število da velja `p`
- `h : ∃ n, p n` dokaz da obstaja neko naravno število za katerega velja `p`

Definicija kromatskega števila v Leanu je potem

```
def ChromaticNumber (G: Graph): ℕ :=
  @Nat.find (is_k_colorable G) _ (by {
    unfold is_k_colorable valid_coloring GraphConnected
    use G.vertexSize, λ i=>i
    intro a b hab
    by_contra h
    simp [← h] at hab
    exact (G.irreflexive a) hab
  })
```

Vzamemo torej najmanjše število barv da je graf G k barvljiv, zraven pa dokažemo da vsaj eno tako število obstaja. Najbolj preprosto to naredimo tako da vzamemo število vozlišč grafa G in definiramo barvanje ki vsakemu vozlišču priredi njegov indeks. V poglavju 2 smo se prepričali, da je to res veljavno barvanje.

5 Računanje kromatskega števila

Funkcijo `chromatic_number` lean prepozna kot izračunljivo in jo lahko že uporabljamo za interaktivno računanje kromatskega števila. Na primer

```
#eval ChromaticNumber (K_n 8)
```

nam vrne 8 (po nekaj sekundah). Podobno nam

```
#eval ChromaticNumber (C_n 13 (by linarith))
```

vrne 3 in

```
#eval ChromaticNumber (C_n 14 (by linarith))
```

vrne 2. Za računanje kromatskega števila seveda obstajo hitrejši algoritmi kot je naš, ki to dela na način da preveri vsa možna barvanja pri nekem k dokler ne najde veljavnega, ali pa gre na naslednji k, vendar pa je dokazovanje pravilnosti kompleksnejših algoritmov izredno kompleksno.

Lahko pa si pomagamo s t.i. pričami. Vemo, da če imamo neko barvanje grafa s tremi barvami, je kromatsko število največ 3. Barvanje lahko poišče nek kompleksen algoritem, Lean pa mora potem le preveriti da je veljavno. Da si bomo lahko pomagali s takimi prijemi, pa potrebujemo nekaj pomožnih lem.

Najprej bomo pokazali da če obstaja k barvanje grafa G, je kromatsko število grafa G največ k. Da to storimo pa potrebujemo lemo ki nam pove da iz barvanja grafa G s k barvami lahko tvorimo barvanje grafa G z l barvami, če je le $l \geq k$.

```
def extend_coloring (G:Graph) (k1 k2: Nat) (h:k2≥k1) (coloring: Coloring
    G k1) (h2: valid_coloring G coloring):∃ coloring2:Coloring G k2,
    valid_coloring G coloring2
```

Nato lahko dokažemo da k barvanje grafa G obstaja če in samo če je kromatsko število grafa G največ k.

```
lemma colorable_gives_ub (G: Graph) (k: Nat):
    is_k_colorable G k ↔ chromatic_number G ≤ k
```

Dokaz. Najprej dokažimo implikacijo v desno. Kromatsko število smo definirali kot najmanjši i da je graf G i barvljiv. Če je torej graf G k barvljiv potem je kromatsko število grafa G največ k. V levo, če vemo da je kromatsko število grafa G največ k potem obstaja barvanje grafa G z največ k barvami ki ga lahko razširimo do barvanja s k barvami po lemi `extend_coloring`. □

Posledica je da če in samo če barvanje s k barvami ne obstaja je k manjši od kromatskega števila grafa G.

```
lemma not_colorable_gives_lb (G: Graph) (k: Nat) :
    ¬ is_k_colorable G k ↔ k < chromatic_number G
```

Iz navedenega dobimo glavno posledico. Če najdemo nek k da graf G ni k barvljiv je pa $k+1$ barvljiv potem je kromatsko število grafa G enako $k+1$.

```
theorem bounds_give_chromatic (G:Graph) (k: Nat):
    ¬is_k_colorable G k ∧ is_k_colorable G (k+1) ↔ chromatic_number G =
    k+1
```

Če sedaj na dokazano gledamo iz vidika uporabnosti je precej preprosto Leanu razložiti da je graf G k barvljiv (če je to res, razlaganje napačnih stvari načeloma naj ne bi bilo mogoče) recimo tako da najdemo neko barvanje. Večji problem pa je s spodnjo mejo kromatskega števila. V tem delu se bomo spodnje meje lotili s pomočjo podgrafov.

5.1 Podgrafi

Dokažimo naslednji izrek

```
theorem subgraph_chromatic_number_le (G G2:Graph) (f:Fin G2.vertexSize →
    Fin G.vertexSize) (f_inherits: ∀ a b, G2.connected a b → G.connected
    (f a) (f b)):
    ChromaticNumber G2 ≤ ChromaticNumber G
```

Uporabili smo definicijo podgrafa ki je bolj primerna za Lean kot standardna definicija. Graf $G2$ je podgraf grafa G če obstaja preslikava f ki vozlišča $G2$ preslika v vozlišča G tako, da če sta vozlišči v $G2$ povezani sta povezani tudi preslikani verziji teh vozlišč v G.

Izrek pravi da če je G' podgraf G, potem je kromatsko število grafa G' manjše ali enako kromatskemu številu grafa G.

Dokaz. Po definiciji kromatskega števila je dovolj dokazati, da lahko graf G' pobarvamo z $\chi(G)$ barvami. Naj bo c_G neko $\chi(G)$ barvanje grafa G. Trdimo da je $c_{G'}$ definirano kot $c_{G'}(i) = c_G(f(i))$ veljavno barvanje grafa G'. Barvanje je veljavno če sta sosednji vozlišči vedno pobarvani z različnima barvama. Naj bosta a, b sosednji vozlišči grafa G'. Potem sta po definiciji podgrafa sosednji tudi vozlišči $f(a), f(b)$ v grafu G. Ker je c_G veljavno barvanje grafa G, je $c_G(f(a)) \neq c_G(f(b))$. Ker je $c_{G'}(a) = c_G(f(a))$ in $c_{G'}(b) = c_G(f(b))$ je tudi $c_{G'}(a) \neq c_{G'}(b)$. □

Opomba: pri definiciji podgrafa bi načeloma morali zahtevati injektivnost vendar pa je nikjer v dokazih ne potrebujemo zato tega v Leanu nismo naredili.

Zdaj si lahko pomagamo pri iskanju spodnjih mej kromatskega števila tako da najdemo nek manjši podgraf ki ima čim večje kromatsko število. V praksi se recimo pogosto uporablja K_n klike ki imajo kromatsko število n, ali pa C_n ciklični grafi ki imajo kromatsko število 2 če je n sodo in 3 če je n liho število.

V delu se bomo osredotočili na ciklične grafe.

5.2 Kromatsko število cikličnih grafov

Dokažimo da je kromatsko število cikličnega grafa s sodim številom vozlišč enako 2.

```
theorem chromatic_number_of_C_n_even (n:Nat) (h:2*n≥2):
    chromatic_number (C_n (2*n) h) = 2
```

Dokaz. Potrebno je dokazati da barvanje z 1 barvo ni mogoče in da obstaja barvanje z 2 barvama. Prvi del je očiten, dovolj je pokazati da ima graf vsaj eno povezavo kar lahko naredimo ker ima graf vsaj dve vozlišči.

Za drugi del pa moramo najti barvanje z dvema barvama, edina rešitev (do permutacije barv) je da vozlišče n pobarvamo z $n \bmod 2$. Dokaz da je barvanje pravilno je sicer v Leanu nekoliko dolg, ni pa se o tem težko prepričati, preveriti je potrebno le da sta vozlišči 0 in $2n-1$, ter za vsak $0 \leq i < 2n-1$, i in $i+1$ različne barve. □

Za pokritje vseh cikličnih grafov potrebujemo še dokaz da je kromatsko število cikličnega grafa s lihim številom vozlišč enako 3.

```
theorem chromatic_number_of_C_n_odd (n:Nat) (h:2*n+1≥2):
    chromatic_number (C_n (2*n+1) h) = 3
```

Dokaz. Kot prej je dovolj dokazati da 2 barvanje ne obstaja ter najti 3 barvanje. Za 3 barvanje vzamemo preslikavo ki vozlišče i slika v i mod 2, razen če je $i = 0$ ki ga slika v 2. Podobno kot pri sodem številu vozlišč se je precej enostavno prepričati da je to barvanje res veljavno.

Sedaj moramo dokazati da 2 barvanje ne obstaja. Bšs je vozlišče 0 pobarvano z barvo 0. Z indukcijo po i bomo pokazali da je vozlišče i pobarvano z barvo i mod 2. Za $i = 0$ je to res. Predpostavimo da je za nek i vozlišče i pobarvano z barvo i mod 2. Ker sta vozlišči i in $i + 1$ povezani je vozlišče $i + 1$ pobarvano z barvo i mod $2 + 1 = (i + 1)$ mod 2. S tem smo končali indukcijo. Poglejmo si sedaj vozlišči 0 in $2n$. Po lemi sta obe pobarvani z barvo 0, vendar pa sta povezani kar je protislovje. Torej 2 barvanje ne obstaja. □

6 2 barvljivost

Hitri algoritmi za računanje kromatskega števila v splošnem niso znani, ve se recimo da je problem določiti ali je graf 3 barvljiv NP poln. Za 2 barvljivost pa hiter algoritem obstaja in ga bomo tudi implementirali v Leanu ter dokazali njegovo pravilnost. Naš trenuten algoritem porabi $O(n^2 2^n)$ časa, kjer je n število vozlišč grafa ker mora pregledati vsa možna barvanja.

Implementacija je dostopna na https://github.com/grekiki2/Lean-tests/blob/chromatic-2/Graph/2Colorable.lean, tukaj pa bomo opisali princip delovanja. Za enostavnost predpostavimo da je graf povezan. Algoritem vozlišče 0 pobarva z barvo 0, in začne z iskanjem v širino. Za vsako vozlišče se tudi beleži predhodnika oziroma vozlišče iz katerega smo prišli. Ko obiščemo novo vozlišče ga poskusimo pobarvati drugače od vseh sosedov. Če barvanje vseh vozlišč uspe, potem vrnemo barvanje ki ga Lean lahko preveri in s tem dokaže da je graf 2 barvljiv. Če barvanje ne uspe potem obstaja vozlišče i s sosedoma j in k ki sta različnih barv. To pa pomeni da imata vozlišči j in k tudi različno parnost razdalje do vozlišča 0(vsa vozlišča barve 0 so na sodi razdalji, vsa vozlišča barve 1 pa na lihi razdalji). Iz tega sledi da lahko tvorimo lih cikel kot pot $0 \to j \to i \to k \to 0$. Lih cikel pa je v resnici pograf enak C_{2n+1} za katerega vemo da ima kromatsko število 3 po prejšnjem poglavju, to pa nam pove da graf ni 2 barvljiv.

Lean v obeh primerih(torej da funkcija vrne barvanje ali pa lih cikel) preveri veljavnost priče. Če priča ni veljavna Lean uporabi počasno verzijo algoritma za iskanje barvanja. S tem dobimo hiter in pravilen algoritem za določanje 2 barvljivost grafa, je pa res, da smo uspeli dokazati le pravilnost ne pa tudi hitrosti. V praksi nam ta algoritem omogoča da v pod sekundi določimo 2 barvljivost grafov z do 1000 vozlišči kar je za naše potrebe dovolj.

Slovar strokovnih izrazov

Chromatic number Kromatsko število