Zadatak 9

Grupa 8

Decembar 2024

1 Stabla

1.1 Definicija stabla

Stablo je hijerarhijska struktura podataka koja se koristi u diskretnoj matematici i računarstvu. Sastoji se od čvorova (eng. nodes), pri čemu:

- Jedan čvor je označen kao korenski (root).
- Svaki čvor, osim korenskog, ima tačno jednog roditelja.
- Čvorovi mogu imati nula ili više dece.
- Izmedju čvorova postoji samo jedan put, što znači da stablo ne sadrži cikluse.

Osnovni pojmovi stabla:

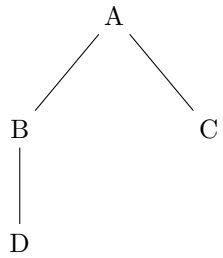
- **Korenski čvor:** Prvi čvor u stablu, od kojeg sve počinje.
- **Listovi:** Čvorovi bez dece.
- **Unutrašnji čvorovi:** Čvorovi koji imaju barem jedno dete.
- **Dubina čvora:** Udaljenost od korenskog čvora do datog čvora.
- **Visina stabla:** Maksimalna dubina bilo kog čvora u stablu.

Primer stabla:

Korenski čvor: `A`

- Deca: `B` i `C`
- Čvor `B` ima dete `D`

Grafički prikaz:



1.2 Definicija šume

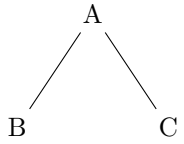
Šuma je skup ili kolekcija nepovezanih stabala. Svako stablo u šumi zadovoljava pravila za stablo, a izmedju stabala u šumi ne postoje veze.

Osnovne karakteristike šume:

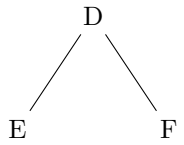
- Sastoji se od jednog ili više disjunktnih stabala.
- Ako uklonimo korenski čvor iz stabla, dobijamo šumu.

Primer šume:

Stablo 1:



Stablo 2:



Cela šuma grafički izgleda kao dva odvojena stabla.

Praktična primena stabala i šuma:

- Stabla se koriste za modelovanje hijerarhija, poput porodičnih stabala, datotečnih sistema i struktura podataka (npr. binarno pretraživačko stablo).
- Šume se pojavljuju u analizi algoritama i modelovanju problema gde je potrebno raditi sa više nepovezanih struktura hijerarhije.

Napomena: U teoriji grafova, stabla su poseban slučaj povezanih acikličnih grafova, dok je šuma skup nepovezanih acikličnih grafova.

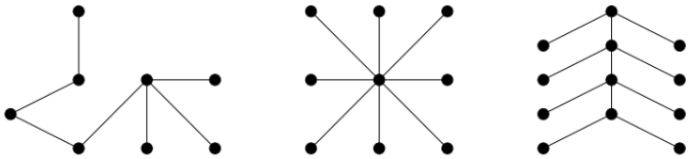
1.3 Još definicija stabla

Za graf koji ne sadrzi nijednu konturu, kazemo da je *aciklican*.

Definicija. Za prost graf $G = (V, E)$ kazemo da je **stablo** ako vazi:
(i) G je povezan graf

(ii) G je aciklican graf

Na sledecoj slici su prikazana tri stabla.



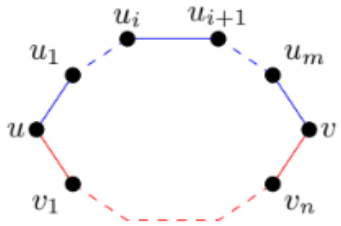
U nastavku cemo dati niz ekvivalentnih tvrdjenja koja karakterisu stablo.

Teorema. Neka je $G = (V, E)$ i $|V| = n \geq 2$. Tada je G stablo ako i samo ako za svaka dva cvora $u, v \in V$ postoji jedinstven uv-put.

Dokaz. Za $n = 2$ tvrdjenje sledi direktno. Pretpostavicemo da je $n \geq 3$.

($\Rightarrow$) Pretpostavimo suprotno, da u stablu G postoje cvorovi u i v sa osobinom da izmedju njih postoje dva razlicita uv-puta. Neka su to putevi U_1 i U_2, sa osobinom da $\{u_i, u_{i+1}\} \in U_1$, $\{u_i, u_{i+1}\} \notin U_2$:

$$U_1 = uu_1 \dots u_i u_{i+1} \dots u_m v$$
$$U_2 = uv_1 \dots v_n v$$



(ako su razliciti putevi, onda postoji grana koja pripada jednom, a ne pripada drugom).

Ovde cemo prikazati slucaj kada je $u, v \notin \{u_i, u_{i+1}\}$, ostali slucajevi se izvode slicno. Sada je:

$$u_i \dots u_1 u v_1 \dots v_n v u_m \dots u_{i+1}$$

$u_i u_{i+1}$ setnja u grafu $G - \{u_i, u_{i+1}\}$. Ako u grafu $G - \{u_i, u_{i+1}\}$ postoji $u_i u_{i+1}$-setnja, onda postoji i $u_i u_{i+1}$-put. Dodavanjem grane $u_i u_{i+1}$ dobijamo konturu u grafu G, sto je u suprotnosti sa pretpostavkom da je G stablo.

($\Leftarrow$) Ako za svaka dva cvora $u, v \in V$ postoji uv-put, onda je G po definiciji povezan graf. Treba jos pokazati da je G aciklican. Pretpostavimo suprotno, da u grafu G postoji kontura oblika:

$$w_1 w_2 w_3 \dots w_l w_1.$$

Tada postoje bar dva puta od w_1 do w_l:

$$w_1 w_l \quad \text{i} \quad w_1 w_2 w_3 \dots w_l,$$

sto je u kontradikciji sa pretpostavkom da za svaka dva cvora postoji jedinstven put od jednog do drugog. To znaci da je nasa pretpostavka netacna i da je G aciklican graf. □

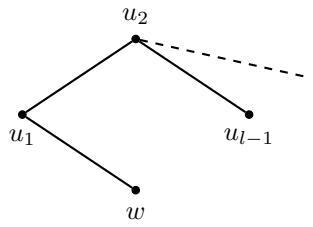
Lema. Neka je $G = (V, E)$ stablo i neka je $|V| = n \geq 2$. Tada postoje bar dva cvora stepena 1.

Dokaz. Kako je G stablo, G je povezan graf. Pretpostavimo da je

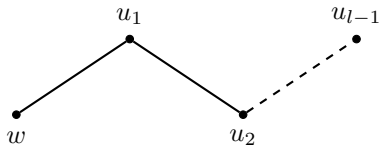
$$u_1 u_2 \dots u_l \tag{3.1}$$

najduzi put u grafu G (moze biti vise takvih puteva iste duzine). Pokazacemo da je tada $d_G(u_1) = d_G(u_l) = 1$. Pretpostavimo da je $d_G(u_1) \geq 2$ (slicno za $d_G(u_l) \geq 2$). Tada postoji cvor $w \neq u_2$ sa osobinom $uw \in E$.

Ako $w \in \{u_3, \dots, u_l\}$, onda G ima konturu, sto je u kontradikciji sa pretpostavkom da je G stablo.



Ako $w \notin \{u_3, \ldots, u_l\}$, onda je put $wu_1u_2 \ldots u_l$ duzi od (3.1), sto dovodi do kontradikcije.



□

Lema. Neka je $G = (V, E)$, $|V| = n \geq 2$, i neka je $d_G(u) = 1$ za neki cvor $u \in V$. Tada je G stablo ako i samo ako je $G - u$ stablo.

Dokaz. ($\Rightarrow$) Pretpostavimo da je G stablo. Da bismo pokazali da je $G - u$ stablo, treba pokazati sledece: (i) $G - u$ je povezan; (ii) $G - u$ je aciklican.

(i) Posmatrajmo dva proizvoljna cvora $v, w \in V(G - u)$. Kako je G povezan, postoji vw-put u G. Taj put ne sadrzi cvor stepena 1 koji je razlicit od v i w, sto znaci da ne sadrzi u. Znaci, taj put je ujedno i put u $G - u$, sto pokazuje da je $G - u$ povezan.

(ii) Kako je G aciklican, to je i $G - u$ aciklican, zato sto brisanjem grane iz aciklicnog grafa ne mozemo dobiti konturu.

($\Leftarrow$) Neka je $G - u$ stablo. Od aciklicnog grafa, dodavanjem nazad lista u ne mozemo dobiti ciklus u tom grafu. Svaki cvor konture ima stepen bar dva, a cvor u je stepena 1. Svaka dva cvora koja su povezana u $G-u$ ostaju povezana i u G. Ostaje jos da pokazemo da za dati par cvorova u, v pri cemu $v \in V(G-u)$, postoji put. Kako je $d_G(u) = 1$ i postoji $v \in V(G)$ sa osobinom $\{u, v\} \in E(G)$, iz pretpostavke da je $G - u$ stablo, sledi da je $G - u$ povezan, odakle za svako $w \in V(G - u)$ postoji wv-put u $G - u$. Dodavanjem grane $\{u, v\}$ tom putu, dobijamo put u G. □

Teorema. Neka je $G = (V, E)$ i $|V| = n \geq 2$. Tada je G stablo ako i samo ako je G povezan graf i $|E| = n - 1$.

Dokaz. ($\Rightarrow$) Prema definiciji stabla, G je povezan graf. Indukcijom po n cemo pokazati da je $|E| = n - 1$.

Baza $n = 2$**:** Stablo sa dva cvora ima tacno jednu granu.

Induktivni korak $T_n \Rightarrow T_{n+1}$**:** Ako je G stablo, onda postoji cvor u sa osobinom $d_G(u) = 1$. Graf $G' = G - u$ ima osobinu

$$|V(G')| = |V(G)| - 1 = n - 1 \quad \text{i} \quad |E(G')| = |E(G)| - 1.$$

Ako je G stablo, onda je prema Lemi 72 G' stablo. Prema induktivnoj pretpostavci je $|E(G')| = n-2$, a odatle je $|E(G)| = |E(G')|+1 = n-1$.

($\Leftarrow$) Indukcijom po n.

Baza $n = 2$**:** Povezan graf sa dva cvora i jednom granom je stablo.

Induktivni korak $T_n \Rightarrow T_{n+1}$**:** Ako je $|E(G)| = |V(G)| - 1$, onda prema posledici postoji cvor u sa osobinom $d_G(u) = 1$. Kako je G povezan, mora vaziti $d_G(u) = 1$ i graf $G' = G - u$ je povezan graf sa osobinom $|V(G')| = |V(G)| - 1 = n$ i $|E(G')| = |E(G)| - 1 = n-2$. Prema induktivnoj pretpostavci je sada $G' - u$ stablo. Prema Lemi 72, G je stablo. □

Teorema. Neka je $G = (V, E)$ i $|V| = n \geq 2$. Tada je G stablo ako je G povezan i brisanjem proizvoljne grane se dobija nepovezan graf.

Dokaz. ($\Rightarrow$) Ako je G stablo, onda je G po definiciji povezan graf. Neka je $\{u, v\} \in E$ proizvoljna grana. Ako pretpostavimo da je $G - \{u, v\}$ povezan, onda postoji uv-put i dodavanjem grane uv bismo dobili konturu u G, sto je u suprotnosti sa pretpostavkom da je G aciklican.

($\Leftarrow$) Ako je G povezan i brisanjem proizvoljne grane se dobija nepovezan graf, onda treba pokazati da je G aciklican. Pretpostavimo da je G povezan i sadrzi konturu C. Tada za svaku granu $uv \in C$ sledi da je $G - \{u, v\}$ povezan, sto je u suprotnosti sa pretpostavkom. □

Teorema. Neka je $G = (V, E)$ i $|V| = n \geq 2$. Tada je G stablo ako je G aciklican i dodavanjem grane se dobija graf koji sadrzi konturu.

Dokaz. ($\Rightarrow$) Ako je G stablo, onda je G aciklican graf po definiciji. Posmatrajmo proizvoljna dva cvora u, v sa osobinom $uv \notin E(G)$. Kako je G povezan, postoji uv-put u G. Dodavanjem grane uv dobijamo konturu u $G + uv$.

($\Leftarrow$) Treba pokazati da je G povezan. Neka su u i v proizvoljni cvorovi iz V. Imamo dva slucaja:

(i) Ako je $uv \in E$, onda je to uv-put.

(ii) Ako $uv \notin E$, onda $G + uv$ sadrzi konturu koja sadrzi uv. Oduzimanjem sa konture grane uv dobijamo uv-put u G.

□

Teorema 78 (Karakterizacija stabla) Neka je $G = (V, E)$ prost graf. Sledeca tvrdjenja su ekvivalentna:

(i) G je stablo.

(ii) Za svaka dva cvora $u, v \in V(G)$ postoji jedinstven put od u do v.

(iii) G je povezan i $|E(G)| = |V(G)| - 1$.

(iv) G je povezan i brisanjem proizvoljne grane dobija se nepovezan graf
(tj. G je minimalan povezan graf).

(v) G je aciklian i dodavanjem grane se dobija graf koji sadrzi konturu
(tj. G je maksimalan aciklian graf).

Dokaz. Dokazali smo sledeci niz ekvivalencija:

$$(i) \Leftrightarrow (ii) \Leftrightarrow (iii) \Leftrightarrow (iv) \Leftrightarrow (v).$$

Odatle mozemo izvesti i sve ostale parove ekvivalencija. □

1.4 Težinski grafovi

Neka je $G = (V, E)$ (neorijentisani) graf čijim su granama pridruženi realni brojevi, takozvane težine grana. Drugim rečima, pored grafa G, data je i *težinska funkcija*

$$w : E \to \mathbb{R}.$$

U raznim primenama, težine grana označavaju njihovu dužinu, cenu, propusnu moć itd. Pri tome je važno istaći da u prvom slučaju dužine ne moraju zadovoljavati aksiome metrike.

Pod gornjim pretpostavkama, uredjen par $M = (G, w)$ predstavlja *težinski graf*, ili kraće *mrežu*. U većini razmatranja težina grafa je jednaka zbiru težina njegovih grana, tj. važi

$$w(G) = \sum_{e \in E} w(e).$$

Na sličan način se težine mogu dodeliti i čvorovima, odnosno i samom samom grafu.

1.5 Pokrivajuća stabla

Kada se razmatraju problemi optimizacije na grafovima, često se dešava da optimalno rešenje ima ne-nula vrednosti samo na nekim podgrafovima koja su stabla i čiji skup čvorova je isti kao u polaznom grafu. Za takav podgraf kažemo da je pokrivajuće (ili razapinjuće ili razapeto) stablo.

Definicija: Graf G_1 je pokrivajuće stablo grafa G ako važe sledeće dve osobine:

- G_1 je pokrivajući podgraf od G: V(G_1) = V(G) i E(G_1) $\subseteq$ E(G)
- G_1 je stablo

Lema: *Neka je $n \geq 3$. Ako je G povezan i $|E(G)| = k \geq n$, onda G ima pokrivajuće stablo.*

Dokaz. Indukcijom po k. Podsetimo se prvo da, prema Teoremi 75, graf sa n čvorova i bar n grana ima konturu.
Baza $k = n$: Oduzimanjem iz grafa jedne grane konture, graf ostaje povezan i pokriva i dalje sve čvorove. Povezan graf sa $n-1$ čvorova je stablo.
Induktivni korak $T_{k-1} \Rightarrow T_k$: Ako G sadrži konturu, onda možemo konstruisati povezan graf G' brisanjem proizvoljne grane konture. Primetimo da je $V(G') = V(G)$ i $E(G') \subseteq E(G)$. Kako je G' povezan, prema induktivnoj pretpostavci, G' ima pokrivajuće stablo, a to je ujedno i pokrivajuće stablo grafa G. □

Teorema: *Graf G ima pokrivajuće stablo ako i samo ako je povezan.*

Dokaz. ($\Rightarrow$) Ako G ima pokrivajuće stablo, onda postoji put izmedju svaka dva čvora stabla, a onda je to put i u grafu G.

($\Leftarrow$) Neka je G povezan. Posmatraćemo dva slučaja.

(1) $|V(G)| = 2$: Povezan graf sa dva čvora ima jednu granu i sopstveno je pokrivajuće stablo.

(2) $|V(G)| = n \geq 3$: Za povezan graf važi da je $|E(G)| \geq n - 1$.

   (a) Ako je $|E(G)| = n - 1$, povezan graf sa $n - 1$ grana je stablo. Znači, G je stablo, a ujedno i sopstveno pokrivajuće stablo.

   (b) Neka je $|E(G)| = k \geq n$. U ovom slučaju tvrdjenje važi na osnovu Leme.

1.6 Definicija Pruferovog koda

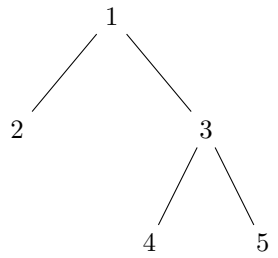
Pruferov kod je način kodiranja označenog stabla pomoću niza od $n-2$ elemenata, gde je n broj čvorova stabla. Dobija se sledećim postupkom:

1. Dokle god stablo ima više od dva čvora:
   - Pronaji list sa najmanjom oznakom.
   - Dodaj oznaku njegovog suseda u Pruferov niz.
   - Ukloni taj čvor iz stabla zajedno sa njegovom granom.
2. Kada ostanu samo dva čvora, proces se završava. Rezultat je niz od $n-2$ elemenata, koji predstavlja Pruferov kod stabla.

Primer: Odredjivanje Pruferovog koda za dato stablo

Razmotri stablo sa $n = 5$ čvorova i sledećim granama:

$$(1, 2), (1, 3), (3, 4), (3, 5)$$



Koraci za odredjivanje Pruferovog koda:

1. Pronaji list sa najmanjom oznakom. To je čvor 2.
   - Njegov sused je čvor 1. Dodaj 1 u Pruferov niz.
   - Ukloni čvor 2 i njegovu granu $(1, 2)$.

   Pruferov niz do sada: $[1]$.

2. Sada je čvor 1 list. Njegov sused je čvor 3.
   - Dodaj 3 u Pruferov niz.
   - Ukloni čvor 1 i njegovu granu $(1, 3)$.

   Pruferov niz do sada: $[1, 3]$.

3. Sledeći list je čvor 4. Njegov sused je 3.
   - Dodaj 3 u Pruferov niz.
   - Ukloni čvor 4 i njegovu granu $(3, 4)$.

   Pruferov niz do sada: $[1, 3, 3]$.

4. Preostaju čvorovi 3 i 5, ali proces se završava, jer preostala dva čvora ne ulaze u Pruferov kod.

Rezultat

Pruferov kod za dato stablo je:

$$[1, 3, 3]$$

Primer: Odredjivanje stabla za dati Pruferov kod

Razmotri Pruferov kod:

$$[4, 4, 4, 5]$$

Broj čvorova n se dobija kao $n = |P| + 2 = 6$.

Koraci za odredjivanje stabla iz Pruferovog koda:

1. Priprema skupa čvorova: {1, 2, 3, 4, 5, 6}
2. Prvi list sa najmanjom oznakom je 1.
   - Povezi 1 sa prvim čvorom iz Pruferovog koda, a to je 4.
   - Ukloni 1 iz skupa i ukloni prvi element iz Pruferovog koda.

   Preostali čvorovi: {2, 3, 4, 5, 6}, Pruferov kod: $[4, 4, 5]$
3. Sledeci list je 2.
   - Povezi 2 sa prvim čvorom iz Pruferovog koda, a to je 4.
   - Ukloni 2 iz skupa i ukloni prvi element iz Pruferovog koda.

   Preostali čvorovi: {3, 4, 5, 6}, Pruferov kod: $[4, 5]$
4. Sledeci list je 3.
   - Povezi 3 sa prvim čvorom iz Pruferovog koda, a to je 4.
   - Ukloni 3 iz skupa i ukloni prvi element iz Pruferovog koda.

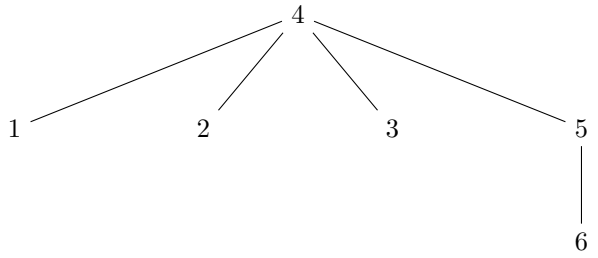
   Preostali čvorovi: {4, 5, 6}, Pruferov kod: $[5]$
5. Sledeci list je 6.
   - Povezi 6 sa prvim čvorom iz Pruferovog koda, a to je 5.
   - Ukloni 6 iz skupa i ukloni prvi element iz Pruferovog koda.

   Preostali čvorovi: {4, 5}
6. Povezi preostale čvorove 4 i 5.

Rezultat

Stablo koje odgovara Pruferovom kodu $[4, 4, 4, 5]$ je definisano granama:

$$(1, 4), (2, 4), (3, 4), (4, 5), (5, 6)$$



1.7 Program za rad sa grafovima

```
from itertools import permutations
from collections import deque

#klasa za cvor grafa
class Cvor:
    def __init__(self,broj) -> None:
        #kao sadrzaj stavljamo samo neki broj
        self._sadrzaj = broj
        self._stepen = 0

    @property
    def sadrzaj(self):
        return self._sadrzaj

    @property
    def stepen(self):
        return self._stepen

    def __str__(self):
        return str(self._sadrzaj)

#klasa za granu grafa
class Grana:
    def __init__(self,pocetak,kraj,tezina=0) ->
        None:
        #inicijalizacija pocetka i kraja grane, to
            su cvorovi
        self._pocetak = pocetak
        self._kraj = kraj
        self._tezina = tezina

    @property
    def pocetak(self):
        return self._pocetak

    @property
    def kraj(self):
        return self._kraj

    @property
    def tezina(self):
        return self._tezina

    #metoda koja vraca krajeve
    def krajevi(self):
        return self._pocetak,self._kraj

    #metoda koja vraca suprotni cvor
```

```
    def suprotan(self, v):
        if not isinstance(v, Cvor):
            print('v nije cvor')
            return
        if self._kraj == v:
            return self._pocetak
        elif self._pocetak == v:
            return self._kraj
        print('v nije cvor grane')

    #metoda koja omogucava da grana bude kljuc mape
    def __hash__(self):
        return hash((self._pocetak,self._kraj))

    def __str__(self):
        return
            str(self._pocetak.sadrzaj)+"-"+str(self._kraj.sadrzaj)+",
            "+str(self._tezina)

#klasa za graf
class Graf:
    def __init__(self):
        self._ulazne_grane = {}
        self._izlazne_grane = {}

    #metoda koja proverava da li je cvor u grafu
    def validacija(self,x):
        if not isinstance(x,Cvor):
            print('Objekat nije cvor')
            return False
        if x not in self._ulazne_grane:
            print('Cvor ne pripada grafu')
            return False
        return True

    #metoda koja proverava da li je cvor u grafu po
        sadrzaju
    def validacija_po_sadrzaju(self,x):
        for cvor in self._ulazne_grane:
            if cvor.sadrzaj==x:
                return True
        return False

    #metoda koja proverava da li je grana u grafu
    def validacija_grane_po_sadrzaju(self,u,v):
        for cvor in self._ulazne_grane:
            if cvor.sadrzaj==u:
                for drugi_cvor in
                    self._ulazne_grane[cvor]:
                    if drugi_cvor.sadrzaj==v:
```

```
                    return True

        for cvor in self._izlazne_grane:
            if cvor.sadrzaj==u:
                for drugi_cvor in
                    self._izlazne_grane[cvor]:
                    if drugi_cvor.sadrzaj==v:
                        return True

        return False

    #metoda koja vraca broj cvorova u grafu
    def broj_cvorova(self):
        return len(self._ulazne_grane)

    #metoda koja vraca broj grana u grafu
    def broj_grana(self):
        broj=0
        for cvor in self._ulazne_grane:
            broj+=len(self._ulazne_grane[cvor])
        return broj

    #metoda koja vraca sve cvorove u grafu
    def cvorovi(self):
        return self._ulazne_grane.keys()

    #metoda koja vraca sve grane u grafu
    def grane(self):
        skup=set()
        for cvor in self._ulazne_grane:
            for grana in
                self._ulazne_grane[cvor].values():
                skup.add(grana)
        return skup

    #metoda koja vraca granu izmedju dva cvora
    def grana(self,u,v):
        validacija=self.validacija(u)
        if not validacija:
            return None
        validacija=self.validacija(v)
        if not validacija:
            return None
        return self._ulazne_grane[v].get(u)

    #metoda koja dodaje cvor u graf
    def dodaj_cvor(self,x):
        cvor=Cvor(x)
        self._ulazne_grane[cvor]={}
        self._izlazne_grane[cvor]={}
```

```
        return cvor

    #metoda koja dodaje granu u graf i poveca rang
        cvora
    def dodaj_granu(self,u,v,tezina=0):
        #provera da li cvorovi postoje
        validacija=self.validacija(u)
        if not validacija:
            return
        validacija=self.validacija(v)
        if not validacija:
            return

        grana=Grana(u,v,tezina)
        self._ulazne_grane[v][u]=grana
        self._izlazne_grane[u][v]=grana
        u._stepen+=1
        v._stepen+=1

    #metoda koja proverava da li je graf prost
    def prost(self):
        for cvor in self._ulazne_grane:
            if len(self._ulazne_grane[cvor])>1:
                return False
        return True

    #metoda koja proverava da li je graf potpuni
    def potpuni(self):
        for cvor in self._ulazne_grane:
            if
                len(self._ulazne_grane[cvor])!=self.broj_cvorova()-1:
                return False
        return True

    #metoda koja proverava da li su dva grafa
        jednaka
    def jednaki(self,graf):
        if self.broj_cvorova()!=graf.broj_cvorova()
            or self.broj_grana()!=graf.broj_grana():
            return False
        for cvor in self._ulazne_grane:
            if cvor not in graf._ulazne_grane:
                return False
            for grana in self._ulazne_grane[cvor]:
                if grana not in
                    graf._ulazne_grane[cvor]:
                    return False
        return True

    #metoda koja proverava da li su dva grafa
```

```
    izomorfna
def izomorfni(self, graf):
    if self.broj_cvorova() !=
        graf.broj_cvorova() or self.broj_grana()
        != graf.broj_grana():
        return False

    stepeni1 = [cvor.stepen for cvor in
        self._ulazne_grane]
    stepeni2 = [cvor.stepen for cvor in
        graf._ulazne_grane]

    stepeni1.sort()
    stepeni2.sort()

    if stepeni1 != stepeni2:
        return False

    cvorovi_self = list(self.cvorovi())
    cvorovi_graf = list(graf.cvorovi())
    for perm in permutations(cvorovi_graf):
        mapa = {cvorovi_self[i]: perm[i] for i
            in range(len(cvorovi_self))}

        is_valid = True
        for cvor in self._ulazne_grane:
            for drugi_cvor in
                self._ulazne_grane[cvor]:
                if (mapa[cvor] not in
                    graf._ulazne_grane or
                    mapa[drugi_cvor] not in
                    graf._ulazne_grane[mapa[cvor]]):
                    is_valid = False
                    break
            if not is_valid:
                break

        if is_valid:
            return True

    return False

#metoda koja proverava da li je graf podgraf
def podgraf(self,graf):
    for cvor in self._ulazne_grane:
        ima_cvor=False
        for cvor_grafa in graf._ulazne_grane:
            if cvor_grafa.sadrzaj==cvor.sadrzaj:
                ima_cvor=True
                break
```

```
            if not ima_cvor:
                return False

            for drugi_cvor in
                self._ulazne_grane[cvor]:
                ima_granu=False

                for drugi_cvor_grafa in
                    graf._ulazne_grane[cvor_grafa]:
                    if
                        drugi_cvor_grafa.sadrzaj==drugi_cvor.sadrzaj:
                        ima_granu=True
                        break

                if not ima_granu:
                    return False
        return True

    #metoda koja proverava da li je graf
        pokrivajuci podgraf
    def pokrivajuci_podgraf(self,graf):
        for cvor in self._ulazne_grane:
            ima_cvor=False
            for cvor_grafa in graf._ulazne_grane:
                if cvor_grafa.sadrzaj==cvor.sadrzaj:
                    ima_cvor=True
                    break
            if not ima_cvor:
                return False

            for drugi_cvor in
                self._ulazne_grane[cvor]:
                ima_granu=False

                for drugi_cvor_grafa in
                    graf._ulazne_grane[cvor_grafa]:
                    if
                        drugi_cvor_grafa.sadrzaj==drugi_cvor.sadrzaj:
                        ima_granu=True
                        break

                if not ima_granu:
                    return False

        for cvor in graf._ulazne_grane:
            ima_cvor=False
            for cvor_grafa in self._ulazne_grane:
                if cvor_grafa.sadrzaj==cvor.sadrzaj:
                    ima_cvor=True
                    break
```

```
            if not ima_cvor:
                return False

        return True

    #metoda koja proverava da li je graf indukovan
        podgraf
    def indukovan_podgraf(self,graf):

        cvorovi=[]
        for cvor in self._ulazne_grane:
            cvorovi.append(cvor.sadrzaj)
            ima_cvor=False
            for cvor_grafa in graf._ulazne_grane:
                if cvor_grafa.sadrzaj==cvor.sadrzaj:
                    ima_cvor=True
                    break
            if not ima_cvor:
                return False

            for drugi_cvor in
                self._ulazne_grane[cvor]:
                ima_granu=False

                for drugi_cvor_grafa in
                    graf._ulazne_grane[cvor_grafa]:
                    if
                        drugi_cvor_grafa.sadrzaj==drugi_cvor.sadrzaj:
                        ima_granu=True
                        break

                if not ima_granu:
                    return False

        for cvor in graf._ulazne_grane:
            if cvor.sadrzaj not in cvorovi:
                continue

            cvor_prvog_grafa=None
            for cvor_grafa in self._ulazne_grane:
                if cvor_grafa.sadrzaj==cvor.sadrzaj:
                    cvor_prvog_grafa=cvor_grafa
                    break

            for drugi_cvor in
                graf._ulazne_grane[cvor]:
                if drugi_cvor.sadrzaj not in
                    cvorovi:
                    continue
```

```
                ima_granu=False

                for drugi_cvor_grafa in
                    self._ulazne_grane[cvor_prvog_grafa]:
                    if
                        drugi_cvor_grafa.sadrzaj==drugi_cvor.sadrzaj:
                        ima_granu=True
                        break

                if not ima_granu:
                    return False

        return True

    #metoda koja proverava da li postoji put
        izmedju dva cvora po sadrzaju
    def put(self,u,v):
        validacija=self.validacija_po_sadrzaju(u)
        if not validacija:
            print("Cvor ",u," ne postoji")
            return False
        validacija=self.validacija_po_sadrzaju(v)
        if not validacija:
            print("Cvor ",v," ne postoji")
            return False

        pocetni_cvor=None
        for cvor in self._izlazne_grane:
            if cvor.sadrzaj==u:
                pocetni_cvor=cvor
                break

        prethodnik = {pocetni_cvor.sadrzaj: None}
        poseceni=[]
        dek=deque([pocetni_cvor])

        ima=False
        while dek:
            cvor=dek.popleft()
            if cvor.sadrzaj==v:
                ima=True
                break
            poseceni.append(cvor)
            for sused in self._izlazne_grane[cvor]:
                if sused not in poseceni:
                    dek.append(sused)
                    prethodnik[sused.sadrzaj]=cvor.sadrzaj

        if ima:
            cvor=v
```

```
            ispis=""
            while cvor is not None:
                ispis+=cvor+"<-"
                cvor=prethodnik[cvor]
            ispis=ispis[:-2]
            print(ispis)
            return True

        return False

    #metoda koja proverava da li postoji kontura za
        neki cvor po sadrzaju
    def kontura(self,u):
        validacija=self.validacija_po_sadrzaju(u)
        if not validacija:
            print("Cvor ",u," ne postoji")
            return False

        pocetni_cvor=None
        for cvor in self._izlazne_grane:
            if cvor.sadrzaj==u:
                pocetni_cvor=cvor
                break

        poseceni=[]
        dek=deque([pocetni_cvor])
        prethodnik = {}

        ima=False
        while dek:
            cvor=dek.popleft()
            poseceni.append(cvor)

            ima_nov=False
            for sused in self._izlazne_grane[cvor]:
                if sused.sadrzaj==u:
                    ima=True
                    poseceni.append(sused)
                    prethodnik[sused.sadrzaj]=cvor.sadrzaj
                    break
                if sused not in poseceni:
                    dek.append(sused)
                    prethodnik[sused.sadrzaj]=cvor.sadrzaj
                    ima_nov=True

            if not ima_nov:
                break

        if ima:
            cvor=u
```

```
            ispis=""
            vratio_se=False
            while not vratio_se:
                ispis+=cvor+"<-"
                cvor=prethodnik[cvor]
                if cvor==u:
                    vratio_se=True
            ispis+=u
            print(ispis)
            return True

        return False

    #metoda koja proverava da li je graf povezan
    def povezan(self):
        for cvor in self._ulazne_grane:
            for drugi_cvor in self._ulazne_grane:
                if cvor.sadrzaj!=drugi_cvor.sadrzaj:
                    ima_put=self.put(cvor.sadrzaj,drugi_cvor.sadrzaj)
                    if not ima_put:
                        return False
        return True

    #metoda koja proverava da li je graf stablo
    def stablo(self):
        if not self.povezan():
            return False
        if
           self.broj_grana()!=2*self.broj_cvorova()-2:
            return False
        return True

    #metoda koja proverava da li je graf suma
    def suma(self):
        stabla=[]
        for cvor in self._ulazne_grane:
            ima_vec=False
            for stablo in stabla:
                if cvor in stablo:
                    ima_vec=True
                    break
            if ima_vec:
                continue

            novo_stablo=[]
            novo_stablo.append(cvor)
            for drugi_cvor in self._ulazne_grane:
                ima_vec=False
                for stablo in stabla:
                    if drugi_cvor in stablo:
```

```
                    ima_vec=True
                    break
            if ima_vec:
                continue

            if cvor.sadrzaj!=drugi_cvor.sadrzaj:
                ima_put=self.put(cvor.sadrzaj,drugi_cvor.sadrzaj)
                if ima_put:
                    novo_stablo.append(drugi_cvor)
        stabla.append(novo_stablo)

    if len(stabla)==1:
        return False

    for stablo in stabla:
        graf=Graf()
        for cvor in stablo:
            graf.dodaj_cvor(cvor.sadrzaj)
        for grana in self.grane():
            pocetak=grana.pocetak
            kraj=grana.kraj

            ima_pocetak=False
            ima_kraj=False
            for cvor in graf._ulazne_grane:
                if
                    cvor.sadrzaj==pocetak.sadrzaj:
                    ima_pocetak=True
                    pocetak=cvor
                if cvor.sadrzaj==kraj.sadrzaj:
                    ima_kraj=True
                    kraj=cvor
                if ima_pocetak and ima_kraj:
                    break

            if not ima_pocetak:
                continue
            if not ima_kraj:
                continue

            graf.dodaj_granu(pocetak,kraj)
        if not graf.stablo():
            return False
    return True

#metoda koja pravi stablo iz Priferovog koda
def stablo_iz_prifera(self,kod):
    graf=Graf()
    str_kod=str(kod)
```

```
        cvorovi=[]
        for i in range(1,len(str_kod)+2):
            cvorovi.append(Cvor(i))
            graf._ulazne_grane[cvorovi[i-1]]={}
            graf._izlazne_grane[cvorovi[i-1]]={}

        for i in range(1,len(str_kod)+1):
            cvor_pocetak=cvorovi[int(str_kod[i-1])-1]
            cvor_kraj=cvorovi[i]
            graf.dodaj_granu(cvor_pocetak,cvor_kraj)

        return graf

    #metoda koja pravi Priferov kod iz stabla
    def priferov_kod(self):
        kod=[]

        susedi={}
        for cvor in self._ulazne_grane:
            susedi[cvor]=[]
            for sused in self._ulazne_grane[cvor]:
                susedi[cvor].append(sused)

        listovi=sorted([cvor for cvor in
            self._ulazne_grane if
            len(self._ulazne_grane[cvor])==1],key=lambda
            x: x.sadrzaj)

        while len(listovi)>0:
            list=listovi.pop(0)

            kod.append(susedi[list][0].sadrzaj)

            sused=susedi[list].pop(0)

            if len(susedi[sused])==1:
                listovi.append(sused)
                listovi.sort(key=lambda x:
                    x.sadrzaj)

        if len(kod)==0:
            kod.append(susedi[listovi[0]][0].sadrzaj)

        return kod

    #metoda koja nalazi minimalno pokrivajuce stablo
    def minimalno_pokrivajuce_stablo(self):
        #Kruskalov algoritam
        # Sortiranje grana prema tezini
        grane = sorted(self.grane(), key=lambda x:
```

```
    x.tezina)

# Kreiranje novog grafa za MST(minimalno
    pokrivajuce stablo)
graf = Graf()
for cvor in self._ulazne_grane:
    graf.dodaj_cvor(cvor.sadrzaj)

# Struktura za pracenje skupova cvorova
    (union-find)
roditelj = {}
rang = {}

def pronadji(cvor):
    if roditelj[cvor] != cvor:
        roditelj[cvor] =
            pronadji(roditelj[cvor])
    return roditelj[cvor]

def unija(cvor1, cvor2):
    cvor1_koren = pronadji(cvor1)
    cvor2_koren = pronadji(cvor2)

    if rang[cvor1_koren] <
       rang[cvor2_koren]:
        roditelj[cvor1_koren] = cvor2_koren
    elif rang[cvor1_koren] >
       rang[cvor2_koren]:
        roditelj[cvor2_koren] = cvor1_koren
    else:
        roditelj[cvor2_koren] = cvor1_koren
        rang[cvor1_koren] += 1

for cvor in self._ulazne_grane:
    roditelj[cvor] = cvor
    rang[cvor] = 0

# Dodavanje grana u MST
for grana in grane:
    pocetak, kraj = grana.krajevi()
    if pronadji(pocetak) != pronadji(kraj):
        pocetak_cvor = None
        kraj_cvor = None
        for cvor in graf._ulazne_grane:
            if cvor.sadrzaj ==
               pocetak.sadrzaj:
                pocetak_cvor = cvor
            if cvor.sadrzaj == kraj.sadrzaj:
                kraj_cvor = cvor
            if pocetak_cvor is not None and
```

```
                    kraj_cvor is not None:
                    break

            graf.dodaj_granu(pocetak_cvor,kraj_cvor,
                grana.tezina)
            unija(pocetak, kraj)

    return graf

# Metoda koja nalazi najkraci put izmedju dva
    cvora
def najkraci_put(self, u, v):
    # Dijkstra algoritam
    # Inicijalizacija
    udaljenost = {}
    prethodnik = {}
    cvorovi = []

    # Inicijalizuj udaljenosti i prethodnike
    for cvor in self._ulazne_grane:
        cvorovi.append(cvor)
        if cvor.sadrzaj == u:
            udaljenost[cvor] = 0
        else:
            udaljenost[cvor] = float('inf')
        prethodnik[cvor] = None

    # Glavna petlja
    while cvorovi:
        # Pronadji cvor sa najmanjom udaljenoscu
        min_udaljenost = float('inf')
        min_cvor = None
        for cvor in cvorovi:
            if udaljenost[cvor] <
                min_udaljenost:
                min_udaljenost =
                    udaljenost[cvor]
                min_cvor = cvor

        if min_cvor is None:
            break

        cvorovi.remove(min_cvor)

        # Azuriraj udaljenosti
        for sused in
            self._ulazne_grane[min_cvor]:
            tezina =
                self._ulazne_grane[min_cvor][sused].tezina
            nova_udaljenost =
```

```
                    udaljenost[min_cvor] + tezina
                if nova_udaljenost <
                    udaljenost[sused]:
                    udaljenost[sused] =
                        nova_udaljenost
                    prethodnik[sused] = min_cvor

        # Rekonstrukcija puta
        put = []
        for cvor in self._ulazne_grane:
            if cvor.sadrzaj == v:
                break

        while prethodnik[cvor] is not None:
            put.append(cvor.sadrzaj)
            cvor = prethodnik[cvor]
        put.append(cvor.sadrzaj)
        put.reverse()

        return put

if __name__ == "__main__":
    grafovi={}

    izabran_graf=None
    ime_grafa=None
    while True:
        if izabran_graf is not None:
            print("Izabran graf: ",ime_grafa)
        else:
            print("Nijedan graf nije trenutno
                izabran")
            prvi_graf=Graf()
            ime=input("Unesite ime prvog grafa:")
            grafovi[ime]=prvi_graf
            izabran_graf=prvi_graf
            ime_grafa=ime
            continue

        print("Izaberite opciju:")
        print("0. Izaberi ili dodaj graf")
        print("1. Dodaj cvor")
        print("2. Dodaj granu")
        print("3. Prikazi broj cvorova")
        print("4. Prikazi broj grana")
        print("5. Prikazi sve cvorove")
        print("6. Prikazi sve grane")
        print("7. Proveri da li postoji grana")
        print("8. Proveri da li postoji cvor")
        print("9. Pronadji direktnog suseda")
```

```
print("10. Stepen cvora")
print("11. Da li je prost graf")
print("12. Da li je potpuni graf")
print("13. Da li su dva grafa jednaka")
print("14. Da li su dva grafa izomorfna")
print("15. Da li je podgraf")
print("16. Da li je pokrivajuci podgraf")
print("17. Da li je indukovan podgraf")
print("18. Da li postoji put izmedju dva
    cvora")
print("19. Da li postoji kontura izmedju
    dva cvora")
print("20. Da li je graf povezan")
print("21. Da li je graf stablo")
print("22. Da li je graf suma")
print("23. Napravi novo stablo iz
    Priferovog koda")
print("24. Napravi Priferov kod iz stabla")
print("25. Nadji minimalno pokrivajuce
    stablo")
print("26. Nadji najkraci put u grafu")
print("27. Kraj")

opcija=int(input())

if opcija==0:
    print("Izaberite opciju:")
    print("1. Izaberi graf")
    print("2. Dodaj graf")
    opcija=int(input())

    if opcija==1:
        print("Unesite koji graf zelite da
            izaberete:")
        for kljuc in grafovi:
            print(kljuc)
        unos=input()

        if unos in grafovi:
            izabran_graf=grafovi[unos]
            ime_grafa=unos
        else:
            print("Ne postoji graf sa tim
                imenom")
    elif opcija==2:
        ime=input("Unesite ime novog
            grafa:")
        grafovi[ime]=Graf()
        izabran_graf=grafovi[ime]
        ime_grafa=ime
```

```
        else:
            print("Nepostojeca opcija")
    elif opcija==1:
        sadrzaj=input("Unesite sadrzaj cvora:")
        cvor=izabran_graf.dodaj_cvor(sadrzaj)
        print("Dodat cvor: ",cvor)
    elif opcija==2:
        prvi=input("Unesite sadrzaj prvog
            cvora:")
        drugi=input("Unesite sadrzaj drugog
            cvora:")
        u=None
        v=None

        validacija=grafovi[ime_grafa].validacija_po_sadrzaju(prvi)
        if validacija:
            for cvor in
                grafovi[ime_grafa].cvorovi():
                if cvor.sadrzaj==prvi:
                    u=cvor
        else:
            continue
        validacija=grafovi[ime_grafa].validacija_po_sadrzaju(drugi)
        if validacija:
            for cvor in
                grafovi[ime_grafa].cvorovi():
                if cvor.sadrzaj==drugi:
                    v=cvor
        else:
            continue

        print("Unesite tezinu grane ili / ako
            necete:")
        tezina=input()
        tezina=tezina.strip()
        tezina=int(tezina)

        if tezina=="/":
            izabran_graf.dodaj_granu(u,v)
            print("Dodata grana izmedju cvora
                ",u," i cvora ",v)
        else:
            izabran_graf.dodaj_granu(u,v,tezina)
            print("Dodata grana izmedju cvora
                ",u," i cvora ",v)
    elif opcija==3:
        print("Broj cvorova u grafu:
            ",izabran_graf.broj_cvorova())
    elif opcija==4:
        print("Broj grana u grafu:
```

```
        ",izabran_graf.broj_grana())
elif opcija==5:
    print("Cvorovi u grafu: ")
    for cvor in izabran_graf.cvorovi():
        print(cvor)
elif opcija==6:
    print("Grane u grafu: ")
    for grana in izabran_graf.grane():
        print(grana)
elif opcija==7:
    print("Unesite sadrzaj prvog cvora:")
    prvi=input()
    print("Unesite sadrzaj drugog cvora:")
    drugi=input()

    u=None
    v=None

    validacija=izabran_graf.validacija_po_sadrzaju(prvi)
    if validacija:
        for cvor in izabran_graf.cvorovi():
            if cvor.sadrzaj==prvi:
                u=cvor.sadrzaj
    else:
        print("Cvor ne postoji")
        continue
    validacija=izabran_graf.validacija_po_sadrzaju(drugi)
    if validacija:
        for cvor in izabran_graf.cvorovi():
            if cvor.sadrzaj==drugi:
                v=cvor.sadrzaj
    else:
        print("Cvor ne postoji")
        continue

    validacija=izabran_graf.validacija_grane_po_sadrzaju(u,v)
    if validacija:
        print("Grana postoji")
    else:
        print("Grana ne postoji")
elif opcija==8:
    print("Unesite sadrzaj cvora:")
    sadrzaj=input()

    validacija=izabran_graf.validacija_po_sadrzaju(sadrzaj)
    if validacija:
        print("Cvor postoji")
    else:
        print("Cvor ne postoji")
elif opcija==9:
```

```
            print("Unesite sadrzaj cvora:")
            sadrzaj=input()
            validacija=izabran_graf.validacija_po_sadrzaju(sadrzaj)
            if validacija:
                cvor_grafa=None
                for cvor in izabran_graf.cvorovi():
                    if cvor.sadrzaj==sadrzaj:
                        cvor_grafa=cvor
                        break

                vrednosti=[]
                for cvor in
                    izabran_graf._ulazne_grane[cvor_grafa]:
                    vrednosti.append(cvor.sadrzaj)

                for cvor in
                    izabran_graf._izlazne_grane[cvor_grafa]:
                    for vrednost in vrednosti:
                        if vrednost==cvor.sadrzaj:
                            break
                    vrednosti.append(cvor.sadrzaj)

                print("Direktni susedi cvora
                    ",cvor_grafa," su: ",vrednosti)
            else:
                continue
        elif opcija==10:
            print("Unesite sadrzaj cvora:")
            sadrzaj=input()
            validacija=izabran_graf.validacija_po_sadrzaju(sadrzaj)
            if validacija:
                cvor=None
                for cvor_grafa in
                    izabran_graf.cvorovi():
                    if cvor_grafa.sadrzaj==sadrzaj:
                        cvor=cvor_grafa
                        break
                print("Stepen cvora ",cvor," je:
                    ",cvor.stepen)
            else:
                continue
        elif opcija==11:
            prost=izabran_graf.prost()
            if prost:
                print("Graf je prost")
            else:
                print("Graf nije prost")
        elif opcija==12:
            potpuni=izabran_graf.potpuni()
            if potpuni:
```

```
            print("Graf je potpuni")
        else:
            print("Graf nije potpuni")
    elif opcija==13:
        print("Unesite ime drugog grafa:")
        ime=input()
        if ime in grafovi:
            graf=grafovi[ime]
            jednaki=izabran_graf.jednaki(graf)
            if jednaki:
                print("Grafovi su jednaki")
            else:
                print("Grafovi nisu jednaki")
        else:
            print("Ne postoji graf sa tim
                imenom")
    elif opcija==14:
        print("Unesite ime drugog grafa:")
        ime=input()
        if ime in grafovi:
            graf=grafovi[ime]
            izomorfni=izabran_graf.izomorfni(graf)
            if izomorfni:
                print("Grafovi su izomorfni")
            else:
                print("Grafovi nisu izomorfni")
        else:
            print("Ne postoji graf sa tim
                imenom")
    elif opcija==15:
        print("Unesite ime drugog grafa:")
        ime=input()

        if ime in grafovi:
            graf=grafovi[ime]
            podgraf=izabran_graf.podgraf(graf)
            if podgraf:
                print("Graf je podgraf")
            else:
                print("Graf nije podgraf")
        else:
            print("Ne postoji graf sa tim
                imenom")
    elif opcija==16:
        print("Unesite ime drugog grafa:")
        ime=input()

        if ime in grafovi:
            graf=grafovi[ime]
            pokrivajuci_podgraf=izabran_graf.pokrivajuci_podgraf(graf)
```

```
            if pokrivajuci_podgraf:
                print("Graf je pokrivajuci
                    podgraf")
            else:
                print("Graf nije pokrivajuci
                    podgraf")
        else:
            print("Ne postoji graf sa tim
                imenom")
    elif opcija==17:
        print("Unesite ime drugog grafa:")
        ime=input()

        if ime in grafovi:
            graf=grafovi[ime]

            indukovan_podgraf=izabran_graf.indukovan_podgraf(graf)
            if indukovan_podgraf:
                print("Graf je indukovan
                    podgraf")
            else:
                print("Graf nije indukovan
                    podgraf")
        else:
            print("Ne postoji graf sa tim
                imenom")
    elif opcija==18:
        print("Unesite sadrzaj prvog cvora:")
        prvi=input()
        print("Unesite sadrzaj drugog cvora:")
        drugi=input()

        setnja=izabran_graf.put(prvi,drugi)
        if setnja:
            print("Postoji put izmedju dva
                cvora")
        else:
            print("Ne postoji put izmedju dva
                cvora")
    elif opcija==19:
        print("Unesite sadrzaj cvora:")
        sadrzaj=input()

        kontura=izabran_graf.kontura(sadrzaj)
        if kontura:
            print("Postoji kontura za cvor")
        else:
            print("Ne postoji kontura za cvor")
    elif opcija==20:
        povezan=izabran_graf.povezan()
```

```
        if povezan:
            print("Graf je povezan")
        else:
            print("Graf nije povezan")
    elif opcija==21:
        stablo=izabran_graf.stablo()
        if stablo:
            print("Graf je stablo")
        else:
            print("Graf nije stablo")
    elif opcija==22:
        suma=izabran_graf.suma()
        if suma:
            print("Graf je suma")
        else:
            print("Graf nije suma")
    elif opcija==23:
        print("Unesite Priferov kod:")
        kod=input()
        kod=eval(kod)
        stablo=izabran_graf.stablo_iz_prifera(kod)
        print("Stablo iz Priferovog koda:")
        for cvor in stablo.cvorovi():
            print(cvor)
        for grana in stablo.grane():
            print(grana)
    elif opcija==24:
        kod=izabran_graf.priferov_kod()
        print("Priferov kod: ",kod)
    elif opcija==25:
        minimalno_pokrivajuce_stablo=izabran_graf.minimalno_pokrivajuce_s
        print("Minimalno pokrivajuce stablo:")
        for cvor in
            minimalno_pokrivajuce_stablo.cvorovi():
            print(cvor)
        for grana in
            minimalno_pokrivajuce_stablo.grane():
            print(grana)
    elif opcija==26:
        print("Unesite sadrzaj prvog cvora:")
        prvi=input()
        print("Unesite sadrzaj drugog cvora:")
        drugi=input()

        put=izabran_graf.najkraci_put(prvi,drugi)

        if put:
            print("Najkraci put: ",put)
        else:
            print("Ne postoji put izmedju dva
```

```
                cvora")
    elif opcija==27:
        print("Kraj")
        break
    else:
        print("Nepostojeca opcija")
```

```
21. Da li je graf stablo
22. Da li je graf suma
23. Napravi novo stablo iz Priferovog koda
24. Napravi Priferov kod iz stabla
25. Nadji minimalno pokrivajuce stablo
26. Nadji najkraci put u grafu
27. Kraj
```

Figure 1: Prikaz dodatnih operacija nad grafovima u odnosu na prethodnu verziju programa.

1.8 Zadaci

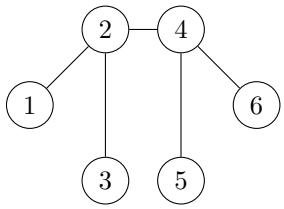
1. Nacrtaj stablo sa čvorovima $\{1, 2, 3, 4, 5, 6\}$ čiji je Pruferov niz jednak $(4, 4, 5, 5)$.

   Rešenje: Početni skup čvorova je $\{1, 2, 3, 4, 5, 6\}$, a dužina Pruferovog niza je 4, što znači da stablo ima 6 čvorova. Rekonstrukcija stabla ide ovako:

   - Nadjemo najmanji čvor koji se ne pojavljuje u Pruferovom nizu: to je 1. Povezujemo ga sa prvim brojem u nizu: 4. Dodajemo granu $\{1, 4\}$
   - Uklanjamo 1 i prvi element iz niza. Preostalo: $(4, 5, 5)$, čvorovi: $\{2, 3, 4, 5, 6\}$
   - Najmanji čvor koji se ne pojavljuje je 2. Povezujemo sa 4 $\Rightarrow \{2, 4\}$
   - Uklanjamo 2 i sledeći element iz niza. Preostalo: $(5, 5)$, čvorovi: $\{3, 4, 5, 6\}$
   - Sada je 3 najmanji nepomenut u kodu: $\{3, 5\}$
   - Uklanjamo 3, sledeći kod: (5), čvorovi: $\{4, 5, 6\}$
   - Sledeći: 6 (nije u kodu), povezuje se sa 5: $\{6, 5\}$
   - Ostaju čvorovi 4 i 5: povežemo ih: $\{4, 5\}$

Ukupne grane: $\{1,4\},\{2,4\},\{3,5\},\{6,5\},\{4,5\}$

2. Odredi Pruferov kod za sledeće stablo:



Rešenje: Pruferov kod se dobija uklanjanjem listova (čvorova stepena 1), redom po najmanjem broju. Svaki put u niz dodajemo čvor koji je bio sused uklonjenom listu:

- 1 i 3 su listovi, uklanjamo najmanji: $1 \Rightarrow 2$
- Sada je 3 najmanji list: $\Rightarrow 2$
- 2 postaje list, uklanjamo: $\Rightarrow 4$
- 5 je novi list: $\Rightarrow 4$
- Ostaje 4 i 6: $\Rightarrow$ nema više elemenata

Pruferov niz: $(2,2,4,4)$

3. Koliko različitih stabala sa 8 čvorova postoji?

Rešenje: Broj različitih označenih stabala sa n čvorova je n^{n-2} (Kejlijeva formula). Za $n = 8$:

$$8^{8-2} = 8^6 = 262144$$

4. Konstruisi stablo čiji je Pruferov niz $(2,2,3,3,4)$ i napiši skup grana.

Rešenje: Čvorova ima $n = 7$, jer niz ima $n - 2 = 5$ elemenata.

Koraci rekonstrukcije:

- Čvorovi: $\{1,2,3,4,5,6,7\}$
- Frekvencije: $2 \to 2$, $3 \to 2$, $4 \to 1$
- Prvi nepomenut: $1 \to \{1,2\}$
- Sledeći nepomenut: $5 \to \{5,2\}$
- Sledeći: $6 \to \{6,3\}$
- Sledeći: $7 \to \{7,3\}$
- Ostalo: $2,4 \to \{2,4\}$, ostaje $3,4 \to \{3,4\}$

Grane: $\{1,2\},\{5,2\},\{6,3\},\{7,3\},\{2,4\},\{3,4\}$

5. Za koje prirodne brojeve n važi da je broj Pruferovih kodova za stabla sa n čvorova deljiv sa n?

   Rešenje: Broj Pruferovih kodova je n^{n-2}. Ova vrednost je deljiva sa n za $n \geq 2$, jer n je u osnovi izraza. Npr:

   $$3^1 = 3 \div 3 = 1, \quad 4^2 = 16 \div 4 = 4, \ldots$$

6. Ako je Pruferov niz stabla jednak $(3,3,3,3,3,3)$, koliko čvorova ima stablo i koliki je stepen čvora 3?

   Rešenje:

   $$\text{Dužina koda} = 6 \Rightarrow n = 8$$

   Čvor se pojavljuje $k-1$ puta u kodu, pa:

   $$\text{Step.} = 6 + 1 = 7$$

7. Konstruisi stablo sa čvorovima $\{1,\ldots,7\}$ gde je čvor 4 list, a čvor 2 ima stepen 4.

   Rešenje: Ako 2 ima stepen 4 $\rightarrow$ pojavljuje se 3 puta u kodu. Ako je 4 list $\rightarrow$ ne sme biti u kodu.

   Primer niza: $(2,2,2,3,5)$ $\rightarrow$ ima 5 elemenata, dakle $n = 7$.

   Rekonstrukcija pokazuje da 4 nije u kodu $\rightarrow$ mora biti list. 2 je 3 puta u kodu $\rightarrow$ stepen 4.

8. Ako je čvor v stepena k, koliko puta se pojavljuje u Pruferovom kodu?

   Rešenje: Čvor se pojavljuje u kodu tačno $k-1$ puta. Svaki put kada čvor ne bude list, već se koristi za povezivanje sa nekim listom, dodaje se u kod. Pošto će biti u $k-1$ takvih situacija (svi osim zadnjeg), formula važi.

9. Ako je stablo definisano Pruferovim nizom $(1,1,2,3,4,5)$, nacrtaj odgovarajuće stablo.

   Rešenje: $n = 8$ jer niz ima 6 elemenata. Čvorovi su $\{1,\ldots,8\}$. Pratimo korake:

   - 6 nije u kodu: $\{6,1\}$
   - 7: $\{7,1\}$

- 8: $\{8, 2\}$
- 2 sada ima stepen 1: $\{2, 3\}$
- $3 \to 4$: $\{3, 4\}$
- $4 \to 5$: $\{4, 5\}$
- Preostali: 1 i 5: $\{1, 5\}$

10. Dokaži da je broj čvorova u stablu uvek dva više od dužine Pruferovog koda.

    Rešenje: Po definiciji, Pruferov kod za stablo sa n čvorova ima $n - 2$ elemenata. Dakle:

    $$n = \text{dužina koda} + 2$$

    što znači da broj čvorova uvek nadmašuje dužinu koda za 2.