Zadatak 10

Grupa 8

Januar 2025

1 Ojlerov graf

1.1 Istorijski osvrt na problem sedam mostova Kenigsberga

Problem sedam mostova Kenigsberga potiče iz 18. veka i vezan je za grad Kenigsberg (današnji Kalinjingrad, Rusija). Grad je bio poznat po svojim mostovima koji su povezivali ostrva i obale reke Pregel. Postojalo je ukupno sedam mostova, i postavljalo se pitanje: da li je moguće proći kroz grad tako da se svaki most predje tačno jednom, a da se na kraju vratite na početnu tačku?

Ovaj problem nije bio samo zabavan izazov, već je imao značaj u razvoju matematike. Rešenje ovog problema dao je švajcarski matematičar Leonard Ojler (Leonhard Euler) 1736. godine, uvodeći temelje teorije grafova. Pokazao je da nije moguće pronaći takvu putanju zbog rasporeda mostova i pravila za tzv. *eulerovski ciklus*.

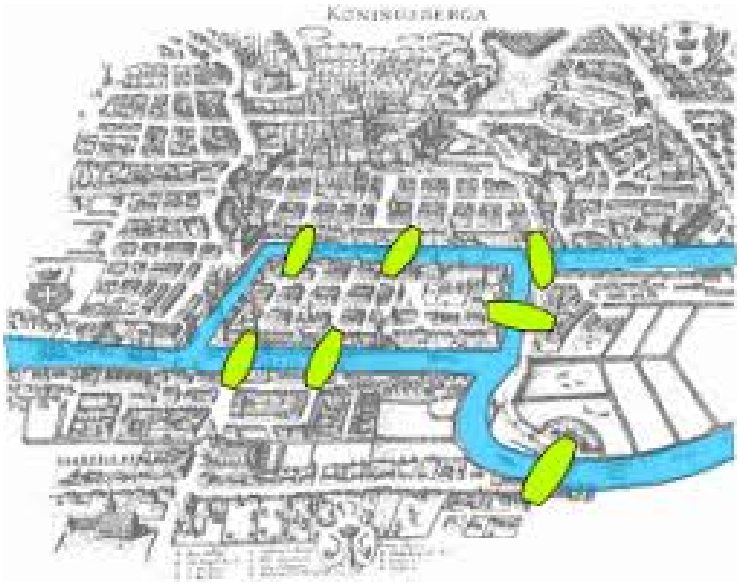


Figure 1: Mapa mostova Kenigsberga iz 18. veka.

Ojlerov rad na ovom problemu predstavljao je prvi formalni korak u razvoju teorije grafova, koja je danas ključna u mnogim oblastima nauke i tehnologije.

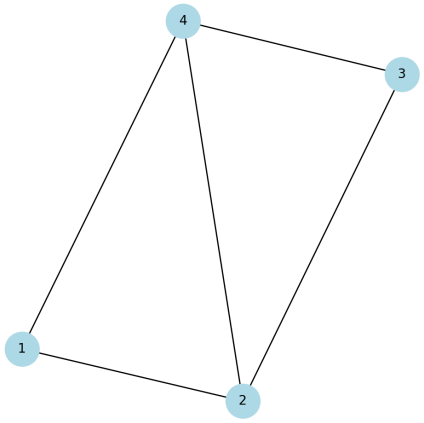
1.2 Ojlerov i Polu-Ojlerov Graf

Ojlerov Graf

Graf G je **Ojlerov graf** ako sadrži **Ojlerov ciklus**. Ojlerov ciklus je ciklus koji prolazi kroz svaku granu grafa tačno jednom i završava se na istom čvoru na kojem je započeo.

Uslovi:

- Graf je povezan (osim možda izolovanih čvorova).
- Svi čvorovi u grafu imaju paran stepen.



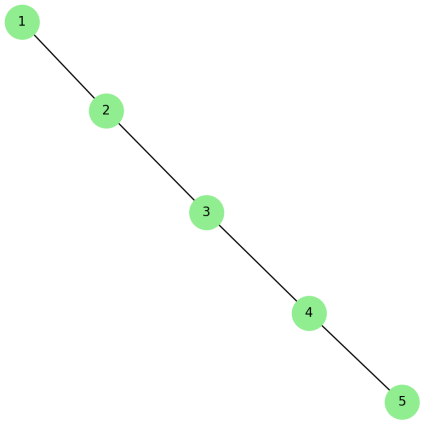
Slika 1: Primer Ojlerovog grafa.

Polu-Ojlerov Graf

Graf G je **polu-Ojlerov graf** ako sadrži **Ojlerovu stazu**, ali nije Ojlerov graf. Ojlerova staza je staza koja prolazi kroz svaku granu grafa tačno jednom, ali nije ciklična (ne završava se na početnom čvoru).

Uslovi:

- Graf je povezan.
- Tačno dva čvora imaju neparan stepen (početni i krajnji čvor staze).



Slika 2: Primer polu-Ojlerovog grafa.

Ojlerov Graf — Definicije i Primeri

Definicija. Neka je G graf bez petlji. *Ojlerov put* (ili *Ojlerova staza*) je setnja u grafu koja sadrzi sve cvorove i grane tog grafa. *Ojlerova tura* je Ojlerova staza u kojoj su pocetni i krajnji cvor jednaki.

Znaci, za setnju u grafu cemo reci da je Ojlerov put ako zadovoljava sledeca dva uslova:

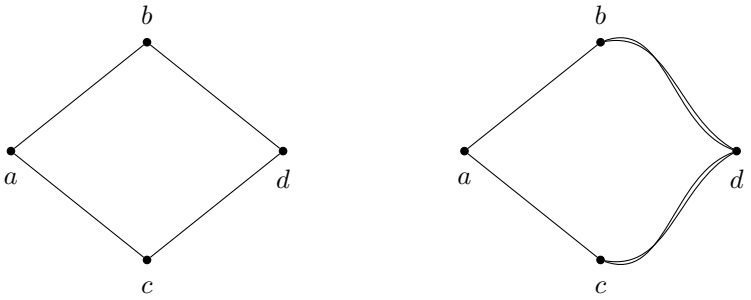
(i) setnja sadrzi sve cvorove grafa;

(ii) ne postoje dve jednake grane u setnji;

(iii) svaka grana grafa se pojavljuje u setnji.

Primer

Posmatrajmo multigrafove G_1 i G_2 predstavljene na slici.



Primer jednog Ojlerovog puta u G_1 je `adcbdac`, dok je primer Ojlerove ture u G_2 odgovarajuca kruzna setnja kroz sve grane.

Definicija. Graf je *Ojlerov* ako sadrzi Ojlerovu turu. Graf je *polu Ojlerov* ako sadrzi Ojlerov put.

Primeri polu Ojlerovog i Ojlerovog grafa prikazani su u prethodnom primeru. U sledecem tvrdjenu dajemo potreban i dovoljan uslov za postojanje Ojlerove ture u grafu.

Teorema. Neka je G graf. Graf G je Ojlerov ako i samo ako je povezan i svaki cvor u G je parnog stepena.

Dokaz. ($\Rightarrow$) Graf je povezan po definiciji Ojlerovog grafa. Neka je

$$u_1e_1u_2e_2\dots u_ne_nu_1$$

Ojlerova tura u G. Posmatrajmo sada proizvoljan cvor $v \in V(G)$. Ako se cvor v pojavljuje l puta u konturi u slucaju kada je $v \neq u_1$, odnosno $l+1$ ako je $v = u_1$, onda je stepen tog cvora $d_G(v) = 2l$.

($\Leftarrow$) Posmatrajmo u G stazu najvece duzine:

$$u_1e_1u_2e_2\dots u_ne_nu_{n+1}$$

Pokazacemo da su prvi i poslednji cvor isti, kao i da se svi cvorovi i grane grafa pojavljuju u toj stazi.

(i) $u_1 = u_{n+1}$: Ako pretpostavimo suprotno, da je $u_1 \neq u_{n+1}$, onda je u toj konturi neparan broj grana incidentan sa u_1 u toj stazi. Kako je stepen cvora u_1 paran, postoji grana $e \in E(G)$ koja nije sadrzana u posmatranoj stazi. U tom slucaju bismo mogli kreirati duzu stazu od posmatrane, sto dovodi do kontradikcije.

(ii) $\{u_1, \dots, u_n\} = V(G)$:

Pretpostavimo suprotno, da postoji cvorovi koji nisu na posmatranoj stazi. Kako je G povezan, postoji grana $\{u_i, v\}$, $i \in \{1, \dots, n\}$, sa osobinom $v \notin \{u_1, \dots, u_n\}$. U tom slucaju mozemo konstruisati stazu vece duzine od posmatrane.

(iii) $\{e_1, \dots, e_n\} = E(G)$:

Sada kada znamo da je posmatrana staza u stvari kontura koja sadrzi sve cvorove grafa, pokazacemo da su sve grane grafa na toj konturi. Ako pretpostavimo suprotno, da postoji grana koja nije na toj konturi, onda bismo ponovo mogli konstruisati duzu stazu od posmatrane.

□

Teorema. Neka je $G = (V, E)$ graf koji nije Ojlerov. Graf G je polu Ojlerov ako i samo ako je G povezan i ima tacno dva cvora neparnog stepena.

Dokaz. ($\Rightarrow$) Slicnim rezonovanjem kao u prethodnom dokazu, za svaki cvor grafa koji nije na krajevima Ojlerovog puta, na tom putu se pojavljuje paran broj grana koje su incidentne sa tim cvorom. Za dva cvora na kraju puta imamo, osim eventualne parnosti broja grana unutar staze, jos po jednu granu koja im je incidentna, dakle dobijamo da ta dva cvora imaju neparne stepene.

($\Leftarrow$) Neka su u i v jedini cvorovi grafa neparnog stepena. Posmatrajmo sada graf G' koji dobijamo od grafa G dodavanjem nove grane koja je incidentna sa cvorovima u i v (ona moze biti paralelna nekim vec postojecim granama). U grafu G' su sada svi cvorovi parnog stepena. Prema Teoremi 90, G' sadrzi Ojlerovu turu. Po definiciji, ta tura sadrzi granu uv. Oduzimanjem grane iz Ojlerove ture grafa G' dobijamo Ojlerov put u grafu G. □

1.3 Program za rad sa grafom

```
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

    # Provera da li graf ima Ojlerovu turu ili stazu
    def ojlerova_tura(self):
        stepeni = {cvor: len(self._ulazne_grane[cvor])
            for cvor in self._ulazne_grane}
        neparni = [cvor for cvor in stepeni if
            stepeni[cvor] % 2 != 0]

        if len(neparni) not in [0, 2]:  # Ako nema
            Ojlerove ture ili staze
            return None

        # Proveri povezanost grafa
        def dfs(cvor, poseceni):
            poseceni.add(cvor)
            for sused in self._ulazne_grane[cvor]:
                if sused not in poseceni:
                    dfs(sused, poseceni)

        svi_cvorovi = list(self._ulazne_grane.keys())
        poseceni = set()
        dfs(svi_cvorovi[0], poseceni)
        if len(poseceni) != len(svi_cvorovi):
            return None

        # Inicijalizacija
        stek = []
        put = []
        trenutni_cvor = neparni[0] if neparni else
            svi_cvorovi[0]  # Pocni sa cvorom neparnog
            stepena ako postoji
```

```
            while True:
                if stepeni[trenutni_cvor] > 0:
                    stek.append(trenutni_cvor)

                    # Pronadji sledeci cvor
                    sledeci =
                        next(iter(self._ulazne_grane[trenutni_cvor]))
                         # Uzmi bilo koji sused

                    # Ukloni granu
                    self._ulazne_grane[trenutni_cvor].pop(sledeci)
                    self._izlazne_grane[sledeci].pop(trenutni_cvor)

                    stepeni[trenutni_cvor] -= 1
                    stepeni[sledeci] -= 1
                    trenutni_cvor = sledeci
                else:
                    put.append(trenutni_cvor.sadrzaj)
                    if stek:
                        trenutni_cvor = stek.pop()
                    else:
                        break

            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. Nadji Ojlerovu turu")
print("28. 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_stabl
        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:
        ojlerova_tura=izabran_graf.ojlerova_tura()
        if ojlerova_tura:
            print("Ojlerova tura: ",ojlerova_tura)
        else:
            print("Ne postoji Ojlerova tura")
    elif opcija==28:
        print("Kraj")
```

```
            break
        else:
            print("Nepostojeca opcija")
```

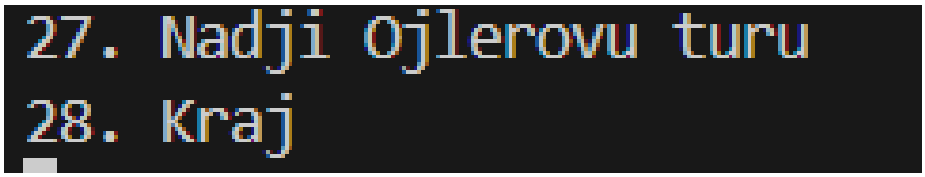


Figure 2: Prikaz dodatnih operacija nad grafovima u odnosu na prethodnu verziju programa.

1.4 Zadaci

1. Neka je dat graf G sa čvorovima $\{a, b, c, d\}$ i granama $\{ab, bc, cd, da\}$. Da li je G Ojlerov graf?

   Rešenje: Graf ima ciklus: $a \to b \to c \to d \to a$. Svi čvorovi imaju stepen 2, što je paran broj.

   Po definiciji, graf je Ojlerov ako je povezan i svi čvorovi imaju paran stepen. Dakle, **jeste Ojlerov graf** i poseduje Ojlerov ciklus.

2. Nacrtaj graf sa 6 čvorova koji nije Ojlerov graf i objasni zašto.

   Rešenje: Razmotrimo graf: $\{a, b, c, d, e, f\}$ sa granama: ab, bc, cd, de, ef.

   Ovaj graf je lanac. Stepeni čvorova:

   $$d(a) = d(f) = 1, \text{ ostali } 2$$

   Imamo dva čvora neparnog stepena. Takav graf ne može imati Ojlerov **ciklus**, ali ima **stazu**.

   Dakle, **nije Ojlerov graf**.

3. Da li graf K_4 (potpuni graf nad 4 čvora) ima Ojlerov ciklus?

   Rešenje: U K_4 svaki čvor je povezan sa ostala 3. Dakle:

   $$d(v) = 3 \text{ za svaki čvor}$$

   Neparan stepen $\to$ graf nije Ojlerov. Potrebno je da su svi stepeni parni.

   Odgovor: K_4 nije Ojlerov.

4. Napiši uslov kada prost graf ima Ojlerov ciklus.

   Rešenje: Graf G ima Ojlerov ciklus ako i samo ako je:

(a) povezan

(b) svi čvorovi imaju paran stepen

5. Konstruisi povezani graf sa 7 čvorova koji ima tačno dva čvora neparnog stepena. Ima li on Ojlerovu stazu? Ciklus?

   Rešenje: Primer: $a \to b \to c \to d \to e \to f \to g$

   Dodamo granu $g \to a$ i $d \to a$

   Stepeni:

   $$d(a) = 3, d(d) = 3, \text{ ostali } 2$$

   Tačno 2 čvora neparnog stepena $\to$ graf **ima Ojlerovu stazu** (ali ne i ciklus).

6. Ako graf ima 10 čvorova i svi čvorovi imaju paran stepen, da li sigurno ima Ojlerov ciklus?

   Rešenje: Ne nužno. Mora da bude i **povezan**. Ako je nepovezan, ne može imati stazu ili ciklus.

   Dakle, odgovor: **ne uvek**.

7. Dokaži: Svaki konektan graf sa tačno dva čvora neparnog stepena ima Ojlerovu stazu.

   Rešenje:

   - Po teoremi: Ako graf ima tačno 2 čvora neparnog stepena, postoji put (staza) koji prolazi kroz svaku granu tačno jednom i počinje i završava se u tim čvorovima.

   Zaključak: takav graf ima Ojlerovu stazu, ali ne i ciklus.

8. Da li graf $K_{3,3}$ ima Ojlerov ciklus?

   Rešenje: $K_{3,3}$ je potpuni bipartitni graf. Svaki čvor ima stepen 3.

   Dakle, svi čvorovi imaju **neparan** stepen $\to$ graf **nije Ojlerov**.

9. Ako graf ima 8 čvorova i 10 grana, može li biti Ojlerov?

   Rešenje: Ukupan broj stepena mora biti paran broj (jer se računa kao $2m$). Ali da bi graf bio Ojlerov, svi čvorovi moraju imati paran stepen.

   Teško je garantovati bez konkretne strukture, ali:

   Ako broj grana nije dovoljan da se svi čvorovi ravnomerno povežu (npr. ako su neki čvorovi stepena 1 ili 3), onda nije.

   Odgovor: Može, ali zavisi od raspodele stepena i povezanosti.

10. Napravi graf koji ima Ojlerovu stazu ali ne i Ojlerov ciklus i napiši tu stazu.

    Rešenje:

    Čvorovi: $\{a, b, c, d\}$

    Grane: ab, bc, cd, db

    Stepeni: $a = 1, b = 3, c = 2, d = 2 \rightarrow a$ i b su neparni

    Ojlerova staza: $a \rightarrow b \rightarrow c \rightarrow d \rightarrow b$

    Graf ima Ojlerovu stazu, ali ne i ciklus.