FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA,ZAVOD ZA PRIMIJENJENU MATEMATIKU

TEORIJA GRAFOVA

Dokumentacija

melita mihaljevic

9/7/2007

Sadržaj

PRIKAZ GRAFA U RAČUNALU

PROGRAMSKO RJEŠENJE

PROGRAMSKO RJEŠENJE

KOTAČ

PROGRAMSKO RJEŠENJE

NAJKRAĆA ZATVORENA STAZA

PROBLEM NAJKRAĆEG PUTA

PROGRAMSKO RJEŠENJE

TRGOVAČKOG PUTNIKA

POHLEPNI HEURISTIČKI ALGORITAM

PROGRAMSKO RJEŠENJE

RAMSKO RJEŠENJE

PROGRAMSKO RJEŠENJE

PROGRAMSKO RJEŠENJE

PROGRAMSKO RJEŠENJE

PROGRAMSKO RJEŠENJE

ŠETANJE PO USMJERENIM GRAFOVIMA

PROGRAMSKO RJEŠENJE

PROGRAMSKO RJEŠENJE

PROGRAMSKO RJEŠENJE

PROBLEM SKAKAČA (KNIGHT'S PROBLEM)

KONSTRUKCIJA SKAKAČEVOG PUTA

PROGRAMSKO RJEŠENJE

TREĆI LABOS

ČETVRTI LABOS

ŠESTI LABOS

KORAK VIŠE

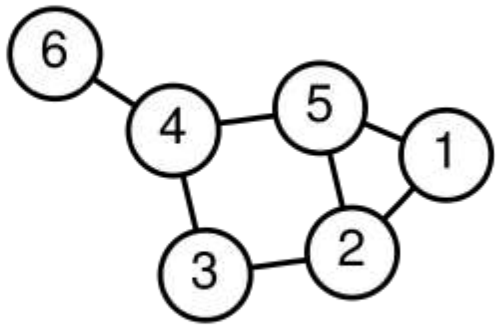
KORAK VIŠE(2)

1. Prikaz grafa u računalu. Lista susjedstva, matrica susjedstva, matrica incidencije. Oduzimanje brida zadanome grafu.
2. Primjeri grafova. Potpuni graf, potpuni bipartitni graf, kotač, kocka, ciklus.
3. Povezanost. U zadanom jednostavnom grafu nađi najkraću i najdulju zatvorenu stazu.
4. Problem najkraćeg puta. Dijkstrin algoritam.
5. Problem trgovačkog putnika. Usporedi pohlepni heuristički algoritam i iscrpnu pretragu za nalaženje najkraćeg hamiltonovskog ciklusa u potpunom težinskom grafu.
6. Stabla. U zadanom grafu prebroji sva razapinjuća stabla. Ispiši barem jedno od njih.
7. Stabla. U zadanom potpunom težinskom grafu nađi razapinjuće stablo minmalne ukupne duljine.
8. Planarnost. Ispitaj je li zadani graf planaran.
9. Bojanje vrhova grafa. Ispitaj je li zadani graf 3-obojiv (v), tj. mogu li se njegovi vrhovi obojati s 3 boje, tako da su susjedni vrhovi raznobojni.
10. Bojanje bridova grafa. Ispita je li zadani graf bridno 3-obojiv, tj. mogu li se njegovi bridovi obojati s 3 boje, tako da su susjedni bridovi raznobojni.
11. Šetnje po usmjerenim grafovima. U zadanom težinskom usmjerenom grafu nađi kritični put.
12. Sparivanja. U zadanom bipartitnom grafu s obilježenim vrhovima nađi i prebroji sva potpuna sparivanja.
13. Korak vise – Clique problem. U zadanom grafu pronaći najveći potpuni graf.
14. Korak vise(2)- Knight's tour problem

Za implementaciju zadanih problema odabran je programski jezik Python. Programski jezik Python podrzava objektno orijentiranu programsku paradigmu, te funkcijsku paradigmu. Zbog svoje prilagođenosti za rad s listama pogodan je za implementaciju prikaza grafova i izvođenje operacija nad grafovima.

PRIKAZ GRAFA U RAČUNALU

Lista susjedstva, matrica susjedstva, matrica incidencije. Oduzimanje brida zadanome grafu.



Za svaki vrh grafa generira se lista susjedstva koja sadrži vrhove susjedne zadanom vrhu. Za gore zadani graf lista susjedstva je sljedeća:

Matrica susjedstva je matrica veličine n x n , gdje je n broj vrhova zadanog grafa. Matrica sadrži vrijednost 1 na mjestu [i][j] ukoliko su vrhovi i susjedni, te 0 na svim ostalim mjestima. Za zadani primjer matrica susjedstva je sljedeća:

Matrica susjedstva simetrična je s obzirom na x=y. Ukoliko je graf težinski, vrši se modifikacija te se umjesto 1 na zadanom mjestu nalazi vrijednost težine zadanog brida kojem pripadaju vrhovi i

Matrica incidencije je matrica veličine n x m, pri čemu je n broj vrhova u zadanom grafu, a m broj bridova u zadanom grafu. Vrijednost polja [i][j] je 1 ukoliko vrh i pripada bridu j te 0 ukoliko vrh i ne pripada bridu j. Za zadani graf matrica incidencije je sljedeća:

Ukoliko je graf težinski vrši se jednaka modifikacija zamjenom 1 težinom zadanog brida.

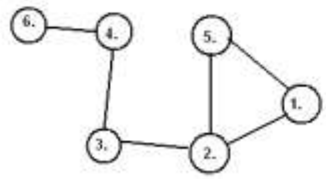
Brisanje brida zadanom grafu G-e vrši se jednostavnim micanjem zadanog brida. Ukoliko je potrebno obrisati brid AB u prezentaciji listom susjedstva: iz liste susjedstva vrha A potrebno je maknuti vrh B, te je iz liste susjedstva vrha B potrebno maknuti vrh A.

Ukoliko je graf prezentiran matricom susjedstva na mjestima [i][j] I [j][i] koje predstavlja brid AB (BA) potrebno je umjesto 1 upisati 0.

Ukoliko je graf prezentiran matricom incidencije potrebno je obrisati stupac j koji predstavlja zadani brid.

Brisanjem brida moguće je da neki od vrhova grafa ostane izdvojen.

Primjer na gore navedenom grafu: brisanje brida ‘4,5’



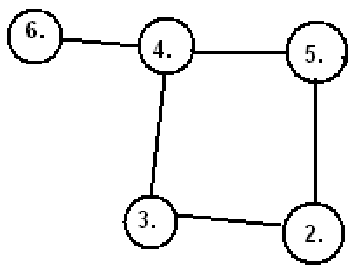
Lista susjedstva:

Martica susjedstva:

Matrica incidencije:

Brisanje vrha u zadanom grafu vrši se na sljedeći način: potrebno je obrisati vrh te sve bridove kojima je taj vrh pripada.

Primjer : u zadanom grafu potrebno je obrisati vrh ‘1’.



Lista susjedstva

Matrica susjedstva:

Matrica incidencije:

PROGRAMSKO RJEŠENJE

Problem zadavanja grafa te brisanja brida i vrha iz zadanog grafa riješen je je klasom `Graph()` koja predstavlja prezentaciju grafa te osnovne operacije nad njim.

Graf je moguće zadati datotekom na 4 različita načina, od kojih su 3 prezentacije grafa listom susjedstva, matricom susjedstva , te matricom incidencije. Moguće je i generirati jedan od primjera grafova(ciklus, potpuni bipartitni graf, kotač, potpuni graf, te kocku) Metoda koja učitava graf:

```
ucitaj_graf(self,tip,ime_datoteke)
```

prima kao parametre način učitavanja grafa (dat, ls, ms, mi, pr), te ime datoteke. Radi lakšeg učitavanja grafova svaka ime datoteke kojom je zadan graf završava njenim tipom npr. Prvi_ls označava da je graf zadan listom susjedstva, prvi_ms – graf zadan matricom susjedstva, prvi_mi – graf zadan matricom incidencije. Zadana metoda poziva funkcije za pretvaranje zadane prezentacije grafa u preostale prezentacije grafa. Npr, ukoliko je zadana prezentacija grafa listom susjedstva, pozivaju se funkcije koje pretvaraju listu susjedstva u matricu susjedstva i matricu incidencije. Zadana metoda vraća pokazivač na strukturu grafa.

Lista susjedstva prestavljena je programskom strukturom dictionary kojima je ključ vrh, a vriijednost je lista vrhova koji su susjedni zadanom vrhu.

Ostale metode u klasi `Graph()` su:

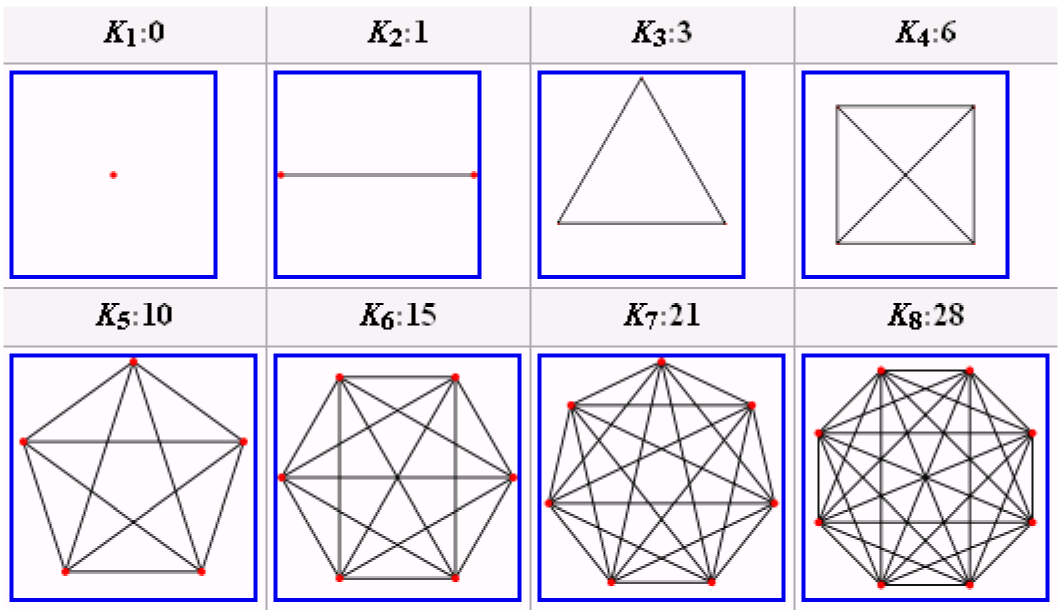
- `dodaj_brid(self,vrh1,vrh2,weight)` – u zadani graf dodaje brid , brid je zadan sa 2 vrha i težinom brida
- `pvrhovi(self)` – vraća listu vrhova zadanih grafova
- `pbridovi(self)` – vraća dictionary bridova
- `get_weight(self,vrh1,vrh2)` vraća težinu zadanog brida
- `pripada_bridu(self)` – vraća vrhove koji pripadaju pojedinom bridu – pomoćna metoda za realizaciju matrice incidencije
- `stupanj(self, vrh)` – vraća stupanj zadanog vrha
- `is_susjed(self,vrh1,vrh2)` – provjerava jesu li vrhovi susjedni (čine li brid)

- `makni_iz_liste(self)` ukoliko su neki vrhovi ili bridovi obrisani u grafu, miče ih iz liste susjedstva
- `pmatrica_susjedstva(self)` – vraća matricu susjedstva
- `pmatrica_incidencije(self`)- vraća matricu incidencije
- `update(self)` – nakon operacije nad grafom osvježava podatke
- `obrisi_vrh(self,vrh)` – iz zadanog grafa briše vrh
- `obrisi_brid(self,brid,u= False)` – iz zadanog grafa briše brid
- `coppy_graph(self)` – vraća kopiju grafa

bipartitni graf, kotač, kocka, ciklus

PROGRAMSKO RJEŠENJE

Primjeri grafova zadani su klasom `Primjeri()` , te je opis i implementacija primjerana opisana u nastavku.



Potpuni graf je graf kojemu su svi vrhovi međusobno povezani.

PROGRAMSKO RJEŠENJE

Generiranju potpunog grafa sa zadanim brojem vrhova implementirano je metodom:

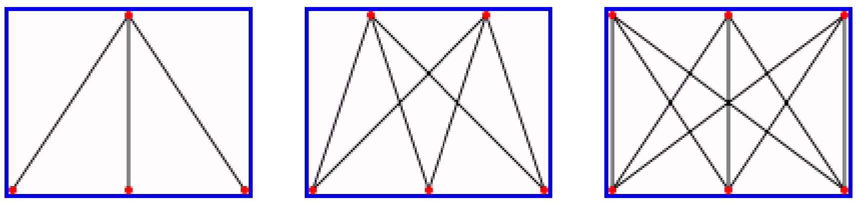
```
get_potpuni(self,broj_vrhova)
```

koja prima kao parametar zadani broj vrhova a vraća pokazivač na izgenerirani potpuni graf.

PRIMJER

Potpuni (5):

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py drugi pr potpuni 5
potpuni 5
1 : ['0', '4', '2', '3']
0 : ['4', '3', '2', '1']
3 : ['0', '4', '1', '2']
2 : ['0', '1', '4', '3']
4 : ['0', '3', '1', '2']
```



$K_{1,3}$ $K_{2,3}$ $K_{3,3}$

Potpuni bipartitni graf je graf kojemu se vrhovi mogu podijeliti u dva disjunktna skupa na način da su svi vrhovi iz jednog skupa susjedi vrhovima iz drugog skupa te da vrhovi iz istog skupa nisu susjedni.

PROGRAMSKO RJEŠENJE

Generiranje potpunog bipartitnog grafa impelementirano je metodom

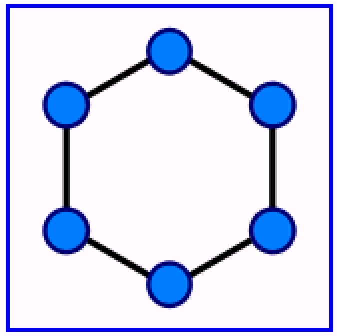
```
get_potpuni_bipartitni(self,r,s)
```

koja kao parametre prima broj vrhova oba skupa (r,s)

PRIMJER

Potpuni bipartini (3,3):

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py drugi pr bipartitni 3 3
1 : ['4', '5', '3']
0 : ['5', '4', '3']
3 : ['0', '1', '2']
2 : ['5', '4', '3']
5 : ['0', '1', '2']
4 : ['0', '1', '2']
potpuni bipartitni 3 3
```



Ciklus je povezani graf kojemu su svi vrhovi stupnja dva.

PROGRAMSKO RJEŠENJE

Generiranje ciklusa implementirano je metodom

```
get_ciklus(self, n)
```

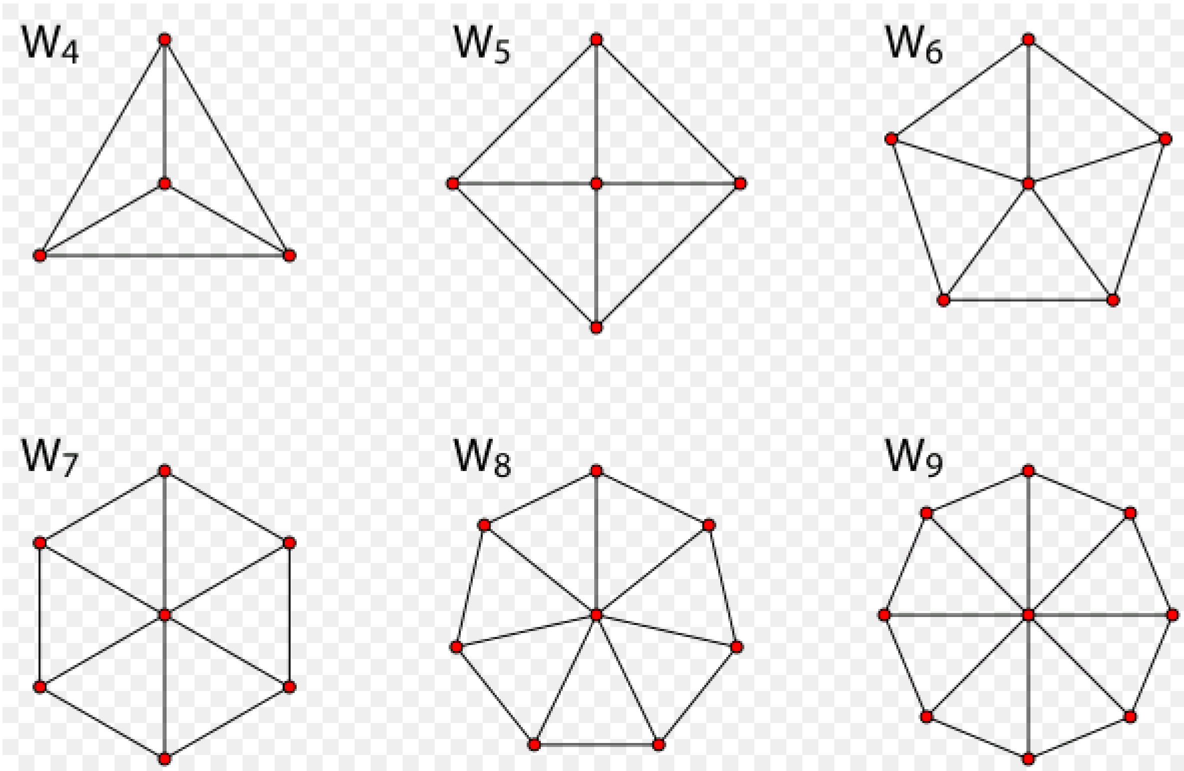
koji kao parametar prima broj vrhova, a vraća pokazivač na generirani graf

PRIMJER

Ciklus(6)

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py drugi pr ciklus 6
ciklus 6
1 : ['0', '2']
0 : ['1', '5']
3 : ['4', '2']
2 : ['1', '3']
5 : ['4', '0']
4 : ['5', '3']
```

KOTAČ



Kotač je graf koji se sastoji od ciklusa (n-1)te središnjeg vrha koji je povezan sa svim vrhovima ciklusa.

PROGRAMSKO RJEŠENJE

Generiranje kotača implementirano je metodom

```
get_kotac(self,n)
```

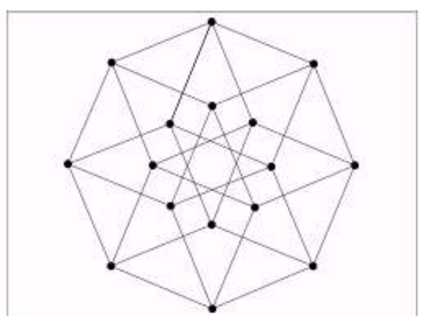
koja kao parametar prima broj vrhova, u sebi poziva generiranje ciklusa duljine n-1 te povezivanjem n-tog vrha vrhovima ciklusa.

PRIMJER

Kotac(5)

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py drugi pr kotac 5
ciklus 4
1 : ['2', '0']
0 : ['3', '1']
3 : ['0', '2']
2 : ['1', '3']
kotac 5
1 : ['2', '0', '4']
0 : ['3', '1', '4']
3 : ['0', '2', '4']
2 : ['1', '3', '4']
4 : ['0', '3', '1', '2']
```

U ovom primjeru se vidi da se najprije generira ciklus, a zatim kotac



Kocka je graf u kojem su 2 vrha susjedi ukoliko se njihove numeracije u binarnoj reprezentaciji razlikuju točno za 1.

PROGRAMSKO RJEŠENJE

Generiranje kocke dimenzije n implementirano je metodom :

```
get_kocka(self,n)
```

pri čemu je n dimenzija. Broj vrhova kocke je 2^n. Zadana metoda sadrži dvije funkcije:

```
bin(br) dec(br)
```

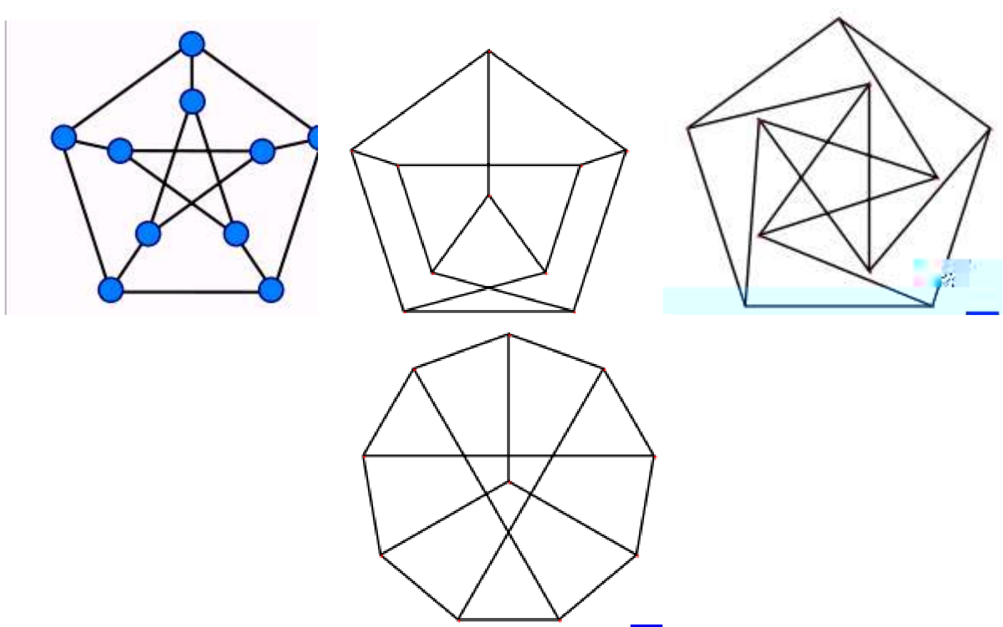
koje pretvaraju zadani dekadski broj u binarni , te obrnuto.

PRIMJER

Kocka(4)

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py drugi pr kocka 4
kocka 4
11 : ['3', '9', '10', '15']
10 : ['2', '8', '11', '14']
13 : ['5', '9', '12', '15']
12 : ['4', '8', '13', '14']
15 : ['7', '11', '13', '14']
14 : ['6', '10', '12', '15']
1 : ['0', '3', '5', '9']
0 : ['1', '2', '4', '8']
3 : ['1', '2', '7', '11']
2 : ['0', '3', '6', '10']
5 : ['1', '4', '7', '13']
4 : ['0', '5', '6', '12']
7 : ['3', '5', '6', '15']
6 : ['2', '4', '7', '14']
9 : ['1', '8', '11', '13']
```

Petersenov graf je 3-regularan graf . Neke od prezentacija grafa u ravnini prikazana je u nastavku:



Petersenov graf je specifičan graf, 3-regularan graf, nije planaran, vršno je 3-obojiv, te bridno 4-obojiv.

PROGRAMSKO RJEŠENJE

Generiranje Petersenovog grafa implementirano je metodom:

```
get_petersen(self)
```

koji vraća pokazivač na zadani petersenov graf.

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py drugi pr petersen
petersen
1 : ['0', '6', '2']
0 : ['5', '4', '1']
3 : ['8', '4', '2']
2 : ['1', '7', '3']
5 : ['0', '7', '8']
4 : ['0', '9', '3']
7 : ['9', '5', '2']
6 : ['8', '1', '9']
9 : ['7', '4', '6']
8 : ['3', '6', '5']
```

U zadanom jednostavnom grafu nađi najkraću i najdulju zatvorenu stazu.

PROGRAMSKO RJEŠENJE

Zadatak je riješen implementacijom klase `Povezanost()` koja sadrži niz metoda za traženje najkraće i najdulje zatvorene staze, te metode

```
izvrsi(self,g)
```

koja poziva ostale glavne metode.

NAJKRAĆA ZATVORENA STAZA

Najkraća zatvorena staza pronalazi se algoritmom koji prelazi sve bridove te se pokušava vratiti do početnog vrha što kraćim putem. Ukoliko je pronađena najkraća staza duljine 3, algoritam se zaustavlja.

PROGRAMSKO RJEŠENJE

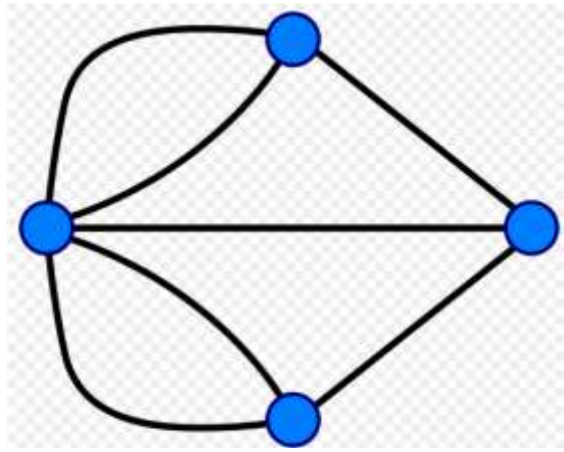
Traženje najkraće zatvorene staze implementirano je metodom

```
najkraca_staza(self,g)
```

koja prima graf g kao parametar, a vraća duljinu najkrace staze i samu stazu u obliku [v1,v2,....,vn, v1]

Staza je šetnja u kojoj su svi brdovi različiti.

Ukoliko je graf Eulerovski najdulja zatvorena staza prolazi svim bridovima zadanog grafa, međutim ukoliko graf nije Eulerovski potrebno je provesti brisanje bridova zadanog grafa do Eulerovosti, te nakon što graf postane Eulerovski Fleuryevim algoritmom je potrebno pronaći Eulerovsku stazu. Povezani graf G je Eulerovski ako postoji zatvorena staza koja sadrži svaki brid od G. Prema Eulerovom teoremu graf G je Eulerovski onda i samo onda ako je stupanj svakog vrha paran broj.
Ukoliko graf nije eulerovski, brisnje bridova do eulerovosti vrši se ispunjavajući dolje navedene uvjete. Pri tome je potrebno paziti ukoliko graf postane nepovezan da se ispita svaka pojedinačna komponenta povezanosti.



Uvjeti koje treba ispunjavati su sljedeći:

1. Ukoliko zadani vrh ima samo jednog susjeda, potrebno je obrisati zadani vrh
2. Ukoliko vrh ima paran broj susjeda , zaobiđi ga
3. Ostali kriteriji odnose se na vrh koji ima neparan broj susjeda :
   a. Ukoliko ima sve susjede koji imaju paran stupanj, te ukoliko neki od njih ima neparan broj susjeda, obrisi zadani vrh
   b. Jedan neparan susjed, ukoliko nije most, obrisi bird
   c. Ukoliko je most i ne postoji niti jedna druga mogućnost, potrebno je odvojiti komponente i ispitati svaku komponentu zasebno
   d. Ukoliko zadani vrh ima 2 neparna susjeda za svakog gledam susjedi ako imaju parne susjede koji cine brid obrisi v v1, ako imaju jedan zajednicki parni vrh obrisi taj.
   e. Ukoliko zadani vrh ima 2 neparna te oni imaju neparne susjede i ako su povezani obrisi bilo koji brid
   f. Ako ima 3 susjeda koji su neparni, ukoliko neki od vrhova ima parnog susjeda obrisem brid koji se sastoji od trenutnog vrha i vrha koji ima parnog susjeda
   g. ukoliko vrh ima sve neparne susjede, ako vrhovi nemaju zajednicki vrh obrisi trenutni_vrh
   h. ako imaju zajednicki vrh onda bilo koji brid jednog od vrhova i trenutnog obrisi

Gore navedeni uvjeti omogućuju da se iz grafa obrise najmanji broj bridova da bi graf postao Eulerovski.
Nakon što je brisanjem bridova pridružavajući se zadanih kriterija dobiven Eulerovski graf, po Fleuryevom algoritmu pronađena je max zatvorena staza.

Neka je G Eulerovski graf, tada je sljedeća konstrukcija uvijek moguća i vodi do Eulerovske staze.
Započni u proizvoljnom vrhu u i šeći se vrhovima u bilo kojem redosljedu pazeći pritom na sljedeća pravila:

1. Prebriši bridove kojima si prošao, a ako nakon prolaska vrh ostane izoliran, pobriši i njega.
2. Prijeđi mostom samo ako nemaš druge mogućnosti

PROGRAMSKO RJEŠENJE

Metode kojima je je implementiran gore opisan postupak su :

- `get_eulerian_graph(self,g)` – implementacija samog postupka brisanja bridova (vrhova) do eulerovosti
- `ispitaj (self, vrh1, vrh2)` – pomoćna metoda koja ispituje parnost susjeda dva vrha
- `pronadji_eulerovsku_stazu(self,g)` – po Fleuryevom algoritmu pronalazi eulerovsku stazu
- `is_euler(self)` – provjerava je li zadani graf eulerovski
- `is_most(self,vrh1,vrh2)` – provjerava je li zadani brid most u grafu G
- `odvoji(self)` – ukoliko je brid most odvaja dvije komponente povezanosti

Za petersenov graf :

```
NAJKRACA 5 ['2', '1', '5', '4', '3', '2']
10 : ['8', '7', '1'] [illegible]
[illegible] ['5', '7', [illegible]
3 [illegible] ['8', '4', '7' [illegible]
7 [illegible] ['1', '9', '3' [illegible]
5 [illegible] ['4', '6', '1' [illegible]
4 [illegible] ['5', '3', '7' [illegible]
7 [illegible] ['9', '10', '4' [illegible]
6 [illegible] ['9', '5', '8' [illegible]
9 [illegible] ['6', '7', '2' [illegible]
8 [illegible] ['10', '3', '6' [illegible]
NAJDULJA 9 ['10', '8', '6', '5', '4', '3', '2', '9', '7', '10' [illegible]
```

Kotač 5:

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py treci pr kotac 5
ciklus 4
1 : ['2', '0']
0 : ['3', '1']
3 : ['0', '2']
2 : ['1', '3']
kotac 5
1 : ['2', '0', '4']
0 : ['3', '1', '4']
3 : ['0', '2', '4']
2 : ['1', '3', '4']
4 : ['0', '3', '1', '2']
NAJKRACA 3 ['4', '0', '3', '4']
1 : ['2', '0', '4']
0 : ['3', '1', '4']
3 : ['0', '2', '4']
2 : ['1', '3', '4']
4 : ['0', '3', '1', '2']
NAJDULJA 6 ['0', '1', '4', '3', '2', '4', '0']
```

Primjer ukoliko zadani graf sadrži most te je potrebno svaku komponentu povezanosti posebno

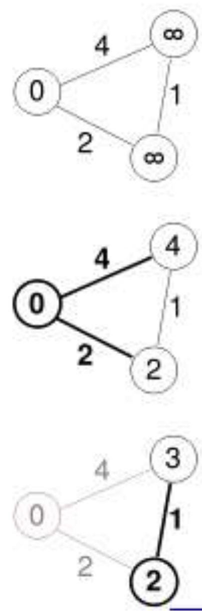
```
NAJKRACA 4 ['1', '2', '4', '3', '1']
1 : ['2', '3']
3 : ['4', '1', '5']
2 : ['1', '4']
5 : ['6', '7', '3']
4 : ['3', '2']
7 : ['8', '5']
6 : ['5', '8']
8 : ['7', '6']
1 : ['2', '3']
3 : ['4', '1']
2 : ['4', '1']
4 : ['3', '2']
graf je eulerovski
8 : ['7', '6']
5 : ['6', '7']
7 : ['8', '5']
6 : ['8', '5']
graf je eulerovski
NAJDULJA 4 ['5', '6', '8', '7', '5']
```

Kao što je vidljivo u primjerom, za svaku komponentu povezanosti pronađena je eulerovska staza, te je odabrana ona dulja. U ovom slučaju obje komponente povezanosti simetrične su te

PROBLEM NAJKRAĆEG

Implementirati Dijkstrin algoritam

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:
        dist[v] := infinity
        previous[v] := undefined
    dist[source] := 0
    Q := copy(Graph)
    while Q is not empty:
        u := extract_min(Q)
        for each neighbor v of u:
            alt = dist[u] + length(u, v)
            if alt < dist[v]
                dist[v] := alt
                previous[v] := u
```

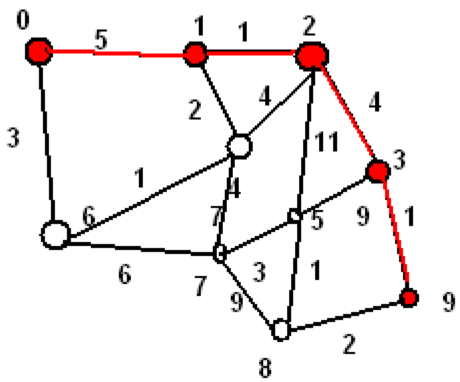


PROGRAMSKO RJEŠENJE

Zadani algoritam je implementiran u klasi `Dijkstra()` metodom

```
get_dijkstra(self,g,start='0',end=None)
```

koja vraća najkraću stazu između početne i završne točke zadanog grafa. Zadana klasa također sadrži pomoćnu metodu get_G(self,g) koja vraća listu susjedstva sa težinama bridova. Potrebno je naglasiti da zadani grafovi moraju biti težinski.




```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py cetvrti dat dijkstra
['0', '1', '6', '2', '4', '5', '3', '9', '7', '8']
{'7,8': ['9'], '0,6': ['3'], '5,8': ['1'], '4,7': ['7'], '3,9': ['1'], '0,1': ['5'],
 '3,5': ['9'], '1,4': ['2'], '1,2': ['1'], '5,7': ['3'], '6,7': ['6'], '8,9': ['2'],
 '2,5': ['11'], '2,4': ['4'], '2,3': ['4'], '4,6': ['1']}
1 : ['0', '4', '2']
0 : ['6', '1']
3 : ['9', '5', '2']
2 : ['1', '5', '4', '3']
5 : ['8', '3', '7', '2']
4 : ['7', '1', '2', '6']
7 : ['8', '4', '5', '6']
6 : ['0', '7', '4']
9 : ['3', '8']
8 : ['7', '5', '9']
DIJKSTRA 11 ['0', '1', '2', '3', '9']
```

PROBLEM TRGOVAČKOG PUTNIKA

Usporedi pohlepni heuristički algoritam i iscrpnu pretragu za nalaženje najkraćeg hamiltonovskog ciklusa u potpunom težinskom grafu.

Problem se svodi na traženje Hamiltonovog ciklusa najkraće duljine u potpunom težinskom grafu. Potpuni graf je uvijk hamiltonovski

HEURISTIČKI ALGORITAM

Algoritam je sljedeći:

1. Započni pretragu u bilo kojem vrhu
2. Odaberi sljedeći vrh najbliži zadanom vrhu koji nije još prijeđen.
3. Kada su obiđeni svi vrhovi vrati se u početni

U većini slučajeva ovakav pristup daje brzo i točno rješenje, međutim može se dogoditi dase odabirom najbližih vrhova koji nisu obiđeni propusti točno rješenje.

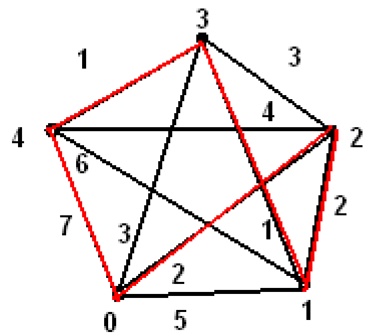
Iscrpna pretraga ispituje svaki mogući hamiltonovski ciklus u zadanom grafu, te odabire onaj najkrace duljine. Za velike grafove ovakav pristup vrlo jezahtjevan,međutim uvijek daje točno rješenje.

PROGRAMSKO RJEŠENJE

Zadatak je implementiran klasom `Salesman()`, te metodama

- `iscrpna_pretraga(self)` – implementacija iscrpne pretrage za trazenje hamiltonovog ciklusa minimalne duljine
- `heuristicki(se0:u1>rqw(t`

- `usporedi(self,g)` – metoda koja poziva implementacije




```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py peti dat tsp
['0', '1', '2', '3', '4']
{'0,4': ['7'], '0,3': ['4'], '0,2': ['2'], '0,1': ['5'], '3,4': ['1'], '1,4': ['6'],
 '1,2': ['2'], '1,3': ['1'], '2,4': ['4'], '2,3': ['3']}
1 : ['0', '4', '2', '3']
0 : ['4', '3', '2', '1']
3 : ['0', '4', '1', '2']
2 : ['0', '1', '4', '3']
4 : ['0', '3', '1', '2']
MIN 13 ['0', '4', '3', '1', '2', '0']
lokalna pohlepa
MIN 13 ['0', '2', '1', '3', '4', '0'] [('0', '2'), ('2', '1'), ('1', '3'), ('3', '4'
), ('4', '0')]
```

Primjer u kojem je moguće da se lokalnom pohlepom "promaši" točno rješenje:

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py peti dat tsp1
['0', '1', '2', '3', '4']
{'0,4': ['7'], '0,3': ['12'], '0,2': ['2'], '0,1': ['5'], '3,4': ['19'], '1,4': ['1'
], '1,2': ['2'], '1,3': ['3'], '2,4': ['4'], '2,3': ['3']}
1 : ['0', '4', '2', '3']
0 : ['4', '3', '2', '1']
3 : ['0', '4', '1', '2']
2 : ['0', '1', '4', '3']
4 : ['0', '3', '1', '2']
MIN 16 ['0', '4', '1', '3', '2', '0']
lokalna pohlepa
MIN 36 ['0', '2', '1', '4', '3', '0'] [('0', '2'), ('2', '1'), ('1', '4'), ('4', '3'
), ('3', '0')]
```

U zadanom grafu prebroji sva razapinjuća stabla. Ispiši barem jedno od njih.

Neka je zadan povezani graf G s n vrhova. Neka su $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$ vrijednosti Laplaceove matrice različiti od 0. Tada je broj razapinjujućih stabala:

$$t(G) = \frac{1}{n} \lambda_1 \lambda_2 \cdots \lambda_{n-1} .$$

LAPLACEOVA MATRICA(KIRCHOFFOVA):

Laplaceova matrica je jedan od oblika prezentacije grafa G . Ukolko je zadan graf G matrica ima sljedeći oblik

$$l_{i,j} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ adjacent } v_j \\ 0 & \text{otherwise} \end{cases}$$

Odnosno:

za i različito od j :

  i. ako je vrh i susjedan vrhu j u zadanom grafu G, $q_{i,j} = -1$
  ii. U ostalim slučajevima $q_{i,j}$ je 0

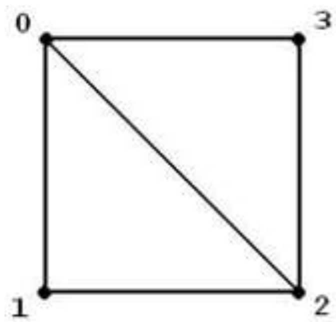
Za i = j:

  i. $q_{i,j}$ je jednak stupnju vrha i

Broj razapinjućih stabala u zadanom grafu računa se na sljedeći način:

- potrebno je konstruirati Laplaceovu matricu Q
- Zadanoj matrici potrebno je obrisati bilo koji stupac s i bilo koji red r-> Q'
- Broj razapinjućih stabala jednak je determinanti matrice Q'

PRIMJER:



$$Q = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

$$Q^* = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

PROGRAMSKO RJEŠENJE

Implementacije oba zadatka vezana uz stabla izvedena su klasom `Stabla()`. Metode koje su vezane uz zadani zadatak a nalaze se u klasi `Stablo()` su sljedeće:

- `ispisi_razapinjujuce_stablo(self –` bilo koje razapinjujuće stablo
- `kirchoff(self):-` prema kirchoffovom teoremu računa broj razapinjućih stabala u zadanom grafu

Za realizaciju zadatka ukljuceni su moduli za rad s matricama : `numarray`, `numeric`, `LinearAlgebra`

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py sesti dat petersen
['1', '2', '5', '10', '3', '9', '4', '8', '7', '6']
{'6,9': [1], '8,10': [1], '3,8': [1], '4,5': [1], '7,9': [1], '3,4': [1], '5,6': [1]
, '7,10': [1], '1,5': [1], '1,2': [1], '1,10': [1], '6,8': [1], '2,9': [1], '4,7': [
1], '2,3': [1]}
10 : ['8', '7', '1']
1 : ['5', '2', '10']
3 : ['8', '4', '2']
2 : ['1', '9', '3']
5 : ['4', '6', '1']
4 : ['5', '3', '7']
7 : ['9', '10', '4']
6 : ['9', '5', '8']
9 : ['6', '7', '2']
8 : ['10', '3', '6']
Razapinjuce stablo [('1', '2'), ('2', '3'), ('3', '4'), ('4', '5'), ('5', '6'), ('6'
, '9'), ('9', '7'), ('7', '10'), ('10', '8')]
Broj razapinjucih stabala 2000
```

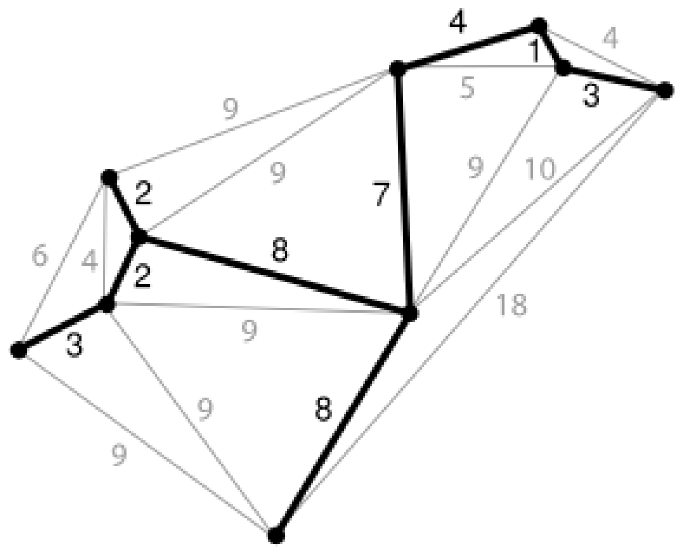
Za zadani Petersenov graf ispiseano je razapinjuće stablo , te je broj razapinjućih stabala 2000.

KOCKA (4):

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py sesti pr kocka 4
kocka 4
11 : ['3', '9', '10', '15']
10 : ['2', '8', '11', '14']
13 : ['5', '9', '12', '15']
12 : ['4', '8', '13', '14']
15 : ['7', '11', '13', '14']
14 : ['6', '10', '12', '15']
1 : ['0', '3', '5', '9']
0 : ['1', '2', '4', '8']
3 : ['1', '2', '7', '11']
2 : ['0', '3', '6', '10']
5 : ['1', '4', '7', '13']
4 : ['0', '5', '6', '12']
7 : ['3', '5', '6', '15']
6 : ['2', '4', '7', '14']
9 : ['1', '8', '11', '13']
8 : ['0', '9', '10', '12']
Razapinjuce stablo [('0', '1'), ('1', '3'), ('3', '11'), ('11', '10'), ('10', '14'),
 ('14', '12'), ('12', '13'), ('13', '15'), ('15', '7'), ('7', '5'), ('5', '4'), ('4'
, '6'), ('6', '2'), ('13', '9'), ('9', '8')]
Broj razapinjucih stabala 42467328
```

ZADATAK

U zadanom potpunom težinskom grafu nađi razapinjuće stablo minmalne ukupne duljine.

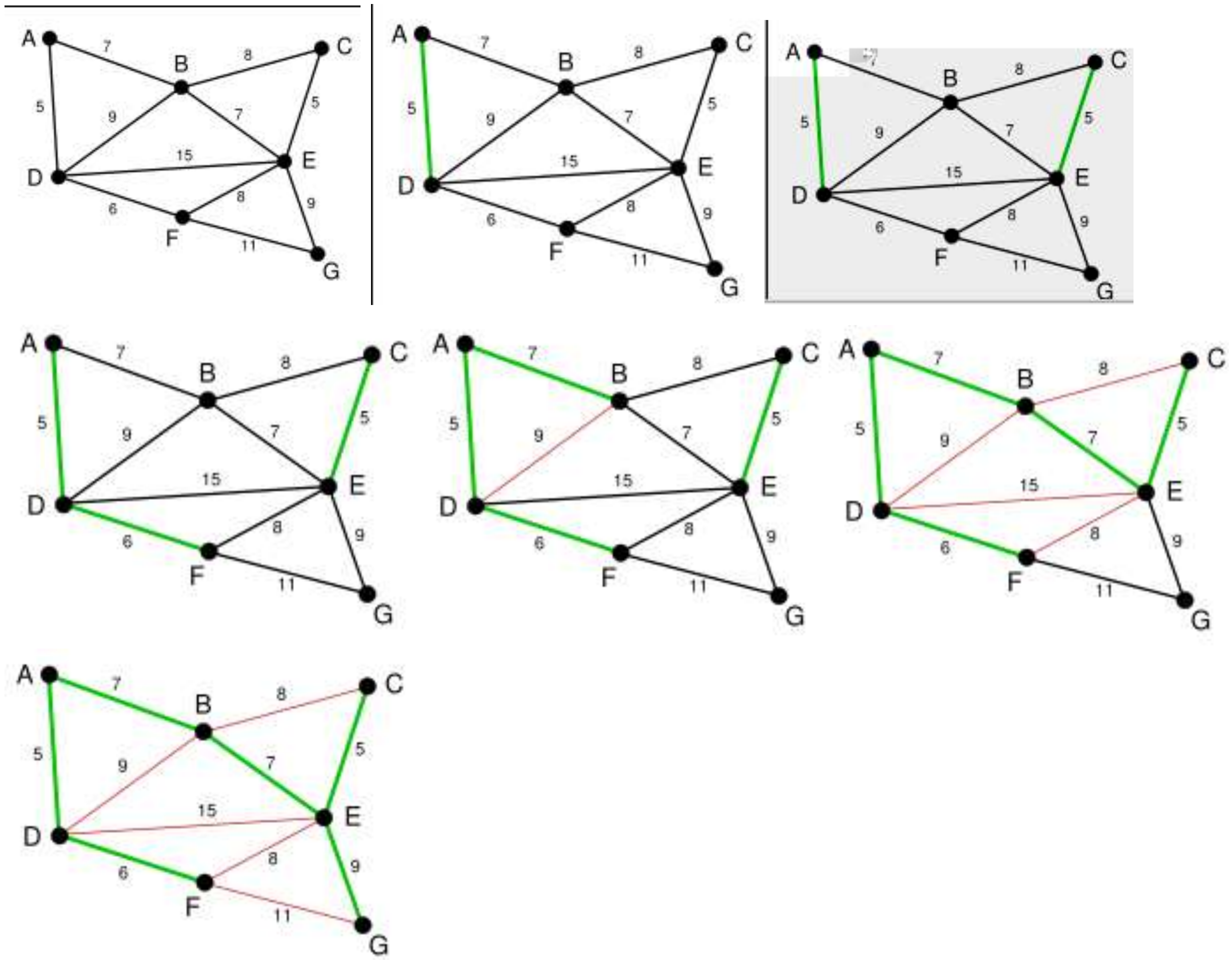


Zadani algoritam je pohlepan algoritam. Odabire bridove najmanje tezine pazeći pri tome da u svakom sljedećem koraku odabere vrh koji još nije prijeđen, te da ne zatvori ciklus.

PSEUDOKOD:

```
function Kruskal(G)
  for each vertex v in G do
    Define an elementary cluster C(v) ← {v}.
  Initialize a priority queue Q to contain all edges in G, using the weights as keys.
  Define a tree T ← Ø       //T will ultimately contain the edges of the MST
   // n is total number of vertices
  while T has fewer than n-1 edges do
    // edge u,v is the minimum weighted route from/to v
    (u,v) ← Q.removeMin()
    // prevent cycles in T. add u,v only if T does not already contain an edge consisting of u and v.
    // Note that the cluster contains more than one vertex only if an edge containing a pair of
    // the vertices has been added to the tree.
    Let C(v) be the cluster containing v, and let C(u) be the cluster containing u.
    if C(v) ≠ C(u) then
      Add edge (v,u) to T.
      Merge C(v) and C(u) into one cluster, that is, union C(v) and C(u).
  return tree T
```

PRIMJER:



PROGRAMSKO RJEŠENJE

Kruskalov algoritam implementiran je u klasi `Stabla()` glavnom metodom:

- `kruskal(self,g)`

te pomoćnim metodama

- `obrisi_brid_ciklus(self)` – brise brid iz grafa koji bi omogućio zatvaranje ciklusa
- `pronadji_ciklus(self,brid)` – pronalazi mogućnost ciklusa u grafu
- `postoji_zajednicki_brid(self,brid1,brid2)` – pronalazi zajednicki brid između 2 brida koja su već prijeđena

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py sedmi dat foo
['0', '1', '11', '2', '12', '9', '3', '4', '5', '6', '8', '7', '10']
{'7,8': ['4'], '5,8': ['1'], '8,10': ['6'], '4,5': ['1'], '0,1': ['1'], '3,4': ['2']
, '5,6': ['3'], '4,9': ['8'], '1,2': ['1'], '5,12': ['3'], '10,11': ['3'], '6,8': ['
2'], '1,12': ['3'], '2,9': ['7'], '6,7': ['6'], '8,9': ['7'], '9,10': ['2'], '11,12'
: ['2'], '0,11': ['2'], '9,12': ['4'], '2,3': ['9']}
11 : ['10', '12', '0']
10 : ['8', '11', '9']
12 : ['5', '1', '11', '9']
1 : ['0', '2', '12']
0 : ['1', '11']
3 : ['4', '2']
2 : ['1', '9', '3']
5 : ['8', '4', '6', '12']
4 : ['5', '3', '9']
7 : ['8', '6']
6 : ['5', '8', '7']
9 : ['4', '2', '8', '10', '12']
8 : ['7', '5', '10', '6', '9']
KRUSKAL
put {'7,8': 4, '5,8': 1, '0,1': 1, '4,5': 1, '3,4': 2, '1,2': 1, '10,11': 3, '6,8':
2, '5,12': 3, '9,10': 2, '11,12': 2, '0,11': 2}
```

Ispitaj je li zadani graf planaran.

Graf G je planaran ukoliko se može smjestiti u ravninu bez presjecanja, tako da se bridovi grafa geometrijski ne sijeku u niti jedno jtočki osim u krajnjim točkama.

Kako je poznato da grafovi K5 i K33 nisu planarni, ispitivanje je li zadani graf planaran svodi se na pokušaj svođenja zadanog grafa na K5 ili K33 :

TM1. Graf G je planaran ako ne sadrži podgraf homeomorfan s K_5 ili K_{33}

TM2. Graf G je planaran ako ne sadrži podgraf stezljiv do K_5 ili K_{33}

PROGRAMSKO RJEŠENJE

Problem ispitivanja je li zadadani graf G planaran ili ne implementiran je u klasi `Planarnost()` koja se sastoji od glavne metode za ispitivanje planarnosti te nekoliko pomoćnih metoda. Algoritam se svodi na brisanje bridova te na stezanje bridova pokušavajući dobiti K_5, odnosno K_{33} kao pograf :

- `planarity(self,g)` – glavna metoda za ispitivanje planarnosti zadanog grafa
- `non_planar(self)` – pomoćna metoda koja na temelju parametara (broj vrhova i bridova) pokušava odrediti je li zadani graf planaran ili ne
- `stegni(self,v1,v2)` – pomoćna metoda koja steže zadani brid
- `sadrzi_K5(self)` – provjerava sadrži li zadani graf podgraf K_5
- `sadrzi_K33(self)` – provjerava sadrži li zadani graf podgraf $K_{3,3}$

PETERSEN

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py osmi dat petersen 0 0
['1', '2', '5', '10', '3', '9', '4', '8', '7', '6']
{'6,9': [1], '8,10': [1], '3,8': [1], '4,5': [1], '7,9': [1], '3,4': [1], '5,6': [1]
, '7,10': [1], '1,5': [1], '1,2': [1], '1,10': [1], '6,8': [1], '2,9': [1], '4,7': [
1], '2,3': [1]}
10 : ['8', '7', '1']
1 : ['5', '2', '10']
3 : ['8', '4', '2']
2 : ['1', '9', '3']
5 : ['4', '6', '1']
4 : ['5', '3', '7']
7 : ['9', '10', '4']
6 : ['9', '5', '8']
9 : ['6', '7', '2']
8 : ['10', '3', '6']
planarnost
{'3': ['8', '4', '5', '6'], '5': ['4', '6', '8', '3'], '4': ['5', '3', '8', '6'], '6
': ['5', '8', '3', '4'], '8': ['3', '6', '5', '4']}
Sadrzi K5
nije planaran
```

KOCKA (4)

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py osmi pr kocka 4
kocka 4
11 : ['3', '9', '10', '15']
10 : ['2', '8', '11', '14']
13 : ['5', '9', '12', '15']
12 : ['4', '8', '13', '14']
15 : ['7', '11', '13', '14']
14 : ['6', '10', '12', '15']
1 : ['0', '3', '5', '9']
0 : ['1', '2', '4', '8']
3 : ['1', '2', '7', '11']
2 : ['0', '3', '6', '10']
5 : ['1', '4', '7', '13']
4 : ['0', '5', '6', '12']
7 : ['3', '5', '6', '15']
6 : ['2', '4', '7', '14']
9 : ['1', '8', '11', '13']
8 : ['0', '9', '10', '12']
planarnost
{'15': ['7', '3', '2', '4'], '3': ['2', '7', '4', '15'], '2': ['3', '4', '7', '15'],
 '4': ['3', '2', '7', '15'], '7': ['3', '15', '2', '4']}
Sadrzi K5
nije planaran
```

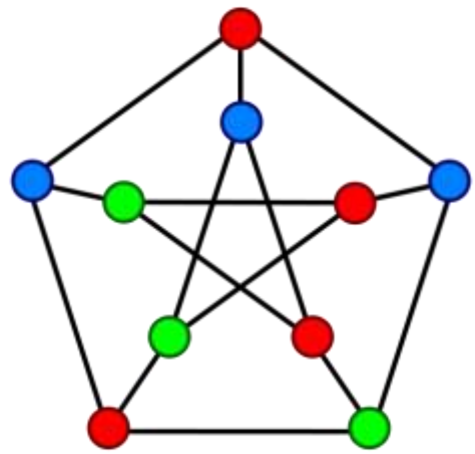
KOCKA(3):

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py osmi pr kocka 3
kocka 3
1 : ['0', '3', '5']
0 : ['1', '2', '4']
3 : ['1', '2', '7']
2 : ['0', '3', '6']
5 : ['1', '4', '7']
4 : ['0', '5', '6']
7 : ['3', '5', '6']
6 : ['2', '4', '7']
planarnost
graf je planaran
root@ubuntu:/home/gizmo/teograf#
```

BIPARTITNI (4,4):

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py osmi pr bipartitni 4 4
1 : ['6', '7', '4', '5']
0 : ['7', '6', '5', '4']
3 : ['7', '4', '5', '6']
2 : ['5', '4', '7', '6']
5 : ['0', '1', '3', '2']
4 : ['0', '1', '3', '2']
7 : ['0', '1', '3', '2']
6 : ['0', '1', '2', '3']
potpuni bipartitni 4 4
planarnost
{'1': ['6', '7', '4', '5'], '0': ['7', '6', '5', '4'], '3': ['7', '4', '5', '6'], '2
': ['5', '4', '7', '6'], '5': ['0', '1', '3', '2'], '4': ['0', '1', '3', '2'], '7':
['0', '1', '3', '2'], '6': ['0', '1', '2', '3']}
SADRZI K33
Nije planaran
root@ubuntu:/home/gizmo/teograf#
```

Ispitaj je li zadani graf 3-obojiv (v), tj. mogu li se njegovi vrhovi obojati s 3 boje, tako da su susjedni vrhovi raznobojni.



Algoritam za ispitivanje vršne 3-obojivosti zadanog grafa je sljedeći:

1. odaberi proizvoljan vrh v1
2. za svaki vrh susjedan s v1 provjeri imaju li zajedničke susjede.
3. Ukoliko zadani vrhovi imaju zajedničke susjede provjeri čine l ti susjedi brid. Ukoliko susjedi ne čine brid, ponovi algoritam dok ne obiđeš sve vrhove
   Ako zadani susjedi čine brid , graf nije vršno 3 –obojiv
4. Ako su provjereni svi vrhovi, graf je vršno 3 obojiv

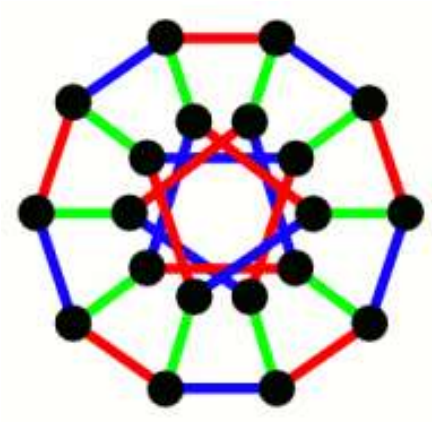
PROGRAMSKO RJEŠENJE

Ispitivanje vršne I bridne obojivosti implementirano je klasom `Bojanja()` dok je specificno provjera vršne 3-obojivosti implementirana je metodama:

- `susjedstvo_1(self)` ko niti jedna 2 vrha nemaju 2 zajednicka susjeda onda su oni sigurno 3 obojivi
- `is_susjedstvo_vise(self)`– ukoliko pojedini vrhovi imaju vise od jednog zajednickog susjeda provjeri da li ti susjedi cine brid
- `vrh_3_obojiv(self,g)` pozivajući pomoćne metode ispituje je li graf vršno 3-obojiv

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py deveti dat petersen 0 0
['1', '2', '5', '10', '3', '9', '4', '8', '7', '6']
{'6,9': [1], '8,10': [1], '3,8': [1], '4,5': [1], '7,9': [1], '3,4': [1], '5,6': [1]
, '7,10': [1], '1,5': [1], '1,2': [1], '1,10': [1], '6,8': [1], '2,9': [1], '4,7': [
1], '2,3': [1]}
10 : ['8', '7', '1']
1 : ['5', '2', '10']
3 : ['8', '4', '2']
2 : ['1', '9', '3']
5 : ['4', '6', '1']
4 : ['5', '3', '7']
7 : ['9', '10', '4']
6 : ['9', '5', '8']
9 : ['6', '7', '2']
8 : ['10', '3', '6']
1
po prvom kriteriju
zadani graf je vrsno 3 obojiv
```

Ispitaj je li zadani graf bridno 3-obojiv, tj. mogu li se njegovi bridovi obojati s 3 boje, tako da su susjedni bridovi raznobojni.



Za zadani graf najprije se provjerava stupanj svakog vrha. Ukoliko postoji vrh stupnja većeg od 3 zadani graf nije bridno 3-obojiv. Ukoliko je najveći stupanj vrha 3 vrši se ispitivanje. Na početku za svaki vrh postoji mogućnost bojanja s jednom od 3 boje. Boja za svaki vrh odabire se na osnovu jednog od sljedećih slučajeva:

mogućnosti:

- 3 3 (3) - potpuno opcenito odaberi koji brid obojati kojom bojom
- 2 2 -ovdje nije opcenito je potrebno nego odabrati za svaki brid jednu boju , a drugu mogucnost s trenutnim stanjem staviti na stog
- 1 2- obojati onaj koji ima jednu mogucnost s onom kojom se može obojati, a drugu s ovom koja ostane
- 1 i ima jos u listi vise od 1 vrhova koji nisu prijedjeni - prijedji taj vrh i idi dalje
- 1 i ima u listi jedan vrh koji je upravo susjed - obojaj brid i vrati 1 jer je moguce graf
  obojati bridno s 3 boje
- 0 i ima na stogu - uzmi sa stoga mogucnost bojanja i kreni s tim
- 0 i nema na stogu - vrati 0 jer su obidjene sve mogucnosti, nije moguce obojat s 3 boje

PROGRAMSKO RJEŠENJE

Ispitivanje bridne 3-obojivosti implementirano je metodom

```
provjeri_3_bridno(self)
```

koja pokusava obojati zadani graf na temelju gore navedenih mogućnosti

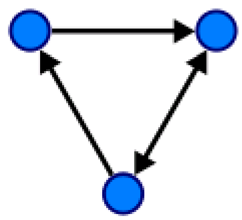
PRIMJER

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py deseti dat petersen 0 0
['1', '2', '5', '10', '3', '9', '4', '8', '7', '6']
{'6,9': [1], '8,10': [1], '3,8': [1], '4,5': [1], '7,9': [1], '3,4': [1], '5,6': [1]
, '7,10': [1], '1,5': [1], '1,2': [1], '1,10': [1], '6,8': [1], '2,9': [1], '4,7': [
1], '2,3': [1]}
10 : ['8', '7', '1']
1 : ['5', '2', '10']
3 : ['8', '4', '2']
2 : ['1', '9', '3']
5 : ['4', '6', '1']
4 : ['5', '3', '7']
7 : ['9', '10', '4']
6 : ['9', '5', '8']
9 : ['6', '7', '2']
8 : ['10', '3', '6']
graf nije 3 bridno obojiv
```

ŠETANJE PO USMJERENIM GRAFOVIMA

U zadanom težinskom usmjerenom grafu nađi kritični put.

Usmjereni graf ili digraf D sastoji de od konačnog nepovezanog skupa vrhova V(D) različitih od 0 i konačne familije A(D) uređenih parova od V(D)



Problem kritičnog puta poznat je ne samo u teoriji grafova nego i u s
u timovima, problema paralelnog izvršavanja poslova, …

Primjer opisa problema u ekonimskim i poslovnim razmjerima je sljedeći:
treba dovršiti prema planu da bi cjelokupni projekt bio dovrš
radnom planu. Ako aktivnost koja se nalazi na kritičnom putu kasni za jedan dan, čitav projekt će kasniti za jedan dan. (osim u slučaju da se neka druga aktivnost, također na kritičnom putu, može ubrzati za

U teoriji grafova svodi se na pronalazak puta koji ima najveću moguću težinu krenuvši od početne do konačne točke u zadanom grafu. Pritom prelaskom određenih točaka zbrajaju se težine bridova prateći usmjerenjegrafa.

Zadani problem rješava se modifikacijom Dijskstrinog algoritma s pojednostavljenjem zbog zadane usmjerenosti grafa.

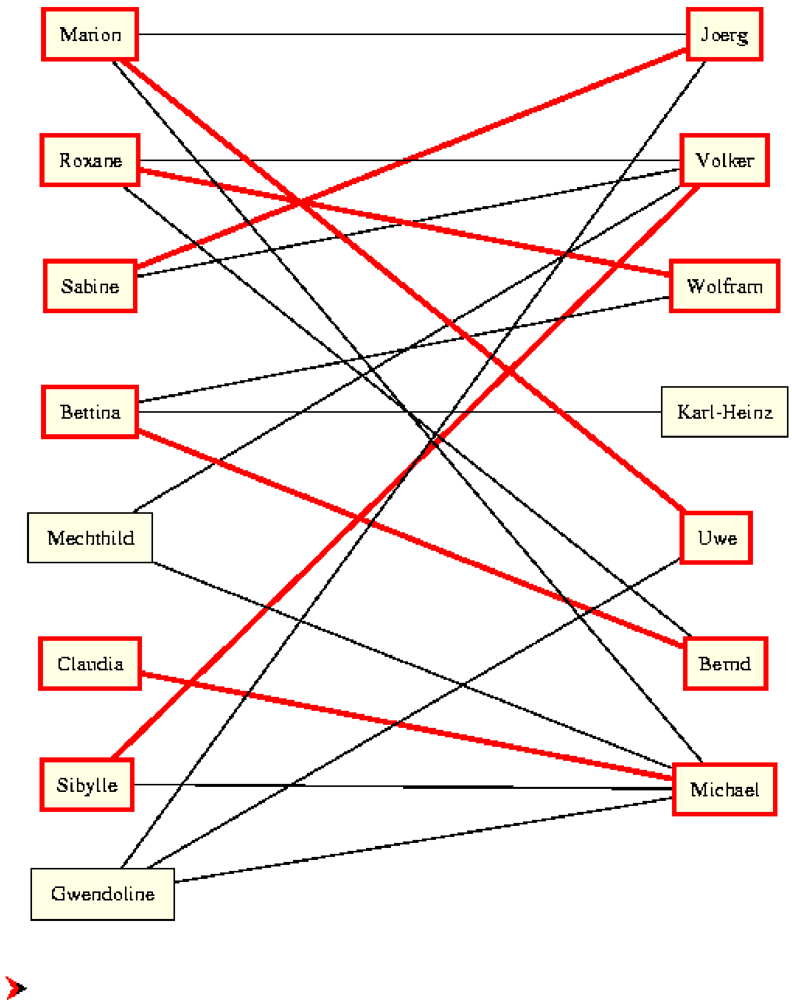
PROGRAMSKO RJEŠENJE

Zadani problem implementiran je klasom `Setnje()` te glavnom metodom za implementaciju pronalaska kriticnog puta, te pomoćnom metodom koja generia strukturu usmjerene liste susjedstva:

- `kritican_put(self,g)` – glavna metoda koja pronalazi kritčan put u zadanom grafu . Ulazni parametar je pokazivač na graf g, a izlaz je kritičan put
- `usmjerena_lista_susjedstva(self)` – pomoćna metoda koja vraća usmjerenu listu susjedstva koja je prikaz usmjerenog grafa i koristi se pri pronalasku kritičnog puta

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py jedanaesti dat usmjereni
['1', '2', '4', '3', '6', '5', '11', '9', '7', '10', '8', '12']
{'1,4': ['2'], '1,2': ['3'], '10,12': ['2'], '2,4': ['2'], '2,6': ['5'], '6,5': ['2'
], '6,4': ['3'], '2,3': ['4'], '7,8': ['6'], '9,7': ['1'], '5,9': ['5'], '4,5': ['6'
], '3,5': ['6'], '4,9': ['1'], '5,7': ['9'], '10,8': ['5'], '3,11': ['3'], '8,12': [
'7'], '7,10': ['5'], '5,11': ['1'], '9,10': ['3'], '11,12': ['4']}
11 : ['3', '5', '12']
10 : ['12', '8', '7', '9']
12 : ['10', '8', '11']
1 : ['4', '2']
3 : ['2', '5', '11']
2 : [ , 4 , 6 , 3 ]
'7', '11']      5 : [ 6 , 9 , 4 , 3 ,
'9']            4 : [ , 2 , 6 , 5 ,
                7 : [ 8 , 9 , 5 , 10
                6 : [ 2 , 5 , 4 ]
                9 : [ 7 , 5 , 4 , 10
                8 : [ 7 , 10 , 12 ]
MAX 43
KRITICAN PUT ['1', '2', '6', '4', '5', '7', '10', '8', '12']
```

U zadanom bipartitnom grafu s obilježenim vrhovima nađi i prebroji sva potpuna sparivanja.



Potpuno sparivanje iz V1 u V2 u bipartitnom grafu G(V1,V2) je bijektivna korespodencija vrhova skupa V1 I nekog podskupa skupa vrhova V2, takva das u korespodentni vrhovi susjedni.

Zadani problem poznat je kao ženidbeni problem.

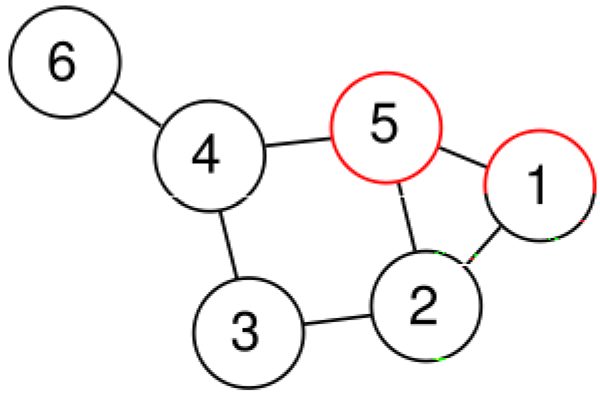
PROGRAMSKO RJEŠENJE

Zadani problem implementiran je klasom `Sparivanja()` te glavnom metodom koji pronalazi i broji sva potpuna sparivanja , te pomoćnom metodom koja radi jednostavnijeg ispitivanja boji vrhove bipartitnog grafa u crne i bijele:

- `potpuna_sparivanja(self,g)` – glavna metoda koja pronalazi I broji sva potpuna sparivanja u zadanom grafu
- `color_bipartite(self)` – pomoćna metoda za bojanje bipartitnih grafova

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py dvanaesti pr bipartitni 3 3
1 : ['4', '5', '3']
0 : ['5', '4', '3']
3 : ['0', '1', '2']
2 : ['5', '4', '3']
5 : ['0', '1', '2']
4 : ['0', '1', '2']
potpuni bipartitni 3 3
rezultat 27
```

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py dvanaesti pr bipartitni 3 4
1 : ['6', '4', '5', '3']
0 : ['6', '5', '4', '3']
3 : ['0', '1', '2']
2 : ['5', '4', '6', '3']
5 : ['0', '1', '2']
4 : ['0', '1', '2']
6 : ['0', '1', '2']
potpuni bipartitni 3 4
rezultat 64
```



Clique problem (problem grupe) je NP-potpuni problem. Problem je predstavljen kao jedan od Richard Knapovih originalnih 21 problema 1972. "Reducibility Among Combinatorial Problem.

Grupa u terminologiji teorije grafova predstavlja podgraf u zadanom grafu G koji čini potpuni graf (svi vrhovi grupe su međusobno povezani).

Problem grupe svodi se na problem traženja grupe u grafu G veličine barem k. Prateći optimizacijski problem je tzv maximum clique problem, tj. Problem pronalaska najveće moguće grupe u zadanom grafu.

Za rješavanje problema pronalaska grupe veličine barem k potrebno je primijeniti iscrpnu pretragu. Potrebno je pronaći podgraf sa barem k vrhova te provjeriti je li zadani podgra grupa ili ne.

Broj mogućih grupa veličine k grafa G veličine V je:

$$\binom{V}{k} = \frac{V!}{k!(V-k)!}$$

Heuristička metoda svodi se na to da je svaki vrh grupa veličine 1. Povezivanje grupa u veći grupu moguće je obaviti ukoliko svaki vrh iz grupe A je susjedan sa svakim vrhom iz grupe B. Vrijeme izvođenja zadanog algoritma je linearno, međutim moguće je da se ovakovom metodom ne pronađe najveća grupa ukoliko su vrhovi “velike” grupe u nekom od koraka spojeni s vrhovima koji nisu u “velikoj” grupi.

PROGRAMSKO RJEŠENJE

Maximum clique problem programski je riješen klasom `Clique()`

```
potpuni(self,g)
```

Koja u zadanom grafu pronalazi najveći potpuni podgraf.

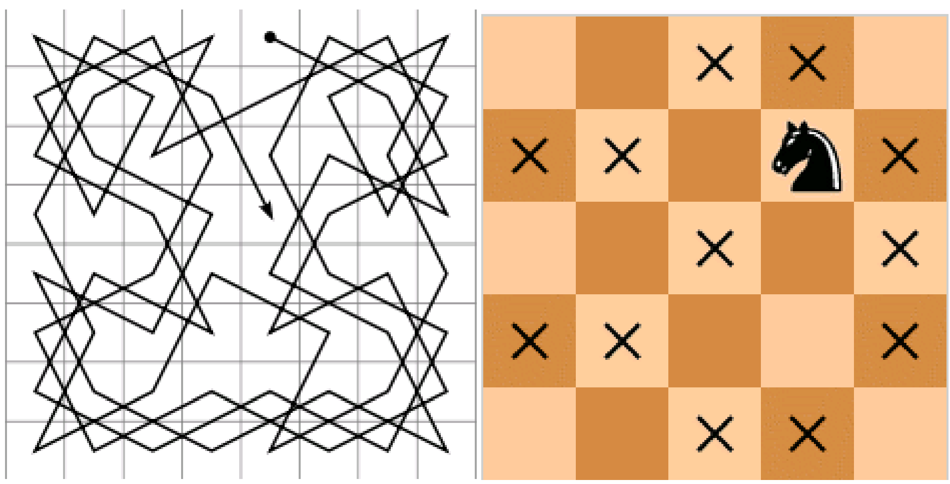
```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py trinaesti pr kotac 5
ciklus 4
1 : ['2', '0']
0 : ['3', '1']
3 : ['0', '2']
2 : ['1', '3']
kotac 5
1 : ['2', '0', '4']
0 : ['3', '1', '4']
3 : ['0', '2', '4']
2 : ['1', '3', '4']
4 : ['0', '3', '1', '2']
Max potpuni podgraf 3
root@ubuntu:/home/gizmo/teograf#
```

PROBLEM SKAKAČA (KNIGHT'S PROBLEM)

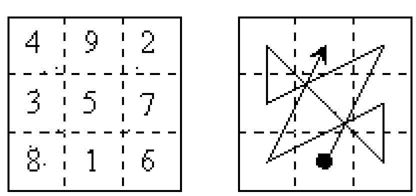
Ideja za promatranjem skakača na šahovskoj ploči seže u 18.st. , kada je švicarski matematičar Leonhard Euler predstavio svoj problem u Berlinu 1758. Problem je najčešće poznat pod imenom Springerproblem ili The knight's problem I glasi:

> Postoji li put kojim skakač može posjetiti svako polje šahovske ploče točno jednom kretajući se prema pravilima u šahu krećući s proizvoljnog polja šahovske ploče.

Problem je vrlo zanimljiv zbog specifičnog načina na koji se skakač kreće na šahovskoj ploči (u obliku slova L alternirajući između crnih i bijelih polja).



Iako je problem postao poznat u 18.st najranije spominjanje problema s pločama seže još u Kinu 2200 pr.Kr pronalaskom magičnog kvadrata veličine 3x3.



Prelaskom polja magičnog kvadrata zadanim redosljedom prema numeraciji polja vidimo veliku sličnost sa retanjem skakača te simetričnu putanju koja je vrlo zanimljiva i u problem skakačevog puta.

Mnogo općenitiji problem svodi se na proučavanje u kojim slučajevima je moguće konstruirati skakačev put , točnije za koju veličinu ploče. Rješenje tog problema opisuje Schwenkov teorem dan u nastavku.

Za ploču veličine $m \times n$, pri čemu je m manje ili jednako n, postoji skakačev put u svim slučajevima osim u dolje navedenima:

1. m i n su oboje neparni
2. m = 1, 2, ili 4; m i n su oboje različiti od 1
3. m = 3 i n = 4, 6, ili 8

Ukoliko je prvi uvijet ispunjen, vrlo je jednostavno pokazati da ne postoji skakačev put. Ukoliko zamislimo šahovsku ploču čija su polja obojana u dvije boje, u svakom koraku obilaženja polja skakač obilazi polje suprotne boje (c-b-c-b…).

Kako su m i n oba neparni brojevi, broj bijelih polja i broj crnih polja je različit. Npr za ploču veličibe 5x5 postoji 13 polja jedne boje te 12 polja druge boje.

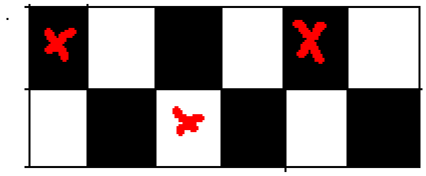
Da bi postojao skakačev put potrebno je obići jednak broj bijelih te jednak broj crnih polja. Kako su m i n neparni, postoji različit broj crnih i bijelih polja, te ne postoji hamiltonovski ciklus.

U drugom uvjetu vrijedi: ukoliko je "kraća" stranica duljine 1,2 ili 4, ne postoji skakačev put.

Jednostavno je pokazati da uvjet vrijedi kada je m 1 ili 2, s iznimkom trivijalnog slučaja 1x1.

Ukoliko je m= 1 nije moguće obići niti jedno drugo polje. Te, u skladu s time nije moguće obići sva polja.

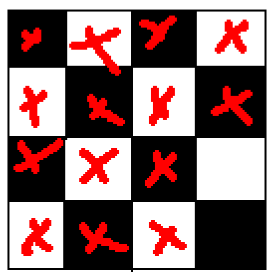
Za m = 2:

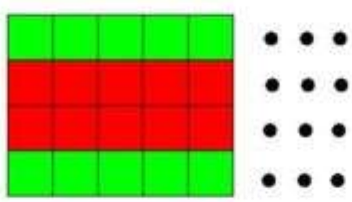


Vidljivo je da nije moguće obići sva polja jer je moguće napredovati prema naprijed kolika je duljina n, kada više nije moguće napredovati, jedini mogući potez je vraćanje na polje koje je prethodno obiđeno (što nije dopušteno), te za m=2 nije moguć skakačev put.

Situacija za m = 4, je nešto složenija, međutim moguće je pokazati da za ploču veličine 4x n nije moguć skakačev put.

Pretpostavimo suprotno, da za ploču veličine 4 x n postoji skakačev put. Konstruirajmo da skupa polja A_1 i B_1, A_1 sadrži prvu polovicu polja, sa B_1 drugu polovicu polja, prema bojama na šahovskoj ploči neka su A_1 polja bijele boje, a B_1 polja crne boje.





Definirajmo skup A_2 kao skupa zelenih polja te skup B_2 kao skup crvenih polja kako je prikazano na slici. Broj crvenih i zelenih polja je jednak. Primjetimo da ukoliko se skakač nalazi na polju iz skupa A_2 mora skočiti na polje iz skupa B_2 , te ukoliko se nalazi na polju B_2 skakač prelazi na polje A_2.

Ukoliko slijedimo skakača, dolazimo do kontradikcije s prethodnom tvrdnjom:

1. Prvo polje s kojeg kreće skakač je polje koje pripada skupu A_1 i skupu A_2. Ukoliko tvrdnja ne vrijedi, zamijenimo A_2 i B_2 kako bi tvrdnja vrijedila.

2. Drugo polje mora biti element iz skupova B_1 i B_2
3. Treće polje mora biti iz skupova A_1 i A_2
   ... tako dalje

Promatrajući skupove, skup A_1 ima jednake elemente kao I A_2 ,te B_1 ima jednake elemente kao i B_2. Međutim to nije istina zbog toga što je crveno-zeleni uzorak različit od šahovske ploče. Iz toga proizlazi da je pretpostavka pogrešna te ne postoji skakačev put ploče 4 x n, za svaki n

TREĆI UVJET

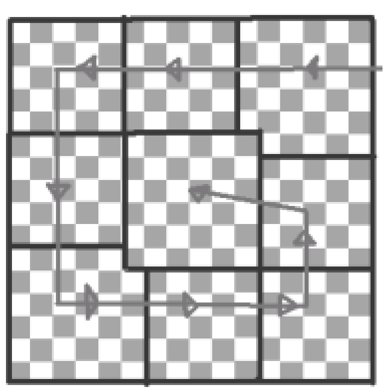
Treći uvijet ispituje se posebno za slucajeve 3x4, 3x6 te 3x8 pokušavajući konstruirati skakačev put, te se pokazuje da to nije moguće.

Treba napomenuti da dokazivanje prethodna 3 uvjeta ne dokazuje prethodni teorem. Još je potrebno dokazati da sve ploče koje ne spadaju u zadana 3 uvjeta spadaju u kategoriju koji imaju skakačev put.

SKAKAČEVOG

Problem pronalaska jednostavnog rešenja skakačevog puta za bilo koju ploču za koju je rješenje moguće riješen je 90-ih godina prošlog stoljeća kao project grupe njemačkih studenata.

Osnovna ideja je jednostavna rekurzija. Ploča se dijeli na manje ploče za koje je jednostavno pronači skakačev put. Jedini problem je bio pronaći skakačev put za zadanu početnu i konačnu točku da bi se na točno određenim mjestima mogli putovi spojiti u cjelovit skakačev put zadane ploče. Osnovna ideja pronalaska zadanog puta za ploču 16x 16 prikazan je na slici. Opis ovog rješenja nije obuhvaćen ovim radom nego je u nastavku jednostavno opisan jedan od mogućih algoritama za pronalazak skakačevog puta.



Najjednostavniji algoritam za pronalazak puta je *backtracking algoritam* koji radi na principu kretanja skakača u određenom smjeru, međutim ukoliko sljedeći korak nije moguć, a nije pronađen put, vraća se na prethodni korak i pokušava nastaviti obilazak u nekom od mogućih putova. Ukoliko nije moguće krenuti niti jednim drugim putem vraća se na prethodni korak. Iz opisa algoritma vidljiva je njegova vremenska složenost.

Primjer konstrukcije puta za ploču veličine 8 x 8 prikazan je na slici:

18	59	50	1	48	15	22	63
51	2	17	60	21	64	47	14
58	19	4	49	16	23	62	45
3	52	57	20	61	46	13	24
34	5	40	53	36	25	44	11
39	56	35	8	41	12	29	26
6	33	54	37	28	31	10	43
55	38	7	32	9	42	27	30

Back tracking algoritam implementiran je za pronalazak puta skakača na šahovskoj ploči

Veličine n x m za koju postoji skakačev put uz zadane koordinate skakača na ploči.. Skakač se pomiče u obliku slova L u bilo kojem smjeru za svako polje koje nije označeno. Ukoliko nemože više napredovat, vraća se na prethodnu poziciju te se pokušava kretati u nekom od mogućih smjerova.

PROGRAMSKO RJEŠENJE

`Knight()` koja u sebi sadrži glavnu metodu za pronalazak skakačevog puta te nekoliko pomoćnih metoda:

- `knights_tour` – glavna metoda za pronalazak skakačevog puta ukoliko je takav put moguć. Kriteriji za ispitivanje postoji li skakačev put

  Metoda prima veličinu ploče n x m , te početne koordinate skakača na ploči, a vraća putanju skakača označenu brojevima 1

- `sahovska (self,n,m)` – pomoćna metoda kojom se generira ploča veličine n x m . Prima kao parameter n i m, a vraća šahovsku ploču u sljedećim oznakama : 0

- pomoćne metode koje ispituju mogućnost kretanja `if_L_ul(self,i,j)` – ispituje mogućnost kretanja gore lijevo (ostale mogućnosti su ur,dr,dl,rr,l

- `vrati(self,smjer,i,j)` – pomoćna metoda koja omogućava vraćanje na prethodni korak skakača, metoda prima kao parameter smjer kojim je skakač došao u trenutni položaj te treutne koordinate skakača, a vrać

- pomocna metoda koja nakon što je pronađen put skakača ispisuje kojim je putem prešao skakač.

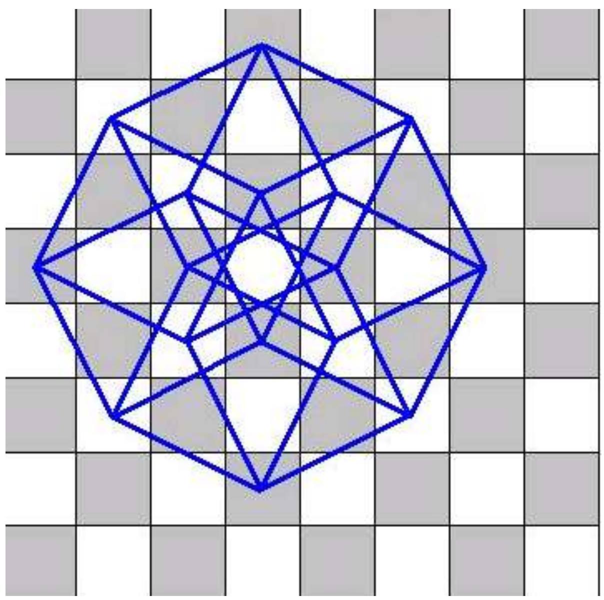
```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py cetrnaesti 6 6 1 1
['0', '1', '0', '1', '0', '1']
['1', '0', '1', '0', '1', '0']
['0', '1', '0', '1', '0', '1']
['1', '0', '1', '0', '1', '0']
['0', '1', '0', '1', '0', '1']
['1', '0', '1', '0', '1', '0']
NASO PUT
['UL', 'UL', 'UL', 'RL', 'RL', 'RL']
['LL', 'P', 'RR', 'RR', 'UR', 'RR']
['LR', 'UR', 'RR', 'LL', 'DL', 'DR']
['UL', 'UR', 'LR', 'RL', 'UR', 'DR']
['LR', 'DR', 'RR', 'LR', 'DL', 'DR']
['LL', 'LL', 'DR', 'LL', 'DL', 'DR']
4 5
[25, 8, 5, 2, 27, 10]
[6, 1, 26, 9, 34, 3]
[31, 24, 7, 4, 11, 28]
[18, 21, 30, 33, 14, 35]
[23, 32, 19, 16, 29, 12]
[20, 17, 22, 13, 36, 15]
root@ubuntu:/home/gizmo/teograf#
```

U primjeru prikazano je traženje skakačevog puta za šahovsku ploču veličine 6x6 s početnom kakača 1 1, prva matrica je prikaz šahovske ploče, druga matrica prikazuje kojim se putem kretao skakač u oznakama up
kretao skakač od početne pozicije koja je označena brojem 1 do trenutne pozici
označena brojem 36 (odnosno n*m)

Primjer ukoliko je zadana veličina ploče za koju nije moguće konstruirati skakačev put je sljedeći:

```
root@ubuntu:/home/gizmo/teograf# python tg_lab.py cetrnaesti 3 3 0 1
Ukoliko su m i n neparni ne postoji skakacev put 3 3
root@ubuntu:/home/gizmo/teograf# python tg_lab.py cetrnaesti 4 4 0 0
Nije moguce konstruirati skakacev put za 4xn - m= 4
root@ubuntu:/home/gizmo/teograf# python tg_lab.py cetrnaesti 3 4 0 0
Nije moguce konstruirati skakacev put za 4xn - m= 4
root@ubuntu:/home/gizmo/teograf# 
```

Izvršavanje primjera za ploču veličine 8 x 8 vremenski i memorijski je vrlo zahtjevno, te se ručava ispitivati za ploče veličine 6 x 6 ili manje.



Ukoliko kocku 4 prikažemo u ravnini te projiciramo na šahovsku ploču, dobijemo “mini” put skakača koji je duljine 16. Put skakača je sljedeći:

Svi labosi pozivaju se na sljedeći način

```
$ python tg_lab.py <broj_labosa> <nacin> <ime_datoteke ili ime grafa><opcionalni parametri>
```

pri čemu je broj_labosa napisan slovima (prvi, drugi,..), nacin označava način zadavanja grafa (ls- lista susjedstva, ms- matrica susjedstva, mi- matrica incidencije, pr- gneriranje), zatim slijedi ime datoteke ili ime grafa koji se generira, ukoliko je graf jedan od "gotovih" primjera zadaju se opcionlni parametric (npr. broj vrhova)

-
    - `python tg_lab.py prvi ls prvi_ls 0 0`
-
    - `python tg_lab.py prvi ms prvi_ms 0 0`
-
    - `python tg_lab.py prvi mi prvi_mi 0 0`
-
    - `python tg_lab.py prvi ls prvi_ls ms`
-
    - `python tg_lab.py prvi ls prvi_ls obrisi brid 1,2`
-
    - `python tg_lab.py prvi ls prvi_ls obrisi vrh 1`

-
    - `python tg_lab.py drugi pr potpuni 5`
-
    - `python tg_lab.py drugi pr bipartitni 5 2`
-
    - `python tg_lab.py drugi pr ciklus 3`
- **kotač**
    - `python tg_lab.py drugi pr kotac 7`
-
    - `python tg_lab.py drugi pr kocka 4`
-
    - `python tg_lab.py drugi pr petersen`

TREĆ

- `python tg_lab.py treci dat most`

ČETVRTI LABOS

- `python tg_lab.py cetvrti dat dijkstra`

- `python tg_lab.py peti dat tsp`

ŠESTI LABOS

- `python tg_lab.py sedmi pr kocka 3`

- `python tg_lab.py sedmi dat bridno`

- `python tg_lab.py osmi pr kocka 4`

- `python tg_lab.py deveti pr Petersen`
- `python tg_lab.py deveti pr potpuni 4`

- `python tg_lab.py deseti pr potpuni 4`
- `python tg_lab.py deseti dat bridno`

- `python tg_lab.py jedanaesti dat usmjereni`

- `python tg_lab.py dvanaesti pr bipartitni 2 4`

KORAK VIŠ

- `python tg_lab.py trinaesti dat bridno`

KORAK VIŠE(2)

- `python tg_lab.py cetrnaesti 6 6 0 1`