Naslov Skupina 19: Graffiti conjecture 232

Urban Merhar, Martin Kokošinek

1 Navodilo

Računalniško generirana domneva trdi: Če je G enostaven povezan graf, potem

$$2\gamma_t(G) \geq rad(G) + ecc(B).$$

Preveri domnevo na različne načine za male in velike grafe. Z uporabo populacijske metahevristike, preveri domnevo v upanju, da jo ovržeš.

Nekaj pripomb:

1. $ecc(v)$ je *ekscentričnost* od vozlišča v. Ekscentričnost od v je razdalja do najbolj oddaljenega vozlišča od vozlišča v, i.e., $max\{d(v,u) : u$ je vozlišče na grafu $\}$.
2. $rad(G)$ je *radij* grafa, t.j., minimum vseh ekscentričnosti vozlišč grafa G.
3. B je *obrobje* grafa G, t.j., množica vozlišč z maksimalno ekscentričnostjo.
4. $ecc(S)$ je *ekscentričnost* množice vozlišč S. Definirana je kot: Naj bo S podmnožica množice vozlišč V. Razdalja med vozliščem v in množico S, definirajmo kot razdaljo od v do najbližjega volišča v S. $ecc(S)$ je maksimum razadalj od vozlišča v $V \backslash S$ do množice S.

2 Kratek opis

Computer generated conjectures so računalniško ustvarjene domneve. $Graffiti$ je računalniški program, ki generira te matematične domneve oziroma odprte probleme. Računalniši program $Graffiti$ je ustvaril *Siemion Fajtlowicz*.

V najinem projektu pri predmetu Finančni praktikum si bova ogledala $Graffiti$ *conjecture* 232, ki jo bova testirala za majhne in velike grafe v upanju, da najdeva protiprimer. Ideja je, da enačbo zapiševa v programskem jeziku *Sage* in generirava naključne grafe. Na vsakem od teh grafov pa predpostavko testirava.

Že vgrajene funkcije, ki jih bova uporabila v programu:

1. $dominating\_set(total = True, value\_only = True)$ vrne najmanjšo dominirajočo množico na grafu G.
2. $radius()$ vrne radij grafa G.
3. $eccentricity()$ vrne ekscentričnost vozlišča v.
4. $periphery()$ vrne množico vozlišč iz obrobja grafa G.

2.1 Razlaga pojmov

- Dominirajoča Množica D: D je množica, kjer je vsako vozlišče iz $G \backslash D$ sosed nekega vozlišča iz D.
- Totalno Dominirajoča množica (TDM): Dominirajoči množici D dodamo pogoj, da so tudi vozlišča dominirajoče množice D sosedi vozlišč iz D.
- Totalno Dominirajoče Število (TDŠ): Moč totalno dominirajoče množice grafa G.
- $\gamma_t(G)$ je TDŠ grafa G.

2.1.1 Populacijska metahevristika

Hevristika (iz Grščine: 'najdem, odkrijem'): V računalništvu in matematični optimizaciji je visokonivojski način reševanja problemov, ko so klasični postopki prepočasni oziroma, ko klasične metode ne vrnejo točnih rezultatov. V zameno za polnost, optimalnost, natančnost, raje pridobimo na časovni zahtevnosti.
Meta-hevristika (meta iz Grščine: 'za, onstran') oziroma v prevodu Izčrpna-hevristika: Metahevristika vzame množico rešitev, ki je prevelika za analizo in s pomočjo določenih predpostavk glede optimizacije vrne zadovoljivo rešitev. Ta ni nujno globalno optimalna.
Populacijska metahevristika: Ohranjamo večje število kandidatov za rešitev in jih izboljšujemo s pomočjo populacijsih karakteristik. Primer je particle swarm optimization (PSO).

3 Plan dela

Zapisati učinkovit algoritem, ki bo za vsak generiran graf preverila lastnosti grafa in posledično domnevo. Za grafe, kjer domneva ne bi držala pa nam izpiše graf in vrne vrednosti lastnosti, ki so potrebovane v domnevi.

4 Potek dela

Za preverjanje dane domneve sva napisala program, ki jo testira. Samo domnevo pa sva nekoliko obrnila na način, da lahko učinkovito gledava tudi njeno razliko. To sva storila z namenom da opazujeva kateri grafi so bližje temu, da bi bili lahko protiprimer domneve. Kot je vidno v spodnjem programu sva tako na grafih preverjala $2\gamma_t(G) - rad(G) - ecc(B) \geq 0$. Na ta način sva s pomočjo razlike $2\gamma_t(G) - rad(G) - ecc(B)$ ocenjevala kako blizu pride nek graf k temu, da bi bil protiprimer.

Funkcija z imenom *predpostavka*() sprejme dan graf G in na njem testira domnevo ter vrne razliko $2\gamma_t(G) - rad(G) - ecc(B)$, če predpostavka velja, sicer pa pove, da predpostavka ne velja.

V spodnji kodi je naveden še primer in potek izračuna vrednosti domneve ter tudi način generiranja velikih in malih grafov.

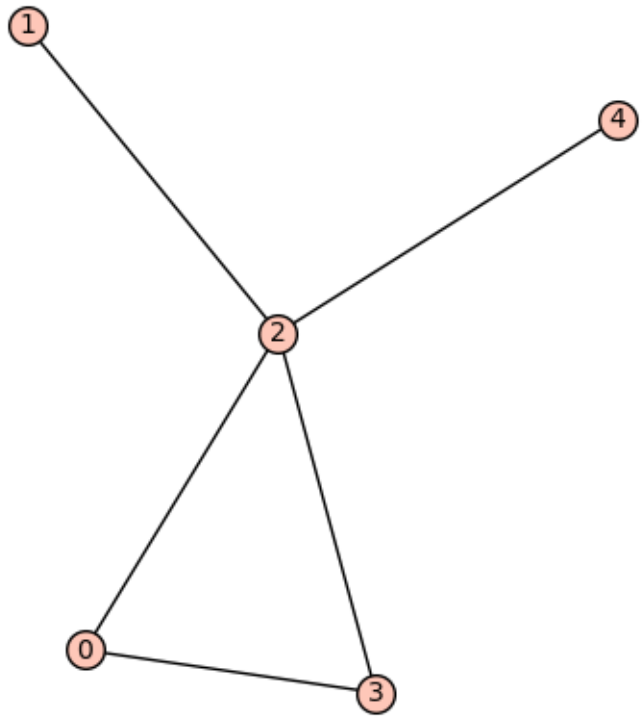
[1]:
```
# Projektna naloga pri predmetu Financni praktikum

# Naloga 19: Graffiti conjecture 232

def predpostavka(G):
    '''Funkcija, ki testira predpostavko Graffiti conjecture 232.'''
    if G.is_connected == False:
        return('Graf ni povezan!')
    B = set(G.periphery())
    d = G.distance_all_pairs()
    razdalja = set(d[u][v] for u in B for v in G if v not in B)
    if razdalja == set():
        razdlaja = 0
    # ce je razdalja slucajno prazna mnozica nastavimo njeno vrednost na nic
    razlika = 2 * G.dominating_set(total=True, value_only=True) - G.radius() -␣
 ↪min(razdalja) if razdalja else 0
    if razlika >= 0:
        print('Razlika med levo in desno stranjo predpostavke je:')
        print(razlika)
        return('Predpostavka velja!')
    else:
        return('Predpostavka NE velja!')
```

[2]:
```
# VELIKE GRAFE GENERIRAMO Z:
# graphs.RandomGNP(n, p) generira nakljucni preprost graf kjer je n stevilo vozlisc␣
 ↪in p verjetnost povezave med dvema vozliscema
```

```
G = graphs.RandomGNP(5, 0.4)
```

[3]:
```
G.show()
```

[3]:



[4]:
```
# Spodaj so loceno izracunane vse vrednosti v predpostavki
```

[5]:
```
# total=True da bo total dominating set
# value_only=True -> ker nas zanima samo vrednost in ne seznam vozlisc
G.dominating_set(total=True, value_only=True)
```

[5]: 2

[6]:
```
# radij grafa
G.radius()
```

[6]: 1

[7]:
```
# B je obrobje grafa
B = set(G.periphery())
d = G.distance_all_pairs()
razdalja = set(d[u][v] for u in B for v in G if v not in B)
# razdalja je ekscentricnost obrobja grafa B
razdalja
```

[7]: {1}

[8]:
```
# Preizkusimo funkcijo na zgornjem grafu:
predpostavka(G)
```

```
Razlika med levo in desno stranjo predpostavke je:
2
```

[8]:
```
'Predpostavka velja!'
```

[9]:
```
# GENERIRANJE MAJHNIH GRAFOV
# ker grafe generiras z nauty_geng so vsi preprosti (tako pise v dokumentaciji)
seznam = [G for G in graphs.nauty_geng('3 -c')] # povezani grafi na "n" vozliscih,␣
↪"-c" pomeni da so grafi povezani
```

[10]:
```
protiprimeri = []
for graf in seznam:
    if predpostavka(graf) == 'Predpostavka NE velja!':
        graf.show()
        protiprimeri.append(graf)
```

```
Razlika med levo in desno stranjo predpostavke je:
2
Razlika med levo in desno stranjo predpostavke je:
0
```

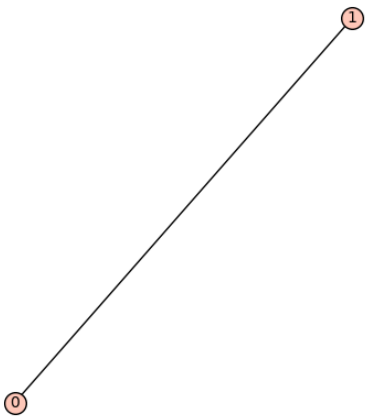
[11]:
```
# Ali obstaja protiprimer?
protiprimeri
```

[11]:
```
[]
```

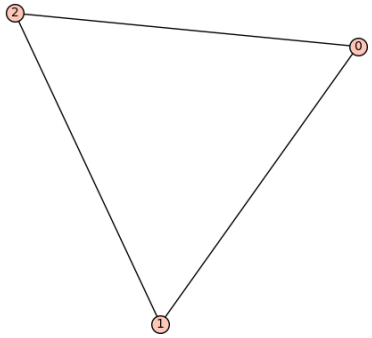
5 Majhni grafi

Majhne grafe sva v *sagu* generirala s pomočjo funkcije $graphs.nauty\_geng('v\ -c')$. Kjer je v število vozlišč grafa, $-c$ pa pomeni, da so grafi povezani. Na dovolj majhnih grafih sva generirala vse možne enostavne in povezane grafe ter na njih testirala predpostavko. Prav vsi grafi so ji ustrezali, zato sva se odločila podrobneje pogledati grafe, kjer je bila razlika med levo in desno stranjo predpostavke najmanjša.

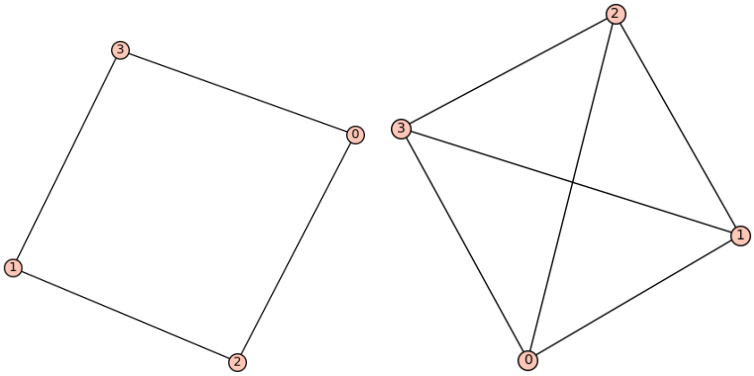
Graf na dveh vozliščih. Razlika v predpostavki je $2\gamma_t(G) - rad(G) - ecc(B) = 0$.



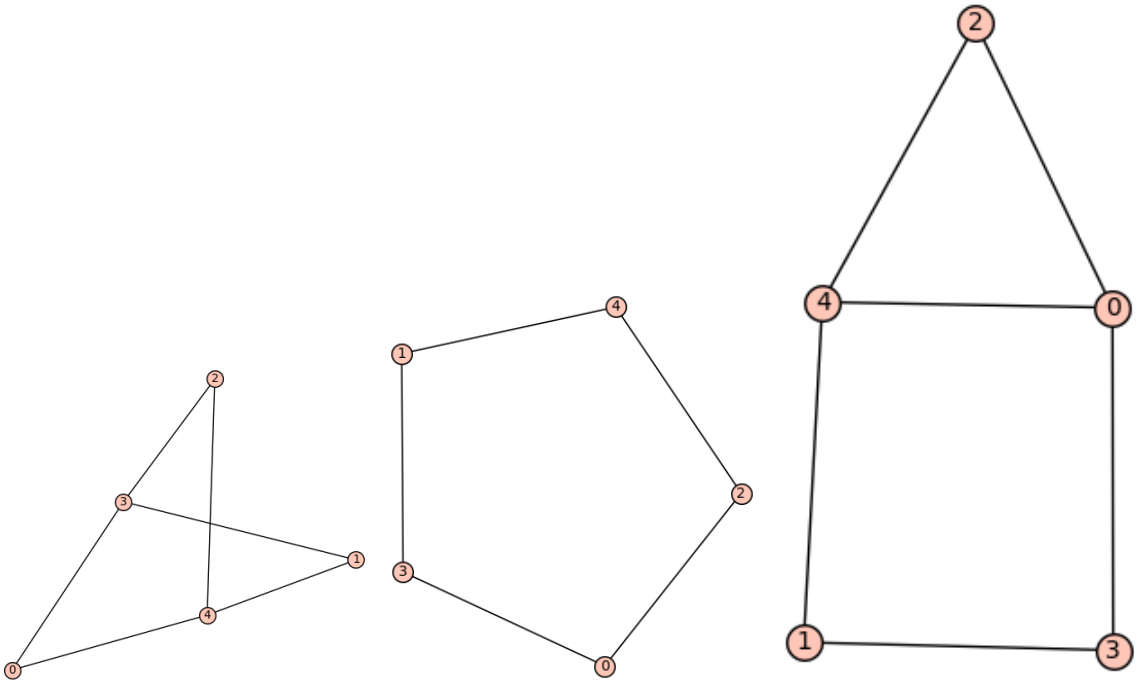
Graf na treh vozliščih. Razlika je 0.

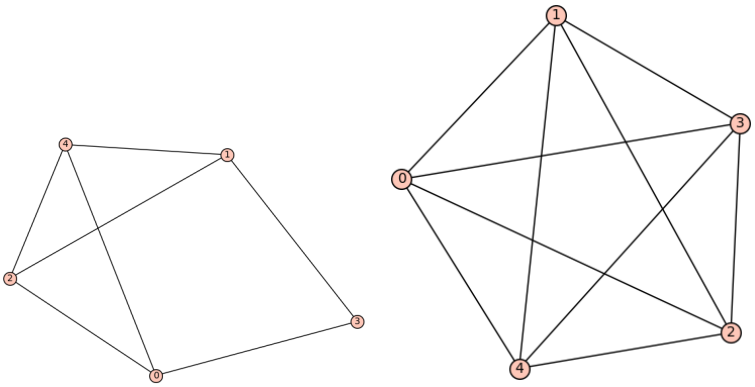


Grafi na štirih vozliščih imajo ponovno razliko predpostavke enako 0. Izmed 6 generiranih grafov imamo 2 z razliko predpostavke 0.

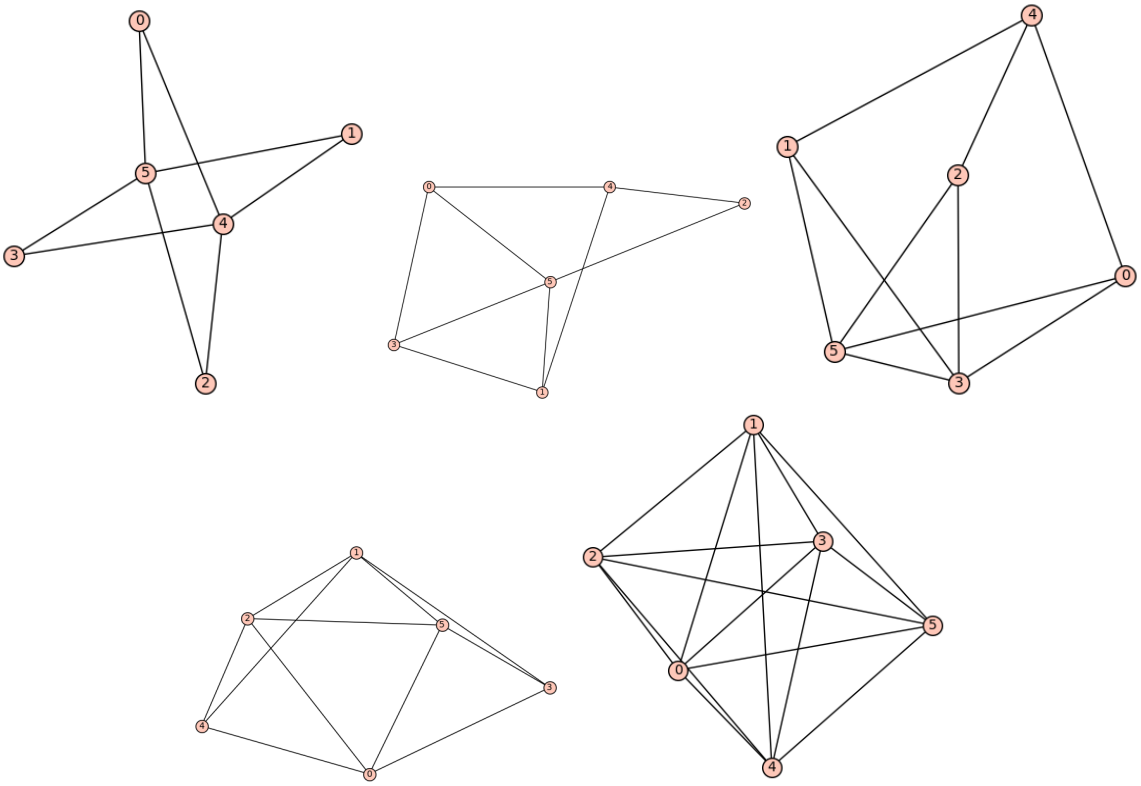


Pri grafih generiranih na petih vozliščih dobiš več grafov katerih razlika predpostavke je 0. Generiranih je skupno 21 grafov na pet vozliščih, od katerih jih ima 5 razliko predpostavke 0. Program testira vse te grafe v manj kot 0.07 sekunde.

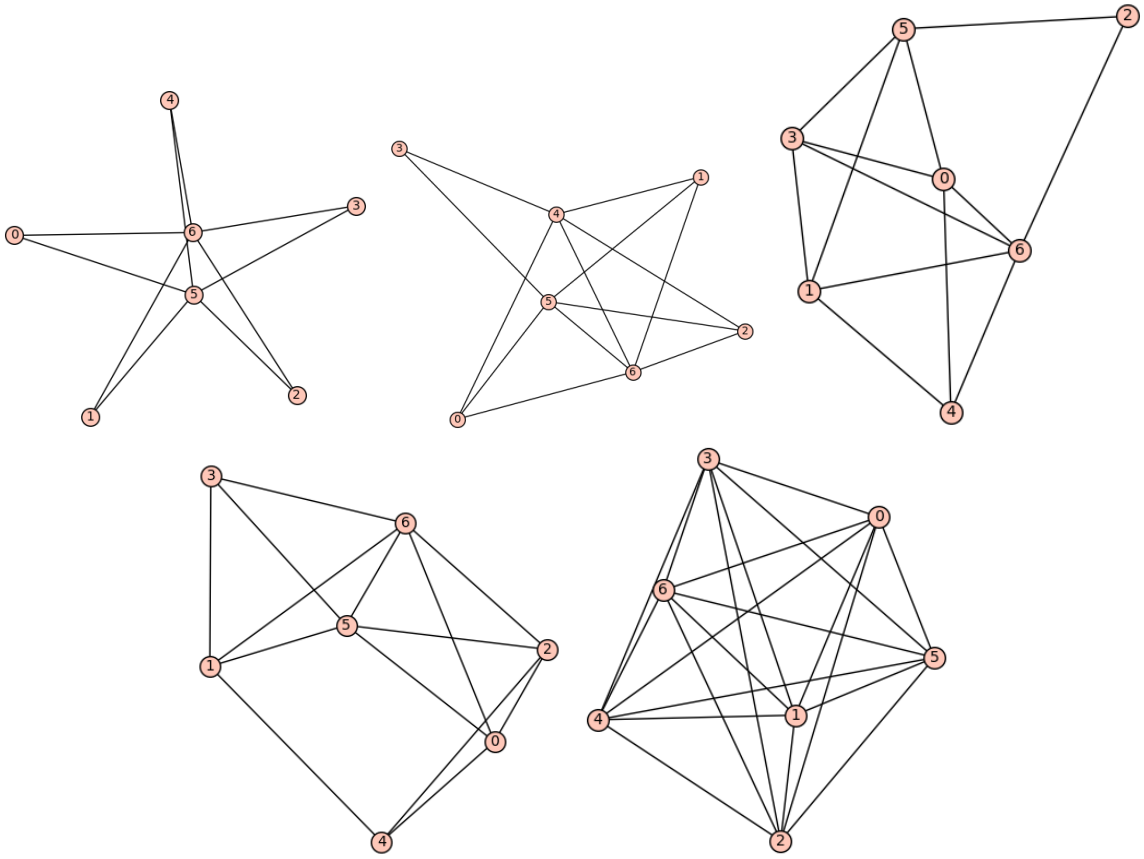




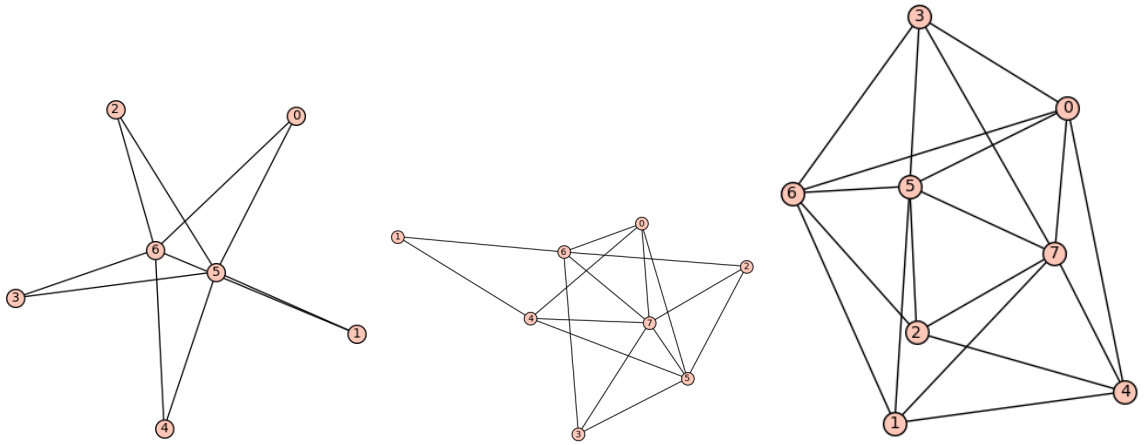
Pri grafih na šest vozliščih je razlika ponovno 0. Od skupno 112 vseh možnih grafov na šest vozliščih ima razliko 0 natanko 28 grafov. Program testira vse šestvozliščne grafe v približno 0.3 sekunde. Spodaj je prikazanih par primerov.

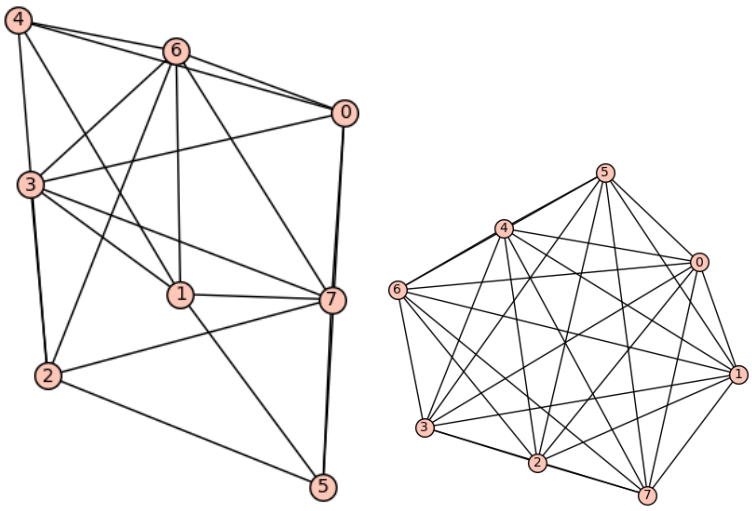


Za grafe na sedem vozliščih je minimalna razlika predpostavke ponovno 0. Sedaj je možno generirati 853 različnih grafov, razliko 0 pa jih ima 223. Program testira to predpostavko na vseh generiranih grafih na sedmih vozliščih v približno 2.5 sekunde. Ponovno si poglejmo nekaj primerov teh grafov.



Grafi na osmih vozliščih imajo razliko leve in desne strani ponovno enako 0. Zdaj generiramo že 11117 različnih grafov in za njihovo testiranje porabimo že kar približno 55 sekund. Grafov z osmimi vozlišči, kjer je razlika predpostavke enaka 0 je 3374.





Za grafe na devetih vozliščih se konča naše preizkušanje majhnih grafov, ker na tej točki računalnik ni več sposoben generirati vseh preprostih povezanih grafov na devetih vozliščih preko generatorja $graphs.nauty\_geng('v\ -c')$. Portal *CoCalc* sam terminira generator, če je ta zagnan.

Zaključimo lahko, da dana predpostavka *Graffiti conjecture* 232 vsekakor velja na majhnih grafih z osem ali manj vozlišči.

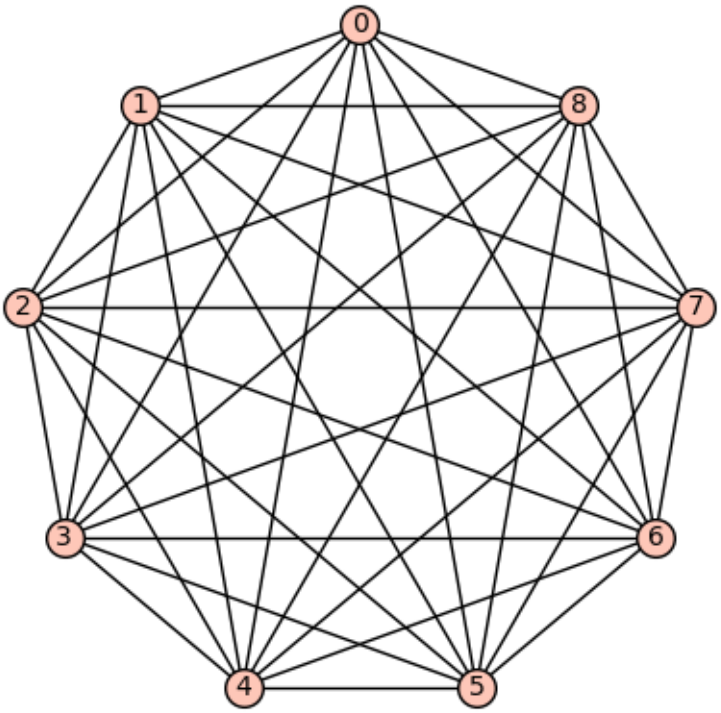
Iz vseh preverjenih majhnih grafov na do osem vozlišč pa lahko opazimo več stvari. Kot prvič, je bila minimalna razlika v predpostavki ne glede na število vozlišč enaka 0. Tudi z izjemo grafa na dveh vozliščih, je bila razlika predpostavke enaka 0, ko je imelo vsako vozlišče najmanj 2 povezavi. Opazimo pa lahko še, da je bil med grafi, kjer je bila razlika predpostavke enaka 0 vedno graf, kjer so bila vsa vozlišča povezana med seboj.

Glede na zadnjo opazko si poglejmo še kakšen graf kjer so vsa vozlišča povezana med sabo in poglejmo ali je razlika v predpostavki ponovno 0. Te grafe ustvarimo kar s pomočjo ugrajenega generatorja $graphs.RandomGNP(n,\ p)$, ki generira naključni preprost graf, kjer je n število vozlišč in p verjetnost povezave med dvema vozliščema. Torej bo za preverjanje naše opazke verjetnost $p = 1$, da bodo vsa vozlišča povezana med sabo. Idejo preizkusimo na grafih z 9, 12, 49, 76, 100 in 500 vozlišči. Spodaj je testiranje navedeno kar s kodo.

[2]:
```
# VELIKE GRAFE GENERIRAMO Z:
# graphs.RandomGNP(n, p) generira nakljucni preprost graf kjer je n stevilo vozlisc␣
 ↪in p verjetnost povezave med dvema vozliscema
G = graphs.RandomGNP(9, 1)
```

[3]:
```
G.show()
```

[3]:



[4]:
```
# Preizkusimo funkcijo na zgornjem grafu:
predpostavka(G)
```

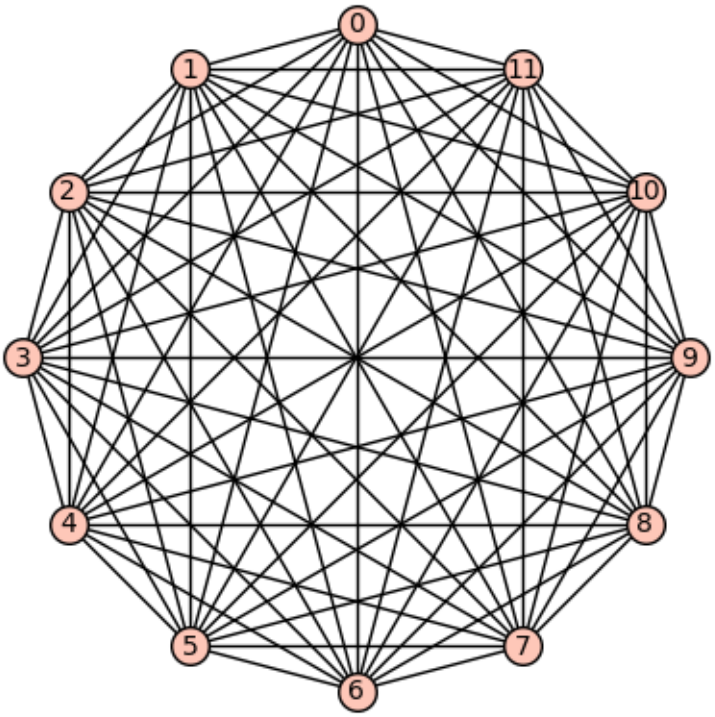
```
Razlika med levo in desno stranjo predpostavke je:
0
```

[4]:
```
'Predpostavka velja!'
```

[5]:
```
G = graphs.RandomGNP(12, 1)
```

[6]:
```
G.show()
```

[6]:



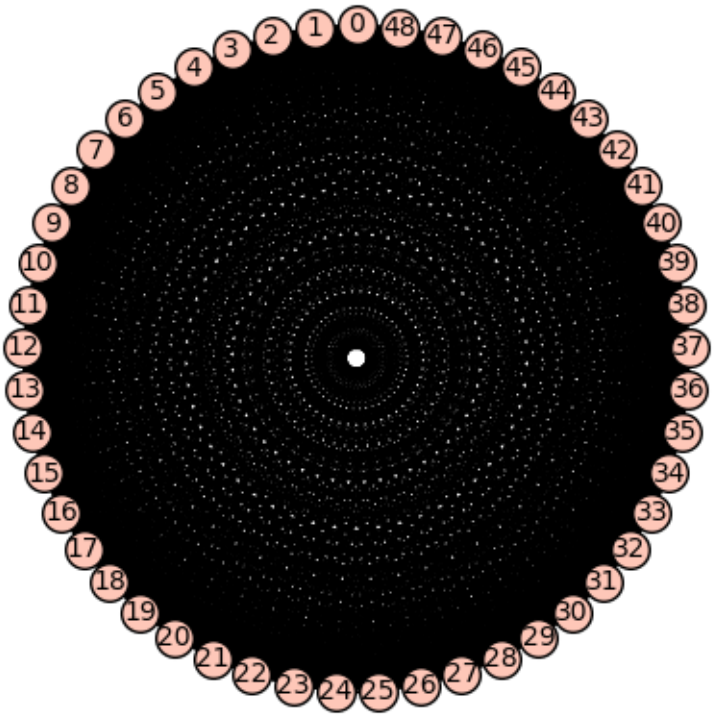
[7]:
```
# Preizkusimo funkcijo na zgornjem grafu:
predpostavka(G)
```

```
Razlika med levo in desno stranjo predpostavke je:
0
```

[7]:
```
'Predpostavka velja!'
```

[8]:
```
G = graphs.RandomGNP(49, 1)
G.show()
```

[8]:



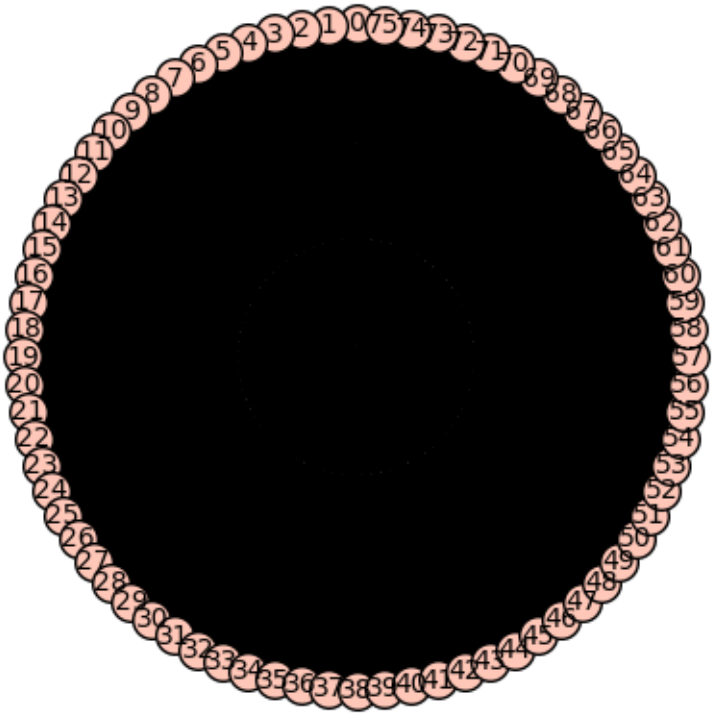
[9]:
```
# Preizkusimo funkcijo na zgornjem grafu:
predpostavka(G)
```

```
Razlika med levo in desno stranjo predpostavke je:
0
```

[9]:
```
'Predpostavka velja!'
```

[10]:
```
G = graphs.RandomGNP(76, 1)
G.show()
```

[10]:



[11]:
```
# Preizkusimo funkcijo na zgornjem grafu:
predpostavka(G)
```

```
Razlika med levo in desno stranjo predpostavke je:
0
```

[11]: 'Predpostavka velja!'

[12]:
```
G = graphs.RandomGNP(100, 1)
predpostavka(G)
```

```
Razlika med levo in desno stranjo predpostavke je:
0
```

[12]: 'Predpostavka velja!'

[13]:
```
G = graphs.RandomGNP(500, 1)
predpostavka(G)
```

```
Razlika med levo in desno stranjo predpostavke je:
0
```

[13]: 'Predpostavka velja!'

Kot so pokazali primeri na grafih z 2, 3, 4, 5, 6, 7, 8, 9, 12, 49, 76, 100 ali 500 vozlišči lahko vidimo, da ima razlika predpostavke na grafih, kjer so vsa vozlišča povezana med sabo vrednost 0.