Kompleksne mreže – primjeri zadataka za ZI

[File podsjetnik\_plus\_auditorne.pdf]

Napomene:

Završni ispit nosi 80 bodova. Ovaj ogledni primjer nosi više bodova, jer je dano više zadataka za vježbu nego što se očekuje u završnom ispitu.

Studente se savjetuje da samostalno pokušaju riješiti sve praktične zadatke programiranjem u Jupyter Notebooku.

1.

Pomoću računala:

a) Iz zadane liste veza konstruirajte usmjerenu mrežu i prikažite ju. (2) ✅

1 2 {}

1 3 {}

2 1 {}

2 3 {}

3 1 {}

3 2 {}

3 4 {}

3 5 {}

5 6 {}

6 7 {}

7 5 {}

import networkx as nx

import matplotlib.pyplot as plt

edges = [

(1, 2, {}),

(1, 3, {}),

(2, 1, {}),

(2, 3, {}),

(3, 1, {}),

(3, 2, {}),

(3, 4, {}),

(3, 5, {}),

(5, 6, {}),

(6, 7, {}),

(7, 5, {})

]

D = nx.DiGraph()

D.add\_edges\_from(edges)

plt.title('Usmjerena mreža')

nx.draw\_networkx(D, pos=nx.spring\_layout(D), with\_labels=True)

plt.show()

b) Ispišite ulazni stupanj svakog čvora i izračunajte prosječni ulazni stupanj čvora.(2) ✅

in\_degrees = D.in\_degree()

average\_in\_degree = sum(dict(in\_degrees).values()) / len(D.nodes)

print("Ulazni stupnjevi čvorova mreže D:", in\_degrees)

print("Prosječni ulazni stupanj čvora:", average\_in\_degree)

c) Izračunajte gustoću mreže. (1) ✅

density = nx.density(D)

print("Gustoća mreže D:", density)

d) Pronađite klike. Čvorovima koji tvore klike od točno 3 člana postavite atribut 'boja' u zelenu('green'), a ostalima postavite atribut 'boja' u crvenu('red'). Dobivenu mrežu s pobojanim čvorovima prikažite. (3) ✅

G = nx.Graph(D) # Convert the directed graph to an undirected graph

G\_cliques = nx.find\_cliques(G)

clique\_colors = ['red', 'green']

for clique in G\_cliques:

if len(clique) == 3:

nx.set\_node\_attributes(G.subgraph(clique), {n: 'green' for n in clique}, "color")

else:

nx.set\_node\_attributes(G.subgraph(clique), {n: 'red' for n in clique}, "color")

plt.title('Klike s više od 2 člana')

nx.draw\_networkx(G, node\_color = list(nx.get\_node\_attributes(G, "color").values()), pos = nx.spring\_layout(G, seed = 7))

e) Izdvojite podmrežu koju čine čvorovi 3, 4, 5 i prikažite ju . Spremite listu veza dobivene podmreže na disk pod imenom 'podmreza\_zad\_1.edgelist' u trenutni direktorij. (2) ✅

subgraph = G.subgraph([3, 4, 5])

nx.write\_edgelist(subgraph, 'podmreza\_zad\_1.edgelist')

2.

Pomoću računala:

a) Učitajte mrežu interakcije proteina iz datoteke 'protein\_interaction.edgelist'.(1)

b) Provjerite povezanost mreže. (1)

c) Izračunajte asortativnost. (1)

d) Izračunajte prosječan najkraći put. (1)

e) Izračunajte dijametar. (1)

f) Izračunajte koeficijent klasteriranja čvorova. (1)

g) Prikažite distribuciju stupnja čvora. (2)

h) Prikažite distribuciju bliskosti. (2)

i) Prikažite distribuciju međupoloženosti. (2)

j) Izračunajte heterogenost (2)

k) Odredite i prikažite 2-jezgru i 2-ljusku zadane mreže.(3)

l) Nacrtajte graf usporedbe veličine najveće komponente u slučaju kvara i napada za uklonjenih 0, 10 i 20 čvorova. (5)

# a) Učitajte mrežu interakcije proteina iz datoteke 'protein\_interaction.edgelist'. ✅

G\_protein = nx.read\_edgelist('protein\_interaction.edgelist')

# b) Provjerite povezanost mreže. ✅

print(f"Mreža je povezana? {nx.is\_connected(G\_protein)}")

# c) Izračunajte asortativnost. ✅

print(f"Asortativnost proteina : {nx.degree\_assortativity\_coefficient(G\_protein)}")

# d) Izračunajte prosječan najkraći put. ✅

print(f"Prosječan najkraći put proteini : {nx.average\_shortest\_path\_length(G\_protein)}")

# e) Izračunajte dijametar. ✅

print(f"Dijametar proteini : {nx.diameter(G\_protein)}")

# f) Izračunajte koeficijent klasteriranja čvorova. ✅

print(f"Koeficijent klasteriranja čvorova proteini : {nx.average\_clustering(G\_protein)}")

# g) Prikažite distribuciju stupnja čvora. ✅

def degree\_calc(G):

degrees = [val for (node, val) in G.degree()]

avg\_degree = np.mean(degrees)

med\_degree = np.median(degrees)

return degrees, avg\_degree, med\_degree

def degree\_distribution\_plot(degree\_list, avg\_degree, med\_degree, cumulative, title):

plt.hist(degree\_list,label='Distribucija stupnja', cumulative = cumulative)

plt.axvline(avg\_degree,color='r',linestyle='dashed',label='Prosječni stupanj')

plt.axvline(med\_degree,color='g',linestyle='dashed',label='Medijan stupanj')

plt.legend()

plt.ylabel('Postotak čvorova')

plt.xlabel('Iznos stupnja')

plt.title(title)

d\_list\_protein, avg\_d\_protein, med\_d\_protein = degree\_calc(G\_protein)

degree\_distribution\_plot(d\_list\_protein, avg\_d\_protein, med\_d\_protein, False, 'Proteini')

# h) Prikažite distribuciju bliskosti. ✅

def closeness\_calc(G):

closeness = [val for (node, val) in nx.closeness\_centrality(G).items()]

avg\_closeness = np.mean(closeness)

med\_closeness = np.median(closeness)

return closeness, avg\_closeness, med\_closeness

def closeness\_distribution\_plot(closeness\_list, avg\_closeness, med\_closeness, cumulative, title):

plt.hist(closeness\_list,label='Distribucija bliskosti', cumulative = cumulative)

plt.axvline(avg\_closeness, color='r' ,linestyle='dashed', label = "Prosječna bliskost")

plt.axvline(med\_closeness, color='g' ,linestyle='dashed', label = "Medijan bliskosti")

plt.legend()

plt.ylabel('Postotak čvorova')

plt.xlabel('Iznost bliskosti')

plt.title(title)

c\_list\_protein, avg\_c\_protein, med\_c\_protein = closeness\_calc(G\_protein)

closeness\_distribution\_plot(c\_list\_protein, avg\_c\_protein, med\_c\_protein, False, 'Proteini')

# i) Prikažite distribuciju međupoloženosti. ✅

def calc\_betweenness(G):

betweenness = [val for (node, val) in nx.betweenness\_centrality(G).items()]

avg\_betweenness = np.mean(betweenness)

med\_betweenness = np.median(betweenness)

return betweenness, avg\_betweenness, med\_betweenness

def betweenness\_distribution\_plot(betweenness\_list, avg\_betweenness, med\_betweenness, cumulative, title):

plt.hist(betweenness\_list,label='Distribucija međupoloženosti', cumulative = cumulative)

plt.axvline(avg\_betweenness,color='r',linestyle='dashed',label='Prosječna međupoloženost')

plt.axvline(med\_betweenness,color='g',linestyle='dashed',label='Medijan međupoloženosti')

plt.legend()

plt.ylabel('Postotak čvorova')

plt.xlabel('Iznos međupoloženosti')

plt.title(title)

b\_list\_protein, avg\_b\_protein, med\_b\_protein = calc\_betweenness(G\_protein)

betweenness\_distribution\_plot(b\_list\_protein, avg\_b\_protein, med\_b\_protein, False, 'Proteini')

# j) Izračunajte heterogenost ✅

def calc\_heterogeneity(g):

average\_squared\_degree = sum([degree\*\*2 for (node, degree) in g.degree()])/g.number\_of\_nodes()

average\_degree = sum(degree for (node, degree) in g.degree())/g.number\_of\_nodes()

heterogeneity = average\_squared\_degree/(average\_degree\*\*2)

return heterogeneity

print(f"Heterogenost proteini: {calc\_heterogeneity(G\_protein)}")

# k) Odredite i prikažite 2-jezgru i 2-ljusku zadane mreže.

k\_core = nx.k\_core(G\_protein, k=2)

k\_shell = nx.k\_shell(G\_protein, k=2)

nx.draw\_networkx(k\_core)

# l) Nacrtajte graf usporedbe veličine najveće komponente u slučaju kvara i napada za uklonjenih 0, 10 i 20 čvorova.

largest\_components\_failure = [len(c) for c in nx.connected\_components(G\_protein)]

largest\_components\_attack = [len(c) for c in nx.connected\_components(G\_protein)]

plt.plot([0, 10, 20], largest\_components\_failure, label='Failure')

plt.plot([0, 10, 20], largest\_components\_attack, label='Attack')

plt.legend()

3.

Pomoću računala:

a) Implementirajte funkciju za generiranje mreže Gilbertovog modelom.(3)

def gilbert\_model\_graph(n, p):

return nx.gnp\_random\_graph(n, p, directed=False)

b) Implementirajte funkciju za generiranje mreže Erdos-Reny modelom.(3) ✅  
  
def erdos\_renyi\_model\_graph(n, p):

return nx.erdos\_renyi\_graph(n, p)

c) Implementirajte funkciju za generiranje mreže Watts-Strogatz modelom.(5) ✅  
  
def watts\_strogatz\_model\_graph(n, k, p):

return nx.watts\_strogatz\_graph(n, k, p)

d) Implementirajte funkciju za generiranje mreže Barabasi-Albertov modelom.(5) ✅

def barabasi\_albert\_model\_graph(n, m):

return nx.barabasi\_albert\_graph(n, m)

e) Generirajte mrežu Erdos-Reny modelom s 20 čvorova i 35% vjerojatnošću stvaranja veze, koristite ugrađenu funkciju.(2) ✅  
  
G = erdos\_renyi\_model\_graph(20, 0.35)

f) Napišite funkciju početnog stanja mreže. Za početno stanje svakom čvoru nasumično dodijelite stanje odabirom slova od A do D. (2)  
  
import random

def initial\_state(graph):

for node in graph.nodes():

graph.nodes[node]['state'] = random.choice(['A', 'B', 'C', 'D'])

g) Napišite funkciju prijelaza. Za svaki čvor nasumično odaberite susjeda i postavite stanje trenutnog čvora na istu vrijednost kao stanje susjeda.(3)  
  
def state\_transition(graph):

for node in graph.nodes():

neighbor = random.choice(list(graph.neighbors(node)))

graph.nodes[node]['state'] = graph.nodes[neighbor]['state']

h) Koristeći datoteku simulation.py koja sadrži klasu Simulation, kreirajte simulaciju naziva 'model\_glasanja'. Koristite mrežu generiranu u e) i funkcije početnog stanja i prijelaza iz f) i g). Pokrenite 4 koraka simulacije te prikažite udio čvorova u svakom stanju u svakom koraku simulacije. (2)   
  
from simulation import Simulation

model\_glasanja = Simulation(G, initial\_state, state\_transition, name='model\_glasanja')

model\_glasanja.run(4)

model\_glasanja.plot()

[File sve\_preze.pdf]

Primjeri teorijskih pitanja:

1. Objasnite razliku između trokuta i triade. ✅
2. Objasnite koeficijent klasteriranja koristeći formulu ili svojim riječima. ✅
3. Kada proučavamo robusnost mreže, koja je razlika kvara i napada. ✅
4. Objasnite kosinusnu sličnost, formulom ili svojim riječima. ✅
5. Objasnite svojim riječima paradoks prijateljstva i dajte dodatni primjer. ✅

1.Triada - skup od tri čvora – kada su svi povezani -> trokut

2. Postotak parova susjeda promatranog čvora koji su međusobno

Povezani

Omjer broja trokuta koji uključuju čvor i maksimalan broj trokuta u

kojima može sudjelovati

ci = tau i / tau max (i) = tau (i) / (ki povrh 2), tau(i) broj trokuta koji ukljucuje i tau max(i) max broj trokuta koji ukljucuje i i njegove susjede k > 1

Mjerenje broj čvorova u najvećoj

komponenti u odnosu na

početnu mrežu

3. Kvar – slučajno uklanjanje

Napad – ciljano uklanjanje (prvo hubovi)

Otpornost na kvar i ranjivost na napad

4. Kosinusna sličnost je mjera koja se koristi za određivanje sličnosti između vektora. Može se izračunati kao kosinus kuta između vektora. Što je veći kosinusni rezultat, to su vektori sličniji.

Cos (d1,d2) = d1 / || d1 || \* d2 / || d2 ||, ||d|🡪 normizacija (dugackim dokumentima smanjuje slicnost) normom vektora,

5.Paradoks prijateljstva je fenomen u socijalnim mrežama gdje se može dogoditi da bliži prijatelji imaju manje veza od daljih prijatelja. Na primjer, ako osoba A ima dva prijatelja B i C koji se ne poznaju, može se dogoditi da B i C imaju više veza s drugim ljudima nego što imaju s osobom A. Ovo je paradoks jer se očekivalo da će bliži prijatelji imati više veza međusobno.

Sources: p.62, p.146, p.32, p.118

[File podsjetnik\_plus\_auditorne.pdf]

1. Za danu mrežu izračunajte koeficijent klasteriranja za čvor 3. Chart, line chart

   Description automatically generated

8 je spojeno na 2, 2 je spojeno na 5,3,7, 7 je spojeno na 6,1,2,3, 3 je spojeno na 7,2,5, 5 je spojeno na 4,2,3

Koeficijent klasteriranja za čvor 3 može se izračunati koristeći formulu: C = (2\*broj trokuta koji uključuju čvor 3) / (broj mogućih trokuta koje bi mogao imati)

Prvo ćemo pronaći broj trokuta koji uključuju čvor 3:

Čvor 3 je spojen na čvorove 7, 2, i 5, što znači da postoji trokut s vrhovima 3, 7 i 2 te trokut s vrhovima 3, 2 i 5.

Sada ćemo pronaći broj mogućih trokuta koje bi čvor 3 mogao imati:

Čvor 3 ima ukupno 3 susjeda (7, 2, 5), što znači da bi mogao imati 3\*(3-1)/2 = 3 trokuta.

Sada možemo izračunati koeficijent klasteriranja za čvor 3: C = 2 / 3 = 2/3 ≈ 0.67

Dakle, koeficijent klasteriranja za čvor 3 je otprilike 0.67.

Sources: p.146, p.60, p.149, p.62

1. Isimulirajte zajednice pomoći networkX na proizvoljnom grafu. Detektirajte zajednice algoritmom. Objasite.

Da biste simulirali zajednice pomoću networkX na proizvoljnom grafu, možete koristiti algoritme detekcije zajednica poput Girvan-Newman algoritma ili Louvain algoritma. Ovi algoritmi mogu detektirati zajednice analizom strukture grafa kako bi identificirali grupe čvorova koji su gusto povezani unutar zajednice, ali slabo povezani između zajednica.

1. Iterativnom metodom izračunajte modularity score za n zajednica od 2 do 6 na karate glub grafu. Objasnite kako odabrati koji je broj zajednica? Zašto?

Modularity score se izračunava kako bi se odredio broj zajednica koje najbolje odgovaraju grafovima. Iterativnom metodom možete izračunati modularity score za različit broj zajednica (n) od 2 do 6 na karate klub grafu. Odabir broja zajednica može se temeljiti na maksimiziranju modularity score-a. Broj zajednica koji rezultira najvećim modularity score-om smatra se optimalnim brojem zajednica za graf.

1. Isimulirajte Beyesian learning na grafu koristeći jednostavni Bala Goyal 98 pristup s s dva A ili B stanja. Koristite barbell graf koji će biti povezan putem centralnog čvora. Detektirajte kojem stanju će konvergirati mreža. Objasnite koncept.  
     
   Bayesian learning na grafu može se simulirati koristeći jednostavni Bala Goyal 98 pristup s dva A ili B stanja na barbell grafu koji je povezan putem centralnog čvora. Detekcija kojem stanju će konvergirati mreža može se izvršiti praćenjem vjerojatnosti konvergencije na određeno stanje. Koncept se temelji na Bayesovom pristupu učenju putem zaključivanja na temelju prethodnog znanja i novih informacija.
2. Isimulirajte DeGroot širenje mišljena na random grafu od 10 ljudi. Izračunajte kada će sustav konvergirati. Objasnite koncept DeGroot učenja.   
     
   DeGroot širenje mišljenja na random grafu od 10 ljudi uključuje iterativno proces konvergencije mišljenja. Može se izračunati kada će sustav konvergirati promatranjem promjena u mišljenju svakog pojedinog čvora tijekom iteracija. Koncept DeGroot učenja se temelji na sociološkoj teoriji da ljudi mijenjaju mišljenje na temelju mišljenja drugih u mreži.
3. Učitajte karate glub graf. Primjenite k-clique community algoritam s proizvoljno odabranim parametrima. Objasnite konceptualno dobivene rezultate s obzirom na odabrane parametre.

Primjenom k-clique community algoritma na karate klub graf, možete detektirati k-clique zajednice unutar grafa. K-clique algoritam identificira grupiranje čvorova koji tvore kompletne podgrafove od k čvorova. Rezultati ovog algoritma mogu biti interpretirani s obzirom na odabrane parametre, poput broja čvorova k i ostalih parametara algoritma, kako bi se razumjelo kako su zajednice identificirane unutar grafa.