پروژه آمار و احتمال مهندسی **فاز ۱**

نیم‌سال اول ۱۴۰۱-۱۴۰۲ **انتشار جواب: بهمن ۱۷**

سوال ۱. سوال تئوری ۱

اگر جامعه را بتوان به خوشه های مجزا از افرادی با سلیقه ی یکسان افراز کرد، به طوری که افرادِ یک خوشه با افرادِ خوشه ی دیگر هیچ اشتراکی نداشته باشند، آنگاه ماتریس مجاورت این افراد به چه صورت خواهد بود؟

اولا که ماتریس متقارن است زیرا که روابط دوطرفه هستند. ثانیا به علت اینکه همه‌ی افراد یک خوشه با هم در ارتباط هستند و یال دارند، میتوان ادعا کرد که یک جایگشت سطری ستونی از ماتریس مجاورت وجود دارد به گونه‌ای روی قطر آن چندین ماتریس مربعی کاملا یک باشند و مابقی ماتریس صفر. مثل ماتریس جوردن ولی همه‌ی درایه‌های بلوک‌ها یک هستند و نه صفر.

سوال ۲. سوال تئوری ۲

اولا که ماتریس متقارن است زیرا که روابط دوطرفه هستند. ثانیا به علت اینکه همه‌ی افراد یک خوشه با هم در ارتباط هستند و یال دارند، میتوان ادعا کرد که یک جایگشت سطری ستونی از ماتریس مجاورت وجود دارد به گونه‌ای روی قطر آن چندین ماتریس مربعی کاملا یک باشند و مابقی ماتریس صفر. مثل ماتریس جوردن ولی همه‌ی درایه‌های بلوک‌ها یک هستند و نه صفر.

در این حالت می‌توان سطر و ستون‌های ماتریس را به گونه‌ای جابجا کرد که هر بخش متعلق به یک خوشه باشد و هر فرد یا با تمام افراد خوشه در ارتباط هست و یا کلا با آن در ارتباط نیست. پس شکل مثل زیر خواهد شد. (هر رنگ یک خوشه و سلیقه‌است و بخش‌های رنگی یک و غیرنگی صفر است.

سوال ۳. سوال تئوری ۳

فرض کنید M ژانر مختلف فیلم داریم. اگر دو نفر به ژانر ۲، ۱- i ام علاقه داشته باشند، به احتمال i با یکدیگر هم‌سلیقه‌اند، حال فرض کنید دو شخص خاص هر کدام به چند ژانر مختلف علاقه دارند. اگر فرض کنیم علاقه به pi ژانرهای مختلف از هم مستقلند، احتمال اینکه این دو نفر با یکدیگر هم سلیقه باشند چقدر است؟

چرا اگر دو نفر در جوامع خاص زیادی عضو باشند احتمال اینکه با هم در ارتباط باشند بیشتر می شود؟ آیا می توانید این مسئله را به صورت شهودی هم توجیه کنید؟ به نظر شما این مدل چه نقصی در مدل کردن رابطه‌ی افراد و گروه ها دارد؟ شکل ۳ را در این خصوص ببینید.

احتمال هم‌سلیقه بودن با فرض علاقه‌به ژانرهای مجموعه‌ی A :

$$۱ - \prod_{i \in A} (۱ - p_i)$$

زیرا هر چه در جوامع بیشتری باشند، تعداد جملات پای در فرمول بالا بیشتر می‌شود و چون هر جمله کمتر از یک است، پس در نهایت احتمال هم‌سلیقگی بیشتر می‌شود. تعداد ژانرها محدود است و هر چه تعداد بیشتری را دوست داشته باشید در ارتباط با افراد بیشتری که آن را دوست دارند قرار می‌گیرید و احتمال اینکه با فردی هم‌سلیقه شوید بیشتر خواهد بود بود. پس به طور شهودی نیز تطابق دارد. - نقص‌ها - تعلق به یک گروه صفر و یکی است و وزن ندارد درصورتی که وزنی بهتر توصیف می‌کند - حالات بین خوشه‌ای به طور بهینه مدل نشده‌اند، زیرا ممکن است

یک ترکیب از تعدادی خوشه خودش یک خوشه‌ی دارای اصالت باشد. - ممکن است خوشه‌ها بر اساس ژانر فیلم نباشند و معیارهای دیگری مثل کارگردان و بازیگران و کمپانی و ... نیز در خوشه‌ها باشند.

سوال ۴. سوال تئوری ۴

$$P_{uv} = ۱ - \prod_{i=۱}^{c} \exp\left(\_ F_{ui}F_{vi}\right) = ۱ - \exp\left(-\sum_{i=۱}^{c}\left(F_{ui}F_{vi}\right)\right)$$

سوال ۵. سوال تئوری ۵

رابطه‌ها رو می‌نویسیم و بدست می‌آوریم:

$$P\left(A|F\right) = \prod_{u,v=۱}^{n}\left[A_{uv}P_{uv} + \left(۱ - A_{uv}\right)\left(۱ - P_{uv}\right)\right]$$

$$=> l\left(F\right) = \sum_{u,v=۱}^{n}\log\left(A_{uv}P_{uv} + \left(۱ - A_{ar}\right)\left(۱ - P_{uv}\right)\right)$$

$$= \sum_{u,v=۱}^{n}\log\left(۲A_{uv}P_{uv} + ۱ - A_{uv} - P_{uv}\right)$$

$$= \sum_{u,v=۱}^{n}\log\left(۲A_{uv}\left(۱ - \exp\left(-\sum_{i=۱}^{c}F_{ui}F_{vi}\right)\right) + A_{uv} + \exp\left(-\sum_{i=۱}^{c}F_{ui}F_{vi}\right)\right)$$

$$= \sum_{u,v=۱}^{n}\log\left(۳A_{uv} + \left(۱ - ۲A_{uv}\right)\exp\left(-\sum_{i=۱}^{c}F_{ui}F_{vi}\right)\right)$$

سوال ۶. سوال تئوری ۶

این روش در اصل همان روش descend gradient است که کاربرد زیادی در محاسبات عددی دارد. در این روش از نقطه یا نقاطی روی نمودار شروع می‌کنیم و در هر گام مقدار اندکی در راستای گرادیان در آن نقطه جابجا میشویم و به نقطه‌ي جدید می‌رویم. اگر این کار را با تعداد گام و طول گام مناسب انجام بدهیم انتظار داریم که به نقطه‌ی اکسترمم موضعی برسیم. اما برای این کار شرایطی لازم است. اولا اینکه تابع پوسته باشد که تابع ما جمع چند لگاریتم پیوسته است پس پیوسته است. همچنین نوسانات شدید در تابع باعث می‌شوند که روش به خوبی کار نکند. در این تابع ما نوسانات شدید هم وجود ندارد. هم‌چنین نکته‌ی دیگر اینست که این روش به ما اکسترمم موضعی را می‌دهد و گلوبال و به شرطی که فقط یک اکسترمم داشته داشته باشیم قطعا به اکسترمم گلوبال می‌رسیم. همچنین نواحی با گرادیان صفر نباید وجود داشته باشند تا الگوریتم متوقف نشود و یا در دور نیافتد.

سوال ۷. سوال تئوری ۷

$$\nabla F_{w,i} =$$
$$۲\sum_{v=۱}^{n} \frac{(۱ - ۲A_{wv})\left(-\sum_{i=۱}^{c} F_{vi}\right)\exp\left(-\sum_{i=۱}^{c} F_{wi}F_{vi}\right)}{۳A_{wv} + (۱ - ۲A_{\omega v})\exp\left(-\sum_{i=۱}^{c} F_{wi}F_{vi}\right)}$$

سوال ۸. سوال شبیه‌سازی ۱

```
def log_likelihood(F, A):
    N = A.shape[0]
    C = F.shape[1]
    log_likelihood = 0
    for u in range(N):
        for v in range(N):
            summ=0
            for i in range(C):
                summ += F[u][i]*F[v][i]
            log_likelihood += numpy.log(3*A[u][v]+(1 2*A[u][v])*
            numpy.exp( summ))
    return log_likelihood

def gradient(F, A, i):
    gradient = list()
    N = A.shape[0]
    C = F.shape[1]
    for w in range(N):
        element = 0
        for v in range(N):
            summ = 0
            summP = 0
            for i in range(C):
                summp += F[v][i]
                summ += F[v][i] * F[u][i]
            element += ((1 2*A[w][v])*( summp)*numpy.exp( summ))
/  (3*A[w][v] + (1   2*A[w][u]) * (numpy.exp( summ)))
        gradient.append(element)

    return gradient

def train(A, C, iterations = 200):
 # initialize an F
    N = A.shape[0]
    F = np.random.rand(N,C)
```

```
    for n in range(iterations):
        for person in range(N):
            grad = gradient(F, A, person)
            F[person] += 0.005*grad # updating F
            F[person] = np.maximum(0.001 , F[person])
            # F should be nonnegative
    ll = log_likelihood(F, A)
    print('At step %4i logliklihood is %5.4f'%(n,ll))

    return F
```

سوال ۹. سوال تئوری ۸

در کدها انجام شده!

سوال ۱۰. سوال تئوری ۹

$$A_{ij} = \begin{cases} p, z_i = z_j \\ q, z_i \neq z_j \end{cases}$$

سوال ۱۱. سوال تئوری ۱۰

خیر- درایه‌های روی قطر اصلی باید همگی یک باشند ولی برابر با p هستند. همچنین درایه‌های متقارن باید با هم برابر باشند درصورتی که در توصیف احتمالاتی بالا ممکن است متفاوت باشند.

$$A_{ij} = \begin{cases} ۱, i = j \\ p, z_i = z_j, i < j \\ q, z_i = z_j, i < j \\ A_{ji}, i > j \end{cases}$$

سوال ۱۲. سوال شبیه‌سازی ۲

```
import numpy as np
import random
from scipy.stats import bernoulli
import networkx
# import networkx
```

```
n = 15
k = 3
p = 0.6
q = 0.1
Q = np.arange(k*k, dtype=float).reshape(k,k)
for i in range(k):
    for j in range(k):
        Q[i][j] = p if i==j else q

def create_uniform_z(n,k):
    poeple_at_table = [0 for i in range(k)]
    z = list()
    for i in range(n):
        v = random.randint(1,k)
        while(poeple_at_table[v 1]==n/k): v = random.randint(1, k)
        poeple_at_table[v 1] += 1
        z.append(v)
    return np.array(z)


def create_A():
    z = create_uniform_z(n, k)
    A = np.arange(n*n).reshape(n,n)
    for i in range(n):
        for j in range(n):
            value = int(bernoulli(Q[z[i] 1][z[j] 1]).rvs(1))
            if(i==j): value = 1
            elif(i>j): value = A[j][i]
            A[i][j] = value
    return A,z

for i in range(10):
    A,_ = create_A()
    print(A)
```

سوال ۱۳. سوال شبیه‌سازی ۳

```
import numpy as np
import random
from scipy.stats import bernoulli
```

```
import networkx
import networkx as nx

n = 15
k = 3
p = 0.6
q = 0.1
Q = np.arange(k*k, dtype=float).reshape(k,k)
for i in range(k):
    for j in range(k):
        Q[i][j] = p if i==j else q

def create_uniform_z(n,k):
    poeple_at_table = [0 for i in range(k)]
    z = list()
    for i in range(n):
        v = random.randint(1,k)
        while(poeple_at_table[v 1]==n/k): v = random.randint(1, k)
        poeple_at_table[v 1] += 1
        z.append(v)
    return np.array(z)


def create_A():
    z = create_uniform_z(n, k)
    A = np.arange(n*n).reshape(n,n)
    for i in range(n):
        for j in range(n):
            value = int(bernoulli(Q[z[i] 1][z[j] 1]).rvs(1))
            if(i==j): value = 1
            elif(i>j): value = A[j][i]
            A[i][j] = value
    return A,z

A,z = create_A()
G = nx.Graph()
for i in range(n):
    G.add_node(i, table=z[i])

for i in range(n):
    for j in range(n):
        if  i == j:
```

```
            A[i][j] == 0
        if A[i][j] == 1:
            G.add_edge(i, j)
print("hello")
# nx.draw_networkx_labels(G, pos)
nx.draw(G)
```

سوال ۱۴. سوال شبیه‌سازی ۴

```
def hamming_distance(a:np.array,b:np.array):
    return int(np.linalg.norm(a   b,ord=0))
a= np.array([2,2,1,2,2,1])
b= np.array([1,1,2,1,2,2])
print(hamming_distance(a, b))
```

سوال ۱۵. سوال شبیه‌سازی ۵

```
def permutation(lst):
    if len(lst) == 0:
        return []

    if len(lst) == 1:
        return [lst]

    l = [] # empty list that will store current permutation

    for i in range(len(lst)):
       m = lst[i]

       remLst = lst[:i] + lst[i+1:]
       for p in permutation(remLst):
           l.append([m] + p)
    return l


def hamming_distance(a:np.array,b:np.array):
    return int(np.linalg.norm(a   b,ord=0))

def get_class_of_vector(a:np.array, n:int):
    nlist = list(range(1,n+1))
```

```
    aClass = list()
    for p in permutation(nlist):
        aCopy = a.copy()
        for i,v in enumerate(aCopy):
            aCopy[i] = p[v 1]
        aClass.append(aCopy)
    return aClass

def min_hamming_distance(a:np.array, b:np.array, k:int):
    # k table counts
    aClass = get_class_of_vector(a, k)
    return min(list([ hamming_distance(x,b) for x in aClass]))

print(min_hamming_distance(a, b, 2))
```

سوال ۱۶. سوال تئوری ۱۱

$$\frac{N^{۲} - N}{۲}$$

سوال ۱۷. سوال تئوری ۱۲

$$L\left(z\right) = \prod_{i,j=۱,i<j}^{n} A_{ij}$$

سوال ۱۸. سوال تئوری ۱۳

$$\sum_{i,j=۱,i<j}^{n} \log\left(A_{ij}\right)$$

سوال ۱۹. سوال شبیه‌سازی ۶

```
def Liklyhood(A,z):
    p = 0
    for i in range(n):
        for j in range(n):
            # print(z[i], z[j])
            p +=   np.log(Q[z[i] 1][z[j] 1] if A[i][j]==1
```

```
            else (1 Q[z[i] 1][z[j] 1]))
    return p

A,z = create_A()
Liklyhood(A,z)
```

سوال ۲۰. سوال شبیه‌سازی ۷

```
def get_z0(n,k):
    result = list()
    for i in range(int(k)):
        for j in range(int(n/k)):
            result.append(i+1)
    # print("z0 is:", result)
    return np.array(result)

# Q7
def z_fit(A, realz, z0=get_z0(n, k),):
    dlist = list()
    # z0 = get_z0(n, k)
    if len(z0) != n:
        # print("FAAALSE!")
        pass
    T = 5
    bestL = 1e9
    for t in range(T):
        initial_L = Liklyhood(A, z0)
        bestL = initial_L
        MaxDelta = 0
        best = (0,0)
        for i in range(n):
            for j in range(n):
                temp = z0[i]
                z0[i] = z0[j]
                z0[j] = temp
                delta = initial_L   Liklyhood(A, z0)
                if delta > MaxDelta:
                    MaxDelta = delta
                    best = (i,j)
                    bestL = initial_L   delta
                temp = z0[i]
```

```
                z0[i] = z0[j]
                z0[j] = temp
        if best[0]==0:
            print("not completed")
            break
        temp = z0[best[0]]
        z0[best[0]] = z0[best[1]]
        z0[best[1]] = temp
        dlist.append(min_hamming_distance(z0, realz, k))
    return bestL, z0, dlist

A, z0 = create_A()
bestL, z, dlist = z_fit(A, z0)
dlist
```

سوال ۲۱. سوال شبیه‌سازی ۸

```
A,z = create_A()
realL = Liklyhood(A,z)
ZList = list()
LList = list()
print("best L:", realL)
for i in range(10):
    startZ = create_uniform_z(n,k)
    L,z0,_ = z_fit(A, z, startZ)
    ZList.append(z0)
    LList.append(L)
    print("L=",L, "\tZ=",z0)
```

سوال ۲۲. سوال شبیه‌سازی ۹

```
for LItem, ZItem in zip(LList, ZList):
    if LItem == realL:
        print("found a perfect Z", ZItem)
        print("min hamming distance is",
        min_hamming_distance(ZItem, z, k))
```

سوال ۲۳. سوال شبیه‌سازی ۱۰

کد این سوال همان کد بالا است در واقع. همان‌طور که در بالا یافتیم تعدادی هستند که فاصله‌شان صفر شده است.

سوال ۲۴. سوال شبیه‌سازی ۱۱

کد این سوال نیز همان کد سوال ۲۲ می‌باشد. برای این‌کار کد بخش بالا را دوبار دیگر اجرا می‌کنیم.

سوال ۲۵. سوال تئوری ۱۴

درایه‌های ماتریس $A_{i,j}$ نشان می‌دهند که بین دوتا راس i و j یالی وجود دارد یا خیر وقتی یالی باشد یک نشان می‌دهد و اگر نه، ۰ نشان می‌دهد. PMF حاصل:

$$P_X(x) = \begin{cases} \frac{p+q}{۲}, & x = ۱ \\ ۱ - \frac{p+q}{۲} & x = ۰ \end{cases}$$

همچنین

$$A_{i,j}\ Bern(\frac{p+q}{۲})$$

سوال ۲۶. سوال تئوری ۱۵

در متغیر برنولی، exp با احتمال برابر است بنابرین خواهیم داشت:

$$E[Ai, i] = \frac{p+q}{۲}$$

سوال ۲۷. سوال تئوری ۱۶

ماتریس A مربوط به این سوال به شکل زیر خواهد بود

$$\begin{bmatrix} ۱ & ۱ & ۰ & ۰ \\ ۱ & ۱ & ۰ & ۰ \\ ۰ & ۰ & ۱ & ۱ \\ ۰ & ۰ & ۱ & ۱ \end{bmatrix}$$

همجنین ماتریس W به شکل زیر خواهد بود:

$$\begin{bmatrix} p & p & q & q \\ p & p & q & q \\ q & q & p & p \\ q & q & p & p \end{bmatrix}$$

$$|W - I\lambda| = ۰$$

برای محاسبه‌ی مقدار ویژه، باید از ماتریس داده شده، لاندا برابر از ماتریس همانی را کم کرده، از ماتریس حاصل دترمینان بگیریم و آنرا مساوی صفر قرار دهیم. با حل کردن معادله، مقادیر مختلف لاندا بدست می‌آید که مقادیر ویژه‌ی ما هستند
بنابرین داریم:

$$\begin{bmatrix} p-\lambda & p & q & q \\ p & p-\lambda & q & q \\ q & q & p-\lambda & p \\ q & q & p & p-\lambda \end{bmatrix}$$

میدانیم در محاسبه‌ی ماتریس، برای سهولت کار میتوانیم ضریبی از یکی از سطر ها را از سطری دیگر کم کنیم. پس با استناد به این، دترمینان ماتریس بالا با این ماتریس برابر خواهد بود:

$$\begin{bmatrix} -\lambda & \lambda & ۰ & ۰ \\ p & p-\lambda & q & q \\ q & q & p-\lambda & p \\ ۰ & ۰ & \lambda & -\lambda \end{bmatrix}$$

دترمینان حاصل خواهد بود:

$$\lambda^۲(\lambda^۲ - ۴p\lambda + ۴(p^۲ - q^۲))$$

با برابر صفر قرار دادن عبارت بالا، جواب‌های ممکن برای لاندا بدین شکل خواهند بود:

$(۰, ۰, ۲p + ۲q, ۲p - ۲q)$
با داشتن لاندا ها، بردار ویژه‌ها را محاسبه خواهیم کرد:

$$\frac{۱}{\sqrt{۴}}\begin{bmatrix} ۱ \\ -۱ \\ ۱ \\ -۱ \end{bmatrix}$$

سوال ۲۸. سوال تئوری ۱۷

از آن‌جایی که در کل دو خوشه سلایق داریم و هرکس در یکی از این دو است، به این نتیجه می‌رسیم که افراد هم خوشه، ستون‌ها و سطر های یکسانی در ماتریس مجاورت خواهند داشت. پس کل ستون ها به دو نوع تقسیم می‌شوند. به همین دلیل تعداد ستون‌های مستقل خطی دوتا است و می‌دانیم تعداد مقادیر ویژه غیر صفر برابر با رنک ماتریس است در نتیجه دو مقدار ویژه غیرصفر داریم. می‌دانیم هر ماتریس متقارن، n بردار ویژه عمود به هم دارد. پس با جای‌گذاری مقادیر ویژه در معادله اولیه n، بردار ویژه بدست خواهد آمد.

می دانیم ماتریس W به شکل ماتریس یک گراف منتظم است و برای این ماتریس‌ها، بردار ۱، یک بردار ویژه است، پس داریم: (فرض کنیم m تا راس از یک دسته و باقی از دسته دیگر باشند.)

$$W۱ = \lambda۱ \rightarrow mp + (n-m)q = \lambda$$

$$trace(W) = \lambda_۱ + \lambda_۲ = np$$

$$\rightarrow \lambda_۱ = mp + (n-m)q, \lambda_۲ = (n-m)p + (m-n)q$$

جمع درایه های روی قطر ماتریس برابر با جمع مقادیر ویژه آن است.

بردارهای ویژه نیز با جایگذاری مقادیر ویژه بدست می‌آیند.

سوال ۲۹. سوال تئوری ۱۸

در واقع باید سطر های W را جمع بزنیم و در درایه قطری مورد نظر قراردهیم. اگر m نفر از دسته یک و $n-m$ نفر از دسته دوباشند، برای هر راس از جمله خودش به همین تعداد p و q داریم. تمام درایه های قطری برابر با $mp + (n-m)q$ است m = ۲ بود.

در حالت $m = n/۲$:

$$\begin{bmatrix} \frac{n(p+q)}{۲} & ۰ & . & . & ۰ \\ ۰ & \frac{n(p+q)}{۲} & . & . & ۰ \\ . & . & . & . & . \\ . & . & . & . & . \\ ۰ & . & . & . & \frac{n(p+q)}{۲} \end{bmatrix}$$

سوال ۳۰. سوال تئوری ۱۹

$$L_W = \begin{bmatrix} (\frac{n}{۲}-۱)p+\frac{nq}{۲} & -p & . & . & -q \\ -p & (\frac{n}{۲}-۱)p+\frac{nq}{۲} & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ -q & -q & . & . & (\frac{n}{۲}-۱)p+\frac{nq}{۲} \end{bmatrix}$$

$$L_W v = \lambda v \to Wv = D_W v - \lambda v = (\frac{n}{۲}(p+q)-\lambda)v \to eigenvaluesofW = \begin{cases} ۰ \to \lambda = \frac{n}{۲}(p+q) \\ \frac{n}{۲}(p+q) \to \lambda = ۰ \\ \frac{n}{۲}(p-q) \to \lambda = nq \end{cases}$$

سوال ۳۱. سوال تئوری ۲۰

با توجه به بردار ویژه‌های محاسبه شده برای L_w نقاط جدید به صورت زیر تبدیل می‌شوند:

$$\lambda_۱ = ۰ \to v_۱ = (۱,۱,۱,۱)^T$$

$$\lambda_۲ = ۴q \to v_۱ = (۱,۱,-۱,-۱)^T$$

$$۱ : (۱,۱)$$

$$۲ : (۱,۱)$$

$$۳ : (۱,-۱)$$

$$۴ : (۱,-۱)$$

پس خوشه‌ها به خوبی از هم جدا می‌شوند وقتی نمودار را بکشیم.

سوال ۳۲. سوال شبیه‌سازی ۱۲

```
import numpy as np
import random
n = 1000
p = 0.1
q = 0.01
```

```
A = np.zeros((n,n))
W = np.zeros((n,n))
jamee = np.array([0]* (n))
for i in jamee:
    i = random.randint(0,1)

for i in range(n):
    for j in range(i+1, n):
        if jamee[i] == jamee[j] and random.random() < p:
            A[i,j] = 1
            A[i, j] = 1
        elif jamee[i] != jamee[j] and random.random() < q:
            A[i,j] =1
            A[i, j] = 1
D_A = np.diag(np.sum(A, axis = 1))
L_A = D_A   A
eigenValues, eigenVectors = np.linalg.eig(L_A)
n = eigenValues.copy()
n.sort()
a = b =0
for i in range(len(eigenValues)):
    if eigenValues[i] == n[0]:
        a = i
        break
for i in range(len(eigenValues)):
    if eigenValues[i] == n[1] and i != a:
        b = i
print(eigenVectors[a], eigenVectors[b])
```

سوال ۳۳. سوال تئوری ۲۱

برای دست‌یابی به احتمال بالای $\epsilon - ۱$ داریم:

$$۱ - ۴e^{-n} > ۱ - \epsilon \rightarrow ۴e^{n} < \epsilon$$
$$\rightarrow e^{-n} < \frac{\epsilon}{۴} \rightarrow -n < \ln \frac{\epsilon}{۴}$$

پس بدست می‌آوریم که:

$$\rightarrow n > -\ln \frac{\epsilon}{۴} = \ln \frac{۴}{\epsilon}$$

سوال ۳۴. سوال شبیه‌سازی ۱۳

```
import random
from math import ceil

import numpy as np


def second_smallest_index(arr):
    import math
    first = second = math.inf
    i_first = i_second = 0

    for i in range(0, len(arr)):
        if arr[i] < first:
            second = first
            i_second = i_first
            first = arr[i]
            i_first = i
        elif arr[i] < second and arr[i] != first:
            second = arr[i]
            i_second = i
    return i_second


def cluster(n):
    p = 0.1
    q = 0.01
    a_array = [[0] * n for _ in range(n)]
    w_array = [[0] * n for _ in range(n)]
    real_clusters = np.random.choice([ 1 , 1], size=(n,),
    p=[0.5, 0.5])

    for j in range(n):
        for i in range(n):
            if real_clusters[i] == real_clusters[j]:
                w = p
                a = random.choices([0, 1], weights=(1   p, p))[0]
            else:
                w = q
                a = random.choices([0, 1], weights=(1   q, q))[0]
            w_array[i][j] = w
            a_array[i][j] = a
```

```
    A = np.matrix(a_array)
    W = np.matrix(w_array)
    discovered_clusters_a = discover_clusters(A, n)
    discovered_clusters_w = discover_clusters(W, n)
    a_errors = min(
        sum(discovered_clusters_a[i] != real_clusters[i]
        for i in range(n)),
        sum(discovered_clusters_a[i] == real_clusters[i]
        for i in range(n))
    )
    w_errors = min(
        sum(discovered_clusters_w[i] != real_clusters[i]
        for i in range(n)),
        sum(discovered_clusters_w[i] == real_clusters[i]
        for i in range(n))
    )
    print("A errors: ", a_errors)
    print("W errors: ", w_errors)


def discover_clusters(ma, n):
    degree = np.zeros(len(ma))
    row_sum = ma.sum(axis=1)
    for j in range(n):
        degree[j] = row_sum[j, 0]
    D = np.diag(degree)
    L = D   ma
    w, v = np.linalg.eig(L)
    u2 = v[:, second_smallest_index(w)]
    discovered_clusters = [ceil(x.real) * 2   1 for x in u2]
    return discovered_clusters


n_values = [25, 50, 100, 200, 400, 800]
for n in n_values:
    print("n = ", n)
    cluster(n)
```

سوال ۳۵. سوال تئوری ۲۲

خلاصه‌ی تحقیق و گزارش رو این‌جا می‌نویسیم. یکی از منابع این تحقیق، همان فایل $Kmeans$ داده شده هستش.

در واقع این الگوریتم برای خوشه‌بندی داده‌هایی به کار می‌رود که به شکل پارتیشن‌بندی قرار ندارند. یعنی داده‌هایی که در یک خوشه نزدیک به هم هستند. سپس داده‌ها تبدیل می‌شوند و در فضایی n بعدی می‌روند و خوشه‌ها آن‌جا پارتیشن‌بندی می‌شوند. در این‌جا می‌توان با الگوریتمی مثل $k-means$ داده‌ها را به خوشه و پارتیشن‌های مختلف افراز کرد.

سوال ۳۶. سوال شبیه‌سازی ۱۴

```
import random
from math import ceil
from sklearn.datasets import fetch_california_housing
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


def second_smallest_index(arr):
    import math
    first = second = math.inf
    i_first = i_second = 0

    for i in range(0, len(arr)):
        if arr[i] < first:
            second = first
            i_second = i_first
            first = arr[i]
            i_first = i
        elif arr[i] < second and arr[i] != first:
            second = arr[i]
            i_second = i
    return i_second


def cluster(n):
    p = 0.1
    q = 0.01
    a_array = [[0] * n for _ in range(n)]
    w_array = [[0] * n for _ in range(n)]
    real_clusters = np.random.choice([ 1 , 1], size=(n,), p=[0.5, 0.5])
```

```
    for j in range(n):
        for i in range(n):
            if real_clusters[i] == real_clusters[j]:
                w = p
                a = random.choices([0, 1], weights=(1   p, p))[0]
            else:
                w = q
                a = random.choices([0, 1], weights=(1   q, q))[0]
            w_array[i][j] = w
            a_array[i][j] = a

    A = np.matrix(a_array)
    W = np.matrix(w_array)
    discovered_clusters_a = discover_clusters(A, n)
    discovered_clusters_w = discover_clusters(W, n)
    a_errors = min(
        sum(discovered_clusters_a[i] != real_clusters[i]
        for i in range(n)),
        sum(discovered_clusters_a[i] == real_clusters[i]
        for i in range(n))
    )
    w_errors = min(
        sum(discovered_clusters_w[i] != real_clusters[i]
        for i in range(n)),
        sum(discovered_clusters_w[i] == real_clusters[i]
        for i in range(n))
    )
    print("A errors: ", a_errors)
    print("W errors: ", w_errors)


def discover_clusters(ma, n):
    degree = np.zeros(len(ma))
    row_sum = ma.sum(axis=1)
    for j in range(n):
        degree[j] = row_sum[j, 0]
    D = np.diag(degree)
    L = D   ma
    w, v = np.linalg.eig(L)
    u2 = v[:, second_smallest_index(w)]
    discovered_clusters = [ceil(x.real) * 2   1 for x in u2]
    return discovered_clusters
```

```
california_housing = fetch_california_housing(as_frame=True)
med_incs = np.array(california_housing.data[:]['MedInc'])
.reshape( 1 , 1)
kmeans = KMeans(n_clusters=3, random_state=0, max_iter=1000,
n_init="auto").fit(med_incs)
labels = kmeans.labels_
colors = [label + 100 for label in labels]
df = pd.DataFrame(data=california_housing.data,
columns=california_housing.feature_names)
df.plot(kind='scatter', x='Latitude', y='Longitude',
color=colors,alpha=0.5)
```

سوال ۳۷. سوال شبیه‌سازی ۱۵

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
from sklearn.cluster import KMeans

def spectral_cluster(matrix, k):
    n = len(matrix)
    D = [[0] * n for i in range(n)]
    for i in range(len(matrix)):
        sum = 0
        for j in range(len(matrix[i])):
            sum += matrix[i][j]
        D[i][i] = sum

    L = [[0] * n for i in range(n)]
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            L[i][j] = D[i][j]   matrix[i][j]

    u, v = np.linalg.eig(L)
    indices = np.argsort(u)[1:]
    V = []
    for i in range(k):
        V.append(v[:, indices[i]])

    return np.real(V)
```

```
G = nx.karate_club_graph()
n = len(G.nodes)

A = [[0 for i in range(n)] for j in range(n)]
for i in G.edges:
    A[i[0]][i[1]] = 1
    A[i[1]][i[0]] = 1

V = np.array(spectral_cluster(A, 2)).T

for k in range(2, 5):
    kmeans = KMeans(n_clusters=k, random_state=0, n_init="auto").fit(V)
    nx.draw(G, node_color=kmeans.labels_+1, with_labels=True)
    plt.show()
```

سوال ۳۸. سوال شبیه‌سازی ۱۶

```
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from sklearn.cluster import KMeans

def spectral_cluster(matrix, k):
    n = len(matrix)
    D = [[0] * n for i in range(n)]
    for i in range(len(matrix)):
        sum = 0
        for j in range(len(matrix[i])):
            sum += matrix[i][j]
        D[i][i] = sum

    L = [[0] * n for i in range(n)]
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            L[i][j] = D[i][j]   matrix[i][j]

    u, v = np.linalg.eig(L)
    indices = np.argsort(u)[1:]
    V = []
    for i in range(k):
```

```
            V.append(v[:, indices[i]])

        return np.real(V)

    def conductivity(A, z, s):
        cs = 0
        ms = 0
        for i in range(len(A)):
            if z[i] == s:
                for j in range(len(A)):
                    if A[i][j] == 1:
                        if z[j] == s:
                            ms += 1
                        else:
                            cs += 1
        return cs / (ms + cs)


    def mean_conductivity(A, z):
        sum = 0
        counter = 0
        for i in range(np.min(z), np.max(z) + 1):
            sum += conductivity(A, z, i)
            counter += 1

        return sum / counter


    if __name__ == "__main__":
        G = nx.karate_club_graph()
        n = len(G.nodes)

        A = [[0 for i in range(n)] for j in range(n)]
        for i in G.edges:
            A[i[0]][i[1]] = 1
            A[i[1]][i[0]] = 1

        V = np.array(spectral_cluster(A, 2)).T

        conductivities = []
        for k in range(2, 11):
            kmeans = KMeans(n_clusters=k, random_state=0, n_init="auto").fit(V)
```

```
        conductivities.append(mean_conductivity(A, kmeans.labels_ + 1))

    plt.plot(range(2, 11), conductivities)
    plt.xlabel("clusters count")
    plt.ylabel("mean conductivity")
    plt.show()
```

سوال ۳۹. سوال تئوری ۲۳

هر جفت دوست به احتمال p درست تعیین شده و هر جفت غیردوست هم به احتمال q = p - ۱ درست تعیین میشود.

تعداد جفت‌های غیردوست

$$n * (n - ۱)/۲ - m$$

است. پس طبق اصل ضرب داریم:

$$p^m (۱ - p)^{\binom{n}{۲} - m}$$

سوال ۴۰. سوال تئوری ۲۴

تعداد کل حالات برابر

$$\binom{\binom{n}{۲}}{m}$$

و یکی از آنها را به احتمال برابر انتخاب می‌کنیم.

سوال ۴۱. سوال تئوری ۲۵

هر جفت مستقل از بقیه به احتمال p یال می‌کشیم و به احتمال

$$q = ۱ - p$$

یال نمی‌کشیم. برای m تا از جفت ها در صورت یال کشیدن رابطه درست تعیین شده است. یعنی به احتمال p برای $k = \binom{n}{۲} - m$ جفت هم به احتمال q رابطه هم‌سلیقگی درست تعیین می‌شود.

X را تعداد یال‌هایی تعریف می‌کنیم که به درستی کشیده شده‌اند و Y را هم تعداد یال‌هایی تعریف می‌کنیم که به درستی کشیده نشده‌اند. هدف سوال $P(X + Y \geq \frac{m+k}{۵})$ است.

می‌دانیم که:

$$X \sim Binom(m, p)$$

$$Y \sim Binom(k, q)$$

$$E[X] = mp \qquad Var[x] = mpq$$

$$E[Y] = kq \qquad Var[x] = kqp$$

جواب را می‌توان به صورت جمع یک سری نوشت ولی برای محاسبه رابطه مستقیم، با قضیه حد مرکزی X و Y را تخمین می‌زنیم.

$$X \approx Normal(mp, mpq)$$

$$Y \approx Normal(kq, kpq)$$

و جمع دو متغیر تصادفی که از توزیع نورمال پیروی می‌کنند، یک توزیع نورمال است.

$$X + Y \approx Normal(mp + kq, mpq + kpq = (m + k)pq)$$

$$Z = \frac{X+Y-mp-kq}{\sqrt{(m+k)pq}} \qquad Z \approx Normal(۰, ۱)$$

$$P(X + Y \geq \frac{m+k}{۵}) = P(Z \geq \frac{\frac{m+k}{۵}-mp-kq}{\sqrt{(m+k)pq}}) = ۱ - \phi(\frac{\frac{m+k}{۵}-mp-kq}{\sqrt{(m+k)pq}})$$

سوال ۴۲. سوال شبیه‌سازی ۱۷

```
from random import random

n=1000
p=0.0034
m=3000
N=10
def simul():
        edges=0
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                edges+=1
        return edges

average=0
for i in range(N):
        average+=simul()
average/=N
print(average)
```

خیر. جواب به دست آمده حدودا نصف m است.

سوال ۴۳. سوال تئوری ۲۶

طبق قانون اعداد بزرگ، این مقدار میانگین باید حدود امید ریاضی خروجی کد باشد.

$\binom{n}{2}$ جفت راس داریم و هر جفت به احتمال p و مستقل از بقیه یال دارند. پس از توزیع باینومیال پیروی میکنند و امید ریاضی تعداد یال‌ها برابر $p\binom{n}{2}$ است.

برای مقادیر پرسش شبیه‌سازی جواب برابر ۱۶۹۸/۳ است.

به طور کلی هم باید رابطه $|\frac{m}{\binom{n}{2}} - p| < \epsilon$ برقرار باشد که مقدار اپسیلون به دقت مورد نظر بستگی دارد.

سوال ۴۴. سوال شبیه‌سازی ۱۸

```
from random import random
import matplotlib.pyplot as plt

n=1000
p=0.00016
N=10

def simul():
        deg=[0]*n
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                deg[i]+=1
                                deg[j]+=1
        L=sum(deg)/n
        res=0
        for i in range(n):
                if (deg[i]>=L):
                        res+=1
        return res

results=[simul() for i in range(N)]
average=sum(results)/N
print(average)


deg=[0]*n
for i in range(n):
        for j in range(i):
                if random()<p:
                        deg[i]+=1
                        deg[j]+=1
plt.hist(deg)
```

```
plt.show()
```

$Output$: ۱۵۱/۳

سوال ۴۵. سوال تئوری ۲۷

$$(n-۱)p = ۹۹۹ \times ۰/۰۰۰۱۶ = ۰/۱۵۹۸۴$$

سوال ۴۶. سوال تئوری ۲۸

با توجه به پرسش قبل و بزرگ بودن n طبق قضیه حد مرکزی میتوانیم متغیر تصادفی deg_i که درجه راس i است را با توزیع نورمال تخمین بزنیم.

$$Var[deg_i] = (n-۱)p(۱-p) \simeq ۰/۱۶$$

پس می‌توانیم فرض کنیم احتمال اینکه راس درجه بیشتر از ۲ داشته باشیم صفر است. و اینکه احتمال اینکه میانگین درجات زیاد باشد نیز نزدیک صفر است، پس از آن هم صرف نظر می‌کنیم. در نتیجه احتمال اینکه یک نفر هم‌رنگ نباشد برابر احتمال این است که این راس هیچ یالی نداشته باشد. و این مقدار برابر $(۱-p)^{n-۱}$ است. در نتیجه احتمال هم‌رنگ بودن یک نفر برابر $۱-(۱-p)^{n-۱} \simeq ۰/۱۴۷$ است و امیدریاضی تعداد هم‌رنگ‌ها برابر ۱۴۷ می‌شود.

سوال ۴۷. سوال شبیه‌سازی ۱۹

```
from random import random

n=3000
p=0.01
N=5

def simul():
        G=[[0]*n for i in range(n)]
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G[i][j]=G[j][i]=1
        res1=0
        res2=0
        for v in range(n):
```

```
                adj=[]
                for u in range(v):
                        if G[u][v]:
                                adj.append(u)
                res=0
                for u in adj:
                        for w in adj:
                                if u<w:
                                        if G[u][w]:
                                                res1+=1
                                        else:
                                                res2+=1
        return res1, res2

mean1=0
mean2=0
for i in range(N):
        res1, res2 = simul()
        mean1+=res1
        mean2+=res2
mean1/=N
mean2/=N
print(mean1)
print(mean2)
```

$$Output:$$

۴۵۰۳/۸

۴۴۷۶۹۱/۲

سوال ۴۸. سوال تئوری ۲۹

خاصیت تراگذری = مثلث به دلیل خطی بودن امیدریاضی، احتمال وجود خاصیت را برای ۳ راس حساب کرده و در $\binom{n}{۳}$ ضرب می‌کنیم

تراگذری:

$$\binom{n}{۳}p^۳$$

زنجیره‌ای:

$$\binom{n}{۳}۳p^۲(۱-p)$$

سوال ۴۹. سوال تئوری ۳۰

$$P(\text{۳ edges} \mid \text{at least ۲ edges}) = \frac{P(\text{۳ edges})}{P(\text{at least ۳ edges})} = \frac{p^{۳}}{p^{۳} + ۳p^{۲}(۱-p)} = \frac{p}{p + ۳(۱-p)} = \frac{p}{۳-۲p} = ۰/۰۰۳$$

سوال ۵۰. سوال شبیه‌سازی ۲۰

```
from random import random

n=1000
p=0.003

G=[[0]*n for i in range(n)]
for i in range(n):
        for j in range(i):
                if random()<p:
                        G[i][j]=G[j][i]=1

mean=0
for v in range(n):
        adj=[]
        for u in range(v):
                if G[u][v]:
                        adj.append(u)
        res=0
        for u in adj:
                for w in adj:
                        if u<w and G[u][w]:
                                res+=1
        mean+=res

mean/=n
print(mean)
```

$Output$: ۰/۰۱

سوال ۵۱. سوال تئوری ۳۱

امیدریاضی تعداد مثلث‌های حاوی یک راس خاص را می‌خواهیم. پس به $\binom{n-۱}{۲}$ حالت ۲ راس انتخاب کرده و به احتمال $p^۳$ یک مثلث می‌سازند. چون امید ریاضی خطی است جواب برابر با $p^۳\binom{n-۱}{۲}$ می‌شود.

سوال ۵۲. سوال شبیه‌سازی ۲۱

```
from random import random
import networkx as ntx

N=1
n=1000
p=0.033
# in problem statement p=0.0033,
# but the problem author is an idiot
# who does not know basic graph theory and probability
# if p=0.0033, its easy to prove the probability of
# having a connected graph is almost 0
# **** EE project


def gen_graph(n, p)  > ntx.Graph:
        G=ntx.empty_graph(n)
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G.add_edge(i, j)
        return G

def simul():
        G=gen_graph(n, p)
        sp = dict(ntx.all_pairs_shortest_path_length(G))
        mean=0
        for i in range(n):
                for j in range(i):
                        mean+=sp[i][j]
        return mean/(n*(n 1)/2)
```

```
results=[simul() for i in range(N)]
average=sum(results)/N
print(average)
```

Output : ۲/۲۹۵۴۶۱۴۶۱۴۶۱۴۶۱۵

سوال ۵۳. سوال شبیه‌سازی ۲۲

```
from random import random
import networkx as ntx

N=100
n=50
p=0.34

def gen_graph(n, p)  > ntx.Graph:
        G=ntx.empty_graph(n)
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G.add_edge(i, j)
        return G

def simul():
        G=gen_graph(n, p)
        return ntx.diameter(G)

results=[simul() for i in range(N)]
average=sum(results)/N
print(average)
```

Output : ۲/۸۲

سوال ۵۴. سوال شبیه‌سازی ۲۳

```
from random import random
import networkx as ntx
import matplotlib.pyplot as plt
```

```
p=0.34
N=100

def gen_graph(n, p)  > ntx.Graph:
        G=ntx.empty_graph(n)
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G.add_edge(i, j)
        return G

def simul(n):
        G=gen_graph(n, p)
        if ntx.connected.is_connected(G):
                return ntx.diameter(G)
        return None

def mean_simul(n):
        results=[]
        for i in range(N):
                res=simul(n)
                if res!=None:
                        results.append(res)

        return sum(results)/len(results)


x=range(10, 201, 10)
y=[mean_simul(n) for n in x]

plt.plot(x, y, color='red')
plt.show()
```

این نمودار نزولی است و با افزایش تعداد راس‌ها به سمت ۲ میل می‌کند.

اگر قطر یک گراف ۲ باشد بدین معنی اسد که هر دو راس غیرهمسایه، یک همسایه مشترک دارند.

سوال ۵۵. سوال تئوری ۳۲

$E_{u,v}$ را برابر یال داشتن دو راس تعریف می‌کنیم.

$$P(I_{u,v}) = \prod_{w \neq u,v} (۱ - P(I_{u,w})P(I_{u,w}))$$

$$P(I_{u,v}) = \prod_{w \neq u,v} (۱ - p^۲)$$
$$P(I_{u,v}) = (۱ - p^۲)^{n-۲}$$

سوال ۵۶. سوال تئوری ۳۳

امیدریاضی خطی است پس امیدریاضی تعداد جفت‌های بدون همسایه مشترک برابر جمع اختمال همسایه مشترک نداشتن تمام جفت راس‌ها است. و این احتمال در پرسش قبلی محاسبه شده، پس جواب برابر است با:

$$\binom{n}{۲}(۱ - p^۲)^{n-۲}$$

سوال ۵۷. سوال تئوری ۳۴

$$MarkovInequality:$$
$$P(X_n \geq ۱) \leq \frac{E[X_n]}{۱} = \binom{n}{۲}(۱ - p^۲)^{n-۲}$$
$$\lim_{n\to\infty} \binom{n}{۲}(۱ - p^۲)^{n-۲} =$$
$$\lim_{n\to\infty} \frac{n(n-۱)}{۲\ (۱ - p^۲)^{۲-n}} =$$
$$\lim_{n\to\infty} \frac{n(n-۱)}{۲\ (e^{ln(۱-p^۲)})^{۲-n}} =$$
$$\lim_{n\to\infty} \frac{n(n-۱)}{۲\ e^{ln(۱-p^۲)(۲-n)}} =$$
$$\lim_{n\to\infty} \frac{۲n-۱}{-۲\ ln(۱-p^۲)e^{ln(۱-p^۲)(۲-n)}} =$$
$$\lim_{n\to\infty} \frac{۲}{-۲\ ln(۱-p^۲)^۲e^{ln(۱-p^۲)(۲-n)}} =$$
$$\frac{-۱}{ln(۱-p^۲)^۲} \lim_{n\to\infty} e^{ln(۱-p^۲)(n-۲)} =$$
$$\frac{-۱}{ln(۱-p^۲)^۲} \lim_{n\to\infty} (۱-p^۲)^{n-۲} = ۰$$

سوال ۵۸. سوال تئوری ۳۵

طبق پرسش قبل احتمال اینکه حداقل یک جفت راس همسایه مشترک نداشته باشند، با زیاد شدن n به سمت صفر میل می‌کند. در نتیجه برای n های بزرگ به احتمال تفریبن یک هر دو راسی همسایه مشترک دارند و در نتیجه فاصله آنحا حداکثر ۲ است. همچنین احتمال ۱ بودن قطر گراف هم با زیاد شدن n به سمت صفر میل می‌کند چون این احتمال برابر

$$p^{\binom{n}{2}}$$

است و

$$\lim_{n\to\infty} p^{\binom{n}{2}} = 0$$

پس با زیاد شدن n به احتمال تقریبا ۱ قطر گراف ۲ است. و این عدد به p وابسته نیست. تنها کافی است که مقدار p برابر صفر یا یک نباشد. این نتیجه با نتیجه شبیه‌سازی نیز تطابق دارد. در آنجا هم قطر گراف به ۲ میل می‌کند

سوال ۵۹. سوال شبیه‌سازی ۲۴

```
from random import random
import networkx as ntx

n=100
p=0.34
N=100

def gen_graph(n, p)  > ntx.Graph:
        G=ntx.empty_graph(n)
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G.add_edge(i, j)
        return G

def simul(n):
        G=gen_graph(n, p)
        return sum(list(ntx.triangles(G).values()))//3

def mean_simul(n):
        results=[]
        for i in range(N):
                res=simul(n)
                if res!=None:
```

```
                        results.append(res)

        return sum(results)/len(results)


print(mean_simul(n))
```

Output : ۶۳۲۳/۸۴

سوال ۶۰. سوال شبیه‌سازی ۲۵

```
from random import random
import networkx as ntx
import matplotlib.pyplot as plt

N=100

def gen_graph(n, p)  > ntx.Graph:
        G=ntx.empty_graph(n)
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G.add_edge(i, j)
        return G

def simul(n):
        G=gen_graph(n, 60/n**2)
        return sum(list(ntx.triangles(G).values()))//3

def mean_simul(n):
        results=[]
        for i in range(N):
                res=simul(n)
                if res!=None:
                        results.append(res)

        return sum(results)/len(results)

x=range(10, 101, 10)
y=[mean_simul(n) for n in x]
```

```
plt.plot(x, y, color='red')
plt.show()
```

سوال ۶۱. سوال شبیه‌سازی ۲۶

خیر. این مقدار به عدد خاصی میل نمی‌کند و با افزایش تعداد راس‌ها زیاد می‌شود.

```
from random import random
import networkx as ntx
import matplotlib.pyplot as plt

p=0.34
N=100

def gen_graph(n, p)  > ntx.Graph:
        G=ntx.empty_graph(n)
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G.add_edge(i, j)
        return G

def simul(n):
        G=gen_graph(n, p)
        return sum(list(ntx.triangles(G).values()))//3

def mean_simul(n):
        results=[]
        for i in range(N):
                res=simul(n)
                if res!=None:
                        results.append(res)

        return sum(results)/len(results)


x=range(10, 101, 10)
y=[mean_simul(n) for n in x]

plt.plot(x, y, color='red')
plt.show()
```

سوال ۶۲. سوال شبیه‌سازی ۲۷

```
from random import random
import networkx as ntx
import matplotlib.pyplot as plt

N=100

def gen_graph(n, p)  > ntx.Graph:
        G=ntx.empty_graph(n)
        for i in range(n):
                for j in range(i):
                        if random()<p:
                                G.add_edge(i, j)
        return G

def simul(n):
        G=gen_graph(n, 1/n)
        return sum(list(ntx.triangles(G).values()))//3

def mean_simul(n):
        results=[]
        for i in range(N):
                res=simul(n)
                if res!=None:
                        results.append(res)

        return sum(results)/len(results)

x=range(50, 1201, 50)
y=[mean_simul(n) for n in x]

plt.plot(x, y, color='red')
plt.show()
```

این مقدار با وجود زیاد شدن تعداد راس‌ها باز هم در بازه ۰/۱ تا ۰/۲ قرار دارد. پس می‌توان نتیجه گرفت با زیاد شدن تعداد راس‌ها به عددی در این بازه مایل می‌شود

سوال ۶۳. سوال تئوری ۳۶

$$P_{i,j} = \begin{cases} \frac{۱}{d(i)} & A_{i,j} = ۱ \\ \\ ۰ & A_{i,j} = ۰ \end{cases}$$

سوال ۶۴. سوال تئوری ۳۷

$\frac{۱}{d(i)}$ قرار دارد.

$$\implies P = AD^{-۱}$$

سوال ۶۵. سوال تئوری ۳۸

$$\sum_{x=۱}^{n} P_{ix} \cdot P_{xj}$$

که می‌توان آن را به شکل مسیرهای به طول دو از i به j تعریف کرد.

سوال ۶۶. سوال تئوری ۳۹

$$P_{i,j}^{(t)} = [\boldsymbol{P}^t]_{i,j}$$

سوال ۶۷. سوال تئوری ۴۰

مسیر $i \to x_۱ \to ... \to x_{t-۱} \to j$ را در نظر بگیرید. احتمال اینکه با شروع از i این مسیر طی شود برابر:

$$A_۱ = \frac{۱}{d(x_۱)} \cdot ... \cdot \frac{۱}{d(x_{t-۱})} \cdot \frac{۱}{d(j)}$$

همچنین احتمال اینکه برعکس این مسیر با شروع از j طی شود برابر:

$$A_۲ = \frac{۱}{d(x_{t-۱})} \cdot ... \cdot \frac{۱}{d(x_۱)} \cdot \frac{۱}{d(i)}$$

اگر نسبت دو مسیر $A_۱$ و $A_۲$ را بگیریم، می‌فهمیم:

$$\frac{A_۱}{A_۲} = \frac{d(i)}{d(j)}$$

و این به ازای هر مسیر از i به j برقرار است.

$$\implies \frac{P_{i,j}^{(t)}}{P_{j,i}^{(t)}} = \frac{d(i)}{d(j)}$$

سوال ۶۸. سوال تئوری ۴۱

اگر این دو کاربر i و j با هم هم‌سلیقه باشند، احتمال اینکه در یک خوشه قرار بگیرند بیشتر است که در نتیجه‌ی آن، $P_{ij}^{(t)}$ افزایش می‌یابد. در این صورت، k اگر i و j هم‌خوشه باشند، با احتمال زیادی با آنها هم‌خوشه است که باعث می‌شود احتمال $P_{jk}^{(t)}$ و $P_{ik}^{(t)}$ بالا رود.

سوال ۶۹. سوال تئوری ۴۲

برای بدست آوردن احتمال رفتن از یک خوشه C به راسی مثل ،k میانگین احتمال رئوس آن خوشه را در نظر می‌گیریم (البته احتمالها برای اعضای یک خوشه بسیار مشابه است):

$$P_{C,k}^{(t)} = \frac{\sum_{i=۱}^{size\ of\ C} P_{i,k}^{(t)}}{size\ of\ C}$$

واضحا به طور برعکس $P_{k,C}^{(t)}$ پس خواهیم داشت:

$$r_{C_۱,C_۲} = \sqrt{\sum_{k=۱}^{n} \frac{(P_{C_۱,k}^{(t)} - (P_{k,C_۲}^{(t)})^۲}{d(k)}}$$

سوال ۷۰. سوال تئوری ۴۳

تعداد یال‌های درون و بیرون خوشه‌ها رو می‌شه معیاری برای یافتن خوشه‌بندی بهینه در نظر بگیریم و مثلا زمانی که تعداد یال‌های درون اون‌ها از بیرون اون‌ها بیشتر باشه، معیار خوبی می‌دونیم. در واقع برتری تعداد یال‌های درون به بیرون خوشه‌ها رو می‌شه معیاری در نظر گرفت.

سوال ۷۱. سوال شبیه‌سازی ۲۸

```
import math
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx


def P_t(t, A, D):
    P = np.dot(np.linalg.inv(D), A)

    result = P
```

```
    for i in range(t   1):
        result = np.dot(result, P)
    return result


def R_i_j(n, t, A, D, d, i, j):
    result = 0
    P = P_t(t, A, D)
    for k in range(n):
        result += ((P[i][k]   P[j][k]) ** 2) / d[k]
    return math.sqrt(result)


def P_Ci(P, ci, A, k):
    result = 0
    for i in ci:
        result += P[i][k]

    return result / len(ci)


def RC_iC_j(n, t, A, D, d, C1, C2):
    P = P_t(t, A, D)
    result = 0
    for k in range(n):
        result += ((P_Ci(P, C1, A, k)   P_Ci(P, C2, A, k)) ** 2) / d[k]
    return math.sqrt(result)


G = nx.karate_club_graph()
n = len(G.nodes)
t = 2

A = [[0 for i in range(n)] for j in range(n)]

edgesTotal = 0
for i in G.edges:
    edgesTotal += 1
    A[i[0]][i[1]] = 1
    A[i[1]][i[0]] = 1

D = [[0 for i in range(n)] for j in range(n)]
```

```
d = []
for i in range(n):
    s = sum(A[i])
    d.append(s)
    D[i][i] = s

C = []
for i in range(n):
    c = []
    c.append(i)
    C.append(c)

diffs = []

edgesIn = []
edgesOut = []

while len(C) > 1:
    first = 0
    second = 0
    minOfDif = 1e7
    for i in range(len(C)):
        for j in range(i):
            r = RC_iC_j(n, t, A, D, d, C[i], C[j])
            if minOfDif > r:
                minOfDif = r
                first = i
                second = j

    edIn = 0
    edOut = 0
    for i in range(len(C)):
        for j in range(len(C[i])):
            for z in range(j):
                if A[j][z] == 1:
                    edIn += 1

    edOut = edgesTotal   edIn

    edgesIn.append(edIn)
    edgesOut.append(edOut)
```

```
    cfirst = C.pop(first)
    csecond = C.pop(second)
    C.append(cfirst + csecond)

    r = 0
    for i in range(len(C)):
        for j in range(i):
            r += RC_iC_j(n, t, A, D, d, C[i], C[j])

    diffs.append(r)

    if 2 * edOut <= edIn:
        color_map = []
        for node in G:
            counter = 0
            for i in range(len(C)):
                if counter == 0:
                    for j in C[i]:
                        if j == node and counter == 0:
                            color_map.append(i)
        nx.draw(G, node_color=color_map, with_labels=True)
        plt.show()
        plt.clf()
        break

edIn = 0
edOut = 0
for i in range(len(C)):
    for j in range(len(C[i])):
        for z in range(j):
            if A[j][z] == 1:
                edIn += 1

edOut = edgesTotal   edIn

edgesIn.append(edIn)
edgesOut.append(edOut)

print(edgesIn)
print(edgesOut)
```

```
import math
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx


def P_t(t, A, D):
    P = np.dot(np.linalg.inv(D), A)

    result = P
    for i in range(t   1):
        result = np.dot(result, P)
    return result


def R_i_j(n, t, A, D, d, i, j):
    result = 0
    P = P_t(t, A, D)
    for k in range(n):
        result += ((P[i][k]   P[j][k]) ** 2) / d[k]
    return math.sqrt(result)


def P_Ci(P, ci, A, k):
    result = 0
    for i in ci:
        result += P[i][k]

    return result / len(ci)


def RC_iC_j(n, t, A, D, d, C1, C2):
    P = P_t(t, A, D)
    result = 0
    for k in range(n):
        result += ((P_Ci(P, C1, A, k)   P_Ci(P, C2, A, k)) ** 2) / d[k]
    return math.sqrt(result)


G = nx.karate_club_graph()
n = len(G.nodes)
```

```
t = 5

A = [[0 for i in range(n)] for j in range(n)]

edgesTotal = 0
for i in G.edges:
    edgesTotal += 1
    A[i[0]][i[1]] = 1
    A[i[1]][i[0]] = 1

D = [[0 for i in range(n)] for j in range(n)]
d = []
for i in range(n):
    s = sum(A[i])
    d.append(s)
    D[i][i] = s

C = []
for i in range(n):
    c = []
    c.append(i)
    C.append(c)

diffs = []

edgesIn = []
edgesOut = []

while len(C) > 1:
    first = 0
    second = 0
    minOfDif = 1e7
    for i in range(len(C)):
        for j in range(i):
            r = RC_iC_j(n, t, A, D, d, C[i], C[j])
            if minOfDif > r:
                minOfDif = r
                first = i
                second = j

    edIn = 0
    edOut = 0
```

```
    for i in range(len(C)):
        for j in range(len(C[i])):
            for z in range(j):
                if A[j][z] == 1:
                    edIn += 1

    edOut = edgesTotal   edIn

    edgesIn.append(edIn)
    edgesOut.append(edOut)

    cfirst = C.pop(first)
    csecond = C.pop(second)
    C.append(cfirst + csecond)

    r = 0
    for i in range(len(C)):
        for j in range(i):
            r += RC_iC_j(n, t, A, D, d, C[i], C[j])

    diffs.append(r)

    if 2 * edOut <= edIn:
        color_map = []
        for node in G:
            counter = 0
            for i in range(len(C)):
                if counter == 0:
                    for j in C[i]:
                        if j == node and counter == 0:
                            color_map.append(i)
        nx.draw(G, node_color=color_map, with_labels=True)
        plt.show()
        plt.clf()
        break

edIn = 0
edOut = 0
for i in range(len(C)):
    for j in range(len(C[i])):
        for z in range(j):
            if A[j][z] == 1:
```

```
                edIn += 1

edOut = edgesTotal   edIn

edgesIn.append(edIn)
edgesOut.append(edOut)

print(edgesIn)
print(edgesOut)
```

موفق باشید.