

به نام خدا



## آشنایی با برخی روش‌های خوش‌بندی گراف‌ها

درس : آمار و احتمال مهندسی

استاد : دکتر علی شریفی زارچی

تهیه شده توسط :

مهریار افشاریان مهر - ۴۰۰۱۰۹۷۳۵

مریم شیران - ۴۰۰۱۰۹۴۴۶

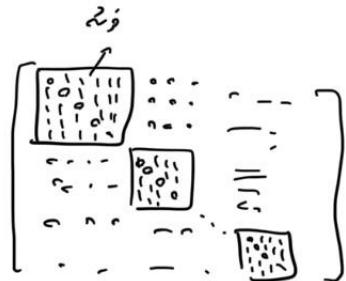
علیرضا امیری باوندپور - ۴۰۰۱۰۰۷۲۴

دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

**بخش دوم: زهرچه رنگ تعلق یزیرد آزاد است!**

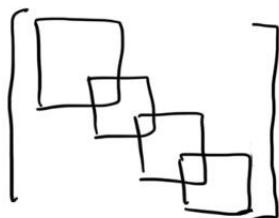
پرسش تئوری ۱ :



۱- ماتریس ب یک شکل مانند شکل در ره تبلیغ از سفر

## ○ پرسش تئوری ۲:

<sup>۲</sup> هاترین نتیجه مانند بالا فواید می‌باشد که خود حای مردمی نسل باهم از آن خواسته داشت.



پرسش تئوری ۳:

**۳- زنگ لینک که هنوز صرده بزارهای ملکه زدن دلیل صرف ظاهیر باش که ام الهم هم بدلیل**

آنھا برائے ما:

$$1 - \rho(\text{نامه} \rightarrow \text{رسانی}) = 1 - (1 - p_{R_1})(1 - p_{R_2}) \dots (1 - p_{R_n})$$

پرسش تئوری ۴:

**۵- احتمال حمل سلیمانی** یعنی احتمال این است که در حداقل یک فوتیه بضم باشند پس باعث کردن حالته علی ادامه داشتم.

$$P = 1 - \sum_{i=1}^c (1 - (1 - e^{-F_i F_{Vi}})) = 1 - \prod_{i=1}^c e^{-F_i F_{Vi}}$$

بہیچے اسے کہا نہ یاد نہیں ۷ امن افکار میں بزرگ تری سُد.

پرسش تئوری ۵ ○

$$\ln(\rho(A|F)) = \sum_{u=1}^n \sum_{v=1}^m \ln((1 - e^{-F_{uv} F_{vi}}) + \sum_{i \neq v} F_{ui} F_{vi}) \quad - 5$$

پرسش تئوری ۶:

۶- بین ازایه مکاریان نه نموده . باید گرانتریان  $\Delta F$  نهان بود رسم.

پرسش تئوری ۷:

$$\frac{\partial L(F)}{\partial F_{uv}} = \sum_v^n \frac{F_{uv} (1 - e^{-F_{uv} F_{vi}})}{1 - e^{-F_{uv} F_{vi}}} - \sum_v^n F_{vi} \quad - 7$$

پرسش تئوری ۸:

۸- احوال حسنه دارد .  $P_{uv|c}$  و توزع این ناجا .

$\sum P_{uv}(i) \rightarrow \text{binomial}$

$$\frac{\partial F}{\partial F_{ui,j}} = \sum \frac{\partial}{\partial F_{ui,j}} \ln(\rho(X_{uv} | \alpha_{uv})) = \sum \frac{P(x)}{P(X_{uv} | \alpha_{uv})} \rightarrow \rho'(X_{uv} | \alpha_{uv}) = \sum P_{ui} P_{uj} (1 - P_j)$$

پرسش شبیه سازی ۱:

```

import numpy as np
import math

def probability(u, v, F):
    tmp = 0.0
    C = F.shape[1]
    for c in range(C):
        tmp += F[u, c] * F[v, c]
    tmp = 1 - math.exp(-tmp)
    return tmp

def d_probability(u, v, c, F):
    ans = probability(u, v, F)
    return F[v, c] * (1 - ans)

def log_likelihood(F, A):
    n = A.shape[0]
    C = F.shape[1]
    log_likelihood = 0.0
    for i in range(n):
        for j in range(n):
            if (A[i, j] == 1):
                log_likelihood += math.log(probability(i, j, F))
            else:
                log_likelihood += math.log(1.001 - probability(i, j, F))
    return log_likelihood

```

```

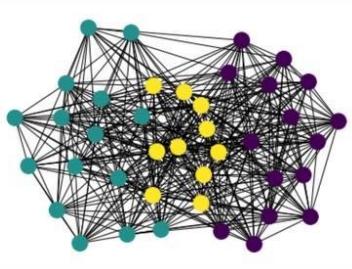
def gradient(F, A, u):
    #todo gradient of log_likelihood respect to person i parameters (F_ic)
    n = A.shape[0]
    C = F.shape[1]
    gradient = []
    for c in range(C):
        ans = 0.0
        for j in range(n):
            if(j != u):
                tmp = 0.1
                if(A[u, j] == 1):
                    tmp = d_probability(u, j, c, F) / probability(j, u, F)
                else:
                    tmp = -d_probability(u, j, c, F) / (1 - probability(j, u, F))
                ans += tmp
        gradient.append(ans)
    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* np.reshape(grad, -1) # 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

#testing in two small groups
A=np.random.rand(40,40)
A[0:15,0:25]=A[0:15,0:25]>1- 0.6 # connection prob people with 1 common group
A[0:15,25:40]=A[0:15,25:40]>1-0.1 # connection prob people with no common group
A[15:40,0:25]=A[15:40,25:40]>1-0.7 # connection prob people with 1 common group
A[15:25,15:25]=A[15:25,15:25]>1-0.8 # connection prob people with 2 common group
for i in range(40):
    A[i,i]=0
    for j in range(i):
        A[i,j]=A[j,i]

import matplotlib.pyplot as plt
import networkx as nx
#plt.imshow(A)
delta=np.sqrt(-np.log(1-0.1)) # epsilon=0.1
F=train(A, 2, iterations = 120)
#print(F,delta)
G=nx.from_numpy_array(A)
C=Fdelta # groups members
nx.draw(G,node_color=10*(C[:,0])+20*(C[:,1]))

```



## بخش سوم: گناه بخت پرسشان و دست کوته ماست!

### ○ پرسش تئوری ۹:

هر دو نفر به احتمال یک سوم دور یک میز نشسته اند و اگر دور یک میز باشند یعنی به احتمال ۰,۶ با هم دوست هستند: هر دو نفر سری میز نشسته به احتمال شش سی ام دوست هستند.

هر دو نفر به احتمال دو سوم سری یک میز نشسته اند و به احتمال ۱,۰ یال دارند (دوستند): هر دو نفر سری یک میز نشسته به احتمال دو سی ام دوست هستند.

بنابراین هر دو نفر به احتمال هشت سی ام دوستند.

نتیجه می شود که یال داشتن هر دو فرد یک متغیر برنولی با پارامتر هشت سی ام می باشد. این بدین معنی است که هر یک از درایه های ماتریس از یک توزیع برنولی با پارامتر هشت سی ام می آیند و از باقی درایه ها مستقل اند.

### ○ پرسش تئوری ۱۰:

این پرسش به بیانی دقیق تر همان پرسش قبلی است :

خیر به نظر می رسد که بهتر است درایه ها نسبت به قطر اصلی متقارن باشند مثلا منطقی نیست که علی با رضا دوست باشد ولی رضا با علی نه و درایه های روی قطر نیز ۱ باشند ( هرچند که در پیاده سازی درایه های روی قطر را صفر میگذاریم تا طوقه نداشته باشیم و گراف ساده تر باشد) و دیگر اینکه مثلا فرض کنید علی و حسن سری یک میز هستند ولی رضا سر میز دیگری ...

حال احتمال دوستی علی با رضا منطقی نیست که از همان توزیعی پیروی کند که احتمال دوستی علی با حسن پیروی می کند.

بنابراین ایده زیر را اجرا می کنیم:

فرض کنید نفر یک تا ۵ سر میز اول و ۶ تا ۱۰ سر دومی و ۱۱ تا ۱۵ سر سومی باشند.

10 to 15	1	
5 to 10	2	
1 to 5	3	
	1 to 5	5 to 10

اگر ماتریس مجاورت را مانند بالا در نظر بگیریم منطقی است که بخش های ۱ و ۲ و ۳ از توزیع برنولی با پارامتر ۰,۰ پیروی کنند ولی بقیه بخش ها از توزیع برنولی با پارامتر ۱,۰ پیروی کنند.

### ○ پرسش شبیه سازی ۲:

```

1 import numpy as np
2
3
4 def getNAs(N): # returns N sample A matrices
5     NAs = []
6     for h in range(N):
7         A = np.random.binomial(size = (15, 15), n = 1, p = 0.1)
8         for i in range(15):
9             for j in range(15):
10                 if i < 5 :
11                     if j < 5:
12                         A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
13                 elif i < 10:
14                     if j < 10 and j >= 5:
15                         A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
16                 else:
17                     if j >= 10:
18                         A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
19
20             for i in range(15):
21                 A[i,i]=0
22                 for j in range(15):
23                     A[j][i] = A[i][j]
24         NAs.append(A)
25     return NAs
26 tenAs = getNAs(10)
27

```

تابع `getNAs` مقدار  $N$  به عنوان ورودی میگیرد و  $N$  تا از ماتریس های  $A$  (گفته شده در ۱۰ تیوری را برمیگرداند)

این سوال خروجی ندارد.

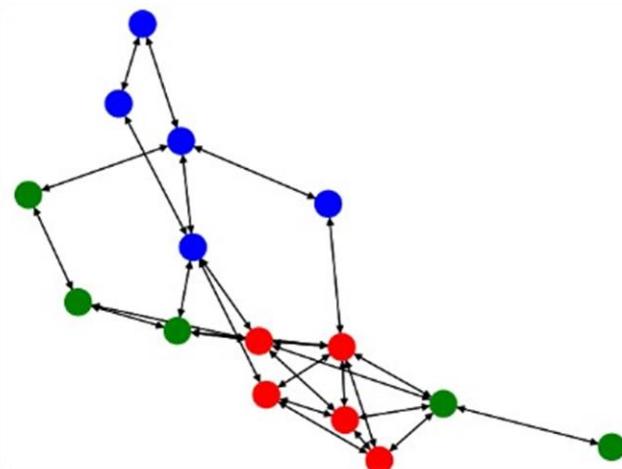
پرسش شبیه سازی ۳: ○

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import networkx as nx
4
5
6 def getNAs(N): # returns N sample A matrices
7     NAs = []
8     for h in range(N):
9         A = np.random.binomial(size = (15, 15), n = 1, p = 0.1)
10        for i in range(15):
11            for j in range(15):
12                if i < 5 :
13                    if j < 5:
14                        A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
15                elif i < 10:
16                    if j < 10 and j >= 5:
17                        A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
18                else:
19                    if j >= 10:
20                        A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
21
22        for i in range(15):
23            A[i,i]=0
24            for j in range(15):
25                A[j][i] = A[i][j]
26        NAs.append(A)
27    return NAs
28
29
30
31 G = nx.DiGraph(getNAs(1)[0])
32 nx.draw(G, node_color = ['green', 'green', 'green', 'green', 'green',
33                                         'red', 'red', 'red', 'red', 'red',
34                                         'blue', 'blue', 'blue', 'blue', 'blue'])
35 plt.show()
36

```

همان تابع نوشته شده در سوال دوم را به ازای ۱ صدا زده و گراف آن را می‌کشد.



پرسش شبیه سازی ۴

```
1 import numpy as np
2
3 def d(z1, z2):
4     acc = 0
5     for i in range(len(z1)):
6         if z1[i] != z2[i]:
7             acc += 1
8     return acc
9
10 z1 = np.array([1, 1, 2, 3, 2, 3, 2, 2, 3, 1, 1, 1, 2, 3, 3])
11 z2 = np.array([1, 2, 3, 3, 2, 3, 2, 2, 3, 1, 1, 1, 1, 3, 3])
12 print(d(z1, z2))
13
```

۲ تا numpy array به عنوان ورودی گرفته شده و خانه هایی از آنها که یکی نیستند را می شمارد ( فاصله همینگ )

خروجی کد در مثال زده شده : ۳

پرسش شبیه سازی ۵

```
1 import numpy as np
2
3 def permutation_helper(0, cur, nex):
4
5     if len(nex) == 0:
6         O.append(cur)
7     for i in range(len(nex)):
8         cur2 = cur.copy()
9         cur2.append(nex[i])
10        nex2 = nex.copy()
11        del nex2[i]
12        permutation_helper(0, cur2, nex2)
13
14 def permutation(n):
15     a = []
16     cur = []
17     nex = []
18     for i in range(1, n + 1):
19         nex.append(i)
20     permutation_helper(a, cur, nex)
21     return a
22
23 def mapper(z, permutation):
24     new_z = []
25     for i in range(len(z)):
26         new_z.append(permutation[z[i] - 1])
27     return np.array(new_z)
28
29 def d(z1, z2):
30     acc = 0
31     for i in range(len(z1)):
32         if z1[i] != z2[i]:
33             acc += 1
34     return acc
35
36 def d_H(z1, z2):
37     all_permutations = permutation(3)
38     ds = []
39     for p in all_permutations:
40         ds.append(d(z1, mapper(z2, p)))
41     return min(ds)
42
43 z1 = np.array([3, 2, 1, 2, 2, 3, 1])
44 z2 = np.array([1, 3, 2, 3, 3, 1, 2])
45 print(d_H(z1, z2))
```

در این سوال باید فاصله همینگ کمینه را حساب کنیم بردار اولی ثابت بوده اما درباره بردار دومی همه حالت های هم نظری با آن را می سازیم کافی است به ازای هر جایگشت از مپ استفاده کنیم اما تولید همه جایگشت ها خود با دوتابع `permutation_helper` و `permutation` انجام می شود محاسبه فاصله همینگ معمولی با تابع سوال قبل انجام شده همه این مقادیر در آرایه ای ریخته شده و مینیمم آن ارایه توسط  $H_d$  ریترن می شود.

### ○ پرسش تئوری ۱۱:

اگر  $Z$  به ما داده شده باشد و این سوال پرسیده شود پاسخ این است که همه درایه های بالا مساوی قطر نسبت به هم مستقل هستند. در غیر این صورت نمی توان پاسخ قاطعانه ای به سوال داد.

### ○ پرسش تئوری ۱۲:

در ماتریس مجاورت درباره دو نفر هم خوشه اگر درایه مربوطه ۱ باشد (باهم دوست باشند) Likelihood را در  $6 \cdot 0$  ضرب و اگر باشد که باهم دوست نیستند در  $4 \cdot 0$  ضرب می کنیم.

و درباره ۲ نفر از خوشه های متفاوت اگر باشد که باهم دوست اند در  $1 \cdot 0$  ضرب و در غیر این صورت در  $9 \cdot 0$  ضرب می کنیم.

در ضمن درایه های روی قطر اصلی اهمیتی ندارند.

### ○ پرسش تئوری ۱۳:

برای محاسبه این بخش می توان ابتدا حاصل ضرب بالا را محاسبه و سپس از آن لگاریتم گرفت و یا می توان این گونه عمل کرد که دوباره ماتریس مجاورت را پرمایش کنیم به ازای هر دو نفر هم خوشه اگر درایه مورد نظر ۱ بود لگاریتم  $6 \cdot 0$  را با حاصل جمع کنیم و اگر صفر بود لگاریتم  $4 \cdot 0$  را جمع کنیم و به ازای هر دو نفر غیر هم خوشه اگر این درایه ۱ بود لگاریتم  $1 \cdot 0$  را جمع و اگر صفر بود لگاریتم  $9 \cdot 0$  را جمع کنیم.

### ○ پرسش شبیه سازی ۶:

```

1 import numpy as np
2
3 def getNAs(N): # returns N sample A matrices
4     NAs = []
5     for h in range(N):
6         A = np.random.binomial(size = (15, 15), n = 1, p = 0.1)
7         for i in range(15):
8             for j in range(15):
9                 if i < 5 :
10                     if j < 5:
11                         A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
12                     elif i < 10:
13                         if j < 10 and j >= 5:
14                             A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
15                         else:
16                             if j >= 10:
17                                 A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
18
19             for i in range(15):
20                 A[i,i]=0
21                 for j in range(15):
22                     A[j][i] = A[i][j]
23         NAs.append(A)
24     return NAs
25
26 def manfi_log_likelihood_A(A, z):
27     likelihood = 0
28     for i in range(15):
29         for j in range(i + 1, 15):
30             if z[i] == z[j]:
31                 if A[i, j] == 0:
32                     likelihood += np.log(0.4)
33                 if A[i, j] == 1:
34                     likelihood += np.log(0.6)
35             if z[i] != z[j]:
36                 if A[i, j] == 0:
37                     likelihood += np.log(0.9)
38                 if A[i, j] == 1:
39                     likelihood += np.log(0.1)
40     return likelihood * (-1)
41 A = getNAs(1)[0]
42 z0 = np.array([1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3])
43
44 np.random.shuffle(z0)
45 print(manfi_log_likelihood_A(A, z0))

```

عملی همان کاری را میکند که در ۱۳ تئوری توضیح دادم.

یک خروجی رندوم از کد: ۶۹,۴۹۹۸۵۷۰۸۰۴۱۲۰۳

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def getAByZ(z):
5     A = np.random.binomial(size = (15, 15), n = 1, p = 0.1)
6     for i in range(15):
7         for j in range(15):
8             if z[i] == z[j]:
9                 A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
10
11    for i in range(15):
12        A[i,i]=0
13        for j in range(15):
14            A[j][i] = A[i][j]
15
16    return A
17
18 def permutation_helper(0, cur, nex):
19
20     if len(nex) == 0:
21         0.append(cur)
22     for i in range(len(nex)):
23         cur2 = cur.copy()
24         cur2.append(nex[i])
25         nex2 = nex.copy()
26         del nex2[i]
27         permutation_helper(0, cur2, nex2)
28
29 def permutation(n):
30     a = []
31     cur = []
32     nex = []
33     for i in range(1, n + 1):
34         nex.append(i)
35     permutation_helper(a, cur, nex)
36
37     return a
```

```
35  def mapper(z, permutation):
36      new_z = []
37      for i in range(len(z)):
38          new_z.append(permutation[z[i] - 1])
39      return np.array(new_z)
40  def d(z1, z2):
41      acc = 0
42      for i in range(len(z1)):
43          if z1[i] != z2[i]:
44              acc += 1
45      return acc
46
47  def d_H(z1, z2):
48      all_permutations = permutation(3)
49      ds = []
50      for p in all_permutations:
51          ds.append(d(z1, mapper(z2, p)))
52      return min(ds)
53  def manfi_log_likelihood_A(A, z):
54      likelihood = 0
55      for i in range(15):
56          for j in range(i + 1, 15):
57              if z[i] == z[j]:
58                  if A[i, j] == 0:
59                      likelihood += np.log(0.4)
60                  if A[i, j] == 1:
61                      likelihood += np.log(0.6)
62              if z[i] != z[j]:
63                  if A[i, j] == 0:
64                      likelihood += np.log(0.9)
65                  if A[i, j] == 1:
66                      likelihood += np.log(0.1)
67      return likelihood * (-1)
68
```

```

69  def train(A, z0, T):
70      likelihoods = []
71      distances = []
72      reference_z = z0.copy()
73      z = z0
74      for t in range(T):
75          i_best = 0
76          j_best = 0
77          likelihood_best = manfi_log_likelihood_A(A, z0)
78          for i in range(15):
79              for j in range(15):
80                  z_test = z.copy()
81                  c = z_test[i]
82                  z_test[i] = z_test[j]
83                  z_test[j] = c
84                  cur_likelihood = manfi_log_likelihood_A(A, z_test)
85                  if cur_likelihood < likelihood_best:
86                      i_best = i
87                      j_best = j
88                      best_likelihood_A = cur_likelihood
89                      c = z[i_best]
90                      z[i_best] = z[j_best]
91                      z[j_best] = c
92                      likelihoods.append(likelihood_best)
93                      distances.append(d_H(z, reference_z))
94                      print(likelihood_best)
95      result = [likelihoods, distances]
96      return result
97
98
99  z0 = np.array([1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3])
100 np.random.shuffle(z0)
101 A = getAByZ(z0)
102 print("real z: ", z0)
103 np.random.shuffle(z0)
104 print("shuffled z: ", z0)
105
106 result = train(A, z0, 20)
107 print(z0)
108 plt.plot(result[0])
109 plt.title("cost/time")
110 plt.show()
111 plt.plot(result[1])
112 plt.title("d_H/time")
113 plt.show()

```

خروجی:

[۳۱۱۲۳۳۱۲۱۳۱۳۲۲۲] :real z

[۱۱۱۳۳۲۳۱۲۲۳۲۱۳۲] :shuffled z

۶۸,۱۱۳۰۷۳۲۱۹۲۹۲۱۳

۶۸,۱۱۳۰۷۳۲۱۹۲۹۲۱۲

۷۰,۰۱۰.۸۷۳۰۳۳۸۴۷۸

۷۰,۰۱۰.۸۷۳۰۳۳۸۴۷۷۴

۷۲,۹۰۰.۸۱۸۳۸۴۸۴.۳۴۲

۷۰,۳۰۰.۰۴۹۴۱۶۲۹۰۹.۴

۵۷,۷۰۰.۴۴۷۷۰۱۴۶۶

۵۷,۷۰۰.۴۴۷۷۰۱۴۷۰۴

۵۰,۱۰۰.۱۱۴۷۹۲.۷۰۰.۲۷

۵۰,۱۰۰.۱۱۴۷۹۲.۷۰۰.۲۷۰

۴۹,۸۹۴۷۳۰۴۲۱۱۸۱۰۱

۴۹,۸۹۴۷۳۰۴۲۱۱۸۱۰

۴۷,۲۹۲.۴۰۷۳۰۷۳۷۱۲

۴۴,۶۸۹۳۰۷.۰.۲۹۲۷۲

۴۲,۰۸۶۶۶۶۳۶۴۸۴۸۳۰

۴۲,۰۸۶۶۶۶۳۶۴۸۴۸۳۰

۴۲,۰۸۶۶۶۶۳۶۴۸۴۸۳۰

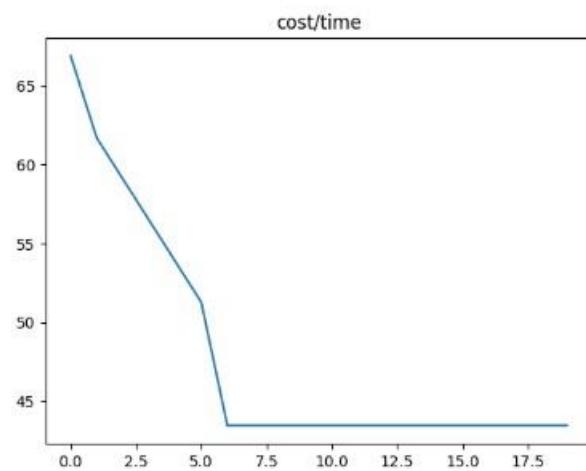
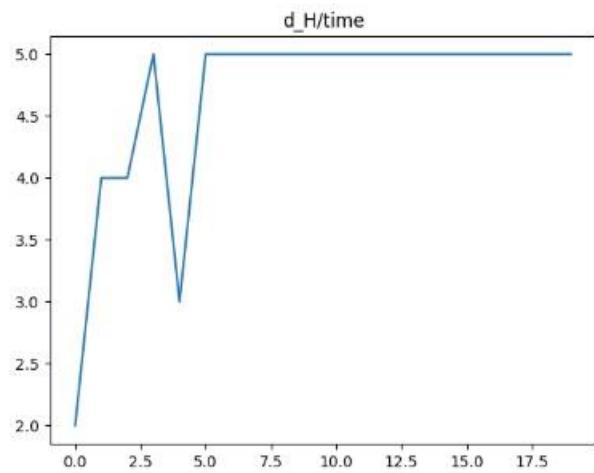
۴۲,۰۸۶۶۶۶۳۶۴۸۴۸۳۰

۴۲,۰۸۶۶۶۶۳۶۴۸۴۸۳۰

۴۲,۰۸۶۶۶۶۳۶۴۸۴۸۳۰

۴۲,۰۸۶۶۶۶۳۶۴۸۴۸۳۰

[۱۳۳۲۱۱۳۲۳۲۳۱۱۲۲]



پرسش شبیه سازی ۸:

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def getAByZ(z):
5      A = np.random.binomial(size = (15, 15), n = 1, p = 0.1)
6      for i in range(15):
7          for j in range(15):
8              if z[i] == z[j]:
9                  A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
10
11     for i in range(15):
12         A[i,i]=0
13         for j in range(15):
14             A[j][i] = A[i][j]
15
16     return A
17
18 def permutation_helper(0, cur, nex):
19
20     if len(nex) == 0:
21         0.append(cur)
22     for i in range(len(nex)):
23         cur2 = cur.copy()
24         cur2.append(nex[i])
25         nex2 = nex.copy()
26         del nex2[i]
27         permutation_helper(0, cur2, nex2)
28
29 def permutation(n):
30     a = []
31     cur = []
32     nex = []
33     for i in range(1, n + 1):
34         nex.append(i)
35     permutation_helper(a, cur, nex)
36     return a
37
38 def mapper(z, permutation):
39     new_z = []
40     for i in range(len(z)):
41         new_z.append(permutation[z[i] - 1])
42     return np.array(new_z)
43
44 def d(z1, z2):
45     acc = 0
46     for i in range(len(z1)):
47         if z1[i] != z2[i]:
48             acc += 1
49
50     return acc
51
52 def d_H(z1, z2):
53     all_permutations = permutation(3)
54     ds = []
55     for p in all_permutations:
56         ds.append(d(z1, mapper(z2, p)))
57
58     return min(ds)

```

```

def manfi_log_likelihood_A(A, z):
    likelihood = 0
    for i in range(15):
        for j in range(i + 1, 15):
            if z[i] == z[j]:
                if A[i, j] == 0:
                    likelihood += np.log(0.4)
                if A[i, j] == 1:
                    likelihood += np.log(0.6)
            if z[i] != z[j]:
                if A[i, j] == 0:
                    likelihood += np.log(0.9)
                if A[i, j] == 1:
                    likelihood += np.log(0.1)
    return likelihood * (-1)

def train(A, z0, T):
    likelihoods = []
    distances = []
    reference_z = z0.copy()
    z = z0
    for t in range(T):
        i_best = 0
        j_best = 0
        likelihood_best = manfi_log_likelihood_A(A, z0)
        for i in range(15):
            for j in range(15):
                z_test = z.copy()
                c = z_test[i]
                z_test[i] = z_test[j]
                z_test[j] = c
                cur_likelihood = manfi_log_likelihood_A(A, z_test)
                if cur_likelihood < likelihood_best:
                    i_best = i
                    j_best = j
                    best_likelihood_A = cur_likelihood
        c = z[i_best]
        z[i_best] = z[j_best]
        z[j_best] = c
        likelihoods.append(likelihood_best)
        distances.append(d_H(z, reference_z))
    result = [likelihoods, distances]
    return result

```

```

100     z0 = np.array([1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3])
101     A = getAByz(z0)
102     print("real z: ", z0)
103     for i in range(10):
104         print("n = ", i + 1)
105         np.random.shuffle(z0)
106         print("first_z : ", z0)
107         train(A, z0, 20)
108         print("last_z : ", z0)
109

```

: خروجی

برای ۱۰ تا  $z$  اولیه الگوریتم را تکرار می‌کنیم صرفا در مرحله‌های اول و آخر هربار الگوریتم نگه داشته شده و مقدار  $z$  نشان داده می‌شود.

real z[۳ ۳ ۳ ۳ ۳ ۲ ۲ ۲ ۲ ۲ ۱ ۱ ۱ ۱ ۱] :

$n = 1$

first\_z[۳ ۳ ۲ ۱ ۱ ۳ ۱ ۲ ۳ ۳ ۱ ۲ ۲ ۲ ۱] :

last\_z[۱ ۳ ۱ ۲ ۳ ۳ ۱ ۲ ۳ ۳ ۱ ۲ ۲ ۲ ۱] :

$n = 2$

first\_z[۱ ۱ ۳ ۲ ۲ ۱ ۳ ۱ ۳ ۲ ۲ ۳ ۱ ۳ ۲] :

last\_z[۲ ۱ ۱ ۱ ۱ ۳ ۲ ۱ ۳ ۳ ۲ ۳ ۲ ۳ ۲] :

$n = 3$

first\_z[۳ ۱ ۲ ۱ ۳ ۲ ۳ ۲ ۱ ۳ ۱ ۳ ۲ ۱ ۲] :

last\_z[۲ ۳ ۱ ۱ ۳ ۲ ۱ ۱ ۲ ۲ ۱ ۳ ۳ ۲ ۲] :

$n = 4$

first\_z[۳ ۱ ۱ ۳ ۳ ۱ ۲ ۲ ۲ ۳ ۳ ۱ ۲ ۱ ۲] :

last\_z[۳ ۲ ۱ ۳ ۲ ۲ ۳ ۳ ۲ ۲ ۳ ۱ ۱ ۱ ۱] :

$n = 5$

first\_z[۲ ۳ ۱ ۱ ۱ ۳ ۱ ۳ ۲ ۳ ۲ ۳ ۲ ۲ ۱] :

last\_z[۱ ۱ ۱ ۳ ۳ ۲ ۳ ۳ ۳ ۲ ۲ ۲ ۱] :

n = 6

first\_z[۱ ۱ ۲ ۱ ۱ ۲ ۳ ۲ ۱ ۲ ۳ ۲ ۳ ۳ ۳] :

last\_z[۲ ۱ ۳ ۲ ۱ ۱ ۲ ۲ ۱ ۱ ۲ ۳ ۳ ۳ ۳] :

n = 7

first\_z[۳ ۱ ۱ ۲ ۲ ۳ ۲ ۳ ۱ ۱ ۲ ۲ ۱ ۳ ۳] :

last\_z[۲ ۱ ۳ ۲ ۱ ۱ ۲ ۲ ۱ ۱ ۲ ۳ ۳ ۳ ۳] :

n = 8

first\_z[۲ ۲ ۳ ۱ ۲ ۱ ۲ ۱ ۱ ۳ ۲ ۳ ۱ ۳ ۳] :

last\_z[۳ ۲ ۲ ۱ ۲ ۱ ۲ ۱ ۱ ۱ ۲ ۳ ۳ ۳ ۳] :

n = 9

first\_z[۱ ۱ ۱ ۲ ۳ ۲ ۳ ۱ ۱ ۲ ۲ ۳ ۳ ۲ ۳] :

last\_z[۱ ۱ ۱ ۱ ۱ ۲ ۲ ۲ ۲ ۲ ۳ ۳ ۳ ۳ ۳] :

n = 10

first\_z[۳ ۱ ۲ ۳ ۳ ۱ ۳ ۲ ۱ ۱ ۳ ۲ ۱ ۲ ۲] :

last\_z[۳ ۱ ۲ ۳ ۱ ۱ ۳ ۳ ۱ ۱ ۳ ۲ ۲ ۲ ۲] :

پرسش شبیه سازی ۹ ○

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def getAByz(z):
5      A = np.random.binomial(size = (15, 15), n = 1, p = 0.1)
6      for i in range(15):
7          for j in range(15):
8              if z[i] == z[j]:
9                  A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
10
11     for i in range(15):
12         A[i,i]=0
13         for j in range(15):
14             A[j][i] = A[i][j]
15
16     return A
17
18  def permutation_helper(0, cur, nex):
19
20      if len(nex) == 0:
21          0.append(cur)
22
23      for i in range(len(nex)):
24          cur2 = cur.copy()
25          cur2.append(nex[i])
26          nex2 = nex.copy()
27          del nex2[i]
28          permutation_helper(0, cur2, nex2)
29
30  def permutation(n):
31      a = []
32      cur = []
33      nex = []
34      for i in range(1, n + 1):
35          nex.append(i)
36      permutation_helper(a, cur, nex)
37
38  def mapper(z, permutation):
39      new_z = []
40      for i in range(len(z)):
41          new_z.append(permutation[z[i] - 1])
42
43  def d(z1, z2):
44      acc = 0
45      for i in range(len(z1)):
46          if z1[i] != z2[i]:
47              acc += 1
48
49  def d_H(z1, z2):
50      all_permutations = permutation(3)
51      ds = []
52      for p in all_permutations:
53          ds.append(d(z1, mapper(z2, p)))
54
55      return min(ds)

```

```

53  def manfi_log_likelihood_A(A, z):
54      likelihood = 0
55      for i in range(15):
56          for j in range(i + 1, 15):
57              if z[i] == z[j]:
58                  if A[i, j] == 0:
59                      likelihood += np.log(0.4)
60                  if A[i, j] == 1:
61                      likelihood += np.log(0.6)
62              if z[i] != z[j]:
63                  if A[i, j] == 0:
64                      likelihood += np.log(0.9)
65                  if A[i, j] == 1:
66                      likelihood += np.log(0.1)
67      return likelihood * (-1)
68
69  def train(A, z0, T):
70      likelihoods = []
71      distances = []
72      reference_z = z0.copy()
73      z = z0
74      for t in range(T):
75          i_best = 0
76          j_best = 0
77          likelihood_best = manfi_log_likelihood_A(A, z0)
78          for i in range(15):
79              for j in range(15):
80                  z_test = z.copy()
81                  c = z_test[i]
82                  z_test[i] = z_test[j]
83                  z_test[j] = c
84                  cur_likelihood = manfi_log_likelihood_A(A, z_test)
85                  if cur_likelihood < likelihood_best:
86                      i_best = i
87                      j_best = j
88                      best_likelihood_A = cur_likelihood
89                  c = z[i_best]
90                  z[i_best] = z[j_best]
91                  z[j_best] = c
92                  likelihoods.append(likelihood_best)
93                  distances.append(d_H(z, reference_z))
94      result = [likelihoods, distances]
95      return result
96

```

```

z0 = np.array([1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3])

print("real z = ", z0)
A = getAByZ(z0)
l0 = manfi_log_likelihood_A(A, z0)
np.random.shuffle(z0)

for i in range(10):
    print()
    print("n = ", i + 1)
    np.random.shuffle(z0)

    print("likelihood: ", manfi_log_likelihood_A(A, z0))
    train(A, z0, 25)
    print("likelihood: ", manfi_log_likelihood_A(A, z0))
    print("likelihood_defference = ", l0 - manfi_log_likelihood_A(A, z0))
    print()

```

اینجا نیز مانند ۸ ، ۱۰ بار الگوریتم را اجرا می کنیم

پرسش شبیه سازی ۱۰ :

بله وجود دارد، معمولا با در این ابعاد برای مساله الگوریتم پس از حدود ۱۵ مرحله خیلی به پاسخ نزدیک می شود و به احتمال بالای ۸۰ درصد در این تعداد قدم چنین حالتی پیدا می شود.

پرسش شبیه سازی ۱۱ :

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def getAByZ(z):
5      A = np.random.binomial(size = (15, 15), n = 1, p = 0.1)
6      for i in range(15):
7          for j in range(15):
8              if z[i] == z[j]:
9                  A[i][j] = np.random.binomial(size = 1, n = 1, p = 0.6)
10
11     for i in range(15):
12         A[i,i]=0
13         for j in range(15):
14             A[j][i] = A[i][j]
15     return A
16
17 def permutation_helper(0, cur, nex):
18
19     if len(nex) == 0:
20         O.append(cur)
21     for i in range(len(nex)):
22         cur2 = cur.copy()
23         cur2.append(nex[i])
24         nex2 = nex.copy()
25         del nex2[i]
26         permutation_helper(0, cur2, nex2)
27
28 def permutation(n):
29     a = []
30     cur = []
31     nex = []
32     for i in range(1, n + 1):
33         nex.append(i)
34     permutation_helper(a, cur, nex)
35     return a
36
37 def mapper(z, permutation):
38     new_z = []
39     for i in range(len(z)):
40         new_z.append(permutation[z[i] - 1])
41     return np.array(new_z)
42
43 def d(z1, z2):
44     acc = 0
45     for i in range(len(z1)):
46         if z1[i] != z2[i]:
47             acc += 1
48     return acc
49
50 def d_H(z1, z2):
51     all_permutations = permutation(3)
52     ds = []
53     for p in all_permutations:
54         ds.append(d(z1, mapper(z2, p)))
55     return min(ds)

```

```

def manfi_log_likelihood_A(A, z):
    likelihood = 0
    for i in range(15):
        for j in range(i + 1, 15):
            if z[i] == z[j]:
                if A[i, j] == 0:
                    likelihood += np.log(0.4)
                if A[i, j] == 1:
                    likelihood += np.log(0.6)
            if z[i] != z[j]:
                if A[i, j] == 0:
                    likelihood += np.log(0.9)
                if A[i, j] == 1:
                    likelihood += np.log(0.1)
    return likelihood * (-1)

def train(A, z0, T):
    likelihoods = []
    distances = []
    reference_z = z0.copy()
    z = z0
    for t in range(T):
        i_best = 0
        j_best = 0
        likelihood_best = manfi_log_likelihood_A(A, z0)
        for i in range(15):
            for j in range(15):
                z_test = z.copy()
                c = z_test[i]
                z_test[i] = z_test[j]
                z_test[j] = c
                cur_likelihood = manfi_log_likelihood_A(A, z_test)
                if cur_likelihood < likelihood_best:
                    i_best = i
                    j_best = j
                    best_likelihood_A = cur_likelihood
        c = z[i_best]
        z[i_best] = z[j_best]
        z[j_best] = c
        likelihoods.append(likelihood_best)
        distances.append(d_H(z, reference_z))
    result = [likelihoods, distances]
    return result

```

```

z0 = np.array([1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3])
A1 = getAByZ(z0)
A2 = getAByZ(z0)
l01 = manfi_log_likelihood_A(A1, z0)
l02 = manfi_log_likelihood_A(A2, z0)
np.random.shuffle(z0)
for i in range(10):
    print()
    print("n = ", i + 1)
    np.random.shuffle(z0)

    print("likelihood: ", manfi_log_likelihood_A(A1, z0))
    train(A1, z0, 25)
    print("likelihood: ", manfi_log_likelihood_A(A1, z0))
    print("likelihood_defference = ", l01 - manfi_log_likelihood_A(A1, z0))
    print()
np.random.shuffle(z0)
for i in range(10):
    print()
    print("n = ", i + 1)
    np.random.shuffle(z0)

    print("likelihood: ", manfi_log_likelihood_A(A2, z0))
    train(A2, z0, 25)
    print("likelihood: ", manfi_log_likelihood_A(A2, z0))
    print("likelihood_defference = ", l02 - manfi_log_likelihood_A(A2, z0))
    print()

```

مانند همان ۹ است صرفاً دوبار تکرار شده است با اعداد گذاشته شده معمولاً در هر دوبار بردار  $z$  به دست می‌آید

یعنی بهترین نتیجه در پایان به آن میرسیم اکثر موقع خود  $z$  است

$n = 1$

likelihood: 72.10254726585647

likelihood: 46.07565041141258

likelihood\_defference = 0.0

$n = 2$

likelihood: 69.49985758041208

likelihood: 48.678340096856964

likelihood\_defference = -2.602689685444382

n = 3

likelihood: 72.10254726585647

likelihood: 46.07565041141258

likelihood\_defference = 0.0

n = 4

likelihood: 66.89716789496768

likelihood: 48.678340096856964

likelihood\_defference = -2.602689685444382

n = 5

likelihood: 72.10254726585647

likelihood: 46.07565041141258

likelihood\_defference = 0.0

n = 6

likelihood: 79.91061632218964

likelihood: 46.07565041141258

likelihood\_defference = 0.0

n = 7

likelihood: 79.91061632218963

likelihood: 48.678340096856964

likelihood\_defference = -2.602689685444382

n = 8

likelihood: 74.70523695130086

likelihood: 46.07565041141258

likelihood\_defference = 0.0

n = 9

likelihood: 77.30792663674525

likelihood: 46.07565041141258

likelihood\_defference = 0.0

n = 10

likelihood: 79.91061632218964

likelihood: 46.07565041141258

likelihood\_defference = 0.0

n = 1

likelihood: 63.719114064619724

likelihood: 37.69221721017586

likelihood\_defference = 2.602689685444389

n = 2

likelihood: 68.9244934355085

likelihood: 37.69221721017586

likelihood\_defference = 2.602689685444389

n = 3

likelihood: 63.71911406461973

likelihood: 37.69221721017587

likelihood\_defference = 2.602689685444382

n = 4

likelihood: 58.51373469373096

likelihood: 37.69221721017587

likelihood\_defference = 2.602689685444382

n = 5

likelihood: 74.12987280639729

likelihood: 37.69221721017586

likelihood\_defference = 2.602689685444389

n = 6

likelihood: 63.71911406461973

likelihood: 37.69221721017586

likelihood\_defference = 2.602689685444389

n = 7

likelihood: 68.92449343550848

likelihood: 37.69221721017586

likelihood\_defference = 2.602689685444389

n = 8

likelihood: 66.3218037500641

likelihood: 37.69221721017586

likelihood\_defference = 2.602689685444389

n = 9

likelihood: 66.32180375006409

likelihood: 37.69221721017586

likelihood\_defference = 2.602689685444389

n = 10

likelihood: 71.52718312095287

likelihood: 37.69221721017587

likelihood\_defference = 2.602689685444382

---

## بخش چهارم: او نه خیال است و نه طیف!

○ پرسش تئوری ۱۴:

$$P_{X \sim N(0,1)} = \begin{cases} 1 & \frac{x_0}{\sigma} \\ 0 & 1 - \frac{x_0}{\sigma} \end{cases}$$

$$CDF = \begin{cases} 0 & x < 0 \\ \frac{1}{2} & 0 < x < 1 \\ 1 & x \geq 1 \end{cases}$$

سؤال ۱۴: یک متغیر تصادفی بروزی طیع.

○ پرسش تئوری ۱۵:

سؤال ۱۵:

حتماً در مردم یک متغیر تصادفی بروزی به صورت  $P_{X \sim N(0,1)} = \frac{1}{2}$  است. بدین‌گاه نیز خواهد راند

$$\begin{bmatrix} 0 & 2 & 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 & 2 & 0 \end{bmatrix}$$

$E[A_{ij}] = \frac{P_{ij}}{6} = \frac{1}{2}$  و ماتریس  $A$  ماتریس همه عرضه می‌شود

○ پرسش تئوری ۱۶:

سؤال ۱۶:

$$\begin{bmatrix} PP & PQ \\ PQ & QQ \end{bmatrix} \rightarrow \det \begin{bmatrix} P-\lambda & Q \\ Q & P+\lambda \end{bmatrix} = 0 \Rightarrow (P-\lambda)^2 - 2(P-\lambda)Q^2 - 2(P-\lambda)Q^2$$

$$- \lambda(Q-P+\lambda)^2 + \lambda^2Q^2 - 2\lambda P^2 + 2\lambda PQ$$

$$= \lambda^2(-1 + 2P - 2Q)(-1 + 2P + 2Q) \rightarrow \lambda_1 = 2P - 2Q, \quad \lambda_2 = 2P + 2Q, \quad \lambda_3 = 0$$

$$\lambda_4 = -2P - 2Q, \quad \lambda_5 = 0, \quad \lambda_6 = 0$$

○ پرسش تئوری ۱۷:

سوال ۱۷

و داین که وضیع دلیل تعداد مرتبه است. تابع  $\rho$  که این تعداد مرتبه است بردار مرتبه های دارد. در این صورت

۲ تعداد مرتبه غیر صفر داشته

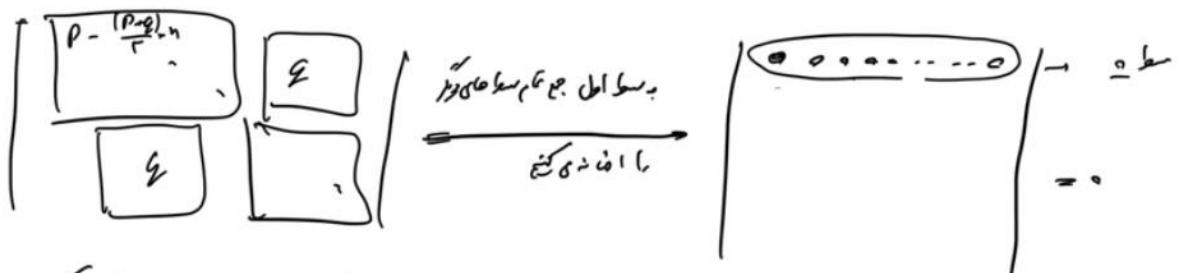
$$\begin{pmatrix} P & Q \\ Q & P \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = 0 \rightarrow P(\overbrace{v_1, \dots, v_{\frac{n}{2}}}^{\infty}) + Q(\overbrace{v_{\frac{n}{2}+1}, \dots, v_n}^{\infty}) = 0 \\ Q(v_1, \dots, v_{\frac{n}{2}}) - P(v_{\frac{n}{2}+1}, \dots, v_n) = 0$$

$$\rightarrow P\alpha - Q\gamma = 0 \quad \text{and} \quad (P-Q)(\alpha - \gamma) = 0, \quad P \neq Q \rightarrow \alpha = \gamma \rightarrow P\alpha - Q\alpha = 0 \\ (P-Q)\alpha = 0 \rightarrow \alpha = 0 = \gamma$$

$$\Rightarrow v_1 = -v_2 - \dots - v_{\frac{n}{2}} \quad \left. \begin{array}{l} \\ \vdots \\ v_{\frac{n}{2}-1} = -v_{\frac{n}{2}+1} - \dots - v_n \end{array} \right\} \rightarrow v_2 = \sum_{i=1}^{\frac{n}{2}} v_i \begin{bmatrix} -1 \\ \vdots \\ 1 \end{bmatrix} + \sum_{i=\frac{n}{2}+1}^n v_i \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix}$$

$$i \rightarrow \begin{bmatrix} -1 \\ \vdots \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} \quad \text{له } n-2 \text{ تعداد مرتبه های داریم. به صفت}$$

و داین تعداد  $(P-Q)$  نیز تعداد مرتبه است. لذا زیرا



پس  $(P-Q)$  نیز می توانیم کارایی کنیم و سطیان را کارایی کنیم.

برور ماتریس دلیل آن نیز هم شد  $\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ \vdots \\ 1 \end{pmatrix}$

پرسش تئوری ۱۸ ○

سوال ۱۸:

$$D_w = \begin{bmatrix} n\frac{P_{qq}}{r} & 0 & 0 & \cdots & 0 \\ 0 & n\frac{P_{qq}}{r} & -P & \cdots & 0 \\ 0 & -P & n\frac{P_{qq}}{r} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & n\frac{P_{qq}}{r} \end{bmatrix}$$

پرسش تئوری ۱۹:

سوال ۱۹:

$$L_{w2} = \begin{bmatrix} n\frac{P_{qq}}{r} - P & -P & -P \\ -P & n\frac{P_{qq}}{r} - P & -P \\ -P & -P & n\frac{P_{qq}}{r} - P \\ -P & -P & -P \end{bmatrix}$$

آن را در ۱- ضرب می کنید

مانند ماتریس سوال ۱۸ می شود

و خارج از دایگوی می شود

$$P - n\frac{P_{qq}}{r} - I = P, \quad P - \frac{P_{qq}}{r}, \quad P - \frac{P_{qq}}{r} \rightarrow \lambda_1, \frac{(P+q)}{r}a, \quad \lambda_2 = 0, \quad \lambda_3 = q, \quad \lambda_4 = -q$$

$$|q|n < \frac{n}{r}P_{qq}$$

برای حذف ریشه آن هم مانند سکونت قبل است دقتی.

پرسش تئوری ۲۰:

سوال ۲۰:

$$\lambda_{-1} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \lambda_1 \rightarrow \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}$$

از توان از کار با بسطیه حاصل شده را با هم دسته بندی کرد

معنی ازراحت کردن بعد درجه درجه یک فقط تراویح گیرند.

①<sub>۰۰۰۱</sub> : (1, -1)

②<sub>۰۰۰۱</sub> : (1, -1)

③<sub>۰۰۰۱</sub> : (1, 1)

④<sub>۰۰۰۱</sub> : (1, 1)

پرسش تئوری ۲۲:

- مرض کرنے کے نقطہ نظر میں داریم وہ خاصہ ملکات آنھا را بے نیک آئندہ، کیجیے۔ (ماتریس) ⑤
- پڑھ ایں " نقطہ نظر میں تکلیف جو ہم ہیں۔ یہ میں سمجھ کر رہا ہم وہ ہستہ اگر نیک آنھا رکھے خدا رکھا ہے پوچھتا ہے۔"
- ماتریس مبارکت آنہ ما وہ نظر کیجیے۔
- لاہیں آنے والے سبھ کہنے
- ۴۷ بیوار مریڑہ اولن را پہتے ہو آریں (۴ عالی ترکوں میں مطلب ٹھیک ہے اسے)
- ماتریس ایں ۴۷ بیوار مانگیہ مدد ہم باسم ۰
- زلاھا را درست ہے ۰ ہی کیجیے۔
- ۶۰ الگریت دیکھنا طور پر، لا را بے فرد ہی ۶۱، ۶۲ نہیں ہو سکتے۔

## بخش پنجم: من از دیار حبیبم نه از بلاد غریب!

در این قسمت توجه خود را به یک خوشه خاص معطوف می‌کنیم. این خوشه با یک گراف  $n$  راسی مدل می‌شود که به احتمال  $p$  بین دو راس آن یال وجود دارد. به عبارتی به احتمال  $p$  دو فرد در این خوشه دارای سلیقه‌ی مشابه هم هستند.

### ○ پرسش تئوری ۲۳:

در این گراف به طور کلی  $C(n, 2) = \frac{n(n-1)}{2}$  یال داریم. تعداد یال‌ها را برای اختصار در نوشتمن برابر  $X$  نامیم. یک  $X$  بیتی (مبنای ۲) در نظر بگیرید که هر بیت آن در صورت یک بودن یک رابطه‌ی هم سلیقگی را نشان دهد. در کل  $m$  رابطه‌ی هم سلیقگی داریم که یعنی باید  $m$  بیت برابر ۱ باشند. احتمال خواسته شده مسئله هم ارز این است که  $m$  بیت مورد نظر برابر ۱ و بقیه برابر صفر باشند. بنابراین احتمال خواسته شده برابر خواهد بود با:

$$\text{Probability} = p^m(1-p)^{X-m} = p^m(1-p)^{n(n-1)/2-m}$$

### ○ پرسش تئوری ۲۴:

برای مشخص کردن دقیقا  $m$  رابطه‌ی هم سلیقگی  $C(X, m)$  حالت داریم که تنها یکی از این حالات همان حالت مطلوب ما خواهد بود. بنابراین خواهیم داشت:

$$\text{Probability} = \frac{1}{C(X, m)} = \frac{m! \left(\frac{n(n-1)}{2} - m\right)!}{\left(\frac{n(n-1)}{2}\right)!}$$

### ○ پرسش تئوری ۲۵:

برای محاسبه‌ی احتمال اینکه دقیقا ۲۰ درصد روابط هم سلیقگی را به درستی تعیین کنیم، ابتدا ۲۰ درصدی را که به درستی تعیین شده‌اند را مشخص می‌کنیم که برای آن  $C(m, 0.2m)$  حالت داریم. (در تمام این بخش فرض شده که  $0.2m$  عددی طبیعی است). حال به احتمال  $p^{0.2m}(1-p)^{0.8m}$  دقیقا این ۲۰ درصد انتخاب شده به درستی تعیین شده‌اند. همچنین برای  $m - X$  یال دیگر نیز هر دو حالت یک و یا صفر بودن درایه مورد نظر آن‌ها در ماتریس سوال، در حالات مطلوب مورد پذیرش است. بنابراین خواهیم داشت:

$$\begin{aligned} \text{Probability} &= C(m, 0.2m)p^{0.2m}(1-p)^{0.8m} \\ &= \frac{m!}{(0.2m)!(0.8m)!} p^{0.2m}(1-p)^{0.8m} \end{aligned}$$

### ○ پرسش شبیه سازی ۱۷:

```

1 import numpy as np
2
3 counts = []
4 n = 1000
5 p = 0.0034
6 repetitions = 10
7
8 def mean(l):
9     sum = 0
10    for i in range(0, len(l)):
11        sum += l[i]
12    return sum / len(l)
13
14
15 for i in range(0, repetitions):
16    counts.append(sum(np.random.binomial(1, p, n*(n-1)//2)))
17
18 print(counts)
19 print(mean(counts))

```

خروجی:

[1702, 1736, 1692, 1693, 1713, 1646, 1752, 1654, 1679, 1675]  
1694

همانطور که مشاهده می شود این مقدار میانگین با  $m$  داده شده بسیار متفاوت است.

#### پرسش تئوری ۲۶ ○

دقت کنید که برای هر یال گراف مورد نظر می توانیم یک متغیر تصادفی برنولی با پارامتر  $p$  در نظر بگیریم. بنابراین تعداد روابط هم سلیقگی که همان تعداد یال های این گراف است، یک متغیر تصادفی باینومیال خواهد بود. این متغیر را  $\gamma$  نامیده و امیدریاضی آن را حساب می کنیم که همان مقدار متوسط خواسته شده است: (دقت کنید که  $X$  تعداد کل یال های گراف است)

$$E[Y] = Xp = \frac{n(n-1)}{2}p = 1698.3$$

حال برای اینکه این مقدار خواسته شده دقیقا برابر  $m$  باشد، باید رابطه زیر برقرار باشد:

$$E[Y] = m \rightarrow \frac{n(n-1)}{2}p = m$$

اگر بخواهیم که مقدار میانگین محاسبه شده تقریبا (با خطای حداقل ۵ درصد) با مقدار  $m$  برابر شود، باید نامساوی زیر برقرار باشد:

$$0.95m \leq E[Y] \leq 1.05m \rightarrow 0.95m \leq \frac{n(n-1)}{2}p \leq 1.05m$$

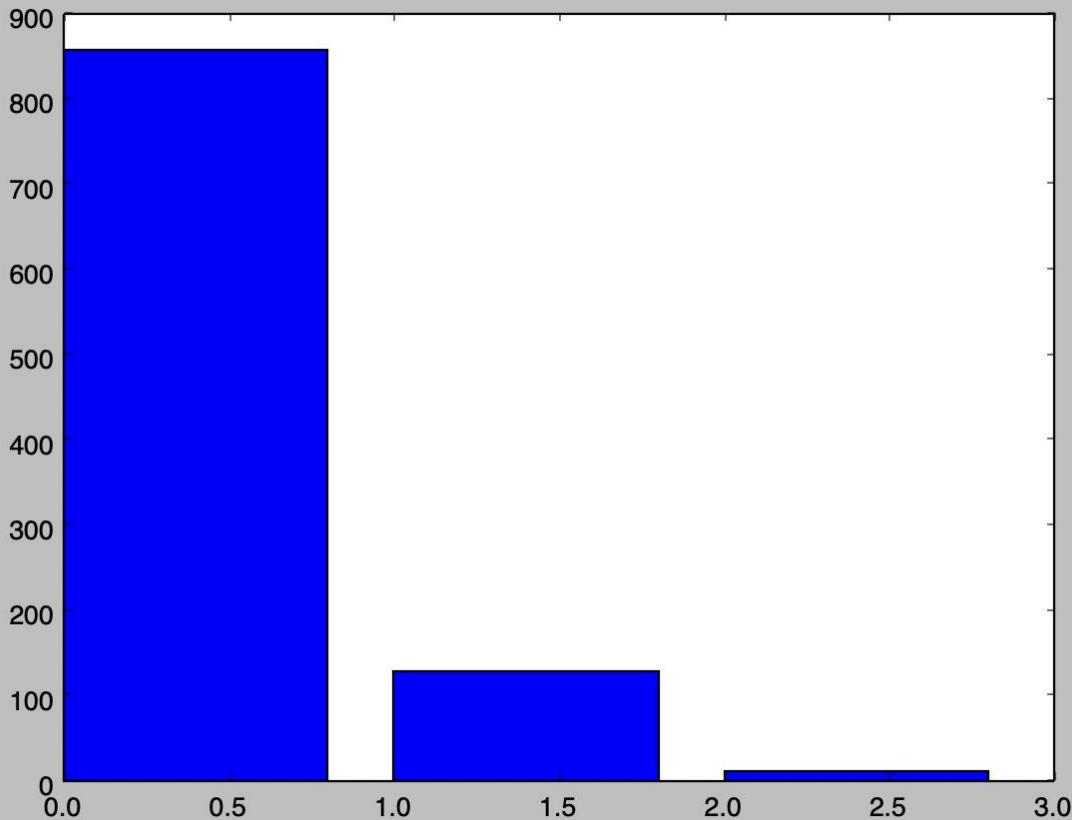
#### پرسش شبیه سازی ۱۸ ○

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  sameColorsCount = []
5  n = 1000
6  p = 0.00016
7  repetitions = 10
8
9  x= []
10 for i in range(0, n):
11     x.append(i)
12 y = [0 for i in range(n)]
13
14 def mean(l):
15     sum = 0
16     for i in range(0, len(l)):
17         sum += l[i]
18     return sum / len(l)
19
20 def generateGraph():
21     graph = [[0 for i in range(n)] for j in range(n)]
22     for i in range(0, n-1):
23         for j in range(i + 1, n):
24             graph[i][j] = np.random.binomial(1, p, 1)
25             graph[j][i] = graph[i][j]
26     return graph
27
28 def calculateAverageDegree(graph):
29     s = 0
30     for i in range(0, n):
31         c = sum(graph[i])
32         s += c
33         y[int(c)] += 1
34     return s / n
35
```

```
36 ✓ def countSameColors(graph, l):
37     count = 0
38 ✓     for i in range(0, n):
39 ✓         if(sum(graph[i]) > l):
40             count+=1
41     return count
42
43 ✓ for i in range(0, repetitions):
44     graph = generateGraph()
45     l = calculateAverageDegree(graph)
46     sameColorsCount.append(countSameColors(graph, l))
47
48 print(mean(sameColorsCount))
49
50 ✓ for i in range(n):
51     y[i] /= repetitions
52
53 plt.bar(x, y)
54 plt.show()
55
56
```

خروجی:

141



### پرسش تئوری ۲۷:

تعداد افراد هم سلیقه با یک فرد را با متغیر تصادفی  $A_v$  (که  $v$  راس گراف است) و تعداد کل هم سلیقه‌گرها را با متغیر تصادفی  $Y$  نشان می‌دهیم. بنابراین مقدار متوسط  $A_v$  مطلوب سوال است که داریم:

$$\sum_{v \in V} A_v = \sum_{v \in V} \deg(v) = 2Y$$

حال از دو طرف معادله بالا امیدریاضی می‌گیریم. دقت کنید که امیدریاضی  $A_v$  ها باهم برابر است:

$$nE[A_v] = 2E[Y] \rightarrow E[A_v] = \frac{2}{n}E[Y] = (n-1)p = 0.15984$$

### پرسش تئوری ۲۸:

ابتدا احتمال اینکه یک نفر هم رنگ باشد را محاسبه می‌کنیم. واضح است داریم:

$$Probability = \sum_{i=[L]+1}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-1-i}$$

در اینجا از بخش قبل داریم:

$$L = 0.15984 \rightarrow \lfloor L \rfloor + 1 = 1$$

حال از بسط دو جمله‌ای می‌دانیم:

$$\sum_{i=0}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-1-i} = (1-p+p)^{n-1} = 1$$

پس برای احتمال خواسته شده داریم:

$$Probability = 1 - \binom{n-1}{0} p^0 (1-p)^{n-1-0} = 1 - (1-p)^{n-1} = 0.14773$$

حال به هم رنگ بودن هر فرد یک *indicator variable* نسبت می‌دهیم: ( $I_i$  یک است اگر فرد  $i$  ام همنگ باشد).

$$I_i = \begin{cases} 1 & p = 0.14773 \\ 0 & q = 1 - p \end{cases}$$

حال تعداد افراد هم رنگ را با متغیر تصادفی  $B$  نمایش می‌دهیم و از خطی بودن امید ریاضی استفاده می‌کنیم. داریم:

$$B = \sum_{i=1}^n I_i \rightarrow E[B] = \sum_{i=1}^n E[I_i] = nE[I_i] = n \times 0.14773 = 147.73$$

پرسش شبیه سازی ۱۹ ○

```
1 import numpy as np
2
3 n = 3000
4 p = 0.01
5 repetitions = 5
6
7 def mean(l):
8     sum = 0
9     for i in range(0, len(l)):
10        sum += l[i]
11    return sum / len(l)
12
13 def generateGraph():
14     graph = [[0 for i in range(n)] for j in range(n)]
15     for i in range(0, n-1):
16         for j in range(i + 1, n):
17             graph[i][j] = np.random.binomial(1, p, 1)[0]
18             graph[j][i] = graph[i][j]
19     return graph
20
21 transferProperties = []
22 chainProperties = []
23
```

```

23
24     for i in range(0, repetitions):
25         graph = generateGraph()
26         graph = np.asarray(graph)
27         graph2 = graph @ graph
28         transfer = 0
29         chain = 0
30         for i in range(0, n):
31             for j in range(0, n):
32                 if(i != j):
33                     if(graph2[i][j] > 0 and graph[i][j] > 0):
34                         transfer += graph2[i][j]
35                     elif(graph2[i][j] > 0):
36                         chain += graph2[i][j]
37         transferProperties.append(transfer / 6)
38         chainProperties.append(chain / 2)
39
40     print(mean(transferProperties))
41     print(mean(chainProperties))
42

```

خروجی:

**4492.4**  
**1330363.6**

پرسش تئوری ۲۹ ○

یک سه تایی  $A$ ،  $B$  و  $C$  از رئوس را در نظر بگیرید. شکل های زیر نشان دهنده وضعیت یال های بین این سه راس در هر یک از ویژگی هاست:



ایندا امید ریاضی تعداد روابط تراگذاری را محاسبه می کنیم. برای اینکار یک *Indicator variable* برای هر سه تایی از رئوس (سه تایی  $i$ ) تعریف می کنیم:

$$I_i = \begin{cases} 1 & \text{if the triple has transfer property,} \\ 0 & \text{otherwise} \end{cases} \quad \text{probability} = p^3$$

$$\rightarrow E[\#transfer\ property] = \sum_{i=1}^{\binom{n}{3}} E[I_i] = \binom{n}{3} p^3$$

حال برای محاسبه تعداد روابط زنجیره‌ای نیز مشابها *indicator variable* زیر را تعریف می‌کنیم:

$$I_i = \begin{cases} 1 & \text{if the triple has chain property,} \\ 0 & \text{otherwise} \end{cases} \quad \text{probability} = 3p^2(1-p)$$

$$\rightarrow E[\#chain\ property] = \sum_{i=1}^{\binom{n}{3}} E[I_i] = 3 \binom{n}{3} p^2(1-p)$$

### پرسش تئوری ۳۰

طبق رابطه‌ی بخش قبل نسبت داده شده برابر است با:

$$\frac{p^3}{p^3 + 3p^2(1-p)}$$

برای  $n$  و  $p$  پرسش شبیه سازی ۱۹ این کسر برابر ۰.۰۰۳۳۵۵۷۰۴۶۹ می‌شود که با مقدار شبیه سازی شده همخوانی دارد.

### پرسش شبیه سازی ۲۰

```

1 import numpy as np
2
3 n = 1000
4 p = 0.003
5 repetitions = 100
6
7 def mean(l):
8     sum = 0
9     for i in range(0, len(l)):
10        sum += l[i]
11    return sum / len(l)
12
13 def generateGraph():
14     graph = [[0 for i in range(n)] for j in range(n)]
15     for i in range(0, n-1):
16         for j in range(i + 1, n):
17             graph[i][j] = np.random.binomial(1, p, 1)
18             graph[j][i] = graph[i][j]
19     return graph
20
21 counts = []
22 for k in range(0, repetitions):
23
24     graph = generateGraph()
25
26     l = []
27     for i in range(0, n):
28         if(graph[0][i]==1):
29             l.append(i)
30
31     count = 0
32     for i in range(len(l)-1):
33         for j in range(i+1, len(l)):
34             if(graph[i][j]==1):
35                 count+=1
36     counts.append(count)
37
38 print(mean(counts))
39

```

خروجی:

**0.04**

پرسش تئوری ۳۱ ○

یک راس دلخواه را در نظر بگیرید. برای این راس تعداد مثلث‌هایی که می‌سازد را در نظر بگیرید. به ازای هر مثلث ساخته شده یک رابطه‌ی هم سلیقه‌های هم سلیقه‌های این راس ساخته می‌شود.

پس پاسخ مسئله همان امید ریاضی تعداد مثلثهای یک راس خواهد بود. تعداد مثلثهای هر راس را با متغیر تصادفی  $X_i$  نشان می‌دهیم. جمع کل  $X_i$  ها ۳ برابر تعداد کل مثلثهای است. (چراکه هر مثلث یک بار به ازای هر راس خود شمرده می‌شود). تعداد مثلثهای را با متغیر تصادفی  $T$  نشان می‌دهیم. حال از رابطه‌ی گفته شده امید ریاضی می‌گیریم و داریم:

$$\sum_{i=1}^n X_i = 3T \rightarrow nE[X_i] = 3 \binom{n}{3} p^3 \rightarrow E[X_i] = \frac{3}{n} \binom{n}{3} p^3$$

پرسش شبیه سازی ۲۱ ○

```

1   import numpy as np
2
3   class Graph:
4
5       adj = []
6
7       def __init__(self, v):
8
9           self.v = v
10          Graph.adj = [[0 for i in range(v)]
11                         | | | | for j in range(v)]]
12
13      def addEdge(self, start, e):
14          Graph.adj[start][e] = 1
15          Graph.adj[e][start] = 1
16
17      def BFS(self, start):
18          s = 0
19          visited = [False] * self.v
20          q = [start]
21          visited[start] = True
22          dist = visited = [0] * self.v
23
24          while q:
25              vis = q[0]
26              q.pop(0)
27
28              for i in range(self.v):
29                  if (Graph.adj[vis][i] == 1 and
30                      | (not visited[i])):
31                      q.append(i)
32                      visited[i] = True
33                      dist[i] = dist[vis] + 1
34
35              for i in range(0, n):
36                  s += dist[i]
37
38      return s
39
40

```

```
40 n = 1000
41 p = 0.0033
42 graph = Graph(n)
43
44 def generateGraph():
45     for i in range(0, n-1):
46         for j in range(i + 1, n):
47             if(np.random.binomial(1, p, 1) == 1):
48                 graph.addEdge(i, j)
49
50 generateGraph()
51
52 distancesSum = 0
53 for i in range(0, n):
54     distancesSum += graph.BFS(i)
55
56 print(distancesSum / (n * (n-1)))
```

خروجی:

5.388174174174174

پرسش شبیه سازی ۲۲ ○

```
1 import numpy as np
2
3 class Graph:
4
5     adj = []
6
7     def __init__(self, v):
8
9         self.v = v
10        Graph.adj = [[0 for i in range(v)]
11                     | | | | for j in range(v)]]
12
13    def addEdge(self, start, e):
14        Graph.adj[start][e] = 1
15        Graph.adj[e][start] = 1
16
17    def BFS(self, start):
18        s = 0
19        visited = [False] * self.v
20        q = [start]
21        visited[start] = True
22        dist = visited = [0] * self.v
23
24        while q:
25            vis = q[0]
26            q.pop(0)
27
28            for i in range(self.v):
29                if (Graph.adj[vis][i] == 1 and
30                    | (not visited[i])):
31                    q.append(i)
32                    visited[i] = True
33                    dist[i] = dist[vis] + 1
34
35            for i in range(0, n):
36                if(dist[i] > s):
37                    | | s = dist[i]
38
39        return s
40
```

```

41     n = 50
42     p = 0.34
43     repetitions = 100
44     maxDistances = []
45
46     def generateGraph(graph):
47         for i in range(0, n-1):
48             for j in range(i + 1, n):
49                 if(np.random.binomial(1, p, 1) == 1):
50                     graph.addEdge(i, j)
51
52     def mean(l):
53         sum = 0
54         for i in range(0, len(l)):
55             sum += l[i]
56         return sum / len(l)
57
58     for i in range(0, repetitions):
59         graph = Graph(n)
60         generateGraph(graph)
61         max = 0
62         for j in range(0, n):
63             dist = graph.BFS(j)
64             if(max < dist):
65                 max = dist
66         maxDistances.append(max)
67
68     print(mean(maxDistances))
69
70
71

```

خروجی:

2.72

پرسش شبیه سازی: ۲۳ ○



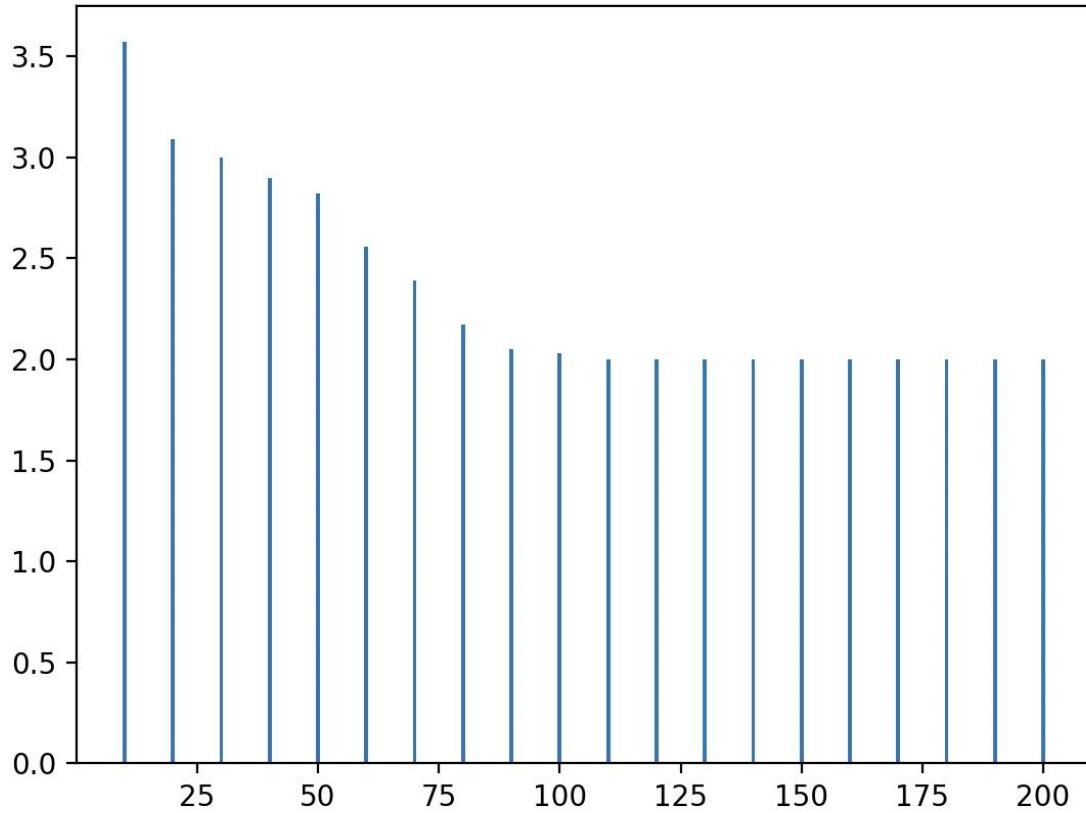
```

42     n = []
43     for i in range(0, 20):
44         n.append((i + 1)*10)
45     p = 0.34
46     repetitions = 100
47     means = []
48
49     def generateGraph(graph, n):
50         for i in range(0, n-1):
51             for j in range(i + 1, n):
52                 if(np.random.binomial(1, p, 1) == 1):
53                     graph.addEdge(i, j)
54
55     def mean(l):
56         sum = 0
57         for i in range(0, len(l)):
58             sum += l[i]
59         return sum / len(l)
60
61     for i in range(0, 20):
62         maxDistances = []
63         for k in range(0, repetitions):
64             graph = Graph(n[i])
65             generateGraph(graph, n[i])
66             max = 0
67             for j in range(0, n[i]):
68                 dist = graph.BFS(j)
69                 if(max < dist):
70                     max = dist
71             maxDistances.append(max)
72         means.append(mean(maxDistances))
73
74     print(means)
75     plt.bar(n,means)
76     plt.show()

```

خروجی:

`[3.57, 3.09, 3.0, 2.9, 2.82, 2.56, 2.39, 2.17, 2.05, 2.03, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]`



این نمودار نزولی است و همانطور که نشان داده شده در  $n$  بزرگ به ۲ میل می‌کند.

پرسش تئوری ۳۲:

به جز رئوس  $u$  و  $v$ ، هر راس دیگر به احتمال  $p^2$  با هر دو راس یال دارد. پس راس دیگر به احتمال  $1 - p^2$  همسایه مشترک آن‌ها نیست. پس داریم:

$$P(I_{u,v}) = (1 - p^2)^{n-2}$$

پرسش تئوری ۳۳:

مطابق *indicator variable* در بخش قبل داریم:

$$X_n = \sum_{u,v \text{ are graph vertices}} I_{u,v}$$

پس:

$$E[X_n] = \binom{n}{2} E[I_{u,v}] = \binom{n}{2} (1 - p^2)^{n-2} = \frac{n(n-1)}{2} (1 - p^2)^{n-2}$$

پرسش تئوری ۳۴:

از نامساوی مارکوف داریم:

$$P(X_n \geq 1) \leq \frac{E[X_n]}{1} = \frac{n(n-1)}{2} (1 - p^2)^{n-2}$$

حال  $n$  را به بینهایت می‌دهیم:

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n(n-1)}{2} (1-p^2)^{n-2} \\ = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{n^2 - n}{e^{(n-2) \ln\left(\frac{1}{1-p^2}\right)}} \\ = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{2n-1}{\ln\left(\frac{1}{1-p^2}\right) * e^{(n-2) \ln\left(\frac{1}{1-p^2}\right)}} \\ = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{2}{\left(\ln\left(\frac{1}{1-p^2}\right)\right)^2 * e^{(n-2) \ln\left(\frac{1}{1-p^2}\right)}} = 0 \end{aligned}$$

پس در حد  $n$  به سمت بی‌نهایت داریم:

$$P(X_n \geq 1) \leq 0$$

○ پرسش تئوری ۳۵:

با توجه به نامساوی آخر در  $n$  های بزرگ و با توجه به اصول کولموگروف داریم:

$$P(X_n \geq 1) = 0 \rightarrow P(X_n = 0) = 1$$

پس می‌توان گفت که هر دو راسی دارای همسایه‌ی مشترک هستند. بنابراین قطرگراف حداکثر برابر ۲ است و به  $p$  وابسته نیست. این نتیجه با نتیجه‌ی حاصل از شبیه‌سازی کاملاً همخوانی دارد، چراکه در آن جا هم با افزایش  $n$  قطرگراف به ۲ میل می‌کرد.

○ پرسش شبیه‌سازی ۲۴:

```

1 import numpy as np
2
3 n = 100
4 p = 0.34
5 repetitions = 100
6
7 def mean(l):
8     sum = 0
9     for i in range(0, len(l)):
10        sum += l[i]
11    return sum / len(l)
12
13 def generateGraph():
14     graph = [[0 for i in range(n)] for j in range(n)]
15     for i in range(0, n-1):
16         for j in range(i + 1, n):
17             graph[i][j] = np.random.binomial(1, p, 1)[0]
18             graph[j][i] = graph[i][j]
19     return graph
20
21 transferProperties = []
22
23 for i in range(0, repetitions):
24     graph = generateGraph()
25     graph = np.asarray(graph)
26     graph2 = graph @ graph
27     transfer = 0
28     for i in range(0, n):
29         for j in range(0, n):
30             if(i != j):
31                 if(graph2[i][j] > 0 and graph[i][j] > 0):
32                     transfer += graph2[i][j]
33     transferProperties.append(transfer / 6)
34
35 print(mean(transferProperties))
36
37

```

خروجی:

**6330.56**

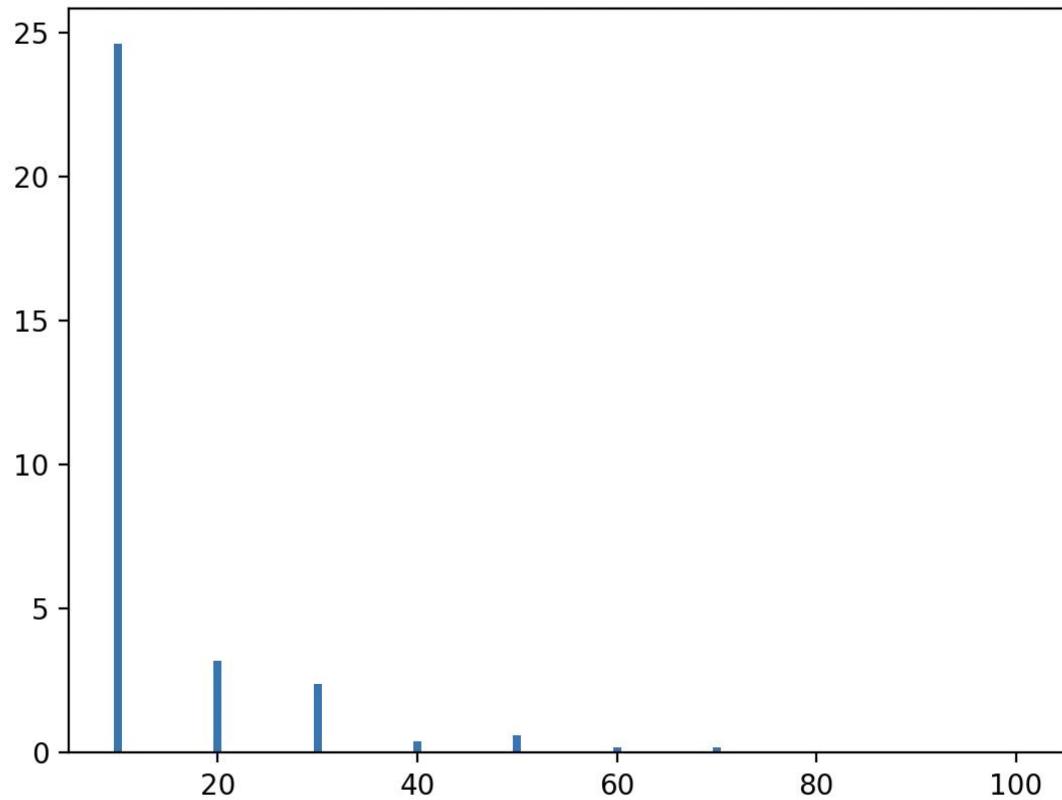
پرسش شبیه سازی ۲۵ ○

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n = []
5 for i in range(0, 10):
6     n.append((i + 1)*10)
7
8 repetitions = 5
9 means = []
10
11 def mean(l):
12     sum = 0
13     for i in range(0, len(l)):
14         sum += l[i]
15     return sum / len(l)
16
17 def generateGraph(n, p):
18     graph = [[0 for i in range(n)] for j in range(n)]
19     for i in range(0, n-1):
20         for j in range(i + 1, n):
21             graph[i][j] = np.random.binomial(1, p, 1)[0]
22             graph[j][i] = graph[i][j]
23     return graph
24
25 for k in range(0, 10):
26     transferProperties = []
27     for i in range(0, repetitions):
28         graph = generateGraph(n[k], 60 / (n[k] * n[k]))
29         graph = np.asarray(graph)
30         graph2 = graph @ graph
31         transfer = 0
32         for i in range(0, n[k]):
33             for j in range(0, n[k]):
34                 if(i != j):
35                     if(graph2[i][j] > 0 and graph[i][j] > 0):
36                         transfer += graph2[i][j]
37         transferProperties.append(transfer / 6)
38     means.append(mean(transferProperties))
39
40 plt.bar(n,means)
41 plt.show()

```

خروجی:



با افزایش  $n$  این مقدار به صفر میل می‌کند که دلیل آن را می‌توان به این ربط داد که  $p$  به صفر میل کرده و عملاً تعداد مثلث‌ها به صفر میل می‌کند.

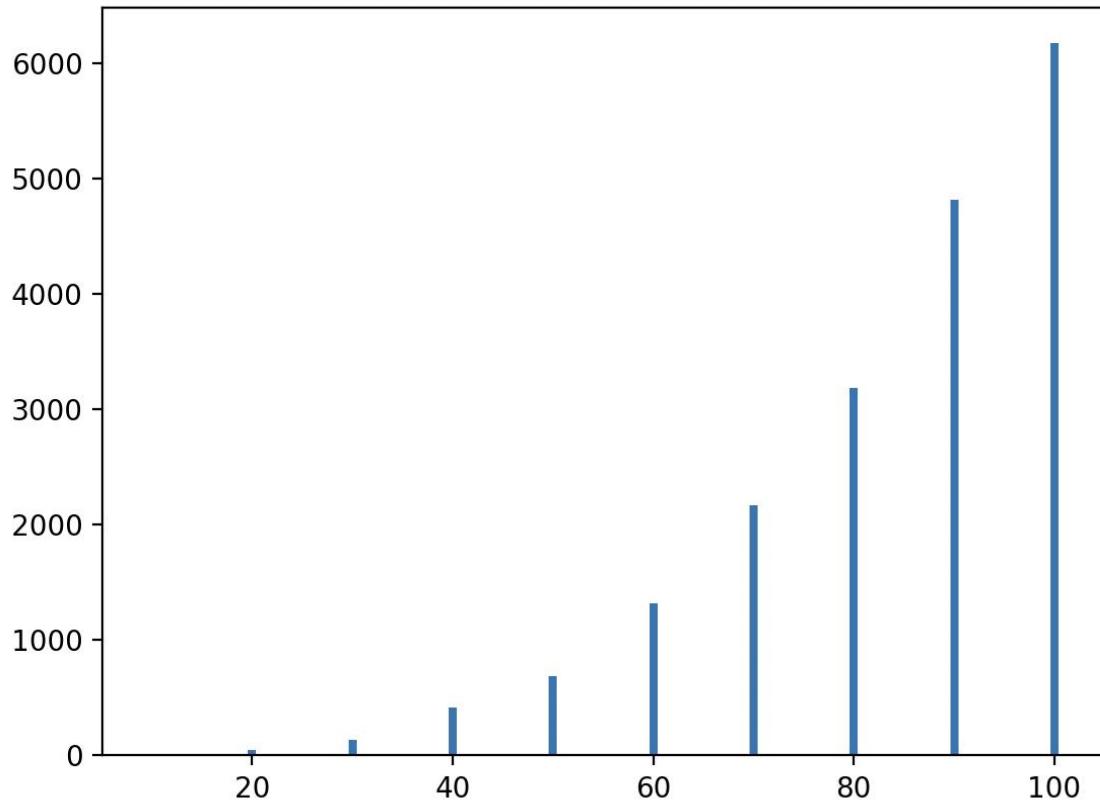
○ پرسش شبیه سازی ۲۶:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n = []
5 for i in range(0, 10):
6     n.append((i + 1)*10)
7
8 repetitions = 5
9 means = []
10
11 def mean(l):
12     sum = 0
13     for i in range(0, len(l)):
14         sum += l[i]
15     return sum / len(l)
16
17 def generateGraph(n, p):
18     graph = [[0 for i in range(n)] for j in range(n)]
19     for i in range(0, n-1):
20         for j in range(i + 1, n):
21             graph[i][j] = np.random.binomial(1, p, 1)[0]
22             graph[j][i] = graph[i][j]
23     return graph
24
25 for k in range(0, 10):
26     transferProperties = []
27     for i in range(0, repetitions):
28         graph = generateGraph(n[k], 0.34)
29         graph = np.asarray(graph)
30         graph2 = graph @ graph
31         transfer = 0
32         for i in range(0, n[k]):
33             for j in range(0, n[k]):
34                 if(i != j):
35                     if(graph2[i][j] > 0 and graph[i][j] > 0):
36                         transfer += graph2[i][j]
37                     transferProperties.append(transfer / 6)
38     means.append(mean(transferProperties))
39
40 plt.bar(n,means)
41 plt.show()
42

```

خروجی:



در این قسمت  $n$  ثابت بوده و مقدار خواسته شده صعودی چراکه عملاً فقط به  $n$  وابسته بوده و با افزایش  $n$ ، افزایش می‌یابد. پس به عدد خاصی میل نمی‌کند.

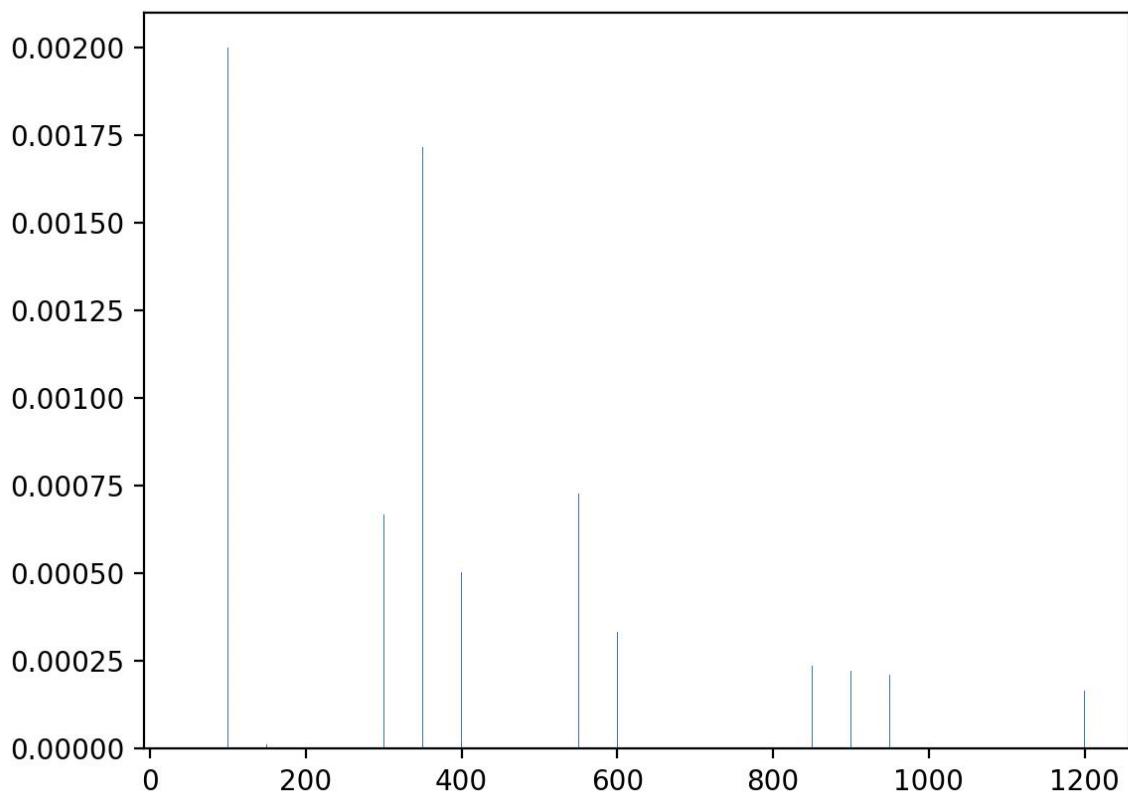
○ پرسش شبیه سازی ۲۷:

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  n = []
5  for i in range(0, 24):
6      n.append((i + 1)*50)
7
8  repetitions = 5
9  means = []
10
11 def mean(l):
12     sum = 0
13     for i in range(0, len(l)):
14         sum += l[i]
15     return sum / len(l)
16
17 def generateGraph(n, p):
18     graph = [[0 for i in range(n)] for j in range(n)]
19     for i in range(0, n-1):
20         for j in range(i + 1, n):
21             graph[i][j] = np.random.binomial(1, p, 1)[0]
22             graph[j][i] = graph[i][j]
23     return graph
24
25 for k in range(0, 24):
26     transferProperties = []
27     for i in range(0, repetitions):
28         graph = generateGraph(n[k], 1 / n[k])
29         graph = np.asarray(graph)
30         graph2 = graph @ graph
31         transfer = 0
32         for i in range(0, n[k]):
33             for j in range(0, n[k]):
34                 if(i != j):
35                     if(graph2[i][j] > 0 and graph[i][j] > 0):
36                         transfer += graph2[i][j]
37         transferProperties.append(transfer / 6)
38     means.append(mean(transferProperties))
39
40 for i in range(1, len(means)):
41     means[i] += means[i-1]
42     means[i] /= n[i]
43
44 plt.bar(n,means)
45 plt.show()

```

:خروجی:



طبق نمودار رسم شده به مقدار خاصی میل نمی‌کند.

## بخش ششم: سل المchanع رکبا تهیم فی الفلوات!

در این قسمت به بررسی الگوریتم قدم زدن تصادفی خواهیم پرداخت.

### ○ پرسش تئوری ۳۶:

احتمال انتقال از  $i$  به  $j$  وجود دارد اگر بین این دو راس یال وجود داشته باشد. از جایی که فرایند قدم زدن به صورت یکنواخت و تصادفی است، داریم:

$$P_{i,j} = \begin{cases} \frac{1}{d_i} & \text{if there is an edge between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

### ○ پرسش تئوری ۳۷:

برای درایه‌های ماتریس‌های داده شده داریم:

$$A_{i,j} = d_i P_{i,j}$$

پس خواهیم داشت:

$$A = DP \rightarrow P = D^{-1}A$$

### ○ پرسش تئوری ۳۸:

مطابق ضرب ماتریس‌ها خواهیم داشت:

$$[P_{i,j}^2] = \sum_{k=1}^n P_{i,k} P_{k,j}$$

درایه‌ی  $[P_{i,j}^2]$  از ماتریس  $P^2$  نشان دهنده احتمال انتقال از  $i$  به  $j$  است در دو مرحله (با طی کردن دو یال در گراف اصلی)

### ○ پرسش تئوری ۳۹:

مطابق بخش قبل می‌توان گفت که درایه‌های ماتریس  $P^t$  نشان دهنده احتمال رسیدن از راس  $i$  به  $j$  در  $t$  مرحله را نشان می‌دهند. اگر رابطه‌ی ضرب ماتریسی را برای  $P^t$  بسط دهیم داریم:

$$P_{i,j}^{(t)} = \sum_{k_1=1}^n \left( \sum_{k_2=1}^n \left( \sum_{k_3=1}^n \left( \dots \left( \sum_{k_t=1}^n P_{i,k_t} P_{k_t,k_{t-1}} \right) \dots \right) P_{k_3,k_2} \right) P_{k_2,k_1} \right) P_{k_1,j}$$

اگر ماتریس  $P$  دارای تمام درایه‌های ناصفر باشد، رابطه‌ی بالا به راحتی به شکل زیر در می‌آید:

$$P_{i,j}^{(t)} = \frac{1}{d_i} \left( \sum_{k=1}^n \frac{1}{d_k} \right)^{t-1}$$

### ○ پرسش تئوری ۴۰:

برای اثبات خواسته‌ی سوال روی تمام مسیرهای ممکن از  $i$  به  $j$  به طول  $t$  سیگما می‌بندیم:

$$P_{i,j}^{(t)} = \sum_{\text{path } a \text{ from } i \text{ to } j: (i, a_1, \dots, a_{t-1}, j)} \frac{1}{d_i} \left( \prod_{k=1}^{t-1} \frac{1}{d_{a_k}} \right)$$

$$P_{j,i}^{(t)} = \sum_{\text{path } a \text{ from } i \text{ to } j: (i, a_1, \dots, a_{t-1}, j)} \frac{1}{d_j} \left( \prod_{k=1}^{t-1} \frac{1}{d_{a_k}} \right)$$

حال در محاسبه‌ی نسبت دو عبارت جملات متناظر دو سیگما را نسبت به هم تقسیم می‌کنیم. پس خواهیم داشت:

$$\frac{P_{i,j}^{(t)}}{P_{j,i}^{(t)}} = \frac{\frac{1}{d_i}}{\frac{1}{d_j}} = \frac{d_j}{d_i}$$

#### پرسش تئوری ۴۱:

اگر  $a$  و  $z$  در یک خوش باشند، احتمال  $P_{i,j}^{(t)}$  بزرگتر خواهد بود نسبت به زمانی که در دو خوشی متفاوت باشند. در ضمن اگر  $a$  و  $z$  در یک خوش باشند، دو احتمال  $P_{i,k}^{(t)}$  و  $P_{j,k}^{(t)}$  به هم نزدیک بوده و تقریباً برابرند، ولی در صورتی که در یک خوش نباشند، این دو احتمال عملاً از هم مستقل بوده و برابر نیستند.

#### پرسش تئوری ۴۲:

احتمال رفتن از  $a$  به  $z$  را برای تمام اعضای یک خوش میانگین گرفته و به آن خوش نسبت می‌دهیم:

$$P_{C_1, C_2}^{(t)} = \frac{1}{|C_1||C_2|} \sum_{i \in C_1, j \in C_2} P_{i,j}^{(t)}$$

اختلاف سلیقه‌ی دو خوش را به این شکل تعریف می‌کنیم:

$$r_{i,j} = \sqrt{\sum_{k=1}^m \frac{(P_{C_i, C_k}^{(t)} - P_{C_j, C_k}^{(t)})^2}{d_{C_k}}}$$

که  $m$  تعداد کل خوشها و  $d_{C_k}$  درجه‌ی خروجی هر خوش است.

#### پرسش تئوری ۴۳:

از جایی که باید یال‌های داخل خوش از بیرون بیشتر باشد، معیار  $Q$  را به این شکل تعیین می‌کنیم:

$$Q = \sum_{k=1}^n \left( \frac{l_k}{m} - \left( \frac{d_k}{2m} \right)^2 \right)$$

که  $n$  تعداد خوشها،  $m$  تعداد یال‌ها و  $l_k$  تعداد یال‌ها درون یک خوش و  $d_k$  مجموع درجات رئوس در یک خوش است.

#### پرسش شبیه سازی ۲۸:

#### پرسش شبیه سازی ۲۹: