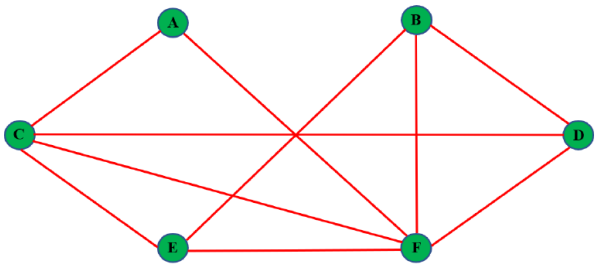
تمرین اول زیست شناسی سامانه ها

ندا اصفهانی – 610498099

گراف زیر را در نظر گرفته و به سوالات پاسخ دهید.



1. **لیست گره ها (nodes) و یال ها (edges) را مشخص نمایید.**

برای کارکردن با گراف یکی از راه ها استفاده از کتابخانه NetworkX در زبان پایتون است. ابتدا با استفاده از متود های add_nodes_from() و add_edges_from() گراف را تعریف میکنیم سپس با استفاده از متود nodes() و edges() میتوان گره ها و نود های یک گراف را مشاهده کرد.

```python
import networkx as nx
import matplotlib.pyplot as plt

# make a graph named G
G = nx.Graph()
G.add_nodes_from(["A", "B", "C", "D", "E", "F"])
G.add_edges_from([
    ("A", "C"), ("A", "F"),
    ("B", "E"), ("B", "F"), ("B", "D"),
    ("C", "A"), ("C", "D"), ("C", "F"), ("C", "E"),
    ("D", "B"), ("D", "C"), ("D", "F"),
    ("E", "C"), ("E", "B"), ("E", "F"),
    ("F", "A"), ("F", "B"), ("F", "C"), ("F", "D"), ("F", "E")
])
print("Graph Nodes: ", G.nodes)
print("Graph Edges: ")
for edge in G.edges:
    print(edge)
print()
```

```
PS C:\Users\eyesun.net\OneDrive\پ اتکس‌د\python>
11/python.exe c:/Users/eyesun.net/OneDrive/پ اتکس‌د
Graph Nodes:  ['A', 'B', 'C', 'D', 'E', 'F']
Graph Edges:
('A', 'C')
('A', 'F')
('B', 'E')
('B', 'F')
('B', 'D')
('C', 'D')
('C', 'F')
('C', 'E')
('D', 'F')
('E', 'F')
```

2. **درجه همه گره ها را مشخص کنید.**

مجموع تعداد یال های ورودی یا خروجی از هر گره برابر درجه آن گره میشود. بنابراین به ترتیب گره A درجه 2 و گره B درجه 3 و گره C درجه 4 و گره D و E درجه 3 و گره F درجه 5 هستند. برای این کار با استفاده از دستور Degree() در کتابخانه nx نیز ابتدا یک دیکشنری ساخته میشود که کلید های آن برابر گره ها و مقادیر هر کلید برابر درجه آن است. سپس میتوان این دیکشنری را پرینت کرد.

```python
# get the degree of each node
print("Degree of each node: ")
degree_dict = dict(G.degree())
for node, degree in degree_dict.items():
    print(f"Node {node} has degree {degree}")
print()
```

```
Degree of each node:
Node A has degree 2
Node B has degree 3
Node C has degree 4
Node D has degree 3
Node E has degree 3
Node F has degree 5
```

3. **سه مسیر بین گره A و D و shortest path ها بین این دو گره را مشخص کنید.**

برای این کار ابتدا نیاز داریم تمام مسیرهای بین دو گره مد نظر را پیدا کنیم. به این منظور تابعی تعریف میکنیم که به صورت بازگشتی بین همسایه های گره مدنظر بگردد و مسیر های منتهی به گره نهایی را پیدا کند. در نهایت تابعی برای پیدا کردن کوتاه ترین مسیر بین دو گره (بر اساس طول لیست) بین تمام مسیرهای پیدا شده مینویسیم.

```python
# define function to get all paths between two nodes
# use a graph traversal algorithm named Depth-First Search (DFS)
def get_all_paths(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return [path]
    paths = []
    for neighbor in graph.neighbors(start):
        if neighbor not in path:
            new_paths = get_all_paths(graph, neighbor, end, path)
            for new_path in new_paths:
                paths.append(new_path)
    return paths

All_path = get_all_paths(G, "A", "D")
print("All possible paths between node A and D: ")
[print (path) for path in All_path]
print()

def shortest_path(All_path):
    sp = All_path[0]
    for path in All_path:
        if len(path) < len(sp):
            sp = path
    sps = []
    for path in All_path:
        if len(path) == len(sp):
            sps.append(path)
    return sps

shortest_paths = shortest_path(All_path)
print("Shortest path(s): " ,shortest_paths)
print()
```

خروجی توابع ذکر شده به صورت زیر خواهد بود:

```
All possible paths between node A and D:
['A', 'C', 'D']
['A', 'C', 'F', 'B', 'D']
['A', 'C', 'F', 'D']
['A', 'C', 'F', 'E', 'B', 'D']
['A', 'C', 'E', 'B', 'F', 'D']
['A', 'C', 'E', 'B', 'D']
['A', 'C', 'E', 'F', 'B', 'D']
['A', 'C', 'E', 'F', 'D']
['A', 'F', 'B', 'E', 'C', 'D']
['A', 'F', 'B', 'D']
['A', 'F', 'C', 'D']
['A', 'F', 'C', 'E', 'B', 'D']
['A', 'F', 'D']
['A', 'F', 'E', 'B', 'D']
['A', 'F', 'E', 'C', 'D']

Shortest path(s):  [['A', 'C', 'D'], ['A', 'F', 'D']]
```

4. بیشینه یال ممکن برای این گراف چند یال می باشد؟ دانسیته گراف را محاسبه نمایید.

برای محاسبه بیشینه یال ممکن برای این گراف نیز از فرمول زیر استفاده میکنیم(n تعداد گره ها)

```
#calculate maximum edges
def max_edge():
    n = len(G.nodes)
    Max_edge = n * (n - 1) / 2
    return Max_edge

print("Maximum possible edges: ", max_edge())
print()
```

دانسیته گراف هم با استفاده از فرمول زیر بدست می آید.

```
# calculate density
def density():
    n = len(G.nodes)
    m = len(G.edges)
    Density = 2 * m / (n * (n-1))
    return Density
print("Density of the graph: ", density())
print()
```

خروجی توابع بالا به این صورت خواهد بود:

```
Maximum possible edges:  15.0

Density of the graph:  0.6666666666666666
```

5. آیا در گراف بالا clique وجود دارد؟ در صورت وجود نشان دهید.

Clique مجموعه ای از گره هاست که هر کدام به باقی گره ها متصل است. اگر به شکل این گراف نگاه کنیم Clique های سه گره ای در آن یافت میشود. گره های F/A/C, F/C/D, F/C/E, F/B/D, F/B/E با هم Cliqueهای این گراف را تشکیل داده اند. برای پیدا کردن clique ها نیز میتوان از دستور find_cliques() در کتابخانه ذکر شده استفاده کرد. در صورت وجود، این برنامه لیست تمام clique ها را پرینت خواهد کرد.

```
# find clique in the graph
def clique():
    cliques = list(nx.algorithms.clique.find_cliques(G))
    if len(cliques) > 0:
        print("Clique Found!")
        return print("Clique(s): ", cliques)
    else:
        return print("No Clique!")
clique()
print()
```

خروجی برنامه:

```
Clique Found!
Clique(s):  [['F', 'C', 'A'], ['F', 'C', 'D'], ['F', 'C', 'E'], ['F', 'B', 'D'], ['F', 'B', 'E']]
```

6. ماتریس مجاورت، لیست مجاورت و ماتریس فاصله گراف بالا را بنویسید.

ابتدا برای پیدا کردن ماتریس مجاورت از تابعی استفاده میکنیم که با پیدا کردن index هرکدام از نود هایی که در یال های گراف هست، خانه متناظر با آنها را در ماتریس مجاورت برابر 1 قرار میدهد.

```python
#Adjacency Matrix
def adjacency_matrix():
    nodes = list(G.nodes())
    n = len(nodes)
    adj_matrix = [[0] * n for _ in range(n)]
    for x, y in G.edges:
        i, j = nodes.index(x), nodes.index(y)
        adj_matrix[i][j] = 1
        adj_matrix[j][i] = 1
    return adj_matrix
print("Adjacency Matrix:")
for row in adjacency_matrix():
    print(row)
print()
```

خروجی برنامه به شکل زیر خواهد بود:

```
Adjacency Matrix:
[0, 0, 1, 0, 0, 1]
[0, 0, 0, 1, 1, 1]
[1, 0, 0, 1, 1, 1]
[0, 1, 1, 0, 0, 1]
[0, 1, 1, 0, 0, 1]
[1, 1, 1, 1, 1, 0]
```

برای بدست آوردن لیست مجاورت نیز ابتدا یک دیکشنری میسازیم و سپس هرکدام از گره ها را به عنوان کلید این دیکشنری قرار میدهیم و در صورت تکرار آن گره در یال های دیگر، گره دیگر حاضر در یال را به عنوان مقدار به آن گره در دیکشنری اضافه میکنیم

```python
#Adjacency list
def adjacency_list():
    adj_list = {}
    for u, v in G.edges:
        if u not in adj_list:
            adj_list[u] = []
        if v not in adj_list:
            adj_list[v] = []
        adj_list[u].append(v)
        adj_list[v].append(u)
    return adj_list
print("Adjacency list:")
for node, neighbors in adjacency_list().items():
    print(f"{node}: {neighbors}")
print()
```

خروجی برنامه:

```
Adjacency list:
A: ['C', 'F']
C: ['A', 'D', 'F', 'E']
F: ['A', 'B', 'C', 'D', 'E']
B: ['E', 'F', 'D']
E: ['B', 'C', 'F']
D: ['B', 'C', 'F']
```

برای بدست آوردن ماتریس فاصله ها نیز ابتدا یک ماتریس n*n با مقادیر اولیه بی نهایت میسازیم سپس مشابه الگوریتمی که در ماتریس مجاورت گفتیم ماتریس فاصله را نیز پر میکنیم با این تفاوت که قطر این ماتریس همواره صفر خواهد بود زیرا فاصله هر گره از خودش برابر صفر است. سپس برای پیدا کردن کوچکترین فاصله بین دو گره از Floyd-Warshall Algorithm استفاده خواهیم کرد.

```python
#Distance Matrix
def distance_matrix():
    nodes = list(G.nodes)
    n = len(nodes)
    dist = [[float('inf')]*n for _ in range(n)] #build a n*n infinity matrix
    for i in range(n):
        dist[i][i] = 0
    for x, y in G.edges:
        i, j = nodes.index(x), nodes.index(y)
        dist[i][j] = 1
        dist[j][i] = 1
    for k in range(n):
        for i in range(n):
            for j in range(n):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
    return dist
print("Distance Matrix:")
for row in distance_matrix():
    print(row)
print()
```

در این برنامه ابتدا عناصر روی قطر ماتریس را برابر صفر قرار میدهیم و باقی عناصر را برابر بی نهایت.

```
      A    B    C    D    E    F
   ----------------------------
A  |  0  inf  inf  inf  inf  inf
B  | inf   0  inf  inf  inf  inf
C  | inf  inf  0   inf  inf  inf
D  | inf  inf  inf  0   inf  inf
E  | inf  inf  inf  inf  0   inf
F  | inf  inf  inf  inf  inf  0
```

سپس عناصر معادل یال های بین هر دو گره را در ماتریس برابر وزن یال(در اینجا یال ها بدون وزن است و معادل 1) قرار میدهیم.

```
      A    B    C    D    E    F
   ----------------------------
A  |  0  inf   1   inf  inf   1
B  | inf   0  inf   1    1    1
C  |  1   inf  0    1    1    1
D  | inf   1   1    0   inf   1
E  | inf   1   1   inf   0    1
F  |  1    1   1    1    1    0
```

برای هر جفت یال (i,j) اگر فاصله بین i و j که از گره k عبور میکند کمتر از فاصله حال حاضر بین i و j بود، آن را در ماتریس آپدیت میکنیم و این کار را برای تمام یال ها تکرار میکنیم. ماتریس نهایی فاصله به شکل زیر خواهد بود.

	A	B	C	D	E	F
A	0	2	1	2	2	1
B	2	0	3	1	1	1
C	1	3	0	1	1	1
D	2	1	1	0	2	1
E	2	1	1	2	0	1
F	1	1	1	1	1	0

7. قطر شبکه و Average Path Length را برای شبکه بالا محاسبه کنید.

قطر شبکه به معنای بزرگترین shortest path بین گره ها است. پس بالاترین عددی که در ماتریس فاصله وجود دارد همان قطر شبکه میشود. بنابراین طبق ماتریس فاصله ای که در بالا محاسبه کردیم، قطر شبکه برابر 3 است.

برای بدست آوردن Average Path Length نیز باید تمام shortest path ها بین گره ها پیدا شود و باهم جمع شود سپس به تعداد کل مسیر ها تقسیم شود تا میانگین طول مسیر در گراف بدست آید.

```python
#Diameter of Network
diameter = max(max(row) for row in distance_matrix())
print("Diameter of the graph:", diameter)
print()

#Average Path Length
def average_path_length():
    num_paths = 0
    total_path_length = 0

    nodes = list(G.nodes)
    n = len(nodes)

    for i in range(n):
        for j in range(i+1, n):
            path = get_all_paths(G, nodes[i], nodes[j])
            num_paths += len(path)
            for z in path:
                total_path_length += len(z)-1

    return (total_path_length / num_paths)
print("Average Path Length: ", average_path_length())
print()
```

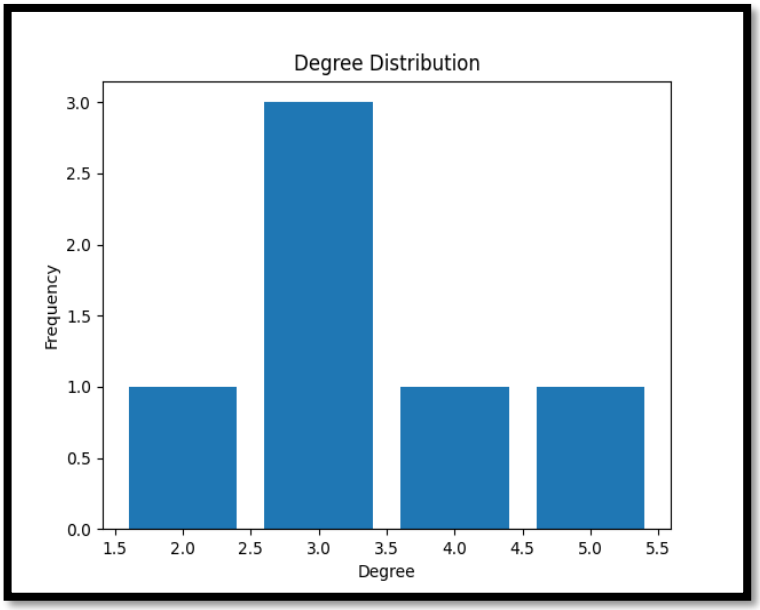
```
Average Path Length:  3.3855421686746987
```

8. توزیع درجه ها در شبکه بالا را رسم کنید.

برای رسم توزیع درجه ها از کتابخانه matplotlib.pyplot استفاده میکنیم. با استفاده از دیکشنری که حین بدست آوردن درجه گره ها ایجاد کردیم میتوان نمودار های توزیع درجه ها را در این گراف رسم کرد.

```python
#Degree Distribution Plot
plt.bar(degree_dict.keys(), degree_dict.values())
plt.xlabel("Degree")
plt.ylabel("Number of Nodes")
plt.title("Degree Distribution")
plt.show()

distribution = {}
for i in degree_dict.values():
    if i in distribution:
        distribution[i] += 1
    else:
        distribution[i] = 1
plt.bar(distribution.keys(), distribution.values())
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.title("Degree Distribution")
plt.show()
```



9. مقدار clustering coefficient را برا ی گره F محاسبه کنید.

در اینجا نیاز است تا تعداد یال های واقعی بین گره مشخص شده و دیگر گره های گراف را محاسبه کنیم سپس تعداد کل این یال های بدست آمده را به تعداد کل یال هایی که میتوانست بین این گره و گره های دیگر وجود داشته باشد تقسیم کنیم تا clustering coefficient گره بدست آید. بین همسایه های گره F به اندازه k(k-1)/2 که k = تعداد همسایه های گره F است، یال میتواند وجود داشته باشد ولی تعداد یال های واقعی بین این همسایه ها را از روی شکل اگر بشماریم برابر 5 تاست. پس Clustering coefficient این گره به شکل زیر محاسبه میشود:

$$Clustering\ Coefficient = \frac{Actual\ Edges}{Possible\ Edges} = \frac{5}{5 \times 4 \div 2} = 0.5$$

```python
#Clustering Coefficient
def clustering_coefficient(node):
    neighbors = G[node]
    k = len(neighbors)
    if k < 2:
        return 0
    else:
        num_actual_edges = 0
        for i, ni in enumerate(neighbors):
            for j, nj in enumerate(neighbors):
                if i < j and ni in G[nj]:
                    num_actual_edges += 1
        num_possible_edges = k * (k - 1) / 2
        return num_actual_edges / num_possible_edges
print("Clustring Coefficient of Node F: ", clustering_coefficient("F"))
```

```
Clustring Coefficient of Node F:  0.5
PS C:\Users\eyesun.net\OneDrive\دیاتکس\python>
```