Mahire Zühal Özdemir

19360859015

Proje (Final 6)

Networkx ile Graph Oluşturma:

Graph düğümlerden oluşan bir yapıdır.

```
import networkx as nx
G = nx.Graph()
```

Herhangi bir düğüm eklenmediği için çıktı aşağıdaki gibidir.Sadece graph oluşturulmuştur.

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py

Process finished with exit code 0
```

Tek seferde bir düğüm eklemek için

```
G.add_node(1)
```

Ve çıktısı aşağıdaki gibidir

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 1 nodes and 0 edges

Process finished with exit code 0
```

Veya liste şeklinde düğüm eklemek için:

```
G.add_nodes_from([2, 3])
```

Çıktısı aşağıdaki gibidir

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 2 nodes and 0 edges

Process finished with exit code 0
```

Düğümler aynı zamanda düğüm nitelikleriyle birlikte tanımlanabilir:

```
G.add_nodes_from([
    (4, {"color": "red"}),
    (5, {"color": "green"}),
])
```

Bir grafikteki düğümler başka bir grafiğe dahil edilebilir:

```
H = nx.path_graph(10)
G.add_nodes_from(H)
print(G)
```

Çıktı aşağıdaki gibidir:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 10 nodes and 0 edges

Process finished with exit code 0
```

G,H'nin düğümlerini içerdiği için H'yi de G düğümü olarak ifade edebiliriz.

Edge(Kenar) Ekleme:

Her seferinde tek tek kenar eklenerek de graph oluşturulabilir:

```
G.add_edge(1, 2)
e = (2, 3)
G.add_edge(*e)
print(G)
```

Çıktı:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 10 nodes and 2 edges

Process finished with exit code 0
```

Veya liste olarak da kenar eklenebilir:

```
G.add_edges_from([(1, 2), (1, 3)])
print(G)
```

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 10 nodes and 2 edges

Process finished with exit code 0
```

Var olan bir graphın kenarlarını farklı bir grapha ekleyebilriz:

```
G.add_edges_from(H.edges)
```

Graph Temizleme:

Clear metodu ile graph temizlenip yeni düğümler ve kenarlar eklenebilir:

```
G.clear()
G.add_edges_from([(1, 2), (1, 3)])
G.add_node(1)
G.add_edge(1, 2)
G.add_node("spam")        # adds node "spam"
G.add_nodes_from("spam")  # adds 4 nodes: 's', 'p', 'a', 'm'
G.add_edge(3, 'm')
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 8 nodes and 3 edges

Process finished with exit code 0
```

G.edges hem düğüm hem de kenar sıralarını belirler:

```
DG = nx.DiGraph()
DG.add_edge(2, 1)   # adds the nodes in order 2, 1
DG.add_edge(1, 3)
DG.add_edge(2, 4)
DG.add_edge(1, 2)
assert list(DG.successors(2)) == [1, 4]
assert list(DG.edges) == [(2, 1), (2, 4), (1, 3), (1, 2)]
print(DG.number_of_nodes())
print(DG.number_of_edges())
```

Çıktı:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
4
4

Process finished with exit code 0
```

Graph Ögeleri:

Bir graphta G.nodes, G.edges, G.adj ve G.degree ögeleri bulunur. Bunlar, bir graphtaki düğümlerin, kenarların, komşuların (bitişikliklerin) ve düğümlerin derecelerinin özellikleridir

```
print(list(DG.nodes))

print(list(DG.edges))

print(list(DG.adj[1]))  # or list(G.neighbors(1))

print(DG.degree[1])  # the number of edges incident to 1
```

DG graphının özellikleri aşağıdaki gibidir:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
[2, 1, 3, 4]
[(2, 1), (2, 4), (1, 3), (1, 2)]
[3, 2]
3

Process finished with exit code 0
```

Tüm düğümlerin bir alt kümesinden kenarları ve dereceyi bildirmek için kullanabiliriz:

```
print(DG.edges([2, 4]))
print(DG.degree([2, 3]))
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
[(2, 1), (2, 4)]
[(2, 3), (3, 1)]

Process finished with exit code 0
```

Graphtan Eleman Silme:

```
print(list(G.nodes))
G.remove_node(2)
G.remove_nodes_from("spam")
print(list(G.nodes))
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
[1, 2, 3, 'spam', 's', 'p', 'a', 'm']
[1, 3, 'spam']

Process finished with exit code 0
```

Graph Yapıcıları Kullanma:

DiGraph ile ikili yönlü graph oluşturulabilir.

```
G.add_edge(1, 2)
H = nx.DiGraph(G)  # create a DiGraph using the connections from G
print(list(H.edges())) #[(1, 2), (2, 1)]
edgelist = [(0, 1), (1, 2), (2, 3)]
H = nx.Graph(edgelist)  # create a graph from an edge list
print(list(H.edges())) #[(0, 1), (1, 2), (2, 3)]
adjacency_dict = {0: (1, 2), 1: (0, 2), 2: (0, 1)}
H = nx.Graph(adjacency_dict)  # create a Graph dict mapping nodes to nbrs
print(list(H.edges())) #[(0, 1), (0, 2), (1, 2)]
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
[(1, 2), (2, 1)]
[(0, 1), (1, 2), (2, 3)]
[(0, 1), (0, 2), (1, 2)]

Process finished with exit code 0
```

Kenarlara ve Komşulara Erişim:

Graph.edges ve Graph.adj görünümlerine ek olarak, alt simge gösterimi kullanılarak kenarlara ve komşulara erişim mümkündür.

```
G = nx.Graph([(1, 2, {"color": "yellow"})])
print(G[1]) # same as G.adj[1] --> {2: {'color': 'yellow'}}
print(G[1][2]) #{'color': 'yellow'}
print(G.edges[1, 2]) #{'color': 'yellow'}
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
{2: {'color': 'yellow'}}
{'color': 'yellow'}
{'color': 'yellow'}

Process finished with exit code 0
```

Kenar ekleyip attribute seçilebilir:

```
G.add_edge(1, 3)
G[1][3]['color'] = "blue"
G.edges[1, 2]['color'] = "red"
print(G.edges[1, 2]) #{'color': 'red'}
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
{'color': 'red'}

Process finished with exit code 0
```

Tüm düğüm çiftlerinin incelenmesi G.adjacency() veya G.adj.items() kullanılarak gerçekleştirilir.

```
FG = nx.Graph()
FG.add_weighted_edges_from([(1, 2, 0.125), (1, 3, 0.75), (2, 4, 1.2), (3, 4, 0.375)])
for n, nbrs in FG.adj.items():
    for nbr, eattr in nbrs.items():
        wt = eattr['weight']
        if wt < 0.5: print(f"({n}, {nbr}, {wt:.3})")
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
(1, 2, 0.125)
(2, 1, 0.125)
(3, 4, 0.375)
(4, 3, 0.375)

Process finished with exit code 0
```

Kenarlar özelliği ile tüm kenarlara erişim sağlanır.

```
for (u, v, wt) in FG.edges.data('weight'):
    if wt < 0.5:
        print(f"({u}, {v}, {wt:.3})")
```

Ağırlık özelliği 0.5'den küçük olan kenarlar:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
(1, 2, 0.125)
(3, 4, 0.375)

Process finished with exit code 0
```

Graph Öznitelikleri:

Yeni graph oluşturuken öznitelik atamaları yapma:

```
G = nx.Graph(day="Friday")
print(G.graph)
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
{'day': 'Friday'}

Process finished with exit code 0
```

Öznitelikler daha sonra düzenlenebilir:

```
G = nx.Graph(day="Friday")
print(G.graph)
G.graph['day'] = "Monday"
print(G.graph)
```

Friday → Monday

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
{'day': 'Friday'}
{'day': 'Monday'}

Process finished with exit code 0
```

Düğüm Öznitelikleri:

add_node(), add_nodes_from() veya G.nodes kullanarak düğüm öznitelikleri ekleyin:

```
G.add_node(1, time='5pm')
G.add_nodes_from([3], time='2pm')
print(G.nodes[1])

G.nodes[1]['room'] = 714
print(G.nodes.data())
```

Tüm verileri nodes.data() ile yazdırdık. Belirli bir düğüme de özniteliği node,nodes_from,ve nodes kullanarak ekledik:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
{'time': '5pm'}
[(1, {'time': '5pm', 'room': 714}), (3, {'time': '2pm'})]

Process finished with exit code 0
```

Kenar Öznitelikleri:

add_edge(), add_edges_from() veya alt simge gösterimini kullanarak kenar nitelikleri ekleyip değiştirebiliriz.

```
G.add_edge(1, 2, weight=4.7 )
G.add_edges_from([(3, 4), (4, 5)], color='red')
G.add_edges_from([(1, 2, {'color': 'blue'}), (2, 3, {'weight': 8})])
G[1][2]['weight'] = 4.7
G.edges[3, 4]['weight'] = 4.2
print(G.edges.data())
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
[(1, 2, {'weight': 4.7, 'color': 'blue'}), (3, 4, {'color': 'red', 'weight': 4.2}), (3, 2, {'weight': 8}), (4, 5, {'color': 'red'})]

Process finished with exit code 0
```

Yönlendirilmiş Graphlar:

```
DG = nx.DiGraph()
DG.add_weighted_edges_from([(1, 2, 0.5), (3, 1, 0.75)])
print(DG.out_degree(1, weight='weight')) #0.5

print(DG.degree(1, weight='weight')) #1.25

print(list(DG.successors(1))) #[2]

print(list(DG.neighbors(1))) #[2]
```

neighbors() and successors() aynı şeydir. Düğümün ardıllarıdır.

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
0.5
1.25
[2]
[2]

Process finished with exit code 0
```

Yönlendirilmiş graphtan yönlendirilmemiş graph oluşturmak için Graph.to_undirect() metodu kullanılır ya da aşağıdaki gibi yapılır:

```
H = nx.Graph(G)  # create an undirected graph H from a directed graph G
```

Multigraphs:

Çoklu kenarları sağlayan bir yönlendirilmiş graph sınıfıdır.

Çoklu kenarlar, iki düğüm arasındaki birden fazla kenardır. Her kenar isteğe bağlı verileri veya öznitelikleri tutabilir. Bir MultiDiGraph yönlendirilmiş kenarları tutar.

```
MG = nx.MultiGraph()
MG.add_weighted_edges_from([(1, 2, 0.5), (1, 2, 0.75), (2, 3, 0.5)])
print(dict(MG.degree(weight='weight'))) #{1: 1.25, 2: 1.75, 3: 0.5}

GG = nx.Graph()
for n, nbrs in MG.adjacency():
   for nbr, edict in nbrs.items():
       minvalue = min([d['weight'] for d in edict.values()])
       GG.add_edge(n, nbr, weight = minvalue)

print(nx.shortest_path(GG, 1, 3)) #[1, 2, 3]
```

shortest_path: graphdaki en kısa yolu hesaplar. Örnekte 1. Düğümden 3. Düğüme kadar olan yolu hesaplar.

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
{1: 1.25, 2: 1.75, 3: 0.5}
[1, 2, 3]

Process finished with exit code 0
```

Graph İşlemleri:

1-)Klasik graph işlemleri:

subgraph(G, nbunch) → nbunch içindeki düğümler üzerinde indüklenen alt grafiği döndürür.

union(G, H[, rename]) → G ve H grafiklerini birleştirir.

disjoint_union(G, H) → G ve H grafiklerini birleştirir

cartesian_product(G, H) → G ve H'nin Kartezyen çarpımını döndürür.

compose(G, H) → Düğümleri ve kenarları tek bir grafta birleştirerek G grafını H ile oluşturur.

complement(G) → G'nin çizge tümleyenini döndürür.

create_empty_copy(G[, with_data]) → G grafiğinin tüm kenarları kaldırılmış bir kopyasını döndürür.

to_undirected(graph) → Çizge grafiğinin yönlendirilmemiş bir görünümünü döndürür.

to_directed(graph) → Çizge grafiğinin yönlendirilmiş bir görünümünü döndürür.

2-)Klasik graphlara çağrı işlemleri:

petersen_graph([create_using]) → Petersen grafiğini döndürür.

tutte_graph([create_using]) → Tutte grafiğini döndürür.

sedgewick_maze_graph([create_using]) → Bir döngüye sahip küçük bir labirent döndürür.

tetrahedral_graph([create_using]) → düzenli Platonik Tetrahedral grafiği döndürür.

3-)Klasik bir graph için yapıcı kullanmak:

complete_graph(n[, create_using]) → Düğümlü K_n tam grafiğini döndürür.

complete_bipartite_graph(n1, n2[, create_using]) → K_{n_1,n_2} tam iki parçalı grafiğini döndürür.

barbell_graph(m1, m2[, create_using]) → Barbell Grafiğini döndürür: bir yol ile bağlanmış iki tam grafik.

lollipop_graph(m, n[, create_using]) → Lollipop Grafiğini döndürür; P_n'ye bağlı K_m.

```
K_5 = nx.complete_graph(5)
K_3_5 = nx.complete_bipartite_graph(3, 5)
barbell = nx.barbell_graph(10, 10)
lollipop = nx.lollipop_graph(10, 20)
print(K_5)
print(K_3_5)
print(barbell)
print(lollipop)
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 5 nodes and 10 edges
Graph named 'complete_bipartite_graph(3, 5)' with 8 nodes and 15 edges
Graph with 30 nodes and 101 edges
Graph with 30 nodes and 65 edges

Process finished with exit code 0
```

4-)Stokastik bir graph üreteci kullanmak:

erdos_renyi_graph(n, p[, seed, directed]) →Bir Erdős-Rényi grafiği veya binom grafiği olarak da bilinen rastgele grafik Gn,p döndürür.

watts_strogatz_graph(n, k, p[, seed]) →Bir Watts-Strogatz small-world grafiği döndürür.

barabasi_albert_graph(n, m[, seed, ...]) → Barabási-Albert tercihli bağlanma kullanarak rastgele bir çizge döndürür

random_lobster(n, p1, p2[, seed]) → Rastgele bir lobster grafiği döndürür.

```
er = nx.erdos_renyi_graph(100, 0.15)
ws = nx.watts_strogatz_graph(30, 3, 0.1)
ba = nx.barabasi_albert_graph(100, 5)
red = nx.random_lobster(100, 0.9, 0.9)
print(er)
print(ws)
print(ba)
print(red)
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
Graph with 100 nodes and 751 edges
Graph with 30 nodes and 30 edges
Graph with 100 nodes and 475 edges
Graph with 5681 nodes and 5680 edges

Process finished with exit code 0
```

5-)Yaygın graph formatlarını kullanarak bir dosyada saklanan bir grafiği okuma:

NetworkX, kenar listeleri, bitişiklik listeleri, GML, GraphML, LEDA vb. birçok popüler formatı destekler.

```
nx.write_gml(red, "file.gml")
mygraph = nx.read_gml("file2.gml")
```

Graph Analizi:

G'nin yapısı, aşağıdaki gibi çeşitli graph teorisi fonksiyonları kullanılarak analiz edilebilir.

```
G = nx.Graph()
G.add_edges_from([(1, 2), (1, 3)])
G.add_node("spam")       # adds node "spam"
print(list(nx.connected_components(G))) #[{1, 2, 3}, {'spam'}]

print(sorted(d for n, d in G.degree())) #[0, 1, 1, 2]

print(nx.clustering(G)) #{1: 0, 2: 0, 3: 0, 'spam': 0}
```

connected_components: Bağlı bileşenler oluşturur.

nx.clustering: G'nin ortalama kümeleme katsayısını tahmin eder.

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
[{1, 2, 3}, {'spam'}]
[0, 1, 1, 2]
{1: 0, 2: 0, 3: 0, 'spam': 0}

Process finished with exit code 0
```

Büyük çıktılı bazı fonksiyonlar (node, value) 2-tuples üzerinde yineleme yapar. İsterseniz bunlar kolayca bir dict yapısında saklanabilir

```
sp = dict(nx.all_pairs_shortest_path(G))
print(sp[3]) #{3: [3], 1: [3, 1], 2: [3, 1, 2]}
```

Output:

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje6final.py
{3: [3], 1: [3, 1], 2: [3, 1, 2]}

Process finished with exit code 0
```

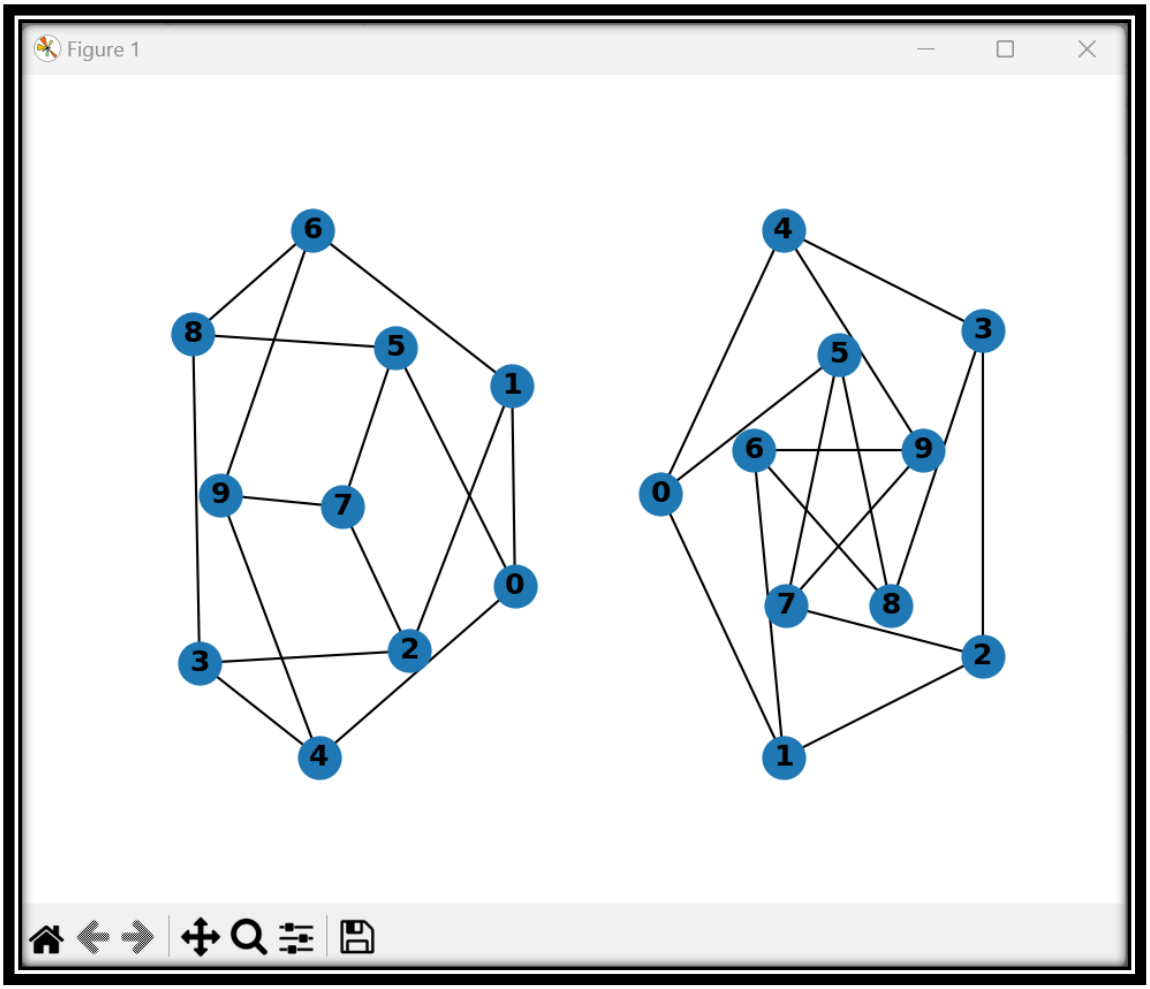
Graph çizme:

Matplotlib kütüphanesi kullanılarak graphlar ile görsel arayüz oluşturulur.

```
import matplotlib.pyplot as plt

G = nx.petersen_graph()
subax1 = plt.subplot(121)
nx.draw(G, with_labels=True, font_weight='bold')
subax2 = plt.subplot(122)
nx.draw_shell(G, nlist=[range(5, 10), range(5)], with_labels=True, font_weight='bold')
plt.show()
```

Output:



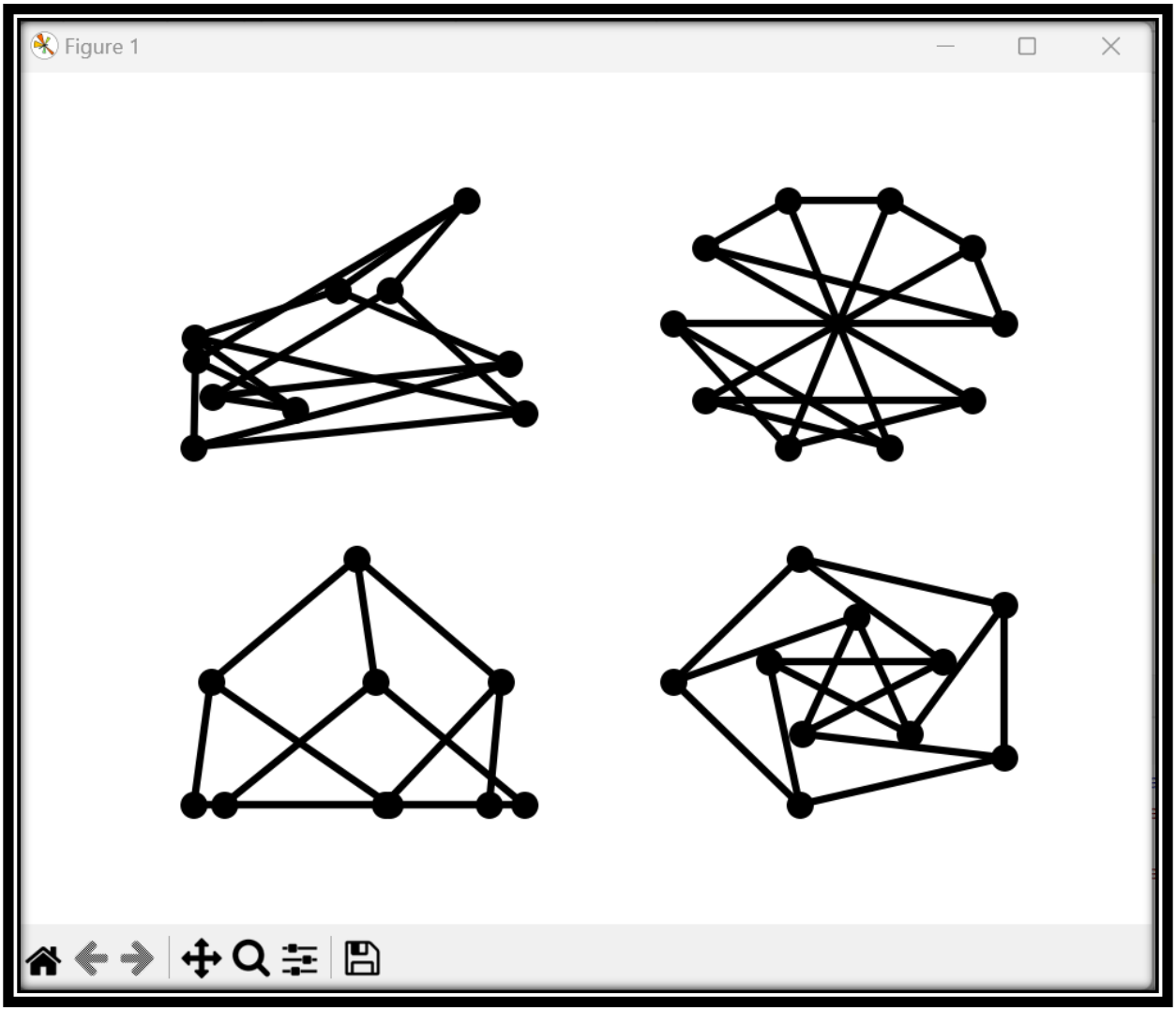
Çeşitli seçenekleri belirlemek için:

```
options = {
    'node_color': 'black',
    'node_size': 100,
    'width': 3,
}
subax1 = plt.subplot(221)
nx.draw_random(G, **options)
subax2 = plt.subplot(222)
nx.draw_circular(G, **options)
subax3 = plt.subplot(223)
nx.draw_spectral(G, **options)
subax4 = plt.subplot(224)
nx.draw_shell(G, nlist=[range(5,10), range(5)], **options)

plt.show()
```

draw_shell() ile birden fazla shell kullanılabilir.

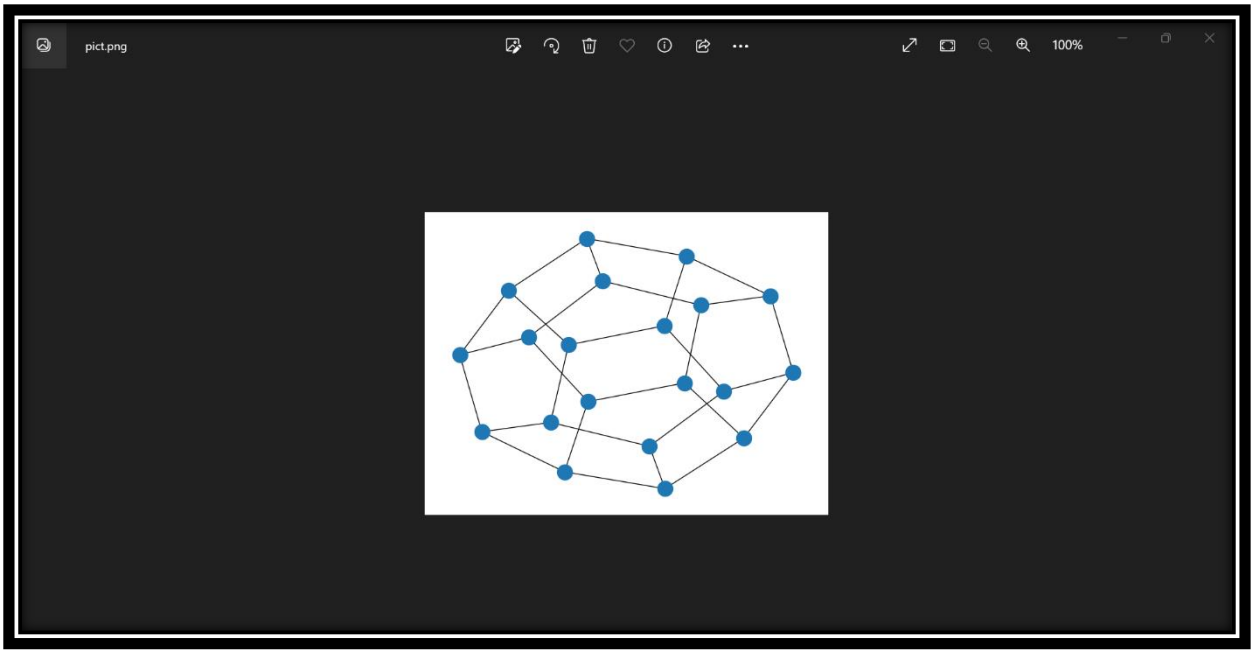
Output:



Graph Kaydetme:

```
nx.draw(G)
plt.savefig("C://Users//zuhal//Desktop//bioinformaticProjects//final6//pict.png")
```

Oluşan graph verilen dosya yoluna verilen isimde kaydedilir.



Graphviz ve PyGraphviz veya pydot sisteminizde mevcutsa, düğüm konumlarını almak veya grafiği daha fazla işlem için nokta biçiminde yazmak için networkx.drawing.nx_agraph.graphviz_layout veya networkx.drawing.nx_pydot.graphviz_layout'u da kullanabilirsiniz.

```
from networkx.drawing.nx_pydot import write_dot
pos = nx.nx_agraph.graphviz_layout(G)
nx.draw(G, pos=pos)
write_dot(G, 'C://Users//zuhal//Desktop//bioinformaticProjects//final6//file.dot')
```