```
import numpy as np
from itertools import combinations
from networkx.drawing.nx_pydot import write_dot
from matplotlib import pyplot as plt
import pydot #çoklu graf, graflari görselleştirme için
import os
import sys
import turtle
import graphviz
import networkx as nx
from pandas.core.indexes import multi
```

Şekil 1

Şekil 1 de gerekli kütüphaneler import edilmiştir.

```
komsuluk_matrisi = []
with open('komsuluk.csv') as f:
    for satir in f.readlines():
        komsuluk_matrisi.append( list(map(int, satir.split(','))))
```

Şekil 2

Şekil 2 de komsuluk.csv dosyasından veri alınmıştır.

```
A = np.array(komsuluk_matrisi)
G1 = nx.from_numpy_array(A,create_using=nx.MultiGraph)

print("**********************")
print(nx.is_eulerian(G1))
print("**********************")
```

Şekil 3

Şekil 3 te G1 grafı oluşturulup, euler grafı olup olmadığı kontrol edilmiştir.

```
#euler grafı yapmak için tek ola tepeleri çift dereceli hale getirmeliyiz.
#PYDOT için etiketler
for e in G1.edges():
    G1[e[0]][e[1]][0]['label'] = G1[e[0]][e[1]][0]['weight']
```

Şekil 4

Şekil 4 te grafın etiketleri verilmiştir.

```python
#Bir grafiğin Eulerian olup olmadığını kontrol eder.
def is_eulerian(G1):

    degree_list = [val for (node, val) in G1.degree()]
    for i in degree_list:
        if i % 2 == 1:
            return False
    return True
```

Şekil 5

Şekil 5 te grafın derecelerine bakılarak euler grafı olup olmadığını döndüren fonksiyon yazılmıştır.

```python
#Tek dereceli tepe noktasının listesini döndürür
def get_odd_degree_list(G1):
    odd_degree_list = []
    for i in G1.degree():
        if i[1] % 2 == 1:
            odd_degree_list.append(i[0])
            print(odd_degree_list)
    return odd_degree_list
```

Şekil 6

Şekil 6 da grafta bulunan tek dereceli düğümlerin sayısı ve hangi düğümler olduğunu döndüren fonksiyon yazılmıştır. Buradaki amaç tek dereceli düğümlerin sayısını bulmaktır. Daha sonraki işlem de gerekli olan liste buradan dönecektir.

```python
#tek dereceli köşelerin tüm olası eşleşmelerinin (kombinasyonları) listesini döndürür
def pairing_vertex(od_list, ret_list, singular_ord, len_od_list):

    i = 0
    while i < len_od_list and od_list[i] == "_":
        i += 1

    if i >= len_od_list:
        ret_list.append([])
        ret_list[-1] = [sublist.copy() for sublist in singular_ord]

    else:
        singular_ord.append([od_list[i]])
        od_list[i] = "_"
        print("od_list",od_list[i])

        print("************************")

        for j in range(i + 1, len_od_list):
            if od_list[j] != "_":
                #print("od_list",od_list[i])
                #print("///////////////**********************")

                singular_ord[-1].append(od_list[j])
                print("singular_ord",singular_ord)
                od_list[j] = "_"
                pairing_vertex(od_list, ret_list, singular_ord, len_od_list)

                od_list[j] = singular_ord[-1][-1]
                print("singular_ord",singular_ord)
                del singular_ord[-1][-1]

        od_list[i] = singular_ord[-1][0]
        print("od_list",od_list[i])
        print("************************")
        del singular_ord[-1]
    print("ret_lis",ret_list)
    return ret_list
```

Şekil 7

Şekil 7 de, tek dereceli düğümlerin tüm olası eşleşmelerinin listesini döndürülmüştür.

```
************************
singular_ord [[0, 7]]
ret_lis [[[0, 7]]]
singular_ord [[0, 7]]
od_list 0
************************
ret_lis [[[0, 7]]]
```

Şekil 8

Şekil 8 de Şekil 7 nin ekran çıktısı verilmiştir.

```
def find_shortest_distance(G1):

    list_of_rows = komsuluk_matrisi
    g = Graph(len(G1.nodes))
    g.graph = list_of_rows
    #print(g)
    distances = g.length_from_source()
    #print(distances)
    return distances
```

Şekil 9

Şelik 9 da, en kısa mesafeyi bulan foksiyon yazılmıştır.

```
#Her kaynak düğümünün grafikteki diğer düğümlere en kısa yollarını döndürür.
def find_shortest_path(G1):

    list_of_rows = komsuluk_matrisi
    g = Graph(len(G1.nodes))
    g.graph = list_of_rows
    path = g.path()
    return path
```

Şekil 10

Şelik 10 da en kısa yolu bulan foksiyon yazılmıştır. Yol geri döndürülmüştür.

```
#Her kaynak düğümünün grafikteki diğer düğümlere en kısa yollarını döndürür.
def find_shortest_path(G1):

    list_of_rows = komsuluk_matrisi
    g = Graph(len(G1.nodes))
    g.graph = list_of_rows
    path = g.path()
    return path

#Grafikteki bir düğüm alt çiftini bağlayan ayrıntılı bir yolunu bulan fonk
def previous_vertex(G1, pairings, distance):
    node_ids = find_node_ids(G1)
    shortest_pairing = find_shortest_pairing(G1, pairings, distance)
    path = find_shortest_path(G1)

    previous_vertices = []
    for sub_pair in pairings[shortest_pairing]:
        src = node_ids.get(sub_pair[0])
        des = node_ids.get(sub_pair[1])
        pair_path = path[src]

        src_to_des = []
        src_to_des.append(des)
        while src_to_des[0] != src:
            prev = pair_path[src_to_des[0]]
            src_to_des.insert(0, prev)

        previous_vertices.append(src_to_des)

    for i in range(len(previous_vertices)):
        for j in range(len(previous_vertices[i])):
            for keys in node_ids:
                if node_ids[keys] == previous_vertices[i][j]:
                    previous_vertices[i][j] = keys

    return previous_vertices
```

Şekil 11

Şekil 11 de, grafikteki bir düğümün alt çiftini bağlayan ve ayrıntılı bir yol vermesini sağlayan fonksiyon yazılmıştır.

```
#Grafta bulunan düğümlerin idsi
def find_node_ids(G1):
    node_ids = {}
    i = 0
    for node in G1.nodes:
        node_ids[node] = i
        i += 1

    return node_ids
```

Şekil 12

Şekil 12 de, G1 de bulunan düğümlerin id sini döndüren fonksiyon yazılmıştır.

```python
#Her bir alt çifti tek dereceli tepe noktasının kombinasyonuna göre toplam yol uzunluğunu verir
def len_path(pairing, node_ids, distance):
    len = 0
    for sub_pair in pairing:
        src = node_ids.get(sub_pair[0])
        des = node_ids.get(sub_pair[1])
        len += distance[src][des]

    return len
```

Şekil 13

Şekil 13 te, grafta belirlenen yolun uzunluğu bulunmuştur.

```python
#Alt çiftleri birbirine bağlayan minimum en kısa yolla eşleştirme dizinini döndürür
def find_shortest_pairing(G1, pairings, distance):
    node_ids = find_node_ids(G1)
    ret = 0
    min_len = len_path(pairings[0], node_ids, distance)
    for i in range(1, len(pairings)):
        if len_path(pairings[i], node_ids, distance) < min_len:
            ret = i

    return ret
```

Şekil 14

Şekil 14 te, grafın düğümlerine göre mesafelerin uzunluğu bulunmuştur.

```python
def get_shortest_walk (G1):

    # euler değilse
    if not is_eulerian(G1):
        # tek dereceli köşelerin listesini alır
        odd_degree_list = get_odd_degree_list(G1)
        #print(len(odd_degree_list))
        # tek dereceli köşelerin kombinasyonlarını alır
        pairings = pairing_vertex(odd_degree_list, [], [], len(odd_degree_list))
        #print("pairings",pairings)
        #print("*****************************")
        # Her kaynak düğümden en kısa mesafeyi bulmak için Dijkstra algoritmasını uygular
        distances = find_shortest_distance(G1)
        #alt çiftleri birbirine bağlayan minimum en kısa yolla eşleştirmesini döndür
        shortest_pairing = find_shortest_pairing(G1, pairings, distances)

        # her alt çifti birbirine bağlayan en kısa yolları (kenarların sıralaması) döndürür.
        path = find_shortest_path(G1)
        print("path",path)
        # bulunan kenarları grafiğe ekler
        previous_vertices = previous_vertex(G1, pairings, distances)
        new_edges = []
        for i in range(len(previous_vertices)):
            new_edges = list(zip(previous_vertices[i], previous_vertices[i][1:]))
        for edge in new_edges:
            if edge in G1.edges:

                G1.add_edge(edge[0], edge[1],label=G1[e[0]][e[1]][0]['weight'], weight=G1[e[0]][e[1]][0]['weight'])
                #G1[edge[0]][edge[1]][0]['weight'] = G1[edge[0]][edge[1]][0]['weight']
```

Şekil 15

Şekil 15 te euler olmayan graf işleme alınmış ve tek dereceli düğümler, çift dereceli düğümler haline gelmek üzere diğer fonksiyonlara gönderilmiştir.

```
        # G1[edge[0]][edge[1]][1]['weight'] = G1[edge[0]][edge[1]][0]['weight']
def cost(G1, ret_list):

    while sayi < len(ret_list)-1:
        min_maliyet = 10000
        min_sira = -1
        for ind,k in enumerate(kombs):
            maliyet1=nx.shortest_path_length(G1,k[0],k[1],'weight')
            maliyet2=nx.shortest_path_length(G1,kombs[sayi-ind][0],kombs[sayi-ind][1],'weight')
            if(maliyet1+maliyet2 < min_maliyet):
                min_maliyet = maliyet1+maliyet2
                min_sira = ind
        sp1 = nx.shortest_path(G1, source=kombs[min_sira][0], target=kombs[min_sira][1], weight='weight')

        while sayi < len(ret_list)-1:
            sp2 = nx.shortest_path(G1, source=kombs[sayi - min_sira][0], target=kombs[sayi -min_sira][1], weight='weight')
            print("in_sira",min_sira)
```

Şekil 16

Şekil 16 da maliyet hesaplama fonksiyonu yazılmıştır.

```
class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]
    def minDistance(self, dist, sptSet):

        min = sys.maxsize

        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v

        return min_index

    def dijkstra(self, src):

        dist = [sys.maxsize] * self.V
        dist[src] = 0
        sptSet = [False] * self.V
        path = [0] * self.V

        for cout in range(self.V):

            u = self.minDistance(dist, sptSet)

            sptSet[u] = True

            for v in range(self.V):
                if self.graph[u][v] > 0 and sptSet[v] == False and dist[v] > dist[u] + self.graph[u][v]:
                    dist[v] = dist[u] + self.graph[u][v]
                    path[v] = u

        return dist, path

    def length_from_source(self):
        distance = []
        for i in range(self.V):
            distance.append(self.dijkstra(i)[0])
           # print("distance", distance)
        return distance

    def path(self):
        path = []
        for i in range(self.V):
            path.append(self.dijkstra(i)[1])
        return path
```

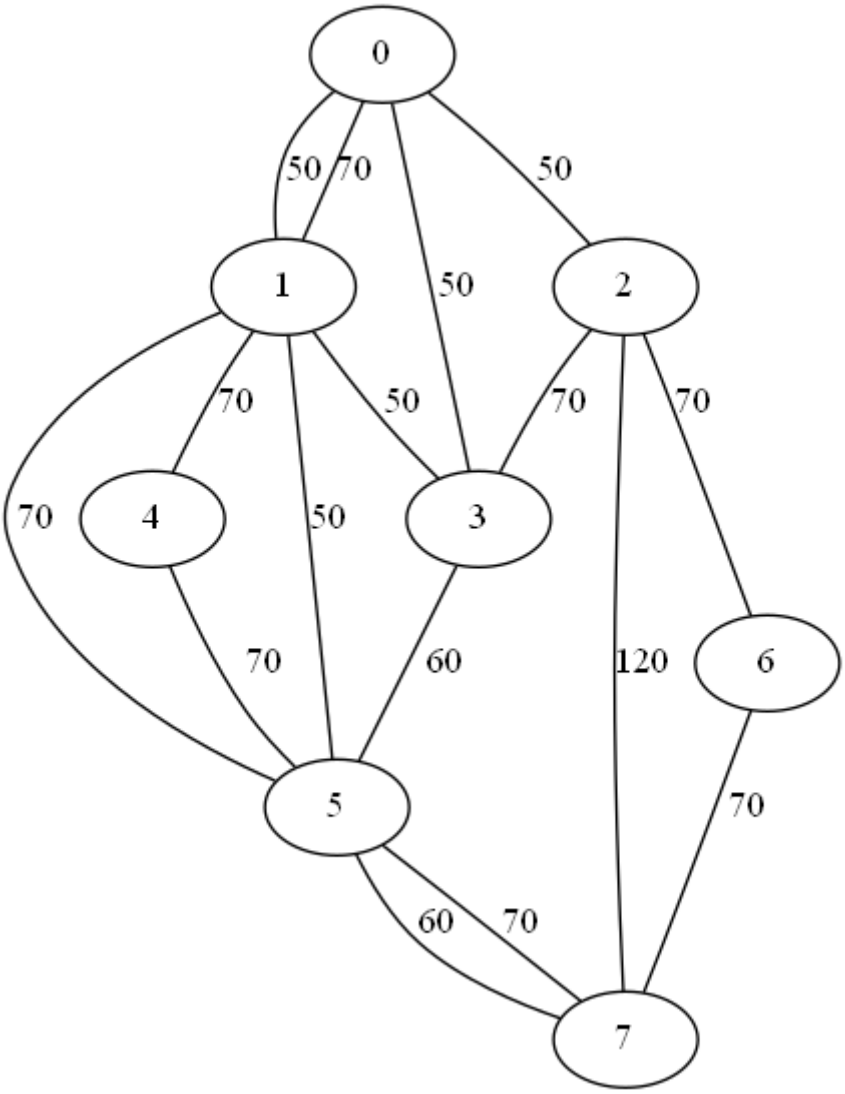
Şekil 17

Şekil 17 de Graph sınıfı oluşturulmuştur. Yolun uzunluğu, mesafe, düğümler burada işleme aluınmıştır.

```
get_shortest_walk(G1)
print(nx.is_eulerian(G1))
A = nx.drawing.nx_pydot.to_pydot(G1)
A.write_png("graf.png")
```

Şekil 18

Şekil 18 de graf get_shortest_walk fonksiyonuna gönderilerek euler graf haline getirilmiştir. Daha sonra graf .png olarak kaydedilmiştir.



Şekil 19

Artık Euler grafı olan, grafın şekli Şekil 19 da gösterilmiştir.

```
In [2]: runfile('C:/Users/Elif/Desktop/Yüksek Lisans/2. Dönem/Graf Teorisi/
Hafta6/ders.py', wdir='C:/Users/Elif/Desktop/Yüksek Lisans/2. Dönem/Graf
Teorisi/Hafta6')
**********************
False
**********************
[0]
[0, 7]
od_list _
**********************
singular_ord [[0, 7]]
ret_lis [[[0, 7]]]
singular_ord [[0, 7]]
od_list 0
**********************
ret_lis [[[0, 7]]]
path [[0, 0, 0, 0, 1, 1, 2, 5], [1, 0, 0, 1, 1, 1, 2, 5], [2, 0, 0, 2, 1, 3, 2,
2], [3, 3, 3, 0, 1, 3, 2, 5], [1, 4, 0, 1, 0, 4, 7, 5], [1, 5, 3, 5, 5, 0, 7,
5], [2, 0, 6, 2, 5, 7, 0, 6], [1, 5, 7, 5, 5, 7, 7, 0]]
True
```

Şekil 20

Şekil 20 de ilk olarak euler olmayan graf, sonrasında euler graf haline getirilmiştir.