Kuwait University

College of Engineering and Petroleum

Computer Engineering Department

CpE-Course No. 0612465: Design Automation Of Digital Circuits.

Semester: Fall 2021

Section No. 01A

Assignment No. Project Report -Design Phase

Student Name: Yassine Serrid & Rami Saleh

Student Id: 2181156439 & 2161122487

Instructor Name: Prof. Sami Habib

TA Name: Eng. Zainab Bahbahani

Date: 25th Of January 2022

Contents

Table Of Figures

0. Table Of Terminology And Definitions:

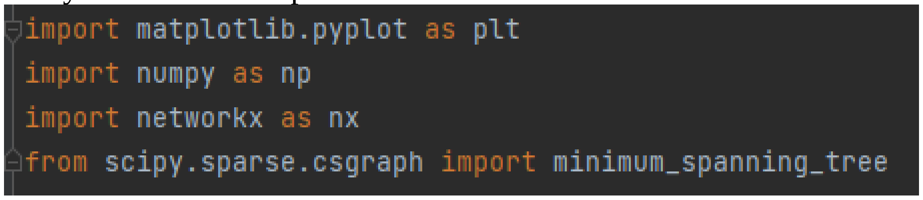
MST	Minimum Spanning Tree
undirected connected graph	An undirected graph is graph, i.e., a set of objects (called vertices or nodes) that are connected, where all the edges are bidirectional
adjacent nodes	Two node or vertices are adjacent if they are connected to each other through an edge.
degree of nodes	the number of connections that it has to other nodes in the network.
connectivity of the matrix	number of rows and cells equivalent to the number of nodes in the network.
BFS	BFS stands for Breadth First Search is a vertex-based technique for finding a shortest path in graph.
Kruskal's algorithm	Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph.

1. Problem Description:

This project aims to illustrate the dense area of the MST (Minimum Spanning Tree) by clustering the adjacent nodes, starting with the node that has the highest degree and highlighting each cluster with a specific color. To do that we have to calculate and recalculate the degree of all nodes for the MST. In this project we started with three programming assignments. In the first programming assignment, we generated a 2D array (matrix) of size NxN, where N is the number of nodes that will be selected by the user (only integer numbers are accepted and we will test our code on 25, 50, 75, 100 nodes). The matrix contains zeros and ones. In the original matrix of zeros and ones (zeros represent no connections and ones represent connections), in the adjacency matrix, 1 represents that there is an edge from node A to node B, and 0 represents that there is no edge from B to A. In the second programming assignment, we validated the connectivity of the matrix by using the NumPy library by filling in the diagonal of the original matrix with zeros to ensure no cycles in the matrix and we used BFS (Breadth First Search) that uses queue data structure for finding the shortest path It uses a queue to keep track of the next location to visit. We made the upper and lower triangles of the matrix symmetric so that we can ensure that the matrix is an undirected graph, symmetric and connected. In assignment three, we generated an extra matrix that contains the weight of the edges where the number of edges equals N-1. The weight of the edges ranges between 0 and 100. After that, we replaced the ones in the upper triangle that reflect the lower triangle in the original matrix with the weight (the upper triangle ones and the lower triangle ones have the same value, so we can ensure that the matrix is symmetric) . Kruskal's algorithm was used to find the MST because the SciPy library provides us with a built-in method to generate the MST from the matrix that contains the weight of the edges.

2. Python Libraries And Data Structures (Design Approach):

In this project we implemented our code by using the Python programming language. Python is known for its powerful libraries that can minimize effort and time for any programming implementation. For this project, we are going to use four powerful libraries as shown below in Figure 1 to design our project. Each library is used to do a specific task.



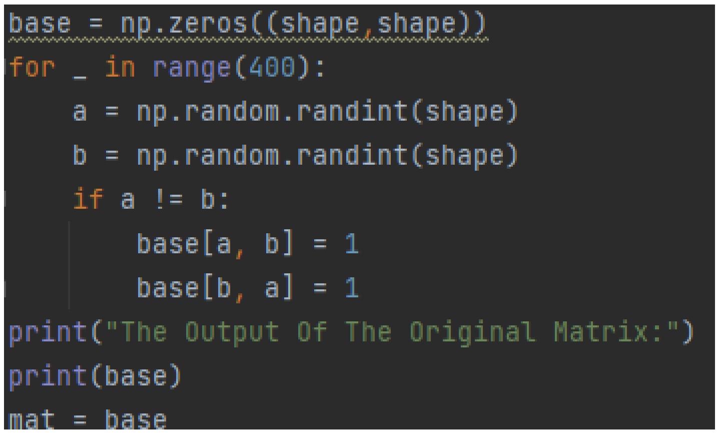
```
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from scipy.sparse.csgraph import minimum_spanning_tree
```

Figure 1. Libraries Used To Implement The Project

2.1 NumPy Library:

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

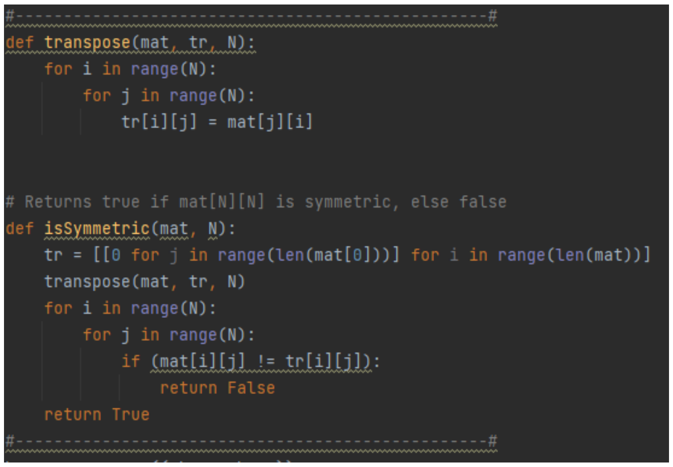
We used NumPy to generate an NxN matrix of zeros by using np.zeros(N,N) to return a new array of a given shape, filled with zeros. We located the ones in random indices, provided that this location is not on the diagonal of this matrix and is symmetrical on the matrix's lower and upper triangles, as shown in Figure 2.



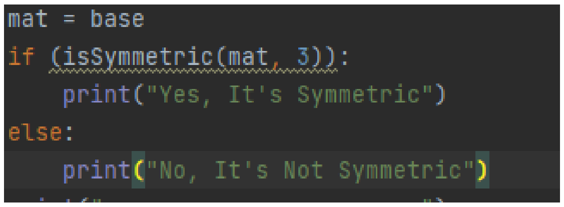
```python
base = np.zeros((shape,shape))
for _ in range(400):
    a = np.random.randint(shape)
    b = np.random.randint(shape)
    if a != b:
        base[a, b] = 1
        base[b, a] = 1
print("The Output Of The Original Matrix:")
print(base)
mat = base
```

Figure 2. The Implementation That Generates An NxN Matrix That Is Symmetric And Whose Diagonal Is Zero.

And to make sure that this matrix is symmetric, we used a function to check its symmetry, and it prints if the original matrix that contains ones and zeros is symmetrical or not, as shown below in Figures 3 and4.



```python
#-------------------------------------------------#
def transpose(mat, tr, N):
    for i in range(N):
        for j in range(N):
            tr[i][j] = mat[j][i]


# Returns true if mat[N][N] is symmetric, else false
def isSymmetric(mat, N):
    tr = [[0 for j in range(len(mat[0]))] for i in range(len(mat))]
    transpose(mat, tr, N)
    for i in range(N):
        for j in range(N):
            if (mat[i][j] != tr[i][j]):
                return False
    return True
#-------------------------------------------------#
```

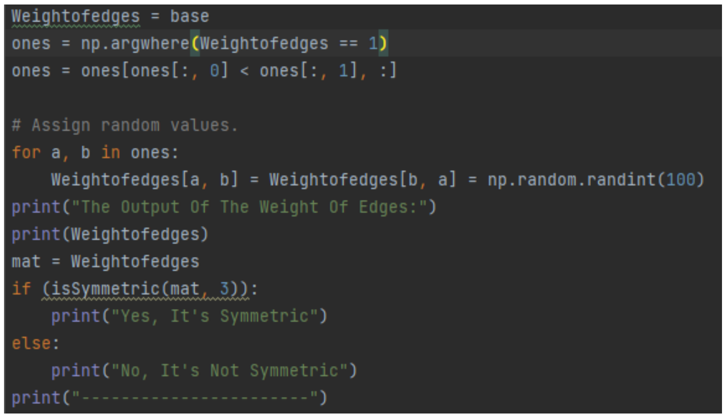


```python
mat = base
if (isSymmetric(mat, 3)):
    print("Yes, It's Symmetric")
else:
    print("No, It's Not Symmetric")
```

Figure 3. Calling The Functions To Check The Symmetricity Of The Matrix

Figure 4. Two Functions To Ensure That The Matrix Is Symmetric.

To generate the matrix that contains the weight of the edges. We used the NumPy built it function np.argwhere(condition) to find the indices of non-zero array elements, grouped by element, and then we generated the weight of the edges, which ranged between 0 and 100. Then, in the upper triangle, we replaced the ones that reflect the ones in the lower triangle in the original matrix with the weights (the upper triangle ones and the lower triangle ones have the same value, so we can ensure that the matrix is symmetric) as shown in Figure 5. To ensure the symmetry of the new matrix with the weight of the edges, we used the two functions mentioned above in Figures 3 and 4. We used BFS to ensure that the graph is undirected and there are no cycles in the matrix.



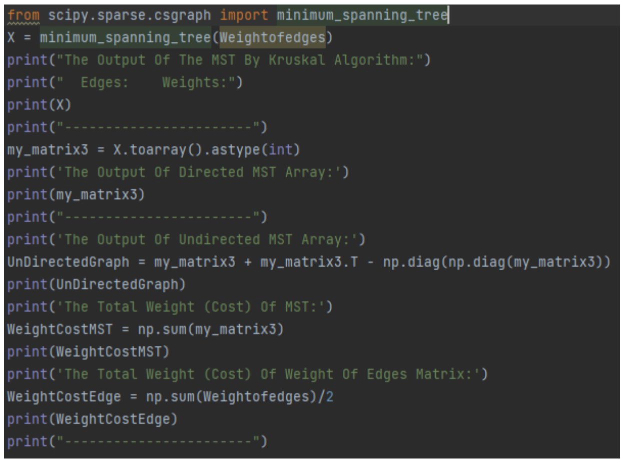
```
Weightofedges = base
ones = np.argwhere(Weightofedges == 1)
ones = ones[ones[:, 0] < ones[:, 1], :]

# Assign random values.
for a, b in ones:
    Weightofedges[a, b] = Weightofedges[b, a] = np.random.randint(100)
print("The Output Of The Weight Of Edges:")
print(Weightofedges)
mat = Weightofedges
if (isSymmetric(mat, 3)):
    print("Yes, It's Symmetric")
else:
    print("No, It's Not Symmetric")
print("----------------------")
```

Figure 5. Matrix That Contains The Weight Of The Edges And Checks Its Symmetricity

2.2 SciPy Library:

SciPy in Python is an open-source library used for solving mathematical, scientific, engineering, and technical problems. It allows users to manipulate the data and visualize the data using a wide range of high-level Python commands. SciPy is built on the Python NumPy extension. SciPy is also pronounced as “Sigh Pi.”

This library is powerful. We got the minimum spanning tree (MST) by only giving the matrix with the weight of the edges in the parameter. "from scipy.sparse.csgraph import minimum_spanning_tree" will return the undirected graph's minimum spanning tree. A minimum spanning tree is a graph consisting of the subset of edges that together connect all connected nodes, while minimizing the total sum of weights on the edges. This is computed using the **Kruskal** algorithm. We used toarray() to return a dense ndarray representation of this matrix, which we will use to illustrate the MST.



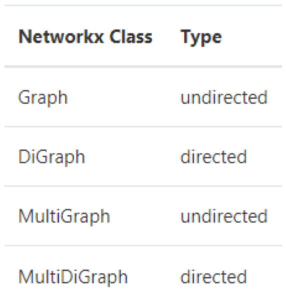
```
from scipy.sparse.csgraph import minimum_spanning_tree
X = minimum_spanning_tree(Weightofedges)
print("The Output Of The MST By Kruskal Algorithm:")
print("  Edges:    Weights:")
print(X)
print("----------------------")
my_matrix3 = X.toarray().astype(int)
print('The Output Of Directed MST Array:')
print(my_matrix3)
print("----------------------")
print('The Output Of Undirected MST Array:')
UnDirectedGraph = my_matrix3 + my_matrix3.T - np.diag(np.diag(my_matrix3))
print(UnDirectedGraph)
print('The Total Weight (Cost) Of MST:')
WeightCostMST = np.sum(my_matrix3)
print(WeightCostMST)
print('The Total Weight (Cost) Of Weight Of Edges Matrix:')
WeightCostEdge = np.sum(Weightofedges)/2
print(WeightCostEdge)
print("----------------------")
```

Figure 6. MST And The Dense ND array Implementation.

2.3 Networkx Library:

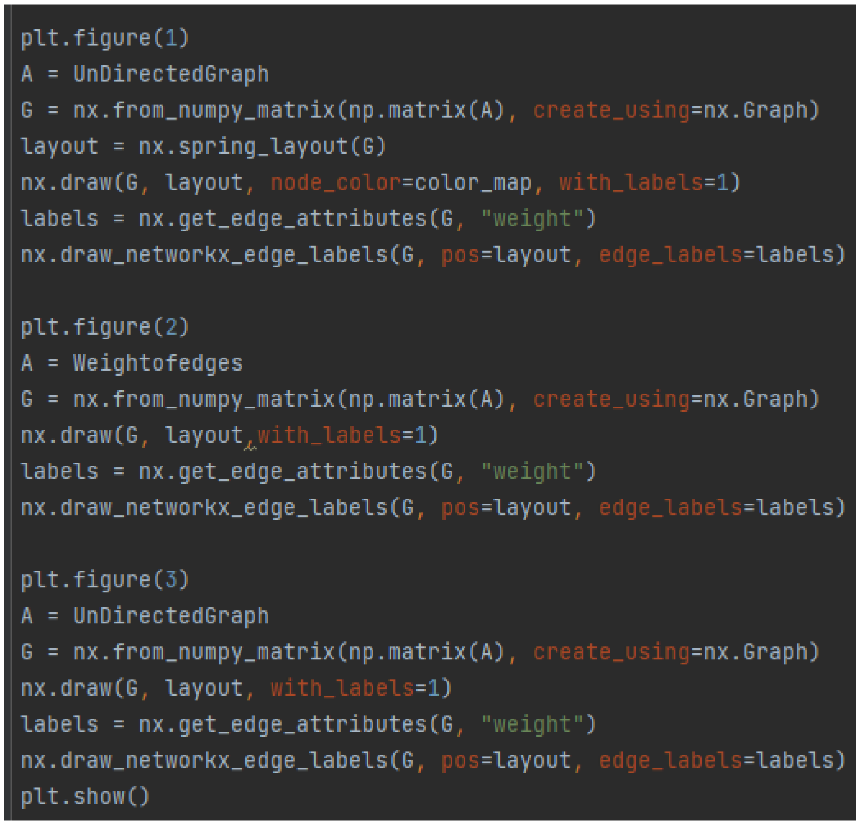
NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

To illustrate the MST, we used this powerful library and its built-in functions, like this one, which will return a graph from a NumPy matrix. The NumPy matrix is interpreted as an adjacency matrix for the graph "G = nx.from_numpy_matrix (A, create_using=None (Use specified graph for result like shown in Figure 7.)"



Networkx Class	Type
Graph	undirected
DiGraph	directed
MultiGraph	undirected
MultiDiGraph	directed

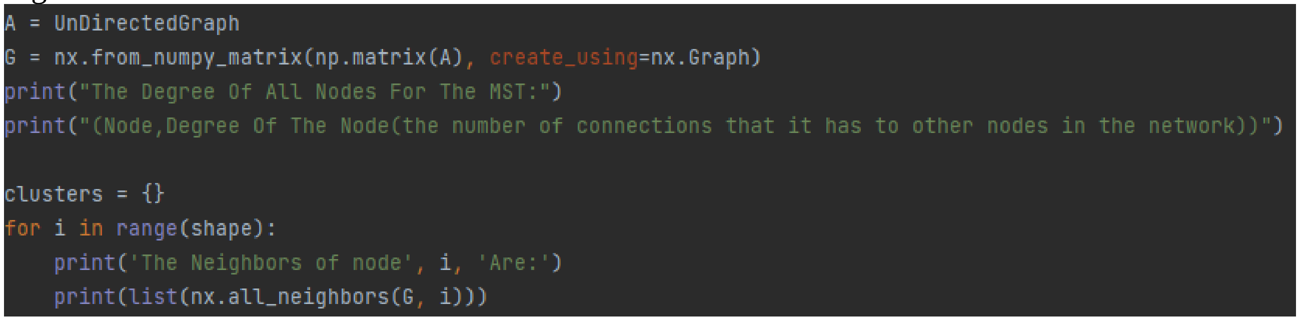
Figure 7. NetworkX Graph Class And Type



```
plt.figure(1)
A = UnDirectedGraph
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
layout = nx.spring_layout(G)
nx.draw(G, layout, node_color=color_map, with_labels=1)
labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=labels)

plt.figure(2)
A = Weightofedges
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
nx.draw(G, layout,with_labels=1)
labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=labels)

plt.figure(3)
A = UnDirectedGraph
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
nx.draw(G, layout, with_labels=1)
labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=labels)
plt.show()
```

Figure 8. Illustration Of The Weight Of edges graph, MST And MST Partitioning The Circuit.

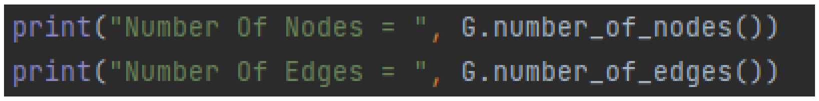
To calculate the degree of all nodes (the number of connections that it has to other nodes in the network) for the MST we used G.degree() built-in function in NetworkX library as shown below in Figure 9.



```
A = UnDirectedGraph
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
print("The Degree Of All Nodes For The MST:")
print("(Node,Degree Of The Node(the number of connections that it has to other nodes in the network))")

clusters = {}
for i in range(shape):
    print('The Neighbors of node', i, 'Are:')
    print(list(nx.all_neighbors(G, i)))
```

Figure 9. Calculating the degree of all nodes for the MST.

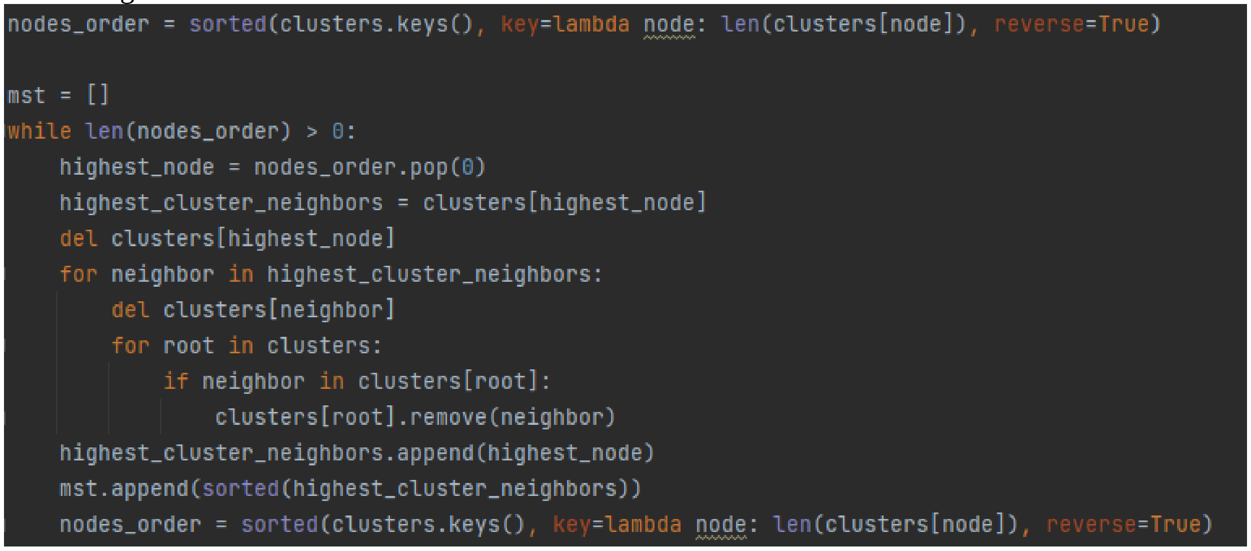
We used G.number_of_nodes() and G.number_of_edges() to calculate the number of nodes and edges for each graph generated and we found the neighbors for each node so that we can cluster the nodes and find the dense area.



```
print("Number Of Nodes = ", G.number_of_nodes())
print("Number Of Edges = ", G.number_of_edges())
```

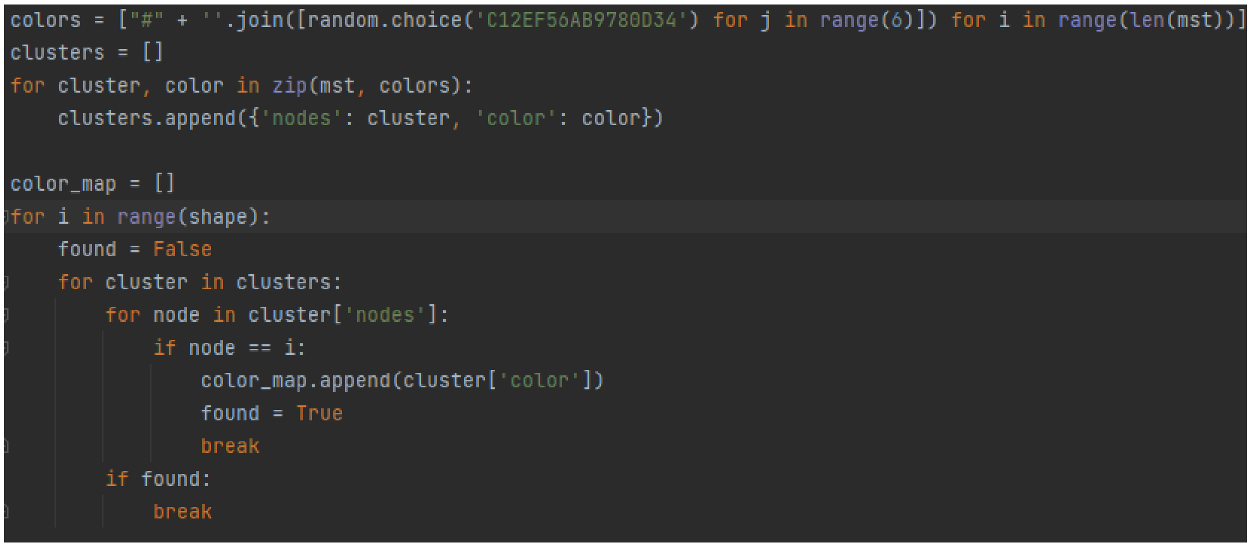
Figure 10. Calculating The Total Number Of Edges And Nodes.

We used list data structure so we can store the nodes and its neighbors and sort them to cluster the nodes only one time and give the node with highest degree and its neighbor nodes the same color generated randomly as shown in Figure 12.



```
nodes_order = sorted(clusters.keys(), key=lambda node: len(clusters[node]), reverse=True)

mst = []
while len(nodes_order) > 0:
    highest_node = nodes_order.pop(0)
    highest_cluster_neighbors = clusters[highest_node]
    del clusters[highest_node]
    for neighbor in highest_cluster_neighbors:
        del clusters[neighbor]
        for root in clusters:
            if neighbor in clusters[root]:
                clusters[root].remove(neighbor)
    highest_cluster_neighbors.append(highest_node)
    mst.append(sorted(highest_cluster_neighbors))
    nodes_order = sorted(clusters.keys(), key=lambda node: len(clusters[node]), reverse=True)
```

Figure 11. Clustering The Nodes



```
colors = ["#" + ''.join([random.choice('C12EF56AB9780D34') for j in range(6)]) for i in range(len(mst))]
clusters = []
for cluster, color in zip(mst, colors):
    clusters.append({'nodes': cluster, 'color': color})

color_map = []
for i in range(shape):
    found = False
    for cluster in clusters:
        for node in cluster['nodes']:
            if node == i:
                color_map.append(cluster['color'])
                found = True
                break
        if found:
            break
```

Figure 12. Generating Random Colors

2.4 Matplotlib Library:

Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.
In this simple library, we used the built-in function plt.show() as shown in Figure 8, to plot the graph that is created by the NetworkX library and display it on the window screen as shown in Figures 16 and 17

3. Flowchart Of The Design:

All steps in this flow chart were explained in section 1 and 2 in this report .

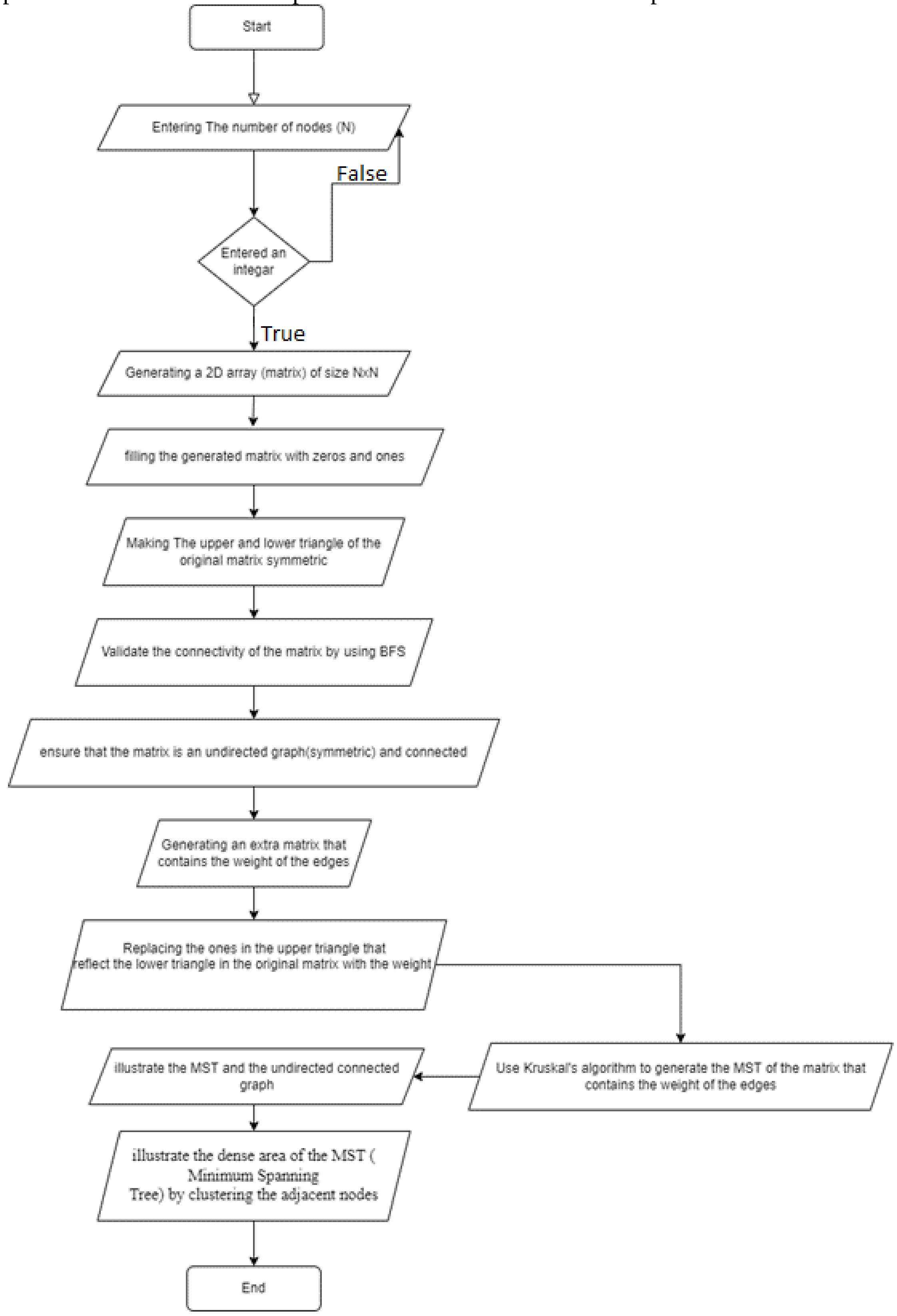

Start
Entering The number of nodes (N)
False
Entered an integar
True
Generating a 2D array (matrix) of size NxN
filling the generated matrix with zeros and ones
Making The upper and lower triangle of the original matrix symmetric
Validate the connectivity of the matrix by using BFS
ensure that the matrix is an undirected graph(symmetric) and connected
Generating an extra matrix that contains the weight of the edges
Replacing the ones in the upper triangle that reflect the lower triangle in the original matrix with the weight
Use Kruskal's algorithm to generate the MST of the matrix that contains the weight of the edges
illustrate the MST and the undirected connected graph
illustrate the dense area of the MST (Minimum Spanning Tree) by clustering the adjacent nodes
End

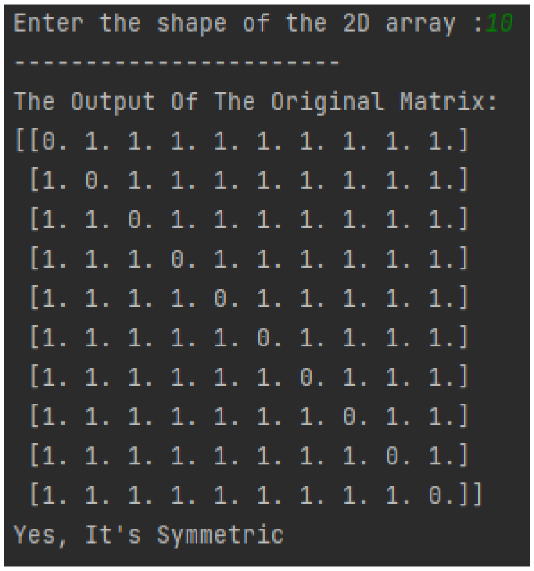
Figure 13. Flowchart Of The Design.

4. Sample Output:

This is a sample output of our design where N=10 (N == Number of Nodes).

4.1 The Output Of The Original Matrix:

Here we generated a 2D array (matrix) of size NxN containing zeros and ones (only integer numbers are accepted: 25, 50, 75, 100 nodes), In the adjacency matrix, 1 represents that there is an edge from node A to node B, and 0 represents that there is no edge from B to A. We can validate the connectivity of the matrix because the diagonal of the original matrix is filled with zeros. And we made the upper and lower triangles of the matrix symmetric. We checked the symmetry of our function too, to be sure that our matrix is symmetric.

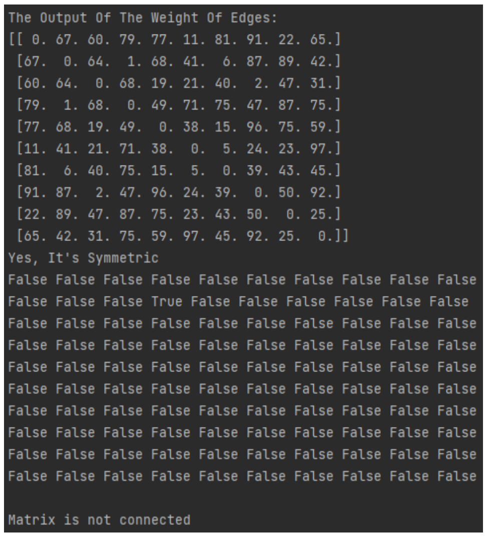


```
Enter the shape of the 2D array :10
-----------------------
The Output Of The Original Matrix:
[[0. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 0. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 0. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 0. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 0. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 0. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 0. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 0. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]]
Yes, It's Symmetric
```

Figure 14. The Output Of The Original Matrix.

4.2 The Output Of The Weight Of Edge And BFS.

We generated an extra matrix that contains the weight of the edges where the number of edges equals N-1.The weight of the edges ranges between 0 and 100. By replacing the ones in the upper triangle that reflect the lower triangle in the original matrix with the weight ,(the upper triangle ones and the lower triangle ones have the same value, so we can ensure that the matrix is symmetric).And we checked the symmetricity by our function too to be sure that our matrix that contain the weight of the edges is symmetric.

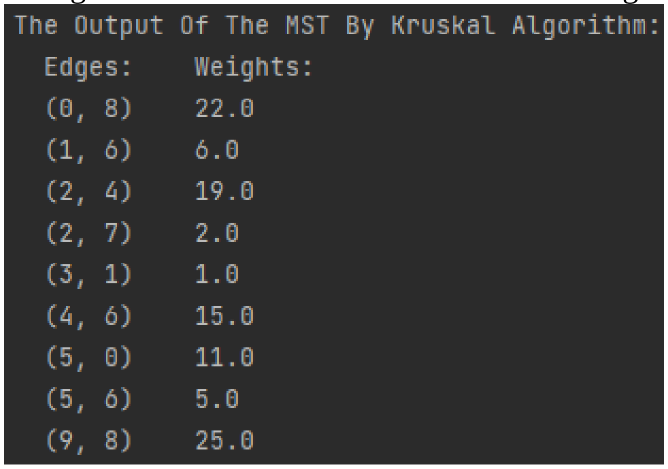


```
The Output Of The Weight Of Edges:
[[ 0. 67. 60. 79. 77. 11. 81. 91. 22. 65.]
 [67.  0. 64.  1. 68. 41.  6. 87. 89. 42.]
 [60. 64.  0. 68. 19. 21. 40.  2. 47. 31.]
 [79.  1. 68.  0. 49. 71. 75. 47. 87. 75.]
 [77. 68. 19. 49.  0. 38. 15. 96. 75. 59.]
 [11. 41. 21. 71. 38.  0.  5. 24. 23. 97.]
 [81.  6. 40. 75. 15.  5.  0. 39. 43. 45.]
 [91. 87.  2. 47. 96. 24. 39.  0. 50. 92.]
 [22. 89. 47. 87. 75. 23. 43. 50.  0. 25.]
 [65. 42. 31. 75. 59. 97. 45. 92. 25.  0.]]
Yes, It's Symmetric
False False False False False False False False False False
False False False True False False False False False False
False False False False False False False False False False
False False False False False False False False False False
False False False False False False False False False False
False False False False False False False False False False
False False False False False False False False False False
False False False False False False False False False False
False False False False False False False False False False
False False False False False False False False False False

Matrix is not connected
```

Figure 15. The Output Of The Weight Of Edge And BFS.

4.3 The Output Of The MST By Kruskal Algorithm:

We used Kruskal's algorithm by using SciPy library to generate the MST of the matrix that contains the weights of the edges. Here we can see the nodes and the weights between those nodes.

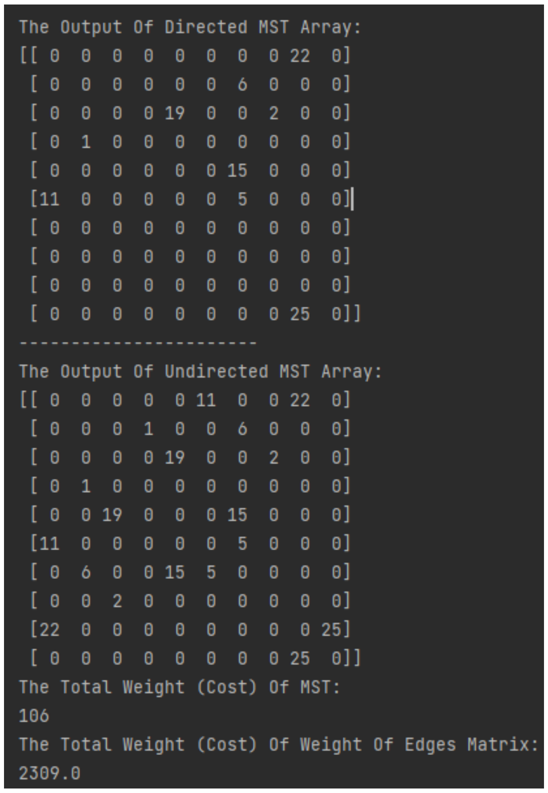


```
The Output Of The MST By Kruskal Algorithm:
  Edges:    Weights:
  (0, 8)    22.0
  (1, 6)    6.0
  (2, 4)    19.0
  (2, 7)    2.0
  (3, 1)    1.0
  (4, 6)    15.0
  (5, 0)    11.0
  (5, 6)    5.0
  (9, 8)    25.0
```

Figure 16. The Output Of The MST By Kruskal Algorithm.

4.4 The Output Of Directed And Undirected MST Array :

We used Kruskal's algorithm (the SciPy library .toarray()) to generate the MST of the matrix that contains the weight of the edges. .toarray() is used to return a dense ndarray representation of this matrix, which we will use to illustrate the MST.

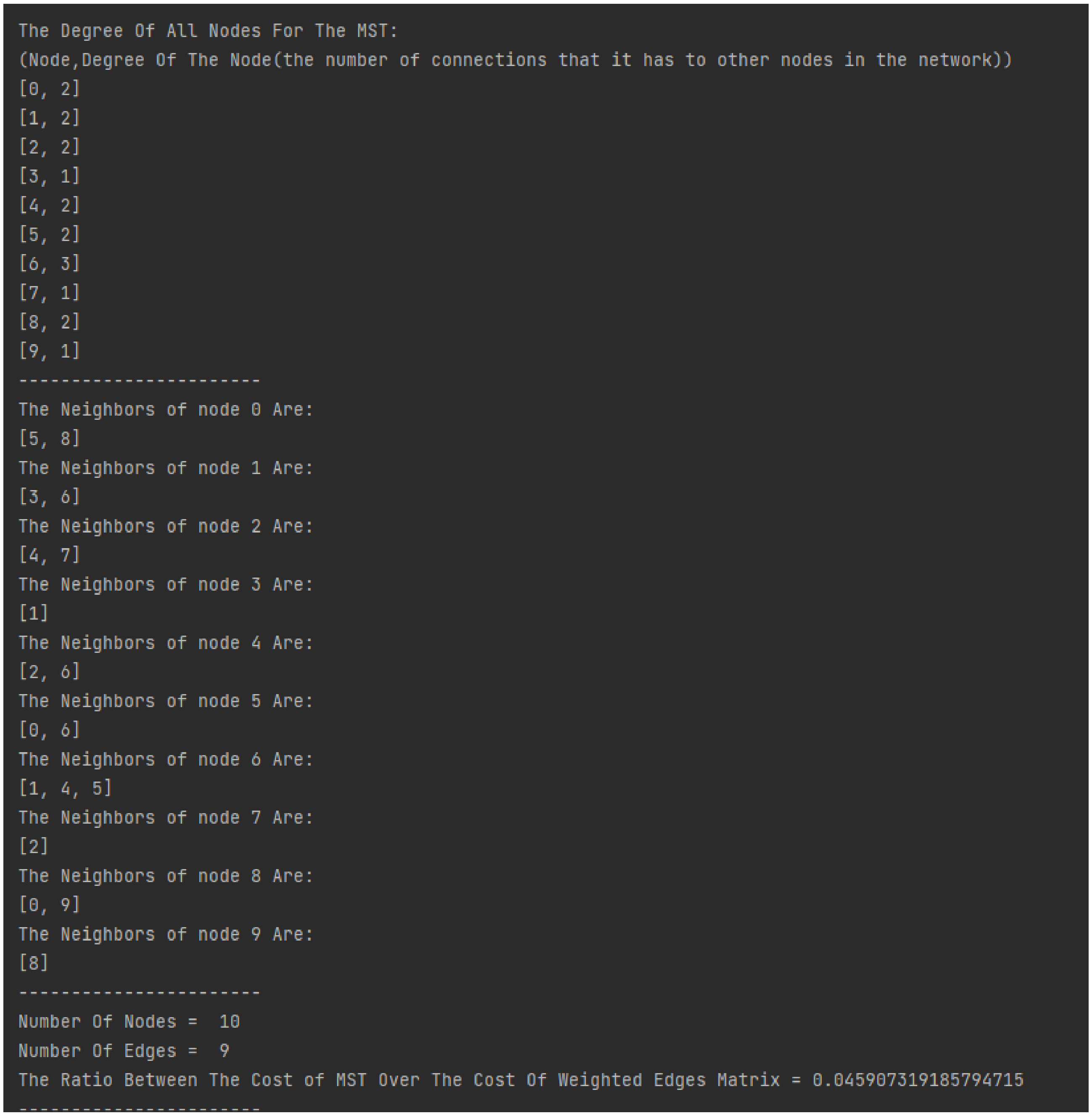


```
The Output Of Directed MST Array:
[[ 0  0  0  0  0  0  0  0 22  0]
 [ 0  0  0  0  0  0  6  0  0  0]
 [ 0  0  0  0 19  0  0  2  0  0]
 [ 0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 15  0  0  0]
 [11  0  0  0  0  0  5  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 25  0]]
----------------------
The Output Of Undirected MST Array:
[[ 0  0  0  0  0 11  0  0 22  0]
 [ 0  0  0  1  0  0  6  0  0  0]
 [ 0  0  0  0 19  0  0  2  0  0]
 [ 0  1  0  0  0  0  0  0  0  0]
 [ 0  0 19  0  0  0 15  0  0  0]
 [11  0  0  0  0  0  5  0  0  0]
 [ 0  6  0  0 15  5  0  0  0  0]
 [ 0  0  2  0  0  0  0  0  0  0]
 [22  0  0  0  0  0  0  0  0 25]
 [ 0  0  0  0  0  0  0  0 25  0]]
The Total Weight (Cost) Of MST:
106
The Total Weight (Cost) Of Weight Of Edges Matrix:
2309.0
```

Figure 17. The Output Of Directed And Undirected MST Array.

4.5 The Degree Of All Nodes For The MST And The Neighbors:

This sample output represents the degree of the nodes for an MST and the Neighbor for each node that we used to cluster the nodes. We calculated the total number of nodes and edges using NetworkX and we calculated the ratio between the cost of the weighted edge matrix over the cost of the MST.



```
The Degree Of All Nodes For The MST:
(Node,Degree Of The Node(the number of connections that it has to other nodes in the network))
[0, 2]
[1, 2]
[2, 2]
[3, 1]
[4, 2]
[5, 2]
[6, 3]
[7, 1]
[8, 2]
[9, 1]
----------------------
The Neighbors of node 0 Are:
[5, 8]
The Neighbors of node 1 Are:
[3, 6]
The Neighbors of node 2 Are:
[4, 7]
The Neighbors of node 3 Are:
[1]
The Neighbors of node 4 Are:
[2, 6]
The Neighbors of node 5 Are:
[0, 6]
The Neighbors of node 6 Are:
[1, 4, 5]
The Neighbors of node 7 Are:
[2]
The Neighbors of node 8 Are:
[0, 9]
The Neighbors of node 9 Are:
[8]
----------------------
Number Of Nodes =  10
Number Of Edges =  9
The Ratio Between The Cost of MST Over The Cost Of Weighted Edges Matrix = 0.045907319185794715
```

Figure 18. The Degree Of All Nodes For The MST And The Neighbors.

4.6 Undirected Weighted Connected Graph Illustration using both NetworkX and Matplotlib Libraries:

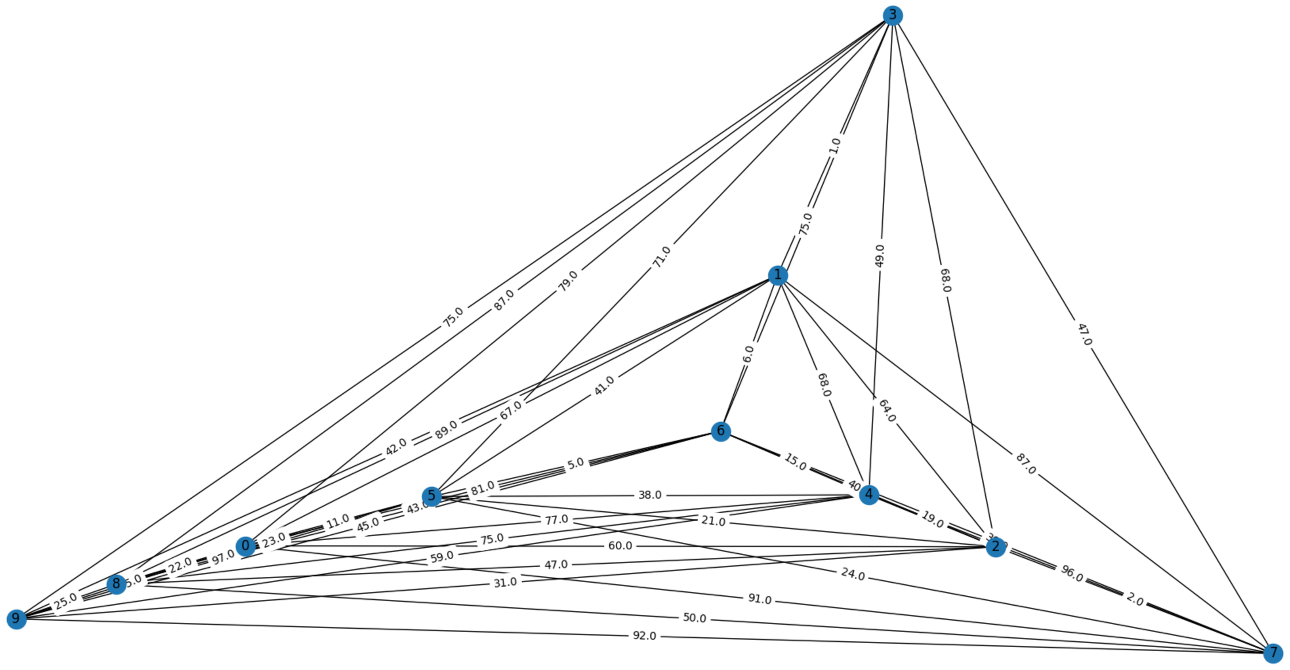


Figure 19. Undirected Connected Graph (Weighted) Illustration.

4.7 Minimum Spanning Tree Illustration (Kruskal's Algorithm):

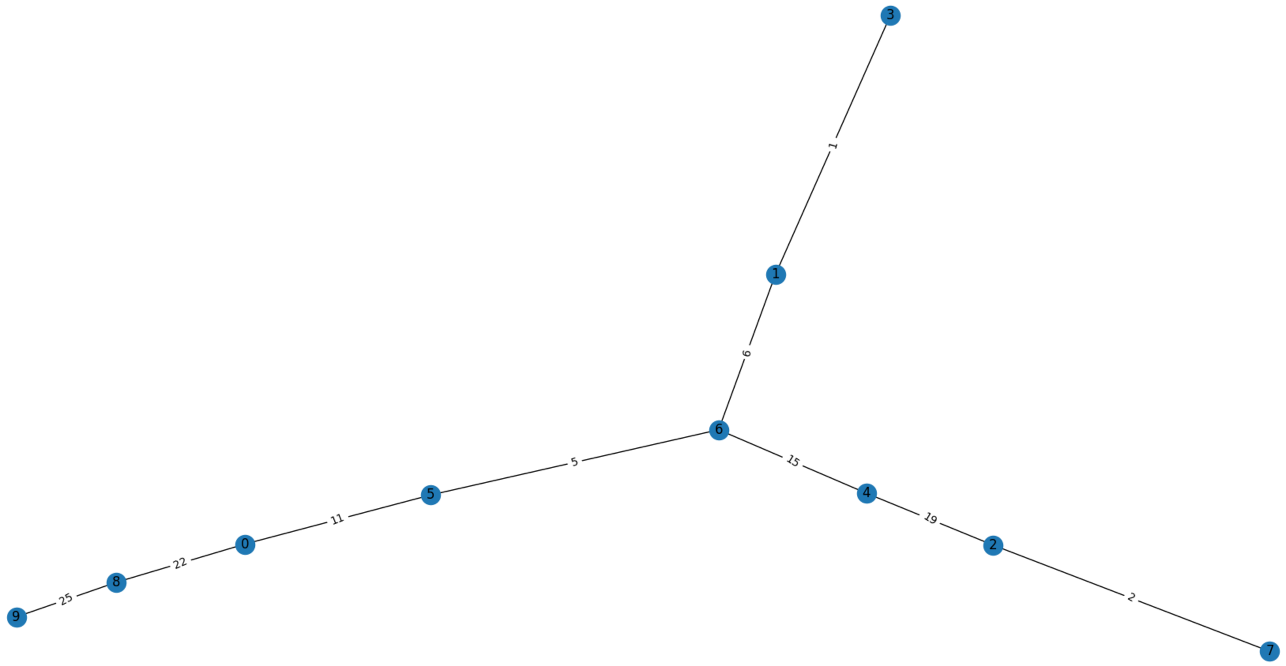


Figure 20. Minimum Spanning Tree Illustration (Kruskal's Algorithm)

4.8 Partitioning The MST:

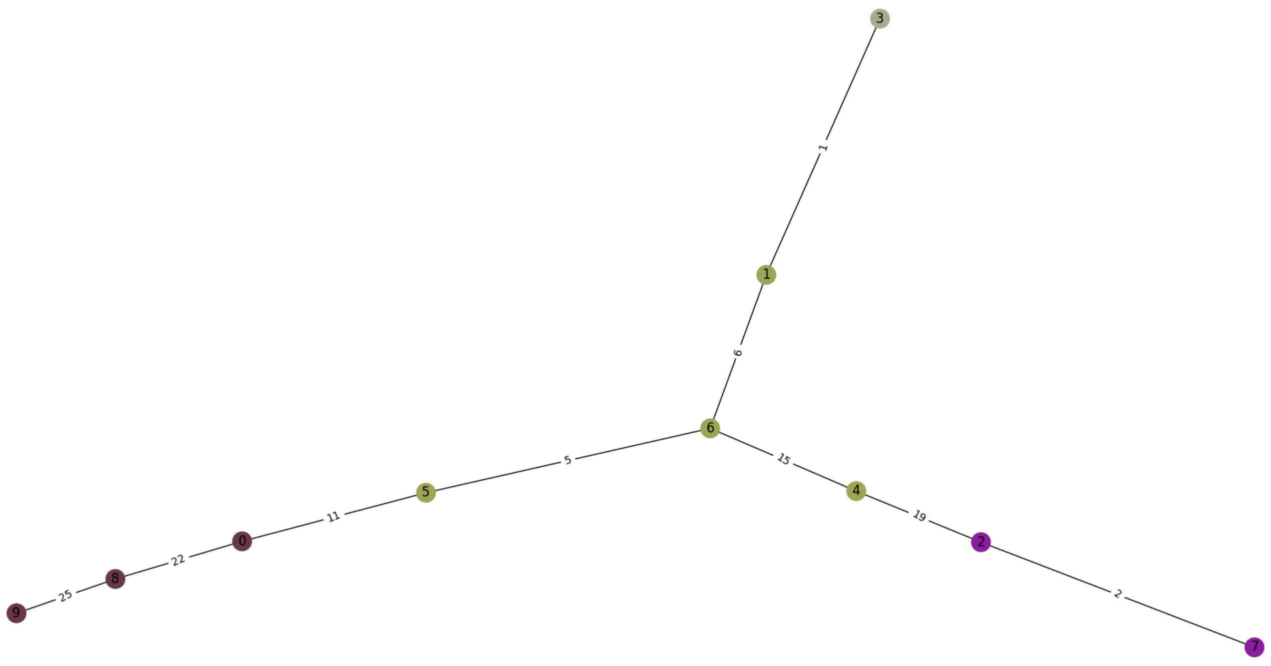


Figure 21. Partitioning The MST.

4.9 Recorded Outputs (25,50,75,100 nodes) :

https://www.youtube.com/watch?v=TR1C-1rGGN8

Figure 22. Video Of Recorded Outputs.

5. Conclusion:

From this project we can conclude that finding the minimum spanning tree with either Prim's or Kruskal's algorithms and clustering the nodes with the highest degree and finding its neighbors can help us to find the dense area and partition the graph and work with each partition alone and know the cluster with highest dense and know its cost.

6. Source Code:

```
# Yassine Nader Serrid 2181156439
# Rami Mohammed Saleh 2161122487
# Final Project
import random

import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from collections import deque as queue
import sys
np.set_printoptions(threshold=sys.maxsize)
np.set_printoptions(linewidth=np.inf)

shape = None
while True:
    try:
        shape = int(input("Enter the shape of the 2D array :"))
        break
    except ValueError:
        print('Please input an integer value ...')
print("-----------------------")
#--------------------------------------------------#
def transpose(mat, tr, N):
    for i in range(N):
        for j in range(N):
            tr[i][j] = mat[j][i]


# Returns true if mat[N][N] is symmetric, else false
def isSymmetric(mat, N):
    tr = [[0 for j in range(len(mat[0]))] for i in range(len(mat))]
    transpose(mat, tr, N)
    for i in range(N):
        for j in range(N):
            if (mat[i][j] != tr[i][j]):
                return False
    return True
#--------------------------------------------------#
# make N*N matrix and initialize it will all value as false
visited = [[False for i in range(shape)] for j in range(shape)]


# function which visits all node using BFS algorithm
def matrixbfs(mat):
    # define queue
    q = queue()

    # Mark the starting cell as visited
    # and push it into the queue

    # find the first cell whose value as 1 and start BFS algorithm from
that node
    starti = 0
    startj = 0
    flag = 0
    for i in range(0, shape):
        for j in range(0, shape):
            if (mat[i][j] == 1):
                starti = i
                startj = j
                # if we found first 1 then make flag 1 and break the loop
                flag = 1
                break
            if flag == 1:
                break

    # append first node in queue
    q.append((starti, startj))
    # make it visited
    visited[starti][startj] = True

    # iterate loop until all reachable nodes are not visited
    while len(q) > 0:
        # pop front from queue
        current = q.popleft()

        x = current[0]
        y = current[1]

        # to visit above node of current node
        # check for negative value of index which is for 1st row
        if (x - 1 >= 0) and (mat[x - 1][y] == 1 and visited[x - 1][y] ==
False):
            # print(x-1," ",y)
            q.append((x - 1, y))
            visited[x - 1][y] = True

        # to visit left side node of current node
        # check for negative value of index which is for 1st column
        if (y - 1 >= 0) and (mat[x][y - 1] == 1 and visited[x][y - 1] ==
False):
            # print(x," ",y-1)
            q.append((x, y - 1))
            visited[x][y - 1] = True

        # to visit below  node of current node
        # check for index out of bound for last row
        if (y + 1 < shape) and (mat[x][y + 1] == 1 and visited[x][y + 1]
== False):
            # print(x," ",y+1)
            q.append((x, y + 1))
            visited[x][y + 1] = True

        # to visit right side node of current node
        # check for index out of bound for last column
        if (x + 1 < shape) and (mat[x + 1][y] == 1 and visited[x + 1][y]
== False):
            # print(x+1," ",y)
            q.append((x + 1, y))
            visited[x + 1][y] = True
#--------------------------------------------------#
base = np.zeros((shape,shape))
for _ in range(400):
    a = np.random.randint(shape)
    b = np.random.randint(shape)
    if a != b:
        base[a, b] = 1
        base[b, a] = 1
print("The Output Of The Original Matrix:")
print(base)
mat = base
if (isSymmetric(mat, 3)):
    print("Yes, It's Symmetric")
else:
    print("No, It's Not Symmetric")
print("-----------------------")
#--------------------------------------------------#
# Fetch the location of the 1s.
Weightofedges = base
ones = np.argwhere(Weightofedges == 1)
ones = ones[ones[:, 0] < ones[:, 1], :]

# Assign random values.
for a, b in ones:
    Weightofedges[a, b] = Weightofedges[b, a] = np.random.randint(100)
print("The Output Of The Weight Of Edges:")
print(Weightofedges)
mat = Weightofedges
if (isSymmetric(mat, 3)):
    print("Yes, It's Symmetric")
else:
    print("No, It's Not Symmetric")
matrixbfs(mat)
# print visited matrix
for i in range(shape):
    for j in range(shape):
        print(visited[i][j], end=" ")
    print()
print()

# check matrix is connected or not using visited matrix
isconnected = True
for i in range(shape):
    for j in range(shape):
        if mat[i][j] == 1 and visited[i][j] == False:
            isconnected = False
            break

# print result
if isconnected == True:
    print("Matrix is connected")
else:
    print("Matrix is not connected")
print("-----------------------")
#--------------------------------------------------#
from scipy.sparse.csgraph import minimum_spanning_tree
X = minimum_spanning_tree(Weightofedges)
print("The Output Of The MST By Kruskal Algorithm:")
print("  Edges:    Weights:")
print(X)
print("-----------------------")
my_matrix3 = X.toarray().astype(int)
print('The Output Of Directed MST Array:')
print(my_matrix3)
print("-----------------------")
print('The Output Of Undirected MST Array:')
UnDirectedGraph = my_matrix3 + my_matrix3.T - np.diag(np.diag(my_matrix3))
print(UnDirectedGraph)
print('The Total Weight (Cost) Of MST:')
WeightCostMST = np.sum(my_matrix3)
print(WeightCostMST)
print('The Total Weight (Cost) Of Weight Of Edges Matrix:')
WeightCostEdge = np.sum(Weightofedges)/2
print(WeightCostEdge)
print("-----------------------")

#-------------------------------------------------#
A = UnDirectedGraph
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
print("The Degree Of All Nodes For The MST:")
print("(Node,Degree Of The Node(the number of connections that it has to
other nodes in the network))")
for i in G.degree():
    print(list(i))

print("-----------------------")
clusters = {}
for i in range(shape):
    print('The Neighbors of node', i, 'Are:')
    print(list(nx.all_neighbors(G, i)))
    clusters[i] = list(nx.all_neighbors(G, i))

nodes_order = sorted(clusters.keys(), key=lambda node:
len(clusters[node]), reverse=True)

mst = []
while len(nodes_order) > 0:
    highest_node = nodes_order.pop(0)
    highest_cluster_neighbors = clusters[highest_node]
    del clusters[highest_node]
    for neighbor in highest_cluster_neighbors:
        del clusters[neighbor]
        for root in clusters:
            if neighbor in clusters[root]:
                clusters[root].remove(neighbor)
    highest_cluster_neighbors.append(highest_node)
    mst.append(sorted(highest_cluster_neighbors))
    nodes_order = sorted(clusters.keys(), key=lambda node:
len(clusters[node]), reverse=True)
print("-----------------------")
print("Number Of Nodes = ", G.number_of_nodes())
print("Number Of Edges = ", G.number_of_edges())
print("The Ratio Between The Cost of MST Over The Cost Of Weighted Edges
Matrix =", WeightCostMST/WeightCostEdge)
print("-----------------------")

colors = ["#" + ''.join([random.choice('C12EF56AB9780D34') for j in
range(6)]) for i in range(len(mst))]
clusters = []
for cluster, color in zip(mst, colors):
    clusters.append({'nodes': cluster, 'color': color})

color_map = []
for i in range(shape):
    found = False
    for cluster in clusters:
        for node in cluster['nodes']:
            if node == i:
                color_map.append(cluster['color'])
                found = True
                break
        if found:
            break


plt.figure(1)
A = UnDirectedGraph
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
layout = nx.spring_layout(G)
nx.draw(G, layout, node_color=color_map, with_labels=1)
labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=labels)

plt.figure(2)
A = Weightofedges
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
nx.draw(G, layout,with_labels=1)
labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=labels)

plt.figure(3)
A = UnDirectedGraph
G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.Graph)
nx.draw(G, layout, with_labels=1)
labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(G, pos=layout, edge_labels=labels)
plt.show()
#-------------------------------------------------#
```

7. Screenshot Of The Outputs:

7.1 Test 1 (25 Nodes)

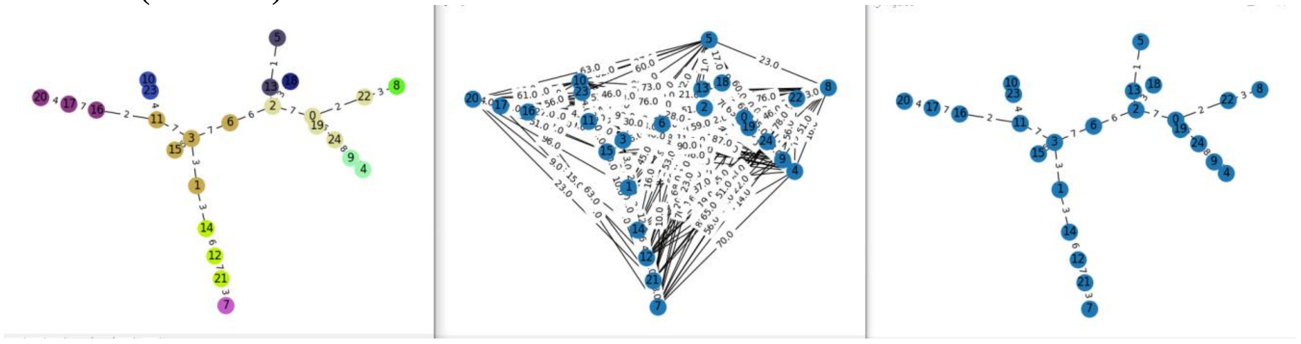


Figure 23. Test 1 (25 Nodes).

7.2 Test 2 (50 Nodes)

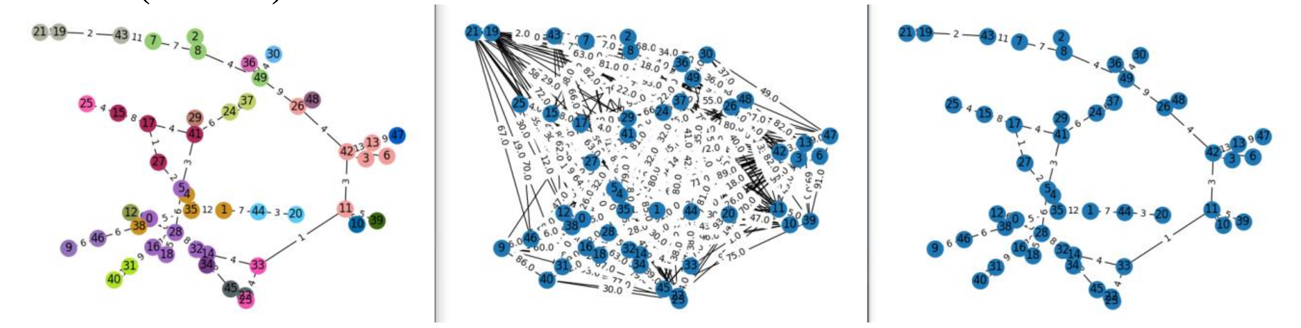


Figure 24. Test 2 (50 Nodes).

7.3 Test 3 (75 Nodes)

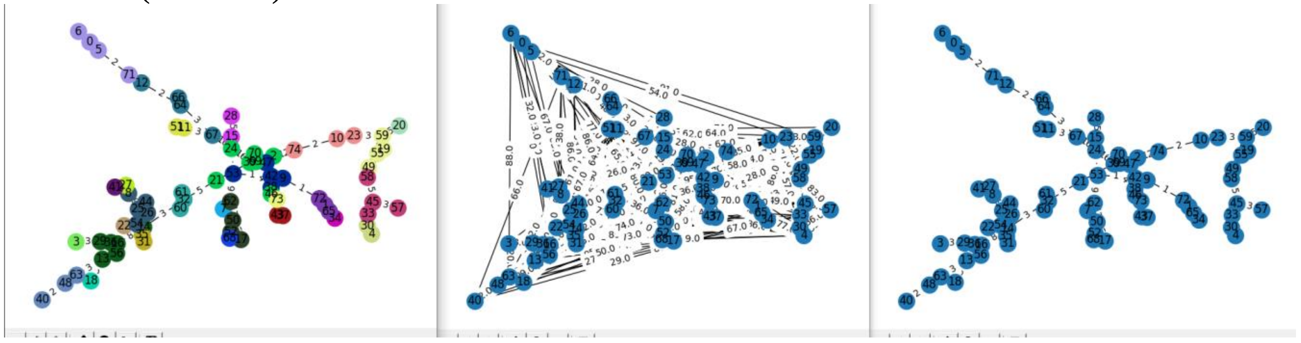


Figure 25. Test 3 (75 Nodes).

7.4 Test 4 (100 Nodes)

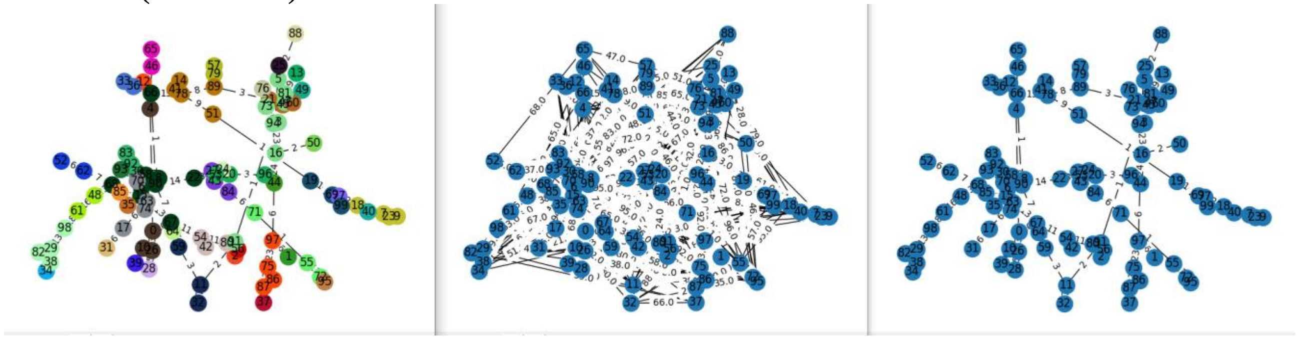


Figure 26. Test 4 (100 Nodes).

8. Time Management:

8.1 Work Time Management:

Data & Time	22/Dec/2021 – 7:30 PM
Meeting Held By	**Yassin Nader – Rami Saleh**
1- Understand Project. 2- Choose Coding Language. 3- Identify Objectives	

Data & Time	25/Dec/2021 – 4:30 PM
Meeting Held By	**Yassin Nader – Rami Saleh**
1- Start Coding. 2- Generating A 2D Array [Matrix] Of Size NxN . 3- Filling The Generated Matrix With Zeros And Ones.	

Data & Time	27/Dec/2021 – 8:30 PM
Meeting Held By	**Yassin Nader – Rami Saleh**
1- Validate The Connectivity Of The Matrix By Using BFS. 2- Ensure That The Matrix Is An Undirected Graph(symmetric) And Connected**.**	

Data & Time	10/January/2022 – 10:30 PM
Meeting Held By	**Yassin Nader – Rami Saleh**
1- Replacing The Ones In The Upper Triangle That Reflect The Lower Triangle In The Original Matrix With The Weight 2- Use Kruskal's Algorithm To Generate The MST Of The Matrix That Contains The Weight Of The Edges	

Data & Time	19/January/2021 – 4:30 PM
Meeting Held By	**Yassin Nader – Rami Saleh**
1- Illustrate The Dense Area Of The MST (Minimum Spanning Tree) By Clustering The Adjacent Nodes	

9. Resources

[1] What is NumPy? — NumPy v1.22 Manual. (2022). Retrieved 7 January 2022, from https://numpy.org/doc/stable/user/whatisnumpy.html

[2] scipy.sparse.csgraph.minimum_spanning_tree — SciPy v1.7.1 Manual. (2022). Retrieved 7 January 2022, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.minimum_spanning_tree.html

[3] random — Generate pseudo-random numbers — Python 3.10.1 documentation. (2022). Retrieved 7 January 2022, from https://docs.python.org/3/library/random.html

[4] Python Random Module. (2022). Retrieved 7 January 2022, from https://www.w3schools.com/python/module_random.asp

[5] Python Random randint() Method. (2022). Retrieved 7 January 2022, from https://www.w3schools.com/python/ref_random_randint.asp

[6] (2022). Retrieved 7 January 2022, from https://www.youtube.com/watch?v=HzkqIHqFnvM

[7] diagonal elements zero using numpy Code Example. (2022). Retrieved 7 January 2022, from https://www.codegrepper.com/code-examples/python/diagonal+elements+zero+using+numpy

[8] numpy.random.randint — NumPy v1.22 Manual. (2022). Retrieved 7 January 2022, from https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html