CS6385.0W1 - Algorithmic Aspects of Telecommunication Networks

Project – 1 Report

Submitted by

Chandan Raju Vysyaraju (cxv200012)

CONTENTS

INTRODUCTION

Communication is vital in every aspect of life. Telecommunication happens when there is exchange of information between individuals or organizations with the help of technology. Due to development in telecommunication a lot of information is being easy to attain from any location in the world. But the reliability and ability to convey the information quickly play an important role. Many ways are introduced to exchanging information quickly, comprehensively, and flexibly. One of the ways to achieve this is to map the given requirements into the networks. The quality and cost of the network implementation is most important to any organization or institution. This project is based on one of the algorithms which is used for implementing the network with low cost and more reliability.

K-CORE ALGORTIHM

The k-core of a graph is known as the largest subset of nodes in which every node in the subset has at least k neighbors for all k >= 1. This project is implemented using the k-core algorithm for all the given number of nodes. The k-core algorithm uses a symmetric matrix and a k-value as input of the function. The function is implemented from the following algorithm:

Algorithm to find the k-core:

Step-1: Calculate the degree of all the nodes that are present in the given adjacency matrix.

Step-2: Find all the nodes which have degree less than k and remove the nodes from the subgraph.

Step-3: Check whether the subgraph is empty or not. If it is not empty then compute the degree for all the nodes in subgraph and check if the degree of nodes is > k.

Step-4: If the subgraph have nodes with degree < k then again perform the algorithm to remove the nodes with degree < k.

Step-5: If the subgraph has degree > k for each node then provide the list of nodes as output of the function.

Input:

- A positive integer k
- Adjacency matrix that is generated using UTD student id.

Output:

- List of nodes with at least k neighbors.

IMPLEMENTATION

This project is implemented using the Python programming language by executing all the tasks that are mentioned in the project description. To find the cluster formed by the multiple nodes in the graph the K-Core Algorithm is used.

3.1 Task -1:

To Implement the k- core algorithm function the given input is adjacency matrix and the k value and it produces the nodes in the matrix which are connected to the graph for the given k value.

- Initially, to calculate the degree of each node in the matrix and find the nodes with degree less than the k-value and store them in the variable for nodes to be removed.

```
matlen = len(mtrx)
nodeDegree = npy.sum(mtrx, axis=1)
rem_nodes = [i for i in range(matlen) if nodeDegree[i] < k]
```

- Now, add all the connected nodes except nodes in rem_nodes to the subgraph.
- This process is repeated until there are no nodes left with degree less than k.
- Finally all the connected nodes for the k-core are provided as output of the k_Algorithm function.

```
subDeg = [sum(subgraph[i]) for i in range(len(subgraph))]
if not any([d < k for d in subDeg]) or not subgraph:
    return [l for l in range(matlen) if l not in rem_nodes]
else:
    return k_Algorithm(subgraph, k)
```

3.2 Task -2 :

To generate the adjacency matrix using the utd student id. Here the function takes student id as input to the function and convert the 10 digit id into a matrix of size 27 x 27.

- Initially, the input id is converted into binary format and multiply by 73 times.

```
bitSeq = ''
for i in uid:
        if int(i) % 2 == 1:
                        bitSeq += '1'
        else:
                        bitSeq += '0'
bitSeq *= 73
```

- Now, the matrix is populated with the 730 bits of bit sequence that is stored in bitSeq. All the isolation nodes are removed and the values of first element and last element of the node are set to 1.

- For all the diagonal elements that is matrix[i][i] in each row the value is set to 0 and made the matrix symmetric using below code.

```
for i in range(totalRw):
    if npy.sum(adjMatrx[i]) == 0:
        adjMtrx[i][0] = adjMatrx[i][-1] = adjMtrx[0][i] = adjMatrx[-1][i] = 1
    adjMatrx[i][i] = 0
    for j in range(i):
        adjMatrx[i][j] = adjMatrx[j][i]
```

- After removing the isolation nodes and making the matrix symmetric the function returns the adjacency matrix.

3.3 Task-3:

This task computes the connected nodes for all the k values from 1 to 27. So, the k_Algorithm function is executed 27 times and all the k-nodes are retrieved for each k value.

```
nodes = {}
for k in range(1, 28):
        nodes[k] = k_Algorithm(adjMtrx, k)
        if nodes[k]:
                print(f'k={k}: {nodes[k]}')
```

3.4 Task-4:

In this task networkx is used to draw the graph of the cluster with all the connected nodes that are obtained for k = 1 to 27. All the nodes are stored in nodes variable from the output of k_Algorithm function.

```
for k in range(1, 28):
        nodes = k_Algorithm(adjMtrx, k)
        if len(nodes) > 0:
                clstr_edges += get_edges(nodes, adjMtrx)
```

- The nodes and adjacency matrix are given as input to get_edges function and retrieve all the edges that are connected to the corresponding nodes of k cores.
- Using all the edges and nodes that are obtained the graph is plotted to represent the cluster.

3.5 Task-5:

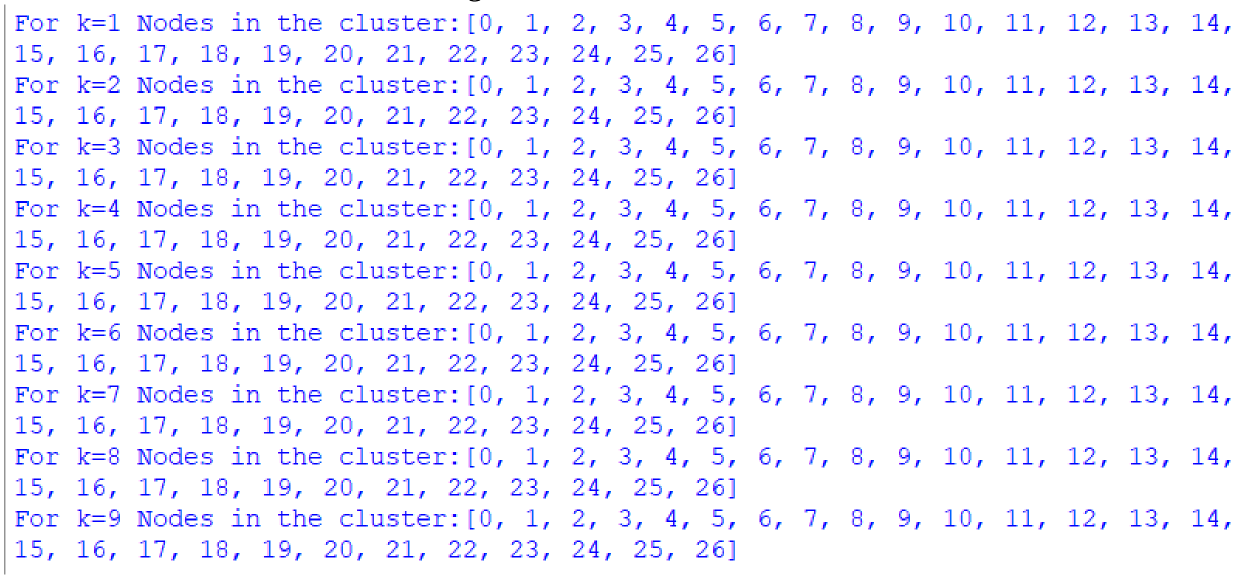
In this task random edges are deleted by replacing the value of element in the matrix for the connected nodes to 0.

```
for i,j in delEdges:
    amt[i][j] = 0
    amt[j][i] = 0
```

- Now, call the function get_nodes to get the value of the k-core after removing the desired edge. And call the genGraph function to generate the graph for the resultant nodes.
- After generating the graph again replace the value of element in the matrix with 1 to reconnect the nodes with the edge.

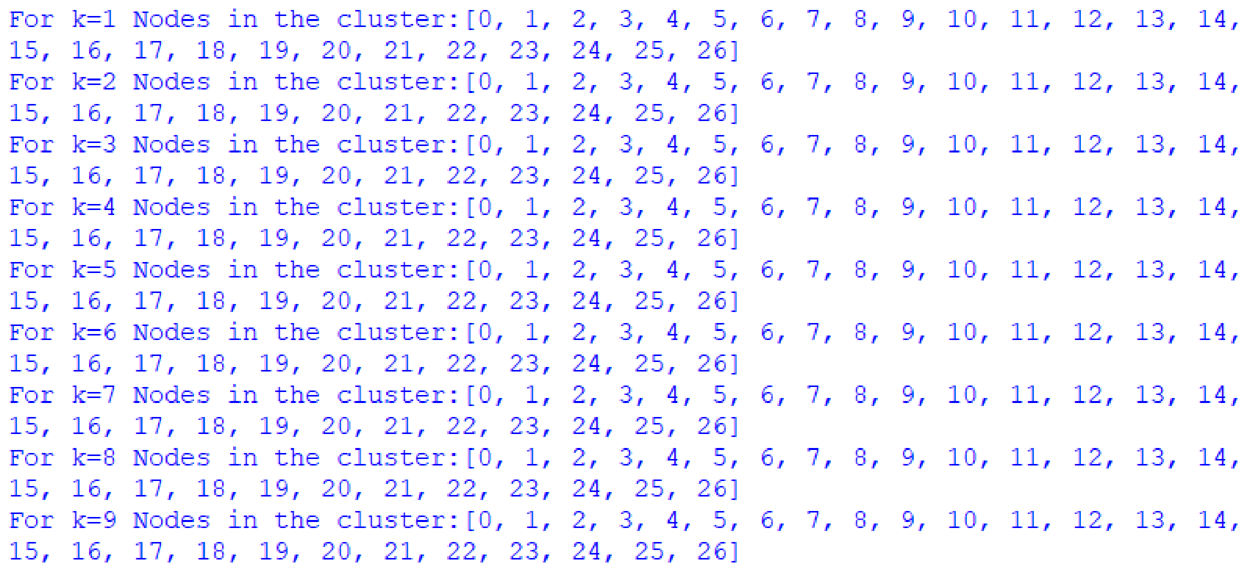
OUTPUT

- For the student id “2021594665”, the below output shows the obtained k-core which contains all nodes with at least k neighbors.



```
For k=1 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=2 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=3 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=4 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=5 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=6 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=7 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=8 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=9 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

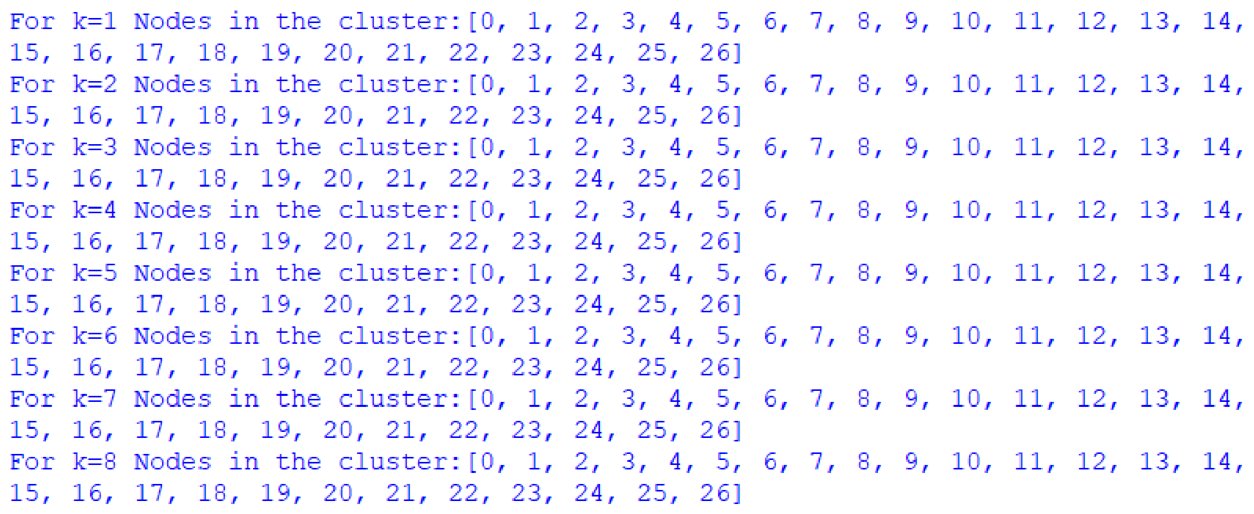
- In the above output it is observed that for the id “2021594665” the value of k-core = 9. This indicates that the all the 27 nodes have at least 9 connected neighbors.
- Now after removing an edge, for example edge: (2, 15). The below output is observed.



```
For k=1 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=2 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=3 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=4 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=5 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=6 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=7 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=8 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=9 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

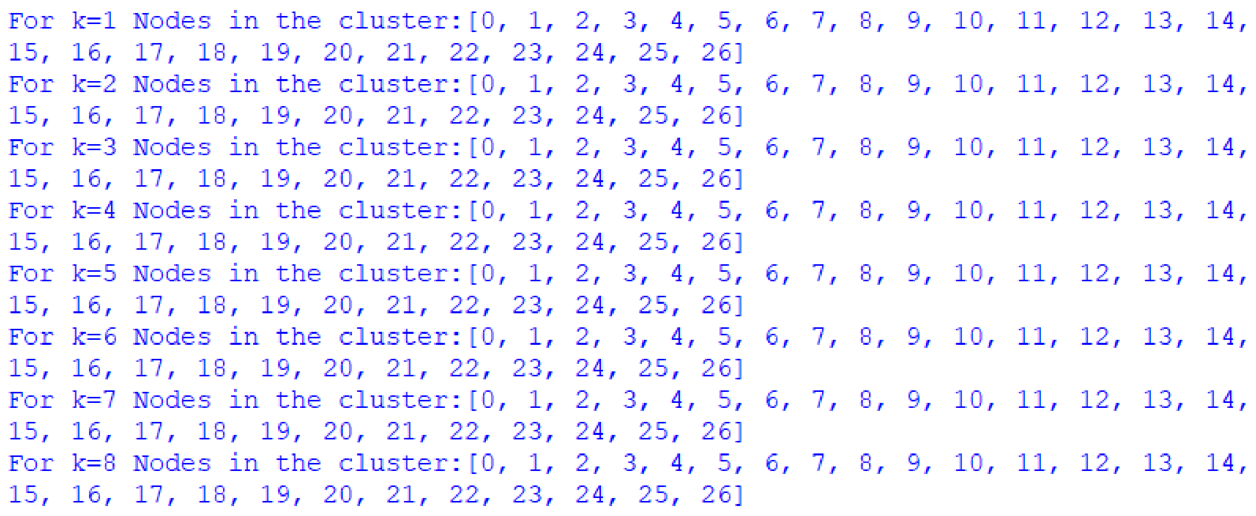
- Here, it is observed that the value of k-core = 9 and does not change because for the nodes 2, 15 there are more than 9 neighbors. So, after removing the edge (2, 15) there is no change in the value of the k-core.

- Now, replace the edge (2, 15) and if another edge (8, 19) is removed then the below result can be observed:



```
For k=1 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=2 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=3 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=4 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=5 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=6 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=7 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=8 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

- For the node 8 there are only 9 neighbors, so after deleting the edge (8, 19) the node 8 has only 8 neighbors. So the value of k-core becomes 8. Now, the cluster is 8-core.
- Now if another edge (18, 23) is deleted then the obtained result is shown below:



```
For k=1 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=2 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=3 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=4 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=5 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=6 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=7 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=8 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

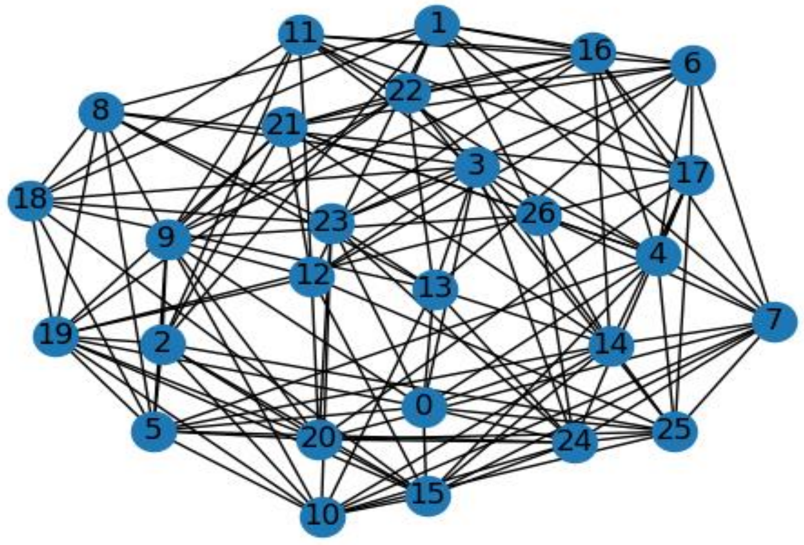
- For the node 18 there are only 8 neighbors after deleting the edge (18, 23). So, the cluster becomes 8-core.
- Now, if the edge (24, 26) is deleted the output of the k-core algorithm is

```
For k=1 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=2 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=3 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=4 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=5 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=6 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=7 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=8 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
For k=9 Nodes in the cluster:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

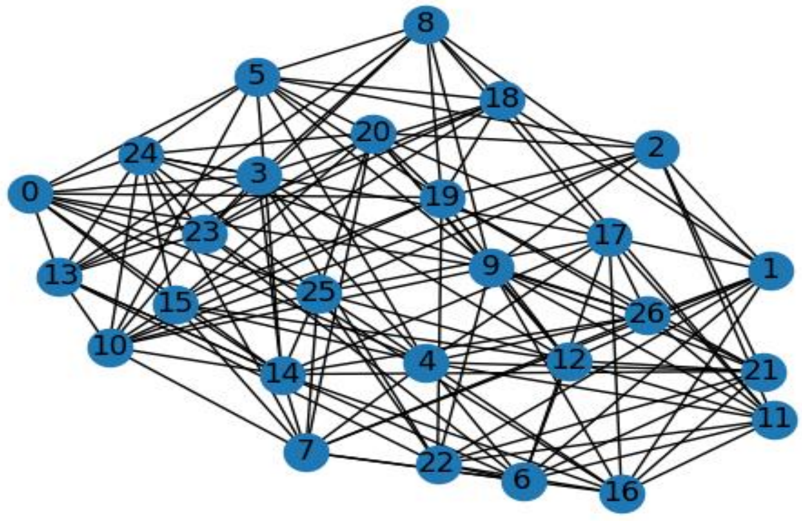
- Here, the cluster is 9-core because both nodes 24, 26 have more than or equal to 9 neighbors even after removing the edge (24, 26).
- Based on the observed outputs for the input student id = “2021594665” the value of k-core is 9.
- So, this observation shows that upon deleting an edge the algorithm is again executed and the k- core value is updated based on the minimum number of neighbors that are available for all the 27 nodes.

GRAPHICAL REPRESENTATION

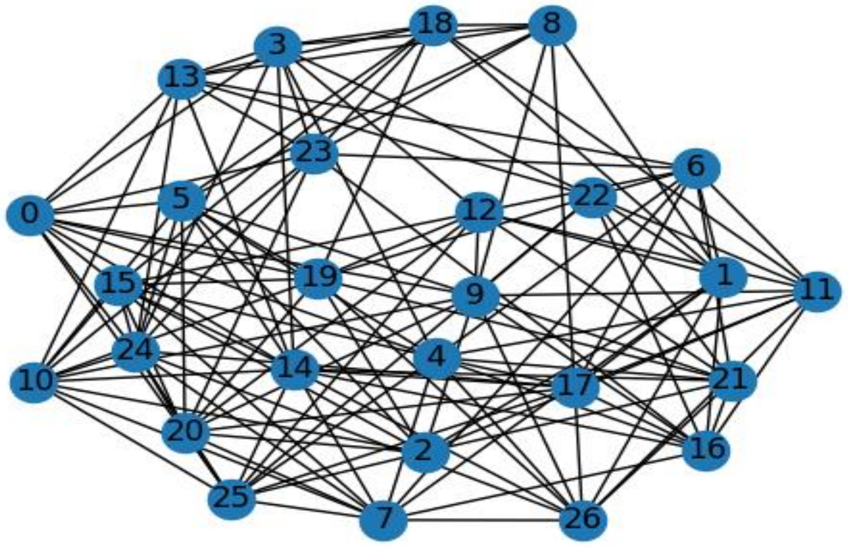
- The cluster that is obtained for the student id : “2021594665” is shown below:



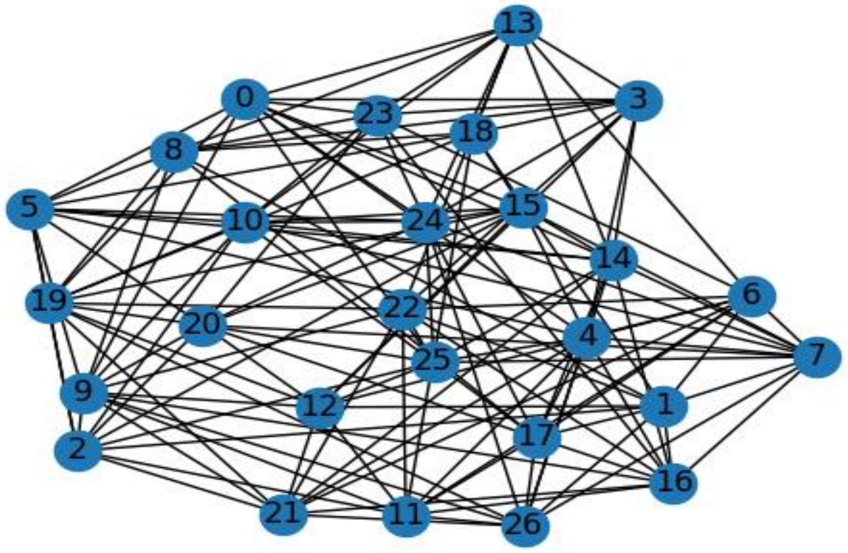
- After removing the edge (2, 15) the obtained graphical representation of the cluster is



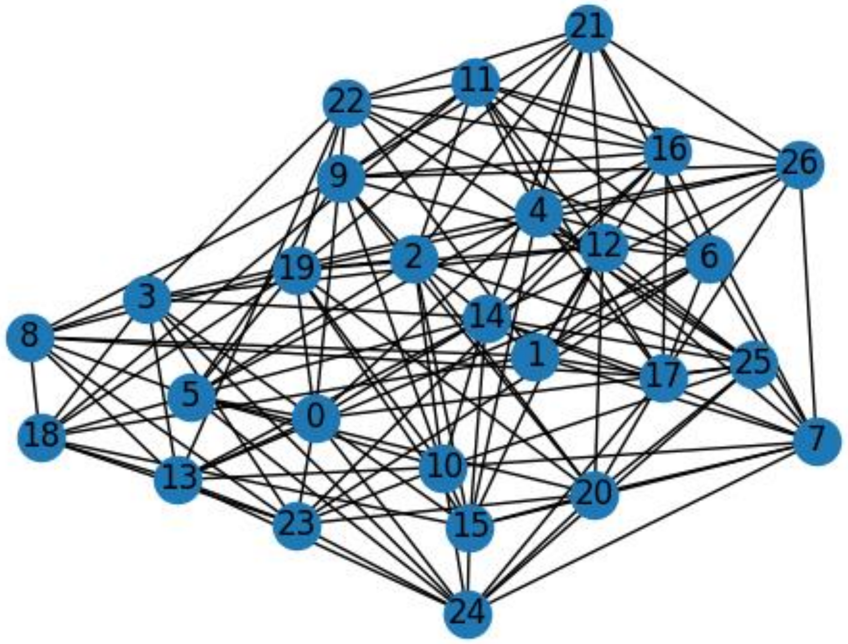
- Now removing the edge (8, 19) results in the graph as shown below:



- Removing the edge (18, 23) generates the graph below:



- After replacing the edge (18, 23) and removing the new edge (24, 26) the graph that is generated is shown below



APPENDIX

Source code for the project:

```
import numpy as npy

import networkx as ntwx

import matplotlib.pyplot as matplt


# Task -1: Function to implement the k-core algorithm to find the k neighbors
def k_Algorithm(mtrx, k):


    matlen = len(mtrx)

    nodeDegree = npy.sum(mtrx, axis=1)


    rem_nodes = [i for i in range(matlen) if nodeDegree[i] < k]


    subgraph = []

    for i in range(matlen):

        if i not in rem_nodes:

            row = []

            for j in range(matlen):

                if j not in rem_nodes:

                    row.append(mtrx[i][j])

            subgraph.append(row)


    subDeg = [sum(subgraph[i]) for i in range(len(subgraph))]

    if not any([d < k for d in subDeg]) or not subgraph:

        return [l for l in range(matlen) if l not in rem_nodes]

    else:

        return k_Algorithm(subgraph, k)
```

```
# Task - 2: Generate the adjacency matrix from the given UTD Student ID.
def genAdjacencyMatrix(uid):

    bitSeq = ''
    for i in uid:
        if int(i) % 2 == 1:
            bitSeq += '1'
        else:
            bitSeq += '0'
    bitSeq *= 73

    totalRw = 27
    totalCl = 27
    adjMatrx = [[0 for j in range(totalCl)] for i in range(totalRw)]

    for row in range(totalRw):
        fIndex = row*totalCl
        lIndex = fIndex + totalCl
        bin_Numbers = bitSeq[fIndex:lIndex]
        dcml_Numbers = [int(b, 2) for b in bin_Numbers]
        adjMatrx[row] = dcml_Numbers

    for i in range(totalRw):
        if npy.sum(adjMatrx[i]) == 0:
            adjMtrx[i][0] = adjMatrx[i][-1] = adjMtrx[0][i] = adjMatrx[-1][i] = 1
        adjMatrx[i][i] = 0
        for j in range(i):
            adjMatrx[i][j] = adjMatrx[j][i]

    return adjMatrx

# Task - 3: Perform k-core algorithm for k-values from 1 to 27 on the adjacency matrix generated
from student id.

def get_nodes(adjMtrx):
    nodes = {}
    for k in range(1, 28):
        nodes[k] = k_Algorithm(adjMtrx, k)
        if nodes[k]:
            print(f'For k={k} Nodes in the cluster:{nodes[k]}')



# Task - 4: Generate the graph for the edges obtained for k-nodes from all k-values from 1 to 27

def get_edges(clstr_nodes, adjMtrx):
    edges = []
    for i in clstr_nodes:
        for j in clstr_nodes:
            if i < j and adjMtrx[i][j] == 1:
                edges.append((i, j))
    return edges



def genGraph(adjMtrx):
    clstr_edges = []
    for k in range(1, 28):
        nodes = k_Algorithm(adjMtrx, k)
        if len(nodes) > 0:
```

```
        clstr_edges += get_edges(nodes, adjMtrx)

    Clstr = ntwx.Graph()
    Clstr.add_edges_from(clstr_edges)

    p = ntwx.spring_layout(Clstr)
    ntwx.draw_networkx_edges(Clstr, p)
    ntwx.draw_networkx_labels(Clstr, p)
    ntwx.draw_networkx_nodes(Clstr, p)

    matplt.axis('off')
    matplt.show()

# Task - 5: Remove random edges and perform all the the previous tasks.
def rem_EdgeFunc(amt, delEdges):

    for i,j in delEdges:
        amt[i][j] = 0
        amt[j][i] = 0

        get_nodes(amt)
        genGraph(amt)

        amt[i][j] = 1
        amt[j][i] = 1
```

```
if __name__ == '__main__':

    ID = "2021594665"
    Matrix = genAdjacencyMatrix(ID)

    get_nodes(Matrix)
    genGraph(Matrix)

    delEdges = [(2, 15), (8, 19), (18, 23), (24, 26)]
    rem_EdgeFunc(Matrix, delEdges)
```

REFERENCES

1. https://en.wikipedia.org/wiki/Degeneracy_(graph_theory)
2. https://networkx.org/documentation/stable/_downloads/networkx_reference.pdf
3. Lecture Notes from E-Learning
4. https://sites.engineering.ucsb.edu/~shell/che210d/numpy.pdf