Algorithmic Concepts of Telecommunications

CS 6385 - PROJECT 1

Submitted By:

Anunitya Alla: Axa210108

Algorithm:

The core concept of the algorithm is to find the k-core neighbours of a graph. The algorithm follows these steps:

1. Loop through the given adjacency matrix to find if any node has a degree less than K
2. If there is such a node, then remove the node from the matrix by modifying the row and column values to be 0
3. Break from the loop when there are no updates made to the matrix, i.e. no changes in the node degree.
4. The nodes in the matrix are in the K-core

We can identify the nodes that belong to the k-core and ensure that their neighbours also satisfy the k-core criteria.

Codebase description:

The BuildAG class is designed to facilitate various operations on a graph. It has the following features:

Constructor: The class constructor takes an adjacency matrix as input and builds a graph. It initialises the graph data structure and populates it based on the adjacency matrix.

Functionality Methods:

FindKC0re(): This method takes the input of an adjacency matrix and an integer k. Returns the updated matrix with the nodes which do not belong in K-core removed.

Helper Functions:

printAG(): This function prints the graph, displaying the nodes and their corresponding edges.

printKCN(): This graph prints a clear k-core of the original graph, highlighting all the nodes present in the given k-core.
printB(): This function prints a banner for the tasks, providing a clear indication of the current task being performed.

This class provides comprehensive functions to build a graph from an adjacency matrix and manipulate the graph to perform calculations on it. Overall, the BuildAG class encapsulates the functionality necessary to work with graphs represented by an adjacency matrix, enabling tasks such as k-core calculation and node traversal.

Task1:

For this part of the project, the goal is to determine the k-core neighbours of a given graph. The graph is represented by an adjacency matrix, which provides information about the connections between nodes. The task is to identify if the k-core neighbors exist in the graph, based on the specified value of k.

Results:

Input:

```
K = 3
adjacency_matrix = [
    [0, 1, 1, 1, 0],
    [1, 0, 1, 1, 1],
    [1, 1, 0, 1, 1],
    [1, 1, 1, 0, 0],
    [0, 1, 1, 0, 0]]
```

For the above input k and the adjacency matrix, we get the following plots of the graph and their corresponding k-cores. The graph has a maximum of 3 cores.

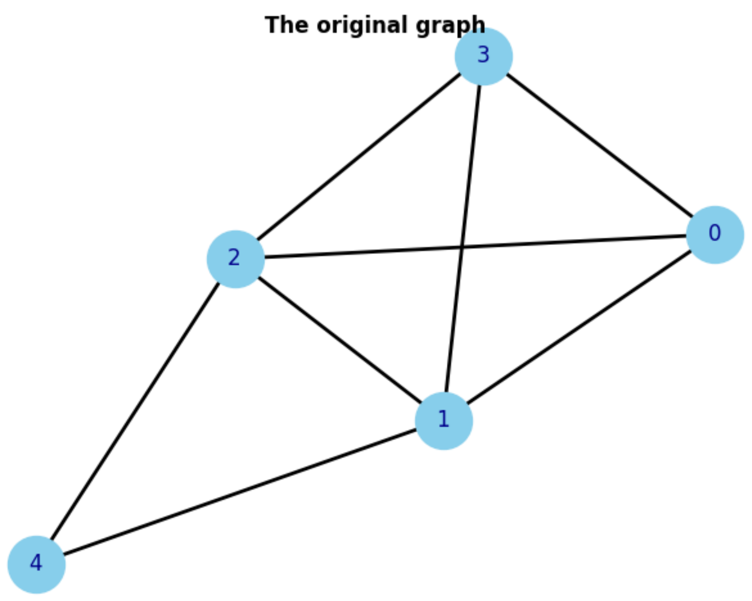


Fig1: The graph plotted from the input adjacency matrix

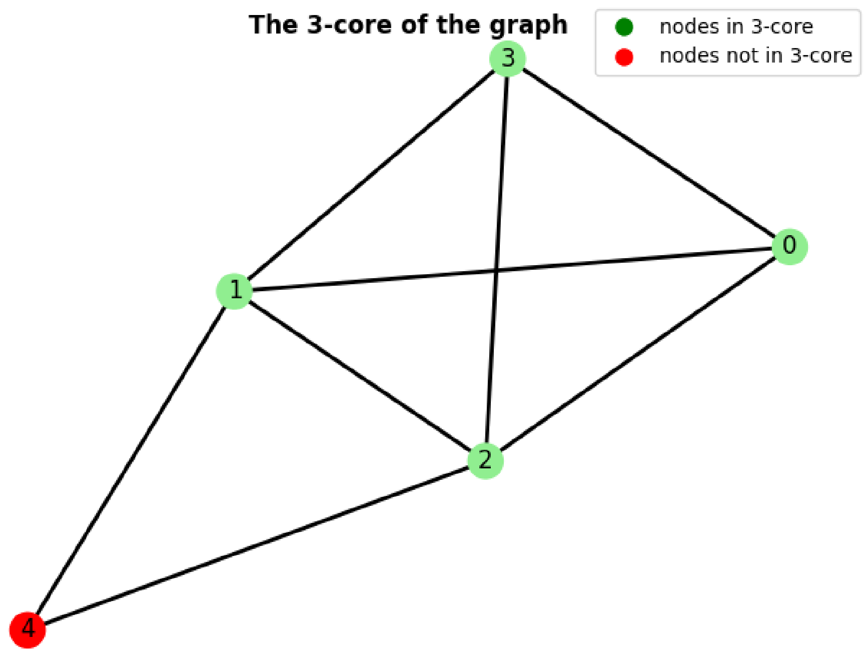


Fig2: The 3-core graph of the adjacency matrix

For the input value k=2, we get the respective 2-core graph. All the nodes in the graph are at least 2-core in the following Fig.3

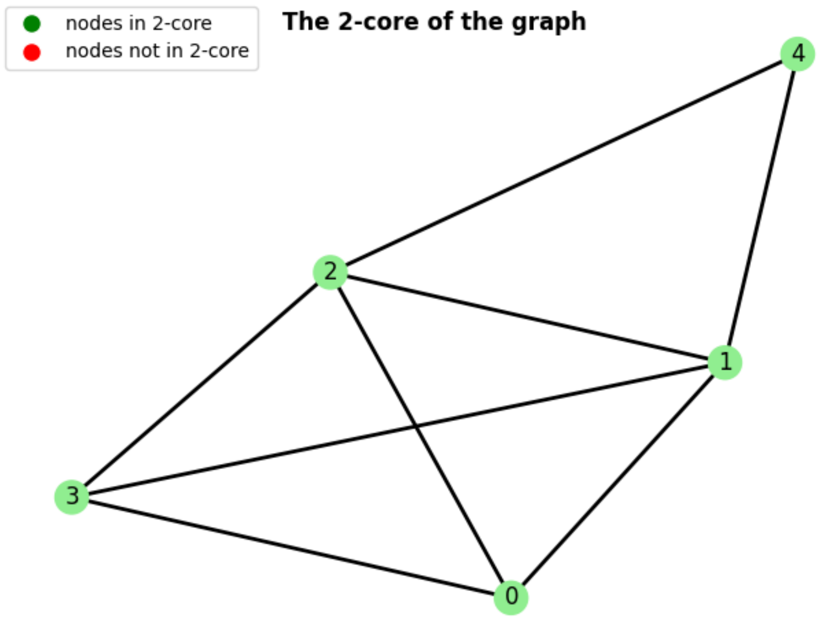


Fig3: the 2-core graph of the adjacency matrix

We also output the nodes in the k-core to the terminal. Example output of a 3-core is as follows,

```
PS C:\Users\ana32\Desktop\ACN\P1> python task1.py
------------------------------------------------------------
       Task 1: FINDING KCORES OF AN ADJACENCY MATRIX
------------------------------------------------------------
3-core nodes are [0, 1, 2, 3]
```

Example output of a 2-core is as follows,

```
PS C:\Users\ana32\Desktop\ACN\P1> python task1.py
------------------------------------------------------------
       Task 1: FINDING KCORES OF AN ADJACENCY MATRIX
------------------------------------------------------------
2-core nodes are [0, 1, 2, 3, 4]
```

Task2:

For task 2 of the project, we performed the following steps:

1. Extracted the bit sequence from the UTD-ID
2. Extended the bit sequence and populated an adjacency matrix.
3. Identify and remove isolated nodes
4. Remove self-loops
5. Make the matrix symmetric.

The final obtained matrix is 26x26 long.

Output:

```
PS C:\Users\ana32\Desktop\ACN\P1> python task2.py
The UTD-ID is 2021615994
The bit sequence for the above ID is 0001011110
Checking for isolated nodes ...
Checking for self loops ...
The matrix is now symmetric
Printing the adjacency matrix generated from the given UTD-ID ...
Number of rows: 26
Number of columns: 26
 0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1
 0  0  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0
 0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1
 1  0  1  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1
 0  0  1  0  0  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1  1  0
 1  0  1  1  0  0  1  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1
 1  0  1  0  0  1  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0
 1  1  0  1  0  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1
 1  0  0  1  0  1  0  0  0  1  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1
 0  1  0  1  1  0  1  0  1  0  0  1  1  1  1  0  0  0  0  1  0  1  1  1  1  0
 0  1  0  1  0  0  1  0  1  0  0  0  0  1  0  1  1  1  1  0  0  0  0  1  0  1
 0  1  1  0  1  0  1  1  0  1  0  0  1  0  0  0  0  1  0  1  1  1  1  0  0  0
 0  1  0  0  1  0  1  0  0  1  0  1  0  1  1  1  1  0  0  0  0  1  0  1  1  1
 1  0  1  0  1  1  0  1  0  1  1  0  1  0  0  1  0  1  1  1  1  0  0  0  0  1
 0  0  1  0  1  0  0  1  0  1  0  0  1  0  0  0  0  0  0  1  0  1  1  1  1  0
 1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  0  1  1  1  0  0  0  0  1  0  1
 1  0  1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  1  1  1  1  0  0  0
 1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  0  0  0  1  0  1  1  1
 1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  0  0  1  1  0  0  0  0  1
 0  1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  1  0  0  1  1  1  1  0
 0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  0  0  0  1  0  1
 0  1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  0  1  0  0  0
 0  1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  1  1  1
 1  0  1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  0  1
 0  0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  1  0  0  1  0  0  0
 1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  1  0  1  1  0  0
PS C:\Users\ana32\Desktop\ACN\P1>
```

Task3:

In this part of the project, we utilize the functions provided by the BuildAG class to construct a graph from the matrix generated in task 2. Subsequently, we repeatedly execute the findKC0re() function on the resulting matrix until we obtain all the K-core. The maximum k-core present in the matrix is 10.

Results:

We plot the following graphs of the generated matrix.

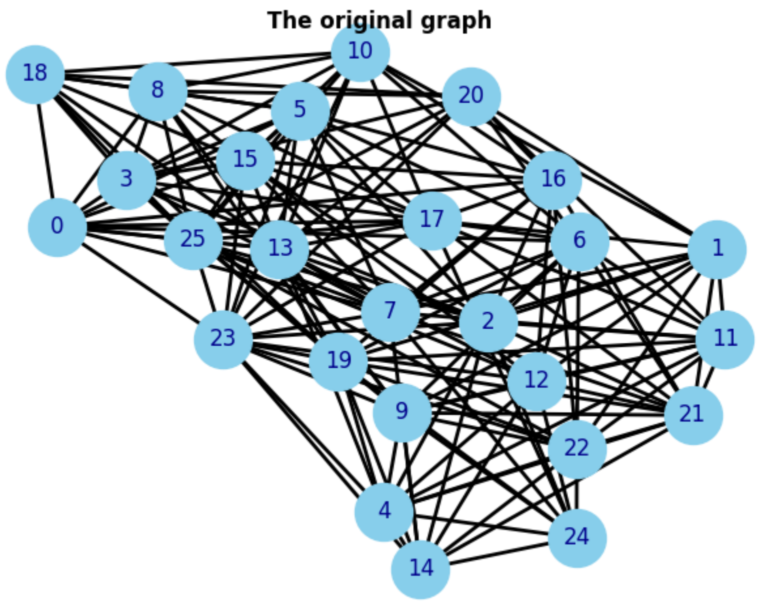


Fig4: The generated graph from task2

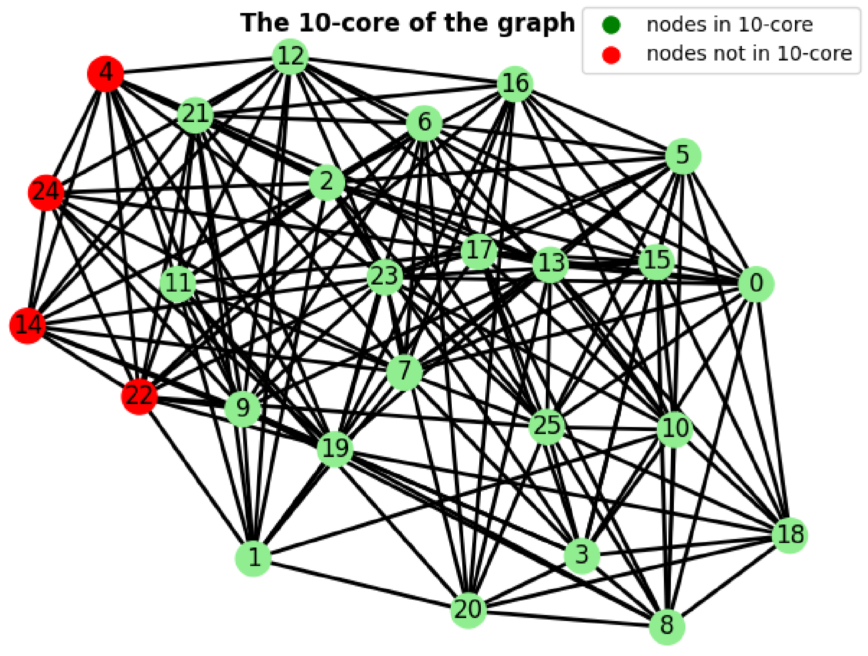


Fig5: The 10-core of the generated graph

Output:

```
1-core nodes are [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]
2-core nodes are [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]
3-core nodes are [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]
4-core nodes are [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]
5-core nodes are [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]
6-core nodes are [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]
7-core nodes are [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]
8-core nodes are [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]
9-core nodes are [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]
10-core nodes are [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18,
19, 20, 21, 23, 25]
11-core nodes are []
```

```
12-core nodes are []
13-core nodes are []
14-core nodes are []
15-core nodes are []
16-core nodes are []
17-core nodes are []
14-core nodes are []
15-core nodes are []
16-core nodes are []
17-core nodes are []
18-core nodes are []
19-core nodes are []
20-core nodes are []
21-core nodes are []
22-core nodes are []
23-core nodes are []
24-core nodes are []
```

Task4:

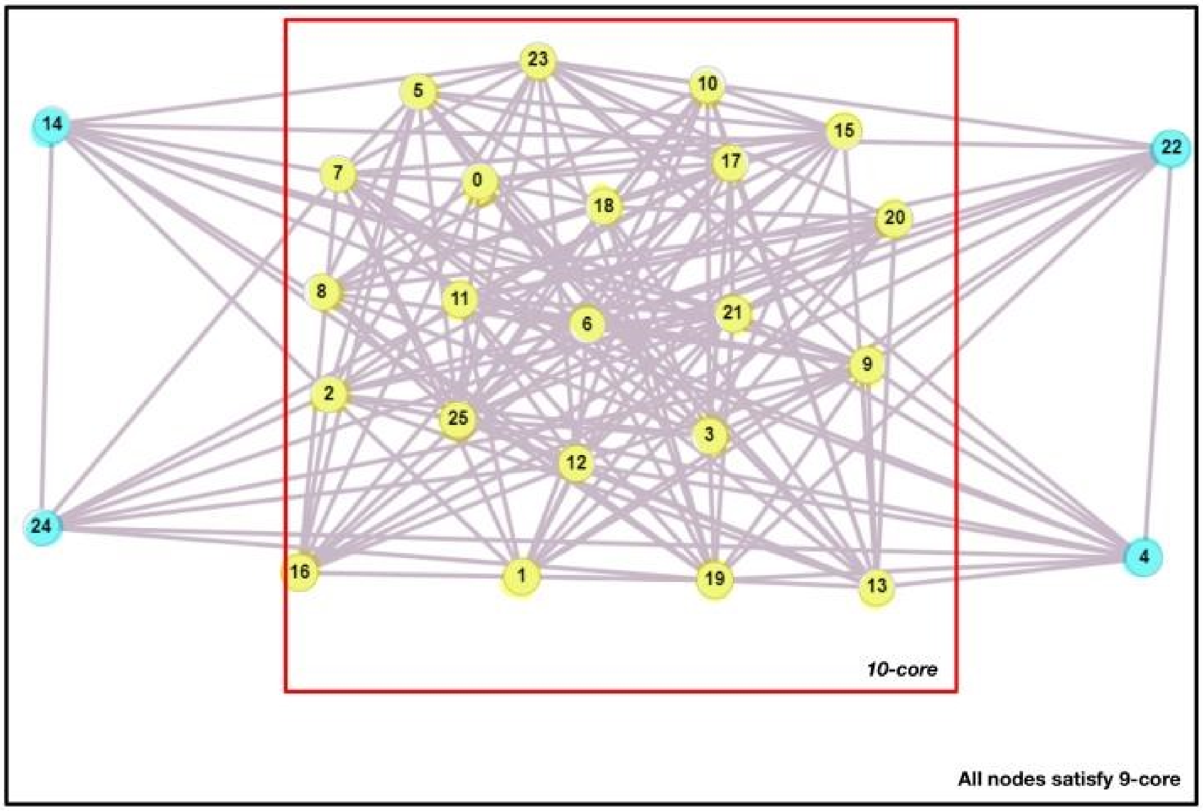


Fig6: The K-core of the generated graph

In the generated graph, all nodes satisfy the conditions up to the 9-core, meaning all nodes are part of the 9-core. Additionally, the maximum core observed in the graph is the 10-core.

Task 5:

CASE 1: Deleting Edge1 from Node1 → Node22

After removing the edge between node 22 and node 1, the maximum k-cores in the graph changed to 9. All the nodes participate in 9-core. 10-core is an empty list.

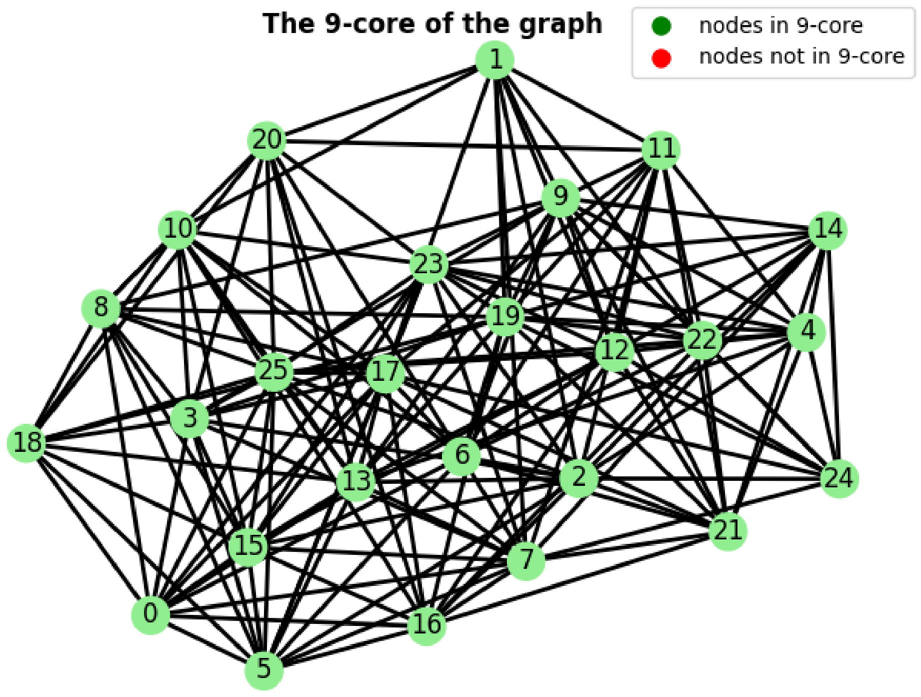


Fig7: The 9-core of the generated graph after deleting edge1

Output:
```
1-core nodes are [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]
2-core nodes are [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]
3-core nodes are [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]
4-core nodes are [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]
5-core nodes are [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]
6-core nodes are [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]
```

```
7-core nodes are [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]
8-core nodes are [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]
9-core nodes are [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]
10-core nodes are []
11-core nodes are []
12-core nodes are []
13-core nodes are []
14-core nodes are []
15-core nodes are []
16-core nodes are []
17-core nodes are []
18-core nodes are []
19-core nodes are []
20-core nodes are []
21-core nodes are []
22-core nodes are []
23-core nodes are []
24-core nodes are []
```

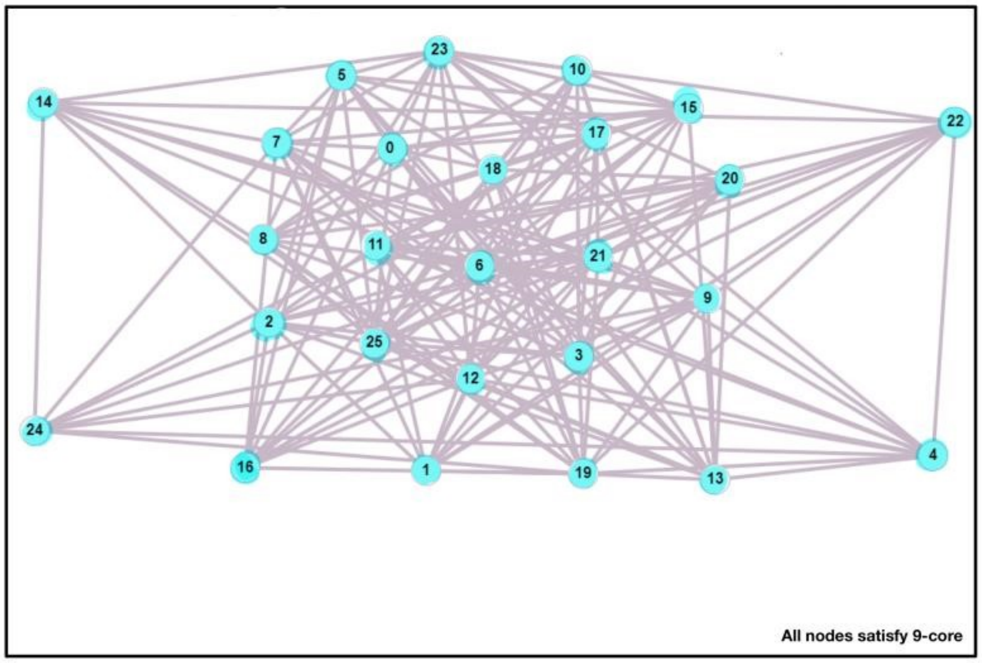


Fig8: The K-core of the generated graph after deleting edge1

CASE 2: Deleting Edge2 from Node 0 → node 16:

After removing the edge between node 0 and node 16, the maximum k-cores in the graph remain at 10. The K-core graph is identical to the original graph

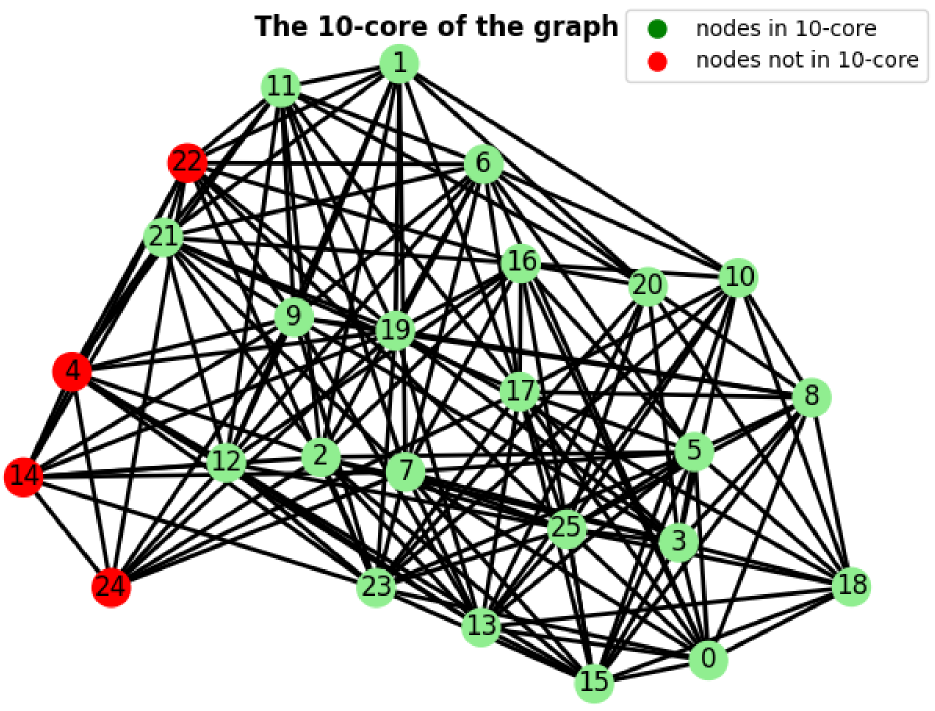


Fig9: 11-core of the modified graph after deleting Edge2

Output:

```
PS C:\Users\ana32\Desktop\ACN\P1> python task3.py
1-core nodes are [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]
2-core nodes are [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]
3-core nodes are [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]
4-core nodes are [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]
5-core nodes are [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]
6-core nodes are [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]
7-core nodes are [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]
```

```
8-core nodes are [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]
9-core nodes are [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]
10-core nodes are [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18,
19, 20, 21, 23, 25]
11-core nodes are []
12-core nodes are []
13-core nodes are []
14-core nodes are []
15-core nodes are []
16-core nodes are []
17-core nodes are []
18-core nodes are []
19-core nodes are []
20-core nodes are []
21-core nodes are []
22-core nodes are []
23-core nodes are []
24-core nodes are []
PS C:\Users\ana32\Desktop\ACN\P1>
```

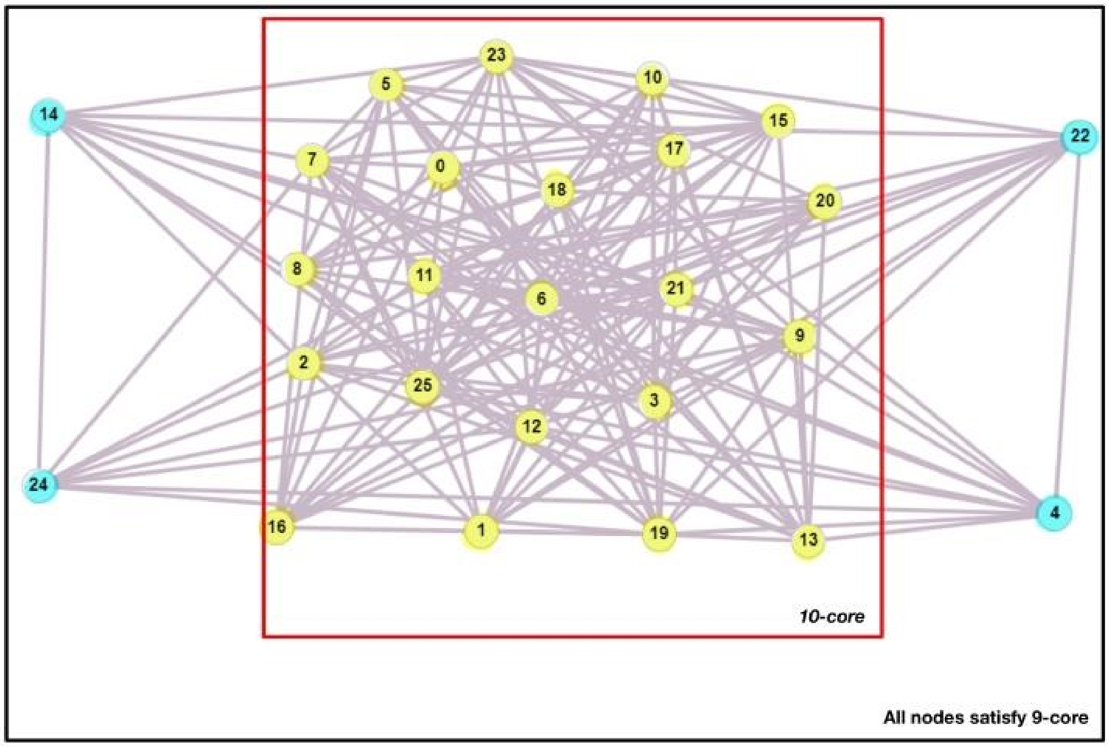


Fig10: The K-core of the generated graph after deleting edge2

CASE 3: Deleting Edge3 from Node3 → Node0:

The 10-core retains its structure and node composition even after removing the edge connecting Node3 to Node0. The original 10-core remains unchanged. The K-core graph is the same as the original graph.

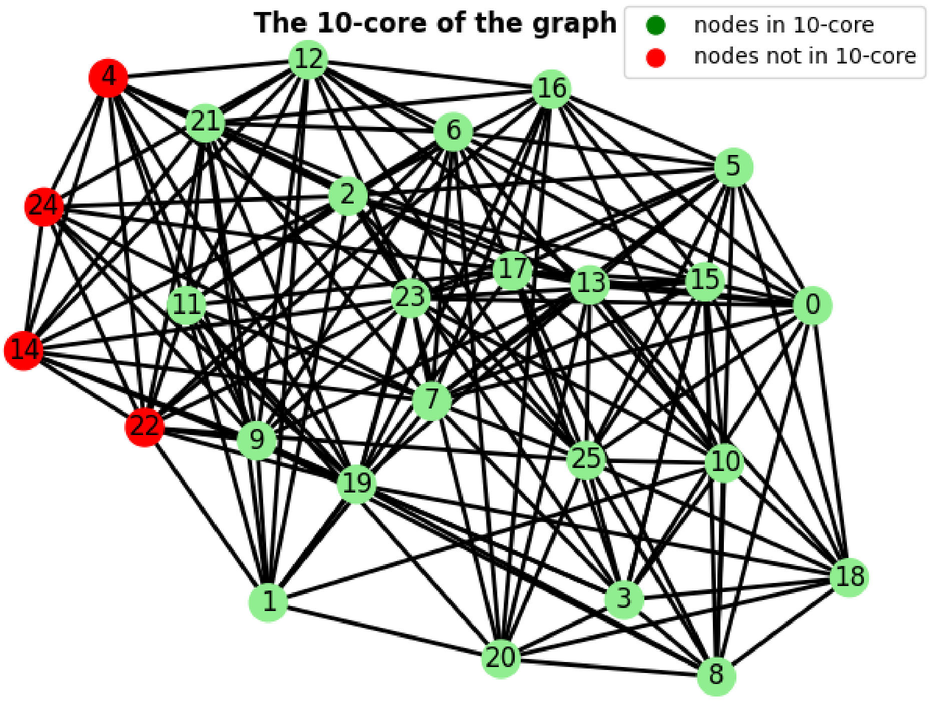


Fig11: 11-core of the modified graph after deleting Edge3

Output:

```
PS C:\Users\ana32\Desktop\ACN\P1> python task3.py
1-core nodes are [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]
2-core nodes are [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]
3-core nodes are [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]
4-core nodes are [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]
5-core nodes are [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]
6-core nodes are [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]
```

```
7-core nodes are [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]
8-core nodes are [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]
9-core nodes are [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]
10-core nodes are [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18,
19, 20, 21, 23, 25]
9-core nodes are [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]
10-core nodes are [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18,
19, 20, 21, 23, 25]
11-core nodes are []
12-core nodes are []
13-core nodes are []
14-core nodes are []
15-core nodes are []
16-core nodes are []
17-core nodes are []
18-core nodes are []
19-core nodes are []
20-core nodes are []
21-core nodes are []
22-core nodes are []
23-core nodes are []
24-core nodes are []
```

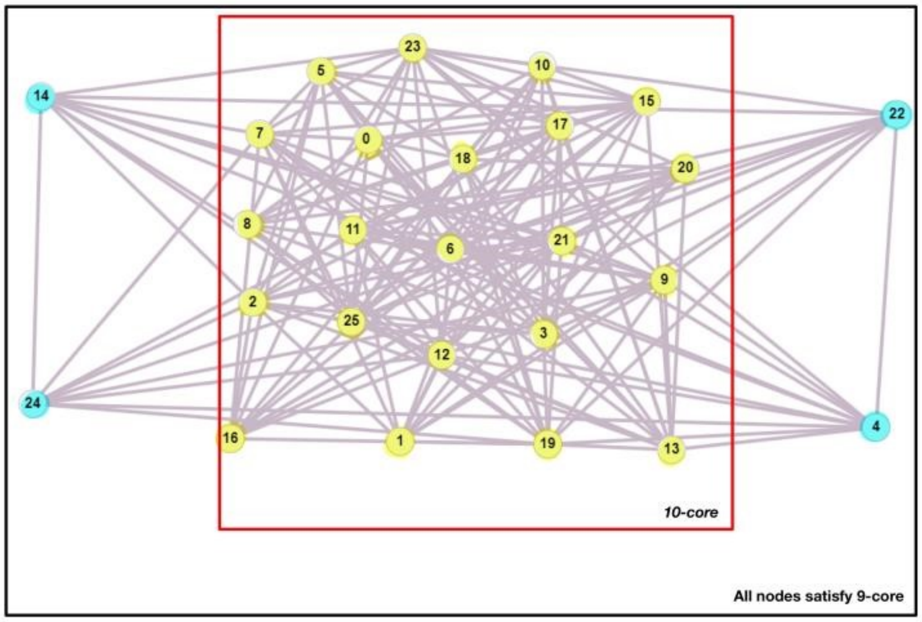


Fig12: The K-core of the generated graph after deleting edge3

Appendix:

Codebase - Submission.py

```python
import networkx as netX
import matplotlib.pyplot as Mplt

from collections import defaultdict


class BuildAG: # Build A Graph

        #Constructor takes a adjacency matrix and builds a graph
    def __init__(self,aM):

        self.adjM = aM
        n = len(self.adjM)
        self.g = defaultdict(list)


        for i in range(n):
            for j in range(n):
                if aM[i][j] == 1:
                    self.addAE(i,j)

        # Adds An Edge between two vertices
    def addAE(self, vertexU, vertexV):
        self.g[vertexU].append(vertexV)


    def printAG(self): # Print A Graph

        g = netX.Graph()

            #Build netX graph to draw th eplot
        for vertex, neighboursList in self.g.items():
            for neighbour in neighboursList:
                g.add_edge(vertex,neighbour)

        layout = netX.spring_layout(g)
        netX.draw(g, layout, with_labels=True, node_size=1000,
node_color="skyblue", font_size=12, font_color="darkblue",
edge_color="black", width=2)
        Mplt.suptitle(f"The original graph", fontweight='bold')

        Mplt.show(block = False)
        Mplt.pause(120)

    def FindKCOre(self, aM, k):
            numR = len(aM)
            numC = len(aM[0])
            removed = set()
            while True:
                rowS = [sum(i) for i in aM]
                emptyKC = False


                for i in range(numR):
                    if (rowS[i] < k and  i not in removed):
                        #print(f"removing {i}")
                        removed.add(i)
                        for j in range(numC):

                            aM[i][j] = 0
                            aM[j][i] = 0
                        emptyKC = True

                if not emptyKC:
                    break

            return aM


    def printKCN(self, adjM, k: int): # Print KCOre Nodes

        grph = self.g

        nodeColour = ['red' for i in range(len(grph.keys()))]
        greennodeset = set()
        #print(f"Printing the nodes in {k} core along with neighbours ")

        g = netX.Graph()

        for vertex, neighboursList in grph.items():
            for neighbour in neighboursList:
                g.add_edge(vertex,neighbour)

        for i in range(len(adjM)):
            if sum(adjM[i]) >= k:
                greennodeset.add(i)
                nodeColour[i] = "lightgreen"

        layout = netX.spring_layout(g)
        node_list = set(range(len(grph.keys())))

        netX.draw(g, layout, with_labels=True, node_color=nodeColour,
nodelist = node_list, font_size=12,edge_color="black", width=2)


        Mplt.suptitle(f"The {k}-core of the graph", fontweight='bold')
        legend_elements = [Mplt.Line2D([0], [0], marker='o', color='w',
label=f"nodes in {k}-core", markerfacecolor='green', markersize=10),
Mplt.Line2D([0], [0], marker='o', color='w', label=f"nodes not in
{k}-core", markerfacecolor='red', markersize=10),
        ]

        Mplt.legend(handles=legend_elements)
        Mplt.show()


    def printB(self, taskNum, taskDesc): # Print A Banner on th eoutput
terminal to indicate the task number
        banner_width = 60
        taskTitle = f"Task {taskNum}:"
        taskBanner = taskTitle + " " + taskDesc
        bannerLine = "-" * banner_width

        print(bannerLine)
        print(taskBanner.center(banner_width))
        print(bannerLine)
```

Task1.py

```
from submission import BuildAG

def main():

        '''
        ------------------------------------------------------------
            Task 1: FINDING KCORES OF AN ADJACENCY MATRIX"
        ------------------------------------------------------------

        '''

        k = 2
        adjacency_matrix = [
    [0, 1, 1, 1, 0],
    [1, 0, 1, 1, 1],
    [1, 1, 0, 1, 1],
    [1, 1, 1, 0, 0],
    [0, 1, 1, 0, 0]
]

        task1 = BuildAG(adjacency_matrix)    # Build a grph from the input
matrix
        task1.printB(1, "FINDING KCORES OF AN ADJACENCY MATRIX") # print
banner
        task1.printAG() # print the grph to terminal
        adjM = task1.FindKC0re(adjacency_matrix,k)
        KC0reNodes = set()


        for i in range(len(adjM)):
            if sum(adjM[i]) >= k:
                KC0reNodes.add(i)
            #print(f"{i} degree is {sum(adjM[i])}")
        print(f"{k}-core nodes are {[i for i in KC0reNodes]}")
        task1.printKCN(adjM, k)


if __name__ == "__main__":
    main()
```

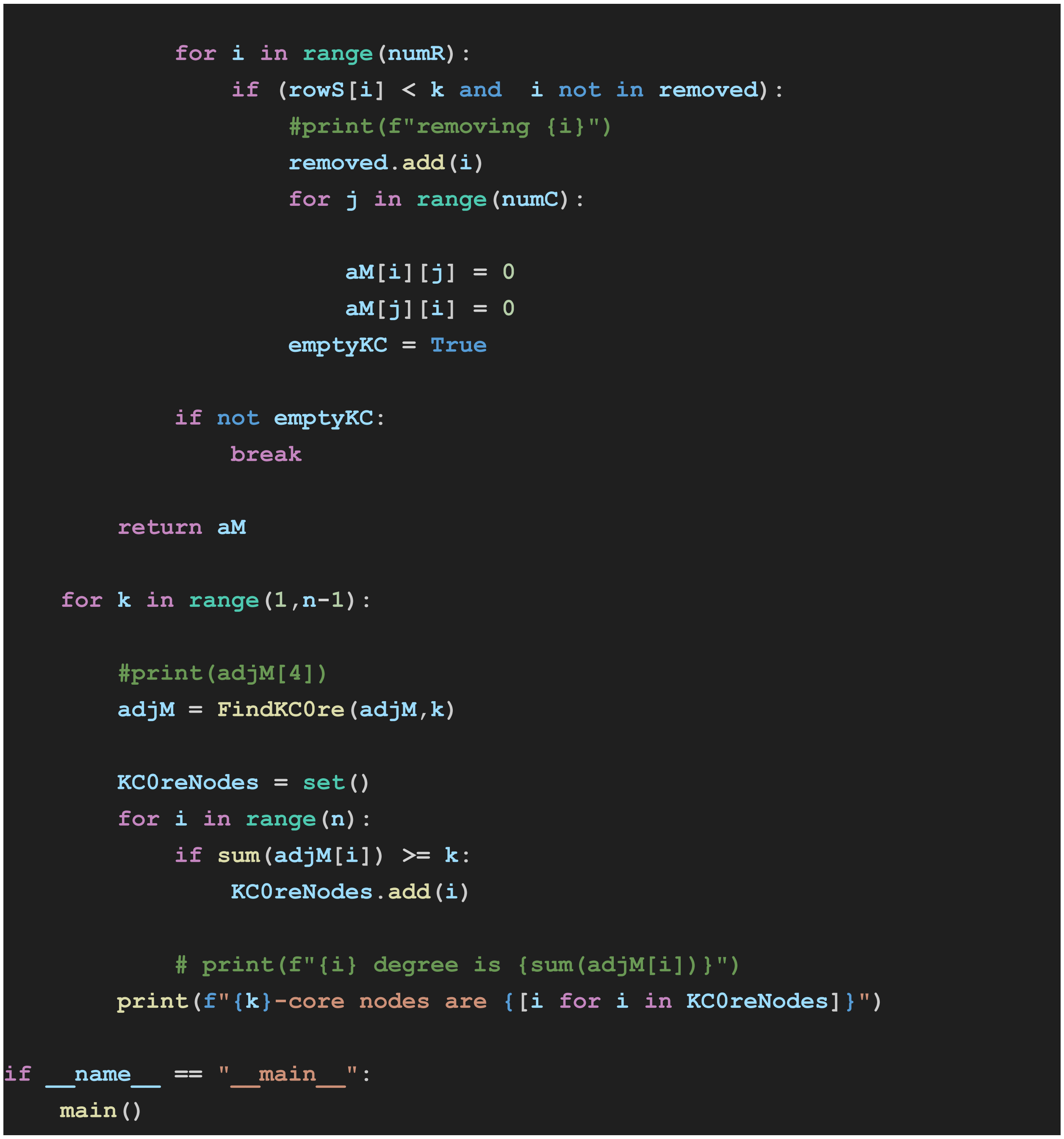
Task2.py

```
#from submission import BuildAG

def genereateAM(print_output=True):

    '''
        ------------------------------------------------------------
            Task 2: BUILDING AN ADJACENCY MATRIX FROM UTD-ID
        ------------------------------------------------------------


    '''

    ID = "2021615994"    # Input UTD-ID
    digits = [int(d) for d in ID]    # Convert the ID string to integer
array

    bitS = [0 if d%2 ==0 else 1 for d in digits]     # Extract the bit
sequence from the digit array
    bitSstr = ''.join(map(str,bitS))
    extBS = bitS * 68    # Extend the bit sequence 68 times

    print(f"The UTD-ID is {ID}")
    print(f"The bit sequence for the above ID is {bitSstr}")

    n = 26  # Total number of vertices

        # Initialize and populate the matrix
    adjM = [[0 for i in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(0,n):
            adjM[i][j] = extBS.pop(0)

    print("Checking for isolated nodes ... ")

         # Find the isolated nodes and make required changes to the matrix
    cardinality = [sum(adjM[i]) for i in range(n)]

    for node in cardinality:
        if cardinality[node] == 0 :

            # Change first position of row and coloumn
            adjM[node][0] = 1
            adjM[0][node] = 1

            #change in last position of row and column
            adjM[node][n-1] = 1
            adjM[n-1][node] = 1

    print("Checking for self loops ...")

        # Check diagonal values to prevent any self-loops
    for i in range(n):
        if adjM[i][i] == 1:
            adjM[i][i] = 0

        # Make matrix symmetric
    for i in range(n):
        for j in range(i+1,n):
            adjM[j][i] = adjM[i][j]

        # symmetric matrix check
    print("The matrix is now symmetric") if isSymmetric(adjM) else
print("The matrix is not symmetric")

    printM(adjM)    # Print the adjacency matrix

    return adjM

    # Function to print the adjacency matrix
```

```python
def printM(aM):

    print("Printing the adjacency matrix generated from the given UTD-ID
...")

    print(f"Number of rows: {len(aM)}")
    print(f"Number of columns: {len(aM[0])}")

    for row in aM:
        for i in row:
            print(f"{i:2}", end=" ")
        print()

    # function to check the if the matrix is symmetric
def isSymmetric(aM):
    rows = len(aM)
    columns = len(aM[0])

    if rows != columns:
        return False

    for i in range(rows):
        for j in range(columns):
            if aM[i][j] != aM[j][i]:
                return False
    return True

    # Function to print task banner
def printB( taskNum, taskDesc):
        banner_width = 60
        taskTitle = f"Task {taskNum}:"
        taskBanner = taskTitle + " " + taskDesc
        bannerLine = "-" * banner_width

        print(bannerLine)
        print(taskBanner.center(banner_width))
        print(bannerLine)

if __name__ == "__main__":
    genereateAM()
```

Task3.py

```
from submission import BuildAG
from task2  import genereateAM

import contextlib

def main():

    '''
        --------------------------------------------------------------
            Task 3: FINDING K-CORES OF THE GENERATED MATRIX"
        --------------------------------------------------------------

    '''
    with contextlib.redirect_stdout(None):  # Redirect print output

        adjM = genereateAM() #  Get matrix from task 2
     # print(adjM)
        n = len(adjM)
    print(adjM[21])

    # adjM[1][21] = 0
    # adjM[21][1] = 0

    print(adjM[21])


    def FindKCOre(aM, k):
        numR = len(aM)
        numC = len(aM[0])
        removed = set()
        while True:
            rowS = [sum(i) for i in aM]
            emptyKC = False
```



```python
            for i in range(numR):
                if (rowS[i] < k and  i not in removed):
                    #print(f"removing {i}")
                    removed.add(i)
                    for j in range(numC):

                        aM[i][j] = 0
                        aM[j][i] = 0
                    emptyKC = True

            if not emptyKC:
                break

        return aM

    for k in range(1,n-1):

        #print(adjM[4])
        adjM = FindKCOre(adjM,k)

        KCOreNodes = set()
        for i in range(n):
            if sum(adjM[i]) >= k:
                KCOreNodes.add(i)

            # print(f"{i} degree is {sum(adjM[i])}")
        print(f"{k}-core nodes are {[i for i in KCOreNodes]}")

if __name__ == "__main__":
    main()
```