

PROJECT NO – 1

K-CORE CLUSTERING

CS 6385

**ALGORITHMIC ASPECTS OF
TELECOMMUNICATION NETWORKS**

SUBMITTED BY:

SANJANA MAJETI

SXM220015

CONTENTS

Objective.....	03
Algorithm.....	04
Flowchart.....	05
Result Analysis.....	06
Appendix.....	10
References.....	15

OBJECTIVE

The objective is to identify and extract k-core clusters from an input graph represented by a 26x26 matrix. The input graph is constructed by generating a 10-digit sequence from an individual's student ID, where each digit is replaced with either 1 or 0 based on its parity (even or odd). This sequence is then repeated 68 times to form a 680-bit long sequence.

The concept of k-core clustering refers to the identification of dense subgraphs within a larger graph structure. In this case, the goal is to identify k-core clusters within the input graph. A k-core cluster is a subgraph where each node has at least k connections within the subgraph.

By applying k-core clustering to the input graph, we aim to extract cohesive groups of nodes that are densely interconnected, revealing the underlying structure and patterns within the graph. This process helps in understanding the relationships and communities present within the graph, which can have various applications in network analysis, social network analysis, and graph mining.

Algorithm

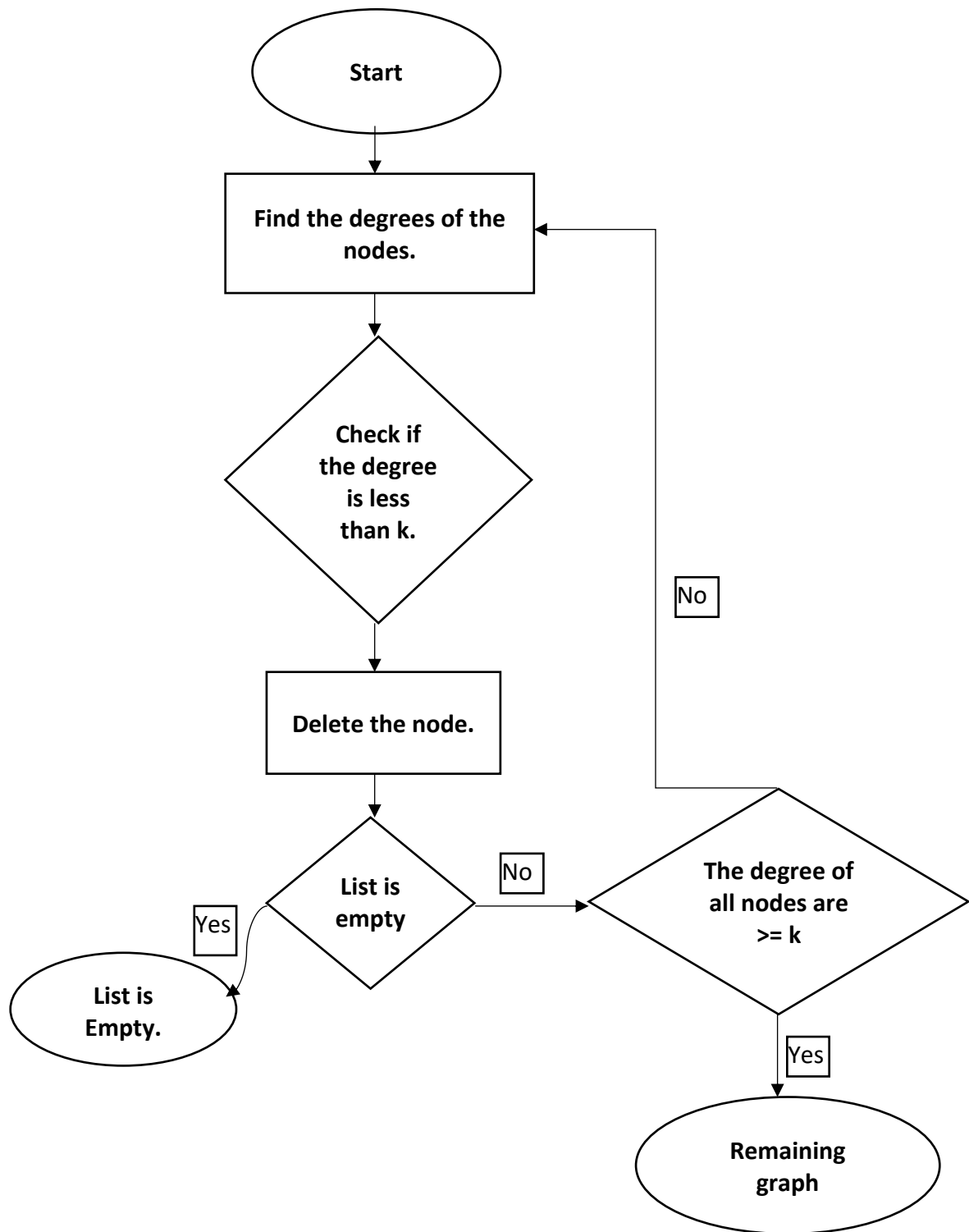
Step-1: The nodes with degrees lower than k are pruned or removed from the graph.

Step-2: Check if the remaining graph is Empty. If it is empty, then output the graph is empty.

Step-3: If the remaining graph is not empty then:

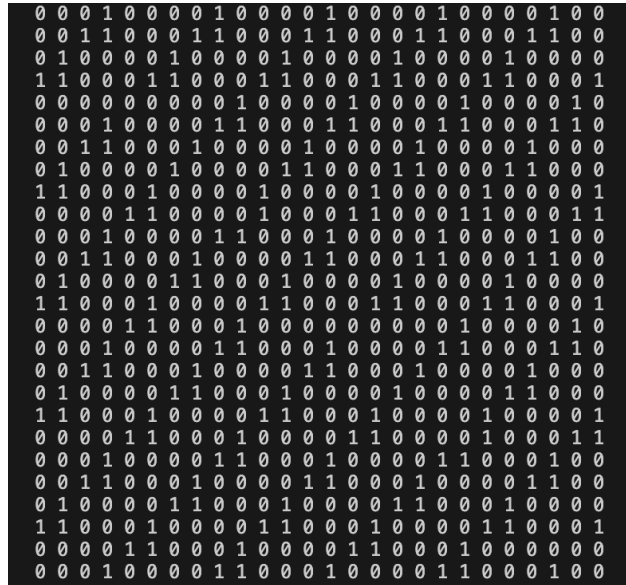
1. Check whether the remaining nodes have degrees that are greater than or equal to k . If all the nodes satisfy this condition, then display the resulting graph.
2. If the degrees of the remaining nodes are not all greater than or equal to k , repeat step 1.

Flowchart



Result Analysis

The unique student ID I'm registered to is 2021664655. The input 26x26 matrix generated such that there are no self-loops, no parallel edges and no isolated nodes is shown in Fig.1.



0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0
1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0
0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0
0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0
1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1
0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0
0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	1	0
0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0
0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0
0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0
0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0
0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0
0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0

Fig.1. The input matrix

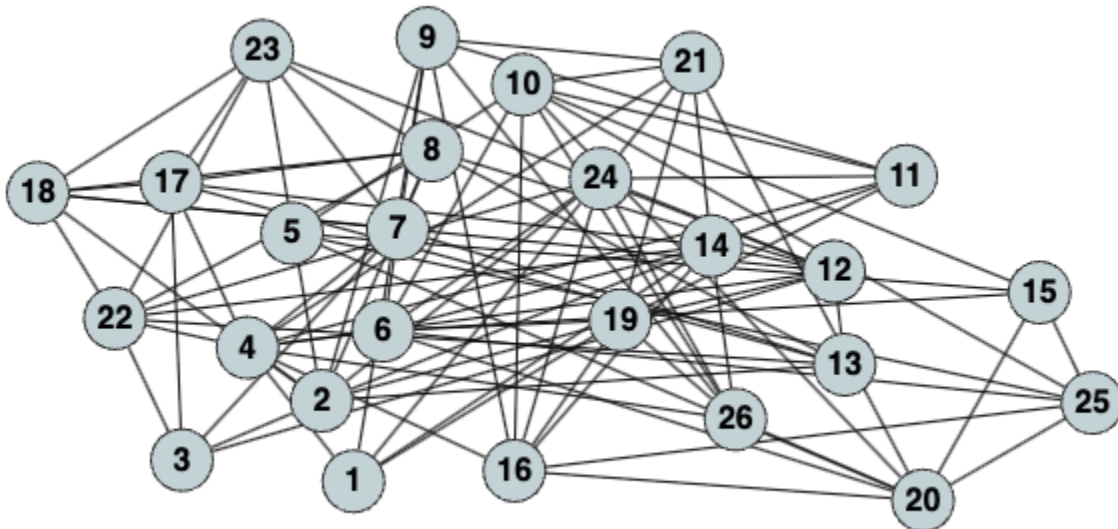


Fig.2. The network for the above input matrix

The nodes 5, 15, 25 were deleted in the 5th core and the rest of the nodes were deleted in the 6th core.

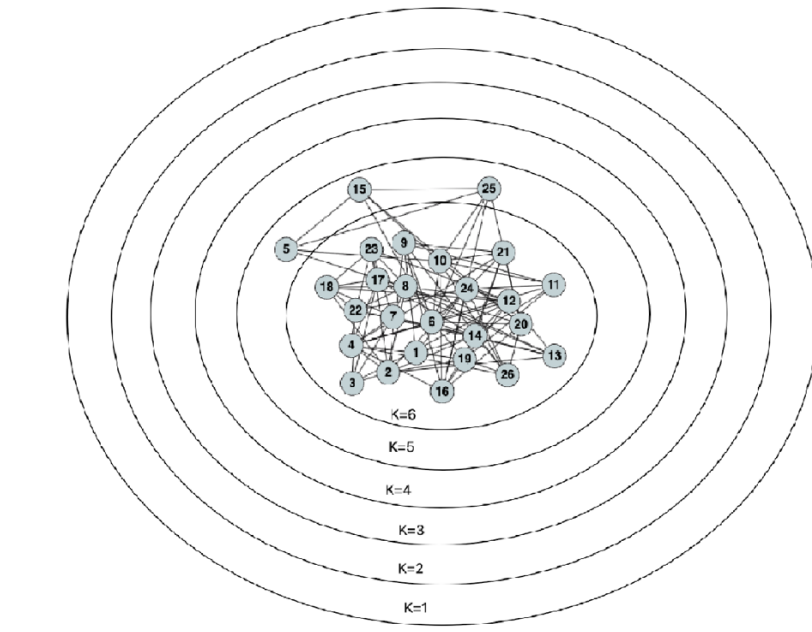


Fig.3. The K-core clusters in the graph for the input matrix

Analyzing the graph change, by deleting the edge between node 3 and node 7. By removing the edge between these nodes, the nodes 3,5,15,25 were deleted in the 5th core and the rest of the nodes were deleted in the 6th core.

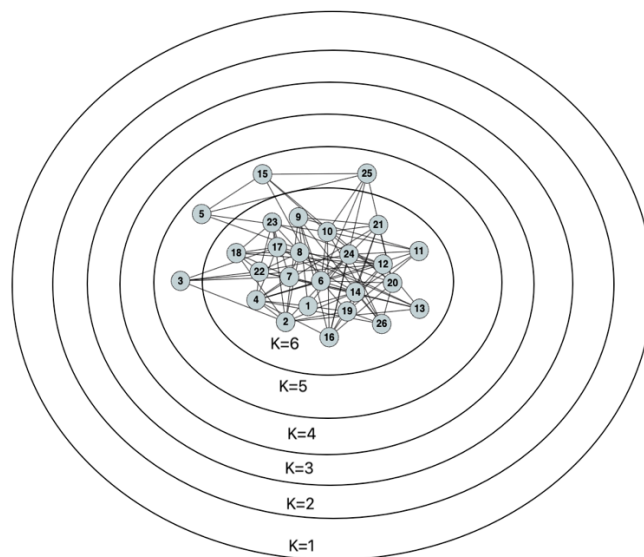


Fig.4. The K-core clusters in the graph after deleting edge between node 3 and 7.

Analyzing the graph change, by deleting the edge between node 1 and node 9. By removing the edge between these nodes, the nodes 1,5,15,25 were deleted in the 5th core and the rest of the nodes were deleted in the 6th core.

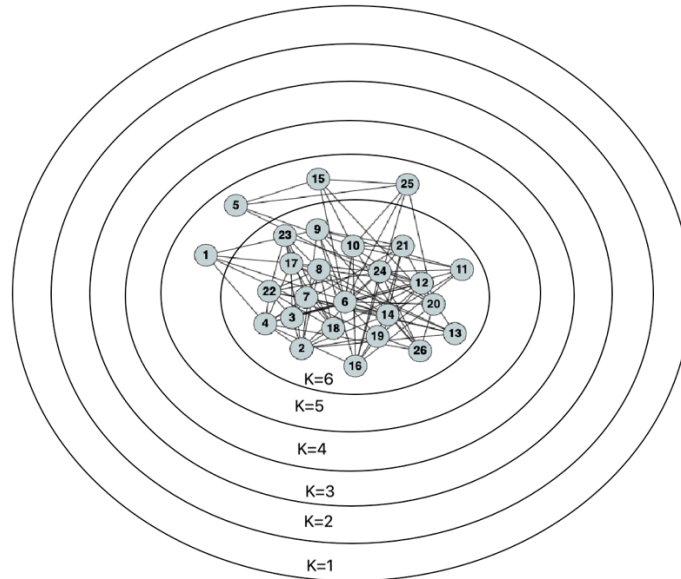


Fig.5. The K-core clusters in the graph after deleting edge between node 1 and 9.

Analyzing the graph change, by deleting the edge between node 5 and node 10. By removing the edge between these nodes, the node 5 is deleted in the 4th core, the nodes 15,25 were deleted in the 5th core and the rest of the nodes were deleted in the 6th core.

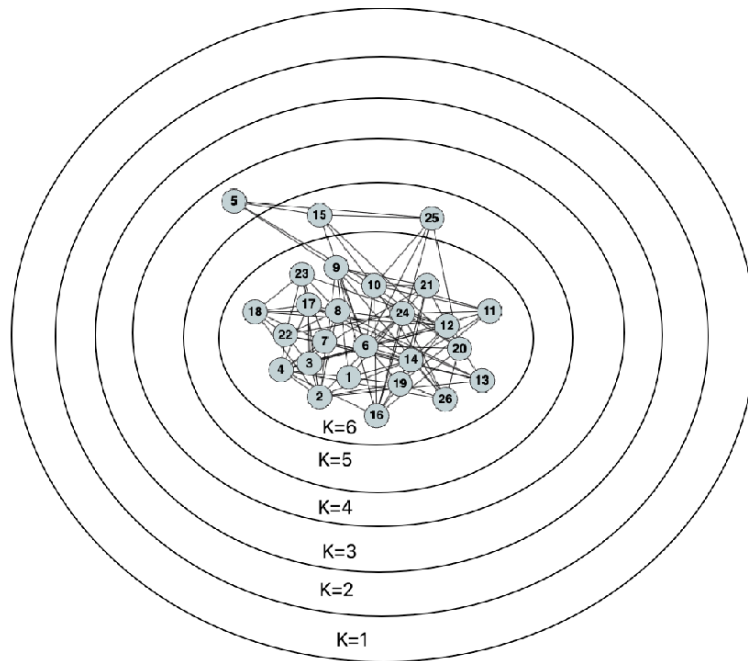


Fig.6. The K-core clusters in the graph after deleting edge between node 5 and 10.

Appendix

```
import java.util.*;

public class App {
    public static void main(String[] args) throws Exception {

        int[][] arr =
            {{0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,},
             {0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,},
             {0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,},
             {1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,},
             {0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,},
             {0,0,0,1,0,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,},
             {0,0,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,},
             {0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,},
             {1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,},
             {0,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,},
             {0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,},
             {0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,},
             {0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,},
             {1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,},
             {0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,},
             {0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,1,0,},
             {0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,0,0,0,},
             {0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,},
             {1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,},
             {0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,0,1,0,0,0,1,1,},
             {0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,},
             {0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,},
             {0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,},
             {1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,},
             {0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0,},
             {0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,0,},
            };

        k_core_network (arr,6);

    }

    // To find the adjacent node count
    public static int[] find_adjacent_nodes (int[][] adjacency_matrix){
```

```

int[] node_data =new int[26];

for(int i=0;i<adjacency_matrix.length;i++){
    int count=0;

    for(int j=0;j<adjacency_matrix.length;j++){

        if(adjacency_matrix[i][j]==1){
            count++;
        }
    }
    node_data[i]= count;
}
return node_data;
}

public static void k_core_network (int[][] adjacency_matrix, int k){
    System.out.println("The value of k is "+ k);
    List<Integer> deleted_nodes = new ArrayList<>();
    int n= -1;
    // To check if all the nodes have value greater than k else add to remove list
    while(!isEmpty(adjacency_matrix)){
        int[] adjacentnodes = find_adjacent_nodes(adjacency_matrix);
        n=deleted_nodes.size();
        for(int j=0;j<adjacentnodes.length;j++){
            //check for nodes
            if(adjacentnodes[j]<k && adjacentnodes[j]>0){
                System.out.println("deleted node "+j);
                deleted_nodes.add(adjacentnodes[j]+1);
                remove_node(adjacency_matrix, j);
            }
        }
        System.out.println("The size of deleted nodes:"+deleted_nodes.size());
        if(n == deleted_nodes.size()){
            break;
        }
    }

    if(isEmpty(adjacency_matrix)==0) {
        System.out.println("The List is empty");
    }

    else{
        if(deleted_nodes.size()==0){

```

```

        System.out.println("All nodes are greater than this core");
    }

}

// 0---> empty
// 1---->not empty

public static int isEmpty (int[][] adjacency_matrix){

    for(int i=0;i<adjacency_matrix.length;i++){
        for(int j=0;j<adjacency_matrix.length;j++){
            if(adjacency_matrix[i][j]==1){
                return 1;
            }
        }
    }
    return 0;
}

public static void remove_node(int[][] adjacency_matrix, int index){
    int node =index;
    for(int j=0;j<adjacency_matrix.length;j++){
        adjacency_matrix[node][j]=0;
        adjacency_matrix[j][node]=0;
    }
}
}

```

The code to generate the input matrix:

```

public class AdjMatrixGenerator {
    public static void main(String[] args) {
        String UTD_ID = "2021664655";
        StringBuilder bit_seq = new StringBuilder();

        for (int i = 0; i < UTD_ID.length(); i++) {
            char digit = UTD_ID.charAt(i);
            int num = Character.getNumericValue(digit);
            bit_seq.append(num % 2 == 0 ? "0" : "1");
        }

        StringBuilder bit_seq680 = new StringBuilder();

        int repeat=0;
    }
}

```

```

while(repeat<68){
    bit_seq680.append(bit_seq);
    repeat++;
}

String bit_seqString = bit_seq680.toString();

int[][] adjMatrix = new int[26][26];
int count = 0;
//Loop to convert 680 length sequence to 26x26 matrix
for (int i = 0; i < 26; i++) {
    String bit_seq_sub = bit_seqString.substring(count, count + 26);
    for (int j = 0; j < 26; j++) {
        adjMatrix[i][j] = Character.getNumericValue(bit_seq_sub.charAt(j));
    }
    count += 26;
}

//Loop to check isolated nodes
for (int i = 0; i < 26; i++) {
    boolean isolated = true;
    for (int j = 0; j < 26; j++) {
        if (adjMatrix[i][j] == 1) {
            isolated = false;
            break;
        }
    }
    if (isolated) {
        adjMatrix[i][0] = 1;
        adjMatrix[i][26-1] = 1;
        adjMatrix[0][i] = 1;
        adjMatrix[26-1][i] = 1;
    }
}

//Loop to check symmetry
for (int i = 0; i < 26; i++) {
    adjMatrix[i][i] = 0;
    for (int j = i + 1; j < 26; j++) {
        adjMatrix[i][j] = adjMatrix[j][i];
    }
}

// Print the adjacency matrix
System.out.print("{}");
for (int i = 0; i < 26; i++) {
    System.out.print("{}");
    for (int j = 0; j < 26; j++) {
        System.out.print(adjMatrix[i][j] + ",");
    }
}

```

```
        System.out.println("{}");  
    }  
    System.out.print("{}");  
    }  
}
```

References

1. <https://www.youtube.com/watch?v=8sNZ5d8eNC8>
2. <https://www.programiz.com/dsa/graph-adjacency-matrix>
3. https://graphonline.ru/en/create_graph_by_matrix
4. <https://paint-x.com/>