CS 6385.0W1 - TE 6385.0W1

ALGORITHMIC ASPECTS OF TELECOMMUNICATION NETWORKS

PROJECT-1

Submitted by:

Shubham Singh

sxs190413

INDEX

1.Objective
2.Algorithms
3.Flowchart
4.Results
5.Graphs
6.Readme File
7.Reference
8.Appendix-1 (Source Code)

1. OBJECTIVE

The objective of this project is to create different network topologies with varying maximum outgoing degrees (k) for each topology and to investigate the effect of varying k on the density and maximum cost of the topology's links.

Uses randomly generated samples of the cost and demand of traffic between each pair of nodes in each topology as input, as stated in the project description.

The project leverages the Djikstra algorithm, that finds the shortest routes between each pair of nodes for any value of k and uses the shortest path based fast solution method to find the maxcost of the network.

For each value of k, the density and MaxCost, or the sum total cost of all edges that form shortest pathways in the topology, are calculated.

2. ALGORITHMS

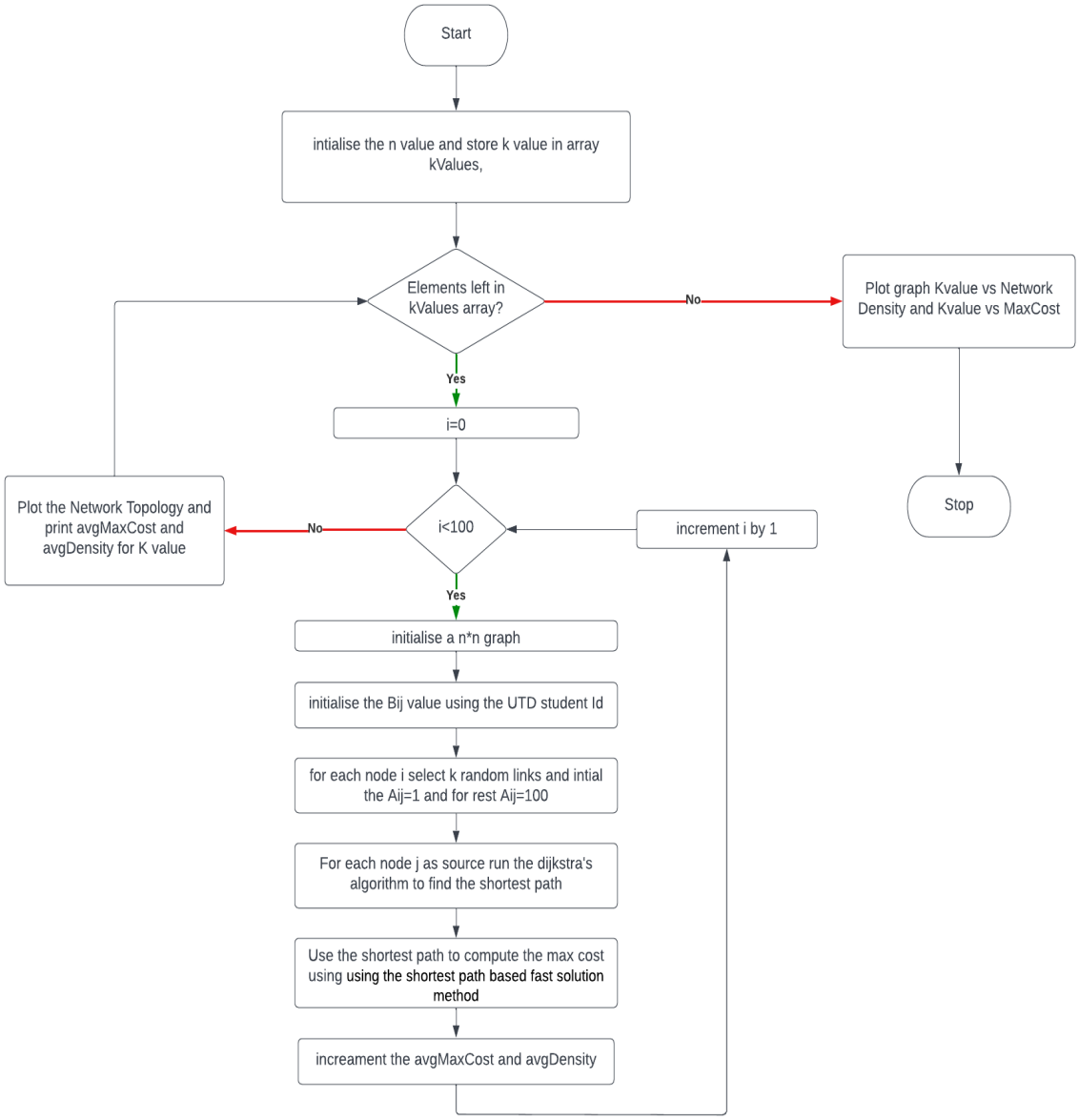
Dijkstra’s algorithm

In the project I have made use of the Dijkstra’s algorithm which is used to find the shortest path between two nodes, called source node and destination node. Dijkstra’s algorithm used for graphs that don’t have negative weighted edges.

Fast Solution Method

With edge weights aij, find the shortest route between each pair of nodes (k, l). Any conventional shortest-path algorithm can be used. This project uses Dijkstra's algorithm. Set edge (i, j) capacity which is on the shortest path of (k,l) as to the total sum of the all Bkl values for which (i,j) is on the lowest-cost.

3. FLOWCHART


Start
intialise the n value and store k value in array kValues,
Elements left in kValues array?
No
Plot graph Kvalue vs Network Density and Kvalue vs MaxCost
Yes
i=0
Stop
Plot the Network Topology and print avgMaxCost and avgDensity for K value
No
i<100
increment i by 1
Yes
initialise a n*n graph
initialise the Bij value using the UTD student Id
for each node i select k random links and intial the Aij=1 and for rest Aij=100
For each node j as source run the dijkstra's algorithm to find the shortest path
Use the shortest path to compute the max cost using using the shortest path based fast solution method
increament the avgMaxCost and avgDensity

4. RESULT

Max cost of the network for K-value 3: $8074.57
Avg density of the network for K-value 3: 0.19204761904761902

Max cost of the network for K-value 4: $3664.9
Avg density of the network for K-value 4: 0.20914285714285685

Max cost of the network for K-value 5: $2615.68
Avg density of the network for K-value 5: 0.253095238095238

Max cost of the network for K-value 6: $2090.35
Avg density of the network for K-value 6: 0.297857142857143

Max cost of the network for K-value 7: $1891.79
Avg density of the network for K-value 7: 0.3431428571428572

Max cost of the network for K-value 8: $1809.66
Avg density of the network for K-value 8: 0.38569047619047614

Max cost of the network for K-value 9: $1744.75
Avg density of the network for K-value 9: 0.4289047619047615

Max cost of the network for K-value 10: $1687.58
Avg density of the network for K-value 10: 0.4667619047619045

Max cost of the network for K-value 11: $1631.16
Avg density of the network for K-value 11: 0.5069761904761906

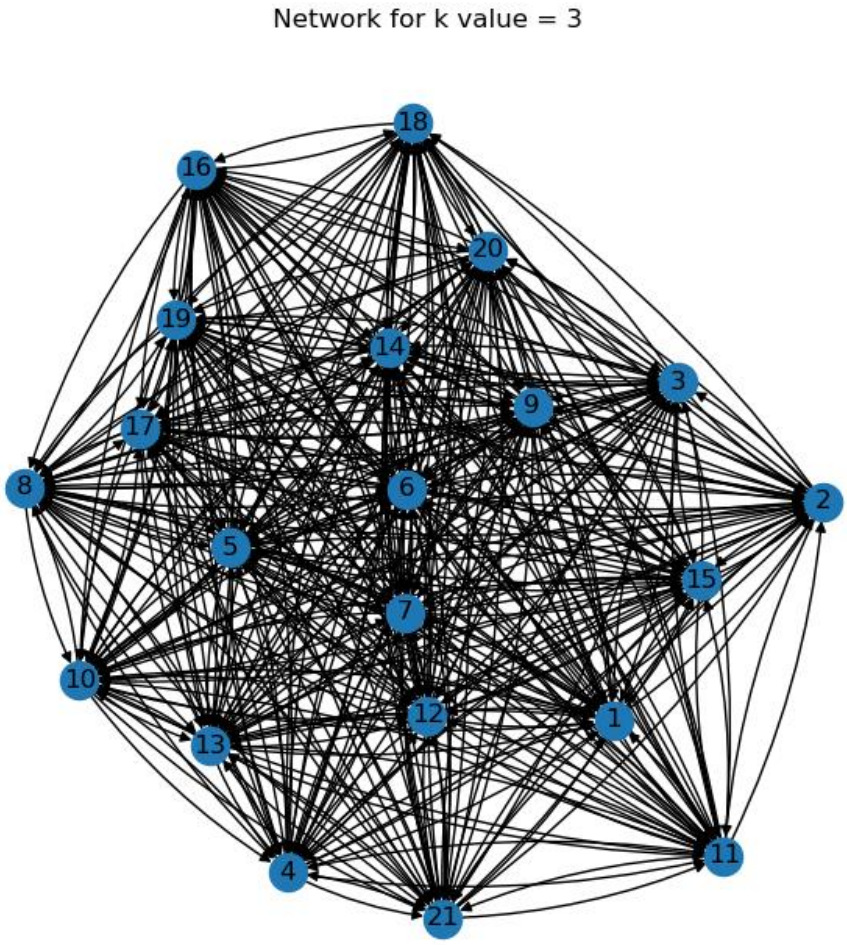
Max cost of the network for K-value 12: $1575.67
Avg density of the network for K-value 12: 0.5483809523809526

Max cost of the network for K-value 13: $1516.59
Avg density of the network for K-value 13: 0.5878571428571431

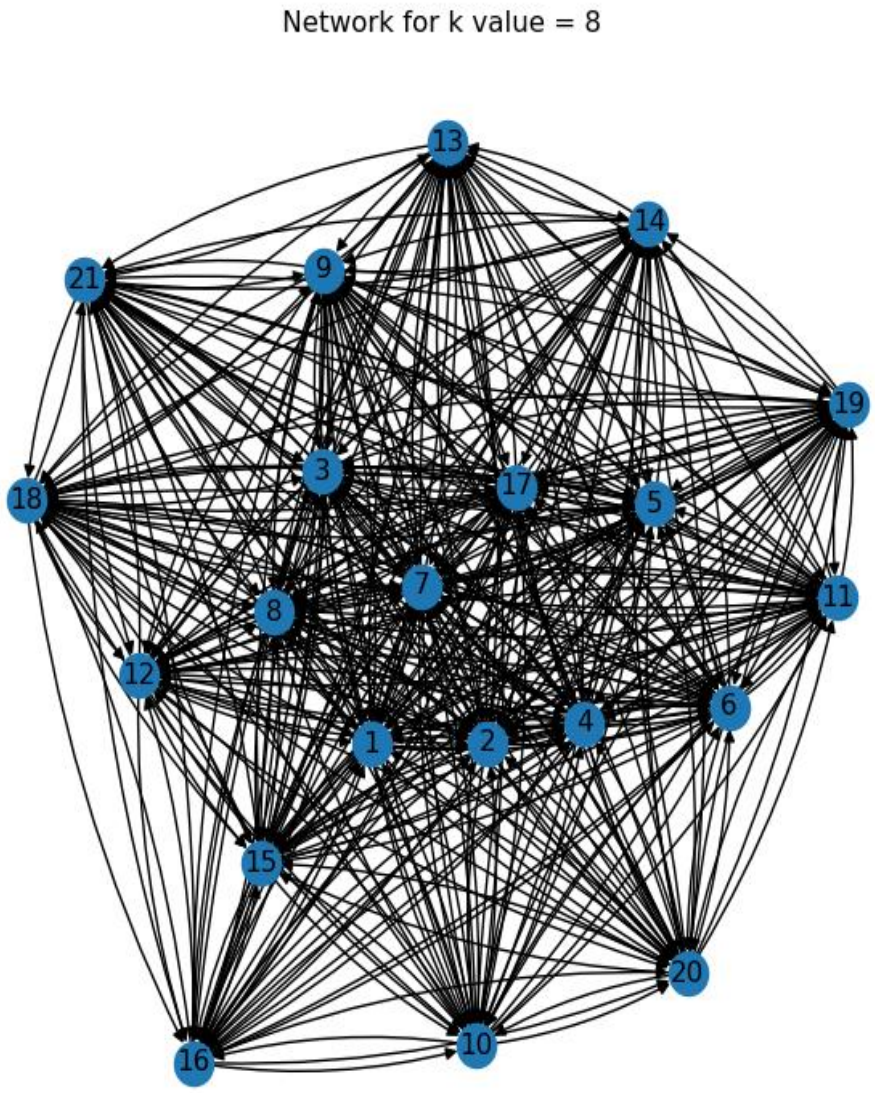
Max cost of the network for K-value 14: $1463.34
Avg density of the network for K-value 14: 0.6265952380952382

5. GRAPHS

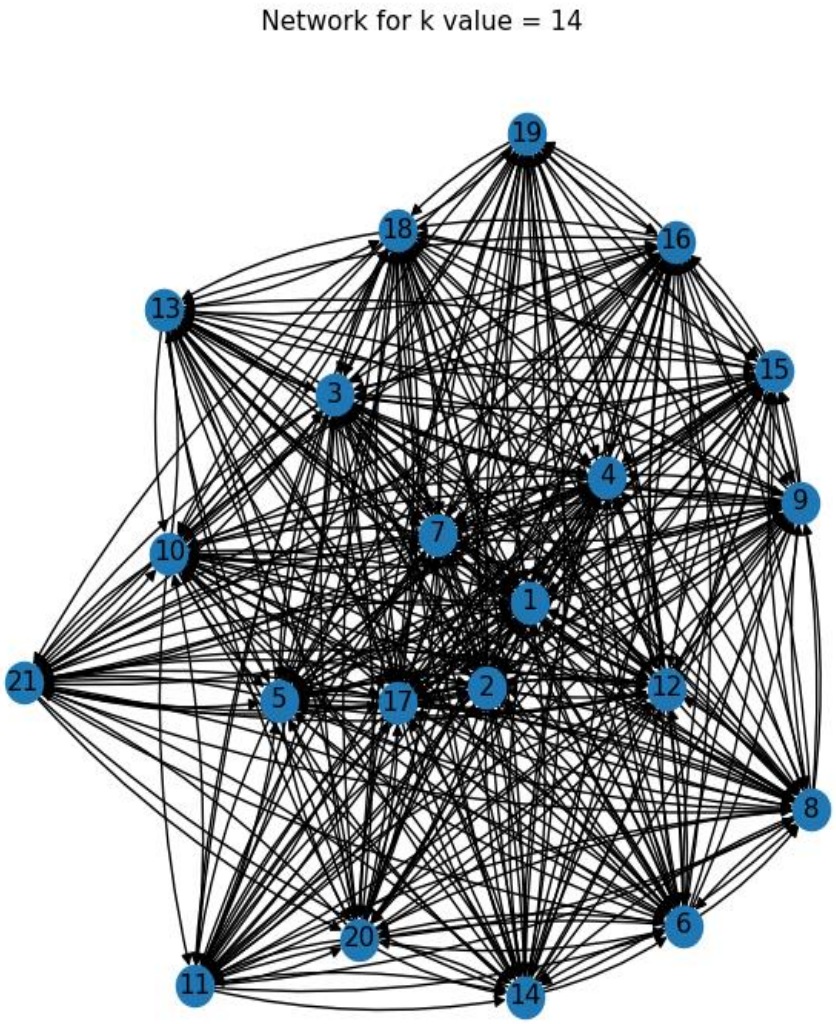
- Network topology graph for k-value 3



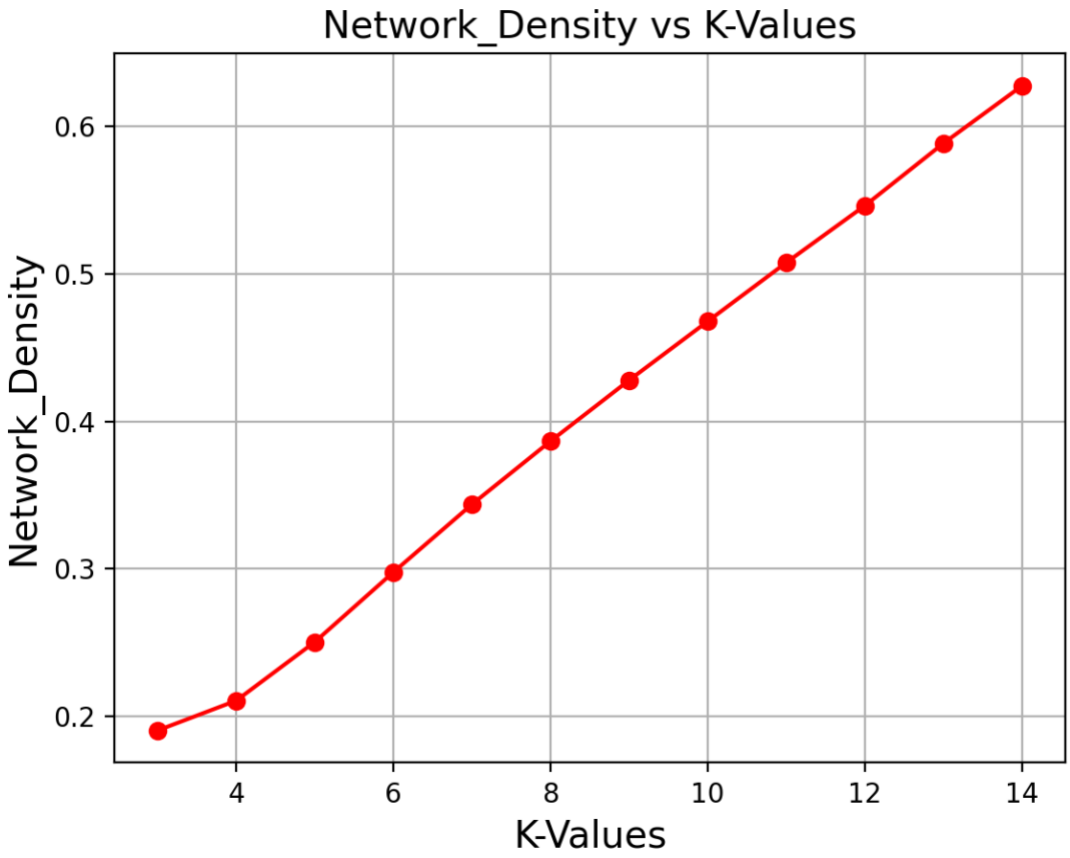
- Network topology graph for k-value 8



- Network topology graph for k-value 14

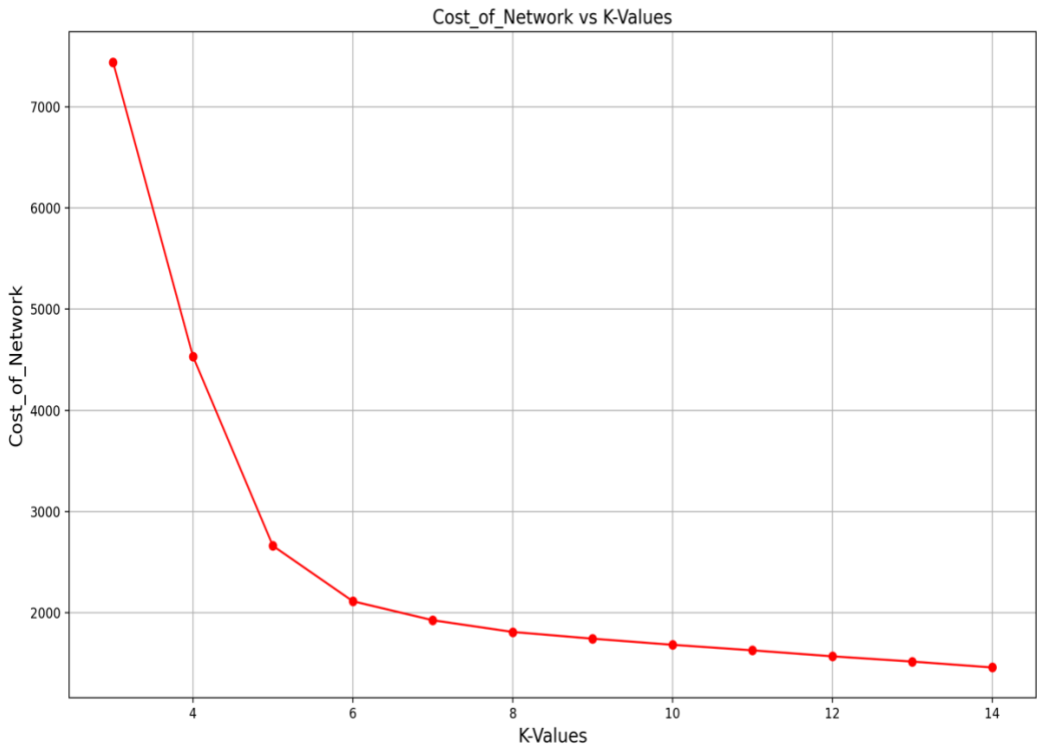


- Network Density VS K-Values



As the value of the K increase, which corresponds to each node having more links to choose from when finding the shortest path from a node i to node j. This results in network having more links and thus the network density increase as the value of the K increase.

- Min Cost of Network VS K-Values



As the value of the K increase, the cost to build the network decreases. This occurs as the K value increase the number of ways to find the shortest path increase which results in shorter shortest path are found. This results in decrease of the overall cost as we have shorter path compared to previous smaller K value.

6. ReadMe File

- Install python 3 and pip3 using this link.
- Move to the folder where the source code is saved.
- Run the below command to install the dependences
    - pip3 install network
    - pip3 install matplotlib
    - pip3 install random
- Save the project file with name network.py
- Run the file using the below command
    - python3 network.py

7. Reference

- Shortest path algorithm. Link

Appendix – 1

1. Source Code

```
import networkx as nx
import random
import matplotlib.pyplot as plt

class Network:
    def __init__(self,numberOfNodes,k):
        self.numberOfNodes=numberOfNodes
        self.k=k
        self.cometId="2021539425"
        self.trafficGraph=nx.DiGraph(directed=True)
        self.costGraph=nx.DiGraph(directed=True)
        self.linkGraph=nx.DiGraph(directed=True)
        self.cost=0
        self.numberOfEdge=0
        self.avgCost=0
        self.avgNumberOfEdge=0

    def developGraph(self):
        cometIdList=[int(num) for num in self.cometId]
        bitValueList=['#']
        for node in range(self.numberOfNodes):
            bitValueList.append(cometIdList[node%10])
        for node in range(1,self.numberOfNodes+1):
            self.trafficGraph.add_node(node)
            self.costGraph.add_node(node)
            self.linkGraph.add_node(node)

        for iNode in range(1,self.numberOfNodes+1):
            for jNode in range(1,self.numberOfNodes+1):
                if iNode==jNode:
                    continue
                Bij=abs(bitValueList[iNode]-bitValueList[jNode])
                self.trafficGraph.add_edge(iNode,jNode,weight=Bij)
```

```
        for iNode in range(1,self.numberOfNodes+1):
            edgeChoice=list(range(1,self.numberOfNodes+1))
            edgeChoice.remove(iNode)
            shortestNode=set(random.sample(edgeChoice,self.k))
            for jNode in range(1,self.numberOfNodes+1):
                if iNode==jNode :
                    continue
                if jNode  in shortestNode:
                    self.costGraph.add_edge(iNode,jNode ,weight=1)
                else:
                    self.costGraph.add_edge(iNode,jNode,weight=100)

    def findShortestPath(self):
        capacityHashMap={}
        for iNode in range(1,self.numberOfNodes+1):
            for jNode  in range(1,self.numberOfNodes+1):
                if iNode==jNode :
                    continue
                shortestPath=nx.dijkstra_path(self.costGraph, source=iNode,
target=jNode )
                for curNode in range(len(shortestPath)-1):
                    src,dst=shortestPath[curNode],shortestPath[curNode+1]
                    if (src,dst) in capacityHashMap:

capacityHashMap[(src,dst)]+=self.trafficGraph.get_edge_data(iNode,jNode
)['weight']
                    else:

capacityHashMap[(src,dst)]=self.trafficGraph.get_edge_data(iNode,jNode
)['weight']
        for src,dst in capacityHashMap:
            if capacityHashMap[(src,dst)]!=0:
                self.numberOfEdge+=1

self.cost+=(int(capacityHashMap[(src,dst)])*self.costGraph.get_edge_data(src,dst)
['weight'])

self.linkGraph.add_edge(src,dst,weight=int(capacityHashMap[(src,dst)]))
```

```
    def drawNetwork(self,k):
        plt.figure(figsize=(8,8))
        plt.title(" Network for k value = {}".format(k))
        pos = nx.spring_layout(self.linkGraph)
        nx.draw(self.linkGraph,pos,connectionstyle='arc3, rad = 0.1',with_labels =
True)
        plt.savefig("Network_graph_{}.png".format(k))
        plt.show()
def plotMetricsLineGraph(xList,yList,xLabel,yLable):
    plt.clf()
    plt.plot(xList,yList, color='red', marker='o')
    plt.title('{} vs {}'.format(yLable,xLabel), fontsize=14)
    plt.xlabel(xLabel, fontsize=14)
    plt.ylabel(yLable, fontsize=14)
    plt.grid(True)
    plt.savefig('{}_vs_{}.png'.format(yLable,xLabel))
    plt.show()

if __name__ == "__main__":
    kValues=[]
    costList=[]
    densityList=[]
    n=21
    for k in range(3,n):
        if k>14:
            break
        kValues.append(k)
    for k in kValues:
        obj=Network(n,k)
        for _ in range(100):
            obj.cost=0
            obj.numberOfEdge=0
            obj.developGraph()
            obj.findShortestPath()
            obj.avgCost+=obj.cost
            obj.avgNumberOfEdge+=obj.numberOfEdge/(n*(n-1))
        if k in [3,8,14]:
            obj.drawNetwork(k)
        obj.avgCost=obj.avgCost/100
        obj.avgNumberOfEdge=obj.avgNumberOfEdge/100
```

```
        costList.append(obj.avgCost)
        densityList.append(obj.avgNumberOfEdge)
        print("Max cost of the network for K-value {}: ${}".format(k,obj.avgCost))
        print("Avg density of the network for K-value {}:
{}".format(k,obj.avgNumberOfEdge))

    plotMetricsLineGraph(kValues,costList,"K-Values","Cost_of_Network")
    plotMetricsLineGraph(kValues,densityList,"K-Values","Network_Density")
```