Source Code and Outputs

1. Reading from file:

import networkx as nx

import matplotlib.pyplot as plt

G = nx.Graph()

print("Choose any input file")

print(" 1. input10")

print(" 2. input20")

print(" 3. input30")

print(" 4. input40")

print(" 5. input50")

print(" 6. input60")

print(" 7. input70")

print(" 8. input80")

print(" 9. input90")

print(" 10. input100")

file = input("Enter file : ")

if file == "1":

f = open("input10.txt","r")

nodes = 10

elif file == "2":

f = open("input20.txt","r")

nodes = 20

elif file == "3":

f = open("input30.txt","r")

nodes = 30

elif file == "4":

f = open("input40.txt","r")

nodes = 40

elif file == "5":

f = open("input50.txt","r")

nodes = 50

elif file == "6":

f = open("input60.txt","r")

nodes = 60

elif file == "7":

f = open("input70.txt","r")

nodes = 70

elif file == "8":

f = open("input80.txt","r")

nodes = 80

elif file == "9":

f = open("input90.txt","r")

nodes = 90

elif file == "10":

f = open("input100.txt","r")

nodes = 100

for i in range(0,nodes+5):

f.readline()

for line in f:

arr\_list = []

for word in line.split():

arr\_list.append(word)

if not line.strip():

break

for x in range(1,len(arr\_list),4):

w = float(arr\_list[x+2])/pow(10,7)

u = arr\_list[0]

v = arr\_list[x]

if G.has\_edge(u,v)==0:

G.add\_edge(u,v,weight=w)

elif G.has\_edge(u,v)==1:

if w < G[u][v]['weight']:

G[u][v]['weight'] = w

pos=nx.circular\_layout(G)

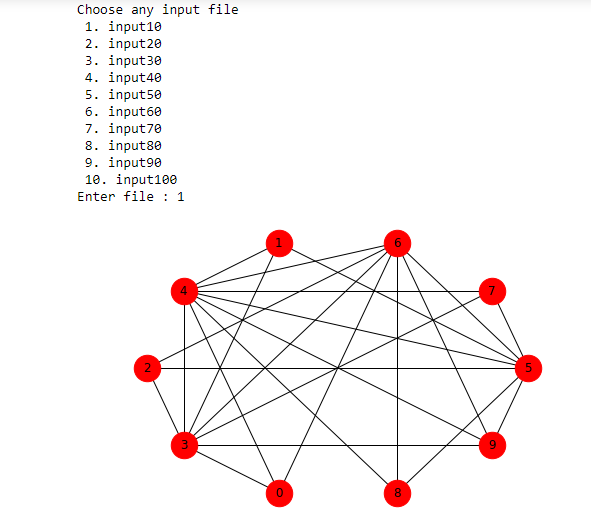
nx.draw(G,pos,with\_labels=1,node\_size=700)

plt.show()

edgesss = G.number\_of\_edges()

print("Total no. of edges : " + str(edgesss))

start = f.readline()



1. Prims Algorithm:

print("Prim's Algorithm")

Prim = nx.Graph()

visited=['1']

while (len(Prim) != len(G)):

min = 99999999999/pow(10,7)

i='0'

j='0'

w=0

for u in G.node():

if u in visited:

for v in G.node():

if v not in visited and G.has\_edge(u,v)==1 :

if min > G[u][v]['weight']:

min = G[u][v]['weight']

i = u

j = v

w = G[u][v]['weight']

visited.append(j)

Prim.add\_edge(i,j,weight=w)

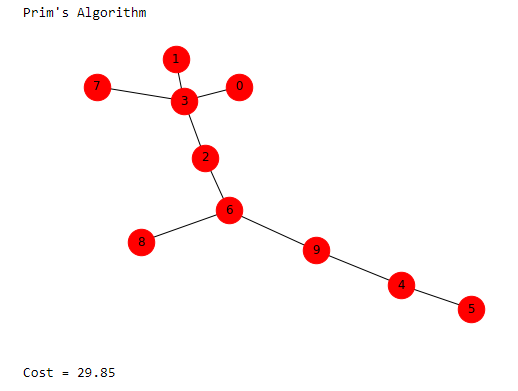
nx.draw(Prim,with\_labels=1,node\_size=700)

plt.show()

prims\_cost = 0

for (u,v,w) in Prim.edges(data=True):

prims\_cost = prims\_cost + Prim[u][v]['weight']

print('Cost = '+ str(prims\_cost) + ' Mbps')

1. Kruskal Algorithm:

from collections import defaultdict

class Graph:

def \_\_init\_\_(self,vertices):

self.V= vertices

self.graph = []

def addEdge(self,u,v,w):

self.graph.append([u,v,w])

def find(self, parent, i):

if parent[i] == i:

return i

return self.find(parent, parent[i])

def union(self, parent, rank, x, y):

xroot = self.find(parent, x)

yroot = self.find(parent, y)

if rank[xroot] < rank[yroot]:

parent[xroot] = yroot

elif rank[xroot] > rank[yroot]:

parent[yroot] = xroot

else :

parent[yroot] = xroot

rank[xroot] += 1

def KruskalMST(self):

result =[]

i = 0

e = 0

self.graph = sorted(self.graph,key=lambda item: item[2])

parent = [] ; rank = []

for node in range(self.V):

parent.append(node)

rank.append(0)

while e < self.V -1 :

u,v,w = self.graph[i]

i = i + 1

x = self.find(parent, u)

y = self.find(parent ,v)

if x != y:

e = e + 1

result.append([u,v,w])

self.union(parent, rank, x, y)

print('Following are the edges in the constructed MST')

for (u,v,weight) in result:

print ("%d -- %d == %.2f" % (u,v,weight))

return result

print("Kruskal Algorithm")

g = Graph(len(G))

for (u,v,w) in G.edges(data=True):

g.addEdge(int(u),int(v),float(w['weight']))

x = g.KruskalMST()

kruskal\_cost=0

kruskal = nx.Graph()

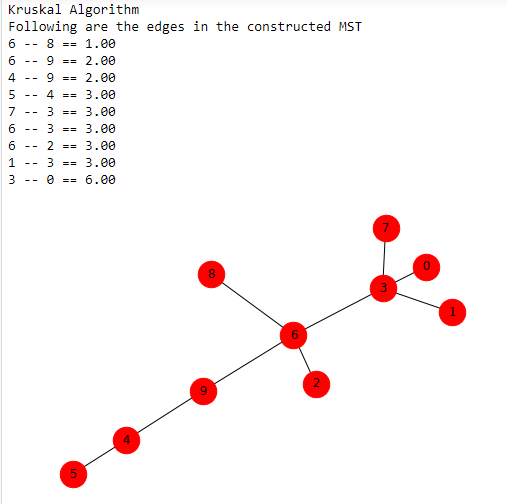
for (u,v,w) in x:

kruskal.add\_edge(u,v,weight=w)

kruskal\_cost = kruskal\_cost + w ;

nx.draw(kruskal,with\_labels=1,node\_size=700)

plt.show()

print('Cost = '+ str(kruskal\_cost) + ' Mbps')

1. Bellman Ford’s Algorithm:

print("Bellman Ford's Algorithm")

print(" ")

Ford = nx.Graph()

rows, cols = (nodes, nodes)

edge\_weight = [[0.0 for i in range(cols)] for j in range(rows)]

for i in G.nodes():

for j in G.nodes():

if G.has\_edge(i,j)==1:

edge\_weight[int(i)][int(j)] = G[i][j]['weight']

path = nx.single\_source\_bellman\_ford\_path(G,'1')

Ford\_cost=0

for i in path:

for j in range(len(path[i])-1):

Ford\_cost = Ford\_cost + edge\_weight[int(path[i][j])][int(path[i][j+1])]

print('Cost = '+ str(Ford\_cost) + " Mbps")

for u in path:

if len(path[u])==1:

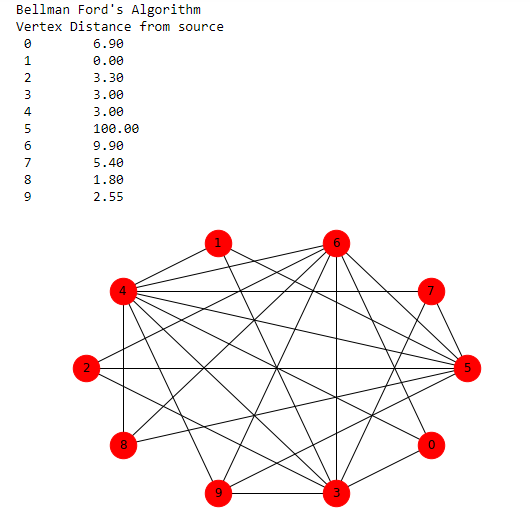
continue

else:

Ford.add\_edge(str(path[u][len(path[u])-2]), str(path[u][len(path[u])-1]), weight = edge\_weight[int(path[u][len(path[u])-2])][int(path[u][len(path[u])-1])])

pos=nx.circular\_layout(Dijkstra)

nx.draw(Ford,pos,with\_labels=1,node\_size=700)

plt.show() 

1. Dijkstra Algorithm

print("Dijkstra's Algorithm")

print(" ")

Dijkstra = nx.Graph()

rows, cols = (nodes, nodes)

edge\_weight = [[0.0 for i in range(cols)] for j in range(rows)]

for i in G.nodes():

for j in G.nodes():

if G.has\_edge(i,j)==1:

edge\_weight[int(i)][int(j)] = G[i][j]['weight']

length, path = nx.single\_source\_dijkstra(G,'1')

Dijkstra\_count = 0

for i in length:

Dijkstra\_count = Dijkstra\_count+length[i]

print('Cost = '+ str(Dijkstra\_count) + " Mbps")

for u in path:

if len(path[u])==1:

continue

else:

Dijkstra.add\_edge(str(path[u][len(path[u])-2]), str(path[u][len(path[u])-1]), weight = edge\_weight[int(path[u][len(path[u])-2])][int(path[u][len(path[u])-1])])

pos=nx.circular\_layout(Dijkstra)

nx.draw(Dijkstra,pos,with\_labels=1,node\_size=700)

plt.show()

1. Floyd Warshall Algorithm:

print("Floyd Warshall's Algorithm")

print(" ")

Floyd = nx.Graph()

INF = 99999999999/pow(10,7)

rows, cols = (nodes, nodes)

cost = [[0.0 for i in range(cols)] for j in range(rows)]

for u in G.nodes():

for v in G.nodes():

if u == v :

cost[int(u)][int(v)] = 0

elif G.has\_edge(u,v)==1:

cost[int(u)][int(v)] = G[u][v]['weight']

elif G.has\_edge(u,v)==0:

cost[int(u)][int(v)] = INF

print("Weights of edges before applying algorithm")

for i in cost:

print(i)

for u in G.nodes():

for v in G.nodes():

for w in G.nodes():

if cost[int(v)][int(u)] + cost[int(u)][int(w)] < cost[int(v)][int(w)]:

cost[int(v)][int(w)] = (cost[int(v)][int(u)] + cost[int(u)][int(w)])

print(" ")

print("Weights of edges after applying algorithm")

for i in cost:

print(i)

for (u,v) in G.edges():

Floyd.add\_edge(u,v,weight=str(cost[int(u)][int(v)]))

pos=nx.circular\_layout(Floyd)

nx.draw(Floyd,pos,with\_labels=1,node\_size=700)

plt.show()

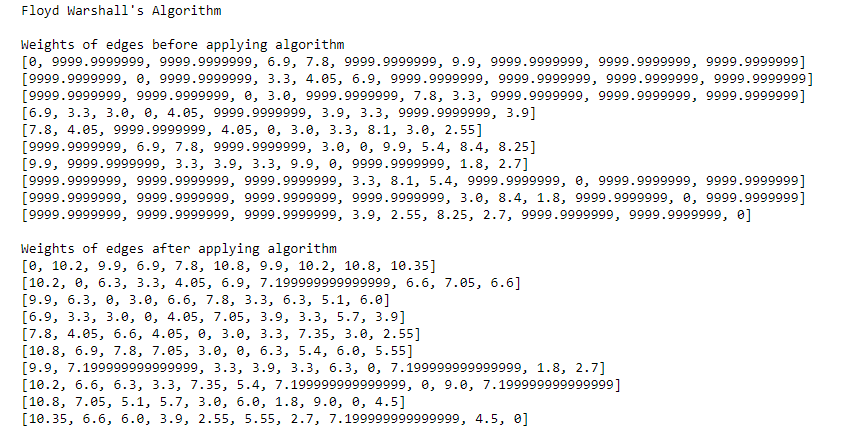
floyd\_cost=0

for i in range(rows):

for j in range(1,cols):

floyd\_cost = floyd\_cost + cost[i][j]

print(floyd\_cost)



1. Clustering Coefficient:

print("Clustering Coefficient")

print(" ")

#local = nx.clustering(G)

avg = nx.average\_clustering(G)

#for i in G.nodes():

# print("Local Clustering : " + i + " ---- " + str(local[i]))

#print(" ")

print("Final Cost : " + str(avg) + " Mbps")

