Project

on

ESTABLISHING ELECTRIC GRID IN TOWN USING BORUKA’S ALGORITHM

Design and Analysis of Algorithms

By

V.ABHIRAM (2010030180)

K.SIDHARTH RAO (2010030443)

A .RAGHAVENDRA GOUD (2010030394)

R .VIVEK VARDHAN REDDY (2010030142)

under the supervision of

# P . Sree Lakshmi



Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India.March, 2022

**DECLARATION**

The Project Report entitled “Establishing Electric grid using Bourka’s algorithm” is a record of bonafide work of V.Abhiram (2010030180), K.Sidharth Rao (2010030443),A .Raghavendra Goud (2010030394), R .Vivek Vardhan Reddy (2010030142) , submitted as a requirement for the completion of the course Design and Analysis of Algorithm in the Department of Computer Science and Engineering to the K L University, Hyderabad. The results embodied in this report have not been copied from any other Departments/University/Institute.

**CERTIFICATE**

This is to certify that the Project Report entitled “Establishing Electric grid using Bourka’s algorithm” is being submitted by V.Abhiram (2010030180), K.Sidharth Rao (2010030443),A .Raghavendra Goud (2010030394),R .Vivek Vardhan Reddy (2010030142) as a requirement for the completion of the course **Design and Analysis of Algorithms** in the Department of Computer Science and Engineering, K L University, Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/Institute.

## Signature of the Supervisor

P . Sree Lakshmi

## Signature of the HOD Signature of the Examiner

**ACKNOWLEDGEMENT**

First and foremost, we thank the lord almighty for all his grace & mercy showered upon us, for completing this project successfully.

We take grateful opportunity to thank our beloved Founder and Chairman who has given constant encouragement during our course and motivated us to do this project. We are grateful to our Principal **Dr. L. Koteswara Rao** who has been constantly bearing the torch for all the curricular activities undertaken by us.

We pay our grateful acknowledgement & sincere thanks to our Head of the Department **Dr. Chiranjeevi Manike** for his exemplary guidance, monitoring and constant encouragement throughout the course of the project. We thank **P. Sree Lakshmi** of our department who has supported throughout this project holding a position of supervisor.

We whole heartedly thank all the teaching and non-teaching staff of our department without whom we won’t have made this project a reality. We would like to extend our sincere thanks especially to our parents, our family members and friends who have supported us to make this project a grand success.

**ABSTRACT**

Optimization is important in an algorithm. It will show us the best path which has to be travelled to obtain the best and optimal solution, thereby saving the operational costs of an activity. In the Minimum Spanning Tree, the goal is to achieve how all vertices are connected with the smallest weights. There have been many algorithms which have been developed in obtaining the minimum path which has to be covered to complete a certain task. Such developed algorithms have many uses and one such use can be developing an electric grid in a town. We can choose the towns which are having the least distance among them, and thus, all those cities can be connected with each other, and further it will be connected to the electric grid. The purpose of this study is to find out the Primary electricity distribution network graph model and correct algorithm to determine the minimum spanning tree.

**INDEX**

Page No.

1. DECLARATION
2. ACKNOWLEDGEMENT
3. ABSTRACT
4. INTRODUCTION
5. LITERATURE SURVEY
6. HARDWARE AND SOFTWARE REQUREMENTS
7. FUNCTIONAL AND NON FUNCTIONAL REQUIRMENTS
   1. Algorithm
   2. Flowchart
   3. Numerical problem
8. CODE
9. RESULT
10. CONCLUSION
11. REFERENCES

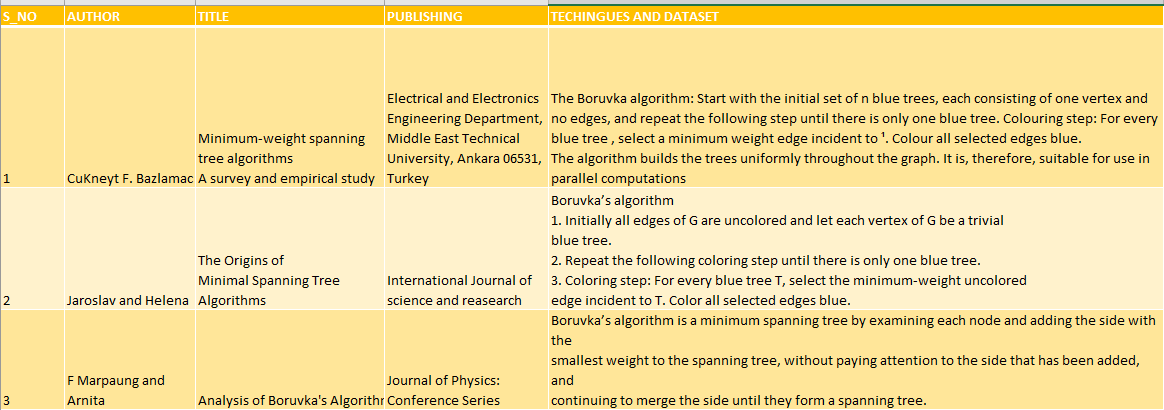
**INTRODUCTION**

Let’s take a situation where we have a city which has so many towns in it, and the distances between pairs of the town are known. The goal is to create a system of electric power lines that would minimize the total distance (and thus the construction cost), yet reach every town. In this scenario, it is not necessary to connect each town to the “source of electricity” directly – it is enough to connect it to another town that already got the power.

The problem which we chose comes under the domain of GREEDY METHOD. Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So, the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

In this scenario, we have to find the MINIMUM COST SPANNING TREE. The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees.

**LITERATURE SURVEY**

****

**FUNCTIONAL AND NON FUNCTIONAL REQUIRMENTS**

**ALGORITHM**

**BORUVKA’S ALGORITHM**

Boruvka's algorithm is a greedy algorithm for finding a minimum spanning tree in a graph. The algorithm begins by finding the minimum-weight edge incident to each vertex of the graph, and adding all of those edges to the tree. Then, it repeats a similar process of finding the minimum-weight edge from each tree constructed so far to a different tree, and adding all of those edges to the forest. Each repetition of this process reduces the number of trees, within each connected component of the graph, to at most half of this former value, so after logarithmically many repetitions the process finishes.

This algorithm can be applied in our problem in the following way. For each town, connect it with its nearest neighboring town. Now, for each of the resulting groups of towns, connect them to their nearest neighbor... Proceed until there is just one group left.

algorithm Borůvka is

input: A weighted undirected graph G = (V, E).

output: F, a minimum spanning forest of G.

Initialize a forest F to (V, E') where E' = {}.

completed := false

while not completed do

Find the connected components of F and assign to each vertex its component

Initialize the cheapest edge for each component to "None"

for each edge uv in E, where u and v are in different components of F:

let wx be the cheapest edge for the component of u

if is-preferred-over(uv, wx) then

Set uv as the cheapest edge for the component of u

let yz be the cheapest edge for the component of v

if is-preferred-over(uv, yz) then

Set uv as the cheapest edge for the component of v

if all components have cheapest edge set to "None" then

// no more trees can be merged -- we are finished

completed := true

else

completed := false

for each component whose cheapest edge is not "None" do

Add its cheapest edge to E'

function is-preferred-over(edge1, edge2) is

return (edge2 is "None") or

(weight(edge1) < weight(edge2)) or

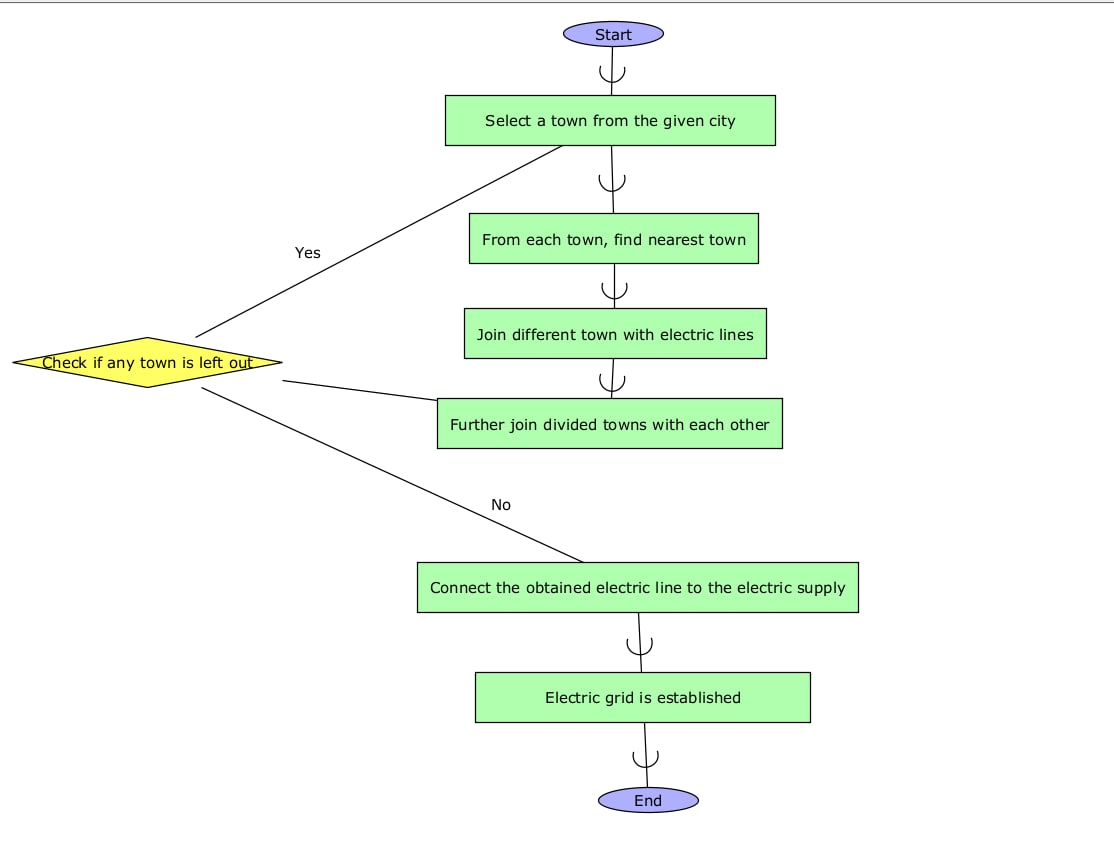
(weight(edge1) = weight(edge2) and tie-breaking-rule(edge1, edge2))

function tie-breaking-rule(edge1, edge2) is

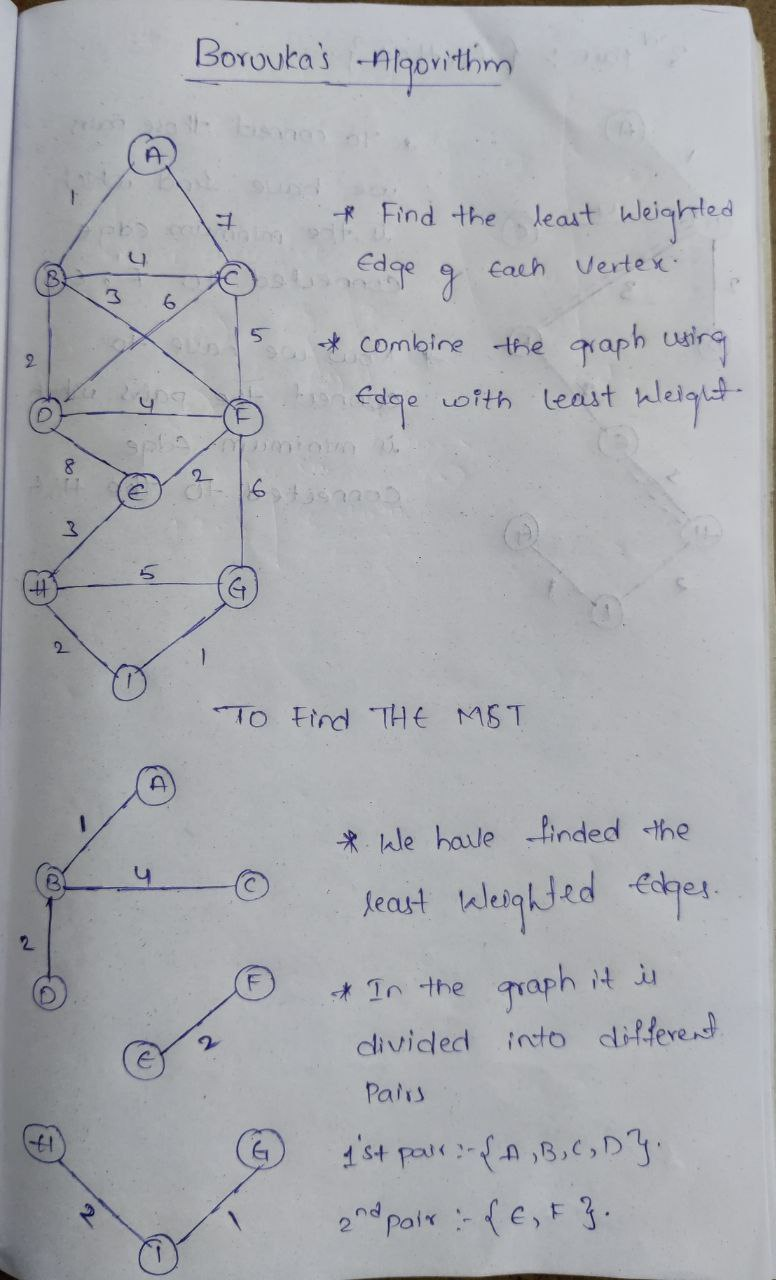
The tie-breaking rule; returns true if and only if edge1

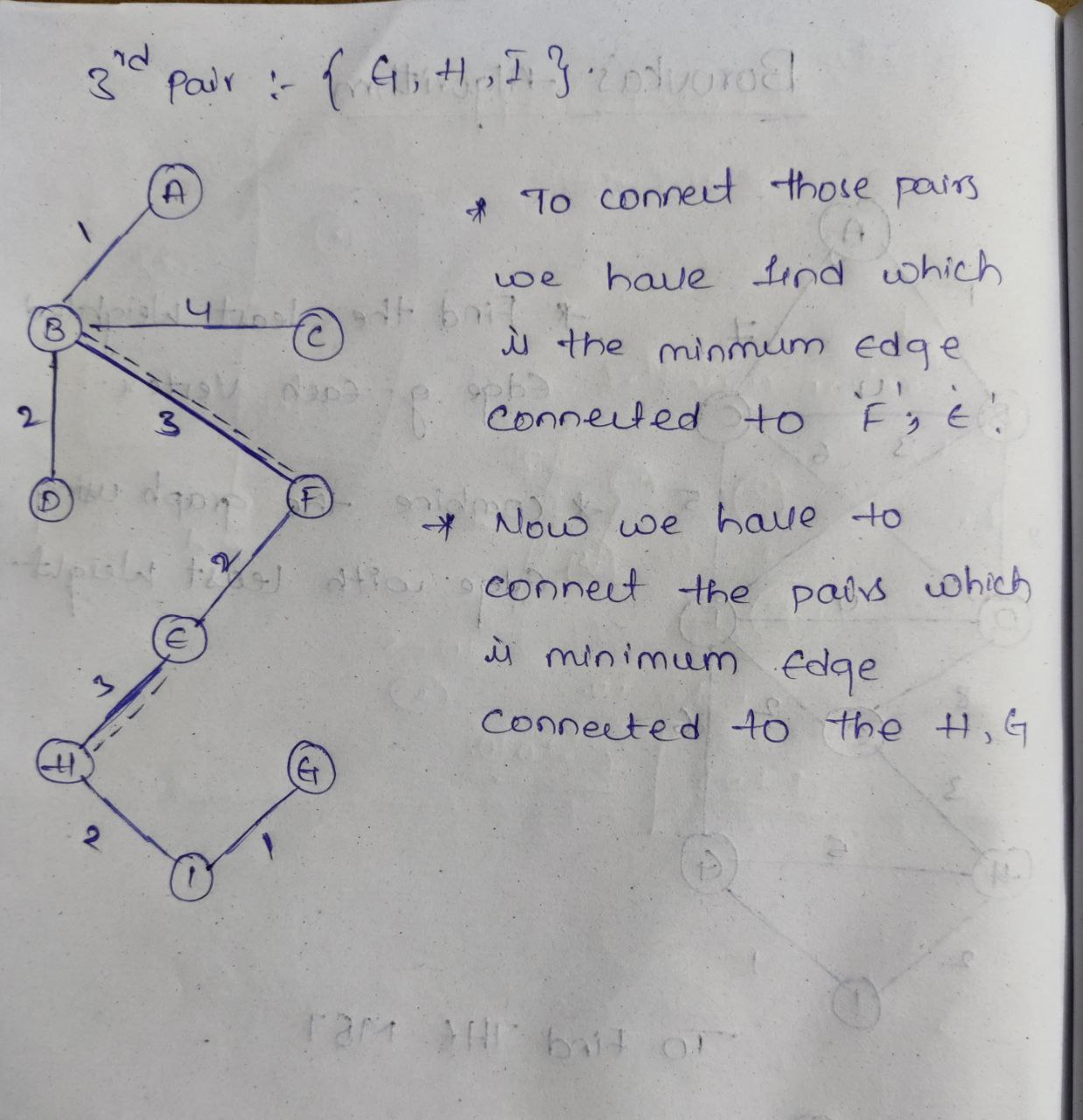
is preferred over edge2 in the case of a tie.

**FLOWCHART**



**NUMERICAL PROBLEM**

******

******

**CODE**

Boruvkas.py

class BoruvkasAlgorithm:

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

self.graph = graph

self.graph\_edges = graph.edges\_with\_weight

self.V = len(self.graph.nodes)

self.cheapest = [-1] \* self.V

self.cheapest\_nodes,self.parents, self.rank = [], [], []

self.MST\_weight = 0

def find\_cheapest(self):

for i in range(len(self.graph.edges\_with\_weight)):

u,v, w = self.graph.edges\_with\_weight[i]

set1, set2 = u, v

if set1 != set2:

if self.cheapest[set1] == -1 or self.cheapest[set1][2] > w:

self.cheapest[set1] = [u, v, w]

if self.cheapest[set2] == -1 or self.cheapest[set2][2] > w:

self.cheapest[set2] = [u, v, w]

def union(self,x, y):

xroot = self.find(self.parents, x)

yroot = self.find(self.parents, y)

if self.rank[xroot] < self.rank[yroot]:

self.parents[xroot] = yroot

elif self.rank[xroot] > self.rank[yroot]:

self.parents[yroot] = xroot

else:

self.parents[yroot] = xroot

self.rank[xroot] += 1

def find\_MST(self):

numTrees = self.V

MSTweight = 0

for n in range(self.V):

self.parents.append(n)

self.rank.append(0)

while numTrees > 1:

self.find\_cheapest()

for node in range(self.V):

if self.cheapest[node] != 1:

u, v, w = self.cheapest[node]

set1 = self.find(self.parents, u)

set2 = self.find(self.parents, v)

if set1 != set2:

MSTweight += w

self.union(set1, set2)

print("Towns %d-%d with distance %d are included in establishing electric grid " % (u, v, w))

numTrees = numTrees - 1

self.cheapest\_nodes = self.cheapest

self.cheapest = [-1] \* self.V

self.MST\_weight = MSTweight

return MSTweight

def find(self, parent, i):

if parent[i] == i:

return i

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

test.py

import boruvkas

import networkx as nx

import random

import graph

def MST\_problem(n):

weightsvector = random.sample(range(1,100), 25)

weightindex = 0

G = nx.Graph()

G.edges\_with\_weight = []

for i in range(n):

for j in range(i+1, n):

weight = weightsvector[weightindex]

G.add\_edge(i, j, weight=weight)

G.edges\_with\_weight.append((i,j,weight))

weightindex += 1

A = boruvkas.BoruvkasAlgorithm(G)

graph.display(G)

A.find\_MST()

graph.display(G, path=A.cheapest\_nodes)

if \_\_name\_\_ == "\_\_main\_\_":

MST\_problem(4)

Graph.py

import networkx as nx

import matplotlib.pyplot as plt

def display(G, path=None):

pos = nx.spring\_layout(G)

nx.draw(G, pos, node\_color='k')

nx.draw\_networkx\_labels(G, pos, font\_size=20)

if path is not None:

nx.draw\_networkx\_edges(G, pos, edgelist=path, width=6, edge\_color='r')

else:

nx.draw\_networkx\_edges(G, pos, width=1)

nx.draw\_networkx\_edge\_labels(G, pos, font\_size=10)

plt.axis('off')

plt.plot(weight=True)

plt.show()

main.py

# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.

# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.

def print\_hi(name):

# Use a breakpoint in the code line below to debug your script.

print(f'Hi, {name}') # Press Ctrl+F8 to toggle the breakpoint.

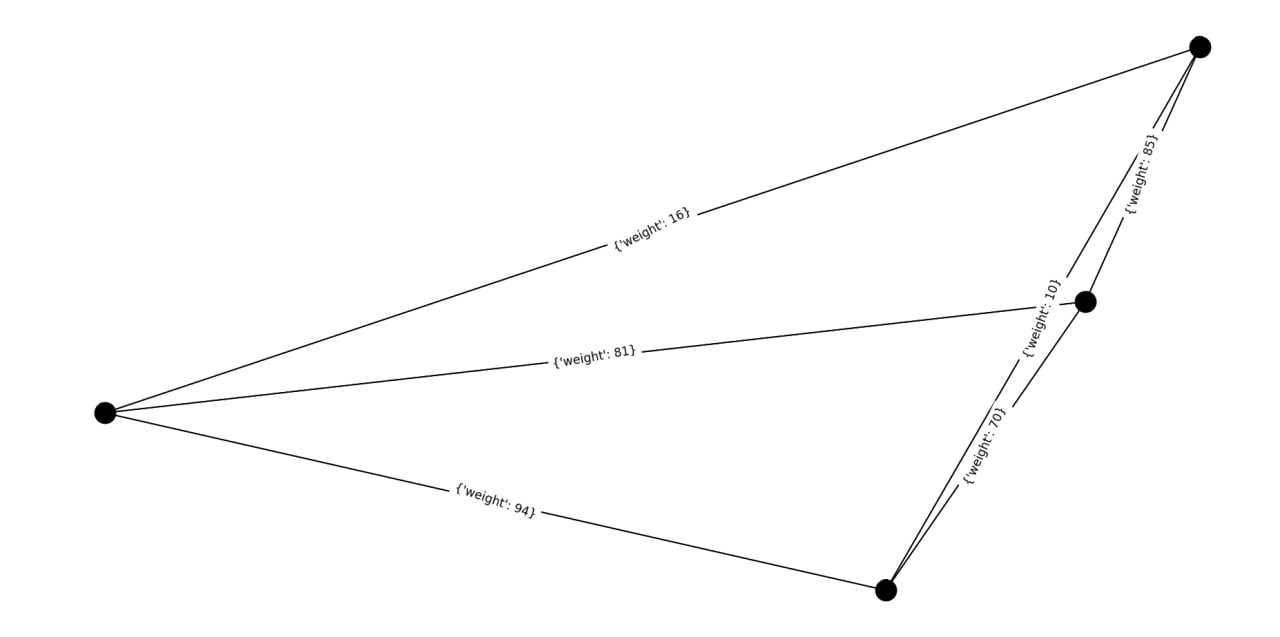
# Press the green button in the gutter to run the script.

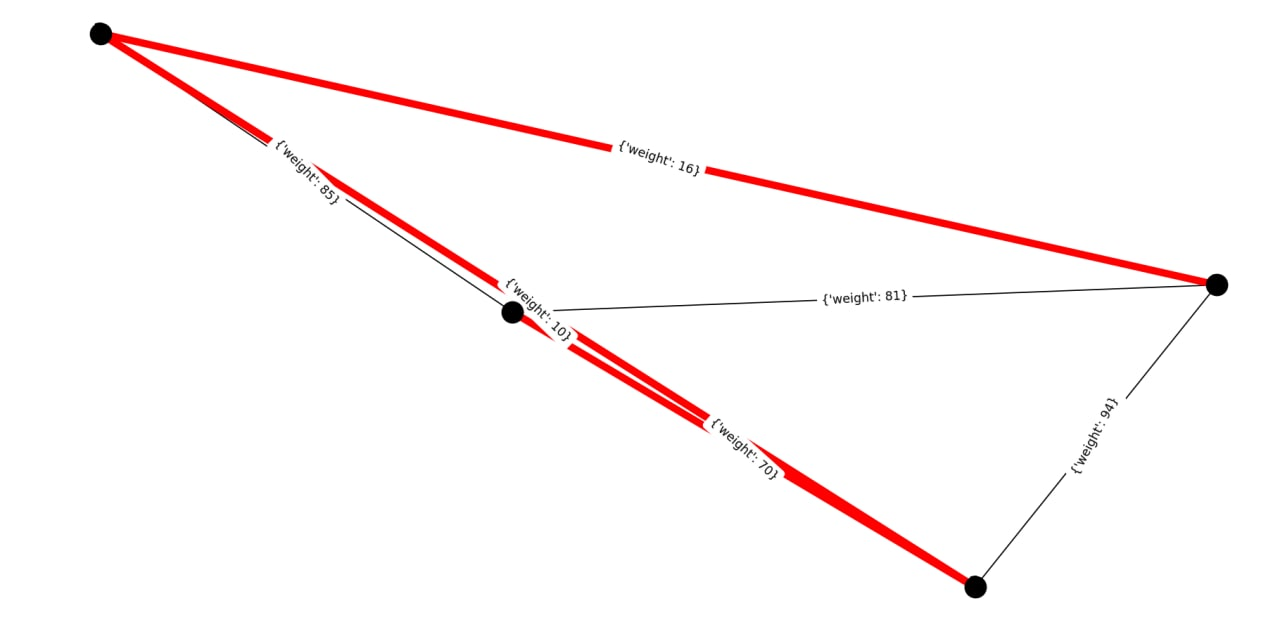
if \_\_name\_\_ == '\_\_main\_\_':

print\_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/

**RESULTS**





**CONCLUSION**

In the future, we will explore and test our developed algorithm “BORUVKA’S” in various domains. It is an important combinatorial optimization problem which is improved in recent times. The availability of reliable software, extremely fast and inexpensive hardware and high –level languages that make the modeling of complex problems faster have led to much greater demand for optimization tools. Keeping the above points of view our future work will more emphasize much larger problems on personal computers, much of the necessary data is routinely collected and tools exist to speed up both the modeling and the post optimality analysis.The front end can be also added to this.