Prims Algo:

import sys

# Graph class for Prim's Algorithm

class Graph:

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

self.V = vertices

# Initializing adjacency matrix with 0s

self.graph = [[0 for \_ in range(vertices)] for \_ in range(vertices)]

def printMST(self, parent):

print("\nMinimum Spanning Tree using Prim's Algorithm:")

print("Edge \tWeight")

for i in range(1, self.V):

print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

def minKey(self, key, mstSet):

# Find the vertex with minimum key value

min\_val = sys.maxsize

min\_index = -1

for v in range(self.V):

if key[v] < min\_val and not mstSet[v]:

min\_val = key[v]

min\_index = v

return min\_index

def primMST(self):

key = [sys.maxsize] \* self.V # Values to pick minimum weight edge

parent = [None] \* self.V # Array to store MST

mstSet = [False] \* self.V # Included vertices in MST

key[0] = 0 # Start from vertex 0

parent[0] = -1 # Root node of MST

for \_ in range(self.V):

u = self.minKey(key, mstSet)

mstSet[u] = True

for v in range(self.V):

if self.graph[u][v] > 0 and not mstSet[v] and key[v] > self.graph[u][v]:

key[v] = self.graph[u][v]

parent[v] = u

self.printMST(parent)

print("\nHeuristic Function Used in Prim's Algorithm:")

print("f(v) = key[v] if vertex v is not in MST")

# Main code for Prim's Algorithm

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

print("Name: Prachi Karande")

print("Roll no.: TACO22134")

vertices = int(input("Enter the number of vertices: "))

g = Graph(vertices)

print("Enter the graph as adjacency matrix (one row per line):")

for i in range(vertices):

row = list(map(int, input().split()))

g.graph[i] = row

source\_vertex = int(input("Enter the source vertex (Prim's always starts from 0): "))

g.primMST()

Kruskall Algo:

# Disjoint Set for Kruskal's Algorithm

class DisjointSet:

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

self.parent = list(range(n))

self.rank = [0] \* n

def find(self, x):

# Heuristic: Path Compression

if self.parent[x] != x:

self.parent[x] = self.find(self.parent[x])

return self.parent[x]

def union(self, x, y):

# Heuristic: Union by Rank

rootX = self.find(x)

rootY = self.find(y)

if rootX != rootY:

if self.rank[rootX] > self.rank[rootY]:

self.parent[rootY] = rootX

elif self.rank[rootX] < self.rank[rootY]:

self.parent[rootX] = rootY

else:

self.parent[rootY] = rootX

self.rank[rootX] += 1

def kruskal(n, edges):

ds = DisjointSet(n)

mst = []

total\_weight = 0

# Sort edges by weight

edges.sort(key=lambda x: x[2])

for u, v, weight in edges:

if ds.find(u) != ds.find(v):

ds.union(u, v)

mst.append((u, v, weight))

total\_weight += weight

return mst, total\_weight

# Main code for Kruskal's Algorithm

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

print("Name: Prachi Karande")

print("Roll no.: TACO22134")

n = int(input("Enter the number of vertices: "))

e = int(input("Enter the number of edges: "))

edges = []

for \_ in range(e):

u, v, weight = map(int, input("Enter edge (u v weight): ").split())

edges.append((u, v, weight))

mst, total\_weight = kruskal(n, edges)

print("\nMinimum Spanning Tree using Kruskal's Algorithm:")

for u, v, weight in mst:

print(f"{u} - {v} \t{weight}")

print(f"\nTotal cost of the Minimum Spanning Tree: {total\_weight}")

print("\nExpression for Edge Selection in Kruskal's Algorithm:")

print("f(u, v) = w(u, v) if find(u) != find(v)")