68.

Input: [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]

Output:19

krushkals

class DisjointSet:

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

self.parent = list(range(n))

self.rank = [0] \* n

def find(self, u):

if self.parent[u] != u:

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

return self.parent[u]

def union(self, u, v):

root\_u = self.find(u)

root\_v = self.find(v)

if root\_u != root\_v:

if self.rank[root\_u] > self.rank[root\_v]:

self.parent[root\_v] = root\_u

elif self.rank[root\_u] < self.rank[root\_v]:

self.parent[root\_u] = root\_v

else:

self.parent[root\_v] = root\_u

self.rank[root\_u] += 1

def kruskal(n, edges):

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

ds = DisjointSet(n)

mst = []

total\_cost = 0

for u, v, weight in edges:

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

ds.union(u, v)

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

total\_cost += weight

return mst, total\_cost

edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]

n = 4

mst, cost = kruskal(n, edges)

print("Edges in the Minimum Spanning Tree:", mst)

print("Total cost of the Minimum Spanning Tree:", cost)

69.boruvkas algorithm.

Input:[( 0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]

Output:

Edge 0 - 3 with weight 5 included in MST

Edge 0 - 1 with weight 10 included in MST

Edge 2 - 3 with weight 4 included in MST

Total weight of MST is 19

class Graph:

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

self.V = vertices

self.graph = []

def add\_edge(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 boruvka\_mst(self):

parent = []

rank = []

cheapest = []

num\_trees = self.V

mst\_weight = 0

for node in range(self.V):

parent.append(node)

rank.append(0)

cheapest = [-1] \* len(self.graph)

while num\_trees > 1:

for i in range(len(self.graph)):

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

set\_u = self.find(parent, u)

set\_v = self.find(parent, v)

if set\_u != set\_v:

if cheapest[set\_u] == -1 or cheapest[set\_u][2] > w:

cheapest[set\_u] = [u, v, w]

if cheapest[set\_v] == -1 or cheapest[set\_v][2] > w:

cheapest[set\_v] = [u, v, w]

for node in range(self.V):

if cheapest[node] != -1:

u, v, w = cheapest[node]

set\_u = self.find(parent, u)

set\_v = self.find(parent, v)

if set\_u != set\_v:

mst\_weight += w

self.union(parent, rank, set\_u, set\_v)

print(f"Edge {u} - {v} with weight {w} included in MST")

cheapest = [-1] \* len(self.graph)

num\_trees -= 1

print(f"Total weight of MST is {mst\_weight}")

g = Graph(4)

g.add\_edge(0, 1, 10)

g.add\_edge(0, 2, 6)

g.add\_edge(0, 3, 5)

g.add\_edge(1, 3, 15)

g.add\_edge(2, 3, 4)

g.boruvka\_mst()