# function to find the node with the lowest cost def find\_min\_cost\_node(costs, visited):

min\_cost = float('inf') min\_node = None

for node, cost in costs.items():

if cost < min\_cost and node not in visited: min\_cost = cost

min\_node = node return min\_node

# function to implement Prim's algorithm def prims\_algorithm(graph, start):

visited = set()

costs = {node: float('inf') for node in graph} costs[start] = 0

mst\_cost = 0

while len(visited) < len(graph):

# find the node with the lowest cost

node = find\_min\_cost\_node(costs, visited)

if node is None:

# there are disconnected components in the graph break

# mark the node as visited visited.add(node) mst\_cost += costs[node]

# update the costs of the node's neighbors for neighbor, weight in graph[node]:

if neighbor not in visited and weight < costs[neighbor]: costs[neighbor] = weight

return mst\_cost # main function

if name == ' main ':

graph = {}

n = int(input("Enter the number of nodes: ")) # initialize the graph

for i in range(n): graph[i] = []

# add edges to the graph

m = int(input("Enter the number of edges: ")) for i in range(m):

u, v, weight = map(int, input("Enter an edge and its weight: ").split()) graph[u].append((v, weight))

graph[v].append((u, weight))

# run Prim's algorithm starting from node 0 mst\_cost = prims\_algorithm(graph, 0)

print("The cost of the minimum spanning tree is:", mst\_cost)