Name: Faraz Pathan

Rollno: COTB70

**PROGRAM:**

class Graph:

    def \_\_init\_\_(self):

        self.graph = {}

    def add\_edge(self, u, v, w):

        if u in self.graph:

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

        else:

            self.graph[u] = [(v, w)]

        if v in self.graph:

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

        else:

            self.graph[v] = [(u, w)]

    def prim\_mst(self):

        mst = []

        visited = set()

        start = next(iter(self.graph))  # Start from any vertex

        visited.add(start)

        while len(visited) < len(self.graph):

            min\_edge = None

            min\_weight = float('inf')

            for vertex in visited:

                for neighbor, weight in self.graph[vertex]:

                    if neighbor not in visited and weight < min\_weight:

                        min\_weight = weight

                        min\_edge = (vertex, neighbor, weight)

            if min\_edge:

                mst.append(min\_edge)

                visited.add(min\_edge[1])

        return mst

# Example usage:

g = Graph()

g.add\_edge('A', 'B', 2)

g.add\_edge('A', 'C', 3)

g.add\_edge('B', 'C', 1)

g.add\_edge('B', 'D', 1)

g.add\_edge('C', 'D', 2)

mst = g.prim\_mst()

print("Prim's Minimum Spanning Tree:")

for edge in mst:

    print(edge)

**OUTPUT:**

Prim's Minimum Spanning Tree:

('A', 'B', 2)

('B', 'C', 1)

('B', 'D', 1)