In [17]:

```python
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.edges = []

    def add_edge(self, u, v, weight):
        self.edges.append((u, v, weight))

    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 kruskal_mst(self):
        result = []

        # Sort all the edges in non-decreasing order of their weight
        self.edges = sorted(self.edges, key=lambda edge: edge[2])

        parent = [i for i in range(self.V)]
        rank = [0] * self.V

        i = 0  # Index used to pick next edge
        e = 0  # Index used to count edges

        while e < self.V - 1:
            u, v, weight = self.edges[i]
            i += 1
            x = self.find(parent, u)
            y = self.find(parent, v)

            if x != y:
                e += 1
                result.append((u, v, weight))
                self.union(parent, rank, x, y)

        return result

# Example usage:
g = Graph(5)
g.add_edge(0, 1, 2)
g.add_edge(0, 3, 6)
g.add_edge(1, 2, 3)
g.add_edge(1, 3, 8)
g.add_edge(1, 4, 5)
```

```python
g.add_edge(2, 4, 7)
g.add_edge(3, 4, 9)

mst = g.kruskal_mst()
print("Edges in the Minimum Spanning Tree:")
for u, v, weight in mst:
    print(f"{u} - {v} : {weight}")
```

```
Edges in the Minimum Spanning Tree:
0 - 1 : 2
1 - 2 : 3
1 - 4 : 5
0 - 3 : 6
```

In [ ]: