In [1]:

```
class UnionFind:
    def __init__(self, size):
        self.parent = list(range(size))
        self.rank = [0] * size

    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

class Graph:
    def __init__(self, edges):
        self.edges = edges

    def kruskal(self):

        self.edges.sort(key=lambda x: x[2])
        uf = UnionFind(len(self.edges))
        mst_edges = []
        total_weight = 0

        for u, v, weight in self.edges:
            if uf.find(u) != uf.find(v):
                uf.union(u, v)
                mst_edges.append((u, v, weight))
                total_weight += weight

        return mst_edges, total_weight


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


graph = Graph(edges)


mst_edges, total_weight = graph.kruskal()
```

```python
print("Edges in the Minimum Spanning Tree (MST):")
for u, v, weight in mst_edges:
    print(f"Edge: {u} - {v}, Weight: {weight}")

print(f"Total weight of the MST: {total_weight}")
```

```
Edges in the Minimum Spanning Tree (MST):
Edge: 2 - 3, Weight: 4
Edge: 0 - 3, Weight: 5
Edge: 0 - 1, Weight: 10
Total weight of the MST: 19
```

```
In [ ]:
```