In [4]:

```
from collections import defaultdict
class Graph: #Class to represent a graph
    def __init__(self, vertices): #Constructor to initialise variables
        self.V = vertices
        self.graph = []

    def addEdge(self, u, v, w): #Function to add an edge to graph
        self.graph.append([u, v, w])

    def find(self, parent, i): #Function to find set of an element
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y): #Function for union of two sets
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)

        # Attach smaller rank tree under root of high rank tree (Union by Rank)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else: # If ranks are same, then make one as root and increment its rank by
            parent[yroot] = xroot
            rank[xroot] += 1

    def KruskalMST(self): #Function to construct MST using Kruskal's algorithm
        MST = []  # MST Variable
        i = 0 #index variable, used for sorted edges
        e = 0 #index variable, used for result[]
        self.graph = sorted(self.graph,key=lambda item: item[2]) #Sorting all the e
        parent = []
        rank = []
        # Create V subsets with single elements
        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while e < self.V - 1:
            #Pick the smallest edge and increment the index for next iteration
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)

            if x != y: #If including this edge doesn't cause cycle, include it in r
                e = e + 1
                MST.append([u, v, w])
                self.union(parent, rank, x, y)
            # Else discard the edge

        minimumCost = 0
        print ("Edges in the constructed MST")
        for u, v, weight in MST:
            minimumCost += weight
            print("%d -- %d == %d" % (u, v, weight))
        print("Minimum Spanning Tree" , minimumCost)

#Input
g = Graph(4)
g.addEdge(0, 1, 10)
```

```
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)
g.KruskalMST()
```

```
Edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19
```