Python Code :-

| from collections import defaultdict   # Class to represent a graph **class** **Graph**:    **def** **\_\_init\_\_**(**self**, vertices):  **self**.V = vertices # No. of vertices  **self**.graph = [] # default dictionary  # to store graph    # function to add an edge to graph  **def** **addEdge**(**self**, u, v, w):  **self**.graph.append([u, v, w])    # A utility function to find set of an element i  # (uses path compression technique)  **def** **find**(**self**, parent, i):  **if** parent[i] == i:  **return** i  **return** **self**.find(parent, parent[i])    # A function that does union of two sets of x and y  # (uses union by rank)  **def** **union**(**self**, parent, rank, x, y):  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    # If ranks are same, then make one as root  # and increment its rank by one  else:  parent[yroot] = xroot  rank[xroot] += 1    # The main function to construct MST using Kruskal's  # algorithm  **def** **KruskalMST**(**self**):    result = [] # This will store the resultant MST    # An index variable, used for sorted edges  i = 0    # An index variable, used for result[]  e = 0    # Step 1: Sort all the edges in  # non-decreasing order of their  # weight. If we are not allowed to change the  # given graph, we can create a copy of graph  **self**.graph = sorted(**self**.graph,  key=lambda item: item[2])    parent = []  rank = []    # Create V subsets with single elements  **for** node **in** range(**self**.V):  parent.append(node)  rank.append(0)    # Number of edges to be taken is equal to V-1  **while** e < **self**.V - 1:    # Step 2: 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 including this edge doesn't  # cause cycle, include it in result  # and increment the indexof result  # for next edge  **if** x != y:  e = e + 1  result.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** result:  minimumCost += weight  print("%d -- %d == %d" % (u, v, weight))  print("Minimum Spanning Tree" , minimumCost)   # Driver code 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)   # Function call g.KruskalMST() |
| --- |

Output:-

Following are the edges in the constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning Tree: 19