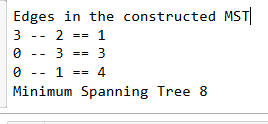
PRACTICAL 2 - MST

A] kruskal algo

| #kruskal algo for mst  # Python program for Kruskal's algorithm to find  # Minimum Spanning Tree of a given connected,  # undirected and weighted graph  # Class to represent a graph  class Graph:  def \_\_init\_\_(self, vertices):  self.V = vertices  self.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  # (truly uses path compression technique)  def find(self, parent, i):  if parent[i] != i:  # Reassignment of node's parent  # to root node as  # path compression requires  parent[i] = self.find(parent, parent[i])  return 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):  # Attach smaller rank tree under root of  # high rank tree (Union by Rank)  if rank[x] < rank[y]:  parent[x] = y  elif rank[x] > rank[y]:  parent[y] = x  # If ranks are same, then make one as root  # and increment its rank by one  else:  parent[y] = x  rank[x] += 1  # The main function to construct MST  # using Kruskal's algorithm  def KruskalMST(self):  # This will store the resultant MST  result = []  # An index variable, used for sorted edges  i = 0  # An index variable, used for result[]  e = 0  # Sort all the edges in  # non-decreasing order of their  # weight  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 less than to V-1  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 including this edge doesn't  # cause cycle, then include it in result  # and increment the index of 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  if \_\_name\_\_ == '\_\_main\_\_':  g = Graph(4)  g.addEdge(0, 1, 4)  g.addEdge(0, 3, 3)  g.addEdge(1, 2, 7)  g.addEdge(3, 2, 1)  # Function call  g.KruskalMST() |
| --- |

Output:



B] prims algorithm

| # Prim's Algorithm in Python  INF = 9999999  # number of vertices in graph  N = 5  #creating graph by adjacency matrix method  G = [[0, 9, 75, 0, 0],  [9, 0, 95, 19, 42],  [75, 95, 0, 51, 65],  [0, 19, 51, 0, 31],  [0, 42, 66, 31, 0]]  selected\_node = [0, 0, 0, 0, 0]  no\_edge = 0  selected\_node[0] = True  # printing for edge and weight  print("Edge : Weight\n")  while (no\_edge < N - 1):    minimum = INF  a = 0  b = 0  for m in range(N):  if selected\_node[m]:  for n in range(N):  if ((not selected\_node[n]) and G[m][n]):  # not in selected and there is an edge  if minimum > G[m][n]:  minimum = G[m][n]  a = m  b = n  print(str(a) + "-" + str(b) + ":" + str(G[a][b]))  selected\_node[b] = True  no\_edge += 1 |
| --- |

Output:

