Program:

```
# Class to represent a graph
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

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

    # Utility function to find the set of an element (with path compression)
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    # Function to perform union of two sets (by rank)
    def union(self, parent, rank, x, y):
        root_x = self.find(parent, x)
        root_y = self.find(parent, y)
        if rank[root_x] < rank[root_y]:
            parent[root_x] = root_y
        elif rank[root_x] > rank[root_y]:
            parent[root_y] = root_x
        else:
            parent[root_y] = root_x
            rank[root_x] += 1

    # Kruskal's algorithm to find MST
    def kruskal_algorithm(self):
```

```
result = []  # This will store the resulting MST

# Step 1: Sort all the edges in ascending order of their weights
self.graph = sorted(self.graph, key=lambda item: item[2])

parent = []
rank = []

# Create V disjoint sets (each node is its own parent)
for node in range(self.V):
    parent.append(node)
    rank.append(0)

e = 0  # Initialize the number of edges in the MST
i = 0  # Index variable for sorted edges

# Step 2: Pick the smallest edge and check if it forms a cycle
while e < self.V - 1:
    u, v, w = self.graph[i]
    i += 1
    x = self.find(parent, u)
    y = self.find(parent, v)

    # If including this edge doesn't cause a cycle
    if x != y:
        e += 1
        result.append([u, v, w])
        self.union(parent, rank, x, y)

print("Edge : Weight")
for u, v, weight in result:
```

```
            print(f"{u} - {v} : {weight}")

# Input number of vertices and edges
vertices = int(input("Enter the number of vertices: "))
edges = int(input("Enter the number of edges: "))

# Create a graph and add edges
graph = Graph(vertices)
print("Enter each edge in the format: vertex1 vertex2 weight (use 0-based index)")
for _ in range(edges):
    u, v, w = map(int, input().split())
    if u >= vertices or v >= vertices:
        print(f"Error: Invalid vertex index {u} or {v}. It should be between 0 and {vertices-1}.")
    else:
        graph.add_edge(u, v, w)

# Execute Kruskal's algorithm
graph.kruskal_algorithm()
```

Output: