```python
def find(parent, i):
    # Find the root of the set containing vertex i
    if parent[i] == i:  # If the parent of vertex i is itself, i is the root of the set
        return i
    return find(parent, parent[i])  # Recursively find the root of the set containing vertex i

# ************************************************************************************************ #


def union(parent, x, y):
    # Union two sets represented by x and y
    x_root = find(parent, x)  # Find the root of the set containing vertex x
    y_root = find(parent, y)  # Find the root of the set containing vertex y
    parent[x_root] = y_root  # Set the root of set containing x to be the root of set containing y

# ********************************************************************************************* #
def kruskal(graph):
    n = len(graph)
    result = []  # Initialize the result list to store edges of the MST

    # Initialize parent array for Union-Find operations of size n
    # intially parent of each vertex will be vertex itself (i for i in range(n))
    parent = [i for i in range(n)]

    # Sort edges of the graph by weight in ascending order
    # for i in range(n): This outer loop iterates over each row index i in the graph matrix.
    # for j in range(i+1, n): This inner loop iterates over each column index j in the graph matrix, starting from i+1.
    # This ensures that each edge is considered only once and prevents considering
    # edges in both directions (e.g., both (i, j) and (j, i)).
    # sorted() function sorts the list of tuples (edges) based on the first element of each tuple, which is the weight
    edges = sorted((graph[i][j], i, j) for i in range(n) for j in range(i+1, n) if graph[i][j] != 0)

    for weight, u, v in edges:
        # Iterate over sorted edges
        u_root = find(parent, u)  # Find the root of the set containing vertex u
        v_root = find(parent, v)  # Find the root of the set containing vertex v
        if u_root != v_root:  # If vertices u and v are not in the same set (no cycle is formed)
            result.append((u, v, weight))  # Add the edge (u, v) to the MST
            union(parent, u_root, v_root)  # Union the sets containing u and v


    return result

# ************************************************************************************************ #
graph = [
    [0, 4, 6, 0, 0, 0],
    [4, 0, 6, 3, 4, 0],
    [6, 6, 0, 1, 8, 0],
    [0, 3, 1, 0, 2, 3],
    [0, 4, 8, 2, 0, 7],
    [0, 0, 0, 3, 7, 0]
]
mst = kruskal(graph)  # Find the minimum spanning tree (MST) of the given graph
for u, v, weight in mst:
    print(f"Edge: {u} - {v}, Weight: {weight}")  # Print edges of the MST
```