```
INF = 9999999
# number of vertices in graph
V = 5
# create a 2d array of size 5x5print
# for adjacency matrix to represent graph
G = [[0, 9, 75, 0, 0],
     [9, 0, 95, 19, 42],
     [75, 95, 0, 51, 66],
     [0, 19, 51, 0, 31],
     [0, 42, 66, 31, 0]]
# create a array to track selected vertex
# selected will become true otherwise false
selected = [0, 0, 0, 0, 0]
# set number of edge to 0
no_edge = 0
# the number of egde in minimum spanning tree will be
# always less than(V - 1), where V is number of vertices in
# graph
# choose 0th vertex and make it true
selected[0] = True
# print for edge and weight
print("Edge : Weight\n")
while (no_edge < V - 1):
    # For every vertex in the set S, find the all adjacent vertices
    #, calculate the distance from the vertex selected at step 1.
    # if the vertex is already in the set S, discard it otherwise
    # choose another vertex nearest to selected vertex  at step 1.
    minimum = INF
    x = 0
    y = 0
    for i in range(V):
        if selected[i]:
            for j in range(V):
                if ((not selected[j]) and G[i][j]):
                    # not in selected and there is an edge
                    if minimum > G[i][j]:
                        minimum = G[i][j]
                        x = i
                        y = j
    print(str(x) + "-" + str(y) + ":" + str(G[x][y]))
    selected[y] = True
    no_edge += 1

# The given code implements Prim's algorithm, not Kruskal's algorithm. Prim's algorithm is a greedy algorithm that finds a minimum spanni

# 1. Initialize some variables:
#    - `INF` is set to a very large value (9999999) to represent infinity.
#    - `V` is the number of vertices in the graph (5 in this case).
#    - `G` is a 2D array representing the adjacency matrix of the graph. Each value `G[i][j]` represents the weight of the edge between v
#    - `selected` is an array to keep track of the selected vertices in the minimum spanning tree. Initially, all elements are set to 0 (
#    - `no_edge` is a counter to keep track of the number of edges in the minimum spanning tree. It is initialized to 0.

# 2. Choose the first vertex (vertex 0) as the starting vertex for the minimum spanning tree and mark it as selected by setting `selected

# 3. Enter a loop that continues until `no_edge` reaches `V - 1` (the number of vertices minus 1), which is the maximum number of edges i

# 4. Inside the loop, find the minimum weight edge that connects a vertex in the selected set `S` to a vertex not yet selected.
#    - Initialize `minimum` to `INF` and `x` and `y` to 0.
#    - Iterate over all vertices in the graph.
#      - If the current vertex `i` is selected (`selected[i]` is True), iterate over all vertices `j` in the graph.
#        - If `j` is not selected (`not selected[j]`) and there is an edge between `i` and `j` (`G[i][j] != 0`), then check if the weight
#          - If it is, update `minimum` with the new minimum weight, and update `x` and `y` with the indices of the vertices that form th

# 5. Print the selected edge and its weight.
#    - Print the values of `x`, `y`, and `G[x][y]`, which represent the indices of the vertices and the weight of the edge between them.

# 6. Mark the vertex `y` as selected by setting `selected[y]` to True.

# 7. Increment the `no_edge` counter by 1.

# This process repeats until `no_edge` reaches `V - 1`, meaning that a minimum spanning tree with `V - 1` edges has been found.
```

```
Edge : Weight

0-1:9
1-3:19
3-4:31
3-2:51
```