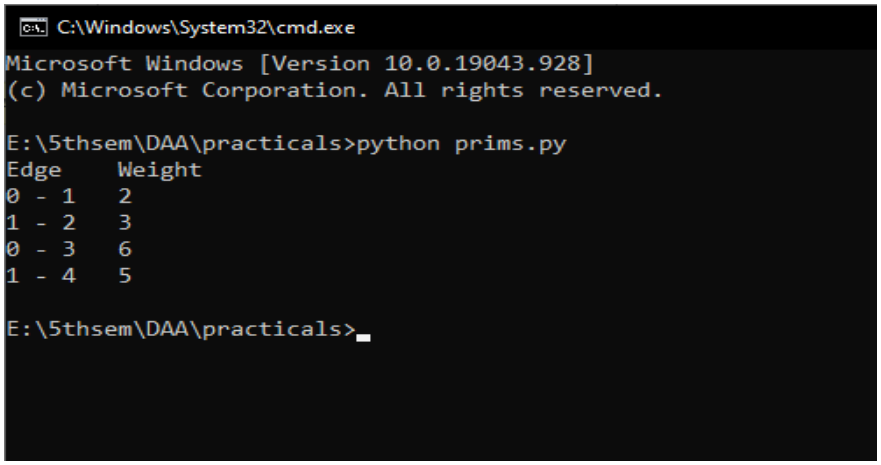
```
import sys

def prims_algorithm(graph):
    num_vertices = len(graph)
    selected = [False] * num_vertices
    key = [sys.maxsize] * num_vertices
    parent = [-1] * num_vertices

    key[0] = 0
    parent[0] = -1

    for _ in range(num_vertices):
        min_key = sys.maxsize
        u = 0

        # Find the vertex with the minimum key value
        for v in range(num_vertices):
            if not selected[v] and key[v] < min_key:
                min_key = key[v]
                u = v

        selected[u] = True

        # Update key values of adjacent vertices
        for v in range(num_vertices):
            if graph[u][v] and not selected[v] and graph[u][v] < key[v]:
                key[v] = graph[u][v]
                parent[v] = u

    return parent, key

def print_mst(parent, key):
    print("Edge \tWeight")
    for i in range(1, len(parent)):
        print(f"{parent[i]} - {i} \t{key[i]}")

# Example input graph as an adjacency matrix
graph = [
    [0, 2, 0, 6, 0],
```

```
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 0]
]

parent, key = prims_algorithm(graph)
print_mst(parent, key)
```



```
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n

    def find(self, u):
        if self.parent[u] != u:
            self.parent[u] = self.find(self.parent[u])  # Path compression
        return self.parent[u]

    def union(self, u, v):
        root_u = self.find(u)
        root_v = self.find(v)
        if root_u != root_v:
            if self.rank[root_u] > self.rank[root_v]:
                self.parent[root_v] = root_u
            elif self.rank[root_u] < self.rank[root_v]:
                self.parent[root_u] = root_v
            else:
                self.parent[root_v] = root_u
                self.rank[root_u] += 1

def kruskal_algorithm(vertices, edges):
    mst = []
    edges.sort(key=lambda x: x[2])  # Sort edges based on weight
    ds = DisjointSet(vertices)

    for u, v, weight in edges:
        if ds.find(u) != ds.find(v):
            ds.union(u, v)
            mst.append((u, v, weight))

    return mst

def print_mst(mst):
    print("Edge \tWeight")
    for u, v, weight in mst:
        print(f"{u} - {v} \t{weight}")
```

```
# Example input edges
edges = [
   (0, 1, 2),
   (1, 2, 3),
   (0, 3, 6),
   (1, 4, 5),
   (2, 4, 7),
   (3, 4, 9)
]

vertices = 5
mst = kruskal_algorithm(vertices, edges)
print_mst(mst)
```