Name – Aryan Sharma

BT21GCS161

DAA Assignment 11

Question – Prism's Algorithm

Code –

01.py > ...

```python
import heapq

def prim(graph):

    # Start with an arbitrary vertex
    start_vertex = list(graph.keys())[0]

    mst = {start_vertex: []}
    visited = set([start_vertex])

    edges = [(weight, start_vertex, neighbor) for neighbor, weight in graph[start_vertex]]
    heapq.heapify(edges)

    # Loop until all vertices have been visited
    total_weight = 0
    while edges:
        weight, vertex1, vertex2 = heapq.heappop(edges)

        if vertex2 in visited:
            continue

        if vertex1 not in mst:
            mst[vertex1] = [(vertex2, weight)]
        else:
            mst[vertex1].append((vertex2, weight))

        total_weight += weight
        visited.add(vertex2)
        for neighbor, weight in graph[vertex2]:
            if neighbor not in visited:
```

```python
        if vertex1 not in mst:
            mst[vertex1] = [(vertex2, weight)]
        else:
            mst[vertex1].append((vertex2, weight))

        total_weight += weight
        visited.add(vertex2)
        for neighbor, weight in graph[vertex2]:
            if neighbor not in visited:
                heapq.heappush(edges, (weight, vertex2, neighbor))

    return mst, total_weight
graph = {
    'A': [('B', 5), ('D', 4)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 6)],
    'D': [('B', 2), ('A', 4)]
}

mst, total_weight = prim(graph)
print("Minimum Spanning Tree:", mst)
print("Total Weight:", total_weight)
```

Output –

```
PS C:\Users\aryan\OneDrive - st.niituniversity.in\DAA Assignment\Assignmet -11> & C:/U
.in/DAA Assignment/Assignmet -11/01.py"
Minimum Spanning Tree: {'A': [('D', 4)], 'D': [('B', 2)], 'B': [('C', 3)]}
Total Weight: 9
```

Analysis-

Implementation assumes that the graph is represented using an adjacency list, where each vertex is associated with a list of its neighbours and the weights of the edges that connect it to its neighbours. If the graph is represented using an adjacency matrix

Time and Space complexity of the algorithm would be different, since accessing an element in an adjacency matrix takes constant time, whereas accessing a neighbour in an adjacency list takes time proportional to the degree of the vertex.

Time Complexity – O(E log V)

Space Complexity - O(E+V)