MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 7

Assignment Title: Represent a given graph using an adjacency list or array and generate a minimum spanning tree using Kruskal's or Prim’s algorithm.

Aim: Implement Graph and Kruskal's or Prim’s algorithm to generate a minimum spanning tree.

Prerequisite:

1. Basic concepts of Graph.
2. Kruskal's or Prim’s algorithm of Graph to generate a minimum spanning tree

Objectives:

Implement a Program in python for the following operations on Graph:

1. Create a Graph of N nodes using Adjacency Matrix/adjacency list.
2. Recognize and define the basic attributes of a Graph.
3. Generate a minimum spanning tree of Graph using Kruskal's or Prim’s algorithm.

Outcomes:

Upon Completion of the assignment the students will be able to

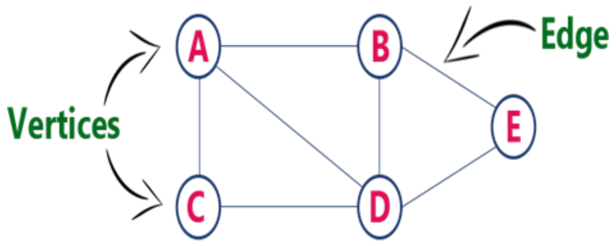
1. Create and implement Graph using adjacency list/adjacency matrix.
2. Understand and analyse Kruskal's or Prim’s algorithm.

Theory:

Graph: Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices.

Generally, a graph G is represented as G = (V , E), where V is set of vertices and E is set of edges.

Example:



The figure shows is a graph with 5 vertices and 6 edges.

This graph G can be defined as G = (V , E)

Where V = {A,B,C,D,E} and

E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.

Storage representation:

1. **Adjacency Matrix**
2. **Adjacency list**

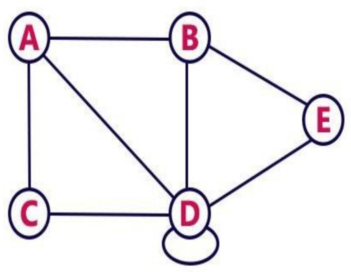
1.Adjacency Matrix

Let G = (V,E) be a graph with ‘v’ vertices v >= 1.

The adjacency matrix of graph G is a 2-dimensional n x n array say A[n:n]., with the property that,

A(i, j) = 1 iff the edge (Vi, Vj) is in E(G) for undirected graph or the edge <Vi, Vj> is in E(G) for directed graph and

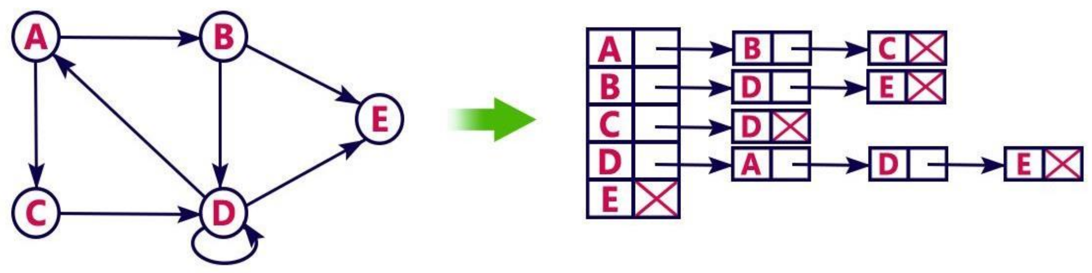
A(i, j) = 0 if there is no such edge in graph G from vertex Vi to Vj.



$$\begin{array}{c|ccccc|} & A & B & C & D & E \\ A & 0 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 1 & 1 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

2.Adjacency list

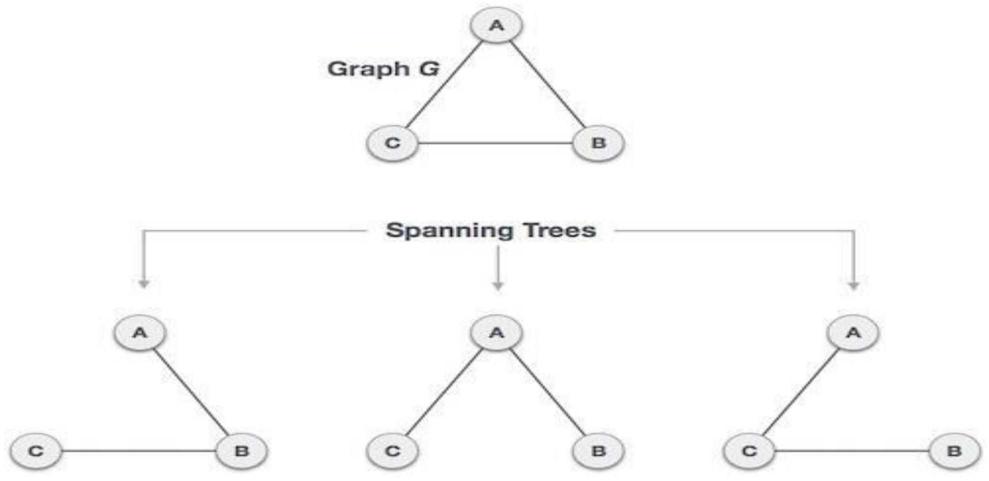
In this representation the ‘n’ rows of adjacency matrix are represented as ‘n’ linked lists. There is one list for each vertex in a graph. The nodes in list i represent the vertices that are adjacent to vertex i.



Spanning tree:

A spanning tree is defined as a subset of a connected undirected graph that has all the vertices covered with the minimum number of edges possible.

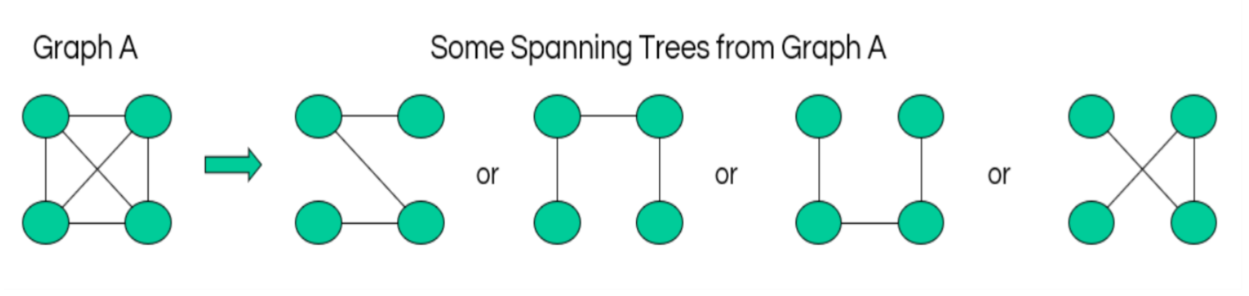
It includes all the vertices of graph in the subgraph and the least number of edges that can connect every vertex without forming a loop or cycle.



Properties of Spanning tree:

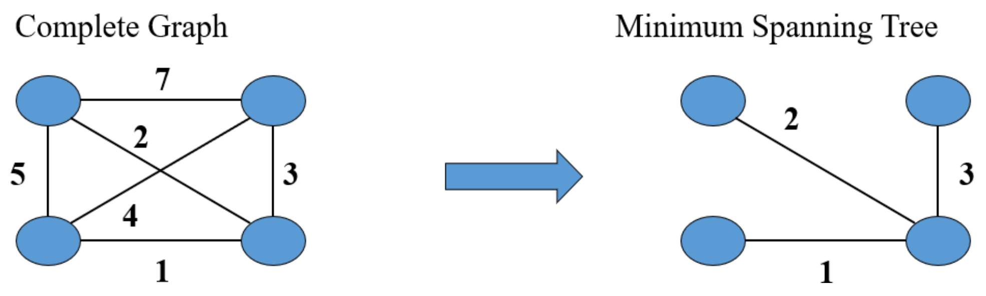
- A connected graph can have multiple spanning trees. A graph with n vertices can have an n^(n-2) number of spanning trees.
- Spanning trees does not have any loop or cycle.
- Spanning trees have n vertices and n-1 number of edges.

- All spanning tree must have an equal number of vertices as of the given graph.
- Removing a single edge from the spanning tree will make the graph disconnected as the spanning tree is minimal connected.
- Adding any edge can create a loop or cycle in the spanning tree.
- Spanning trees can be formed on connected graphs only, disconnected graphs cannot form spanning trees.
- The edges of the spanning tree may or may not have weight with them, it depends that the edges of the actual graph have weight or not.



Minimum Spanning Tree:

The Minimum Spanning Tree for a given graph is the Spanning Tree of minimum cost for that graph. To find Minimum Spanning Tree for a given graph, we need to consider smallest cost edges from graph until we get spanning tree.



Kruskal's algorithm:

In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached.

The steps to implement Kruskal's algorithm are listed as follows -

1. First, sort all the edges from low weight to high.

2. Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
3. Continue to add the edges until we reach all vertices, and a minimum spanning tree is created.

Prim's Algorithm:

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

Prim's algorithm is a greedy algorithm that starts from one vertex and continue to add the edges with the smallest weight until the goal is reached.

The steps to implement the prim's algorithm are given as follows -

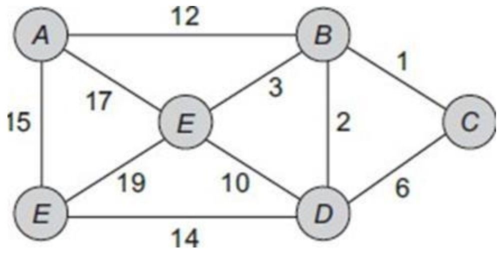
1. First, we have to initialize an MST with the randomly chosen vertex.
2. Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.
3. Repeat step 2 until the minimum spanning tree is formed.

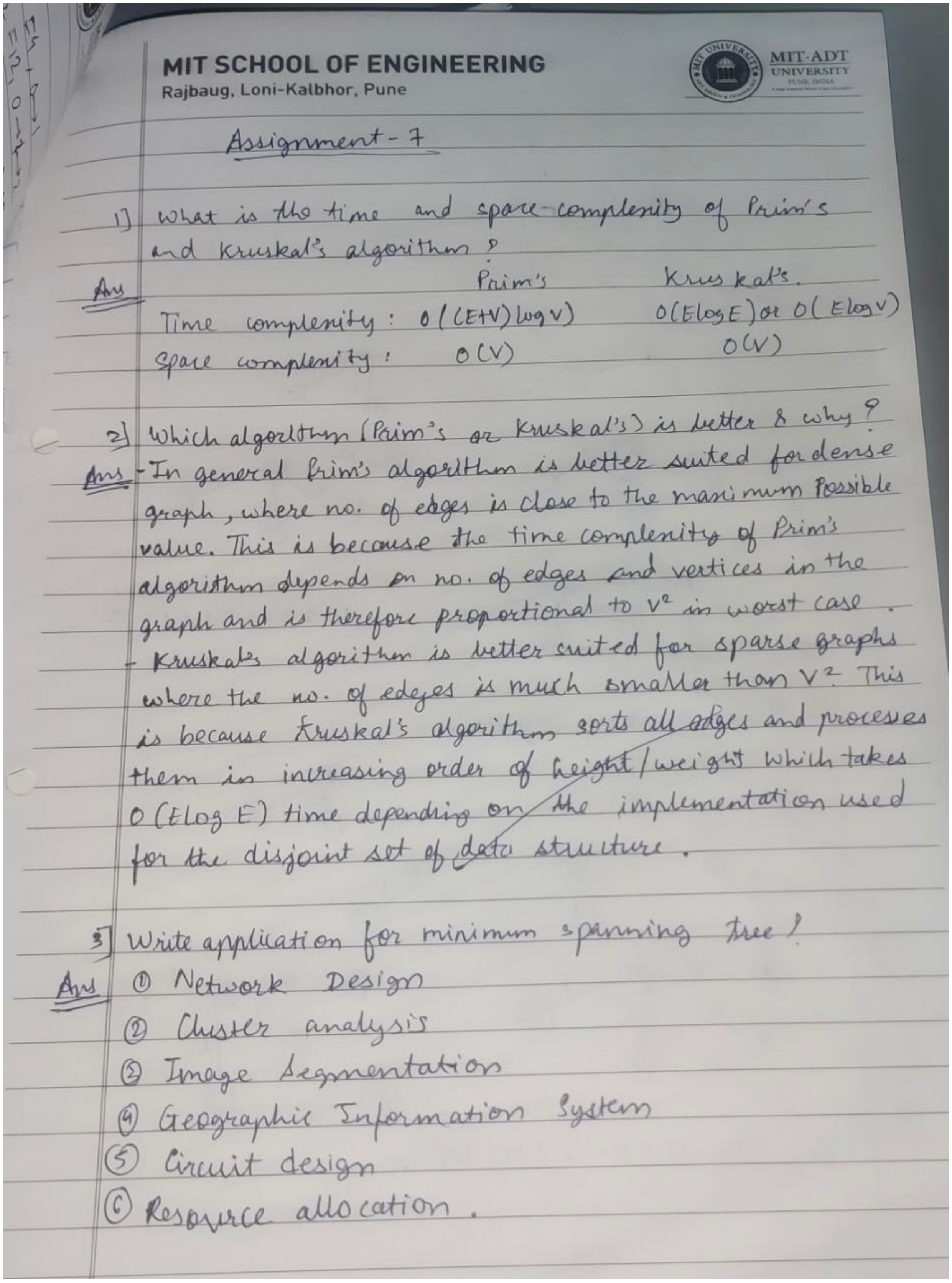
Conclusion:

Prim’s /Kruskal’s algorithm has implemented and executed successfully.

Frequently Asked Questions:

1. What is the time and space complexity of Prim’s and Kruskal’s algorithm?
2. Which algorithm (Prim’s and Kruskal’s) is better and why?
3. Write applications of Minimum Spanning Tree.
4. Construct a minimum spanning tree (step-by step) from the following graph using Prim’s and Kruskal’salgorithm.





MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY PUNE, INDIA

Assignment - 7

1] What is the time and space-complexity of Prim's and Kruskal's algorithm?

Ans

	Prim's	Kruskal's.
Time complexity :	$O((E+V)\log V)$	$O(E\log E)$ or $O(E\log V)$
Space complexity :	$O(V)$	$O(V)$

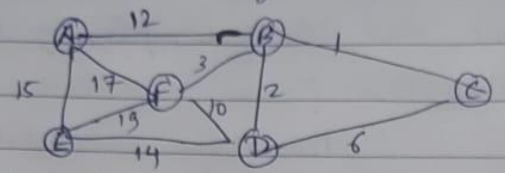
2] Which algorithm (Prim's or Kruskal's) is better & why?

Ans - In general Prim's algorithm is better suited for dense graph, where no. of edges is close to the maximum possible value. This is because the time complexity of Prim's algorithm depends on no. of edges and vertices in the graph and is therefore proportional to V^2 in worst case.
- Kruskal's algorithm is better suited for sparse graphs where the no. of edges is much smaller than V^2. This is because Kruskal's algorithm sorts all edges and processes them in increasing order of height/weight which takes $O(E\log E)$ time depending on the implementation used for the disjoint set of data structure.

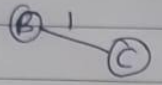
3] Write application for minimum spanning tree!

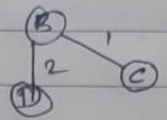
Ans
① Network Design
② Cluster analysis
③ Image segmentation
④ Geographic Information System
⑤ Circuit design
⑥ Resource allocation.

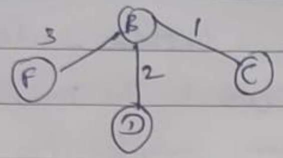
4) Construct a MST for the following graph using Prim's and kruskal's algorithm.

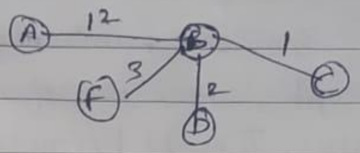


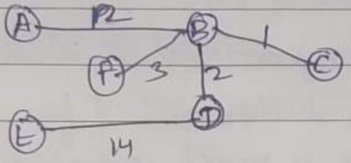
Ans ① Using Prim's algorithm:





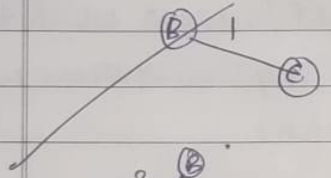


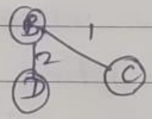


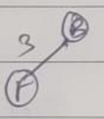


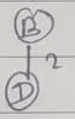
$1+2+3+12+14=32$.

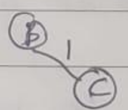
② Using Kruskal's algorithm:

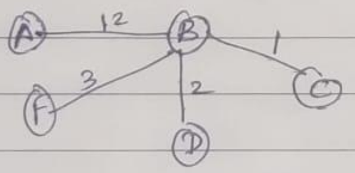


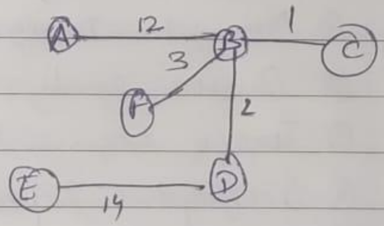










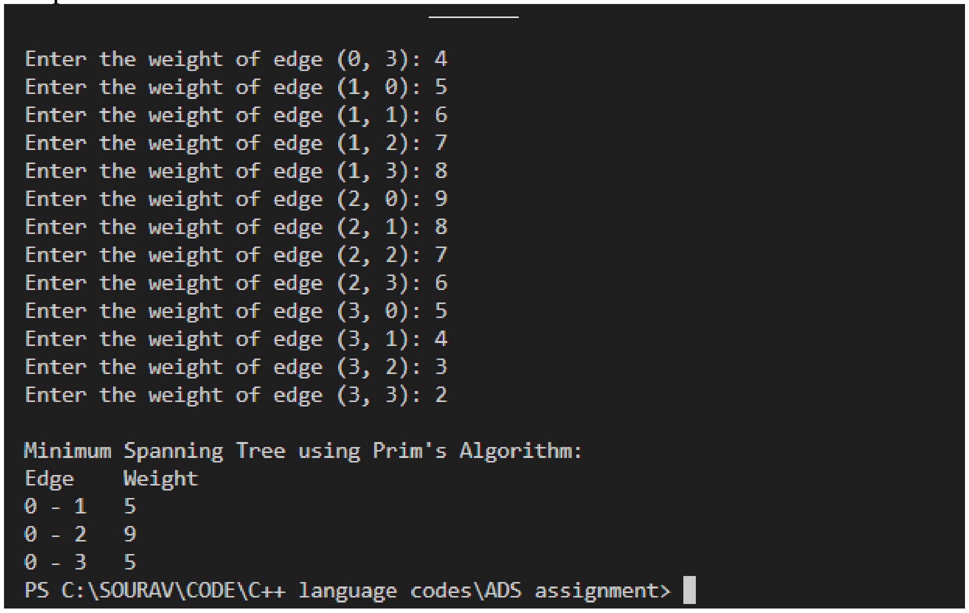


$12+1+3+2+14=32$.

Code:

```
import sys


def min_key_vertex(key, mst_set, vertices):
    min_key = sys.maxsize
    min_vertex = None
    for v in vertices:
        if key[v] < min_key and mst_set[v] == False:
            min_key = key[v]
            min_vertex = v
    return min_vertex



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



def prim_mst(graph):
    vertices = list(graph.keys())
    key = {v: sys.maxsize for v in vertices}
    parent = {v: None for v in vertices}
    key[0] = 0
    mst_set = {v: False for v in vertices}


    for _ in range(len(vertices)):
        u = min_key_vertex(key, mst_set, vertices)
        mst_set[u] = True


        for v in vertices:
            if graph[u][v] > 0 and mst_set[v] == False and key[v] >
graph[u][v]:
                key[v] = graph[u][v]
                parent[v] = u


    print_mst(parent, graph)



if __name__ == "__main__":
    graph = {}
    num_vertices = int(input("Enter the number of vertices in the graph: 
"))
    for i in range(num_vertices):
        graph[i] = {}
        for j in range(num_vertices):
            weight = int(input(f"Enter the weight of edge ({i}, {j}): "))
            graph[i][j] = weight


    print("\nMinimum Spanning Tree using Prim's Algorithm:")
    prim_mst(graph)
```

Output:



```
Enter the weight of edge (0, 3): 4
Enter the weight of edge (1, 0): 5
Enter the weight of edge (1, 1): 6
Enter the weight of edge (1, 2): 7
Enter the weight of edge (1, 3): 8
Enter the weight of edge (2, 0): 9
Enter the weight of edge (2, 1): 8
Enter the weight of edge (2, 2): 7
Enter the weight of edge (2, 3): 6
Enter the weight of edge (3, 0): 5
Enter the weight of edge (3, 1): 4
Enter the weight of edge (3, 2): 3
Enter the weight of edge (3, 3): 2

Minimum Spanning Tree using Prim's Algorithm:
Edge    Weight
0 - 1   5
0 - 2   9
0 - 3   5
PS C:\SOURAV\CODE\C++ language codes\ADS assignment>
```