5.12 VERIFYING UNIQUENESS OF MINIMUM SPANNING TREE (MST)

Question:

Given a graph with weights and a potential Minimum Spanning Tree (MST), verify if the given MST is unique. If it is not unique, provide another possible MST.

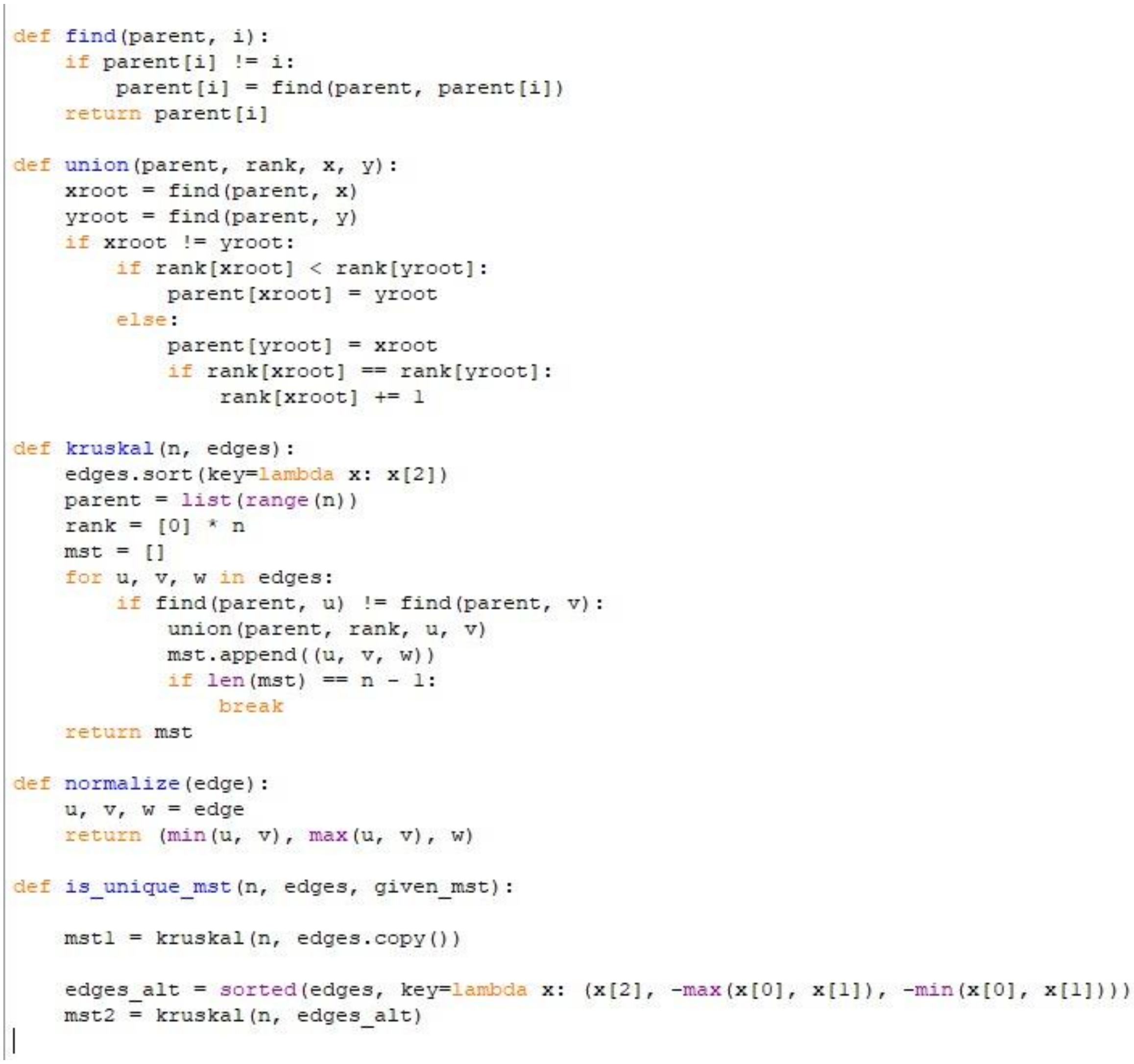
AIM

To verify the uniqueness of a given MST by checking for multiple valid edge choices with equal weights during Kruskal's execution.

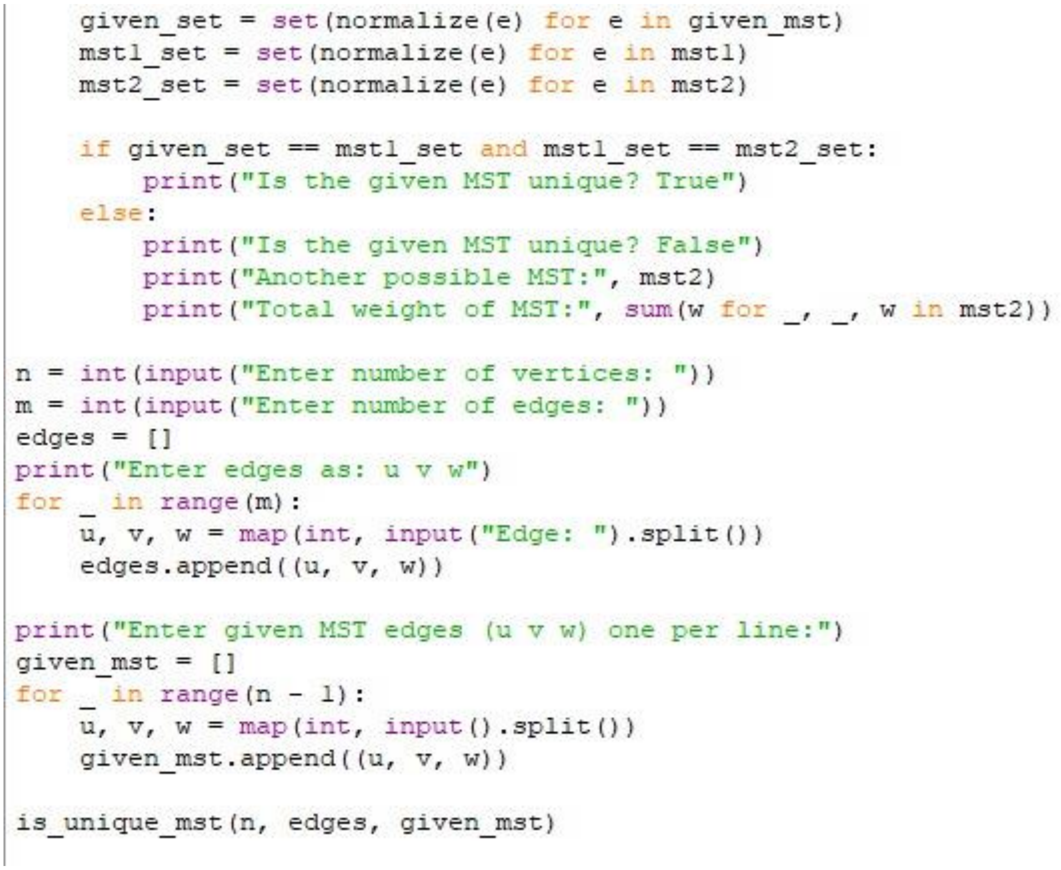
ALGORITHM

1. Sort all edges by weight.
2. Use Union-Find to build the MST using Kruskal's algorithm.
3. Track all edges added to the MST.
4. If at any point multiple edges with the same weight connect different components, and choosing any of them leads to a valid MST, then the MST is not unique.
5. Compare the given MST with the one generated by Kruskal.
6. If they differ but have the same total weight, the MST is not unique.

PROGRAM



```
def find(parent, i):
    if parent[i] != i:
        parent[i] = find(parent, parent[i])
    return parent[i]

def union(parent, rank, x, y):
    xroot = find(parent, x)
    yroot = find(parent, y)
    if xroot != yroot:
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        else:
            parent[yroot] = xroot
            if rank[xroot] == rank[yroot]:
                rank[xroot] += 1

def kruskal(n, edges):
    edges.sort(key=lambda x: x[2])
    parent = list(range(n))
    rank = [0] * n
    mst = []
    for u, v, w in edges:
        if find(parent, u) != find(parent, v):
            union(parent, rank, u, v)
            mst.append((u, v, w))
            if len(mst) == n - 1:
                break
    return mst

def normalize(edge):
    u, v, w = edge
    return (min(u, v), max(u, v), w)

def is_unique_mst(n, edges, given_mst):

    mst1 = kruskal(n, edges.copy())

    edges_alt = sorted(edges, key=lambda x: (x[2], -max(x[0], x[1]), -min(x[0], x[1])))
    mst2 = kruskal(n, edges_alt)
```



```python
    given_set = set(normalize(e) for e in given_mst)
    mst1_set = set(normalize(e) for e in mst1)
    mst2_set = set(normalize(e) for e in mst2)

    if given_set == mst1_set and mst1_set == mst2_set:
        print("Is the given MST unique? True")
    else:
        print("Is the given MST unique? False")
        print("Another possible MST:", mst2)
        print("Total weight of MST:", sum(w for _, _, w in mst2))

n = int(input("Enter number of vertices: "))
m = int(input("Enter number of edges: "))
edges = []
print("Enter edges as: u v w")
for _ in range(m):
    u, v, w = map(int, input("Edge: ").split())
    edges.append((u, v, w))

print("Enter given MST edges (u v w) one per line:")
given_mst = []
for _ in range(n - 1):
    u, v, w = map(int, input().split())
    given_mst.append((u, v, w))

is_unique_mst(n, edges, given_mst)
```

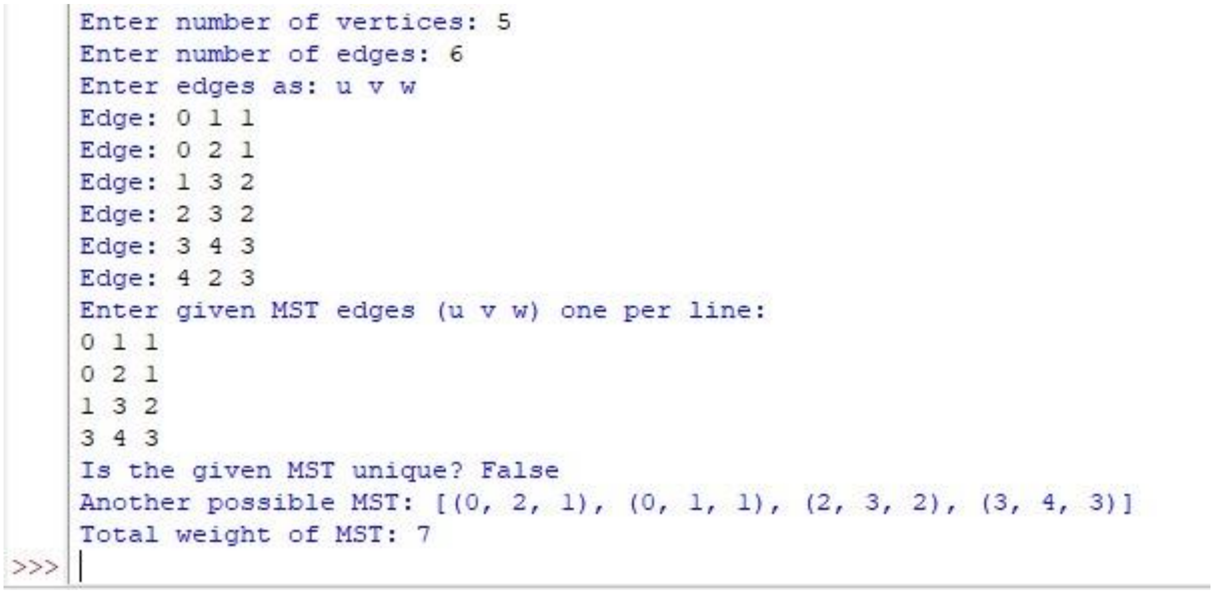
Input:

Enter number of vertices: 5
Enter number of edges: 6
Enter each edge as: u v w

Edge: 0 1 1
Edge: 0 2 1
Edge: 1 3 2
Edge: 2 3 2
Edge: 3 4 2
Edge: 4 2 3
Enter given MST edges (u v w) one

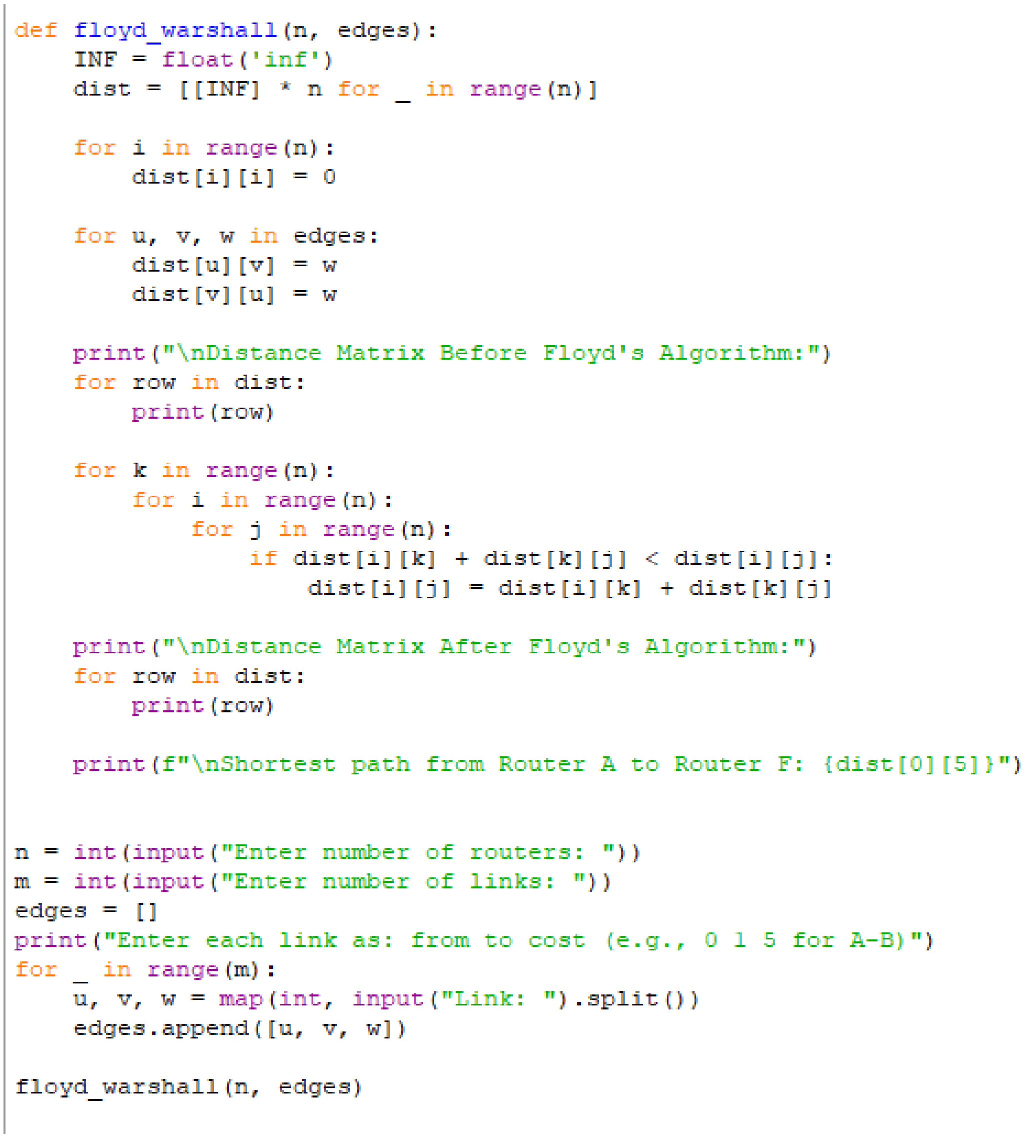
0 1 1
0 2 1
1 3 2
3 4 3

Output:



```
Enter number of vertices: 5
Enter number of edges: 6
Enter edges as: u v w
Edge: 0 1 1
Edge: 0 2 1
Edge: 1 3 2
Edge: 2 3 2
Edge: 3 4 3
Edge: 4 2 3
Enter given MST edges (u v w) one per line:
0 1 1
0 2 1
1 3 2
3 4 3
Is the given MST unique? False
Another possible MST: [(0, 2, 1), (0, 1, 1), (2, 3, 2), (3, 4, 3)]
Total weight of MST: 7
>>> 
```

b)

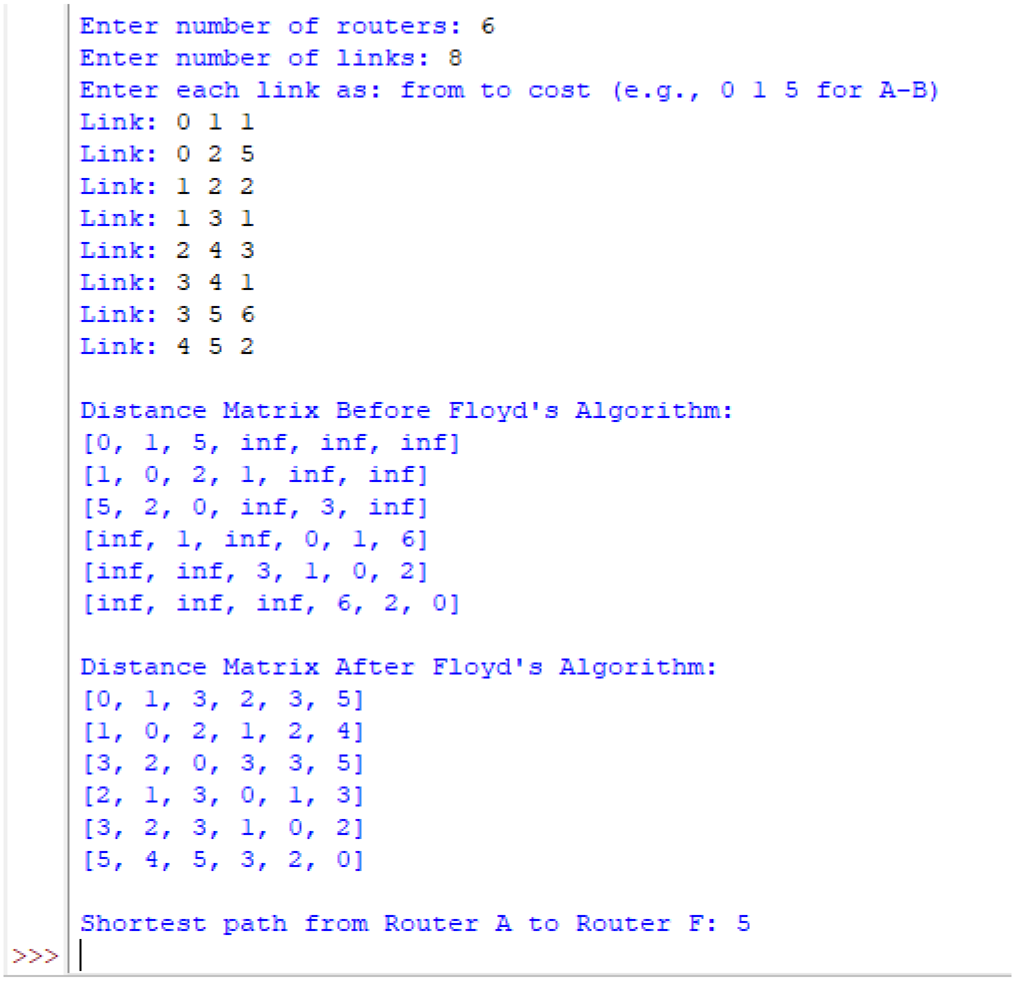


```python
def floyd_warshall(n, edges):
    INF = float('inf')
    dist = [[INF] * n for _ in range(n)]

    for i in range(n):
        dist[i][i] = 0

    for u, v, w in edges:
        dist[u][v] = w
        dist[v][u] = w

    print("\nDistance Matrix Before Floyd's Algorithm:")
    for row in dist:
        print(row)

    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    print("\nDistance Matrix After Floyd's Algorithm:")
    for row in dist:
        print(row)

    print(f"\nShortest path from Router A to Router F: {dist[0][5]}")


n = int(input("Enter number of routers: "))
m = int(input("Enter number of links: "))
edges = []
print("Enter each link as: from to cost (e.g., 0 1 5 for A-B)")
for _ in range(m):
    u, v, w = map(int, input("Link: ").split())
    edges.append([u, v, w])

floyd_warshall(n, edges)
```

Input

```
Enter number of routers: 6
Enter number of links: 8
Enter each link as: from to cost (e.g., 0 1 5 for A-B)

Edge: 0 1 1
Edge: 0 2 5
Edge: 1 2 2
Edge: 1 3 1
Edge: 2 4 3
Edge: 3 4 1
Edge: 3 5 6
Edge: 4 5 2
```

Output



```
Enter number of routers: 6
Enter number of links: 8
Enter each link as: from to cost (e.g., 0 1 5 for A-B)
Link: 0 1 1
Link: 0 2 5
Link: 1 2 2
Link: 1 3 1
Link: 2 4 3
Link: 3 4 1
Link: 3 5 6
Link: 4 5 2

Distance Matrix Before Floyd's Algorithm:
[0, 1, 5, inf, inf, inf]
[1, 0, 2, 1, inf, inf]
[5, 2, 0, inf, 3, inf]
[inf, 1, inf, 0, 1, 6]
[inf, inf, 3, 1, 0, 2]
[inf, inf, inf, 6, 2, 0]

Distance Matrix After Floyd's Algorithm:
[0, 1, 3, 2, 3, 5]
[1, 0, 2, 1, 2, 4]
[3, 2, 0, 3, 3, 5]
[2, 1, 3, 0, 1, 3]
[3, 2, 3, 1, 0, 2]
[5, 4, 5, 3, 2, 0]

Shortest path from Router A to Router F: 5
>>> |
```

RESULT:

Thus search program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

· **Time Complexity:** O(E log E), for sorting and Kruskal’s execution

· **Space Complexity:** O(E + N), for edge storage and Union-Find