```
# Kruskal's Algorithm
def kruskal(n, edges, names):
    # sort edges by weight (without lambda)
    for i in range(len(edges)):
        for j in range(i + 1, len(edges)):
            if edges[i][2] > edges[j][2]:
                edges[i], edges[j] = edges[j], edges[i]

    parent = list(range(n))  # parent array

    def find(i):
        while parent[i] != i:
            i = parent[i]
        return i

    mst = []
    total_cost = 0
    for u, v, w in edges:
        pu, pv = find(u), find(v)
        if pu != pv:  # if no cycle
            mst.append((u, v, w))
            total_cost += w
            parent[pu] = pv  # union

    print("\nMST using Kruskal's Algorithm:")
    for u, v, w in mst:
        print(f"{names[u]}--{names[v]}=={w}")
    print("Total Cost of MST =", total_cost)


# ------Prim's Algorithm------
def prim(n, adj, names):
    selected = [False] * n
    selected[0] = True  # start from node 0
    mst = []
    total_cost = 0

    for _ in range(n - 1):
        u = v = -1
        min_w = 999999
        for i in range(n):
```

```
            if selected[i]:
                for j in range(n):
                    if not selected[j] and adj[i][j] > 0:
                        if adj[i][j] < min_w:
                            u, v, min_w = i, j, adj[i][j]

        mst.append((u, v, min_w))
        total_cost += min_w
        selected[v] = True

    print("\nMST using Prim's Algorithm :")
    for u, v, w in mst:
        print(f"{names[u]} -- {names[v]} == {w}")
    print("Total Cost Of MST =", total_cost)


# ---------MAIN PROGRAM---------
n = int(input("Enter number of departments: "))
names = []
print("\n---------:Enter department names:---------")
for i in range(n):
    names.append(input(f"Department {i + 1} Name: "))

edges = []
m = int(input("\nEnter number of paths (edges): "))
print("\n--------:Enter edges:-------")
for _ in range(m):
    dept1 = input("\nDept1_Name=")
    dept2 = input("Dept2_Name=")
    w = int(input("Distance="))
    u = names.index(dept1)
    v = names.index(dept2)
    edges.append((u, v, w))

# adjacency matrix
adj = [[0] * n for _ in range(n)]
for u, v, w in edges:
    adj[u][v] = adj[v][u] = w

# Run algorithms
```

kruskal(n, edges, names)
prim(n, adj, names)