**Practical No.08**

**Title :**

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used.

**Objective :**

* To create a graph.
* To represent the graph in adjacency list or adjacency matrix.
* To check whether the graph is connected or not.

**Source Code :**

class Graph:

    def \_\_init\_\_(self):

        self.graph = {}

    def add\_edge(self, src, dest, cost):

        for node in [src, dest]:

            self.graph.setdefault(node, [])

        self.graph[src].append((dest, cost))

        self.graph[dest].append((src, cost))

    def dfs(self, start, visited):

        visited.add(start)

        [self.dfs(neighbor, visited) for neighbor, \_ in self.graph[start] if neighbor not in visited]

    def is\_connected(self):

        visited = set()

        self.dfs(next(iter(self.graph)), visited)

        return len(visited) == len(self.graph)

    def adjacency\_matrix(self):

        nodes = sorted(self.graph.keys())

        matrix = [[float('inf')] \* len(nodes) for \_ in nodes]

        indices = {node: i for i, node in enumerate(nodes)}

        for node, edges in self.graph.items():

            for neighbor, cost in edges:

                matrix[indices[node]][indices[neighbor]] = cost

        return matrix, nodes

def take\_input():

    g = Graph()

    for \_ in range(int(input("Enter the number of flight connections: "))):

        src, dest, cost = [input(f"Enter {s}: ").strip().upper() for s in ("source city/airport", "destination city/airport", "the cost")]

        g.add\_edge(src, dest, float(cost))

    return g

def main():

    g = take\_input()

    print("The graph is" + [" not", ""][g.is\_connected()] + " connected.")

    matrix, nodes = g.adjacency\_matrix()

    print("\nAdjacency Matrix:")

    print("\t" + "\t".join(nodes))

    for i, row in enumerate(matrix):

        print(nodes[i], end="\t")

        for cost in row:

            print(cost if cost != float('inf') else "-", end="\t")

        print()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output :**

