```
from collections import deque

# BFS
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    order = []
    while queue:
        node = queue.popleft()
        if node not in visited:
            visited.add(node)
            order.append(node)
            queue.extend(graph[node])
    return order

def bfs_practical():
    graph = {}

    n = int(input("Enter number of locations: "))
    print("Enter location names (e.g., A B C ...):")
    locations = input().split()

    for loc in locations:
        graph[loc] = []

    m = int(input("Enter number of routes: "))
    print("Enter routes as pairs (e.g., A B means route between A and B):")
    for _ in range(m):
        u, v = input().split()
        graph[u].append(v)
        graph[v].append(u)  # undirected
```

```
    start = input("Enter starting location: ")
    bfs_order = bfs(graph, start)
    print("BFS visiting sequence:", " -> ".join(bfs_order))

# DFS
class Graph:
    def _init_(self):
        self.city = []
        self.a = []
        self.n = 0

    def input_data(self):
        self.n = int(input("\nEnter number of cities: "))
        print("\nEnter the names of cities: ")
        self.city = [input(f"City {i+1}: ") for i in range(self.n)]
        self.a = [[0] * self.n for _ in range(self.n)]
        print("\nEnter the distances: ")
        for i in range(self.n):
            for j in range(i, self.n):
                if i == j:
                    self.a[i][j] = 0
                else:
                    d = int(input(f"Enter the distance between {self.city[i]} and {self.city[j]}: "))
                    self.a[i][j] = d
                    self.a[j][i] = d

    def display(self):
        print("\nAdjacency Matrix:")
        for i in range(self.n):
            for j in range(self.n):
```

```
            print(self.a[i][j], end="\t")
        print()

    def dfs(self):
        print("\nDFS Traversal:")
        visited = [0] * self.n
        start = input("Enter starting city: ")
        if start not in self.city:
            print("Invalid city!")
            return
        index = self.city.index(start)

        stack = [index]
        visited[index] = 1
        print(self.city[index], end=" -> ")
        while stack:
            current = stack[-1]
            found = False
            for i in range(self.n):
                if self.a[current][i] != 0 and not visited[i]:
                    visited[i] = 1
                    stack.append(i)
                    print(self.city[i], end=" -> ")
                    found = True
                    break
            if not found:
                stack.pop()
        print()

def dfs_practical():
    g = Graph()
```

```
    while True:
        print("\nDFS Practical Menu")
        print("1. Input data")
        print("2. Display data")
        print("3. DFS Traversal")
        print("4. Back to main menu")

        choice = int(input("Enter your choice: "))
        if choice == 1:
            g.input_data()
        elif choice == 2:
            g.display()
        elif choice == 3:
            g.dfs()
        elif choice == 4:
            break
        else:
            print("Invalid choice! Try again.")

# Main menu
def main():
    while True:
        print("\nMain Menu")
        print("1. BFS Practical")
        print("2. DFS Practical")
        print("3. Exit")

        choice = int(input("Enter your choice: "))
        if choice == 1:
            bfs_practical()
        elif choice == 2:
```

```
            dfs_practical()
        elif choice == 3:
            print("Exiting program.")
            break
        else:
            print("Invalid choice! Try again.")

if __name__ == "__main__":
    main()
```