Case Study

Map Navigation System

Problem Statement

In modern cities, navigating between locations efficiently is essential for transportation, logistics, and travel applications. Traditional maps require manual route tracing, which can be time-consuming and inaccurate. The problem is to develop a Map Navigation System that helps users find the shortest path between two locations using a graph-based algorithm.

Aim

To design and implement a Map Navigation System that finds the shortest route between two points using Dijkstra's Algorithm and displays the optimal path and its total distance.

Objectives

1. To represent city locations and roads using a graph data structure.
2. To implement Dijkstra's algorithm to find the shortest path.
3. To allow users to input source and destination nodes.
4. To calculate and display the minimum distance and path taken.
5. To demonstrate real-world navigation logic using a programming approach.

Description

The Map Navigation System models cities or locations as nodes (vertices) and the roads connecting them as edges with associated weights (distances). The system uses Dijkstra's algorithm, which explores all possible routes to determine the shortest path between two specified nodes. This approach is used in applications like Google Maps, Uber, and delivery routing systems, where optimal navigation saves time and resources.

Algorithm: Dijkstra's Algorithm

Step 1: Start from the source node.
Step 2: Set the initial distance to all nodes as infinity, except for the source (0).
Step 3: Mark all nodes as unvisited.
Step 4: For the current node, check all its unvisited neighbors and calculate the tentative distances.
Step 5: Update the neighbor's distance if the calculated value is less than the current known distance.
Step 6: Once all neighbors are checked, mark the current node as visited.
Step 7: Select the unvisited node with the smallest tentative distance as the next "current node."
Step 8: Repeat Steps 4–7 until all nodes are visited or the destination is reached.
Step 9: Output the shortest path and total distance.

Program

```
import heapq

def dijkstra(graph, start):

    # Initialize distances and priority queue

    distances = {node: float('inf') for node in graph}

    distances[start] = 0

    pq = [(0, start)]

    previous = {node: None for node in graph}

    while pq:

        current_distance, current_node = heapq.heappop(pq)

        if current_distance > distances[current_node]:

            continue

        for neighbor, weight in graph[current_node].items():
```

```
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous[neighbor] = current_node
                heapq.heappush(pq, (distance, neighbor))
    return distances, previous
def shortest_path(graph, start, target):
    distances, previous = dijkstra(graph, start)
    path = []
    node = target
    while node is not None:
        path.insert(0, node)
        node = previous[node]
    if distances[target] == float('inf'):
        print(f"No path from {start} to {target}")
    else:
        print(f"\nShortest distance from {start} to {target} = {distances[target]}")
        print(f"Path: {' -> '.join(path)}")
graph = {
    'A': {'B': 6, 'D': 1},
    'B': {'A': 6, 'C': 5, 'D': 2, 'E': 2},
    'C': {'B': 5, 'E': 5},
    'D': {'A': 1, 'B': 2, 'E': 1},
```

```
    'E': {'B': 2, 'C': 5, 'D': 1}
}

start = input("Enter the starting node: ").upper()

target = input("Enter the destination node: ").upper()

shortest_path(graph, start, target)
```

Sample Input

Enter the starting node: A

Enter the destination node: C

Sample Output

Shortest distance from A to C = 6

Path: A -> D -> E -> B -> C

Conclusion

The Map Navigation System developed using Python and Dijkstra's Algorithm effectively finds the shortest path between two locations. It demonstrates how navigation software optimizes routes to reduce travel time. This project can be extended by integrating real map data, graphical interfaces, and GPS coordinates for practical navigation applications.