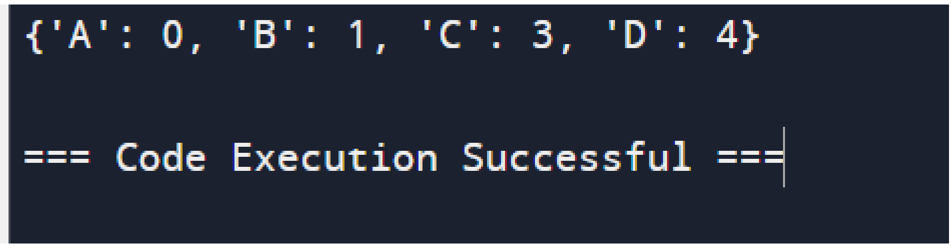
91. Single Source Shortest Paths: Dijkstra's Algorithm

PROGRAM:-

```
import heapq

def dijkstra(graph, start):
    # Initialize the priority queue
    priority_queue = [(0, start)]  # (distance, node)

    # Dictionary to store the shortest distance to each node
    distances = {node: float('inf') for node in graph}
    distances[start] = 0

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        # If the popped node has a greater distance than the recorded distance, skip it
        if current_distance > distances[current_node]:
            continue

        # Explore the neighbors of the current node
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight

            # If a shorter path to the neighbor is found
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances

# Example usage:
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

start_node = 'A'
print(dijkstra(graph, start_node))
# Output: {'A': 0, 'B': 1, 'C': 3, 'D': 4}
```

OUTPUT:-



```
{'A': 0, 'B': 1, 'C': 3, 'D': 4}

=== Code Execution Successful ===
```

TIME COMPLEXITY:-O((V+E)log V)