**Practical -5**

**Aim:** Write program to find shortest path in the given graph using Dijkstra algorithm to

demonstrate Greedy method techniques.

**Theory:**

**Dijkstra's Algorithm:**

Dijkstra's algorithm is a greedy algorithm used to find the shortest path between nodes in a graph. It works by maintaining a priority queue of vertices and continuously selecting the vertex with the minimum distance from a source vertex. The algorithm iteratively explores neighboring vertices and updates their distances if a shorter path is found. This process continues until all reachable vertices have been visited.The algorithm ensures optimality in finding the shortest path for non-negative edge weights in the graph.

**Acceptance Criteria:**

* The program should accept user input for defining a graph with edges and weights.
* Upon receiving the input, it should correctly implement Dijkstra's algorithm to find the shortest paths from the specified starting vertex to all other vertices.
* The output should display the shortest distances from the starting vertex to all other vertices in the graph.

**Code:**

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(dict)

def add\_edge(self, u, v, w):

self.graph[u][v] = w

self.graph[v][u] = w

def dijkstra(self, start):

vertices = set(self.graph.keys())

distances = {vertex: float('inf') for vertex in vertices}

distances[start] = 0

visited = set()

while vertices:

current\_vertex = None

for vertex in vertices:

if vertex in visited:

continue

if current\_vertex is None or distances[vertex] < distances[current\_vertex]:

current\_vertex = vertex

if current\_vertex is None:

break

vertices.remove(current\_vertex)

visited.add(current\_vertex)

for neighbor, weight in self.graph[current\_vertex].items():

new\_distance = distances[current\_vertex] + weight

if new\_distance < distances[neighbor]:

distances[neighbor] = new\_distance

return distances

def input\_graph():

graph = Graph()

num\_edges = int(input("Enter the number of edges: "))

print("Enter edges in the format 'source destination weight':")

for \_ in range(num\_edges):

source, destination, weight = input().split()

graph.add\_edge(source, destination, int(weight))

return graph

print("Input the graph:")

user\_graph = input\_graph()

start\_vertex = input("Enter the starting vertex: ")

shortest\_distances = user\_graph.dijkstra(start\_vertex)

print("Shortest distances from", start\_vertex, "to other vertices:")

for vertex, distance in shortest\_distances.items():

print(f"To {vertex}: Distance {distance}")

**Input:**

Input the graph:

Enter the number of edges: 5

Enter edges in the format 'source destination weight':

A B 4

A C 2

B C 5

B D 10

C D 3

Enter the starting vertex: A

**Output:**

Shortest distances from A to other vertices:

To D: Distance 5

To C: Distance 2

To B: Distance 4

To A: Distance 0

**Conclusion:**

The provided Python code demonstrates the implementation of Dijkstra's algorithm for finding the shortest paths in a graph. It allows users to input the graph structure and starting vertex, and then it calculates and displays the shortest distances from the starting vertex to all other vertices in the graph.

The acceptance criteria can be validated by running the code with different graph structures, ensuring it computes the correct shortest paths and displays the expected shortest distances from the specified starting vertex