**Name : Ayush patil**

**Roll no : TYCOC210**

**Batch C-3**

**Prn no : 122B1B210**

**Assignment – 4**

**Title :**

You have been given a network of ‘N’ nodes from 1 to ‘N’ and ‘M’ edges. For each edge, you are given three values (ui, vi, wi) where “ui” and “vi” denote the nodes and “wi” denotes an integer value which represents the time taken by a signal to travel from “ui” to “vi”. Now, you are supposed to find the time which a signal takes to travel from a given node ‘K’ to all nodes. If it is impossible for all nodes to receive the signal then print -1. Implement the given Network Delay Time using Dijkstra’s algorithm.

**Theory :**

Dijkstra’s algorithm is specifically designed to find the shortest paths in a graph with non-negative edge weights. In this problem, the time it takes for a signal to travel along an edge is always non-negative, which aligns with the assumptions of Dijkstra’s algorithm. The algorithm computes the shortest path from a given starting node (the source node) to every other node in the network. Dijkstra’s algorithm prioritizes nodes based on the shortest accumulated time (distance) from the source. It processes nodes in increasing order of the time it takes to reach them. It uses a greedy approach: at each step, the node with the smallest known travel time is selected and explored. The travel times to its neighboring nodes are updated if a shorter path is found.

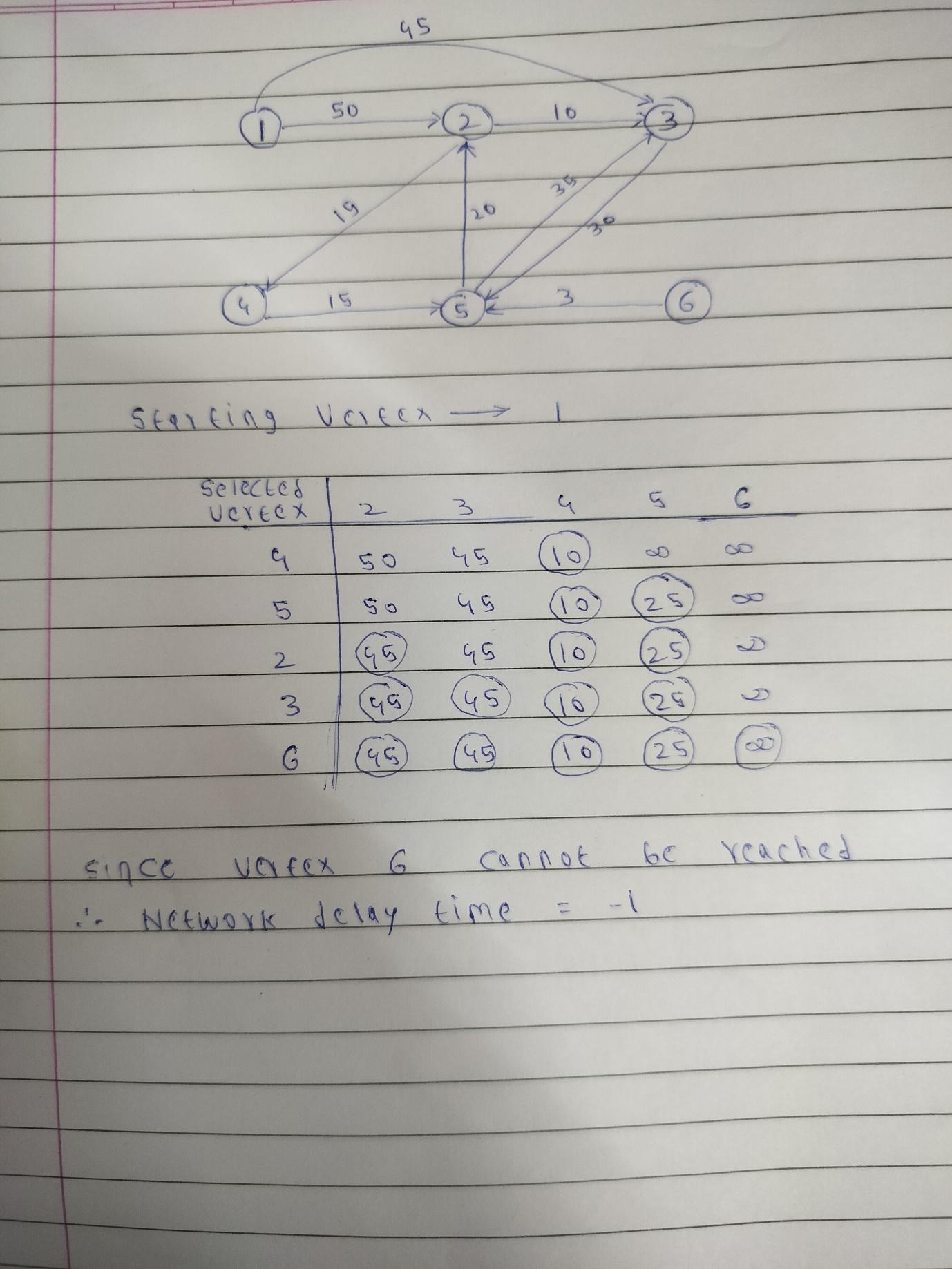
Time complexity : O((N+M)logN)

Space complexity : O(N^2)

Where,

N be the number of nodes

M be the number of edges in the graph.



**Code :**

import heapq

from collections import defaultdict

def network\_delay\_time\_dijkstra(times, N, K):

graph = defaultdict(list)

for u, v, w in times:

graph[u].append((v, w))

min\_heap = [(0, K)]

shortest\_time = {}

while min\_heap:

current\_time, node = heapq.heappop(min\_heap)

if node in shortest\_time:

continue

shortest\_time[node] = current\_time

for neighbor, travel\_time in graph[node]:

if neighbor not in shortest\_time:

heapq.heappush(min\_heap, (current\_time + travel\_time, neighbor))

if len(shortest\_time) == N:

return max(shortest\_time.values())

else:

return -1

def take\_user\_input():

N = int(input("Enter the number of nodes (N): "))

M = int(input("Enter the number of edges (M): "))

K = int(input("Enter the starting node (K): "))

print(f"Enter the {M} edges in the format (u, v, w):")

times = []

for \_ in range(M):

u, v, w = map(int, input().split())

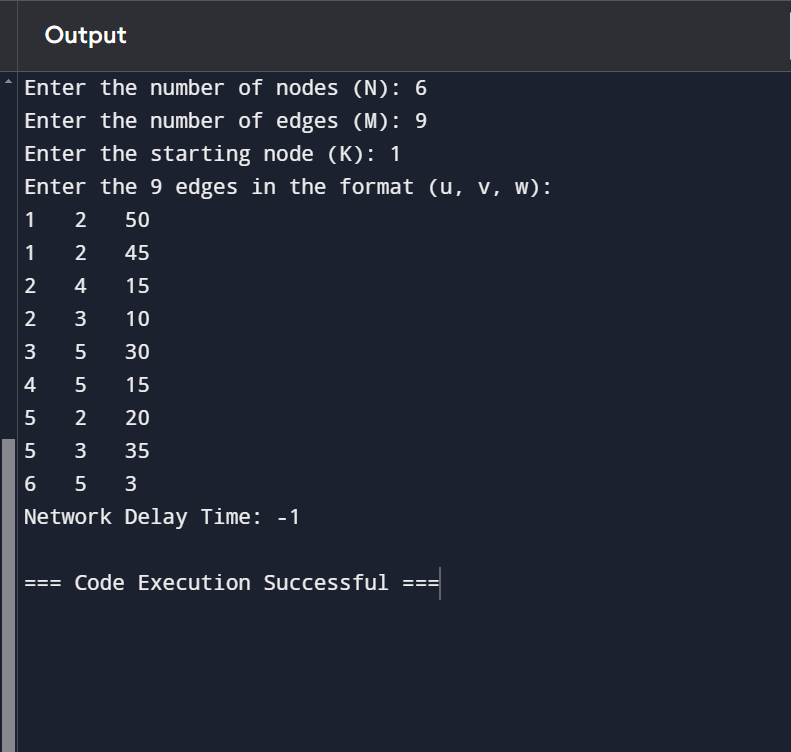
times.append([u, v, w])

result = network\_delay\_time\_dijkstra(times, N, K)

print(f"Network Delay Time: {result}")

take\_user\_input()

**Output :**

****