Week 3  
Name: Chaitanya  
Roll No: 24075A7206  
Date: 4-08-2025  
Subject: CH&EN LAB  
Aim: Implementation of Distance Vector Routing Algorithm.

🛰️ Distance Vector Routing (DVR) Algorithm – Notes  
  
🔹 Overview:  
Distance Vector Routing (DVR) is a dynamic routing algorithm used in computer networks to determine the best path for data packet transmission.  
It is commonly used in protocols such as:  
- RIP (Routing Information Protocol) – Versions 1 and 3  
- IGRP (Interior Gateway Routing Protocol)  
  
🔹 Key Features:  
In IGRP, routers are called gateways.  
Each router advertises a vector that includes:  
- The distance (cost) to each destination  
- The direction (next hop)  
  
Every router maintains a routing table that stores:  
- The best-known distances to reach every destination  
- The corresponding next-hop routers  
  
Routers periodically share their distance vectors with immediate neighbors, allowing them to dynamically adapt to changes in the network (e.g., failures, new routers).  
  
🔹 Underlying Algorithm:  
DVR uses the Bellman-Ford Shortest Path Algorithm to compute the least-cost paths across the network.  
  
📌 Bellman-Ford Formula:  
d\_x(y) = min\_v { cost(x, v) + d\_v(y) }  
  
Where:  
- x = source node  
- y = destination node  
- v = intermediate node (neighbor)  
- cost(x, v) = cost from node x to neighbor v  
- d\_v(y) = distance from v to destination y  
  
📝 Important Notes:  
1. If the source and destination are the same:  
 - Distance = 0  
 - Next node = same as source  
  
2. In each iteration, consider:  
 a. Minimum cost  
 b. Minimum number of intermediate nodes (if multiple paths have the same cost)  
  
3. Number of iterations:  
 - Perform (n - 2) iterations (where n = total number of nodes in the network)  
  
💻 Python Code:

n = int(input("Enter number of nodes: "))  
cost = [list(map(int, input().split())) for \_ in range(n)]  
dist = [[cost[i][j] if i != j else 0 for j in range(n)] for i in range(n)]  
nxt = [[j if i != j else i for j in range(n)] for i in range(n)]  
  
for \_ in range(n - 2):  
 for i in range(n):  
 for j in range(n):  
 for v in range(n):  
 if dist[i][j] > cost[i][v] + dist[v][j]:  
 dist[i][j] = cost[i][v] + dist[v][j]  
 nxt[i][j] = v  
  
print("\nDistance Table:")  
for i in range(n):  
 for j in range(n):  
 print(f"From {i} to {j}: cost = {dist[i][j]}, next hop = {nxt[i][j]}")

📥 Sample Input:  
Enter number of nodes: 4  
0 3 999 7  
3 0 1 999  
999 1 0 2  
7 999 2 0  
  
🖨️ Output:  
Distance Table:  
From 0 to 0: cost = 0, next hop = 0  
From 0 to 1: cost = 3, next hop = 1  
From 0 to 2: cost = 4, next hop = 1  
From 0 to 3: cost = 6, next hop = 3  
From 1 to 0: cost = 3, next hop = 0  
From 1 to 1: cost = 0, next hop = 1  
From 1 to 2: cost = 1, next hop = 2  
From 1 to 3: cost = 3, next hop = 2  
From 2 to 0: cost = 4, next hop = 1  
From 2 to 1: cost = 1, next hop = 1  
From 2 to 2: cost = 0, next hop = 2  
From 2 to 3: cost = 2, next hop = 3  
From 3 to 0: cost = 6, next hop = 0  
From 3 to 1: cost = 3, next hop = 2  
From 3 to 2: cost = 2, next hop = 2  
From 3 to 3: cost = 0, next hop = 3