COMPUTER NETWORKS

LAB ASSIGNMENT-10

REDDIPALLI SAI CHARISH

CS22B1095

BELLMAN-FORD:

```
#include <stdio.h>
#include <stdbool.h>

#define NODES 4
#define INF 9999  // Representing infinity
void print_routing_tables(int dist[NODES][NODES]) {
   printf("\nRouting tables:\n");
   for (int i = 0; i < NODES; i++) {
      printf("Node %c: ", 'A' + i);
      for (int j = 0; j < NODES; j++) {
         if (dist[i][j] == INF) {
            printf(" INF ");
         } else {
            printf("  %d  ", dist[i][j]);
         }
      }
      printf("\n");
   }
}
void distance_vector_routing(int graph[NODES][NODES]) {
   int dist[NODES][NODES];
   bool updated;
   for (int i = 0; i < NODES; i++) {
      for (int j = 0; j < NODES; j++) {
         dist[i][j] = graph[i][j];
      }
   }
   int step = 0;
   do {
      updated = false;
```

```
        printf("\n--- Step %d ---", step++);
        for (int src = 0; src < NODES; src++) {
            for (int dest = 0; dest < NODES; dest++) {
                for (int via = 0; via < NODES; via++) {
                    if (graph[src][via] != INF && dist[via][dest] != INF) {
                        int new_distance = graph[src][via] + dist[via][dest];
                        if (new_distance < dist[src][dest]) {
                            dist[src][dest] = new_distance;
                            updated = true;
                        }
                    }
                }
            }
        }
        print_routing_tables(dist);
    } while (updated);
}
int main() {
    int graph[NODES][NODES] = {
        {0, 1, 4, INF},
        {1, 0, 2, 6},
        {4, 2, 0, 3},
        {INF, 6, 3, 0}
    };
    printf("Initial topology matrix:\n");
    print_routing_tables(graph);
    // Perform distance vector routing
    distance_vector_routing(graph);
    printf("\n--- Convergence achieved ---\n");

    return 0;
}
```

```
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 10/output$ ./"bellman_ford"
Initial topology matrix:

Routing tables:
Node A:   0    1    4   INF
Node B:   1    0    2    6
Node C:   4    2    0    3
Node D:  INF   6    3    0

--- Step 0 ---
Routing tables:
Node A:   0    1    3    7
Node B:   1    0    2    5
Node C:   3    2    0    3
Node D:   6    5    3    0

--- Step 1 ---
Routing tables:
Node A:   0    1    3    6
Node B:   1    0    2    5
Node C:   3    2    0    3
Node D:   6    5    3    0

--- Step 2 ---
Routing tables:
Node A:   0    1    3    6
Node B:   1    0    2    5
Node C:   3    2    0    3
Node D:   6    5    3    0

--- Convergence achieved ---
```

DIJKSTRA'S:

```
#include <stdio.h>
#include <stdbool.h>

#define NODES 4
#define INF 9999  // Representing infinity for no direct link

void print_shortest_paths(int dist[], int prev[], int src) {
   printf("Shortest paths from node %c:\n", 'A' + src);
   for (int i = 0; i < NODES; i++) {
      if (i != src) {
         printf("To %c: Cost = %d, Path = %c", 'A' + i, dist[i], 'A' + src);
         int j = i;
         while (prev[j] != -1 && prev[j] != src) {
            printf(" -> %c", 'A' + prev[j]);
            j = prev[j];
         }
         printf(" -> %c\n", 'A' + i);
      }
   }
}

// Dijkstra's algorithm to find the shortest paths from a given source node
void dijkstra(int graph[NODES][NODES], int src) {
   int dist[NODES];     // Distance from src to each node
   bool visited[NODES];  // To check if the node is visited
   int prev[NODES];
   for (int i = 0; i < NODES; i++) {
      dist[i] = INF;
      visited[i] = false;
      prev[i] = -1;
   }
   dist[src] = 0;
   for (int count = 0; count < NODES - 1; count++) {
      int min = INF, u = -1;
      for (int v = 0; v < NODES; v++) {
         if (!visited[v] && dist[v] <= min) {
```

```
            min = dist[v];
            u = v;
          }
        }
        visited[u] = true;
        for (int v = 0; v < NODES; v++) {
          if (!visited[v] && graph[u][v] && dist[u] != INF &&
            dist[u] + graph[u][v] < dist[v]) {
            dist[v] = dist[u] + graph[u][v];
            prev[v] = u;
          }
        }
      }
      print_shortest_paths(dist, prev, src);
}
int main() {
      int graph[NODES][NODES] = {
        {0, 1, 4, INF},
        {1, 0, 2, 6},
        {4, 2, 0, 3},
        {INF, 6, 3, 0}
      };
      for (int i = 0; i < NODES; i++) {
        printf("\n--- Shortest paths from node %c ---\n", 'A' + i);
        dijkstra(graph, i);
      }
      return 0;
}
```

```
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 10/output$ ./"dijkstra"

--- Shortest paths from node A ---
Shortest paths from node A:
To B: Cost = 1, Path = A -> B
To C: Cost = 3, Path = A -> B -> C
To D: Cost = 6, Path = A -> C -> B -> D

--- Shortest paths from node B ---
Shortest paths from node B:
To A: Cost = 1, Path = B -> A
To C: Cost = 2, Path = B -> C
To D: Cost = 5, Path = B -> C -> D

--- Shortest paths from node C ---
Shortest paths from node C:
To A: Cost = 3, Path = C -> B -> A
To B: Cost = 2, Path = C -> B
To D: Cost = 3, Path = C -> D

--- Shortest paths from node D ---
Shortest paths from node D:
To A: Cost = 6, Path = D -> B -> C -> A
To B: Cost = 5, Path = D -> C -> B
To C: Cost = 3, Path = D -> C
```