**Lab Practical #13:**

To develop network using distance vector routing protocol and link state routing protocol.

**Practical Assignment #13:**

1. **C/Java Program: Distance Vector Routing Algorithm using Bellman Ford's Algorithm.**
2. #include <stdio.h>
3. #include <stdlib.h>
4. #define INF 99999
5. // Structure to represent an edge
6. struct Edge {
7. int src, dest, weight;
8. };
9. // Function to run Bellman-Ford algorithm
10. void BellmanFord(int V, int E, struct Edge edges[], int src) {
11. int dist[V];
12. // Step 1: Initialize distances
13. for (int i = 0; i < V; i++)
14. dist[i] = INF;
15. dist[src] = 0;
16. // Step 2: Relax all edges V-1 times
17. for (int i = 1; i <= V - 1; i++) {
18. for (int j = 0; j < E; j++) {
19. int u = edges[j].src;
20. int v = edges[j].dest;
21. int w = edges[j].weight;
22. if (dist[u] != INF && dist[u] + w < dist[v])
23. dist[v] = dist[u] + w;
24. }
25. }
26. // Step 3: Check for negative-weight cycles
27. for (int j = 0; j < E; j++) {
28. int u = edges[j].src;
29. int v = edges[j].dest;
30. int w = edges[j].weight;
31. if (dist[u] != INF && dist[u] + w < dist[v]) {
32. printf("Graph contains negative weight cycle!\n");
33. return;
34. }
35. }
36. // Print distances
37. printf("Vertex\tDistance from Source (%d)\n", src);
38. for (int i = 0; i < V; i++)
39. printf("%d\t%d\n", i, dist[i]);
40. }
41. int main() {
42. int V, E;
43. printf("Enter number of vertices and edges: ");
44. scanf("%d %d", &V, &E);
45. struct Edge edges[E];
46. printf("Enter edges (src dest weight):\n");
47. for (int i = 0; i < E; i++)
48. scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
49. int src;
50. printf("Enter source vertex: ");
51. scanf("%d", &src);
52. BellmanFord(V, E, edges, src);
53. return 0;
54. }

**2. C/Java Program: Link state routing algorithm.**

#include <stdio.h>

#define INF 9999

int main() {

    int V;

    printf("Enter number of vertices: ");

    scanf("%d", &V);

    // Dynamic adjacency matrix (VLA)

    int graph[V][V];

    printf("Enter adjacency matrix (%d x %d):\n", V, V);

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            scanf("%d", &graph[i][j]);

        }

    }

    int dist[V];     // shortest distances

    int visited[V];  // visited array

    int src;

    printf("Enter source vertex (0-%d): ", V - 1);

    scanf("%d", &src);

    // initialize

    for (int i = 0; i < V; i++) {

        dist[i] = INF;

        visited[i] = 0;

    }

    dist[src] = 0;

    // Dijkstra main loop

    for (int c = 0; c < V - 1; c++) {

        // find unvisited vertex with minimum distance

        int min = INF, u = -1;

        for (int i = 0; i < V; i++) {

            if (!visited[i] && dist[i] < min) {

                min = dist[i];

                u = i;

            }

        }

        visited[u] = 1;

        // update distances of neighbors

        for (int v = 0; v < V; v++) {

            if (graph[u][v] && !visited[v] && dist[u] + graph[u][v] < dist[v]) {

                dist[v] = dist[u] + graph[u][v];

            }

        }

    }

    // print result

    printf("\nShortest distances from source %d:\n", src);

    for (int i = 0; i < V; i++) {

        printf("To %d = %d\n", i, dist[i]);

    }

    return 0;

}