CSA 0317 DATA STRUCTURES

PROGRAM 23

```
#include <stdio.h>
#include <limits.h>
#include <string.h>

#define MAX_NODES 10
#define INF INT_MAX

// Structure to represent a graph
struct Graph {
    char nodes[MAX_NODES][10];
    int matrix[MAX_NODES][MAX_NODES];
    int node_count;
};

// Function to find node index
int findNodeIndex(struct Graph* g, char* node) {
    for (int i = 0; i < g->node_count; i++) {
        if (strcmp(g->nodes[i], node) == 0) {
            return i;
        }
    }
    return -1;
}

// Dijkstra's algorithm
void dijkstra(struct Graph* g, char* start, char* end) {
    int start_idx = findNodeIndex(g, start);
    int end_idx = findNodeIndex(g, end);
```

```
if (start_idx == -1 || end_idx == -1) {
    printf("Error: Start or end node not found!\n");
    return;
}

int dist[MAX_NODES];
int visited[MAX_NODES] = {0};
int previous[MAX_NODES];

// Initialize distances
for (int i = 0; i < g->node_count; i++) {
    dist[i] = INF;
    previous[i] = -1;
}
dist[start_idx] = 0;

// Dijkstra's algorithm
for (int count = 0; count < g->node_count - 1; count++) {
    // Find minimum distance vertex
    int min_dist = INF;
    int min_index = -1;

    for (int v = 0; v < g->node_count; v++) {
        if (!visited[v] && dist[v] <= min_dist) {
            min_dist = dist[v];
            min_index = v;
        }
    }

    if (min_index == -1) break;
```

```
    visited[min_index] = 1;

    // Update distances
    for (int v = 0; v < g->node_count; v++) {
        if (!visited[v] && g->matrix[min_index][v] != 0 &&
            dist[min_index] != INF &&
            dist[min_index] + g->matrix[min_index][v] < dist[v]) {
            dist[v] = dist[min_index] + g->matrix[min_index][v];
            previous[v] = min_index;
        }
    }
}

// Print result
printf("Shortest distance from %s to %s: %d\n", start, end, dist[end_idx]);

// Reconstruct path
if (dist[end_idx] == INF) {
    printf("No path found!\n");
    return;
}

// Build path in reverse
int path[MAX_NODES];
int path_length = 0;
int current = end_idx;

while (current != -1) {
    path[path_length++] = current;
    current = previous[current];
}
```

```
    printf("Path: ");
    for (int i = path_length - 1; i >= 0; i--) {
        printf("%s", g->nodes[path[i]]);
        if (i > 0) printf(" -> ");
    }
    printf("\n");
}

int main() {
    struct Graph g;
    g.node_count = 6;

    // Initialize node names
    strcpy(g.nodes[0], "A");
    strcpy(g.nodes[1], "B");
    strcpy(g.nodes[2], "C");
    strcpy(g.nodes[3], "D");
    strcpy(g.nodes[4], "E");
    strcpy(g.nodes[5], "F");

    // Initialize adjacency matrix with zeros
    for (int i = 0; i < MAX_NODES; i++) {
        for (int j = 0; j < MAX_NODES; j++) {
            g.matrix[i][j] = 0;
        }
    }

    // Add edges (undirected graph)
    g.matrix[0][1] = 4;  // A-B
    g.matrix[0][2] = 2;  // A-C
```

```
    g.matrix[1][0] = 4;  // B-A
    g.matrix[1][2] = 1;  // B-C
    g.matrix[1][3] = 5;  // B-D
    g.matrix[2][0] = 2;  // C-A
    g.matrix[2][1] = 1;  // C-B
    g.matrix[2][3] = 8;  // C-D
    g.matrix[2][4] = 10; // C-E
    g.matrix[3][1] = 5;  // D-B
    g.matrix[3][2] = 8;  // D-C
    g.matrix[3][4] = 2;  // D-E
    g.matrix[3][5] = 6;  // D-F
    g.matrix[4][2] = 10; // E-C
    g.matrix[4][3] = 2;  // E-D
    g.matrix[4][5] = 2;  // E-F
    g.matrix[5][3] = 6;  // F-D
    g.matrix[5][4] = 2;  // F-E

    // Find shortest path from A to F
    dijkstra(&g, "A", "F");

    return 0;
}
```

Output:

```
Output
Shortest distance from A to F: 12
Path: A -> C -> B -> D -> E -> F


=== Code Execution Successful ===
```