FLYOD'S ALGORITHM

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

// Function to find the minimum of two numbers
int min(int a, int b) {
    return (a < b) ? a : b;
}

// Function to apply Floyd's Algorithm to find all-pairs shortest paths
void floydAlgorithm(int graph[][100], int V) {
    int dist[V][V];

    // Initialize the distance matrix
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            dist[i][j] = graph[i][j];
        }
    }

    // Calculate the shortest paths using dynamic programming
    for (int k = 0; k < V; k++) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }

    // Print the shortest paths
    printf("Shortest paths between all pairs of vertices:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INT_MAX) {
                printf("INF\t");
            } else {
                printf("%d\t", dist[i][j]);
            }
        }
        printf("\n");
    }
}

int main() {
    int V;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &V);
```

```
    int graph[100][100];

    printf("Enter the adjacency matrix (INF for no edge):\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
            if (graph[i][j] == -1) {
                // Assume -1 represents infinity (no edge)
                graph[i][j] = INT_MAX;
            }
        }
    }

    floydAlgorithm(graph, V);

    return 0;
}
```