Experiment-6

Dijkstra’s Single source shortest path Algorithm

Program:

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

// Number of vertices in the graph
#define V 9

int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
        {
            min = dist[v];
            min_index = v;
        }
    return min_index;
}

void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t\t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
    int dist[V];
    bool sptSet[V];
    for (int i = 0; i < V; i++)
    {
        dist[i] = INT_MAX;
        sptSet[i] = false;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++)
    {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;

        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] !=
INT_MAX && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    // print the constructed distance array
        printSolution(dist);
    }

    // driver's code
    int main()
    {
        int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                            { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

        dijkstra(graph, 0);

        return 0;
    }
```

Experiment-7

Topological Ordering

Program:

```c
#include <stdio.h>
#define MAX 100
int adj[MAX][MAX];
int visited[MAX];
int n;
int order[MAX], idx;
void DFS(int v)
{
    visited[v] = 1;
    int i;
    for(i=0; i<n; i++)
    {
        if(adj[v][i] && !visited[i])
        {
            DFS(i);
        }
    }
    order[--idx] = v;
}
void topological_sort()
{
    int i;
    idx = n;
    for(i=0; i<n; i++)
    {
        visited[i] = 0;
    }
    for(i=0; i<n; i++)
    {
        if(!visited[i])
        {
            DFS(i);
        }
    }
}

int main()
{
    int i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++)
    {
            for(j=0; j<n; j++)
            {
                scanf("%d", &adj[i][j]);
            }
        }
        topological_sort();
        printf("Topological ordering of vertices:\n");
        for(i=0; i<n; i++)
        {
            printf("%d ", order[i]);
        }
        printf("\n");
        return 0;
    }
```