**AIM** :

TOPOLOGICAL SORT

To implement a topological sorting algorithm on a graph using a queue data structure, and to display the sorted vertices.

**ALGORITHM** :

Step 1: Start the program.

Step 2: Implement the ‘CreateGraph’ function to create a graph with n vertices.

Step 3: Input the number of vertices (n) and the adjacency matrix representing the graph.

Step 4: Initialize the indegree array to store the indegree of each vertex.

Step 5: Define the ‘AddEdge’ function to add edges to the graph and update the indegree array.

Step 6: Create the ‘Topsort’ function to perform the topological sort.

Step 7: Initialize a queue to store vertices with an indegree of 0.

Step 8: Enqueue vertices with an indegree of 0 and dequeue them when their indegree becomes 0.

Step 9: Update the ‘topnum’ array to store the topological order of the vertices.

Step 10: Define ‘DisplayTopSort’ function to display the topological order of the vertices.

Step 11: End the program.

**PROGRAM** :  
  
 #include <stdio.h>

#include <stdlib.h>

#define MAX 5

struct Queue {

int data[MAX];

int front, rear;

};

struct Graph {

int vertices[MAX];

int edges[MAX][MAX];

int indegree[MAX];

int topnum[MAX];

};

struct Queue\* CreateQueue() {

struct Queue\* q = (struct Queue\*)malloc(sizeof(struct Queue));

q->front = q->rear = 0;

return q;

}

void MakeEmpty(struct Queue\* q) {

q->front = q->rear = 0;

}

int IsEmpty(struct Queue\* q) {

return q->front == q->rear;

}

void Enqueue(int vertex, struct Queue\* q) {

q->data[q->rear++] = vertex;  
}

int Dequeue(struct Queue\* q) {

return q->data[q->front++];

}

void CreateGraph(struct Graph\* g) {

for (int i = 0; i < MAX; i++) {

g->vertices[i] = i;

g->indegree[i] = 0;

for (int j = 0; j < MAX; j++) {

g->edges[i][j] = 0;

}

}

}

void AddEdge(struct Graph\* g, int src, int dest) {

g->edges[src][dest] = 1;

g->indegree[dest]++;

}

void Topsort(struct Graph\* g) {

struct Queue\* q = CreateQueue();

MakeEmpty(q);

int counter = 0;

for (int i = 0; i < MAX; i++) {

if (g->indegree[i] == 0) {

Enqueue(i, q);

}

}

while (!IsEmpty(q)) {

int v = Dequeue(q);

g->topnum[v] = ++counter;

for (int w = 0; w < MAX; w++) {

if (g->edges[v][w] && --g->indegree[w] == 0) {

Enqueue(w, q);

}

}

}

if (counter != MAX) {

printf("Graph has a cycle\n");

}

free(q);

}

void DisplayTopSort(struct Graph\* g) {

printf("Topological Sorting: ");

for (int i = 0; i < MAX; i++) {

for (int j = 0; j < MAX; j++) {

if (g->topnum[j] == i + 1) {

printf("%d ", j);

break;

}

}

}

printf("\n");

}

int main() {

struct Graph g;

int numEdges, src, dest;

CreateGraph(&g);

printf("Enter the number of edges: ");

scanf("%d", &numEdges);

printf("Enter the edges one by one:\n");

for (int i = 0; i < numEdges; i++) {

scanf("%d %d", &src, &dest);

if (src >= MAX || dest >= MAX || src < 0 || dest < 0) {

printf("Invalid edge. Please enter vertices between 0 and %d.\n", MAX - 1);

i--;

} else {

AddEdge(&g, src, dest);

}

}

Topsort(&g);

DisplayTopSort(&g);

return 0;

}

**OUTPUT** :  
  
Enter the number of edges: 4

Enter the edges one by one:

0 1

1 2

2 3

3 4

Topological Sorting: 0 1 2 3 4

**RESULT** :

Thus, the program has been successfully executed and verified.