**DATASTRUCTURE**

**PROGRAMS:**

**1.Topological Sort**

*#include <stdio.h>*

*#include <stdlib.h>*

*#define MAX\_VERTICES 100*

*typedef struct {*

*int adj[MAX\_VERTICES][MAX\_VERTICES];*

*int in\_degree[MAX\_VERTICES];*

*int num\_vertices;*

*} Graph;*

*void initGraph(Graph \*g, int num\_vertices) {*

*g->num\_vertices = num\_vertices;*

*for (int i = 0; i < num\_vertices; i++) {*

*g->in\_degree[i] = 0;*

*for (int j = 0; j < num\_vertices; j++) {*

*g->adj[i][j] = 0;*

*}*

*}*

*}*

*void addEdge(Graph \*g, int start, int end) {*

*g->adj[start][end] = 1;*

*g->in\_degree[end]++;*

*}*

*void topologicalSort(Graph \*g) {*

*int in\_degree[MAX\_VERTICES];*

*int queue[MAX\_VERTICES];*

*int front = 0, rear = 0;*

*int count = 0;*

*for (int i = 0; i < g->num\_vertices; i++) {*

*in\_degree[i] = g->in\_degree[i];*

*}*

*for (int i = 0; i < g->num\_vertices; i++) {*

*if (in\_degree[i] == 0) {*

*queue[rear++] = i;*

*}*

*}*

*while (front < rear) {*

*int u = queue[front++];*

*printf("%d ", u);*

*count++;*

*for (int v = 0; v < g->num\_vertices; v++) {*

*if (g->adj[u][v]) {*

*in\_degree[v]--;*

*if (in\_degree[v] == 0) {*

*queue[rear++] = v;*

*}*

*}*

*}*

*}*

*if (count != g->num\_vertices) {*

*printf("\nGraph has a cycle");*

*}*

*printf("\n");*

*}*

*int main() {*

*Graph g;*

*initGraph(&g, 6);*

*addEdge(&g, 5, 2);*

*addEdge(&g, 5, 0);*

*addEdge(&g, 4, 0);*

*addEdge(&g, 4, 1);*

*addEdge(&g, 2, 3);*

*addEdge(&g, 3, 1);*

*printf("Topological Sort: ");*

*topologicalSort(&g);*

*return 0;*

*}*

**OUTPUT:**

*Topological Sort: 4 5 0 2 3 1*

**2.Terminology Sort**

*#include <stdio.h>*

*#include <stdlib.h>*

*#define MAX\_VERTICES 100*

*typedef struct {*

*int adj[MAX\_VERTICES][MAX\_VERTICES];*

*int in\_degree[MAX\_VERTICES];*

*int num\_vertices;*

*} Graph;*

*void initGraph(Graph \*g, int num\_vertices) {*

*g->num\_vertices = num\_vertices;*

*for (int i = 0; i < num\_vertices; i++) {*

*g->in\_degree[i] = 0;*

*for (int j = 0; j < num\_vertices; j++) {*

*g->adj[i][j] = 0;*

*}*

*}*

*}*

*void addEdge(Graph \*g, int start, int end) {*

*if (start >= g->num\_vertices || end >= g->num\_vertices) {*

*printf("Error: Vertex index out of bounds.\n");*

*return;*

*}*

*g->adj[start][end] = 1;*

*g->in\_degree[end]++;*

*}*

*void topologicalSort(Graph \*g) {*

*int in\_degree[MAX\_VERTICES];*

*int queue[MAX\_VERTICES];*

*int front = 0, rear = 0;*

*int count = 0;*

*for (int i = 0; i < g->num\_vertices; i++) {*

*in\_degree[i] = g->in\_degree[i];*

*}*

*for (int i = 0; i < g->num\_vertices; i++) {*

*if (in\_degree[i] == 0) {*

*queue[rear++] = i;*

*}*

*}*

*while (front < rear) {*

*int u = queue[front++];*

*printf("%d ", u);*

*count++;*

*for (int v = 0; v < g->num\_vertices; v++) {*

*if (g->adj[u][v]) {*

*in\_degree[v]--;*

*if (in\_degree[v] == 0) {*

*queue[rear++] = v;*

*}*

*}*

*}*

*}*

*if (count != g->num\_vertices) {*

*printf("\nGraph has a cycle\n");*

*} else {*

*printf("\nTopological sort completed successfully\n");*

*}*

*}*

*void printAdjMatrix(Graph \*g) {*

*printf("Adjacency Matrix:\n");*

*for (int i = 0; i < g->num\_vertices; i++) {*

*for (int j = 0; j < g->num\_vertices; j++) {*

*printf("%d ", g->adj[i][j]);*

*}*

*printf("\n");*

*}*

*}*

*void printInDegrees(Graph \*g) {*

*printf("In-degrees:\n");*

*for (int i = 0; i < g->num\_vertices; i++) {*

*printf("Vertex %d: %d\n", i, g->in\_degree[i]);*

*}*

*}*

*int main() {*

*Graph g;*

*int num\_vertices = 6;*

*initGraph(&g, num\_vertices);*

*addEdge(&g, 5, 2);*

*addEdge(&g, 5, 0);*

*addEdge(&g, 4, 0);*

*addEdge(&g, 4, 1);*

*addEdge(&g, 2, 3);*

*addEdge(&g, 3, 1);*

*printAdjMatrix(&g);*

*printInDegrees(&g);*

*printf("Topological Sort: ");*

*topologicalSort(&g);*

*return 0;*

*}*

**OUTPUT:**

*Adjacency Matrix:*

*0 0 0 0 0 0*

*0 0 0 0 0 0*

*0 0 0 1 0 0*

*0 1 0 0 0 0*

*1 1 0 0 0 0*

*1 0 1 0 0 0*

*In-degrees:*

*Vertex 0: 2*

*Vertex 1: 2*

*Vertex 2: 1*

*Vertex 3: 1*

*Vertex 4: 0*

*Vertex 5: 0*

*Topological Sort: 4 5 0 2 3 1*

*Topological sort completed successfully*