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