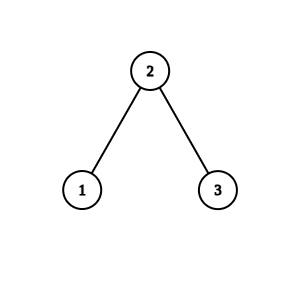
**TASK 10**

**Greedy Technique – Topological Sort CO3 K3**

Chef has a tree  *G* with  *N* nodes (numbered 1 through  *N*) and  *N*−1 undirected edges. For each node  *u* in the tree, he can assign directions to the edges in such a way that  *u* is reachable from all the other nodes. Let  *Cu*​ be the number of valid [topological orderings](https://en.wikipedia.org/wiki/Topological_sorting) in the resulting directed acyclic graph.

For example, consider the tree below:

**Input:**

Enter the no of vertices:

3

Enter the adjacency matrix:

Enter row 1

0 1 1

Enter row 2

0 0 1

Enter row 3

0 0 0

**Output:**

The topological order is:1 2 3

**Test Case 1:** Assume a Directed Acyclic Graph (DAG) as an input and check whether Topological sorting can be applied or not?

**Test Case 2:** Implement topological sorting using Breadth First Search (BFS)

**Aim:**

Implement a c program for topological ordering

**Algorithm:**

**Step 1:** Find indegree for all nodes.

**Step 2:** Identify a node with no incoming edges.

**Step 3:** Remove the node from the graph and add it to the ordering.

**Step 4:** Remove the node’s out-going edges from the graph.

**Step 5:** Decrement the in-degree where connected edges were deleted.

**Step 6:**Repeat Steps 1 to 4 till there are no nodes left with zero in-degree.

**Step 7:**Check if all elements are present in the sorted order.

**Step 8:** If the result of Step 6 is true, we have the sorted order. Else no topological ordering exists.

**Step 9:** Exit.

**Program:**

#include <stdio.h>

int main() {

int i, j, k, n, a[10][10], indeg[10], flag[10], count = 0;

printf("Enter the number of vertices:\n");

scanf("%d", &n);

printf("Enter the adjacency matrix:\n");

for (i = 0; i < n; i++) {

printf("Enter row %d: ", i + 1);

for (j = 0; j < n; j++)

scanf("%d", &a[i][j]);

}

for (i = 0; i < n; i++) {

indeg[i] = 0;

flag[i] = 0;

}

// Compute in-degree of each vertex

for (i = 0; i < n; i++)

for (j = 0; j < n; j++)

indeg[i] += a[j][i];

printf("\nThe topological order is: ");

while (count < n) {

for (k = 0; k < n; k++) {

if ((indeg[k] == 0) && (flag[k] == 0)) {

printf("%d ", k + 1);

flag[k] = 1;

count++;

// Reduce in-degree of adjacent vertices

for (i = 0; i < n; i++) {

if (a[k][i] == 1)

indeg[i]--;

}

}

}

}

printf("\n");

return 0;

}

**Sample Input/Output:**

Enter the number of vertices: 6

Enter the adjacency matrix:

Enter row 1: 0 1 1 0 0 0

Enter row 2: 0 0 0 1 0 0

Enter row 3: 0 0 0 1 1 0

Enter row 4: 0 0 0 0 0 1

Enter row 5: 0 0 0 0 0 1

Enter row 6: 0 0 0 0 0 0

The topological order is: 1 2 3 4 5 6

**Test Case 1:** Assume a Directed Acyclic Graph (DAG) as an input and check whether Topological sorting can be applied or not.

**Testcase 1:**

Enter the no of vertices:

4

Enter the adjacency matrix:

Enter row 1

0 1 1 0

Enter row 2

0 0 0 1

Enter row 3

0 0 0 1

Enter row 4

0 0 0 0

The topological order is:1 2 3 4

**Test Case 2:** Implement topological sorting using Breadth First Search (BFS)

Algorithm :

Program:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_NODES 100005

typedef struct {

int vertex;

struct Node\* next;

} Node;

Node\* graph[MAX\_NODES];

int indegree[MAX\_NODES];

// Function to add an edge to the graph

void addEdge(int u, int v) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->vertex = v;

newNode->next = graph[u];

graph[u] = newNode;

}

// Function to perform topological sort using BFS

void topologicalSort(int N) {

int queue[MAX\_NODES], front = 0, rear = 0;

for (int i = 1; i <= N; i++) {

if (indegree[i] == 0)

queue[rear++] = i;

}

while (front < rear) {

int u = queue[front++];

printf("%d ", u); // Output the node in topological order

Node\* temp = graph[u];

while (temp != NULL) {

int v = temp->vertex;

indegree[v]--;

if (indegree[v] == 0)

queue[rear++] = v;

temp = temp->next;

}

}

}

int main() {

int N, M;

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

scanf("%d %d", &N, &M);

// Initialize graph and arrays

for (int i = 1; i <= N; i++) {

graph[i] = NULL;

indegree[i] = 0;

}

// Read edges and construct the graph

printf("Enter the edges (u v):\n");

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

int u, v;

scanf("%d %d", &u, &v);

addEdge(u, v);

indegree[v]++;

}

printf("Topological ordering: ");

topologicalSort(N);

printf("\n");

return 0;

}

Input:

Output;

**Result:**

Thus the c program for topological ordering was executed.