

**LAB 5:**

/\* 1). Write a program to determine the Topological sort of a given graph using  
i. Depth-First technique  
ii. Source removal technique \*/

```
#include <stdio.h>
#include <stdlib.h>

int a[50][50], visit[50], stack[100], n, t=0;

void dfs(int v)
{
    visit[v]=1;

    for(int i=0; i<n; i++)
    {
        if(a[v][i] && !visit[i])
        {
            dfs(i);
        }
    }

    stack[t++]=v;
}

void printStack()
{
    for(int i=n-1; i>=0; i--)
    {
        printf("%d\n", stack[i]);
    }

    printf("\n");
}

int main()
{
    printf("Enter the Number of Vertices : \n");
    scanf("%d", &n);
    printf("Enter the Adjacency Matrix : \n");
    for(int i = 0; i<n; i++)
    {
        for(int j = 0; j<n; j++)
```

```

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

for(int i = 0; i<n; i++)
{
if(!visit[i])
{
dfs(i);
}
}

printf("The Topological Sort Order is :\n");
printStack();

return 0;
}

```

## OUTPUT :

The screenshot displays the Visual Studio Code editor with a C program for topological sorting. The code is in a file named `topologicalSort.c`. It includes `stdio.h` and `stdlib.h`, and defines arrays for the adjacency matrix `a`, visit status `visit`, and a stack. The `dfs` function recursively visits nodes and pushes them onto the stack. The `printStack` function prints the stack contents in reverse order. The terminal output shows the program being compiled and run, with the following interaction:

```

linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ gcc topologicalSort.c -o topo
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./topo
Enter the Number of Vertices :
4
Enter the Adjacency Matrix :
1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1
The Topological Sort Order is :
1
3
0
2
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$

```

```
/* 1). Write a program to determine the Topological sort of a given graph using  
i. Depth-First technique  
ii. Source removal technique  
*/
```

```
#include <stdio.h>  
#include <stdlib.h>
```

```
int queue[100], k_1 = 0, k = 0, arr[100][100], n, indegree[100];  
void calc()  
{  
    for(int i = 0; i<n; i++)  
    {  
        for(int j = 0; j<n; j++)  
        {  
            if(arr[j][i] && i!=j)  
            {  
                indegree[i]++;  
            }  
        }  
    }  
}
```

```
void initQueue()  
{  
    for(int i = 0; i<n; i++)  
    {  
        queue[i] = -1;  
    }  
}
```

```
void dec(int v)  
{  
    for(int i = 0; i<n; i++)  
    {  
        if(arr[v][i])  
        {  
            indegree[i]--;  
        }  
        if(indegree[i] == 0)  
        {  
            queue[k++] = i;  
        }  
    }  
}
```

```
int queueEmpty()
```

```

{
for(int i = 0; i<n; i++)
{
if(queue[i] != -1)
{
return 0;
}
}

return 1;
}

int main()
{
printf("Enter the Number of Vertices : \n");
scanf("%d", &n);
printf("Enter the Adjacency Matrix : \n");

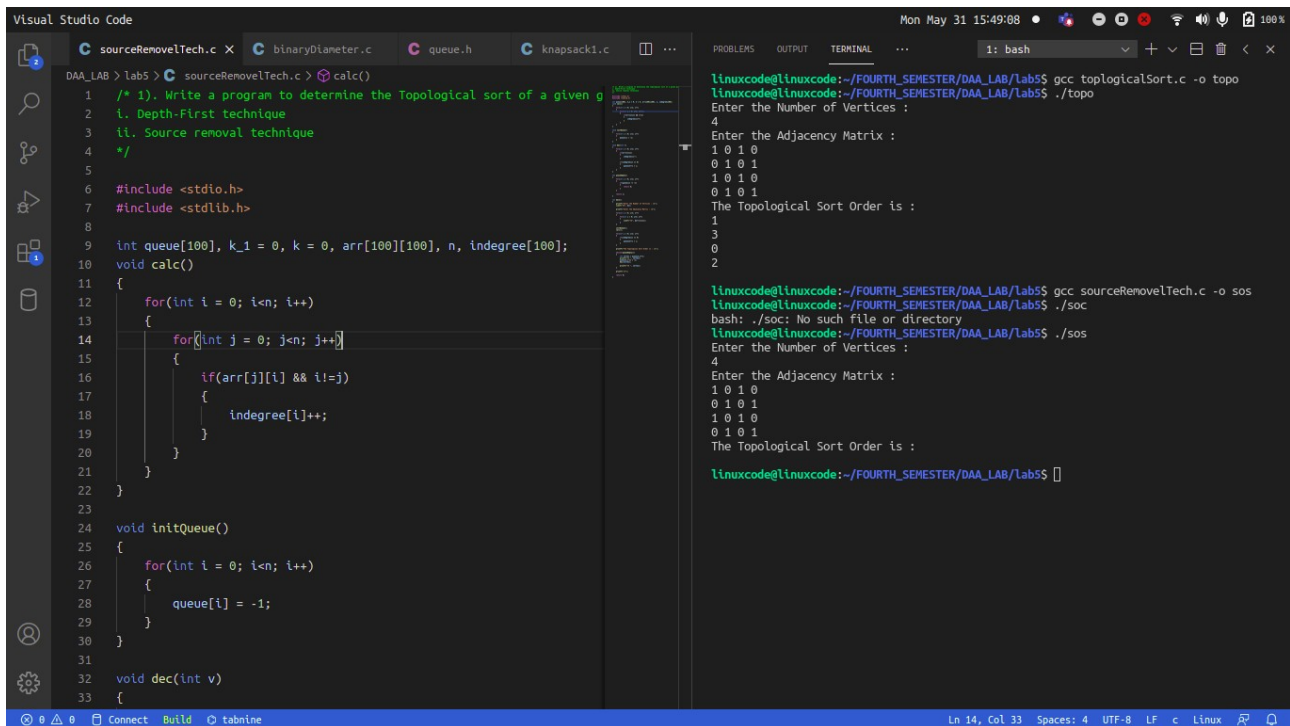
for(int i = 0; i<n; i++)
{
for(int j = 0; j<n; j++)
{
scanf("%d", &arr[i][j]);
}
}
initQueue();
calc();
for(int i = 0; i<n; i++)
{
if(indegree[i] == 0)
{
queue[k++] = i;
}
}
printf("The Topological Sort Order is : \n");
while(!queueEmpty())
{
int vertex = queue[k_1++];
printf("%d ", vertex);
queue[k_1-1] = -1;
dec(vertex);

printf("%d ", vertex);
}
printf("\n");

return 0;
}

```

## OUTPUT :



The screenshot shows the Visual Studio Code editor with a C program for topological sorting. The code is in a file named `sourceRemovelTech.c`. The program prompts the user to enter the number of vertices and the adjacency matrix, then outputs the topological sort order. The terminal window shows the execution of the program, with the user entering 4 vertices and the adjacency matrix [[0,1,0],[0,1,0],[0,1,0],[0,1,0]]. The output is the topological sort order: 1, 3, 0, 2.

```
1  /* 1). Write a program to determine the Topological sort of a given g
2  i. Depth-First technique
3  ii. Source removal technique
4  */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  int queue[100], k_1 = 0, k = 0, arr[100][100], n, indegree[100];
10 void calc()
11 {
12     for(int i = 0; i<n; i++)
13     {
14         for(int j = 0; j<n; j++)
15         {
16             if(arr[j][i] && i!=j)
17             {
18                 indegree[i]++;
19             }
20         }
21     }
22 }
23
24 void initQueue()
25 {
26     for(int i = 0; i<n; i++)
27     {
28         queue[i] = -1;
29     }
30 }
31
32 void dec(int v)
33 {
```

```
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ gcc topologicalSort.c -o topo
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./topo
Enter the Number of Vertices :
4
Enter the Adjacency Matrix :
1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1
The Topological Sort Order is :
1
3
0
2

linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ gcc sourceRemovelTech.c -o sos
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./sos
bash: ./sos: No such file or directory
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./sos
Enter the Number of Vertices :
4
Enter the Adjacency Matrix :
1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1
The Topological Sort Order is :

linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$
```

/\*2. Write a program to find diameter of a binary tree. Diameter of a binary tree is

the longest path between any two nodes.

Program: \*/

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

```
#include <stdlib.h>
```

```
struct node {
```

```
int val;
```

```
struct node *left, *right;
```

```
};
```

```
struct node* newNode(int value)
```

```
{
```

```
struct node* node
```

```
= (struct node*)malloc(sizeof(struct node));
```

```
node->val = value;
```

```
node->left = NULL;
```

```
node->right = NULL;
```

```
return (node);
```

```
}
```

```
int max(int a, int b)
```

```
{
```

```
return (a > b) ? a : b;
```

```
}
```

```
int height(struct node* node)
```

```
{
```

```

if (node == NULL)
return 0;
return 1 + max(height(node->left), height(node->right));
}
int diameter(struct node* tree)
{
if (tree == NULL)
return 0;
int lheight = height(tree->left);
int rheight = height(tree->right);
int ldiam = diameter(tree->left);
int rdiam = diameter(tree->right);
return max(lheight + rheight + 1, max(ldiam, rdiam));
}
int main()
{
struct node* root = newNode(1);
root->left = newNode(2);
root->left->left = newNode(4);
root->left->right = newNode(5);
root->left->right->left = newNode(6);
root->left->right->right = newNode(7);

root->right = newNode(3);
root->right->right = newNode(8);
root->right->right->right = newNode(9);
root->right->right->right->right = newNode(10);
root->right->right->right->left = newNode(11);
root->right->right->right->left->left = newNode(12);
root->right->right->right->left->right = newNode(13);
root->right->right->right->left->right->left = newNode(14);
root->right->right->right->left->right->right = newNode(15);
printf("Diameter of the given binary tree is %d\n",
diameter(root));
return 0;
}

```

## OUTPUT :

The image shows a Visual Studio Code editor with a C program to find the diameter of a binary tree. The code is split into two files: `sourceRenovelTech.c` and `binaryDiameter.c`. The `binaryDiameter.c` file contains the main logic for finding the diameter.

```
1 /*2. Write a program to find diameter of the longest path between any two nodes.
2 Program: */
3 #include <stdio.h>
4 #include <stdlib.h>
5 struct node {
6     int val;
7     struct node *left, *right;
8 };
9 struct node* newNode(int value)
10 {
11     struct node* node
12     = (struct node*)malloc(sizeof(struct node));
13     node->val = value;
14     node->left = NULL;
15     node->right = NULL;
16     return (node);
17 }
18 int max(int a, int b)
19 {
20     return (a > b) ? a : b;
21 }
22 int height(struct node* node)
23 {
24     if (node == NULL)
25         return 0;
26     return 1 + max(height(node->left), height(node->right));
27 }
28 int diameter(struct node* tree)
29 {
30     if (tree == NULL)
31         return 0;
32     int lheight = height(tree->left);
```

The terminal output shows the compilation and execution of the program. It prompts for the number of vertices (4) and the adjacency matrix, then displays the topological sort order (1, 3, 0, 2) and the diameter of the given binary tree (10).

```
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ gcc topologicalSort.c -o topo
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./topo
Enter the Number of Vertices :
4
Enter the Adjacency Matrix :
1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1
The Topological Sort Order is :
1
3
0
2

linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ gcc sourceRenovelTech.c -o sos
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./sos
bash: ./sos: No such file or directory
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./sos
Enter the Number of Vertices :
4
Enter the Adjacency Matrix :
1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1
The Topological Sort Order is :

linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ gcc binaryDiameter.c -o bb
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$ ./bb
Diameter of the given binary tree is 10
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab5$
```