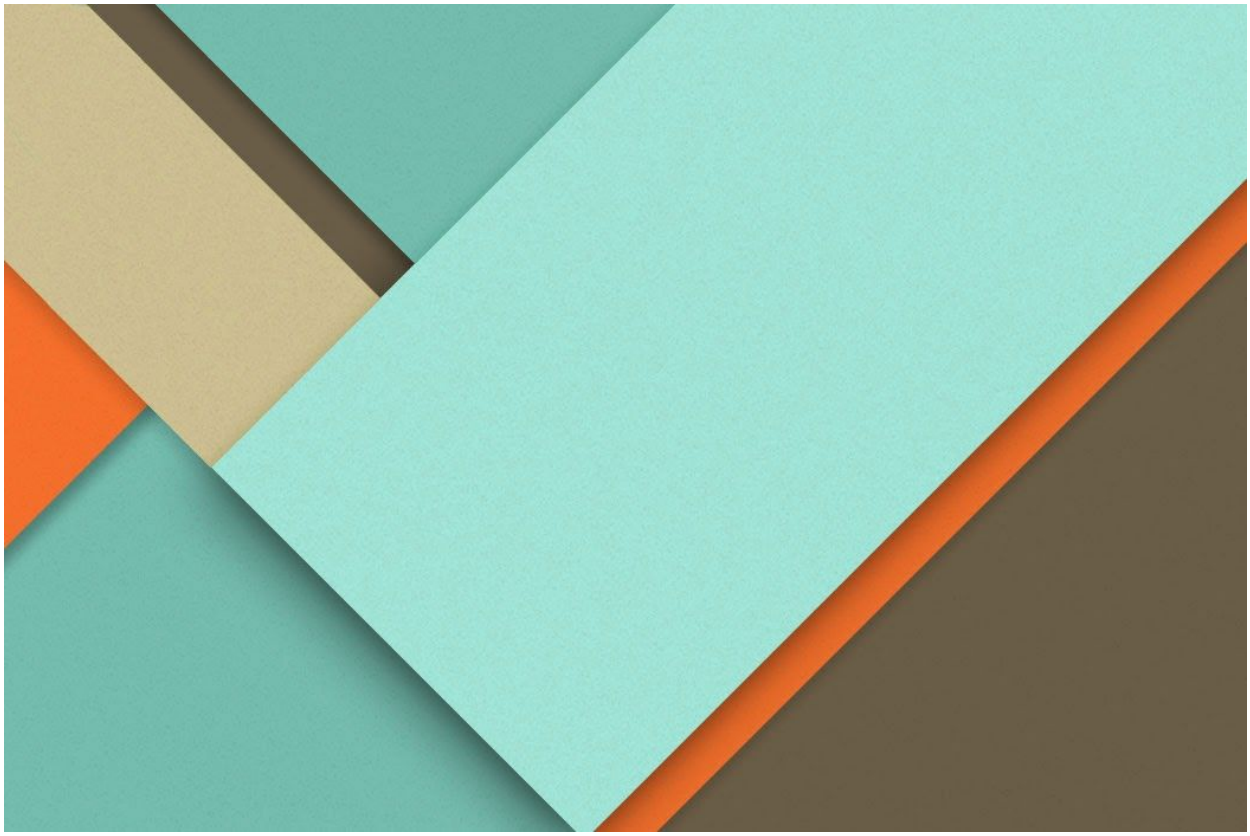

DSA Assignment

NAME: AISWARYA REDDY DEVIREDDY

REG.NO: AP19110010179

1st YEAR, CSE-D

Title:

Write a C program to create an adjacency matrix of order n*n for the given communication network and print the output in the matrix form.

Objective:

At the end of this activity, we shall be able to

Use for loops and data types in C

Problem Statement:

In this program, we aim to understand and use various data types and for loops to find the sum and average of an array. It is required to get the input from the user such as:

Enter no of vertices:

Enter no of edges:

Once we collect this information, we print the matrix.

Algorithm:

START

DEFINE VARIABLES: n, edges, u, v

INPUT: Read the input from the user.

COMPUTATION: Adds each element and prints the output matrix

DISPLAY: Prints the output matrix.

STOP

Program in C:

```
#include <stdio.h>
int main()
{
    int n,edges,u,v;
    printf("Enter No of vertices: ");
    scanf("%d",&n);
    printf("Enter no of edges: ");
    scanf("%d",&edges);
    int mat[100][100];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            mat[n][n]=0;
        }
    }
    for(int i=0;i<edges;i++)
    {
        printf("Enter from vertex: ");
        scanf("%d",&u);
        printf("Enter to vertex: ");
        scanf("%d",&v);
        mat[u][v]=1;
        mat[v][u]=1;
    }
    for(int i=0;i<n;i++)
```

```
    {
        for(int j=0;j<n;j++)
        {
            printf("%d\t",mat[i][j]);
        }
        printf("\n");
    }
}
```

Output:

TEST CASE-1:

Enter No of vertices: 8

Enter no of edges: 13

Enter from vertex: 0

Enter to vertex: 1

Enter from vertex: 0

Enter to vertex: 2

Enter from vertex: 0

Enter to vertex: 6

Enter from vertex: 1

Enter to vertex: 2

Enter from vertex: 1

Enter to vertex: 3

Enter from vertex: 2

Enter to vertex: 3

Enter from vertex: 2

Enter to vertex: 4

Enter from vertex: 3

Enter to vertex: 4

Enter from vertex: 3

Enter to vertex: 7

Enter from vertex: 4

Enter to vertex: 5

Enter from vertex: 4

Enter to vertex: 6

Enter from vertex: 5

Enter to vertex: 6

Enter from vertex: 5

Enter to vertex: 7

0	1	1	0	0	0	1	0
1	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	0	1	1	0	1	1	0
0	0	0	0	1	0	1	1
1	0	0	0	1	1	0	0
0	0	0	1	0	1	0	0

Process returned 0 (0x0) execution time : 92.143 s

Press any key to continue.

Conclusion:

Adjacency matrix: It occupies n2/8 byte space (one bit per entry). It is an N*N binary matrix in which the value of [i,j]th cell is 1 if there exists an edge originating from the ith vertex and terminating to jth vertex, otherwise the value is 0.

Adjacency list: It occupies 8e space, where e is the number of edges (32bit computer). It is an array of separate lists. Each element of the array is a list of corresponding neighbor (or directly connected) vertices. In other words, ith list of Adjacency List is a list of all those vertices which are directly connected to the ith vertex.

2. Repeat question 1 but this time create an adjacency list.

Program on c:

```
// To demonstrate the adjacency list
// Representation of graphs
#include <stdio.h>
#include <stdlib.h>
// A structure to represent an adjacency list node
struct AdjListNode
{
 int dest;
 struct AdjListNode* next;
};
// A structure to represent an adjacency list
struct AdjList
{
```

```
 struct AdjListNode *head;
};
// Size of array will be V (number of vertices in graph)
struct Graph
{
 int V;
 struct AdjList* array;
};
// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest)
{
 struct AdjListNode* newNode =
 (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
 newNode->dest = dest;
 newNode->next = NULL;
 return newNode;
}
struct Graph* createGraph(int V)
{
 struct Graph* graph =
 (struct Graph*) malloc(sizeof(struct Graph));
 graph->V = V;
 graph->array =
 (struct AdjList*) malloc(V * sizeof(struct AdjList));
 // Initialize each adjacency list as empty by making head as NULL
 int i;
 for (i = 0; i < V; ++i)
```

```
 graph->array[i].head = NULL;
 return graph;
}
// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest)
{
 struct AdjListNode* newNode = newAdjListNode(dest);
 newNode->next = graph->array[src].head;
 graph->array[src].head = newNode;
 // Since graph is undirected, add an edge from dest to src also
 newNode = newAdjListNode(src);
 newNode->next = graph->array[dest].head;
 graph->array[dest].head = newNode;
}
// A utility function to print the adjacency list representation of graph
void printGraph(struct Graph* graph)
{
 int v;
 for (v = 0; v < graph->V; ++v)
 {
 struct AdjListNode* pCrawl = graph->array[v].head;
 printf("\n Adjacency list of vertex %d\n head ", v);
 while (pCrawl)
 {
 printf("-> %d", pCrawl->dest);
 pCrawl = pCrawl->next;
 }
```

```
 printf("\n");
 }
}
int main()
{
 // create the graph
 int V = 8;
 struct Graph* graph = createGraph(V);
 addEdge(graph, 0, 1);
 addEdge(graph, 0, 2);
 addEdge(graph, 0, 6);
 addEdge(graph, 1, 2);
 addEdge(graph, 1, 3);
 addEdge(graph, 2, 3);
 addEdge(graph, 2, 4);
 addEdge(graph, 3, 4);
 addEdge(graph, 3, 7);
 addEdge(graph, 4, 5);
 addEdge(graph, 4, 6);
 addEdge(graph, 5, 6);
 addEdge(graph, 5, 7);
 // print the adjacency list representation of the above graph
 printGraph(graph);
 return 0;
}
```