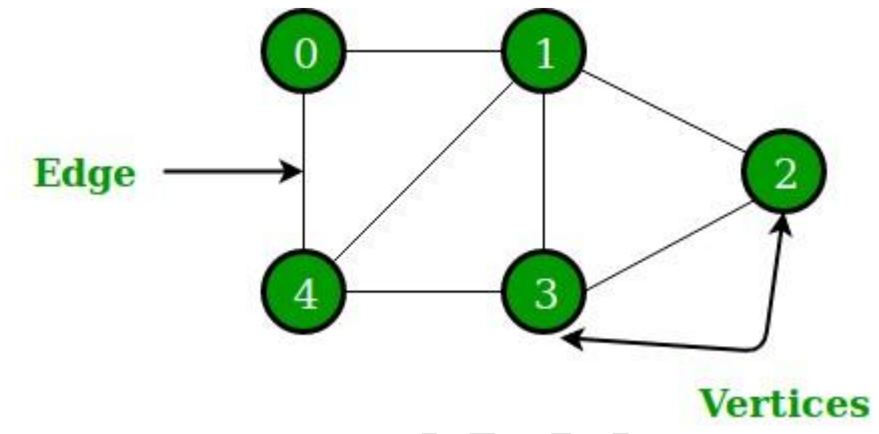
	PUNE INSTITUTE OF COMPUTER TECHNOLOGY PUNE - 411043	
	Department of Electronics & Telecommunication	
	ASSESMENT YEAR: 2021-2022	CLASS: SE-5
	SUBJECT: DATA STRUCTURES	
EXPT No:	LAB Ref: SE/2021-22/	Starting date: 15/11/2021
	Roll No: 22108	Submission date: 22/11/2021
Title:	**Graph**	
Problem statement	Implement Graph using adjacency Matrix with BFS & DFS traversal.	
Prerequisites:	Basics of C programming	
	Data Structures, Queues, Stack	
	Graphs and knowledge of adjacency matrix	
Objectives:	Learn the concepts nonlinear data structure in implementation of graph (Cyclic data structure).	
	Apply BFS traversal to visit graph nodes using queue structure.	
	Apply DFS traversal to visit graph nodes using stack structure	
Theory:		

Graphs:

1. A Graph is a non-linear data structure consisting of nodes and edges.
2. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.
3. More formally a Graph can be defined as, A Graph consists of a finite set



of vertices (or nodes) and set of Edges which connect a pair of nodes.
-In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}

1. Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.

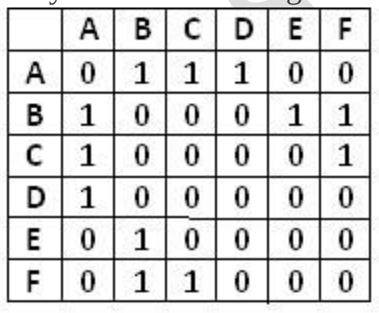
2. Unlike trees, in graphs, a node can have many parents.
3. The link between the nodes may have values or weights.

BFS and DFS:

1. **Breadth First Search** (BFS) and **Depth First Search** (DFS) which uses adjacency list for the graph representation.
2. BFS (Breadth First Search) uses Queue data structure for finding the shortest path. BFS can be used to find single source shortest path in an unweighted graph.
3. DFS (Depth First Search) uses Stack data structure. In DFS, we might traverse through more edges to reach a destination vertex from a source.

Adjacency Matrix:

It is a two-dimensional array with Boolean flags.



	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0

In the given graph, A is connected with B, C and D nodes, so adjacency matrix will have 1s in the 'A' row for the 'B', 'C' and 'D' column.
The advantages of representing the edges using adjacency matrix are:

- Simplicity in implementation as you need a 2-dimensional array
- Creating edges/removing edges is also easy as you need to update the Booleans.

Algorithm	A.Start B.Define structure for stack and queue. C.declare two variables one for stack and other for queue. D.Ask user for number of vertices and then ask for 1 or 0 according to connected nodes. E.Ask user's choice: 1. BFS 2. DFS F.Ask user if wants to continue or not if yes then go to step I else go to step E. G.if (choice==1) 1. Queue structure 2. Declare variable p and i. 3. Enqueue the source vertex and dequeue one element store it in p. 4. If p!=0 is true then display p else go to step 4. 5. If p!=0 is true then make i=1 and go to step else go to step 6. 6. Check whether i<=n and a[p][i]!=0 if this is true then enqueue I and again go to step 5 else dequeue one element and store in p and display p if p is not equal to 0. 7. If i<=n is true then increment I and go to step else again go to step G. H.if (choice==2) 1. Stack data structure 2. Declare variable k and i. 3. Enqueue the source vertex and dequeue one element store it in k. 4. If k!=0 is true then display k else go to step 4. 5. If k!=0 is true then make i=1 and go to step else go to step 6. 6. Check whether i<=n and a[k][i]!=0 if this is true then enqueue I and again go to step 5 else dequeue one element and store in k and display k if k is not equal to 0. 7. If i<=n is true then increment I and go to step else again go to step H. I.End.
ERROR and REMEDY	-

Code	

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

#define MAX 20
int a[MAX][MAX],vis[MAX] ;

typedef struct Queue
{
        int data[MAX];
        int front,rear;
}queue;

typedef struct Stack
{
                                int arr[MAX];
                int top;
}stack;

void initialize(queue *q,stack
*st); int dequeue(queue *); void
enqueue(int item,queue *); void
bfs(int s, int n,queue *); void
dfs(int s, int n,stack *); void
push(int item,stack *); int
pop(stack *);

void end()
{
        printf("End of execution!");
}

void initialize(queue *q,stack *st)
{
        q->front=q->rear=-1;
        st->top=-1;
}

void bfs(int s, int n,queue *q)
{
        int p, i;
enqueue(s,q);   vis[s] =
1;      p =
dequeue(q);     if (p !=
0)
```

```
        {
                printf(" %d", p);
        }
        while (p != 0)
        {
                for (i = 1; i <= n; i++)
                {
                        if ((a[p][i] != 0) && (vis[i] == 0))
                        {
                                enqueue(i,q);
                        vis[i] = 1;
                        }
      }
                p = dequeue(q);
                if (p != 0)
                {
                        printf(" %d", p);
                }
        }

        for (i = 1; i <= n; i++)
        {
                if (vis[i] == 0)
                {
                        bfs(i, n,q);
                }
    }

}

int empty(queue *q)
{ if(q->front==-1)
        {
                return 1;
        }
        else
        {
                return  0;
        }
}


int full(queue *q)
```

```
{
	if(q->rear==MAX-1)
	{
		return 1;
	}
	return 0;
}

void enqueue(int ele,queue *q)
{
	if(q->rear==MAX)
	{
		printf("\nqueue is full");
	}
	else
	{
		if(q->rear==-1)
		{
			q->data[++(q->rear)]=ele;
			q->front++;
		}
		else
		{
			q->data[++(q->rear)]=ele;
		}
	}
}

int dequeue(queue *q)
{
	int k;
	if(q->front > q->rear || q->front==-1)
	{
		return 0;
	}
	else
	{	k=q->data[(q-
	>front)++];	return k;
	}
}

void dfs(int s, int n,stack *st)
{
        int i, k;
        push(s,st);
        vis[s] = 1;
        k = pop(st);
        if (k != 0)
        {
                printf(" %d ", k);
        }
        while (k != 0)
        {
                for (i = 1; i <= n; i++)
                {
                        if ((a[k][i] != 0) && (vis[i] == 0))
                        {
                                push(i,st);
                                vis[i] = 1;
                        }
      }
                k = pop(st);
                if (k != 0)
                {
                        printf(" %d ", k);
                }
        }

        for (i = 1; i <= n; i++)
        {
                if (vis[i] == 0)
                {
                        dfs(i, n,st);
                }
   }
}

void push(int ele,stack *st)
{
        if(st->top==MAX-1)
        {
                printf("\nStack overflow");
        }
        else
        {
                st->arr[++(st->top)]=ele;
        }
}

int pop(stack *st)
{
        if(st->top==-1)
        {
                return 0;
        }
        else
        {
                int ele;
        ele=st->arr[st->top];
        (st->top)--;
        return ele;
  }
}

int main()
{

        queue q;
        stack st;
        initialize(&q,&st);
        int n, i, s, ch, j;
        char c, d;
        printf("\n\nEnter number of vertices:
");     scanf("%d", &n);        printf("\n");
        for (i = 1; i <= n; i++)
        {
                for (j = 1; j <= n; j++)
                {
                        printf("Enter 1 if %d has a node with %d else 0 ", i, j);
                        scanf("%d", &a[i][j]);
                }
        }

        printf("The adjacency matrix is: \n");
        for (i = 1; i <= n; i++)
        {
                for (j = 1; j <= n; j++)
                {
                        printf(" %d ", a[i][j]);
                }

                printf("\n");
        }
```

```
        do {
                                for (i = 1; i <= n; i++)
                        {
                                vis[i] = 0;
           }
        printf("\nMENU");
                printf("\n1.BFS");
        printf("\n2.DFS");
                printf("\nEnter your choice: ");
        scanf("%d", &ch);
                printf("\nEnter source vertex: :");
        scanf(" %d", &s);


                        switch (ch)
                        {
                                case 1:
                                        bfs(s, n, &q);
                                break;
                case 2:
                                                dfs(s, n, &st);
                                        break;
                        }

                printf("\nDo you want to continue(Y/N) ? ");
                scanf("%c", &d);                scanf("%c",
&c);
        } while ((c == 'y') || (c == 'Y'));
end();
                return 0;
}
```

Output	
	```
	Enter the number of vertices: 3

	Enter 1 if 1 has a node with 1 else 0 :1
	Enter 1 if 1 has a node with 2 else 0 :1
	Enter 1 if 1 has a node with 3 else 0 :0
	Enter 1 if 2 has a node with 1 else 0 :1
	Enter 1 if 2 has a node with 2 else 0 :0
	Enter 1 if 2 has a node with 3 else 0 :1
	Enter 1 if 3 has a node with 1 else 0 :0
	Enter 1 if 3 has a node with 2 else 0 :1
	Enter 1 if 3 has a node with 3 else 0 :1

	The adjacency matrix is:
	 1  1  0
	 1  0  1
	 0  1  1

	MENU
	```

```
1.BFS
2.DFS
ENTER YOUR CHOICE1

Enter the source vertex: 2
 2 1 3
DO U WANT TO CONTINUE(Y/N) ? y

MENU
1.BFS
2.DFS
ENTER YOUR CHOICE2

Enter the source vertex: 2
 2  3  1
DO U WANT TO CONTINUE(Y/N) ? n
End of Execution!
```

CONCLUSION:

1.Successfully implemented BFS and DFS traversal using adjacency matrix.

2.Used concept of structure of stack and queue for DFS and BFS ,respectively.

3.Also,we learned that BFS is more suitable for searching vertices which are closer to the given source and DFS is more suitable when there are solutions away from source

REFERENCES:

Seymour Lipschutz, Data Structure with C, Schaum's Outlines, Tata McGrawHill

E Balgurusamy - Programming in ANSI C, Tata McGraw-Hill (Third Edition)

Yashavant Kanetkar- Let Us C, BPB Publication, 8th Edition.

Continuous Assessment for DS AY 2021-22			
RPP (5)	**SPO (5)**	**Total (10)**	**Signature:**
			Assessed By: Mr. V. B. Vaijapurkar
Start date	**Submission date**		**Date:**

15/11/2021	**22/11/2021**	**Roll. No.22108**
***Regularity, Punctuality, performance** ***Submission, Presentation, orals**		