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

int sz=0;//stack size
int n=4;
int v=4;
int count,curr=-1,array[50],visited[4];
int graph[4][4] = {{0, 1, 1, 1},
                                    {1, 0, 0, 1},
                                    {1, 0, 0, 0},
                                    {1, 1, 0, 0}};

void push(int val) {
        curr++;
        array[curr] = val;
}
int pop() {
        int val;
        if (curr > -1) {
                int val=array[curr];
            curr--;
            // printf("Top is poped");
         return val;

        }
        else
                printf("stack Empty\n");
        return -1;
}

// queue
int que[50] ,front=-1 ,rear=-1 ;
void enqueue(int que[] ,int* front ,int* rear ,int item){
        if( *front ==-1 )
                *front=*front+1;
        *rear=*rear+1;
        que[*rear]=item;

        //printf("**%d,%d**",*front,*rear);
}

void dequeue(int que[] ,int* front ,int* rear ){
        int item;
        if( *front==-1 || *rear==-1 )
            printf("Empty Queue\n");
        else{
        item=que[*front];
                if( *front==*rear ){
                        *rear=-1;
                        *front=-1;
```

```
                }
                else
                        *front=*front+1;
        }
}

void printvis(int arr[]){
        printf("vis = ");
        for(int i=0;i<v;i++)
                printf("%d\t", arr[i]);
        //printf("\n");
}

void bfs(int start){
    int f;
    visited[start]=1;
    enqueue(que,&front,&rear,start);

    while(front!=-1){
        f=que[front];
        for(int i=0;i<v;i++){
            if (graph[f][i] == 1 && (visited[i]==0)) {
                // printvis(visited);
                visited[i]= 1;
                // printvis(visited);
                // printf("%d\n", i);
                enqueue(que,&front,&rear,i);
            }
        }
        printf(" ->%d",f);
        dequeue(que,&front,&rear);
    }

}

void dfs(int start){

    printf(" ->%d",start);
    visited[start]= 1;     //curr = top
    for(int i=0;i<v;i++){
        if ((graph[start][i]==1) && (visited[i]==0)) {
            dfs(i);
        }
    }
}

void displayMatrix(){
    printf("\tAdjcency Matrix\n");
    for(int i=0;i<v;i++){
        for(int j=0;j<v;j++){
```

```
            printf("%d\t",graph[i][j]);
        }
        printf("\n");
    }
}
void displayadjlist(){
    printf("\tAdjcency List\n");
    for(int i=0;i<v;i++){
        printf("%d :",i);
        for(int j=0;j<v;j++){
            if(graph[i][j]==1)
                printf(" ->%d",j);
        }
        printf("\n");
    }
}

void setvisitedtofalse(){
    for(int i=0;i<v;i++)
        visited[i]=0;
}


int main(){

    // int visited[4];
    int choice;
    int start;

    // displayadjlist();
    printf("1...display\n");
    printf("2...dfs\n");
    printf("3...bfs\n");
    printf("4...quit\n");

    int quit=1;
    while(quit!=0){
        setvisitedtofalse();
        printf("\nOption : ");
        scanf("%d",&choice);
        switch(choice){
                case 1: displayadjlist();
                        break;
                case 2:printf("enter the start element: ");
                    scanf("%d",&start);
                    printf("dfs[%d]",start);
                    dfs(start);
                        break;
                case 3:printf("enter the start element: ");
                    scanf("%d",&start);
```

```
                    printf("bfs[%d]",start);
                    bfs(start);
                        break;
                case 4: quit=0;
                break;
            default:
                printf("\n1...display\n");
                printf("2...dfs\n");
                printf("3...bfs\n");
                printf("4...quit\n");
        }
    }
    return 0;
}
```

```
> ./main
1...display
2...dfs
3...bfs
4...quit

Option : 2
enter the start element: 1
dfs[1] ->1 ->0 ->2 ->3
Option : 3
enter the start element: 1
bfs[1] ->1 ->0 ->3 ->2
Option : 1
    Adjcency List
0 : ->1 ->2 ->3
1 : ->0 ->3
2 : ->0
3 : ->0 ->1

Option : 2
enter the start element: 0
dfs[0] ->0 ->1 ->3 ->2
Option : 3
enter the start element: 0
bfs[0] ->0 ->1 ->2 ->3
Option : 1
    Adjcency List
0 : ->1 ->2 ->3
1 : ->0 ->3
2 : ->0
3 : ->0 ->1
```

Dfs _ n _ Bfs

Algorithm

I. create a graph using adj. List or Mat

II. void setvisitedtofalse ()

1. START
2. for(int i = 0 to total vertices)
    1. visited[i] = false
3. End for
4. STOP

III void dispadjlist()

1. START
2. for(int i = 0 to tot vertices)
    1. print("i + ": ")
    2. for(int j = 0 to tot vertices)
        1. if(graph[i][j] != 0)
            1. print("→" + j)
        2. End if
    3. End for
3. End for
4. STOP

IV void dfs (int start)

1. START
2. print (" → " + start)
3. visited [start] = True ;
4. for (int i = 0 to tot vertices)
    1. if (graph [start] [i] == 1 && visited [i] == False)
        1. dfs (i)
    2. End If
5. End for
6. STOP

V void bfs (int start)

1. START
2. visited [start] = True
3. enqueue (start)
4. int f ;
5. while (top != -1)
    1. f = queue [top]
    2. for (int i = 0 to total vertices)
        1. if (graph [f] [i] == 1 && visited [i] == False)
            1. visited [i] = True
            2. enqueue (i),
        2. End If
    3. End for

4. print (" → " + f)
5. dequeue ();
6. End While
7. STOP.

III int main ()

1. START
2. give the user menu of choices to display, dfs & bfs.
3. while (True) {
    1. ~~vis~~ setVisitedToFalse ();
    2. using switch create the menu and call display, dfs & bfs
4. End } While
5. STOP.