

Graph traversal BFS

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

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

```
#define MAX 100
```

```
#define initial 1
```

```
#define waiting 2
```

```
#define visited 3
```

```
int n; /*Number of vertices in the  
graph*/
```

```
int adj[MAX][MAX]; /*Adjacency  
Matrix*/
```

```
int state[MAX]; /*can be initial,  
waiting or visited*/
```

```
void create_graph();
```

```
void BF_Traversal();
```

```
void BFS(int v);
```

```
int queue[MAX], front = -1, rear = -1;
```

```
void insert_queue(int vertex);
```

```
int delete_queue();
```

```
int isEmpty_queue();
```

```
int main()
```

```
{
```

```
    create_graph();
```

```
    BF_Traversal();
```

```
    return 0;
```

```
}/*End of main()*/
```

```
void BF_Traversal()
```

```
{
```

```
int v;
```

```
for(v=0; v<n; v++)
```

```
state[v] = initial;
```

```
printf("\nEnter starting vertex for  
Breadth First Search : ");
```

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

```
BFS(v);
```

```
}/*End of BF_Traversal()*/
```

```
void BFS(int v)
```

```
{
```

```
int i;
```

```
insert_queue(v);
```

```
state[v] = waiting;
```

```
while(!isEmpty_queue())
```

```
{  
  
v = delete_queue( );  
  
printf("%d ",v);  
  
state[v] = visited;  
  
for(i=0; i<n; i++)  
{  
  
/*Check for adjacent unvisited  
vertices */  
  
if(adj[v][i] == 1 && state[i] ==  
initial)  
  
{  
  
insert_queue(i);  
  
state[i] = waiting;  
  
}  
  
}  
  
}  
  
printf("\n");  
  
}/*End of BFS()*/
```

```
void insert_queue(int vertex)

{

    if(rear == MAX-1)

        printf("\nQueue Overflow\n");

    else

    {

        if(front == -1) /*If queue is initially
        empty */

            front = 0;

            rear = rear+1;

            queue[rear] = vertex ;

        }

    }/*End of insert_queue()*/

int isEmpty_queue()

{

    if(front == -1 || front > rear)

        return 1;
```

```
else  
  
return 0;  
  
}/*End of isEmpty_queue()*/
```

```
int delete_queue()  
{  
  
int del_item;  
  
if(front == -1 || front > rear)  
{  
  
printf("\nQueue Underflow\n");  
  
exit(1);  
  
}
```

```
  
del_item = queue[front];  
  
front = front+1;  
  
return del_item;  
  
}/*End of delete_queue() */
```

```
void create_graph()

{

    int i,max_edges,origin,destin;


    printf("\nEnter number of vertices
: ");

    scanf("%d",&n);

    max_edges = n*(n-1);


    for(i=1; i<=max_edges; i++)

    {

        printf("\nEnter edge %d( -1 -1 to
quit ) : ",i);

        scanf("%d %d",&origin,&destin);


        if((origin == -1) && (destin == -1))

            break;

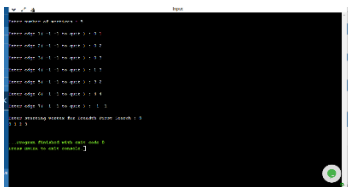

        if(origin>=n || destin>=n ||
origin<0 || destin<0)
```

```
{
    printf("\nInvalid edge!\n");
    i--;
}

else
{
    adj[origin][destin] = 1;
}

}

}
```



Graph traversal DFS:

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

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



```
#define MAX 100
```

```
#define initial 1
```

```
#define visited 2
```

```
int n; /* Number of nodes in the  
graph */
```

```
int adj[MAX][MAX]; /*Adjacency  
Matrix*/
```

```
int state[MAX]; /*Can be initial or  
visited */
```

```
void DF_Traversal();
```

```
void DFS(int v);
```

```
void create_graph();
```

```
int stack[MAX];
```

```
int top = -1;
```

```
void push(int v);
```

```
int pop();
```

```
int isEmpty_stack();
```

```
main()
```

```
{
```

```
    create_graph();
```

```
    DF_Traversal();
```

```
}/*End of main()*/
```

```
void DF_Traversal()
```

```
{
```

```
    int v;
```

```
    for(v=0; v<n; v++)
```

```
        state[v]=initial;
```

```
    printf("\nEnter starting node  
for Depth First Search : ");
```

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

```
    DFS(v);
```

```
        printf("\n");

    }/*End of DF_Traversal( )*/

void DFS(int v)
{
    int i;

    push(v);

    while(!isEmpty_stack())
    {
        v = pop();

        if(state[v]==initial)
        {
            printf("%d ",v);

            state[v]=visited;
        }

        for(i=n-1; i>=0; i--)
        {
            if(adj[v][i]==1 &&
state[i]==initial)
```

```
        push(i);
    }
}
}/*End of DFS( )*/
```

```
void push(int v)
{
    if(top == (MAX-1))
    {
        printf("\nStack
Overflow\n");
        return;
    }
    top=top+1;
    stack[top] = v;

}/*End of push()*/
```

```
int pop()
```

```
{  
    int v;  
    if(top == -1)  
    {  
        printf("\nStack  
Underflow\n");  
        exit(1);  
    }  
    else  
    {  
        v = stack[top];  
        top=top-1;  
        return v;  
    }  
}/*End of pop()*/
```

```
int isEmpty_stack( )  
{  
    if(top == -1)
```

```

        return 1;

    else

        return 0;

}/*End if isEmpty_stack()*/

void create_graph()
{
    int i,max_edges,origin,destin;

    printf("\nEnter number of
nodes : ");

    scanf("%d",&n);

    max_edges=n*(n-1);

    for(i=1;i<=max_edges;i++)
    {

        printf("\nEnter edge %d( -
1 -1 to quit ) : ",i);

        scanf("%d
%d",&origin,&destin);

```

```
        if( (origin == -1) && (destin  
== -1) )
```

```
            break;
```

```
        if( origin >= n || destin >=  
n || origin<0 || destin<0)
```

```
        {
```

```
            printf("\nInvalid  
edge!\n");
```

```
            i--;
```

```
        }
```

```
        else
```

```
        {
```

```
            adj[origin][destin] = 1;
```

```
        }
```

```
    }
```

```
}
```

```
graph TD; 1((1)) --> 2((2)); 1 --> 3((3)); 2 --> 3; 3 --> 4((4)); 4 --> 5((5)); 5 --> 6((6)); 6 --> 7((7)); 7 --> 8((8)); 8 --> 9((9)); 9 --> 10((10)); 10 --> 11((11)); 11 --> 12((12)); 12 --> 13((13)); 13 --> 14((14)); 14 --> 15((15)); 15 --> 16((16)); 16 --> 17((17)); 17 --> 18((18)); 18 --> 19((19)); 19 --> 20((20)); 20 --> 21((21)); 21 --> 22((22)); 22 --> 23((23)); 23 --> 24((24)); 24 --> 25((25)); 25 --> 26((26)); 26 --> 27((27)); 27 --> 28((28)); 28 --> 29((29)); 29 --> 30((30)); 30 --> 31((31)); 31 --> 32((32)); 32 --> 33((33)); 33 --> 34((34)); 34 --> 35((35)); 35 --> 36((36)); 36 --> 37((37)); 37 --> 38((38)); 38 --> 39((39)); 39 --> 40((40)); 40 --> 41((41)); 41 --> 42((42)); 42 --> 43((43)); 43 --> 44((44)); 44 --> 45((45)); 45 --> 46((46)); 46 --> 47((47)); 47 --> 48((48)); 48 --> 49((49)); 49 --> 50((50)); 50 --> 51((51)); 51 --> 52((52)); 52 --> 53((53)); 53 --> 54((54)); 54 --> 55((55)); 55 --> 56((56)); 56 --> 57((57)); 57 --> 58((58)); 58 --> 59((59)); 59 --> 60((60)); 60 --> 61((61)); 61 --> 62((62)); 62 --> 63((63)); 63 --> 64((64)); 64 --> 65((65)); 65 --> 66((66)); 66 --> 67((67)); 67 --> 68((68)); 68 --> 69((69)); 69 --> 70((70)); 70 --> 71((71)); 71 --> 72((72)); 72 --> 73((73)); 73 --> 74((74)); 74 --> 75((75)); 75 --> 76((76)); 76 --> 77((77)); 77 --> 78((78)); 78 --> 79((79)); 79 --> 80((80)); 80 --> 81((81)); 81 --> 82((82)); 82 --> 83((83)); 83 --> 84((84)); 84 --> 85((85)); 85 --> 86((86)); 86 --> 87((87)); 87 --> 88((88)); 88 --> 89((89)); 89 --> 90((90)); 90 --> 91((91)); 91 --> 92((92)); 92 --> 93((93)); 93 --> 94((94)); 94 --> 95((95)); 95 --> 96((96)); 96 --> 97((97)); 97 --> 98((98)); 98 --> 99((99)); 99 --> 100((100));
```

Topological sorting:

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

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

```
#define MAX 100
```

```
int n; /*Number of vertices in the graph*/
```

```
int adj[MAX][MAX]; /*Adjacency Matrix*/
```

```
void create_graph();
```



```
int queue[MAX], front = -1, rear = -1;
```

```
void insert_queue(int v);
```

```
int delete_queue();
```

```
int isEmpty_queue();
```

```
int indegree(int v);
```

```
int main()
```

```
{
```

```
    int
```

```
    i,v,count,topo_order[MAX],indeg[  
    MAX];
```

```
    create_graph();
```

```
    /*Find the indegree of each  
    vertex*/
```

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

```
    {
```

```

        indeg[i] = indegree(i);

        if( indeg[i] == 0 )

            insert_queue(i);

    }

    count = 0;

    while( !isEmpty_queue( ) &&
count < n )

    {

        v = delete_queue();

        topo_order[++count] = v;
/*Add vertex v to topo_order
array*/

        /*Delete all edges going
from vertex v */

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

        {

            if(adj[v][i] == 1)

            {

```

```

adj[v][i] = 0;

indeg[i] =

indeg[i]-1;

if(indeg[i] == 0)

insert_queue(i);

    }

    }

}

if( count < n )

{

    printf("\nNo topological
ordering possible, graph contains
cycle\n");

    exit(1);

}

printf("\nVertices in
topological order are :\n");

for(i=1; i<=count; i++)

```

```

        printf( "%d ",topo_order[i]
    );

    printf("\n");

    return 0;

}/*End of main()*/

void insert_queue(int vertex)
{
    if (rear == MAX-1)

        printf("\nQueue
Overflow\n");

    else

    {

        if (front == -1) /*If queue
is initially empty */

            front = 0;

            rear = rear+1;

            queue[rear] = vertex ;

    }

```

```
 }/*End of insert_queue()*/
```

```
int isEmpty_queue()
```

```
{
```

```
    if(front == -1 || front > rear )
```

```
        return 1;
```

```
    else
```

```
        return 0;
```

```
 }/*End of isEmpty_queue()*/
```

```
int delete_queue()
```

```
{
```

```
    int del_item;
```

```
    if (front == -1 || front > rear)
```

```
    {
```

```
        printf("\nQueue  
Underflow\n");
```

```
        exit(1);
```

```
    }
```

```
        else
        {
            del_item = queue[front];
            front = front+1;
            return del_item;
        }
    }/*End of delete_queue() */
```

```
int indegree(int v)
{
    int i,in_deg = 0;
    for(i=0; i<n; i++)
        if(adj[i][v] == 1)
            in_deg++;
    return in_deg;
}/*End of indegree() */
```

```
void create_graph()
```

```
{  
  
    int i,max_edges,origin,destin;  
  
    printf("\nEnter number of  
vertices : ");  
  
    scanf("%d",&n);  
  
    max_edges = n*(n-1);  
  
    for(i=1; i<=max_edges; i++)  
    {  
  
        printf("\nEnter edge %d(-1  
-1 to quit): ",i);  
  
        scanf("%d  
%d",&origin,&destin);  
  
        if((origin == -1) && (destin  
== -1))  
  
            break;
```

```

        if( origin >= n || destin >=
n || origin<0 || destin<0)

        {

            printf("\nInvalid
edge!\n");

            i--;

        }

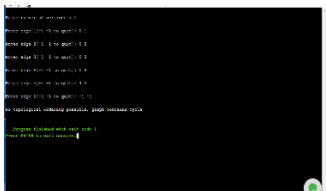
        else

            adj[origin][destin] = 1;

    }

}

```



```

C++14
Adjacency list vertices: 10
Enter edge (1-10) its weight: 10
Enter edge (2-1) its weight: 2
Enter edge (3-1) its weight: 2
Enter edge (3-2) its weight: 2
Enter edge (4-3) its weight: 4
Enter edge (4-5) its weight: 4
Enter edge (5-6) its weight: 5
Enter edge (6-7) its weight: 10
no topological ordering possible, graph contains cycle
...
Program finished with exit code: 0
Press ENTER to exit console.

```