## Aim: - Traversal of Graph using BFS and DFS

**Theory: -** Graph traversal means visiting every vertex and edge exactly once in a well-defined order. While using certain graph algorithms, you must ensure that each vertex of the graph is visited exactly once. The order in which the vertices are visited are important and may depend upon the algorithm.

During a traversal, it is important that you track which vertices have been visited. The most common way of tracking vertices is to mark them.

**Breadth First Search (BFS)**

There are many ways to traverse graphs. BFS is the most commonly used approach.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

**Applications of BFS**

**1. Shortest path in a graph.2. Web Crawler 3. Broadcasting in a network.**

**4. Social Network 5. Cycle detection in an undirected path**

**Depth First Search** (DFS)

 DFS starts the traversal from the root node and explore the search as far as possible from the root node i.e. depth wise.

This algorithm works in two stages – in the first stage the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped-off.

**Applications of DFS**  
> Topological sorting  
> In Connectivity testing  
> Finding a path between V and W in the graph.

**Program:-**

#include<stdio.h>

#include<stdlib.h>

#define MAX -32678

//this structure is for head pointers

typedef struct node

{

int data;

struct node\* next;

}node;

//this is stack of visited elements in DFS

typedef struct stack

{

struct node\* pointer;

struct stack \*next;

}stack;

//this is to check if element visited.

typedef struct checkarray

{

int value;

int boole;

}checkarray;

//this is to implement traversal in BFS

typedef struct queue

{

int data;

struct queue \*next;

}queue;

//initializing head pointers of adjacency list

void initialise(node list[],int index,int ver)

{

list[index].data=ver;

list[index].next=NULL;

}

//inserting vertices according to their relation.

int insert(node list[],int a,int b,int v)

{

int index=0,ind=0;

while(list[index].data!=a && index!=v)

{

index++;

}

while(list[ind].data!=b && ind!=v)

{

ind++;

}

if(index!=v && ind!=v)

{

node\* p=(node\*)malloc(sizeof(node));

p->data=b;

p->next=NULL;

if(list[index].next==NULL)

list[index].next=p;

else

{

struct node\* temp=list[index].next;

list[index].next=p;

p->next=temp;

}

return 1;

}

else

{

printf("No such vertex found\n");

return 0;

}

}

//displaying graph as adjacency list.

void display(node list[],int v)

{

printf("displaying adjacency list\n");

for(int i=0;i<v;i++)

{

printf("%d->",list[i].data);

node\* temp=list[i].next;

while(temp!=NULL)

{

printf("%d,",temp->data);

temp=temp->next;

}

printf("\n");

}

}

//this function checks whether vertex present in list or not,returns 1 if present else 0

int isVertex(node list[],int v,int el)

{

for(int i=0;i<v;i++)

{

if(list[i].data==el)

return 1;

}

return 0;

}

//enqueeing the element in queue(BFS)

void enqueue(queue \*\*rear,queue \*\*front,int val)

{

queue \*p=(queue\*)malloc(sizeof(queue));

p->data=val;

p->next=NULL;

if((\*front)==NULL)

{

\*rear=p;

\*front=p;

}

else

{

(\*rear)->next=p;

(\*rear)=p;

}

}

//dequeeing element in queue(BFS)

int dequeue(queue \*\*rear,queue \*\*front)

{

if((\*front)==NULL)

{

printf("List is empty");

return 0;

}

else

{

queue \*p;

p=\*front;

printf("%d\t",p->data);

int a=p->data;

(\*front)=p->next;

free(p);

return a;

}

}

//depth first traversal returns address

node\* address(node list[],int v,int el)

{

int i;

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

if(list[i].data==el)

return &list[i];

}

//push for stack

void push(stack \*\*head,node \*nodeAdd)

{

stack \*p=(stack\*)malloc(sizeof(stack));

p->pointer=nodeAdd;

p->next=\*head;

\*head=p;

printf("%d\t",nodeAdd->data);

}

//pop function for stack

void pop(stack \*\*head)

{

stack \*temp=\*head;

if(head==NULL)

printf("stack is empty\n");

else{

\*head=temp->next;

free(temp);

}

}

//returns top of stack

node\* top(stack \*head)

{

if(head==NULL)

return NULL;

else

return head->pointer;

}

//elements in checkarray are inserted by hashing and linearprobing

void LinearProbingInsert(checkarray answer[],int n,int a,int key)

{

if(answer[key].value==MAX)

answer[key].value=a;

else

{

int i=++key;

while(i!=n && answer[i].value!=MAX)

++i;

if(i!=n)

answer[i].value=a;

else

{

i=0;

while(i!=key && answer[i].value!=MAX)

++i;

answer[i].value=a;

}

}

}

//search result of linearprobing

int LinearSearch(checkarray answer[],int v,int a)

{

int key=a%v,i;

if(answer[key].value==a)

return key;

else

{

for(i=key++;i<v;i++)

if(answer[i].value==a)

return i;

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

if(answer[i].value==a)

return i;

}

}

//initializing array of vertex to zero

void initArray(node list[],int v,checkarray answer[])

{

int a,key;

for(int i=0;i<v;i++)

{

answer[i].value=MAX;

answer[i].boole=0;

}

for(int i=0;i<v;i++)

{

a=list[i].data;

key=a%v;

LinearProbingInsert(answer,v,a,key);

}

}

//function to know whether node is visited

int isvisited(checkarray answer[],int el,int v)

{

int i=LinearSearch(answer,v,el);

if(answer[i].boole==1)

return 1;

else 0;

}

//function that marks node visited

void visited(checkarray answer[],int v,int el)

{

int i=LinearSearch(answer,v,el);

answer[i].boole=1;

}

//this is depth first search function

void depthFirstSearch(node list[],int v,int el)

{

checkarray answer[v];

initArray(list,v,answer);

stack \*head=NULL;

push(&head,address(list,v,el));

visited(answer,v,el);

node\* p=address(list,v,el);

while(head!=NULL)

{

p=p->next;

if(p!=NULL && isvisited(answer,p->data,v)==0)

{

el=p->data;

push(&head,address(list,v,el));

visited(answer,v,el);

p=address(list,v,el);

}

else

{

if(p==NULL)

{

pop(&head);

p=top(head);

}

else

{

while(p!=NULL&& isvisited(answer,p->data,v)==1)

p=p->next;

if(p!=NULL)

{

el=p->data;

push(&head,address(list,v,el));

visited(answer,v,el);

p=address(list,v,el);

}

else

{

pop(&head);

p=top(head);

}

}

}

}

}

//this is f breadth first search traversal

void breadthFirstTraverse(node list[],int v,int el)

{

checkarray answer[v];

initArray(list,v,answer);//this is using list

queue \*rear=NULL,\*front=NULL;

enqueue(&rear,&front,el);

visited(answer,v,el);

int i,e=el;

node \*p;

while(front!=NULL)

{

e=dequeue(&rear,&front);

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

{

if(list[i].data==e)

break;

}

node \*p=list[i].next;

while(p!=NULL)

{

int a=p->data;

if(isvisited(answer,a,v)==0)

{

enqueue(&rear,&front,a);

visited(answer,v,a);

p=p->next;

}

else

p=p->next;

}

}

}

//main function

void main()

{

printf("Enter number of vertices in your graph\n");

int e,v;

scanf("%d",&v);

printf("Enter number of edges in your graph\n");

scanf("%d",&e);

node list[v];

int ver,k=0;

printf("enter vertices \n");

while(k!=v)

{

printf("enter vertex %d\n",k+1);

scanf("%d",&ver);

initialise(list,k,ver);

k++;

}

int choice;

printf("Enter 1 if your graph is directed\n");

scanf("%d",&choice);

int a,b,f;

if(choice==1)

{

int count=0;

printf("your graph is directed\n");

while(count!=e)

{

printf("enter element 1 of realtion ");

scanf("%d",&a);

printf("enter element 2 of realtion ");

scanf("%d",&b);

f=insert(list,a,b,v);

printf("\n\n");

if(f==1)

count++;

}

}

else

{

int count=0;

printf("your graph is undirected\n");

while(count!=e)

{

printf("enter element 1 of realtion\t");

scanf("%d",&a);

printf("enter element 2 of realtion\t");

scanf("%d",&b);

f=insert(list,a,b,v);

f=insert(list,b,a,v);

printf("\n\n");

if(f==1)

count++;

}

}

display(list,v);

int el,ch=1,chh=1,choicee=1;

while(choicee!=3)

{

printf("enter 1 for BFS or 2 for DFS traversals 3. exit\n");

scanf("%d",&choicee);

switch(choicee)

{

case 1: while(chh==1)

{

printf("enter the vertex from which BFS traversal to be started\n");

scanf("%d",&el);

if(isVertex(list,v,el))

{

breadthFirstTraverse(list,v,el);

printf("if you want to do BFS for another vertex enter '1'\n");

scanf("%d",&chh);

printf("\n");

}

else

{

printf("entered element is not a vertex in graph list\n");

printf("if you want to do BFS for valid vertex enter 1\n");

scanf("%d",&chh);

}

}

break;

case 2:

while(ch==1)

{

printf("enter the vertex from which DFS traversal to be started\n");

scanf("%d",&el);

if(isVertex(list,v,el))

{

depthFirstSearch(list,v,el);

printf("if you want to do DFS for another vertex enter '1'\n");

scanf("%d",&ch);

printf("\n");

}

else

{

printf("entered element is not a vertex in graph list\n");

printf("if you want to do DFS for valid vertex enter 1\n");

scanf("%d",&ch);

}

}

break;

default :

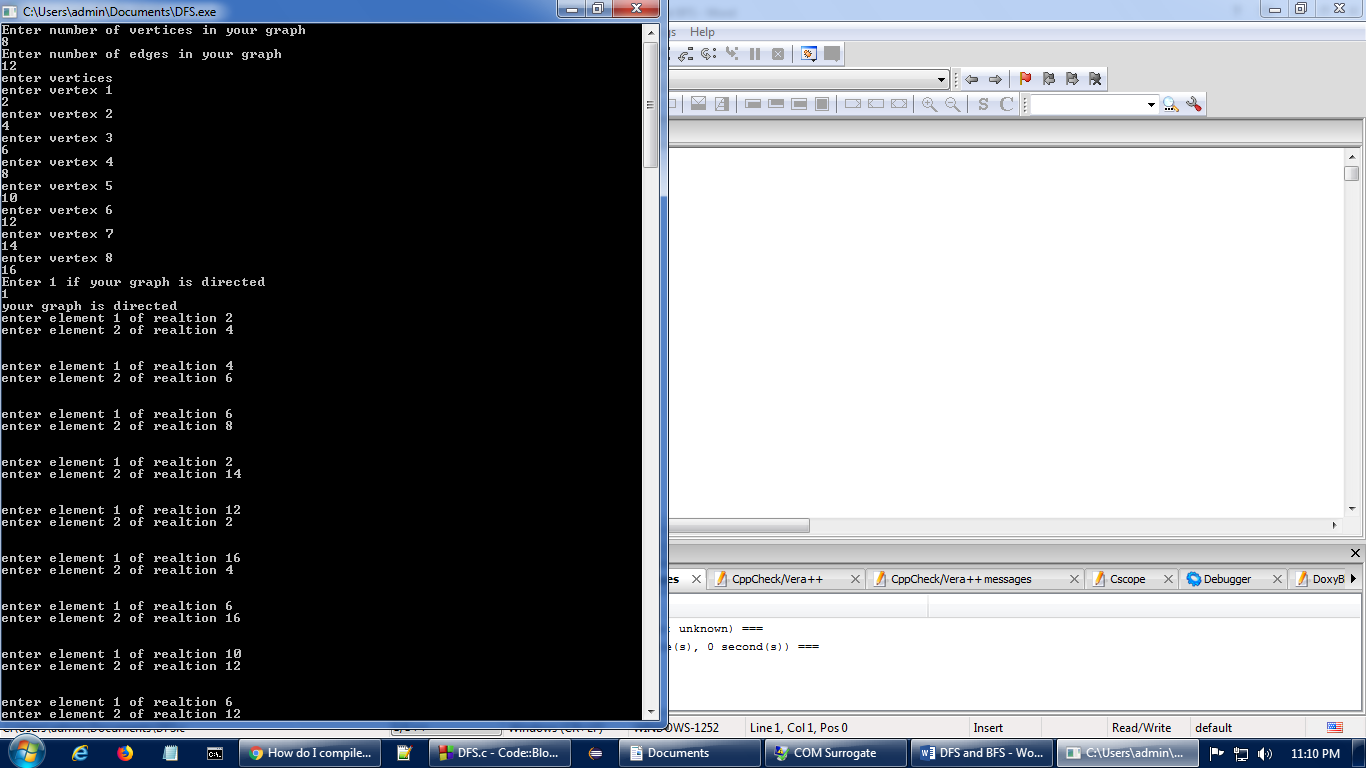
printf("you did not enter proper choice \n");

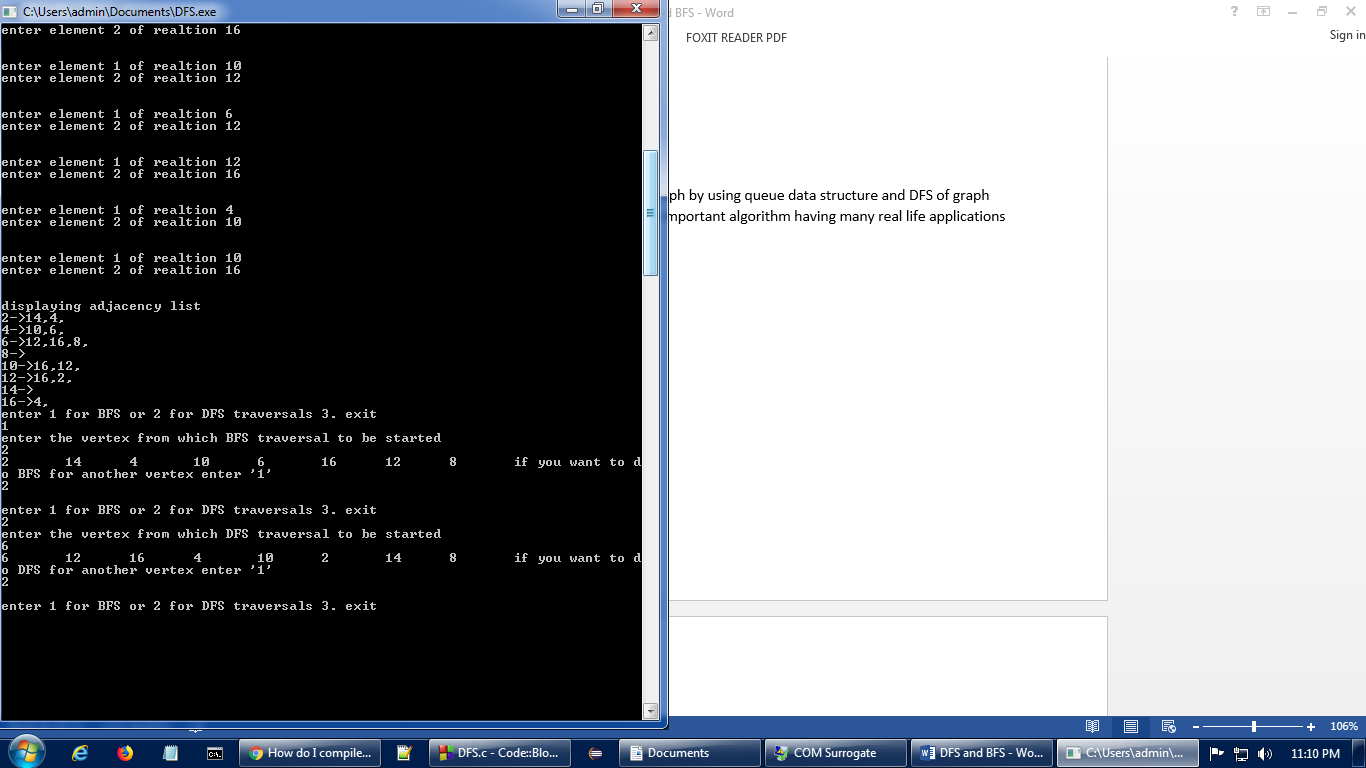
}

}

}

**Output:-**





**Conclusion:**-This experiment I performed BFS of graph by using queue data structure and DFS of graph using STACK by iterative method. DFS and BFS are important algorithm having many real life applications as mentioned above.