**EXPERIMENT NO 9**

~ ARYA RAUL

COMPS 20

**Experiment No. 9: Depth First Search and Breath First Search**

**Aim :** **Implementation of DFS and BFS traversal of graph**.

**Objective:**

1. Understand the Graph data structure and its basic operations.
2. Understand the method of representing a graph.
3. Understand the method of constructing the Graph ADT and defining its operations

**Theory:**

A graph is a collection of nodes or vertex, connected in pairs by lines referred as edges. A graph can be directed or undirected graph.

One method of traversing through nodes is depth first search. Here we traverse from starting node and proceeds from top to bottom. At a moment we reach a dead end from where the further movement is not possible and we backtrack and then proceed according to left right order. A stack is used to keep track of a visited node which helps in backtracking.



**DFS Traversal –0 1 2 3 4**

**Algorithm**

Algorithm: DFS\_LL(V)

Input: V is a starting vertex

Output : A list VISIT giving order of visited vertices during traversal.

Description: linked structure of graph with gptr as pointer

1. if gptr = NULL then

print “Graph is empty” exit

1. u=v
2. OPEN.PUSH(u)
3. while OPEN.TOP !=NULL do

u=OPEN.POP()

if search(VISIT,u) = FALSE then

INSERT\_END(VISIT,u)

Ptr = gptr(u)

While ptr.LINK != NULL do

Vptr = ptr.LINK

OPEN.PUSH(vptr.LABEL)

End while

End if

       End while

1. Return VISIT
2. Stop

**BFS Traversal**

**BFS Traversal – 0 1 4 2 3**

**Algorithm**

Algorithm: DFS()

i=0

count=1

visited[i]=1

print("Visited vertex  i")

repeat this till queue is empty or all nodes visited

repeat this for all nodes from first till last

if(g[i][j]!=0&&visited[j]!=1)

{

push(j)

}

i=pop()

print("Visited vertex  i")

visited[i]=1

count++

Algorithm: BFS()

i=0

count=1

visited[i]=1

print("Visited vertex  i")

repeat this till queue is empty or all nodes visited

repeat this for all nodes from first till last

if(g[i][j]!=0&&visited[j]!=1)

{

enqueue(j)

}

i=dequeue()

print("Visited vertex  i")

visited[i]=1

count++

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*leftChild, \*rightChild;

};

struct node \*root = NULL;

struct node \*newNode(int item){

struct node \*temp = (struct node \*)malloc(sizeof(struct node));

temp->data = item;

temp->leftChild = temp->rightChild = NULL;

return temp;

}

void insert(int data){

struct node \*tempNode = (struct node\*) malloc(sizeof(struct node));

struct node \*current;

struct node \*parent;

tempNode->data = data;

tempNode->leftChild = NULL;

tempNode->rightChild = NULL;

//if tree is empty

if(root == NULL) {

root = tempNode;

} else {

current = root;

parent = NULL;

while(1) {

parent = current;

//go to left of the tree

if(data < parent->data) {

current = current->leftChild;

//insert to the left

if(current == NULL) {

parent->leftChild = tempNode;

return;

}

}//go to right of the tree

else {

current = current->rightChild;

if(current == NULL) {

parent->rightChild = tempNode;

return;

}

}

}

}

}

struct node\* search(int data){

struct node \*current = root;

printf("\nVisiting elements: ");

while(current->data != data) {

if(current != NULL) {

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

//go to left tree

if(current->data > data) {

current = current->leftChild;

}//else go to right tree

else {

current = current->rightChild;

}

//not found

if(current == NULL) {

return NULL;

}

}

}

return current;

}

void printTree(struct node\* Node){

if(Node == NULL)

return;

printTree(Node->leftChild);

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

printTree(Node->rightChild);

}

int main(){

insert(10);

insert(14);

insert(19);

insert(26);

insert(27);

insert(31);

insert(33);

insert(35);

insert(42);

insert(44);

printf("Insertion done\n");

printTree(root);

struct node\* k;

k = search(35);

if(k != NULL)

printf("\nElement %d found", k->data);

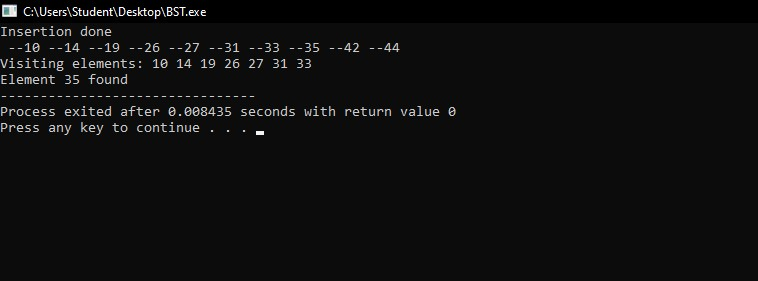
else

printf("\nElement not found");

return 0;

}

**Output:**



**Conclusion:**

the implementations of Depth-First Search (DFS) and Breadth-First Search (BFS) traversal algorithms are fundamental tools in graph analysis. DFS is excellent for finding paths and connected components, while BFS is ideal for finding the shortest path. Both have broad applications, and the choice between them depends on the problem's nature. These algorithms are essential in computer science and of