NAME: GAURAV KISHOR PATIL

DIV: 2 ROLL NO:54

BATCH: C

Experiment No.9
Implementation of Graph traversal techniques - Depth First Search, Breadth First Search
Date of Performance:
Date of Submission:

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

Aim : Implementation of DFS and BFS traversal of graph.

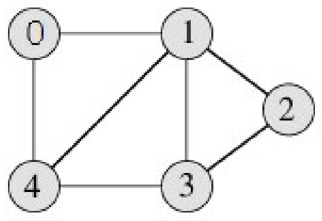
Objective:

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.



$$\begin{array}{c|ccccc|}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}$$

DFS Traversal –0 1 2 3 4

Algorithm

Algorithm: DFS_LL(V)

Input: V is a starting vertex

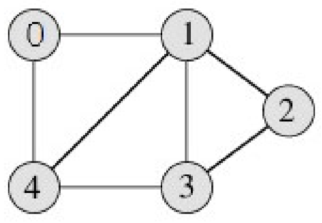
Output : A list VISIT giving order of visited vertices during traversal.

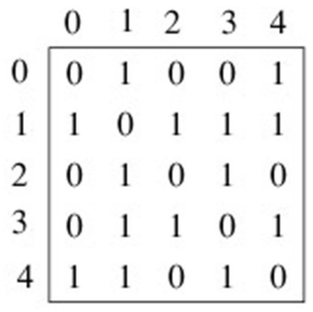
Description: linked structure of graph with gptr as pointer

```
1. if gptr = NULL then
       print "Graph is empty" exit
2. u=v
3. OPEN.PUSH(u)
4. 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
  5. Return VISIT
  6. Stop
```

BFS Traversal





$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 0 & 1 & 0 \end{array}$$

BFS Traversal – 0 1 4 2 3

Algorithm

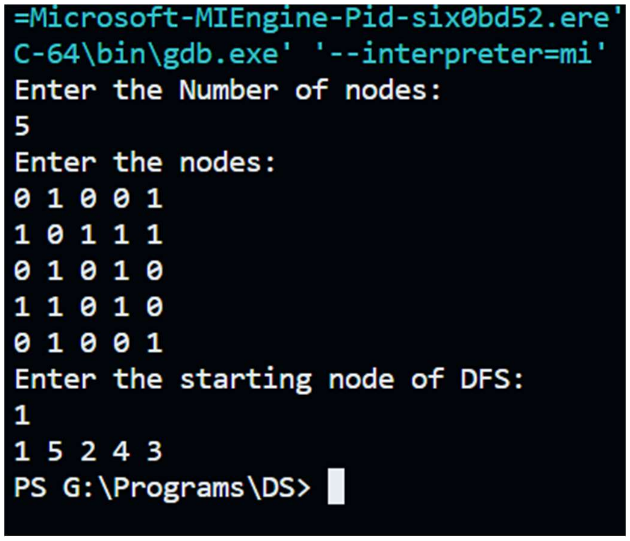
```
Algorithm: DFS()
i=0
count=1
visited[i]=1
print("Visited vertex  i")

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>


void DFS(int v);

typedef enum boolean { false, true } bool;

int n, arr[10][10];

bool visited[10];
```

```
void main() {
    int i, j, v;
    printf("Enter the Number of nodes:\n");
    scanf("%d", &n);
    printf("Enter the nodes:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &arr[i][j]);
        }
    }
    printf("Enter the starting node of DFS:\n");
    scanf("%d", &v);
    for (i = 1; i <= n; i++) {
        visited[i] = false;
    }
    DFS(v);
}

void DFS(int v) {
    int i, stack[10], top = -1, pop;
    top++;
    stack[top] = v;
    while (top >= 0) {
        pop = stack[top];
        top--;
        if (visited[pop] == false) {
            printf("%d ", pop);
            visited[pop] = true;
        } else
            continue;
        for (i = 1; i <= n; i++) {
            if (arr[pop][i] == 1 && visited[i] == false) {
                top++;
                stack[top] = i;
            }
        }
    }
}
```

Output:



```
=Microsoft-MIEngine-Pid-six0bd52.ere'
C-64\bin\gdb.exe' '--interpreter=mi'
Enter the Number of nodes:
5
Enter the nodes:
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
1 1 0 1 0
0 1 0 0 1
Enter the starting node of DFS:
1
1 5 2 4 3
PS G:\Programs\DS>
```

Conclusion:

Time Complexity about the above DFS Program is O(n^2)

Depth-First Search (DFS) and Breadth-First Search (BFS) are fundamental graph traversal algorithms implemented in the C programming language, serving as versatile tools for exploring the structure of graphs, both directed and undirected. DFS delves deep into a graph, visiting nodes recursively along a branch before backtracking, making it suitable for tasks such as cycle detection and topological sorting. On the other hand, BFS systematically explores the graph level by level, facilitating the shortest path discovery and connected component identification. These algorithms are crucial in various applications, such as pathfinding in maps, network analysis, and puzzle-solving, offering efficient means to uncover relationships and uncover hidden patterns within graph data structures. Their flexibility and adaptability make them essential techniques for solving a wide range of real-world problems in C programming.