

---

Yash Sarang.  
Roll No: 47, Class : D6AD.  
Data Structures. Experiment-12.

---

**AIM:**

Implement Graph Traversal techniques:

- a) Depth First Search
  - b) Breadth First Search.
- 

**-Breadth First Algorithm.**

**Theory-**

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

---

**Algorithm:**

- Step 1: SET STATUS = 1 (ready state)  
for each node in G
- Step 2: Enqueue the starting node A  
And set its STATUS = 2 (waiting state)

- Step 3: Repeat Steps 4 and 5 until QUEUE is empty
- Step 4: Dequeue a node N.  
Process it and set its STATUS = 3 (processed state).
- Step 5: Enqueue all the neighbours of N that are in the ready state  
(whose STATUS = 1) and set their STATUS = 2 (waiting state)  
[END OF LOOP]
- Step 6: EXIT
- 
- 

## **Program-**

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

int G[][5] = { {0, 1, 1, 1, 0},
               {1, 0, 1, 0, 0},
               {1, 1, 0, 0, 1},
               {1, 0, 0, 0, 0},
               {0, 0, 1, 0, 0}};

int start = 0, n = 5;

struct Node
{
    int data;
    struct Node *next;
};

struct Node *front = NULL;
struct Node *rear = NULL;

void enqueue(int x)
{
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node));
    temp->data = x;
    temp->next = NULL;
    if (front == NULL)
    {
        front = temp;
        rear = temp;
    }
}
```

```

}
else
{
    rear->next = temp;
    rear = temp;
}

}

int dequeue()
{
    int x = -1;
    if (front == NULL)
        printf("Queue is empty\n");
    else
    {
        struct Node *temp = front;
        x = temp->data;
        front = front->next;
        free(temp);
    }
    return x;
}

int isEmpty()
{
    if (front == NULL)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void BFSTraversal(int G[][5], int start, int n)
{
    int i, j;
    int visited[5] = {0};
    enqueue(start);
    visited[start] = 1;
    while (!isEmpty())
    {
        i = dequeue();

```

```

printf("%d ", i);
for (j = 1; j < n; j++)
{
    if (G[i][j] == 1 && visited[j] == 0)
    {
        enqueue(j);

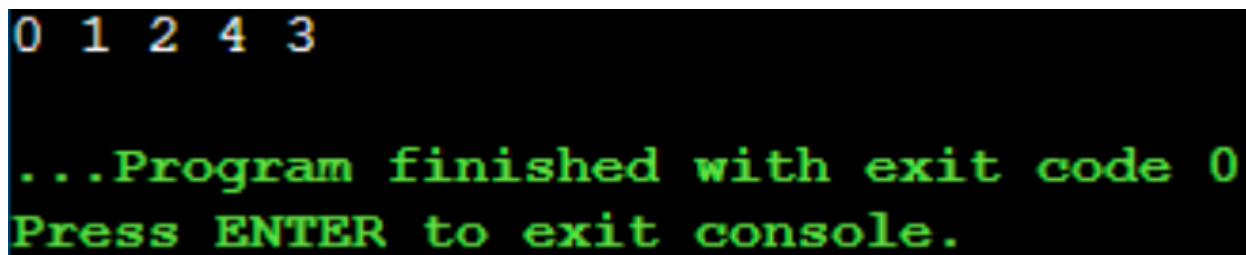
        visited[j] = 1;
    }
}
}
}
}
int main()
{
    BFSTraversal(G, start, n);
    return 0;
}

```

---

-

### Output-



```

0 1 2 4 3

...Program finished with exit code 0
Press ENTER to exit console.

```

---

-

## Depth First Algorithm.

### Theory-

Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node or the

node which has no children. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored. The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

---

-

### **Algorithm-**

- Step 1: SET STATUS = 1 (ready state)  
for each node in G
  - Step 2: Push the starting node A on the stack  
and set its STATUS = 2 (waiting state)
  - Step 3: Repeat Steps 4 and 5 until STACK is empty
  - Step 4: Pop the top node N.  
Process it and set its STATUS = 3 (processed state)
  - Step 5: Push on the stack all the neighbours of N  
that are in the ready state (whose STATUS = 1)  
and set their STATUS = 2 (waiting state)  
[END OF LOOP]
  - Step 6: EXIT
- 
- 

### **Program-**

```
#include <stdio.h>
int G[][5] = {    {0, 1, 1, 1, 0},
                  {1, 0, 1, 0, 0},
                  {1, 1, 0, 0, 1},
                  {1, 0, 0, 0, 0},
```

```

        {0, 0, 1, 0, 0} };

int start = 0, n = 5;

void DFSTraversal(int G[][5], int start, int n)
{
    static int visited[5] = {0};
    int j;
    if (visited[start] == 0)
    {
        printf("%d ", start);
        visited[start] = 1;
        for (j = 1; j < n; j++)
        {
            if (G[start][j] == 1 && visited[j] == 0)
                DFSTraversal(G, j, n);
        }
    }
}

int main()
{
    DFSTraversal(G, start, n);
    return 0;
}

```

## Output-

```

0 1 2 4 3

...Program finished with exit code 0
Press ENTER to exit console.

```

## CONCLUSION:

We have successfully implemented Breadth-First Search and Depth First Search algorithm.

-----

-