Practical no: 10

Aim: Write a program to traverse the tree using breadth first search.

breadth first search:-

The breadth-first search (BFS) algorithm is used to search a tree or graph data structure for a node that meets a set of criteria. It starts at the tree's root or graph and searches/visits all nodes at the current depth level before moving on to the nodes at the next depth level. Breadth-first search can be used to solve many problems in graph theory.

Breadth-First Traversal (or Search) for a graph is similar to the Breadth-First Traversal of a tree (See method 2 of this post).
The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:

- Visited and
- Not visited.

A boolean visited array is used to mark the visited vertices. For simplicity, it is assumed that all vertices are reachable from the starting vertex. BFS uses a **queue data structure** for traversal.

BFS algorithm

A standard BFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

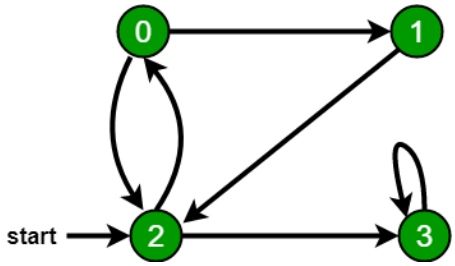
The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node.

Example:

In the following graph, we start traversal from vertex 2.



When we come to **vertex 0**, we look for all adjacent vertices of it.

- 2 is also an adjacent vertex of 0.
- If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process.

There can be multiple BFS traversals for a graph. Different BFS traversals for the above graph : 2, 3, 0, 1
2, 0, 3, 1

Program:-

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

struct queue
{
    int size;
    int f;
    int r;
    int* arr;
};



int isEmpty(struct queue *q){
    if(q->r==q->f){
        return 1;
    }
    return 0;
}

int isFull(struct queue *q){
    if(q->r==q->size-1){
        return 1;
    }
    return 0;
}
```

```c
void enqueue(struct queue *q, int val){
    if(isFull(q)){
        printf("This Queue is full\n");
    }
    else{
        q->r++;
        q->arr[q->r] = val;
        // printf("Enqued element: %d\n", val);
    }
}

int dequeue(struct queue *q){
    int a = -1;
    if(isEmpty(q)){
        printf("This Queue is empty\n");
    }
    else{
        q->f++;
        a = q->arr[q->f];
    }
    return a;
}

int main(){
    // Initializing Queue (Array Implementation)
    struct queue q;
    q.size = 400;
    q.f = q.r = 0;
    q.arr = (int*) malloc(q.size*sizeof(int));

    // BFS Implementation
    int node;
    int i = 1;
    int visited[7] = {0,0,0,0,0,0,0};
    int a [7][7] = {
        {0,1,1,1,0,0,0},
        {1,0,1,0,0,0,0},
        {1,1,0,1,1,0,0},
        {1,0,1,0,1,0,0},
        {0,0,1,1,0,1,1},
        {0,0,0,0,1,0,0},
        {0,0,0,0,1,0,0}
```

```
    };
    printf("%d", i);
    visited[i] = 1;
    enqueue(&q, i); // Enqueue i for exploration
    while (!isEmpty(&q))
    {
        int node = dequeue(&q);
        for (int j = 0; j < 7; j++)
        {
            if(a[node][j] ==1 && visited[j] == 0){
                printf("%d", j);
                visited[j] = 1;
                enqueue(&q, j);
            }
        }
    }
    return 0;
}
```

the output we received was:

```
0123456
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

BFS program in Matrix form

```
#include<stdio.h>
                int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;
                void bfs(int v) {
                 for(i = 1; i <= n; i++)
                 if(a[v][i] && !visited[i])
                 q[++r] = i;
                 if(f <= r) {
                 visited[q[f]] = 1;
                 bfs(q[f++]);
                 }
                }
                void main() {
                 int v;
                 printf("\n Enter the number of vertices:");
                 scanf("%d", &n);

                 for(i=1; i <= n; i++) {
                 q[i] = 0;
                 visited[i] = 0;
                 }

                 printf("\n Enter graph data in matrix form:\n");
                 for(i=1; i<=n; i++) {
```

```
 for(j=1;j<=n;j++) {
 scanf("%d", &a[i][j]);
 }
 }

 printf("\n Enter the starting vertex:");
 scanf("%d", &v);
 bfs(v);
 printf("\n The node which are reachable are:\n");

 for(i=1; i <= n; i++) {
 if(visited[i])
 printf("%d\t", i);
 else {
 printf("\n Bfs is not possible. Not all nodes are reachable");
 break;
 }
 }
}
```

Sample Output :