Asssignment No. 14

Title :-

Store a graph using adjacency matrix or adjacency list representation and perform Breadth First Traversal.(Recursive/Non Recursive)

Description :-

# Breadth First Search or BFS for a Graph

Unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

For 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. A Breadth First Traversal of the following graph is 2, 0, 3, 1.

[](https://media.geeksforgeeks.org/wp-content/uploads/bfs-5.png)

SOURCE CODE:-

#include<iostream>

#include <list>

using namespace std;

class Graph

{

    int V;

    list<int> \*adj;

public:

    Graph(int V);

    void addEdge(int v, int w);

    void BFS(int s);

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w);

}

void Graph::BFS(int s)

{

    bool \*visited = new bool[V];

    for(int i = 0; i < V; i++)

        visited[i] = false;

    list<int> queue;

    visited[s] = true;

    queue.push\_back(s);

    list<int>::iterator i;

    while(!queue.empty())

    {

        s = queue.front();

        cout << s << " ";

        queue.pop\_front();

        for (i = adj[s].begin(); i != adj[s].end(); ++i)

        {

            if (!visited[\*i])

            {

                visited[\*i] = true;

                queue.push\_back(\*i);

            }

        }

    }

}

int main()

{

    Graph g(4);

    g.addEdge(0, 1);

    g.addEdge(0, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "

         << "(starting from vertex 2) \n";

    g.BFS(2);

    return 0;

}

Output:

Following is Breadth First Traversal (starting from vertex 2)

2 0 3 1

Conclusion :-

1. To know about the BFS (Breadth First Search ) for A Graph how to taverse it using BFS.
2. To know about the representation of the of a graph using Adjecency list .