**Practical No 13**

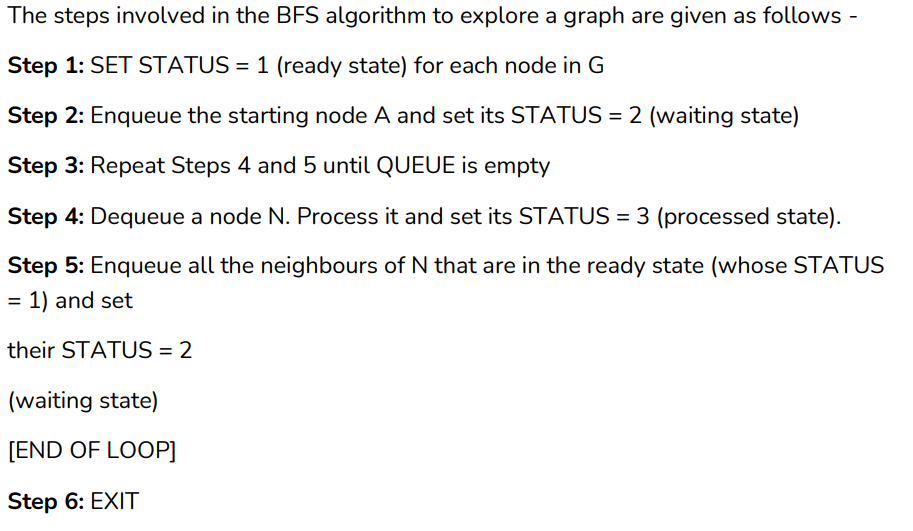
**Aim :** Implementation of Graph Traversal (Breadth First Search).

**Theory :**

**Breadth-first search** is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and unvisited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

**Algorithm**



**Applications of BFS algorithm**

The applications of breadth-first-algorithm are given as follows -

o BFS can be used to find the neighboring locations from a given source location.

o In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.

o BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.

o BFS is used to determine the shortest path and minimum spanning tree.

o BFS is also used in Cheney's technique to duplicate the garbage collection.

o It can be used in the ford-Fulkerson method to compute the maximum flow in a flow network.

**Complexity of BFS algorithm**

Time complexity of BFS depends upon the data structure used to represent the graph. The time complexity of the BFS algorithm is O(V+E), since in the worst case, the BFS algorithm explores every node and edge. In a graph, the number of vertices is O(V), whereas the number of edges is O(E).

The space complexity of BFS can be expressed as O(V), where V is the number of vertices.

**Code :**

#include <bits/stdc++.h>

using namespace std;

class Graph {

int V;

vector<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.resize(V);

}

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

{

adj[v].push\_back(w);

}

void Graph::BFS(int s)

{

vector<bool> visited;

visited.resize(V, false);

list<int> queue;

visited[s] = true;

queue.push\_back(s);

while (!queue.empty()) {

s = queue.front();

cout << s << " ";

queue.pop\_front();

for (auto adjecent : adj[s]) {

if (!visited[adjecent]) {

visited[adjecent] = true;

queue.push\_back(adjecent);

}

}

}

}

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 "

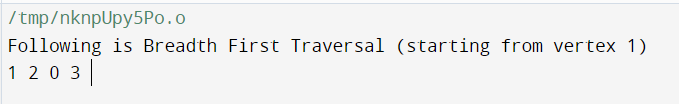
<< "(starting from vertex 1) \n";

g.BFS(1);

return 0;

}

**Output :**

****

**Conclusion : learned implementation of Breadth first search.**