**Experiment-6**

**Aim-** From a given starting node in a digraph, print all the nodes reachable by using BFS/DFS method.

**Theory-**

In graph traversal, Breadth-First Search (BFS) and Depth-First Search (DFS) are used to explore all nodes reachable from a given starting node in a directed graph. BFS explores level-by-level, ensuring the shortest path in unweighted graphs, while DFS explores as far as possible before backtracking. Both algorithms are fundamental for exploring connected components, cycle detection, and reachability in graphs.

**Software Used –** Visual Studio Code

**Code-**

**#**include<iostream>

#include<vector>

#include<list>

#include<queue>

using namespace std;

class Graph{

int V;

list<int> \*l;

public:

Graph(int V){

this->V = V;

l = new list<int>[V];

}

void addEdge(int u, int v){ // u -> v (directed graph)

l[u].push\_back(v);

}

void bfs(int start){

vector<bool> vis(V, false);

queue<int> q;

q.push(start);

vis[start] = true;

while(!q.empty()){

int u = q.front(); // current vertex

q.pop();

cout << u << " ";

for(int v : l[u]){ // visiting all neighbours of the current vertex

if(!vis[v]){

vis[v] = true;

q.push(v);

}

}

}

cout << endl;

}

void dfsHelper(int u, vector<bool>& vis){

vis[u] = true;

cout << u << " ";

for(int v : l[u]){

if(!vis[v]){

dfsHelper(v, vis);

}

}

}

void dfs(int start){

vector<bool> vis(V, false);

dfsHelper(start, vis);

cout << endl;

}

};

int main(){

Graph graph(10);

graph.addEdge(1, 6);

graph.addEdge(6, 4);

graph.addEdge(4, 3);

graph.addEdge(4, 9);

graph.addEdge(3, 8);

graph.addEdge(3, 7);

graph.addEdge(0, 2);

graph.addEdge(2, 5);

int startNode;

cout << "Enter the starting node for BFS and DFS: ";

cin >> startNode;

cout << "BFS from node " << startNode << ": ";

graph.bfs(startNode);

cout << "DFS from node " << startNode << ": ";

graph.dfs(startNode);

return 0;

}

**Output-**

A black background with white text

Description automatically generated