**Experiment – 3.3**

**Student Name: Vivek UID:**

**Branch: CSE Section/Group:**

**Date of Performance: Semester: 3**

**Subject Name: OOPs using JAVA Subject Code: 21CSH-218**

**Aim of the practical:** Write a program to demonstrate the traversal of graph using Breadth first search

Depth first search

# Algorithm:

## Breadth First search

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 settheir STATUS = 2(waiting state)[END OF LOOP]

Step 6: EXIT

## Depth First Search

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 neighbors 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 code:

**Breadth first Search** #include <iostream> #include <list>

using namespace std; class Graph {

int numVertices; list<int>\* adjLists; bool\* visited;

public:

Graph(int vertices);

void addEdge(int src, int dest); void BFS(int startVertex);

};

Graph::Graph(int vertices) { numVertices = vertices;

adjLists = new list<int>[vertices];

}

void Graph::addEdge(int src, int dest) { adjLists[src].push\_back(dest); adjLists[dest].push\_back(src);

}

void Graph::BFS(int startVertex) { visited = new bool[numVertices]; for (int i = 0; i < numVertices; i++) visited[i] = false;

list<int> queue; visited[startVertex] = true; queue.push\_back(startVertex); list<int>::iterator i;

while (!queue.empty()) {

int currVertex = queue.front();

cout << "Visited " << currVertex << " "; queue.pop\_front();

for (i = adjLists[currVertex].begin(); i != adjLists[currVertex].end(); ++i) { int adjVertex = \*i;

if (!visited[adjVertex]) { visited[adjVertex] = true; queue.push\_back(adjVertex);

}

}

}

}

int main() { Graph g(5); g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

g.BFS(3);

return 0;

}

**Depth first Search-** #include<bits/stdc++.h> using namespace std; class Graph

{

int V;

list<int> \*adjList; public:

Graph(int V)

{

this->V = V;

adjList = new list<int>[V];

}

void addEdge(int v, int w)

{

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

}

void DFS();

void DFSUtil(int s, vector<bool> &visited);

};

void Graph::DFSUtil(int s, vector<bool> &visited)

{

stack<int> dfsstack; dfsstack.push(s);

while (!dfsstack.empty())

{

s = dfsstack.top(); dfsstack.pop();

if (!visited[s])

{

cout << s << " "; visited[s] = true;

}

for (auto i = adjList[s].begin(); i != adjList[s].end(); ++i) if (!visited[\*i])

dfsstack.push(\*i);

}

}

void Graph::DFS()

{

vector<bool> visited(V, false); for (int i = 0; i < V; i++)

if (!visited[i]) DFSUtil(i, visited);

}

int main()

{

Graph gidfs(7); gidfs.addEdge(0, 1);

gidfs.addEdge(0, 2);

gidfs.addEdge(0, 3);

gidfs.addEdge(1, 2);

gidfs.addEdge(2, 4);

gidfs.addEdge(3, 3);

gidfs.addEdge(4, 4);

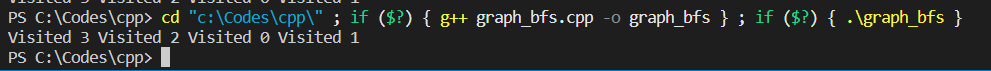
cout << "Output of Iterative Depth-first traversal:\n"; gidfs.DFS();

return 0;

}

# Output:

## Breadth first Search-



**Depth first Search-**

