Ruchita Abhang TC201

**AI Assignment No. 1**

**DFS:**

// C++ program to print DFS traversal from

#include <bits/stdc++.h>

using namespace std;

class Graph {

public:

map<int, bool> visited;

map<int, list<int> > adj;

// function to add an edge to graph

void addEdge(int v, int w);

// DFS traversal of the vertices

// reachable from v

void DFS(int v);

};

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

{

adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::DFS(int v)

{

// Mark the current node as visited and

// print it

visited[v] = true;

cout << v << " ";

// Recur for all the vertices adjacent

// to this vertex

list<int>::iterator i;

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

if (!visited[\*i])

DFS(\*i);

}

// Driver code

int main()

{

// Create a graph given in the above diagram

cout<<"Ruchita Abhang TC201"<<endl;

Graph g;

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 Depth First Traversal"

" (starting from vertex 2) \n";

g.DFS(2);

return 0;

}

**BFS:**

// Program to print BFS traversal from a given

.

#include<iostream>

#include <list>

using namespace std;

// This class represents a directed graph using

// adjacency list representation

class Graph

{

int V; // No. of vertices

// Pointer to an array containing adjacency

// lists

list<int> \*adj;

public:

Graph(int V); // Constructor

// function to add an edge to graph

void addEdge(int v, int w);

// prints BFS traversal from a given source s

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); // Add w to v’s list.

}

void Graph::BFS(int s)

{

// Mark all the vertices as not visited

bool \*visited = new bool[V];

for(int i = 0; i < V; i++)

visited[i] = false;

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.push\_back(s);

// 'i' will be used to get all adjacent

// vertices of a vertex

list<int>::iterator i;

while(!queue.empty())

{

// Dequeue a vertex from queue and print it

s = queue.front();

cout << s << " ";

queue.pop\_front();

// Get all adjacent vertices of the dequeued

// vertex s. If a adjacent has not been visited,

// then mark it visited and enqueue it

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

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

// Driver program to test methods of graph class

int main()

{

// Create a graph given in the above diagram

cout<<"Ruchita Abhang TC201"<<endl;

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;

}