**SHIVAM GUPTA**

**19569**

**ALGORITHM PRACTICALS**

**DFS:**

#include <cstdio>

#include <algorithm>

#include <iostream>

#include <vector>

#define MAX 101

using namespace std;

enum colors {BLACK, WHITE, GRAY};

int color[MAX], d[MAX], p[MAX], f[MAX], t, vertex, edge;

int NIL = numeric\_limits<int>::min();

void DFS(vector<int>[]);

void DFS\_VISIT(vector<int>[],int);

int main(void)

{

vector<int> adjList[MAX];

int u, v;

cout<<"Enter the total vertex : ";

cin>>vertex;

cout<<"Enter the total edges : ";

cin>>edge;

cout<<"Enter the Connection of vertex : "<<endl;

cout<<"u :: v"<<endl;

for(int e=1; e<=edge; e++) {

cin >> u >> v;

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

}

DFS(adjList);

cout<<"DEPTH FIRST TRAVERSAL"<<endl;

for(int v=1; v<=vertex; v++) {

cout<<"v"<<v<<" ("<<d[v]<<"/"<<f[v]<<") "<<endl;;

}

return 0;

}

void DFS(vector<int> G[]) {

for(int u=0; u<=vertex; u++) {

color[u] = WHITE;

p[u] = NIL;

}

t = 0;

for(int u=1; u<=vertex; u++) {

if(color[u] == WHITE) {

DFS\_VISIT(G,u);

}

}

}

void DFS\_VISIT(vector<int> G[], int u) {

t = t + 1;

d[u] = t;

color[u] = GRAY;

for(int v=0; v<G[u].size(); v++) {

if(color[G[u][v]] == WHITE) {

p[G[u][v]] = u;

DFS\_VISIT(G,G[u][v]);

}

}

color[u] = BLACK;

t = t + 1;

f[u] = t;

}

**BFS:**

#include<iostream>

#include <list>

using namespace std;

class Graph

{

int V;

// Pointer to an array containing adjacency lists

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); // 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;

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);

}

}

}

}

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;

}