**POST – LAB**

#include <iostream>

#include <vector>

#include <list>

using namespace std;

class Graph

{

public:

Graph(int numVertex);

int GetNumVertices();

int numberOfEdges();

void insertEdge(int frmVertex, int toVertex);

void removeEdge(int frmVertex, int toVertex);

int degree(int vertex);

void Graph::DFSUtil(int v, bool visited[]);

void depthfirstSearch(int s);

void breadthfirstSearch(int s);

private:

list <int>\*adj\_matrix;

int numVertices,edge;

};

//part1: constructor initializes adjacency matrix

Graph::Graph(int numVertex)

{ this->numVertices = numVertices;

adj\_matrix= new list<int>[numVertices];

}

//part2: returns the number of vertices in the graph

int Graph:: GetNumVertices()

{return numVertices;

}

//part3: returns the number of edges in the graph

int Graph::numberOfEdges()

{return edge;

}

//part4: inserts edge going from one vertex to another

void Graph::insertEdge(int frmVertex, int toVertex)

{adj\_matrix[frmVertex].push\_back(toVertex);

}

//part5: removes edge going from one vertex to another

void Graph::removeEdge(int frmVertex, int toVertex)

{adj\_matrix[frmVertex].push\_back(toVertex);

}

//part6: returns the degree of the node passed

int Graph::degree(int vertex)

{ return vertex;

}

void Graph::DFSUtil(int v, bool visited[])

{

// 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\_matrix[v].begin(); i != adj\_matrix[v].end(); ++i)

if (!visited[\*i])

DFSUtil(\*i, visited);

}

void Graph::depthfirstSearch(int s)

{bool\* visited = new bool[numVertices];

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

visited[i] = false;

int v;

// Call the recursive helper function

// to print DFS traversal

DFSUtil(v, visited);

}

void Graph::breadthfirstSearch(int start)

{

bool \*visited = new bool[numVertices];

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

visited[i] = false;

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[start] = true;

queue.push\_back(start);

// '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

start = queue.front();

cout << start << " ";

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\_matrix[start].begin(); i != adj\_matrix[start].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

int main()

{

Graph \*g;

vector<bool> v;

vector<bool>visited=v;

//creating an object of graph with 5 vertices

g=new Graph(5);

//inserting edges in the graph

g->insertEdge(0,1);

g->insertEdge(0,4);

g->insertEdge(1,0);

g->insertEdge(1,2);

g->insertEdge(1,3);

g->insertEdge(1,4);

g->insertEdge(2,1);

g->insertEdge(2,3);

g->insertEdge(3,1);

g->insertEdge(3,2);

g->insertEdge(3,4);

g->insertEdge(4,0);

g->insertEdge(4,1);

g->insertEdge(4,3);

//display total number of edges

cout<<"Number of edges are "<<g->numberOfEdges()<<endl;

//display degree of vertex number 4

cout<<"Degree of vertex "<<g->degree(4)<<endl;

cout<<"Output for Depth first search starting from vertex 0 "<<endl;

g->depthfirstSearch(0);

cout<<"Output for Breadth first search starting from vertex 0 "<<endl;

g->breadthfirstSearch(0);

return 0;

}