**Muhammad Abdullah**

**SE(3A) | 19F-0916**

Data Structures Lab

Graphs Implementation

**Question # 1: (ADJECENCY MATRIX)**

**Program**

#include <iostream>

using namespace std;

class Graph // Class For Graph

{

public:

int \*\* Matrix; //2D Array for Adjecency Matrix

int Visited;

Graph(int Visited) // Constructor for Making Matrix

{

this->Visited = Visited;

Matrix = new int\*[Visited];

for (int i = 1; i <= Visited; i++)

{

Matrix[i] = new int[Visited];

for (int j = 1; j <= Visited; j++)

Matrix[i][j] = 0;

}

}

void Add\_Edge(int i, int j) // Adding Edge

{

Matrix[i][j] = 1;

Matrix[j][i] = 1;

}

void Remove\_Edge(int i, int j) //Removing Edge (if needed)

{

Matrix[i][j] = 0;

Matrix[j][i] = 0;

}

void Adjencency\_Matrix() // Displaying Adjececy Matrix

{

cout << endl;

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

cout << " " << Matrix[i][j] << " ";

cout << endl << endl;;

}

}

};

int main() // Main Code

{

int size = 0, i = 1, opt = 0;

cout << endl << " Enter Size of the Matrix : ";

cin >> size;

Graph Adj\_Matrix(size);

while (i == 1)

{

system("cls");

cout << "----------------------------------------------------" << endl; //Menu Driven Program

cout << " Press 1 to Make a Graph with Adjacency Matrix !" << endl;

cout << " Press 2 to Remove an Edge From the Graph !" << endl;

cout << " Press 3 to Display the Adjecency Matrix !" << endl;

cout << " Press 0 to Exit from the System !" << endl;

cout << "----------------------------------------------------" << endl;

cout << endl << " Enter Choice : ";

cin >> opt;

switch (opt)

{

case 1: // To Add Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to add Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to add Node : ";

cin >> col;

Adj\_Matrix.Add\_Edge(row, col);

system("pause");

break;

}

case 2: // To Delete Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to Remove Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to Remove Node : ";

cin >> col;

Adj\_Matrix.Remove\_Edge(row, col);

system("pause");

break;

}

case 3: // To Show Matrix

{

cout << endl << " Adjeceny Matrix is : " << endl;

Adj\_Matrix.Adjencency\_Matrix();

cout << endl;

system("pause");

break;

}

case 0: // To Exit from System

{

cout << endl << " You have exited from the System !" << endl;

i = 0;

system("pause");

break;

}

default:

{

cout << endl << " Invalid Option, Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**Initializing MATRIX**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**MAIN MENU**

**A picture containing text, monitor, screenshot, indoor

Description automatically generated**

**ADDING EDGE**

**A picture containing text, monitor, indoor, electronics

Description automatically generated**

**DISPLAYING ADJECENCY LIST**

**A picture containing text, monitor, electronics, indoor

Description automatically generated**

**Question # 2: (ADJECENCY LIST)**

**Program**

#include <iostream>

using namespace std;

struct Node // Edge Creation

{

int Data;

Node \*Next\_Node;

};

struct Adjecency\_List // Pointer Head (to make a List)

{

Node \*Head;

};

class Graph // Class Implementation

{

public:

Adjecency\_List \*List; // 1D List for List

int Visited;

Graph(int Visited) // Starting From Zero

{

this->Visited = Visited;

List = new Adjecency\_List[Visited];

for (int i = 1; i <= Visited; i++)

{

List[i].Head = NULL;

}

}

Node \*List\_Node(int Data) // Making List accordingly

{

Node \*New\_Node = new Node;

New\_Node->Data = Data;

New\_Node->Next\_Node = NULL;

return New\_Node;

}

void Add\_Edge(int Coming, int Data) // Adding Edge

{

Node \*New\_Node = List\_Node(Data);

New\_Node->Next\_Node = List[Coming].Head;

List[Coming].Head = New\_Node;

New\_Node = List\_Node(Data);

New\_Node->Next\_Node = List[Data].Head;

List[Data].Head = New\_Node;

}

void Adjencency\_List() // Displaying Adjecency List

{

cout << endl;

for (int i = 1; i <= Visited; i++)

{

Node \*Traverse = List[i].Head;

cout << "Vertex " << i << " : ";

while (Traverse != NULL)

{

cout << " -> " << Traverse->Data;

Traverse = Traverse->Next\_Node;

}

cout << endl;

}

}

};

int main() // Main Code

{

int size = 0, i = 1, opt = 0;

cout << endl << " Enter Size of the List : ";

cin >> size;

Graph Adj\_List(size);

while (i == 1)

{

system("cls");

cout << "----------------------------------------------------" << endl; //Menu Driven Program

cout << " Press 1 to Make a Graph with Adjacency List !" << endl;

cout << " Press 2 to Display the Adjecency List !" << endl;

cout << " Press 0 to Exit from the System !" << endl;

cout << "----------------------------------------------------" << endl;

cout << endl << " Enter Choice : ";

cin >> opt;

switch (opt)

{

case 1: // To Add Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to add Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to add Node : ";

cin >> col;

Adj\_List.Add\_Edge(row, col);

system("pause");

break;

}

case 2: // To Show List

{

cout << endl << " Adjeceny List is : " << endl;

Adj\_List.Adjencency\_List();

cout << endl;

system("pause");

break;

}

case 0: // To Exit from System

{

cout << endl << " You have exited from the System !" << endl;

i = 0;

break;

}

default:

{

cout << endl << " Invalid Option, Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**Initializing List**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**MAIN MENU**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**ADDING EDGE**

**A picture containing text, monitor, screenshot, indoor

Description automatically generated**

**DISPLAYING LIST  
A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**Question # 3: (VERTEX INDEGREE ZERO)**

**Program**

#include <iostream>

using namespace std;

class Graph // Class For Graph

{

public:

int \*\* Matrix; //2D Array for Adjecency Matrix

int Visited;

Graph(int Visited) // Constructor for Making Matrix

{

this->Visited = Visited;

Matrix = new int\*[Visited];

for (int i = 1; i <= Visited; i++)

{

Matrix[i] = new int[Visited];

for (int j = 1; j <= Visited; j++)

Matrix[i][j] = 0;

}

}

void Add\_Edge(int i, int j) // Adding Edge

{

Matrix[i][j] = 1; //It will be a Directed Graph

}

void Remove\_Edge(int i, int j) //Removing Edge (if needed)

{

Matrix[i][j] = 0;

}

void Find\_New\_Vertex\_of\_InDegree\_Zero() //Checking Vertex with 0 InDegree

{

int \*Checker = new int[Visited];

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

{

Checker[i] = 0;

}

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

{

if (Matrix[i][1] == 1)

{

Checker[i]++;

}

if (Matrix[i][j] == Matrix[j][i])

{

Checker[i]++;

}

}

}

for (int i = 0; i < Visited; i++) //Displaying Zero Indegree vertex

{

if (Checker[i] == 0)

{

cout << "Vertex " << i+1 << " Have Zero Inorder " << endl;

}

}

}

void Adjencency\_Matrix() // Displaying Adjececy Matrix

{

cout << endl;

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

cout << " " << Matrix[i][j] << " ";

cout << endl << endl;;

}

}

};

int main() // Main Code

{

int size = 0, i = 1, opt = 0;

cout << endl << " Enter Size of the Matrix : ";

cin >> size;

Graph Adj\_Matrix(size);

while (i == 1)

{

system("cls");

cout << "----------------------------------------------------" << endl; //Menu Driven Program

cout << " Press 1 to Make a Graph with Adjacency Matrix !" << endl;

cout << " Press 2 to Remove an Edge From the Graph !" << endl;

cout << " Press 3 to Check Vertex with Zero In-Degree !" << endl;

cout << " Press 4 to Display the Adjecency Matrix !" << endl;

cout << " Press 0 to Exit from the System !" << endl;

cout << "----------------------------------------------------" << endl;

cout << endl << " Enter Choice : ";

cin >> opt;

switch (opt)

{

case 1: // To Add Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to add Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to add Node : ";

cin >> col;

Adj\_Matrix.Add\_Edge(row, col);

system("pause");

break;

}

case 2: // To Delete Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to Remove Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to Remove Node : ";

cin >> col;

Adj\_Matrix.Remove\_Edge(row, col);

system("pause");

break;

}

case 3: // To Check Vertices

{

cout << endl;

Adj\_Matrix.Find\_New\_Vertex\_of\_InDegree\_Zero();

cout << endl;

system("pause");

break;

}

case 4: // To Show Matrix

{

cout << endl << " Adjeceny Matrix is : " << endl;

Adj\_Matrix.Adjencency\_Matrix();

cout << endl;

system("pause");

break;

}

case 0: // To Exit from System

{

cout << endl << " You have exited from the System !" << endl;

i = 0;

system("pause");

break;

}

default:

{

cout << endl << " Invalid Option, Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**ADJECENCY MATRIX OF GIVEN GRAPH**

**A picture containing text, monitor, computer, indoor

Description automatically generated**

**VERTEX WITH ZERO INDEGREE (GIVEN GRAPH)**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**Question # 4: (BFS TRAVERSAL)**

**Program**

#include <iostream>

using namespace std;

struct Arrayy //Implementaiting Stack

{

int Size;

int top = -1;

int \*Array = NULL;

};

class Stack // Stack Class

{

public:

Stack(int size)

{

ARRAY.Size = size;

ARRAY.Array = new int[size];

}

Arrayy ARRAY;

void push(int Value) //push function

{

if (ARRAY.top != ARRAY.Size - 1)

{

ARRAY.top++;

ARRAY.Array[ARRAY.top] = Value;

}

else

cout << endl << "Stack is FULL !!" << endl;

}

int pop() //pop function

{

if (ARRAY.top != -1)

{

int temp = 0;

temp = ARRAY.Array[ARRAY.top];

ARRAY.top--;

return temp;

}

else

cout << endl << "Stack is Empty !!";

}

};

class Graph // Class For Graph

{

public:

int \*\* Matrix; //2D Array for Adjecency Matrix

int Visited;

Graph(int Visited) // Constructor for Making Matrix

{

this->Visited = Visited;

Matrix = new int\*[Visited];

for (int i = 1; i <= Visited; i++)

{

Matrix[i] = new int[Visited];

for (int j = 1; j <= Visited; j++)

Matrix[i][j] = 0;

}

}

void Add\_Edge(int i, int j) // Adding Edge

{

Matrix[i][j] = 1;

Matrix[j][i] = 1;

}

void Remove\_Edge(int i, int j) //Removing Edge (if needed)

{

Matrix[i][j] = 0;

Matrix[j][i] = 0;

}

void BFS(int Vertex) // BFS Traversal and Displaying

{

int count = 1;

Stack stack(Visited);

cout << endl << " BFS is : ";

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

{

if (count == stack.ARRAY.top)

{

cout << stack.pop() << " ";

}

if (Matrix[i][j] == 1)

{

stack.push(count);

count++;

}

}

}

while (stack.ARRAY.top != -1)

{

cout << stack.pop() << " ";

}

cout << endl;

}

void Adjencency\_Matrix() // Displaying Adjececy Matrix

{

cout << endl;

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

cout << " " << Matrix[i][j] << " ";

cout << endl << endl;;

}

}

};

int main() // Main Code

{

int size = 0, i = 1, opt = 0;

cout << endl << " Enter Size of the Matrix : ";

cin >> size;

Graph Adj\_Matrix(size);

while (i == 1)

{

system("cls");

cout << "----------------------------------------------------" << endl; //Menu Driven Program

cout << " Press 1 to Make a Graph with Adjacency Matrix !" << endl;

cout << " Press 2 to Remove an Edge From the Graph !" << endl;

cout << " Press 3 to Display the BFS Traversal !" << endl;

cout << " Press 4 to Display the Adjecency Matrix !" << endl;

cout << " Press 0 to Exit from the System !" << endl;

cout << "----------------------------------------------------" << endl;

cout << endl << " Enter Choice : ";

cin >> opt;

switch (opt)

{

case 1: // To Add Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to add Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to add Node : ";

cin >> col;

Adj\_Matrix.Add\_Edge(row, col);

system("pause");

break;

}

case 2: // To Delete Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to Remove Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to Remove Node : ";

cin >> col;

Adj\_Matrix.Remove\_Edge(row, col);

system("pause");

break;

}

case 3: // To Show Matrix

{

cout << endl;

Adj\_Matrix.BFS(0);

cout << endl;

system("pause");

break;

}

case 4: // To Show Matrix

{

cout << endl << " Adjeceny Matrix is : " << endl;

Adj\_Matrix.Adjencency\_Matrix();

cout << endl;

system("pause");

break;

}

case 0: // To Exit from System

{

cout << endl << " You have exited from the System !" << endl;

i = 0;

system("pause");

break;

}

default:

{

cout << endl << " Invalid Option, Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**MAIN MENU**

**A picture containing text, monitor, electronics, indoor

Description automatically generated**

**BFS TRAVERSAL GENERIC**

**A picture containing text, monitor, screenshot, indoor

Description automatically generated**

**Question # 5:**

**Program**

#include <iostream>

using namespace std;

struct Node // Edge Creation

{

int Data;

Node \*Next\_Node;

};

struct Adjecency\_List // Pointer Head (to make a List)

{

Node \*Head;

};

class Graph // Class Implementation

{

public:

Adjecency\_List \*List; // 1D List for List

int Visited;

Graph(int Visited) // Starting From Zero

{

this->Visited = Visited;

List = new Adjecency\_List[Visited];

for (int i = 1; i <= Visited; i++)

{

List[i].Head = NULL;

}

}

Node \*List\_Node(int Data) // Making List accordingly

{

Node \*New\_Node = new Node;

New\_Node->Data = Data;

New\_Node->Next\_Node = NULL;

return New\_Node;

}

void Add\_Edge(int Coming, int Data) // Adding Edge (Directed Graph)

{

Node \*New\_Node = List\_Node(Data);

New\_Node->Next\_Node = List[Coming].Head;

List[Coming].Head = New\_Node;

}

void Adjencency\_List() // Displaying Adjecency List

{

cout << endl;

for (int i = 1; i <= Visited; i++)

{

Node \*Traverse = List[i].Head;

cout << "Vertex " << i << " : ";

while (Traverse != NULL)

{

cout << " -> " << Traverse->Data;

Traverse = Traverse->Next\_Node;

}

cout << endl;

}

}

};

int main() // Main Code

{

int size = 0, i = 1, opt = 0;

cout << endl << " Enter Size of the List : ";

cin >> size;

Graph Adj\_List(size);

while (i == 1)

{

system("cls");

cout << "----------------------------------------------------" << endl; //Menu Driven Program

cout << " Press 1 to Make a Graph with Adjacency List !" << endl;

cout << " Press 2 to Display the Adjecency List !" << endl;

cout << " Press 0 to Exit from the System !" << endl;

cout << "----------------------------------------------------" << endl;

cout << endl << " Enter Choice : ";

cin >> opt;

switch (opt)

{

case 1: // To Add Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to add Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to add Node : ";

cin >> col;

Adj\_List.Add\_Edge(row, col);

system("pause");

break;

}

case 2: // To Show List

{

cout << endl << " Adjeceny List is : " << endl;

Adj\_List.Adjencency\_List();

cout << endl;

system("pause");

break;

}

case 0: // To Exit from System

{

cout << endl << " You have exited from the System !" << endl;

i = 0;

break;

}

default:

{

cout << endl << " Invalid Option, Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**MAIN MENU CODE  
A picture containing text, monitor, screenshot, indoor

Description automatically generated**

**ADJECENCY LIST ACCORDING TO GRAPH**

**A picture containing text, monitor, screenshot, indoor

Description automatically generated**