Pakistan Institute of Engineering and Applied Sciences

LAB REPORT

Creating Graph, Traversing and Sorting using Topological Sort

Group no.	**17th**
Group Members	**Muhammad Ali Qadri, Fajar Saleem and Bushra Fatima**
Course	**Data Structures and Algorithms**
Lab no.	**10**
Lab Title	**Creating Graph, Traversing and Sorting using Topological Sort**
Lab Date	**23-December-2023**

Objective

The lab aims to implement and compare Breadth-First Search (BFS) using adjacency matrix and Topological sort using an adjacency list.

Description

The lab involves implementing and comparing two graph traversal algorithms:
Breadth-First Search using adjacency matrix and Topological sort using adjacency list. The goal is to understand the trade-offs between space complexity and performance in graph representation. The practical application of BFS and Topological Sort highlights their significance in addressing real-world graph theory problems, contributing to informed algorithmic choices.

Conclusion

This lab offers valuable insights into graph representation and traversal through different data structures. The comparison between adjacency matrix and adjacency list emphasizes the space complexity and performance considerations. The practical application of BFS and Topological Sort showcases the significance of these data structures in addressing real-world problems related to graph theory. This knowledge contributes to a deeper understanding of algorithmic choices when working with graph-based problems and the impact on computational efficiency.

Lab Task

BFS By Using Adjacency Matrix

Header File

```
#ifndef ADJMATRIX_H
#define ADJMATRIX_H
#include <string>
using namespace std;
class AdjMatrix
{
public:
AdjMatrix();
//Initializing the matrix to zero
AdjMatrix(int noOfVertices);
void addEdge(int r,int c);
virtual ~AdjMatrix();
void BFS(int vertexId);
void Path(int path[]);
void PrintVertices();

private:
```

```
string NameOfVertices[15]={"PIEAS","Chirah","Thanda Paani","Mohara","Ali Pur","Pindi
Begwal","Taramari","Jagiot","Athal","Chatta Bakhtawar","Malot","Bhara Kahu","Kuri","Chak
Shahazad","Bani Gala"};
int** Matrix;
int noOfVertices=10;
};
#endif // ADJMATRIX_H
```

.cpp File

```
#include "AdjMatrix.h"
#include <iostream>
#include <queue>
#include <string>
#include<iomanip>

using namespace std;

AdjMatrix::AdjMatrix()
{
//Nothing Special Has Been Done Here.
}

AdjMatrix::AdjMatrix(int noOfVertices)
{
AdjMatrix();
//Initializing the no of vertices.
this->noOfVertices=noOfVertices;

//Matrix is pointing to the array of pointers to strings.
Matrix = new int*[noOfVertices];

//Matrix[i] is now pointing to the array of strings for i-th row.
for(int i=0;i<noOfVertices;i++)
Matrix[i] = new int[noOfVertices];

//Initializing all the entries in the matrix to zero.
for(int i=0;i<noOfVertices;i++)
for(int j=0;j<noOfVertices;j++)
Matrix[i][j]=0;

}
```

```
void AdjMatrix::addEdge(int r,int c)
{
//Initializing the Matrix[c][r] and Matrix[c][r] to 1 which indicates that an edge exists between r
and c.
Matrix[r][c]=1;
Matrix[c][r]=1;
}

void AdjMatrix::BFS(int vertexId)
{
queue<int> Q;
int path[noOfVertices];
bool flag[noOfVertices];

//Initializing the visited table entries to false and recording the path.
for(int i=0;i<noOfVertices;i++)
{
flag[i]=false;
path[i]=-1;
}

//Declaring that we've visited the source node.
flag[vertexId]=true;
Q.push(vertexId);
while(!Q.empty())
{
//Dequeueing the element to find its adjacent vertices.
vertexId=Q.front();
Q.pop();
cout<<"|"<<NameOfVertices[vertexId]<<"|"<<" -> ";

//Finding all adjacent vertices for vertice specified with "vertexId".
for(int i=0;i<noOfVertices;i++)
{
//Checking adjacent vertices has been visited or not?
if(i!=vertexId && Matrix[vertexId][i]==1 && flag[i]!=true)
{
flag[i]=true;

//Storing the path.
path[i]=vertexId;
//Pushing the vertice on queue.
Q.push(i);
}
}
}
cout<<endl;
}

void AdjMatrix::PrintVertices()
{
cout<<"\n"<<setw(90)<<"===========================\n";
cout<<setw(90)<<"Vertices Of The Graph Are.\n";
cout<<setw(90)<<"===========================\n";
cout<<"\n";
for(int i=0;i<noOfVertices;i++)
{
cout<<"|"<<NameOfVertices[i]<<"|"<<" ";
}
cout<<endl;
}

AdjMatrix::~AdjMatrix()
{
//Nothing is required here.
}
```

Topological Sort By Using Adjacency List
Header File

```
#ifndef ADJLIST_H
#define ADJLIST_H
#include <string>
using namespace std;

struct List
{
int vertex;
int degree=0;
List* next;
};
```

```
class AdjList
{
public:
AdjList();
AdjList(int VerA);
void addEdge(int src, int des);
void TopologicalSort();
void InDegree(void);
void PrintVertices();
virtual ~AdjList();

private:
string NameOfVertices[15]={"PIEAS","Chirah","Thanda Paani","Mohara","Ali Pur","Pindi
Begwal","Taramari","Jagiot","Athal","Chatta Bakhtawar","Malot","Bhara Kahu","Kuri","Chak
Shahazad","Bani Gala"};
int InDegreeOfVertices[15]={0},noOfVertices;
List** adj;
};
#endif // ADJLIST_H
```

.cpp File

```
#include "AdjList.h"
#include <iostream>
#include <queue>
#include <string>
using namespace std;

AdjList::AdjList()
{
//Nothing Special Has Been Done Here.
}

AdjList::AdjList(int VerA) : noOfVertices(VerA)
{
//AdjList();
//Adj is pointing to the array of Lists.
adj = new List*[noOfVertices];
for(int i=0;i<noOfVertices;i++)
{
//Declaring as NULL.
```

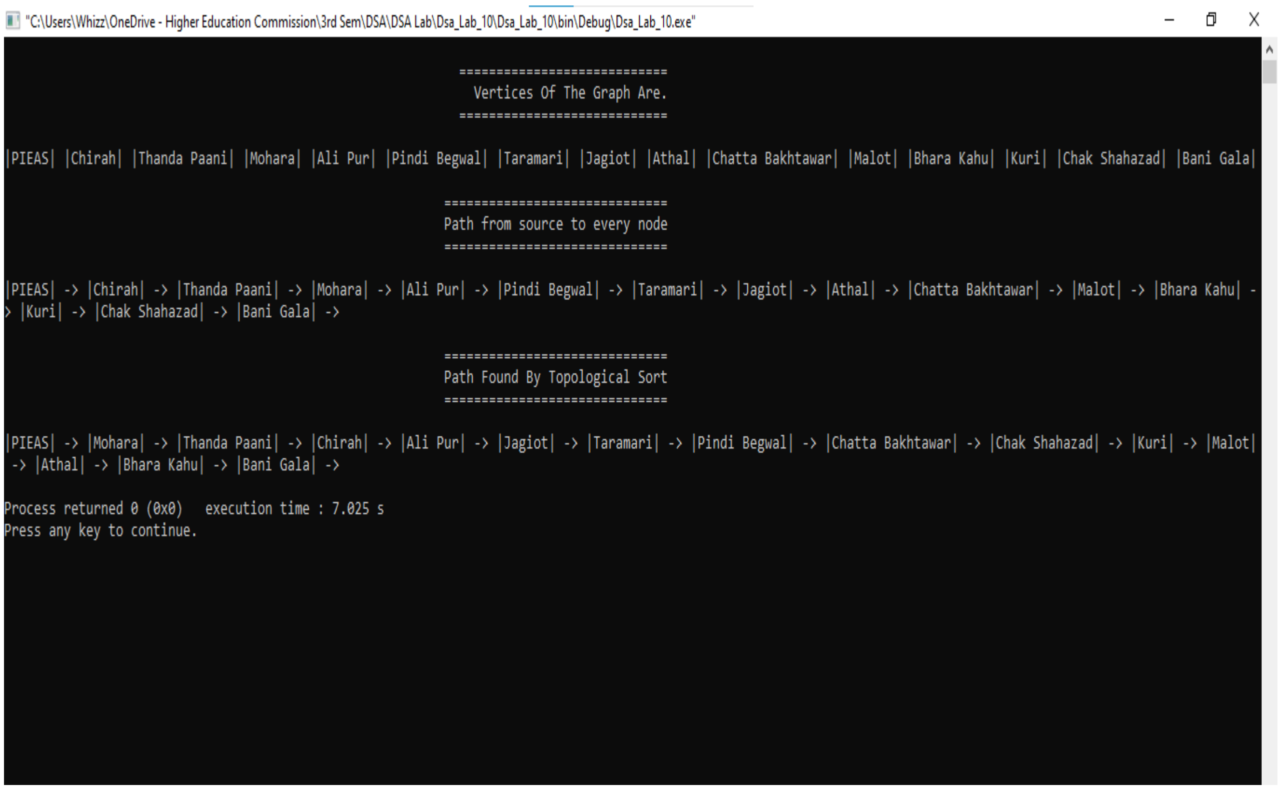
```
adj[i]=NULL;
}
}

void AdjList::addEdge(int src, int des)
{
//Creating a new Node.
List* newNode=new List;

//Initializing the vertex value.
newNode->vertex=des;

//Adding a new node at the head of specified array of list index.
newNode->next=adj[src];
adj[src]=newNode;
}

void AdjList::InDegree(void)
{
for(int i=0;i<noOfVertices;i++)
{
//Finding In Degree of all the vertices.
List* node=adj[i];
while(node!=NULL)
{
//Setting In Degree values for nodes.
InDegreeOfVertices[node->vertex]+=1;
node=node->next;
}
}
}

void AdjList::TopologicalSort()
{
InDegree();
int flag=0;
queue<int> Q;

//Checking which vertex has zero InDegree.
if(flag!=1)
{
for(int i=0;i<noOfVertices;i++)
```

```
{
if(InDegreeOfVertices[i]==0)
//Pushing the vertex which has zero InDegree.
Q.push(i);
}
}


while (!Q.empty())
{
int vertex=Q.front();
Q.pop();
cout<<"|"<<NameOfVertices[vertex]<<"|"<<" -> ";

// Populating the array of Lists to initialize the value of degree.
List* node=adj[vertex];
while (node!=NULL)
{
InDegreeOfVertices[node->vertex]-=1;
if (InDegreeOfVertices[node->vertex]==0)
Q.push(node->vertex);
node=node->next;
}
}
cout<<endl;
}

void AdjList::PrintVertices()
{
cout<<"Vertices Of The Graph Are.\n";
for(int i=0;i<noOfVertices;i++)
{
cout<<"|"<<NameOfVertices[i]<<"|"<<" ";
}
cout<<endl;
}

AdjList::~AdjList()
{
//Freeing the memory.
delete[] adj;
}
```

Main

```
#include <iostream>
#include <AdjMatrix.h>
#include <AdjList.h>
#include<iomanip>
using namespace std;

int main()
{
AdjMatrix M(15);
//Calling addEdge function multiple times to add edges.
M.addEdge(0,1);
M.addEdge(0,2);
M.addEdge(0,3);
M.addEdge(2,4);
M.addEdge(3,5);
M.addEdge(4,6);
M.addEdge(4,7);
M.addEdge(5,7);
M.addEdge(5,8);
M.addEdge(6,9);
M.addEdge(7,10);
M.addEdge(8,10);
M.addEdge(8,11);
M.addEdge(9,12);
M.addEdge(9,13);
M.addEdge(10,12);
M.addEdge(11,14);
M.addEdge(13,14);
M.PrintVertices();

cout<<right<<setw(90)<<"==============================\n";
cout<<right<<setw(90)<<"Path from source to every node\n";
cout<<right<<setw(90)<<"==============================\n";
cout<<endl;
//BFS Traversing the graph for Node '0' (PIEAS) using BFS algorithm.
M.BFS(0);

//Calling parametrized constructor to initialize all values of the matrix to zero.
AdjList G(15);
//Adding edges to the vertices whose are adjacent.
G.addEdge(0,1);
G.addEdge(0,2);
G.addEdge(0,3);
G.addEdge(2,4);
G.addEdge(3,5);
G.addEdge(4,6);
G.addEdge(4,7);
G.addEdge(5,8);
G.addEdge(6,9);
G.addEdge(7,5);
G.addEdge(8,11);
G.addEdge(9,12);
G.addEdge(9,13);
G.addEdge(10,8);
G.addEdge(11,14);
G.addEdge(12,10);
G.addEdge(13,14);

cout<<endl;
cout<<right<<setw(90)<<"==============================\n";
cout<<right<<setw(90)<<"Path Found By Topological Sort\n";
cout<<right<<setw(90)<<"==============================\n";
cout<<endl;
//BFS Traversing the graph for Node '0' (PIEAS) using BFS algorithm.
G.TopologicalSort();


return 0;
}
```

Output:

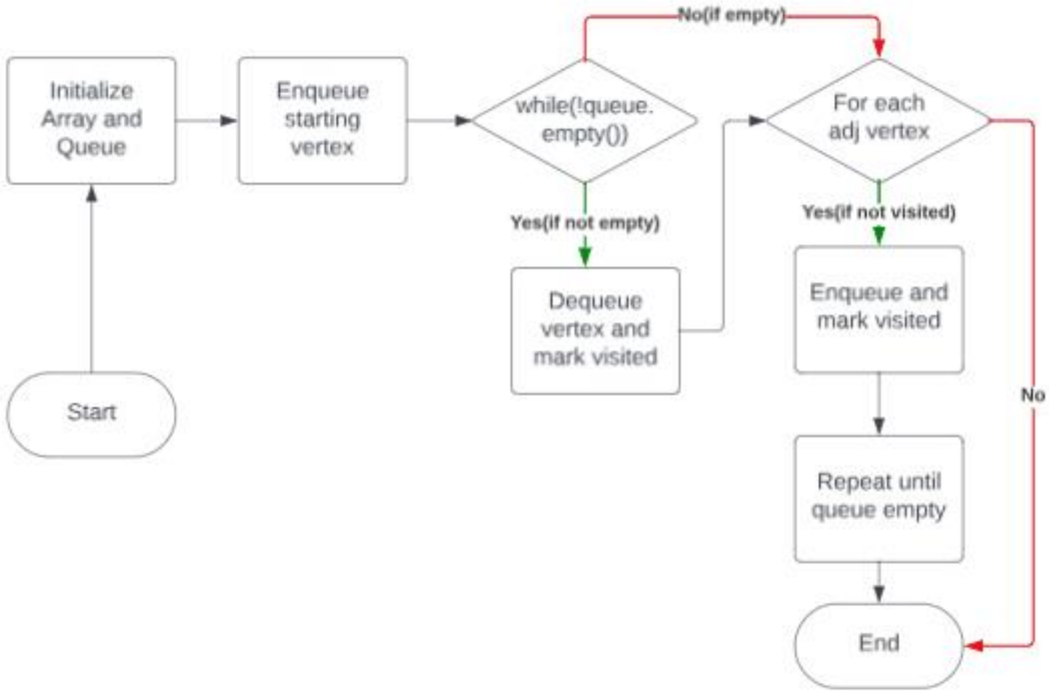


"C:\Users\Whizz\OneDrive - Higher Education Commission\3rd Sem\DSA\DSA Lab\Dsa_Lab_10\Dsa_Lab_10\bin\Debug\Dsa_Lab_10.exe"

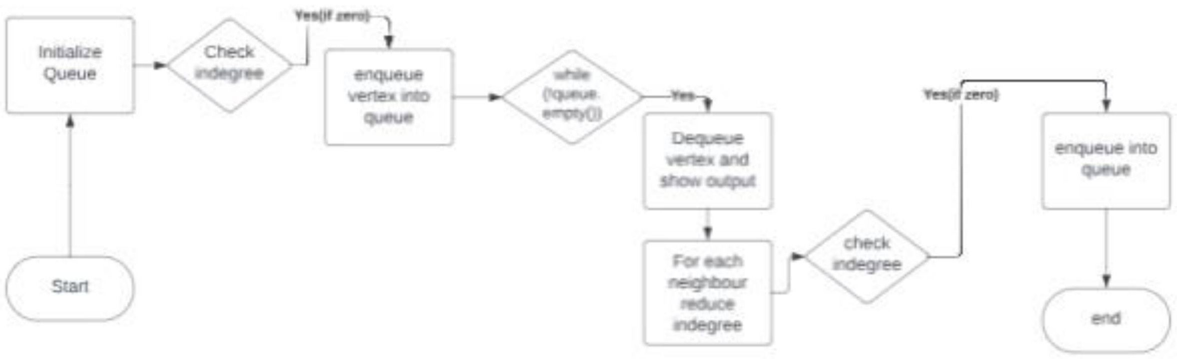
```
                                                              ============================
                                                                Vertices Of The Graph Are.
                                                              ============================

|PIEAS| |Chirah| |Thanda Paani| |Mohara| |Ali Pur| |Pindi Begwal| |Taramari| |Jagiot| |Athal| |Chatta Bakhtawar| |Malot| |Bhara Kahu| |Kuri| |Chak Shahazad| |Bani Gala|

                                                            ==============================
                                                            Path from source to every node
                                                            ==============================

|PIEAS| -> |Chirah| -> |Thanda Paani| -> |Mohara| -> |Ali Pur| -> |Pindi Begwal| -> |Taramari| -> |Jagiot| -> |Athal| -> |Chatta Bakhtawar| -> |Malot| -> |Bhara Kahu| -
> |Kuri| -> |Chak Shahazad| -> |Bani Gala| ->

                                                            ==============================
                                                            Path Found By Topological Sort
                                                            ==============================

|PIEAS| -> |Mohara| -> |Thanda Paani| -> |Chirah| -> |Ali Pur| -> |Jagiot| -> |Taramari| -> |Pindi Begwal| -> |Chatta Bakhtawar| -> |Chak Shahazad| -> |Kuri| -> |Malot|
 -> |Athal| -> |Bhara Kahu| -> |Bani Gala| ->

Process returned 0 (0x0)   execution time : 7.025 s
Press any key to continue.
```

Flowcharts

BFS



Topological Sort


Yes(if zero)
Initialize Queue
Check indegree
enqueue vertex into queue
while (!queue. empty())
Yes
Yes(if zero)
Dequeue vertex and show output
enqueue into queue
Start
For each neighbour reduce indegree
check indegree
end