KATHMANDU UNIVERSIY

REPORT : LAB 6

Submitted By:

Name: Bikraj Shrestha
Roll:50

Name: Janak Sitaula
Roll:51

Group : CE
2nd Year/1st Sem

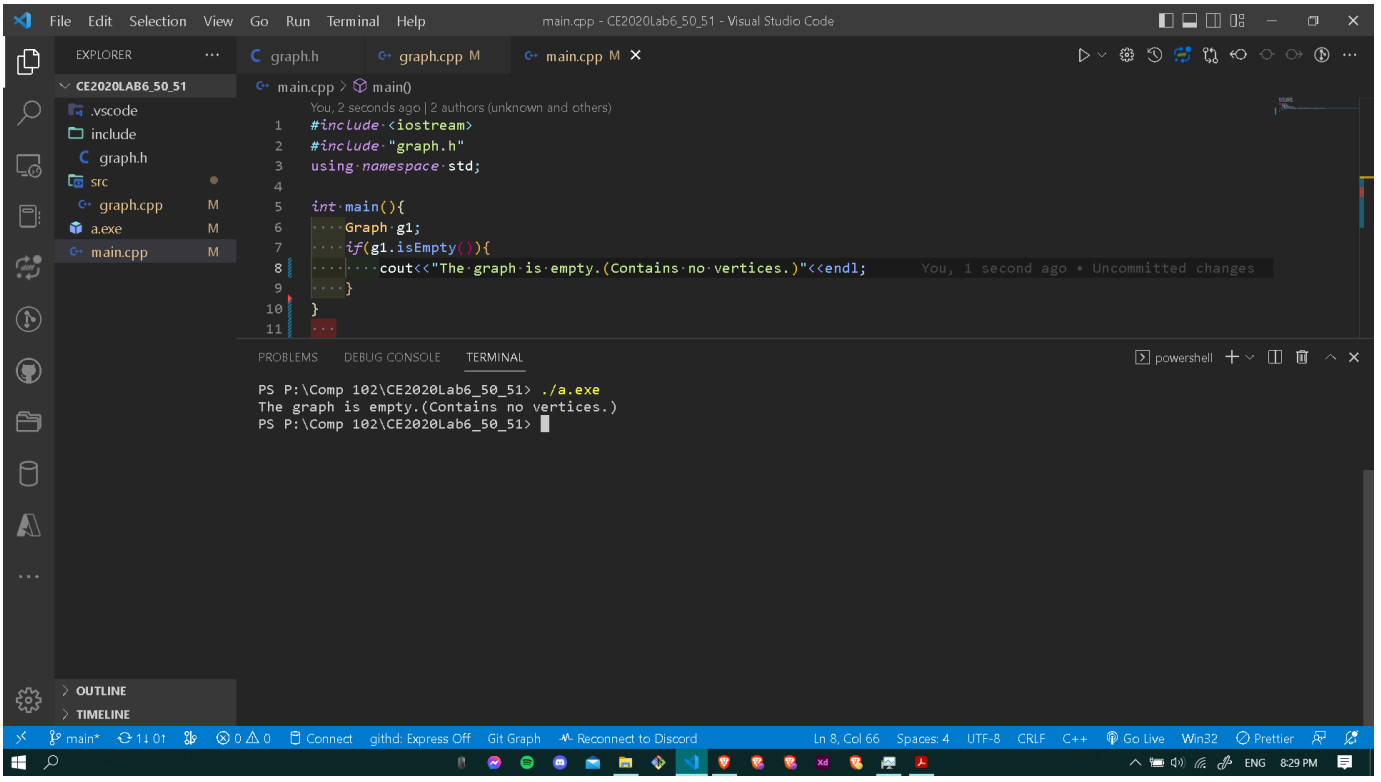
Submitted To:

Dr. Rajani Chulyadyo
Undergraduate Coordinator CE Program
Department of Computer Science and Engineering

GRAPH

1.isEmpty()

This function checks if the graph is empty.It checks the size of the vertices by counting all the vertex in the vertices.



2.isDirected()

This function checks if the graph is directed. The member variable called isDirected is returned that is responsible for tracking if the graph is directed or not.

graph.h | graph.cpp M | main.cpp M

src > graph.cpp > addVertex(char)

```cpp
        count++;
    }
    return count == 0 ? true : false;
}
bool Graph::isDirected()
{
    return isDirectedGraph;
}

void Graph::addVertex(char newVertex)
{
    Vertex vertex;
    vertex.name = newVertex;
    vertex.position = vertices.size();      sitjan, 2 days ago • added …
    vertices.push_back(vertex);
    if (data.size() == 0)
    {
        data.push_back(vector<int>(1, 0)); // vector 1 = {1,23,4,5,6,4}//vector.pushback(4)
    }
```

PROBLEMS | DEBUG CONSOLE | TERMINAL | powershell

```
^
PS P:\Comp 102\CE2020Lab6_50_51> ./a.exe
The graph is empty
  A
A 0,
The graph is not directed
The graph is not empty
  A
A 0,
  A B C
A 0,0,0,
B 0,0,0,
```

Connect | githd: Express Off | Git Graph | Reconnect to Discord | Ln 26, Col 39 | Spaces: 4 | UTF-8 | CRLF | C++ | Go Live | Win32

3.addVertex(char vertex1)

The function adds a vertex to the graph. At first the vertex is added to the vertices vector that is responsible for tracking the vertices. Then the size of the data vector is increased by one (initialized with zeros.). The vertex added to the vertices has a name member that stores the name of the vertex. For eg: 'A'.

graph.h M graph.cpp M main.cpp M

main.cpp > main()

```cpp
int main(){
    Graph g1;
    if(g1.isEmpty()){
        cout<<"The graph is empty.(Contains no vertices.)"<<endl;
    }
    if(g1.isDirected()){
        cout<<"The graph is directed."<<endl;
    }else{
        cout<<"The graph is not directed."<<endl;
    }
    g1.addVertex('A');
}
```

PROBLEMS DEBUG CONSOLE TERMINAL powershell

```
PS P:\Comp 102\CE2020Lab6_50_51> ./a.exe
The graph is empty.(Contains no vertices.)
The graph is not directed.
PS P:\Comp 102\CE2020Lab6_50_51>
```

4. addEdge(vertex1, vertex2): Adds an edge from vertex1 to vertex2

This function adds edge in the graph. The added graph will have value on its adjacency matrix (can be seen on the output matrix).

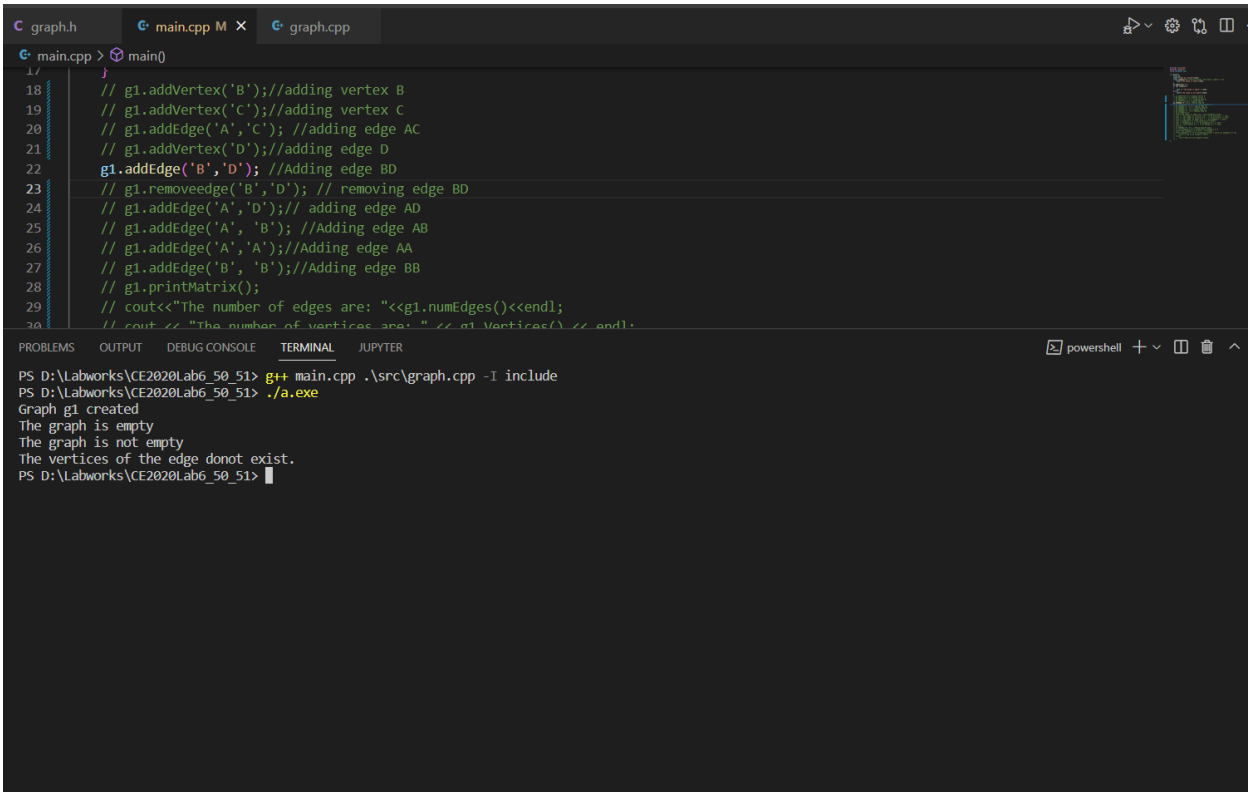


Figure 1 Here the output says that the vertices of the edge do not exist because we have not added any vertex and directly tried to add an edge which is not possible

5. removeVertex(vertexToRemove): Remove a vertex from the graph

This function removes the desired vertex from the graph. This can be seen when all the corresponding edges are removed and denoted by zero in the matrix. The value for any assigned vertex will be assigned as ‘z’.

6. removeEdge(vertex1, vertex2): Remove an edge from the graph

This function removes edge from the graph. This is done by assigning zeroes to the desired edges corresponding elements in the adjacency matrix.

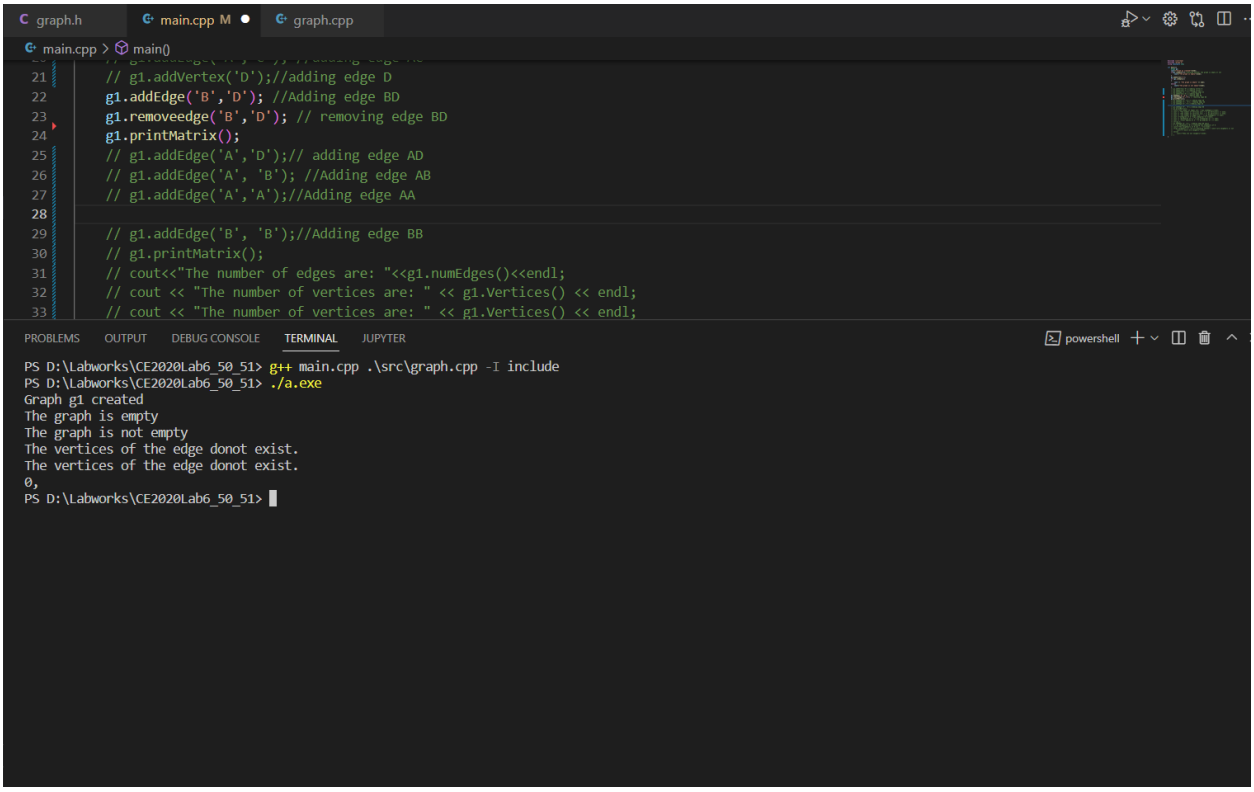


Figure 2 Here the output says the vertices of the edge do not exist as we have not added any vertices but tried to add and remove an edge

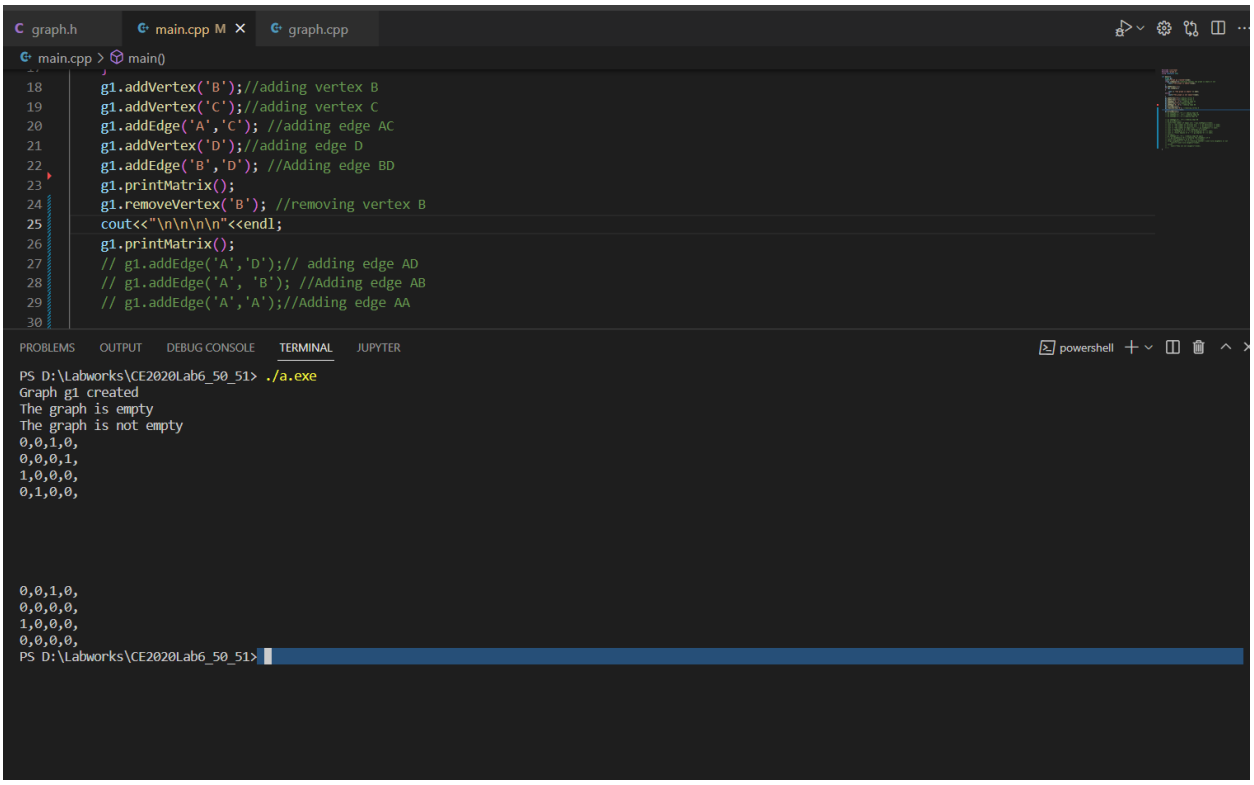


Figure 3 Here we can see as the vertex and edge have been added and removed, the corresponding matrix show their existence with 1's and 0's

7. numVertices(): Returns the number of vertices in the graph

This function returns the total number of vertices in the graph. This is done by counting the total number of alphabets in the data. This excludes ‘z’, which is assigned for any removed vertex in the graph.

8. numEdges(): Returns the number of edges in the graph

This function returns the total number of edges in the graph. This is performed by calculating the total number of 1’s in the adjacency matrix in the data.

```
cout<<"\n\n\n\n"<<endl;
g1.printMatrix();
g1.removeedge('A','C');
g1.addEdge('A','D');// adding edge AD
g1.addEdge('A', 'B'); //Adding edge AB
g1.addEdge('A','A');//Adding edge AA
g1.addEdge('B', 'B');//Adding edge BB
g1.printMatrix();
cout<<"The number of edges are: "<<g1.numEdges()<<endl;
cout << "The number of vertices are: " << g1.Vertices() << endl;
cout << "The number of vertices are: " << g1.Vertices() << endl;
// cout << "The number of edges are: " << g1.numEdges() << endl;
// cout <<"InDegree of A "<<g1.indegree('A')<<endl;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER powershell

```
Graph g1 created
The graph is empty
The graph is not empty
0,0,1,0,
0,0,0,1,
1,0,0,0,
0,1,0,0,




0,0,1,0,
0,0,0,0,
1,0,0,0,
0,0,0,0,
The vertices of the edge donot exist.
The vertices of the edge donot exist.
1,0,0,1,
0,0,0,0,
0,0,0,0,
1,0,0,0,
The number of edges are: 3
The number of vertices are: 3
The number of vertices are: 3
PS D:\Labworks\CE2020Lab6_50_51>
```

Figure 4 Here we can see all the functions in work together with their respective adjacency matrices.

9.inDegree()

The function returns the number of the edge that are coming out of the vertex. A loop runs to check all the 1's present in the row of the adjacency matrix. And that value is returned.

graph.h | graph.cpp M | main.cpp M

src > graph.cpp > isEmpty()

```cpp
        }
    }
    return neighbours;
}
int Graph::indegree(char vertex1)
{
    int vertex1Posi = findPostion(vertex1);
    int indegree = 0;
    for (int i = 0; i < vertices.size(); i++)
    {
        if (data[i][vertex1Posi] == 1)
        {
            indegree++;
        }
    }
    return indegree;
}
```

PROBLEMS DEBUG CONSOLE TERMINAL

```
B 0,0,0,
C 0,0,0,
The size of hte verteiasdf3
The number of edges are: 8
The number of vertices are: 4
The number of vertices are: 4
The number of edges are: 8
InDegree of A 4
OutDegree of A 4
Total degree of A 8
The neighbours of B are :: AD
They are not neigbors
PS P:\Comp 102\CF2020Lab6_50_51>
```

9.outDegree()

The function returns the number of the edge that are coming out of the vertex. A loop runs to check all the 1's present in the column of the adjacency matrix. And that value is returned.

graph.h | graph.cpp M | main.cpp M

src > graph.cpp > isEmpty()

```cpp
        }
    }
    return indegree;
}
int Graph::outdegree(char vertex1)//function to calculate the outdegree of the vertex
{
    int vertex1Posi = findPostion(vertex1);
    int outdegree = 0;
    for (int i = 0; i < vertices.size(); i++)
    {
        if (data[vertex1Posi][i] == 1)
        {
            outdegree++;
        }
    }
    return outdegree;
}
```

PROBLEMS | DEBUG CONSOLE | TERMINAL

```
B 0,0,0,
C 0,0,0,
The size of hte verteiasdf3
The number of edges are: 8
The number of vertices are: 4
The number of vertices are: 4
The number of edges are: 8
InDegree of A 4
OutDegree of A 4
Total degree of A 8
The neighbours of B are :: AD
They are not neigbors
PS P:\Comp 102\CE2020Lab6_50_51>
```

11.degree()char vertex)

The function returns the sum of indegree and outdegree of the vertex.

graph.h | graph.cpp M | main.cpp M

src > graph.cpp > isEmpty()

```
        }
    }
    return outdegree;
}

int Graph::degree(char vertex1)
{
    return indegree(vertex1) + outdegree(vertex1);
}
void Graph:: printMatrix(){//function to caluclate the total degree of the function

    cout<<"  ";
    for (int i = 0; i < vertices.size(); i++)
    {

        if (vertices[i].name == 'z')
        {
            continue;
```

PROBLEMS DEBUG CONSOLE TERMINAL powershell

```
B 0,0,0,
C 0,0,0,
The size of hte verteiasdf3
The number of edges are: 8
The number of vertices are: 4
The number of vertices are: 4
The number of edges are: 8
InDegree of A 4
OutDegree of A 4
Total degree of A 8
The neighbours of B are :: AD
They are not neigbors
PS P:\Comp 102\CE2020Lab6_50_51>
```

12.neighbors(char vertex1)

The function returns all the neighbors of the vertex. The row and the column of the given vertex is checked if any of them have 1. If 1 is found then , it is a neighbors and the value is returned.

graph.h | graph.cpp M | main.cpp M

src > graph.cpp > isEmpty()

```cpp
    }
    return false;
}
string Graph::neighbours(char vertex1)
{
    // check all the posible position of that row and column of the vertex if they contain 1
    int vertex1Posi = findPostion(vertex1);
    string neighbours = "";
    for (size_t i = 0; i < data.size(); i++)
    {
        if (i == vertex1Posi)
        {
            continue;
        }
        else if (data[i][vertex1Posi] == 1 || data[vertex1Posi][i] == 1)
        {
            neighbours = neighbours + vertices[i].name;
        }
```

PROBLEMS DEBUG CONSOLE TERMINAL powershell

```
B 0,0,0,
C 0,0,0,
The size of hte verteiasdf3
The number of edges are: 8
The number of vertices are: 4
The number of vertices are: 4
The number of edges are: 8
InDegree of A 4
OutDegree of A 4
Total degree of A 8
The neighbours of B are :: AD
They are not neigbors
PS P:\Comp 102\CE2020Lab6_50_51>
```

13.neigbors(char vertex1 , char vertex2)

This function checks if the an edge runs through the vertices. For this the possible edge i.e veretx1Vertex2 or vertex2Vertex1 is checked. If it exists then they are neigbors.

graph.h | graph.cpp M | main.cpp M

main.cpp > main()

```
    cout << "The number of edges are: " << g1.numEdges() << endl;
    cout << "The number of vertices are: " << g1.Vertices() << endl;
    cout << "The number of vertices are: " << g1.Vertices() << endl;
    cout << "The number of edges are: " << g1.numEdges() << endl;
    g1.printMatrix();
    cout << "InDegree of A " << g1.indegree('A') << endl;
    cout << "OutDegree of A " << g1.outdegree('A') << endl;
    cout << "Total degree of A " << g1.degree('A') << endl;
    string n;
    g1.addEdge('B', 'D');
    n = g1.neighbours('B');
```

You, now • Uncommitted changes

PROBLEMS | DEBUG CONSOLE | TERMINAL | powershell

```
  A B C
A 0,0,0,
B 0,0,0,
C 0,0,0,
The size of hte verteiasdf3
The number of edges are: 8
The number of vertices are: 4
The number of vertices are: 4
The number of edges are: 8
  A B C D
A 1,1,1,1,
B 1,1,0,0,
C 1,0,0,0,
D 1,0,0,0,
InDegree of A 4
OutDegree of A 4
Total degree of A 8
The neighbours of B are :: AD
They are not neigbors
PS P:\Comp 102\CE2020Lab6_50_51>
```