Bahria University, Islamabad

Department of Software Engineering

Data Structre And Algorithms

(Fall-2024)

Teacher: Engr. Aleem Ahmad

Student : Lotfullah Muslimwal

Enrollment : 01-131232-039

Lab Journal: X

Date:

Task No:	Task Wise Marks		Documentation Marks		Total Marks (20)
	Assigned	Obtained	Assigned	Obtained	
1	3		5		
2	3				
3	3				
4	3				
5	3				

Comments:

Signature

LAB

Lab No: 10

Task 1: Graph Management and Traversal System

Code:

```
#include <iostream>
#include <stack>
#include <limits>
using namespace std;

struct Vertex {
   int value;
   Vertex* next;
   struct Edge* edges;
   bool visited;
};

struct Edge {
   Vertex* connectedVertex;
   Edge* next;
};

class Graph {
   Vertex* head;

public:
   Graph() : head(nullptr) {}

   void addVertex(int value);
   void removeVertex(int value);
   void addEdge(int from, int to);
   void removeEdge(int from, int to);
   void showAdjacentVertices(int value);
   bool isGraphEmpty();
   Vertex* findVertex(int value);
   void performDFS(int startValue);
   void resetVisitedStatus();
   void displayGraphStructure();
};

void Graph::addVertex(int value) {
   if (findVertex(value)) {
```

```
        cout << "Vertex " << value << " already exists!\n";
        return;
    }

    Vertex* newVertex = new Vertex();
    newVertex->value = value;
    newVertex->next = nullptr;
    newVertex->edges = nullptr;
    newVertex->visited = false;

    if (!head) {
        head = newVertex;
    }
    else {
        Vertex* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newVertex;
    }
    cout << "Vertex " << value << " added successfully.\n";
}

void Graph::removeVertex(int value) {
    Vertex* current = head;
    Vertex* prev = nullptr;

    while (current && current->value != value) {
        prev = current;
        current = current->next;
    }

    if (!current) {
        cout << "Vertex " << value << " not found.\n";
        return;
    }

    Vertex* temp = head;
    while (temp) {
        removeEdge(temp->value, value);
        temp = temp->next;
    }

    if (prev) {
```

```
        prev->next = current->next;
    }
    else {
        head = current->next;
    }
    delete current;
    cout << "Vertex " << value << " deleted successfully.\n";
}

void Graph::addEdge(int source, int destination) {
    Vertex* sourceVertex = findVertex(source);
    Vertex* destinationVertex = findVertex(destination);

    if (!sourceVertex || !destinationVertex) {
        cout << "One or both vertices not found!\n";
        return;
    }

    Edge* newEdge = new Edge{ destinationVertex, nullptr };
    if (!sourceVertex->edges) {
        sourceVertex->edges = newEdge;
    }
    else {
        Edge* temp = sourceVertex->edges;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newEdge;
    }

    cout << "Edge added between " << source << " and " << destination << ".\n";
}

void Graph::removeEdge(int source, int destination) {
    Vertex* sourceVertex = findVertex(source);

    if (!sourceVertex) {
        cout << "Vertex " << source << " not found.\n";
        return;
    }

    Edge* current = sourceVertex->edges;
    Edge* prev = nullptr;
```

```
    while (current && current->connectedVertex->value != destination) {
        prev = current;
        current = current->next;
    }

    if (!current) {
        cout << "Edge between " << source << " and " << destination << " not found.\n";
        return;
    }

    if (prev) {
        prev->next = current->next;
    }
    else {
        sourceVertex->edges = current->next;
    }
    delete current;
    cout << "Edge between " << source << " and " << destination << " deleted successfully.\n";
}

void Graph::showAdjacentVertices(int value) {
    Vertex* vertex = findVertex(value);

    if (!vertex) {
        cout << "Vertex " << value << " not found.\n";
        return;
    }

    cout << "Adjacent vertices of " << value << ": ";
    Edge* edge = vertex->edges;
    while (edge) {
        cout << edge->connectedVertex->value << " ";
        edge = edge->next;
    }
    cout << endl;
}

bool Graph::isGraphEmpty() {
    return head == nullptr;
}

Vertex* Graph::findVertex(int value) {
    Vertex* temp = head;
    while (temp) {
```

```
        if (temp->value == value) {
            return temp;
        }
        temp = temp->next;
    }
    return nullptr;
}

void Graph::performDFS(int startValue) {
    Vertex* startVertex = findVertex(startValue);
    if (!startVertex) {
        cout << "Starting vertex not found!\n";
        return;
    }

    stack<Vertex*> s;
    s.push(startVertex);

    cout << "DFS Traversal: ";
    while (!s.empty()) {
        Vertex* current = s.top();
        s.pop();

        if (!current->visited) {
            cout << current->value << " ";
            current->visited = true;

            Edge* edgeTemp = current->edges;
            while (edgeTemp) {
                if (!edgeTemp->connectedVertex->visited) {
                    s.push(edgeTemp->connectedVertex);
                }
                edgeTemp = edgeTemp->next;
            }
        }
    }
    cout << endl;

    resetVisitedStatus();
}

void Graph::resetVisitedStatus() {
    Vertex* temp = head;
    while (temp) {
```

```
        temp->visited = false;
        temp = temp->next;
    }
}

void Graph::displayGraphStructure() {
    Vertex* temp = head;
    cout << "Graph adjacency list:\n";
    while (temp) {
        cout << temp->value << ": ";
        Edge* edgeTemp = temp->edges;
        while (edgeTemp) {
            cout << edgeTemp->connectedVertex->value << " ";
            edgeTemp = edgeTemp->next;
        }
        cout << endl;
        temp = temp->next;
    }
}

int getValidInt() {
    int value;
    while (!(cin >> value)) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid input. Please enter an integer: ";
    }
    return value;
}

int main() {
    Graph g;
    int choice, value, source, destination, startValue;

    do {
        cout << "\n\n============================\n";
        cout << "          MENU OPTIONS      \n";
        cout << "============================\n";
        cout << "1. Add Vertex\n";
        cout << "2. Remove Vertex\n";
        cout << "3. Add Edge\n";
        cout << "4. Remove Edge\n";
        cout << "5. Show Adjacent Vertices\n";
        cout << "6. Check if Graph is Empty\n";
```

```
cout << "7. Display Graph\n";
cout << "8. Perform DFS\n";
cout << "9. Exit\n";
cout << "==============================\n";
cout << "Enter your choice: ";
choice = getValidInt();

switch (choice) {
case 1:
   cout << "Enter vertex value: ";
   value = getValidInt();
   g.addVertex(value);
   break;
case 2:
   cout << "Enter vertex value to remove: ";
   value = getValidInt();
   g.removeVertex(value);
   break;
case 3:
   cout << "Enter source vertex of edge: ";
   source = getValidInt();
   cout << "Enter destination vertex of edge: ";
   destination = getValidInt();
   g.addEdge(source, destination);
   break;
case 4:
   cout << "Enter source vertex of edge you want to remove: ";
   source = getValidInt();
   cout << "Enter destination vertex of edge you want to remove: ";
   destination = getValidInt();
   g.removeEdge(source, destination);
   break;
case 5:
   cout << "Enter vertex to find adjacent vertices: ";
   value = getValidInt();
   g.showAdjacentVertices(value);
   break;
case 6:
   cout << (g.isGraphEmpty() ? "Graph is empty." : "Graph is not empty.") << endl;
   break;
case 7:
   g.displayGraphStructure();
   break;
case 8:
```

```
            cout << "Enter starting vertex for DFS: ";
            startValue = getValidInt();
            g.performDFS(startValue);
            break;
        case 9:
            cout << "Exiting program...\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 9);

    return 0;
}
```

GitHub-Link: https://github.com/lotfullahmsl/DSA-Lab-FA2024

Screenshot: