INFORMATION
TECHNOLOGY
UNIVERSITY

Software Engineering Department - ITU
SE200L: Data Structures & Algorithms Lab

Course Instructor: Usama Bin Shakeel	**Dated: 02/11/2024**
Teaching Assistant: Zainab Bashir & Ryan Naveed	**Semester: Fall 2024**
Lab Engineer: Sadia Ijaz	**Batch: BSSE2023B**

Lab 9. Basic Implementation of Graphs using Linked List

Name	Roll number	Report (out of 35)

Checked on: ______________________________

Signature: ________________________________

1.1 Objective

The objective of this lab is to practice problems related to Basic Implementation of Graphs.

1.2 Equipment and Component

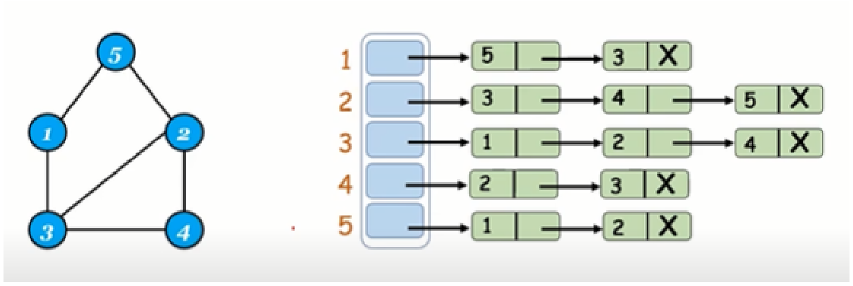
Component Description	Value	Quantity
Computer	Available in lab	1

1.3 Conduct of Lab

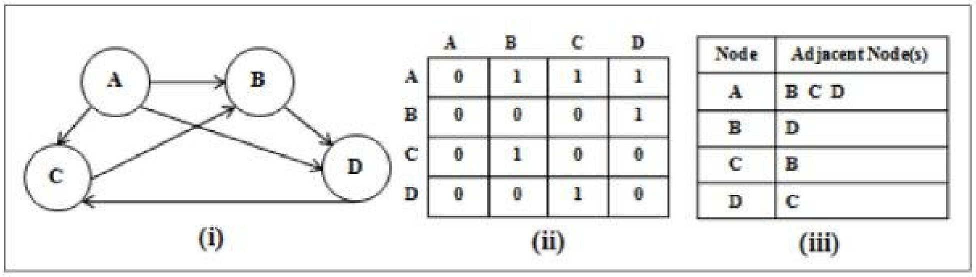
1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

1.4 Theory and Background

A graph data structure is a collection of nodes, or vertices, connected by edges. Think of Facebook: everything User, Photo, Event, Group is a node, and every interaction (like posting or joining) creates an edge between nodes. Formally, a graph is represented as (V, E), where V is the set of vertices and E is the set of edges connecting pairs of vertices.



A **directed** graph is graph, i.e., a set of objects (called vertices or nodes) that are connected together, where all the edges are directed from one vertex to another. A directed graph is sometimes called a digraph or a directed network.



	A	B	C	D
A	0	1	1	1
B	0	0	0	1
C	0	1	0	0
D	0	0	1	0

(ii)

Node	Adjacent Node(s)
A	B C D
B	D
C	B
D	C

(iii)

Lab Tasks

Task 1:

Implement the **Edge** class

Data Members:

- **int destination**: The value of the destination node this edge points to.
- **int cost**: The cost associated with traversing this edge.
- ***Edge next**:** A pointer to the next edge in the linked list of edges originating from the same source node.

Member Functions to Implement:

- **Edge(int dest, int c)**: Constructor that initializes an edge with a destination and a cost.
- **int getDestination()**: Returns the destination of the edge.
- **int getCost()**: Returns the cost of the edge.
- ***Edge** getNext():** Returns the pointer to the next edge.
- ***void setNext(Edge** nextEdge):** Sets the pointer to the next edge in the list.
- **void setDestination(int dest)**: Updates the destination of the edge.

Implement the **Node** class.

Data Members:

- **int data**: The value of the node.
- ***Node next**:** A pointer to the next node in the linked list of nodes.
- ***Edge edges**:** A pointer to the head of the linked list of edges connected to this node.
- **int edgeCount**: The number of edges connected to this node.

Member Functions to Implement:

- **Node(int val)**: Constructor that initializes a node with a specified value and sets the edges and next node pointers to nullptr while setting edgeCount to zero.
- **int getData()**: Returns the value stored in the node.
- ***Node** getNext():** Returns the pointer to the next node in the linked list.
- ***Edge** getEdges():** Returns the pointer to the head of the edges list connected to this node.
- **int getEdgeCount()**: Returns the number of edges associated with the node.
- ***void setNext(Node** nextNode):** Sets the pointer to the next node in the linked list.
- ***void setEdges(Edge** edgeList):** Assigns a new linked list of edges to this node.

- **void setData(int value)**: Updates the value stored in the node.
- **void incrementEdgeCount()**: Increments the edge count by one, called when a new edge is added.
- **void decrementEdgeCount()**: Decrements the edge count by one, called when an edge is removed.

Task 2:

Class: Graph

Data Members:

- ***Node nodes****: A pointer to the head of the linked list of nodes in the graph.
- **int nodeCount**: The total number of nodes currently in the graph.

Member Functions to Implement:

- **Graph()**: Constructor that initializes an empty graph by setting nodes to nullptr and nodeCount to zero.
- **~Graph()**: Destructor that cleans up memory by deleting all nodes and their associated edges to prevent memory leaks.
- **void addNode(int nodeValue)**: Adds a new node with the specified value to the graph, avoiding duplicates. It creates a new node and links it at the front of the list.
- **void addEdge(int source, int destination, int cost)**: Adds a directed edge from the source node to the destination node with an associated cost, ensuring both nodes exist and no duplicate edge exists.
- **int getEdgeCost(int source, int destination)**: Retrieves the cost of the edge from the source node to the destination node. Returns -1 if no such edge exists.
- **int getEdgeCountForNode(int nodeValue)**: Returns the number of edges connected to the specified node. Returns -1 if no such node exists
- **int getNodeCount()**: Returns the total number of nodes in the graph.
- **void updateNode(int oldValue, int newValue)**: Updates the value of a node from oldValue to newValue only if newValue node doesn't exist. It also updates any edges pointing to oldValue to point to newValue.
- **void deleteNode(int nodeValue)**: Removes a node from the graph and deletes all edges connected to it, ensuring that the graph maintains integrity.
- **bool hasNode(int nodeValue)**: Checks if a node with the specified value exists in the graph.
- **void deleteEdge(int source, int destination)**: Removes the directed edge from the source node to the destination node, if it exists.
- **void display()**: Displays the entire graph, including each node's value, its edges, and the costs associated with those edges.

Please read the following instructions carefully:

1. ***Do Not Modify test.cpp:*** *You are strictly prohibited from making any changes to the test.cpp file. This file is designed to test your implementation and any modifications will lead to the assignment being graded as zero.*
2. ***Class Definitions:*** *All class definitions and implementations must be provided solely within the files functions.h and functions.cpp. You are not allowed to create any additional files for your class definitions or implementations.*

Any deviation from these rules, including creating additional files or modifying the test.cpp file, will result in your assignment receiving a grade of zero.

Assessment Rubric for Lab

Performance metric	CLO	Able to complete the task over 80% (4-5)	Able to complete the task 50-80% (2-3)	Able to complete the task below 50% (0-1)	Marks
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. The testing has been completed.	Executes without errors, user prompts are understandable,minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non- existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answer all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in an effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and do help the reader to understand the code.	Code comments are added and do not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					