Department of Computer and Software Engineering – ITU
SE200T: Data Structures & Algorithms

Course Instructor: Usama Bin Shakeel	**Dated: 30th Oct 2024**
Teaching Assistant: Zainab, Sadia & Ryan	**Semester: Fall 2024**
Session: 2024-2028	**Batch: BSSE2023B**

Assignment 10. Finding Shortest Paths with Dijkstra's Algorithm

Name	Roll number	Obtained Marks/35

Checked on: ______________________________

Signature: _____________________________

Submission:

- Email instructor or TA if there are any questions. You cannot look at others' solutions or use others' solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.
- Submission after due time will not be accepted.

In this assignment you have to do following tasks:

Task 1: Ensure that you have installed all three softwares in your personal computer (Github, Cygwin & CLion). Now, accept the assignment posted in the classroom (e.g Google, LMS etc) and after accepting, clone the repository to your computer. Make sure you have logged into the github app with your account.

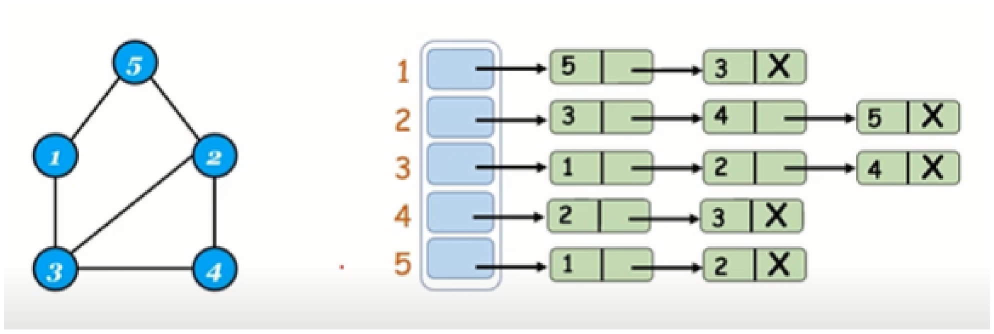
Task 2: Open Cygwin app, Move to your code directory with following command "cd <path_of_folder>", <path_of_folder> can be automatically populated by dragging the folder and dropping it to the cygwin window.
Run the code through Cygwin, use command "make run", to get the output of the code

Task 3: Solve the given problems, write code using **CLion** or any other IDE.
Task 4: Keep your code in the respective git cloned folder.
Task 5: Commit and Push the changes through the Github App
Task 5: Write the code in separate files **(as instructed**). Ensure that file names are in lowercase (e,g **main.cpp**).

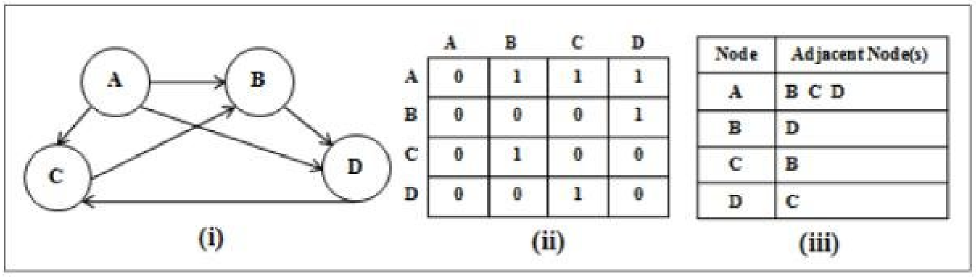
Task 6: Run '**make run**' to run C++ code
Task 7: Run '**make test**' to test the C++ code
Write code in functions, after completing each part, verify through running code using **"make run"** on Cygwin. Make sure to test the code using **"make test".**

Theory and Background

In computer science, A **graph** data structure is a collection of nodes that have data and are connected to other nodes. Let's try to understand this through an example. On Facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, note...anything that has data is a node. Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship. More precisely, a graph is a data structure (V, E) that consists of a collection of vertices V and a collection of edges E, represented as ordered pairs of vertices (u, v).



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.



(i)

	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)

A **pointer** is a variable that stores the address of another variable. Unlike other variables that hold values of a certain type, pointer holds the address of a variable. For example, an integer variable holds (or you can say stores) an integer value, however an integer pointer holds the address of a integer variable.

A **linked list** is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence.

Overview

This assignment implements a directed graph using linked lists. The graph consists of nodes and directed edges that connect them. Each node can have multiple edges pointing to other nodes, and the graph supports operations like adding/removing nodes and edges, updating nodes, and checking connections.Additionally, the graph includes a method for calculating the shortest path between two nodes using Dijkstra's algorithm.

Class Descriptions

1. Class: Edge

Purpose: Represents a directed edge in the graph.

Private Attributes:

- **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.

Public Methods:

- **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.

2. Class: Node

Purpose: Represents a node in the graph.

Private Attributes:

- **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.

Public Methods:

- **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.

3. Class: Graph

Purpose: Represents the directed graph, managing nodes and edges.

Priavte Attributes:

- ***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.

Methods:

- **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.

- **int dijkstra(int src, int dest)**: Implements Dijkstra's algorithm to find the shortest distance from the source node to the destination node. Returns the shortest distance if a path exists; otherwise, returns -1.

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.

INFORMATION TECHNOLOGY UNIVERSITY

Assessment Rubric for Assignment

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	3	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	2	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	4	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	4	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	2	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	2	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	3	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):					