INFORMATION TECHNOLOGY UNIVERSITY

Department of Computer and Software Engineering – ITU

SE200T: Data Structures & Algorithms

Course Instructor: Usama Bin Shakeel	**Dated: 15th Nov 2024**
Teaching Assistant: Zainab, Sadia & Ryan	**Semester: Fall 2024**
Session: 2024-2028	**Batch: BSSE2023B**

Assignment 12. Implementation of KruskalAlgorithm

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".**

Objective

In this assignment, you are required to implement Kruskal's algorithm using C++. You will be given a graph consisting of nodes and edges, and your task is to find the Minimum Spanning Tree (MST) of the graph using Kruskal's algorithm. The graph will be implemented using linked lists for both nodes and edges, and a Union-Find data structure will be used to detect cycles and manage connected components efficiently.

Classes Overview:

1. Class: Edge

- **Purpose**: Represents an edge in the graph, connecting two nodes with a specified weight (cost).
- **Attributes**:
  - **`int source`:** The source node of this edge.
  - **`int destination`:** The destination node this edge points to.
  - **`int cost`:** The cost associated with traversing this edge.
  - **`Edge* next`:** Pointer to the next edge in the linked list of edges originating from the same node.
- **Methods**:
  - **`Edge(int dest, int c)`:** Constructor that initializes an edge with a destination and a cost.
  - **`int getDestination()`:** Returns the destination node of the edge.
  - **`int getSource()`:** Returns the destination node 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.
  - **`void setDestination(int dest)`:** Sets the destination of the edge.
  - **`void setSource(int s)`:** Sets the source of the edge
  - **`void setCost(int cost)`:** Sets the cost of the edge.

2. Class: Node

- **Purpose**: Represents a node in the graph, containing a linked list of edges connected to it.
- **Attributes**:
  - **`int data`:** The value stored in the node.
  - **`Node* next`:** Pointer to the next node in the graph.
  - **`Edge* edges`:** Pointer to the head of the linked list of edges originating from this node.
  - **`int edgeCount`:** The number of edges connected to this node.
- **Methods**:
  - **`Node(int val)`:** Constructor that initializes a node with a specified value and sets pointers to `nullptr`, with `edgeCount` set to zero.
  - **`int getData()`:** Returns the value of the node.
  - **`Node* getNext()`:** Returns the pointer to the next node.
  - **`Edge* getEdges()`**: Returns the pointer to the list of edges connected to this node.
  - **`int getEdgeCount()`:** Returns the number of edges connected to this node.
  - **`void setNext(Node* nextNode)`:** Sets the pointer to the next node in the graph.
  - **`void incrementEdgeCount()`:** Increments the edge count by one.
  - **`void decrementEdgeCount()`:** Decrements the edge count by one.

3. Class: UnionFind

- **Purpose**: Represents the Union-Find data structure used for cycle detection and managing connected components during Kruskal's algorithm.
- **Attributes**:
  - **`int* parent`:** Array of parent nodes for each node.
  - **`int* rank`:** Array of ranks for union by rank optimization.
- **Methods**:
  - **`UnionFind(int n)`:** Constructor that initializes the parent and rank arrays.
  - **`int find(int x)`:** Finds the representative (parent) of the node `x`.

- **`void unionSets(int x, int y):`** Unites the sets containing nodes `x` and `y` using union by rank.

4. Class: Graph

- **Purpose**: Represents the graph structure and manages the nodes and edges. This class implements Kruskal's algorithm to find the MST.
- **Attributes**:
  - **`Node* nodes:`** Pointer to the head of the linked list of nodes.
  - **`int nodeCount:`** The total number of nodes in the graph.
  - **`Node* mstEdges`**: Pointer to the start node of MST and in the edges list it will store the complete MST.
- **Methods**:
  - **`Graph():`** Constructor that initializes an empty graph.
  - **`void addNode(int nodeValue):`** Adds a new node with the specified value to the graph.
  - **`void addEdge(int src, int dest, int cost):`** Adds a directed edge from `src` to `dest` with the specified cost.
  - **`void deleteNode(int nodeValue):`** Deletes a node and removes all edges connected to it.
  - **`void deleteEdge(int src, int dest):`** Removes the edge between the specified nodes.
  - **`void updateNode(int oldValue, int newValue):`** Updates the value of a node in the graph and updates edges accordingly.
  - **`void kruskalMST():`** Implements Kruskal's algorithm to find the Minimum Spanning Tree (MST) and display it.
  - **`Edge* sortEdges():`** Gathers all edges from the graph and sorts them by cost.
  - **`Edge* getMSTEdges():`** Returns the list of edges that form the Minimum Spanning Tree (MST).
  - **`int getMSTWeight():`** Returns the weight of the Minimum Spanning Tree (MST).

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 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):					