TRINITY COLLEGE
DEPARTMENT OF COMPUTER SCIENCE

CPSC 215: Data Structures and Algorithms
Instructor: Dr. Chandranil Chakraborttii
Spring 2025

Laboratory 11

Academic Honesty Policy

In working on programming assignments, you may discuss broad issues of interpretation and understanding and general approaches to a solution. However, conversion to a specific solution or to program code must be your own work. Programming assignments are expected to be the work of the individual student, designed, and coded by him or her alone. Violations are easy to identify and will be dealt with promptly according to the Academic Integrity and Intellectual Dishonesty outlined in the Student Handbook.

- Copying another person's programs or encouraging or assisting another person to commit plagiarism is cheating. In particular, the following activities are strictly prohibited:
- Giving and receiving help in the actual development of code or writing of an assignment.
- Looking at another person's code or showing your code to another person.
- Sharing a copy of all or part of your code regardless of whether that copy is on paper or in a computer file.
- Turning in the work of any other person(s) (former students, friends, textbook authors, people on the Internet, etc.) and representing it as your own work.
- Fabricating compilation or execution results.
- Use of AI ChatBots for any help regarding the assignment is **strictly prohibited.**

The penalty for cheating is a failing grade (F) for the course, and the student is asked to appear at an Academic Dishonesty Hearing. In addition, the College also places a record of the incident in the student's permanent record. It is your responsibility to protect your work from unauthorized access. Do not discard copies of your programs in public places. Do not leave computers unattended and copies of output lying around.

Objectives

- Understand how graphs are represented as data structures using an Adjacency Matrix.
- Implement graph traversal algorithms.

Part I: Creating a Graph Data structure.

In this lab, we will program a Graph data structure that implements graph traversal algorithms discussed in class.

Input Files (Starter Code)

- Graph.java: Graph interface (No Need to change)
- Graphm.java: Adjacency matrix-based implementation of graph (No Need to change)
- Queue.java: Queue interface (No Need to change)
- AQueue.java: Array based Queue Implementation (No Need to change)
- Stack.java: Stack interface (No Need to change)
- LinkedStack.java: Linked List based Stack Implementation (No Need to change)
- SinglyLinkedList.java: Single Linked List Implementation (No Need to change)
- GraphTest.java: Implement traversals. Contains main method **[Need to change]**
- testfile.in: File containing graph details (No Need to change)

Tasks:

- Review the implemented classes (Graph, Graphm)
- Write Java code in **GraphTest**.java to read a data file **(testfile.in)** and create a graph using the input data.
- Using various methods defined in **Graphm**.**java**, create a graph which includes all vertices and edges defined in the data file.
- The input file should have the following format:

```
# Graph example from book for DFS/BFS, MST: Figure 7.10
6       # Number of vertices
U       # Undirected graph
0 4 9   # List of Edges
0 2 7
2 3 1
2 1 5
2 5 2
1 5 6
3 5 2
4 5 1
```

Here, the number on the first line represents the number of vertices in the graph. The letter on the second line indicates whether this is a directed graph ('D') or an undirected graph ('U'). The remainder of the input file must contain one or more lines of the form (*i*, *j*, *w*), which represents a weighted edge between vertex *i* and *j* with the weight *w*.

Use **FileInputSteamReader** class to read in the data file as follows:

```
String file_name = "testfile.in"
BufferedReader br = new BufferedReader (new InputStreamReader(new FileInputStream(file_name)));
```

Expected Output (Part I):

```
Reading the graph from file :
Initializing the graph with 6 vertices
Graph is undirected
Reading new line from file,  Vertex 1 :0 Vertex 2 : Weight  :9
Reading new line from file,  Vertex 1 :0 Vertex 2 : Weight  :7
Reading new line from file,  Vertex 1 :2 Vertex 2 : Weight  :1
Reading new line from file,  Vertex 1 :2 Vertex 2 : Weight  :5
Reading new line from file,  Vertex 1 :2 Vertex 2 : Weight  :2
Reading new line from file,  Vertex 1 :1 Vertex 2 : Weight  :6
Reading new line from file,  Vertex 1 :3 Vertex 2 : Weight  :2
Reading new line from file,  Vertex 1 :4 Vertex 2 : Weight  :1
Graph read from file and initiliazed
```

Part II: Graph Traversals

For this part, you will also need the remaining Input files - Queue and Stack Implementations:

- Write a method **DFS** for this algorithm in **GraphTest.java**. Test your method by calling it from the main method. Your program should print vertex numbers in the order they are visited. Please check sample output at the end of the document.

- Add a method **BFS** to **GraphTest.java** to traverse the graph in BFS order. Each vertex number must be printed when BFS(G,0) is called. Please check sample output at the end of the document.

Depth First Search (DFS)

The DFS strategy of graph traversal proceeds from a given vertex *v* along a path from *v* as deeply into the graph as possible before backing up. As we discussed in class, the DFS algorithm has the following recursive form:

```
DFS(G, v) {
  Mark v as visited.
  for each unvisited vertex u adjacent to v
    DFS(u)
}
```

Breadth First Search (BFS)

After visiting a given vertex v, the BFS strategy of graph traversal visits every vertex adjacent to *v* before visiting any other vertices. The BFS algorithm has a simple iterative form using a queue:

```
BFS(G, v) {
  Mark v as visited
  Add v to queue
  while queue is not empty {
    Remove w from queue
    for each unvisited vertex u adjacent to w {
      Mark u as visited
      add u to queue
    }
  }
}
```

Expected Output (Part II):

BFS: 0 2 4 1 3 5

DFS: 0 2 1 5 3 4

Submission Guidelines:

- Make sure you completed all sections with “To do” statements and added JavaDoc.
- Upon the completion of your lab, upload your submission to the Moodle course website as a single Zip file.
- See your TA and get your Assignment graded before leaving the lab. Otherwise, inform the instructor.

——