ENSF 694 – Summer 2025
Lab 5

Department of Electrical & Software Engineering
University of Calgary
Written by: M. Moussavi, PhD, PEng.

Objective:

This lab focuses on two important data structure concepts:

1. AVL tree, which is a self-balancing binary search tree. It maintains the tree balanced to ensure the height of the tree remains logarithmic with respect to the number of nodes.
2. Graph data structures

Marking Scheme:

Exercise A 20 marks
Exercise B 15 marks

Total marks: 35

Due Date: Friday August 8th before 2:00 PM.

Exercise A – AVL tree

In this exercise you are going to complete an incomplete implementation of a C++ program that builds an AVL tree, allows to insert or search for a record (a key/value pair).

What to Do:

Download files `AVL_tree.h, AVL_tree.cpp,` and `AVL_tree_tester.cpp`. If you read these files carefully, you will see that implementation of some of the member functions are missing. Your task, in this exercise to complete the missing parts in the file `AVL_tree.cpp.`

Please notice that some the function implementations are supposed to be recursive. Also, in some cases such as function insert and find there are overloaded functions (two functions with the same name). One function is declared as public interface and the other function is declared as a helper function (a private function).

Please don't make any changes to the files: `AVL_tree.h,` and `AVL_tree_tester.cpp`
Your also recommended to start with completing the implementation of function insert, followed by its associated files righRotate and leftRotate, then other function.

What to Submit:

1. Submit your source code, `AVL_tree.cpp`, and a screenshot of your program's output, as part of your lab report in PDF format.
2. Your source code (.cpp and .h files) in zipped file.

Exercise B – Graph Data Structure

The objective of this exercise is to become familiar with the concept of graph structures. Graphs are important data structures in many engineering, science and business applications.

What to Do:

1. Copy files `graph.cpp, graph.h, test_graph.cpp, graph.txt,` from D2L. Then, compile, run, and observe the program output.

2. Study the header file called `graph.h` and try to understand the details of `class Graph`, two structures `Edge`, and `Vertex`, and the class called `PriorityQueue`. The priority will be used for the purpose finding shortest path in the function `Dijkstra`. It is also can be used in `unweighted` function that finds the shortest path for an unweighted graph (it sets the distance to 0 before calculates the shortest path.

3. When you run the give code, because of a few missing functions your program should generate the following output (red text is the user input):

```
Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 1

Enter the start vertex: A
A -> A     0   A
A -> B     inf   No path
A -> E     inf   No path
A -> C     inf   No path
A -> D     inf   No path
A -> M     inf   No path

Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 2

Enter the start vertex: A
A -> A     0   A
A -> B     inf   No path
A -> E     inf   No path
A -> C     inf   No path
A -> D     inf   No path
A -> M     inf   No path

Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2):3
// End of program
```

4. Noe, draw an activation record (AR) diagram for the program at point one in `addEdge` (you don't need to show the file I/O object in your diagrams). Also, **You don't need to submit this diagram, but drawing this AR diagram can help you to understand how the program builds graph objects.**

5. Complete the definition of member function `Dijkstra` that uses Dijkstra's algorithm to calculate the shortest distance from a designated vertex v to any vertex w connected to v. And, function unweighted that calculates unweighted-shortest path.

The program must receive the file name from command line:

```
graph graph.txt
```

Where `graph` is the name of the executable, and `graph.txt` is the name of the input file. The input file contains the graph data in the following format:

```
A B 10
A E 5
B C 1
B E 2
C D 4
```

```
D A 7
D C 6
E B 3
E C 9
E D 2
E M 100
```

The first column in this file is the name of a source vertex, the second column is the name of the destination vertex, and the third column is the cost/weight for an edge, from source vertex to the destination vertex.

To better understand how the program interacts with the user to display the possible distances between any vertex to any other vertex, the following lines show a sample run, with sample user inputs (in red). The first column is the start and end vertex, second column shows the distance and the third column show the path from start to end vertex.

```
Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 1

Enter the start vertex: A
A -> A      0    A
A -> B      1    A B
A -> E      1    A E
A -> C      2    A E C
A -> D      2    A E D
A -> M      2    A E M

Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 2

Enter the start vertex: A
A -> A      0    A
A -> B      8    A E B
A -> E      5    A E
A -> C      9    A E B C
A -> D      7    A E D
A -> M      105    A E M

Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 2

Enter the start vertex: C
C -> A      11    C D A
C -> B      19    C D A E B
C -> E      16    C D A E
C -> C      0     C
C -> D      4     C D
C -> M      116   C D A E M

Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 1

Enter the start vertex: C
C -> A      2    C D A
C -> B      3    C D A B
C -> E      3    C D A E
C -> C      0    C
C -> D      1    C D
C -> M      4    C D A E M

Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 1

Enter the start vertex: M
M -> A      inf    No path
M -> B      inf    No path
M -> E      inf    No path
```

```
M -> C      inf    No path
M -> D      inf    No path
M -> M      0    M

Choose the type of graph:
1. Unweighted Graph
2. Weighted Graph
3. Quit
Enter your choice (1 or 2): 3

// Program ended
```

What to hand in:

1. The copy of your source code, and the screenshot of your program output, as part of your lab report in PDF format. We may give you a different input file with more data. Which in this case you need to submit your program output, using this data set.
2. Your source code (.cpp and .h files), in a zipped file.