CSC/DSCI 2720 Data Structure
Lab 13

Due: 04/07/2023 @ 11:59PM

Requirements:

(Failure to follow the requirements will result in a score of Zero.)

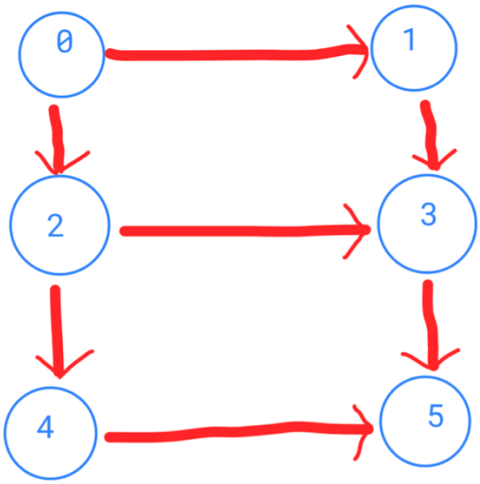
1. You may use whatever IDEs / editors you like, but you must submit your responses on iCollege as **.py files**.
2. Your submissions will work exactly as required.
3. Make sure you submit a file that compiles.
4. Your submission will show an output. Should you recive a Zero for no output shown do not bother to email with "but the logic is perfect".
5. Your program's output must exactly match the specs (design, style) given here for each problem to pass the test cases.
6. Design refers to how well your code is written (i.e., is it clear, efficient, and elegant), while Style refers to the readability of your code (correct indentation, good variable names). Add comments to have necessary explanations for your program.
7. Add a "heading" at the very beginning of your .java files as follow:
   Your Name
   CSc 2720 Lab #N
   Lab time: put your lab time here
   Due time: put the due date here
8. *** If you used any website/online resources as reference, please cite in your comments what website did you use for studying the lab content. Also, you are supposed to make changes to the resource you use to make it your own version. Otherwise, your submission will be considered as plagiarism. ***

PROBLEM STATEMENT:
In today's Lab we will explore a specific way to perform a Depth First Search (DFS) of a given Graph.

You will implement the traversal using one of the two ways stated below:

[1] With Recursion.

[2] Iteratively by using an explicit Stack.



Graph for Traversal

```
/* Class representing a directed graph using adjacency lists */
class Graph:
    def __init__(self, V):
        self.V = V    #number of vertices
        self.adj = {}  #adjacent lists

        # Initialize adjacency lists for all vertices
        for i in range(V):
            self.adj[i] = []

    def add_edge(self, v, w):
        # Ensure v is in the dictionary
        if v not in self.adj:
            self.adj[v] = []

        # Add w to the adjacency list of v
        self.adj[v].append(w)
```

The edges of the Graph *g* is given to you.

```
g = Graph(6)
g.add_edge (0, 1);
g.add_edge (0, 2);
g.add_edge (2, 3);
g.add_edge (2, 4);
g.add_edge (4, 5);
g.add_edge (1, 3);
g.add_edge (3, 5);
```

Your code will need to return the traversal of the nodes in DFS order, where the traversal starts from Node/Vertex 0.

When you follow the traversal process as specified - the complexity of the solution will be linear as shown below.

Time Complexity: O(V + E), where V is the number of Vertices and E is the number of Edges respectively.
Space Complexity: O(V)

The linear space complexity would come from the recursion (AKA "recursion stack") you employ to traverse the Graph. If you solve the problem without recursion (using an explicit Stack), then the mentioned space complexity is obvious.

Submissions that don't meet the linear Time and Space complexities will only receive 50% credit.

Very Very Important:

1. Your code should be well commented which explains all the steps you are performing to solve the problem. A submission without code comments will immediately be deducted 15 points!
2. As a comment in your code, please write your test-cases on how you would test your solution assumptions and hence your code.
   A submission without test cases (as comments) will immediately be de- ducted 15 points!
   Please Remember: Although, written as comments - You will address your test cases in the form of code and not prose :)