CSE-2212: Design and Analysis of Algorithms-I Lab
Practice Lab 2– October 7, 2024

Experiment: Extending the Graph Class with Depth-First Search (DFS) and Topological Sort in Java.

You will continue working on the Graph class you created in the previous lab. The new features you need to implement include:

1. Depth-First Search (DFS): Perform DFS traversal on the graph and print the order of vertices visited.
2. Topological Sort: Implement a topological sorting algorithm using DFS, and return the vertices in topologically sorted order. [Check Lecture 4 in classroom]

As in the previous lab, the graph will be initialized using input from a file. For example, given the following file: **[Notice that the vertices start from 0 today]**

```
6 6
5 2
5 0
4 0
4 1
2 3
3 1
```

This input represents a directed graph with 6 vertices and 6 edges.

New Methods to Implement:
- **DFS(v):** Perform DFS starting from vertex `v` and print the order of traversal.
- **topologicalSort():** Perform Topological Sorting and return the list of vertices in topologically sorted order.

DFS(v):
- Implement DFS traversal starting from a given vertex.
- Print the order of vertices visited during the DFS traversal.

topologicalSort():

- Implement Topological Sort using DFS for Directed Acyclic Graphs (DAG) only.
- The method should return a list of vertices in topologically sorted order.

This should be your main class:

```
public class Lab2 {
    public static void main(String[] args) throws IOException {

        Graph graph = new Graph("input.txt");

        System.out.println("Graph adjacency list:");
        graph.displayGraph();

        System.out.println("\nPerforming DFS starting from vertex 5:");
        graph.DFS(5);

        System.out.println("\nPerforming Topological Sort:");
        List<Integer> topoOrder = graph.topologicalSort();
        System.out.println("Topological Sort order: " + topoOrder);
    }
}
```

Sample input.txt: This represents a directed acyclic graph with 6 vertices and 6 edges:

```
6 6
5 2
5 0
4 0
4 1
2 3
3 1
```

Expected Output:

```
Graph adjacency list:
0 ->
1 ->
2 -> 3
3 -> 1
```

4 -> 0 1
5 -> 2 0

Performing DFS starting from vertex 5:
DFS Traversal starting from vertex 5: 5 2 3 1 0

Performing Topological Sort:
Topological Sort order: [5, 4, 2, 3, 1, 0]