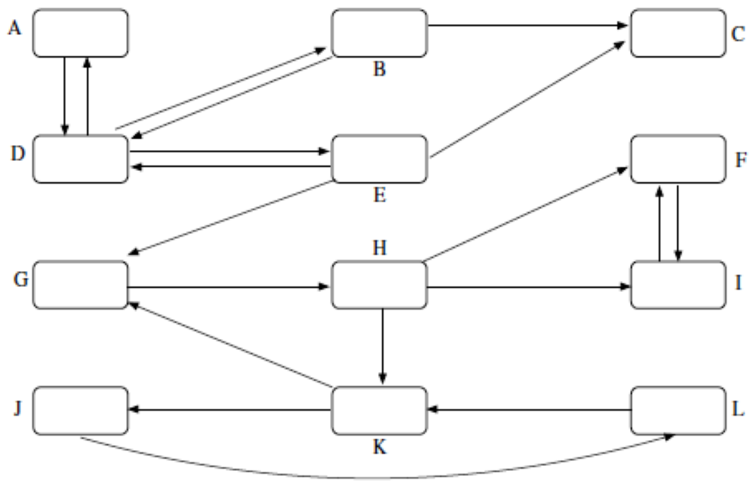
CS590 homework 5 – Graphs exploration (BFS, DFS)

The due date for this assignment is Wednesday, May 19th, at 11.59pm. This assignment is worth 10% of your final grade.

Any sign of collaboration will result in a 0 and being reported to the Graduate Academic Integrity Board. Late submission policy described in the syllabus will be applied.

(100 points)

1. You are give the following graph $G = (V, E)$:



   (a) Perform a DFS (depth-first search) on the given graph. You will visit the nodes in increasing order starting with **A** (increasing from **A** to **L**). Give the discovery $u.d$ and finishing times $u.f$ for every vertex u in the graph. Classify all edges (tree, back, forward, and cross edges).

   (b) Perform a DFS (depth-first search) on the given graph. You will visit the nodes in decreasing order starting with **L** (decreasing from **L** to **A**). Give the discovery $u.d$ and finishing times $u.f$ for every vertex u in the graph. Classify all edges (tree, back, forward, and cross edges).

   (c) Determine the strongly connected components (SCC) of the given graph $G = (V, E)$. Use your output of (a) as a starting point for the remainder of the SCC algorithm. Give the discovery and finishing times for the second run of DFS. Give the sets of vertices that form an SCC and draw the corresponding SCC graph.

   (d) Determine the strongly connected components (SCC) of the given graph $G = (V, E)$. Use your output of (b) as a starting point for the remainder of the SCC algorithm. Give the discovery and finishing times for the second run of DFS. Give the sets of vertices that form an SCC and draw the corresponding SCC graph.

   (40 points)

2. Provide an implementation in C++ of the DFS, and BFS algorithms. You are given a class declaration in the header file *graph.h*, which you should implement in *graph.cpp*. You should strictly implement the method signatures provided in the header file. As you will see in the skeleton code provided, an Adjacency Matrix representation is used for the graphs. We assume that nodes are represented with unique IDs that are integer values starting from 0 to match the indexes in the Adjacency Matrix. You are also given a test case for each algorithm in *main.cpp.* You can implement additional test cases for testing purposes, but you don't have to submit a modified main.cpp. Your submission should run successfully using the main method provided.

Regarding the output of your code, you don't need to store the distance attribute for BFS, or the discovery and finishing times for DFS. For both algorithms the only required output is the following: whenever you visit a node of the graph, print the node ID using a *cout* instruction to show the sequence that the nodes are visited.

In order to modify your implementation compared to the pseudocode provided in the lecture notes, you are asked to use a C++ set to keep track of the visited nodes instead of coloring the nodes. This is a good opportunity to familiarize yourselves with C++ sets if you haven't used it before.

(60 points)

Remarks:

- You are not allowed to use code from online resources. Your submission will be tested against that, and will receive a 0, and a report to the Graduate Academic Integrity Board if it is detected.
- You can use containers from the C++ Std library.
- A Makefile is provided to build the code in the Virtual Box.
- Your code has to compile, and will be graded on the Virtual Box.
- The programming, and testing will take some time. Start early.
- Your report has to be typed, and submitted in a pdf file.
- Feel free to use the provided source code for your implementation. You have to document your code.