COP 3530 Data Structure and Algorithm Analysis
Homework 6

Feng-Hao Liu

In this assignment, you are given several classes in the cpp file "Graph.cpp". Your task is to complete the implementation of the classes specified as below. You **must** finish the basic part, and the advanced part **can be** used as an alternative for the final.

1 Your Tasks: Basic Part

The Setting. In this assignment, we are going to build a class "Graph". We are using the adjacency matrix method to represent a graph. That is, suppose there are n vertices which we call them $v_0, v_1, \ldots, v_{n-1}$. We use a 2-D matrix pointed by adjmatrix to store the edges. In particular, adjmatrix[i][j] stores the weight of the edge $v_i \to v_j$. In this assignment, we consider directed graphs, and if adjmatrix[i][j] is 0, this means the edge $v_i \to v_j$ does not exist. For simplicity, we assume that all the edges, if existing, have positive weights.

Task 1. You are going to build two constructors, the default constructor and another one that takes inputs a graph with n nodes and an adjacency matrix pointed by m. For the default constructor, you can initialize the graph in any way you like. For the second constructor, you need to copy the input graph to the current object. Make sure you new a correct 2-D matrix for the current object, and then copy the input graph to the current object's adjacency matrix.

Task 2. Write the following three functions. (1) The adj function takes input two indices i and j. Return true if nodes i and j have a direct edge. (2) The outdegree function takes input a node index, and returns an integer that represents how many out-edges are connected to the input node. (3) The indegree function is similar to the outdegree function.

Task 3. Write some test cases for your code.

2 Your Tasks: Advance Part

Task 4. You are going to implement the two traversal methods we discussed in the class, namely the BFS and DFS. For each method, you need to print (i.e., cout) the vertices when the algorithm traverses a node. It would be helpful if you write some helper functions/data structures, but you should make your choices.

Task 5. Write a function that determines whether the graph is strongly connected or not. A graph is strongly connected if and only if every node can reach any other nodes. You can use the ideas in Task 4 to help you.

Task 6. Write a function that determines whether the graph contains a cycle. You should notice that the graphs we consider here are **directed** graphs. The idea of DFS can be helpful in this task.

Task 7. Write some test cases for your code.